Algorithmique et traduction pour Xcas

Renée De Graeve

8 septembre 2019

Chapitre 1

Vue d'ensemble de Xcas pour le programmeur

1.1 Installation de Xcas

Le programme Xcas est un logiciel libre écrit en C++, (disponible sous licence GPL). La version à jour se récupère sur :
http : //www − fourier.ujf − grenoble.fr/~parisse/giac_fr.html ou
ftp://fourier.ujf-grenoble.fr/xcas
où l'on trouve le code source (giac.tgz) ou des versions précompilées pour Linux (PC ou ARM), Windows, Mac OS.

1.2 Les différents modes

Xcas propose un mode de compatibilité avec Maple, MuPAD et la TI89/92 : pour cela, il suffit de le spécifier dans Prog style du menu de configuration du cas (bouton Config ou menu Cfg->Configuration du CAS) ou avec le menu Cfg->Mode (syntax). On peut choisir, en cliquant sur la flèche située à coté de Prog style : Xcas ou Maple ou MuPAD ou TI89/92.
On a aussi la possibilité d'importer une session Maple ou une archive TI89/92 en choisissant Importer du menu Fich, ou importer dans un niveau éditeur de programmes un fichier écrit en syntaxe Maple, Mupad ou TI89/92 par le menu Prog->Inserer.

On présente ici le mode Xcas qui est proche de la syntaxe C. On a aussi la possibilité d'avoir toutes les instructions en français de façon à être proche du langage Algorithmique.

1.3 Éditer, sauver, exécuter un programme avec la syntaxe Xcas

On édite un programme ou un script (i.e. une suite de commandes séparées par des ;) avec son éditeur préféré : on peut écrire, dans un même fichier, la définition de plusieurs fonctions séparées par des points virgules (;) (que l'on sauve par exemple sous le nom de bidon), puis dans Xcas on tape :read("bidon");

et cela a pour effet, de compiler les différentes fonctions de `bidon`, de les mettre comme réponse (avec `Success..` dans la zone des résultats intermédiaires pour indiquer les fonctions valides).
En reéditant le programme, ou le script, avec son éditeur préféré, on peut le corriger, le sauver sous un autre nom etc..., mais il est préférable de le recopier dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`) pour cela on peut :

- soit écrire directement le programme (ou le script), dans un niveau éditeur de programmes,
- soit utiliser le menu `Fich` sous-menu `Charger` de l'éditeur de programmes, si le programme est dans un fichier,
- soit le recopier avec la souris, si le programme est dans la ligne de commande (par exemple après avoir fait `Charger` du menu `Fich` de la session) ou si le programme est dans son éditeur préféré,

En effet, depuis un niveau éditeur de programmes, on peut :

- avoir de l'aide sur les commandes de `Xcas` : il suffit d'écrire la commande et d'appuyer sur la touche F1 de vôtre ordinateur,
- indenter facilement : il suffit d'appuyer sur la touche de tabulation de vôtre ordinateur,
- tester facilement si le programme est syntaxiquement correct : il suffit d'appuyer sur le bouton `OK` de la barre des menus ou sur la touche F9 de vôtre ordinateur : la ligne où se trouve la faute de syntaxe est indiquée en bleu dans la zone intermédiaire.

On corrige les fautes si il y en a...

- Quand le script est syntaxiquement correct, en appuyant sur le bouton `OK` ou la touche F9 le script s'exécute et on obtient le résultat de l'exécution si on n'a pas terminé son écriture par `:;` ou `Done` si on a terminé l'écriture du programme par `:;`
  Exemple On tape dans l'éditeur :
  `S:=0;for (j:=1;j<5;j++) {print(S);S:=S+1/j;}` On obtient dans la zone intermédiaire :
  `S:0`
  `S:1`
  `S:3/2`
  `S:11/6`
  et en réponse : `(0,25/12)`
- Quand le programme est syntaxiquement correct, en appuyant sur le bouton `OK` ou la touche F9 il y a `Success compilling ...` dans la zone intermédiaire et on a le programme en réponse ou `Done` si on a terminé l'écriture du programme par `:;`. On peut alors exécuter le programme dans une ligne de commande.

Vous sauvez le programme (ou le script) avec le bouton `Save` du niveau éditeur de programmes sous le nom que vous voulez lui donner en le terminant par le suffixe `.cxx` (ce nom s'inscrit alors à côté du bouton `Save` du niveau éditeur de programmes). Si ensuite, vous voulez lui donner un autre nom il faut le faire avec le menu `Prog` sous-menu `Sauver comme` de l'éditeur de programmes.

1.4 Débugger un programme avec la syntaxe Xcas

Pour utiliser le débuggeur, il faut que ce programme soit syntaxiquement correct : vous avez par exemple un programme syntaxiquement correct, mais qui ne fait pas ce qu'il devrait faire, il faut donc le corriger.

Avec le débuggeur, on a la possibilité d'exécuter le programme au pas à pas (`sst`), ou d'aller directement (`cont`) à une ligne précise marquée par un point d'arrêt (`break`), de voir (`voir` ou `watch`) les variables que l'on désire surveiller, d'exécuter au pas à pas les instructions d'une fonction utilisateur utilisée dans le programme (`dans` ou `sst_in`), ou de sortir brutalement du débuggeur (`tuer` ou `kill`).
On tape : `debug(nom _du_programme(valeur_des_ arguments))`.
Il faut bien sûr que le programme soit validé :

- si le programme est dans un niveau éditeur de programme, on appuie sur `OK` pour le compiler, on corrige les fautes de syntaxe éventuelles et on appuie sur `OK` jusqu'à obtenir `Success compiling...`
- si le programme qui est syntaxiquement correct se trouve dans un fichier, on tape : `read("toto")` si `toto` est le nom du fichier où se trouve ce programme.

Par exemple, si `pgcd` a été validé, on tape :
`debug(pgcd(15,25))`
L'écran du débugger s'ouvre : il est formé par trois écrans séparés par une ligne `eval` et une barre de boutons `sst,dans,cont...` :

1. dans l'écran du haut, le programme source est écrit et la ligne en surbrillance sera exécutée grâce au bouton `sst`.

2. dans la ligne `eval`, `Xcas` marque automatiquement l'action en cours par exemple `sst`. Cette ligne permet aussi de faire des calculs dans l'environnement du programme ou de modifier une variable, par exemple on peut y écrire `a:=25` pour modifier la valeur de `a` en cours de programme,

3. dans l'écran du milieu, on trouve, le programme, les points d'arrêts, le numéro de la ligne du curseur.

4. une barre de boutons `sst,dans,cont...`
   - `sst` exécute la ligne courante (celle qui est en surbrillance) sans entrer dans les fonctions et met en surbrillance l'instruction suivante,
   - `dans` ou `sst_in` exécute la ligne courante (celle qui est en surbrillance) en entrant dans les fonctions utilisées dans le programme et qui ont été définies précédemment par l'utilisateur, puis met en surbrillance l'instruction suivante du programme en incluant les instructions de la fonction. Cela permet ainsi d'exécuter pas à pas les instructions de cette fonction.
   - `cont` exécute les instructions du programme situées entre la ligne courante et la ligne d'un point d'arrêt et met en surbrillance cette ligne,
   - `tuer` ou `kill` ferme brutalement l'écran du débuggeur.
     Attention il faut fermer l'écran du débuggeur pour pouvoir utiliser `Xcas`.

— `break` ajoute un point d'arrêt. Les points d'arrêts permettent d'aller directement à un point précis avec le bouton `cont`. On marque les points d'arrêts grâce au bouton `break` ou à la commande `breakpoint` d'arguments le nom du programme et le numéro de la ligne où l'on veut un point d'arrêt : par exemple `breakpoint(pgcd,3)`. Pour faciliter son utilisation, il suffit de cliquer dans l'écran du haut sur la ligne où l'on veut le point d'arrêt pour avoir : `breakpoint` dans la ligne `eval`, avec le nom du programme et le bon numéro de ligne, puis de valider la commande. Il suffit donc de cliquer et de valider !
— `rmbrk` enlève un point d'arrêt. On doit, pour réutiliser d'autres points d'arrêts, d'effacer les points d'arrêts utilisés précédemment avec la commande `rmbreakpoint` qui a les mêmes arguments que `breakpoint`. Là encore, pour faciliter son utilisation, il suffit de cliquer sur la ligne où l'on veut enlever le point d'arrêt pour avoir : `rmbreakpoint` dans la ligne de commande, avec le nom du programme et le bon numéro de ligne. **Attention** si il n'y a pas de point d'arrêt à cet endroit `Xcas` en mettra un !
— `voir` ou `watch` ajoute la variable que l'on veut voir évoluer. Si on ne se sert pas de `voir` ou `watch` toutes les variables locales et tous les arguments du programme sont montrées. Si on se sert de `voir` ou `watch` seules les variables designées seront montrées : on appuie sur le bouton `voir` ou `watch` et la commande `watch` s'écrit dans la ligne d'évaluation `eval`. On tape alors, les arguments de `watch` qui sont les noms des variables que l'on veut surveiller, par exemple : `watch(b,r)` et on valide la commande.
— `rmwtch` efface les variables désignées précédemment avec `watch` et que l'on ne veut plus voir, par exemple : `rmwatch(r)`.

5. dans l'écran du bas, on voit soit l'évolution de toutes les variables locales et de tous les arguments du programme, soit l'évolution des variables désignées par `watch`.

1.5 Présentation générale des instructions avec la syntaxe Xcas

1.5.1 Les commentaires

Les commentaires sont des chaînes de caractères, ils sont précédés de `//` ou sont parenthésés par `/* */`

1.5.2 Le bloc

Une `action` ou `bloc` est une séquence d'une ou plusieurs instructions.
Quand il y a plusieurs instructions il faut les parenthéser avec `{ }` et séparer les instructions par un point virgule (`;`)
Un `bloc` est donc parenthésé par `{ }` et commence éventuellement par la déclaration des variables locales (`local...`).

1.5.3 Les variables globales et les variables locales

Voir aussi 2.2 Les variables sont les endroits où l'on peut stocker des valeurs, des nombres, des expressions, des objets.
Le nom des variables est formé par une suite de caractères et commence par une lettre : attention on n'a pas droit aux mots réservés ...ne pas utiliser par exemple la variable `i` dans un `for` si vous avez coché `pas de test de i` dans la `configuration generale` car `i` représente le nombre complexe de module 1 et d'argument $\frac{\pi}{2}$.
L'affectation se fait avec `:=` (par exemple `a:=2; b:=a;`) ou avec `=>` (par exemple `2=>a; a=>b;`).

Les variables locales non symboliques

Une variable utilisée uniquement à l'intérieur d'une fonction (resp d'un bloc) pour contenir des résultats intermédiaires est une variable locale à la fonction (resp au bloc). Les variables locales doivent être déclarées au début de la fonction (resp au début d'un bloc) par le mot réservé `local` puis on met les noms des variables séparés par des virgules (`,`).
Attention
Cette déclaration n'initialise pas ces variables locales à `0`.

Les variables locales peuvent être initialisées lors de leur déclaration à une valeur en mettant les affectations entre parenthèses et separées par des virgules. Mais attention l'initialisation des variables locales faites dans la ligne de `local` se fait en utilisant le contexte global d'évaluation , par exemple :

```
n:=5;
f():={
  local (n:=1),(d:=n+1);
  return d;
}
```

`f()` renvoie `6` et non `2` : c'est la valeur de `n+1` ou `n` est global. Il faut initialiser `d` après la déclaration locale pour utiliser le contexte local en tapant :

```
f():={
  local (n:=1),d;
  d:=n+1;
  return d;
}
```

et alors `f()` renvoie 2.

Les variables locales symboliques

Attention Les variables locales ne sont pas affectées lors de leur déclaration MAIS ce ne sont pas des variables formelles : il faut les obligatoirement les initialiser dans le corps du programme.
Si on veut utiliser dans un programme des variables formelles, on a 2 solutions :

— On considère les variables formelles du programme comme globales, MAIS alors il faut s'assurer que ces variables sont purgées avant l'exécution du programme...ce qui contraignant !
— On déclare les variables formelles du programme avec `local` (par exemple `local x;`), PUIS on utilise `assume(x,symbol);` pour spécifier que la variable `x` devient symmbolique ou on utilise `purge(x)` pour purger la variable `x` qui devient symmbolique. Ainsi au cours du programme, `x` pourra devenir non symbolique si on l'affecte, puis, redevenir symbolique après l'instruction `assume(x,symbol);` ou l'instruction `purge(x)`.

Il est donc préférable de définir la variable formelle `var`, avec `local var;` suivi de `assume(var,symbol);` ou de `purge(var)`.

Exemple

Voici le programme qui donne la valeur de la suite de Fibonnacci u définie par $u_0 = u0, u_1 = u1, u_{n+2} = u_{n+1} + u_n$.

On sait que si a et b sont les racines de $x^2 - x - 1$, les suites vérifiant la relation de récurrence $u_{n+2} = u_{n+1} + u_n$ sont des combinaisons linéaires des suites géométriques a^n et b^n, c'est-à-dire que l'on a :

$$u_n = Aa^n + Bb^n$$

pour A et B solutions du système $[u_0 = A + B,\ u_1 = Aa + Bb]$.

Voici les deux façons de faire :

— les variables formelles sont globales,
Dans le programme qui suit, on utilise les variables formelles `x,A,B` qui doivent être purgées et qui seront des variables globales.
On tape :

```
u(n,uo,u1):={
local L,a,b;
//verifier que A,B,x ne sont pas affect\'ees
[a,b]:=solve(x^2-x-1,x);
L:=linsolve([A+B=uo,A*a+B*b=u1],[A,B]);
return normal(L[0]*a^n+L[1]*b^n);
};
```

Lors de la compliation, `Xcas` dit :

```
//Warning: x A B declared as global variable(s)
compiling u
```

On tape :

```
u(3,0,1)
```

On obtient :

```
2
```

— les variables formelles sont declarées locales,
Dans le programme qui suit, on utilise les variables formelles `A,B,x` qui seront symboliques ou formelles grâce aux commandes :

```
assume(A,symbol);
assume(B,symbol);
assume(x,symbol);
```

ou à la commande `purge(A,B,x)`.

Remarque : pour retrouver des variables non formelles il suffira de les affecter.
On tape :

```
u(n,uo,u1):={
local L,a,b,A,B,x;
assume(A,symbol);
assume(B,symbol);
assume(x,symbol);
[a,b]:=solve(x^2-x-1,x);
L:=linsolve([A+B=uo,A*a+B*b=u1],[A,B]);
return normal(L[0]*a^n+L[1]*b^n);
};
```

Ou on tape :

```
u(n,uo,u1):={
local L,a,b,A,B,x;
purge(A,B,x);
[a,b]:=solve(x^2-x-1,x);
L:=linsolve([A+B=uo,A*a+B*b=u1],[A,B]);
return normal(L[0]*a^n+L[1]*b^n);
};
```

Lors de la compliation, `Xcas` dit :

```
// Success compiling u
```

On tape :

```
u(3,0,1)
```

On obtient :

```
2
```

Pour bien comprendre ce qui se passe, on rajoute des `print` :

```
u(n,uo,u1):={
local L,a,b,A,B,x;
print(A);
assume(A,symbol);
print(A);
assume(B,symbol);
assume(x,symbol);
[a,b]:=solve(x^2-x-1,x);
L:=linsolve([A+B=uo,A*a+B*b=u1],[A,B]);
A:=5;
print(A);
return normal(L[0]*a^n+L[1]*b^n);
};
```

On tape :

```
A:=30
u(3,0,1)
```

On obtient écrit en bleu :
`A:0` (`A` est locale et n'est pas symbolique et vaut `0`)
`A:A` (`A` est locale et symbolique)
`A:5` (`A` est locale etn'est pas symbolique et vaut `5`)
puis la réponse :

`2`
On tape :
`A`
On obtient :
`30` (la variable globale `A` n'est pas symbolique et vaut `30`)

Variables locales internes à un bloc

Voici comme exemple le programme de la fonction qui donne le quotient et le reste de la division euclidienne de 2 entiers (c'est la fonction `iquorem` de `Xcas`) :

```
idiv2(a,b):={
  local (q:=0),(r:=a);
  if (b!=0) {
    q:=iquo(a,b);
    r:=irem(a,b);
  }
  return [q,r];
};
```

Voici le programme de la même fonction mais avec les variables locales internes au bloc du `if` :

```
idiv2(a,b):={
  if (b==0) {return [b,a];}
  if (b!=0) {
    local q,r;
    q:=iquo(a,b);
    r:=irem(a,b);
    return [q,r];
  }
};
```

ou encore avec les variables locales internes au bloc du `else` :

```
idiv2(a,b):={
  if (b==0) {return [b,a];}
  else {
    local q,r;
    q:=iquo(a,b);
    r:=irem(a,b);
    return [q,r];
  }
};
```

1.5.4 Les programmes et les fonctions

Les paramètres sont mis après le nom du programme ou de la fonction entre parenthèses (par exemple `f(a,b):=...`).
Ces paramètres sont initialisés lors de l'appel du programme ou de la fonction et se comportent comme des variables locales. **Remarque** On peut donner aux

paramètres d'une fonction, des valeurs par défaut avec le sigeb =, on écrit par exemple :
`f(a,b=10,c,t=pi/2):=....`
cela signifie dans cet exemple que `f` a 2 ou 3 ou 4 paramètres, on a :
`f(40,20)` est identique à `f(40,10,20,pi/2)` et
`f(40,30,10)` est identique à `f(40,30,10,pi/2)`.

L'affectation se fait avec `:=` (par exemple `a:=2; b:=a;`) ou se fait avec `=>` (par exemple `2=>a; a=>b;`).

Les entrées se font par passage de paramètres ou avec `input`.
Les sorties se font en mettant le nom de la variable à afficher (ou la séquence des variables à afficher ou entre crochets les variables à afficher séparées par une virgule) précedé du mot réservé `print`.
Il n'y a pas de distinction entre programme et fonction : la valeur d'une fonction est précedée du mot réservé `return`.
Remarque `return` n'est pas obligatoire car `Xcas` renvoie toujours la valeur de la dernière instruction, mais `return` est très utile car il fait sortir de la fonction : les instructions situées après `return` ne sont jamais effectuées.

1.5.5 Les tests

Avec le langage `Xcas` les tests ont soit une syntaxe similaire au langage `C++` soit une version française proche du langage algorithmique.
Pour les tests, les syntaxes admises sont :

— `if` *`(condition) instruction`*`;`
on met {..} lorsqu'il faut faire plusieurs instructions :
`if` *`(condition)`* {*`instructions`*}
ou
`si` *`condition`* `alors` *`instructions`* `fsi`
on teste la condition : si elle est vraie, on fait les instructions et si elle est fausse on ne fait rien c'est à dire on passe aux instructions qui suivent le `if` ou le `si`.
Par exemple :

```
testif1(a,b):={
if  (a<b)
    b:=b-a;
return [a,b];
};
```

ou

```
testsi1(a,b):={
si a<b alors
    b:=b-a;
fsi;
return [a,b];
};
```

et on a :

```
testif1(3,13)=testsi1(3,13)=[3,10]
testif1(13,3)=testsi1(13,3)=[13,3]
```

— `if` *`(condition) instruction1`*`; else` *`instruction2`*`;`
on met {..} lorsqu'il faut faire plusieurs instructions :
`if` *`(condition)`* `{`*`instructions1`*`} else {`*`instructions2`*`}`
ou
`si` *`condition`* `alors` *`instructions1`* `sinon` *`instructions2`*
`fsi`
on teste la condition : si elle est vraie, on fait les instructions1 et si elle est fausse on fait les instructions2.
Par exemple :

```
testif(a,b):={
if (a==10 or a<b)
    b:=b-a;
else
    a:=a-b;
return [a,b];
};
```

ou

```
testsi(a,b):={
si a==10 or a<b alors
    b:=b-a;
sinon
    a:=a-b;
fsi;
return [a,b];
};
```

et on a :

```
testif(3,13)=testsi(3,13)=[3,10]
testif(13,3)=testsi(13,3)=[10,3]
testif(10,3)=testsi(10,3)=[10,-7]
```

1.5.6 Les boucles

Avec le langage `Xcas` les boucles ont soit une syntaxe similaire au langage `C++` soit une version française proche du langage algorithmique.
La commande `break` permet de sortir d'une boucle.
Si vous avez fait une boucle infinie, il faut appuyer sur `STOP` pour arrêter votre programme (ou sur `Shift+STOP` si plusieurs sessions travaillent en parallèle).
Pour les boucles, les syntaxes admises sont :

— la boucle `for` ou `pour` qui permet de faire des instructions un nombre de fois qui est connu :
`for (`*`init;condition;increment`*`)` *`instruction`*`;`
on met {..} lorsqu'il faut faire plusieurs instructions :
`for (`*`init;condition;increment`*`) {`*`instructions`*`}`
Le plus souvent (*init ;condition ;increment*) s'écrit en utilisant une variable (par exemple `j` ou `k`...mais pas `i` qui désigne un nombre complexe, si vous avez coché `pas de test de i` dans la `configuration generale`).
Cette variable sera initialisée dans *init*, utilisée dans *condition* et incrementée dans *increment*, on écrit par exemple :

(*j :=1 ;j<=10 ;j++*) (l'increment ou le pas est de 1) ou
(*j :=10 ;j>=1 ;j–*) (l'increment ou le pas est de -1) ou
(*j :=1 ;j<=10 ;j :=j+2*) (l'increment ou le pas est de 2)
ou

```
pour j de 1 jusque 10 faire instructions fpour;
```

ou

```
pour j de 10 jusque 1 pas -1 faire instructions fpour;
```

ou

```
pour j de 1 jusque 10 pas 2 faire instructions fpour;
```

On initialise `j` puis on teste la condition :

— si elle est vraie, on fait les instructions puis on incrémente `j`, puis, on teste la condition : si elle est vraie, on fait les instructions etc...
— si elle est fausse on ne fait rien c'est à dire on passe aux instructions qui suivent le `for` ou le `pour`.

Par exemple :

```
testfor1(a,b):={
local j,s:=0;
for (j:=a;j<=b;j++)
   s:=s+1/j^2;
return s;
};
```

ou

```
testpour1(a,b):={
local j,s:=0;
pour j de a jusque b faire
   s:=s+1/j^2;
fpour;
return s;
};
```

Si `a>b`, l'instruction ou le bloc d'instructions du `for` ou du `pour` ne se fera pas et la fonction retournera `0`,
Si `a<=b` la variable `j` va prendre successivement les valeurs `a, a+1,...b` (on dit que le pas est de 1) et pour chacune de ces valeurs l'instruction ou le bloc d'instructions qui suit sera exécuté. Par exemple `testfor1(1,2)` renverra `1+1/4=5/4`.

```
testfor2(a,b):={
local j,s:=0;
for (j:=b;j>=a;j--)
    s:=s+1/j^2;
return s;
};
```

ou

```
testpour2(a,b):={
local j,s:=0;
pour j de b jusque a pas -1 faire
   s:=s+1/j^2;
fpour;
return s;
```

```
};
```
Dans ce cas, si `a<=b` la variable `j` va prendre successivement les valeurs `b, b-1,...a` (on dit que le pas est de -1) et pour chacune de ces valeurs l'instruction ou le bloc d'instructions qui suit sera exécuté. Par exemple `testfor2(1,2)` renverra `1/4+1=5/4`.
```
testfor3(a,b):={
local j,s:=0;
for (j:=a;j<=b;j:=j+3)
    s:=s+1/j^2;
return s;
};
```
ou
```
testpour3(a,b):={
local j,s:=0;
pour j de a jusque b pas 3 faire
   s:=s+1/j^2;
fpour;
return s;
};
```
Dans ce cas, si `a<=b` la variable `j` va prendre successivement les valeurs `a, a+3,...a+3k` avec `k` le quotient de `b-a` par `3` (`3k<=b-a<3(k+1)`) (on dit que le pas est de 3) et pour chacune de ces valeurs l'instruction ou le bloc d'instructions qui suit sera exécuté. Par exemple `testfor3(1,5)` renverra `1+1/16=17/16`.

Attention

Le pas doit être numérique et non une expression car il faut que le compilateur puisse transformer le `pour` en `for` avec le bon test.

Par exemple :
```
testpour4(a,b,p):={
local j,s:=0;
pour j de a jusque b pas p faire
   afficher(j);
   s:=s+j;
fpour;
return s;
};
```
Selon le signe de `p` le `pour` peut se traduire :

— si `p>0` en
```
   testpour41(a,b,p):={
   local j,s:=0;
   for (j:=a;j<=b;j:=j+p){
      afficher(j);
      s:=s+j;
   };
   return s;
   };
```
— si `p<0` en
```
   testpour42(a,b,p):={
```

```
local j,s:=0;
for (j:=a;j>=b;j:=j+p){
   afficher(j);
   s:=s+j;
};
return s;
};
```

— lorsque le compilateur ne connait pas le signe du pas, il suppose alors le pas positif et il le remplace par sa valeur absolue.

```
testpour43(a,b,p):={
local j,s:=0;
for (j:=a;j<=b;j:=j+abs(p)){
   afficher(j);
   s:=s+j;
};
return s;
};
```

— la boucle `while` ou `tantque` permet de faire plusieurs fois des instructions avec une condition d'arrêt au début de la boucle : `while (`*condition*`)` *instruction*`;`
on met {..} lorsqu'il faut faire plusieurs instructions :
`while (`*condition*`) {`*instructions*`}`
ou
`tantque` *condition* `faire` *instructions* `ftantque;`
On teste la condition :
— si elle est vraie, on fait les instructions puis, on teste la condition : si elle est vraie, on fait les instructions etc...
— si elle est fausse on ne fait rien c'est à dire on passe aux instructions qui suivent le `while` ou le `tantque`.
Par exemple :

```
testwhile(a,b):={
while (a==10 or a<b)
   b:=b-a;
return [a,b];
};
```

ou

```
testtantque(a,b):={
tantque a==10 or a<b faire
   b:=b-a;
ftantque;
return [a,b];
};
```

Un exemple : le PGCD d'entiers
- Version itérative

```
pgcd(a,b):={
 local r;
 while (b!=0) {
    r:=irem(a,b);
```

```
    a:=b;
    b:=r;
 }
 return a;
};
```

ou

```
pgcd(a,b):={
 local r;
 tantque b!=0 faire
    r:=irem(a,b);
    a:=b;
    b:=r;
 ftantque;
 return a;
};
```

-Version récursive

```
pgcdr(a,b):={
 if (b==0) return a;
 return pgcdr(b,irem(a,b));
};
```

ou

```
pgcdr(a,b):=if (b==0)
               return a;
             else
               return pgcdr(b,irem(a,b));
```

— La boucle `repeat` ou `repeter` permet de faire plusieurs fois des instructions avec une condition d'arrêt à la fin de la boucle :

`repeat` *`instructions`* `until` *`condition`*`;`

ou

`repeter` *`instructions`* `jusqua` *`condition`*`;`

ou

`repeter` *`instructions`* `jusqu_a` *`condition`*`;`

On fait les instructions, puis on teste la condition :

— si elle est vraie, on fait les instructions puis, on teste la condition etc...

— si elle est fausse on passe aux instructions qui suivent le `repeat` ou le `repeter`.

Par exemple :

```
f():={
  local a;
  repeat
     saisir("entrez un reel entre 1 et 10",a);
   until a>=1 && a<=10;
  return a;
}:;
```

ou

```
f():={
  local a;
  repeter
```

```
    saisir("entrez un reel entre 1 et 10",a);
  jusqua a>=1 et a<=10;
  return a;
};
```

Voici la fonction qui renvoie un point fixe de `f` en utilisant `repeat`. Cette fonction donne la solution à `eps` près de `f(x)=x` en utilisant au plus `n` itérations et `x0` comme début de l'itération donné par le théorème du point fixe.

```
point_fixe(f,n,x0,eps):={
 local j,x1,x00;
 j:=0;
 repeat
   x1:=evalf(f(x0));
   j:=j+1;
   x00:=x0;
   x0:=x1;
 until (abs(x00-x0)<eps) or (j>=n);
 return j,x0;
};
```

ou

```
point_fixe(f,n,x0,eps):={
 local j,x1,x00;
 j:=0;
 repeter
   x1:=evalf(f(x0));
   j:=j+1;
   x00:=x0;
   x0:=x1;
 jusqua (abs(x00-x0)<eps) or (j>=n);
 return j,x0;
};
```

— La boucle `do` *instructions* `od` ou `faire` *instructions* `ffaire` est une boucle infinie il faut donc mettre une condition d'arrêt avec un `break` ou un `return` dans le corps de la boucle.
Par exemple la fonction précédente en utilisant `do` :

```
point_fixe(f,n,x0,eps):={
 local j,x1;
 j:=0;
 do
   x1:=evalf(f(x0));
   if (abs(x1-x0)<eps) return x1;
   x0:=x1;
   j:=j+1;
   if (j>=n) break;
 od;
 return "non trouve au bout de ", n," iterations";
};
```

ou en utilisant `faire`

```
point_fixe(f,n,x0,eps):={
 local j,x1;
 j:=0;
 faire
   x1:=evalf(f(x0));
   if (abs(x1-x0)<eps) return x1;
   x0:=x1;
   j:=j+1;
   if (j>=n) break;
 ffaire;
 return "non trouve au bout de ", n," iterations";
}:;
```

On peut aussi écrire cela avec un `for`

```
point_fixe(f,n,x0,eps):={
 local j,x1;
 for (j:=0;j<n;j++){
   x1:=evalf(f(x0));
   if (abs(x1-x0)<eps) return x1;
   x0:=x1;
 }
 return "non trouve au bout de ", n," iteration";
}:;
point_fixe(x->cos(x),100,1.0,1e-12)=0.739085133215
```

On vérifie : `cos(0.739085133215)=0.739085133215`

Remarque

Le `for` permet de faire tous les types de boucle à condition de mettre, soit `for (;`*test_arrêt*`;)...` soit et `for (;;)...` et une instruction `break` dans le corps du `for`, par exemple :

```
pgcd(a,b):={
local r;
for (;b!=0;) {
r:=irem(a,b);
a:=b;
b:=r;
}
return a;
}
:;
```

ou

```
pgcd(a,b):={
local r;
for (;;) {
if (b==0) break;
r:=irem(a,b);
a:=b;
b:=r;
}
return a;
```

```
}
:;
```

Chapitre 2

Les différentes instructions selon le mode choisi

Xcas propose un mode de compatibilité avec Maple, MuPAD et la TI89/92 : pour cela, il suffit de le spécifier dans Prog style du menu de configuration du cas (bouton Config ou menu Cfg->Configuration du CAS) ou avec le menu Cfg->Mode (syntax). On peut choisir, en cliquant sur la flèche située à coté de Prog style : Xcas ou Maple ou MuPAD ou TI89/92.
On a aussi la possibilité d'importer une session Maple ou une archive TI89/92 en choisissant Importer du menu Fich, ou importer dans un niveau éditeur de programmes un fichier écrit en syntaxe Maple, Mupad ou TI89/92 par le menu Prog->Inserer.

On présente ici le mode Xcas qui est proche de la syntaxe C. On a aussi la possibilité d'avoir toutes les instructions en français de façon à être proche du langage Algorithmique.

2.1 Les commentaires

2.1.1 Traduction Algorithmique

Il faut prendre l'habitude de commenter les programmes. En algorithmique un commentaire commence par // et se termine par un passage à la ligne.
Exemple :
//ceci est un commentaire

2.1.2 Traduction Xcas

Avec Xcas un commentaire commence par // et se termine par un passage à la ligne.
Exemple :
//ceci est un commentaire

2.1.3 Traduction MapleV

Un commentaire commence par # et se termine par un passage à la ligne.
Exemple :

ceci est un commentaire

2.1.4 Traduction MuPAD

Un commentaire est entouré de deux # ou commence par /* et se termine par */.
Exemple :
ceci est un commentaire
/* ceci est un commentaire */

2.1.5 Traduction TI89 92

Le commentaire commence par © (F2 9) et se termine par un passage à la ligne.
Exemple :
© ceci est un commentaire

2.2 Les variables

Voir aussi 1.5.3

2.2.1 Leurs noms

Ce sont les endroits où l'on peut stocker des valeurs, des nombres, des expressions.
Avec Xcas les noms des variables ou des fonctions commencent par une lettre et sont formés par des lettres ou des chiffres.
Par exemple :
azertyuiop:=2 met la valeur 2 dans la variable azertyuiop
azertyuio?:=1 renvoie un message d'erreur car azertyuio? contient ? qui n'est pas une lettre.
Étant donné que Xcas fait du calcul formel il faut quelquefois purger une variable var pour qu'elle redevienne formelle avec la commande purge(var) ou encore utiliser la commande assume(var,symbol) pour que var redevienne formelle.
On tape :
x:=2; x contient 2 et n'est pas formelle.
On tape :
purge(x) : cela renvoie la valeur x si x est affecté et x redevient une variable formelle, et, si x n'est pas affecté purge(x) renvoie "x not assigned".
ou bien, on tape :
assume(x,symbol)
et cela renvoie dans tous les cas DOM_SYMBOLIC.
Avec MapleV et MuPAD, un nom doit commencer par une lettre, ne pas contenir d'espace, de symbole opératoire (+, -, ...) et ne pas être un mot réservé comme D, I, ... pour MapleV.
On peut utiliser des noms ayant plus de 8 caractères.

Pour TI89/92 un nom doit commencer par une lettre, ne pas contenir d'espace, de symbole opératoire (+, -, ...), ne pas être un mot réservé et ne doit pas avoir plus de 8 caractères.

2.2.2 Notion de variables locales

Pour Xcas il faut définir les variables locales en début de programme en écrivant :

```
local a,b,c;
```

Attention

Dans un programme toutes les variables sont des variables qui sont initialisées par défaut à 0.

Mais ces variables peuvent être initialisées au moment de leur déclaration avec local, par exemple : `local a,(b:=1),c;`

Remarque

Xcas accepte qu'il y ait plusieurs local dans un programme, par exemple :

```
essailoc(n):={
local s:=0;
local j;
for (j:=1;j<=n;j++){
s:=s+1/j;
}
return s;
}
```

Pour MapleV, il faut définir les variables locales en début de programme en écrivant :

```
local a,b
```

Pour MuPAD, il faut définir les variables locales en début de programme en écrivant :

```
local a,b
```

Pour la TI89/92 il faut définir les variables locales en début de programme en écrivant :

```
:local a,b
```

2.3 Les paramètres

Quand on écrit une fonction il est possible d'utiliser des paramètres.
Par exemple si A et B sont les paramètres de la fonction PGCD on écrit :

```
PGCD(A,B)
```

Ces paramètres se comportent comme des variables locales, la seule différence est qu'ils sont initialisés lors de l'appel de la fonction. L'exécution se fait en demandant par exemple : `PGCD(15,75)`

2.3.1 Traduction Xcas

Avec Xcas on peut définir une fonction :

— en lui donnant un nom :
on met alors le nom de la fonction puis, entre parenthèses et séparés par une virgule, le nom des paramètres, par exemple :
`addition(a,b):=a+b;`
L'exécution se fera alors en tapant : `addition(2,5).`

— en mettant le nom des paramètres entre parenthèses puis `->` puis entre des accolades le corps de la fonction, par exemple :
`(a,b)->a+b;.`
cette façon d'écrire une fonction peut être utile quand on doit mettre une fonction comme argument d'une commande par exemple :
`makelist((j)->j^2+1,1..3)` ou
`makelist((j)->j^2+1,1..9,2).`
Remarque
Le langage de `Xcas` est fonctionnel puisque on peut passer des programmes ou des fonctions en paramètre.

Autre exemple :

```
pgcd(a,b):={
  local r;
  while (b!=0){
  r:=irem(a,b);
  a:=b;
  b:=r;
  }
  return(a);
};
```

ou encore :

```
pgcd:=(a,b)-> {
  local r;
  while (b!=0){
  r:=irem(a,b);
  a:=b;
  b:=r;
  }
  return(a);
};
```

L'exécution se fera alors en tapant : `pgcd(15,75).`

2.3.2 Traduction MapleV

```
PGCD:=proc(A,B)
....
....
end:
```

Attention
Ces paramètres NE se comportent PAS comme des variables locales : ils sont initialisés lors de l'appel de la fonction mais ne peuvent pas être changés au cours de la procédure. L'exécution se fait en demandant par exemple : `PGCD(15,75)`

2.3.3 Traduction MuPAD

```
PGCD:=proc(A,B)
begin
....
end_proc:
```

Ces paramètres se comportent comme des variables locales, la seule différence est qu'ils sont initialisés lors de l'appel de la fonction. L'exécution se fait en demandant par exemple : `PGCD(15,75)`

2.3.4 Traduction TI89/92

Pour les `TI 89/92` on met le nom des paramètres dans le nom de la fonction par exemple :

```
:addition(a,b)
:pgcd(a,b)
```

2.4 Les Entrées

2.4.1 Traduction Algorithmique

Pour que l'utilisateur puisse entrer une valeur dans la variable `A` au cours de l'exécution d'un programme, on écrira, en algorithmique :

```
saisir A
```

Et pour entrer des valeurs dans `A` et `B` on écrira :

```
saisir A,B
```

2.4.2 Traduction Xcas

On écrit :

```
input(A);
```

ou

```
saisir(A);
```

2.4.3 Traduction MapleV

On peut utiliser :

```
A := readline(); ou
A:=readstat('A=?');
```

2.4.4 Traduction MuPAD

```
input("A=",A)
input("A=",A,"B=",B )
```

2.4.5 Traduction TI89/92

```
:Prompt A
:Prompt A,B
```
ou encore :
```
:Input "A=",A
```

2.5 Les Sorties

2.5.1 Traduction Algorithmique

En algorithmique on écrit :
`afficher A`
ou si on veut connaitre le nom de la variable qui est affichée
`afficher "A=",A`

2.5.2 Traduction Xcas

On écrit :
```
A:=2;B:=3;
print(A);
```
ou
`afficher(A);`
On obtient, en bleu dans la zone intermédaire `2,3`
`print(A,B);`
ou
`afficher(A,B);`
On obtient, en bleu dans la zone intermédaire `2,3`

ou encore
`print("A=",A);`
ou
`afficher("A=",A);` On obtient, en bleu dans la zone intermédaire `"A=",2`
ou encore
`print("A="+A);`
ou
`afficher("A="+A);` On obtient, en bleu dans la zone intermédaire `A=2`
`print("A=",A,"B=",B);`
ou
`afficher("A=",A,"B=",B);` On obtient, en bleu dans la zone intermédaire `"A=",2,"B=",3`
`print` ou `afficher` permet de réaliser des affichages en cours de programmme. Ces affichages s'écriront alors en bleu dans la zone intermédaire .

On tape par exemple dans un niveau éditeur de programmes (que l'on ouvre avec Alt+p) :

```
carres(n):={
local j;
for (j:=1;j<n+1;j++) {
print(j^2);
}
return n^2;
}
```

Puis on complile ce programme en cliquant sur OK.
On tape ensuite dans une ligne de commandes : carres(5)
On obtient :
1
4
9
16
25
écrit en bleu dans la zone intermédaire
et
25
le résultat écrit en noir dans la zone des réponses.

2.5.3 Traduction MapleV

print(`A=`.A):
On préférera ` (accent grave) à " (guillemet), comme délimiteur de chaîne, car le premier n'est pas affiché, alors que le deuxième l'est.

2.5.4 Traduction MuPAD

print("A=",A)
Il fait savoir que lorsqu'une procèdure est appelée, la suite d'instructions entre begin et end_proc est exécutée, et le résultat est égal au résultat de la dernière évaluation.

2.5.5 Traduction TI89/92

:Disp "A=",A
:ClrIO efface l'écran.
:ClrHome efface l'écran pour la TI 83+.
:Pause arrête le programme (on appuie sur ENTER pour reprendre l'exécution).

2.6 La séquence d'instructions ou action ou bloc

Une action ou bloc est une séquence d'une ou plusieurs instructions.
En langage algorithmique, on utilisera l'espace ou le passage à la ligne pour terminer une instruction.

2.6.1 Traduction Xcas

Avec `Xcas` la séquence d'instructions est parenthèsées par `{ }`. La séquence d'instructions est appelée un `bloc` ou une `action` et `;` termine chaque instruction. Seule la dernière instruction d'un programme génére la réponse. Si on veut des sorties intermédiaires il faudra le faire à l'aide de la commade `print` ou `afficher` et ces sorties se feront alors avant la réponse, dans la zone intermédaire en écriture bleue.
Par exemple si on écrit dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`):

```
pgcd(a,b):={
  local r;
  while (b!=0){
  r:=irem(a,b);
  print(r);
  a:=b;
  b:=r;
  }
  return(a);
};
```

On compile en appuyant sur `OK` et on tape :
`pgcd(15,25)`
On obtient :
`5` comme réponse et :
`r:15`
`r:10`
`r:5`
`r:0`
s'écrivent en bleu dans un écran appelé zone intermédaire qui se met avant la réponse.
Si la dernière instruction est géométrique, elle génère une sortie dans unécran géométrique. Les instructions `print` ou `afficher` se feront alors en bleu dans la zone intermédaire située avant l'écran géométrique où se trouve la réponse. Mais les instructions géométriques intermédiaires se feront dans l'écran `DispG` qui est un écran géométrique que l'on obtient avec la commande `DispG()` ou avec le menu `Cfg->Montrer->DispG`.
Par exemple si on écrit dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`):

```
pgcdg(a,b):={
  local r;
  while (b!=0){
  r:=irem(a,b);
  print(r);
  point(a+i*b);
  segment(a+i*b,b+i*r);
  a:=b;
  b:=r;
```

```
  }
  return(point(a+i*b));
};
```

On compile en appuyant sur `OK` et on tape :
`A:=pgcdg(15,25)`
On obtient :
Un écran de graphique s'ouvre avec le point `A` de coordonnées (0 ;5) et :
`r:15`
`r:10`
`r:5`
`r:0`
s'écrivent en bleu dans un écran appelé zone intermédaire qui se met avant l'écran de géométrie.
On ouvre l'écran géométrique `DispG` (avec la commande `DispG()` ou avec le menu `Cfg->Montrer->DispG`) Sur l'écran `DispG` il y a les différents segments et les points (sans la lettre `A`) Pour tout voir, vous devez soit appuyer sur le bouton `auto` de l'écran `DispG`, soit changer la configuration avec le bouton `cfg` de l'écran `DispG`.

2.6.2 Traduction MapleV

`:` ou `;` indique la fin d'une instruction.
Une instruction terminée par point-virgule (`;`) génére une sortie et celle terminée par deux points (`:`) n'en génére pas.

2.6.3 Traduction MuPAD

Le `:` ou `;` est un séparateur d' instructions.
Le `;` génére une sortie alors que le `:` n'en génére pas.

2.6.4 Traduction TI89/92

`:` indique la fin d'une instruction. Il faut noter qu'à chaque passage à la ligne le `:` est mis automatiquement.

2.7 L'instruction d'affectation

L'affectation est utilisée pour stocker une valeur ou une expression dans une variable.

2.7.1 Traduction Algorithmique

En algorithmique on écrira par exemple :
`3=>A`
`2*A=>B`
pour stocker `3` dans `A`, `2*A` (c'est à dire 6) dans `B`

2.7.2 Traduction Xcas

Avec Xcas on écrira :
a:=3;
b:=2*a;
ou encore
3=>a;
2*a=>b;
pour stocker 3 dans a, 2*a (c'est à dire 6) dans b

En tapant le nom d'une variable dans Xcas on peut voir le contenu de cette variable. Si on tape le nom d'une fonction, on verra la définition de la fonction.
Attention
On peut écrire :
(a,b):=(1,2) ou (a,b):=[1,2] ou [a,b]:=[1,2] ou [a,b]:=(1,2)
qui est équivalent à a:=1;b:=2
c'est à dire ici à a:=1;b:=2 mais
a:=1;b:=2; (a,b):=(a+b,a-b) est équivalent à c:=a;a:=a+b;b:=c-b
donc ici à a:=3;b:=-1
Donc si on tape :
(a,b):=(1,2);(a,b):=(a+b,a-b)
On obtient :
3 dans a et -1 dans b
Donc si on tape :
a:=1;(a,b):=(2,a) On obtient :
2 dans a et 1 dans b
mais purge(a);(a,b):=(2,a)
On obtient :
2 dans a et 2 dans b
Donc méfiance ....et utiliser (a,b):=(b,a) qu'avec prudence !

2.7.3 Traduction Maple

Avec Maple on écrit :
b:=2*a pour stocker 2*a dans b.

2.7.4 Traduction MuPAD

Avec MuPAD on écrit :
b:=2*a pour stocker 2*a dans b.

2.7.5 Traduction TI89/92

On écrit 2*A=>B pour stocker 2*A dans B.

2.8 L'instruction d'affectation par référence

L'affectation := est utilisée pour stocker une valeur ou une expression dans une variable. Mais lorsque Xcas réalise cette affectation il effectue une recopie de la valeur dans la variable ce qui peut être long lorsqu'il s'agit d'une liste de

grande taille. C'est pourquoi quand la variable contient une liste, avec `Xcas`, il faut utiliser l'opérateur infixé =< qui stocke par référence le deuxième argument dans la variable donnée en premier argument.
On tape par exemple :
`L=<makelist(x->x^2,0,10000);` puis, par exemple :
`for(j:=0;j<=1000;j++) L[10*j]=<0`

2.9 L'instruction pour faire des hypothèses sur une variable formelle

L'instruction `assume` ou `supposons` permet de faire une hypothèse sur une variable formelle par exemple d'attribuer une valeur à une variable pour faire une figure, tout en laissant cette variable reste formelle lorsqu'on l'utilise pour faire des calculs ou encore de supposer que `n` est une v ariable entère ce qui permet de simplifier certaines expressions comme `cos(n*pi)`....

2.9.1 Traduction Algorithmique

En algorithmique on écrira par exemple :
```
supposons(n,entier)
supposons(a>2)
supposons(a=2)
```

2.9.2 Traduction Xcas

Avec `Xcas` on écrira :
`assume(n,integer);` ou `supposons(n,integer);` puis par exemple `cos(n*pi)` et on obtient `(-1)^n`
`assume(a>2);`
On peut faire plusieurs hypothèses sur une variable avec `assume` et `additionally` par exemple, pour dire que `n` est un entier plus grand que 3, on tape :
`assume(n,integer);additionally(n>3);` Il faut noter que si on écrit dans l'écran de géométrie :
`assume(a=2);` ou `supposons(a=2);`
ou encore
`assume(a:=2);` ou `supposons(a:=2);`
cela a pour effet de mettre en haut et à droite un curseur noté `a`. En effet, cela veut dire que l'on va faire une figure de géométrie en donnant à `a` la valeur `2`, mais que les différents calculs se feront avec `a` formelle. Bien sur la valeur donnée à `a` pourra être modifiée à l'aide du curseur et ainsi modifier la figure selon les valeurs de `a`.

2.9.3 Traduction Maple

Avec `Maple` on écrit :
```
assume(n,integer);
assume(a>2);
```

2.9.4 Traduction MuPAD

Avec `MuPAD` on écrit :
```
assume(n,integer);
assume(a>2);
```

2.9.5 Traduction TI89/92

Il n'y a pas d'hypothèse en mode `TI`.

2.10 L'instruction pour connaitre les contraintes d'une variable

2.10.1 Traduction Algorithmique

En algorithmique on écrira par exemple :
```
domaine(A)
```

2.10.2 Traduction Xcas

Avec `Xcas` on écrira :
`about(a)` ou `domaine(a)`
`assume(a)` ou `supposons(a)`

2.10.3 Traduction Maple

Avec `Maple` on écrit :
`about(a)` ou `domaine(a)`
`assume(a)`

2.10.4 Traduction MuPAD

Avec `MuPAD` on écrit :
```
getprop(a)
```

2.10.5 Traduction TI89/92

Il n'y a pas d'hypothèse en mode `TI`.

2.11 Les instructions conditionnelles

2.11.1 Traduction Algorithmique

si *condition* alors *action* fsi
si *condition* alors *action1* sinon *action2* fsi
Exemple :
```
si A = 10 ou A < B alors B-A=>B sinon A-B=>A fsi
```

2.11.2 Traduction Xcas

```
   if (condition) {action;}
```
ou encore
```
if (condition) then action; end
```
ou encore
```
si (condition) alors action; fsi
```
et
```
if (condition) {
action1;
} else {
action2;
}
```

ou encore
```
if (condition) then
action1;
else action2;
end
```
ou encore
```
si (condition) alors
action1;
sinon action2;
fsi
```

Exemples

```
if ((a==10) or (a<b)) {b:=b-a;} else {a:=a-b;}
```

```
essaiif(a,b):={
  if ((a==10) or (a<b)) {
    b:=b-a;
  } else {
    a:=a-b;
  }
  return([a,b]);
};
```

```
essaisi(a,b):={
  si ((a==10) or (a<b))
   alors
    b:=b-a;
   sinon
    a:=a-b;
  fsi;
  return([a,b]);
};
```

```
idivsi(a,b):={
  local (q:=0),(r:=a);
```

```
  if (b!=0){
    q:=iquo(a,b);
    r:=irem(a,b);
  }
  return([q,r]);
};

idivsi(a,b):={
  local (q:=0),(r:=a);
  si (b!=0)
   alors
    q:=iquo(a,b);
    r:=irem(a,b);
  fsi
  return([q,r]);
};
```

Avec Xcas, lorsqu'il y a plusieurs `if...else if...` à la suite on peut aussi utiliser un `elif` qui est une écriture condensée de `else if` et se traduit par :

```
if (condition1) then
action1;
elif (condition2) then
action2;
elif (condition3) then
action3;
end
```

ou bien

```
if (condition) then
action1;
elif (condition) then
action2;
elif (condition) then
action3;
else action4;
end
```

Exemples

```
s(a):={
local r;
if a>=10 then
r:=5;
elif  a>=8 then
r:=4;
elif  a>=6 then
r:=3;
elif  a>=4 then
r:=2
else
r:=1;
end;
```

```
return r;
}
```

Avec Xcas, on peut aussi utiliser un "case" lorsque les différentes instructions à effectuer correspondantes à la valeur d'une expression qui doit être une valeur entière et qui se traduit par :

```
switch (<nom_de_variable>){
case val_de_la_variable : {
.....
break;
}
case val_de_la_variable : {
.....
break;
} default : {
...
}
}
```

Exemple :

```
s(a):={
local r;
switch(a) {
case 1 :{
r:=1;
break;
}
case 2 :{
r:=-1;
break;
}
default :{
r:=0;
}
}
return r;
}
```

2.11.3 Traduction MapleV

si ... alors
`if <condition> then <action> fi;`
si ... alors ... sinon ...
`if <condition> then <action1> else <action2> fi;`
Le schéma général à n cas :
`if <condition_1> then <action_1>`
`elif <condition_2> then <action_2>`

```
... elif <condition_n-1> then <action_n-1> else <action_n>
fi;
```

2.11.4 Traduction MuPAD

```
if <condition> then <action> end_if
if <condition> then <action1> else <action2> end_if
```
Exemple :
```
if a = 10 or A < B then b:=b-a else a:=a-b end_if
```
Lorsque il y a plusiuers `if else` à la suite on écrit `elif` au lieu de `else if`.

2.11.5 Traduction TI89/92

Si ..alors
```
:If condition Then : action : EndIf
```
Si..alors sinon
```
:If condition Then : action1 : Else : action2: EndIf
```
Exemple :
```
:If A = 10 or A < B Then : B-A=>B : Else : A-B=>A : EndIf
```

2.12 Les instructions "Pour"

On utilise l'instruction "pour" lorsqu'on connait le nombre de fois que l'on doit effectuer les instructions ("pour" peut aussi servir à faire des boucles complètement génériques).

2.12.1 Traduction Algorithmique

```
pour I de A jusque B faire action fpour
pour I de A jusque B (pas P) faire action fpour
```

2.12.2 Traduction Xcas

Attention si vous avez coché `pas de test de i` dans la `configuration generale`, on ne doit pas employer la variable `i` car `i` represente le nombre complexe de module 1 et d'argument $\pi/2$.
```
for (j:=1;j<=b;j:=j+1) {action;} ou encore
for (j:=1;j<=b;j:=j++) {action;}
```
ou encore
```
pour j de 1 jusque b faire action; fpour
```
et
```
for (j:=1;j<=b;j:=j+p) {action;}
```
ou encore
```
pour j de 1 jusque b pas p faire action; fpour
```
Exemples

```
essaifor(a,b):={
  local s:=0;
  for (j:=a;j<b+1;j++){
```

```
    s:=s+1/j^2;
  }
  return(s);
};

essaiford(a,b):={
  local s:=0;
  for (j:=b;j>a-1;j--){
    s:=s+1/j^2;
  }
  return(s);
};
```

2.12.3 Traduction MapleV

Voici la syntaxe exacte :
`for <nom> from <expr> by <expr> to <expr>`
`do <action> od;`
Par exemple
`for i from 1 to n do <action:> od`
`for i from 1 by p to n do <action:> od`

2.12.4 Traduction MuPAD

`for i from a to b do <action> end_for`
`for i from b downto a do <action> end_for`
`for i from a to b step p do <action> end_for`
Vous pouvez aussi ouvrir le menu `MuPAD` sous menu `Shapes` et séléctionner `for`.

2.12.5 Traduction TI89 92

`:For I,A,B : action : EndFor`
`:For I,A,B,P : action : EndFor`

2.13 L'instruction "Tant que"

On effectue les instructions tant que la condition est vraie :
on teste la condition si elle est fausse on s'arrête (arret= (condition =fausse)) sinon on effectue les instructions etc ... On arrête la boucle quand la condition devient fausse c'est à dire :
tantque (non arret) on effectue les instructions.

2.13.1 Traduction Algorithmique

`tantque condition faire`
`action`
`ftantque`

2.13.2 Traduction Xcas

```
while (condition) {
action;
}
```
ou encore
```
tantque (condition) faire
action;
ftantque
```

Exemple

```
essaiwhile(a,b):={
  while ((a==10) or (a<b)) {
    b:=b-a;
  }
  return([a,b]);
};
```

2.13.3 Traduction MapleV

```
while <condition> do <action> od:
```

2.13.4 Traduction MuPAD

```
while <condition> do <action> end_while
```
Vous pouvez aussi ouvrir le menu `MuPAD` sous menu `Shapes` et séléctionner `while`.

2.13.5 Traduction TI89/92

```
:While condition :action :EndWhile
```

2.14 L'instruction "repeter"

On répéte les instructions jusqu'à ce que la condition devienne vraie. On fait les instructions (les instructions se font donc au moins une fois) puis on teste la condition si elle est vraie on s'arrête (arret= (condition=vraie)) sinon on effectue les instructions etc ...On arrête la boucle quand la condition devient vraie.

2.14.1 Traduction Algorithmique

```
repeter
action
jusqua  condition
```

2.14.2 Traduction Xcas

```
repeat
action
until  (condition)
```
ou encore
```
repeter
action
jusqua  (condition)
```
L'instruction "repeter" est traduite en :
```
while (true){
action;
if (condition) break;
}
```
cette traduction reste invisible à l'utilisateur (tant qu'il n'exécute pas son programme au débugueur).

2.14.3 Traduction MapleV

Pas de schéma "repeter" en `MapleV`, on utilise une boucle infinie et un `break`
```
do action  if condition then break; fi; od;
```

2.14.4 Traduction MuPAD

```
repeat
action
until  condition
end_repeat
```

2.14.5 Traduction TI89/92

Pas de schéma "repeter" chez `TI`, on utilise une boucle infinie et un `Exit`
```
:Loop :action :If (condition) :Exit :Endloop
```

2.15 Les conditions ou expressions booléennes

Une condition est une fonction qui a comme valeur un booléen, à savoir elle est soit `vraie` soit `fausse`.

2.15.1 Les opérateurs relationnels

Traduction Algorithmique

Pour exprimer une condition simple on utilise en algorithmique les opérateurs :
$= > < \leq \geq \neq$

Traduction Xcas

Ces opérateurs se traduisent pour `Xcas` par :

`== > < <= >= !=`

Attention pour `Xcas` l'égalité se traduit comme en langage `C` par : `==`

Traduction MapleV, MuPAD, TI89/92

Ces opérateurs se traduisent pour `MapleV`, `MuPAD` et `TI89/92` par :

`= > < <= >= <>`

2.15.2 Les opérateurs logiques

Pour traduire des conditions complexes, on utilise en algorithmique, les opérateurs logiques :
`ou et non`
Pour `Xcas`, ces opérateurs se traduisent par :
`or and not` ou encore par || &&
Pour `MapleV`, `MuPAD` et `TI89/92`, ces opérateurs se traduisent par :
`or and not`

2.16 Les fonctions

Dans une fonction on ne fait pas de saisie de données : on utilise des paramètres qui seront initialisés lors de l'appel.
Les entrées se font donc par passage de paramètres.
Dans une fonction on veut pouvoir réutiliser le résultat :
en algorithmique, c'est la commande `retourne` qui renvoie la valeur de la fonction.
Si la fonction est utilisée seule, sa valeur sera affichée et si la fonction est utilisée dans une expression sa valeur sera utilisée pour calculer cette expression.

2.16.1 Traduction Algorithmique

On écrit par exemple en algorithmique :

```
fonction addition(A,B)
retourne A+B
ffonction
```

Cela signifie que :
- Si on fait exécuter la fonction, ce qui se trouve juste après `retourne` sera la valeur de la fonction, mais les instructions qui suivent `retourne` seront ignorées (`retourne` fait sortir immédiatement de la fonction).
- On peut utiliser la fonction dans une expression ou directement dans la ligne de commande et, dans ce cas, sa valeur sera affichée.

2.16.2 Traduction Xcas

`Xcas` utilise `return` ou `retourne`.

```
addition(a,b):={
  return(a+b);
}
```

Remarques :

— `return` n'est pas obligatoire on peut aussi écrire :
`addition(a,b):=a+b;`

— `return` fait sortir immédiatement de la fonction. Pour définir le minimum de 2 nombres on écrit soit avec 1 instruction

```
mini1(a,b):={
si(a>b) alors
 return b
sinon
 return a;
fsi
}
```

soit avec 2 instructions

```
mini2(a,b):={
si(a>b) alors
 return b;
fsi
return a;
}
```

2.16.3 Traduction MapleV

`retourne` se traduit par `RETURN`

```
addition:= proc(a,b)
RETURN(a+b);
end:
```

Remarque
`RETURN` fait sortir immédiatement de la fonction.

2.16.4 Traduction MuPAD

`retourne` se traduit `MuPAD` par `return` :

```
addition:=proc(a,b)
begin
return(a+b)
end_proc;
```

Remarque
`return` fait sortir immédiatement de la fonction.

2.16.5 Traduction TI89 92

```
:addition(a,b)
:Func
:Return a+b
:EndFunc
```

Remarque
`Return` fait sortir immédiatement de la fonction.

2.17 Les listes

2.17.1 Traduction Algorithmique

On utilise les { } pour délimiter une liste.
Attention ! ! !
En algorithmique, on a choisi cette notation car c'est celle qui est employée par les calculatrices...à ne pas confondre avec la notion d'ensemble en mathématiques : dans un ensemble l'ordre des éléments n'a pas d'importance mais dans une liste l'ordre est important...
Par exemple {} désigne la liste vide et {1, 2, 3} est une liste de 3 éléments.
`concat` sera utilisé pour concaténer 2 listes ou une liste et un élément ou un élément et une liste :
`{1, 2, 3}=>TAB`
`concat(TAB,4)=>TAB` (maintenant `TAB` désigne {1, 2, 3, 4}
`TAB[2]` désigne le deuxième élément de `TAB` ici 2.

2.17.2 Traduction Xcas

Avec `Xcas` il existe différentes notions : la liste ou le vecteur, la séquence et l'ensemble.

- Les listes et les vecteurs
  Une liste ou un vecteur est délimité par `[ ]` et les éléments situés à l'intérieur des crochets sont séparés par des virgules.
- Les séquences
  Une séquence n'a pas de délimiteurs, les éléments sont séparés par des virgules, mais on doit parfois parenthèser par `( )` (on écrit `1,2,3,4` ou `(1,2,3,4)` ou encore `seq[1,2,3]`).
- Les ensembles
  Un ensemble est délimité par `%{ %}` et les éléments situés à l'intérieur des délimiteurs sont séparés par des virgules (on écrit `%{ 1,2,3,4%}` ou encore `set[1,2,3,4]`).

Les fonctions pour les listes

La liste vide est désignée par `[ ]` et la séquence vide par `NULL`.
La commande `makelist` permet de fabriquer une liste à partir d'une fonction f.
Les paramètres sont : la fonction, l'intervalle de variation de l'argument de f et le

pas de son incrémentation.
On tape :
$\texttt{f(x)} := \texttt{x}^2$
$\texttt{l} := \texttt{makelist(f, 2, 10, 3)}$ ou encore
$\texttt{l} := \texttt{makelist(x} -> \texttt{x}^2, \texttt{2, 10, 3)}$ ou encore
$\texttt{l} := \texttt{makelist(x} -> \texttt{x}^2, \texttt{2..10, 3)}$ ou encore
$\texttt{l} := \texttt{makelist(sq, 2, 10, 3)}$
On obtient `l= [4,25,64]`
Pour avoir une liste constante on peut taper par exemple :
$\texttt{l} := \texttt{makelist(3, 1..5)}$ on obtient `l= [3,3,3,3,3]]`
On peut écrire `l:=[1,2,3]`.
Attention Les éléments sont indicés à partir de zéro (contrairement à `Maple` ou `MuPAD` qui commencent les indices à 1) :
dans l'exemple `l[0]` vaut `1`.
Si on tape ensuite :
`l[0]:=4` : après cette instruction `l` sera la liste `[4,2,3]`.
La commande `append(l,elem)` permet de mettre à la fin d'une liste `l`, un élément (ou une liste) `elem`.
La commande `prepend(l,elem)` permet de mettre au début d'une liste `l`, un élément (ou une liste) `elem`.
La commande `tail(l)` renvoie la liste `l` privée de son premier élément et,
la commande `head(l)` renvoie le premier élément de la liste.
La commande `concat` permet de concaténer deux listes ou une liste et un élément.
La commande `augment` permet de concaténer deux listes.
La commande `size` ou `nops` renvoie la longueur d'une liste ou d'une séquence.

Les fonctions pour les séquences

La commande `op` transforme une liste en une séquence.
On a la relation :
si `l` est une liste `op(l)` est une séquence.
Exemple
`l:=[1,2,3]`
`s:=op(l)` (`s` est la séquence `1,2,3`),
`a:=[s]` (`a` est la liste `l` égale à `[1,2,3]`),
Pour concaténer deux séquences il suffit d'écrire :
`s1:=(1,2,3)`
`s:=(s1,4,5)`
ou encore
`s:=s1,4,5` car la virgule (`,`) est prioritaire par rapport à l'affectation (`:=`).
La commande `seq` permet de fabriquer une séquence à partir d'une expression. Les paramètres sont : l'expression, la variable=l'intervalle de variation (le pas d'incrémentation de la variable est toujours 1).
On tape :
`seq(j^2,j=1..4)`
On obtient ;
`(1,4,9,16)`
On peut aussi utiliser $ qui est une fonction infixée.

On tape :
`(j^2) $ (j=1..4)`
On obtient ;
`(1,4,9,16)`

Les fonctions pour les ensembles

Soit `A:=set[1,2,3,4]; B:=set[3,4,4,6];`
`union(A,B)` désigne l'union de `A` et `B`,
`intersect(A,B)` désigne l'intersection de `A` et `B`,
`minus(A,B)` désigne la différence de `A` et `B`.
On a :
`union(A,B)=set[1,2,3,4,5,6]`
`intersect(A,B)=set[3,4]`
`minus(A,B)=set[1,2]`

2.17.3 Traduction MapleV

En `Maple` on utilise `{ }`, comme en mathématiques, pour représenter un **ensemble**.
Exemple : `De := {1, 2, 3, 4, 5, 6}`
Dans un ensemble, l'ordre n'a pas d'importance. La répétition est interdite.
Pour délimiter une **liste**, on utilise `[ ]`.
L'ordre est pris en compte, la répétition est possible.
Exemple : `[Pile, Face, Pile]`
Une **séquence** est une suite d'objets, séparés par une virgule.
Si `C` est un ensemble ou une liste, `op(C)` est la séquence des objets de `C`.
`nops(C)` est le nombre d'éléments de `C`.
Ainsi, si `L` est une liste, `{op(L)}` est l'ensemble des objets (non répétés) de `L`.
Exemples
On écrit :
`S:=NULL:` (pour la séquence vide)
`S:=S,A:` (pour ajouter un élément à `S`)
Une liste est une séquence entourée de crochets :
`L:=[S]:`
`L[i]` est le iéme élément de la liste `L`.
On peut aussi revenir à la séquence : `S:=op(L):`

2.17.4 Traduction MuPAD

Une liste est une suite d'expressions entre un crochet ouvrant `[` et un crochet fermant `]`.
`[ ]` désigne la liste vide.
Exemple :
`l:=[1,2,2,3]`
`nops(l)` renvoie le nombre d'éléments de la liste `l`.
`l[1]` ou `op(l,1)` renvoie le premier élément de la liste `l` : les éléments sont numérotés de 1 à `nops(l)`.

`op(l)` renvoie `1,2,2,3`
`append(l,4)` ajoute l'élément `4` à la fin de la liste `l`.
De plus, les listes peuvent être concaténées avec le signe `.` (un point), par exemple `[1,2].[2,3]=[1,2,2,3]`.
`Attention` une suite d'expressions entre une accolade ouvrante { et une accolade fermante } désigne un ensemble.
Exemple d'ensembles et de fonctions agissant sur les ensembles :
`A:={a,b,c}; B:={a,d}`
`A union B` désigne `{a,b,c,d}`
`A intersect B` désigne `{a}`
`A minus B` désigne `{b,c}`

2.17.5 Traduction TI89/92

`augment` permet de concaténer deux listes.
{} désigne la liste vide. Pour travailler avec des listes, on peut initialiser une liste de `n` éléments avec la commande `newlist`, par exemple :
`newlist(10)=>L` (`L` est alors une liste de 10 éléments nuls).
On peut utiliser les commandes suivantes :
`seq(i*i, i, 1, 10)` qui désigne la liste des carrés des 10 premiers entiers, ou `seq(i*i, i, 0, 10, 2)` qui désigne la liste des carrés des 5 premiers entiers pairs (le pas est ici égal à 2).
Exemple :
$\texttt{seq}(\texttt{i}*\texttt{i},\texttt{i},0,10,2) \Rightarrow \texttt{L}$ va par exemple créer la liste :
{0, 4, 16, 36, 64, 100} c'est à dire la liste des carrés de 0 à 10 avec un pas de 2 que l'on stocke dans `L`.
`L[j]` qui désigne le jème élément de la liste `L`.
On peut aussi écrire :
`2=>L[2]`
La liste `L` est alors {0, 2, 16, 36, 64, 100}
ou si `L` est de longueur `n` on peut rajouter un élément (par exemple 121) à `L` en écrivant :
`121 => L[n+1]`
Dans l'exemple précédent `n=6` on peut donc écrire :
`121 => L[7]` (`L` est alors égale à {0, 2, 16, 36, 64, 100, 121}).
`left (L, 5)` désigne les 5 premiers éléments de la liste `L`.

2.18 Un exemple : le crible d'Eratosthène

2.18.1 Description

Pour trouver les nombres premiers inférieurs ou égaux à N :

1. On écrit les nombres de 2 à N dans une liste.
2. On met 2 dans la case P .
3. Si $P \times P \leq N$ il faut traiter les éléments de P à N : on barre tous les multiples de P à partir de $P \times P$.
4. On augmente P de 1.
   Si $P \times P$ est strictement supérieur à N, on arrête

5. On met le plus petit élément non barré de la liste dans la case P. On reprend à l'étape 3.

2.18.2 Écriture de l'algorithme

```
Fonction crible(N)
local TAB PREM I P
// TAB et PREM sont des listes
{} =>TAB
{} =>PREM
//on suppose que les indices d'une liste debutent par 0
//si ils commencent par 1, mettre pour I de 1 a N
pour I de 0 a N faire
  concat(TAB, I) => TAB
fpour
//On met 0 dans TAB[1] car 1 n'est pas premier
//barrer 1 a ete realise en le remplacant par 0
0 => TAB[1]
//TAB est la liste 0 0 2 3 4 ...N
2 => P
// On a fait les points 1 et 2
tantque P*P <= N faire
  pour I de P a E(N/P) faire
//E(N/P) designe la partie entiere de N/P
    0 => TAB[I*P]
  fpour
// On a barre tous les multiples de P a partir de P*P
  P+1 => P
//On cherche le plus petit nombre <= N non barre (non nul)
// entre P et N
  tantque (P*P <= N) et (TAB[P]=0) faire
    P+1 => P
  ftantque
ftantque
//on ecrit le resultat dans une liste PREM
pour I de 2 a N faire
  si TAB[I]!= 0 alors
     concat(PREM, I) => PREM
  fsi
fpour
retourne PREM
```

2.18.3 Traduction Xcas

Première version

```
//renvoie la liste des nombres premiers<=n selon eratosthene
crible0(n):={
  local tab,prem,p,j;
```

```
  tab:=[0,0];
  prem:=[];
  for (j:=2;j<=n;j++){
    tab:=append(tab,j);
  }
  p:=2;
  while (p*p<=n) {
    for (j:=p;j*p<=n;j++){
      tab[eval(j*p)]:=0;
    }
    p:=p+1;
    while ((p*p<=n) and (tab[p]==0)) {
      p:=p+1;
    }
  }
  for (j:=2;j<=n;j++) {
    if (tab[j]!=0) {
      prem:=append(prem,j);
    }
  }
  return(prem);
};
```

ou avec les instructions françaises

```
//renvoie la liste des nombres premiers<=n selon eratosthene
crible0(n):={
  local tab,prem,p,j;
  tab:=[0,0];
  prem:=[];
  pour j de 2 jusque n faire
    tab:=append(tab,j);
  fpour;
  2=>p;
  tantque (p*p<=n) faire
    pour j de p jusque n/p faire
      tab[eval(j*p)]:=0;
    fpour
    p+1=>p;
    tantque ((p*p<=n) et (tab[p]==0)) faire
      p:=p+1;
    ftantque;
  ftantque;
  pour j de 2 jusque n faire
    si (tab[j]!=0) alors
      prem:=append(prem,j);
    fsi
  fpour
  retourne(prem);
```

```
};
```

Une version plus efficace

1. On utilise `seq` pour définir `tab` et on remplace

   ```
   tab:=[0,0];
   pour j de 2 jusque n faire
   tab:=append(tab,j);
   fpour;
   ```

   par

   ```
   tab:=seq(j,j,0,n);
   tab[0]=<0; tab[1]=<0;
   ```

2. On remplace tout d'abord les affectations `:=` d'éléments d'une liste par une affectation par référence `=<` c'est à dire sans faire de recopie ce qui est plus efficace.

3. Puis, lorsque qu'on barre les multiples de `p` premier on remplace les multiplications de `tab[eval(j*p)]:=0;` par des additions on faisant varier `j` avec un pas : On remplace donc

   ```
   pour j de p jusque n/p faire
         tab[eval(j*p)]:=0;
       fpour;
   ```

   par

   ```
   pour j de p*p jusque n pas p faire
         tab[j]=<0;
       fpour;
   ```

On obtient :

```
//renvoie la liste des nombres premiers<=n selon erathostene
crible(n):={
  local tab,prem,p;
  tab:=seq(j,j,0,n);
  prem:=[];
  tab[0]=<0; tab[1]=<0;
  p:=2;
  while (p*p<=n) {
    for (j:=p*p;j<=n;j+=p){
      tab[j]=<0;
    }
    p:=p+1;
    //afficher(tab);
    while ((p*p<=n) and (tab[p]==0)) {
      p:=p+1;
    }
  }
```

```
  for (j:=2;j<=n;j++) {
    if (tab[j]!=0) {
      prem:=append(prem,j);
    }
  }
  return(prem);
}:;
```

ou avec la version française :

```
//renvoie la liste des nombres premiers<=n selon eratosthene
crible(n):={
  local tab,prem,p,j;
  tab:=seq(j,j,0,n);
  tab[0]=<0; tab[1]=<0;
  prem:=[];
  p:=2;
  tantque (p*p<=n) faire
    pour j de p*p jusque n pas p faire
      tab[j]=<0;
    fpour
    p:=p+1;
    tantque ((p*p<=n) et (tab[p]==0)) faire
      p=<p+1;
    ftantque;
  ftantque;
  pour j de 2 jusque n faire
    si (tab[j]!=0) alors
      prem=<append(prem,j);
    fsi
  fpour
  retourne(prem);
};
```

On tape :
`crible0(10000)` On obtient la réponse avec "Temps mis pour l'évaluation : 12.35" On tape :
`crible0(20000)` On obtient la réponse avec "Temps mis pour l'évaluation : 49.6" le temps est quadratique (on a $12.35 * 2^2 = 49.4$).
On tape :
`crible(100000)` On obtient la réponse avec "Temps mis pour l'évaluation : 6.52" On tape :
`crible(200000)` On obtient la réponse avec "Temps mis pour l'évaluation : 17.77".
Le temps est en théorie en $n \ln(n)$ (on a $6.52 * 2 * \ln(20000)/\ln(10000) \simeq 14$).

2.18.4 Traduction TI89/92

Voici la fonction crible :
- n est le paramètre de cette fonction.
- crible(n) est égal à la liste des nombres premiers inférieurs ou égaux à n.

```
:crible(n)
:Func
:local tab,prem,i,p
:newList(n)=>tab
:newList(n)=>prem
:seq(i,i,1,n) =>tab
:0 => tab[1]
:2 => p
:While p*p <= n
:For i,p,floor(n/p)
:0 => tab[i*p]
:EndFor
:p+1 => p
:While p*p<= n and tab[p]=0
:p+1 => p
:EndWhile
:EndWhile
:0 => p
:For i,2,n
:If tab[i]!= 0 Then
:p+1 =>p
:i =>prem[p]
:EndIf
:EndFor
:Return left(prem,p)
:EndFunc
```

2.19 Un exemple de fonction récursive

2.19.1 L'énoncé de l'exercice

On considère la fonction f définie sur $\mathbb{R}$ à valeurs dans $\mathbb{R}$ et qui vérifie :
$f(x) = x - 10$ si $x > 100$
$f(x) = f(f(x + 11))$ si $x \leq 100$
Calculer f(x) pour $x \in \mathbb{R}$.
Tracer le graphe de f lorsque $x \in [-5, 5]$.

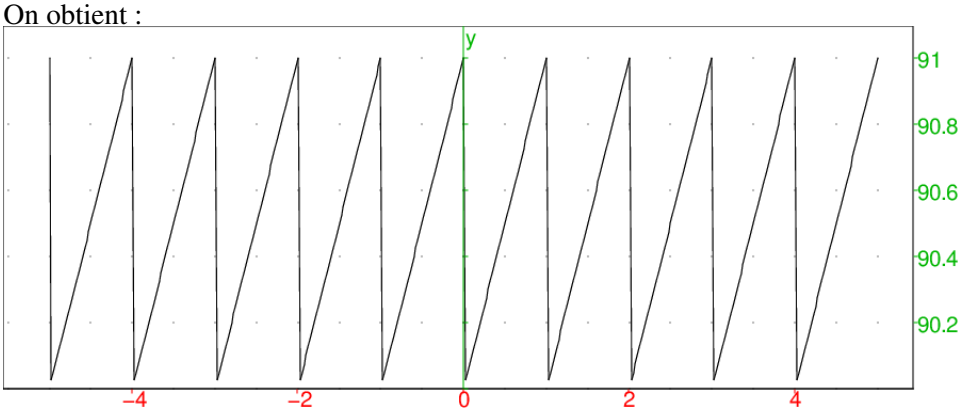
2.19.2 Solution avec un programme récursif `Xcas`

On tape :

```
fz(x):={
```

```
si x> 100 alors retourne x-10; fsi;
retourne fz(fz(x+11));
}:;
```

On tape :
`fz(k)$(k=0..10)`
On obtient :
`[91,91,91,91,91,91,91,91,91,91,91]`
On tape :
`fz(k)$(k=-10..0)`
On obtient :
`[91,91,91,91,91,91,91,91,91,91,91]`
On tape :
`fz(90+k)$(k=0..10)`
On obtient :
`[91,91,91,91,91,91,91,91,91,91,91]`
On tape :
`fz(90+k/10)$(k=0..10)`
On obtient :
`[91,901/10,451/5,903/10,452/5,181/2,453/5,907/10,454/5,`
`909/10,91]`
soit avevec`evalf` :
`[91.0,90.1,90.2,90.3,90.4,90.5,90.6,90.7,90.8,90.9,91.0]`
On tape :
`plotfunc(fz(x),x=-5..5)`
On obtient :



On tape :
`evalf(fz(k/1000)$(k=0..10))`
On obtient :
`[91.0,90.001,90.002,90.003,90.004,90.005,90.006,90.007,`
`90.008,90.009,90.01]`

2.19.3 Solution papier-crayon

On calcule tout d'abord les valeurs de $f(n)$ pour $n \in \mathbb{Z}$ et en particulier :
$f(111), f(101), f(100), f(99)..f(n)$ pour $n \in \mathbb{Z}$
On a :
$f(111) = 111 - 10 = 101$

$f(110) = 110 - 10 = 100$
.. $f(102) = 102 - 10 = 92$
$f(101) = 101 - 10 = 91$
$f(100) = f(f(111)) = f(101) = 91$
$f(99) = f(f(110)) = f(100) = 91$
$f(98) = f(f(109)) = f(99) = 91$
$f(90) = f(f(101)) = f(101 - 10) = f(91) = f(92) = ...f(100) = 91$ Supposons que pour tout n vérifiant $n \geq 91$ on a $f(n + 10) = 91$, montrons alors que $f(n - 1) = 91$.
On a $f(n - 1) = f(f(n + 10)) = f(91) = 91$
Calcul de $f(x)$ pour $99 < x \geq 100$.
On a $x = 99 + a$ avec $0 < a < 1$ donc $f(x) = f(99 + a) = f(f(110 + a)) = f(100 + a) = 90 + a$ car $100 + a > 100$ Donc pour $0 < a \geq 1$ on a $f(99 + a) = 90 + a$.
pour $98 < x \geq 99$ on a $f(x) = f(98+a) = f(f(109+a)) = f(99+a) = 90+a$.
De même :
pour $n < x \geq n + 1 \geq 99$, on a pour $0 < a \geq 1$:
$f(x) = f(n + a) = f(f(n + 11 + a)) = f(n + 1 + a) = 90 + a$.

2.20 Un exemple de fonction vraiement récursive

2.20.1 La définition

Voici la définition de la fonction `a` dite fonction de `ackermann` qui est une fonction de $\mathbb{N} \times \mathbb{N}$ dans $\mathbb{N}$:

```
a(0,y)=y+1,
a(x,0)=a(x-1,1) si x>0,
a(x,y)=a(x-1,a(x,y-1) si x>0 et si y>0.
```

Ainsi on a :

```
a(0,0)=1
a(1,0)=a(0,1)=2
a(1,1)=a(0,a(1,0))=a(0,2)=3
a(1,2)=a(0,a(1,1))=4
a(1,n)=a(0,a(1,n-1))=1+a(1,n-1)=...=n+2
a(2,0)=a(1,1)=3
a(2,1)=a(1,a(2,0))=a(1,3)=5
a(2,2)=a(1,a(2,1))=2+a(2,1)=7
a(2,n)=a(1,a(2,n-1))=2+a(2,n-1)=2n+3
a(3,0)=a(2,1)=5
a(3,1)=a(2,a(3,0))=2*a(3,0)+3=13
a(3,2)=a(2,a(3,1))=2*a(3,1)+3=29
a(3,n)=a(2,a(3,n-1))=2*a(3,n-1)+3
       =2^(n+1)+3*(2^n+2^(n-1)...+1)
       =2^(n+1)+3*(2^(n+1)-1)= 2^(n+3)-3
```

On a donc par exemple : `a(3,5)=2^8-3=253`
Les calculs sont vite gigantesques on a par exemple :

`a(4,1)=a(3,a(4,0))=a(3,a(3,1))=a(3,13)=65533`
`a(4,2)=a(3,a(4,1))=a(3,65533)=2^65536-3`
`a(4,3)=a(3,a(4,2))=a(3,2^65533-3)=2^(2^65536-3)-3`
On a donc :
`a(4,y)=a(3,a(4,y-1))=2^(a(4,y-1)+3)-3`
`=2^(2^(a(4,y-2)+3)-3+3)-3=2^(2^(a(4,y-2)+3))-3`
et donc
`a(4,y)=2^(2^..(2^(a(4,0)+3))..)-3= 2^(2^..(2^16)..)-3`,
avec `2^` qui se répète y fois, et comme `16=2^(2^2)` on a,
`a(4,y)=2^(2^..(2^2)..)-3`,
avec `2` qui se répète $y+3$ fois.

2.20.2 Le programme

Voici un premier programme :

```
akc(x,y):={
if (x==0) return y+1;
if (y==0) return akc(x-1,1);
return ack(x-1,ack(x,y-1));
}
```

ou bien en utilisant `ifte` :

```
ack(x,y):=ifte(x==0,y+1,
               ifte(y==0,ack(x-1,1),ack(x-1,ack(x,y-1))));
```

On remarque que le temps pour calculer la valeur pour `a(3,5)` est de 5.06s ce qui est tres long, mais on peut donc améliorer le programme en arrêtant la récursivité lorsque `x==3`.
On écrit :

```
a(x,y):={
if (x==0) return y+1;
if (x==1) return y+2;
if (x==2) return 2*y+1;
if (x==3) return 2^(y+3)-3;
if (y==0) return a(x-1,1);
return a(x-1,a(x,y-1));
}
```

On peut aussi améliorer le programme en arrêtant la récursivité lorsque `x==4`.
On écrit :

```
a(x,y):={
if (x==0) return y+1;
if (x==1) return y+2;
if (x==2) return 2*y+1;
if (x==3) return 2^(y+3)-3;
if (x==4) {
local p:=1;
```

```
for (j:=1;j<=y+3;j++) p:=2^p;
return p-3;
}
if (y==0) return a(x-1,1);
return a(x-1,a(x,y-1));
}
```

Essayez `a(4,1)=65533`, `a(4,2)`....

