

Symbolic computation
and
Mathematics
with
the calculator HP Prime

Renée De Graeve
Lecturer at Grenoble I

Translated from French by Jean-Michel Lécointre

© 2013 Renée De Graeve, renee.degraeve@wanadoo.fr

Copy, translation and redistribution of this document on electronic support or paper are permitted for non-commercial purpose only. Use of this document for commercial purpose is forbidden without the written consent of the owner of the copyright. This documentation is provided "as is", without warranty of any kind. The owner of the copyright shall not be held responsible in any case for any damage resulting from the use of this document.

This document is available at the following Internet address:

http://www-fourier.ujf-grenoble.fr/~parisse/hp-prime_cas.pdf

INDEX

- , 41, 131, 293, 324
- inf, 196
- infinity, 196
- !, 227
- !=, 196, 507
- ", 295
- ▶, 493
- %, 130, 208
- %e, 196
- %i, 196
- %pi, 196
- &&, 197
- &*, 324
- &^, 324
- *, 41, 132, 324
- .-, 293, 324
- .*, 293, 325
- ./, 294, 325
- .^, 325
- .+, 292, 324
- /, 41, 134, 136
- //, 491, 506
- /laplace, 95
- :=, 60, 493
- :=, 45
- [[]], 295
- [], 295
- ^, 134, 324
- ^, 41
- _, 367, 369, 375
- ||, 197
- " , 61
- ' , 61, 225
- +, 41, 131, 292, 296, 303, 324
- + infinity, 196
- <, 196, 507
- <=, 196, 197, 507
- ≠, 196, 507
- ==, 196, 507
- =>, 369, 372, 493
- >, 196, 507
- >=, 196, 507
- ∅, 61
- ≤, 196
- ≥, 196
- , 45, 60
- a2q**, 104
- abcuv**, 149
- about**, 500
- abs**, 210
- ABS**, 210, 335
- abscissa**, 430
- acos, 216
- ACOS, 216
- acos2asin**, 114
- acos2atan**, 115
- acosh**, 221
- ACOSH**, 221
- acot**, 219
- ACOT**, 219
- acsc**, 217
- ACSC**, 217
- ADDCOL**, 316
- additionally**, 499
- ADDROW**, 322
- affix**, 431
- algvar**, 223
- ALOG**, 213
- alog10**, 213
- altitude**, 393
- AMORT**, 460
- and, 197, 507
- AND**, 507
- angle**, 431
- angleat**, 423
- angleatraw**, 423
- Ans**, 501
- ans(n)**, 501
- append**, 276
- apply**, 287
- approx**, 225
- arc**, 411
- ARC**, 411
- arclen**, 432

area, 434
areaat, 424
areaatraw, 424
arg, 209
ARG, 209
asc, 297
ASC, 297
asec, 218
ASEC, 218
asin, 216
ASIN, 216
asin2acos, 114
asin2atan, 114
asinh, 220
ASINH, 220
assume, 496
atan, 217
ATAN, 217
atan2acos, 116
atan2asin, 116
atanh, 221
ATANH, 221
atrig2ln, 117

barycenter, 380
basis, 328
Beta, 358
binomial, 239
BINOMIAL, 239
binomial_cdf, 242
BINOMIAL_CDF, 242
binomial_icdf, 246
BINOMIAL_ICDF, 246
bisector, 393
bitand, 198
bitor, 198
bitxor, 198
bounded_function, 67
BREAK, 505

canonical_form, 152
CASE, 503
cat, 296
ceiling, 199
CEILING, 199
Celsius2Fahrenheit, 374
center, 392
cfactor, 54
cFactor, 54
char, 298
CHAR, 298
charpoly, 327
CHECK, 510
chinrem, 150, 177
chisquare, 238
CHISQUARE, 238
chisquare_cdf, 241
CHISQUARE_CDF, 241
chisquare_icdf, 245
CHISQUARE_ICDF, 245
cholesky, 345
CHOOSE, 508
Ci, 365
circle, 409
circumcircle, 412
coeff, 156
col, 318
colDim, 316
collect, 50
colnorm, 337
COLNORM, 337
colSwap, 317
comb, 227
COMB, 227
comDenom, 56
comment, 491
common_perpendicular, 377
companion, 165
compare, 495
complexroot, 153
concat, 274
CONCAT, 274
cond, 339
COND, 339
conic, 105, 412
conj, 209
CONJ, 209
contains, 282
content, 170
CONTINUE, 504
convert, 369, 372
convexhull, 409
coordinates, 434
CopyVar, 493
correlation, 256
cos, 216
COS, 216
cos2sintan, 115
cosh, 220
COSH, 220
cot, 218
COT, 218
count, 282
covariance, 254
covariance_correlation, 257
cpartfrac, 56
cross, 355
CROSS, 355
csc, 217
CSC, 217
csolve, 90
cSolve, 90
cumSum, 291, 296
curl, 73

cyclotomic, 178
cZeros, 89, 91

degree, 169
DELCOL, 318
delcols, 318
DELROW, 319
delrows, 319
deltalist, 285
desolve, 91
deSolve, 91
det, 137, 326
DET, 326
diag, 344
diff, 61
dim, 278, 302, 315
DIM, 278, 302
Dirac, 366
distance, 437
distance2, 437
distanceat, 425
distanceatraw, 426
divergence, 73
divis, 141, 157
division_point, 389
divpc, 84
DOM_COMPLEX, 494
DOM_FLOAT, 494
DOM_IDENT, 494
DOM_INT, 494
DOM_LIST, 494
DOM_RAT, 494
DOM_STRING, 494
DOM_SYMBOLIC, 494
dot, 356
DOT, 356
DrawSlp, 392

e, 196
EDITMAT, 509
egcd, 148
Ei, 363
EIGENVAL, 342
eigenvals, 342
eigenvects, 342
EIGENVV, 342
eigVI, 343
element, 387
ellipse, 412
equation, 438
equilateral_triangle, 401
erf, 361
erfc, 362
euler, 126
euler_gamma, 196
eval, 224
evalc, 210

evalf, 225
even, 122
exact, 226
exbisector, 394
excircle, 413
exp, 214
EXP, 214
exp2pow, 111
exp2trig, 112
expand, 52
expexpand, 113
EXPM1, 214
exponential_regression, 261
expr, 299
extract_measure, 438
ezgcd, 146

f2nd, 57
factor, 53, 137, 140
factor_xn, 170
factorial, 227
factors, 158
fadeev, 328
Fahrenheit2Celsius, 374
false, 196
FALSE, 196
fcoeff, 163
fft, 87
fisher, 239
FISHER, 239
fisher_cdf, 242
FISHER_CDF, 242
fisher_icdf, 246
FISHER_ICDF, 246
floor, 200
FLOOR, 200
fMax, 60
fMin, 60
FNROOT, 207
FOR, 513
FOR FROM TO DO END, 504
FOR FROM TO STEP DO END, 504
FP, 201
frac, 204
fracmod, 136
FREEZE, 508
froot, 154
fsolve, 102
function_diff, 61

Gamma, 359
gauss, 104
gbasis, 179
gcd, 123, 136, 145, 159
GETKEY, 508
GF, 138
grad, 74

gramschmidt, 104
greduce, 179

half_line, 394
halftan, 116
halftan_hyp2exp, 109
hamdist, 199
harmonic_conjugate, 391
harmonic_division, 390
has, 223
head, 304
Heaviside, 366
hermite, 180
hessenberg, 345
hessian, 74
hexagon, 407
hilbert, 333
histogram, 254
homothety, 417
hyp2exp, 113
hyperbola, 413
HypZ1mean, 461
HypZ2mean, 461

i, 196, 208
iabcuv, 124
ibasis, 328
ibpdv, 78
ibpu, 80
ichinrem, 129
id, 199
IDENMAT, 332
identity, 332
idivis, 122
iegcd, 124
IF, 502, 512
IF THEN ELSE END, 503
ifactor, 123
ifactors, 123
IFERR, 504
ifft, 87
IFTE, 502
igcd, 145
ihermite, 345
ilaplace, 95
im, 209
IM, 209
image, 329
incircle, 414
inf, 196
infinity, 196
input, 491
INPUT, 491, 508
InputStr, 491
inString, 297, 301
INSTRING, 297
int, 64

integrate, 76
inter, 385
interval2center, 59
inv, 136, 138, 199
inversion, 418
invlaplace, 85, 95
invztrans, 100
IP, 200
iPart, 204
iquo, 128
iquorem, 129
irem, 129
is_collinear, 442
is_concyclic, 442
is_conjugate, 442
is_coplanar, 443
is_element, 444
is_equilateral, 444
is_harmonic, 450
is_harmonic_circle_bundle, 450
is_harmonic_line_bundle, 450
is_isosceles, 445
is_orthogonal, 446
is_parallel, 446
is_parallelogram, 447
is_perpendicular, 447
is_rectangle, 448
is_rhombus, 448
is_square, 449
ISKEYDOWN, 508
ismith, 346
isobarycenter, 383
isopolygon, 407
isosceles_triangle, 402
isprime, 125
isPrime, 125
ITERATE, 505
ithprime, 125

jacobi_symbol, 127
jordan, 344
JordanBlock, 333

ker, 329

l1norm, 290, 357
l2norm, 290, 356
lagrange, 180
laguerre, 184
laplace, 85
laplacian, 75
lcm, 124, 147, 160
lcoeff, 163, 170
left, 58, 281, 301
legendre, 184
legendre_symbol, 127
length, 278, 302

lgcd, 124
limit, 66, 68
lin, 108
line, 394
Line, 396
linear_interpolate, 260
linear_regression, 260
LineHorz, 396
LineTan, 392
LineVert, 396
linsolve, 102
list2mat, 289
ln, 211
LN, 211
lname, 222
lncollect, 108
lnexpand, 108
LNP1, 214
locus, 414
log, 211
LOG, 212
log10, 212
logarithmic_regression, 262
logb, 213
logistic_regression, 265
LQ, 347
LSQ, 348
lu, 350
LU, 349
lvar, 222

makelist, 273
MAKELIST, 273
MAKEMAT, 331
MANT, 204
map, 287
mat2list, 290
matpow, 344
matrix, 332
max, 206
MAX, 206
maxnorm, 290, 357
MAXREAL, 196
mean, 248, 267, 270
MEAN, 460
median, 251, 267, 270
median_line, 397
member, 282
mid, 302
midpoint, 382
min, 206
MIN, 206
MINREAL, 196
mkisom, 333
mksa, 370, 374
MOD, 206
modgcd, 146

mRow, 323
mRowAdd, 323
MSGBOX, 508
mult_c_conjugate, 211
mult_conjugate, 52

nDeriv, 63
neg, 199
nextprime, 125
norm, 290
normal, 51, 131, 132, 134
normald, 238
NORMALD, 238
normald_cdf, 240
NORMALD_CDF, 240
normald_icdf, 244
NORMALD_ICDF, 244
normalize, 291
not, 197
NOT, 507
nSolve, 101
NTHROOT, 207
numer, 57

odd, 122
odesolve, 98
ofnom, 57
open_polygon, 408
or, 507
OR, 507
order_size, 84
ordinate, 439
orthocenter, 386

pa2b2, 128
pade, 70
parabola, 416
parallel, 398
parallelogram, 406
parameq, 439
partfrac, 55
pcoef, 157, 163
pcoeff, 157, 163
perimeter, 440
perimeterat, 427
perimeteratraw, 428
perm, 228
PERM, 228
perpen_bisector, 398
perpendicular, 398
pi, 196
Pi, 196
PI, 196
piecewise, 48
PIECEWISE, 48
pivot, 340
plotcontour, 192

plotdensity, 190
plotfield, 191
plotfunc, 187
plotimplicit, 189
plotlist, 193, 259
plotode, 193
plotparam, 187
plotpolar, 188
plotseq, 189
pmin, 164
point, 382
point2d, 383
poisson, 239
POISSON, 239
poisson_cdf, 244
POISSON_CDF, 244
poisson_icdf, 247
POISSON_ICDF, 247
polar, 391
polar_coordinates, 437
polar_point, 384
pole, 391
poly2symb, 162
POLYCOEF, 305
polyEval, 164
POLYEVAL, 305
POLYFORM, 306
polygon, 408
polygonplot, 258
polygonscatterplot, 259
polynomial_regression, 263
POLYROOT, 308
POS, 277
potential, 75
pow2exp, 112
power_regression, 264
powerpc, 417
powexpand, 109
powmod, 130, 134
PredX, 461
PredY, 461
prepend, 276
preval, 65, 81
prevprime, 126
primpart, 171
print, 493
PRINT, 509
product, 286
projection, 419
proot, 152
propfrac, 56
Psi, 360
ptayl, 154
purge, 501

q2a, 104
qr, 351

QR, 351
quadrilateral, 406
quantile, 253, 267, 270
quartile1, 252, 267
quartile3, 252, 267
quartiles, 251, 267, 270
quo, 132, 141, 166
quorem, 133, 144
quote, 225, 295
QUOTE, 225

radical_axis, 400
radius, 440
ramn, 332
rand, 229
randexp, 236
RANDINT, 229
randMat, 332
RANDMAT, 332
randmatrix, 332
randNorm, 235
RANDNORM, 235
RANDOM, 228
randperm, 232
randpoly, 166
randPoly, 166
RandSeed, 236
RANDSEED, 236
randvector, 232
rank, 340
RANK, 340
re, 210
RE, 210
reciprocation, 391
rectangle, 404
rectangular_coordinates, 436
REDIM, 321
reduced_conic, 106
ref, 103
reflection, 419
regroup, 51
REGRS, 460
rem, 133, 142, 167
remove, 280
reorder, 166
REPEAT UNTIL, 505
REPLACE, 321
residue, 70
restart, 501
resultant, 174
REVERSE, 274
revlist, 278
rhombus, 403
right, 58, 281, 301
right_triangle, 402
romberg, 65
rootof, 155

rotate, 279, 302
rotation, 420
round, 201
ROUND, 201
row, 318
rowAdd, 323
rowDim, 316
rownorm, 336
ROWNORM, 336
rowSwap, 317
rref, 138, 329
RREF, 329
rsolve, 312

SCALE, 323
SCALEADD, 323
schur, 352
SCHUR, 352
sec, 218
SEC, 218
segment, 399
select, 285
seq, 506
seqsolve, 310
series, 69
shift, 279
shift_phase, 118
Si, 364
sign, 210
SIGN, 210
signature, 59
similarity, 421
simplify, 50
simult, 330
sin, 216
SIN, 216
sin2costan, 114
sincos, 112
single_inter, 384
sinh, 220
SINH, 220
size, 278, 302
SIZE, 278, 302
slope, 441
slopeat, 428
slopeatraw, 429
snedecor, 239
snedecor_cdf, 242
snedecor_icdf, 246
solve, 88
sort, 274
SORT, 274
SPECNORM, 337
SPECRAD, 338
spline, 182
sq, 199
sqrfree, 54

sqrt, 199
square, 405
srand, 236
STARTVIEW, 511
STAT1, 460
stddev, 249, 267, 270
stdDev, 249
stddevp, 249, 267
Sto ▶, 60
string, 299, 300
student, 238
STUDENT, 238
student_cdf, 241
STUDENT_CDF, 241
student_icdf, 245
STUDENT_ICDF, 245
sturm, 171
sturmab, 172
sturmseq, 172
SUB, 320
subMat, 320
subst, 55
sum, 72, 286
SUM, 460
sum_riemann, 81
suppress, 280
surd, 207
svd, 353
SVD, 353
svl, 354
SVL, 354
SWAPCOL, 317
SWAPROW, 317
sylvester, 174
symb2poly, 161

table, 271
tail, 280, 304
TAN, 217
tan2sincos, 116
tan2sincos2, 115
tangent, 399
tanh, 221
TANH, 221
taylor, 84
tchebyshev1, 185
tchebyshev2, 186
tcollect, 120
texpand, 109
time, 41
tlin, 118
trace, 341
TRACE, 341
translation, 422
transpose, 326
triangle, 401
trig2exp, 120

trigcos, 117
trigexpand, 120
trigsin, 117
trigtan, 118
trn, 326
TRN, 326
true, 196
TRUE, 196
trunc, 203
truncate, 157
TRUNCATE, 203
tsimplify, 113
type, 494
TYPE, 494

ufactor, 370, 375
unapply, 45
UNCHECK, 511
usimplify, 371, 375
UTPC, 236
UTPF, 236
UTPN, 237
UTPT, 237

valuation, 169
vandermonde, 335

variance, 250, 267, 270
vector, 396
vertices, 386
vertices_abca, 387
vpotential, 76

WAIT, 509
when, 47
WHILE, 513
WHILE DO END, 505

xor, 197
XOR, 507
XPON, 205

zeros, 89
Zeta, 361
zip, 289
ztrans, 99

ΔLIST, 285

π , 196
ΠLIST, 286

ΣLIST, 286

Table of content

PART I	GETTING STARTED	32
	Generalities	33
	CAS and HOME keys	34
	Reset and clear	36
	Tactile screen	36
	Keys	36
	General settings	37
	CAS settings: <code>Shift CAS</code>	37
	Calculator settings: <code>Shift HOME</code>	37
	Symbolic computation functions	37
PART II	MENU CAS OF THE TOOLBOX KEY	38
CHAPTER 1	GENERALITIES	40
1.1	Calculations in the CAS	40
1.2	Priority of operators	40
1.3	Implicit multiplication	40
1.4	Duration of a calculation: <code>time</code>	40
1.5	Lists and sequences in the CAS	41
1.6	Difference between expressions and functions	42
1.6.1	Defining a function by an expression	43
1.6.2	Definition of a function of one or several variables	44
1.6.3	To define a function by two expressions: <code>when</code>	46
1.6.4	Defining a function by n values: <code>PIECEWISE</code> <code>piecewise</code>	46
1.6.5	Exercise on expressions	47
1.6.6	Exercise on the functions (to be followed)	47
CHAPTER 2	MENU ALGEBRA	49
2.1	Simplifying an expression: <code>simplify</code>	49
2.2	Factorizing a polynomial on the integers: <code>collect</code>	49
2.3	Regrouping and simplifying: <code>regroup</code>	50
2.4	Expanding and simplifying: <code>normal</code>	50

2.5	Expanding an expression: <code>expand</code>	50
2.6	Multiply by the conjugate quantity: <code>mult_conjugate</code>	51
2.7	Factorizing an expression: <code>factor</code>	51
2.8	Factorization without square factor: <code>sqrfree</code>	52
2.9	Factorization in \mathbb{C}: <code>cFactor</code> <code>cfactor</code>	53
2.10	Substituting a variable by a value: <code>subst</code>	54
2.11	Fractions	54
2.11.1	Decompose into simple elements: <code>partfrac</code>	54
2.11.2	Decomposition in simple elements on \mathbb{C} : <code>cpartfrac</code>	54
2.11.3	Put to common denominator: <code>comDenom</code>	55
2.11.4	Integer part and fractional part: <code>propfrac</code>	55
2.12	Extract	56
2.12.1	Numerator of a fraction after simplification: <code>numer</code>	56
2.12.2	Denominator of a fraction after simplification: <code>ofnom</code>	56
2.12.3	Numerator and denominator: <code>f2nd</code>	56
2.12.4	Get the left member of an equation: <code>left</code>	57
2.12.5	Get the right member of an equation: <code>right</code>	57
2.12.6	Center of an interval: <code>interval2center</code>	57
2.12.7	Signature of a permutation: <code>signature</code>	58
CHAPTER 3	MENU CALCULUS	59
3.1	Definition of a function: <code>:= and</code> \rightarrow (Sto ►)	59
3.2	Maximum and minimum of an expression: <code>fMax</code> <code>fMin</code>	59
3.3	Differentiate	60
3.3.1	Derivative function of a function: <code>function_diff</code>	60
3.3.2	Differentiate: <code>∂ diff ' \'</code>	60
3.3.3	Approximate calculation of the derivative number: <code>nDeriv</code>	62
3.4	Integration	63
3.4.1	Primitive: <code>int</code>	63
3.4.2	Evaluate a primitive: <code>preval</code>	64
3.4.3	Approximate calculation of integrals with the Romberg method: <code>romberg</code>	64
3.5	Limites: <code>limit</code>	64
3.6	Limit and integral	66
3.7	Series: <code>series</code>	68
3.8	Residue of an expression in a point: <code>residue</code>	68
3.9	Pade approximation: <code>pade</code>	69
3.10	Indexed finite and infinite sum and discrete primitive: <code>sum</code>	70
3.11	Differential	72

	13	
3.11.1	Rotational curl: <code>curl</code>	72
3.11.2	Divergence: <code>divergence</code>	72
3.11.3	Gradient: <code>grad</code>	72
3.11.4	Hessian matrix: <code>hessian</code>	73
3.11.5	Laplacian: <code>laplacian</code>	73
3.11.6	Potential: <code>potential</code>	74
3.11.7	Conservative vector field: <code>vpotential</code>	74
3.12	Integral	75
3.12.1	Primitive and definite integral: <code>integrate</code>	75
3.12.2	Integration by parts: <code>ibpdv</code>	77
3.12.3	Integration by parts: <code>ibpu</code>	78
3.12.4	Evaluate a primitive: <code>preval</code>	79
3.13	Limits	80
3.13.1	Riemann sum: <code>sum_riemann</code>	80
3.13.2	Series expansion: <code>taylor</code>	82
3.13.3	Division by increasing power order: <code>divpc</code>	83
3.14	Transform	83
3.14.1	Laplace transform: <code>laplace</code>	83
3.14.2	Laplace transform inverse: <code>invlaplace</code>	84
3.14.3	Fast Fourier transform: <code>fft</code>	85
3.14.4	inverse of the fast Fourier transform: <code>ifft</code>	85
CHAPTER 4	MENU SOLVE	87
4.1	Solve equations: <code>solve</code>	87
4.2	Zeros of an expression: <code>zeros</code>	88
4.3	Complex Zeros of an expression: <code>cZeros</code>	88
4.4	Solve equations in \mathbb{C}: <code>cSolve</code> <code>csolve</code>	89
4.5	Complex zeros of an expression: <code>cZeros</code>	90
4.6	Differential equations	90
4.6.1	Solve differential equations: <code>deSolve</code> <code>desolve</code>	90
4.6.2	Laplace transform and inverse Laplace transform: <code>/laplace</code> <code>ilaplace</code> <code>invlaplace</code>	94
4.7	Approximate solution of $y' = f(t, y)$: <code>odesolve</code>	96
4.8	z transform and z inverse transform	98
4.8.1	z transform of a series: <code>ztrans</code>	98
4.8.2	z transform inverse of a rational fraction: <code>invztrans</code>	99
4.9	Solve numerical equations: <code>nSolve</code>	100
4.10	Solve equations with <code>fsolve</code>	100
4.11	Linear systems	101
4.11.1	Solve a linear system: <code>linsolve</code>	101
4.11.2	Gauss reduction of a matrix: <code>ref</code>	101
4.12	Quadratic forms	102

4.12.1	Matrix of a quadratic form: <code>q2a</code>	102
4.12.2	Transform a matrix in a quadratic form: <code>a2q</code>	103
4.12.3	Gauss method: <code>gauss</code>	103
4.12.4	Gramschmidt process: <code>gramschmidt</code>	103
4.13	Conics	104
4.13.1	Plot of a conic: <code>conic</code>	104
4.13.2	Reduction of a conic: <code>reduced_conic</code>	104
CHAPTER 5	MENU REWRITE	106
5.1	Collect the logarithms: <code>lncollect</code>	106
5.2	Expand the logarithms: <code>lnexpand</code>	106
5.3	Linearize the exponentials: <code>lin</code>	106
5.4	Transform a power in product of powers: <code>powexpand</code>	107
5.5	Transform the trigonometric and hyperbolic expressions in $\tan(x/2)$ and in ex: <code>halftan_hyp2exp</code>	107
5.6	Expand a transcendental and trigonometric expression: <code>texpand</code>	107
5.7	Exp & Ln	109
5.7.1	Transform $\exp(n * \ln(x))$ in power: <code>exp2pow</code>	109
5.7.2	Transform a power into an exponential: <code>pow2exp</code>	110
5.7.3	Transform the complex exponentials into sin and cos: <code>sincos exp2trig</code>	110
5.7.4	Transform the functions hyperbolic in exponentials: <code>hyp2exp</code>	111
5.7.5	Write with complex exponentials: <code>tsimplify</code>	111
5.7.6	Expand the exponentials: <code>expexpand</code>	111
5.8	Sine	111
5.8.1	Transform the arcsin into arccos: <code>asin2acos</code>	111
5.8.2	Transform the arcsin in arctan: <code>asin2atan</code>	112
5.8.3	Transform $\sin(x)$ in $\cos(x)*\tan(x)$: <code>sin2costan</code>	112
5.9	Cosine	112
5.9.1	Transform the arccos into arcsin: <code>acos2asin</code>	112
5.9.2	Transform the arccos into arctan: <code>acos2atan</code>	112
5.9.3	Transform $\cos(x)$ into $\sin(x)/\tan(x)$: <code>cos2sintan</code>	113
5.10	Tangent	113
5.10.1	Transform $\tan(x)$ with $\sin(2x)$ and $\cos(2x)$: <code>tan2sincos2</code>	113
5.10.2	Transform the arctan into arcsin: <code>atan2asin</code>	113
5.10.3	Transform the arctan into arccos: <code>atan2acos</code>	114
5.10.4	Transform $\tan(x)$ into $\sin(x)/\cos(x)$: <code>tan2sincos</code>	114
5.10.5	Transform a trigonometric expression in term of $\tan(x/2)$: <code>halftan</code>	114
5.11	Trigonometry	115
5.11.1	Simplify by privileging sine: <code>trigsin</code>	115
5.11.2	Simplify by privileging cosine: <code>trigcos</code>	115
5.11.3	Transform trigonometric inverse functions to logarithms: <code>atrig2ln</code>	115
5.11.4	Simplify by privileging tangent: <code>trigtan</code>	115
5.11.5	Linearize a trigonometric expression: <code>tlin</code>	116
5.11.6	Shift the phase by $\pi/2$ in trigonometric expressions: <code>shift_phase</code>	116

5.11.7	Collect the sine and cosine of a same angle: <code>tcollect</code>	118
5.11.8	Expand a trigonometric expression: <code>trigexpand</code>	118
5.11.9	Transform a trigonometric expression into complex exponentials: <code>trig2exp</code>	118
CHAPTER 6 MENU INTEGER		119
6.1	Test of parity: <code>even</code>	119
6.2	Test of non parity: <code>odd</code>	119
6.3	Divisors of an integer: <code>idivis</code>	119
6.4	Prime factors decomposition of an integer: <code>ifactor</code>	120
6.5	List of prime factors and their multiplicity: <code>ifactors</code>	120
6.6	GCD of one or several integers: <code>gcd</code>	120
6.6.1	GCD of a list of integers: <code>lgcd</code>	120
6.7	LCM of one or several integers: <code>lcm</code>	121
6.7.1	Bezout identity: <code>iegcd</code>	121
6.7.2	Solve $au + bv = c$ in \mathbb{Z} : <code>iabcuv</code>	121
6.8	Primality	122
6.8.1	Check whether a number is prime: <code>isPrime isprime</code>	122
6.8.2	The N-th prime number: <code>ithprime</code>	122
6.8.3	<code>nextprime</code>	122
6.8.4	<code>prevprime</code>	123
6.8.5	Euler's totient: <code>euler</code>	123
6.8.6	Legendre symbole: <code>legendre_symbol</code>	123
6.8.7	Jacobi symbol: <code>jacobi_symbol</code>	124
6.8.8	Solve $a^2 + ab^2 = p$ in \mathbb{Z} : <code>pa2b2</code>	125
6.9	Division	125
6.9.1	Quotient of the Euclidean division: <code>iquo</code>	125
6.9.2	Remainder of the Euclidean division: <code>irem</code>	125
6.9.3	Quotient and remainder of the Euclidean division: <code>iquorem</code>	126
6.9.4	Chinese remainder for integers: <code>ichinrem</code>	126
6.9.5	Calculation of $an \bmod p$: <code>powmod</code>	127
6.10	Modular calculus in $\mathbb{Z}/p\mathbb{Z}$ or in $\mathbb{Z}/p\mathbb{Z}[x]$	127
6.10.1	Expand and factorise: <code>normal</code>	127
6.10.2	Addition in $\mathbb{Z}/p\mathbb{Z}$ or in $\mathbb{Z}/p\mathbb{Z}[x]$: <code>+</code>	128
6.10.3	Substraction in $\mathbb{Z}/p\mathbb{Z}$ or in $\mathbb{Z}/p\mathbb{Z}[x]$: <code>-</code>	128
6.10.4	Multiplication in $\mathbb{Z}/p\mathbb{Z}$ or $\mathbb{Z}/p\mathbb{Z}[x]$: <code>*</code>	129
6.10.5	Quotient: <code>quo</code>	129
6.10.6	Remainder: <code>rem</code>	129
6.10.7	Quotient and remainder: <code>quorem</code>	130
6.10.8	Division in $\mathbb{Z}/p\mathbb{Z}$ or $\mathbb{Z}/p\mathbb{Z}[x]$: <code>/</code>	130
6.10.9	Power in $\mathbb{Z}/p\mathbb{Z}$ or $\mathbb{Z}/p\mathbb{Z}[x]$: <code>^</code>	131
6.10.10	Calculation of $an \bmod p$ or of $A(x)n \bmod \mathfrak{f}(x), p$: <code>powmod</code>	131
6.10.11	Inverse in $\mathbb{Z}/p\mathbb{Z}$: <code>inv</code> or <code>/</code>	132
6.10.12	Transform an integer into its fraction modulus p : <code>fracmod</code>	133
6.10.13	GCD in $\mathbb{Z}/p\mathbb{Z}[x]$: <code>gcd</code>	133
6.10.14	Factorization in $\mathbb{Z}/p\mathbb{Z}[x]$: <code>factor</code>	134
6.10.15	Determinant of a matrix of $\mathbb{Z}/p\mathbb{Z}$: <code>det</code>	134

6.10.16	Inverse of a matrix of $\mathbb{Z}/p\mathbb{Z}$: <code>inv</code>	134
6.10.17	Solve a linear system of $\mathbb{Z}/p\mathbb{Z}$: <code>rref</code>	134
6.10.18	Creation of a Galois field: <code>GF</code>	135
6.10.19	Factorization of a polynomial with coefficients in a Galois field: <code>factor</code>	137
6.11	Arithmetic of polynomials	137
6.11.1	List of divisors of a polynomial: <code>divis</code>	137
6.11.2	Euclidean quotient of two polynomials: <code>quo</code>	138
6.11.3	Euclidean remainder of two polynomials: <code>rem</code>	139
6.11.4	Quotient and Euclidean remainder: <code>quorem</code>	140
6.11.5	GCD of polynomials by Euclid's algorithm: <code>gcd igcd</code>	141
6.11.6	Choose the algorithm of the GCD of two polynomials: <code>ezgcd modgcd</code>	143
6.11.7	LCM of two polynomials: <code>lcm</code>	144
6.11.8	Bezout identity: <code>egcd</code>	145
6.11.9	Solve polynomial of the form $au + bv = c$: <code>abcuv</code>	146
6.11.10	Chinese remainder: <code>chinrem</code>	146
CHAPTER 7	MENU POLYNOMIAL	148
7.1	Canonical form: <code>canonical_form</code>	148
7.2	Numerical roots of a polynomial: <code>root</code>	148
7.3	Roots exact of a polynomial	149
7.3.1	Exact boundaries of complex roots of a polynomial: <code>complexroot</code>	149
7.3.2	Exact values of complex rational roots of a polynomial: <code>crationalroot</code>	149
7.4	Fraction rational, its roots and its exact poles	150
7.4.1	Roots and exact poles of a rational fraction: <code>root</code>	150
7.5	Writing in powers of $(x - a)$: <code>ptayl</code>	150
7.6	Calculation with the exact roots of a polynomial: <code>rootof</code>	151
7.7	Coefficients of a polynomial: <code>coeff</code>	152
7.8	Coefficients of a polynomial defined by its roots: <code>pcoeff pcoef</code>	153
7.9	Truncation of order n: <code>truncate</code>	153
7.10	List of divisors of a polynomial: <code>divis</code>	153
7.11	List of factors of a polynomial: <code>factors</code>	154
7.12	GCD of polynomials by Euclid's algorithm: <code>gcd</code>	154
7.13	LCM of two polynomials: <code>lcm</code>	156
7.14	Create	157
7.14.1	Transform a polynomial into a list (internal recursive dense format): <code>symb2poly</code>	157
7.14.2	Transform the internal sparse distributed format of the polynomial into a polynomial writing: <code>poly2symb</code>	158
7.14.3	Coefficients of a polynomial defined by its roots: <code>pcoeff pcoef</code>	158
7.14.4	Coefficients of a rational fraction defined by its roots and its poles: <code>fcoeff</code>	159
7.14.5	Coefficients of the term of highest degree of a polynomial: <code>lcoeff</code>	159
7.14.6	Evaluation of a polynomial: <code>polyEval</code>	159

7.14.7	Minimal polynomial: <code>pmin</code>	160
7.14.8	Companion matrix of a polynomial: <code>companion</code>	161
7.14.9	Random polynomials: <code>randpoly</code> <code>randPoly</code>	161
7.14.10	Change the order of variables: <code>reorder</code>	162
7.15	Algebra	162
7.15.1	Euclidean quotient of two polynomials: <code>quo</code>	162
7.15.2	Euclidean remainder of two polynomials: <code>rem</code>	163
7.15.3	Degree of a polynomial: <code>degree</code>	164
7.15.4	Valuation of a polynomial: <code>valuation</code>	165
7.15.5	Coefficient of the term of highest degree of a polynomial: <code>lcoeff</code>	165
7.15.6	Put in factor of x^n in a polynomial: <code>factor_xn</code>	166
7.15.7	GCD of coefficients of a polynomial: <code>content</code>	166
7.15.8	Primitive part of a polynomial: <code>primpart</code>	166
7.15.9	Sturm sequence and number of changes of the sign of P on $[a; b]$: <code>sturm</code>	166
7.15.10	Number of changes of sign on $[a; b]$: <code>sturmab</code>	167
7.15.11	Sequence of Sturm: <code>sturmseq</code>	168
7.15.12	Sylvester matrix of two polynomials: <code>sylvester</code>	169
7.15.13	Resultant of two polynomials: <code>resultant</code>	169
7.15.14	Chinese remainder: <code>chinrem</code>	172
7.16	Special	173
7.16.1	Cyclotomic polynomial: <code>cyclotomic</code>	173
7.16.2	Groebner basis: <code>gbasis</code>	174
7.16.3	Reduction according to a Groebner basis: <code>greduce</code>	175
7.16.4	Hermite polynomial: <code>hermite</code>	175
7.16.5	Lagrange interpolation: <code>lagrange</code>	176
7.16.6	Natural splines: <code>spline</code>	177
7.16.7	Laguerre polynomial: <code>laguerre</code>	179
7.16.8	Legendre polynomial: <code>legendre</code>	180
7.16.9	Tchebyshev polynomial of first kind: <code>tchebyshev1</code>	180
7.16.10	Tchebyshev polynomial of second kind: <code>tchebyshev2</code>	181
CHAPTER 8	MENU PLOT	182
8.1	Plot of a function: <code>plotfunc</code>	182
8.2	Parametric curve: <code>plotparam</code>	182
8.3	Polar curve: <code>plotpolar</code>	183
8.4	Plot of a recurrent sequence: <code>plotseq</code>	184
8.5	Implicit plot in 2D: <code>plotimplicit</code>	184
8.6	Plot of a function by colors levels: <code>plotdensity</code>	185
8.7	The field of tangents: <code>plotfield</code>	185
8.8	Level curves: <code>plotcontour</code>	187
8.9	Plot of solutions of a differential equation: <code>plotode</code>	187
8.10	Polygonal line: <code>plotlist</code>	188
PART III	THE MENU MATH OF THE TOOLBOX KEY	190

CHAPTER 9	FUNCTIONS ON REALS	191
9.1	HOME constants	191
9.2	The symbolic constants of the CAS: <code>e pi i infinity inf euler_gamma</code>	191
9.3	Booleans	191
9.3.1	Boolean values: <code>true false</code>	191
9.3.2	Tests: <code>== != > >= < <=</code>	191
9.3.3	Boolean operators: <code>or xor and not</code>	192
9.4	Bit to bit operators	193
9.4.1	operators <code>bitor, bitxor, bitand</code>	193
9.4.2	Bit to bit Hamming distance of: <code>hamdist</code>	194
9.5	Usual functions	194
9.6	The smallest integer greater than or equal to the argument: <code>CEILING ceiling</code>	194
9.7	Integer part of a real: <code>FLOOR floor</code>	195
9.8	Argument without its fractional part: <code>IP</code>	195
9.9	Fractional part: <code>FP</code>	196
9.10	Round a real or a complex to n decimal places: <code>ROUND round</code>	196
9.11	Truncate a real or a complex to n decimal places: <code>TRUNCATE trunc</code>	198
9.12	The fractional part of a real: <code>frac</code>	199
9.13	The real without its fractional part: <code>iPart</code>	199
9.14	Mantissa of a real: <code>MANT</code>	199
9.15	Integer part of the logarithm basis 10 of a real: <code>XPON</code>	200
CHAPTER 10	ARITHMETIC	201
10.1	Maximum of two or several values: <code>MAX max</code>	201
10.2	Minimum of two or several values: <code>MIN min</code>	201
10.3	<code>MOD</code>	201
10.4	<code>FNROOT</code>	202
10.5	N-th root: <code>NTHROOT surd</code>	202
10.6	<code>%</code>	203
10.7	Complex	203
10.7.1	The key <code>i</code>	203
10.7.2	Argument: <code>ARG arg</code>	204
10.7.3	Conjugate: <code>CONJ conj</code>	204
10.7.4	Imaginary part: <code>IM im</code>	204

10.7.5	Real part: <code>RE re</code>	204
10.7.6	Sign: <code>SIGN sign</code>	205
10.7.7	The key Shift +/-: <code>ABS abs</code>	205
10.7.8	Write of complex in the form of $\mathbf{re}(z) + i * \mathbf{im}(z)$: <code>evalc</code>	205
10.7.9	Multiply by the complex conjugate: <code>mult_c_conjugate</code>	206
10.8	Exponential and Logarithms	206
10.8.1	Function neperian logarithm: <code>LN ln log</code>	206
10.8.2	Function logarithm basis 10: <code>LOG log10</code>	207
10.8.3	Function logarithm basis b : <code>logb</code>	207
10.8.4	Function antilogarithm: <code>ALOG alog10</code>	208
10.8.5	Function exponential: <code>EXP exp</code>	208
10.8.6	Function <code>EXPM1</code>	209
10.8.7	Function <code>LNP1</code>	209
CHAPTER 11	TRIGONOMETRIC FUNCTIONS	211
11.1	The keys of trigonometric functions	211
11.2	Cosecant: <code>CSC csc</code>	212
11.3	Arccosecant: <code>ACSC acsc</code>	212
11.4	Secant: <code>SEC sec</code>	213
11.5	Arcsecant: <code>ASEC asec</code>	213
11.6	Cotangent: <code>COT cot</code>	213
11.7	Arccotangent: <code>ACOT acot</code>	214
CHAPTER 12	HYPERBOLIC FUNCTIONS	215
12.1	Hyperbolic sine: <code>SINH sinh</code>	215
12.2	Hyperbolic arc sine: <code>ASINH asinh</code>	215
12.3	Hyperbolic cosine: <code>COSH cosh</code>	215
12.4	Hyperbolic arc cosine: <code>ACOSH acosh</code>	216
12.5	Hyperbolic tangent: <code>TANH tanh</code>	216
12.6	Hyperbolic arc tangent: <code>ATANH atanh</code>	216
12.7	Other functions	217
12.7.1	List of variables: <code>lname</code>	217
12.7.2	List of variables and expressions: <code>lvar</code>	217
12.7.3	List of variables and algebraic expressions: <code>algvar</code>	218
12.7.4	Testing the presence of a variable in an expression: <code>has</code>	218
12.7.5	Evaluate an expression: <code>eval</code>	219
12.7.6	Not evaluating an expression: <code>QUOTE quote '</code>	220
12.7.7	Numerical evaluation: <code>evalf approx</code>	220
12.7.8	Rational approximation: <code>exact</code>	221

CHAPTER 13	PROBABILITY FUNCTIONS	222
13.1	Factorial: factorial !	222
13.2	Number of combinations of p objects among n: COMB comb	222
13.3	Number of permutations of p objects among n: PERM perm	222
13.4	Random numbers	223
13.4.1	Random number (real or integer): RANDOM	223
13.4.2	Random integer: RANDINT	224
13.4.3	Rand function of the CAS: rand	224
13.4.4	Random permutation: randperm	227
13.4.5	Generating a random list: randvector	227
13.4.6	Draw according to a multinomial law with programs	229
13.4.7	Draw according to a normal distribution: RANDNORM randNorm	230
13.4.8	Draw according to an exponential law: randexp	230
13.4.9	Initializing the series of random numbers: RANDSEED RandSeed srand	231
13.4.10	Function UTPC	231
13.4.11	Function UTPF	231
13.4.12	Function UTPN	231
13.4.13	Function UTPT	232
13.5	Density of probability	232
13.5.1	Density of probability of the normal distribution: NORMALD normald	232
13.5.2	Density of probability of the Student law: STUDENT student	233
13.5.3	Density of probability of the χ^2 : CHISQUARE chisquare	233
13.5.4	Density of probability of the Fisher law: FISHER fisher snedecor	233
13.5.5	Density of probability of the binomial law: BINOMIAL binomial	233
13.5.6	Density of probability of the Poisson law: POISSON poisson	234
13.6	Function of distribution	234
13.6.1	Function of distribution of the normal distribution: NORMALD_CDF normald_cdf	234
13.6.2	Function of distribution of the Student law: STUDENT_CDF student_cdf	235
13.6.3	Function of distribution of the χ^2 law: CHISQUARE_CDF chisquare_cdf	236
13.6.4	The function of distribution of the Fisher-Snedecor law: FISHER_CDF fisher_cdf snedecor_cdf	236
13.6.5	Function of distribution of the binomial law: BINOMIAL_CDF binomial_cdf	237
13.6.6	Function of distribution of the Poisson law: POISSON_CDF poisson_cdf	238
13.7	Inverse distribution function	238
13.7.1	Inverse normal distribution function: NORMALD_ICDF normald_icdf	238
13.7.2	Inverse distribution Student's function: STUDENT_ICDF student_icdf	239
13.7.3	Inverse function of the function of distribution of the χ^2 law: CHISQUARE_ICDF chisquare_icdf	240
13.7.4	Inverse of the function of distribution of the Fisher-Snedecor law: FISHER_ICDF fisher_icdf snedecor_icdf	240
13.7.5	Inverse distribution function of the binomial law: BINOMIAL_ICDF binomial_icdf	240
13.7.6	Inverse distribution function of Poisson: POISSON_ICDF poisson_icdf	241
CHAPTER 14	STATISTICS FUNCTIONS	242
14.1	Statistics functions at one variable	242
14.1.1	The mean: mean	242
14.1.2	The standard deviation: stddev	243
14.1.3	The standard deviation of the population: stddevp stdDev	243

14.1.4	The variance: <code>variance</code>	244
14.1.5	The median: <code>median</code>	245
14.1.6	Different statistics values: <code>quartiles</code>	245
14.1.7	The first quartile: <code>quartile1</code>	246
14.1.8	The third quartile: <code>quartile3</code>	246
14.1.9	The quantile: <code>quantile</code>	246
14.1.10	The histogram: <code>histogram</code>	247
14.1.11	The covariance: <code>covariance</code>	248
14.1.12	The correlation: <code>correlation</code>	250
14.1.13	Covariance and correlation: <code>covariance_correlation</code>	251
14.1.14	Polygonal line: <code>polygonplot</code>	252
14.1.15	Polygonal line: <code>plotlist</code>	252
14.1.16	Polygonal line and cloud of plots: <code>polygonscatterplot</code>	253
14.1.17	Linear interpolation: <code>linear_interpolate</code>	253
14.1.18	Linear regression: <code>linear_regression</code>	254
14.1.19	Exponential regression: <code>exponential_regression</code>	255
14.1.20	Logarithmic regression: <code>logarithmic_regression</code>	255
14.1.21	Polynomial regression: <code>polynomial_regression</code>	257
14.1.22	Power regression: <code>power_regression</code>	257
14.1.23	Logistic regression: <code>logistic_regression</code>	258
 CHAPTER 15 STATISTICS		 261
15.1	Statistics functions on a list: <code>mean</code> , <code>variance</code> , <code>stddev</code> , <code>stddevp</code> , <code>median</code> , <code>quantile</code> , <code>quartiles</code> , <code>quartile1</code> , <code>quartile3</code>	261
15.1.1	Statistics functions on the columns of a matrix: <code>mean</code> , <code>stddev</code> , <code>variance</code> , <code>median</code> , <code>quantile</code> , <code>quartiles</code>	263
15.2	Tables indexed by two strings: <code>table</code>	265
 CHAPTER 16 LISTS		 267
16.1	Function <code>MAKELIST</code> <code>makelist</code>	267
16.2	Function <code>SORT</code> <code>sort</code>	268
16.3	Function <code>REVERSE</code>	268
16.4	Concatenate: <code>CONCAT</code> <code>concat</code>	268
16.4.1	Add an element at the end of a list: <code>append</code>	270
16.4.2	Add an element at the beginning of a list: <code>prepend</code>	270
16.5	Position in a list: <code>POS</code>	271
16.6	Function <code>DIM</code> <code>dim</code> <code>SIZE</code> <code>size</code> <code>length</code>	271
16.6.1	Get the reversed list: <code>revlist</code>	272
16.6.2	Get the list swapped starting from its n-th element: <code>rotate</code>	273
16.6.3	Get the list shifted starting from its n-th element: <code>shift</code>	273
16.6.4	Removing an element from a list: <code>suppress</code>	274
16.6.5	Get the list without its first element: <code>tail</code>	274
16.6.6	Removing elements from a list: <code>remove</code>	274
16.6.7	Right and left part straight of a list: <code>right</code> , <code>left</code>	275
16.6.8	Checking whether an element is in a list: <code>member</code>	275
16.6.9	Checkin whether an element is in a list: <code>contains</code>	276
16.6.10	Counting the elements of a list or of a matrix such as a property: <code>count</code>	276
16.6.11	Select elements of a list: <code>select</code>	278

16.7	List of differences between consecutive terms: <code>ΔLIST</code> <code>deltalist</code>	279
16.8	Sum of the elements of a list: <code>ΣLIST</code> <code>sum</code>	279
16.9	Product of the elements of a list: <code>ΠLIST</code> <code>product</code>	280
16.9.1	Apply a function of one variable to the elements of a list: <code>map</code> <code>apply</code>	280
16.9.2	Apply a function of two variables to elements of two lists: <code>zip</code>	282
16.10	Convert a list to a matrix: <code>list2mat</code>	283
16.11	Convert a matrix to a list: <code>mat2list</code>	283
16.12	Useful functions for the lists and the components of a vector	283
16.12.1	Norms of a vector: <code>maxnorm</code> <code>l1norm</code> <code>l2norm</code> <code>norm</code>	283
16.12.2	Normalizing the components of a vector: <code>normalize</code>	284
16.12.3	Cumulated sums of the elements of a list: <code>cumSum</code>	285
16.12.4	Term by term sum of two lists: <code>+</code> <code>.+</code>	285
16.12.5	Term by term difference of two lists: <code>-</code> <code>.-</code>	286
16.12.6	Term by term product of two lists: <code>.*</code>	287
16.12.7	Quotient term by term of two lists: <code>./</code>	287
CHAPTER 17	STRINGS OF CHARACTERS	288
17.1	Write a string or a character: <code>"</code>	288
17.1.1	To concatenate two numbers and strings: <code>cat</code> <code>+</code>	289
17.1.2	Concatenating a sequence of words: <code>cumSum</code>	289
17.1.3	Finding a character in a string: <code>INSTRING</code> <code>inString</code>	290
17.2	ASCII codes: <code>ASC</code> <code>asc</code>	290
17.3	Character from ASCII code: <code>CHAR</code> <code>char</code>	291
17.3.1	Converting a real or an integer into a string: <code>string</code>	291
17.4	Use a string as a number or a command: <code>expr</code>	292
17.4.1	Use a string as a number	292
17.4.2	Use a string as a command name	293
17.5	Evaluate an expression in the form of a string: <code>string</code>	293
17.6	inString	294
17.7	Left part of a string: <code>left</code>	294
17.8	Right part of a string: <code>right</code>	294
17.9	Mid part of a string: <code>mid</code>	295
17.10	Rotate last character: <code>rotate</code>	295
17.11	Length of a string: <code>dim</code> <code>DIM</code> <code>size</code> <code>SIZE</code> <code>length</code>	295
17.12	Concatenate two strings: <code>+</code>	296
17.13	Get the list or the string without its first element: <code>tail</code>	297
17.14	First element of a list or of a string: <code>head</code>	297

CHAPTER 18	POLYNOMIALS	298
18.1	Coefficients of a polynomial: <code>POLYCOEF</code>	298
18.2	Polynomial from coefficients: <code>POLYEVAL</code>	298
18.3	Expand a polynomial: <code>POLYFORM</code>	299
18.4	Roots of a polynomial from its coefficients: <code>POLYROOT</code>	301
CHAPTER 19	RECURRENT SEQUENCES	302
19.1	Values of a recurrent sequence or of a system of recurrent sequences: <code>seqsolve</code>	302
19.2	Values of a recurrent sequence or of a system of recurrent sequences: <code>rsolve</code>	304
CHAPTER 20	MATRICES	307
20.1	Generalities	307
20.2	Definition	307
20.2.1	Dimension of a matrix: <code>dim</code>	307
20.2.2	Number of rows: <code>rowDim</code>	308
20.2.3	Number of columns: <code>colDim</code>	308
20.3	Operations on rows and columns useful in programming	308
20.3.1	Add a column to a matrix: <code>ADDCOL</code>	308
20.3.2	Swap rows: <code>SWAPROW rowSwap</code>	309
20.3.3	Swap columns: <code>SWAPCOL colSwap</code>	309
20.3.4	Extract rows from a matrix: <code>row</code>	310
20.3.5	Extract columns from a matrix: <code>col</code>	310
20.3.6	Remove columns from a matrix: <code>DELCOL delcols</code>	310
20.3.7	Remove rows from a matrix: <code>DELROW delrows</code>	311
20.3.8	Extract a sub-matrix from a matrix: <code>SUB subMat</code>	312
20.3.9	Redimension a matrix or a vector: <code>REDIM</code>	313
20.3.10	Replace a portion of a matrix or of a vector: <code>REPLACE</code>	313
20.3.11	Add a row to a matrix: <code>ADDROW</code>	314
20.3.12	Add a row to another: <code>rowAdd</code>	314
20.3.13	Multiply a row by an expression: <code>SCALE mRow</code>	315
20.3.14	Add k times a row to another: <code>SCALEADD mRowAdd</code>	315
20.4	Creation and arithmetic of matrices	315
20.4.1	Addition and subtraction of matrices: <code>+ - .+ .-</code>	315
20.4.2	Multiplication of matrices: <code>* &*</code>	316
20.4.3	Rising a matrix to an integer power: <code>^ &^</code>	316
20.4.4	Hadamard product (infix version): <code>.*</code>	317
20.4.5	Hadamard division (infix version): <code>./</code>	317
20.4.6	Hadamard power (infix version): <code>.^</code>	317
20.5	Transpose matrix: <code>transpose</code>	317
20.6	Conjugate transpose matrix: <code>TRN trn</code>	317
20.7	Determinant: <code>DET det</code>	318
20.7.1	Characteristic polynomial: <code>charpoly</code>	318

20.8	Vectorial field and linear applications	320
20.8.1	Basis of a vectorial subspace: <code>basis</code>	320
20.8.2	Intersection basis of two vectorial subspaces: <code>ibasis</code>	320
20.8.3	Image of a linear application: <code>image</code>	320
20.8.4	Kernel of a linear application: <code>ker</code>	320
20.9	Solve a linear system: RREF <code>rref</code>	321
20.9.1	Solve of $A * X = B$: <code>simult</code>	322
20.10	Make matrices	323
20.10.1	Make a matrix from an expression: <code>MAKEMAT</code> <code>makemat</code>	323
20.10.2	Matrix of zeros: <code>matrix</code>	323
20.10.3	Matrix identity: <code>IDENMAT</code> <code>identity</code>	323
20.10.4	Matrix random: <code>RANDMAT</code> <code>randMat</code> <code>randmatrix</code> <code>ramn</code>	324
20.10.5	Jordan block: <code>JordanBlock</code>	325
20.10.6	N-th Hilbert matrix: <code>hilbert</code>	325
20.10.7	Matrix of an isometry: <code>mkisom</code>	325
20.10.8	Vandermonde matrix: <code>vandermonde</code>	326
20.11	Basics	326
20.11.1	Schur norm or Frobenius norm of a matrix: <code>ABS</code>	326
20.11.2	Maximum of the norms of the rows of a matrix: <code>ROWNORM</code> <code>rownorm</code>	327
20.11.3	Maximum of matrix norms of matrix columns of a matrix: <code>COLNORM</code> <code>colnorm</code>	328
20.11.4	Spectral norm of a matrix: <code>SPECNORM</code>	329
20.11.5	Spectral radius of a square matrix: <code>SPECRAD</code>	329
20.11.6	Condition number of an invertible square matrix: <code>COND</code> <code>cond</code>	330
20.11.7	Rank of a matrix: <code>RANK</code> <code>rank</code>	331
20.11.8	Step of the Gauss-Jordan reduction of a matrix: <code>pivot</code>	332
20.11.9	Trace of a square matrix: <code>TRACE</code> <code>trace</code>	332
20.12	Advanced	333
20.12.1	Eigenvalues: <code>EIGENVAL</code> <code>eigenvals</code>	333
20.12.2	Eigenvectors: <code>EIGENVV</code> <code>eigenvects</code>	334
20.12.3	Jordan matrix: <code>eigVl</code>	334
20.12.4	Jordan matrix and its transfer matrix: <code>jordan</code>	335
20.12.5	Power n of a square matrix: <code>matpow</code>	335
20.12.6	Diagonal matrix and its diagonal: <code>diag</code>	336
20.12.7	Cholesky matrix: <code>cholesky</code>	336
20.12.8	Hermite normal form of a matrix: <code>ihermite</code>	336
20.12.9	Matrix reduction to Hessenberg form: <code>hessenberg</code>	336
20.12.10	Smith normal form of a matrix: <code>ismith</code>	338
20.13	Factorization	338
20.13.1	LQ decomposition of a matrix: <code>LQ</code>	338
20.13.2	Minimal norm of the linear system $A * X = B$: <code>LSQ</code>	339
20.13.3	LU decomposition of a square matrix: <code>LU</code>	340
20.13.4	LU decomposition: <code>lu</code>	341
20.13.5	QR decomposition of a square matrix: <code>QR</code> <code>qr</code>	342
20.13.6	Matrix reduction to Hessenberg form: <code>SCHUR</code> <code>schur</code>	343
20.13.7	Singular value decomposition: <code>SVD</code> <code>svd</code>	343
20.13.8	Singular values: <code>SVL</code> <code>svl</code>	345
20.14	Vector	346
20.14.1	Cross product: <code>CROSS</code> <code>cross</code>	346
20.14.2	Dot product: <code>DOT</code> <code>dot</code>	346
20.14.3	Norm l2: <code>l2norm</code>	347
20.14.4	Norm l1: <code>l1norm</code>	347

20.14.5	Norm of the maximum: <code>maxnorm</code>	348
CHAPTER 21 SPECIAL FUNCTIONS		349
21.1	β function: <code>Beta</code>	349
21.2	Γ function: <code>Gamma</code>	350
21.3	Derivatives of the DiGamma function: <code>Psi</code>	351
21.4	The ζ function: <code>Zeta</code>	352
21.5	<i>erf</i> function: <code>erf</code>	352
21.6	<i>erfc</i> function: <code>erfc</code>	353
21.7	Exponential integral function: <code>Ei</code>	354
21.8	Sine integral function: <code>Si</code>	355
21.9	Cosine integral function: <code>Ci</code>	356
21.10	<i>Heaviside</i> function: <code>Heaviside</code>	356
21.11	<i>Dirac</i> distribution: <code>Dirac</code>	357
CHAPTER 22 CONSTANTS AND CALCULATIONS WITH UNITS		358
22.1	Shifted key Units	358
22.2	Units	358
22.2.1	Notation of units	358
22.2.2	Avalaible prefixes for units names	358
22.2.3	Calculations with units	359
22.3	Tools	360
22.3.1	Conversion of a unit object to another unit: <code>convert =></code>	360
22.3.2	Units conversion to MKSA units: <code>mksa</code>	361
22.3.3	Factorize a unit in a unit object: <code>ufactor</code>	361
22.3.4	Simplify a unit: <code>usimplify</code>	362
22.4	Physics constants	362
22.5	Units	362
22.5.1	Units notation	362
22.5.2	Calculations with units	362
22.5.3	Conversion of a unit object into another unit: <code>convert =></code>	363
22.5.4	Units conversion to MKSA units: <code>mksa</code>	365
22.5.5	Conversions between degree Celsius and Fahrenheit: <code>Celsius2Fahrenheit</code> <code>Fahrenheit2Celsius</code>	365
22.5.6	Factorization of a unit: <code>ufactor</code>	366
22.5.7	Simplify a unit: <code>usimplify</code>	366
22.6	Constants	366
22.6.1	Notation of chemical, physics or quantum mechanics constants.	366
22.6.2	Physics constants library	367

CHAPTER 23	FUNCTIONS OF 3D GEOMETRY	368
23.1	Common perpendicular to two 3D lines: <code>common_perpendicular</code>	368
PART IV THE APPLICATIONS AND THE APPS KEY		369
CHAPTER 24	THE MENU GEOMETRY	370
24.1	Generalities	370
24.2	Point	371
24.2.1	Point defined as barycenter of n points: <code>barycenter</code>	371
24.2.2	Point in geometry: <code>point</code>	372
24.2.3	Midpoint of a segment: <code>midpoint</code>	373
24.2.4	Isobarycenter of n points: <code>isobarycenter</code>	374
24.2.5	Randomly define a 2D point: <code>point2d</code>	374
24.2.6	Polar point in plane geometry: <code>polar_point</code>	375
24.2.7	One of the intersection points of two geometrical objects: <code>single_inter</code>	375
24.2.8	All intersection points of two geometrical objects: <code>inter</code>	376
24.2.9	Orthocenter of a triangle: <code>orthocenter</code>	377
24.2.10	Vertices of a polygon: <code>vertices</code>	377
24.2.11	Vertices of a polygon: <code>vertices_abca</code>	378
24.2.12	Point on a geometrical object: <code>element</code>	378
24.2.13	Point dividing a segment: <code>division_point</code>	380
24.2.14	Harmonic division: <code>harmonic_division</code>	381
24.2.15	Harmonic conjugate: <code>harmonic_conjugate</code>	381
24.2.16	Pole and polar: <code>pole polar</code>	382
24.2.17	Reciprocal polar: <code>reciprocation</code>	382
24.2.18	The center of a circle: <code>center</code>	382
24.3	Line	383
24.3.1	Line defined by a point and a slope: <code>DrawSlp</code>	383
24.3.2	Tangent to the curve of $y = f(x)$ in $x = a$: <code>LineTan</code>	383
24.3.3	Altitude of a triangle: <code>altitude</code>	383
24.3.4	Internal bisector of an angle: <code>bisector</code>	384
24.3.5	External bisector of an angle: <code>exbisector</code>	384
24.3.6	Half line: <code>half_line</code>	384
24.3.7	Line and oriented line: <code>line</code>	385
24.3.8	Segment: <code>Line</code>	386
24.3.9	Plot of a 2D horizontal line: <code>LineHorz</code>	386
24.3.10	Plot of a 2D vertical line: <code>LineVert</code>	386
24.3.11	Vector in plane geometry: <code>vector</code>	387
24.3.12	Median line of a triangle: <code>median_line</code>	388
24.3.13	Parallel lines: <code>parallel</code>	388
24.3.14	Perpendicular bisector: <code>perpen_bisector</code>	388
24.3.15	Line perpendicular to a line: <code>perpendicular</code>	389
24.3.16	Segment: <code>segment</code>	389
24.3.17	Tangent to a geometrical object or tangent to a curve in a point: <code>tangent</code>	389
24.3.18	Radical axis of two circles: <code>radical_axis</code>	391
24.4	Polygon	391
24.4.1	Scalene triangle: <code>triangle</code>	391
24.4.2	Equilateral triangle: <code>equilateral_triangle</code>	391
24.4.3	Right triangle: <code>right_triangle</code>	392
24.4.4	Isosceles triangle: <code>isosceles_triangle</code>	393

24.4.5	Rhombus: rhombus	393
24.4.6	Rectangle: rectangle	394
24.4.7	Square: square	395
24.4.8	Quadrilateral: quadrilateral	396
24.4.9	Parallelogram: parallelogram	396
24.4.10	Isopolygon: isopolygon	397
24.4.11	Hexagon: hexagon	397
24.4.12	Polygon: polygon	398
24.4.13	Polygonal line: open_polygon	398
24.4.14	Convex hull of points of the plan: convexhull	399
24.5	Curves	399
24.5.1	Circle and arcs: circle	399
24.5.2	Arcs of circle: arc ARC	401
24.5.3	Circumcircle: circumcircle	401
24.5.4	Plot of a conic: conic	402
24.5.5	Ellipse: ellipse	402
24.5.6	Excircle: excircle	403
24.5.7	Hyperbola: hyperbola	403
24.5.8	Incircle: incircle	404
24.5.9	Locus and envelope: locus	404
24.5.10	Parabola: parabola	406
24.5.11	Power of a point according to a circle: powerpc	406
24.6	Transformation	407
24.6.1	Homothety: homothety	407
24.6.2	Inversion: inversion	407
24.6.3	Orthogonale projection: projection	408
24.6.4	Symmetry line and symmetry point: reflection	409
24.6.5	Rotation: rotation	410
24.6.6	Similarity: similarity	411
24.6.7	Translation: translation	411
24.7	Measure and graphics	412
24.7.1	Measure of a angle: angleat	412
24.7.2	Measure of a angle: angleatraw	413
24.7.3	Display of the area of a polygon: areaat	413
24.7.4	Area of a polygon: areaatraw	414
24.7.5	Length of a segment: distanceat	414
24.7.6	Length of a segment: distanceatraw	415
24.7.7	Perimeter of a polygon: perimeterat	416
24.7.8	Perimeter of a polygon: perimeteratraw	417
24.7.9	Slope of a line: slopeat	418
24.7.10	Slope of a line: slopeatraw	418
24.8	Measure	420
24.8.1	Abscissa of a point or of a vector: abscissa	420
24.8.2	Affix of a point or of a vector: affix	420
24.8.3	Measure of a angle: angle	421
24.8.4	Length of an arc of curve: arcLen	422
24.8.5	Area of a polygon: area	423
24.8.6	Coordinates of a point, a vector or a line: coordinates	423
24.8.7	Rectangular coordinates of a point: rectangular_coordinates	425
24.8.8	Polar coordinates of a point: polar_coordinates	426
24.8.9	Length of a segment and distance between two geometrical objects: distance	426
24.8.10	Square of the length of a segment: distance2	427
24.8.11	Cartesian equation of a geometrical object: equation	427

24.8.12	Get as answer the value of a measure displayed: <code>extract_measure</code>	427
24.8.13	Ordinate of a point or of a vector: <code>ordinate</code>	428
24.8.14	Parametric equation of a geometrical object: <code>parameq</code>	429
24.8.15	Perimeter of a polygon: <code>perimeter</code>	429
24.8.16	Radius of a circle: <code>radius</code>	429
24.8.17	Slope of a line: <code>slope</code>	430
24.9	Test	431
24.9.1	Check whether three points are collinear: <code>is_collinear</code>	431
24.9.2	Check whether four points are concyclic: <code>is_concyclic</code>	431
24.9.3	Check whether elements are conjugates: <code>is_conjugate</code>	431
24.9.4	Check whether points or/and lines are coplanar: <code>is_coplanar</code>	432
24.9.5	Check whether a point is on a geometrical object: <code>is_element</code>	433
24.9.6	Check whether a triangle is equilateral: <code>is_equilateral</code>	433
24.9.7	Check whether a triangle is isoscele: <code>is_isosceles</code>	434
24.9.8	Orthogonality of two lines or two circles: <code>is_orthogonal</code>	434
24.9.9	Check whether two lines are parallel: <code>is_parallel</code>	435
24.9.10	Check whether a polygon is a parallelogram: <code>is_parallelogram</code>	435
24.9.11	Check whether two lines are perpendicular: <code>is_perpendicular</code>	436
24.9.12	Check whether a triangle is right or a polygon is a rectangle: <code>is_rectangle</code>	437
24.9.13	Check whether a polygon is a rhombus: <code>is_rhombus</code>	437
24.9.14	Check whether a polygon is a square: <code>is_square</code>	438
24.9.15	Check whether 4 points form an harmonic division: <code>is_harmonic</code>	439
24.9.16	Check whether lines are in harmonic bundle: <code>is_harmonic_line_bundle</code>	439
24.9.17	Check whether circles are in harmonic bundle: <code>is_harmonic_circle_bundle</code>	439
24.10	Exercises of geometry	440
24.10.1	Transformations	440
24.10.2	Loci	440
24.11	Geometry activities	441
CHAPTER 25	THE SPREADSHEET	448
25.1	Generalities	448
25.2	Screen of the spreadsheet	448
25.2.1	Copy the content of a cell to another	448
25.2.2	Relative and absolute referencces	448
25.3	Functions of the spreadsheet	449
25.3.1	Function <code>SUM</code>	449
25.3.2	Function <code>MEAN</code>	449
25.3.3	Function <code>AMORT</code>	449
25.3.4	Function <code>STAT1</code>	449
25.3.5	Function <code>REGRS</code>	449
25.3.6	Functions <code>PredY</code> <code>PredX</code>	449
25.3.7	Functions <code>HypZ1mean</code> <code>HypZ2mean</code>	450
25.4	Use of the spreadsheet based on examples	450
25.4.1	Exercise 1	450
25.4.2	Exercise 2	451
CHAPTER 26	OTHER APPLICATIONS	454
26.1	Function application	454

26.2	Sequence application	454
26.2.1	Fibonacci sequence	454
26.2.2	GCD	455
26.2.3	Bezout identity	455
26.3	Parametric application	456
26.4	Polar application	456
26.5	Solve application	457
26.6	Finance application	457
26.7	Linear Solver application	458
26.8	Triangle Solver application	459
26.9	1-Var Statistics	459
26.10	2-Var statistics	460
26.10.1	Exercises	461
26.11	Inference application	467
26.11.1	Frequency of a parameter and hypothesis based on samples	468
26.11.2	Samples extracted from a normal distribution	472
26.11.3	Samples extracted from a Student distribution	474
PART V	PROGRAMMING	476
CHAPTER 27	GENERALITIES	477
27.1	Syntax of HOME programs and CAS programs	477
27.2	Writing a program slightly different from an existing program	477
CHAPTER 28	PROGRAMMING INSTRUCTIONS	479
28.1	Variables	479
28.1.1	Variables names	479
28.1.2	Comments: <code>comment //</code>	479
28.1.3	Inputs: <code>INPUT input InputStr</code>	479
28.1.4	Outputs: <code>print</code>	480
28.1.5	Assignment instruction: <code>=> :=</code> 	481
28.1.6	Copy without evaluating the content of a variable: <code>CopyVar</code>	481
28.1.7	Function testing the type of its argument: <code>TYPE type</code>	482
28.1.8	Function testing the type of its argument: <code>compare</code>	483
28.1.9	Stating an assumption about a variable: <code>assume</code>	484
28.1.10	State an additional assumption about a variable: <code>additionally</code>	487
28.1.11	Know the assumptions stated about a variable: <code>about</code>	488
28.1.12	Delete the content of a variable: <code>purge</code>	488
28.1.13	Delete the content of all the variables: <code>restart</code>	489
28.1.14	Access to answers: <code>Ans ans(n)</code>	489
28.2	Conditionnal instructions	489

28.3	Loops	492
28.3.1	Instructions FOR FROM TO DO END and FOR FROM TO STEP DO END	492
28.3.2	Iterative loops: ITERATE	492
28.3.3	Instruction WHILE DO END	492
28.3.4	Instruction REPEAT UNTIL	492
28.3.5	Instruction BREAK	493
28.3.6	Function seq	493
28.4	Comments: //	494
28.5	Variables	494
28.6	Boolean operators: < <= == != > >=	494
28.7	Commands of applications	497
CHAPTER 29 HOW TO PROGRAM		499
29.1	Conditional instruction IF	499
29.2	FOR and WHILE loops	500
29.2.1	Make the calculator count by step of one and display the result	500
29.2.2	Make the calculator count by step of 1 by using a list or a sequence	501
29.3	Approximate value of the sum of a sequence	502
29.3.1	Sequence of general term $un = 1n^2$	502
29.3.2	Sequence of general term $vn = -1n + 1n$	503
29.3.3	The sequence of general term $wn = 1n$ is divergent	504
29.4	Decimal form of a fraction	505
29.4.1	With no program	505
29.4.2	With a CAS program	506
29.5	29.5 Newton method and Heron algorithm	507
29.5.1	29.5.1 Newton method	507
29.5.2	Newton algorithm	508
29.5.3	Heron algorithm	508
CHAPTER 30 EXAMPLE OF PROGRAMS		510
30.1	GCD and Bezout identity from Home	510
30.1.1	GCD	510
30.1.2	Bezout identity for A and B	510
30.2	GCD and Bezout identity from the CAS	512
30.2.1	GCD with the CAS with no program	512
30.2.2	GCD with a CAS program	512
30.2.3	Bezout identity with the CAS, with no program	512
30.2.4	Bezout identity with a CAS program	512

Part I **Getting started**

Generalities

With the HP Prime calculator you have two calculators in one: one to do symbolic and exact computation (key CAS), the other to do approximate calculation (key HOME). This is the fruit of the union in the calculator of two softwares; Giac/Xcas for the CAS and the software developed by HP for their scientific and graphic calculators in HOME. These two logics are often contradictory, which required a huge effort of consistency to allow the use of HOME data in the CAS and reciprocally, which effort being still continued up to today.

Then, the logic of a symbolic computation software is to not have pre-assigned variable and to allow to store any kind of data in a variable which name is free (in particular a name of variable may be more than one letter long) whereas the logic of calculators HP38/39/40 was to have pre-assigned variables which name is a letter or a letter followed by a digit, and storing one kind only of data: A, B..Z for reals, Z0, ..Z9 for complex, L0, L1..L9 for lists, M0, M1..M9 for vectors or matrices etc., This has of course major consequences, if we write `ab` in the CAS, this designates a variable with a two-letter name, whereas `AB` in HOME designates the product (implicit multiplication) of variables A and B.

To avoid confusion, it is advised to use names of variables in lower case in the CAS, names of CAS commands being in lower case (exceptions aside), while names of command in HOME are in upper case. This choice is eased by the lock of alphabetic keyboard in lower case in the CAS and in upper case in HOME. Many commands exist in the two versions (HOME in upper case, CAS in lower case), most of the time they do the same thing, but, unfortunately there are exceptions, for example `size` and `SIZE` (see below).

Please also note that in HOME, there is a difference between the lists (`L1:={1,2,3}`) and the vectors (`M1:=[1,2,3]`) and the notion of sequence does not exist, whereas in the CAS there is no difference between lists and vectors (`v:=[1,2,3]` or `v:={1,2,3}`) and we may work with a sequence (`s:=1,2,3`).

More to say, warning! The history does not always reflect what has been typed in, in the history of HOME the lower case letters are changed into upper case letters whereas in the history of CAS, it depends on whether `Textbook` is checked or not in General Setting (Shift (Settings)).

Example:

In HOME screen or in the CAS screen, we enter:

`SIZE(1,2,3)` or `size(1,2,3)`

We get in the history `SIZE(1,2,3)` and as a result: 3

In HOME screen, we enter:

`SIZE([1,2,3])` or `size([1,2,3])`

We get in the history `SIZE([1,2,3])` and as a result: {3}

In the CAS screen, we enter:

`SIZE([1,2,3])` or `size([1,2,3])`

We get in the history (if we did not check Textbook):

`SIZE([1,2,3])` or `size([1,2,3])` and as a result: 3

In HOME screen, we enter:

`SIZE([[1,2,3],[4,5,6]])` or `size([[1,2,3],[4,5,6]])`

We get in the history `SIZE([[1,2,3],[4,5,6]])` and as a result: {2,3}

In the CAS screen, we enter:

`size([[1,2,3],[4,5,6]])`

We get in the history (if we did not check Textbook):

`size([[1,2,3],[4,5,6]])`

and as a result: 2

but if we enter in the CAS:

`SIZE([[1,2,3],[4,5,6]])`

We get in the history (if we did not check Textbook):

`SIZE([[1,2,3],[4,5,6]])` and as a result: [2,3]

ADVICE: make your choice: either you always work in HOME, either in the CAS because commands with same name not returning the same thing in HOME or in CAS quickly becomes a true brain teaser! Note for the users of Xcas: the getting started phase of the calculator mode CAS should be quick. However, please note what follows:

- some commands are not available, as HP did not wish to implement them (for example all the commands on permutations)
- some synonyms are not available, and unfortunately HP did not make the choice of Xcas native commands in lower case but the choice of mixed commands with a mixed name with an upper case letter in the middle of the command name.
- interface for the use of the programming language of Xcas is still perfectible (for example the alphabetic keyboard is locked in upper case even if we select a CAS program, the interface of the function of debugging debug is experimental...)

CAS and HOME keys

With the HP Prime calculator you can choose of working in exact mode or in approximate mode: there are two screens, one to do the exact calculation it is the CAS screen, the other to do the approximate calculation, it is the HOME screen.

In CAS screen, we can also do approximate calculation for example $1/2$ is an exact number and `evalf(1/2)=0.5` is an approximate number. If in one expression there is an approximate number the result will be approximate, for example: $1/2 + 1/3$ returns $5/6$ whereas $0.5 + 1/3$ returns 0.833333333333 .

In CAS screen, commands are in lower case whereas they are in upper case in HOME screen. If you press on CAS, you work in exact mode, if you press on HOME you work in approximate mode.

What does this change ?

For example, we will consider 2 sequences u and v defined by:

$$u_0 = \frac{2}{3}, u_{n+1} = 2u_n - \frac{2}{3} (n \geq 0)$$

and

$$v_0 = \frac{2}{3}, v_{n+1} = 2(u_n - \frac{1}{3}) (n \geq 0)$$

In the CAS screen

We press CAS and we enter to get the first terms of u :

$2/3$ then Enter and we get $2/3$.

We enter:

`2*Ans-2/3` then Enter, Enter, ...

and we get $2/3, 2/3, 2/3...$

In exact mode, *i.e.* in the CAS screen, the sequence u is then stationary and equals $\frac{2}{3}$.

In this case the result is in accordance with the theoretical result.

Still in the CAS, to get the first terms of v , we enter:

$2/3$ then Enter and we get $2/3$.

We enter:

`2*(Ans-1/3)` then Enter, Enter...

and we get $2/3, 2/3, 2/3...$

In exact mode, *i.e.* in the CAS screen, the series v is then stationary and equals $\frac{2}{3}$.

The result, here, is still in accordance with the theoretical result.

In HOME screen

Now we press HOME and to get the first terms of u , we enter the value of u_0 :

$2/3$ then Enter and we get 0.666666666667 then we enter:

`2*Ans-2/3` then Enter, Enter, Enter...

and we get $0.666666666663, 0.666666666663...$

The result is here almost in accordance with the theoretical result.

In approximate mode *i.e.* in HOME screen (key \rightarrow), the sequence u is then stationary starting from $n > 0$ and equals 0.666666666663 .

Still in HOME (key \rightarrow), we enter to get the first terms of v :

$2/3$ then Enter and we get 0.666666666667 .

We enter:

`2*(Ans-1/3)` then Enter, Enter, Enter, ...

and we get

$v_1 = 0.666666666668,$

$v_2 = 0.666666666670,$
 $v_3 = 0.666666666674,$
 then
 $0.666666666682,$
 $0.666666666682,$
 $0.666666666698,$
 $0.6666666666730,$
 $0.6666666666794,$
 0.6666666666922

etc., ...

and after having pressed Enter 51 or 52 times, we get:

$v_{40} = 1.76617829443$ and $v_{50} = 2252.46648036$ etc.. In approximate mode, i.e. in HOME screen (key), the series v then tends to $+\infty$.

We clearly see that, in approximate mode, calculations errors accumulate themselves and that the results displayed are not always in accordance with the theoretical results!

How the calculations are performed in HOME.

In HOME, the real numbers are displayed with at most 12 significative digits but the calculations are performed with more digits and then rounded to be displayed, for example:

$1/3$ will be represented by 0.333333333333 (with 12 times the digit 3)

$2/3$ will be represented by 0.666666666667 (with 11 times the digit 6 and a 7)

$4/3$ will be represented by 1.333333333333 (with 1 then 11 times the digit 3)

$2*0.666666666667$ or $2*0.666666666663$ will be represented by 1.333333333333 (with 1 then 11 times the digit 3)

For the calculation of u we enter u_0 :

$2/3$ we get 0.666666666667 then,

$2*Ans-2/3$ we get $1.333333333333-0.666666666667=0.666666666663$ then,

$2*Ans-2/3$ we get because $1.333333333333-0.666666666667=0.666666666663$

etc., ... The sequence u is then stationary for $n > 0$ and equals 0.666666666663 .

For the sequence v the calculation is done once 2 has been put in factor.

We enter v_0 :

$2/3$ we get 0.666666666667 then, $2*(Ans-1/3)$ in the different operations one always has 12 decimal places, we get:

$2*(0.666666666667-0.333333333333)=2*0.333333333334=0.666666666668.$

Then, we have:

If $A = 0.666666666666$ and $B = 0.333333333333$, we have $A = 2 * B$ and $B = A - B$ but, $2/3 = A + 10^{-12}$ and $1/3 = B$

Then, we have:

$$v_0 = \frac{2}{3} = A + 10^{-12}$$

$$v_1 = 2 * (A + 10^{-12} - B) = 2 * (B + 10^{-12}) = A + 2 * 10^{-12}$$

then

$$v_2 = 2 * (A + 2 * 10^{-12} - B) = 2 * (B + 10^{-12}) = A + 2^2 * 10^{-12}$$

then...

$$v_{38} = A + 2^{38} * 10^{-12} = 0.94154457361$$

$$v_{39} = A + 2^{39} * 10^{-12} = 1.21642248055$$

$$v_{40} = A + 2^{40} * 10^{-12} = 1.76617829445$$

...

$$v_{50} = A + 2^{50} * 10^{-12} = 1126.56657351$$

$$v_{51} = A + 2^{51} * 10^{-12} = 2252.46648036$$

then the formula might not be true anymore due to rounding errors...

If we use the command ITERATE which iterates, starting by the value $2/3$, 90 times the function which to X matches $2*(X-1/3)$, we enter:

ITERATE(2*(X-1/3), x, 2/3, 90)

we get:

1.23794003934E15

and

ITERATE(2*(X-1/3), x, 2/3, 91)

we get:

2.4758800788=2*1.23794003934E15

So $v_n = 2^{n-90} * u_{90}$ and when n tends to the infinite $v_n = 2^{n-90} * u_{90}$ tends to the infinite.

Reset and clear

To reset the calculator:

- Press the keys F O C (not in ALPHA mode),
- Perform a reset with a paper clip by keeping the keys pressed,
- Release the keys then choose 4 FLS Utility, then 3 Format Disk C, then Esc then 9 Reset.

To clear:

- the last character entered, press Del (the big black arrow).
- the entry line, press ON
- the last result or the last command of the history, press Shift-Del
- all the history, press Shift-Esc (Clear) .

Tactile screen

We notice that the menus at the bottom of the screen (here named push buttons) can only be accessed by touching with the finger: there are no soft keys F1 . . . F6 anymore!

The screen is tactile and this allows to easily copy a entry line or an answer to the history, or read or re-read a too long answer, to select a menu then a command of the key .

For this:

- it is enough to look for the command or the answer to be copied by scrolling in the history with a finger, then to select the command or the answer to be copied still with a digit and press Copy on the push buttons when the line is highlighted or to press twice quickly with the finger on the line to be copied,
- to read a too long answer it is enough to sweep the line of the answer with a finger
- we open a menu with a finger or by its number, we do the same if there is a sub-menu, then we select the function with a finger or with its number and that causes the function to be written at left of the entry line: all that is left is to enter the parameters of this function and to make with Enter . The result is then written at the right.

Keys

- CAS
You must press the key CAS to do the symbolic computation. The letters in lower case can then be accessed in ALPHA mode and the key xt0n allows to directly get x.
- HOME
You must press the key HOME to quit the symbolic computation and do numerical calculation.
- Apps
You must press the key Apps to use the different Application which have, each one, 3 views: a Symbolic view which stores the commands that were called (key Symb), a Plot view which executes the graphical commands (key Plot) and a Numeric view for the numerical results (key Num).
- Menu

The key Menu returns a specific menu depending on what we are doing. For instance, from the CAS or from HOME you can exchange data between the CAS screen and the HOME screen, from the Plot screen of the geometry application you can change the color of objects or do filled figures with the Options command (push buttons) or by filling with color, in the Symbolic view, the square located between the cell used to set and the name of the object (by touching this square one opens the color palet).

- Help
The key Help gives help on the different commands that are in the Cmds menu (push buttons) or in the menu of the key:
You must highlight this command with the arrows then press Help or we enter this command and we press Help.
- Esc
The key Esc allows to cancel the command in progress

General settings

We open the screen of the general setting with `Shift-HOME`.

You can for example choose:

- to enter the commands in 2D (choose `Entry: Textbook`),
- to get the answers in 2D (check `Textbook Display`),
- to have the menus displaying the name of the commands rather than a theme (set `Display Menu`),
- to set the calculator in exam mode

CAS settings: `Shift CAS`

We enter: `Shift CAS (Settings)`.

To be in complex mode you must check `i`.

To use of complex variables you must check `Complex`.

For instance:

`solve(x^3+2*x^2+x+2=0, x)` returns `[-2]` in real mode

`solve(x^3+2*x^2+x+2=0, x)` returns `[-2, -i, i]` in complex mode

To use square roots in a factorization you must check:

Use $\sqrt{\quad}$

For instance:

`factor(x^2+x-1)` returns `x^2+x-1` if Use $\sqrt{\quad}$ is not checked

`factor(x^2+x-1)` returns `(x+(-(sqrt(5))+1)/2)*(x+(sqrt(5)+1)/2)` if Use $\sqrt{\quad}$ is checked

Calculator settings: `Shift HOME`

The key `Shift HOME (Settings)` allows to do the settings of the calculator.

To get in the menus or the sub-menus the names of the commands, `Display Menu` must not be checked.

If `Display Menu` is checked, the menus and the sub-menus describe the commands and returns the command when a menu or a sub-menu is selected.

Symbolic computation functions

We access functions of symbolic computation by pressing the key `.`

These functions are sorted by category.

Use `Shift (Settings)` and uncheck `Display Menu` to get the name of the functions and not the description of these functions.

Part II **Menu CAS of the Toolbox key**

Chapter 1 Generalities

1.1 Calculations in the CAS

With the CAS, we do exact calculation.

With the CAS we can use the variables of Home which have as a name one of the upper case letters and which have by default the value 0 but also variables which have as names a string of lower case letters or of digits starting by a letter. These variables have by default no value: these variables are symbolic (without value) as long as we do not affect one to them.

In CAS, the commands are in general in lower case, it is why the key `ALPHA` allows to enter a lower case and `ALPHA`, `ALPHA` locks the keyboard in lower case (no need to press `Shift`).

With the CAS, the simplifications are not done automatically, only the useless parentheses are removed and the fractions are simplified. To get the simplified form of an expression, you must use the command `simplify`.

We notice that the answer can be provided in an equation editor.

1.2 Priority of operators

The four following operations are infix operators.

- + designates the addition,
- designates the subtraction,
- * designates the multiplication,
- / designates the division.

The raising to power is obtained with the key x^y and is written with `^` in the history.

To do the calculations:

- we do the calculations between the parentheses,
- we do the raising to powers,
- we do the multiplications and the divisions in the order from left to right,
- we do the additions and the subtractions in the order from left to right.

1.3 Implicit multiplication

In CAS, to do a multiplication, the sign `*` can be omitted when we do the multiplication of a number by a variable. It is allowed to write `2x` but you must write `a*b` to do the product of the variable `a` by `b`, because `ab` is also a name of variable.

We can write for example:

`2x+3i+4pi`

We cannot write:

`(2)x`, `(2)(x+y)`, `(2x+3)(x+y)`

We must write:

`2x` or `2*(x+y)` or `(2x+3)*(x+y)`

Warning! `x2` and `xy` designate the name of a variable and `f(x+1)` is the value of the function `f` at `x+1`.

1.4 Duration of a calculation: `time`

The evaluation of the duration of a long calculation is written in blue.

This evaluation of the duration is approximate, if you want more precision on the duration of your calculation, you must use the command `time` which returns the time taken for the evaluation in seconds.

`time` takes as argument a command and returns the time counted in seconds.

We enter:

```
time(factor(x^10-1))
```

We get in real mode:

```
0.0045
```

We enter:

```
time(factor(x^100-1))
```

We get in real mode:

```
0.0092
```

We enter:

```
time(factor(x^10-1))
```

We get in complex mode (set `i` in the CAS Settings):

```
0.272
```

We enter:

```
time(factor(x^100-1))
```

We get in complex mode:

```
29.794
```

1.5 Lists and sequences in the CAS

With the CAS, the lists (resp. the vectors) are put between brackets by `{ }` or by `[]` and the indices are put between brackets or between parentheses.

All indices start at 1.

For instance, we enter:

```
l:= [1, 2, 3, 4];
```

```
ll:= {1, 2, 3, 4}; l[2] or ll[2] returns 2
```

```
l(2) or ll(2) returns 2
```

With the CAS, the type sequence is also available, which is a series of objects. The indices of a sequence also start at 1.

For instance, we enter:

```
s:= 1, 2, 3, 4
```

```
s[2] or s(2) returns 2.
```

With this type sequence, the concatenation is easy.

To define the empty sequence, we enter:

```
s:=NULL;
```

If we did not check `Textbook` or `Algebraic` in the general setting (Shift HOME), we get:

```
NULL
```

Then, we enter:

```
s:=s,1,2
```

We get:

```
seq[1,2]
```

We enter:

```
s[1])
```

We get:

```
2
```

Whereas with the type list, to define the empty list, we enter:

```
l:=[];
```

Then, we enter:

```
l:=concat(l,[1,2])
```

We get:

```
[1,2]
```

We enter:

```
l[1])
```

We get:

```
2
```

To transform a list into a sequence; we use the operator `op`.

We enter:

```
op(l)
```

We get:

```
seq[1,2]
```

To transform a sequence into a list, it is enough to put the sequence between `[]`.

We enter:

```
[s])
```

We get:

```
[1,2]
```

1.6 Difference between expressions and functions

You must clearly distinguish expression and function.

An **expression** is a series of terms separated by the sign of an operation.

A term is a number, or a name of variable, or a product, or a pair of parentheses containing an expression.

Convention: the multiplication and the division have priority over the addition and the subtraction.

The sign $*$ is sometimes omitted in the writing, for example one writes: $2x$ instead of $2*x$.

A real **function** f defined on I part of \mathbb{R} is an application which at each number x of I maps an expression $f(x)$. The value of the function in one point x is then given by an expression.

Example

With HP Prime we enter in the CAS:

`xpr:=3*x+2`

We then define the expression `xpr`

We enter:

$$f(x) := 3*x + 2$$

We then define the function f

We enter:

`subst(xpr, x=1)` and we get 5

We enter:

`f(1)` and we get 5

We enter:

`plotfunc(3*x+2)` or, `plotfunc(xpr)` or, `plotfunc(f(x))`

we get one single graph which is the graph of the function f .

Note:

The plot of most of the commands starting by `plot` is working well from the CAS screen: then, it is better to use the geometry application to do the graphs related to these commands.

1.6.1 Defining a function by an expression

To define $f(x) = x \sin(x)$ we enter:

$$f(x) := x * \sin(x)$$

We enter:

$$f(1)$$

We get:

$$\sin(1)$$

but, take care, if we enter:

$$xpr := x * \sin(x)$$

then:

$$g(x) := xpr$$

this is not correct, because the variable x does not appear in `xpr`.

You must enter:

$$g := \text{unapply}(xpr, x)$$

We enter:

$$g(1)$$

We get:

$$\sin(1)$$

The command `unapply` returns a function which is defined by an expression and a variable: for example here `unapply(xpr, x)` designates the function:

$$x \rightarrow x * \sin(x)$$

1.6.2 Definition of a function of one or several variables

Definition of a function of \mathbb{R}^p in \mathbb{R}

To define the function $f(x) \rightarrow x * \sin(x)$:
we enter:

```
f(x) := x * sin(x)
```

Or we enter:

```
f := x -> x * sin(x)
```

We get:

```
(x) -> x * sin(x)
```

To define the function $f: (x, y) \rightarrow x * \sin(y)$
we enter:

```
f(x, y) := x * sin(y)
```

Or we enter:

```
f := (x, y) -> x * sin(y)
```

We get:

```
(x, y) -> x * sin(y)
```

Warning! What is after \rightarrow is not evaluated.

Definition of a function of \mathbb{R}^p in \mathbb{R}^q

To define the function $h: (x, y) \rightarrow (x * \cos(y), x * \sin(y))$:
we enter:

```
h(x, y) := (x * cos(y), x * sin(y))
```

To define the function $h: (x, y) \rightarrow [x * \cos(y), x * \sin(y)]$
we enter:

```
h(x, y) := [x * cos(y), x * sin(y)];
```

Or we enter:

```
h := (x, y) -> [x * cos(y), x * sin(y)];
```

Or we enter:

```
h(x, y) := { [x * cos(y), x * sin(y)] };;
```

Or we enter:

```
h := (x, y) -> return [x * cos(y), x * sin(y)];
```

Or we enter

```
h(x,y) := {return [x*cos(y), x*sin(y)];}
```

We get:

```
(x,y) -> {return ([x*cos(y), x*sin(y)]);}
```

Warning! What is after \rightarrow is not evaluated.

Definition of a function of \mathbb{R}^{p-1} in \mathbb{R}^q from a function of \mathbb{R}^p in \mathbb{R}^q

We define the function $f: (x,y) \rightarrow x * \sin(y)$, then we want to define the family of functions depending on the parameter t by $g(t)(y) := f(t,y)$.

Since what is after \rightarrow is not evaluated, we cannot define $g(t)$ by $g(t) := y \rightarrow f(t,y)$ and we must use the command `unapply`.

To define the functions $f(x,y) = x * \sin(y)$ and $g(t) = y \rightarrow f(t,y)$, we enter:

```
f(x,y) := x*sin(y); g(t) := unapply(f(t,y), y)
```

We get:

```
((x,y) -> x*sin(y), (t) -> unapply(f(t,y), y))
```

We enter:

```
g(2)
```

We get:

```
y -> 2 * sin(y)
```

We enter:

```
g(2)(1)
```

We get:

```
2 * sin(1)
```

We define the function $h: (x,y) \rightarrow (x * \cos(y), x * \sin(y))$, then we want to define the family of functions depending on the parameter t by $k(t)(y) := h(t,y)$.

Since what is after \rightarrow is not evaluated, we cannot define $k(t)$ by $k(t) := y \rightarrow h(x,y)$ and we must use the command `unapply`.

To define the function $h(x,y)$, we enter:

```
h(x,y) := (x*cos(y), x*sin(y))
```

To define the function $k(t)$, we enter:

```
k(t) := unapply(h(x,t), x)
```

We get:

```
(t) -> unapply(h(x,t), x)
```

We enter:

```
k(2)
```

We get:

```
(x) -> (x*cos(2), x*sin(2))
```

We enter:

```
k(2)(1)
```

We get:

```
(2*cos(1), 2*sin(1))
```

Or else we define the function $h : (x, y) \rightarrow [x \cdot \cos(y), x \cdot \sin(y)]$, then we want to define the family of functions depending on the parameter t by $k(t)(y) := h(t, y)$.

Since what is after \rightarrow is not evaluated, we cannot define $k(t)$ by $k(t) := y \rightarrow h(x, y)$ and we must use the command `unapply`.

To define the function $h(x, y)$, we enter:

```
h(x, y) := { [x*cos(y), x*sin(y)] }
```

To define the function $k(t)$, we enter:

```
k(t) := unapply(h(x, t), x)
```

We get:

```
(t) -> unapply(h(x, t), x)
```

We enter:

```
k(2)
```

We get:

```
(x) -> { [x*cos(2), x*sin(2)] ; }
```

We enter:

```
k(2)(1)
```

We get:

```
[2 * cos(1), 2 * sin(1)]
```

1.6.3 To define a function by two expressions: `when`

We enter: `g(x) := when (x > 0, x, -x)`

`g(-2)` returns 2

`g(2)` returns -2

1.6.4 Defining a function by n values: `PIECEWISE` `piecewise`

For instance, to define the function g which equals -1 if $x < -1$, 0 if $-1 \leq x \leq 1$ and 1 if $x > 1$, we enter:

```
g(x) := piecewise(x < -1, -1, x <= 1, 0, 1)
```

`piecewise` uses pairs condition/value where value is returned if condition is true, which implies that the previous conditions are false. If the number of arguments is odd, the last value is the default value (as in `switch`).

`piecewise` is the generalization of `when`.

To define the function f which equals -2 if $x < -2$, $3x + 4$ if $-2 \leq x < -1$, 1 if $-1 \leq x < 0$ and $x + 1$ if $x \geq 0$, we enter:

$$f(x) := \text{piecewise}(x < -2, -2, x < -1, 3x + 4, x < 0, 1, x + 1)$$

Then, we can do the graph of f by entering:

$$\text{plotfunc}(f(x))$$

1.6.5 Exercise on expressions

Here are 6 expressions formed from $T = 1 - x * 2 + x$ by adding parentheses:

$$A = (1 - x) * 2 + x$$

$$B = 1 - (x * 2) + x$$

$$C = 1 - x * (2 + x)$$

$$D = (1 - x * 2) + x$$

$$F = 1 - (x * 2 + x)$$

$$G = (1 - x) * (2 + x)$$

- 1) Is there one (or several) expression(s) equals to T ?
If so, why ?
- 2) Calculate the values of these expressions for $x = 1$ and for $x = -1$.
- 3) Among the expressions A, B, C, D, F, G :
 - Which are a sum of two terms?
 - Which are a difference of two terms?
 - Which are an algebraic sum of 3 terms?
 - Which are a product of two terms?
 - Which are equal?
- 4) Simplify the expressions A, B, C, D, F, G .
- 5) Write all the expressions formed from $S = 1 + \frac{x}{2*x}$ by adding parentheses.

Let us check with HP Prime. We enter:

$$T := 1 - x * 2 + x$$

$$A := (1 - x) * 2 + x$$

$$B := 1 - (x * 2) + x$$

$$C := 1 - x * (2 + x)$$

$$D := (1 - x * 2) + x$$

$$F := 1 - (x * 2 + x)$$

$$G := (1 - x) * (2 + x)$$

Then, we enter to check which expression equals T :

$A == T$, $B == T$, etc., ...

We find out that the answer to $A == T$ is 0 which means that the expression A is different from T .

We find out that the answer to $B == T$ is 1 which means that the expression B is identical to T , etc., ...

1.6.6 Exercise on the functions (to be followed)

- 1) Define 6 functions having for respective values the expressions A, B, C, D, F, G .
- 2) Plot the graphs of these functions and look at them on the same graphical representation.
- 3) Among these graphs there are lines and parabolae. Recognize the graph of each function. Let us check with HP Prime. To define the 6 functions, we enter:

$$a(x) := (1 - x) * 2 + x$$

$$b(x) := 1 - (x * 2) + x$$

$$c(x) := 1 - x * (2 + x)$$

$$d(x) := (1 - x * 2) + x$$

$$f(x) := 1 - (x^2 + x)$$

$$g(x) := (1-x) * (2+x)$$

Then, we enter to display the graphs:

```
plotfunc([a(x), b(x), c(x), d(x), f(x), g(x)])
```

We get only 5 curves of different colors.

We can enter successively:

```
plotfunc([a(x)]), plotfunc([a(x), b(x)]), etc., ...
```

Then, we notice that:

- the graph of a is the black line,
- the graph of b is the red line,
- the graph of c is the green parabola,
- the graph of d is the yellow line which overlaps the red line,
- the graph of f is the blue line, and
- the graph of g is the green parabola.

Chapter 2 Menu Algebra

2.1 Simplifying an expression: `simplify`

`simplify` simplifies an expression in an automatic way.

We enter:

```
simplify(x^5+1/((x-1)^4)+1/((x+1)^4)+1/((x+i)^4)+1/((x-i)^4))
```

We get:

$$(x^9 - x^5 + x^3) / (x^4 - 1)$$

We enter:

```
simplify(3-54*sqrt(1/162))
```

We get:

$$-3\sqrt{2} + 3$$

Warning! `simplify` is more efficient when to simplify trigonometric expressions when being in radian: for this reason, we check `radian` in the CAS configuration.

We enter:

```
simplify((sin(3*x)+sin(7*x))/sin(5*x))
```

We get:

$$4 * (\cos(x))^2 - 2$$

2.2 Factorizing a polynomial on the integers: `collect`

`collect` takes as parameter a polynomial or a list of polynomials and eventually `sqrt(n)`.

`collect` factors the polynomial (or the polynomials of the list) on the integers when the coefficients of the polynomial are integers or one $\mathbb{Q}(\sqrt{n})$, if the coefficients of the polynomial are in $\mathbb{Q}(\sqrt{n})$ or if `sqrt(n)` is the second argument.

We enter:

```
collect(x^3-2*x^2+1)
```

We get:

$$(x-1) * (x^2-x-1)$$

We enter:

```
collect(x^3-2*x^2+1, sqrt(5))
```

We get:

$$(x + (-\sqrt{5}) - 1) / 2 * (x - 1) * (x + (\sqrt{5}) - 1) / 2$$

See also `factor` depending on we have checked or not $\sqrt{\quad}$ in the CAS configuration.

2.3 Regrouping and simplifying: `regroup`

`regroup` takes as parameter an expression.

`regroup` does the obvious simplifications on an expression by grouping terms.

We enter:

```
regroup(x+3*x+5*4/x)
```

We get:

$$20/x+4*x$$

2.4 Expanding and simplifying: `normal`

`normal` takes as parameter an expression.

`normal` returns the developed and simplified expression.

We enter:

```
normal(x+3*x+5*4/x)
```

We get:

$$(4*x^2+20)/x$$

We enter:

```
normal((x-1)*(x+1))
```

We get:

$$x^2-1$$

Warning! `normal` is less efficient than `simplify` and, sometimes, it might be necessary to invoke several times the command `normal`.

We enter:

```
normal(3-54*sqrt(1/162))
```

We get:

$$(-9*\sqrt{2}+9)/3$$

We enter:

```
normal((-9*sqrt(2)+9)/3)
```

We get:

$$-(3*\sqrt{2})+3$$

2.5 Expanding an expression: `expand`

`expand` applies on an expression the distributive of the multiplication over the addition.

We enter:

```
expand((x+1)*(x+2))
```

We get:

```
x^2+3*x+2
```

We enter:

```
expand((a+b)^5)
```

We get:

```
5*a^4*b+10*a^3*b^2+10*a^2*b^3+5*a*b^4+b^5+a^5
```

2.6 Multiply by the conjugate quantity: `mult_conjugate`

`mult_conjugate` takes as argument an expression with a denominator or a numerator comprising square roots:

- `mult_conjugate` takes as argument an expression with a denominator comprising square roots.
`mult_conjugate` multiplies the numerator and the denominator of this expression by the conjugate quantity of the denominator.
- `mult_conjugate` takes as argument an expression with a denominator not comprising square roots.
`mult_conjugate` multiplies the numerator and the denominator of this expression by the conjugate quantity of the numerator.

We enter:

```
mult_conjugate((2+sqrt(2))/(2+sqrt(3)))
```

We get:

```
(2+sqrt(2))*(2-sqrt(3))/((2+sqrt(3))*(2-sqrt(3)))
```

We enter:

```
mult_conjugate((2+sqrt(2))/(sqrt(2)+sqrt(3)))
```

We get:

```
(2+sqrt(2))*(-sqrt(2)+sqrt(3))/
((sqrt(2)+sqrt(3))*(-sqrt(2)+sqrt(3)))
```

We enter:

```
mult_conjugate((2+sqrt(2))/2)
```

We get:

```
(2+sqrt(2))*(2-sqrt(2))/(2*(2-sqrt(2)))
```

2.7 Factorizing an expression: `factor`

We enter:

```
factor(x^6-1)
```

We get in real mode:

$$(x-1) * (x+1) * (x^2-x+1) * (x^2+x+1)$$

We enter:

$$\text{factor}(x^6+1)$$

We get in real mode:

$$(x^2+1) * (x^4-x^2+1)$$

We get in complex mode with $\sqrt{\quad}$ not checked:

$$(x+i) * (x-i) * (x^2+(i)*x-1) * (x^2+(-i)*x-1)$$

We get in complex mode with $\sqrt{\quad}$ checked:

$$(x+i) * (x-i) * (x+(-\sqrt{3})-i)/2) * (x+(-\sqrt{3})+i)/2) * (x+(\sqrt{3}-i)/2) * (x+(\sqrt{3}+i)/2)$$

We enter:

$$\text{factor}(x^6+1, \sqrt{3})$$

We get in complex mode with $\sqrt{\quad}$ checked or not:

$$(x+i) * (x-i) * (x+(-\sqrt{3})-i)/2) * (x+(-\sqrt{3})+i)/2) * (x+(\sqrt{3}-i)/2) * (x+(\sqrt{3}+i)/2)$$

We enter:

$$\text{factor}(x^3-2*x^2+1)$$

We get, if we have not checked $\sqrt{\quad}$ in the CAS configuration:

$$(x-1) * (x^2-x-1)$$

We enter:

$$\text{factor}(x^3-2*x^2+1)$$

We get, if we have checked $\sqrt{\quad}$ in the CAS configuration:

$$(x+(-\sqrt{5})-1)/2) * (x-1) * (x+(\sqrt{5}-1)/2)$$

We enter:

$$\text{factor}(\text{expexpand}(\exp(5*x)) - \exp(x))$$

We get in complex mode:

$$\exp(x) * (-1+\exp(x)) * (1+\exp(x)) * (i+\exp(x)) * (-i+\exp(x))$$

2.8 Factorization without square factor: `sqrfree`

`sqrfree` takes as parameter a polynomial.

`sqrfree` factors this polynomial by grouping the terms having the same exponent.

