Calcul formel
et
Mathématiques
avec
la calculatrice `HP Prime`

Renée De Graeve
Maître de Conférence à Grenoble I

© 2013 Renée De Graeve, renee.degraeve@wanadoo.fr
La copie, la traduction et la redistribution de ce document sur support électronique ou papier sont autorisés pour un usage non commercial uniquement. L'utilisation de ce document à des fins commerciales est interdite sans l'accord écrit du détenteur du copyright. Cette documentation est fournie en l'état, sans garantie d'aucune sorte. En aucun cas le détenteur du copyright ne pourra être tenu pour responsable de dommages résultant de l'utilisation de ce document.

Ce document est disponible à l'adresse Internet suivante :
http://www-fourier.ujf-grenoble.fr/~parisse/hprime.pdf

Table des matières

I Pour commencer **21**

0.1 Les touches CAS et 23
0.2 Pour reinitialiser et pour effacer 25
0.3 L'écran tactile 25
0.4 Le touches 26
0.5 La configuration générale 26
0.6 La configuration du CAS 26
0.7 La configuration de la calculatrice Shift- 27
0.8 Les fonctions de calcul formel 27

II Le menu CAS de la touche **29**

1 Généralités **31**

1.1 Les calculs dans le CAS 31
1.2 La priorité des opérateurs 31
1.3 La multiplication implicite 31
1.4 Les listes et les séquences dans le CAS 32
1.5 Différence entre expressions et fonctions 33
1.5.1 Exercice sur les expressions 33
1.5.2 Vérifions avec HPprime 34
1.5.3 Exercice (suite) sur les fonctions 34
1.5.4 Vérifions avec HPprime 34

2 Le menu Algebra **37**

2.1 Simplifier une expression : simplify 37
2.2 Factoriser un polynôme sur les entiers : collect 37
2.3 Développer une expression : expand 38
2.4 Factoriser une expression : factor 38
2.5 Substituer une variable par une valeur : subst 39
2.6 Décomposer en éléments simples : partfrac 40
2.7 Extract 40
2.7.1 Numérateur d'une fraction après simplification : numer . 40
2.7.2 Dénominateur d'une fraction après simplification : denom 41
2.7.3 Pour avoir le membre de gauche d'une équation : left . 41
2.7.4 Pour avoir le membre de droite d'une équation : right . 41

3 Le menu Calculus **43**
3.1 Définition d'une fonction : := et -> (Sto▶) 43
3.2 diff ou ' 43
3.3 int 44
3.4 limit 45
3.5 series 46
3.6 sum 47
3.7 Differential 48
3.7.1 Le rotationnel : curl 48
3.7.2 La divergence : divergence 48
3.7.3 Le gradient : grad 48
3.7.4 La hessienne : hessian 49
3.8 Integral 50
3.8.1 Intégration par parties : ibpdv 50
3.8.2 Intégration par parties : ibpu 51
3.8.3 Évaluer une primitive : preval 52
3.9 Limits 53
3.9.1 Somme de Riemann : sum_riemann 53
3.9.2 Développement limité : taylor 53
3.9.3 Division selon les puissances croissantes : divpc 54
3.10 Transform 54
3.10.1 Transformée de Laplace : laplace 54
3.10.2 Transformée de Laplace inverse : invlaplace 55
3.10.3 invlaplace 55
3.10.4 La transformée de Fourier rapide : fft 57
3.10.5 L'inverse de la transformée de Fourier rapide : ifft . . . 57

4 Le menu Solve **59**
4.1 Résolution d'équations : solve 59
4.2 Zéros d'une expression : zeros 60
4.3 Résoudre des équations dans $\mathbb{C}$: cSolve 61
4.4 Zéros complexe d'une expression : cZeros 61
4.4.1 Équations différentielles : deSolve 62
4.5 Résolution numérique d'équationsnSolve 63
4.6 Solution approchée de y'=f(t,y) :odesolve 63
4.6.1 Résoudre un système linéaire : linsolve 65

5 Le menu Rewrite **67**
5.1 Regrouper les log : lncollect 67
5.2 Transformer une puissance en produit de puissances : powexpand 67
5.3 Développer une expression transcendante et de trigo : tExpand . 68
5.4 Exp & Ln 69
5.4.1 Transformer exp(n*ln(x)) en puissance : exp2pow 69
5.4.2 Transformer une puissance en une exponentielle : pow2exp 70
5.4.3 Transformer les exponentielles complexes en sin et en cos : exp2trig 70
5.4.4 Développer les exponentielles : expexpand 71

5.5 Sine 71
5.5.1 Transformer les arcsin en arccos : asin2acos 71
5.5.2 Transformer les arcsin en arctan : asin2atan 71
5.5.3 Transformer sin(x) en cos(x)*tan(x) : sin2costan . . . 72
5.6 Cosine 72
5.6.1 Transformer les arccos en arcsin : acos2asin 72
5.6.2 Transformer les arccos en arctan : acos2atan 72
5.6.3 Transformer cos(x) en sin(x)/tan(x) : cos2sintan . . . 72
5.7 Tangent 73
5.7.1 Transformer les arctan en arcsin : atan2asin 73
5.7.2 Transformer les arctan en arccos : atan2acos 73
5.7.3 Transformer tan(x) en sin(x)/cos(x) : tan2sincos . . . 73
5.7.4 Transformer une expression trigonométrique en fonction de tan(x/2) : halftan 73
5.8 Trig 74
5.8.1 Simplifier en privilégiant les sinus : trigsin 74
5.8.2 Simplifier en privilégiant les cosinus : trigcos 74
5.8.3 Transformer avec des fonctions trigonométriques inverses en logarithmes : atrig2ln 74
5.8.4 Simplifier en privilégiant les tangentes : trigtan 75
5.8.5 Linéariser une expression trigonométrique : tlin 75
5.8.6 Rassembler les sinus et cosinus de même angle : tCollect 75
5.8.7 Développer une expression trigonométriques : trigexpand 76

6 Le menu Integer 77
6.1 Les diviseurs d'un nombre entier : idivis 77
6.2 Décomposition en facteurs premiers d'un entier : ifactor . . . 77
6.3 Liste des facteurs premiers et de leur multiplicité : ifactors . . 77
6.4 PGCD de deux ou plusieurs entiers : gcd 78
6.5 PPCM de deux ou plusieurs entiers : lcm 78
6.6 Prime 78
6.6.1 Test pour savoir si un nombre est premier : isPrime . . 78
6.6.2 Le N-ième nombre premier : ithprime 79
6.6.3 nextprime 79
6.6.4 prevprime 79
6.6.5 Indicatrice d'Euler : euler 80
6.7 Division 80
6.7.1 Quotient de la division euclidienne : iquo 80
6.7.2 Reste de la division euclidienne : irem 80
6.7.3 Restes chinois pour des entiers : ichinrem 81
6.7.4 Calcul de $a^n \mod p$: powmod 81

7 Le menu Polynomial 83
7.1 Racines numériques d'un polynôme : proot 83
7.2 Coefficients d'un polynôme : coeff 84
7.3 Liste des diviseurs d'un polynôme : divis 84
7.4 Liste des facteurs d'un polynôme : factors 85
7.5 PGCD de polynômes par l'algorithme d'Euclide : gcd 85

7.6 PPCM de deux polynômes : lcm 87
7.7 Create 88
7.7.1 Transformer un polynôme en une liste (format interne récursif dense) : symb2poly 88
7.7.2 Transformer le format interne creux distribué du polynôme en une écriture polynômiale : poly2symb 89
7.7.3 Coefficients d'un polynôme défini par ses racines : pcoeff pcoef 89
7.7.4 Coefficients d'une fraction rationnelle définie par ses racines et ses pôles : fcoeff 90
7.7.5 Coefficient du terme de plus haut degré d'un polynôme : lcoeff 90
7.7.6 Polynômes aléatoires : randPoly 91
7.8 Algebra 92
7.8.1 Quotient euclidien de 2 polynômes : quo 92
7.8.2 Reste euclidien de 2 polynômes : rem 93
7.8.3 Degré d'un polynôme : degree 94
7.8.4 Mise en facteur de x^n dans un polynôme : factor_xn . 94
7.8.5 PGCD des coefficients d'un polynôme : content 95
7.8.6 Nombre de changements de signe sur $]a;\ b]$: sturmab . 95
7.8.7 Les restes chinois : chinrem 96
7.9 Special 97
7.9.1 Polynôme cyclotomique : cyclotomic 97
7.9.2 Base de Gröbner : gbasis 98
7.9.3 Réduction par rapport à une base de Gröbner : greduce . 98
7.9.4 Polynôme de Hermite : hermite 99
7.9.5 Interpolation de Lagrange : lagrange 99
7.9.6 Polynôme de Laguerre : laguerre 100
7.9.7 Polynôme de Legendre : legendre 101
7.9.8 Polynôme de Tchebychev de 1-ière espèce : tchebyshev1 101
7.9.9 Polynôme de Tchebychev de 2-nde espèce : tchebyshev2 102

8 Le menu Plot 105
8.1 Graphe d'une fonction : plotfunc 105
8.2 Courbe implicite en 2-d : plotimplicit 106
8.3 Graphe d'une fonction par niveaux de couleurs : plotdensity . 107
8.4 Le champ des tangentes : plotfield 107
8.5 Lignes de niveaux : plotcontour 108
8.6 Tracé de solutions d'équation différentielle : plotode 109
8.7 Ligne polygonale : plotlist 110

III Le menu MATH de la touche 111

9 Les fonctions sur les réels 113
9.1 Le plus petit entier >= à l'argument : CEILING 113
9.2 Partie entière d'un réel : FLOOR 113
9.3 Argument sans sa partie fractionnaire : IP 114

9.4 Partie fractionnaire : FP 114
9.5 Arrondir avec n décimales un réel ou un complexe : ROUND . . . 115
9.6 Tronquer avec n décimales un réel ou un complexe : TRUNCATE . 116
9.7 Mantisse d'un réel : MANT 116
9.8 Partie entière du logarithme à base 10 d'un réel : XPON 117

10 Arithmétique **119**
10.1 Maximum de 2 ou plusieurs valeurs :MAX 119
10.2 Minimum de 2 ou plusieurs valeurs : MIN 119
10.3 MOD 119
10.4 FNROOT 120
10.5 % 120
10.6 Complexe 120
10.6.1 CONJ 120
10.6.2 IM 121
10.6.3 RE 121
10.6.4 SIGN 121
10.6.5 La touche Shift-+/− : ABS 121
10.7 Exponentielle 121
10.7.1 ALOG alog10 121
10.7.2 EXPM1 122
10.7.3 LNP1 122

11 Fonctions trigonométriques **123**
11.1 CSC 123
11.2 ACSC 123
11.3 SEC 123
11.4 ASEC 124
11.5 COT 124
11.6 ACOT 124

12 Fonctions hyperboliques **125**
12.1 SINH 125
12.2 ASINH 125
12.3 COSH 125
12.4 ACOSH 126
12.5 TANH 126
12.6 ATANH 126

13 Fonctions de probabilité **127**
13.1 Factorielle : ! 127
13.2 Nombre de combinaisons de p objets pris parmi n : COMB 127
13.3 nombre d'arrangements de p objets pris parmi n : PERM 128
13.4 Nombres aléatoires 128
13.4.1 Nombre aléatoire (réel ou entier) : RANDOM 128
13.4.2 Nombre entier aléatoire RANDINT 128
13.4.3 RANDNORM 129
13.4.4 Pour initialiser la suite de nombres aléatoires : RANDSEED 129
13.5 Densité de probabilité 130

13.5.1 Densité de probabilité de la loi normale : NORMALD . . . 130
13.5.2 Densité de probabilité de la loi de Student : STUDENT . . 130
13.5.3 Densité de probabilité du χ^2 : CHISQUARE 130
13.5.4 Densité de probabilité de la loi de Fisher : FISHER 130
13.5.5 Densité de probabilité de la loi binomiale : BINOMIAL . . 131
13.5.6 Densité de probabilité de la loi de Poisson : POISSON . . 131
13.6 Fonction de répartition 131
13.6.1 Fonction de répartition de la loi normale : NORMALD_CDF 131
13.6.2 Fonction de répartition de la loi de Student : STUDENT_CDF 132
13.6.3 Fonction de répartition de la loi du χ^2 CHISQUARE_CDF 133
13.6.4 La fonction de répartition de la loi de Fisher-Snédécor : FISHER_CDF 133
13.6.5 Fonction de répartition de la loi binomiale : BINOMIAL_CDF 134
13.6.6 Fonction de répartition de Poisson POISSON_CDF 135
13.7 Fonction de répartition inverse 135
13.7.1 Fonction de répartition inverse normale : NORMALD_ICDF 135
13.7.2 Fonction de répartition inverse de Student : STUDENT_ICDF 136
13.7.3 Fonction inverse de la fonction de répartition de la loi du χ^2CHISQUARE_ICDF 136
13.7.4 Inverse de la fonction de répartition de la loi de Fisher-Snédécor : FISHER_ICDF 136
13.7.5 Fonction de répartition inverse de la loi binomiale : BINOMIAL_ICDF 137
13.7.6 Fonction de répartition inverse de Poisson :POISSON_ICDF 137

14 Les listes 139
14.1 MAKELIST 139
14.2 SORT 139
14.3 REVERSE 140
14.4 CONCAT 140
14.5 POS 142
14.6 SIZE 142
14.7 ΔLIST 143
14.8 ΣLIST 143
14.9 ΠLIST 143

15 Les chaînes de caractères 145
15.1 asc 145
15.2 char 145
15.3 Pour utiliser une chaîne comme un nombre ou une commande : expr 146
15.3.1 Pour utiliser une chaîne comme un nombre 146
15.3.2 Pour utiliser une chaîne comme nom de commande 147
15.4 Évaluer une expression sous la forme d'une chaîne : string . . . 147
15.5 inString 148
15.6 left 148
15.7 right 148
15.8 mid 148
15.9 rotate 149

15.10 dim 149
15.11 + ... 149
15.12 Avoir la liste ou la chaîne privée de son premier élément : tail . 150
15.13 Début d'une liste ou d'une chaîne : head 150

16 Les matrices **151**
16.1 Matrice transposée : tran 151
16.2 Matrice transposée : TRN ou trn 151
16.3 Déterminant : DET ou det 151
16.4 Résolution d'un système linéaire : RREF ou rref 152
16.5 Création de matrices 153
16.5.1 Créer une matrice à partir d'une expression : MAKEMAT et makemat 153
16.5.2 Matrice identité : IDENMAT ou identity 154
16.5.3 Matrice aléatoire : RANDMAT et ramn 154
16.5.4 JordanBlock 155
16.5.5 N-ième matrice de Hilbert hilbert 155
16.5.6 Matrice d'une isométrie : mkisom 155
16.5.7 Matrice de Vandermonde : vandermonde 156
16.6 Basique 157
16.6.1 Norme de Schur ou de Frobenius d'une matrice : ABS . . 157
16.6.2 Maximum des normes des lignes d'une matrice : ROWNORM ou rownorm 158
16.6.3 Max des normes des colonnes d'une matrice : COLNORM ou colnorm 158
16.6.4 Norme spectrale d'une matrice : SPECNORM 159
16.6.5 Rayon spectral d'une matrice carrée : SPECRAD 160
16.6.6 Conditionnement d'une matrice carrée inversible : COND . 160
16.6.7 Rang d'une matrice : RANK ou rank 160
16.6.8 Étape de la réduction de Gauss-Jordan d'une matrice : pivot 161
16.6.9 Trace d'une matrice carrée : TRACE ou trace 161
16.7 Avancée 162
16.7.1 Valeurs propres : EIGENVAL et eigenvals 162
16.7.2 Vecteurs propres : EIGENVV et eigenvects 163
16.7.3 Matrice de Jordan jordan 164
16.7.4 Matrice diagonale et sa diagonale : diag 164
16.7.5 Matrice de Cholesky : cholesky 165
16.7.6 Forme normale de Hermite d'une matrice : ihermite . . 165
16.7.7 Réduction de Hessenberg d'une matrice : hessenberg . 165
16.7.8 ismith 166
16.8 Factorisation 167
16.8.1 Décomposition LQ d'une matrice : LQ 167
16.8.2 Norme minimale du système linéaire A*X=B : LSQ 168
16.8.3 Décomposition LU d'une matrice carrée : LU 169
16.8.4 Décomposition QR d'une matrice carrée : QR 170
16.8.5 Réduction de Hessenberg d'une matrice : SCHUR 170
16.8.6 Singular value decomposition : SVD et svd 171
16.8.7 Valeurs singulières : SVL 172

16.9 Vecteur 173
16.9.1 Produit vectoriel : CROSS ou cross 173
16.9.2 Produit scalaire : DOT ou dot 174
16.9.3 Norme l^2 :l2norm 174
16.9.4 Norme l^1 :l1norm 175
16.9.5 Norme du maximum :maxnorm 175

17 Les fonctions spéciales 177
17.1 La fonction β : Beta 177
17.2 La fonction Γ : Gamma 178
17.3 Les derivées de la fonction DiGamma : Psi 179
17.4 La fonction ζ : Zeta 180
17.5 La fonction erf : erf 180
17.6 La fonction $erfc$: erfc 181
17.7 La fonction exponentielle integrale Ei : Ei 182
17.8 La fonction sinus integral Si : Si 183
17.9 La fonction cosinus integral Ci : Ci 184

IV Les Applications et la touche Apps 187

18 Le menu Geometry 189
18.1 Généralités 189
18.2 Point 191
18.2.1 Point défini comme barycentre de n points : barycenter 191
18.2.2 Le point en géométrie : point 193
18.2.3 Orthocentre d'un triangle : orthocenter ou orthocenter 194
18.2.4 Le milieu d'un segment : midpoint 194
18.2.5 L'isobarycentre de n points : isobarycenter 195
18.2.6 Définir au hasard un point 2-d : point2d 195
18.2.7 Point sur un objet géométrique : element 195
18.2.8 Point divisant un segment : division_point 198
18.2.9 Le centre d'un cercle : center 199
18.3 Line 199
18.3.1 Droite définie par un point et une pente : DrawSlp 199
18.3.2 Tangente au graphe de $y = f(x)$ en $x = a$: LineTan . . 199
18.3.3 Hauteur d'un triangle : altitude 200
18.3.4 Bissectrice intérieure d'un angle : bisector 200
18.3.5 Bissectrice extérieur d'un angle : exbisector 201
18.3.6 Demi-droite : half_line 201
18.3.7 La droite et la droite orientée : line 201
18.3.8 Médiane d'un triangle : median_line 203
18.3.9 Droites parallèles : parallel 203
18.3.10 Médiatrice : perpen_bisector 204
18.3.11 Perpendiculaire à une droite : perpendicular 204
18.3.12 Segment : segment 204
18.3.13 Les tangentes à un objet géométrique ou la tangente en un point d'un graphe : tangent 205

18.4 Polygon 206
18.4.1 Triangle équilatèral : `equilateral_triangle` 206
18.4.2 Triangle rectangle : `right_triangle` 206
18.4.3 Losange : `rhombus` 207
18.4.4 `rectangle` 208
18.4.5 `square` 209
18.4.6 `quadrilateral` 209
18.4.7 `parallelogram` 209
18.4.8 `isoployon` 210
18.4.9 `isosceles_triangle` 210
18.5 Courbes 211
18.5.1 Le cercle et ses arcs : `circle` 211
18.5.2 Le cercle circonscrit : `circumcircle` 213
18.5.3 Tracé d'une conique : `conic` 213
18.5.4 L'ellipse : `ellipse` 213
18.5.5 Le cercle exinscrit : `excircle` 214
18.5.6 L'hyperbole : `hyperbola` 214
18.5.7 Le cercle inscrit : `incircle` 215
18.5.8 Lieu et enveloppe : `locus` 215
18.5.9 La parabole : `parabola` 217
18.6 Transformation 218
18.6.1 L'homothétie : `homothety` 218
18.6.2 L'inversion : `inversion` 219
18.6.3 La projection orthogonale : `projection` 220
18.6.4 La symétrie droite et la symétrie point : `reflection` . . 221
18.6.5 La rotation : `rotation` 222
18.6.6 La similitude : `similarity` 222
18.6.7 La translation : `translation` 223
18.7 Mesure et graphique 224
18.7.1 Affichage de la mesure d'un angle : `angleat` 224
18.7.2 Affichage de la mesure d'un angle : `angleatraw` 225
18.7.3 Affichage de l'aire d'un polygone : `areaat` 225
18.7.4 Affichage de l'aire d'un polygone : `areaatraw` 226
18.7.5 Affichage de la longueur d'un segment : `distanceat` . 227
18.7.6 Affichage de la longueur d'un segment : `distanceatraw` 227
18.7.7 Affichage du périmètre d'un polygone :`perimeterat` . 228
18.7.8 Affichage du périmètre d'un polygone : `perimeteratraw`229
18.7.9 Affichage de la pente d'une droite : `slopeat` 230
18.7.10 Affichage de la pente d'une droite : `slopeatraw` 231
18.8 Measure 232
18.8.1 L'abscisse d'un point ou d'un vecteur : `abscissa` 232
18.8.2 L'affixe d'un point ou d'un vecteur : `affix` 232
18.8.3 La mesure d'un angle : `angle` 233
18.8.4 Longueur d'un arc de courbe : `arcLen` 234
18.8.5 Aire d'un polygone : `area` 235
18.8.6 Les coordonnées d'un point, d'un vecteur ou d'une droite : `coordinates` 235

18.8.7 La longueur d'un segment et distance entre les deux objets géométriques : distance 237
18.8.8 Le carré de la longueur d'un segment : distance2 . . . 238
18.8.9 L'équation cartésienne d'un objet géométrique : equation 238
18.8.10 Avoir comme réponse la valeur d'une mesure affichée : extract_measure 239
18.8.11 L'ordonnée d'un point ou d'un vecteur : ordinate . . . 239
18.8.12 L'équation paramétrique d'un objet géométrique : parameq 240
18.8.13 Périmètre d'un polygone : perimeter 241
18.8.14 Le rayon d'un cercle : radius 241
18.8.15 Pente d'une droite : slope 241
18.9 Test 242
18.9.1 Savoir si 3 points sont alignés : is_colinear 242
18.9.2 Savoir si 4 points sont cocycliques : is_concyclic . . 243
18.9.3 Savoir si des éléments sont conjugués : is_conjugate 243
18.9.4 Savoir si des points ou /et des droites sont coplanaires : is_coplanar 244
18.9.5 Savoir si 1 point est sur un objet graphique : is_element 245
18.9.6 Savoir si on a un triangle équilatéral : is_equilateral 245
18.9.7 Savoir si on a un triangle isocéle : is_isosceles . . . 246
18.9.8 Orthogonalité de 2 droites ou 2 sphères : is_orthogonal 246
18.9.9 Savoir si 2 droites sont parallèles : is_parallel 247
18.9.10 Savoir si on a un parallélogramme : is_parallelogram 248
18.9.11 Savoir si 2 droites sont perpendiculaire : is_perpendicular 248
18.9.12 Savoir si on a un triangle rectangle ou si on a un rectangle : is_rectangle 249
18.9.13 Savoir si on a un losange : is_rhombus 250
18.9.14 Savoir si on a un carré : is_square 250
18.10 Exercices de géométrie 251
18.10.1 Les transformations 251
18.10.2 Les lieux 252
18.11 Activités de géométrie 252

19 Le tableur 259
19.1 Généralités 259
19.2 Description de l'écran du tableur 259
19.2.1 Pour copier la formule d'une cellule 259
19.2.2 Références absolues et relatives 259
19.3 Les fonctions du tableur 260
19.3.1 SUM 260
19.3.2 AVERAGE 260
19.3.3 AMORT 260
19.3.4 STAT1 261
19.3.5 REGRS 261
19.3.6 PredY PredX 261
19.3.7 HypZ1mean HypZ2mean 261
19.4 Utilisation du tableur sur des exemples 261

19.4.1 Exercice 1 261
19.4.2 Exercice 2 262

20 Les autres Applications 267
20.1 L'Application Fonction 267
20.2 L'Application Suite 267
20.2.1 La suite de Fibonnacci 267
20.2.2 Le PGCD 268
20.2.3 L'identité de Bézout 268
20.3 L'Aplet Paramétrique 269
20.4 L'Aplet Polaire 270
20.5 L'Aplet Résoudre 270
20.6 L'Aplet Finance 270
20.7 L'Aplet Système Linéaire 271
20.8 Aplet Triangle Solver 271
20.9 Statistiques 1-Var 272
20.10 Statistiques 2-Var 273
20.11 Aplet Inférence 275
20.11.1 Fréquence d'un caractère et jugements sur échantillons . . 276
20.11.2 Échantillons extraits d'une distribution normale 280
20.11.3 Échantillons extraits d'une distribution de Student 282
20.12 L'Aplet Linear Explorer 283
20.13 L'Aplet Quadratic Explorer 283
20.13.1 L'Aplet Trig Explorer 283
20.14 L'Aplet Données du Steamer 283

V La programmation avec le CAS 285

21 Généralités 287

22 Les instructions de programmation 289
22.1 L'instruction d'affectation 289
22.2 Les instructions conditionnelles 289
22.3 Les boucles 293
22.4 Les commentaires 294
22.5 Les variables 294
22.6 Les opérateurs booléens 294
22.7 Les commandes d'invite 296
22.8 Les commandes des Applications 299

23 Pour apprendre à programmer 301
23.1 L'instruction conditionnelle IF 301
23.2 Les boucles FOR et WHILE 302
23.2.1 Faire compter la calculatrice de 1 en 1 en utilisant un affichage 302
23.2.2 Faire compter la calculatrice de 1 en 1 en utilisant une liste ou une séquence 304
23.3 Valeur approchée de la somme d'une série 306

23.3.1 Série de terme général $u_n = 1/n^2$ 306
23.3.2 Série de terme général $v_n = (-1)^(n+1)/n$ 307
23.3.3 La série de terme général $w_n = 1/n$ est divergente 308
23.4 Écriture décimale d'une fraction 309
23.4.1 Sans programme 309
23.4.2 Avec un programme du CAS 310
23.5 La mèthode de Newton et l'algorithme de Héron 312
23.5.1 La méthode de Newton 312
23.5.2 Traduction de l'algorithme de Newton 312
23.5.3 Traduction de l'algorithme de Héron 313

24 Exemple de programmes 315
24.1 Le PGCD et l'identité de Bézout depuis Home 315
24.1.1 Le PGCD 315
24.1.2 L'identité de Bézout pour A et B 316
24.2 Le PGCD et l'identité de Bézout depuis CAS 317
24.2.1 Le PGCD avec le CAS sans faire de programme 317
24.2.2 Le PGCD avec un programme CAS 317
24.2.3 L'identité de Bézout avec le CAS sans faire de programme 318
24.2.4 L'identité de Bézout avec un programme CAS 318

Index

ΔLIST, 143
ΠLIST, 143
', 43
", 43
+, 149
->, 43
//, 294
:=, 43, 289
<,<=,==,
=,>,>=, 294
%, 120
!, 127

ΣLIST, 143
MOD, 119
STUDENT, 130

ABS, 121, 157
abscissa, 232
acos2asin, 72
acos2atan, 72
ACOSH, 126
ACOT, 124
ACSC, 123
affix, 232
ALOG, 121
alog10, 121
altitude, 200
AND, 295
angle, 233
angleat, 224
angleatraw, 225
arcLen, 234
area, 235
areaat, 225
areaatraw, 226
asc, 145
ASEC, 124
asin2acos, 71
asin2atan, 71
ASINH, 125
atan2acos, 73
atan2asin, 73
ATANH, 126
atrig2ln, 74

barycenter, 191
Beta, 177
BINOMIAL, 131
BINOMIAL_CDF, 134
BINOMIAL_ICDF, 137
bisector, 200
black, 190
blue, 190
BREAK, 293

CASE, 291
CEILING, 113
center, 199
char, 145
CHECK, 299
chinrem, 96
CHISQUARE, 130
CHISQUARE_CDF, 133
CHISQUARE_ICDF, 136
cholesky, 165
CHOOSE, 296
Ci, 184
circle, 211
circumcircle, 213
coeff, 84
collect, 37
COLNORM, 158
colnorm, 158
COMB, 127
CONCAT, 140, 149
COND, 160
conic, 213
CONJ, 120
content, **95**
CONTINUE, 292
coordinates, 235

cos2sintan, 72
COSH, 125
COT, 124
CROSS, 173
cross, 173
CSC, 123
cSolve, 61
curl, 48
cyan, 190
cyclotomic, 97
cZeros, 61

degree, 94
denom, 41
deSolve, 62
DET, 151
det, 151
diag, 164
diff, 43
dim, 149
DISP, 296
display, 190
distance, 237
distance2, 238
distanceat, 227
distanceatraw, 227
divergence, 48
divis, 84
division_point, 198
divpc, 54
DO, 293
DOT, 174
dot, 174
DrawSlp, 199

EDITMAT, 298
Ei, 182
EIGENVAL, 162
eigenvals, 162
eigenvects, 163
EIGENVV, 163
element, 195
ellipse, 213
ELSE, 291
END, 289, 291, 293
equation, 238
equilateral_triangle, 206
erf, 180
erfc, 181
euler, 80
exbisector, 201
excircle, 214
exp2pow, 69
exp2trig, 70
expand, 38
expexpand, 71
EXPM1, 122
expr, 146
extract_measure, 239

factor, 38
factor_xn, 94
factors, 85
fcoeff, 90
fft, 57
filled, 190
FISHER, 130
FISHER_CDF, 133
FISHER_ICDF, 136
FLOOR, 113
FNROOT, 120
FOR, 293
FP, 114
FREEZE, 296
FROM, 293

Gamma, 178
gbasis, 98
gcd, 78, 85
GETKEY, 296
grad, 48
greduce, 98
green, 190

half_line, 201
halftan, 73
head, 150
hermite, 99
hessenberg, 165
hessian, 49
hilbert, 155
homothety, 218
hyperbola, 214

ibpdv, 50
ibpu, 51
ichinrem, 81
IDENMAT, 154

identity, 154
idivis, 77
IF, 289, 291
ifactor, 77
ifactors, 77
IFERR, 292
ifft, 57
IFTE, 290
ihermite, 165
IM, 121
incircle, 215
INPUT, 297
inString, 148
int, 44
inversion, 219
invlaplace, 55
IP, 114
iquo, 80
irem, 80
is_colinear, 242
is_concyclic, 243
is_conjugate, 243
is_coplanar, 244
is_element, 245
is_equilateral, 245
is_isosceles, 246
is_orthogonal, 246
is_parallel, 247
is_parallelogram, 248
is_perpendicular, 248
is_rectangle, 249
is_rhombus, 250
is_square, 250
ISKEYDOWN, 297
ismith, 166
isobarycenter, 195
isoploygon, 210
isosceles_triangle, 210
isPrime, 78
ithprime, 79

jordan, 164
JordanBlock, 155

l1norm, 175
l2norm, 174
lagrange, 99
laguerre, 100
laplace, 54
lcm, 78, 87
lcoeff, 90
left, 41, 148
legendre, 101
line, 201
LineTan, 199
linsolve, 65
lncollect, 67
LNP1, 122
LOCAL, 294
locus, 215
LQ, 167
LSQ, 168
LU, 169

magenta, 190
MAKELIST, 139
MAKEMAT, 153
makemat, 153
MANT, 116
MAX, 119
maxnorm, 175
median_line, 203
mid, 148
midpoint, 194
MIN, 119
mkisom, 155
MSGBOX, 297

nextprime, 79
NORMALD, 130
NORMALD_CDF, 131
NORMALD_ICDF, 135
NOT, 295
nSolve, 63
numer, 40

odesolve, 63
OR, 295
order_size, **53**
ordinate, 239
orthocenter, 194
orthocentre, 194

parabola, 217
parallel, 203
parallelogram, 209
parameq, 240
partfrac, 40

pcoef, 89
pcoeff, 89
perimeter, 241
perimeterat, 228
perimeteratraw, 229
PERM, 128
perpen_bisector, 204
perpendicular, 204
pivot, 161
plotcontour, 108
plotdensity, 107
plotfield, 107
plotfunc, 105
plotimplicit, **106**
plotlist, 110
plotode, 109
pmin, 90
point, 193
point2d, 195
POISSON, 131
POISSON_CDF, 135
POISSON_ICDF, 137
poly2symb, 89
POS, 142
pow2exp, 70
powexpand, 67
powmod, 81
preval, 44, 52
prevprime, 79
PRINT, 297
projection, 220
proot, 83
Psi, 179

QR, 170
quadrilateral, 209
quo, 92

radius, 241
ramn, 154
RANDINT, 128
RANDNMAT, 154
RANDNORM, 129
RANDOM, 128
randPoly, 91
RANDSEED, 129
RANK, 160
rank, 160
RE, 121
rectangle, 208
red, 190
reflection, 221
rem, 93
REPEAT, 293
REVERSE, 140
rhombus, 207
right, 41, 148
right_triangle, 206
rotate, 149
rotation, 222
ROUND, 115
ROWNORM, 158
rownorm, 158
RREF, 152
rref, 152

SCHUR, 170
SEC, 123
segment, 204
SELECT, 300
Si, 183
SIGN, 121
similarity, 222
simplify, 37
sin2costan, 72
SINH, 125
SIZE, 142, 149
slope, 241
slopeat, 230
slopeatraw, 231
solve, 59
SORT, 139
SPECNORM, 159
SPECRAD, 160
square, 209
STARTAPLET, 300
STARTVIEW, 300
STEP, 293
STO, 289
Sto►, 43
string, 147
STUDENT_CDF, 132
STUDENT_ICDF, 136
sturmab, 95
subst, 39
sum_riemann, 53
SVD, 171

svd, 171
SVL, 172
symb2poly, 88

tail, 150
tan2sincos, 73
tangent, 205
TANH, 126
taylor, 53
tchebyshev1, 101
tchebyshev2, 102
tCollect, 75
tExpand, 68
THEN, 289, 291
tlin, 75
TO, 293
TRACE, 161
trace, 161, 190, 196
tran, 151
translation, 223
trig2exp, 76
trigcos, 74
trigexpand, 76
trigsin, 74
trigtan, 75
TRN, 151
trn, 151
TRUNCATE, 116

UNCHECK, 300
UNTIL, 293

vandermonde, 156

WAIT, 298
WHILE, 293
white, 190

XOR, 295
XPON, 117

yellow, 190

zeros, 60
Zeta, 180

Première partie

Pour commencer

0.1 Les touches CAS et

Avec la calculatrice HPrime vous pouvez choisir de travailler en mode exact ou en mode approché : il y a deux écrans, l'un pour faire du calcul exact c'est l'écran CAS, l'autre pour faire du calcul approché c'est l'écran HOME.
Dans l'écran CAS ont peut aussi faire du calcul approché par exemple $1/2$ est un nombre exact et `evalf(1/2)=0.5` est un nombre approché. Si dans une expression il y a un nombre approché le résultat sera approché, par exepmle : $1/2 + 1/3$ renvoie $5/6$ alors que $0.5 + 1/3$ renvoie 0.833333333333.
Dans l'écran CAS les commandes sont en minuscules et dans l'écran HOME les commandes sont en majuscules. Si vous appuyez sur CAS vous travaillez en mode exact, et
Si vous appuyez sur vous travaillez en mode aprroché.
Qu'est-ce que cela change ?
On va considérer par exemple 2 suites u et v définies par ;
$u_0 = \frac{2}{3},\ u_{n+1} = 2u_n - \frac{2}{3}$ $(n \geq 0)$ et
$v_0 = \frac{2}{3},\ v_{n+1} = 2(u_n - \frac{1}{3})$ $(n \geq 0)$.
Dans l'écran CAS
On appuit sur CAS et on tape pour avoir les premiers termes de u :
`2/3` puis `Enter` et on obtient `2/3`.
On tape :
`2*Ans-2/3` puis `Enter`,`Enter`...
et on obtient `2/3`, `2/3`, `2/3`...
En mode exact, i.e.dans l'écran CAS, la suite u est donc stationnaire et vaut $\frac{2}{3}$.
Le résultat est ici conforme au résultat théorique.
Toujours dans le CAS, on tape pour avoir les premiers termes de v :
`2/3` puis `Enter` et on obtient `2/3`.
On tape :
`2*(Ans-1/3)` puis `Enter`,`Enter`...
et on obtient `2/3`, `2/3`, `2/3`...
En mode exact, i.e. dans l'écran CAS, la suite v est donc stationnaire et vaut $\frac{2}{3}$.
Le résultat est encore ici conforme au résultat théorique.
Dans l'écran HOME
Maintenant on appuit sur et pour avoir les premiers termes de u, on tape la valeur de u_0 :
`2/3` puis `Enter` et on obtient `0.666666666667` puis.
on tape :
`2*Ans-2/3` puis `Enter`, `Enter`, `Enter`...
et on obtient `0.666666666663`,`0.666666666663`...
Le résultat est ici presque conforme au résultat théorique.
En mode approché i.e. dans l'écran HOME (touche), la suite u est donc stationnaire à partir de $n > 0$ et vaut 0.666666666663.
Toujours dans HOME (touche), on tape pour avoir les premiers termes de v :
`2/3` puis `Enter` et on obtient `0.666666666667`.
On tape :

`2*(Ans-1/3)` puis `Enter`, `Enter`, `Enter`...
et on obtient
v_1=`0.666666666668`, v_2=`0.666666666670`, v_3=`0.666666666674`puis
`0.666666666682`, `0.666666666682`, `0.666666666698`
`0.666666666730`, `0.666666666794`, `0.666666666922`
etc... et apres 51 ou 52 `Enter` on obtient :
v_{40}`1.76617829443` et v_{50}`2 252.46648036` etc.. En mode approché, i.e. dans l'écran HOME (touche), la suite v tend donc vers $+\infty$.
On voit donc qu'en mode approchè les erreurs de calculs se cumulent et que les résultats affichés ne sont pas toujours conformes aux résultats théoriques !
Comment sont fait les calculs dans HOME
Dans HOME, les nombres réels sont affichés au plus 12 chiffres significatifs mais les calculs sont faits avec plus de chiffres puis sont arrondis pour être affichés par exemple :
1/3 sera représenté par 0.333333333333 (avec 12 fois le chiffre 3)
2/3 sera représenté par 0.666666666667 (avec 11 fois le chiffre 6 et un 7)
4/3 sera représenté par 1.33333333333 (avec 1 puis 11 fois le chiffre 3)
2*0.666666666667 ou 2*0.666666666663 sera représenté par 1.33333333333 (avec 1 puis 11 fois le chiffre 3)
Pour le calcul de u on tape u_0 :
`2/3` on obtient 0.666666666667 puis,
`2*Ans-2/3` on obtient 1.33333333333-0.666666666667=0.666666666663 puis,
`2*Ans-2/3` on obtient puisque 1.33333333333-0.666666666667=0.666666666663
etc...la suite u est donc stationnaire pour $n > 0$ et vaut 0.666666666663.
Pour la suite v le calcul est fait après avoir mis 2 en facteur.
On tape v_0 :
`2/3` on obtient 0.666666666667 puis,
`2*(Ans-1/3)` dans les différentes opérations on a toujours 12 décimales, on obtient :
2*(0.666666666667-0.333333333333)=2*0.333333333334=0.666666666668.
On a donc :
si $A := 0.666666666666$ et $B := 0.333333333333$, on a $A == 2 * B$ et $B == A - B$ mais, $2/3 = A + 10^-12$ et $1/3 = B$
On a donc :
$v_0 = 2/3 = A + 10^-12$ $v_1 = 2 * (A + 10^-12 - B) = 2 * (B + 10^-12) = A + 2 * 10^-12$ puis
$v_2 = 2 * (A + 2 * 10^-12 - B) = A + 2^2 * 10^-12$ puis...
$v_{38} = A + 2^{38} * 10^-12 = 0.94154457361$
$v_{39} = A + 2^{39} * 10^-12 = 1.21642248055$
$v_{40} = A + 2^{40} * 10^-12 = 1.76617829445$
...
$v_{50} = A + 2^{50} * 10^-12 = 1126.56657351$
$v_{51} = A + 2^{51} * 10^-12 = 2252.46648036$
puis la formule risque de n'être plus vraie à cause des erreurs d'arrondis près....
Si on utilise la commande `ITERATE` qui itère, en débutant par la valeur `2/3`, 90 fois la fonction qui à `X` fait correspondre `2*(X-1/3)`, on tape :
`ITERATE(2*(X-1/3),x,2/3,90)` on obtient :
`1.23794003934E15`

et
`ITERATE(2*(X-1/3),x,2/3,91)` on obtient :
`2.4758800788=2*1.23794003934E15`
Donc $v_n = 2^{n-90} * u_{90}$ et quand n tend vers l'infini $v_n = 2^{n-90} * u_{90}$ tend vers l'infini.

0.2 Pour reinitialiser et pour effacer

Pour reinitialiser la calculatrice :

- Enfoncer les touches F O C (sans être en mode `ALPHA`),
- Faire reset avec le trombone en laissant les touches enfoncées,
- Relacher les touches puis choisir 4 FLS Utility, puis 3 Format Disk C, puis Esc puis 9 Reset.

Pour effacer :

- le dernier caractère entré il faut faire `Del` (c'est la grosse flèche noire).
- la ligne d'entrée il faut faire `ON`.
- le dernier résultat ou la dernière commande de l'historique il faut faire `Shift-Del`
- tout l'historique il faut faire `Shift-Esc` (`Clear`).

0.3 L'écran tactile

On remarquera que les menus au bas de l'écran (nommé ici menus du bandeau) ne sont accessibles qu'en les touchant avec le doigt : il n'y a plus de touches `F1..F6`!
L'écran est tactile et cela permet de recopier tres facilement une ligne de commande ou une réponse de l'historique ou de lire ou relire une réponse trop longue, de sélectionner un menu puis une commande de la touche .
Pour cela :

- il suffit de rechercher la commande ou la réponse à copier en naviguant dans l'historique avec un doigt, puis de sélectionner la commande ou la réponse à copier toujours avec un doigt et d'appuyer sur `Copier` du bandeau quand la ligne est en surbrillance ou on appuie 2 fois rapidement avec le doigt sur la ligne à recopier,
- pour lire une réponse trop longue il suffit de balayer la ligne de la réponse avec un doigt.
- on ouvre un menu avec un doigt ou avec son numéro, on fait de même si il y a un sous-menu, puis on sélectionne la fonction avec un doigt ou avec son numéro et cela provoque l'écriture de la fonction dans la ligne de commande à gauche : il ne reste plus qu'à taper les paramètres de cette fonction et de valider avec `Enter`. Le résultat s'écrit alors à droite.

0.4 Le touches

- CAS
  Il faut appuyer sur la touche CAS pour faire du calcul formel. Les lettres minuscules sont alors accessibles en mode ALPHA et la touche xtθn permet d'avoir x directement.

-
  Il faut appuyer sur la touche pour quitter le calcul formel et faire du calcul numérique.
- Apps
  Il faut appuyer sur la touche Apps pour utiliser les différentes Applications qui ont chacune 3 vues : une vue symbolique qui contient les commandes demandées (touche Symb), une vue graphique qui exécute les commandes graphiques (touche Plot) et une vue numérique pour les résultats numériques (touche Num).
- Menu
  La touche Menu renvoie un menu spécifique selon ce que l'on fait.
  Par exemple, depuis le CAS ou depuis vous pouvez echanger des donnèes entre l'écran CAS et l'écran ,
  depuis l'écran Plot de l'application de géométrie vous pouvez changer la couleur des objets ou faire des figures pleines.
- Help
  La touche Help donne de l'aide sur les différentes commandes qui se trouvent dans le menu Cmds du bandeau ou dans le menu de la touche : il faut mettre cette commande en surbrillance avec les flèches puis appuyer sur Help
- Esc
  La touche Esc permet d'annuler la commande en cours

0.5 La configuration générale

On ouvre l'écran de configuration générale avec Shift-.
Vous pouvez par exemple choisir de :
- Taper les commandes en 2d (choisir Entry : Textbook),
- Avoir les réponses en 2d (cocher Textbook Display),
- que les menus contiennent le nom des commandes plutôt qu'un thème (cocher Menu Display),
- Mettre la calculatrice dans les conditions d'examen

0.6 La configuration du CAS

On tape : Shift-CAS (Settings).
Pour être en mode complexe il faut coché i.
Pour utiliser des variables complexes il faut coché Complex.

Par exemple :
`solve(x^3+2*x^2+x+2=0,x)` renvoie `[-2]` en mode réel
`solve(x^3+2*x^2+x+2=0,x)` renvoie `[-2,-i,i]` en mode complexe
Pour utiliser des racines carrées dans une factoristion il faut cocher :
`Utiliser` √
Par exemple :
`factor(x^2+x-1)` renvoie `x^2+x-1` si `Utiliser` √ est décoché,
`factor(x^2+x-1)` renvoie `(x+(-(sqrt(5))+1)/2)*(x+(sqrt(5)+1)/2)`
si `Utiliser` √ est coché.

0.7 La configuration de la calculatrice `Shift`-⌂

Les touches `Shift`-⌂(`Settings`) permettent de faire les réglages de la calculatrice.
Pour avoir dans les menus ou les sous-menus les noms des commandes `Menu Display` ne doit pas être coché.
Si `Menu Display` est coché les menus et les sous-menus décrivent les commandes et renvoie la commande quand un menu ou un sous-menu est sélectionné.

0.8 Les fonctions de calcul formel

On a accès aux fonctions de calcul formel en appyant sur la touche ⌨.
Ces fonctions sont classées par thème.
Utiliser `Shift`-⌂(`Settings`) et décocher `Menu Display` pour avoir le nom des fonctions et non la déscription de ces fonctions.

Deuxième partie

Le menu CAS de la touche

Chapitre 1

Généralités

1.1 Les calculs dans le CAS

Avec le CAS, on fait du calcul exact.
Avec le CAS on peut utiliser les variables de Home qui ont comme noms une des lettres majuscules et qui ont par défaut la valeur 0 mais aussi des variables qui ont comme noms une chaine de lettres minuscules ou de chiffres commencant par une lettre. Ces variables n'ont par défaut aucune valeur : ces variables sont symboliques (sans valeur) tant qu'on ne leur à pas fait une affectation.
Dans le CAS, les commandes sont en général en minuscules, c'est pourquoi la touche ALPHA permet de taper une minuscule et ALPHA,ALPHA bloque le clavier en minuscules (pas besoin de taper SHIFT).
Avec le CAS, les simplifications ne se font pas automatiquement, seules les parenthèses inutiles sont enlevées et les fractions sont simplifiées. Pour avoir la forme simplifiée d'une expression, il faut utiliser la commande simplify.
On remarquera que la réponse peut se faire dans un éditeur d'équations.

1.2 La priorité des opérateurs

Pour faire les calculs :
- On effectue les calculs mis entre les parenthèses,
- On effectue les puissances,
- On effectue les multiplications et les divitions dans l'ordre de gauche à droite.
- On effectue les additions et les soustractions dans l'ordre de gauche à droite.

1.3 La multiplication implicite

Dans le CAS, pour faire une multiplication, le signe * peut être omis lorsque l'on fait la multiplication d'un nombre par une variable. On a le droit d'écrire 2x mais il faut écrire a*b pour faire le produit de la variable apat b, puisque ab est aussi un nom de variable.
On peut écrire par exemple :
2x+3i+4pi

On ne peut pas écrire :
`(2)x, (2)(x+y), (2x+3)(x+y)`
il faut écrire :
`2x` ou `2*(x+y)` ou `(2x+3)*(x+y)`
Attention `x2` et `xy` désignent le nom d'une variable et `f(x+1)` est la valeur de la fonction `f` en `x+1`.

1.4 Les listes et les séquences dans le CAS

Avec le CAS, les listes (resp les vecteurs) sont parenthésées par `{ }` ou par `[ ]`.
Les indices commencent à `0` si l'indice est mis entre `[ ]` et commencent à 1 si l'indice est mis entre `()`.
Par exemple,on tape :
`l:=[1,2,3,4];`
`ll:={1,2,3,4}; l[2]` ou `ll[2]` renvoie 3
`l(2)` ou `ll(2)` renvoie 2

Avec le CAS, il existe aussi le type séquence qui est une suite d'objets. Les indices d'une séquence commencent à `0`.
Avec ce type la concaténation est aisée.
On tape pour définir la séquence vide :
`s:=NULL;`
On obtient si on n'a pas coché `Textbook` ou `Livre` dans la configuration générale (`Shift HOME`) :
`NULL`
Puis, on tape :
`s:=s,1,2`
On obtient :
`seq[1,2]`
On tape :
`s[1])`
On obtient :
`2`
Alors qu'avec le type liste, on tape pour définir la liste vide :
`l:=[];`
Puis, on tape :
`l:=concat(l,[1,2])`
On obtient :
`[1,2]`
On tape :
`l[1])`
On obtient :
`2`
Pour transformer une liste en séquence on utilise l'opérateur `op`.
On tape :
`op(l)`
On obtient :

`seq[1,2]`
Pour transformer une séquence en liste, il suffit de parenthéser la séquence par `[]`.
On tape :
`[s])`
On obtient :
`[1,2]`

1.5 Différence entre expressions et fonctions

Faire la distinction entre expression et fonction.
Une **expression** est une suite de termes séparés par un signe d'opération.
Un terme est un nombre ou un nom de variable ou un produit ou une parenthése contenant une expression.
Convention La multiplication et la division sont prioritaires sur l'addition et la soustraction.
Le signe $*$ est quelqufois omis dans l'écriture, par exemple on écrit : $2x$ au lieu de $2 * x$.
Une **fonction** rèelle f définie sur I partie de $\mathbb{R}$ est une application qui à tout nombre de x de I fait correspondre une expression $f(x)$. La valeur de la fonction en un point x est donc donnée par une expression.
Exemple avec `HPprime` Je tape dans le CAS :
`xpr:=3*x+2`
je définis ainsi l'expression `xpr`
Je tape :
`f(x):=3*x+2`
je définis ainsi la fonction `f`
Je tape :
`subst(xpr,x=1)` et j'obtiens 5
Je tape :
`f(1)` et j'obtiens 5
Je tape :
`plotfunc(3*x+2)` ou,
`plotfunc(xpr)` ou,
`plotfunc(f(x))`
j'obtiens un seul graphe qui est le graphe de la fonction `f`.
Remarque
Le tracé de la plupart des commandes commencant par `plot` ne se fait pas bien depuis l'écran CAS : il est donc préfierable d'utiliser l'application de géométrie pour faire les graphes correspondant à ces commandes.

1.5.1 Exercice sur les expressions

Voici 6 expressions formées à partir de $T = 1 - x * 2 + x$ en rajoutant des parenthèses :

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

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

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

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

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

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

1/ Y-a-t-il une (ou des) expression(s) égale à T ?
Si oui, pourquoi ?
2/ Calculer les valeurs de ces expressions pour $x = 1$ et pour $x = -1$.
3/ Parmi les expressions A, B, C, D, F, G :
- Lesquelles sont une somme de 2 termes ?
- Lesquelles sont une différence de 2 termes ?
- Lesquelles sont une somme algébrique de 3 termes ?
- Lesquelles sont un produit de 2 termes ?
- Lesquelles sont égales ?
4/ Simplifier les expressions A, B, C, D, F, G.
5/ Écrire toutes les expressions formées à partir de $S = 1 + x/2 * x$ en rajoutant des parenthèses.

1.5.2 Vérifions avec `HPprime`

On tape :

```
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)
```

Puis on tape pour connaitre les expressions égales à `T` :
`A==T`, `B==T`, etc...
On trouve que la réponse de `A==T` est `0` ce qui veut dire que l'expression `A` est différente de `T`.
On trouve que la réponse de `B==T` est `1` ce qui veut dire que l'expression `B` est identique à `T` etc...

1.5.3 Exercice (suite) sur les fonctions

1/ Définir 6 fonctions ayant pour valeurs respectives les expressions A, B, C, D, F, G.
2/ Tracer les graphes de ces fonctions et observer les sur un même graphique.
3/ Parmi ces graphes il y a des droites et des paraboles. Retrouver le graphe de chaque fonction.

1.5.4 Vérifions avec `HPprime`

On tape pour définir les 6 fonctions :

```
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)
```

Puis on tape pour visualiser les graphes :

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

On obtient que 5 courbes de couleurs différentes.

On peut taper progressivement :

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

On voit ainsi que :

- le graphe de `a` est la droite noire,
- le graphe de `b` est une droite rouge,
- le graphe de `c` est la parabole verte,
- le graphe de `d` est la droite jaune qui se supperpose à la droite rouge,
- le graphe de `f` est la droite bleue et,
- le graphe de `g` est la parabole verte.

Chapitre 2

Le menu Algebra

2.1 Simplifier une expression : `simplify`

`simplify` simplifie une expression de façon automatique.
On tape :

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

On obtient :

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

On tape :

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

On obtient :

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

Attention `simplify` est plus efficace lorsqu'on est en `radian` pour simplifier des expressions trigonométriques : pour cela on coche `radian` dans la configuration du `CAS`.
On tape :

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

On obtient :

```
4*(cos(x))^2-2
```

2.2 Factoriser un polynôme sur les entiers : `collect`

`collect` a comme paramètre un polynôme ou une liste de polynômes et éventuellement `sqrt(n)`.
`collect` factorise le polynôme (ou les polynômes de la liste) sur les entiers lorsque les coefficients du polynôme sont entiers ou sur $\mathbb{Q}(\sqrt(n))$, si les coefficients du polynôme sont dans $\mathbb{Q}(\sqrt(n))$ ou si `sqrt(n)` est le second argument.
On tape :

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

On obtient :

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

On tape :

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

On obtient :

```
(x+(-(sqrt(5))-1)/2)*(x-1)*(x+(sqrt(5)-1)/2)
```

Voir aussi `factor` selon que dans la configuration du `CAS` on a coché $\sqrt{}$ ou pas.

2.3 Développer une expression : `expand`

`expand` effectue, sur une expression, la distributivité de la multiplication par rapport à l'addition.
On tape :

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

On obtient :

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

On tape :

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

On obtient :

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

2.4 Factoriser une expression : `factor`

On tape :

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

On obtient en mode rèel :

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

On tape :

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

On obtient en mode rèel :

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

On obtient en mode complexe avec $\sqrt{}$ non coché :

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

On obtient en mode complexe avec $\sqrt{}$ coché :

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

On tape :

```
factor(x^6+1,sqrt(3))
```

On obtient en mode complexe avec $\sqrt{}$ coché ou pas :

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

On tape :

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

On obtient si on a décoché $\sqrt{}$ dans la configuration du CAS :

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

On tape :

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

On obtient si on a coché $\sqrt{}$ dans la configuration du CAS :

```
(x+(-(sqrt(5))-1)/2)*(x-1)*(x+(sqrt(5)-1)/2)
```

On tape :

```
factor(expexpand(exp(5*x))-exp(x))
```

On obtient en mode complexe :

```
exp(x)*(-1+exp(x))*(1+exp(x))*(i+exp(x))*(-i+exp(x))
```

2.5 Substituer une variable par une valeur : subst

subst a deux ou trois arguments : une expression dépendant d'un paramètre et une égalité (paramètre=valeur de substitution) ou une expression dependant d'un paramètre , le paramètre et la valeur de substitution.
subst effectue la substitution demandée dans l'expression à condition que le paramètre ne soit pas affecté car subst évalue tout d'abord l'expression et remplace donc le paramètre (si il a été affecté) par sa valeur sans tenir compte de la valeur de substitution donné par le deuxième paramètre.
On tape :

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

On obtient :

On tape :

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

On obtient :

```
(2,5)
```

On tape :

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

On obtient :

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

2.6 Décomposer en éléments simples : `partfrac`

`partfrac` a comme argument une fraction rationnelle.
`partfrac` renvoie sa décomposition en éléments simples.
On tape :

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

On obtient :

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

On tape :

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

On obtient :

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

2.7 Extract

2.7.1 Numérateur d'une fraction après simplification : `numer`

`numer` a comme argument une fraction ou une fraction rationnelle et renvoie le numérateur de cette fraction simplifiée.
On tape :

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

On obtient :

```
7
```

On tape :

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

On obtient :

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

2.7.2 Dénominateur d'une fraction après simplification : `denom`

`denom` a comme argument une fraction ou une fraction rationnelle et renvoie le dénominateur de cette fraction simplifiée.
On tape :

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

On obtient :

```
2
```

On tape :

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

On obtient :

```
x^4-1
```

2.7.3 Pour avoir le membre de gauche d'une équation : `left`

`left` ou `lhs` a comme paramètre une équation ou un intervalle.
`left` ou `lhs` renvoie le membre de gauche de l'équation ou de la borne gauche de l'intervalle.
On tape :On tape :

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

On obtient :

```
a
```

On tape :

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

On obtient :

```
a
```

2.7.4 Pour avoir le membre de droite d'une équation : `right`

`right` ou `rhs` a comme paramètre une équation ou un intervalle.
`right` ou `rhs` renvoie le membre de droite de l'équationou de la borne droite de l'intervalle.
On tape :

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

On obtient :

```
3
```

On tape :

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

On obtient :

```
2*a+1
```

Chapitre 3

Le menu Calculus

3.1 Définition d'une fonction : := et -> (Sto▶)

Pour définir par exemple la fonction `f` qui à `x` fait correspondre `x^3+ln(x)`, on tape :
`f:=x-> x^3+ln(x)`
ou on tape :
`f(x):= x^3+ln(x)`

3.2 diff ou '

`diff` ou `'` calcule la dérivée d'une expression ou d'une fonction d'une variable et calcule aussi les dérivées partielles d'une expression de plusieurs variables.