Chapitre 3

Exercices simples

3.1 Savoir si une liste est croissante

Une liste de taille 1 est croissante.
Si la taille est de la liste `l` est `s` il faut faire attention que les indices de `l` vont de `0` à `s-1`. On compare donc `l[0]` et `l[1]`, puis `l[j-1]` et `l[j]` donc on initialise `j` à `1` et tantque `j<s` on compare `l[j-1]` et `l[j]` si ils ne sont pas dans le bon ordre on s'arrête sinon on augmente `j` de 1.
On écrit :

```
est_crois(l):={
local j,s;
s:=size(l);
j:=1;
tantque j<s faire
  si l[j-1]>l[j] alors return false fsi;
  j:=j+1;
ftantque;
return true;
}:;
```

On tape :
`est_crois([1,2,0,4])`
On obtient :
`faux`
On tape :
`est_crois([1,2,3,4])`
On obtient :
`vrai`
On tape :
`est_crois([1,1,1,1])`
On obtient :
`vrai`
On peut améliorer ce programme pour répondre 2 lorsque la suite est constante.

```
est_croisoucste(l):={
local j,s;
```

```
s:=size(l);
j:=1;
tantque j<s faire
  si l[j-1]>l[j] alors return false fsi;
  j:=j+1;
ftantque;
si l[0]==l[s-1] alors return 2; fsi;
return true;
}:;
```

Exercice
Écrire un programme qui renvoie :
0 si la liste n'est pas monotone
1 si la liste est croissante
2 si la liste est décroissante
3 si la liste est constante.
On tape par exemple :

```
est_monotone(l):={
local j,s;
s:=size(l);
j:=1;
tantque (j<s-1) et (l[j-1]==l[j]) faire j:=j+1 ftantque;
si l[j]==l[j-1] alors return 3; fsi;
si l[j-1]<l[j] alors
tantque j<s faire
  si l[j-1]>l[j] alors return 0 fsi;
  j:=j+1;
ftantque;
return 1;
sinon
tantque j<s faire
  si l[j-1]<l[j] alors return 0 fsi;
  j:=j+1;
ftantque;
return 2;
fsi;
}:;
```

3.2 Écriture décimale et fractionnaire

Avec sa calculatrice un élève a fait le quotient de 2 entiers inférieurs à 1000 et a trouvé 0.6786389.
Son professeur veut plus de décimales, mais lélève ne se souvient plus des 2 entiers. Pouvez-vous l'aider à retrouver ces 2 entiers à l'aide d'un programme ? Pouvez-vous l'aider à retrouver ces 2 entiers à l'aide des fractions continues ? **Une solution**

1. Avec un programme qui fait un balayage :
   Si $a/b \simeq 0.6786389$ on a $a \simeq= b * 0.6786389$ avec $0 < b \geq 1000$ et $0 < a \geq 1000$ On tape :

```
exodf():={
local L,a,b;
L:=NULL;
pour b de 1 jusque 1000 faire
  a:=b*0.6786389;
  si abs(a-round(a))<1e-06 alors
    L:=L,[a,b];
  fsi;
fpour;
return L;
}:;
```

On tape :
`exodf()`
On obtient :
`[358.9999781,529]`
On vérifie et on tape :
`evalf(359/529)`
On obtient :
`0.678638941399`
Les 2 entiers sont donc 359 et 529.

2. Avec un développement de 0.6786389 en une fraction continue
On fait le programme :

```
exodfc2f(d):={
local a,j,L,f;
a:=floor(d);
L:=a;
f:=d-a;
pour j de 1 jusque 5 faire
 d:=1/f;
a:=floor(d);
f:=d-a;
L:=L,a;
d:=1/f;
fpour;
f:=1/L[5]
pour j de 4 jusque 1 pas -1 faire
  f:=f+L[j];
  f:=1/f;
 fpour;
return f+L[0],[L];
}:;
```

On tape :
`d:=0.6786389`
On tape :
`exodfc2f(d)`
On obtient :
`[359/529,[0,1,2,8,1,18]]`

Ou on utilise `dfc` pour obtenir le développement de 0.6786389 en une fraction continue, puis `dfc2f` pour transformer une fraction continue en une fraction.
On tape :
```
dfc(d,1e-07)
```
On obtient :
```
[0,1,2,8,1,18]
```
On tape :
```
dfc2f([0,1,2,8,1,18])
```
On obtient :
```
359/529
```
On tape :
```
evalf(359/529)
```
On obtient :
```
0.678638941399
```
On tape :
```
dfc(d,1e-09)
```
On obtient :
```
[0,1,2,8,1,18,86]
```
On tape :
```
dfc2f([0,1,2,8,1,18,86])
```
On obtient :
```
30893/45522
```
On tape :
```
evalf(30893/45522)
```
On obtient :
```
0.678638899873
```
On tape :
```
L:=dfc(d)
```
On obtient :
```
[0,1,2,8,1,18,86,3,1]
```
On tape :
```
apply(dfc2f,[[0,1,2],[0,1,2,8],[0,1,2,8,1],[0,1,2,8,1,18],
[0,1,2,8,1,18,86],[0,1,2,8,1,18,86,3],L])
```
On obtient :
```
[2/3,17/25,19/28,359/529,30893/45522,93038/137095,123931/182617]
```

Chapitre 4

Les exercices d'algorithmiques au baccalauréat série S

4.1 Trois exercices classiques

4.1.1 La série harmonique

On considère la suite $u_n = \sum_{j=1}^{n} \frac{1}{j}$.

— Montrer que $u_{2n} - u_n$ est une somme de n termes et que chaque terme est supérieur ou égal à $\frac{1}{2n}$. En déduire que pour tout n, on a $u_{2n} - u_n \geq \frac{1}{2}$.

— En déduire que u_n tend vers $+\infty$ quand n tend vers $+\infty$.

— On écrit $u_{16} = \sum_{j=1}^{16} \frac{1}{j} = 1 + \frac{1}{2} + (\frac{1}{3} + \frac{1}{4}) + (\frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8}) + \sum_{j=9}^{16} \frac{1}{j}$. Montrer que $u_{16} > 3$.

— Écrire un algorithme permettant de trouver la plus petite valeur de n pour laquelle $3 < u_n$.

— Écrire un algorithme permettant de trouver la plus petite valeur de n pour laquelle $p \leq u_n$ avec $p \in \mathbb{R}$. Tester ce programme pour $p = 3, 5, 10, 11$

La solution

$u_{2n} - u_n = \dfrac{1}{n+1} + .. + \dfrac{1}{n+n}$ et on a : $\dfrac{1}{n+n} \leq \dfrac{1}{n+k}$ pour $k = 1..n$.

$u_{2n} - u_n$ a donc n termes et chaque terme est supérieur ou égal à $\dfrac{1}{2n}$ donc

$u_{2n} - u_n \geq \dfrac{n}{2n} = \dfrac{1}{2}$.

Donc $u_{2n} - u_n$ ne tend pas vers zéro quand n tend vers $+\infty$ donc u_n n'est pas convergente. Comme u_n est croissante et non convergente, u_n n'est pas bornée donc u_n tend vers $+\infty$ quand n tend vers $+\infty$

L'algorithme en langage naturel

```
Entrée :          p entier
Variables :       j entier,S reel
Initialisation :  Affecter à S la valeur 0
                  Affecter à j la valeur 0
Traitement :      Tant que S<p faire
                    Affecter à j la valeur j+1
                    Affecter à S la valeur S+1/j
                  FinTantque
Sortie :          j
```

La solution en langage `Xcas`

```
harmonique(p):={
local j, S;
S:=0;
j:=0;
tantque S<p faire
  j:=j+1;
  S:=S+1/j;
ftantque;
retoune j;
}:;
```

On tape : `harmonique(3)`
On obtient : `11`
On tape : `harmonique(5)`
On obtient : `83`
On tape : `harmonique(10)`
On obtient : `12367`
On tape : `harmonique(11)`
On obtient : `33617`

4.1.2 Le compte bancaire

Lors de la naissance de Pierre son grand-père dépose sur un compte bancaire 100 euros. À chaque anniversaire de Pierre, il dépose sur ce compte 100 euros auquel il ajoute le double de l'age de Pierre.

- Écrire un algorithme permettant de trouver le montant se trouvant sur son compte le lendemain des 10 ans de Pierre
- Modifier l'algorithme précédent pour déterminer à quel age Pierre pourra-t-il acheter un objet de 2000 euros ? de P euros ?
- Modifier les algorithmes précédents lorsque le compte sur lequel est déposé l'argent rapporte 2.5% l'an (net d'impots, de taxes et de droit de succession !)

La solution en langage naturel
Montant du compte lorsque Pierre a n ans :

```
Entrée :         n entier
Variables :      S reel, j entier
Initialisation : Affecter à S la valeur 100
Traitement :     Pour j allant de 1 à n faire
                     Affecter à S la valeur S+100+2*j
                 FinPour
Sortie :         S
```

Somme disponible>P et age correspondant de Pierre

```
Entrée :         P reel
Variables :      S reel, j entier
Initialisation : Affecter à S la valeur 100
```

```
                Affecter à j la valeur 0
Traitement :    Tantque S<P
                  Affecter à j la valeur j+1
                  Affecter à S la valeur S+100+2*j
                FinTantque
Sortie :        S,j
```

Si le compte rapporte 2.5% par an, on écrira dans le traitement des deux algorithmes précédents :
`Affecter à S la valeur S*1.0.25+100+2*j` (au lieu de `Affecter à S la valeur S+100+2*j`)
Éventuellement on renverra la valeur de S arrondie avec seulement 2 chiffres après la virgule (`evalf(S,2)`).
La solution en langage `Xcas`

```
banque(n):={
//Montant du compte lorsque Pierre a n ans
local S,j;
S:=100;
pour j de 1 jusque n faire
  S:=S+100+2*j;
fpour;
retourne S;
}:;
banques(P):={
//Somme disponible>=P et age correspondant de Pierre
local S,j;
S:=100;
j:=0;
tantque S<P faire
  j:=j+1;
  S:=S+100+2*j;
ftantque;
retourne S,j;
}:;
banquier(n):={
//On applique un interet de 2.5 pour cent
//Montant du compte lorsque Pierre a n ans
local S,j;
S:=100;
pour j de 1 jusque n faire
  S:=S*1.025+100+2*j;
fpour;
retourne evalf(S,2);
}:;
banquiers(P):={
//On applique un interet de 2.5 pour cent
//Somme disponible>=P et age correspondant de Pierre
local S,j;
```

```
S:=100;
j:=0;
tantque S<P faire
  j:=j+1;
  S:=S*1.025+100+2*j;
ftantque;
retourne evalf(S,2),j;
}:;
```

On tape : `banque(10)`
On obtient : `1210`
On tape : `banques(2000)`
On obtient : `2106,17`
On tape : `banquier(10)`
On obtient : `1367.02`
On tape : `banquiers(2000)`
On obtient : `2027.75,14`

4.1.3 La suite de Syracuse

Soit a un entier positif. On veut étudier la suite de Syracuse définie par :
$u_0 = a$
$u_n = u_{n-1}/2$ si u_{n-1} est pair
$u_n = 3 * u_{n-1} + 1$ si u_{n-1} est impair.
Cette suite se termine toujours (? ? ?) par 1, 4, 2, 1, 4, 2, 1... mais on ne sait pas le démontrer ! ! !

- — Écrire un algorithme permettant de trouver la première valeur n de k pour laquelle $u_k = 1$.
- — Modifier cet algorithme afin de connaitre en plus la plus grande valeur m prise par u_k lorsque $k = 0..n$.
- — Tester ce programme pour $a = 3$, $a = 5$, $a = 7$, $a = 75$ et $a = 97$
- — Tracer dans un repère orthogonal, pour $a = 1...100$, les points de coordonnées (a, n) (en rouge) et les points de coordonnées (a, m) (en noir).

La solution en langage naturel
Algorithme qui renvoie n. n est la première valeur de k pour laquelle $u_k = 1$

```
Entrée :          a
Variables :       k
Initialisation :  Affecter à k la valeur 0
Traitement :      Tant que a!=1 faire
                    Si a est pair alors
                      Affecter à a la valeur a/2
                    Sinon
                      Affecter à a la valeur 3a+1
                    FinSi
                    Affecter à k la valeur k+1
                  FinTantque
Sortie :          k
```

Algorithme qui renvoie m, n où m est la plus grande valeur prise par u_k lorsque $k = 0..n$ et n est la première valeur de k pour laquelle $u_k = 1$

```
Entrée :          a
Variables :       k,m
Initialisation :  Affecter à m la valeur a
                  Affecter à k la valeur 0
Traitement :      Tant que a!=1 faire
                    Si a est pair alors
                      Affecter à a la valeur a/2
                    Sinon
                      Affecter à a la valeur 3a+1
                      Si a>m alors Affecter à m la valeur a
                    FinSi
                    Affecter à k la valeur k+1
                  FinTantque
Sortie :          m,k
```

La solution en langage `Xcas`

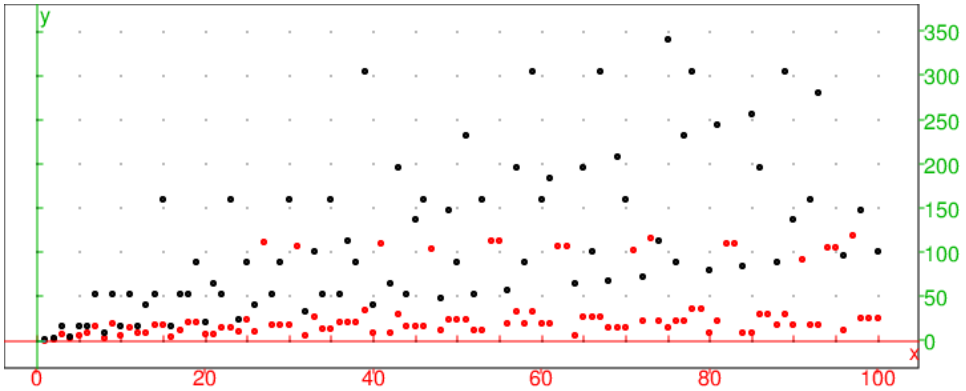
```
syracuse(a):={
  local k,m;
  k:=0;
  m:=a;
  tantque a!=1 faire
    si irem(a,2)==0 alors
      a:=iquo(a,2);
    sinon
      a:=a*3+1;
      si a>m alors m:=a; fsi;
    fsi;
    k:=k+1;
  ftantque;
retourne m,k;
}:;
```

On tape : `syracuse(3)` On obtient : `16,7`
On tape : `syracuse(5)` On obtient : `16,5`
On tape : `syracuse(7)` On obtient : `52,16`
On tape : `syracuse(75)` On obtient : `340,14`
On tape : `syracuse(97)` On obtient : `9232,118`
On tape le programme qui affiche en rouge les points (a, n) et en noir les points (a, m) lorsque $a = 1..100$.

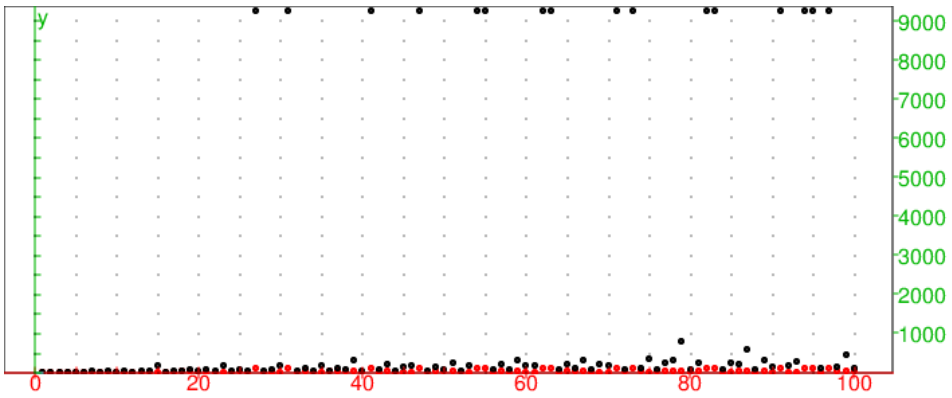
```
syracuse100():={
local a,n,m,L;
L:=NULL;
pour a de 1 jusque 100 faire
  m,n:=syracuse(a);
  L:=L,point(a,n,affichage=1),point(a,m,affichage=1);
```

```
fpour;
retourne L;
}:;
```

On tape : `syracuse100()` et on obtient :



Mais en changeant le repère, on voit les points tels que point(97,9232)



4.2 En 2009 Centre étranger

On considère la fonction $f(x) = xe^x - 1$ sur $\mathbb{R}$.

— Calculer $f(0$ et $f(1)$. En étudiant les variations de f sur $\mathbb{R}$, montrer que $f(x) = 0$ admet une solution unique dans [0 ;1].

— On considère l'algorithme :

```
Entrée :          f la fonction pr\'ec\'edente
                  n un entier
Variables :       a,b,m,p
Initialisation :  Affecter à a la valeur 0
                  Affecter à b la valeur 1
Traitement :      Tant que b-a>10^-n faire
                    Affecter à m la valeur (a+b)/2
                    Affecter à p la valeur f(a)*f(m)
                    Si p>0
                      Affecter à a la valeur m
                    Sinon
                      Affecter à b la valeur m
```

```
                    FinSi
                  FinTantque
Sortie :          a,b
```

On fait fonctionner cet algorithme avec $n = 1$.
Donner les valeurs que prennent successivement les différents paramètres.

— Que détermine cet algorithme ?
Quel influence le nombre n a-t-il sur le résultat obtenu ?

La solution

On a $f(0) = -1$ et $f(1) \simeq 1.71828182846$
Sur $]-\infty;0]$ $f(x) = xe^x - 1 \leq -1 < 0$ donc f ne s'annule pas.
La fonction f est continue et est strictement croissante sur $[0;+\infty[$ puisque sa dérivée $f'(x) = e^x(x+1)$ est négative sur $]-\infty;-1[$ et positive sur $]-1;+\infty[$.
Donc d'après le théorème des valeurs intermédiaires $f(x) = 0$ a une solution unique comprise entre 0 et 1 puisque $f(0) < 0$ et $f(1) > 0$.
L'algorithme trouve un encadrement de longueur inférieure à $1O^{-n}$ de cette solution : a chaque étape on partage l'intervalle $[a;b]$ en deux (dichotomie). Si m est le milieu de $[a;b]$, on regarde si $f(a)$ et $f(m)$ sont de même signe : si oui, m peut remplacer a et sinon m peut remplacer b et le zéro se trouve toujours entre a et b.
Lorsque $n = 1$ cet encadrement est de longueur $0,1$
Initialisation : a=0 et b=1
Etape 1 : m=0.5, $p = f(0)f(0.5) = 0.17563936465$, a=0.5, b=1
Etape 2 : m=0.75, $p = f(0.5)f(0.75) = -0.103232038761$, a=0.5, b=0.75
Etape 3 : m=0.625, $p = f(0.5)f(0.625) = -0.02944659346$,$a$=0.5, b=0.625
Etape 4 : m=0.5625, $p = f(0.5)f(0.5625) = 0.002244979408$,$a$=0.5625, b=0.625
Résultat : 0.5625,0.625
Arrêt du tantque car $(b-a)$<0.1 et le résultat est donc 0.5625,0.625.

La traduction en langage `Xcas`

```
dichotomie(f,n):={
local a,b,m,p;
a:=0.;
b:=1.;
tantque b-a>10^-n faire
  m:=(a+b)/2;
  p:=f(a)*f(m);
  si p>0 alors
    a:=m;
  sinon
    b:=m;
  fsi;
ftantque;
retourne a,b;
}:;
```

On tape : `dichotomie(f,1)`
On obtient : `0.5625,0.6250`
On tape : `dichotomie(f,2)`

On obtient : `0.5625,0.5703125`
On tape : `dichotomie(f,5)`
On obtient : `0.567138671875,0.56714630127` **Compléments** Dans la fonction `dichotomie` si dessus on a supposé que la fonction `f` avait un zéro sur]0.0,1.0[. Voici une fonction `dichotomie` plus générale que je nomme `dichotom`

```
dichotom(f,a,b,n):={
  local m,p;
  a:=evalf(a);b:=evalf(b);
  si a>b alors m:=b;b:=a;a:=m; fsi;
  p:=f(b)*f(a);
  si p>0 alors return("f(",a,")*f(",b,")>0"); fsi;
  DIGITS:=n+2;
  tantque (b-a)>10.0^-n faire
    m:=(a+b)/2;
    p:=f(a)*f(m);
    si p>0 alors
      a:=m;
    sinon
      b:=m;
    fsi;
  ftantque ;
  retourne a,b;
}:;
```

On tape :
`f(x):=x*exp(x)-1; dichotom(f,0,1,15)` On obtient :
`0.56714329040978351`
On tape :
`dichotom(f,2,1,5)` On obtient :
`"f(",1.0,")*f(",2.0,")>0"`

4.3 En 2010 Amérique du sud

— On considère l'algorithme :

```
Entrée :          n un entier
Variables :       u,S,j
Initialisation :  Affecter à u la valeur 1
                  Affecter à S la valeur 1
                  Affecter à j la valeur 0
Traitement :      Tant que j<n faire
                    Affecter à u la valeur 2u+1-j
                    Affecter à S la valeur S+u
                    Affecter à j la valeur j+1
                  FinTantque
Sortie :          u,S
```

Justifier que pour $n = 3$, le résultat de cet algorithme est 11,21

— On considère les suites $u_n n$ et S_n définies par :
$u_0 = 1$ et $u_{n+1} = 2u_n + 1 - n$

$S_n = u_0 + u_+ ... + u_n$.
Que représente les valeurs données par cet algorithme ?

— Le but de cette question est de trouver u_n en fonction de n Modifier l'algorithme pour qu'il renvoie aussi $u_n - n$. Montrer que $u_n - n = 2^n$

— Calculer $1 + 2 + ... + n$ et $1 + 2 + 2^2 + ... + 2^n$.
En déduire S_n en fonction de n.

La solution
Pour $n = 3$, on a :
Initialisation : u=1, S=1, j=0
Etape 1 : u=3, S=4, j=1
Etape 2 : u=6, S=10, j=2
Etape 3 : u=11, S=21, j=3
Résultat : 11,21
Arrêt du tantque car j>=3 et le résultat est donc 11,21.
Dans le corps du tantque on calcule u, S et j et on a $u = u_j$ et $S = S_j$.
Lorsque $j = n$ le tantque s'arrête et renvoie u_n, S_n.

La traduction en langage `Xcas`

```
suiteserie(n):= {
local u,S,j;
u:=1;
S:=1;
j:=0;
tantque j<n faire
  u:=2u+1-j;
  S:=S+u;
  j:=j+1;
ftantque;
retourne u,S
}:;
```

On veut trouver u_n en fonction de n, on modifie l'algorithme pour qu'il renvoie aussi $u_n - n$, on modifie seulement la sortie en `u,s-n,S`
Pour $n = 0$, on a :
Initialisation : u=1, S=1, j=0
Résultat : 1,1,1
Pour $n = 1$, on a :
Initialisation : u=1, S=1, j=0
Etape 1 : u=3, S=4, j=1
Résultat : 3,2,4
Pour $n = 2$, on a :
Initialisation : u=1, S=1, j=0
Etape 1 : u=3, S=4, j=1
Etape 2 : u=6, S=10, j=2
Résultat : 6,4,10
Pour $n = 3$, on a :
Initialisation : u=1, S=1, j=0

Etape 1 : u=3, S=4, j=1
Etape 2 : u=6, S=10, j=2
Etape 3 : u=11, S=21, j=3
Résultat : 11,8,21
Il semble que $u_n - n = 2^n$.
On a en effet $u_{n+1} - (n+1) = 2u_n + 1 - n - (n+1) = 2(u_n - n)$.
On a $1 + 2 + .. + n = n(n+1)/2$ et $1 + 2 + 2^2 + .. + 2^n = 2^{n+1} - 1$
Donc $S_n = n(n+1)/2 + 2^{n+1} - 1$
On vérifie pour $n = 3$ $u_3 = 2^3 + 3 = 8 + 3 = 11$ et $S_3 = 6 + 2^4 - 1 = 16 + 5 = 21$
On a bien $u_5 = 2^5 + 5 = 32 + 5 = 37$ et $S_5 = 5 * 3 + 64 - 1 = 78$
On tape : `suiteserie(3)` On obtient : `11,21`
On tape : `suiteserie(5)` On obtient : `37,78`

Remarque Il me semble préférable d'écrire cet algorithme avec un `pour`.
Mais attention ! ! ! On doit utiliser la relation de récurrence sous la forme :
$u_n = 2u_{n-1} + 2 - n$ ou bien $u_{n+1} = 2u_n + 2 - (n+1)$
car dans le corps du `pour` on calcule successivement :
u_1, S_1 lorsque $j = 1$, u_2, S_2 lorsque $j = 2$ et u_n, S_n lorsque $j = n$.
Alors que précédement avec `tantque`, on utilise la relation $u_{n+1} = 2u_n + 1 - n$
car on calcule successivement u_1, S_1 lorsque $j = 0$, u_2, S_2 lorsque $j = 1$ et u_n, S_n lorsque $j = n - 1$ et c'est pourquoi la condition d'arrêt du `tantque` est `j<n`.
On tape :

```
suiteserie1(n):= {
local u,S,j;
u:=1;
S:=1;
pour j de 1 jusque n faire
  u:=2u+2-j;
  S:=S+u;
fpour;
retourne u,u-n,S
}:;
```

4.4 En 2011 La Réunion

On considère la fonction $f(x) = 4e^{x/2} - 5$ sur $\mathbb{R}$.
On note C_f le graphe de f dans un repère orthogonal

— Calculer $f(0)$ et $f(1)$.
En étudiant les variations de f sur $\mathbb{R}$, montrer que $f(x) = 0$ admet une solution unique dans [0 ;1].

— On considère l'algorithme :

```
Entrée :          f la fonction precedente
                  p un réel >0
Variables :       a,b
Initialisation :  Affecter à a la valeur 0
                  Affecter à b la valeur -1
Traitement :      Tant que b<0 faire
```

```
                    Affecter à a la valeur a+p
                    Affecter à b la valeur f(a)
                  FinTantque
Sortie :          a-p,a
```

Que fait cet algorithme ?
Que renvoie ce programme lorsque $p = 1$? $p = 0.1$? $p = 0.01$? $p = 0.001$?

La solution et traduction en langage `Xcas`
On a $f(0) = -1$ et $f(1) \simeq 1.5948850828$
La fonction f est continue et est strictement croissante sur $\mathbb{R}$ puisque sa derivé qui vaut $f'(x) = 2e^{x/2}$ est positive.
Donc d'après le théorème des valeurs intermédiaires $f(x) = 0$ a une solution unique comprise entre 0 et 1 puisque $f(0) < 0$ et $f(1) > 0$.
L'algorithme trouve un encadrement de longueur p de cette solution.
Lorsque $p = 1$ cet encadrement est $0, 1$
Lorsque $p = 0.1$ cet encadrement est $0.4, 0.5$ car $f(0.4) \simeq -0.114388967359$<0 et $f(0.5) \simeq 0.136101666751$>0
Lorsque $p = 0.1$ cet encadrement est $0.44, 0.45$ car $f(0.44) \simeq -0.0156930776507$<0 et $f(0.45) \simeq 0.00929086476731$>0
Avec `Xcas`, on tape pour définir la fonction f :
`f(x):=4*exp(x/2)-5)`
On tape la traduction de l'algorithme avec `Xcas` :

```
zeroapprox(f,p):={
local a,b;
a:=0;
b:=f(a);
tantque b<0 faire
  a:=a+p;
  b:=f(a);
ftantque;
retourne a-p,a
}:;
```

On tape : `zeroapprox(f,0.1)`
On obtient : `0.4,0.5`
On tape : `zeroapprox(f,0.01)`
On obtient : `0.44,0.45`
On tape : `zeroapprox(f,0.001)`
On obtient : `0.445999999998,0.446999999998`
On tape : `zeroapprox(f,0.0001)`
On obtient : `0.446199999974,0.446299999974`
On remarquera que cet algorithme est valable pour toutes les fonctions continues f qui vérifient $f(0) < 0$ et $f(1) > 0$.
Remarque Ce programme est moins performant que le programme de dichotomie vu précédemment.

4.5 En 2012 France

Soit (u_n) la suite définie pour tout entier strictement positif par :

$$u_n = 1 + \frac{1}{2} + \frac{1}{3}... + \frac{1}{n} - \ln(n)$$

1. On considère l'algorithme suivant :

```
Entrée :          l'entier n
Variables :       j  est un entier
                  u est un réel
Initialisation :  Affecter à u la valeur 0
Traitement :      Pour j variant de 1 à n
                    Affecter à u la valeur u +1/j
                  fPour
Sortie :         Afficher u
```

   Donner la valeur exacte affichée par cet algorithme lorsque l'utilisateur entre la valeur $n = 3$.

2. Recopier et compléter l'algorithme précédent afin qu'il affiche la valeur de u_n lorsque l'utilisateur entre la valeur de n.

3. Voici les résultats fournis par l'algorithme modifié, arrondis à 10^{-3}.

n	6	7	8	9	10	100	1000	1500	2000
u_n	0.658	0.647	0.638	0.632	0.626	0.582	0.578	0.578	0.577

   À l'aide de ce tableau, formuler des conjectures sur le sens de variation de la suite (u_n) et son éventuelle convergence.

La solution et la traduction avec `Xcas`
Le but de l'exercice est de trouver une approximation de la constante d'Euler :
$\gamma = \lim_{n\to+\infty}(1 + \frac{1}{2} + \frac{1}{3}... + \frac{1}{n} - \ln(n))$.
On montre dans un premier temps que cette limite existe car :
u_n est décroissante en effet $u_{n+1} - u_n = \dfrac{1}{n+1} + \ln(\dfrac{n}{n+1}) < 0$ et
l'étude de $f(x) = \dfrac{1}{x+1} + \ln(\dfrac{x}{x+1})$ montre que f est négative sur $[1; +\infty[$.
de plus u_n est minorée par 0. En effet pour $p \in \mathbb{N}^*$,on a :
$\displaystyle\int_{x=p}^{x=p+1} \frac{dx}{x} = \ln(p+1) - \ln(p) < \frac{1}{p}$
En sommant cette inégalité pour $p = 1..n$ on a :
$\ln(n+1) < 1 + \dfrac{1}{2} + \dfrac{1}{3}... + \dfrac{1}{n}$ donc
$0 < \ln(1 + 1/n) < 1 + \dfrac{1}{2} + \dfrac{1}{3}... + \dfrac{1}{n-1} + \dfrac{1}{n} - \ln(n)$
Ainsi u_n est décroissante et minorée par 0 elle a donc une limite positive appelée "constante d'Euler".
L'algorithme calcule $1 + \frac{1}{2} + \frac{1}{3}... + \frac{1}{n}$
Pour $n = 3$
Initialisation : u=0
Etape 1 : j=1, u=1
Etape 2 : j=2, u=3/2

Etape 3 : j=3, u=11/6
Resultat : 11/6 ou 1.83333333333
On modifie l'algorithme pour qu'il affiche la suite $u_n = 1 + \frac{1}{2} + ... + \frac{1}{n} - \ln(n)$, pour cela, il suffit de modifier la sortie :

```
Entrée :          l'entier n
Variables :       j et n sont des entiers naturels
                  u est un réel
Initialisation :  Affecter à u la valeur 0
Traitement :      Pour j variant de 1 à n
                    Affecter à u la valeur u +1/j
                  fPour
Sortie :         Afficher u-ln(n)
```

La traduction avec `Xcas`
On retourne une valeur numérique grâce à `evalf(u)` qui transforme le rationnel u en un nombre décimal.

```
csteuler(n):={
local j, u;
u:=0;
pour j de 1 jusque n faire
  u:=u+1/j;
fpour;
retourne evalf(u)-ln(n);
}:;
```

On tape : `csteuler(10)`
On obtient : `0.626383160974`
On tape : `csteuler(100)`
On obtient : `0.582207331651`
On tape : `csteuler(1000)`
On obtient : `0.577715581568`
On tape : `csteuler(2000)`
On obtient : `0.577465644068`
On tape : `csteuler(20000)`
On obtient : `0.577240664693`
Cela montre que la constante d'Euler est proche de 0.577240664693.
On tape car `Xcas` connait cette constante :
`evalf(euler_gamma)`
On obtient :
`0.5772156649018`
Remarque Les deux variantes de `csteuler` écrites ci-dessous font à chaque étape un calcul numérique car on a mis `1./j` au lieu de `1/j`. `csteuler0` calcule la somme des $\frac{1}{k}$ pour k allant de 1 à n, alors que `csteuler1` calcule la somme des $\frac{1}{k}$ pour k allant de n à 1

```
csteuler0(n):={
local j, u;
u:=0;
```

```
pour j de 1 jusque n faire
  u:=u+1./j;
fpour;
retourne u-ln(n);
}
:;
csteuler1(n):={
local j, u;
u:=0;
pour j de n jusque 1 pas -1 faire
  u:=u+1./j;
fpour;
retourne u-ln(n);
}:;
```

On tape : `csteuler0(2000)`
On obtient : `0.577465643831`
On tape : `csteuler1(2000)`
On obtient : `0.577465644032`
On tape : `csteuler(2000)`
On obtient : `0.577465644068`
Il faut comprendre la différence des résultats obtenus entre les fonctions `csteuler0`, `csteuler1` et `csteuler` :
`csteuler1` donne un résultat meilleur que `csteuler0` car il commence par ajouter des petits nombres donc la somme conserve plus de décimales.
Le résultat de `csteuler` est encore meilleur car il ne fait l'approximation décimale qu'à la fin du programme.

4.6 D'autres algorithmes sur ce modèle

4.6.1 Calcul de 1+2+..+n^2

Soit la suite $u_n = 1 + 4 + .. + n^2$.
Écrire un algorithme qui calcule u_n en fonction de n.
Puis calculer successivement u_n/n et $u_n/(n(n+1))$ pour $n = 1..10$
Montrer que $u_n = \dfrac{n(n+1)(2n+1)}{6}$
La solution
On écrit l'algorithme :

```
Entrée :          l'entier n
Variables :       j est un entier
                  S est un réel
Initialisation :  Affecter à S la valeur 0
Traitement :      Pour j variant de 1 à n
                    Affecter à S la valeur S+j^2
                  fPour
Sortie :         Afficher S
```

Avec `Xcas` :

```
Scarre(n):={
local j,S;
S:=0;
pour j de 1 jusque n faire
  S:=S+j^2;
fpour
retourne S;
}:;
```

On tape : `Scarre(p)$(p=0..10)`
On obtient :
`0,1,5,14,30,55,91,140,204,285,385`
On tape : `(Scarre(p)/p)$(p=0..10)`
On obtient :
`1,5/2,14/3,15/2,11,91/6,20,51/2,95/3,77/2`
On tape : `(Scarre(p)/(p*(p+1)))$(p=1..10)`
On obtient :
`1/2,5/6,7/6,3/2,11/6,13/6,5/2,17/6,19/6,7/2`
On tape : `((2p+1)/6)$(p=1..10)` et on obtient le résultat précédent.
Il reste donc à démontrer par récurrence que :
$u_n = 1 + 2^2 + .. + n^2 = \dfrac{n(n+1)(2n+1)}{6}$

4.6.2 Calcul de 1+1/4+..+1/n^2

Écrire un algorithme qui calcule $u_n = 1 + 1/4 + .. + 1/n^2$ en fonction de n.
La solution : on écrit l'algorithme :

```
Entrée :          l'entier n>=1
Variables :       j un entier, S un réel
Initialisation :  Affecter à S la valeur 0
Traitement :      Pour j variant de 1 à n
                    Affecter à S la valeur S+1/j^2
                  fPour
Sortie :         Afficher S
```

Avec `Xcas`

```
Scarre(n):={
local j,S;
S:=0;
pour j de 1 jusque n faire
  S:=S+1/j^2;
fpour
retourne S;
}:;
```

On tape : `Sinvcarre(p)$(p=1..9)`
On obtient : `1,5/4,49/36,205/144,5269/3600,5369/3600,`
`266681/176400,1077749/705600,9778141/6350400`
On tape : `evalf(Sinvcarre(p))$(p=0..9)`

On obtient : `1.0,1.25,1.36111111111,1.42361111111,1.46361111111,`
`1.49138888889,1.51179705215,1.52742205215,1.53976773117,`
On tape (on admet que u_n tend vers $\pi^2/6$ quand n tend vers $+\infty$) :
`sqrt(6.*Sinvcarre(1000))`
On obtient : `3.14063805621`

4.6.3 Calcul des termes de la suite de Fibonacci

La suite de Fibonacci u_n est définie par :
$u_0 = 1,\ u_1 = 1,\ u_{n+2} = u_{n+1} + u_n$ pour $n \in \mathbb{N}$
Écrire un algorithme qui calcule u_n en fonction de n.
La solution : on écrit l'algorithme :

```
Entrée :          l'entier n.
Variables :       j,a,b,c sont des entiers
Initialisation :  Affecter à a la valeur 1
                  Affecter à b la valeur 1
Traitement :      Pour j variant de 2 à n
                    Affecter à c la valeur a+b
                    Affecter à a la valeur b
                    Affecter à b la valeur c
                  fPour
Sortie :          Afficher b
```

Avec `Xcas`

```
fibo(n):={
local j,a,b,c;
a:=1;
b:=1;
pour j de 2 jusque n faire
  c:=a+b;
  a:=b;
  b:=c;
fpour;
retourne b;
}:;
```

On tape :
`fibo(p)$(p=0..10)`
On obtient les 11 premiers termes de la suite de Fibonacci :
`1,1,2,3,5,8,13,21,34,55,89`
On tape :
`fibo(101)/fibo(100)`
On obtient :
`927372692193078999176/573147844013817084101`
On tape :
`evalf(fibo(101)/fibo(100)),(1+sqrt(5))/2.`
On obtient :
`1.61803398875, 1.61803398875`

Il reste à montrer que $v_n = \dfrac{u_{n+1}}{u_n}$ tend vers à $\dfrac{1+\sqrt{5}}{2}$ qui est le nombre d'or.

Prolongement

On considère une une tour faite avec des briques de Lego rouges et vertes.
Combien peut-on construire de tours de hauteur n si on impose que l'on ne met pas 2 briques vertes à la suite ?
On peut alors demander aux élèves de faire toutes les tours de n briques pour $n = 2, 3, 4$ et de regarder comment on forme toutes les tours de 4 briques à partir des des tours de 3 briques et de 2 briques. Une tour de hauteur 4 se termine :
- soit par une brique rouge,
- soit par une brique verte.

Si on enlève la brique du haut :
- si elle est rouge on obtient une tour de hauteur 3 se terminant soit par une brique rouge, soit par une brique verte.
- si elle est verte, on obtient une tour de hauteur 3 se terminant par une brique rouge et donc en enlevant encore cette brique rouge, on obtient une tour de hauteur 2 se terminant soit par une brique rouge, soit par une brique verte.

On voit sur le dessin qu'une tour de hauteur 4 est composée :
- soit d'une tour de hauteur 3 plus une brique rouge,
- soit d'une tour de hauteur 2 plus une brique rouge et plus une brique verte.

De façon générale si `nbtour(n)` désigne le nombre de tours de hauteur `n`, on a :
`nbtour(n)=nbtour(n-1)+nbtour(n-2)` avec :
`nbtour(1)=2` et `nbtour(2)=3`
Donc `nbtour(3)=2+3=5` et `nbtour(4)=3+5=8`.

Activité de programmation

Faire un programme `brique(a,c)` qui dessine une brique de Lego de couleur `c` au point d'affixe `a`, Faire un programme `tour3(a,c0,c1,c2)` un tour de hauteur 3 avec des briques de couleur `c0,c1,c2`,
Faire un programme `tourn(a,C)` qui dessine une tour de hauteur `n` au point d'affixe `a` avec des briques dont la couleur est donnée par la liste `C` de longueur `n`.
Faire un programme `nbtour(n)` qui renvoie le nombre de tours de hauteur `n`,
Faire un programme `couleurtour(n)` qui renvoie une matrice qui a comme ligne les couleurs des tours de hauteur `n`,
Faire un programme `tours(a,n)` qui dessine les tours de hauteur `n` à partir du point d'affixe `a`.
On tape :

```
brique(a,c):={
[affichage(carre(a,a+1),c+rempli),carre(a,a+1)]
}:;
tourn(a,C):={
local L,j,n;
L:=NULL;
n:=size(C);
pour j de 0 jusque n-1 faire
  L:=L,brique(a+i*j,C[j]);
fpour;
return L;
}:;
```

```
nbtour(n):={
local a,b,c,k;
si n<0 alors retourne NULL; fsi;
si n==0 alors retourne 1;fsi;
si n==1 alors retourne 2; fsi;
a:=1;
b:=2;
pour k de 2 jusque n faire
    c:=a+b;
    a:=b;
    b:=c;
fpour;
retourne c;
}:;

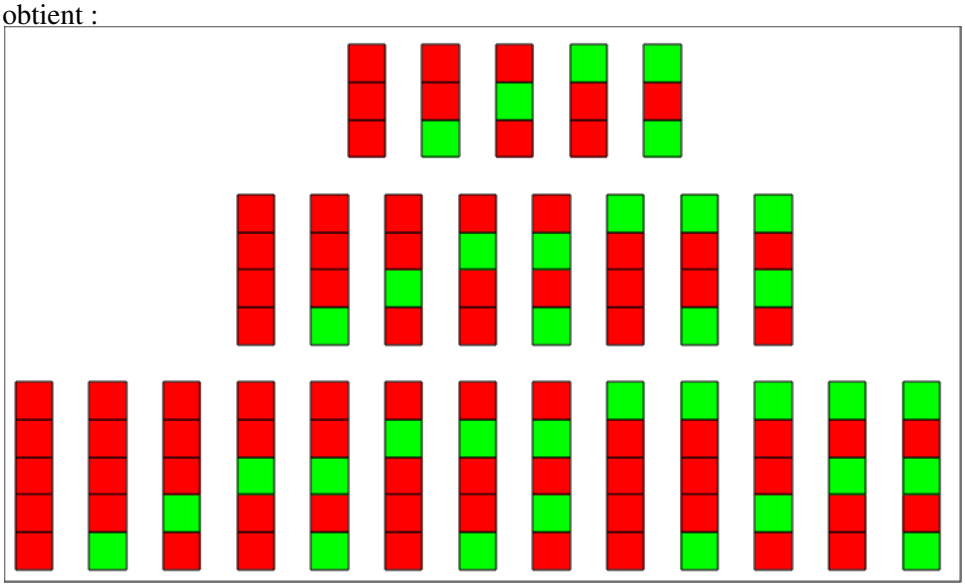
couleurtour(n):={
local A,L,C,j,k,s0,s1,N;
L:=[[1],[2]],[[1,1],[2,1],[1,2]];
j:=3 ;
N:=nbtour(n);
tantque j<=n faire
   s0:=size(L[0])-1;
   s1:=size(L[1])-1;
   A:=L[0];
   L[0]:=L[1];
   pour k de 0 jusque s1 faire
     L[1,k]:=append(L[1,k],1);
   fpour;
   pour k de 0 jusque s0 faire
     A[k]:=concat(A[k],[1,2]);
   fpour;
   j:=j+1;
   L[1]:=concat(L[1],A);
 ftantque;
 return L[1];
}:;

tours(a,n):={
 local L,k,R,N;
 R:=NULL;
 L:=couleurtour(n);
 N:=nbtour(n)-1;
 pour k de 0 jusque N faire
   R:=R,tourn(a,L[k]);
   a:=a+2;
 fpour;
 return R;
 }:;
```

On ouvre un écran de géométrie et on tape :

```
tours(-3+5i,3);
tours(-6,4);
tours(-12-6i,5);
```

On obtient :



On voit bien comment on a constitué les tours de hauteur 5 :
On rajoute une brique rouge aux tours de hauteur 4 et on rajoute une brique rouge et une brique verte aux tours de hauteur 3.

Chapitre 5

Des programmes tres simples sur les chaînes de caractères

5.1 Compter un nombre d'occurences

5.1.1 Nombre d'occurences d'une lettre

On parcourt la chaîne S et on augmente le compteur n de 1 lorsqu'il y a égalité avec le caractère c.

```
Occurc(c,S):={
  local n,d,j;
  d:=dim(S)-1;
  n:=0;
  pour j de 0 jusque d faire
    si c==S[j]  alors n:=n+1 fsi;
  fpour;
  retourne n;
}:;
```

On tape :
`Occurc("e","occurrences")`
On obtient : `2`

5.1.2 Nombre d'occurences d'une sous-chaîne

Il faut comparer à chaque étape la sous-chaîne `ch` avec un morceau de la chaîne `S` qui a même longueur.
On va utiliser `mid(S,j,k)` qui renvoie la sous chaîne de `S` de longueur `k` qui commence à l'indice `j` ou `mid(S,j)` qui renvoie la sous-chaîne fin de `S` commençant à l'indice `j`.
Remarque À la place de `mid(S,j,k)` on peut aussi utiliser `S[j..j+k-1]` (on met les indices de début et de fin de la sous-chaîne) et à la place de `mid(S,j)` on peut aussi utiliser `S[j..dim(S)-1]`.
On considère que dans "aaa" on voit une seule sous-chaîne "aa".

```
Occurch(ch,S):={
```

```
  local n,d,j,k;
  d:=dim(S)-1;
  k:=dim(ch);
  n:=0;
  j:=0;
  tantque j<= d-k+1 faire
    si ch==mid(S,j,k)  alors
      n:=n+1;
      j:=j+k;
      sinon
        j:=j+1
    fsi;
  fpour;
  retourne n;
}:;
```

On tape :
`Occurch("e","occurrences")`
On obtient : `2`
On tape :
`Occurch("az","aaazaaazaaaz")`
On obtient : `3`
On tape :
`Occurch("aa","aaazaaazaaaz")`
On obtient : `3`

5.2 Supprimer une lettre et sous-chaîne

5.2.1 Supprimer une lettre

On parcourt la chaîne S lorsqu'il y a égalité avec le caractère c il faudra supprimer ce caractère en faisant une concaténation entre ce qu'il y a avant c (si c est d'indice j c'est `mid(S,0,j)` car ce qu'il y a avant c est de longueur j) et ce qu'il y a après c (`mid(S,j+1)`). On met alors à jour la longueur de S.

```
Supprimec(c,S):={
  local d,j;
  d:=dim(S)-1;
  j:=0;
  tantque j<=d faire
    si c==S[j]  alors
      S:=mid(S,0,j)+mid(S,j+1);
      d:=d-1;
    sinon
      j:=j+1
    fsi;
  ftantque;
  retourne S;
}:;
```

On tape :
`Supprimec("e","occurrences")`
On obtient : `"occurrncs"`

5.2.2 Supprimer une sous-chaîne

On parcourt la chaîne s lorsqu'il y a égalité avec la sous-chaîne ch il faudra supprimer cette sous-chaîne en faisant une concaténation entre ce qu'il y a avant ch et ce qu'il y a après ch. On met alors à jour la longueur de S.

```
Supprimech(ch,S):={
  local k,d,j;
  d:=dim(S)-1;
  k:=dim(ch);
  j:=0;
  tantque j<=d faire
    si ch==mid(S,j,k)  alors
      S:=mid(S,0,j)+mid(S,j+k);
      d:=d-k;
    sinon
      j:=j+1
    fsi;
  ftantque;
  retourne S;
}:;
```

On tape :
`Supprimech("e","occurrences")`
On obtient : `"occurrncs"`
On tape :
`Supprimech("az","azerazerazaz")`
On obtient : `"erer"`
On tape :
`Supprimesch("aa","aaazaaazaaaz")`
On obtient : `"azazaz"`

5.3 Remplacer une lettre ou une sous-chaîne par une autre chaîne

5.3.1 Remplacer une lettre par une autre lettre

Pour remplacer le caractére a par b dans S, on parcourt S et quand on trouve le caractére a on change ce caractére.

```
Remplaceab(a,b,S):={
local d,j;
  d:=dim(S)-1;
  j:=0;
  tantque j<=d faire
```

```
    si a==S[j]  alors
      S:=mid(S,0,j)+b+mid(S,j+1);
    sinon
      j:=j+1
    fsi;
  ftantque;
  retourne S;
}:;
```

On tape :
`Remplaceab("a","e","azerazerazaz")`
On obtient : `"ezerezerezez"`

5.3.2 Remplacer une sous-chaîne par une autre

Pour remplacer la sous-chaîne cha par la sous-chaîne chb dans S, on parcourt S et quand on trouve la sous-chaîne cha. on fait une concaténation entre ce qu'il y a avant cha, la sous-chaîne chb et ce qu'il y a après cha. On met alors à jour la longueur de S.

```
Remplacechab(cha,chb,S):={
local ka,kb,d,j;
  d:=dim(S)-1;
  ka:=dim(cha);
  kb:=dim(chb);
  j:=0;
  tantque j<=d faire
    si cha==mid(S,j,ka)  alors
      S:=mid(S,0,j)+chb+mid(S,j+ka);
      d:=d-ka+kb;
    sinon
      j:=j+1
    fsi;
  ftantque;
  retourne S;
}:;
```

On tape :
`Remplacechab("a","e","azerazerazaz")`
On obtient : `"ezerezerezez"`
On tape :
`Remplacechab("az","bcd","azerazerazaz")`
On obtient : `"bcderbcderbcdbcd"`

Chapitre 6

Des programmes tres simples pour les Mathématiques

6.1 Définir le minimum

6.1.1 Minimum de 2 nombres

Pour trouver le minimum de a et b et on compare a et b. Le minimum vaut a si $a <= b$ et sinon il vaut b.
On remarquera que puisque l'instruction `retourne` fait sortir du programme, on peut écrire le programme sans utiliser de `sinon`.

```
Mini(a,b):={
  si a<=b alors retourne a;fsi;
  retourne b;
};;
```

On tape :
`Mini(23,4*6)`
On obtient : `23`
On tape :
`Mini(1.5,sqrt(2))`
On obtient : `sqrt(2)`

6.1.2 Minimum de 3 nombres

On utilise le fait que :
Min(a, b, c)=Min(Min(a, b), c)
et on utilise le programme précédent.

```
Mini3(a,b,c):={
  local m;
  m:=Mini(a,b);
  m:=Mini(m,c);
  retourne m;
};;
```

On tape :
`Mini3(3^3,4*6,5^2)`
On obtient : `24`
On tape :
`Mini3(1.5,sqrt(2),1.41)`
On obtient : `1.41`

6.1.3 Minimum d'une liste de nombres

Pour trouver le minimum de la liste L, on parcourt la liste L en utilisant une variable m qui sera le minimum de ce que l'on vient de parcourir et qui sera mis à jour au fur et à mesure que l'on parcourt la liste L.
On renvoie m et l'indice jm qu'il a dans L.

```
MiniL(L):={
  local m,j,d,a,jm;
  d:=dim(L)-1;
  m:=L[0];
  jm:=0;
  pour j de 1 jusque d faire
    a:=L[j];
    si a<m alors
      m:=a;
      jm:=j;
    fsi;
  fpour;
  retourne m,jm;
}:;
```

On tape :
`MiniL([12,32,3,23,5,2,45])`
On obtient : `2`

6.2 Trier

6.2.1 Ordonner 2 nombres par ordre croissant

```
Trier(a,b):={
  si a<=b alors retourne a,b;fsi;
  retourne b,a;
}:;
```

On tape :
`Trier(125/3,163/4)`
On obtient : `163/4,125/3`

6.2.2 Ordonner 3 nombres par ordre croissant

```
Trier3(a,b,c):={
  si a>b alors a,b:=b,a;fsi;
  si c<=a alors retourne c,a,b;fsi;
  si c<=b alors retourne a,c,b; fsi;
  retourne a,b,c;
}:;
```

On tape :
`Trier3(12,1,23)`
On obtient : `1,12,23`

6.2.3 Ordonner une séquence de nombres par ordre croissant

Tri par recherche du minimum

On utilise une liste $Lrep$ pour mettre la liste triée. On parcourt la liste L pour chercher l'indice jm du plus petit élément m, puis on le met dans la liste $Lrep$ et et on enlève cet élément de L on enlève cet élément de L Puis on refait la même chose avec la liste privée de son premier élément etc..
On va utiliser `mid(L,j,k)` qui renvoie la sous liste de `L` de longueur `k` qui commence à l'indice `j` ou `mid(S,j)` qui renvoie la liste fin de `L` commençant à l'indice `j` .
Remarque À la place de `mid(L,j,k)` on peut aussi utiliser `L[j..j+k-1]` (on met les indices de début et de fin de la sous liste) et à la place de `mid(L,j)` on peut aussi utiliser `L[j..dim(L)-1]`.

```
TrierLr(L):={
  local j,k,m,jm,d,Lrep;
  d:=dim(L)-1;
  Lrep:=[];
  pour j de 0 jusque d faire
    m,jm:=MiniL(L);
    Lrep:=append(Lrep,m);
    L:=concat(mid(L,0,jm),mid(L,jm+1));
  fpour
 retourne Lrep;
}:;
```

On utilise la même liste L pour mettre la liste triée. On parcourt la liste L pour chercher l'indice jm du plus petit élément m, puis on l'échange avec le premier élémment de L. Puis on refait la même chose avec la liste privée de son premier élément etc...C'est le tri par recherche du minimum.

```
TrierL(L):={
  local j,k,m,jm,d;
  d:=dim(L)-1;
  pour k de 0 jusque d-1 faire
    jm:=k;
```

```
    m:=L[k];
    pour j de k+1 jusque d faire
      si m>L[j] alors m:=L[j];jm:=j; fsi;
    fpour;
  L[jm]:=L[k];
  L[k]:=m;
  fpour
 retourne L;
}:;
```

On tape :
`TrierLr([23,12,1,14,21,4,45,11])`
Ou on tape :
`TrierL([23,12,1,14,21,4,45,11])`
On obtient : `[1,4,11,12,14,21,23,45]`

Tri par insertion

On utilise une liste la même lisre L pour mettre la liste triée. À chaque étape on insère l'élément suivant `a=L[k]` dans le début de la liste qui est déjà triée. Quand on a trouvé où» il fallait insérer a par exemple entre `L[j-1]` et `L[j]` il faut lui faire de la place en décalant d'un cran vers la droite les éléments de `L` de j jusque k. C'est le tri par insertion.

```
TrieL(L):={
  local j,k,d,a,p;
  d:=dim(L)-1;
  pour k de 1 jusque d faire
    j:=0;
    a:=L[k];
    tantque a>L[j] faire j:=j+1; ftantque
    si j<k alors
      // on d\'ecale d'un cran vers la droite
      pour p de k jusque j+1 pas -1 faire
        L[p]:=L[p-1]
      fpour;
      L[j]:=a;
    fsi
  fpour
  retourne L;
}:;
```

On tape :
`TrieL([23,12,1,14,21,4,45,11])`
On obtient : `[1,4,11,12,14,21,23,45]`

Tri par fusion

À chaque étape on partage la liste L en deux listes L_1 er L_2 de même longueur. On trie ces 2 listes grâce à 2 appels récursifs, puis on les fusionnne. Pour cela on

écrit la fonction `fusion` qui fusionne 2 listes triées : à chaque étape on compare un élément de L_1 avec un élément de L_2, on met le plus petit des 2 dans la liste réponse et on avance l'indice correspondant au plus petit d'un cran et on recommence...
On tape :

```
fusion(L1,L2):={
local d1,d2,j1,j2,L;
 d1:=dim(L1)-1;
 d2:=dim(L2)-1;
 L:=[];
 j1:=0;
 j2:=0;
tantque j1<=d1 et j2<=d2 faire
 si L1[j1]<L2[j2] alors L:=append(L,L1[j1]); j1:=j1+1;
   sinon  L:=append(L,L2[j2]); j2:=j2+1;
 fsi;
ftantque;
si j1<=d1 alors L:=concat(L,mid(L1,j1);
 sinon   L:=concat(L,mid(L2,j2);
fsi;
retourne L;
}:;

Trifusion(L):={
  local d,d1,L1,L2;
  d:=dim(L);
  si d==1 ou d==0 alors retourne L;fsi;
  d1:=iquo(d,2);
  L1:=mid(L,0,d1);
  L2:=mid(L,d1);
  L1:=Trifusion(L1);
  L2:=Trifusion(L2);
  retourne fusion(L1,L2);
}:;
```

On peut améliorer le programme précédent en utilisant une liste que l'on modifie en place (avec l'opérateur =<) afin de ne pas recopier la liste `L` à chaque affectation par `:=`. Attention, cela nécessite de faire une copie de la liste vide initiale par `copy` sinon c'est la liste du programme lui-même qui sera modifiée et ne sera donc plus initialisée à une liste vide.

```
fusionenplace(L1,L2):={
local d1,d2,j1,j2,k,j,L;
 d1:=dim(L1)-1;
 d2:=dim(L2)-1;
 L:=copy([]);
 j1:=0;
 j2:=0;
 k:=0;
```

```
 tantque j1<=d1 et j2<=d2 faire
   si L1[j1]<L2[j2] alors L[k]=<L1[j1]; j1:=j1+1;
   sinon  L[k]=<L2[j2]; j2:=j2+1;
   fsi;
   k:=k+1;
 ftantque;
 pour j de j1 jusque d1 faire
   L[k]=<L1[j];
   k:=k+1;
 fpour;
 pour j de j2 jusque d2 faire
   L[k]=<L2[j];
   k:=k+1;
 fpour;
 retourne L;
}
:;
Trifusionenplace(L):={
  local L1,L2,d1,d;
  d:=dim(L);
  si d==1 ou d==0 alors retourne L;fsi;
  d1:=iquo(d,2);
  L1:=mid(L,0,d1);
  L2:=mid(L,d1);
  L1:=Trifusionenplace(L1);
  L2:=Trifusionenplace(L2);
  retourne fusionenplace(L1,L2);
}
:;
```

6.3 Définir une fonction par morceaux

On peut utiliser l'instruction `si...sinon` ou l'instruction `ifte` ou mieux l'instruction `when` (ou avec la version infixée de `when` qui est `?`).
Soit la fonction f définie par :

$$f(x) = \begin{cases} -1 & \text{si } x < 0 \\ 0 & \text{si } x = 0 \\ +1 & \text{si } x > 0 \end{cases}$$

On tape

```
 f(x):={
  si x<0 alors retourne -1;fsi;
  si x==0  alors retourne 0 fsi;
  si x>0 alors retourne 1;fsi;
}:;
```

mais on peut aussi écrire la même chose avec l'instruction `ifte` :

```
f(x):=ifte(x<0,1,ifte(x==0,0,1)
```

Mais il faut alors savoir que pour que `f(a)` soit valable il faut que `a` contienne une valeur.
Par contre si on tape :
`g(x):=when(x<0,1,when(x==0,0,1)`
ou
`g(x):=(x>0)? 1 : ((x==0)? 0 : -1)`
`g(a)` est valable même si `a` est symbolique i.e. ne contient pas de valeur.

6.4 Convertir

6.4.1 Des secondes en jours, heures, minutes et secondes

On se donne un nombre ns de secondes que l'on veut convertir en heures h, minutes mn et secondes s. On a :

$$ns = 3600h + 60mn + s = s + 60(mn + 60h)$$

On tape :

```
converth(ns):={
  local h,mn,s;
  s:=irem(ns,60);
  ns:=iquo(ns,60);
  mn:=irem(ns,60);
  h:=iquo(ns,60);
retourne h,mn,s;
}:;
```

On tape :
`converth(123456789)`
On obtient : `34293,33,9`
Si on veut aussi convertir en jours j, heures h, minutes mn et secondes s. On a :

$$ns = 24 * 3600j + 3600h + 60mn + s = s + 60(mn + 60(h + 24j))$$

Ou bien, on tape :

```
convertj(ns):={
  local j,h,mn,s;
  s:=irem(ns,60);
  ns:=iquo(ns,60);
  mn:=irem(ns,60);
  ns:=iquo(ns,60);
  h:=irem(ns,24);
  j:=iquo(ns,24);
retourne j,h,mn,s;
}:;
```

On tape :
`convertj(123456789)`
On obtient : `1428,21,33,9`

6.4.2 Des degrés en radians

Si la mesure d'un angle est rad en radians et deg en degrés, on a :

$$rad = deg * \pi/180$$

```
deg2rad(deg):=deg*pi/180;
```

On tape :
`deg2rad(48.2384062423)`
On obtient : `0.841919014843`

6.4.3 Des radians en degrés

Si la mesure d'un angle est rad en radians et deg en degrés, on a :

$$deg = rad * 180/\pi$$

```
rad2deg(rad):=rad*180/pi;
```

On tape :
`rad2deg(0.841919014843)`
On obtient : `48.2384062423`

6.5 Les fractions

6.5.1 Simplifier une fraction

On suppose que l'on donne la fraction F sous la forme de la liste $[N, D]$ de son numérateur et de son dénominateur. Pour la simplifier il suffit de diviser le numérateur et le dénominateur par leur pgcd.
On utilise ici la fonction `gcd` de `Xcas` pour le calcul du pgcd. On tape :

```
Simplifie(F):={
 local pgcd,N,D;
 N:=F[0];
 D:=F[1];
 pgcd:=gcd(N,D);
 retourne [N/pgcd,D/pgcd];
}:;
```

On tape :
`Simplifie([5544,55]);`
On obtient : `[504,5]`

6.5.2 Additionner 2 fractions

On commence par simplifier les 2 fractions, puis on cherche leur dénominateur commun qui est le ppcm de leur dénominateurs, On réduit ces fractions à ce dénominateur commun et on ajoute les numérateurs.
On suppose que l'on donne les fraction $F1$ et $F2$ sous la forme de la liste $[N, D]$.
Puis on simplifie le résultat.
On utilise ici la fonction `lcm` de `Xcas` pour le calcul du ppcm.

```
Ajoute(F1,F2):={
 local ppcm,N1,D1,N2,D2,N,D;
 F1:=Simplifie(F1);
 F2:=Simplifie(F2);
 N1:=F1[0];
 D1:=F1[1];
 N2:=F2[0];
 D2:=F2[1];
 D:=lcm(D1,D2);
 N1:=N1*D/D1;
 N2:=N2*D/D2;
 retourne Simplifie([N1+N2,D]);
}:;
```

On tape :
`Ajoute([1234,22],[5549,55])`
On obtient : `[8634,55]`

6.5.3 Multiplier 2 fractions

On commence par simplifier les 2 fractions, puis on multiplie les numérateurs entre eux et les dénominateurs entre eux. Puis on simplifie le résultat.

```
Multiple(F1,F2):={
 local N1,D1,N2,D2;
 F1:=Simplifie(F1);
 F2:=Simplifie(F2);
 N1:=F1[0];
 D1:=F1[1];
 N2:=F2[0];
 D2:=F2[1];
 retourne Simplifie([N1*N2,D1*D2]);
}:;
```

On tape :
`Multiple([1234,22],[5549,55])`
On obtient : `[3423733,605]`

Chapitre 7

Des programmes tres simples pour les Statistiques

7.1 Simulation du lancer d'une pièce

On veut tirer au hasard 0 ou 1 (par exemple 0=pile et 1=face).
On utilise la fonction `rand(n)` qui renvoie un entier aléatoire uniformément distribué dans `0.. n-1`.
On tape :

```
tirage_piece():=rand(2);
```

7.2 Simulation d'un dé

On veut tirer au hasard un nombre entier dans [1.. 6].
On utilise la fonction `rand(n)` qui renvoie un entier aléatoire uniformément distribué dans `0.. n-1`.
On tape :

```
tirage_de():=1+rand(6);
```

7.3 Simulation d'une variable aléatoire

On tire au hasard un nombre entier n de [0,1,2,3].
On tire au hasard un nombre réel m de [0,1[.
On veut simuler une variable aléatoire X qui vaut la partie entière de $m * n$.
On utilise la fonction `rand(p,n)` qui renvoie un réel aléatoire uniformément distribué dans `[p.. n[`.
On tape :

```
X():={
local n;
n:=rand(4);
return floor(n*rand(0,1));
}
```

Quelle est la fonction de répartition de X ? X vaut :
0 si (($n = 0$ ou $n = 1$) et $m \in [0, 1[$) ou ($n = 2$ et $m \in [0, 0.5[$) ou ($n = 3$ et $m \in [0, 1/3[$)
1 si ($n = 2$ et $m \in [0.5, 1[$) ou ($n = 3$ et $m \in [1/3, 2/3[$)
2 si $n = 3$ et $m \in [2/3, 1[$
On a donc :
Proba($X = 0$)=1/4+1/4+1/4*1/2+1/4*1/3=17/24
Proba($X = 1$)=1/4*1/2+1/4*1/3=5/24
Proba($X = 2$)=1/4*1/3=1/12
On vérifie que l'on a : 17/24+5/24+1/12=1

7.4 Simulation d'une variable aléatoire

On tire au hasard un nombre entier n de [1,2,3].
On veut simuler une variable aléatoire Y qui vaut $\sum_{j=1}^{n} j$.
On tape :

```
Y():={
local n,j,x;
n:=1+rand(3);
x:=0;
pour j de 1 jusque n faire
x:=x+j;
fpour;
return x;
}:;
```

Quelle est la fonction de répartition de Y ? Y vaut :
1 si $n = 1$
3 si $n = 2$
6 si $n = 3$
On a donc :
Proba($X = 1$))=1/3
Proba($X = 3$))=1/3
Proba($X = 6$))=1/3
On aurait pu aussi simuler cette loi en tapant :

```
Z():={
local n;
n:=rand(3);
si n==0 alors return 1 fsi;
return 3*n;
}:;
```

7.5 Comment simplifier $\sqrt{a + \sqrt{b}}$ lorsque $(a, b) \in \mathbb{N}^2$

On veut écrire $\sqrt{a + \sqrt{b}}$ sous la forme $\sqrt{x/2} + \sqrt{y/2}$ bien sur quand cela est possible.

Si $\sqrt{x/2}+\sqrt{y/2}=\sqrt{a+\sqrt{b}}$ on a :
$(\sqrt{x/2}+\sqrt{y/2})^2=a+\sqrt{b}=(x+y)/2+\sqrt{xy}$
d'où $x+y=2a$ et $xy=b$
donc x et y sont solutions de l'équation $X^-2aX+b=0$
donc si a^2-b est un carré parfait i.e. $a^2-b=c^2$ avec c entier on a :
$x=a+c$ et $y=a-c$
Donc si a^2-b est un carré parfait on a si $a^2-b=c^2$:

$$\sqrt{a+\sqrt{b}}=\sqrt{\frac{a+c}{2}}+\sqrt{\frac{a-c}{2}}=\frac{\sqrt{2(a+c)}+\sqrt{2(a-c)}}{2}$$

On tape la fonction `simply` qui doit avoir `r=sqrt(a+sqrt(b))` comme paramètre où `b` est sans facteur carré :

```
simply(r):={
local f,a,b,c,d;
si sommet(r)=='^' alors
  f:=feuille(r);
  a,d:=feuille(f[0]);
  si type(a)==integer alors
  b:=feuille(d)[0];
  c:=sqrt(a^2-b);
  si (round(c))^2==a^2-b alors
    retourne sqrt((a+c)/2)+sqrt((a-c)/2)
  fsi;
  sinon
  d,a:=feuille(f[0]);
  b:=feuille(d)[0];
  c:=sqrt(a^2-b);
  si (round(c))^2==a^2-b alors
    retourne sqrt((a+c)/2)+sqrt((a-c)/2)
  fsi;
  fsi;
fsi;
retourne r;
}:;
```

On tape :
`simply(sqrt(9+sqrt(17)))`
ou
`simply(sqrt(sqrt(17)+9))`
On obtient :
`(sqrt(34))/2+(sqrt(2))/2` on tape la fonction `simplyf` qui doit avoir comme paramètre `r` de la forme `sqrt(a+s*sqrt(b))` ou `sqrt(a+sqrt(b))` :

```
simplyf(r):={
local f,a,b,c,d,s,sb;
si sommet(r)=='^' alors
  f:=feuille(r);
  a,d:=feuille(f[0]);
```

```
  si type(a)==integer alors
     f:=feuille(d);
    si f[1]==1/2 alors
      s:=1;
      b:=f[0];
    sinon
      s,sb:=f;
      si type(s)==integer alors
        b:=feuille(sb)[0];
      sinon
       sb,s:=f;
       b:=feuille(sb)[0];
      fsi;
    fsi;
    c:=sqrt(a^2-s^2*b);
    si (round(c))^2==a^2-s^2*b alors
      retourne sqrt((a+c)/2)+sqrt((a-c)/2);
    fsi;
  sinon
    d,a:=feuille(f[0]);
    f:=feuille(d);
    si f[1]==1/2 alors
      b:=f[0];
      s:=1;
    sinon
      s,sb:=f;
      si type(s)==integer alors
        b:=feuille(sb)[0];
      sinon
       sb,s:=f;
       b:=feuille(sb)[0];
      fsi;
    fsi;
    c:=sqrt(a^2-s^2*b);
    si (round(c))^2==a^2-s^2*b alors
      retourne sqrt((a+c)/2)+sqrt((a-c)/2);
    fsi;
  fsi;
fsi;
retourne r;
}:;
```

On tape (car `Xcas` remplace `sqrt(128)` par `8*sqrt(2)` :

```
simplyf(sqrt(18+sqrt(128)))
```

ou

```
simplyf(sqrt(sqrt(128)+18))
```

ou

```
simplyf(sqrt(8*sqrt(2)+18))
```

ou

`simplyf(sqrt(18+8*sqrt(2)))`
ou
`simplyf(sqrt(sqrt(2)*8+18))`
ou
`simplyf(sqrt(18+sqrt(2)*8))`
On obtient :
`4+sqrt(2)` On tape :
`simplyf(sqrt(194320499737857776523212040+19713979797994*sqrt(9716024986892888`
On obtient :
`sqrt(971602498689288882616060 31)+9856989898997`

Chapitre 8

Les programmes d'arithmétique

8.1 Quotient et reste de la division euclidienne

8.1.1 Les fonctions iquo, irem et smod de Xcas

Si a et b sont des entiers ou des entiers de Gauss :
`iquo(a,b)` renvoie le quotient q de la division euclidienne de a par b et
`irem(a,b)` renvoie le reste r de la division euclidienne de a par b.
q et r vérifient :
si a et b sont entiers $a = b * q + r$ avec $0 \leq r < b$
si a et b sont des entiers de Gauss $a = b * q + r$ avec $|r|^2) \leq \frac{|b|^2}{2}$.
Par exemple si $a = -2 + 6 * i$ et si $b = 1 + 3 * i$ on a :
$q = 2 + i$ et $r = -1 - i$
Si a et b sont des entiers relatifs `smod(a,b)` renvoie le reste symétrique rs de la division euclidienne de a par b.
q et rs vérifient :
$a = b * q + rs$ avec $-\frac{b}{2} < rs \leq \frac{b}{2}$
Exemples :

```
smod(7,4)=-1
smod(-10,4)=-2
smod(10,4)=2
```

Remarque `mod` (ou `%`) est une fonction infixée et désigne un élément de Z/nZ.
On a : `7 mod 4=-1%4` désigne un élément de $Z/4Z$ mais
`smod(7,4)=(7%4)%0=-1` désigne un entier.

8.1.2 Quotient et du reste sans utiliser iquo et irem

Soient a et b 2 entiers.
On cherche 2 entiers q et $r < a$ tels que $a = bq + r$.
On va utiliser la soustraction on doit avoir soustraire b à a jusqu'à ce que $(a - b - b.. - b) = r < b$. On tape :

```
Iquorem(a,b):={
local q,r;
r:=b;q:=0;
while (a>=b) {
```

```
a:=a-b
q:=q+1;
}
return (q,a);
}:;
```

8.1.3 Activité

le texte de l'exercice

1. Vérifier que :
$$\frac{13}{18} = \frac{1}{2} + \frac{1}{5} + \frac{1}{45}$$
2. On donne deux entiers a et b vérifiant : $0 < b < a$. On note q et r le quotient et le reste de la division euclidienne de a par b ($a = bq + r$ avec $0 \leq r < b$).
Démontrer que :
$$q > 0$$
$$\frac{1}{q+1} < \frac{b}{a} \leq \frac{1}{q}$$
3. On définit u et v par :
$$\frac{b}{a} - \frac{1}{q+1} = \frac{v}{u}$$
et
$$u = a(q+1)$$
Exprimer v en fonction de b et r.
Démontrer que :
$$v \leq b < a < u$$
Si $r = 0$, vérifier que :
$$\frac{b}{a} = \frac{1}{q}$$
4. Si r est différent de zéro, on pose : $a_1 = u$ et $b_1 = v$.
Puis, on recommence : on divise a_1 par b_1.
On trouve un quotient q_1 et un reste r_1.Si r_1 est nul, vérifier que :
$$\frac{b}{a} = \frac{1}{q+1} + \frac{1}{q_1}$$
Si r_1 n'est pas nul, on recommence.
Montrer qu'il existe une suite finie d'entiers $Q_0, Q_1, ...Q_n$ strictement croissante telle que :
$$\frac{b}{a} = \frac{1}{Q_0} + \frac{1}{Q_1} + ... + \frac{1}{Q_n}$$
5. Rédiger l'algorithme décrit ici et l'appliquer à la fraction :
$$\frac{151}{221}$$

L'algorithme

On suppose que la fraction $\frac{\texttt{NUM}}{\texttt{DENOM}} \in]0;1[$.

L'algorithme s'écrit en langage non fonctionnel :

```
DENOM →A
NUM→B
Quotient(A, B)→Q
Reste(A, B)→R
tantque R ≠0 faire
   Q+1→D
   Afficher D
   B-R→B
   A*D→A
   Quotient(A, B)→Q
   Reste(A, B)→R
ftantque
Afficher Q
```

Traduction Xcas

On écrit la fonction `decomp` qui va décomposer selon l'algorithme la fraction `frac`. Cette fonction va renvoyer la liste `lres` égale à $[\texttt{Q}_0, ..\texttt{Q}_\texttt{n}]$ avec $0 < \texttt{Q}_0 <, .. < \texttt{Q}_\texttt{n}$ et $\texttt{frac} = 1/\texttt{Q}_0 + .. + 1/\texttt{Q}_\texttt{n}$.
Attention `frac=b/a` et donc `fxnd(frac)=fxnd(b/a)=[b,a]`.

```
decomp(frac):={
local a,b,l,q,r,lres;
l:=fxnd(frac);
b:=l[0];
a:=l[1];
q:=iquo(a,b);
r:=irem(a,b);
lres:=[];
while (r!=0) {
lres:= concat(lres, q+1);
b:=b-r;
a:=a*(q+1);
q:=iquo(a,b);
r:=irem(a,b);
}
lres:=concat(lres,q);
return lres;
}
```

Application à $\frac{151}{221}$

On tape :

```
decomp(151/221)
```

On obtient :
`[2,6,61,5056,40895962,4181199228867648]`
On vérifie :
`1/2+1/6+1/61+1/5056+1/40895962+1/4181199228867648`
On obtient :
`151/221`
On peut écrire un programme pour faire le vérification :
`size(l)` est égal à la longueur de la liste `l`

```
verifie(l):={
local s,k,res;
s:=size(l);
res:=0;
for (k:=0;k<s;k++){
res:=res+1/l[k];
}
return res;
}
```

On tape :
`verifie([2,6,61,5056,40895962,4181199228867648])`
On obtient :
$\frac{151}{221}$

8.2 Calcul du PGCD par l'algorithme d'Euclide

Soient A et B deux entiers positifs dont on cherche le $PGCD$.
L'algorithme d'Euclide est basé sur la définition récursive du $PGCD$:

$$\begin{aligned} PGCD(A,0) &= A \\ PGCD(A,B) &= PGCD(B, A \mod B) \; si \; B \neq 0 \end{aligned}$$

où $A \mod B$ désigne le reste de la division euclidienne de A par B.
Voici la description de cet algorithme :
on effectue des divisions euclidiennes successives :

$$\begin{aligned} A = &\quad B \times Q_1 + R_1 \quad 0 \leq R_1 < B \\ B = &\quad R_1 \times Q_2 + R_2 \quad 0 \leq R_2 < R_1 \\ R_1 = &\quad R_2 \times Q_3 + R_3 \quad 0 \leq R_3 < R_2 \\ &..... \end{aligned}$$

Après un nombre fini d'étapes, il existe un entier n tel que : $R_n = 0$.
on a alors :
$PGCD(A,B) = PGCD(B,R_1) = ...$
$PGCD(R_{n-1},R_n) = PGCD(R_{n-1},0) = R_{n-1}$

8.2.1 Traduction algorithmique

-Version itérative
Si $B \neq 0$ on calcule $R = A \mod B$, puis avec B dans le rôle de A (en mettant B

dans A) et R dans le rôle de B (en mettant R dans B) on recommence jusqu'à ce que $B = 0$, le $PGCD$ est alors égal à A.

```
fonction PGCD(A,B)
local R

tantque B ≠ 0 faire

  A mod B=>R
  B=>A
  R=>B
ftantque
retourne A
ffonction
```

-Version récursive
On écrit simplement la définition récursive vue plus haut.

```
fonction PGCD(A,B)

Si B ≠ 0 alors

  retourne PGCD(B,A mod B)
  sinon
  retourne A
fsi
ffonction
```

8.2.2 Traduction Xcas

- Version itérative :

```
pgcd(a,b):={
  local r;
  while (b!=0){
   r:=irem(a,b);
   a:=b;
   b:=r;
  }
  return(a);
};
```

- Version récursive.

```
pgcdr(a,b):={
  if (b==0)
    return(a);
  else
    return(pgcdr(b,irem(a,b)));
};
```

8.2.3 Traduction MapleV

-Version itérative :

```
pgcd:=proc(x,y)
local a,b,r:
a:=x:
b:=y:
while (b>0) do
r:=irem(a,b):
a:=b:
b:=r:
od:
RETURN(a):
end:
```

-Version récursive :

```
pgcd:=proc(a,b)
if (b=0) then
RETURN(a)
 else
RETURN(pgcd(b,irem(a,b))):
fi:
end:
```

8.2.4 Traduction MuPAD

-Version itérative :

```
pgcd:=proc(a,b)
local r:
begin
while (b>0) do
r:=a mod b:
a:=b:
b:=r:
end_while:
return(a):
end_proc;
```

-Version récursive :

```
pgcd:=proc(a,b)
begin
if (b=0) then
 return(a)
 else
return(pgcd(b,a mod b)):
end_if:
end_proc;
```

8.2.5 Traduction TI89 92

-Version itérative

```
:pgcd(a,b)
:Func
:Local r

:While b ≠ 0

:mod(a,b)=>r
:b=>a
:r=>b
:EndWhile
:Return a
:EndFunc
```

-Version récursive

```
:pgcd(a,b)
:Func

:If b ≠ 0 Then

:Return  pgcd(b, mod(a,b))
:Else
:Return a
:EndIf
:EndFunc
```

8.2.6 Le pgcd soustractif (sans utiliser `iquo` et `irem`)

Si $a == b$ alors c'est facile le pgcd$(a, b) = a$. Si $a > b$ on cherche le reste de la division euclidienne de a par b en faisant :
$a := a - b$ jusqu'à ce que l'on ait $a < b$ et le reste r est alors égal à a puisque $a = r < b$,
sinon si $a < b$ on cherche le reste de b par a en faisant :
$b := b - a$ jusqu'à ce que l'on ait $b < a$ et le reste est alors égal à b Puis tant que $a! = b$ on refait la m%me chose .
On tape :

```
Pgcd(a,b):={
tantque a!=b faire
  si(a>b) alors
    a:=a-b;
  sinon
    b:=b-a;
  fsi;
ftantque;
retourne a;
}:;
```

8.2.7 Le pgcd dans $\mathbb{Z}[i]$

On rappelle que $\mathbb{Z}[i] = \{m + n * i,\ (m,n) \in \mathbb{Z}^2\}$.
Soient $a \in \mathbb{Z}[i]$ et $b \in \mathbb{Z}[i] - \{0\}$, alors on dit que le quotient q de a par b est l'affixe du (ou des) point(s), le plus proche pour le module, du point d'affixe a/b.

— Montrer que $|q - a/b|^2 \leq 1/2$. En déduire que $|a - bq|^2 \leq |b|^2/2$ et que l'algorithme d'Euclide se termine lorsqu'on prend q comme quotient euclidien.

— Écrire un programme qui calcule le pgcd de 2 nombres de $\mathbb{Z}[i]$. On normalisera le résultat (en multipliant le résultat par 1,-1,i ou -i) pour que le pgcd soit un nombre de partie rèelle strictement positive et de partie imaginaire positive ou nulle.

On tape :

```
quotient(a,b):={
local q1,q2,c;
c:=normal(a/b);
q1:=re(c);
q2:=im(c);
return round(q1)+i*round(q2);
}:;
pgcdzi(a,b):={
local q,r;
tantque b!=0 faire
  q:=quotient(a,b);
  r:=a-b*q;
  a:=b;
  b:=r;
ftantque;
//on normalise
si re(a)<0 et im(a)<=0 alors retourne -a;fsi;
si im(a)<0 alors retourne i*a;fsi;
si re(a)<=0 alors retourne -i*a;fsi;
retourne a;
}:;
```

On tape :
`pgcdzi(3+i,3-i)`
On obtient : `1+i`
On tape :
`pgcdzi(7+i,-6+17*i)`
On obtient : `3+4*i`

8.2.8 Le pgcd dans $\mathbb{Z}[i\sqrt{2}]$

On rappelle que $\mathbb{Z}[i\sqrt{2}] = \{m + i * n\sqrt{2},\ (m,n) \in \mathbb{Z}^2\}$.
Soient $a \in \mathbb{Z}[i\sqrt{2}]$ et $b \in \mathbb{Z}[i\sqrt{2}] - \{0\}$, alors on dit que le quotient q de a par b est l'affixe du (ou des) point(s), le plus proche pour le module, du point d'affixe a/b.
On tape :

```
quorest(a,b):={
local q1,q2,q,r,c;
c:=normal(a/b);
q1:=normal(round(re(c)));
q2:=normal(round(im(c)/sqrt(2)));
q:= q1+i*q2*sqrt(2);
r:=simplify(a-b*q);
return q,r;
}:;
pgcdzis2(a,b):={
local r;
tantque b!=0 faire
  r:=quorest(a,b)[1];
  a:=b;
  b:=r;
ftantque;
//on normalise
si re(a)<0  alors retourne -a;fsi;
retourne a;
}:;
```

On tape :
`pgcdzis2(3+i*sqrt(2),3-i*sqrt(2))`
On obtient : `1`
On tape :
`pgcdzis2(4+5*i*sqrt(2),-2+3*i*sqrt(2))`
On obtient : `2-(3*i)*sqrt(2)`

8.3 Identité de Bézout par l'algorithme d'Euclide

Dans ce paragraphe la fonction `Bezout(A,B)` renvoie la liste $\{U, V, PGCD(A, B)\}$ où U et V vérifient : $A \times U + B \times V = PGCD(A, B)$

8.3.1 Version itérative sans les listes

L'algorithme d'Euclide permet aussi de trouver un couple U et V vérifiant :
$A \times U + B \times V = PGCD(A, B)$
En effet, si on note A_0 etB_0 les valeurs de A et de B du début on a :

$$\begin{aligned} A &= A_0 \times U + B_0 \times V \text{ avec } U = 1 \, et \, V = 0 \\ B &= A_0 \times W + B_0 \times X \text{ avec } W = 0 \, et \, X = 1 \end{aligned}$$

Puis on fait évoluer $A,\ B,\ U,\ V,\ W,\ X$ de façon à ce que ces deux relations soient toujours vérifiées. Voici comment $A,\ B,\ U,\ V,\ W,\ X$ évoluent :
- on pose : $A = B \times Q + R \ \ 0 \leq R < B \ \ (R = A \bmod B \, et \, Q = E(A/B))$
- on écrit alors :

$$R = A - B \times Q = A_0 \times (U - W \times Q) + B_0 \times (V - X \times Q) = A_0 \times S + B_0 \times T$$
$$\text{avec } S = U - W \times Q \text{ et } T = V - X \times Q$$

Il reste alors à recommencer avec B dans le rôle de A (`B=>A W=>U X=>V`) et R dans le rôle de B (`R=>B S=>W T=>X`) d'où l'algorithme :

```
fonction Bezout(A,B)
local U,V,W,X,S,T,Q,R
1=>U 0=>V 0=>W 1=>X

tantque B ≠ 0 faire

A mod B=>R
E(A/B)=>Q
//R=A-B*Q
U-W*Q=>S
V-X*Q=>T
B=>A W=>U X=>V
R=>B S=>W T=>X
ftantque
retourne {U, V, A}
ffonction
```

8.3.2 Version itérative avec les listes

On peut simplifier l'écriture de l'algorithme ci-dessus en utilisant moins de variables : pour cela on utilise les listes LA, LB, LR pour mémoriser les triplets $\{U,\ V,\ A\}$, $\{W,\ X,\ B\}$ et $\{S,\ T,\ R\}$. Ceci est très commode car les logiciels de calcul savent ajouter des listes de même longueur (en ajoutant les éléments de même indice) et savent aussi multiplier une liste par un nombre (en multipliant chacun des éléments de la liste par ce nombre).

```
fonction Bezout(A,B)
local LA LB LR
{1, 0, A}=>LA
{0, 1, B}=>LB

tantque LB[3] ≠ 0 faire

LA-LB*E(LA[3]/LB[3])=>LR
LB=>LA
LR=>LB
ftantque
retourne LA
ffonction
```

8.3.3 Version récursive sans les listes

Si on utilise des variables globales pour `A, B, D, U, V, T`, on peut voir la fonction `Bezout` comme calculant à partir de `A B`, des valeurs qu'elle met dans `U, V, D (AU+BV=D)`, grâce à une variable locale `Q`.
On écrit donc une fonction sans paramètre : seule la variable `Q` doit être locale à la foncton alors que les autres variables `A, B ...` sont globales.