We enter:

$$\text{sqrfree}((x^2-1) * (x-1) * (x+2))$$

We get:

$$(x^2+3x+2) * (x-1)^2$$

We enter:

$$\text{sqrfree}((x^2-1)^2 * (x-1) * (x+2)^2)$$

We get:

$$(x^2+3x+2) * (x-1)^3$$

2.9 Factorization in \mathbb{C} : `cFactor` `cfactor`

`cFactor` or `cfactor` takes as parameter the expression we want to factor in the complex field without being in complex mode.

When there are more than two variables, the factorization is performed on Gauss integers.

Examples

1. Factorizing in \mathbb{C} :

$$x^4 - 1$$

We enter:

$$\text{cFactor}(x^4-1)$$

We get:

$$((x+i) * ((-i) * x+1) * ((-i) * x+i) * (x+1))$$

2. Factorizing in \mathbb{C} :

$$x^4 + 1$$

We enter:

$$\text{cfactor}(x^4+1)$$

We get:

$$(x^2+i) * (x^2-i)$$

Then, we enter:

$$\text{cfactor}(\text{sqrt}(2) * (x^2+i)) * \text{cFactor}(\text{sqrt}(2) * (x^2-i))$$

We get:

$$\text{sqrt}(2) * 1/2 * (\text{sqrt}(2) * x+1-i) * (\text{sqrt}(2) * x-1+i) * \text{sqrt}(2) * 1/2 * (\text{sqrt}(2) * x+1+i) * (\text{sqrt}(2) * x-1-i)$$

but if we enter,:

$$\text{cfactor}(\text{sqrt}(2) * (x^4+1))$$

We get:

$$\text{sqrt}(2) * (x^2+\text{sqrt}(2) * x+1) * (x^2+(-\text{sqrt}(2))) * x+1)$$

2.10 Substituting a variable by a value: `subst`

`subst` takes two or three arguments: an expression depending on a parameter and an equality (parameter=substitution value) or an expression depending on a parameter, the parameter and the substitution value.

`subst` does the requested substitution in the expression provided that the parameter is not assigned because `subst` first evaluates the expression and then replaces the parameter (if it has been assigned) by its value without taking into account the substitution value supplied by the second parameter.

We enter:

```
subst (a^2+1, a=3)
```

We get:

```
10
```

We enter:

```
a:=2; subst (a^2+1, a=3)
```

We get:

```
(2, 5)
```

We enter:

```
a:=2; purge (a) ; subst (a^2+1, a=3)
```

We get:

```
(2, 2, 10)
```

2.11 Fractions

2.11.1 Decompose into simple elements: `partfrac`

`partfrac` takes as argument a rational fraction.

`partfrac` returns its decomposition into simple elements.

We enter:

```
partfrac (x^5+x^3/ (x^4-1) )
```

We get:

```
x^5+1/ ((x-1) *4)+1/ ((x+1) *4)+x/ ((x^2+1) *2)
```

We enter:

```
partfrac (x^5+x^3/ (x^4-1) )
```

We get:

```
x^5+1/ ((x-1) *4)+1/ ((x+1) *4)+1/ ((x+i) *4)+1/ ((x-i) *4)
```

2.11.2 Decomposition in simple elements on \mathbb{C} : `cpartfrac`

`cpartfrac` takes as argument a rational fraction.

`cpartfrac` returns its decomposition into simple elements on \mathbb{C} be it in real mode or complex mode.

Example:

Decompose into simple elements the rational fraction:

$$\frac{x^5 - 2x^3 + 1}{x^4 - 2x^3 + 2x^2 - 2x + 1}$$

We use the command `cpartfrac`.

We enter:

```
cpartfrac((x^5-2*x^3+1)/(x^4-2*x^3+2*x^2-2*x+1))
```

We get in real mode or in complex mode:

$$x+2+(-1+2*i)/((2-2*i)*(i*x+1))+1/(2*(-x+1))+(-1-2*i)/((2-2*i)*(x+i))$$

2.11.3 Put to common denominator: `comDenom`

`comDenom` takes as parameter a sum of rational fractions.

`comDenom` returns this sum as a rational fraction, that is to say returns this sum once the rational fractions it is composed with have been put to common denominator.

We enter:

```
comDenom(x-1/(x-1)-1/(x^2-1))
```

We get:

$$(x^3-2*x-2)/(x^2-1)$$

2.11.4 Integer part and fractional part: `propfrac`

`propfrac` takes as argument a rational fraction.

`propfrac` returns this rational fraction written in a way that brings out its integer part.

`propfrac(A(x)/B(x))` writes the rational fraction $\frac{A(x)}{B(x)}$

after simplification

in the form of:

$$Q(x) + \frac{R(x)}{B(x)}$$

with $R(x) = 0$ or $0 \leq \text{degree}(R(x)) < \text{degree}(B(x))$.

We enter:

```
propfrac((5*x+3)*(x-1)/(x+2))
```

We get:

$$5*x-12+21/(x+2)$$

2.12 Extract

2.12.1 Numerator of a fraction after simplification: `numer`

`numer` takes as argument a fraction or a rational fraction and returns the numerator of this simplified fraction.

We enter:

```
numer(42/12)
```

We get:

```
7
```

We enter:

```
numer(x^5+x^3/(x^4-1))
```

We get:

```
x^9-x^5+x^3
```

2.12.2 Denominator of a fraction after simplification: `ofnom`

`denom` takes as argument a fraction or a rational fraction and returns the denominator of this simplified fraction.

We enter:

```
denom(42/12)
```

We get:

```
2
```

We enter:

```
denom(x^5+1/((x-1)^4)+1/((x+1)^4)+x/((x^2+1)^2))
```

We get:

```
x^4-1
```

2.12.3 Numerator and denominator: `f2nd`

`f2nd` takes as argument a fraction or a rational fraction and returns the list formed by the numerator and the denominator of this simplified fraction.

We enter:

```
f2nd(42/12)
```

We get:

```
[7,2]
```

We enter:

```
f2nd((x^2-1)/(x-1))
```

We get:

```
[x+1, 1]
```

We enter:

```
f2nd((x^2+2*x+1)/(x^2-1))
```

We get:

```
[x+1, x-1]
```

2.12.4 Get the left member of an equation: `left`

`left` takes as parameter an equation or an interval.

`left` returns the left member of the equation or the left boundary of the interval.

We enter:

```
left(a=3)
```

We get:

```
a
```

We enter:

```
left(a..2*a+1)
```

We get:

```
a
```

2.12.5 Get the right member of an equation: `right`

`right` takes as parameter an equation or an interval.

`right` returns the right member of the equation or the right boundary of the interval.

We enter:

```
right(a=3)
```

We get:

```
3
```

We enter:

```
right(a..2*a+1)
```

We get:

```
2*a+1
```

2.12.6 Center of an interval: `interval2center`

`interval2center` takes as argument an interval or a list of intervals.

`interval2center` returns the center of the interval or the list of centers of these intervals.

We enter:

```
interval2center(3..5)
```

We get:

```
4
```

We enter:

```
interval2center([2..4,4..6,6..10])
```

We get:

```
[3,5,8]
```

2.12.7 Signature of a permutation: `signature`

`signature` takes as argument a permutation.

`signature` returns the signature of the permutation supplied as argument.

The signature of a permutation equals:

- 1 if it can be decomposed into an even product of transpositions,
- -1 if it can be decomposed into an odd product of transpositions.

The signature of a cycle of order k is: $(-1)^{k+1}$.

We enter:

```
signature(3,4,5,2,1)
```

We get:

```
-1
```

Indeed, this permutation is decomposed into the cycles:

(1,3,5) and (2,4) that is to say in 3 transpositions:

(1,3), (3,5) and (2,4).

Chapter 3 Menu Calculus

3.1 Definition of a function: := and \rightarrow (Sto ►)

To define for example the function f which at x maps $x^3 + \ln(x)$, we enter:

```
f:=x-> x^3+ln(x)
```

or we enter:

```
f(x) := x^3+ln(x)
```

3.2 Maximum and minimum of an expression: fMax fMin

f_{Max} and f_{Min} have as argument: an expression of one variable and the name of this variable (by default x).

f_{Max} returns the abscissa of the main solution of the maximum of the expression.

f_{Min} returns the abscissa of the main solution of the minimum of the expression.

We enter:

```
fMax(sin(x), x)
```

Or we enter:

```
fMax(sin(x))
```

Or we enter:

```
fMax(sin(y), y)
```

We get:

```
pi/2
```

We enter:

```
fMin(sin(x), x)
```

Or we enter:

```
fMin(sin(x))
```

Or we enter:

```
fMin(sin(y), y)
```

We get:

```
-pi/2
```

We enter:

```
fMin(sin(x)^2, x)
```

We get:

0

3.3 Differentiate

3.3.1 Derivative function of a function: `function_diff`

`function_diff` takes as argument a function.

`function_diff` returns the derivative function of this function.

We enter:

```
function_diff(sin)
```

We get:

```
(` x`) -> cos(` x`)
```

We enter:

```
function_diff(sin)(x)
```

We get:

```
cos(x)
```

We enter:

```
f(x) := x^2 + x*cos(x)
```

```
function_diff(f)
```

We get:

```
(` x`) -> 2*` x` + cos(` x`) + ` x`*(-(sin(` x`)))
```

We enter:

```
function_diff(f)(x)
```

We get:

```
cos(x) + x*(-(sin(x))) + 2*x
```

3.3.2 Differentiate : `∂ diff ' ```

`diff` or `'` returns the derivative of an expression or of a function of a variable and returns also the partial derivatives of an expression of several variables.

We can also use the key showing letter `∂` and use:

$\frac{\partial \square}{\partial \square}$

$\frac{\partial \square}{\partial \square}$

and if we have checked the display configuration in Textbook mode (cf. Settings of HOME), it is enough to fill in the two \square .

– Derivative of an expression of one variable

We enter:

$$\text{diff}(x^3+\ln(x))$$

Or we enter (' is obtained with Shift- () (' ') by clearing a '):

$$(x^3+\ln(x))'$$

We get the expression of the derivative of $x^3+\ln(x)$ according to x :

$$3*x^2+1/x$$

We enter:

$$\text{diff}(y^3+\ln(y), y)$$

Or we enter (' is obtained with Shift- () (' ') by clearing a '):

$$(y^3+\ln(y), y)'$$

We get the expression of the derivative of $y^3+\ln(y)$ according to y :

$$3*y^2+1/y$$

– Second derivative (or Nth) of an expression of one variable

We enter:

$$\text{diff}(\text{diff}(x^3+\ln(x)))$$

Or we enter (' ' is obtained with Shift- ()):

$$(x^3+\ln(x))''$$

We get the expression of the second derivative of $x^3+\ln(x)$ according to x :

$$3*2*x-1/x^2$$

We enter:

$$\text{diff}(\text{diff}(\text{diff}(\text{diff}(x^3+\ln(x)))))$$

Or we enter (' ' ' ' is obtained with Shift- () Shift- ()):

$$(x^3+\ln(x))''''$$

We get the expression of the 4th derivative of $x^3+\ln(x)$ according to x :

$$-2*3/x^4$$

– Partial derivative of an expression of several variables.

We enter:

$$\text{diff}(x*y*z, \{x, y, z\})$$

We get the expression of the partial derivative according to x , according to y and according to z , of $x*y*z$:

$$\{y*z, x*z, x*y\}$$

– Derivative of a function

To define the function f , we enter:

$$f(x) := x^3 + \ln(x)$$

We get:

$$(x) \rightarrow x^3 + \ln(x)$$

We enter:

$$g := \text{diff}(f)$$

Or we enter (' is obtained with `Shift-()` (' ') by clearing a '):

$$g := f'$$

We get the function g which is the derivative function of f :

$$x \rightarrow 3x^2 + 1/x$$

– Second derivative (or Nth) of a function

To define the function f , we enter:

$$f(x) := x^3 + \ln(x)$$

We get

$$(x) \rightarrow x^3 + \ln(x)$$

We enter:

$$h := \text{diff}(\text{diff}(f))$$

Or we enter (' ' is obtained with `Shift-()`):

$$h := f''$$

We get the function h which is the second derivative function of f :

$$x \rightarrow 3 \cdot 2 \cdot x - 1/x^2$$

3.3.3 Approximate calculation of the derivative number: `nDeriv`

`nDeriv` takes as arguments: an expression X_{pr} , the name of the variable of this expression (by default x), and h (by default $h=0.001$).

`nDeriv(f(x), x, h)` calculates, in an approximate way, the value of the derivative of the expression $f(x)$ at point x and returns:

$$(f(x+h) - f(x-h)) / 2 \cdot h.$$

We enter:

$$\text{nDeriv}(x^2, x)$$

We get:

$$((x+0.001)^2 - (x-0.001)^2) * 500.0$$

We enter:

```
subst(nDeriv(x^2,x),x=1)
```

We get:

2

3.4 Integration

3.4.1 Primitive: `int`

`int` allows to calculate a primitive of an expression or of a function or a definite integral.

We can also use the key showing letter `C` and use:

$$\int_{\square}^{\square} \square \partial \square$$

and if we have chosen the `Textbook` mode as display configuration (cf. Settings of HOME), it is enough to fill in the boxes.

– Primitive of an expression

We enter:

```
int(x^3+ln(x))
```

We get a primitive of $x^3 + \ln(x)$ according to x :

$$x \ln(x) - x + x^4/4$$

We enter:

```
int(y^3+ln(y),y)
```

We get a primitive of $y^3 + \ln(y)$ according to y :

$$y \ln(y) - y + y^4/4$$

– Primitive of a function

To define the function f , we enter:

```
f(x) := x^3+ln(x)
```

We get:

```
(x) -> x^3+ln(x)
```

We enter:

```
g:=int(f)
```

We get the function g which is a primitive of f :

```
(x) -> x*ln(x) - x + x^4/4
```

– Definite integral

We enter:

```
int(x^3+ln(x),x,1,2)
```

Or we enter:

```
int (x^3+ln(x), x=1..2)
```

Or we enter:

```
int (y^3+ln(y), y=1..2)
```

Or we enter when $f(x) := x^3 + \ln(x)$ and $g := \text{int}(f)$:

```
preval (g(x), 1, 2)
```

We get the value of $\int_1^2 x^3 + \ln(x) dx$:

$$2 * \ln(2) - (-3/4)$$

3.4.2 Evaluate a primitive: `preval`

`preval` has three parameters: an expression $F(x)$ depending on the variable x , and two expressions a and b .

`preval` does $F(b) - F(a)$.

`preval` is useful to calculate a definite integral by a primitive: we calculate a primitive, then one evaluates this primitive between the boundaries of the integral.

We enter:

```
preval (x^2+x, 2, 3)
```

We get:

6

3.4.3 Approximate calculation of integrals with the Romberg method: `romberg`

`romberg` takes as arguments: an expression X_{pr} , the name of the variable of this expression (by default x), and two values a, b .

`romberg(Xpr, x, a, b)` returns the approximate integral $\int_a^b X_{pr} dx$ by the Romberg method.

We enter:

```
romberg (exp (x^2), x, 0, 1)
```

We get:

1.46265174591

3.5 Limites: `limit`

Provided we are in radians, `limit` allows to calculate the limit of an expression in a finite point (or infinite).

By means of an additional parameter, we can tell if we look for a limit by greater values or by lower values (1 to tell " by greater values " and -1 to tell " by lower values ").

`limit` takes three or four arguments:

an expression, the name of the variable (for example x), the limit point (for example a) and an optional argument which tells if the limit is unidirectional or bidirectional (by default 0). This argument equals -1 as left limit ($x < a$) or equals 1 as right limit ($x > a$) or equals 0 for a limit.

The optional argument is then used when we want to calculate right limit (+1) or a left limit(-1).

`limit` returns the requested limit (if any).

When we use `limit` from the menus and if we have chosen the `Textbook` mode as display configuration (cf HOME Settings) it is displayed on the entry line: `lim_{□→□}(□)` and it is enough to fill in the □.

For example, we get `limit` in the menu `CAS → Calculus → Limit`:

`lim_{x→0} abs(x)/x`

We get in the history:

```
limit(abs(x)/x,x,0,1)
```

and the answer:

```
1
```

We enter:

```
limit(sin(x)+ln(x))/x,x,1
```

We get:

```
sin(1)
```

We enter:

```
limit(1/x,x,0)
```

We get:

```
infinity
```

this means that `abs(1/x)` tends to $+\infty$ when `x` tends to 0.

We enter:

```
limit(1/x,x,0,1)
```

We get:

```
+infinity
```

We enter:

```
limit(1/x,x,0,-1)
```

We get:

```
-infinity
```

Note:

if we enter `limit((-1)^n,n=inf)`, then the CAS returns `bounded_function(5)` which means that the function is bounded but has no limit at the infinite.

We enter:

```
limit(sin(x),x,inf)
```

We get:

```
bounded_function(2)
```

We enter:

$$\text{limit}(\cos(x), x, \text{inf})$$

We get:

$$\text{bounded_function}(7)$$

Exercises:

- Find for $n > 2$, the limit when x tends to 0 of:

$$\frac{n \tan(x) - \tan(nx)}{\sin(nx) - n \sin(x)}$$

We enter:

$$\text{limit}((n*\tan(x)-\tan(n*x))/(sin(n*x)-n*\sin(x)), x=0)$$

We get:

$$2$$

- Find the limit when x tends to $+\infty$ of:

$$\sqrt{x + \sqrt{x + \sqrt{x}} - \sqrt{x}}$$

We enter:

$$\text{limit}(\text{sqrt}(x+\text{sqrt}(x+\text{sqrt}(x)))-\text{sqrt}(x), x=+\text{infinity})$$

We get:

$$1/2$$

- Find the limit when x tends to 0 of:

$$\frac{\sqrt{1+x+\frac{x^2}{2}} - e^{\frac{x}{2}}}{(1-\cos(x))\sin(x)}$$

We enter:

$$\text{limit}((\text{sqrt}(1+x+x^2/2)-\exp(x/2))/((1-\cos(x))*\sin(x)), x, 0)$$

We get:

$$-1/6$$

Sometimes, to calculate limits more easily, it can be judicious to quote the first argument. By example, we enter:

$$\text{limit}('(2*x-1)*\exp(1/(x-1))', x=+\text{infinity})$$

We notice that we have quoted here the first argument so that it is not evaluated, that is to say so that it is not simplified.

We get:

$$+(\text{infinity})$$

3.6 Limit and integral

We give here some examples:

- Determinate the limit when a tends to the infinite of:

$$\int_2^a \frac{1}{x^2} dx$$

We enter:

$$\text{limit}(\text{int}(1/(x^2), x, 2, a), a, +(\text{infinity}))$$

We get (check that a is formal otherwise do `purge(a)`):

$$\text{Indeed } \int_2^a \frac{1}{x^2} dx = \frac{1}{2} - \frac{1}{a}$$

Thus $\int_2^a \frac{1}{x^2} dx$ tends to $\frac{1}{2}$ when a tends to the infinite.

- **Determinate the limit when a tends to the infinite of:**

$$\int_2^a \left(\frac{x}{x^2-1} + \ln\left(\frac{x+1}{x-1}\right) \right) dx$$

We enter:

$$\text{limit}(\text{int}(x/(x^2-1)+\ln((x+1)/(x-1)), x, 2, a), a, +(\text{infinity}))$$

We get (check that a is formal otherwise do `purge(a)`):

$$+(\text{infinity})$$

Indeed:

$$\int_2^a \frac{x}{x^2-1} dx = \frac{1}{2} (\ln(a^2-1) - \ln(3))$$

and

$$\int_2^a \ln\left(\frac{x+1}{x-1}\right) dx = \ln(a+1) + \ln(a-1) + a * \ln\left(\frac{a+1}{a-1}\right) - 3\ln(3)$$

So when a tends to $+\infty$ the integral tends to $+\infty$.

- **Determinate the limit when a tends to 0 of:**

$$\int_a^{3a} \frac{\cos(x)}{x} dx$$

$$\text{limit}(\text{int}(\cos(x)/x, x, a, 3a), a, 0)$$

We get (check that a is formal otherwise do `purge(a)`):

$$\ln(3)$$

To find this limit, we bound $\frac{\cos(x)}{x}$ because we do not know the primitive of $\frac{\cos(x)}{x}$.

Knowing that:

$$1 - 2\sin^2\frac{x}{2} = \cos(x) \leq 1 \text{ and } \sin^2\frac{x}{2} \leq \frac{x^2}{4} \text{ thus, } 1 - \frac{x^2}{2} = \cos(x) \leq 1 \text{ and}$$

$$\frac{1}{x} - \frac{x}{2} \leq \frac{\cos(x)}{x} \leq \frac{1}{x}$$

Thus:

$$\int_a^{3a} \left(\frac{1}{x} - \frac{x}{2} \right) dx \leq \int_a^{3a} \frac{\cos(x)}{x} dx \leq \int_a^{3a} \frac{1}{x} dx$$

$$\ln(3) - \frac{9a^2}{4} + \frac{a^2}{4} \leq \int_a^{3a} \frac{\cos(x)}{x} dx \leq \ln(3)$$

Thus $\int_a^{3a} \frac{\cos(x)}{x} dx$ tends to $\ln(3)$ when a tends to 0.

3.7 Series: series

`series` allows to do the series expansion of an expression of the variable `Var` in `Var=0` (by default in `x=0`) at a supplied order (by default 5).

We enter:

```
series(tan(x))
```

We get:

$$x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + x^6 \text{order_size}(x)$$

`order_size` designates a function so that, regardless than `r` positive: $x^r \text{order_size}(x)$ tends to zero when x tends to zero.

So, when we have in the answer $(x-a)^n \text{order_size}(x-a)$, this means that we have a series expansion at order $n - 1$ in the vicinity of $x=a$.

We enter:

```
series(tan(x), x=0, 9)
```

We get:

$$x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + \frac{62}{2835}x^9 + x^{10} \text{order_size}(x)$$

We enter:

```
series(atan(x), x=+infinity, 5)
```

We get:

$$\frac{1}{2}\pi - \frac{1}{x} + \frac{1}{3}\left(\frac{1}{x}\right)^3 - \frac{1}{5}\left(\frac{1}{x}\right)^5 + \left(\frac{1}{x}\right)^6 \text{order_size}\left(\frac{1}{x}\right)$$

here $\left(\frac{1}{x}\right)^6 \text{order_size}\left(\frac{1}{x}\right)$ means that we have a series expansion at order $6 - 1 = 5$ in the vicinity of $\frac{1}{x} = 0$ i.e. in the vicinity of $+\infty$

We enter:

```
series(atan(x), x=-infinity, 5)
```

We get:

$$-\frac{1}{2}\pi - \frac{1}{x} - \frac{1}{3}\left(-\frac{1}{x}\right)^3 + \frac{1}{5}\left(-\frac{1}{x}\right)^5 + \left(-\frac{1}{x}\right)^6 \text{order_size}\left(-\frac{1}{x}\right)$$

here $\left(-\frac{1}{x}\right)^6 \text{order_size}\left(-\frac{1}{x}\right)$ means that we have a series expansion at order $6 - 1 = 5$ in the vicinity of $\frac{1}{x} = 0$ i.e. in the vicinity of $-\infty$

3.8 Residue of an expression in a point: residue

`residue` takes as argument an expression depending on a variable, the name of this variable and a complex `a` or else an expression depending on a variable and the equality: `name_of_variable=a`.

`residue` returns the residue of this expression at point `a`.

We enter:

```
residue(cos(x)/x^3, x, 0)
```

Or we enter:

```
residue(cos(x)/x^3,x=0)
```

We get:

$$(-1)/2$$

We enter:

```
int(exp(i*t)/(2*exp(i*t)-1),t=0..2*pi)
```

We get:

Searching \int of $1/(2*t-1)$ where t is on the unit circle, using residues

$$(2*\pi)/2$$

We enter:

```
int(exp(2*i*t)/(2*exp(i*t)-1)^2,t=0..2*pi)
```

We get:

Searching \int of $t/(4*t^2-4*t+1)$ where t is on the unit circle, using residues

$$(2*\pi)/4$$

3.9 Pade approximation: pade

`pade` has four arguments:

- an expression,
- the name of the used variable,
- an integer n or a polynomial N ,
- an integer p .

`pade` returns a rational fraction P/Q (with the order of $P < p$) which has, in the vicinity of 0, the same Taylor series expansion at order n as the expression, or which equals the expression modulus x^{n+1} (resp. modulus N).

We enter:

```
pade(exp(x),x,5,3)
```

Or we enter:

```
pade(exp(x),x,x^6,3)
```

We get:

$$(3*x^2+24*x+60)/(-x^3+9*x^2-36*x+60)$$

We check by entering:

```
taylor((3*x^2+24*x+60)/(-x^3+9*x^2-36*x+60))
```

We get:

$$1+x+1/2*x^2+1/6*x^3+1/24*x^4+1/120*x^5+x^6*\text{order_size}(x)$$

We recognize the Taylor series expansion at order 5 of $\exp(x)$ in the vicinity of 0.

We enter:

```
pade((x^15+x+1)/(x^12+1), x, 12, 3)
```

Or we enter:

```
pade((x^15+x+1)/(x^12+1), x, x^13, 3)
```

We get:

$$x+1$$

We enter:

```
pade((x^15+x+1)/(x^12+1), x, 14, 4)
```

Or we enter:

```
pade((x^15+x+1)/(x^12+1), x, x^15, 4)
```

We get:

$$(-2*x^3-1)/(-x^11+x^10-x^9+x^8-x^7+x^6-x^5+x^4-x^3-x^2+x-1)$$

We check by entering:

```
series(ans(), x=0, 15)
```

We get:

$$1+x-x^{12}-x^{13}+2x^{15}+x^{16}*\text{order_size}(x)$$

then by entering:

```
series((x^15+x+1)/(x^12+1), x=0, 15)
```

We get:

$$1+x-x^{12}-x^{13}+x^{15}+x^{16}*\text{order_size}(x)$$

Both expressions have the same Taylor series expansion at order 14 in the vicinity of 0.

3.10 Indexed finite and infinite sum and discrete primitive: `sum`

`sum` does the finite summation and infinite summation or returns the discrete primitive of an expression.

`sum` also does the sum of the elements of a list (see 16.8).

- **sum of a list or of a sequence**

We enter:

```
l:= [1, 2, 3, 4, 5, 6, 7, 8]
```

Or we enter:

```
l:=1, 2, 3, 4, 5, 6, 7, 8
```

Then, we enter:

```
sum(1)
```

We get the sum $1+2+\dots+8=8*9/2$:

```
36
```

– indexed finite sum

We enter:

```
sum(k, k=1..8)
```

Or we enter:

```
sum(k, k, 1, 8)
```

We get:

```
36
```

– indexed infinite sum

We enter:

```
sum(1/2^k, k, 0, inf)
```

We get:

```
2
```

– discrete primitive of an expression

The discrete primitive of the expression $f(x)$ is the function G which makes:

$$G(x+1) - G(x) = f(x)$$

Then `sum` has two arguments: an expression of one variable (for example $f(x)$) and the variable (for example x):

We enter:

```
sum(x, x)
```

We get:

```
(x^2-x) / 2
```

Thus:

$$4 + 5 + \dots + 19 = G(20) - G(4) = 190 - 6 = 184$$

Check that: `sum(k,k=4..19)` returns 184

We enter:

```
sum(1/(x*(x+1)), x)
```

We get:

```
-1/x
```

Thus:

$$1/(1*2) + 1/(2*3) + \dots + 1/(9*10) = -1/10 + 1 = 9/10$$

We check that: `sum(seq(1/(k*(k+1)), k, 1, 9))` returns 9/10

3.11 Differential

3.11.1 Rotational curl: `curl`

`curl` has two parameters: an expression F depending on three real variables and a vector dimension 3 storing the name of these variables.

`curl` designates the rotational curl of F .

We enter:

```
curl([x*z, -y^2, 2*x^y], [x, y, z])
```

We get:

```
[2*ln(x)*x^y, x-2*y*x^(y-1), 0]
```

Indeed:

```
diff(2*x^y, y) - diff(-y^2, z) returns 2*ln(x)*x^y
diff(x*z, z) - diff(2*x^y, x) returns x-2*y*x^(y-1)
diff(-y^2, x) - diff(x*z, y) returns 0
```

We enter:

```
curl([x*y*z, -y^2, 2*x], [x, y, z])
```

We get:

```
[0, x*y-2, -x*z]
```

3.11.2 Divergence: `divergence`

`divergence` has two parameters: an expression F depending on n real variables and a dimension n vector storing the name of these variables.

`divergence` designates the divergence of F .

We enter:

```
divergence([x^2+y, x+z+y, z^3+x^2], [x, y, z])
```

We get:

```
2*x+3*z^2+1
```

Indeed:

```
diff(x^2+y, x) + diff(x+z+y, y) + diff(z^3+x^2, z) returns 2*x+1+3*z^2
```

3.11.3 Gradient: `grad`

`grad` has two parameters: an expression F depending on n real variables and a dimension n vector storing the name of these variables.

`derive` returns the gradient of F .

We enter:

```
grad(2*x^2*y-x*z^3, [x, y, z])
```

We get:

```
[2*2*x*y-z^3, 2*x^2, -x*3*z^2]
```

Indeed:

`diff(2*x^2*y-x*z^3, x)` returns $4*x*y-z^3$

`diff(2*x^2*y-x*z^3, y)` returns $2*x^2$

`diff(2*x^2*y-x*z^3, z)` returns $-3*x*z^2$

3.11.4 Hessian matrix: `hessian`

`hessian` has two parameters: an expression F depending on n real variables and a dimension n vector storing the name of these variables.

`hessian` returns the Hessian of F which is the matrix of derivatives of degree two namely `diff(diff(F, [x, y, z]), [x, y, z])`.

We enter:

```
hessian(2*x^2*y-x*z, [x, y, z])
```

We get:

```
[[4*y, 4*x, -1], [2*2*x, 0, 0], [-1, 0, 0]]
```

Indeed:

`diff(diff(2*x^2*y-x*z, x), [x, y, z])` returns $[4*y, 4*x, -1]$

`diff(diff(2*x^2*y-x*z, y), [x, y, z])` returns $[4*x, 0, 0]$

`diff(diff(2*x^2*y-x*z, z), [x, y, z])` returns $[-1, 0, 0]$

Note:

To get the Hessian at critical points, we look for the critical points.

We enter:

```
solve(diff(2*x^2*y-x*z^3, [x, y, z]), [x, y, z])
```

We get:

```
[[0, y, 0]]
```

Then, we calculate the Hessian at these points.

We enter:

```
subst([[4*y, 4*x, -(3*z^2)], [2*2*x, 0, 0], [-
(3*z^2), 0, 6*x*z]], [x, y, z], [0, y, 0])
```

We get:

```
[[4*y, 4*0, -(3*0^2)], [4*0, 0, 0], [-(3*0^2), 0, 6*0*0]]
```

and after simplification:

```
[[4*y, 0, 0], [0, 0, 0], [0, 0, 0]]
```

3.11.5 Laplacian: `laplacian`

`laplacian` has two parameters: an expression F depending on n real variables and a dimension n vector storing the name of these variables.

laplacian returns the laplacian of F ($\nabla^2(F) = \frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} + \frac{\partial^2 F}{\partial z^2}$ if $n = 3$).

Example

Determine the laplacian of $F(x, y, z) = 2x^2y - xz^3$.

We enter:

```
laplacian(2*x^2*y-x*z^3, [x, y, z])
```

We get:

$$4*y + -6*x*z$$

3.11.6 Potential: potential

potential has two arguments: a vector \vec{V} of \mathbb{R}^n depending on n variables and the vector storing the name of these variables.

potential returns a function U so that $\overrightarrow{\text{grad}}(U) = \vec{V}$

when possible. Then, the potential U will return \vec{V} .

The general solution is the sum of a particular solution and a constant.

We know that a vector \vec{V} is a gradient if and only if its rotational curl is null:

in other words if $\text{curl}(\vec{V}) = 0$.

potential is the reciprocal function of derive.

We enter:

```
potential([2*x*y+3, x^2-4*z, -4*y], [x, y, z])
```

We get:

$$2*y*x^2/2 + 3*x + (x^2 - 4*z - 2*x^2/2) * y$$

3.11.7 Conservative vector field: vpotential

vpotential has two arguments: a vector \vec{V} of \mathbb{R}^n depending on n variables and the vector storing the name of these variables.

vpotential returns a vector \vec{U} such as $\overrightarrow{\text{Rot}}(\vec{U}) = \vec{V}$ when possible. Then, we say that \vec{V} is a conservative vector field or a solenoidal field.

The general solution is the sum of a particular solution and the gradient of an arbitrary function, the calculator returns the particular solution vector whose first component is null.

One knows that a vector \vec{V} is a rotational curl if and only if its divergence is null: in other words if $\text{divergence}(\vec{V}) = 0$.

In electromagnetism science, we have:

$\vec{V} = \vec{B}$ = the magnetic field and

$\vec{U} = \vec{A}$ = the potential vector.

vpotential is the reciprocal function of curl.

We enter:

```
vpotential([2*x*y+3, x^2-4*z, -2*y*z], [x, y, z])
```

We get:

$$[0, -(2*y) * z * x, -x^3/3 - (4*z) * x + 3*y]$$

3.12 Integral

3.12.1 Primitive and definite integral: `integrate`

`integrate` (or `int`) allows to calculate a primitive or a definite integral.

We enter:

```
int(exp(x), x, 0, 1)
```

or

```
integrate(exp(x), x, 0, 1)
```

We get:

```
exp(1) - 1
```

We enter:

```
evalf(integrate(exp(x^2), x, 0, 1))
```

Or we enter:

```
evalf(int(exp(x^2), x, 0, 1))
```

We get:

```
1.46265174591
```

`integrate` (or `int`) has one, two or four arguments.

- with an argument which is an expression of the variable x , (resp. a function).
`integrate` (or `int`) then returns an expression which is a primitive of the expression according to the variable x (resp. returns a primitive function of the function supplied as argument)

We enter:

```
integrate(x^2)
```

We get:

```
x^3/3
```

We enter:

```
f(t) := t^2
g := integrate(f)
```

We get:

```
(t) -> t^3/3
```

- with two arguments which are:
an expression and a variable, `integrate` (or `int`) then returns a primitive of the expression according to the variable supplied as second parameter.
We enter:

```
integrate(x^2)
```

We get:

$$x^3/3$$

We enter:

```
integrate(t^2,t)
```

We get:

$$t^3/3$$

- with four arguments which are:
an expression, a variable and the boundaries of the definite integral, `integrate` (or `int`) then returns the value of the definite integral.

We enter:

```
integrate(x^2,x,1,2)
```

We get:

$$7/3$$

We enter:

```
integrate(1/(sin(x)+2),x,0,2*pi)
```

We get after simplification (thanks to `simplify`):

$$2\pi\sqrt{3}/3$$

Exercise 1

Let

$$f(x) = \frac{x}{x^2-1} + \ln\left(\frac{x+1}{x-1}\right)$$

Calculate a primitive of f .

We enter:

```
int(x/(x^2-1)+ln((x+1)/(x-1)))
```

We find:

$$x \cdot \log\left(\frac{x+1}{x-1}\right) + \log(x^2-1) + 1/2 \cdot \log(2x^2/2-1)$$

Or else, we define the function f by entering:

```
f(x):=x/(x^2-1)+ln((x+1)/(x-1))
```

then we enter:

```
int(f(x))
```

We get, of course, the same result.

Warning!

In CAS, `log` is similar to `ln` (neperian logarithm), and `log10` is the logarithm in basis 10.

Exercise 2

Calculate:

$$\int \frac{2}{x^6 + 2 \cdot x^4 + x^2} dx$$

We enter:

```
int(2/(x^6+2*x^4+x^2))
```

We find:

$$2 * ((3*x^2+2) / (-2*(x^3+x))) - 3/2 * \operatorname{atan}(x)$$

Exercise 3

Calculate:

$$\int \frac{1}{\sin(x) + \sin(2 \cdot x)} dx$$

We enter:

```
integrate(1/(sin(x)+sin(2*x)))
```

We find:

$$(1/-3 * \log((\tan(x/2))^2 - 3) + 1/12 * \log((\tan(x/2))^2)) * 2$$

3.12.2 Integration by parts: `ibpdv`

`ibpdv` allows to look for a primitive (or to calculate a definite integral) of an expression of the form $u(x) \cdot v'(x)$.

`ibpdv` has two parameters for the primitives and five parameters for the integrals defined:

- either an expression of the form $u(x) \cdot v'(x)$ and $v(x)$ (or a list of two expressions $[F(x), u(x) * v'(x)]$ and $v(x)$),
- either an expression of the form $g(x)$ and 0 (or a list of two expressions $[F(x), g(x)]$ and 0).
- for the defined integrals, three other parameters must be added: the name of the variable and the boundaries.

Value returned by `ibpdv` depending on its parameters:

- `ibpdv(u(x).v'(x), v(x))` (resp. `ibpdv([F(x), u(x).v'(x)], v(x))`) returns:
if $v(x) \neq 0$, a list formed of $u(x).v(x)$ and $-v(x).u'(x)$ (resp. a list formed of $F(x) + u(x).v(x)$ and $-v(x).u'(x)$),
- `ibpdv(g(x), 0)` (resp. `ibpdv([F(x), g(x)], 0)`) returns:
a primitive $G(x)$ of $g(x)$ (resp. $F(x) + G(x)$) where $\operatorname{diff}(G(x)) = g(x)$.
- `ibpdv(u(x)*v'(x), v(x), x, a, b)` (resp. `ibpdv([F(x), u(x)*v'(x)], v(x), x, a, b)`) returns:
- if $v(x) \neq 0$, a list formed of $u(b).v(b) - u(a).v(a)$ and $-v(x).u'(x)$ (resp. a list formed of $F(b) + u(b).v(b) - F(a) - u(a).v(a)$ and $-v(x).u'(x)$),
- if the second argument is null, `ibpdv(g(x), 0, x, a, b)` returns:
 $G(b) - G(a)$ where $G(x)$ is a primitive of the first argument $g(x)$ (resp. `ibpdv([F(x), g(x)], 0, x, a, b)` returns $F(x) + G(b) - G(a)$ where $G(x)$ is a primitive of $g(x)$).

We enter:

```
ibpdv(ln(x), x)
```

We get:

```
[x.ln(x), -1]
```

then we enter

```
ibpdv([x.ln(x), -1], 0)
```

We get:

$$-x+x.\ln(x)$$

We enter:

$$\text{ibpdv}(\ln(x), x, x, 1, 2)$$

We get:

$$[2*\ln(2), -1]$$

We enter:

$$\text{ibpdv}(\ln(x), x, x, 2, 3)$$

We get:

$$[3*\ln(3) - 2*\ln(2), -1]$$

then we enter:

$$\text{ibpdv}([3*\ln(3) - 2*\ln(2), -1], 0, x, 2, 3)$$

We get:

$$-1+3*\ln(3) - 2*\ln(2)$$

3.12.3 Integration by parts: `ibpu`

`ibpu` allows to look for a primitive (or to calculate a definite integral) of an expression of the form $u(x).v'(x)$.

`ibpu` has two parameters for the primitives and five parameters for the integrals defined:

- either an expression of the form $u(x).v'(x)$ and $u(x)$ (or a list of two expressions $[F(x), u(x) * v'(x)]$ and $u(x)$),
- either an expression of the form $g(x)$ and 0 (or a list of expressions $[F(x), g(x)]$ and 0).
- for the defined integrals, three other parameters must be added: the name of the variable and the boundaries.

Value returned by `ibpu` according to its parameters:

- `ibpu(u(x).v'(x), u(x))` (resp. `ibpu([F(x), u(x).v'(x)], u(x))`) returns:
if $u(x) \neq 0$, a list formed of $u(x).v(x)$ and $-v(x).u'(x)$ (resp. a list formed of $F(x) + u(x).v(x)$ and $-v(x).u'(x)$),
- `ibpu(g(x), 0)` (resp. `ibpu([F(x), g(x)], 0)`) returns:
 $G(x)$ a primitive of $g(x)$ (resp. $F(x) + G(x)$ where $\text{diff}(G(x)) = g(x)$).
- `ibpu(u(x)*v'(x), u(x), x, a, b)` (resp. `ibpu([F(x), u(x)*v'(x)], u(x), x, a, b)`) returns:
if $u(x) \neq 0$, a list formed of $u(b).v(b) - u(a).v(a)$ and $-v(x).u'(x)$ (resp. a list formed of $F(b) + u(b).v(b) - F(a) - u(a).v(a)$ and $-v(x).u'(x)$),
- if the second argument is null, `ibpu(g(x), 0, x, a, b)` returns:
 $G(b) - G(a)$ where $G(x)$ a primitive of $g(x)$ (resp. $F(x) + G(b) - G(a)$ where $G(x)$ is a primitive of $g(x)$).

We enter:

$$\text{ibpu}(\ln(x), \ln(x))$$

We get:

$$[x.\ln(x), -1]$$

then we enter:

```
ibpu([x.ln(x), -1], 0)
```

We get:

$$-x+x.\ln(x)$$

We enter:

```
ibpu(ln(x), ln(x), x, 2, 3)
```

We get:

$$[3*\ln(3) - 2*\ln(2), -1]$$

then we enter:

```
ibpu([3*\ln(3) - 2*\ln(2), -1], 0, x, 2, 3)
```

We get:

$$-1+3*\ln(3) - 2*\ln(2)$$

3.12.4 Evaluate a primitive: `preval`

`preval` has three parameters: an expression $F(x)$ depending on the variable x , and two expressions a and b .

`preval` does $F(b)-F(a)$.

`preval` is useful to calculate a definite integral by a primitive: we calculate a primitive, then one evaluates this primitive between the two boundaries of the integral.

We enter:

```
preval(x^2+x, 2, 3)
```

We get:

$$6$$

We enter:

```
int(ln(x))
```

We get:

$$x*\ln(x) - x$$

We enter:

```
preval(x*\ln(x) - x, 2, 3)
```

We get:

$$3*\ln(3) - 3 - 2*\ln(2) + 2$$

3.13 Limits

3.13.1 Riemann sum: `sum_riemann`

`sum_riemann` has two arguments: an expression `Xpr` depending on two variables and the list of names of these two variables.

`sum_riemann(Xpr(n,k), [n,k])` returns an equivalent, in the vicinity of $n = +\infty$, of $\sum_{k=1}^n Xpr(n,k)$ or of $\sum_{k=0}^{n-1} Xpr(n,k)$ or of $\sum_{k=1}^{n-1} Xpr(n,k)$, when the sum considered is a Riemann sum associated to a function continuous on $[0,1]$ or, when the search was unsuccessful, returns "This is probably not a Riemann sum".

Let

$$S_n = \sum_{k=1}^n \frac{k^2}{n^3}$$

Calculate

$$\lim_{n \rightarrow +\infty} S_n$$

We enter:

```
sum_riemann(k^2/n^3, [n,k])
```

We get:

$$1/3$$

because:

$$\sum_{k=1}^n \frac{k^2}{n^3} = \frac{1}{n} \sum_{k=1}^n \frac{k^2}{n^2}$$

is the Riemann sum associated to:

$$\int_0^1 x^2 dx = \frac{1}{3}$$

Let

$$S_n = \sum_{k=1}^n \frac{k^3}{n^4}$$

Calculate

$$\lim_{n \rightarrow +\infty} S_n$$

We enter:

```
sum_riemann(k^3/n^4, [n,k])
```

We get:

$$1/4$$

because:

$$\sum_{k=1}^n \frac{k^3}{n^4} = \frac{1}{n} \sum_{k=1}^n \frac{k^3}{n^3}$$

is the Riemann sum associated to:

$$\int_0^1 x^3 dx = \frac{1}{4}$$

Let

$$S_n = \sum_{k=1}^n \frac{32n^3}{16n^4 - k^4}$$

Calculate

$$\lim_{n \rightarrow +\infty} S_n$$

We enter:

$$\text{sum_riemann}(32*n^3/(16*n^4-k^4), [n, k])$$

We get:

$$2*\text{atan}(1/2)+\log(3)$$

because:

$$\sum_{k=1}^n \frac{32n^3}{16n^4 - k^4} = \sum_{k=1}^n \frac{32}{16 - \left(\frac{k}{n}\right)^4}$$

is the Riemann sum associated to:

$$\int_0^1 \frac{32}{16 - x^4} dx = \int_0^1 \frac{1}{x+2} - \frac{1}{x-2} \frac{4}{x^2+4}$$

which then equals $\ln(3) - \ln(2) + \ln(2) - \ln(1) + 2 \text{atan}(1/2) = \ln(3) + 2 \text{atan}(1/2)$

Calculate

$$\lim_{n \rightarrow +\infty} \left(\frac{1}{n+1} + \frac{1}{n+2} + \dots + \frac{1}{n+n} \right)$$

We enter:

$$\text{sum_riemann}(1/(n+k), [n, k])$$

We get:

$$\ln(2)$$

because:

$$\sum_{k=1}^n \frac{1}{n+k} = \frac{1}{n} \sum_{k=1}^n \frac{1}{1 + \left(\frac{k}{n}\right)}$$

is the Riemann sum associated to:

$$\int_0^1 \frac{1}{1+x} dx = \ln(1+1) = \ln(2)$$

Calculate

$$\lim_{n \rightarrow +\infty} \left(\frac{n}{n^2+1^2} + \frac{n}{n^2+2^2} + \dots + \frac{n}{n^2+n^2} \right)$$

We enter:

$$\text{sum_riemann}(n/(n^2+k^2), [n, k])$$

We get:

$$\text{pi}/4$$

because:

$$\sum_{k=1}^n \frac{n}{n^2 - k^2} = \frac{1}{n} \sum_{k=1}^n \frac{1}{1 + \left(\frac{k}{n}\right)^2}$$

is the Riemann sum associated to:

$$\int_0^1 \frac{1}{1-x^2} dx = \operatorname{atan}(1) = \frac{\pi}{4}$$

Calculate

$$\lim_{n \rightarrow +\infty} \left(\frac{1}{\sqrt{n^2 + 1^2}} + \frac{1}{\sqrt{n^2 + 2^2}} + \dots + \frac{1}{\sqrt{n^2 + n^2}} \right)$$

We enter:

```
sum_riemann(1/sqrt(n^2+k^2), [n, k])
```

We get:

```
-ln(sqrt(2)-1)
```

because:

$$\sum_{k=1}^n \frac{n}{\sqrt{n^2 - k^2}} = \frac{1}{n} \sum_{k=1}^n \frac{1}{\sqrt{1 - \left(\frac{k}{n}\right)^2}}$$

is the Riemann sum associated to:

$$\int_0^1 \frac{1}{\sqrt{1-x^2}} dx = \ln(1 + \sqrt{1+1^2}) - \ln(0 + \sqrt{1+0^2}) = \ln(1 + \sqrt{2})$$

3.13.2 Series expansion: `taylor`

`taylor` has one to four parameters:

the expression to be developed, `x=a` (by default `x=0`), the order of development (by default 5), or:
the expression to be developed, `x`, the order of development (by default 5) and the point in the vicinity of which we want the development (by default 0).

Note: we can also put `x, a, n` instead of `x=a, n`

`taylor` returns a polynomial in `x-a`, plus a rest that the calculator writes:

```
(x-a)^n*order_size(x-a)
```

This means that we have a series expansion at order $n-1$ (or at order $p < n$).

Indeed, `order_size` designates a function so that, regardless r positive:

$x^r \cdot \operatorname{order_size}(x)$ tends to zero when x tends to zero.

For instance, the constant functions, the `log` (or `ln`) function, are `order_size` functions.

We enter:

```
taylor(sin(x), x=1, 2)
```

Or we enter (mind the order of arguments!):

```
taylor(sin(x), x, 2, 1)
```

We get:

```
sin(1)+cos(1)*(x-1)-(sin(1)/2)*(x-1)^2+(x-1)^3*order_size(x-1)
```

3.13.3 Division by increasing power order: `divpc`

`divpc` has three arguments: two polynomials $A(x)$, $B(x)$ (with $B(0) \neq 0$) and an integer n .
`divpc` returns the quotient $Q(x)$ of the division of $A(x)$ by $B(x)$ by increasing power order with $\text{degree}(Q) \leq n$ or $Q = 0$.

$Q(x)$ is then the series expansion order n of $\frac{A(x)}{B(x)}$ in the vicinity of $x = 0$.

We enter:

```
divpc(1+x^2+x^3,1+x^2,5)
```

We get:

```
-x^5+x^3+1
```

Warning! This command does not work if the polynomials are written with the list of their coefficients.

3.14 Transform

3.14.1 Laplace transform: `laplace`

`laplace` has one, two or three arguments:

the expression to be transformed and eventually the name of one or two variables.

The expression is an expression of the current variable (here x) or the expression to be transformed is an expression of the supplied variable as second argument.

`laplace` is the Laplace transform of the expression supplied as argument .

The result of `laplace` is an expression of variable: the third argument, or by default the second argument, or by default x .

We enter:

```
laplace(sin(x))
```

We get:

```
1/(x^2+1)
```

Or we enter:

```
laplace(sin(t),t)
```

We get:

```
1/(t^2+1)
```

Or we enter:

```
laplace(sin(x),x,t)
```

We get:

```
1/(t^2+1)
```

Or we enter:

```
laplace(sin(t),t,s)
```

We get:

$$1/(s^2+1)$$

3.14.2 Laplace transform inverse: `invlaplace`

`invlaplace` (or `ilaplace`) has one, two or three arguments:

the expression to be transformed and eventually the name of one or two variables.

The expression is an expression of the current variable (here x) or the expression to be transformed is an expression of the supplied variable as second argument.

`invlaplace` is the inverse Laplace transform of the expression supplied as argument. The result of `invlaplace` is an expression of variable: the third argument, or by default the second argument, or by default x .

We enter:

```
invlaplace(1/(x^2+1))
```

We get:

```
sin(x)
```

Or we enter:

```
invlaplace(1/(t^2+1), t)
```

We get:

```
sin(t)
```

Or we enter:

```
invlaplace(1/(t^2+1), t, x)
```

We get:

```
sin(x)
```

Note:

We use the Laplace transform (`laplace`) and the inverse Laplace transform (`ilaplace` or `invlaplace`) to solve differential equations linear at constant coefficients, for example:

$$y'' + p.y' + q.y = f(x)$$

$$y(0) = a \quad y'(0) = b$$

By noting \mathcal{L} the Laplace transform, we have the following relations:

$$\mathcal{L}(y)(x) = \int_0^{+\infty} e^{-x.u} y(u) du$$

$$\mathcal{L}^{-1}(g)(x) = \frac{1}{2i\pi} \int_{\mathcal{C}} e^{z.x} g(z) dz$$

where \mathcal{C} is a closed curve containing the poles of g .

Example:

Solve:

$$y'' - 6.y' + 9.y = x.e^{3.x}, \quad y(0) = c_0, \quad y'(0) = c_1$$

Here, $p = -6$, $q = 9$.

We enter:

```
laplace(x*exp(3*x))
```

We get:

$$1/(x^2 - 6x + 9)$$

We enter:

$$\text{ilaplace}((1/(x^2 - 6x + 9) + (x - 6) * c_0 + c_1) / (x^2 - 6x + 9))$$

We get

$$(216 * x^3 - 3888 * x * c_0 + 1296 * x * c_1 + 1296 * c_0) * \exp(3 * x) / 1296$$

after simplification and factorization (command `factor`) the solution reads:

$$(-18 * c_0 * x + 6 * c_0 + x^3 + 6 * x * c_1) * \exp(3 * x) / 6$$

One can, of course, press directly:

$$\text{desolve}(y'' - 6 * y' + 9 * y = x * \exp(3 * x), y)$$

We get:

$$\exp(3 * x) * (-18 * c_0 * x + 6 * c_0 + x^3 + 6 * x * c_1) / 6$$

3.14.3 Fast Fourier transform: `fft`

`fft` takes as argument a list (or a sequence) $[a_0, \dots, a_{N-1}]$ where N is a power of two.

`fft` returns the list $[b_0, \dots, b_{N-1}]$ such as for $k=0 \dots N-1$ such as:

$$\text{fft}([a_0, \dots, a_{N-1}])[k]$$

$$= b_k = \sum_{j=0}^{N-1} x_j \omega_N^{-k \cdot j}$$

with ω_N N -th root of the unity.

We enter:

$$\text{fft}(0, 1, 1, 0)$$

We get:

$$[2., -1-i, 0., -1+i]$$

Note: we can also work on a field $\mathbb{Z}/p\mathbb{Z}$, by giving an N -th primitive root of unity as second argument and p as third argument of `fft`.

3.14.4 inverse of the fast Fourier transform: `ifft`

`ifft` takes as argument a list or a sequence $[b_0, \dots, b_{N-1}]$ where N is a power of two.

`ifft` returns the list $[a_0, \dots, a_{N-1}]$ such as:

$$\text{ifft}([a_0, \dots, a_{N-1}]) = [b_0, \dots, b_{N-1}].$$

We enter:

$$\text{ifft}([2, -1-i, 0, -1+i])$$

Or we enter:

$$\text{ifft}(2, -1-i, 0, -1+i)$$

We get:

[0., 1., 1., 0.]

Note: we can also work on a field $\mathbb{Z}/p\mathbb{Z}$, by giving an N -th primitive root of unity as second argument and p as third argument of `ifft`.

Chapter 4 Menu Solve

4.1 Solve equations: solve

`solve` allows to solve an equation or a set of polynomial equations.

`solve` takes one or two arguments which are an expression xpr in x or an expression xpr of a variable var and the name of this variable var .

`solve` solves $xpr = 0$, the unknown value being x or var

Warning! The second variable can specify an interval, for example $x = a..b$, to only have the solutions in the interval $[a; b]$ but in this case the solutions will be numerical and `solve` is then similar to `fsolve`, for example:

`solve(t^2-2, t=0..2)` or `fsolve(t^2-2, t=0..2)` returns `[1.41421356237]`

whereas `solve(t^2-2, t)` returns `[-(sqrt(2)), sqrt(2)]`.

We enter:

```
solve(x^2-3*x+2=0)
```

We get:

```
{1, 2}
```

We enter:

```
solve(x^4-1=0)
```

We get:

```
{-sqrt(2), sqrt(2)}
```

We enter:

```
solve([x+y=3, x*y=2], [x, y])
```

Or we enter:

```
solve({x+y=3, x*y=2}, {x, y})
```

We get:

```
{[1, 2], [2, 1]}
```

We enter:

```
solve([-x^2+y=2, x^2+y=0], [x, y])
```

Or we enter:

```
solve({-x^2+y=2, x^2+y=0}, {x, y})
```

We get:

```
{}
```

4.2 Zeros of an expression: `zeros`

`zeros` takes as parameter an expression.

`zeros` returns the list of elements which cause the expression to vanish.

Depending on the chosen mode, if we are in real mode (`complex_mode:=0` or if `i` is not checked in the CAS Settings) the zero will be real and if we are in complex mode (`complex_mode:=1` or if `i` is checked in the CAS Settings) the zero will be complex.

We enter:

```
zeros(x^2-3*x+2)
```

We get:

```
[2,1]
```

We enter:

```
zeros(x^4-1)
```

We get:

```
[1,-1]
```

We enter:

```
zeros([x+y-3,x*y-2],[x,y])
```

Or we enter:

```
zeros({x+y-3,x*y-2},{x,y})
```

We get:

```
[[1,2],[2,1]]
```

We enter:

```
zeros([-x^2+y-2,x^2+y],[x,y])
```

Or we enter:

```
zeros({-x^2+y-2,x^2+y},{x,y})
```

We get:

```
[]
```

4.3 Complex Zeros of an expression: `cZeros`

`cZeros` takes as parameter an expression.

`cZeros` returns the list of complex elements which make the expression equals zero.

Note:

Difference between `zeros` and `cZeros`: in complex mode, `zeros` returns the same result as `cZeros` (as far as `cZeros` is concerned, being in complex mode or real mode does not matter much). Thus, if we do not want that the result depends on the mode, it is better to use `cZeros` to get the complex solutions.

We enter in real or complex mode:

```
cZeros(x^2+4)
```

We get:

```
[-2*i, 2*i]
```

We enter:

```
cZeros(ln(x)^2-2)
```

We get:

```
[exp(sqrt(2)), exp(-sqrt(2))]
```

We enter:

```
cZeros(ln(y)^2-2, y)
```

We get:

```
[exp(sqrt(2)), exp(-sqrt(2))]
```

We enter:

```
cZeros(x*(exp(x))^2-2*x-2*(exp(x))^2+4)
```

We get:

```
[[log(sqrt(2)), log(-sqrt(2)), 2]]
```

4.4 Solve equations in \mathbb{C} : `cSolve` `csolve`

`cSolve` or `csolve` solves an equation or a set of polynomial equations in \mathbb{C} without needing to be in complex mode.

Note:

Difference between `solve` and `csolve`: in complex mode `solve` returns the same result as `csolve` (as far as `csolve` is concerned, being in complex mode or real mode does not matter much). Thus, if we do not want that the result depends on the mode, for it is better to use `csolve` to get the complex solutions.

We enter in real or complex mode:

```
cSolve(x^4-1=3)
```

or

```
csolve(x^4-1=3)
```

We get:

```
[sqrt(2), -sqrt(2), sqrt(2)*i, -sqrt(2)*i]
```

We enter:

```
cSolve([-x^2+y=2, x^2+y=0], [x, y])
```

Or we enter:

```
cSolve({-x^2+y=2, x^2+y=0}, {x, y})
```

We get:

```
{[i, 1], [-i, 1]}
```

4.5 Complex zeros of an expression: `cZeros`

`cZeros` takes as parameter an expression.

`cZeros` returns the list of complex elements which make the expression equals zero.

Note:

Difference between `zeros` and `cZeros`: in complex mode `zeros` returns the same result as `cZeros` (as far as `cZeros` is concerned, being in complex mode or real mode does not matter much). Thus, if we do not want that the result depends on the mode, it is better to use `cZeros` to get the complex solutions.

We enter in real or complex mode:

```
cZeros(x^4-1)
```

We get:

```
[1, -1, -i, i]
```

We enter:

```
cZeros([-x^2+y-2, x^2+y], [x, y])
```

Or we enter:

```
cZeros({-x^2+y-2, x^2+y}, {x, y})
```

We get:

```
[[-i, 1], [i, 1]]
```

4.6 Differential equations

For the numerical calculation of solutions of differential equations please refer to `odesolve` and for the graphical representation of solutions of differential equations please refer to `plotfield`, `plotode`.

4.6.1 Solve differential equations: `deSolve` `desolve`

`deSolve` or `desolve` allows to solve:

- the linear differential equations with constant coefficients of order one or order two,
- the linear differential equations of order one,
- the differential equations of order one incomplete in y ,
- the differential equations of order one incomplete in x ,
- the differential equations of order one of separate variables,
- the differential equations of order one homogeneous ($y' = F(y/x)$),
- the differential equations of order one having one integrating factor,
- the differential equations of Bernoulli ($a(x)y' + b(x)y = c(x)y^n$),
- the differential equations of Clairaut ($y = x * y' + f(y')$).

Parameters of `desolve`:

- when the differential equation is of order one, the variable is x and the unknown value is y , the parameters are:
the differential equation or

the differential equation followed by the list $[x_0, y_0]$ which sets as initial condition $y(x_0) = y_0$.

- when the variable is x , the parameters are: the differential equation (or the list formed by the differential equation and the initial conditions) and the unknown y .

In the differential equation y reads y , y' reads y' , y'' reads y'' , because we derivate according to the variable x . For instance: `desolve(y''+2*y'+y, y)` and `desolve([y''+2*y'+y, y(0)=1, y'(0)=0], y)`.

- when the variable is not x (for example t), the parameters are: the differential equation (or the list formed by the differential equation and the initial conditions), the variable t and the unknown y or the unknown $y(t)$ (the variable is then t and the unknown is y).

In the differential equation y reads $y(t)$ and y' reads `diff(y(t), t)`, y'' reads `diff(y(t), t$2)`.

For instance:

```
deSolve(diff(y(t), t$2)+2*diff(y(t), t)+y(t), y(t));
```

or

```
deSolve(diff(y(t), t$2)+2*diff(y(t), t)+y(t), t, y);
```

and

```
deSolve([diff(y(t), t$2)+2*diff(y(t), t)+y(t), y(0)=1, y'(0)=0], y(t));
```

or

```
deSolve([diff(y(t), t$2)+2*diff(y(t), t)+y(t), y(0)=1, y'(0)=0], t, y);
```

We enter (by pressing `Shift-()` for `''`):

```
deSolve(y''+y=cos(x), y)
```

or else:

```
deSolve((diff(diff(y))+y)=(cos(x)), y)
```

We find:

```
G_0*cos(x)+(x+2*G_1)/2*sin(x)
```

c_0 , c_1 are the integration constants: $y(0)=c_0$ and $y'(0)=c_1$.

We enter, if we want the solutions that make $y(0) = 1$:

```
deSolve([y''+y=cos(x), y(0)=1], y)
```

We get

```
[cos(x)+(x+2*c_1)/2*sin(x)]
```

the components of this vector are solutions (here we have one single component because we get one single solution depending on the constant c_1).

Exercise

Find the differentiable functions f which make:

$$f'(x) = f(-x).$$

The function f' is then differentiable and we have:

$$f''(x) = -f'(-x) = -f(x).$$

f then makes the differential equation $y'' + y = 0$ which is easy to integrate.

So f is solution of the differential equation: $y'' + y = 0$.

We enter:

```
desolve(y''+y=0)
```

We get:

```
c_0*cos(x)+c_1*sin(x)
```

So $f(x)$ is of the form $c_0*cos(x)+c_1*sin(x)$

Let us look for the values of c_0 and c_1 to get $f'(x) - f(-x) = 0$ for all the values of x .

We enter:

```
f(x):=c_0*cos(x)+c_1*sin(x)
factor(f'(x)-f(-x))
```

We get:

```
(-sin(x)-cos(x))*(c_0-c_1)
```

So $c_0=c_1$

So the differentiable functions f which make $f'(x) = f(-x)$ are the functions equal to: $c * (\cos(x) + \sin(x))$ where c is an arbitrary constant.

Or $f(x) = k \cos(x - \pi/4)$ where k is an arbitrary constant.

We check by entering:

```
(cos(x)+sin(x))'-(cos(-x)+sin(-x))
```

or by entering

```
cos(x-pi/4)'-cos(-x-pi/4)
```

We get

```
0
```

Similar exercise

Find the differentiable functions f of \mathbb{R}^+ in \mathbb{R} which make:

$$f'(x) = f(1/x).$$

The function f' of \mathbb{R}^* in \mathbb{R} is then differentiable and we have:

$$f''(x) = -f'(1/x)/x^2 = -f(x)/x^2$$

So f then makes the differential equation $x^2y'' + y = 0$ which is more difficult to integrate.

We enter:

```
factor(desolve(x^2*y''+y=0))
```

We get:

```
sqrt(x)*(c_0*cos(2*sqrt(3)*ln(x)/4)+c_1*sin(2*sqrt(3)*ln(x)/4))
```

Let us look for the values of c_0 and c_1 so that f makes:

$$f'(x) - f(1/x) = 0 \text{ for all the values of } x.$$

We enter:

```
f(x):=sqrt(x)*(c_0*cos(sqrt(3)*ln(x)/2)+c_1*sin(sqrt(3)*ln(x)/2))
factor(f'(x)-f(1/x))
```

We get:

```
sqrt(x)*(cos(sqrt(3)*ln(x)/2)+sqrt(3)*sin(sqrt(3)*ln(x)/2))*(-c_0-(-sqrt(3)))
```

So $(-c_0 - (-\sqrt{3})) * c_1 = 0$ that is to say $-c_0 = c_1 * \sqrt{3}$.

So the differentiable functions f which make $f'(x) = f(1/x)$ are the functions equal to:

$$c * (\sqrt{3} \cos(x) + \sin(x)) \text{ where } c \text{ is an arbitrary constant.}$$

We check by entering:

```
f(x) := sqrt(x) * (sqrt(3) * cos(sqrt(3) * ln(x) / 2) + sin(sqrt(3) * ln(x) / 2))
```

Then, `simplify(f'(x) - f(1/x))` returns 0.

To do the integration by hand, we consider $t = \ln(x)$ i.e. $x = e^t$.

We have:

$$y'_x = y'_t/x \text{ and } y''_x = y''_t/x^2 - y'_t/x^2 = 1/x^2(y''_t - y'_t)$$

So $g(t) = f(e^t)$ checks the differential equation:

$y''_t - y'_t + y = 0$ whose characteristic equation is $r^2 - r + 1 = 0$. Thus $g(t) = f(e^t)$ is of the form:

$$e^{\frac{t}{2}} \left(a \cos\left(\frac{\sqrt{3} \ln(x)}{2}\right) + b \sin\left(\frac{\sqrt{3} \ln(x)}{2}\right) \right)$$

i.e. $f(x)$ is of the form:

$$f(x) = e^{\frac{\ln(x)}{2}} \left(a \cos\left(\frac{\sqrt{3} \ln(x)}{2}\right) + b \sin\left(\frac{\sqrt{3} \ln(x)}{2}\right) \right)$$

Let us look for the values of a and b to get:

$$f'(x) = f(1/x).$$

$$f'(x) = e^{\frac{\ln(x)}{2}} \frac{\left(a \left(-\sqrt{3} \sin\left(\frac{\sqrt{3} \ln(x)}{2}\right) + \cos\left(\frac{\sqrt{3} \ln(x)}{2}\right) \right) + b \left(\sin\left(\frac{\sqrt{3} \ln(x)}{2}\right) + \sqrt{3} \cos\left(\frac{\sqrt{3} \ln(x)}{2}\right) \right) \right)}{2x}$$

and

$$f'(x) - f(1/x) = 0 \text{ causes:}$$

$$(-a + b\sqrt{3}) = 0 \text{ then } a = b\sqrt{3}.$$

If we look for the differentiable functions f of \mathbb{R}^* in \mathbb{R} which makes:

$$f'(x) = f(1/x).$$

You must consider for $x < 0$: $x = -\exp(t)$.

We get the same differential equation but the relation $y'(x) = y(1/x)$ gives as condition

$$c_1 = -c_0\sqrt{3}.$$

Then, we consider:

```
f(x) := exp(ln(abs(x)) / 2) * (sqrt(3) * cos(sqrt(3) / 2 * ln(abs(x))) + sin(sqrt(3) / 2 * ln(abs(x))))
```

```
g(x) := exp(ln(abs(x)) / 2) * (cos(sqrt(3) / 2 * ln(abs(x))) - sqrt(3) * sin(sqrt(3) / 2 * ln(abs(x))))
```

Then:

```
h(x) := ifte(x > 0, f(x), g(x))
```

```
k(x) := ifte(x > 0, f'(x), g'(x))
```

because if we enter $k(x) := h'(x)$ we have as a result:

```
ifte: impossible to perform the test Error: Incorrect Argument Value.
```

or we enter:

```
h(x) := when(x > 0, f(x), g(x)) and k(x) := h'(x) because ifte performs the test, but not when, or the expression is evaluated to derivate.
```

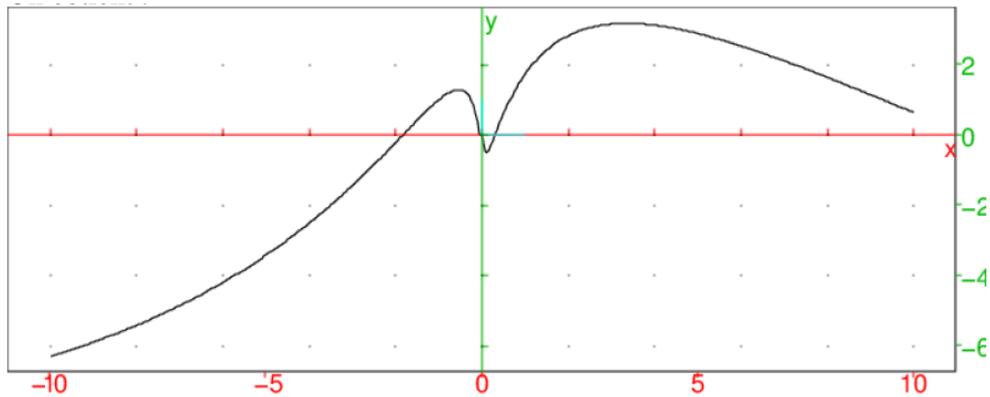
The functions $c \cdot h(x)$ where c is an arbitrary constant make $c \cdot h'(x) = c \cdot k(x) =$

$$c \cdot h(1/x)$$

We enter:

```
plotfunc(h(x))
```

We get:



We notice that:

`limit(f(x),x,0,1)` returns 0 and

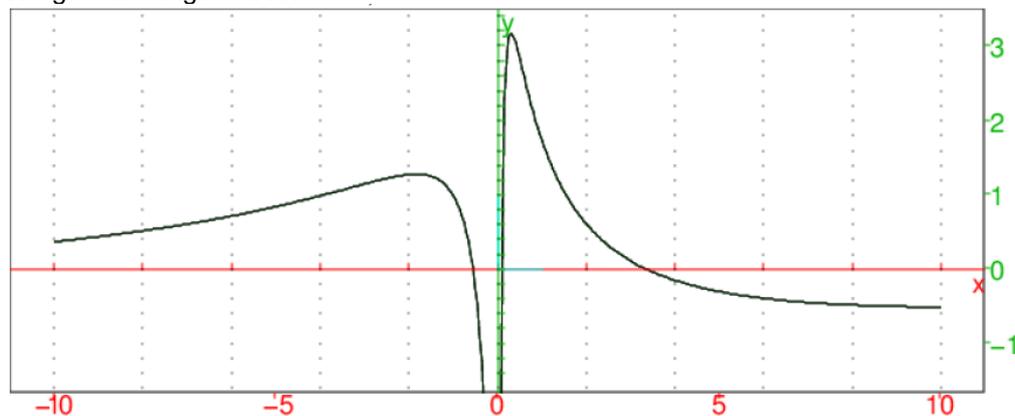
`limit(g(x),x,0,-1)` returns 0 so

$h(0)=0$ but h is not differentiable in 0 because $\lim_{x \rightarrow 0} (h(x)/x, x, 0)$ equals the infinite.

We enter:

```
plotfunc([k(x),h(1/x)])
```

We get one single curve:



4.6.2 Laplace transform and inverse Laplace transform: `/laplace ilaplace invlaplace`

`laplace` and `ilaplace` (or `invlaplace`) have one, two or three arguments: the expression to be transformed and eventually the name of two variables.

The expression is an expression of the current variable (here x) or the expression that we transform is an expression of the supplied variable as second argument.

`laplace` is the Laplace transform of the expression supplied as argument and `ilaplace` (or `invlaplace`) is the inverse Laplace transform of the expression supplied as argument. The result of `laplace` and `ilaplace` (or `invlaplace`) is an expression of variable the third argument, or by default the second argument, or by default x .

Warning! The second argument is the name of the variable of the first argument and is also the name of the variable of the result when there is no third argument, by example: `laplace(sin(x),t)` returns $\sin(x)/t$

We use the Laplace transform (`laplace`) and the Laplace inverse transform (`ilaplace` or `invlaplace`) to solve linear differential equations at constant coefficients, for example:

$$y'' + p.y' + q.y = f(x)$$

$$y(0) = a \quad y'(0) = b$$

By noting \mathcal{L} the Laplace transform, we have the following relations:

$$\mathcal{L}(y)(x) = \int_0^{+\infty} e^{-x.u} y(u) du$$

$$\mathcal{L}^{-1}(g)(x) = \frac{1}{2i\pi} \int_C e^{z.x} g(z) dz$$

where C is a closed curve containing the poles of g .

laplace:

We enter:

```
laplace(sin(x))
```

here we do not specify the variable, then the expression to be transformed (here $\sin(x)$) is an expression of the current variable (here x) and the transform will also be a function of the variable x .

We get:

```
1/(x^2+1)
```

Or we enter:

```
laplace(sin(t), t)
```

here we specify the name of the variable of the function to be transformed (here t) and this name of variable will be used for the Laplace transform.

We get:

```
1/(t^2+1)
```

Or we enter:

```
laplace(sin(t), t, s)
```

here we specify the name of the variable of the function to be transformed (here t) and the name of the variable that we wish to get for the Laplace transform (here s).

We get:

```
1/(s^2+1)
```

ilaplace or invlaplace:

We enter:

```
ilaplace(1/(x^2+1))
```

We get:

```
sin(x)
```

We enter:

```
ilaplace(1/(t^2+1), t)
```

We get:

```
sin(t)
```

We enter:

```
ilaplace(1/(t^2+1), t, x)
```

We get:

$$\sin(x)$$

We use the following properties:

$$\begin{aligned}\mathcal{L}(y')(x) &= -y(0) + x.\mathcal{L}(y)(x) \\ \mathcal{L}(y'')(x) &= -y'(0) + x.\mathcal{L}(y')(x) \\ &= -y'(0) - x.y(0) + x^2.\mathcal{L}(y)(x)\end{aligned}$$

We have then if $y''(x) + p.y'(x) + q.y(x) = f(x)$:

$$\begin{aligned}\mathcal{L}(f)(x) &= \mathcal{L}(y'' + p.y' + q.y)(x) \\ &= -y'(0) - x.y(0) + x^2.\mathcal{L}(y)(x) - p.y(0) + p.x.\mathcal{L}(y)(x) + q.\mathcal{L}(y)(x) \\ &= (x^2 + p.x + q).\mathcal{L}(y)(x) - y'(0) - (x + p).y(0)\end{aligned}$$

soit, if $a = y(0)$ and $b = y'(0)$:

$$\mathcal{L}(f)(x) = (x^2 + p.x + q).\mathcal{L}(y)(x) - (x + p).a - b$$

The solution is then:

$$y(x) = \mathcal{L}^{-1}((\mathcal{L}(f)(x) + (x + p).a + b)/(x^2 + p.x + q))$$

Example:

Solve:

$$y'' - 6.y' + 9.y = x.e^3.x, \quad y(0) = c_0, \quad y'(0) = c_1$$

Here, $p = -6, q = 9$.

We enter:

$$\text{laplace}(x*\exp(3*x))$$

We get:

$$1/(x^2 - 6*x + 9)$$

We enter:

$$\text{ilaplace}((1/(x^2 - 6*x + 9) + (x - 6)*c_0 + c_1)/(x^2 - 6*x + 9))$$

We get

$$(216*x^3 - 3888*x*c_0 + 1296*x*c_1 + 1296*c_0)*\exp(3*x)/1296$$

after simplification and factorization (command `factor`) the solution reads:

$$(-18*c_0*x + 6*c_0 + x^3 + 6*x*c_1)*\exp(3*x)/6$$

We can, of course, press directly:

$$\text{desolve}(y'' - 6*y' + 9*y = x*\exp(3*x), y)$$

We get:

$$\exp(3*x)*(-18*c_0*x + 6*c_0 + x^3 + 6*x*c_1)/6$$

4.7 Approximate solution of $y' = f(t, y)$: `odesolve`

Be f a function of \mathbb{R}^2 $y' = f(t, y)$ in \mathbb{R} .

`odesolve` returns the approximate value $y(t_1)$ of the solution of the differential equation $y' = f(t, y)$ when $y(t_0) = y_0$.

`odesolve` takes as parameters:

`odesolve(f(t,y), [t,y], [t0,y0], t1)` or
`odesolve(f(t,y), t=t0..t1, y, y0)` or
`odesolve(t0..t1, f, y0)` or
`odesolve(t0..t1, (t,y)->f(t,y), y0)`
 returns the approximate value of $y(t_1)$ when $y(t)$ is the solution of $y'(t) = f(t, y(t))$ which checks $y(t_0) = y_0$.
 – We can add an optional parameter for tell the wished discretization in time (`tstep=value`). This value is not necessarily respected by the solver.
 – We can add curve as optional parameter to get the list of $[t, [y(t)]]$ calculated instead of the only value of $y(t_1)$.

We enter:

```
odesolve(sin(t*y), [t,y], [0,1], 2)
```

or:

```
odesolve(sin(t*y), t=0..2, y, 1)
```

or:

```
odesolve(0..2, (t,y)->sin(t*y), 1)
```

or else we define the function:

```
f(t,y):=sin(t*y)
```

and we enter:

```
odesolve(0..2, f, 1)
```

We get:

```
[1.82241255674]
```

then we enter:

```
odesolve(0..2, f, 1, tstep=0.3)
```

We get:

```
[1.82241255675]
```

We enter:

```
odesolve(sin(t*y), t=0..2, y, 1, tstep=0.5)
```

We get:

```
[1.82241255675]
```

We enter:

```
odesolve(sin(t*y), t=0..2, y, 1, tstep=0.5, curve)
```

We get:

```
[[0.0, [1.0]], [0.3906, [1.07811817892]], [0.760963058921, [1.30972370161]],  
[1.07086790074, [1.60476137064]], [1.39334557444, [1.86417104883]],  
[1.78645581533, [1.90374891395]], [2.0, [1.82241253071]]]
```

We enter:

```
odesolve(sin(t*y), t=0..2, y, 1, curve)
```

Or we enter:

```
odesolve(sin(t*y), t=0..2, y, 1, tstep=0.3, curve)
```

We get:

```
[[0.0, [1.0]], [0.3781, [1.07309655677]], [0.6781, [1.24392692452]],
[0.9781, [1.51224777765]], [1.2781, [1.7904830809]], [1.5781, [1.92164503333]],
[1.8781, [1.87481063533]], [2.0, [1.82241255617]]]
```

4.8 z transform and z inverse transform

4.8.1 z transform of a series: ztrans

`ztrans` has one or three arguments:

- a sequence supplied by its general term a_x : the used variable to define the general term is x and x will also be the name of the used variable in the function returned by `ztrans`
- a sequence supplied by its general term a_n , the name of the used variable to define this general term (here n) and the name of the used variable in the function returned by `ztrans` (for example z).

`ztrans` returns the z transform of the series supplied as argument.

We have by definition:

if $f(x) = ztrans(ax)$ we have

$$f(x) = \sum_{n=0}^{\infty} \frac{a_n}{x^n}$$

if $f(z) = ztrans(a_n, n, z)$ we have

$$f(z) = \sum_{n=0}^{\infty} \frac{a_n}{z^n}$$

We enter:

```
ztrans(1)
```

We get:

```
x/(x-1)
```

We have indeed:

$$\sum_{n=0}^{\infty} \frac{1}{x^n} = \frac{1}{1 - \frac{1}{x}} = \frac{x}{x-1}$$

We enter:

```
ztrans(1, n, z)
```

We get:

```
z/(z-1)
```

We have indeed:

$$1 + \frac{1}{z} + \frac{1}{z^2} + \frac{1}{z^3} + \frac{1}{z^4} + \dots = \sum_{n=0}^{\infty} \frac{1}{z^n} = \frac{1}{1 - \frac{1}{z}} = \frac{z}{z-1}$$

We enter:

```
ztrans(x)
```

We get:

$$x / (x^2 - 2x + 1)$$

We enter:

```
ztrans(n, n, z)
```

We get:

$$z / (z^2 - 2z + 1)$$

We have indeed:

$$\frac{1}{z-1} = \sum_{n=1}^{\infty} \frac{n}{z^{n-1}}$$

$$\frac{1}{(z-1)^2} = -\left(\frac{1}{z-1}\right)' = \sum_{n=1}^{\infty} \frac{n}{z^{n-1}}$$

Thus

$$\frac{z}{(z-1)^2} = \sum_{n=1}^{\infty} \frac{n}{z^n}$$

4.8.2 z transform inverse of a rational fraction: `invztrans`

`invztrans` has one or three arguments:

- a rational fraction supplied by its expression by using the variable x and x will also be the name of the used variable in the function returned by `ztrans`,
- three arguments: a rational fraction supplied by its expression, the name of the used variable to define this expression (for example the variable z), and the name of the used variable in the function returned by `invztrans` (for example n).

`invztrans` returns the z inverse transform of the rational fraction supplied as argument.

We have by definition:

if $\text{invztrans}(R_x) = ax$ we have

$$R_x = \sum_{n=0}^{\infty} \frac{a_n}{x^n}$$

if $a_n = \text{invztrans}(R_z, z, n)$ we have

$$R_z = \sum_{n=0}^{\infty} \frac{a_n}{z^n}$$

We enter:

```
invztrans(x/(x-1))
```

We get:

1

We enter:

```
invztrans(z/(z-1), z, n)
```

We get:

$$1$$

We have indeed:

$$\frac{z}{z-1} = \frac{1}{1-\frac{1}{z}} = 1 + \frac{1}{z} + \frac{1}{z^2} + \frac{1}{z^3} + \frac{1}{z^4} + \dots = \sum_{n=0}^{\infty} \frac{1}{z^n}$$

We enter:

$$\text{invztrans}(x/(x-1)^2)$$

We get:

$$x$$

We enter:

$$\text{invztrans}(z/(z-1)^2, z, n)$$

We get:

$$n$$

4.9 Solve numerical equations: nSolve

nSolve allows to solve numerically non polynomial equations:

$f(x) = 0$ for $x \in]a, b[$.

The parameters of nSolve are $f(x)=0$, $x=x_0$ where x_0 is a point of $]a, b[$.

We enter:

$$\text{nSolve}(x^2-2=0, x=1)$$

We get:

$$1.41421356237$$

We enter:

$$\text{nSolve}(x^2-2=0, x=-1)$$

We get:

$$-1.41421356237$$

4.10 Solve equations with fsolve

fsolve allows to solve numerically non polynomial equations:

$f(x) = 0$ for $x \in]a, b[$.

fsolve takes as arguments $f(x) = 0$ and $x = a..b$ or $f(x) = 0, x$ and $a..b$.

We enter:

$$\text{fsolve}(\sin(x)=0, x=0..10)$$

Or we enter:

```
fsolve(sin(x)=0,x,0..10)
```

We get:

```
[0.0, 3.14159265359, 6.28318530718, 9.42477796077]
```

We can add as last argument the value of the sample by specifying the value of `xstep` or the value of `nstep` (step of the interval $]a, b[$).

We enter:

```
fsolve(sin(x)=0,x=0..10,xstep=1 )
```

We get:

```
[0.0, 3.14159265359, 6.28318530718, 9.42477796077]
```

We enter:

```
fsolve(sin(x)=0,x=0..10,nstep=10)
```

We get:

```
[0.0, 3.14159265359, 6.28318530718, 9.42477796077]
```

4.11 Linear systems

4.11.1 Solve a linear system: `linsolve`

`linsolve` allows to solve a linear equations system where each equation is of the form $Xpr = 0$ where Xpr is an expression.

`linsolve` takes as parameters the list of equations and the list of variables.

`linsolve` returns a list which is a solution of the equations system.

`linsolve` allows to solve also a linear equations system in $\mathbb{Z}/n\mathbb{Z}$.

We enter:

```
linsolve([2*x+y+z=1,x+y+2*z=1,x+2*y+z=4],[x,y,z])
```

We get:

```
[1/-2, 5/2, 1/-2]
```

so

$$x = -\frac{1}{2}, y = \frac{5}{2}, z = -\frac{1}{2}$$

is the solution of the linear system:

$$\begin{cases} 2x + y + z = 1 \\ x + y + 2z = 1 \\ x + 2y + z = 4 \end{cases}$$

4.11.2 Gauss reduction of a matrix: `ref`

`ref` allows to solve a linear equations system that we write under matrix form:

$$A * X = B$$

The parameter of `ref` is the "expanded matrix" of the system (the one formed by the matrix A of the system and having as last column vector the second member B).

The result is a matrix $[A1, B1]$: $A1$ has two zeros below its diagonal and the solutions of:

$$A1 * X = B1$$

are the same as those of:

$$A * X = B$$

`ref` may work in $\mathbb{Z}/p\mathbb{Z}$.

For instance, be the system in \mathbb{R} and in $\mathbb{Z}/5\mathbb{Z}$ to be solved:

$$\begin{cases} 3x + y = -2 \\ 3x + 2y = 2 \end{cases}$$

To solve the system in \mathbb{R} , we enter:

```
ref([[3,1,-2],[3,2,2]])
```

We get:

```
[[1,1/3,-2/3],[0,1,4]]
```

so this means:

$y = 4$ and $x = -2$ are solutions of the system. To solve the system in $\mathbb{Z}/5\mathbb{Z}$, we enter:

```
ref([[3,1,-2],[3,2,2]]%5)
```

We get:

```
[[1%5,2%5,1%5],[0%5,1%5,-1%5]]
```

so this means:

$y = -1\%5$ and $x = 3\%5$ are solutions of the system.

Note:

When the number of columns equals the number of lines +1 `ref` does not divide by the pivot of the last column, for example, we enter:

```
ref([[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],[1,0,0,1,-a4]])
```

We get:

```
[[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],[0,0,0,0,a1-a2+a3-a4]]
```

So one learns that if $a1-a2+a3-a4$ is not null, there is no solution.

4.12 Quadratic forms

4.12.1 Matrix of a quadratic form: `q2a`

`q2a` has two arguments: a quadratic form q and the vector whose components the used variables.

`q2a` returns the matrix A associated to q .

We enter:

```
q2a(2*x*y, [x, y])
```

We get:

```
[[0,1],[1,0]]
```

4.12.2 Transform a matrix in a quadratic form: `a2q`

`a2q` has two arguments: a symmetric matrix A representing a quadratic form q and the vector whose components are the used variables.

`a2q` returns the quadratic form q .

We enter:

```
a2q([[0,1],[1,0]],[x,y])
```

We get:

$$2*x*y$$

We enter:

```
a2q([[1,2],[2,4]],[x,y])
```

We get:

$$x^2+4*x*y+4*y^2$$

4.12.3 Gauss method: `gauss`

`gauss` has two arguments: a quadratic form q and the vector whose components are the used variables.

`gauss` returns the writing of q under the form of a sum and difference of squares.

We enter:

```
gauss(2*x*y,[x,y])
```

We get:

$$(y+x)^2/2+(-(y-x)^2)/2$$

4.12.4 Gramschmidt process: `gramschmidt`

`gramschmidt` has one or two parameters:

- a matrix seen as a list of row vectors, the dot product of being the canonical scalar product, or
- a vector containing the basis of a vector subspace and a function which defines a scalar product.

`gramschmidt` gives a orthonormal basis according to this scalar product.

We enter:

```
normal(gramschmidt([[1,1,1],[0,0,1],[0,1,0]]))
```

Or we enter:

```
normal(gramschmidt([[1,1,1],[0,0,1],[0,1,0]],dot))
```

We get:

$$\begin{aligned} &[(\sqrt{3})/3, (\sqrt{3})/3, (\sqrt{3})/3], \\ &[-(\sqrt{6})/6, -(\sqrt{6})/6, (\sqrt{6})/3], \end{aligned}$$

```
[(-sqrt(2))/2, (sqrt(2))/2, 0]]
```

Example

For the polynomials of degree $< n$, we consider the dot product defined by:

$$P.Q = \int_{-1}^1 P(x).Q(x)dx$$

We enter:

```
gramschmidt([1,1+x],(p,q)->integrate(p*q,x,-1,1))
```

Or we write the function `p_scal`, we enter:

```
p_scal(p,q):=integrate(p*q,x,-1,1)
```

and we enter:

```
gramschmidt([1,1+x],p_scal)
```

We get:

```
[1/(sqrt(2)),(1+x-1)/sqrt(2/3)]
```

4.13 Conics

4.13.1 Plot of a conic: `conic`

`conic` takes as argument the expression of a conic.

`conic` plots the conic having for equation the argument equals zero.

We enter:

```
conic(2*x^2+2*x*y+2*y^2+6*x)
```

We get:

```
the plot of the ellipse of center -2+i and equation
2*x^2+2*x*y+2*y^2+6*x=0
```

Note:

Use `reduced_conic` to get the parametric equation of the conic.

4.13.2 Reduction of a conic: `reduced_conic`

`reduced_conic` takes one or two arguments: the expression of a conic and the vector whose components are the used variables if it is different from $[x,y]$.

`reduced_conic` returns a list of elements:

- the origin of the conic,
- the matrix of a basis in which the conic is reduced,
- 0 or 1 to tell whether the conic is degenerate or not,
- the reduced equation of the conic in this basis,
- a vector containing its parametric equation or its parametrics equations when the conic is multi-napped.

We enter:

```
reduced_conic(2*x^2+2*x*y+2*y^2+5*x+3,[x,y])
```

We get:

```

[[-5/3, 5/6], [[-1/(sqrt(2)), 1/(sqrt(2))], [-1/(sqrt(2)),
-1/(sqrt(2))]], 1, 3*x^2+y^2+-7/6, [[(-10+5*i)/6+(1/(sqrt(2))+
(i)/(sqrt(2)))*((sqrt(14)*cos(' t')))/6+
((i)*sqrt(42)*sin(t))/6), t, 0, 2*pi, (2*pi)/60]]]

```

The conic is not degenerate and its reduced equation is:

$$3x^2 + y^2 - \frac{7}{6} = 0$$

in the origin basis $-5/3 + 5 * i/6$ and with axis parallel to vectors $(-1, 1)$ and $(-1, -1)$.

Its parametric equation is:

$$-\frac{10 + 5 * i}{6} + \frac{1 + i}{\sqrt{2}} * \frac{\sqrt{14} * \cos(t) + i * \sqrt{42} * \sin(t)}{6}$$

and for the plot, the parameter t varies from 0 to 2π by step $t_{step}=2\pi/60$.

Note:

When the conic is degenerate in one or two line(s), each line is not supplied by its parametric equation but by the list constituted by a vector normal to the line and a point of the line.

We enter:

```
reduced_conic(x^2-y^2+3*x+y+2)
```

We get:

```

[[-3)/2, 1/2], [[1, 0], [0, 1]], 0, x^2-y^2, [[(-1+2*i)/(1-i), (1+2*i)/(1-
i)], [(-1+2*i)/(1-i), (-1)/(1-i)]]]

```

We get:

```
(2*sqrt(5*23297^2*126757^*21302293^2))/62906903119301897
```

That is to say:

$$2 * \sqrt{5}$$

We enter:

```
H1:=projection(D1,M)
```

```
length(M, F1)/length(M, H1)
```

We get:

```

(2^14*3*13*17*89*311*521*563*769*2609*
sqrt(2*3*49409^2*112249^2*126757^2*
21302293^2*568000439^2*6789838247809^2))/
(2^14*3^2*13*17*89*311*521*563*769*
2609*49409*112249*126757*21302293*568000439*6789838247809)

```

That is to say:

$$(\sqrt{6})/3$$

Chapter 5 Menu Rewrite

5.1 Collect the logarithms: `lncollect`

`lncollect` takes as argument an expression containing logarithms.

`lncollect` collects the terms in logarithms. It is better to use it on a factorized expression (by using `factor`).

We enter:

```
lncollect(ln(x+1)+ln(x-1))
```

We get:

```
ln((x+1)*(x-1))
```

We enter:

```
lncollect(exp(ln(x+1)+ln(x-1)))
```

We get:

```
(x+1)*(x-1)
```

5.2 Expand the logarithms: `lnexpand`

`lnexpand` takes as argument an expression containing logarithms.

`lnexpand` expands this expression.

We enter:

```
lnexpand(ln(3*x^2)+ln(2*x+2))
```

We get:

```
ln(3)+2*ln(x)+ln(2)+ln(x+1)
```

5.3 Linearize the exponentials: `lin`

`lin` takes as argument an expression containing exponentials.

`lin` linearizes this expression (rewrites it according to $\exp(n \cdot x)$).

Examples

– We enter:

```
lin(sinh(x)^2)
```

We get:

```
1/4*exp(2*x)+1/-2+1/4*exp(-(2*x))
```

- We enter:

```
lin((exp(x)+1)^3)
```

We get:

```
exp(3*x)+3*exp(2*x)+3*exp(x)+1
```

5.4 Transform a power in product of powers: powexpand

powexpand allows to transformer a power in a product of powers.

We enter:

```
powexpand(a^(x+y))
```

We get:

```
a^x*a^y
```

5.5 Transform the trigonometric and hyperbolic expressions in $\tan(x/2)$ and in e^x : halftan_hyp2exp

halftan_hyp2exp takes as argument a trigonometric or hyperbolic expression.

halftan_hyp2exp transforms the $\sin(x)$, $\cos(x)$ and $\tan(x)$ of the expression in terms of $\tan\left(\frac{x}{2}\right)$ and e^x .

We enter:

```
halftan_hyp2exp(tan(x)+tanh(x))
```

We get:

```
(2*tan(x/2))/((1-(tan(x/2))^2))+(((exp(x))^2-1)/((exp(x))^2+1))
```

We enter:

```
halftan_hyp2exp(sin(x)^2+cos(x)^2-sinh(x)^2+cosh(x)^2)
```

We get, after simplification with normal(ans):

```
2
```

5.6 Expand a transcendental and trigonometric expression: texpand

texpand takes as argument a transcendental and trigonometric expression.

texpand is the generalization of expexpand, lnexpand and trigexpand because it expands the transcendental and trigonometric expressions.

For example, texpand allows to transform $\ln(x^n)$ in $n \ln(x)$, e^{x^n} in e^{nx} , and $\sin(2x)$ in $2\sin(x)\cos(x)$.

- texpand takes as argument a transcendental and trigonometric expression.

Example:

Expand $\exp(x+y) + \cos(x+y) + \ln(3x^2)$.

We enter:

```
texpand(exp(x+y)+cos(x+y)+ln(3*x^2))
```

We get:

```
cos(x)*cos(y)-sin(x)*sin(y)+exp(x)*exp(y)+ln(3)+2*ln(x)
```

- `texpand` takes as argument a trigonometric expression.
`texpand` expands this expression in term of $\sin(x)$ and $\cos(x)$.

Examples

1. Expand $\cos(x + y)$.

We enter:

```
texpand(cos(x+y))
```

We get:

```
cos(x)*cos(y)-sin(x)*sin(y)
```

2. Expand $\cos(3x)$.

We enter:

```
texpand(cos(3*x))
```

We get:

```
4*(cos(x))^3-3*cos(x)
```

3. Expand $\frac{\sin(3*x) + \sin(7*x)}{\sin(5*x)}$.

We enter:

```
texpand((sin(3*x)+sin(7*x))/sin(5*x))
```

We get

```
(4*(cos(x))^2-1)*(sin(x)/(16*(cos(x))^4-12*(cos(x))^2+1))/sin(x)+(64*(cos(x))^6-80*(cos(x))^4+24*(cos(x))^2-1)*sin(x)/(16*(cos(x))^4-12*(cos(x))^2+1)/sin(x)
```

And, after simplification by entering `simplify(Ans)`, we get:

```
4*(cos(x))^2-2
```

- `texpand` takes as argument a transcendental expression.
`texpand` expands this expression.

Examples

1. Expand $\exp(x + y)$.

We enter:

```
texpand(exp(x+y))
```

We get:

```
exp(x)*exp(y)
```

2. Expand $\ln(x + y)$.

We enter:

```
texpand(log(x*y))
```

We get:

```
log(x)+log(y)
```

3. Expand $\ln(x^n)$.

We enter:

```
texpand(ln(x^n))
```

We get:

```
n*ln(x)
```

4. Expand $\ln(e^2 + \exp(2 * \ln(2)) + \exp(\ln(3) + \ln(2)))$.

We enter:

```
texpand(log(e^2)+exp(2*log(2))+exp(log(3)+log(2)))
```

We get:

```
6+3*2
```

Or we enter:

```
texpand(log(e^2)+exp(2*log(2)))+lncollect(exp(log(3)+log(2)))
```

We get:

```
12
```

5.7 Exp & Ln

5.7.1 Transform $\exp(n * \ln(x))$ in power: `exp2pow`

`exp2pow` allows to transform an expression of the form $\exp(n * \ln(x))$ into a power of x .

We enter:

```
exp2pow(exp(n*ln(x)))
```

We get:

```
x^n
```

Please note the difference with `lncollect`:

```
lncollect(exp(n*ln(x))) = exp(n*ln(x))
lncollect(exp(2*ln(x))) = exp(2*ln(x))
exp2pow(exp(2*ln(x))) = x^2
```

But:

```
lncollect(exp(ln(x)+ln(x))) = x^2
exp2pow(exp(ln(x)+ln(x))) = x^(1+1)
```

5.7.2 Transform a power into an exponential: pow2exp

pow2exp allows to transform a power into exponential.

We enter:

```
pow2exp(a^(x+y))
```

We get:

```
exp((x+y)*ln(a))
```

5.7.3 Transform the complex exponentials into sin and cos: sincos exp2trig

sincos or exp2trig takes as argument an expression containing complex exponentials.
sincos or exp2trig transforms this expression in term of $\sin(x)$ and $\cos(x)$.

We enter:

```
sincos(exp(i*x))
```

or

```
exp2trig(exp(i*x))
```

We get, if `Complex` is not checked in the CAS configuration (`Shift-CAS`):

```
cos(x)+i*sin(x)
```

We get, if `Complex` is checked in the CAS configuration:

```
exp(im(x))*(cos(re(x))+(i)*sin(re(x)))
```

We enter:

```
sincos(exp(i*x)+exp(-i*x))
```

or

```
exp2trig(exp(i*x)+exp(-i*x))
```

We get:

```
cos(x)+i*sin(x)+cos(x)-i*sin(x)
```

then we select this answer and we press simplify. We get:

```
2*cos(x)
```

5.7.4 Transform the functions hyperbolic in exponentials: `hyp2exp`

`hyp2exp` takes as argument an hyperbolic expression.

`hyp2exp` transforms the hyperbolic functions hyperbolic in exponentials WITHOUT linearizing.

We enter:

```
hyp2exp(sinh(x))
```

We get:

```
(exp(x)-1/(exp(x)))/2
```

5.7.5 Write with complex exponentials: `tsimplify`

`tsimplify` simplifies all the expressions by transforming them into complex exponentials.

We do use `tsimplify` as last resort only.

We enter:

```
tsimplify((sin(7*x)+sin(3*x))/sin(5*x))
```

We get:

```
((exp((i)*x))^4+1)/(exp((i)*x))^2
```

5.7.6 Expand the exponentials: `expexpand`

`expexpand` takes as argument an expression containing exponentials.

`expexpand` expands this expression.

We enter:

```
expexpand(exp(3*x))
```

We get:

```
exp(x)^3
```

We enter:

```
expexpand(exp(3*x)+exp(2*x+2))
```

We get:

```
exp(x)^3+exp(x)^2*exp(2)
```

5.8 Sine

5.8.1 Transform the arcsin into arccos: `asin2acos`

`asin2acos` takes as argument a trigonometric expression.

`asin2acos` transforms this expression by replacing:

$\arcsin(x)$ by $\frac{\pi}{2} - \arccos(x)$.

We enter:

$$\text{asin2acos}(\text{acos}(x) + \text{asin}(x))$$

We get after simplification:

$$\pi/2$$

5.8.2 Transform the arcsin in arctan: asin2atan

`asin2atan` takes as argument a trigonometric expression.

`asin2atan` transforms this expression by replacing:

$\arcsin(x)$ by $\arctan\left(\frac{x}{\sqrt{1-x^2}}\right)$.

We enter:

$$\text{asin2atan}(\text{asin}(x))$$

We get:

$$\text{atan}(x/\text{sqrt}(1-x^2))$$

5.8.3 Transform sin(x) in cos(x)*tan(x): sin2costan

`sin2costan` takes as argument a trigonometric expression.

`sin2costan` transforms this expression by replacing:

$\sin(x)$ by $\cos(x) * \tan(x)$.

We enter:

$$\text{sin2costan}(\sin(2*x))$$

We get:

$$\cos(2*x) * \tan(2*x)$$

5.9 Cosine

5.9.1 Transform the arccos into arcsin: acos2asin

`acos2asin` takes as argument a trigonometric expression.

`acos2asin` transforms this expression by replacing:

$\arccos(x)$ by $\frac{\pi}{2} - \arcsin(x)$.

We enter:

$$\text{acos2asin}(\text{acos}(x) + \text{asin}(x))$$

We get after simplification:

$$\pi/2$$

5.9.2 Transform the arccos into arctan: acos2atan

`acos2atan` takes as argument a trigonometric expression.

`acos2atan` transforms this expression by replacing:

$$\arccos(x) \text{ by } \frac{\pi}{2} - \arctan\left(\frac{x}{\sqrt{1-x^2}}\right).$$

We enter:

```
acos2atan(acos(x))
```

We get:

```
pi/2-atan(x/sqrt(1-x^2))
```

5.9.3 Transform $\cos(x)$ into $\sin(x)/\tan(x)$: `cos2sintan`

`cos2sintan` takes as argument a trigonometric expression.

`cos2sintan` transforms this expression by replacing:

$$\cos(x) \text{ by } \frac{\sin(x)}{\tan(x)}.$$

We enter:

```
cos2sintan(cos(2*x))
```

We get:

```
sin(2*x)/tan(2*x)
```

5.10 Tangent

5.10.1 Transform $\tan(x)$ with $\sin(2x)$ and $\cos(2x)$: `tan2sincos2`

`tan2sincos2` takes as argument a trigonometric expression.

`tan2sincos2` transforms this expression by replacing:

$$\tan(x) \text{ by } \frac{\sin(2x)}{1+\cos(2x)}.$$

We enter:

```
tan2sincos2(tan(x))
```

We get:

```
sin(2*x)/(1+cos(2*x))
```

5.10.2 Transform the \arctan into \arcsin : `atan2asin`

`atan2asin` takes as argument a trigonometric expression.

`atan2asin` transforms this expression by replacing:

$$\arctan(x) \text{ by } \arcsin\left(\frac{x}{\sqrt{1+x^2}}\right).$$

We enter:

```
atan2asin(atan(x))
```

We get:

```
asin(x/sqrt(1+x^2))
```

5.10.3 Transform the arctan into arccos: atan2acos

atan2acos takes as argument a trigonometric expression.

atan2acos transforms this expression by replacing:

$\arctan(x)$ by $\frac{\pi}{2} - \arccos\left(\frac{x}{\sqrt{1+x^2}}\right)$.

We enter:

```
atan2acos(atan(x))
```

We get:

```
pi/2-acos(x/sqrt(1+x^2))
```

5.10.4 Transform tan(x) into sin(x)/cos(x): tan2sincos

tan2sincos takes as argument a trigonometric expression.

tan2sincos transforms this expression by replacing:

$\tan(x)$ by $\frac{\sin(x)}{\cos(x)}$.

We enter:

```
tan2sincos(tan(2*x))
```

We get:

```
sin(2*x)/cos(2*x)
```

5.10.5 Transform a trigonometric expression in term of tan(x/2): halftan

halftan takes as argument a trigonometric expression.

halftan transforms the $\sin(x)$, $\cos(x)$ and $\tan(x)$ of the expression in term of $\tan\left(\frac{x}{2}\right)$.

We enter:

```
halftan(sin(x))
```

We get:

```
2*tan(x/2)/(1+tan(x/2)^2)
```

We enter:

```
halftan(sin(2*x)/(1+cos(2*x)))
```

We get:

```
2*tan(2*x/2)/((tan(2*x/2))^2+1)/(1+(1-
(tan(2*x/2))^2)/((tan(2*x/2))^2+1))
```

And, after simplification with `simplify(Ans)`, we get:

```
tan(x)
```

5.11 Trigonometry

5.11.1 Simplify by privileging sine: `trigsin`

`trigsin` takes as argument a trigonometric expression.
`trigsin` simplifies this expression using formulas:
 $\sin(x)^2 + \cos(x)^2 = 1$, $\tan(x) = \frac{\sin(x)}{\cos(x)}$ and by privileging sine.