- Dérivée d'une expression d'une variable
  On tape :
  `diff(x^3+ln(x))`
  Ou on tape (`'` est obtenu avec `Shift-()` (") en effaçant un `'`) :
  `(x^3+ln(x))'`
  On obtient l'expression de la derivèe de `x^3+ln(x)` par rapport à `x` :
  `3*x^2+1/x`
  On tape :
  `diff(y^3+ln(y),y)`
  Ou on tape (`'` est obtenu avec `Shift-()` (") en effaçant un `'`) :
  `(y^3+ln(y),y)'`
  On obtient l'expression de la derivèe de `y^3+ln(y)` par rapport à `y` :
  `3*y^2+1/y`
- Dérivée seconde (ou n-ième) d'une expression d'une variable
  On tape :
  `diff(diff(x^3+ln(x)))`
  Ou on tape (" est obtenu avec `Shift-()`) :
  `(x^3+ln(x))"`
  On obtient l'expression de la derivèe seconde de `x^3+ln(x)` par rapport à `x` :
  `3*2*x-1/x^2`

On tape :

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

Ou on tape (”” est obtenu avec `Shift-() Shift-()`):

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

On obtient l'expression de la derivèe 4-ième de `x^3+ln(x)` par rapport à `x` :

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

– Dérivées partielles d'une expression de plusieurs variables. On tape :

```
diff(x*y*z,{x,y,z})
```

On obtient l'expression les derivèes partielles par rapport à `x`, par rapport à `y` et par rapport à `z`,de `x*y*z` :

```
{y*z,x*z,x*y}
```

– Dérivée d'une fonction
On définit la fonction `f`, on tape :

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

On obtient :

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

On tape :

```
g:=diff(f)
```

Ou on tape (’ est obtenu avec `Shift-()` (”) en effaçant un ’) :

```
g:=f'
```

On obtient la fonction `g` qui est la fonction dérivée de `f` :

```
x->3*x^2+1/x
```

– Dérivée seconde (ou n-ième) d'une fonction
On définit la fonction `f`, on tape :

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

On obtient

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

On tape :

```
h:=diff(diff(f))
```

Ou on tape (” est obtenu avec `Shift-()`):

```
h:=f"
```

On obtient la fonction `h` qui est la fonction dérivée seconde de `f` :

```
x->3*2*x-1/x^2
```

3.3 int

`int` permet de calculer une primitive d'une expression ou d'une fonction ou une intégrale définie.

– Primitive d'une expression
On tape :

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

On obtient une primitive de `x^3+ln(x)` par rapport à `x` :

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

On tape :

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

On obtient une primitive de `y^3+ln(y)` par rapport à `y` :

`y*ln(y)-y+y^4/4`

– Primitive d'une fonction
On définit la fonction `f`, on tape :

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

On obtient :

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

On tape :

`g:=int(f)`

On obtient la fonction `g` qui est une primitive de `f` :

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

– Intégrale définie
On tape :

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

Ou on tape :

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

Ou on tape :

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

Ou on tape lorsque `f(x):= x^3+ln(x)` et `g:=int(f)` :

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

On obtient la valeur de $\int_1^2$ `x^3+ln(x)dx` :

`2*ln(2)-(-3/4`

3.4 limit

`limit` permet de calculer la limite d'une expression en un point fini (ou infini). En utilisant un paramètre supplementaire, on peut indiquer si on cherche une limite par valeurs supérieures ou par valeurs inférieures (1 pour dire "par valeurs supérieures" et -1 pour dire "par valeurs inférieures").
On tape :

$\texttt{limit}_{x->1}\texttt{(sin(x)+ln(x))/x}$

On obtient :

`sin(1)`

On tape :

$\texttt{limit}_{x->0}\texttt{(1/x)}$

On obtient :

`infinity`

On tape :

$\texttt{limit}_{x->0^1}\texttt{(1/x)}$

On obtient :

`+infinity`

On tape :

$$\texttt{limit}_{x->0^{-1}}\texttt{(1/x)}$$

On obtient :

```
-infinity
```

3.5 series

`series` permet de faire le développement limité d'une expression de la variable `Var` en `Var=0` (par défaut en `x=0`) à un ordre donné (par défaut 5).
On tape :

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

On obtient :

```
x+1/3*x^3+2/15*x^5+x^6*order_size(x)
```

`order_size` désigne une fonction telle que, quelque soit `r` positif :
`x^r*order_size(x)` tend vers zéro quand `x` tend vers zéro.
Donc lorsqu'on a dans la réponse `(x-a)^n*order_size(x-a)`
cela signifie que l'on a un développement limité à l'ordre $n-1$ au voisinage de `x=a`.
On tape :

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

On obtient :

```
x+1/3*x^3+2/15*x^5+17/315*x^7+62/2835*x^9+x^10*order_size(x)
```

On tape :

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

On obtient :

```
1/2*pi-1/x+1/3*(1/x)^3-1/5*(1/x)^5+
(1/x)^6*order_size(1/x)
```

ici `(1/x)^6*order_size(1/x)` signifie que l'on a un développement limité à l'ordre $6-1=5$ au voisinage de `1/x=0` i.e. au voisinage de $+\infty$
On tape :

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

On obtient :

```
-1/2*pi-1/x-1/3*(-1/x)^3+1/5*(-1/x)^5+
(-1/x)^6*order_size(-1/x)
```

ici `(-1/x)^6*order_size(-1/x)` signifie que l'on a un développement limité à l'ordre $6-1=5$ au voisinage de `-1/x=0` i.e. au voisinage de $-\infty$

3.6 sum

`sum` effectue la somme des éléments d'une liste ou la somme indicée finie et infinie ou calcule la primitive discrète d'une expression.

- somme d'une liste ou d'une séquence
  On tape :
  `l:=[1,2,3,4,5,6,7,8]`
  Ou on tape :
  `l:=1,2,3,4,5,6,7,8`
  Puis, on tape :
  `sum(l)`
  On obtient la somme 1+2+..+8=8*9/2 :
  `36`
- somme indicée finie
  On tape :
  `sum(k,k=1..8)`
  Ou on tape :
  `sum(k,k,1,8)`
  On obtient :
  `36`
- somme indicée infinie
  On tape :
  `sum(1/2^k,k,0,inf)`
  On obtient :
  `2`
- primitive discréte d'une expression
  La primitive discrète de l'expression $f(x)$ est la fonction G verifiant :
  $G(x+1) - G(x) = f(x)$
  `sum` a alors deux arguments : une expression d'une variable (par exemple $f(x)$) et la variable (par exemple x) :
  On tape :
  `sum(x,x)`
  On obtient :
  `(x^2-x)/2`
  Donc :
  $4 + 5 + ...19 = G(20) - G(4) = 190 - 6 = 184$
  On vérifie : `sum(k,k=4..19)` renvoie bien `184`
  On tape :
  `sum(1/(x*(x+1)),x)`
  On obtient :
  `-1/x`
  Donc :
  $1/(1*2) + 1/(2*3) + ...1/(9*10) = -1/10 + 1 = 9/10$
  On vérifie : `sum(seq(1/(k*(k+1)),k,1,9))` renvoie bien `9/10`

3.7 Differential

3.7.1 Le rotationnel : `curl`

`curl` a deux paramètres : une expression F dependant de 3 variables rèelles et un vecteur de dimension 3 indiquant le nom de ces variables.
`curl` désigne le rotationnel de F.
On tape :

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

On obtient :

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

En effet :
`diff(2*x^y,y)-diff(-y^2,z)` renvoie `2*ln(x)*x^y`
`diff(x*z,z)-diff(2*x^y,x)` renvoie `x-2*y*x^(y-1)`
`diff(-y^2,x)-diff(x*z,y)` renvoie `0`
On tape :

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

On obtient :

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

3.7.2 La divergence : `divergence`

`divergence` a deux paramètres : une expression F dependant de n variables rèelles et un vecteur de dimension n indiquant le nom de ces variables.
`divergence` désigne la divergence de F.
On tape :

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

On obtient :

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

En effet :
`diff(x^2+y,x)+diff(x+z+y,y)+diff(z^3+x^2,z)` renvoie :
`2*x+1+3*z^2`

3.7.3 Le gradient : `grad`

`diff` ou `grad`) a deux paramètres : une expression F dependant de n variables rèelles et un vecteur de dimension n indiquant le nom de ces variables.
`derive` renvoie le gradient de F.
On tape :

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

On obtient :

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

En effet :
`diff(2*x^2*y-x*z^3,x)` renvoie `4*x*y-z^3`
`diff(2*x^2*y-x*z^3,y)` renvoie `2*x^2`
`diff(2*x^2*y-x*z^3,z)` renvoie `-3*x*z^2`

3.7.4 La hessienne : `hessian`

`hessian` a deux paramètres : une expression F dependant de n variables rèelles et un vecteur de dimension n indiquant le nom de ces variables.
`hessian` renvoie la hessienne de F qui est la matrice des dérivées d'ordre 2 à savoir `diff(diff(F,[x,y,z]),[x,y,z])`.
On tape :

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

On obtient :

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

En effet :
`diff(diff(2*x^2*y-x*z,x),[x,y,z])` renvoie : `[4*y,4x,-1]`
`diff(diff(2*x^2*y-x*z,y),[x,y,z])` renvoie : `[4x,0,0]`
`diff(diff(2*x^2*y-x*z,z),[x,y,z])` renvoie : `[-1,0,0]`
Remarque
Pour avoir la hessienne aux points critiques, on cherche les points critiques, on tape :

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

On obtient :

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

Puis, on calcule la hessienne en ces points, on tape :

```
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])
```

On obtient :

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

et après simplification :

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

3.8 Integral

3.8.1 Intégration par parties : `ibpdv`

`ibpdv` permet de chercher une primitive (ou de calculer une intégrale définie) d'une expression de la forme $u(x).v'(x)$.
`ibpdv` a deux paramètres pour les primitives et cinq paramètres pour les intégrales définies :

- soit une expression de la forme $u(x).v'(x)$ et $v(x)$ (ou une liste de deux expressions $[F(x), u(x) * v'(x)]$ et $v(x)$),
- soit une expression de la forme $g(x)$ et 0 (ou une liste de deux expressions $[F(x), g(x)]$ et 0).
- pour les intégrales définies, il faut rajouter trois autres paramètres : le nom de la variable et les bornes.

Valeur renvoyée par `ibpdv` selon ses paramètres :

- `ibpdv(u(x).v'(x),v(x))` (resp `ibpdv([F(x),u(x).v'(x)],v(x))`) renvoie :
  si $v(x) \neq 0$, une liste formée de $u(x).v(x)$ et de $-v(x).u'(x)$ (resp une liste formée de $F(x) + u(x).v(x)$ et de $-v(x).u'(x)$),
- `ibpdv(g(x),0)` (resp `ibpdv([F(x),g(x)],0)`) renvoie :
  une primitive $G(x)$ de $g(x)$ (resp $F(x) + G(x)$) où `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)`) renvoie :
  - si $v(x) \neq 0$, une liste formée de $u(b).v(b) - u(a).v(a)$ et de $-v(x).u'(x)$ (resp une liste formée de $F(b) + u(b).v(b) - F(a) - u(a).v(a)$ et de $-v(x).u'(x)$),
  - si le deuxième argument est nul, `ibpdv(g(x),0,x,a,b)` renvoie : $G(b) - G(a)$ où $G(x)$ est une primitive du premier argument $g(x)$ (resp `ibpdv([F(x),g(x)],0,x,a,b)` renvoie $F(x) + G(b) - G(a)$ où $G(x)$ est une primitive de $g(x)$).

On tape :

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

On obtient :

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

puis on tape

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

On obtient :

```
-x+x.ln(x)
```

On tape :

```
ibpdv(ln(x),x,x,1,2)
```

On obtient :

```
[2*ln(2),-1]
```

On tape :

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

On obtient :

```
[3*ln(3)-2*ln(2),-1]
```

puis on tape :

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

On obtient :

```
-1+3*ln(3)-2*ln(2)
```

3.8.2 Intégration par parties : `ibpu`

`ibpu` permet de chercher une primitive (ou de calculer une intégrale définie) d'une expression de la forme $u(x).v'(x)$.
`ibpu` a deux paramètres pour les primitives et cinq paramètres pour les intégrales définies :

- soit une expression de la forme $u(x).v'(x)$ et $u(x)$ (ou une liste de deux expressions $[F(x), u(x) * v'(x)]$ et $u(x)$),
- soit une expression de la forme $g(x)$ et 0 (ou une liste de deux expressions $[F(x), g(x)]$ et 0).
- pour les intégrales définies, il faut rajouter trois autres paramètres : le nom de la variable et les bornes.

Valeur renvoyée par `ibpu` selon ses paramètres :

- `ibpu(u(x).v'(x),u(x))` (resp `ibpu([F(x),u(x).v'(x)],u(x))`) renvoie :
  si $u(x) \neq 0$, une liste formée de $u(x).v(x)$ et de $-v(x).u'(x)$ (resp une liste formée de $F(x) + u(x).v(x)$ et de $-v(x).u'(x)$),
- `ibpu(g(x),0)` (resp `ibpu([F(x),g(x)],0)`) renvoie :
  $G(x)$ une primitive de $g(x)$ (resp $F(x) + G(x)$ où `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)`) renvoie :
  - si $u(x) \neq 0$, une liste formée de $u(b).v(b) - u(a).v(a)$ et de $-v(x).u'(x)$ (resp une liste formée de $F(b) + u(b).v(b) - F(a) - u(a).v(a)$ et de $-v(x).u'(x)$),
  - si le deuxième argument est nul, `ibpu(g(x),0,x,a,b)` renvoie :
    $G(b) - G(a)$ où $G(x)$ une primitive de $g(x)$ (resp $F(x) + G(b) - G(a)$ où $G(x)$ est une primitive de $g(x)$).

On tape :

```
ibpu(ln(x),ln(x))
```

On obtient :

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

puis on tape :

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

On obtient :

```
-x+x.ln(x)
```

On tape :

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

On obtient :

```
[3*ln(3)-2*ln(2),-1]
```

puis on tape :

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

On obtient :

```
-1+3*ln(3)-2*ln(2)
```

3.8.3 Évaluer une primitive : `preval`

`preval` a trois paramètres : une expression `F(x)` dépendant de la variable `x`, et deux expressions `a` et `b`.
`preval` effectue `F(b)-F(a)`.
`preval` est utile pour calculer une intégrale définie à partir d'une primitive : on calcule une primitive, puis on évalue cette primitive entre les deux bornes de l'intégrale.
On tape :

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

On obtient :

```
6
```

On tape :

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

On obtient :

```
x*ln(x)-x
```

On tape :

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

On obtient :

```
3*ln(3)-3-2*ln(2)+2
```

3.9 Limits

3.9.1 Somme de Riemann : `sum_riemann`

`sum_riemann` a deux arguments : une expression `Xpr` dépendant de deux variables et la liste des noms de ces deux variables.
`sum_riemann(Xpr(n,k),[n,k])` renvoie un équivalent, au voisinage de $n = +\infty$, de $\sum_{k=1}^{n} Xpr(n,k)$ ou de $\sum_{k=0}^{n-1} Xpr(n,k)$ ou de $\sum_{k=1}^{n-1} Xpr(n,k)$, lorsque la somme considérée est une somme de Riemann associée à une fonction continue sur [0,1] ou répond quand la recherche a été infructueuse `"ce n'est probablement pas une somme de Riemann"`.
Soit $S_n = \sum_{k=1}^{n} \frac{k^2}{n^3}$.
Calculer $\lim_{n\to+\infty} S_n$.
On tape :

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

On obtient :

```
1/3
```

Soit $S_n = \sum_{k=1}^{n} \frac{k^3}{n^4}$.
Calculer $\lim_{n\to+\infty} S_n$.
On tape :

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

On obtient :

```
1/4
```

Calculer $\lim_{n\to+\infty} (\frac{1}{n+1} + \frac{1}{n+2} + ... + \frac{1}{n+n})$.
On tape :

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

On obtient :

```
ln(2)
```

3.9.2 Développement limité : `taylor`

`taylor`
peut avoir de un à quatre paramètres :
l'expression à développer, `x=a` (par défaut `x=0`), l'ordre du développement (par défaut `5`), ou encore :
l'expression à développer, `x`, l'ordre du développement (par défaut `5`) et le point au voisinage duquel on veut le développement (par défaut `0`).
Remarque on peut aussi mettre `x,`a`,`n au lieu de `x=`a`,`n
`taylor` renvoie un polynôme en `x-a`, plus un reste que `Xcas` écrit :

`(x-a)^n*order_size(x-a)`
cela signifie que l'on a un développement limité à l'ordre $n-1$ (ou à l'ordre $p < n$).
En effet `order_size` désigne une fonction telle que, quelque soit `r` positif :
`x^r*order_size(x)` tend vers zéro quand `x` tend vers zéro.
Par exemple, les fonctions constantes, la fonction $\log$ (ou $\ln$), sont des fonctions `order_size`.
On tape :

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

Ou on tape (attention à l'ordre des arguments !) :

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

On obtient :

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

3.9.3 Division selon les puissances croissantes : `divpc`

`divpc` a trois arguments : deux polynômes $A(x),\ B(x)$ (avec $B(0) \neq 0$) et un entier n.
`divpc` renvoie le quotient $Q(x)$ de la division de $A(x)$ par $B(x)$ selon les puissances croissantes avec `degree`$(Q) \leq n$ ou $Q = 0$.
$Q(x)$ est donc le développement limité, d'ordre n, de $\dfrac{A(x)}{B(x)}$ au voisinage de $x = 0$.
On tape :

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

On obtient :

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

Attention ! ! ! cette commande ne marche pas si les polynômes sont écrits avec la liste de leurs coefficients.

3.10 Transform

3.10.1 Transformée de Laplace : `laplace`

`laplace` a 1, 2 ou 3 arguments :
l'expression que l'on transforme et éventuellement le nom d'une ou deux variables.
L'expression est une expression de la variable courante (ici x) ou l'expression que l'on transforme est une expression de la variable donnèe comme deuxième argument.
`laplace` est la transformée de Laplace de l'expression donnée comme argument.
Le résultat de `laplace` est une expression de variable le troisième argument ou par défaut le second argument ou par défaut x.
On tape :

```
laplace(sin(x))
```

On obtient :

```
1/(x^2+1)
```

Ou on tape :

```
laplace(sin(t),t)
```

On obtient :

```
1/(t^2+1)
```

Ou on tape :

```
laplace(sin(x),x,t)
```

On obtient :

```
1/(t^2+1)
```

Ou on tape :

```
laplace(sin(t),t,s)
```

On obtient :

```
1/(s^2+1)
```

3.10.2 Transformée de Laplace inverse : `invlaplace`

3.10.3 invlaplace

`ilaplace` a 1, 2 ou 3 arguments :
l'expression que l'on transforme et éventuellement le nom d'une ou deux variables. L'expression est une expression de la variable courante (ici x) ou l'expression que l'on transforme est une expression de la variable donnèe comme deuxième argument.
`invlaplace` est la transformée de Laplace inverse de l'expression donnée comme argument. Le résultat de `invlaplace` est une expression de variable le troisième argument ou par défaut le second argument ou par défaut x.
On tape :

```
invlaplace(1/(x^2+1))
```

On obtient :

```
sin(x)
```

Ou on tape :

```
invlaplace(1/(t^2+1),t)
```

On obtient :

```
sin(t)
```

Ou on tape :

```
invlaplace(1/(t^2+1),t,x)
```

On obtient :

```
sin(x)
```

Remarque :
On utilise la transformée de Laplace (`laplace`) et la transformée de Laplace inverse (`ilaplace` ou `invlaplace`) pour résoudre des équations différentielles linéaires à coefficients constants, par exemple :

$$y'' + p.y' + q.y \ = \ f(x)$$

$$y(0) = a \ y'(0) = b$$

En notant $\mathcal{L}$ la transformée de Laplace, on a les relations suivantes :

$$\begin{aligned} \mathcal{L}(y)(x) &= \int_0^{+\infty} e^{-x.u} y(u) du \\ \mathcal{L}^{-1}(g)(x) &= \frac{1}{2i\pi} \int_C e^{z.x} g(z) dz \end{aligned}$$

où C est une courbe fermée contenant les pôles de `g`.
Exemple :
Résoudre :

$$y'' - 6.y' + 9.y \ = \ x.e^{3.x}, \quad y(0) = c_0, \quad y'(0) = c_1$$

Ici, $p = -6, \ q = 9$.
On tape :

```
laplace(x*exp(3*x))
```

On obtient :

```
1/(x^ 2-6*x+9)
```

On tape :

```
ilaplace((1/(x^2-6*x+9)+(x-6)*c_0+c_1)/(x^2-6*x+9))
```

On obtient

```
(216*x^3-3888*x*c_0+1296*x*c_1+1296*c_0)*exp(3*x)/1296
```

après simplification et factorisation (commande `factor`) la solution y s'écrit :

```
(-18*c_0*x+6*c_0+x^3+6*x*c_1)*exp(3*x)/6
```

On peut bien sûr taper directement :

```
desolve(y''-6*y'+9*y=x*exp(3*x),y)
```

On obtient bien :

```
exp(3*x)*(-18*c_0*x+6*c_0+x^3+6*x*c_1)/6
```

3.10.4 La transformée de Fourier rapide : `fft`

`fft` a comme argument une liste (ou une séquence) $[\mathtt{a}_0, ..\mathtt{a}_{\mathtt{N}-1}]$ où `N` est une puissance de deux.
`fft` renvoie la liste $[\mathtt{b}_0, ..\mathtt{b}_{\mathtt{N}-1}]$ tel que pour `k=0..N-1` on ait :
$\mathtt{fft}([\mathtt{a}_0, ..\mathtt{a}_{\mathtt{N}-1}])[\mathtt{k}] = \mathtt{b}_\mathtt{k} = \sum_{\mathtt{j}=0}^{\mathtt{N}-1} \mathtt{x}_\mathtt{j}\omega_\mathtt{N}^{-\mathtt{k}\cdot\mathtt{j}}$ avec ω_N racine N-ième de l'unité.
On tape :

```
fft(0,1,1,0)
```

On obtient :

```
[2., -1-i, 0., -1+i]
```

Remarque On peut aussi travailler sur un corps fini $\mathbb{Z}/p\mathbb{Z}$, en indiquant une racine N-ième primitive de l'unité en 2ième argument et p en 3ième argument de `fft`.

3.10.5 L'inverse de la transformée de Fourier rapide : `ifft`

`ifft` a comme argument une liste ou une séquence $[\mathtt{b}_0, ..\mathtt{b}_{\mathtt{N}-1}]$ où `N` est une puissance de deux.
`ifft` renvoie la liste $[\mathtt{a}_0, ..\mathtt{a}_{\mathtt{N}-1}]$ tel que :
$\mathtt{fft}([\mathtt{a}_0, ..\mathtt{a}_{\mathtt{N}-1}]) = [\mathtt{b}_0, ..\mathtt{b}_{\mathtt{N}-1}]$.
On tape :

```
ifft([2,-1-i,0,-1+i])
```

Ou on tape :

```
ifft(2,-1-i,0,-1+i)
```

On obtient :

```
[0., 1., 1., 0.]
```

Remarque On peut aussi travailler sur un corps fini $\mathbb{Z}/p\mathbb{Z}$, en indiquant une racine N-ième primitive de l'unité en 2ième argument et p en 3ième argument de `ifft`.

Chapitre 4

Le menu Solve

4.1 Résolution d'équations : `solve`

`solve` permet de résoudre une équation ou un système d'équations polynômiales.
`solve` a 1 ou 2 arguments qui sont une expression xpr en x ou une expression xpr d'une variable var et le nom de cette variable var.
`solve` resout $xpr = 0$ l'inconnue étant x ou var **Attention**
La deuxième variable peut spécifier un intervalle par exemple $x = a..b$ pour n'avoir que les solutions dans l'intervalle $[a; b]$ **mais** dans ce cas les solutions seront numériques et `solve` est alors identique à `fsolve`, par exemple :
`solve(t^2-2,t=0..2)` ou `fsolve(t^2-2,t=0..2)` renvoie `[1.41421356237]` alors que `solve(t^2-2,t)` renvoie `[-(sqrt(2)),sqrt(2)]`. On tape :

```
solve(x^2-3*x+2=0)
```

On obtient :

```
{1,2}
```

On tape :

```
solve(x^4-1=0)
```

On obtient :

```
{-sqrt(2),sqrt(2)}
```

On tape :

```
solve([x+y=3,x*y=2],[x,y])
```

Ou on tape :

```
solve({x+y=3,x*y=2},{x,y})
```

On obtient :

```
{[1,2],[2,1]}
```

On tape :

```
solve([-x^2+y=2,x^2+y=0],[x,y])
```

Ou on tape :

```
solve({-x^2+y=2,x^2+y=0},{x,y})
```

On obtient :

```
{}
```

4.2 Zéros d'une expression : `zeros`

`zeros` a comme paramètre une expression.
`zeros` renvoie la liste des éléments qui annulent l'expression.
Selon le mode choisi, si on est en mode réel (`complex_mode:=0` ou si `i` n'est pas coché dans le Settings du CAS) les zéros seront réels et si on est en mode complexe (`complex_mode:=1` ou si `i` est coché dans le Settings du CAS)) les zéros seront complexes.
On tape :

```
zeros(x^2-3*x+2)
```

On obtient :

```
[2,1]
```

On tape :

```
zeros(x^4-1)
```

On obtient :

```
[1,-1]
```

On tape :

```
zeros([x+y-3,x*y-2],[x,y])
```

Ou on tape :

```
zeros({x+y-3,x*y-2},{x,y})
```

On obtient :

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

On tape :

```
zeros([-x^2+y-2,x^2+y],[x,y])
```

Ou on tape :

```
zeros({-x^2+y-2,x^2+y},{x,y})
```

On obtient :

```
[]
```

4.3 Résoudre des équations dans $\mathbb{C}$: cSolve

cSolve résout une équation ou un système d'équations polynômiales dans $\mathbb{C}$ sans avoir besoin d'être en mode complexe. **Remarque**
Différence entre solve et csolve : En mode complexe solve renvoie le même résultat que csolve (pour csolve que l'on soit en mode complexe ou réel cela importe peu). Ainsi, si on ne veut pas que le résultat dépende du mode, pour avoir les solutions complexes il est préférable d'utiliser csolve.
On tape en mode réel ou complexe : On tape :

```
cSolve(x^4-1=3)
```

On obtient :

```
[sqrt(2),-sqrt(2),sqrt(2)*i,-sqrt(2)*i]
```

On tape :

```
cSolve([-x^2+y=2,x^2+y=0],[x,y])
```

Ou on tape :

```
cSolve({-x^2+y=2,x^2+y=0},{x,y})
```

On obtient :

```
{[i,1],[-i,1]}
```

4.4 Zéros complexe d'une expression : cZeros

cZeros a comme paramètre une expression.
cZeros renvoie la liste des éléments complexes qui annulent l'expression.
Remarque
Différence entre zeros et cZeros : En mode complexe zeros renvoie le même résultat que cZeros (pour cZeros que l'on soit en mode complexe ou réel cela importe peu). Ainsi, si on ne veut pas que le résultat dépende du mode, pour avoir les solutions complexes il est préférable d'utiliser cZeros.
On tape en mode réel ou complexe :

```
cZeros(x^4-1)
```

On obtient :

```
[1,-1,-i,i]
```

On tape :

```
cZeros([-x^2+y-2,x^2+y],[x,y])
```

Ou on tape :

```
cZeros({-x^2+y-2,x^2+y},{x,y})
```

On obtient :

```
[[-i,1],[i,1]]
```

4.4.1 Équations différentielles : `deSolve`

`deSolve` permet de résoudre :

- les équations différentielles linéaires à coefficients constants du premier ou du deuxième ordre,
- les équations différentielles linéaires du premier ordre,
- les équations différentielles du premier ordre incomplète en y,
- les équations différentielles du premier ordre incomplète en x,
- les équations différentielles du premier ordre à variables séparées,
- les équations différentielles du premier ordre homogènes ($y' = F(y/x)$),
- les équations différentielles du premier ordre ayant un facteur intégrant,
- les équations différentielles de Bernoulli ($a(x)y' + b(x)y = c(x)y^n$),
- les équations différentielles de Clairaut ($y = x * y' + f(y')$).

Les paramètres de `deSolve` :

- quand l'équation différentielle est du premier ordre, que la variable est x et que l'inconnue est y, les paramètres sont :
  l'équation différentielle ou
  l'équation différentielle suivie de la liste $[x_0, y_0]$ qui donne comme condition initiale $y(x_0) = y_0$.

- quand la variable est x,
  les paramètres sont : l'équation différentielle (ou la liste formée par l'équation différentielle et les conditions initiales) et l'inconnue y.
  Dans l'équation différentielle y s'écrit y et y' s'écrit y' et y" s'écrit y" car on dérive par rapport à la variable x. Par exemple : `deSolve(y"+2*y'+y,y)` et `deSolve([y"+2*y'+y,y(0)=1,y'(0)=0],y)`.
- quand la variable n'est pas x (par exemple t),
  les paramètres sont : l'équation différentielle (ou la liste formée par l'équation différentielle et les conditions initiales), la variable t et l'inconnue y ou l'inconnue $y(t)$ (la variable est alors t et l'inconnue est y).
  Dans l'équation différentielle y s'écrit $y(t)$ et $y\prime$ s'écrit `diff(y(t),t)`, y'' s'écrit `diff(y(t),t$2)`.
  Par exemple :
  `deSolve(diff(y(t),t$2)+2*diff(y(t),t)+y(t),y(t))` ; ou
  `deSolve(diff(y(t),t$2)+2*diff(y(t),t)+y(t),t,y)` ;
  et

```
deSolve([diff(y(t),t$2)+2*diff(y(t),t)+y(t),
         y(0)=1,y'(0)=0],y(t)); ou
deSolve([diff(y(t),t$2)+2*diff(y(t),t)+y(t),
         y(0)=1,y'(0)=0],t,y);
```

On tape (en tapant `Shift-()` pour ") :

```
deSolve(y"+y=cos(x),y)
```

ou encore :

```
deSolve((diff(diff(y))+y)=(cos(x)),y)
```

On trouve :

```
G_0*cos(x)+(x+2*G_1)/2*sin(x)
```

`c_0, c_1` sont les constantes d'intégration : `y(0)=c_0` et `y'(0)=c_1`.
On tape, si veut les solutions vérifiant $y(0) = 1$:

```
deSolve([y"+y=cos(x),y(0)=1],y)
```

On obtient

```
[cos(x)+(x+2*c_1)/2*sin(x)]
```

les composantes de ce vecteur sont solutions (ici on a une seule composante car on obtient une seule solution dépendant de la constante `c_1`).

4.5 Résolution numérique d'équations`nSolve`

`nSolve` permet de résoudre numériquement des équations non polynomiales : $f(x) = 0$ pour $x \in]a, b[$.
Les paramètres de `nSsolve` sont `f(x)=0`, `x=x0` où `x0` est un point de $]a, b[$.
On tape :

```
nSolve(x^2-2=0,x=1)
```

On obtient :

```
1.41421356237
```

On tape :

```
nSolvex^2-2=0,x=-1)
```

On obtient :

```
-1.41421356237
```

4.6 Solution approchée de y'=f(t,y) :`odesolve`

Soit f une fonction de $\mathbb{R}^2$ y'=f(t,y) dans $\mathbb{R}$.
`odesolve` renvoie la valeur approchée $y(t1)$ de la solution de léquation differentielle $y' = f(t, y)$ lorsque $y(t_0) = y_0$.
`odesolve` a comme paramètres :

- `odesolve(f(t,y),[t,y],[t0,y0],t1)` ou
  `odesolve(f(t,y),t=t0..t1,y,y0)` ou
  `odesolve(t0..t1,f,y0)` ou
  `odesolve(t0..t1,(t,y)->f(t,y),y0)`
  renvoie la valeur approchée de $y(t1)$ lorsque $y(t)$ est la solution de $y'(t) = f(t, y(t))$ qui vérifie $y(t0) = y0$.
- On peut ajouter un paramètre optionnel pour indiquer la discrétisation en temps souhaitée (`tstep=valeur`). Cette valeur n'est pas forcément respectée par le solver.
- On peut indiquer en paramètre optionnel `curve` pour obtenir la liste des $[t, [y(t)]]$ calculés au lieu de la seule valeur de $y(t1)$.

On tape :

```
odesolve(sin(t*y),[t,y],[0,1],2)
```

ou :

```
odesolve(sin(t*y),t=0..2,y,1)
```

ou :

```
odesolve(0..2,(t,y)->sin(t*y),1)
```

ou encore on définit la fonction :

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

et on tape :

```
odesolve(0..2,f,1)
```

On obtient :

```
[1.82241255674]
```

puis on tape :

```
odesolve(0..2,f,1,tstep=0.3)
```

On obtient :

```
[1.82241255675]
```

On tape :

```
odesolve(sin(t*y),t=0..2,y,1,tstep=0.5)
```

On obtient :

```
[1.82241255675]
```

On tape :

```
odesolve(sin(t*y),t=0..2,y,1,tstep=0.5,curve)
```

On obtient :

```
[[0.0,[1.0]],[0.3906,[1.07811817892]],[0.760963058921,[1.30972370161]
```

On tape :

```
odesolve(sin(t*y),t=0..2,y,1,curve)
```

Ou on tape :

```
odesolve(sin(t*y),t=0..2,y,1,tstep=0.3,curve)
```

On obtient :

```
[[0.0,[1.0]],[0.3781,[1.07309655677]],[0.6781,[1.24392692452]],[0.978
```

4.6.1 Résoudre un système linéaire : `linsolve`

`linsolve` permet de résoudre un système d'équations linéaires où chaque équation est de la forme $Xpr = 0$ où Xpr est une expression.
`linsolve` a comme paramètres la liste des équations et la liste des variables.
`linsolve` renvoie une liste qui est solution du système d'équations.
`linsolve` permet de résoudre aussi un système d'équations linéaires dans $\mathbb{Z}/n\mathbb{Z}$.
On tape :

```
linsolve([2*x+y+z=1,x+y+2*z=1,x+2*y+z=4],[x,y,z])
```

On obtient :

```
[1/-2,5/2,1/-2]
```

donc

$$x = -\frac{1}{2}, y = \frac{5}{2}, z = -\frac{1}{2}$$

est la solution du système :

$$\begin{cases} 2x + y + z &= 1 \\ x + y + 2z &= 1 \\ x + 2y + z &= 4 \end{cases}$$

Chapitre 5

Le menu Rewrite

5.1 Regrouper les log : `lncollect`

`lncollect` a comme argument une expression contenant des logarithmes.
`lncollect` regroupe les termes en logarithmes. Il est donc préférable de l'utiliser sur une expression factorisée (en utilisant `factor`).
On tape :

```
lncollect(ln(x+1)+ln(x-1))
```

On obtient :

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

On tape :

```
lncollect(exp(ln(x+1)+ln(x-1)))
```

On obtient :

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

5.2 Transformer une puissance en produit de puissances : `powexpand`

`powexpand` permet de transformer une puissance en un produit de puissances.
On tape :

```
powexpand(a^(x+y))
```

On obtient :

```
a^x*a^y
```

5.3 Développer une expression transcendante et de trigo : `tExpand`

`tExpand` a comme argument une expression transcendante et trigonométrique.
`tExpand` est la généralisation de `expexpand`, `lnexpand` et `trigexpand` car elle développe les expressions transcendantes et trigonométriques.
`tExpand` permet, par exemple, de transformer $\ln(x^n)$ en $n\ln(x)$, $\exp(x)^n$ en $\exp(nx)$ et $\sin(2x)$ en $2\sin(x)\cos(x)$.

- `tExpand` a comme argument une expression transcendante et trigonométrique.
  Exemple :
  Développer $\exp(x+y)+\cos(x+y)+\ln(3x^2)$.
  On tape :
  ```
  tExpand(exp(x+y)+cos(x+y)+ln(3*x^2))
  ```
  On obtient :
  ```
  cos(x)*cos(y)-sin(x)*sin(y)+exp(x)*exp(y)+
  ln(3)+2*ln(x)
  ```
- `tExpand` a comme argument une expression trigonométrique.
  `tExpand` développe cette expression en fonction de $\sin(x)$ et $\cos(x)$.
  Exemples
  1. Développer $\cos(x+y)$.
     On tape :
     ```
     tExpand(cos(x+y))
     ```
     On obtient :
     ```
     cos(x)*cos(y)-sin(x)*sin(y)
     ```
  2. Développer $\cos(3x)$.
     On tape :
     ```
     tExpand(cos(3*x))
     ```
     On obtient :
     ```
     4*(cos(x))^ 3-3*cos(x)
     ```
  3. Développer $\dfrac{\sin(3*x)+\sin(7*x)}{\sin(5*x)}$.
     On tape :
     ```
     tExpand((sin(3*x)+sin(7*x))/sin(5*x))
     ```
     On obtient
     ```
     (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)
     ```
     Et, après une simplification en tapant `simplify(Ans)`, on obtient :
     ```
     4*(cos(x))^2-2
     ```

– `tExpand` a comme argument une expression transcendante.
`tExpand` développe cette expression.
Exemples

1. Développer $\exp(x+y)$.
   On tape :

   ```
   tExpand(exp(x+y))
   ```

   On obtient :

   ```
   exp(x)*exp(y)
   ```

2. Développer $\ln(x+y)$.
   On tape :

   ```
   tExpand(log(x*y))
   ```

   On obtient :

   ```
   log(x)+log(y)
   ```

3. Développer $\ln(x^n)$.

   ```
   tExpand(ln(x^n))
   ```

   On obtient :

   ```
   n*ln(x)
   ```

4. Développer $\ln((e^2)+\exp(2*\ln(2))+exp(\ln(3)+\ln(2)))$.
   On tape :

   ```
   tExpand(log(e^2)+exp(2*log(2))+exp(log(3)+log(2)))
   ```

   On obtient :

   ```
   6+3*2
   ```

   Ou on tape :

   ```
   tExpand(log(e^2)+exp(2*log(2)))+
     lncollect(exp(log(3)+log(2)))
   ```

   On obtient :

   ```
   12
   ```

5.4 Exp & Ln

5.4.1 Transformer exp(n*ln(x)) en puissance : `exp2pow`

`exp2pow` permet de transformer une expression de la forme $\exp(n*\ln(x))$ en une puissance de x.
On tape :

```
exp2pow(exp(n*ln(x)))
```

On obtient :

```
x^n
```

Bien voir la différence avec `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`
Mais :
`lncollect(exp(ln(x)+ln(x))) = x^2`
`exp2pow(exp(ln(x)+ln(x))) = x^(1+1)`

5.4.2 Transformer une puissance en une exponentielle : `pow2exp`

`pow2exp` permet de transformer une puissance en exponentielle.
On tape :

```
pow2exp(a^(x+y))
```

On obtient :

```
exp((x+y)*ln(a))
```

5.4.3 Transformer les exponentielles complexes en sin et en cos : `exp2trig`

`exp2trig` a comme argument une expression contenant des exponentielles complexes.
`exp2trig` transforme cette expression en fonction de $\sin(x)$ et de $\cos(x)$.
On tape :

```
exp2trig(exp(i*x))
```

On obtient si `Complex` n'est pas coché dans la configuration du CAS (`Shift-CAS`) :

```
cos(x)+i*sin(x)
```

On obtient si `Complex` est coché dans la configuration du CAS :

```
exp(im(x))*(cos(re(x))+(i)*sin(re(x)))
```

On tape :

```
exp2trig(exp(i*x)+exp(-i*x))
```

On obtient :

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

puis on sélectionne cette réponse et on appuie sur `simplify` On obtient :

```
2*cos(x)
```

5.4.4 Développer les exponentielles : `expexpand`

`expexpand` a comme argument une expression contenant des exponentielles.
`expexpand` développe cette expression.
On tape :

```
expexpand(exp(3*x))
```

On obtient :

```
exp(x)^3
```

On tape :

```
expexpand(exp(3*x)+exp(2*x+2))
```

On obtient :

```
exp(x)^3+exp(x)^2*exp(2)
```

5.5 Sine

5.5.1 Transformer les arcsin en arccos : `asin2acos`

`asin2acos` a comme argument une expression trigonométrique.
`asin2acos` transforme cette expression en remplaçant :
$\arcsin(x)$ par $\frac{\pi}{2} - \arccos(x)$.
On tape :

```
asin2acos(acos(x)+asin(x))
```

On obtient après simplification :

```
pi/2
```

5.5.2 Transformer les arcsin en arctan : `asin2atan`

`asin2atan` a comme argument une expression trigonométrique.
`asin2atan` transforme cette expression en remplaçant :
$\arcsin(x)$ par $\arctan(\frac{x}{\sqrt{1-x^2}})$.
On tape :

```
asin2atan(asin(x))
```

On obtient :

```
atan(x/sqrt(1-x^2))
```

5.5.3 Transformer sin(x) en cos(x)*tan(x) : `sin2costan`

`sin2costan` a comme argument une expression trigonométrique.
`sin2costan` transforme cette expression en remplaçant :
$\sin(x)$ par $\cos(x) * \tan(x)$.
On tape :

```
sin2costan(sin(2*x))
```

On obtient :

```
cos(2*x)*tan(2*x)
```

5.6 Cosine

5.6.1 Transformer les arccos en arcsin : `acos2asin`

`acos2asin` a comme argument une expression trigonométrique.
`acos2asin` transforme cette expression en remplaçant :
$\arccos(x)$ par $\frac{\pi}{2} - \arcsin(x)$.
On tape :

```
acos2asin(acos(x)+asin(x))
```

On obtient après simplification :

```
pi/2
```

5.6.2 Transformer les arccos en arctan : `acos2atan`

`acos2atan` a comme argument une expression trigonométrique.
`acos2atan` transforme cette expression en remplaçant :
$\arccos(x)$ par $\frac{\pi}{2} - \arctan(\frac{x}{\sqrt{1-x^2}})$.
On tape :

```
acos2atan(acos(x))
```

On obtient :

```
pi/2-atan(x/sqrt(1-x^2))
```

5.6.3 Transformer cos(x) en sin(x)/tan(x) : `cos2sintan`

`cos2sintan` a comme argument une expression trigonométrique.
`cos2sintan` transforme cette expression en remplaçant :
$\cos(x)$ par $\frac{\sin(x)}{\tan(x)}$.
On tape :

```
cos2sintan(cos(2*x))
```

On obtient :

```
sin(2*x)/tan(2*x)
```

5.7 Tangent

5.7.1 Transformer les arctan en arcsin : `atan2asin`

`atan2asin` a comme argument une expression trigonométrique.
`atan2asin` transforme cette expression en remplaçant :
$\arctan(x)$ par $\arcsin(\frac{x}{\sqrt{1+x^2}})$.
On tape :

```
atan2asin(atan(x))
```

On obtient :

```
asin(x/sqrt(1+x^2))
```

5.7.2 Transformer les arctan en arccos : `atan2acos`

`atan2acos` a comme argument une expression trigonométrique.
`atan2acos` transforme cette expression en remplaçant :
$\arctan(x)$ par $\frac{\pi}{2} - \arccos(\frac{x}{\sqrt{1+x^2}})$.
On tape :

```
atan2acos(atan(x))
```

On obtient :

```
pi/2-acos(x/sqrt(1+x^2))
```

5.7.3 Transformer tan(x) en sin(x)/cos(x) : `tan2sincos`

`tan2sincos` a comme argument une expression trigonométrique.
`tan2sincos` transforme cette expression en remplaçant :
$\tan(x)$ par $\frac{\sin(x)}{\cos(x)}$.
On tape :

```
tan2sincos(tan(2*x))
```

On obtient :

```
sin(2*x)/cos(2*x)
```

5.7.4 Transformer une expression trigonométrique en fonction de tan(x/2) : `halftan`

`halftan` a comme argument une expression trigonométrique.
`halftan` transforme les $\sin(x)$, $\cos(x)$ et $\tan(x)$ contenus dans l'expression en fonction de $\tan(\frac{x}{2})$.
On tape :

```
halftan(sin(x))
```

On obtient :

```
2*tan(x/2)/(1+tan(x/2)^2)
```

On tape :

```
halftan(sin(2*x)/(1+cos(2*x)))
```

On obtient :

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

Et, après simplification avec `simplify(Ans)`, on obtient :

```
tan(x)
```

5.8 Trig

5.8.1 Simplifier en privilégiant les sinus : `trigsin`

`trigsin` a comme argument une expression trigonométrique.
`trigsin` simplifie cette expression à l'aide des formules :
$\sin(x)^2 + \cos(x)^2 = 1$, $\tan(x) = \dfrac{\sin(x)}{\cos(x)}$ et en privilégiant les sinus.
On tape :

```
trigsin(cos(x)^2+1)
```

On obtient :

```
-sin(x)^2+2
```

5.8.2 Simplifier en privilégiant les cosinus : `trigcos`

`trigcos` a comme argument une expression trigonométrique.
`trigcos` simplifie cette expression à l'aide des formules :
$\sin(x)^2 + \cos(x)^2 = 1$, $\tan(x) = \dfrac{\sin(x)}{\cos(x)}$ et en privilégiant les cosinus.
On tape :

```
trigcos(sin(x)^4+2)
```

On obtient :

```
cos(x)^4-2*cos(x)^2+3
```

5.8.3 Transformer avec des fonctions trigonométriques inverses en logarithmes : `atrig2ln`

`atrig2ln` réécrit l'expression contenant des fonctions trigonométriques inverses avec des logarithmes.
On tape :

```
atrig2ln(asin(x))
```

On obtient :

```
i*ln(x+sqrt(x^2-1))+pi/2
```

5.8.4 Simplifier en privilégiant les tangentes : `trigtan`

`trigtan` a comme argument une expression trigonométrique.
`trigtan` simplifie cette expression à l'aide des formules :
$\sin(x)^2 + \cos(x)^2 = 1$, $\tan(x) = \dfrac{\sin(x)}{\cos(x)}$ et en privilégiant les tangentes.
On tape :

```
trigtan(sin(x)^4+cos(x)^2+1)
```

On obtient :

```
((tan(x))^2/(1+(tan(x))^2))^2+1/(1+(tan(x)^2)+1
```

et après simplification avec `simplify(Ans)` on a :

```
(2*tan(x)^4+3*tan(x)^2+2)/(tan(x)^4+2*tan(x))^2+1)
```

5.8.5 Linéariser une expression trigonométrique : `tlin`

`tlin` a comme argument une expression trigonométrique.
`tlin` linéarise cette expression en fonction de $\sin(n.x)$ et $\cos(n.x)$.
Exemples

- Linéariser $\cos(x) * \cos(y)$.
  On tape :
  ```
  tlin(cos(x)*cos(y))
  ```
  On obtient :
  ```
  1/2*cos(x-y)+1/2*cos(x+y)
  ```
- Linéariser $\cos(x)^3$.
  On tape :
  ```
  tlin(cos(x)^3)
  ```
  On obtient :
  ```
  3/4*cos(x)+1/4*cos(3*x)
  ```
- Linéariser $4\cos(x)^2 - 2$.
  On tape :
  ```
  tlin(4*cos(x)^2-2)
  ```
  On obtient :
  ```
  2*cos(2*x)
  ```

5.8.6 Rassembler les sinus et cosinus de même angle : `tCollect`

`tCollect` a comme argument une expression trigonométrique.
`tCollect` linéarise cette expression en fonction de $\sin(n.x)$ et $\cos(n.x)$ puis rassemble les sinus et les cosinus de même angle.
On tape :

```
tcollect(sin(x)+cos(x))
```

On obtient :

```
sqrt(2)*cos(x-pi/4)
```

On tape :

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

On obtient :

```
sqrt(2)*cos(2*x-pi/4)
```

5.8.7 Développer une expression trigonométriques : `trigexpand`

`trigexpand` a comme argument une expression trigonométrique.
`trigexpand` développe cette expression en fonction de $\sin(x)$ et $\cos(x)$.
On tape :

```
trigexpand(cos(x+y))
```

On obtient :

```
cos(x)*cos(y)-sin(x)*sin(y)
```

subsectionTransformer une expression trigonométrique en des exponentielles complexes : `trig2exp` `trig2exp` a comme argument une expression trigonométrique.
`trig2exp` transforme les fonctions trigonométriques en exponentielles complexes SANS linéariser.
On tape :

```
trig2exp(tan(x))
```

On obtient :

```
((exp((i)*x))^2-1)/((i)*((exp((i)*x))^2+1))
```

On tape :

```
trig2exp(sin(x))
```

On obtient :

```
(exp((i)*x)-1/(exp((i)*x)))/(2*i)
```

Chapitre 6

Le menu Integer

6.1 Les diviseurs d'un nombre entier : `idivis`

`idivis` renvoie le vecteur de composantes les diviseurs d'un nombre entier.
On tape :

```
idivis(45)
```

On obtient :

```
[1,3,9,5,15,45]
```

6.2 Décomposition en facteurs premiers d'un entier : `ifactor`

`ifactor` renvoie la décomposition en facteurs premiers d'un nombre entier.
On tape :

```
ifactor(20!)
```

On obtient :

```
2^18*3^8*5^4*7^2*11*13*17*19
```

6.3 Liste des facteurs premiers et de leur multiplicité : `ifactors`

`ifactors` renvoie la liste des facteurs premiers d'un entier avec leur multiplicité.
On tape :

```
ifactors(45)
```

On obtient :

```
[3,2,5,1]
```

en effet $45 = 3^2 * 5^1$

6.4 PGCD de deux ou plusieurs entiers : `gcd`

`gcd` renvoie le PGCD de deux ou plusieurs entiers (voir 7.5 pour le PGCD de polynômes).
On tape :

```
gcd(45,10)
```

On obtient :

```
5
```

On tape :

```
gcd(40,12,16,24)
```

On obtient :

```
4
```

6.5 PPCM de deux ou plusieurs entiers : `lcm`

`lcm` renvoie le PPCM de deux ou plusieurs entiers.
On tape :

```
lcm(45,10)
```

On obtient :

```
90
```

On tape :

```
lcm(45,10,25,30)
```

On obtient :

```
450
```

6.6 Prime

6.6.1 Test pour savoir si un nombre est premier : `isPrime`

`isPrime(n)` renvoie `true` si `n` est premier et sinon renvoie `false`.
On tape :

```
isPrime(1234567)
```

On obtient :

```
false
```

On tape :

```
isPrime(1234547)
```

On obtient :

```
true
```

6.6.2 Le N-ième nombre premier : `ithprime`

`ithprime(n)` renvoie le `n`-ième nombre premier.
On tape :

```
ithprime(10)
```

On obtient :

```
29
```

En effet, les 10 premiers nombres premiers sont : $2, 3, 5, 7, 11, 13, 17, 19, 23, 29$.
On tape :

```
ithprime(100)
```

On obtient :

```
541
```

541 est donc le 100-ième nombre premier.