`Bezout` fabrique `U, V, D` vérifiant `A*U+B*V=D` à partir de `A` et `B`. Avant l'appel récursif (on préserve `E(A/B)=Q` et on met `A` et `B` à jour (nouvelles valeurs), après l'appel les variables `U, V, D` vérifient `A*U+B*V=D` (avec `A` et `B` les nouvelles valeurs), il suffit alors de revenir aux premières valeurs de `A` et `B` en écrivant : `B*U+(A-B*Q)*V=A*V+B*(U-V*Q)`
On écrit alors :

```
fonction Bezout
local Q
Si B != 0 faire
E(A/B)=>Q
A-B*Q=>R
B=>A
R=>B
Bezout
U-V*Q=>W
V=>U
W=>V
sinon
1=>U
0=>V
A=>D
fsi
ffonction
```

8.3.4 Version récursive avec les listes

On peut définir récursivement la fonction Bezout par :
$Bezout(A, 0) = \{1,\ 0,\ A\}$
Si $B \neq 0$ il faut définir $Bezout(A, B)$ en fonction de $Bezout(B, R)$ lorsque $R = A - B \times Q$ et $Q = E(A/B)$.
On a :

$$Bezout(B, R) = LT = \{W, X, pgcd(B, R)\}$$
$$\text{avec } W \times B + X \times R = pgcd(B, R)$$

Donc :

$$\begin{aligned} W \times B + X \times (A - B \times Q) &= pgcd(B, R) \text{ ou encore} \\ X \times A + (W - X \times Q) \times B &= pgcd(A, B). \end{aligned}$$

D'où si $B \neq 0$ et si $Bezout(B, R) = LT$ on a :
$Bezout(A, B) = \{LT[2],\ LT[1] - LT[2] \times Q,\ LT[3]\}$.

```
fonction Bezout(A,B)
local LT Q R

Si B ≠ 0 faire
```

```
E(A/B)=>Q
A-B*Q=>R
Bezout(B,R)=>LT
retourne {LT[2], LT[1]-LT[2]*Q, LT[3]}
sinon retourne {1, 0, A}
fsi
ffonction
```

8.3.5 Traduction Xcas

- Version itérative avec les listes

```
bezout(a,b):={
//renvoie [u,v,d] tels que a*u+b*v=pgcd(a,b) (fct iterative)
local la,lb,lr,q,lb2;
la:=[1,0,eval(a)];
lb:=[0,1,eval(b)];
lb2:=eval(b);
while (lb2 !=0){
q:=iquo(la[2],lb2);
lr:=la+(-q)*lb;
la:=lb;
lb:=lr;
lb2:=lb[2];
}
return(la);
};
```

- Version récursive avec les listes

```
bezoutr(a,b):={
//renvoie [u,v,d] tels que a*u+b*v=pgcd(a,b) (fct recursive)
local lb,q,r;
if (b!=0) {
q:=iquo(a,b);
r:=irem(a,b);
lb:=bezoutr(b,r);
return([lb[1],lb[0]+(-q)*lb[1],lb[2]]);
} else
return([1,0,a]);
};
```

8.4 Décomposition en facteurs premiers d'un entier

Dans cette section, on ne suppose pas connue une table de nombres premiers : on ne se sert donc pas du programme crible.

8.4.1 Les algorithmes et leurs traductions algorithmiques

— Premier algorithme
Soit N un entier.
On teste, pour tous les nombres D de 2 à N, la divisibilité de N par D.
Si D divise N, on cherche alors les diviseurs de N/D etc...N/D joue le rôle de N et on s'arrête quand $N = 1$
On met les diviseurs trouvés dans la liste `FACT`.

```
fonction factprem(N)
local D FACT
2 => D
{} => FACT
tantque N<= 1 faire
si N mod D = 0 alors
   concat(FACT,D) => FACT
    N/D => N
   sinon
    D+1 => D
 fsi
ftantque
retourne FACT
ffonction
```

— Première amélioration
On ne teste que les diviseurs D entre 2 et $E(\sqrt{N})$.
En effet si $N = D1 * D2$ alors on a :
soit $D1 \leq E(\sqrt{N})$, soit $D2 \leq E(\sqrt{N})$ car sinon on aurait :
$D1 * D2 \geq (E(\sqrt{N}) + 1)^2 > N$.

```
fonction factprem1(N)
local D FACT
2 => D
{} => FACT
tantque D*D!= N faire
si N mod D = 0 alors
    concat(FACT,D) => FACT
    N/D=> N
   sinon
    D+1 => D
 fsi
ftantque
concat(FACT,N) => FACT
retourne FACT
ffonction
```

Dans la liste `FACT`, on a les diviseurs premiers éventuellement plusieurs fois, par exemple :
`factprem1(12)={2,2,3}`.

— Deuxième amélioration
On cherche si 2 divise N, puis on teste les diviseurs impairs D entre 3 et $E(\sqrt{N})$.

Dans la liste FACT, on fait suivre chaque diviseur premier par son exposant, par exemple :

```
factprem2(12)={2,2,3,1}.
fonction  facprem2(N)
local K D FACT
{}=>FACT
0 => K
tantque N mod 2 == 0 faire
    K+1 => K
    N/2 => N
ftantque
si K !=0 alors
    concat(FACT,{2 K}) => FACT
fsi
3 =>D
tantque D*D<= N faire
    0 => K
    tantque N mod D = 0 faire
      K+1 => K
      N/D => N
    ftantque
    si K !=0 alors
      concat(FACT,{D K})=> FACT
    fsi
     D+2 => D
ftantque
si N != 1 alors
    concat(FACT,{N 1})=> FACT
fsi
retourne FACT
ffonction
```

— Troisième amélioration

On cherche si 2 et 3 divisent N, puis on teste les diviseurs D entre 5 et $E(\sqrt{N})$ de la forme $6 * k - 1$ ou $6 * k + 1$.

On remarque que si :

$D = 6 * k - 1$ on a $D + (4 * D \bmod 6) = 6 * k + 1$

et que si :

$D = 6 * k + 1$ on a $D + (4 * D \bmod 6) = 6 * (k + 1) - 1$

Dans la liste FACT, on fait suivre chaque diviseur par son exposant, par exemple :

```
factprem3(12)={2,2,3,1}.
fonction  factprem3(N)
local J,D,FACT
2=>D
{}=>FACT
tantque (D*D<=N) faire
 0=>J
 tantque (N mod D=0) faire
```

```
        N/D=>N
        J+1=>J
      ftantque
     si (J!= 0) alors concat(FACT,{D,J})=>FACT fsi
      si (D<4) alors
        2*D-1=>D
      sinon
        D+(4*D mod 6)=>D
      fsi
     ftantque
     si (N !=1) alors concat(FACT,{N,1})=>FACT fsi
     retourne(FACT)
     ffonction
```

8.4.2 Traduction Xcas

On traduit la troisième amélioration.

```
factprem(n):={
//decompose n en facteur premier dans la liste l de dimension s
local j,d,s,l;
d:=2;
s:=0;
l:=[];
while (d*d<=n) {
j:=0;
while (irem(n,d)==0){
n:=iquo(n,d);
j:=j+1;
}
if (j!=0) {
l:=concat(l,[d,j]);
s:=s+2;
}
if (d<4) {
d:=2*d-1;
}
else {
d:=d+irem(4*d,6);
}
}
if (n!=1) {
l:=concat(l,[n,1]);
s:=s+2;
}
return([l,s]);
};
```

8.5 Décomposition en facteurs premiers en utilisant le crible

Pour effectuer la décomposition en facteurs premiers de n, on utilise la table des nombres premiers fabriquée par le crible : on ne teste ainsi que des nombres premiers.
Si on peut écrire $N = A * D^J$ avec $PGCD(A, D) = 1$ et $J > 0$ alors D^J est un facteur de la décomposition de N.
On écrit tout d'abord la fonction `ddiv(N,D)` qui renvoie :
- soit la liste :
`[ N,[]]` si D n'est pas un diviseur de N,
- soit la liste :
`[A,[D,J]]` si $N = A * D^J$ avec $PGCD(A, D) = 1$ et $J > 0$.
D^J est alors un diviseur de N et $A = N/D^J$.

8.5.1 Traduction Algorithmique

```
fonction ddiv(N,D)
//ddiv renvoie [a,[d,j]] (n=a*d^j, pgcd(a,d)=1) si j!=0 sinon [n,[]]
local L,J
0=>J
tantque  (N mod D)=0) faire
N/D=>N
J+1=>J
ftantque
si (J=0) alors
{N,{}}=>L
sinon
{N,{D,J}}=>L
fsi
retourne(L)
ffonction
```

On cherche la liste des nombres premiers plus petit que $\sqrt{N}$ et on met cette liste dans la variable $PREM$. Lorsque N>1, on teste si ces nombres premiers sont des diviseurs de N en utilisant `ddiv`.

```
fonction criblefact(N)
//decomposition en facteurs premiers de n
//en utilisant ddiv et crible
local D,PREM,S,LD,LDIV;
PREM:=crible(floor(sqrt(N)));
S:=dim(PREM);
LDIV:={};
1=>K
tantque (K<=S et N>1) faire
  ddiv(N,PREM[K])=>LD
  concat(LDIV,ld[2])=>LDIV;
  LD[1]=>N
  K+1=>K
```

```
ftantque
si (N != 1) alors
 concat(LDIV,[N,1])=>LDIV;
fsi
retourne(LDIV);
}
```

8.5.2 Traduction Xcas

```
ddiv(n,d):={
//ddiv renvoie [a,[d,j]] (n=a*d^j, pgcd(a,d)=1) si j!=0
//sinon [n,[]]
local l,j;
j:=0;
while (irem(n,d)==0){
n:=iquo(n,d);
j:=j+1;
}
if (j==0){
l:=[n,[]];
} else {
l:=[n,[d,j]];
}
return(l);
}
```

```
criblefact(n):={
//decomposition en facteurs premiers de n
//en utilisant ddiv et crible
local d,prem,s,ld,ldiv;
prem:=crible(floor(sqrt(n)));
s:=size(prem);
ldiv:=[];
for (k:=0;k<s;k++){
ld:=ddiv(n,prem[k]);
ldiv:=concat(ldiv,ld[1]);
n:=ld[0];
k:=k+1;
}
if (n!=1){
ldiv:=concat(ldiv,[n,1]);
}
return(ldiv);
}
```

8.6 La liste des diviseurs

8.6.1 Les programmes avec les élèves

L'algorithme naïf :

```
pour j de 1 a n faire
   si (j divise n) alors
     afficher j
   fsi
fpour
```

Les élèves remarquent que l'on peut avoir les diviseurs deux par deux.

```
pour j de 1 a E(√n) faire
   si (j divise n) alors
     afficher j, n/j
   fsi
fpour
```

Malheureusement, lorsque l'entier n est le carré de p, p figure deux fois dans l'affichage des diviseurs.
On améliore donc l'algorithme :

```
1 → j
tantque j<√n faire
   si (j divise n) alors
     afficher j, n/j
   fsi
   j+1 → j
ftantque
si j·j=n alors
   afficher j
fsi
```

Remarque Les programmes sont ensuite mis sur des calculatrices, c'est pourquoi les algorithmes précédents utilisent la commande `afficher`. Si on veut écrire un programme avec `Xcas` on fera une fonction : on mettera les diviseurs dans une liste qui sera à la fin la valeur de la fonction en utilisant la commande `retourne` par exemple :

Traduction Xcas de l'algorithme naïf

```
nbdivis(n):={
local j, L,sn;
L:=[];
j:=1;
sn:=sqrt(n)
tantque j<sn faire
si irem(n,j)==0 alors L:=concat(L,[j,n/j]); fsi;
j:=j+1;
ftantque;
si j*j==n alors L:=append(L,j) fsi;
retourne L;
```

```
};
```

8.6.2 Le nombre de diviseurs d'un entier n

On décompose n en facteurs premiers, puis on donne aux exposants de ces facteurs premiers toutes les valeurs possibles. Si $n = a^\alpha * b^\beta * c^\gamma$ l'exposant de a paut prendre $\alpha + 1$ valeurs (0..α), celui de b peut prendre $\beta + 1$ valeurs et celui de c peut prendre $\gamma + 1$ valeurs donc le nombre de diviseurs de n est $(\alpha + 1) * (\beta + 1) * (\gamma + 1)$.

8.6.3 L'algorithme sur un exemple

Déscription de l'algorithme sur un exemple :
$n = 360 = 2^3 * 3^2 * 5$
n a donc (3+1)*(2+1)*(1+1)=24 diviseurs.
On les écrit en faisant varier le triplet représentant les exposants avec l'ordre :
(0,0,0),(1,0,0),(2,0,0),(3,0,0),
(0,1,0),(1,1,0),(2,1,0),(3,1,0),
(0,2,0),(1,2,0),...,
(0,2,1),(1,2,1),(2,2,1),(3,2,1))
On a $(a_1, \beta, \gamma) < (b_1, b_2, b_3)$ si :
$\gamma < b_3$ ou
$\gamma = b_3$ et $\beta < b_2$ ou
$\gamma = b_3$ et $\beta = b_2$ et $a_1 < b_1$.
On obtient les 4*3*2=24 diviseurs de 360 :
1,2,4,8,3,6,12,24,9,18,36,72,5,10,20,40,15,30,60,120,45,90,180,360.
que l'on peut écrire en le tableau suivant :
1,2,4,8 (les puissances de 2)
3,6,12,24 (3*les puissances de 2)
9,18,36,72 (3*3*les puissances de 2)
5,10,20,40 (5*les puissances de 2)
15,30,60,120 (5*3*les puissances de 2)
45,90,180,360 (5*3*3*les puissances de 2).
Comment obtient-on la liste des diviseurs de $a^\alpha * b^\beta * c^\gamma$ à partir de la liste `L1` des diviseurs de $a^\alpha * b^\beta$?
Il suffit de rajouter à `L1` la liste `L2` constituée par :
$\texttt{c} * \texttt{L1}, ..., \texttt{c}^\gamma * \texttt{L1}$
Dans le programme cette liste de diviseurs (`L1`) sera donc constituée au fur et à mesure au moyen d'une liste (`L2`) qui correspond au parcours de l'arbre.
On initialise `L1` avec `{1}`, puis on rajoute à `L1` la liste `L2` formée par :
$\texttt{a} * \texttt{L1}, ..., \texttt{a}^\alpha * \texttt{L1}$.
Puis on recommence avec le diviseur suivant :
on rajoute à `L1` la liste `L2` formée par $\texttt{b} * \texttt{L1}, ..., \texttt{b}^\beta * \texttt{L1}$ etc...

8.6.4 Les algorithmes donnant la liste des diviseurs de n

La liste `L1` est la liste destinée à contenir les diviseurs de `N`.
Au dèbut $\texttt{L1} = \{1\}$ et $\texttt{L2} = \{\}$.
Pour avoir la liste des diviseurs de `N`, on cherche `A` le premier diviseur de `N` et on

cherche a la puissance avec quelle A divise N.
On définit la liste L2 :
L2 est obtenue en concaténant, les listes L1*A, $L1*A^2$, ...,$L1*A^a$: au début L1={1} donc $L2=\{A,A^2,...,A^a\}$.
On modifie la liste L1 en lui concaténant la liste L2, ainsi $L1=\{1,A,A^2,...,A^a\}$.
Puis, on vide la liste L2. On cherche B le deuxième diviseur éventuel de N et on cherche b la puissance avec quelle B divise N.
On définit la nouvelle liste L2 :
L2 est obtenue en concaténant, les listes L1*B, $L1*B^2$, ..., $L1*B^b$ (c'est à dire $L2=\{B,B*A,B*A^2,..,B*A^aB^2,B^2*A,B^2*A^2,..,B^2*A^a,..,B^b*A^a,\}$)
On modifie la liste L1 en lui concaténant la liste L2, ainsi :
$L1=\{1,A,A^2,...,A^a,B,B*A,B*A^2,...,B*A^a,...,B^b,B^b*A,B^b*A^2,...,B^b*A^a\}$.
Et ainsi de suite, jusqu'à avoir epuisé tous les diviseurs de N.

Traduction Algorithmique

```
fonction NDIV0(N)
local D,L1,L2,K
2 => D
{1} => L1
tantque (N ≠ 1) faire
{}=>L2
0 => K :
tantque ((N MOD D) =0) faire
N/D => N
K + 1 => K
concat(L2,L1 * D^K) => L2
ftantque
concat(L1,L2) => L1
D + 1 => D
ftantque
retourne(L1)
```

Traduction Xcas

```
ndiv0(n):={
  local d,l1,l2,k;
  d:=2;
  l1:=[1];
  while (n!=1) {
    l2:=[];
    k:=0;
    while (irem(n,d)==0) {
      n:=iquo(n,d);
      k:=k+1;
      l2:=concat(l2,l1*d^k);
    }
    l1:=concat(l1,l2);
```

```
    d:=d+1;
  }
  return(l1);
}
```

On peut améliorer ce programme en calculant `l1*d^k` au fur et à mesure sans utiliser `k`....
On a en effet sur l'exemple précédent $n = 360 = 2^3 * 3^2 * 5$:
`l1:=[1];`
les puissances de 2 sont obtenus avec `l2:=l1` on a alors `l2:=[1]`
`l2:=l2*2;l1:=concat(l1,l2)` on a alors `l2:=[2];l1:=[1,2]`
`l2:=l2*2;l1:=concat(l1,l2)` on a alors `l2:=[4];l1:=[1,2,4]`
`l2:=l2*2;l1:=concat(l1,l2)` on a alors `l2:=[8];l1:=[1,2,4,8]`
les puissances de 3 sont obtenus avec `l2:=l1` on a alors `l2:=[1,2,4,8]`
`l2:=l2*3;l1:=concat(l1,l2)` on a alors `l2:=[3,6,12,24]; l1:=[1,2,4,8,3,6,12,24]`
`l2:=l2*3;l1:=concat(l1,l2)` on a alors `l2:=[9,18,36,72]; l1:=[1,2,4,8,3,6,12,24,9,18`
les puissances de 5 sont obtenus avec `l2:=l1` on a alors `l2:=[1,2,4,8,3,6,12,24,9,18,36,72]`
`l2:=l2*5;l1:=concat(l1,l2)` on a alors `l2:=[5,10,20,40,15,30,60,120,45,90,180,360]`
donc
`l1:=[1,2,4,8,3,6,12,24,9,18,36,72,5,10,20,40,15,30,60,120,45,90,180,360]`
On tape :

```
ndiv1(n):={
  local d,l1,l2;
  d:=2;
  l1:=[1];
  while (n!=1) {
    l2:=l1;
    while (irem(n,d)==0) {
      n:=iquo(n,d);
      l2:=l2*d
      l1:=concat(l1,l2);
    }
    d:=d+1;
  }
  return(l1);
}:;
```

On peut encore améliorer ce programme si on tient compte du fait qu'après avoir éventuellement divisé `N` par 2 autant de fois qu'on le pouvait, les diviseurs potentiels de `N` sont impairs.
On remplace alors :
`D + 1 => D`
par :
`si D=2 alors D + 1 => D sinon`
`D + 2 => D fsi`

On améliore le programme précédent en remarquant que, si le diviseur potentiel `D` est tel que $\mathtt{D} > \sqrt{\mathtt{N}}$, c'est que `N` est premier ou vaut 1. On ne continue donc pas

la recherche des diviseurs de N et quand N est diffèrent de 1 on complète L1 par L1*N.
Et aussi, on ne teste comme diviseur potentiel de N, que les nombres 2, 3, puis les nombres de la forme $6*k-1$ ou de la forme $6*k+1$ (pour $k \in \mathbb{N}$).
On remplace donc :
si D=2 alors D + 1 => D sinon
D + 2 => D fsi
par :
si D<4 alors 2 * D − 1 => D sinon
D + (4 * D mod 6) => D fsi

Traduction Algorithmique

```
fonction ndiv2(N)
local D,L1,L2,K
2 => D
{1} => L1
tantque (D ≤ √N) faire
{}=>L2
0 => K :
tantque ((N MOD D) =0) faire
N/D => N
K + 1 => K
concat(L2, L1 * D^K) => L2
ftantque
concat(L1, L2) => L1
si D<4 alors 2 * D − 1 => D sinon
D + (4 * D mod 6) => D fsi
ftantque
si N ≠ 1 alors
concat(L1, L1 * N) => L1
fsi
retourne(L1)
```

Traductions Xcas des améliorations

```
ndivi(n):={
  local d,l1,l2,k;
  d:=2;
  l1:=[1];
  l2:=l1
  while (irem(n,d)==0) {
      n:=iquo(n,d);
      l2:=l2*d
      l1:=concat(l1,l2);
    }
  d:=3;
   while (d<=sqrt(n) and n>1) {
```

```
    l2:=l1;
    while (irem(n,d)==0) {
      n:=iquo(n,d);
      l2:=l2*d
      l1:=concat(l1,l2);
    }
    d:=d+2;
  };
  if (n!=1) {l1:=concat(l1,l1*n)};
  return(l1);
};:
```

Si on ne teste pas `d<sqrt(n)` le programme est plus simple :

```
ndivis(n):={
  local d,l1,l2,k;
  d:=2;
  l1:=[1];
  l2:=l1
  while (irem(n,d)==0) {
      n:=iquo(n,d);
      l2:=l2*d
      l1:=concat(l1,l2);
    }
  d:=3;
   while (n>1) {
    l2:=l1;
    while (irem(n,d)==0) {
      n:=iquo(n,d);
      l2:=l2*d
      l1:=concat(l1,l2);
    }
    d:=d+2;
  };
  return(l1);
};:
```

Ou bien on utilise `ifactors` : on obtient un programme plus rapide surtout lorsqu'il y a de grands facteurs premiers car la décomposition en facteurs premiers est optimisée.
`ndivfact(n)` renvoie la liste des diviseurs de `n` et la longueur de cette liste. On calcule la longueur de cette liste en faisant le produit des exposants augmentés de 1. Par exemple si `n=360`$= 2^3 * 3^2 * 5$, la liste des diviseurs de `n=360` a comme longueur `sd:=(3+1)*(2+1)*(1+1)`$= 24$
La liste `l1` va contenir la liste des diviseurs et pour chaque nouveau diviseur de n, la liste `l2` contient les nouveaux diviseurs qui doivent être rajoutés à `l1`.
Par exemple si `n=360` au début `l1:=[1]` et `l2:=l1`
`d:=2` est un diviseur donc
`l2:=2*l2` i.e. `l2:=[2]` et `l1:=concat(l1,l2)` i.e. `l1=[1,2]`

`d:=2` est encore un diviseur donc
`l2:=2*l2` i.e. `l2:=[4]` et `l1:=concat(l1,l2)` i.e. `l1=[1,2,4]`
`d:=2` est encore un diviseur donc
`l2:=2*l2` i.e. `l2:=[8]` et `l1:=concat(l1,l2)` i.e. `l1=[1,2,4,8]`
On a épuiser le diviseur 2. On recopie `l1` dans `l2`
`l2:=l1` i.e. `l2=[1,2,4,8]`
le nouveau diviseur est 3 donc
`l2:=3*l2` i.e. `l2:=[3,6,12,24]` et `l1:=concat(l1,l2)` i.e. `l1=[1,2,4,3,6,12,24]`
`d:=3` est encore un diviseur donc
`l2:=3*l2` i.e. `l2:=[9,18,36,72]` et `l1:=concat(l1,l2)` i.e. `l1=[1,2,4,8,3,6,12,24`
On a épuiser le diviseur 3. On recopie `l1` dans `l2`
`l2:=l1` i.e. `l2=[1,2,4,8,3,6,12,24,9,18,36,72]` le nouveau diviseur
est 5 donc
`l2:=5*l2` i.e. `l2:=[5,10,20,40,15,30,60,120,45,90,180,360]` et
`l1:=concat(l1,l2)` i.e.
`l1=[1,2,4,8,3,6,12,24,9,18,36,72,5,10,20,40,15,30,60,120,45,90,180,360`
On a épuiser tous les diviseurs donc les diviseurs de 360 sont `l1=[1,2,4,8,3,6,12,24,9,18,36,`
On tape :

```
ndivfact(n):={
  local F,d,l1,l2,k,kd,sf,sd,j;
  si n==0 alors retourne "erreur"; fsi;
  si n<0 alors n:=-n; fsi;
  F:=ifactors(n);
  sf:=size(F)-1;
  sd:=1;
  pour k de 1 jusque sf pas 2 faire
   sd:=sd*(F[k]+1);
  fpour;
  k:=1;
  l1:=[1];
    while (k<=sf) {
    l2:=l1;kd:=F[k];
    d:=F[k-1];
    pour j de 1 jusque kd faire
      l2:=l2*d
      l1:=concat(l1,l2);
    fpour;
  k:=k+2;
  };
  return l1,sd;
  }:;
```

Comparons les temps d'exécution sur 2 exemples.
On tape :
`ndivis(30!);`
On obtient :
`Temps mis pour l'évaluation: 7.53`
On tape :

```
ndivi(30!);
```
On obtient :
```
Temps mis pour l'évaluation: 8.03
```
On tape :
```
ndivfact(30!);
```
On obtient :
```
Temps mis pour l'évaluation: 7.33
```
Mais si on tape :
```
ndivis(30!*907);
```
On obtient :
```
Temps mis pour l'évaluation: 19.28
```
On tape :
```
ndivi(30!*907);
```
On obtient :
```
Temps mis pour l'évaluation: 17.07
```
On tape :
```
ndivfact(30!*907);
```
On obtient :
```
Temps mis pour l'évaluation: 17.07
```
Donc dans le 1ier cas `ndivis` va plus vite et dans le 2nd cas c'est `ndivi` qui va plus vite.

8.7 La liste des diviseurs avec la décomposition en facteurs premiers

8.7.1 FPDIV

On utilise le programme `factprem` (qui donne la liste des facteurs premiers de `N` (cf 8.4.2) pour obtenir la liste des diviseurs de `N` selon l'algorithme utilisé dans `NDIV1`.

Traduction Algorithmique

```
fonction fpdiv(N)
//renvoie la liste des diviseurs de n en utilisant factprem
local L1,L2,L3,D,ex,S
factprem(N)=>L3
dim(L3)=>S
{1}=>L1
pour  K de 1 a S-1 pas 2 faire
{}=>L2
L3[K]=>D
L3[K+1]=>ex
pour  J de 1 a ex faire
concat(L2,L1*(D^J))=>L2
}
concat(L1,L2)=>L1
}
```

```
retourne(L1)
}
```

Traduction Xcas

```
fpdiv(n):={
//renvoie la liste des diviseurs de n en utilisant factprem
local l1,l2,l3,d,ex,s;
l3:=factprem(n);
s:=size(l3);
l1:=[1];
for (k:=0;k<s-1;k:=k+2) {
l2:=[];
d:=l3[k];
ex:=l3[k+1];
for (j:=1;j<=ex;j++) {
l2:=concat(l2,l1*(d^j));
}
l1:=concat(l1,l2);
}
return(l1);
}
```

8.7.2 CRIBLEDIV

Pour obtenir la liste des diviseurs de N selon l'algorithme utilisé dans NDIV1, on utilise le programme criblefact (cf 8.5.2) qui donne la liste des facteurs premiers de N.

8.7.3 Traduction Algorithmique

```
fonction cribledivi(N)
//renvoie la liste des diviseurs de n en utilisant factprem
local L1,L2,L3,D,ex,S
criblefact(N)=>L3
dim(L3)=>S
{1}=>L1
pour  K de 1 a S-1 pas 2 faire
{}=>L2
L3[K]=>D
L3[K+1]=>ex
pour  J de 1 a ex faire
concat(L2,L1*(D^J))=>L2
}
concat(L1,L2)=>L1
}
retourne(L1)
}
```

Traduction Xcas

```
criblediv(n):={
//renvoie la liste des diviseurs de n en utilisant criblefact
local l1,l2,l3,d,ex;
l3:=criblefact(n);
s:=size(l3);
l1:=[1];
for (k:=0;k<s-1;k:=k+2) {
l2:=[];
d:=l3[k];
ex:=l3[k+1];
for (j:=1;j<=ex;j++) {
l2:=concat(l2,l1*(d^j));
}
l1:=concat(l1,l2);
}
return(l1);
}
```

8.8 Calcul de A^P mod N

8.8.1 Traduction Algorithmique

-Premier algorithme
On utilise deux variables locales PUIS et I.
On fait un programme itératif de façon qu'à chaque étape, PUIS représente $A^I \bmod N$

```
fonction puimod1 (A, P, N)
local PUIS, I
1=>PUIS
pour I de 1 a P faire
  A*PUIS mod N =>PUIS
fpour
retourne PUIS
ffonction
```

-Deuxième algorithme
On n'utilise ici qu'une seule variable locale PUI, mais on fait varier P de façon qu'à chaque étape de l'itération on ait :
$PUI*A^P \bmod N = constante$. Au début $PUI = 1$ donc $constante = A^P \bmod N$ (pour la valeur initiale du paramètre P, c'est à dire que cette $constante$ est égale à ce que doit retourner la fonction), et, à chaque étape, on utilise l'égalité $PUI*A^P \bmod N = (PUI*A \bmod N)*A^{P-1} \bmod N$, pour diminuer la valeur de P, et pour arriver à la fin à $P = 0$, et alors on a la $constante = PUI$.

```
fonction puimod2 (A, P, N)
local PUI
1=>PUI
```

```
tantque  P>0  faire
  A*PUI mod N =>PUI
  P-1=>P
ftantque
retourne PUI
ffonction
```

-Troisième algorithme
On peut aisément modifier ce programme en remarquant que :
$A^{2*P} = (A * A)^P$.
Donc quand P est pair, on a la relation :
$PUI * A^P = PUI * (A * A \bmod N)^{P/2} \bmod N$
et quand P est impair, on a la relation :
$PUI * A^P = (PUI * A \bmod N) * A^{P-1} \bmod N$.
On obtient alors, un algorithme rapide du calcul de $A^P \bmod N$.

```
fonction puimod3 (A, P, N)
local PUI
1=>PUI
tantque  P>0  faire
  si P mod 2 =0 alors
    P/2=>P
    A*A mod N=>A
  sinon
    A*PUI mod N =>PUI
    P-1=>P
  fsi
ftantque
retourne PUI
ffonction
```

On peut remarquer que si P est impair, $P - 1$ est pair.
On peut donc écrire :

```
fonction puimod4 (A, P, N)
local PUI
1=>PUI
tantque  P>0  faire
  si P mod 2 =1 alors
    A*PUI mod N =>PUI
    P-1=>P
  fsi
P/2=>P
A*A mod N=>A
ftantque
retourne PUI
ffonction
```

-Programme récursif
On peut définir la puissance par les relations de récurrence : $A^0 = 1$
$A^{P+1} \bmod N = (A^P \bmod N) * A \bmod N$

```
fonction puimod5(A, P, N)
si P>0 alors
retourne puimod5(A, P-1, N)*A mod N
sinon
retourne 1
fsi
ffonction
```

-Programme récursif rapide

```
fonction puimod6(A, P, N)
si P>0 alors
  si P mod 2 =0 alors
    retourne puimod6((A*A mod N), P/2, N)
  sinon
    retourne puimod6(A, P-1, N)*A mod N
  fsi
sinon
retourne 1
fsi
ffonction
```

8.8.2 Traduction Xcas

```
puimod(a,p,n):={
//calcule recursivement la puissance rapide a^p modulo n
 if (p==0){
    return(1);
 }
 if (irem(p,2)==0){
    return(puimod(irem(a*a,n),iquo(p,2),n));
 }
 return(irem(a*puimod(a,p-1,n),n));
}
```

8.8.3 Un exercice

Étant donné deux entiers $a \in \mathbb{N}^*$ et $n \in \mathbb{N}$, $n \geq 2$, on veut connaitre les différentes valeurs de $a^p \bmod n$ pour $p \in \mathbb{N}$, c'est à dire l'orbite de a dans $(\mathbb{Z}/n\mathbb{Z}, \times)$. On démontre que l'orbite se termine toujours par un cycle puisque $\mathbb{Z}/n\mathbb{Z}$ a un nombre fini d'éléments.

— Trouver l'orbite de $2^p \bmod 24$

— Écrire une fonction de a et n qui renvoie les plus petis entiers $h \geq 0$ et $T > 0$ vérifiant :

$$a^h = a^{h+T} \bmod n$$

et la liste des $a^p \bmod n$ pour $p = 0..h + T - 1$

— Écrire une fonction de a, et n qui représente graphiquement $a^p \bmod n$ en fonction de p.

Solution On tape : `(irem(2^p ,24)$(p=0..10))`
On obtient : `1,2,4,8,16,8,16,8,16,8,16` donc $h = 3$ et $T = 2$
On utilise la commande `member` qui teste si un élément est dans une liste et `member` renvoie soit l'indice +1, soit 0. On peut utiliser soit un `tantque` soit un `repeter` (`tantque non arrêt faire....tantque` ou `repeter ...jusqua arrêt`) et on remarquera le test d'arrêt. On sait qu'une affectation renvoie la valeur affectée, donc `k:=member(b,L)` renvoie soit 0 soit un nombre non nul. On fait une boucle et on s'arrete quand `k:=member(b,L)` est non nul.

```
orbite1(a,n):={
local k,h,T,p,b,L;
L:=[1];
p:=1;
b:=irem(a,n);
tantque !(k:=member(b,L)) faire
L:=append(L,b);
b:=irem(b*a,n);
p:=p+1;
ftantque;
h:=k-1;
T:=p-h;
return h,T,L;
}:;

orbite2(a,n):={
local k,h,T,p,b,L;
L:=[];
p:=0;
b:=1;
repeter
L:=append(L,b);
b:=irem(b*a,n);
p:=p+1;
jusqua (k:=member(b,L));
h:=k-1;
T:=p-h;
return h,T,L;
}:;
```

On dessine les points du cycle et de 2 périodes avec la couleur a ou la couleur 0 lorsque $a = 7$:

```
dessin(a,n):={
local k,h,T,L,P,s,LT;
P:=NULL;
h,T,L:=orbite1(a,n);
s:=dim(L);
LT:=mid(L,h);
L:=concat(concat(L,LT),LT);
```

```
pour k de 0 jusque s+2*T-1 faire
P:=P,point(k,L[k]);
fpour;
si a==7 alors return affichage(P,epaisseur_point_3);fsi;
return affichage(P,a+epaisseur_point_3);
}:;
```

On tape : `dessin(2,11)`
On tape : `dessin(3,11)`
On tape : `dessin(2,9)`
On tape : `dessin(3,9)`
et on observe.....
On peut rappeler

— le Théorème de Fermat :si n est premier et $0 < a < n$, alors :

$$a^{n-1} = 1 \bmod n$$

— la Généralisation du théorème de Fermat : si a et n sont premiers entre eux, alors si `euler(n)`=$\phi(n)$ est l'indicatrice d'Euler, on a :

$$a^{\phi(n)} = 1 \bmod n$$

On tape :

```
est_premier_avec(n):={
local L,a;
L:=NULL;
pour a de 1 jusque n-1 faire
si gcd(a,n)==1 alors L:=L,a; fsi;
fpour;
return L;
}:;
```

Puis, on tape : `E:=est_premier_avec(9)`
On obtient : `1,2,4,5,7,8`
et on a bien : `euler(9)=6=dim(E)`
Une démonstration rapide de ces théorèmes :
Si a et n sont premiers entre eux, a est inversible dans $\mathbb{Z}/n\mathbb{Z}, \times$ Soit E l'ensemble des nombres de $[1..n]$ qui sont premiers avec n (`E:=est_premier_avec(n)`) et soit `Ea` l'ensemble des $k*a$ pour $k \in E$. Tous les éléments de Ea sont distincts et inversibles dans $\mathbb{Z}/n\mathbb{Z}, \times$: donc les ensembles E et Ea sont les mêmes. En faisant le produit de tous ces éléments on obtient $\Pi_{k\in E}k = \Pi_{k\in Ea}k = a^{\phi}(n)\Pi_{k\in E}k$, $\Pi_{k\in E}k$ étant inversible dans $\mathbb{Z}/n\mathbb{Z}, \times$ on en déduit que $a^{\phi}(n) = 1$.

8.9 La fonction "estpremier"

8.9.1 Traduction Algorithmique

- Premier algorithme

On va écrire un fonction booléenne de paramètre N, qui sera égale à $VRAI$ quand

N est premier, et, à $FAUX$ sinon.
Pour cela, on cherche si N posséde un diviseur différent de 1 et inférieur ou égal à $E(\sqrt{N})$ (partie entière de racine de N).
On traite le cas N=1 à part !
On utilise une variable booléenne $PREM$ qui est au départ à $VRAI$, et qui passe à $FAUX$ dès que l'on rencontre un diviseur de N.

```
Fonction estpremier(N)
local PREM, I, J

E(√N) =>J

Si N = 1 alors
  FAUX=>PREM
  sinon
  VRAI=>PREM
fsi
2=>I

tantque PREM et I ≤J faire

  si N mod I = 0 alors
     FAUX=>PREM
     sinon
     I+1=>I
  fsi
ftantque
retourne PREM
ffonction
```

-Première amélioration
On peut remarquer que l'on peut tester si N est pair, et ensuite, tester si N posséde un diviseur impair.

```
Fonction estpremier(N)
local PREM, I, J

E(√N) =>J
Si (N = 1) ou (N mod 2 = 0) et N≠2 alors

  FAUX=>PREM
  sinon
  VRAI=>PREM
fsi
3=>I

tantque PREM et I ≤J faire

  si N mod I = 0 alors
     FAUX=>PREM
     sinon
```

```
      I+2=>I
  fsi
ftantque
retourne PREM
ffonction
```

- Deuxième amélioration
On regarde si N est divisible par 2 ou par 3, sinon on regarde si N posséde un diviseur de la forme $6 \times k - 1$ ou $6 \times k + 1$ (pour $k \in \mathbb{N}$).

```
Fonction estpremier(N)
local PREM, I, J
```

E($\sqrt{N}$) =>J

```
Si (N = 1) ou (N mod 2 = 0) ou ( N mod 3 = 0) alors
  FAUX=>PREM
  sinon
  VRAI=>PREM
fsi
si N=2 ou N=3 alors
VRAI=>PREM
fsi
5=>I
```

tantque PREM et I $\leq$J faire

```
  si (N mod I = 0) ou (N mod I+2 =0) alors
     FAUX=>PREM
     sinon
     I+6=>I
  fsi
ftantque
retourne PREM
ffonction
```

8.9.2 Traduction Xcas

```
estprem(n):={
//teste si n est premier
  local prem,j,k;
  if ((irem(n,2)==0) or (irem(n,3)===0) or (n==1)) {
     return(false);
  }
  if ((n==2) or (n==3)) {
     return(true);
  }
  prem:=true;
  k:=5;
  while ((k*k<=n) and prem) {
     if (irem(n,k)==0 or irem(n,k+2)==0) {
```

```
        prem:=false;
     }
     else {
        k:=k+6;
     }
  }
  return(prem);
}
```

8.10 La fonction estpremc en utilisant le crible

8.10.1 Traduction algorithmique

```
fonction estpremc(N)
//utilise la fonction crible pour tester si n est premier
local PREM,S;
crible(floor(sqrt(N)))=>PREM
dim(PREM)=>S
si  (N=1) retourne(FAUX)
pour K de 1 a S faire
 si  (N mod ,PREM[K])=0)
    retourne(FAUX);
 fsi
fpour
retourne(VRAI)
ffonction
```

8.10.2 Traduction Xcas

```
estpremc(n):={
//utilise la fonction crible pour tester si n est premier
local prem,s;
prem:=crible(floor(sqrt(n)));
s:=size(prem);
if (n==1) return(false);
for (k:=0;k<s;k++){
 if (irem(n,prem[k])==0){
    return(false);
 }
}
return(true);
}
```

8.11 Méthode probabiliste de Mr Rabin

Si N est premier alors tous les nombres K strictement inférieurs à N sont premiers avec N, donc d'après le petit théorème de Fermat on a :
$K^{N-1} = 1 \bmod N$

Par contre, si N n'est pas premier, les entiers K $(1 < K < N)$ vérifiant :
$K^{N-1} = 1 \mod N$ sont peu nombreux.
La méthode probabiliste de Rabin consiste à prendre au hasard un nombre K dans l'intervalle $[2\ ;\ N-1]$ ($1 < K < N$) et à calculer :
$K^{N-1} \mod N$
Si $K^{N-1} = 1 \mod N$ on refait un autre tirage du nombre K, et, si $K^{N-1} \neq 1 \mod N$ on est sûr que N n'est pas premier.
Si on obtient $K^{N-1} = 1 \mod N$ pour 20 tirages successifs de K on peut conclure que N est premier avec une probabilité d'erreur faible :
on dit alors que N est pseudo-premier.
Bien sûr cette méthode est employée pour savoir si de grands nombres sont pseudo-premiers mais on préfére utiliser la méthode de Miller-Rabin (cf 8.12) qui est aussi une méthode probabiliste mais qui donne N premier avec une probabilité d'erreur plus faible (inférieure à $(0.25)^{20}$ si on a effectué 20 tirages, soit, une erreur de l'ordre de 10^{-12}).

8.11.1 Traduction Algorithmique

On suppose que :
`hasard(N)` donne un nombre entier au hasard entre 0 et $N-1$.
Le calcul de $K^{N-1} \mod N$ se fait grâce à l'algorithme de la puissance rapide (cf page 125).
On suppose que :
`powmod(K, P, N)` calcule $K^P \mod N$

```
Fonction estprem(N)
local K, I, P
1=>I
1=>P
tantque P = 1 et I < 20 faire
hasard(N-2)+2=>K
powmod(K, N-1, N)=>P
I+1=>I
ftantque
Si P =1 alors
retourne VRAI
sinon
retourne FAUX
fsi
ffonction
```

8.11.2 Traduction Xcas

La fonction `powmod` existe dans `Xcas` : il est donc inutile de la programmer.

```
rabin(n):={
//teste par la methode de Rabin si n est pseudo-premier
local k,j,p;
j:=1;
```

```
p:=1;
while ((p==1) and (j<20)) {
k:=2+rand(n-2);
p:=powmod(k,n-1,n);
j:=j+1;
}
if (p==1) {
return(true);
}
return(false);
}
```

8.12 Méthode probabiliste de Mr Miller-Rabin

8.12.1 Un exemple

Rappel Le théorème de Fermat :
Si n est premier et si k est un entier quelconque alors $k^n = k \bmod n$.
et donc
Si n est premier et si k est premier avec n alors $k^{n-1} = 1 \bmod n$.

Soit $N = 561 = 3 * 11 * 17$. Il se trouve que l'on a : pour tout A $(A < N)$, on a $A^N = A \mod N$, donc si A est premier avec N on a $A^{N-1} = 1 \mod N$, le test de Rabin est donc en defaut, seulement pour A non premier avec N. Par exemple on a :
$3^{560} = 375 \mod 561$
$11^{560} = 154 \mod 561$
$17^{560} = 34 \mod 561$
$471^{560} = 375 \mod 561$
mais pour tous les nombres A non multiples de 3, 11 ou 17 on a :
$A^{N-1} = 1 \mod 561$.
Par exemple on a :
$5^{560} = 1 \mod 561$.
$52^{N-1} = 1 \mod 561$.
On risque donc de dire avec le test de Rabbin que 561 est pseudo-premier.
Il faut donc affiner le test en remarquant que si N est premier l'équation : $X^2 = 1 \bmod N$ n'a pour solution que $X = 1 \bmod N$ ou $X = -1 \bmod N$.
Le test de Miller-Rabin est basé sur cette remarque.
Pour $N = 561$, $N - 1 = 560$, on a : $560 = 35 * 2^{16}$
$13^{35} = 208 \mod 561$
$13^{35*2} = 67 \mod 561$
$13^{35*4} = 1 \mod 561$
$13^{35*8} = 1 \mod 561$...
On vient de trouver que 67 est solution de $X^2 = 1 \bmod 561$ donc on peut affirmer que 561 n'est pas premier.
$A = 13$ vérifie le test de Rabin car $13^{560} = 1 \mod 561$
mais ne vérifie pas le test de Miller-Rabin car
$13^{35*2} \neq -1 \bmod 561$ et $13^{35*2} \neq 1 \bmod 561$

et pourtant $13^{35*4} = 13^{35*4} = 1 \mod 561$
Par contre ce test ne suffit pas pour affirmer qu'un nombre est premier car :
$101^{35} = 560 = -1 \bmod 561$ et donc $101^{35*2} = 1 \bmod 561$ et cela ne fournit pas de solutions autre que 1 ou -1 à l'équation $X^2 = 1 \bmod 561$.

8.12.2 L'algorithme

L'algorithme est basé sur :
1/ Le petit théorème de Fermat :
$A^{N-1} = 1 \bmod N$ si N est premier et si $A < N$.
2/ Si N est premier, l'équation $X * X = 1 \bmod N$ n'a pas d'autres solutions que $X = 1 \bmod N$ ou $X = -1 \bmod N$.
En effet il existe un entier k vérifiant $X * X - 1 = (X + 1) * (X - 1) = k * N$ donc,
puisque N est premier, N divise $X + 1$ ou $X - 1$. On a donc soit $X = 1 \bmod N$ ou $X = -1 \bmod N$.
On élimine les nombres pairs que l'on sait ne pas être premiers.
On suppose donc que N est impair et donc que $N - 1$ est pair et s'écrit :
$N - 1 = 2^t * Q$ avec $t > 0$ et Q impair.
Si $A^{N-1} = 1 \bmod N$ c'est que $A^{N-1} \bmod N$ est le carré de $B = A^{\frac{N-1}{2}} = A^{2^{t-1}Q} \bmod N$.
Si on trouve $B \neq 1 \bmod N$ et $B \neq -1 \bmod N$ on est sûr que N n'est pas premier.
Si $B = -1 \mod N$ on recommence avec une autre valeur de A.
Si $B = 1 \mod N$ on peut recommencer le même raisonnement si $\frac{N-1}{2}$ est encore pair ($B = A^{\frac{N-1}{2}} = (A^{\frac{N-1}{4}})^2 \bmod N$) ou
si $\frac{N-1}{2}$ est impair, on recommence avec une autre valeur de A.
On en déduit que :
si $N - 1 = 2^t.Q$ et
si $A^{N-1} = 1 \bmod N$ et
si $A^Q \neq 1 \bmod N$ et
si pour $0 \leq ex < t$ on a $A^{2^{ex}.Q} \neq -1 \bmod N$ c'est que N n'est pas premier.
D'où la définition :
Soit N un entier positif impair égal à $1 + 2^t * Q$ avec Q impair.
On dit que N est pseudo-premier fort de base A si :
soit $A^Q = 1 \bmod N$
soit si il existe e, $0 \leq e < t$ tel que $A^{Q*2^e} = -1 \bmod N$.
On voit facilement qu'un nombre premier impair est pseudo-premier fort dans n'importe quelle base A non divisible par N.
Réciproquement on peut montrer que si $N > 4$ n'est pas premier, il existe au plus $N/4$ bases A ($1 < A < N$) pour lesquelles N est pseudo-premier fort de base A.

L'algorithme va choisir au hasard au plus 20 nombres A_k compris entre 2 et $N-1$: si N est pseudo-premier fort de base A_k pour $k = 1..20$ alors N est premier avec une tres forte probabilité égale à $(1/4)^{20} (< 10^{-12})$.
Bien sûr cette méthode est employée pour savoir si de grands nombres sont pseudo-premiers.

8.12.3 Traduction Algorithmique

On suppose que :
`hasard(N)` donne un nombre entier au hasard entre 0 et $N-1$.
Le calcul de $K^{N-1} \bmod N$ se fait grâce à l'algorithme de la puissance rapide (cf page 125).
On notera :
`powmod(K, P, N)` la fonction qui calcule $K^P \bmod N$

```
Fonction Miller(N)
local Q,P,t,C,A,B,ex
si (N=2) alors retourne FAUX
si (N mod 2)==0) alors retourne FAUX
N-1=>Q
0=>t
tantque (Q mod 2 =0) faire
t+1=>t
E(Q/2)=>Q
ftantque
//N-1=2^t*Q
20=>C
VRAI=>P
tantque (C>0 et P) faire
hasard(N-2)+2=>A
0=>ex
powmod(A, Q, N)=>B
si B<>1 alors
tant que (B<>1) et (B<>N-1) et (ex<t-1) faire
ex+1=>ex
powmod(B,2,n)=>B
ftantque
si (B<>N-1) alors
FAUX=>P
fsi
C-1=>C
ftantque
retourne P
ffonction
```

8.12.4 Traduction Xcas

La fonction `powmod` existe dans `Xcas` : il est donc inutile de la programmer.

```
miller(n):={
local p,q,t,c,a,b,ex;
if (n==2){return(true);}
if (irem(n,2)==0) {return(false);}
q:=n-1;
t:=0;
while (irem(q,2)==0) {
```

```
t:=t+1;
q:=iquo(q,2);
}
//ainsi n-1=q*2^t
c:=20;
p:=true;
while ((c>0) and p) {
//rand(k) renvoie un nombre entier de [0,k-1] si k<999999999
if (n<=10^9) {a:=2+rand(n-2);} else {a:=2+rand(999999999);}
ex:=0;
b:=powmod(a,q,n);
//si b!=1 on regarde si b^{2^(ex)}=-1 mod n (ex=0..t-1)
if (b!=1) {
while ((b!=1) and (b!=n-1) and (ex<=t-2)) {
b:=powmod(b,2,n);
ex:=ex+1;}
//si b!=n-1 c'est que n n'est pas premier
if (b!=n-1) {p:=false;}
}
c:=c-1;
}
return(p);
};
```

8.13 Numération avec Xcas

On a besoin ici des fonctions de `Xcas` :
- `asc` qui convertit un caractère ou une chaîne de caractères, en une liste de nombres et,
- `char` qui convertit un nombre ou une liste de nombres en un caractère ou une chaîne de caractères.

On a :
`char(n)` pour n entier, $(0 \leq n \leq 255)$ donne le caractère ayant comme code ASCII l'entier n.
`char(l)` pour une liste d'entiers l $(0 \leq l[j] \leq 255)$, donne la chaîne de caractères dont les caractères ont pour code ASCII les entiers $l[j]$ qui composent la liste l.
`asc(mot)` renvoie la liste des codes ASCII des lettres composant le mot.

Exemples

```
asc("A")=[65]
char(65)="A"
asc("Bonjour")= [66,111,110,106,111,117,114]
char([66,111,110,106,111,117,114])="Bonjour"
```

Remarque :
Il existe aussi la fonction `ord` qui a pour argument une chaîne de caractères mais qui renvoie le code ASCII de la première lettre de la chaîne de caractères :

```
ord("B")= 66 ord("Bonjour")= 66
```

8.13.1 Passage de l'écriture en base dix à une écriture en base b

La base b est inférieure ou égale à 10

- Version itérative

Si $n < b$, il n'y a rien à faire : l'écriture en base b est la même que l'écriture en base dix et est n. On divise n par b : $n = b * q + r$ avec $0 \leq r < b$).
Le reste r de la division euclidienne de n par b (`r:=irem(n,b)`) donne le dernier chiffre de l'écriture en base b de n. L'avant dernier chiffre de l'écriture en base b de n sera donné par le le reste de la division euclidienne de q (`q:=iquo(n,b)`) par b. On fait donc une boucle en remplacant n par q (`n:=iquo(n,b)`) tant que $n \geq b$ en mettant à chaque étape `r:=irem(n,b)` au début de la liste qui doit renvoyer le résultat.
On écrit la fonction itérative `ecritu` qui renvoie la liste des chiffres de n en base b :

```
ecritu(n,b):={
//n est en base 10 et b<=10, ecrit est une fonction iterative
//renvoie la liste des caracteres de l'ecriture de n en base b
local L;
L:=[];
while (n>=b){
L:=concat([irem(n,b)],L);
n:=iquo(n,b);
}
L:=concat([n],L);
return(L);
}
```

- Version récursive

Si $n < b$, l'écriture en base b est la même que l'écriture en base dix et est n.
Si $n \geq b$, l'écriture en base b de n est formée par l'écriture en base b de q suivi de r, lorsque q et r sont le quotient et le reste de la division euclidienne de n par b ($n = b * q + r$ avec $0 \leq r < b$).
On écrit la fonction récursive `ecritur` qui renvoie la liste des chiffres de n en base b :

```
ecritur(n,b):={
//n est en base 10 et b<=10, ecritur est recursive
//renvoie la liste des caracteres de l'ecriture de n en base b
if (n>=b)
return(concat(ecritur(iquo(n,b),b),irem(n,b)));
else
return([n]);
}
```

La base b est inférieure ou égale à 36

On choisit 36 symboles pour écrire un nombre : les 10 chiffres 0,1..9 et les 26 lettres majuscules $A, B, .., Z$.

On transforme tout nombre positif ou nul n ($n < b$) en un caractère : ce caractére est soit un chiffre (si $n < 10$) soit une lettre ($A, B...Z$) (si $9 < n < 36$).

```
chiffre(n,b):={
//transforme n (0<=n<b) en son caractere ds la base b
if (n>9)
n:=char(n+55);
else
n:=char(48+n);
return(n);
}
```

On obtient alors la fonction itérative `ecritu` :

```
ecritu(n,b):={
//n est en base 10 et b<=36, ecritu est une fonction iterative
//renvoie la liste des caracteres de l'ecriture de n en base b
local L,r,rc;
L:=[];
while (n>=b){
r:=irem(n,b);
rc:=chiffre(r,b);
L:=concat([rc],L);
n:=iquo(n,b);
}
n:=chiffre(n,b);
L:=concat([n],L);
return(L);
}
```

- Version recursive

```
ecriture(n,b):={
//n est en base 10 et b<=36, ecriture est une fonction recursive
//renvoie la liste des caracteres de l'ecriture de n en base b
local r,rc;
if (n>=b){
r:=irem(n,b);
rc:=chiffre(r,b);
return(append(ecriture(iquo(n,b),b),rc));
}
else {
return([chiffre(n,b)]);
}
}
```

En utilisant la notion de séquence on peut aussi écrire :

```
ecrit(n,b):= {
//renvoie la sequence des chiffres de n dans la base b
local m,u,cu;
```

```
  m:=(NULL);
  while(n!=0){
       u:=(irem(n,b));
       if (u>9) {
           cu:=(char(u+55));
         }
        else {
           cu:=(char(u+48));
         };
       m:=(cu,m);
       n:=(iquo(n,b));
    };
  return(m);
}
```

8.13.2 Passage de l'écriture en base b de n à l'entier n

Il faut convertir ici chaque caractère en sa valeur (on convertit le caractère contenu dans m en le nombre nm).
Si $m = (c0, c1, c2, c3)$ alors $n = c0 * b^3 + c1 * b^2 + c2 * b + c3$.
On calcule n en se servant de l'algorithme de Hörner (cf 8.15). En effet le calcul de $n = c0 * b^3 + c1 * b^2 + c2 * b + c3$ revient à calculer la valeur du polynôme $P(x) = c0 * x^3 + c1 * x^2 + c2 * x + c3$ pour $x = b$.
n va contenir successivement :
0 ($n := 0$) puis
$c0$ ($n := n * b + c0$) puis
$c0 * b + c1$ ($n := n * b + c1$) puis
$c0 * b^2 + c1 * b + c2 = (c0 * b + c1) * b + c2$ ($n := n * b + c2$) et enfin
$c0 * b^3 + c1 * b^2 + c2 * b + c3$ ($n := n * b + c3$).
On écrit donc la fonction nombre dans Xcas :

```
nombre(m,b):={
local s,k,am,nm,n;
  s:=(size(m));
  n:=(0);
  k:=(0);
  if (s!=0) {
       while(k<s){
            am:=(asc(m[k])[0]);
            if (am>64) {
                nm:=(am-55);
              }
             else {
                nm:=(am-48);
              };
            if (nm>(b-1)) {
                return("erreur");
              }
            n:=(n*b+nm);
```

```
            k:=(k+1);
          };
      }
  return(n);
}
```

8.13.3 Un exercice et sa solution

L'énoncé

On veut afficher en base dix la suite ordonnée des entiers dont l'écriture en base trois ne comporte que des 0 ou des 1 (pas de 2).

1. Calculer à la main les huit premiers termes de cette suite.
2. Décrire un algorithme qui donne les 128 premiers termes de cette suite.
3. Écrire une fonction qui renvoie la liste des n premiers termes de cette suite.

La correction

1. Voici les 8 premiers termes de cette suite :
   `[0,1,3,4,9,10,12,13]` dont lécriture en base 3 est :
   `[[0],[1],[1,0],[1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]]`

2. Il y a plusieurs algorithmes possibles :
   — On écrit les nombres de 0 à N en base 3 et met dans la liste réponse ceux qui ne contiennent pas de 2 dans leur écriture en base 3. On s'arrete quand la liste réponse contient n éléments,
   — On écrit les nombres de 0 à n en base 2 et on considère cette écriture comme étant celle d'un nombre écrit en base 3.
   — On regarde comment on peut les termes de la suite se déduisent les uns des autres. On peut remarquer que :
   0=3*0,1=3*0+1,3=3*1,4=3*1+1,9=3*3,10=3*3+1,12=3*4,13=3*4+1....
   — On regarde comment sont faits les termes de la suite. On peut remarquer que :
   3=0+3,4=1+3 donc 3 et 4 ont ètè obtenu à partir de 0 et 1 en leur ajoutant 3,
   9=0+9,10=1+9,12=3+9,13=4+9 donc 9,10,11 et 12 ont ètè obtenu à partir de 0,1,3 et 4 en leur ajoutant 9... Les 8 prochains termes de cette suite seront obtenus en ajoutant 3^3 à chacun des 8 premiers. On obtient ainsi les 16 premiers termes de cette suite. Puis les 16 prochains termes de cette suite seront obtenus en ajoutant 3^4 à chacun des 16 premiers termes etc...

3. On va traduire avec `Xcas` chacun des algorithmes ci-dessus. Voici les fonctions de `Xcas` que l'on va utiliser :
   — La fonction `est_element(L,a)` qui teste si `a` est dans la liste `L` et qui renvoie `0` ou `n+1` si `n` est l'indice de la première occurence de `a` dans `L`,
   — La fonction `convert(n,base,b)` (resp `convert(L,base,b)`) qui convertit un entier `n` en la liste des coefficients en base `b` dans

l'ordre croissant (resp qui convertit la liste `L` des coefficients en base `b` dans l'odre croissant en un entier `n`).

— Pour ajouter un nombre `a` à chacun des termes d'une liste `L` on peut utiliser : la fonction `map` et écrire `map(L,x->x+a)` ou bien utiliser l'addition de `L` et de la liste `[a,a..a]` ayant la longueur de `L` et écrire `L+[a$dim(L)]`

— Pour faire les opérations : multiplier par 3 et multiplier par 3 puis ajouter 1 sur chacun des termes d'une liste `L`, on peut utiliser : la fonction `map` et écrire `mat2list(map(L,x->[3*x,3*x+1]))` car `map(L,x->[3*x,3*x+1])` renvoie une matrice et `mat2list` transforme une matrice en liste.

Voici les programmes correspondants aux algorithmes décrits précédement à la question 2. On tape les fonctions `pasde21(n)` ..`pasde24(n)` qui renvoient les `n` premiers termes de la liste demandée. La variable `p` contient à chaque étape la dimension de la liste `L`.

—
```
pasde21(n):={
  local L,j,J,p;
  L:=[0];
  p:=1;
  j:=1;
  tantque p<n faire
    J:=convert(j,base,3);
    si not(est_element(2,J)) alors
      L:= append(L,j);
      p:=p+1;
    fsi;
    j:=j+1;
  ftantque;
  retourne L;
}
:;
```

—
```
pasde22(n):={
  local J,a,p,L;
  L:=[];
  pour p de 0 jusque n-1 faire
    J:=convert(p,base,2);
    a:=convert(J,base,3);
    L:=append(L,a)
  fpour
  retourne L;
}:;
```

— À la fin de la boucle `tantque`, `L` a $p \geq n$ éléments : il faut donc raccourcir la liste `L` (`L:=mid(L,0,n)`)

```
pasde23(n):={
  local L,p;
  L:=[0];
  p:=1;
```

```
  tantque p<n faire
    L:=mat2list(map(L,x->[x*3,x*3+1]));
    p:=2*p;
  ftantque;
  L:=mid(L,0,n);
  retourne L;
}
:;
```

Dans le programme ci-dessus la liste `L` est recréée à chaque itération, il est donc préférable de modifier ce programme pour qu'à chaque itération on ne calcule que les nouveaux termes dans la liste `LA` et ce sont ces nouveaux termes qui créront les termes suivants...

```
pasde23b(n):={
  local L,p,LA;
  L:=[0,1];
  LA:=[1];
  p:=2;
  tantque p<n faire
  LA:=mat2list(map(LA,x->[x*3,x*3+1]));
  L:=concat(L,LA);
  p:=2*p;
  ftantque;
  L:=mid(L,0,n);
  retourne L;
}
:;
```

À la fin de la boucle `tantque`, `L` a $2^j = p \geq n$ éléments : il faut raccourcir la liste `L` (`L:=mid(L,0,n)`) et on calcule des termes pour rien...On modifie encore le programme, mais comme la liste `LA` engendre les termes 2 par 2, on calcule quand même un terme de trop si `n` est impair.

```
pasde23t(n):={
  local L,p,LA;
  L:=[0,1];
  LA:=[1];
  p:=2;
  tantque 2p<=n faire
  LA:=mat2list(map(LA,x->[x*3,x*3+1]));
  L:=concat(L,LA);
  p:=2*p;
  ftantque;
  LA:=mat2list(map(mid(LA,0,iquo(n-p+1,2)),x->[x*3,x*3+1]));
  L:=concat(L,LA);
  retourne mid(L,0,n);
}
:;
```

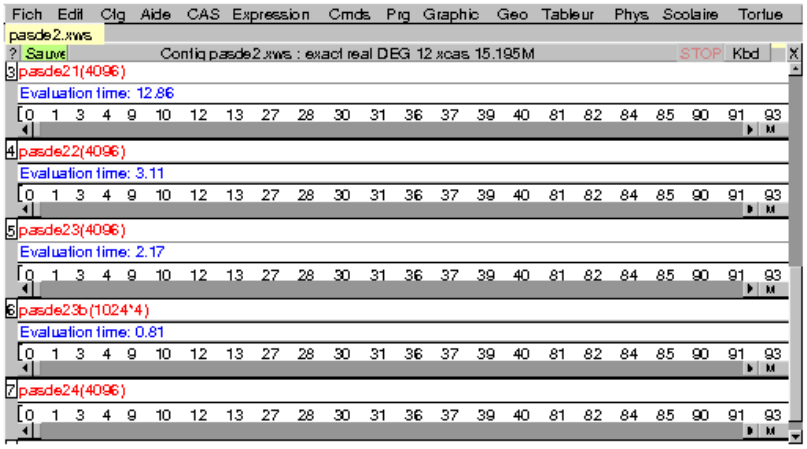
— La variable `j` contient le nombre d'iterations : à chaque étape on a $p = 2^j$ et $puis3j = 3^j$.

```
pasde24(n):={
  local L,j,p,puis3j;
  L:=[0];
  j:=0;
  p:=1;
  puis3j:=1;
  tantque p<n faire
    L:=concat(L,L+[puis3j$p]);
    //L:=concat(L,map(L,x->x+puis3j));
    j:=j+1;
    puis3j:=3*puis3j;
    p:=2*p;
  ftantque;
  L:=mid(L,0,n);
  retourne L;
}
;;
```

À la fin de la boucle `tantque`, `L` a $2^j = p >= n$ éléments : il faut donc raccourcir la liste `L` (`L:=mid(L,0,n)`) et on calcule des termes pour rien...On modifie donc le programme :

```
pasde24b(n):={
  local L,j,p,puis3j;
  L:=[0];
  j:=0;
  p:=1;
  puis3j:=1;
  tantque 2*p<=n faire
    L:=concat(L,L+[puis3j$p]);
    j:=j+1;
    puis3j:=3*puis3j;
    p:=2*p;
  ftantque;
  L:=concat(L,mid(L,0,n-p)+[puis3j$(n-p)]);;
  retourne L;
}:;
```

Voici ce que l'on obtient :



Ce qui montre que le dernier algorithme est le meilleur...

8.14 Écriture d'un entier dans une base rationnelle

Soient deux entiers p, q, premiers entre eux tel que $q < p$. On veut écrire un entier n sous la forme :

$$n = \sum_{k=0}^{s-1} n_k (\frac{p}{q})^k \text{ avec } n_k < p$$

On définit la suite :
$u(0) = n, u(1) = iquo(u(0), p) * q, u(k+1) = iquo(u(k), p) * q$ u est une suite décroissante donc il existe s tel que $u(s) = 0$.
On a :
$u(0) = iquo(u(0, p) * p + irem(u(0), p) = u(1) * p/q + irem(u(0), p)$
$u(1) = iquo(u(1, p) * p + irem(u(1), p) = u(2) * p/q + irem(u(1), p)$
On pose $n_0 = irem(u(0), p)$
Donc :
$q^{s-1}u(0) = u(1) * p * q^{s-2} + q^{s-1}n_0$
et par itération :
$pq^{s-2}u(1) = u(2)p^2q^{s-3} + pq^{s-2}n_1$ avec $n_1 == irem(u(1), p)$
...
$q^{s-1}u(0) = \sum_{k=0}^{s-1} p^k q^{s-k-1} n_k$
ou encore :
$n = u(0) = \sum_{k=0}^{s-1} p^k q^{-k} n_k$
Cette écriture est unique : on raisonne par récurrencesur n.
Le développement est unique pour tous les $n < p$.
Si il y a unicité pour tous les entiers $m < n$ alors si on a 2 développements de n :
$n == \sum_{k=0}^{s-1} p^k q^{-k} a_k$ et $n == \sum_{k=0}^{s-1} p^k q^{-k} b_k$ puisque $n = a_0 + p * \sum_{k=1}^{s-1} p^{k-1} q^{-k} a_k = b_0 + p * \sum_{k=1}^{s-1} p^{k-1} q^{-k} b_k =$ on en déduit que $a_0 = b_0 = \texttt{irem}(n, p)$ et on applique l'hypothèse de récurrence à $\texttt{iquo}(n, p) * q$ qui est strictement inférieur à n.
On écrit le programme `dev` qui renvoie la liste de dimension s :
$[n_0, n_1..n_{s-1}]$ et le programme `verif` qui effectue la vérification.

```
dev(n,p,q):={
local L,s,u;
si gcd(p,q)!=1 ou q>p-1 alors return "erreur"; fsi;
L:=NULL;
si n<p alors return n; fsi;
s:=n;
tantque s>0 faire
u:=irem(s,p);
s:=iquo(s,p)*q;
L:=L,u;
}
return [L];
}
:;
verif(L,p,q):={
local n,s,k;
n:=L[0];
s:=size(L);
pour k de 1 jusque s-1 faire
n:=n+L[k]*(p/q)^k;
fpour;
return n;
}:;
```

On tape :
`L:=dev(33,3,2)` On obtient :
`[0,1,2,2,1,2]`
On tape :
`verif(L,3,2)` On obtient :
`33`
On tape :
`L:=dev(133,13,8)` On obtient :
`[3,2,9,11,8]`
On tape :
`verif(L,13,8)` On obtient :
`133`

8.15 Traduction Xcas de l'algorithme de Hörner

Soit un polynôme P donné sous la forme d'une liste `l` formée par les coefficients de P selon les puissances décroissantes.
`hornerl(l,a)` renvoie une liste formée par la valeur `val` du polynôme en $x = a$ et par la liste `lq` des coefficients selon les puissances décroissantes du quotient $Q(x)$ de $P(x)$ par $(x - a)$.
On a :
$\mathtt{P(a)} = \mathtt{l[0]} * \mathtt{a^p} + \mathtt{l[1]} * \mathtt{a^{p-1}} + ... + \mathtt{l[p]} =$
$\mathtt{l[p]} + \mathtt{a} * (\mathtt{l[p-1]} + \mathtt{a} * (\mathtt{l[p-2]} + ... + \mathtt{a} * (\mathtt{l[1]} + \mathtt{a} * \mathtt{l[0]})))$
$\mathtt{P(x)} = \mathtt{l[0]} * \mathtt{x^p} + \mathtt{l[1]} * \mathtt{x^{p-1}} + ... + \mathtt{l[p]} =$

$(\mathtt{x}-\mathtt{a})*(\mathtt{lq}[0]*\mathtt{x}^{\mathtt{p}-1}+...\mathtt{lq}[\mathtt{p}-1])+\mathtt{val}$
donc $\mathtt{val}=\mathtt{P}(\mathtt{a})$ et `p=s-1` si `s` est la longueur de la liste `l` donc :

```
lq[0]=l[0]
lq[1]=a*lq[0]+l[1]
lq[j]=a*lq[j-1]+l[j]
....
val=a*lq[p-1]+l[p]
```

```
hornerl(l,a):={
local s,val,lq,j;
s:=size(l);
//on traite les polys constants (de degre=0)
if (s==1) {return [l[0],[0]]};
// si s>1
lq:=[];
val:=0;
for (j:=0;j<s-1;j++) {
val:=val*a+l[j];
lq:=append(lq,val);
}
val:=val*a+l[s-1];
return([val,lq]);
};
```

On tape :

```
hornerl([1,2,4],12)
```

On obtient :

```
[172,[1,14]]
```

ce qui veut dire que :
$x^2+2x+4=(x+14)(x-12)+172$
Si le polynôme est donné avec son écriture habituelle.
Pour utiliser la fonction précédente on a alors besoin des deux fonctions :
`symb2poly` qui transforme un polynôme en la liste de ses coefficients selon les puissances décroissantes.
`poly2symb` qui transforme une liste en l'écriture habituelle du polynôme ayant cette pour coefficients selon les puissances décroissantes.

```
hornerp(p,a,x):={
//ne marche pas pour les polys constants (de degre=0)
local l,val,lh;
l:=symb2poly(p,x);
lh:=hornerl(l,a);
p:=poly2symb(lh[1],x);
val:=lh[0];
return([val,p]);
};
```

On tape :

```
hornerp(x^2+2x+4,12,x)
```

On obtient :

```
172,x+14
```
On tape :
```
hornerp(y^2+2y+4,12,y)
```
On obtient :
```
172,y+14
```
Dans `Xcas`, il existe la fonction `horner` qui calcule selon la méthode de Hörner la valeur d'un polynôme (donné sous forme de liste ou par son expression) en un point :
On tape :
```
horner(x^2+2x+4,12)
```
On obtient :
```
172
```
On tape :
```
horner(y^2+2y+4,12,y)
```
On obtient :
```
172
```
On tape :
```
horner([1,2,4],12)
```
On obtient :
```
172
```

8.15.1 Un autre exercice et sa solution

Trouver le plus petit entier positif n, tel que n, $2n$, $3n$, $4n$, $5n$, $6n$ contiennent exactement les mêmes chiffres.
On tape la fonction booléenne qui teste si les entiers n et m ont des chiffres identiques. On se sert de la fonction `string` qui transforme un entier en une chaine de caractères, puis on transforme cette chaine en la liste de ses caractères ou en son code de Ascii, puis on trie cette liste.
On tape :

```
chiffreid(n,m):={
local S1,S2,s1,s2,L1,L2,k;
  S1:=string(n);s1:=size(S1);
  S2:=string(m);s2:=size(S2);
  si s1!=s2 alors retourne faux; fsi;
  L1:=[sort(S1[k]$(k=0..s1-1))];
  L2:=[sort(S2[k]$(k=0..s1-1))];
  retourne L1==L2;
}:;
```

ou

```
chiffreid(n,m):={
local S1,S2,s1,s2,L1,L2,k;
  S1:=string(n);s1:=size(S1);
  S2:=string(m);s2:=size(S2);
  si s1!=s2 alors retourne faux; fsi;
  L1:=sort(asc(S1));
```

```
  L2:=sort(asc(S2));
  retourne L1==L2;
}:;
```

On tape la fonction booléenne qui teste si les entiers n, $2n$, $3n$, $4n$, $5n$, $6n$ ont des chiffres identiques. Si cela est le cas on sait que n est divisible par 3 puisque la somme des chiffres de n est égale la somme des chiffres de $3n$.
On tape :

```
chiffreid16(n):={
  local k;
  si irem(n,3)!=0 alors retourne faux; fsi;
  pour k de 6 jusque 2 pas -1 faire
  si chiffreid(n,k*n)==faux alors retourne faux fsi;
  fpour;
  retourne vrai;
  }:;
```

On tape la fonction qui renvoie le plus petit entier positif n, tel que n, $2n$, $3n$, $4n$, $5n$, $6n$ contiennent exactement les mêmes chiffres.
On tape :

```
ppchiffreid():={
local n;
n:=3;
tantque chiffreid16(n)==faux faire
  n:=n+3;
 ftantque;
retourne n;
  }:;
```

On tape :
`ppchiffreid()`
On obtient :
`142857`
On vérifie :
On tape :
`n:=142857;2*n;3*n;4*n;5*n;6*n`
On obtient ;
`142857,285714,428571,571428,714285,857142` On peut changer le programme ci-dessus pour savoir qui est le n suivant en initialisant n à 142860. On trouve alors 1428570.
Puis on change à nouveau le programme ci-dessus pour savoir qui est le n suivant en initialisant n à 1428573. On trouve alors 1429857.
Puis par curiosité, on cherche le suivant (mais c'est long !), on trouve 14285700.
On a donc le début de cette suite : `142857, 1428570, 1429857, 14285700, 14298570`

8.16 Savoir si le polynôme A est divisible par B

8.16.1 Programmation de la fonction booléenne `estdivpoly`

On va écrire la fonction récursive `estdivpoly` qui a comme arguments, deux polynômes `A` et `B` écrits sous forme symbolique et qui renverra `1` si `A` est divisible par `B` et `0` sinon.
On rappelle que `degree(A)` renvoie le degré de `A` et que `valuation(A)` renvoie la valuation de `A` (la plus petite puissance de `A`).

Pour Savoir si `A` est divisible par `B`, on s'interesse aux termes de plus haut degré et de plus bas degré de `A` et `B` : c'est à dire qu'a chaque étape on essaye de faire la division par les 2 bouts ....
Par exemple si :
`A=x^3+2*x-3` et `B=x^2+x` on sait que `A` n'est pas divisible par `B` car `-3` n'est pas divisible par `x`,
ou encore si :
`A=x^3+2*x^2` et `B=x^2+1` on sait que `A` n'est pas divisible par `B` car le quotient aurait pour degré `3-2=1` et pour valuation `2-0=2`, ce qui est impossible `1<2` (le degré n'peut pas être inférieur à la valuation.

```
estdivpoly(A,B):={
  local da,db,va,vb,dq,vq,dva,dvb,dvq,Q,Ca,Cb;
  da:=degree(A);
  va:=valuation(A);
  dva:=da-va;
  db:=degree(B);
  vb:=valuation(B);
  dvb:=db-vb;
  if (A==0) then return 1;end_if;
  if ((da<db) or (va<vb)) then return 0;end_if;
  if ((dva==0) and (dvb>0)) then return 0;end_if;
  if ((dva>=0) and (dvb==0)) then return 1;end_if;
  Cb:=coeffs(B);
  if ((dva>0) and (dvb>0)) then
  dq:=da-db;
  vq:=va-vb;
  dvq:=dq-vq;
  if (dvq<0) then return 0;end_if;
  Ca:=coeffs(A);
  Q:=Ca[0]/Cb[0]*x^(dq);
  if (dvq==0) then
  A:=normal(A-B*Q);
  else
  Q:=Q+Ca[dva]/Cb[dvb]*x^(vq);
  A:=normal(A-B*Q);
  end_if;
  da:=degree(A);
  va:=valuation(A);
```

```
   end_if;
  return estdivpoly(A,B);
};
```

On tape : `A:=normal((x^4-x^3+x^2-1)*(x^5-x^3+x^2-1)`
puis,
`estdivpoly(A,x^4-x^3+x^2-1)`
On obtient :
`1`

8.16.2 Autre version du programme précedent : `quoexpoly`

Lorsque `A` est divisible par `B` on peut en modifiant le programme précédent avoir facilement le quotient exact de `A` par `B`.
On écrit la fonction récursive `quoexpoly` qui a trois arguments, deux polynômes `A` et `B` écrits sous forme symbolique et `0`. `quoexpoly` renverra `1,Q` si `A=B*Q` et `0` sinon.
Puis on écrit la fonction `quopoly(A,B)` qui est égale à `quoexpoly(A,B,0)`.

```
quoexpoly(A,B,SQ):={
  local da,db,va,vb,dq,vq,dva,dvb,dvq,Q,Ca,Cb;
  da:=degree(A);
  va:=valuation(A);
  dva:=da-va;
  db:=degree(B);
  vb:=valuation(B);
  dvb:=db-vb;
  if (A==0) then return 1,SQ;end_if;
  if ((da<db) or (va<vb)) then return 0;end_if;
  if ((dva==0) and (dvb>0)) then return 0;end_if;
  if ((dva>=0) and (dvb==0)) then return 1,normal(SQ+normal(A/B));end_if;
  Cb:=coeffs(B);
  if ((dva>0) and (dvb>0)) then
  dq:=da-db;
  vq:=va-vb;
  dvq:=dq-vq;
  if (dvq<0) then return 0;end_if;
  Ca:=coeffs(A);
  Q:=Ca[0]/Cb[0]*x^(dq);
  if (dvq==0) then
  A:=normal(A-B*Q);
  SQ:=normal(SQ+Q);
  else
  Q:=Q+Ca[dva]/Cb[dvb]*x^(vq);
  A:=normal(A-B*Q);
   SQ:=normal(SQ+Q);
  end_if;
  da:=degree(A);
  va:=valuation(A);
```

```
   end_if;
  return quoexpoly(A,B,SQ);
};
estquopoly(A,B):=quoexpoly(A,B,0);
```

On tape : `A:=normal((x^4-x^3+x^2-1)*(x^5-x^3+x^2-1)`
puis,
`estquopoly(A,x^4-x^3+x^2-1)`
On obtient :
`1,x^5-x^3+x^2-1`

8.17 Affichage d'un nombre en une chaîne comprenant des espaces

8.17.1 Affichage d'un nombre entier par tranches de p chiffres

Pour rendre plus facile la lecture d'un grand nombre entier, on veut l'afficher par tranches, c'est à dire selon une chaîne de caractères constituées par les p premiers chiffres du nombre et d'un espace, puis les p suivants etc... On écrit le programme qui va afficher le nombre n par tranches de p chiffres :

```
affichen(n,p):={
local reste,result,s;
result:="";
while (n>10^p) {
//on transforme irem(n,10^p) en une chaine
reste:=cat(irem(n,10^p),"");
s:=size(reste);
//on ajoute l'espace et les zeros qui manquent
reste:=cat(" ",op(newList(p-s)),reste);
n:=iquo(n,10^p);
//on concatene reste avec result
result:=cat(reste,result);
}
reste:=cat(n);
return cat(reste,result);
};
```

On tape :
`affichen(1234567,3)` On obtient :
`"1 234 567"`

8.17.2 Transformation d'un affichage par tranches en un nombre entier

Pour avoir la transformation inverse, on va transformer une chaîne comportant des chiffres et un autre caractère (par exemple un espace) en un nombre entier.
On écrit le programme :

```
enleve(chn,ch):={
local l,s;
s:=length(chn)-1;
//on transforme chn en une liste de ces lettres
//puis, on enleve le caractere ch de cette liste
l:=remove(x->(ord(x)==ord(ch)),seq(chn[k],k,0,s));
//on transforme la liste en chaine
return expr(char(ord(l)));
};
```

On peut aussi remplacer la dernière ligne :
`return char(ord(l))`
(`ord(l)` transforme la liste de caractères en la liste de leurs codes ascii et `char` transforme la liste des codes ascii en une chaîne).
par :
`return cat(op(l))`
car `op(l)` transforme la liste en une séquence et `cat` concatène les éléments de cette séquence en une chaîne. On tape :
`enleve("1 234 567"," ")` On obtient :
`1234567`

8.17.3 Affichage d'un nombre décimal de [0,1[par tranches de p chiffres

Pour rendre plus facile la lecture d'un nombre décimal de [0,1[, on veut l'afficher par tranches, c'est à dire selon une chaîne de caractères constituées par les p premières décimales du nombre et d'un espace, puis les p suivants etc... On suppose que l'écriture de d comporte un point (.) suivi des décimales et ne comporte pas d'exposant (pas de $e4$)

On écrit le programme qui va afficher le nombre d par tranches de p chiffres :

```
affiched(d,p):={
local deb,result;
//on suppose 0<=d<1
d:=cat(d,"");
if (d[0]=="0") {d:=tail(d);}
if (expr(tail(d))<10^p){return d;}
deb:=mid(d,0,p+1);
result:=cat(deb," ");
d:=mid(d,p+1);
while (expr(d)>10^p) {
deb:=mid(d,0,p);
result:=cat(result,deb," ");
d:=mid(d,p);
}
return cat(result,d);
};
```

On tape :
`affiched(0.1234567,3)`

On obtient :
".123 456 7"
Remarque
La commande `enleve(affiched(d,3)," ")` permet encore de retrouver `d`.

```
enleve(chn,ch):={
local l,s;
s:=length(chn)-1;
//on transforme chn en une liste de ces lettres
//puis, on enleve le caractere ch de cette liste
l:=remove(x->(ord(x)==ord(ch)),seq(chn[k],k,0,s));
//on transforme la liste en chaine
return expr(char(ord(l)));
};
```

8.17.4 Affichage d'un nombre décimal par tranches de p chiffres

Pour rendre plus facile la lecture d'un nombre décimal, on veut l'afficher par tranches, c'est à dire selon une chaîne de caractères constituées par sa partie entière écrite par tranches de p chiffres, puis ses p premières décimales du nombre et d'un espace, puis les p suivants etc...
Ici, le nombre f peut comporter un exposant à la fin de son écriture.
On écrit le programme qui va afficher le nombre décimal f par tranches de p chiffres :

```
//pour les flottants f utiliser affichef
// appelle affichen et affiched
//par exemple affichef(1234.12345,3)
affichef(f,p):={
local deb,result,s,indicep,fn,fd,indicee;
//on suppose f>1
f:=cat(f);
s:=size(f)-1;
indicep:=member(".",seq(f[k],k,0,s));
indicee:=member("e",seq(f[k],k,0,s));
if (indicep!=0) {
fn:=mid(f,0,indicep-1);
fd:=mid(f,indicep-1);
if (indicee!=0) {
return affichen(expr(fn),p)+affiched(expr(mid(fd,0,
indicee-1)),p)+mid(fd,indicee-1);}
return affichen(expr(fn),p)+affiched(expr(fd),p)
}
return affichen(expr(f),p);
};
```

On tape :
`affichef(1234567.1234567,3)`
On obtient (pour 12 chiffres significatifs) :

```
"1 234 567.123 46"
```
On obtient (pour 14 chiffres significatifs) :
```
"1 234 567.123 456 7"
```
On obtient (pour 15 chiffres significatifs) :
```
"0.123 456 712 345 670 0*e7"
```
Remarque
La commande `enleve(affichef(q,3)," ")` permet encore de retrouver `q`.

```
enleve(chn,ch):={
local l,s;
s:=length(chn)-1;
//on transforme chn en une liste de ces lettres
//puis, on enleve le caractere ch de cette liste
l:=remove(x->(ord(x)==ord(ch)),seq(chn[k],k,0,s));
//on transforme la liste en chaine
return expr(char(ord(l)));
};
```

8.18 Écriture décimale d'un nombre rationnel

8.18.1 Algorithme de la potence

Pour obtenir la partie entière et le développement décimal de $\frac{a}{b}$, on va construire deux listes : `L1` la liste des restes et `L2` la liste des quotients obtenus par l'algorithme de la potence .
On met le quotient q dans `L1` et le reste r dans `L2`.
On a ainsi, la partie entière de $\frac{a}{b}$ dans `L1` et comme $\frac{a}{b} = q + \frac{r}{b}$ on cherche la partie entière de $\frac{10 * r}{b}$ qui va rallonger `L1` etc...