We enter:

```
trigsin(cos(x)^2+1)
```

We get:

```
-sin(x)^2+2
```

5.11.2 Simplify by privileging cosine: `trigcos`

`trigcos` takes as argument a trigonometric expression.
`trigcos` simplifies this expression using formulas:
 $\sin(x)^2 + \cos(x)^2 = 1$, $\tan(x) = \frac{\sin(x)}{\cos(x)}$ and by privileging cosine.

We enter:

```
trigcos(sin(x)^4+2)
```

We get:

```
cos(x)^4-2*cos(x)^2+3
```

5.11.3 Transform trigonometric inverse functions to logarithms: `atrig2ln`

`atrig2ln` rewrites the expression containing trigonometric inverse functions with logarithms.

We enter:

```
atrig2ln(asin(x))
```

We get:

```
i*ln(x+sqrt(x^2-1))+pi/2
```

5.11.4 Simplify by privileging tangent: `trigtan`

`trigtan` takes as argument a trigonometric expression.
`trigtan` simplifies this expression using formulas:
 $\sin(x)^2 + \cos(x)^2 = 1$, $\tan(x) = \frac{\sin(x)}{\cos(x)}$ and by privileging tangent.

We enter:

```
trigtan(sin(x)^4+cos(x)^2+1)
```

We get:

```
((tan(x))^2/(1+(tan(x))^2))^2+1/(1+(tan(x))^2)+1
```

and after simplification with `simplify(Ans)`, we have:

$$(2*\tan(x)^4+3*\tan(x)^2+2)/(\tan(x)^4+2*\tan(x))^2+1)$$

5.11.5 Linearize a trigonometric expression: `tlin`

`tlin` takes as argument a trigonometric expression.

`tlin` linearizes this expression in term of $\sin(n.x)$ and $\cos(n.x)$.

Examples

- Linearize $\cos(x) * \cos(y)$.

We enter:

```
tlin(cos(x)*cos(y))
```

We get:

$$1/2*\cos(x-y)+1/2*\cos(x+y)$$

- Linearize $\cos(x)^3$.

We enter:

```
tlin(cos(x)^3)
```

We get:

$$3/4*\cos(x)+1/4*\cos(3*x)$$

- Linearize $4\cos(x)^2 - 2$.

We enter:

```
tlin(4*cos(x)^2-2)
```

We get:

$$2*\cos(2*x)$$

5.11.6 Shift the phase by $\frac{\pi}{2}$ in trigonometric expressions: `shift_phase`

`shift_phase` takes as argument a trigonometric expression.

`shift_phase` allows to shift the phase by $\frac{\pi}{2}$ in trigonometric expressions once the automatic simplification has been performed.

We enter:

```
shift_phase(x+sin(x))
```

We get:

$$x-\cos((\pi+2*x)/2)$$

We enter:

```
shift_phase(x+cos(x))
```

We get:

$$x--\sin((\pi+2*x)/2)$$

We enter:

```
shift_phase(x+tan(x))
```

We get:

```
x+1/(tan((pi+2*x)/2))
```

Should the expression not be evaluated (*i.e.* no automatic simplification), just quote the argument.

We enter:

```
shift_phase('sin(x+pi/2)')
```

We get:

```
-(cos(pi+x))
```

but if we enter without quoting the sine:

```
shift_phase(sin(x+pi/2))
```

We get:

```
sin((pi+2*x)/2)
```

because $\sin(x+\pi/2)$ is evaluated (*i.e.* simplified) into $\cos(x)$ before the command `shift_phase` is called and then `shift_phase(cos(x))` returns $\sin((\pi+2*x)/2)$.

Exercise

Calculate

$$\sum_{n=1}^{+\infty} \frac{\sin(n * x)}{n}$$

We enter:

```
normal(sum((sin(n*x))/n,n=1..+infinity))
```

We get:

```
-atan((sin(x))/(cos(x)-1))
```

We enter:

```
normal(shift_phase(halftan(atan(sin(x)/(-cos(x)+1))))))
```

We get:

```
pi*floor(((pi+x)/2)/pi+1/2)+(-1)/2*pi+(-1)/2*x
```

if we enter:

```
tsimplify(atan((sin(x))/(-cos(x)+1)))
```

Because `tsimplify` is not rigorous with respect to $2k\pi$, we get:

```
-1/2*pi-1/2*x
```

5.11.7 Collect the sine and cosine of a same angle: `tcollect`

`tcollect` takes as argument a trigonometric expression.

`tcollect` linearizes this expression in term of $\sin(n \cdot x)$ and $\cos(n \cdot x)$ then collects the sine and cosine of same angle.

We enter:

```
tcollect(sin(x)+cos(x))
```

We get:

```
sqrt(2)*cos(x-pi/4)
```

We enter:

```
tcollect(2*sin(x)*cos(x)+cos(2*x))
```

We get:

```
sqrt(2)*cos(2*x-pi/4)
```

5.11.8 Expand a trigonometric expression: `trigexpand`

`trigexpand` takes as argument a trigonometric expression.

`trigexpand` expands this expression in term of $\sin(x)$ and $\cos(x)$.

We enter:

```
trigexpand(cos(x+y))
```

We get:

```
cos(x)*cos(y)-sin(x)*sin(y)
```

5.11.9 Transform a trigonometric expression into complex exponentials: `trig2exp`

`trig2exp` takes as argument a trigonometric expression.

`trig2exp` transforms the trigonometric functions into complex exponentials WITHOUT linearizing.

We enter:

```
trig2exp(tan(x))
```

We get:

```
((exp((i)*x))^2-1)/((i)*((exp((i)*x))^2+1))
```

We enter:

```
trig2exp(sin(x))
```

We get:

```
(exp((i)*x)-1/(exp((i)*x)))/(2*i)
```

Chapter 6 Menu Integer

6.1 Test of parity: `even`

`even` takes as argument an integer `n`.
`even` returns 1 if `n` is even and 0 if `n` is odd.

We enter:

```
even(148)
```

We get:

```
1
```

We enter:

```
even(149)
```

We get:

```
0
```

6.2 Test of non parity: `odd`

`odd` takes as argument an integer `n`.
`odd` returns 1 if `n` is odd and 0 if `n` is even.

We enter:

```
odd(148)
```

We get:

```
0
```

We enter:

```
odd(149)
```

We get:

```
1
```

6.3 Divisors of an integer: `idivis`

`idivis` returns the vector whose components are the divisors of an integer.

We enter:

```
idivis(45)
```

We get:

```
[1, 3, 9, 5, 15, 45]
```

6.4 Prime factors decomposition of an integer: `ifactor`

`ifactor` returns the prime factors decomposition of an integer.

We enter:

```
ifactor(20!)
```

We get:

```
2^18*3^8*5^4*7^2*11*13*17*19
```

6.5 List of prime factors and their multiplicity: `ifactors`

`ifactors` returns the list of prime factors of an integer with their multiplicity.

We enter:

```
ifactors(45)
```

We get:

```
[3, 2, 5, 1]
```

indeed $45 = 3^2 * 5^1$

6.6 GCD of one or several integers: `gcd`

`gcd` returns the greatest common divisor of one or several integers (see 7.12 for the GCD of polynomials).

We enter:

```
gcd(45, 10)
```

We get:

```
5
```

We enter:

```
gcd(40, 12, 16, 24)
```

We get:

```
4
```

6.6.1 GCD of a list of integers: `lgcd`

`lgcd` designates the GCD of the elements of a list of integers (or of a list of polynomials).

We enter:

```
lgcd([18,15,21,36])
```

We get:

```
3
```

6.7 LCM of one or several integers: lcm

`lcm` returns the lowest common multiple of two or several integers.

We enter:

```
lcm(45,10)
```

We get:

```
90
```

We enter:

```
lcm(45,10,25,30)
```

We get:

```
450
```

6.7.1 Bezout identity: iegcd

`iegcd(a,b)` designates the extended GCD (Bezout identity) of two integers.

`iegcd(a,b)` returns $[u,v,d]$ that make $au+bv=d$ and such as $d=\text{gcd}(a,b)$.

We enter:

```
iegcd(48,30)
```

We get:

```
[2,-3,6]
```

Indeed:

$$2 \cdot 48 + (-3) \cdot 30 = 6$$

6.7.2 Solve $au + bv = c$ in \mathbb{Z} : iabcuv

`iabcuv(a,b,c)` gives $[u,v]$ that make $au+bv=c$.

Of course, c has to be a multiple of $\text{gcd}(a,b)$ to get a solution.

We enter:

```
iabcuv(48,30,18)
```

We get:

```
[6,-9]
```

6.8 Primality

6.8.1 Check whether a number is prime: `isPrime` `isprime`

`isPrime(n)` or `isprime` returns `true` if `n` is prime and `false` otherwise.

We enter:

```
isPrime(1234567)
```

We get:

```
false
```

We enter:

```
isPrime(1234547)
```

We get:

```
true
```

6.8.2 The N-th prime number: `ithprime`

`ithprime(n)` returns the N-th prime number.

We enter:

```
ithprime(10)
```

We get:

```
29
```

Indeed, the ten first prime numbers are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.

We enter:

```
ithprime(100)
```

We get:

```
541
```

541 is then the 100-nth prime number.

6.8.3 `nextprime`

`nextprime(n)` returns the prime number `p` which is just after `n` ($p > n$).

We enter:

```
nextprime(11)
```

We get:

```
13
```

We enter:

```
nextprime(1234567)
```

We get:

```
1234577
```

6.8.4 prevprime

`prevprime(n)` returns the prime number p which is just before n ($p < n$).

We enter:

```
prevprime(11)
```

We get:

```
7
```

We enter:

```
prevprime(1234567)
```

We get:

```
1234547
```

6.8.5 Euler's totient: euler

`euler(n)` returns the cardinal of the set of numbers lower than n which are relatively prime to n .
`euler(n)` designates then the Euler's totient of the integer n .

We enter:

```
euler(18)
```

We get:

```
6
```

Indeed, the set:

$E = \{5, 7, 11, 13, 15, 17\}$ corresponds to numbers lower than 18 which are prime to 18, and E takes as cardinal 6. With the euler function, we have the generalization of the Fermat's theorem (which says that "if n is prime and if a is prime to n then $a^{n-1} = 1 \pmod n$."

The generalization is (because if n is prime, $euler(n) = n - 1$):
 $a^{euler(n)} = 1 \pmod n$ if a and n are prime to each other.

We enter:

```
powmod(5, 6, 18)
```

We get:

```
1
```

6.8.6 Legendre symbol: legendre_symbol

When n is prime, we define the Legendre symbol of a written $\left(\frac{a}{n}\right)$ by:

$$\left(\frac{a}{n}\right) = \begin{cases} 0 & \text{if } a = 0 \pmod n \\ 1 & \text{if } a \neq 0 \pmod n \text{ and if } a = b^2 \pmod n \\ -1 & \text{if } a \neq 0 \pmod n \text{ and if } a \neq b^2 \pmod n \end{cases}$$

Some properties

– if n is prime:

$$a^{\frac{n-1}{2}} = \left(\frac{a}{n}\right) \pmod n$$

– $\left(\frac{p}{q}\right) \cdot \left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}} \cdot (-1)^{\frac{q-1}{2}}$ if p and q are odd and positive

$$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$$

$$\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$$

`legendre_symbol` has two parameters a and n and returns the Legendre symbol $\left(\frac{a}{n}\right)$.

We enter:

```
legendre_symbol(26,17)
```

We get:

1

We enter:

```
legendre_symbol(27,17)
```

We get:

-1

We enter:

```
legendre_symbol(34,17)
```

We get:

0

6.8.7 Jacobi symbol: `jacobi_symbol`

When n is not prime, we define the Jacobi symbol of a , also written $\left(\frac{a}{n}\right)$, from the Legendre symbol and the decomposition of n in prime factor.

Let

$$n = p_1^{\alpha_1} \cdot \dots \cdot p_k^{\alpha_k}$$

where p_j is prime and α_j is an integer for $j = 1..k$. The Jacobi symbol of a is defined by:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \cdot \dots \cdot \left(\frac{a}{p_k}\right)^{\alpha_k}$$

`jacobi_symbol` has two parameters a and n and returns the symbol of Jacobi $\left(\frac{a}{n}\right)$.

We enter:

```
jacobi_symbol(25,12)
```

We get:

1

We enter:

```
jacobi_symbol(35,12)
```

We get:

```
-1
```

We enter:

```
jacobi_symbol(33,12)
```

We get:

```
0
```

6.8.8 Solve $a^2 + ab^2 = p$ in \mathbb{Z} : pa2b2

pa2b2 decomposes a prime integer p , congruent to 1 modulus 4, in the sum of two squares: $p = a^2 + b^2$.

The result is supplied as a list.

We enter:

```
pa2b2(17)
```

We get:

```
[4,1]
```

indeed $17 = 4^2 + 1^2$

6.9 Division

6.9.1 Quotient of the Euclidean division: iquo

iquo(a, b) returns the quotient of the Euclidean division of a by b when a and b are integers.

We enter:

```
iquo(45,10)
```

We get:

```
4
```

indeed $45 = 4 * 10 + 5$

6.9.2 Remainder of the Euclidean division: irem

irem(a, b) returns the remainder of the Euclidean division of a by b when a and b are integers.

We enter:

```
irem(45,10)
```

We get:

indeed $45 = 4 * 10 + 5$

6.9.3 Quotient and remainder of the Euclidean division: `iquorem`

`iquorem` gives the list of the quotient q and the integer remainder r of the Euclidean division of the integers a and b supplied as argument ($a = b * q + r$ with $0 \leq r < b$).

We enter:

```
iquorem(148,5)
```

We get:

```
[29,3]
```

6.9.4 Chinese remainder for integers: `ichinrem`

`ichinrem([a,n],[b,p])` returns the vector $[c, \text{lcm}(p, q)]$ formed of two integers. The first number c is such as

$$\forall k \in \mathbb{Z}, \quad d = c + k \times \text{lcm}(p, q)$$

l checks

$$d = a \pmod{p}, \quad d = b \pmod{q}$$

If n and p are prime then there is always a solution and $q=n*p$

Example:

Find the solutions of:

$$\begin{cases} x = 3 \pmod{5} \\ x = 9 \pmod{13} \end{cases}$$

We enter:

```
ichinrem([3,5],[9,13])
```

Or we enter:

```
ichinrem({3,5},{9,13})
```

We get:

```
[-17,65]
```

So the solutions are $x = -17 + k * 65$ with $k \in \mathbb{Z}$.

We have indeed $-17 = -5 * 4 + 3 = 3 \pmod{5}$ and $-17 = -2 * 13 + 9 = 9 \pmod{13}$

`ichinrem` returns the Chinese remainder for integers.

We enter:

```
ichinrem({2,7},{3,5})
```

Or we enter:

```
ichinrem([2,7],[3,5])
```

We get:

```
[-12,35]
```

We do have $-12 + 35k = 2 \pmod{7}$ and $-12 + 35k = 3 \pmod{5}$ for $k \in \mathbb{Z}$

6.9.5 Calculation of $a^n \bmod p$: `powmod`

`powmod(a, n, p)` returns a^n modulus p with the method of the fast exponentiation.

We enter:

```
powmod(5, 21, 13)
```

We get:

```
5
```

$5^2 = 25 = -1 \bmod 13$ then $5^{21} = 5 * 5^{20} = 5 \bmod 13$

We enter:

```
powmod(37, 25, 11)
```

We get:

```
1
```

indeed $37 = 4 \bmod 11$ and $4^5 = 4^2 * 4^2 * 4 = 25 * 4 = 1 \bmod 11$

6.10 Modular calculus in $\mathbb{Z}/p\mathbb{Z}$ or in $\mathbb{Z}/p\mathbb{Z}[x]$

We can perform calculations modulus p that is to say in $\mathbb{Z}/p\mathbb{Z}$ or in $\mathbb{Z}/p\mathbb{Z}[x]$.
The numbers n of $\mathbb{Z}/p\mathbb{Z}$ are written $n\%p$.

Examples of notation

- an integer n of $\mathbb{Z}/13\mathbb{Z}$
`n:=12%13.`
- a vector v of coordinates in $\mathbb{Z}/13\mathbb{Z}$
`V:=[1,2,3]%13` or `V:=[1%13,2%13,3%13].`
- a matrix A of coefficients in $\mathbb{Z}/13\mathbb{Z}$
`A:=[[1,2,3],[2,3,4]]%13`
or
`A:=[[1%13,2%13,3%13],[2%13,3%13,4%13]].`
- a polynomial A of $\mathbb{Z}/13\mathbb{Z}[x]$ in symbolic notation
`A:=(2*x^2+3*x-1)%13`
or
`A:=2%13*x^2+3%13*x-1%13.`
- a polynomial A of $\mathbb{Z}/13\mathbb{Z}[x]$ represented with a list
`A:=poly1[1,2,3]%13`
or
`A:=poly1[1%13,2%13,3%13].`

To transform an object o of modular coefficients into an object of integer coefficients, press:

`o % 0`. For instance, if we enter `o:=4%7` then `o%0`, we get

```
-3.
```

Notes

- For some commands in $\mathbb{Z}/p\mathbb{Z}$ or in $\mathbb{Z}/p\mathbb{Z}[x]$, we have to choose a number p which is prime.
- The chosen representation is the symmetrical representation:
`11%13 = -2%13`

6.10.1 Expand and factorise: `normal`

`normal` takes as argument a polynomial expression.

`normal` expands and factors this expression in $\mathbb{Z}/p\mathbb{Z}[x]$.

We enter:

```
normal(((2*x^2+12)*(5*x-4))%13)
```

We get:

```
(-3%13)*x^3+(5%13)*x^2+(-5%13)*x+4%13
```

6.10.2 Addition in $\mathbb{Z}/p\mathbb{Z}$ or in $\mathbb{Z}/p\mathbb{Z}[x]$: +

To perform an addition in $\mathbb{Z}/p\mathbb{Z}$, we use the usual + and, for polynomials of $\mathbb{Z}/p\mathbb{Z}[x]$, we use the usual + and the command `normal` to simplify.

For the integers in $\mathbb{Z}/p\mathbb{Z}$, we enter:

```
3%13+10%13
```

We get:

```
0%13
```

For the polynomials with coefficients in $\mathbb{Z}/p\mathbb{Z}$, we enter:

```
normal(11%13*x+5%13+8%13*x+6%13)
```

or else

```
normal((11*x+5)%13+(8*x+6)%13)
```

We get:

```
(6%13)*x+-2%13
```

6.10.3 Substraction in $\mathbb{Z}/p\mathbb{Z}$ or in $\mathbb{Z}/p\mathbb{Z}[x]$: -

To perform a subtraction in $\mathbb{Z}/p\mathbb{Z}$, we use the usual - and, for polynomials of $\mathbb{Z}/p\mathbb{Z}[x]$, we use the usual - and the command `normal` to simplify.

For the integers in $\mathbb{Z}/p\mathbb{Z}$, we enter:

```
31%13-10%13
```

We get:

```
-5%13
```

For the polynomials with coefficients in $\mathbb{Z}/p\mathbb{Z}$, we enter:

```
normal(11%13*x+5%13-8%13*x+6%13)
```

or else

```
normal((11*x+5)%13-(8*x+6)%13)
```

We get:

```
(3%13)*x+-1%13
```

6.10.4 Multiplication in $\mathbb{Z}/p\mathbb{Z}$ or $\mathbb{Z}/p\mathbb{Z}[x]$: *

To perform a multiplication in $\mathbb{Z}/p\mathbb{Z}$, we use the usual * and, for polynomials of $\mathbb{Z}/p\mathbb{Z}[x]$, we use the usual * then the command `normal` to simplify.

For the integers in $\mathbb{Z}/p\mathbb{Z}$, we enter:

```
31% 13*10% 13
```

We get:

```
-2% 13
```

For the polynomials with coefficients in $\mathbb{Z}/p\mathbb{Z}$, we enter:

```
normal((11% 13*x+5% 13)*(8% 13*x+6% 13))
```

or else we enter:

```
normal((11*x+5)% 13*(8*x+6)% 13)
```

We get:

```
(-3% 13)*x^2+(2% 13)*x+4% 13
```

6.10.5 Quotient: quo

`quo` takes as arguments two polynomials A and B with coefficients in $\mathbb{Z}/p\mathbb{Z}$. A and B may be supplied by a symbolic polynomial expression (of x or of the name of the variable supplied as third argument) or by the list of their coefficients.

`quo` returns the quotient of the Euclidean division of A by B in $\mathbb{Z}/p\mathbb{Z}[x]$.

We enter:

```
quo((x^3+x^2+1)% 13,(2*x^2+4)% 13)
```

Or we enter:

```
quo((x^3+x^2+1),2*x^2+4)% 13)
```

We get:

```
(-6% 13)*x+-6% 13
```

indeed $x^3 + x^2 + 1 = (2x^2 + 4)\left(\frac{x+1}{2}\right) + \frac{5x-4}{4}$
and $-3 * 4 = -6 * 2 = 1 \pmod{13}$

6.10.6 Remainder: rem

`rem` takes as arguments two polynomials A and B with coefficients in $\mathbb{Z}/p\mathbb{Z}$. A and B may be supplied by a symbolic polynomial expression (of x or of the name of the variable supplied as third argument) or by the list of their coefficients.

`rem` returns the remainder of the Euclidean division of A by B in $\mathbb{Z}/p\mathbb{Z}[x]$.

We enter:

```
rem((x^3+x^2+1)% 13,(2*x^2+4)% 13)
```

Or we enter:

```
rem((x^3+x^2+1,2*x^2+4)% 13)
```

We get:

```
(-2% 13)*x+-1% 13
```

indeed $x^3 + x^2 + 1 = (2x^2 + 4)\left(\frac{x+1}{2}\right) + \frac{5x-4}{4}$
and $-3 * 4 = -6 * 2 = 1 \pmod{13}$

6.10.7 Quotient and remainder: quorem

`quorem` takes as arguments two polynomials A and B with coefficients in $\mathbb{Z}/p\mathbb{Z}$.

A and B may be supplied by a symbolic polynomial expression (of x or of the name of variable supplied as third argument) or by the list of their coefficients.

`quorem` returns the list of the quotient and the remainder of the Euclidean division of A by B in $\mathbb{Z}/p\mathbb{Z}[x]$ (see also 6.9.3 and 6.11.4).

We enter:

```
quorem(5% 13,2% 13)
```

Or we enter:

```
quorem((5,2)% 13)
```

and because $2 * -4 = 5 - 13$

We get:

```
[-4% 13,0]
```

We enter:

```
quorem((x^3+x^2+1)% 13,(2*x^2+4)% 13)
```

Or we enter:

```
quorem((x^3+x^2+1,2*x^2+4)% 13)
```

because $x^3 + x^2 + 1 = (2x^2 + 4)\left(\frac{x+1}{2}\right) + \frac{5x-4}{4}$

and $-3 * 4 = -6 * 2 = 1 \pmod{13}$

We get:

```
[(-6% 13)*x+-6% 13,(-2% 13)*x+-1% 13]
```

6.10.8 Division in $\mathbb{Z}/p\mathbb{Z}$ or $\mathbb{Z}/p\mathbb{Z}[x]$: /

`/` divides two integers in $\mathbb{Z}/p\mathbb{Z}$, or divides two polynomials A and B in $\mathbb{Z}/p\mathbb{Z}[x]$.

For the polynomials, the result is the rational fraction $\frac{A}{B}$ simplified in $\mathbb{Z}/p\mathbb{Z}[x]$.

For integers in $\mathbb{Z}/p\mathbb{Z}$, we enter:

```
5% 13/2% 13
```

We get:

```
-4% 13
```

because 2 is invertible in $\mathbb{Z}/13\mathbb{Z}$.

For the polynomials with coefficients in $\mathbb{Z}/p\mathbb{Z}$.

We enter:

$$(2*x^2+5) \% 13 / (5*x^2+2*x-3) \% 13$$

We get:

$$((6 \% 13) * x + 1 \% 13) / ((2 \% 13) * x + 2 \% 13)$$

6.10.9 Power in $\mathbb{Z}/p\mathbb{Z}$ or $\mathbb{Z}/p\mathbb{Z}[x]$: \wedge

To calculate a at the power n in $\mathbb{Z}/p\mathbb{Z}$ we use the operator \wedge .

We enter:

$$(5 \% 13) \wedge 2$$

We get:

$$-1 \% 13$$

To calculate A at the power n in $\mathbb{Z}/p\mathbb{Z}[x]$ we use the operator \wedge and the command `normal`.

We enter:

$$\text{normal}(((2*x+1) \% 13) \wedge 5)$$

We get:

$$(6 \% 13) * x^5 + (2 \% 13) * x^4 + (2 \% 13) * x^3 + (1 \% 13) * x^2 + (-3 \% 13) * x + 1 \% 13$$

because:

$$10 = -3 \pmod{13} \quad 40 = 1 \pmod{13} \quad 80 = 2 \pmod{13} \quad 32 = 6 \pmod{13}.$$

6.10.10 Calculation of $a^n \bmod p$ or of $A(x)^n \bmod \mathfrak{f}(x), p$: `powmod`

- To calculate in $[0; p-1] a^n \bmod p$ we use the command `powmod` or `powermod` with as argument a, n, p .

We enter:

$$\text{powmod}(5, 21, 13)$$

We get:

$$5$$

We enter:

$$\text{powmod}(5, 21, 8)$$

We get:

$$5$$

- To calculate $A(x)^n \bmod \mathfrak{f}(x), p$ with as a result a polynomial with coefficients in \mathbb{Z} (which will be symmetrical remainders of division by p), we use the command `powmod` or `powermod` with as argument $A(x), n, p, P(x)$.

We enter:

```
powmod(x+1, 17, 5, x^4+x+1)
```

We get:

```
-x^3-x^2
```

We have indeed:

```
rem((x+1)^17, x^4+x+1)
```

which returns:

```
29144*x^3+36519*x^2+12270*x-4185
```

and

```
(29144*x^3+36519*x^2+12270*x-4185)% 5
```

which returns:

```
(-1 % 5)*x^3+(-1 % 5)*x^2
```

and

```
((-1 % 5)*x^3+(-1 % 5)*x^2)% 0
```

which returns:

```
-x^3-x^2
```

Note (cf section 6.10.9)

If we can calculate a power in $\mathbb{Z}/p\mathbb{Z}$ we enter for example:

```
(5% 13)^21
```

We get:

```
5% 13
```

We enter:

```
(5% 8)^21
```

We get:

```
-3% 8
```

6.10.11 Inverse in $\mathbb{Z}/p\mathbb{Z}$: `inv` or `/`

We calculate the inverse of an integer n in $\mathbb{Z}/p\mathbb{Z}$ by entering `1/n% p` or `inv(n%p)` or `inverse(n%p)`.

We enter:

```
inv(3% 13)
```

We get:

```
-4% 13
```

Indeed: $3 \times -4 = -12 = 1 \pmod{13}$

6.10.12 Transform an integer into its fraction modulus p : `fracmod`

`fracmod` has two arguments, an integer n (or an integer expression) and an integer p .
`fracmod` returns a fraction a/b such as:

$$-\frac{\sqrt{p}}{2} < a \leq \frac{\sqrt{p}}{2}, 0 \leq b < \frac{\sqrt{p}}{2}, n \times b = a \pmod{p}$$

In other words $n = \frac{a}{b} \pmod{p}$.

We enter:

```
fracmod(3,13)
```

We get:

```
-1/4
```

Indeed: $3 * -4 = -12 = 1 \pmod{13}$ then $3 = -1/4 \pmod{13}$.

We enter:

```
fracmod(13,121)
```

We get:

```
-4/9
```

Indeed: $13 \times -9 = -117 = 4 \pmod{121}$ then $13 = -4/9 \pmod{13}$.

6.10.13 GCD in $\mathbb{Z}/p\mathbb{Z}[x]$: `gcd`

When `gcd` has two polynomials with coefficients in $\mathbb{Z}/p\mathbb{Z}$ as arguments (p must be prime), `gcd` returns the GCD of the two polynomials in $\mathbb{Z}/p\mathbb{Z}[x]$ (see also 7.12 for polynomials with non modular coefficients).

We enter:

```
gcd((2*x^2+5)%13,(5*x^2+2*x-3)%13)
```

We get:

```
(-4%13)*x+5%13
```

We enter:

```
gcd(x^2+2*x+1,x^2-1) mod 5
```

We get:

```
1
```

but if we enter:

```
gcd((x^2+2*x+1,x^2-1) mod 5)
```

`gcd` is calculated in $\mathbb{Z}[x]$ then the modular calculus is performed, we get:

```
x%5
```

6.10.14 Factorization in $\mathbb{Z}/p\mathbb{Z}[x]$: `factor`

`factor` takes as argument a polynomial with coefficients in $\mathbb{Z}/p\mathbb{Z}$.
`factor` factors this polynomial in $\mathbb{Z}/p\mathbb{Z}[x]$ (p must be prime).

We enter:

```
factor((-3*x^3+5*x^2-5*x+4)% 13)
```

We get:

```
((1% 13)*x+-6% 13)*((-3% 13)*x^2+-5% 13)
```

6.10.15 Determinant of a matrix of $\mathbb{Z}/p\mathbb{Z}$: `det`

`det` takes as argument a matrix A with coefficients in $\mathbb{Z}/p\mathbb{Z}$.
`det` returns the determinant of this matrix A .

We enter:

```
det([[1,2,9]% 13,[3,10,0]% 13,[3,11,1]% 13])
```

Or we enter:

```
det([[1,2,9],[3,10,0],[3,11,1]]% 13)
```

We get:

```
5% 13
```

thus, in $\mathbb{Z}/13\mathbb{Z}$, the determinant of the matrix $A = [[1, 2, 9], [3, 10, 0], [3, 11, 1]]$ is $5\% 13$ (on a $\det(A) = 31$).

6.10.16 Inverse of a matrix of $\mathbb{Z}/p\mathbb{Z}$: `inv`

`inverse` (or `inv`) takes as argument a matrix A with coefficients in $\mathbb{Z}/p\mathbb{Z}$.
`inv` returns the inverse of the matrix A in $\mathbb{Z}/p\mathbb{Z}$.

We enter:

```
inv([[1,2,9]% 13,[3,10,0]% 13,[3,11,1]% 13])
```

Or we enter:

```
inv([[1,2,9],[3,10,0],[3,11,1]]% 13)
```

We get:

```
[[2% 13,-4% 13,-5% 13],[2% 13,0% 13,-5% 13],[-2%13,-1% 13,6% 13]]
```

It is the inverse of the matrix $A = [[1, 2, 9], [3, 10, 0], [3, 11, 1]]$ in $\mathbb{Z}/13\mathbb{Z}$.

6.10.17 Solve a linear system of $\mathbb{Z}/p\mathbb{Z}$: `rref`

`rref` allows to solve, in $\mathbb{Z}/p\mathbb{Z}$, a linear equation system of the form: $Ax = B$ (see also 20.9).
The argument is a matrix formed by A combined with B as the last column vector. The result is a matrix formed of A_1 and B_1 where A_1 has two zeros from either side of the diagonal, and where the system $A_1x = B_1$ is equivalent to $Ax = B$.

Solve in $\mathbb{Z}/13\mathbb{Z}$

$$\begin{cases} x + 2 \cdot y = 9 \\ 3 \cdot x + 10 \cdot y = 0 \end{cases}$$

We enter:

```
rref([[1, 2, 9]% 13, [3,10,0]% 13])
```

Or we enter:

```
rref([[1, 2, 9], [3,10,0]])%13
```

We get:

```
[[1% 13, 0% 13, 3% 13], [0% 13, 1% 13, 3% 13]]
```

which means that $x=3\% 13$ and $y=3\% 13$.

6.10.18 Creation of a Galois field: GF

In its simplest form, GF takes as arguments a prime number p and an integer $n > 1$ or the power of a prime number p^n and an optional argument which is the name of variable chosen for the generator of the field (the variable must be purged first).

GF creates a Galois field of characteristic p and having p^n elements, the elements of the field are then 0 and the powers from 0 to $p^n - 2$ of the generator. The field itself is stored in a free variable (by default K , this variable is displayed by the system, at the same time as the names of the generator and the free variable, by default k , used to represent the elements of the field such as the quotient $\mathbb{Z}/p\mathbb{Z}[k]/P(k)$ where P is a irreducible polynomial and primitive).

For instance:

- $\text{GF}(3, 5)$ or $\text{GF}(3^5)$ creates a field having 3^5 elements whose generator is g (or h , ... if g is assigned). We can create an element of the field by taking a polynomial in term of g , for example $g^{10} + 5g + 1$.
- $\text{GF}(2, 8, a)$ creates a field having 2^8 elements, and uses the variable a to designate the generator (Warning, do `purge(a)` first if necessary).
- The command `pmin` allows knowing the minimal polynomial of an element of the field.

We can then create polynomials or matrices having coefficients in the field, and handle them with the usual instructions `+`, `-`, `*`, `/`, `inv`, `sqrt`, `quo`, `rem`, `quorem`, `diff`, `factor`, `gcd`, `egcd`, ... for example:

- `GF(3, 5, b); A:=[[1,b],[b,1]]; inv(A)` returns the inverse of a matrix with coefficients in the field of 3^5 elements
- `GF(5, 3, c); p:=x^2-c-1; factor(p)` factors the polynomial p as polynomial with coefficients in the field at 5^3 elements, we deduce from it a value of square root of $c + 1$.
- `p:=randpoly(x, 5, g); q:=diff(p); gcd(p, q)` generates a polynomial with random coefficients, then returns its derivative and the GCD, which allows to know if p has multiples roots.

There are still limitations due to the incomplete implementation of some algorithms (for example factorization with several variables when the polynomial is not unitary).

In its most comprehensive form (but more difficult to handle and less legible), the elements of this field and the field itself are represented by $\text{GF}(\dots)$ where \dots is a sequence composed of:

- the characteristic p ($px = 0$),
- the minimal irreducible polynomial (primitive if created by the CAS) generating an ideal I in $\mathbb{Z}/p\mathbb{Z}[X]$, the Galois field is then the quotient of $\mathbb{Z}/p\mathbb{Z}[X]$ by I ,
- the name of the variable of the polynomial, by default x ,
- a polynomial (a remainder modulus the minimal polynomial) to designate an element of the field (These elements have an additive representation) or `undef` to designate the whole field which is the quotient of polynomials with coefficients in $\mathbb{Z}/p\mathbb{Z}$ by I .

Usually, we give a name to the created field (for example $G := \text{GF}(p, n)$), in order to build a particular element of the group from a polynomial of $\mathbb{Z}/p\mathbb{Z}[X]$, by writing for example $G(x^3+x)$. Note that $G(x)$ is a generator of the multiplicative group G^* when the minimal polynomial is generated by the CAS.

We enter:

```
G:=GF(2,8)
```

We get (for example):

```
GF(2, k^8-k^7-k^6-k-1, k, undef)
```

The field G has $2^8 = 256$ elements and $g = G(k)$ generates the multiplicative group of this field $(\{1, g, g^2, \dots, g^{254}\})$.

We enter:

```
K(k^9)
```

We get:

```
g^6+g^2+1)
```

We enter:

```
K(k)^255
```

We get

```
1
```

As you notice on the previous examples, when we work with the same field, the answers content redundant informations. This is why the definition of a field may have a third argument: the name of the generator or a list containing two or three names of formal variables, (the name of the undetermined of the irreducible polynomial and the name of the Galois field that has to be quoted so that these variables are not evaluated as well as the name of the generator). This allows to get a more compact display of the elements of the field.

We enter:

```
G:=GF(2,2,['w','G'])::G(w^2)
```

We get:

```
Done, G(w+1)
```

We enter:

```
G(w^3)
```

We get:

```
G(1)
```

The elements of $\text{GF}(2, 2)$ are then: $0, 1, w, w^2=w+1$.

We can then tell which irreducible polynomial we wish to use, by mentioning it as second parameter (instead of n), for example:

```
G:=GF(2, w^8+w^6+w^3+w^2+1, ['w','G'])
```

If we give a non primitive irreducible polynomial, the calculator tells it and proposes a replacement by a primitive polynomial, for example:

```
G:=GF(2,w^8+w^7+w^5+w+1,['w','G'])
```

We get:

```
G:=GF(2,w^8-w^6-w^3-w^2-1,['w','G'],undef)
```

6.10.19 Factorization of a polynomial with coefficients in a Galois field: `factor`

We can factorize a polynomial with coefficients in a Galois field with `factor`.

By example, to get $G = \mathbb{F}_4$, we enter:

```
GF(2,2,a)
```

We get:

```
GF(2,k^2+k+1,[k,K,a],undef)
```

By example, we enter:

```
factor(a^2*x^2+1)
```

We get:

```
(a+1)*(x+a+1)^2
```

6.11 Arithmetic of polynomials

Polynomials are represented by expressions or by the list of their coefficients listed with decreasing powers. In the first case the variable used by default is x . For the polynomials with coefficients in $\mathbb{Z}/n\mathbb{Z}$, apply `% n` to the expression or to each coefficient of the list.

6.11.1 List of divisors of a polynomial: `divis`

`divis` takes as argument a symbolic polynomial (or a list of polynomials) and returns the list of divisors.

We enter:

```
divis(x^2-1)
```

We get:

```
[1,x-1,x+1,(x-1)*(x+1)]
```

We enter:

```
divis(t^2-1)
```

We get:

```
[1,t-1,t+1,(t-1)*(t+1)]
```

We enter:

```
divis(x^4-1)
```

Or we enter:

```
divis(poly2symb([1,0,0,0,-1],x))
```

We get:

```
[1,x^2+1,x+1,(x^2+1)*(x+1),x-1,(x^2+1)*(x-1),(x+1)*(x-1),
(x^2+1)*(x+1)*(x-1)]
```

We enter:

```
divis([t^2,x^2-1])
```

We get:

```
[[1,t,t^2],[1,x+1,x-1,(x+1)*(x-1)]]
```

6.11.2 Euclidean quotient of two polynomials: quo

`quo` gives the quotient of the Euclidean division of polynomials (division by decreasing power order). We can enter the polynomials either by the list of their coefficients by decreasing power order, either under symbolic forms, and in this case the variable must be added as third argument (by default the variable is x).

We enter:

```
quo(x^2+2x+1,x+3)
```

We get:

```
x-1
```

We enter:

```
quo(t^2+2t+1,t+3,t)
```

We get:

```
t-1
```

or we enter:

```
quo([1,2,1],[1,3])
```

We get:

```
[] 1,-1 []
```

that is to say the polynomial `poly1[1,-1]`.

To get the quotient of $x^3 + 2x + 4$ by $x^2 + x + 2$, we enter:

```
quo(x^3+2x+4,x^2+x+2)
```

We get:

```
x-1
```

Or we enter:

```
quo([1, 0, 2, 4], [1, 1, 2])
```

We get:

```
[ ] 1, -1 [ ]
```

that is to say the polynomial `poly1[1, -1]` or the polynomial $x-1$.

We enter:

```
quo(t^3+2t+4, t^2+t+2, t)
```

We get:

```
t-1
```

If we do not put the variable t as last argument, we enter:

```
quo(t^3+2t+4, t^2+t+2)
```

We get:

```
(t^3+2*t+4) / (t^2+t+2)
```

6.11.3 Euclidean remainder of two polynomials: `rem`

`rem` gives the remainder of the Euclidean division of two polynomials (division by decreasing power order).

We can enter the polynomials either by the list of their coefficients by decreasing power order, either under symbolic forms, and in this case the variable must be added as third argument (by default the variable is x).

We enter:

```
rem(x^3-1, x^2-1)
```

We get:

```
x-1
```

We enter:

```
rem(t^3-1, t^2-1, t)
```

We get:

```
t-1
```

We enter:

```
rem(x^2+2x+1, x+3)
```

Or we enter:

```
rem(t^2+2t+1, t+3, t)
```

We get:

```
4
```

or we enter:

```
rem([1, 2, 1], [1, 3])
```

We get:

```
[] 4
```

that is to say the polynomial `poly1[4]` or else the polynomial 4.
To get the remainder of $x^3 + 2x + 4$ by $x^2 + x + 2$:

```
rem(x^3+2x+4, x^2+x+2)
```

We get:

```
x+6
```

Or we enter:

```
rem([1, 0, 2, 4], [1, 1, 2])
```

We get:

```
[] 1, 6[]
```

that is to say the polynomial `poly1[1, 6]` or the polynomial $x+6$.
We enter:

```
rem(t^3+2t+4, t^2+t+2, t)
```

We get:

```
t+6
```

We enter, if we do not put the variable t as last argument:

```
rem(t^3+2t+4, t^2+t+2)
```

We get:

```
0
```

6.11.4 Quotient and Euclidean remainder: `quorem`

`quorem` (or `divide`) gives the list of the quotient and the remainder of the Euclidean division (by decreasing power order) of two polynomials. (See also 6.9.3 and 6.10.7).

We can enter the polynomials either by the list of their coefficients by decreasing power order, either under symbolic forms, and in this case the variable must be added as third argument (by default the variable is x).

To get the quotient and the remainder of the division of $x^3 + 2x + 4$ by $x^2 + x + 2$, we enter:

```
quorem(x^3+2x+4, x^2+x+2)
```

We get:

```
[x-1, x+6]
```

Or we enter:

```
quorem([1, 0, 2, 4], [1, 1, 2])
```

We get:

$$[[1, -1], [1, 6]]$$

that is to say the list of polynomials $[\text{poly1}[1, -1], \text{poly1}[1, 6]]$ then the quotient is the polynomial $x-1$ and the remainder is the polynomial $x+6$.

We enter:

$$\text{quorem}(t^3+2t+4, t^2+t+2, t)$$

We get:

$$[t-1, t+6]$$

We enter:

$$\text{quorem}(t^3+2t+4, t^2+t+2)$$

We get:

$$[(t^3+2t+4)/(t^2+t+2), 0]$$

We enter:

$$\text{quorem}(x^3-1, x^2-1)$$

We get:

$$[x, x-1]$$

We enter:

$$\text{quorem}(t^3-1, t^2-1, t)$$

We get:

$$[t, t-1]$$

6.11.5 GCD of polynomials by Euclid's algorithm: `gcd igcd`

`gcd` or `igcd` designates the GCD (Greatest Common Divisor) of two polynomials which may have several variables and also the GCD of a list of polynomials, or of a sequence of polynomials which may have several variables (see 6.6 for the GCD of integers). We can also put as parameters two lists of same length (or a matrix of two lines), in this case `gcd` returns the greatest common divisor of elements of same index (or of same column).

We enter:

$$\text{gcd}([x^2-4, x*y-y], [x^3-8, y^2-x^2*y])$$

Or we enter:

$$\text{gcd}([[x^2-4, x*y-y], [x^3-8, y^2-x^2*y]])$$

We get:

$$[x-2, y]$$

Examples

We enter:

$$\text{gcd}(x^2+2x+1, x^2-1)$$

142

We get:

$$x+1$$

We enter:

$$\text{gcd}(x^2-2x+1, x^3-1, x^2-1, x^2+x-2)$$

or

$$\text{gcd}([x^2-2x+1, x^3-1, x^2-1, x^2+x-2])$$

We get:

$$x-1$$

We enter:

$$A:=z^2+x^2*y^2*z^2+(-y^2)*z^2+(-x^2)*z^2$$

$$B:=x^3*y^3*z+(-y^3)*z+x^3*z-z$$

$$C:=\text{gcd}(A,B)$$

We get:

$$z*x*y+z*x-z*y-z$$

We enter:

$$\text{factor}(A)$$

We get:

$$(y-1)*(y+1)*(x-1)*(x+1)*z^2$$

We enter:

$$\text{factor}(B)$$

We get:

$$(x^2+x+1)*(x-1)*(y+1)*(y^2-y+1)*z$$

We enter:

$$\text{factor}(C)$$

We get:

$$(y+1)*(x-1)*z$$

For the polynomials with modular coefficients, we enter for example:

$$\text{gcd}((x^2+2x+2) \bmod 5, (x^2-1) \bmod 5)$$

We get:

$$(1 \bmod 5)*x-1 \bmod 5$$

but if we enter:

$$\text{gcd}(x^2+2x+2, x^2-1) \text{ mod } 5)$$

We get:

$$1 \pmod{5}$$

because the modular operation modular is done after the calculation of the GCD which has been calculated in $\mathbb{Z}[X]$.

6.11.6 Choose the algorithm of the GCD of two polynomials: `ezgcd` `modgcd`

`ezgcd` and `modgcd` designate the GCD (Greatest Common Divisor) of two polynomials (or of a list of polynomials, or of a sequence of polynomials) of several variables.

`ezgcd` is calculated with the algorithm `ezgcd`,

`modgcd` is calculated with the modular algorithm.

We enter:

$$\text{gcd}(x^2-2xy+y^2-1, x-y)$$

or

$$\text{ezgcd}(x^2-2xy+y^2-1, x-y)$$

or

$$\text{modgcd}(x^2-2xy+y^2-1, x-y)$$

We get:

$$1$$

We enter:

$$\text{gcd}((x+y-1)*(x+y+1), (x+y+1)^2)$$

or we enter:

$$\text{ezgcd}((x+y-1)*(x+y+1), (x+y+1)^2)$$

or

$$\text{modgcd}((x+y-1)*(x+y+1), (x+y+1)^2)$$

We get:

$$x+y+1$$

We enter:

$$\text{ezgcd}((x+1)^4-y^4, (x+1-y)^2)$$

We get:

"GCD not successfull Error: Bad Argument Value"

but if we enter:

$$\text{gcd}((x+1)^4-y^4, (x+1-y)^2)$$

or

$$\text{modgcd}((x+1)^4 - y^4, (x+1-y)^2)$$

We get:

$$x-y+1$$

6.11.7 LCM of two polynomials: lcm

`lcm` designates the LCM (Lowest Common Multiple) of two polynomials which may have several variables and also the LCM of a list of polynomials or of a sequence of polynomials which may have several variables (see 6.7 for the LCM of integers).

We enter:

$$\text{lcm}(x^2+2*x+1, x^2-1)$$

We get:

$$(x+1) * (x^2-1)$$

We enter:

$$\text{lcm}(x, x^2+2*x+1, x^2-1)$$

or

$$\text{lcm}([x, x^2+2*x+1, x^2-1])$$

We get:

$$(x^2+x) * (x^2-1)$$

We enter:

$$A:=z^2+x^2*y^2*z^2+(-y^2)*z^2+(-x^2)*z^2$$

$$B:=x^3*y^3*z+(-y^3)*z+x^3*z-z$$

$$D:=\text{lcm}(A, B)$$

We get:

$$(x*y*z-x*z+y*z-z) * (x^3*y^3*z+(-y^3)*z+x^3*z-z)$$

We enter:

$$\text{factor}(A)$$

We get:

$$(y-1) * (y+1) * (x-1) * (x+1) * z^2$$

We enter:

$$\text{factor}(B)$$

We get:

$$(x^2+x+1) * (x-1) * (y+1) * (y^2-y+1) * z$$

We enter:

```
factor(D)
```

We get:

$$(x-1) * (x+1) * (x^2+x+1) * (y-1) * (y+1) * (y^2-y+1) * z^2$$

6.11.8 Bezout identity: egcd

It is the Bezout identity for polynomials (Extended Greatest Common Divisor).

`egcd` takes two or three arguments: the polynomials A and B which are either in the form of expressions of one variable, (if the variable is not specified it is x), either supplied by the list of their coefficients by decreasing power order.

Given 2 polynomials $A(x)$, $B(x)$, `egcd` or `gcdex` returns 3 polynomials

$[U(x), V(x), D(x)]$ such as:

$$U(x) * A(x) + V(x) * B(x) = D(x) = \text{GCD}(A(x), B(x))$$

We enter:

```
egcd(x^2+2*x+1, x^2-1)
```

We get:

```
[1, -1, 2*x+2]
```

We enter:

```
egcd([1, 2, 1], [1, 0, -1])
```

We get:

```
[[1], [-1], [2, 2]]
```

We enter:

```
egcd(t^2+2*t+1, t^2-1, t)
```

We get:

```
[1, -1, 2*t+2]
```

We enter:

```
egcd(x^2-2*x+1, x^2-x+2)
```

We get:

```
[x-2, -x+3, 4]
```

We enter:

```
egcd([1, -2, 1], [1, -1, 2])
```

We get:

```
[[1, -2], [-1, 3], [4]]
```

We enter:

```
egcd (t^2-2*t+1, t^2-t+2, t)
```

We get:

```
[t-2, -t+3, 4]
```

6.11.9 Solve polynomial of the form $au + bv = c$: `abcuv`

It is still the Bezout identity.

`abcuv` solves the polynomial equation

$$C(x) = U(x) * A(x) + V(x) * B(x)$$

in which the unknowns are the polynomials U and V and the parameters are the three polynomials, A, B, C where C must be a multiple of the GCD of A and B .

`abcuv` takes as argument three polynomials expressions A, B, C and the name of their variable (by default x) (resp. 3 lists representing the coefficients by decreasing power order of 3 polynomials A, B, C). `abcuv` returns the list of two polynomial expressions U and V (resp. of two lists which are the coefficients by decreasing power order of U and V).

We enter:

```
abcuv (x^2+2*x+1, x^2-1, x+1)
```

We get:

```
[1/2, 1/-2]
```

We enter:

```
abcuv (x^2+2*x+1, x^2-1, x^3+1)
```

We get:

```
[1/2*x^2+1/-2*x+1/2, -1/2*x^2-1/-2*x-1/2]
```

We enter:

```
abcuv ([1, 2, 1], [1, 0, -1], [1, 0, 0, 1])
```

We get:

```
[poly1 [1/2, 1/-2, 1/2], poly1 [1/-2, 1/2, 1/-2]]
```

6.11.10 Chinese remainder: `chinrem`

`chinrem` takes as argument two lists having each as components two polynomials eventually supplied by the list of their coefficients by decreasing power order.

`chinrem` returns a list of components of two polynomials.

`chinrem([A, R], [B, Q])` returns the list of polynomials P and S such as:

$$S = R \cdot Q, P = A \pmod{R}, P = B \pmod{Q}$$

There is always a solution P if R and Q are prime to each other, and all the solutions are congruent modulus $S = R * Q$

Find the solutions $P(x)$ of:

$$\begin{cases} P(x) = x \pmod{x^2 + 1} \\ P(x) = x - 1 \pmod{x^2 - 1} \end{cases}$$

We enter:

```
chinrem([[1, 0], [1, 0, 1]], [[1, -1], [1, 0, -1]])
```

We get:

```
[[1/-2, 1, 1/-2], [1, 0, 0, 0, -1]]
```

or we enter:

```
chinrem([x, x^2+1], [x-1, x^2-1])
```

We get:

```
[1/-2*x^2+x+1/-2, x^4-1]
```

so $P(x) = -\frac{x^2 - 2x + 1}{2} \pmod{x^4 - 1}$

Other example:

We enter:

```
chinrem([[1, 2], [1, 0, 1]], [[1, 1], [1, 1, 1]])
```

We get:

```
[[-1, -1, 0, 1], [1, 1, 2, 1, 1]]
```

or we enter:

```
chinrem([x+2, x^2+1], [x+1, x^2+x+1])
```

We get:

```
[-x^3-x^2+1, x^4+x^3+2*x^2+x+1]
```

Chapter 7 Menu Polynomial

7.1 Canonical form: `canonical_form`

`canonical_form` takes as parameter a trinomial of the second degree that we want to put into the canonical form.

Example:

Transform into canonical form:

$$x^2 - 6x + 1$$

We enter:

```
canonical_form(x^2-6*x+1)
```

We find:

$$(x-3)^2-8$$

7.2 Numerical roots of a polynomial: `root`

`root` takes as argument a polynomial or the vector whose components are the coefficients of a polynomial (by decreasing order).

`root` returns a vector whose components are the numerical roots of the polynomial.

To find the numerical roots of $P(x) = x^3 + 1$, we enter:

```
root([1,0,0,1])
```

or we enter:

```
root(x^3+1)
```

We get:

```
[-1,0.5+0.866025403784*i,0.5-0.866025403784*i]
```

To get the numerical roots of $x^2 - 3$, we enter:

```
root([1,0,-3])
```

or:

```
root(x^2-3)
```

We get:

```
[1.73205080757,-1.73205080757]
```

To find the numerical roots of $P(x) = x^3 - 5x^2 + 8x - 4$, we enter:

```
root([1,-5,8,-4])
```

or we enter:

```
root(x^3-5x^2+8x-4)
```

We get:

```
[1., 2., 2.]
```

7.3 Roots exact of a polynomial

7.3.1 Exact boundaries of complex roots of a polynomial: `complexroot`

`complexroot` has two or four arguments: a polynomial and a real number, and eventually two complex α, β .

- if `complexroot` has two arguments, `complexroot` returns the list of vectors of coordinates the value of complex and exact roots of the polynomial and their multiplicity, or of coordinates an interval (the boundaries of the interval are the opposite vertices of a rectangle with sides parallel to the axis and in which is a root complex of the polynomial) and the multiplicity of this root.
If the interval is $[a_1 + ib_1, a_2 + ib_2]$ we have $|a_1 - a_2| < \varepsilon$ and $|b_1 - b_2| < \varepsilon$ and the root $a + ib$ checks $a_1 \leq a \leq a_2$ and $b_1 \leq b \leq b_2$.
- if `complexroot` has four arguments, `complexroot` only returns the roots laying in the rectangle with sides parallel to the axis and of opposite vertices α, β .

To get the roots of $x^3 + 1$, we enter:

```
complexroot(x^3+1, 0.1)
```

We get:

```
[[-1, 1], [(4-7*i)/8, (8-13*i)/16], 1], [(8+13*i)/16, (4+7*i)/8], 1]]
```

So for $x^3 + 1$:

-1 is a root of multiplicity 1, $1/2i * b$ is a root of multiplicity 1 with $-7/8 \leq b \leq -13/16$, $1/2i * c$ is root of multiplicity 1 with $13/16 \leq c \leq 7/8$.

To get the roots of $x^3 + 1$ in the rectangle of opposite vertices $-1, 1 + 2 * I$, we enter:

```
complexroot(x^3+1, 0.1, -1, 1+2*i)
```

We get:

```
[[-1, 1], [(8+13*i)/16, (4+7*i)/8], 1]]
```

7.3.2 Exact values of complex rational roots of a polynomial: `crationalroot`

`crationalroot` has one or three arguments: a polynomial and eventually two complex α, β .

- if `crationalroot` has one argument, `crationalroot` returns the list of values of complex roots rational of the polynomial without tell the multiplicity of these roots.
- if `crationalroot` has three arguments, `crationalroot` only returns the complex rational roots laying in the rectangle of opposite vertices $[\alpha, \beta]$.

To get the roots rational and complex of $(x^2 + 4) * (2x - 3) = 2 * x^3 - 3 * x^2 + 8 * x - 12$, we enter:

```
crationalroot(2*x^3-3*x^2+8*x-12)
```

We get:

```
[2*i, 3/2, -2*i]
```

7.4 Fraction rational, its roots and its exact poles

7.4.1 Roots and exact poles of a rational fraction: `froot`

`froot` takes as argument a rational fraction $F(x)$.

`froot` returns a vector whose components are the roots and the poles of $F(x)$ followed by their multiplicity.

The calculator returns the exact values of these roots or poles when possible and otherwise returns their numerical values.

We enter:

```
froot((x^5-2*x^4+x^3)/(x-2))
```

We get:

```
[1, 2, 0, 3, 2, -1]
```

so for $F(x) = \frac{x^5 - 2x^4 + x^3}{x-2}$:

1 is a double root,

0 is a triple root

and 2 is a pole of order 1.

We enter:

```
froot((x^3-2*x^2+1)/(x-2))
```

We get:

```
[1, 1, (1+sqrt(5))/2, 1, (1-sqrt(5))/2, 1, 2, -1]
```

Note: to get the roots and the complex poles, we must have checked Complex in the CAS configuration (key giving the status line).

We enter:

```
froot((x^2+1)/(x-2))
```

We get:

```
[-i, 1, i, 1, 2, -1]
```

7.5 Writing in powers of $(x - a)$: `ptayl`

It is to write a polynomial $P(x)$ in powers of $(x - a)$.

`ptayl` has two parameters: a polynomial P supplied in symbolic form or by the list of its coefficients, and a number a .

`ptayl` returns the polynomial Q such as $Q(x - a) = P(x)$.

We enter:

```
ptayl(x^2+2*x+1, 2)
```

We get the polynomial $Q(x)$:

```
x^2+6*x+9
```

We enter:

```
ptayl([1,2,1],2)
```

We get:

```
[1,6,9]
```

Warning!

We have:

$$P(x) = Q(x - a)$$

that is to say for the example:

$$x^2 + 2x + 1 = (x - 2)^2 + 6(x - 2) + 9$$

7.6 Calculation with the exact roots of a polynomial: `rootof`

Be P and Q two polynomials supplied by the list of their coefficients, so `rootof(P,Q)` designates the value $P(\alpha)$ where α is the "largest" root of Q (one first compares the real parts and in case of equality one compare the imaginary parts).

Then, we can perform calculations with this value.

We enter:

```
normal(rootof([1,0],[1,2,-3]))
```

We get:

```
1
```

indeed $x^2 + 2x - 3 = (x - 1)(x + 3)$ takes as largest root 1.

Other example :

Be α the largest root in norm of $Q(x) = x^4 + 10x^2 + 1$.

- Calculate $\frac{1}{\alpha}$

We enter:

```
normal(1/rootof([1,0],[1,0,10,0,1]))
```

because $P(x) = x$ is represented by [1,0].

We get:

```
rootof([-1,0,-10,0],[1,0,10,0,1])
```

which means that:

$$\frac{1}{\alpha} = -(\alpha)^3 - 10.\alpha$$

- Calculate $(\alpha)^2$.

We enter:

```
normal(rootof([1,0],[1,0,10,0,1])^2)
```

We have $\alpha = \text{rootof}([1,0],[1,0,10,0,1])$ because $P(x) = x$ is represented by [1,0], and to get α^2 , we raise α to square.

We get:

```
-5-2*sqrt(6)
```

or to get α^2 directly, we enter:

```
normal(rootof([1,0,0],[1,0,10,0,1])^2)
```

because $P(x) = x^2$ is represented by [1,0,0].

We get:

```
-5-2*sqrt(6)
```

This result can be checked because we have a biquadratic equation of reduced discriminant $25 - 1 = 24 = 4 * 6$.

We enter:

```
csolve(x^4+10x^2+1)
```

We get:

```
[(i)*sqrt(-2*sqrt(6)+5),
(-i)*sqrt(-2*sqrt(6)+5),
(i)*sqrt(2*sqrt(6)+5),
(-i)*sqrt(2*sqrt(6)+5)]
```

So $\alpha = i * \sqrt{2 * \sqrt{6} + 5}$

We enter:

```
((i)*sqrt(2*sqrt(6)+5))^2
```

We get:

```
-5-2*sqrt(6)
```

7.7 Coefficients of a polynomial: `coeff`

`coeff` has three arguments: the polynomial, the name of the variable (or the list of names of the variables) the order (or the list of orders of variables).

`coeff` returns the coefficient of the polynomial of specified order.

We enter:

```
coeff(x^3-5x^2+8x-4,2)
```

We get:

```
-5
```

We enter:

```
coeff(-x^4+3*x*y^2+x,y,2)
```

We get:

```
3*x
```

We enter:

```
coeff(-x^4+3*x*y^2+x,[x,y],[1,2])
```

We get:

7.8 Coefficients of a polynomial defined by its roots: `pcoeff` `pcoef`

`pcoeff` (or `pcoef`) takes as argument a list whose components are the roots of a polynomial P . `pcoeff` (or `pcoef`) returns a list of components the coefficients of the polynomial univariate P (by decreasing order).

We enter:

```
pcoef([1, 2, 0, 0, 3])
```

We get:

```
[1, -6, 11, -6, 0, 0]
```

that is to say $(x - 1)(x - 2)(x^2)(x - 3) = x^5 - 6x^4 + 11x^3 - 6x^2$.

7.9 Truncation of order n : `truncate`

`truncate` allows to truncate a polynomial at a supplied order. `truncate` is useful when we do series expansions by hand, or to transform a series expansion into a polynomial.

`truncate` has two arguments: a polynomial and an integer n .

`truncate` returns the polynomial truncated at order n (no terms of order greater than or equal to $n+1$).

We enter:

```
truncate((1+x+x^2/2)^3, 4)
```

We get:

```
(9*x^4+16*x^3+18*x^2+12*x+4)/4
```

We enter:

```
truncate(series(sin(x)), 4)
```

We get:

```
(-x^3-(-6)*x)/6
```

We notice that the returned polynomial is reduced to common denominator.

7.10 List of divisors of a polynomial: `divis`

`divis` takes as argument a symbolic polynomial (or a list of polynomials) and returns the list of divisors.

We enter:

```
divis(x^2-1)
```

We get:

```
[1, x-1, x+1, (x-1)*(x+1)]
```

We enter:

```
divis(2t^2-2)
```

We get:

```
[1, 2, t-1, 2*(t-1), t+1, 2*(t+1), (t-1)*(t+1), 2*(t-1)*(t+1)]
```

We enter:

```
divis([t^2, x^2-1])
```

We get:

```
[[1, t, t^2], [1, x+1, x-1, (x-1)*(x+1)]]
```

7.11 List of factors of a polynomial: `factors`

`factors` takes as argument a polynomial or a list of polynomials.
`factors` gives the list of factors of the polynomial with their multiplicity.

We enter:

```
factors(x^2+2*x+1)
```

We get:

```
[x+1, 2]
```

We enter:

```
factors(x^4-2*x^2+1)
```

We get:

```
[x-1, 2, x+1, 2]
```

We enter:

```
factors([x^3-2*x^2+1, x^2-x])
```

We get:

```
[[x-1, 1, x^2-x-1, 1], [x, 1, x-1, 1]]
```

We enter:

```
factors([x^2, x^2-1])
```

We get:

```
[[x, 2], [x+1, 1, x-1, 1]]
```

7.12 GCD of polynomials by Euclid's algorithm: `gcd`

`gcd` designates the GCD (Greatest Common Divisor) of two polynomials pou-vant get several variables and also the GCD of a list of polynomials or of a sequence of polynomials which may have several variables (see 6.6 for the GCD of integers).

We can also put as parameters two lists of same length (or a matrix of two lines), in this case `gcd` returns the greatest common divisor of elements of same index (or of a same column).

We enter:

$$\text{gcd}(x^2+2x+1, x^2-1)$$

We get:

$$x+1$$

We enter:

$$\text{gcd}([x^2-4, x*y-y], [x^3-8, y^2-x^2*y])$$

Or we enter:

$$\text{gcd}([[x^2-4, x*y-y], [x^3-8, y^2-x^2*y]])$$

We get:

$$[x-2, y]$$

We enter:

$$\text{gcd}(x^2-2x+1, x^3-1, x^2-1, x^2+x-2)$$

or

$$\text{gcd}([x^2-2x+1, x^3-1, x^2-1, x^2+x-2])$$

We get:

$$x-1$$

We enter:

$$A:=z^2+x^2*y^2*z^2+(-y^2)*z^2+(-x^2)*z^2$$

$$B:=x^3*y^3*z+(-y^3)*z+x^3*z-z$$

$$C:=\text{gcd}(A, B)$$

We get:

$$z*x*y+z*x-z*y-z$$

We enter:

$$\text{factor}(A)$$

We get:

$$(y-1)*(y+1)*(x-1)*(x+1)*z^2$$

We enter:

$$\text{factor}(B)$$

We get:

$$(x^2+x+1) * (x-1) * (y+1) * (y^2-y+1) * z$$

We enter:

$$\text{factor}(C)$$

We get:

$$(y+1) * (x-1) * z$$

For the polynomials with modular coefficients, we enter, for example, because %% is there used to designate a modular number:

$$\text{gcd}((x^2+2*x+2) \% 5, (x^2-1) \% 5)$$

We get:

$$(1 \% 5) * x - 1 \% 5$$

but if we enter:

$$\text{gcd}(x^2+2*x+2, x^2-1) \% 5$$

We get:

$$1 \% 5$$

because the modular operation is performed after the calculation of the GCD which has been calculated in $Z[X]$.

7.13 LCM of two polynomials: lcm

`lcm` designates the LCM (lowest common multiple) of two polynomials which may have several variables and also the LCM of a list of polynomials or of a sequence of polynomials which may have several variables (see 6.7 for the LCM of integers).

We enter:

$$\text{lcm}(x^2+2*x+1, x^2-1)$$

We get:

$$(x+1) * (x^2-1)$$

We enter:

$$\text{lcm}(x, x^2+2*x+1, x^2-1)$$

or

$$\text{lcm}([x, x^2+2*x+1, x^2-1])$$

We get:

$$(x^2+x) * (x^2-1)$$

We enter:

$$A := z^2 + x^2 * y^2 * z^2 + (- (y^2)) * z^2 + (- (x^2)) * z^2$$

$$B:=x^3*y^3*z+(-y^3)*z+x^3*z-z$$

$$D:=\text{lcm}(A,B)$$

We get:

$$(x*y*z-x*z+y*z-z)*(x^3*y^3*z+(-y^3)*z+x^3*z-z)$$

We enter:

$$\text{factor}(A)$$

We get:

$$(y-1)*(y+1)*(x-1)*(x+1)*z^2$$

We enter:

$$\text{factor}(B)$$

We get:

$$(x^2+x+1)*(x-1)*(y+1)*(y^2-y+1)*z$$

We enter:

$$\text{factor}(D)$$

We get:

$$(x-1)*(x+1)*(x^2+x+1)*(y-1)*(y+1)*(y^2-y+1)*z^2$$

7.14 Create

7.14.1 Transform a polynomial into a list (internal recursive dense format):

`symb2poly`

`symb2poly` takes as argument a polynomial, supplied with a polynomial writing, of a variable (resp. several variables), and the name of this formal variable (by default `x`) (resp. the sequence of names of these variables).

`symb2poly` transforms this polynomial writing, into the list of coefficients by decreasing power order according to the name of the variable supplied as second argument (resp. the recursive writing of the list of coefficients by decreasing power order according to the names of variables supplied as second argument: the result is the list of coefficients of the first variable, coefficients which are itself polynomials which will be supplied in form of the list of coefficients of the second variable, etc., ...).

Warning! If the second argument is a list, the result is the writing of the polynomial under internal format.

We enter:

$$\text{symb2poly}(x^2-1)$$

Or we enter:

$$\text{symb2poly}(x^2-1, x)$$

Or we enter:

$$\text{symb2poly}(y^2-1, y)$$

We get:

$$[1, 0, -1]$$

We enter:

```
symb2poly(x*y^2+2y-1,x)
```

We get:

$$[y^2, 2y-1]$$

We enter:

```
symb2poly(x*y^2+2y-1,y)
```

We get:

$$[x, 2, -1]$$

7.14.2 Transform the internal sparse distributed format of the polynomial into a polynomial writing: `poly2symb`

`poly2symb` takes as argument a list of coefficients by decreasing power order of a polynomial and a name of formal variable (by default `x`) (resp. the internal sparse distributed format of the polynomial that is to say the sum of monomials such as: `%%{c, [px, py, pz] %%}`) and a list of formal variables such as `[x, y, z]` which represents the monomial $cx^{px}y^{py}z^{pz}$).

`poly2symb` transforms the list of coefficients by decreasing power order of a polynomial (resp. the sum of `%%{c, [px, py, pz] %%}`), in its polynomial writing (according to Horner), by using the name of the variable supplied in second argument (resp. by using the list of variables supplied in second argument `[x, y, z]`).

We enter:

```
poly2symb([1, 0, -1])
```

Or we enter:

```
poly2symb([1, 0, -1], x)
```

We get:

$$x*x-1$$

We enter:

```
poly2symb([1, 0, -1], y)
```

We get:

$$y*y-1$$

7.14.3 Coefficients of a polynomial defined by its roots: `pcoeff` `pcoef`

`pcoeff` (or `pcoef`) takes as argument a list of components the roots of a polynomial `P`.

`pcoeff` (or `pcoef`) returns a list of components the coefficients of the polynomial univariate `P` (by decreasing order).

We enter:

```
pcoef([1,2,0,0,3])
```

We get:

```
[1,-6,11,-6,0,0]
```

that is to say $(x - 1)(x - 2)(x^2)(x - 3) = x^5 - 6x^4 + 11x^3 - 6x^2$.

7.14.4 Coefficients of a rational fraction defined by its roots and its poles: `fcoeff`

`fcoeff` takes as argument a vector whose components are the roots and the poles of a rational fraction $F(x)$ followed by their multiplicity.

`fcoeff` returns the rational fraction $F(x)$.

We enter:

```
fcoeff([1,2,0,3,2,-1])
```

We get:

```
(x-1)^2*x^3*(x-2)^-1
```

7.14.5 Coefficients of the term of highest degree of a polynomial: `lcoeff`

`lcoeff` takes as argument a polynomial supplied in symbolic form or by the list of its coefficients.

`lcoeff` returns the coefficient of highest degree of this polynomial (`lcoeff`=leading coefficient).

We enter:

```
lcoeff([2,1,-1,0])
```

We get:

```
2
```

We enter:

```
lcoeff(3*x^2+5*x,x)
```

We get:

```
3
```

We enter:

```
lcoeff(3*x^2+5*x*y^2,y)
```

We get:

```
5*x
```

7.14.6 Evaluation of a polynomial: `polyEval`

`polyEval` takes as argument a polynomial p supplied by the list of its coefficients and a real a .

`polyEval` returns the numerical or exact value of $p(a)$.

We enter:

```
polyEval([1,0,-1],sqrt(2))
```

We get:

```
sqrt(2)*sqrt(2)-1
```

Then:

```
normal(sqrt(2)*sqrt(2)-1)
```

We get:

```
1
```

We enter:

```
polyEval([1,0,-1],1.4)
```

We get:

```
0.96
```

7.14.7 Minimal polynomial: `pmin`

`pmin` has one (resp. two) argument(s).

`pmin` takes as argument a matrix A of degree n (resp. a matrix A of degree n and a name of formal variable).

`pmin` returns the minimal polynomial of A written as a list of its coefficients (resp. the minimal polynomial P of A written in symbolic form by using the name of variable supplied as argument).

The minimal polynomial P of A is the polynomial of lowest degree which makes A equals zero ($P(A) = 0$).

We enter:

```
pmin([[1,0],[0,1]])
```

We get:

```
[1,-1]
```

Or we enter:

```
pmin([[1,0],[0,1]],x)
```

We get:

```
x-1
```

So the minimal polynomial of $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is $x - 1$.

We enter:

```
pmin([[1,1,0],[0,1,1],[0,0,1]])
```

We get:

```
[1,-3,3,-1]
```

We enter:

```
pmin([[1,1,0],[0,1,1],[0,0,1]],x)
```

We get:

$$x^3 - 3x^2 + 3x - 1$$

So the minimal polynomial of $[[1,1,0], [0,1,1], [0,0,1]]$ is $x^3 - 3x^2 + 3x - 1$.

We enter:

```
pmin([[2,1,0],[0,2,0],[0,0,2]])
```

We get:

$$[1, -4, 4]$$

We enter:

```
pmin([[2,1,0],[0,2,0],[0,0,2]],x)
```

We get:

$$x^2 - 4x + 4$$

So the minimal polynomial of $[[2,1,0], [0,2,0], [0,0,2]]$ is $x^2 - 4x + 4$.

7.14.8 Companion matrix of a polynomial: `companion`

`companion` takes as argument a univariate polynomial P and the name of its variable.

`companion` returns the matrix which has for characteristic polynomial the polynomial P .

If $P(x) = x + a_{n-1}x^{n-1} + \dots + a_{-1}x + a_0$, this matrix equals the matrix unity of order $n - 1$ combined with $[0, 0, \dots, 0, -a_0]$ as first line, and by $[-a_0, -a_1, \dots, -a_{n-1}]$ as last column.

We enter:

```
companion(x^2+5x-7,x)
```

We get:

$$[[0, 7], [1, -5]]$$

We enter:

```
companion(x^4+3x^3+2x^2+4x-1,x)
```

We get:

$$[[0, 0, 0, 1], [1, 0, 0, -4], [0, 1, 0, -2], [0, 0, 1, -3]]$$

7.14.9 Random polynomials: `randpoly` `randPoly`

`randpoly` `randPoly` takes as parameter an integer n .

`randPoly` returns the coefficients of a polynomial of degree n and whose coefficients are random integers equally distributed on $-99..+99$.

We enter in HOME or in the CAS:

```
randPoly(4)
```

or

```
randpoly(4)
```

We get for example:

```
[4, 53, -45, 80, -99)
```

7.14.10 Change the order of variables: `reorder`

`reorder` has two parameters: an expression and a list containing the names of variables in a certain order.

`reorder` expands the expression by order of the variables supplied as second parameter.

We enter:

```
reorder(x^2+2*x*a+a^2+z^2-x*z, [a, x, z])
```

We get:

```
a^2+2*a*x+x^2-x*z+z^2
```

Warning! The variables must not be assigned.

7.15 Algebra

7.15.1 Euclidean quotient of two polynomials: `quo`

`quo` gives the quotient of the Euclidean division of two polynomials (division by decreasing power order).

We can give the polynomials either by the list of their coefficients by decreasing power order, either in symbolics forms, and in this case the variable must be added as third argument (by default the variable is x).

We enter:

```
quo(x^2+2x+1, x+3)
```

We get:

```
x-1
```

We enter:

```
quo(t^2+2t+1, t+3, t)
```

We get:

```
t-1
```

or we enter:

```
quo([1, 2, 1], [1, 3])
```

We get:

```
[ 1, -1]
```

that is to say the polynomial `poly1[1, -1]`.

To get the quotient of $x^3 + 2x + 4$ by $x^2 + x + 2$, we enter:

```
quo(x^3+2x+4, x^2+x+2)
```

We get:

$$x-1$$

Or we enter:

```
quo([1,0,2,4],[1,1,2])
```

We get:

```
[] 1,-1 []
```

that is to say the polynomial `poly1[1,-1]` or else the polynomial $x-1$.

We enter:

```
quo(t^3+2t+4,t^2+t+2,t)
```

We get:

$$t-1$$

If we do not put the variable t as last argument, we enter:

```
quo(t^3+2t+4,t^2+t+2)
```

We get:

```
(t^3+2*t+4)/(t^2+t+2)
```

7.15.2 Euclidean remainder of two polynomials: `rem`

`rem` gives the remainder of the Euclidean division of two polynomials (division by decreasing power order).

We can give the polynomials either by the list of their coefficients by decreasing power order, either in symbolic forms, and in this case the variable must be added as third argument (by default the variable is x).

We enter:

```
rem(x^3-1,x^2-1)
```

We get:

$$x-1$$

We enter:

```
rem(t^3-1,t^2-1,t)
```

We get:

$$t-1$$

We enter:

```
rem(x^2+2x+1,x+3)
```

Or we enter:

```
rem(t^2+2t+1,t+3,t)
```

We get:

4

or we enter:

```
rem([1, 2, 1], [1, 3])
```

We get:

[] 4

that is to say the polynomial `poly1[4]` or else the polynomial 4.

To get the remainder of $x^3 + 2x + 4$ by $x^2 + x + 2$, we enter:

```
rem(x^3+2x+4, x^2+x+2)
```

We get:

x+6

Or we enter:

```
rem([1, 0, 2, 4], [1, 1, 2])
```

We get:

[1, 6]

that is to say the polynomial `poly1[1, 6]` or else the polynomial $x+6$.

We enter:

```
rem(t^3+2t+4, t^2+t+2, t)
```

We get:

t+6

If we do not put the variable t as last argument, we enter:

```
rem(t^3+2t+4, t^2+t+2)
```

We get:

0

7.15.3 Degree of a polynomial: `degree`

`degree` takes as argument a polynomial supplied in symbolic form or by the list of its coefficients. `degree` returns the degree of this polynomial (degree of the monomial of highest degree).

We enter:

```
degree(x^3+x)
```

We get:

3

We enter:

```
degree([1,0,1,0])
```

We get:

3

7.15.4 Valuation of a polynomial: `valuation`

`valuation` or `ldegree` takes as argument a polynomial supplied in symbolic form or by the list of its coefficients.

`valuation` or `ldegree` returns the valuation of this polynomial, it is the degree of the monomial of lowest (`ldegree=low degree`).

We enter:

```
valuation(x^3+x)
```

We get:

1

We enter:

```
valuation([1,0,1,0])
```

We get:

1

7.15.5 Coefficient of the term of highest degree of a polynomial: `lcoeff`

`lcoeff` takes as argument a polynomial supplied in symbolic form or by the list of its coefficients.

`lcoeff` returns the coefficient of highest degree of this polynomial (`lcoeff=leading coefficient`).

We enter:

```
lcoeff([2,1,-1,0])
```

We get:

2

We enter:

```
lcoeff(3*x^2+5*x, x)
```

We get:

3

We enter:

```
lcoeff(3*x^2+5*x*y^2, y)
```

We get:

5*x

7.15.6 Put in factor of x^n in a polynomial: `factor_xn`

`factor_xn` takes as argument a polynomial P .

`factor_xn` returns the polynomial P in which we have put in factor x^n where n is the degree of P ($n = \text{degree}(P)$).

We enter:

```
factor_xn(-x^4+3)
```

We get:

```
x^4*(-1+3*x^-4)
```

7.15.7 GCD of coefficients of a polynomial: `content`

`content` takes as arguments a polynomial P supplied in symbolic form or by the list of its coefficients and the name of the variable (by default x).

`content` designates the GCD (Greatest Common Divisor) of coefficients the polynomial P .

We enter:

```
content(6*x^2-3*x+9)
```

or we enter:

```
content(6*t^2-3*t+9,t)
```

or:

```
content([6,-3,9])
```

We get:

```
3
```

7.15.8 Primitive part of a polynomial: `primpart`

`primpart` takes as argument a polynomial P supplied in symbolic form or by the list of its coefficients.

`primpart` returns the polynomial P divided by the GCD (Greatest Common Divisor) of its coefficients.

We enter:

```
primpart(6x^2-3x+9)
```

or:

```
primpart([6,-3,9],x)
```

We get:

```
2*x^2-x+3
```

7.15.9 Sturm sequence and number of changes of the sign of P on $]a; b[$: `sturm`

`sturm` has two or four parameters: a polynomial expression P , or a rational fraction P/Q , and the name of the variable, or a polynomial expression P , the name of the variable and two numbers a and b .

When there are two parameters, `sturm` returns the list of Sturm series and their multiplicity for P or for P and for Q (`sturm` is then similar to `sturmseq`).

When there are four parameters, `sturm` behaves as `sturmab`:

- if a and b are reals, `sturm` returns the number of changes of sign of P on $]a; b]$
- if a or b is complex, `sturm` returns the number of complex roots laying inside the rectangle of opposite vertices a and b .

We enter:

```
sturm(2*x^3+2, x)
```

We get:

```
[2, [[1, 0, 0, 1], [3, 0, 0], -9], 1]
```

We enter:

```
sturm((2*x^3+2)/(x+2), x)
```

We get:

```
[2, [[1, 0, 0, 1], [3, 0, 0], -9], 1, [[1, 2], 1]]
```

We enter:

```
sturm(x^2*(x^3+2), x, -2, 0)
```

We get:

1

7.15.10 Number of changes of sign on $]a; b]$: `sturmab`

`sturmab` has four parameters: a polynomial expression P , the name of the variable and two numbers a and b .

- if a and b are reals, `sturmab` returns either a strictly positive number which is the number of changes of sign of P on $]a; b]$, either 0 if P remains of constant positive sign or null on $]a; b]$, either -1 if P remains of constant negative sign or null on $]a; b]$. Thus, `sturmab` let us know the number of roots on $[a, b[$ of the polynomial P/G with $G = \text{gcd}(P, \text{diff}(P))$.
- if a or b is complex, the number of complex roots laying inside the rectangle of opposite vertices a and b .

We enter:

```
sturmab(x^2*(x^3+2), x, -2, 0)
```

We get:

1

We enter:

```
sturmab(x^3-1, x, -2-i, 5+3i)
```

We get:

3

We enter:

```
sturmab(x^3-1,x,-i,5+3i)
```

We get:

```
1
```

Warning!

P must be supplied by its symbolic expression, and, if we enter:

```
sturmab([1,0,0,2,0,0],x,-2,0)
```

we get:

```
Bad argument type.
```

7.15.11 Sequence of Sturm: `sturmseq`

`sturmseq` takes as parameter a polynomial expression P or a rational fraction P/Q .

`sturmseq` returns the list of Sturm series and their multiplicity for P , or for P and Q .

The series of sturm R_1, R_2, \dots is obtained from the factor F without square of P . To get F starting from the decomposition of P in prime factors, one removes the square terms and one transforms the odd powers in powers 1.

R_1 is the opposite of the remainder of the Euclidean division of F by F' then, R_2 is the opposite of the remainder of the Euclidean division of F' by R_1

....

and so on until $R_k = 0$.

We enter:

```
sturmseq(2*x^3+2)
```

or

```
sturmseq(2*y^3+2,y)
```

We get:

```
[2, [[1,0,0,1], [3,0,0], -9], 1]
```

The first term gives the GCD of the coefficients of the numerator (here 2), the last term gives the denominator (here 1). Between these two terms, we have the series of polynomials $[x^3 + 1, 3x^2, -9]$.

We enter:

```
sturmseq((12*x^3+4)/(6*x^2+3),x)
```

We get:

```
[4, [[3,0,0,1], [9,0,0], -81], 3, [[2,0,1], [4,0], -16]]
```

The first term gives the GCD of the coefficients of the numerator (here 4), then the Sturm series of the numerator ($[[3,0,0,1], [9,0,0], -81]$), then the the GCD of the coefficient of the denominator (here 3), and the Sturm series of the denominator ($[[2,0,1], [4,0], -16]$). We have the series of polynomials $[3x^3 + 1, 9x^2, -81]$ for the numerator and, $[2x^2 + 1, 4x, -16]$ for the denominator.

We enter:

```
sturmseq((x^3+1)^2,x)
```

We get:

[1, 1]

Indeed, the square terms are removed and $F = 1$.

We enter:

```
sturmseq(3*(3*x^3+1)/(2*x+2), x)
```

We get:

```
[3, [[3, 0, 0, 1], [9, 0, 0], -81], 2, [[1, 1], 1]]
```

The first term gives the GCD of the coefficients of the numerator (here 3), the second term gives the series of polynomials (here $3x^3+1$, $9x^2$, 81), the third term gives the GCD of the coefficients of the denominator (here 2), the fourth term gives the series of polynomials of the denominator ($x+1$, 1).

Warning!

P must be supplied by its symbolic expression, and, if we enter:

```
sturmseq([1, 0, 0, 1], x)
```

we get:

```
Bad argument type.
```

7.15.12 Sylvester matrix of two polynomials: `sylvester`

`sylvester` takes as arguments two polynomials.

`sylvester` returns the Sylvester matrix S of two polynomials.

For two polynomials $A(x) = \sum_{i=0}^n a_i x^i$ and $B(x) = \sum_{i=0}^m b_i x^i$, the Sylvester matrix S is a square matrix of dimension $m+n$ whose $m = \text{degree}(B(x))$ first lines are composed of coefficients of $A(x)$:

$$\begin{pmatrix} s_{11} = a_n & s_{12} = a_{n-1} & \cdots & s_{1(n+1)} = a_0 & 0 & \cdots & 0 \\ s_{21} = 0 & s_{22} = a_n & \cdots & s_{2(n+1)} = a_1 & s_{2(n+2)} = a_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{m1} = 0 & s_{m2} = 0 & \cdots & s_{m(n+1)} = a_{m-1} & s_{m(n+2)} = a_{m-2} & \cdots & a_0 \end{pmatrix}$$

and the $n = \text{degree}(A(x))$ following lines are composed the same way of coefficients of $B(x)$:

$$\begin{pmatrix} s_{(m+1)1} = b_m & s_{(m+1)2} = b_{m-1} & \cdots & s_{(m+1)(m+1)} = b_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{(m+n)1} = 0 & s_{(m+n)2} = 0 & \cdots & s_{(m+n)(m+1)} = b_{n-1} & b_{n-2} & \cdots & b_0 \end{pmatrix}$$

We enter:

```
sylvester(x^3-p*x+q, 3*x^2-p, x)
```

We get:

```
[[1, 0, -p, q, 0], [0, 1, 0, -p, q], [3, 0, -p, 0, 0], [0, 3, 0, -p, 0], [0, 0, 3, 0, -p]]
```

We enter:

```
det([[1, 0, -p, q, 0], [0, 1, 0, -p, q], [3, 0, -p, 0, 0], [0, 3, 0, -p, 0], [0, 0, 3, 0, -p]])
```

We get:

```
-4*p^3-27*q^2
```

7.15.13 Resultant of two polynomials: `resultant`

`resultant` takes as arguments two polynomials.

`resultant` returns the resultant of two polynomials.

The resultant is the determinant of the Sylvester matrix S .

For the two polynomials $A(x) = \sum_{i=0}^{n-1} a_i x^i$ and $B(x) = \sum_{i=0}^{m-1} b_i x^i$, the Sylvester matrix S is a square matrix of dimension $m + n$ whose m first lines are composed of coefficients of $A(x)$:

$$\begin{pmatrix} s_{00} = a_n & s_{01} = a_{n-1} & \cdots & s_{0n} = a_0 & 0 & \cdots & 0 \\ s_{10} = 0 & s_{11} = a_n & \cdots & s_{1n} = a_1 & s_{1(n+1)} = a_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{(m-1)0} = 0 & s_{(m-1)1} = 0 & \cdots & s_{(m-1)n} = a_{m-1} & s_{(m-1)(n+1)} = a_{m-2} & \cdots & a_0 \end{pmatrix}$$

and the n following lines are composed the same way of from coefficients of $B(x)$:

We enter:

```
resultant(x^3-p*x+q, 3*x^2-p, x)
```

We get:

$$-4 * p^3 - 27 * q^2$$

We look for two polynomials $U(x) = \alpha * x + \beta$ (of degree 1) and $V(x) = \gamma * x^2 + \delta * x + \varepsilon$ (of degree 2) so that $U(x) * (x^3 - p * x + q) + V(x) * (3 * x^2 - p) = 1$

Then, we must solve a linear system of five equations and five unknowns, which are $\alpha, \dots, \delta, \eta$ (**Warning!** $\varepsilon = 10^{-10}$).

We enter:

```
symb2poly((alpha*x+beta)*(x^3-p*x+q)+(gamma*x^2+delta*x+eta)*(3*x^2-p), x)
```

We get:

```
poly1[alpha+3*gamma, beta+3*delta, -alpha*p-p*gamma+3*eta,
      alpha*q-beta*p-p*delta, beta*q-p*eta]
```

The matrix A of this system is then:

$$A = \begin{pmatrix} 1 & 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 \\ -p & 0 & -p & 0 & 3 \\ q & -p & 0 & -p & 0 \\ 0 & q & 0 & 0 & -p \end{pmatrix}$$

the Sylvester matrix S is the transpose matrix of A :

$$S = \begin{pmatrix} 1 & 0 & -p & q & 0 \\ 0 & 1 & 0 & -p & q \\ 3 & 0 & -p & 0 & 0 \\ 0 & 3 & 0 & -p & 0 \\ 0 & 0 & 3 & 0 & -p \end{pmatrix}$$

We have $\det(A) = \det(S) = -4 * p^3 + 27 * q^2$

In fact, we solve $UP + VQ = C$ with any C such as $\deg(C) < \deg(P) + \deg(Q)$ i.e. we look for U and V such as $\deg(U) < \deg(Q)$ and $\deg(V) < \deg(P)$ (strict inequalities) which make

$UP + VQ = 1$. When the system is a Cramer system, there is a unique solution and, arithmetically speaking, it corresponds to P and Q prime to each other (and reciprocally). So, if $\det(A) = \det(S)$ is not null, U and V exist and are unique, so the two polynomials

$x^3 - p * x + q$ and $3 * x^2 - p$ are prime to each other and reciprocally if the two polynomials $x^3 - p * x + q$ and $3 * x^2 - p$ are prime to each other U and V such as

$\deg(U) < \deg(Q)$ and $\deg(V) < \deg(P)$ exist and are unique so $\det(A) = \det(S)$ is not null.

So if this determinant is null, the two polynomials $x^3 - p * x + q$ and $3 * x^2 - p$ are not prime to each other.

Note:

We have: $\text{discriminant}(P) = \text{resultant}(P, P') / \text{lcoeff}(P)$.

Example of the use of the resultant:

Be F_1 and F_2 two fixed points, and a variable point A on the circle of center F_1 and radius $2a$. We want to find the cartesian equation of the locus of the points M , intersection of F_1A and the perpendicular bisector of F_2A : we have $MF_1 + MF_2 = MF_1 + MA = F_1A = 2a$ then M draws an ellipse of foci F_1 and F_2 and major axis $2a$.

Let us choose as orthonormal basis the one of center F_1 and axis Ox of vector

$\overrightarrow{F_1F_2}$. We have:

$A = (2a \cos(\theta); 2a \sin(\theta))$ where θ is the angle (Ox, OA) . We choose as parameter $t = \tan(\theta/2)$ so that the coordinates of A are a rational function of the parameter t . Then, we have:

$$A = (ax; ay) = \left(2a \frac{1-t^2}{1+t^2}; 2a \frac{2t}{1+t^2} \right)$$

We consider $F_1F_2 = 2c$ and we note I the midpoint of AF_2 . We have:

$F_2 = (2c, 0)$ and

$$I = \left(c + \frac{ax}{2}; \frac{ay}{2} \right) = \left(c + a \frac{1-t^2}{1+t^2}; a \frac{2t}{1+t^2} \right)$$

IM is perpendicular to AF_2 then $M = (x; y)$ checks the equation $eq1 = 0$ with:

$$eq1 := (x - ix) * (ax - 2 * c) + (y - iy) * ay$$

$M = (x; y)$ is on F_1A then M checks the equation $eq2 = 0$ with:

$$eq2 := \frac{y}{x} - \frac{ay}{ax}$$

We have:

$\text{resultant}(eq1, eq2, t)$ is a polynomial $eq3$ of x and y , $eq3$ is independent from t and there are polynomials of t , U and V such as: $U(t) * eq1 + V(t) * eq2 = eq3$.

We enter:

```
ax:=2*a*(1-t^2)/(1+t^2); ay:=2*a*2*t/(1+t^2);
ix:=(ax+2*c)/2; iy:=(ay/2)
eq1:=(x-ix)*(ax-2*c)+(y-iy)*ay
eq2:=y/x-ay/ax
factor(resultant(eq1,eq2,t))
```

We get as resultant:

$$-(64 \cdot (x^2 + y^2) \cdot (x^2 \cdot a^2 - x^2 \cdot c^2 + -2 \cdot x \cdot a^2 \cdot c + 2 \cdot x \cdot c^3 - a^4 + 2 \cdot a^2 \cdot c^2 + a^2 \cdot y^2 - c^4))$$

The factor $-64 \cdot (x^2 + y^2)$ never vanishes, then the equation of the locus is:

$$x^2 a^2 - x^2 c^2 + -2 x a^2 c + 2 x c^3 - a^4 + 2 a^2 c^2 + a^2 y^2 - c^4 = 0$$

By taking as origin of the basis in O midpoint of F_1F_2 , one finds back the cartesian equation of the ellipse. To do this change of origin, we have

$\overrightarrow{F_1M} = \overrightarrow{F_1O} + \overrightarrow{OM}$, so we enter:

```
normal(subst(x^2*a^2-x^2*c^2+-2*x*a^2*c+2*x*c^3-a^4+2*a^2*c^2+a^2*y^2-c^4, [x,y]=[c+X,Y]))
```

We get:

$$-c^2 X^2 + c^2 a^2 + X^2 a^2 - a^4 + a^2 Y^2$$

or else if we put $b^2 = a^2 - c^2$

```
normal(subst(c^2*X^2+c^2*a^2+X^2*a^2-a^4+a^2*Y^2, c^2=a^2-b^2))
```

We get:

$$a^2 \cdot b^2 + a^2 \cdot Y^2 + b^2 \cdot X^2$$

that is to say after division by $a^2 b^2$, M checks the equation:

$$\frac{X^2}{a^2} + \frac{Y^2}{b^2} = 1$$

Another example of use of the resultant

Be $F1$ and $F2$ two fixed points and a variable point A on the circle of center $F1$ and radius $2a$. We want to find the cartesian equation of the envelop of the perpendicular bisector D of $F2A$ (we know that the perpendicular bisector of $F2A$ is tangent to the ellipse of foci $F1$ and $F2$ and major axis $2a$).

Let us choose as orthonormal basis the one of center $F1$ and axis Ox of vector

$\overrightarrow{F1F2}$. We have:

$A = (2a \cos(\theta); 2a \sin(\theta))$ where θ is the angle (Ox, OA) . We choose as parameter $t = \tan(\theta/2)$ so that the coordinates of A are a rational function of parameter t . We have then:

$$A = (ax; ay) = \left(2a \frac{1 - t^2}{1 + t^2}; 2a \frac{2t}{1 + t^2} \right)$$

We consider $F1F2 = 2c$ and we note I the midpoint of $AF2$. We have:

$$F2 = (2c, 0)$$

and

$$I = \left(c + \frac{ax}{2}; \frac{ay}{2} \right) = \left(c + a \frac{1 - t^2}{1 + t^2}; a \frac{2t - t^2}{1 + t^2} \right)$$

D is perpendicular to $AF2$ then D has for equation: $eq1 = 0$ with:

$$eq1 := (x - ix) * (ax - 2 * c) + (y - iy) * ay$$

The envelop of D is then the locus of M intersection of D and D' of equation $eq2 = 0$ with

$$eq2 := \text{diff}(eq1, t).$$

We enter:

```
ax:=2*a*(1-t^2)/(1+t^2); ay:=2*a*2*t/(1+t^2);
ix:=(ax+2*c)/2; iy:=(ay/2)
eq1:=normal((x-ix)*(ax-2*c)+(y-iy)*ay)
eq2:=normal(diff(eq1,t))
factor(resultant(eq1,eq2,t))
```

We get as resultant:

$$\begin{aligned} &(- (64 \cdot a^2)) \cdot (x^2 + y^2) \cdot (x^2 \cdot a^2 - x^2 \cdot c^2 + -2 \cdot x \cdot a^2 \cdot c + 2 \cdot x \cdot c^3 \\ &\quad - a^4 + 2a^2 \cdot c^2 + a^2 \cdot y^2 - c^4) \end{aligned}$$

The factor $-64 \cdot (x^2 + y^2)$ never vanishes then the equation of the locus is:

$$x^2 a^2 - x^2 a c^2 + -2 x a^2 c + 2 x c^3 - a^4 + 2 a^2 c^2 + a^2 c y^2 - c^4 = 0$$

By taking as origin of the basis in O midpoint of $F1F2$, we find as previously the cartesian equation of the ellipse:

$$\frac{X^2}{a^2} + \frac{Y^2}{b^2} = 1$$

7.15.14 Chinese remainder: chinrem

`chinrem` takes as argument two lists having each as components two polynomials eventually supplied by the list of their coefficients by decreasing power order.

`chinrem` returns a list of components of two polynomials.

`chinrem([A, R], [B, Q])` returns the list of polynomials P and S such as:

$$S = R \cdot Q, P = A \pmod{R}, P = B \pmod{Q}$$

There is always a solution P if R and Q are prime to each other, and all the solutions are congruent modulus $S = R \cdot Q$

Find the solutions $P(x)$ of:

$$\begin{cases} P(x) = x & \pmod{x^2 + 1} \\ P(x) = x - 1 & \pmod{x^2 - 1} \end{cases}$$

We enter:

```
chinrem([[1, 0], [1, 0, 1]], [[1, -1], [1, 0, -1]])
```

We get:

```
[[1/-2, 1, 1/-2], [1, 0, 0, 0, -1]]
```

or we enter:

```
chinrem([x, x^2+1], [x-1, x^2-1])
```

We get:

```
[-1/2*x^2+x-1/2, x^4-1]
```

so

$$P(x) = \frac{-x^2 - 2x + 1}{2} \pmod{x^4 - 1}$$

Other example:

We enter:

```
chinrem([[1, 2], [1, 0, 1]], [[1, 1], [1, 1, 1]])
```

We get:

```
[[ -1, -1, 0, 1], [1, 1, 2, 1, 1]]
```

or we enter:

```
chinrem([x+2, x^2+1], [x+1, x^2+x+1])
```

We get:

```
[-x^3-x^2+1, x^4+x^3+2*x^2+x+1]
```

7.16 Special

7.16.1 Cyclotomic polynomial: `cyclotomic`

`cyclotomic` takes as parameter an integer n .

`cyclotomic` returns the list of coefficients of the cyclotomic polynomial of degree n . It is the polynomial whose zeros are all the n -th roots and primitives of the unity (an n -th root of the unity is primitive if its powers generate all the others n -th roots of the unity).

For example, for $n = 4$, the fourth root of the unity are: $\{1, i, -1, -i\}$, and the primitive roots are: $\{i, -i\}$.

So the cyclotomic polynomial of degree four is $(x - i) \cdot (x + i) = x^2 + 1$.

We enter:

```
cyclotomic(4)
```

We get:

```
[1, 0, 1]
```

We enter:

```
cyclotomic(5)
```

We get:

```
[1, 1, 1, 1, 1]
```

So the cyclotomic polynomial of degree 5 is $x^4 + x^3 + x^2 + x + 1$ and we have $(x - 1) * (x^4 + x^3 + x^2 + x + 1) = x^5 - 1$.

We enter:

```
cyclotomic(10)
```

We get:

```
[1, -1, 1, -1, 1]
```

So the cyclotomic polynomial of degree 10 is $x^4 - x^3 + x^2 - x + 1$ and we have $(x^5 - 1) * (x + 1) * (x^4 - x^3 + x^2 - x + 1) = x^{10} - 1$

We enter:

```
cyclotomic(20)
```

We get:

```
[1, 0, -1, 0, 1, 0, -1, 0, 1]
```

So the cyclotomic polynomial of degree 20 is $x^8 - x^6 + x^4 - x^2 + 1$ and we have

$$(x^{10} - 1) * (x^2 + 1) * (x^8 - x^6 + x^4 - x^2 + 1) = x^{20} - 1$$

7.16.2 Groebner basis: `gbasis`

`gbasis` a at least two arguments: a list of polynomials of several variables and the list of the names of these variables.

`gbasis` returns a Groebner basis of the polynomial ideal generated by the polynomials supplied as first argument.

We choose to order the monomials by lexicographical order in accordance with the list supplied as last argument and by decreasing power order: for example we write $x^2 * y^4 * z^3$ then $x^2 * y^3 * z^4$ if the second argument is $[x, y, z]$ because $(2, 4, 3) > (2, 3, 4)$ but we write $x^2 * y^3 * z^4$ then $x^2 * y^4 * z^3$ if the second argument is $[x, z, y]$.

If I is an ideal and if $(G_k)_{k \in K}$ is a Groebner basis of the ideal I , so, if F is a polynomial not null of I , the dominant term of F is divisible by the dominant term of a G_k .

Property: if we do the Euclidean division of F by one of the G_k and then, if we do it again with the rest obtained and the following G_k , we end up with a null rest.

We enter:

```
gbasis([2*x*y-y^2, x^2-2*x*y], [x, y])
```

We get:

$$[y^3, x*y + (-1/2)*y^2, x^2 - y^2]$$

7.16.3 Reduction according to a Groebner basis: `greduce`

`greduce` has three arguments: a polynomial of several variables, a list of polynomials forming a Groebner basis depending on the same variables and the list of the names of these variables.

`greduce` returns the reduction (a multiplicative constant excepted) of the polynomial supplied as first argument according to the Groebner basis supplied as second argument.

We enter:

$$\text{greduce}(x*y-1, [x^2-y^2, 2*x*y-y^2, y^3], [x, y])$$

We get:

$$1/2*y^2-1$$

which means that $xy - 1 = \frac{1}{2}(y^2 - 2) \bmod I$ where I is the ideal generated by the Groebner basis $[x^2 - y^2, 2xy - y^2, y^3]$, because $y^2 - 2$ is the remainder of the Euclidean division of $2(xy - 1)$ by $G_2 = 2xy - y^2$.

Note:

The multiplicative constant can be determined by looking at how the constant coefficient is transformed. In the example, the constant term -1 is transformed into the constant term -2 , then the multiplicative coefficient is $1/2$.

7.16.4 Hermite polynomial: `hermite`

`hermite` takes as argument an integer n and eventually the name of the variable (x by default).

`hermite` returns the Hermite polynomial of degree n .

The Hermite polynomial of degree n written $P(n, x)$ checks the relations:

$$P(0, x) = 1$$

$$P(1, x) = 2x$$

$$P(n, x) = 2xP(n-1, x) - 2(n-1)P(n-2, x)$$

These polynomials are orthogonal for the scalar product:

$$\langle f, g \rangle = \int_{-\infty}^{+\infty} f(x)g(x)e^{-x^2} dx$$

We enter:

$$\text{hermite}(6)$$

We get:

$$64*x^6 - 480*x^4 + 720*x^2 - 120$$

We enter:

$$\text{hermite}(6, y)$$

We get:

$$64*y^6 - 480*y^4 + 720*y^2 - 120$$

7.16.5 Lagrange interpolation: lagrange

lagrange takes as argument two lists of length n or a matrix of two lines and n columns and eventually the name of the variable var (by default x):

the first list (or line) corresponds to values of abscissa x_k , and the second list (or line) corresponds to values of ordinates y_k for k from 1 to n .

lagrange returns a polynomial expression $P(\text{var})$ of degree $n-1$ such as

$$P(x_k) = y_k.$$

We enter:

```
lagrange ([[1,3], [0,1]])
```

Or we enter:

```
lagrange ([1,3], [0,1])
```

We get:

$$1/2 * (x-1)$$

indeed for $x = 1$ we have $\frac{x-1}{2} = 0$ and for $x = 3$ we have $\frac{x-1}{2} = 1$.

We enter:

```
lagrange ([1,3], [0,1], y)
```

We get:

$$1/2 * (y-1)$$

Warning! lagrange ([1,2], [3,4], y) does not return a function but an expression, but we can define a function by putting:

```
f(x) :=lagrange ([1,2], [3,4], x)
```

or

```
f(y) :=lagrange ([1,2], [3,4], y)
```

and so

f(4) returns 6 because $f(x)=x+2$

Please note the difference between:

```
g(x) :=lagrange ([1,2], [3,4])
```

and

```
f(x) :=lagrange ([1,2], [3,4], x) .
```

$g(x) :=lagrange ([1,2], [3,4])$ does not define a function, for example, $g(2)=x-1+3$ whereas $f(2)=4$.

That said, the definition of f is not efficient because the polynomial will be recalculated from the beginning at each call of f (when we define a function the right member is not evaluated, the evaluation is made only when we call f).