6.6.3 `nextprime`

`nextprime(n)` renvoie le nombre premier `p` qui se trouve juste après `n` (`p>n`).
On tape :

```
nextprime(11)
```

On obtient :

```
13
```

On tape :

```
nextprime(1234567)
```

On obtient :

```
1234577
```

6.6.4 `prevprime`

`prevprime(n)` renvoie le nombre premier `p` qui se trouve juste avant `n` (`p<n`).
On tape :

```
prevprime(11)
```

On obtient :

```
7
```

On tape :

```
prevprime(1234567)
```

On obtient :

```
1234547
```

6.6.5 Indicatrice d'Euler : `euler`

`euler(n)` renvoie le cardinal de l'ensemble des nombres inférieurs à `n` qui sont premiers avec `n`.
`euler(n)` désigne donc l'indicatrice d'Euler de l'entier `n`.
On tape :

```
euler(18)
```

On obtient :

```
6
```

En effet l'ensemble :
E={5,7,11,13,15,17} correspond aux nombres inférieurs à 18 qui sont premiers avec 18, et E a comme cardinal 6. Avec la fonction `euler`, on a la généralistion du petit théorème de Fermat (qui dit que "si n est premier et si a est premier avec n alors $a^{n-1} = 1 \mod n$)").
La généralisation est (puisque si n est premier, $euler(n) = n - 1$) :
$a^{euler(n)} = 1 \mod n$ si a et n sont premiers entre eux.
On tape :

```
powmod(5,6,18)
```

On obtient :

```
1
```

6.7 Division

6.7.1 Quotient de la division euclidienne : `iquo`

`iquo(a,b)` renvoie le quotient de la division euclidienne de `a` par `b` lorsque `a` et `b` sont des entiers.
On tape :

```
iquo(45,10)
```

On obtient :

```
4
```

en effet $45 = 4 * 10 + 5$

6.7.2 Reste de la division euclidienne : `irem`

`irem(a,b)` renvoie le reste de la division euclidienne de `a` par `b` lorsque `a` et `b` sont des entiers.
On tape :

```
irem(45,10)
```

On obtient :

```
5
```

en effet $45 = 4 * 10 + 5$

6.7.3 Restes chinois pour des entiers : `ichinrem`

`ichinrem([a,n],[b,p])` renvoie le vecteur `[c,lcm(p,q)]` formée de deux entiers.
Le premier nombre `c` est tel que

$$\forall k \in \mathbb{Z}, \quad d = c + k \times \text{lcm}(p, q)$$

l vérifie

$$d = a \pmod p, \quad d = b \pmod q$$

Si `n` et `p` sont premiers alors il y a toujours une solution et `q=n*p`
Exemple :
Trouver les solutions de :

$$\begin{cases} x = & 3 \pmod 5 \\ x = & 9 \pmod{13} \end{cases}$$

On tape :

```
ichinrem([3,5],[9,13])
```

Ou on tape :

```
ichinrem({[3,5],{9,13})
```

On obtient :

```
[-17,65]
```

Donc les solutions sont `x=-17+k*65)` avec `k`$\in \mathbb{Z}$.
On a en effet $-17 = -5 * 4 + 3 = 3 \bmod 5$ et $-17 = -2 * 13 + 9 = 9 \bmod 13$
`ichinrem` renvoie Restes chinois pour des entiers.
On tape :

```
ichinrem({2,7},{3,5})
```

Ou on tape :

```
ichinrem([2,7],[3,5])
```

On obtient :

```
[-12,35]
```

On a bien $-12 + 35k = 2 \bmod 7$ et $-12 + 35k = 3 \bmod 5$ pour $k \in \mathbb{Z}$

6.7.4 Calcul de $a^n \mod p$: `powmod`

`powmod(a,n,p)` calcule a^n modulo p avec la méthode de la puissance rapide).
On tape :

```
powmod(5,21,13)
```

On obtient :

```
5
```

$5^2 = 25 = -1 \bmod 13$ donc $5^{21} = 5 * 5^{20} = 5 \bmod 13$
On tape :
`powmod(37,25,11)`
On obtient :
`1`
en effet $37 = 4 \bmod 11$ et $4^5 = 4^2 * 4^2 * 4 = 25 * 4 = 1 \bmod 11$

Chapitre 7

Le menu Polynomial

7.1 Racines numériques d'un polynôme : `proot`

`proot` a comme argument un polynôme ou le vecteur de composantes les coefficients d'un polynôme (par ordre décroissant).
`proot` renvoie un vecteur dont les composantes sont les racines numériques du polynôme.
Pour chercher les racines numériques de $P(x) = x^3 + 1$, on tape :

```
proot([1,0,0,1])
```

ou on tape :

```
proot(x^3+1)
```

On obtient :

```
[-1,0.5+0.866025403784*i,0.5-0.866025403784*i]
```

On tape pour avoir les racines numériques de $x^2 - 3$:

```
proot([1,0,-3])
```

ou :

```
proot(x^2-3)
```

On obtient :

```
[1.73205080757,-1.73205080757]
```

Pour chercher les racines numériques de $P(x) = x^3 - 5 * x^2 + 8 * x - 4$, on tape :

```
proot([1,-5,8,-4])
```

ou on tape :

```
proot(x^3-5x^2+8x-4)
```

On obtient :

```
[1.,2.,2.]
```

7.2 Coefficients d'un polynôme : `coeff`

`coeff` a trois arguments : le polynôme, le nom de la variable (ou la liste des noms des variables) le degré (ou la liste des degrés des variables).
`coeff` renvoie le coefficient du polynôme de degré spécifié.
On tape :

```
coeff(x^3-5x^2+8x-4,2)
```

On obtient :

```
-5
```

On tape :

```
coeff(-x^4+3*x*y^2+x,y,2)
```

On obtient :

```
3*x
```

On tape :

```
coeff(-x^4+3*x*y^2+x,[x,y],[1,2])
```

On obtient :

```
3
```

7.3 Liste des diviseurs d'un polynôme : `divis`

`divis` a pour argument un polynôme symbolique (ou une liste de polynômes) et renvoie la liste des diviseurs.
On tape :

```
divis(x^2-1)
```

On obtient :

```
[1,x-1,x+1,(x-1)*(x+1)]
```

On tape :

```
divis(2t^2-2)
```

On obtient :

```
[1,2,t-1,2*(t-1),t+1,2*(t+1),(t-1)*(t+1),2*(t-1)*(t+1)]
```

On tape :

```
divis([t^2,x^2-1])
```

On obtient :

```
[[1,t,t^2],[1,x+1,x-1,(x-1)*(x+1)]]
```

7.4 Liste des facteurs d'un polynôme : factors

factors a pour argument un polynôme ou une liste de polynômes.
factors donne la liste des facteurs du polynôme avec leur multiplicité.
On tape :

```
factors(x^2+2*x+1)
```

On obtient :

```
[x+1,2]
```

On tape :

```
factors(x^4-2*x^2+1)
```

On obtient :

```
[x-1,2,x+1,2]
```

On tape :

```
factors([x^3-2*x^2+1,x^2-x])
```

On obtient :

```
[[x-1,1,x^2-x-1,1],[x,1,x-1,1]]
```

On tape :

```
factors([x^2,x^2-1])
```

On obtient :

```
[[x,2],[x+1,1,x-1,1]]
```

7.5 PGCD de polynômes par l'algorithme d'Euclide : gcd

gcd désigne le PGCD (plus grand commun diviseur) de deux polynômes pouvant avoir plusieurs variables et aussi le PGCD d'une liste de polynômes ou d'une séquence de polynômes pouvant avoir plusieurs variables (voir 6.4 pour le PGCD d'entiers).
On peut aussi mettre comme paramètres deux listes de même longueur (ou une matrice ayant 2 lignes), dans ce cas gcd renvoie le PGCD des éléments de même indice (ou d'une même colonne). On tape :

```
gcd(x^2+2*x+1,x^2-1)
```

On obtient :

```
x+1
```

On tape :

```
gcd([x^2-4,x*y-y],[x^3-8,y^2-x^2*y])
```

Ou on tape :

```
gcd([[x^2-4,x*y-y],[x^3-8,y^2-x^2*y]])
```

On obtient :

```
[x-2,y]
```

On tape :

```
gcd(x^2-2*x+1,x^3-1,x^2-1,x^2+x-2)
```

ou

```
gcd([x^2-2*x+1,x^3-1,x^2-1,x^2+x-2])
```

On obtient :

```
x-1
```

On tape :

```
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:=gcd(A,B)
```

On obtient :

```
z*x*y+z*x-z*y-z
```

On tape :

```
factor(A)
```

On obtient :

```
(y-1)*(y+1)*(x-1)*(x+1)*z^2
```

On tape :

```
factor(B)
```

On obtient :

```
(x^2+x+1)*(x-1)*(y+1)*(y^2-y+1)*z
```

On tape :

```
factor(C)
```

On obtient :

```
(y+1)*(x-1)*z
```

Pour les polynômes à coefficients modulaire , on tape, par exemple, car `%%` sert ici à désigner un nombre modulaire :

```
gcd((x^2+2*x+2) %% 5,(x^2-1) %% 5)
```

On obtient :

```
(1 %% 5)*x -1 %% 5
```

Mais si on tape :

```
gcd(x^2+2*x+2,x^2-1) %% 5
```

On obtient :

```
1 %% 5
```

car l'opération modulaire se fait après le calcul du PGCD qui a èté calculé dans $\mathbb{Z}[X]$.

7.6 PPCM de deux polynômes : `lcm`

`lcm` désigne le PPCM (plus petit commun multiple) de deux polynômes pouvant avoir plusieurs variables et aussi le PPCM d'une liste de polynômes ou d'une séquence de polynômes pouvant avoir plusieurs variables (voir **??** pour le PPCM d'entiers).
On tape :

```
lcm(x^2+2*x+1,x^2-1)
```

On obtient :

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

On tape :

```
lcm(x,x^2+2*x+1,x^2-1)
```

ou

```
lcm([x,x^2+2*x+1,x^2-1])
```

On obtient :

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

On tape :

```
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:=lcm(A,B)
```

On obtient :

```
(x*y*z-x*z+y*z-z)*(x^3*y^3*z+(-(y^3))*z+x^3*z-z)
```

On tape :

```
factor(A)
```

On obtient :

```
(y-1)*(y+1)*(x-1)*(x+1)*z^2
```

On tape :

```
factor(B)
```

On obtient :

```
(x^2+x+1)*(x-1)*(y+1)*(y^2-y+1)*z
```

On tape :

```
factor(D)
```

On obtient :

```
(x-1)*(x+1)*(x^2+x+1)*(y-1)*(y+1)*(y^2-y+1)*z^2
```

7.7 Create

7.7.1 Transformer un polynôme en une liste (format interne récursif dense) : `symb2poly`

`symb2poly` a comme argument polynôme, donné avec une écriture polynômiale, d'une variable (resp plusieurs variables), et le nom de cette variable formelle (par défaut `x`) (resp la séquence des noms de ces variables).
`symb2poly` transforme cette écriture polynômiale, en la liste des coefficients selon les puissances décroissantes selon le nom de la variable donné en deuxième argument (resp l'écriture récursive de la liste des coefficients selon les puissances décroissantes selon les noms des variables donnés en deuxième argument : le résultat est la liste des coefficients de la première variable, coefficients qui sont eux-mêmes des polynômes qui seront donnés sous la forme la liste des coefficients de la deuxième variable etc...).
Attention Si le deuxième argument est une liste, le résultat est l'écriture du polynôme au format interne.
On tape :

```
symb2poly(x^2-1)
```

Ou on tape :

```
symb2poly(x^2-1,x)
```

Ou on tape :

```
symb2poly(y^2-1,y)
```

On obtient :

```
[1,0,-1]
```

On tape :

```
symb2poly(x*y^2+2y-1,x)
```

On obtient :

```
[y^2,2y-1]
```

On tape :

```
symb2poly(x*y^2+2y-1,y)
```

On obtient :

```
[x,2,-1]
```

7.7.2 Transformer le format interne creux distribué du polynôme en une écriture polynômiale : `poly2symb`

`poly2symb` a comme argument la liste des coefficients par puissances décroissantes d'un polynôme et un nom de variable formelle (par défaut `x`) (resp le format interne creux distribué du polynôme c'est à dire la somme de monômes tels que : `%%%{c,[px,py,pz] %%%}` et une liste de variables formelles tel que `[x,y,z]` ce qui représente le monôme $cx^{px}y^{py}z^{pz}$).
`poly2symb` transforme la liste des coefficients par puissances décroissantes d'un polynôme (resp la somme de `%%%{c,[px,py,pz] %%%}`), en son écriture polynômiale (selon Horner), en utilisant le nom de la variable donné en deuxième argument (resp en utilisant la liste de variables donné en deuxième argument `[x,y,z]`).
On tape :

```
poly2symb([1,0,-1])
```

Ou on tape :

```
poly2symb([1,0,-1],x)
```

On obtient :

```
x*x-1
```

On tape :

```
poly2symb([1,0,-1],y)
```

On obtient :

```
y*y-1
```

7.7.3 Coefficients d'un polynôme défini par ses racines : `pcoeff pcoef`

`pcoeff` (ou `pcoef`) a comme argument, une liste de composantes les racines d'un polynôme P.
`pcoeff` (ou `pcoef`) renvoie une liste de composantes, les coefficients du polynôme unitaire P (par ordre décroissant).
On tape :

```
pcoef([1,2,0,0,3])
```

On obtient :

```
[1,-6,11,-6,0,0]
```

c'est à dire $(x-1)(x-2)(x^2)(x-3) = x^5 - 6x^4 + 11x^3 - 6x^2$.

7.7.4 Coefficients d'une fraction rationnelle définie par ses racines et ses pôles : `fcoeff`

`fcoeff` a comme argument un vecteur de composantes les racines et les pôles d'une fraction rationnelle $F(x)$ suivis de leur multiplicité.
`fcoeff` renvoie la fraction rationnelle $F(x)$.
On tape :

```
fcoeff([1,2,0,3,2,-1])
```

On obtient :

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

7.7.5 Coefficient du terme de plus haut degré d'un polynôme : `lcoeff`

`lcoeff` a comme argument un polynôme donné sous forme symbolique ou par la liste de ses coefficients.
`lcoeff` renvoie le coefficient de plus haut degré de ce polynôme (`lcoeff`=leading coefficient).
On tape :

```
lcoeff([2,1,-1,0])
```

On obtient :

```
2
```

On tape :

```
lcoeff(3*x^2+5*x,x)
```

On obtient :

```
3
```

On tape :

```
lcoeff(3*x^2+5*x*y^2,y)
```

On obtient :

```
5*x
```

subsectionPolynôme minimal : `pmin` `pmin` a un (resp deux) argument(s).
`pmin` a comme argument une matrice A d'ordre n (resp une matrice A d'ordre n et un nom de variable formelle).
`pmin` renvoie le polynôme minimal de A écrit selon la liste de ses coefficients (resp le polynôme minimal P de A écrit sous forme symbolique en utilisant le nom de variable donnée en argument).
Le polynôme minimal P de A est le polynôme de plus petit degré qui annule A ($P(A) = 0$).
On tape :

```
pmin([[1,0],[0,1]])
```

On obtient :

```
[1,-1]
```

Ou on tape :

```
pmin([[1,0],[0,1]],x)
```

On obtient :

```
x-1
```

Donc le polynôme minimal de [[1,0],[0,1]] est $x - 1$.
On tape :

```
pmin([[1,1,0],[0,1,1],[0,0,1]])
```

On obtient :

```
[1,-3,3,-1]
```

On tape :

```
pmin([[1,1,0],[0,1,1],[0,0,1]],x)
```

On obtient :

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

Donc le polynôme minimal de [[1,1,0],[0,1,1],[0,0,1]] est $x^3 - 3 * x^2 + 3 * x - 1$.
On tape :

```
pmin([[2,1,0],[0,2,0],[0,0,2]])
```

On obtient :

```
[1,-4,4]
```

On tape :

```
pmin([[2,1,0],[0,2,0],[0,0,2]],x)
```

On obtient :

```
x^2-4*x+4
```

Donc le polynôme minimal de [[2,1,0],[0,2,0],[0,0,2]] est $x^2 - 4x + 4$.

7.7.6 Polynômes aléatoires : `randPoly`

`randPoly`) a comme paramètre un entier `n`.
`randPoly` renvoie les coefficients d'un polynôme de degré `n` et dont les coefficients sont des entiers aléatoires équirépartis sur -99..+99.
On tape :

```
randPoly(4)
```

On obtient par exemple :

```
[4,53,-45,80,-99)
```

7.8 Algebra

7.8.1 Quotient euclidien de 2 polynômes : `quo`

`quo` donne le quotient de la division euclidienne de deux polynômes (division selon les puissances décroissantes).
On peut donner les polynômes soit par la liste de leurs coefficients selon les puissances décroissantes, soit sous leurs formes symboliques et dans ce cas la variable doit être rajoutée comme troisième argument (par défaut la variable est `x`).
On tape :

```
quo(x^2+2x+1,x+3)
```

On obtient :

```
x-1
```

On tape :

```
quo(t^2+2t+1,t+3,t)
```

On obtient :

```
t-1
```

ou on tape :

```
quo([1,2,1],[1,3])
```

On obtient :

```
[ 1,-1]
```

c'est à dire le polynôme `poly1[1,-1]`.
Pour avoir le quotient de $x^3 + 2x + 4$ par $x^2 + x + 2$, on tape :

```
quo(x^3+2x+4,x^2+x+2)
```

On obtient :

```
x-1
```

Ou on tape :

```
quo([1,0,2,4],[1,1,2])
```

On obtient :

```
[] 1,-1 []
```

c'est à dire le polynôme `poly1[1,-1]` ou encore le polynôme `x-1`.
On tape :

```
quo(t^3+2t+4,t^2+t+2,t)
```

On obtient :

```
t-1
```

On tape si on ne met pas la variable t comme dernier argument :

```
quo(t^3+2t+4,t^2+t+2)
```

On obtient :

```
(t^3+2*t+4)/(t^2+t+2)
```

7.8.2 Reste euclidien de 2 polynômes : `rem`

`rem` donne le reste de la division euclidienne de deux polynômes (division selon les puissances décroissantes).
On peut donner les polynômes soit par la liste de leurs coefficients selon les puissances décroissantes, soit sous leurs formes symboliques et dans ce cas la variable doit être rajoutée comme troisième argument (par défaut la variable est `x`).
On tape :

```
rem(x^3-1,x^2-1)
```

On obtient :

```
x-1
```

On tape :

```
rem(t^3-1,t^2-1,t)
```

On obtient :

```
t-1
```

On tape :

```
rem(x^2+2x+1,x+3)
```

Ou on tape :

```
rem(t^2+2t+1,t+3,t)
```

On obtient :

```
4
```

ou on tape :

```
rem([1,2,1],[1,3])
```

On obtient :

```
[] 4
```

c'est à dire le polynôme `poly1[4]` ou encore le polynôme `4`.
On tape pour avoir le reste de $x^3 + 2x + 4$ par $x^2 + x + 2$:

```
rem(x^3+2x+4,x^2+x+2)
```

On obtient :

```
x+6
```

Ou on tape :

```
rem([1,0,2,4],[1,1,2])
```

On obtient :

```
[1,6]
```

c'est à dire le polynôme `poly1[1,6]` ou encore le polynôme `x+6`.
On tape :

```
rem(t^3+2t+4,t^2+t+2,t)
```

On obtient :

```
t+6
```

On tape si on ne met pas la variable t comme dernier argument :

```
rem(t^3+2t+4,t^2+t+2)
```

On obtient :

```
0
```

7.8.3 Degré d'un polynôme : `degree`

`degree` a comme argument un polynôme donné sous forme symbolique ou par la liste de ses coefficients.
`degree` renvoie le degré de ce polynôme (degré du monôme de plus grand degré).
On tape :

```
degree(x^3+x)
```

On obtient :

```
3
```

On tape :

```
degree([1,0,1,0])
```

On obtient :

```
3
```

7.8.4 Mise en facteur de x^n dans un polynôme : `factor_xn`

`factor_xn` a comme argument un polynôme `P`.
`factor_xn` renvoie le polynôme `P` dans lequel on a mis en facteur x^n où n est le degré de `P` (`n=degree(P)`).
On tape :

```
factor_xn(-x^4+3)
```

On obtient :

```
x^4*(-1+3*x^-4)
```

7.8.5 PGCD des coefficients d'un polynôme : `content`

`content` a comme arguments un polynôme `P` donné sous forme symbolique ou par la liste de ses coefficients et le nom de la variable (par défaut c'est x).
`content` désigne le PGCD (plus grand commun diviseur) des coefficients du polynôme `P`.
On tape :

```
content(6*x^2-3*x+9)
```

ou on tape :

```
content(6*t^2-3*t+9,t)
```

ou :

```
content([6,-3,9]))
```

On obtient :

```
3
```

7.8.6 Nombre de changements de signe sur $]a;\ b]$: `sturmab`

`sturmab` a quatre paramètres : une expression polynômiale P, le nom de la variable et deux nombres a et b.

- si a et b sont réels, `sturmab` renvoie soit un nombre strictement positif qui est le nombre de changements de signe de P sur $]a;\ b]$, soit 0 si P reste de signe constant positif ou nul sur $]a;\ b]$, soit -1 si P reste de signe constant négatif ou nul sur $]a;\ b]$. Ainsi, `sturmab` permet d'avoir le nombre de racines sur $[a, b[$ du polynôme P/G avec $G = \text{gcd}(P, \text{diff}(P))$.
- si a ou b est complexe, le nombre de racines complexes à l'intérieur du rectangle de sommets opposés a et b.

On tape :

```
sturmab(x^2*(x^3+2),x,-2,0)
```

On obtient :

```
1
```

On tape :

```
sturmab(x^3-1,x,-2-i,5+3i)
```

On obtient :

```
3
```

On tape :

```
sturmab(x^3-1,x,-i,5+3i)
```

On obtient :

```
1
```

Attention ! ! ! !
P doit être donné par son expression symbolique et, si on tape :
`sturmab([1,0,0,2,0,0],x,-2,0)`,
on obtient :
`Bad argument type.`

7.8.7 Les restes chinois : `chinrem`

`chinrem` a comme argument deux listes ayant chacun comme composantes deux polynômes éventuellement donnés par la liste de leurs coefficients par ordre décroissant.
`chinrem` renvoie une liste de composantes deux polynômes.
`chinrem([A,R],[B,Q])` renvoie la liste des polynômes `P` et `S` vérifiant :

$$S = R.Q, \quad P = A \pmod{R}, P = B \pmod{Q}$$

Il existe toujours une solution `P` si `R` et `Q` sont premiers entre eux, et toutes les solutions sont congrues modulo `S=R*Q`
Trouver les solutions $P(x)$ de :

$$\begin{cases} P(x) = & x & \mod (x^2+1) \\ P(x) = & x-1 & \mod (x^2-1) \end{cases}$$

On tape :

```
chinrem([[1,0],[1,0,1]],[[1,-1],[1,0,-1]])
```

On obtient :

```
[[1/-2,1,1/-2],[1,0,0,0,-1]]
```

ou on tape :

```
chinrem([x,x^2+1],[x-1,x^2-1])
```

On obtient :

```
[-1/2*x^2+x-1/2,x^4-1]
```

donc $P(x) = -\dfrac{x^2-2.x+1}{2} \pmod{x^4-1}$
Autre exemple :
On tape :

```
chinrem([[1,2],[1,0,1]],[[1,1],[1,1,1]])
```

On obtient :

```
[[-1,-1,0,1],[1,1,2,1,1]]
```

ou on tape :

```
chinrem([x+2,x^2+1],[x+1,x^2+x+1])
```

On obtient :

```
[-x^3-x^2+1,x^4+x^3+2*x^2+x+1]
```

7.9 Special

7.9.1 Polynôme cyclotomique : `cyclotomic`

`cyclotomic` a comme paramètre un entier n.
`cyclotomic` renvoie la liste des coefficients du polynôme cyclotomique d'ordre n. C'est le polynôme dont les zéros sont toutes les racines n-ième et primitives de l'unité (une racine n-ième de l'unité est primitive si ses puissances engendrent toutes les autres racines n-ième de l'unité).
Par exemple pour $n = 4$, les racines quatrième de l'unité sont : $\{1, i, -1, -i\}$, et les racines primitives sont : $\{i, -i\}$.
Donc le polynôme cyclotomique d'ordre 4 est $(x - i).(x + i) = x^2 + 1$.
On tape :

```
cyclotomic(4)
```

On obtient :

```
[1,0,1]
```

On tape :

```
cyclotomic(5)
```

On obtient :

```
[1,1,1,1,1]
```

Donc le polynôme cyclotomique d'ordre 5 est $x^4 + x^3 + x^2 + x + 1$ et on a $(x - 1) * (x^4 + x^3 + x^2 + x + 1) = x^5 - 1$.
On tape :

```
cyclotomic(10)
```

On obtient :

```
[1,-1,1,-1,1]
```

Donc le polynôme cyclotomique d'ordre 10 est $x^4 - x^3 + x^2 - x + 1$ et on a

$$(x^5 - 1) * (x + 1) * (x^4 - x^3 + x^2 - x + 1) = x^{10} - 1$$

On tape :

```
cyclotomic(20)
```

On obtient :

```
[1,0,-1,0,1,0,-1,0,1]
```

Donc le polynôme cyclotomique d'ordre 20 est $x^8 - x^6 + x^4 - x^2 + 1$ et on a

$$(x^{10} - 1) * (x^2 + 1) * (x^8 - x^6 + x^4 - x^2 + 1) = x^{20} - 1$$

7.9.2 Base de Gröbner : `gbasis`

`gbasis` a au moins deux arguments : une liste de polynômes de plusieurs variables et la liste du nom de ces variables.
`gbasis` renvoie une base de Gröbner de l'idéal polynomial engendré par les polynômes donnés dans le premier argument.
On choisit d'ordonner les monômes selon l'ordre lexicographique en accord avec la liste donnée par le dernier argument et selon les puissances décroissantes : par exemple on écrira $x^2 * y^4 * z^3$ puis $x^2 * y^3 * z^4$ si le deuxième argument est $[x, y, z]$ car $(2, 4, 3) > (2, 3, 4)$ mais on écrira $x^2 * y^3 z^4$ puis $x^2 * y^4 * z^3$ si le deuxième argument est $[x, z, y]$.
Si I est un idéal et si $(G_k)_{k \in K}$ est une base de Gröbner de l'idéal I alors, si F est un polynôme non nul de I, le terme dominant de F est divisible par le terme dominant d'un G_k.
Propriété : Si on fait la division euclidienne de F par un des G_k puis, si on recommence avec le reste obtenu et le G_k suivant, on finit par obtenir un reste nul.
On tape :

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

On obtient :

```
[y^3,x*y+(-1/2)*y^2,x^2-y^2]
```

7.9.3 Réduction par rapport à une base de Gröbner : `greduce`

`greduce` a trois arguments : un polynôme de plusieurs variables, une liste de polynômes formant une base de Gröbner dépendant des mêmes variables et la liste du nom de ces variables.
`greduce` renvoie la réduction (à une constante multiplicative près) du polynôme donné dans le premier argument par rapport à la base de Gröbner donnée dans le deuxième argument.
On tape :

```
greduce(x*y-1,[x^2-y^2,2*x*y-y^2,y^3],[x,y])
```

On obtient :

```
1/2*y^2-1
```

ce qui veut dire que $xy - 1 = \frac{1}{2}(y^2 - 2) \mod I$ où I est l'idéal engendré par la base de Gröbner $[x^2 - y^2, 2xy - y^2, y^3]$, puisque $y^2 - 2$ est le reste de la division euclidienne de $2(xy - 1)$ par $G_2 = 2xy - y^2$.
Remarque
La constante multiplicative peut être déterminée en regardant comment le coefficient constant est transformé. Dans l'exemple, le terme constant `-1` est transformé en le terme constant `-2`, donc le coefficient multiplicatif est `1/2`.

7.9.4 Polynôme de Hermite : `hermite`

`hermite` a comme argument un entier n et éventuellement le nom de la variable (x par défaut).
`hermite` renvoie le polynôme de Hermite de degré n.
Le polynôme de Hermite de degré n noté $P(n,x)$ vérifie les relations :
$P(0,x) = 1$
$P(1,x) = 2x$
$P(n,x) = 2xP(n-1,x) - 2(n-1)P(n-2,x)$
Ces polynômes sont orthogonaux pour le produit scalaire :
$< f,g >= \int_{-\infty}^{+\infty} f(x)g(x)e^{-x^2}dx$
On tape :

```
hermite(6)
```

On obtient :

```
64*x^6+-480*x^4+720*x^2-120
```

On tape :

```
hermite(6,y)
```

On obtient :

```
64*y^6+-480*y^4+720*y^2-120
```

7.9.5 Interpolation de Lagrange : `lagrange`

`lagrange` a comme argument deux listes de longueur `n` ou une matrice de deux lignes et `n` colonnes et eventuellement le nom de la variable `var` (par défaut `x`) : la première liste (ou ligne) correspond à des valeurs d'abscisses x_k, et la deuxième liste (ou ligne) correspond à des valeurs d'ordonnées y_k pour k allant de 1 à `n`.
`lagrange` renvoie une expression polynômiale `P(var)` de degré `n-1` tel que $P(x_k) = y_k$.
On tape :

```
lagrange([[1,3],[0,1]])
```

Ou on tape :

```
lagrange([1,3],[0,1])
```

On obtient :

```
1/2*(x-1)
```

en effet pour $x = 1$ on a $\frac{x-1}{2} = 0$ et pour $x = 3$ on a $\frac{x-1}{2} = 1$.
On tape :

```
lagrange([1,3],[0,1],y)
```

On obtient :

```
1/2*(y-1)
```

Attention `lagrange([1,2],[3,4],y)` ne renvoie pas une fonction mais une expression. mais on peut définir une fonction en mettant :
`f(x):=lagrange([1,2],[3,4],x)` ou
`f(y):=lagrange([1,2],[3,4],y)` et alors
`f(4)` renvoie `6` car `f(x)=x+2`)
Bien voir la différence entre :
`g(x):=lagrange([1,2],[3,4])` et
`f(x):=lagrange([1,2],[3,4],x)`.
`g(x):=lagrange([1,2],[3,4])` ne definit pas une fonction, par exemple, `g(2)=x-1+3` alors que `f(2)=4`.
Ceci dit, la définition of `f` n'est pas efficace car le polynôme sera recalculé depuis le début à chaque appel de `f` (quand on définit une fonction le membre de droite n'est pas évalué, l'évaluation est faite seulement quand on appelle `f`).
Pour être efficace il faut utiliser `unapply` :
`f:=unapply(lagrange([1,2],[3,4]),x)` ou
`f:=unapply(lagrange([1,2],[3,4],y),y)` **Exercice** Soient $f(x) = \frac{1}{x}$, $x_0 = 2$ $x_1 = 2.5$ et $x_2 = 4$. On demande de calculer le polynôme L d'interpolation de Lagrange et sa valeur en $x = 3$ et $x = 4.5$.
On tape :

```
f(x):=1/x
L:=unapply(normal(lagrange([2,2.5,4],[f(2),f(2.5),f(4)])),x)
```

On obtient :

```
x->0.05*x^2-0.425*x+1.15
```

On tape :

```
L(3),L(4.5)
```

On obtient :

```
0.325,0.25
```

7.9.6 Polynôme de Laguerre : `laguerre`

`laguerre` a comme argument un entier n et eventuellement le nom de la variable (x par défaut) et du paramètre (a par défaut).
`laguerre` renvoie le polynôme de Laguerre de degré n et de paramètre a.
Le polynôme de Laguerre de degré n de paramètre a noté $L(n, a, x)$ vérifie les relations :
$L(0, a, x) = 1$
$L(1, a, x) = 1 + a - x$
$L(n, a, x) = \frac{2n+a-1-x}{n} L(n-1, a, x) - \frac{n+a-1}{n} L(n-2, a, x)$
Ces polynômes sont orthogonaux pour le produit scalaire :
$< f, g >= \int_0^{+\infty} f(x)g(x)x^a e^{-x} dx$
On tape :

```
laguerre(2)
```

On obtient :

```
1/2*a^2-a*x+3/2*a+1/2*x^2-2*x+1
```

On tape :

```
laguerre(2,y)
```

On obtient :

```
1/2*a^2-a*y+3/2*a+1/2*y^2-2*y+1)
```

On tape il faut que `b` soit purgé (`b:='b'`) :

```
laguerre(2,y,b)
```

On obtient :

```
1/2*b^2-b*y+3/2*b+1/2*y^2-2*y+1
```

7.9.7 Polynôme de Legendre : `legendre`

`legendre` a comme argument un entier n et eventuellement le nom de la variable (x par défaut).
`legendre` renvoie le polynôme de Legendre de degré n : c'est le polynôme non nul, solution de l'équation différentielle :

$$(x^2-1).y''-2.x.y'-n(n+1).y=0$$

Le polynôme de Legendre de degré n noté $P(n,x)$ vérifie les relations :
$P(0,x)=1$
$P(1,x)=x$
$P(n,x)=\frac{2n-1}{n}xP(n-1,x)-\frac{n-1}{n}P(n-2,x)$
Ces polynômes sont orthogonaux pour le produit scalaire :
$<f,g>=\int_{-1}^{+1}f(x)g(x)dx$
On tape :

```
legendre(4)
```

On obtient :

```
35/8*x^4+-15/4*x^2+3/8
```

On tape :

```
legendre(4,y)
```

On obtient :

```
35/8*y^4+-15/4*y^2+3/8
```

7.9.8 Polynôme de Tchebychev de 1-ière espèce : `tchebyshev1`

`tchebyshev1` a comme argument un entier n et eventuellement le nom de la variable (x par défaut).
`tchebyshev1` renvoie le polynôme de Tchebychev de première espèce, de degré n, noté $T(n,x)$.
On a :

$$T(n,x)=\cos(n.\arccos(x))$$

$T(n,x)$ vérifie les relations :
$T(0,x)=1$
$T(1,x)=x$
$T(n,x)=2xT(n-1,x)-T(n-2,x)$
Les polynômes $T(n,x)$ sont orthogonaux pour le produit scalaire :
$<f,g>=\int_{-1}^{+1}\frac{f(x)g(x)}{\sqrt{1-x^2}}dx$
On tape :

```
tchebyshev1(4)
```

On obtient :

```
8*x^4-8*x^2+1
```

On tape :

```
tchebyshev1(4,y)
```

On obtient :

```
8*y^4-8*y^2+1
```

et on a bien :
$\cos(4.x)=Re((\cos(x)+i.\sin(x))^4)$
$\cos(4.x)=\cos(x)^4-6.\cos(x)^2.(1-\cos(x)^2)+(1-\cos(x)^2)^2.$
$\cos(4.x)=T(4,\cos(x)).$

7.9.9 Polynôme de Tchebychev de 2-nde espèce : `tchebyshev2`

`tchebyshev2` a comme argument un entier n et eventuellement le nom de la variable (x par défaut).
`tchebyshev2` renvoie le polynôme de Tchebychev de seconde espèce, de degré n, noté $U(n,x)$.
On a :

$$U(n,x)=\frac{\sin((n+1).\arccos(x))}{\sin(\arccos(x))}$$

ou encore

$$\sin((n+1)x)=\sin(x)*U(n,\cos(x))$$

$U(n,x)$ vérifie les relations :
$U(0,x)=1$
$U(1,x)=2x$
$U(n,x)=2xU(n-1,x)-U(n-2,x)$
Les polynômes $U(n,x)$ sont orthogonaux pour le produit scalaire :
$<f,g>=\int_{-1}^{+1}f(x)g(x)\sqrt{1-x^2}dx$
On tape :

```
tchebyshev2(3)
```

On obtient :

```
8*x^3+-4*x
```

On tape :

```
tchebyshev2(3,y)
```

On obtient :

```
8*y^3+-4*y
```

en effet :
$\sin(4.x) = \sin(x) * (8 * \cos(x)^3 - 4. \cos(x)) = \sin(x) * U(3, \cos(x))$.

Chapitre 8

Le menu Plot

Remarque
Le tracé de la plupart des commandes commencant par `plot` ne se fait pas bien depuis l'écran CAS : il est donc préfierable d'utiliser l'application de géométrie pour faire les graphes correspondant à ces commandes.

8.1 Graphe d'une fonction : `plotfunc`

`plotfunc(f(x),x)` trace la représentation graphique de $y = f(x)$ et `plotfunc(f(x),x=a..b)` trace la représentation graphique de $y = f(x)$ lorsque $a \leq x \leq b$.
On tape :

```
plotfunc(x^2-2)
```

ou

```
plotfunc(a^2-2,a=-1..2)
```

On obtient :

```
la représentation graphique de y=x^2-2
```

Ou bien
On tape :

```
gf:=plotfunc(x^2-2)
```

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$\sqrt{}$`GC:=`

On complète en

$\sqrt{}$`GC:=`gf

Puison appuie sur la touche `Plot` pour avoir la vue graphique de l'application de géométrie, on obtient :

```
le graphe de x^2-2
```

8.2 Courbe implicite en 2-d : `plotimplicit`

`plotimplicit` permet de tracer des courbes définies de façon implicite par une expression. Pour que `Xcas` ne cherche pas à factoriser l'expression, la commande `plotimplicit` peut être utilisée avec l'option `unfactored` ou `sans_factoriser` mise comme dernier paramètre, :

- avec `unfactored` l'expression ne sera pas modifiée,
- sans `unfactored` `Xcas` réduit l'expression au même dénominateur puis cherche à factoriser le numérateur.
- `plotimplicit(f(x,y),x,y)` ou `plotimplicit(f(x,y),[x,y])` trace la représentation graphique de la courbe définie implicitement par $f(x,y) = 0$ lorsque x (resp y) varie selon `WX-, WX+` (resp `WY-, WY+`) défini dans `cfg`,
- `plotimplicit(f(x,y),x=0..1,y=-1..1)` ou `plotimplicit(f(x,y),[x=0..1,y=-1..1])` trace la représentation graphique de la courbe définie implicitement par $f(x,y) = 0$ lorsque $0 \leq x \leq 1$ et $-1 \leq y \leq 1$ (mettre des bornes un peu plus grandes pour ne pas avoir de manques !).

On tape :

```
plotimplicit(x^2+y^2-1,[x,y])
```

Ou on tape :

```
plotimplicit(x^2+y^2-1,{x,y},unfactored)
```

Ou on tape :

```
plotimplicit(x^2+y^2-1,x,y,unfactored)
```

On obtient :

```
circle(point(0,0),1)
```

On tape :

```
g:=plotimplicit(x^2+y^2-1,[x,y])
```

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$\sqrt{}$`GC:=`

On complète en

$\sqrt{}$`GC:=g`

Puis dans la vue graphique de l'application de géométrie (`Plot`), on obtient :

```
le dessin du cercle unité
```

8.3 Graphe d'une fonction par niveaux de couleurs : `plotdensity`

`plotdensity(f(x,y),[x,y])` trace le graphe de $z = f(x, y)$ dans le plan en représentant z par une des couleurs de l'arc en ciel.
On tape :

```
plotdensity(x^2-y^2,[x=-2..2,y=-2..2],xstep=0.1,ystep=0.1)
```

On obtient :

```
Un graphique en 2-d représentant pour chaque z,
l'hyperbole définie par x^2-y^2=z par une couleur de
l'arc en ciel
```

On remarquera que l'on a l'echelle des couleurs en dessous du graphe.

8.4 Le champ des tangentes : `plotfield`

On peut tracer le champ des tangentes de léquation différentielle $y' = f(t, y)$ ou du système déquations différentielles $x' = u(x, y), y' = v(x, y)$ et on peut spécifier les plages de valeurs des paramètres.

- Soit $f(t, y)$ une expression dependant de deux variables t et y, alors `plotfield(f(t,y),[t,y])` trace le champ des tangentes de l'équation différentielle $y' = f(t, y)$ où y représente une variable réelle et t est représenté en abscisse,
- Soit $V = [u(x, y), v(x, y)]$ est un vecteur 2-d de coordonnées deux expressions dépendant de 2 variables x, y mais indépendant du temps, alors `plotfield(V,[x,y])` trace le champ des tangentes du système $[x'(t) = u(x, y), y'(t) = v(x, y)]$,
- Les plages de valeurs de t, y ou de x, y peuvent être spécifiées par `t=tmin..tmax`, `x=xmin..xmax`, `y=ymin..ymax` à la place du nom de variable seul.
- On peut spécifier le cadrage en mettant par exemple : `plotfield(f(t,y),[t=tmin..tmax,y=ymin..ymax])`
- On peut spécifier que le champ des tangentes soit, dans un repère orthonormé, de norme 1 avec l'option `normalize`. Sans l'option `normalize` le point de contact est l'origine du vecteur tangent et avec l'option `normalize` le point de contact se trouve au milieu des tangentes.
- On peut aussi spécifier la valeur des pas en t et en y avec `xstep=...` et `ystep=....`

On tape :

```
plotfield(4*sin(t*y),[t=0..2,y=-3..7])
```

On obtient :

```
Des segments de pente 4*sin(t*y) sont tracés en
différents points. Ces segments représentent les
vecteurs tangents dirigés selon les t croissants et
dont l'origine est le point de contact
```

On tape :

```
plotfield(4*sin(t*y),[t=0..2,y=-3..7],normalize,
xstep=0.7,ystep=0.7))
```

On obtient :

Des segments de longueur 1 et de pente 4*sin(t*y) qui représentent les tangentes au point situé en leur milieu. Ces points espacés de 0.7

On tape :

```
plotfield(5*[-y,x],[x=-1..1,y=-1..1])
```

On obtient :

Des vecteurs $[-y, x]$ sont tracés aux points (x, y). Ces vecteurs représentent des vecteurs tangents en leur origine aux courbes solutions du système $x(t)' = -y, y(t)' = x$. Ils sont dirigés selon les t croissants.

On tape :

```
plotfield(5*[-y,x],[x=-1..1,y=-1..1],normalize)
```

On obtient :

Des segments de longueur 1 et de pente $-y/x$ qui représentent les tangentes au point situé en leur milieu aux courbes solutions du système $x(t)' = -y, y(t)' = x$.

8.5 Lignes de niveaux : `plotcontour`

`plotcontour(f(x,y),[x,y])` trace les lignes de niveaux $z = -10$, $z = -8$,.., $z = 0$, $z = 2$,.., $z = 10$ de la surface définie par $z = f(x, y)$.
On tape :

```
g:=plotcontour(x^2+y^2,[x=-3..3,y=-3..3],[1,2,3],
display=[vert,rouge,noir]+[filled$3])
```

On obtient :

```
[polygon(point(....))..]
```

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$\sqrt{}$`GC:=`

On complète en

$\sqrt{}$`GC:=g`

Puis dans la vue graphique de l'application de géométrie (`Plot`), on obtient :

```
le graphe des trois ellipses x^2-y^2=n pour n=1,2,3;
les zones comprises entre ces ellipses sont remplies
        avec la couleur verte,rouge ou noire
```

On tape :

```
plotcontour(x^2-y^2,[x,y])
```

On obtient :

```
[polygon(point(-4.8,-5),point(-3.9,-4)....)]
```

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$\surd$`GC:=`

On complète en

$\surd$`GC:=g`

Puis dans la vue graphique de l'application de géométrie (`Plot`), on obtient :

```
le graphe des 11 hyperboles x^2-y^2=n pour
n=-10,-8,..10
```

8.6 Tracé de solutions d'équation différentielle : `plotode`

On peut tracer les solutions de léquation différentielle $y' = f(t, y)$ ou du système déquations différentielles $x' = u(t, x, y), y' = v(t, x, y)$ et on peut spécifier les plages de valeurs des paramètres.

- `plotode(f(t,y),[t,y],[t0,y0])` trace en fonction du temps la solution $y(t)$ de l'équation différentielle $y' = f(t, y)$ passant par le point `(t0,y0)`, où $f(t, y)$ désigne une expression dépendant de la variable de temps t et de la variable y.
- Par défaut, t varie dans les 2 directions. On peut spécifier la plage du temps par le paramètre optionnel `t=tmin..tmax`.
- Lorsque $y = (X, Y)$ est un vecteur de longueur 2 et f à valeurs dans $\mathbb{R}^2$, on peut également représenter dans l'espace (t, X, Y) ou dans le plan (X, Y) la solution d'une équation différentielle $y' = f(t, y)$ c'est à dire $[X', Y'] = [f(t, X, Y)]$. Pour cela, il suffit de remplacer y par le noms des variables X, Y et la valeur initiale par les deux valeurs initiales des variables au temps t_0.

On tape :

```
plotode(sin(t*y),[t,y],[0,1])
```

On obtient :

Le graphe de la solution de y'=sin(t,y) passant par le point (0,1) est tracé

Pour visualiser les valeurs de la solution, se reporter à la section 4.6.
On tape :

```
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)
```

On obtient :

Le graphe de la solution de x'=-y,y'=x pour t=0 passant par le point (0.3,0.7)

8.7 Ligne polygonale : `plotlist`

`plotlist` a pour argument une liste `l` ou une matrice ayant deux colonnes. `plotlist` permet de visualiser les segments reliant le nuage de points ayant pour abscisse `[0,1,2...n]` et pour ordonnée `l` ou pour coordonnées une ligne de la matrice. `listplot` ou `plotlist` relie par des segments de droites, les différents points du nuage, mais sans réordonner les points contrairement à `polygonplot` qui réordonne les points selon leur abscisse puis les relie.
On tape :

```
a:=plotlist([0,1,4,9,16])
```

Ou on tape :

```
a:=plotlist([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$\sqrt{}$`GD:=`

On complète en

$\sqrt{}$`GD:=a`

Puis dans la vue graphique de l'application de géométrie (`Plot`), on obtient :

```
le dessin des 5 points ((0,0),(1,1),...(4,16)) reliés
par 4 segments
```

Attention
`plotlist([0,1,2,3,4],[0,1,4,9,16])` ou
`plotlist([[0,1,2,3,4],[0,1,4,9,16]])` n'est pas valide !

Troisième partie

Le menu MATH de la touche

Chapitre 9

Les fonctions sur les réels

9.1 Le plus petit entier >= à l'argument : `CEILING`

`CEILING(a)` renvoie le plus petit entier supérieur ou égal à l'argument a.
On tape :

```
CEILING(45/8)
```

On obtient :

```
6
```

On tape :

```
CEILING(-45/8)
```

On obtient :

```
-5
```

On tape :

```
CEILING(2.5)
```

On obtient :

```
3
```

9.2 Partie entière d'un réel : `FLOOR`

`FLOOR(a)` renvoie le plus grand entier inférieur ou égal à l'argument a.
On tape :

```
FLOOR(45/8)
```

On obtient :

```
5
```

On tape :

```
FLOOR(-45/8)
```

On obtient :

```
-6
```

On tape :

```
FLOOR(2.5)
```

On obtient :

```
2
```

9.3 Argument sans sa partie fractionnaire : `IP`

`IP(a)` renvoie l'argument réel a sans sa partie fractionnaire.
On tape :

```
IP(45/8)
```

On obtient :

```
5
```

On tape :

```
IP(-45/8)
```

On obtient :

```
-5
```

9.4 Partie fractionnaire : `FP`

`FP(a)` renvoie la partie fractionnaire de l'argument réel a.
On tape dans :

```
FP(45/8)
```

On obtient :

```
0.625
```

On tape dans CAS :

```
FP(45/8)
```

On obtient :

```
5/8
```

On tape dans :

```
FP(-45/8)
```

On obtient :

```
-0.625
```

On tape dans CAS :

```
FP(-45/8)
```

On obtient :

```
-5/8
```

9.5 Arrondir avec n décimales un réel ou un complexe : ROUND

`ROUND(a)` (resp `ROUND(a,n)`) arrondit le réel a selon un entier (resp selon un nombre décimal ayant n décimales) le plus proche.
`ROUND(a)` (resp `ROUND(a,n)`) arrondit le complexe a à selon un élément de $\mathbb{Z}[i]$ le plus proche, (resp avec n décimales). On tape :

```
ROUND(45/8)
```

On obtient :

```
6
```

On tape :

```
ROUND(45/8,2)
```

On obtient :

```
5.63
```

On tape :

```
ROUND(-45/8)
```

On obtient :

```
-6
```

On tape :

```
ROUND(-45/8,2)
```

On obtient :

```
-5.62
```

On tape :

```
ROUND(0.5+i*pi)
```

On obtient :

```
1+3*i
```

On tape :

```
ROUND(0.5+i*pi,4)
```

On obtient :

```
0.5+3.1416*i
```

9.6 Tronquer avec n décimales un réel ou un complexe : TRUNCATE

TRUNCATE renvoie l'argument tronqué avec n décimales (par défaut n=0) . On tape :

```
TRUNCATE(45/8)
```

On obtient :

```
5
```

On tape :

```
TRUNCATE(45/8,2)
```

On obtient :

```
5.62
```

On tape :

```
TRUNCATE(-45/8)
```

On obtient :

```
-5
```

On tape :

```
TRUNCATE(-45/8,2)
```

On obtient :

```
-5.63
```

On tape :

```
TRUNCATE(sqrt(2)+i*sqrt(5),4)
```

On obtient :

```
1.4142+2.236*i
```

9.7 Mantisse d'un réel : MANT

MANT(a) renvoie $|a|/10^n$ où l'entier n vérifie :
$10^n \leq |a| < 10^{n+1}$.
MANT(a) renvoie donc la mantisse d'un réel a c'est à dire les chiffres significatifs de a.
On tape :

```
MANT(45/8)
```

On obtient :

```
5.625
```

On tape :

```
MANT(-45/8)
```

On obtient :

```
5.625
```

9.8 Partie entière du logarithme à base 10 d'un réel : `XPON`

`XPON(a)` renvoie l'entier n tel que $10^n \leq |a| < 10^{n+1}$.
On tape :

```
XPON(45/8)
```

On obtient :

```
0
```

On tape :

```
XPON(45000/8)
```

On obtient :

```
3
```

On tape :

```
XPON(1234*sqrt(2))
```

On obtient :

```
3
```

en effet $10^3 < 1234 * \sqrt{2} \simeq 1745.13953597 < 10^4$

Chapitre 10

Arithmétique

10.1 Maximum de 2 ou plusieurs valeurs :`MAX`

`MAX` renvoie le maximum des éléments d'une séquence ou d'une liste de réels.
On tape :

```
MAX(4,5,8,2,6)
```

On obtient :

```
8
```

10.2 Minimum de 2 ou plusieurs valeurs : `MIN`

`MIN` renvoie le minimum des éléments d'une séquence ou d'une liste de réels.
On tape :

```
MIN(4,5,8,2,6)
```

On obtient :

```
2
```

10.3 `MOD`

`MOD` est une fonction infixée.
`a MOD b` renvoie le reste de la division euclidienne de a par b.
On tape :

```
22 MOD 5
```

On obtient :

```
2
```

10.4 FNROOT

FNROOT renvoe une racine approchée de l'expression donnèe en premier argument pour la variable donnée en second argument et qui soit proche du 3ième argument.
On tape en mode réel (i non coché dans le Shift-CAS (Settings)):

```
FNROOT(x^4+3x-4)
```

On obtient :

```
-1.74295920217,1.
```

On tape en mode réel (i non coché dans le Shift-CAS (Settings)):

```
FNROOT(x^4+3x-4,x,-2)
```

On obtient :

```
-1.74295920217
```

On tape en mode complexe (i coché dans le Shift-CAS (Settings)):

```
FNROOT(x^4+3x-4)
```

On obtient :

```
[-1.74295920217,0.371479601083+1.46865601291*i,
      0.371479601083-1.46865601291*i,1.0]
```

10.5 %

%(a,b) renvoie $\frac{a}{100} * b$ (a pour cent de b).
On tape :

```
%(5,70)
```

On obtient :

```
3.5
```

On tape :

```
%(5,90)
```

On obtient :

```
4.5
```

10.6 Complexe

10.6.1 CONJ

CONJ On tape :

```
CONJ(1+3*i)
```

On obtient :

```
1-3*i
```

10.6.2 IM

On tape :

```
IM(1+3*i)
```

On obtient :

```
3
```

10.6.3 RE

On tape :

```
RE(1+3*i)
```

On obtient :

```
1
```

10.6.4 SIGN

On tape :

```
SIGN(1+3*i)
```

On obtient :

```
(1+3*i)/sqrt(10)
```

10.6.5 La touche Shift-+/− : ABS

ABS renvoie :
- la valeur absolue d'un réel,
- le module d'un nombre complexe,
- la longueur d'un vecteur v_j ($(\sum_{j=1}^{n} |v_j|^2)^{1/2}$),
- la norme de Schur ou de Frobenius d'une matrice $a_{j,k}$ ($(\sum_{j,k=1}^{n} |a_{j,k}|^2)^{1/2}$).

On tape :

```
ABS(1+3*i)
```

On obtient :

```
sqrt(10)
```

10.7 Exponentielle

10.7.1 ALOG alog10

alog10 désigne la fonction anti-logarithme à base dix, c'est la fonction :
$x- > 10^x$.
On tape dans CAS :

```
ALOG(3/2)
```

On obtient :

```
sqrt(10)*10^1
```

On tape :

```
alog10(10)
```

On obtient :

```
10000000000
```

Remarque Si on sélectionne `ALOG` depuis , on a :
`ALOG(3/2)` renvoie `31.6227766017` qui est la valeur approchée de `sqrt(10)*10` à 10^{-10} près.

10.7.2 EXPM1

`EXPM1` désigne la fonction $x->EXP(x)-1$.
On tape :

```
EXPM1(4)
```

On obtient :

```
EXP(4)-1
```

On tape :

```
EXPM1(2.*10^-4)
```

On obtient :

```
0.00020002000133
```

MAIS si on tape :

```
EXP(2.*10^-4)-1
```

On obtient :

```
0.00020002
```

10.7.3 LNP1

`LNP1` désigne la fonction $x->LN(x+1)$
On tape :

```
LNP1(4)
```

On obtient :

```
LN(5)
```

On tape :

```
LNP1(2.*10^-4)
```

On obtient :

```
1.99980002666E-4
```

Chapitre 11

Fonctions trigonométriques

11.1 CSC

`CSC(x)` renvoie `1/SIN(x)` : c'est la fonction cosécante.
On tape :

```
CSC(pi/3)
```

On obtient après simplification :

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

11.2 ACSC

`ACSC(x)` renvoie `ASIN(1/x)` : c'est la fonction réciproque de la fonction cosécante.
On tape :

```
ACSC(sqrt(2))
```

On obtient après simplification :

```
pi/4
```

11.3 SEC

`SEC(x)` renvoie `1/COS(x)` : c'est la fonction sécante.
On tape :

```
SEC(pi/3)
```

On obtient :

```
2
```

11.4 ASEC

`ASEC(x)` renvoie `ACOS(1/x)` : c'est la fonction réciproque de la fonction sécante.
On tape :

```
ASEC(2)
```

On obtient :

```
1/3*pi
```

11.5 COT

`COT(x)` renvoie `COS(x)/SIN(x)` : c'est la fonction cotangente.
On tape :

```
COT(pi/3)
```

On obtient après simplification :

```
sqrt(3)/3
```

11.6 ACOT

`ACOT(x)` renvoie `pi/2-ATAN(x)` : c'est la fonction réciproque de la fonction cotangente.
On tape :

```
ACOT(sqrt(3))
```

On obtient après simplification :

```
pi/6
```

Chapitre 12

Fonctions hyperboliques

12.1 SINH

`SINH(x)` renvoie : $\dfrac{\exp(x) - \exp(-x)}{2}$: c'est la fonction sinus hyperbolique,
On tape :

```
hyp2exp(SINH(ln(2)))
```

On obtient :

```
3/4
```

12.2 ASINH

`ASINH` est la fonction réciproque de la fonction sinus hyperbolique,
On tape :

```
ASINH(3/4)
```

On obtient :

```
ln(2)
```

12.3 COSH

`COSH(x)` renvoie : $\dfrac{\exp(x) + \exp(-x)}{2}$: c'est la fonction cosinus hyperbolique,
On tape :

```
COSH(0)
```

On obtient :

```
1
```

12.4 ACOSH

ACOSH est la fonction réciproque de la fonction cosinus hyperbolique,
On tape :

```
ACOSH(1)
```

On obtient :

```
0
```

12.5 TANH

TANH(x) renvoie : $\dfrac{\exp(2x)-1}{\exp(2x)+1}$: c'est la fonction tangente hyperbolique,
On tape :

```
TANH(ln(3))
```

On obtient :

```
4/5
```

12.6 ATANH

ATANH est la fonction réciproque de la fonction tangente hyperbolique,
On tape :

```
ATANH(4/5)
```

On obtient :

```
ln(9)/2
```

Chapitre 13

Fonctions de probabilité

13.1 Factorielle : !

`!` est une fonction post fixée.
`n!` renvoie la factorielle de n si n est entier et
`a!` renvoie valeur de la fonction Gamma en a+1 si a est réel.
On tape :

```
20!
```

On obtient :

```
2432902008176640000
```

On tape :

```
5.2!
```

On obtient :

```
169.406099462
```

13.2 Nombre de combinaisons de p objets pris parmi n : COMB

`COMB(n,p)` renvoie le nombre de combinaisons de p éléments pris parmi n (n et p sont des entiers).
On a : `COMB(n,p)` renvoie $\dfrac{n!}{p!(n-p)!}$.
On tape :

```
COMB(5,2)
```

On obtient :

```
10
```

13.3 nombre d'arrangements de p objets pris parmi n : `PERM`

`PERM(n,p)` renvoie le nombre d'arrangements de p éléments pris parmi n (n et p sont des entiers).
On a : `PERM(n,p)` renvoie $\dfrac{n!}{(n-p)!}$.
On tape :

```
PERM(5,2)
```

On obtient :

```
20
```

13.4 Nombres aléatoires

13.4.1 Nombre aléatoire (réel ou entier) : `RANDOM`

- Pour avoir un nombre aléatoire réel entre 0 et 1, on ne met pas d'argument.
  On tape :
  ```
  RANDOM
  ```
  On obtient un nombre réel de 0,1, par exemple :
  ```
  0.291424166081
  ```
- Pour avoir un nombre aléatoire entier a entre 1 et n ($1 \leq a \leq n$) on met n comme argument sans mettre de parenthèse.
  On tape :
  ```
  RANDOM 3
  ```
  On obtient 1,2 ou 3, par exemple :
  ```
  1
  ```
- Pour avoir un nombre aléatoire réel a entre b et c ($b \leq a \leq c$), on met b, c comme arguments sans mettre de parenthèse.
  On tape :
  ```
  RANDOM 3,5
  ```
  On obtient un nombre réel de 3,5 par exemple :
  ```
  4.81731227506
  ```
- Pour avoir k nombres aléatoires entiers a entre p et n ($p \leq a \leq n$) on met k, p, n comme arguments sans mettre de parenthèse.
  On tape :
  ```
  RANDOM 3,2,5
  ```
  On obtient 3 nombres entiers compris entre 2 et 5, par exemple :
  ```
  [5,3,2]
  ```

13.4.2 Nombre entier aléatoire `RANDINT`

- Pour avoir un nombre aléatoire entier a entre 1 et n ($1 \leq a \leq n$) on met n comme argument.
  On tape :
  ```
  RANDINT(4)
  ```
  On obtient 0,1,2,3 ou 4, par exemple :

```
2
```

– Pour avoir un nombre aléatoire entier a entre b et c ($b \leq a \leq c$), on met b et c comme arguments.
  On tape :
  ```
  RANDINT(4,6)
  ```
  On obtient 4,5 ou 6, par exemple :
  ```
  5
  ```
– Pour avoir k nombres aléatoires entiers a entre p et n ($p \leq a \leq n$) on met k, p et n comme arguments. On tape :
  ```
  RANDINT(4,2,6)
  ```
  On obtient 4 nombres compris entre 2 et 6, par exemple :
  ```
  [2,6,2,6]
  ```

13.4.3 RANDNORM

`RANDNORM(mu,sigma)` renvoie un réel aléatoirement distribué selon la loi normale $N(mu, sigma)$ (par défaut $mu = 0$ et $sigma = 1$).
On tape :

```
RANDNORM()
```

Ou on tape :

```
RANDNORM(0,1)
```

On obtient, par exemple :

```
1.2440525851
```

On tape :

```
RANDNORM(1,2)
```

On obtient, par exemple :

```
-1.8799815939
```

13.4.4 Pour initialiser la suite de nombres aléatoires : RANDSEED

`RANDSEED` initialiser la suite des nombres aléatoires fournie par `RANDOM`. Si on ne met pas de paramètre, `RANDSEED` utilise la valeur du temps comme paramètre.
On tape :

```
RANDSEED()
```

On obtient :

```
1
```

On tape :

```
RANDSEED(pi)
```

On obtient :

```
1
```

13.5 Densité de probabilité

13.5.1 Densité de probabilité de la loi normale : `NORMALD`

`NORMALD`
On tape :

```
NORMALD(0.5)
```

Ou on tape :