Si on veut, par exemple, le développement décimale de $\frac{278}{31}$ on cherche :
le quotient $q = 8$ et le reste $r = 30$ de la divison euclidienne de 278 par 31.
La partie entière est donc 8 et, on met 8`L1` . Pour avoir la partie décimale de $\frac{278}{31}$, on fait comme à la main l'algorithme de la potence : on multiplie le reste trouvé par 10, on trouve 300 puis on le divise par 31 : le quotient trouvé 9 est rajouté à `L1` et le reste est rajouté à `L2` etc...
On écrit la fonction potence qui renvoie dans la première liste la partie entière puis les n décimales de $\frac{a}{b}$ et dans la deuxième liste les restes successifs obtenus.

```
potence(a,b,n):={
 local L1,L2,k;
 b0:=b;
 b:=iquo(a,b0);
 a:=irem(a,b0);
 L1:=[b];
 L2:=[a];
 for (k:=1;k<=n and a!=0;k++){
    b:=iquo(a*10,b0);
```

```
    a:=irem(a*10,b0);
    L2:=append(L2,a);
    L1:=append(L1,b);
 };
 return([L1,L2]);
};
```

En exécutant `potence(278,31,20)`, on lit la partie entière de $\frac{278}{31}$ et les chiffres de sa partie décimale dans la première liste et, la suite des restes dans la deuxième liste.

Exercice

Écrire la partie entière et le développement décimal de :
$a = \frac{11}{7}$, $b = \frac{15}{14}$ et $c = \frac{17}{28}$.
Calculer $a - b$ et $a - c$ et donner leur partie entière et leur développement décimal.
Que remarqez-vous ?

Exercice Comment modifier `L1` et `L2` pour que les chiffres de la partie décimale de $\frac{a}{b}$ se lisent par paquet de trois chiffres dans `L1`.
Avec l'exemple $\frac{278}{31}$ on veut obtenir : `L1=[8,967,741,935 ...]`
Tester votre modification pour $\frac{349}{1332}$.
Que remarquez vous ?

8.18.2 Avec un programme

`division(a,b,n,t)` donne la partie entière suivie de n paquets de t décimales (i.e. des $n * t$ premières décimales) de $\frac{a}{b}$.

```
division(a,b,n,t):={
local L1,L2,p,q,r,k;
L1:=[iquo(a,b)];
r:=irem(a,b);
for (k:=1;k<=n and r!=0;k++) {
q:=iquo(r*10^t,b);
//10^(p-1)<= q <10^p
if (q==0) {p:=1} else {p:=floor(ln(q)/ln(10)+1)};
//on complete par des zeros pour avoir un paquet de t decimales
for (j:=p+1;j<=t;j++){
L1:=append(L1,0);
}
L1:=append(L1,q);
r:=irem(r*10^t,b);
}
return(L1,r);
};
```

On tape pour avoir 5*6=30 decimales :
`division(2669,201,6,5)`

On obtient :
`[13,27860,69651,74129,35323,38308,45771],29`

8.18.3 Construction d'un rationnel

Trouver un nombre rationnel qui s'écrit :
0.123123123123... se terminant par une suite illimitée de 123.
Trouver un nombre rationnel qui s'écrit :
0.120123123123... se terminant par une suite illimitée de 123.
Écrire un programme qui permet de trouver un nombre rationnel à partir d'un développement décimal périodique.
Réponse :
On écrit la fonction `rationnel` qui a comme le paramètre deux listes `l1` et `l2` :
- `l1` désigne la partie non périodique de ce développement et `l1[0]` désigne la partie entière.
- `l2` représente un développement décimal périodique.

```
rationnel(l1,l2):={
//l1 et l2 sont non vides
local pui,s1,s2,n,p,np,pui,k;
pui:=10;
s2:=size(l2);
n:=l2[0];
for (k:=1;k<s2;k++){
pui:=pui*10;
n:=n*10+l2[k];
}
// 0.123123...=123/999
p:=n/(pui-1);
//np partie non periodique
np:=l1[0];
s1:=size(l1);
pui:=1;
for (k:=1;k<s1;k++){
pui:=pui*10;
np:=np+l1[k]/pui;
}
//pui=10^(s1-1)
return(np+p/pui);
};
```

8.19 Développement en fraction continue

8.19.1 Développement en fraction continue d'un rationnel

Les définitions

Théorème1 Si a et b sont des entiers naturels premiers entre eux, alors il existe des entiers naturels $a_0, a_1, ..., a_n$ $(0 \leq n)$ tels que :

$$\frac{a}{b} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{...a_{n-2} + \frac{1}{a_{n-1} + \frac{1}{a_n}}}}}$$

Si $b \leq a$ les aj sont non nuls et, si $a < b$ alors $a_0 = 0$ et les autres a_j sont non nuls.

Définition On pose alors $\frac{a}{b} = (a_0, a1, ...a_n)$ et on dit que $(a_0, a1, ...a_n)$ est une fraction continue : c'est le développement en fraction continue de $\frac{a}{b}$.

Remarque si $b \leq a$ et si $\frac{a}{b} = (a_0, a1, ...a_n)$ alors $\frac{b}{a} = (0, a_0, a1, ...a_n)$.

Réduite et reste On dit que la fraction $\frac{P_p}{Q_p}$ égale à la fraction continue $(a_0, a1, ...a_p)$, où $p \leq n$, est la réduite de rang p de $\frac{a}{b}$ ou que c'est le développement en fraction continue d'ordre p de $\frac{a}{b}$.

On dit que $r = (0, a_{p+1}, .., a_n)$ est le reste du développement d'ordre p $(r < 1)$ et on a $\frac{a}{b} = (a_0, a1, ...a_p + r) = (a_0, a1, ...a_p, 1/r)$,

$$\frac{a}{b} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{...a_{p-3} + \frac{1}{a_{p-2} + \frac{1}{a_p + r}}}}}.$$

Propriétés des réduites

Si $\frac{P_p}{Q_p}$ égale à la fraction continue $(a_0, a1, ...a_p)$, où $p \leq n$, est la réduite de rang p de $\frac{a}{b} = (a_0, a1, ...a_n)$, on a :

$P_0 = a_0$

$Q_0 = 1$

$P_1 = a_0 * a_1 + 1$

$Q_1 = a_1$

$P_p = P_{p-1} * a_p + P_{p-2}$

$Q_p = Q_{p-1} * a_p + Q_{p-2}$

En effet on le montre par récurrence :

$P_2/Q_2 = a_0 + a_2/(a_1 a_2 + 1)$ donc

$P_2 = a_2(a_0 + a_1 + 1) + a_0 = a_2 P_1 + P_0$ et

$Q_2 = a_2 a_1 + 1 = a_2 Q_1 + Q_0$

$(a_0, a_1, ..., a_p + 1/a_{p+1}) = \frac{P_{p+1}}{Q_{p+1}}$ donc

$P_{p+1}/Q_{p+1} = ((a_p + 1/a_{p+1})P_{p-1} + P_{p-2})/((a_p + 1/a_{p+1})Q_{p-1} + Q_{p-2})$

$P_{p+1} = a_{p+1}(a_p P_{p-1} + P_{p-2}) + P_{p-1} = a_{p+1} P_p + P_{p-1}$ et

$Q_{p+1} = a_{p+1}(a_p Q_{p-1} + Q_{p-2}) + Q_{p-1} = a_{p+1} Q_p + Q_{p-1}$

Les programmes

Le programme f2dfc :
On veut transformer une fraction en son développement en fraction continue :
`f2dfc(a/b)` $= (\mathtt{a}_0, \mathtt{a}_1, ...\mathtt{a}_n)$.
Pour obtenir le développement en fraction continue de a/b, on cherche le quotient q et le reste r de la division euclidienne de a par b. On a : $q = a_0$ et $a/b = a_0 + r/b = a_0 + 1/(b/r)$ et, on continue en cherchant la partie entiére de b/r qui sera a_1....On reconnait l'algorithme d'Euclide : la suite $(a_0, a_1, ...a_n)$ est donc la suite des quotients de l 'algorithme d'Euclide.
On écrit le programme :

```
f2dfc(fract):={
local r,q,l,lres,a,b;
l:=f2nd(fract);
a:=l[0];
b:=l[1];
lres:=[];
while (b>0) {
q:=iquo(a,b)
lres:=concat(lres,q);
r:=irem(a,b);
a:=b;
b:=r:
}
return lres;
}
```

On tape :
`f2dfc(2599/357)`
On obtient :
`[7,3,1,1,3,14]`
Le programme f2reduites d'un rationnel et l'identité de Bézout : On veut obtenir la suite des réduites de a/b.
L'algorithme pour obtenir les réduites ressemble beaucoup à l'algorithme utilisé pour obtenir les coefficients u et v de l'identité de Bézout (cf 8.3.5).
En effet on a :
$P_0 = a_0 = a_0 * 1 + 0$ alors que $v_0 = 0$
$Q_0 = 1 = a_0 * 0 + 1$ alors que $u_0 = 1$
$P_1 = a_0a_1 + 1 = P_0 * a_1 + 1$ alors que $v_1 = 1$
$Q_1 = a_1 = a_1 * Q_0 + 0$ alors que $u_1 = 0$
$P_p = P_{p-1} * a_p + P_{p-2}$ alors que $v_p = v_{p-2} - a_{p-2}v_{p-1}$
$Q_p = Q_{p-1} * a_p + Q_{p-2}$ alors que $u_p = u_{p-2} - a_{p-2}u_{p-1}$
Ainsi :
$P_0 = 0 + a_0 * 1 = v_0 - a_0 * v_1 = -v_2$
$P_1 = 1 + P_0 * a_1 = v_1 - v_2 * a_1 = v_3$
$P_2 = P_0 + P_1 * a_2 = -v_2 + v_3 * a_2 = -(v_2 - v_3 * a_2) = -v_4$
On a donc pour tout $p \geq 0$, si a_p est la suite des quotients de l'algorithme d'Euclide :

$Q_p = Q_{p-1} * a_p + Q_{p-2}$ avec $Q_{-2} = 1 = u_0$ et $Q_{-1} = 0 = u_1$ et,
$P_p = P_{p-1} * a_p + P_{p-2}$ avec $P_{-2} = 0 = v_0$ et $P_{-1} = 1 = v_1$
Dans l'algorithme de Bézout on a :
$u_p = -u_{p-1} * a_{p-2} + u_{p-2}$ et $v_p = -v_{p-1} * a_{p-2} + v_{p-2}$ $Q_0 = 0 + u_2$ donc $Q_1 = -u_3$, $Q_2 = u_4$ etc...donc $Q_n = (-1)^n u_{n+2}$ et,
$P_0 = -v_2 + 0$ donc $P_1 = v_3$, $P_2 = -v_4$ etc...donc $P_n = (-1)^{n+1} v_{n+2}$.
Donc $P_n/Q_n = -v_{n+2}/u_{n+2}$
Donc la suite Q_j/P_j est donc la suite des $-u/v$.
On écrit le programme (calqué sur le programme `Bezout` avec les listes) qui transforme une fraction en son développement en fraction continue suivi de la suite de ces réduites :

```
f2reduites(fract):={
local lr,q,l,lres,la,lb,lq;
l:=f2nd(fract);
//a:=l[0];b:=l[1];
la:=[1,0,l[0]];
lb:=[0,1,l[1]];
lq:=[];
lres:=[];
while (lb[2]>0) {
q:=iquo(la[2],lb[2])
lr:=la-q*lb;
lq:=concat(lq,q);
lres:=concat(lres,-lr[1]/lr[0]);
la:=lb;
lb:=lr;
}
return lq,lres;
}
```

On tape :
`f2reduites(2599/357)`
On obtient :
`[7,3,1,1,3,14],[7,22/3,29/4,51/7,182/25,2599/357]`
Remarque :
On ne peut pas remplacer :
`lr:=la-q*lb;`
`lres:=concat(lres,-lr[1]/lr[0])`
par :
`lr:=la+q*lb;`
`lres:=concat(lres,lr[1]/lr[0]);`
car alors `lr` n'est plus la suite des restes !
Le programme dfc2reduites d'un rationnel et l'identité de Bézout : On veut obtenir la suite des réduites d'une liste l (qui sera par exemple le développement en fraction continue d'un rationnel a/b) On écrit le programme (calqué sur le programme `Bezout` sans les listes) qui transforme une liste $[a_0, a_1, ..a_n]$ en la liste $[a_0, a_0 + 1/a_1, ..., (a_0 + 1/a_1 + 1/... + 1/a_{n-1} + 1/a_n)]$:

```
dfc2reduites(l):={
```

```
local s,p0,q0,p1,q1,p,q,lres,j;
s:=size(l);
lres:=[];
p0:=0;
p1:=1;
q0:=1;
q1:=0;
for (j:=0;j<s;j++){
  p:=p0+l[j]*p1;
  q:=q0+l[j]*q1;
  lres:=concat(lres,p/q);
  p0:=p1;
  q0:=q1;
  p1:=p;
  q1:=q;
}
return lres;
}
```

On remarquera que :
-la suite des P est initialisée par $p0$ et $p1$, puis, quand $j = 0$, on fait le calcul de P_0 qui est mis dans p, puis, quand $j = 1$ on fait le calcul de P_1 qui est mis dans p, etc... et que
- la suite des Q est initialisée par : $q0$ et $q1$, puis, quand $j = 0$ on fait le calcul de Q_0 qui est mis dans q, quand $j = 1$, on fait le calcul de Q_1 est mis dans q, etc...
On tape :
`dfc2reduites([7,3,1,1,3,14])`
On obtient :
`[7,22/3,29/4,51/7,182/25,2599/357]`

8.19.2 Développement en fraction continue d'un réel quelconque

Théorème2 Si α est un nombre réel non rationnel, alors il existe des entiers naturels non nuls $a_0, a_1, ..., a_n$ et un réel $\beta < 1$ tels que :

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\cdots a_{n-2} + \cfrac{1}{a_{n-1} + \cfrac{1}{a_n + \beta}}}}}$$

On dit que $(a_0, a1, ...a_n)$ est le développement en fraction continue d'ordre $n + 1$ de α et que β est le reste de ce développement.
Un rationnel a un développement en fraction continue fini et réciproquement, un développement en fraction continue fini représente un rationnel.
Un réel non rationnel a un développement en fraction continue infini.
Si α est un nombre quadratique (i.e. α est racine d'une équation du second degré), α a un développement en fraction continue périodique et réciproquement, un développement en fraction continue périodique représente un nombre quadratique.

8.19.3 Les programmes

On va écrire deux fonctions `r2dfc` et `dfc2r`.

La fonction r2dfc

`r2dfc(alpha,n)` qui transforme un réel `alpha` en son développement en fraction continue et qui renvoie deux listes, soit :
- $[a_0, a_1 ... a_p], []$ avec $p \leq n$ où les a_j sont des entiers, la deuxième liste est vide et la première liste est le développement en fraction continue de `alpha` (les a_j sont des entiers et donc `alpha` est une fraction)
- $[a_0, a_1 ... a_{n-1}, b], []$, la deuxième liste est vide et la première liste est le développement en fraction continue d'ordre $n-1$ de `alpha` suivi de $b > 1$ (le reste est égal à $1/b$), où les a_j sont des entiers et b est un réel plus grand que 1, soit,
- $[a_0, a_1 ... a_p], [a_r, .. a_p]$ avec $r \leq p < n$

où les a_j sont des entiers, la première liste est le développement en fraction continue d'ordre p de `alpha` et la deuxième liste représente la période de ce développement en fraction continue (les a_j sont des entiers et donc `alpha` est un nombre quadratique)
. On remarquera dans le programme ci-dessous que :
$a_0 = \texttt{floor}(\texttt{alpha}) = q$ remplace $q := \texttt{iquo}(a, b)$ lorsque `alpha=a/b`
et que `r=alpha-q` remplace $\texttt{irem}(a, b)/b$ lorsque `alpha=a/b`
et donc que si `r=alpha-q`, $a_1 = \texttt{floor}(1/r)$ etc...
Le problème ici est de pouvoir comparer `alpha` et `q` c'est à dire savoir si `r==0` et pour cela on est obligé de faire les calculs avec beaucoup de décimales c'est à dire d'augmenter le nombre de digits (on tape par exemple `DIGITS:=30`). Il faut bien sûr repérer la période, pour cela on forme la liste `lr` des restes successifs. La liste `lq` des parties entières successives forme le début du développement en fraction continue.

```
r2dfc(alpha,n):={
local r,q,lq,lr,p,j;
q:=floor(alpha);
r:=normal(alpha-q);
lq:=[];
lr:=[];
for (j:=1;j<=n;j:=j+1) {
lq:=concat(lq,q);
if (r==0){return (lq,[]);}
p:=member(r,lr);
if (p) {return (lq,mid(lq,p));}
lr:=concat(lr,r);
alpha:=normal(1/r);
q:=floor(alpha);
r:=normal(alpha-q);
};
return (concat(lq,alpha),[]);
};
```

On tape :
`r2dfc(sqrt(2),1)`
On obtient :
`([1,sqrt(2)+1],[])`
On tape :
`r2dfc(sqrt(2),2)`
On obtient :
`([1,2],[2])`
On tape :
`r2dfc(pi,6)`
On obtient :
`([3,7,15,1,292,1,(-33102*pi+103993)/(33215*pi-104348)],[])`
`r2dfc(sqrt(11212),90)`
On obtient :
```
[[105,1,7,1,4,1,5,19,12,2,2,7,2,3,1,2,6,17,2,25,1,69,1,1,1,2,3,1,
1,1,1,1,2,1,5,1,2,3,1,4,23,3,8,2,52,2,8,3,23,4,1,3,2,1,5,1,2,1,1,
1,1,1,3,2,1,1,1,69,1,25,2,17,6,2,1,3,2,7,2,2,12,19,5,1,4,1,7,1,210],
[1,7,1,4,1,5,19,12,2,2,7,2,3,1,2,6,17,2,25,1,69,1,1,1,2,3,1,
1,1,1,1,2,1,5,1,2,3,1,4,23,3,8,2,52,2,8,3,23,4,1,3,2,1,5,1,2,1,1,
1,1,1,3,2,1,1,1,69,1,25,2,17,6,2,1,3,2,7,2,2,12,19,5,1,4,1,7,1,210]]
```

Remarque Le premier argument doit être un nombre exact, car sinon les calculs sont faits en mode approché et le test r==0 n'est jamais réalisé... **Ou bien** :
On tape le programme `devfracont` qui renvoie une seule liste :
$[a_0, a_1 ... a_{n-1}, b], []$
sans mettre en évidence la période :

```
devfracont(alpha,n):={
  local j,q,r,lq,lr;
  lq:=[];
  lr:=[];
  q:=floor(alpha);
  r:=normal(alpha-q);
  for(j:=1;j<=n;j++){
    lq:=append(lq,q);
    if (r==0){return(lq,[]);};
    lr:=append(lr,r);
    alpha:=simplify(1/r);
    q:=floor(alpha);
    r:=normal(alpha-q);
    };
    lq:=append(lq,alpha);
    return lq;
  }:;
```

On tape :
`devfracont(sqrt(2),1)`
On obtient :

```
[1,1+sqrt(2)]
```
On tape :
```
devfracont(sqrt(2),2)
```
On obtient :
```
[1,2,1+sqrt(2)]
```
et si on tape on a bien :
```
simplify(1+1/(2+1/(1+sqrt(2))))]
```
on obtient bien :
```
sqrt(2)
```
On tape :
```
devfracont(pi,6)
```
On obtient :
```
([3,7,15,1,292,1,(103993-33102*pi)/(-104348+33215*pi)])
```
et si on tape on a bien :
```
simplify(3+1/(7+1/(15+1/(1+1/(292+
1/((355-113*pi)/(-103993+33102*pi))))))
```
on obtient bien :
```
pi
```

La fonction dfc2r

On écrit la fonction réciproque de `r2dfc` qui à partir d'un développement en fraction continue et d'un reste éventuel ou d'un développement en fraction continue et d'une période éventuelle renvoie un réel.
`dfc2r(d,t)` transforme en un réel, la liste `d` représente un développement en fraction continue et la liste `t` représente la période.
On remarquera que lorsque la liste `t` n'est pas vide il faut déterminer le nombre `0<y<1` qui admet cette liste périodique comme développement en fraction continue et pour ce faire résoudre l'équation :
$y = (0, t_0, ...t_{st-1} + y)$ le reste est alors $y + d_{s-1}$ (`s:=size(d)`).
On écrit le programme :

```
dfc2r(d,t):={
local s,st,alpha,l,ap,k;
s:=size(d);
alpha:=d[s-1];
for (k:=s-2;k>=0;k:=k-1) {alpha:=normal(d[k]+1/alpha);}
if (t==[]) {return normal(alpha);}
st:=size(t);
purge(y);
ap:=t[st-1]+y;
for (k:=st-2;k>=0;k:=k-1) {ap:=normal(t[k]+1/ap);}
l:=solve(y=1/ap,y);
if (l[0]>0){y:=normal(l[0]);}else{y:=normal(l[1]);};
alpha:=d[s-1]+y;
for (k:=s-2;k>=0;k:=k-1) {alpha:=normal(d[k]+1/alpha);}
return(normal(a)lpha);
};
```

ou avec une écriture plus concise :

```
dfc2r(d,t):={
local s,st,alpha,l,ap,k;
s:=size(d);
st:=size(t);
if (st==0)
  {y:=0;}
   else
 {purge(y);
  ap:=t[st-1]+y;
  for (k:=st-2;k>=0;k:=k-1) {ap:=normal(t[k]+1/ap);}
  l:=solve(y=1/ap,y);
  if (l[0]>0){y:=normal(l[0]);}else{y:=normal(l[1]);};
  }
alpha:=d[s-1]+y;
for (k:=s-2;k>=0;k:=k-1) {alpha:=normal(d[k]+1/alpha);}
return(normal(alpha));
};
```

8.19.4 Exemples

1/ Développement en fraction continue de : $\frac{1393}{972}$, $1+\sqrt{13}$ et $1-\sqrt{13}$.
On a :
`r2dfc(1393/972,3)=[1,2,3,130/31],[]`
`r2dfc(1393/972,7)=[1,2,3,4,5,6],[]`
et on a bien :
`r2dfc(130/31,3)=[4,5,6],[]`
`r2dfc(31/130,4)=[0,4,5,6],[]`
On peut vérifier que :
`dfc2r([1,2,3,4,5,6],[])=1393/972`
`dfc2r([1,2,3+31/130],[])=dfc2r([1,2,3,130/31],[])=1393/972`
On a :
`r2dfc(1+sqrt(13),3)=[4,1,1,(sqrt(13)+2)/3],[]`
`r2dfc(1+sqrt(13),6)=[4,1,1,1,1,6],[1,1,1,1,6]`
`r2dfc(1-sqrt(13),7)=[-3,2,1,1,6,1,1],[1,1,6,1,1]`

2/ Trouver les réels qui ont comme développement en fraction continue :
`[2,4,4,4,4,4....]` (suite illimitée de `4`) et
`[1,1,1,1,1,1....]` (suite illimitée de `1`).
On a :
`dfc2r([2,4],[4])=sqrt(5)` ou encore `dfc2r([2],[4])=sqrt(5)` On a :
`dfc2r([1],[1])=(sqrt(5)+1)/2`

8.19.5 Suite des réduites successives d'un réel

Si `alpha` a comme développement en fraction continue $(a_0, a_1, ...a_n....)$, la suite des réduites est la suite des nombres rationnels ayant comme développement en fraction continue : $(a_0), (a_0, a_1), .., (a_0, a_1....a_n), ...$
On écrit le programme permettant d'obtenir les `p` premières réduites de `alpha`.

On écrit le programme `reduiten` (on recalcule les réduites sans se servir des relations de récurrence) :

```
reduiten(alpha,p):={
local l,k,ld,lt,st,s,q,lred,redu;
ld:=r2dfc(alpha,p);
l:=ld[0];
s:= size(l);
if (s<p) {
  lt:=ld[1];
  st:=size(lt);
  if (st!=0){
    q:=iquo(p-s,st);
    for (j:=0;j<=q; j++){
      l:= concat(l,lt)
    }
  }
  else {
  p:=s;
  }
}
lred:=[];
for (k:=1;k<=p;k++){
  redu:=dfc2r(mid(l,0,k),[]);
  lred:=append(lred,redu);
}
return (lred);
};
```

```
reduiten(sqrt(53),5)
```

On obtient :

```
[7,22/3,29/4,51/7,182/25]
```

On écrit maintenant le programme `reduite` permettant d'obtenir les `p` premières réduites de `alpha`, en se servant de la fonction `dfc2reduites` écrite auparavant et qui utilise les relations de récurrence.

```
reduite(alpha,p):={
local l,ld,lt,st,s,q,lred;
ld:=r2dfc(alpha,p);
l:=ld[0];
s:= size(l);
if (s<p) {
  lt:=ld[1];
  st:=size(lt);
  if (st!=0){
    q:=iquo(p-s,st);
    for (j:=0;j<=q; j++){
      l:= concat(l,lt)
```

```
    }
  }
}
l:= mid(l,0,p);
lred:=dfc2reduites(l);
return lred;
}
```

On tape :
`reduite(sqrt(53),5)`
On obtient :
`[7,22/3,29/4,51/7,182/25]`
On tape :
`reduite(11/3,2)`
On obtient :
`[3,4]`

8.19.6 Suite des réduites "plus 1" successives d'un réel

Si `alpha` a comme développement en fraction continue $(a_0, a_1, ...a_n....)$, la suite des réduites "plus 1" est la suite des nombres rationnels ayant comme développement en fraction continue : $(a_0 + 1), (a_0, a_1 + 1), .., (a_0, a_1....a_n + 1), ...$
On écrit le programme permettant d'obtenir les `p` premières réduites "plus 1" de `alpha`.

```
reduite1(alpha,p):={
local l,ld,lt,st,s,q,lred;
ld:=r2dfc(alpha,p);
l:=ld[0];
s:= size(l);
if (s<p) {
  lt:=ld[1];
  st:=size(lt);
  if (st!=0){
    q:=iquo(p-s,st);
    for (j:=0;j<=q; j++){
    l:= concat(l,lt)
    }
  }
}
l:= mid(l,0,p)+1;
lred:=dfc2reduites(l);
return lred;
}
```

8.19.7 Propriété des réduites

Propriété des réduites de `alpha` :
Une réduite `p/q` approche `alpha` à moins de $1/q^2$ et si `s/t` est la réduite plus 1

de même rang n on a :
- $|p/q - s/t| < 1/q^2$
- si n est pair $p/q \leq$ alpha $\leq r/s$
- si n est impair $r/s \leq$ alpha $\leq p/q$
- les réduites de rang pair et les réduites de rang impair forment deux suites adjacentes qui convergent vers alpha
- les réduites plus 1 de rang pair et les réduites plus 1 de rang impair forment deux suites adjacentes qui convergent vers alpha

Donc, si on pose :
lred:=reduite(alpha,10) et lred1:=reduite1(alpha,10), ces deux suites lred et lred1 fournissent un encadrement de alpha plus précisément on a :
lred[0] $\leq$ lred1[1] $\leq .. \leq$ lred[2p] $\leq$ lred1[2p+1]<alpha
alpha<lred[2p+1] $\leq$ lred1[2p] $\leq ... \leq$ lred[1] $\leq$ lred1[0]
c'est à dire que l'encadrement fait avec 2 réduites successives de rang $p-1$ et p est moins bon que l'encadrement fait avec la réduite de rang p et la réduite plus 1 de rang p.

Exemple
On a :
```
r2dfc(sqrt(53),5)= [7,3,1,1,3,sqrt(53)+7],[]
dfc2r([7,3,1,1,3],[])=182/25
reduite(sqrt(53),5)[4]=182/25=7.28
reduite1(sqrt(53),5)[4]=233/32=7.28125
reduite(182/25,5)[4]=182/25=7.28
reduite1(182/25,5)[4]=233/32=7.28125
```
et donc $7.28 < \sqrt{53} < 7.28125$
```
r2dfc(sqrt(53),6)= [7,3,1,1,3,14],[3,1,1,3,14]
dfc2r([7,3,1,1,3,14],[])=2599/357
reduite(sqrt(53),6)[5]=2599/357=7.28011204482
reduite1(sqrt(53),6)[5]=2781/382=7.28010471204
reduite(2599/357,5)[4]=2599/357=7.28011204482
reduite1(2599/357,5)[4]=2781/382=7.28010471204
```
et donc $7.28010471204 < \sqrt{53} < 7.28011204482$
On a $1/357^2 = 7.84627576521e-06$ et $1/382^2 = 6.8528823223e-06$

8.20 Suite de Hamming

8.20.1 La définition

La suite de Hamming est la suite des nombres entiers qui n'ont pour diviseurs premiers que 2, 3 et 5.
Cette suite commence par : [2,3,4,5,6,8,9,10,12,15,16,18,20,24,25...]

8.20.2 L'algorithme à l'aide d'un crible

On écrit tous les nombres de Hamming de 0 à n>0 et on barre les nombres qui sont de la forme $2^a * 3^b * 5^c$ avec a, b, c variant de 0 à un nombre tel que

$2^a * 3^b * 5^c \leq n$: les nombres barrés (excepté 1) sont les nombres de Hamming inférieurs à `n>0`.
Voici la fonction `hamming(n)` écrite en `Xcas` pour obtenir les nombres de Hamming inférieurs à `n>0`.

```
hamming(n):={
  local H,L,a,b,c,j,d;
  L:=makelist(x->x,0,n);
  //les nbres de Hamming sont 2^a*3^b*5^c
  c:=0; b:=0;a:=0;
  d:=1;
  while (d<=n) {
    while (d<=n){
      while (d<=n) {
L[d]:=0;
//d:=5*d
c:=c+1;
d:=2^a*3^b*5^c;
      }
      c:=0;
      b:=b+1;
      //d:=2^a*3^b*5^c
      d:=2^a*3^b;
    }
    //c:=0;
    b:=0;
    a:=a+1;
    //d:=2^a*3^b*5^c
    d:=2^a;
  }
  H:=[];
  for (j:=2;j<=n;j++) {
    if (L[j]==0) H:=append(H,j);
  }
  return H;
}
```

ou encore en supprimant la variable `c` :

```
hamming(n):={
  local H,L,a,b,j,d;
  L:=makelist(x->x,0,n);
  //les nbres de Hamming sont 2^a*3^b*5^c
 a:=0;
  d:=1;
  while (d<=n) {
    b:=0;
    while (d<=n){
      while (d<=n) {
```

```
L[d]:=0;
d:=5*d;
      }
      b:=b+1;
      d:=2^a*3^b;
    }
    a:=a+1;
    d:=2^a;
  }
  H:=[];
  for (j:=2;j<=n;j++) {
    if (L[j]==0) H:=append(H,j);
  }
  return H;
}
```

ou encore en supprimant a,b,c et en preservant la valeur de d avant les while :

```
hamming(n):={
  local H,L,d,j,k;
  L:=makelist(x->x,0,n);
  //les nbres de Hamming sont 2^a*3^b*5^c
  d:=1;
  while (d<=n) {
    j:=d;
    while (j<=n){
      k:=j;
      while (k<=n) {
L[k]:=0;
k:=5*k;
      }
      j:=3*j;
    }
    d:=2*d;
  }
  H:=[];
  for (j:=2;j<=n;j++) {
    if (L[j]==0) H:=append(H,j);
  }
  return H;
}
```

On tape :
`hamming(20)`
On obtient :
`[2,3,4,5,6,8,9,10,12,15,16,18,20]`
On tape :
`hamming(40)`
On obtient :
`[2,3,4,5,6,8,9,10,12,15,16,18,20,24,25,27,30,32,36,40]`

8.20.3 L'algorithme sans faire un crible

Supposons que l'on ait trouvé les premiers éléments de cette suite par exemple : 2,3,4,5.
L'élément suivant est obtenu en multipliant une des cases précédentes par 2, 3 ou 5.
Le problème c'est d'avoir les éléments suivants dans l'ordre....
Comment trouver lélément suivant de `H:=[2,3,4,5]` :
on a déja multiplié `H[0]=2` par 2 pour obtenir 4 donc
on peut multiplier `H[1]=3` par 2 pour obtenir m=6 ou
multiplier `H[0]=2` par 3 pour obtenir p=6 ou
multiplier `H[0]=2` par 5 pour obtenir q=10.
L'élément suivant est donc 6=min(6,6,10) et `H:=[2,3,4,5,6]`.
Maintenant, on a déja multiplier `H[0]=2` par 2 et par 3 pour obtenir 4 et 6 et
on a déja multiplier `H[1]=3` par 2 pour obtenir 6 donc
donc
on peut multiplier `H[2]=4` par 2 pour obtenir m=8 ou
multiplier `H[1]=3` par 3 pour obtenir p=9 ou
multiplier `H[0]=2` par 5 pour obtenir q=10.
L'élément suivant est donc 8=min(8,9,10) et `H:=[2,3,4,5,6,8]`.
Pour que chaque terme de la suite soit multiplié par 2, par 3 et par 5, il faut donc pévoir 3 indices :
`k0` qui sera l'indice de l'élément qu'il faut multiplier par 2,
`k1` qui sera l'indice de l'élément qu'il faut multiplier par 3,
`k2` qui sera l'indice de l'élément qu'il faut multiplier par 5.
Cela signifie que :
pour tout `r<k0` les `2*H[r]` ont déjà été rajoutés,
pour tout `r<k1` les `3*H[r]` ont déjà été rajoutés,
pour tout `r<k2` les `5*H[r]` ont déjà été rajoutés,
Naturellement `k0`$\geq$ `k1`$\geq$ `k2`.
Les 3 candidats pour être l'élément suivant sont donc :
`2*H[k0], 3*H[k1], 5*H[k2]`
l'un de ces éléments est plus petit que les autres et on le rajoute à la suite. Il faut alors augmenter l'indice correspondant de 1 : par exemple si c'est `3*H[k1]` qui est le minimum il faut augmenter `k1` de 1 et si `3*H[k1]= 5*H[k2]` est le minimum, il faut augmenter `k1` et `k2` de 1.

8.20.4 La traduction de l'algorithme avec Xcas

`hamming(n)` va renvoyer les `n` premiers éléments de la suite de Hamming.
L'indice `j` sert simplement à compter les éléments de `H`.
`k` est une suite qui contient les indices `k0,k1,k2`.
On peut initialiser `H` à `[2,3,4,5]` donc `j` à `4`, et `k` à `[1,0,0]` (car `H[0]=2` a été multiplié par 2, mais pas par 3, ni par 5) mais cela suppose `n>3`.
On peut aussi initialiser `H` à `[1]`, `k` à `[0,0,0]` (`H[0]=1` n'a pas été multiplié par 2, ni par 3, ni par 5) et `j` à `0` puis enlever `1` de `H` à la fin car `1` n'est pas un terme de la suite.

Voici la fonction `hamming(n)` écrite en `Xcas` pour `n>3`.

```
//pour n>3
hamming(n):={
  local H,j,k,m,p,q,mi;
  H:=[2,3,4,5];
  j:=4;
  k:=[1,0,0];
  while (j<n) {
  m:=2*H[k[0]];
  p:=3*H[k[1]];
  q:=5*H[k[2]];
  mi:=min(m,p,q);
  H:=append(H,mi);
  j:=j+1;
  if (mi==m) {k[0]:=k[0]+1};
  if (mi==p) {k[1]:=k[1]+1};
  if (mi==q) {k[2]:=k[2]+1};
  }
  return H;
}
```

Voici la fonction `hamming(n)` écrite en `Xcas` pour `n>0`.

```
//pour n>0
hamming(n):={
  local H,j,k,m,p,q,mi;
  H:=[1];
  j:=0;
  k:=[0,0,0];
  while (j<n) {
  m:=2*H[k[0]];
  p:=3*H[k[1]];
  q:=5*H[k[2]];
  mi:=min(m,p,q);
  H:=append(H,mi);
  j:=j+1;
  if (mi==m) {k[0]:=k[0]+1};
  if (mi==p) {k[1]:=k[1]+1};
  if (mi==q) {k[2]:=k[2]+1};
  }
  return tail(H);
}:;
```

On tape :
`hamming(20)`
On obtient :
`[2,3,4,5,6,8,9,10,12,15,16,18,20,24,25,27,30,32,36,40]`

8.21 Développement diadique de $\frac{a}{b} \in [0;1[$

8.21.1 L'énoncé

Le développement diadique de $\frac{a}{b} \in [0;1[$ est lécriture de $\frac{a}{b}$ sous la forme : $\frac{a}{b} = \frac{d_1}{2} + \frac{d_2}{2^2} + ... + \frac{d_k}{2^k}$ avec $d_k \in \{0,1\}$.

1. Écrire un programme qui affiche la liste des d_k se terminant par la liste des premiers termes $d_1, d_2.., d_n$ de la suite d,

2. Écrire un programme qui affiche la liste des d_k se terminant par la liste des d_k qui forme la période. Par exemple, on a :
$\frac{7}{12} = \frac{1}{2} + \frac{0}{2^2} + \frac{0}{2^3} + \frac{1}{2^4} + \frac{0}{2^5} + \frac{1}{2^6+....}$ et on écrit [1,0,0,1,0,[1,0]].

8.21.2 La solution

1. Si $q = \frac{a}{b}$ et si $d_1 = floor(2 * q)$, on a $\frac{a}{b} = \frac{d_1}{2} + \frac{2a-b*d_1}{2*b}$. Donc, le développement diadique de $\frac{a}{b} \in [0;1[$ est lécriture de $q = \frac{a}{b}$ sous la forme : d_1 suivi du développement diadique de la fraction $\frac{2a-b*d_1}{2*b}$ de $[0;1[$.
On tape pour avoir les premiers termes $d_1, d_2.., d_n$ de la suite d :

```
diadiquen(a,b,n):={
local d,q,k,p,L;
p:=2;
L:=NULL;
q:=a/b;
pour k de 1 jusque n faire
d:=floor(p*q);
L:=L,d;
q:=q-d/p;
p:=2*p
fpour;
retourne L;
}:;
```

On tape : `diadiquen(3,10,15)`
On obtient : `0,1,0,0,1,1,0,0,1,1,0,0,1,1,0`

2. Pour trouver la période, il faut savoir que l'on commence une période lorsque l'on retrouve parmi la liste des nouvelles fractions à développer un même numérateur. On garde donc dans `A` la liste des numérateurs en mettant un 0 quand le terme correspondant de d est nul.
On tape pour avoir les premiers termes de la suite d et sa période :

```
diadiques(a,b):={
local d,q,k,p,L,A;
L:=NULL;
A:=NULL;
q:=a/b;
```

```
a:=numer(q);
p:=2;
d:=floor(p*q);
repeter
L:=L,d;
si d!=0 alors
A:=A,a;
sinon
A:=A,0;
fsi;
q:=q-d/p;
a:=numer(q);
p:=2*p
d:=floor(p*q);
k:=member(a,[A]);
jusqua k!=0 and d!=0;
retourne [L,mid([L],k-1)];
}:;
```

On tape : `diadiques(3,10)`
On obtient : `[0,1,0,0,1,[1,0,0,1]]`

8.22 Écriture d'un entier comme $\sum_{j\geq 1} a_j j!$ avec $0 \leq a_j < j$

8.22.1 L'énoncé

On veut écrire un entier $n \in \mathbb{N}$ sous la forme $\sum_{j\geq 1} a_j j!$ avec $0 \leq a_j < j$ pour tout j.
Par exemple $43 = 1 \cdot 4! + 3 \cdot 3! + 0 \cdot 2! + 1 \cdot 1!$.

1. Quel est le plus grand entier J tel que $a_J \neq 0$?
2. Écrire un programme `ecritfac` qui renvoie les coefficients a_j du développement dans l'ordre décroissant : par exemple `ecritfac(43)` renverra `(1,3,0,1)`.

8.22.2 La solution

1. Montrons par recurrence que :

$$\sum_{j=1}^{j=N-1} j \cdot j! < N!$$

vrai pour $N = 2$ car 1<2 !=2
si $\sum_{j=1}^{j=N-1} j \cdot j! < N!$ alors $\sum_{j=1}^{j=N-1} j \cdot j! + N \cdot N! < N! + N \cdot N! = (N+1)!$
Si $n = \sum_{j=1}^{j=J} a_j j!$ avec $0 \leq a_j < j$ et $a_J \neq 0$ on a :
$J! \leq n = a_J J! + \sum_{j=1}^{j=J-1} a_j j! < J \cdot J! + \sum_{j=1}^{j=J-1} j \cdot j!$
donc $J! \leq n < (J+1)!$

2. On cherche d'abord la valeur de J, puis on fait le quotient de n par $J!$ et on recommence avec comme valeur de n le reste de la division de n par $J!$.
On tape :

```
ecritfac(n):={
  local j,J,k,L,a;
  L:=NULL;
  j:=1;
  tantque n>=j! faire j:=j+1 ftantque;
  J:=j-1;
  pour k de J jusque 1 pas -1 faire
    a:=iquo(n,k!);
    L:=L,a;
    n:=irem(n,k!);
  fpour;
return L;
}:;
```

On tape : `ecritfac(43)`
On obtient : `(1,3,0,1)` On tape : `ecritfac(150)`
On obtient : `(1,1,1,0,0)`

8.23 Les nombres de Fermat et les nombres de Mersenne

8.23.1 Définitions et téorèmes

Définitions
Les nombres de **Fermat** sont les entiers de la forme :
$2^{(2^p)} + 1$ pour $p \in \mathbb{N}$.
Lorsque pour $p \in \mathbb{N}$, $F_p = 2^{(2^p)}+1$ est premier on dit que c'est un nombre premier de **Fermat**.
Les nombres de **Mersenne** sont les entiers de la forme :
$2^p - 1$ pour p st un nombre premier.
Lorsque pour $p \in \mathbb{N}$, $M_p = 2^p - 1$ est premier on dit que c'est un nombre premier de **Mersenne**.
Téorème 1
Si $M_n = 2^n - 1$ est premier, alors n est aussi premier.
La réciproque est fausse (voir le **Test de Lucas-Lehmer** ci-après), par exemple, M_{11} n'est pas premier :
$M_{11} = 2^{11} - 1 = 2047 = 23 * 89$
Téorème 2
Soient 2 nombres premiers p et q. Si q divise $M_p = 2^p - 1$, alors $q = +/- 1(\mathrm{mod}8)$ et il existe $k \in \mathbb{N}$ tel que $q = 2kp + 1$.
Téorème 3
Si p un nombre premier vérifiant $p = 3(\mathrm{mod}4)$ alors $2p + 1$ est un nombre premier si et seulement si $2p + 1$ divides M_p.

8.23.2 Exercices (niveau classe de terminale

— Nombres de Fermat

1. Montrer que si $n \in \mathbb{N}$ et $p \in \mathbb{N}$ alors n^{2p+1} est divisible par $n+1$.
2. En déduire que si $p \in \mathbb{N}$ et k un entier impair alors $2^{(2^p k)} + 1$ est divisible par $2^{(2^p)} + 1$.
3. Montrer que si $2^n + 1$ est premier ($n \in \mathbb{N}$) alors n est une puissance de 2.
4. Montrer que $2^{32} + 1$ n'est pas premier et en déduire que la réciproque de la question 3 est fausse.

— Nombres de Mersenne

1. Montrer que si $p \in \mathbb{N}$ et $q \in \mathbb{N}$ alors $2^{pq} - 1$ est divisible par $2^p - 1$ et par $2^q - 1$.
2. En déduire que si $2^n - 1$ est premier ($n \in \mathbb{N}$) alors n est un nombre premier.
3. Montrer que $2^{11} - 1$ n'est pas premier et en déduire que la réciproque de la question 2 est fausse.

Solution

— Nombres de Fermat

1. Si $n \in \mathbb{N}$ et $p \in \mathbb{N}$ alors $n^{2p+1} + 1 = (n+1)(n^{2p} - n^{2p-1} + ... + n^2 - n + 1 = (n+1)\sum_{k=0}^{2p}(-1)^k n^k$ donc $n^{2p+1} + 1$ est divisible par $n+1$.
   Avec `Xcas`, on tape :
   `simplify((n+1)*sum((-1)^k*n^k,k=0..2p))`
   On obtient :
   `1+n^(1+2*p)`
2. k un entier impair on a $k = 2q + 1$ avec $q \in \mathbb{N}$ donc :
   $2^p k = 2^p(2q+1)$ et $2^{(2^p k)} + 1 = \left(2^{(2^p)}\right)^{2q+1} + 1$
   Donc d'après la question 1 (en posant $n = 2^{(2^p)}$) $2^{(2^p k)} + 1$ est divisible par $2^{(2^p)} + 1$.
3. si $2^n + 1$ est premier ($n \in \mathbb{N}$) alors n est une puissance de 2 car
   si n n'est pas une puissance de 2 il esiste $p \in \mathbb{N}$ et k un entier impair tel que $n = 2^p * k$ et d'après la question 2, $2^{(2^p k)} + 1$ n'est pas premier puisqu'il est divisible par $2^{(2^p)} + 1$.
4. On tape :
   `2^32+1`
   On obtient :
   `4294967297`
   On tape :
   `ifactor(4294967297)`
   On obtient :
   `641*6700417`
   Donc $2^{32} + 1$ n'est pas premier et on en déduit que la réciproque de la question 3 est fausse.

— Nombres de Mersenne

1. Pour tout entier n on a $x^n-1=(x-1)(x^{n-1}+x^{n-2}+...x^2+x+1$.
   On tape :
   `simplify((x-1)*sum(x^k,k=0..n-1))`
   On obtient :
   `-1+x^n`
   Si $p\in\mathbb{N}$ et $q\in\mathbb{N}$ alors on a :
   $2^{pq}-1=((2^p)^q-1)=((2^q)^p-1)$ donc :
   $2^{pq}-1$ est divisible par 2^p-1 et par 2^q-1.
2. 2^n-1 est premier ($n\in\mathbb{N}$) alors n est un nombre premier puisque si n n'est pas premier il existe 2 entiers $p>1$ et $q>1$ tel que $n=pq$ et d'après la question 1 cela entraine que 2^n-1 n'est pas premier puisqu'il est divisible par 2^p-1.
3. On tape :
   `2^11-1`
   On obtient :
   `2047`
   On tape :
   `ifactor(4294967297)`
   On obtient :
   `23*89`
   Donc $2^{11}-1$ n'est pas premier et on en déduit que la réciproque de la question 2 est fausse.

8.24 Les nombres parfaits

8.24.1 Définitions et téorèmes

Un nombre entier $n\geq 2$ est **parfait** si il est égal à la somme de ses diviseurs propres (1 est compris mais pas n).
Par exemple 6 et 28 sont parfaits car 6=1+2+3 et 28=1+2+4+7+14.
Téorème 1
k est un nombre parfait pair si et seulement si il est de la forme $k=2^{n-1}(2^n-1)$ avec $M_n=2^n-1$ premier.
Téorème 2
Si on fait la somme des chiffres d'un nombre parfait pair différent de 6, puis la somme des chiffres du résultat et que l'on continue le processus alors on obtient 1.
Exercice
Écrire un programme qui teste si un nombre n vérifie le théorème 5.

Il faut donc utiliser la fonction `revlist(convert(n,base,10))` de `Xcas` qui renvoie la liste des chiffres de l'écriture en base 10 de l'entier `n`.
On tape :

```
sumchiffre(n):={
local L,s;
si n==6 alors retourne 1 fsi;
```

```
s:=n;
tantque s>9 faire
L:=convert(n,base,10);
s:=sum(L);
ftantque;
si s==1 alors retourne 1;
sinon  retourne s;
fsi;
}:;
```

8.24.2 Test de Lucas-Lehmer

Test de Lucas-Lehmer
Si p est un nombre premier alors le nombre de Mersenne $M_p = 2^p - 1$ est premier si et seulement si $2^p - 1$ divise $S(p-1)$ lorsque $S(n)$ est la suite définie par $S(n+1) = S(n)^2 - 2$, et $S(1) = 4$. **Exercice**
Écrire le programme correspondant à ce test : on calculera la suite $S(n)$ modulo $2^p - 1$ pour gagner du temps.
On tape :

```
Test_LL(p):={
local s,j;
s := 4;
pour j de 2 jusque p-1 faire
  s := s^2-2 mod n;
fpour;
si s == 0 alors
     return "2^"+string(p)+"-1 est premier";
  sinon
     return "2^"+string(p)+"-1 est non premier";
fsi;
}:;
```

On tape :
`Test_LL(11213)` On obtient (Evaluation time : 6.43) :
`2^11213-1 est premier`
On tape :
`Test_LL(11351)` On obtient :
`2^11351-1 est non premier`
Remarque
En janvier 1998, un élève ingénieur a prouvé que M_p était premier pour $p = 3021377$ (M_p a 909526 chiffres !).

8.25 Les nombres parfaits et les nombres amiables

8.25.1 Les nombres parfaits

Définition
Un nombre entier $n \geq 2$ est **parfait** si il est égal à la somme de ses diviseurs

propres (1 est compris mais pas n).
Par exemple 6 et 28 sont parfaits car 6=1+2+3 et 28=1+2+4+7+14.

L'énoncé

Quels sont les nombres parfaits inférieurs à 11000 ?
Montrer que si 2^p-1 est premier alors $2^{p-1}(2^p-1)$ est parfait. **La solution**
On utilise l'instruction `idivis(n)` qui renvoie la liste des diviseurs de l'entier `n` et l'instruction `sum(L)` qui renvoie la somme de la liste `L`.
On tape avec les instructions françaises :

```
parfait(n):={
  local j,a,b,L;
  L:=NULL;
  pour j de 2 jusque n faire
    a:=sum(idivis(j))-j;
    si a==j alors L:=L,j; fsi;
  fpour;
  retourne L;
}:;
```

On tape pour avoir les nombres parfaits inférieur à 11000 :
`parfait(11000)` 0n obtient :
`6,28,496,8128`

Si 2^p-1 est premier alors les diviseurs de $2^{p-1}(2^p-1)$ sont :
$1,(2^p-1),2,2(2^p-1),2^2,2^2(2^p-1),..2^{p-2},2^{p-2}(2^p-1),2^{p-1},2^{p-1}(2^p-1)$.
La somme de ces diviseurs est :
On tape pour avoir cette somme simplifiée et factorisée :
`factor(normal(sum(2^k*(1+2^p-1),k=0..p-1)))`
0n obtient :
`2^p*(2^p -1)`
La somme de tous les diviseurs propres de $2^{p-1}(2^p-1)$ est $2^{p-1}(2^p-1)$ donc si 2^p-1 est premier $2^{p-1}(2^p-1)$ est parfait.
Euler a montré que tous les nombres parfaits pairs sont de cette forme.

Donc $2^{p-1}(2^p-1)$ est parfait si $M_p=2^p-1$ est premier.
Pour $p=2$ on a $2^2-1=3$ est premier donc 2*3=6 est parfait.
Pour $p=3$ on a $2^3-1=7$ est premier donc 4*7=28 est parfait.
Pour $p=5$ on a $2^5-1=31$ est premier donc 16*31=496 est parfait.
Pour $p=7$ on a $2^7-1=127$ est premier donc 64*127=8128 est parfait.
Pour $p=13$ on a $2^{13}-1=8191$ est premier donc 4096*8191=33550336 est parfait.
Pour $p=17$ on a $2^{17}-1=13107$ est premier donc 65536*131071=8589869056 est parfait (il a été découvert en 1588 par Cataldi).
Pour $p=19$ on a $2^{19}-1=524287$ est premier donc 262144*524287=137438691328 est parfait (il a été découvert en 1588 par Cataldi).
Puis pour $p=31,61,89$ on a encore 2^p-1 premier ....
En 1936 le Dr Samuel I. Krieger dit que pour $p=257$ $2^{513}-2^{256}$ est parfait (il a 155 chiffres) malheureusement le nombre $2^{257}-1$ n'est pas premier..... .

On refait donc un programme qui teste si $2^p - 1$ est premier et on en déduit le nombre parfait correspondant.

```
parfait2(p):={
  local j,a,b,L;
  L:=NULL;
  pour j de 2 jusque p faire
   a:=2^(j-1);
   b:=2*a-1;
   si isprime(b) alors L:=[L,a*b,j]; fsi;
  fpour;
  retourne L;
}:;
```

On tape :
`A:=parfait2(1100)`
`size(A)`
On obtient :
`14`
On tape :
`A[13]`
On obtient le 14ième nombre parfait :
`[2^606*(2^607-1),607]`
On tape :
`B:=[A]:;`
`col(B,1)`
On obtient la liste des nombres $p \leq 1100$ tels que $2^p - 1$ soit premier :
`[2,3,5,7,13,17,19,31,61,89,107,127,521,607]`
Remarque : relation entre les nombres parfaits et les cubes
0n remarque qu'à part 6 chaque nombre parfait est égal à la somme des cubes de nombres impairs consécutifs :
$28 = 1^3 + 3^3$=`sum((2*n+1)^3,n=0..1)`
$496 = 1^3 + 3^3 + 5^3 + 7^3$=`sum((2*n+1)^3,n=0..3)`
$8128 = 1^3 + 3^3 + 5^3 + 7^3 + 9^3 + 11^3 + 13^3 + 15^3$=`sum((2*n+1)^3,n=0..7)`
33550336 =`sum((2*n+1)^3,n=0..63)`
8589869056 =`sum((2*n+1)^3,n=0..255)`
137438691328 =`sum((2*n+1)^3,n=0..511)`
0n remarque aussi que l'on fait la somme pour n variant de 0 à :
$1 = 2^1 - 1$
$3 = 2^2 - 1$
$7 = 2^3 - 1$
$63 = 2^6 - 1$
$255 = 2^8 - 1$
$511 = 2^9 - 1$

Question ouverte Existe-t-il des nombres parfaits impairs ? ? ? ? ? ? ? ? ? ? ? ? Ce que l'on sait c'est que si il en existe un alors il est tres grand ! Cete question est certainement le plus vieux problème de mathematiques non résolu....

8.25.2 Les nombres amiables

L'énoncé

On se propose d'écrire un programme qui donne la suite des couples amiables dont l'un des nombres est inférieur ou égal à un entier n. Voici les définitions des nombres parfaits et des nombres amiables :

Définitions

Un nombre n est **parfait** si il est égal à la somme de ses diviseurs propres (1 est compris mais pas n).

Deux nombres a et b sont **amiables** ou amis, si l'un est égal à la somme des diviseurs propres de l'autre et inversement.

Les nombres parfaits a sont des nombres amiables avec eux mêmes. **La solution**

On utilise l'instruction `idivis(n)` qui renvoie la liste des diviseurs de l'entier `n` et l'instruction `sum(L)` qui renvoie la somme de la liste `L`.

Pour ne pas avoir 2 fois le même couple on n'affiche que les couples $[j, a]$ avec $j \leq a$.

On tape avec les instructions françaises :

```
amiable(n):={
  local j,a,b,L;
  L:=NULL;
  pour j de 2 jusque n faire
    a:=sum(idivis(j))-j;
    b:=sum(idivis(a))-a;
    si b==j et j<=a alors L:=L,[j,a]; fsi;
  fpour;
  retourne L;
}:;
```

On tape pour avoir les nombres amiable inférieur à 11000 :

`amiable(11000)` 0n obtient :

```
[6,6],[28,28],[220,284],[496,496],[1184,1210],[2620,2924],
[5020,5564],[6232,6368],[8128,8128],[10744,10856]
```

Les nombres parfaits a sont les nombres amiables $[a, a]$.

8.26 Les parallélépipèdes rectangles presque parfaits

8.26.1 L'énoncé

On se propose d'écrire un programme qui donne les dimensions des parallélépipèdes rectangles presque parfaits dont les côtés sont inférieurs ou égaux à un entier $n \leq 1000$. Voici la définition d'un parallélépipède rectangle presque parfait :

Définitions

Un parallélépipède rectangle est presque parfait si :

1. les longueurs de ses côtés sont des nombres entiers,
2. les longueurs des dagonales de ses 3 faces sont aussi des nombres entiers.

Par exemple, le parallélépipède rectangle de côtés 44 17 240 est presque parfait.

Attention les 6 permutations de (44 17 240) représentent le même parallélépipède rectangle.

8.26.2 La solution

Si 3 entiers (a, b, c) vérifiant $a < b < c$ représentent un parallélépipède rectangle, pour su'il soit presque parfait il faut que :

1. $\sqrt{a^2 + b^2}$ soit un entier,
2. $\sqrt{a^2 + c^2}$ soit un entier,
3. $\sqrt{b^2 + c^2}$ soit un entier

Comment tester qu'un nombre p est un carré parfait ? On peut écrire :
`frac(sqrt(p))==0` ou
`floor(sqrt(p))^2-p==0`
mais cela demande un calcul ! Il est donc préferable d'utiliser la commande `type` qui renvoie le type de l'argument. Par exemple :
`type(sqrt(4))==integer` renvoie `1` et `type(sqrt(5))==integer` renvoie `0`.
On tape :

```
parapparfait(n):={
  local a,b,c,L;
  L:=NULL;
  pour a de 1 jusque n faire
    pour b de a+1 jusque n faire
      si type(sqrt(a^2+b^2))==integer  alors
         pour c de b+1 jusque n faire
           si type(sqrt(a^2+c^2))==integer alors
             si type(sqrt(c^2+b^2))==integer alors
                L:=L,[a,b,c];
             fsi;
           fsi;
        fpour;
      fsi;
    fpour;
  fpour;
  retourne L;
}:;
```

0n tape : `L:=parapparfait(1000)`
On obtient (c'est long !) :
`[44,117,240],[85,132,720],[88,234,480],[132,351,720],`
`[140,480,693],[160,231,792],[176,468,960],[240,252,275],`
`[480,504,550],[720,756,825]`

On peut modifier le programme pour avoir pour chaque a, une liste provisoire `P` qui sera la liste $a, b_1, b_2 ... b_p$ telle que $a^2 + b_j^2$ soit un carré. Puis dans cette liste on cherchera les b_j et les b_k tels que $b_j^2 + b_k^2$ soit un carré. Le triplet $[a, b_j, b_k]$ répond alors à la quesstion.
On tape :

```
paralparfait(n):={
  local a,b,c,L,P,j,s,k,b2;
  L:=NULL;
```

```
  pour a de 1 jusque n faire
    P:=a;
    pour b de a+1 jusque n faire
      si type(sqrt(a^2+b^2))==integer  alors
         P:=P,b;
      fsi;
    fpour;
    s:=size(P)-1;
    pour j de 1 jusque s-1 faire
      b:=P[j];
      b2:=b^2;
      pour k de j+1 jusque s faire
        c:=P[k];
        si type(sqrt(b2+c^2))==integer alors
          L:=L,[a,b,c];
        fsi;
       fpour;
     fpour;
   fpour;
   retourne L;
}:;
```

0n tape : `L:=paralparfait(1000)`
On obtient (c'est 2 fois moins long !) :

```
[44,117,240],[85,132,720],[88,234,480],[132,351,720],
[140,480,693],[160,231,792],[176,468,960],[240,252,275],
[480,504,550],[720,756,825]
```

Remarque

On peut facilement avoir les couples a, b tel que $a^2 + b^2$ soit un carré.
On tape :

```
sommecarre(n):={
  local a,b,L,P;
  L:=NULL;
  pour a de 1 jusque n faire
    P:=a;
    pour b de a+1 jusque n faire
      si type(sqrt(a^2+b^2))==integer  alors
        P:=P,b;
      fsi;
    fpour;
    si size(P)>1 alors L:=L,[P];fsi;
  fpour;
  retourne L;
}:;
```

On tape : `sommecarre(100)`
On obtient :

```
[3,4],[5,12],[6,8],[7,24],[8,15],[9,12,40],[10,24],
```

```
[11,60],[12,16,35],[13,84],[14,48],[15,20,36],
[16,30,63],[18,24,80],[20,21,48,99],[21,28,72],
[24,32,45,70],[25,60],[27,36],[28,45,96],[30,40,72],
[32,60],[33,44,56],[35,84],[36,48,77],[39,52,80],
[40,42,75,96],[42,56],[45,60],[48,55,64,90],[51,68],
[54,72],[56,90],[57,76],[60,63,80,91],[63,84],[65,72],
[66,88],[69,92],[72,96],[75,100],[80,84]
```

Par exemple, on voit que :
$20^2 + 15^2$ est un carré : c'est 25^2,
$20^2 + 21^2$ est un carré : c'est 29^2,
$20^2 + 48^2$ est un carré : c'est 52^2,
$20^2 + 99^2$ est un carré : c'est 101^2.
ou encore
$60^2 + 11^2$ est un carré : c'est 61^2,
$60^2 + 25^2$ est un carré : c'est 65^2,
$60^2 + 32^2$ est un carré : c'est 68^2,
$60^2 + 45^2$ est un carré : c'est 75^2,
$60^2 + 63^2$ est un carré : c'est 87^2,
$60^2 + 80^2$ est un carré : c'est 100^2.
$60^2 + 91^2$ est un carré : c'est 109^2.
On peut aussi en déduire que :
$60^2 + 144^2$ est un carré : c'est 156^2 (car $15^2 + 36^2 = 39^2$).
$60^2 + 175^2$ est un carré : c'est 185^2 (car $12^2 + 35^2 = 37^2$).
$60^2 + 297^2$ est un carré : c'est 303^2.(car $20^2 + 99^2 = 101^2$).
Mais si on veut tous les nombres $b \leq 300$ tels que $n^2 + b^2$ soit un carré il est préférable d'écrire le programme :

```
n2plusb2(n,N):= {
local b,P;
P:=n;
pour b de 1 jusque N faire
si type(sqrt(n^2+b^2))==integer  alors
P:=P,b;
fsi;
fpour ;
P;
}:;
```

On tape : `n2plusb2(20,1000)`
On obtient :
`20,15,21,48,99`
On tape : `n2plusb2(60,300)`
On obtient :
`60,11,25,32,45,63,80,91,144,175,221,297`
seul $60^2+221^2 = 229^2$ n'avait pas été trouvé précédemment car 229 est un nombre premier !
On tape : `n2plusb2(60,1000)`
On obtient :
`60,11,25,32,45,63,80,91,144,175,221,297,448,899`

8.27 Les nombres heureux

8.27.1 L'énoncé

On se propose d'écrire un programme qui donne la suite des nombres heureux inférieurs ou égaux à un entier n. Voici un algorithme définissant cette suite :

- On écrit la suite des nombres entiers de 2 à n,
- On entoure 2 et on supprime les nombres de 2 en 2,
- On entoure 3 et on supprime les nombres de 3 en 3,
- On continue de la même façon : à chaque fois, on entoure m le premier nombre non entouré et on supprime les nombres de m en m,
- On s'arrête quand il ne reste que des nombres entourés : ce sont les nombres heureux

Par exemple :
après la première étape on a : $2, 3, 5, 7, 9, 11, 13, 15, 17$...
après la deuxième étape on a : 2,3,5,7,11,17...

8.27.2 La solution

Ce programme ressemble au crible d'Eratosthène, mais si m est le nombre que l'on vient d'entourer et qu'il est d'indice p, on supprime les nombres d'indices $p+m$,$p+2m$...$p+km$ mais dans la liste `tab` modifiée et non les multiples du nombre m.
On tape avec les instructions françaises :

```
heureux(n):={
  local tab,heur,m,j,p,k;
  tab:=j$(j=2..n);
  tab:=concat([0,0],[tab]);
  heur:=[];
  p:=2;
  tantque (p<=n) faire
    m:=p;
    k:=0;
    pour j de p+1 jusque n faire
      si tab[j]!=0 alors k:=k+1; fsi;
      si irem(k,m)==0 alors tab[j]:=0 fsi;
    fpour;
    p:=p+1;
    si p<=n alors
      tantque tab[p]==0 and p<n faire p:=p+1 ftantque;
      si p==n and tab[p]==0 alors p:=n+1;fsi;
    fsi;
  ftantque;
  pour p de 2 jusque n faire
    si (tab[p]!=0) alors
      heur:=append(heur,p);
    fsi;
```

```
  fpour;
  retourne(heur);
};
```

Dans ce programme on peut se passer de la liste `heur` : il suffit de supprimer la dernière instruction `pour` et de mettre :
`retourne remove(x->x==0,tab);` à la place de `retourne heur` On tape : `heureux(100)`
On obtient :
`[2,3,5,7,11,13,17,23,25,29,37,41,43,47,53,61,67,71,77,83,89,91,97]`

On peut aussi et ce sera plus rapide, modifier la liste `tab` au fur et à mesure en supprimant à chaque étape les valeurs barrées c'est à dire les valeurs mises à 0 en utilisant l'instruction `remove` et en mettant au fur et à mesure les nombres heureux dans `heur`.
On tape avec les instructions françaises :

```
//renvoie la liste des nombres heureux<=n
heureux2(n):={
  local tab,heur,m,j,k,s;
  tab:=[j$(j=2..n)];
  heur:=[];
  s:=dim(tab);
  k:=0;
  tantque (s>0) faire
    j:=0;
    m:=tab[0];
    heur[k]:=m;
    tantque j<s faire
      tab[j]:=0;
      j:=j+m;
    ftantque;
      tab:=remove(x->x==0,tab);
      s:=dim(tab);
      k:=k+1;
  ftantque;
  retourne(heur);
};
```

On tape : `heureux2(100)`
On obtient :
`[2,3,5,7,11,13,17,23,25,29,37,41,43,47,53,61,67,71,77,`
`83,89,91,97]`

8.28 L'équation de Pell

8.28.1 Les proriétés

Résoudre l'équation de Pell c'est trouver les plus petits entiers x, y qui sont solutions de $x^2 - n * y^2 = 1$ lorsque n est un entier.

Euler à montré que l'on pouvait résoudre cette équation à l'aide du développement en fraction continue de $\sqrt{n}$ (par exemple `dfc(n,20)`). Supposons que le développement en fraction continue de $\sqrt{n}$ soit de période k. On a :

— pour $k = 2h - 1$,
$\sqrt{n} = [a_0, a_1, .., a_{h-1}, a_{h-1}, .., a_1, 2a_0], [a_1, .., a_{h-1}, a_{h-1}, .., a_1, 2a_0]]$.
Par exemple
`dfc(sqrt(13),20)=[3,1,1,1,1,6,[1,1,1,1,6]]`

— pour $k = 2h$,
$\sqrt{n} = [a_0, a_1, .., a_{h-1}, a_h, a_{h-1}, .., a_1, 2a_0], [a_1, .., a_{h-1}, a_h, a_{h-1}, .., a_1, 2a_0]]$
(ici $k = 5$,)
Par exemple
`dfc(sqrt(23),20)=[4,1,3,1,8,[1,3,1,8]]` (ici $k = 4$)

Soit le développement en fraction continue de $\sqrt{n}$:
$\sqrt{n} = [a_0, a_1, ...a_k, x_k]$
Soient :

$$\frac{A_k}{B_k} = [a_0, a_1, ...a_k]$$

$$x_k = \frac{P_k + \sqrt{n}}{Q_k}$$

On a les relations :

$$A_2 = 0,\ A_1 = 1,\ A_k = a_k A_{k-1} + A_{k-2}$$

$$B_2 = 1,\ B_1 = 0,\ B_k = a_k B_{k-1} + B_{k-2}$$

$$a_k = \text{floor}(\frac{P_k + \sqrt{n}}{Q_k})$$

Puisque $x_{k+1} = 1/(x_k - a_k)$ on a :

$$P_0 = 0,\ P_{k+1} = a_k Q_k - P_k$$

$$Q_0 = 1,\ Q_{k+1} = a_k(P_k - P_{k-1}) + Q_{k-1}$$

Donc

$$\frac{A_k}{B_k} - \frac{A_{k+1}}{B_{k+1}} = (-1)^{k+1}\frac{1}{B_k B_{k+1}}$$

On a :

$$\sqrt{n} = \frac{A_{k-1}x_k + A_{k-2}}{B_{k-1}x_k + B_{k-2}} = \frac{A_{k-1}\sqrt{n} + P_k A_{k-1} + Q_k A_{k-2}}{B_{k-1}\sqrt{n} + P_k B_{k-1} + Q_k B_{k-2}}$$

Donc

$$Q_k A_{k-2} + P_k A_{k-1} = nB_{k-1}$$

$$Q_k B_{k-2} + P_k B_{k-1} = A_{k-1}$$

On a donc

$$(A_{k-2}B_{k-1} - A_{k-1}B_{k-2})Q_k = nB_{k-1}^2 - A_{k-1}^2 = (-1)^{k-1}Q_k$$

Si $(P_n+\sqrt{n})/Q_n$ est le n-ième quotient du développement en fraction continue de $\sqrt{n}$ alors si $Q_h = Q_{h-1}$ (resp $P_h = P_{h-1}$) on a $k = 2h-1$ (resp $k = 2h-2$). On en déduit donc la valeur k de la période.
La plus petite solution de $x^2 - ny^2 = 1$ est donnée par :
$A_{k-1} + B_{k-1}\sqrt{n}$ où A_n/B_n est la n-ième fraction convergeant vers $\sqrt{n}$.
Remarque
Soit n un entier. Si x_0, y_0 sont solutions de $x^2 - n * y^2 = 1$ alors x_k, y_k sont d'autres solutions grace à la formule :

$$x_k + y_k\sqrt{n} = (x_0 + y_0\sqrt{n})^k$$

En effet, par récurrence si on a $x_k + y_k\sqrt{n} = (x_0 + y_0\sqrt{n})^k$ et (x_k, y_k) solution de $x^2 - ny^2 = 1$ alors :
$x_{k+1} + y_{k+1}\sqrt{n} = (x_0 + y_0\sqrt{n}) * (x_k + y_k\sqrt{n})$ donc
$x_{k+1} = x_0x_k + ny_0y_k$ et
$y_{k+1} = x_0y_k + y_0x_k$ donc
$x_{k+1}^2 = (x_0x_k + ny_0y_k)^2 = x_0^2x_k^2 + n^2y_0^2y_k^2 + 2nx_0x_ky_0y_k =$
$x_0^2x_k^2 + (1 + x_0^2)(1 + x_k^2) + 2nx_0x_ky_0y_k = 2x_0^2x_k^2 + x_0^2 + 1 + x_k^2 + 2nx_0x_ky_0y_k$
$ny_{k+1}^2 = n(x_0y_k + y_0x_k)^2 = nx_0^2y_k^2 + ny_0^2x_k^2 + 2nx_0y_ky_0x_k =$
$= x_0^2(1 + x_k^2) + (1 + x_0^2)x_k^2 + 2nx_0y_ky_0x_k = 2x_0^2x_k^2 + x_0^2 + x_k^2$
donc

$$x_{k+1}^2 - ny_{k+1}^2 = 1$$

8.28.2 Le programme

Le programme ci dessous trouve les plus petits entiers A, B qui sont solutions de $A^2 - n * B^2 = 1$ lorsque n est entier qui n'est pas un carré parfait.
Les différentes valeurs de a sont le développement en fraction continue de $\sqrt{n}$. A_p/B_p sont les réduites de rang p ($A_p/B_p = [a_0, ...a_p]$) On a P_k et Q_k qui sont tels que :
$\sqrt{n} = [a-0, a-1, ...a_{k-1} + Q_k/(\sqrt{n} + P_k)]$

```
Pell(n):={
local A,B,P,Q,R,a,sn,AA,BB,NA,NB;
if (type(n)!=DOM_INT) {return "erreur"};
if (round(sqrt(n))^2==n) {return "erreur"};
R:=0;
Q:=1;
P:=0;
sn:=floor(sqrt(n));
a:=sn;
AA:=1;A:=a;
BB:=0;B:=1;
print(a,P,Q,R,A,B);
while (A^2-n*B^2!=1) {
P:=sn-R;
Q:=(n-P^2)/Q;
a:=floor((P+sn)/Q);
R:=irem(P+sn,Q);
```

```
NA:=a*A+AA;
NB:=a*B+BB;
AA:=A;
BB:=B;
A:=NA;
B:=NB;
print(a,P,Q,R,A,B);
}
return (A,B);
}
:;
```

On tape :
`Pell(13)`
On obtient :
`649,180`
En effet : $649^2 - 13 * 180^2 = 1$ On tape :
`Pell(43)`
On obtient :
`3482,531`
En effet : $3482^2 - 43 * 531^2 = 1$

Chapitre 9

Exercices de combinatoire

9.1 Fonction partage ou nombre de partitions de $n \in \mathbb{N}$

9.1.1 L'énoncé

Définition
Un partition de $n \in \mathbb{N}$ est une suite de nombres entiers : $\lambda_1 \geq \lambda_2 \geq ...\lambda_m > 0$ tels que : $\sum_{j=1}^{m} \lambda_j = n$.
On note $p(n)$ la fonction partage de n : c'est le nombre de partitions distinctes de $n \in \mathbb{N}$ et on convient que p(0)=1.
Exemples
$p(5) = 7$ car les 7 partitions distinctes de 5 sont :
$1+1+1+1+1=5$
$2+1+1+1=5$
$2+2+1=5$
$3+1+1=5$
$3+2=5$
$4+1=5$
$5=5$

1. Chercher à la main : $p(1), p(2), p(3), p(4), p(5), p(6), p(7)$
2. Écrire un programme qui renvoie les valeurs de $p(0), p(1), p(2), p(3)..p(n)$
3. Montrer ce que Euler a remarqué, à savoir que le développement en séries tronqué à l'ordre n de : $\prod_{j=1}^{n} \frac{1}{1-x^j}$ vaut $\sum_{j=1}^{n} p(j)x^j$.
4. Écrire un programme qui renvoie les coefficients du développement en séries tronqué à l'ordre n de : $\prod_{j=1}^{n} \frac{1}{1-x^j}$

9.1.2 La solution

1. On trouve :
   $p(0) = 1, p(1) = 1, p(2) = 2, p(3) = 3, p(4) = 5, p(5) = 7, p(6) = 11, p(7) = 15$
2. Soit A une matrice triangulaire inférieure tel que :
   si $k \leq j$, $A[j,k]$ represente le nombre de partitions de j tel que $\lambda_1 = k \geq \lambda_2 \geq ...\lambda_m > 0$. **Sur l'exemple**

$p(5)$=7 car les 7 partitions distinctes de 5 sont :
$1+1+1+1+1=5$ donc $A[5,1]=1$
$2+1+1+1=5$
$2+2+1=5$ donc $A[5,2]=2$
$3+1+1=5$
$3+2=5$ donc $A[5,3]=2$
$4+1=5$ donc $A[5,4]=1$
$5=5$ donc $A[5,5]=1$
On a alors $p(j)=\sum_{k=0}^{j} A[j,k]$.
On remarque donc :
$A[0,0]=1$ et $A[j,0]=0$ (car $\lambda_m>0$ sauf pour $j=0$ puisque $p(0)=1$)
$A[j,j]=1$ puisque $j=j$
$A[j,1]=A[j-1,1]=1$ puisque $j=j-1+1=j-2+1+1=1+1...+1$
$A[j,2]=A[j-2,1]+A[j-2,2]$ puisque $j=2+1+1+...=2+2+....$
$A[j,3]=A[j-3,1]+A[j-3,2]+A[j-3,3]$ puisque $j=3+1+1+...1=3+2+....=3+3+....$
etc...
$A[j,j-1]=A[j-1,j-1]=1$ puisque $j=(j-1)+1$
ou encore :
les coefficients d'indice 0 de toutes les lignes sont nuls et
pour $k=1..n$ le coefficient d'indice k de la n-ième ligne est obtenu en faisant la somme des coefficients d'indice $0..k$ de la $n-k$-ième ligne c'est à dire :

$$A[n,k]=\sum_{j=1}^{k} A[n-k,j] \text{ et}$$

$$A[n,n]=1 \text{ et } A[n,0]=0 \text{ si } n>0$$

Voici l'illustration pour $n=5$ avec
$k=2$, on a 5=0+1+4 :
$k=3$, on a 8=0+1+3+4 :
$k=4$, on a 9=0+1+3+3+2 :
....

$$\left(\begin{array}{cccccccccc|}
\underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0}| \\
\underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0}| & 0 \\
\underline{0} & \underline{1} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0}| & 0 & 0 \\
\underline{0} & \underline{1} & \underline{1} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0}| & 0 & 0 & 0 \\
\underline{0} & \underline{1} & \underline{2} & \underline{1} & \underline{1} & \underline{0} & \underline{0}| & 0 & 0 & 0 & 0 \\
\underline{0} & \underline{1} & \underline{2} & \underline{2} & \underline{1} & \underline{1}| & 0 & 0 & 0 & 0 & 0 \\
\underline{0} & \underline{1} & \underline{3} & \underline{3} & \underline{2}| & 1 & 1 & 0 & 0 & 0 & 0 \\
\underline{0} & \underline{1} & \underline{3} & \underline{4}| & 3 & 2 & 1 & 1 & 0 & 0 & 0 \\
\underline{0} & \underline{1} & \underline{4}| & 5 & 5 & 3 & 2 & 1 & 1 & 0 & 0 \\
\underline{0} & \underline{1}| & 4 & 7 & 6 & 5 & 3 & 2 & 1 & 1 & 0 \\
0 & 1 & 5 & 8 & 9 & 7 & 5 & 3 & 2 & 1 & 1
\end{array}\right)$$

On va faire un programme qui va calculer les coefficients de la matrice A.
On tape :

```
partitions(n):={
  local A,j,k,m,S;
```

```
  A:=idn(n+1);
  pour j de 2 jusque n faire
    pour k de 1 jusque j-1 faire
      S:=0;
      pour m de 1 jusque k faire
        S:=S+A[j-k,m];
      fpour;
      A[j,k]:=S;
    fpour;
  fpour;
  retourne A;
}:;
```

On tape : `A:=partitions(10)`
On obtient :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 3 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 4 & 3 & 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 5 & 5 & 3 & 2 & 1 & 1 & 0 & 0 \\ 0 & 1 & 4 & 7 & 6 & 5 & 3 & 2 & 1 & 1 & 0 \\ 0 & 1 & 5 & 8 & 9 & 7 & 5 & 3 & 2 & 1 & 1 \end{pmatrix}$$

On tape : `sum(A[k])$(k=0..10)`
On obtient : `1,1,2,3,5,7,11,15,22,30,42`

On peut faire un autre programme qui renverra la matrice B qui sera tel que : si $k \leq j$, $A[j,k]$ represente le nombre de partitions de j tel que $k >= \lambda_1 \geq \lambda_2 \geq ...\lambda_m > 0$. Autrement dit les lignes de B sont les sommes partielles des lignes de A : $B[j,k] = \sum_{m=0}^{k} A[j,m]$ donc

$$B[j,k] = B[j-k,k] + B[j,k-1]$$

On a donc $p(j) = B[j,j]$.
On tape :

```
partition(n):={
  local B,j,k,m,S;
  B:=idn(n+1);
  pour k de 1 jusque n faire
    B[0,k]:=1;
  fpour;
  pour j de 1 jusque n faire
    pour k de 1 jusque j faire
      B[j,k]:=B[j-k,k]+B[j,k-1];
    fpour;
```

```
    pour k de j+1 jusque n faire
      B[j,k]:=B[j,j];
    fpour;
  fpour;
  retourne B;
}
:;
```

On tape : `B:=partition(10)`
On obtient :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 0 & 1 & 3 & 4 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 0 & 1 & 3 & 5 & 6 & 7 & 7 & 7 & 7 & 7 & 7 \\ 0 & 1 & 4 & 7 & 9 & 10 & 11 & 11 & 11 & 11 & 11 \\ 0 & 1 & 4 & 8 & 11 & 13 & 14 & 15 & 15 & 15 & 15 \\ 0 & 1 & 5 & 10 & 15 & 18 & 20 & 21 & 22 & 22 & 22 \\ 0 & 1 & 5 & 12 & 18 & 23 & 26 & 28 & 29 & 30 & 30 \\ 0 & 1 & 6 & 14 & 23 & 30 & 35 & 38 & 40 & 41 & 42 \end{pmatrix}$$

On remarquera que l'on retrouve la matrice A dans les diagonales montantes de la matrice B.
On tape : `tran(B)[10]`
ou on tape : `col(B,10)`
On obtient : `[1,1,2,3,5,7,11,15,22,30,42]` ou on tape : `B[j,10]$(j=0..10)`
On obtient : `1,1,2,3,5,7,11,15,22,30,42` On tape : `B:=partition(1000):;`
On obtient (Evaluation time : 972.05) enfin le résultat...
On tape : `B[200,1000]`
On obtient `3972999029388`
On tape : `B[1000,1000]`
On obtient `240614678640326224736921497279 91`
Remarque On peut aussi pour avoir une relation de récurrence considérer les les partitions $P(n,p)$ de n en une somme d'exactement p éléments $\lambda_1 \geq ... \geq \lambda_p$>0. On dit alors :
$P(n,p)$ =(nombre de partage de n tels que $\lambda_p = 1$)+(nombre de partage de n tels que $\lambda_p > 1$).
On a :
(nombre de partage de n tels que $\lambda_p = 1$)=(nombre de partage de $n-1$ en $p-1$ λ_j)=$P(n-1,p-1)$
(nombre de partage de n tels que $\lambda_p > 1$)=(nombre de partage de $n-p$ en p λ_j)=$P(n-p,p)$

$$P(n,p) := P(n-1,p-1) + P(n-p,p)$$

avec $P(n,0) = 0$, $P(n,n) = 1$ et $P(n,p) = 0$ si $p > n$
On tape et on renvoie la somme des lignes de P :

```
partage(n):={
```

```
  local P,j,k,m,S;
  P:=idn(n+1);
  pour k de 1 jusque n faire
    P[k,0]:=0;
  fpour;
  pour j de 1 jusque n faire
    pour k de 1 jusque j faire
      P[j,k]:=P[j-1,k-1]+P[j-k,k];
    fpour;
  fpour;
  S:=[];
  pour k de 1 jusque n faire
    S[k]:=sum(row(P,k));
  fpour;
  retourne S;
}:;
```

On tape : `P:=partage(1000):;`
On obtient (Evaluation time : 455.81) enfin le résultat...mais c'est deux fois plus rapide qu'avec `B:=partition(1000):;`
On tape : `P[200]`
On obtient `3972999029388`
On tape : `P[1000]`
On obtient `24061467864032622473692149727991`

relation entre P et A :
On a :
$P[n,p] = P[n-1,p-1] + P[n-p,p]$
$P[n-1,p-1] = P[n-2,p-2] + P[n-p,p-1]$....
donc

$$P[n,p] = P[n-p,0] + \sum_{j=1}^{p} P[n-p,j]$$

avec $P[n,0] = 0$, $P[n,n] = 1$ et $P[n,p] = 0$ si $p > n$
Si on compare avec les relations obtenus pour A :

$$A[n,k] = \sum_{j=1}^{k} A[n-k,j] \text{ et}$$

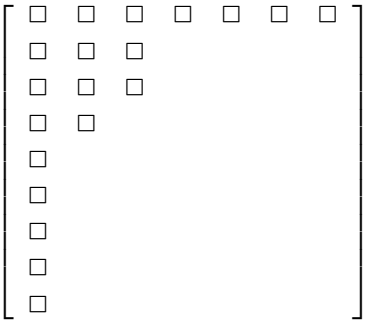
$$A[n,n] = 1 \text{ et } A[n,0] = 0 \text{ si } n > 0$$

On a les mêmes relations de récurrence, donc :

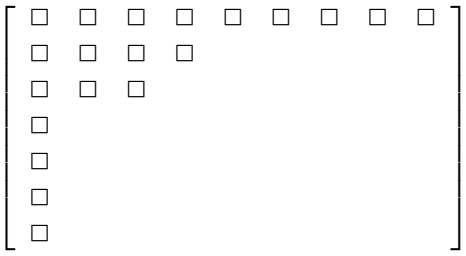
$$P[n,p] = A[n,p]$$

Représentation de Young
On peut voir facilement cette égalité, grâce à la repésentation de Young qui repésente par exemple le partage :
20=7+3+3+2+1+1+1+1+1
par le tableau formé par 20 carrés disposés selon 9 lignes :

7 carrés sur la 1-ière ligne,
3 carrés sur la 2-ième ligne,
3 carrés sur la 3-ième ligne,
2 carrés sur la 4-ième ligne,
1 carré sur de la 5-ième jusqu'à la 9-ième ligne :



Le nombre p de lignes correspond a un partage en une somme de p éléments et le plus grand élément m de ce partage est le nombre d'éléments de la première ligne.
Si maintenant on échange les lignes et les colonnes (on prend le transposé de ce tableau) on obtient une transformation qui au partage :
20=7+3+3+2+1+1+1+1+1 en une somme de 9 termes ayant comme plus grand élément 7 fait correspondre le partage :
20=9+4+3+1+1+1+1+1 en une somme de 7 termes ayant comme plus grand élément 9 :



3. **Notations**
1 sera noté 1_1, 1+1 sera noté 2_1, 1+1+1 sera noté 3_1 etc...
2 sera noté 2_2, 2+2 sera noté 4_2, 2+2+2 sera noté 6_2 etc...
Le développement en série de ;
$\frac{1}{1-x^1}$ sera noté $1 + x^{1_1} + x^{2_1} + ... + x^{n_1} + ...$
$\frac{1}{1-x^2}$ sera noté $1 + x^{2_2} + x^{4_2} + ... + x^{2n_2} + ...$
$\frac{1}{1-x^3}$ sera noté $1 + x^{3_3} + x^{6_3} + ... + x^{3n_3} + ...$
.....
$\frac{1}{1-x^p}$ sera noté $1 + x^{p_p} + x^{2p_p} + ... + x^{pn_p} + ...$
Lorsque l'on fait le produit de ces développements en série le terme en x^j a pour coefficient $p(j)$ car :
$x^j = x^{j_1} = x^{(j-2)_1}x^{2_2} = x^{(j-4)_1}x^{4_2}$ ...
$x^j = x^{(j-2)_{j-2}}x^{2_1} = x^{(j-2)_{j-2}}x^{2_2} = x^{(j-1)_{j-1}}x^{1_1} = x^{j_j}$
On tape : `P:=truncate(product([(1-x^j)$(j=1..20)]),20)`
On obtient : `-x^15-x^12+x^7+x^5-x^2-x+1`
On tape : `A:=truncate(series(1/P,x=0,20),20)`

On obtient :
```
627*x^20+490*x^19+385*x^18+297*x^17+231*x^16+176*x^15+
135*x^14+101*x^13+77*x^12+56*x^11+42*x^10+30*x^9+22*x^8+
15*x^7+11*x^6+7*x^5+5*x^4+3*x^3+2*x^2+x+1
```
On tape :`symb2poly(A)`
ou on tape : `coeff(A,x)`
On obtient la liste $p(20), p(19), .., p(1), p(0)$:
```
[627,490,385,297,231,176,135,101,77,56,42,30,22,15,11,
7,5,3,2,1,1]
```
On tape :`lcoeff(A)`
On obtient $p(20)$: `627`

4. On tape mais cela n'est pas très efficace lorsque n est grand :

```
partitioner(n):={
local A,P;
P:=truncate(product([(1-x^j)$(j=1..n)]),n);
A:=truncate(series(1/P,x=0,n),n);
return revlist(symb2poly(A));
}:;
```

On tape : `partitioner(20)`
On obtient la liste $p(20), p(19), .., p(1), p(0)$:
```
[627,490,385,297,231,176,135,101,77,56,42,30,22,15,11,
7,5,3,2,1,1]
```

9.1.3 Une méthode plus rapide

On utilise le théorème du nombre pentagonal d'Euler. Ce théorème donne une relation de récurrence entre $p(n)$ et $p(j)$ pour $j < n$ avec la convention que $p(j) = 0$ lorsque $j < 0$, $p(0) = 1$. Cette relation est :

$$p(n) = p(n-1)+p(n-2)-p(n-5)-p(n-7)+p(n-12)+p(n-15)-p(n-22)-p(n-26)....$$

$$p(n) = \sum_{m>0} (-1)^{m+1} p(n - m(3m-1)/2) + (-1)^{m+1} p(n - m(3m+1)/2)$$

Les nombres $m(3m-1)/2$ et $m(3m+1)/2$ sont les nombres pentagonaux généralisés. On tape :

```
partition_euler(n):={
local sg,s1,s2,m,k,j,p;
p:=[1];
pour j de 1 jusque n faire
m:=1;
sg:=1;
s1:=0;
k:=j-m*(3*m-1)/2;
tantque k>=0 faire
s1:=s1+sg*p[k]
sg:=-sg;
```

```
m:=m+1;
k:=j-m*(3*m-1)/2;
ftantque;
m:=1;
sg:=1;
s2:=0;
k:=j-m*(3*m+1)/2;
tantque k>=0 faire
s2:=s2+sg*p[k]
sg:=-sg;
m:=m+1;
k:=j-m*(3*m+1)/2;
ftantque;
p[j]:=s1+s2;
fpour;
retourne p;
}:;
```

On tape :
`partition_euler(20)`
On obtient :
`[1,1,2,3,5,7,11,15,22,30,42,56,77,101,135,176,231,297,385,490,627]`
On tape :
`P:=partition_euler(1000)`
On obtient la liste après 2.77s
On tape : `P[200]`
On obtient `3972999029388`
On tape : `P[1000]`
On obtient `24061467864032622473692149727991`

9.1.4 Estimation asymptotique

Ramanjan et Hardy ont trouvé une approximation de $p(n)$ qui est :
$p(n) \simeq \frac{e^{\pi\sqrt{\frac{2n}{3}}}}{4n\sqrt{3}}$ quand n tend vers $+\infty$.
On tape :

```
P(n):=exp(pi*sqrt(2n/3))/(4n*sqrt(3))
floor(evalf(P(1000),31))
```

On obtient :
`2440199631680247628826341494290 4` au lieu de
`24061467864032622473692149727991`

9.2 Un exercice de combinatoire et son programme

9.2.1 L'énoncé

Un jury est composé de p personnes que l'on choisit parmi n personnes : h hommes et f femmes ($h + f = n$).