To be efficient, you must use unapply:

```
f:=unapply(lagrange ([1,2], [3,4]), x)
```

or

```
f:=unapply(lagrange([1,2],[3,4],y),y)
```

Exercise:

Be $f(x) = \frac{1}{x}$, $x_0 = 2$, $x_1 = 2.5$ and $x_2 = 4$. You are asked to calculate the polynomial L of Lagrange interpolation, and its value in $x = 3$ and $x = 4.5$.

We enter:

```
f(x):=1/x
```

```
L:=unapply(normal(lagrange([2,2.5,4],[f(2),f(2.5),f(4)])),x)
```

We get:

```
x->0.05*x^2-0.425*x+1.15
```

We enter:

```
L(3),L(4.5)
```

We get:

```
0.325,0.25
```

7.16.6 Natural splines: spline

Definition

Be a subdivision σ_n of the interval $[a, b]$:

$$a = x_0, x_1, \dots, x_n = b$$

We say that s is a spline function of degree l if s is an application of $[a, b]$ in \mathbb{R} such as:

- s has continuous derivatives up to degree $l - 1$,
- s restricted at each interval of the subdivision is a polynomial of degree lower than or equals l .

Theorem

The set of splines functions of degree l on σ_n is a \mathbb{R} -vector field of dimension $n + l$.

Indeed:

We $[a, x_1]$, s is a polynomial A of degree lower than or equals l , then on $[a, x_1]$,

$$s = A(x) = a_0 + a_1x + \dots + a_lx^l \text{ and } A \text{ is a linear combination of } 1, x, \dots, x^l.$$

We $[x_1, x_2]$, s is a polynomial B of degree lower than or equals l , then on $[x_1, x_2]$,

$$s = B(x) = b_0 + b_1x + \dots + b_lx^l.$$

Since s has continuous derivatives up to degree $l - 1$ we must have:

$$\forall 0 \leq j \leq l - 1, \quad B^{(j)}(x_1) - A^{(j)}(x_1) = 0$$

$$\text{so } B(x) - A(x) = \alpha_1(x - x_1)^l$$

$$\text{or else } B(x) = A(x) + \alpha_1(x - x_1)^l.$$

Be the function:

$$q_1(x) = \begin{cases} 0 & \text{on } [a, x_1] \\ (x - x_1)^l & \text{on } [x_1, b] \end{cases}$$

Thus:

$$s|_{[a, x_2]} = a_0 + a_1x + \dots + a_lx^l + \alpha_1q_1(x).$$

On $[x_2, x_3]$, s is a polynomial C of degree lower than or equal to l , then on $[x_2, x_3]$,

$$s = C(x) = c_0 + c_1x + \dots + c_lx^l.$$

Since s has continuous derivatives up to degree $l - 1$ we must have:

$$\forall 0 \leq j \leq l - 1, \quad C^{(j)}(x_2) - B^{(j)}(x_2) = 0$$

$$\text{so } C(x) - B(x) = \alpha_2(x - x_2)^l$$

$$\text{or else } C(x) = B(x) + \alpha_2(x - x_2)^l.$$

Be the function:

$$q_2(x) = \begin{cases} 0 & \text{on } [a, x_2] \\ (x - x_2)^l & \text{on } [x_2, b] \end{cases}$$

Thus:

$$s|_{[a, x_3]} = a_0 + a_1 x + \dots + a_l x^l + \alpha_1 q_1(x) + \alpha_2 q_2(x)$$

and so on, we define the functions:

$$\forall 1 \leq j \leq n - 1,$$

$$\forall 1 \leq j \leq n - 1, q_j(x) = \begin{cases} 0 & \text{on } [a, x_j] \\ (x - x_j)^l & \text{on } [x_j, b] \end{cases}$$

thus,

$$s|_{[a, b]} = a_0 + a_1 x + \dots + a_l x^l + \alpha_1 q_1(x) + \alpha_{n-1} q_{n-1}(x)$$

and s is a linear combination of $n + l$ independant functions $1, x, \dots, x^l, q_1, \dots, q_{n-1}$.

Interpolation with splines functions

We can ask to interpolate a function f on σ_n by a spline function s of degree l , which forces s to check $s(x_k) = y_k = f(x_k)$ for all $0 \leq k \leq n$.

We have then $n + 1$ conditions, then remain $l - 1$ degrees of freedom. So can we still impose $l - 1$ additional conditions which will be conditions on the derivatives of s in a and b . There are then three kinds of interpolation (Hermite interpolation, natural interpolation, periodic interpolation) which are obtained by adding three kind of constraints. We can show that for each of these kinds of interpolation the solution to the interpolation problem is unique.

Let us assume l odd, $l = 2m - 1$, there are then $2m - 2$ degrees of freedom. We add the following constraints:

- Hermite interpolation

$$\forall 1 \leq j \leq m - 1, \quad s^{(j)}(a) = f^{(j)}(a), s^{(j)}(b) = f^{(j)}(b)$$

- natural interpolation

$$\forall m \leq j \leq 2m - 2, \quad s^{(j)}(a) = s^{(j)}(b) = 0$$

- periodic interpolation

$$\forall 1 \leq j \leq 2m - 2, \quad s^{(j)}(a) = s^{(j)}(b)$$

Let us assume l even, $l = 2m$, there are then $2m - 1$ degrees of freedom. We add the following constraints:

- Hermite interpolation

$$\forall 1 \leq j \leq m - 1, \quad s^{(j)}(a) = f^{(j)}(a), s^{(j)}(b) = f^{(j)}(b)$$

and

$$s^{(m)}(a) = f^{(m)}(a)$$

- natural interpolation

$$\forall m \leq j \leq 2m - 2, \quad s^{(j)}(a) = s^{(j)}(b) = 0$$

and

$$s^{(2m-1)}(a) = 0$$

- periodic interpolation

$$\forall 1 \leq j \leq 2m - 1, \quad s^{(j)}(a) = s^{(j)}(b)$$

A natural spline of supplied degree passing by two supplied points is a spline function that makes the natural interpolation.

The instruction `spline` returns a natural spline of supplied degree passing by points whose lists of abscissae by increasing order and ordinates are supplied has argument. It returns the spline function under the form of a list of polynomials, each polynomial being valide in an interval. We give the list of abscissae in increasing order, the list of ordinates, the names of variables wished for the polynomials, and the degree.

For instance, we want a natural spline of degree 3, passing by the points $x_0 = 0, y_0 = 1, x_1 = 1, y_1 = 3$ and $x_2 = 2, y_2 = 0$.

We enter:

```
spline([0, 1, 2], [1, 3, 0], x, 3)
```

We get a list of two polynomials function of x :

$$\left[-\frac{5 * x^3}{4} + \frac{13 * x}{4} + 1, \quad \frac{5 * (x - 1)^3}{4} - \frac{15 * (x - 1)^2}{4} + \frac{(x - 1)}{-2} + 3 \right]$$

valid respectively on the intervals $[0, 1]$ and $[1, 2]$.

For instance, we want a natural spline of degree 4, passing by the points $x_0 = 0, y_0 = 1, x_1 = 1, y_1 = 3, x_2 = 2, y_2 = 0$ and $x_3 = 3, y_3 = -1$, we enter:

```
spline([0, 1, 2, 3], [1, 3, 0, -1], x, 4)
```

We get a list of three polynomials function of x :

$$\left[\frac{-62 * x^4 + 304 * x}{121} + 1, \right. \\ \left. \frac{201 * (x - 1)^4 - 248 * (x - 1)^3 - 372 * (x - 1)^2 + 56 * (x - 1)}{121} + 3, \right. \\ \left. \frac{-139 * (x - 2)^4 + 556 * (x - 2)^3 + 90 * (x - 2)^2 - 628 * (x - 2)}{121} \right]$$

valid respectively on the intervals $[0, 1]$, $[1, 2]$ and $[2, 3]$.

For instance, to get the natural interpolation of \cos on $\left[0, \frac{\pi}{2}, \frac{3\pi}{2}\right]$

We enter:

```
spline([0, pi/2, 3*pi/2], cos([0, pi/2, 3*pi/2]), x, 3)
```

We get:

$$\left[\frac{(3\pi^3 + (-7\pi^2)x + 4x^3)\left(\frac{1}{3}\right)}{\pi^3}, \frac{(15\pi^3 + (-46\pi^2)x + 36\pi x^2 - 8x^3)\left(\frac{1}{12}\right)}{\pi^3} \right]$$

7.16.7 Laguerre polynomial: laguerre

`laguerre` takes as argument an integer n and eventually the name of the variable (x by default) and the parameter (a by default).

`laguerre` returns the Laguerre polynomial of degree n and parameter a .

The Laguerre polynomial of degree n of parameter a written $L(n, a, x)$ checks the relations:

$$L(0, a, x) = 1 \\ L(1, a, x) = 1 + a - x \\ L(n, a, x) = \frac{2n + a - 1 - x}{n} L(n - 1, a, x) - \frac{n + a - 1}{n} L(n - 2, a, x)$$

These polynomials are orthogonal for the scalar product:

$$\langle f, g \rangle = \int_0^{+\infty} f(x)g(x)x^a e^{-x} dx$$

We enter:

```
laguerre(2)
```

We get:

$$1/2 * a^2 - a * x + 3/2 * a + 1/2 * x^2 - 2 * x + 1$$

We enter:

```
laguerre(2, y)
```

We get:

$$1/2 * a^2 - a * y + 3/2 * a + 1/2 * y^2 - 2 * y + 1$$

b must be purged ($b := 'b'$), we enter:

```
laguerre(2, y, b)
```

We get:

$$1/2*b^2-b*y+3/2*b+1/2*y^2-2*y+1$$

7.16.8 Legendre polynomial: `legendre`

`legendre` takes as argument an integer n and eventually the name of the variable (x by default). `legendre` returns the Legendre polynomial of degree n : it is the polynomial, not null, solution of the differential equation:

$$(x^2 - 1).y'' - 2.x.y' - n(n + 1).y = 0$$

The Legendre polynomial of degree n written $P(n, x)$ checks the relations:

$$P(0, x) = 1$$

$$P(1, x) = x$$

$$P(n, x) = \frac{2n-1}{n}xP(n-1, x) - \frac{n-1}{n}P(n-2, x)$$

These polynomials are orthogonal for the scalar product:

$$\langle f, g \rangle = \int_{-1}^{+1} f(x)g(x)dx$$

We enter:

```
legendre(4)
```

We get:

$$35/8*x^4-15/4*x^2+3/8$$

We enter:

```
legendre(4, y)
```

We get:

$$35/8*y^4-15/4*y^2+3/8$$

7.16.9 Tchebyshev polynomial of first kind: `tchebyshev1`

`tchebyshev1` takes as argument an integer n and eventually the name of the variable (x by default). `tchebyshev1` returns the Tchebyshev polynomial of first kind, of degree n , written $T(n, x)$.

We have:

$$T(n, x) = \cos(n.\arccos(x))$$

$T(n, x)$ checks the relations:

$$T(0, x) = 1$$

$$T(1, x) = x$$

$$T(n, x) = 2xT(n-1, x) - T(n-2, x)$$

The polynomials $T(n, x)$ are orthogonal for the scalar product:

$$\langle f, g \rangle = \int_{-1}^{+1} \frac{f(x)g(x)}{\sqrt{1-x^2}} dx$$

We enter:

```
tchebyshev1(4)
```

We get:

$$8*x^4-8*x^2+1$$

We enter:

```
tchebyshev1 (4, y)
```

We get:

$$8*y^4-8*y^2+1$$

and we do have:

$$\cos(4.x) = \operatorname{Re}((\cos(x) + i.\sin(x))^4)$$

$$\cos(4.x) = \cos(x)^4 - 6.\cos(x)^2.(1 - \cos(x)^2) + (1 - \cos(x)^2)^2$$

$$\cos(4.x) = T(4, \cos(x))$$

7.16.10 Tchebyshev polynomial of second kind: tchebyshev2

tchebyshev2 takes as argument an integer n and eventually the name of the variable (x by default). tchebyshev2 returns the Tchebyshev polynomial of second kind, of degree n , written $U(n, x)$.

We have:

$$U(n, x) = \frac{\sin((n + 1).\arccos(x))}{\sin(\arccos(x))}$$

or else

$$\sin((n + 1)x) = \sin(x) * U(n, \cos(x))$$

$U(n, x)$ checks the relations:

$$U(0, x) = 1$$

$$U(1, x) = 2x$$

$$U(n, x) = 2xU(n - 1, x) - U(n - 2, x)$$

Polynomials $U(n, x)$ are orthogonal for the scalar product:

$$\langle f, g \rangle = \int_{-1}^{+1} f(x)g(x)\sqrt{1-x^2}dx$$

We enter:

```
tchebyshev2 (3)
```

We get:

$$8*x^3-4*x$$

We enter:

```
tchebyshev2 (3, y)
```

We get:

$$8*y^3-4*y$$

indeed:

$$\sin(4.x) = \sin(x) * (8 * \cos(x)^3 - 4.\cos(x)) = \sin(x) * U(3, \cos(x)).$$

Chapter 8 Menu Plot

Note:

The plot of most of the commands starting by `plot` is not well done from the CAS screen: you will then preferably use the geometry application to do the plots corresponding to these commands.

8.1 Plot of a function: `plotfunc`

We open the geometry application and we press the key `Plot`. Then, we use `Cmds` of the push buttons. We choose `6 Plot` then `1 Function`. `plotfunc(` shows on the entry line and it is enough to fill up with the expression $f(x)$ which we want to get the plot of.

The Symbolic view stores then an additional line containing the command entered for example:

```
plotfunc(f(x))
```

```
plotfunc(f(x), x) plots the graphical representation of  $y = f(x)$  and plotfunc(f(x), x=a..b)
```

```
plots the graphical representation of  $y = f(x)$ 
```

```
when  $a \leq x \leq b$ .
```

We enter:

```
plotfunc(x^2-2)
```

or

```
plotfunc(a^2-2, a=-1..2)
```

We get:

```
the graphical representation of  $y=x^2-2$ 
```

Or else, we enter:

```
gf:=plotfunc(x^2-2)
```

Then, in the Symbolic view of the geometry application (Symb), we press `New`, which displays, for example:

```
√GC:=
```

We fill up with

```
√GC:=gf
```

Then, we press the key `Plot` to get the Plot view of the geometry application, we get:

```
the plot of  $x^2-2$ 
```

8.2 Parametric curve: `plotparam`

```
plotparam(f(t)+i*g(t), t) (resp. plotparam(f(t)+i*g(t), t=t1..t2))
```

```
plots the parametric representation of the curve defined by  $x = f(t)$ ,  $y = g(t)$ 
```

```
(resp. by  $x = f(t)$ ,  $y = g(t)$  and  $t1 \geq t \geq t2$ ).
```

We enter:

```
plotparam(cos(x)+i*sin(x),x)
```

or

```
plotparam([cos(x),sin(x)],x)
```

We get:

```
The plot of the circle unity
```

We can specify the boundaries of the interval of variation of the parameter.

We enter:

```
plotparam(sin(t)+i*cos(t),t=-4..1)
```

or else:

```
plotparam(sin(x)+i*cos(x),x=-4..1)
```

Or we enter, if in the plot configuration t goes from -4 to 1 :

```
plotparam(sin(t)+i*cos(t))
```

We get:

```
The plot of the arc of circle unity starting from -4 at 1
```

We can add a parameter to specify the step of sampling of the parameter t with $tstep=$, that is to say the step t we want to use to do the plot.

We enter, if in the plot configuration t goes from -4 to 1 :

```
plotparam(sin(t)+i*cos(t),t,tstep=0.5)
```

Or we enter:

```
plotparam(sin(t)+i*cos(t),t=-4..1,tstep=0.5)
```

We get:

```
The raw plot of the arc of circle unity starting from -4 to 1
```

8.3 Polar curve: `plotpolar`

`plotpolar(f(t),t)` plots the Polar representation of the curve defined by:
 $\rho = f(t)$.

We enter:

```
plotpolar(t,t=0..10)
```

We enter, if in the plot configuration t goes from 0 to 10 :

```
plotpolar(t,t)
```

We get:

The spiral $\rho=t$ is plotted

We can add a parameter to specify the step of sampling of the parameter t with `tstep=`, that is to say the step t we want to use to do the plot.

We enter, in the plot configuration t goes from 0 to 10:

```
plotpolar(t,t,tstep=1)
```

or:

```
plotpolar(t,t=0..10,tstep=1)
```

We get:

The spiral $\rho=t$ is plotted roughly.

8.4 Plot of a recurrent sequence: `plotseq`

`plotseq(f(x),a,n)` or `plotseq(f(t),t=a,n)` allows to display the n first terms of a recurrent sequence defined by:

$$u_0 = a, a = f(u_{n-1})$$

We enter:

```
plotseq(sqrt(1+x),3,5)
```

We get:

The plot of $y=\sqrt{1+x}$, of $y=x$ and the five first terms
of the sequence $u_0=3$ and $u_n=\sqrt{1+u_{n-1}}$

8.5 Implicit plot in 2D: `plotimplicit`

`plotimplicit` allows to plot curves defined in an implicit way by an expression. In order to not have the calculator starting to factorize the expression, the command `plotimplicit` can be used with the option `unfactored` put as last parameter:

- with `unfactored`, the expression will not be modified,
- without `unfactored`, the calculator factors the expression to the same denominator and then tries to factorize the numerator.
- `plotimplicit(f(x,y),x,y)` or `plotimplicit(f(x,y),[x,y])` plots the graphical representation of the curve defined implicitly by $f(x,y) = 0$ when x (resp. y) varies according to $WX -$, $WX +$ (resp. $WY-$, $WY +$) defined in `cfg`,
- `plotimplicit(f(x,y),x=0..1,y=-1..1)`
or
`plotimplicit(f(x,y),[x=0..1,y=-1..1])` plots the graphical representation of the curve defined implicitly by $f(x,y) = 0$ when $0 \leq x \leq 1$ and $-1 \leq y \leq 1$ (set the boundaries slightly larger to avoid losing part of the plot!).

We enter:

```
plotimplicit(x^2+y^2-1,[x,y])
```

Or we enter:

```
plotimplicit(x^2+y^2-1, {x, y}, unfactored)
```

Or we enter:

```
plotimplicit(x^2+y^2-1, x, y, unfactored)
```

We get:

```
circle(point(0,0),1)
```

We enter:

```
g:=plotimplicit(x^2+y^2-1, [x, y])
```

Then, in the Symbolic view of the geometry application (`Symb`), we press `New`, which displays, for example:

```
√GC:=
```

We fill up with

```
√GC:=g
```

Then, in the Plot view of the geometry application (`Plot`), we get:

```
the plot of the circle unity
```

8.6 Plot of a function by colors levels: `plotdensity`

`plotdensity(f(x, y), [x, y])` plots the graph of $z = f(x, y)$ in the plane by representing z by one of the colors of the rainbow.

We enter:

```
plotdensity(x^2-y^2, [x=-2..2, y=-2..2], xstep=0.1, ystep=0.1)
```

We get:

```
A 2D-plot representing, for each z, the hyperbola defined by x^2-
y^2=z
of one of the colors of the rainbow.
```

We notice that we have the color scale below the plot.

8.7 The field of tangents: `plotfield`

We can plot the field of tangents of the differential equation $y' = f(t, y)$ or of the system of differential equations $x' = u(x, y), y' = v(x, y)$ and we can specify the ranges of values of the parameters.

- Be $f(t, y)$ an expression depending on two variables t and y , then `plotfield(f(t, y), [t, y])` plots the field of tangents of the differential equation $y' = f(t, y)$ where y represents a real variable and t is represented in abscissa,
- Be $V = [u(x, y), v(x, y)]$ a 2D-vector of coordinates two expressions depending on two variables x, y , but not depending from the time, so `plotfield(V, [x, y])` plots the field of tangents of the system $[x'(t) = u(x, y), y'(t) = v(x, y)]$,

- The ranges of values of t, y or of x, y can be specified by $t=tmin..tmax$, $x=xmin..xmax$, $y=ymin..ymax$ instead of the name of the variable only.
- We can specify the plot view by putting, for example:
`plotfield(f(t,y), [t=tmin..tmax, y=ymin..ymax])`
- We can specify that the field of tangents is, in an orthonormal basis, of norm 1 with the option `normalize`. Without the option `normalize` the contact point is the origin of the vector tangent, and with the option `normalize`, the contact point is located at the midpoint of the tangent lines.
- We can also specify the value of the step in t and in y with `xstep=...` and `ystep=...`

We enter:

```
plotfield(4*sin(t*y), [t=0..2, y=-3..7])
```

We get:

Segments of slope $4*\sin(t*y)$ are plotted in different points.
These lines represent the vectors tangent oriented toward the increasing t and whose origin is the point of contact.

We enter:

```
plotfield(4*sin(t*y), [t=0..2, y=-3..7], normalize,  
xstep=0.7, ystep=0.7)
```

We get:

Segments of length 1 and slope $4*\sin(t*y)$
Representing the tangent lines at their midpoint.
These points are spaced of 0.7.

We enter:

```
plotfield(5*[-y,x], [x=-1..1, y=-1..1])
```

We get:

vectors $[-y, x]$ are plotted at points (x, y) .
These vectors represent vectors tangents at their origin to curves solutions of the system $x'(t) = -y$, $y'(t) = x$.
They are orientated toward the increasing t .

We enter:

```
plotfield(5*[-y,x], [x=-1..1, y=-1..1], normalize)
```

We get:

Segments of length 1 and slope $-y/x$
representing the lines tangent at their midpoint
to curves solutions of the system $x'(t) = -y$, $y'(t) = x$.

8.8 Level curves: `plotcontour`

`plotcontour(f(x,y), [x,y])` plots the level curves $z = -10, z = -8, \dots, z = 0, z = 2, \dots, z = 10$ of the surface defined by $z = f(x,y)$.

We enter:

```
g:=plotcontour(x^2+y^2, [x=-3..3, y=-3..3], [1,2,3],
  display=[green,red,black]+[filled$3])
```

We get:

```
[polygon(point(...))..]
```

Then, in the Symbolic view of the geometry application (`Symb`), we press `New`, which displays, for example:

$$\sqrt{GC}:=$$

We fill up with

$$\sqrt{GC}:=g$$

Then, in the Plot view of the geometry application (`Plot`), we get:

```
the plot of three ellipses x^2-y^2=n for n=1,2,3;
```

the areas in between these ellipses are filled with the color green, red or black.

We enter:

```
plotcontour(x^2-y^2, [x,y])
```

We get:

```
[polygon(point(-4.8,-5),point(-3.9,-4)...)]
```

Then, in the Symbolic view of the geometry application (`Symb`), we press `New`, which displays, for example:

$$\sqrt{GC}:=$$

We fill up with

$$\sqrt{GC}:=g$$

Then, in the Plot view of the geometry application (`Plot`), we get:

```
the plot of 11 hyperbolae x^2-y^2=n for n=-10,-8,..10
```

8.9 Plot of solutions of a differential equation: `plotode`

We can plot the solutions of the differential equation $y' = f(t,y)$ or of the system of differential equations $x' = u(t,x,y), y' = v(t,x,y)$ and we can specify the ranges of values of the parameters.

- `plotode(f(t,y), [t,y], [t0,y0])` plots according to the time the solution $y(t)$ of the differential equation $y' = f(t,y)$ passing by the point (t_0, y_0) , where $f(t,y)$ designates an expression depending on the time variable t and the variable y .
- By default, t varies along the two directions. We can specify the range of the time by the optional parameter `t=tmin..tmax`.

- When $y = (X, Y)$ is a vector of length 2 and f has values in \mathbb{R}^2 , we can also represent in the space (t, X, Y) or in the plane (X, Y) the solution of a differential equation $y' = f(t, y)$, that is to say $[X', Y'] = [f(t, X, Y)]$. For that, it is enough to replace y by the names of variables X, Y and the initial value by the two initial values of the variables at time t_0 .

We enter:

```
plotode(sin(t*y), [t, y], [0, 1])
```

We get:

The graph of the solution of $y' = \sin(t, y)$ passing by the point $(0, 1)$.

To display the values of the solution, please refer to section 4.7.

We enter:

```
plotfield(5*[-y, x], [x=-1..1, y=-1..1], normalize)
plotode(5*[-y, x], [t=0..1, x, y], [0, 0.3, 0.7], tstep=0.05, plan)
```

We get:

The graph of the solution of $x' = -y, y' = x$ for $t=0$
passing by the point $(0.3, 0.7)$

8.10 Polygonal line: `plotlist`

`plotlist` takes as argument a list l or a matrix of two columns.

`plotlist` allows to display the segments connecting the cloud of dots having for abscissa $[0, 1, 2, \dots, n]$ and for ordinate l , or for coordinates a line of the matrix. `listplot` or `plotlist` connects by two line segments the different points of the cloud, but without reorder the points unlike `polygonplot` which reorders the points according to their abscissa then connects them.

We enter:

```
a:=plotlist([0, 1, 4, 9, 16])
```

Or we enter:

```
a:=plotlist([[0, 0], [1, 1], [2, 4], [3, 9], [4, 16]])
```

Then, in the Symbolic view of the geometry application (`Symb`), we press `New`, which displays, for example:

$$\sqrt{GD} :=$$

We fill up with

$$\sqrt{GD} := a$$

Then, in the Plot view of the geometry application (`Plot`), we get:

the plot of 5 points $((0, 0), (1, 1), \dots, (4, 16))$ joined by 4 segments

Warning!

```
plotlist([0,1,2,3,4],[0,1,4,9,16])  
or  
plotlist([[0,1,2,3,4],[0,1,4,9,16]])  
is not valid!
```

Part III The menu MATH of the Toolbox key

Chapter 9 Functions on reals

9.1 HOME constants

- `e` for `exp(1)`
- `i` for the complex number of modulus 1 and argument $\pi/2$ or $(0,1)$
- `MAXREAL` it is $+\infty$
- `MINREAL` it is 0
- `Pi` or `pi` or `PI` for π

9.2 The symbolic constants of the CAS: `e pi i infinity inf euler_gamma`

`e` or `%e` designates the number `exp(1)`;
`pi` or `%pi` designates the number π ;
`infinity` designates ∞ .
`+infinity` or `inf` designates $+\infty$.
`-infinity` or `-inf` designates $-\infty$.
`i` or `%i` designates the complex number i .
`euler_gamma` designates the Euler constant.

We have:

```
euler_gamma=limit(sum(1/k,k,1,n)-ln(n),n,+infinity)
```

and

```
evalf(euler_gamma) returns 0.577215664902
```

9.3 Booleans

9.3.1 Boolean values: `true false`

A boolean takes as value `true` or `false`.

We have the following synonyms:

`true` or `TRUE` or `1` and,

`false` or `FALSE` or `0`.

In HOME, `TRUE` is replaced in the history by `1`, and `FALSE` by `0`.

The tests or the conditions are Boolean functions.

9.3.2 Tests: `== != > >= < <=`

`==, !=, >, >=, <, <=` are infix operators.

`>=, <=, !=` are obtained with the keys \geq, \leq and \neq .

`a==b` tests the equality between `a` and `b` and returns `1` if `a` equals `b` and `0` otherwise.

`a!=b` returns `1` if `a` is different from `b` and `0` otherwise.

`a>=b` returns `1` if `a` is greater than or equals `b` and `0` otherwise.

`a>b` returns `1` if `a` is strictly greater than `b` and `0` otherwise.

`a<=b` returns `1` if `a` is lower than or equals `b` and `0` otherwise.

`a<b` returns `1` if `a` is strictly lower than `b` and `0` otherwise.

To define the boolean function which is `true` on $]0; +\infty[$ and `false` on $] - \infty; 0]$, we enter:

```
f(x) := ifte(x>0, true, false)
```

We enter:

```
f(0) == 0
```

We get:

```
1
```

Warning!

`a=b` is not a boolean!

To test the equality between `a` and `b`, you have to write `a==b`.

9.3.3 Boolean operators: `or` `xor` and `not`

`or` (`or ||`), `xor`, and (`or &&`) are infix operators.

`not` is a prefix operator.

Let `a` and `b` be two booleans: (`a or b`) or (`a || b`) returns 0 (or false) if `a` and `b` equal 0 and returns 1 (or true) if not.

(`a xor b`) returns 1 if `a` equals 1 and `b` equals 0 or if `a` equals 0 and `b` equals 1 and returns 0 if `a` and `b` equal 0 or if `a` and `b` equal 1 (it is the "exclusive or").

(`a and b`) or (`a && b`) returns 1 (or true) if `a` and `b` equal 1 and 0 (or false) if not.

`not(a)` returns 1 (or true) if `a` equals 0 (or false), and 0 (or false) if `a` equals 1 (or true).

We enter:

```
1 >= 0 or 1 < 0
```

We get:

```
1
```

We enter:

```
1 >= 0 xor 1 > 0
```

We get:

```
0
```

We enter:

```
1 >= 0 and 1 > 0
```

We get:

```
1
```

We enter:

```
not(0 == 0)
```

We get:

```
0
```

9.4 Bit to bit operators

9.4.1 operators `bitor`, `bitxor`, `bitand`

The integers can be entered with the notation `0x...` in hexadecimal. By example, `0x1f` represents $16 + 15 = 31$ in decimal. We can display the integers in hexadecimal (key of the status line of the CAS with the key Base (Integers)).

- `bitor` is the inclusive logic bit to bit or.

We enter:

```
bitor(0x12,0x38)
```

or we enter:

```
bitor(18,56)
```

We get:

```
58
```

indeed:

18 reads `0x12` in basis 16 and `0b010010` in basis 2,

56 reads `0x38` in basis 16 and `0b111000` in basis 2,

`bitor(18,56)` reads `0b111010` in basis 2 and then equals 58.

- `bitxor` is the logic exclusive bit to bit or.

We enter:

```
bitxor(0x12,0x38)
```

or we enter:

```
bitxor(18,56)
```

We get:

```
42
```

indeed:

18 reads `0x12` in basis 16 and `0b010010` in basis 2,

56 reads `0x38` in basis 16 and `0b111000` in basis 2,

`bitxor(18,56)` reads `0b101010` in basis 2 and then equals 42.

- `bitand` is the bit to bit logic and.

We enter:

```
bitand(0x12,0x38)
```

or we enter:

```
bitand(18,56)
```

We get:

16

indeed:

18 reads 0x12 in basis 16 and 0b010010 in basis 2,

56 reads 0x38 in basis 16 and 0b111000 in basis 2,

bitand(18,56) reads 0b010000 in basis 2 and then equals 16.

9.4.2 Bit to bit Hamming distance of: `hamdist`

The bit to bit Hamming distance is the sum of absolute values of bit to bit differences between two numbers, that is to say the number of different bits.

We enter:

```
hamdist(0x12,0x38)
```

or we enter

```
hamdist(18,56)
```

We get:

3

indeed:

18 reads 0x12 in basis 16 and 0b010010 in basis 2,

56 reads 0x38 in basis 16 and 0b111000 in basis 2,

`hamdist(18,56)` equals 1+0+1+0+1+0 and then equals 3.

9.5 Usual functions

The usual functions can be accessed by pressing the corresponding keys.

We need their programming names, for example:

- `id` designates the function identity,
- `sq` designates the function square,
- `sqrt` designates the function square root,
- `neg` designates the function $x \rightarrow -x$,
- `inv` designates the function $x \rightarrow \frac{1}{x}$.

9.6 The smallest integer greater than or equal to the argument: `CEILING` `ceiling`

`CEILING(a)` or `ceiling(a)` returns the smallest integer greater than or equal to the argument a .

We enter:

```
CEILING(45/8)
```

We get:

6

We enter:

```
CEILING(-45/8)
```

We get:

-5

We enter:

CEILING(2.5)

We get:

3

9.7 Integer part of a real: FLOOR floor

FLOOR(a) or floor(a) returns the greater integer lower than or equal to the argument a .

We enter:

FLOOR(45/8)

We get:

5

We enter:

FLOOR(-45/8)

We get:

-6

We enter:

FLOOR(2.5)

We get:

2

9.8 Argument without its fractional part: IP

IP(a) returns the argument real a without its fractional part.

We enter:

IP(45/8)

We get:

5

We enter:

IP(-45/8)

We get:

-5

9.9 Fractional part: FP

$\text{FP}(a)$ returns the fractional part of the real argument a .

We enter in:

$$\text{FP}(45/8)$$

We get:

$$0.625$$

We enter in the CAS:

$$\text{FP}(45/8)$$

We get:

$$5/8$$

We enter in:

$$\text{FP}(-45/8)$$

We get:

$$-0.625$$

We enter in the CAS:

$$\text{FP}(-45/8)$$

We get:

$$-5/8$$

9.10 Round a real or a complex to n decimal places: ROUND round

$\text{ROUND}(a)$ or $\text{round}(a)$ (resp. $\text{ROUND}(a, n)$ or $\text{round}(a, n)$) rounds the real a according to the closest integer (resp. the closest decimal number having n decimal places).

$\text{ROUND}(a)$ (resp. $\text{ROUND}(a, n)$) rounds the complex a according to the closest element of $\mathbb{Z}[i]$, (resp. with n decimal places).

We enter:

$$\text{ROUND}(45/8)$$

or

$$\text{round}(45/8)$$

We get:

$$6$$

We enter:

$\text{ROUND}(45/8, 2)$

or

 $\text{round}(45/8, 2)$

We get:

5.63

We enter:

 $\text{ROUND}(-45/8)$

or

 $\text{round}(-45/8)$

We get:

-6

We enter:

 $\text{ROUND}(-45/8, 2)$

or

 $\text{round}(-45/8, 2)$

We get:

-5.62

We enter:

 $\text{ROUND}(0.5+i*\text{pi})$

or

 $\text{round}(0.5+i*\text{pi})$

We get:

 $1+3*i$

We enter:

 $\text{ROUND}(0.5+i*\text{pi}, 4)$

or

 $\text{round}(0.5+i*\text{pi}, 4)$

We get:

 $0.5+3.1416*i$

9.11 Truncate a real or a complex to n decimal places: TRUNCATE trunc

TRUNCATE or trunc returns the argument truncated to n decimal places (by default $n = 0$).

We enter in HOME or in the CAS:

```
TRUNCATE (45/8)
```

or

```
trunc (45/8)
```

We get:

```
5
```

We enter in HOME or in the CAS:

```
TRUNCATE (45/8, 2)
```

or

```
trunc (45/8, 2)
```

We get:

```
5.62
```

We enter in HOME or in the CAS:

```
TRUNCATE (-45/8)
```

or

```
trunc (-45/8)
```

We get:

```
-5
```

We enter in HOME or in the CAS:

```
TRUNCATE (-45/8, 2)
```

or

```
trunc (-45/8, 2)
```

We get:

```
-5.62
```

We enter in HOME or in the CAS:

```
TRUNCATE (sqrt(2)+i*sqrt(5), 4)
```

or

```
trunc (sqrt(2)+i*sqrt(5), 4)
```

We get:

$$1.4142+2.236*i$$

Warning! In CAS, `truncate(P,n)` truncates the polynomial P at degree n .

We enter:

$$\text{truncate}(x^5+x^4+x^2+x+1,2)$$

We get:

$$x^2+x+1$$

9.12 The fractional part of a real: `frac`

`frac` of a real returns its fractional part.

We enter in the CAS or in HOME:

$$\text{frac}(22/7)$$

We get:

$$1/7$$

We enter in the CAS or in HOME:

$$\text{frac}(\text{sqrt}(2))$$

We get:

$$\text{sqrt}(2)-1$$

9.13 The real without its fractional part: `iPart`

`iPart` of a real returns a real which equals the real argument without its fractional part.

We enter:

$$\text{iPart}(\text{sqrt}(2))$$

We get:

$$1.0$$

9.14 Mantissa of a real: `MANT`

`MANT(a)` returns $|a|/10^n$ where the integer n checks:

$$10^n \leq |a| < 10^{n+1}.$$

`MANT(a)` then returns the mantissa of a real a , that is to say the significant digits of a .

We enter:

$$\text{MANT}(45/8)$$

We get:

$$5.625$$

200

We enter:

MANT (-45/8)

We get:

5.625

9.15 Integer part of the logarithm basis 10 of a real: XPON

XPON (a) returns the integer n such as $10^n \leq |a| < 10^{n+1}$.

We enter:

XPON (45/8)

We get:

0

We enter:

XPON (45000/8)

We get:

3

We enter:

XPON (1234*sqrt(2))

We get:

3

indeed $10^3 < 1234 * \sqrt{2} \approx 1745.13953597 < 10^4$

Chapter 10 Arithmetic

10.1 Maximum of two or several values: MAX max

MAX or max returns the maximum of elements of a sequence or of a list of reals.

We enter:

```
MAX(4, 5, 8, 2, 6)
```

or

```
max(4, 5, 8, 2, 6)
```

We get:

8

10.2 Minimum of two or several values: MIN min

MIN or min returns the minimum of elements of a sequence or of a list of reals.

We enter:

```
MIN(4, 5, 8, 2, 6)
```

or

```
min(4, 5, 8, 2, 6)
```

We get:

2

10.3 MOD

MOD is an infix function.

$a \text{ MOD } b$ returns the remainder of the Euclidean division of a by b .

We enter:

```
22 MOD 5
```

We get:

2

10.4 FNROOT

`FNROOT` returns an approximate root of the expression supplied as first argument, for the variable supplied as second argument, and which is close to the third argument.

We enter in real mode (*i* not checked in the CAS Settings):

```
FNROOT (x^4+3x-4)
```

We get:

```
-1.74295920217, 1.
```

We enter in real mode (*i* not checked in the CAS Settings):

```
FNROOT (x^4+3x-4, x, -2)
```

We get:

```
-1.74295920217
```

We enter in complex mode (*i* checked in the CAS Settings):

```
FNROOT (x^4+3x-4)
```

We get:

```
[-1.74295920217, 0.371479601083+1.46865601291*i, 0.371479601083-  
1.46865601291*i, 1.0]
```

10.5 N-th root: NTHROOT surd

`NTHROOT` is an infix function whereas `surd` is a prefix command of the CAS .

`NTHROOT` comes with the shifted key $\sqrt[n]{}$ (Shift x^y).

p `NTHROOT` n returns the value of $n^{1/p}$

`surd(n, p)` returns $n^{1/p}$

We enter in real mode in HOME (*i* not checked in the CAS Settings):

```
3 NTHROOT 8
```

We get:

```
2
```

We enter in complex mode in HOME (*i* checked in the CAS Settings):

```
3 NTHROOT -1+i
```

We get:

```
0.85502540378-0.5*i
```

We enter in real mode in the CAS (*i* not checked in the CAS Settings):

```
3 NTHROOT 8
```

or

```
surd(8,3)
```

We get:

```
2
```

We enter in complex mode in the CAS:

```
3 NTHROOT -1+i
```

or

```
surd(-1+i,3)
```

We get:

```
exp(ln(-1+i)/3)
```

10.6 %

`% (a, b)` returns $\frac{a}{100} * b$ (a percent of b).

We enter:

```
% (5, 70)
```

We get:

```
3.5
```

We enter:

```
% (5, 90)
```

We get:

```
4.5
```

10.7 Complex

10.7.1 The key `i`

The key `i` is a shifted key (Shift 2).

`i` is the complex number of modulus 1 and argument $\pi/2$.

We enter:

```
1+3*i
```

or

```
1+3i
```

We get:

```
1+3*i
```

10.7.2 Argument: ARG arg

ARG or arg returns the argument of the complex number supplied as argument (in degrees or in radians according to the chosen configuration).

We enter in HOME:

```
ARG(2+6*i)
```

We get, if we are in radians:

```
1.2490457724
```

We get, if we are in degrees:

```
71.5650511771
```

We enter in the CAS:

```
ARG(2+6*i)
```

We get, if we are in radians:

```
atan(3)
```

We get, if we are in degrees:

```
71.5650511771
```

10.7.3 Conjugate: CONJ conj

CONJ or conj returns the conjugate of the complex number supplied as argument.

We enter:

```
CONJ(1+3*i)
```

or

```
conj(1+3*i)
```

We get:

```
1-3*i
```

10.7.4 Imaginary part: IM im

IM or im returns the imaginary part of the complex number supplied as argument.

We enter:

```
IM(1+3*i)
```

We get:

```
3
```

10.7.5 Real part: RE re

RE or re returns the real part of the complex number supplied as argument.

We enter:

```
RE (1+3*i)
```

We get:

```
1
```

10.7.6 Sign: SIGN sign

`SIGN` or `sign` returns the complex number supplied as argument divided by its modulus.

We enter:

```
SIGN (1+3*i)
```

We get:

```
(1+3*i)/sqrt(10)
```

10.7.7 The key Shift +/- : ABS abs

The key `|x|` is a shifted key (`Shift +/-`).

The key `|x|` returns `ABS(x)`, which equals:

- the absolute value of a real,
- the modulus of a complex number,
- the length of a vector $v_j \left(\left(\sum_{j=1}^n |v_j|^2 \right)^{1/2} \right)$,
- the Schur norm or Frobenius norm of a matrix $a_{j,k} \left(\left(\sum_{j,k=1}^n |a_{j,k}|^2 \right)^{1/2} \right)$.

We enter:

```
ABS (1+3*i)
```

We get:

```
3.1622776602
```

We use the key `|x|` in the CAS:

```
ABS (1+3*i)
```

or we enter:

```
abs (1+3*i)
```

We get:

```
sqrt(10)
```

10.7.8 Write of complex in the form of $\text{re}(z) + i * \text{im}(z)$: evalc

`evalc` takes as argument a complex number z .

`evalc` returns this complex number, written in the form $\text{re}(z) + i * \text{im}(z)$.

We enter:

```
evalc(sqrt(2)*exp(i*pi/4))
```

We get:

$$1+i$$

10.7.9 Multiply by the complex conjugate: `mult_c_conjugate`

If an expression has a complex denominator, `mult_c_conjugate` multiplies the numerator and the denominator of this expression by the complex conjugate of the denominator.

If an expression does not have a complex denominator, `mult_c_conjugate` multiplies the numerator and the denominator of this expression by the complex conjugate of the numerator.

We enter:

```
mult_c_conjugate((2+i)/(2+3*i))
```

We get:

$$(2+i) * (2+3*(-i)) / ((2+3*(i)) * (2+3*(-i)))$$

We enter:

```
mult_c_conjugate((2+i)/2)
```

We get:

$$(2+i) * (2+-i) / (2 * (2+-i))$$

10.8 Exponential and Logarithms

10.8.1 Function neperian logarithm: `LN` `ln` `log`

`LN` or `ln` or `log` designates the function neperian logarithm.

`LN` (or `ln` in the CAS) is accessed with the key `LN`.

Warning! The neperian log is `LN` in HOME, and `ln` or `log` in the CAS.

We enter in HOME:

```
LN(e)
```

We get:

```
1
```

We enter:

```
LN(2)
```

We get:

```
0.69314718056
```

but in the CAS, we enter:

```
ln(e)
```

or

```
log(e)
```

We get:

$$1$$

We enter:

$$\ln(2)$$

or

$$\log(2)$$

We get:

$$\ln(2)$$

10.8.2 Function logarithm basis 10: LOG log10

Warning! The log basis 10 is LOG in HOME and log10 in the CAS and log designates the neperian log.

LOG or log10 designates the function logarithm basis ten, LOG (or log10 in the CAS) can be accessed with the key LOG.

We enter in HOME:

$$\text{LOG}(10)$$

We get:

$$1$$

We enter:

$$\text{LOG}(7)$$

We get:

$$0.84509804001$$

We enter in the CAS:

$$\log_{10}(10)$$

We get:

$$1$$

We enter:

$$\log_{10}(7)$$

We get:

$$\ln(7)/\ln(10)$$

10.8.3 Function logarithm basis b: logb

logb designates the function logarithm with the basis supplied as second argument:

We enter in the CAS:

$$\text{logb}(7, 7)$$

We get:

$$1$$

We enter in the CAS:

$$\text{logb}(7, 10)$$

We get:

$$\ln(7) / \ln(10)$$

$$\text{logb}(7, 10) = \text{log}_{10}(7) = \log(7) / \log(10)$$

10.8.4 Function antilogarithm: ALOG alog10

`alog10` designates the function antilogarithm basis ten, it is the function:
 $x \rightarrow 10^x$.

We enter in HOME:

$$\text{ALOG}(3/2)$$

We get:

$$31.6227766017$$

it is the approximate value of $\text{sqrt}(10) * 10$ at the nearest 10^{-10} .

We enter in the CAS:

$$\text{ALOG}(3/2)$$

or

$$\text{alog}(3/2)$$

or we get:

$$\text{sqrt}(10) * 10^1$$

We enter:

$$\text{alog10}(10)$$

We get:

$$10000000000$$

10.8.5 Function exponential: EXP exp

`EXP` or `exp` designates the function exponential.
`EXP` (or `exp` in the CAS) can be accessed with the key `EXP`.

We enter in HOME:

EXP (2)

We get:

7.38905609893

but in the CAS, we enter:

exp (2)

We get:

exp (2)

10.8.6 Function EXPM1

EXPM1 designates the function $x \rightarrow EXP(x) - 1$.

We enter:

EXPM1 (4)

We get:

EXP (4) -1

We enter:

EXPM1 (2.*10^-4)

We get:

0.00020002000133

BUT if we enter:

EXP (2.*10^-4) -1

We get:

0.00020002

10.8.7 Function LNP1

LNP1 designates the function $x \rightarrow LN(x + 1)$

We enter:

LNP1 (4)

We get:

LN (5)

We enter:

LNP1 (2.*10^-4)

We get:

1.99980002666E-4

Chapter 11 Trigonometric functions

11.1 The keys of trigonometric functions

- `SIN` or `sin` designates the function sine.

We enter in the CAS:

$$\text{SIN}(\pi/3)$$

or

$$\text{sin}(\pi/3)$$

We get:

$$\text{sqrt}(3)/2$$

- `ASIN` or `asin` designates the function arc sine.

We enter in the CAS:

$$\text{ASIN}(1/2)$$

or

$$\text{asin}(1/2)$$

We get:

$$\pi/6$$

- `COS` or `cos` designates the function cosine.

We enter in the CAS:

$$\text{COS}(\pi/3)$$

or

$$\text{cos}(\pi/3)$$

We get:

$$1/2$$

- `ACOS` or `acos` designates the function arc cosine.

We enter in the CAS:

$$\text{ACOS}(1/2)$$

or

$$\text{acos}(1/2)$$

We get:

$$\pi/3$$

- TAN or tan designates the function tangent.

We enter in the CAS:

$$\text{TAN}(\pi/3)$$

or

$$\text{tan}(\pi/3)$$

We get:

$$\text{sqrt}(3)$$

- ATAN or atan designates the function arc tangent.

We enter in the CAS:

$$\text{ATAN}(\text{sqrt}(3)/3)$$

or

$$\text{atan}(\text{sqrt}(3)/3)$$

We get:

$$\pi/6$$

11.2 Cosecant: CSC csc

CSC(x) or csc returns $1/\text{SIN}(x)$: it is the function cosecant.

We enter in the CAS:

$$\text{CSC}(\pi/3)$$

or

$$\text{csc}(\pi/3)$$

We get after simplification:

$$2*\text{sqrt}(3)/3$$

11.3 Arccosecant: ACSC acsc

ACSC(x) or acsc returns $\text{ASIN}(1/x)$: it is the reciprocal function of the function cosecant.

We enter in the CAS:

$$\text{ACSC}(\text{sqrt}(2))$$

or

$$\text{acsc}(\sqrt{2})$$

We get after simplification:

$$\pi/4$$

11.4 Secant: SEC sec

SEC(x) or sec returns $1/\cos(x)$: it is the function secant.

We enter in the CAS:

$$\text{SEC}(\pi/3)$$

or

$$\text{sec}(\pi/3)$$

We get:

$$2$$

11.5 Arcsecant: ASEC asec

ASEC(x) or asec returns $\arccos(1/x)$: it is the reciprocal function of the function secant.

We enter in the CAS:

$$\text{ASEC}(2)$$

or

$$\text{asec}(2)$$

We get:

$$1/3*\pi$$

11.6 Cotangent: COT cot

COT(x) or cot returns $\cos(x)/\sin(x)$: it is the function cotangent.

We enter in the CAS:

$$\text{COT}(\pi/3)$$

or

$$\text{cot}(\pi/3)$$

We get after simplification:

$$\sqrt{3}/3$$

11.7 Arccotangent: ACOT acot

$\text{ACOT}(x)$ or acot returns $\pi/2 - \text{ATAN}(x)$: it is the reciprocal function of the function cotangent.

We enter in the CAS:

```
ACOT(sqrt(3))
```

or

```
acot(sqrt(3))
```

We get after simplification:

```
pi/6
```

Chapter 12 Hyperbolic functions

12.1 Hyperbolic sine: SINH sinh

`SINH(x)` or `sinh(x)` returns:

$$\frac{\exp(x) - \exp(-x)}{2}$$

it is the function hyperbolic sine

We enter in the CAS:

```
hyp2exp(SINH(ln(2)))
```

We enter in the CAS:

```
hyp2exp(sinh(ln(2)))
```

We get:

$$3/4$$

Indeed, the command `hyp2exp` transforms the functions hyperbolic into exponentials.

12.2 Hyperbolic arc sine: ASINH asinh

`ASINH` or `asinh` is the reciprocal function of the function hyperbolic sine.

We enter in the CAS:

```
ASINH(3/4)
```

or

```
asinh(3/4)
```

We get:

$$\ln(2)$$

12.3 Hyperbolic cosine: COSH cosh

`COSH(x)` or `cosh` returns:

$$\frac{\exp(x) + \exp(-x)}{2}$$

it is the function hyperbolic cosine.

We enter in the CAS:

```
COSH(0)
```

or

$$\cosh(0)$$

We get:

$$1$$

12.4 Hyperbolic arc cosine: ACOSH acosh

ACOSH or acosh is the reciprocal function of the function hyperbolic cosine.

We enter in the CAS:

$$\text{hyp2exp}(\text{ACOSH}(1))$$

or

$$\text{hyp2exp}(\text{acosh}(1))$$

We get:

$$0$$

12.5 Hyperbolic tangent: TANH tanh

TANH(x) or tanh returns:

$$\frac{\exp(2x) - 1}{\exp(2x) + 1}$$

it is the function hyperbolic tangent.

We enter in the CAS:

$$\text{hyp2exp}(\text{TANH}(\ln(3)))$$

or

$$\text{hyp2exp}(\text{tanh}(\ln(3)))$$

We get:

$$4/5$$

Indeed, the command `hyp2exp` transforms the functions hyperbolic into exponentials.

12.6 Hyperbolic arc tangent: ATANH atanh

ATANH or atanh is the reciprocal function of the function hyperbolic tangent.

We enter in the CAS:

$$\text{ATANH}(4/5)$$

or

$$\text{atanh}(4/5)$$

We get:

$$1/2 * \ln(9)$$

12.7 Other functions

12.7.1 List of variables: `lname`

`lname` takes as parameter an expression.

`lname` returns a vector whose components are the name of symbolic variables used in this expression.

We enter:

```
lname(x*y*sin(x))
```

We get:

```
[x, y]
```

We enter:

```
a:=2; assume(b>0); assume(c=3);
lname(a*x^2+b*x+c)
```

We get:

```
[x, b, c]
```

12.7.2 List of variables and expressions: `lvar`

`lvar` takes as parameter an expression.

`lvar` returns a vector whose components are the names of variables and expressions which this expression depends logically.

We enter:

```
lvar(x*y*sin(x)^2+ln(x)*cos(y))
```

We get:

```
[x, y, sin(x)]
```

We enter:

```
lvar(x*y*sin(x)^2)
```

We get:

```
[x, y, sin(x), ln(x), cos(y)]
```

We enter:

```
lvar(y+x*sqrt(z)+y*sin(x))
```

We get:

```
[y, x, sqrt(z), sin(x)]
```

12.7.3 List of variables and algebraic expressions: `algvar`

`algvar` takes as parameter an expression.

`algvar` returns a vector whose components are the name of symbolic variables, by order of algebraic extension, used in this expression.

We enter:

```
algvar(y+x*sqrt(z))
```

We get:

```
[[y, x], [z]]
```

We enter:

```
algvar(y*sqrt(x)*sqrt(z))
```

We get:

```
[[y], [z], [x]]
```

We enter:

```
algvar(y*sqrt(x*z))
```

We get:

```
[[y], [x, z]]
```

We enter:

```
algvar(y+x*sqrt(z)+y*sin(x))
```

We get:

```
[[x, y, sin(x)], [z]]
```

12.7.4 Testing the presence of a variable in an expression: `has`

`has` takes as parameter an expression and the name of a variable.

`has` returns 1, or 0, depending on the variable is present, or not present, in the expression.

We enter:

```
has(x*y*sin(x), y)
```

We get:

```
1
```

We enter:

```
has(x*y*sin(x), z)
```

We get:

```
0
```

12.7.5 Evaluate an expression: `eval`

`eval` is used to evaluate an expression.

`eval` takes one or two arguments: an expression and eventually the wished level of the evaluation.

You have to know that the CAS always evaluates the expressions without having to call the command `eval`: the level of evaluation is indicated in the cell `Recursive Evaluation` of the CAS configuration (`Shift CAS`) and is checked by default to 5.

The command `eval` is mostly useful when we want to evaluate a subexpression in the expression editor.

We enter:

```
a:=2
```

We get:

```
2
```

We enter:

```
eval(2+3*a)
```

or

```
2+3*a
```

We get:

```
8
```

We enter:

```
purge(r);purge(p);a:=1+i*r
r:=p+1;p:=-4;
```

We can then get different evaluation of `a` according to the level of evaluation asked:

– we enter:

```
a
```

We get:

```
1-3*i
```

– we enter:

```
eval(a,1)
```

We get:

```
i*r+1
```

– we enter:

```
eval(a,2)
```

We get:

```
i*(p+1)+1
```

– we enter:

```
eval(a,3)
```

We get:

```
1-3*i
```

12.7.6 Not evaluating an expression: QUOTE quote '

If we do not want an expression to be evaluated in a calculation, we need to quote it, either with `'`, either thanks to the function `quote`.

Note

When we enter for example `a:=quote(a)`, this purges the variable `a`, and this instruction returns the value of this variable (or the assumptions made on this variable).

So `a:=quote(a)` is synonymous of `purge(a)`.

We enter:

```
a:=2;quote(2+3*a)
```

or

```
a:=2;'2+3*a'
```

We get:

```
(2,2+3.a)
```

12.7.7 Numerical evaluation: evalf approx

`evalf` or `approx` takes as parameter a numerical expression or a matrix.

`evalf` returns the numerical value of the expression or of the matrix.

By adding a second argument `n` to `evalf` (or `approx`), we can specify the number of significant digits of the approximation.

We enter:

```
evalf(sqrt(2))
```

We get:

```
1.41421356237
```

We enter:

```
evalf(sqrt(2),5)
```

We get:

```
1.4142
```

We enter:

```
evalf([[1,sqrt(2)],[0,1]])
```

We get:

```
[[1.0,1.41421356237],[0.0,1.0]]
```

We enter:

```
evalf([[1, sqrt(2)], [0, 1]], 5)
```

We get:

```
[[1.0, 1.4142], [0.0, 1.0]]
```

12.7.8 Rational approximation: `exact`

`exact` takes as parameter a real numerical expression.

`exact` gives a rational approximation of all the decimal numbers r present which validate $|r - \text{exact}(r)| < \varepsilon$, where ε is defined by `epsilon` in the CAS configuration (key Shift CAS).

We enter:

```
exact(1.5)
```

We get:

```
3/2
```

We enter:

```
exact(1.414)
```

We get:

```
707/500
```

We enter:

```
exact(1.41421356237^2)
```

We get:

```
2
```

Chapter 13 Probability functions

13.1 Factorial: `factorial` `!`

`!` is a postfix function whereas `factorial` is a prefix function.

`n!` or `factorial(n)` returns the factorial of n if n is n integer, and `a!` returns the value of the Gamma function for $a + 1$ if a is real.

We enter in the CAS:

```
20!
```

or

```
factorial(20)
```

We get:

```
2432902008176640000
```

We enter:

```
5.2!
```

or

```
factorial(5.2)
```

We get:

```
169.406099462
```

13.2 Number of combinations of p objects among n : `COMB` `comb`

`COMB(n, p)` or `comb(n, p)` returns the number of combinations of p elements among n (n and p are integers).

We have: `COMB(n, p)` returns

$$\frac{n!}{p!(n-p)!}$$

We enter:

```
COMB(5, 2)
```

We get:

```
10
```

13.3 Number of permutations of p objects among n : `PERM` `perm`

`PERM(n, p)` or `perm(n, p)` returns the number of permutations of p elements among n (n and p are integers).

We have: `PERM(n, p)` returns

$$\frac{n!}{(n - p)!}$$

We enter:

```
PERM(5, 2)
```

We get:

20

13.4 Random numbers

13.4.1 Random number (real or integer): `RANDOM`

- To get a random real number between 0 and 1, we do not put any argument.

We enter:

```
RANDOM
```

We get a real number of the range 0,1, for example:

0.291424166081

- To get a random integer a between 1 and n ($1 \leq a \leq n$), we put n as argument without brackets.

We enter:

```
RANDOM 3
```

We get 1, 2 or 3, for example:

1

- To get a random real number a between b and c ($b \leq a \leq c$), we put b, c as arguments without brackets.

We enter:

```
RANDOM 3, 5
```

We get a real number of 3,5 for example:

4.81731227506

- To get k random integers a between p and n ($p \leq a \leq n$) we put k, p, n as arguments without brackets.

We enter:

```
RANDOM 3, 2, 5
```

We get 3 integers between 2 and 5, for example:

[5, 3, 2]

13.4.2 Random integer: `RANDINT`

- To get a random integer a between 1 and n ($1 \leq a \leq n$) we put n as argument.

We enter:

```
RANDINT(4)
```

We get 0,1,2,3 or 4, for example:

```
2
```

- To get a random integer a between b and c ($b \leq a \leq c$), we put b and c as arguments.

We enter:

```
RANDINT(4, 6)
```

We get 4,5 or 6, for example:

```
5
```

- To get k random integers a between p and n ($p \leq a \leq n$) we put k, p and n as arguments.

We enter:

```
RANDINT(4, 2, 6)
```

We get 4 numbers between 2 and 6, for example:

```
[2, 6, 2, 6]
```

13.4.3 Rand function of the CAS: `rand`

Equally distributed draw on $[0, 1[$: `rand()`

`rand()` returns randomly, in an equiprobable manner, a real number of $[0, 1[$.

We enter:

```
rand()
```

We get for example:

```
0.912569261115
```

To get, randomly, in an equiprobable manner, a number of $[0; 1[$, we can also use (see following paragraph):

```
rand(0, 1)
```

We get:

```
0.391549611697
```

Equally distributed draw on the interval $[a; b[$: `rand(a, b)`

If a and b are reals, `rand(a, b)` designates a random decimal number of the interval $[a; b[$.

Thus, `rand(a, b)` returns randomly, and in an equiprobable manner, a decimal number of $[a; b[$.

To get, randomly and in an equiprobable manner, a decimal number of $[0; 1[$, we enter:

```
rand(0, 1)
```

We get:

```
0.391549611697
```

We get for example:

```
0.912569261115
```

To get, randomly and in an equiprobable manner, a decimal number of $[0; 0.5[$, we enter:

```
rand(0,0.5)
```

We get:

```
0.303484437987
```

To get, randomly and in an equiprobable manner, a decimal number of $] - 0.5; 0]$, we enter:

```
rand(0,-0.5)
```

or we enter:

```
rand(-0.5,0)
```

We get for example:

```
-0.20047219703
```

If a and b are reals, `rand(a..b)` designates a function which is a generator of random numbers of the interval $[a; b[$.

Thus, `rand(a..b)()` returns randomly, and in an equiprobable manner, a decimal number of $[a; b[$.

To get, randomly and in an equiprobable manner, a decimal number of $[0; 1[$, we enter:

```
rand(0..1)()
```

We get:

```
0.391549611697
```

To get, randomly and in an equiprobable manner, several random decimal numbers of the interval $[1; 2[$, we enter:

```
r:=rand(1..2)
```

then, it is enough to press `r()`.

We enter:

```
r()
```

We get:

```
1.14160255529
```

Random draw of equally distributed integers on $0, \dots, n$: `rand(n)`

If n is a relative integer, `rand(n)` returns randomly, and in an equiprobable manner, an integer of $[0, 1, \dots, n[$ (or of $]n, \dots, 0]$ if n is negative).

We enter:

```
rand(2)
```

We get:

1

or we get:

0

We enter:

`rand(-2)`

We get:

-1

or we get:

0

To get a random integer between 6 and 10, boundaries included, we enter:

`6+rand(11-6)`

We get for example:

8

Random draw without replacement of p objects among n : `rand`

`rand` has, in this case, either two, either three arguments.

If `rand` has two arguments: the arguments are an integer p and a list L , then `rand(p, L)` returns, randomly, p elements of the list L .

If `rand` has three arguments: the arguments are three integers p, \min, \max , then `rand(p, min, max)` returns, randomly, p integers of $[\min, \dots, \max]$

We enter:

`rand(3, ["r", "r", "r", "r", "v", "v", "v"])`

We get:

`["r", "r", "v"]`

We enter:

`rand(2, 1, 10)`

We get:

`[3, 7]`

We enter:

`rand(2, 4, 10)`

We get:

`[5, 7]`

13.4.4 Random permutation: `randperm`

`randperm` takes as argument an integer n .
`randperm` returns a random permutation of $[0..n - 1]$.

We enter:

```
randperm(3)
```

We get:

```
[2, 0, 1]
```

13.4.5 Generating a random list: `randvector`

`randvector` generates a list of random numbers.
`randvector` takes as argument an integer n and eventually a second argument, either an integer k , either the quoted or not quoted name of the distribution law of the random numbers of the list (see also 13.4.8, 13.5.4, 13.5.5 and 13.5.6).

`randvector` returns a list of degree n constituted of random integers equally distributed between -99 and 99 (by default) or between 0 and $k - 1$ or a list of degree n of random numbers distributed according to the quoted law or as parameter.

When `randvector` takes as argument an integer n and a random law of the calculator you have to quote or not in this case, `randvector` returns a list of dimension n whose elements are taken randomly according to the function supplied as third argument.

The functions supplied as second argument, which must be quoted or not, can be:

```
'rand(n)'  

'binomial(n,p)' or binomial,n,p or 'randbinomial(n,p)'  

'poisson( $\lambda$ )' or poisson, $\lambda$  or 'randpoisson( $\lambda$ )'  

'normald( $\mu,\sigma$ )' or normald, $\mu,\sigma$  or 'randnorm( $\mu,\sigma$ )'  

'exponential(a)' or exponential,a or 'randexp(a)'  

'fisher(n,m)' or fisher,n,m or 'randfisher(n,m)'
```

Warning! The syntax without quote suits to the laws, but not to the corresponding command `rand...`, then, for example, the commands `randvector(3,normald,0,1)` or `randvector(3,'normald(0,1)')` or `randvector(3,'randnorm(0,1)')` are valid but `randvector(3,randnorm,0,1)` is not valid.

We enter:

```
randvector(3)
```

We get for example:

```
[-54, 78, -29]
```

We enter:

```
randvector(3,5)
```

We enter:

```
randvector(3,'rand(5)')
```

We get for example:

```
[1, 2, 4]
```

We enter:

```
randvector(3,normald,0,1)
```

Or we enter:

```
randvector(3,'normald(0,1)')
```

We get for example:

```
[1.39091705476,-0.136794772167,0.187312440336]
```

We enter:

```
randvector(3,2..4)
```

We get for example:

```
[3.92450003885,3.50059241243,2.7322040787]
```

We enter:

```
randvector(6,binomial,4,0.2)
```

Or we enter:

```
randvector(6,'binomial(4,0.2)')
```

We get for example:

```
[0,1,0,2,2,0]
```

We enter:

```
randvector(6,poisson,1.3)
```

Or we enter:

```
randvector(6,'poisson(1.3)')
```

We get for example:

```
[1,0,1,1,1,1]
```

We enter:

```
randvector(4,exponential,1.2)
```

Or we enter:

```
randvector(4,'exponential(1.2)')
```

We get for example:

```
[1.67683756526,0.192937941271,0.580820253805,0.709352619633]
```

We enter:

```
randvector(5,fisher,4,6)
```

Or we enter:

```
randvector(5,'fisher(4,6)')
```

We get for example:

```
[0.17289703163,1.03709368317,0.161051043162,1.4407877128,0.3586901042
75]
```

13.4.6 Draw according to a multinomial law with programs

We write the programs `randmult` and `randmultiname` which simulate the multinomial law. `randmult(n,P)` chooses randomly n numbers among $1..k$ according to the multinomial law of probability P ($k=\text{size}(P)$). This means that we do a draw with replacement of n objects among $k=\text{size}(P)$ objects. `randmult(n,P)` returns a list R of $k=\text{size}(P)$ elements where $R[j]$ is the number of objects of probability $P[j]$ which have been drawn.

We enter the program `randmult`:

```
(n,p)->BEGIN
  local k,j,l,r,x,y;
  k:=size(p);
  x:=cumSum(p);
  if x[k]!=1 then return "error"; end;
  y:=makelist(0,1,k)
  for j from 1 to n do
    r:=rand(0,1);
    l:=1;
    while r>x[l] do
      l:=l+1;
    end;
    y[l]:=y[l]+1
  end;
  return (y);
END;
```

We do six times the draw of an object among three objects (draw with replacement). Each object has a probability $[1/2,1/3,1/6]$ to be drawn.

To simulate a draw, we enter:

```
randmult(6,[1/2,1/3,1/6])
```

We get for example:

```
[3,2,1]
```

`randmultinom(n,P,C)` chooses randomly n objects among the elements of the list C . If $k=\text{size}(C)$, the object $C[j]$ has the probability $P[j]$ to be drawn for ($j=1..k$). We must get $k=\text{size}(C)=\text{size}(P)$ and $\text{sum}(P)=1$.

`randmultinom(n,P,C)` returns the sequence of k lists constituted of the name of the objects and their number of occurrence.

We enter the program `randmultinom`:

```
(n,p,c)->BEGIN
  local k,j,l,r,x,y;
  k:=size(p);
  if size(c)!=k then return "error"; end;
  x:=cumSum(p);
  if x[k]!=1 then return "error"; end;
  y:=MAKELIST([c[j],0],j,1,k);
  for j from 1 to n do
    r:=rand(0,1);
    l:=1;
    while r>x[l] do
      l:=l+1;
    end;
    y[l,2]:=y[l,2]+1
  end;
```

```
return y;
END;
```

We do six times the draw of an object among three objects ["A", "B", "C"] (draw with replacement). Each object has the probability [1/2,1/3,1/6] to be drawn. To simulate a draw, we enter:

```
randmultinom(6, [1/2,1/3,1/6], ["A", "B", "C"])
```

We get for example:

```
[["A", 3], ["B", 1], ["C", 2]]]
```

13.4.7 Draw according to a normal distribution: RANDNORM randNorm

RANDNORM(mu, sigma) or randNorm(mu, sigma) returns a real randomly distributed according to the normal distribution $N(\mu, \sigma)$ (by default $\mu = 0$ and $\sigma = 1$).

We enter:

```
RANDNORM()
```

Or we enter:

```
RANDNORM(0, 1)
```

We get, for example:

```
1.2440525851
```

We enter:

```
RANDNORM(1, 2)
```

We get, for example:

```
-1.8799815939
```

13.4.8 Draw according to an exponential law: randexp

randexp(a) returns numbers randomly distributed according to the exponential law of positive parameter a.

The density of probability is proportional to $\exp(-a * t)$ and we have:

$$Proba(X \leq t) = a \int_0^t \exp(-a * u) du.$$

We enter:

```
randexp(1)
```

We get for example:

```
0.310153677284
```

or we get for example:

```
0.776007926195
```

13.4.9 Initializing the series of random numbers: RANDSEED RandSeed srand

RANDSEED or RandSeed or srand initializes the series of random numbers supplied by RANDOM. If we do not put a parameter, RANDSEED uses the time value as parameter.

We enter:

```
RANDSEED ()
```

We get:

```
1
```

We enter:

```
RANDSEED (pi)
```

We get:

```
1
```

13.4.10 Function UTPC

UTPC ($n, x0$) returns the probability that a random Chisquare variable with n degrees of freedom be greater than $x0$.

We enter:

```
UTPC (2, 6.1)
```

We get:

```
0.0473589243911
```

13.4.11 Function UTPF

UTPF ($n, m, x0$) returns the probability that a random Fisher-Snedecor variable with n, m degrees of freedom be greater than $x0$.

We enter:

```
UTPF (4, 10, 3.5)
```

We get:

```
0.0491881403249
```

13.4.12 Function UTPN

UTPN ($\mu, v, x0$) returns the probability that a random Normal variable be greater than $x0$ with μ the mean and v the variance (by default $\mu = 0$ and $v = 1$).

We enter:

```
UTPN (1.96) or UTPN (1, 4, 4.92)
```

We get:

```
0.0249978951482
```

indeed $(x - 1)/\sqrt{4} > 1.96$ is equivalent to $x > 4.92$.

We enter:

```
UTPN(0.98)
```

or

```
UTPN(1, 4, 2.96)
```

We get:

```
0.163543059328
```

indeed $(x - 1)/\sqrt{4} > 0.98$ is equivalent to $x > 2.96$.

13.4.13 Function UTPT

`UTPT(n, x0)` returns the probability that a random Student variable with n degrees of freedom be greater than $x0$.

We enter:

```
UTPT(3, 2.35)
```

We get:

```
0.050152899407
```

We enter:

```
UTPT(3, -2.35)
```

We get:

```
0.949847100593
```

13.5 Density of probability

13.5.1 Density of probability of the normal distribution: `NORMALD` `normald`

`NORMALD(x)` or `normald(x)` is the density of probability of the normal reduced centered distribution (of mean 0 and standard deviation 1).

`NORMALD(μ, σ, x)` or `normald(μ, σ, x)` is the density of probability of the normal distribution of mean μ and standard deviation σ .

We enter:

```
NORMALD(0.5)
```

Or we enter:

```
NORMALD(0, 1, 0.5)
```

We get:

```
0.352065326764
```

We enter:

NORMALD(1, 2, 0.5)

We get:

0.193334058401

13.5.2 Density of probability of the Student law: STUDENT student

STUDENT(n, x) or student(n, x) is the density of probability of the Student law having n degrees of freedom.

We enter:

STUDENT(3, 5.2)

We get:

0.00366574413491

13.5.3 Density of probability of the χ^2 : CHISQUARE chisquare

CHISQUARE(n, x) or chisquare(n, x) is the density of probability of the χ^2 law having n degrees of freedom.

We enter:

CHISQUARE(2, 3.2)

We get:

0.100948258997

13.5.4 Density of probability of the Fisher law: FISHER fisher snedecor

FISHER(n, m, x) or fisher(n, m, x) or snedecor(n, m, x) returns the density of probability in x of the Fisher-Snedecor law (n and m are the numbers of degrees of freedom).

We enter:

FISHER(4, 10, 2.1)

We get:

0.141167840452

13.5.5 Density of probability of the binomial law: BINOMIAL binomial

BINOMIAL(n, k, p) or binomial(n, k, p) returns $\text{COMB}(n, k) * p^k * (1 - p)^{(n - k)}$ and BINOMIAL(n, k) or binomial(n, k, p) returns $\text{COMB}(n, k)$ if there is no third argument.

We enter:

BINOMIAL(4, 2)

We get:

We enter:

```
BINOMIAL(4,2,0.5)
```

We get:

```
0.375
```

We enter in the CAS:

```
binomial(4,2)
```

We get:

```
6
```

We enter in the CAS:

```
binomial(4,2,1/2)
```

We get:

```
3/8
```

13.5.6 Density of probability of the Poisson law: POISSON poisson

The density of probability of the Poisson law of parameter μ , that is to say of mean μ and standard deviation μ is:

POISSON(μ , k) or poisson(μ , k) returns

$$\frac{\exp(-\mu) * \mu^k}{k!}$$

We enter:

```
POISSON(0.5,2)
```

We get:

```
0.0758163324641
```

13.6 Function of distribution

13.6.1 Function of distribution of the normal distribution: NORMALD_CDF normald_cdf

When a random variable X follows a normal reduced centered distribution, we have:

$Proba(X \leq x) = \text{NORMALD_CDF}(x) = \text{normald_cdf}(x)$ and

$Proba(x \leq X \leq y) = \text{NORMALD_CDF}(x, y) = \text{normald_cdf}(x, y)$.

When a random variable X follows a normal distribution of mean μ and standard deviation σ , we have:

$Proba(X \leq x) = \text{NORMALD_CDF}(\mu, \sigma, x)$.

$Proba(x \leq X \leq y) = \text{NORMALD_CDF}(\mu, \sigma, x, y)$.

We enter:

```
NORMALD_CDF(0.96)
```

Or we enter:

NORMALD_CDF(0,1,0.96)

We get:

0.831472392533

We enter:

NORMALD_CDF(1.96)

We get:

0.975002104852

We enter:

NORMALD_CDF(0,1.96)

We get:

0.475002104852

because $\text{NORMALD_CDF}(0) = 1/2$ and $0.975002104852 - 0.5 = 0.475002104852$

We enter:

NORMALD_CDF(1,2,1.96)

We get:

0.684386303484

We enter:

NORMALD_CDF(1,2,1.1,2.9)

We get:

0.309005067853

13.6.2 Function of distribution of the Student law: STUDENT_CDF student_cdf

When a random variable X follows a Student law having n degrees of freedom, we have:

$\text{Proba}(X \leq x) = \text{STUDENT_CDF}(n, x) = \text{student_cdf}(n, x)$.

$\text{Proba}(x \leq X \leq y) = \text{STUDENT_CDF}(n, x, y) = \text{student_cdf}(n, x, y)$.

We enter:

STUDENT_CDF(5,2)

We get:

0.949030260585

We enter:

STUDENT_CDF(5,-2)

We get:

0.0509697394149

13.6.3 Function of distribution of the χ^2 law: CHISQUARE_CDF chisquare_cdf

When a random variable X follows a χ^2 law having n degrees of freedom, we have:

$$\text{Proba}(X \leq x) = \text{CHISQUARE_CDF}(n, x) = \text{chisquare_cdf}(n, x).$$

$$\text{Proba}(x \leq X \leq y) = \text{CHISQUARE_CDF}(n, x, y) = \text{chisquare_cdf}(n, x, y).$$

We enter:

CHISQUARE_CDF(5, 11)

We get:

0.948620016517

We enter:

CHISQUARE_CDF(5, 3)

We get:

0.300014164121

We enter:

CHISQUARE_CDF(5, 3, 11)

We get:

0.648605852396

because $0.948620016517 - 0.300014164121 = 0.648605852396$

13.6.4 The function of distribution of the Fisher-Snedecor law: FISHER_CDF fisher_cdf snedecor_cdf

When a random variable X follows a Fisher-Snedecor law having as degrees of freedom n_1, n_2 , we have:

$$\text{Proba}(X \leq x) = \text{FISHER_CDF}(n_1, n_2, x) = \text{fisher_cdf}(n_1, n_2, x).$$

$$\text{Proba}(x \leq X \leq y) = \text{FISHER_CDF}(n_1, n_2, x, y) = \text{fisher_cdf}(n_1, n_2, x, y) = \text{snedecor_cdf}(n_1, n_2, x, y).$$

We enter:

FISHER_CDF(5, 3, 9)

We get:

0.949898927032

We enter:

FISHER_CDF(3, 5, 9.)

We get:

0.981472898262

We enter:

FISHER_CDF(3,5,2.)

We get:

0.767376082

We enter:

FISHER_CDF(3,5,2.,9.)

We get:

0.214096816262

because $0.981472898262 - 0.767376082 = 0.214096816262$

13.6.5 Function of distribution of the binomial law: BINOMIAL_CDF binomial_cdf

When a random variable X follows a binomial law $B(n, p)$.

We have:

$BINOMIAL_CDF(n, p, x) = \text{binomial_cdf}(n, p, x) = \text{Proba}(X \leq x) =$
 $BINOMIAL(n, 0, p) + \dots + BINOMIAL(n, \text{floor}(x), p).$

$BINOMIAL_CDF(n, p, x, y) = \text{binomial_cdf}(n, p, x, y) = \text{Proba}(x \leq X \leq y) =$
 $BINOMIAL(n, \text{ceil}(x), p) + \dots + BINOMIAL(n, \text{floor}(y), p).$

We enter:

BINOMIAL_CDF(4,0.5,2)

We get:

0.6875

We can check that:

$BINOMIAL(4, 0, 0.5) + BINOMIAL(4, 1, 0.5) + BINOMIAL(4, 2, 0.5)$
 $= 0.6875$

We enter:

BINOMIAL_CDF(2,0.3,1)

We get:

0.91

We enter:

BINOMIAL_CDF(2,0.3,1,2)

We get:

0.51

We enter in the CAS:

```
binomial_cdf(4,1/2,2)
```

We get:

```
11/16
```

We enter:

```
binomial_cdf(2,3/10,1)
```

We get:

```
91/100
```

We enter:

```
binomial_cdf(2,3/10,1,2)
```

We get:

```
51/100
```

13.6.6 Function of distribution of the Poisson law: POISSON_CDF poisson_cdf

When a random variable X follows a Poisson law of parameter μ , of mean μ , we have:

$Proba(X \leq x) = POISSON_CDF(\mu, x) = poisson_cdf(\mu, x)$ with $X \in P(\mu)$. And

$Proba(x \leq X \leq y) = POISSON_CDF(\mu, x, y) = poisson_cdf(\mu, x, y)$

$POISSON_CDF(\mu, x)$ is the function of distribution of the Poisson law of parameter μ .

We enter:

```
POISSON_CDF(10.0,3)
```

We get:

```
0.0103360506759
```

13.7 Inverse distribution function

13.7.1 Inverse normal distribution function: NORMALD_ICDF normald_icdf

When a random variable X follows a normal reduced centered distribution, if we have $NORMALD_ICDF(x) = normald_icdf(x) = h$, it is what we have:

$Proba(X \leq h) = x = NORMALD_CDF(h) = normald_cdf(h)$.

When a random variable X follows a normal distribution of mean μ and standard deviation σ , if we have:

$NORMALD_ICDF(\mu, \sigma, x) = normald_icdf(\mu, \sigma, x) = h$

it is that we have:

$Proba(X \leq h) = x = NORMALD_CDF(\mu, \sigma, h) = normald_cdf(\mu, \sigma, h)$.

We enter:

```
NORMALD_ICDF(0.95)
```

Or we enter:

`NORMALD_ICDF(0,1,0.95)`

We get:

1.64485362695

We enter:

`NORMALD_ICDF(0.975)`

We get:

1.95996398454

We enter:

`NORMALD_ICDF(1,2,0.495)`

We get:

0.974933060984

We enter:

`NORMALD_ICDF(1,2,NORMALD_CDF(1,2,0.975))`

We get:

0.975

We enter:

`NORMALD_CDF(1,2,NORMALD_ICDF(1,2,0.495))`

We get:

0.495

We enter:

`NORMALD_ICDF(1,2,2.96*sqrt(2))`

We get:

0.944423950497

13.7.2 Inverse distribution Student's function: `STUDENT_ICDF` `student_icdf`

When a random variable X follows a Student law having n degrees of freedom, if we have $STUDENT_ICDF(n, x) = student_icdf(n, x) = h$ it is that:

$Proba(X \leq h) = x = STUDENT_CDF(n, h) = student_cdf(n, h)$.

We enter:

`STUDENT_ICDF(5,0.95)`

We get:

2.01504837333

13.7.3 Inverse function of the function of distribution of the χ^2 law:

CHISQUARE_ICDF chisquare_icdf

When a random variable X follows a χ^2 law having n degrees of freedom, if we have
 $\text{CHISQUARE_ICD}(n, x) = \text{chisquare_icdf}(n, x) = h$ it is that:

$\text{Proba}(X \leq h) = x = \text{CHISQUARE_CDF}(n, h) = \text{chisquare_cdf}(n, h)$.

We enter:

CHISQUARE_ICDF(5, 0.95)

We get:

11.0704976935

13.7.4 Inverse of the function of distribution of the Fisher-Snedecor law:

FISHER_ICDF fisher_icdf snedecor_icdf

When a random variable X follows a Fisher-Snedecor law having as degrees of freedom n_1, n_2 , if we have:

$\text{FISHER_ICDF}(n_1, n_2, x) = \text{fisher_icdf}(n_1, n_2, x) = \text{snedecor_icdf}(n_1, n_2, x) = h$

it is that:

$\text{Proba}(X \leq h) = x = \text{FISHER_CDF}(n_1, n_2, h) = \text{fisher_cdf}(n_1, n_2, h) = \text{snedecor_cdf}(n_1, n_2, h)$

We enter:

FISHER_ICDF(5, 3, 0.95)

We get:

9.01345516752

We enter:

1/FISHER_ICDF(3, 5, 0.05)

We get:

9.01345516752

Note:

$\text{FISHER_ICDF}(n_1, n_2, p) = 1/\text{FISHER_ICDF}(n_2, n_1, 1-p)$

13.7.5 Inverse distribution function of the binomial law: BINOMIAL_ICDF

binomial_icdf

When a random variable X follows a binomial law $B(n, p)$, if we have:

$\text{BINOMIAL_ICDF}(n, p, x) = \text{binomial_icdf}(n, p, x) = h$ it is that

$\text{Proba}(X \leq h) = x = \text{BINOMIAL_CDF}(n, p, h) = \text{binomial_cdf}(n, p, h)$.

We enter:

BINOMIAL_ICDF(4, 0.5, 0.9)

We get:

3

We enter:

```
BINOMIAL_ICDF(2,0.3,0.95)
```

We get:

2

We enter in the CAS:

```
binomial_icdf(4,1/2,0.9)
```

We get:

3

We enter:

```
binomial_icdf(2,3/10,0.95)
```

We get:

2

13.7.6 Inverse distribution function of Poisson: POISSON_ICDF poisson_icdf

When a random variable X follows a Poisson law of parameter μ , of mean μ , we have:

$POISSON_ICDF(\mu, t) = \text{poisson_icdf}(\mu, t) = h$ is equivalent to

$Proba(X \leq h) = t = \text{poisson_cdf}(\mu, h) = POISSON_CDF(\mu, h)$ with $X \in P(\mu)$.

$POISSON_ICDF(\mu, t)$ is the inverse of the function of distribution of the Poisson law of parameter μ .

We enter:

```
POISSON_ICDF(10.0,0.975)
```

We get:

0.125110035721

Chapter 14 Statistics functions

14.1 Statistics functions at one variable

We will describe the different statistics functions thanks to an example:

with the list $A := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$

- taking as statistical sequence of size 1 the list A_1 , or
- taking as statistical sequence the list A_1 with as size again the list A_1 .

We enter:

```
A1 := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

We can also refer to 15.1 when the arguments are lists and at 15.1.1 when the arguments are matrices.

14.1.1 The mean: `mean`

`mean` returns the numerical mean of the elements of a list (or of each column of a matrix).

We enter:

```
A := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
mean(A)
```

We get:

```
11/2
```

Indeed, $(0 + 1 + \dots + 11) = 66$ and $66/12 = 11/2$

We enter:

```
mean([[1, 2], [3, 4]])
```

We get:

```
[2, 3]
```

Indeed, $(1 + 3)/2 = 2$ and $(2 + 4)/2 = 3$.

`mean` returns the numerical mean of the elements of a list (respectively of each column of a matrix) weighted by a list (respectively a matrix) of same size, supplied as second argument.

We enter:

```
A := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
mean(A, A)
```

We get:

```
23/3
```

Indeed: $1 * 1 + 2 * 2 + \dots + 11 * 11 = 23 * 12 * 11 / 6 = 23 * 2 * 11$ and $1 + 2 + \dots + 11 = 66$ then:
 $\text{mean}(A, A) = 23 * 2 * 11 / 66 = 23/3$

We enter:

```
mean([[1, 2], [3, 4]], [[1, 2], [3, 4]])
```

We get:

```
[5/2, 10/3]
```

Indeed: $(1 * 1 + 3 * 3) / (1 + 3) = 5/2$ and $(2 * 2 + 4 * 4) / (2 + 4) = 10/3$

14.1.2 The standard deviation: `stddev`

`stddev` returns the numerical standard deviation of the elements of a list (or of each column of a matrix).

We enter:

```
A:= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
stddev(A)
```

We get:

```
sqrt(143/12)
```

We enter:

```
stddev([[1, 2], [3, 4]])
```

We get:

```
[1, 1]
```

`stddev` returns the numerical standard deviation of the elements of a list weighted by another list supplied as second argument.

We enter:

```
A:= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
stddev(A, A)
```

We get:

```
sqrt(65/9)
```

14.1.3 The standard deviation of the population: `stddevp` `stdDev`

`stddevp` takes as argument a (or two) list(s):

`stddevp(l)` returns an estimation of the numerical standard deviation of the population whose is the sample described by the elements of the list `l`, of length `n`, supplied as argument (`size(l)=n` and `n` must be large). We have:
 $\text{stddevp}(l)^2 = n / (n-1) * \text{stddev}(l)^2$.

We enter:

```
A1:= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
stddevp(A1)
```

We get:

```
sqrt(13)
```

Indeed: $n=\text{size}(A1)=12$ and $12/11*\text{stddev}(A1)^2=12/11*143/12=13$.

We enter:

```
stddevp([[1,2],[3,4]])
```

We get:

```
[sqrt(2),sqrt(2)]
```

`stddevp(l1,l2)` returns the numerical standard deviation of the population whose is issued by the sample described by the elements of a list `l1` weighted by another list `l2` supplied as second argument.

We have:

$\text{stddevp}(l1,l2)^2=n/(n-1)*\text{stddev}(l1,l2)^2$ if n is the size of the sample, that is to say if n is the sum of the list `l2` ($\text{sum}(l2)=n$).

We enter:

```
stddevp(A1,A1)
```

We get:

```
sqrt(22/3)
```

Indeed, $\text{sum}(A1)=66$ and $\frac{22}{3} = \frac{66}{65} * \frac{65}{9}$

Note `stddev` is the standard deviation after division by n (size of the sample) whereas `stddevp` is divided by $n-1$ and gives the non biased estimator of the standard deviation of a population from the standard deviation calculated with a sample (the division by $n-1$ allows to remove the bias).

For the variance, we just give one command (division by n), but it is very easy to define a "variance of sample" by taking the square of the standard deviation `stddevp`.

14.1.4 The variance: `variance`

`variance` returns the numerical variance of the elements of a list.

We enter:

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
variance(A1)
```

We get:

```
143/12
```

`variance` returns the numerical variance of the elements of a list weighted by another list supplied as second argument.

We enter:

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
variance(A1,A1)
```

We get:

65/9

We enter:

```
variance([[1,2],[3,4]])
```

We get:

[1,1]

14.1.5 The median: median

`median` returns the median of the elements of a list.

We enter:

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]
median(A1)
```

We get:

5.0

`median` returns the numerical median of the elements of a list weighted by another list supplied as second argument.

We enter:

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]
median(A1,A1)
```

We get:

8

We have indeed: $1 + 2 + 3 + \dots + 7 = 28$ and $9 + 10 + 11 = 30$ there are then 28 elements before 8 and 30 elements after 8.

14.1.6 Different statistics values: quartiles

`quartiles` returns the matrix column formed by: the minimum, the first quartile, the median, the third quartile and the maximum of the elements of a list.

We enter:

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]
quartiles(A1)
```

We get:

```
[[0.0],[2.0],[5.0],[8.0],[11.0]]
```

We enter:

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
quartiles(A1,A1)
```

We get:

```
[1, 6, 8, 10, 11]
```

14.1.7 The first quartile: `quartile1`

`quartile1` returns the first quartile of the elements of a list.

We enter:

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
quartile1(A1)
```

We get the first quartile of A1:

```
2.0
```

`quartile1` returns the first quartile of the elements of a list weighted by another list supplied as second argument.

We enter:

```
quartile1(A1,A1)
```

We get the first quartile of A1 weighted by A1:

```
6
```

14.1.8 The third quartile: `quartile3`

`quartile3` returns the third quartile of the elements of a list.

We enter:

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
quartile3(A1)
```

We get the third quartile of A1:

```
8.0
```

`quartile3` returns the third quartile of the elements of a list weighted by another list supplied as second argument.

We enter:

```
quartile3(A1,A1)
```

We get the first quartile of A1 weighted by A1:

```
10
```

14.1.9 The quantile: `quantile`

`quantile(L1,p)` where `L1` is the statistical sequence and `p` a real of $[0,1[$, tells the value of the character starting from which the cumulated frequency of `L1` reaches or exceeds `p`.

We enter:

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
quantile(A1,0.1)
```

We get the first quantile:

```
1.0
```

We enter:

```
quantile(A1,0.25)
```

We get the first quartile:

```
2.0
```

We enter:

```
quantile(A1,0.5)
```

We get the median:

```
5.0
```

We enter:

```
quantile(A1,0.75)
```

We get the third quartile:

```
8.0
```

We enter:

```
quantile(A1,0.9)
```

We get the ninth quantile:

```
10.0
```

`quantile(l1,l2,p)` returns the quantile specified by the last argument of the elements of the list `l1` weighted by the list `l2`.

We enter:

```
quantile(A1,A1,0.25)
```

We get the first quartile of the list `A` weighted by `A`:

```
6
```

14.1.10 The histogram: `histogram`

`histogram` plots the histogram of data. We can specify a list of numbers of items, or a number `nc` of classes, or the minimum `classmin` of the classes and the size `classsize` of the classes.

`histogram` allows to display the function density of frequencies: we put as abscissa the classes and as ordinate the density of frequency (if we have discret values, they are considered as being the center of the class). The histogram is then a stairs graph in which the frequency of different classes are represented by the areas of different rectangles located under the different steps.

We point out that, if the size of the class $[a_{j-1}; a_j]$ is n_j , the frequency of the classe $[a_{j-1}; a_j]$ is $f_j = \frac{n_j}{N}$ (if N is the total number of items) and the density of frequency of the class $[a_{j-1}; a_j]$ is $\frac{f_j}{a_j - a_{j-1}}$.

We enter:

```
histogram([[1.5..1.65,50],[1.65..1.7,20],[1.7..1.8,30]])
```

The graphic windows automatically opens and we get the histogram of the sequence $[[1.5..1.65,50],[1.65..1.7,20],[1.7..1.8,30]]$, provided that the plot configuration has been correctly defined (menu Cfg).

The argument of `histogram` can also be a list of discrete values. In this case, the classes start at a value (`class_min`) and are all of same size (`class_size`), either defined by default (at 0 and 1, values to be checked in the graphic settings), either put as second and third arguments.

We enter:

```
histogram([0,1,2,1,1,2,1,2,3,3])
```

so `class_min=0` and `class_size=1` and then the values 0,1,2,3 are not centered.
but if we enter:

```
histogram([0,1,2,1,1,2,1,2,3,3],-0.5,1)
```

so `class_min=-0.5` and `class_size=1` and the values 0,1,2,3 are then centered, and it returns the same thing as:

```
histogram([[0,1],[1,4],[2,3],[3,2]])
```

We enter:

```
histogram(seq(rand(1000),k,1,100),0,100)
```

Here we have chosen `class_min=0` and `class_size=100`.

We enter:

```
histogram(seq(rand(10),k,1,100),0,1)
```

Here we have chosen `class_min=0` and `class_size=1`.

14.1.11 The covariance: `covariance`

The covariance of random variables X and Y is:

$$\text{cov}(X,Y) = E((X - \bar{X})(Y - \bar{Y})).$$

`covariance` has different kinds of arguments:

- when the sizes equal 1, `covariance` takes as argument two lists of same length or a matrix of two columns.

`covariance` returns the numerical variance of two lists or two columns of this matrix.

We enter:

```
covariance([1,2,3,4],[1,4,9,16])
```

We get:

25/4

We enter:

```
covariance([[1,1],[2,4],[3,9],[4,16]])
```

We get:

25/4

Because we have:

$$1/4 * (1 + 8 + 27 + 64) - 75/4 = 25/4$$

Provided that `A1:= [0,1,2,3,4,5,6,7,8,9,10,11]`, we enter:

```
covariance(A1,A1^2)
```

We get:

1573/12

– when the sizes are different from 1:

- if the paired values $a[j], b[j]$ have as size $n[j]$ ($j = 0..p-1$), `covariance` takes as argument three lists a, b, n of same length p , or a matrix of three columns a, b, n and p lines $[a[j], b[j], n[j]]$.
`covariance` returns the numerical variance of the two first lists weighted by the list supplied as last argument, or of the two columns of this matrix weighted by the third column.

We enter:

```
covariance([1,2,3,4],[1,4,9,16],[3,1,5,2])
```

Or we enter:

```
covariance([[1,1,3],[2,4,1],[3,9,5],[4,16,2]])
```

We get:

662/121

- if the paired values $a[j], b[k]$ have for size $N[j,k]$ ($j = 1..p, k = 1..q$), `covariance` takes as argument two lists a, b of respective lengths p and q , and a matrix N of p rows and q columns, or also, in order to write the data in a pleasant way in the table, `covariance` can also have two arguments, a matrix M and -1 . M is then a double entry table equal to:

$$M = \begin{bmatrix} a \setminus b & b[1] & \cdots & b[q] \\ a[1] & N[1,1] & \cdots & N[1,q] \\ \vdots & \vdots & \ddots & \vdots \\ a[p] & N[p,0] & \cdots & N[p,q] \end{bmatrix}$$

`covariance(a,b,N)` or `covariance(M,-1)` returns the numerical covariance of paired values $a[j], b[k]$ weighted by $N_{j,k}$.

We enter:

```
covariance([1,2,3,4],[1,4,9,16],[[3,0,0,0],
[0,1,0,0],[0,0,5,0],[0,0,0,2]])
```

We get:

662/121

We enter:

```
covariance([[b\|a, 1, 2, 3, 4], [1, 3, 0, 0, 0],
[4, 0, 1, 0, 0], [9, 0, 0, 5, 0], [16, 0, 0, 0, 2]], -1)
```

We get:

662/121

14.1.12 The correlation: correlation

The coefficient of linear correlation of two random variables X and Y is $\rho = \frac{\text{cov}(X,Y)}{\sigma(X)\sigma(Y)}$ where $\sigma(X)$ (resp. $\sigma(Y)$) designates the standard deviation of X (resp. Y).

`correlation` has the same arguments as `covariance`.

When the sizes equal 1, `correlation` takes as argument two lists of same length or a matrix of two columns.

We enter:

```
correlation([1, 2, 3, 4], [1, 4, 9, 16])
```

We get:

100/(4*sqrt(645))

We enter:

```
correlation([[1, 1], [2, 4], [3, 9], [4, 16]])
```

We get:

100/(4*sqrt(645))

Provided that `A1 := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]`, we enter:

```
correlation(A1, A1^2)
```

We get:

18876/(572*sqrt(1173))

When the sizes are different from 1:

- if the paired values $a[j], b[j]$ have as size $n[j]$ ($j = 0..p - 1$), `correlation` takes as argument three lists a, b, n of same length p , or a matrix of three columns a, b, n and p rows $[a[j], b[j], n[j]]$.

`correlation` returns the numerical correlation of the two first lists which are weighted by the list supplied as last argument or returns the numerical correlation of two columns of this matrix which are weighted by the third column.

We enter:

```
correlation([1, 2, 3, 4], [1, 4, 9, 16], [3, 1, 5, 2])
```

Or we enter:

```
correlation([[1, 1, 3], [2, 4, 1], [3, 9, 5], [4, 16, 2]])
```

We get:

662/(180*sqrt(14))

- if the paired values $a[j], b[k]$ have for size $N[j, k]$ ($j = 1..p, k = 1..q$), `correlation` takes as argument two lists a, b of respective lengths p and q and a matrix N of p rows and q columns or else, in order to write the data in a pleasant way in the table, `correlation` can also get for argument, a matrix M and -1 .

M is then a double entry table equal to:

$$M = \begin{bmatrix} a \setminus b & b[1] & \cdots & b[q] \\ a[0] & N[1,1] & \cdots & N[1,q] \\ \vdots & \vdots & \ddots & \vdots \\ a[p] & N[p,1] & \cdots & N[p,q] \end{bmatrix}$$

`correlation(a, b, N)` or `correlation(M, -1)` returns the numerical correlation of paired values $a[j], b[k]$ weighted by $N_{j,k}$.

We enter:

```
correlation([1,2,3,4],[1,4,9,16],[[3,0,0,0],[0,1,0,0],
[0,0,5,0],[0,0,0,2]])
```

We get:

```
662/(180*sqrt(14))
```

We enter:

```
correlation(["b\ a",1,2,3,4],[1,3,0,0,0],
[4,0,1,0,0],[9,0,0,5,0],[16,0,0,0,2]),-1)
```

We get:

```
662/(180*sqrt(14))
```

14.1.13 Covariance and correlation: `covariance_correlation`

`covariance_correlation` has the same arguments as `covariance`: if the sizes equal 1, `covariance_correlation` takes as argument two lists of same length or a matrix of two columns representing two random variables X and Y and otherwise `covariance_correlation` takes as argument three lists of same length, or a matrix of three columns representing two random variables X and Y and the weighting of their sizes or else a matrix M and -1 , where M gives the weighting of X (the first column of M without $M[0,0]$) and Y (the first line of M without $M[0,0]$).

`covariance_correlation` returns the list of the covariance $cov(X, Y)$ and the coefficient of linear correlation ρ of two random variables X and Y .

We have $\rho = \frac{cov(X,Y)}{\sigma(X)\sigma(Y)}$ where $\sigma(X)$ (resp. $\sigma(Y)$) designates the standard deviation of X (resp. Y).

We enter:

```
covariance_correlation([[1,1],[2,4],[3,9],[4,16]])
```

We get:

```
[25/4,100/(4*sqrt(645))]
```

Provided that $A1 := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$, we enter:

```
covariance_correlation(A1,A1^2)
```

We get:

```
[1573/12,18876/(572*sqrt(1173))]
```

We enter:

```
covariance_correlation([1,2,3,4],[1,4,9,16],[3,1,5,2])
```

Or we enter:

```
covariance_correlation([[1,1,3],[2,4,1],[3,9,5],[4,16,2]])
```

We get:

```
[662/121,662/(180*sqrt(14))]
```

We enter:

```
covariance_correlation([1,2,3,4],[1,4,9,16],
  [[3,0,0,0],[0,1,0,0],[0,0,5,0],[0,0,0,2]])
```

We get:

```
[662/121,662/(180*sqrt(14))]
```

We enter:

```
covariance_correlation(["b\a",1,2,3,4],[1,3,0,0,0],
  [4,0,1,0,0],[9,0,0,5,0],[16,0,0,0,2]),-1)
```

We get:

```
[662/121,662/(180*sqrt(14))]
```

14.1.14 Polygonal line: `polygonplot`

`polygonplot` takes as arguments two lists or a matrix of two columns.

`polygonplot` allows to display the line segments joining the different points of the cloud of dots defined by the argument and ordinates according to the increasing abscissae. If you want that the points are joined in the order supplied, you must use `listplot`.

We enter:

```
polygonplot([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

Or we enter, because the points will be ordered according to the increasing abscissae:

```
polygonplot([[2,4],[0,0],[3,9],[1,1],[4,16]])
```

Or we enter:

```
polygonplot([0,1,2,3,4],[0,1,4,9,16])
```

The graphic windows automatically opens and we get the plot of 4 segments joining the 5 points $((0,0), \dots, (4,16))$, provided that the plot configuration has been correctly defined (menu `Cfg`).

14.1.15 Polygonal line: `plotlist`

`plotlist` takes as argument a list `l` or a matrix of two columns.

`listplot` or `plotlist` allows to display the segments joining the cloud of plots having for abscissa $[0,1,2,\dots,n]$ and for ordinate `l` or for coordinates a line of the matrix. `plotlist` connects by two line segments the different points of the cloud, but without reordering the points, unlike `polygonplot` which reorders the points according to their abscissa, then connects them.

We enter:

```
plotlist([0,1,4,9,16])
```

Or we enter:

```
plotlist([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

The graphic windows automatically opens and we get, provided that the plot configuration has been correctly defined (menu Cfg):

the plot of 5 points $((0,0), (1,1), \dots (4,16))$ connected by 4 segments

We enter, if A is a matrix of 5 rows and 2 columns:

```
A:=[[0,0],[1,1],[5,4],[3,9],[4,16]]
listplot(A[0..4,0..1])
```

The graphic windows automatically opens and we get:

The 5 points joined by 4 segments

Please note the difference between:

```
listplot([[0,0],[1,1],[5,4],[3,9],[4,16]])
polygonplot([[0,0],[1,1],[5,4],[3,9],[4,16]])
```

Warning!

```
listplot([0,1,2,3,4],[0,1,4,9,16])
or
listplot([[0,1,2,3,4],[0,1,4,9,16]])
is not valid!
```

14.1.16 Polygonal line and cloud of plots: `polygonscatterplot`

`polygonscatterplot` takes as arguments two lists or a matrix of two columns.
`polygonscatterplot` allows to display the cloud of dots defined by the argument, by joining by line segments the different points of the cloud, ordering them according to the increasing abscissae .

We enter:

```
polygonscatterplot([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

Or we enter:

```
polygonscatterplot([0,1,2,3,4],[0,1,4,9,16])
```

The graphic windows automatically opens and we get the plot of 5 points $((0,0), \dots (4,16))$ joined by 4 segments, provided that the plot configuration has been correctly defined (menu Cfg).

14.1.17 Linear interpolation: `linear_interpolate`

Considering a matrix of two lines giving points coordinates: once the abscissae of these points have been sorted, these points define a polygonal line. We want to get the points coordinates of this line regularly distributed.

`linear_interpolate` has four arguments, a two line matrix $A1$ giving the coordinates of the points of a polygonal line, the minimum value of x (x_{min}), the maximum value of x (x_{max}), and the step (x_{step}).

`linear_interpolate` returns the coordinates of the points of the polygonal line for x growing from x_{min} to x_{max} with a step of x_{step} .

Note: we must have `xmin` and `xmax` being in the interval `[min(A1[0]);max(A1[0])]`.

We enter:

```
linear_interpolate([[1,2,6,9],[3,4,6,12]],1,9,1)
```

We get:

```
[[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0],
 [3.0,4.0,4.5,5.0,5.5,6.0,8.0,10.0,12.0]]
```

We enter:

```
linear_interpolate([[1,2,6,9],[3,4,6,12]],2,7,1)
```

We get:

```
[[2.0,3.0,4.0,5.0,6.0,7.0],[4.0,4.5,5.0,5.5,6.0,8.0]]
```

We enter:

```
linear_interpolate([[1,2,9,6],[3,4,6,12]],1,9,1)
```

We get:

```
[[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0],
 [3.0,4.0,6.0,8.0,10.0,12.0,10.0,8.0,6.0]]
```

14.1.18 Linear regression: `linear_regression`

To fit the data the best way by the line of the least squares having for equation $y = mx + b$, we use `linear_regression` which returns the paired value (m, b) .

If the data are x_i, y_i with $i = 1..n$, we have:

$$m = \frac{\text{cov}(X,Y)}{\sigma(X)^2} \text{ and } b = \bar{Y} - m\bar{X}$$

because the sum of the squares of the distances $d_i = |y_i - mx_i - b_i|$ is minimal for these values and this minimum (which is then the average vertical quadratic error) equals $(1 - \rho^2)\sigma(Y)^2$ where r is the correlation coefficient ($\rho = \frac{\text{cov}(X,Y)}{\sigma(X)\sigma(Y)}$).

`linear_regression` has the same arguments as `covariance`.

We enter:

```
linear_regression([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

Or we enter:

```
linear_regression([0,1,2,3,4],[0,1,4,9,16])
```

We get:

```
4, -2
```

it is then the linear function equation $y = 4x - 2$ which fits the data the best.