```
NORMALD(0,1,0.5)
```

On obtient :

```
0.352065326764
```

On tape :

```
NORMALD(1,2,0.5)
```

On obtient :

```
0.193334058401
```

13.5.2 Densité de probabilité de la loi de Student : `STUDENT`

`STUDENT`
On tape :

```
STUDENT(3,5.2)
```

On obtient :

```
0.00366574413491
```

13.5.3 Densité de probabilité du χ^2 : `CHISQUARE`

`CHISQUARE`
On tape :

```
CHISQUARE(2,3.2)
```

On obtient :

```
0.100948258997
```

13.5.4 Densité de probabilité de la loi de Fisher : `FISHER`

`FISHER(n,m,x)` renvoie la densité de probabilité en x de la loi de Fisher-Snedecor (n et m sont les nombres de degrés de liberté).
On tape :

```
FISHER(4,10,2.1)
```

On obtient :

```
0.141167840452
```

13.5.5 Densité de probabilité de la loi binomiale : `BINOMIAL`

`BINOMIAL(n,k,p)` renvoie `COMB`$(n,k) * p^k * (1-p)^(n-k)$ et
`BINOMIAL(n,k)` renvoie `COMB`(n,k) si il n'y a pas de 3ème argument.
On tape :

```
BINOMIAL(4,2)
```

On obtient :

```
6
```

On tape :

```
BINOMIAL(4,2,0.5)
```

On obtient :

```
0.375
```

13.5.6 Densité de probabilité de la loi de Poisson : `POISSON`

`POISSON(mu,k)` renvoie $\frac{\exp(-mu) * mu^k}{k!}$.
On tape :

```
POISSON(0.5,2)
```

On obtient :

```
0.0758163324641
```

13.6 Fonction de répartition

13.6.1 Fonction de répartition de la loi normale : `NORMALD_CDF`

Lorsqu'une variable aléatoire X suit une loi normale centrée réduite, on a :
$Proba(X \leq x)$= `NORMALD_CDF(x)` et
$Proba(x \leq X \leq y)$= `NORMALD_CDF(x,y)`.
Lorsqu'une variable aléatoire X suit une loi normale de moyenne μ et d'écart-type σ, on a :
$Proba(X \leq x)$= `NORMALD_CDF(`μ,σ`,x)`.
$Proba(x \leq X \leq y)$= `NORMALD_CDF(`μ,σ`,x,y)`.
On tape :

```
NORMALD_CDF(0.96)
```

Ou on tape :

```
NORMALD_CDF(0,1,0.96)
```

On obtient :

```
0.831472392533
```

On tape :

```
NORMALD_CDF(1.96)
```

On obtient :

```
0.975002104852
```

On tape :

```
NORMALD_CDF(0,1.96)
```

On obtient :

```
0.475002104852
```

car `NORMALD_CDF(0)=1/2` et 0.975002104852-0.5=0.475002104852
On tape :

```
NORMALD_CDF(1,2,1.96)
```

On obtient :

```
0.684386303484
```

On tape :

```
NORMALD_CDFnormal_cdf(1,2,1.1,2.9)
```

On obtient :

```
0.309005067853
```

13.6.2 Fonction de répartition de la loi de Student : `STUDENT_CDF`

Lorsqu'une variable aléatoire X suit une loi de Student ayant n degrés de liberté, on a :
$Proba(X \leq x)$= `STUDENT_CDF(n,x)`.
$Proba(x \leq X \leq y)$= `STUDENT_CDF(n,x,y)`.
On tape :

```
STUDENT_CDF(5,2)
```

On obtient :

```
0.949030260585
```

On tape :

```
STUDENT_CDF(5,-2)
```

On obtient :

```
0.0509697394149
```

13.6.3 Fonction de répartition de la loi du χ^2 `CHISQUARE_CDF`

Lorsqu'une variable aléatoire X suit une loi du χ^2 ayant n degrés de liberté, on a :
$Proba(X \leq x)$= `CHISQUARE_CDF(n,x)`.
$Proba(x \leq X \leq y)$= `CHISQUARE_CDF(n,x,y)`.
On tape :

```
CHISQUARE_CDF(5,11)
```

On obtient :

```
0.948620016517
```

On tape :

```
CHISQUARE_CDF(5,3)
```

On obtient :

```
0.300014164121
```

On tape :

```
CHISQUARE_CDF(5,3,11)
```

On obtient :

```
0.648605852396
```

car 0.948620016517-0.300014164121=0.648605852396

13.6.4 La fonction de répartition de la loi de Fisher-Snédécor : `FISHER_CDF`

Lorsqu'une variable aléatoire X suit une loi de Fisher-Snédécor ayant degrés de liberté $n1, n2$, on a :
$Proba(X \leq x)$= `FISHER_CDF(n1,n2,x)`.
$Proba(x \leq X \leq y)$= `FISHER_CDF(n1,n2,x,y)`.
On tape :

```
FISHER_CDF(5,3,9)
```

On obtient :

```
0.949898927032
```

On tape :

```
FISHER_CDF(3,5,9.)
```

On obtient :

```
0.981472898262
```

On tape :

```
FISHER_CDF(3,5,2.)
```

On obtient :

```
0.767376082
```

On tape :

```
FISHER_CDF(3,5,2.,9.)
```

On obtient :

```
0.214096816262
```

car 0.981472898262-0.767376082=0.214096816262

13.6.5 Fonction de répartition de la loi binomiale : `BINOMIAL_CDF`

Lorsqu'une variable aléatoire X suit une loi binomiale $\mathcal{B}(n,p)$, on a :
`BINOMIAL_CDF(n,p,x)`=$Proba(X \leq x)$=
`BINOMIAL(n,0,p)+...+BINOMIAL(n,floor(x),p)`.
`BINOMIAL_CD(n,p,x,y)`=$Proba(x \leq X \leq y$=
`BINOMIAL(n,ceil(x),p)+...+BINOMIAL(n,floor(y),p)`.
On tape :

```
BINOMIAL_CDF(4,0.5,2))
```

On obtient :

```
0.6875
```

On peut vérifier que :

```
BINOMIAL(4,0,0.5)+BINOMIAL(4,1,0.5)+BINOMIAL(4,2,0.5)

=0.6875
```

On tape :

```
BINOMIAL_CDF(2,0.3,1)
```

On obtient :

```
0.91
```

On tape :

```
BINOMIAL_CDF(2,0.3,1,2)
```

On obtient :

```
0.51
```

13.6.6 Fonction de répartition de Poisson `POISSON_CDF`

Lorsqu'une variable aléatoire X suit une loi de Poisson de paramètre `mu`, c'est à dire de moyenne `mu` et d'écart-type `mu`, on a :
$Proba(X \leq x)$=`POISSON_CDF( mu,x)` avec $X \in \mathcal{P}(mu)$. et
$Proba(x \leq X \leq y)$=`POISSON_CDF(mu,x,y)`
`POISSON_CDF(mu,x)` est la fonction de répartition de la loi de Poisson de paramètre `mu`, On tape :

```
POISSON_CDF(10.0,3)
```

On obtient :

```
0.0103360506759
```

13.7 Fonction de répartition inverse

13.7.1 Fonction de répartition inverse normale : `NORMALD_ICDF`

Lorsqu'une variable aléatoire X suit une loi normale centrée réduite, si on a `NORMALD_ICDF(x)=h` c'est que l'on a :
$Proba(X \leq h)$=`x=NORMALD_CDF(h)`.
Lorsqu'une variable aléatoire X suit une loi normale de moyenne μ et d'écart-type σ, si on a `NORMALD_ICDF(`μ, σ`,x)=h` c'est que l'on a :
$Proba(X \leq h)$=`x=NORMALD_CDF(`μ, σ`,h)`.
On tape :

```
NORMALD_ICDF(0.95)
```

Ou on tape :

```
NORMALD_ICDF(0,1,0.95)
```

On obtient :

```
1.64485362695
```

On tape :

```
NORMALD_ICDF(0.975)
```

On obtient :

```
1.95996398454
```

On tape :

```
NORMALD_ICDF(1,2,0.495)
```

On obtient :

```
0.974933060984
```

On tape :

```
NORMALD_ICDF(1,2,NORMALD_CDF(1,2,0.975))
```

On obtient :

```
0.975
```

On tape :

```
NORMALD_CDF(1,2,NORMALD_ICDF(1,2,0.495))
```

On obtient :

```
0.495
```

On tape :

```
NORMALD_ICDF(1,2,2.96*sqrt(2))
```

On obtient :

```
0.944423950497
```

13.7.2 Fonction de répartition inverse de Student : `STUDENT_ICDF`

Lorsqu'une variable aléatoire X suit une loi de Student ayant n degrés de liberté, si on a `STUDENT_ICDF(n,x)=h` c'est que :
$Proba(X \leq h)$=`x=STUDENT_CDF(n,h)`.
On tape :

```
STUDENT_ICDF(5,0.95)
```

On obtient :

```
2.01504837333
```

13.7.3 Fonction inverse de la fonction de répartition de la loi du χ^2`CHISQUARE_ICDF`

Lorsqu'une variable aléatoire X suit une loi du χ^2 ayant n degrés de liberté, si on a `CHISQUARE_ICD(n,x)=h` c'est que :
$Proba(X \leq h)$=`x= CHISQUARE_CDF(n,h)`.
On tape :

```
CHISQUARE_ICDF(5,0.95)
```

On obtient :

```
11.0704976935
```

13.7.4 Inverse de la fonction de répartition de la loi de Fisher-Snédécor : `FISHER_ICDF`

Lorsqu'une variable aléatoire X suit une loi de Fisher-Snédécor ayant comme degrés de liberté $n1, n2$, si on a :
`FISHER_ICDF(n1,n2,x)=h` c'est que :
$Proba(X \leq h)$=`x=FISHER_CDF(n1,n2,h)`.
On tape :

```
FISHER_ICDF(5,3,0.95)
```

On obtient :

```
9.01345516752
```

On tape :

```
1/FISHER_ICDF(3,5,0.05)
```

On obtient :

```
9.01345516752
```

Remarque :
`FISHER_ICDF(n1,n2,p)=1/FISHER_ICDF(n2,n1,1-p)`

13.7.5 Fonction de répartition inverse de la loi binomiale : `BINOMIAL_ICDF`

Lorsqu'une variable aléatoire X suit une loi binomiale $\mathcal{B}(n,p)$, si on a :
`BINOMIAL_ICDF(n,p,x)=h` c'est que
$Proba(X \leq h) = x$=`BINOMIAL_ICDF(n,p,h)`.
On tape :

```
BINOMIAL_ICDF(4,0.5,0.9)
```

On obtient :

```
3
```

On tape :

```
BINOMIAL_ICDF(2,0.3,0.95)
```

On obtient :

```
2
```

13.7.6 Fonction de répartition inverse de Poisson :`POISSON_ICDF`

Lorsqu'une variable aléatoire X suit une loi de Poisson de paramètre `mu`, c'est à dire de moyenne `mu` et d'écart-type `mu`, on a :
`POISSON_ICDF(mu,t)`= h équivaut à
$Proba(X \leq h)$ =`t= POISSON_CDF(mu,h)` avec $X \in \mathcal{P}(mu)$.
`POISSON_ICDF(mu,t)` est l'inverse de la fonction de répartition de la loi de Poisson de paramètre `mu`, On tape :

```
POISSON_ICDF(10.0,0.975)
```

On obtient :

```
0.125110035721
```

Chapitre 14

Les listes

14.1 MAKELIST

MAKELIST crée une liste à partir d'une expression symbolique.
Par exemple, on crèe une liste à partir de X^2+1, en faisant varier la variable X de 2 à 6 avec un pas de 1 (1 peut être omis), on tape :

```
MAKELIST(X^2+1,X,2,6)
```

On obtient :

```
{5,10,17,26,37}
```

On tape :

```
MAKELIST(0,X,1,10)
```

On obtient :

```
{0,0,0,0,0,0,0,0,0,0}
```

On crèe une liste à partir de X^2+1, en faisant varier la variable X de 2 à 6 avec un pas de 2, on tape :

```
MAKELIST(X^2+1,X,2,6,2)
```

On obtient :

```
{5,17,37}
```

14.2 SORT

SORT trie dans l'ordre croissant les composantes d'une liste. On tape :

```
SORT([12,2,31,24,15])
```

On obtient :

```
[2,12,15,24,31]
```

On tape :

```
SORT({12,2,31,24,15})
```

On obtient :

```
{2,12,15,24,31}
```

14.3 REVERSE

REVERSE crée une liste en inversant l'ordre des éléments.
On tape :

```
REVERSE([1,22,3,4,5])
```

On obtient :

```
[5,4,3,22,1]
```

On tape :

```
REVERSE({1,22,3,4,5})
```

On obtient :

```
{5,4,3,22,1}
```

14.4 CONCAT

CONCAT concaténe deux listes ou deux vecteurs ou deux chaines de caractères.
On tape :

```
CONCAT([1,2,3],[4,5])
```

On obtient :

```
[1,2,3,4,5]
```

On tape :

```
CONCAT({1,2,3},{4,5})
```

On obtient :

```
{1,2,3,4,5}
```

On tape :

```
CONCAT("BON","JOUR")
```

On obtient :

```
"BONJOUR"
```

On tape :

```
2=>A
```

```
CONCAT([1,A,3],[4,5])
```

On obtient :

```
[1,2,3,4,5]
```

On tape :

```
2=>A
CONCAT({1,A,3},{4,5})
```

On obtient :

```
{1,2,3,4,5}
```

Pour concaténer une chaine et une liste en une liste on utilise CONCAT.
On tape :

```
2=>A
CONCAT([1,A,3]),"L1"
```

On obtient :

```
[1,2,3,"L1"]
```

On tape :

```
2=>A
CONCAT("L1",[1,A,3])
```

On obtient :

```
["L1",1,2,3]
```

Attention
Pour concaténer une chaine et une liste en une chaine on utilise +.
On tape :

```
2=>A
"L1="+[1,A,3]
```

On obtient :

```
"L1=[1,2,3]"
```

On tape :

```
2=>A
[1,A,3]+"L1="
```

On obtient :

```
"[1,2,3]L1="
```

On tape :

```
2=>A
"L1="+{1,A,3}
```

On obtient :

```
"L1={1,2,3}"
```

14.5 POS

POS renvoie la position d'un élément dans une liste c'est à dire POS renvoie l'indice de la première occurence de l'élément ou 0 si l'élément n'est pas dans la liste.
On tape :

```
POS([4,3,1,2,3,4,5],4)
```

Ou on tape :

```
POS({4,3,1,2,3,4,5},4)
```

On obtient :

```
1
```

On tape :

```
POS([4,3,1,2,3,4,5],2)
```

Ou on tape :

```
POS({4,3,1,2,3,4,5},2)
```

On obtient :

```
4
```

On tape :

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

Ou on tape :

```
POS({4,3,1,2,3,4,5},6)
```

On obtient :

```
0
```

14.6 SIZE

SIZE renvoie la longueur de la liste donnée en argument.
On tape :

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

On obtient :

```
3
```

On tape :

```
SIZE({1,2,3})
```

On obtient :

```
3
```

14.7 ΔLIST

ΔLIST renvoie la liste des différences entre les composantes de la liste donnée en argument.
On tape :

```
ΔLIST([1,21,34,41,52])
```

On obtient :

```
[20,13,7,11]
```

On tape :

```
ΔLIST({1,21,34,41,52})
```

On obtient :

```
{20,13,7,11}
```

14.8 ΣLIST

ΣLIST renvoie la somme des composantes de la liste donnée en argument.
On tape :

```
ΣLIST([1,2,3,4,5])
```

On obtient :

```
15
```

On tape :

```
ΣLIST({1,2,3,4,5})
```

On obtient :

```
15
```

14.9 ΠLIST

ΠLIST renvoie le produit des composantes de la liste donnée en argument.
On tape :

```
ΠLIST([1,2,3,4,5])
```

On obtient :

```
120
```

On tape :

```
ΠLIST({1,2,3,4,5})
```

On obtient :

```
120
```

Chapitre 15

Les chaînes de caractères

15.1 `asc`

`asc` renvoie la liste des codes ASCII des caractères de la chaine.
On tape :

```
asc("A")
```

On obtient :

```
[65]
```

On tape :

```
asc("ABC")
```

On obtient :

```
[65,66,67]
```

15.2 `char`

`char` renvoie la chaine correspondant aux caractères ayant comme code ASCII ceux de l'argument. On tape :

```
char(65)
```

ou

```
char([65])
```

ou

```
char({65})
```

On obtient :

```
"A"
```

On tape :

```
char([65,66,67])
```

ou

```
char({65,66,67})
```

On obtient :

```
"ABC"
```

15.3 Pour utiliser une chaîne comme un nombre ou une commande : `expr`

15.3.1 Pour utiliser une chaîne comme un nombre

`expr` permet d'utiliser une chaîne de chiffres ne commençant pas par un zéro comme un nombre entier écrit en base 10 ou une chaîne de chiffres comportant un point comme un nombre décimal écrit en base 10.
`expr` renvoie ce nombre entier On tape :

```
expr("123")+1
```

On obtient :

```
124
```

On tape :

```
expr("45.67")+2.12
```

On obtient :

```
47.79
```

`expr` permet aussi d'utiliser une chaîne de chiffres ne comportant pas de `8`, ni de `9`, et commençant par un zéro comme un nombre entier écrit en base `8`.
On tape :

```
expr("0123")
```

On obtient :

```
83
```

En effet $1 * 8^2 + 2 * 8 + 3 = 83$
Remarque :
Si on tape `expr("018")` on obtient le nombre décimale `18.0`.
`expr` permet d'utiliser une chaîne contenant des chiffres et les lettres `a,b,c,d,e,f`, et commençant par `0x` comme un nombre entier écrit en base 16.
On tape :

```
expr("0x12f")
```

On obtient :

```
303
```

En effet $1 * 16^2 + 2 * 16 + 15 = 303$

15.3.2 Pour utiliser une chaîne comme nom de commande

`expr` permet d'utiliser une chaîne de caractères comme une commande.
`expr` est surtout utile dans un programme.
`expr` a comme argument une chaîne de caractères pouvant être interprétée comme une commande (ou le nom d'une variable qui contient une chaîne ou une expression renvoyant une chaîne).
`expr` transforme la chaîne en une expression, puis évalue cette expression :
pour faire une affectation, on ne doit pas écrire `expr("a"):=2`, mais on doit écrire `expr("a:=2")` (voir aussi `expr` **??**)
On tape :

```
expr("c:=1")
```

On obtient :

```
La variable c contient 1
```

On tape :

```
a:="ifactor(54)";expr(a)
```

ou :

```
expr("ifactor(54)")
```

On obtient :

```
2*3^3
```

15.4 Évaluer une expression sous la forme d'une chaîne : `string`

.
`string` évalue une expression et renvoie sa valeur sous la forme d'une chaîne de caractères.
On peut aussi utiliser la concaténation de l'expression avec une chaîne vide.
On tape :

```
string(ifactor(6))
```

Ou on tape :

```
ifactor(6)+""
```

Ou on tape :

```
""+ifactor(6)
```

On obtient :

```
"2*3"
```

On tape :

```
string('(ifactor(6)'))
```

On obtient :

```
"ifactor(6)"
```

15.5 inString

`inString(l,c)` teste si c est dans la chaine l et renvoie l'indice de c ou 0.
On tape :

```
inString"ABCDEF","C"
```

On obtient :

```
3
```

On tape :

```
inString"ABCDEF","G"
```

On obtient :

```
0
```

15.6 left

`left(l,n)` renvoie la partie gauche de longueur n de la chaine l.
On tape :

```
left("ABCDEF",3)
```

On obtient :

```
"ABC"
```

15.7 right

`right(l,n)` renvoie la partie droite de longueur n de la chaine l.
On tape :

```
right("ABCDEF",2)
```

On obtient :

```
"EF"
```

15.8 mid

`mid(l,d,n)` renvoie la chaine extraite de la chaine l, d'ebutant par le caractère d'indice d, et de longueur n (par défaut n=dim(l)-d). On tape :

```
mid("ABCDEF",2,3)
```

On obtient :

```
"BCD"
```

On tape :

```
mid("ABCDEF",2)
```

On obtient :

```
"BCDEF"
```

15.9 rotate

rotate renvoie la chaine obtenue en mettant le dernier caractère en premier.
On tape :

```
rotate("ABC")
```

On obtient :

```
("CAB")
```

15.10 dim

dim renvoie la longueur de la chaine.
On tape :

```
dim("ABC")
```

On obtient :

```
3
```

On peut aussi utiliser SIZE
On tape :

```
SIZE("ABC")
```

On obtient :

```
3
```

15.11 +

+ de concaténe 2 chaines.
On tape :

```
"ABC"+"DEF"
```

On obtient :

```
"ABCDEF"
```

On peut aussi utiliser CONCAT
On tape :

```
CONCAT("ABC","DEF")
```

On obtient :

```
"ABCDEF"
```

15.12 Avoir la liste ou la chaîne privée de son premier élément : `tail`

`tail(s)` renvoie la liste ou la chaîne `s` privée de son premier élément.
On tape :

```
tail([0,1,2,3])
```

On obtient :

```
[1,2,3]
```

`l:=tail([0,1,2,3])` est équivalent à `l:=suppress([0,1,2,3],0)`
On tape :

```
tail("abcdef")
```

On obtient :

```
"bcdef"
```

`l:=tail("abcdef")` est équivalent à `l:=suppress("abcdef","a")`

15.13 Début d'une liste ou d'une chaîne : `head`

`head(s)` renvoie le premier élément de la liste `s` ou le premier caractère de la chaîne `s`.
On tape :

```
head([0,1,2,3])
```

On obtient :

```
0
```

On tape :

```
head("abcdef")
```

On obtient :

```
"a"
```

Chapitre 16

Les matrices

16.1 Matrice transposée : `tran`

Dans CAS, `tran` renvoie la transposée de la matrice donnée en argument.
On tape :

```
tran([[i,2],[4,5-i]])
```

On obtient :

```
[[i,4],[2,5-i]]
```

16.2 Matrice transposée : `TRN` ou `trn`

Dans CAS, `TRN` renvoie la transposée de la conjuguée de la matrice donnée en argument. On tape :

```
TRN([[i,2],[4,5-i]])
```

Ou on tape :

```
trn([[i,2],[4,5-i]])
```

On obtient :

```
[[-i,4],[2,5+i]]
```

16.3 Déterminant : `DET` ou `det`

Dans HOME, `DET` renvoie le déterminant d'une matrice carrée.
On tape :

```
DET([[1/2,2,4],[4,5,6],[7,8,9]])
```

On obtient :

```
-1.5
```

Dans CAS, `DET` ou `det` renvoie le déterminant d'une matrice carrée.
On tape :

```
DET([[1/2,2,4],[4,5,6],[7,8,9]])
```

Ou on tape :

```
det([[1,2,4],[4,5,6],[7,8,9]])
```

On obtient :

```
-3/2
```

16.4 Résolution d'un système linéaire : `RREF` ou `rref`

Dans HOME, `RREF` permet de résoudre un systéme linéaire de matrice M_1 et de second membre $M3$.
`ADDCOL(M1,M3)=>M2` et `RREF(M2)` renvoie la forme réduite échelonnée de M_2.
Par exemple, on veut résoudre le système : $\{3x + y = 2, 3x + 2y = -2\}$ par rapport à x, y.
On tape :

```
RREF([[3,1,2],[3,2,-2]])
```

On obtient :

```
[[1,0,2],[0,1,-4]]
```

et on en déduit que $x = 2$ et $y = -4$.
On veut résoudre le système :
$\{x + y - z = 5, 2x - y = 7, x - 2y + z = 2\}$ par rapport à x, y, z.
On tape :

```
RREF([[1,1,-1,5],[2,-1,0,7],[1,-2,1,2]])
```

On obtient :

```
[[1,0,-0.333333333333,4],[0,1,-0.666666666667,1],[0,0,0,0]]
```

et on en déduit que $x = 4 + z/3$, $y = 1 + 2z/3$ et $z = z$.

Dans CAS, `rref` ou `RREF` renvoie la forme réduite échelonnée de la matrice argument.
Par exemple, on veut résoudre le système : $\{3x + y = 2, 3x + 2y = -2\}$ par rapport à x, y.
On tape :

```
rref([[3,1,2],[3,2,-2]])
```

Ou on tape :

```
RREF([[3,1,2],[3,2,-2]])
```

On obtient :

```
[[1,0,2],[0,1,-4]]
```

et on en déduit que $x = 2$ et $y = -4$.
On veut résoudre le système :
$\{x + y - z = 5, 2x - y = 7, x - 2y + z = 2\}$ par rapport à x, y, z.
On tape :

```
rref([[1,1,-1,5],[2,-1,0,7],[1,-2,1,2]])
```

Ou on tape :

```
RREF([[1,1,-1,5],[2,-1,0,7],[1,-2,1,2]])
```

On obtient :

```
[[1,0,-1/3,4],[0,1,-2/3,1],[0,0,0,0]]
```

et on en déduit que $x = 4 + z/3$, $y = 1 + 2z/3$ et $z = z$.

16.5 Création de matrices

16.5.1 Créer une matrice à partir d'une expression : `MAKEMAT` et `makemat`

Dans HOME, `MAKEMAT(Expr(I,J),n,p)` crée une matrice à partir d'une expression selon les variables I et J.
I représente l'indice des lignes et J représente l'indice des colonnes et l''indice I va de 1 à n et l'indice J va de 1 à p.
`MAKEMAT(Expr(I,J),n,p)` renvoie la matice $M_{I,J} = Expr(I, J)$ pour $I = 1..n$ et $J = 1..p$.
On tape :

```
MAKEMAT(I*J,2,3)
```

On obtient :

```
[[2,3,4],[3,4,5]]
```

Dans CAS, on met une fonction comme premier argument : la première variable est l'indice de la ligne et la deuxième variable est l'indice de la colonne. Le deuxième argument représente le nombre de ligne et le troisième argument représente le nombre de colonne.
Attention les indices commencent à 0.
On tape :

```
MAKEMAT((j,k)->(j+k),2,3)
```

Ou on tape :

```
makemat((j,k)->(j+k),2,3)
```

On obtient :

```
[[0,1,2],[1,3,5]]
```

16.5.2 Matrice identité : `IDENMAT` ou `identity`

Dans HOME, `IDENMAT(n)` crée la matrice identité d'ordre n.
On tape :

```
IDENMAT(3)
```

On obtient la matrice identité d'ordre 3 :

```
[[1,0,0],[0,1,0],[0,0,1]]
```

Dans CAS,`IDENMAT(n)` crée la matrice identité d'ordre n.
On tape :

```
IDENMAT(3)
```

Ou on tape :

```
identity(3)
```

On obtient la matrice identité d'ordre 3 :

```
[[1,0,0],[0,1,0],[0,0,1]]
```

16.5.3 Matrice aléatoire : `RANDMAT` et `ramn`

Dans HOME, `RANDMAT(M1,n,p)` crée une matrice aléatoire M_1 de n lignes et p colonnes et formée d'entiers entre -99 et +99.
On tape :

```
RANDMAT(M1,2,3)
```

On obtient :

```
[[-24.0,-67.0,38.0],[-73.0,-3.0,72.0]]
```

On tape :

```
M1
```

On obtient :

```
[[-24,-67,38],[-73,-3,72]]
```

Dans CAS, `RANDMAT(n,p)` ou `ranm(n,p)` crée une matrice aléatoire de n lignes et p colonnes et formée d'entiers entre -99 et +99.
On tape :

```
RANDMAT(2,3)
```

Ou on tape :

```
ranm(2,3)
```

Ou on tape :

```
A:=ranm(2,3)
```

On obtient :

```
[[-57,17,39],[-61,23,4]]
```

16.5.4 `JordanBlock`

Dans HOME, on l'utilise sous la forme `CAS.JordanBlock(a,n)` car c'est une fonction du CAS.

Dans CAS, `JordanBlock(a,n)` renvoie une matrice carrée d'ordre n avec a sur la diagonale principale, 1 au-dessus de cette diagonale et 0 ailleurs.
On tape :

```
JordanBlock(7,3)
```

On obtient :

```
[[7,1,0],[0,7,1],[0,0,7]]
```

16.5.5 N-ième matrice de Hilbert `hilbert`

`hilbert` s'utilise dans le CAS (dans HOME il faut mettre `CAS.hilbert`).
`hilbert(n)` renvoie la n-ième matrice de Hilbert c'est à dire :
$H_{j,k} = \dfrac{1}{j+k+1}$ pour $j = 1..n$ et $k = 1..n$.
On tape :

```
hilbert(3)
```

On obtient :

```
[[1,1/2,1/3],[1/2,1/3,1/4],[1/3,1/4,1/5]]
```

16.5.6 Matrice d'une isométrie : `mkisom`

`mkisom` s'utilise dans le CAS (dans HOME il faut mettre `CAS.mkisom`).
`mkisom` a comme argument :

- En dimension 3, la liste des éléments caractéristiques (vecteur directeur de l'axe et angle de la rotation ou vecteur de la normale au plan de symétrie) et `+1` ou `-1` (`+1` pour les isométries directes et `-1` pour les indirectes).
- En dimension 2, l'élément caractéristique (un angle ou un vecteur) et `+1` ou `-1` (`+1` pour les isométries directes et `-1` pour les indirectes).

`mkisom` renvoie la matrice de l'isométrie définie par les arguments.
On tape :

```
mkisom([[-1,2,-1],pi],1)
```

On obtient la matrice d'une rotation d'axe $[-1, 2, -1]$ et d'angle π :

```
[[-2/3,-2/3,1/3],[-2/3,1/3,-2/3],[1/3,-2/3,-2/3]]
```

On tape :

```
mkisom([pi],-1)
```

On obtient la matrice d'une symétrie par rapport à O :

```
[[-1,0,0],[0,-1,0],[0,0,-1]]
```

On tape :

```
mkisom([1,1,1],-1)
```

On obtient la matrice d'une symétrie par rapport au plan $x + y + z = 0$:

```
[[1/3,-2/3,-2/3],[-2/3,1/3,-2/3],[-2/3,-2/3,1/3]]
```

On tape :

```
mkisom([[1,1,1],pi/3],-1)
```

On obtient la matrice produit d'une rotation d'axe $[1, 1, 1]$ et d'angle $\frac{\pi}{3}$ et d'une symétrie par rapport au plan $x + y + z = 0$:

```
[[0,-1,0],[0,0,-1],[-1,0,0]]
```

On tape :

```
mkisom(pi/2,1)
```

On obtient la matrice en dimension 2, de la rotation plane d'angle $\frac{\pi}{2}$:

```
[[0,-1],[1,0]]
```

On tape :

```
mkisom([1,2],-1)
```

On obtient la matrice en dimension 2, de la symétrie plane par rapport à la droite d'équation $x + 2y = 0$:

```
[[3/5,-4/5],[-4/5,-3/5]]
```

On tape pour avoir la matrice en dimension 2 d'une rotation de centre O et d'angle 1 :

```
mkisom(1,1)
```

On obtient :

```
[[cos(1),-sin(1)],[sin(1),cos(1)]]
```

16.5.7 Matrice de Vandermonde : `vandermonde`

`vandermonde` s'utilise dans le CAS (dans HOME il faut mettre `CAS.vandermonde`).
`vandermonde` a comme argument un vecteur de composantes x_j.
`vandermonde` renvoie la matrice de Vandermonde correspondante : elle a pour k^{ime} ligne est le vecteur de composantes x_j^k ($k = 0..n - 1$).
Attention !
`Xcas` numérote les lignes et les colonnes à partir de 0.
On tape :

```
vandermonde([a,2,3])
```

On obtient (si `a` n'est pas affecté) :

```
[[1,1,1],[a,2,3],[a*a,4,9]]
```

16.6 Basique

16.6.1 Norme de Schur ou de Frobenius d'une matrice : `ABS`

On rappelle que `ABS` permet d'avoir :
- la valeur absolue d'un réel,
- le module d'un nombre complexe,
- la longueur d'un vecteur v_j ($(\sum_{j=1}^{n} |v_j|^2)^{1/2}$),
- la norme de Schur ou de Frobenius d'une matrice $a_{j,k}$: $(\sum_{j,k=1}^{n} |a_{j,k}|^2)^{1/2}$.

On tape :

```
ABS([[1,2],[3,4]])
```

On obtient :

```
sqrt(30)
```

en effet $\sqrt{1+4+9+16)} = \sqrt{30}$
On tape :

```
ABS([[1,2],[3,4],[5,11]])
```

On obtient :

```
4*sqrt(11)
```

en effet $\sqrt{1+4+9+16+25+121)} = \sqrt{176} = 4\sqrt{11}$
Dans CAS, on tape :

```
abs([[i,2],[4,5+i]])
```

Ou on tape `||` avec la touche située à côté de la malette :

```
|[[i,2],[4,5+i]]|
```

On obtient :

```
sqrt(47)
```

car $\sqrt{1+4+16+25+1} = \sqrt{47}$
Dans HOME, on tape :

```
ABS([[i,2],[4,5+i]])
```

Ou on tape `||` avec la touche située à côté de la malette :

```
|[[i,2],[4,5+i]]|
```

On obtient :

```
6.8556546004
```

car $\sqrt{1+4+16+25+1} = \sqrt{47} \simeq 6.8556546004$

16.6.2 Maximum des normes des lignes d'une matrice : `ROWNORM` ou `rownorm`

Dans HOME, `ROWNORM` a comme argument une matrice.
`ROWNORM` renvoie le maximum des normes des lignes de cette matrice (la norme d'une ligne étant la somme des valeurs absolues des composantes de la ligne).
On tape :

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

On obtient :

```
15
```

en effet on a $1 + 2 + 3 = 6 < 4 + 5 + 6 = 15$
Dans CAS, `ROWNORM` ou `rownorm` a comme argument une matrice.
`ROWNORM` ou `rownorm` renvoie le maximum des normes des lignes de cette matrice (la norme d'une ligne étant la somme des valeurs absolues des composantes de la ligne).
On tape :

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

Ou on tape :

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

On obtient :

```
15
```

en effet on a $1 + 2 + 3 = 6 < 4 + 5 + 6 = 15$

16.6.3 Max des normes des colonnes d'une matrice : `COLNORM` ou `colnorm`

Dans HOME, `COLNORM` a comme argument une matrice.
`COLNORM` renvoie le maximum des normes des colonnes de cette matrice (la norme d'une colonne étant la somme des valeurs absolues des composantes de la colonne).
On tape :

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

On obtient :

```
9
```

en effet on a $1 + 4 < 2 + 5 < 3 + 6 = 9$
Dans CAS, `COLNORM` ou `colnorm` a comme argument une matrice.
`COLNORM` ou `colnorm` renvoie le maximum des normes des colonnes de cette matrice (la norme d'une colonne étant la somme des valeurs absolues des composantes de la colonne).
On tape :

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

Ou on tape :

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

On obtient :

```
9
```

en effet on a $1+4<2+5<3+6=9$

16.6.4 Norme spectrale d'une matrice : SPECNORM

Dans HOME, SPECNORM sécrit CAS.SPECNORM et a comme argument une matrice M_1.
SPECNORM renvoie la norme spectrale de cette matrice M_1 : c'est la plus grande valeur singulière de la matrice M_1 i.e. la racine de la plus grande valeur propre de la matrice symétrique M_1*TRN(M_1).
On tape :

```
CAS.SPECNORM([[1,1],[0,2]])
```

On obtient :

```
2.28824561127
```

car
`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]`
et
`SVL([[1,1],[0,2]])=[sqrt(sqrt(5)+3),sqrt(-sqrt(5)+3)]` et
$\sqrt{\sqrt{5.}+3}\sim 2.28824561127$,

Dans CAS, SPECNORM a comme argument une matrice A.
SPECNORM renvoie la norme spectrale de cette matrice A : c'est la plus grande valeur singulière de la matrice A i.e. la racine de la plus grande valeur propre de la matrice symétrique $A*$TRAN(A).
On tape :

```
SPECNORM([[1,1],[0,2]])
```

On obtient :

```
2.28824561127
```

car
`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]`
et
`SVL([[1,1],[0,2]])=[sqrt(sqrt(5)+3),sqrt(-sqrt(5)+3)]` et
$\sqrt{\sqrt{5.}+3}\sim 2.28824561127$.

16.6.5 Rayon spectral d'une matrice carrée : `SPECRAD`

Dans HOME, `SPECRAD` sécrit `CAS.SPECRAD` et a comme argument une matrice carrée.
`SPECRAD` renvoie le rayon spectral de cette matrice carrée : le rayon spectral est ègal à la plus grande valeur propre en valeur absolue.
On tape :

```
CAS.SPECRAD([[1,1],[0,-2]])
```

On obtient :

```
2.
```

Dans CAS, `SPECNORM` a comme argument une matrice A.
`SPECNORM` renvoie le rayon spectral de cette matrice carrée : le rayon spectral est ègal à la plus grande valeur propre en valeur absolue.
On tape :

```
SPECRAD([[1,1],[0,-2]])
```

On obtient :

```
2.
```

16.6.6 Conditionnement d'une matrice carrée inversible : `COND`

Dans HOME, `COND` a comme argument une matrice carrée inversible.
`COND` renvoie le conditionnement de cette matrice carrée inversible.
Le conditionnement d'une matrice carrée inversible M_1 est le produit de la norme colonne de M_1 par la norme colonne de M_1^{-1}.
La norme colonne de M_1 de dimension p, q est :
$\texttt{MAX}_{1\leq j\leq p}(\sum_{k=1}^{q}\texttt{ABS(M1[j,k])})$.
On tape :

```
COND([[1,2],[5,6]])
```

On obtient :

```
22
```

On a en effet :
Norme colonne de $[[1,2],[5,6]]$ est 2+6=8
Norme colonne de $[[1,2],[5,6]]^{-1} = [[-1.5, 0.5], [1.25, -0.25]]$ est 1.5+1.25=2.75.
On a bien 2.75*8=22.

16.6.7 Rang d'une matrice : `RANK` ou `rank`

Dans HOME, `RANK` renvoie le rang de la matrice donnée en argument.
On tape :

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

On obtient :

```
2
```

Dans CAS, `RANK` ou `rank` renvoie le rang de la matrice donnée en argument.
On tape :

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

Ou on tape :

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

On obtient :

```
2
```

16.6.8 Étape de la réduction de Gauss-Jordan d'une matrice : `pivot`

`pivot` s'utilise dans le CAS (dans HOME il faut mettre `CAS.pivot`).
`pivot` a trois arguments : une matrice de n lignes et p colonnes et deux entiers l et c vérifiant : $0 \leq l < n$ et $0 \leq c < p$.
`pivot(A,l,c)` renvoie la matrice obtenue en créant des zéros dans la colonne `c` de `A`, avec la méthode de Gauss-Jordan, en utilisant comme pivot l'élément `A[l,c]`.
On tape :

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

On obtient :

```
[[-2,0],[3,4],[2,0]]
```

On tape :

```
pivot([[1,2],[3,4],[5,6]],0,1)
```

On obtient :

```
[[1,2],[2,0],[4,0]]
```

16.6.9 Trace d'une matrice carrée : `TRACE` ou `trace`

Dans HOME, `TRACE` renvoie la trace de la matrice carrée donnée en argument.
On tape :

```
TRACE([[1/2,2,3],[4,5,6],[7,8,9]])
```

On obtient :

```
14.5
```

On tape :

```
TRACE([[i,2],[4,5-i]])
```

On obtient :

```
5
```

Dans CAS, `TRACE` ou `trace` renvoie la trace de la matrice carrée donnée en argument.
On tape :

```
TRACE([[1/2,2,3],[4,5,6],[7,8,9]])
```

Ou on tape :

```
trace([[1,2,3],[4,5,6],[7,8,9]])
```

On obtient :

```
29/2
```

On tape :

```
TRACE([[i,2],[4,5-i]])
```

Ou on tape :

```
trace([[i,2],[4,5-i]])
```

On obtient :

```
5
```

16.7 Avancée

16.7.1 Valeurs propres : `EIGENVAL` et `eigenvals`

Dans HOME, `EIGENVAL` renvoie le vecteur des valeurs propres calculables d'une matrice numérique diagonalisable.
On tape :

```
EIGENVAL([[1,1],[0,2]])
```

On obtient :

```
[2,1]
```

Dans CAS `eigenvals` renvoie le vecteur des valeurs propres calculables d'une matrice.
On tape :

```
eigenvals([[1,1,2],[0,1,1],[0,0,1]])
```

On obtient :

```
[1,1,1]
```

On tape :

```
eigenvals([[1,1,2],[0,2,1],[0,0,3]])
```

On obtient :

```
[3,2,1]
```

On tape :

```
eigenvals([[1,1,2],[0,1,1],[0,0,1]])
```

On obtient :

```
[1,1,1]
```

On tape :

```
eigenvals([[1,1,2],[0,2,1],[0,0,3]])
```

On obtient :

```
[3,2,1]
```

16.7.2 Vecteurs propres : `EIGENVV` et `eigenvects`

Dans HOME, `EIGENVV` renvoie la liste de deux vecteurs celui des vecteurs propres et celui des valeurs propres calculables d'une matrice numérique diagonalisable.
On tape :

```
EIGENVV([[1,1],[0,2]])
```

On obtient :

```
{[[0.707106781187,-1.41421356237],[0.707106781187,0]],[[2,0],[0,1]]}
```

Dans CAS `eigenvects` renvoie la matrice des vecteurs propres d'une matrice diagonalisable.
On tape :

```
eigenvects([[1,1],[0,2]])
```

On obtient :

```
[[1,-1],[1,0]]
```

On tape :

```
eigenvects([[1,1,2],[0,1,1],[0,0,1]])
```

On obtient :

```
"Non diagonalisable, Try Jordan"
```

On tape :

```
eigenvects([[1,1,2],[0,2,1],[0,0,3]])
```

On obtient :

```
[[3,-1,1],[2,-1,0],[2,0,0]],[[3,0,0],[0,2,0],[0,0,1]]
```

16.7.3 Matrice de Jordan `jordan`

`jordan` s'utilise dans le CAS (dans HOME il faut mettre `CAS.jordan` et le résultat sera exact).
`jordan` renvoie la liste formée par la matrice de passage et la forme de Jordan d'une matrice.
On tape :

```
jordan([[1,1,2],[0,2,1],[0,0,3]])
```

On obtient :

```
[[3,-1,1],[2,-1,0],[2,0,0]]
```

On tape :

```
jordan([[1,1,2],[0,1,1],[0,0,1]])
```

On obtient :

```
[[1,2,0],[0,1,0],[0,0,1]],[[1,1,0],[0,1,1],[0,0,1]]
```

Si `A=[[1,1,2],[0,1,1],[0,0,1]]`,`P=[[1,2,0],[0,1,0],[0,0,1]]` et `B=[[1,1,0],[0,1,1],[0,0,1]]`, on a :
`inv(P)*A*P` renvoie `B`.

16.7.4 Matrice diagonale et sa diagonale : `diag`

`diag` s'utilise dans le CAS (dans HOME il faut mettre `CAS.diag`).
Lorsque `diag` a comme argument une matrice, `diag` renvoie le vecteur form' par les éléments de sa diagonale.
Lorsque `diag` a comme argument un vecteur, `diag` renvoie la matrice diagonale de diagonale les éléments de ce vecteur. On tape :

```
diag([[1,0],[0,2]])
```

On obtient :

```
[1,2]
```

On tape :

```
diag([1,2])
```

On obtient :

```
[[1,0],[0,2]]
```

16.7.5 Matrice de Cholesky : `cholesky`

`cholesky` s'utilise dans le CAS (dans HOME il faut mettre `CAS.cholesky`).
`cholesky` a comme argument une matrice symétrique `A`.
`cholesky` renvoie la matrice `L` tel que `A=L*tran(L)`
On tape :

```
cholesky([[3,1],[1,4]])
```

On obtient :

```
[[3/(sqrt(3)),0],[1/(sqrt(3)),(sqrt(33))/3]]
```

et on a bien `[[3/(sqrt(3)),0],[1/(sqrt(3)),(sqrt(33))/3]]*tran([[3/(sqrt(3)),0],[1/(sqrt(3)),(sqrt(33))/3]])`
renvoie `[[3,1],[1,1/3+11/3]]`

16.7.6 Forme normale de Hermite d'une matrice : `ihermite`

`ihermite` calcule la forme normale de Hermite pour une matrice A à coefficients entiers.
`ihermite` renvoie U, B tels que U est inversible dans $\mathbb{Z}$, B est triangulaire supérieure et $B = U * A$.
On tape :

```
ihermite([[1,2],[2,3]])
```

On obtient :

```
[[-3,2],[2,-1]],[[1,0],[0,1]]
```

16.7.7 Réduction de Hessenberg d'une matrice : `hessenberg`

`hessenberg` s'utilise dans le CAS (dans HOME il faut mettre `CAS.hessenberg`).
`hessenberg` a comme premier argument une matrice A et comme deuxième argument 0, -1 ou -2 ou $n > 1$ et n premier.
`hessenberg` renvoie la matrice P de passage et la matrice B semblable à A dont les coefficients sous-sous-diagonaux sont nuls. On dit que B est une matrice de Hessenberg et on a $B = P^{-1}AP$ ou $B \sim P^{-1}AP$ selon le
deuxième argument.

- Avec un seul argument ou comme deuxième argument 0, les calculs sont exacts.
- Avec comme deuxième argument -1, les calculs sont approchés et la matrice B est triangulaire.
- Avec comme deuxième argument -2, les calculs sont approchés et la matrice P est orthogonale et la matrice B a ses coefficients sous-sous-diagonaux nuls.
- Avec comme deuxième argument $n > 1$ et n premier, les calculs sont modulo n et la matrice B est triangulaire.

On tape :

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]])
```

On obtient :

```
[[[1,0,0],[0,4/7,1],[0,1,0]],
[[1,29/7,2],[7,39/7,8],[0,278/49,3/7]]]
```

On tape :

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]],-1)
```

On obtient :

```
[[[-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]]]
```

On tape :

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]],-2)
```

On obtient :

```
[[[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]]]
```

On tape :

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]],3)
```

On obtient :

```
[[[1,0,0],[0,1,0],[0,1,1]],[[1,-1,0],[1,-1,0],[0,1,1]]]
```

16.7.8 `ismith`

`ismith` s'utilise dans le CAS (dans HOME il faut mettre `CAS.ismith`).
`ismith(A)` calcule la forme normale de Smith de la matrice A à coefficients entiers et renvoie les matrices U, B, V avec U et V inversibles dans $\mathbb{Z}$ et où B est une matrice diagonale telle que $B[j, j]$ divise $B[j + 1, j + 1]$ et $B = U * A * V$.
On tape :

```
ismith([[1,2],[2,3]])
```

On obtient :

```
[[1,0],[2,-1]],[[1,0],[0,1]],[[1,-2],[0,1]]
```

On tape :

```
ismith([[9,-36,30],[-36,192,-180],[30,-180,180]])
```

On obtient :

```
[[-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]]
```

16.8 Factorisation

16.8.1 Décomposition LQ d'une matrice : `LQ`

Dans HOME `LQ(M1)`, (resp dans CAS `LQ(A)`), renvoie la décomposition LQ d'une matrice numérique M_1 (resp A) de dimension $m \times n$ en une matrice triangulaire inférieure M_2 (resp L) de dimension $m \times n$, une matrice orthogonale M_3 (resp Q) de dimension $n \times n$ et une matrice de permutation M_4 (resp P) de dimension $n \times n$ telles que l'on ait $M_4 * M_1 = M_2 * M_3$ (resp $P * A = L * Q$).
0n tape :

```
LQ([[4,0,0],[8,-4,3]])
```

On obtient :

```
[[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]]
```

0n tape :

```
LQ([[0.8,0.6],[2.2,0.4]])
```

On obtient :

```
[[1.0,0],[2.0,-1.0]],[[0.8,0.6],
[-0.6,0.8]],[[1,0],[0,1]]
```

0n tape :

```
LQ([[4,3],[11,2]])
```

On obtient :

```
[[5.0,0],[10.0,-5.0]],[[0.8,0.6],
[-0.6,0.8]],[[1,0],[0,1]]
```

0n tape :

```
LQ([[1,2],[3,4]])
```

On obtient :

```
[[2.2360679775,0.],[4.9193495505,0.894427191]],
[[0.4472135955,0.894427191],[0.894427191,-0.4472135955]],
[[1,0],[0,1]]]
```

ce qui signifie que :
`[[1,2],[3,4]]=[[2.2360679775,0.0],[4.9193495505,0.894427191]]*`
`[[0.4472135955,0.894427191],[0.894427191,-0.4472135955]]`
0n tape :

```
[[1,2,3],[3,4,5],[5,6,7]]=>M3
```

```
LQ(M3)
```

On obtient :

```
[[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]]
```

16.8.2 Norme minimale du système linéaire `A*X=B` : `LSQ`

`LSQ(A,B)` calcule la norme minimale selon la méthode des moindres carrés du système linéaire `A*X=B` sur- ou sous-déterminé c'est pour estimer la solution d'un système linéaire `A*X=B` (si `B` est un vecteur) ou des systèmes linéaires `A*X=B` (si `B` est une matrice) pour :

- un système sur-déterminé (on a plus de lignes que de colonnes)
  - si `B` est un vecteur : on cherche `X` de norme euclidienne minimum qui minimise la norme euclidienne de `(AX-B)`.
  - si `B` est une matrice : on cherche `Xj` de norme euclidienne minimum parmi les solutions qui minimise la norme euclidienne de `(AXj-Bj)`
- un système sous determiné (en général on a plus de colonnes que de lignes) On cherche `X` qui minimise la norme de Frobenius de `(AX-B)` (la norme de Frobenius d'une matrice M est $sqrt\sum|M(j,k)|^2$).

- un système exactement déterminé (le nombre de colonnes est égal au nombre de lignes et `A` est inversible).
  On utilise `inv(A)*B` pour avoir `X` qui produit des résultats faux en calcul approché si la matrice est mal conditionnée (équations indépendantes proche l'une de l'autre)

0n tape :

```
LSQ([[1,2],[3,4]],[[5,-1],[11,-1]])
```

On obtient :

```
[[1,1],[2,-1]]
```

0n tape :

```
LSQ([[1,2]],[[5,-1]])
```

On obtient :

```
[[1,-1.2],[2,-0.4]]
```

0n tape :

```
LSQ([[1,2],[3,4],[3,6]],[[5,-1],[11,-1],[15,-3]])
```

On obtient :

```
[[1,1],[2,-1]]
```

0n tape :

```
LSQ([[1,2],[3,4],[3,6]],[[5,-1],[11,-1],[15,-1]])
```

On obtient :

```
[[1,-0.2],[2,-0.1]]
```

16.8.3 Décomposition LU d'une matrice carrée : `LU`

Dans HOME `LU(M1)`, (resp dans CAS `LU(A)`), renvoie la décomposition LU d'une matrice carrée M_1 (resp A) en une matrice M_2 (resp L) triangulaire inférieure (de diagonale 1) et une matrice M_3 (resp U) triangulaire supérieure telles que si M_4 (resp P) est une matrice de permutation on ait $M_4 * M_1 = M_2 * M_3$ (resp $P * A = L * U$).
0n tape :

```
LU([[1,2],[1,4]])
```

On obtient :

```
{[[1,0],[1,1]],[[1,2],[0,2]],[[1,0],[0,1]]}
```

0n tape :

```
LU([[1,2],[3,4]])
```

On obtient :

```
{[[1,0],[0.333333333333,1]],
[[3,4],[0,0.666666666667]], [[0,1],[1,0]]}
```

car on a choisit de mettre des 1 sur la diagonale de `L`.
Cela signifie que :
`[[0,1],[1,0]]*[[1,2],[3,4]])= [[3,0],[1,0.666666]]*[[1,1.333333],[0,1]]`
0n tape :

```
LU([[1,2,4],[4,5,6],[7,8,9]])
```

On obtient :

```
[[[1,0,0],[4,1,0],[7,2,1]],
[[1,2,4],[0,-3,-10],[0,0,1]],[0,1,2]]
```

ce qui signifie que :
`[[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]]`
car la matrice associée à la permutation [0,1,2] est la matrice identité d'ordre 3.
0n tape :

```
LU([[6,12,18],[5,14,31],[3,8,18]])
```

On obtient :

```
[[[1,0,0],[2,1,0],[5/3,(-1)/6,1]],[[3,8,18],[0,-4,-18],[0,0,-2]],[2,0,1]]
```

ce qui signifie que :
`[[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]]`
car la matrice associée à la permutation [2,0,1] est la matrice [[0,0,1],[1,0,0],[0,1,0]].

16.8.4 Décomposition QR d'une matrice carrée : `QR`

Dans HOME `QR(M1)`, (resp dans CAS `QR(A)`), renvoie la décomposition QR d'une matrice carrée M_1 (resp A) en une matrice Q orthogonale et une matrice R triangulaire supérieure telles que si P est une matrice de permutation on ait $M_1 * P = Q * R$ (resp $A * P = Q * R$).
0n tape :

```
QR([[4,11,-2],[3,2,11]])
```

On obtient :

```
{[[0.8,-0.6],[0.6,0.8]],[[5.0,10.0,5.0],[0,-5.0,10.0]],
[[1,0],[0,1]]}
```

0n tape :

```
QR([[1,2],[3,4]])
```

On obtient :

```
[[0.316227766017,0.948683298051],
[0.94868329805,-0.316227766017]],
[[3.16227766017,4.42718872424], [0.0,0.632455532034]],
[[1,0],[0,1]]
```

ce qui signifie que :
`[[0.316227766017,0.948683298051],[0.94868329805,-0.316227766017]]*`
`[[3.16227766017,4.42718872424],[0.0,0.632455532034]]=`
`[[1,2],[3,4]]`
0n tape :

```
QR([[1,2,4],[4,5,6],[7,8,9]])
```

On obtient :

```
[[[1,0,0],[4,1,0],[7,2,1]],
[[1,2,4],[0,-3,-10],[0,0,1]],[0,1,2]]
```

ce qui signifie que :
`[[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]]`

16.8.5 Réduction de Hessenberg d'une matrice : `SCHUR`

Dans HOME `SCHUR(M1)`, (resp dans CAS `SCHUR(A)`), renvoie les matrices numériques $[P, B]$ tel que $B = inv(P) * M1 * P$ (resp $B = inv(P) * A * P$) avec B triangulaire. On a `SCHUR(A)=hessenberg(A,-1)`. B est la matrice de Hessenberg semblable à la matrice M_1 (res A).
On tape :

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

On obtient deux matrices `P` (matrice orthogonale de passage `tran(P)=inv(P)`) et `B` (matrice triangulaire semblable à l'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]]]
```

et on a :
`B~ inv(P)*[[1,2,3],[4,5,6],[-1,3,-2]]*P`.
On tape :

```
SCHUR([[1,2,4],[4,5,6],[7,8,9]])
```

On obtient deux matrices `P` (matrice orthogonale de passage `tran(P)=inv(P)`) et `B` (matrice triangulaire semblable á l'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]]]
```

et on a :
`MB~ inv(P)*[[1,2,3],[4,5,6],[-1,3,-2]]*P`

16.8.6 Singular value decomposition : `SVD` et `svd`

Dans HOME `SVD(M1)` renvoie 1 matrice `M2`, 1 vecteur `M3`, 1 matrice `M4`.
Cela donne la factorisation de la matrice rectangulaire numérique réelle M_1 (de dimension $m * n$) en `M2*diag(M3)*TRN(M4)` où `M2` est une matrice orthogonale $m * m$,`M4` est une matrice orthogonale $n * n$, et `diag(M3)` est une matrice diagonale de dimension $m * n$ ayant comme diagonale les valeurs singulières `M3` de M_1.
Dans CAS `svd(A)` renvoie 1 matrice `U`, 1 vecteur `S`, 1 matrice `Q`.
Cela donne la factorisation de la matrice rectangulaire numérique réelle A (de dimension $m*n$) en `U*diag(S)*TRN(Q)` où `U` est une matrice orthogonale $m*m$, `Q` est une matrice orthogonale $n * n$, et `diag(S)` est une matrice diagonale de dimension $m * n$ ayant comme diagonale les valeurs singulières `S` de A.
On tape dans HOME :

```
M2,M3,M4:=SVD([[1,2],[2,1]])
```

On obtient :

```
{[[[0.707106781187,-0.707106781187],[0.707106781187,0.707106781187]],
                          [3,1],
[[0.707106781187,0.707106781187],[0.707106781187,-0.707106781187]]]}
```

et on a (ici $M4$ est symétrique) :
`M2*diag(M3)*TRN(M4)`.
On tape dans CAS :