Application numérique : $p = 10, n = 17, h = 9, f = 8$.

1. Écrire un programme qui renvoie la liste contenant les différents jurys possibles.
2. On impose comme contrainte que Monsieur Y et Madame X ne doivent pas se trouver ensemble.
   Écrire un programme qui renvoie la liste contenant tous les jurys respectant cette contrainte.
3. On veut que le jury respecte la parité "homme-femme".
   Pour cela, si $p = 2 * k$ ou $p = 2 * k + 1$, on suppose $f \geq k$ et $h = n - f \geq p - k$.
   Écrire un programme qui renvoie la liste contenant les jurys ayant $p - k$ hommes et k femmes.
4. Avec l'hypothèse précédente, écrire un programme qui renvoie la liste contenant les jurys respectant la parité "homme-femme" et respectant la contrainte Monsieur Y et Madame X ne doivent pas se trouver ensemble.
5. Rappels Avec `Xcas`, la commande `comb(n,p)` renvoie le nombre de combinaisons de p objets pris parmi n.
   Avec `Xcas`, la commande `convert(k,base,2)` renvoie la liste K de longueur l vérifiant $k = \sum_{j=0}^{l-1} K[j] * 2^j$. `revlist(K)` renvoie alors l'écriture en base 2 de k.

9.2.2 Les programmes

1. On peut avoir `comb(n,p)` jurys différents.
   On tape : `comb(17,10)`
   On obtient : `19448`
   Supposons que n cases C numérotées de 0 à $n-1$ représentent les membres du jury.
   Dans ces cases on met 1 si la personne fait partie du jury et 0 sinon. Un jury est donc une liste de 0 et de 1 et cela fait penser à l'écriture en base 2 d'un entier.
   Avec `Xcas`, la commande `convert(k,base,2)` renvoie la liste K de longueur l vérifiant $k = \sum_{j=0}^{l-1} K[j] * 2^j$ On va donc considérer un jury comme un entier dont l'écriture en base 2 a au plus n chiffres et comporte exactement p 1. Pour faire le programme on va repésenter chaque jury par un nombre dont l'écriture en base 2 comporte au plus n chiffres dont p 1 et pas plus de $n - p$ zéros.
   Le plus petit nombre correspondant à un jury est :
   `m=sum(2^j,j,0,p-1)=2^p-1`
   et le plus grand nombre correspondant à un jury est :
   `M=sum(2^j,j,n-p,n-1)=2^n-2^(n-p)`
   car les n cases sont numerotés de 0 à $n - 1$.
   On peut initialiser la liste `L` soit avec `L:=makelist(0,1,1);` soit avec `L:=makelist(0,1,comb(n,p));` puisque l'on connait sa longueur.
   On écrit le programme `jurys(n,p)` :

```
jurys(n,p):={
local j,k,L,m,M;
//L:=makelist(0,1,comb(n,p));
L:=makelist(0,1,1);
k:=0;
m:=2^p-1;
M:=2^n-2^(n-p);
for (j:=m;j<=M;j++){
if (sum(convert(j,base,2))==p){
L[k]=<j;
k:=k+1;
};
}
return L;
}
:;
```

On tape : `J:=jurys(17,10):;`
On obtient, au bout d'environ 11s, une liste J de taille 19448 et contenant des entiers qui lorsqu'on les décompose en base 2 donne la composition du jury.
On tape : `jurytire0:=J[rand(19448)]`
On obtient par exemple : `127255`
On tape : `convert(127255,base,2)`
On obtient par exemple : `[1,1,1,0,1,0,0,0,1,0,0,0,1,1,1,1,1]`
Donc, les personnes de numéros : `0,1,2,4,8,12,13,14,15,16` ont été tirées au sort pour faire partie du jury.
On peut bien sûr taper directement :
`jurytire0:=convert(J[rand(19448)],base,2)`
On obtient par exemple : `[1,0,0,0,1,1,1,0,1,1,1,0,0,1,0,1,1]`
Donc, les personnes de numéros : `0,4,5,6,8,9,10,13,15,16` ont été tirées au sort pour faire partie du jury.

2. On peut supposer que la case 0 représente Monsieur Y et que la case $n-1$ représente Madame X. Donc le jury repésenté par le nombre j est valable si :
`irem(j,2)==0` ou si `j<2^(n-1)`.
On peut réecrire un programme `jurysXY(n,p)` qui utilise `jurys` ou un programme `juryXY(n,p)` qui n'utilise pas `jurys` ou utiliser la liste précédente `J` et rejeter les mauvais tirages.
   — Le programme `jurysXY(n,p)`.
   On peut initialiser la liste `L`, soit avec
   `L:=makelist(0,1,1);` soit avec
   `L:=makelist(0,1,comb(n,p)-comb(n-2,p-2));` puisque l'on connait sa longueur.

```
jurysXY(n,p):={
local LP,L,j,k,l;
LP:=jurys(n,p);
s:=comb(n,p);
```

```
l:=s-comb(n-2,p-2);
//L:=makelist(0,1,1);
L:=makelist(0,1,l);
l:=0;
for(k:=0;k<s;k++){
j:=LP[k];
if (irem(j,2)==0 or j<2^(n-1)) {
L[l]=<j;
l:=l+1;
}
}
return L;
}:;
```

On tape : `JXY:=jurysXY(17,10)`
On a : `size(JXY)=13013`
On tape : `jurytire1:=convert(JXY[rand(13013)],base,2)`
On obtient par exemple : `[0,1,1,0,1,1,1,1,1,0,0,0,1,0,1,0,1]`
Mais ce programme appelle le programme `jurys` précédent....

— Un programme `juryXY(n,p)` qui n'utilise pas `jurys`
Les nombres qui correspondent aux jurys ne contenant pas X et Y en même temps ont dans la décomposition en base 2 soit le coefficient de 2^{n-1} nul, soit est un nombre pair avec le coefficient de 2^{n-1} égal à 1 donc égal à $2 * j + 2^{n-1}$.
Les nombres qui ont p 1 dans leur décomposition en base 2 et le coefficient de 2^{n-1} nul, vont de $m = 2^p - 1$ à $M = 2^(n-1) - 2^(n-p-1)$.
Les nombres pairs de la forme $2 * j + 2^{n-1}$ ont j qui a $p-1$ 1 dans sa décomposition en base 2. Les j vont donc de $2^p - 2$ à $2^(n-1) - 2^(n-p)$ par pas de 2.
On peut initialiser la liste `L` soit avec `L:=makelist(0,1,1);` soit avec `L:=makelist(0,1,comb(n,p)-comb(n-2,p-2));` puisque l'on connait sa longueur.On tape le programme `juryXY(n,p)`

```
juryXY(n,p):={
local L,j,k,ls,m,M;
s:=comb(n,p);
l:=s-comb(n-2,p-2);
L:=makelist(0,1,l);
//L:=makelist(0,1,1);
k:=0;
m:=2^p-1;
M:=2^(n-1)-2^(n-p-1);
for(j:=m;j<=M;j++){
if (sum(convert(j,base,2))==p){
L[k]=<j;
k:=k+1;
}
}
M:=2^(n-1)-2^(n-p);
for(j:=m-1;j<=M;j:=j+2){
```

```
if (sum(convert(j,base,2))==p-1){
L[k]=<j+2^(n-1);
k:=k+1;
}
}
return L;
};
```

On tape : `JXY:=juryXY(17,10)`
On a : `size(JXY)=13013`
On tape : `jurytire2:=convert(JXY[rand(13013)],base,2)`
On obtient par exemple : `[1,1,1,0,0,1,1,0,0,0,1,1,1,1,0,1]`

— programme `jurytireXY(J)` qui se sert de la liste `J` obtenue précédemment (`J:=jurys(17,10):;`)
On tape :

```
jurytireXY(J):={
local k,n,(s:=size(J));
n:=size(convert(J[s-1],base,2));
k:=1+2^(n-1);
while (irem(k,2)!=0 and k>=2^(n-1)){
k:=J[rand(s)];
}
return k;
}
```

On tape : `jurytireXY(J)` On obtient par exemple : `54762`
On tape : `convert(109242,base,2)`
On obtient : `[0,1,0,1,1,1,0,1,0,1,0,1,0,1,0,1,1]`

3. On peut supposer que les cases de 0 à $h-1$ représentent les hommes et que les cases de h à $n-1$ représentent les femmes. On écrit un nouveau programme `jury55(h,f,p)` qui utilise le programme `jurys(n,p)` écrit précédemment ou on se sert de la liste `J` trouvée précedemment.

— Le programme `jury55(h,f,p)`.
On peut initialiser la liste `L` soit avec
`L:=makelist(0,1,1);` soit avec
`L:=makelist(0,1,comb(h,5)*comb(f,5));`
puisque l'on connait sa longueur.

```
jury55(h,f,p):={
local L,j,k,M,F,l,p2;
p2:=iquo(p,2);
L:=makelist(0,1,1);
//L:=makelist(0,1,comb(h,5)*comb(f,5));
M:=jurys(h,p-p2);
F:=jurys(f,p2)*2^h;
l:=0;
for(j:=0;j<comb(h,p-p2);j++){
for(k:=0;k<comb(f,p2);k++){
L[l]=<M[j]+F[k];
l:=l+1;
```

```
}
}
return L
}:;
J55:=jury55(9,8,10)
size(J55)=7056
```
On tape : `jurytire3:=convert(J55[rand(7056)],base,2)`
On obtient par exemple : `[1,0,1,0,1,0,1,0,1,1,1,1,1,1]`

— Le programme `jurytire55(J,h)` qui se sert de la liste `J` obtenue précédemment :
On tape :
```
jurytire55(J,h):={
local k,n,p,la,j,(s:=size(J));
la:=convert(J[s-1],base,2);
n:=size(la);
p:=sum(la);
k:=iquo(p,2);
j:=1;
while (sum(convert(irem(j,2^h),base,2))!=p-k){
j:=J[rand(s)];
}
return j;
}
```
On tape : `jurytire4:=convert(jurytire55(J,9),base,2)`
On obtient par exemple : `[1,1,1,0,1,0,0,0,1,1,1,1,1,0,0,1]`

4. On peut supposer que la case 0 représente Monsieur Y et que la case $n-1$ représente Madame X. Donc le jury repésenté par le nombre j est valable si :
`irem(j,2)==0` ou si `j<2^(n-1)`.
On peut réecrire un programme `jury55XY(h,f,p)` qui utilise `jury55`, ou un programme `jurys55XY(n,p)` qui n'utilise pas `jury55` mais `jurys`, ou utiliser la liste précédente `J55` et rejeter les mauvais tirages, ou encore utiliser la liste `J` du début et rejeter les mauvais tirages.
— Le programme `juryXY55(h,f,p)` qui utilise `jury55`.
On peut initialiser la liste `L` soit avec
`L:=makelist(0,1,1);` soit avec
`L:=makelist(0,1,comb(h,p-p2)*comb(f-1,p2-1));`
puisque l'on connait sa longueur.
```
juryXY55(h,f,p):={
local LP,L,j,k,l,p2;
p2:=iquo(p,2);
LP:=jury55(h,f,p);
//L:=makelist(0,1,1);
s:=comb(h,p-p2)*comb(f,p2);
l:=s-comb(h-1,p-p2-1)*comb(f-1,p2-1);
L:=makelist(0,1,l);
```

```
l:=0;
for(j:=0;j<s;j++){
k:=LP[j];
if (irem(k,2)==0 or k<2^(h+f-1)) {
L[l]=<k;
l:=l+1;
}
}
return L;
}:;
```

On tape : `JXY55:=juryXY55(9,8,10)`
On a : `size(JXY55)=4606`
On tape : `jurytire5:=convert(JXY[rand(40606)],base,2)`
On obtient par exemple : `[1,0,0,0,0,1,1,1,1,1,1,0,1,1,0,1]`
Mais ce programme appelle le programme `jurys` précédent....

— Un programme `jurys55XY(n,p)` utilise `jurys`
On tape le programme `jurys55XY(h,f,p)`.
On peut initialiser la liste `L` soit avec
`L:=makelist(0,1,1);` soit avec

```
L:=makelist(0,1,comb(h,p-p2)*comb(f-1,p2)+
              comb(h-1,p-p2)*comb(f-1,p2-1));
```

puisque l'on connait sa longueur.

```
jurys55XY(h,f,p):={
local L,j,k,M,F,l,p2;
p2:=iquo(p,2);
L:=makelist(0,1,1);
//L:=makelist(0,1,comb(h,p-p2)*comb(f-1,p2)+
               comb(h-1,p-p2)*comb(f-1,p2-1));
M:=jurys(h,p-p2);
F:=jurys(f-1,p2)*2^h;
l:=0;
for(j:=0;j<comb(h,p-p2);j++){
for(k:=0;k<comb(f-1,p2);k++){
L[l]=<M[j]+F[k];
l:=l+1;
}
}
M:=jurys(h-1,p-p2);
F:=jurys(f-1,p2-1)*2^h;
for(j:=0;j<comb(h-1,p-p2);j++){
for(k:=0;k<comb(f-1,p2-1);k++){
L[l]=<2*M[j]+F[k]+2^(h+f-1);
l:=l+1;
}
}
return L
}:;
```

On tape : `JJXY55:=jurys55XY(9,8,10)`

On a : `size(JJXY55)=4606`
On tape : `jurytire6:=convert(JJXY55[rand(4606)],base,2)`
On obtient par exemple : `[0,0,0,1,1,0,1,1,1,1,1,1,0,1,0,0,1]`

— Le programme `jurytireXY55(J55)` qui se sert de la liste `J55` obtenue précédemment (`J55:=jury55(h,f,p)`) :
On tape :

```
jurytireXY55(J55):={
local n,k,(s=size(J55));
n:=size(convert(J55[s-1],base,2));
k:=1+2^(n-1);
while (irem(k,2)!=0 and k>=2^(n-1)){
k:=J55[rand(s)];
}
return k;
}
```

On tape : `jurytire()` On obtient par exemple : `63798`
On tape : `convert(63798,base,2)`
On obtient : `[0,1,1,0,1,1,0,0,1,0,0,1,1,1,1,1]`

— Le programme `jurytireXY5(J,h)` qui se sert de la liste `J` obtenue précédemment (`J:=jurys(n,p)`) :
On tape :

```
 jurytireXY5(J,h):={
local n,k,la,p,(s=size(J));
la:=convert(J[s-1],base,2);
n:=size(la);
p:=sum(la);
k:=iquo(p,2);
j:=1;
while ((sum(convert(irem(j,2^h),base,2))!=p-k) or
       (irem(j,2)!=0 and j>=2^(n-1))){
j:=J[rand(s)];
}
return j;
}
```

On tape : `jurytireXY5(J,9)`
On obtient par exemple : `24359`
On tape : `convert(24359,base,2)`
On obtient : `[1,1,1,0,0,1,0,0,1,1,1,1,1,0,1]`

9.3 Visualistion des combinaisons modulo 2

9.3.1 L'énoncé

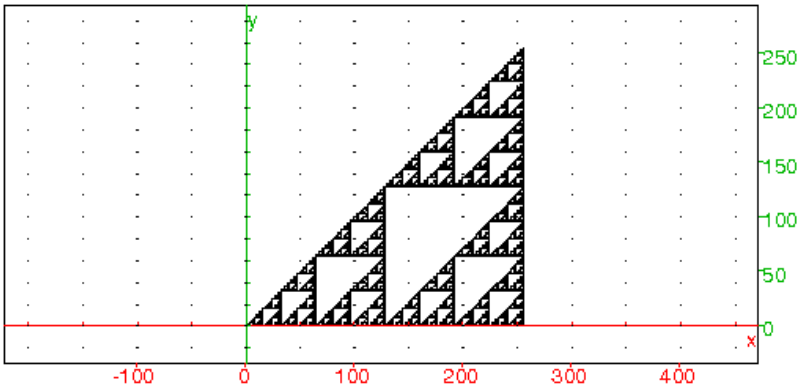
On veut visualiser les points k, j ($j \leq k$) tel que $comb(k, j) = 0 \bmod 2$

9.3.2 Le programme

On tape :

```
dessincomb(n):={
  local j,k,L;
  L:=NULL;
  pour j de 1 jusque n faire
    pour k de j jusque n faire
      si irem(comb(k,j),2) alors
        L:=L,point(k,j,affichage=point_point);
      fsi;
    fpour;
  fpour;
  retourne L;
};
```

On tape :
`dessincomb(255)`
On obtient :



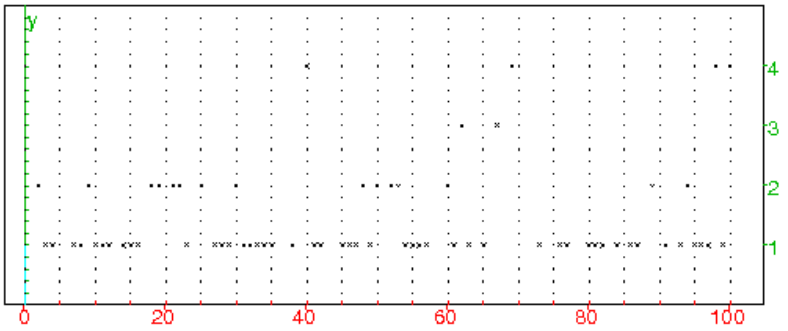
9.4 Un exercice

On tire 100 fois de suite au hasard de façon équiprobable un entier p parmi $1, 2..100$ et on note n_p le nombre de fois où il a été tiré.
Écrire un programme qui représente les points p, n_p On tape :

```
tirage():={
local p,np,j,L,P;
L:=0$(j=1..100)
pour j de 0 jusque 99 faire
p:=rand(100);
L[p]:=L[p]+1;
fpour;
P:=NULL;
pour p de 0 jusque 99 faire
```

```
np:=L[p];
si np!=0 alors P:=P,point(p+1+i*np); fsi;
fpour;
print(max(L));
retourne P;
}:;
```

On tape `tirage()`
On obtient :



9.5 Valeur de e et le hasard

9.5.1 L'énoncé

On tire des nombres au hasard dans $[0;1[$ et on les ajoute. On s'arrête quand la somme obtenue est strictement plus grande que 1. On peut montrer que le nombres de tirages nécessaires est en moyenne égal à e. Écrire un programme vérifiant expérimentalement ce résultat.

9.5.2 La solution avec `Xcas`

Dans `Xcas`, on peut soit utiliser `rand(0,1)`, soit utiliser la fonction `f:=rand(0..1)` pour obtenir un nombre au hasard entre 0 et 1.
On tape :`rand(0.1)`
On obtient : `0.482860322576`
On tape : `f:=rand(0..1)`
puis : `f()` On obtient : `0.8627617261373`
puis : `f()` On obtient : `0.3095522336662` etc....
On écrit le programme :

```
approxe(n):={
local j,S,k,N;
N:=0;
pour k de 1 jusque n faire
S:=0;
j:=0;
tantque S<=1 faire
S:=S+rand(0,1);
```

```
j:=j+1;
ftantque;
N:=N+j;
fpour;
retourne evalf(N/n);
}:;
```

On tape : `approxe(100000)`
On obtient : `2.71750000000`
alors que `evalf(e)=2.71828182846`

9.6 Distance moyenne entre de 2 points

9.6.1 L'énoncé

On veut évaluer la distance moyenne entre 2 points choisis au hasard : dans un segment S de longueur 1, dans le carré unité C ou dans le cube unité K.

1. Écrire un programme qui calcule cette distance moyenne lorsque l'on choisit au hasard n fois de suite 2 points dans S,dans C ou dans K, puis généraliser.
2. Faire une représentation graphique permettant de visualiser la répartition des distances dans le cube unité K.

9.6.2 La solution avec `Xcas`

1. ```
   distS(n):={
   local xa,xb,j,d,D;
   D:=0;
   pour j de 1 jusque n faire
   xa:=rand(0,1);
   xb:=rand(0,1);
   d:=abs(xa-xb);
   D:=D+d;
   fpour;
   retourne evalf(D/n);
   }:;

   distC(n):={
   local xa,ya,xb,yb,j,d,D;
   D:=0;
   pour j de 1 jusque n faire
   xa:=rand(0,1);ya:=rand(0,1);
   xb:=rand(0,1);yb:=rand(0,1);
   d:=distance([xa,ya],[xb,yb]);
   D:=D+d;
   fpour;
   retourne evalf(D/n);
   }:;
   ```
   ```

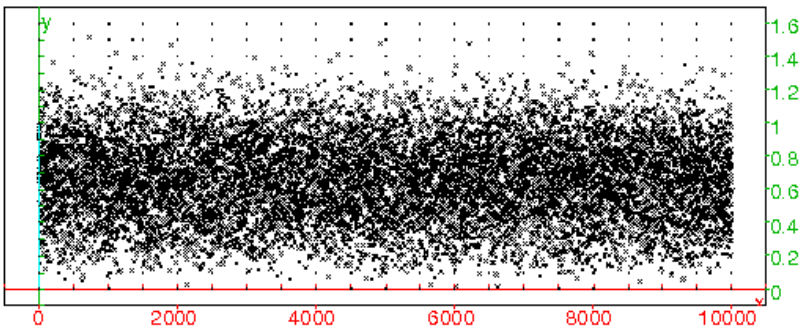
```
distK(n):={
local xa,ya,za,xb,yb,zb,j,d,D;
D:=0;
pour j de 1 jusque n faire
xa:=rand(0,1);ya:=rand(0,1);za:=rand(0,1);
xb:=rand(0,1);yb:=rand(0,1);zb:=rand(0,1);
d:=distance([xa,ya,za],[xb,yb,zb]);
D:=D+d;
fpour;
retourne evalf(D/n);
}:;

distp(p,n):={
local a,b,j,d,D;
D:=0;
pour j de 1 jusque n faire
a:=op(randMat(1,p,0..1));
b:=op(randMat(1,p,0..1));
d:=sqrt(sum((a-b)^2));
D:=D+d;
fpour;
retourne evalf(D/n);
}:;
```

On tape : `distS(100000)`
On obtient : `0.333424530881`
On tape : `distC(100000)`
On obtient : `0.522140638365`
On tape : `distK(100000)`
On obtient : `0.6627691590747`
On tape : `distp(4,100000)`
On obtient : `0.777235961163`
On tape : `distp(5,100000)`
On obtient : `0.877837288394`

2.
```
graphdist(n):={
local xa,ya,za,xb,yb,zb,j,d,L;
L:=NULL;
pour j de 1 jusque n faire
xa:=rand(0,1);ya:=rand(0,1);za:=rand(0,1);
xb:=rand(0,1);yb:=rand(0,1);zb:=rand(0,1);
d:=distance([xa,ya,za],[xb,yb,zb]);
L:=L,point(j+i*d);
fpour;
retourne L;
}:;
```

On tape : `graphdist(10000)`
On obtient :



Chapitre 10

Les graphes et l'algorithme de Dijkstra

Soit un graphe ayant n sommets numérotés de 0 à $n-1$. Certains de ces sommets sont reliés par des arêtes de longueur données.
L'algorithme de Dijkstra permet de trouver le chemin de distance minimale qui relie le sommet de numéro `deb` aux autres sommets.
Cet algorithme procéde de proche en proche en se servant de la remarque :
si par exemple, le chemin le plus court pour aller du sommet 0 au sommet 2 passe par le sommet 4, le début de ce chemin est aussi le chemin le plus court pour aller du sommet 0 au sommet 4.

10.1 L'algorithme sur un exemple

Soient $n = 4$ et le graphe de matrice `A` :

$$A = \begin{pmatrix} 0 & 27 & 11 & 23 \\ 27 & 0 & 14 & 1 \\ 11 & 14 & 0 & 10 \\ 23 & 1 & 10 & 0 \end{pmatrix}$$

Je veux partir du sommet 0 pour aller au sommet 1 (resp 2,3) par le plus court chemin.
J'initialise :
`dist` à `[0,27,11,23]` (c'est à dire à la ligne 0 de `A`) `afaire` à `[1,2,3]` et `sprec` à `[-1,0,0,0]` (`sprec[0]=-1` car on part du sommet 0 et pour `j!=0`, `sprec[j]=0` car `dist` est la distance provisoire minimum pour aller de 0 à `j`).
Les 2 valeurs `dist[0]` et `sprec[0]` sont définitives car le sommet 0 est ici le départ. Je sais aussi que le chemin le plus court du sommet 0 au sommet 1 (resp 2,3) ne repassera pas par le sommet 0.
Première étape
Je cherche le minimum des valeurs de `dist` pour les sommets dont les numéros sont dans `afaire=[1,2,3]` c'est à dire le minimum de `[27,11,23]`.
Le minimum 11 de `[27,11,23]` est réalisé pour le sommet 2 (car `dist[2]=11`).
Je supprime le numéro 2 de la liste `afaire` qui devient `afaire=[1,3]` car je

connais maintenant le plus court chemin pour aller de 0 à 2, il est direct et a pour longueur 11 je ne change donc pas la valeur de `dist[2]` ni celle de `sprec[2]` : ces 2 valeurs sont maintenant définitives.
On a donc encore `dist=[0,27,11,23]` et `sprec=[-1,0,0,0]` où les valeurs d'indice 0 et 2 sont définitives.
Maintenant, le chemin provisoirement le plus court allant du sommet 0 au sommet 1 (resp du sommet 0 au sommet 3) est :

- soit le chemin direct allant de 0 à 2, suivi par le chemin le plus court allant de 2 à 1 (resp de 2 à 3),
- soit le chemin le plus court allant de 0 à 1 (resp de 0 à 3) sans passer pas par 2.

L'étape suivante consiste donc à comparer le chemin le plus court de 0 à 2 de longueur `dist[2]`, suivi par le chemin direct de 2 à 1 de longueur `A[2,1]` (resp 3 de longueur `A[2,3]`) avec le chemin le plus court provisoire qui va de 0 à 1 `dist[1]` (respde 0 à 3 `dist[3]`).
Je compare donc `27=dist[1]` à 11+14=25 (11=`dist[2]`=longueur du chemin minimum allant de 0 à 2 et `14=A[2,1]`=longueur du chemin direct allant de 2 à 1). On a 25<27 donc je modifie `dist` en `[0,25,11,23]` et `sprec` en `[-1,2,0,0]` puisque 25 est la longueur du chemin qui passe par 2.
Je compare donc `23=dist[3]` à 11+10=21 (11=longueur du chemin minimum allant de 0 à 2 et `10=A[2,3]`=longueur du chemin direct allant de 2 à 3). On a 21<23 donc je modifie `dist` en `[0,25,11,21]` et `sprec` en `[-1,2,0,2]` puisque 21 est la longueur du chemin qui passe par 2.
Donc maintenant `dist=[0,25,11,21]` et `sprec=[-1,2,0,2]`
Deuxième étape
Je cherche le minimum des valeurs de `dist` pour les sommets de numéros `afaire=[1,3]` c'est à dire le minimum de `[25,21]`.
Le minimum 21 de `[25,21]` est réalisé pour le sommet 3 car `dist[3]=21`. Je supprime le numéro 3 de la liste `afaire` qui devient `afaire=[1]` car je connais maintenant le plus court chemin pour aller de 0 à 3, il est de longueur 21 et il passe par 2 car `sprec[3]=2`.
Je cherche enfin le plus court chemin pour aller de 0 à 1 en empruntant :

- soit le chemin minimum allant de 0 à 2 de longueur 11 (car `dist[2]=11`), puis le chemin direct allant de 2 à 1 de longueur `14=A[2,1]` (donc un chemin de longueur 11+14=25=`dist[1]`),
- soit le chemin minimum qui va de 0 à 3 de longueur 21 (car `dist[3]=21`), puis le chemin direct allant de 3 à 1 de longueur 1=A[3,1] (donc un chemin de longueur 21+1=22).

Je compare donc 25 à 22. On a 22<25 donc je modifie `dist` en `[0,22,11,21]` et `sprec` en `[-1,3,0,2]`.
Donc maintenant `dist=[0,22,11,21]` et `sprec=[-1,3,0,2]`
Troisième étape
Il reste à chercher le minimum de `[22]` est obtenu pour le sommet de numéro 1, numéro que l'on supprime de la liste `afaire` qui devient vide.
Le résultat final est donc :
`dist=[0,22,11,21]` et `sprec=[-1,3,0,2]`

10.2 Déscription de l'algorithme de Dijkstra

Soit A la matrice donnant la longueur des arêtes i.e. A[j,k] est la longueur de l'arête reliant le sommet j au sommet k avec la convention de mettre A[j,k]=inf=$+\infty$ quand il n'y a pas d'arête qui relie le sommet j au sommet k.
Soientt dist un vecteur donnant les distances provisoires reliant le sommet de numéro deb aux autres sommets et sprec[j] le numéro du sommet précedent j par lequel on doit passer pour avoir la distance minimale provisoire.
Par exemple si n=5 et si en fin de programme sprec=[3,2,0,-1,2] cela veut dire que l'on part du sommet 3 car sprec[3]=-1.
Si on cherche le plus court chemin pour aller du sommet 3 au sommet j=4 le chemin sera 3,???,4. Mais comme sprec[4]=2 le chemin sera 3,??,2,4 puis, comme sprec[2]=0 et sprec[0]=3 le chemin sera 3,0,2,4.
Par contre le chemin minimum pour aller du sommet 3 à 0 sera direct de 3 à 0 puique sprec[0]=3.
afaire est la liste des indices restant à faire.
Initialisation :
Au début dist=A[deb] (A[deb] est la ligne d'indice deb de A) et sprec[deb]=-1 et sprec[j]=deb pour j!:=deb.
afaire est la liste des indices dans laquelle on a enlevé deb I.E. une liste de longueur n-1.
Etapes suivantes :
On cherche le minimum m des distances provisoires dist reliant le sommet deb aux sommets dont les numéros sont dans afaire et on note jm le numéro du sommet réalisant ce minimum. On supprime jm de la liste afaire.
On compare ensuite, pour tous les numéros des sommets restant afaire, la longueur autredist des chemins qui passent par jm à la valeur provisoire dist et si pour le sommet de numéro k=afaire[j] le chemin qui passe par jm est plus court on modifie dist[k] et on modifie sprec[k] qui vaut alors jm.
On recommence jusqu'a épuisement de la liste afaire , c'est à dire que l'on fait cela n-1 fois.
Remarque
Attention aux indices ! ! ! !
afaire est la liste des indices ou numéros des sommets restant à traiter et il ne faut pas confondre le numéro des sommets et les indices qu'ils ont dans la liste afaire i.e. ne pas confondre k=afaire[j] avec j (si afaire=[2,0,1] le sommet 0 a pour indice 1 dans afaire).

10.3 Le programme

```
dijkstra(A,deb):={
local j,k,n,na,dist,sprec,distaf,afaire,
      m,jm,autredist,jma;
// initialisation
n:=size(A);
dist:=A[deb];
sprec:=[deb $n];
sprec[deb]:=-1;
```

```
n:=n-1;
//afaire liste des indices restant a faire
afaire:=suppress([j$(j=0..n)],deb);
na:=size(afaire)-1;
pour k de 0 jusque n-1 faire
//le sommet jm realise la dist m minimum de distaf
//jma est l'indice de m dans la liste distaf
//jm est l'indice de m dans la liste afaire
distaf:=[dist[afaire[j]]$(j=0..na)];
m:=min(distaf);
//jma indice du minimum m dans afaire
jma:=member(m,distaf)-1;
//jm indice du minimum m dans dist
jm:=afaire[jma];
//fin prematuree
si m==inf alors return dist,sprec; fsi;
afaire:=suppress(afaire,jma);
na:=na-1;
  pour j de 0 jusque na faire
     autredist:=m+A[jm,afaire[j]];
     si autredist<dist[afaire[j]] alors
        dist[afaire[j]]:=autredist;
        sprec[afaire[j]]:=jm;
     fsi;
  fpour;
fpour;
retourne dist,sprec;
}:;
```

On tape :
`M:=[[0,27,11,23],[27,0,14,1],[11,14,0,10],[23,1,10,0]]`
`dijkstra(M,0)`
On obtient (cf la section précédente) :
`[0,22,11,21],[-1,3,0,2]`
On tape :
`A:=[[0,1,6,7],[1,0,4,3],[6,4,0,1],[7,3,1,0]] dijkstra(A,2)`
On obtient :
`[5,4,0,1],[1,2,-1,2]`
cela veut dire par exemple pour aller de 2 à 0 la distance minimale est 5 et le chemin est 2,1,0.
On tape :
`B:=[[0,1,6,7],[1,0,4,2],[6,4,0,1],[7,2,1,0]] dijkstra(B,0)`
On obtient :
`[0,1,4,3],[-1,0,3,1]`
cela veut dire par exemple pour aller de 0 à 2 la distance minimale est 4 et le chemin est 0,1,3,2.

10.4 Le chemin le plus court d'un sommet à un autre

On tape en utilisant le programme précédent :

```
dijkstra2(A,deb,fin):={
local dist,sprec,long,chemin,j;
dist,sprec:=dijkstra(A,deb);
long:=dist[fin];
j:=sprec[fin];
chemin:=fin;
tantque j!=-1 faire
chemin:=j,chemin;
j:=sprec[j];
ftantque;
retourne long,[chemin];
}:;
```

ou bien en arrétant le programme dès que l'on a atteint le sommet `fin`, on tape :

```
dijkstra3(A,deb,fin):={
local j,k,n,na,dist,sprec,distaf,afaire,m,
      jm,autred,jma,long,chemin;
n:=size(A);
//dist:=[inf$ n];dist[deb]:=0;
dist:=A[deb];
sprec:=[deb $n];
sprec[deb]:=-1;
n:=n-1;
//afaire liste des indices restant a faire
afaire:=suppress([j$(j=0..n)],deb);
na:=n-1;
pour k de 0 jusque n-1 faire
   //minimum des distances dist[afaire[j]]
   distaf:=[dist[afaire[j]]$(j=0..na)];
   m:=min(distaf);
   //jma indice du minimum m dans afaire
   jma:=member(m,distaf)-1;
   //jm indice du minimum m dans dist
   jm:=afaire[jma];
   si m==inf alors return dist,sprec; fsi;
   si jm==fin alors
      long:=dist[fin];
      chemin:=jm;
      j:=sprec[jm];
      tantque j!=-1 faire
        chemin:=j,chemin;
        j:=sprec[j];
      ftantque;
      retourne long,[chemin];
```

```
   fsi;
   afaire:=suppress(afaire,jma);
   na:=na-1;
   pour j de 0 jusque na faire
      autred:=m+A[jm,afaire[j]];
      si autred<dist[afaire[j]] alors
         dist[afaire[j]]:=autred;
         sprec[afaire[j]]:=jm;
      fsi;
   fpour;
fpour;
}:;
```

On tape :
`M:=[[0,27,11,23],[27,0,14,1],[11,14,0,10],[23,1,10,0]]`
`dijkstra3(M,0,1)`
On obtient (cf la section précédente) :
`22,[0,2,3,1]`
`dijkstra3(M,0,2)`
On obtient (cf la section précédente) :
`11,[0,2]`
`dijkstra3(M,0,3)`
On obtient (cf la section précédente) :
`21,[0,2,3]`
On tape :
`A:=[[0,1,6,7],[1,0,4,3],[6,4,0,1],[7,3,1,0]]`
`dijkstra2(A,2,0)` ou `dijkstra3(A,2,0)`
On obtient : `5,[2,1,0]`
On tape :
`B:=[[0,1,6,7],[1,0,4,2],[6,4,0,1],[7,2,1,0]]`
`dijkstra2(B,0,2)` ou `dijkstra3(B,0,2)`
On obtient :
`4,[0,1,3,2]`
On tape :
`dijkstra2(B,2,0)` ou `dijkstra3(B,2,0)`
On obtient :
`4,[2,3,1,0]`

Exemple avec une matrice créee aléatoirement

On tape :
```
MR:=randmatrix(5,5,'alea(50)+1')
M:=MR+tran(MR)
pour j de 0 jusque 4 faire M[j,j]=<0; fpour;
M
```
On obtient :
`[[0,47,91,57,60],[47,0,58,18,50],[91,58,0,22,74],`
`[57,18,22,0,70],[60,50,74,70,0]]`

$$\begin{pmatrix} 0 & 47 & 91 & 57 & 60 \\ 47 & 0 & 58 & 18 & 50 \\ 91 & 58 & 0 & 22 & 74 \\ 57 & 18 & 22 & 0 & 70 \\ 60 & 50 & 74 & 70 & 0 \end{pmatrix}$$

On tape :
`dijkstra(M,0)`
On obtient :
`[0,47,79,57,60],[-1,0,3,0,0]`
On tape :
`dijkstra2(M,0,2)`
On obtient :
`79,[0,3,2]`

Chapitre 11

Exercices sur trigonométrie et complexes

11.1 Les polynômes de Tchebychev

11.1.1 L'énoncé

Les polynômes de Tchebychev T_n sont tels que $\cos(nx) = T_n(\cos(x))$.
On a ainsi :
$T_0(x) = 1$
$T_1(x) = x$
$T_2(x) = 2x^2 - 1$

1. Montrer que pour $n \geq 1$ on a :
   $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$.
2. Écrire une fonction de n qui renvoie le polynôme T_n, en utilisant la relation
   $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$.
3. Écrire une fonction de n qui renvoie le polynôme T_n, en utilisant les nombres complexes et la formule de Moivre.

11.1.2 La solution avec `Xcas`

Dans `Xcas`, la fonction `tchebyshev1` qui renvoie le nième polynôme de Tchebyshev de 1ière espèce existe. Cela va pouvoir permettre la vérification de vos programmes

1. La relation $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$ est vraie pour $n = 1$ car $T_2(x) = 2xT_1(x) - T_0(x) = 2x * x - 2$ On a : $\cos((n+1)x) = \cos(x) * \cos(nx) - \sin(x) * \sin(nx)$ et
   $\cos((n-1)x) = \cos(x) * \cos(nx) + \sin(x) * \sin(nx)$
   donc $\cos((n+1)x) + \cos((n-1)x) = 2\cos(x) * \cos(nx)$
   ou encore $\cos((n+1)x) = 2\cos(x) * \cos(nx) - \cos((n-1)x)$ c'est à dire :
   $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$
2. On écrit une fonction `Tcheb(n)` qui renvoie le polynôme T_n, en utilisant la relation $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$. :

```
Tcheb(n):={
local j,T0,T1,Tj;
T0:=1;
T1:=x;
pour j de 2 jusque n faire
Tj:=2*x*T1-T0;
T0:=T1;
T1:=Tj;
fpour;
return T1;
}:;
```

On tape : `Tcheb(7)`
On obtient :`64*x^7-112*x^5+56*x^3-7*x`

3. On écrit une fonction `Tchebich(n)` qui renvoie le polynôme T_n, en utilisant la formule de Moivre ($\cos(nx) = re((\cos(x)+i\sin(x))^n)$) et l'égalité $\sin(x)^2 = 1-\cos(x)^2$:

```
Tchebich(n):={
local f;
f(x,y):=normal(re((x+i*y)^n));
return f(x,sqrt(1-x^2));
}:;
```

On tape : `Tchebich(7)`
On obtient :`64*x^7-112*x^5+56*x^3-7*x`

Remarque
On peut vérifier car cette fontion existe dans `Xcas`, on tape On tape : `tchebyshev1(7)`
On obtient :`64*x^7-112*x^5+56*x^3-7*x`

Chapitre 12

Codage

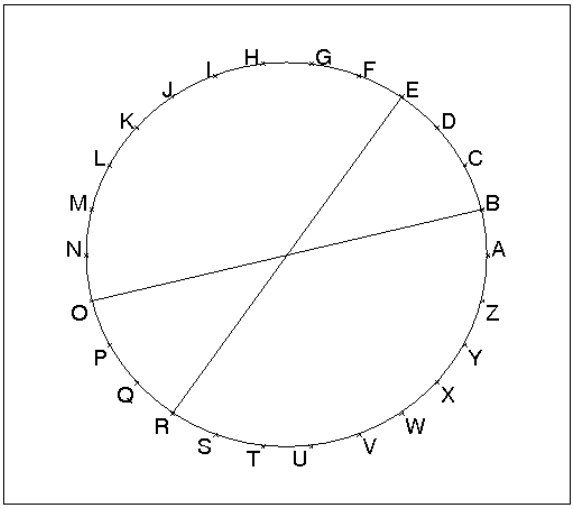
12.1 Codage de Jules Cesar

12.1.1 Introduction

Le principe est simple : on écrit un message en n'utilisant que les 26 lettres de l'alphabet et on le code en remplacant une lettre par une autre lettre. Ceci peut être considéré comme une application f de l'ensemble des lettres {A,B,C,...X,Y,Z} dans lui-même.
Pour pouvoir décoder, il faut que l'application f ci-dessus soit bijective !
Il parait que Jules César utilisait cette méthode pour communiquer ses ordres.

12.1.2 Codage par symétrie point ou par rotation d'angle π

Une façon simple de coder est la suivante :
on écrit les lettres sur un cercle (de façon équirépartie) et on remplace chaque lettre par la lettre symétrique par rapport au centre du cercle :



Le décodage s'obtient de la même façon car ici $f \circ f = Id$

12.1.3 Avec les élèves

On explique que l'on va coder un message en remplacant une lettre par une autre (on suppose que le message n'utilise que les 26 lettres de l'alphabet et que l'on n'écrit pas les espaces).
Pour cela on distribue une feuille sur laquelle figure quatre cercles divisés en 26 parties égales.
On écrit sur ce cercle les 26 lettres de l'alphabet et le codage consiste à remplacer chaque lettre du message par la lettre diamétralement opposéesur le cercle.
Par exemple voici le message à coder selon cette méthode :
"BONJOURLESAMIS"
Le message à décoder est donc :
"OBAWBHEYRFNZVF"
Quel sont les éléments pertinents qu'il faut transmettre pour que le décodage soit possible ?
Parmi les réponses il est apparu qu'il fallait ajouter $+13$ pour décoder.
Puis chaque élève invente un codage et écrit un message selon son codage et le donne à décoder à son voisin, par écrit avec les explications nécessaires pour le décoder.
Voici quelques codages obtenus :
- codage obtenu en remplacant chaque lettre par celle qui la suit dans l'alphabet,
- codage obtenu en remplacant chaque lettre par celle qui est obtenue en avançant de +4 sur la roue (ou en reculant de 3 etc...),
- codage obtenu en remplaçant chaque lettre par celle qui est obtenue par symétrie par rapport à la droite [A,N] (verticale sur le dessin),ou par symétrie par rapport à la droite horizontale sur le dessin,
- codage par symétrie par rapport au centre du cercle mais où l'ordre des lettres sur le cercle n'est pas respecté,
- d'autres codages comme de remplacer le message par une suite de nombres (intéressant mais cela ne repond pas à la question posée),
- codage qui dépend de la position de la lettre dans le message. Ce codage n'est pas une application puisque une même lettre peut avoir des codages différents.

Combien y-a-t-il de codages (i.e de bijections) possibles ?
Il a fallut parler de bijections :
un étudiant a dit qu'il fallait que deux lettres différentes soient codées par des lettres différentes pour que le décodage soit possible (injection).
Ceci entraine que toutes les lettres sont le codage d'une autre lettre (surjection).
Pour simplifier on a préféré parler de permutations des lettres avec comme exemple : trouver tous les codages possibles si on suppose que l'alphabet utilisé ne comporte que les trois lettres A, B, C.
Donnez un ordre de grandeur de 26 !
A supposer que vous vouliez écrire les 26 ! codages possibles sur un cahier et que votre rythme est de 1 codage par seconde (vous êtes super rapide ! ! !... félicitations ! ! !) combien de temps (réponse en heures, en mois, en années... ?) vous faut-il ?
Combien de cahiers de 10000 lignes (200 pages de 50 lignes) vous faut-il ? Donner la longueur occupée par ces cahiers dans une bibliothèque, si chaque cahier occupe 1 cm. Combien y-a-t-il de codages involutifs possibles qui sont sans point double

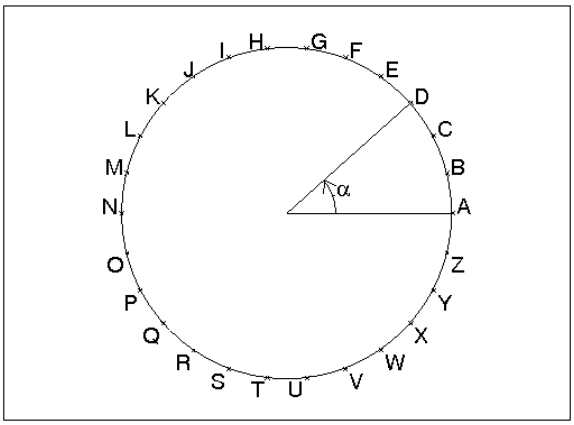
(c'est à dire de bijections f vérifiant $f = f^{-1}$ et $f(x) \neq x$ pour tout x) ?
Comment faire pour que la clé du décodage soit simple ?

12.1.4 Travail dans Z/26Z

Le codage de Jules César consiste à faire une symétrie point ou encore une rotation d'angle π.
Si on numérote les lettres de 0 à 25, le codage consiste donc à faire une addition de 13 (modulo 26). (voir figure 12.1.2).
Exemple :
"BONJOUR" sera codé par "OBAWBHE"

12.1.5 Codage par rotation d'angle $\alpha = k * \pi/13$

Le principe est le même :
on écrit les lettres sur un cercle (de façon équirépartie) et on remplace chaque lettre par la lettre obtenue par rotation d'angle $\alpha = k * \pi/13$ $(0 \leq k \leq 25)$ (k est un entier).
Si on numérote les lettres de 0 à 25, le codage consiste donc à faire subir à chaque lettre un décalage de `k` c'est à dire à faire subir à son numéro une addition de `k` (modulo 26) (voir figure 12.1.5).
On remarquera que si le paramètre de codage par rotation est `k`, le décodage sera un codage par rotation de paramètre `-k` ou encore `26-k`.



12.2 Écriture des programmes correspondants

12.2.1 Passage d'une lettre à un entier entre 0 et 25

À chaque lettre on peut faire correspondre son code ASCII.
Avec `Xcas`, `asc("A")=[65]` et `asc("BON")=[66,79,78]`.
Donc pour avoir un entier entre 0 et 25 il suffit de retrancher 65 :
`asc("A")-65` (=0) ou `asc("BON")-[65,65,65]` (=`[1,14,13]`).
On écrit donc la procédure `c2n` qui transforme une chaîne de caractères `m` en une liste d'entiers `l=c2n(m)` entre 0 et 25 (le 2 de `c2n` veut dire "to" ou "vers" en français).

Il faut créer une liste formée des nombres `65` et de même longueur que le message avec `makelist(65,1,size(m))`.
On écrit :

```
c2n(m):={
return(asc(m)-makelist(65,1,size(m)));
}
```

Exemple :
`c2n("BONJOUR")=[1,14,13,9,14,20,17]`

12.2.2 Passage d'un entier entre 0 et 25 à une lettre

À chaque entier n compris entre 0 et 25, on fait correspondre la $(n-1)^{ieme}$ lettre en majuscule de l'alphabet (à 0 correspond "A", 'a 1 correspond "B" etc...).
Avec `Xcas`, `char(65)="A"` et `char([66,79,78])="BON"`.
On écrit donc la procédure `n2c` qui transforme une liste d'entiers `l` entre 0 et 25 en une chaîne de caractères `m`.
Il faut penser à ajouter `65` à tous les éléments de la liste `l` (on forme une liste formée de `65` avec la fonction `makelist` : `l+makelist(65,1,size(l))`).
On écrit :

```
n2c(l):={
return(char(l+makelist(65,1,size(l))));
}
```

Exemple :
`n2c([1,14,13,9,14,20,17])="BONJOUR"`

12.2.3 Passage d'un entier k entre 0 et 25 à l'entier n+k mod 26

On écrit donc la procédure `decal` de paramètres `n` et `l` qui transforme une liste `l` d'entiers `k` entre 0 et 25 en la liste d'entiers $\texttt{n}+\texttt{k} \bmod 26$.
On écrit :

```
decal(n,l):={
return(irem(l+makelist(n,1,size(l)),26));
}
```

Exemple :
`decal(13,[1,14,13,9,14,20,17])=[14,1,0,22,1,7,4]`

12.2.4 Codage d'un message selon Jules César

On écrit la procédure finale `CESAR` qui a deux paramètres :
l'entier `n` de décalage et le message `m`.
On écrit :

```
cesar(n,m):={
return(n2c(decal(n,c2n(m))));
}
```

Exemple :
`cesar(13,"BONJOUR")="OBAWBHE"`

12.3 Codage en utilisant une symétrie par rapport à un axe

FIGURE 12.1 – Codage par symétrie par rapport à Ox

12.3.1 Passage d'un entier k entre 0 et 25 à l'entier n-k mod 26

On reprend les procédures `c2n` et `n2c` vues précédemment :
`c2n` transforme une chaîne de caractères `m` en une liste d'entiers entre 0 et 25 et `n2c` transforme une liste `l` d'entiers entre 0 et 25 en une chaîne de caractères `m`.
On écrit ensuite la procédure `sym` de paramètres `n` et `l` qui transforme une liste `l` d'entiers `k` entre 0 et 25 en la liste d'entiers $\texttt{n} - \texttt{k} \bmod 26$. On peut considérer que le paramètre `n` détermine le diamètre `D` perpendiculaire à la corde `[0,n]` (joignant `A` à la `(n-1)ième` lettre).
La procédure `sym` de paramètre `n` est donc une symétrie par rapport à la droite `D`.
On écrit :

```
sym(n,l):={
return(irem(makelist(n,1,size(l))-l,26));
}
```

12.3.2 Codage d'un message selon une symétrie droite D

On écrit la procédure finale `cesarsym` qui a deux paramètres :
l'entier `n` (définissant la corde `[0,n]` normale au diamètre `D`) et le message `m`.
On écrit :

```
cesarsym(n,m):={
return(n2c(sym(n,c2n(m))));
}
```

Exemple :
Si on prend `n=13` on réalise une symétrie par rapport à la droite `Ox` (cf figure 12.1).
`cesarsym(13,"BONJOUR")="MZAEZTW"`

12.4 Codage en utilisant une application affine

Comme précédemment, on écrit la procédure `c2n` qui transforme une chaîne de caractères `m` en une liste d'entiers entre 0 et 25 et on écrit la procédure `n2c` qui transforme une liste `l` d'entiers entre 0 et 25 en une chaîne de caractères `m`.
On écrit ensuite la procédure `affine` de paramètre `a,b,l` qui transforme une liste `l` d'entiers `k` entre 0 et 25 en la liste d'entiers $\texttt{a} * \texttt{k} + \texttt{b} \bmod 26$.
On écrit :

```
affine(a,b,l):={
return(irem((a*l+makelist(b,1,size(l))),26));
}
```

On écrit ensuite :

```
cesaraffine(a,b,m):={
return(n2c(affine(a,b,c2n(m))));
}
```

Question :
Pour quelles valeurs de a et b le codage obtenu par `cesaraffine` peut-il être décodé ?

12.5 Codage en utilisant un groupement de deux lettres

On écrit la procédure `c2n2` qui transforme une chaîne de caractères `m` en une liste `l` d'entiers entre 0 et $26^2 - 1 = 675$:
On fait des groupements de deux lettres (quitte à terminer le message par la lettre "F" pour avoir un nombre pair de lettres), chaque groupement est considéré comme l'écriture en base 26 d'un entier en utilisant comme "chiffre" les lettres majuscules. Ainsi, `"BC"` est l'écriture en base 26 de 28 (28=1*26+2).
On écrit :

```
c2n2(m):={
local s,lr,l,n;
s:=size(m);
if (irem(s,2)==1){
m:=append(m,"F");
s:=s+1;
}
lr:=[];
l:=asc(m);
for (k:=0;k<s;k:=k+2){
n:=l[k]*26+l[k+1];
lr:=append(lr,n);
}
return(lr);
}
```

On écrit ensuite la procédure `n2c2` qui transforme une liste d'entiers entre 0 et 675 (675=25*26+25=26*26-1) en une chaîne de caractères `m` :
chaque entier étant écrit en base 26 avec comme "symboles" les 26 lettres majuscules.
On écrit :

```
n2c2(l):={
local s,n,m;
s:=size(l);
m:="";
for (k:=0;k<s;k++){
n:=l[k];
m:=append(m,char(iquo(n,26)+65));
m:=append(m,char(irem(n,26)+65));
```

```
}
return(m);
}
```

On écrit ensuite la fonction `affin2` de paramètre `a,b,l` qui transforme une liste `l` d'entiers `k` entre 0 et 675 en la liste d'entiers $\texttt{a} * \texttt{k} + \texttt{b} \bmod 676$ (entiers encore compris entre 0 et 675).
On écrit :

```
affin2(a,b,l):={
local s;
s:=size(l);
for (k:=0;k<s;k++){
l[k]:=irem(a*l[k]+b,676);
}
return(l);
}
```

On écrit ensuite la fonction `cesar2` qui réalise le codage par groupement de 2 lettres utilisant l'application affine `affin2` :

```
cesar2(a,b,m):={
return(n2c2(affin2(a,b,c2n2(m))));
}
```

Question :
Pour quelles valeurs $a1$ de a et $b1$ de b, le codage obtenu par `cesaraffine` peut-il être décodé ?
Réponse :
On doit avoir $a1 * (a * n + b) + b1 = a1 * a * n + a1 * b + b1 = n$.
Il suffit donc de prendre : $b1 = -a1 * b \bmod 676$ et $a1 * a = 1 \bmod 676$

12.6 Le codage Jules César et le codage linéaire

12.6.1 Les caractères et leurs codes

Ici, on ne se borne plus aux 26 lettres de l'alphabet, mais on veut pouvoir utiliser les 101 caractères de la table ci-dessous.
Dans cette table, on a le code du caractère, puis, le caractère : ainsi "5" a pour code 21 et "R" a pour code 50 et l'espace " " a pour code 0.

0									
1 !	2 "	3 #	4 $	5 %	6 &	7 ’	8 (	9)	10 *
11 +	12 ,	13 -	14 .	15 /	16 0	17 1	18 2	19 3	20 4
21 5	22 6	23 7	24 8	25 9	26 :	27 ;	28 <	29 =	30 >
31 ?	32 @	33 A	34 B	35 C	36 D	37 E	38 F	39 G	40 H
41 I	42 J	43 K	44 L	45 M	46 N	47 O	48 P	49 Q	50 R
51 S	52 T	53 U	54 V	55 W	56 X	57 Y	58 Z	59 [	60 \
61]	62 ^	63 _	64 ‘	65 a	66 b	67 c	68 d	69 e	70 f
71 g	72 h	73 i	74 j	75 k	76 l	77 m	78 n	79 o	80 p
81 q	82 r	83 s	84 t	85 u	86 v	87 w	88 x	89 y	90 z
91 {	92 |	93 }	94 ~	95 ê	96 ù	97 ç	98 à	99 è	100 é

12.6.2 Les différentes étapes du codage

Voici les différentes étapes du codage (elles seront programmées avec `Xcas` dans le paragraphe suivant) :

1. À chaque lettre ou symbole, on associe un nombre, comme dans la table 12.6.1. Un texte devient ainsi une suite de nombres : par exemple ê sera codée par 95 et `BONJOUR` par 34 47 46 42 47 53 50.
   La fonction `codec2n` réalise cette étape : elle transforme un caractère en un nombre n ($0 \leq n < 101$), selon la table 12.6.1.
2. On effectue une ou plusieurs opérations sur ces nombres :
   — pour le codage de Jules César, on ajoute à ces nombres un nombre fixé appelé clef de chiffrement (par exemple 17), puis on prend le reste de la division par 101 des nombres obtenus. Ainsi avec ce codage, ê est transformé 95 qui est transformé en 11 (95+17=112 =101+11, ou encore, 11 est le reste de la division de 112 par 101).
   Avec `Xcas`, on effectue la transformation de `n` avec la commande `irem(n+clef,101)`, où `clef` est une variable qui contient la clef de chiffrement.
   — pour le codage linéaire on multiplie ces nombres par un nombre fixé appelé clef de chiffrement (par exemple 17) puis on prend le reste de la division par 101 des nombres obtenus. Ainsi avec ce codage ê est transformé 95 qui est transformé en 100 (95*17=1615 =15*101+100, ou encore, 100 est le reste de la division de 1615 par 101).
   Avec `Xcas`, on effectue la transformation de `n` avec la commande :
   `irem(n*clef,101)`
   si `clef` est une variable qui contient la clef de chiffrement.
3. On transforme ensuite cette suite de nombre en une suite de caractères, c’est le message crypté. Ainsi, avec le codage de Jules César de clef 17, ê devient + et `BONJOUR` devient `S'_|'fc` et avec le codage linéaire de clef 17, ê devient é et `BONJOUR` devient `i|k'|}J`.
   La fonction `coden2c` réalise cette étape : elle transforme un nombre n ($0 \leq n < 101$) en un caractère c, selon la table 12.6.1.
4. Le décryptage nécessite d’inverser les opérations 3, 2 et 1. Dans les exemples :
   — avec le codage de Jules César, il faut enlever 17 ou rajouter 84 et prendre le reste de la division par 101, (84 est la clef de déchiffrement car 84+17=101). Ainsi, 11 est décripté par 95 puisque 11+84=95.

— avec le codage linéaire, il faut multiplier par 6 et prendre le reste de la division par 101, (6 est la clef de déchiffrement car 6*17=102=101+1). Ainsi, 100 est décripté par 95 puisque 100*6=5*101+95.

Le calcul de la clef de déchiffrement à partir de la clef de chiffrement fait intervenir l'arithmétique des entiers :
- savoir trouver l'opposé u d'un élément n de $Z/101Z$ pour le codage de Jules César ($u = 101 - n$ car $u + n = 101$).
- savoir utiliser l'identité de Bézout pour trouver l'inverse u d'un élément de n de $Z/101Z$ pour le codage linéaire (on a $u =$ `iegcd(n,101)[0]` car $u * n + v * 101 = 1$).

12.6.3 Le programme Xcas

```
codec2n(c):={
if (c=="\'e") return 100;
if (c=="\`e") return 99;
if (c=="\`a") return 98;
if (c=="\c{c}") return 97;
if (c=="\`u") return 96;
if (c=="\^e") return 95;
return(asc(c)-32);
};

coden2c(k):={
if (k== 100) return "\'e";
if (k==99) return "\`e";
if (k==98) return "\`a";
if (k==97) return "\c{c}";
if (k==96) return "\`u";
if (k==95) return "\^e";
return(char(k+32));
};

jules_cesar(message,clef):={
local s,j,messcode;
s:=size(message);
messcode:="";
for (j:=0;j<s;j++) {
messcode:=append(messcode,
                 coden2c(irem(clef+codec2n(message[j]),101)));
}
return (messcode);
};

lineaire(message,clef):={
local s,j,messcode;
s:=size(message);
messcode:="";
```

```
for (j:=0;j<s;j++) {
messcode:=messcode+coden2c(irem(clef*codec2n(message[j]),101));
}
return (messcode);
};
```

`codec2n` transforme un caractère c "autorisé" en un entier n, $0 \leq n < 101$ selon la table 12.6.1.
`coden2c` transforme un entier n, $0 \leq n < 101$ en un caractère c selon la table 12.6.1.
`jules_cesar` (respectivement `lineaire`) code le message selon la clé choisie.
On remarquera que les deux fonctions ne diffèrent que par l'opération effectuée :
`clef+codec2n(message[j])` pour la fonction `jules_cesar` et
`clef+codec2n(message[j])` pour la fonction `lineaire`.
Pour décoder, il suffit d'employer le même programme en utilisant la clef de décodage associée à la clef de codage et à la méthode utilisée : par exemple, pour `jules_cesar` de clef de codage 99 la clé de décodage associée est 2 ($2 + 99 = 101 = 0 \bmod 101$) et
pour `lineaire` de clef de codage 99 la clef de décodage associée est 50 puisque $99 * 50 = -2 * 50 = -100 = 1 \bmod 101$.

12.6.4 Le programme C++

```
//#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

//using namespace std;

int codec2n(char c){
  int i=c;
  switch (c){
  case  '\'e':
    i=100;
    break;
  case '\`e':
    i=99;
    break;
  case '\`a':
    i=98;
    break;
  case '\c{c}':
    i=97;
    break;
  case '\`u':
    i=96;
    break;
  case '\^e':
```

```
    i=95;
    break;
  default:
    i -= 32;
  }
  return i;
}

char coden2c(int i){
  if (i<95)
    return i+32;
  switch (i){
  case 95:
    return '\^e';
  case 96:
    return '\`u';
  case 97:
    return '\c{c}';
  case 98:
    return '\`a';
  case 99:
    return '\`e';
  case 100:
    return '\'e';
  }
}

int main(int argc,char ** argv){
  char * s=0,ch;
  size_t n=0;
  int i,d,fois;
  if (!strcmp(argv[0],"./table")){
    for (i=0;i<101;++i){
      ch=coden2c(i);
      printf("%d:%c &",i,ch);
      if (i%10==0)
printf("\n");
    }
    return 0;
  }
  if (!strcmp(argv[0],"./coden2c")){
    for (i=1;i<argc;++i){
      d=atoi(argv[i]);
      ch=coden2c(d);
      printf("%c",ch);
    }
    printf("\n");
    return 0;
```

```
  }
  if (argc==3)
    fois=atoi(argv[2]);
  else
    fois=0;
  if (argc>=2){
    s=argv[1];
    n=strlen(s);
  }
  else {
    printf("Entrez un message \'a num\'eriser\n");
    getline(&s,&n,stdin);
    n=strlen(s)-1;
  }
  for (i=0;i<n;++i){
    d=codec2n(s[i]);
    if (fois)
      printf("%c",coden2c(d*fois % 101));
    else
      printf("%d ",d);
  }
  printf("\n");
  return 0;
}
```

12.6.5 Exercices de décodage

Avec le codage Jules César

message 1, codage Jules César clef 10

```
\'uy\^e}*kvvo\'e*qkqxo|*\^ex*MN
```

message 2, codage Jules César clef 10

```
vk*}yv\^e~syx*x1o}~*zk}*)\'usnox~o
```

message 3, codage Jules César clef 23

```
(\'u(|7\'e|%7\'e!~\'uz\'u|\'e%7\'e\'uy$|%
```

message 4, codage Jules César clef 23

```
$\'u| 7 |7%|$&7{|7z!'$\'u$
```

message 5, codage Jules César clef 35

```
\'e..#*-,1C3,C!&\'e2C3,C!&\'e2
```

message 6, codage Jules César clef 35

```
*éC0B.-,1#C0#*A4#C"3C"B$'
```

message 7, codage Jules César clef 99

```
f_`gjcr\`aq\`ea_jasj_rmgpcq\`em`jge_rmgpcq
```

message 8, codage Jules César clef 99

```
gj\`ed_gr\`e`c_s\`ecr\`eaf_sb
```

message 9, codage Jules César clef 51

```
:ZCB7:7A/B7=<S23S13S1=2/53S3ABS27A1CB/0:3
```

message 10, codage Jules César clef 51

```
<=CASD=C:=<ASRD7B3@S:/S1=<4CA7=<
```

message 11, codage Jules César clef 45

```
*-)=+7=8M,-M<:)>)14M87=:M:1-6
```

message 12, codage Jules César clef 45

```
;+1-6+-M;)6;M+76;+1-6+-M6T-;<M9=-M:=16-M,-M4T)5-
```

Avec le codage linéaire

message 1, codage linéaire clef 10

```
TsJ6 LUUt| #L#it, Ji OY
```

message 2, codage linéaire clef 10

```
UL 6sUJ@7si ift6@ }L6 {T7jti@t
```

message 3, codage linéaire clef 23

```
[_[h ?h{ ?\'e1_:_h?{ ?_#dh{
```

message 4, codage linéaire clef 23

```
d_hm mh {hd- Qh :\'eDd_d
```

message 5, codage linéaire clef 35

```
Uii|BF#m N# 6\`uU+ N# 6\`uU+
```

message 6, codage linéaire clef 35

```
 BU JbiF#m| J|B?q| YN Yb:>
```

message 7, codage linéaire clef 99

```
ZhfXR`B"D dhRd@RhBLXF`D LfRX\hBLXF`D
```

message 8, codage linéaire clef 99

```
XR ^hXB f`h@ `B dZh@b
```

message 9, codage linéaire clef 51

```
FV}JwFw|sJwzG Bu tu tzBsvu u|J Bw|t}JsAFu
```

message 10, codage linéaire clef 51

```
Gz}| Kz}FzG| RKwJuI Fs tzGC}|wzG
```

message 11, codage linéaire clef 45

```
Ik\c{c}xv4xa >k KV\c{c}@\c{c}Uw a4xV VUkl
```

message 12, codage linéaire clef 45

```
\`evUklvk \`e\c{c}l\`e v4l\`evUklvk l,k\`eK )xk VxUlk
```

12.6.6 Solutions des exercices de décodage Jules César et linéaire

1 : vous allez gagner un CD
2 : la solution n'est pas évidente
3 : vive les logiciels libres
4 : rien ne sert de courir
5 : appelons un chat un chat
6 : la réponse relève du défi
7 : habiletés calculatoires obligatoires
8 : il fait beau et chaud
9 : l'utilisation de ce codage est discutable
10 : nous voulons éviter la confusion
11 : beaucoup de travail pour rien
12 : science sans conscience n'est que ruine de l'ame

12.7 Chiffrement affine : premier algorithme

12.7.1 L'algorithme

Pour écrire le message on ne se restreint plus aux 26 lettres de l'alphabet. On suppose que le message à coder n'utilise que les caractères dont le code ASCII va de 32 à 127 (avant 32, les caractères ne sont pas imprimables, et après 127 il s'agit de caractères spéciaux...).
On choisit dans ce premier algorithme de coder chaque lettre : cela à l'incovénient de décrypter facilement le message en analysant la fréquence des lettres du message crypté, c'est pourquoi dans le deuxième algorithme, on code des groupements de trois lettres.
Il y a alors trois choses à faire qui sont les trois instructions de la fonction `cod1` et qui code un caractère par un autre caractère :
- on transforme chaque caractère en un entier n de 0 à 95 (en enlevant 32 à son code ASCII)
- puis, on applique à cet entier n le chiffrement affine :
$f(n) = a \times n + b \pmod{96}$ avec $f(n) \in [0..95]$.
Pour que cette application soit bijective il faut et il suffit que a soit premier avec 96. En effet d'après l'identité de Bézout il existe u et v tels que :
$a \times u + 96 \times v = 1$ donc $a \times u = 1 \pmod{96}$).
On a donc $f^{-1}(m) = u.(m - b) \pmod{96}$
- on transforme le nombre trouvé $f(n)$ en un caractère de code $f(n) + 32$.
Pour coder le message, il suffit ensuite de coder chaque caractère, c'est ce que fait la fonction `codm1`.

Pour décoder, il suffit de remplacer la valeur de a par $a1 = u \pmod{96}$ si $a \times u + 96 \times v = 1$
et la valeur de b par $b1 = -a1 \times b \pmod{96}$
car alors on a $n = a1 \times f(n) + b1 \pmod{96}$
Les fonctions de décodage et de codage sont donc les mêmes , seuls les paramètres sont différents !

Exemple
$a = 85$ $b = 2$
On a par l'identité de Bézout :
$85 \times 61 - 96 \times 54 = 1$ et $-2 \times 61 = -122 = 70 \pmod{96}$
donc on obtient :
$a1 = 61$ $b1 = 70$

12.7.2 Traduction Algorithmique

On note `char` la fonction qui à un nombre `n` associe le caractère de code ASCII `n` et `asc` la fonction qui à un caractère associe son code ASCII.
Voici le codage d'une lettre `c` par la fonction cod1 (`a,b` sont les paramètres du chiffrement affine) :

```
fonction cod1(c,a,b)
local n
asc(c)-32 => n
a.n+b mod 96 => n
résultat char(n+32)
ffonction
```

On suppose que l'on a accés au `k`-ième caractère du mot `m` en mettant `m[k]`.
On suppose que la concaténation de deux mots se fait avec `concat`.
Voici le codage du message `m` par la fonction coda1 (`a,b` sont les paramètres du chiffrement affine) :

```
fonction codm1(m,a,b)
local r,k,n
"" =>r
k=>0 longueur_mot(m)=>s tantque k<s
m[k]=>c
k+1=>k
concat(r,cod1(c,a,b))=>r
ftantque
retourne r
ffonction
```

12.7.3 Traduction Xcas

On dispose de :
- `char` la fonction qui à un nombre `n` associe le caractère de code ASCII `n` et, qui a une liste de nombres associe la chaîne des caractères dont les codes correspondent aux nombres de la liste.
- `asc` la fonction qui à une chaîne de caractères associe la liste des codes ASCII des caractères composant la chaîne.

Attention `asc("A")=[65]` et donc `(asc("A"))[0]=65`.

Voici le codage d'une lettre par la fonction cod1 :

```
cod1(c,a,b):={
local n;
```

```
n:=(asc(c))[0]-32;
n:=irem(a*n+b,96);
return(char(n+32));
}
```

Voici le codage du message par la fonction codm1 :

```
codm1(m,a,b):={
local r,c,s;
r:="";
s:=size(m);
for (k:=0;k<s;k++){
c:=m[k];
r:=concat(r,cod1(c,a,b));
}
return(r);
}
```

On peut aussi coder directement le message `mess` : en effet avec `Xcas` les fonctions `asc` et `char` gérent les chaînes et les listes.
On transforme donc le message (i.e une chaîne de caractères) en une liste `l` de nombres avec `asc(mess)`, puis on transforme cette liste de nombres par l'application affine $f(n) = a \times n + b \pmod{96}$ en la liste `mc`, puis on transformer la liste `lc` des nombres ainsi obtenus en une chaîne de caractères (avec `char(lc)`, c'est ce que fait la fonction `codm`).
Dans ce qui suit on a écrit les fonctions :
`bonpara(para)` (avec `para=[a,b]`) renvoie la liste des paramètres de décodage si le paramètre `a` est premier avec 96. Cette fonction utilise la fonction `decopara(para)` qui calcule les paramètres de décodage.

```
decopara(para):={
//decopara permet de trouver les parametres de decodage
local bez,a,b;
a:=para[0];
b:=para[1];
bez:=bezout(a,96);
a:=irem(bez[0],96);
if (a<0) a:=a+96;
b:=irem(-b*a,96);
if (b<0) b:=b+96;
return([a,b]);
};
bonpara(para):={
//teste si a est premier avec 96
if (pgcd(para[0],96)==1) return(decopara(para)); else return(false);
};
codm(mess,para):={
//codage par chiffrement affine de parametres para=[a,b] mod 96
//codm code le message mess (="....") avec para=[a,b]
local l,lc,sl,a,b,c;
```

```
a:=para[0];
b:=para[1];
l:=asc(mess);
sl:=size(l);
for (k:=0;k<sl;k++){
//les caracteres codes entre 0 et 31 ne sont pas lisibles
l[k]:=l[k]-32;
}
lc:=[];
for (j:=0;j<sl;j++){
c:=irem(a*l[j]+b,96);
lc:=concat(lc,32+c);
}
return(char(lc));
}
```

12.8 Chiffrement affine : deuxième algorithme

12.8.1 L'algorithme

On peut aussi choisir de coder le message en le découpant par paquets de 3 lettres que l'on appelle mot et en rajoutant, éventuellement, des espaces à la fin du message pour que le nombre de caractères du message soit un multiple de 3.
Comme précédemment, à chaque caractère on fait correspondre un nombre entier de l'intervalle $[0..95]$.
On considére alors un mot de 3 lettres comme l'écriture dans la base 96 d'un nombre entier n :
Exemple
le mot `BAL` est la représentation de $n = 34 \times 96^2 + 33 \times 96 + 44 = 316556$.
En effet `B` est codé par 34 puisque son code ASCII est 66 (66-32=34), `A` est codé par 33 et `L` est codé par 44.
On code les mots de 3 lettres par un autre mot, puis on en déduit le codage du message tout entier. Le programme `mot2n` transforme un mot m de 3 lettres en un nombre entier n dont l'écriture en base 96 est m. On a donc, si m a 3 lettres $n < 96^3$.
Par exemple `mot2n("BAL")=316559`.

Le progamme `codaff` transforme n selon le chiffrement affine :
$f(n) = a \times n + b \mod p$.
- il faut choisir $p \geq 96^3$; si $p > 96^3$, le nombre $(f(n))$ obtenu aprés transformation affine de n, peut avoir une représentation de plus de 3 lettres dans la base 96. Mais, au décodage tous les mots auront exactement 3 lettres. Pour les calculateurs qui limitent la représentation d'un entier à 12 chiffres il faut choisir $p \leq 10^6$ pour que $a \times n + b < 10^{12}$
- pour que f soit inversible il faut que a et p soient premiers entre eux (cf p 235) .

Exemple
$a = 567$ $b = 2$ $p = 10^6$
On obtient par Bézout :
$567 \times 664903 + 10^6 \times 377 = 1$

et $-2 \times 664903 = 670164 \mod 10^6$
donc $a1 = 664903$ et $b1 = 670164$

Le programme `n2mot` fait l'opération inverse et transforme un nombre entier n en un mot m (d'au moins 3 symboles) qui est la représentation de n dans la base 96.
Il faut faire attention aux espaces en début de mot ! ! ! En effet, l'espace est codé par `0` et il risque de disparaître si on ne fait pas attention, au décodage ! ! !
Exemple
On a `n2mot(34)` = " B" (c'est à dire la chaîne formée par 2 espaces et B).
Le programme `codmot3` code les mots d'au moins 3 lettres en un mot d'au moins 3 lettres à l'aide du chiffrement affine. En changeant les paramètres a et b `codmot3` décode les mots codés avec `codmot3`.

Le programme `codmess3` code les messages à l'aide du chiffrement affine. Pour décoder, il suffit d'utiliser la fonction `codmess3` en changeant les paramètres a et b.