We enter:

```
X1:= [0,1,2,3,4,5,6,7,8,9,10]
```

```

Y1:=[7.3,9.53,12.47,16.3,21.24,27.73,36.22, 47.31,61.78,80.68,105]
      Z1:=log(Y1)
      linear_regression(X1,Z1)

```

We get:

```
0.266729219953,1.98904252589
```

it is then the linear function equation $z = \ln(y) = 0.267x + 1.99$ which fits the data the best.

14.1.19 Exponential regression: `exponential_regression`

To fit the data by an exponential function equation $y = be^{mx} = ba^x$, we use `exponential_regression` which returns the paired value (a, b) . `exponential_regression` has the same arguments as `covariance`.

We enter:

```
evalf(exponential_regression([[1,1],[2,4],[3,9],[4,16]]))
```

Or we enter:

```
evalf(exponential_regression([1,2,3,4],[1,4,9,16]))
```

We get:

```
2.49146187923,0.5
```

it is then the exponential function of equation $y = 0.5 * (2.49146187923)^x$ which fits the data the best.

We enter:

```

X1:=[0,1,2,3,4,5,6,7,8,9,10]
Y1:=[7.3,9.53,12.47,16.3,21.24,27.73,36.22,47.31, 61.78,80.68,105]
      exponential_regression(X1,Y1)

```

We get:

```
1.30568684451,7.30853268031
```

it is then the function exponential of equation $y = 7.3 * (1.3)^x$ which fits the data the best. We check by entering:

```
e^[linear_regression(X1,ln(Y1))]
```

We get:

```
1.30568684451,7.30853268031
```

14.1.20 Logarithmic regression: `logarithmic_regression`

To fit the data by a logarithmic function equation $y = m\ln(x) + b$, we use `logarithmic_regression` which returns the paired value (m, b) . `logarithmic_regression` has the same arguments as `covariance`.

We enter:

```
evalf(logarithmic_regression([[1,1],[2,4],[3,9],[4,16]]))
```

Or we enter:

```
evalf(logarithmic_regression([1,2,3,4],[1,4,9,16]))
```

We get:

```
10.1506450002,-0.564824055818
```

it is then the logarithmic function of equation $y = 10.15 \ln(x) - 0.565$ which fits the data the best.

We enter:

```
X1:=[1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8]
```

```
Y1:=[1.6,2.15,2.65,3.12,3.56,3.99,4.4,4.8,5.18,  
5.58,5.92,6.27,6.62,7.06,7.3]
```

```
logarithmic_regression(X1,Y1)
```

We get:

```
2.83870854646,0.843078064152
```

it is then the logarithmic function of equation $y = 0.84 \ln(x) + 2.84$ which fits the data the best.

We check by entering:

```
linear_regression(ln(X1),Y1)
```

We get:

```
2.83870854646,0.843078064152
```

and the correlation coefficient is:

```
correlation(ln(X1),Y1)
```

We get:

```
0.977939822434
```

We can also enter to look for a better approximation:

```
logarithmic_regression(X1,log(Y1))
```

We get:

```
0.732351031846,0.467599676658
```

it is then the function logarithmique of equation $z = \ln(y) = 0.73 \ln(x) + 0.47$ which fits the data the best.

We check by entering:

```
linear_regression(ln(X1),ln(Y1))
```

We get:

```
0.732351031846,0.467599676658
```

and the correlation coefficient is:

```
correlation(ln(X1),ln(Y1))
```

We get:

```
0.999969474543
```

14.1.21 Polynomial regression: `polynomial_regression`

To fit the data by a polynomial function of degree $\leq n$ of equation $y = a_0x^n + \dots + a_n$, we use, putting the degree n as last parameter, `polynomial_regression` which returns the list $[a_n, \dots, a_0]$.

`polynomial_regression` has the same first arguments as `covariance`, the last argument being the degree of the polynomial returned.

We enter:

```
polynomial_regression([[1,1],[2,4],[3,9],[4,16]],3)
```

Or we enter:

```
polynomial_regression([1,2,3,4],[1,4,9,16],3)
```

We get:

```
[0,1,0,0]
```

it is then the polynomial function equation $y = 0 * x^3 + x^2 + 0 * x + 0 = x^2$ which fits the data the best.

Note: we will notice that the equation of the curve represented as well as the value of the correlation coefficient of data are written in blue.

If we want to get the equation and/or the correlation coefficient on the plot we must add as last argument the option `equation` and/or `correlation`.

14.1.22 Power regression: `power_regression`

To fit the data by a function power equation $y = bx^m$, we use `power_regression` which returns the paired value (m, b) .

`power_regression` has the same arguments as `covariance`.

We enter:

```
evalf(power_regression([[1,1],[2,4],[3,9],[4,16]]))
```

Or we enter:

```
evalf(power_regression([1,2,3,4],[1,4,9,16]))
```

We get:

```
(2.0,1.0)
```

so $y = x^2$ is the function power which fits the data the best.

We enter:

```
X1:= [1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8]
Y1:= [1.6,2.15,2.65,3.12,3.56,3.99,4.4,4.8,5.18,
      5.58,5.92,6.27,6.62,7.06,7.3]
```

```
power_regression(X, Y)
```

We get:

```
0.732351031846, 1.59615829535
```

it is then the function power of of equation $y = 1.6 * x^{0.73}$ which fits the data the best.

We check by entering:

```
linear_regression(ln(X1), ln(Y1))
```

We get:

```
0.732351031846, 0.467599676658
```

We do have:

```
e^0.467599676658=1.59615829535
```

so

$\ln(y) = \ln(1.59615829535) + \ln(x) * 0.732351031846$

$\ln(y) = 0.467599676659 + \ln(x) * 0.732351031846$

and the correlation coefficient is:

```
correlation(ln(X1), ln(Y1))
```

We get:

```
0.999969474543
```

14.1.23 Logistic regression: `logistic_regression`

The logistic curves are of curves whose equation $y = y(x)$ are solutions of a differential equation of the form:

$y'/y = a * y + b$ and $y_0 = y(x_0)$ with $a < 0$ and $b > 0$.

The solutions are of the form: $y(x) = C/(1 + \exp(-\alpha(x - x_0 - k)))$ with $C = -b/a$, $\alpha = -b$ and $y_0 = (-b/a)/(1 + \exp(-b * k))$ thus

$k = -1/b * (\ln(-((a * y_0 + b)/(a * y_0))))$ To check, we can enter:

```
normal(desolve(y'/y=a*y+b)
```

We get:

```
(-b*exp(-(b*c_0-b*x)))/(a*exp(-(b*c_0-b*x))-1)
```

Then, we can enter to check:

```
normal(desolve([y'/y=a*y+b, y(x0)=y0], y)
```

We get:

```
[(-b*exp(b*x-b*x0+ln(y0/(a*y0+b))))/(a*exp(b*x-b*x0+ln(y0/(a*y0+b)))-1)]
```

We have then: $c_0 = x_0 - \ln(y_0/(a * y_0 + b))/b$

Thus, by multiplying the numerator and denominator of $y(x)$ by $\exp(b * c_0 - b * x)$, we have:

$y(x) = (-b/(\exp(b * c_0 - b * x) * a * \exp(-b * c_0 - b * x)) - 1)$

so $y(x) = -b/(a - \exp(b * (x - c_0))) = (-b/(a * (1 - \exp(b * (x - c_0))/a))$

We have $1/a = -\exp(-\ln(-a))$ because $a < 0$

then $y(x) = (-b/a) * (1/(1 + \exp(b * (x - c_0) - \ln(-a))))$ which is indeed the form announced.

When we know the values of f' at $x = x_0, x_0 + 1 \dots x_0 + n$, we look for a logistic function $y(x)$ such as $y'(x)$ fits the different values of $f'(x)$ the best.

`logistic_regression` takes as parameters:

- a list `L1` which stores the values of y' to $x = x_0, x_0 + 1 \dots x_0 + n$,
- the value `x0` of x_0
- the value `y0` of $y(x_0)$ when we know it, otherwise the calculator gets to estimate it...

`logistic_regression(L1, x0, y0)` returns the functions $y(x)$ and $y'(x)$, the constant `C`, `y1M` and `xM` with `y1M` is the value $y'(xM)$ which is the maximum of y' obtained in $x = xM$, and then the linear correlation coefficient `R` of $Y = y'/y$ function of y with the line $Y = a * y + b$.

From the list `L1`, the calculator returns the list `Ly` by using the formula $y(t + 1) - y(t) = y'(t)$, thus, we have `Ly=[y0, y0+y0', y0+y0'+y1', ...]`.

Then, the CAS performs a linear regression of `L/Ly` in term of `Ly` to get the values of a and b ($y'/y = a * y + b$ and $y_0 = y(x_0)$) then finds the solution of this differential equation.

We enter:

```
logistic_regression([0.0,1.0,2.0,3.0,4.0],0,1)
```

We get, written in blue, the signification of the values returned:

```
[(-17.77)/(1+exp(-0.496893925384*x+2.82232341488+3.14159265359*i)),
      (-2.48542227469)/(1+cosh(-
      0.496893925384*x+2.82232341488+3.14159265359
-17.77,-1.24271113735,5.67993141131+6.32246138079*i, 0.307024935856]
```

We enter:

```
evalf(logistic_regression([1,2,4,6,8,7,5],0,2))
```

Or we enter:

```
logistic_regression(evalf([1,2,4,6,8,7,5]),0,2.0))
```

We get:

```
[64.8358166583/(1.0+exp(-0.551746244591*x+2.95837880348)),
14.4915280084/(1.0+cosh(-0.551746244591*x+2.95837880348)),
64.8358166583,7.24576400418,5.36184674112,-0.81176431297]
```

To retrieve the value -0.81176431297 of the correlation coefficient, we enter:

```
L:=[1,2,4,6,8,7,5];
```

```
y0:=2.0;
```

```
Ly:=makelist(y0,1,size(L))+cumSum(L)
```

We get:

```
[3,5,9,15,23,30,35]
```

then

```
correlation(L/Ly,Ly)
```

which returns

```
-0.81176431297
```


Chapter 15 Statistics

15.1 Statistics functions on a list: mean, variance, stddev, stddevp, median, quantile, quartiles, quartile1, quartile3

See also 15.1.1 and 14.

Useful functions for statistics whose data are lists:

- `mean` to calculate the mean of the elements of a list.

We enter:

```
mean([3,4,2])
```

We get:

```
3
```

We enter:

```
mean([1,0,1])
```

We get

```
2/3
```

- `stddev` to calculate the numerical standard deviation of the elements of a list.

We enter:

```
stddev([3,4,2])
```

We get:

```
sqrt(2/3)
```

We have indeed the mean which equals 3 and the standard deviation which equals:

$$\sqrt{\frac{(3-3)^2 + (4-3)^2 + (2-3)^2}{3}} = \sqrt{\frac{2}{3}}$$

- `stddevp` to calculate an estimation of the numerical standard deviation of the population from a sample whose elements are supplied in a list.

We enter:

```
stddevp([3,4,2])
```

We get:

```
1
```

We have indeed the mean which equals 3 and the standard deviation which equals:

$$\sqrt{\frac{(3-3)^2 + (4-3)^2 + (2-3)^2}{2}} = \sqrt{\frac{2}{2}} = 1$$

We have the relation:

$$\text{stddevp}(1)^2 = \text{size}(1) * \text{stddev}(1)^2 / (\text{size}(1) - 1).$$

- `variance` to calculate the numerical variance of the elements of a list.

We enter:

```
variance([3,4,2])
```

We get:

2/3

- `median` to calculate the median of the elements of a list.

We enter:

```
median([0,1,3,4,2,5,6])
```

We get:

3.0

- `quantile` to calculate the deciles of the elements of a list.

We enter:

```
quantile([0,1,3,4,2,5,6],0.25)
```

We get the first quartile:

[1.0]

We enter:

```
quantile([0,1,3,4,2,5,6],0.5)
```

We get the median:

[3.0]

We enter:

```
quantile([0,1,3,4,2,5,6],0.75)
```

We get the third quartile:

[5.0]

- `quartiles` returns the minimum, the first quartile, the median, the third quartile and the maximum of a statistical series.

We enter:

```
quartiles([0,1,3,4,2,5,6])
```

We get:

```
[[0.0],[1.0],[3.0],[5.0],[6.0]]
```

- `quartile1` returns the first quartile of a statistical series.

We enter:

```
quartile1([0,1,3,4,2,5,6])
```

We get:

```
1.0
```

- `quartile3` returns the third quartile of a statistical series.

We enter:

```
quartile3([0,1,3,4,2,5,6])
```

We get:

```
5.0
```

Be A the list `[0,1,2,3,4,5,6,7,8,9,10,11]`.

We enter:

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

We get:

```
11/2 for mean(A)
sqrt(143/12) for stddev(A)
0 for min(A)
[1.0] for quantile(A,0.1)
[2.0] for quantile(A,0.25)
[5.0] for median(A) or for quantile(A,0.5)
[8.0] for quantile(A,0.75)
[9.0] for quantile(A,0.9)
11 for max(A)
[[0.0],[2.0],[5.0],[8.0],[11.0]] for quartiles(A)
```

See also these functions for matrices at section 15.1.1 and for weighted lists at chapter 14.

15.1.1 Statistics functions on the columns of a matrix: `mean`, `stddev`, `variance`, `median`, `quantile`, `quartiles`

See also 15.1 and 14.

Useful functions for statistics whose data are the columns of a matrix:

- `mean` to calculate the mean numerical of statistical series which are the columns of a matrix.

We enter:

```
mean([[3,4,2],[1,2,6]])
```

We get a vector whose components are the mean of columns:

```
[2, 3, 4]
```

We enter:

```
mean([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
```

We get

```
[1/3, 1/3, 1/3]
```

- `stddev` to calculate the numerical standard deviation of statistical series which are the columns of a matrix.

We enter:

```
stddev([[3, 4, 2], [1, 2, 6]])
```

We get a vector whose components are the standard deviation of columns:

```
[1, 1, 2]
```

- `variance` to calculate the numerical variance of statistical series which are the columns of a matrix.

We enter:

```
variance([[3, 4, 2], [1, 2, 6]])
```

We get a vector whose components are the variance of columns:

```
[1, 1, 4]
```

- `median` to calculate the median of statistical series which are the columns of a matrix.

We enter:

```
median([[6, 0, 1, 3, 4, 2, 5], [0, 1, 3, 4, 2, 5, 6], [1, 3, 4, 2, 5, 6, 0],
        [3, 4, 2, 5, 6, 0, 1], [4, 2, 5, 6, 0, 1, 3], [2, 5, 6, 0, 1, 3, 4]])
```

We get a vector whose components are the median of columns:

```
[2.0, 2.0, 3.0, 3.0, 2.0, 2.0, 3.0]
```

- `quantile` to calculate the decile according to the second argument, of statistical series which are the columns of a matrix.

We enter:

```
quantile([[6, 0, 1, 3, 4, 2, 5], [0, 1, 3, 4, 2, 5, 6], [1, 3, 4, 2, 5, 6, 0],
          [3, 4, 2, 5, 6, 0, 1], [4, 2, 5, 6, 0, 1, 3], [2, 5, 6, 0, 1, 3, 4]], 0.25)
```

We get a vector whose components are the first quartile of columns:

```
[1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0]
```

We enter:

```
quantile([[6, 0, 1, 3, 4, 2, 5], [0, 1, 3, 4, 2, 5, 6], [1, 3, 4, 2, 5, 6, 0],
          [3, 4, 2, 5, 6, 0, 1], [4, 2, 5, 6, 0, 1, 3], [2, 5, 6, 0, 1, 3, 4]], 0.75)
```

We get a vector whose components are the third quartile of columns:

```
[4.0,4.0,5.0,5.0,5.0,5.0,5.0]
```

- `quartiles` to calculate the minimum, the first quartile, the median, the third quartile and the maximum of statistical series which are the columns of a matrix.

We enter:

```
quartiles([[6,0,1,3,4,2,5],[0,1,3,4,2,5,6],[1,3,4,2,5,6,0],
          [3,4,2,5,6,0,1],[4,2,5,6,0,1,3],[2,5,6,0,1,3,4]])
```

We get the matrix, of first line the minimum of each column, of second line the first quartile of each column, of third line the median of each column, of fourth line the third quartile of each column and last line the maximum of each column:

```
[[0.0,0.0,1.0,0.0,0.0,0.0,0.0],[1.0,1.0,2.0,2.0,1.0,1.0,1.0],
 [2.0,2.0,3.0,3.0,2.0,2.0,3.0],[4.0,4.0,5.0,5.0,5.0,5.0,5.0],
 [6.0,5.0,6.0,6.0,6.0,6.0,6.0]]
```

15.2 Tables indexed by two strings: `table`

A table is a list indexed by something more general than integers.

A table can be used, for example, to store of telephone numbers indexed by two names.

In CAS, the index of a table can be any objects of the CAS.

The access is done by an algorithm which sorts by type then uses the order of each type (for example < for numerical type, lexicographical order for strings, etc., ...).

`table` takes as argument a list or a sequence of equalities of the form:

```
"index_name"=value_element.
```

`table` returns this table.

We enter:

```
T:=table(3=-10,"a"=10,"b"=20,"c"=30,"d"=40)
```

We enter:

```
T["b"]
```

We get:

```
20
```

We enter:

```
T[3]
```

We get:

```
-10
```

Example

We want to encode the letters "a","b",... "z" by 1,2,...26.

We enter:

```
alphab:="abcdefghijklmnopqrstuvwxy";
```

then:

```
code:=table(seq(alphab[j]=j+1,j=0..25));
```

We enter

```
code["c"]
```

We get

```
3
```

or we write a function:

```
Code(a):={
local code,alphab,j;
alphab:="abcdefghijklmnopqrstuvwxy";
code:=table(seq(alphab[j]=j+1,j=0..25));
return code(a);
};
```

We enter

```
Code("c")
```

We get

```
3
```

Note:

If we do an assignment of the type $T[n] := \dots$ where T is the name of a variable and n an integer

- if the variable T stores a list or a sequence, then the n -th element of T is modified,
- if the variable T is not assigned, a table T is created with an entry (corresponding to the index n). Note that once this assignment is done, T is not a list, even though n is an integer.

Chapter 16 Lists

16.1 Function MAKELIST makelist

In HOME, `MAKELIST` creates a list from a symbolic expression.

For instance, we create a list starting from $X^2 + 1$, by having the variable X growing from 2 to 6 with a step of 1 (1 can be omitted), we enter:

```
MAKELIST (X^2+1, X, 2, 6)
```

We get:

```
{5, 10, 17, 26, 37}
```

We enter:

```
MAKELIST (0, X, 1, 10)
```

We get:

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

We create a list starting from $X^2 + 1$, by having the variable X growing from 2 to 6 with a step of 2, we enter:

```
MAKELIST (X^2+1, X, 2, 6, 2)
```

We get:

```
{5, 17, 37}
```

In CAS, we can use `MAKELIST` and `makelist`. `makelist` has a function as first argument, the second argument represents the initial value of the variable and the third argument represents its final value. We can put a fourth argument which represents the step of the variable.

Warning! The index also starts at 1.

We enter:

```
makelist (x->x^2, 1, 10)
```

We get:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

We enter:

```
makelist (x->x^2, 1, 10, 2)
```

We get:

```
[1, 9, 25, 49, 81]
```

16.2 Function SORT sort

`SORT` or `sort` sorts by increasing order the components of a list.

We enter:

```
SORT ([12, 2, 31, 24, 15])
```

We get:

```
[2, 12, 15, 24, 31]
```

We enter:

```
SORT ({12, 2, 31, 24, 15})
```

We get:

```
{2, 12, 15, 24, 31}
```

16.3 Function REVERSE

`REVERSE` creates a list by reversing the order of the elements.

We enter:

```
REVERSE ([1, 22, 3, 4, 5])
```

We get:

```
[5, 4, 3, 22, 1]
```

We enter:

```
REVERSE ({1, 22, 3, 4, 5})
```

We get:

```
{5, 4, 3, 22, 1}
```

16.4 Concatenate: CONCAT concat

`CONCAT` or `concat` concatenates two lists or two vectors or two strings of characters or two matrices (the two matrices must have the same number of rows, and will be concatenated line by line.)

We enter:

```
CONCAT ([1, 2, 3], [4, 5])
```

We get:

```
[1, 2, 3, 4, 5]
```

We enter:

```
CONCAT ({1, 2, 3}, {4, 5})
```

We get:

```
{1, 2, 3, 4, 5}
```

We enter:

```
CONCAT ("HE", "LLO")
```

We get:

```
"HELLO"
```

We enter:

```
2=>A
```

```
CONCAT ([1, A, 3], [4, 5])
```

We get:

```
[1, 2, 3, 4, 5]
```

We enter:

```
CONCAT ([[1, 2], [3, 4]], [[4, 5, 6], [6, 7, 8]])
```

We get:

```
[[1, 2, 4, 5, 6], [3, 4, 6, 7, 8]]
```

We enter:

```
2=>A
```

```
CONCAT ({1, A, 3}, {4, 5})
```

We get:

```
{1, 2, 3, 4, 5}
```

To concatenate a string and a list into a list we use `CONCAT`.

We enter:

```
2=>A
```

```
CONCAT ([1, A, 3]), "L1"
```

We get:

```
[1, 2, 3, "L1"]
```

We enter:

```
2=>A
```

```
CONCAT ("L1", [1, A, 3])
```

We get:

```
["L1", 1, 2, 3]
```

Warning!

To concatenate a string and a list into a string we use +.

We enter:

```
2=>A
"L1="+[1,A,3]
```

We get:

```
"L1=[1,2,3]"
```

We enter:

```
2=>A
[1,A,3]+"L1="
```

We get:

```
"[1,2,3]L1="
```

We enter:

```
2=>A
"L1="+{1,A,3}
```

We get:

```
"L1={1,2,3}"
```

16.4.1 Add an element at the end of a list: `append`

`append` adds an element at the end of a list.

We enter:

```
append([3,4,2],1)
```

We get:

```
[3,4,2,1]
```

We enter:

```
append([1,2],[3,4])
```

We get:

```
[1,2,[3,4]]
```

16.4.2 Add an element at the beginning of a list: `prepend`

`prepend` adds an element at the beginning of a list.

We enter:

```
prepend([3,4,2],1)
```

We get:

```
[1, 3, 4, 2]
```

We enter:

```
prepend([1, 2], [3, 4])
```

We get:

```
[[3, 4], 1, 2]
```

16.5 Position in a list: POS

`POS` returns the position of an element in a list, that is to say `POS` returns the index of the first occurrence of the element or 0 if the element is not in the list.

We enter:

```
POS([4, 3, 1, 2, 3, 4, 5], 4)
```

Or we enter:

```
POS({4, 3, 1, 2, 3, 4, 5}, 4)
```

We get:

```
1
```

We enter:

```
POS([4, 3, 1, 2, 3, 4, 5], 2)
```

Or we enter:

```
POS({4, 3, 1, 2, 3, 4, 5}, 2)
```

We get:

```
4
```

We enter:

```
POS([4, 3, 1, 2, 3, 4, 5], 6)
```

Or we enter:

```
POS({4, 3, 1, 2, 3, 4, 5}, 6)
```

We get:

```
0
```

16.6 Function DIM dim SIZE size length

`SIZE` or `size` or `DIM` or `dim` or `length` returns the length of the list (or of the strings) supplied as argument.

Warning!

In HOME and in the CAS `SIZE` returns the dimension of a matrix whereas in the CAS `size` returns the number of line of a matrix.

We enter in HOME:

```
SIZE({1,2,3})
```

We get:

```
3
```

We enter:

```
SIZE([[1,2,3],[4,5,6]])
```

We get:

```
{2,3}
```

We enter in the CAS:

```
size([1,2,3])
```

We get:

```
3
```

We enter:

```
size([[1,2,3],[4,5,6]])
```

We get:

```
2
```

Warning!

We enter in the CAS:

```
SIZE([[1,2,3],[4,5,6]])
```

We get:

```
[2,3]
```

16.6.1 Get the reversed list: revlist

`revlist` takes as argument a list (resp. a sequence).
`revlist` returns the list (resp. the sequence) in reversed order.

We enter:

```
revlist([0,1,2,3,4])
```

We get:

```
[4,3,2,1,0]
```

We enter:

```
revlist([0,1,2,3,4],3)
```

We get:

```
3, [0, 1, 2, 3, 4]
```

16.6.2 Get the list swapped starting from its n-th element: `rotate`

`rotate` takes as argument a list and an relative integer (by default $n=-1$).

`rotate` returns:

- if $n>0$: the list obtained by swapping the n first elements with the end of the list,
- if $n<0$: the list obtained by swapping the $-n$ last elements with the beginning of the list. By default, $n=-1$ and we put the last element in first position.

We enter:

```
rotate([0, 1, 2, 3, 4])
```

We get:

```
[4, 0, 1, 2, 3]
```

We enter:

```
rotate([0, 1, 2, 3, 4], 2)
```

We get:

```
[2, 3, 4, 0, 1]
```

We enter:

```
rotate([0, 1, 2, 3, 4], -2)
```

We get:

```
[3, 4, 0, 1, 2]
```

16.6.3 Get the list shifted starting from its n-th element: `shift`

`shift` takes as argument a list and an relative integer (by default $n=-1$).

`shift` returns:

- if $n>0$: the list obtained by replacing the n first elements of the list by `undef`, then by swapping these n first elements with the end of the list,
- if $n<0$: the list obtained by replacing the $-n$ last elements of the list by `undef`, then by swapping the $-n$ last elements with the beginning of the list. By default ($n=-1$) the first element equals `undef` and is followed by the list whose last element is removed.

We enter:

```
shift([0, 1, 2, 3, 4])
```

We get:

```
[undef, 0, 1, 2, 3]
```

We enter:

```
shift([0, 1, 2, 3, 4], 2)
```

We get:

```
[2, 3, 4, undef, undef]
```

We enter:

```
shift([0, 1, 2, 3, 4], -2)
```

We get:

```
[undef, undef, 0, 1, 2]
```

16.6.4 Removing an element from a list: `suppress`

`suppress` removes from a list the element of supplied index.

Warning! The index of the first element is 0.

We enter:

```
suppress([3, 4, 2], 1)
```

We get:

```
[3, 2]
```

16.6.5 Get the list without its first element: `tail`

`tail` returns the list without its first element.

We enter:

```
tail([0, 1, 2, 3])
```

We get:

```
[1, 2, 3]
```

`l:=tail([0, 1, 2, 3])` is equivalent to `l:=suppress([0, 1, 2, 3], 0)`

16.6.6 Removing elements from a list: `remove`

`remove` has two parameters: a boolean function `f` and a list `l`.

`remove` removes the elements `c` from the list `l`, which checks `f(c)=true`.

We enter:

```
remove(x->(x>=2), [0, 1, 2, 3, 4, 5])
```

We get:

```
[0, 1]
```

Note

To do the same thing with a string of characters, for example, remove all the "a" from a string:

We enter:

```
ord("a")
```

We get:

We enter:

```
f(chn) := { local l := length(chn); return
remove(x->(ord(x) == 97), seq(chn[k], k, 1, l)); }
```

Then, we enter:

```
f("abracadabra")
```

We get:

```
["b", "r", "c", "d", "b", "r"]
```

Then, we enter:

```
char(ord(["b", "r", "c", "d", "b", "r"]))
```

We get:

```
"brcdbr"
```

16.6.7 Right and left part straight of a list: `right`, `left`

- `right(l, n)` returns the n last elements of a list l .

We enter:

```
right([1, 2, 3, 4, 5, 6], 4)
```

We get:

```
[3, 4, 5, 6]
```

- `left(l, n)` returns the n first elements of a list l .

We enter:

```
left([1, 2, 3, 4, 5, 6, 7, 8], 3)
```

We get:

```
[1, 2, 3]
```

16.6.8 Checking whether an element is in a list: `member`

`member` has two parameters: an element c and a list (or a set) L .
`member` is a function which checks whether the element c is in the list L .
`member` returns 0 if c is not in L , and returns otherwise:

```
"the index of its first occurrence".
```

Warning! For sake of compatibility, please mind the order of the parameters!

We enter:

```
member(2, [1, 2, 3, 4, 2])
```

We get:

2

We enter:

```
member(2, % {1, 2, 3, 4, 2% })
```

We get:

2

16.6.9 Checkin whether an element is in a list: `contains`

`contains` has two parameters: a list (or a set) L and an element c .
`contains` is a function which checks whether the element c is in the list L .
`contains` returns 0 if c is not in L , and otherwise returns:

```
"the index of its first occurrence".
```

Warning! For sake of compatibility, please mind the order of the parameters!

We enter:

```
contains([1, 2, 3, 4, 2], 2)
```

We get:

2

We enter:

```
contains(% {1, 2, 3, 4, 2% }, 2)
```

We get:

2

16.6.10 Counting the elements of a list or of a matrix such as a property: `count`

Depending on its parameters, `count` is able to count in a list l the number of elements:

- equal to a with `count(x->x==a, l)`,
- greather than a with `count(x->x>a, l)`,
- lower than a with `count(x->x<a, l)`,
- with `count(x->>1, l)`

Indeed, `count` has one, two or three parameters:

1. a list of integers l
2. a real function f ,
 - list l of length n or a matrix a of dimension $p \times q$,
 - an optional argument `row` or `col`, in case of the second parameter is a matrix a .

When `count` has:

- one parameter which is a list of integers l , `count(l)` counts the number of occurrences by returning a matrix of first column the elements of the list l sorted, and second column the number of occurrences of this element in the list.
- two parameters, `count` applies the function to the elements of the list (or of the matrix) and in returns the sum, that is to say, `count(f, l)` returns the number $f(l[0]) + f(l[1]) + \dots + f(l[n-1])$ or `count(f, a)` returns the number $f(a[0, 0]) + \dots + f(a[p-1, q-1])$.
 If f is a boolean function, `count` returns the number of elements of the list (or of the matrix) for which the boolean function is true.

- three parameters, `count` applies the function to the elements of each line (resp. column) of the matrix `a` if the optional argument is `row` (resp. `col`) and returns a list of length `p` having as `k`-nth element:

$f(a[k,0]) + \dots + f(a[k,q-1])$ (resp. a list of length q having as k -nth element: $f(a[0,k]) + \dots + f(a[p-1,k])$).

We enter:

```
count([1,3,1,1,2,10,3])
```

We get:

```
[[1,3],[2,1],[3,2],[10,1]]
```

We enter:

```
count((x)->x,[2,12,45,3,7,78])
```

Or we enter:

```
count((x)->x,[[2,12,45],[3,7,78]])
```

We get:

```
147
```

because we have: $2 + 12 + 45 + 3 + 7 + 78 = 147$.

We enter:

```
count((x)->x,[[2,12,45],[3,7,78]],row)
```

We get:

```
[59,88]
```

because we have: $2 + 12 + 45 = 59$ and $3 + 7 + 78 = 88$.

We enter:

```
count((x)->x,[[2,12,45],[3,7,78]],col)
```

We get:

```
[5,19,123]
```

because we have: $2+3=5, 12+7=19, 45+78=123$ Tapez une équation ici..

We enter:

```
count((x)->x<12,[2,12,45,3,7,78])
```

We get:

```
3
```

Indeed, $(x) \rightarrow x < 12$ is a boolean function which equals 1 if $x < 12$ and 0 otherwise. We have then $1 + 0 + 0 + 1 + 1 + 0 = 3$.

We enter:

```
count((x) -> x == 12, [2, 12, 45, 3, 7, 78])
```

Or we enter:

```
count((x) -> x == 12, [[2, 12, 45], [3, 7, 78]])
```

We get:

1

Indeed, $(x) \rightarrow x == 12$ is a boolean function which equals 1 if $x == 12$ and 0 otherwise. We get then the number of terms equal to 12. Here it is 1.

We enter:

```
count((x) -> x > 12, [2, 12, 45, 3, 7, 78])
```

We get:

2

Indeed, $(x) \rightarrow x > 12$ is a boolean function which equals 1 if $x > 12$ and 0 otherwise. We have then $0 + 0 + 1 + 0 + 0 + 1 = 2$.

We enter:

```
count(x -> x^2, [3, 5, 1])
```

We get:

35

Indeed, we have: $3^2 + 5^2 + 1^1 = 35$.

We enter:

```
count(id, [3, 5, 1])
```

We get:

9

Indeed, `id` is the function identity and we have: $3 + 5 + 1 = 9$.

We enter:

```
count(1, [3, 5, 1])
```

We get:

3

Indeed, 1 is the constant function equal to 1 and we have: $1 + 1 + 1 = 3$.

16.6.11 Select elements of a list: `select`

In CAS, `select` has two parameters: a boolean function f and a list L . `select` selects the elements c of the list L which checks $f(c) = \text{true}$.

We enter:

```
select (x->(x>=2), [0,1,2,3,4,5])
```

We get:

```
[2,3,4,5]
```

16.7 List of differences between consecutive terms: Δ LIST deltalist

In HOME, (resp. CAS), Δ LIST (resp. deltalist) returns the list of differences between the components of the list supplied as argument.

We enter in HOME:

```
 $\Delta$ LIST([1,21,34,41,52])
```

We get:

```
[20,13,7,11]
```

We enter in HOME:

```
 $\Delta$ LIST({1,21,34,41,52})
```

We get:

```
{20,13,7,11}
```

We enter in the CAS:

```
deltalist([1,21,34,41,52])
```

We get:

```
[20,13,7,11]
```

16.8 Sum of the elements of a list: Σ LIST sum

In HOME, Σ LIST returns the sum of components of the list supplied as argument.

We enter:

```
 $\Sigma$ LIST([1,2,3,4,5])
```

We get:

```
15
```

We enter:

```
 $\Sigma$ LIST({1,2,3,4,5})
```

We get:

```
15
```

In CAS, `sum` returns the sum of the components of the list supplied as argument.

We enter:

```
sum([1, 2, 3, 4, 5])
```

We get:

```
15
```

16.9 Product of the elements of a list: `ΠLIST` product

In HOME, `ΠLIST` returns the product of the components of the list supplied as argument.

We enter:

```
ΠLIST([1, 2, 3, 4, 5])
```

We get:

```
120
```

We enter:

```
ΠLIST({1, 2, 3, 4, 5})
```

We get:

```
120
```

In CAS, `product` returns the product of the components of the list supplied as argument.

We enter:

```
product([1, 2, 3, 4, 5])
```

We get:

```
120
```

16.9.1 Apply a function of one variable to the elements of a list: `map` `apply`

`map`, or `apply`, is used to apply a function to the elements of a list, but these two instructions are not of synonymous. We have:

- `apply` has two parameters: a function `f` and a list `L`.
`apply(f, L)` returns `[f(L[0]), f(L[1]), ... f(L[size(L)-1])]`.
Warning! `apply` answers `[]` if the second element is not a list.
- `map` has two parameters: an expression `E` or a list `L`, and a function `f`.
`map(E, f)` returns `f(E)` and `map(L, f)` returns `[f(L[0]), f(L[1]), ... f(L[size(L)-1])]`.
Warning! Please mind that, for sake of compatibility, the orders of the parameters are different for `map` and `apply`.
 When the list is a matrix and the function must apply to each element of a matrix, `matrix` must be put as optional argument to `map`.

We enter:

```
apply(x->x+1, [3, 5, 1])
```

or

```
map([3, 5, 1], x->x+1)
```

this adds 1 to each element of the list, and we get:

```
[4, 6, 2]
```

Example with a matrix

We enter:

```
apply(x->x+1, [[3, 5, 1], [3, 5, 1], [3, 5, 1]])
```

or

```
map([[3, 5, 1], [3, 5, 1], [3, 5, 1]], x->x+1)
```

this adds 1 to each element of the list, that is to say to each line of the matrix and since $[3, 5, 1]+1=[3, 5, 2]$, we get:

```
[[3, 5, 2], [3, 5, 2], [3, 5, 2]]
```

We enter:

```
map([[3, 5, 1], [3, 5, 1], [3, 5, 1]], x->x+1, matrix)
```

this adds 1 to each element of the matrix, and we get:

```
[[4, 6, 2], [4, 6, 2], [4, 6, 2]]
```

Other examples. We enter:

```
apply(x->x^2, [3, 5, 1])
```

or

```
map([3, 5, 1], x->x^2)
```

or we define the function $h(x) = x^2$ by entering:

```
h(x) := x^2
```

then

```
apply(h, [3, 5, 1])
```

or

```
map([3, 5, 1], h)
```

We get:

```
[9, 25, 1]
```

We enter:

```
apply(h, [[3, 5, 1], [3, 5, 1], [3, 5, 1]])
```

or

```
map([[3, 5, 1], [3, 5, 1], [3, 5, 1]], h)
```

or

```
map([[3,5,1],[3,5,1],[3,5,1]],h,matrix)
```

We get each element raised to square:

```
[[9,25,1],[9,25,1],[9,25,1]]
```

We define the function $g(x) = [x, x^2, x^3]$ by entering:

```
g(x):=[x,x^2,x^3]
```

or

```
g:=(x)->[x,x^2,x^3]
```

then, we enter:

```
apply(g,[3,5,1])
```

or

```
map([3,5,1],g)
```

We make g proceed on 3, on 5, then on 1, and we get:

```
[[3,9,27],[5,25,125],[1,1,1]]
```

Note:

If l_1, l_2, l_3 are lists:

```
sizes([l1,l2,l3])=map(size,[l1,l2,l3])
```

16.9.2 Apply a function of two variables to elements of two lists: `zip`

`zip` is used to apply a function of two variables to elements of two lists.

We enter:

```
zip('sum',[a,b,c,d],[1,2,3,4])
```

We get:

```
[a+1,b+2,c+3,d+4]
```

We enter:

```
zip((x,y)->x^2+y^2,[4,2,1],[3,5,1])
```

Or we enter:

```
f:=(x,y)->x^2+y^2
```

then,

```
zip(f,[4,2,1],[3,5,1])
```

We get:

```
[25,29,2]
```

We enter:

```
f:=(x,y)->[x^2+y^2,x+y]
```

then,

```
zip(f,[4,2,1],[3,5,1])
```

We get:

```
[[25,7],[29,7],[2,2]]
```

16.10 Convert a list to a matrix: `list2mat`

`list2mat` allows to get the matrix of the terms of the list supplied as argument by splitting the list according to the number of columns specified. If terms are missing, the list is supplemented by zeros.

We enter:

```
list2mat([5,8,1,9,5,6],2)
```

We get:

```
[[5,8],[1,9],[5,6]]
```

We enter:

```
list2mat([5,8,1,9],3)
```

We get:

```
[[5,8,1],[9,0,0]]
```

Note:

In the answer, the delimiters of a matrix are `[[` and `]]`, whereas the delimiters of a list are `[` and `]` (the vertical line of the brackets is thicker for the matrices).

16.11 Convert a matrix to a list: `mat2list`

`mat2list` allows to get the list of the terms of the matrix supplied as argument.

We enter:

```
mat2list([[5,8],[1,9]])
```

We get:

```
[5,8,1,9]
```

16.12 Useful functions for the lists and the components of a vector

16.12.1 Norms of a vector: `maxnorm` `l1norm` `l2norm` `norm`

See also 20.11.1 for the different instructions to get the norms of a matrix.

The different instructions to get the norms of a vector are:

- `maxnorm` to calculate the norm l^∞ of a vector: it is the maximum of the absolute values of its coordinates.

We enter:

```
maxnorm([3,-4,2])
```

Or we enter:

```
maxnorm(vector(3,-4,2))
```

We get:

4

Indeed: $x = 3, y = -4, z = 2$ and $4 = \max(|x|, |y|, |z|)$.

- `l1norm` to calculate the norm l^1 of a vector: it is the sum of the absolute values of its coordinates.

We enter:

```
l1norm([3,-4,2])
```

Or we enter:

```
l1norm(vector(3,-4,2))
```

We get:

9

Indeed: $x = 3, y = -4, z = 2$ and $9 = |x| + |y| + |z|$.

- `norm` or `l2norm` to calculate the norm l^2 of a vector: it is the square root of the sum of the squares of its coordinates.

We enter:

```
norm([3,-4,2])
```

Or we enter:

```
norm(vector(3,-4,2))
```

We get:

`sqrt(29)`

Indeed: $x = 3, y = -4, z = 2$ and $29 = |x|^2 + |y|^2 + |z|^2$.

16.12.2 Normalizing the components of a vector: `normalize`

`normalize` normalizes the components of a vector and returns the components of a vector of norm 1 according to the norm l^2 (the square root of the sum of the squares of its coordinates).

We enter:

```
normalize([3,4,5])
```

We get:

```
[3/(5*sqrt(2)), 4/(5*sqrt(2)), 5/(5*sqrt(2))]
```

Indeed: $x = 3, y = 4, z = 5$ and $50 = |x|^2 + |y|^2 + |z|^2$.

16.12.3 Cumulated sums of the elements of a list: cumSum

`cumSum` allows to do the cumulated sums of the elements of a list, or of a sequence of real numbers, or of decimals, or of string of characters.

`cumSum` takes as argument a list or a sequence.

`cumSum` returns a list or a sequence, the element of index k being obtained by doing the sum of the elements of index $1..k$.

If l is a list, `cumSum` returns the list l_r which equals $[\text{sum}(l[j], j=1..k) \text{ } (k=1..size(l))]$.

If l is a sequence, `cumSum` returns the sequence l_r which equals $\text{sum}(l[j], j=1..k) \text{ } (k=1..size(l))$.

We enter:

```
L:=cumSum(1,2,3)
```

We get:

```
1,3,6
```

We enter:

```
L:=cumSum([1,2,3])
```

We get:

```
[1,3,6]
```

We enter:

```
c[2]
```

We get:

```
3
```

16.12.4 Term by term sum of two lists: + .+

The term by term sum of two lists is done with the infix operator `+` or `.+` and also with the prefix operator `'+'`.

If the two lists are not of same length, the shortest list is supplemented by zeros.

Please note the difference with sequences: if the infix operator `+` takes as arguments two sequences, it returns the sum of the terms of the two sequences.

We enter:

```
[1,2,3]+[4,3,5]
```

Or we enter:

```
[1,2,3] .+[4,3,5]
```

Or we enter:

```
'+'([1,2,3],[4,3,5])
```

Or we enter:

```
'+'([[1,2,3],[4,3,5]])
```

We get:

$$[5, 5, 8]$$

We enter:

$$[1, 2, 3, 4, 5, 6] + [4, 3, 5]$$

Or we enter:

$$[1, 2, 3, 4, 5, 6] . + [4, 3, 5]$$

Or we enter:

$$' + ' ([[1, 2, 3, 4, 5, 6], [4, 3, 5]])$$

Or we enter:

$$' + ' ([[1, 2, 3, 4, 5, 6], [4, 3, 5]])$$

We get:

$$[5, 5, 8, 4, 5, 6]$$

Warning!

When the operator + is prefix, it must be quoted, that is to say written ' + ' .

If we enter:

$$[1, 2, 3, 4, 5, 6] + 4$$

We get, because the list is considered as the coefficients of a polynomial:

$$[1, 2, 3, 4, 5, 10]$$

16.12.5 Term by term difference of two lists: - . -

The term by term difference of two lists is done with the infix operator - or . - and also with the prefix operator ' - ' .

If the two lists are not of same length, the shorted list is supplemented by zeros.

Please note the difference with sequences: if the infix operator - takes as arguments two sequences, it returns the difference of the sums of the terms of each of the sequences.

We enter:

$$[1, 2, 3] - [4, 3, 5]$$

Or we enter:

$$[1, 2, 3] . - [4, 3, 5]$$

Or we enter:

$$' - ' ([1, 2, 3], [4, 3, 5])$$

Or we enter:

$$' - ' ([[1, 2, 3], [4, 3, 5]])$$

We get:

$$[-3, -1, -2]$$

Warning!

When the operator `-` is prefix, it must be quoted, that is to say written `' - '`.

16.12.6 Term by term product of two lists: `.*`

See also product for lists and matrices (cf 20.4.4 and ??)

The term by term product of two lists of same length is done with the infix operator `.*`.

We enter:

```
[1, 2, 3] .* [4, 3, 5]
```

We get:

```
[4, 6, 15]
```

We enter:

```
[[1, 2], [4, 3]] .* [[4, 3], [5, 6]]
```

We get:

```
[[4, 6], [20, 18]]
```

16.12.7 Quotient term by term of two lists: `./`

The quotient term by term of two lists of same length is done with the infix operator `./`.

We enter:

```
[1, 2, 3] ./ [4, 3, 5]
```

We get:

```
[1/4, 2/3, 3/5]
```

Chapter 17 Strings of characters

17.1 Write a string or a character: "

Strings of characters are written by using the quotes as delimiters (" " it is the key ALPHA 0). A character is a string of one character; indeed the delimiters ' ' or `quote` key Shift ()) are used to specify that the variable put between the quotes must not be evaluated.

Example:

"a" is a character but 'a' or `quote(a)` designates the variable a non evaluated.

The characters of a string are designated by an index (as for the lists).

To access an element of a string, we enter the index of this element between two brackets (the index which start at 1): [] [[]].

Example:

We enter:

```
"hello"[2]
```

We get:

```
"e"
```

We enter:

```
"hello"[[2]]
```

We get:

```
"e"
```

Note:

When we put a string of characters on the entry line, this generates an echo as a result.

Example:

We enter:

```
"hello"
```

We have "hello" written as a question and we get hello as answer.

We enter:

```
"hello"+" , how do you do?"
```

We get:

```
"hello, how do you do?"
```

17.1.1 To concatenate two numbers and strings: `cat` +

`+` or `cat` evaluates the arguments and concatenates them in a string. This allows so to convert a real number into a string of characters.

We enter:

```
"="+123
```

Or we enter:

```
cat("=",123)
```

We get:

```
"=123"
```

We enter:

```
a:=123
```

then,

```
"We get: "+a)
```

or

```
cat("We get: ",a)
```

We get:

```
"We get: 123"
```

17.1.2 Concatenating a sequence of words: `cumSum`

`cumSum` allows to do the concatenation of a list of strings.

`cumSum` takes as argument a list of strings.

`cumSum` returns a list of strings, the element of index k being obtained by concatenating the strings before it (*i.e* those of index $1..k-1$) with the string of index k .

If l is a list of k strings, `cumSum` returns the list `lr` equal to

```
[sum(l[j],j=1..k)$(k=1..size(l))]
```

We enter:

```
c:=cumSum("Hello ","my ","friend")
```

We get:

```
" Hello "," Hello my "," Hello my friend"
```

We enter:

```
c[2]
```

We get:

```
" Hello my "
```

17.1.3 Finding a character in a string: INSTRING inString

`inString` has two parameters: a string of characters `S` and a character `c`.

`inString` is a function which checks whether the character `c` is in the string of characters `S`.

`inString` returns 0 if `c` is not in `S` and, otherwise, returns "the index of its first occurrence".

We enter:

```
inString("abcded", "d")
```

We get:

```
4
```

We enter:

```
inString("abcd", "e")
```

We get:

```
0
```

17.2 ASCII codes: ASC asc

`ASC` or `asc` returns the list of ASCII codes of the characters of the string.

We enter the " " thanks to the key ALPHA 0.

We enter in HOME:

```
ASC("A")
```

We get:

```
[65]
```

We enter in HOME:

```
ASC("ABC")
```

We get:

```
[65, 66, 67]
```

We enter in the CAS:

```
asc("A")
```

We get:

```
[65]
```

We enter in the CAS:

```
asc("ABC")
```

We get:

```
[65, 66, 67]
```

17.3 Character from ASCII code: `CHAR` `char`

`CHAR` or `char` returns the string corresponding to the characters having as ASCII code those of the argument.

We enter in HOME:

```
char(65)
```

or

```
char({65})
```

We get:

```
"A"
```

We enter:

```
char([65,66,67])
```

or

```
char({65,66,67})
```

We get:

```
"ABC"
```

We enter in the CAS:

```
char(65)
```

or

```
char([65])
```

or

```
char({65})
```

We get:

```
"A"
```

We enter:

```
char([65,66,67])
```

or

```
char({65,66,67})
```

We get:

```
"ABC"
```

17.3.1 Converting a real or an integer into a string: `string`

`string` evaluates its argument and converts it into a string of characters.

We enter:

```
string(1.32*10^20)
```

We get:

```
"1.23e+20"
```

We enter:

```
a:=1.32*10^4
string(a+a)
```

We get:

```
"26400"
```

17.4 Use a string as a number or a command: `expr`

17.4.1 Use a string as a number

`expr` allows to use a string of digits without leading zero as an integer written in basis 10, or a string of digits with a point as a decimal number written in basis 10.

`expr` returns this integer.

We enter:

```
expr("123")+1
```

We get:

```
124
```

We enter:

```
expr("45.67")+2.12
```

We get:

```
47.79
```

`expr` also allows to use a string of digits with no 8, nor 9, and with no leading zero as an integer written in basis 8.

We enter:

```
expr("0123")
```

We get:

```
83
```

Indeed, $1 * 8^2 + 2 * 8 + 3 = 83$

Note:

If we enter `expr("018")`, we get the decimal number 18.0.

`expr` allows to use a string containing digits and the letters `a, b, c, d, e, f`, and with the prefix `0x` as an integer written in basis 16.

We enter:

```
expr("0x12f")
```

We get:

```
303
```

Indeed, $1 * 16^2 + 2 * 16 + 15 = 303$

17.4.2 Use a string as a command name

`expr` allows to use a string of characters as a command.

`expr` is mostly useful in a program.

`expr` takes as argument a string of characters which can be interpreted as a command (or the name of a variable which stores a string or an expression returning a string).

`expr` transforms the string in an expression, then evaluates this expression:

to do an assignment, we should not write `expr("a"):=2`, but `expr("a:=2")` (see also `expr 17.4`)

We enter:

```
expr("c:=1")
```

We get:

```
The variable c stores 1
```

We enter:

```
a:="ifactor(54)";expr(a)
```

or:

```
expr("ifactor(54)")
```

We get:

```
2*3^3
```

17.5 Evaluate an expression in the form of a string: `string`

`string` evaluates an expression and returns its value in the form of a string of characters.

We can also use the concatenation of the expression with an empty string.

We enter:

```
string(ifactor(6))
```

Or we enter:

```
ifactor(6)+" "
```

Or we enter:

```
" "+ifactor(6)
```

We get:

```
"2*3"
```

We enter:

```
string(' (ifactor(6) '))
```

We get:

```
"ifactor(6) "
```

17.6 inString

`inString(l, c)` checks whether `c` is in the string `l` and returns the index of `c` or 0.

We enter:

```
inString"ABCDEF", "C"
```

We get:

```
3
```

We enter:

```
inString"ABCDEF", "G"
```

We get:

```
0
```

17.7 Left part of a string: left

`left(l, n)` returns the left part of length `n` of the string `l`.

We enter:

```
left("ABCDEF", 3)
```

We get:

```
"ABC"
```

17.8 Right part of a string: right

`right(l, n)` returns the right part of length `n` of the string `l`.

We enter:

```
right("ABCDEF", 2)
```

We get:

```
"EF"
```

17.9 Mid part of a string: `mid`

`mid(l, d, n)` returns the string extracted from the string l , starting by the character of index d , and length n (by default $n = \text{dim}(l) - d$).

We enter:

```
mid("ABCDEF", 2, 3)
```

We get:

```
"BCD"
```

We enter:

```
mid("ABCDEF", 2)
```

We get:

```
"BCDEF"
```

17.10 Rotate last character: `rotate`

`rotate` returns the string obtained by turning the last character first.

We enter:

```
rotate("ABC")
```

We get:

```
("CAB")
```

17.11 Length of a string: `dim` `DIM` `size` `SIZE` `length`

`DIM` (or `dim` or `size` or `SIZE` or `length`) returns the length of the string (or of the list).

We enter in HOME or in the CAS:

```
DIM("ABC")
```

We get:

```
3
```

We can also use `SIZE`

We enter:

```
SIZE("ABC")
```

We get:

```
3
```

Note

In HOME, `DIM` and `SIZE` are equivalent for matrices.

We enter in HOME:

```
DIM([[1,2,3],[4,5,6]])
```

We get:

```
{2,3}
```

We enter in HOME:

```
SIZE([[1,2,3],[4,5,6]])
```

We get:

```
{2,3}
```

In CAS, `dim` and `size` are not equivalent for matrices.

`dim` returns the list giving the dimension of a matrix whereas `size` returns the length of the list.

We enter in the CAS:

```
dim([[1,2,3],[4,5,6]])
```

We get:

```
[2,3]
```

We enter in the CAS:

```
size([[1,2,3],[4,5,6]])
```

We get:

```
2
```

17.12 Concatenate two strings: +

`+` concatenates two strings.

We enter:

```
"ABC"+"DEF"
```

We get:

```
"ABCDEF"
```

We can also use `CONCAT`

We enter:

```
CONCAT("ABC","DEF")
```

We get:

```
"ABCDEF"
```

17.13 Get the list or the string without its first element: `tail`

`tail(s)` returns the list or the string `s` without its first element.

We enter:

```
tail([0,1,2,3])
```

We get:

```
[1,2,3]
```

`l:=tail([0,1,2,3])` is equivalent to `l:=suppress([0,1,2,3],0)`

We enter:

```
tail("abcdef")
```

We get:

```
"bcdef"
```

`l:=tail("abcdef")` is equivalent to `l:=suppress("abcdef","a")`

17.14 First element of a list or of a string: `head`

`head(s)` returns the first element of the list `s` or the first character of the strings `s`.

We enter:

```
head([0,1,2,3])
```

We get:

```
0
```

We enter:

```
head("abcdef")
```

We get:

```
"a"
```

Chapter 18 Polynomials

18.1 Coefficients of a polynomial: POLYCOEF

POLYCOEF returns the coefficients of a polynomial knowing its roots.

We enter:

```
POLYCOEF([2,1])
```

or we enter:

```
POLYCOEF(2,1)
```

We get:

```
poly1[1,-3,2]
```

this represents the polynomial $X^2 - 3X + 2 = (X - 2) * (X - 1)$

We enter:

```
POLYCOEF([2,-1,3,-4])
```

We get:

```
poly1[1,0,-15,10,24]
```

this represents the polynomial $X^4 - 15X^2 + 10X + 24$

18.2 Polynomial from coefficients: POLYEVAL

POLYEVAL returns the symbolic writing of a polynomial supplied by the list of its coefficients or POLYEVAL evaluates at a point a polynomial supplied by the list of its coefficients.

We enter:

```
POLYEVAL({1,0,-15,10,24})
```

or we enter:

```
POLYEVAL([1,0,-15,10,24])
```

We get:

```
X^4-15*X^2+10*X+24
```

We enter:

```
POLYEVAL({1,0,-15,10,24},4)
```

or we enter:

```
POLYEVAL([1,0,-15,10,24],4)
```

We get:

80

because $4^4 - 15 * 4^2 + 10 * 4 + 24 = 80$

18.3 Expand a polynomial: POLYFORM

POLYFORM expands a polynomial supplied by an expression of one or several variables.

POLYFORM also permits to factorize a polynomial of one or several variables and do the decomposition into simple elements of a rational fraction.

We enter:

POLYFORM((X+2)^3+5)

or

POLYFORM((X+2)^3+5,X)

We get:

$X^3+6*X^2+12*X+13$

We enter:

POLYFORM((X+Y)^3+5)

or

POLYFORM((X+Y)^3+5,X,Y)

We get:

$X^3+3*X^2*Y+3*X*Y^2+Y^3+5$

We enter:

POLYFORM((X+Y)^3+5,Y,X)

We get:

$Y^3+3*Y^2*X+3*Y*X^2+X^3+5$

We enter:

POLYFORM((X+2)^2+5,X)

We get:

$X^2+4*X+8$

We enter:

POLYFORM((X+Y)^2+5)

We get:

$X^2+2*X*Y+Y^2+5$

We enter:

```
POLYFORM((X^2+2*X+1)*(X-1))
```

We get:

$$X^3+X^2-X-1$$

To factorize we enter:

```
POLYFORM(X^3+X^2-X-1,'*')
```

We get:

$$(X-1)*(X+1)^2$$

We enter:

```
POLYFORM(X^2-Y^2,'*')
```

We get:

$$(X-Y)*(X+Y)$$

We enter:

```
POLYFORM(1/(X^2-Y^2),'*')
```

We get:

$$1/((X-Y)*(X+Y))$$

To perform the decomposition into simple elements of a rational fraction, we enter:

```
POLYFORM(1/(1-X^2)^2,'+')
```

We get:

$$-1/4/(X-1)^2-1/4/(X-1)+1/4/(X+1)^2+1/4/(X+1)$$

We enter:

```
POLYFORM(1/(X^2-Y^2),'+')
```

We get:

$$1/(2*Y)/(X+Y)-1/(2*Y)/(X-Y)$$

Note:

We can also use `STO` to get a rewriting of an expression:

- `STO STO` to evaluate formally an expression,
- `STO +` to expand or to do a decomposition into simple elements
- `STO *` to factorize

We enter from Home:

```
X*SIN(X) STO STO
```

We get:

$$X*SIN(X)$$

We enter:

$$\partial (X * \sin(X), X) \text{ STO STO}$$

We get:

$$\sin(X) + X * \cos(X)$$

We enter:

$$\int (\sin(X), X) \text{ STO STO}$$

We get:

$$-\cos(X)$$

We enter:

$$(X+Y)^2 + 5 \text{ STO +}$$

We get:

$$X^2 + 2 * X * Y + Y^2 + 5$$

We enter:

$$1 / (1 - X^2)^2 \text{ STO +}$$

We get:

$$-1/4 / (X-1)^2 - 1/4 / (X-1) + 1/4 / (X+1)^2 + 1/4 / (X+1)$$

We enter:

$$X^3 + X^2 - X - 1 \text{ STO *)}$$

We get:

$$(X-1) * (X+1)^2$$

18.4 Roots of a polynomial from its coefficients: POLYROOT

POLYROOT returns the roots of a polynomial knowing its coefficients.

We enter:

$$\text{POLYROOT}(\{1, 0, -15, 10, 24\})$$

Or we enter:

$$\text{POLYROOT}([1, 0, -15, 10, 24])$$

We get:

$$[-1, 2, 3, -4]$$

which are the four roots of the polynomial $X^4 - 15X^2 + 10X + 24$

Chapter 19 Recurrent sequences

19.1 Values of a recurrent sequence or of a system of recurrent sequences:

`seqsolve`

See also `rsolve` 19.0.2.

`seqsolve` takes as argument the expression or the list of expressions which define(s) (one of) the relation(s) of recurrence, for example $f(x, n)$ if the recurrence relation is $u_{n+1} = f(u_n, n)$ (resp. $g(x, y, n)$ if the recurrence relation is $u_{n+2} = g(u_n, u_{n+1}, n) = g(x, y, n)$), the name of variables used (for example $[x, n]$ (resp. $[x, y, n]$)) and the start values of the sequences: for example a if $u_0 = a$ (resp. $[a, b]$ if $u_0 = a$ and $u_1 = b$).

The recurrence relation must include a linear homogenous part, the non homogenous part must be a linear combination of products of polynomial in n by a geometrical sequence in n . `seqsolve` then returns the value of the sequence on n .

Examples:

- Values of the sequence $u_0 = 3, u_{n+1} = 2u_n + n$

We enter:

```
seqsolve(2x+n, [x, n], 3)
```

We get:

$$-n-1+4*2^n$$

We can also press `rsolve(u(n+1)=2*u(n)+n, u(n), u(0)=3)` (cf19.0.2)

- Values of the sequence $u_0 = 3, u_{n+1} = 2u_n + n3^n$

We enter:

```
seqsolve(2x+n*3^n, [x, n], 3)
```

We get:

$$(n-3)*3^n+6*2^n$$

- Values of the sequence $u_0 = 0, u_1 = 1, u_{n+1} = a + u_{n-1}$ for $n > 0$.

We enter:

```
seqsolve(x+y, [x, y, n], [0, 1])
```

We get:

$$(5+\sqrt{5})/10*((\sqrt{5}+1)/2)^{(n-1)}+$$

$$(5-(\sqrt{5}))/10*((-\sqrt{5}+1)/2)^{(n-1)}$$

- Values of the sequence $u_0 = 0, u_1 = 1, u_{n+2} = 2 * u_{n+1} + u_n + n + 1$ for $n > 0$.
By hand, we find $u_2 = 3, u_3 = 9, u_4 = 24$, etc., ...

We enter:

```
seqsolve(x+2y+n+1, [x, y, n], [0, 1])
```

We get:

$$(-4^n - 3 \cdot (-\sqrt{2} - 1)^n \sqrt{2} + 2 \cdot (-\sqrt{2} - 1)^{n+3} + 3 \cdot (\sqrt{2} + 1)^n$$

Check that $n:=4$ returns 24

Or we enter because we have $u_{n+1} = 2u_n + v_n + n$ and $v_{n+1} = u_n$ (so $v_n = u_{n-1}$) with $u_0 = 0$ and $u_1 = 2u_0 + v_0 + 0 = 1$ then $v_0 = 1$:

$$\text{seqsolve}([2x+y+n, x], [x, y, n], [0, 1])$$

We get:

$$\left[\frac{(-1)/2 - (-2 - 3\sqrt{2})/8 \cdot (\sqrt{2} + 1)^n - (-2 + 3\sqrt{2})/8 \cdot (-\sqrt{2} + 1)^{n-1}/2^n, -(-4 + \sqrt{2})/8 \cdot (\sqrt{2} + 1)^n - (-4 - \sqrt{2})/8 \cdot (-\sqrt{2} + 1)^{n-1}/2^n \right]$$

Check that $n:=4$ returns 24

- Values of the sequence $u_0 = 0, v_0 = 1, u_{n+1} = u_n + 2v_n, v_{n+1} = u_n + n + 1$ for $n > 0$.

We enter:

$$\text{seqsolve}([x+2*y, n+1+x], [x, y, n], [0, 1])$$

We get:

$$\left[\frac{(-2^n - (-1)^{n+2} \cdot 2^{n-4} - 3)/2, (-1)^{n+2} \cdot 2^{n-1}/2} \right]$$

- Values of the sequence $u_0 = 0, v_0 = 1, u_{n+1} = u_n + 2v_n + n + 1, v_{n+1} = u_n$ for $n > 0$.

We enter:

$$\text{seqsolve}([x+2*y+n+1, x], [x, y, n], [0, 1])$$

We get:

$$\left[\frac{(-2^n - (-1)^{n+3} + 2^{n-8} - 5)/4, (-2^n + (-1)^{n+3} + 2^{n-4} - 3)/4} \right]$$

- Values of the sequence $u_0 = 0, v_0 = 1, u_{n+1} = u_n + v_n, v_{n+1} = u_n - v_n$ for $n > 0$.

We enter:

$$\text{seqsolve}([x+y, x-y], [x, y, n], [0, 1])$$

We get:

$$\left[\begin{aligned} & (-4^n - 3 \cdot (-\sqrt{2} - 1)^n \sqrt{2} + \\ & 2 \cdot (-\sqrt{2} - 1)^{n+3} + 3 \cdot (\sqrt{2} + 1)^n \sqrt{2} + \\ & 2 \cdot (\sqrt{2} + 1)^{n-4} / 8, \quad (-4^n + (-\sqrt{2} - 1)^n \sqrt{2} + \\ & 4 \cdot (-\sqrt{2} - 1)^n - (\sqrt{2} + 1)^n \sqrt{2} + \\ & 4 \cdot (\sqrt{2} + 1)^n / 8 \end{aligned} \right]$$

- Values of the sequence $u_0 = 2, v_0 = 0, u_{n+1} = 4 \cdot v_n + n + 1, v_{n+1} = u_n$, for $n > 0$.

We enter:

$$\text{seqsolve}([4y+n+1, x], [x, y, n], [2, 0])$$

We get:

$$\begin{aligned} & [(-8)/9 + 2 \cdot 2^{n-1} - (-8)/9 \cdot (-1)^{n-1} \cdot 2^{n-1} / 3^n, \\ & (-5)/9 + 2^{n-4} / 9 \cdot (-1)^{n-1} \cdot 2^{n-1} / 3^n] \end{aligned}$$

19.2 Values of a recurrent sequence or of a system of recurrent sequences:

`rsolve`

See also `seqsolve` 19.1.

`rsolve` takes as argument the relation(s) of recurrence, the name of the variables used and the start value of the sequence.

The recurrence relation is:

- either a linear homogenous part, the non homogenous part must be a linear combination of products of polynomial in n by a geometrical sequence in n . For example, $u_{n+1} = 2u_n + n3^n$
- either an homographic function. For example, $u_{n+1} = \frac{u_{n-1}}{u_n - 2}$

`rsolve` then returns a matrix whose rows are the values of the sequence in n .

Notes

Unlike `seqsolve`, `rsolve` is more flexible because with `rsolve`:

- starting the sequence by $u(0)$ is not compulsory,
- we can give several start values, for example $u(0)^2=1$, that is why `rsolve` returns a list,
- we write the recurrence relation as in mathematics.

Examples:

- Values of the sequence $u_0 = 3, u_{n+1} = 2u_n + n$

We enter:

$$\text{rsolve}(u(n+1)=2u(n)+n, u(n), u(0)=3)$$

We get:

$$[-1+4 \cdot 2^{(n+1-1)} - n]$$

- Values of the sequence $u_1^2 = 1, u_{n+1} = 2u_n + n$

We enter:

$$\text{rsolve}(u(n+1)=2u(n)+n, u(n), u(1)^2=1)$$

We get:

$$[[-1 - (-3)/2 \cdot 2^{(n+1-1)} - n, -1 - (-1)/2 \cdot 2^{(n+1-1)} - n]]$$

- Values of the sequence $u_0 = 3, u_{n+1} = 2u_n + n3^n$

We enter:

$$\text{rsolve}(u(n+1)=2u(n) + (n) \cdot 3^n, u(n), u(0)=3)$$

We get:

$$[-3 \cdot 3^{(n+1-1)} + 6 \cdot 2^{(n+1-1)} + n \cdot 3^{(n+1-1)}]$$

- Values of the sequence $u_0 = 4, u_{n+1} = \frac{u_{n-1}}{u_n - 2}$

We enter:

$$\text{rsolve}(u(n+1)=(u(n)-1)/(u(n)-2), u(n), u(0)=4)$$

We get:

$$\left[\left((10\sqrt{5}+30) \cdot \left(\frac{\sqrt{5}-3}{2} \right)^{n+30} \sqrt{5}-70 \right) / \right. \\ \left. (20 \cdot \left(\frac{\sqrt{5}-3}{2} \right)^{n+10} \sqrt{5}-30) \right]$$

- Values of the sequence $u_0 = 0, u_1 = 1, u_{n+1} = a + u_{n-1}$ for $n > 0$.

We enter:

$$\text{rsolve}(u(n+1)=u(n)+u(n-1), u(n), u(0)=0, u(1)=1)$$

We get:

$$\left[\frac{5+\sqrt{5}}{10} \cdot \left(\frac{\sqrt{5}+1}{2} \right)^{n+1-1-1} + \right. \\ \left. \frac{5-\sqrt{5}}{10} \cdot \left(\frac{-\sqrt{5}+1}{2} \right)^{n+1-1-1} \right]$$

- Values of the sequence $u_0 = 0, u_1 = 1, u_{n+1} = 2 \cdot a + u_{n-1} + n$ for $n > 0$.

We enter:

$$\text{rsolve}(u(n+1)=2 \cdot u(n)+u(n-1)+n, u(n), u(0)=0, u(1)=1)$$

We get:

$$\left[\frac{-1}{2} - \frac{-2-3\sqrt{2}}{8} (\sqrt{2}+1)^{n+1-1} - \frac{-2+3\sqrt{2}}{8} (\sqrt{2}+1)^{n+1-1} - \frac{1}{2}n \right]$$

Or we enter:

$$\text{rsolve}([u(n+1)=2 \cdot u(n)+v(n)+n, v(n+1)=u(n)], [u(n), v(n)], u(0)=0, v(0)=1)$$

We get:

$$\left[\left[\frac{-1}{2} - \frac{-2-3\sqrt{2}}{8} (\sqrt{2}+1)^{n+1-1} - \frac{-2+3\sqrt{2}}{8} (\sqrt{2}+1)^{n+1-1} - \frac{1}{2}n, \right. \right. \\ \left. \left. - \frac{-4+\sqrt{2}}{8} (\sqrt{2}+1)^{n+1-1} - \frac{-4-\sqrt{2}}{8} (-\sqrt{2}+1)^{n+1-1} - \frac{1}{2}n \right] \right]$$

- Values of the sequence $u_0 = 0, v_0 = 1, u_{n+1} = a + v_n, v_{n+1} = a - v_n$.

We enter:

$$\text{rsolve}([u(n+1)=u(n)+v(n), v(n+1)=u(n)-v(n)], [u(n), v(n)], [u(0)=0, v(0)=1])$$

We get:

$$\left[\left[\frac{1}{2} \cdot 2^{\frac{(n-1)}{2}} + \frac{1}{2} \cdot (-\sqrt{2})^{n-1}, \frac{-1+\sqrt{2}}{2} \cdot 2^{\frac{(n-1)}{2}} + \right. \right. \\ \left. \left. \frac{-1-\sqrt{2}}{2} \cdot (-\sqrt{2})^{n-1} \right] \right]$$

- Values of the sequence $u_0 = 2, v_0 = 0, u_{n+1} = 4 \cdot v_n + n + 1, v_{n+1} = u_n$.

We enter:

$$\text{rsolve}([u(n+1)=4 \cdot v(n)+n+1, v(n+1)=u(n)], [u(n), v(n)], [u(0)=2, v(0)=0])$$

We get:

$$\left[\left(\frac{-8}{9} + 2 \cdot 2^{(n+1)-1} \right) - \left(\frac{-8}{9} \cdot (-1)^{(n+1)-1} \cdot 2^{(n+1)-1} \right) - \frac{1}{3} \cdot n, \right. \\ \left. \left(\frac{-5}{9} + 2^{(n+1)-1} \right) - \frac{4}{9} \cdot (-1)^{(n+1)-1} \cdot 2^{(n+1)-1} - \frac{1}{3} \cdot n \right]$$

Chapter 20 Matrices

20.1 Generalities

To write a matrix, we put between two brackets a series of row vectors, for example: `[[1,2],[3,4]]`.
 The numerical matrices are stored in variables `M0, M1...M9`.
 The index of rows and columns of a matrix start at 1, and we put the index between two brackets or parentheses.

20.2 Definition

To define the matrix `M1` which equals `[[1,2],[3,4]]`, we enter:
`[[1,2],[3,4]]=>M1` or we use the matrix editor (Shift 4 (Matrix))
 We get:

```
[[1,2],[3,4]]
```

To define the matrix `M2` which equals `[[1,2],[3,4],[5,6]]`, we enter:
`[[1,2],[3,4],[5,6]]=>M2` or we use the matrix editor (Shift 4 (Matrix))
 We get:

```
[[1,2],[3,4],[5,6]]
```

To get the element 3 of `M2`, located at the beginning of the second row: this will be the element on row of index 2 and on column of index 1 if we designate it by `M2[2,1]` or `M2(2,1)`.
 We enter:

```
M1[2,1]
```

We get:

```
3
```

We enter:

```
M1(2,1)
```

We get:

```
3
```

20.2.1 Dimension of a matrix: `dim`

`dim` takes as argument a matrix `A`.
`dim` returns the dimension of the matrix `A` in the form of a list formed by its number of rows and number of columns.

We enter:

```
dim([[1,2,3],[3,4,5]])
```

We get:

```
[2,3]
```

20.2.2 Number of rows: `rowDim`

`rowDim` takes as argument a matrix A .
`rowDim` returns the number of rows of the matrix A .

We enter:

```
rowDim([[1,2,3],[3,4,5]])
```

We get:

2

20.2.3 Number of columns: `colDim`

`colDim` takes as argument a matrix A .
`colDim` returns the number of columns of the matrix A .

We enter:

```
colDim([[1,2,3],[3,4,5]])
```

We get:

3

20.3 Operations on rows and columns useful in programming

20.3.1 Add a column to a matrix: `ADDCOL`

`ADDCOL(M1, col, n)` adds the column `col` which will be the column of index `n` of the matrix $M1$.

We enter:

```
[[1,2],[3,4]]=>M1
ADDCOL(M1,[5,6],1)
```

We get the new matrix $M1$:

```
[[5,1,2],[6,3,4]]
```

We enter:

M1

We get the new matrix $M1$:

```
[[5,1,2],[6,3,4]]
```

To add a last column (which, in this case, will be the column of index 3) to the matrix $M1$, we enter:

```
[[1,2],[3,4]]=>M1
ADDCOL(M1,[5,6],3)
```

We get the new matrix $M1$:

```
[[1,2,5],[3,4,6]]
```

We enter:

M1

We get the new matrix M1:

$[[1, 2, 5], [3, 4, 6]]$

20.3.2 Swap rows: SWAPROW rowSwap

SWAPROW or rowSwap has three arguments: a matrix and two integers $n1$ and $n2$.

SWAPROW or rowSwap returns the matrix obtained by swapping the rows $n1$ and $n2$ in the matrix given as argument.

We enter in HOME (the index starts at 1):

`SWAPROW ([[1, 2], [3, 4]], 1, 2)`

We get:

$[[3, 4], [1, 2]]$

We enter in the CAS (the index also starts at 1):

`SWAPROW ([[1, 2], [3, 4]], 1, 2)`

or

`rowSwap ([[1, 2], [3, 4]], 1, 2)`

We get:

$[[3, 4], [1, 2]]$

20.3.3 Swap columns: SWAPCOL colSwap

SWAPCOL or colSwap has three arguments: a matrix and two integers $n1$ and $n2$.

SWAPCOL or colSwap returns the matrix obtained by swapping the columns $n1$ and $n2$ in the matrix given as argument.

We enter in HOME or in the CAS (the index starts at 1):

`SWAPCOL ([[1, 2], [3, 4]], 1, 2)`

We get:

$[[2, 1], [4, 3]]$

We enter in the CAS

`SWAPCOL ([[1, 2], [3, 4], [5, 6]], 1, 2)`

or

`colSwap ([[1, 2], [3, 4], [5, 6]], 1, 2)`

We get:

$[[2, 1], [4, 3], [6, 5]]$

20.3.4 Extract rows from a matrix: `row`

`row` allows to extract one or several rows from a matrix.

`row` has two arguments: a matrix and an integer n or an interval $n1..n2$.

`row` returns the row of index n of the matrix supplied as argument, or the sequence of rows of index starting from $n1$ to $n2$ of this matrix.

We enter in HOME or in the CAS (the index starts at 1):

```
row([[1,2,3],[4,5,6],[7,8,9]],1)
```

We get:

```
[1,2,3]
```

We enter:

```
row([[1,2,3],[4,5,6],[7,8,9]],1..2)
```

We get:

```
[[1,2,3],[4,5,6]]
```

20.3.5 Extract columns from a matrix: `col`

`col` allows to extract one or several columns from a matrix.

`col` has two arguments: a matrix, and an integer n or an interval $n1..n2$.

`col` returns the column of index n of the matrix supplied as argument, or the sequence of columns of index starting from $n1$ to $n2$ of this matrix.

We enter in HOME or danc CAS (the index starts at 1):

```
col([[1,2,3],[4,5,6],[7,8,9]],1)
```

We get:

```
[1,4,7]
```

We enter:

```
col([[1,2,3],[4,5,6],[7,8,9]],1..2)
```

We get:

```
([1,4,7],[2,5,8])
```

20.3.6 Remove columns from a matrix: `DELCOL delcols`

In HOME, `DELCOL(M1,n)` removes the column of index n from the matrix $M1$.

In CAS, `delcols` has two arguments: a matrix A , and an integer n or an interval $n1..n2$.

`delcols` returns the matrix obtained by removing the column n or the columns $n1$ up to $n2$ from the matrix A .

We enter:

```
[[1,2,5],[3,4,6]]=>M1
```

```
DELCOL(M1,2)
```

or

```
DELCOL (M1, 2..2)
```

We get:

```
[[1, 5], [3, 6]]
```

To remove the columns of index 2 and 3 from the matrix $M1$, we enter:

```
[[1, 2, 5], [3, 4, 6]]=>M2
```

```
DELCOL (M1, 2..3)
```

We get:

```
[[1], [3]]
```

We enter:

```
delcols ([[1, 2, 3], [4, 5, 6], [7, 8, 9]], 2)
```

We get:

```
[[1, 3], [4, 6], [7, 9]]
```

We enter:

```
delcols ([[1, 2, 3], [4, 5, 6], [7, 8, 9]], 1..2)
```

We get:

```
[[3], [6], [9]]
```

20.3.7 Remove rows from a matrix: DELROW delrows

In HOME, `DELROW (M1, n)` removes the row of index n from the matrix $M1$

In CAS, `delrows` has two arguments: a matrix A , and an integer n or an interval $n1..n2$.

`delrows` returns the matrix obtained by removing the row n or the rows $n1$ up to $n2$ from the matrix A .

We enter:

```
[[1, 2], [3, 4], [5, 6]]=>M1
```

```
DELROW (M1, 2)
```

or

```
DELROW (M1, 2..2)
```

We get the new matrix $M1$:

```
[[1, 2], [5, 6]]
```

To remove the rows of index 2 and 3 from the matrix $M1$, we enter:

```
[[1, 2], [3, 4], [5, 6]]=>M1
```

```
DELROW (M1, 2..3)
```

We get the new matrix $M1$:

```
[[1, 2]]
```

We enter:

```
delrows([[1,2,3],[4,5,6],[7,8,9]],2)
```

We get:

```
[[1,2,3],[7,8,9]]
```

We enter:

```
delrows([[1,2,3],[4,5,6],[7,8,9]],1..2)
```

We get:

```
[[7,8,9]]
```

20.3.8 Extract a sub-matrix from a matrix: SUB subMat

In HOME, SUB has three arguments: a matrix $M1$, and two lists of indexes $\{nl1, nc1\}, \{nl2, nc2\}$.

Warning! Indices starts at 1.

These indices are:

$nl1$ index of the beginning of row,
 $nc1$ index of the beginning of column,
 $nl2$ index of end of row,
 $nc2$ index of end of column.

SUB($M1, \{nl1, nc1\}, \{nl2, nc2\}$) extracts the sub-matrix from the matrix A of first element $A[nl1, nc1]$ and last element $A[nl2, nc2]$.

We enter:

```
SUB([[3,4,5],[1,2,6]],{1,2},{2,3})
```

We get:

```
[[4,5],[2,6]]
```

In CAS, subMat has three arguments: a matrix A , and two lists of indexes $[nl1, nc1], [nl2, nc2]$ or $\{nl1, nc1\}, \{nl2, nc2\}$.

Warning! Indices also start at 1.

These indices are:

$nl1$ index the beginning of row,
 $nc1$ index of the beginning of column,
 $nl2$ index of end of row
 $nc2$ index of end of column.

subMat($A, nl1, nc1, nl2, nc2$) extracts the sub-matrix from the matrix A of first element $A[nl1, nc1]$ and last element $A[nl2, nc2]$.

To define the matrix A , we enter:

```
A:=[[1,2,3],[4,5,6],[7,8,9]]
```

We enter:

```
subMat(A,[1,2],[2,3])
```

We get:

```
[[2,3],[5,6]]
```

20.3.9 Redimension a matrix or a vector: REDIM

REDIM takes as argument a matrix A (resp. a vector) and a list of two integers (resp. one integer). REDIM redimension this matrix (resp. this vector) either by reducing it, either by filling it with 0.

We enter:

```
REDIM([[4,1,-2],[1,2,-1]],[3,4])
```

We get:

```
[[4,1,-2,0],[1,2,-1,0],[0,0,0,0]]
```

We enter:

```
REDIM([[4,1,-2],[1,2,-1],[2,1,0]],[2,1])
```

We get:

```
[[4],[1]]
```

We enter:

```
REDIM([4,1,-2,1,2,-1],8)
```

We get:

```
[4,1,-2,1,2,-1,0,0]
```

We enter:

```
REDIM([4,1,-2,1,2,-1],3)
```

We get:

```
[4,1,-2]
```

20.3.10 Replace a portion of a matrix or of a vector: REPLACE

REPLACE takes as argument a matrix A (resp. a vector) and a list of two indices (resp. one integer) and the matrix (resp. the vector) which must be replaced starting from these two indices.

REPLACE does this replacement by eventually reducing the matrix (resp. the vector) if it is oversized.

We enter in HOME:

```
REPLACE([[1,2,3],[4,5,6]},{1,1},[[5,6],[7,8]])
```

Or we enter in the CAS:

```
REPLACE([[1,2,3],[4,5,6]],[1,1],[[5,6],[7,8]])
```

We get:

```
[[5,6,3],[7,8,6]]
```

We enter in HOME:

```
REPLACE([[1,2,3],[4,5,6]},{1,2},[[7,8],[9,0]])
```

Or we enter in the CAS:

```
REPLACE ([[1, 2, 3], [4, 5, 6]], [1, 2], [[7, 8, 10], [9, 0, 11]])
```

We get:

```
[[1, 7, 8], [4, 9, 0]]
```

We enter in HOME or in the CAS:

```
REPLACE ([1, 2, 3, 4], 2, [5, 6])
```

We get:

```
[1, 5, 6, 4]
```

We enter in HOME or in the CAS:

```
REPLACE ([1, 2, 3, 4], 2, [5, 6, 7, 8])
```

We get:

```
[1, 5, 6, 7]
```

20.3.11 Add a row to a matrix: `ADDROW`

`ADDROW(M1, row, n)` adds the row `row` which will be the row of index `n` to the matrix `M1`.

We enter:

```
[[1, 2], [3, 4]]=>M1
```

```
ADDROW(M1, [5, 6], 1)
```

We get the new matrix `M1`:

```
[[5, 6], [1, 2], [3, 4]]
```

To add a last row (which will be here the row of index 3) to the matrix `M1`, we enter:

```
[[1, 2], [3, 4]]=>M1
```

```
ADDROW(M1, [5, 6], 3)
```

We get the new matrix `M1`:

```
[[1, 2], [3, 4], [5, 6]]
```

20.3.12 Add a row to another: `rowAdd`

In CAS, `rowAdd` has three arguments: a matrix `A` and two integers `n1` and `n2`.

`rowAdd` returns the matrix obtained by replacing in `A` the row `n2` by the sum of rows `n1` and `n2`.

We enter:

```
rowAdd ([[1, 2], [3, 4]], 1, 2)
```

We get:

```
[[1, 2], [4, 6]]
```

20.3.13 Multiply a row by an expression: SCALE mRow

Warning! SCALE and mRow do not have their arguments listed in the same order.

SCALE has three arguments: a matrix A , an expression and an integer n .

mRow has three arguments: an expression, a matrix A and an integer n .

SCALE or mRow returns the matrix obtained by replacing in A the row n by the multiplication of the row n by the expression.

We enter:

```
SCALE ([[1, 2], [3, 4]], 12, 2)
```

Or we enter:

```
mRow(12, [[1, 2], [3, 4]], 2)
```

We get:

```
[[1, 2], [36, 48]]
```

20.3.14 Add k times a row to another: SCALEADD mRowAdd

Warning! SCALEADD and mRowAdd do not have their arguments listed in the same order.

SCALEADD has four arguments: a matrix A , a real k and two integers $n1$ and $n2$.

mRowAdd has four arguments: a real k , a matrix A and two integers $n1$ and $n2$.

mRowAdd returns the matrix obtained by replacing in A the row $n2$ by the sum of the row $n2$ and k times the row $n1$.

We enter:

```
SCALEADD ([[5, 7], [3, 4], [1, 2]], 1.1, 2, 3)
```

Or we enter:

```
mRowAdd(1.1, [[5, 7], [3, 4], [1, 2]], 2, 3)
```

We get:

```
[[5, 7], [3, 4], [4.3, 6.4]]
```

20.4 Creation and arithmetic of matrices

20.4.1 Addition and subtraction of matrices: + - .+ .-

The addition (resp. the subtraction) of matrices is done thanks to the infix operator + or .+ (resp. - or .-).

We enter:

```
[[1, 2], [3, 4]] + [[5, 6], [7, 8]]
```

We get:

```
[[6, 8], [10, 12]]
```

We enter:

```
[[1, 2], [3, 4]] - [[5, 6], [7, 8]]
```

We get:

```
[[ -4, -4], [-4, -4]]
```

Note:

+ can also be prefix, in this case it must be quoted.

We enter:

```
'+' ([[1, 2], [3, 4]], [[5, 6], [7, 8]], [[2, 2], [3, 3]])
```

We get:

```
[[8, 10], [13, 15]]
```

20.4.2 Multiplication of matrices: * &*

The multiplication of matrices is done thanks to the infix operator * (or &*).

We enter:

```
[[1, 2], [3, 4]] * [[5, 6], [7, 8]]
```

Or we enter:

```
[[1, 2], [3, 4]] &* [[5, 6], [7, 8]]
```

We get:

```
[[19, 22], [43, 50]]
```

20.4.3 Raising a matrix to an integer power: ^ &^

The raising of a matrix to a power is done thanks to the infix operator ^ (or &^).

We enter:

```
[[1, 2], [3, 4]] ^ 5
```

Or we enter:

```
[[1, 2], [3, 4]] &^ 5
```

We get:

```
[[1069, 1558], [2337, 3406]]
```

We enter:

```
normal ([[1, 2], [3, 4]] ^ n)
```

Or we enter:

```
normal ([[1, 2], [3, 4]] &^ n)
```

We get:

```
[[ (11-sqrt(33))/22*((sqrt(33)+5)/2)^n+
(11+sqrt(33))/22*((-sqrt(33)+5)/2)^n, (2
```

20.4.4 Hadamard product (infix version): `.*`

See also 16.12.6 and ??.

`.*` takes as arguments two matrices or two lists A and B of same order.

`.*` is a infix operator which returns the matrix or the list constituted by the product terme at term of elements of A and B .

We enter:

```
[[1, 2], [3,4]] .* [[5, 6], [7, 8]]
```

We get:

```
[[5,12], [21,32]]
```

20.4.5 Hadamard division (infix version): `./`

`./` takes as arguments of matrices or two lists A and B of same degree.

`./` is a infix operator which returns the matrix or the list constituted by the division term by term of elements of A and B .

We enter:

```
[[1, 2], [3,4]] ./ [[5, 6], [7, 8]]
```

We get:

```
[[1/5,1/3], [3/7,1/2]]
```

20.4.6 Hadamard power (infix version): `.^`

`.^` takes as arguments a matrix A and a real number b .

`.^` is an infix operator which returns the matrix constituted by each element of A rised to the power b .

We enter:

```
[[1, 2], [3,4]] .^ 2
```

We get:

```
[[1, 4], [9,16]]
```

20.5 Transpose matrix: `transpose`

In CAS, `transpose` returns the transpose matrix of the matrix supplied as argument.

We enter:

```
transpose([[i,2], [4,5-i]])
```

We get:

```
[[i,4], [2,5-i]]
```

20.6 Conjugate transpose matrix: `TRN` `trn`

In CAS, `TRN` returns the transpose of the conjugatee of the matrix supplied as argument.

We enter:

```
TRN([[i, 2], [4, 5-i]])
```

Or we enter:

```
trn([[i, 2], [4, 5-i]])
```

We get:

```
[-i, 4], [2, 5+i]
```

20.7 Determinant: DET det

In HOME, `DET` returns the determinant of a square matrix.

We enter:

```
DET([[1/2, 2, 4], [4, 5, 6], [7, 8, 9]])
```

We get:

```
-1.5
```

In CAS, `DET` or `det` returns the determinant of a square matrix.

We enter:

```
DET([[1/2, 2, 4], [4, 5, 6], [7, 8, 9]])
```

Or we enter:

```
det([[1, 2, 4], [4, 5, 6], [7, 8, 9]])
```

We get:

```
-3/2
```

20.7.1 Characteristic polynomial: charpoly

`charpoly` has one (resp. two) argument(s).

`charpoly` takes as argument a matrix A of order n (resp. a matrix A of order n and a name of formal variable).

`charpoly` returns the characteristic polynomial P of A written as a list of its coefficients (resp. the characteristic polynomial P of A written in symbolic form by using the name of variable supplied as argument).

The characteristic polynomial P of A is defined by

$$P(x) = \det(x.I - A)$$

We enter:

```
charpoly([[4, 1, -2], [1, 2, -1], [2, 1, 0]])
```

We get:

```
[1, -6, 12, -8]
```

So the characteristic polynomial of $[[4, 1, -2], [1, 2, -1], [2, 1, 0]]$ is

$$x^3 - 6x^2 + 12x - 8$$

We can also get the symbolic form by entering:

```
normal(poly2symb([1,-6,12,-8]))
```

We enter:

```
purge(x); charpoly([[4,1,-2],[1,2,-1],[2,1,0]],x)
```

We get:

$$x^3 - 6x^2 + 12x - 8$$

We can specify by an optional argument the algorithm used to do this calculations, among the following:

- `lagrange`: calculation by Lagrange interpolation, by giving to x the values between 0 and the dimension.

We enter:

```
pcar([[4,1,-2],[1,2,-1],[2,1,0]],lagrange)
```

We get:

$$x * ((x-1) * (-x+5) - 7) + 8$$

and after simplification:

$$-x^3 + 6x^2 - 12x + 8$$

- `hessenberg`: calculation by tridiagonal reduction, then recurrence formula, efficient on a finite field.

We enter:

```
pcar([[4,1,-2],[1,2,-1],[2,1,0]],hessenberg)
```

We get:

$$[1, -6, 12, -8]$$

- `fadeev`: simultaneous calculation of the characteristic polynomial and the comatrix of $xI - A$

We enter:

```
pcar([[4,1,-2],[1,2,-1],[2,1,0]],fadeev)
```

We get:

$$[1, -6, 12, -8]$$

- `pmin`: calculation of the minimal polynomial relative to a randomly chosen vector. It is the characteristic polynomial if it is of maximal degree.

We enter:

```
pcar([[4,1,-2],[1,2,-1],[2,1,0]],pmin)
```

We get:

$$[1, -6, 12, -8]$$

For matrices with integer coefficients, the algorithm used by default is modular, we calculate the characteristic polynomial modulus several prime numbers, either by the minimal polynomial, either by Hessenberg, and we rebuild by the chinese remainders coefficient by coefficient. The stop condition is probabilistic, when the rebuilt polynomial does not vary anymore for prime numbers whose product is greater than the inverse of the value of `proba_epsilon` (which can be tuned in the CAS configuration). If `proba_epsilon` is null, the result is determinist (an increase a priori of the coefficients is then used). In all cases, the calculation time is of the range $O(n^4 \ln(n))$, but it is quicker with the probabilistic method.

20.8 Vectorial field and linear applications

20.8.1 Basis of a vectorial subspace: `basis`

`basis` takes as argument the list of components of the vectors which generate a vectorial subspace of \mathbb{R}^n .

`basis` returns a list constituted of vectors of a basis of this vectorial subspace.

We enter:

```
basis([[1,2,3],[1,1,1],[2,3,4]])
```

We get:

```
[[1,0,-1],[0,1,2]]
```

20.8.2 Intersection basis of two vectorial subspaces: `ibasis`

`ibasis` takes as argument two lists of vectors generating two vectorial subspaces of \mathbb{R}^n .

`ibasis` returns a list constituted of vectors forming a basis of the intersection of these vectorial subspaces.

We enter:

```
ibasis([[1,2]],[[2,4]])
```

We get:

```
[[1,2]]
```

20.8.3 Image of a linear application: `image`

`image` takes as argument the matrix of a linear application f in the canonical basis.

`image` returns a list of vectors forming a basis of the image of f .

We enter:

```
image([[1,1,2],[2,1,3],[3,1,4]])
```

We get:

```
[[ -1, 0, 1], [ 0, -1, -2]]
```

20.8.4 Kernel of a linear application: `ker`

`ker` takes as argument the matrix of a linear application f in the canonical basis.

`ker` returns a list of vectors forming a basis of the kernel of f .

We enter:

```
ker([[1,1,2],[2,1,3],[3,1,4]])
```

We get:

```
[[1,1,-1]]
```

The kernel is then generated by the vector $[1, 1, -1]$.

20.9 Solve a linear system: RREF `rref`

In HOME, `RREF` allows to solve a linear system of matrix $M1$ and second member $M3$.

`ADDCOL(M1,M3)=>M2` and `RREF(M2)` returns the reduced row-echelon form of $M2$.

For instance, we want to solve the system: $\{3x + y = 2, 3x + 2y = -2\}$ with respect to x, y .

We enter:

```
RREF([[3,1,2],[3,2,-2]])
```

We get:

```
[[1,0,2],[0,1,-4]]
```

and we deduce that $x = 2$ and $y = -4$.

We want to solve the system:

$\{x + y - z = 5, 2x - y = 7, x - 2y + z = 2\}$ with respect to x, y, z .

We enter:

```
RREF([[1,1,-1,5],[2,-1,0,7],[1,-2,1,2]])
```

We get:

```
[[1,0,-0.33333333333333,4],[0,1,-0.66666666666667,1],[0,0,0,0]]
```

and we deduce that $x = 4 + z/3$, $y = 1 + 2z/3$ and $z = z$.

In CAS, `rref` or `RREF` returns the reduced row-echelon form of the matrix supplied as argument.

For instance, we want solve the system: $\{3x + y = 2, 3x + 2y = -2\}$ with respect to x, y .

We enter:

```
rref([[3,1,2],[3,2,-2]])
```

Or we enter:

```
RREF([[3,1,2],[3,2,-2]])
```

We get:

```
[[1,0,2],[0,1,-4]]
```

and we deduce that $x = 2$ and $y = -4$.

We want to solve the system:

$\{x + y - z = 5, 2x - y = 7, x - 2y + z = 2\}$ with respect to x, y, z .

We enter:

```
rref([[1, 1, -1, 5], [2, -1, 0, 7], [1, -2, 1, 2]])
```

Or we enter:

```
RREF([[1, 1, -1, 5], [2, -1, 0, 7], [1, -2, 1, 2]])
```

We get:

```
[[1, 0, -1/3, 4], [0, 1, -2/3, 1], [0, 0, 0, 0]]
```

and we deduce that $x = 4 + z/3$, $y = 1 + 2z/3$ and $z = z$.

20.9.1 Solve of $A * X = B$: `simult`

`simult` allows to solve a system of linear equations (resp. several systems of linear equations which differ by their second member only).

We write the system(s) in matrix form (see also 6.10.17):

$$A * X = b \quad (\text{resp. } A * X = B)$$

The parameters of `simult` are the matrix A of the system and the column vector b (i.e. a matrix of one column) formed by the second member of the system to be solved (resp. the matrix B whose columns are the vectors b of the second members of the systems to be solved).

The result is a column vector solution of the system (resp. a matrix whose columns are the solutions of the different systems).

For instance, be the following system to be solved:

$$\begin{cases} 3x + y = -2 \\ 3x + 2y = 2 \end{cases}$$

We enter:

```
simult([[3, 1], [3, 2]], [[-2], [2]])
```

We get:

```
[[ -2], [ 4]]
```

so this means:

$x = -2$ and $y = 4$ are solutions of the system.

We enter:

```
simult([[3, 1], [3, 2]], [[-2, 1], [2, 2]])
```

We get:

```
[[ -2, 0], [ 4, 1]]
```

so this means that:

$x = -2$ and $y = 4$ are solutions of the system

$$\begin{cases} 3x + y = -2 \\ 3x + 2y = 2 \end{cases}$$

and that $x = 0$ and $y = 1$ are solutions of the system

$$\begin{cases} 3x + y = 1 \\ 3x + 2y = 2 \end{cases}$$

20.10 Make matrices

20.10.1 Make a matrix from an expression: `MAKEMAT` `makemat`

In HOME, `MAKEMAT(Expr(I,J),n,p)` creates a matrix from an expression according to the variables I and J .

I represents the index of rows and J represents the index of columns and the index I goes from 1 to n and the index J goes from 1 to p .

`MAKEMAT(Expr(I,J),n,p)` returns the matrix $M_{I,J} = Expr(I,J)$ for $I = 1..n$ and $J = 1..p$.

We enter:

```
MAKEMAT(I*J,2,3)
```

We get:

```
[[2,3,4],[3,4,5]]
```

In CAS, we can use `MAKEMAT` and `makemat`. `makemat` has a function as first argument: the first variable, the index of the row and the second variable, the index of the column. The second argument represents the number of rows and the third argument represents the number of columns.

Warning! The indices also start at 1.

We enter:

```
MAKEMAT((I+J),2,3)
```

Or we enter:

```
makemat((j,k)->(j+k),2,3)
```

We get:

```
[[0,1,2],[1,3,5]]
```

20.10.2 Matrix of zeros: `matrix`

`matrix(n,p)` returns the matrix of n rows and p columns formed by zeros.

We enter:

```
matrix(4,3)
```

We get:

```
[[0,0,0],[0,0,0],[0,0,0],[0,0,0]]
```

20.10.3 Matrix identity: `IDENMAT` `identity`

In HOME, `IDENMAT(n)` creates the identity matrix of size n .

We enter:

```
IDENMAT(3)
```

We get the identity matrix of size 3:

```
[[1,0,0],[0,1,0],[0,0,1]]
```

In CAS, `IDENMAT(n)` creates the identity matrix of size n .

We enter:

```
IDENMAT(3)
```

Or we enter:

```
identity(3)
```

We get the identity matrix of size 3:

```
[[1,0,0],[0,1,0],[0,0,1]]
```

20.10.4 Matrix random: `RANDMAT` `randMat` `randmatrix` `ranm`

In HOME, `RANDMAT(M1,n,p)` creates a random matrix $M1$ of n rows and p columns and formed of integers between -99 and $+99$.

We enter:

```
RANDMAT(M1,2,3)
```

We get:

```
[[ -24.0, -67.0, 38.0], [-73.0, -3.0, 72.0]]
```

We enter:

```
M1
```

We get:

```
[[ -24, -67, 38], [-73, -3, 72]]
```

In CAS, we have two arguments only: `RANDMAT(n,p)` creates a random matrix of n rows and p columns, and formed of integers between -99 and $+99$.

We enter:

```
RANDMAT(2,3)
```

In CAS, or in HOME, `randMat(n,p)` or `randmatrix` or `ranm(n,p)` creates a random matrix of n rows and p columns, and formed of integers between -99 and $+99$.

We enter:

```
randMat(2,3)
```

Or we enter:

```
randmatrix(2,3)
```

Or we enter:

```
ranm(2,3)
```

We get for example:

```
[[ -57, 17, 39], [-61, 23, 4]]
```

20.10.5 Jordan block: `JordanBlock`

In HOME, we use it in the form of `CAS.JordanBlock(a,n)` because it is a CAS function.

In CAS, `JordanBlock(a,n)` returns a square matrix of size n , filled with a on the main diagonal, 1 above and 0 elsewhere.

We enter:

```
JordanBlock(7,3)
```

We get:

```
[[7,1,0],[0,7,1],[0,0,7]]
```

20.10.6 N-th Hilbert matrix: `hilbert`

`hilbert` is used in the CAS (in HOME, use `CAS.hilbert`).

`hilbert(n)` returns the n -th Hilbert matrix, that is to say:

$$H_{j,k} = \frac{1}{j+k+1} \text{ to } j = 1..n \text{ and } k = 1..n.$$

We enter:

```
hilbert(3)
```

We get:

```
[[1,1/2,1/3],[1/2,1/3,1/4],[1/3,1/4,1/5]]
```

20.10.7 Matrix of an isometry: `mkisom`

`mkisom` is used in the CAS (in HOME, use `CAS.mkisom`).

`mkisom` takes as argument:

- In dimension 3, the list of characteristic elements (axis direction, angle for a rotation or normal to the plane for a symmetry) and +1 or -1 for direct isometries or -1 indirect isometries).
- In dimension 2, the characteristic element (an angle or a vector) and +1 or -1 (+1 for direct isometries and -1 for indirect isometries).

`mkisom` returns the matrix of the isometry defined by the arguments.

We enter:

```
mkisom([-1,2,-1],pi,1)
```

We get the matrix of a rotation of axis $[-1, 2, -1]$ and angle π :

```
[-2/3,-2/3,1/3],[-2/3,1/3,-2/3],[1/3,-2/3,-2/3]]
```

We enter:

```
mkisom([pi],-1)
```

We get the matrix of a symmetry with respect to O :

```
[-1,0,0],[0,-1,0],[0,0,-1]]
```

We enter:

```
mkisom([1,1,1],-1)
```

We get the matrix of a symmetry with respect to the plane $x + y + z = 0$:

$$[[1/3, -2/3, -2/3], [-2/3, 1/3, -2/3], [-2/3, -2/3, 1/3]]$$

We enter:

$$\text{mkisom}([[1, 1, 1], \pi/3], -1)$$

We get the matrix product of a rotation of axis $[1, 1, 1]$ and angle $\frac{\pi}{3}$ and a symmetry with respect to the plane $x + y + z = 0$:

$$[[0, -1, 0], [0, 0, -1], [-1, 0, 0]]$$

We enter:

$$\text{mkisom}(\pi/2, 1)$$

We get the matrix, in dimension 2, of the plane rotation of angle $\frac{\pi}{2}$:

$$[[0, -1], [1, 0]]$$

We enter:

$$\text{mkisom}([1, 2], -1)$$

We get the matrix, in dimension 2, of the plane symmetry with respect to the line of equation $x + 2y = 0$:

$$[[3/5, -4/5], [-4/5, -3/5]]$$

To get the matrix in dimension 2 of rotation center O and angle 1, we enter:

$$\text{mkisom}(1, 1)$$

We get:

$$[[\cos(1), -\sin(1)], [\sin(1), \cos(1)]]$$

20.10.8 Vandermonde matrix: `vandermonde`

`vandermonde` is used in the CAS (in HOME, use `CAS.vandermonde`).

`vandermonde` takes as argument a vector whose components are x_j .

`vandermonde` returns the corresponding Vandermonde matrix: elle a for k -nth row the vector whose components are x_j^{k-1} ($k = 1..n$).

Warning!

The calculator numbers the rows and the columns starting from 1.