```
U,S,Q:=SVD([[1,2],[2,1]])
```

Ou on tape dans CAS :

```
U,S,Q:=svd([[1,2],[2,1]])
```

On obtient :

```
{[[[0.707106781187,-0.707106781187],[0.707106781187,0.707106781187]],
                        [3.,1.],
[[0.707106781187,0.707106781187],[0.707106781187,-0.707106781187]]]}
```

et on a (Q est symétrique) :
`[[0.707106781187,-0.707106781187],[0.707106781187,0.707106781187]]*`
`[[3,0],[0,1]]*`
`[[0.707106781187,0.707106781187],[0.707106781187,-0.707106781187]]=`
`[[1,2],[2,1]]`
On tape dans CAS :

```
SVD([[1,2],[3,4]])
```

Ou on tape dans CAS :

```
svd([[1,2],[3,4]])
```

On obtient :

```
[[-0.404553584834,-0.914514295677],[-0.914514295677,0.404553584834]],

            [5.46498570422,0.365966190626]


[[-0.576048436767,0.81741556047],[-0.81741556047,-0.576048436766]],
```

car $[[-0.404553584834, -0.914514295677], [-0.914514295677, 0.404553584834]]*$
$[[5.46498570422, 0], [0, 0.365966190626]]*$
$TRN([[-0.576048436767, 0.81741556047], [-0.81741556047, -0.576048436766]]) =$
$[[1.0, 2.0], [3.0, 4.0]]$

16.8.7 Valeurs singulières : `SVL`

Dans HOME `SVL(M1)`, (resp dans CAS `SVL(A)`), renvoie la liste des valeurs singulières de la matrice numérique réelle M_1 (resp A) i.e. les racines carrés des valeurs propres de la matrice réelle symétrique M_1*TRN(M_1) (resp($A*$TRN(A)).
On tape :

```
SVL([[1,4],[1,1]])
```

On obtient :

```
[4.30277563773,0.697224362268]
```

car `eigenvals([[1,4],[1,1]]*[[1,1],[4,1]])` renvoie ;
`(5*sqrt(13)+19)/2,(-5*sqrt(13)+19)/2` et
`sqrt((5*sqrt(13)+19)/2.),sqrt((-5*sqrt(13)+19)/2.)` renvoie
`4.30277563773,0.697224362268`
On tape :

```
SVL([[1,2],[2,1]])
```

On obtient :

```
[3,1]
```

car `EIGENVAL([[1,2],[2,1]]*[[1,2],[2,1]])` renvoie `[9,1]`.
On tape :

```
SVL([[1,2],[3,4]])
```

On obtient :

```
[5.46498570422,0.365966190626]
```

car `EIGENVAL([[1,2],[3,4]]*[[1,3],[2,4]])` renvoie
`[29.8660687473,0.133931252682]`
qui sont les valeurs approchées de $\sqrt{221}+15$ et $-\sqrt{221})15$.
On a :
$\sqrt{\sqrt{221}+15} \sim 5.46498570422$, $\sqrt{-\sqrt{221}+15} \sim 0.365966190626$

16.9 Vecteur

16.9.1 Produit vectoriel : `CROSS` ou `cross`

Dans HOME, `CROSS` renvoie le produit vectoriel des deux vecteurs.
On tape :

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

ou

```
CROSS({1,2,3},{4,5,6})
```

On obtient :

```
[-3,6,-3]
```

car $2*6-5*3=-3, 4*3-1*6=6, 5-4*2=-3$
Dans CAS, `CROSS` ou `cross` renvoie le produit vectoriel des deux vecteurs.
On tape :

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

ou

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

On obtient :

```
[-3,6,-3]
```

car $2*6-5*3=-3, 4*3-1*6=6, 5-4*2=-3$

16.9.2 Produit scalaire : `DOT` ou `dot`

Dans HOME, `DOT` renvoie le produit scalaire des deux vecteurs.
On tape :

```
DOT([1,2,3],[3,4,5])
```

On obtient :

```
26
```

car $1*3+2*4+3*5=26$
Dans CAS `DOT` renvoie le produit scalaire des deux vecteurs.
On tape :

```
DOT([1,2,3],[3,4,5])
```

Ou on tape :

```
dot([1,2,3],[3,4,5])
```

On obtient :

```
26
```

car $1*3+2*4+3*5=26$

16.9.3 Norme l^2 :`l2norm`

`l2norm` s'utilise dans le CAS (dans HOME il faut mettre `CAS.l2norm` et la réponse sera exacte).
`l2norm` calcule la norme l^2 d'un vecteur : c'est la racine carrée de la somme des carrés de ses coordonnées.
On tape :

```
l2norm([3,-4,2])
```

Ou on tape :

```
l2norm(vecteur(3,-4,2))
```

On obtient :

```
sqrt(29)
```

En effet : `x=3, y=-4, z=2` et $29=|x|^2+|y|^2+|z|^2$.

16.9.4 Norme l^1 :`l1norm`

`l1norm` s'utilise dans le CAS (dans HOME il faut mettre `CAS.l1norm` et la réponse sera exacte).
`l1norm` calcule la norme l^1 d'un vecteur : c'est la somme des valeurs absolues de ses coordonnées.
On tape :

```
l1norm([3,-4,2])
```

Ou on tape :

```
l1norm(vecteur(3,-4,2))
```

On obtient :

```
9
```

En effet : `x=3, y=-4, z=2` et `9=|x|+|y|+|z|`.

16.9.5 Norme du maximum :`maxnorm`

`maxnorm` s'utilise dans le CAS (dans HOME il faut mettre `CAS.maxnorm` et la réponse sera exacte).
`maxnorm` calcule la norme l^∞ d'un vecteur : c'est le maximum des valeurs absolues de ses coordonnées.
On tape :

```
maxnorm([3,-4,2])
```

Ou on tape :

```
maxnorm(vecteur(3,-4,2))
```

On obtient :

```
4
```

En effet : `x=3, y=-4, z=2` et `4=max(|x|,|y|,|z|)`.

Chapitre 17

Les fonctions spéciales

17.1 La fonction β : `Beta`

`Beta` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Beta` et la réponse sera exacte).
`Beta` a comme argument deux réels a, b ou trois réels a, b, p ou trois réels et 1 $a, b, p, 1$.

- avec 2 arguments a, b, `Beta` calcule les valeurs de la fonction β au point a, b de $\mathbb{R}^2$.
  On a par définition :
  $$\beta(x, y) = \frac{\Gamma(x) * \Gamma(y)}{\Gamma(x + y)}$$
  On a :
  $$\beta(1, 1) = 1$$
  $$\beta(n, 1) = \frac{1}{n}$$
  et :
  $$\beta(n, 2) = \frac{1}{n(n + 1)}$$
  On a :
  `Beta(a,b)`=$\int_0^1 t^{a-1} * (1 - t)^{b-1} dt$
  `Beta(a,b)` est défini pour a et b réels positifs (pour que l'intégrale soit convergente).
  On tape :
  ```
  Beta(5,2)
  ```
  On obtient :
  ```
  1/30
  ```
  On tape :
  ```
  simplify(Beta(5,-3/2))
  ```
  On obtient :
  ```
  256/15
  ```
  On tape :
  ```
  Beta(x,y)
  ```

On obtient :

```
Gamma(x)*Gamma(y)/Gamma(x+y)
```

On tape :

```
Beta(5.1,2.2)
```

On obtient :

```
0.0242053671402
```

– avec 3 arguments a, b, p c'est la fonction Beta incomplète pour p entre 0 et 1, c'est :
`Beta(a,b,p)`=$\int_0^p t^(a-1)*(1-t)^(b-1)dt$, l'integrale va de 0 à p au lieu de 0 à 1 pour la fonction Beta. On tape :

```
Beta(5,2,0.5)
```

On obtient :

```
0.00364583333333
```

– avec 4 arguments $a, b, p, 1$ si on met 1 en 4eme argument cela calcule la fonction Beta incomplète regularisée i.e. la fonction Beta incomplète qui est divisé par Beta(a,b). On tape :

```
Beta(5,2,0.5,1)
```

On obtient :

```
0.109375
```

en effet `Beta(5,2)=1/30` et `0.00364583333333*30=0.109375`

17.2 La fonction Γ : `Gamma`

`Gamma` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Gamma` et la réponse sera exacte).
`Gamma` a comme argument un nombre a.
`Gamma` calcule les valeurs de la fonction Γ au point a.
On a par définition :

$$\Gamma(a) = \int_0^{+\infty} e^{-t} t^{a-1} dt, \text{ si } a > 0$$

et on utilise la formule :

$$\Gamma(a+1) = a * \Gamma(a) \text{ si } a \text{ n'est pas un entier negatif}$$

Donc :

$$\Gamma(1) = 1$$

$$\Gamma(a+1) = a * \Gamma(a)$$

et ainsi :

$$\Gamma(n+1) = n!$$

On tape :

```
Gamma(5)
```

On obtient :

```
24
```

On tape :

```
Gamma(1/2)
```

On obtient :

```
sqrt(pi)
```

On tape :

```
Gamma(0.7)
```

On obtient :

```
1.29805533265
```

On tape :

```
Gamma(-0.3)
```

On obtient :

```
-4.32685110883
```

En effet : `Gamma(0.7)=-0.3*Gamma(-0.3)`
On tape :

```
Gamma(-1.3)
```

On obtient :

```
3.32834700679
```

En effet :
`Gamma(0.7)=-0.3*Gamma(-0.3)=(-0.3)*(-1.3)*Gamma(-1.3)`

17.3 Les derivées de la fonction DiGamma : `Psi`

`Psi` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Psi` et la réponse sera exacte).
`Psi` a comme arguments un réel a et un entier n (par défault $n = 0$).
`Psi` est la valeur de la n-ième dérivée de la fonction DiGamma au point a.
La fonction DiGamma est la dérivée de $\ln(\Gamma(x))$.
On tape :

```
Psi(3,1)
```

On obtient :

```
pi^2/6-5/4
```

On peut ommettre le paramètre n lorsque $n = 0$.
Lorsque `Psi` a comme seul paramètre un nombre a, `Psi` renvoie la valeur de la fonction DiGamma au point a :
on a donc `Psi(a,0)=Psi(a)`.
On tape :

```
Psi(3)
```

On obtient :

```
Psi(1)+3/2
```

On tape :

```
evalf(Psi(3))
```

On obtient :

```
.922784335098
```

17.4 La fonction ζ : `Zeta`

`Zeta` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Zeta` et la réponse sera exacte).
`Zeta` a comme argument un réel x.
`Zeta` renvoie pour $x > 1$: $\sum_{n=1}^{+\infty} \frac{1}{n^x}$.
On tape :

```
Zeta(2)
```

On obtient :

```
pi^2/6
```

On tape :

```
Zeta(4)
```

On obtient :

```
pi^4/90
```

17.5 La fonction erf : `erf`

`erf` s'utilise dans le CAS (dans HOME il faut mettre `CAS.erf` et la réponse sera exacte).
`erf` a comme argument un nombre a.
`erf` calcule les valeurs de la fonction erf au point a.
On a par définition :

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

On a :

$$erf(+\infty) = 1$$
$$erf(-\infty) = -1$$

En effet on sait que :

$$\int_0^{+\infty} e^{-t^2} dt = \frac{\sqrt{\pi}}{2}$$

On tape :

```
erf(1)
```

On obtient :

```
0.84270079295
```

On tape :

```
erf(1/(sqrt(2)))*1/2+0.5
```

On obtient :

```
0.841344746069
```

Remarque
Il y a une relation entre les fonctions `erf` et `normal_cdf` :
`normal_cdf`$(x) = \frac{1}{2} + \frac{1}{2}$`erf`$(\frac{x}{\sqrt{2}})$
En effet :
normal_cdf$(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt$
donc avec le changement de variables $t = u * \sqrt{2}$ on a :
normal_cdf$(x) = \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-u^2} du = \frac{1}{2} + \frac{1}{2}erf(\frac{x}{\sqrt{2}})$
On vérifie en tapant :
`normal_cdf(1)=0.841344746069`

17.6 La fonction $erfc$: erfc

`erfc` s'utilise dans le CAS (dans HOME il faut mettre `CAS.erfc` et la réponse sera exacte).
`erfc` a comme argument un nombre a.
`erfc` calcule les valeurs de la fonction $erfc$ au point a.
On a par définition :

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^{+\infty} e^{-t^2} dt = 1 - erf(x)$$

On a :

$$erfc(0) = 1$$

$$erfc() = -1$$

En effet on sait que :

$$\int_0^{+\infty} e^{-t^2} dt = \frac{\sqrt{\pi}}{2}$$

On tape :

```
erfc(1)
```

On obtient :

```
0.15729920705
```

On tape :

```
1- erfc(1/(sqrt(2)))*1/2
```

On obtient :

```
0.841344746069
```

Remarque
Il y a une relation entre les fonctions `erfc` et `normal_cdf` :
`normal_cdf`$(x) = 1 - \frac{1}{2}$`erfc`$(\frac{x}{\sqrt{2}})$
En effet :
normal_cdf$(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt$
donc avec le changement de variables $t = u * \sqrt{2}$
normal_cdf$(x) = \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-u^2} du = 1 - \frac{1}{2}$erfc$(\frac{x}{\sqrt{2}})$
On vérifie en tapant :
`normal_cdf(1)=0.841344746069`

17.7 La fonction exponentielle integrale Ei : `Ei`

`Ei` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Ei` et la réponse sera exacte).
`Ei` a comme argument un nombre complexe a.
`Ei` calcule les valeurs de la fonction Ei au point a.
On a par définition :

$$Ei(x) = \int_{t=-\infty}^{x} \frac{\exp(t)}{t} dt$$

Pour $x > 0$, on prolonge par la valeur principale de l'intégrale (les morceaux en 0^- et 0^+ se compensent). On a :

$$Ei(0) = -\infty, \quad Ei(-\infty) = 0$$

Lorsque l'on est proche de $x = 0$ on sait que :

$$\frac{\exp(x)}{x} = \frac{1}{x} + 1 + \frac{x}{2!} + \frac{x^2}{3!} + ... + \frac{x^n}{(n-1)!}....$$

on a donc pour $x \in \mathbb{C} - \mathbb{R}^+$, (la fonction est discontinue sur $\mathbb{R}^+$) :

$$Ei(x) = \ln(-x) + \gamma + x + \frac{x^2}{2.2!} + \frac{x^3}{3.3!} + ...$$

où γ = la constante d'Euler = 0.57721566490..
sur l'axe $x > 0$ on prend :

$$Ei(x) = \ln(x) + \gamma + x + \frac{x^2}{2.2!} + \frac{x^3}{3.3!} + ...$$

On tape :

```
Ei(1.)
```

On obtient :

```
1.89511781636
```

On tape :

```
Ei(-1.)
```

On obtient :

```
-0.219383934396
```

On tape :

```
Ei(1.)-Ei(-1.)
```

On obtient :

```
2.11450175075
```

On tape :

```
int((exp(x)-1)/x,x=-1..1.)
```

On obtient :

```
2.11450175075
```

On tape :

```
evalf(Ei(-1)-sum((-1)^n/n/n!,n=1..100))
```

On obtient la constante d'Euler γ :

```
0.577215664901532860606507
```

17.8 La fonction sinus integral Si : Si

Si s'utilise dans le CAS (dans HOME il faut mettre CAS.Si et la réponse sera exacte).
Si a comme argument un nombre complexe a.
Si calcule les valeurs de la fonction Si au point a.
On a par définition

$$Si(x) = \int_{t=0}^{x} \frac{\sin(t)}{t} dt$$

On a $Si(0) = 0$, $Si(-\infty) = -\frac{\pi}{2}$, $Si(+\infty) = \frac{\pi}{2}$. Lorsque l'on est proche de $x = 0$ on sait que :

$$\frac{\sin(x)}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} + ... + (-1)^n \frac{x^{2n}}{(2n+1)!} ....$$

ce qui donne par intégration le développement en séries de Si en 0.
On observe aussi que Si est une fonction impaire.
On tape :

```
Si(1.)
```

On obtient :

```
0.946083070367
```

On tape :

```
Si(-1.)
```

On obtient :

```
-0.946083070367
```

On tape :

```
Si(1.)+Si(-1.)
```

On obtient :

```
0
```

On tape :

```
Si(1.)-Si(-1.)
```

On obtient :

```
1.89216614073
```

On tape :

```
int(sin(x)/x,x=-1..1.)
```

On obtient :

```
1.89216614073
```

17.9 La fonction cosinus integral Ci : `Ci`

`Ci` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Ci` et la réponse sera exacte).
`Ci` a comme argument un nombre complexe a.
`Ci` calcule les valeurs de la fonction Ci au point a.
On a par définition :

$$Ci(x) = \int_{t=+\infty}^{x} \frac{\cos(t)}{t} dt = \ln(x) + \gamma + \int_0^x \frac{cos(t) - 1}{t}\, dt$$

On a : $Ci(0) = -\infty$, $Ci(-\infty) = i\pi$, $Ci(+\infty) = 0$. Lorsque l'on est proche de $x = 0$ on sait que

$$\frac{\cos(x)}{x} = \frac{1}{x} - \frac{x}{2} + \frac{x^3}{4!} + ... + (-1)^n \frac{x^{2n-1}}{(2n)!} ....$$

ce qui donne par intégration le développement en séries de Ci.
On tape :

```
Ci(1.)
```

On obtient :

```
0.337403922901
```

On tape :

```
Ci(-1.)
```

On obtient :

```
0.337403922901+3.14159265359*i
```

On tape :

```
Ci(1.)-Ci(-1.)
```

On obtient :

```
-3.14159265359*i
```

On tape :

```
int((cos(x)-1)/x,x=-1..1.)
```

On obtient :

```
-3.14159265359*i
```

Quatrième partie

Les Applications et la touche Apps

Chapitre 18

Le menu Geometry

18.1 Généralités

Nous allons décrire ici l'application de géométrie qui permet de faire de la géométrie interactive.
Dans le bandeau figure les commandes classées par thèmes :
`Point Droite Polygon Courbe Transform`.
Comme toutes les Applications, l'Application de géométrie plane a 3 vues :

- `Symb` est la vue symbolique.
  Dans la vue symbolique se trouve toutes les commandes que l'on a effectuées pour avoir la vue graphique. Ces commandes ont soit été tapées par l'utilisateur dans la vue symbolique à l'aide du menu `New->Cmds` ou `Edit->Cmds` du bandeau, soit ont été enregistées automatiquement lorsque l'on fait la figure directement dans la vue graphique à l'aide du bandeau de la vue graphique et de son doigt ou du curseur.

  Il faut noter que les points ou les objets géométriques créés directement dans l'écran graphique sont notés de façon automatique :
  les objets géométriques sont nommés et désignés par `A, B, C...` mais sont sauvés dans les variables `GA, GB, GC...` variables que l'on retrouve dans la vue symbolique. Il faut noter que le point `F` n'existe pas car `GF` est une commande du CAS.
  La touche `Symb` permet aussi d'insérer des commandes ou de modifier les noms des objets géométriques :

  - vous pouvez mettre en surbrillance une ligne de commande en utilisant les flèches,
  - vous pouvez changer l'ordre des commandes en utilisant les flèches ↑ et ↓ du bandeau : avec ces flèches vous déplacez la ligne mise en surbrillance. Cela permet d'insérer, par exemple, la nouvelle commande créée là où vous le souhaitez.
  - si un nom ne vous convient pas vous pouvez le changer dans la vue symbolique avec `Edit` du bandeau.

Lorsqu'on est dans le CAS ou dans l'Application de `géométrie`, les commandes de géométrie sont en anglais et sont classées par thème dans le menu `App->Geometry` de la touche . Il y a 7 thèmes :
`Point, Line, Polygon, Curve, Transform, Measure, Test.`
Lorsqu'on est dans l'Application de `géométrie`, la touche `Symb` permet d'accéder plus facilement à ces commandes : il faut faire `New` du bandeau puis `Cmds` du bandeau et cela ouvre un menu contenant les 5 premiers thèmes du menu `App->Geometry` de la touche (`Point, Line, Polygon, Curve, Transform`).
Cela permet d'avoir les commandes permettant de faire une figure géométrique depuis la vue symbolique.
Remarque
La commande `trace` permet de garder la trace d'un objet lorsqu'on le déplace. Cette commande peut être activer depuis la vue graphique avec le menu `Point->Plus->trace` (cf 1 pour un exemple). Pour effacer la trace ou pour l'arrêter, il faut utiliser ce menu `Point->Plus`.
La commande `display` permet de changer la couleur d'un objet géométrique.
Cette commande a comme option le nom des couleurs ou leur numéro :
`black(0) red(1) green(2) yellow(3) blue(4) cyan(5) magenta(6) white(7)`
et `filled` qui permet de faire des figures pleines.
Par exemple on tape :

```
circle(0,1,display=red)
```

Ou on tape :

```
circle(0,1,display=1)
```

On obtient :

```
le cercle de centre 0 et de rayon 1 en rouge dans
la vue numérique
```

On tape :

```
circle(0,1,display=cyan+filled)
```

Ou on tape :

```
circle(0,1,display=5+filled)
```

On obtient :

```
le disque de centre 0 et de rayon 1 en cyan dans
la vue numérique
```

vous pouvez aussi activer ces option depuis la vue graphique avec la touche `Menu`

– `Plot` est la vue graphique
Pour faire la figure directement dans la vue graphique on utilise le bandeau de la vue graphique et de son doigt ou du curseur ou du clavier car on peut désigner les arguments des commandes autrement qu'en les montrant avec le doigt ou avec le curseur. En effet, on peut taper par exemple `A` en mode `ALPHA` pour désigner l'objet géométrique `GA` (point, droite, courbe...) de la figure, ou `A,B` pour désigner les objets `GA,GB` comme arguments d'une commande.
On retrouve dans le bandeau de la vue graphique `Zoom` et les 5 premiers thèmes du menu `App->Geometry` :
– `Point`

  - Line
  - Polygon
  - Curve
  - Transform

  Les étapes de la construction sont enregistrées sous la forme de commandes et sont visibles en appuyant sur la touche Symb.
  La touche Menu pressée depuis la vue graphique permet de changer la couleur du dessin et faire des formes pleines.
- Num est la vue numérique
  La vue numérique permet d'utiliser les commandes numériques liées au graphique et d'avoir des résultats numériques.
  Les commandes donnant un résultat numérique (coordonnées des points, équation des droites ou des courbes...) peuvent être exécutées dans lécran Num en utilisant New puis Cmds du bandeau de lécran Num. On retrouve à l'aide du menu New->Cmds ou Edit->Cmds du bandeau de la vue numérique les 2 derniers thèmes du menu App->Geometry :
  - Measure
  - Test

Au fur et à mesure de leur construction,les objets géométriques sont nommés et désignés par A, B, C... mais sont sauvés dans les variables GA, GB, GC....
Remarque
Toutes les fonctions de géométrie peuvent être utilisées depuis le CAS, mais alors la réponse sera par exemple point(1,2) ou line(x=1) mais ne sera pas dessinée.
On peut donc faire de la géométrie analytique $2-d$ et même $3-d$ depuis le CAS : on peut par exemple taper depuis le CAS :
g:=line(x=1) puis rajouter dans l'écran Symb GK:=g.
Attention
On peut retrouver la valeur de ces variables dans la vue symbolique ou dans la vue numérique ou dans l'écran du CAS. Si par exemple dans l'application de géométrie on a : GA:=point(4.16+2.13*i) et que dans le CAS, on tape GA:=5, GA sera égal à 5 tant que l'on a pas utilisé Plot de l'application de géométrie car après son utilisation on aura à nouveau GA:=point(4.16+2.13*i).
Il est donc dangereux d'utiliser les variables GA, GB... dans le CAS.

18.2 Point

18.2.1 Point défini comme barycentre de n points : barycenter

barycenter, en géométrie plane, a comme argument n listes de longueur 2 (resp une matrice ayant n lignes et deux colonnes) :
le premier élément de la liste j (resp le $j$ième élément de la première colonne de la matrice) contient le point A_j ou le nombre complexe a_j représentant l'affixe de ce point, le deuxième élément de la liste j (resp le $j$ième élément de la deuxième colonne) contient le coefficient réel α_j affecté à A_j.
barycenter renvoie et trace le point qui est le barycentre des points A_j d'affixes a_j affectés des coefficients réels α_j lorsque $\sum \alpha_j \neq 0$.

Si $\sum \alpha_j = 0$, `barycentre` ou `barycenter` renvoie une erreur.
On tape :

```
barycenter([1+i,1],[1-i,1])
```

Ou on tape :

```
barycenter([point(1,1),1],[point(1,-1),1])
```

Ou on tape :

```
barycenter([[1+i,1],[1-i,1]])
```

Ou on tape :

```
barycenter([[point(1,1),1],[point(1,-1),1]])
```

On obtient dans l'Application de géométrie :

```
Le point d'affixe 1 est tracé avec une croix avec son
nom
```

On obtient dans le CAS :

```
point(1)
```

Attention dans l'Application de géométrie pour avoir comme réponse un nombre complexe il faut demander l'affixe du barycentre, sinon on a le tracé du point barycentre.
On tape :

```
affix(barycenter([1+i,1],[1-i,1]))
```

Ou on tape :

```
affix(barycenter([[1+i,1],[1-i,1]]))
```

On obtient :

```
1
```

Dans l'écran CAS `barycenter` est aussi utilisable en géométrie $3-d$ et a comme argument n listes de longueur 2 (ou une matrice ayant n lignes et 2 colonnes). Le premier élément de la liste j (resp le $j$ième élément de la première colonne de la matrice) contient le point A_j, le deuxième élément de la liste j (resp le $j$ième élément de la deuxième colonne) contient le coefficient réel α_j affecté à A_j.
`barycenter` renvoie `point([a,b,c])` où `[a,b,c]` sont les coordonnées du barycentre de ces n points.
On tape :

```
barycenter([point(0,0,0),1],[point(3,3,3),2])
```

On obtient :

```
pnt(pn[(point[2,2,2],0)])
```

18.2.2 Le point en géométrie : `point`

Dans la vue graphique, pour obtenir un point il suffit d'être en mode `point` (i.e. `Point` du bandeau, puis sélectionner `point` et `Enter`) et d'amener le curseur à l'endroit voulu (avec le doigt ou les flèches) puis de valider avec `Enter` : un point s'affiche avec un nom.
Ce nom est crée automatiquement : `A` puis `B` etc...
On peut aussi utiliser la commande `point` :
`point` a comme argument un nombre complexe ou un couple de 2 nombres réels.
Attention
Si `a,b` est un couple de 2 nombres complexes dont 1 est non réel, `GK:=point(a,b)` renvoie 2 points de même nom (ici `K`) l'un d'affixe `a`, l'autre d'affixe `b`.
Lorsque `a,b` est un couple de 2 nombres réels, `GA:=point(a,b)` renvoie et dessine le point ayant pour affixe `a+ib`.
On tape :

```
GA:=point(1+i)
```

On obtient :

```
Le point A d'affixe 1+i est tracé avec une croix
```

On tape :

```
GB:=point(-2,1)
```

On obtient :

```
Le point B d'affixe -2+i est tracé avec une croix
```

On tape :

```
GC:=point(-2,i)
```

On obtient :

```
Les 2 points d'affixe -2 et i sont tracés avec une
croix et sont notés C
```

Remarque Lorqu'on fait une affectation par exemple `GA:=point(-2+i)` cela a pour effet de stocker le `point(-2+i)` dans la variable `GA`, de dessiner le point avec une croix et de lui mettre comme légende le nom qui est situé à gauche de `:=` en supprimant la lettre `G` : ici `A`.
Si on fait plusieurs affectations avec un seul signe `:=` par exemple :
`GD,GE:=point(-2+i),point(2+i)` la variable `GD` contient le `point(-2+i)`, la variable `GE` contient `point(-2+i)` mais on ne pourra pas deplacer ces points en mode `pointeur`.
Pour éviter cela on doit taper :
`GL:=point(-2+i),point(2+i):;GD:=L[0];GE:=L[1]`
ou
`GL:=point(-2+i,2+i):;GD:=L[0];GE:=L[1]` ce qui définit le point `D` d'affixe -2 et le point `E` d'affixe i (car l'affixe de `GD` n'est pas réel !).

18.2.3 Orthocentre d'un triangle : `orthocenter ou orthocenter`

`orthocenter` ou `orthocentre` a comme argument un triangle ou trois points ou trois nombres complexes désignant l'affixe de trois points.
`orthocenter` ou `orthocentre` trace et renvoie le point qui est l'orthocentre du triangle ou du triangle formé par ces trois points.
On tape :

```
orthocentre(0,1+i,-1+i)
```

Ou on tape :

```
orthocentre(triangle(0,1+i,-1+i))
```

On obtient :

```
Le point d'affixe 0 est tracé avec une croix
```

On tape :

```
T:=triangle(-i,2+i,-1+i);H:=orthocentre(T)
```

On obtient :

```
Le triangle T et le point H d'affixe 0 sont tracés
```

18.2.4 Le milieu d'un segment : `midpoint`

`midpoint`, en géométrie plane, a comme argument 2 points ou 2 nombres complexes représentant l'affixe de ces points (ou encore une liste de 2 points ou de 2 complexes).
`midpoint` renvoie et dessine le point milieu du segment défini par ces deux points.
On tape :

```
midpoint(-1,1+i)
```

On obtient dans l'Application de géométrie :

```
Le point d'affixe i/2 est tracé avec son nom
```

Dans l'écran CAS `midpoint` est utilisable en géométrie $3-d$ et renvoie le point milieu du segment défini par deux points.
On tape :

```
midpoint(point(0,0,0),point(2,2,2))
```

On obtient :

```
point(1,1,1)
```

18.2.5 L'isobarycentre de n points : `isobarycenter`

`isobarycenter` a comme argument la liste (ou la séquence) de n points ou de n nombres complexes représentant l'affixe de ces points.
`isobarycenter` renvoie et trace un point qui est l'isobarycentre de ces n points.
On tape :

```
isobarycenter(0,2,2*i)
```

On obtient :

```
Le point d'affixe 2/3+2*i/3 est tracé avec une croix
dans l'Application de géométrie
```

Dans l'écran CAS `isobarycenter` est utilisable en géométrie $3-d$ a comme argument la liste (ou la séquence) de n points.
`isobarycenter` renvoie `point([a,b,c])` où `[a,b,c]` sont les coordonnées de l'isobarycentre de ces n points.
On tape :

```
isobarycenter(point(0,0,0),point(3,3,3))
```

On obtient :

```
pnt(pn[(point[3/2,3/2,3/2],0]))
```

18.2.6 Définir au hasard un point 2-d : `point2d`

`point2d` a comme argument une séquence de noms de points.
`point2d` définit au hasard, les coordonnées entières (entre -5 et +5) des points 2d donnés en argument.
On tape :

```
point2d(A,B,C)
```

Puis on tape :

```
triangle(A,B,C)
```

On obtient :

```
Le tracé d'un triangle ABC
```

Attention
Les points définit par la commande `point2d` sont fixés une fois pour toute et donc ne pourront pas être déplacé.

18.2.7 Point sur un objet géométrique : `element`

`element` peut avoir différents types d'arguments :

1. un intervalle $a..b$ et deux réels la valeur et le pas (par défaut la valeur vaut $(a+b)/2$ et le pas $(b-a)/100$), par exemple, on tape dans la vue symbolique :
   `GC:=element(-pi..pi)` ou
   `GC:=element(-pi..pi,pi/2)` ou
   `GC:=element(-pi..pi,pi/2,pi/100.0)`
   cela signifie que `GC` peut prendre une valeur quelconque de $[-\pi;\pi]$ et le deuxième argument $\pi/2$ donne la valeur qui définit `GC` au début et $\pi/100.0$ est le pas choisi.
   Puis dans la vue graphique :
   – on a en bas et à droite on a `Pointeur x1,y1` où `x1,y1` sont les coordonnées du pointeur. On tape `C` en mode `Alpha`,
   – en bas et à droite on a alors `Pointeur GC` on appuie sur `Enter` pour valider.
   – maintenant on pointe sur `C`.
   Un trait apparait en haut de l'écran graphique : c'est un curseur qui permet de changer la valeur de `GC` avec les flèches de direction ($\leftarrow$ et $\rightarrow$) et en bas et à droite on a `Déplacer GC` À gauche de ce trait on a `GC=xc` (`xc` est la valeur de `GC`). Les flèches de direction $\leftarrow$ et $\rightarrow$ permettent de changer la valeur `xc` de `GC`.
   Exemple.
   On définit `GC` comme ci-dessus :
   On tape dans la vue symbolique :
   `GC:=element(-pi..pi,pi/2)`
   `GD:=line(y+x*TAN(GC)-2*SIN(GC)=0)`
   `GD` est donc une droite de paramètre `GC`.
   Quand on fait bouger le curseur `GC`, la doite `GD` bouge.
   On peut conserver la trace de cette droite `GD` en tapant dans la vue symbolique : `trace(GD)` ou en utilisant dans le menu de la vue graphique `Point->Plus->trace` qui permet de choisir le nom de l'objet dont on veut la trace et ainsi `trace(GD)` s'écrit automatiquement dans la vue symbolique.
   On peut ainsi voir que l'enveloppe de ces droites est une astroide.
   Remarque
   Pour effacer la trace ou pour l'arrêter, il faut utiliser le menu `Point->Plus`.
2. un objet géométrique et un réel (par défaut ce réel vaut $1/2$), par exemple :
   `GA:=element(circle(0,2),1)` signifie que `A` se trouve sur le cercle de centre 0 et de rayon 2 et a comme affixe $2*\exp(i)$ (car $2*\exp(i*t)$ est l'équation paramétrique de ce cercle et le deuxième argument 1 donne la valeur du paramètre t pour définir `GA`).
   Par exemple, `GA:=element(circle(0,1))` signifie que `A` se trouve sur le cercle de centre 0 et de rayon 1, le point `A` sera tracé en prenant $t = 1/2$ comme valeur du paramètre de l'équation paramétrique de l'objet géométrique (ici `affix(GA)` = $2*\exp(i/2)$). Lorsque ensuite on déplacera `A` avec les flèches, `A` se déplacera sur l'objet géométrique.
   Attention c'est la projetion du curseur sur le cercle qui définit le point `A` : il faut donc veiller à déplacer le curseur dans la fenêtre graphique de façon à ce qu'il définisse un point `A`.
3. un objet géométrique et un nom de variable (par exemple `GC`) défini aupara-

vant par la commande `element` : par exemple `GC:=element(0..pi)`.
Si on tape `GD:=element(circle(0,2),GC)`, alors `GC` est la variable de paramétrage de l'obget géométrique défini par le premier argument de `element`, c'est à dire que `GD` se trouve sur le cercle de centre 0 et de rayon 2 et que `GD` a comme affixe $2 * \exp(i * GC)$, car $2 * \exp(i * t)$ est l'équation paramétrique du circle(0,2). Il est donc obligatoire dans ce cas de définir auparavant le deuxième argument (ici `GC`) comme étant l'élément d'un intervalle.
On tape par exemple :

```
GC:=element(0..pi)
```

puis

```
GD:=element(circle(0,2),GC)
```

Puis on pointe le curseur sur `GC` (`Pointeur GC`) puis `Enter` cela a pour effet d'avoir en haut un curseur noté `GC` que l'on peut faire bouger avec les flèches de direction ($\leftarrow$ et $\rightarrow$) de 0 à π, avec à gauche de ce curseur un nombre égal à la valeur de du curseur. Ce curseur permet de faire bouger le point `A` sur le demi-cercle supérieur du cercle de centre 0 et de rayon 1 (car $0 \leq t \leq \pi$) et cela sans tracer ce demi-cercle. On tape par exemple :

```
GA:=point(1);GB:=point(2+i)
GC:=element(0..2)
```

puis

```
GD:=element(line(GA,GB),GC)
```

`D` est un point de la droite `AB` et on a `M=A+t*(B-A)` i.e. `M=(1-t)*A+t*B` pout parcourir le segment `AB` , il faut mettre `GC:=element(0..1)` ou encore `GD:=element(segment(GA,GB),GC)` qui aura pour effet de laisser `D` en `A` si `t<0` et de laisser `D` en `B` si `t>1`.

4. une ligne polygonale `GP` et `[floor(GC),frac(GC)]` avec `GC` défini auparavant par la commande `element` : par exemple `GC:=element(0..5)` si `GP` a 5 côtés.
   Les côtés de la ligne polygonale `GP` ont comme numèro : 0,1....
   Si par exemple `GP` a 5 côtés et a pour sommets `S(0),...S(4),S(5)=S(0)`, on tapera :
   `GC:=element(0..5)`
   `GD:=element(GP,[floor(GC),frac(GC)])`
   Ainsi selon les valeurs de `GC`, `D` va parcourir les 5 côtés de `GP` : `D` sera situé sur le côté de numéro `n=floor(GC)` et on aura :
   `D=frac(GC)*S(n)+(1-frac(GC))*S(n+1)`.
   Par exemple :

```
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)]);
```

Attention Si à un point M d'affixe m, défini comme élément d'une courbe C, on ajoute un complexe a, cela définit un point N de la courbe C qui est le projeté du point d'affixe $m+a$ sur C.
Par contre si un point M d'affixe m, défini comme élément d'une courbe C, on ajoute un point A d'affixe a, cela définit un point P d'affixe $m+a$ Par exemple, étant donné 3 points M, A, B, si on veut définir le point `N` vérifiant par exemple : $\overrightarrow{MN} = \overrightarrow{AB}$, on peut taper : `GN:=GM+(GB-GA)` à condition que M ne soit pas défini comme élément d'une courbe C. En effet si on a tapé `GM:=element(GC)` il faut définir N en tapant : `GN:=affix(GM)+GB-GA` ou `GN:=GM+GB-GA` (sans parenthèses) car `GN:=GM+GB-GA` est interprété en `GN:=(GM+GB)-GA` car il n'y a pas de règle de priorité entre + et − alors que
`GN:=GM+(GB-GA)` renvoie un élément de la courbe C qui est le projeté de N sur C.
On a donc, si on tape :
`GA:=point(-2,2);GB:=point(1,3);GC:=circle(0,1);`
`GM:=element(GC);GN:=affix(GM)+GB-GA`;(ou `GN:=GM+GB-GA`;)
`GN` n'est pas sur la courbe `C`
mais si on tape :
`GP:=GM+(GB-GA)` (ou GP :=projection(GC,GN) ;) `P` est sur la courbe `C`.

18.2.8 Point divisant un segment : `division_point`

`division_point` a trois arguments : deux points (ou deux nombres complexes a et b) et un nombre complexe k.
`division_point` renvoie et dessine le point d'affixe z tel que :
$\frac{z-a}{z-b} = k$ On tape :

```
GA:=division_point(i,2+i,3+i)
```

On obtient :

```
le point A d'affixe (5+4*i)/(2+i)
```

car `csolve(z-i=(3+i)*(z-2-i),z)` renvoie `[(14+3*i)/5]` et `(5+4*i)/(2+i)` renvoie `(14+3*i)/5`
On tape :

```
GB:=division_point(point(i),point(2+i),3)
```

On obtient :

```
le point B d'affixe 3+i
```

car `csolve(z-i=3*(z-2-i),z)` renvoie `[3+i]`

18.2.9 Le centre d'un cercle : `center`

`center` a comme argument le nom d'un cercle (voir la définition du cercle **??**).
`center` renvoie et trace le centre de ce cercle.
On tape :

```
GC:=center(circle(0,point(2*i)))
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté C
```

On tape :

```
GM:=center(circle(point(1+i),1))
```

On obtient :

```
Le point d'affixe 1+i est tracé avec une croix et est
noté M
```

18.3 Line

18.3.1 Droite définie par un point et une pente : `DrawSlp`

`DrawSlp(a,b,m)` dessine la droite de pente `m` `passant par le point (a,b)`
On tape :

```
DrawSlp(1,2,-1)
```

On obtient :

```
La droite passant par le point d'affixe 1+2i et de
pente -1
```

18.3.2 Tangente au graphe de $y = f(x)$ en $x = a$: `LineTan`

`LineTan(f(x),x=a)` trace la tangente au graphe de $y = f(x)$ en $x = a$.
On tape :

```
LineTan(sin(x),pi/6)
```

Ou on tape :

```
LineTan(sin(t),t,pi/6)
```

On tape :

```
LineTan(sin(t),t=pi/6)
```

On obtient :

Le tracé de la tangente à la courbe $y = \sin(x)$ au point d'abscisse $x = \pi/6$)

18.3.3 Hauteur d'un triangle : `altitude`

`altitude(GA,GB,GC)` trace la hauteur du triangle ABC issue de A.
On tape :

```
altitude(1,0,1-i)
```

On obtient dans l'Application de géométrie :

Le tracé de la hauteur du triangle $(1, 0, 1-i)$ issue du point d'affixe 1

On tape :

```
altitude(0,1,2-i)
```

On obtient dans l'Application de géométrie :

Le tracé de la hauteur du triangle $(0, 1, 2-i)$ issue du point d'affixe 0

On tape dans l'écran `CAS` :

```
a:=altitude(1,0,1-i)
```

On obtient :

```
line(y=x-1)
```

On tape dans l'écran `CAS` :

```
a:=altitude(1,0,1-i)
```

On obtient :

```
line(y=x)
```

18.3.4 Bissectrice intérieure d'un angle : `bisector`

`bisector(GA,GB,GC)` trace la bissectrice intérieure de l'angle $\widehat{BAC}$.
On tape :

```
bisector(0,1,i)
```

On obtient dans l'Application de géométrie :

Le tracé de la bissectrice intérieure de l'angle $\widehat{(0, 1, i)}$

On tape dans l'écran `CAS` :

```
bisector(0,1,i)
```

On obtient :

```
line(y=x)
```

18.3.5 Bissectrice extérieur d'un angle : `exbisector`

`exbisector(GA,GB,GC)` trace la bissectrice intérieure de l'angle $\widehat{BAC}$.
On tape :

```
exbisector(0,1,i)
```

On obtient dans l'Application de géométrie :

`Le tracé de la bissectrice extérieure de l'angle` $\widehat{(0,1,i)}$

On tape dans l'écran `CAS` :

```
exbisector(0,1,i)
```

On obtient :

```
line(y=-x)
```

18.3.6 Demi-droite : `half_line`

`half_line(GA,GB)` trace la demi droite A, B.
On tape :

```
half_line(1,2+i)
```

On obtient dans l'Application de géométrie :

```
Le tracé de la demi-droite d'origine le point d'affixe
        1 et contenant le point d'affixe 2+i.
```

On tape dans l'écran `CAS` :

```
half_line(1,2+i)
```

On obtient :

```
line(y=x-1)
```

18.3.7 La droite et la droite orientée : `line`

en géométrie plane, a comme argument deux points (ou deux nombres complexes représentant l'affixe de ces points) ou une liste de deux points (ou de deux complexes) ou a comme argument un point et `slope=m` ou encore une équation de droite de la forme a*x+by+c=0.
`droite` renvoie et trace la droite définie par les deux arguments.
`line(GA,GB)` trace la droite A, B.
Remarque`slope` est aussi une commande donnant la pente d'une droite. Il est préférable d'utiliser `DrawSlp` pour définir une droite avec un point et sa pente (`DrawSlp(a,b,m)` définit la droite `line(point(a,b),slope=m)`).
Dans l'écran `CAS` :
On tape :

```
line(1,2+i)
```

On obtient :

```
line(y=x-1)
```

Dans l'Application de géométrie :
On tape :

```
line(1,2+i)
```

On obtient :

```
Le tracé de la droite passant par le point d'affixe 1
            et par le point d'affixe 2+i.
```

On tape :

```
line(0,1+i)
```

On obtient :

```
La droite d'équation y=x est tracée
```

On tape :

```
line(i,slope=2)
```

Ou on tape :

```
DrawSlp(0,1,2)
```

On obtient :

```
La droite d'équation y=2x+1 est tracée
```

On tape :

```
line(y-x=0)
```

On obtient :

```
La droite d'équation y=x est tracée
```

Remarque : l'orientation de la droite

- Lorsque la droite est donnée par deux points, son orientation est définie par l'ordre des points donnés en argument. Par exemple `ine(GA,GB)` définit une droite orientée par le vecteur $\overrightarrow{AB}$.

- Lorsque c'est une équation qui définit la droite on écrit l'équation sous la forme :"membre_de_gauche-membre_de_droite=0" pour avoir une équation de la droite de la forme `a*x+by+c=0` et alors le vecteur orientant la droite est `[b,-a]` ou encore son orientation est définie par le produit vectoriel 3-d de son vecteur normal (de cote 0) et de [0,0,1]. Par exemple `line(y=2*x)` est orientée par `[1,2]` car son équation est `-2*x+y=0` et `cross([-2,1,0],[0,0,1])=[1,2,0]`.

– Lorsque la droite est donnée par un point A et sa pente m, son orientation est définie par le vecteur $\overrightarrow{AB}$ avec $B = A + 1 + i * m$.

Attention
Si on veut tracer la droite en couleur en utilisant un argument supplémentaire par exemple, `display=1`, il faut impérativement que cet argument soit le troisième argument : donc si la droite est définie par une liste il faut transformer cette liste en une séquence avec `op`, par exemple, on tape :

```
GA:=point(0,0);
GB:=point(2,1);
GC:=circle(A,B-A);
GD:=circle(B,A-B);
line(op(inter(GC,GD)),display=1)
```

alors que `line(inter(C1,C2),color=1)` vous répondra `"Invalid dimension"`

18.3.8 Médiane d'un triangle : `median_line`

`median_line(GA,GB,GC)` trace la médiane du triangle ABC issue de A.
On tape :

```
median_line(0,1,2+i)
```

On obtient dans l'Application de géométrie :

```
Le tracé de la droite passant par le point d'affixe 0
 et par le point d'affixe (3+i)/2 (milieu du segment
                    (0,2+i))
```

On tape dans l'écran `CAS` :

```
median_line(0,1,2+i)
```

On obtient :

```
line(y=x/3)
```

18.3.9 Droites parallèles : `parallel`

`parallel(GA,GD)` trace la parallèle à la droite D passant par A.
On tape :

```
parallel(0, line(1,i))
```

On obtient dans l'Application de géométrie :

```
le tracé de la droite d'équation y=-x
```

On tape dans l'écran `CAS` :

```
parallel(0, line(1,i))
```

On obtient :

```
line(y=(-x))
```

18.3.10 Médiatrice : `perpen_bisector`

`perpen_bisector(GA,GB)` trace la médiatrice du segment AB On tape :

```
perpen_bisector(1,i)
```

On obtient dans l'Application de géométrie :

```
le tracé de la droite d'équation y=x
```

On tape dans l'écran `CAS` :

```
perpen_bisector(1,i)
```

On obtient :

```
line(y=x)
```

18.3.11 Perpendiculaire à une droite : `perpendicular`

`perpendicular(GA,GB,GC)` ou `perpendicular(GA,line(GB,GC))` trace la perpendiculaire à la droite BC droite passant par le point A.
On tape :

```
perpendicular(1,1,2-i)
```

On obtient dans l'Application de géométrie :

```
le tracé de la perpendiculaire à la droite (1,2-i)
etpassant par le point d'affixe 1
```

On tape dans l'écran `CAS` :

```
perpendicular(1,1,2-i)
```

On obtient :

```
line(y=(x-1))
```

18.3.12 Segment : `segment`

`segment(GA,GB)` trace le segment AB.
On tape :

```
segment(0,1+i)
```

On obtient dans l'Application de géométrie :

```
Le tracé du segment (0,1+i)
```

On tape dans l'écran `CAS` :

```
segment(0,1+i)
```

On obtient :

```
segment(point(0),point(1+i))
```

18.3.13 Les tangentes à un objet géométrique ou la tangente en un point d'un graphe : `tangent`

`tangent` a deux arguments : un objet géométrique et un point `A`.

- l'objet géométrique est le graphe `G` d'une fonction 2-d
  Dans ce cas, le deuxième argument peut être soit, un nombre réel x_0, soit un point `A` situé sur `G`.
  Par exemple si on a défini la fonction `g`, on tape :
  ```
  GG:=plotfunc(g(x),x)
  tangent(GG, 1.2)
  ```
  trace la tangente au graphe `G` de la fonction `g` au point d'abscisse `x=1.2`,
  ou on tape :
  ```
  GA:=point(1.2+i*g(1.2))
  tangent(GG, GA)
  ```
  trace la tangente au point `A` du graphe `G` de la fonction `g`.
  Par exemple, pour avoir le tracé de la tangente au graphe de $g(x) = x^2$ au point d'abscisse $x_0 = 1$, on tape dans le CAS :
  ```
  g(x):=x^2
  ```
  Puis dans `Symb`, on tape :
  ```
  GG:=plotfunc(g(x),x)
  GT:=tangent(G,1)
  ```
  ou on tape :
  ```
  GT:=tangent(G,point(1+i))
  ```
  On obtient
  ```
  La tangente au graphe de g(x)=x^2 au point 1+i
  ```
  L'équation de la tangente est alors obtenue en tapant dans `Num` :
  ```
  equation(GT)
  ```
- l'objet géométrique n'est pas un graphe
  `tangent` peut avoir comme arguments :
  - un objet géométrique G et un point A ou,
  - un point A défini par `element` dont les paramètres sont, un objet géométrique G et un réel représentant la valeur du paramètre de l'équation paramétrique de G.
  `tangent` renvoie une liste de droites et dessine ces droites qui sont les tangentes à cet objet géométrique G et qui passent par le point A.
  On tape :
  ```
  tangent(circle(0,1),point(1+i))
  ```
  On obtient :
  ```
  La droite d'équation x=1 et la droite d'équation
  y=1
  ```
  On tape :
  ```
  tangent(element(circle(0,1),1))
  ```
  On obtient :
  ```
  La tangente au cercle de centre 0 et de rayon 1,
  au point d'affixe exp(i)
  ```
  On tape :
  ```
  tangent(circle(i,1+i),point((1+i*sqrt(3))*2))
  ```
  On obtient :

```
2 tangentes au cercle de centre i et de rayon √2
issues du point((1+i*sqrt(3))*2)
```

18.4 Polygon

18.4.1 Triangle équilatèral : `equilateral_triangle`

`equilateral_triangle`, en géométrie plane, a deux ou trois arguments :

- 2 arguments
  Ces 2 arguments sont : 2 points ou 2 nombres complexes représentant l'affixe de ces points (ou encore une liste de 2 points ou de 2 complexes).
  `equilateral_triangle(GA,GB)` renvoie et trace le triangle équilatéral direct ABC mais sans définir le point C. On tape :
  `equilateral_triangle(0,2)`
  On obtient :
  `le triangle équilatèral de sommets les points d'affixe 0,2,1+i*sqrt(3)`
  Pour définir le troisième sommet C, on peut donner un nom au triangle (par exemple `GT:=equilateral_triangle(0,2)`) et utiliser la commande `sommets(GT)` qui renvoie la liste des sommets de T. On définira alors `GC:= sommets(GT)[2]` mais il est plus simple de rajouter GC le nom du dernier sommet comme troisième argument.

- 3 arguments
  Ces 3 arguments sont les 2 arguments précédents et le 3ième argument est le nom d'une variable qui servira à définir et à tracer le troisième sommet avec sa légende. On tape :
  `equilateral_triangle(0,2,GC)`
  On obtient :
  `le triangle équilatèral de sommets les points d'affixe 0,2,1+i*sqrt(3)`
  On tape :
  `normal(affix(GC))`
  On obtient :
  `1+i*sqrt(3)`

18.4.2 Triangle rectangle : `right_triangle`

`right_triangle`, en géométrie plane, a trois ou quatre arguments :

- 3 arguments,
  ces 3 arguments sont : 2 points A et B (ou 2 nombres complexes représentant l'affixe de ces points) et un réel k non nul.
  `right_triangle(GA,GB,k)` renvoie et trace le triangle ABC rectangle en A : ce triangle est direct si $k > 0$, indirect si $k < 0$ et est tel que $AC = |k| * AB$.
  Ainsi si l'angle $(\overrightarrow{BC}, \overrightarrow{BA}) = \beta$ radians (ou degrés), on a $\tan(\beta) = k$.
  On remarquera que si C est le transformé de B dans la similitude de centre

A de rapport $|k|$ et d'angle $(k/|k|) * \pi/2$.
On tape :

```
right_triangle(i,-i,2)
```

On obtient :

```
Le triangle rectangle de sommets i, -i, 4+i
```

On tape :

```
right_triangle(i,-i,-2)
```

On obtient :

```
Le triangle rectangle de sommets i, -i, -4+i
```

– 4 arguments
ces 4 arguments sont les 3 arguments précédents et un dernier argument qui est le nom d'une variable qui servira à définir le troisième sommet.
On tape :

```
right_triangle(i,-i,2,GD)
```

On obtient :

```
Le triangle rectangle de sommets i, -i, 4+i
```

On tape :

```
normal(affix(GD))
```

On obtient :

```
4+i
```

18.4.3 Losange : `rhombus`

`rhombus`, en géométrie plane, peut avoir de trois à cinq arguments :

– 3 arguments,
ces 3 arguments sont : 2 points ou 2 nombres complexes représentant l'affixe de ces points et un nombre réel a.
`rhombus(GA,GB,a)` renvoie et trace le losange $ABCD$ tel que :
$\overrightarrow{AB}, \overrightarrow{AD}) = a$ radians (ou degrés), mais sans définir les points C et D.
On tape :

```
rhombus(-2*i,sqrt(3)-i,pi/3)
```

On obtient si on a coché `radian` dans la configuration du `CAS` ou de (`Shift-CAS` ou `Shift-`)

```
Le losange de sommets -2*i, √3-i, √3+i, 0
```

– 4 (resp 5) arguments,
ces 4 (resp 5) arguments sont les 3 arguments précédents, le dernier paramètre (resp les 2 derniers paramètres) est (resp sont) le (resp les) nom(s) d'une (resp deux) variable(s) qui servent à définir l'avant-dernier sommet (resp les 2 derniers sommets).
On tape :

```
rhombus(-2*i,sqrt(3)-i,pi/3,E,F)
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

```
Le losange de sommets -2*i, √3-i, √3+i, 0
```

On tape :

```
normal(affix(E))
```

On obtient :

```
sqrt(3)+i
```

On tape :

```
normal(affix(F))
```

On obtient :

```
0
```

18.4.4 `rectangle`

`rectangle`, en géométrie plane, peut avoir de trois à cinq arguments :

- 3 arguments,
  ces trois arguments sont : deux points (ou deux nombres complexes représentant l'affixe de ces points) et un nombre réel k non nul.
  `rectangle(GA,GB,k)` renvoie et trace le rectangle $ABCD$ tel que :
  $AD = |k| * AB$ et $(\overrightarrow{AB}, \overrightarrow{AD}) = (k/|k|) * \pi/2$,
  c'est à dire tel que :
  $affix(GD) = affix(GA) + k * \exp(i * \pi/2) * (affix(GB) - affix(GA))$
  mais sans définir les points C et D.
  Remarque Si k est complexe, on a :
  $affix(GD) = affix(GA) + k * \exp(i * \pi/2) * (affix(GB) - affix(GA))$
  et on peut ainsi se retrouver avec le tracé d'un parallélogramme.
  On tape :
  ```
  rectangle(0,1+i,1/2)
  ```
  On obtient :
  ```
  Le rectangle de sommets 0,1+i,1/2+3*i/2,-1/2+i/2
  ```
  On tape :
  ```
  rectangle(0,1+i,-1/2)
  ```
  On obtient :
  ```
  Le rectangle de sommets 0,1+i,3/2+i/2,1/2-i/2
  ```
  On tape :
  ```
  rectangle(0,1,1+i)
  ```
  On obtient :
  ```
  Le parallélogramme de sommets 0,1,i,-1+i car
  ```
  $$-1 + i = (1 + i) * \exp(i * \pi/2)$$
- 5 arguments, ces cinq arguments sont les 3 arguments précédents et les 2 derniers arguments sont les noms de deux variables qui serviront à définir les 2 derniers sommets.
  On tape :
  ```
  rectangle(0,1+i,-1/2,GG,GH)
  ```
  On obtient :
  ```
  Le rectangle de sommets 0,1+i,3/2+i/2,1/2-i/2
  ```
  On tape :
  ```
  normal(affix(GG))
  ```
  On obtient :
  ```
  (3+i)/2
  ```
  On tape :
  ```
  normal(affix(GH))
  ```
  On obtient :
  ```
  (1-i)/2
  ```

18.4.5 `square`

`square`, en géométrie plane, peut avoir de deux à quatre arguments :

- 2 arguments
  Ces 2 arguments sont : 2 points ou 2 nombres complexes représentant l'affixe de ces points (ou encore une liste de 2 points ou de 2 complexes).
  `squareGA,GB)` renvoie et trace le carrè $ABCD$ de sens direct, mais sans définir les points D et C.
  On tape :
  ```
  square(0,1+i)
  ```
  On obtient :
  ```
  Le carré de sommets 0, 1+i, 2*i, -1+i
  ```
- 3 (resp 4) arguments
  Ces 3 (resp 4) arguments sont les 2 arguments orécédents suivi par le le nom d'une (resp deux) variable(s) qui serviront à définir l'avant-dernier sommet (resp les deux autres sommets). On tape :
  ```
  square(0,1+i,GC,GD)
  ```
  On obtient :
  ```
  Le carré de sommets 0, 1+i, 2*i, -1+i
  ```
  On tape :
  ```
  affix(GC)
  ```
  On obtient :
  ```
  2*i
  ```
  On tape :
  ```
  affix(GD)
  ```
  On obtient :
  ```
  -1+i
  ```

18.4.6 `quadrilateral`

`quadrilateral(GA,GB,GC,GD)`, en géométrie plane, renvoie et trace le quadrilatère $ABCD$.
On tape :

```
quadrilateral(0,1,1+i,-1+2*i)
```

On obtient :

```
Le "cerf-volant" de sommets 0, 1, 1+i, 1+2*i
```

18.4.7 `parallelogram`

`parallelogram`, en géométrie plane, a trois arguments ou quatre arguments :

- 3 arguments,
  Ces 3 arguments sont : trois points (ou trois nombres complexes représentant l'affixe de ces points).
  `parallelogram(GA,GB,GC)` renvoie et trace le parallélogramme $ABCD$ tel que : $\overrightarrow{AD} = \overrightarrow{BC}$ mais sans définir le point D.
  On tape :
  ```
  parallelogram(0,1,2+i)
  ```

On obtient :

```
Le parallélogramme de sommets 0,1,2+i,1+i
```

On tape :

```
parallelogram(1,0,-1+i)
```

On obtient :

```
Le parallélogramme de sommets 1,0,-1+i,i
```

– 4 arguments,
Ces 4 arguments sont les 3 arguments précédent et le 4ième argument est le nom d'une variable qui servira à définir le sommet manquant.
On tape :

```
parallelogram(0,1,2+i,GF)
```

On obtient :

```
Le parallélogramme de sommets 0,1,2+i,1+i et le
point F d'affixe 1+i
```

On tape :

```
normal(affix(GF))
```

On obtient :

```
1+i
```

18.4.8 isoploygon

`isoploygon`, en géométrie plane, a trois arguments.
Les argument sont :
- soit deux points ou deux nombres complexes et un nombre entier positif k
- soit deux points ou deux nombres complexes et un nombre entier négatif k.
Lorsque $k > 0$, `isopolygone` trace le polygone régulier direct ayant k cotés et comme sommets consécutifs les deux premiers arguments.
On tape :

```
isopolygone(0,1,4)
```

On obtient :

```
Le carré de sommets 0,1,1+i,i
```

Lorsque $k < 0$, `isopolygone` trace le polygone régulier direct ayant $-k$ cotés, comme centre le premier argument, et comme sommet le deuxième argument.
On tape :

```
isopolygone(0,1,-4)
```

On obtient :

```
carré de sommets 1,i,-1,-i
```

18.4.9 isosceles_triangle

`isosceles_triangle`, en géométrie plane, a trois ou quatre arguments :

– 3 arguments
Ces 3 arguments sont : 2 points A et B (ou 2 nombres complexes représentant l'affixe de ces points) et un réel qui désigne la mesure en radians (ou en degrés) de l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$.
`isosceles_triangle(GA,GB,c)` renvoie et trace le triangle ABC isocèle de sommet A ($AB = AC$) et tel que l'angle $(\overrightarrow{AB}, \overrightarrow{AC}) = c$ radians (ou degrés), sans définir le point C).
On tape :

```
isosceles_triangle(i,1,-3*pi/4)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

`Le triangle isocéle de sommets -1, i, -`$\sqrt{2}$`+i`

– 4 arguments
Ces 3 arguments sont les 3 arguments précédents et le 4ième argument est le nom d'une variable qui servira à définir le troisième sommet.
On tape :

```
isosceles_triangle(i,1,-3*pi/4,GC)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

`Le triangle isocéle de sommets -1, i, -`$\sqrt{2}$`+i`

On tape :

```
normal(affix(GC))
```

On obtient :

```
-sqrt(2)+i
```

18.5 Courbes

18.5.1 Le cercle et ses arcs : `circle`

`circle` a un ou deux arguments pour dessiner un cercle et de quatre à six arguments pour dessiner un arc de cercle :

– Avec un argument :
l'argument de `cercle` est alors l'équation du cercle ayant comme variables x et y,
– Avec deux arguments :
le premier argument de `cercle` est un point ou un nombre complexe considéré comme l'affixe d'un point.
le deuxième argument détermine si on trace le cercle avec la donnée de son centre et de son rayon (le deuxième argument est alors un nombre complexe de module le rayon) ou avec la donnée de son diamètre (le deuxième argument est un point).
Ainsi :

  – `circle(GC,r)` où `GC` est un point (ou un nombre complexe) et `r` un nombre complexe, trace le cercle de centre `C` et de rayon le module de `r`. Cela est utile, par exemple, pour avoir le cercle de centre `A` qui passe par `B` on tape `circle(GA,GB-GA)`.