12.8.2 Traduction Algorithmique

Voici la transformation d'un mot `m` en un nombre entier `n` par la fonction `mot2n` :

```
fonction mot2n(mo)
local k,p,n
0=>n
0=>k
tantque k<longueur_mot(mo) ≠ ""
asc(mo[k])-32=>p
k+1=>k
n*96+p=>n
ftantque
retourne n
ffonction
```

Voici la transformation d'un nombre entier `n` en son écriture en base 96 (c'est à dire en un mot `m` d'au moins 3 lettres) par la fonction `n2mot` :

```
fonction n2mot(n)
local m,r,j
""=>m
0=>j
tantque n >0 ou j< 3
char((n mod 96)+32)=>r
int(n/96)=>n
r+m=>m
j+1=>j
ftantque
retourne m
ffonction
```

Voici le codage d'un mot d'au moins 3 lettres par la fonction `codmot3` :

```
fonction codmot3(m,a,b,p)
local n
```

```
mot2n(m)=>n
a.n+b mod p =>n
n2mot(n)=>m
retourne m
ffonction
```

Voici le codage d'un message par la fonction `codmess` :

```
fonction codmess3(m,a,b,p)
local n,i,r,d
int(dim(m)/3)+1=>n
{}=>r
1=>i
tantque i<n
debut(m,3)=>d
fin(m,4)=>m
codmot3(d,a,b,p)=>r[i]
i+1=>i
ftantque
si dim(m)=2 alors
m + " "=>m
codmot3(m,a,b,p)=>r[i]
sinon
si dim(m)=1 alors
m + " " =>m
codmot3(m,a,b,p)=>r[i]
sinon
fin(r,i-1)=>r
fsi
fsi
retourne r
ffonction
```

12.8.3 Traduction Xcas

Voici la fonction `decopara3(para)` qui donne les paramètres de décodage quand les paramètres sont corrects.
On prend comme chiffrement affine $a * n + b \mod 96^3$ car on veut mettre le message codé dans une chaîne et donc transformer un paquet de 3 lettres en un paquet d'exactement 3 lettres.

```
decopara3(para):={
//=le parametrage de decodage du parametrage para (liste).
local a,b,l;
a:=para[0];
b:=para[1];
l:=bezout(a,96^3);
if (l[2]!=1) return(false);
a:=l[0];
if (a<0) a:=a+96^3;
b:=-irem(b*a,96^3)+96^3;
```

```
return([a,b]);
}
```

Voici la transformation d'un mot s d'au moins 3 lettres en un nombre entier par la fonction `mot2n` :

```
mot2n(s):={
//transforme un mot s de 3 lettres en n
//n a pour ecriture s en base 96
local l,n;
l:=asc(s);
n:=(l[0]-32)*96^2+(l[1]-32)*96+l[2]-32;
return(n);
}
```

Voici la transformation d'un nombre entier n en son écriture en base 96 (c'est à dire en un mot d'au moins 3 lettres) par la fonction `n2mot` : cette fonction utilise la fonction `ecritu96` qui écrit `n` dans la base 96 comme un mot de 1,2,3 etc caractères. Pour obtenir un mot d'au moins 3 lettres il suffit de rajouter des espaces devant le mot puisque le code ASCII de l'espace vaut 32, cela revient à rajouter des zéros devant l'écriture de `n`.

```
ecritu96(n):={
//transforme l'entier n en la chaine s
//s est l'ecriture de n en base 96
local s,r;
// n est un entier et b=96
// ecritu96 est une fonction iterative
//ecritu96(n)=l'ecriture de n en base 96
s:="";
while (n>=96){
r:=irem(n,96);
r:=char(r+32);
s:=r+s;
n:=iquo(n,96);
}
n:=char(n+32);
s:=n+s;
return(s);
};

n2mot(n):={
local mot,s;
mot:=ecritu96(n);
s:=size(mot);
//on suppose n<96^3 on transforme n en un mot de 3 caracteres
//on rajoute des espaces si le mot n'a pas 3 lettres
if (s==2) {mot:=" "+mot;}
else {
if (s==1) {mot:="  "+mot;}
```

```
}
return(mot);
}
```

Voici le codage d'un mot d'au moins 3 lettres par la fonction codmot3 : en prenant toujours $p = 96^3$

```
codmot3(mot,para):={
//codage d'un mot de 3 lettres avec le parametrage para=[a,b]
local n,m,a,b;
//para:[569,2] mod 96^3
//decopara3=[674825, 419822]
a:=para[0];
b:=para[1];
n:=mot2n(mot);
m:=irem(a*n+b,96^3);
return(n2mot(m));
}
```

Le décodage d'un mot codé avec `codmot3` se fait aussi avec la fonction `codmot3`.
Voici le codage d'un message par la fonction `codmess3` :

```
codmess3(mess,para):={
//code le message mess,parametrage para et paquet de 3 lettres
  local s,messcod,mess3;
  s:=size(mess);
  if (irem(s,3)==2){
    mess:=mess+" ";
    s:=s+1;
  }
  else {
    if (irem(s,3)==1) {
       mess:=mess+"  ";
       s:=s+2;
    }
  }
  messcod:="";
  for (k:=0;k<s;k:=k+3){
    mess3:=mess[k..k+2];
    mess3:=codmot3(mess3,para);
    messcod:=messcod+mess3;
  }
return(messcod);
}
```

Le décodage du message se fait aussi par la fonction `codmess3`

12.9 Devoir à la maison

On écrit un message plus ou moins long selon le nombre d'élèves de la classe.
Puis on le partage en groupement de 8 lettres.

Chaque groupe de 8 lettres est ensuite codé (lettre par lettre) à l'aide d'un chiffrement affine de paramètres diffèrents selon les élèves.

Le chiffrement affine lettre à lettre est déterminé par la donnée de 3 paramètres : a, b, p qui transforme l'entier n en $m = a * n + b \pmod{p}$.
Pour avoir une fonction de décodage il faut et il suffit que a soit inversible dans Z/pZ c'est à dire que a et p soient premiers entre eux.
La fonction de décodage est alors :
$a_1 * m + b_1$ avec,
a_1 inverse de a dans Z/pZ ($a_1 = u \pmod{p}$ si $a * u + p * v = 1$ (identité de Bézout)) et $b_1 = -b * a_1 \pmod{p}$.

Pour ce chiffrement affine, on peut utiliser tous les caractères dont les codes ASCII vont de 32 à 127 (les caractères de code 0 à 31 ne sont pas imprimables et au delà de 127 ils ne sont pas standards).

Étant donnés $a,\ b,\ p = 96$, comment coder ?
À chaque caractère (de code ASCII compris entre 32 et 127) on fait correspondre un entier n entre 0 et 95 ; n est égal à : (code ASCII du caractère) - 32.
Par exemple, à `B` on fait correspondre 34 (66-32).
Puis, on calcule $m = a * n + b \mod 96$:
pour $a = 55$, $b = 79$ et $n = 34$ on obtient $m = 29$.
Puis, on cherche le caractère de code ASCII $m+32$. Le caractère qui a comme code ASCII 29+32=61 est le signe =) : c'est ce caractère qui sera donc choisi comme codage de la lettre `B`.
Exemple :
On va coder la phrase :
`BEAUCOUP DE TRAVAIL POUR RIEN!?`
en la coupant en trois morceaux (le caractère espace termine les deux premiers morceaux).
Par exemple :
`BEAUCOUP` sera codé avec $a = 55$ $b = 79$ et $p = 96$
`DE TRAVAIL` sera codé avec $a = 49$ $b = 25$ et $p = 96$
`POUR RIEN!?` sera codé avec $a = 73$ $b = 48$ et $p = 96$
On code `BEAUCOUP` avec $a = 55$ $b = 79$ et $p = 96$.
On obtient :
`="f2th2?o`
Le devoir du premier élève est donc :
avec les paramètres de codage $a = 55$ $b = 79$ et $p = 96$, décoder `="f2th2?o`
L'élève doit :
- trouver les paramètres de décodage (ici $a_1 = 7$ $b_1 = 23$) :
en effet l'inverse a_1 de $a \pmod{96}$ est obtenu en ecrivant l'identité de Bézout pour a et p :
$55 * 7 - 96 * 4 = 1$ et $b_1 = 79 * 7 \pmod{96} = 23$
- puis décoder à l'aide d'une table de code ASCII :
le code ASCII du caractère = est 61 donc :
$m = 61 - 32 = 29$
$a_1 * n + b_1 \pmod{96} = 7 * 29 + 23 \pmod{96} = 226 \pmod{96} = 34$
La lettre de code 34+32 = 66 est `B`

12.9.1 Le code Ascii

Voici pour Xcas, la table des codes ASCII compris entre 30 et 127.

	0	1	2	3	4	5	6	7	8	9
30	^^	^_		!	"	#	$	%	&	’
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	^?		
130										

12.10 Codage de Vigenère

12.10.1 Le principe du codage

On veut coder un message `mess` qui n’utilise que les 26 lettres majuscules de l’alphabet.
Dans le codage de César à chaque caractère correspond toujours le même caractère : c’est une fonction qui à une lettre de l’alphabet fait correspondre une lettre de l’alphabet.
Dans le codage de Vigenère à un caractère $cara$ du message correspond un caractère qui dépend de la position du caractère $cara$ dans le message : c’est une fonction f de 2 variables qui à un entier n ($0 \leq n \leq 25$) et et à une lettre de l’alphabet fait correspondre une lettre de l’alphabet. Cette correspondance se fait à l’aide d’une matrice carrée symétrique de dimension 26 appelée : carré de Vigenère.

12.10.2 Le carré de Vigenère

On a `asc("A")=65`. Si le rang de `"A"` dans la liste alphabétique est `0` alors le rang d’une lettre `c` dans la liste alphabétique est : `op(asc(cara))-65`. Il faut utiliser la commande `op` pour transformer la liste de dimension 1 renvoyée par `asc` en un nombre.
Si à une lettre on fait correspondre son rang dans la liste alphabétique, au carré de Vigenère on peut faire correspondre la matrice carrée de dimension 26 :
`A[j,k]:=irem(j+k, 26)` pour `j=0..25` et `k=0..25`.
Réciproquement si à un entier `j` de l’intervalle 0..25] on fait correspondre la lettre de rang `j` dans la liste alphabétique, alors, à la matrice la matrice carrée de dimension 26 `A[j,k]:=irem(j+k, 26)` on fait correspondre le carré de Vigenère.
On tape pour obtenir le carré de Vigenère `CV` :
`CV:=makemat((j,k)->char(65+irem(j+k,26)),26,26)`
On tape pour obtenir la fonction `f` de codage :

```
f(n,cara):=CV[n,op(asc(cara))-65]
```

Propriétés La matrice `A[j,k]:=irem(j+k, 26)` est symétrique donc le carré de Vigenère est symétrique.
Si connaissant `j` et `r`, on cherche `k` pour que `r=A[j,k]`, alors on a `k=A[26-j,r]=A[r,26-j]`.
En effet, `r=irem(j+k,26)` donc `k=irem(r-j,26)=irem(r+26-j,26`.
Donc si `r=A[j,k]` alors `k=A[26-j,r]=A[r,26-j]`.

On peut utiliser le tableau à double entrées ci-dessous pour coder à la main :

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Si à une lettre on fait correspondre son rang dans la liste alphabétique, au carré de Vigenère on peut faire correspondre la matrice :
`A[j,k]:=irem(j+k, 26)`.
Ce carré donne la valeur de la fonction `f` du codage qui est : `f(0,"D")="D"` (ligne 0 et colonne 3) et `f(2,"D")="F"` (ligne 2 et colonne 3) pour coder par exemple les `"D"` de `"DINDON"` (Attention les indices commencent à 0).
Ainsi `"BONJOUR"` sera codé par :
`f("B",0)+f("O",1)+f("N",2)+f("J",3)+f("O",4)+f("U",5)+f("R",6)`
c'est à dire `"BPPMSZX"` On pourra voir le programme qui code, décode avec ou sans clé en 12.10.10.

12.10.3 Le programme du codage lettre par lettre

On utilise CV dans le programme.
On veut coder la lettre c qui est d'indice j dans le message. Cette lettre c est de rang op(asc(c))-65 dans l'alphabet.
Il faut donc chercher dans le tableau CV la lettre d'indice ligne irem(j,26) et d'indice colonne op(asc(c))-65.
Par exemple si on cherche à coder "O" d'indice 4 du message (la 5ième lettre du message à coder).
"O" a comme rang op(asc("O"))-65=14 dans l'alphabet ("O" est la 15ième lettre de l'alphabet).
Le codage est donc la lettre située dans la ligne d'indice irem(4,26)=4 (ligne débutant par "E") et dans la colonne 14 du tableau CV.
"O" a donc pour code CV[4,14]="S" car :

E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D

On tape :

```
f(n,cara):=CV[irem(n,26),op(asc(cara))-65]:;
CV:=makemat((j,k)->char(65+irem(j+k,26)),26,26):;
codage(str):={
local j,k,c,s,code,CV;
s:=size(str);
CV:=makemat((j,k)->char(65+irem(j+k,26)),26,26):;
code:="";
pour j de 0 jusque s-1 faire
c:=str[j];
code:=code+CV[irem(j,26),op(asc(c))-65];
//code:=code+f(j,c);
fpour;
return code;
}:;
```

On peut aussi ne pas utiliser CV :
On veut coder la lettre c qui est d'indice j dans le message. Cette lettre c est de rang op(asc(c))-65 dans l'alphabet.
Dans le tableau CV la lettre qui commence la ligne d'indice irem(j,26) est char(irem(j,26))+65 et a comme code ASCII irem(j,26)+65.
La lettre de cette ligne située à la colonne d'indice op(asc(c))-65 a donc comme rang k:=irem(op(asc(c))-65+irem(j,26),26) dans l'alphabet.
La lettre de codage de c est donc char(k+65).
Par exemple si on cherche à coder "O" d'indice 4 du message (la 5ième lettre du message à coder).
"O" a comme rang op(asc("O"))-65=14 dans l'alphabet ("O" est la 15ième lettre de l'alphabet).
Le codage est donc la lettre située dans la ligne d'indice irem(4,26)=4 (ligne débutant par "E" ayant comme rang 4 dans l'alphabet) et dans la colonne 14 c'est donc la lettre "S" de rang 4+14=18 de l'alphabet (car "S" est la 19ième lettre de l'alphabet).

Attention si la somme des 2 indices se fait modulo 26 car cette somme est l'indice d'une lettre de l'alphabet et cette somme ne doit pas dépasser 25, c'est pourquoi
`k:=irem(op(asc(c))-65+irem(j,26),26).`
`"O"` a donc pour code `"S"`.
On tape :

```
codage(str):={
local j,k,c,s,code;
s:=size(str);
code:="";
pour j de 0 jusque s-1 faire
c:=str[j];
k:=irem(op(asc(c))-65+irem(j,26),26);
code:=code+char(k+65);
fpour;
return code;
}:;
```

On tape :
`codage("BONJOURBONJOURBONJOURBONJOUR")`
On obtient :
`"BPPMSZXIWWTZGEPDDAGNLWKKHNUS"`

12.10.4 Le programme du décodage lettre par lettre

On utilise `CV` dans le programme :
Si la lettre à déchiffrer `c` est d'indice `j` dans le message à décoder, il faut chercher l'indice de la lettre `c` sur la ligne d'indice `irem(j,26)]` et trouver son indice `k` dans cette ligne sans utiliser la fonction `member`.
Propriété de la matrice `CV` :
On remarque que si pour $j \in 0..25$ et $l \in$ "A".."Z" on définit :
`f(j,l):=CV[op(asc(l))-65,irem(j,26)]`
`g(j,l):=CV[irem(26-j,26),op(asc(l))-65]`
alors
`f(j,g(j,l))=l`
Autrement dit si la lettre `c` est d'indice `j` dans le message codé alors la lettre d'indice `j` dans le message non codé est CV[irem(26-j,26),op(asc(c))-65].
Par exemple si on cherche à déchiffrer `"S"` d'indice 4 (la 5ième lettre du message à décoder), il faut chercher l'indice de `"S"` dans la ligne d'indice 4. La lettre cherchèe a donc comme indices dans `CV` :
`[irem(26-4,26), op(asc("S"))-65]=[22,18].`
C'est donc `CV[22,18]="O"`.
On tape :

```
g(n,cara):=CV[irem(26-n,26),op(asc(cara))-65];;
CV:=makemat((j,k)->char(65+irem(j+k,26)),26,26):;
decodage(str):={
local j,k,c,s,mess,CV;
s:=size(str);
```

```
CV:=makemat((j,k)->char(65+irem(j+k,26)),26,26):;
mess:="";
pour j de 0 jusque s-1 faire
c:=str[j];
//mess:=mess+g(j,c);
mess:=mess+CV[irem(26-j,26),op(asc(c))-65]
fpour;
return mess;
}:;
```

On peut aussi ne pas utiliser `CV`.
Si la lettre à déchiffrer `c` est d'indice `j` dans le message à décoder, il faut chercher la lettre `c` sur la ligne d'indice `irem(j,26)` et trouver son indice `k` dans cette ligne sans utiliser la fonction `member`.
Il suffit donc de faire la différence modulo 26, entre le rang de la lettre `c` dans l'alphabet (`op(asc(c))-65`) et l'indice `irem(j,26)` soit :
`irem(op(asc(c))-65-irem(j,26),26)=irem(op(asc(c))-65-j,26)`.
Par exemple si on cherche à déchiffrer `"S"` d'indice 4 dans le message (la 5ième lettre du message à décoder), il faut chercher l'indice de `"S"` dans la ligne d'indice 4 débutant par`"E"` :

E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D

L'indice de `"S"` dans la ligne d'indice 4 débutant par`"E"` est donc :
`irem(op(asc("S"))-65-4,26)=irem(18-4,26)=14`.
La lettre cherchée a donc comme rang `14` dans l'alphabet.
C'est donc `char(14+65)="O"`.
On tape :

```
decodage1(str):={
local j,k,c,s,mess;
s:=size(str);
mess:="";
pour j de 0 jusque s-1 faire
c:=str[j];
k:=irem(op(asc(c))-65-j,26);
mess:=mess+char(k+65);
fpour;
return mess;
}:;
```

On tape :
`decodage("BPPMSZXIWWTZGEPDDAGNLWKKHNUS")`
On obtient :
`"BONJOURBONJOURBONJOURBONJOUR"`

12.10.5 Le codage de Vigenère avec une clé

On voit qu'il est facile de faire le décodage d'un message si on ne donne pas une clé.
La clé est un mot qui va nous servir à coder le message.
Prenons comme exemple une clé simple : "XCAS".

Si le message à coder est "BONJOUR", on écrit "XCAS" plusieurs fois sous le mot à coder :
`"BONJOUR"`
`"XCASXCA"`
puis, on utilise seulement le moceau du carré de Vigenère qui correspond à la clè á savoir que les lignes qui commencent par "X","C","A","S" :

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
2	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
3	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R

Pour coder "BONJOUR", on code "B" avec "Y", (ligne 0 débutant par "X" et colonne "B"), "O" avec "Q" (ligne 1 débutant par "A" et colonne "O"), "N" avec "N", "J" avec "B", "O" avec "L", "U" avec "W" et "R" avec "R".
Le codage de "BONJOUR" est donc "YQNBLWR".

12.10.6 Le programme du tableau de Vigenère avec une clé

Voici le programme qui donne en fonction de la clé utilisée, la matrice extraite de la matrice `CV` i.e. le carré de Vigenère.
On tape :

```
TVAC(cle):={
local sc,VAC,CV;
sc:=size(cle);
CV:=makemat((j,k)->char(65+irem(j+k,26)),26,26);
VAC:=makemat(0,sc,26);
pour j de 0 jusque sc-1 faire
VAC[j]:=CV[op(asc(cle[j])-65)];
fpour;
return VAC;
}:;
```

On tape :
`TVAC("XCAS")`
On obtient :

X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R

12.10.7 Le programme du codage

On peut utiliser la fonction `TVAC` pour programmer le codage de Vigenère avec une clé.
On veut coder la lettre `c` qui est d'indice `j` dans le message. Cette lettre `c` est de rang `op(asc(c))-65` dans l'alphabet.
Il faut donc chercher dans le tableau `VAC` la lettre d'indice ligne `irem(j,sc)` et d'indice colonne `op(asc(c))-65` .

Par exemple si la clé est "XCAS" et que l'on cherche à coder "O" d'indice 4 du message (la 5ième lettre du message à coder).
"O" a comme rang op(asc("O"))-65=14 dans l'alphabet ("O" est la 15ième lettre de l'alphabet).
Le codage est donc la lettre située dans la ligne d'indice irem(4,4)=0 (ligne débutant par "X") et dans la colonne 14 du tableau VAC.
"O" a donc pour code VAC[0,14]="L" car :

X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

On tape :

```
codagec0(str,cle):={
local j,k,c,s,sc,VAC,code;
s:=size(str);
VAC:=TVAC(cle);
sc:=size(cle);
code:="";
pour j de 0 jusque s-1 faire
c:=str[j];
code:=code+VAC[irem(j,sc),op(asc(c))-65];
fpour;
return code;
}:;
```

On tape :
codagec0("BONJOUR","XCAS")
On obtient :
"YQNBLWR"
On tape :
codagec0("BONJOURBONJOUR","XCAS")
On obtient :
"YQNBLWRTLPJGRT"

On peut aussi n'utiliser que le carré de Vigenère CV pour coder.
On veut coder la lettre c qui est d'indice j dans le message. Cette lettre c est de rang op(asc(c))-65 dans l'alphabet.
Il faut donc chercher dans le tableau CV la lettre située dans la ligne débutant par cle[irem(j,sc)] qui est d'indice op(asc(cle[irem(j,sc)])-65) et d'indice colonne op(asc(c))-65.
Par exemple si la clé est "XCAS" et que l'on cherche à coder "O" d'indice 4 du message (la 5ième lettre du message à coder).
"O" a comme rang op(asc("O"))-65=14 dans l'alphabet ("O" est la 15ième lettre de l'alphabet et 14 est l'indice colonne de "O" du tableau CV).
Le codage est donc la lettre située dans le tableau CV à la ligne d'indice 23 (23 est obtenu avec op(asc(cle[irem(4,4)])-65)=op(asc(cle[0]))-65) et à la colonne d'indice 14.
"O" a donc pour code CV[23,14]="L".
On tape :

```
)
```

```
codagec1(str,cle):={
local j,k,c,s,sc,code,CV;
s:=size(str);
sc:=size(cle);
CV:=makemat((j,k)->char(65+irem(j+k,26)),26,26);
code:="";
pour j de 0 jusque s-1 faire
c:=str[j];
code:=code+CV[op(asc(cle[irem(j,sc)]))-65,op(asc(c))-65];
fpour;
return code;
}:;
```

On tape :
`codagec1("BONJOUR","XCAS")`
On obtient :
`"YQNBLWR"`
On tape :
`codagec1("BONJOURBONJOUR","XCAS")`
On obtient :
`"YQNBLWRTLPJGRT"`
On tape :
`codagec1("BONJOURBONJOURBONJOURBONJOUR","XCAS")`
On obtient :
`"YQNBLWRTLPJGRTBGKLOMODOFGQUJ"`

12.10.8 Le programme du décodage

Comme pour le codage on peut utiliser la fonction `TVAC` pour programmer le décodage de Vigenère avec une clé.
Si la lettre à déchiffrer `c` est d'indice `j` dans le message à décoder, il faut chercher l'indice de la lettre `c` sur la ligne débutant par la lettre `cle[irem(j,sc)]=VAC[0,irem(j,sc)]` et trouver son indice `k` dans cette ligne sans utiliser la fonction `member`. Il suffit donc de faire la différence modulo 26, entre le code ASCII de la lettre `c` et celui de la lettre `VAC[0,irem(j,sc)]`.
Par exemple si la clé est `"XCAS"` et que l'on cherche à déchiffrer `"L"` d'indice 4 (la 5ième lettre du message à décoder), il faut chercher l'indice de `"L"` dans la ligne débutant par `VAC[0,0]="X"` (`"X"` est `VAC[0,0]`) :

X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

On a :
`VAC[0,0]=VAC[0,0]="X"` `asc("L")=[76]` et `asc("X")=[88]`
`op(asc("L"))=76` et `op(asc("X"))=88`
`op(asc("L"))-op(asc("X"))=-12`
L'indice de `"L"` dans la ligne est donc `irem(-12,26)=14` et `char(14+65)="O"`. Le décodage de `"L"` d'indice 4 dans le message à décoder est donc `"O"` lorsque la clé est `"XCAS"`.
On tape :

```
decodagec0(str,cle):={
```

```
local j,k,c,s,sc,VAC,mess;
s:=size(str);
sc:=size(cle);
VAC:=TVAC(cle);
mess:="";
pour j de 0 jusque s-1 faire
c:=str[j];
//k:=member(c,VAC[irem(j,sc)])-1;
//k:=irem(op(asc(c))-op(asc(cle[irem(j,sc)])),26);
k:=irem(op(asc(c))-op(asc(VAC[0,irem(j,sc)])),26);
mess:=mess+char(k+65);
fpour;
return mess;
}:;
```

On tape :
`decodagec0("YQNBLWR","XCAS")`
On obtient :
`"BONJOUR"` On tape :
`decodagec0("YQNBLWRTLPJGRT","XCAS")`
On obtient :
`"BONJOURBONJOUR"` On tape :
`decodagec0("CYCPAQGKE","CHAISE")`
On obtient :
`"ARCHIMEDE"`
Comme pour le codage on peut n'utiliser que le carré de Vigenère `CV` pour décoder.
On tape :

```
decodagec1(str,cle):={
local j,k,c,s,sc,VAC,mess,CV;
s:=size(str);
sc:=size(cle);
CV:=makemat((j,k)->char(65+irem(j+k,26)),26,26):;
mess:="";
pour j de 0 jusque s-1 faire
c:=str[j];
//mess:=mess+CV[26-op(asc(cle[irem(j,sc)]))+65,op(asc(c))-65];
k:=irem(op(asc(c))-op(asc(cle[irem(j,sc)])),26);
mess:=mess+char(k+65);
fpour;
return mess;
}:;
```

On tape :
`decodagec("YQNBLWR","XCAS")`
On obtient :
`"BONJOUR"` On tape :
`decodagec("YQNBLWRTLPJGRT","XCAS")`
On obtient :

"BONJOURBONJOUR" On tape :
decodagec("CYCPAQGKE","CHAISE")
On obtient :
"ARCHIMEDE"
Remarque
Lorsqu'on code avec le carré de Vigenère c'est comme si on avait choisit comme clé les 26 lettres de l'alphabet.
Pour passer des programmes de codage et de décodage sans clé avec les programmes de codage et de décodage avec clé qui n'utilise que CV, il suffit de remplacer l'indice j de la lettre à coder par jc:=op(asc(cle[irem(j,sc)]))-65 où sc représente la longueur de la clé.
Pour le codage on remplace :
code:=code+CV[irem(j,26),op(asc(c))-65];
par :
code:=code+CV[op(asc(cle[irem(j,sc)]))-65,op(asc(c))-65];
comme 0<=jc=op(asc(cle[irem(j,sc)]))-65<=25 il est inutile de chercher le reste de sa division par 26.
Pour le décodage on remplace :
mess:=mess+CV[irem(26-j,26),op(asc(c))-65];
par :
mess:=mess+CV[26-op(asc(cle[irem(j,sc)]))+65,op(asc(c))-65];
ici encore le reste de de la division par 26 de 26-jc est inutile !

12.10.9 Peut-on décrypter sans connaitre la clé ?

Dans ce type de codage dit polyalphabétiques, on ne peut pas utiliser l'analyse des fréquences puisqu peusqu'un même caractère peut être codé par des caractères différents.
Mais le britannique Ch Babbage au 19 ième siecle a trouvé la parade.
Si on a choisit une clé de 4 lettres, chaque caractère peut être codé par au plus 4 caractères différents.
Si le texte est suffisamment long, on peut essayer de repérer dans le texte codé les mots qui se repétent et d'en déduire le nombre de lettres de la clé.
Par exemple :
codagec1("BONJOURBONJOURBONJOURBONJOURBONJOUR","XCAS")
renvoie
"YQNBLWRTLPJGRTBGKLOMODOFGQUJYQNBLWR"
On remarque que le premier et le dernier "BONJOUR" sont codés de la même façon et qu'entre 2 "Y" y a 27 lettres, 28 est donc un multiple du nombre de lettres de la clé..... Si on a plusieurs répétitions le nombre de lettres de la clé est un diviseur commun du nombre de lettre +1 qui sépare ces répétitions.
Si le nombre de lettres de la clé est 4 il reste alors à décoder 4 chiffrements monoalphabétiques ....

12.10.10 Exercice : codage et décodage globalement

Codage sans une clé

Au lieu de coder ou de décoder lettre par lettre on va le faire globalement
```
ordres(mess):=asc(mess)-[65$(k=1..size(mess))];transfCV(L):=[irem(j+L
codeA(LT):=[char(LT[j]+65)$(j=0..size(LT)-1)];
mot(L):=ifte(L==[],"",L[0]+mot(tail(L)));
```
Par exemple :
```
L:=ordres(("BONJOURBONJOURBONJOURBONJOURBONJOUR")
```
renvoie
```
[1,14,13,9,14,20,17,1,14,13,9,14,20,17,1,14,13,9,14,20,17,1,14,13,9,1
LT:=transfCV(L)
```
renvoie
```
[1,15,15,12,18,25,23,8,22,22,19,25,6,4,15,3,3,0,6,13,11,22,10,10,7,13,
LA:=codeA(LT)
```
renvoie
```
["B","P","P","M","S","Z","X","I","W","W","T","Z","G","E","P","D","D","
messcode:=mot(LA)
```
renvoie
```
"BPPMSZXIWWTZGEPDDAGNLWKKHNUS"
```

Décodage sans une clé

Il suffit de transformer `transfCV` en `transgCV` en utilisant la fonction g inverse de f :
`transgCV(L):=[irem(26-j+L[j],26)$(j=0..size(L)-1)];` Par exemple :
`Lde:=ordres("BPPMSZXIWWTZGEPDDAGNLWKKHNUS")` renvoie `[1,15,15,12,18,25,23,`
```
LTde:=transgCV(Lde)
```
renvoie
```
[1,14,13,9,14,20,17,1,14,13,9,14,20,17,1,14,13,9,14,20,17,1,14,13,9,1
mot(codeA(LTde))
```
renvoie
```
"BONJOURBONJOURBONJOURBONJOUR"
```

Codage avec une clé

```
   ordres(mess):=asc(mess)-[65$(k=1..size(mess))];transfCVcle(L,cle):=
codeA(LT):=[char(LT[j]+65)$(j=0..size(LT)-1)];
mot(L):=ifte(L==[],"",L[0]+mot(tail(L)));
```
Par exemple :
```
L:=ordres(("BONJOURBONJOURBONJOURBONJOURBONJOUR")
```
renvoie
```
[1,14,13,9,14,20,17,1,14,13,9,14,20,17,1,14,13,9,14,20,17,1,14,13,9,1
LTcle:=transfCVcle(L,"XCAS"
```
renvoie
```
[24,16,13,1,11,22,17,19,11,15,9,6,17,19,1,6,10,11,14,12,14,3,14,5,6,1
LAcle:=codeA(LTcle)
```
renvoie

```
["Y","Q","N","B","L","W","R","T","L","P","J","G","R","T","B","G","K","L","O",
messcodecle:=mot(LAcle)
```
renvoie
```
"YQNBLWRTLPJGRTBGKLOMODOFGQUJ"
```

Décodage avec une clé

Il suffit de transformer `transfCVcle` en `transgCVcle` en utilisant la fonction g inverse de f :
```
transgCVcle(L,cle):=[irem(26-(ordres(cle))[irem(j,size(cle))]+L[j],26)$(j=0..
```
Par exemple :
`Ldecle:=ordres("YQNBLWRTLPJGRTBGKLOMODOFGQUJ")` renvoie `[24,16,13,1,11,22,17,19,1`
```
LTdecle:=transgCVcle(Ldecle,"XCAS")
```
renvoie
```
[1,14,13,9,14,20,17,1,14,13,9,14,20,17,1,14,13,9,14,20,17,1,14,13,9,14,20,17]
mot(codeA(LTde))
```
renvoie
```
"BONJOURBONJOURBONJOURBONJOUR"
```

Le programme qui code et décode avec ou sans clé

Soit un message `mess` qui n'utilise que les 26 lettres majuscules de l'alphabet. Voici le programme définitif qui code (si `n>0`), qui décode (si `n<0`) avec une clé ou sans clé (si `cle=""`). On écrit la procédure récursive `mot` qui à une liste de caractères renvoie le mot formé par ses caractères.

```
mot(L):=ifte(L==[],"",L[0]+mot(tail(L)));
codage(mess,cle,n):={
local s,sc,messc,messcT,messTL;
  si (n==0) alors retourne "erreur n!=0" fsi;
  si (cle=="") alors cle:=char((65+j)$(j=0..25)); fsi;
  s:=size(mess);
  sc:=size(cle);
  messc:=asc(mess)-[65$ s];
  clec:=asc(cle)-[65$ sc];
  messcT:=[irem(sign(n)*clec[irem(j,sc)]+messc[j],26)$(j=0..s-1)];
  messTL:=[char(messcT[j]+65)$(j=0..s-1)];
  retourne mot(messTL);
}:;
```

On tape :
```
codage("BONJOURBONJOURBONJOURBONJOUR","",1)
```
On obtient :
```
"BPPMSZXIWWTZGEPDDAGNLWKKHNUS"
```
On tape :
```
codage("BPPMSZXIWWTZGEPDDAGNLWKKHNUS","",-1)
```
On obtient :
```
"BONJOURBONJOURBONJOURBONJOUR"
```
On tape :

```
codage("BONJOURBONJOURBONJOURBONJOUR","XCAS",1)
```
On obtient :
```
"YQNBLWRTLPJGRTBGKLOMODOFGQUJ"
```
On tape :
```
codage("YQNBLWRTLPJGRTBGKLOMODOFGQUJ","XCAS",-1)
```
On obtient :
```
"BONJOURBONJOURBONJOURBONJOUR"
```

12.11 Codage RSA

12.11.1 Le principe du codage avec clé publique et clé secrète

On suppose que l'on dispose d'un annuaire de clés (ce sont les clés publiques comme un a nnuaire de téléphone) qui permet d'envoyer à quelqu'un (par exemple à Tartanpion) un message que l'on code en se servant de la clé publique de Tartanpion. Mais seul Tartanpion pourra décoder les messages qu'il reçoit : il pourra le faire grâce à sa clé secréte.
Autrement dit tout le monde sait comment il faut coder les messages pour Tartanpion : la fonction f_T de codage pour Tartanpion est connue mais la fonction g_T de décodage n'est connue que de Tartanpion car il existe des fonctions inversibles f dont l'inverse g est difficile à trouver.

12.11.2 Codage avec signature, clé publique et clé secrète

Si Tartanpion reçoit un message provenant de Martin. Comment Tartanpion peut-il être sûr que c'est bien Martin qui lui a envoyé ce message vu que sa clé publique est accessible à tout le monde ?
Martin et Tartanpion peuvent convenir de faire un codage avec signature :
Si la fonction f_T de codage pour Tartanpion est connue mais la fonction g_T de décodage n'est connue que de Tartanpion et si la fonction f_M de codage pour Martin est connue mais la fonction g_M de décodage n'est connue que de Martin alors,
Martin codera les messages pour Tartanpion avec $f_T \circ g_M$ c'est à dire en utilisant sa clé secréte puis la clé publique de Tartanpion et
Tartanpion décodera les messages provvenant de Martin avec $f_M \circ g_T$ c'est à dire en utilisant sa clé secréte et la clé publique de Martin.

12.11.3 Le cryptage des nombres avec la méthode RSA

Cette méthode est due à Rivest Shamir et Adleman en 1977. Elle est basée sur le fait qu'il est facile de savoir si un nombre entier très grand p est premier, il est facile de faire le produit n de 2 nombres premiers tres grands p et q MAIS très difficile étant donné n de retrouver p et q c'est à dire de trouver la décomposition de n en facteurs premiers.
Tartanpion pour établir ses clés, choisit deux grand nombres premiers p et q et pose $n = pq$, puis il choisit m un nombre premier avec $(p-1)(q-1)$ (par exemple il prend pour m un nombre premier plus grand que $(p-1)/2$ et que $(q-1)/2$).

Il calcule l'entier u pour que $u * m = 1 \bmod (p-1) * (q-1)$ (d'après l'identité de Bézout il existe des entiers u et v tel que $u * m = 1 + v(p-1) * (q-1)$). Puis il met dans l'annuaire les nombres u et n (quand n est grand p et q sont difficiles à obtenir à partir de n), le couple (u, n) est la clé publique alors que (m, n) est la clé secréte qui va servir à décoder le message : bien sûr p et q restent secrets, car sinon n'importe qui peut calculer m en fonction de u avec l'identité de Bézout.

La fonction de codage f est la suivante :
à un entier a inférieur à $n = pq$ f fait correspondre $a^u \bmod n$.

La fonction de décodage g est la suivante :
à un entier b, g on fait correspondre $b^m \bmod n$.

Pour montrer que $g(f(a)) = a$, on utilise le petit théorème de Fermat amélioré :
si p et q sont premiers, si $n = pq$ si a est inférieur à $n = pq$ alors :
$a^{k(p-1)(q-1)+1} = a \mod n$
On peut appliquer ce théorème ici car : - si a est premier avec n, p et q sont premiers, $n = pq$ et donc a est premier avec p et est premier avec q donc :
$a^{v(p-1)(q-1)} = 1^v = 1 \mod n$ (d'après le petit théorème de Fermat) $a^{p-1} = 1 \mod p$ et $a^{q-1} = 1 \mod q$ donc
$a^{v(p-1)(q-1)} = 1^v = 1 \mod p$ et $a^{v(p-1)(q-1)} = 1^v = 1 \mod q$
donc
$a^{v(p-1)(q-1)} = 1^v = 1 \mod n$.
- si a n'est pas premier avec n, c'est que a est soit un multiple de p soit un multiple de q (puisque $a < n = pq$). Supposons que a soit un multiple de p, a est donc premier avec q puisque $a < p * q$ et on a :
$a = 0 \bmod p$ donc $a^{k(p-1)(q-1)+1} = a = 0 \mod p$
$a^{q-1} = 1 \mod q$ car q est premier et a est premier avec q (th de Fermat), donc
$a^{k(p-1)(q-1)+1} = a \mod q$
Donc $a^{k(p-1)(q-1)+1} - a$ est un multiple de p et de q donc est un multiple de $n = pq$ (car p et q sont premiers).
on a donc bien :
$g(f(a)) = g(a^u \mod n) = a^{um} \mod n = a^{v(p-1)(q-1)+1} \mod n = a \bmod n$
Un exemple :
$p = 123456791$ et $q = 1234567891$
: $n = p * q = 152415790094497781$
$\varphi = (p-1)(q-1) = 152415788736473100$
$m = 12345701$ (m est un nombre premier et m ne divise pas φ)
On cherche u en tapant `inv(m%φ)` on trouve :
`(-36645934363466299) % 152415788736473100` et on a :
$u = -36645934363466299 + \varphi = 115769854373006801$
Pour coder, on utilise la clé publique u et n et pour décoder, on utilise la clé secréte m et n.

Remarque
Pour passer d'un message à une suite de nombres, on groupe plusieurs caractères (car sinon on pourrait décrypter le message en utilisant des statistiques de fréquence des caractères en fonction de la langue), le groupement est l'écriture d'un nombre en une base donnée (256 ici correspondant au codage ASCII d'un caractère), par exemple, puisque :
`asc("BONJOUR")=[66,79,78,74,79,85,82]`, si on groupe par 3, `BONJOUR`

devient les nombres (66*256+79)*256+78=4345678, (74*256+79)*256+85=4869973, (82*256+0)*256+0 =5373952 qui seront transformés par f en 156330358492191937, 126697584810299952, 50295601528998788 car powmod(4345678,u,n)=15633035849219193 etc...
Pour décoder on applique à ces nombres la fonction g on a :
powmod(15633035849219193,m,n)=4345678 etc...

12.11.4 La fonction de codage

Tartanpion choisit 2 grands nombres premiers p et q et met dans l'annuaire le nombre $n = p * q$. Il est facile d'obtenir n à partir de p et q mais par contre p et q sont difficiles à obtenir à partir de n car la décomposition en facteurs premiers de grands nombres est longue et presque impossible si n a plus de 130 chiffres.

Première étape

On transforme le message en une suite de nombre.
Pour cela, on découpe le message en tranche ayant $ncara$ caractères. On choisit $ncara$ en fonction des nombres p et q pour que 256^{ncara} soit inférieur à p et à q. En effet, on considère que la tranche du message que l'on veut coder est l'écriture en base 256 d'un nombre : par exemple si $ncara = 5$, on transforme "BABAR" en le nombre $a = 66 * 256^4 + 65 * 256^3 + 66 * 256^2 + 65 * 256 + 82 = 284562702674$ et on verra dans la section suivante que ces nombres a doivent être premiers avec $n = p*q$, donc par exemple être inférieurs à p et q (ce qui est vérifié si $256^{ncara} < p$ et $256^{ncara} < q$ puisque $a < 256^{ncara}$).)

Deuxième étape : un peu de maths

Le petit théorème de Fermat dit que :
si n est premier et si a est premier avec n alors $a^{n-1} = 1 \mod n$.
Une généralisation (simple) du petit théorème de Fermat est :
si p et q sont premiers, si a est quelconque, si $n = p * q$ et si k est un entier, alors :
$a^{k(p-1)*(q-1)+1} = a \mod p * q$.
Montrons cette généralisation simple :
- si a est un multiple de n c'est évident puisque
$a = 0 \bmod n$ donc $a^{k(p-1)*(q-1)+1} = a = 0 \mod n$ - si a est premier avec n alors a est premier avec q (car q est un diviseur de n),
puisque a est premier avec q on a :
$a^{q-1} = 1 \mod q$ (application du petit théorème de Fermat) et
donc $a^{k(q-1)*(p-1)} = 1 \mod q$
puisque a est premier avec p on a :
$a^{p-1} = 1 \mod p$ (application du petit théorème de Fermat)
donc $a^{k(p-1)*(q-1)} = 1 \mod p$.
On en déduit donc que :
$a^{k(p-1)*(q-1)} - 1 = 0 \mod p$ et $a^{k(p-1)*(q-1)} - 1 = 0 \mod q$ c'est à dire que :
$a^{k(p-1)*(q-1)} - 1$ est un multiple de p et de q donc de $n = p * q$ puisque p et q sont premiers.
donc $a^{k(q-1)*(p-1)} = 1 \mod n$ et donc
$a^{k(q-1)*(p-1)+1} = a \mod n$

- si a n'est pas premier avec n et si $a < n$, c'est que a est soit un multiple de p soit un multiple de q (puisque $a < n = pq$).
Supposons que a soit un multiple de p :
$a = 0 \bmod p$ donc $a^{k(p-1)(q-1)+1} = a = 0 \bmod p$
$a^{q-1} = 1 \bmod q$ car q est premier et a est premier avec q (th de Fermat), donc
$a^{k(p-1)(q-1)+1} = a \bmod q$
Donc $a^{k(p-1)(q-1)+1} - a$ est un multiple de p et de q donc est un multiple de $n = pq$ (car p et q sont premiers).

Donc si $n = p * q$ avec p et q premiers quelque soit a et quelque soit k entier on a :
$a^{k(p-1)*(q-1)} = a \mod n$.

Revenons au codage.
Soit m un nombre premier avec $(p-1)*(q-1)$ (par exemple on peut choisir pour m un nombre premier assez grand).
D'après l'identité de Bézout il existe deux entiers u et v tels que :
$u * m + v * (p-1) * (q-1) = 1$
donc :
$a^{u*m+v*(p-1)*(q-1)} = a^1$ et comme $a^{(p-1)*(q-1)} = 1 \mod n$,
$a^{u*m} = a \mod n$
La fonction f de codage sera alors :
$a -> a^u \bmod n$ pour $a < p$ et $a < q$ (pour avoir pgcd$(a, n) = 1$).
La fonction g de décodage sera alors :
$b -> b^m \bmod n$.
et on a bien $g(f(a)) = a^{u*m} = a \bmod n$ ou encore $g(f(a)) = a$ car $a < n$
La clé publique se trouvant dans l'annuaire sera (u, n),
la clé secrète sera (m, n), mais bien sûr, p et q devront rester secrets.
Remarque : u et m jouent un rôle symétrique, par exemple u est aussi premier avec $(p-1)(q-1)$ et donc si on connait u, p et q il sera aisé de retrouver m avec l'identité de Bézout ($u * m + v * (p-1) * (q-1) = 1$).

Troisième étape : le choix des clés

Comment vais-je choisir ma clé publique et ma clé secrète ?
Si on tape :
`p:=nextprime(123456789)` (`p` est un grand nombre premier),
`q:=nextprime(1234567890)` (`q` est un grand nombre premier),
`n:=p*q`
`m:=nextprime(12345678)` (`m` est un nombre premier),
`phi:=(p-1)*(q-1)`
On vérifie que m et premier avec phi en tapant :
`gcd(m,(p-1)*(q-1))`, on obtient bien `1`.
On obtient :
$p = 123456791$, $q = 1234567891$, $m = 12345701$, $phi = 152415788736473100$
et on a $n = 152415790094497781$ (n a 18 chiffres).

On cherche u et v en tapant : `iegcd(m,phi)` (`u*m+v*phi=1`)

on obtient : [-36645934363466299,2968326,1]
On tape `u:=-36645934363466299+phi` donc $u = 115769854373006801$.
Donc, ma clé publique qui se trouvera dans l'annuaire sera (u, n),
ma clé secrète sera (m, n) p et q devront rester secrets. Avec ce choix de p et de q, on va choisir de découper le message en tranches de 3 caractères ($ncara = 3$) car $256^3 = 16777216 < p < q$ et ainsi tout nombre inférieur à 256^3 sera premier avec n

Quatrième étape : le codage

Vous voulez m'envoyer le message "BABAR". Dans l'annuaire, vous trouvez en face de mon nom :
$u = 115769854373006801$ et $n = 152415790094497781$
Grâce à la première étape le mot "BABAR" est transformé en la liste de nombres `l=[4342082,16722]` car
`chaine2n("BAB") = 4342082 et chaine2n("AR") = 16722`.
Vous calculez : $f(a) = a^u \mod n$ grâce à la commande `powmod(a,u,n)`
Vous obtenez :
$f(4342082) = 4342082^{115769854373006801} = 6243987715571440 \mod n$ car
`powmod(4342082,u,n) = 6243987715571440`.
$f(16722) = 16722^{115769854373006801} = 70206283680955159 \mod n$ car
`powmod(16722,u,n) =70206283680955159`.
Le message codé est donc :
`l=[6243987715571440,70206283680955159]` et c'est cette liste de nombres que vous m'envoyez. Remarque : On ne transforme pas cette liste de nombres en un message de caractères car on risque d'avoir des caractères non imprimables.
Le codage transforme donc le message en une suite de nombres.

Cinqième étape : le décodage

Le décodage transforme une suite de nombres en un message.
Je reçois `l=[6243987715571440,70206283680955159]` pour le décoder je calcule pour chaque élément b de la liste :
$g(b) = b^m \mod n$ grâce à la commande `powmod(b,m,n)`
$g(6243987715571440) = 6243987715571440^m = 4342082 \mod n$ car
`powmod(6243987715571440,m,n)=4342082`.
$g(70206283680955159) = 70206283680955159^m = 16722 \mod n$ car
`powmod(70206283680955159,m,n)=16722`.
Il suffit maintenant de traduire le nombre $a = 4342082$ en écrivant ce nombre dans la base 256 les symboles pour écrire $0 < k < 256$ étant le caractére de code ASCII k.
Je tape :
`irem(a,256)=66`
`a:=iquo(a,256)= 16961`
puis `irem(a,256)=65`
`a:=iquo(a,256)=66`
`irem(a,256)=66`
`a:=iquo(a,256)=0`

on obtient la liste l=[66,65,66] qui correspond à "BAB"
puis pour $a = 16722$
Je tape :
`irem(a,256)=82`
`a:=iquo(a,256)= 65`
puis `irem(a,256)=65`
`a:=iquo(a,256)=0`
on obtient la liste l=[65,82] qui correspond à "AR" c'est ce que fait la fonction `ecritu256(a)` (cf 12.12), on a :
`ecritu256(4342082)="BAB"` et `ecritu256(16722)="AR"`

12.12 Les programmes correspondants au codage et décodage RSA

les programmes qui suivent se trouvent dans le fichier `rsa.xws` du menu `Exemples->arit`.

12.12.1 La première et la dernière étape

```
chaine2n(m):={
//chaine2n(m) transforme la chaine m en l'entier n
//asc(m) est l'ecriture de n dans la base 256
local l,n,s,k;
s:=size(m);
l:=asc(m);
n:=0;
for (k:=0;k<s;k++){
n:=n*256+l[k];
}
return(n);
}:;

ecritu256(n):={
//transforme l'entier n en son ecriture en base 256
local s,r;
//n est un entier et b=256, ecritu256 est une fonction iterative
//ecritu256(n)=le mot de caracteres l'ecriture de n en base 256
s:="";
while (n>=256){
r:=irem(n,256);
r:=char(r);
s:=r+s;
n:=iquo(n,256);
}
n:=char(n);
s:=n+s;
return(s);
```

```
}:;
```

12.12.2 Le codage

En principe les valeurs de p et q sont beaucoup plus grandes et donc $ncara$ le nombre de caractères par tranche peut être choisi plus grand que 3, il suffira alors dans le programme qui suit de d'initialiser `ncara` par la valeur de $ncara$ que l'on a choisie (ou de rajouter le paramètre $ncara$ et remplacer tous les 3 par $ncara$).

```
//mess est une chaine u:=115769854373006801 n:=152415790094497781
codrsa(mess,u,n):={
local s,j,j3,l,mot,ncara,k,a;
s:=size(mess);
j:=0;
ncara:=3;
j3:=ncara;
l:=[];
//j est le nombre de paquets de 3 lettres
while (j3<s) {
mot:="";
for (k:=j;k<j3;k++){
mot:=mot+mess[k];
}
//on code le mot
a:=chaine2n(mot);
l:=append(l,powmod(a,u,n));
j:=j3;
j3:=j+ncara;
}
//on code la derniere tranche du message
mot:="";
for (k:=j;k<s;k++){
mot:=mot+mess[k];
}
a:=chaine2n(mot);
l:=append(l,powmod(a,u,n));
return(l);
}:;
```

12.12.3 Le décodage

```
//l=codrsa(mess,u,n) m:=12345701 n:=152415790094497781
decodrsa(l,m,n):={
local mess,s,a,j,b;
s:=size(l);
mess:="";
for (j:=0;j<s;j++){
b:=l[j];
a:=powmod(b,m,n);
```

```
mess:=mess+ecritu256(a);
}
return(mess);
}:;
```

12.12.4 Un exemple

On envoie une liste de nombres

On veut coder le message : "Mon chat prefere Xcas" Avec la clé publique u :=115769854373006801,n :=1524157900
et
la clé secrète m :=12345701,n :=152415790094497781.
On tape :
`l:=codrsa("Mon chat prefere Xcas",u,n)`
On obtient la liste `l` de nombres qu'on envoie :

```
[31195019450305671,28742122827339904,95711325368572864,
31278264398990889,6962857192209694l,10184568821703026,4849962211742328]
```

Le receveur tape :
`decodrsa(l,m,n)`
et il obtient :
`"Mon chat prefere Xcas"`

On envoie une liste de mots

Si on veut envoyer une liste `s` de mots il faut transformer la liste `l` en une liste de chaines de caractères l'aide de `seq` (ou de `$`) et de `ecritu256`.
On tape :
`s:=seq(ecritu256(l[j]),j,0,size(l)-1)`
ou `s:=[ecritu256(l[j])$(j=0..size(l)-1)]`
On obtient la liste `s` des mots qu'on envoie :
Le receveur tape :
`decodrsa(seq(chaine2n(s[j]),j,0,size(s)-1),m,n)`
ou `decodrsa([chaine2n(s[j])$(j=0..size(s)-1)],m,n)`
et il obtient :
`"Mon chat prefere Xcas"`

12.12.5 Exercices de décodage RSA avec différents paramètres

message 1, clefs u=115769854373006801, n=15241579009449778

```
[19997497666981017, 33496307064035264,
72208210789185231, 38104201170888279,
56130089351291689, 10872985337522715 7,
13681790376832420 5, 8145824135992953 7]
```

message 2, clefs u=115769854373006801, n=152415790094497781

```
[5834920343570953 1, 7502863100031789 0,
1019566127374431 04, 1101750825485934 87,
1184930166171944 52,1065384238233700 2,
6570991809001483 7, 2450934362584911 7,
7722620718024227 9, 1131563552163370 45]
```

message 3, clefs u=115769854373006801, n=152415790094497781

```
[3398048298823510 9, 1168259168532919 98,
1138959243675306 43, 9577505724898785 7,
2497760833545064 8, 1349681494824893 39,
7633443621034964 8, 9892507587764063 5,
2055528828480682 8]
```

message 4, clefs u=115769854373006801, n=152415790094497781

```
[9935288724599470 2, 7452725220300033,
9909751526214318 8, 3501895711969483 6,
7614940334689785 1, 1705274290394831 5,
3440100126332340 2, 1469339642118936 03]
```

message 5, clefs u=115769854373006801, n=152415790094497781

```
[6988500542307453 0, 7148268064017490 2,
7656605918169581 5, 1368179037683242 05,
3410697347415599 8, 3362040476775297 1,
1272929923465425 9,205315941338105 98]
```

message 6, clefs u=115769854373006801, n=152415790094497781

```
[58349203435709531, 6861418969816861 3,
9589476884466006 2, 130069211243087116,
130376871729616040, 7430617831751448 2,
125801418681172709, 1289225337694278 56,
13890216847974905 4]
```

message 7, clefs u=115769854373006801, n=152415790094497781

```
[82887698188763362, 14731779436255034 0,
11063280558757506, 6234756056463983 1,
6659119245519999 4, 10868746079603436 2,
6869845641870417 1, 11389592436753064 3,
9784074299104039 6, 10313006117740585 1,
2174406457316784 3, 8181038488770470 6]
```

message 8, clefs u=115769854373006801, n=152415790094497781

```
[42711799087786740, 13487874449017248 2,
14943935892612023 8, 2544247918436293 5,
1828072730594369, 2874212282733990 4,
7733348672374875 8]
```

message 9, clefs u=115769854373006801, n=152415790094497781

```
[14362386639974804 5, 7966012486327335,
8244655567157720 7, 5936371884570574 4,
1168695406844930 84, 2721907916351248 9,
2721907916351248 9, 1152569140375993 94,
8112317737182418 1, 9916682644608358 8,
9064828288305782 0, 2831442569765061 4,
1477449663994837 01, 2490350695468404 6]
```

message 10, clefs u=115769854373006801, n=152415790094497781

```
[21958089817862266, 12334910996796687O,
5192784531555568 9, 95894768844660062,
24509343625849117, 31027419256533256,
125503703895953175, 33160330760344892,
61040361422718323, 9544287545681754,
61022858667046639]
```

message 11, clefs u=115769854373006801, n=152415790094497781

```
[629819766882008 42, 645363026003100 87,
613108405169442 12, 7486629931368896,
17472057692137769, 130815067487053887,
533516635949841 81, 144381092812007128,
41251258816315103, 114751369092267608]
```

message 12, clefs u=115769854373006801, n=152415790094497781

```
[123477331568497546, 464988847984841 69,
99097515262143188, 7837029050945492,
17052742903948315, 106657774868209674,
486901668996752 67, 79883234756846796,
11849301661719445 2, 10653842382337002,
12851441215419853 9, 14257929600934347 7,
28886262313123302, 76149403346897851,
12346636557859071 1, 12311534895655687 7]
```

12.12.6 Solutions des exercices de décodage

1 : vous allez gagner un CD
2 : la solution n'est pas evidente
3 : vive les logiciels libres
4 : rien ne sert de courir
5 : appelons un chat un chat

6 : la reponse releve du defi
7 : habiletes calculatoires obligatoires
8 : il fait beau et chaud
9 : l'utilisation de ce codage est discutable
10 : nous voulons eviter la confusion
11 : beaucoup de travail pour rien
12 : science sans conscience n'est que ruine de l'ame

12.13 Codage RSA avec signature

Avec des codages à clé publique comme RSA, n'importe qui peut vous envoyer un message codé. La question qui se pose est : comment être sûr de de l'identité de l'envoyeur ?
Avec le codage RSA, c'est assez facile car si Tartanpion m'envoie un message codé avec signature, il va coder et signer le message en utilisant **ma** clé publique et **sa** clé sectète.
Voici par exemple, les clés de codage et de décodage de Tartanpion.

```
ptar:=nextprime(223456789)
qtar:=nextprime(823456789)
mtar:=nextprime(32345678)
phitar:=(ptar-1)*(qtar-1)
ntar:=ptar*qtar
```

On obtient :
`ptar=223456811`,
`qtar= 823456811`,
`mtar= 32345689` et
`phitar=184007031935376100`
`ntar=184007032982289721` (`ntar` a 18 chiffres)
et on vérifie que `mtar` et premier avec `phitar` en tapant :
`gcd(mtar,(ptar-1)*(qtar-1))` on obtient bien `1`.
On cherche `utar` et `vtar` en tapant : `egcd(mtar,phitar)`
on obtient : [-44971265178398091,7905277,1].
On tape `utar:=-44971265178398091+phitar` donc
`utar=139035766756978009`.
Donc, la clé publique de Tartanpion (celle qui se trouve dans l'annuaire) sera `(utar,ntar)`,
sa clé secrète sera `(mtar,ntar)` mais `ptar` et `qtar` devront rester secrets.
Tartanpion va coder le message `mess` qu'il veut m'envoyer selon un programme analogue à `codrsa(mess,u,n)` mais avant de mettre les nombres `b` dans la liste `l` il va utiliser sa fonction de décodage selon sa clé secrète et mettera dans `l` les nombres `powmod(b,mtar,ntar)`.
Il m'envoie donc `codrsas(mess,u,n,mtar,ntar)` (voir le programme ci-dessous).
Voici le détail du (les programmes qui suivent se trouvent dans le fichier `rsas`) :

```
//mess=chaine
//u:=115769854373006801;n:=152415790094497781; (ma cle publique)
```

```
//ntar:=184007032982289721; mtar:=32345689; (cle secrete de Tar)
codrsas(mess,u,n,mtar,ntar):={
local s,j,j3,l,mot,a,b,ncara;
s:=size(mess);
j:=0;
ncara:=3;
j3:=ncara;
l:=[];
//j est l'indice du premier élément d'un paquet de 3 lettres
while (j3<=s) {
mot:="";
for (k:=j;k<j3;k++){
mot:=mot+mess[k];
}
//on code le mot
a:=chaine2n(mot);
b:=powmod(a,u,n);
//fct de codage selon la cle publique (u,n) du receveur puis
//fct de decodage selon la cle secrete de l'envoyeur (mtar,ntar)
l:=append(l,powmod(b,mtar,ntar));
j:=j3;
j3:=j+ncara;
}
//on code la derniere tranche du message
mot:="";
for (k:=j;k<s;k++){
mot:=mot+mess[k];
}
a:=chaine2n(mot);
b:=powmod(a,u,n);
l:=append(l,powmod(b,mtar,ntar));
return(l);
}:;
```

Pour décoder il me suffira de coder les nombres `b` de la liste `l` en utilisant la clé publique de celui qui a signé le message (`a:=powmod(b,utar,ntar)`) puis, de décoder `a` en utilisant ma clé secréte (`b:=powmod(a,u,n)`).

```
//l=codrsas(mess,u,n,mtar,ntar)
// m:=12345701; n:=152415790094497781; ma cle secrete (receveur)
//ntar:=184007032982289721;utar:=139035766756978009; cle pub de T
decodrsas(l,m,n,utar,ntar):={
local mess,s,a,j,b;
s:=size(l);
mess:="";
for (j:=0;j<s;j++){
b:=l[j];
//codage selon la cle publique (utar,ntar) de l'envoyeur (T)
a:=powmod(b,utar,ntar);
```

```
//decodage selon la cle secrete du receveur (m,n) (moi)
b:=powmod(a,m,n);
mess:=mess+ecritu256(b);
}
return(mess);
}:;
```

Je reçois un message `l` signé de Tartanpion : je le décode en utilisant sa clé publique et ma clé secrète en tapant :
`decodrsas(l,m,n,utar,ntar)`
Voici le détail avec `mess:="demain 10 heures gare de Grenoble"`.

```
l:=codrsas(mess,u,n,mtar,ntar)
l:= [137370234628529043,113626149789068692,125222577739438308,
33473651820936779,42708525589347295,23751805405519257,
66289870504591745]
decodrsas(l,m,n,utar,ntar)="demain 10 heures gare de Grenoble"
```

12.13.1 Quelques précautions

Lorsqu'on envoie un message, il ne faut utiliser que les caractères dont les codes ASCII vont de 32 à 127. Il faut par exemple se méfier des caractères accentués qui n'ont pas toujours le même code... Pour un codage avec signature on a des problèmes si on tape :
`messcs:=decodrsa(codrsa(mess,u,n),mtar,ntar)` car alors dans `messcs` il figure très certainement des caractères qui ont des codes inférieurs à 32 ou des codes supérieurs à 127.
Pour mémoire :
`decodrsa(codrsa(mess,u,n),m,n)=mess` car `mess` n'a que des caractères qui on des codes compris entre 32 et 127.

Chapitre 13

Algorithmes sur les suites et les séries

L'objectif ici est de traduire les algorithmes en l'écriture de programmes. On écrit ici des programmes permettant d'avoir les termes d'une suite ou d'une série et de trouver des valeurs approchées de leur limites.
Mais, pour édutier les suites et les séries, on peut aussi utiliser le tableur ce qui est souvent plus facile que d'écrire un programme.

13.1 Les suites

Soit u_n une suite de réels définie soit par $u_n = f(n)$, soit par une relation de récurrence $u_n = f(u_{n-m}, .., u_{n-1})$ et la donnée de ses premiers termes. On veut ici, calculer les valeurs de u_n. Pour les fonctions qui suivent, il suffira de rajouter la fonction `evalf` dans le `return` pour avoir une valeur approchée de u_n : par exemple `return evalf(uk)`.

13.1.1 Les suites $u_n = f(n)$

Pour avoir le n-ième terme u_n il suffit :
de définir la fonction f et de taper `f(n)`.
On peut aussi mettre f comme paramètre et taper :

```
u(f,n):=f(n)
```

Ainsi `u(sq,3)` vaut `9` et `u(sqrt,3)` vaut `sqrt(3)`.
On remarquera qu'il est souvent préférable de simplifier l'écriture de `u(f,n)` avec la commande `normal` : mettre plutôt `normal` dans la définition de `f`.
Par exemple on définit : `f(x):=normal(x/sqrt(3)+sqrt(3))`.
On tape :
`u(f,3)`
On obtient :
`2*sqrt(3)`
On peut aussi considérer qu'il n'y a qu'un paramètre l qui est la séquence f, n et définir `u` par :

```
u(l):=l[0](l[1])
```

Pour avoir la suite des termes u_k, pour k allant de $k0$ à n, on écrit :

```
utermes(f,k0,n):={
local k,lres;
lres:=NULL;
for (k:=k0;k<=n;k++){
     lres:=lres,f(k);
}
return lres;
}
```

On a choisit de mettre tous les termes cherchés dans une séquence.
On a : `lres:=NULL;` initialise la séquence à vide.
Par exemple, avec la fonction :
`f(x):=normal(x/sqrt(3)+sqrt(3))`.
On tape :
`utermes(f,0,5)`
On obtient :
`sqrt(3),4*sqrt(3)/3,5*sqrt(3)/3,2*sqrt(3),7*sqrt(3)/3,8*sqrt(3)/3`

13.1.2 La représentation des suites $u_n = f(n)$

On va représenter une suite u_n par des segments verticaux : le terme u_p sera représenté par le segment joignant le point $(p, 0)$ au point $(p, f(p))$.
Pour faire cette représentation, on définit la fonction f et on valide le programme suivant qui permet de représenter $u_{j1} = f(j1)..u_{j2} = f(j2)$.

```
plotsuite(f,j1,j2):={
local j,P,L;
L:=NULL;
for (j:=j1;j<=j2;j++) {
P:=point(j+i*u(j),couleur=point_width_4+noir);
L:=L,segment(j,P,couleur=ligne_tiret+rouge),P;
}
return L;
};
```

Exemple
`u(n):=1+(-1)^n/n`
puis,
`plotsuite(u,0,10)`
puis,
`plotsuite(u,20,30)`

13.1.3 La représentation des suites récurrentes $u_0 = a, u_n = f(u_{n-1})$

Pour avoir les termes $u_0 = a, ...u_p$, on définit la fonction f puis on tape :

```
plotsuiterec1(f,a,p):={
local j,P,L;
```

```
L:=NULL;
a:=evalf(a);
for (j:=0;j<=p;j++) {
P:=point(j+i*a,couleur=point_width_4+noir);
L:=L,segment(j,P,couleur=ligne_tiret+rouge),P;
a:=f(a);
}
return L;
};
```

Exemple
`f(x):=x^2-2`
puis,
`plotsuiterec1(f,0,10)`
ou,
`plotsuiterec1(sq-2,0,10)`

13.1.4 La représentation des suites récurrentes $[u_0, u_1, ...u_{s-1}] = la$, $u_n = f(u_{n-s}..., u_{n-1})$ si $n >= s$

Pour avoir les termes $u_0 = la[0], u_1 = la[1], ...u_p$, on définit la fonction f puis on tape :

```
plotsuiterec(f,la,p):={
local j,P,L,s,a;
L:=NULL;
s:=size(la);
la:=evalf(la);
for (j:=0;j<s;j++) {
P:=point(j+i*la[j],couleur=point_width_4+noir);
L:=L,segment(j,P,couleur=ligne_tiret+rouge),P;
}
for (j:=s;j<=p;j++) {
a:=f(op(la));
P:=point(j+i*a,couleur=point_width_4+noir);
L:=L,segment(j,P,couleur=ligne_tiret+rouge),P;
la:=append(tail(la),a);
}
return L;
};
```

Exemple
`f(x,y):=x+y`
puis,
`plotsuiterec(f,[0,1],6)`

13.1.5 L'escargot des suites récurrentes $u(0) = a$, $u(n) = f(u(n-1)$ si $n > 0$

Cette session se trouve dans `plottoile.xws`.
On rappelle que la commande `plotseq(f(x),a,p)` permet de visualiser les

p premiers termes de la suite récurrente $u_0 = a$, $u_n = f(u_{n-1})$ si $n > 0$ en visualisant "l'escargot". On se propose de réécrire cette commande de façon a bien mettre en evidence la construction des différents termes de la suite.
À la différence de `plotseq` la fonction `plottoile` a comme premier argument la fonction f et non l'expression $f(x)$. On représente le premier terme u_0 par une croix noire sur l'axe des x d'abscisse le deuxième argument. Pour avoir u_1, on trace le segment vertical allant de la croix au graphe de $f(x)$, puis le segment horizontal allant, du point du graphe de $f(x)$ au graphe de la première bissectrice (pour reporter la valeur de $f(u_0)$ sur l'axe des x), puis, un segment vertical en pointillés allant, du graphe de la première bissectrice à l'axe des x pour tracer une croix rouge.
Pour avoir les termes suivants, on trace ensuite un segment vertical allant du point du graphe de la première bissectrice, au graphe de $f(x)$, puis le segment horizontal allant, du point du graphe de $f(x)$ au graphe de la première bissectrice, puis, un segment vertical jusqu'à l'axe des x pour tracer une croix rouge etc... On s'arrête lorsque l'on a dessiné p croix rouges, p etant le troisième argument.
On tape :

```
plottoile(f,u,n):={
local j,v,L,P;
u:=evalf(u);
P:=point(u,couleur=point_width_4+noir);
if (n<=0 ) {return P;}
v:=f(u);
L:=segment(u,u+i*v,couleur=rouge),P;
L:=L,segment(u+i*v,v+i*v,couleur=rouge);
u:=v;
v:=f(u);
P:=point(u,couleur=point_width_4+rouge);
L:=L,segment(u,u+i*u,couleur=ligne_tiret+rouge),P;
for (j:=2;j<=n;j++) {
L:=L,segment(u+i*u,u+i*v,couleur=rouge);
L:=L,segment(u+i*v,v+i*v,couleur=rouge);
u:=v;
v:=f(u);
P:=point(u,couleur=point_width_4+rouge);
L:=L,segment(u,u+i*u,couleur=ligne_tiret+rouge),P;}
return plotfunc(f(x),x,couleur=vert),plotfunc(x,x,couleur=bleu),L;
};
```

Par exemple pour voir les premiers termes de :
$u_0 = 2$, si $n \geq 1$, $u_n = \cos(u_{n-1})$, on tape : `plottoile(cos,2,5)`

On peut aussi noter les indices des termes de la suite pour la croix représentant u_j en rajoutant `legende(u,j,quadrant4)`.

```
plottoilegende(f,u,n):={
local j,v,L,P;
u:=evalf(u);
```

```
P:=point(u,couleur=point_width_4+noir),legende(u,0,quadrant4);
if (n<=0 ) {return P;}
v:=f(u);
L:=segment(u,u+i*v,couleur=rouge),P;
L:=L,segment(u+i*v,v+i*v,couleur=rouge);
u:=v;
v:=f(u);
P:=point(u,couleur=point_width_4+rouge),legende(u,1,quadrant4);
L:=L,segment(u,u+i*u,couleur=ligne_tiret+rouge),P;
for (j:=2;j<=n;j++) {
L:=L,segment(u+i*u,u+i*v,couleur=rouge);
L:=L,segment(u+i*v,v+i*v,couleur=rouge);
u:=v;
v:=f(u);
P:=point(u,couleur=point_width_4+rouge),legende(u,j,quadrant4);
L:=L,segment(u,u+i*u,couleur=ligne_tiret+rouge),P;}
return plotfunc(f(x),x,couleur=vert),plotfunc(x,x,couleur=bleu),L;
};
```

Par exemple pour voir les premiers termes avec leur indice de :
$u_0 = 2$, si $n \geq 1$, $u_n = \cos(u_{n-1})$, on tape : `plottoilegende(cos,2,5)`

On peut aussi faire une animation qui montrera la progression de la construction. Pour cela on modifie la fonction `plottoile` en `toile` pour avoir dans `LT`, la progression du tracé. On remarquera que l'on met entre crochet les objets graphiques qui seront affichés simultanément lors de l'animation.
On tape :

```
toile(f,u,n):={
local j,v,L,P,LT;
u:=evalf(u);
P:=point(u,couleur=point_width_4+noir);
v:=f(u);
LT:=P;
L:=segment(u,u+i*v,couleur=rouge);
L:=L,segment(u+i*v,v+i*v,couleur=rouge);
u:=v;
v:=f(u);
P:=point(u,couleur=point_width_4+rouge);
L:=L,segment(u,u+i*u,couleur=ligne_tiret+rouge),P;
LT:=LT,[LT,L];
for (j:=2;j<=n;j++) {
L:=L,segment(u+i*u,u+i*v,couleur=rouge);
L:=L,segment(u+i*v,v+i*v,couleur=rouge);
u:=v;
v:=f(u);
P:=point(u,couleur=point_width_4+rouge);
L:=L,segment(u,u+i*u,couleur=ligne_tiret+rouge),P;
LT:=LT,[LT,L];
```

```
}
return LT;
};
```

Puis on anime la liste `LT` renvoyée par `toile`.

```
animtoile(f,u,n):={
local LT;
LT:=toile(f,u,n);
return plotfunc(f(x),x,couleur=vert),
       plotfunc(x,x,couleur=bleu),
       animation(LT);
};
```

Par exemple pour voir en animation les premiers termes de :
$u_0 = 2$, si $n \geq 1$, $u_n = \cos(u_{n-1})$, on tape : `animtoile(cos,2,5)`
On peut régler la vitesse d'animation avec `Menu ->Animation` (situé dans le pavé de boutons à droite de la fenêtre graphique).
On peut arrêter l'animation avec le bouton ▶| (à gauche de `Menu`) : il suffit alors, de cliquer dans la fenêtre graphique, pour que l'animation se déroule au pas à pas.

13.1.6 Les suites récurrentes définies par une fonction de plusieurs variables

Un exemple : la suite de Fibonnacci

Commençons par un exemple : la suite de Fibonnacci définie par :
$u_0 = a$
$u_1 = b$
$u_n = u_{n-1} + u_{n-2}$ pour $n \geq 2$
On écrit pour avoir u_n :

```
fibon(a,b,n):={
local k,uk;
for (k:=2;k<=n;k++) {
    uk:=a+b;
    a:=b;
    b:=uk;
}
return uk;
}
```

On écrit pour avoir $u_0, u_1 ... u_n$:

```
fibona(a,b,n):={
local k,uk,res;
res:=a,b;
for (k:=2;k<=n;k++) {
    uk:=a+b;
    a:=b;
    b:=uk
```

```
    res:=res,uk;
}
return res;
}
```

On écrit pour avoir $u_c, u_{c+1}...u_n$ pour $c \geq 0$:

```
fibonac(a,b,c,n):={
local k,uk,res;
for (k:=2;k<c;k++) {
    uk:=a+b;
    a:=b;
    b:=uk
};
if c>1 res:=NULL else
    if c==0 {res:=a,b;c:=2;} else
        if c==1 {res:=b;c:=2};
for (k:=c;k<=n;k++) {
    uk:=a+b;
    a:=b;
    b:=uk
    res:=res,uk;
}
return res;
}
```

Remarque On peut bien sûr écrire un programme récursif qui donne la valeur de u_n. Mais cela n'est pas efficace car on calcule plusieurs fois le même terme. Car par exemple pour calculer u_5 on doit calculer u_3 et u_4 et pour calculer u_4, il faudra à nouveau calculer u_3 et u_2, donc u_3 sera calculé 2 fois et u_2 sera calculé 3 fois. Pour s'en rendre compte on peut imprimer les valeurs des variables pour chacun des appels récursifs.

On a :
$u_0 = u0$
$u_1 = u1$
$u_n = u_{n-1} + u_{n-2}$ pour $n \geq 2$
Donc

```
fibonr(u0,u1,n):={
if (n==0) {print(u0,u1,n);return u0;}
if (n==1) {print(u0,u1,n)return u1;}
print(u0,u1,n)
return fibonr(u0,u1,n-2)+fibonr(u0,u1,n-1);
}:;
```

On peut aussi compter le nombre de fois que la fonction a ètè appelée c'est à dire le nombre de `print` du programme `fibonr` précédent :

```
fibona(u0,u1,n):={
if (n==0) {return [u0,1];}
```

```
if (n==1) {return [u1,1];}
print(u0,u1,n)
return fibona(u0,u1,n-2)+fibona(u0,u1,n-1)+[0,1];
}
```

On tape : `fibona(1,1,6)`
On obtient : `[13,25]`
On a :
1 appel avec $n = 6$,
1 appel avec $n = 5$,
2 appels avec $n = 4$,
3 appels avec $n = 3$,
5 appels avec $n = 2$,
8 appels avec $n = 1$,
5 appels avec $n = 0$,
On remarque que le nombre d'appels est une suite de Fibonacci et on voit que pour calculer u_n on doit calculer u_2 `fibon(1,1,n-2` fois !

Suites récurrentes definies par une fonction de m variables

On suppose maintenant que la suite est définie par une relation de récurrence definie par une fonction f de m variables : pour définir la suite on se donne les m premiers termes :
$u_0, u_1, .., u_{m-1}$ et la relation :
$u_n = f(u_{n-m}, u_{n-m+1}, .., u_{n-1})$ pour $n \geq m$.
On veut calculer u_n, et on suppose que les valeurs de $u_0, u_1, .., u_{m-1}$ sont dans la liste `l0`.
On écrit :

```
urec(f,n,l0):={
local s,k,uk;
s:=size(l0);
l0:=op(l0);
for (k:=s;k<=n;k++) {
    uk:=f(l0);
    l0:=tail(l0),uk;
}
return uk;
}
```

On utilise `op` au début, pour transformer la liste `l0` en une séquence et `tail(l0)` pour enlever le premier élément et ainsi `l0:=tail(l0),uk` est une séquence qui a toujours `s` éléments.

On peut aussi considérer que le paramètre l contient toutes les variables à savoir $l = f, n, u_0, .., u_{m-1}$. On écrit mais c'est inutilement compliqué (!) :

```
urecs(l):={
local f,n,s,k,uk;
f:=l[0];
```

```
n:=l[1];
l:=tail(tail(l));
s:=size(l);
//f est une fonction de s variables
for (k:=s;k<=n;k++) {
    uk:=f(l);
    l:=tail(l),uk;
}
return uk;
}
```

Pour avoir tous les termes u_k de la suite pour k allant de 0 à n, On considère que le paramètre l contient toutes les variables à savoir $l = f, n, u_0, .., u_{m-1}$.
On écrit :

```
urec_termes(l):={
local f,n,s,k,uk,lres;
f:=l[0];
n:=l[2];
l:=tail(tail(tail(l)));
s:=size(l);
//f est une fonction de s variables
lres:=l;
for (k:=s;k<=n;k++) {
    uk:=f(l);
    lres:=lres,uk;
    l:=tail(l),uk;
}
return lres;
}
```

Par exemple on définit :
`f(x,y):=normal(x+y)`
On tape :
`urec_termes(f,5,1,1)`
On obtient la suite de Fibonacci :
`1,1,2,3,5,8`
On tape :
`urec_termes(f,5,1,(sqrt(5)+1)/2)`
On obtient :
`1,(sqrt(5)+1)/2,(sqrt(5)+3)/2,sqrt(5)+2,`
`(3*sqrt(5)+7)/2,(5*sqrt(5)+11)/2`
On tape, pour vérifier que l'on a obtenu la suite géométrique de raison `(sqrt(5)+1)/2` :
`seq(normal(((sqrt(5)+1)/2)^k),k=0..5)`
On obtient :
`1,(sqrt(5)+1)/2,(sqrt(5)+3)/2,sqrt(5)+2,`
`(3*sqrt(5)+7)/2,(5*sqrt(5)+11)/2`

Pour avoir tous les termes u_k de la suite pour k allant de $k0$ à n, On considère que le paramètre l contient toutes les variables à savoir $l = f, k0, n, u_0, .., u_{m-1}$.

On écrit :

```
urec_termekn(l):={
local f,n,s,k,uk,k0,lres;
f:=l[0];
k0:=l[1];
n:=l[2];
l:=tail(tail(tail(l)));
s:=size(l);
//f est une fonction de s variables
for (k:=s;k<k0;k++) {
    uk:=f(l);
    l:=tail(l),uk;
};
if k0>1 res:=NULL else
  if k0==0 {res:=a,b;k0:=2;} else
    if k0==1 {res:=b;k0:=2};
for (k:=k0;k<=n;k++) {
    uk:=f(l);
    lres:=lres,uk;
    l:=tail(l),uk;
}
return lres;
}
```

Par exemple on définit :
`f(x,y):=normal(x+y)`
On tape :
`urec_termekn(f,5,10,1,1)`
On obtient la suite de Fibonacci :
`8,13,21,34,55,89`
On tape :
`urec_termes(f,5,9,1,(sqrt(5)+1)/2)`
On obtient :
`5*sqrt(5)+11)/2,4*sqrt(5)+9,(13*sqrt(5)+29)/2,`
`(21*sqrt(5)+47)/2,17*sqrt(5)+38`

13.2 Les séries

Soit u_n une suite de réels telle que la série $\sum_{k=0}^{\infty} u_k$ converge vers S. On veut ici, calculer une valeur appochée de cette somme. Si la série converge rapidement, il suffit de calculer $\sum_{k=0}^{n} u_k$ pour n assez grand, sinon il faut procéder à une accélération de convergence, en construisant une série de même somme et convergeant plus rapidement.

13.2.1 Les sommes partielles

On écrit :

```
sum_serie(f,n0,n):={
local s,k;
//un=f(n) ou f est une fonction de 1 variable
s:=0;
for (k:=n0;k<=n;k++) {
    s:=s+evalf(f(k));
}
return s;
}
```

Il est plus précis de faire le calcul de la somme en commençant par les plus petits termes, on écrit :

```
serie_sum(f,n0,n):={
local s,k;
//un=f(n) ou f est une fonction de 1 variable
s:=0;
for (k:=n;k>=n0;k--) {
    s:=s+evalf(f(k));
}
return s;
}
```

On peut avoir aussi besoin de la suite des sommes partielles : par exemple pour les séries alternées deux sommes partielles successives encadrent la somme de la série.
On écrit en utilisant un paramètre supplémentaire `alt` pour repérer les séries alternées de la forme $\mathtt{u_n} = \mathtt{alt}^{\mathtt{n}} * \mathtt{f}(\mathtt{n})$:

```
sums_serie(f,n0,n,alt):={
local ls,s,k;
//un=(alt)^n*f(n) ou f est une fonction de 1 variable
s:=0;
ls:=[];
if (alt<0){
  if (irem(n0,2)==0) {alt:=-alt;}
  for (k:=n0;k<=n;k++) {
    s:=s+evalf(alt*f(k));
    alt:=-alt;
    ls:=concat(ls,s);
  }
}
else {
  for (k:=n0;k<=n;k++) {
    s:=s+evalf(alt*f(k));
    ls:=concat(ls,s);
```

```
  }
}
return ls;
}
```

13.2.2 Un exemple simple : une approximation de e

On va calculer la somme :
$\sum_{k=0}^{n} \frac{1}{k!}$ en commencant par les plus petits termes :

```
vale0(n):={
local S,k;
S:=0;
for (k:=n;k>=0;k--){
S:=S+1/k!;
}
return S;
}
:;
```

On tape :
```
SS:=vale0(22)
iquo(numer(SS)*10^22,denom(SS))
```
On obtient :
```
2718281828459045235360 2
```

On va calculer la somme :
$S = \sum_{k=0}^{n} \frac{1}{k!}$ en calculant $k!$ au fur et à mesure dans f

```
vale1(n):={
local S,f,k;
f:=1;
S:=1;
for (k:=1;k<=n;k++){
f:=f*k;
S:=S+1/f;
}
return S;
}
:;
```

On tape :
```
S:=vale1(22)
iquo(numer(S)*10^22,denom(S))
```
On obtient :
```
2718281828459045235360 2
```

On va calculer la somme :
$S = \sum_{k=0}^{n} \frac{1}{k!}$ en calculant son numérateur p et son dénominateur f à chaque étape : on a $p/(k-1)! + 1/k! = (p*k+1)/k!$ et $k! = f*k$. Le résultat obtenu

en cherchant le quotient de $10^n * p$ par f donnera les n chiffres significatils de e i.e $e = vale(n) * 10^{-n}$ lorsque $n \geq 22$ car $1./23! < 4e - 23$

```
vale(n):={
local p,f,k;
f:=1;
p:=1;
for (k:=1;k<=n;k++){
f:=f*k;
p:=p*k+1;
}
return iquo(p*10^n,f);
}
:;
```

On tape :
`vale(22)`
On obtient :
`2718281828459045235360 2`

13.2.3 Exemple d'accélération de convergence des séries à termes positifs

On suppose que $u_n = f(n)$ et que $f(n)$ admet un développement limité à tous les ordres par rapport à $1/n$.
On suppose que $u_k \sim a/k^p$ et on pose :
$v_k = u_k - \dfrac{a}{(k+1)(k+2)...(k+p)}$
On a alors, $v_k = O(\dfrac{1}{k^{p+1}})$ et on connait :
$\sum_{k=0}^{\infty} \dfrac{a}{(k+1)(k+2)...(k+p)}$
En effet :
$$\frac{a}{(k+1)(k+2)...(k+p)} = \frac{a}{p-1}(\frac{1}{(k+1)(k+2)...(k+p-1)} - \frac{a}{(k+2)(k+3)...(k+p)})$$
donc
$$\sum_{k=0}^{\infty} \frac{a}{(k+1)(k+2)...(k+p)} = \frac{a}{p-1}(\frac{1}{1\cdot 2\cdot ...\cdot (p-1)}) = \frac{a}{(p-1)(p-1)!} \text{ et,}$$
$$\sum_{k=k_0}^{\infty} \frac{a}{(k+1)(k+2)...(k+p)} = \frac{a}{(p-1)(k_0+1)(k_0+2)..(k_0+p-1)}$$
On a :
$$\sum_{k=0}^{\infty} u_k = \frac{a}{(p-1)(p-1)!} + \sum_{k=0}^{\infty} v_k$$
On peut ensuite continuer à appliquer la même méthode à v_k.