We enter:

$$\text{vandermonde}([a, 2, 3])$$

We get (if a is not assigned):

$$[[1, 1, 1], [a, 2, 3], [a^2, 4, 9]]$$

20.11 Basics

20.11.1 Schur norm or Frobenius norm of a matrix: `ABS`

We point out that `ABS` let us get:

- the absolute value of a real,
- the modulus of a complex number,
- the length of a vector $v_j \left(\left(\sum_{j=1}^n |v_j|^2 \right)^{\frac{1}{2}} \right)$,
- the Schur norm or Frobenius norm of a matrix $a_{j,k} : \left(\left(\sum_{j,k=1}^n |a_{j,k}|^2 \right)^{\frac{1}{2}} \right)$.

We enter:

```
ABS ([[1, 2], [3, 4]])
```

We get:

```
sqrt(30)
```

indeed $\sqrt{1 + 4 + 9 + 16} = \sqrt{30}$

We enter:

```
ABS ([[1, 2], [3, 4], [5, 11]])
```

We get:

```
4*sqrt(11)
```

indeed $\sqrt{1 + 4 + 9 + 16 + 25 + 121} = \sqrt{176} = 4\sqrt{11}$

In CAS, we enter:

```
abs ([[i, 2], [4, 5+i]])
```

Or we enter `| |` with the key on the right of the toolbox:

```
| [[i, 2], [4, 5+i]] |
```

We get:

```
sqrt(47)
```

because $\sqrt{1 + 4 + 16 + 25 + 1} = \sqrt{47}$

In HOME, we enter:

```
ABS ([[i, 2], [4, 5+i]])
```

Or we enter `| |` with the key on the right of the toolbox:

```
| [[i, 2], [4, 5+i]] |
```

We get:

```
6.8556546004
```

because $\sqrt{1 + 4 + 16 + 25 + 1} = \sqrt{47} \approx 6.8556546004$

20.11.2 Maximum of the norms of the rows of a matrix: `ROWNORM` `rownorm`

In HOME, `ROWNORM` takes as argument a matrix.

`ROWNORM` returns the maximum of the norms of the rows of this matrix (the norm of a row being the sum of the absolute values of the components of the line).

We enter:

```
ROWNORM([ [1, -2, 3], [4, 5, -6] ])
```

We get:

15

indeed we have $1 + 2 + 3 = 6 < 4 + 5 + 6 = 15$

In CAS, `ROWNORM` or `rownorm` takes as argument a matrix.

`ROWNORM` or `rownorm` returns the maximum of the norms of the rows of this matrix (the norm of a row being the sum of matrix absolute values of matrix components of the line).

We enter:

```
ROWNORM([ [1, -2, 3], [4, 5, -6] ])
```

Or we enter:

```
rownorm([ [1, -2, 3], [4, 5, -6] ])
```

We get:

15

indeed we have $1 + 2 + 3 = 6 < 4 + 5 + 6 = 15$

20.11.3 Maximum of matrix norms of matrix columns of a matrix: `COLNORM` `colnorm`

In HOME, `COLNORM` takes as argument a matrix.

`COLNORM` returns the maximum of the norms of the columns of this matrix (the norm of a column being the sum of the absolute values of the components of the column).

We enter:

```
COLNORM([ [1, -2, 3], [4, 5, -6] ])
```

We get:

9

indeed we have $1 + 4 < 2 + 5 < 3 + 6 = 9$

In CAS, `COLNORM` or `colnorm` takes as argument a matrix.

`COLNORM` or `colnorm` returns the maximum of the norms of the columns of this matrix (the norm of a column being the sum of the absolute values of the components of the column).

We enter:

```
COLNORM([ [1, -2, 3], [4, 5, -6] ])
```

Or we enter:

```
colnorm([ [1, -2, 3], [4, 5, -6] ])
```

We get:

9

indeed we have $1 + 4 < 2 + 5 < 3 + 6 = 9$

20.11.4 Spectral norm of a matrix: SPECNORM

In HOME, SPECNORM is written CAS.SPECNORM and takes as argument a matrix $M1$.

SPECNORM returns the spectral norm of this matrix $M1$: it is the largest singular value of the matrix $M1$ i.e. the root of the largest eigenvalue of the symmetric matrix $M1 * \text{TRN}(M1)$.

We enter:

```
CAS.SPECNORM([[1,1],[0,2]])
```

We get:

```
2.28824561127
```

because

```
eigenvals([[1,1],[0,2]]*trn([[1,1],[0,2]]))=[sqrt(5)+3,-sqrt(5)+3]
```

```
EIGENVAL([[1,1],[0,2]]*TRN([[1,1],[0,2]]))
```

```
=[0.7639320225,5.2360679775]
```

and

```
SVL([[1,1],[0,2]])=[sqrt(sqrt(5)+3),sqrt(-sqrt(5)+3)]
```

and

$$\sqrt{\sqrt{5} + 3} \approx 2.28824561127$$

In CAS, SPECNORM takes as argument a matrix A .

SPECNORM returns the spectral norm of this matrix A : it is the largest singular value of the matrix A i.e. the root of the largest eigenvalue of the symmetric matrix $A * \text{TRAN}(A)$.

We enter:

```
SPECNORM([[1,1],[0,2]])
```

We get:

```
2.28824561127
```

because

```
eigenvals([[1,1],[0,2]]*trn([[1,1],[0,2]]))=[sqrt(5)+3,-sqrt(5)+3]
```

```
EIGENVAL([[1,1],[0,2]]*TRN([[1,1],[0,2]]))=[0.7639320225,5.2360679775]
```

and

```
SVL([[1,1],[0,2]])=[sqrt(sqrt(5)+3),sqrt(-sqrt(5)+3)]
```

and

$\sqrt{\sqrt{5} + 3} \approx 2.28824561127$. Indeed, SVL(A), returns the list of singular values (i.e. the positive square roots of the eigenvalues of $A * \text{trn}(A)$) of the real numerical matrix A supplied as argument.

20.11.5 Spectral radius of a square matrix: SPECRAD

In HOME, SPECRAD is written CAS.SPECRAD and takes as argument a square matrix.

`SPECRAD` returns the spectral radius of this square matrix: the spectral radius is the largest eigenvalue in absolute value.

We enter:

```
CAS.SPECRAD ([[1, 1], [0, -2]])
```

We get:

2.

In CAS, `SPECNORM` takes as argument a matrix A.

`SPECNORM` returns the spectral radius of this square matrix: the spectral radius is the largest eigenvalue in absolute value.

We enter:

```
SPECRAD ([[1, 1], [0, -2]])
```

We get:

2.

20.11.6 Condition number of an invertible square matrix: `COND` `cond`

In HOME (resp. in the CAS), `COND` (resp. `cond`) takes as argument an invertible square matrix and as second argument 1, 2 or ∞ (obtained with Shift 9) (resp. 1, 2 or `inf`), by default 1, specifying the norm used (l_1 , l_2 or l_∞).

`COND` returns the condition number of this invertible square matrix for the norm specified:

- if the second argument is 1, the condition number of an invertible square matrix is the product of the column norm (it is `colnorm`) of this matrix by the column norm of its inverse matrix. The column norm of $M1$ of dimension p, q is:

$$\text{MAX}_{(1 \leq k \leq q)} \left(\sum_{j=1}^p \text{ABS}(M1[j, k]) \right)$$

We enter:

```
COND ([[1, 2], [5, 6]])
```

or

```
cond ([[1, 2], [5, 6]])
```

We get:

22

We have indeed:

Column norm of $[[1, 2], [5, 6]]$ is $2 + 6 = 8$

Column norm of $[[1, 2], [3, -4]]^{-1} = [[-1.5, 0.5], [1.25, -0.25]]$ is $1.5 + 1.25 = 2.75$.

We do have $2.75 * 8 = 22$.

- if the second argument is 2, the condition number of an invertible square matrix is the product of the spectral norm (it is `SPECNORM`, and also `max(SVL())`) of this matrix by the spectral norm of its inverse matrix.

We enter:

```
COND ([[1, 2], [3, -4]], 2)
```

or

```
cond([[1, 2], [3, -4]], 2)
```

We get:

```
2.6180339888
```

We have indeed:

```
SPECNORM(INV([[1, 2], [3, -4]]))=5.116672736
```

```
SPECNORM(INV([[1, 2], [3, -4]]))=0.5116672736.
```

We do have $5.116672736 * 0.5116672736 = 2.6180339888$.

- if the second argument is `inf`, the condition number of an invertible square matrix is the product of the norm row (it is `rownorm`) of this matrix by the norm row of its inverse matrix.

The norm row of $M1$ of dimension p, q is:

$$\text{MAX}_{(1 \leq j \leq p)} \left(\sum_{k=1}^q \text{ABS}(M1[j, k]) \right)$$

We enter:

```
COND([[1, 2], [5, 6]], inf)
```

or

```
cond([[1, 2], [5, 6]], inf)
```

We get:

```
22.0
```

We have indeed:

Norm row of $[[1, 2], [5, 6]]$ is $5 + 6 = 11$

Norm row of $[[1, 2], [3, -4]]^{-1} = [[-1.5, 0.5], [1.25, -0.25]]$ is $1.5 + 0.5 = 2$.

We do have $2 * 11 = 22$.

20.11.7 Rank of a matrix: `RANK rank`

In `HOME`, `RANK` returns the rank of the matrix supplied as argument.

We enter:

```
RANK([[1, 2, 3], [4, 5, 6]])
```

We get:

```
2
```

In `CAS`, `RANK` or `rank` returns the rank of the matrix supplied as argument.

We enter:

```
RANK([[1, 2, 3], [4, 5, 6]])
```

Or we enter:

```
rank([[1, 2, 3], [4, 5, 6]])
```

We get:

20.11.8 Step of the Gauss-Jordan reduction of a matrix: `pivot`

`pivot` is used in the CAS (in HOME, use `CAS.pivot`).

`pivot` has three arguments: a matrix of n rows and p columns and two integers l and c such as: $0 \leq l < n$ and $0 \leq c < p$.

`pivot(A, l, c)` returns the matrix obtained by putting zeros in the column c of A , with the method of Gauss-Jordan, using the element $A[l, c]$ as pivot.

We enter:

```
pivot([[1,2],[3,4],[5,6]],1,1)
```

We get:

```
[[ -2, 0 ], [ 3, 4 ], [ 2, 0 ]]
```

We enter:

```
pivot([[1,2],[3,4],[5,6]],0,1)
```

We get:

```
[[ 1, 2 ], [ 2, 0 ], [ 4, 0 ]]
```

20.11.9 Trace of a square matrix: `TRACE` `trace`

In HOME, `TRACE` returns the trace of the square matrix supplied as argument.

We enter:

```
TRACE([[1/2,2,3],[4,5,6],[7,8,9]])
```

We get:

```
14.5
```

We enter:

```
TRACE([[i,2],[4,5-i]])
```

We get:

```
5
```

In CAS, `TRACE` or `trace` returns the trace of the square matrix supplied as argument.

We enter:

```
TRACE([[1/2,2,3],[4,5,6],[7,8,9]])
```

Or we enter:

```
trace([[1,2,3],[4,5,6],[7,8,9]])
```

We get:

```
29/2
```

We enter:

```
TRACE ([[i, 2], [4, 5-i]])
```

Or we enter:

```
trace ([[i, 2], [4, 5-i]])
```

We get:

5

20.12 Advanced

20.12.1 Eigenvalues: EIGENVAL eigenvals

In HOME, `EIGENVAL` returns the vector of the calculable eigenvalues of a diagonalizable numerical matrix.

We enter:

```
EIGENVAL ([[1, 1], [0, 2]])
```

We get:

[2, 1]

In CAS, `eigenvals` returns the vector of the calculable eigenvalues of a matrix.

We enter:

```
eigenvals ([[1, 1, 2], [0, 1, 1], [0, 0, 1]])
```

We get:

[1, 1, 1]

We enter:

```
eigenvals ([[1, 1, 2], [0, 2, 1], [0, 0, 3]])
```

We get:

[3, 2, 1]

We enter:

```
eigenvals ([[1, 1, 2], [0, 1, 1], [0, 0, 1]])
```

We get:

[1, 1, 1]

We enter:

```
eigenvals ([[1, 1, 2], [0, 2, 1], [0, 0, 3]])
```

We get:

$[3, 2, 1]$

20.12.2 Eigenvectors: EIGENVV eigenvects

In HOME, `EIGENVV` returns a list of two matrices: the one of eigenvectors and the one of the calculable eigenvalues of a numerical diagonalizable matrix.

We enter:

```
EIGENVV([[1,1],[0,2]])
```

We get:

```
{[[0.707106781187,-1.41421356237],[0.707106781187,0]],[[2,0],[0,1]]}
```

In CAS, `eigenvects` or `eigVc` returns the matrix of eigenvectors of a diagonalizable matrix.

We enter:

```
eigenvects([[1,1],[0,2]])
```

or

```
eigVc([[1,1],[0,2]])
```

We get:

```
[[1,-1],[1,0]]
```

We enter:

```
eigenvects([[1,1,2],[0,1,1],[0,0,1]])
```

or

```
eigVc([[1,1,2],[0,1,1],[0,0,1]])
```

We get:

```
"Low accuracy or not diagonalizable at some eigenvalue.  
Try jordan if the matrix is exact."
```

We enter:

```
eigenvects([[1,1,2],[0,2,1],[0,0,3]])
```

or

```
eigVc([[1,1,2],[0,2,1],[0,0,3]])
```

We get:

```
[[3,-1,1],[2,-1,0],[2,0,0]],[[3,0,0],[0,2,0],[0,0,1]]
```

20.12.3 Jordan matrix: eigVl

`eigVl` takes as argument a matrix of size n .
`eigVl` returns the matrix of Jordan associated to this matrix.

Note: if the matrix is symbolic, we can get as a numerical result eigenvalues because the CAS must formally factorize the characteristic polynomial!

We enter:

```
egV1([[4, 1, -2], [1, 2, -1], [2, 1, 0]])
```

We get:

```
[[2, 1, 0], [0, 2, 1], [0, 0, 2]]
```

We enter:

```
egV1([[4, 1, 0], [1, 2, -1], [2, 1, 0]])
```

We get:

```
[0.324869129433, 0, 0], [0, 4.21431974338, 0], [0, 0, 1.46081112719]]
```

20.12.4 Jordan matrix and its transfer matrix: `jordan`

`jordan` is used in the CAS (in HOME, use `CAS.jordan` and the result will be exact).

`jordan` returns the list formed by the transfer matrix and the Jordan form of a matrix.

We enter:

```
jordan([[1, 1, 2], [0, 2, 1], [0, 0, 3]])
```

We get:

```
[[3, -1, 1], [2, -1, 0], [2, 0, 0]]
```

We enter:

```
jordan([[1, 1, 2], [0, 1, 1], [0, 0, 1]])
```

We get:

```
[[1, 2, 0], [0, 1, 0], [0, 0, 1]], [[1, 1, 0], [0, 1, 1], [0, 0, 1]]
```

If $A = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, $P = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and

$B = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, we have:

$\text{inv}(P) * A * P$ returns B .

20.12.5 Power n of a square matrix: `matpow`

`matpow` raises a square matrix to the power n by jordanization.

We enter:

```
matpow([[1, 2], [2, 1]], n)
```

We get:

```
[((-1)^(n+3^n))/2, (-(-1)^(n+3^n))/2], [(-(-1)^(n+3^n))/2, ((-1)^(n+3^n))/2]]
```

We have indeed:

`jordan([[1, 2], [2, 1]])` returns:

```
[[1, -1], [1, 1]], [[3, 0], [0, -1]]
```

20.12.6 Diagonal matrix and its diagonal: `diag`

`diag` is used in the CAS (in HOME, use `CAS.diag`).

When `diag` takes as argument a matrix, `diag` returns the vector formed by the elements of its diagonal.

When `diag` takes as argument a vector, `diag` returns the diagonal matrix, of diagonal the elements of this vector.

We enter:

```
diag([[1,0],[0,2]])
```

We get:

```
[1,2]
```

We enter:

```
diag([1,2])
```

We get:

```
[[1,0],[0,2]]
```

20.12.7 Cholesky matrix: `cholesky`

`cholesky` is used in the CAS (in HOME, use `CAS.cholesky`).

`cholesky` takes as argument a symmetric matrix A .

`cholesky` returns the matrix L such as $A=L*\text{tran}(L)$

We enter:

```
cholesky([[3,1],[1,4]])
```

We get:

```
[[3/(sqrt(3)),0],[1/(sqrt(3)),(sqrt(33))/3]]
```

and we do have `[[3/(sqrt(3)),0],[1/(sqrt(3)),(sqrt(33))/3]]*`

`tran([[3/(sqrt(3)),0],[1/(sqrt(3)),(sqrt(33))/3]])`

returns `[[3,1],[1,1/3+11/3]]`

20.12.8 Hermite normal form of a matrix: `ihermite`

`ihermite` returns the Hermite normal form of a matrix A of integer coefficients.

`ihermite` returns U, B such as U is invertible in \mathbb{Z} , B is upper triangular and $B = U * A$.

We enter:

```
ihermite([[1,2],[2,3]])
```

We get:

```
[[ -3, 2 ], [ 2, -1 ]], [[ 1, 0 ], [ 0, 1 ]]
```

20.12.9 Matrix reduction to Hessenberg form: `hessenberg`

`hessenberg` is used in the CAS (in HOME, use `CAS.hessenberg`).

`hessenberg` takes as first argument a matrix A and as second argument $0, -1$ or -2 or $n > 1$ and n prime.

`hessenberg` returns the transfer matrix P and the matrix B similar to A whose coefficients sous-sous-diagonaux are null. We say that B is a Hessenberg matrix and we have $B = P^{-1}AP$ or $B \sim P^{-1}AP$ according to the second argument.

- With one single argument or as second argument 0, calculations are exact.
- With as second argument -1 , the calculations are approximate and the matrix B is triangular.
- With as second argument -2 , the calculations are approximate and the matrix P is orthogonal and the matrix B has its subdiagonal coefficients are null.
- With as second argument $n > 1$ and n prime, the calculations are given modulus n and the matrix B is triangular.

We enter:

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]])
```

We get:

```
[[[1,0,0],[0,4/7,1],[0,1,0]], [[1,29/7,2],[7,39/7,8],[0,278/49,3/7]]]
```

We enter:

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]],-1)
```

We get:

```
[[[-0.293737737475,0.802770468103,0.518919759814],
[-0.69005396727,-0.553745992027,0.466037443312],
[-0.661470833702,0.221189854777,-0.716611041155]],
[[12.4541647409,-2.25953233593,-4.26290461387],
[8.03071937292e-17,-0.379762185881,0.849798726727],
[4.52383345971e-20,-9.66694414605e-19,-5.07440255497]]]
```

We enter:

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]],-2)
```

We get:

```
[[[1,0.0,0.0],
[0,0.496138938357,0.868243142124],
[0,0.868243142124,-0.496138938357]],
[[1.0,3.59700730309,0.248069469178],
[8.0622577483,8.01538461538,6.27692307692],
[0,4.27692307692,-2.01538461538]]]
```

We enter:

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]],3)
```

We get:

```
[[[1,0,0],[0,1,0],[0,1,1]], [[1,-1,0],[1,-1,0],[0,1,1]]]
```

20.12.10 Smith normal form of a matrix: `ismith`

`ismith` is used in the CAS (in HOME, use `CAS.ismith`).

`ismith(A)` returns the Smith normal form of the matrix A with integer coefficients and returns the matrices U, B, V with U and V invertible in \mathbb{Z} and where B is a diagonal matrix so that $B[j, j]$ is a divider of $B[j + 1, j + 1]$ and $B = U * A * V$.

We enter:

```
ismith([[1,2],[2,3]])
```

We get:

```
[[1,0],[2,-1]], [[1,0],[0,1]], [[1,-2],[0,1]]
```

We enter:

```
ismith([[9,-36,30],[-36,192,-180],[30,-180,180]])
```

We get:

```
[[ -3, 0, 1], [ 6, 4, 3], [20, 15, 12]], [[ 3, 0, 0], [ 0, 12, 0], [ 0, 0, 60]],
[[ 1, 24, -30], [ 0, 1, 0], [ 0, 0, 1]]
```

20.13 Factorization

20.13.1 LQ decomposition of a matrix: `LQ`

In HOME `LQ(M1)`, (resp. in the CAS `LQ(A)`), returns the LQ decomposition of a numerical matrix $M1$ (resp. A) of dimension $m \times n$ in a lower triangular matrix $M2$ (resp. L) of dimension $m \times n$, an orthogonal matrix $M3$ (resp. Q) of dimension $n \times n$ and a permutation matrix $M4$ (resp. P) of dimension $n \times n$ such as $M4 * M1 = M2 * M3$ (resp. $P * A = L * Q$).

We enter:

```
LQ([[4,0,0],[8,-4,3]])
```

We get:

```
[[4.0,0,0],[8.0,5.0,0]], [[1,0,0],
[0,-0.8,0.6],[0,-0.6,-0.8]], [[1,0],[0,1]]
```

We enter:

```
LQ([[0.8,0.6],[2.2,0.4]])
```

We get:

```
[[1.0,0],[2.0,-1.0]], [[0.8,0.6],[-0.6,0.8]], [[1,0],[0,1]]
```

We enter:

```
LQ([[4,3],[11,2]])
```

We get:

```
[[5.0,0],[10.0,-5.0]], [[0.8,0.6],[-0.6,0.8]], [[1,0],[0,1]]
```

We enter:

```
LQ([[1,2],[3,4]])
```

We get:

```
[[2.2360679775,0.],[4.9193495505,0.894427191]],
[[0.4472135955,0.894427191],[0.894427191,-0.4472135955]],
[[1,0],[0,1]]]
```

which means that:

```
[[1,2],[3,4]]=[[2.2360679775,0.],[4.9193495505,0.894427191]]*
[[0.4472135955,0.894427191],[0.894427191,-0.4472135955]]
```

We enter:

```
[[1,2,3],[3,4,5],[5,6,7]]=>M3
LQ(M3)
```

We get:

```
[[3.74165738677,0,0],[6.94879228972,1.30930734142,0],
[10.1559271927,2.61861468283,1]],
[[0.267261241912,0.534522483825,0.801783725737],
[0.872871560944,0.218217890236,-0.436435780472],
[-9.09494701773e-13,-2.27373675443e-13,6.8212102633e-13]],
[[1,0,0],[0,1,0],[0,0,1]]]
```

20.13.2 Minimal norm of the linear system $A * X = B$: LSQ

LSQ(A,B) returns the minimal norm to the least squares method of the over or underdetermined linear system $A * X = B$, to estimate the solution of a linear system $A * X = B$ (if B is a vector) or of linear systems $A * X = B$ (if B is a matrix) for:

- an overdetermined system (more rows than columns)
- if B is a vector: we look for X, of Euclidean norm minimal which minimizes the Euclidean norm of $(AX-B)$.
- if B is a matrix: we look for X_j of Euclidean norm minimal among the solutions which minimize the Euclidean norm of $(AX_j - B_j)$
- an underdetermined system (most of the time: more columns than rows)
We look for X which minimizes the Frobenius norm of $(AX-B)$ (the Frobenius norm of a matrix M is $\sqrt{\sum |M(j,k)|^2}$).
- a system exactly determined (the number of columns equals the number of rows and A is invertible).
We use $\text{inv}(A) * B$ to get X which gives false results with approximate calculation if the matrix is badly determined (independent equations close one to the other)

We enter:

```
LSQ([[1,2],[3,4]],[[5,-1],[11,-1]])
```

We get:

```
[[1,1],[2,-1]]
```

We enter:

```
LSQ([[1,2]], [[5,-1]])
```

We get:

```
[[1,-1.2],[2,-0.4]]
```

We enter:

```
LSQ([[1,2],[3,4],[3,6]], [[5,-1],[11,-1],[15,-3]])
```

We get:

```
[[1,1],[2,-1]]
```

We enter:

```
LSQ([[1,2],[3,4],[3,6]], [[5,-1],[11,-1],[15,-1]])
```

We get:

```
[[1,-0.2],[2,-0.1]]
```

20.13.3 LU decomposition of a square matrix: LU

Warning! `LU` and `lu` do not return the same result.

In HOME `LU(M1)`, returns the LU decomposition of a square matrix $M1$ (resp. $A1$) in a lower triangular matrix $M2$ (resp. L) (of diagonal 1) and an upper triangular matrix $M3$ (resp. U) such as, if $M4$ (resp. P) is a permutation matrix we have $M4 * M1 = M2 * M3$ (resp. $P * A1 = L * U$).

We enter:

```
LU([[1,2],[1,4]])
```

We get:

```
{[[1,0],[1,1]], [[1,2],[0,2]], [[1,0],[0,1]]}
```

We enter:

```
LU([[1,2],[3,4]])
```

We get:

```
{[[1,0],[0.333333333333,1]],
 [[3,4],[0,0.666666666667]],
 [[0,1],[1,0]]}
```

because we have chosen to put 1 on the diagonal of L .

This means that:

```
[[0,1],[1,0]]*[[1,2],[3,4]]) = [[3,0],[1,0.666666]]*[[1,1.333333],[0,1]]
```

We enter:

```
LU([[1,2,4],[4,5,6],[7,8,9]])
```

We get:

$$[[[1,0,0],[4,1,0],[7,2,1]], [[1,2,4],[0,-3,-10],[0,0,1]],[0,1,2]]$$

which means that:

$$[[1,2,4],[4,5,6],[7,8,9]] =$$

$$[[1,0,0],[4,1,0],[7,2,1]] * [[1,2,4],[0,-3,-10],[0,0,1]]$$

because the matrix associated to the permutation [0,1,2] is the identity matrix of size 3.

We enter:

$$\text{LU}([[6,12,18], [5,14,31], [3,8,18]])$$

We get:

$$[[[1,0,0],[2,1,0],[5/3,(-1)/6,1]],[[3,8,18],[0,-4,-18],[0,0,-2]],[2,0,1]]$$

which means that:

$$[[0,0,1],[1,0,0],[0,1,0]] * [[6,12,18],[5,14,31],[3,8,18]] =$$

$$[[1,0,0],[2,1,0],[5/3,(-1)/6,1]] * [[3,8,18],[0,-4,-18],[0,0,-2]]$$

because the matrix associated to the permutation [2,0,1] is the matrix [[0,0,1],[1,0,0],[0,1,0]].

20.13.4 LU decomposition: `lu`

Warning! `LU` and `lu` do not return the same result.

In HOME (resp. in the CAS) `lu` takes as argument a square matrix $M1$ (resp. $A1$) of size n (numerical or symbolic).

`lu(M1)` returns a permutation p of $1..n$ (because in HOME the indices start at 1), a lower triangular matrix L with 1 on its diagonal and an upper triangular matrix U .

`lu(A)` returns a permutation p of $1..n$ (because in the CAS the indices also start at 1), a lower triangular matrix L with 1 on its diagonal and an upper triangular matrix U .

These matrices are such as:

- $P * M1 = L * U$ (resp. $P * A1 = L * U$), where P is the permutation matrix associated to p (that we can calculate in the CAS with `P:=permu2mat(p)`),
- the equation $A * x = B$ equals:
 $L * U * x = P * B = p(B)$ where $p(B) = [b_{p(1)}, b_{p(2)} \dots b_{p(n)}]$, $B = [b_1, b_2 \dots b_n]$

We can also define from p the permutation matrix P_n by:

$$P_n[i, p(i)] := 1$$

and

$$P_n[i, j] := 0 \text{ if } j \neq p(i).$$

It is the matrix obtained by permuting, according to permutation p , the rows of the matrix unity.

We can use the function `permu2mat`: `permu2mat(p)` returns the matrix P of size n .

We enter in HOME:

$$(p, L, U) := \text{lu}([[3, 5], [4, 5]])$$

We get:

$$[1, 2], [[1, 0], [1.33333333333, 1]], [[4, 5], [0, -1.66666666667]]$$

We enter in the CAS:

$$(p, L, U) := \text{lu}([[3, 5], [4, 5]])$$

We get:

$$[1, 2], [[1, 0], [4/3, 1]], [[3, 5], [0, -5/3]]$$

We have indeed $n = 2$, then:

$$P[0, p(0)] = P_2[0, 1] = 1, P[1, p(1)] = P_2[1, 0] = 1, P = [[0, 1], [1, 0]]$$

Checking:

We enter:

$$\text{permu2mat}(p) * A1; L * U$$

We get:

$$[[4.0, 5.0], [3.0, 5.0]], [[4.0, 5.0], [3.0, 5.0]]$$

Please note that the permutation is different when the data are exact (the choice of the pivot is easier).

We enter in the CAS:

$$\text{lu}([[1, 2], [3, 4]])$$

We get:

$$[1, 2], [[1, 0], [3, 1]], [[1, 2], [0, -2]]$$

20.13.5 QR decomposition of a square matrix: QR qr

In HOME $\text{QR}(M1)$, (resp. in the CAS $\text{qr}(A)$), returns the QR decomposition of a square matrix $M1$ (resp. A) in a matrix Q orthogonal and an upper triangular matrix R such as, if P is a permutation matrix, we have $M1 * P = Q * R$ (resp. $A * P = Q * R$).

We enter:

$$\text{QR}([[4, 11, -2], [3, 2, 11]])$$

or

$$\text{qr}([[4, 11, -2], [3, 2, 11]])$$

We get:

$$\{ [[0.8, -0.6], [0.6, 0.8]], [[5.0, 10.0, 5.0], [0, -5.0, 10.0]], [[1, 0], [0, 1]] \}$$

We enter:

$$\text{QR}([[1, 2], [3, 4]])$$

or

$$\text{qr}([[1, 2], [3, 4]])$$

We get:

$$[[0.316227766017, 0.948683298051], [0.94868329805, -0.316227766017]], [[3.16227766017, 4.42718872424], [0.0, 0.632455532034]], [[1, 0], [0, 1]]$$

which means that:

$$[[0.316227766017, 0.948683298051], [0.94868329805, -0.316227766017]] * [[3.16227766017, 4.42718872424], [0.0, 0.632455532034]] = [[1, 2], [3, 4]]$$

We enter:

```
QR([[1, 2, 4], [4, 5, 6], [7, 8, 9]])
```

We get:

```
[[[1, 0, 0], [4, 1, 0], [7, 2, 1]], [[1, 2, 4], [0, -3, -10], [0, 0, 1]], [0, 1, 2]]
```

which means that:

```
[[1, 2, 4], [4, 5, 6], [7, 8, 9]] =
[[1, 0, 0], [4, 1, 0], [7, 2, 1]] * [[1, 2, 4], [0, -3, -10], [0, 0, 1]]
```

20.13.6 Matrix reduction to Hessenberg form: SCHUR schur

In HOME `SCHUR(M1)`, (resp. in the CAS `schur(A)`), returns the numerical matrices $[P, B]$ such as $B = \text{inv}(P) * M_1 * P$ (resp. $B = \text{inv}(P) * A * P$) with B triangular.

We have `SCHUR(A)=hessenberg(A, -1)`. B is the Hessenberg matrix similar to the matrix M_1 (resp. A).

We enter:

```
SCHUR([[1, 2, 3], [4, 5, 6], [-1, 3, -2]])
```

We get two matrices P (orthogonal transfer matrix $\text{tran}(P)=\text{inv}(P)$) and B (triangular matrix similar to the argument):

```
[[[0.353452714748, -0.31069680265, 0.882348386557],
[0.905760021954, -0.122092619725, -0.405822836763],
[0.23381608385, 0.94263507734, 0.238262774897]],
[[8.10977222864, 3.79381095693, 2.32899008373],
[0.0, -3.0, -3.03031411127], [0.0, 0.0, -1.10977222865]]]
```

and we have:

```
B~ inv(P) * [[1, 2, 3], [4, 5, 6], [-1, 3, -2]] * P.
```

We enter:

```
SCHUR([[1, 2, 4], [4, 5, 6], [7, 8, 9]])
```

We get two matrices P (orthogonal transfer matrix $\text{tran}(P)=\text{inv}(P)$) and B (triangular matrix similar to the argument):

```
[[[-0.275726630766, -0.921788330317, -0.272545591008],
[-0.518148584403, -0.0962872203049, 0.849853408352],
[-0.809627611663, 0.375546329096, -0.451074367633]],
[[16.5037894003, 3.99680014234, -0.803622341526],
[-4.55776517517e-20, -1.61625693617, 0.616262731649],
[4.1752316389e-20, -2.72155736143e-15, 0.112467535861]]]
```

and we have:

```
MB~ inv(P) * [[1, 2, 3], [4, 5, 6], [-1, 3, -2]] * P
```

20.13.7 Singular value decomposition: SVD svd

In HOME `SVD(M1)` returns 1 matrix M_2 , 1 vector M_3 , 1 matrix M_4 .

This gives the factorization of the rectangular numerical matrix real M_1 (of size $m * n$) in $M_2 * \text{diag}(M_3) * \text{TRN}(M_4)$ where M_2 is an orthogonal matrix $m * m$, M_4 is an orthogonal matrix $n * n$, and $\text{diag}(M_3)$ is a diagonal matrix of dimension $m * n$ having as diagonal the singular values M_3 of M_1 .

In CAS, `svd(A)` returns 1 matrix U , 1 vector S , 1 matrix Q .

This gives the factorization of the rectangular numerical matrix real A (of size $m * n$) in $U * \text{diag}(S) * \text{TRN}(Q)$ where U is an orthogonal matrix $m * m$, Q is an orthogonal matrix $n * n$, and $\text{diag}(S)$ is a matrix diagonal of size $m * n$ having as diagonal the singular values S of A .

We enter in HOME:

```
M2,M3,M4:=SVD([[1,2],[2,1]])
```

We get:

```
{[[[0.707106781187,-0.707106781187],
[0.707106781187,0.707106781187]],
[3,1],
[[0.707106781187,0.707106781187],
[0.707106781187,-0.707106781187]]]}
```

and we have (here M_4 is symmetrical):

```
M2*diag(M3)*TRN(M4).
```

We enter in the CAS:

```
U,S,Q:=SVD([[1,2],[2,1]])
```

Or we enter in the CAS:

```
U,S,Q:=svd([[1,2],[2,1]])
```

We get:

```
{[[[0.707106781187,-0.707106781187],
[0.707106781187,0.707106781187]],
[3.,1.],
[[0.707106781187,0.707106781187],
[0.707106781187,-0.707106781187]]]}
```

and we have (Q is symmetrical):

```
[[[0.707106781187,-0.707106781187],[0.707106781187,0.707106781187]]*
[[3,0],[0,1]]*
[[0.707106781187,0.707106781187],[0.707106781187,-0.707106781187]]]=
[[1,2],[2,1]]
```

We enter in the CAS:

```
SVD([[1,2],[3,4]])
```

Or we enter in the CAS:

```
svd([[1,2],[3,4]])
```

We get:

```

[[-0.404553584834, -0.914514295677],
 [-0.914514295677, 0.404553584834]],
 [5.46498570422, 0.365966190626]
 [[-0.576048436767, 0.81741556047],
 [-0.81741556047, -0.576048436766]],

```

because

```

[[-0.404553584834, -0.914514295677], [-0.914514295677, 0.404553584834]]*
 [[5.46498570422, 0], [0, 0.365966190626]]*
 TRN([[[-0.576048436767, 0.81741556047], [-0.81741556047, -0.576048436766]])
 =
 [[1.0, 2.0], [3.0, 4.0]]

```

20.13.8 Singular values: SVL svl

In HOME `SVL(M1)`, (resp. in the CAS `svl(A)`), returns the list of singular values of the numerical real matrix M_1 (resp. A) i.e. the positive square roots of the eigenvalues of the real symmetrical matrix $M_1 * TRN(M_1)$ (resp. $A * TRN(A)$).

We enter:

```
SVL([[1, 4], [1, 1]])
```

Or:

```
svl([[1, 4], [1, 1]])
```

We get:

```
[4.30277563773, 0.697224362268]
```

because

```

eigenvals([[1, 4], [1, 1]]*[[1, 1], [4, 1]])
 returns
 (5*sqrt(13)+19)/2, (-5*sqrt(13)+19)/2
 and
 sqrt((5*sqrt(13)+19)/2.), sqrt((-5*sqrt(13)+19)/2.)
 returns
 4.30277563773, 0.697224362268

```

We enter:

```
SVL([[1, 2], [2, 1]])
```

or

```
svl([[1, 2], [2, 1]])
```

We get:

```
[3, 1]
```

because

```

EIGENVAL([[1, 2], [2, 1]]*[[1, 2], [2, 1]])
 returns
 [9, 1].

```

We enter:

```
SVL([[1,2],[3,4]])
```

or

```
svl([[1,2],[3,4]])
```

We get:

```
[5.46498570422,0.365966190626]
```

because

```
EIGENVAL([[1,2],[3,4]]*[[1,3],[2,4]])
```

returns

```
[29.8660687473,0.133931252682]
```

which are the aproached values of $\sqrt{221} + 15$ and $-\sqrt{221} + 15$.

We have:

$$\sqrt{\sqrt{221} + 15} \simeq 5.46498570422, \sqrt{-\sqrt{221} + 15} \simeq 0.365966190626$$

20.14 Vector

20.14.1 Cross product: CROSS cross

In HOME, `CROSS` returns the cross product of two vectors.

We enter:

```
CROSS([1,2,3],[4,5,6])
```

or

```
CROSS({1,2,3},{4,5,6})
```

We get:

```
[-3,6,-3]
```

because $2 * 6 - 5 * 3 = -3, 4 * 3 - 1 * 6 = 6, 5 - 4 * 2 = -3$

In CAS, `CROSS` or `cross` returns the cross product of two vectors.

We enter:

```
CROSS([1,2,3],[4,5,6])
```

or

```
cross([1,2,3],[4,5,6])
```

We get:

```
[-3,6,-3]
```

because $2 * 6 - 5 * 3 = -3, 4 * 3 - 1 * 6 = 6, 5 - 4 * 2 = -3$

20.14.2 Dot product: DOT dot

In HOME, `DOT` returns the dot product of two vectors.

We enter:

```
DOT([1,2,3],[3,4,5])
```

We get:

```
26
```

because $1 * 3 + 2 * 4 + 3 * 5 = 26$

In CAS, `DOT` returns the dot product of two vectors.

We enter:

```
DOT([1,2,3],[3,4,5])
```

Or we enter:

```
dot([1,2,3],[3,4,5])
```

We get:

```
26
```

because $1 * 3 + 2 * 4 + 3 * 5 = 26$

20.14.3 Norm l^2 : `l2norm`

`l2norm` is used in the CAS (in HOME, use `CAS.l2norm` and the answer will be exact).

`l2norm` returns the norm l^2 of a vector: it is the square root of the sum of the squares of its coordinates.

We enter:

```
l2norm([3,-4,2])
```

Or we enter:

```
l2norm(vector(3,-4,2))
```

We get:

```
sqrt(29)
```

Indeed: $x = 3, y = -4, z = 2$ and $29 = |x|^2 + |y|^2 + |z|^2$

20.14.4 Norm l^1 : `l1norm`

`l1norm` is used in the CAS (in HOME, use `CAS.l1norm` and the answer will be exact).

`l1norm` returns the norm l^1 of a vector: it is the sum of absolute values of its coordinates.

We enter:

```
l1norm([3,-4,2])
```

Or we enter:

```
l1norm(vector(3,-4,2))
```

We get:

Indeed: $x = 3, y = -4, z = 2$ and $9 = |x| + |y| + |z|$.

20.14.5 Norm of the maximum: `maxnorm`

`maxnorm` is used in the CAS (in HOME, use `CAS.maxnorm` and the answer will be exact).
`maxnorm` returns the norm l^∞ of a vector: it is the maximum of absolute values of its coordinates.

We enter:

```
maxnorm([3,-4,2])
```

Or we enter:

```
maxnorm(vector(3,-4,2))
```

We get:

4

Indeed: $x = 3, y = -4, z = 2$ and $4 = \max(|x|, |y|, |z|)$.

Chapter 21 Special functions

21.1 β function: Beta

Beta is used in the CAS (in HOME, use CAS.Beta and the answer will be exact).

Beta takes as argument two reals a, b , or three reals a, b, p , or three reals and 1: $a, b, p, 1$.

- with two arguments a, b , Beta returns the values of the function β at the point a, b of \mathbb{R}^2 .
We have by definition:

$$\beta(x, y) = \frac{\Gamma(x) * \Gamma(y)}{\Gamma(x + y)}$$

We have:

$$\beta(1, 1) = 1$$

$$\beta(n, 1) = \frac{1}{n}$$

and:

$$\beta(n, 2) = \frac{1}{n(n + 1)}$$

We have:

$$\text{Beta}(a, b) = \int_0^1 t^{a-1} * (1 - t)^{b-1} dt$$

Beta(a, b) is defined for a and b positive reals (so that the integral is convergent).

We enter:

Beta(5, 2)

We get:

1/30

We enter:

simplify(Beta(5, -3/2))

We get:

256/15

We enter:

Beta(x, y)

We get:

Gamma(x) * Gamma(y) / Gamma(x+y)

We enter:

Beta(5.1, 2.2)

We get:

0.0242053671402

- with three arguments a, b, p , it is the incomplete Beta function for p between 0 and 1, it is:

- $\text{Beta}(a, b, p) = \int_0^p t^{a-1} * (1-t)^{b-1} dt$, the integral goes from 0 to p instead of 0 to 1 for the Beta function.

We enter:

Beta(5,2,0.5)

We get:

0.0036458333333333

- with four arguments $a, b, p, 1$, if we put 1 as fourth argument, this returns the regularized incomplete beta function, *i.e.* the incomplete Beta function divided by $\text{Beta}(a, b)$.

We enter:

Beta(5,2,0.5,1)

We get:

0.109375

indeed $\text{Beta}(5,2)=1/30$ and $0.0036458333333333*30=0.109375$

21.2 Γ function: Gamma

Gamma is used in the CAS (in HOME, use CAS.Gamma and the answer will be exact).

Gamma takes as argument a number a .

Gamma returns the values of the function Γ at point a .

We have by definition:

$$\Gamma(a) = \int_0^{+\infty} e^{-t} t^{a-1} dt, \text{ if } a > 0$$

and we use the formula:

$\Gamma(a + 1) = a * \Gamma(a)$ if a is not a negative integer

Thus:

$$\begin{aligned} \Gamma(1) &= 1 \\ \Gamma(a + 1) &= a * \Gamma(a) \end{aligned}$$

and thus:

$$\Gamma(n + 1) = n!$$

We enter:

Gamma(5)

We get:

24

We enter:

Gamma(1/2)

We get:

sqrt(pi)

We enter:

Gamma(0.7)

We get:

1.29805533265

We enter:

Gamma (-0.3)

We get:

-4.32685110883

Indeed: $\text{Gamma}(0.7) = -0.3 * \text{Gamma}(-0.3)$

We enter:

Gamma (-1.3)

We get:

3.32834700679

Indeed:

$\text{Gamma}(0.7) = -0.3 * \text{Gamma}(-0.3) = (-0.3) * (-1.3) * \text{Gamma}(-1.3)$

21.3 Derivatives of the DiGamma function: Psi

Psi is used in the CAS (in HOME, use CAS.Psi and the answer will be exact).

Psi takes as arguments a real a and an integer n (by default $n = 0$).

Psi is the value of the n -th derivative of the DiGamma function at point a .

The DiGamma function is the derivative of $\ln(\Gamma(x))$.

We enter:

Psi (3,1)

We get:

$\pi^2/6 - 5/4$

We can omit the parameter n when $n = 0$.

When Psi takes as single parameter a number a , Psi returns the value of the DiGamma function at point a :

then we have $\text{Psi}(a, 0) = \text{Psi}(a)$.

We enter:

Psi (3)

We get:

$\text{Psi}(1) + 3/2$

We enter:

evalf(Psi(3))

We get:

.922784335098

21.4 The ζ function: Zeta

Zeta is used in the CAS (in HOME, use CAS.Zeta and the answer will be exact).

Zeta takes as argument a real x .

Zeta returns for $x > 1$:

$$\sum_{n=1}^{+\infty} \frac{1}{n^x}$$

We enter:

Zeta(2)

We get:

pi^2/6

We enter:

Zeta(4)

We get:

pi^4/90

21.5 erf function: erf

erf is used in the CAS (in HOME, use CAS.erf and the answer will be exact).

erf takes as argument a number a .

erf returns the values of the erf function at point a .

We have by definition:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

We have:

$$\begin{aligned} \operatorname{erf}(+\infty) &= 1 \\ \operatorname{erf}(-\infty) &= -1 \end{aligned}$$

Indeed, we know that:

$$\int_0^{+\infty} e^{-t^2} dt = \frac{\sqrt{\pi}}{2}$$

We enter:

erf(1)

We get:

0.84270079295

We enter:

erf(1/(sqrt(2)))*1/2+0.5

We get:

0.841344746069

Note:

There is a relationship between the functions erf and normal_cdf:

$$\operatorname{normal_cdf}(x) = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)$$

Indeed:

$$\text{normal_cdf}(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}} dt$$

so with the change of variable $t = u * \sqrt{2}$, we have:

$$\text{normal_cdf}(x) = \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-u^2} du = \frac{1}{2} + \frac{1}{2} \text{erf}\left(\frac{x}{\sqrt{2}}\right)$$

We check by entering:

$$\text{normal_cdf}(1) = 0.841344746069$$

21.6 erfc function: erfc

`erfc` is used in the CAS (in HOME, use `CAS.erfc` and the answer will be exact).

`erfc` takes as argument a number a .

`erfc` returns the values of the erfc function at point a .

We have by definition:

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{+\infty} e^{-t^2} dt = 1 - \text{erf}(x)$$

We have:

$$\begin{aligned} \text{erfc}(0) &= 1 \\ \text{erfc}() &= -1 \end{aligned}$$

Indeed, we know that:

$$\int_x^{+\infty} e^{-t^2} dt = \frac{\sqrt{\pi}}{2}$$

We enter:

$$\text{erfc}(1)$$

We get:

$$0.15729920705$$

We enter:

$$1 - \text{erfc}(1/(\text{sqrt}(2))) * 1/2$$

We get:

$$0.841344746069$$

Note:

There is a relationship between the functions `erfc` and `normal_cdf`:

$$\text{normal_cdf}(x) = 1 - \frac{1}{2} \text{erfc}\left(\frac{x}{\sqrt{2}}\right)$$

Indeed:

$$\text{normal_cdf}(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}} dt$$

so with the change of variable $t = u * \sqrt{2}$

$$\text{normal_cdf}(x) = \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-u^2} du = 1 - \frac{1}{2} \text{erfc}\left(\frac{x}{\sqrt{2}}\right)$$

We check by entering:

$$\text{normal_cdf}(1) = 0.841344746069$$

21.7 Exponential integral function: Ei

Ei is used in the CAS (in HOME, use CAS.Ei and the answer will be exact).

Ei takes as argument a complex number a .

Ei returns the values of the Ei function at point a .

We have by definition:

$$\text{Ei}(x) = \int_{t=-\infty}^x \frac{e^t}{t} dt$$

For $x > 0$, we extend by the principal value of the integral (the parts of 0^- and 0^+ compensate themselves). We have:

$$\text{Ei}(0) = -\infty, \quad \text{Ei}(-\infty) = 0$$

When we are close to $x = 0$ we know that:

$$\frac{\exp(x)}{x} = \frac{1}{x} + 1 + \frac{x}{2!} + \frac{x^2}{3!} + \dots + \frac{x^n}{(n-1)!} \dots$$

then we have for $x \in \mathbb{C} - \mathbb{R}^+$, (the function is discontinuous on \mathbb{R}^+):

$$\text{Ei}(x) = \ln(-x) + \gamma + x + \frac{x^2}{2.2!} + \frac{x^3}{3.3!} + \dots$$

where $\gamma = \text{Euler constant} = 0.57721566490..$

on the axis $x > 0$ we take:

$$\text{Ei}(x) = \ln(x) + \gamma + x + \frac{x^2}{2.2!} + \frac{x^3}{3.3!} + \dots$$

We enter:

Ei(1.)

We get:

1.89511781636

We enter:

Ei(-1.)

We get:

-0.219383934396

We enter:

Ei(1.)-Ei(-1.)

We get:

2.11450175075

We enter:

int((exp(x)-1)/x,x=-1..1.)

We get:

2.11450175075

We enter:

evalf(Ei(-1)-sum((-1)^n/n/n!,n=1..100))

We get the Euler constant γ :

0.577215664901532860606507

21.8 Sine integral function: Si

Si is used in the CAS (in HOME, use CAS.Si and the answer will be exact).

Si takes as argument a complex number a .

Si returns the values of the function Si at point a .

We have by definition

$$\text{Si}(x) = \int_{t=0}^x \frac{\sin(t)}{t} dt$$

We have $\text{Si}(0) = 0$, $\text{Si}(-\infty) = -\frac{\pi}{2}$, $\text{Si}(+\infty) = \frac{\pi}{2}$.

When we are close to $x = 0$, we know that:

$$\frac{\sin(x)}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} + \dots + (-1)^n \frac{x^{2n}}{(2n+1)!} \dots$$

which gives by integration the development in sequences of Si in 0.

We also note that Si is an odd function.

We enter:

Si(1.)

We get:

0.946083070367

We enter:

Si(-1.)

We get:

-0.946083070367

We enter:

Si(1.)+Si(-1.)

We get:

0

We enter:

Si(1.)-Si(-1.)

We get:

1.89216614073

We enter:

int(sin(x)/x, x=-1..1.)

We get:

1.89216614073

21.9 Cosine integral function: Ci

Ci is used in the CAS (in HOME, use CAS.Ci and the answer will be exact).

Ci takes as argument a complex number a.

Ci returns the values of the function Ci at point a.

We have by definition:

$$Ci(x) = \int_x^{t=+\infty} \frac{\cos(t)}{t} dt = \ln(x) + \gamma + \int_0^x \frac{\cos(t) - 1}{t} dt$$

We have: $Ci(0) = -\infty$, $Ci(-\infty) = i\pi$, $Ci(+\infty) = 0$.

When we are close to $x = 0$ we know that

$$\frac{\cos(x)}{x} = \frac{1}{x} - \frac{x}{2} + \frac{x^3}{4!} + \dots + (-1)^n \frac{x^{2n-1}}{(2n)!} \dots$$

which gives by integration the development in sequences of Ci.

We enter:

Ci(1.)

We get:

0.337403922901

We enter:

Ci(-1.)

We get:

0.337403922901+3.14159265359*i

We enter:

Ci(1.)-Ci(-1.)

We get:

-3.14159265359*i

We enter:

int((cos(x)-1)/x,x=-1..1.)

We get:

-3.14159265359*i

21.10 Heaviside function: Heaviside

Heaviside takes as argument a number a.

Heaviside returns the values of the Heaviside function at point a.

We have by definition:

$$Heaviside(x) = 0 \text{ if } x < 0 \text{ and } 1 \text{ otherwise}$$

We enter:

Heaviside(2)

We get:

1

We enter:

Heaviside(-4)

We get:

0

21.11 Dirac distribution: Dirac

Dirac takes as argument a number a .

Dirac is the Dirac distribution, it is the distribution associated to the function Heaviside.

We have by definition:

$$Dirac(x) = 0 \text{ if } x \neq 0 \text{ and } \infty \text{ otherwise}$$

and if $a \geq 0$ and $b \neq 0$ we have:

$$\int_b^a Dirac(x) dx = 1$$

$$\int_b^a Dirac(x)f(x)dx = [Heaviside(x)f(x)]_b^a - \int_b^a Heaviside(x)f'(x)dx = f(0)$$

$$\int_{-\infty}^{+\infty} Dirac(x) * f(x)dx = f(0)$$

We enter:

int(Dirac(x)*sin(x), x, -1, 2)

We get:

sin(0)

We enter:

int(Dirac(x-1)*sin(x), x, -1, 2)

We get:

sin(1)

Chapter 22 Constants and calculations with units

22.1 Shifted key Units

With the shifted key Units, and then Tools of the push buttons, we get the functions allowing to perform calculations with units:

convert, mksa, ufactor, usimplify.

With the shifted key Units, and then Units of the push buttons, we get the units sorted by category, in menu 1 the available prefixes.

With the shifted key Units, and then Const of the push buttons, we get the Mathematics, Chemistry, Physics and Quantum Mechanics constants.

22.2 Units

22.2.1 Notation of units

The names of units are prefixed with the symbol `_` ("underscore"). For instance `2_m` for 2 meters.

You can add a prefix ahead of the name of a unit meaning a multiplication by a power of 10. For example, `k` or `K` for kilo (means multiplication by 10^3), `D` for deca (means multiplication by 10), `d` for deci (means multiplication by 10^{-1}), etc..

When combining a real number with units, we create a unit object.

We enter:

```
10.5_m
```

We get:

```
a unit object of 10.5 meters
```

We enter:

```
10.5_km
```

We get:

```
a unit object of 10.5 kilometers
```

22.2.2 Available prefixes for units names

You can add prefixes ahead of the names of the units: each prefix corresponds to the name of the unit multiplied by a power of 10.

Here are the different available prefixes:

Prefix	Name	(*10 ⁿ)	Prefix	Name	(*10 ⁿ)
Y	yota	24	d	deci	-1
Z	zeta	21	c	cent	-2
E	exa	18	m	mili	-3
P	peta	15	mu	micro	-6
T	tera	12	n	nano	-9
G	giga	9	p	pico	-12
M	mega	6	f	femto	-15
k or K	kilo	3	a	atto	-18
h or H	hector	2	z	zepto	-21
D	deca	1	y	yocto	-24

Note:

You can of course not use a prefix with an integrated unit if this combination leads to another integrated unit.

For instance, 1_a is an are and 1_Pa is a Pascal, and not 10¹⁵_a.

22.2.3 Calculations with units

We can do the basic operations (+, -, *, /) with unit objects.

In the operations, we can use different units (provided they are compatibles for + and -) and the result will be expressed according to the corresponding unit. For the multiplication and the division of two different units _u1 and _u2 the result unity reads _(u1*u2) or _(u1/u2) (Mind to not forget the parentheses!)

We can also rise a unit object to an integer power: we get the corresponding unit object.

Please note that, as far as the addition or subtraction is concerned, the result will be expressed with the unit of the first term of the operation.

We enter:

$$1_m + 100_cm$$

We get:

$$2_m$$

We enter:

$$100_cm + 1_m$$

We get:

$$200_cm$$

We enter:

$$1_m * 100_cm$$

We get:

$$100_ (cm*m)$$

We enter:

$$3_h + 10_mn - (1_h + 45_mn)$$

We get:

$$1.416666666667_h$$

We enter:

$$10_mn + 3_h - (1_h + 45_mn)$$

We get:

$$85.0_mn$$

22.3 Tools

22.3.1 Conversion of a unit object to another unit: `convert` =>

`convert` allows to get the conversion of a unit object into another unit given as second parameter.
=> is the infix version of `convert`.

We enter:

```
convert(2_h+30_mn, _mn)
```

or else

```
2_h+30_mn=>_mn
```

We get:

```
150_mn
```

We enter:

```
convert(1_m*100_cm, _m^2)
```

or else

```
convert(100_(cm*m), _m^2)
```

or else

```
100_(cm*m)=>_m^2
```

We get:

```
1_m^2
```

We enter:

```
convert(1_h, _s)
```

Or we enter:

```
1_h=>_s
```

We get:

```
3600_s
```

We enter:

```
convert(60_mn, _h)
```

Or we enter:

```
60_mn=>_h
```

We get:

```
1.0_h
```

Note:

You must insert a space before the unit if the number of unit is stored in a variable or if it is a constant:
We enter:

```
convert(pi _rad, _deg)
```

Or we enter:

```
pi _rad=>_deg
```

We get:

```
180.0_deg
```

We enter:

```
a:=180
convert(a _deg, _rad)
```

Or we enter:

```
a _deg=>_rad
```

We get:

```
3.14159265358_rad
```

22.3.2 Units conversion to MKSA units: `mksa`

`mksa` allows to get the conversion of a unit object into a unit object expressed in MKSA units.

We enter:

```
mksa(15_C)
```

We get:

```
15_A*s
```

We enter:

```
mksa(1_Hz)
```

We get:

```
1_s^(-1)
```

22.3.3 Factorize a unit in a unit object: `ufactor`

`ufactor` allows to factorize the compound unit of a unit object to get a unit object expressed in constituent units (*i.e.* multiplied by the necessary MKSA units).

We enter:

```
ufactor(3_J, _W)
```

We get:

```
3_(W*s)
```

We enter:

```
ufactor(3_W,_J)
```

We get:

```
3_(J/s)
```

22.3.4 Simplify a unit: `usimplify`

`usimplify` allows to simplify a unit in a unit object.

We enter:

```
usimplify(3_(W*s))
```

We get:

```
3_J
```

22.4 Physics constants

With the shifted key `Units` then `Const` of the push buttons, then `3: Physics` we get the physics constants sorted by category. With the shifted key `Units` then `Unit` of the push buttons, we get the units sorted by category. The physics constants, and the units sorted by category, are in the menu `Physics`.

22.5 Units

22.5.1 Units notation

The names of units are prefixed with the symbol `_` ("underscore"). For instance `2_m` for 2 meters. You can add a prefix ahead of the name of a unit meaning a multiplication by a power of 10. For example, `k` or `K` for kilo (multiplication by 10^3), `D` for deca (means multiplication by 10), `d` for deci (means multiplication by 10^{-1}) etc., ...

When combining a real number with units, we create a unit object.

We enter:

```
10.5_m
```

We get:

```
a unit object of 10.5 meters
```

We enter:

```
10.5_km
```

We get:

```
a unit object of 10.5 kilometers
```

22.5.2 Calculations with units

In the operations, we can use different units (provided they are compatibles for `+` and `-`) and the result will be expressed according to the corresponding unit. For the multiplication and the division of two

different units `_u1` and `_u2` the result unity reads `_(u1*u2)` or `_(u1/u2)` (Mind to not forget the parentheses!)

We can also rise a unit object to an integer power: we get the corresponding unit object.

Please note that, as far as the addition or subtraction is concerned, the result will be expressed with the unit of the first term of the operation.

We enter:

```
1_m+100_cm
```

We get:

```
2_m
```

We enter:

```
100_cm+1_m
```

We get:

```
200_cm
```

We enter:

```
1_m*100_cm
```

We get:

```
100_(cm*m)
```

We enter:

```
3_h +10_mn-(1_h+45_mn)
```

We get:

```
1.416666666667_h
```

We enter:

```
10_mn+3_h-(1_h+45_mn)
```

We get:

```
85.0_mn
```

22.5.3 Conversion of a unit object into another unit: `convert =>`

`convert` allows to get the conversion of a unit object into another unit given as second parameter. `=>` is the infix version of `convert`.

We enter:

```
convert(2_h+30_mn, _mn)
```

or else

```
2_h+30_mn=>_mn
```

We get:

150_mn

We enter:

```
convert(1_m*100_cm, _m^2)
```

or else

```
convert(100_(cm*m), _m^2)
```

or else

```
100_(cm*m)=>_m^2
```

We get:

1_m^2

We enter:

```
convert(1_h, _s)
```

Or we enter:

```
1_h=>_s
```

We get:

3600_s

We enter:

```
convert(60_mn, _h)
```

Or we enter:

```
60_mn=>_h
```

We get:

1.0_h

Note:

You must insert a space before the unit if the numerical value of the unit is stored in a variable or if it is a constant:

We enter:

```
convert(pi _rad, _deg)
```

Or we enter:

```
pi _rad=>_deg
```

We get:

180.0_deg

We enter:

```
a:=180
```

```
convert(a _deg, _rad)
```

Or we enter:

```
a _deg=>_rad
```

We get:

```
3.14159265358_rad
```

22.5.4 Units conversion to MKSA units: `mksa`

`mksa` allows to get the conversion of a unit object to a unit object expressed in MKSA units.

We enter:

```
mksa(15_C)
```

We get:

```
15_A*s
```

We enter:

```
mksa(1_Hz)
```

We get:

```
1_s^(-1)
```

22.5.5 Conversions between degree Celsius and Fahrenheit: `Celsius2Fahrenheit` `Fahrenheit2Celsius`

`Celsius2Fahrenheit` allows to convert the Celsius degrees into Fahrenheit degrees.

We enter:

```
Celsius2Fahrenheit(a)
```

We get:

```
(a*9)/5+32
```

We enter:

```
Celsius2Fahrenheit(0)
```

We get:

```
32
```

`Fahrenheit2Celsius` allows to convert the Fahrenheit degrees into Celsius degrees.

We enter:

```
Fahrenheit2Celsius(a)
```

We get:

```
((a-32)*5)/9
```

We enter:

```
Fahrenheit2Celsius(212)
```

We get:

```
100
```

22.5.6 Factorization of a unit: `ufactor`

`ufactor` allows to factorize the compound unit of a unit object to get a unit object expressed in constituent units (*i.e.* multiplied by the necessary MKSA units).

We enter:

```
ufactor(3_J, _W)
```

We get:

```
3_(W*s)
```

We enter:

```
ufactor(3_W, _J)
```

We get:

```
3_(J/s)
```

22.5.7 Simplify a unit: `usimplify`

`usimplify` allows to simplify a unit in a unit object.

We enter:

```
usimplify(3_(W*s))
```

We get:

```
3_J
```

22.6 Constants

22.6.1 Notation of chemical, physics or quantum mechanics constants.

The names of physics constants start and end by the character `_` ("underscore"). Do not mix physics constants and symbolic constants. For example, e , π are symbolic constants whereas `_c_`, `_NA_` are physics or chemical constants.

We enter:

```
_c_
```

We get the light speed in vacuum:

```
299792458_m*s^-1
```

We enter:

```
_NA_
```

We get the Avogadro number:

$6.0221367e23_{\text{g mol}^{-1}}$

22.6.2 Physics constants library

Here is the constants library:

Name	Description
<code>_NA_</code>	Avogadro Number
<code>_k_</code>	Boltzmann constant
<code>_Vm_</code>	Molar volume of ideal gas
<code>_R_</code>	Molar gas constant
<code>_StdT_</code>	Standard temperature
<code>_StdP_</code>	Standard pressure
<code>_sigma_</code>	Stefan-Boltzmann constant
<code>_c_</code>	Light speed
<code>_epsilon0_</code>	Vacuum permittivity
<code>_mu0_</code>	Vacuum permeability
<code>_g_</code>	Acceleration of gravity
<code>_G_</code>	Gravitation
<code>_h_</code>	Planck constant
<code>_hbar_</code>	Dirac constant
<code>_q_</code>	Electronic charge
<code>_me_</code>	Electron mass
<code>_qme_</code>	q/me ratio (charge/mass of the electron)
<code>_mp_</code>	Proton mass
<code>_mpme_</code>	mp/me ratio (proton mass /electron mass)
<code>_alpha_</code>	fine structure
<code>_phi_</code>	Magnetic flux quantum
<code>_F_</code>	Faraday constant
<code>_Rinfinity_</code>	Rydberg constant
<code>_a0_</code>	Bohr radius
<code>_muB_</code>	Bohr magneton
<code>_muN_</code>	Nuclear magneton
<code>_lambda0_</code>	Photon wavelength
<code>_f0_</code>	Photon frequency
<code>_lambdac_</code>	Compton wavelength
<code>_rad_ 1</code>	radian
<code>_twopi_</code>	2*pi radians
<code>_angl_</code>	Angle of 180 degrees
<code>_c3_</code>	Wien displacement law constant
<code>_kq_</code>	k/q (Boltzmann/electronic charge)
<code>_epsilon0q_</code>	epsilon0/q (permittivity/electronic charge)
<code>_qepsilon0_</code>	q*epsilon0 (electronic charge*permittivity)
<code>_epsilonsi_</code>	Silicium dielectric constant
<code>_epsilonox_</code>	Silicium dioxyd dielectric constant
<code>_I0_</code>	Sound reference intensity

Chapter 23 Functions of 3D geometry

23.1 Common perpendicular to two 3D lines: `common_perpendicular`

`common_perpendicular` takes as argument two lines `D1` and `D2`.

`common_perpendicular(D1, D2)` draws the common perpendicular of lines `D1` and `D2`.

We enter:

```
D1:=line([1,1,0],[0,1,1]);
D2:=line([0,-1,0],[1,-1,1]);
```

Then, we enter:

```
d:=common_perpendicular(D1,D2)
```

We get:

```
pnt([[1,0,-1],[-1/3,-2/3,-1/3]],0,d)
```

Which means that the common perpendicular to `D1` and `D2` passes through the points $[1,0,-1]$ and $[-1/3,-2/3,-1/3]$.

Then, we enter:

```
equation(d)
```

We get:

```
[2/3-2*x/3+4*y/3=0,-4/3-8*x/9-4*y/9-20*z/9=0]
```

Part IV **The Applications and the Apps key**

Chapter 24 The menu Geometry

Warning! The documentation on the geometry application is subject to modification.

24.1 Generalities

We will describe here the Geometry application which allows to do interactive geometry.

As all the Apps, the plane geometry application has three views: the Symbolic view, the Plot view and the Numeric view.

Cmnds, in the push buttons of these three views, lists the geometry commands, sorted by categories useful in each of the views:

Point Line Polygon Curv Plot Transformation for the Symbolic view,

Zoom Point Line Polygon Curv Plot Transformation, Cartesian, Measure, Tests for the Plot view,

Cartesian, Measure, Tests for the Numeric view.

Let us detail these three views with an example:

1. the key `Symb` opens the Symbolic view.
Tap `Edit` or `Insert` in the push buttons, and fill in the cell `GC:circle(1,2)`
For that:
 - enter `circle` (spelt out) or tap `Cmnds` in the push buttons (`Cmnds->Curv>Circle`).
 - If it is the first command you enter, it will be named `GA`. You can modify this name by selecting `GA` and editing it by tapping `Edit` in the push buttons. Change `A` for `C` and press `Enter`.
 - directly plot a circle in the Plot view so that `circle(point(GA),GB-GA)` is written in `GC`, in the Symbolic view. You can then modify the values of `GA` and `GB - GA` by `point(1)` and

Notes

- a. To have the command stored in `GC` executed, check the cell ahead of `GC`.
 - b. To highlight a command line, use the cursor.
 - c. To reorder the commands, use the arrows `↑` and `↓` of the push buttons: with them you can move the highlighted line.
 - d. To insert a new command, press `Insert` of the push buttons.
 - e. To change the name of a variable in the Symbolic view, select this name, edit it with `Edit` of the push buttons, modify it, and confirm with `Enter`.
 - f. To delete a command line, highlight it, then press the delete key.
2. The key `Plot` opens the Plot view.
To directly plot a circle in the Plot view, use `Cmnds` (push buttons) (`Cmnds->Curv>Circle`), then designate with your finger the point which will be the center (you can then refine with the cursor), and press `Enter` to confirm your choice: the command `point(2.375,1.24)` (for example) has been automatically registered in `GA` in the Symbolic view.
Then, directly with your finger a second point, this time on the circle (you can then refine with the cursor) and press `Enter` to confirm your choice.
The command `point(2.375,3.24)` (for example) is then written in `GB` and the command `circle(point(GA),GB-GA)` is then written in `GC` in the Symbolic view.
Once the circle is drawn, you can:
 - a. change the color.
Put your finger close to the contour of the circle to have `Options` appear in the push buttons, then tap `Options->Choose` to choose a color and `Enter`. The color palette opens: select a color with your finger and confirm with `Enter`.
 - b. Fill it with color
Put your finger close to the contour of the circle to have `Options` appear in the push buttons, then tap `Options->Filled` and `Enter`, which fills the circle with the chosen color.
 - c. Hide its name

Put your finger close to the contour of the circle to have `Options` appear in the push buttons, then tap `Options->Hide Label`: in case of ambiguity, the calculator will propose several choices.

d. Move it

Put your finger close to the name of the circle (refine with the cursor) to have `Options` appear in the push buttons, then `Enter`. Then, you can drag it anywhere with your finger or with the cursor.

Confirm with `Enter`.

Note:

Please note that the points or the geometrical objects directly created in the Plot view are written automatically: the geometrical objects are named and designated by `A`, `B`, `C`... then stored in the variables `GA`, `GB`, `GC`... variables also listed in the Symbolic view. Please note that the point `F` does not exist because `GF` is a CAS command.

3. the key `Num` opens the Numeric view.

If you want the equation of the circle previously defined, press: `equation(GC)` and you get:

$$(x-1.0)^2+y^2=4.0$$

The Numeric view allows to use the numerical commands in relationship with the graphic and to get numerical results.

The commands giving a numerical result (points coordinates, lines equations or of curves...) can be executed from the Num screen by using `New`, then `Cmds` (push buttons) of the Num screen.

When we are in the CAS or in the Geometry Application, the commands of geometry are sorted by category in the menu `Apps->Geometry` of the key `Toolbox`. We find there the nine categories of the `Cmds` menu of each (push buttons) of the various views:

`Point`, `Line`, `Polygon`, `Curve`, `Plot`, `Transformation`, `Cartesian`, `Measure`, `Tests`.

Note:

All the geometry functions can be executed from the CAS, but, in this case, the answer will be such as, for example: `point(1,2)`, or `line(x=1)`, but not a plot.

Then, we can do 2D, or even 3D analytic geometry from the CAS.

For example, let us enter from the CAS:

`g:=line(x=1)` then add in the Symb screen `GK:=g`.

Warning!

We can retrieve the value of these variables in the Symbolic view, in the Numeric view, or in the CAS screen. If, for example, in the geometry application, we have: `GA:=point(4.16+2.13*i)` and in the CAS we enter `GA:=5`, `GA` will be equal to 5, as long as we have not used `Plot` of the geometry application, because, once we will have done this, we will have back `GA:=point(4.16+2.13*i)`.

For this reason, it is not safe to use the variables `GA`, `GB`, ... in the CAS.

24.2 Point

24.2.1 Point defined as barycenter of n points: `barycenter`

In plane geometry, `barycenter` takes as argument n lists of length 2 (resp. a matrix of n lines and two columns):

the first element of the list j (resp. the j -th element of the first column of the matrix) stores the point A_j or the complex number a_j representing the affix of this point, the second element of the list j (resp. the j -th element of the second column) stores the real coefficient α_j assigned to A_j .

`barycenter` returns and plots the point which is the barycenter of points A_j of affixes a_j assigned of real coefficients α_j when $\sum \alpha_j \neq 0$.

If $\sum \alpha_j = 0$, `barycenter` returns an error.

We enter:

```
barycenter([1+i,1],[1-i,1])
```

Or we enter:

```
barycenter([point(1,1),1],[point(1,-1),1])
```

Or we enter:

```
barycenter([[1+i,1],[1-i,1]])
```

Or we enter:

```
barycenter([[point(1,1),1],[point(1,-1),1]])
```

We get in the geometry application:

The point of affix 1 is plotted with a cross and its label (name)

We get in the CAS:

```
point(1)
```

Warning! In the geometry application, if you want to get as answer a complex number, you have to ask for the affix of the barycenter, otherwise we get the plot of the barycenter point.

We enter:

```
affix(barycenter([1+i,1],[1-i,1]))
```

Or we enter:

```
affix(barycenter([[1+i,1],[1-i,1]]))
```

We get:

```
1
```

In CAS screen, `barycenter` may also be used in 3D geometry and takes as argument n lists of length 2 (or a matrix of n lines and two columns). The first element of the list j (resp. the j -th element of the first column of the matrix) stores the point A_j , the second element of the list j (resp. the j -th element of the second column) stores the real coefficient α_j assigned to A_j .

`barycenter` returns `point([a,b,c])`, where `[a,b,c]` are the coordinates of the barycenter of these n points.

We enter:

```
barycenter([point(0,0,0),1],[point(3,3,3),2])
```

We get:

```
pnt(pn([point(2,2,2),0]))
```

24.2.2 Point in geometry: `point`

In the `Plot` view, to get a point, it is enough to be in mode `point` (*i.e.* `Point` of the push buttons, then select `Point` and `Enter`) and locate the cursor at the wished place (with the finger or the arrows) then to confirm with `Enter`: a point is displayed as well as a label.

This label is automatically generated: A, then B, etc., ...

We can also use the command `point`:

`point` takes as argument a complex number or a paired value of two real numbers.

Warning!

If a, b is a paired value of two complex numbers whose one is non real, `GK:=point(a,b)` returns two points of same label (here K): one of affix a , the other of affix b .

When a, b is a paired value of two real numbers, `GA:=point(a,b)` returns and plots the point having for affix $a+ib$.

We enter:

```
GA:=point(1+i)
```

We get:

```
The point A of affix 1+i is plotted with a cross
```

We enter:

```
GB:=point(-2,1)
```

We get:

```
The point B of affix -2+i is plotted with a cross
```

We enter:

```
GC:=point(-2,i)
```

We get:

```
The two points of affix -2 and i are plotted with a cross and are
written C
```

Note When doing an assignment, for example `GA:=point(-2+i)`, this stores the `point(-2+i)` in the variable `GA`, to plot the point with a cross and to assign it as a label the name on the left of `:=` by omitting the letter `G`: here `A`.

In the case we do several assignments with one single `:=` sign, such as:

`GD,GE:=point(-2+i),point(2+i)`, the variable `GD` stores the `point(-2+i)`, and `GE` the `point(2+i)`, but it will not be possible to move these points by pointing them.

To avoid this, we must enter:

```
GL:=point(-2+i),point(2+i);GD=L[0];GE=L[1]
```

or

```
GL:=point(-2+i,2+i);GD=L[0];GE=L[1] which defines the point D of affix -2 and the point
E of affix i (because the affix of GD is not real!).
```

24.2.3 Midpoint of a segment: `midpoint`

In plane geometry, `midpoint` takes as argument two points or two complex numbers representing the affixes of these points (or else a list of two points or of two complex numbers).

`midpoint` returns and plots the point midpoint of the segment defined by these two points.

We enter:

```
midpoint(-1,1+i)
```

We get in the geometry application:

```
The point of affix i/2 is plotted with its label
```

In CAS screen, `midpoint` may also be used in 3D geometry and returns the point midpoint of the segment defined by two points.

We enter:

```
midpoint(point(0,0,0),point(2,2,2))
```

We get:

```
point(1,1,1)
```

24.2.4 Isobarycenter of n points: `isobarycenter`

`isobarycenter` takes as argument the list (or the sequence) of n points or n complex numbers representing the affixes of these points.

`isobarycenter` returns and plots a point which is the isobarycenter of these n points.

We enter:

```
isobarycenter(0,2,2*i)
```

We get:

```
The point of affix 2/3+2*i/3 is plotted with a cross in the geometry application
```

In CAS screen, `isobarycenter` may also be used in 3D geometry and takes as argument the list (or the sequence) of n points.

`isobarycenter` returns `point([a,b,c])` where $[a,b,c]$ are the coordinates of the isobarycenter of these n points.

We enter:

```
isobarycenter(point(0,0,0),point(3,3,3))
```

We get:

```
pnt(pn[(point[3/2,3/2,3/2],0)])
```

24.2.5 Randomly define a 2D point: `point2d`

`point2d` takes as argument a sequence of names of points.

`point2d` randomly defines the integer coordinates (between -5 and $+5$) of the 2D points supplied as argument.

We enter:

```
point2d(A,B,C)
```

Then, we enter:

```
triangle(A,B,C)
```

We get:

```
The plot of a triangle ABC
```

Warning!

The points defined by the command `point2d` are fixed once and for all, and hence, they may not be moved.

24.2.6 Polar point in plane geometry: `polar_point`

`polar_point(r,t)` returns the 2D point of polar coordinates the arguments `r` and `t`, that is to say the point of affix $r \cdot \exp(i \cdot t)$.

We enter:

```
polar_point(2,pi/4)
```

We get:

```
The plot of the point of affix 2*exp(i*pi/4)
```

24.2.7 One of the intersection points of two geometrical objects: `single_inter`

`single_inter` takes two or three arguments which are two geometrical objects and eventually a third argument which is either a point or a list of points.

`single_inter` returns one of the intersection points of these two geometrical objects.

If we have supplied a point `GA` (or its affixe) as third argument, `single_inter` returns the intersection point the closest to `GA`, and if we have supplied a list of points `l` (or a list of affixes), `single_inter` returns the intersection point which is not in the list `l`.

We enter:

```
GA:=single_inter(line(0,1+i),line(1,i))
```

We get:

```
The point of affix 1/2+i/2 is plotted with a cross and is labeled A
```

We enter:

```
GB:=single_inter(circle(0,1),line(-1,i))
```

We get:

```
The point of affix i is plotted with a cross and is labeled B
```

We enter:

```
GB1:=single_inter(circle(0,1),line(-1,i),[i])
```

We get:

```
The point of affix -1 is plotted with a cross and is labeled B1
```

We enter:

```
GB2:=single_inter(circle(0,1),line(-1,1+2*i),1+2*i)
```

We get:

```
The point of affix i is plotted with a cross and is labeled B2
```

We enter:

```
GC:=single_inter(circle(1,sqrt(2)),circle(0,1))
```

We get:

```
The point of affix i is plotted with a cross and is labeled C
```

We enter:

```
GC1:=single_inter(circle(1,sqrt(2)),circle(0,1),[i])
```

We get:

The point of affix $-i$ is plotted with a cross and is labeled C1

We enter:

```
GC2:=single_inter(circle(1,sqrt(2)),circle(0,1),i/2)
```

We get:

The point of affix i is plotted with a cross and is labeled C2

24.2.8 All intersection points of two geometrical objects: `inter`

`inter` takes two arguments or three arguments:

- if `inter` is supplied with two geometrical objects as arguments, it returns the list of points of intersection of these two geometrical objects.
- if `inter` is supplied with two geometrical objects and a point as arguments, it returns the intersection point of these two geometrical objects the closest of the point supplied as third argument.

We enter in geometry:

```
GA:=inter(line(0,1+i),line(1,i))[0]
```

We get:

The point of affix $1/2+i/2$ is plotted with a cross and is labeled A

We enter in geometry:

```
GB:=inter(circle(0,1),line(1,i))[0]
```

```
GC:=inter(circle(0,1),line(1,i))[1]
```

We get:

The point of affix i is plotted with a cross and is labeled B

The point of affix 1 is plotted with a cross and is labeled C

We enter in the CAS:

```
inter(circle(0,1),line(1,i))
```

We get:

```
[point(1),point(i)]
```

We enter in geometry in the Symbolic view:

```
GL inter(circle(0,1),line(1,i))
```

We get in the Plot view:

The points of affix i and 1 are plotted with a cross and are written
L

We enter in plane geometry:

```
GD:=inter(circle(0,1),line(1,i),point(1/2))
```

We get:

The point of affix 1 is plotted with a cross and is labeled D

24.2.9 Orthocenter of a triangle: orthocenter

`orthocenter` takes as argument a triangle, or three points, or three complex numbers specifying the affix of three points.

`orthocenter` plots and returns the point which is the orthocenter of the triangle, or of the triangle formed by these three points.

We enter:

```
orthocentre(0,1+i,-1+i)
```

Or we enter:

```
orthocenter(triangle(0,1+i,-1+i))
```

We get:

The point of affix 0 is plotted with a cross

We enter in the Symbolic view of the geometry application:

```
GT triangle(-i,2+i,-1+i);GH orthocenter(T)
```

We get:

The triangle T and the point H of affix 0 are plotted

24.2.10 Vertices of a polygon: vertices

`vertices` takes as argument a polygon.

`vertices` returns the list of the vertices of this polygon and plots them.

Warning! If the polygon has n vertices the list will be of length n .

We enter:

```
vertices(equilateral_triangle(0,2))
```

We get:

the points `[pnt(0,0),pnt(2,0),pnt((2*(sqrt(3)*i)+1)/2,0)]` are plotted with a cross

We enter:

```
GC:=vertices(equilateral_triangle(0,2))[2]
```

We get:

The point of affix $1 + i\sqrt{3}$ is plotted with a cross and is labeled C

Warning! If we enter:

```
GT:=equilateral_triangle(0,2,C);
```

We get:

```
the triangle T and the point C
```

Whereas, if we enter

```
GT:=equilateral_triangle(0,2,C)::vertices(GT[0])
```

We get:

```
the vertices of T plotted with a cross without label
```

24.2.11 Vertices of a polygon: `vertices_abca`

`vertices_abca` takes as argument the name of a polygon.

`vertices_abca` returns the "closed" list of vertices of this polygon and plot them.

Warning! If the polygon has n vertices the list will be of length $n + 1$ because it starts and ends by the first vertex ("closed" list).

We enter:

```
vertices_abca(equilateral_triangle(0,2))
```

We get:

```
[pnt(0,0),pnt(2,0),pnt((2*(sqrt(3)*(i)+1))/2,0),pnt(0,0)]
```

24.2.12 Point on a geometrical object: `element`

`element` may take different kind of arguments:

1. an interval $a..b$ and two reals, the value and the step (by default the value equals $(a + b)/2$ and the step $(b - a)/100$). For example, we enter in the Symbolic view:

```
GC:=element(-pi..pi) or
GC:=element(-pi..pi,pi/2) or
GC:=element(-pi..pi,pi/2,pi/100.0)
```

this means that `GC` can take any value in the range $[-\pi; \pi]$, the second argument $\pi/2$ is the start value of `GC` and $\pi/100.0$ is the chosen step.

Then, in the Plot view:

- we have at the bottom-left the coordinates x_1, y_1 of the pointer. We enter `C` in Alpha mode,
- then we get at the bottom-right Pick `GC`; Press `Enter` to validate.
- then pick `C`.

A line appears at the top of the Plot view: it is a cursor which allows to change the value of `GC` with the arrows (\leftarrow and \rightarrow) and at the bottom-right we have Move `GC` to the left of this line we have `GC=xC` (x_C is the value of `GC`). The arrows \leftarrow and \rightarrow allow to change the value x_C of `GC`.

Example.

We define `GC` as above:

We enter in the Symbolic view:

```
GC:=element(-pi..pi,pi/2)
```

```
GD:=line(y+x*TAN(GC)-2*SIN(GC)=0)
```

`GD` is then a line of parameter `GC`.

When we move the cursor `GC`, the line `GD` moves.

We can keep the trace of this line `GD` by entering in the Symbolic view: `trace(GD)` or by using in the menu of the Plot view `Point->Plus->trace`, which allows to choose the name

of the object whose we want the trace of and thus `trace(GD)` reads automatically in the Symbolic view.

Thus, we can see that the envelop of these lines is an astroid.

Note:

To delete the trace or to stop it, use the menu `Point->Plus`.

2. a geometrical object and a real (by default this real equals $1/2$), for example:

`GA:=element(circle(0,2),1)` means that `A` is on the circle of center 0 and radius 2, and has as affix $2 * \exp(i)$ (because $2 * \exp(i * t)$ is the parametric equation of this circle and the second argument 1 gives the value of the parameter t to define `GA`).

For instance, `GA:=element(circle(0,1))` means that `A` is on the circle of center 0 and radius 1, the point `A` will be plotted by having $t = 1/2$ as value of the parameter of the parametric equation of the geometrical object (here $\text{affix}(GA) = 2 * \exp(i/2)$). When moving `A` with the arrows, `A` will follow the outline of the geometrical object.

Warning! It is the projection of the cursor on the circle which defines the point `A`: take care to move the cursor in the Plot view so that it defines a point `A`.

3. a geometrical object and a name of variable (for example `GC`) previously defined by the command `element`: for example `GC:=element(0..pi)`.

If we enter `GD:=element(circle(0,2),GC)`, then `GC` is the variable of setting of the geometrical object defined by the first argument of `element`, that is to say that `GD` is on the circle of center 0 and radius 2, and `GD` has as affix $2 * \exp(i * GC)$, because $2 * \exp(i * t)$ is the parametric equation of the `circle(0,2)`. It is compulsory in this case to previously define the second argument (here `GC`) as being an element of an interval.

By example, we enter:

```
GC:=element(0..pi)
```

then

```
GD:=element(circle(0,2),GC)
```

Then, we place the cursor on `GC` (Pointer `GC`), then Enter. As a result, we have at the top a cursor labelled `GC` that we can move with the arrows (\leftarrow and \rightarrow) from 0 to π , with at the left of this cursor a number equal to the value of the cursor. This cursor allows to move the point `A` on the top half-circle of the circle of center 0 and radius 1 (because $0 \leq t \leq \pi$) and this without plotting this half-circle.

By example, we enter:

```
GA:=point(1);GB:=point(2+i)
```

```
GC:=element(0..2)
```

then

```
GD:=element(line(GA,GB),GC)
```

`D` is a point of the line `AB` and we have $M=A+t*(B-A)$ *i.e.* $M=(1-t)*A+t*B$

to follow the segment `AB`, you have to put `GC:=element(0..1)` or else `GD:=element(segment(GA,GB),GC)` which leaves `D` in `A` if $t < 0$ and leave `D` in `B` if $t > 1$.

4. a polygonal line `GP` and `[floor(GC),frac(GC)]` with `GC` previously defined by the command `element`: for example `GC:=element(0..5)` if `GP` has 5 sides.

The sides of the polygonal line `GP` have as number: 0,1,...

If, for example, `GP` has 5 sides and as vertices `S(0),...S(4),S(5)=S(0)`, we enter:

```
GC:=element(0..5)
```

```
GD:=element(GP,[floor(GC),frac(GC)])
```

Thus, according to the values of `GC`, `D` will follow the 5 sides of `GP`: `D` will be located on the side number $n=floor(GC)$ and we will have:

$D=frac(GC)*S(n)+(1-frac(GC))*S(n+1)$.

For instance:

```

GA:=point(0);
GB:=point(4);
GC:=point(4*i);
Gd:=element(0..3);
GT:=triangle(A,B,C);
GM:=element(GT,[floor(GD),frac(GD)]);

```

Warning! If we add a complex a to a point M of affix m , defined as element of a curve C , this defines a point N of the curve C which is the projection the point of affix $m + a$ on C .

Par contre if a point M of affix m , defined as element of a curve C , we add a point A of affix a , this defines a point P of affix $m + a$. For instance, being supplied 3 points M, A, B , if we want to define the point N that makes for example:

$$\overrightarrow{MN} = \overrightarrow{AB},$$

We can enter: $GN:=GM+(GB-GA)$ provided that M is not defined as element of a curve C . Indeed, if we have entered $GM:=element(GC)$, we must define N by entering: $GN:=affix(GM)+GB-GA$ or $GN:=GM+GB-GA$ (without parenthesis) because $GN:=GM+GB-GA$ is interpreted as $GN:=(GM+GB)-GA$ because there are no precedence rules for $+$ and $-$ whereas

$GN:=GM+(GB-GA)$ returns an element of the curve C which is the projection of N on C .

Thus, if we enter:

```

GA:=point(-2,2);GB:=point(1,3);GC:=circle(0,1);
GM:=element(GC);GN:=affix(GM)+GB-GA;(or GN:=GM+GB-GA;)

```

GN is not on the curve C

but if we enter:

```
GP:=GM+(GB-GA) (or GP:=projection(GC,GN);)
```

P is on the curve C .

24.2.13 Point dividing a segment: `division_point`

`division_point` takes three arguments: two points (or two complex numbers a and b) and a complex number k .

`division_point` returns and plots the point of affix z such as:

$$\frac{z-a}{z-b} = k$$

We enter:

```
GA:=division_point(i,2+i,3+i)
```

We get:

```
the point A of affix (5+4*i)/(2+i)
```

because `csolve(z-i=(3+i)*(z-2-i),z)` returns `[(14+3*i)/5]` and `(5+4*i)/(2+i)` returns `(14+3*i)/5`

We enter:

```
GB:=division_point(point(i),point(2+i),3)
```

We get:

```
the point B of affix 3+i
```

because `csolve(z-i=3*(z-2-i),z)` returns `[3+i]`

24.2.14 Harmonic division: `harmonic_division`

Four points aligned A, B, C, D are in harmonic division if we have:

$$\frac{\overline{CA}}{\overline{CB}} = -\frac{\overline{DA}}{\overline{DB}} = k$$

We also say that C and D divide the segment AB with the ratio k and that the point D is the conjugate harmonic of C according to A and B or, shortly, D is the harmonic conjugate of A, B, C .

Four concurrent or parallel lines d_1, d_2, d_3, d_4 are in harmonic division if they define an harmonic division on each secant line.

We also say that d_1, d_2, d_3, d_4 form an harmonic bundle.

`harmonic_division` takes as arguments three points aligned or their three affixes (resp. three concurrent or parallel lines) and the name of a variable.

`harmonic_division` modifies the last argument so that we get an harmonic division and returns the list of four points (resp. list of four lines) and plots the points (resp. the lines).

We enter:

```
harmonic_division(0,2,3/2,GD)
```

We get:

```
[0,2,3/2,pnt(3,0,"GD")] and only the point D is plotted
```

We enter:

```
harmonic_division(point(0),point(2),point(3/2),GD)
```

We get:

```
[pnt(0,0),pnt(2,0),pnt(3/2,0), pnt(3,0,"D")] and the four points are plotted
```

Note: 0 stands for the color of the point.

We enter:

```
harmonic_division(line(i,0),line(i,1+i), line(i,3+2*(i)),GD)
```

We get:

```
[pnt([[i,0],0)),pnt([[i,1+i],0)), pnt([[i,3+2*i],0)),  
pnt([[i,-3+2*i],0,"GD"])] and the four lines are plotted
```

24.2.15 Harmonic conjugate: `harmonic_conjugate`

`harmonic_conjugate` takes as arguments three points aligned GA, GB, GC (resp. three concurrent or parallel lines).

`harmonic_conjugate` returns and draws the conjugate harmonic of GC with respect to GA and GB .

We enter:

```
harmonic_conjugate(0,2,3/2)
```

We get:

```
pnt(3,0) and the plot of this point
```

We enter:

```
harmonic_conjugate(line(0,1+i),line(0,3+i),line(0,i))
```

We get:

```
pnt([[0,3+2*i],0]) and the plot of this line
```

24.2.16 Pole and polar: `pole polar`

`polar` takes as argument a circle GC and a point GA (or a complex number).

`polar` returns and draws the polar of the point GA with respect to the circle GC : it is the line which is the locus of conjugates of GA with respect to the circle GC .

`pole` takes as argument a circle GC and a line Gd .

`pole` returns and draws the pole of Gd with respect to the circle GC : it is the point GA having Gd as polar according to GC .

We enter:

```
polaire(circle(0,1), (point(1+i))/2)
```

We get:

```
pnt([[2,2*i],0]) and the plot of this line
```

We enter:

```
pole(circle(0,1), line(i,1))
```

We get:

```
pnt(1+i,0) and the plot of this point
```

24.2.17 Reciprocal polar: `reciprocation`

`reciprocation` takes as argument a circle GC and a list of points and lines.

`reciprocation` returns the list obtained by replacing in the list supplied as argument a point (resp. a line) by its polar (resp. its pole) with respect to the circle GC .

We enter:

```
reciprocation(circle(0,1), [point((1+i)/2), line(1,-1+i)])
```

We get:

```
the line of equation  $y = (-x + 2)$  and the point of affix  $1+2i$ 
```

24.2.18 The center of a circle: `center`

`center` takes as argument the name of a circle (see the definition of the circle ??).

`center` returns and plots the center of this circle.

We enter:

```
GC:=center(circle(0,point(2*i)))
```

We get:

```
The point of affix  $i$  is plotted with a cross and is labeled C
```

We enter:

```
GM:=center(circle(point(1+i),1))
```

We get:

```
The point of affix  $1+i$  is plotted with a cross and is labeled M
```

24.3 Line

24.3.1 Line defined by a point and a slope: DrawSlp

`DrawSlp(a,b,m)` draws the line of slope m passing by the point (a,b)

We enter:

```
DrawSlp(1,2,-1)
```

We get:

The line passing by the point of affix $1+2i$ and slope -1

24.3.2 Tangent to the curve of $y = f(x)$ in $x = a$: LineTan

`LineTan(f(x),x=a)` plots the tangent to the curve of $y = f(x)$ in $x = a$.

We enter:

```
LineTan(sin(x),pi/6)
```

Or we enter:

```
LineTan(sin(t),t,pi/6)
```

We enter:

```
LineTan(sin(t),t=pi/6)
```

We get:

The plot of the tangent to the curve $y = \sin(x)$ at the point of abscissa $x = \pi/6$

24.3.3 Altitude of a triangle: altitude

`altitude(GA,GB,GC)` plots the altitude of the triangle ABC through A .

We enter:

```
altitude(1,0,1-i)
```

We get in the geometry application:

The plot of the altitude of the triangle $(1, 0, 1 - i)$ through the point of affix 1

We enter:

```
altitude(0,1,2-i)
```

We get in the geometry application:

The plot of the altitude of the triangle $(0, 1, 2 - i)$ through the point of affix 0

We enter in the CAS screen:

```
a:=altitude(1,0,1-i)
```

We get:

```
line(y=x-1)
```

We enter in the CAS screen:

```
a:=altitude(1,0,1-i)
```

We get:

```
line(y=x)
```

24.3.4 Internal bisector of a angle: `bisector`

`bisector(GA,GB,GC)` plots the internal bisector of the angle \widehat{BAC} .

We enter:

```
bisector(0,1,i)
```

We get in the geometry application:

The plot of the internal bisector of the angle $(\widehat{0,1,i})$

We enter in the CAS screen:

```
bisector(0,1,i)
```

We get:

```
line(y=x)
```

24.3.5 External bisector of a angle: `exbisector`

`exbisector(GA,GB,GC)` plots the internal bisector of the angle \widehat{BAC} .

We enter:

```
exbisector(0,1,i)
```

We get in the geometry application:

The plot of the external bisector of the angle $(\widehat{0,1,i})$

We enter in the CAS screen:

```
exbisector(0,1,i)
```

We get:

```
line(y=-x)
```

24.3.6 Half line: `half_line`

`half_line(GA,GB)` plots the half line A,B .

We enter:

```
half_line(1,2+i)
```

We get in the geometry application:

The plot of the half-line of origin the point of affix 1 and passing by the point of affix $2+i$.

We enter in the CAS screen:

```
half_line(1,2+i)
```

We get:

```
line(y=x-1)
```

24.3.7 Line and oriented line: `line`

In plane geometry, `line` takes as argument two points (or two complex numbers representing the affixes of these points), or a list of two points (or two complex numbers), or takes as argument a point and slope= m , or else a line equation of the form $a * x + by + c = 0$.

`line` returns and plots the line defined by the two arguments.

`line(GA, GB)` plots the line A, B .

Note: `slope` is also a command giving the slope of a line. You would better use `DrawSlp` to define a line with a point and its slope. (`DrawSlp(a, b, m)` defines the line `line(point(a, b), slope=m)`).

In the CAS screen:

We enter:

```
line(1,2+i)
```

We get:

```
line(y=x-1)
```

In the geometry application:

We enter:

```
line(1,2+i)
```

We get:

The plot of the line passing by the point of affix 1 and through the point of affix $2+i$.

We enter:

```
line(0,1+i)
```

We get:

The line equation $y=x$ is plotted

We enter:

```
line(i,slope=2)
```

Or we enter:

```
DrawSlp(0,1,2)
```

We get:

```
The line equation y=2x+1 is plotted
```

We enter:

```
line(y-x=0)
```

We get:

```
The line equation y=x is plotted
```

Note: orientation of the line

- When the line is defined by two points, its orientation is defined by the order in which the points are supplied as argument. For example, `line(GA,GB)` defines a line oriented by the vector \overline{AB} .
- When the line is defined by an equation, we rewrite the equation in the form: "*left_member - right_member = 0*" to get a line equation of the form $a * x + by + c = 0$ and then the vector giving the orientation of the line is $[b, -a]$, or else its orientation is defined by the 3D cross product of its normal vector (third coordinate 0) and $[0,0,1]$. For instance, `line(y=2*x)` is orientated by $[1,2]$ because its equation is $-2 * x + y = 0$ and `cross([-2,1,0], [0,0,1])=[1,2,0]`.
- When the line is defined by a point A and its slope m , its orientation is defined by the vector \overline{AB} with $B = A + 1 + i * m$.

24.3.8 Segment: `Line`

`Line` takes as argument four real numbers giving the coordinates of two points.

`Line(a,b,c,d)` returns and plots the segment defined by the two points $a + i * b$ and $c + i * d$.

We enter:

```
Line(-1,1,2,-2)
```

We get:

```
The segment -1+i,2-2*i
```

24.3.9 Plot of a 2D horizontal line: `LineHorz`

`LineHorz` takes as argument an expression Xpr .

`LineHorz` plots the horizontal line $y = Xpr$.

We enter:

```
LineHorz(1)
```

We get:

```
the plot of the line y=1
```

24.3.10 Plot of a 2D vertical line: `LineVert`

`LineVert` takes as arguments an expression Xpr .

`LineVert` plots the the vertical line $x = Xpr$.

We enter:

```
LineVert(1)
```

We get:

the plot of the line $x=1$

24.3.11 Vector in plane geometry: `vector`

In plane geometry, `vector` takes as arguments:

- either two points GA and GB , or two complex numbers representing the affixes of these points, or two lists of the points coordinates.

`vector` defines and draws the vector \overrightarrow{GAGB}

- either a point GA (or a complex number representing the affix of this point or a list representing the coordinates of this point) and a vector \overrightarrow{GV} (recursive definition).

`vector` defines and draws the vector \overrightarrow{AB} such as $\overrightarrow{GAGB} = \overrightarrow{GV}$.

If $GW:=\text{vector}(GA, GV)$, so the point \overrightarrow{GB} such as $\overrightarrow{GAGB} = \overrightarrow{GV}$ is point $(GW[1, 1])$ or

$\text{point}(\text{coordinates}(GV)+\text{coordinates}(GA))$ or $GA+(\text{affix}(V)[1]-\text{affix}(GV)[0])$.

We enter:

```
vector(point(-1), point(i))
```

Or we enter:

```
vector(-1, i)
```

Or we enter:

```
vector([-1, 0], [0, 1])
```

We get:

The plot of the vector of origin -1 and end i

We enter:

```
GV:=vector(point(-1), point(i))
```

We enter:

```
vector(point(-1+i), GV)
```

Or we enter:

```
vector(-1+i, GV)
```

Or we enter:

```
vector([-1, 1], GV)
```

We enter:

```
point([-1, 1], coordinates(GV))
```

We get:

The plot of the vector of origin $-1+i$ and end $2*i$

We enter:

```
GD:=point([-1, 1]+coordinates(GV))
```

We get:

GD the point(2*i)

Note:

In symbolic computation, we work with the list of coordinates of the vectors that the we get thanks to the command `coordinates` (cf 24.8.6).

24.3.12 Median line of a triangle: `median_line`

`median_line(GA, GB, GC)` plots the median line of the triangle ABC through A .

We enter:

`median_line(0, 1, 2+i)`

We get in the geometry application:

The plot of the line passing by the point of affix 0 and through the point of affix $(3+i)/2$ (midpoint of the segment $(0, 2+i)$)

We enter in the CAS screen:

`median_line(0, 1, 2+i)`

We get:

`line(y=x/3)`

24.3.13 Parallel lines: `parallel`

`parallel(GA, GD)` plots the line parallel to the line D passing by A .

We enter:

`parallel(0, line(1, i))`

We get in the geometry application:

the plot of the line equation $y=-x$

We enter in the CAS screen:

`parallel(0, line(1, i))`

We get:

`line(y=(-x))`

24.3.14 Perpendicular bisector: `perpen_bisector`

`perpen_bisector(GA, GB)` plots the perpendicular bisector of the segment AB .

We enter:

`perpen_bisector(1, i)`

We get in the geometry application:

the plot of the line equation $y=x$

We enter in the CAS screen:

```
perpen_bisector(1,i)
```

We get:

```
line (y=x)
```

24.3.15 Line perpendicular to a line: perpendicular

`perpendicular(GA,GB,GC)` or `perpendicular(GA,line(GB,GC))` plots the line perpendicular to the line BC passing by the point A .

We enter:

```
perpendicular(1,1,2-i)
```

We get in the geometry application:

```
the plot of the line perpendicular to the line (1,2-i) and passing by
the point of affix 1
```

We enter in the CAS screen:

```
perpendicular(1,1,2-i)
```

We get:

```
line (y=(x-1))
```

24.3.16 Segment: segment

`segment (GA,GB)` plots the segment AB .

We enter:

```
segment (0,1+i)
```

We get in the geometry application:

```
The plot of the segment (0,1+i)
```

We enter in the CAS screen:

```
segment (0,1+i)
```

We get:

```
segment (point(0),point(1+i))
```

24.3.17 Tangent to a geometrical object or tangent to a curv in a point: tangent

`tangent` takes two arguments: a geometrical object and a point A .

- the geometrical object is the graph G of a function 2D
In this case, the second argument can be, either a real number x_0 , either a point A located on G .

For example, if we have defined the function g , we enter:

```
GG:=plotfunc(g(x),x)
```

```
tangent(GG, 1.2)
```

plots the tangent to the graph G of the function g at point of abscissa $x=1.2$, or we enter:

```
GA:=point(1.2+i*g(1.2))
tangent(GG, GA)
```

plots the tangent to point A of the graph G of the function g .

For instance, to get the plot of the tangent to the curve of $g(x) = x^2$ at point of abscissa $x_0 = 1$, we enter in the CAS:

```
g(x):=x^2
```

Then, in `Symb`, we enter:

```
GG:=plotfunc(g(x), x)
GT:=tangent(G, 1)
```

or we enter:

```
GT:=tangent(G, point(1+i))
```

We get

The tangent to the curve of $g(x) = x^2$ at point $1+i$

We get the equation of the tangent by entering in `Num`:

```
equation(GT)
```

- the geometrical object is not a graph

`tangent` may take as arguments:

- either a geometrical object G and a point A ,
- either a point A defined by `element` whose parameters are: a geometrical object G and a real representing the value of the parameter of the parametric equation of G .

`tangent` returns a list of lines and draws these lines which are the tan-gentes at this geometrical object G and which passent through the point A .

We enter:

```
tangent(circle(0,1), point(1+i))
```

We get:

The line equation $x=1$ and the line equation $y=1$

We enter:

```
tangent(element(circle(0,1), 1))
```

We get:

The tangent to the circle of center 0 and radius 1, at point of affix $\exp(i)$

We enter:

```
tangent(circle(i, 1+i), point((1+i*sqrt(3))*2))
```

We get:

2 tangents to the circle of center i and radius $\sqrt{2}$ through the
`point((1+i*sqrt(3))*2)`

24.3.18 Radical axis of two circles: `radical_axis`

The radical axis of two circles C_1 and C_2 is the locus of points which have the same power with respect to C_1 and at C_2 .

We enter:

```
radical_axis(circle(0,1+i),circle(1,1+i))
```

We get:

The plot of the line equation $x = 1/2$

Indeed: the line $x = 1/2$ is the perpendicular bisector of the segment $[0; 1]$

24.4 Polygon

24.4.1 Scalene triangle: `triangle`

In plane geometry, `triangle` takes as arguments: three points (or three complex numbers representing the affixes of these points, or else a list of three points or of three complex numbers). `triangle` returns and plots the triangle having for vertices these three points.

We enter:

```
triangle(-1,i,1+i)
```

We get:

The triangle of vertices $-1, i, 1+i$

24.4.2 Equilateral triangle: `equilateral_triangle`

In plane geometry, `equilateral_triangle`, takes two a or three arguments:

- two arguments: two points or two complex numbers representing the affixes of these points (or else a list of two points or of two complex numbers).
`equilateral_triangle(GA,GB)` returns and plots the equilateral triangle direct ABC but without defining the point C . We enter:

```
equilateral_triangle(0,2)
```

We get:

the equilateral triangle of vertices the points of affix
 $0, 2, 1+i\sqrt{3}$

To define the third vertex C , we can give the triangle a label (for example `GT:=equilateral_triangle(0,2)`) and use the command `vertices(GT)` which returns the list of vertices of T . Then, we will define `GC:=vertices(GT)[2]` but it is easier to add `GC`, name of the last vertex, as third argument.