– `circle(GA,GB)` où `A` est un point ou un nombre complexe et `B` un point, trace le cercle de diamètre `AB`.

On tape :

```
circle(x^2+y^2-2*x-2*y)
```

On obtient :

```
Le cercle de centre 1+i et de rayon sqrt(2) est
tracé
```

On tape :

```
circle(-1,i)
```

On obtient :

```
Le cercle de centre -1 et de rayon 1 est tracé
```

On tape :

```
circle(-1, point(i))
```

On obtient :

```
Le cercle de diamétre -1,i
```

– Avec de quatre à six arguments :
`cercle` désigne un arc de cercle. Dans ce cas les deux premiers arguments déterminent le cercle qui porte l'arc (voir ci-dessus) et les deux arguments suivants sont les angles au centre des points qui délimitent l'arc et les deux derniers arguments sont des noms de variables qui contiendront les points qui délimitent l'arc. Le troisième et le quatrième argument sont les mesures des angles au centre des points qui délimitent l'arc, ces angles sont mesurés en radians (ou en degrés) à partir de l'axe défini par les deux premiers arguments si le deuxième argument est un point (définition du cercle par son diamètre) ou de l'axe défini par son centre C et le point $A = C + r$ si le deuxième argument est un complexe égal à r (définition du cercle par son centre et un complexe dont le module est égal au rayon).
Le cinquième et le sixième argument ne sont pas obligatoires et servent à définir les extrémités de l'arc.
On tape :

```
circle(-1,1,0,pi/4,A,B)
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

L'arc AB (GA:=point(0) et GB:=point($\frac{-1+\sqrt{2}+i*\sqrt{2}}{2}$)) du cercle de centre -1 et de rayon 1 est tracé

En effet l'angle est compté à partir de l'axe (-1,0) et donc l'angle 0 est le point(0).
On tape :

```
circle(-1,i,0,pi/4,A,B)
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

L'arc AB (GA:=point(-1+i) et GB:=point($\frac{-1-\sqrt{2}+i*\sqrt{2}}{2}$)) du cercle de centre -1 et de rayon 1 est tracé

En effet, l'angle est compté à partir de l'axe (-1,i-1) et donc l'angle 0 est le point d'affixe i-1.
On tape :

```
circle(-1, point(i),0,pi/4,A,B)
```

On obtient :

L'arc AB (GA:=point(i) et GB:=point($\frac{-1+i*(1+\sqrt{2})}{2}$)) du

```
cercle de diamétre -1,i
```

En effet, l'angle est compté à partir de l'axe (-1,i) et donc l'angle 0 est le point d'affixe i.

18.5.2 Le cercle circonscrit : `circumcircle`

`circumcircle` a trois paramètres qui définissent les sommets d'un triangle.
`circonscrit` dessine et renvoie le cercle circonscrit de ce triangle.
On tape :

```
circumcircle(-1,i,1+i)
```

On obtient :

```
Le cercle circonscrit du triangle(-1,i,1+i)
```

18.5.3 Tracé d'une conique : `conic`

`conic` a comme argument l'expression d'une conique.
`conique` trace la conique ayant pour équation l'argument égalé à zéro.
On tape :

```
conique(2*x^2+2*x*y+2*y^2+6*x)
```

On obtient :

```
le tracé de l'ellipse de centre -2+i et d'équation
2*x^2+2*x*y+2*y^2+6*x=0
```

Remarque :
Utiliser `reduced_conic` pour avoir l'équation paramétrique de la conique : On tape :

```
reduced_conic(2*x^2+2*x*y+2*y^2+6*x)[4]
```

On obtient :

```
[-2+i+(1+i)*(cos(t)+sqrt(3)*i*sin(t)),t,0,2π,2π/60]
```

18.5.4 L'ellipse : `ellipse`

`ellipse`, en géométrie plane, a 1 ou 3 paramètres :

- un paramètre :
  son équation de variables x et y. `ellipse(p(x,y))` trace la conique d'équation `p(x,y)=0` si `p(x,y)` est un polynôme de degré 2.
- trois paramètres : ses deux foyers et un de ces points (ou son affixe si cette affixe n'est pas réelle) ou ses deux foyers et un réel (son demi-grand axe).
  `ellipse(GF1,GF2,GA)` trace l'ellipse passant par `A` et de foyers `F1` et `F2` ou,
  `ellipse(GF1,GF2,a)` où `a` est un nombre réel, trace l'ellipse de foyers `F1` et `F2` et de demi-grand axe `|a|`.

On tape :

```
ellipse(-i,i,1+i)
```

On obtient :

```
L'ellipse de foyers -i, i et passant par 1+i
```

On tape :

```
ellipse(-i,i,sqrt(5)-1)
```

On obtient :

```
L'ellipse de foyers -i, i et de demi-grand axe
sqrt(5)-1
```

On tape :

```
ellipse(x^2+2*y^2-1)
```

ou on tape :

```
ellipse(sqrt(2)/2,-sqrt(2)/2,1)
```

On obtient :

```
L'ellipse de centre 0 et de demi-grand axe 1 et de
foyers sqrt(2)/2 et -sqrt(2)/2
```

18.5.5 Le cercle exinscrit : `excircle`

`excircle` a trois paramètres qui définissent les sommets d'un triangle.
`exinscrit` dessine et renvoie le cercle exinscrit dans l'angle intérieur du premier sommet de ce triangle.
On tape :

```
excircle(-1,i,1+i)
```

On obtient :

```
Le cercle exinscrit dans l'angle de sommet -1 du
triangle(-1,i,1+i) est tracé
```

18.5.6 L'hyperbole : `hyperbola`

`hyperbola`, en géométrie plane, a 1 ou 3 paramètres :

- un paramètre :
  son équation de variables x et y. `hyperbole(p(x,y))` trace la conique d'équation `p(x,y)=0` si `p(x,y)` est un polynôme de degré 2.
- trois paramètres :
  ses deux foyers et un de ces points (ou son affixe si cette affixe n'est pas réelle) ou ses deux foyers et un réel (son demi-grand axe).
  `hyperbole(GF1,GF2,GA)` trace l'hyperbole passant par `A` et de foyers `F1` et `F2` ou,
  `hyperbole(GF1,GF2,a)` où `a` est un nombre réel, trace l'hyperbole de foyers `F1` et `F2` et de demi-grand axe `|a|`.

On tape :

```
hyperbola(-i,i,1+i)
```

On obtient :

```
L'hyperbole de foyers -i, i et passant par 1+i
```

On tape :

```
hyperbola(-i,i,1/2)
```

On obtient :

```
L'hyperbole de foyers -i, i et de demi-grand axe 1/2
```

On tape :

```
hyperbole(x^2+2*y^2-1)
```

ou on tape :

```
hyperbole(sqrt(6)/2,-sqrt(6)/2,1)
```

On obtient :

```
L'hyperbole de centre 0 et de demi-grand axe 1 et de
foyers sqrt(6)/2 et -sqrt(6)/2
```

18.5.7 Le cercle inscrit : `incircle`

`incircle` a trois paramètres qui définissent les sommets d'un triangle.
`inscrit` dessine et renvoie le cercle inscrit de ce triangle.
On tape :

```
incircle(-1,i,1+i)
```

On obtient :

```
Le cercle inscrit du triangle(-1,i,1+i)
```

18.5.8 Lieu et enveloppe : `locus`

`locus` permet de tracer le lieu d'un point qui dépend d'un autre point qui doit être défini avec la fonction `element`.
`locus` permet aussi de tracer l'enveloppe d'une droite qui dépend d'un point qui doit être défini avec la fonction `element`

- lieu d'un point.
  `locus` a 2 à 4 arguments.
  Les deux premiers argument sont deux noms de variables :
  le premier argument est le nom du point (par exemple `B`) dont on veut connaitre le lieu et ce point est fonction du deuxième argument,
  le deuxième argument est le nom du point (par exemple `A`) qui se déplace sur une courbe `C` et qui doit être défini par `GA:=element(GC)`.
  On peut préciser éventuellement en troisième argument l'intervalle où se

trouve le paramètre utilisé pour le paramétrage de `C` lorsque le deuxième argument décrit `C` et en quatrième argument préciser la valeur de `tstep`.

Remarque

Pour connaitre le paramétrage de la courbe `C` on utilise la commande `parameq(C)`.
`locus` dessine le lieu du premier argument quand le deuxième argument se déplace selon ce que l'on a spécifié comme argument de `element`.

Conseils

Il faut avoir le moins possible d'instructions entre la définition de M et l'instruction `locus`.

On tape, pour avoir le lieu du centre de gravité B du triangle de sommets point(-1), point(1) et A lorsque A décrit la droite d'équation $y = 1$:

```
GA:=element(line(i,1+i))
GB:=isobarycenter(-1,1,GA)
GC:=locus(GB,GA)
```

On obtient :

```
La droite parallèle à l'axe des x passant par i/3
```

On tape dans la vue numérique :

```
equation(GC)
```

On obtient :

```
equation(GC):y=1/3
```

– enveloppe d'une droite fonction d'un point mobile sur une courbe.
`locus` a comme arguments deux noms de variables : le premier argument est le nom de la droite dont on veut connaitre l'enveloppe et cette droite est fonction du deuxième argument. Le deuxième argument est le nom du point qui se déplace et qui doit être défini avec la fonction `element`.
`locus` dessine l'enveloppe du premier argument quand le deuxième argument se déplace selon ce que l'on a spécifié comme argument de `element`.
On tape, pour avoir le l'enveloppe de la médiatrice de FH lorsque H décrit la droite d'équation $x = 0$:

```
GF:=point(1)
GH:=element(line(x=0))
GD:=perpend_bisector(GF,GH)
locus(GD,GH)
```

On obtient :

```
La parabole de foyer F et de directrice l'axe des
y dont l'èquation est 2*x-y^2-1=0
```

– enveloppe d'une droite donnée par une équation dépendant d'un paramètre,
Dans ce cas, il faut dire que la paramètre est l'affixe d'un point de la droite $y = 0$.
Par exemple, enveloppe d'une famille de droites d'équation $y + x\tan(t) - 2\sin(t) = 0$ lorsque $t \in \mathbb{R}$. (cf 1)
On tape :

```
GH:=element(line(y=0));
GD:=line(y+x*tan(affix(M))-2*sin(affix(M)))
locus(GD,GH)
```

On obtient :

```
L'astroide d'équation paramérique
2*cos(t)^3+2*i*sin(t)^3
```

Si on veut l'enveloppe lorsque $t = 0..\pi$, on tape :

```
locus(GD,GH,t=0..pi)
```

On obtient :

La partie au dessus de $y = 0$ de l'astroide d'équation paramérique 2*cos(t)^3+2*i*sin(t)^3

On peut aussi chercher l'intersection de GD et de GE (voir leur définition ci-dessous) pour avoir l'équation paramétrique du lieu.

```
GD:=y+x*tan(t)-2*sin(t)
GE:=diff(GD,t)
GM:=linsolve([GD=0,GE=0],[x,y])
GP:=plotparam(affix(GM))
```

On obtient :

L'astroide d'équation paramérique 2*cos(t)^3+2*i*sin(t)^3

en effet `simplify(GM)` renvoie :
`[2*cos(t)^3,2*sin(t)^3]`

18.5.9 La parabole : `parabola`

`parabola`, en géométrie plane, a 1 ou 2 paramètres :

- un paramètre :
  son équation de variables x et y. `parabole(p(x,y))` trace la conique d'équation `p(x,y)=0` si `p(x,y)` est un polynôme de degré 2.
- deux paramètres :
  deux points (ou leurs affixes si la deuxième affixe n'est pas réelle), représentant son foyer et son sommet ou encore un point (le sommet) ou l'affixe de son sommet et un nombre réel `c`.
  `parabole(GF,GS)` renvoie et dessine la parabole de foyer `F` et de sommet `S`.
  `parabole(GS,c)` renvoie et dessine la parabole de sommet `S`=$x_s + iy_s$ et d'équation $y = y_S + c * (x - x_S)^2$. Il faut savoir que si p est le paramètre de la parabole, on a $FS = p/2$ et $c = 1/(2 * p)$.

On tape :

```
parabola(0,i)
```

On obtient :

La parabole de foyer 0 et de sommet i

On tape :

```
parabola(0,1)
```

On obtient :

La parabole de sommet 0 et déquation $y = x^2$

On tape :

```
parabola(x^2-y-1)
```

ou on tape :

```
parabola(-i,1)
```

ou on tape :

```
parabola(i,-i)
```

On obtient :

```
La parabole de sommet -i et de foyers i
```

18.6 Transformation

18.6.1 L'homothétie : `homothety`

`homothety` en géométrie plane, a deux ou trois arguments : un point (le centre de l'homothétie), un réel (la valeur du rapport de l'homothétie) et éventuellement l'objet géométrique à transformer.
Lorsque `homothetie` a deux arguments, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
h:=homothety(i,2)
```

Puis :

```
h(1+i)
```

On obtient :

```
Le point 2+i tracé avec une croix (x) noire
```

Lorsque `homothety` a trois arguments, `homothety` dessine et renvoie le transformé du troisième argument dans l'homothétie de centre le premier argument et de rapport le deuxième argument.
On tape :

```
homothety(i,2,1+i)
```

On obtient :

```
Le point 2+i tracé avec une croix (x) noire
```

On tape :

```
homothety(i,2,circle(1+i,1))
```

On obtient :

```
Le cercle de centre 2+i et de rayon 2
```

Remarque
Lorsque la valeur du rapport de l'homothétie est un nombre complexe k non réel `homothety(GA,k)` est la similitude de centre le point `A`, de rapport `abs(k)` et d'angle `arg(k)`.

18.6.2 L'inversion : `inversion`

`inversion`, en géométrie plane, a deux ou trois arguments : un point (le centre de l'inversion), un réel (la valeur du rapport de l'inversion) et éventuellement l'objet géométrique à transformer.
Lorsque `inversion` a deux arguments, c'est une fonction qui agit sur un objet géométrique.
Si `GF:=inversion(GC,k)` et `GB:=GF(GA)`, on a $\overline{CA} * \overline{CB} = k$.
On tape :

```
GF:=inversion(i,2)
```

Puis :

```
GF(circle(1+i,1))
```

On obtient :

```
La droite verticale d'équation x=1
```

On tape :

```
GF(circle(1+i,1/2))
```

On obtient :

```
Le cercle de centre 8/3+i et de rayon 4/3 (il passe
par le point 4+i)
```

Lorsque `inversion` a trois arguments, `inversion` dessine et renvoie le transformé du troisième argument dans l'inversion de centre le premier argument et de rapport le deuxième argument.
Si `A1:=inversion(C,k,A)` on a $\overline{CA} * \overline{CA1} = k$.
On tape :

```
inversion(i,2,circle(1+i,1))
```

On obtient :

```
La droite verticale d'équation x=1
```

On tape :

```
inversion(i,2,circle(1+i,1/2))
```

On obtient :

```
Le cercle de centre 8/3+i et de rayon 4/3, il passe
par le point 4+i
```

18.6.3 La projection orthogonale : `projection`

`projection`, en géométrie plane, a un ou deux arguments : un objet géométrique et éventuellement un point.
Lorsque `projection` a un argument, c'est une fonction qui agit sur un point et qui projette orthogonalement ce point sur l'objet géométrique.
On tape :

```
p1:=projection(line(-1,i))
```

Puis :

```
p1(1+i)
```

On obtient :

```
Le point 1/2+3/2*i apparait avec une croix (x) noire
```

On tape :

```
p2:=projection(circle(-1,1))
```

```
p2(i)
```

On obtient :

```
Le point d'affixe, sqrt(2)/2+(i)*sqrt(2)/2-1, apparait
avec une croix (x) noire
```

Lorsque `projection` a deux arguments, `projection` dessine et renvoie le transformé du point donné en 2ième argument par la projection orthogonale sur le premier argument.
On tape :

```
projection(line(-1,i),1+i)
```

On obtient :

```
Le point 1/2+3/2*i apparait avec une croix (x) noire
```

On tape :

```
projection(circle(-1,1),i)
```

On obtient :

```
Le point d'affixe, -1+sqrt(2)/2+(i)*sqrt(2)/2,
apparait avec une croix (x) noire
```

18.6.4 La symétrie droite et la symétrie point : `reflection`

`reflection`, en géométrie plane, a un ou deux arguments : un point ou une droite et éventuellement l'objet géométrique à transformer.
Lorsque `reflection` a un argument, c'est une fonction qui agit sur un objet géométrique : quand le premier argument est un point (ou un nombre complexe) il s'agit de la symétrie par rapport à ce point (ou par rapport au point d'affixe ce nombre complexe) et quand le premier argument est une droite il s'agit de la symétrie par rapport à cette droite.
On tape :

```
sp:=reflection(-1)
```

Puis :

```
sp(1+i)
```

On obtient :

```
Le point -3-i tracé avec une croix (x) noire
```

On tape :

```
sd:=reflection(line(-1,i))
```

Puis :

```
sd(1+i)
```

On obtient :

```
Le point 2*i tracé avec une croix (x) noire
```

Lorsque `reflection` a deux arguments, `reflection` dessine et renvoie le transformé du deuxième argument dans la symétrie définie par le premier argument : quand le premier argument est un point (ou un nombre complexe) il s'agit de la symétrie par rapport à ce point (ou par rapport au point d'affixe ce nombre complexe) et quand le premier argument est une droite il s'agit de la symétrie par rapport à cette droite.
On tape :

```
reflection(-1,1+i)
```

On obtient :

```
Le point -3-i tracé avec une croix (x) noire
```

On tape :

```
reflection(line(-1,i),1+i)
```

On obtient :

```
Le point 2*i tracé avec une croix (x) noire
```

18.6.5 La rotation : `rotation`

`rotation`, en géométrie plane, a deux ou trois arguments.
Lorsque `rotation` a deux arguments ce sont : un point (le centre de rotation) et un réel (la mesure de l'angle de rotation) ; `rotation` est alors une fonction qui agit sur un objet géométrique (point, droite etc...)
On tape :

```
r:=rotation(i,-pi/2)
```

Puis :

```
r(1+i)
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

```
Le point 0 tracé avec une croix (x) noire
```

Lorsque `rotation` a trois arguments, ce sont : un point (le centre de rotation), un réel (la mesure de l'angle de rotation) et l'objet géométrique à transformer ; `rotation` dessine et renvoie alors le transformé du troisième argument dans la rotation de centre le premier argument et d'angle de mesure le deuxième argument.
On tape :

```
rotation(i,-pi/2,1+i)
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

```
Le point 0 tracé avec une croix (x) noire
```

On tape :

```
rotation(i,-pi/2,line(1+i,-1))
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

```
La droite passant par 0 et -1+2*i
```

18.6.6 La similitude : `similarity`

`similarity`, en géométrie plane, a trois ou quatre arguments : un point (le centre de rotation), un réel (la valeur du rapport k de la similitude), un réel (la mesure a de l'angle de rotation en radians (ou degrés)) et éventuellement l'objet géométrique à transformer.
Remarque : si le rapport k est négatif, l'angle de la similitude est alors de mesure $-a$ radians (ou degrés).
Lorsque `similarity` a trois arguments, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
GS:=similarity(i,2,-pi/2)
```

Puis :

```
GS(1+i)
```

On obtient si on a choisi `radian` dans la configuration du `CAS` :

```
Le point -i tracé avec une croix (x) noire
```

On tape :

```
GS(circle(1+i,1))
```

On obtient si on a choisi `radian` dans la configuration du `CAS` :

```
Le cercle de centre -i et de rayon 2
```

Lorsque `similarity` a quatre arguments, `similarity` dessine et renvoie le transformé du quatrième argument dans la similitude de centre le premier argument de rapport le deuxième argument et d'angle le troisième argument.
On tape :

```
similarity(i,2,-pi/2,1+i)
```

On obtient si on a choisi `radian` dans la configuration du `CAS` :

```
Le point -i tracé avec une croix (x) noire
```

On tape :

```
similarity(i,2,-pi/2,circle(1+i,1))
```

On obtient si on a choisi `radian` dans la configuration du `CAS` :

```
Le cercle de centre -i et de rayon 2
```

Remarque
En 2d la similitude de centre le point `A`, de rapport `k` et d'angle `a` se traduit par : `similarity(GA,k,a)` ou par `homothety(GA,k*exp(i*a))`.

18.6.7 La translation : `translation`

`translation`, en géométrie plane, a un ou deux arguments : le vecteur de translation donné par un vecteur géométrique ou par la liste de ses coordonnées ou par son affixe (la différence de deux points ou un nombre complexe) et éventuellement l'objet géométrique à transformer.
Lorsque `translation` a un argument, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
t:=translation(1+i)
```

Puis :

```
t(-2)
```

On obtient :

```
Le point -1+i tracé avec une croix (x) noire
```

Lorsque `translation` a deux arguments, `translation` dessine et renvoie le transformé du deuxième argument dans la translation de vecteur le premier argument.
On tape :

```
translation([1,1],-2)
```

Ou on tape :

```
A:=point(1);B:=point(2+i);translation(vecteur(A,B),-2)
```

Ou on tape :

```
translation(1+i,-2)
```

Ou on tape :

```
A:=point(1);B:=point(2+i);translation(B-A,-2)
```

On obtient :

```
Le point -1+i tracé avec une croix (x) noire
```

On tape :

```
translation(1+i,line(-2,-i))
```

On obtient :

```
La droite passant par -1+i et 1
```

18.7 Mesure et graphique

18.7.1 Affichage de la mesure d'un angle : `angleat`

`angleat` a comme argument le nom de 3 points et 1 point (ou 1 nombre complexe qui est l'affixe de ce point).
Attention Il faut que les 3 premiers arguments soient des noms.
`angleat` renvoie le 4-ième point, calcule la mesure (en radians ou en degrés) de l'angle orienté de sommet le premier argument, le deuxième argument se trouve sur le premier coté de l'angle et le troisième argument se trouve sur le deuxiéme coté et cette mesure est affichée, précédée d'une légende, à l'endroit du 4-ième point.
Ainsi `anglat(GA,GB,GC,GD)` désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB}, \overrightarrow{AC})$ et cette messue sera affichée, pécédée de αA=, à l'endroit du point `D`.
On tape cette commande dans La vue symbolique.
On tape :

```
GA:=point(-1);GB:=point(1+i);GC:=point(i);
segment(GA,GB);segment(GA,GC);
angleat(GA,GB,GC,0.2i)
```

On obtient si on a coché `radian` dans la configuration du `CAS` (`Shift-CAS`) :

```
αA=atan(1/3) s'écrit au point(0.4i)
```

18.7.2 Affichage de la mesure d'un angle : `angleatraw`

`angleatraw` a comme argument 4 points (ou 4 nombres complexes qui sont l'affixe de ces points).
`angleatraw` renvoie le 4-ième point, calcule la mesure (en radians ou en degrés) de l'angle orienté de sommet le premier argument, le deuxième point se trouve sur le premier coté de l'angle et le troisième point se trouve sur le deuxiéme coté et cette mesure est affichée à l'endroit du 4-ième point.
Ainsi `angleatraw(GA,GB,GC,GD)` désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB}, \overrightarrow{AC})$ et cette mesure sera affichée à l'endroit du point `D`.
On tape :

```
GA:=point(-1);GB:=point(1+i);GC:=point(i);

segment(GA,GB);segment(GA,GC);

angleatraw(GA,GB,GC,0.2i)
```

On obtient si on a coché `radian` dans la configuration du `CAS` (`Shift-CAS`) :

```
atan(1/3) s'écrit au point(0.4i)
```

18.7.3 Affichage de l'aire d'un polygone : `areaat`

`areaat` a comme arguments le nom d'un cercle ou d'un polygone étoilé par rapport à son premier sommet et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`areaat` renvoie le point, calcule l'aire du cercle ou polygone étoilé par rapport à son premier sommet et affiche cette aire à l'endroit du point avec une légende.
Attention Il faut que le premier argument soit le nom d'un cercle ou d'un polygone.
On tape :

```
t:=triangle(0,1,i)

areaat(t,(1+i)/2)
```

On obtient :

```
1/2 s'écrit au point(1+i)/2 avec la légende
```

On tape :

```
cc:=circle(0,2)

areaat(cc,2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2) avec une légende
```

On tape :

```
c:=square(0,2)
areaat(c,2.2)
```

On obtient :

```
4 s'écrit au point(2.2) avec une légende
```

On tape :

```
h:=hexagon(0,1)
areaat(h,1.2)
```

On obtient :

```
3*sqrt(3)/2 s'écrit au point(1.2) avec une légende
```

18.7.4 Affichage de l'aire d'un polygone : `areaatraw`

`areaatraw` a comme arguments un cercle ou un polygone étoilé par rapport à son premier sommet et 1 point (ou 1 nombre complexe qui est l'affixe de ce point). `areaatraw` renvoie le point, calcule l'aire du cercle ou du polygone étoilé par rapport à son premier sommet et affiche cette aire à l'endroit du point.
On tape :

```
areaatraw(triangle(0,1,i),(1+i)/2)
```

On obtient :

```
1/2 s'écrit au point(1+i)/2
```

On tape :

```
areaatraw(circle(0,2),2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2)
```

On tape :

```
areaatraw(square(0,2),2.2)
```

On obtient :

```
4 s'écrit au point(2.2)
```

On tape :

```
areaatraw(hexagon(0,1),1.2)
```

On obtient :

```
3*sqrt(3)/2 s'écrit au point(1.2)
```

18.7.5 Affichage de la longueur d'un segment : `distanceat`

`distanceat` est une commande qui permet d'afficher en un point la longueur d'un segment avec une légende.
On tape cette commande dans La vue symbolique.
`distanceat` a 3 arguments : le nom de 2 points et 1 point (ou l'affixe de ce point) ou encore le nom de 2 objets géométriques et un point (ou l'affixe de ce point).
Attention Il faut que les 2 premiers arguments soient des noms de point.
`distanceat` renvoie le point donné en 3-ième argument, calcule la longueur du segment défini par les deux premiers points ou la distance entre les 2 objets géométriques et affiche à l'endroit du 3-ième point, cette longueur précédée d'une légende.
On tape (on doit donner le nom des objets) :

```
GA:=point(-1);GB:=point(1+i);

distanceat(GA,GB,0.4i)
```

On obtient :

```
"GAB=sqrt(5)" s'écrit au point(0.4i)
```

On tape (on doit donner le nom des objets) :

```
GC:=point(0);GD:=line(-1,1+i)

distanceat(GC,GD,i/2)
```

On obtient :

```
"GCD=sqrt(5)/5" s'écrit au point(i/2)
```

On tape (on doit donner le nom des objets) :

```
GK:=circle(0,1); GL:=line(-2,1+3i)

distanceat(GK,GL,0)
```

On obtient :

```
"GKL=sqrt(2)-1" s'écrit au point(0)
```

18.7.6 Affichage de la longueur d'un segment : `distanceatraw`

`distanceatraw` est une commande qui permet d'afficher en un point la longueur d'un segment mais sans légende.
On tape cette commande dans La vue symbolique.
`distanceatraw` a comme argument trois points (ou 2 points et 1 nombre complexe qui est l'affixe d'un point) ou encore 2 objets géométriques et un point (ou l'affixe de ce point).
`distanceatraw` renvoie le point donné en 3-ième argument, calcule la longueur du segment défini par les deux premiers points ou la distance entre les 2 objets géométriques et affiche cette longueur à l'endroit du 3-ième point.
On tape :

```
GA:=point(-1);GB:=point(1+i);
distanceatraw(GA,GB,0.4i)
```

Ou on tape directement :

```
distanceatraw(point(-1),point(1+i),0.4i)
```

On obtient :

```
sqrt(5) s'écrit au point(0.4i)
```

On tape :

```
GC:=point(0);GD:=line(-1,1+i)
distanceatraw(GC,GD,i/2)
```

Ou on tape directement :

```
distanceatraw(point(0),line(-1,1+i),0.4i)
```

On obtient :

```
sqrt(5)/5 s'écrit au point(i/2)
```

On tape :

```
GK:=circle(0,1); GL:=line(-2,1+3i)
distanceatraw(GK,GL,0)
```

Ou on tape directement :

```
distanceatraw(circle(0,1),line(-2,1+3i),0.4i)
```

On obtient :

```
sqrt(2)-1 s'écrit au point(0)
```

18.7.7 Affichage du périmètre d'un polygone :`perimeterat`

`perimeterat` a comme argument le nom d'un cercle ou d'un polygone et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`perimeterat` renvoie le point, calcule le périmètre du cercle ou du polygone et affiche ce périmètre à l'endroit du point avecune légende.
On tape cette commande dans La vue symbolique.
Attention Il faut que le premier argument soit le nom d'un cercle ou d'un polygone.
On tape :

```
t:=triangle(0,1,i)
perimeterat(t,(1+i)/2)
```

On obtient :

```
2+sqrt(2) s'écrit au point((1+i)/2) avec une légende
```

On tape :

```
c:=square(0,2)
perimeterat(c,2.2)
```

On obtient :

```
8 s'écrit au point(2.2) avec une légende
```

On tape :

```
cc:=circle(0,2)
perimeterat(cc,2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2) avec une légende
```

On tape :

```
h:=hexagon(0,1)
perimeterat(h,1.2)
```

On obtient :

```
6 s'écrit au point(1.2) avec une légende
```

18.7.8 Affichage du périmètre d'un polygone : `perimeteratraw`

`perimeteratraw` a comme argument un cercle un polygone et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`perimeteratraw` renvoie le point, calcule le périmètre du cercle ou du polygone et affiche ce périmètre à l'endroit du point.
On tape cette commande dans La vue symbolique.
On tape :

```
perimeteratraw(triangle(0,1,i),(1+i)/2)
```

On obtient :

```
2+sqrt(2) s'écrit au point((1+i)/2)
```

On tape :

```
perimeteratraw(circle(0,2),2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2)
```

On tape :

```
perimeteratraw(hexagon(0,1),1.2)
```

On obtient :

```
6 s'écrit au point(1.2)
```

On tape :

```
perimeteratraw(square(0,2),2.2)
```

On obtient :

```
8 s'écrit au point(2.2)
```

18.7.9 Affichage de la pente d'une droite : `slopeat`

`slopeat` est une commande qui permet d'afficher en un point la pente d'une droite ou d'un segment avec une légende.
On tape cette commande dans La vue symbolique.
`slopeat` a 2 arguments le nom d'une droite (ou d'un segment) et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`slopeat` renvoie le point, calcule la pente de la droite (ou du segment) et affiche cette pente à l'endroit du point avec une légende.
Attention Il faut que le premier argument soit le nom d'une droite ou d'un segment.
On tape :

```
GD:=line(1,2i)
```

Ou on tape :

```
GD:=segment(1,2i),i)
slopeat(GD,i)
```

On obtient :

```
"sD=-2" s'écrit au point(i)
```

On tape :

```
GP:=line(2y-x=3),2*i)
slopeat(GP,2*i)
```

On obtient :

```
"sP=1/2" s'écrit au point(2*i)
```

On tape :

```
GT:=tangent(plotfunc(sin(x)),pi/4)
```

Ou on tape :

```
GT:=LineTan(sin(x),pi/4)
```

Puis :

```
slopeat(GT,i)
```

On obtient :

```
"sT=(sqrt(2))/2" s'écrit au point(i)
```

18.7.10 Affichage de la pente d'une droite : `slopeatraw`

`slopeatraw` est une commande qui permet d'afficher en un point la pente d'une droite ou d'un segment mais sans légende.
On tape cette commande dans La vue symbolique.
`slopeatraw` a 2 arguments une droite (ou un segment) et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`slopeatraw` renvoie le point, calcule la pente de la droite (ou du segment) et affiche cette pente à l'endroit du point.
On tape :

```
GD:=line(1,2i)
slopeatraw(GD,i)
```

Ou on tape directement :

```
slopeatraw(line(1,2i),i)
```

On obtient :

```
-2 s'écrit au point(i)
```

On tape :

```
GE:=segment(1,2i),i)
slopeatraw(GE,1)
```

Ou on tape directement :

```
slopeatraw(segment(1,2i),1)
```

On obtient :

```
-2 s'écrit au point(1)
```

On tape :

```
GP:=line(2y-x=3,2*i)
slopeatraw(GP,2*i)
```

Ou on tape directement :

```
slopeatraw(line(2y-x=3,2*i),2*i)
```

On obtient :

```
1/2 s'écrit au point(2*i)
```

On tape :

```
GT:=tangent(plotfunc(sin(x)),pi/4)
slopeat(GT,i)
```

Ou on tape directement :

```
slopeatraw(tangent(plotfunc(sin(x)),pi/4),i)
```

On obtient :

```
(sqrt(2))/2 s'écrit au point(i)
```

18.8 Measure

18.8.1 L'abscisse d'un point ou d'un vecteur : `abscissa`

`abscissa`, en géométrie plane, est une fonction ayant comme argument un point ou un vecteur ou un nombre complexe.
`abscissa` renvoie l'abscisse du point ou du vecteur :
- si le point `A` a pour coordonnées cartésiennes (x_A, y_A), `abscisse(GA)` renvoie x_A,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B), `abscisse(GA-GB)` renvoie $x_A - x_B$ (car `GA-GB` désigne le vecteur $\overrightarrow{BA}$).
On tape :

```
abscissa(point(1+2*i))
```

On obtient :

```
1
```

On tape :

```
abscissa(point(i)-point(1+2*i))
```

On obtient :

```
-1
```

On tape :

```
abscissa(1+2*i)
```

On obtient :

```
1
```

On tape :

```
abscissa([1,2])
```

On obtient :

```
1
```

18.8.2 L'affixe d'un point ou d'un vecteur : `affix`

`affix` est une fonction qui a comme argument soit un point soit un vecteur soit les coordonnées d'un point ou d'un vecteur 2-d.
`affix` renvoie l'affixe du point ou du vecteur :
- si le point `A` a pour coordonnées cartésiennes (x_A, y_A),
  `affix(GA)` renvoie $x_A + i * y_A$,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B),
  `affix(GA-GB)` ou `affix(vector(GB,GA))` renvoie
  $x_A - x_B + i * (y_A - y_B)$
  (car `A-B` désigne le vecteur $\overrightarrow{BA}$) et `coordinates(vector(GB,GA))` renvoie $[x_A + i * y_A, x_B + i * y_B]$.

On tape :

```
affix(point(i))
```

On obtient :

```
i
```

On tape :

```
affix(point(i)-point(1+2*i))
```

On obtient :

```
-1-i
```

18.8.3 La mesure d'un angle : `angle`

`angle` a comme argument trois points (ou trois nombres complexes qui sont l'affixe de ces points) et éventuellement une chaine de caractères pour afficher un petit arc de cercle avec ce symbole en légende afin de représenter l'angle sur la figure (l'arc est remplacé par le symbole quart de carre si l'angle vaut pi/2 ou -pi/2).
`angle` renvoie la mesure (en radians ou en degrés) de l'angle orienté de sommet le premier argument, le deuxième argument se trouve sur le premier coté de l'angle et le troisième argument se trouve sur le deuxiéme coté.
Ainsi :
`angle(GA,GB,GC)` désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB}, \overrightarrow{AC})$.
`angle(GA,GB,GC,"")` trace l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ avec comme légende un petit arc orienté.
`angle(GA,GB,GC,"a")` trace l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ avec comme légende un petit arc orienté noté a.
`angle(GA,GB,GC,"")[0]` ou `angle(GA,GB,GC,"a")[0]` désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB}, \overrightarrow{AC})$.
On tape :

```
angle(0,1,1+i)
```

On obtient si on a choisi `radian` dans la configuration du `CAS` :

```
pi/4
```

On tape :

```
angle(0,1,1+i,"")
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

```
[pi/4,circle(point(0,0),1/5)]
```

et l'angle est repéré par un arc de cercle sans légende.
On tape :

```
angle(0,1,1+i,"a")
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

```
[pi/4,circle(point(0,0),1/5)]
```

et l'angle est repéré par un arc de cercle avec `a` comme légende.
On tape :

```
angle(0,1,i,"a")
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

```
[pi/2,polygon(point(1/5,0),point(1/5,1/5),point(0,1/5),point(0,1/5))]
```

et l'angle droit est repéré par un quart de carré avec `a` comme légende.

18.8.4 Longueur d'un arc de courbe : `arcLen`

`arcLen` a quatre paramètres : une expression $expr$ (resp une liste de 2 expressions $[expr1, expr2]$), le nom d'un paramètre et deux valeurs a et b de ce paramètre.
`arcLen` calcule la longueur de l'arc de courbe définie par l'équation $y = f(x) = expr$ (resp par $x = expr1, y = expr2$) pour les valeurs du paramètre comprises entre a et b.
On a donc `arcLen(f(x),x,a,b)=` :
`integrate(sqrt(diff(f(x),x)^2+1),x,a,b)`
ou
`integrate(sqrt(diff(x(t),t)^2+diff(y(t),t)^2),t,a,b)`.
Exemples

- Calculer la longueur de l'arc de parabole $y = x^2$ pour x allant de 0 à $x = 1$.
  On tape
  ```
  arcLen(x^2,x,0,1)
  ```
  ou
  ```
  arcLen([t,t^2],t,0,1)
  ```
  On obtient :
  ```
  -1/4*log(sqrt(5)-2)-(-(sqrt(5)))/2
  ```
- Calculer la longueur de l'arc de la courbe $y = \cosh(x)$ pour x allant de 0 à $x = \ln(2)$.
  On tape :
  ```
  arcLen(cosh(x),x,0,log(2))
  ```
  On obtient :
  ```
  3/4
  ```
- Calculer la longueur de l'arc de cercle $x = \cos(t), y = \sin(t)$ pour t allant de 0 à $t = 2 * \pi$.
  On tape
  ```
  arcLen([cos(t),sin(t)],t,0,2*pi)
  ```
  On obtient :
  ```
  2*pi
  ```

18.8.5 Aire d'un polygone : `area`

`area` calcule l'aire d'un cercle ou d'un polygone étoilé par rapport à son premier sommet.
On tape :

```
aire(triangle(0,1,i))
```

On obtient :

```
1/2
```

On tape :

```
aire(square(0,2))
```

On obtient :

```
4
```

18.8.6 Les coordonnées d'un point, d'un vecteur ou d'une droite : `coordinates`

`coordinates`, en géométrie plane, est une fonction ayant comme argument un point ou un nombre complexe ou un vecteur ou une droite.
`coordinates` renvoie la liste de l'abscisse et de l'ordonnée du point ou du vecteur ou la liste des affixes de deux points de la droite orientée.

- si le point `A` a pour coordonnées cartésiennes (x_A, y_A), `coordinates(GA)` renvoie $[x_A, y_A]$,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B), `coordinates(vector(GA,GB))` ou `coordinates(GB-GA)` renvoie $[x_B - x_A, y_B - y_A]$ (alors que `B-A` renvoie $(x_B - x_A) + i * (y_B - y_A)$ car `B-A` désigne l'affixe du vecteur $\overrightarrow{AB}$ en géométrie plane),
- si le vecteur `V` a pour coordonnées cartésiennes (x_V, y_V), `coordinates(GV)` ou `coordinates(vector(GA,GV))` renvoie $[x_V, y_V]$,
- si une droite `D` est définie par deux points `A` et `B`, `coordinates(GD)` renvoie `[affix(GA),affix(GB)]`, si `D` est définie par son équation, `coordinates(GD)` renvoie `[affix(GA),affix(GB)]` où `A` et `B` sont deux points de la droite `D`, le vecteur `AB` ayant même orientation que `d`.

On tape :

```
coordinates(point(1+2*i))
```

Ou on tape :

```
coordinates(1+2*i)
```

On obtient :

```
[1,2]
```

On tape :

```
coordinates(point(1+2*i)-point(i))
```

Ou on tape :

```
coordinates(point(1+2*i)-point(i))
```

On obtient :

```
[1,1]
```

On tape :

```
coordinates(vector(point(i),point(1+2*i)))
```

Ou on tape :

```
coordinates(vector(i,1+2*i))
```

Ou on tape :

```
coordinates(vector([0,1],[1,2]))
```

On obtient :

```
[1,1]
```

On tape :

```
coordinates(1+2*i)
```

Ou on tape :

```
coordinates(vector(1+2*i))
```

Ou on tape :

```
coordinates(vecteur(point(i),vecteur(1+2*i)))
```

On obtient :

```
[1,2]
```

On tape :

```
coordinates(point(i),vector(1+2*i))
```

On obtient :

```
[1,2]
```

On tape :

```
d:=line(-1+i,1+2*i)
```

Ou on tape

```
d:=line(point(-1,1),point(1,2))
```

Puis,

```
coordinates(d)
```

On obtient :

```
[-1+i,1+2*i]
```

On tape :

```
d:=line(y=(1/2*x+3/2))
```

On obtient :

```
[(3*i)/2,1+2*i]
```

On tape :

```
d:=line(x-2*y+3=0)
```

On obtient :

```
[(3*i)/2,(-4+i)/2]
```

Attention
`coordinates` peut aussi avoir comme argument une séquence ou une liste de points. `coordinates` renvoie alors la séquence ou la liste des listes des coordonnées de ces points, par exemple :
`coordinates(i,1+2*i)` ou `coordinates(point(i),point(1+2*i))`
renvoie la séquence :
`[0,1],[1,2]`
et
`coordinates([i,1+2*i])` ou `coordinates([point(i),point(1+2*i)])`
renvoie la matrice :
`[[0,1],[1,2]]` donc
`coordinates([1,2])` renvoie la matrice :
`[[1,0],[2,0]]` car `[1,2]` est considéré comme la liste de 2 points d'affixe `1` et `2`.

18.8.7 La longueur d'un segment et distance entre les deux objets géométriques : `distance`

`distance` a comme argument deux points (ou deux nombres complexes qui sont l'affixe de ces points) ou deux objets géométriques.
`distance` renvoie la longueur du segment défini par ces deux points ou la distance entre les deux objets géométriques.
On tape :

```
distance(-1,1+i)
```

On obtient :

```
sqrt(5)
```

On tape :

```
distance(0,line(-1,1+i))
```

On obtient :

```
sqrt(5)/5
```

On tape :

```
distance(circle(0,1),line(-2,1+3i))
```

On obtient :

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

18.8.8 Le carré de la longueur d'un segment : `distance2`

`distance2` a comme argument deux points (ou 2 points et 1 nombre complexe qui est l'affixe d'un point).
`distance2` renvoie le carré de la longueur du segment défini par ces deux points.
On tape :

```
distance2(-1,1+i)
```

On obtient :

```
5
```

18.8.9 L'équation cartésienne d'un objet géométrique : `equation`

`equation` permet d'avoir l'équation cartésienne d'un objet géométrique.
Attention ! ! ! il faut auparavant purger les variables `x` et `y` en tapant `purge(x)` et `purge(y)` ou `x:='x'` et `y:='y'`.
On tape :

```
equation(line(point(0,1,0),point(1,2,3)))
```

On obtient :

```
(x-y+1=0,3*x+3*y-2*z=0)
```

On tape :

```
equation(sphere(point(0,1,0),2))
```

On obtient :

```
x^2+y^2+-2*y+z^2-3=0
```

qui est l'équation de la sphère de centre (0,1,0) et de rayon `2`.

18.8.10 Avoir comme réponse la valeur d'une mesure affichée : `extract_measure`

`extract_measure` permet d'obtenir la valeur d'une mesure qui a été affichée.
`extract_measure` a pour argument la commande qui a affiché cette mesure.
On tape :

```
GA:=point(-1);GB:=point(1+i);GC:=segment(GA,GB)
```

```
extract_measure(distanceat(GA,GB,i))
```

On obtient :

```
sqrt(5)
```

On tape :

```
extract_measure(distanceatraw(GA,GB,i))
```

On obtient :

```
sqrt(5)
```

On tape :

```
extract_measure(slopeat(GC,i))
```

On obtient :

```
1/2
```

On tape :

```
extract_measure(slopeatraw(GC,i))
```

On obtient :

```
1/2
```

18.8.11 L'ordonnée d'un point ou d'un vecteur : `ordinate`

`ordinate`, en géométrie plane, est une fonction ayant comme argument un point ou un vecteur ou un nombre complexe.
`ordinate` renvoie l'ordonnée du point ou du vecteur :
- si le point `A` a pour coordonnées cartésiennes (x_A, y_A), `ordinate(GA)` renvoie y_A,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B), `ordinate(GA-GB)` renvoie $y_A - y_B$ (`A-B` désigne le vecteur $\overrightarrow{BA}$).
On tape :

```
ordinate(point(1+2*i))
```

On obtient :

```
2
```

On tape :

```
ordinate(point(i)-point(1+2*i))
```

On obtient :

```
-1
```

On tape :

```
ordinate(1+2*i)
```

On obtient :

```
2
```

On tape :

```
ordinate([1,2])
```

On obtient :

```
2
```

18.8.12 L'équation paramétrique d'un objet géométrique : `parameq`

`parameq`, en géométrie plane, permet d'avoir l'équation paramétrique d'un objet géométrique sous la forme du nombre complexe $x(t) + i * y(t)$.
Attention ! ! ! il faut auparavant purger la variable `t` en tapant : `purge(t)` ou `t:='t'`.
On tape :

```
parameq(line(-1,i))
```

On obtient :

```
-t+(1-t)*(i)
```

On tape :

```
parameq(circle(-1,i))
```

On obtient :

```
-1+exp(i*t)
```

On tape :

```
normal(parameq(ellipse(-1,1,i)))
```

On obtient :

```
sqrt(2)*cos(t)+(i)*sin(t)
```

18.8.13 Périmètre d'un polygone : `perimeter`

`perimeter` calcule le périmètre d'un cercle ou d'un polygone.
On tape :

```
perimeter(triangle(0,1,i))
```

On obtient :

```
2+sqrt(2)
```

On tape :

```
perimeter(square(0,2))
```

On obtient :

```
8
```

18.8.14 Le rayon d'un cercle : `radius`

`radius` a comme argument un cercle.
`radius` renvoie la longueur du rayon de ce cercle.
On tape :

```
radius(circle(-1,i))
```

On obtient :

```
1
```

On tape :

```
radius(circle(-1,point(i)))
```

On obtient :

```
sqrt(2)/2
```

18.8.15 Pente d'une droite : `slope`

`slope` est soit une commande soit un attribut de la commande `droite` pour cela voir 18.3.7
Lorsque `slope` est une commande son argument est une droite ou un segment ou deux points ou deux nombres complexes.
`slope` renvoie la pente de la droite définie par le segment ou par les 2 points ou leurs affixes.
On tape :

```
slope(line(1,2i))
```

Ou on tape :

```
slope(segment(1,2i))
```

Ou on tape :

```
slope(point(1),point(2i))
```

Ou on tape :

```
slope(1,2i)
```

On obtient :

```
-2
```

On tape :

```
slope(line(2y-x=3))
```

On obtient :

```
1/2
```

On tape :

```
slope(tangent(plotfunc(sin(x)),pi/4))
```

Ou on tape :

```
slope(LineTan(sin(x),pi/4))
```

On obtient :

```
(sqrt(2))/2
```

18.9 Test

18.9.1 Savoir si 3 points sont alignés : `is_colinear`

`is_colinear` est une fonction booléenne ayant comme argument une liste ou une séquence de points.
`is_colinear` vaut 1 si les points sont alignés, et vaut 0 sinon.
On tape :

```
is_colinear(0,1+i,-1-i)
```

On obtient :

```
1
```

On tape :

```
is_colinear(i/100,1+i,-1-i)
```

On obtient :

```
0
```

18.9.2 Savoir si 4 points sont cocycliques : `is_concyclic`

`is_concyclic` est une fonction booléenne ayant comme argument une liste ou une séquence de points.
`is_concyclic` vaut 1 si les points sont cocycliques, et vaut 0 sinon.
On tape :

```
is_concyclic(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
is_concyclic(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

18.9.3 Savoir si des éléments sont conjugués : `is_conjugate`

`is_conjugate` permet de savoir si 4 points sont conjugués ou si 2 points ou 2 droites ou 1 droite et 1 point sont conjugués par rapport à un cercle ou deux droites.
`is_conjugate` est une fonction booléenne ayant comme arguments deux points (resp deux droites ou un cercle) suivi de deux points ou de deux droites ou d'une droite et d'un point.
`is_conjugate` vaut 1 si les arguments sont conjugués, et vaut 0 sinon.
On tape :

```
is_conjugate(circle(0,1+i),point(1-i),point(3+i))
```

On obtient :

```
1
```

On tape :

```
is_conjugate(circle(0,1),point((1+i)/2),line(1+i,2))
```

Ou on tape :

```
is_conjugate(circle(0,1),line(1+i,2),point((1+i)/2))
```

On obtient :

```
1
```

On tape :

```
is_conjugate(circle(0,1),line(1+i,2), line((1+i)/2,0))
```

On obtient :

```
1
```

On tape :

```
is_conjugate(point(1+i),point(3+i),point(i),point(i+3/2))
```

On obtient :

```
1
```

On tape :

```
is_conjugate(line(0,1+i),line(2,3+i),
     line(3,4+i),line(3/2,5/2+i))
```

On obtient :

```
1
```

18.9.4 Savoir si des points ou /et des droites sont coplanaires : `is_coplanar`

`is_coplanar` teste si une liste ou une séquence de points ou de droites sont coplanaires.
On tape :

```
is_coplanar([0,0,0],[1,2,-3],[1,1,-2],[2,1,-3])
```

On obtient :

```
1
```

On tape :

```
is_coplanar([-1,2,0],[1,2,-3],[1,1,-2],[2,1,-3])
```

On obtient :

```
0
```

On tape :

```
is_coplanar([0,0,0],[1,2,-3],line([1,1,-2],[2,1,-3]))
```

On obtient :

```
1
```

On tape :

```
is_coplanar(line([0,0,0],[1,2,-3]),line([1,1,-2],[2,1,-3]))
```

On obtient :

```
1
```

On tape :

```
is_coplanar(line([-1,2,0],[1,2,-3]),
      line([1,1,-2],[2,1,-3]))
```

On obtient :

```
0
```

18.9.5 Savoir si 1 point est sur un objet graphique : `is_element`

`is_element` est une fonction booléenne ayant comme argument un point et un objet géométrique.
`is_element` vaut 1 si le point appartient à l'objet géométrique, et vaut 0 sinon.
On tape :

```
is_element(point(-1-i),line(0,1+i))
```

On obtient :

```
1
```

On tape :

```
is_element(point(i),line(0,1+i))
```

On obtient :

```
0
```

18.9.6 Savoir si on a un triangle équilatéral : `is_equilateral`

`is_equilateral` est une fonction booléenne ayant comme argument trois points ou un objet géométrique.
`is_equilateral` vaut 1 si les trois points forment un triangle équilatéral ou si l'objet géométrique est un triangle équilatéral, et vaut 0 sinon.
On tape :

```
is_equilateral(0,2,1+i*sqrt(3))
```