Exercice calcul de $\pi^2/6$

Utiliser cette méthode pour calculer numériquement : $\pi^2/6 = \sum_{k=0}^{\infty} \dfrac{1}{(k+1)^2} =$

$\sum_{n=1}^{\infty} \frac{1}{(n)^2}$.
On va faire "à la main" trois accélérations successives.
On pose :
$u_k = \frac{1}{(k+1)^2}$

— 1-ière accélération :
$u_k = \frac{1}{(k+1)^2} \sim \frac{1}{k^2}$ donc on pose
$v_k = u_k - \frac{1}{(k+1)(k+2)}$, et donc
$v_k = \frac{1}{(k+1)^2(k+2)}$
puisque $\frac{1}{(k+1)(k+2)} = \frac{1}{(k+1)} - \frac{1}{(k+2)}$, on a :
$\sum_{k=0}^{\infty} \frac{1}{(k+1)(k+2)} = 1$ donc
$\sum_{k=0}^{\infty} u_k = 1 + \sum_{k=0}^{\infty} v_k$

— 2-ième accélération :
$v_k = \frac{1}{(k+1)^2(k+2)} \sim \frac{1}{k^3}$ donc on pose
$w_k = v_k - \frac{1}{(k+1)(k+2)(k+3)}$, et donc
$w_k = \frac{2}{(k+1)^2(k+2)(k+3)}$
puisque $\frac{2}{(k+1)(k+2)(k+3)} = \frac{1}{(k+1)(k+2)} - \frac{1}{(k+2)(k+3)}$, on a :
$\sum_{k=0}^{\infty} \frac{1}{(k+1)(k+2)(k+3)} = \frac{1}{2*2!} = \frac{1}{4}$ donc
$\sum_{k=0}^{\infty} u_k = 1 + \frac{1}{2 \cdot 2!} + \sum_{k=0}^{\infty} w_k$

— 3-ième accélération :
$w_k = \frac{2}{(k+1)^2(k+2)(k+3)} \sim \frac{2}{k^4}$ donc on pose
$t_k = w_k - \frac{2}{(k+1)(k+2)(k+3)(k+4)}$, et donc
$t_k = \frac{6}{(k+1)^2(k+2)(k+3)(k+4)}$
puisque $\frac{3}{(k+1)(k+2)(k+3)(k+4)} = \frac{1}{(k+1)(k+2)(k+3)} - \frac{1}{(k+2)(k+3)(k+4)}$, on a :
$\sum_{k=0}^{\infty} \frac{2}{(k+1)(k+2)(k+3)(k+4)} = \frac{2}{3*3!} = \frac{1}{18}$ donc
$\sum_{k=0}^{\infty} u_k = 1 + \frac{1}{2*2!} + \frac{2}{3*3!} + \sum_{k=0}^{\infty} t_k$

On tape :
`u(k):=1/(k+1)^2`
On tape :
`v(k):=1/((k+1)^2*(k+2))`
On tape :

```
w(k):=2/((k+1)^2*(k+2)*(k+3))
```
On tape :
```
t(k):=6/((k+1)^2*(k+2)(k+3)(k+4))
```
On compare $\frac{\pi^2}{6}$ et les valeurs obtenues pour $n = 200$, car on sait que :

$$S = \sum_{k=0}^{\infty} \frac{1}{(k+1)^2} = \frac{\pi^2}{6} \simeq 1.64493406685$$

On tape :
```
serie_sum(u,0,200)
```
ou
```
evalf(sum(1/(k+1)^2,k=0..200))
```
On obtient S à $5 * 10^{-3}$ prés (1 décimale exacte) :
```
1.63997129788
```
On tape :
```
1+serie_sum(v,0,200)
```
ou
```
evalf(1+sum(1/((k+1)^2*(k+2)),k=0..200))
```
On obtient S à $1.25 * 10^{-5}$ prés (4 décimales exactes) :
```
1.64492179293
```
On tape :
```
1+1/4+serie_sum(w,0,200)
```
ou
```
evalf(1+1/4+sum(2/((k+1)^2*(k+2)*(k+3)),k=0..200))
```
On obtient S à $8.3 * 10^{-8}$ prés (5 décimales exactes) :
```
1.64493398626
```
On tape :
```
1+1/4+1/9+serie_sum(t,0,200)
```
ou
```
evalf(1+1/4+1/9+sum(6/((k+1)^2*(k+2)*(k+3)*(k+4)),k=0..200))
```
On obtient S à $9.2 * 10^{-10}$ prés (8 décimales exactes) :
```
1.64493406596
```

Les erreurs

Si on compare la somme $\sum_{k=n+1}^{\infty} 1/(k+1)^2$ à une intégrale on a :

$$\sum_{k=n+1}^{\infty} \frac{1}{(k+1)^2} < \int_n^{\infty} \frac{1}{(x+1)^2} dx = \frac{1}{n+1}$$

Ou encore, on peut aussi remarquer que :

$$\sum_{k=n+1}^{\infty} \frac{1}{(k+1)^2} < \sum_{k=n+1}^{\infty} \frac{1}{k(k+1)} = \frac{1}{n+1}$$

puisque $\frac{1}{k(k+1)} = \frac{1}{k} - \frac{1}{k+1}$.
Au bout de la p-ième accéleration on calcule la somme de :
$u_k^{(p)} = \frac{p!}{(k+1)^2(k+2)...(k+p+1)}$ et on a :

$$\sum_{k=n+1}^{\infty} \frac{1}{(k+1)^2(k+2)...(k+p+1)} < \sum_{k=n+1}^{\infty} \frac{1}{k(k+1)(k+2)...(k+p+1)}$$

Et puisque :

$$\frac{p+1}{k(k+1)(k+2)...(k+p+1)} = \frac{1}{k(k+1)(k+2)...(k+p)} - \frac{1}{(k+1)(k+2)...(k+p+1)}$$

On a :

$$\sum_{k=n+1}^{\infty} u_n^{(p)} = \sum_{k=n+1}^{\infty} \frac{p!}{(k+1)^2(k+2)...(k+p+1)} < \frac{p!}{(p+1)(n+1)(n+2)...(n+p+1)}$$

Donc

$$\sum_{k=n+1}^{\infty} u_n^{(p)} < \frac{p!}{(p+1)(n+1)^{p+1}}$$

On vérifie ($\frac{\pi^2}{6} \simeq 1.64493406685$) :

$$1.63997129788 < \frac{\pi^2}{6} < 1.63997129788 + 1/201 = 1.64494642226$$

$$1.64492179293 < \frac{\pi^2}{6} < 1.64492179293 + 1/(2 * 201^2) = 1.64493416886$$

$$1.64493398626 < \frac{\pi^2}{6} < 1.64493398626 + 2/(3 * 201^3) = 1.64493406836$$

$$1.64493406596 < \frac{\pi^2}{6} < 1.64493406596 + 6/(4 * 201^4) = 1.64493406688$$

Le programme

On peut écrire un programme qui va demander le nombre d'accélérations pour calculer $\sum_{k=0}^{\infty} 1/(k+1)^2$

```
serie_sumacc(n,acc):={
local p,l,j,k,ls,sf,sg,gk,fact;
ls:=[];
//calcul sans acceleration
sf:=0.0;
for (k:=n;k>=0;k--) {
   sf:=sf+1/(k+1)^2;
}
ls:=[sf];
sf:=0.0;
fact:=1;
for (p:=1;p<=acc;p++){
  //calcul de 1+1/4+..+1/p^2, le terme a rajouter
  sf:=sf+evalf(1/p^2);
  //calcul de p!
  fact:=fact*(p);
  //calcul de sg, somme(de 0 a n) de la serie acceleree p fois
```

```
  sg:=0.0;
  for (k:=0;k<=n;k++) {
    gk:=1/(k+1)^2;
    //calcul du k-ieme terme gk de la serie acceleree p fois (sans p!)
    for (j:=1;j<=p;j++) {
      gk:=evalf(gk/(k+j+1));
    }
    sg:=sg+gk;
  }
  ls:=concat(ls,sf+fact*sg);
}
return(ls);
}
```

13.3 Accélération de convergence de Stirling

13.4 L'énoncé : Problème ESSEC 2001

On étudie dans ce problème la suite (S_n) définie pour $n \geq 1$ par :

$$S_n = 1 + \frac{1}{4} + \frac{1}{9} + ... + \frac{1}{n^2} = \sum_{p=1}^{n} \frac{1}{p^2}$$

Dans la partie I, on détermine la limite S de la suite (S_n). Dans les parties II et III, on explicite la méthode permettant d'accélérer la convergence de (S_n) vers S.

PARTIE I

On considère, pour tout nombre entier $p \geq 0$, les deux intégrales suivantes :

$$I_p = \int_0^{\frac{\pi}{2}} \cos(t)^{2p} dt, \;\; J_p = \int_0^{\frac{\pi}{2}} t^2 \cos(t)^{2p} dt$$

1. Convergence de la suite $\dfrac{J_p}{I_p}$

   (a) Établir l'inégalité suivante pour tout nombre réel t tel que $0 \leq t \leq \frac{\pi}{2}$:

   $$t \leq \frac{\pi}{2} \sin(t)$$

   (b) Établir l'inégalité suivante pour tout nombre entier $p \geq 0$

   $$0 \leq J_p \leq \frac{\pi^2}{4}(I_p - I_{p+1})$$

   :

   (c) Exprimer I_{p+1} en fonction de I_p en intégrant par parties l'intégrale I_{p+1} (on pourra poser $v'(t) = \cos(t)$ et $u(t) = \cos(t)^{2p+1}$ dans l'intégration par parties).

(d) Déduire des résultats précédents que $\frac{J_p}{I_p}$ tend vers 0 quand p tend vers plus l'infini.

2. Convergence et limite de la suite (S_n).

(a) Exprimer I_p en fonction de J_p et J_{p+1}, en intégrant deux fois par parties l'intégrale I_p ($p \geq 1$).

(b) En déduire la relation suivante pour $p \geq 1$:

$$\frac{J_{p-1}}{I_{p-1}} - \frac{J_p}{I_p} = \frac{1}{2p^2}$$

(c) Calculer J_0 et I_0 , puis déterminer la limite S de la suite (S_n).

PARTIE II

On accélère la convergence de (S_n) par une méthode due à Stirling.

On désigne par :

— E l'espace vectoriel des fonctions continues de]0 ; +1[dans $\mathbb{R}$ et de limite nulle en $+\infty$,

— f_k la fonction de E définie pour tout nombre entier naturel k par : $f_0(x) = \frac{1}{x}$ et pour $k \geq 1$, $f_k(x) = \frac{1}{x(x+1)(x+2)...(x+k)}$

— Δ l'application associant à toute fonction f de E, la fonction$\Delta(f)$ définie par : $\Delta(f)(x) = f(x+1) - f(x)$

1. Sommation de séries télescopiques

(a) Établir que Δ est un endomorphisme de l'espace vectoriel E.

(b) Exprimer $\Delta(f_{k-1})$ en fonction de k et de f_k pour $k \geq 1$.

(c) Établir pour tout nombre entier naturel $k \geq 1$ la convergence de la série $\sum_{p=1}^{+\infty} f_k(p)$ et vérifier, pour tout nombre entier naturel n, que :

$$\sum_{p=n+1}^{+\infty} f_k(p) = \frac{1}{k(n+1)(n+2)...(n+k)}$$

2. Accélération de la convergence de (S_n)

(a) Établir la relation suivante pour $p \geq 1$ et $q \geq 1$

$$\frac{1}{p^2} - \sum_{k=1}^{q} (k-1)! f_k(p) = \frac{q!}{p} f_q(p)$$

(b) En déduire l'inégalité suivante pour $n \geq 1$ et $q \geq 1$

$$0 \leq \sum_{p=n+1}^{+\infty} \frac{1}{p^2} - \sum_{k=1}^{q} \frac{(k-1)!}{k(n+1)...(n+k)} \leq \frac{(q-1)!}{(n+1)^2(n+2)..(n+q)}$$

(c) En déduire l'inégalité suivante pour $n \geq 1$ et $q \geq 1$

$$0 \leq \sum_{p=n+1}^{+\infty} \frac{1}{p^2} - \sum_{k=1}^{q} \frac{(k-1)!}{k(n+1)...(n+k)} \leq \frac{(q-1)!}{(n+1)^2(n+2)..(n+q)}$$

(d) En déduire, l'entier $q \geq 1$ étant fixé, une suite (S'_n) de nombres rationnels telle que :

$$0 \leq \frac{\pi^2}{6} - S'_n \leq \frac{(q-1)!}{(n+1)^2(n+2)..(n+q)}$$

Expliciter S'_n et l'inégalité précédente lorsque $q = 2$.

PARTIE III

1. Montrer que :
$\frac{1}{n+1} \leq \sum_{k=n+1}^{+\infty} \frac{1}{k^2} \leq \frac{1}{n}$
En déduire une valeur approchée de $\frac{\pi^2}{6}$ à 10^{-3} près.
2. On souhaite accélérer la convergence des sommes partielles. Montrer que :

$$0 \leq \left(\sum_{j=1}^{n} \frac{1}{j^2} + \frac{1}{n} \right) - \frac{\pi^2}{6} \leq \frac{1}{n(n+1)}$$

3. Calculer $\frac{1}{n} + \sum_{j=1}^{n} \frac{1}{j^2}$ pour $n = 10, n = 100$ et $n = 1000$ et déterminer le nombre de décimales exactes pour ces 3 valeurs de n, justifier ce nombre de décimales pour $n = 10$, $n = 100$ et $n = 1000$.
4. Montrer que pour $k \geq 2$ on a :

$$\frac{1}{k} - \frac{1}{k+1} + \frac{1}{2k(k+1)} - \frac{1}{2(k+1)(k+2)} < \frac{1}{k^2} < \frac{1}{k} - \frac{1}{k+1} + \frac{1}{2(k-1)k} - \frac{1}{2k(k+1)}$$

En déduire que :
$w_n = u_n + 1/(n+1) + 1/2/(n+1)/(n+2)$ converge vers $\pi^2/6$ et que l'on a :
$0 < \pi^2/6 - w_n < 1/n^3$

13.5 La solution

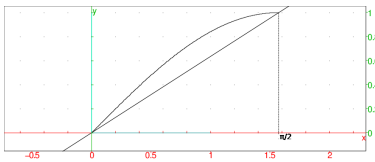
PARTIE I

$$\text{Pour } p \geq 0,\ I_p = \int_0^{\frac{\pi}{2}} \cos(t)^{2p} dt,\ \ J_p = \int_0^{\frac{\pi}{2}} t^2 \cos(t)^{2p} dt$$

On remarque que pour $p \geq 0$ on a $I_p > 0$ et $J_p > 0$

1. Étude de la convergence de la suite $\dfrac{J_p}{I_p}$

(a) Pour $0 \leq t \leq \frac{\pi}{2}$ montrons que $t \leq \frac{\pi}{2} \sin(t)$.
Il suffit de faire un dessin et de justifier ce que l'on voit en disant que la fonction sinus est concave :



sin est concave donc si $0 \leq t \leq \frac{\pi}{2}$, le graphe de $y = \sin(t)$ se trouve au dessus du segment $[0,\pi/2 + i]$ donc : $2t/\pi \leq \sin(t)$ soit $t \leq \frac{\pi}{2} \sin(t)$.

(b) Pour $p \geq 0$, montrons que $0 \leq J_p \leq \frac{\pi^2}{4}(I_p - I_{p+1})$.
On a :
$I_p - I_{p+1} = \int_0^{\frac{\pi}{2}} \cos(t)^{2p}(1-\cos(t)^2)dt = \int_0^{\frac{\pi}{2}} \cos(t)^{2p}\sin(t)^2 dt$
D'après la question précédente on a $t^2 \leq \frac{\pi^2}{4}\sin(t)^2$ donc :
$J_p = \int_0^{\frac{\pi}{2}} t^2\cos(t)^{2p}dt \leq \frac{\pi^2}{4}\int_0^{\frac{\pi}{2}} \cos(t)^{2p}\sin(t)^2 dt = \frac{\pi^2}{4}(I_p - I_{p+1})$

(c) Calcul de I_{p+1} en fonction de I_p.
On intégre par parties $I_{p+1}+$ en posant :
$v'(t) = \cos(t)$ et $u(t) = \cos(t)^{2p+1}$ donc
$v(t) = \sin(t)$ et $u'(t) = -(2p+1)\cos(t)^{2p}\sin(t)$ et
$I_{p+1} = [\cos(t)^{2p+1}\sin(t)]_{t=0}^{\frac{\pi}{2}} + (2p+1)\int_0^{\frac{\pi}{2}} \cos(t)^{2p}\sin(t)^2 dt$ donc
$I_{p+1} = (2p+1)(I_p - I_{p+1})$
Donc
$$I_{p+1} = \frac{2p+1}{2p+2} I_p$$

(d) Montrons que $\frac{J_p}{I_p}$ tend vers 0 quand p tend vers $+\infty$.
D'après ce qui précéde, on a $(I_p - I_{p+1}) = I_p(1 - \frac{2p+1}{2p+2}) = \frac{I_p}{2p+2}$, or d'après 1b $0 \leq J_p \leq \frac{\pi^2}{4}(I_p - I_{p+1})$ donc :
$0 < J_p \leq \frac{\pi^2}{4}\frac{I_p}{2p+2}$ donc puisque $I_p > 0$:
$\frac{J_p}{I_p} \leq \frac{\pi^2}{4}\frac{1}{2p+2}$
Comme pour $p \geq 0$ on a $I_p > 0$ et $J_p > 0$ et que $\frac{1}{2p+2}$ tend vers 0 quand p tend vers $+\infty$ on en déduit que :
$\frac{J_p}{I_p}$ tend vers 0 quand p tend vers $+\infty$

2. É tude de la convergence et limite de la suite (S_n).

(a) Calcul de $I_p = \int_0^{\frac{\pi}{2}} \cos(t)^{2p}dt$ en fonction de J_p et J_{p+1}.
On intégre deux fois par parties I_p ($p \geq 1$).
On pose : $u(t) = \cos(t)^{2p}$ et $v'(t) = 1$ donc
$u'(t) = -2p\cos(t)^{2p-1}\sin(t)$ et $v(t) = t$
On obtient :
$I_p = 0 + 2p\int_0^{\frac{\pi}{2}} t\cos(t)^{2p-1}\sin(t)dt = 2p\int_0^{\frac{\pi}{2}} t\sin(t)\cos(t)^{2p-1}dt$
On pose : $u(t) = \sin(t)\cos(t)^{2p-1}$ et $v'(t) = t$ donc
$u'(t) = \cos(t)^{2p} - (2p-1)\cos(t)^{2p-2}\sin^2(t)$ et $v(t) = t^2/2$
On obtient :
$I_p = 0 - p\int_0^{\frac{\pi}{2}} t^2\cos(t)^{2p}dt + p(2p-1)\int_0^{\frac{\pi}{2}} t^2\cos(t)^{2p-2}\sin^2(t)dt =$
$-pJ_p + p(2p-1)(J_{p-1} - J_p) = p(2p-1)J_{p-1} - 2p^2J_p$
Donc
$$I_p = p(2p-1)J_{p-1} - 2p^2J_p$$

(b) Pour $p \geq 1$, montrons que $\frac{J_{p-1}}{I_{p-1}} - \frac{J_p}{I_p} = \frac{1}{2p^2}$.
D'après 2b on a $I_p = p(2p-1)J_{p-1} - 2p^2J_p$ donc
$1 = p(2p-1)\frac{J_{p-1}}{I_p} - 2p^2\frac{J_p}{I_p}$
D'après 1c $I_{p+1} = \frac{2p+1}{2p+2}I_p$ donc $I_p = \frac{2p-1}{2p}I_{p-1}$

On obtient :
$p(2p-1)\frac{J_{p-1}}{I_p} = p(2p-1)\frac{2pJ_{p-1}}{(2p-1)I_{p-1}} = 2p^2\frac{J_{p-1}}{I_{p-1}}$
$1 = 2p^2\frac{J_{p-1}}{I_{p-1}} - 2p^2\frac{J_p}{I_p}$
On a donc :

$$\frac{J_{p-1}}{I_{p-1}} - \frac{J_p}{I_p} = \frac{1}{2p^2}$$

(c) Calculs de J_0, I_0 et de la limite S de la suite (S_n).
On a :
$I_0 = \int_0^{\frac{\pi}{2}} dt = \frac{\pi}{2}$
$J_0 = \int_0^{\frac{\pi}{2}} t^2 dt = \frac{\pi^3}{24}$
On a :
$S_n = \sum_{p=1}^{n} \frac{1}{p^2} = 2\sum_{p=1}^{n}(\frac{J_{p-1}}{I_{p-1}} - \frac{J_p}{I_p})$
$S_n = 2(\frac{J_0}{I_0} - \frac{J_n}{I_n}) = \frac{\pi^2}{6} - 2\frac{J_n}{I_n}$
Quand n tend vers $+\infty$, $\frac{J_n}{I_n}$ tend vers 0 donc S_n tend vers $\frac{\pi^2}{6}$.
Donc

$$S = \frac{\pi^2}{6}$$

PARTIE II
On accélère ici la convergence de la suite (S_n) vers sa limite S par une méthode due à Stirling.

1. Sommation de séries télescopiques

(a) Δ est un endomorphisme de E.
E est un espace vectoriel sur $\mathbb{R}$:
si $f \in E$, $g \in E$, $a \in \mathbb{R}$, $b \in \mathbb{R}$ alors la fonction $af + bg$ est continue de]0 ; +1[dans $\mathbb{R}$ et a une limite nulle en $+\infty$ donc $af + bg \in E$.
Si $f \in E$ alors la foncion $h = \Delta(f)$ est continue et a une limite nulle en $+\infty$ puisque $h(x) = f(x+1) - f(x)$ avec f est continue et f a une limite nulle en $+\infty$. Donc $\Delta(f) \in E$. De plus :
$\Delta(af + bg)(x) = (af + bg)(x+1) - (af + bg)(x) =$
$af(x+1) + bg(x+1) - af(x) - bg(x) = a\Delta(f)(x) + b\Delta(g)(x)$
Donc Δ est un endomorphisme de E.

(b) Montrons que si $f \in E$, $p \geq 1$ alors la série $\Delta(f)(p)$ converge
On a pour $f \in E$ et $p \geq 1$:
$\sum_{p=1}^{n} \Delta(f)(p) = \sum_{p=1}^{n}(f(p+1) - f(p)) = f(n+1) - f(1)$
Quand n tend vers $+\infty$ $f(n+1)$ tend vers 0 car $f \in E$
Donc la série $\Delta(f)(p)$ converge vers $-f(1)$
Donc

$$\sum_{p=1}^{+\infty} \Delta(f)(p) = -f(1)$$

Pour $n \in \mathbb{N}$, on a :
$\sum_{p=n+1}^{+\infty} \Delta(f)(p) = \sum_{p=1}^{+\infty} \Delta(f)(p) - \sum_{p=1}^{n} \Delta(f)(p) = -f(n+1)$
Donc

$$\sum_{p=n+1}^{+\infty} (\Delta(f)(p) = -f(n+1)$$

(c) Calcul de $\Delta(f_{k-1})$ en fonction de k et de f_k pour $k \geq 1$.
$\Delta(f_{k-1})(x) = f_{k-1}(x+1) - f_{k-1}(x) =$
$\frac{1}{(x+1)...(x+k)} - \frac{1}{x(x+1)...(x+k-1)} = \frac{-k}{x(x+1)...(x+k)}$
Donc :
$f_{k-1}(x+1) - f_{k-1}(x) = -kf_k(x)$ donc

$$f_k(x) = \frac{f_{k-1})(x) - f_{k-1})(x+1)}{k}$$

(d) Pour $k \geq 1$, étude de la convergence de $\sum f_k(p)$
$\sum_{p=1}^{n} f_k(p) = \sum_{p=1}^{n} \frac{f_{k-1})(p) - f_{k-1})(p+1)}{k} = \frac{f_{k-1})(1) - f_{k-1})(n+1)}{k}$.
Quand n tend vers $+\infty$, $f_{k-1}(n+1) = \frac{1}{(n+1)...(n+k)}$ tend vers 0
Donc $\sum_{p=1}^{+\infty} f_k(p) = \frac{f_{k-1})(1)}{k} = \frac{1}{k!k}$.
On a donc tout nombre entier naturel n :
$\sum_{p=n+1}^{+\infty} f_k(p) = \sum_{p=1}^{+\infty} f_k(p) - \sum_{p=1}^{n} f_k(p)) = \frac{f_{k-1})(1)}{k} - \frac{f_{k-1})(1) - f_{k-1})(n+1)}{k} =$
$\frac{f_{k-1})(n+1)}{k} = \frac{1}{k(n+1)...(n+k)}$.
Donc :

$$\sum_{p=n+1}^{+\infty} f_k(p) = \frac{1}{k(n+1)(n+2)...(n+k)}$$

2. Accélération de la convergence de (S_n)

(a) Pour $p \geq 1$, $q \geq 1$, montrons que $\frac{1}{p^2} - \sum_{k=1}^{q}(k-1)!f_k(p) = \frac{q!}{p} f_q(p)$
Pour montrer cette égalité, on va faire une récurrence sur q :
L'égalité est vraie pour $q = 1$ en effet $f_1(p) = \frac{1}{p(p+1))}$ donc
$\frac{1}{p^2} - \sum_{k=1}^{1}(k-1)!f_k(p) = \frac{1}{p^2} - f_1(p) = \frac{1}{p^2(p+1))} = \frac{1!}{p} f_1(p)$
Supposons l'égalité vraie pour q et montrons qu'elle est vraie pour $q+1$.
On suppose que l'on a $\frac{1}{p^2} - \sum_{k=1}^{q}(k-1)!f_k(p) = \frac{q!}{p} f_q(p)$.
Calculons :
$\frac{1}{p^2} - \sum_{k=1}^{q+1}(k-1)!f_k(p) = \frac{1}{p^2} - \sum_{k=1}^{q}(k-1)!f_k(p) - q!f_{q+1}(p) =$

$\frac{q!}{p}f_q(p) + q!f_{q+1}(p)$
Or $f_{q+1}(p) = \dfrac{1}{p(p+1)(p+2)...(p+q+1)} = \dfrac{1}{p+q+1}f_q(p)$
Donc :
$\frac{1}{p^2} - \sum_{k=1}^{q+1}(k-1)!f_k(p) = (\frac{q!}{p} - \frac{q!}{p+q+1})f_q(p) =$
$\frac{q!(q+1)}{p(p+q+1)}f_q(p) = \frac{(q+1)!}{p}f_{q+1}(p)$
Donc l'égalité est vraie pour $q+1$.
Donc pour $p \geq 1$ et $q \geq 1$ on a :

$$\frac{1}{p^2} - \sum_{k=1}^{q}(k-1)!f_k(p) = \frac{q!}{p}f_q(p)$$

(b) Pour $n \geq 1$ et $q \geq 1$ montrons que
$0 \leq \sum_{p=n+1}^{+\infty}\frac{1}{p^2} - \sum_{k=1}^{q}\frac{(k-1)!}{k(n+1)...(n+k)} \leq \frac{(q-1)!}{(n+1)^2(n+2)..(n+q)}$
On peut sommer de $p = n+1$ à $+\infty$ l'égalite de la question précédente. En effet on a : $0 < \frac{q!}{p}f_q(p) < q!f_q(p)$ et la série à termes positifs u_p=$f_q(p)$ est convergente d'après 1d. Donc pour k fixé, les séries à termes positifs $(k$-$1)!f_k(p)$ et $\frac{k!}{p}f_k(p)$ sont convergentes et leur sommes sont positives.
Donc :
$0 < \sum_{p=n+1}^{+\infty}(\frac{1}{p^2} - \sum_{k=1}^{q}(k-1)!f_k(p)) = \sum_{p=n+1}^{+\infty}\frac{q!}{p}f_q(p) = A$

$$A = \sum_{p=n+1}^{+\infty}(\frac{1}{p^2} - \sum_{k=1}^{q}(k-1)!f_k(p)) = \sum_{p=n+1}^{+\infty}\frac{1}{p^2} - \sum_{p=n+1}^{+\infty}\sum_{k=1}^{q}(k-1)!f_k(p))$$

On peut intervertir les 2 sommes puisque la 2ième somme est finie :
$A = \sum_{p=n+1}^{+\infty}\frac{1}{p^2} - \sum_{k=1}^{q}(\sum_{p=n+1}^{+\infty}(k-1)!f_k(p))$
d'après 1d $\displaystyle\sum_{p=n+1}^{+\infty} f_k(p) = \frac{1}{k(n+1)(n+2)...(n+k)}$ donc

$$A = \sum_{p=n+1}^{+\infty}\frac{1}{p^2} - \sum_{k=1}^{q}(\frac{(k-1)!}{k(n+1)(n+2)...(n+k)}$$

On obtient donc :
$\displaystyle\sum_{p=n+1}^{+\infty}\frac{1}{p^2} - \sum_{k=1}^{q}\frac{(k-1)!}{k(n+1)...(n+k)} = \sum_{p=n+1}^{+\infty}\frac{q!}{p}f_q(p) = A$
Puisque $\displaystyle\sum_{p=n+1}^{+\infty}\frac{q!}{p}f_q(p) = q!\sum_{p=n+1}^{+\infty}\frac{1}{p}f_q(p)$ on majore cette somme en minorant p par $n+1$:
$\displaystyle\sum_{p=n+1}^{+\infty}\frac{q!}{p}f_q(p) \leq q!\sum_{p=n+1}^{+\infty}\frac{1}{n+1}f_q(p) = \frac{q!}{n+1}\sum_{p=n+1}^{+\infty}f_q(p)$

D'après 1d $\displaystyle\sum_{p=n+1}^{+\infty} f_q(p) = \frac{1}{q(n+1)(n+2)..(n+q)}$

Donc :

$\displaystyle\sum_{p=n+1}^{+\infty} \frac{q!}{n+1} f_q(p) = \frac{(q-1)!}{(n+1)^2(n+2)..(n+q)}$

on obtient bien :

$$A = \sum_{p=n+1}^{+\infty} \frac{1}{p^2} - \sum_{k=1}^{q} \frac{(k-1)!}{k(n+1)...(n+k)} \leq \frac{q!}{(n+1)^2(n+2)..(n+q)}$$

(c) l'entier $q \geq 1$ étant fixé, montrons qu'il existe $(S'_n(q))$ telle que :

$$0 \leq \frac{\pi^2}{6} - S'_n(q) \leq \frac{(q-1)!}{(n+1)^2(n+2)..(n+q)}$$

On pose :

$S'_n(q) = \sum_{p=1}^{n} \frac{1}{p^2} + \sum_{k=1}^{q} \frac{(k-1)!}{k(n+1)...(n+k)} = S_n + \sum_{k=1}^{q} \frac{(k-1)!}{k(n+1)...(n+k)}$.

On a donc ($S - S_n = \sum_{p=n+1}^{+\infty} \frac{1}{p^2}$) :

$\frac{\pi^2}{6} = S_n + (S - S_n) = S'_n(q) - \sum_{k=1}^{q} \frac{(k-1)!}{k(n+1)...(n+k)} + (S - S_n)$

Donc :

$$\frac{\pi^2}{6} - S'_n(q) = \sum_{p=n+1}^{+\infty} \frac{1}{p^2} - \sum_{k=1}^{q} \frac{(k-1)!}{k(n+1)...(n+k)} \leq \frac{(q-1)!}{(n+1)^2(n+2)..(n+q)}$$

(d) Calcul de $\pi^2/6$ par un programme.

`somme(n,q)` calcule $\sum_{k=1}^{q} \frac{(k-1)!}{k(n+1)...(n+k)}$

`reste(n,q)` calcule $\frac{(q-1)!}{(n+1)^2(n+2)..(n+q)}$

`sommepi2(n,q)` calcule $S = \pi^2/6$

vec `Xcas`, on tape :

```
sommep(n,q):={
 local k,s,a;
 a:=1/(n+1);
 s:=0;
 pour k de 1 jusque q faire
   s:=s+a;
   a:=a*(k^2)/(k+1)/(n+k+1);
 fpour;
 return s;
}:;

restep(n,q):={
 local k,r;
 r:=(q-1)!/(n+1);
 pour k de 1 jusque q faire
   r:=r/(n+k);
 fpour;
```

```
 return r;
}:;

sommepi2(n,q):={
 local j,S,r;
 S:=0;
 pour j de 1 jusque n faire
  S:=S+1/j^2;
 fpour;
 return [evalf(pi^2/6,[S,(S+sommep(n,q))],restep(n,q))];
}:;
```

On tape :
`sommepi2(20,2)`
On obtient (avec 14 chiffres significatifs) :
`[1.6449340668482,[1.596163243913,1.6448645426143],0.00010307153164296`
On tape :
`sommepi2(20,5)`
On obtient (avec 14 chiffres significatifs) :
`[1.6449340668482,[1.596163243913,1.6449339164772],1.7925483763993e-07`
On tape :
`sommepi2(50,2)`
On obtient (avec 14 chiffres significatifs) :
`[1.6449340668482,[1.6251327336215,1.644929113712],7.393606009523e-06]`
On tape :
`sommepi2(50,5)`
On obtient (avec 14 chiffres significatifs) :
`[1.6449340668482,[1.6251327336215,1.6449340659062],1.1272888903408e-0`

PARTIE III

1. On a :
   $\int_k^{k+1} \frac{dt}{t^2} \leq \frac{1}{k^2} \leq \int_{k-1}^{k} \frac{dt}{t^2}$
   donc
   $\frac{1}{k} - \frac{1}{k+1} \leq \frac{1}{k^2} \leq \frac{1}{k-1} - \frac{1}{k}$
   On a :
   $\sum_{k=n+1}^{+\infty} (\frac{1}{k} - \frac{1}{k+1}) = \frac{1}{n+1}$ et $\sum_{k=n+1}^{+\infty} (\frac{1}{k-1} - \frac{1}{k}) = \frac{1}{n}$
   Donc :
   $\frac{1}{n+1} \leq \sum_{k=n+1}^{+\infty} \frac{1}{k^2} \leq \frac{1}{n}$
2. Puisque la fonction $\frac{1}{x^2}$ est décroissante on a :

$$\int_{n+1}^{\infty} \frac{1}{x^2}\, dx \leq \sum_{j=n+1}^{\infty} \frac{1}{j^2} \leq \int_{n}^{\infty} \frac{1}{x^2}\, dx$$

3. Donc :
   $\frac{1}{n+1} \leq \sum_{k=n+1}^{+\infty} \frac{1}{k^2} \leq \frac{1}{n}$
   Ainsi :
   $\sum_{k=1}^{n} \frac{1}{k^2}$ fournit une valeur approchée de $\frac{\pi^2}{6}$ avec une erreur équivalente à $\frac{1}{n}$.

4. Montrons que :
   $\sum_{k=1}^{n} \frac{1}{k^2} + \frac{1}{n+1}$ fournit une valeur approchée de $\frac{\pi^2}{6}$ avec une erreur équivalente à $\frac{1}{2n^2}$ lorsque $n- > +\infty$.
   On a :
   $\frac{\pi^2}{6} - (\sum_{k=1}^{n} \frac{1}{k^2} + \frac{1}{n+1}) = \sum_{k=n+1}^{+\infty} \frac{1}{k^2} - \frac{1}{n+1}$.
   $\sum_{k=n+1}^{+\infty} \frac{1}{k^2} - \frac{1}{n+1} = \sum_{k=n+1}^{+\infty} \frac{1}{k^2} - (\frac{1}{k} - \frac{1}{k+1}) = \sum_{k=n+1}^{+\infty} \frac{1}{k^2(k+1)}$
   La fonction $\frac{1}{t^2(t+1)}$ est décroissante donc :
   $\int_k^{k+1} \frac{1}{(t+1)^3} dt \leq \int_k^{k+1} \frac{1}{t^2(t+1)} dt \leq \frac{1}{k^2(k+1)} \leq \int_{k-1}^{k} \frac{1}{t^2(t+1)} dt \leq \int_{k-1}^{k} \frac{1}{t^3} dt$:
   Donc :
   $\frac{1}{2(k+1)^2} - \frac{1}{2(k+2)^2} = \int_k^{k+1} \frac{1}{(t+1)^3} dt \leq \frac{1}{k^2(k+1)} \leq \int_{k-1}^{k} \frac{1}{t^3} dt = \frac{1}{2(k-1)^2} - \frac{1}{2k^2}$:
   Donc :
   $\frac{1}{2(n+2)^2} \leq \sum_{k=n+1}^{+\infty} \frac{1}{k^2(k+1)} \leq \frac{1}{2n^2}$
   Donc lorsque $n- > +\infty$:
   $\sum_{k=n+1}^{+\infty} \frac{1}{k^2} - \frac{1}{n+1} \sim \frac{1}{2n^2}$
   Donc :
   $\frac{\pi^2}{6} = \sum_{k=1}^{n} (\frac{1}{k^2} + \frac{1}{n+1}) + \frac{1}{2n^2}(1 + \epsilon(n))$
   avec $\epsilon(n)- > 0$ lorsque $n- > +\infty$.

5. Montrons que :
   $\sum_{k=1}^{n} \frac{1}{k^2} + \frac{1}{n+1} + \frac{1}{2(n+1)(n+2)}$ fournit une valeur approchée de $\frac{\pi^2}{6}$ avec une erreur équivalente à $\frac{2}{3n^3}$ lorsque $n- > +\infty$.
   $\sum_{k=n+1}^{+\infty} \frac{1}{k^2} - \frac{1}{n+1} - \frac{1}{2(n+1)(n+2)} =$

$\sum_{k=n+1}^{+\infty}(\frac{1}{k^2}-(\frac{1}{k}-\frac{1}{k+1})-(\frac{1}{2k(k+1)}-\frac{1}{2(k+1)(k+2)}))=\sum_{k=n+1}^{+\infty}\frac{2}{k^2(k+1)(k+2)}$
La fonction $\frac{1}{t^2(t+1)(t+2)}$ est décroissante donc :
$\int_k^{k+1}\frac{2}{(t+2)^4}dt\leq\int_k^{k+1}\frac{2}{t^2(t+1)(t+2)}dt\leq\frac{1}{k^2(k+1)}\leq\int_{k-1}^{k}\frac{2}{t^2(t+1)(t+2)}dt\leq$
$\int_{k-1}^{k}\frac{2}{t^4)}dt$:
Donc $\frac{3}{2(k+2)^3}-\frac{3}{2(k+3)^3}=\int_k^{k+1}\frac{2}{(t+2)^4}dt\leq\frac{2}{k^2(k+1)(k+2)}\leq$
$\int_{k-1}^{k}\frac{2}{t^4}dt=\frac{3}{2(k-1)^3}-\frac{3}{2k^3}$:
Donc :
$\frac{3}{2(n+3)^3}\leq\sum_{k=n+1}^{+\infty}\frac{1}{k^2(k+1)}\leq\frac{3}{2n^3}$
Donc lorsque $n->+\infty$:
$\sum_{k=n+1}^{+\infty}\frac{1}{k^2}-\frac{1}{n+1}-\frac{1}{2(n+1)(n+2)}\sim\frac{3}{2n^3}$
Donc :
$\frac{\pi^2}{6}=\sum_{k=1}^{n}\frac{1}{k^2}+(\frac{1}{n+1}+\frac{1}{2(n+1)(n+2)})+\frac{3}{2n^3}(1+\epsilon(n))$
avec $\epsilon(n)->0$ lorsque $n->+\infty$.
On tape :
```
normal(1/k^2-1/k+1/(k+1)-1/2/k/(k+1)+1/2/(k+1)/(k+2))
```
On obtient :
```
2/(k^4+3*k^3+2*k^2)
```
On tape :
```
normal(1/(k^2)-1/k+1/(k+1)-1/2/k/(k-1)+1/2/(k+1)/k)
```
On obtient :
```
-1/(k^4-k^2)
```

On remarque que ces accélérations correspondent à l'accélération de Stirling lorsque $q=1$ et pour $q=2$.

13.6 Méthodes d'accélération de convergence des séries alternées

13.6.1 Un exemple d'accélération de convergence des séries alternées

Un premier exemple

On suppose que $u_k=(-1)^k f(k)$ avec $f(k)$ tend vers zéro quand k tend vers $+\infty$ et f décroissante de $\mathbb{R}^+$ dans $\mathbb{R}^+$.
On pose :
$g(x)=\frac{1}{2}(f(x)-f(x+1))$ donc
$v_k=(-1)^k\frac{f(k)-f(k+1)}{2}=\frac{u_k+u_{k+1}}{2}=(-1)^k g(k)$

On a :
$\sum_{k=0}^{n} v_k = \frac{1}{2}(\sum_{k=0}^{n} u_k + \sum_{k=0}^{n} u_{k+1})$
donc,
$\sum_{k=0}^{n} v_k = \frac{1}{2}(\sum_{k=0}^{n} u_k + \sum_{k=1}^{n+1} u_k)$
donc,
$\sum_{k=0}^{n} v_k = \frac{u_0}{2} + \frac{u_{n+1}}{2} + \sum_{k=0}^{n} u_k$
Puisque $f(k)$ tend vers zéro quand k tend vers $+\infty$, $g(k) = \frac{1}{2}(f(k) - f(k+1))$ tend aussi vers zéro quand k tend vers $+\infty$.
Si la fonction f est convexe ($f''(x) > 0$), la série $\sum_{k=0}^{\infty} v_k$ vérifie aussi le théorème des séries alternées.
En effet, pour $x > 0$ on a :
$g(x) = \frac{1}{2}(f(x) - f(x+1)) \geq 0$ puisque f décroissante sur $\mathbb{R}^+$
$g'(x) = \frac{1}{2}(f'(x) - f'(x+1)) < 0$ puisque $f''(x) > 0$, f' est négative et croissante sur $\mathbb{R}^+$
donc g est décroissante de $\mathbb{R}^+$ dans $\mathbb{R}^+$ et $g(k)$ tend vers zéro quand k tend vers $+\infty$.
Conclusion : La série $\sum_{k=0}^{\infty} v_k$ est une série alternée de somme $S + \frac{u_0}{2}$.
Si de plus, $f'(x)/f(x)$ tend vers zéro quand x tend vers l'infini, la série $\sum_{k=0}^{\infty} v_k$ converge plus rapidement que $\sum_{k=0}^{\infty} u_k$, puisque il existe c, $x < c < x+1$ d'après le th des accroissements finis tel que :
$0 < g(x) = \frac{1}{2}(f(x) - f(x+1)) = \frac{-1}{2} f'(c)$
on a donc, puisque f' est négative et croissante :
$0 < g(x) < \frac{-1}{2} f'(x) = o(f(x))$.

Un exercice

Utiliser cette méthode pour calculer numériquement : $\sum_{k=0}^{\infty} \frac{(-1)^k}{k+1}$.

Toutes les dérivées de $f(x) = 1/(x+1)$ ont un signe constant sur $[0; +\infty[$ et tendent vers zéro à l'infini, ces dérivées sont donc monotones et on peut donc faire plusieurs accélérations successives.
On va faire "à la main " trois accélérations successives.
On pose :
$u_k = \frac{(-1)^k}{(k+1)}$

— 1-ière accélération :
$v_k = (-1)^k(\frac{1}{2(k+1)} - \frac{1}{2(k+2)})$, et donc
$v_k = (-1)^k(\frac{1}{2(k+1)(k+2)})$
$\sum_{k=0}^{\infty} u_k = \frac{1}{2} + \sum_{k=0}^{\infty} v_k$

— 2-ième accélération :
$w_k = (-1)^k(\frac{1}{4(k+1)(k+2)} - \frac{1}{4(k+2)(k+3)})$, et donc

$w_k = (-1)^k(\frac{1}{2(k+1)(k+2)(k+3)})$
et comme $\frac{v_0}{2} = \frac{1}{8}$ on a :
$\sum_{k=0}^{\infty} u_k = \frac{1}{2} + \frac{1}{8} + \sum_{k=0}^{\infty} w_k$

— 3-ième accélération :
$t_k = (-1)^k(\frac{1}{4(k+1)(k+2)(k+3)} - \frac{1}{4(k+2)(k+3)(k+4)})$, et donc
$t_k = (-1)^k(\frac{3}{4(k+1)(k+2)(k+3)(k+4)})$
et comme $\frac{w_0}{2} = \frac{1}{24}$ on a :
$\sum_{k=0}^{\infty} u_k = \frac{1}{2} + \frac{1}{8} + \frac{1}{24} + \sum_{k=0}^{\infty} t_k$

On tape :
```
u(k):=(-1)^k/(k+1)
```
On tape :
```
v(k):=(-1)^k/(2*(k+1)*(k+2))
```
On tape :
```
w(k):=(-1)^k/(2*(k+1)*(k+2)*(k+3))
```
On tape :
```
t(k):=(-1)^k*3/(4*(k+1)*(k+2)*(k+3)*(k+4))
```
On compare $\ln(2)$ et les valeurs obtenues pour $n = 200$, car on sait que :
$S = \sum_{k=0}^{\infty}(-1)^k\frac{1}{(k+1)} = \ln(2) \simeq 0.69314718056$
On tape :
```
serie_sum(u,0,200)
```
On obtient S à $5 * 10^{-3}$ prés (2 décimales exactes) :
```
0.69562855486
```
On tape :
```
1/2+serie_sum(v,0,200)
```
On obtient S à $1.23 * 10^{-5}$ prés (4 décimales exactes) :
```
0.693153307335
```
On tape :
```
1/2+1/8+serie_sum(w,0,200)
```
On obtient S à $6.1 * 10^{-8}$ prés (8 décimales exactes) :
```
0.693147210666
```
On tape :
```
1/2+1/8+1/24+serie_sum(t,0,200)
```
On obtient S à $4.6 * 10^{-10}$ prés (10 décimales exactes) :
```
0.693147180781
```
Les erreurs
Le reste d'une série alternée est du signe de son premier terme et la valeur absolue du reste est inférieure à la valeur absolue de son premier terme :
$|\sum_{k=n+1}^{\infty}(-1)^k\frac{1}{(k+1)}| < \frac{1}{(n+2)}$
Au bout de la p-ième accéleration on calcule la somme de :

$u_k^{(p)} = (-1)^k \frac{p!}{2^p(k+1)(k+2)...(k+p+1)}$ et on a :

$$|\sum_{k=n+1}^{\infty} \frac{(-1)^k p!}{2^p(k+1)...(k+p+1)}| < \frac{p!}{2^p(n+2)...(n+p+2)} < \frac{p!}{2^p(n+2)^{p+1}}$$

On vérifie ($\ln(2) \simeq 0.69314718055995$) :
$0.69562855486 < \ln(2) < 0.69562855486 + 1/202 = 0.70057904991$
$0.693153307335 < \ln(2) < 0.693153307335 + 1/(2 * 202^2) = 0.693165561036$
$0.693147210666 < \ln(2) < 0.693147210666 + 2/(4 * 202^3) = 0.693147271328$
$0.693147180781 < \ln(2) < 0.693147180781 + 6/(8 * 202^4) = 0.693147181231.$

Le programme

On peut écrire un programme qui va demander le nombre p d'accélérations.
Si $u_k^{(p)}$ désigne le k-ième terme de la série accélérée p fois, on a :
$\sum_{k=0}^{\infty}(-1)^k/(k+1) = \sum_{k=0}^{p} \frac{u_0^{(k-1)}}{2} + \sum_{k=0}^{\infty} u_k^{(p)}$
avec
$u_k^{(p)} = \frac{(-1)^k p!}{2^p(k+1)...(k+p+1)}$
On choisit de multiplier seulement à la fin par $\frac{p!}{2^p}$ et de ne calculer que la somme des n premiers termes :

$$\sum_{k=0}^{n} \frac{(-1)^k}{(k+1)...(k+p+1)}$$

On met cette somme dans la variable `sg`, pour cela on calcule $\frac{(-1)^k}{(k+1)...(k+p+1)}$ que l'on met dans la variable `gk` :
au début `sg=0` et `gk=`$\frac{1}{(p+1)!}$ (c'est la valeur pour $k = 0$)
puis, on ajoute `gk` à la somme `sg`, ensuite on calcule $\frac{(-1)^1 1!}{(p+2)!}$ que l'on met dans `gk` (c'est la valeur pour $k = 1$) etc...
La variable `sf` sert au début à calculer $\sum_{k=0}^{n}(-1)^k/(k+1)$ puis,
`sf` sert à calculer la somme à rajouter $\sum_{k=0}^{p} \frac{u_0^{(k-1)}}{2}$ (qui vaut $1/2+1/8+1/24$ pour $p = 3$ accélérations).
Dans le programme, on utilise la variable `fact` pour calculer `(p+1)!` et la variable `fact2` pour calculer `p!/2^p`.
On écrit :

```
seriealt_sumacc(n,acc):={
local l,j,k,ls,sf,sg,gk,fact,fact2,alt,t0,p;
//calcul sans acceleration
sf:=0.0;
```

```
alt:=1;
for (k:=n;k>=0;k--) {
sf:=sf+alt/(k+1);
alt:=-alt;
}
if (alt==1) {
ls:=[-sf];}
else {
ls:=[sf];
}
t0:=0.5;
// sf maintenant est la somme a rajouter
sf:=0.0;
fact:=1;fact2:=1;
for (p:=1;p<=acc;p++){
  sf:=sf+fact2*t0;
  //calcul de p+1! et de p!/2^p
  fact:=fact*(p+1);
  fact2:=fact2*p/2;
//sg, somme(de k=0 a n) de la serie gk acceleree p fois
  sg:=0.0;
//terme d'indice 0 (ds gk) de la serie acceleree p fois
//(sans p!/2^p=fact2)
  gk:=1/fact;
//on conserve gk/2 dans t0 car il faut rajouter t0
//au prochain sf
  t0:=gk/2;
  sg:=sg+gk;
  alt:=-1;
  for (k:=1;k<=n;k++) {
     gk:=1/(k+1);
//terme d'indice k (ds gk) de la serie acceleree p fois
//(sans p!/2^p=fact2)
     for (j:=1;j<=p;j++) {
       gk:=evalf(gk/(k+j+1));
     }
     sg:=sg+alt*gk;
     alt:=-alt;
  }
ls:=concat(ls,sf+fact2*sg);
}
return(ls);
}
```

On met ce programme dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`), puis on le teste et le valide avec `OK` et on tape dans une ligne de commandes :

`seriealt_sumacc(200,3)`

On obtient :

```
[0.69562855486,0.693153307335,0.693147210666,0.693147180781]
```
On tape :
```
seriealt_sumacc(100,4)
```
On obtient :
```
[0.698073169409,0.693171208625,0.693147412699,
0.693147183892,0.693147180623]
```

13.6.2 La transformation d'Euler pour les series alternées

La transformation d'Euler

On cherche une approximation de :
$\sum_{n=0}^{\infty}(-1)^n u(n)$=`sum((-1)^n*u(n),n,0,infinity)` lorsque la suite `u(n)` converge vers 0 en décroissant.
On pose : `Delta(u))(n)=u(n+1)-u(n)` et
`delta(u,p,n)=(Delta@@p(u))(n)`
On a :
```
delta(u,2,n)=u(n+2)-2*u(n+1)+u(n)
delta(u,3,n)=u(n+3)-3*u(n+2)+3*u(n+1)-u(n)
delta(u,p,N)=u(n+p)-comb(p,1)*u(n+p-1)+comb(p,2)*u(n+p-2)+
                ....+(-1)^p*u(n)
```
c'est à dire :
```
delta(u,p,n)=sum((-1)^(p-j)*comb(p,j)*u(n+j),j,0,p)
```
La transformation d'Euler consiste à écrire :
```
sum((-1)^n*u(n),n,N,infinity)
```
sous la forme :
```
(-1)^N*sum((-1)^p*delta(u,p,N)/2^(p+1),p,0,infinity)
```
Pour prouver cette égalité il suffit de développer la dernière expression et de chercher le coefficient de `u(N+k)` dans la somme :
$\sum_{p=0}^{\infty}, (-1)^p * \frac{delta(u,p,N)}{2^{p+1}}$
Le coefficient de `u(N+k)` est :
```
s(k)=(-1)^k*sum(comb(k+p,p)/2^(k+p+1),p,0,infinity)
```
et cette somme vaut `(-1)^k` quelque soit `k` entier.
En effet par récurrence :
pour `k=0`, `comb(k+p,p)=1` et
```
sum(1/2^(p+1),p,0,infinity)=1/2+1/4+...1/2^n+...=1
```
On a de plus :
- pour `p=0`, `comb(k+p,p)=comb(k+1+p,p)=1`
- pour `p>0`, `comb(k+p,p)=comb(k+1+p,p)-comb(k+1+p-1,p-1)`

donc
```
s(k)=(-1)^k*sum(comb(k+1+p,p)/2^(k+p+1),p,0,infinity)-
(-1)^k*sum(comb(k+1+p-1,p-1)/2^(k+1+p-1+1),p,1,infinity)=
-2*s(k+1)-
(-1)^k*sum(comb(k+1+p,p)/2^(k+1+p+1),p,0,infinity)=
-2*s(k+1)+s(k+1)=-s(k+1).
```
donc si `s(k)=(-1)^k` alors `s(k+1)=(-1)^(k+1)`.
La transformation d'Euler permet une accélération de convergence car la série :

`sum((-1)^p*delta(u,p,N)/2^(p+1),p,0,infinity)`
converge plus rapidement.

Le programme

On définit, tout d'abord, la fonction `delta` :

```
delta(u,p,n):={
  local val,k,s;
  val:=0;
  s:=1;
  for (k:=p;k>=0;k--) {
  val:=val+comb(p,k)*u(n+k)*s;
  s:=s*-1;
  }
  return val;
};
```

On écrit la transforpmation d'Euler :
`trans_euler(u,N,M)` qui approche
`sum((-1)^n*u(n),n,0,infinity)` et vaut :
`sum((-1)^n*u(n),n,0,N-1)+`
`(-1)^N*sum((-1/2)^p*delta(u,p,N)/2,p,0,M).`

```
trans_euler(u,N,M):={
  local S,T,k,s;
  S:=0;
  s:=1;
  for (k:=0;k<N;k++) {
  S:=S+u(k)*s;
  s:=s*-1;
  }
  T:=0;
  s:=s*1/2;
for (k:=0;k<=M;k++) {
  T:=T+delta(u,k,N)*s;
  s:=s*-1/2;
};
  return evalf(normal(S+T));
};
```

Par exemple pour $u(n) = 1/(n+1)$ avec 20 digits, on tape :
`u(n):=1/(n+1);`
`DIGITS:=20;`
`trans_euler(u,10,20);`
On obtient :
`0.69314718055994505611`
`trans_euler(u,9,21);`
On obtient :

```
0.693147180559945594072
```
On remarque que l'on a 16 decimales exactes car on a :
```
evalf(ln(2))=0.693147180559945309415
```

13.6.3 Autre façon de programmer l'accélération d'Euler

`d(u,n,m)` calcule la suite des `delta(u,p,n)` pour `p=0..m`.
Ainsi si on pose `Delta:=d(u,n,m)` on a `Delta[p]` est égal à `delta(u,p,n)`.

```
d(u,n,m):={
  local p,rep,v;
  rep:=NULL;
  pour p de 0 jusque m faire
    v:=unapply(sum(binomial(p,k)*(-1)^(p-k)*u(n+k),k=0..p),n);
   rep:=rep,v(n);
fpour;
 return rep;
}:;
```

On tape :
```
d(w,n,3)
```
On obtient :
```
w(n),w(n+1)-(w(n)), w(n+2)-2*w(n+1)+w(n),
w(n+3)-3*w(n+2)+3*w(n+1)-(w(n))
```

```
Euler_acc(u,n,m):={
local S1,S2,j,p,Delta;
S1:=0;
pour j de 0 jusque n-1 faire
  S1:=S1+(-1)^j*u(j);
fpour;
S2:=0;
Delta:=d(u,n,m);
pour p de 0 jusque m faire
  S2:=S2+(-1)^p*Delta[p]/2^(p+1);
fpour;
return S1+(-1)^n*S2;
}:;
```

On tape :
```
Digits:=20
u(n):=evalf(1/(n+1))
Euler_acc(u,7,13),Euler_acc(u,7,13)-ln(2.)
```
On obtient :
```
0.69314718057773891525,0.17793605835720455177e-10
```
On tape :
```
Euler_acc(u,8,12),ln(2.)-Euler_acc(u,8,12)
```
On obtient :

`0.69314718054024619064,0.19699118773914073288e-10`
On tape :
`s:=(Euler_acc(u,7,13)+Euler_acc(u,8,12))/2;ln(2.)-s`
On obtient :
`0.69314718055899255295,0.95275646909680905550e-12`
On tape :
`s:=(Euler_acc(u,7,13)+Euler_acc(u,6,14))/2;ln(2.)-s`
On obtient :
`0.69314718055899255295,0.95275646909680905550e-12`
On tape :
`s:=(Euler_acc(u,19,61)+Euler_acc(u,20,60))/2;ln(2.)-s`
On obtient :
`0.69314718055994530941,0.67762635780344027125e-20`
Exercice
Calculer une valeur approchée de $S=\sum_{k=0}^{\infty}(-1)^k/(k+1)^3$.
Vérifier que $S=\zeta(3)*3/4$=`Zeta(x)*3/4`.
On tape :
```
Digits:=20
u(n):=evalf(1/(n+1)^3)
Euler_acc(u,7,13),Euler_acc(u,8,12)
```
On obtient :
`0.90154267738164926514,0.90154267735925873554`
On tape :
`Euler_acc(u,19,61),Euler_acc(u,20,60)`
On obtient :
`0.90154267736969571405,0.90154267736969571405`
On tape :
`zeta(3)*3/4.`
On obtient :
`0.90154267736969571405`
On sait que :
`Zeta(x)`=$\zeta(x)=\sum_{n=1}^{\infty}\frac{1}{n^x}$
On a :
$\sum_{n=1}^{\infty}\frac{1}{n^3}=\sum_{n=1}^{\infty}(-1)^{n-1}\frac{1}{n^3}+2\sum_{p=1}^{\infty}\frac{1}{(2p)^3}$
Donc :
$\sum_{n=1}^{\infty}(-1)^{n-1}\frac{1}{n^3}=(1-\frac{1}{4})\sum_{n=1}^{\infty}\frac{1}{n^3}$
Ce qui démontre que :
$S=\sum_{k=0}^{\infty}(-1)^k/(k+1)^3=\frac{3}{4}\zeta(3)$.
On peut aussi utiliser la transformation de Van Wijngaarden (voir 13.6.5)pour démontrer cette égalité. En effet la transformation de Van Wijngaarden transforme une série à termes positifs a_n en la série alternée $(-1)^{m-1}b_m$ avec :
$b_m=\sum_{k\geq 0}2^k a_{2^k m}$.
et on a
$\sum_{n\geq 1}a_n=\sum_{m\geq 1}(-1)^{m-1}b_m$.
Ici on a :
$a_n=1/n^3$ donc $b_m=\sum_{k\geq 0}2^k a_{2^k m}=\sum_{k\geq 0}2^k/(2^{3k}m^3)=$
$1/m^3\sum_{k\geq 0}1/(2^{2k})=4/(3m^3)$ car on a
$\sum_{k\geq 0}1/(2^{2k})=\sum_{k\geq 0}1/4^k=4/3$

Donc :
`Zeta(3)` $=4/3\sum_{m\geq 1}(-1)^{m-1}/m^3$.

13.6.4 Autre approximation d'une série alternée

La méthode présentée dans cette section est trés largement inspirée par le texte "Sommation de séries alternées" de l'épreuve de modélisation de l'agrégation de mathématiques (session 2006).

Le problème

On veut évaluer la somme S de la série alternée : $S=\sum_{n=0}^{\infty}(-1)^n a_n$
avec $(a_n)_{n\geq 0}$ est une suite de nombres positifs qui tend vers 0 en décroissant.
On suppose que l'on a pour $n\geq 0$:
$a_n=\int_0^1 x^n d\mu$
où μ est une mesure positive sur [0,1].
C'est en particulier le cas si $a_n = A(n)$ avec A fonction indéfiniment dérivable pour laquelle les $k$ième dérivées $A^{(k)}$ sont telles que $(-1)^k * A^{(k)}(x)$ soit positif pour $x\geq 0$ pour tout $k\geq 0$.

Le théorème

Théorème :
Soit P_n une suite de polynômes de degré n vérifiant $P_n(-1)\neq 0$. À P_n, on associe les coefficients $c_{n,k}$ pour $0\leq k<n$ définis par :

$$\frac{P_n(-1)-P_n(x)}{1+x}=\sum_{k=0}^{n-1}c_{n,k}x^k$$

et le coefficient d_n défini par :
$d_n=P_n(-1)$
Soient $S=\sum_{k=0}^{\infty}(-1)^k a_k$ la somme de la série alternée (où $(a_n)_{n\geq 0}=\int_0^1 x^n d\mu$ où μ est une mesure positive sur [0,1])

et $S_n=\dfrac{1}{d_n}\sum_{k=0}^{n-1}c_{n,k}a^k$

Alors :

$$|S-S_n|\geq\frac{sup_{x\in[0,1]}|P_n(x)|}{|d_n|}S$$

Démonstration On a, en effet, avec l'hypothèse faite sur les a_k :

$$S=\int_0^1\frac{1}{1+x}d\mu \text{ et}$$

$$S-S_n=\int_0^1\frac{P_n(x)}{d_n(1+x)}d\mu.$$

Le choix des polynômes P_n

Pour calculer S il reste à choisir la suite des polynômes P_n.
On peut choisir :

— $P_n(x) = (1-x)^n$
on aura une convergence en 2^{-n} car $d_n = 2^n$ et $sup_{x\in[0,1]}|P_n(x)| = 1$.
On a :
$P_0(x) = 1$
$P_1(x) = 1 - x$
$P_{n+1}(x) = P_n(x) * (1-x)$ si $n \geq 0$
$d_n = 2^n$
et la formule explicite de P_n :
$P_n(x) = \sum_{k=0}^{n}(-1)^k C_n^k x^k = \sum_{k=0}^{n} p_{n,k}x^k$ si $n \geq 0$
donc les coefficients $p_{n,k}$ vérifient :
$p_{n,0} = 1$
$p_{n,k} = p_{n,k-1} * (k-1-n)/k$ pour $1 \leq k < n$
On a :
$d_n - P_n(x) = (1+x)\sum_{k=0}^{n-1} c_{n,k}x^k = c_{n,0}\sum_{k=1}^{n}(c_{n,k-1} + c_{n,k})x^k$
donc
$c_{n,0} = d_n - p_{n,0} = d_n - 1$
$c_{n,k} = -c_{n,k-1} - p_{n,k}$ pour $1 \leq k < n$

— $P_n(x) = x^q(1-x)^{2q}$ et $n = 3 * q$
on aura une convergence en 3^{-n} car
$|d_n| = 2^{2q}$ et
$sup_{x\in[0,1]}|P_n(x|) = P_n(1/3) = 2^{2q} * 3^{-n}$.
On a :
$P_0(x) = 1$
$P_3(x) = x(1-x)^2$
$P_{n+3}(x) = P_n(x) * x * (1-x)^2$ si $n \geq 0$
$d_n = (-1)^q * 2^{2q}$
et la formule explicite de P_n :
$P_n(x) = \sum_{k=0}^{2q}(-1)^k C_{2q}^k x^{k+q} = \sum_{k=q}^{n}(-1)^{k-q}C_{2q}^{k-q}x^k =$
$\sum_{k=0}^{n} p_{n,k}x^k$ si $n \geq 0$
donc les coefficients $p_{n,k}$ vérifient :
$p_{n,k} = 0$ si $k < q$
$p_{n,q} = 1$
et comme $C_n^p = C_n^{p-1} * (n-p+1)/p$
$p_{n,k} = (-1)^{k-q}C_{2q}^{k-q}$ si $q \leq k \leq n$
$p_{n,k} = -p_{n,k-1} * (k-1-n)/(k-q)$ si $q < k \leq n$
On a :
$d_n - P_n(x) = (1+x)\sum_{k=0}^{n-1} c_{n,k}x^k = c_{n,0} + \sum_{k=1}^{n}(c_{n,k-1} + c_{n,k})x^k$
donc
$c_{n,0} = d_n - p_{n,0} = d_n - 1$
$c_{n,k} = -c_{n,k-1} - p_{n,k}$ pour $1 \leq k < n$

— Si le ploynôme P_n défini $P_n(\sin(t)^2) = \cos(2nt)$
P_n est défini à partir du ploynôme T_n de Chebyshev ($T_n(\cos(t)) = \cos(nt)$) :
$P_n(\sin(t)^2) = \cos(2nt)T_n(\cos(2t))$
on a donc puisque $1 - 2\sin(t)^2 = \cos(2t)$:
$T_n(1 - 2\sin(t)^2) = \cos(2nt)$ et
$P_n(x) = T_n(1-2x)$
on aura une convergence meilleure que dans les cas précédents car la convergence est en :

$2/((3+\sqrt{8})^n + (3-\sqrt{8})^n) \simeq 2/((3+\sqrt{8})^n) \simeq 2/(5.8)^n$
car
$d_n = ((3+\sqrt{8})^n + (3-\sqrt{8})^n)/2$ et $sup_{x\in[0,1]}|P_n(x)| = 1$.
Donc :
$P_0(x) = 1$
$P_1(x) = 1 - 2x$
$P_{n+2}(x) = 2(1-2x)P_{n+1}(x) - P_n(x)$ si $n \geq 0$
$d_n = ((3+\sqrt{8})^n + (3-\sqrt{8})^n)/2$
et la formule explicite de P_n :
$P_n(x) = \sum_{k=0}^{n}(-1)^k \frac{n}{n+k} C_{n+k}^{2k} 2^{2k} x^k =$
$\sum_{k=0}^{n} p_{n,k} x^k$ si $n \geq 0$
donc les coefficients $p_{n,k}$ vérifient :
$p_{n,0} = 1$ $p_{n,k} = p_{k-1,n}(k-1+n)(k-1-n)/((k-1/2)(k))$ pour $1 \leq k < n$ On a :
$d_n - P_n(x) = (1+x)\sum_{k=0}^{n-1} c_{n,k}x^k = c_{n,0}\sum_{k=1}^{n}(c_{n,k-1} + c_{n,k})x^k$ donc
$c_{n,0} = d_n - p_{0,n}$ $c_{n,k} = -c_{n,k-1} - p_{n,k}$ pour $1 \leq k < n$

Les formules de récurrences et le programme pour le polynôme Chebyshev

— **Les formules de récurrences**
On va calculer les coefficients $c_{n,k}$ de proche en proche pour n fixé.
On pose :
```
p:=1;
d:=((3+sqrt(8))^n+(3-sqrt(8))^n)/2;
c:=d-p;
```
Le premier terme de S_n :
```
S:=a(0)*c;
```
puis, pour `k:= 1` jusque `k:= n-1` on calcule $p_{n,k}$ et $c_{n,k}$:
```
p:=p*(k+n-1)*(k-n-1)/(k-1/2)/k;
c:=-p-c;
```
On ajoute le $k$ième terme de S_n :
```
S:=S+a(k)*c;
```

— **Le programme**

```
//n=nombres de termes et a fonction definissant a(n)
//S_n(P_n) =seriealt(n,a)
//S_n(P_n) approche sum((-1)^k*a(k),k,0,+infinity)
//avec P_n=poly de chebyshev
seriealt1(n,a):={
local k,d,c,p,S;
d:=((3+sqrt(8))^n+(3-sqrt(8))^n)/2;
p:=1;
c:=d-p;
S:=a(0)*c;
for (k:=1;k<n;k++) {
p:=p*(k+n-1)*(k-n-1)/(k-1/2)/k;
c:=-p-c;
```

```
S:=S+a(k)*c;
}
return evalf(S/d);
};
```

Les formules et le programme pour le polynôme $P_n(x) = (1-x)^n$

— **Les formules de récurrences**
On va calculer les coefficients $c_{n,k}$ de proche en proche pour n fixé.
On pose :
`p:=1;`
`d:=2^n;`
`c:=d-p;`
Le premier terme de S_n :
`S:=a(0)*c;`
puis, pour `k:= 1` jusque `k:= n-1` on calcule $p_{n,k}$ et $c_{n,k}$:
`p:=p*(k-n-1)/k;`
`c:=-p-c;`
On ajoute le $k$ième terme de S_n :
`S:=S+a(k)*c;`

— **Le programme**

```
//n=nombres de termes et a fonction definissant a(n)
//S_n(P_n) =serialt(n,a)
//S_n(P_n) approche sum((-1)^k*a(k),k,0,+infinity)
//avec P_n(x)=poly (1-x)^n
seriealt2(n,a):={
local k,d,c,p,S;
d:=2^n;
p:=1;
c:=d-p;
S:=a(0)*c;
for (k:=1;k<n;k++) {
p:=p*(k-n-1)/k;
c:=-p-c;
S:=S+a(k)*c;
}
return evalf(S/d);
};
```

Les formules et le programme pour le polynôme $P_{3q}(x) = x^q(1-x)^{2q}$

— **Les formules de récurrences**
On va calculer les coefficients $c_{n,k}$ de proche en proche pour n fixé.
On pose :
`p:=0;` si $0 \leq k < q$
`p:=1;` si $k = q$
`d:=(-1)^q*2^2q;`
`c:=d-p;`

Le premier terme de S_n :
`S:=a(0)*c;`
pour `k:=1` jusque `k:=q-1` on a $p_{n,k} = 0$ et on calcule $c_{n,k}$ (`c:=-p-c;`)
et on ajoute le $k$ième terme de S_n :
`S:=S+a(k)*c;`
puis, pour `k:=q` on a $p_{n,q} = 1$ et on calcule $c_{n,q}$ (`c:=-p-c;`) et on ajoute le $q$ième terme de S_n :
`S:=S+a(q)*c;`
puis, pour `k:=q+1` jusque `k:= n-1` on calcule $p_{n,k}$ et $c_{n,k}$:
`p:=p*(k-n-1)/k;`
`c:=-p-c;`
On ajoute le $k$ième terme de S_n :
`S:=S+a(k)*c;`

— **Le programme**

```
//n=nombres de termes et a fonction definissant a(n)
//S_n(P_n) =seriealt(n,a)
//S_n(P_n) approche sum((-1)^k*a(k),k,0,+infinity)
//avec pour n=3q, P_{3q}(x)=x^q(1-x)^{2q}$
seriealt3(n,a):={
local k,d,c,p,q,S;
q:=ceil(n/3);
n:=3*q;
d:=(-1)^q*2^(2*q);
p:=0;
c:=d-p;
S:=a(0)*c;
for (k:=1;k<q;k++) {
c:=-p-c;
S:=S+a(k)*c;
}
p:=1;
c:=-c-p;
S:=S+a(q)*c;
for (k:=q+1;k<n;k++) {
p:=p*(k-n-1)/(k-q);
c:=-p-c;
S:=S+a(k)*c;
}
return evalf(S/d);
};
```

Les essais

On choisit `n=20`.
On tape :
`evalf(2/(3+sqrt(8))^20,2^-20,3^-21)=`

```
9.77243031253e-16,9.53674316406e-07,9.55990663597e-11
```

On a donc pour n=20 une approximation en 10^{-15} pour Chebyshev, en 10^{-6} pour $(1-x)^{20}$ et en 10^{-10} pour $x^7(1-x)^14$:
On choisit dans la suite `Digits:=20`

Pour calculer une approximation de $\pi/4$.
On a :

```
sum((-1)^n/(2*n+1),n,0,+infinity)=pi/4
```

On tape :

```
b(n):=1/(2*n+1)
seriealt1(20,b);evalf(pi/4)
```

On obtient :

```
0.78539816339744830926, 0.78539816339744830615
```

On tape :

```
seriealt2(20,b);seriealt3(20,b);
```

On obtient :

```
0.78539798191878673159, 0.78539816341320102597
```

Pour calculer une approximation de $\ln(2)$.
On a :

```
sum((-1)^n/(n+1),n,0,+infinity)=ln(2)
```

On tape :

```
a(n):=1/(n+1)
seriealt1(30,a);evalf(ln(2))
```

On obtient :

```
0.69314718055994531245, 0.69314718055994530941
```

On tape :

```
seriealt2(20,a);seriealt3(20,a);
```

On obtient :

```
0.69314713705102893627, 0.69314718057773891525
```

13.6.5 Transformation d'une série à termes de signe constant en une série alternée

On décrit ici la transformation de Van Wijngaarden qui transforme une série à termes de signe constant (par exemple $a_n > 0$) en une série alternée $(-1)^{n-1}b_n$.
On a l'identité formelle :
$\sum_{n\geq 1} a_n = \sum_{m\geq 1}(-1)^{m-1}b_m$ avec
$b_m = \sum_{k\geq 0} 2^k a_{2^k m}$.
Montrons que le coefficient de a_n vaut 1 dans la somme :
$\sum_{m\geq 1}(-1)^{m-1}\sum_{k\geq 0} 2^k a_{2^k m}$.
En effet, pour tout entier n, il existe un couple unique d'entiers (s, m_s) tel que $n = 2^s * m_s$ avec m_s impair.
si n est impair on a $n = 2^0 * n$ donc $s = 0$ et $m_s = n$ d'où
$(-1)^{n-1}2^0 a_{2^0 n} = a_n$
si n est pair on a : $n = 2^s * m_s = 2^k m_k$ pour $(k = 0..s)$ avec m_s impair et $s \neq 0$ et donc $m_k = 2^{s-k}m_s$ est pair si $k = 0..s-1$.
Donc : pour $k = 0..s-1$ on a $(-1)^{m_k-1}2^k a_{2^k m_k} = -2^k a_n$ et

pour $k = s$ on a $(-1)^{m_s-1}2^s a_{2^s m_s} = 2^s a_n$.
Donc :
$\sum_{k=0}^{s}(-1)^{m_k-1}2^k a_{2^k m_k} = a_n(-\sum_{k=0}^{s-1}2^k + 2^s) = a_n(-(2^s-1)+2^s) = a_n$.
Remarques

1-Soit $a(n) > 0$ pour $n > 0$.
Si la série de terme général $a(n)$ converge alors pour $m \geq 1$, les séries de terme général $w(k) = 2^k a(2^k m)$ qui définissent les $b(m)$ ne convergent pas toujours !
En effet, soit $a(n)$ défini par :
si n est une puissance de 2 alors $a(n) = 1/n$ sinon $a(n) = 1/n^2$.
La série de terme général $a(n)$ converge vers $\sum_{k>0} 1/n^2 + \sum_{k\geq 0}(1/2^k - 1/2^{2k}) + \pi^2/6 + 2 - 4/3 = \pi^2/6 + 2/3$
Dans ce cas, lorsque $m = 2^p$ avec $p = 0..infty$ les séries qui définissent les $b(m)$ ne convergent pas i.e.$\sum_{k=0}^{\infty} w(k)$ avec $w(k) = 2^k a(2^k * 2^p) = 1/2^p$ sont divergentes puisque pour $m = 2^p$ on a $w(k) = 1/2^p = cste$.
Donc si on pose $b(m) = \sum_{k=0}^{\infty} 2^k a(2^k * m)$ on a :
$b(1)$, $b(2)$,..., $b(2^p) = sum_{k=0}^{\infty} 2^k a(2^{k+p}) = sum_{k=0}^{\infty}(1/2^p)$ sont infinis.

2-Si la série de terme général $a(n)$ converge vers S et que $a(n)$ tend vers 0 en décroissant à partir d'un certain rang M alors les $b(m) = \sum_{k=0}^{\infty} 2^k a(2^k * m)$ sont bien défini et la suite $b(m)$ est décroissante à partir du rang M.
En effet supposons que $a(n)$ est décroissante à partir du rang $n = M$.
On peut supposer que $M = 1$ car sinon on travaille avec la série de terme général $u(n) = a(n + M)$ pour $n \geq 1$.
On a alors :
$a(2) \leq a(2)$
$2a(4) < a(3) + a(4)$
$4a(8) < a(5) + a(6) + a(7) + a(8)$
.....
$2^{K-1}a(2^K) < a(2^{K-1} + 1) + .. + a(2^K)$
Donc :
$S(K) = \sum_{k=0}^{K} 2^k a(2^k) = a(1) + 2\sum_{k=1}^{K} 2^{k-1}a(2^k) < a(1) + 2(S - a(1)) = 2S - a(1)$
$S(K)$ est une suite croissante et majorée donc elle est convergente donc $b(1) = \sum_{k\geq 0} 2^k a(2^k)$ est bien défini.
Les séries de terme général $w(k) = 2^k a(2^k m)$ pour $m \in \mathbb{N}^*$ sont convergentes puisque on a supposé $a(n)$ décroissante pour $n \geq 1$ on a donc $2^k a(2^k m) \leq 2^k a(2^k)$ pour $m \geq 1$.
) On vient de montrer que la série de terme général $2^k a(2^k)$ converge donc la série de terme général $2^k a(2^k m)$ converge et donc $b(m) = \sum_{k\geq 0} 2^k a(2^k m)$ est bien défini et $b(m)$ est de plus décroissante puisque $a(2^k m)$ est décroissante.

3-On calcule $\sum_{m=1}^{10}(-1)^{m-1}b(m)$, on a :
$(-1)^0 b(1) = a(1) + 2a(2) + 4a(4) + 8a(8) + 16a(16) + \sum_{k=5}^{\infty} 2^k a(2^k)$
$(-1)^1 b(2) = -a(2) - 2a(4) - 4a(8) - 8a(16) - \sum_{k=4}^{\infty} 2^k a(2^k * 2)$
$(-1)^2 b(3) = a(3) + 2a(6) + 4a(12) + 8a(24) \sum_{k=4}^{\infty} 2^k a(2^k * 3)$
$(-1)^3 b(4) = -a(4) - 2a(8) - 4a(16) - \sum_{k=3}^{\infty} 2^k a(2^k * 4)$
$(-1)^4 b(5) = a(5) + 2a(10) + 4a(20) + \sum_{k=3}^{\infty} 2^k a(2^k * 5)$
$(-1)^5 b(6) = -a(6) - 2a(12) - 4a(24) - \sum_{k=3}^{\infty} 2^k a(2^k * 6)$
$(-1)^6 b(7) = a(7) + 2a(14) + 4a(28) \sum_{k=3}^{\infty} 2^k a(2^k * 7)$
$(-1)^7 b(8) = -a(8) - 2a(16) - \sum_{k=2}^{\infty} 2^k a(2^k * 8)$

$(-1)^8 b(9) = a(9) + 2a(18) + \sum_{k=2}^{\infty} 2^k a(2^k * 9)$
$(-1)^9 b(10) = -a(10) - 2a(20) + \sum_{k=2}^{\infty} 2^k a(2^k * 10)$
$\sum_{m=1}^{9}(-1)^{m-1} b(m) =$
$a(1) + (2-1)a(2)) + a(3) + (4-2-1)a(4) + a(5) + (2-1)a(6) + a(7) + (8 -$
$4 - 2 - 1)a(8) + a(9) + a(10) +$
$(4-2)a(12) + 2a(14) + (16 - 8 - 4 - 2)a(16) + 2a(18) + (4-2)a(20) +$
$(8-4)a(24) + 4a(28) + (32 - 16 - 8 - 4)a(32) + 4a(36) + 4a(40) + 8a(48) + ...$
Donc
$\sum_{m=1}^{10}(-1)^{m-1} b(m) = \sum_{m=1}^{10} a(m) + \sum_{m=6}^{10} 2a(2m) + \sum_{m=6}^{10} 4a(4m)$+.....
$\sum_{m=1}^{10}(-1)^{m-1} b(m) = \sum_{m=1}^{10} a(m) + \sum_{k=1}^{+\infty}(\sum_{m=6}^{10} 2^k a(2^k m))$ On a 6=`iquo(10,2)+1`
Plus généralement, si on pose $M_0 =$`iquo`$(M, 2) + 1$, on a :

$$\sum_{m=1}^{M}(-1)^{m-1} b(m) = \sum_{m=1}^{M} a(m) + \sum_{m=M_0}^{M}(\sum_{k=1}^{\infty} 2^k a(2^k m))$$

ou encore :

$$\sum_{m=1}^{M}(-1)^{m-1} b(m) = \sum_{m=1}^{M} a(m) + \sum_{k=1}^{\infty} 2^k \sum_{m=M_0}^{M} a(2^k m)$$

$b(m) = \sum_{k\geq 0} 2^k a(2^k m) = a(m) + \sum_{k\geq 1} 2^k a(2^k m))$
Donc :
$\sum_{m=1}^{M}(-1)^{m-1} b(m) = \sum_{m=1}^{M} a(m) + \sum_{m=M_0}^{M}(b(m) - a(m)) =$
$\sum_{m=1}^{M}(-1)^{m-1} b(m) = \sum_{m=1}^{M_0-1} a(m) + \sum_{m=M_0}^{M} b(m)$
On eb déduit que $\sum_{m=M_0}^{M} b(m)$ tend vers 0 quand M tend vers l'infini mais cela n'entraine pas que la série de terme général b_m converge !

Exercices

1. Si la série de terme général u_n comverge, est ce qu'on peut en déduire que $\lim_{n->+\infty} nu_n = 0$?
   Si $u_n = \frac{(-1)^n}{n}$ la série de terme général u_n comverge et nu_n n'a pas de limite quand $n- > +\infty$
   Soit u_n défini par :
   si $n = p^2$ alors $u_n = \frac{(1}{n}$ sinon $u_n = \frac{(1}{n^2}$
   Ici u_n est positif pour tout $n \in \mathbb{N}^*$ et la série de terme général u_n comverge puisque :
   $\sum n = 1^{\infty} u_n = \sum n = 1^{\infty} \frac{(1}{n^2} + \sum p = 1^{\infty} - \frac{(1}{p^4} + \frac{(1}{p^2} =$
   $2\sum n = 1^{\infty} \frac{1}{n^2} - \frac{(1}{n^4} = pi^2(\frac{1}{3} - \frac{pi^2}{90}) = \frac{30pi^2 - pi^4}{90}$
   Pourtant :
   $\lim_{p->+\infty} p^2 u_{p^2} = 1$ et $\lim_{p->+\infty}(p^2+1)u_{p^2+1} = \lim_{p->+\infty} \frac{(1}{p^2+1} = 0$
   Donc on ne peut pas affirmer que $\lim_{n->+\infty} nu_n = 0$ si la série de terme général u_n comverge.
2. Si la suite de terme général u_n converge vers 0 en décroissant et si la suite nu_n converge vers 0, est ce qu'on peut en déduire que la série de terme général u_n comverge ?

On ne peut pas en déduire que la série de terme général u_n comverge.
Par exemple, la série de terme général $u_n = \dfrac{1}{n\ln(n)}$ $(n > 1)$ diverge et pourtant u_n tend vers 0 en décroissant et la suite $nu_n = \dfrac{1}{\ln(n)}$ converge vers 0.