- three arguments: the two previous arguments and as third argument the name of a variable to define and plot the third vertex with its label.

We enter:

```
equilateral_triangle(0,2,GC)
```

We get:

```
the equilateral triangle of vertices the points of affix
0,2,1+i*sqrt(3)
```

We enter:

```
normal(affix(GC))
```

We get:

```
1+i*sqrt(3)
```

24.4.3 Right triangle: `right_triangle`

In plane geometry, `right_triangle` takes three or four arguments:

- three arguments: two points A and B (or two complex numbers representing the affixes of these points) and a real k not null.

`right_triangle(GA,GB,k)` returns and plots the triangle ABC right angled in A : this triangle is direct if $k > 0$, indirect if $k < 0$ and is such as $AC = |k| * AB$.

Thus, if the angle $(\overrightarrow{BC}, \overrightarrow{BA}) = \beta$ radians (or degrees), we have $\tan(\beta) = k$.

We notice that if C is the transform of B in the similarity of center A of ratio $|k|$ and angle $(k/|k|) * \pi/2$.

We enter:

```
right_triangle(i,-i,2)
```

We get:

```
The right triangle of vertices i, -i, 4+i
```

We enter:

```
right_triangle(i,-i,-2)
```

We get:

```
The right triangle of vertices i, -i, -4+i
```

- four arguments: the three previous arguments and as last argument the name of a variable to define the third vertex.

We enter:

```
right_triangle(i,-i,2,GD)
```

We get:

```
The right triangle of vertices i, -i, 4+i
```

We enter:

```
normal(affix(GD))
```

We get:

```
4+i
```

24.4.4 Isosceles triangle: `isosceles_triangle`

In plane geometry, `isosceles_triangle` takes three or four arguments:

- three arguments: two points A and B (or two complex numbers representing the affixes of these points) and a real which designates the measure in radians (or in degrees) of the angle $(\overrightarrow{AB}, \overrightarrow{AC})$.

`isosceles_triangle(GA, GB, c)` returns and plots the isosceles triangle ABC of vertex A ($AB = AC$) and such as the angle $(\overrightarrow{AB}, \overrightarrow{AC}) = c$ radians (or degrees), without defining the point C .

We enter:

```
isosceles_triangle(i,1,-3*pi/4)
```

We get, if we have checked radian in the CAS configuration (Shift-CAS):

```
The isoscele triangle of vertices -1, i, -sqrt(2)+i
```

- four arguments: the three previous arguments and as fourth argument the name of a variable to define the third vertex.

We enter:

```
isosceles_triangle(i,1,-3*pi/4,GC)
```

We get, if we have checked radian in the CAS configuration (Shift-CAS):

```
The isoscele triangle of vertices -1, i, -sqrt(2)+i
```

We enter:

```
normal(affix(GC))
```

We get:

```
-sqrt(2)+i
```

24.4.5 Rhombus: `rhombus`

In plane geometry, `rhombus` takes three to five arguments:

- three arguments: two points or two complex numbers representing the affixes of these points and a real number a .

`rhombus(GA, GB, a)` returns and plots the rhombus $ABCD$ such as: $(\overrightarrow{AB}, \overrightarrow{AD}) = a$ radians (or degrees), but without defining the points C and D .

We enter:

```
rhombus(-2*i, sqrt(3)-i, pi/3)
```

We get, if we have checked `radian` in the CAS or Home configuration (Shift-CAS or Shift-Home)

```
The rhombus of vertices -2*i, sqrt(3)-i, sqrt(3)+i, 0
```

- four (resp. five) arguments: the three previous arguments, the last parameter (resp. the two last parameters) is (resp. are) the name(s) of a (resp. of the two) variable(s) which define(s) the penultimate vertex (resp. the two last vertices).

We enter:

```
rhombus(-2*i, sqrt(3)-i, pi/3, E, F)
```

We get, if we have checked `radian` in the CAS configuration:

The rhombus of vertices $-2*i, \sqrt{3}-i, \sqrt{3}+i, 0$

We enter:

```
normal(affix(E))
```

We get:

```
sqrt(3)+i
```

We enter:

```
normal(affix(F))
```

We get:

```
0
```

24.4.6 Rectangle: `rectangle`

In plane geometry, `rectangle` takes three to five arguments:

- three arguments: two points (or two complex numbers representing the affixes of these points) and a real number k not null.

`rectangle(GA, GB, k)` returns and plots the rectangle $ABCD$ such as:

$AD = |k| * AB$ and $(\overrightarrow{AB}, \overrightarrow{AD}) = (k/|k|) * \pi/2$ that is to say such as:

$\text{affix}(GD) = \text{affix}(GA) + k * \exp(i * \pi/2) * (\text{affix}(GB) - \text{affix}(GA))$

but without defining the points C and D .

Note If k is complex, we have:

$\text{affix}(GD) = \text{affix}(GA) + k * \exp(i * \pi/2) * (\text{affix}(GB) - \text{affix}(GA))$ and we can thus get the plot of a parallelogram.

We enter:

```
rectangle(0, 1+i, 1/2)
```

We get:

The rectangle of vertices $0, 1+i, 1/2+3*i/2, -1/2+i/2$

We enter:

```
rectangle(0, 1+i, -1/2)
```

We get:

The rectangle of vertices $0, 1+i, 3/2+i/2, 1/2-i/2$

We enter:

```
rectangle(0, 1, 1+i)
```

We get:

The parallelogram of vertices $0, 1, i, -1+i$ because $-1 + i = (1 + i) * \exp(i * \pi/2)$

- five arguments: the three previous arguments and the two last arguments are the names of two variables to define the two last vertices.

We enter:

```
rectangle(0,1+i,-1/2,GG,GH)
```

We get:

```
The rectangle of vertices 0,1+i,3/2+i/2,1/2-i/2
```

We enter:

```
normal(affix(GG))
```

We get:

```
(3+i)/2
```

We enter:

```
normal(affix(GH))
```

We get:

```
(1-i)/2
```

24.4.7 Square: square

In plane geometry, `square` takes one to four arguments:

- two arguments: two points or two complex numbers representing the affixes of these points (or else a list of two points or of two complex numbers).
`square(GA,GB)` returns and plots the square $ABCD$ of direct direction, but without defining the points D and C .

We enter:

```
square(0,1+i)
```

We get:

```
The square of vertices 0, 1+i, 2*i, -1+i
```

- three (resp. four) arguments: the two previous arguments followed by the the name of a (resp. two) variable(s) which define(s) the penultimate vertex (resp. the two other vertices).

We enter:

```
square(0,1+i,GC,GD)
```

We get:

```
The square of vertices 0, 1+i, 2*i, -1+i
```

We enter:

```
affix(GC)
```

We get:

```
2*i
```

We enter:

```
affix(GD)
```

We get:

```
-1+i
```

24.4.8 Quadrilateral: quadrilateral

In plane geometry, `quadrilateral(GA,GB,GC,GD)`, returns and plots the quadrilateral $ABCD$.

We enter:

```
quadrilateral(0,1,1+i,-1+2*i)
```

We get:

```
The "kite" of vertices 0, 1, 1+i, 1+2*i
```

24.4.9 Parallelogram: parallelogram

In plane geometry, `parallelogram` takes three arguments or four arguments:

- three arguments: three points (or three complex numbers representing the affixes of these points).

`parallelogram(GA,GB,GC)` returns and plots the parallelogram $ABCD$ such as:

$\overline{AD} = \overline{BC}$ but without defining the point D .

We enter:

```
parallelogram(0,1,2+i)
```

We get:

```
The parallelogram of vertices 0,1,2+i,1+i
```

We enter:

```
parallelogram(1,0,-1+i)
```

We get:

```
The parallelogram of vertices 1,0,-1+i,i
```

- four arguments: the three previous arguments and as fourth argument the name of a variable which defines the missing vertex.

We enter:

```
parallelogram(0,1,2+i,GF)
```

We get:

```
The parallelogram of vertices 0,1,2+i,1+i and the point F of affix
1+i
```

We enter:

```
normal(affix(GF))
```

We get:

1+i

24.4.10 Isopolygon: `isopolygon`

In plane geometry, `isopolygon` takes three arguments:

- either two points or two complex numbers and a positive integer k
- either two points or two complex numbers and a negative integer k .

When $k > 0$, `isopolygon` plots the direct regular polygon of k sides and consecutive vertices the two first arguments.

We enter:

```
isopolygon(0,1,4)
```

We get:

```
The square of vertices 0,1,1+i,i
```

When $k < 0$, `isopolygon` plots the direct regular polygon having $-k$ sides, as center the first argument, and as vertex the second argument.

We enter:

```
isopolygon(0,1,-4)
```

We get:

```
square of vertices 1,i,-1,-i
```

24.4.11 Hexagon: `hexagon`

See also: ?? for 2D geometry.

In plane geometry, `hexagon` may take from two to six arguments.

Description of the arguments:

- two arguments: two points or two complex numbers representing the affixes of these points (or else a list of two points or of two complex numbers).
`hexagon(A,B)` returns and plots the hexagon $ABCDEF$ of direct orientation, but without defining the points D,C,E and F .

We enter:

```
hexagon(0,1)
```

We get:

```
The hexagon of vertices
```

```
0,1,3/2+i*sqrt(3)/2,1+i*sqrt(3),i*sqrt(3),-1/2+i*sqrt(3)/2
```

- six arguments, the four last parameters are the name of two variables which define the two other vertices.

We enter:

```
hexagon(0,1,C,D,E,F)
```

We get:

```
The hexagon of vertices
```

```
0,1,3/2+i*sqrt(3)/2,1+i*sqrt(3),i*sqrt(3),-1/2+i*sqrt(3)/2
```

We enter:

```
affix(C)
```

We get:

```
3/2+i*sqrt(3)/2
```

We enter:

```
affix(D)
```

We get:

```
1+i*sqrt(3)
```

We enter:

```
affix(E)
```

We get:

```
i*sqrt(3)
```

We enter:

```
affix(F)
```

We get:

```
-1/2+i*sqrt(3)/2
```

24.4.12 Polygon: `polygon`

In plane geometry, `polygon` takes as argument the list (or the sequence) of n points or of n complex numbers representing the affixes of these points.

`polygon` returns and plots the polygon having for vertices these n points.

We enter:

```
polygon(-1, -1+i/2, i, 1+i, -i)
```

We get:

```
The polygon of vertices -1, -1+i/2, i, 1+i, -i
```

We enter:

```
polygon(makelist(x->exp(i*pi*x/3), 0, 5, 1))
```

We get:

```
The hexagon of vertices  $1, e^{\frac{i\pi}{3}}, e^{\frac{2i\pi}{3}}, \dots, e^{\frac{5i\pi}{3}}$ 
```

24.4.13 Polygonal line: `open_polygon`

In plane geometry, `open_polygon` takes as argument the list (or the sequence) of n points or of n complex numbers representing the affixes of these points.

`open_polygon` returns and plots the polygonal line having for vertices these n points.

We enter:

```
open_polygon(-1,-1+i/2,i,1+i,-i)
```

We get:

The polygonal line of vertices $-1, -1+i/2, i, 1+i, -i$

We enter:

```
open_polygon(makelist(x->exp(i*pi*x/3),0,5,1))
```

We get:

The polygonal line of vertices $1, e^{\frac{i\pi}{3}}, e^{\frac{2i\pi}{3}}, \dots, e^{\frac{5i\pi}{3}}$

24.4.14 Convex hull of points of the plan: `convexhull`

The instruction `convexhull` returns the convex hull of a ensemble of points of the plane supplied by two points or of affixes of points, elle returns a list of complex affixes of vertices of the envelop convexe. The algorithm used ist the scan of Graham. We can use `polygon` on the result of `convexhull` to get the plot of the convexe envelop.

We enter:

```
polygon(convexhull(0,1,1+i,1+2i,-1-i,1-3i,-2+i))
```

to get the convex hull of points of affixes $(0,0), (1,0), (1,1), (1,2), (-1,-1), (1,-3), (-2,1)$.

24.5 Curves

24.5.1 Circle and arcs: `circle`

`circle` takes one or two arguments to draw a circle, and four to six arguments to draw an arc of circle:

- with one argument:
the argument of `circle` is then the equation of the circle having as variables x and y ,
- with two arguments:
The first argument of `circle` is a point or a complex number considered as the affix of a point.
The second argument specifies which additional data is supplied to plot the circle: either the radius (as modulus of a complex number), either the diameter (specified by a point).
Then:
 - `circle(GC,r)` where GC is a point (or a complex number) and r a complex number, plots the circle of center C and radius the modulus of r .
This is useful, for example, to get the circle of center A passing by B .

We enter:

```
circle(GA,GB-GA) .
```

- `circle(GA,GB)` where A is a point or a complex number and B a point, plots the circle of diameter AB .

We enter:

```
circle(x^2+y^2-2*x-2*y)
```

We get:

The circle of center $1+i$ and radius $\sqrt{2}$ is drawn.

We enter:

```
circle(-1,i)
```

We get:

The circle of center -1 and radius 1 is drawn.

We enter:

```
circle(-1, point(i))
```

We get:

The circle of diameter $-1,i$

- With four to six arguments:

`circle` designates an arc of circle. In this case, the two first arguments determinate the circle which is the basis of the arc (see above) and the two following arguments are the angles at the center of the points which border the arc, and the two last arguments are the names of the variables storing the points which border the arc. The third and the fourth argument are the measures of the angles at the center of points which border the arc, these angles are measured in radians (or in degrees) starting from the axis defined by the two first arguments if the second argument is a point (definition of the circle by its diameter) or of the axis defined by its center C and the point $A = C + r$ if the second argument is a complex equal to r (definition of the circle by its centre and a complex whose modulus equals the radius). The fifth and the sixth argument are not mandatory and define the ends of the arc.

We enter:

```
circle(-1,1,0,pi/4,A,B)
```

We get, if we have checked radian in the CAS configuration:

The arc AB ($GA:=\text{point}(0)$ and $GB:=\text{point}(\frac{-1+\sqrt{2}+i*\sqrt{2}}{2})$) of the circle of center -1 and radius 1 is drawn.

Indeed, the angle is measured starting from the axis $(-1,0)$ and then the angle 0 is the point (0) .

We enter:

```
circle(-1,i,0,pi/4,A,B)
```

We get, if we have checked radian in the CAS configuration:

The arc AB ($GA:=\text{point}(-1+i)$ and $GB:=\text{point}(\frac{-1-\sqrt{2}+i*\sqrt{2}}{2})$) of the circle of center -1 and radius 1 is drawn.

Indeed, the angle is measured starting from the axis $(-1,i-1)$ and then the angle 0 is the point of affix $i-1$.

We enter:

```
circle(-1, point(i),0,pi/4,A,B)
```

We get:

The arc AB (GA:=point(i) and GB:=point($\frac{-1+i*(1+\sqrt{2})}{2}$)) the circle of diameter -1,i

Indeed, the angle is measured starting from the axis (-1,i) and then the angle 0 is the point of affix i.

24.5.2 Arcs of circle: arc ARC

See also: 24.5.1 for circles and arcs of circle.

arc takes three to five arguments: two points A,B (or two complex numbers a,b) and a real number α representing the measure of the arc AB in radians ($-2 * \pi \leq \alpha \leq 2 * \pi$). The fourth and the fifth arguments are not mandatory and are names of variables storing the center and the radius of the circle the arc is based on.

The arc AB is then based on the circle of centre: $(a + b)/2 + i * (b - a)/(2 * \tan(\alpha/2))$.

arc(A,B, α) is the arc where we see the segment AB from, along the angle $-\pi + \alpha/2$ if $2\pi > \alpha > 0$, or under the angle $\pi + \alpha/2$ if $-2\pi < \alpha < 0$.

To get the arc capable AB of measure β that is to say the arc of where the we see the segment AB from, along the angle $-\pi < \beta < \pi$, you have to enter:

arc(A,B,2*(-pi+ β)) if $\pi > \beta > 0$ or arc(A,B,2*(pi+ β)) if $-\pi < \beta < 0$.

Warning!

The sign of α gives the direction of the arc AB. For example, arc(A,B,3*pi/2) and arc(A,B,-pi/2) draw a full circle.

We enter:

```
arc(1,i,pi/2)
```

We get:

The arc (1,i) of the circle of center 0 and radius 1

We enter:

```
arc(1,i,pi/2,C,r)
```

We get:

The arc (1,i) of the circle of center C=point(0) and radius r=1

We enter:

```
arc(2,2*i,pi,C,r)
```

We get:

The half-circle of center C=point(1+i) and radius r=sqrt(2), starting from the point(2) to the point(2*i) in positive direction.

Note:

When circle has four arguments, circle also draws an arc of circle (cf. ??).

24.5.3 Circumcircle: circumcircle

circumcircle takes three parameters defining the vertices of a triangle.

circumcircle draws and returns the circumcircle of this triangle.

We enter:

```
circumcircle(-1,i,1+i)
```

We get:

```
Circumcircle of the triangle(-1,i,1+i)
```

24.5.4 Plot of a conic: conic

`conic` takes as argument the expression of a conic.

`conic` plots the conic having for equation argument=0.

We enter:

```
conic(2*x^2+2*x*y+2*y^2+6*x)
```

We get:

```
the plot of the ellipse of center -2+i and equation
2*x^2+2*x*y+2*y^2+6*x=0
```

Note:

Use `reduced_conic` to get the parametric equation of the conic.

We enter:

```
reduced_conic(2*x^2+2*x*y+2*y^2+6*x) [4]
```

We get:

```
[-2+i+(1+i)*(cos(t)+sqrt(3)*i*sin(t)),t,0,2π,2π/60]
```

24.5.5 Ellipse: ellipse

In plane geometry, `ellipse` takes one or three parameters:

- one parameter:
its equation of variables x and y . `ellipse(p(x,y))` plots the conic equation $p(x,y) = 0$ if $p(x,y)$ is a polynomial of degree 2.
- three parameters: the two foci and a point on the ellipse (or its affix if this affix is not real) or its two foci and a real (its half-major axis).
`ellipse(GF1,GF2,GA)` plots the ellipse passing by A and of foci $F1$ and $F2$ or,
`ellipse(GF1,GF2,a)` where a is a real number, plots the ellipse of foci $F1$ and $F2$ and half-major axis $|a|$.

We enter:

```
ellipse(-i,i,1+i)
```

We get:

```
The ellipse of foci -i, i and passing by 1+i
```

We enter:

```
ellipse(-i,i,sqrt(5)-1)
```

We get:

```
The ellipse of foci -i, i and half-major axis
```

```
sqrt(5)-1
```

We enter:

```
ellipse(x^2+2*y^2-1)
```

or we enter:

```
ellipse(sqrt(2)/2,-sqrt(2)/2,1)
```

We get:

```
The ellipse of center 0 and half-major axis 1 and foci sqrt(2)/2 and
-sqrt(2)/2
```

24.5.6 Excircle: excircle

`excircle` has three parameters defining the vertices of a triangle.
`excircle` draws and returns the excircle in the inner angle of the first vertex of this triangle.

We enter:

```
excircle(-1,i,1+i)
```

We get:

```
Excircle in the angle of vertex -1 the triangle(-1,i,1+i) is drawn.
```

24.5.7 Hyperbola: hyperbola

In plane geometry, `hyperbola` takes one or three parameters:

- one parameter:
its equation of variables x and y . `hyperbola(p(x,y))` plots the conic equation $p(x,y) = 0$ if $p(x,y)$ is a polynomial of degree 2.
- three parameters:
its two foci and one of these points (or its affix if this affix is not real) or its two foci and a real (its half-major axis).
`hyperbola(GF1,GF2,GA)` plots the hyperbola passing by A and of foci $F1$ and $F2$ or,
`hyperbola(GF1,GF2,a)` where a is a real number, plots the hyperbola of foci $F1$ and $F2$ and half-major axis $|a|$.

We enter:

```
hyperbola(-i,i,1+i)
```

We get:

```
The hyperbola of foci -i, i and passing by 1+i
```

We enter:

```
hyperbola(-i,i,1/2)
```

We get:

```
The hyperbola of foci -i, i and half-major axis 1/2
```

We enter:

```
hyperbola(x^2+2*y^2-1)
```

or we enter:

```
hyperbola(sqrt(6)/2,-sqrt(6)/2,1)
```

We get:

```
The hyperbola of center 0 and half-major axis 1 and foci sqrt(6)/2
and -sqrt(6)/2
```

24.5.8 Incircle: `incircle`

`incircle` has three parameters defining the vertices of a triangle.
`incircle` draws and returns the incircle of this triangle.

We enter:

```
incircle(-1,i,1+i)
```

We get:

```
Incircle of the triangle(-1,i,1+i)
```

24.5.9 Locus and envelope: `locus`

`locus` allows to plot the locus of a point which depends on another point to be defined with the function element.

`locus` also permits to plot the envelop of a line which depends on a point to be defined with the function element.

- locus of a point.

`locus` takes two to four arguments.

The two first argument are names of variables:

the first argument is the name of the point (for example B) whose we want to know the locus, this point being function of the second argument, the second argument is the name of the point (for example A) which follows the curve C and to be defined by `GA:=element(GC)`.

We can eventually specify as third argument the interval in which is the parameter used for the setting of C when the second described argument C and specify as fourth argument the value of $tstep$.

Note:

Use the command `parameq(C)` to know the setting of the curve C .

`locus` draws the locus of the first argument when the second argument moves as specified in the argument given to `element`.

Tip:

Put as few instructions as possible between the definition of M and the instruction `locus`.

To get the locus of the center of gravity B of the triangle of vertices $point(-1)$, $point(1)$ and A , when A follows the line of equation $y = 1$, we enter:

```
GA:=element(line(i,1+i))
GB:=isobarycenter(-1,1,GA)
GC:=locus(GB,GA)
```

We get:

```
The line parallel to the x axis passing by i/3
```

We enter in the Numeric view:

```
equation(GC)
```

We get:

```
equation(GC):y=1/3
```

- envelop of a line function of a point following a curve.
`locus` takes as arguments two names of variables: the first argument is the name of the line we want to know the envelop of, and this line is function of the second argument. The second argument is the name of the point which moves, to be defined with the function element.
`locus` draws the envelop of the first argument when the second argument moves according to what has been supplied as argument of element.
 To get the the envelop of the perpendicular bisector of FH when H follows the line of equation $x = 0$, we enter:

```
GF:=point(1)
GH:=element(line(x=0))
GD:=perpend_bisector(GF,GH)
locus(GD,GH)
```

We get:

The parabola of focus F and directrix line the y axis, whose equation is $2*x-y^2-1=0$

- envelop of a line supplied by an equation depending on a parameter. In this case, you have to specify that the parameter is the affix of a point of the line $y = 0$.
 For instance, envelop of a family of lines of equations $y + x \tan(t) - 2 \sin(t) = 0$ when $t \in \mathbb{R}$. (cf. 1)

We enter:

```
GH:=element(line(y=0));
GD:=line(y+x*tan(affix(M))-2*sin(affix(M)))
locus(GD,GH)
```

We get:

The astroid of parametric equation $2*\cos(t)^3+2*i*\sin(t)^3$

To get the envelop when $t = 0..pi$, we enter:

```
locus(GD,GH,t=0..pi)
```

We get:

The part above $y = 0$ of the astroid of parametric equation $2*\cos(t)^3+2*i*\sin(t)^3$

We can also look for the intersection of GD and GE (detailed below) to get the parametric equation of the locus.

```
GD:=y+x*tan(t)-2*sin(t)
GE:=diff(GD,t)
GM:=linsolve([GD=0,GE=0],[x,y])
GP:=plotparam(affix(simplify(GM)),t)
```

We get:

The astroid of parametric equation

$$2*\cos(t)^3+2*i*\sin(t)^3$$

indeed, `simplify(GM)` returns:
`[2*cos(t)^3,2*sin(t)^3]`

24.5.10 Parabola: `parabola`

In plane geometry, `parabola` takes one or two parameters:

- one parameter:
its equation of variables x and y . `parabola(p(x,y))` plots the conic equation $p(x,y) = 0$ if $p(x,y)$ is a polynomial of degree 2.
- two parameters:
two points (or their affixes if the second affix is not real), representing its focus and its vertex, or else a point (the vertex), or the affix of its vertex and a real number c .
`parabola(GF,GS)` returns and draws the parabola of focus F and vertex S .
`parabola(GS,c)` returns and draws the parabola of vertex $S = x_s + iy_s$ and equation $y = y_s + c * (x - x_s)^2$. You must know that if p is the parameter of the parabola, we have $FS = p/2$ and $c = 1/(2 * p)$.

We enter:

```
parabola (0,i)
```

We get:

The parabola of focus 0 and vertex i

We enter:

```
parabola (0,1)
```

We get:

The parabola of vertex 0 and equation $y = x^2$

We enter:

```
parabola (x^2-y-1)
```

or we enter:

```
parabola (-i,1)
```

or we enter:

```
parabola (i,-i)
```

We get:

The parabola of vertex $-i$ and focus i

24.5.11 Power of a point according to a circle: `powerpc`

If a point A is at a distance d of the center of a circle C of radius r , the power of A with respect to the circle C equals $d^2 - r^2$.

We enter:

```
powerpc(circle(0,1+i),3+i)
```

We get:

8

Indeed: $r = \sqrt{2}$ and $d = \sqrt{10}$ then $d^2 - r^2 = 8$

24.6 Transformation

24.6.1 Homothety: `homothety`

In plane geometry, `homothety` takes two or three arguments: a point (the center of the homothety), a real (the value of the ratio of the homothety) and eventually the geometrical object to be transformed. When `homothety` has two arguments, this function applies on a geometrical object.

We enter:

```
h:=homothety(i,2)
```

Then:

```
h(1+i)
```

We get:

The point $2+i$ plotted as a black cross (x)

When `homothety` has three arguments, `homothety` draws and returns the transform of the third argument in the homothety of center the first argument and ratio the second argument.

We enter:

```
homothety(i,2,1+i)
```

We get:

The point $2+i$ plotted as a black cross (x)

We enter:

```
homothety(i,2,circle(1+i,1))
```

We get:

The circle of center $2+i$ and radius 2

Note:

When the value of the homothety ratio is a non real complex number k , `homothety(GA,k)` is the similarity of center the point A, of ratio $\text{abs}(k)$ and angle $\text{arg}(k)$.

24.6.2 Inversion: `inversion`

In plane geometry, `inversion` takes two or three arguments: a point (the center of the inversion), a real (the value of the ratio of the inversion) and eventually the geometrical object to be transformed.

When `inversion` has two arguments, this function applies on a geometrical object.

If $\text{GF}:=\text{inversion}(\text{GC},k)$ and $\text{GB}:=\text{GF}(\text{GA})$, we have $\overline{\text{CA}} * \overline{\text{CB}} = k$.

We enter:

```
GF:=inversion(i,2)
```

Then:

```
GF(circle(1+i,1))
```

We get:

The vertical line of equation $x=1$

We enter:

```
GF(circle(1+i,1/2))
```

We get:

The circle of center $8/3+i$ and radius $4/3$ (passing by the point $4+i$)

When inversion has three arguments, inversion draws and returns the transform of the third argument in the inversion of center the first argument and ratio the second argument.

If $A1:=inversion(C,k,A)$ we have $CA * CA1 = k$.

We enter:

```
inversion(i,2,circle(1+i,1))
```

We get:

The vertical line of equation $x=1$

We enter:

```
inversion(i,2,circle(1+i,1/2))
```

We get:

The circle of center $8/3+i$ and radius $4/3$, passing by the point $4+i$

24.6.3 Orthogonale projection: projection

In plane geometry, `projection` takes one or two arguments: a geometrical object and eventually a point.

When `projection` has one argument, this function applies on a point and projects this point orthogonally on the geometrical object.

We enter:

```
p1:=projection(line(-1,i))
```

Then:

```
p1(1+i)
```

We get:

The point $1/2+3/2*i$ shows as a black cross (x)

We enter:

```
p2:=projection(circle(-1,1))
```

`p2(i)`

We get:

The point of affix, $\frac{\sqrt{2}}{2} + (i) \frac{\sqrt{2}}{2} - 1$, shows as a black cross
(x)

When projection has two arguments, projection draws and returns the transform of the point supplied as second argument by the orthogonal projection on the first argument.

We enter:

`projection(line(-1,i),1+i)`

We get:

The point $\frac{1}{2} + \frac{3}{2}i$ shows as a black cross (x)

We enter:

`projection(circle(-1,1),i)`

We get:

The point of affix, $-1 + \frac{\sqrt{2}}{2} + (i) \frac{\sqrt{2}}{2}$, shows as a black
cross (x)

24.6.4 Symmetry line and symmetry point: reflection

In plane geometry, `reflection` takes one or two arguments: a point or a line, and eventually the geometrical object to be transformed.

When `reflection` has one argument, this function applies on a geometrical object: when the first argument is a point (or a complex number), it is the symmetry with respect to this point (or with respect to point of affixe this complex number) and when the first argument is a line ,it is the symmetry with respect to this line.

We enter:

`sp:=reflection(-1)`

Then:

`sp(1+i)`

We get:

The point $-3 - i$ plotted as a black cross (x)

We enter:

`sd:=reflection(line(-1,i))`

Then:

`sd(1+i)`

We get:

The point $2i$ plotted as a black cross (x)

When `reflection` has two arguments, `reflection` draws and returns the transform of the second argument in the symmetry defined by the first argument: when the first argument is a point (or a complex number) it is the symmetry with respect to this point (or with respect to point of affix this complex number) and when the first argument is a line, it is the symmetry with respect to this line.

We enter:

```
reflection(-1,1+i)
```

We get:

```
The point -3-i plotted as a black cross (x)
```

We enter:

```
reflection(line(-1,i),1+i)
```

We get:

```
The point 2*i plotted as a black cross (x)
```

24.6.5 Rotation: `rotation`

In plane geometry, `rotation` takes two or three arguments.

When `rotation` has two arguments, these are: a point (the center of rotation) and a real (the measure of the rotation angle); this function applies on a geometrical object (point, line, etc., ...)

We enter:

```
r:=rotation(i,-pi/2)
```

Then:

```
r(1+i)
```

We get, if we have checked radian in the CAS configuration:

```
The point 0 plotted as a black cross (x)
```

When `rotation` has three arguments, these are: a point (the center of rotation), a real (the measure of the rotation angle) and the geometrical object to be transformed;

`rotation` draws and returns the transform of the third argument in the rotation of center the first argument and measure of rotation angle the second argument.

We enter:

```
rotation(i,-pi/2,1+i)
```

We get, if we have checked radian in the CAS configuration:

```
The point 0 plotted as a black cross (x)
```

We enter:

```
rotation(i,-pi/2,line(1+i,-1))
```

We get, if we have checked radian in the CAS configuration:

```
The line passing by 0 and -1+2*i
```

24.6.6 Similarity: `similarity`

In plane geometry, `similarity` takes three or four arguments: a point (the center of rotation), a real (the value of the ratio k of the similarity), a real (the measure a of the rotation angle in radians (or degrees)) and eventually the geometrical object to be transformed.

Note: if the ratio k is negative, the angle of the similarity is then of measure $-a$ radians (or degrees). When `similarity` has three arguments, this function applies on a geometrical object.

We enter:

```
GS:=similarity(i,2,-pi/2)
```

Then:

```
GS(1+i)
```

We get, if we have chosen radian in the CAS configuration:

```
The point -i plotted as a black cross (x)
```

We enter:

```
GS(circle(1+i,1))
```

We get, if we have chosen radian in the CAS configuration:

```
The circle of center -i and radius 2
```

When `similarity` has four arguments, `similarity` draws and returns the transform of the fourth argument in the similarity of center the first argument, of ratio the second argument and angle the third argument.

We enter:

```
similarity(i,2,-pi/2,1+i)
```

We get, if we have chosen radian in the CAS configuration:

```
The point -i plotted as a black cross (x)
```

We enter:

```
similarity(i,2,-pi/2,circle(1+i,1))
```

We get, if we have chosen radian in the CAS configuration:

```
The circle of center -i and radius 2
```

Note:

In 2D, the similarity of center the point GA , ratio k and angle a results in:
`similarity(GA, k, a)` or by `homothety(GA, k*exp(i*a))`.

24.6.7 Translation: `translation`

In plane geometry, `translation` takes one or two arguments: the vector of translation supplied by a geometrical vector, or by the list of its coordinates, or by its affix (difference between the coordinates of two points, or a complex number) and eventually the geometrical object to be transformed.

When `translation` has one argument, this function applies on a geometrical object.

We enter:

```
t:=translation(1+i)
```

Then:

```
t(-2)
```

We get:

The point $-1+i$ plotted as a black cross (x)

When `translation` has two arguments, `translation` draws and returns the transform of the second argument in the translation of vector the first argument.

We enter:

```
translation([1,1],-2)
```

Or we enter:

```
GA:=point(1);GB:=point(2+i);translation(vector(GA,GB),-2)
```

Or we enter:

```
translation(1+i,-2)
```

Or we enter:

```
GA:=point(1);GB:=point(2+i);translation(GB-GA,-2)
```

We get:

The point $-1+i$ plotted as a black cross (x)

We enter:

```
translation(1+i,line(-2,-i))
```

We get:

The line passing by $-1+i$ and 1

24.7 Measure and graphics

24.7.1 Measure of a angle: `angleat`

`angleat` takes as argument the name of three points and a point (or the affix of this point supplied as a complex number).

Warning! Take care that the three first arguments are names.

`angleat` returns the fourth point, calculates the measure (in radians or in degrees) of the oriented angle of vertex the first argument, the second argument is on the first side of the angle and the third argument on the second side, and this measure is displayed, along with a label, at the location of the fourth point.

Thus, `angleat(GA,GB,GC,GD)` designates the measure of the angle in radians (or in degrees) of $(\overline{AB}, \overline{AC})$ and this measure will be displayed, preceded by $\alpha A =$, at the location of point D.

We enter this command in the Symbolic view.

We enter:

```
GA:=point(-1);GB:=point(1+i);GC:=point(i);
```

```
segment (GA,GB); segment (GA,GC);
angleat (GA,GB,GC,0.2i)
```

We get, if we have checked radian in the CAS configuration (Shift-CAS):

```
 $\alpha A = \text{atan}(1/3)$  is displayed at the point(0.4i)
```

24.7.2 Measure of a angle: angleatraw

angleatraw takes as argument four points (or the affixes of these points supplied as four complex numbers).

angleatraw returns the fourth point, returns the measure (in radians or in degrees) of the oriented angle of vertex the first argument, the second point is on the first side of the angle, the third point on the second side and the measure is displayed, along with a label, close to the fourth point.

Thus, angleatraw(GA,GB,GC,GD) designates the measure of the angle in radians (or in degrees) of $(\overrightarrow{AB}, \overrightarrow{AC})$ and this measure will be displayed at the location of point D.

We enter:

```
GA:=point(-1);GB:=point(1+i);GC:=point(i);
segment (GA,GB); segment (GA,GC);
angleatraw (GA,GB,GC,0.2i)
```

We get, if we have checked radian in the CAS configuration (Shift-CAS):

```
atan(1/3) is displayed at the point(0.4i)
```

24.7.3 Display of the area of a polygon: areaat

areaat takes as arguments the name of a circle or of a polygon and a point (or the affix of a point supplied as a complex number).

areaat returns the point, the area of the circle or polygon and displays this area at the location of the point with a label.

Warning! Take care that the first argument is the name of a circle or of a polygon.

We enter:

```
t:=triangle(0,1,i)
areaat (t, (1+i)/2)
```

We get:

```
1/2 is displayed at the point(1+i)/2 with the label
```

We enter:

```
cc:=circle(0,2)
areaat (cc,2.2)
```

We get:

```
4*pi is displayed at the point(2.2) with a label
```

We enter:

```
c:=square(0,2)
areaat(c,2.2)
```

We get:

4 is displayed at the point(2.2) with a label

We enter:

```
h:=hexagon(0,1)
areaat(h,1.2)
```

We get:

$3\sqrt{3}/2$ is displayed at the point(1.2) with a label

24.7.4 Area of a polygon: `areaatraw`

`areaatraw` takes as arguments a circle or a polygon and a point (or the affix of a point supplied as a complex number).

`areaatraw` returns the point, the area of the circle or of the polygon and displays this area at the location of the point.

We enter:

```
areaatraw(triangle(0,1,i),(1+i)/2)
```

We get:

$1/2$ is displayed at the point(1+i)/2

We enter:

```
areaatraw(circle(0,2),2.2)
```

We get:

4π is displayed at the point(2.2)

We enter:

```
areaatraw(square(0,2),2.2)
```

We get:

4 is displayed at the point(2.2)

We enter:

```
areaatraw(hexagon(0,1),1.2)
```

We get:

$3\sqrt{3}/2$ is displayed at the point(1.2)

24.7.5 Length of a segment: `distanceat`

`distanceat` is a command which allows to display at a point the length of a segment with a label.

We enter this command in the Symbolic view.

`distanceat` takes three arguments: the name of two points and a point (or the affix of this point) or else the name of two geometrical objects and a point (or the affix of this point).

Warning! Take care that the two first arguments are names of point.

`distanceat` returns the point supplied in third argument, the length of the segment defined by the two first points or the distance between the two geometrical objects and displays this length at the location of the third point, preceded by a label.

We enter (we must give the name of the objects):

```
GA:=point(-1);GB:=point(1+i);
distanceat(GA,GB,0.4i)
```

We get:

"GAB=sqrt(5)" is displayed at the point(0.4i)

We enter (you must give the name of the objects):

```
GC:=point(0);GD:=line(-1,1+i)
distanceat(GC,GD,i/2)
```

We get:

"GCD=sqrt(5)/5" is displayed at the point(i/2)

We enter (we must give the name of the objects):

```
GK:=circle(0,1);GL:=line(-2,1+3i)
distanceat(GK,GL,0)
```

We get:

"GKL=sqrt(2)-1" is displayed at the point(0)

24.7.6 Length of a segment: `distanceatraw`

`distanceatraw` is a command which allows to display at a point the length of a segment, but with no label.

We enter this command in the Symbolic view.

`distanceatraw` takes as argument three points (or two points and the affix of a point supplied as a complex number) or else two geometrical objects and a point (or the affix of this point).

`distanceatraw` returns the point supplied in third argument, the length of the segment defined by the two first points, or the distance between the two geometrical objects, and displays this length at the location of the third point.

We enter:

```
GA:=point(-1);GB:=point(1+i);
distanceatraw(GA,GB,0.4i)
```

Or we enter directly:

```
distanceatraw(point(-1),point(1+i),0.4i)
```

We get:

`sqrt(5)` is displayed at the point(0.4i)

We enter:

```
GC:=point(0);GD:=line(-1,1+i)
distanceatraw(GC,GD,i/2)
```

Or we enter directly:

```
distanceatraw(point(0),line(-1,1+i),0.4i)
```

We get:

`sqrt(5)/5` is displayed at the point(i/2)

We enter:

```
GK:=circle(0,1); GL:=line(-2,1+3i)
distanceatraw(GK,GL,0)
```

Or we enter directly:

```
distanceatraw(circle(0,1),line(-2,1+3i),0.4i)
```

We get:

`sqrt(2)-1` is displayed at the point(0)

24.7.7 Perimeter of a polygon: `perimeterat`

`perimeterat` takes as argument the name of a circle or of a polygon and a point (or the affix of a point supplied as a complex number).

`perimeterat` returns the point, the perimeter of the circle or of the polygon, and displays this perimeter at the location of the point with a label.

We enter this command in the Symbolic view.

Warning! Take care that the first argument is the name of a circle or of a polygon.

We enter:

```
t:=triangle(0,1,i)
perimeterat(t,(1+i)/2)
```

We get:

`2+sqrt(2)` is displayed at the point((1+i)/2) with a label

We enter:

```
c:=square(0,2)
perimeterat(c,2.2)
```

We get:

`8` is displayed at the point(2.2) with a label

We enter:

```
cc:=circle(0,2)
perimeterat(cc,2.2)
```

We get:

4π is displayed at the point(2.2) with a label

We enter:

```
h:=hexagon(0,1)
perimeterat(h,1.2)
```

We get:

6 is displayed at the point(1.2) with a label

24.7.8 Perimeter of a polygon: `perimeteratraw`

`perimeteratraw` takes as argument a circle or a polygon, and a point (or the affix of a point supplied as a complex number).

`perimeteratraw` returns the point, the perimeter of the circle or of the polygon, and displays this perimeter at the location of the point.

We enter this command in the Symbolic view.

We enter:

```
perimeteratraw(triangle(0,1,i),(1+i)/2)
```

We get:

$2+\sqrt{2}$ is displayed at the point $((1+i)/2)$

We enter:

```
perimeteratraw(circle(0,2),2.2)
```

We get:

4π is displayed at the point(2.2)

We enter:

```
perimeteratraw(hexagon(0,1),1.2)
```

We get:

6 is displayed at the point(1.2)

We enter:

```
perimeteratraw(square(0,2),2.2)
```

We get:

8 is displayed at the point(2.2)

24.7.9 Slope of a line: `slopeat`

`slopeat` is a command which allows to display at a point the slope of a line, or of a segment, with a label.

We enter this command in the Symbolic view.

`slopeat` takes two arguments: the name of a line (or of a segment), and a point (or the affix of a point supplied as a complex number).

`slopeat` returns the point, the slope of the line (or of the segment) and displays this slope at the location of the point, with a label.

Warning! Take care that the first argument is the name of a line or of a segment.

We enter:

```
GD:=line(1,2i)
```

Or we enter:

```
GD:= segment (1,2i),i)
slopeat(GD,i)
```

We get:

```
"sD=-2" is displayed at the point(i)
```

We enter:

```
GP:=line(2y-x=3),2*i)
slopeat(GP,2*i)
```

We get:

```
"sP=1/2" is displayed at the point(2*i)
```

We enter:

```
GT:=tangent(plotfunc(sin(x)),pi/4)
```

Or we enter:

```
GT:=LineTan(sin(x),pi/4)
```

Then:

```
slopeat(GT,i)
```

We get:

```
"sT=(sqrt(2))/2" is displayed at the point(i)
```

24.7.10 Slope of a line: `slopeatraw`

`slopeatraw` is a command which allows to display at a point the slope of a line or of a segment but with no label.

We enter this command in the Symbolic view.

`slopeatraw` takes two arguments: a line (or a segment) and a point (or the affix of a point supplied as a complex number).

`slopeatraw` returns the point, the slope of the line (or of the segment), and displays this slope at the location of the point.

We enter:

```
GD:=line(1,2i)
slopeatraw(GD,i)
```

Or we enter directly:

```
slopeatraw(line(1,2i),i)
```

We get:

```
-2 is displayed at the point(i)
```

We enter:

```
GE:= segment (1,2i),i)
slopeatraw(GE,1)
```

Or we enter directly:

```
slopeatraw(segment (1,2i),1)
```

We get:

```
-2 is displayed at the point(1)
```

We enter:

```
GP:=line(2y-x=3,2*i)
slopeatraw(GP,2*i)
```

Or we enter directly:

```
slopeatraw(line(2y-x=3,2*i),2*i)
```

We get:

```
1/2 is displayed at the point(2*i)
```

We enter:

```
GT:=tangent(plotfunc(sin(x)),pi/4)
slopeat(GT,i)
```

Or we enter directly:

```
slopeatraw(tangent(plotfunc(sin(x)),pi/4),i)
```

We get:

```
(sqrt(2))/2 is displayed at the point(i)
```

24.8 Measure

24.8.1 Abscissa of a point or of a vector: `abscissa`

In plane geometry, `abscissa` takes as argument a point, a vector, or a complex number.

`abscissa` returns the abscissa of the point or of the vector:

- if the point A is of cartesian coordinates (x_A, y_A) , `abscissa(GA)` returns x_A ,

- if the point B is of cartesian coordinates (x_B, y_B) , `abscissa(GA-GB)` returns $x_A - x_B$ (because `GA-GB` designates the vector \overrightarrow{BA}).

We enter:

```
abscissa(point(1+2*i))
```

We get:

1

We enter:

```
abscissa(point(i)-point(1+2*i))
```

We get:

-1

We enter:

```
abscissa(1+2*i)
```

We get:

1

We enter:

```
abscissa([1,2])
```

We get:

1

24.8.2 Affix of a point or of a vector: `affix`

`affix` takes as argument a point, a vector, or the coordinates of a point or of a 2D vector.

`affix` returns the affix of the point or of the vector:

- if the point A is of cartesian coordinates (x_A, y_A) , `affix(GA)` returns $x_A + i * y_A$

- if the point B is of cartesian coordinates (x_B, y_B) , `affix(GA-GB)` or `affix(vector(GB, GA))` returns $x_A - x_B + i * (y_A - y_B)$ (because `GA-GB` designates the vector \overrightarrow{BA} and `coordinates(vector(GB, GA))` returns $[x_A + i * y_A, x_B + i * y_B]$).

We enter:

```
affix(point(i))
```

We get:

i

We enter:

```
affix(point(i)-point(1+2*i))
```

We get:

```
-1-i
```

24.8.3 Measure of a angle: angle

`angle` takes as argument three points (or the affixes of these points supplied as three complex numbers) and eventually a string used as label along with the symbol of an arc of circle which represents the angle on the figure (the arc of circle is replaced by the symbol of the half of a square in the case of the angle equals $\pi/2$ or $-\pi/2$).

`angle` returns the measure (in radians or in degrees) of the oriented angle of vertex the first argument, the second argument is on the first side of the angle and the third argument is on the second side.

Then:

`angle(GA, GB, GC)` designates the measure of the angle in radians (or in degrees) of $(\overrightarrow{AB}, \overrightarrow{AC})$.

`angle(GA, GB, GC, "")` plots the angle $(\overrightarrow{AB}, \overrightarrow{AC})$ with as label a small oriented arc.

`angle(GA, GB, GC, "a")` plots the angle $(\overrightarrow{AB}, \overrightarrow{AC})$ with as label a small oriented arc written a.

`angle(GA, GB, GC, "") [0]` or `angle(GA, GB, GC, "a") [0]` designates the measure of the angle in radians (or in degrees) of $(\overrightarrow{AB}, \overrightarrow{AC})$.

We enter:

```
angle(0,1,1+i)
```

We get, if we have chosen radian in the CAS configuration:

```
pi/4
```

We enter:

```
angle(0,1,1+i,"")
```

We get, if we have checked radian in the CAS configuration:

```
[pi/4,circle(point(0,0),1/5)] and the angle is designated by an arc
of circle without label.
```

We enter:

```
angle(0,1,1+i,"a")
```

We get, if we have checked radian in the CAS configuration:

```
[pi/4,circle(point(0,0),1/5)] and the angle is deisgnated by an arc
of circle with as label.
```

We enter:

```
angle(0,1,i,"a")
```

We get, if we have checked radian in the CAS configuration:

```
[pi/2,polygon(point(1/5,0),point(1/5,1/5),point(0,1/5),point(0,1/5))]
and the right angle is designated by an half of square with the label
a.
```

24.8.4 Length of an arc of curve: `arcLen`

`arcLen` takes one or four parameters.

Warning! Take care to not be in complex mode.

- the parameter is either a circle or an arc of circle, either a polygon.

We enter:

```
arcLen(circle(0,1,0,pi/2))
```

We get:

```
pi/4
```

We enter:

```
arcLen(hexagon(0,1))
```

We get:

```
6
```

- the four parameters are: an expression *expr* (resp. a list of two expressions [*expr1*, *expr2*]), the name of a parameter and two values *a* and *b* of this parameter.

`arcLen` returns the length of the arc of curve defined by the equation $y = f(x) = \text{expr}$ (resp. by $x = \text{expr1}, y = \text{expr2}$) for the values of the parameter between *a* and *b*.

We have then `arcLen(f(x), x, a, b) =:`

```
integrate(sqrt(diff(f(x),x)^2+1),x,a,b)
```

or

```
integrate(sqrt(diff(x(t),t)^2+diff(y(t),t)^2),t,a,b).
```

Examples

- Calculate the length of the arc of circle *AB* (with $A = (0,0)$ and $B = (0,1)$) and angle at center $\pi/2$.

We enter:

```
arcLen(arc(0,1,pi/2))
```

We get:

```
sqrt(2)*pi/4
```

- Calculate the perimeter of the triangle *ABC* (with $A = (0,0)$, $B = (0,1)$ and $C = (1,1)$).

We enter:

```
arcLen(triangle(0,1,1+i))
```

We get:

```
sqrt(2)+2
```

- Calculate the length of the arc of parabola $y = x^2$ to x from 0 to $x = 1$.

We enter:

```
arcLen(x^2,x,0,1)
```

or

```
arcLen([t,t^2],t,0,1)
```

We get:

```
(sqrt(5))/2-ln(sqrt(5)-2)/4
```

- Calculate the length of the arc of the curve $y = \cosh(x)$ for x from 0 to $x = \ln(2)$.

We enter:

```
arcLen(cosh(x),x,0,log(2))
```

We get:

```
3/4
```

- Calculate the length of the arc of circle $x = \cos(t), y = \sin(t)$ for t from 0 to $t = 2 * \pi$.

We enter:

```
arcLen([cos(t),sin(t)],t,0,2*pi)
```

We get:

```
2*pi
```

24.8.5 Area of a polygon: `area`

`area` returns the area of a circle or of a polygon.

We enter:

```
area(triangle(0,1,i))
```

We get:

```
1/2
```

We enter:

```
area(square(0,2))
```

We get:

```
4
```

24.8.6 Coordinates of a point, a vector or a line: `coordinates`

In plane geometry, `coordinates` takes as argument a point, a complex number, a vector or a line. `coordinates` returns the list of the abscissa and the ordinate of the point, or the vector, or the list of affixes of two points of the oriented line.

- if the point A is of cartesian coordinates (x_A, y_A) , `coordinates(GA)` returns $[x_A, y_A]$,
- if the point B is of cartesian coordinates (x_B, y_B) , `coordinates(vector(GA,GB))` or `coordinates(GB-GA)` returns $[x_B - x_A, y_B - y_A]$ (whereas `B-A` returns $(x_B - x_A) + i * (y_B - y_A)$ because `B-A` designates the affix of the vector AB in plane geometry),
- if the vector v is of cartesian coordinates (x_V, y_V) , `coordinates(GV)` or `coordinates(vector(GA,GV))` returns $[x_V, y_V]$,
- if a line D is defined by two points A and B, `coordinates(GD)` returns $[affix(GA), affix(GB)]$. If D is defined by its equation, `coordinates(GD)` returns

$[\text{affix}(GA), \text{affix}(GB)]$ where A and B are two points of the line D, the vector \overrightarrow{AB} having same orientation as d.

We enter:

`coordinates(point(1+2*i))`

Or we enter:

`coordinates(1+2*i)`

We get:

`[1,2]`

We enter:

`coordinates(point(1+2*i)-point(i))`

Or we enter:

`coordinates(point(1+2*i)-point(i))`

We get:

`[1,1]`

We enter:

`coordinates(vector(point(i),point(1+2*i)))`

Or we enter:

`coordinates(vector(i,1+2*i))`

Or we enter:

`coordinates(vector([0,1],[1,2]))`

We get:

`[1,1]`

We enter:

`coordinates(1+2*i)`

Or we enter:

`coordinates(vector(1+2*i))`

Or we enter:

`coordinates(vector(point(i),vector(1+2*i)))`

We get:

`[1,2]`

We enter:

```
coordinates (point (i), vector (1+2*i))
```

We get:

```
[1, 2]
```

We enter:

```
d:=line (-1+i, 1+2*i)
```

Or we enter

```
d:=line (point (-1, 1), point (1, 2))
```

Then,

```
coordinates (d)
```

We get:

```
[-1+i, 1+2*i]
```

We enter:

```
d:=line (y=(1/2*x+3/2))
```

We get:

```
[(3*i)/2, 1+2*i]
```

We enter:

```
d:=line (x-2*y+3=0)
```

We get:

```
[(3*i)/2, (-4+i)/2]
```

Warning!

`coordinates` might also take as argument a sequence or a list of points. Then, `coordinates` returns the sequence or the list of lists of coordinates of these points, for example:

```
coordinates (i, 1+2*i) or coordinates (point (i), point (1+2*i))
```

returns the sequence:

```
[0, 1], [1, 2]
```

and

```
coordinates ([i, 1+2*i]) or coordinates ([point (i), point (1+2*i)])
```

returns the matrix:

```
[[0, 1], [1, 2]] so coordinates ([1, 2]) returns the matrix:
```

```
[[1, 0], [2, 0]] because [1, 2] is considered as the list of two points of affix 1 and 2.
```

24.8.7 Rectangular coordinates of a point: `rectangular_coordinates`

`rectangular_coordinates` returns the list of the abscissa and the ordinate of a point supplied by the list of its polar coordinates.

We enter:

```
rectangular_coordinates (2, pi/4)
```

Or we enter:

```
rectangular_coordinates(polar_point(2,pi/4))
```

We get:

```
[2/(sqrt(2)),2/(sqrt(2))]
```

24.8.8 Polar coordinates of a point: `polar_coordinates`

`polar_coordinates` returns the list of the modulus and the argument of the affixe of a point, of a complex number, or of the list of rectangular coordinates.

We enter:

```
polar_coordinates(1+i)
```

Or we enter:

```
polar_coordinates(point(1+i))
```

Or we enter:

```
polar_coordinates([1,1])
```

We get:

```
[sqrt(2),pi/4]
```

24.8.9 Length of a segment and distance between two geometrical objects: `distance`

`distance` takes as argument two points (or the affixes of these points supplied as two complex numbers) or two geometrical objects.

`distance` returns the length of the segment defined by these two points or the distance between the two geometrical objects.

We enter:

```
distance(-1,1+i)
```

We get:

```
sqrt(5)
```

We enter:

```
distance(0,line(-1,1+i))
```

We get:

```
sqrt(5)/5
```

We enter:

```
distance(circle(0,1),line(-2,1+3i))
```

We get:

```
sqrt(2)-1
```

24.8.10 Square of the length of a segment: `distance2`

`distance2` takes as argument two points (or two points and a the affix of a point supplied as a complex number).

`distance2` returns the square of the length of the segment defined by these two points.

We enter:

```
distance2 (-1, 1+i)
```

We get:

5

24.8.11 Cartesian equation of a geometrical object: `equation`

`equation` allows to get the cartesian equation of a geometrical object.

Warning! Prior to use `equation`, take care of purging the variables `x` and `y` by entering `purge(x)` and `purge(y)` or `x:='x'` and `y:='y'`.

We enter:

```
equation(line(point(0,1,0),point(1,2,3)))
```

We get:

$(x-y+1=0, 3x+3y-2z=0)$

We enter:

```
equation(sphere(point(0,1,0),2))
```

We get:

$x^2+y^2+-2y+z^2-3=0$

which is the equation of the sphere of center $(0,1,0)$ and radius 2.

24.8.12 Get as answer the value of a measure displayed:

`extract_measure`

`extract_measure` allows to get the value of a measure which has been displayed.

`extract_measure` takes as argument the command which previously displayed this measure.

We enter:

```
GA:=point(-1);GB:=point(1+i);GC:=segment(GA,GB)
```

```
extract_measure(distanceat(GA,GB,i))
```

We get:

`sqrt(5)`

We enter:

```
extract_measure(distanceatraw(GA,GB,i))
```

We get:

```
sqrt(5)
```

We enter:

```
extract_measure(slopeat(GC,i))
```

We get:

$$1/2$$

We enter:

```
extract_measure(slopeatraw(GC,i))
```

We get:

$$1/2$$

24.8.13 Ordinate of a point or of a vector: `ordinate`

In plane geometry, `ordinate` takes as argument a point, a vector, or a complex number.

`ordinate` returns the ordinate of the point or of the vector:

- if the point A is of cartesian coordinates (x_A, y_A) , `ordinate(GA)` returns y_A ,
- if the point B is of cartesian coordinates (x_B, y_B) , `ordinate(GA-GB)` returns $y_A - y_B$ ($A-B$ designates the vector \overrightarrow{BA}).

We enter:

```
ordinate(point(1+2*i))
```

We get:

$$2$$

We enter:

```
ordinate(point(i)-point(1+2*i))
```

We get:

$$-1$$

We enter:

```
ordinate(1+2*i)
```

We get:

$$2$$

We enter:

```
ordinate([1,2])
```

We get:

$$2$$

24.8.14 Parametric equation of a geometrical object: `parameq`

In plane geometry, `parameq` allows to get the parametric equation of a geometrical object in the form of the complex number $x(t) + i * y(t)$.

Warning! Prior to use `parameq`, take care of purging the variable `t` by entering: `purge(t)` or `t:='t'`.

We enter:

```
parameq(line(-1,i))
```

We get:

```
-t+(1-t)*(i)
```

We enter:

```
parameq(circle(-1,i))
```

We get:

```
-1+exp(i*t)
```

We enter:

```
normal(parameq(ellipse(-1,1,i)))
```

We get:

```
sqrt(2)*cos(t)+(i)*sin(t)
```

24.8.15 Perimeter of a polygon: `perimeter`

`perimeter` returns the perimeter of a circle or of a polygon. See also the command `arcLen`.

We enter:

```
perimeter(triangle(0,1,i))
```

We get:

```
2+sqrt(2)
```

We enter:

```
perimeter(square(0,2))
```

We get:

```
8
```

24.8.16 Radius of a circle: `radius`

`radius` takes as argument a circle.

`radius` returns the length of the radius of this circle.

We enter:

```
radius(circle(-1,i))
```

We get:

$$1$$

We enter:

$$\text{radius}(\text{circle}(-1, \text{point}(i)))$$

We get:

$$\sqrt{2}/2$$

24.8.17 Slope of a line: slope

`slope` is either a command, either a parameter of the command line (See 24.3.7)

When `slope` is a command, it takes argument a line, a segment, two points or two complex numbers.

`slope` returns the slope of the line defined by the segment, the two points, or their affixes.

We enter:

$$\text{slope}(\text{line}(1, 2i))$$

Or we enter:

$$\text{slope}(\text{segment}(1, 2i))$$

Or we enter:

$$\text{slope}(\text{point}(1), \text{point}(2i))$$

Or we enter:

$$\text{slope}(1, 2i)$$

We get:

$$-2$$

We enter:

$$\text{slope}(\text{line}(2y-x=3))$$

We get:

$$1/2$$

We enter:

$$\text{slope}(\text{tangent}(\text{plotfunc}(\sin(x)), \pi/4))$$

Or we enter:

$$\text{slope}(\text{LineTan}(\sin(x), \pi/4))$$

We get:

$$(\sqrt{2})/2$$

24.9 Test

24.9.1 Check whether three points are collinear: `is_collinear`

`is_collinear` is a boolean function and takes as argument a list or a sequence of points.
`is_collinear` equals 1 if the points are collinear, 0 otherwise.

We enter:

```
is_collinear(0,1+i,-1-i)
```

We get:

1

We enter:

```
is_collinear(i/100,1+i,-1-i)
```

We get:

0

24.9.2 Check whether four points are concyclic: `is_concyclic`

`is_concyclic` is a boolean function and takes as argument a list or a sequence of points.
`is_concyclic` equals 1 if the points are concyclic, 0 otherwise.

We enter:

```
is_concyclic(1+i,-1+i,-1-i,1-i)
```

We get:

1

We enter:

```
is_concyclic(i,-1+i,-1-i,1-i)
```

We get:

0

24.9.3 Check whether elements are conjugates: `is_conjugate`

`is_conjugate` allows to know if four points are conjugates, or if two points, two lines, or a line and a point are conjugates for a circle or for two lines.

`is_conjugate` is a boolean function and takes as arguments two points (resp. two lines, or a circle) followed by two points, two lines, or a line and a point.

`is_conjugate` equals 1 if the arguments are conjugates, 0 otherwise.

We enter:

```
is_conjugate(circle(0,1+i),point(1-i),point(3+i))
```

We get:

1

We enter:

```
is_conjugate(circle(0,1),point((1+i)/2),line(1+i,2))
```

Or we enter:

```
is_conjugate(circle(0,1),line(1+i,2),point((1+i)/2))
```

We get:

1

We enter:

```
is_conjugate(circle(0,1),line(1+i,2),line((1+i)/2,0))
```

We get:

1

We enter:

```
is_conjugate(point(1+i),point(3+i),point(i),point(i+3/2))
```

We get:

1

We enter:

```
is_conjugate(line(0,1+i),line(2,3+i),line(3,4+i),line(3/2,5/2+i))
```

We get:

1

24.9.4 Check whether points or/and lines are coplanar: `is_coplanar`

`is_coplanar` checks whether a list, or a sequence of points, or of lines are coplanar.

We enter:

```
is_coplanar([0,0,0],[1,2,-3],[1,1,-2],[2,1,-3])
```

We get:

1

We enter:

```
is_coplanar([-1,2,0],[1,2,-3],[1,1,-2],[2,1,-3])
```

We get:

0

We enter:

```
is_coplanar([0,0,0],[1,2,-3],line([1,1,-2],[2,1,-3]))
```

We get:

1

We enter:

```
is_coplanar(line([0,0,0],[1,2,-3]),line([1,1,-2],[2,1,-3]))
```

We get:

1

We enter:

```
is_coplanar(line([-1,2,0],[1,2,-3]),line([1,1,-2],[2,1,-3]))
```

We get:

0

24.9.5 Check whether a point is on a geometrical object: `is_element`

`is_element` is a boolean function and takes as argument a point and a geometrical object. `is_element` equals 1 if the point is on the geometrical object, 0 otherwise.

We enter:

```
is_element(point(-1-i),line(0,1+i))
```

We get:

1

We enter:

```
is_element(point(i),line(0,1+i))
```

We get:

0

24.9.6 Check whether a triangle is equilateral: `is_equilateral`

`is_equilateral` is a boolean function and takes as argument three points or a geometrical object. `is_equilateral` equals 1 if the three points form an equilateral triangle, or if the geometrical object is an equilateral triangle, 0 otherwise.

We enter:

```
is_equilateral(0,2,1+i*sqrt(3))
```

We get:

1

We enter:

```
GT:=equilateral_triangle(0,2,GC);is_equilateral(GT[0])
```

We get:

1

Indeed, `GT[0]` designates a triangle because `GT` is a list composed of the triangle and its vertex `GC`.

We enter `affix(GC)` and we get `1+i*sqrt(3)`

We enter:

```
is_equilateral(1+i,-1+i,-1-i)
```

We get:

0

24.9.7 Check whether a triangle is isosceles: `is_isosceles`

`is_isosceles` is a boolean function and takes as argument three points or a geometrical object. `is_isosceles` equals 1 (resp. 2, 3) if the three points form an isosceles triangle or if the geometrical object is an isosceles triangle whose angle between the two equal sides is designated by the first (resp. second, third) argument, or equals 4 if the three points form an equilateral triangle, or if the geometrical object is an equilateral triangle, 0 otherwise.

We enter:

```
is_isosceles(1,1+i,i)
```

We get:

2

We enter:

```
GT:=isosceles_triangle(0,1,pi/4);is_isosceles(GT)
```

We get:

1

We enter:

```
GT:=isosceles_triangle(0,1,pi/4,GC);is_isosceles(GT[0])
```

We get:

1

Indeed, `GT[0]` designates a triangle because `GT` is a list composed of the triangle and its vertex `C`.

We enter `affix(GC)` and we get `(sqrt(2))/2+((i)*sqrt(2))/2`

We enter:

```
is_isosceles(1+i,-1+i,-i)
```

We get:

3

24.9.8 Orthogonality of two lines or two circles: `is_orthogonal`

`is_orthogonal` is a boolean function and takes as argument two lines or two circles.

`is_orthogonal` equals 1 if the two lines or the two circles (i.e if the tangents at their cross points are orthogonal), 0 otherwise.

We enter:

```
is_orthogonal(line(1,i), line(0,1+i))
```

We get:

1

We enter:

```
is_orthogonal(line(2,i), line(0,1+i))
```

We get:

0

We enter:

```
is_orthogonal(circle(0,1), circle(sqrt(2),1))
```

We get:

1

We enter:

```
is_orthogonal(circle(0,1), circle(2,1))
```

We get:

0

24.9.9 Check whether two lines are parallel: `is_parallel`

In plane geometry, `is_parallel` is a boolean function and takes as argument two lines. `is_parallel` equals 1 if the two lines are parallel, 0 otherwise.

We enter:

```
is_parallel(line(0,1+i), line(i,-1))
```

We get:

1

We enter:

```
is_parallel(line(0,1+i), line(i,-1-i))
```

We get:

0

24.9.10 Check whether a polygon is a parallelogram: `is_parallelogram`

`is_parallelogram` is a boolean function and takes as argument four points or a geometrical object.

`is_parallelogram` equals 1 (resp. 2, 3, 4) if the four points form a parallelogram (resp. a rhombus, a rectangle, a square) or if the geometrical object is a parallelogram (resp. a rhombus, a rectangle, a square), 0 otherwise.

We enter:

```
is_parallelogram(i, -1+i, -1-i, 1-i)
```

We get:

0

We enter:

```
is_parallelogram(1+i, -1+i, -1-i, 1-i)
```

We get:

1

We enter:

```
GQ:=quadrilateral(1+i, -1+i, -1-i, 1-i); is_parallelogram(GQ)
```

We get:

4

Warning!

We must enter:

```
GP:=parallelogram(-1-i, 1-i, i, GD); is_parallelogram(GP[0])
```

To get:

1

Indeed, it is `GP[0]` which designates a parallelogram because `GP` is a list composed of a parallelogram and its last vertex `D`.

If we enter `affix(GD)`, we get `-2+i`.

24.9.11 Check whether two lines are perpendicular: `is_perpendicular`

In plane geometry, `is_perpendicular` is a boolean function having as argument two lines.

`is_perpendicular` equals 1 if the two lines are perpendicular, and equals 0 otherwise.

We enter:

```
is_perpendicular(line(0, 1+i), line(i, 1))
```

We get:

1

We enter:

```
is_perpendicular(line(0, 1+i), line(1+i, 1))
```

24.9.12 Check whether a triangle is right or a polygon is a rectangle:

`is_rectangle`

`is_rectangle` is a boolean function and takes as argument three or four points, or a geometrical object.

`is_rectangle` equals 1 (resp. 2 or 3) if the three points form a right triangle, the right angle being specified in the first (resp. second, third) argument or if the geometrical object is a right triangle, `is_rectangle` equals 1 (resp. 2) if the four points form a rectangle (resp. a square) or if the geometrical object is a rectangle (resp. a square), 0 otherwise.

We enter:

```
is_rectangle(1,1+i,i)
```

We get:

2

We enter:

```
is_rectangle(1+i,-2+i,-2-i,1-i)
```

We get:

1

We enter:

```
GR:=rectangle(-2-i,1-i,3,GC,GD);is_rectangle(GR[0])
```

We get:

1

Indeed, `GR[0]` designates a rectangle because `GR` is a list composed of the rectangle and its vertices `C` and `D`.

24.9.13 Check whether a polygon is a rhombus: `is_rhombus`

`is_rhombus` is a boolean function and takes as argument four points or a geometrical object.

`is_rhombus` equals 1 (resp. 2) if the four points form a rhombus (resp. a square)

or if the geometrical object is a rhombus (resp. a square), 0 otherwise.

We enter:

```
is_rhombus(1+i,-1+i,-1-i,1-i)
```

We get:

1

We enter:

```
GK:=rhombus(1+i,-1+i,pi/4);is_rhombus(GK)
```

We get:

1

We enter:

```
GK:=rhombus(1+i,-1+i,pi/4,GC,DD);is_rhombus(GK[0])
```

We get:

1

Indeed, GK[0] designates a rhombus because GK is a list composed of a rhombus and its vertices GC and GD.

If we enter: `normal(coordinates(GC,GD))`, we get `[-sqrt(2)-1,-sqrt(2)+1],[-sqrt(2)+1,-sqrt(2)+1]`.

We enter:

```
is_rhombus(i,-1+i,-1-i,1-i)
```

We get:

0

24.9.14 Check whether a polygon is a square: `is_square`

`is_square` is a boolean function and takes as argument four points or a geometrical object.

`is_square` equals 1 if the four points form a square or if the geometrical object is a square, 0 otherwise.

We enter:

```
is_square(1+i,-1+i,-1-i,1-i)
```

We get:

1

We enter:

```
GK:=square(1+i,-1+i);is_square(GK)
```

We get:

1

We enter:

```
GK:=square(1+i,-1+i,C,D);is_square(GK[0])
```

We get:

1

Indeed, GK[0] designates a square because GK is a list composed of a square and ses vertices C and D.

If we enter `affix(GC,GD)`, we get `-1-i,1-i`.

We enter:

```
is_square(i,-1+i,-1-i,1-i)
```

We get:

0

24.9.15 Check whether 4 points form an harmonic division: `is_harmonic`

`is_harmonic` allows to know if four points are in harmonic division.

`is_harmonic` is a boolean function and takes as arguments four points.

`is_harmonic` equals 1 if the four points are in harmonic division and 0 otherwise.

We enter:

```
is_harmonic(0, 2, 3/2, 3)
```

We get:

1

We enter:

```
is_harmonic(0, 1+i, 1, i)
```

We get:

0

24.9.16 Check whether lines are in harmonic bundle: `is_harmonic_line_bundle`

`is_harmonic_line_bundle` takes as argument a list of lines.

`is_harmonic_line_bundle` returns:

1 if these lines are concurrent in a point,

2 if they are parallel,

3 if they are overlapping,

and 0 otherwise.

We enter:

```
is_harmonic_line_bundle([line(0, 1+i), line(0, 2+i),
                          line(0, 3+i), line(0, 1)])
```

We get:

1

24.9.17 Check whether circles are in harmonic bundle: `is_harmonic_circle_bundle`

`is_harmonic_circle_bundle` takes as argument a list of circles.

`is_harmonic_circle_bundle` returns:

1 if these circles form a beam (that is to say if they have by pair the same radical axis),

2 if these circles are concentric,

3 if these circles are overlapping,

and 0 otherwise.

We enter:

```
is_harmonic_circle_bundle([circle(0, i), circle(4, i),
                            circle(0, point(1/2))])
```

We get:

1

24.10 Exercises of geometry

24.10.1 Transformations

– Translation

Paving: any plane non crossed quadrilateral might form a pavement of the plane as a regular pattern.

We define 4 points A, B, C, D randomly:

```
menu Points->Free points->4 random point
```

In Symb, we change the name of points so that the quadrilateral A, B, C, D is not crossed.

Then, we define the quadrilateral A, B, C, D with the menu Lines->Polygons->Quadrilateral and this will be the basis pattern: $GE := \text{quadrilateral}(GA, GB, GC, GD)$

```
GA:=point()
GB:=point()
GC:=point()
GD:=point
GE:=quadrilateral(GA, GB, GC, GD)
GG:=segment(GA, GB)
GH:= segment (GB, GC)
GI:= segment (GC, GD)
GJ:= segment (GD, GA)
GK:=midpoint(GA, GB)
GL:=reflection(GK, GC)
GM:=reflection(GK, GD)
GN:=quadrilateral(GA, GB, GL, GM)
GO:= segment (GA, GB)
GP:= segment (GB, GL)
GQ:= segment (GL, GM)
GR:= segment (GM, GA)
translation(GB-GD, [GE, GN])
translation(GC-GA, [GE, GN])
```

– Inversion

The Peaucellier inverter

```
GA:=element(-1.6..1.6, 0.6)
GD:=circle(1, 1)
GE:=point(1+EXP(i*GA)
GH:=circle(GE, 2.)
GI:=circle(0, 3.)
GJ:=inter(GH, GI)
GK:=reflection(GJ, GE)
GL:=locus(GK, GA)
GG:=quadrilateral(GE, GJ[0], GK, GJ[1])
GB:=segment(0, GJ[0])
GC:=segment(0, GJ[1])
```

24.10.2 Loci

Be a direct triangle OAB right in O , with $OA = a$ and $OB = b$.

Be $D = At$ a variable half line so that: $(\vec{OA}, \vec{At}) = c, 0 \leq c \leq \pi/2$.

Let A_1 and B_1 be the respective projections of A and B on D .

What is the value of c which causes A_1 and B_1 to be mixed in a point named P ? Find the loci of A_1 and B_1 when c varies.

Show that the triangle PA_1B_1 remains similar to the triangle OAB when c varies.

Find the locus of M midpoint of A_1B_1 when c varies.

```
GA:=point(0., 3.)
GB:=point(5., 0.)
GC:=arc(1.5*i, 1.5)
GD:=(inter(GC, line(GA, GB))) [0]
GE:=arc(1.5*i, -pi/2., pi/2)
GG:=element(0..1.57, 0.25)
GH:=line(y=TAN(GG)*x)
GI:=projection(GH, GA)
```

```
GJ:=projection(GH,GB)
GK:=triangle(GD,GI,GJ)
GL:=triangle(GD,2.5,1.5*i)
GM:=midpoint(GI,GJ)
GN:=trace(GM)
```

24.11 Geometry activities

– Perpendicular bisector of AB

Create a segment AB.

Draw the perpendicular bisector of AB, by using the same geometric construction as with a compass.

Answer:

We tap:

Lines->Segment and we define with the cursor two points A and B and the segment C is automatically defined in Symb ($GC := \text{segment}(GA, GB)$).

perpen_bisector directly draws the perpendicular bisector of AB.

To do the same geometric construction as with a compass:

We tap: Curves->Circles->Circle

We pick the center B (or we enter Alpha B), we confirm with Enter or with \square , then we pick the point A (or we enter Alpha A).

The following is then displayed below the figure:

$\text{circle}(GB, GA-GB)$ (it is the circle D of center B passing by A), we confirm with Enter ($GD := \text{circle}(GB, GA-GB)$ is automatically defined in Symb).

Then, we pick the center A then the point B.

The following is then displayed below the figure:

$\text{circle}(GA, GB-GA)$ (it is the circle E of center A passing by B), we confirm with Enter ($GE := \text{circle}(GA, GB-GA)$ is automatically defined in Symb).

We can then enter in Symb:

$GG := \text{line}(\text{inter}(GD, GE))$ to plot the line joining the two points of the intersection of GD and GE ($\text{inter}(GD, GE)$ is the list of intersection points).

Or else, we define the intersection with Point->Inter and then, by designing the first intersection point, then the second one.

– Midpoint

Create a segment [AB].

Draw the midpoint of AB, either by using the coordinates, either by using the same geometric construction as with a compass.

We enter: $GI := \text{point}(\text{coordinates}(GA)/2 + \text{coordinates}(GB)/2)$

or we add to the construction of the perpendicular bisector (cf above):

$GI := \text{single_inter}(GC, GG)$

As an exercise of programming, we can also define the function Midpoint.

(The first letter of the name of the function must be in upper case, because midpoint is a CAS command).

We enter:

$$\text{Midpoint}(A, B) := \text{point}(\text{coordinates}(A)/2 + \text{coordinates}(B)/2)$$

or else if we have defined the function Perpendicular_bisector:

$$\text{Midpoint}(A, B) := \text{single_inter}(\text{segment}(A, B), \text{Perpendicular_bisector}(A, B))$$

– Isobarycenter

Create 4 points A, B, C, D.

Define the isobarycenter of A, B, C, D, by using the coordinates.

Answer:

We enter in Symb:

```
GE:=point((coordinates(GA)+coordinates(GB)+coordinates(GC)+
          coordinates(GD))/4)
```

As an exercise of programming, we can define the function ISOBAR

Warning! `isobarycenter` is an existing command.

We enter as name of program ISOBAR and we check CAS.

We enter (the variables must be in lower case):

```
(1)->BEGIN
LOCAL s,d;
d:=size(l);
s:=sum(l[k],k,1,d)/d;
RETURN s;
END;
```

By example, we enter:

```
ISOBAR(0,1,1+i,i)
```

We get:

```
(1+i)/2
```

– Barycenter

Create 4 points A, B, C, D.

To define the barycenter of $[A, 1]$, $[B, -2]$, $[C, 1]$, $[D, 3]$, by using the coordinates.

Answer:

We enter:

```
GE:=point((coordinates(GA)-
          2*coordinates(GB)+coordinates(GC)+3*coordinates(GD))/3)
```

As an exercise of programming, we can also define the function BARY

Warning! `barycenter` is an existing command which returns the barycenter of points A, B... weighted of coefficients α, β, \dots

We enter BARY as name of the program and we check CAS.

We enter (the variables must be in lower case) and we assume that l is the list `affix(GA), α , affixGB, β, \dots` :

```
(1)->BEGIN
LOCAL s,d;
d:=size(l);
s:=sum(l[k],k,2,d,2);
IF s==0 THEN RETURN "not defined" END;
RETURN sum(L[k]*L[k+1],k,1,d,2)/s;
END;
```

By example, we enter:

```
ISOBAR(0,2,1,1)
```

We get:

1/3

– Bisector of a angle

Create a triangle ABC .

Draw the bissector of the angle A of the triangle ABC , by using the same geometric construction as with a compass and by using the instruction `perpen_bissector` which plots the perpendicular bisector of a segment.

Answer:

We tap:

Lines->Triangle->Triangle and we define with the cursor three points A , B and C and then the triangle D as well as its sides are automatically defined in Symb
`(GD:=triangle(GA,GB,GC)),`
`GE:=segment([GA,GB]),`
`GG:=segment([GB,GC]),`
`GH:=segment([GC,GA]).`

We tap: Curves->Circles->Circle

We assume that $AB < AC$ so that the circle of center A passing by B intersects the segment H which is the segment AC .

We pick the center A (or we enter Alpha A), we confirm with Enter or with \rightarrow , then we pick the point B (or we enter Alpha B).

The following is then displayed below the figure:

`circle(GA,GB-GA)` which is the circle I of center A passing by B .

We tap Points->Dep.points->Inter and we designate the circle I , then the segment H
`GJ:=inter(GI,GH)` defines the intersection of the circle I with the segment H which is the segment AC .

Then, we select `perpen_bissector` to plot the perpendicular bisector of BJ

`GK:=perpen_bissector(GB,GJ)`

As an exercise of programming, we can also define the functions `Bissector` and `Exbissector`

Warning! `bissector` and `exbissector` are existing commands.

We enter, if we have defined the function `Perpendicular_bissector`:

```
Perpendicular_bissector(GA,GB,GC):=perpen_bissector(single_inter(half_line(GA,GB),circle(A,2)),single_inter(half_line(GA,GC),circle(GA,2)))
Exbissector(GA,GB,GC):={local GC1:=GA+(GA-GC);
Bissector(GA,GB,GC1)}
```

– Offset from a given length

Given three points A , B and C , we want to build a point D so that $AD = BC$.

We use the command `circle` and we enter:

`GD:=element(circle(GA,GB-GC))`

The instruction `distance(GB,GC)` returns the length of the segment BC (units defined in Plot view).

If we want to offset of this length in a given direction, we multiply this length by the unity vector of this direction.

Example:

Given three points A , B and C , build on the half-line AB a point D such as $AD = AC$.

We enter:

$$GD:=GA+distance(GA,GC)*(GB-GA)/distance(GA,GB)$$

or else

```
GD:=single_inter(circle(GA,GC-GA),half_line(GA,GB))
```

– **Offset from a given angle**

Given two points A and B, we want to build C so that the angle $(\overrightarrow{AB}, \overrightarrow{AC})$ let of measure supplied for example 72 degrees or $2 * \pi/5$ radians.

We enter, if we have checked radian in the CAS settings:

```
GD:=rotation(GA,2*pi/5,line(GA,GB))
```

or, if we are in degree (we did not check radian):

```
GD:=rotation(GA,72,line(GA,GB))
```

then we enter:

```
GC:=element(GD)
```

The instruction `angle(GA,GB,GC)` gives the measure in radians (or in degrees) of the angle $(\overrightarrow{AB}, \overrightarrow{AC})$, we can then check the construction requested.

Given two points A and B, we want to build C so that the angle $(\overrightarrow{AB}, \overrightarrow{AC})$ is equal to the angle $(\overrightarrow{OM}, \overrightarrow{OP})$.

We enter:

```
GD:=rotation(GA,angle(GO,GM,GP),line(GA,GB));
```

```
GC:=element(GD)
```

– **Perpendicular to the line BC passing by A**

Create a point A and a line BC not passing by A.

Draw the perpendicular to the line BC passing by A, by using the same geometric construction as with a compass.

Answer:

We tap: `Points->Point` and we define with the cursor the point A, then we tap `Lines->Line` and we define with the cursor the points B and C and the line D is automatically defined in Symb (`GD:=line(GB,GC)`).

We tap: `Curves->Circles->Circle`

We pick the center B (or we enter Alpha B), we confirm with `Enter` or with `,` then we pick the point A (or we enter Alpha A).

The following is then displayed below the figure:

`circle(GB,GA-GB)` which is the circle E of center B passing by A.

We pick the center C (or we enter Alpha C), we confirm with `Enter` or with `,` then we pick the point A (or we enter Alpha A).

The following is then displayed below the figure:

`circle(GC,GA-GC)` which is the circle G of center C passing by A.

We tap `Points->Dep.points->Inter` and we designate the first intersection point of circles E and G. This defines the point H, then we designate their second intersection point and this defines the point K. Then, we draw the line HK.

– **Perpendicular to the line AB passing by A**

Draw the perpendicular to the line AB passing by A by using the same geometric construction as with a compass.

Answer:

We tap:

Lines->Line and we define with the cursor the points A and B, and the line C is automatically defined in Symb ($GC:=line(GA,GB)$).

We tap: Curves->Circles->Circle

We pick the center A (or we enter Alpha A), we confirm with Enter or with , then we pick the point B (or we enter Alpha B).

The following is then displayed below the figure:

$circle(GA,GB-GA)$ which is the circle D of center A passing by B.

We tap Points->Dep.points->inter and we designate the point of intersection (other than B) of the circle D with the line C. This defines the point E, then we plot the perpendicular bisector of BE. $perpen_bissector(GB,GE)$ draws the perpendicular bisector of two points defined by $inter(C3,line(GA,GB))$, this perpendicular bisector passes by A and is perpendicular to AB.

As an exercise of programming, we can also define the function PERP

Warning! `perpendicular` is an existing command.

We enter as name of program PERP and we check CAS.

We enter (the variables must be in lower case or names of geometrical objects) and we assume that GA is a point and that GD is a line:

```
(GA,GD)->BEGIN
LOCAL GM,GE,GL;
GE:=element(GD);
IF is_element(GA,GD) THEN
  GL:=inter(GD,circle(GA,1));
  RETURN equation(perpen_bissector(GL));
END;
IF angle(GE,GA,GD)==pi/2 OR angle(GE,GA,GD)==-pi/2
  RETURN equation(line(GA,GE)); END;
GM:=midpoint(op(inter(GD,circle(GA,GE-GA))));
RETURN equation(line(GA,GM));
END;
```

We enter:

```
PERP(point(i),line(0,1+i))
```

We get:

$$y=-3*x+1$$

We enter:

```
PERP(point(i),line(-2,2+2i))
```

We get:

$$y=-2*x+1$$

We enter:

```
PERP(point(1),line(0,1+i))
```

We get:

$$y=-x+1$$

We enter:

```
PERP(point(1),line(-2,1+i))
```

We get:

$$y = -3x + 3$$

– **Parallel to a line passing by A**

Create a point A and a segment BC not passing by A . Draw the parallel to the line BC passing by A by using the instruction `perpendicular`.

Answer:

We tap with the finger to create three point A , B and C , and the segment BC .

We enter:

```
GD:=perpendicular(GA,line(GB,GC))
```

this plots the perpendicular to BC passing by A

```
GP:=perpendicular(GA,GD)
```

this plots the perpendicular to D passing by A .

As an exercise of programming, we can define the function `PARAL` which returns the equation of the line parallel to D passing by A (Take care to have the first letter of the name of the function in upper case, because `parallel` is an existing geometry command).

We enter `PARAL` as name of program and we check `CAS`.

We enter (variable names must be in lower case or names of geometrical objects) and we assume that GA is a point and GD is a line:

```
(GA,GD)->BEGIN
LOCAL GP:=line(PERP(GA,GD));
RETURN PERP(GA,GP);
};
```

We enter:

```
PARAL(point(i),line(0,1+i))
```

We get:

$y = x + 1$ We enter:

```
PARAL(point(1),line(-2,2+i))
```

We get:

$$y = (1/4)x - 1/4$$

– **Parallels to a line located at a distance d of this line.**

Create a point A and a segment BC not passing by A .

Then, create a point D to define $d = \text{distance}(A, D)$.

Draw the parallel to the line BC located at a distance $d = \text{distance}(A, D)$ of BC .

Answer:

We pick on the screen to create three point A , B and C , the segment BC , then the point D ($d = \text{distance}(A, D)$).

We enter:

```
GD1:=perpendicular(GB,line(GB,GC));
GC1:=circle(GB,distance(GA,GD));
GI:=inter(GD1,GC1);
GE:=GI[0];
GF:=GI[1];
```

or we use the complex numbers to define GE and GF :

$GE := GB + i * (GC - GB) * \text{distance}(GA, GD) / \text{distance}(GB, GC)$ the point E is at a distance $d = \text{distance}(A, D)$ of the line BC, $GF := GB - i * (GC - GB) * \text{distance}(GA, GD) / \text{distance}(GB, GC)$ the point F is at a distance $d = \text{distance}(A, D)$ of the line BC (E and F are symmetrical with respect to BC), and then, `parallel(GE, line(GB, GC))` draws a line parallel to the line BC located at a distance $d = \text{distance}(A, D)$ of BC. `parallel(GF, line(GB, GC))` draws the other line parallel to the line BC, located at a distance $d = \text{distance}(A, D)$ of BC.

Chapter 25 The spreadsheet

25.1 Generalities

Once the Spreadsheet application is open, the Menu key gives access to the following functions:

SUM MEAN AMORT STAT1 REGRS PredY PredX HypZ1mean HypZ2mean

The spreadsheet is a calculation sheet in the form of a grid, composed of rows and columns forming what are called the cells. Cells stores values, commands, or formulas referring to other cells.

25.2 Screen of the spreadsheet

The spreadsheet is a table composed of columns designated by two letters in upper case A, B, C, ... or sometimes in lower case g, l, m, z, and rows numbered 1, 2, 3, ... Thus, A1 designates the first cell of the spreadsheet.

25.2.1 Copy the content of a cell to another

For instance:

We enter 1 in A1 and we want to copy downward the formula =A1+1 valid in A2, to get 1, 2, 3, 4, 5 in the column A.

There are three ways to do it: choose the one which suits you the most!

- first way: we tap `select` (push buttons) to select the block from A2 to A5, and we enter: =A1+1,
- second way: we enter =A1+1 in A2, copy, then `select` (push buttons) to select from A3 to A5, and paste, by choosing the formula among the expressions which are in the clipboard,
- third way: we put =A1+1 in A2, then we select from A2 to A5, then `Edit` and `Enter`

25.2.2 Relative and absolute references

In a cell, we can enter:

- a string of characters,
- an algebraic expression,
- a formula referring to other cells. These references to the cell storing the formula can be either absolute, either relative. Relative references become absolute by adding the symbol \$ ahead of the letter of the column and/or ahead of the number of the line of the cell to refer to.

Relative references allow to designate the cells with respect to another:

Thus, A0 entered in the cell B1 designates the cell located in the preceding column and the preceding row, and this is the information copied when we copying the formula downward or rightward.

Examples:

Given 1 in A1, we enter the following formula in B2:

- \$A\$1+2: the content of B2 is then 3, and if we copy this formula downward, we get 3's in the column B because we copy the same formula \$A\$1+2 in all the cells of the column B. If we copy this formula rightward, we also get 3's on the first row, because we copy in all the cells of the first line the same formula \$A\$1+2, since \$A\$1 is the absolute reference of the cell A1.
- \$A1+2: the content of B2 is then 3, and if we copy this formula downward, it will become \$A2+2 in B2, \$A2+2 in B3. Then, the value of B2 depends on the value of A1, the value of B3 depends on the one of A2, etc., ...

If we copy this formula rightward, it will become \$A1+2 in C2, \$A1+2 in D2 ...we get then a row of 3's. \$A0 always refer to the column A: A is an absolute reference but 0 designates here the preceding line because \$A1 is stored in B2.

- A\$1+2: the content of B2 is then 3, and if we copy this formula downward, we get 3's in column B, but if we copy this formula rightward, it will become B\$1+2 in C2, C\$1+2 in D2 etc., ...
- =A1+2: the content of B2 is then 3, and if we copy this formula downward, it will become =A2+2 in B3, =A3+2 in B4 etc.,... If we copy this formula rightward, it will become =B1+1 in C2, =C1+1 in D2 etc., ...

25.3 Functions of the spreadsheet

25.3.1 Function SUM

SUM does the sum of the cells supplied as argument.

For instance: SUM(A1:B3) does the sum A1+A2+A3+B1+B2+B3

25.3.2 Function MEAN

MEAN

25.3.3 Function AMORT

AMORT allows to calculate the amortization of a loan. The syntax is:

AMORT(range, n, i, pv, pmt, [ppyr=12, cpyr=ppyr, Groupement=ppyr, beg=false, fix=current,], "Configuration")

where:

- range refers to cells in which one pourra lire the results,
- n is the number of periods of the loan (i.e. number of payments),
- i is the interest rate,
- pv is the remaining due sum (present value),
- pmt is the amount of each payment (per-period payment).

For instance:

but it seems easier to use the Finance application.

25.3.4 Function STAT1

STAT1 allows to do statistics at one variable.

The syntax is: STAT1(range, [mode], ["configuration"]).

range is the source of data, for example: A1:B3

mode equals:

- 1 if each column is independent,
- 2 if the columns are used by pairs (data, frequencies),
- 3 if the columns are used by pairs (data, weight),
- 4, but it seems easier to use the application:

Stats-1Var.

25.3.5 Function REGRS

REGRS attempts to fit the input data to a specified function (default is linear).

The syntax is: REGRS(range, [mode], ["configuration"]).

For instance: but it seems easier to use the commands

linear_regression, exponential_regression, etc., ...

25.3.6 Functions PredY PredX

For instance:

25.3.7 Functions HypZ1mean HypZ2mean

For instance: but it seems easier to use the application Inference.

25.4 Use of the spreadsheet based on examples

25.4.1 Exercise 1

Sum of odd integers.

Write in the spreadsheet:

- the integers from 1 to 10 in the column A,
- the squares of integers from 1 to 10 in the column B
- the odd integers $2k - 1$ for $k = 1..10$ in the column C,
- the sum of odd integers $\sum_{j=1}^k 2j - 1$ for $k = 1..10$ in the column D.

Calculate $\sum_{j=1}^k 2j - 1$ when $k \in \mathbb{N}$.

A solution

- We enter 1 in A1.
We enter $=A1+1$ in A2, and then, we copy downward the formula stored in A2: for this we select A2...A10 (when select. of the push buttons is visible and we select A2...A10 with the cursor) and we enter $=A1+1$.
- We put $=A1^2$ in B1, and then, we copy downward the formula stored in B1: for this we select B1...B10 (when select. of the push buttons is visible and we select B1...B10 with the cursor) and we enter $=A1^2$.
- We put $=2*A1-1$ in C1, and then, we copy downward the formula stored in C1: for this we select C1...C10 (when select. of the push buttons is visible and we select C1...C10 with the cursor) and we enter $=2*A1-1$.
- We enter 1 in D1.
We enter $=D1+C2$ in D2, and then, we copy downward the formula stored in D2: for this we select D2...D10 (when select. of the push buttons is visible and we select D2...D10 with the cursor) and we enter $=D1+C2$.

We get:

	A	B	C	D
1	1	1	1	1
2	2	4	3	4
3	3	9	5	9
4	4	16	7	16
5	5	25	9	25
6	6	36	11	36
7	7	49	13	49
8	8	64	15	64
9	9	81	17	81
10	10	100	19	100

We will then demonstrate by recurrence that:

$$\sum_{j=1}^k 2j - 1 = k^2$$

when $k \in \mathbb{N}$ The formula is true for $k = 1..10$ (cf above).

Let us assume that for

$$k = n \sum_{j=1}^k 2j - 1 = n^2$$

so for $k = n + 1$ we have

$$\sum_{j=1}^{n+1} 2j - 1 = \sum_{j=1}^n + 2(n+1) - 1 = n^2 + 2n + 1 = (n+1)^2$$

So the formula is demonstrated.