On obtient :

```
1
```

On tape :

```
GT:=equilateral_triangle(0,2,GC);is_equilateral(GT[0])
```

On obtient :

```
1
```

En effet `GT[0]` désigne un triangle car `GT` est une liste composée du triangle et de son sommet `GC`.
On tape `affix(GC)` et on obtient `1+i*sqrt(3)`
On tape :

```
is_equilateral(1+i,-1+i,-1-i)
```

On obtient :

```
0
```

18.9.7 Savoir si on a un triangle isocéle : `is_isosceles`

`is_isosceles` est une fonction booléenne ayant comme argument trois points ou un objet géométrique.
`is_isoceles` vaut 1 (resp 2, 3) si les trois points forment un triangle isocéle ou si l'objet géométrique est un triangle isocéle dont l'angle portant les deux cotés égaux est désigné par le premier (resp second, troisième) argument, ou vaut 4 si les trois points forment un triangle équilatéral, ou si l'objet géométrique est un triangle équilatéral, et vaut 0 sinon.
On tape :

```
is_isosceles(1,1+i,i)
```

On obtient :

```
2
```

On tape :

```
GT:=isosceles_triangle(0,1,pi/4);is_isoceles(GT)
```

On obtient :

```
1
```

On tape :

```
GT:=isosceles_triangle(0,1,pi/4,GC);is_isoceles(GT[0])
```

On obtient :

```
1
```

En effet `GT[0]` désigne un triangle car `GT` est une liste composée du triangle et de son sommet `C`.
On tape `affix(GC)` et on obtient `(sqrt(2))/2+((i)*sqrt(2))/2`
On tape :

```
is_isosceles(1+i,-1+i,-i)
```

On obtient :

```
3
```

18.9.8 Orthogonalité de 2 droites ou 2 sphères : `is_orthogonal`

`is_orthogonal` est une fonction booléenne ayant comme argument deux droites ou deux sphères ou une droite et un plan ou deux plans.
`is_orthogonal` vaut 1 si les deux droites ou les deux sphères (i.e si les plans tangents en leurs points d'intersection sont orthogonaux), ou la droite et le plan ou les deux plans sont orthogonaux et vaut 0 sinon.
On tape :

```
is_orthogonal(line([2,3,-2],[-1,-1,-1]),
              line([1,0,0],[1,2,8]))
```

On obtient :

```
1
```

On tape :

```
is_orthogonal(line([2,3,-2],[-1,-1,-1]),
  plane([-1,-1,-1],[-1,0,3],[-2,0,0]))
```

On obtient :

```
1
```

On tape :

```
is_orthogonal(plane([0,0,0],[1,2,-3],[1,1,-2]),
      plane([-1,-1,-1],[1,2,-3],[0,0,0]))
```

On obtient :

```
1
```

On tape :

```
is_orthogonal(sphere([0,0,0],sqrt(2)),sphere([2,0,0],sqrt(2)))
```

On obtient :

```
1
```

18.9.9 Savoir si 2 droites sont parallèles : `is_parallel`

`is_parallel`, en géométrie plane, est une fonction booléenne ayant comme argument deux droites.
`is_parallel` vaut 1 si les deux droites sont parallèles, et vaut 0 sinon.
On tape :

```
is_parallel(line(0,1+i),line(i,-1))
```

On obtient :

```
1
```

On tape :

```
is_parallel(line(0,1+i),line(i,-1-i))
```

On obtient :

```
0
```

18.9.10 Savoir si on a un parallélogramme : `is_parallelogram`

`is_parallelogram` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`is_parallelogram` vaut 1 (resp 2, 3, 4) si les quatre points forment un parallélogramme (resp un losange, un rectangle, un carré) ou si l'objet géométrique est un parallélogramme (resp un losange, un rectangle, un carré), et vaut 0 sinon.
On tape :

```
is_parallelogram(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

On tape :

```
is_parallelogram(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
GQ:=quadrilateral(1+i,-1+i,-1-i,1-i);is_parallelogram(GQ)
```

On obtient :

```
4
```

Attention
On doit taper :

```
GP:=parallelogram(-1-i,1-i,i,GD);is_parallelogram(GP[0])
```

Pour obtienir :

```
1
```

En effet c'est `GP[0]` qui désigne un parallélogramme car `GP` est une liste composée d'un parallélogramme et de son dernier sommet `D`.
Si on tape `affix(GD))`, on obtient `-2+i`.

18.9.11 Savoir si 2 droites sont perpendiculaire : `is_perpendicular`

`is_perpendicular`, en géométrie plane, est une fonction booléenne ayant comme argument deux droites.
`is_perpendicular` vaut 1 si les deux droites sont perpendiculaires, et vaut 0 sinon.
On tape :

```
is_perpendicular(line(0,1+i),line(i,1))
```

On obtient :

```
1
```

On tape :

```
is_perpendicular(line(0,1+i),line(1+i,1))
```

18.9.12 Savoir si on a un triangle rectangle ou si on a un rectangle : `is_rectangle`

`is_rectangle` est une fonction booléenne ayant comme argument trois ou quatre points ou un objet géométrique.
`is_rectangle` vaut 1 (resp 2 ou 3) si les trois points forment un triangle rectangle, l'angle droit étant désigné par le premier (resp second, troisième) argument ou si l'objet géométrique est un triangle rectangle,
`is_rectangle` vaut 1 (resp 2) si les quatre points forment un rectangle (resp un carré) ou si l'objet géométrique est un rectangle (resp un carré), et vaut 0 sinon.
On tape :

```
is_rectangle(1,1+i,i)
```

On obtient :

```
2
```

On tape :

```
is_rectangle(1+i,-2+i,-2-i,1-i)
```

On obtient :

```
1
```

On tape :

```
GR:=rectangle(-2-i,1-i,3,GC,GD);is_rectangle(GR[0])
```

On obtient :

```
1
```

En effet `GR[0]` désigne un rectangle car `GR` est une liste composée du rectangle et de ses sommets `C` et `D`.

18.9.13 Savoir si on a un losange : `is_rhombus`

`is_rhombus` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`is_rhombus` vaut 1 (rep 2) si les quatre points forment un losange (resp un carré) ou si l'objet géométrique est un losange (resp un carré), et vaut 0 sinon.
On tape :

```
is_rhombus(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
GK:=rhombus(1+i,-1+i,pi/4);is_rhombus(GK)
```

On obtient :

```
1
```

On tape :

```
GK:=rhombus(1+i,-1+i,pi/4,GC,DD);is_rhombus(GK[0])
```

On obtient :

```
1
```

En effet `GK[0]` désigne un losange car `GK` est une liste composée d'un losange et de ses sommets `GC` et `GD`.
Si on tape : `normal(coordinates(GC,GD))`,
on obtient `[-sqrt(2)-1,-sqrt(2)+1],[-sqrt(2)+1,-sqrt(2)+1]`.
On tape :

```
is_rhombus(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

18.9.14 Savoir si on a un carré : `is_square`

`is_square` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`is_square` vaut 1 si les quatre points forment un carré ou si l'objet géométrique est un carré, et vaut 0 sinon.
On tape :

```
is_square(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
GK:=square(1+i,-1+i);is_square(GK)
```

On obtient :

```
1
```

On tape :

```
GK:=square(1+i,-1+i,C,D);is_square(GK[0])
```

On obtient :

```
1
```

En effet `GK[0]` désigne un carré car `GK` est une liste composée d'un carré et de ses sommets `C` et `D`.
Si on tape `affix(GC,GD)`, on obtient `-1-i,1-i`.
On tape :

```
is_square(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

18.10 Exercices de géométrie

18.10.1 Les transformations

- `La translation` Un pavage :tout quadrilatère plan non croisé pave le plan.
  On définit 4 points `A,B,C,D` au hasard :
  menu `Points->Free points->4 random point`
  Dans `Symb` on change le nom des points de façon que le quadrilatère A, B, C, D ne soit pas croisé.
  Puis on définit le quadrilatère A, B, C, D avec le menu `Lines->Polygons->quadrilateral` et ce sera le pavé de base : `GE:=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])
```

– `L'inversion` L'inverseur de Peaucelier

```
GA:=element(-1.6..1.6,0.6)
GD:=circle(1,1,('display')=(black+hidden_name)
GE:=point(1+EXP(i*GA)
GH:=circle(GE,2.,('display')=(hidden_name)
GI:=circle(0,3.,('display')=(hidden_name)
GJ:=inter(GH,GI)
GK:=reflection(GJ,GE)
GL:=locus(GK,GA)
GG:=quadrilateral(GE,GJ[0],GK,GJ[1])
GB:=segent(0,GJ[0])
GC:=segent(0,GJ[1])
```

18.10.2 Les lieux

Soit un triangle direct OAB rectangle en O avec $OA = a$ et $OB = b$.
Soit $D = At$ une demi droite variable telle que :
$(\overrightarrow{OA}, \overrightarrow{At}) = c, 0 \leq c \leq \frac{\pi}{2}$.
Soient $A1$ et $B1$ les projections respectives de A et B sur D.
Quelle est la valeur de c pour laquelle $A1$ et $B1$ sont confondus en un point que l'on nommera P ? Trouver les lieux de $A1$ et de $B1$ quand c varie.
Montrer que le triangle $PA1B1$ reste semblable au triangle OAB quand c varie.
Trouver le lieu de M milieu de $A1B1$ quand c varie.

```
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)
```

18.11 Activités de géométrie

– Médiatrice de `AB`
  Créer un segment `AB`.

Construire la médiatrice de AB, en utilisant la même construction qu'avec un compas.
Réponse
On sélectionne :
`Lines->segment` et on définit avec le curseur deux points A et B et le segment C est définit automatiquement dans SYMB (`GC:=segment(GA,GB)`).
`perpen_bisector` trace directement la médiatrice de AB.
Pour faire la même construction qu'avec un compas :
On sélectionne : `Curves->Circles->circle`
On pointe le centre B (ou on tape `Alpha B`), on valide avec `ENTER` ou avec `,` puis on pointe le point A (ou on tape `Alpha A`) .
Il s'inscrit en dessous de la figure :
`circle(GB,GA-GB)` (c'est le cercle D de centre B passant par A) que l'on valide avec `ENTER` (GD :=circle(GB,GA-GB) est définit automatiquement dans SYMB).
Puis, on pointe le centre A puis le point B.
Il s'inscrit en dessous de la figure :
`circle(GA,GB-GA)` (c'est le cercle E de centre A passant par B) que l'on valide avec `ENTER` (GE :=circle(GA,GB-GA) est définit automatiquement dans SYMB).
On peut taper dans SYMB :
`GG:=line(inter(GD,GE))` pour tracer la droite joignant les deux points de l'intersection de GD et de GE (`inter(GD,GE)` est la liste des points de cette intersection).
Ou bien, on définit l'intersection avec `Point->inter` puis, en désignant le premier point d'intersection, puis le second point d'intersection.

– Milieu
Créer un segment `[AB]`.
Construire le milieu de AB, soit en utilisant les coordonnées, soit en utilisant la même construction qu'avec un compas.
On tape : `GI:=point(coordinates(GA)/2+coordinates(GB)/2)`
ou bien on rajoute à la construction de la médiatrice (cf ci-dessus) :
`GI:=single_inter(GC,GG)`
On peut aussi comme exercice de programmation définir la fonction `Milieu` (il faut commencer la fonction par une majuscule car `milieu` est une commande de `Xcas`).
On tape :
`Milieu(A,B):=point(coordinates(A)/2+coordinates(B)/2)`
ou encore si on a défini la fonction `Mediatrice` :
`Milieu(A,B):=single_inter(segment(A,B),Mediatrice(A,B))`

– Isobarycentre
Créer 4 points `A,B,C,D`.
Définir l'isobarycentre de `A,B,C,D`, en utilisant les coordonnées.
Réponse
On tape dans SYMB :
`GE:=point((coordonnees(GA)+coordonnees(GB)+coordonnees(GC)+coordonnees(GD))/4)`
On peut, comme exercice de programmation définir la fonction `ISOBAR`

(ATTENTION ! `isobarycentre` est une commande qui existe déjà).
On tape :

```
EXPORT ISOBAR(L)
BEGIN
LOCAL S,D;
D:=SIZE(L);
S:=1/d*sum(L[k],k,0,d-1);
RETURN S;
END
```

– Barycentre
Créer 4 points `A,B,C,D`.
Définir le barycentre de `[A,1],[B,-2],[C,1],[D,3]`, en utilisant les coordonnées.
Réponse
On tape :

```
GE:=point((coordonnees(GA)-2*coordonnees(GB)+coordonnees(GC)+
3*coordonnees(GD))/3)
```

On peut aussi comme exercice de programmation définir la fonction `BARY` (ATTENTION ! `barycenter` est une commande qui existe déjà) qui renvoie le barycentre des points $A, B...$ affectés des coefficients $\alpha, \beta....$
On tape on suppose que L est la liste $A, \alpha, B, \beta...$:

```
EXPORT BARY(L):={
BEGIN
LOCAL S,D;
D:=SIZE(L);
S:=sum(L[k],k,1,d-1,2);
IF (s==0) THEN RETURN "pas defini" END;
RETURN 1/s*sum(L[k]*L[k+1],k,0,d-2,2);
}
```

– Bissectrice d'un angle
Créer un triangle `ABC`.
Construire les bissectrices de l'angle A du triangle ABC, en utilisant la même construction qu'avec un compas et en utilisant l'instruction `perpen_bissector` qui trace la médiatrice d'un segment.
Réponse
On sélectionne :
`Lines->Triangle->triangle` et on définit avec le curseur trois points `A`, `B` et `C` et le triangle `D` ainsi que ses cotés sont définis automatiquement dans `SYMB` (`GD:=triangle(GA,GB,GC)`), `GE:=segment([GA,GB])`,`GG:=segment([`
`GH:=segment([GC,GA])`.
On sélectionne : `Curves->Circles->circle`
On suppose que $AB < AC$ pour que le cercle de centre A qui passe par B coupe le segment `H` qui est le segment AC.
On pointe le centre `A` (ou on tape `Alpha A`), on valide avec `ENTER` ou avec `,` puis on pointe le point `B` (ou on tape `Alpha B`) .
Il s'inscrit en dessous de la figure :
`circle(GA,GB-GA)` qui)=est le cercle `I` de centre `A` passant par `B`.

On sélectionne `Points->Dep.points->inter` et on montre le cercle `I`, puis le segment `H GJ:=inter(GI,GH)` définit l'intersection du cercle `I` avec le segment `H(` AC).
Puis on sélectionne `perpen_bissector` pour tracer la médiatrice de `BJ`
`GK:=perpen_bissector(GB,GJ)`
On peut aussi comme exercice de programmation définir les fonctions `Bissectrice` et `Exbissectrice` (Attention ! `bissector` et `exbissector` sont des commandes qui existent déjà).
On tape si on a défini la fonction `Mediatrice` :

```
Bissectrice(A,B,C):=perpen_bissector(single_inter(half_line(A,B),
circle(A,2)),single_inter(half_line(A,C),circle(A,2)))
Exbissectrice(A,B,C):={local C1:=A+(A-C);Bissectrice(A,B,C1)}
```

– Report d'une longueur
Étant donnés trois points `A B` et `C`, on veut construire un point `D` pour que $AD = BC$.
On utilise la commande `cercle` et on tape :
`D:= element(circle(A,B-C))`

L'instruction `longueur(B,C)` renvoie la longueur du segment BC (les unités étant définies sur la fenêtre graphique).
Si l'on veut reporter une longueur dans une direction donnée, on multiplie cette longueur par le vecteur unitaire de cette direction.
Exemple :
Étant donnés trois points `A B` et `C`, construire sur la demi-droite AB, un point D tel que $AD = AC$.
On tape :
`GD:=GA+distance(GA,GC)*(GB-GA)/distance(GA,GB)`
ou encore
`D:=single_inter(circle(A,C-A),half_line(A,B))`

– Report d'un angle
Étant donnés deux points `A` et `B`, on veut construire `C` pour que l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ soit de mesure donnée par exemple 72 degrés ou $2 * \pi/5$ radians.
On tape, si on a coché `radian` dans la fenêtre de configuration du `cas` :
`D:=rotation(A,2*pi/5,line(A,B))`
ou, si on est en degré (on n'a pas coché `radian`) :
`D:=rotation(A,72,line(A,B))`
puis on tape :
`C:=element(D)`
L'instruction `angle(A,B,C)` donne la mesure en radians (ou en degrés) de l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$, on peut donc vérifier la construction demandée.
Étant donné deux points `A` et `B`, on veut construire `C` pour que l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ soit égal à l'angle $(\overrightarrow{OM}, \overrightarrow{OP})$.
On tape :

```
D:=rotation(A,angle(O,M,P),line(A,B));
C:=element(D)
```

– Perpendiculaire à la droite BC passant par A
  Créer un point `A` et une droite `BC` ne passant pas par `A`.
  Construire la perpendiculaire à la droite BC passant par A, en utilisant la même construction qu'avec un compas.
  Réponse
  On sélectionne :
  `Points->point` et on définit avec le curseur le point `A`, puis on sélectionne `Lines->line` et on définit avec le curseur les points `B` et `C` et la droite `D` est définit automatiquement dans `SYMB` (`GD:=line(GB,GC)`).
  On sélectionne : `Curves->Circles->circle`
  On pointe le centre `B` (ou on tape `Alpha B`), on valide avec `ENTER` ou avec `,` puis on pointe le point `A` (ou on tape `Alpha A`) .
  Il s'inscrit en dessous de la figure :
  `circle(GB,GA-GB)` qui est le cercle `E` de centre `B` passant par `A`.
  On pointe le centre `C` (ou on tape `Alpha C`), on valide avec `ENTER` ou avec `,` puis on pointe le point `A` (ou on tape `Alpha A`) .
  Il s'inscrit en dessous de la figure :
  `circle(GC,GA-GC)` qui est le cercle `G` de centre `C` passant par `A`.
  On sélectionne `Points->Dep.points->inter` et on montre le premier point d'intersection des cercles `E` et `G`. Cela définit le point `H`, puis on montre leur second point d'intersection et cela définit le point `K`. On trace ensuite la droite `HK`.
– Perpendiculaire à la droite AB passant par A
  Construire la perpendiculaire à la droite AB passant par A en utilisant la même construction qu'avec un compas.
  Réponse
  On sélectionne :
  `Lines->line` et on définit avec le curseur les points `A` et `B` et la droite `C` est définie automatiquement dans `SYMB` (`GC:=line(GA,GB)`).
  On sélectionne : `Curves->Circles->circle`
  On pointe le centre `A` (ou on tape `Alpha A`), on valide avec `ENTER` ou avec `,` puis on pointe le point `B` (ou on tape `Alpha B`) .
  Il s'inscrit en dessous de la figure :
  `circle(GA,GB-GA)` qui est le cercle `D` de centre `A` passant par `B`.
  On sélectionne `Points->Dep.points->inter` et on montre le point d'intersection (autre que `B`) du cercle `D` avec la droite `C`. Cela définit le point `E`, Puis on trace la médiatrice de `BE`. `perpen_bissector(GB,GE)` race la mediatrice des deux points définit par `inter(C3,line(GA,GB))`, cette médiatrice passe par A et est perpendiculaire à AB.
  On peut aussi comme exercice de programmation définir la fonction `Perpendiculaire` (Attention `perpendicular` est une commande qui existe déjà).
  On suppose que `A` est un point et que `D` est une droite) :

```
EXPORT Perpend(A,D)
BEGIN
LOCAL E,M:=element(D);
M:=midpoint(op(inter(D,circle(A,E-A))));
RETURN line(A,M);
END
```

– Parallèle à une droite passant par A
Créer un point `A` et un segment `BC` ne passant pas par `A`. Construire la parallèle à la droite BC passant par A en utilisant l'instruction `perpendiculaire`.
Réponse
On clique avec la souris pour avoir un point `A` et deux points `B` et `C` et le segment `BC`.
On tape :
`D:=perpendicular(A,line(B,C))`
cela trace la perpendiculaire à BC passant par A
`P:=perpendicular(A,D)`
cela trace la perpendiculaire à D passant par A.
La liste des instructions se trouve dans `geo5` : créer trois points `A, B, C`, puis faire `Charger session` du menu `Fich` de `Xcas` et selectionner `geo5` du répértoire `examples/geo` pour exécuter ce fichier.

On peut, comme exercice de programmation définir la fonction `Parallele` (il faut commencer la fonction par une majuscule car `parallele` est une commande de `Xcas`).
On tape (`A` est un point et `d` est une droite) :
```
Parallele(A,d):={
local d1:=Perpendiculaire(A,d);
Perpendiculaire(A,d1);
}
```
– Parallèles à une droite situées à une distance d de cette droite Créer un point `A` et un segment `BC` ne passant pas par `A`.
Puis créer un point `D` pour définir $d =$`longueur(A,D)`.
Construire les parallèles à la droite BC situées à une distance $d =$`longueur(A,D)` de BC.
Réponse
On clique avec la souris pour avoir un point `A` et deux points `B` et `C` et le segment `BC`, puis on clique avec la souris pour avoir un point `D` ($d = longueur(A, D)$).
On tape :
```
D1:=perpendicular(B,droite(B,C));
C1:=cercle(B,distance(A,D));
I:=inter(D1,C1);
E:=I[0];
F:=I[1];
```
ou on utilise les nombres complexes pour définir `E` et `F` :
`E:=B+i*(C-B)*distance(A,D)/distance(B,C)` le point `E` est à une distance $d = longueur(A, D)$ de la droite BC,
`F:=B-i*(C-B)*distance(A,D)/distance(B,C)` le point `F` est à une distance $d = distance(A, D)$ de la droite BC (`E` et `F` sont symètriques par rapport à BC),
puis,
`parallele(E,droite(B,C))` trace une parallèle à la droite BC situées à une distance $d = distance(A, D)$ de BC.
`parallele(F,droite(B,C))` trace l'autre parallèle à la droite BC

situées à une distance $d = distance(A, D)$ de BC.
La liste des instructions se trouve dans `geo6` : créer trois points `A, B, C`, puis faire `Charger session` du menu `Fich` de `Xcas` et selectionner `geo6` du répértoire `examples/geo` pour exécuter ce fichier.

– Tangentes à un cercle

Chapitre 19

Le tableur

19.1 Généralités

Quand on a ouvert l'application `Tableur`, la touche `Menu` donne accés aux fonctions :
`SUM AVERAGE AMORT STAT1 REGRS PredY PredX HypZ1mean HypZ2mean`
Le tableur est une feuille de calculs ayant la forme d'un tableau composé de lignes et de colonnes qui déterminent des cases appelées cellules. Les cellules contiennent des valeurs ou des commandes ou encore des formules qui font références aux autres cellules.

19.2 Description de l'écran du tableur

Le tableur est un tableau composé de colonnes désignées par des lettres en majuscules `A,B,C,...` et quelquefois en minuscules `g,l,m, z` et de lignes numérotées par `1,2,....`
Les cases du tableur sont appelées cellules.
Ainsi, `A1` désigne la première cellule du tableur.

19.2.1 Pour copier la formule d'une cellule

Par exemple :
On met `1` dans `A1` et `=A1+1` dans `A2` et on veut recopier vers le bas cette formule.
Il y a 3 moyens de le faire : à vous de choisir celui qui vous convient !

- téchnique 1 : on sélectionne `A2` jusqu'à `A5`, et on tape : `=A1+1`,
- téchnique 2 : entre `=A1+1` dans `A2`, on fait `copy`, puis avec `select` du bandeau on sélectionne `A3` jusqu'à `A5`, et on fait `paste` et on choisit la formule parmi les expressions qui sont dans le presse papier,
- téchnique 3 : on met `=A1+1` dans `A2`, puis on sélectionne `A2` jusqu'à `A5` puis `Edit` et `Enter`

19.2.2 Références absolues et relatives

Dans une cellule on peut mettre :
- une chaine de caractères,

- une expression algébrique,
- une formule faisant référence à d'autres cellules. Ces références peuvent être absolues ou relatives à la cellule qui contient la formule. Les références absolues sont obtenues en rajoutant `$` devant la lettre désignant la colonne ou devant le numéro de la ligne de la cellule référence.
Les références relatives permettent de désigner les cellules par rapport à une autre : ainsi `A0` mis dans la cellule `B1` désigne la cellule située dans la colonne précédente et à la ligne précédente et c'est cette information qui sera recopiée quand on recopiera la formule vers le bas ou vers la droite.
Exemples :
Dans `A1` il y a 1 et je tape dans `B2` la formule :

- `$A$1+2` : dans `B2` il y aura 3, et si je recopie cette formule vers le bas, j'obtiens des 3 dans la colonne `B` car je recopie dans toutes les cases de la colonne `B` la formule `$A$1+2`. Si je recopie cette formule vers la droite, j'obtiens aussi des 3 dans la première ligne, car je recopie dans toutes les cases de la première ligne la formule `$A$1+2` puisque `$A$1` est la référence absolue de la case `A1`.
- `$A1+2` : dans `B2` il y aura 3, et si je recopie cette formule vers le bas cette formule deviendra `$A2+2` dans `B2`, `$A2+2` dans `B3`. La valeur de `B2` dépend donc de la valeur de `A1`, la valeur de `B3` dépend donc de la valeur de `A2` etc...
  Si je recopie cette formule vers la droite, cette formule deviendra `$A1+2` dans `C2`, `$A1+2` dans `D2` ...j'obtiens donc une ligne de 3. `$A0` fait toujours référence à la colonne `A` : `A` est une référence absolue mais `0` désigne ici la ligne précédente puisque `$A1` a été mis dans `B2`.
- `A$1+2` : dans `B2` il y aura 3, et si je recopie cette formule vers le bas, j'obtiens des 3 dans la colonne `B` mais si je recopie cette formule vers la droite, cette formule deviendra `B$1+2` dans `C2`, `C$1+2` dans `D2` etc...
- `=A1+2` : dans `B2` il y aura 3, et si je recopie cette formule vers le bas, cette formule deviendra `=A2+2` dans `B3`, `=A3+2` dans `B4` etc...si je recopie cette formule vers la droite, cette formule deviendra `=B1+1` dans `C2`, `=C1+1` dans `D2` etc...

19.3 Les fonctions du tableur

19.3.1 SUM

`SUM` effectue la somme des cellules mises en argument.
Par exemple : `SUM(A1:B3)` effectue la somme `A1+A2+A3+B1+B2+B3`

19.3.2 AVERAGE

`AVERAGE`

19.3.3 AMORT

`AMORT` permet de calculer l'amortissement d'un prêt. La syntaxe est : `AMORT(plage,n,i,pv,pm`
où :

`plage` fait référence aux cellules dans lesquelles on pourra lire les résultats,
`n` est le nombre de remboursements du prêt,
`i` est le taux d'intérêt, `pv` est la somme restant due `pmt` est le montant de chaque remboursement,
Par exemple :

Mais il me semble que c'est plus facile d'utiliser l'Application `Finance`.

19.3.4 `STAT1`

`STAT1` permet de faire des statistiques à une variable.
La syntaxe est : `STAT1(plage,[mode],["configuration"]).`
`plage` est la source des données, par exemple : `A1:B3`
`mode` vaut :
`1` si chaque colonne sont indépendantes,
`2` si les colonnes sont utilisées par paires (données, fréquences),
`3` si les colonnes sont utilisées par paires (données, poids),
`4`, Mais il me semble que c'est plus facile d'utiliser l'Application :
`Stats-1Var`.

19.3.5 `REGRS`

`REGRS` essaie de faire correspondre les données à une fonction (linéaire par défaut).
La syntaxe est : `REGRS(plage,[mode],["configuration"]).`
Par exemple : Mais il me semble que c'est plus facile d'utiliser les commandes `linear_regression, exponential_regression, etc....`

19.3.6 `PredY PredX`

Par exemple :

19.3.7 `HypZ1mean HypZ2mean`

Par exemple : Mais il me semble que c'est plus facile d'utiliser l'Application `Inférence`.

19.4 Utilisation du tableur sur des exemples

19.4.1 Exercice 1

La somme des nombres entiers impairs.
Écrire dans le tableur :
- les nombres entiers de 1 à 10 dans la colonne `A`,
- les carrés des nombres entiers de 1 à 10 dans la colonne `B`,
- les nombres entiers impairs $2k-1$ pour $k = 1..10$ dans la colonne `C`,
- la somme des entiers impairs $\sum_{j=1}^{k} 2j-1$ pour $k = 1..10$ dans la colonne `D`.

Calculer $\sum_{j=1}^{k} 2j - 1$ lorsque $k \in \mathbb{N}$.
Une solution

- On tape `1` dans `A1`.
  On met `=A1+1` dans `A2`, puis, on recopie vers le bas la formule contenue dans `A2` : pour cela on sélectionne `A2...A10` (lorsque `select.` du bandeau est visible et on sélectionne `A2...A10` avec le curseur) et on tape `=A1+1`.
- On met `=A1^2` dans `B1`, puis, on recopie vers le bas la formule contenue dans `B1` : pour cela on sélectionne `B1...B10` (lorsque `select.` du bandeau est visible et on sélectionne `B1...B10` avec le curseur) et on tape `=A1^2`.
- On met `=2*A1-1` dans `C1`, puis, on recopie vers le bas la formule contenue dans `C1` : pour cela on sélectionne `C1...C10` (lorsque `select.` du bandeau est visible et on sélectionne `C1...C10` avec le curseur) et on tape `=2*A1-1`.
- On tape `1` dans `D1`.
  On met `=D1+C2` dans `D2`, puis, on recopie vers le bas la formule contenue dans `D2` : pour cela on sélectionne `D2...D10` (lorsque `select.` du bandeau est visible et on sélectionne `D2...D10` avec le curseur) et on tape `=D1+C2`.

On obtient :

	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

On va donc montrer par récurrence que :
$\sum_{j=1}^{k} 2j - 1 = k^2$ lorsque $k \in \mathbb{N}$ La formule est vraie pour $k = 1..10$ (cf ci-dessus).
Supposons que pour $k = n$ $\sum_{j=1}^{n} 2j - 1 = n^2$ alors pour $k = n + 1$ on a $\sum_{j=1}^{n+1} 2j - 1 = \sum_{j=1}^{n} +2(n+1) - 1 = n^2 + 2n + 1 = (n+1)^2$ Donc la formule est montrée.

19.4.2 Exercice 2

Le triangle de Pascal et la suite de Fibonacci.

La suite de Fibonacci

La suite de Fibonacci est la suite u définie par :
$u_0 = 1$
$u_1 = 1$
$u_n = u_{n-1} + u_{n-2}$ pour $n > 1$

On veut trouver les 11 premiers termes de cette suite.
On utilise le tableur.
On met :
`1` dans $A1$
`1` dans $A2$
On sélectionne la colonne A de 3 à 11 (on doit avoir dans le bandeau `Select.` et `Aller`$\downarrow$) et on tape : `=A1+A2`
On obtient dans la colonne A :
`1,1,2,3,5,8,13,21,34,55,89`

Le triangle de Pascal

On veut utiliser le tableur pour créer le triangle de Pascal de 0 à 10.
Pour cela on utilise les relations pour $n \in \mathbb{N}$ et $p \in \mathbb{N}$:
$C_n^0 = 1$, $C_n^n = 1$, $C_n^p = 0$ si $p > n$ et $C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$ pour $0 < p \leq n$.
On vérifie et on tape :
`simplify(comb(j-1,k-1)+comb(j-1,k)-comb(j,k))`
On obtient :
`0`
Pour remplir le tableur :
On met `1` dans $A1$ puis on copie $A1$ avec sur les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller`$\downarrow$ dans le bandeau et on sélectionne les cases de 1 à 11, puis on appuie sur les touches `Shift Paste` : cela a pour effet de mettre des 1 dans la colonne A.
On met des `0` dans $B1$ puis on copie $B1$ avec les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller`$\rightarrow$ dans le bandeau et on selectionne les colonnes de B à K, puis on appuie sur les touches `Shift Paste` : cela a pour effet de mettre des 0 dans la ligne `1` à partir de la colonne B.
On met `=A1+B1` dans $B2$ puis on copie $B2$ avec sur les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller`$\rightarrow$ dans le bandeau et on sélectionne les colonnes de B à K, puis on appuie sur les touches `Shift Paste` : cela a pour effet de mettre 1 dans $B2$ et des 0 dans la ligne `2` à partir de la colonne C mais cela recopie aussi les formules `=B1+C1` dans $C2$, `=C1+D1` dans $D2$ etc.....
Puis on copie aussi la formule `=A1+B1` dans la colonne B (avec sur les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller`$\downarrow$ dans le bandeau et on sélectionne les cases de 2 à 11, puis on appuie sur les touches `Shift Paste` : cela a pour effet de mettre, dans la colonne B : `0,1,2,3,4..10`.
Puis on copie aussi la formule `=B1+C1` dans la colonne C (avec sur les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller`$\downarrow$ dans le bandeau et on sélectionne les cases de 2 à 11, puis on appuie sur les touches `Shift Paste` (il faut recopier la formule et donc choisir `0`$\rightarrow$ `Formule`) : cela a pour effet de mettre, dans la colonne C :
`0,0,1,3,6,10,15,21,28,36,45`.
On recopie de même la formule située dans $D2$ (resp $E2...K2$) dans la colonne D (resp $E...K$).
On obtient :

	A	B	C	D	E	F	g	H	I	J	K
1	1	0	0	0	0	0	0	0	0	0	0
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

La suite de Fibonacci et le triangle de Pascal

Lorsqu'on fait la somme des diagonales montantes du triangle de Pascal on obtient la suite de Fibonacci.
Par exemple :

$$\begin{pmatrix} 1 & = \\ 1 & = \\ 2 & = \\ 3 & = \\ 5 & = \\ 8 & = \\ 13 & = \end{pmatrix} \begin{pmatrix} 1 & & 0 & & 0 & & 0 & & 0 & & 0 \\ & \nearrow & & \nearrow & & \nearrow & & \nearrow & & \nearrow & \\ 1 & & 1 & & 0 & & 0 & & 0 & & 0 \\ & \nearrow & & \nearrow & & \nearrow & & \nearrow & & \nearrow & \\ 1 & & 2 & & 1 & & 0 & & 0 & & 0 \\ & \nearrow & & \nearrow & & \nearrow & & \nearrow & & & \\ 1 & & 3 & & 3 & & 1 & & 0 & & 0 \\ & \nearrow & & \nearrow & & \nearrow & & & & & \\ 1 & & 4 & & 6 & & 4 & & 1 & & 0 \\ & \nearrow & & \nearrow & & & & & & & \\ 1 & & 5 & & 10 & & 10 & & 5 & & 1 \\ & \nearrow & & & & & & & & & \\ 1 & & 6 & & 15 & & 20 & & 15 & & 6 \end{pmatrix}$$

On tape :

```
A:=makemat((j,k)->comb(j,k),11,11)
L:=sum(A[j-k, k],k=0..j)$(j=0..10)
```

On obtient :

```
1,1,2,3,5,8,13,21,34,55,89
```

On tape :

```
L1:=0,sum(A[j-k-1,k-1],k=1..j-1)$(j=1..10)
```

On obtient :

```
0,0,1,1,2,3,5,8,13,21,34
```

On tape :

```
L2:=1,sum(A[j-k-1,k],k=0..j-1)$(j=1..10)
```

On obtient :

```
1,1,1,2,3,5,8,13,21,34,55
```

Pour le montrer on utilise la relation pour $n \in \mathbb{N}$:
$C_n^0 = 1$, $C_n^n = 1$ et $C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$ pour $0 < p < n$.
Soit a_n la suite égale à la somme des diagonales montantes du triangle de Pascal.

On a donc :
$a_0 = 1$, $a_1 = 1$ et pour $n > 1$:
$a_n = \sum_{p=0}^{\mathtt{floor(n/2)}} \mathtt{comb(n-p,p)} = 1 + \sum_{p=1}^{\mathtt{floor(n/2)}} \mathtt{comb(n-p,p)}$
$a_n = 1 + \sum_{p=1}^{\mathtt{floor(n/2)}} \mathtt{comb(n-p-1,p-1)} + \sum_{p=1}^{\mathtt{floor(n/2)}} \mathtt{comb(n-p-1,p)}$
On a pour tout n : $\mathtt{floor(n/2)-1} = \mathtt{floor((n-2)/2)}$).
On a si $n = 2k$, on a $\mathtt{floor(n/2)} = \mathtt{floor((n-1)/2)+1} = k$:
$1 + \sum_{p=1}^{\mathtt{floor(n/2)}} \mathtt{comb(n-p-1,p)} =$
$\sum_{p=0}^{\mathtt{floor((n-1)/2)}} \mathtt{comb(n-1-p,p)} + \mathtt{comb(2k-1-k,k)} = a_{n-1}$
car $\mathtt{comb(2k-1-k,k)} = 0$
$\sum_{p=1}^{\mathtt{floor(n/2)}} \mathtt{comb(n-p-1,p-1)} =$
$\sum_{p=0}^{\mathtt{floor((n-2)/2)}} \mathtt{comb(n-2-p,p)} = a_{n-2}$
car `floor(n/2)-1=floor((n-2)/2)`
On a si $n = 2k+1$ alors $\mathtt{floor(n/2)} = \mathtt{floor((n-1)/2)} = \mathtt{k}$:
$1 + \sum_{p=1}^{\mathtt{floor(n/2)}} \mathtt{comb(n-p-1,p)} =$
$\sum_{p=0}^{\mathtt{floor((n-1)/2)}} \mathtt{comb(n-1-p,p)} = a_{n-1}$
$\sum_{p=1}^{\mathtt{floor(n/2)}} \mathtt{comb(n-p-1,p-1)} =$
$\sum_{p=0}^{\mathtt{floor((n-2)/2)}} \mathtt{comb(n-2-p,p)} = a_{n-2}$
car `floor(n/2)-1=floor((n-2)/2)`
Donc :
$a_0 = 1$, $a_1 = 1$ et pour $n > 1$ on a $a_n = a_{n-1} + a_{n-2}$
a_n est donc la suite de Fibonacci.

Chapitre 20

Les autres Applications

20.1 L'Application Fonction

On tape dans la vue symbolique de l'Application Fonction :
`F1(X)=SIN(2X)+X`
`F2(X)=∂(F1(A),A=X)`
On met `∂(F1(A),A=X)` en surbrillance et on appuie sur `EVAL` du bandeau.
On obtient si on est en `Radians` :
`F2(X)=COS(2*X)*2+1`
Dans `HOME`, on tape :
`F1(1)`
On obtient si on est en `Radians` :
`SIN(2)+1`
Dans `HOME`, on tape :
`F2(1)`
On obtient si on est en `Radians` :
`COS(2)*2+1`
Dans `HOME`, on tape :
`∂(F1(X),X=1)`
On obtient si on est en `Radians` :
`0.167706326906`

20.2 L'Application Suite

20.2.1 La suite de Fibonnacci

On tape :
`U1(1)=1`
`U1(2)=1`
`U1(N)=U1(N-1)+U1(N-2)`
On obtient en appuyant su `Num` :
`la suite de Fibonnacci`
On tape la suite des restes :
`U1(1)=1`
`U1(2)=1`
`U1(N)=U1(N-1)+U1(N-2)`

On obtient en appuyant sur `Num` :
`la suite de Fibonnacci`

20.2.2 Le PGCD

Voici une mise en en œuvre de l'algorithme d'Euclide avec la `HPrime`.
Voici la description de cet algorithme :
On effectue des divisions euclidiennes successives :

$$\begin{array}{rcl} A = & B \times Q_1 + R_1 & 0 \leq R_1 < B \\ B = & R_1 \times Q_2 + R_2 & 0 \leq R_2 < R_1 \\ R_1 = & R_2 \times Q_3 + R_3 & 0 \leq R_3 < R_2 \\ ..... & & \\ R_{n-2} = & R_{n-1} \times Q_n + R_n & 0 \leq R_n < R_{n-1} \end{array}$$

Après un nombre fini d'étapes (au plus B), il existe un entier n tel que : $R_n = 0$.
on a alors :
$PGCD(A, B) = PGCD(B, R_1) = ...$
$PGCD(R_{n-1}, R_n) = PGCD(R_{n-1}, 0) = R_{n-1}$
À l'aide de l'Aplet Suite, on écrit la suite des restes.
On tape la suite des restes R :

```
U1(1)=76
U1(2)=56
U1(N)=irem(U1(N-2),U1(N-1))
```

On obtient en appuyant sur `Num` :
`76,56,20,16,4,0 donc ....le PGCD de 76 et 56 vaut 4`

20.2.3 L'identité de Bézout

L'algorithme d'Euclide permet de trouver un couple U, V vérifiant :

$$A \times U + B \times V = PGCD(A, B)$$

Avec l'Aplet Suite, on va définir "la suites des restes" R et deux suites U et V, de façon qu'á chaque étape on ait :
$R_n = U_n \times A + V_n \times B$.

Si Q est "la suites des quotients", $Q_n = iquo(R_{n-2}, R_{n-1})$ car Q_n est le quotient entier de R_{n-2} par R_{n-1} et on a :
$R_n = irem(R_{n-2}, R_{n-1}) = R_{n-2} - Q_n \times R_{n-1}$ car $R_{n-2} = R_{n-1}Q_n + R_n$ avec $0 \leq R_n < R_{n-1}$.
U_n et V_n vont donc vérifier la même relation de récurrence. On a au début :
$R_1 = A$ $R_2 = B$
$U_1 = 1$ $U_2 = 0$ puisque $A = 1 \times A + 0 \times B$
$V_1 = 0$ $V_2 = 1$ puisque $B = 0 \times A + 1 \times B$

On tape dans `U1` la suite des restes R pour $A = 76$ et $B = 56$:

```
U1(1)=76
U1(2)=56
```

```
U1(N)=irem(U1(N-2),U1(N-1))
```

On tape la suite des quotients Q dans `U2` :

```
U2(1)=0
U2(2)=0
U2(N)=iquo(U1(N-2),U1(N-1))
```

0n va mettre U dans `U3` et dans V `U4` pour qu'à chaque étape on ait $76 * U + 56 * V = R$.
Puisque $R(N) = R(N-2) - R(N-1) * Q(N)$ et $Q(N) iquo(R(N-2), R(N-1))$, si on a :
$76 * U(N-2) + 56 * V(N-2) = R(N-2)$ (1) et
$76 * U(N-1) + 56 * V(N-1) = R(N-1)$ (2)
En faisant (1)-(2)*Q(N), on a :
$76 * (U(N-2) - U(N-1) * Q(N)) + 56 * (V(N-2) - V(N-1) * Q(N)) = R(N-2) - R(N-1) * Q(N) = R(N)$
Les relations de récurence sont donc :
$U(N) = U(N-2) - U(N-1) * Q(N) = U(N-2) - U(N-1) * iquo(R(N-2), R(N-1))$
$V(N) = V(N-2) - V(N-1) * Q(N) = V(N-2) - V(N-1) * iquo(R(N-2), R(N-1))$
On tape dans `U3` la suite des U :

```
U3(1)=1
U3(2)=0
U2(N)=U3(N-2)-U3(N-1)*iquo(U1(N-2),U1(N-1))
```

On tape dans `U4` la suite des V :

```
U4(1)=0
U4(2)=1
U4(N)=U4(N-2)-U4(N-1)*iquo(U1(N-2),U1(N-1))
```

Ainsi pour chaque `N` on a `76*U3(N)+56*U4(N)=U1(N)` On obtient en appuyant sur `Num` :
`76,0,1,0` (76=76*1+56*0)
`56,0,0,1` (56=76*0+56*1)
`20,1,1,-1` (20=76*1+56*-1)
`16,2,-2,3` (16=76*-2+56*3)
`4,1,3,-4` (4=76*3+56*-4)
`0,4,-14,19` (0=76*-14+56*19)
`donc 76*3+56*-4=4=PGCD de 76 et 56`

20.3 L'Aplet Paramétrique

On tape :

```
X1(T)=-COS(2T)
Y1(T)=SIN(3T)
```

On obtient :
`le dessin d'une courbe ressemblant à un` α

20.4 L'Aplet Polaire

On tape :
`R1(θ)=COS(3θ)`
On obtient :
`le dessin d'un trèfle`

20.5 L'Aplet Résoudre

On tape dans la vue symbolique de l'Aplet Résoudre :
`COS(X)-X`
en appuyant su `Num` :
On tape :
`1` Pour démarrer l'itération et on presse sur `SOLVE` du bandeau et on obtient :
`0.739085133215`
On tape maintenant dans `HOME` :
`X`
On obtient :
`0.739085133215`
la valeur obtenue a été stockée dans `X`.

20.6 L'Aplet Finance

On remplit les différents champs sauf un pour lequel on appuie sur `SOLVE` du bandeau.
Par exemple, on veut connaitre les mensualités d'un prêt à 4.6 % de 100 000 euros sur 10 ans, on tape :

```
NbPmt 120      IPYR 4.6
PV 100000      PPYR 12
PMT            CPYR 12
FV 0.00        Fin ⊻
Taille du groupe 12
```

On met `PMT` en surbrillance et on appuie sur `SOLVE` du bandeau.
On obtient le montant du remboursement mensuel :
`-1041.21`
Les remboursements de ce prêt sont donc de 1041.21 euros par mois.
On appuie sur `AMORT` du bandeau.
On obtient le tableau d'amortissement par groupement de 12 mois :
`Princ` donne le montant des sommes remboursées
`Inter` donne le montant des interêts versés
`Balan` donne le montant des sommes dues
Par exemple au bout de la 2ième année on a :
`Princ=-16505.07` : on a donc remboursé 16505.07 euros
`Inter=-8484.0` : on a payé 8484 euros d'interêts
`Balan=83494.9` : il reste à rembourser 83494.93 euros (100000-16505.07)
En appuyant sur la touche `PLOT` on a le graphe de l'amortissement.
On peut lire :

`1-12 Principal 8063.12` et `Interêt 4431.41`
et en utilisant la flèche vers la droite
`13-24 Principal 8441.95` et `Interêt 4052.59`
On retrouve ainsi les valeurs précédentes :
8063.12+8441.95=16505.07 et 4431.41+4052.59=8484.0.
On retourne dans l'aplet `Finance` en appuyant sur la touche `Num`.
Si les remboursements sont trimestriels, on change :
`NbPmt 40` et `PPYR 4` et on laisse `CPYR 12`.
On met `PMT` en surbrillance et on appuie sur `SOLVE` du bandeau.
On obtient le montant du remboursement trimestriel :
`-3133.03`
Les remboursements de ce prêt sont donc de 3133.03 euros par trimestre.

20.7 L'Aplet Système Linéaire

Cette Aplet permet de résoudre des systèmes linéaires de 2 équations à 2 inconnues X, Y ou de 3 équations à 3 inconnues X, Y, Z.
Remarque
Pour résoudre un système linéaire de 2 équations à 2 inconnues X, Y, il faut cocher `2×2` du bandeau.
Le canevas du système s'affiche et il ne reste qu'a taper les coefficients.
La réponse s'inscrit en bas de l'écran. On tape :
```
1*X+1*Y+1*Z=3
2*X+(-2)*Y+1*Z=1
3*X+1*Y+1*Z=5
```
On obtient :
`X:1,Y:1,Z:1`
On tape :
```
1*X+1*Y+1*Z=3
2*X+2*Y+2*Z=1
3*X+1*Y+1*Z=5
```
On obtient :
`No Solutions`

20.8 Aplet Triangle Solver

Etant donné un triangle définit par 3 données : longueur de côté et/ou valeur d'angles (cf les cas d'égaliés des triangles).
L'aplet `Triangle Solver` permet de calculer la longueur des autres côtés et/ou la valeur des autres angles de ce triangle.
Soit le triangle de côtés A, B, C tel que $A = 4$ $B = 3$ $C = 5$.
On veut avoir la valeur des angles de ce triangle.
On tape :
`A=4, B=3, C=5`
On appuie sur `SOLVE` du bandeau. On obtient si on est ebn degrés :
α =`5.31301E1` (c'est l'angle opposé au côté `A`)
β =`3.68699E1` (c'est l'angle opposé au côté `B`)

δ =90 (c'est l'angle opposé au côté C)
Dans Home si on tape 5.31301E1 enter, puis on met 5.31301E1 en surbrillance et on appuie sur la touche SHIFT a<->b/c.
On obtient :
53°7'48.36"
Si on appuie à nouveau sur la touche SHIFT a<->b/c.
On obtient :
53.1301

20.9 Statistiques 1-Var

L'aplet Statistiques 1-Var permet de faire des statistiques à une variable avec ou sans effectif.
Exemple On mesure les tailles en cm de 10 personnes.
On obtient :
150,165,170,165,160,170,160,175,165,180.

1. Calculer la médiane, la moyenne et l'écart-type de cette série.

   On ouvre l'aplet Statistiques 1-Var et on utilise une seule colonne.
   On appuie sur START du bandeau et on tape dans D1 :
   150 ENTER 165 ENTER....
   On peut si on le désire trier les données avec TRIER du bandeau.
   On appuie sur Symb, on coche H1 :D1 et on vérifie que l'on a Plot1 : Histogramme
   On appuie sur la touche Num et sur STATS du bandeau.
   On obtient :
   Nb Item 10
   X min 150
   X Q1 160
   X med 165
   X Q3 170
   X max 180
   $\sum$X 1660
   $\sum$X^2 276200
   MoyX 166
   SX 8.432740
   σX 8
   SE X 2.6666667

2. Construire l'histogramme.
   On régle tout d'abord le Plot Setup (SHIFT Plot) : Largeur 5
   HRNG 150 180
   XRNG 150 180
   YRNG -1 10
   XTICK 1 YTICK 1
   Puis on appuie sur Plot.

On obtient :
`l'histogramme`

3. Dessiner la boite à moustache.
On appuie sur `Symb` et on met `Plot1 : Histogramme` en surbrillance.
Avec `CHOOS` du bandeau on choisit `Boite-à-moustache`, puis on appuie sur `Plot`.
On obtient :
`Le dessin de la boite à moustache`

4. Le même exercice en utilisant une colonne pour les effectifs.
On mesure les tailles en cm de 10 personnes.
On obtient :
```
150:1
160:2
165:3
170:2
175:1
180:1
```

On appuie sur `Num` et on tape dans `D2` :
`150 ENTER 160 ENTER 165 ENTER 170 ENTER 175 ENTER 180 ENTER`
et dans `D3` :
`1 ENTER 2 ENTER 3 ENTER 2 ENTER 1 ENTER 1 ENTER`
On appuie sur `Symb` et on décoche `H1 :D1`et on met :
`H2 :D2` et on tape `D3` lorsque `Freq` est en surbrillance. `H2` est coché ainsi que `Plot2 : Histogramme`.
On obtient la même chose que précédemment.
```
Nb Item 10
X min 150
X Q1 160
X med 167.5
X Q3 175
```

20.10 Statistiques 2-Var

L'aplet `Statistiques 2-Var` permet de faire des statistiques à deux variables.
Exemple
On mesure les tailles en cm et le poids en kg de 10 personnes.
On obtient :
```
150:41
165:53
170:70
165:63
160:52
170:68
160:62
```

```
175:72
165:58
180:75
```

1. Calculer la taille moyenne et le poids moyen de ces individus.
   Calculer le coefficient de corrélation.
   On ouvre l'aplet Statistiques 2-Var et on utilise deux colonnes.
   On appuie sur `START` du bandeau.
   On tape dans `C1` :
   `150 ENTER 165 ENTER....`
   On tape dans `C2` :
   `41 ENTER 53 ENTER....`
   On peut si on le désire trier les données avec `TRIER` du bandeau.
   On appuie sur `Symb`, on met `S1 :C1 C2`, on coche `S1` et on vérifie que l'on a :
   `Type1 :Lineaire`
   `Fit1 : m*X+b`
   On appuie sur la touche `Num` et sur `STATS` du bandeau.
   On obtient :
   `Nb Item 10`
   `Corr 0.91798`
   `CoefDet 0.84269`
   `SCov 81.7778`
   `PCov 73.6`
   `X max 180`
   $\sum$`XY 102660`
   En appuyant sur `X` du bandeau on a :
   `MoyX 166`
   $\sum$`X 1660`
   $\sum$`X^2 276200`
   `SX 8.432740`
   σ`X 8`
   `SE X 2.6666667`
   En appuyant sur `Y` du bandeau on a :
   `MoyY 61.4`
   $\sum$`Y 614`
   $\sum$`Y^2 38704`
   `SY 10.5641`
   σ`Y 10.0220`
   `SE Y 3.340659`

2. Dessiner le nuage de points
   On régle tout d'abord le Plot Setup (`SHIFT Plot`):`XRNG 145 185`
   `YRNG 39 77`
   `XTICK 5 YTICK 2`
   Puis on appuie sur `Plot`.
   On obtient :
   `le nuage de points et le tracé de la droite de régression`

3. Déterminer une droite de régression linéaire par la méthode des moindres carrés.
   On obtient l'équation de la droite de régression linéaire en appuyant sur `Symb`.
   On obtient :
   `Fit1:1.15*X+-129.5`

20.11 Aplet Inférence

L'aplet `Inférence` permet de faire des statistiques inférencielles.

Résultats à retenir

Si μ est la moyenne d'une population et m_α la moyenne des moyennes des valeurs d'un échantillon de taille n extrait de cette population, on a : $m_\alpha = \mu$,

Si σ l'écart-type d'une population et σ_α l'écart-type de la moyenne des moyennes des valeurs d'un échantillon de taille n extrait de cette population, on a : $\sigma_\alpha = \frac{\sigma}{\sqrt{n}}$,

σ^2 la variance d'une population et s^2 la moyenne des variances des valeurs d'un échantillon de taille n extrait de cette population : $S^2 = \frac{n-1}{n}\sigma^2$,
Donc on a $\sigma_\alpha \frac{\sigma}{\sqrt{n}} = \frac{S}{\sqrt{n-1}}$.
Lorsqu'on a un échantillon de taille n de moyenne $\overline{x}$ et d'écart-type s on choisit faute de mieux :
$S = s$ et $m_\alpha = \overline{x}$ et donc la moyenne des valeurs des échantillons de taille n a pour moyenne $\overline{x}$ et pour écart-type $\frac{s}{\sqrt{n-1}}$.

En Résumé

- La moyenne et la variance d'une somme sont respectivement la somme des moyennes et la somme des variances.
- La moyenne et la variance d'une différence sont respectivement la différence des moyennes et la somme des variances.
- La moyenne de l'ensemble des moyennes des valeurs d'un échantillon de taille n extrait d'une population de moyenne m est m.
  La variance de l'ensemble des moyennes des valeurs d'un échantillon de taille n extrait d'une population de variance σ^2 est $\frac{\sigma^2}{n}$.
  L'écart-type de l'ensemble des moyennes des valeurs d'un échantillons de taille n extrait d'une population d'écart-type σ est $\frac{\sigma}{\sqrt{n}}$.
- La moyenne s des variances de l'ensemble des variances des valeurs d'un échantillon de taille n extrait d'une population de variance σ^2 est $\frac{n-1}{n}\sigma^2$.
- L'écart-type de l'ensemble des moyennes des valeurs d'un échantillon de taille n s'obtient en divisant la moyenne S des variances de l'ensemble des variances des valeurs d'un échantillon de taille n par $\sqrt{n-1}$:
  l'ensemble des moyennes des valeurs d'un échantillon de taille n a donc pour moyenne m et pour écart-type $\frac{S}{\sqrt{n-1}}$.

20.11.1 Fréquence d'un caractère et jugements sur échantillons

Comparaison d'une fréquence expérimentale et d'une fréquence théorique

Dans une population, la fréquence d'un caractère a comme fréquence p.
Soit un échantillon d'effectif n pour lequel la fréquence de ce caractère est f. La question est de savoir si cet échantillon a été extrait de cette population.
Pour que un échantillon d'effectif n, soit X la variable aléatoire égale au nombre de membres présentant le caractère considéré de probabilité p. X suit une loi binomiamle de moyenne np et d'écart-type $\sqrt{np(1-p)}$. Si n est grand ($n > 50$) la loi binomiamle peut être approchée par la loi normale.
La distribution des fréquences du caractère des échantillons de taille n a pour moyenne p et pour écart-type $\sqrt{\frac{p(1-p)}{n}}$ **Exemple**
On compte le nombre de fois qu'un joueur retourne un as quand il donne les cartes d'un jeu de 32 cartes : pour 800 donnes, ce joueur a retourné 180 fois un as. Ce joueur est-il un tricheur ?
On a $n = 800$, et la probabilité de retourner un as est :
$p = \frac{4}{32} = \frac{1}{8}$.
Donc le nombre X de retournements d'un as suit une loi binomiale de moyenne $n * p = 100$ et d'écart-type $\sqrt{np(1-p)} = 10\sqrt{\frac{7}{8}} \simeq 9.35414346693$.
Cherchons la probabilité pour que $70 < X < 130$, on tape dans `Home` :
`normald_cdf(100,9.354,130)-normald_cdf(100,9.354,70)`
On obtient :
`0.998659588108`
On peut donc affirmer que ce joueur est un tricheur : la probabilité de ce tromper étant inférieure à $\frac{2}{1000}$.
Avec la calculatrice :
On appuie sur `Symb` et on choisit :
`Test d'Hypo`
`Type Z-Test: 1` π
`Hypoth Alt` $\pi \neq \pi_0$
On appuie sur `Num` et on tape :
`x=180`
`n=800`
μ_0`=0.125`
σ`=50.25`
α`=0.02`
On appuie sur `CALC` du bandeau :
On obtient :
`Résultat 0` (Rejet de l'hypothèse à α=0.02)
`Test Z 8.5523597412`
`Test` $\widehat{p}$ `0.225`
`Prob. 1.2059722286E-17`
`Crit Z` $\pm$`2.32634787404`
`Minimum critique 0.0906543...`
`Maximum critique 0.15934...`
On tape dans `Home` :
`(180-100)/9.35414346693`

On obtient :
`8.5523597412`
On tape dans `Home` :
`normald_icdf(100,9.35414346693,0.99)`
On obtient :
`121.760991768`
On tape dans `Home` :
`(121.760991768-100)/9.35414346693`
On obtient :
`2.32634787407`
On tape dans `Home` pour $\alpha = 0.002$:
`normald_icdf(100,9.354,0.001),`
`normald_icdf(100,9.354,0.999)`
On obtient :
`71.0939670081,128.906032992`

Intervalle de confiance d'une proportion ou fréquence

Comment évaluée la fréquence p d'un caractère dans une population au vue d'un échantillon de taille n.
Exemple
On jette un dé 3000 fois et on a obtenu 490 fois le six. Peut-on dire que le dé est régulier ? On a :
$n = 3000, f = 49/300 \simeq 0.163333333333, 1-f = 251/300 \simeq 0.836666666667$ donc $m = nf = 490$ et $s = \sqrt{nf(1-f)} \simeq 20.2476336066$ Au seuil de $\alpha = 0.002$, on tape dans `Home` :
`normald_cdf(3000*49./300,sqrt(3000*49./300*251/300),500)-normald_cdf(3000*49.`
On obtient :
`0.378612513321`
Donc la probabilité que p soit dans l'intervalle $[480, 500]$ est 0.378612513321. Il est donc tout à fait raisonnable d'admettre que $p = \frac{1}{6}$ c'est à dire que le dé est régulier car sinon la probabilité de se tromper serait de $1 - 0.378612513321 \simeq 0.621387486679$ **Avec la calculatrice :**
On appuie sur `Symb` et on choisit :
`Inter de Confiance`
`Type Z-Test: 1 π`
On appuie sur `Num` et on tape :
`x=490`
`n=3000`
μ_0=`0.166666666667`
α=`0.05`
On appuie sur `CALC` du bandeau :
On obtient :
`C 0.95`
`Crit Z ± 1.95996398454`
`Minimum critique 0.150105122453`
`Maximum critique 0.176561544214...`
Avec :

```
C=0.38
```
On appuie sur `CALC` du bandeau :
On obtient :
```
C 0.38
Crit Z ± 0.495850347347
Minimum critique 0.159986734614
Maximum critique 0.166679932052
```

Comparaison des fréquences de 2 échantillons

On a 2 échantillons d'effectifs n_1 et n_2 pour lesquels les fréquences d'un caractère sont f_1 et f_2. Si f_1 et f_2 sont différents comment savoir si les 2 échantillons proviennent d'une même population ou pas ?
Si on fait l'hypothèse que les 2 échantillons proviennent d'une même population on peut estimer la fréquence p de la population entière en reunissant les 2 échantillons avec $p = \frac{n_1 f_1 + n_2 f_2}{n_1 + n_2}$.
L'écart-type de la distribution des fréquences du caractère de l'échantillon 1 est $\sqrt{\frac{f_1(1-f_1)}{n_1}}$ et celui de l'échantillon 2 est $\sqrt{\frac{f_2(1-f_2)}{n_2}}$.
On considère la distibution des valeurs de $f_1 - f_2$, si les 2 échantillons proviennent d'une même population :
la moyenne de $f_1 - f_2$ est 0 et
l'écart-type s de $f_1 - f_2$ est $\sqrt{\frac{p(1-p)}{n_1} + \frac{p(1-p)}{n_2}}$
Exemple
Dans la classe A de 40 élèves il y a eu 23 reçus et
dans la classe B de 40 élèves il y a eu 17 reçus. Les élèves de la classe A et de la classe B appartiennent-ils à la même population ?
Si ils appartiennent à la même population, la distribution des fréquences $f_1 - f_2$ suit une loi normale :
de moyenne $p = \frac{40}{80} = 0.5$ et
d'écart-type $s = \sqrt{\frac{2*0.5*0.5}{40}} = 0.111803398875$
On sait que $f_1 = \frac{23}{40} \simeq 0.575$ et $f_2 = \frac{17}{40} \simeq 0.425$ donc $f_1 - f_2 = 0.15$ et $\frac{f_1 - f_2}{s} \simeq 1.3416407865$
On tape :
```
normal_cdf(0,0.111803398875,0.15)-normal_cdf(0,0.111803398875,-0.15)
```
ou on tape :
```
normal_cdf(0,1,1.3416407865)-normal_cdf(0,1,-1.3416407865)
```
On obtient :
```
0.820287505121
```
On tape :
```
1-0.820287505121
```
On obtient :
```
0.179712494879
```
Cela veut dire que l'on peut dire que les 2 classes proviennent de la même population car si on affirmait le contraire on aurait une probabilité de 17.97 % de se tromper.
On tape si on veut un résultat au seuil de 5% :
```
normal_icdf(0,0.111803398875,0.025),normal_icdf(0,0.111803398875,0.975
```

On obtient :
`-0.219130635144,0.219130635144`

On tape si on veut un résultat au seuil de 5% :
`normal_icdf(0,1,0.025),normal_icdf(0,1,0.975)`
On obtient :
`-1.95996398454,1.95996398454`
Comme 0.15 se trouve dans l'intervalle [-0.219130635144,0.219130635144], on peut dire qu'au seuil de 5% les deux classes proviennent de la même population.
Avec l'Aplet Inférence :
On appuie sur `Symb` et on choisit :
`Test d'Hypo`
`Type Z-Test: 1` $\pi_1 - \pi_2$
`Hypoth Alt` $\pi_1 \neq \pi_2$
On appuie sur `Num` et on tape :
x_1=`23`
x_2=`17`
n_1=`40`
n_2=`40`
α=`0.05`
On appuie sur `CALC` du bandeau :
On obtient :
`Résultat 1` (Non Rejet de H_0 à α=0.05)
`Test Z 1.3416407865`
`Test` $\Delta\widehat{p}$ `0.15`
`Prob. 0.179712494879`
`Crit Z` $\pm$`1.95996398454`
`Minimum critique -0.0666513904072`
`Maximum critique 0.366651390407`
On tape dans `Home` :
`normal_icdf(0.15,0.111803398875,0.025),normal_icdf(0.15,0.111803398875,0.975)`
On obtient :
`-0.0691306351442,0.369130635144`

L'échantillon 1 a comme moyenne :
$f_1 = \frac{23}{40} = 0.575$
et comme variance :
$\frac{f_1(1-f_1)}{n_1} = \frac{0.575*0.425}{40}$
L'échantillon 2 a comme moyenne :
$f_1 = \frac{17}{40} = 0.425$
et comme variance :
$\frac{f_2(1-f_2)}{n_2} = \frac{0.425*0.575}{40}$
On considère la distribution des fréquences $f_1 - f_2$: on peut dire qu'elle a comme moyenne la différence des moyennes soit :
$0.575 - 0.425 = 0.15$. et comme variance la somme de ces 2 variances i.e :
$\frac{2*0.425*0.575}{40}$
On tape dans `Home` pour avoir l'écart-type de la distribution des fréquences $f_1 - f_2$:
`sqrt(2*0.575*0.425/40.)`

On obtient :
`0.1105384548472`
`normal_icdf(0.15,0.1105384548472,0.025),normal_icdf(0.15,0.1105384548`
On obtient le minimum critique et le maximum critique :
`-0.0666513904072,0.366651390407`

20.11.2 Échantillons extraits d'une distribution normale

Comparaison d'une moyenne expérimentale et d'une moyenne théorique

Si la serie statistique suit une distribution normale de moyenne m et d'écart-type σ, la distribution des moyennes m_α des échantillons d'effectif n suit une loi normale de moyenne m et d'écart-type $\sigma_\alpha = \frac{\sigma}{\sqrt{n}}$.
Si S est la moyenne des écarts-types des échantillons d'effectif n alors on a $\frac{S}{sqrtn} = \frac{\sigma}{sqrtn-1}$ et donc $\sigma_\alpha = \frac{S}{sqrtn-1}$.
Exemple
On pèse 100 pains de 400 gr pris au hasard dans une boulangerie et on obtient une moyenne m_1 de 390 gr avec un écart-type s_1 de 50 gr.
Le boulanger respecte-t-il le poids de ses pains de 400 gr pour le seuil de confiance de 95 % ?
Estimer la moyenne de la population à partir de $\overline{x}$ et d'écart-type s de cet échantillon.

- Il s'agit de comparer une moyenne expérimentale $m_\alpha = \overline{x}$ à une moyenne théorique μ.
  L'hypothèse (H_0) est que la différence entre ces 2 moyennes n'est pas significative ie (H_0) : $\overline{x} = \mu$. L'hypothèse alyternative (H_1) est donc (H_1) : $\overline{x} \neq \mu$.
  L'écart-type s de l'échantillon de 100 éléments est 50, on estime donc l'écart-type de la population à :
  $50 * \sqrt{100/99} \simeq 50.25$
  On sait que :
  $$U = \frac{\overline{x} - \mu}{\frac{\sigma}{\sqrt{(n)}}}$$
  suit sensiblement une loi centrée réduite car $n = 100 > 30$.
  On pose donc :
  $$U = \frac{390 - 400}{\frac{50}{\sqrt{(99)}}} = -1.98997487421$$
  Etant donné α, on cherche u_α tel que :
  Prob(-u_α<U<u_α)=$1 - \alpha$.
  Si $\alpha = 0.05$,
  0n tape : `normald_icdf(0.975)`
  On obtient : `1.95996398454 ≃ 1.96`
  0n tape : `normald_icdf(0.025)`
  On obtient : `-1.95996398454 ≃ -1.96`
  Donc Prob(-1.96<U<1.96)=0.975-0.025=0.95=1-0.05 et on a $u_\alpha = 1.96$.
  Comme $U \notin]-u_\alpha, u_\alpha[$, l'hypothèse (H_0) est écartée avec un risque d'erreur

de moins de 5%. Donc on peut dire, avec un risque d'erreur moins de 5%, que le boulanger ne respecte pas le poids de ses pains de 400 gr.

Avec la calculatrice :

On appuie sur `Symb` et on choisit :

`Test d'Hypo`

`Type Z-Test: 1` μ

`Hypoth Alt` $\mu \neq \mu_0$

On appuie sur `Num` et on tape :

`N=100`

μ=400

σ=50.25

α=0.05

On appuie sur `CALC` du bandeau :

On obtient :

`Résultat 0` (Rejet de l'hypothèse à α=0.05)

`Test Z -1.99`

`Test` $\overline{x}$ `390`

`Prob Z 0.0466`

`Crit Z` $\pm$`1.9599`

`Minimum critique 390.151`

`Maximum critique 409.849`

On tape dans `Home` :

`normald_icdf(400,5.025,0.025),`

`normald_icdf(400,5.025,0.975)`

On obtient :

`390.151180978,409.848819022`

– Il s'agit de trouver un intervalle au seuil de 5% de centre $\overline{x}$ dans lequel m se trouve avec une probabilité de 95 %.

On choisit faute de mieux :

$\overline{x} = m_\alpha$

$s = s_\alpha = \sigma\frac{\sqrt{n-1}}{\sqrt{n}}$ et

$s = sigma_\alpha$ donc La moyenne des moyennes des échantillons d'effectif n suit une loi normale de moyenne $\overline{x}$ et d'écart-type $\frac{\sigma}{\sqrt{n}} = \frac{s}{\sqrt{n-1}}$ Donc $\sigma = \frac{500}{\sqrt{99}} \simeq 50.25$.

Avec la calculatrice :

On appuie sur `Symb` et on choisit :

`Inter de Confiance`

`.Type Z-int: 1` μ

On appuie sur `Num` et on tape :

$\overline{x}$ `390`

`N=100`

σ=50.25

`C=0.95`

On appuie sur `CALC` du bandeau :

On obtient :

`C 0.95`

`Crit.Z ± 1.96`
`Minimum critique 380.1511...`
`Maximum critique 399.8488...`
On tape dans `Home` :
`normald_icdf(390,5.025,0.025),`
`normald_icdf(390,5.025,0.975)`
On obtient :
`380.151180978,399.848819022`

20.11.3 Échantillons extraits d'une distribution de Student

On utilise la loi de Student (valable plutôt pour les petits échantillons (n<30)) : La variable $T = \frac{|m_\alpha - m|}{s}\sqrt{n}$ suit une loi de Student de $n-1$ degés de liberté.

Comparaison d'une moyenne expérimentale à une moyenne théorique

On peut faire l'exercice précédent avec la loi de Student :

– On appuie sur `Symb` et on choisit :
  `Test d'Hypo`
  `Type T-Test: 1` μ
  `Hypoth Alt` $\mu \neq \mu_0$
  On appuie sur `Num` et on tape :
  $\overline{x}$=390
  S=50
  N=100
  μ_0=400
  α=0.05
  On appuie sur `CALC` du bandeau :
  On obtient :
  `Résultat 0` (Rejet de l'hypothèse à α=0.05)
  `Test T -2`
  `Test` $\overline{x}$ `390`
  `Prob T 0.048239...`
  `DF=99`
  `Crit T ±1.9842...`
  `Minimum critique 390.0789`
  `Maximum critique 409.921`
  On tape dans `Home` :
  `student_icdf(99,0.025),student_icdf(99,0.975)`
  On obtient :
  `-1.98421695159,1.98421695159`
  On tape dans `Home` :
  `student_icdf(99,0.025)*50/sqrt(100)+400,`
  `student_icdf(99,0.975)*50/sqrt(100)+400`
  On obtient :
  `390.078915242,409.921084758`
– On appuie sur `Symb` et on choisit :

`Inter de Confiance`
`Type T-int: 1` μ
On appuie sur `Num` et on tape :
$\overline{x}$`: 390`
`S:=50`
`N=100`
`C=0.95`
On appuie sur `CALC` du bandeau :
On obtient :
`C 0.95`
`DF=99`
`Crit T` $\pm$`1.9842`
`Min Crit` $\overline{x}$ `380.0789`
`Maxi Crit` $\overline{x}$ `399.9210`
On tape dans `Home` :
`student_icdf(99,0.025)*5+390,`
`student_icdf(99,0.975)*5+390`
On obtient :
`380.078915242,399.921084758`

20.12 L'Aplet Linear Explorer

On tape :

On obtient :

20.13 L'Aplet Quadratic Explorer

On tape :

On obtient :

20.13.1 L'Aplet Trig Explorer

On tape :

On obtient :

20.14 L'Aplet Données du Steamer

Cinquième partie

La programmation avec le CAS

Chapitre 21

Généralités

Les exemples de programmes qui suivent sont des programmes CAS qui utilisent les commandes du CAS.
Pour cela on tape :
`Shift Program` puis,
`Nouv.` du bandeau.
On obtient une fenêtre avec `Nom` et `CAS`.
On coche `CAS` pour écrire un programme CAS et on met comme `Nom` par exemple : `decimales`, puis `OK` et on tape le programme qui donne la partie entière et les p décimales une à une du nombre rationnel f (cf 23.4.2).

Chapitre 22

Les instructions de programmation

22.1 L'instruction d'affectation

Pour stocker un objet dans une variable on utilise `STO` ou `:=`.
Dans HOME on tape per exemple :
`12 STO A`
cela a pour effet de mettre `12` dans la variable `A`.
Dans CAS ou dans un programme CAS on tape par exemple :
`12 STO a`
Ou on tape
`a:=12`
cela a pour effet de mettre `12` dans la variable `a`.
Dans CAS ou dans un programme CAS on peut mettre : `a,b:=12,34` ou
`a,b:=[12,34]`
cela a pour effet de mettre `12` dans la variable `a` et `34` dans la variable `b`.
Par exemple,on tape pour mettre le quotient et le reste de la division euclidienne respectivement dans `q` et`r` en utilisant la commande `iquorem` du CAS :
`q,r:=iquorem(365,12)`
cela a pour effet de mettre `30` dans la variable `q` et `5` dans la variable `r` car $365 = 30 * 12 + 5$.
Attention il faut utiliser cela avec prudence car ces 2 affectations se font simultanément par exemple :
`a:=1;b:=2; a:=a+b;a:=1;b:=a-b;` est équivalent à :
`a:=1;b:=2;c:=a;a:=a+b;b:=c-b;`
donc après `a` vaut `3` et `b` vaut `-1` (et non `1`).
MAIS
`purge(a);a,b:=2,a+1` a pour effet de mettre `2` dans `a` et `3` dans `b`.

22.2 Les instructions conditionnelles

`IF` $< cond >$`THEN` $< inst_1 >$ `END`
Si la condition $< cond >$ est vraie on exécute les instructions $< inst_1 >$ et sinon on ne fait rien.

On tape :
```
3=>X
IF X>0 THEN X+1 END
```
ou
```
IF X>0 THEN X+1;END
```
On obtient :
```
3
```
On tape :
```
-3=>X
IF X>0 THEN X+1 END
```
ou
```
IF X>0 THEN X+1;END
```
On obtient :
```
-3
```

IFTE

Si la condition donnée en premier argument est vraie on exécute le deuxième argument et sinon on exécute le troisième argument.
On tape :
```
3=>X
IFTE(X>0,X+1,X-1)
```
On obtient :
```
4
```
On tape pour définir la valeur absolue :
```
-3=>X
IFTE(X>0,X,-X)
```
On obtient :
```
3
```
On tape :

```
EXPORT ESSAI0(X,A)
BEGIN
RETURN IFTE(X<-ABS(A),-1,IFTE(X<ABS(A),0,1));
END;
```

Puis on tape :
```
ESSAI0(-5,3)
```
On obtient :
```
-1
```
On tape :
```
ESSAI0(-2,3)
```
On obtient :
```
0
```
On tape :
```
ESSAI0(5,3)
```
On obtient :
```
1
```

IF $< cond >$THEN $< inst_1 >$ ELSE $< inst_2 >$ END

Si la condition $< cond >$ est vraie on exécute les instructions $< inst_1 >$ et sinon on exécute les instructions $< inst_2 >$.

On tape :

```
3=>X
IF X>0 THEN X+1 ELSE X-1 END
```

ou

```
IF X>0 THEN X+1; ELSE X-1; END
```

On obtient :

```
3
```

On tape :

```
-3=>X
IF X>0 THEN X+1 ELSE X-1 END
```

ou

```
IF X>0 THEN X+1; ELSE X-1; END
```

On obtient :

```
-4
```

On tape :

```
EXPORT ESSAI(X,A)
BEGIN
IF X<-ABS(A) THEN RETURN -1; END;
IF X<ABS(A) THEN RETURN 0; END;
RETURN 1;
END;
```

Puis on tape :

```
ESSAI(-5,3)
```

On obtient :

```
-1
```

On tape :

```
ESSAI(-2,3)
```

On obtient :

```
0
```

On tape :

```
ESSAI(5,3)
```

On obtient :

```
1
```

CASE ...END

CASE

IF $< cond_1 >$ THEN $< inst_1 >$ END;

IF $< cond_2 >$ THEN $< inst_2 >$ END;

IF $< cond_3 >$ THEN $< inst_3 >$ END;

DEFAULT $< inst_4 >$;

END

On utilise CASE pour éviter de faire des IF imbriqués.

$< cond_1 >$ est évaluée :

- si $< cond_1 >$ est vraie on exécute les instructions $< inst_1 >$ et on termine le `CASE` en faisant les instructions qui suivent le `END` du `CASE`.
- si $< cond_1 >$ est fausse, alors $< cond_2 >$ est évaluée, si elle est vraie on exécute les instructions $< inst_2 >$ et on termine le `CASE` en faisant les instructions qui suivent `END` du `CASE` etc... L'instruction $< inst_4 >$ se fait si les 3 conditions $< cond_1 >$, $< cond_2 >$, $< cond_3 >$ sont fausses.

```
CASE
IF X<-1 THEN -1=>R; END;
IF X<1; THEN 0=>R; END;
IF X>=1 THEN 1=>R; END;
END;
R;
```

ou bien :

```
CASE
IF X<-1 THEN -1=>R; END;
IF X<1 THEN 0=>R; END;
DEFAULT 1=>R;
END;
R;
```

`IFERR`

La syntaxe est :

`IFERR <inst0> THEN <inst1> ELSE <inst2> END`

Si une erreur est dédectée dans les instructions `<inst0>`, on effectue les instructions `<inst1>` et sinon on effectue les instructions `<inst2>`.

On tape (par exemple si on ne sait pas l'ordre des arguments de la commande `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
```

On obtient :

4

`CONTINUE`

Lorsque `CONTINUE;` se trouve parmi les instructions d'une itération, elle a pour effet de sauter les instructions qui la suivent pour aller à la prochaine itération.

On tape pour calculer 1+2+4+5=$\sum_{j\neq 3 et j=1}^{5} j$:

```
A:=0;
FOR J FROM 1 TO 5 DO
IF J==3 THEN CONTINUE; END;
A:=A+J;
END}\\
```

On obtient :
```
12
```

22.3 Les boucles

```
FOR FROM TO DO END
```
ou
```
FOR FROM TO STEP DO END
```
On tape :
```
S:=0; FOR J FROM 1 TO 5 DO S:=S+J;END
```
On obtient :
```
15
```
car 1+2+3+4+5=15
On tape :
```
S:=0; FOR J FROM 2 TO 10 STEP 2 DO S:=S+J;END
```
On obtient :
```
30
```
car 2+4+6+8+10=30

```
WHILE DO END
```
On tape :
```
A:=1; WHILE A<=1 DO A:=A+1; END;A;
```
On obtient :
```
2
```
On tape :
```
S:=0;J:=1;WHILE J<=5 DO S:=S+J;J:=J+1; END;S
```
On obtient comme valeur de `S` :
```
15
```

```
REPEAT UNTIL
```
On tape :
```
A:=1; REPEAT A:=A+1 UNTIL A>1;A;
```
On obtient comme valeur de `A` :
```
2
```
On tape :
```
A:=1; REPEAT A:=A+1 UNTIL A>4;A;
```
On obtient comme valeur de `A` :
```
5
```

```
BREAK
```
On tape :
```
BREAK
```
On obtient :
```
La sortie d'une boucle
```
Par exemple pour avoir la valeur approchée de la somme $6\sum_{j=1}^{\infty} 1/j^2$, on décide de ne pas additionner les termes inférieurs à P et de ne pas faire plus de 100 additions.

On tape

```
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;
```

On tape :
`PI2S6(0.001)`
On obtient :
`1.61319070033`
On tape :
`PI2S6(0.0001)`
On obtient :
`1.63498390018`
On tape :
`PI^2/6`
On obtient :
`1.64493406685`

22.4 Les commentaires

`//`
`//` commence une ligne destinée à faire un commentaire.

22.5 Les variables

En programmation les variables ont comme noms une suite de lettres ou chiffres commencant par une lettre.
Les variables qui sont locales au programme seront déclarées en utilisant le mot clé `LOCAL`, par exemple : `LOCAL A,B,AB,x;`.
Dans ce cas les variables sont initialisées à 0.
Pour avoir une variable formelle on écrira : x :='x'

22.6 Les opérateurs booléens

`<,<=,==,!=,>,>=`
`<,<=,>,>=` sont des opérateurs booléens infixés testant l'inégalité.
`==` est un opérateur booléen infixé testant l'égalité.

<> ou ! = ou $\neq$ est un opérateur booléen infixé testant la non égalité.

`AND`
`AND` est l'opérateur booléen infixé et.
On tape :
`1 AND 0`
On obtient :
`0`
On tape :
`1 AND 1`
On obtient :
`1`
On tape :
`0 AND 0`
On obtient :
`0`

`NOT`
`NOT` renvoie l'inverse logique de l'argument.
On tape :
`NOT 1`
On obtient :
`0`
On tape :
`NOT 0`
On obtient :
`1`

`OR`
`OR` est l'opérateur booléen infixé ou.
On tape :
`1 OR 0`
On obtient :
`1`
On tape :
`1 OR 1`
On obtient :
`1`
On tape :
`0 OR 0`
On obtient :
`0`

`XOR`
`XOR` est l'opérateur booléen infixé ou exclusif.
On tape :
`1 XOR 0`
On obtient :

`1`
On tape :
`1 XOR 1`
On obtient :
`0`
On tape :
`0 XOR 0`
On obtient :
`0`

On tape :

On obtient :

22.7 Les commandes d'invite

`CHOOSE`
On tape pour choisir la valeur de `A` parmi les 3 valeurs, (1,2,3) :
`CHOOSE(A,"TITRE:A=","UN","DEUX","TROIS")`
On obtient :
`L'ouverture d'une boite de dialogue comportant 3 items :`
`si on clique sur le 1ier (resp 2nd, 3ième) item, cela stocke`
`1 (resp 2, 3) dans A`

`DISP`
On tape :
`DISP("A="+A)`
On obtient :

`FREEZE`
On tape :
`FREEZE`
On obtient :
`un gel de l'écran, on appuie sur une touche pour degeler`

`GETKEY`
On tape :
`A:=GETKEY`
On obtient, si on a appuyer sur `.` :
`48`
On tape :
`A:=GETKEY`
On obtient, si on n'a pas pressé de touche :
`-1`

ISKEYDOWN

On tape :

```
ISKEYDOWN(48)
```

On obtient, si on n'a pas pressé la touche . :

```
0
```

On tape :

```
ISKEYDOWN(48)
```

On obtient si on a pressé la touche . :

```
1
```

INPUT

On tape :

```
INPUT(C,"TITRE:C=")
```

On obtient :

```
Un écran permettant d'entrer une valeur qui sera stockée
dans la variable C
```

MSGBOX

On tape :

```
A:=3
MSGBOX(2*A)
```

ou

```
MSGBOX(2*A,0)
```

On obtient :

```
6 et OK dans le bandeau
```

Si on appuie sur `OK` alors `MSGBOX(2*A)` ou `MSGBOX(2*A,0)` renvoie 1.

On tape :

```
A:=3
MSGBOX(2*A,1)
```

On obtient :

```
6 et CANCEL et OK dans le bandeau
```

On tape :

Si on appuie sur `CANCEL` alors `MSGBOX(2*A,1)` renvoie 0.

Si on appuie sur `OK` alors `MSGBOX(2*A,1)` renvoie 1.

On tape :

```
A:=3
MSGBOX("A= "+A)
```

On obtient :

```
"A= 3"
```

PRINT

On tape :

```
A:=3
PRINT(A)
```

On obtient :

```
A:3
```

`WAIT`
On tape :
`WAIT(5)`
On obtient :
`Une interruption du programme pendant 5 secondes`

`EDITMAT`
On tape :
`EDITMAT(M)`
On obtient :
`L'ouverture d'un éditeur de matrice permettant d'entrer la matrice M`

Une utilisation de `GETKEY` et de `ISKEYDOWN`.
Le programme suivant permet de connaitre le code des touches sur lesquelles on appuie et se termine lorsqu'on appuie sur la touche `.`.

```
EXPORT AA()
BEGIN
LOCAL A,L;
L:=[];
REPEAT
REPEAT
A:=GETKEY;
UNTIL A!=-1;
L:=CONCAT(L,A);
UNTIL ISKEYDOWN(48);
RETURN L;
END;
```

On tape :
`AA()` puis `ENTER 1230.`
On obtient :
`[42,43,44,47,48]`

On peut aussi écrire le programme suivant qui donnera le même résultat :

```
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;
```

On tape :
`AAA()` puis `ENTER 1230.`
On obtient :
`[42,43,44,47,48]`

Un exemple un lancer de dés : 2 joueurs `A` et `B` lancent à tour de rôle 2 dés et comptent leur scores `SA` et `SB` en éliminant du score certaines parties (par exemple les parties qui contiennt un 6). Ils décident de s'arreter de jouer au bout de 2mn.
Le programme va afficher le résultat du jet des 2 dés pour chacun des 2 joueurs. On remarquera que l'affichage du jet du joueur `B` : `MSGBOX(N+1+":B="+B,1))` a comme deuxième paramètre `1` et donc on a dans le bandeau `CANCEL` et `OK`. Si on appuie sur `CANCEL`, `MSGBOX(N+1+":B="+B,1)` renvoie `0` et si on n'appuie pas sur `CANCEL`, `MSGBOX(N+1+":B="+B,1)` renvoie `1`. Donc si on appuie sur `CANCEL` on annule la partie.
Pour arrêter de jouer il suffit d'appuyer sur `OK` (touche 5).
Le programme affiche alors le score et la liste des différents coups.

```
EXPORT DEUXDES()
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;
```

22.8 Les commandes des Applications

`CHECK`

On tape si l'aplet courante est `Fonction` :
`F2(X):=COS(X)+X`
`CHECK(2)`
On obtient :
`La définition de F2 dans l'aplet Fonction et la fonction F2 est cochée`

`UNCHECK`
On tape si l'aplet courante est `Fonction` :
`UNCHECK(2)`
On obtient :
`La fonction F2 est décochée`

`SELECT`
On tape :
`SELECT`
On obtient :

`STARTAPLET`
On tape :
`STARTAPLET`
On obtient :

`STARTVIEW`
On tape :
`STARTVIEW(1)`
On obtient :

noir :0 gris fonce :1 gris clair :2 blanc :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 Aplet Note Editor : -5 MatrixCatalog : -6 ListsCatalog : =-8 ProgramCatalog : =-10 NoteCatalog : =-12

Chapitre 23

Pour apprendre à programmer

23.1 L'instruction conditionnelle IF

Trois magasins vendant les mêmes laines au même prix unitaire de p euros, font une promotion.
Dans le magasin1 la promotion précise :
- réduction de 10 % si vous achetez plus de 5 pelotes mais moins que 10,
- réduction de 20 % si vous achetez au moins 10 pelotes.

Dans le magasin2 la promotion précise :
- 1 gratuite pour 8 pelotes achetèes
- 2 pelotes gratuites pour 13 pelotes achetèes.

Dans le magasin3, la promotion précise :
- réduction de 10 % si vous achetez 5 pelotes,
- réduction de 20 % si vous achetez 10 pelotes,
- par exemple si vous achetez 7 pelotes à p euros l'unité, vous avez la réduction seulement sur 5 pelotes et vous payez $2 * p + 5 * 0.9 * p = 6.5 * p$ et si vous achetez 17 pelotes vous avez la réduction seulement sur 10 pelotes et sur 5 pelotes et vous payez $2 * p + 5 * 0.9 * p + 10 * 0.8 * p = 14.5 * p$.

Vous avez besoin de 9 pelotes, quel magasin choisissez-vous ?
Vous avez besoin de 15 pelotes, quel magasin choisissez-vous ?

Faire le programme `prix(n,p)` pour xchaque magasin qui donne le prix que vous allez payer, le nombre de pelotes et l'économie réalisée grâce à la promotion, lors de l'achat de `n` pelotes au prix unitaire de `p`. **Les programmes** prix1

```
(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:;
```

prix2

```
(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:;
```

prix3

```
(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*p
  END:;
```

23.2 Les boucles FOR et WHILE

23.2.1 Faire compter la calculatrice de 1 en 1 en utilisant un affichage

On veut que la calculatrice affiche : `0` puis `1` puis `2` etc..

Sans programme

On tape :
`n:=0;`
Puis `Enter`
On tape ensuite :
`n:=n+1`
Puis `Enter`,`Enter` etc ...
On obtient (chaque `Enter` fait afficher le nombre suivant) :
`1`, puis `2` etc ...

Avec un programme du CAS

La boucle `FOR`
On donne comme nom de programme `comptef` et on coche `CAS`.
On tape, pour que s'affiche la suite `0,1,2..p`, le programme qui utilise une boucle `FOR` :

```
 (p)->BEGIN
  LOCAL n;
  FOR n FROM 0 TO p DO
      PRINT(n);
```

```
END;
RETURN n;
END;
```

On remarquera que :
n est initialisé par la valeur qui suit FROM et la valeur qui suit TO sert à faire le test d'arrêt de la boucle,
l'instruction n:=n+1, puis le test n<=p se fait automatiquement dans une boucle FOR. La boucle s'arrête au premier entier n strictement supérieur à p.
On tape dans le CAS :
comptef(-1)
On obtient :
0 car le test n<=p se fait en début de boucle.
On tape dans le CAS :
comptef(4)
On obtient :

```
0
1
2
3
4
```

puis 5 car 5 est le premier entier strictement supérieur à p=4
En résumé
FOR initialise la variable du FOR fait le test si le test est vrai il exécute le corps de la boucle (i.e. toutes les instructions jusqu'au END du FOR) puis la variable du FOR est incrémentée, puis il fait le test : si le test est vrai il exécute le corps de la boucle etc.. ; et si le test est faux il exécute les instructions qui suivent le END du FOR.
La boucle WHILE
On donne comme nom de programme comptew et on coche CAS.
On tape, pour que s'affiche la suite 0,1,2..p, le programme qui utilise une boucle WHILE :

```
(p)->BEGIN
 LOCAL n;
  n:=0;
  WHILE n <=p DO
     PRINT(n);
     n:=n+1;
 END
 RETURN n;
 END;
```

On remarquera que :
n doit être initialisé avant le commencement de la boucle,
WHILE effectue le test n<=p : si le test est vrai il exécute le corps de la boucle (i.e. toutes les instructions jusqu'au END du WHILE) mais attention il faut changer

dans le corps de la boucle la valeur d'au moins une variable du test pour que le test devienne faux à un moment afin de ne pas avoir une boucle infinie, c'est ici l'instruction `n:=n+1`. Puis `WHILE` effectue le test `n<=p` si le test estvrai il exécute le corps de la boucle etc..., et si le test est faux il exécute les instructions qui suivent le `END` du `WHILE`.
La boucle s'arrête au premier entier `n` strictement supérieur à `p`.
On tape dans le CAS :
`comptew(-1)`
On obtient :
`0` car le test `n<=p` se fait en début de boucle.
On tape dans le CAS :
`comptew(4)`
On obtient :

```
0
1
2
3
4
```

puis
`5` car 5 est le premier entier strictement supérieur à `p=4`

23.2.2 Faire compter la calculatrice de 1 en 1 en utilisant une liste ou une séquence

Plutôt qu'afficher les nombres avec la commande `PRINT` on va mettre ces nombres dans un liste ou une séquence.

Avec une liste

La liste vide est `[]` et la commande `l:=append(l,a)` met l'élément `a` à la fin de la liste `l`. **Avec une boucle** `FOR`
On donne comme nom de programme `comptelf` et on coche `CAS`.
On tape, pour que s'affiche la liste `[0,1,2..p]`, le programme qui utilise une boucle `FOR` :

```
(p)->BEGIN
 LOCAL n,l;
 l:=[];
 FOR n FROM 0 TO p DO
    l:=append(l,n);
 END;
 RETURN l;
 END;
```

Avec une boucle `WHILE`
On donne comme nom de programme `comptelw` et on coche `CAS`.
On tape, pour que s'affiche la liste `[0,1,2..p]`, le programme qui utilise une boucle `WHILE` :

```
 (p)->BEGIN
  LOCAL n,l;
  l:=[];
  n:=0;
   WHILE n <=p DO
     l:=append(l,n);
     n:=n+1;
 END
  RETURN l;
  END;
```

On tape :
`comptelf(4)` ou `comptelw(4)`
On obtient :
`[0,1,2,3,4]`

Avec une séquence

La séquence vide est `NULL` et la commande `l:=l,a` met l'élément `a` à la fin de la séquence `l`. **Avec une boucle** `FOR`
On donne comme nom de programme `comptesf` (ou on modifie le programme précédent) et on coche `CAS`.
On tape, pour que s'affiche la liste `[0,1,2..p]`, le programme qui utilise une boucle `FOR` :

```
 (p)->BEGIN
  LOCAL n,l;
  l:=NULL;
  FOR n FROM 0 TO p DO
     l:=l,n;
  END;
  RETURN l;
  END;
```

Avec une boucle `WHILE`
On donne comme nom de programme `comptesw` (ou on modifie le programme précédent) et on coche `CAS`.
On tape, pour que s'affiche la liste `[0,1,2..p]`, le programme qui utilise une boucle `WHILE` :

```
 (p)->BEGIN
  LOCAL n,l;
  l:=NULL;
  n:=0;
   WHILE n <=p DO
     l:=l,n;
     n:=n+1;
 END;
  RETURN l;
  END;
```

On tape :
`comptesf(4)` ou `comptesw(4)`
On obtient :
`0,1,2,3,4`

23.3 Valeur approchée de la somme d'une série

23.3.1 Série de terme général $u_n = 1/n^2$

Sans programme

On tape dans le CAS :
`s:=0;n:=1;`
Puis `Enter`
On tape ensuite :
`s:=s+1/n^2;n:=n+1`
Puis `Enter`,`Enter` etc ...
On obtient (chaque `Enter` fait une addition de plus) :
`[1,2]`
`[5/4,3]`
`[49/36,4]`
`[205/144,5]`
`[5269/3600,6]`
`[5369/3600,7]`
`[266681/176400,8]` etc...

Avec un programme du CAS

On donne comme nom de programme `sommeu` et on coche `CAS`.
On tape, pour avoir la somme des $1 + 1/2^2 + ... + 1/1/p^2$, le programme :

```
(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;
```

On tape dans le CAS :
`sommeu(7)`
On obtient :

```
[1,1]
[5/4,2]
[49/36,3]
[205/144,4]
```

```
[5269/3600,5]
[5369/3600,6]
[266681/176400,7]
```

```
[266681/176400,8]
```

On remarquera qu'il faudrait mettre `PRINT([s,n+1]);` pour avoir les mêmes résultats car dans le `FOR` l'incrémentation de `n` se fait quand le corps de la boucle a été exécuté. Mais le résultat final est le même car l'incrémentation de `n` a été faite et comme 8>7 le `FOR` s'arrête.

23.3.2 Série de terme général $v_n = (-1)^(n+1)/n$

Sans programme

On tape dans le CAS :
`s:=0;n:=1;`
Puis `Enter`
On tape ensuite :
`s:=s+(-1)^(n+1)/n;n:=n+1`
Puis `Enter`,`Enter` etc ...
On obtient (chaque `Enter` fait une opération de plus) :
`[1,2]`
`[1/2,3]`
`[5/6,4]`
`[7/12,5]`
`[47/60,6]`
`[37/60,7]`
`[319/420,8]` etc

Avec un programme du CAS

On donne comme nom de programme `sommev` et on coche `CAS`.
On tape, pour avoir la somme des $1-1/2+1/3...+(-1)^(p+1)/p$, le programme :

```
(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;
```

On tape dans le CAS :
`sommeu(7)`
On obtient :

```
[1,1]
[1/2,2]
```

```
[5/6,3]
[7/12,4]
[47/60,5]
[37/60,6]
[319/420,7]

[319/420,8]
```

23.3.3 La série de terme général $w_n = 1/n$ est divergente

Pour montrer cela on montre que pour $p > 1$ on a $\sum_{n=2^p-1}^{2^p} 1/k \geq 1/2$

Sans programme

On tape dans le CAS :
`s:=1;n:=1;k:=0;`
Puis `Enter`
On tape ensuite :
`s:=s+sum(1/k,k=n+1..2*n);n:=2*n;k:=1+k;1+k/2<=s`
Puis `Enter`,`Enter` etc ...
On obtient (chaque `Enter` fait une opération de plus) :
`[3/2,2,1]`
`[25/12,4,1]`
`[761/280,5,1]`
`[2436559/720720,6,1]` etc

Avec un programme du CAS

On donne comme nom de programme `sommediv` et on coche `CAS`.
On tape, dans le CAS, pour avoir la somme des `n` :

```
sommediv(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;
```

On tape dans le CAS :
`sommediv(7)`
On obtient :

```
3/2,2,1,vrai]
[25/12,4,2,vrai]
[761/280,8,3,vrai]
[2436559/720720,16,4,vrai]
```

puis, `[2436559/720720,16,5,vrai]` **Exercice** Pour quelle valeur de n $\sum k = 1^n 1/k > p$? On tape, dans le CAS, pour avoir la somme des `n` :

```
 sommesup(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;
```

On tape dans le CAS :
`sommesup(4)`
On obtient :
`4.02724519544,31`

23.4 Écriture décimale d'une fraction

23.4.1 Sans programme

Par exemple on veut trouver le premières décimale de $f = \frac{355}{113}$ (f est une fraction qui donne π avec 6 décimales exactes !) On tape dans le CAS :
`f:=355/113;f1:=floor(f);l:=f1;n:=numer(f-f1);d:=denom(f-f1);`
Puis `Enter`
On tape ensuite :
`ds,n:=iquorem(10*n,d); L:=L,ds;`
Puis `Enter`,`Enter` etc ... On obtient (chaque `Enter` donne une décimale de plus) :

```
[[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]
```

Les décimales obtenus sont dans la liste `l` qui commence par la partie entière de la fraction `f`.
Ou bien on tape :
`f:=355/113;f1:=floor(f);L:=f1;n:=numer(f-f1);d:=denom(f-f1);`
Puis `Enter`
On tape ensuite :
`ds,n:=iquorem(10*n,d):; L:=L,ds:;`

Puis `Enter`,`Enter` etc ... On obtient :
`["Done","Done"],["Done","Done"]` etc
Puis on tape :
`l`
On obtient après 30 `Enter` :
`[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]`
Comme c'est difficile de compter le nombre de `Enter`, on peut faire afficher par exemple 5 décimales supplémentaire chaque fois que l'on appuie sur `Enter`. Le premmier `Enter` affichera la partie entière suivie de 5 décimales. On peut aussi vouloir ou ne pas vouloir afficher ds et n.
On tape :
`f:=355/113;f1:=floor(f);L:=f1;n:=numer(f-f1);d:=denom(f-f1);`
Puis `Enter` puis,
`ds,n:=iquorem(105*n,d); l:=l*10^5+ds;`
ou bien si on ne veut pas les valeurs de ds et n :
`ds,n:=iquorem(10^5*n,d):; l:=l*10^5+ds;`
On obtient après le premier `Enter` :
`[14159 33],314159`
(ou `["Done","Done"],314159`)
On remarquera que : $113 * 314159 + 33 = 355 * 10^5$
Si on appuie 8 fois sur enter on obtient un nombre ayant 41 chiffres : la partie entière 3 suivi des 40 décimales de 355/113 :
`[5309 83],3141592920353982300884955752212389380530 9` Si on tape :
`l`
On obtient :
`3141592920353982300884955752212389380530 9`
Remarquez que 5309 n'a que 4 chiffres c'est donc que les dernières décimales sont : 05309 et on a :
$113 * l + 83 = 355 * 10^{40}$

23.4.2 Avec un programme du CAS

On donne comme nom de programme `decimales` et on coche `CAS`.
Le programme `decimales` renvoie une séquence `l` qui donne la partie entière (`f1`) et les `p` premières decimales (`ds`) d'une fraction `f`. On utilise les fonctions :
`floor` qui donne la partie entière s'un nombre,
`numer` qui donne le numérateur d'une fraction simplifiée,
`denom` qui donne le dénominateur d'une fraction simplifiée,
`iquorem` qui donne le quotient et le reste de la division euclidienne `ds,n:=iquorem(10*n,d);` est équivalent à :
`ds:=iquo(10*n,d);` (pour avoir le quotient de `10*n` par `d`) et `n:=irem(10*n,d);` (pour avoir le reste de `10*n` par `d`).
On tape, `Shift Program` puis `Nouv.` du bandeau.
On obtient une fenêtre avec `Nom` et `CAS`. On coche `CAS` et met comme `Nom` : `decimales` puis `OK` et on tape le programme qui donne la partie entière et les `p` décimales une à une du nombre rationnel `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;
```

On tape dans le CAS :
`decimales(355/311,20)`
On obtient :
`1,1,4,1,4,7,9,0,9,9,6,7,8,4,5,6,5,9,1,6,3`
On tape dans le CAS :
`decimales(355/113,20)`
On obtient une approximation de π à $3 * 10^{-7}$ pès :
`[3,1,4,1,5,9,2,9,2,0,3,5,3,9,8,2,3,0,0,8,8]`
On a en effet : `evalf(pi)` renvoie 3.1415926536 On peut déterminer les décimales par p groupes de g décimales et renvoyer un nombre entier l. L'écriture décimale de la fraction est alors $l * 10^{(} - p * g)$. On donne comme nom de programme `decimalesg` et on coche `CAS` (ou on modifie le programme `decimales`) :

```
 (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;
```

On tape dans le CAS :
`decimalesg(355/311,8,5)`
On obtient :
`11414790996784565916398713826366559485530` On tape dans le CAS :
`decimalesg(355/113,8,5)`
On obtient :
`31415929203539823008849557522123893805309`

23.5 La mèthode de Newton et l'algorithme de Héron

23.5.1 La méthode de Newton

Soit f deux fois dérivable ayant un zéro et un seul r dans l'intervalle $[a\ ;\ b]$. Supposons de plus que f' et f'' ont un signe constant sur $[a\ ;\ b]$. La méthode de Newton consiste à approcher r par l'abscisse x_1 du point commun à Ox et à la tangente en un point M_0 du graphe de f. Si M_0 a pour coordonées $(x_0, f(x_0))$ ($x_0 \in [a\ ;\ b]$), la tangente en M_0 a pour équation :
$y = f(x_0) + f'(x_0) * (x - x_0)$ et donc on a :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

On peut alors réitérer le processus, et on obtient une suite x_n qui converge vers r soit par valeurs supérieures, si $f' * f'' > 0$ sur $[a\ ;\ b]$ (i.e. si $f'(r) > 0$ et si f est convexe ($f'' > 0$ sur $[a\ ;\ b]$) ou si $f'(r) < 0$ et si f est concave ($f'' < 0$ sur $[a\ ;\ b]$)) soit par valeurs inférieures, si $f' * f'' < 0$ sur $[a\ ;\ b]$ (i.e. si $f'(r) < 0$ et si f est convexe ($f'' > 0$ sur $[a\ ;\ b]$) ou si $f'(r) > 0$ et si f est concave ($f'' < 0$ sur $[a\ ;\ b]$)). **L'algorithme de Héron** est un cas particulier de l'application de la méthode de Newton pour chercher les valeurs approchées de $\sqrt{a}$ pour a entier.
Dans ce cas $\sqrt{a}$ est un zéro de $f(x) = x^2 - a$ et $g(x) = f'(x) = 2x$ donc la suite des itérés est donné par :

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2}(x_n - \frac{a}{x_n})$$

23.5.2 Traduction de l'algorithme de Newton

La fonction `newton_rac` renvoie la valeur approchée à p près de la racine de $f(x) = 0$ on commençant l'itération avec $x0$.
On remarquera que le paramètre f est une fonction et donc, que sa dérivée est la fonction `g:=function_diff(f)`.
On cherche une valeur approchée donc il faut écrire :
`x0:=evalf(x0)` car si on ne met pas `evalf`, les calculs de l'itération se feront excactement et seront vite compliqués.
On regarde au début si la suite des x_i ($i = 0..n$) est croissante ou décroissante à partir de $n = 1$ en comparant au début x_1 et x_2. On donne comme nom de programme `newtonrac` et on coche `CAS`.
Le programme `newtonrac` donne un zéro de `f` proche de `x0` à `p` près.
On tape le programme :

```
(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;
```

On tape dans le CAS :
`f(x):=cos(x)-x`
`newtonrac(f,0.4,1e-10)`
On obtient :
`0.739085133215` On tape dans le CAS :
`cos(0.739085133215)-0.739085133215`
On obtient :
`2.70006239589e-13`

23.5.3 Traduction de l'algorithme de Héron

On donne comme nom de programme `heron` et on coche `CAS`.
Le programme `heron` donne une fraction qui approche `sqrt(a)` à `p` près lorsque `x0` est proche de `sqrt(a)`.
On tape

```
(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;
```

On tape dans le CAS :
`heron(2,3/2,10^-10)`
On obtient :
`66587/470832`
On tape dans le CAS :
`decimalesg(66587/470832,2,5)` On obtient :
`14142135623`
On tape dans le CAS :
`f:=heron(2,2,10^-40)`
On obtient :
`157258404803291863335321 7/111198484434986813793811 2`
On tape dans le CAS :
`r2:=decimalesg(f,8,5)` On obtient :
`1414213562373095048801688724209698078569 6`
et $sqrt2 \simeq r2 * 10^-40$ La librairie des flottants longs n'est pasimplémentés dans la HPPrime.

On utilise le logiciel `Xcas` pour vérifié.
On tape dans `Xcas` :
`evalf(r2*10^-40,41)`
On obtient :
`1.4142135623730950488016887242096980785696`
On tape dans `Xcas` :
`evalf(sqrt(2),41)`
On obtient car `Xcas` arrondit la dernière décimale :
`1414213562373095048801688724209698078569 7`

Chapitre 24

Exemple de programmes

24.1 Le PGCD et l'identité de Bézout depuis Home

24.1.1 Le PGCD

On utilise l'algorithme d'Euclide : On tape :

```
EXPORT PGCD(A,B)
BEGIN
LOCAL R;
WHILE B<>0 DO
R:=A MOD B;
A:=B;
B:=R;
END;
RETURN A;
END;
```

Ou bien on utilise la fonction `irem` qui renvoie le reste de la division euclidienne :

```
EXPORT PGCD(A,B)
BEGIN
LOCAL R;
WHILE B<>0 DO
R:=CAS.irem(A,B);
A:=B;
B:=R;
END;
RETURN A;
END;
```

On tape :
`PGCD(45,25)`
On obtient : `5`.

24.1.2 L'identité de Bézout pour A et B

On utilise l'algorithme d'Euclide et les variables `U,V,R` qui vont évoluer de façon qu'à l'ètape k on ait $A * U_k + B * V_k = R_k$.
Ainsi quand R_p est le pgcd de A et B, on aura :
$A * U_p + B * V_p =$pgcd(A, B).
Au début on a :
(1) $A = U_1 * A + V_1 * B$ $(R_1 = A, U_1 = 1, V_1 = 0)$
(2) $B = U_2 * A + V_2 * B$ $(R_2 = B, U_2 = 0, V_2 = 1)$
On veut avoir :
$R_3 = U_3 * A + V_3 * B$
comme $R_3 = A - B * Q_3$ (avec Q_3 quotient entier de $A = R_1$ par $B = R_2$) on trouve en faisant $(1) - Q_3 * (2)$:
$U_3 = U_1 - Q_3 * U_2$ et $V_3 = V_1 - Q_3 * V_2$
et ainsi $R_3 = U_3 * A + V_3 * B$. et à chaque étape on aura $R_k = U_k * A + V_k * B$ avec les relations :
$U_k = U_{k-2}1 - Q_k * U_{k-1}$ et $V_k = V_{k-2} - Q_k * V_{k-1}$.
Pour écrire le programme on a besoin de 3 listes L_1, L_2, L_3 qui seront 3 étapes successives de $[U_k, V_k, R_k]$.
Au début :
$L1 = 1, 0, R1$ $(R1 = A)$
$L2 := 0, 1, R2$ $(R2 = B)$
On calcule $L3$:
$L3$ est obtenu à partir de $L1$ et de $L2$ et si $Q3, R3$:=iquorem$(R1, R2)$, on $R3 = R1 - R2 * Q3$ et donc $L3 = L1 - Q3 * L2$.
Puis, $R1$ prend la valeur de $R2$ et $L1$ la valeur de $L2$ et
$R2$ prend la valeur de $R3$ et $L2$ la valeur de $L3$ ...etc
On s'arrete quand $R2 = 0$ et alors $R1 =$pgcd(A, B).
On tape :

```
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;
```

On peut alléger le nombre de 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;
```

On tape :
`BEZOUT(45,10)`
On obtient : `1,-4,5`.
Ce qui veut dire que `1*45-4*10=5=PGCD(45,25)` On tape :
`BEZOUT(45,25)`
On obtient : `-1,2,5`.
Ce qui veut dire que `-1*45+2*25=5=PGCD(45,25)`

24.2 Le PGCD et l'identité de Bézout depuis CAS

24.2.1 Le PGCD avec le CAS sans faire de programme

On peut mettre en œuvre l'algorithme d'Euclide en utilisant la touche `Enter` de la calculatrice. On tape sur une ligne :
`a:=72;b:=33;` puis sur la touche `Enter`
Puis, on tape sur une ligne :
`r:=irem(a,b);a:=b;b:=r;` puis sur la touche `Enter` plusieurs fois jusqu'a ce que la dernière valeur soit nulle.
Lorsque la dernière valeur est nulle, le pgcd de `a` et `b` est la valeur `3` situé au dessus du 0.
On peut vérifier car la commande `gcd` donnant le pgcd existe :
`gcd(72,33)` renvoie bien `3`.

24.2.2 Le PGCD avec un programme CAS

On utilise la fonction `irem` pour écrire l'algorithme d'Euclide.
On coche `CAS` et on tape le nom du programme : `pgcd` et on tape :

```
(a,b)->
BEGIN
LOCAL r;
WHILE b<>0 DO
r:=irem(a,b);
a:=b;
```

```
b:=r;
END;
RETURN(a);
END;
```

On tape :
`pgcd(45,25)`
On obtient : `5`.

24.2.3 L'identité de Bézout avec le CAS sans faire de programme

On peut mettre en œuvre l'algorithme donnant les coefficients de l'identité de Bézout en utilisant la touche `Enter` de la calculatrice. On tape sur une ligne :
`a:=72;b:=33;l1:=[1,0,a];l2:=[0,1,b]` puis sur la touche `Enter`
Puis, on tape sur une ligne :
`q:=iquo(l1(3),l2(3));l3:=l1-q*l2;l1:=l2;l2:=l3;` puis sur la touche `Enter` plusieurs fois jusqu'a ce que la dernière valeur de la dernière liste soit nulle.
Lorsque cette dernière valeur est nulle, l'identidé de Bezout `[-5,11,3]` est la dernière liste située au dessus du 0 : cela veut dire que `-5*72+11*33=3]`.
On peut vérifier car la commande `iegcd` donnant l'identidé de Bezout existe :
`iegcd(72,33)` renvoie bien `[-5,11,3]`.

24.2.4 L'identité de Bézout avec un programme CAS

On utilise l'algorithme d'Euclide et les variables `u,v,r` qui vont évoluer de façon qu'à l'ètape k on ait $a * u_k + b * v_k = r_k$.
Ainsi quand r_p est le pgcd de a et b, on aura :
$a * u_p + b * v_p =$pgcd(a, b).
Pour écrire le programme on a besoin de 3 listes L_1, L_2, L_3 qui seront 3 étapes successives de $[u_k, v_k, r_k]$.
Au début :
$L1 = 1, 0, r1$ ($r1 = a$)
$L2 := 0, 1, r2$ ($r2 = b$)
On calcule $L3$:
$L3$ est obtenu à partir de $L1$ et de $L2$ et si $q3, r3$:=iquorem$(r1, r2)$, on a : $r3 = r1 - r2 * q3$ et donc $L3 = L1 - q3 * L2$.
Puis, $r1$ prend la valeur de $r2$ et $L1$ la valeur de $L2$ et
$r2$ prend la valeur de $r3$ et $L2$ la valeur de $L3$ ...etc
On s'arrete quand $e2 = 0$ et alors $r1 =$pgcd(a, b).
On tape :

```
(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));
l3:=l1-q*l2;
```

```
l1:=l2;
l2:=l3;
END;
RETURN l1;
END;
```

On tape :
`BEZOUT(45,10)`
On obtient : `1,-4,5`.
Ce qui veut dire que `1*45-4*10=5=PGCD(45,25)` On tape :
`BEZOUT(45,25)`
On obtient : `-1,2,5`.
Ce qui veut dire que `-1*45+2*25=5=PGCD(45,25)`