2	1	1	0	0	0	0	0	0	0	0	0
3	1	2	1	0	0	0	0	0	0	0	0
4	1	3	3	1	0	0	0	0	0	0	0
5	1	4	6	4	1	0	0	0	0	0	0
6	1	5	10	10	5	1	0	0	0	0	0
7	1	6	15	20	15	6	1	0	0	0	0
8	1	7	21	35	35	21	7	1	0	0	0
9	1	8	28	56	70	56	28	8	1	0	0
10	1	9	36	84	126	126	84	36	9	1	0
11	1	10	45	120	210	252	210	120	45	10	1

The Fibonacci sequence and the Pascal triangle

When we do the sum of the diagonals of the Pascal triangle upward, we get the Fibonacci sequence. For instance:

$$\begin{pmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 5 \\ 8 \\ 13 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & \nearrow & \nearrow & \nearrow & \nearrow & \nearrow \\ 1 & \nearrow & 1 & 0 & 0 & 0 \\ 2 & \nearrow & 2 & 1 & 0 & 0 \\ 3 & \nearrow & 3 & 3 & 1 & 0 \\ 5 & \nearrow & 4 & 6 & 4 & 1 \\ 8 & \nearrow & 5 & 10 & 5 & 5 \\ 13 & \nearrow & 6 & 15 & 20 & 15 & 6 \end{pmatrix}$$

We enter:

```
A:=makemat((j,k)->comb(j,k),11,11)
L:=sum(A[j-k, k],k=1..j-1)$(j=2..12)
```

We get:

```
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89
```

We enter:

```
L1:=0, sum(A[j-k-1, k-1], k=2..j-2)$(j=3..12)
```

We get:

```
0, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
```

We enter:

```
L2:=1, sum(A[j-k-1, k], k=1..j-2)$(j=3..12)
```

We get:

```
1, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

We enter:

```
[L1]+[L2]
```

We get:

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

To show it, we use the relation for $n \in \mathbb{N}$:

$$C_n^0 = 1, C_n^n = 1, C_n^p = 0 \text{ and } C_n^p = C_{n-1}^p + C_{n-1}^{p-1} \text{ for } 0 < p \leq n.$$

Be an the series which equals the sum of the upward diagonals of the Pascal triangle.

We have then:

$$a_0 = 1, a_1 = 1 \text{ and for } n > 1:$$

$$a_n = \sum_{p=0}^{\lfloor \frac{n}{2} \rfloor} \text{comb}(n-p, p) = 1 + \sum_{p=1}^{\lfloor \frac{n}{2} \rfloor} \text{comb}(n-p, p)$$

$$a_n = 1 + \sum_{p=1}^{\lfloor \frac{n}{2} \rfloor} \text{comb}(n-p-1, p-1) + \sum_{p=1}^{\lfloor \frac{n}{2} \rfloor} \text{comb}(n-p-1, p)$$

$$\text{We have for any } n: \lfloor \frac{n}{2} \rfloor - 1 = \lfloor \frac{n-2}{2} \rfloor.$$

$$\text{If } n = 2k, \text{ we have } \lfloor \frac{n}{2} \rfloor = \lfloor \frac{(n-1)}{2} \rfloor + 1 = k:$$

$$1 + \sum_{p=1}^{\lfloor \frac{n}{2} \rfloor} \text{comb}(n-p-1, p) = \sum_{p=0}^{\lfloor \frac{n-1}{2} \rfloor} \text{comb}(n-1-p, p) + \text{comb}(2k-1-k, k) = 0$$

$$\text{because } \text{comb}(2k-1-k, k) = 0$$

$$\sum_{p=1}^{\lfloor \frac{n}{2} \rfloor} \text{comb}(n-p-1, p) = \sum_{p=0}^{\lfloor \frac{n-1}{2} \rfloor} \text{comb}(n-1-p, p) = a_{n-1}$$

$$\sum_{p=1}^{\lfloor \frac{n}{2} \rfloor} \text{comb}(n-p-1, p-1) = \sum_{p=0}^{\lfloor \frac{n-2}{2} \rfloor} \text{comb}(n-2-p, p) = a_{n-2}$$

$$\text{because } \lfloor \frac{n}{2} \rfloor - 1 = \lfloor \frac{n-2}{2} \rfloor$$

$$\text{If } n = 2k + 1, \text{ then we have } \lfloor \frac{n}{2} \rfloor = \lfloor \frac{n-1}{2} \rfloor = k:$$

$$1 + \sum_{p=1}^{\lfloor \frac{n}{2} \rfloor} \text{comb}(n-p-1, p) = \sum_{p=0}^{\lfloor \frac{n-1}{2} \rfloor} \text{comb}(n-1-p, p) = a_{n-1}$$

$$\sum_{p=1}^{\lfloor \frac{n}{2} \rfloor} \text{comb}(n-p-1, p-1) = \sum_{p=0}^{\lfloor \frac{n-2}{2} \rfloor} \text{comb}(n-2-p, p) = a_{n-2}$$

$$\text{because } \lfloor \frac{n}{2} \rfloor - 1 = \lfloor \frac{n-2}{2} \rfloor$$

Thus:

$$a_0 = 1, a_1 = 1 \text{ and for } n > 1 \text{ we have } a_n = a_{n-1} + a_{n-2}$$

a_n is then the Fibonacci sequence.

Chapter 26 Other Applications

26.1 Function application

We enter in the Symbolic view of the Function application:

$$F1(X) = \sin(2X) + X$$

$$F2(X) = \partial(F1(A), A=X)$$

We highlight $\partial(F1(A), A=X)$ and we press `EVAL` of the push buttons.

We get, if we are in Radians:

$$F2(X) = \cos(2 \cdot X) \cdot 2 + 1$$

In HOME, we enter:

$$F1(1)$$

We get, if we are in Radians:

$$\sin(2) + 1$$

In HOME, we enter:

$$F2(1)$$

We get, if we are in Radians:

$$\cos(2) \cdot 2 + 1$$

In HOME, we enter:

$$\partial(F1(X), X=1)$$

We get, if we are in Radians:

$$0.167706326906$$

26.2 Sequence application

26.2.1 Fibonacci sequence

We enter:

$$U1(1) = 1$$

$$U1(2) = 1$$

$$U1(N) = U1(N-1) + U1(N-2)$$

We get by pressing `Num` :

the Fibonacci sequence

We enter the sequence of the remainders:

$$U1(1) = 1$$

$$U1(2)=1$$

$$U1(N)=U1(N-1)+U1(N-2)$$

We get by pressing Num:

the Fibonacci sequence

26.2.2 GCD

Here is an implementation of Euclid's algorithm with the HPrime.

Here is the description of this algorithm:

We do successive euclidean divisions:

$$\begin{aligned} A &= B \times Q_1 + R_1 \quad 0 \leq R_1 < B \\ B &= R_1 \times Q_2 + R_2 \quad 0 \leq R_2 < R_1 \\ R_1 &= R_2 \times Q_3 + R_3 \quad 0 \leq R_3 < R_2 \\ &\dots\dots\dots \\ R_{n-2} &= R_{n-1} \times Q_n + R_n \quad 0 \leq R_n < R_{n-1} \end{aligned}$$

After a finite number of steps (at most B), there is an integer n such as: $R_n = 0$.

Then we have:

$$\begin{aligned} GCD(A, B) &= GCD(B, R_1) = \dots \\ GCD(R_{n-1}, R_n) &= GCD(R_{n-1}, 0) = R_{n-1} \end{aligned}$$

Thanks to the Sequence application, we write the series of the remainders.

We enter the series of remainders R :

$$U1(1)=76$$

$$U1(2)=56$$

$$U1(N)=irem(U1(N-2), U1(N-1))$$

We get by pressing Num:

76, 56, 20, 16, 4, 0 then the GCD of 76 and 56 is 4

26.2.3 Bezout identity

The Euclid's algorithm allows to trouver a paired value U, V such as:

$$A \times U + B \times V = GCD(A, B)$$

With the Sequence application, we will define "the sequence of remainders" R and two sequences U and V , so that at each step we have:

$$R_n = U_n \times A + V_n \times B.$$

If Q is "the sequence of quotients", $Q_n = iquo(R_{n-2}, R_{n-1})$ because Q_n is the integer quotient of R_{n-2} by R_{n-1} and we have:

$$R_n = irem(R_{n-2}, R_{n-1}) = R_{n-2} - Q_n \times R_{n-1} \text{ because } R_{n-2} = R_{(n-1)Q_n} + R_n \text{ with } 0 \leq R_n < R_{n-1}.$$

U_n and V_n will then check the same relation of recurrence. We have at the beginning:

$$R_1 = A \quad R_2 = B$$

$$U_1 = 1 \quad U_2 = 0 \text{ because } A = 1 \times A + 0 \times B$$

$$V_1 = 0 \quad V_2 = 1 \text{ because } B = 0 \times A + 1 \times B$$

We enter in U_1 the series of remainders R for $A = 76$ and $B = 56$:

$$U1(1)=76$$

$$U1(2)=56$$

$$U1(N)=irem(U1(N-2), U1(N-1))$$

We enter the series of quotients Q in U_2 :

$$U2(1)=0$$

$$U_2(2) = 0$$

$$U_2(N) = \text{iquo}(U_1(N-2), U_1(N-1))$$

We will put U in U_3 and U_4 in V so that at each step we have $76 * U + 56 * V = R$.

Since $R(N) = R(N-2) - R(N-1) * Q(N)$ and $Q(N) = \text{iquo}(R(N-2), R(N-1))$, if we have:

$$76 * U(N-2) + 56 * V(N-2) = R(N-2) \quad (1)$$

and

$$76 * U(N-1) + 56 * V(N-1) = R(N-1) \quad (2)$$

By doing $(1) - (2) * Q(N)$, we have:

$$76 * (U(N-2) - U(N-1) * Q(N)) + 56 * (V(N-2) - V(N-1) * Q(N)) = R(N-2) - R(N-1) * Q(N) = R(N)$$

The recurrence relations are then:

$$U(N) = U(N-2) - U(N-1) * Q(N) = U(N-2) - U(N-1) * \text{iquo}(R(N-2), R(N-1))$$

$$V(N) = V(N-2) - V(N-1) * Q(N) = V(N-2) - V(N-1) * \text{iquo}(R(N-2), R(N-1))$$

We enter in U_3 the sequence of U :

$$U_3(1) = 1$$

$$U_3(2) = 0$$

$$U_2(N) = U_3(N-2) - U_3(N-1) * \text{iquo}(U_1(N-2), U_1(N-1))$$

We enter in U_4 the sequence of V :

$$U_4(1) = 0$$

$$U_4(2) = 1$$

$$U_4(N) = U_4(N-2) - U_4(N-1) * \text{iquo}(U_1(N-2), U_1(N-1))$$

Thus, for each N we have $76 * U_3(N) + 56 * U_4(N) = U_1(N)$

By pressing Num, we get:

$$76, 0, 1, 0 \quad (76 = 76 * 1 + 56 * 0)$$

$$56, 0, 0, 1 \quad (56 = 76 * 0 + 56 * 1)$$

$$20, 1, 1, -1 \quad (20 = 76 * 1 + 56 * -1)$$

$$16, 2, -2, 3 \quad (16 = 76 * -2 + 56 * 3)$$

$$4, 1, 3, -4 \quad (4 = 76 * 3 + 56 * -4)$$

$$0, 4, -14, 19 \quad (0 = 76 * -14 + 56 * 19)$$

so $76 * 3 + 56 * -4 = 4 = \text{GCD of } 76 \text{ and } 56$

26.3 Parametric application

We enter:

$$X_1(T) = -\cos(2T)$$

$$Y_1(T) = \sin(3T)$$

We get:

the plot of a curve looking like an α

26.4 Polar application

We enter:

$$R1(\theta) = \cos(3\theta)$$

We get:

the plot of a clover

26.5 Solve application

We enter in the Symbolic view of the Solve application:

$$\cos(X) - X$$

By pressing Num:

We enter:

1

To start the iteration, and then we press SOLVE (push buttons) to get:

0.739085133215

Then, we enter in HOME:

X

We get:

0.739085133215

the value obtained has been stored in X.

26.6 Finance application

We fill in all different fields but one, and we press SOLVE of the push buttons.

For instance, we want to know the monthly payments of a 10-year loan of \$ 100,000 at 4.6 % annual interest, we enter:

NbPmt	120	IPYR	4.6
PV	100,000	PPYR	12
PMT		CPYR	12
FV	0.00	Fin Y	
Group Size	12		

We highlight PMT and we press SOLVE of the push buttons.

We get the value of the monthly payment:

-1,041.21

The payments of this loan are then of \$ 1,041.21 per month.

Press AMORT of the push buttons.

We get the amortization table by groups of 12 months:

Princ gives the amount of the paid sums
 Inter gives the amount of the interest
 Balan gives the amount of the due sums

For example, at the end of the second year, we have:

Princ=-16,505.07:	we have then reimbursed \$ 16,505.07
Inter=-8,484.0:	we have paid \$ 8,484 of interest
Balan=83,494.9: \$ 83,494.93	are to be reimbursed (100,000 - 16,505.07)

By pressing the key `PLOT` we get the amortization graph.

We can read:

1-12 Principal 8,063.12 and Interest 4,431.41

and by using the right arrow

13-24 Principal 8,441.95 and Interest 4,052.59

Thus, we recover the previous values:

$8,063.12 + 8,441.95 = 16,505.07$ and $4,431.41 + 4,052.59 = 8,484.0$.

We returns in the application Finance by pressing the key `Num`.

If the payments are quaterly, we change:

`NbPmt 40`

and

`PPYR 4`

and we leave

`CPYR 12`.

We highlight `PMT` and we press `SOLVE` of the push buttons.

We get the value of the quarterly payments:

`-3,133.03`

The payments of this loan are then of \$ 3,133.03 by quarter.

26.7 Linear Solver application

This application allows to solve linear systems of two equations at two unknowns X, Y , or of three equations at three unknowns X, Y, Z .

Note:

To solve a linear system of two equations at two unknowns X, Y , you have to set 2×2 in the push buttons.

The fields of the system are displayed and we only need to enter the coefficients.

The answer appears at the bottom of the screen.

We enter:

$$1 * X + 1 * Y + 1 * Z = 3$$

$$2 * X + (-2) * Y + 1 * Z = 1$$

$$3 * X + 1 * Y + 1 * Z = 5$$

We get:

$$X : 1, Y : 1, Z : 1$$

We enter:

$$1 * X + 1 * Y + 1 * Z = 3$$

$$2 * X + 2 * Y + 2 * Z = 1$$

$$3 * X + 1 * Y + 1 * Z = 5$$

We get:

No Solutions

26.8 Triangle Solver application

Given a triangle defined by three data: length of side and/or value of angles (cf the case of equalities of triangles).

The application Triangle Solver allows to calculate the length of other sides and/or the value of other angles of this triangle.

Be the triangle of sides A, B, C such as $A = 4 B = 3 C = 5$.

We want to get the value of the angles of this triangle.

We enter:

$$A=4, B=3, C=5$$

Tap **SOLVE** of the push buttons. We get, if we are in degrees:

$\alpha = 5.31301E1$ (it is the angle opposite to side A)

$\beta = 3.68699E1$ (it is the angle opposite to side B)

$\delta = 90$ (it is the angle opposite to side C)

In Home, if we enter $5.31301E1$, highlight $5.31301E1$ and press the key **Shift a \leftrightarrow b/c**, we get:

$$53^{\circ}7'48.36''$$

If we press again the key **Shift a \leftrightarrow b/c**, we get:

$$53.1301$$

26.9 1-Var Statistics

The application Statistics 1-Var allows to do statistics at one variable with or without frequency.

Example We measure the sizes in cm of 10 people.

We get:

150, 165, 170, 165, 160, 170, 160, 175, 165, 180.

1. Calculate the median, the mean and the standard deviation of this sample.

We open the application **Statistics 1-Var** and we use one single column.

We tap **Start** (push buttons) and we enter in **D1**:

150 Enter 165 Enter....

We can, if wished, sort the data with **Sort** of the push buttons.

Press **Symb**, we set **H1:D1** and we check that we have **Plot1:Histogram**

Press the key **Num** and tap **Stats** of the push buttons.

We get:

```
Nb Item 10
X min 150
X Q1 160
X med 165
X Q3 170
X max 180
ΣX 1660
ΣX^2 276200
MoyX 166
SX 8.432740
σX 8
SE X 2.6666667
```

2. Draw the histogram.

We first set the **Plot Setup** (**Shift Plot**): **Width 5**

HRNG 150 180

XRNG 150 180

YRNG -1 10

XTICK 1 YTICK 1

Then, we press **Plot**.

We get:

the histogram

3. Display the Box Whisker.

We press `Symb` and highlight `Plot1: Histogram`.

With `Choos` (push buttons) we choose `Box Whisker`, then we press `Plot`.

We get:

The plot of a box whisker

4. Same exercise by using a column for the frequency.

We measure the sizes in cm of 10 people.

We get:

150:1
160:2
165:3
170:2
175:1
180:1

We press `Num` and enter in `D2`:

150 Enter 160 Enter 165 Enter 170 Enter 175 Enter 180 Enter

and in `D3`:

1 Enter 2 Enter 3 Enter 2 Enter 1 Enter 1 Enter

We press `Symb` and we unset `H1:D1` and we put:

`H2:D2` and we enter `D3` when `Freq` is highlighted. Thus `H2` is checked as well as `Plot2: Histogram`.

We get the same thing that previously.

Nb Item 10
X min 150
X Q1 160
X med 167.5
X Q3 175

26.10 2-Var statistics

The application `Statistics 2-Var` allows to do statistics at two variables.

Example

We measure the sizes in cm and the weight in kg of 10 people.

We get:

150:41
165:53
170:70
165:63
160:52
170:68
160:62
175:72
165:58
180:75

1. Calculate the average size and the average weight of these people.

Calculate the correlation coefficient.

We open the application `Statistics 2-Var` and we use two columns.

Tap `Start` of the push buttons.

We enter in `C1`:

150 Enter 165 Enter....

We enter in `C2`:

41 Enter 53 Enter....

We can, if we want, sort the data with `Sort` of the push buttons.

Press `Symb`, we put `S1:C1 C2`, we set `S1` and we check that we have:

Type1:Linear

Fit1: $m \cdot X + b$

Press the key `Num` and tap `Stats` of the push buttons.

We get:

Nb Item 10

Corr 0.91798

CoefDet 0.84269

SCov 81.7778

PCov 73.6

X max 180

P

XY 102660

By tapping `X` (push buttons) we have:

MoyX 166

P

X 1660

P

X^2 276200

SX 8.432740

σ_X 8

SE X 2.6666667

By tapping `Y` (push buttons) we have:

MoyY 61.4

P

Y 614

P

Y^2 38704

SY 10.5641

σ_Y 10.0220

SE Y 3.340659

2. Plot the cloud of dots

We first set the Plot Setup (`Shift Plot`): `XRNG 145 185`

`YRNG 39 77`

`XTICK 5 YTICK 2`

Then, we press `Plot`.

We get:

the cloud of dots and the plot of the regression line

3. Determinate a line of linear regression by the least squares method.

We get the equation of the line of linear regression by pressing `Symb`.

We get:

`Fit1:1.15*X+-129.5`

26.10.1 Exercises

– The aim of this activity is the study of the size (in cm) of a sample of 250 basket players.

Size	17	17	17	17	17	17	17	18	18	18	18	18	18	18	18
	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7
Frequenc y	4	8	7	18	23	22	24	32	26	25	18	19	10	8	6

The activity starts by the calculation of statistic parameters and is followed by the plot of the correlated graphs: bar graph, histograms and polygon of cumulated increasing frequency.

We enter:

`Ts:=(173+j)$(j=0..14)`

We get:

```
173,174,175,176,177,178,179,180,181,182,183,184,185,186,187
```

We enter:

```
Ef:=(4,8,7, 18, 23, 22, 24, 32, 26, 25, 18, 19, 10, 8, 6)
sum(Ef)
```

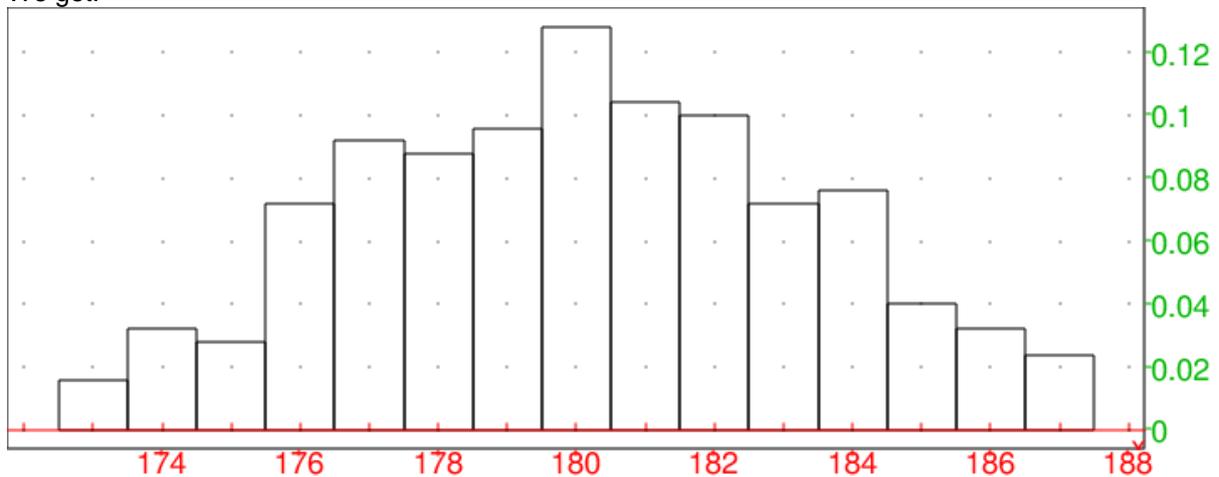
We get:

250

We enter:

```
histogram(tran([[Ts],[Ef]]))
```

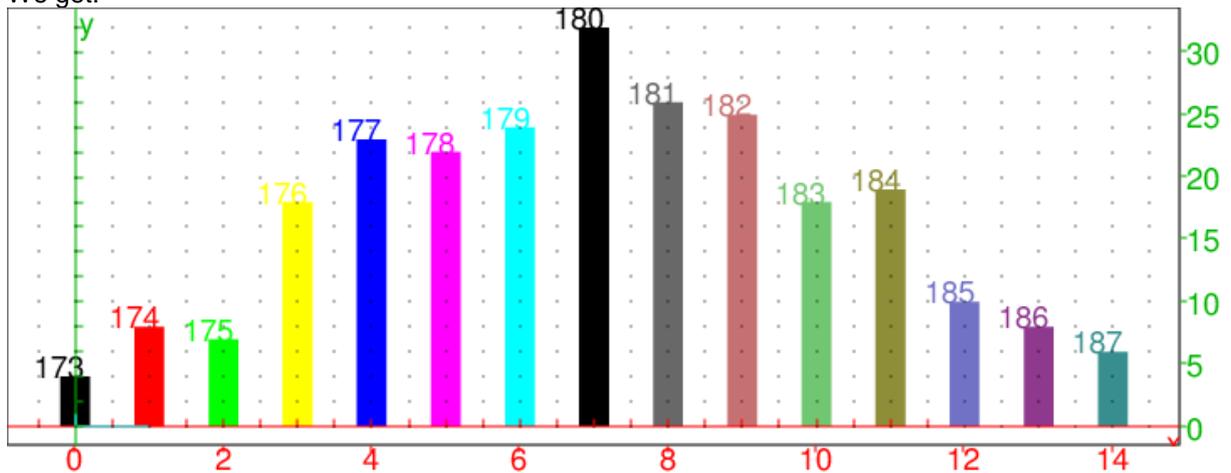
We get:



We enter:

```
bar_plo([[T],[Ef]])
```

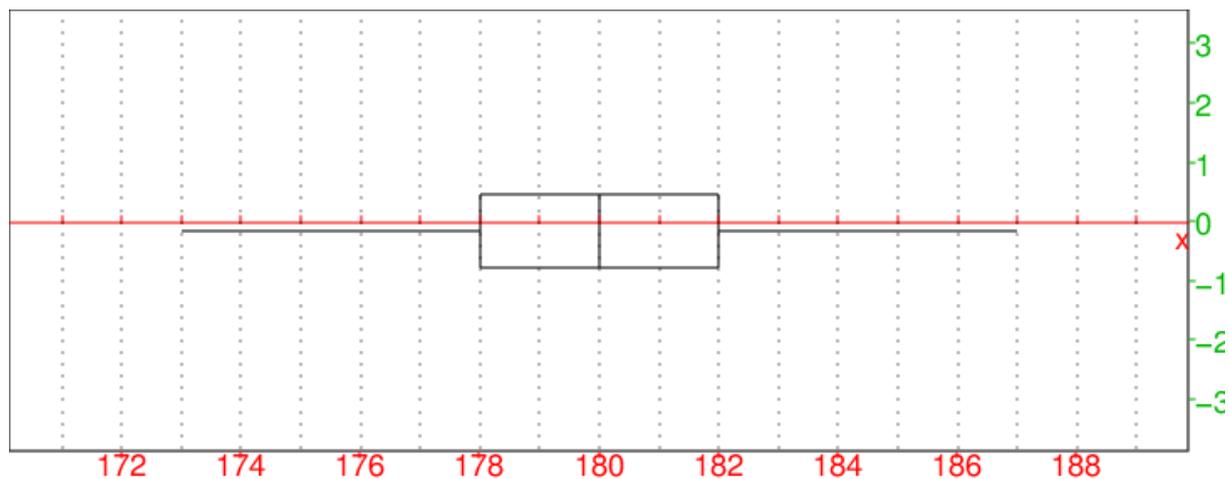
We get:



We enter:

```
boxwhisker([T],[Ef])
```

We get:



We enter:

```
Efc:=(sum(Ef[j],j=1..k)/250.)$(k=1..size(Ef))
```

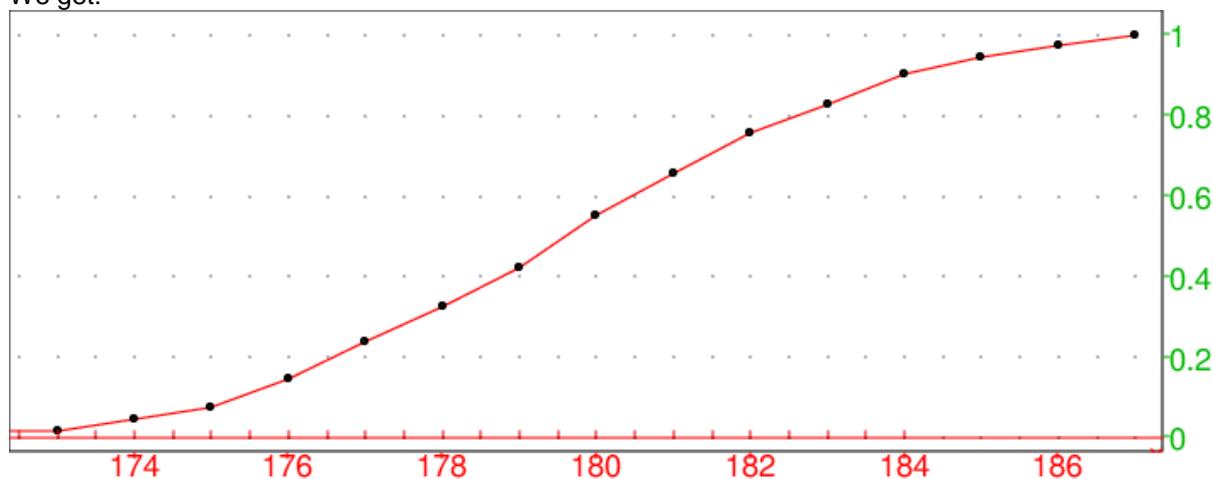
We get:

```
0.016,0.048,0.076,0.148,0.24,0.328,0.424,0.552,0.656,0.756,0.828,0.904,0.944,0.976,1.0
```

We enter:

```
scatterplot([[T],[Efc]],cumulated_frequencies(trn([[T],[Ef]]))
```

We get:



We enter:

```
evalf(mean([T],[Ef]))
```

We get:

```
180.104
```

We enter:

```
evalf(sttdev([T],[Ef]))
```

We get:

```
3.27859482096
```

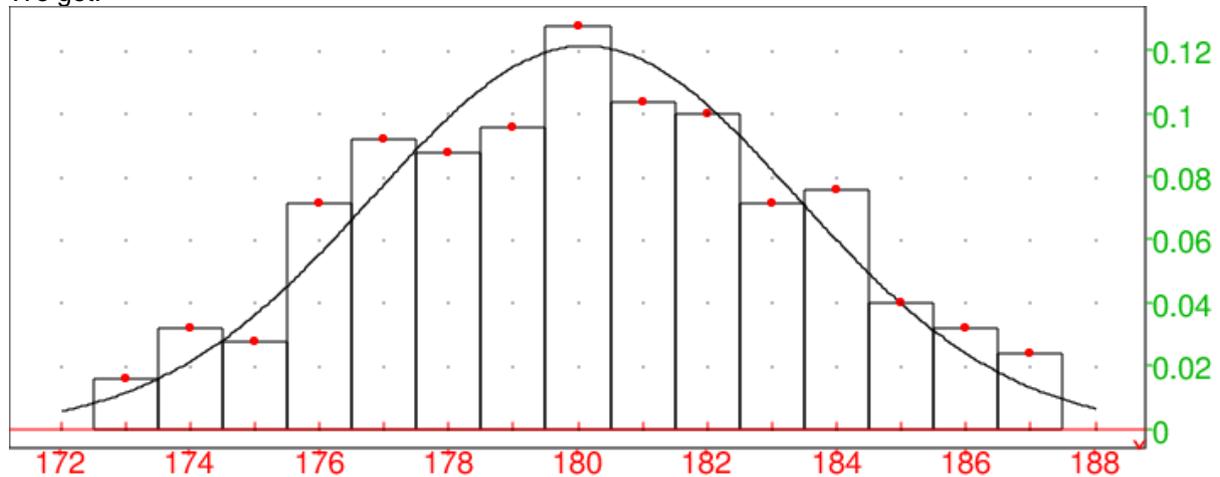
We enter:

```

histogram(trn([[T],[Ef]])),
plotfunc(normald(180.104,3.27859482096,x),x=172..188),
scatterplot(trn([[T],[Ef]/250.]))

```

We get:



An urn contains 12 red balls and 3 green balls. We propose to simulate the draw of a ball from the urn and then of to observe the sampling fluctuation on samples of size 225. Given the content of the urn, the probability to draw a green ball is $\frac{1}{5} = 0.2$.

Is our simulation valid? We create the function `randmultinom` (cf 13.4.6):

```

(n,p,c)->BEGIN
local k,j,l,r,x,y;
k:=size(p);
if size(c)!=k then return "error"; end;
x:=cumSum(p);
if x[k]!=1 then return "error"; end;
y:=MAKELIST([c[j],0],j,1,k);
for j from 1 to n do
  r:=rand(0,1);
  l:=1;
  while r>x[l] do
    l:=l+1;
  end;
  y[l,2]:=y[l,2]+1
end;
return y;
END;

```

Warning! Write `MAKELIST([c[j],0],j,1,k)` with `MAKELIST` in upper case, or write `makelist(j->[c[j],0],1,k)` or `seq([c[j],0],j,1,k)`.

We enter:

```
L:=randmultinom(225,[4/5,1/5],["R","V"]);
```

We get:

```
[["R",179],["V",46]]
```

We get then 46 times the green ball.

We first analyses 50 samples of size 225 to see the fluctuation.

We name:

`N` the number of simulations (one simulation is 225 draws),

`n` the number of times we got a green ball,

`p` the percentage of green balls obtained by this simulation,

L_p the sequence of percentages obtained.

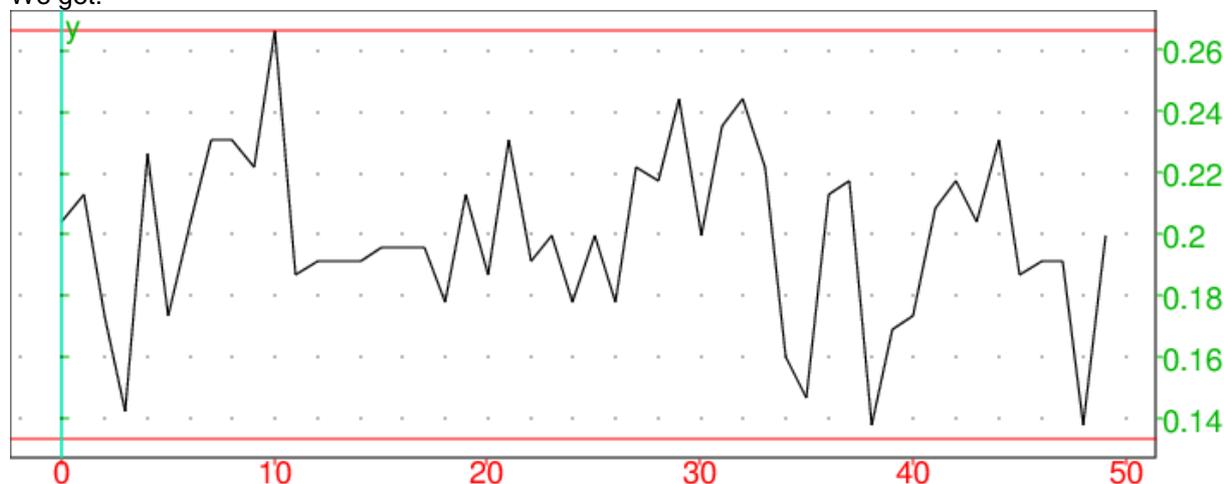
We enter:

```
test0(N):={
local L,p,n,k,Lp;
Lp:=NULL;
for k from 1 to N do
L:=randmultinom(225,[4/5,1/5],["R","V"]);
n:=L[2,2]
p:=n/225.;
Lp:=Lp,p;
fpour;
return Lp;
}::
```

Then:

```
plotlist(test0(50),line(y=2/15),line(y=4/15))
```

We get:



We successively analyse t samples of size $n = 225$ for $t \in 10, 20, 50, 100, 200, 500$. In our case, the fluctuation interval at level of 95% is: $p - \frac{1}{\sqrt{n}}p + \frac{1}{\sqrt{n}}$ with $p = \frac{1}{5}$ and $n = 225$ that is to say $\frac{2}{15}, \frac{4}{15}$.

To check whether the simulation is correct or not, we write a program checking whether we do have or not p laying in the interval $\frac{2}{15}, \frac{4}{15}$ in 95% of the cases.

We note N the number of simulations (one simulation is 225 draws).

For the k -nth simulation, ($k=1 \dots N$), we name:

L the list of 225 draws obtained,

n the number of times we got a green ball,

p the percentage of green balls obtained by this simulation,

s the number of draws such as $2/15 < p < 4/15$ when we have done k simulations,

s_n the number of times we got a green ball when we have done k simulations.

pc_n the percentage of green balls obtained by these $N \cdot 225$ draws, which is then $s_n / (225 \cdot N)$

The number of times we have $2/15 < p < 4/15$ is s , or else the percentage $pc = s/N$.

We check then if $pc > 0.95$

```
test0(N):={
local s,L,p,n,pc,sn,pcn,k,Le;
s:=0;sn:=0;
Le:=NULL;
for k from 1 to N do
L:=randmultinom(225,[4/5,1/5],["R","V"]);
n:=L[2,2]
p:=n/225;
Le:=Le,p;
fpour;
```

```

returns Le;
};
test(N) := {
local s, L, p, n, pc, sn, pcn, k, Le;
s:=0; sn:=0;
Le:=NULL;
for k from 1 to N do
L:=randmultinom(225, [4/5, 1/5], ["R", "V"]);
n:=L[2, 2]
p:=n/225;
Le:=Le, p;
if p>2/15 and p<4/15 then s:=s+1; endif;
sn:=sn+n;
fpour;
pc:=evalf(s/N);
pcn:=evalf(sn/N/225);
if pc>0.95 then returns pcn, pc, "correct"; otherwise returns pcn, pc, "not
correct"; endif;
};

```

We enter:

```
test(10)
```

We get:

```
0.20311111111111, 1.0, "correct"
```

We enter:

```
test(20)
```

We get:

```
0.1948888888889, 0.95, "not correct"
```

We enter:

```
test(50)
```

We get:

```
0.19431111111111, 0.98, "correct"
```

We enter:

```
test(100)
```

We get:

```
0.1988888888889, 0.97, "correct"
```

We enter:

```
test(200)
```

We get:

```
0.1937777777778, 0.99, "correct"
```

```
test(500)
```

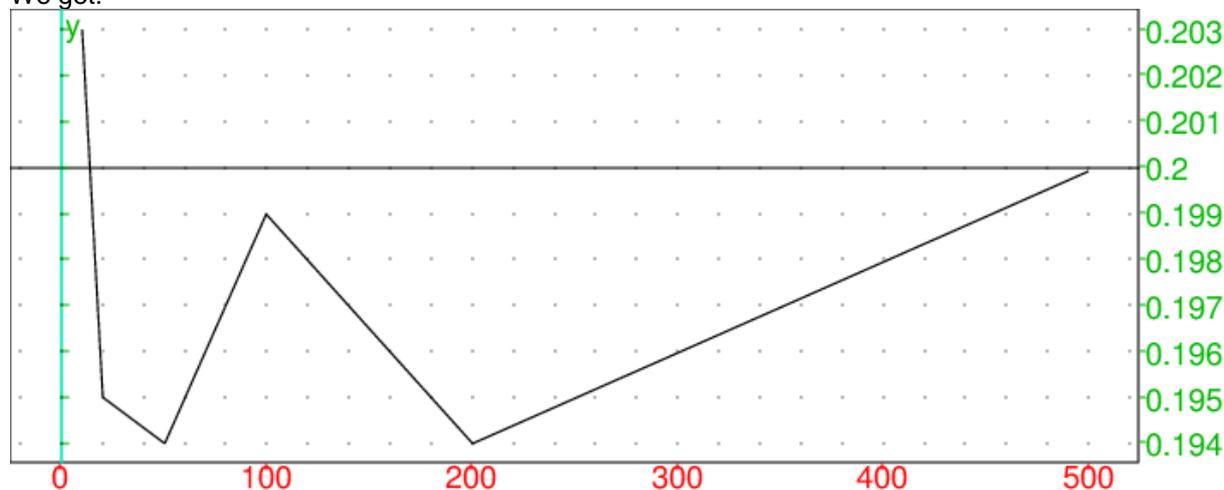
We get:

0.19984,0.984,"correct"

We enter:

```
plotlist([10,20,50,100,200,500],[0.203,0.195,0.194,0.199,0.194,0.1999]
), line(y=0.2)
```

We get:



26.11 Inference application

The application `Inference` allows to do inferential statistics.

Results to keep in mind

If μ is the mean of a population and m_α the mean of means of values of a sample of size n extracted from this population, we have: $m_\alpha = \mu$.

If σ is the standard deviation of a population and σ_α the standard deviation of the mean of means of values of a sample of size n extracted from this population, we have: $\sigma_\alpha = \frac{\sigma}{\sqrt{n}}$.

If σ^2 is the variance of a population and s^2 the mean of variances of values of a sample of size n extracted from this population: $S^2 = n - \frac{1}{n\sigma}$.

So we have $\sigma_\alpha \frac{\sigma}{\sqrt{n}} = \frac{s}{\sqrt{n-1}}$.

When we have a sample of size n , of mean \bar{x} , and standard deviation s , we choose by default: $S = s$ and $m_\alpha = \bar{x}$, and then the mean of values of samples of size n is of mean x and standard deviation $\frac{s}{\sqrt{n-1}}$.

Sum up

- The mean and the variance of a sum are respectively the sum of the means and the sum of the variances.
- The mean and the variance of a difference are respectively the difference of the means and the sum of the variances.
- The mean of the set of means of the values of a sample of size n extracted from a population of mean m is m .
The variance of the set of means of the values of a sample of size n extracted from a population of variance σ^2 is $\frac{\sigma^2}{n}$.
The standard deviation of the set of means of the values of a sample of size n extracted from a population of standard deviation σ is $\frac{\sigma}{\sqrt{n}}$.

- The mean s of the variances of the set of variances of the values of a sample of size n extracted from a population of variance σ^2 is $\frac{n-1}{n}\sigma^2$.
- The standard deviation of the set of means of the values of a sample of size n comes by dividing the mean S of the variances of the set of variances of the values of a sample of size n by $\sqrt{n-1}$:
the set of the means of the values of a sample of size n is of mean m and of standard deviation $\frac{S}{\sqrt{n-1}}$.

26.11.1 Frequency of a parameter and hypothesis based on samples

Comparison of experimental frequency and theoretical frequency

In a population, be p the frequency of a parameter.

Be a sample of n items which frequency of this parameter is f . Let us check whether this sample is extracted from the population or not.

For a sample of n items, be X the random variable counting the size showing the considered parameter of probability p . X follows a binomial law of mean np and standard deviation

$\sqrt{np(1-p)}$. If n is large ($n > 50$), the binomial law can be approximated by the normal distribution. The distribution of the frequencies of the parameter of samples of size n is of mean p and standard deviation $\sqrt{(1-p)n}$.

Example

We count the number of times a player draws an ace when distributing the cards of a 32 card set: for a set of 800 distributions, this player has drawn an ace 180 times. Is this player a cheater?

We have $n = 800$, and the probability to draw an ace is:

$$p = \frac{4}{32} = \frac{1}{8}.$$

So the number X of drawing an ace follows a binomial law of mean $n * p = 100$ and standard

deviation $\sqrt{np(1-p)} = 10\sqrt{\frac{7}{8}} \approx 9.35414346693$.

Let us look for the probability so that $70 < X < 130$, we enter in Home:

$$\text{normald_cdf}(100, 9.354, 130) - \text{normald_cdf}(100, 9.354, 70)$$

We get:

$$0.998659588108$$

Then, we can state that this player is a cheater with a probability to be wrong lower than $\frac{2}{1000}$.

With the calculator:

We press `Symb` and choose:

Method: Hypothesis Test

Type Z-Int: 1 π

Alt Hypoth $\pi \neq \pi_0$

We press `Num` and enter:

$x=180$

$n=800$

$\mu_0=0.125$

$\sigma=50.25$

$\alpha=0.02$

We tap `Calc` of the push buttons and we get:

Result 0 (Assumption rejected at $\alpha=0.02$)

Test Z 8.5523597412

Test

b

p 0.225

Prob. 1.2059722286E-17

Crit Z ± 2.32634787404

Lower 0.0906543...
Upper 0.15934...

We enter in Home:

$(180-100)/9.35414346693$

We get:

8.5523597412

We enter in Home:

`normald_icdf(100,9.35414346693,0.99)`

We get:

121.760991768

We enter in Home:

$(121.760991768-100)/9.35414346693$

We get:

2.32634787407

We enter in Home for $\alpha = 0.002$:

`normald_icdf(100,9.354,0.001), normald_icdf(100,9.354,0.999)`

We get:

71.0939670081,128.906032992

Confidence interval of a proportion or a frequency

How to evaluate the frequency p of a parameter in a population au view of a sample of size n .

Example

We throw a dice 3,000 times and we have get 490 times the six. Can we state that the dice is regular ? We have:

$$n = 3000, f = \frac{490}{3000} \approx 0.163333333333, 1 - f = \frac{2510}{3000} \approx 0.836666666667$$

so $m = nf = 490$ and $s = \sqrt{nf(1-f)} \approx 20.2476336066$

At the level of $\alpha = 0.002$, we enter in Home:

`normald_cdf(3000*49./300,sqrt(3000*49./300*251/300),500) - normald_cdf(3000`

We get:

0.378612513321

So the probability that p is in the interval $[480, 500]$ is 0.378612513321. We can reasonably admit that $p = 1/6$, that is to say that the dice is regular, because, otherwise, the probability to be wrong would be $1 - 0.378612513321 \approx 0.621387486679$

With the calculator:

We press `Symb` and choose:

Method: Confidence interval

Type: Z-Int: 1 π

We press Num and enter:

x=490

n=3000

$\mu_0=0.16666666666667$

$\alpha=0.05$

We tap Calc of the push buttons and we get:

C 0.95

Crit Z ± 1.95996398454

Lower 0.150105122453

Upper 0.176561544214...

With:

C=0.38

We tap Calc of the push buttons and we get:

C 0.38

Crit Z ± 0.495850347347

Lower 0.159986734614

Upper 0.166679932052

Comparison of the frequencies of two samples

We have two samples of sizes n_1 and n_2 for which the frequencies of a parameter are f_1 and f_2 . If f_1 and f_2 are different, how to know whether these two samples come from a same population or not?

If we do the hypothesis that the two samples come from of a same population, we can estimate the frequency p of the entire population by grouping the two samples with $p = \frac{n_1 f_1 + n_2 f_2}{n_1 + n_2}$.

The standard deviation of the distribution of the frequencies of the parameter of the sample 1 is

$$\sqrt{\frac{f_1(1-f_1)}{n_1}} \text{ and the one of the sample 2 is } \sqrt{\frac{f_2(1-f_2)}{n_2}}.$$

We consider the distribution of the values of $f_1 - f_2$, if the two samples come from a same population:

the mean of $f_1 - f_2$ is 0 and the standard deviation s of $f_1 - f_2$ is $\sqrt{\frac{p(1-p)}{n_1} + \frac{p(1-p)}{n_2}}$

Example

Among the 40 class mates of class A, 23 have been admitted, and among the 40 of class B, 17 have been. Are the class mates of classes A and B part of the same population?

If they are, the distribution of frequencies $f_1 - f_2$ follows a normal distribution:

of mean $p = \frac{40}{80} = 0.5$

and standard deviation $s = \sqrt{\frac{2 \times 0.5 \times 0.5}{40}} \approx 0.111803398875$

We know that $f_1 = \frac{23}{40} \approx 0.575$ and $f_2 = \frac{17}{40} \approx 0.425$ then $f_1 - f_2 = 0.15$ and $\frac{f_1 - f_2}{s} \approx 1.3416407865$

We enter:

```
normal_cdf(0,0.111803398875,0.15)-normal_cdf(0,0.111803398875,-0.15)
```

or we enter:

```
normal_cdf(0,1,1.3416407865)-normal_cdf(0,1,-1.3416407865)
```

We get:

0.820287505121

We enter:

1-0.820287505121

We get:

0.179712494879

This means that we can state that the two classes come from the same population, because if we would state the contrary, the probability to be wrong would be 17.97 %
We enter, if we want a result at the level of 5%:

```
normal_icdf(0,0.111803398875,0.025),normal_icdf(0,0.111803398875,0.975)
```

We get:

```
-0.219130635144,0.219130635144
```

We enter, if we want a result at the level of 5%:

```
normal_icdf(0,1,0.025),normal_icdf(0,1,0.975)
```

We get:

```
-1.95996398454,1.95996398454
```

As 0.15 is in the interval $[-0.219130635144, 0.219130635144]$, we can say that, at the level of 5%, the two classes come from the same population.

With the Inference application:

We press **Symb** and choose:

Method: Hypothesis test

Type: Z-Int: $1 \pi_1 - \pi_2$

Alt Hypoth: $\pi_1 \neq \pi_2$

We press **Num** and enter:

$x_1=23$

$x_2=17$

$n_1=40$

$n_2=40$

$\alpha=0.05$

We tap **Calc** of the push buttons and we get:

Result 1 (H_0 not rejected at $\alpha=0.05$)

Test Z 1.3416407865

Test $\Delta \hat{p}$ 0.15

Prob. 0.179712494879

Crit Z ± 1.95996398454

Lower -0.0666513904072

Upper 0.366651390407

We enter in **Home**:

```
normal_icdf(0.15,0.111803398875,0.025),normal_icdf(0.15,0.111803398875,0.975)
```

We get:

```
-0.0691306351442,0.369130635144
```

The sample 1 has as mean:

$f_1 = \frac{23}{40} = 0.575$ and as variance:

$$\frac{f_1(1-f_1)}{n_1} = \frac{0.575 * 0.425}{40}$$

The sample 2 has as mean:

$f_1 = \frac{17}{40} = 0.425$ and as variance:

$$\frac{f_2(1-f_2)}{n_2} = \frac{0.425 * 0.575}{40}$$

We consider the distribution of frequencies $f_1 - f_2$: we can say that it has as mean the difference of means, *i.e.*:

$$0.575 - 0.425 = 0.15$$

and as variance the sum of these two variances, *i.e.*:

$$\frac{2 * 0.425 * 0.575}{40}$$

40

We enter in Home to get the standard deviation of the distribution of frequency $f_1 - f_2$:

$$\text{sqrt}(2 * 0.575 * 0.425 / 40.)$$

We get:

$$0.1105384548472$$

$$\text{normal_icdf}(0.15, 0.1105384548472, 0.025), \text{normal_icdf}(0.15, 0.1105384548472, 0.975)$$

We get the critical lower and upper:

$$-0.0666513904072, 0.366651390407$$

26.11.2 Samples extracted from a normal distribution

Comparison of an experimental mean and a theoretical mean

If the statistical series follows a normal distribution of mean m and standard deviation σ , the distribution of means m_α of samples of size n follows a normal law of mean m and standard deviation

$$\sigma_\alpha = \frac{\sigma}{\sqrt{n}}$$

If S is the mean of standard deviation of samples of size n then we have $\frac{S}{\sqrt{n}} = \frac{\sigma}{\sqrt{n-1}}$ and then $\sigma_\alpha = \frac{S}{\sqrt{n-1}}$

Example

We weight 100 400 g-breads taken randomly in a bakery and we get a mean m_1 of 390 g with a standard deviation s_1 of 50 g.

Does the baker observe the weight of its 400 g breads at the significance level of 95 % ?

Estimate the mean of the population starting from \bar{x} and standard deviation s of this sample.

- It is to compare an experimental mean $m_\alpha = \bar{x}$ to a theoretical mean μ .
The hypothesis (H_0) is that the difference between these 2 means is not significant *i.e.* (H_0): $\bar{x} = \mu$. The alternate hypothesis (H_1) is then (H_1): $\bar{x} \neq \mu$.
The standard deviation s of the sample of 100 elements is 50, we estimate then the standard deviation of the population at:

$$50 * \sqrt{\frac{100}{99}} \approx 50.25$$

Knowing that:

$$U = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

significantly follows a reduced centered law because $n = 100 > 30$.

Then, we consider:

$$U = \frac{390 - 400}{\frac{50}{\sqrt{99}}} \approx -1.98997487421$$

Given α , we look for u_α such as:

$$\text{Prob}(-u_\alpha < U < u_\alpha) = 1 - \alpha.$$

If $\alpha = 0.05$, we enter:

```
normald_icdf(0.975)
```

We get:

```
1.95996398454 ≈ 1.96
```

We enter:

```
normald_icdf(0.025)
```

We get:

```
-1.95996398454 ≈ -1.96
```

So $Prob(-1.96 < U < 1.96) = 0.975 - 0.025 = 0.95 = 1 - 0.05$ and we have $u_\alpha = 1.96$. As $U \notin [-u_\alpha, u_\alpha]$, the hypothesis (H_0) is rejected with a risk of error less than 5%. So, we can say with a risk of error less than 5%, that the baker does not respect the weight of 400 g of its breads.

With the calculator:

We press Symb and we choose:

Hypothesis test

Type Z-Int: 1 μ

Alt Hypoth $\mu \neq \mu_0$

We press Num and enter:

N=100

$\mu=400$

$\sigma=50.25$

$\alpha=0.05$

We tap Calc of the push buttons and we get:

Result 0 (Hypothesis rejected at $\alpha=0.05$)

Test Z -1.99

Test x 390

Prob Z 0.0466

Crit Z ± 1.9599

Lower 390.151

Upper 409.849

We enter in Home:

```
normald_icdf(400,5.025,0.025), normald_icdf(400,5.025,0.975)
```

We get:

```
390.151180978, 409.848819022
```

- It is to find an interval at the level of 5%, of center \bar{x} , in which m is laying with a probability of 95 %.

We choose by default:

$$\bar{x} = m_\alpha$$

$s = s_\alpha = \sigma \frac{\sqrt{n-1}}{\sqrt{n}}$ and $s = \sigma_\alpha$ then the mean of means of samples of size n follows a normal distribution of mean \bar{x} and standard deviation $\frac{\sigma}{\sqrt{n}} = \frac{s}{\sqrt{n-1}}$ Thus $\sigma = \frac{500}{\sqrt{99}} \approx 50.25$.

With the calculator:

We press Symb and we choose:

Inter of Confidence

.Type Z-int: 1 μ

We press Num and enter:

x 390

N=100

$\sigma=50.25$

C=0.95

We tap **Calc** of the push buttons and we get:

```
C 0.95
Crit.Z ± 1.96
Lower 380.1511...
Upper 399.8488...
```

We enter in Home:

```
normald_icdf(390,5.025,0.025), normald_icdf(390,5.025,0.975)
```

We get:

```
380.151180978,399.848819022
```

26.11.3 Samples extracted from a Student distribution

We use the Student law (mostly valid for small samples ($n < 30$)):

The variable $T = \frac{m\alpha - m}{s} \sqrt{n}$ follows a Student law of $n - 1$ degrees of freedom.

Comparison of an experimental mean to a theoretical mean

We can do the previous exercise with the Student law:

- We press **Symb** and we choose:

```
Hypothesis test
Type T-Test: 1 μ
Alt Hypoth μ ≠ μ0
```

We press **Num** and enter:

```
x=390
S=50
N=100
μ0=400
α=0.05
```

We tap **Calc** of the push buttons and we get:

```
Result 0 (Hypothesis rejected at α=0.05)
Test T -2
Test x 390
Prob T 0.048239...
DF=99
Crit T ±1.9842...
Lower 390.0789
Upper 409.921
```

We enter in Home:

```
student_icdf(99,0.025),student_icdf(99,0.975)
```

We get:

```
-1.98421695159,1.98421695159
```

We enter in Home:

```
student_icdf(99,0.025)*50/sqrt(100)+400,
student_icdf(99,0.975)*50/sqrt(100)+400
```

We get:

```
390.078915242,409.921084758
```

- We press **Symb** and we choose:
Inter of Confidence

Type T-int: 1 μ

We press Num and enter:

x: 390

S:=50

N=100

C=0.95

We tap Calc of the push buttons and we get:

C 0.95

DF=99

Crit T ± 1.9842

Min Crit x 380.0789

Maxi Crit x 399.9210

We enter in Home:

```
student_icdf(99,0.025)*5+390, student_icdf(99,0.975)*5+390
```

We get:

```
380.078915242, 399.921084758
```

Part V **Programming**

Chapter 27 Generalities

27.1 Syntax of HOME programs and CAS programs

Following examples are programs using either only HOME commands, either both HOME and CAS commands.

To enter the programming interface, we enter:

Shift Program and then New. of the push buttons.

We get a dialog box with Name and CAS.

We check CAS to write a CAS program and we put ABC as name, by example, then OK and we enter the program.

The syntax of a program is:

- For an HOME program of name ABC:


```
EXPORT ABC(<name of parameters>)
BEGIN
LOCAL <name of local variables>;
<instructions>;
RETURN <result>;
END;
```

With Shift Program we get the list of valid programs among which ABC.
- For a CAS program of name ABC: (we notice that ABC is not shown in the program)


```
(<name of parameters>)->BEGIN
LOCAL <name of local variables>;
<instructions>;
RETURN <result>;
END;
```

With Shift Program we get the list of valid programs among which ABC (CAS).

27.2 Writing a program slightly different from an existing program

You have already written a program and you want to write a slightly different program without rewriting all from scratch. For example, you have written the CAS program PERP returning the equation of the perpendicular to a line through a given point:

```
(GA, GD) ->BEGIN
LOCAL GM, GE, GL;
GE:=element(GD);
IF is_element(GA, GD) THEN
GL:=inter(GD, circle(GA, 1));
RETURN equation(perpen_bisector(GL));
END;
IF angle(GE, GA, GD)==pi/2 OR angle(GE, GA, GD)==-pi/2
THEN RETURN equation(line(GA, GE)); END;
GM:=midpoint(op(inter(GD, circle(GA, GE-GA))));
RETURN equation(line(GA, GM));
END;
```

You want to get the program FOOTP returning the affix of the foot of this perpendicular.

We enter in the CAS:

```
FOOTP:=PERP
```

Then, Shift Program

We see that FOOTP is in the program list. Then, it is enough to edit it and modify it as follows:

```
GA, GD) ->BEGIN
LOCAL GM, GE, GL;
GE:=element(GD);
```

```
IF is_element(GA,GD) THEN
GL:=inter(GD,circle(GA,1));
RETURN affix(GA);
END;
IF angle(GE,GA,GD)==pi/2 OR angle(GE,GA,GD)==-pi/2
THEN RETURN affix(GE); END;
GM:=midpoint(op(inter(GD,circle(GA,GE-GA))));
RETURN affix(GM);
END;
```

We enter:

```
FOOTP(point(1),line(-2,2+2i))
```

We get:

```
2/5+6i/5
```

Chapter 28 Programming instructions

28.1 Variables

28.1.1 Variables names

In HOME, there are no symbolic variables and the names of the variables are predefined, such as:

A..Z for real variables (default value 0),
 L0..L9 for lists (default value { }),
 M0..M9 for matrices (default value [[0]]).

In CAS, apart the names of HOME variables, the names of the variables are strings composed of letters or digits, starting by a letter.

Warning! All names are not valid, some are already used by the system.

In CAS, all the variables are symbolics as long as they are not assigned.

The variables defined in HOME are also valid in the CAS, and the assigned variables defined in the CAS are also valid in HOME.

28.1.2 Comments: `comment //`

`comment` takes as argument a string of characters (take care to use the ") whereas `//` does not require the " but must end by a carriage return character: this means that the argument of `comment`, or what is between `//` and the the carriage return character, is not taken into account by the program and that it is a comment.

We enter the program in a `program editor`, then we validate it with `OK`:

```
f(x) := {comment ("f:R->R^2")
        return [x+1,x-1];}
```

or:

```
f(x) := { //f:R->R^2
        return [x+1,x-1];}
```

We get:

```
the commented definition of the function f(x)=[x+1,x-1]
```

28.1.3 Inputs: `INPUT input InputStr`

`input INPUT` allow to enter expressions, `InputStr` allows to enter strings of characters. `input`, `INPUT` and `InputStr` take as argument the name of a variable (resp. a sequence of names of variables) or a string of characters (strings giving the user instructions on the value to be entered) and the name of a variable (resp. a sequence of strings of characters and a sequence of names of variables).

`input`, `INPUT` and `InputStr` open a dialog box where we can enter the value of the variable supplied as argument and where is displayed, as label, the string of characters supplied as argument (if any).

With `input`, `INPUT`, we can enter numbers or strings of characters (add "...") or names of variables (without "...").

With `InputStr`, we can enter only strings of characters, hence, the "..." are not needed.

We enter:

```
input(a)
```

or:

```
input("a=?", a)
```

We get:

a dialog box where we can enter the value of a

We enter:

12 in this dialog box, then OK

then:

```
a+3
```

We get:

```
15
```

We enter:

```
input("polynomial", p, "value", a)
```

We get:

a dialog box where we can enter the values of p with the label "polynomial" and a with the label "value"

We enter:

```
InputStr(a)
```

or:

```
InputStr("answer=", a)
```

We get:

a dialog box where we can enter the value of a

We enter:

12 in this dialog box, then OK

then:

```
a+3
```

We get:

```
123
```

because a is the string of characters "12", and the "+" sign concatenates the two strings "12" and "3".

28.1.4 Outputs: `print`

`print` takes as argument a sequence of string of characters or of names of variables.

`print` displays the strings of characters and the content of variables on the screen before the answer.

We enter:

```
a:=12
```

then:

```
print("a=",a)
```

We get:

```
"a=",12 is displayed, and the answer is 1
```

28.1.5 Assignment instruction: => :=

To store an object in a variable, in HOME, we use `Sto▶` of the push buttons.

In HOME, we enter for example:

```
12 Sto▶ A
```

as a result, this stores 12 in the variable A.

In CAS, or in a CAS program, we enter for example:

```
12 => a
```

Or we enter

```
a:=12
```

as a result, this stores 12 in the variable a.

In CAS, or in a CAS program, we can write: `a,b:=12,34` or `a,b:=[12,34]`

as a result, this put 12 in the variable a and 34 in the variable b.

For instance, to put the quotient and the remainder of the Euclidean division respectively in q and r by using the CAS command `iquorem`, we enter:

```
q,r:=iquorem(365,12)
```

as a result, this stores 30 in the variable q, and 5 in the variable r, because $365 = 30 * 12 + 5$.

Warning! You must use this carefully because these two assignments are done simultaneously. For example:

```
a:=1;b:=2; a:=a+b;a:=1;b:=a-b;
```

```
a:=1;b:=2;c:=a;a:=a+b;b:=c-b;
```

so, after that, a equals 3 and b equals -1 (and not 1).

BUT

```
purge(a); a,b:=2,a+1
```

stores 2 in a and 3 in b.

28.1.6 Copy without evaluating the content of a variable: CopyVar

`CopyVar` takes as arguments the name of two variables.

`CopyVar copy`, without evaluating it, the content of the first variable in the second variable.

We enter (mind the order):

```
a:=c
```

```
c:=5
```

```
CopyVar(a,b)
```

```
b
```

482

We get:

```
c
```

then we enter:

```
c:=10
```

```
b
```

We get:

```
10
```

A modification of the content of c also modifies the content of b, because b stores c.

We enter:

```
a:=d
```

```
b
```

We get:

```
10
```

We enter:

```
purge(c)
```

```
b
```

We get:

```
c
```

because b stores c.

28.1.7 Function testing the type of its argument: `TYPE` `type`

`TYPE` is an HOME function returning the type of the object supplied as argument. For example:

`TYPE(1)=TYPE(pi)=0`, `TYPE(i)=3`, `TYPE([1,2])=6`, `TYPE({1,2})=6`.

`type` is a CAS function returning the type of the object supplied as argument. For example:

`DOM_FLOAT, DOM_INT, DOM_COMPLEX, ..., DOM_IDENT, DOM_LIST, DOM_SYMBOLIC, DOM_RAT, ..., DOM_SYMBOLIC, DOM_STRING`.

There are 20 different types, each one represented by an integer from 1 to 20.

`type` allows to check for an input error.

We enter:

```
type(3.14)
```

We get:

```
DOM_FLOAT
```

We enter:

```
type(3.14)+0
```

We get:

1

We enter:

type(3)

We get:

DOM_INT

We enter:

type(3)+0

We get:

2

We enter:

type(3% 5)

We get:

15

28.1.8 Function testing the type of its argument: `compare`

`compare` is a function of two arguments returning 1 if they are of different types, or if they are of same type and listed in increasing order of type, and 0 otherwise.

We enter:

compare(1,2)

We get:

1

We enter:

compare(2,1)

We get:

0

We enter:

compare("3","a")

We get:

1

We enter:

compare("a",3)

We get:

0

We enter:

```
compare(3, "a")
```

We get:

1

We enter:

```
compare("a", 3)
```

We get:

0

Indeed, we have: `type(3)=DOM_INT=2` and `type("a")=DOM_STRING=12`

28.1.9 Stating an assumption about a variable: `assume`

`assume` allows to state an assumption on a variable.

`assume` takes as argument a name of variable followed by an equality, or an inequality, representing the assumption stated, or else a name of variable followed by a comma and its type. We can put several assumptions, provided they are linked by `and` or `or`, depending on what we want to do. Though, you must use additionally as second argument of `assume` to specify the type of the variable and a range of values for this variable.

`assume` returns the name of the variable about which we have stated the assumption, or the type of this variable.

Warning! If we state another assumption with `assume`, the previous assumption is deleted: if you want to add another assumption, you must use the command `additionally`, or put `additionally` as second argument of `assume`.

Notes

This allows to do interactive geometry while doing symbolic computations at the same time. For instance, if we put in geometry:

`assume(a=2); assume(b=3); A:=point(a+i*b)`, the figure will be built with the values given to the variables, but the calculations will be performed with the symbolic variables `a` and `b`, because for all the graphic outputs, and these ones only, the variable is evaluated.

We enter:

```
assume(a=2)
```

Or we enter directly:

```
assume(a=[2,-5,5,0.1])
```

We get:

```
a sliding bar allowing to make a vary
```

When we make `a vary`, the command `assume(a=2)` turns into `assume(a=[2.1,-5.0,5.0,0.1])` and the following levels are evaluated. If there is nothing on the following level we will get `undef` as a result.

This means that `a` may vary between `-5` and `5` with a step of `0.1` and that `a` equals `2.1`.

If on the two following levels we have `evalf(a+2)` and `evalf(a+3)`, the answers will evaluate according to the position of the cursor (cursor at `2.1`: we have `4.1` and `5.1`, then cursor at `2.2` we

have 4.2 and 5.2). but if on the two following levels we have $a+2$ and $a+3$, the answers will always be $a+2$ and $a+3$.

To assume that the formal variable a is positive, we enter:

```
assume (a>0)
```

We get:

```
a
```

We enter:

```
assume (a)
```

We get:

```
assume[DOM_FLOAT, [line[0,+(infinity) ]], [0]]
```

this means that a is a real variable laying in $]0; +\infty[$ and that 0 is excluded.

We know the domain, the interval and the excluded values.

To assume that the formal variable a is in $[2; 4[\cup]6; \infty[$, we enter:

```
assume ((a>=2 and a<4) or a>6)
```

We get:

```
a
```

We enter:

```
assume (a)
```

We get:

```
assume[DOM_FLOAT, [[2,4], [6,+(infinity) ]], [4,6]]
```

this means that a is a real variable laying in $[2; 4[\cup]6; \infty[$ and that 4 and 6 are excluded.

We know the domain, the interval, and the excluded values.

We enter:

```
abs(1-a)
```

We get:

```
-1+a
```

We enter to tell that b is an integer:

```
assume (b, integer)
```

We get:

```
DOM_INT
```

We enter:

```
assume (b)
```

We get:

```
[DOM_INT]
```

To tell that b is an integer strictly greater than 5, we enter:

```
assume(b, integer);
assume(b>5, additionally)
```

We get:

```
DOM_INT
```

then

```
b
```

We enter:

```
assume(b)
```

We get:

```
[DOM_INT]
```

Note:

When `assume` takes as argument one single equality and the command is entered from the entry line of the Geometry screen, as a result, this adds a small cursor at the top right of this screen. The name of the parameter is written at the right of the cursor.

This cursor allows to change the value of the parameter, and this value will be written at the left of the cursor.

By example, we enter:

```
assume(a=[2, -10, 10, 0.1])
```

This means that all the calculations will be performed with any value of a, provided that the points have exact coordinates, but also that the figure will be plotted with $a=2$ and that we will be able to have this figure vary thanks to the small cursor according to a from -10 to +10, with a step of 0.1.

If we put `assume(a=[2, -5, 5])`, a varies from -5 to +5 with a step of $(5 - (-5)) / 100$, and if we put `assume(a=2)`, a varies from $WX-$ to $WX+$ and the step is $((WX+) - (WX-)) / 100$.

Warning! As far as geometry is concerned, you have to work with exact coordinates.

For example:

```
A:=point(i); assume(b:=2); B:=point(b);
```

then we enter:

```
length(A, B);
```

We get:

```
sqrt((-b)^2+1)
```

But:

```
A:=point(0.0+i); assume(b:=2); B:=point(b);
```

then we enter:

```
length(A, B);
```

We get the approximate value of $\sqrt{1 + 4}$:

2.2360679775

Warning! A parameter defined by `assume` is evaluated for graphic outputs only, otherwise you must use `evalf`.

Example.

We enter:

```
dr(m) := ifte(m==2, line(x=1), line(x+(m-2)*y-1)) then in a level of geometry, we enter:
assume(a=[2.0, -5, 5, 0.1])
dr(evalf(a))
```

which returns `line(x=2)` when `a:=2` and `line(y=(-5*x+5))` when `a:=2.2`, whereas `dr(a)` returns `line(y=(-1/(a-2)*x+1/(a-2)))` whatever `a` is and this will then lead to an error for `a=2`.

Warning! Mind the difference between `assume` and `element`.

If `b:=element(0..3,1,0.1)` is entered from the entry line of the Geometry screen, this adds a small cursor at the top right of this screen with `b=1` and we will be able to have `b` vary thanks to the small cursor from 0 to 3 with a step of 0.1, but the variable `b` is not formal!

We enter:

```
a;b
```

We get:

```
(a,1)
```

28.1.10 State an additional assumption about a variable: `additionally`

`additionally` allows to state additional assumptions about a variable. Indeed, if we state another assumption with `assume`, the previous assumption is deleted. Thus, if you want to add a new assumption, you must use the command `additionally` or put `additionally` as second argument of `assume`.

`additionally` has the same arguments as `assume`: a name of variable along with an equality or an inequality representing the assumption stated, or else the name and the type of a variable separated by a comma. Several assumptions may be supplied, provided that they are linked by `and` or `or`, depending on what we want to do.

We must use `additionally` to specify the type and the range of values of a variable at the same time.

To tell that `b` is an integer strictly greater than 5, we enter:

```
assume(b, integer);
additionally(b>5)
```

or else

```
assume(b, integer);
assume(b>5, additionally)
```

We get:

```
DOM_INT
```

then

```
b
```

We enter:

```
assume(b)
```

We get:

```
[DOM_INT]
```

28.1.11 Know the assumptions stated about a variable: `about`

`about` takes as argument a name of variable.

`about` allows to know the assumptions stated about this variable.

We enter:

```
assume(a, real); additionally(a>0)
```

or

```
assume(a, real); assume(a>0, additionally)
```

then,

```
about(a)
```

We get:

```
assume[DOM_FLOAT, [0, +(infinity)], [0]]
```

`assume[]` means that we have a list of a specific type.

The last 0 means that 0 is excluded from the interval $[0, +(infinity)]$.

We enter:

```
assume(b, real); additionally(b>=0 and b<2)
```

or

```
assume(b, real); assume(b>=0 and b<2, additionally)
```

then,

```
about(b)
```

We get:

```
assume[DOM_FLOAT, [0, 2], [2]]
```

The last 2 means that 2 is excluded from the interval $[0, 2[$.

We enter:

```
about(x)
```

We get:

```
x
```

which means that `x` is a formal variable.

28.1.12 Delete the content of a variable: `purge`

`purge` allows to delete the content of a variable or to cancel an assumption stated about this variable.

We enter:

```
purge (a)
```

If `a` is not assigned, we get in direct mode "a not assigned", otherwise the previous value is returned (or the assumptions stated on this variable) and the variable turns back to formal with no assumption.

We can also enter:

```
purge (a, b)
```

to delete the content of variables `a` and `b`.

28.1.13 Delete the content of all the variables: `restart`

`restart` allows to delete the content of all the variables and to cancel the assumptions stated about these variables.

We enter:

```
A:=point(1+i); assume(n>0);
```

then

```
restart
```

We get:

```
[A, n]
```

if the variables `[A, n]` would have been the only assigned variables.

28.1.14 Access to answers: `Ans ans (n)`

`Ans` (Shift +) or `Ans()` designates the latest answer, `ans` must be used when working without modifying the lines already validated. Indeed, the questions and the answers are numbered starting from 0, and this number does not correspond to the entry lines numbers. Indeed, we can, for example, modify the first line after having already entered 4 other lines, and this modification will be numbered 4.

If $n \geq 0$, `ans (n)` designates the answer of number $n + 1$,
and,

if $n < 0$, `ans (n)` designates the $(-n)$ -nth previous answer.

Then:

`ans (0)` designates the first answer (the one corresponding to the first requested command).

Warning! If you have deleted some levels, the answers of these levels are not deleted and are taken into account by `ans (n)`.

28.2 Conditionnal instructions

– IF

```
IF < cond > THEN < inst1 > END
```

If the condition `< cond >` is true, the instructions `< inst1 >` are executed, otherwise nothing is done.

We enter:

```
3=>X
```

```
IF X>0 THEN X+1 END
```

```
or
```

```
IF X>0 THEN X+1;END
```

We get:

3

We enter:

-3=>X

IF X>0 THEN X+1 END

or

IF X>0 THEN X+1;END

We get:

-3

- IFTE

IFTE (*cond*, *inst1*, *inst2*)

If the condition supplied as first argument is true, the second argument is executed, otherwise the third argument is executed.

We enter:

3=>X

IFTE (X>0, X+1, X-1)

We get:

4

To define the absolute value, we enter:

-3=>X

IFTE (X>0, X, -X)

We get:

3

We enter:

EXPORT TRIAL0 (X, A)

BEGIN

RETURN IFTE (X<-ABS (A) , -1, IFTE (X<ABS (A) , 0, 1)) ;

END;

Then, we enter:

TRIAL0 (-5, 3)

We get:

-1

We enter:

TRIAL0 (-2, 3)

We get:

0

We enter:

TRIAL0 (5, 3)

We get:

1

- IF THEN ELSE END

IF < *cond* > THEN < *inst1* > ELSE < *inst2* > END

If the condition < *cond* > is true, the instructions < *inst1* > is executed, otherwise the instructions < *inst2* > is executed.

We enter:

3=>X

IF X>0 THEN X+1 ELSE X-1 END

or

IF X>0 THEN X+1; ELSE X-1; END

We get:

3

We enter:

-3=>X

IF X>0 THEN X+1 ELSE X-1 END

or

IF X>0 THEN X+1; ELSE X-1; END

We get:

-4

We enter:

EXPORT TRIAL (X, A)

BEGIN

```
IF X<-ABS(A) THEN RETURN -1; END;
IF X<ABS(A) THEN RETURN 0; END;
RETURN 1;
END;
```

Then, we enter:

```
TRIAL(-5, 3)
```

We get:

```
-1
```

We enter:

```
TRIAL(-2, 3)
```

We get:

```
0
```

We enter:

```
TRIAL(5, 3)
```

We get:

```
1
```

– CASE

```
CASE ... END
```

```
CASE
```

```
IF < cond1 > THEN < inst1 > END;
```

```
IF < cond2 > THEN < inst2 > END;
```

```
IF < cond3 > THEN < inst3 > END;
```

```
DEFAULT < inst4 >;
```

```
END
```

We use CASE to avoid using nested IF.

< cond1 > is evaluated:

- if < cond1 > is true, the instructions < inst1 > are executed, and we end CASE by doing the instructions following END of CASE.
- if < cond1 > is false, then < cond2 > is evaluated, if it is true the instructions < inst2 > are executed and we end CASE by doing the instructions following END of CASE, etc., ... The instruction < inst4 > is done if the three conditions < cond1 >, < cond2 >, < cond3 > are false.

```
CASE
```

```
IF X<-1 THEN -1=>R; END;
```

```
IF X<1; THEN 0=>R; END;
```

```
IF X>=1 THEN 1=>R; END;
```

```
END;
```

```
R;
```

or else:

```
CASE
```

```
IF X<-1 THEN -1=>R; END;
```

```
IF X<1 THEN 0=>R; END;
```

```
DEFAULT 1=>R;
```

```
END;
```

```
R;
```

– IFERR

The syntax is:

```
IFERR < inst0 > THEN < inst1 > ELSE < inst2 > END
```

If an error is detected in the instructions < inst0 >, the instructions < inst1 > are executed, otherwise the instructions < inst2 > are executed.

We enter (for example if we do not know the order of the arguments of the command POS):

```
IFERR(A:=POS(5, [1, 3, 5, 2, 4])); THEN
```

```
A:=POS([1, 3, 5, 2, 4], 5); ELSE
```

```
A:=POS(5, [1, 3, 5, 2, 4]);
```

```
END
```

We get:

```
4
```

– **CONTINUE**

When `CONTINUE;` is among instructions of an iteration, this leads to skip the instructions which follows it and go to the next iteration.

We enter to calculate $1 + 2 + 4 + 5 = \sum_{j \neq 3 \text{ and } j=1}^5 j$:

```
A:=0;
FOR J FROM 1 TO 5 DO
IF J==3 THEN CONTINUE; END;
A:=A+J;
END;
We get:
12
```

28.3 Loops

28.3.1 Instructions `FOR FROM TO DO END` and `FOR FROM TO STEP DO END`

We enter:

```
S:=0; FOR J FROM 1 TO 5 DO S:=S+J;END
```

We get:

15

because $1 + 2 + 3 + 4 + 5 = 15$

We enter:

```
S:=0; FOR J FROM 2 TO 10 STEP 2 DO S:=S+J;END
```

We get:

30

because $2 + 4 + 6 + 8 + 10 = 30$

28.3.2 Iterative loops: `ITERATE`

To do an iteration, we enter:

```
ITERATE (X^2, X, 2, 3)
```

this means that $X:=2$; `FOR J FROM 1 TO 3 DO X^2=>X;END;`

We get:

256

because X equals 2 then $2^2 = 4$ then $4^2 = 16$ then $16^2 = 256$

28.3.3 Instruction `WHILE DO END`

We enter:

```
A:=1; WHILE A<=1 DO A:=A+1; END;A;
```

We get:

2

We enter:

```
S:=0;J:=1;WHILE J<=5 DO S:=S+J;J:=J+1; END;S
```

We get as value of S:

15

28.3.4 Instruction `REPEAT UNTIL`

We enter:

```
A:=1; REPEAT A:=A+1 UNTIL A>1;A;
```

We get as value of A:

2

We enter:

```
A:=1; REPEAT A:=A+1 UNTIL A>4;A;
```

We get as value of A:

5

28.3.5 Instruction BREAK

We enter:

```
BREAK
```

We get:

The exit from a loop

For example, to get the approximate value of the sum $\sum_{j=1}^{\infty} \frac{1}{j^2}$, we decide to not add the terms lower than P and to not do more than 100 additions.

We enter:

```
EXPORT PI2S6(P)
BEGIN
LOCAL J,S,U;
FOR J FROM 1 TO 100 DO
U:=1/J^2;
IF U<P THEN
BREAK;
END;
S:=S+U;
END;
RETURN S;
END;
```

We enter:

```
PI2S6(0.001)
```

We get:

```
1.61319070033
```

We enter:

```
PI2S6(0.0001)
```

We get:

```
1.63498390018
```

We enter:

```
PI^2/6
```

We get:

```
1.64493406685
```

28.3.6 Function seq

`seq` is not an instruction but a function which allows to return the list constituted by the different values of the first argument when the second argument varies according to the values of following arguments: start value, end value, step (by default, step=1).

```
seq(f(k),k,1,3)=[f(1),f(2),f(3)]
```

```
seq(f(k),k,1,5,2)=[f(1),f(3),f(5)]
```

The function `seq` is useful to plot a series of points on the Geometry screen.

Example

We want to represent the ten first terms of the sequence:

$u_n = \left(1 + \frac{1}{n}\right)^n = f(n)$ by the points $n + i * f(n)$.

We open the Geometry application and we enter:

```
f(n):=(1+1/n)^n
```

```
seq(point(k+i*f(k)),k,1,10)
```

We get:

We see the points on this Geometry screen

If we enter:

```
for (k:=1;k<11;k++) {point(k+i*f(k));}
```

We get:

only the last point

but if we enter:

```
L:=[];for (k:=1;k<11;k++)
{L:=append(L,point(k+i*f(k)));};L;
```

We get:

We see the points on the Geometry screen

28.4 Comments: //

// starts a line intended to be a comment.

28.5 Variables

In programming, the variables have as names a string of letters or numbers starting by a letter. The variables which are locals to the program will be declared by using the key word `LOCAL`, for example: `LOCAL A,B,AB,x;`

In this case, the variables are set to 0.

To get a formal variable, we write: `x:='x'`.

28.6 Boolean operators: < <= == != > >=

<, <=, >, >= are boolean infix operators checking for inequality.
 == is a boolean infix operator checking for equality.
 <> or != or ≠ is a boolean infix operator checking for non equality.

– **AND and**

`AND` or `and` is the boolean infix operator *and*.

We enter:

1 AND 0

We get:

0

We enter:

1 AND 1

We get:

1

We enter:

0 AND 0

We get:

0

– **NOT**

`NOT` returns the logic inverse of the argument.

We enter:

NOT 1

We get:

0

We enter:

NOT 0

We get:

1

- **OR or**
OR or or is the boolean infix operator *or*.
We enter:
1 OR 0
We get:
1
We enter:
1 OR 1
We get:
1
We enter:
0 OR 0
We get:
0
- **XOR**
XOR is the boolean infix operator exclusive *or*.
We enter:
1 XOR 0
We get:
1
We enter:
1 XOR 1
We get:
0
We enter:
0 XOR 0
We get:
0

Input commands

- **CHOOSE**
To choose the value of A among the three values (1,2,3), we enter:
CHOOSE(A, "TITLE:A=", "ONE", "TWO", "THREE")
We get:
A dialogue opens displaying three items:
if we pick on the first (resp. the second, the third) item, this
stores 1 (resp. 2, 3) in A
- **FREEZE**
We enter:
FREEZE
We get:
the screen freezes, we press a key to quit
- **GETKEY**
We enter:
A:=GETKEY
We get, if we pressed . :
48
We enter:
A:=GETKEY
We get, if we did not press any key:
-1
- **ISKEYDOWN**
We enter:
ISKEYDOWN(48)
We get, if we did not press the key . :
0
We enter:
ISKEYDOWN(48)
We get, if we have pressed the key . :
1

- **INPUT**
 We enter:
 INPUT (C, "TITLE:C=")
 We get:
 A screen allowing to enter a value to be stored in the variable C
- **MSGBOX**
 We enter:
 A:=3
 MSGBOX (2*A)
 or
 MSGBOX (2*A, 0)
 We get:
 6 and OK in the push buttons
 If we tap OK then MSGBOX (2*A) or MSGBOX (2*A, 0) returns 1.
 We enter:
 A:=3
 MSGBOX (2*A, 1)
 We get:
 6 and CANCEL and OK in the push buttons
 If we press CANCEL then MSGBOX (2*A, 1) returns 0.
 If we press OK then MSGBOX (2*A, 1) returns 1.
 We enter:
 A:=3
 MSGBOX ("A= "+A)
 We get:
 "A= 3"
- **PRINT**
 We enter:
 A:=3
 PRINT (A)
 We get:
 A: 3
- **WAIT**
 We enter:
 WAIT (5)
 We get:
 A 5 seconde pause of the program
- **EDITMAT**
 We enter:
 EDITMAT (M)
 We get:
 A matrix editor opens to enter the matrix M

An example of use of GETKEY and ISKEYDOWN.

The following program let us know the code of each key pressed and ends when pressing the key ..

```
EXPORT AA()
BEGIN
LOCAL A, L;
L:=[];
REPEAT
REPEAT
A:=GETKEY;
UNTIL A!=-1;
L:=CONCAT(L, A);
UNTIL ISKEYDOWN(48);
RETURN L;
END;
```

We enter:

AA() then Enter 1230.

We get:

[42, 43, 44, 47, 48]

We can also write the following program, which gives the same result:

```
EXPORT AAA()
BEGIN
LOCAL A, L, N;
L:=[];
N:=0
WHILE N==0 DO
REPEAT
A:=GETKEY;
UNTIL A!=-1;
L:=CONCAT(L, A);
N:=ISKEYDOWN(48);
END;
RETURN L;
END;
```

We enter:

AAA() then Enter 1230.

We get:

[42, 43, 44, 47, 48]

Example of a dice roll: two players A and B roll alternately two dices and keep their scores SA and SB, putting aside the score of some rolls (for example some giving at least one 6). They decide to stop playing after two minutes.

The program will display the result of the roll of each of the two players. We notice that the display of the roll of player B: `MSGBOX(N+1+":B="+B, 1)` takes as second parameter 1. CANCEL and OK are displayed in the push buttons. If we press CANCEL, `MSGBOX(N+1+":B="+B, 1)` returns 0 and otherwise `MSGBOX(N+1+":B="+B, 1)` returns 1. So if we press CANCEL we cancel the play.

To stop playing, it is enough to press OK (key 5).

The program displays then the score and the list of the rolls.

```
EXPORT TWODICES()
BEGIN
LOCAL SA, SB, A, B, C, N, L;
SA:=0;
SB:=0;
N:=0;
L:=[];
RANDSEED
WHILE ISKEYDOWN(5)==0 DO
A:=(RANDOM 6+RANDOM 6);
MSGBOX(N+1+":A="+A, 0);
B:=(RANDOM 6+RANDOM 6);
C:=MSGBOX(N+1+":B="+B, 1);
IF C==--1 THEN
L[N]:= [A, B];
N:=N+1;
IF A>B THEN
SA:=SA+1;
ELSE
SB:=SB+1;
END;
END;
END;
RETURN SA, SB, L;
END;
```

28.7 Commands of applications

— CHECK

If the current application is Function, we enter:

F2(X) :=COS(X)+X

CHECK(2)

We get:

The definition of F2 in the Function application and the function F2 is checked

– **UNCHECK**

If the current application is Function, we enter:

UNCHECK (2)

We get:

The function F2 is unchecked

– **STARTVIEW**

We enter:

STARTVIEW(1)

We get:

black:	0
drak grey:	1
light grey:	2
white:	3
Symbolic:	0
Plot:	1
Numeric:	2
Symbolic Setup:	3
Plot Setup:	4
Numeric Setup:	5
First special view (Split Screen Plot Detail):	6
Second special view (Split Screen Plot Table):	7
Third special view (Autoscale):	8
Fourth special view (Decimal):	9
Fifth special view (Integer):	10
Sixth special view (Trig):	11
HomeScreen:	-1
Home Modes:	-2
Memory Manager:	-3
APP Library:	-4
Application Note Editor:	-5
MatrixCatalog:	-6
ListsCatalog:	-8
ProgramCatalog:	-10
Note Catalog:	-12

Chapter 29 How to program

29.1 Conditional instruction IF

Three stores selling the same wools at the same unit price of \$ p , decide to offer rebates.

In the store 1, the conditions of the rebates are:

- 10 % rebate when buying over 5 to less than 10 balls of wool,
- 20 % rebate when buying at least 10 balls.

In the store 2, the conditions of the rebates are:

- 1 ball free for 8 bought
- 2 balls free for 13 bought.

In the store 3, the conditions of the rebates are:

- 10 % rebate when buying 5 balls,
 - 20 % rebate when buying 10 balls,
- for example, if you buy 7 balls at \$ p one, you get the rebate on 5 balls only and pay $2 * p + 5 * 0.9 * p = 6.5 * p$ and if you buy 17 balls, you get the rebate on 10 balls only and on 5 balls and you pay $2 * p + 5 * 0.9 * p + 10 * 0.8 * p = 14.5 * p$.

You need 9 balls, which store do you choose?

You need 15 balls, which store do you choose?

Write the program `price(n, p)` for each store, returning the price to pay, the number of balls and the money spared thanks to the rebate, when buying n balls at a unit price p .

The programs

```
- price1
    (n, p) -> BEGIN
    LOCAL p1, p2, n1, n2, r1;
    p1 := 0.8 * p;
    p2 := 0.9 * p;
    r1 := irem(n, 10);
    IF n < 5 THEN return n * p, n, 0; END;
    IF 5 <= n and n < 10 THEN return n * p2, n, n * (p - p2); END;
    IF n > 10 THEN return n * p1, n, n * (p - p1); END;
    END;

- price2
    (n, p) -> BEGIN := {
    LOCAL r1, q1;
    r1 := irem(n, 13);
    q1 := iquo(n, 13);
    IF r1 >= 8 THEN return n * p, n + 2 * q1 + 1, 2 * q1 * p + p; END;
    IF r1 < 8 THEN return n * p, n + 2 * q1, 2 * q1 * p; END;
    END;

- price3
    (n, p) -> BEGIN
    LOCAL p1, p2, n1, n2, r1;
    p1 := 0.8 * p;
    p2 := 0.9 * p;
    r1 := irem(n, 10);
    IF r1 < 5 THEN n1 := n - r1; return n1 * p1 + r1 * p, n, (n - r1) * p - n1 * p1; END;
    IF r1 >= 5 THEN n1 := n - r1; return n1 * p1 + 5 * p2 + (r1 - 5) * p, n, (n - r1 + 5) * p -
    n1 * p1 - 5
```

```
END;
```

29.2 FOR and WHILE loops

29.2.1 Make the calculator count by step of one and display the result

We want the calculator display: 0, then 1, then 2, etc., ...

When not in a program, we enter:

```
n:=0;
```

then, Enter

We enter:

```
n:=n+1
```

then, Enter, Enter, etc., ...

We get (each press of Enter displays the following number):

1, then 2, etc., ...

With a CAS program

The FOR loop

We name the program `countf`, and check CAS.

In order to display the sequence $0, 1, 2, \dots, p$, we enter a program using a FOR loop:

```
(p) ->BEGIN
LOCAL n;
  FOR n FROM 0 TO p DO
    PRINT (n) ;
  END;
RETURN n;
END;
```

We notice that:

n is initialized by the value following FROM, and the value following TO is used to do the loop stop test. The instruction $n:=n+1$, and then the test $n \leq p$ is automatically done in a FOR loop. The loop stops when the first integer n strictly greater than p is reached.

We enter in the CAS:

```
countf(-1)
```

We get:

0, because the test $n \leq p$ is done at the beginning of the loop.

We enter in the CAS:

```
countf(4)
```

We get:

```
0
```

```
1
```

```
2
```

```
3
```

```
4,
```

then 5, because 5 is the first integer strictly greater than $p=4$

Sum up

FOR initializes the variable of the FOR, does the test, if true executes the body of the loop (i.e. all the instructions up to the END of the FOR), then the variable the FOR is incremented, then the test is done: if true executes the body of the loop etc.; and if false executes the instructions following the END of the FOR.

The WHILE loop

We name the program `countw` and we check CAS.

In order to display the sequence $0, 1, 2, \dots, p$, we enter a program using a WHILE loop:

```
(p) ->BEGIN
```

```

LOCAL n;
n:=0;
WHILE n <=p DO
    PRINT(n);
    n:=n+1;
END
RETURN n;
END;

```

We notice that:

n must be initialized before the beginning of the loop,

`WHILE` does the test $n \leq p$:

if the test is true it executes the body of the loop (*i.e.* all the instructions up to the `END` of the `WHILE`)

but, be careful, you have to change

in the body of the loop the value of at least one variable of the test, so that the test is false at a time,

to avoid to get an infinite loop, in this case the instruction $n := n + 1$. Then, `WHILE` does the test $n \leq p$,

if true, executes the body of the loop, etc., ..., otherwise executes the instructions following the `END` of the `WHILE`.

The loop stops when the first integer n strictly greater than p is reached.

We enter in the CAS:

```
countw(-1)
```

We get:

0, because the test $n \leq p$ is done at the beginning of the loop.

We enter in the CAS:

```
countw(4)
```

We get:

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

then 5, because 5 is the first integer strictly greater than $p=4$

29.2.2 Make the calculator count by step of 1 by using a list or a sequence

Rather than displaying the numbers with the command `PRINT`, we will put these numbers into a list or a sequence.

With a list

The empty list is `[]` and the command `l:=append(l,a)` adds the element a at the end of the list l .

With a FOR loop

We name the program `countlf` and we check CAS.

In order to display the list `[0,1,2...p]`, we enter the program using a `FOR` loop:

```

(p)->BEGIN
LOCAL n,l;
l:=[];
FOR n FROM 0 TO p DO
    l:=append(l,n);
END;
RETURN l;
END;

```

With a WHILE loop

We name the program `countlw` and we check CAS.

In order to display the list `[0,1,2...p]`, we enter the program using a `WHILE` loop:

```

(p)->BEGIN
LOCAL n,l;
l:=[];
n:=0;
WHILE n <=p DO

```

```

        l:=append(l,n);
        n:=n+1;
END
RETURN l;
END;

```

We enter:

```
countlf(4) or countlw(4)
```

We get:

```
[0,1,2,3,4]
```

With a sequence

The empty sequence is `NULL` and the command `l:=l,a` adds the element `a` at the end of the sequence `l`.

With a FOR loop

We name the program `countsf` (or we modify the previous program) and we check CAS. In order to display the list `[0,1,2..p]`, we enter the program using a `FOR` loop:

```

(p)->BEGIN
LOCAL n,l;
l:=NULL;
FOR n FROM 0 TO p DO
    l:=l,n;
END;
RETURN l;
END;

```

With a WHILE loop

We name the program `countsw` (or we modify the previous program) and we check CAS. In order to display the list `[0,1,2..p]`, we enter the program using a `WHILE` loop:

```

(p)->BEGIN
LOCAL n,l;
l:=NULL;
n:=0;
WHILE n <=p DO
    l:=l,n;
    n:=n+1;
END;
RETURN l;
END;

```

We enter:

```
countsf(4) or countsw(4)
```

We get:

```
0,1,2,3,4
```

29.3 Approximate value of the sum of a sequence

29.3.1 Sequence of general term $u_n = \frac{1}{n^2}$

With no program

We enter in the CAS:

```
s:=0;n:=1;
```

Then, Enter

We enter then:

```
s:=s+1/n^2;n:=n+1
```

Then, Enter, Enter, etc., ...

We get (each press of Enter does an additional addition):

```
[1, 2]
[5/4, 3]
[49/36, 4]
[205/144, 5]
[5269/3600, 6]
[5369/3600, 7]
[266681/176400, 8] etc., ...
```

With a CAS program

We name the program `sumu` and we check CAS.

We enter, to get the sum of $1 + \frac{1}{2^2} + \dots + \frac{1}{p^2}$, the program:

```
(p) ->BEGIN
LOCAL s, n;
s:=0;
FOR n FROM 1 TO p DO
  s:=s+1/n^2;
  PRINT([s, n]);
END;
RETURN [s, n];
END;
```

We enter in the CAS:

```
sumu(7)
```

We get:

```
[1, 1]
[5/4, 2]
[49/36, 3]
[205/144, 4]
[5269/3600, 5]
[5369/3600, 6]
[266681/176400, 7]
[266681/176400, 8]
```

We notice that we would need to put `PRINT([s, n+1]);` to get the same results because in the `FOR` loop the incrementation of `n` is done when the body of the loop has been executed, but the final result is the same because the incrementation of `n` has been done and since $8 > 7$, the `FOR` loop stops.

29.3.2 Sequence of general term $v_n = \frac{(-1)^{n+1}}{n}$

With no program

We enter in the CAS:

```
s:=0;n:=1;
```

then, Enter

We enter then:

```
s:=s+(-1)^(n+1)/n;n:=n+1
```

Then, Enter, Enter, etc., ...

We get (each Enter does an additional operation):

```
[1, 2]
[1/2, 3]
[5/6, 4]
[7/12, 5]
[47/60, 6]
[37/60, 7]
[319/420, 8] etc.
```

With a CAS program

We name the program `sumv` and we check CAS.

We enter, to get the sum of $1 - \frac{1}{2} + \frac{1}{3} \dots + \frac{(-1)^{p+1}}{p}$, the program:

```
(p)->BEGIN
LOCAL s,n;
s:=0;
FOR n FROM 1 TO p DO
  s:=s+(-1)^(n+1)/n;
  PRINT([s,n]);
END
RETURN [s,n];
END;
```

We enter in the CAS:

`sumu(7)`

We get:

```
[1,1]
[1/2,2]
[5/6,3]
[7/12,4]
[47/60,5]
[37/60,6]
[319/420,7]
[319/420,8]
```

29.3.3 The sequence of general term $w_n = \frac{1}{n}$ is divergent

To demonstrate this, we show that for $p > 1$ we have:

$$\sum_{n=2^{p-1}}^{2^p} \frac{1}{k} \geq \frac{1}{2}$$

With no program

We enter in the CAS:

`s:=1;n:=1;k:=0;`

`Then,Enter`

We enter then:

`s:=s+sum(1/k,k=n+1..2*n);n:=2*n;k:=1+k;1+k/2<=s`

`Then,Enter,Enter etc ...`

We get (each press of `Enter` does an additional operation):

```
[3/2,2,1]
[25/12,4,1]
[761/280,5,1]
[2436559/720720,6,1] etc.
```

With a CAS program

We name the program `sumdiv` and we check CAS.

To get the sum of n , we enter in the CAS:

```
sumdiv(p)->BEGIN
LOCAL s,n,k;
s:=1;
n:=1;
FOR k FROM 1 TO p DO
  s:=s+sum(1/k,k=n+1..2*n);
  n:=2*n;
  1+k/2<=s;
  PRINT([s,n,k,1+k/2,1+k/2<=s]);
END
RETURN [s,n,1+(k-1)/2<=s];
END;
```

We enter in the CAS:

```
sumdiv(7)
We get:
3/2,2,1,true]
[25/12,4,2,true]
[761/280,8,3,true]
[2436559/720720,16,4,true]
then, [2436559/720720,16,5,true]
```

Exercise

Which value of n makes $\sum_{k=1}^n \frac{1}{k} > p$?

To get the sum of n , we enter in the CAS:

```
sumsup(p) -> BEGIN
LOCAL s,n;
s:=0;
n:=0;
WHILE s<p DO
n:=n+1;
s:=s+1/n;
END;
RETURN evalf(s),n;
END;
```

We enter in the CAS:

```
sumsup(4)
We get:
4.02724519544,31
```

29.4 Decimal form of a fraction

29.4.1 With no program

For example, we want to find the first decimal of $f = \frac{355}{113}$

(f is a fraction giving π with 6 exact decimal places)

We enter in the CAS:

```
f:=355/113;f1:=floor(f);l:=f1;n:=numer(f-f1);d:=denom(f-f1);
```

Then, Enter

We enter then:

```
ds,n:=iquorem(10*n,d); L:=L,ds;
```

Then, Enter, Enter, etc., ...

We get (each press of Enter gives one more decimal place):

```
[[1,47],3,1]
[[1,47],3,1,4]
[[1,47],3,1,4,1]
[[1,47],3,1,4,1,5]
[[1,47],3,1,4,1,5,9]
[[1,47],3,1,4,1,5,9,2]
```

The decimal places obtained are in the list l , beginning by the integer part of the fraction f .

Or else we enter:

```
f:=355/113;f1:=floor(f);L:=f1;n:=numer(f-f1);d:=denom(f-f1);
```

Then, Enter

We enter then:

```
ds,n:=iquorem(10*n,d):: L:=L,ds::
```

Then, Enter, Enter, etc., ...

We get:

```
["Done","Done"],["Done","Done"] etc.
```

Then, we enter:

```
l
```

We get, after having pressed Enter 30 times:

```
[3,1,4,1,5,9,2,9,2,0,3,5,3,9,8,2,3,0,0,8,8,4,9,5,5,7,5,2,2,1,2]
```

As it is difficult to count the number of times we have pressed `Enter`, we can display, by example, five more decimal places each time we press `Enter`. The first `Enter` pressed displays the integer part followed by five decimal places. We may also wish or not to display ds and n .

We enter:

```
f:=355/113;f1:=floor(f);L:=f1;n:=numer(f-f1);d:=denom(f-f1);
```

```
Then, Enter and then, ds,n:=iquorem(105*n,d); l:=1*10^5+ds;
```

or, if we do not want the values of ds and n :

```
ds,n:=iquorem(10^5*n,d);; l:=1*10^5+ds;
```

We get after the first `Enter`:

```
[14159 33],314159
```

```
(or ["Done","Done"],314159)
```

We notice that: $113 * 314159 + 33 = 355 * 10^5$

If we press 8 times `Enter` we get a 41-digit number: the integer part 3, followed by the 40 decimal places of $355/113$:

```
[5309 83],31415929203539823008849557522123893805309
```

If we enter:

```
l
```

We get:

```
31415929203539823008849557522123893805309
```

Note that 5309 has only 4 digits, so the last decimal places are: 05309 and we have:

$$113 * l + 83 = 355 * 10^{40}$$

29.4.2 With a CAS program

We name the program `decimal` and we check CAS.

The program `decimal` returns a sequence `l` giving the integer part (f_1) and the p first decimal places (ds) of a fraction f . We use the following functions:

`floor`, which gives the integer part of a number,

`numer`, which gives the numerator of a simplified fraction,

`denom`, which gives the denominator of a simplified fraction,

`iquorem`, which gives the quotient and the remainder of the Euclidean division

`ds,n:=iquorem(10*n,d)`; is equivalent to:

`ds:=iquo(10*n,d)`; (to get the quotient of $10*n$ by d) and `n:=irem(10*n,d)`;

(to get the remainder of $10*n$ by d).

We enter, `Shift Program`, then `New` of the push buttons.

We get a dialog box with `Name` and `CAS`. We check `CAS` and put as `Name`: `decimal`, then `OK`, and we enter the program which gives the integer part and the p decimal places, one by one, of the rational number f :

```
(f,p)->BEGIN
LOCAL n,d,l,f1,j,ds;
f1:=floor(f);
l:=f1;
n:=numer(f-f1);
d:=denom(f-f1);
FOR j FROM 1 TO p DO
    ds,n:=iquorem(10*n,d);
    l:=l,ds;
END;
RETURN l;
END;
```

We enter in the CAS:

```
decimal(355/311,20)
```

We get:

```
1,1,4,1,4,7,9,0,9,9,6,7,8,4,5,6,5,9,1,6,3
```

We enter in the CAS:

```
decimal(355/113,20)
```

We get an approximation of π to the next $3 * 10^{-7}$:

```
[3,1,4,1,5,9,2,9,2,0,3,5,3,9,8,2,3,0,0,8,8]
```

We have indeed: `evalf(pi)` which returns 3.1415926536

We can determinate the decimal places by p groups of g decimal places and return an integer l . The decimal form of the fraction is then $l * 10^{(-p * g)}$. We name the program `decimalg` and we check CAS (or we modify the program `decimal`):

```
(f, p, g) -> BEGIN
LOCAL n, d, l, f1, j, ds;
f1:=floor(f);
l:=f1;
n:=numer(f-f1);
d:=denom(f-f1);
FOR j FROM 1 TO p DO
ds,n:=iquorem(10^g*n,d);
l:=l*10^g+ds;
END;
RETURN l;
END;
```

We enter in the CAS:

```
decimalg(355/311, 8, 5)
```

We get:

```
11414790996784565916398713826366559485530
```

We enter in the CAS:

```
decimalg(355/113, 8, 5)
```

We get:

```
31415929203539823008849557522123893805309
```

29.5 Newton method and Heron algorithm

29.5.1 Newton method

Be f two times differentiable, having one and only one zero r in the interval $[a; b]$.

Let us additionally assume that f' and f'' has a constant sign on $[a; b]$. The Newton method consist in fit r by the abscissa x_1 of the point common to O_x and the tangent at the point M_0 to the graph of f . If M_0 has for coordinates $(x_0, f(x_0))$ ($x_0 \in [a; b]$), the tangent in M_0 has for equation:

$y = f(x_0) + f'(x_0) * (x - x_0)$ and then we have:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

We can then reiterate the process, and we get a sequence x_n converging to r

- either by greater values, if $f' * f'' > 0$ on $[a; b]$
(i.e. if $f'(r) > 0$ and if f is convex ($f'' > 0$ on $[a; b]$))
or
if $f'(r) < 0$ and if f is concave ($f'' < 0$ on $[a; b]$))
- either by lower values, if $f' * f'' < 0$ on $[a; b]$
(i.e. if $f'(r) < 0$ and if f is convex ($f'' > 0$ on $[a; b]$))
or
if $f'(r) > 0$ and if f is concave ($f'' < 0$ on $[a; b]$)).

The Heron algorithm is a specific case of the application of the Newton method to look for the approximate values of \sqrt{a} for a integer.

In this case

\sqrt{a} is a zero of $f(x) = x^2 - a$ and $g(x) = f'(x) = 2x$ then the sequence of the iterations is supplied by:

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2} \left(x_n + \frac{a}{x_n} \right)$$

29.5.2 Newton algorithm

The function `newton_rac` returns the approximate value, at the nearest p , of the root of $f(x) = 0$, starting the iteration with x_0 .

We notice that the parameter f is a function, and thus, that its derivative is the function `g:=function_diff(f)`.

We look for an approximate value, so we must write:

`x0:=evalf(x0)` because if we do not put `evalf`, the calculations of the iteration will be done in the exact way, and hence, will be soon complicated.

We check at the beginning if the sequence of $x_i (i = 0..n)$ is increasing or decreasing, starting from $n = 1$, by comparing at the beginning x_1 and x_2 . We name the program `newton_rac` and we check CAS.

The program `newton_rac` gives a zero of f close to x_0 at the nearest p .

We enter the program:

```
(f, x0, p) -> BEGIN
LOCAL x1, h, g;
g:=function_diff(f)
x0:=evalf(x0);
x0:=x0-f(x0)/g(x0);
x1:=x0-f(x0)/g(x0);
IF (x1>x0) THEN
  h:=p;
ELSE
  h:=-p;
END;
WHILE (f(x1)*f(x1+h)>0) DO
  x1:=x1-f(x1)/g(x1);
END;
RETURN x1;
END;
```

We enter in the CAS:

```
f(x):=cos(x)-x
newtonrac(f, 0.4, 1e-10)
```

We get:

```
0.739085133215
```

We enter in the CAS:

```
cos(0.739085133215)-0.739085133215
```

We get:

```
2.70006239589e-13
```

29.5.3 Heron algorithm

We name the program `heron` and we check CAS.

The program `heron` gives a fraction approaching \sqrt{a} at the nearest p when x_0 is close to \sqrt{a} .

We enter:

```
(a, x0, p) -> BEGIN
LOCAL b;
b:=x0-p;
WHILE b^2>a DO
  x0:=(x0+a/x0)/2;
  b:=x0-p;
END;
RETURN x0;
END;
```

We enter in the CAS:

```
heron(2, 3/2, 10^-10)
```

We get:

66587/470832

We enter in the CAS:

```
decimalg(66587/470832,2,5)
```

We get:

14142135623

We enter in the CAS:

```
f:=heron(2,2,10^-40)
```

We get:

1572584048032918633353217/1111984844349868137938112

We enter in the CAS:

```
r2:=decimalg(f,8,5)
```

We get:

14142135623730950488016887242096980785696

and $\sqrt{2} \approx r2 * 10^{-40}$

The library of long floating point numbers is not implemented in the HP Prime.

We use the CAS to check.

We enter in CAS:

```
evalf(r2*10^-40,41)
```

We get:

1.4142135623730950488016887242096980785696

We enter in CAS:

```
evalf(sqrt(2),41)
```

Because the CAS rounds off the last decimal place, we get:

14142135623730950488016887242096980785697

Chapter 30 Example of programs

30.1 GCD and Bezout identity from Home

30.1.1 GCD

We use Euclid's algorithm.

We enter:

```
EXPORT GCD(A,B)
BEGIN
LOCAL R;
WHILE B<>0 DO
  R:=A MOD B;
  A:=B;
  B:=R;
END;
RETURN A;
END;
```

Or else we use the function `irem` which returns the remainder of the Euclidean division:

```
EXPORT GCD(A,B)
BEGIN
LOCAL R;
WHILE B<>0 DO
  R:=CAS.irem(A,B);
  A:=B;
  B:=R;
END;
RETURN A;
END;
```

We enter:

GCD(45,25)

We get:

5

30.1.2 Bezout identity for A and B

We use Euclid's algorithm and the variables U, V, R which will vary so that at the step k we have $A * U_k + B * V_k = R_k$.

Thus, when R_p is the GCD of A and B , we will have:

$$A * U_p + B * V_p = \text{GCD}(A, B).$$

At the beginning, we have:

$$(1) A = U_1 * A + V_1 * B \quad (R_1 = A, U_1 = 1, V_1 = 0)$$

$$(2) B = U_2 * A + V_2 * B \quad (R_2 = B, U_2 = 0, V_2 = 1)$$

We want to get:

$$R_3 = U_3 * A + V_3 * B$$

since $R_3 = A - B * Q_3$ (with Q_3 integer quotient of $A = R_1$ by $B = R_2$) we find, by doing (1) - $Q_3 * (2)$:

$$U_3 = U_1 - Q_3 * U_2 \text{ and } V_3 = V_1 - Q_3 * V_2 \text{ and thus } R_3 = U_3 * A + V_3 * B \text{ and at each step we will have } R_k = U_k * A + V_k * B$$

with the relations:

$$U_k = U_{k-2} - Q_k * U_{k-1} \text{ and } V_k = V_{k-2} - Q_k * V_{k-1}.$$

To write the program, we need 3 lists $L1, L2, L3$ which will be 3 successive steps of $[U_k, V_k, R_k]$.

At the beginning:

$$L1 = 1, 0, R1 \text{ (} R1 = A \text{)}$$

$$L2 := 0, 1, R2 \text{ (} R2 = B \text{)}$$

We calculate $L3$:

$L3$ is obtained from $L1$ and $L2$ and if $Q3, R3 := \text{iquorem}(R1, R2)$, on $R3 = R1 - R2 * Q3$ and then $L3 = L1 - Q3 * L2$.

Then, $R1$ takes the value of $R2$, $L1$ the value of $L2$, $R2$ the value of $R3$, $L2$ the value of $L3$, etc., ...

We stop when $R2 = 0$ and then $R1 = \text{GCD}(A, B)$.

We enter:

```
EXPORT BEZOUT(A, B)
BEGIN
LOCAL L1, L2, L3, Q3, R1, R2, R3;
R1:=A;
R2:=B;
L1:={1, 0, R1};
L2:={0, 1, R2};
WHILE B<>0 DO
  Q3, R3:=CAS.iquorem(R1, R2);
  R1:=R2;
  R2:=R3;
  L3:=L1-Q3*L2;
  L1:=L2;
  L2:=L3;
END;
RETURN L1;
END;
```

We can reduce the number of variables:

```
EXPORT BEZOUT(A, B)
BEGIN
LOCAL L1, L2, L3, Q;
L1:={1, 0, A};
L2:={0, 1, B};
WHILE L2(3)<>0 DO
  //Q:=iquo(L1(3), L2(3));
  Q:=(L1(3)-L1(3) MOD L2(3))/L2(3)
  L3:=L1-Q*L2;
  L1:=L2;
  L2:=L3;
END;
RETURN L1;
END;
```

We enter:

BEZOUT(45, 10)

We get:

1, -4, 5

Which means that $1 * 45 - 4 * 10 = 5 = \text{GCD}(45, 25)$

We enter:

BEZOUT(45, 25)

We get:

-1, 2, 5

Which means that $-1 * 45 + 2 * 25 = 5 = \text{GCD}(45, 25)$

30.2 GCD and Bezout identity from the CAS

30.2.1 GCD with the CAS with no program

We can apply Euclid's algorithm by using the key `Enter` of the calculator. We enter on the entry line:

`a:=72;b:=33;` then we press `Enter`

Then, we enter on the entry line:

`r:=irem(a,b);a:=b;b:=r;` then we press `Enter` several times until the last value is null.

When the last value is null, the GCD of a and b is the value 3 above the 0.

We can check this thanks to the existing `gcd` command:

`gcd(72,33)` returns 3.

30.2.2 GCD with a CAS program

We use the function `irem` to write the Euclid's algorithm.

We check CAS, and we name the program `GCD`, and we enter:

```
(a,b)->
BEGIN
LOCAL r;
WHILE b<>0 DO
  r:=irem(a,b);
  a:=b;
  b:=r;
END;
RETURN(a);
END;
```

We enter:

`GCD(45,25)`

We get:

5

30.2.3 Bezout identity with the CAS, with no program

We can apply the algorithm giving the coefficients of the Bezout identity by using the `Enter` key of the calculator. We enter on the entry line:

`a:=72;b:=33;l1:=[1,0,a];l2:=[0,1,b]` then we press `Enter`

Then, we enter on the entry line:

`q:=iquo(l1(3),l2(3));l3:=l1-q*l2;l1:=l2;l2:=l3;` then we press `Enter` several times until the last value of the last list is null.

When this last value is null, the Bezout identity $[-5, 11, 3]$ is the last list above the 0: this means that $-5 \cdot 72 + 11 \cdot 33 = 3$.

We can check this thanks to the existing `iegcd` command, giving the Bezout identity:

`iegcd(72,33)` returns $[-5, 11, 3]$.

30.2.4 Bezout identity with a CAS program

We use Euclid's algorithm and the variables u, v, r which will vary so that at the step k we have $a \cdot u_k + b \cdot v_k = r_k$.

Thus, when rp is the GCD of a and b , we will have:

$a \cdot u_p + b \cdot v_p = \text{GCD}(a, b)$.

To write the program, we need 3 lists $L1, L2, L3$ which will be 3 successive steps of $[u_k, v_k, r_k]$.

At the beginning:

$$\begin{aligned} L1 &= 1, 0, r1 \quad (r1 = a) \\ L2 &= 0, 1, r2 \quad (r2 = b) \end{aligned}$$

We calculate $L3$:

$L3$ is obtained from $L1$ and $L2$ and if $q3, r3 := \text{iquorem}(r1, r2)$, we have: $r3 = r1 - r2 \cdot q3$ and then $L3 = L1 - q3 \cdot L2$.

Then, $r1$ takes the value of $r2$, $L1$ the value of $L2$, $r2$ takes the value of $r3$, $L2$ the value of $L3$, etc., ...

We stop when $e_2 = 0$ and then $r_1 = \text{GCD}(a, b)$.

We enter:

```
(a,b)->BEGIN
LOCAL 11,12,13,q;
11:=[1,0,a];
12:=[0,1,b];
WHILE 12(3)<>0 DO
  q:=iquo(11(3),12(3));
  13:=11-q*12;
  11:=12;
  12:=13;
END;
RETURN 11;
END;
```

We enter:

BEZOUT(45,10)

We get:

1, -4, 5

Which means that $1*45-4*10=5=\text{GCD}(45, 25)$

We enter:

BEZOUT(45,25)

We get:

-1, 2, 5

Which means that $-1*45+2*25=5=\text{GCD}(45,25)$