3. si la suite de terme général u_n converge vers 0 en décroissant et si la suite $w_m = \sum_{k=m+1}^{2*m} u_k$ converge vers 0, est ce qu'on peut en déduire que la série de terme général u_n comverge ?
   On ne peut pas en déduire que la série de terme général u_n comverge.
   Par exemple, la série de terme général $u_n = \dfrac{1}{n\ln(n)}$ $(n > 1)$ diverge et pourtant u_n tend vers 0 en décroissant et la suite w_m converge vers 0 puisque :
   $$0 < w_m = \sum_{k=m+1}^{2*m} u_k < mu_{m+1} < \frac{m}{(m+1)\ln(m+1)}$$

Application au calcul de $\sum_{n=0}^{\infty} \frac{1}{n^s}$

Prenons comme exemple la série de terme général $a(n) = \frac{1}{n^s}$ avec $s > 1$.
$a(n)$ est une suite décroissante car si $0 < p < q$ on a $\frac{1}{p^s} > \frac{1}{q^s}$ pour `s=2`
si `a(n)=1/n^2`
on a pour $k \in \mathbb{N}$ et pour $m > 1$:
`2^k*a(2^k*m)=1/(2^k*m^2)`
`b(m)=1/m^2*sum(1/2^k,k,0,+infinity)=2/m^2`
pour `s>1`
si `a(n)=1/n^s`
On a pour $k \in \mathbb{N}$ et pour $m > 1$:
`2^k*a(2^k*m)=1/(2^(k*(s-1))*m^s)`
`b(m)=1/m^s*sum((1/2^(s-1))^k,k,0,+infinity)`
Donc :
`b(m)=2^(s-1)/((2^(s-1)-1)*m^s)`
pour $s = 3/2$
`normal(sum(1/(sqrt(2)^k),k=0..inf))` renvoie `sqrt(2)+2` donc
`b(m):=(2+sqrt(2))/m^(3/2)`
pour $s = 2$
`b(m):=2/(m^2)`
pour $s = 3$
`normal(sum(1/4^k),k=0..inf))` renvoie `4/3` donc
`b(m):=4/(3*m^3)`
pour $s = 4$
`b(m):=8/(7*m^4)`
On a :
`sum((-1)^(m-1)*b(m),1,+infinity)=`
`sum((-1)^(m)*b(m+1),0,+infinity)`

On choisit encore `Digits:=20`
on rappelle les programmes vu précédemment 13.6.4, 13.6.4 et **??**)

```
//n=nombres de termes et a fonction definissant a(n)
//S_n(P_n) =serialt(n,a)
//S_n(P_n) approche sum((-1)^k*a(k),k,0,+infinity)
//avec P_n=poly de chebyshev
seriealt1(n,a):={
local k,d,c,p,S;
d:=((3+sqrt(8))^n+(3-sqrt(8))^n)/2;
p:=1;
c:=d-p;
S:=a(0)*c;
for (k:=1;k<n;k++) {
p:=p*(k+n-1)*(k-n-1)/(k-1/2)/k;
c:=-p-c;
S:=S+a(k)*c;
}
return evalf(S/d);
}:;
```

et

```
//n=nombres de termes et a fonction definissant a(n)
//S_n(P_n) =serialt(n,a)
//S_n(P_n) approche sum((-1)^k*a(k),k,0,+infinity)
//avec P_n(x)=poly (1-x)^n
seriealt2(n,a):={
local k,d,c,p,S;
d:=2^n;
p:=1;
c:=d-p;
S:=a(0)*c;
for (k:=1;k<n;k++) {
p:=p*(k-n-1)/k;
c:=-p-c;
S:=S+a(k)*c;
}
return evalf(S/d);
}:;
```

```
//n=nombres de termes et a fonction definissant a(n)
//S_n(P_n) =serialt(n,a)
//S_n(P_n) approche sum((-1)^k*a(k),k,0,+infinity)
//avec pour n=3q, P_{3q}(x)=x^q(1-x)^{2q}$
seriealt3(n,a):={
local k,d,c,p,q,S;
q:=ceil(n/3);
n:=3*q;
d:=(-1)^q*2^(2*q);
p:=0;
c:=d-p;
```

```
S:=a(0)*c;
for (k:=1;k<q;k++) {
c:=-p-c;
S:=S+a(k)*c;
}
p:=1;
c:=-c-p;
S:=S+a(q)*c;
for (k:=q+1;k<n;k++) {
p:=p*(k-n-1)/(k-q);
c:=-p-c;
S:=S+a(k)*c;
}
return evalf(S/d);
}:;
```

```
d(u,n,m):={
  local p,rep,v;
  rep:=NULL;
  pour p de 0 jusque m faire
    v:=unapply(sum(binomial(p,k)*(-1)^(p-k)*u(n+k),k=0..p),n);
   rep:=rep,v(n);
fpour;
 return rep;
}:;
Euler_acc(u,n,m):={
local S1,S2,j,p,Delta;
S1:=0;
pour j de 0 jusque n-1 faire
  S1:=S1+(-1)^j*u(j);
fpour;
S2:=0;
Delta:=d(u,n,m);
pour p de 0 jusque m faire
  S2:=S2+(-1)^p*Delta[p]/2^(p+1);
fpour;
return S1+(-1)^n*S2;
}:;
```

pour $s = 3/2$, $\sum_{n=1}^{\infty} 1/n^{3/2} = \zeta(3/2)$
On a :
On tape :

```
t1(m):=(2+sqrt(2.)/(m+1)^(3/2)
seriealt1(20,t1),evalf(Zeta(3/2)),sum(1./n^(3/2),n=1..100000)
```

On obtient :

```
2.6123753486854883558,2.6123753486854883433,2.6060508091765003573
```

On tape :

```
seriealt2(20,t1);seriealt3(20,t1);Euler_acc(t1,3,20);
```

On obtient :

```
2.6123750312809107172, 2.6123753487573583463,2.6123753487299233493
```

pour $s=2$, $\sum_{n=1}^{\infty} 1/n^2 = \pi^2/6$

On tape :

```
t2(m):=2/(m+1)^2
seriealt1(20,t2),evalf(pi^2/6),sum(1./n^2,n=1..100000)
```

On obtient :

```
1.6449340668482264 5248, 1.6449340668482264 3645,1.6449240668982262687
```

On tape :

```
seriealt2(20,t2);seriealt3(20,t2);Euler_acc(t2,3,20)
```

On obtient :

```
1.6449337461377753 4516, 1.6449340668880559 9300,1.6449340668811822652
```

pour $s=3$, $\sum_{n=1}^{\infty} 1/n^3 = \zeta(3)$

On tape :

```
t3(m):=4/(3.*(m+1)^3)
seriealt1(20,t3),evalf(Zeta(3)),sum(1./n^3,n=1..100000)
```

On obtient :

```
1.2020569031595943143,1.2020569031595942854,1.2020569031095947871
```

On tape :

```
seriealt2(20,t3);seriealt3(20,t3);Euler_acc(t3,3,20)
```

On obtient :

```
1.2020564626522540078,1.2020569031755323535,1.2020569031830323264
```

pour $s=4$, $\sum_{n=1}^{\infty} 1/n^4 = \pi^4/90$

On tape :

```
t4(m):=8/(7*(m+1)^4)
seriealt1(20,t4);evalf(pi^4/90),sum(1./n^4,n=1..100000)
```

On obtient :

```
1.0823232337111382 2384, 1.0823232337111381 9149,1.0823232337111378597
```

On tape :

```
seriealt2(20,t4);seriealt3(20,t4);Euler_acc(t4,3,20);
```

On obtient :

```
1.0823226519891244 0013, 1.0823232337169792 5335,1.0823232337262781598
```

Application au calcul de la constante d'Euler

Pour calculer une approximation de la constante d'Euler, (voir aussi 15.2.3 et 13.8.3).

```
γ=-psi(1).
```

On a :

```
-psi(1)=sum((-1)^n*ln(n)/n,n,1,+infinity)/ln(2)+ln(2)/2
```

et

```
sum((-1)^n*ln(n)/n,n,1,+infinity)=
-sum((-1)^n*ln(n+1)/(n+1),n,0,+infinity)
c(n):=log(n+1)/(n+1)
-seriealt1(20,c)/ln(2)+ln(2)/2;-evalf(psi(1),0)
```

On obtient :
`0.577215664901532859864, 0.57721566490153`
On tape :
`-seriealt2(20,c)/ln(2)+ln(2)/2,-seriealt3(20,c)/ln(2)+ln(2)/2`
On obtient :
`0.577215550220266823551, 0.577215664918305723256` On tape :
```
Digits:=24;
evalf(euler_gamma)
```
On obtient : `0.577215664901532860606511 9`

Exercice

Si la série de terme général u_n comverge est-ce-qu'on peut en déduire que $\lim_{n->+\infty} nu_n = 0$? Si $u_n = \frac{(-1)^n}{n}$ la série de terme général u_n comverge et nu_n n'a pas de limite quand $n- > +\infty$
Soit u_n défini par :
si $n = p^2$ alors $u_n = \frac{(1}{n}$ sinon $u_n = \frac{(1}{n^2}$ Ici u_n est positif pour tout $n \in \mathbb{N}^*$ et la série de terme général u_n comverge puisque :
$\sum n = 1^\infty u_n = \sum n = 1^\infty \frac{(1}{n^2} + \sum p = 1^\infty - \frac{(1}{p^4} + \frac{(1}{p^2} = 2 \sum n = 1^\infty \frac{1}{n^2} - \frac{(1}{n^4} = pi^2(\frac{1}{3} - \frac{pi^2}{90}) = \frac{30pi^2 - pi^4}{90}$
Pourtant :
$\lim_{p->+\infty} p^2 u_{p^2} = 1$ et $\lim_{p->+\infty}(p^2+1)u_{p^2+1} = \lim_{p->+\infty} \frac{(1}{p^2+1} = 0$
Donc on ne peut pas affirmer que $\lim_{n->+\infty} nu_n = 0$ si la série de terme général u_n comverge.

13.7 Polynômes de Bernstein

13.7.1 Définition et théorème

Définition
Le n-ième polynômes de Bernstein associé à f continue sur [0,1] est :

$$B_n(f)(t) = \sum_{p=0}^{n} C_n^p f(\frac{p}{n})(1-t)^{n-p}t^p$$

Théorème
Si f est continue la suite $B_n(f)$ converge uniformément vers f sur $I = [0,1]$.
Remarque Cette convergence est extêmement lente !

13.7.2 La démonstration

On utilise des théorèmes de probabilité pour montrer que :
pour tout $\epsilon > 0$, il existe $N_0 \in \mathbb{N}$ tel que pour tout $n > N_0$ et pour tout $x \in [0,1]$ on a : $|B_n(f)(x) - f(x)| \leq 2\epsilon$.
On a : $|B_n(f)(x) - f(x)| \leq \sum_{p=0}^{n} C_n^p |f(\frac{k}{n}) - f(x)|(1-x)^{n-k}x^k$

Pour tout $x \in [0,1]$, on considère X une variable aléatoire suivant une loi binomiale $B(n,x)$. Dans une série de n épreuves, X a pour valeur le nombre d'événements, de probabilité x, obtenus. C'est à dire X prend les valeurs $0, 1, ...k, ...n$ avec Prob$(X = k) = C_n^k x^k (1-x)^{n-k}$.
On a :
$\sum_{k=0}^{n} C_n^k x^k (1-x)^n = 1$
$E(X) = \sum_{k=0}^{n} k C_n^k x^k (1-x)^n = nx$
$V(X) = \sum_{k=0}^{n} (k-nx)^2 C_n^k x^k (1-x)^n = nx(1-x)$
Prob$(|X - E(X)| \geq a) \leq \dfrac{V(X)}{a^2}$ (inégalité de Bienaymé-Tchébicheff).
Soient $n \in \mathbb{N}$, $x \in [0,1]$. Pour tout $d > 0$, on définit :
$A_n(d) = \{k \in \mathbb{N},\ \ 0 \leq k \leq n, |k/n - x| \geq d\}$
$B_n(d)$ le complémentaire de $A_n(d)$ dans $\mathbb{N}$.
Si $k \in A_n(d)$, on a $|k - nx| = |X - E(X)| \geq nd$ et d'après l'inégalité de Bienaymé-Tchébicheff on a :
$\sum_{k \in A_n(d)} C_n^k x^k (1-x)^n = \sum_{k \in A_n(d)}$ Prob$(X = k)$= Prob$(|k - nx| \geq nd)$=
Prob$(|X - E(X)| \geq nd) \leq \dfrac{V(X)}{n^2 d^2} = \dfrac{x(1-x)}{nd^2} \leq \dfrac{1}{4nd^2}$
car si $x \in [0,1]$ on a $0 \leq x(1-x) \leq \frac{1}{4}$
f est uniformément continue sur $[0,1]$, donc pour tout $\epsilon > 0$, il existe $\eta > 0$, tel que pour tout $(x_1, x_2) \in [0,1]^2$ vérifiant $|x_1 - x_2| < \eta$ on a $|f(x_1) - f(x_2)| < \epsilon$
Posons $d = \eta$ et $M = \sup_{x \in [0,1]} |f(x)|$.
On a :
$\sum_{A_n(\eta)} C_n^p \ |f(\dfrac{k}{n}) - f(x)|(1-x)^{n-k} x^k \leq 2M \sum_{A_n(\eta)} C_n^p (1-x)^{n-k} x^k \leq \dfrac{2M}{4n\eta^2}$
si $k \in B_n(\eta)$, on a $|k/n - x| < \eta$ donc $|f(\frac{k}{n}) - f(x)| < \epsilon$ donc :
$\sum_{B_n(\eta)} C_n^p \ |f(\dfrac{k}{n}) - f(x)|(1-x)^{n-k} x^k \leq \epsilon * \sum_{k=0}^{n} C_n^k x^k (1-x)^n = \epsilon$
Donc $|B_n(f)(x) - f(x)| \leq \dfrac{M}{2n\eta^2} + \epsilon$
Comme $\dfrac{2M}{4n\eta^2}$ tend vers 0 quand n tend vers l'infini, il existe N_0 tel que pour tout $n > N_0$ on a $\dfrac{M}{2n\eta^2} < \epsilon$ donc :
pour tout ϵ, il existe $N_0 \in \mathbb{N}$ tel que pour tout $n > N_0$ et pour tout $x \in [0,1]$ on a : $|B_n(f)(x) - f(x)| \leq 2\epsilon$.

13.7.3 Le programme

bernstein(f,n,t) approche uniformement f continue sur [0,1].
On tape :

```
bernstein(f,n,t):={
retourne sum(comb(n,p)*f(p/n)*(1-t)^(n-p)*t^p,p=0..n);
  }:;
```

bernab(f,n,t,a,b) approche uniformement f continue sur $[a,b]$.
On tape :

```
bernab(f,n,t,a,b):={
 retourne  sum(comb(n,p)*f(a*(1-p/n)+b*p/n)*(b-t)^(n-p)*(t-a)^p/(b-a)
   }:;
```

On tape :
`plotfunc([bernstein(sin,12,x),sin(x)],x)`
On obtient :
`Un graphe proche de sin(x) sur [0,1]`
On tape :
`plotfunc([bernstein(sin,12,x,-pi/2,pi/2),sin(x)],x)`
On obtient :
`Un graphe proche de sin(x) sur [-pi/2,pi/2]`

13.7.4 Les courbes de Bézier

Définition

Une courbe de Bézier de degré n définie par $n+1$ points $(a_0, a_1..a_n)$ est le graphe du polynôme de Bernstein de degré n $\sum_{k=0}^{n} a_k C_n^k x^k (1-x)^n$.
Cette courbe approche le graphe d'une fonction f qui vérifie pour $k = 0..n+1$, $f(k/n) = a_k$.
Dans la pratique on utilise que des polynômes de Bernstein de degré 2 ou 3.
Ainsi, pour interpoler une fonction f sur $[a,b]$ on utilise plusieurs polynômes P_k ($k = 0..n-1$) avec P_k interpolant f sur $[x_k, x_{k+1}]$ ($x_0 = a, x_n = b$).
Dans le plan une courbe de Bézier définie par 4 points a pour équation paramétrique :
$x(t) = a_0(1-t)^3 + 3a_1t(1-t)^2 + 3a_2t^2(1-t) + a_3t^3$
$y(t) = b_0(1-t)^3 + 3b_1t(1-t)^2 + 3b_2t^2(1-t) + b_3t^3$
cette courbe approche la courbe passant par les points A_k d'affixe $a_k + ib_k$ pour $k = 0..3$: les points A_0 et A_3 sont les points de base et les points A_1 et A_2 sont les points de contrôle.
Dans le plan une courbe de Bézier définie par 3 points a pour équation paramétrique :
$x(t) = a_0(1-t)^2 + 2a_1t(1-t) + a_2t^2$
$y(t) = b_0(1-t)^2 + 2b_1t(1-t)^+b_2t^2$
cette courbe approche la courbe passant par les points A_k d'affixe $a_k + ib_k$ pour $k = 0..2$: les points A_0 et A_2 sont les points de base et le points A_1 est le point de contrôle.
On pourra se reporter au manuel de géométrie pour voir le morphing comme application des courbes de Bézier.

13.8 Développements asymptotiques et séries divergentes

Un développement asymptotique est une généralisation d'un développement de Taylor, par exemple lorsque le point de développement est en l'infini. De nombreuses fonctions ayant une limite en l'infini admettent un développement asymptotique en l'infini, mais ces développements sont souvent des séries qui semblent

commencer par converger mais sont divergentes. Ce type de développement s'avère néanmoins très utile lorsqu'on n'a pas besoin d'une trop grande précision sur la valeur de la fonction.

13.8.1 Exercice : un développement asymptotique

Développement limité à l'ordre 3 de $\text{asin}(\tanh(x))$ au voisinage de $+\infty$.

Avec Xcas

Sans la commande `series`

On sait que :

$\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$

On pose $X = \exp(-x)$, donc X tend vers 0^+ quand x tend vers $+\infty$.

$\text{asin}(\tanh(x)) = \text{asin}(\frac{1-X^2}{1+X^2}) = \text{asin}(1 - \frac{2X^2}{1+X^2})$

Cherchons le développement limité à l'ordre 3 de :

$f(X) = \text{asin}(\frac{1-X^2}{1+X^2})$ au voisinage de $X = 0^+$.

Attention

$f(X)$ n'est pas dérivable en 0 mais a une dérivé à droite et une dérivé à gauche, en effet :

On tape :

```
f(X):=asin((1-X^2)/(1+X^2))
simplify(diff(f(X),X))
```

On obtient :

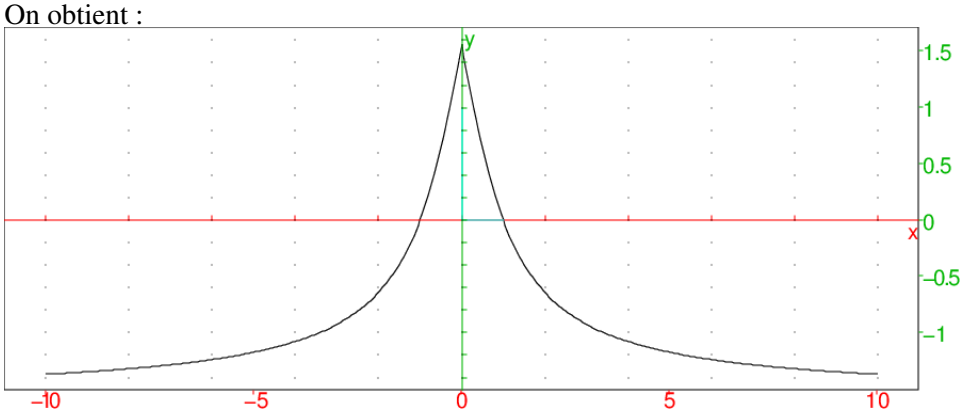
```
-2*X/abs(X+X^3)
```

Quand $X->0^+$, $f'(X)->-2$ et Quand $X->0^-$, $f'(X)->+2$.

On tape :

```
plotfunc(f(X),X)
```

On obtient :



On a pour $X > 0$:

$f(X) = f(0)+Xf((0)+X^2f"(0)/2+X^3f^{(3)}(0)/6+X^3\epsilon(X)$ avec $\epsilon(X)->0$ quand $X->0^+$.

On a :

$f(0) = \text{asin}(1) = \pi/2$

Calcul de $f'(X)$ lorsque $X \geq 0$:

On tape :

```
assume(X>=0)
simplify(diff(f(X),X))
```

On obtient :

`-2/(1+X^2)`
Donc $f'0) = -2$
On tape :
`simplify(diff(-2/(1+X^2),X`
On obtient :
`4*X/(1+X^2)^2` Donc $f''0) = 0$
On tape :
`simplify(diff(4*X/(1+X^2)^2,X`
On obtient :
`(4-12*X^2)/(1+X^2)^3` Donc $f'''0) = 4$
Donc :
$f(X) = \pi/2 - 2X + 2X^3/3 + X^3\epsilon(X)$
On a $X = \exp(-x)$ donc $\text{asin}\,(\tanh(x)) = \pi/2 - 2*\exp(-x) + 2\exp(-x)^3/3 + \exp(-x)^3\epsilon(x)$ avec
$\epsilon(x) -> 0$ quand $x -> +\infty$.

Avec la commande `series`
On sait que :
$\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$
On pose $X = \exp(-x)$, donc X tend vers 0^+ quand x tend vers $+\infty$.
On tape :
`series(asin((1-X^2)/(1+X^2)),X=0,4,1)`
On obtient :
`pi/2-2*X+2*X^3/3+X^4*order_size(X)`
Ou on tape :
`series(asin(tanh(x)),x=inf,4)`
On obtient :
`1/2*pi-2/exp(x)+2*(1/exp(x))^3/3+(1/exp(x))^4*order_size(1/exp(x))`
Remarque
On tape :
`series(asin((1-X^2)/(1+X^2)),X=0,4,-1)`
Donc quand X tend vers 0^-, on obtient :
`pi/2+2*X-2*X^3/3+X^4*order_size(X)`

13.8.2 Un exemple :la fonction exponentielle intégrale

Nous allons illustrer ce type de développement sur un exemple, la fonction exponentielle intégrale, définie à une constante près par

$$f(x) = \int_x^{+\infty} \frac{e^{-t}}{t}\, dt$$

On peut montrer que l'intégrale existe bien, car l'intégrant est positif et inférieur à e^{-t} (qui admet $-e^{-t}$ comme primitive, cette primitive ayant une limite en $+\infty$).
Pour trouver le développement asymptotique de f en $+\infty$, on effectue des intégrations par parties répétées, en intégrant l'exponentielle et en dérivant la fraction

rationnelle :

$$\begin{aligned}
f(x) &= [\frac{-e^{-t}}{t}]_x^{+\infty} - \int_x^{+\infty} \frac{-e^{-t}}{-t^2}\, dt \\
&= \frac{e^{-x}}{x} - \int_x^{+\infty} \frac{e^{-t}}{t^2}\, dt \\
&= \frac{e^{-x}}{x} - ([\frac{-e^{-t}}{t^2}]_x^{+\infty} - \int_x^{+\infty} \frac{-2e^{-t}}{-t^3}) \\
&= \frac{e^{-x}}{x} - \frac{e^{-x}}{x^2} + \int_x^{+\infty} \frac{2e^{-t}}{t^3}\, dt \\
&= \ldots \\
&= e^{-x}\left(\frac{1}{x} - \frac{1}{x^2} + \frac{2}{x^3} + \ldots + \frac{(-1)^n n!}{x^{n+1}}\right) - \int_x^{+\infty} \frac{(-1)^n n! e^{-t}}{t^{n+1}}\, dt \\
&= S(x) + R(x)
\end{aligned}$$

où

$$S(x) = e^{-x}\left(\frac{1}{x} - \frac{1}{x^2} + \frac{2}{x^3} + \ldots + \frac{(-1)^n n!}{x^{n+1}}\right), \quad R(x) = -\int_x^{+\infty} \frac{(-1)^n n! e^{-t}}{t^{n+1}}\, dt \tag{13.1}$$

Le développement en séries est divergent puisque pour $x > 0$ fixé et n tendant vers l'infini

$$\lim_{n\to+\infty} \frac{n!}{x^{n+1}} = +\infty$$

mais si x est grand, au début la série semble converger, de manière très rapide :

$$\frac{1}{x} >> \frac{1}{x^2} >> \frac{2}{x^3}$$

On peut utiliser $S(x)$ comme valeur approchée de $f(x)$ pour x grand si on sait majorer $R(x)$ par un nombre suffisamment petit. On a

$$|R(x)| \leq \int_x^{+\infty} \frac{n! e^{-t}}{x^{n+1}} = \frac{n! e^{-x}}{x^{n+1}}$$

On retrouve une majoration du type de celle des séries alternées, l'erreur est inférieure à la valeur absolue du dernier terme sommé. Pour x fixé assez grand, il faut donc de trouver un rang n, s'il en existe un, tel que $n!/x^n < \epsilon$ où ϵ est la précision relative que l'on s'est fixée. Par exemple, si $x \geq 100$, $n = 12$ convient pour $\epsilon = 12!/100^{12} = 5e-16$ (à peu près la précision relative d'un "double").

13.8.3 Le calcul approché de la constante d'Euler γ

Pour d'autres méthodes concernant le calcul approché de la constante d'Euler voir aussi 13.6.5 et 15.2.3.
On peut montrer que

$$\lim_{n\to+\infty} u_n, \quad u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n) \tag{13.2}$$

existe (par exemple en cherchant un équivalent de $u_{n+1} - u_n$ qui vaut $\frac{-1}{2n^2}$) et on définit γ comme sa limite. Malheureusement, la convergence est très lente et cette

définition n'est pas applicable pour obtenir la valeur de γ avec une très grande précision. Il y a un lien entre γ et la fonction exponentielle intégrale (définie par $f(x) = \int_x^{+\infty} \frac{e^{-t}}{t}\, dt$) , plus précisément lorsque $x \to 0$, $f(x)$ admet $-\ln(x)$ comme singularité, plus précisément $f(x) + \ln(x)$ admet un développement en séries (de rayon de convergence $+\infty$), car :

$$\begin{aligned} f(x) + \ln(x) &= \int_x^1 \frac{e^{-t}-1}{t}\, dt + \int_1^{+\infty} \frac{e^{-t}}{t}\, dt \\ &= \int_0^1 \frac{e^{-t}-1}{t}\, dt + \int_1^{+\infty} \frac{e^{-t}}{t}\, dt - \int_0^x \frac{e^{-t}-1}{t}\, dt \end{aligned}$$

Que vaut la constante du membre de droite :

$$C = \int_0^1 (e^{-t}-1)\frac{1}{t}\, dt + \int_1^{+\infty} e^{-t}\frac{1}{t}\, dt$$

Il se trouve que $C = -\gamma$ (voir plus bas une démonstration condensée) et donc :

$$\gamma = \int_0^x \frac{1-e^{-t}}{t}\, dt - f(x) - \ln(x) \tag{13.3}$$

Pour obtenir une valeur approchée de γ, il suffit donc de prendre un x assez grand pour pouvoir calculer $f(x)$ par son développement asymptotique à la précision requise ($f(x)S(x) + R(x)$ avec $S(x) = e^{-x}(\frac{1}{x} - \frac{1}{x^2} + \frac{2}{x^3} + ... + \frac{(-1)^n n!}{x^{n+1}})$ et $R(x) = -\int_x^{+\infty} \frac{(-1)^n n! e^{-t}}{t^{n+1}}\, dt$ et $|R(x)| \leq \frac{n! e^{-x}}{x^{n+1}}$), puis de calculer l'intégrale du membre de droite par le développement en séries en $x = 0$ (en utilisant une précision intermédiaire plus grande puisque ce développement en séries va sembler diverger au début avant de converger pour n suffisamment grand).

Exemple1 : on pose $x = 13$.
On calcule $f(13)$ par (13.1) avec $n = 13$ et une erreur absolue inférieure à $e^{-13}13!/13^{14} \leq 3.6e-12$.
On a en effet pour $x = x_0$ si $v_n = e^{-x_0}\frac{n!}{x_0^{n+1}}$:
$\frac{v_n}{v_{n-1}} \leq 1$ équivalent à $\frac{n}{x_0} \leq 1$équivalent à $n \leq x_0$.
Donc si $x = x_0 = 13$ on calcule $f(13)$ avec :
$f(13) \simeq \sum_{n=0}^{13} e^{-13}\frac{(-1)^n n!}{x^{n+1}}$ Ou bien, on tape :
`Digits:=2` ; puis
`exp(-13)*n!/13.^(n+1))$(n=0..20)` renvoie :
`1.7e-07,1.3e-08,2.1e-09,4.7e-10,1.5e-10,5.6e-11,2.6e-11,`
`1.4e-11,8.6e-12,5.9e-12,4.6e-12,3.9e-12,3.6e-12, 3.6e-12,`
`3.8e-12,4.4e-12,5.5e-12,7.1e-12,9.9e-12,1.4e-11,2.2e-11`
donc

$$f(13) \approx \texttt{exp(-13)*sum((-1)^n*n!/13.^(n+1),n=0..13)}$$

puis on remplace dans (13.3), avec

$$\int_0^x \frac{1-e^{-t}}{t}\, dt = \sum_{n=0}^{\infty}(-1)^n \frac{x^{n+1}}{(n+1)(n+1)!}$$

dont on obtient une valeur approchée, en faisant la somme jusqu'au rang 49, le reste de cette somme R_{50} est positif et est inférieur à `13.^51/51/51!)` qui est de l'ordre de `8.2e-12`. On a en effet si $v_n = \frac{13^{n+1}}{(n+1)(n+1)!}$:
$|R_N| = \sum_{n=N+1}^{\infty} v_n < v_{n+1} = \frac{13^{N+2}}{(N+2)(N+2)!}$ et $|R_{49}| < 8.2e-12$.

```
evalf(sum((-1)^n*13^(n+1)/(n+1)/(n+1)!,n=0..49))
```

La somme argument de `evalf` étant exacte, il n'y a pas de problèmes de perte de précision.
On obtient finalement comme valeur approchée de γ

```
-exp(-13)*sum((-1)^n*n!/13.^(n+1),n=0..13)-ln(13.)+
 evalf(  sum((-1)^n*13^(n+1)/(n+1)/(n+1)!,n=0..49))
```

On choisit alors 12 chiffres significatif et on tape :
`Digits:=12; f13:=exp(-13.)*evalf(sum((-1)^n*n!/13^(n+1),n=0..13))`
`I13:=evalf(sum((-1)^*13^(n+1)/(n+1)/(n+1)!,n=0..49))`
La constante d'Euler vaut donc à 1.2e-11 près :
`-f13-ln(13.)+I13`
On obtient :
`0.577215664897`
On tape :
`evalf(euler_gamma)`
On obtient :
`0.5772156649018`
soit `0.57721566489` avec une erreur inférieure à `1.2e-11`.

Exemple2 : on pose $x = 40$.
On tape :
`r40:=(exp(-40.)*40!/40.^41)` on obtient `r40` inférieur à 7.2e-36
On choisit alors 36 chiffres significatif et on tape :
`Digits:=36;`
`f40:=exp(-40.)*evalf(sum((-1)^n*n!/40^(n+1),n=0..40))`
puisque :
`40.^168/168./168!` est inférieur à 3.3e-36, on tape :
`I40:=evalf(sum((-1)^n*40^(n+1)/(n+1)/(n+1)!,n=0..166))`
La constante d'Euler vaut donc à (7.2+3.3)e-36 près :
`-f40-ln(40.)+I40`
On obtient avec une erreur inférieure à `1.1e-35` :
`0.577215664901532860606512090082402428 5`
On tape :
`evalf(euler_gamma)`
On obtient :
`0.577215664901532860606512090082402430 8`

Remarques La somme argument de `evalf` étant exacte, il n'y a pas de problèmes de perte de précision, on peut aussi faire les calculs intermédiaires en arithmétique approchée, lorsque $x = 13$ on doit alors prendre 4 chiffres significatifs de plus (pour tenir compte de la valeur du plus grand terme sommé dans la série

$v_n = \frac{13^{n+1}}{(n+1)(n+1)!}$ qui est v_{10}=`13^11/11/11!\simeq 4.08e+03`).
On a en effet :
$\frac{v_n}{v_{n-1}} = \frac{13*n}{(n+1)^2} > 1$ si $n^2 + 11n + 1 > 0$ i.e. $n \leq 10$ et
$\frac{v_n}{v_{n-1}} = \frac{13*n}{(n+1)^2} < 1$ si $n \geq 11$
On tape avec des calculs intermédiaires en arithmétique approchée :

```
Digits:=16; sum((-1)^n*13.^(n+1)/(n+1)/(n+1)!,n=0..49)
```

On obtient dans ce cas comme valeur approchée de γ : `0.57721566489675213`
Bien entendu, cette méthode avec des calculs intermédiaires en arithmétique approchée est surtout intéressante si on veut calculer un grand nombre de décimales de la constante d'Euler c'est à dire quand on prend `x=x0` tres grand, sinon on peut par exemple appliquer la méthode d'accélération de Richardson (cf 15.2.3) à la suite convergente (13.2) qui définit γ.

On peut calculer π de la même manière avec le développement en séries et asymptotique de la fonction sinus intégral (on remplace exponentielle par sinus dans la définition de f, voir plus bas une démonstration condensée) et l'égalité

$$\int_0^{+\infty} \frac{\sin(t)}{t}\, dt = \frac{\pi}{2} \tag{13.4}$$

Calcul de C (et preuve de (13.4)) :
Pour cela on effectue une intégration par parties, cette fois en intégrant $1/t$ et en dérivant l'exponentielle (moins 1 dans la première intégrale).

$$\begin{aligned} C &= \int_0^1 (e^{-t}-1)\frac{1}{t}\, dt + \int_1^{+\infty} e^{-t}\frac{1}{t}\, dt \\ &= [(e^{-t}-1)\ln(t)]_0^1 + \int_0^1 \ln(t)e^{-t}\, dt + [e^{-t}\ln(t)]_1^{+\infty} + \int_1^{+\infty} \ln(t)e^{-t}\, dt \\ &= \int_0^{+\infty} \ln(t)e^{-t}\, dt \end{aligned}$$

Pour calculer cette intégrale, on utilise l'égalité (qui se démontre par récurrence en faisant une intégration par parties) :

$$n! = \int_0^{+\infty} t^n e^{-t}\, dt$$

On va à nouveau intégrer par parties, on intègre un facteur 1 et on dérive l'intégrand, on simplifie, puis on intègre t et on dérive l'autre terme, puis $t^2/2$, etc.

$$\begin{aligned}
C &= [te^{-t}\ln(t)]_0^{+\infty} - \int_0^{+\infty} te^{-t}(\frac{1}{t} - \ln(t))\,dt \\
&= 0 - \int_0^{+\infty} e^{-t}\,dt + \int_0^{+\infty} te^{-t}\ln(t)\,dt \\
&= -1 + [\frac{t^2}{2}e^{-t}\ln(t)]_0^{+\infty} - \int_0^{+\infty} \frac{t^2}{2}e^{-t}(\frac{1}{t} - \ln(t))\,dt \\
&= -1 - \int_0^{+\infty} \frac{t}{2}e^{-t} + \int_0^{+\infty} \frac{t^2}{2}e^{-t}\ln(t)\,dt \\
&= -1 - \frac{1}{2} + \int_0^{+\infty} \frac{t^2}{2}e^{-t}\ln(t)\,dt \\
&= ... \\
&= -1 - \frac{1}{2} - ... - \frac{1}{n} + \int_0^{+\infty} \frac{t^n}{n!}e^{-t}\ln(t)\,dt \\
&= -1 - \frac{1}{2} - ... - \frac{1}{n} + \ln(n) + I_n
\end{aligned}$$

où

$$I_n = \int_0^{+\infty} \frac{t^n}{n!}e^{-t}(\ln(t) - \ln(n))\,dt$$

Pour déterminer I_n on fait le changement de variables $t = nu$

$$\begin{aligned}
I_n &= \int_0^{+\infty} \frac{(nu)^n}{n!}e^{-nu}\ln(u)n\,du \\
&= \frac{n^{n+1}}{n!}\int_0^{+\infty} e^{n(ln(u)-u)}\ln(u)\,du
\end{aligned}$$

Or en faisant le même changement de variables $t = nu$:

$$n! = \int_0^{+\infty} t^n e^{-t}\,dt = n^{n+1}\int_0^{+\infty} e^{n(ln(u)-u)}\,du$$

Donc

$$I_n = \frac{\int_0^{+\infty} e^{n(ln(u)-u)}\ln(u)\,du}{\int_0^{+\infty} e^{n(ln(u)-u)}\,du}$$

Lorsque n tend vers l'infini, on peut montrer que $I_n \to 0$, en effet les intégrales sont équivalentes à leur valeur sur un petit intervalle autour de $u = 1$, point où l'argument de l'exponentielle est maximal, et comme l'intégrand du numérateur a une amplitude $\ln(u)$ qui s'annule en $u = 1$, il devient négligeable devant le dénominateur. Finalement on a bien $C = -\gamma$.
On peut remarquer qu'en faisant le même calcul que C mais en remplacant e^{-t} par $e^{-\alpha t}$ pour $\Re(\alpha) > 0$, donne $\lim I_n = -\ln(\alpha)$ (car le point critique où la dérivée de la phase s'annule est alors $\frac{1}{\alpha}$). Ceci peut aussi se vérifier pour α réel en faisant le changement de variables $\alpha t = u$

$$\int_0^1 (e^{-\alpha t} - 1)\frac{1}{t}\,dt + \int_1^{+\infty} e^{-\alpha t}\frac{1}{t}\,dt = -\gamma - \ln(\alpha)$$

En faisant tendre α vers $-i$, $-\ln(\alpha)$ tend vers $\ln(i) = i\frac{\pi}{2}$ et on obtient

$$\int_0^1 (e^{it}-1)\frac{1}{t}\,dt + \int_1^{+\infty} e^{it}\frac{1}{t}\,dt = -\gamma + i\frac{\pi}{2}$$

dont la partie imaginaire nous donne (13.4), et la partie réelle une autre identité sur γ faisant intervenir la fonction cosinus intégral.

13.9 Solution de $f(x) = 0$ par la méthode de Newton

Dans `Xcas`, il existe déjà une fonction qui calcule la valeur approchée r d'une solution de $f(x) = 0$ par la méthode de Newton, qui est : `newton`.

13.9.1 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 :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

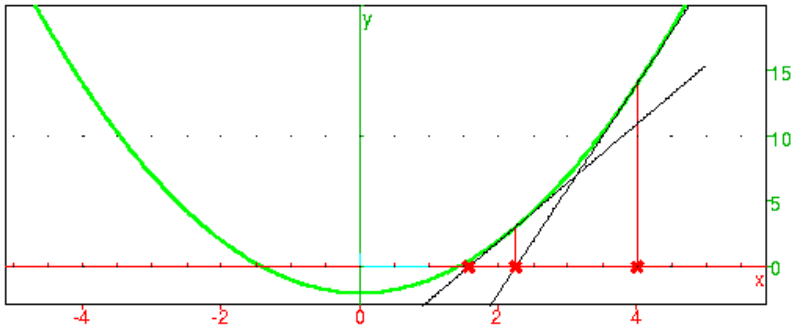
On peut alors réitérer le processus, et on obtient une suite x_n qui converge vers r soit par valeurs supérieures, si $f' * f'' > 0$ sur $[a\ ;\ b]$ (i.e. si $f'(r) > 0$ et si f est convexe ($f'' > 0$ sur $[a\ ;\ b]$) ou si $f'(r) < 0$ et si f est concave ($f'' < 0$ sur $[a\ ;\ b]$)) soit par valeurs 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 $[a\ ;\ b]$)).
On fait le dessin en tapant :

```
f(x):=x*x-2;
x0:=5/2;
G:=plotfunc(f(x));
T0:=tangent(G,x0);
Ox:=droite(0,1);
M1:=inter(T0,Ox)[0];
x1:=affixe(M1)
segment(point(x1,0),point(x1,f(x1)));
T1:=tangent(G,x1);
M2:=inter(T1,droite(0,1))[0]
x2:=affixe(M2):
segment(point(x2,0),point(x2,f(x2)));
```

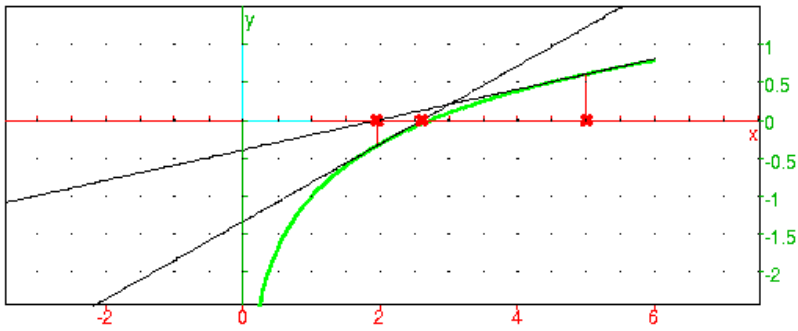
ou encore pour faire le dessin de la méthode de Newton pour la fonction f en partant du point de coordonnées $(a, f(a))$ et obtenir p nouveaux points.

```
plotnewton(f,a,p):={
local L,P,m,j,b;
L:=plotfunc(f(x),x,affichage=vert);
L:=L,point(a,couleur=point_width_4+rouge);
for (j:=1;j<=p;j++) {
b:=f(a);
L:=L,segment(a,a+i*b,couleur=ligne_tiret+rouge);
m:=function_diff(f)(a);
L:=L,plotfunc(b+(x-a)*m,x);
if (m==0){return "pente nulle"}
a:=a-f(a)/m;
P:=point(a,couleur=point_width_4+rouge);
L:=L,P;
}
return affixe(P),L;
};
```

On tape : `plotnewton(sq-2,4,2)`
pour obtenir les termes x_0, x_1, x_2 de la suite de Newton qui converge vers $\sqrt{2}$ et où $x_0 = 4$:



On remarquera que : `plotnewton(sq-2,4,2)[0]` renvoie :
`113/72` $\simeq$ `1.56944444444`
On tape : `plotnewton(ln-1,5,2)`
pour obtenir les termes x_0, x_1, x_2 de la suite de Newton qui converge vers e et où $x_0 = 5$:



On peut aussi faire une animation, pour cela, on tape :

```
newtonsuite(f,a,p):={
```

```
local L,P,m,j,b,LT;
P:=point(a,couleur=point_width_4+rouge);
LT:=P;
f1:=function_diff(f);
for (j:=1;j<=p;j++) {
b:=f(a);
L:=L,segment(a,a+i*b,couleur=ligne_tiret+rouge);
m:=f1(a);
L:=L,plotfunc(b+(x-a)*m,x);
if (m==0){return "pente nulle"}
a:=a-f(a)/m;
P:=point(a,couleur=point_width_4+rouge);
LT:=LT,[LT,L,P];
}
print(affixe(P));
return LT;
};
animnewton(f,a,p):={
local LT;
LT:=newtonsuite(f,a,p);
return plotfunc(f(x),x,affichage=vert),animation(LT);
};
```

On tape : `animnewton(sq-2,4,3)`

Puis, on écrit la fonction `newton_rac` qui renvoie la valeur approchée à eps 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-f(x0)/g(x0))`
car si on ne met pas `evalf`, les calculs de l'itération se feront excactement et seront vite compliqués.

```
newton_rac(f,x0,eps):={
local x1,h,g;
g:=function\_diff(f)
x0:=evalf(x0-f(x0)/g(x0));
x1:=x0-f(x0)/g(x0);
if (x1>x0) {h:=eps;} else {h:=-eps;}
while (f(x1)*f(x1+h)>0){
   x1:=x1-f(x1)/g(x1);
}
return x1;
}
```

13.9.2 Exercices sur la méthode de Newton

1. **L'énoncé**

(a) Étudier rapidement les variations de $f(x) = x\exp(x) + 0.2$ pour montrer que l'équation $f(x) = 0$ a deux solutions a et b qui vérifient $a < -1 < b < 0$

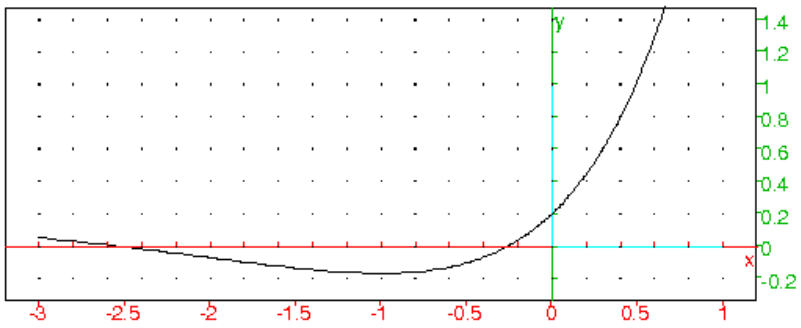
(b) Calculer à l'aide d'un programme par la méthode de Newton, les valeurs approchées de a et de b obtenues après 5 itérations.

(c) Modifier votre programme pour avoir des valeurs de a et b avec une précision de eps (par exemple de $1e-6$).

(d) Écrire un programme qui dessine le graphe de la fonction implicite $x\exp(x) + y\exp(y) = 0$ lorsque $x \geq -5$.

La solution avec `Xcas`

(a) Pour avoir les variations de $f(x) = x\exp(x) + 0.2$ on peut calculer la dérivée et faire le graphe de f.
On tape : `f(x):=x*exp(x)+0.2`
`factor(diff(f(x)))`
On obtient : `(x+1)*exp(x)`
On tape : `G:=plotfunc(f(x),x=-3..1)`
On obtient :



Pour montrer que l'équation $f(x) = 0$ a deux solutions a et b qui vérifient $a < -1 < b < 0$, on calcule, d'après le graphe $f(-3), f(-2), f(-1), f(0)$.
On tape : `f(x)$(x=-3..0)`
On obtient :
`0.0506387948964,-0.0706705664732,-0.167879441171,0.2`
donc puisque f est continue, d'après le théorème des valeurs intermédiaires on a : $-3 < a < -2$ et $-1 < b < 0$

(b) La méthode de Newton consiste à itérer la fonction h définie par $h(x) = x - f(x)/g(x)$ où g est la derivée de f. Pour trouver a, on va commencer en $x_0 = -2$ (car la fonction est concave et décroissante sur $[-3; -2]$ et les x_n seront des valeurs approchées de a par excés) et pour trouver b, on va commencer en $x_0 = 0$ (car la fonction est convexe et croissante sur $[-1; 0]$ et les x_n seront des valeurs approchées de b par excés). On tape :

```
Newtonvaleur(x0):={
  local j,f,g,h;
  f(x):=x*exp(x)+0.2;
```

```
  g(x):=(x+1)*exp(x);
  h(x):=x-f(x)/g(x);
  pour j de 1 jusque 5 faire
    x0:=h(x0);
  fpour;
  retourne x0;
};
```

Remarque

On peut remplacer `g(x):=(x+1)*exp(x);` par `g:=function_diff(f);`

On tape pour avoir la valeur de x_5 qui approche a :

```
Newtonvaleur(-2.)
```

On obtient :

```
-2.54264135777
```

On tape pour avoir la valeur de x_5 qui approche b :

```
Newtonvaleur(0)
```

On obtient :

```
-0.259171101819
```

(c) Si on veut avoir une valeur approchée de a (resp b) à eps prés, il faut avoir un x_j qui vérifie $x_j - eps \leq a < x_j$ (resp $x_j - eps \leq b < x_j$) c'est à dire $f(x_j - eps) > 0$ (resp $f(x_j - eps) < 0$) i.e. on doit avoir dans les 2 cas, $f(x_j - eps) * f(x0) \leq 0$.

On tape :

```
Newtonvalpres(x0,eps):={
  local j,g,h,t,s;
  f(x):=x*exp(x)+0.2;
  g(x):=(x+1)*exp(x);
  h(x):=x-f(x)/g(x);
  j:=0;
  t:=x0-eps;
  //s:=ifte(f(x0)>0,1,-1);
  s:=sign(f(x0));
  tantque s*f(t)>0 faire
    x0:=h(x0);
    t:=x0-eps;
    j:=j+1;
  ftantque;
  print(j);
  retourne t,x0;
};
```

On tape pour avoir la valeur de x_j qui donne un encadrement de a à $1e-6$ prés :

```
Newtonvalpres(-2.,1e-6)
```

On obtient pour $j = 3$:

```
-2.54264235686,-2.54264135686
```

On tape pour avoir la valeur de x_j qui donne un encadrement de a à $1e-10$ prés :

```
Newtonvalpres(-2.,1e-10)
```

On obtient pour $j = 4$:
`-2.54264135787,-2.54264135777`
On tape pour avoir la valeur de x_j qui donne un encadrement de b à $1e-6$ prés :
`Newtonvalpres(0,1e-6)`
On obtient pour $j = 4$:
`-0.259172101477,-0.259171101477` On tape pour avoir la valeur de x_j qui donne un encadrement de b à $1e-10$ prés :
`Newtonvalpres(0,1e-10)`
On obtient pour $j = 5$:
`-0.259171101919,-0.259171101819`

(d) $y\exp(y) + a = 0$ a une solution si $-\exp(-1) + a \leq 0$ i.e si $a < 1/e$
Cette fonction est définie pour des x tels que $x\exp(x) = a < 1/e$. On résout donc l'équation $x\exp(x) - 1/e = 0$:
On modifie le programme en :

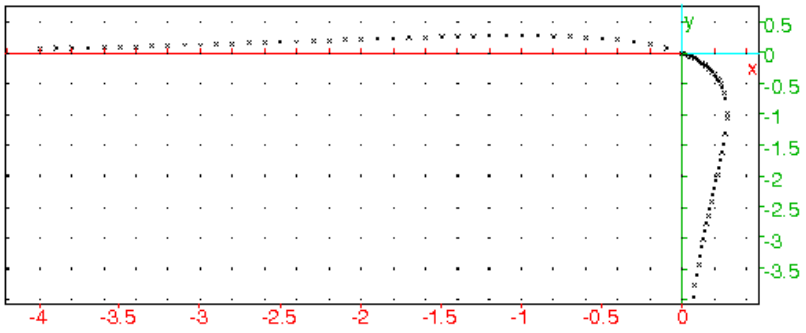
```
Newtonvaleura(x0,a):={
  local j,f,g,h;
  f(x,a):=x*exp(x)+a;
  g(x):=(x+1)*exp(x);
  h(x):=x-f(x,a)/g(x);
  pour j de 1 jusque 5 faire
    x0:=h(x0);
  fpour;
  retourne x0;
}:;
```

lorsque $a = -1/e$, on a $f(0,a) = a < 0$ et $f(1,a) = e - 1/e > 0$. On tape :
`Newtonvaleura(1,-1./e)`
On obtient :
`0.278464542823`
Pour $x \leq 0$, $a = x * \exp(x) \leq 0$ donc l'équation en y, $y * \exp(y) + x * \exp(x) = 0$ a une seule solution positive ($y * \exp(y) + a$ vaut $a \leq 0$ pour $y = 0$ et vaut $(x(exp(x)^2 - 1)/(exp(x)) > 0$ pour $y = -x$). On peut donc l'obtenir par la méthode de Newton : on démarre avec $y_0 = -a$ car pour $x > 0$ la courbe de $f(x,a) = x * \exp(x) + a$ se trouve au dessus de sa tangente en $x = 0$ (puisque $f"(x) > 0$ pour $x > 0$) et que cette tangente d'équation $y = x + a$ coupe l'axe des x en $x = -a$.
Pour $0 < x < 0.278464542823$ l'équation en y, $y * \exp(y) + x * \exp(x) = 0$ a deux solutions que l'on peut obtenir par la méthode de Newton : on démarre soit par $x_0 = 0$ soit par $x_0 = -2$.
On tape (on choisit de prendre x_5 comme valeur approchée) :

```
Newtonimplicit():={
  local j,f,g,h,a,xj,y0,y,L;
  g(y):=(y+1)*exp(y);
  f(y,a):=y*exp(y)+a;
```

```
  pour xj de -4 jusque 0 pas 0.1 faire
  a:=xj*exp(xj);
  h(y):=y-f(y,a)/g(y);
  y0:=-a;
  pour j de 1 jusque 5 faire
    y0:=h(y0);
  fpour;
  L:=L,point(xj+i*y0);
fpour;
L:=L,point(0.28-i);
pour xj de 0.01 jusque 0.28 pas 0.02 faire
  a:=xj*exp(xj);
  h(y):=y-f(y,a)/g(y);
  y0:=0;
  pour j de 1 jusque 5 faire
    y0:=h(y0);
  fpour;
  L:=L,point(xj+i*y0);
  y0:=-2;
  pour j de 1 jusque 5 faire
    y0:=h(y0);
  fpour;
  L:=L,point(xj+i*y0)
fpour;
retourne L;
}:;
```

On tape `Newtonimplicit()`
On obtient :



On peut aussi vouloir calculer y à eps -près. Mais attention lorsqu'on part de $y0 = -2$ on obtient une valeur soit par défaut, soit par excés selon le signe de $f(-2, a)$ (si $f(-2, a) > 0$ ce sera par excès car pour $x = -2$ on a un point d'inflexion).
On tape alors :

```
Newtonimpl(eps):={
  local j,f,g,h,a,xj,y0,y,t,s,L;
  g(y):=(y+1)*exp(y);
```

```
  f(y,a):=y*exp(y)+a;
  L:=NULL;
  pour xj de -5 jusque 0 pas 0.05 faire
  a:=evalf(xj*exp(xj));
  h(y):=y-f(y,a)/g(y);
  y0:=-a;
  t:=y0-eps;
  s:=sign(f(y0,a));
  tantque s*f(t,a)>0 faire
    y0:=h(y0);
    t:=y0-eps;
  ftantque;
  L:=L,point(xj+i*y0);
fpour;
L:=L,point(0.28-i);
pour xj de 0.01 jusque 0.28 pas 0.02 faire
  a:=evalf(xj*exp(xj));
  h(y):=y-f(y,a)/g(y);
  y0:=0.;
  t:=y0-eps;
  s:=sign(f(y0,a));
  tantque s*f(t,a)>0 faire
    y0:=h(y0);
    t:=y0-eps;
  ftantque;
  L:=L,point(xj+i*y0);
fpour;
  pour xj de 0.01 jusque 0.28 pas 0.02 faire
    a:=evalf(xj*exp(xj));
  h(y):=y-f(y,a)/g(y);
  y0:=-2.;
  s:=sign(f(y0,a));
  si s>0  alors eps:=-abs(eps); fsi;
  t:=y0-eps;
  tantque s*f(t,a)>0 faire
    y0:=h(y0);
    t:=y0-eps;
  ftantque;
  L:=L,point(xj+i*y0)
fpour;
retourne L;
}:;
```

On tape `Newtonimpl(0.01)`

On peut vérifier en tapant :

```
plotimplicit(x*exp(x)+y*exp(y),[x,y])
```

2. **L'énoncé**

Donner la valeur approchée de $\cos(x) = x$ pour $x \in [0;1]$ obtenue en

partant de $x_0 = 0$ après 4, puis après 10 itérations lorsque :

(a) On applique la méthode du point fixe à $f(x) = \cos(x)$.

(b) On applique la méthode de Newton.

(c) On applique la méthode du Δ^2 d'Aitken.

(d) On applique la méthode de Steffensen.

Quelle méthode vous semble la meilleure ? Expliquez pourquoi.

La solution avec `Xcas`

On configure `Xcas` avec 20 digits.

(a) fa fonction $f(x) = \cos(x)$ est $\sin(1)$-contractante sur [0 ;1], car d'après le théorème des accroissements finis :
il existe $c \in [0; 1]$ tel que pour tout $x_1 \in [0; 1]$ et tout $x_2 \in [0; 1]$ on a $\cos(x_1) - \cos(x_2) = (x_1 - x_2)\sin(c)$ donc
$|\cos(x_1) - \cos(x_2)| = |x_1 - x_2||\sin(c)| \leq |x_1 - x_2|\sin(1)$.
On tape :

```
ptfixecos(x0,n):={
local j,f;
f(x):=cos(x);
pour j de 1 jusque n faire
  x0:=f(evalf(x0));
fpour;
retourne x0;
}:;
```

On tape : `ptfixecos(0,4)`
On obtient : `0.654289790497779149974`
On tape : `ptfixecos(0,10)`
On obtient : `0.731404042422509858293`

(b) On pose $F(x) = \cos(x) - x$ et on a $F'(x) = -\sin(x) - 1$, on va donc itérer la fonction $g(x) = x - F(x)/F'(x) = (x * \sin(x) + \cos(x))/(\sin(x) + 1)$.
On tape :

```
Newtoncos(x0,n):={
local j,g,F,dF;
F(x):=cos(x)-x;
dF:=function_diff(F);
//g(x):=(x*sin(x)+cos(x))/(sin(x)+1);
g(x):=normal(x-F(x)/dF(x));
pour j de 1 jusque n faire
  x0:=g(evalf(x0));
fpour;
retourne x0;
}:;
```

On tape : `Newtoncos(0,4)`
On obtient : `0.739085133385283969762`
On tape : `Newtoncos(0,10)`

On obtient : `0.739085133215160641651`

(c) La méthode du Δ^2 d'Aitken consiste à transformer la suite des itérées du point fixe par la fonction :
$gs(x) = x - (f(x) - x) * (f(x) - x)/(f(f(x)) - 2f(x) + x)$ avec $f(x) = \cos(x)$.
On tape :

```
Aitkencos(x0,n):={
local j,gs,f,y0;
f(x):=cos(x);
gs(x):=x-(f(x)-x)*(f(x)-x)/(f(f(x))-2f(x)+x);
pour j de 1 jusque n faire
  x0:=f(evalf(x0));
  y0:=gs(x0);
fpour;
print(x0);
retourne y0;
}:;
```

On tape : `Aitkencos(0,4)`
On obtient : `0.738050421371663847259`
On tape : `Aitkencos(0,10)`
On obtient : `0.739076383318955862683`

(d) La méthode de Steffenson consiste à itérer la fonction :
$gs(x) = x - (f(x) - x) * (f(x) - x)/(f(f(x)) - 2f(x) + x)$ avec $f(x) = \cos(x)$.
On tape :

```
Steffensencos(x0,n):={
local j,gs,f;
f(x):=cos(x);
gs(x):=x-(f(x)-x)*(f(x)-x)/(f(f(x))-2f(x)+x);
pour j de 1 jusque n faire
  x0:=gs(evalf(x0));
fpour;
retourne x0;
}:;
```

On tape : `Steffensencos(0,4)`
On obtient : `0.739085133215160534355`
On tape : `Steffensencos(0,10)`
On obtient : `0.739085133215160641651`

Les méthodes de Newton et de Steffensen sont plus performantes car ce sont des méthodes d'ordre 2 (la fonction que l'on iére a une dérivée nulle au point solution de $f(x) = \cos(x) = x$).
Même avec `Digits:=30` on a :
`Steffensencos(0,6)=Steffensencos(0,10)=`

```
Newtoncos(0,6)=Newtoncos(0,10)=
0.739085133215160641655312087673 5
```

3. **L'énoncé**
   Dans un problème de baccalauréat, on se propose de calculer des valeurs approchées des solutions de $\exp(-x)\cos(x) = x$ sur $[-\pi/2; \pi/2]$.
   (a) Déterminer le nombre de solutions.
   (b) Donner un algorithme de calcul et écrire le programme correspondant.
   (c) Donner un encadrement de la solution ?
   (d) Dessiner les points de coordonnées t, x qui vérifient :
   $\exp(-x)\cos(x) - x + t = 0$ pour $t \in [-\pi/2; \pi/2]$ et $x \in [-\pi/2; \pi/2]$

   La solution avec `Xcas`

   (a) On tape : `f(x:)=exp(-x)*cos(x)-x`
   `f1:=function_diff(f);f2:=normal(diff(f1(x)))` On trouve :
   `f1(x)=(-cos(x)-sin(x))*exp(-x)-1` et
   `f2=2*exp(-x)*sin(x)`
   On a :
   $f2 = 0$ en $x = 0$ le point(0,1) est donc un point d'inflexion et
   sur $[-\pi/2; 0]$, on a $f2 < 0$ et sur $[0; \pi/2]$, on a $f2 > 0$
   On a :
   $f1(x) = \exp(\pi/2) - 1 > 0$ pour $x = -\pi/2$
   $f1(0) = -2$
   $f1(x) = -\exp(-\pi/2) - 1 < 0$ pour $x = \pi/2$
   $f1$ s'annule donc pour $x = a < 0$ et donc f est croissante puis décroissante et comme on a $f(-pi/2) = pi/2$, $f(0) = 1$ et $f(pi/2) = -pi/2$ et f continue, f s'annule en un seul point $x = b > 0$ Pour vérifier, on tape :
   `G:=plotfunc(f(x),x=-pi/2..pi/2);tangente(G,0)`

   (b) On peut appliquer la méthode de Newton en partant de $x_0 = 0.0$, la suite $x_n = x_{n-1} - f(x_{n-1})/f1(x_{n-1})$ va donner une valeur approchée par défaut de b car sur $[0; b]$ f est positive décroissante et convexe.
   On tape :
   `Newton0(4)`
   On obtient avec 22 digits :
   `0.5177573636824582982 9471`

   (c) On tape :

```
Newton0(n):={
local j,f,f1,g,x0;
f(x):=exp(-x)*cos(x)-x;
f1:=function_diff(f);
g(x):=normal(x-f(x)/f1(x));
x0:=0.0;
pour j de 1 jusque n faire
```

```
x0:=g(x0)
fpour;
retourne x0;
}:;
```

(d) Pour avoir un encadrement de la solution à `eps` près, on continue l'itération tant que la valeur de $f(x_n + eps)$ reste strictement positive.
On tape

```
Newtoneps(n,eps):={
  local j,f,f1,g,x0;
  f(x):=exp(-x)*cos(x)-x;
  f1:=function_diff(f);
  g(x):=normal(x-f(x)/f1(x));
  x0:=0.0;
  j:=0;
  tantque f(x0+eps)>0 faire
    x0:=g(x0);
    j:=j+1;
  ftantque;
  print(j);
  retourne x0,x0+eps;
}
:;
```

On tape :

```
Newtoneps(0,1e-20)
```

On obtient :

```
0.5177573636824582983277,0.5177573636824582983327
```

(e) Il faut voir que si on remplace $f(x)$ par $f(x)+t$ la suite définie par :$x_0 = 0$ et $x_{n+} = x_n - (f(x_n) + t)/f1(x_n)$ approche la solution en x de $f(x)+t = 0$ par excès si $t < -1$ et par défaut si $t > -1$ car pour $x = 0$ la fonction $f(x)+t$ a un point d'inflexion qui est le point $(0; 1+t)$. De plus $f(x) + t > 1 + t$ pour $x < 0$ et $f(x) + t < 1 + t$ pour $x > 0$. Donc si $1 + t > 0$ la solution sera positive et si $1 + t < 0$ la solution sera negative
On tape :

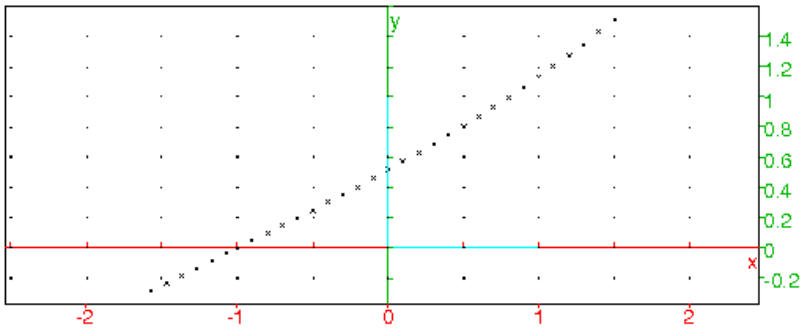
```
Newtonimpl():={
  local j,f,f1,g,x0,t,a,L;
  a:=evalf(pi/2);
  f(x):=exp(-x)*cos(x)-x;
  f1:=function_diff(f);
  L:=NULL;
    pour t de -a jusque -1 pas 0.1 faire
      g(x):=normal(x-(f(x)+t)/f1(x));
      x0:=0.0;
      tantque f(x0-0.01)+t<0 faire
        x0:=g(x0);
      ftantque;
```

```
    L:=L,point(t,x0);
  fpour;
  pour t de -1 jusque a pas 0.1 faire
    g(x):=normal(x-(f(x)+t)/f1(x));
    x0:=0.0;
    tantque f(x0+0.01)+t>0 faire
      x0:=g(x0);
    ftantque;
    L:=L,point(t,x0);
  fpour;
  return L;
 }
:;
```

On tape :

```
Newtonimpl()
```

On obtient :



4. Écrire un programme qui calcule $a^{\frac{1}{5}}$ $a^{\frac{1}{5}}$ est solution de $f(x) = x^5 - a = 0$.
On a $f'(x) = 5x^4$. La méthode de Newton dit que la suite définie par :
$u_0 = u0$ et

$$u_{n+1} = u_n - \frac{f(u_n}{f'(u_n)} = u_n - \frac{u_n^5 - a}{4u_n^4} = \frac{4}{5}u_n + \frac{a}{5 * u_n^4} \text{ pour } n \geq 0$$

converge vers $a^{\frac{1}{5}}$.
On va donc écrire un programme, avec comme paramètre a, $u_0 = u0$ et n effectuant le calcul de u_n qui est la suite définie par :
$u_0 = u0$ et $u_{n+1} = \frac{4}{5}u + \frac{a}{5 * u^4}$ pour $n \geq 0$.
On tape :

```
sqrt5(a,u0,n):={
 local u,j;
 u:=u0;
 pour j de 1 jusque n faire
  u:=4*u/5+a/(5*u^4);
 fpour;
 return evalf(u);
 }:;
```

On tape :

```
evalf(2^(1/5))
```
On obtient :
```
1.1486983549970350068
```
On tape :
```
sqrt5(2,2,6),sqrt5(2,2,8)
```
On obtient :
```
1.1486983577671234219,1.1486983549970350068
```
On tape :
```
evalf(5^(1/5))
```
On obtient :
```
1.3797296614612148324
```
On tape :
```
sqrt5(5,5,6),sqrt5(5,5,8)
```
On obtient :
```
1.4895125973959737840,1.3800502625136162230
```
On tape :
```
sqrt5(5,2,6),sqrt5(5,2,8)
```
On obtient :
```
1.3797296614612151922,1.3797296614612148324
```

13.9.3 La méthode de Newton avec préfacteur

Lorsqu'on part d'une valeur x_0 trop éloignée de la racine de $f(x)$ (si par exemple $|f(x_0)|$ est grand), on a intérêt à utiliser un préfacteur pour se rapprocher plus vite de la solution de $f(x) = 0$.
Posons $n(x) = -\dfrac{f(x)}{f'(x)}$, on a alors :

$$\lim_{h\to 0} \frac{(f(x_0 + h * n(x_0)) - f(x_0))}{h * n(x_0)} = f'(x_0)$$

donc

$$\lim_{h\to 0} \frac{(f(x_0 + h * n(x_0)) - f(x_0))}{h} = n(x_0) * f'(x_0) = -f(x_0)$$

ce qui veut dire que :
$f(x_0 + h * n(x_0)) = f(x_0)(1 - h) + h \cdot \epsilon(h)$ avec $\epsilon(h)$ tend vers 0 quand h tend vers 0.
Donc, il existe h_0 vérifiant :
$|f(x_0 + h_0 * n(x_0))| < |f(x_0)|$
Remarque : Il faut minimiser $|f(x_0 + h_0 * n(x_0))|$. or plus h_0 est proche de 1 et plus $|f(x_0) * (1 - h_0)|$ sera petit. Par exemple, on prendra le plus grand h_0, dans la liste $[1, 3/4, (3/4)^2, ...]$ qui vérifie $|f(x_0 + h_0 * n(x_0))| < |f(x_0)|$
Pour cette valeur de h_0, $x_0 + h_0 * n(x_0)$ est probablement plus proche de la racine que x_0 : on dit que h_0 est le préfacteur de la méthode de Newton.
On va choisir par exemple au début $h_0 = 1$, et on regarde si $|f(x_0 + n(x_0))| < |f(x_0)|$, si ce n'est pas le cas on prend $h_0 = (3/4)$ et on regarde si $|f(x_0 + 3/4 * n(x_0))| < |f(x_0)|$, si ce n'est pas le cas on prend $h_0 = (3/4)^2$ etc...

On change de préfacteurs à chaque étape jusqu'à ce que : $abs(f(x1))-abs(f(x0)) < 0$ sans préfacteur, on continue alors l'itération sans préfacteur, c'est à dire avec la méthode de Newton normale. On écrit donc :

```
newton_prefacts(f,x0,eps):={
local x1,h,h0,prefact,niter;
//prefact est egal par ex a 3/4
h0:=1.0;
niter:=0;
prefact:=0.75;
x1:=x0-h0*f(x0)/function_diff(f)(x0);
while (abs(f(x1))-abs(f(x0))>0) {
   h0:=h0*prefact;
   x1:=x0-h0*f(x0)/function_diff(f)(x0);
}
h:=eps;
while (h0!=1 and niter<100){
   x0:=x1;
   x1:=x1-h0*f(x1)/function_diff(f)(x1);
   while (abs(f(x1))-abs(f(x0))>0) {
      h0:=h0*prefact;
      x1:=x0-h0*f(x0)/function_diff(f)(x0);
   }
   while (abs(f(x1))-abs(f(x0)<0 and h0!=1)) {
      h0:=h0/prefact;
      x1:=x0-h0*f(x0)/function_diff(f)(x0);
   }
niter:=niter+1;
}
while (f(x1-h)*f(x1+h)>0  and niter<200){
   x0:=x1;
   x1:=x1-f(x1)/function_diff(f)(x1);
   niter:=niter+1;
}

if  (niter<200) {return x1;} else {return "pas trouve";}
}
```

On définit la fonction f par $\mathtt{f(x)} := \mathtt{x}^2 - 2$ et on met ce programme dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`), puis on le teste et on le valide avec `OK`.
On tape :
`newton_prefacts(f,100,1e-10)`
On obtient :
`1.41421356237` On tape :
`newton_prefacts(f,3,1e-5)`
On obtient :
`1.41421378005`

13.10 Trouver un encadrement de $x0$ lorsque $f(x0)$ est minimum

Soit f une fonction définie sur $[a;b]$. On suppose que f est unimodale sur $[a;b]$, c'est à dite que f a un seul extremum sur $[a;b]$. On suppose de plus que cet extremum est un minimum (sinon on remplacera f par $-f$.) On se propose de trouver un encadrement à eps près de la valeur pour laquelle f est minimum.

13.10.1 Déscription du principe de la méthode

On partage $[a;b]$ en trois morceaux en considérant c et d vérifiant : $a < c < d < b$.
On calcule $f(c)$ et $f(d)$ et on les compare.
Puisque f a un seul minimum sur $[a;b]$ elle décroit, passe par son minimum, puis f croit. Selon les trois cas possibles on a :

- f(c)<f(d)
  dans ce cas, la valeur pour laquelle f est minimum n'appartient pas à $[d;b]$
- f(c)>f(d)
  dans ce cas, la valeur pour laquelle f est minimum n'appartient pas à $[a;c]$
- f(c)=f(d)
  dans ce cas, la valeur pour laquelle f est minimum n'appartient pas à $[d;b]$ ni à $[a;c]$

Ainsi, l'intervalle de recherche a diminué et on peut recommencer le processus. Pour que l'algorithme soit performant, on veut que l'intervalle de recherche diminue rapidement et que le nombre de valeurs de f à calculer soit le plus petit possible. Pour cela comment doit-on choisir c et d ?

13.10.2 Déscription de 2 méthodes

On fait presque une dichotomie

On choisit c et d proche de $\frac{a+b}{2}$ par exemple :
$c = \frac{a+b-eps}{2}$ et $d = \frac{a+b+eps}{2}$ pour eps donné. Dans ce cas, à chaque étape l'intervalle diminue presque de moitié mais on doit calculer, à chaque étape, deux valeurs de f.

On utilise la suite de Fibonacci

Comment faire pour que l'une des valeurs de f déjà calculée soit calculé à l'étape suivante ?
La solution se trouve dans la suite de Fibonacci, suite définie par :
$u_0 = 1$, $u_1 = 2$, $u_n = u_{n-2} + u_{n-1}$ dont les premiers termes sont :
$1, 2, 3, 5, 8, 13, 21, 34, 55, 89...$
Par exemple si on partage $[a;b]$ en 89 parties égales si $l = (b-a)/89$, on choisit $c = a + 34 * l$ et $d = a + 55 * l$ et ainsi on a :
$c - a = 34 * l$, $d - c = 21 * l$, $b - d = 34 * l$ (car 89=55+34 et 34+21=55 puisque 21,34,55,89 sont des termes consécutifs de la suite de Fibonacci).
On calcule $f(c)$ et $f(d)$ puis on réduit l'intervalle en un intervalle de longueur $(b-a) * 55/89$, par exemple si l'intervalle suivant est $[a;d]$ et, si on recommence

le processus, le point c est le futur point d.
Donc à chaque étape il suffit de calculer une seule valeur de f pour passer de l'intervalle $[a;b]$ (proportionnel à u_n) à l'intervalle $[a;d]$ ou $[c;b]$ (proportionnel à u_{n-1}). Il y a bien sûr le cas $f(c)=f(d)$ où il faut à l'étape suivante calculer deux valeurs de f, mais dans ce cas on gagne 3 étapes car on passe de l'intervalle $[a;b]$ (proportionnel à u_n) à l'intervalle $[c;d]$ (proportionnel à u_{n-3}).
Selon la valeur eps de la longueur de l'encadrement, on calcule $k := ceil(2*(b-a)/eps)$; et la première valeur $t=u_n$ de la suite de Fibonacci supérieure à k. il faut alors diviser l'intervalle $[a;b]$ en t parties égales. On applique alors plusieurs fois le processus et on s'arrête quand $n=1$, c'est à dire quand l'intervalle a été réduit à un intervalle de longueur $2*(b-a)/t$ qui est, grace au choix de t ($t>k>2*(b-a)/eps$) inférieur à eps.

13.10.3 Traduction `Xcas` de l'algorithme avec Fibonacci

```
//f(x):=2*x^4-10*x^3-4*x^2+100
//fibomin(f,1,5,0.000001)
//g(x):=2*x^4-10*x^3+4*x^2+100
//fibomin(g,1,5,1e-20)
//calcul la valeur du min d'une fonction ayant
//un seul extrema sur [a,b]
fibomin(f,a,b,eps):={
  local c,d,F,k,n,t,g,h,l,fc,fd;
  if (a>b) {c:=a;a:=b;b:=c;}
  k:=ceil(2*(b-a)/eps);
  F:=1,2;
  n:=1;
  g:=1;
  t:=2;
  //construction de F=la suite de Fibonacci
  //h,g,t sont 3 termes consecutifs de F
  while (t<k) {
    n:=n+1;
    h:=g;
    g:=t;
    t:=h+g;
    F:=F,t;
  }
  l:=(b-a)/t;
  c:=a+h*l;
  d:=a+g*l;
  fc:=f(c);
  fd:=f(d);
  //on itere le processus et on s'arrete qd n=1
  while (n>1) {
    if (fc>fd) {
      a:=a+h*l;
      fc:=fd;
```

```
      t:=h;
      h:=g-h;
      g:=t;
      fd:=f(a+g*l);
      n:=n-1;
    }else{
      if (fc<fd) {
b:=a+g*l;
t:=h;
h:=g-h;
g:=t;
fd:=fc;
fc:=f(a+h*l);
n:=n-1;
      }else{
a:=a+h*l;
b:=b-h*l;
t:=g-h;
g:=h-t;
h:=t-g;
fc:=f(a+h*l);
fd:=f(a+g*l);
n:=n-3;
      }
    }
  }
  return [a,b];
}
```

On tape :

```
f(x):=x^4-10
fibomin(f,-1,1,1e-10)
```

On obtient :

```
[(-1)/53316291173,1/53316291173]
```

On tape :

```
g(x):=2*x^4-10*x^3-4*x^2+100
fibomin(g,1,5,1e-10)
```

On obtient :

```
[86267571271/21566892818,86267571273/21566892818]
```

On tape :

```
h(x):=2*x^4-10*x^3+4*x^2+100
fibomin(h,1,5,1e-10)
```

On obtient :

```
[74644573011/21566892818,74644573013/21566892818]
```

Chapitre 14

Algorithmes d'algébre

14.1 Méthode pour résoudre des systèmes linéaires

Dans Xcas, il existe déjà les fonctions qui correspondent aux algorithmes qui suivent, ce sont : `ref, rref, ker, pivot`

14.1.1 Le pivot de Gauss quand A est de rang maximum

Étant donné un système d'équations noté $AX = b$, on cherche à le remplacer par un système équivalent et triangulaire inférieur.
À un système d'équations $AX = b$, on associe la matrice M formée par A que l'on borde avec b comme dernière colonne.
La méthode de Gauss (resp Gauss-Jordan) consiste à multiplier A et b (donc M) par des matrices inversibles, afin de rendre A triangulaire inférieure (resp diagonale). Cette transformation, qui se fera au coup par coup en traitant toutes les colonnes de A (donc toutes les colonnes de M, sauf la dernière), consiste par des combinaisons de lignes de M à mettre des zéros sous (resp de part et d'autre) la diagonale de A.
La fonction `gauss_redi` ci-dessous, transforme M par la méthode de Gauss, la variable `pivo` (car `pivot` est une fonction de Xcas) sert à mettre le pivot choisi.
Pour $j = 0..p-2$ ($p-2$ car on ne traite pas la dernière colonne de M), dans chaque colonne j, on cherche ce qui va faire office de pivot à partir de la diagonale : dans le programme ci-dessous on choisit le premier élément non nul, puis par un échange de lignes, on met le pivot sur la diagonale ($pivo = M[j, j]$). Il ne reste plus qu'à former, pour chaque ligne k ($k > j$) et pour $a = M[k, j]$, la combinaison :
$pivo * ligne_k - a * ligne_j$ (et ainsi $M[k, j]$ devient nul).
On écrit pour réaliser cette combinaison :

```
a:=M[k,j];
for (l:=j;l<nc;l++){
M[k,l]:=M[k,l]*pivo-M[j,l]*a;}
```

On remarquera qu'il suffit que l parte de j car :
pour tout $l < j$, on a déjà obtenu, par le traitement des colonnes $l = 0..j - 1$, $M[k, l] = 0$.
Le programme ci-dessous ne sera utile que si on trouve un pivot pour chaque colonne : c'est à dire si la matrice A est de rang maximum. En effet, dans ce programme, si on ne trouve pas de pivot (i. e. si tous les éléments d'une colonne sont nuls sur et sous la diagonale), on continue comme si de rien était...

```
gauss_redi(M):={
local pivo,j,k,nl,nc,temp,l,n,a;
nl:=nrows(M);
nc:=ncols(M);
n:=min(nl,nc-1);
//on met des 0 sous la diagonale du carre n*n
for (j:=0;j<n;j++) {
  //choix du pivot mis ds pivo
    k:=j;
    while (M[k,j]==0 and k<nl-1) {k:=k+1;}
    //on ne fait la suite que si on a pivo!=0
    if (M[k,j]!=0) {
      pivo:=M[k,j];
      //echange de la ligne j et de la ligne k
      for (l:=j;l<nc;l++){
       temp:=M[j,l];
       M[j,l] := M[k,l];
        M[k,l]:=temp;
      }
     //fin du choix du pivot qui est M[j,j]
      for (k:=j+1;k<nl;k++) {
        a:=M[k,j];
        for (l:=j;l<nc;l++){
          M[k,l]:=M[k,l]*pivo-M[j,l]*a;
         }
       }
    }
}
return M;
}
```

On tape :
`M0:= [[1,2,3,6],[2,3,1,6],[3,2,1,6]]`
`gauss_redi(M0)`
On obtient :
`[[1,2,3,6],[0,-1,-5,-6],[0,0,-12,-12]]`
On tape :
`M1:= [[1,2,3,4],[0,0,1,2],[0,0,5,1]]`
`gauss_redi(M1)`
On obtient :
`[[1,2,3,4],[0,0,1,2],[0,0,5,1]]`
On tape :
`M2:= [[1,2,3,4],[0,0,1,2],[0,0,5,1],[0,0,3,2],[0,0,-1,1]]`
`gauss_redi(M2)` On obtient :
`[[1,2,3,4],[0,0,1,2],[0,0,5,1],[0,0,0,7],[0,0,0,6]]`

c'est à dire :

$$\texttt{gauss\_redi}\left(\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 5 & 1 \\ 0 & 0 & 3 & 2 \\ 0 & 0 & -1 & 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

14.1.2 Le pivot de Gauss pour A quelconque

On cherche un programme valable lorsque A est quelconque : on veut, dans ce cas, mettre des zéros sous la "fausse diagonale" de A (ce qu'on appelle "fausse diagonale" c'est la diagonale obtenue en ne tenant pas compte des colonnes pour lesquelles la recherche du pivot a été infructueuse : un peu comme si ces colonnes étaient rejetées à la fin de la matrice).
Donc, dans ce programme, si on ne trouve pas de pivot pour la colonne d'indice jc (i. e. si tous les éléments de la colonne jc sont nuls sur et sous la diagonale), on continue en cherchant, pour la colonne suivante (celle d'indice $jc+1$), un pivot à partir de l'élément situé à la ligne d'indice jc (et non comme précédemment à partir de $jc+1$), pour mettre sur la colonne $jc+1$, des zéros sur les lignes $jc+1, ..., nl-1$. On est donc obligé, d'avoir 2 indices jl et jc pour repérer les indices de la "fausse diagonale".

```
gauss_red(M):={
local pivo,jc,jl,k,nl,nc,temp,l,a;
nl:=nrows(M);
nc:=ncols(M);
//on met des 0 sous la fausse diagonale d'indice jl,jc
jc:=0;
jl:=0;
//on traite chaque colonne (indice jc)
while (jc<nc-1 and jl<nl-1) {
  //choix du pivot que l'on veut mettre en M[jl,jc]
    k:=jl;
    while (M[k,jc]==0 and k<nl-1) {k:=k+1;}
    //on ne fait la suite que si M[k,jc](=pivo)!=0
    if (M[k,jc]!=0) {
      pivo:=M[k,jc];
      //echange de la ligne jl et de la ligne k
      for (l:=jc;l<nc;l++){
       temp:=M[jl,l];
       M[jl,l] := M[k,l];
        M[k,l]:=temp;
      }
     //fin du choix du pivot qui est M[jl,jc]
      //on met des 0 sous la fausse diagonale de
      //la colonne jc
      for (k:=jl+1;k<nl;k++) {
        a:=M[k,jc];
```

```
        for (l:=jc;l<nc;l++){
          M[k,l]:=M[k,l]*pivo-M[jl,l]*a;
         }
       }
      //on a 1 pivot,l'indice-ligne de
      //la fausse diag augmente de 1
      jl:=jl+1;
    }//fin du if (M[k,jc]!=0)
    //colonne suivante,l'indice-colonne de
    //la fausse diag augmente de 1
    jc:=jc+1;
}//fin du while
return M;
}
```

On tape :
`M0:= [[1,2,3,6],[2,3,1,6],[3,2,1,6]]`
`gauss_red(M0)`
On obtient :
`[[1,2,3,6],[0,-1,-5,-6],[0,0,-12,-12]]`
On tape :
`M1:= [[1,2,3,4],[0,0,1,2],[0,0,5,1]]`
`gauss_red(M1)`
On obtient :
`[[1,2,3,4],[0,0,1,2],[0,0,0,-9]]`
On tape :
`M2:= [[1,2,3,4],[0,0,1,2],[0,0,5,1],[0,0,3,2],[0,0,-1,1]]`
`gauss_red(M2)`
On obtient :
`[[1,2,3,4],[0,0,1,2],[0,0,0,-9],[0,0,0,-4],[0,0,0,3]]`

$$\texttt{gauss\_red}\left(\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 5 & 1 \\ 0 & 0 & 3 & 2 \\ 0 & 0 & -1 & 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & -9 \\ 0 & 0 & 0 & -4 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

14.1.3 La méthode de Gauss-Jordan

Étant donnée un système d'équations, on cherche à le remplacer par un système diagonale équivalent. À un système d'équations $AX = b$, on associe la matrice M formée de A, bordée par b comme dernière colonne.
La fonction `gaussjordan_redi` transforme M par la méthode de Gauss-Jordan. On cherche dans chaque colonne j,($j = 0..nc-2$) à partir de la diagonale, ce qui va faire office de pivot : dans le programme ci-dessous on choisit le premier élément non nul, puis par un échange de lignes, on met le pivot sur la diagonale ($pivo = M[j,j]$). Il ne reste plus qu'à former, pour chaque ligne k ($k \neq j$) et pour $a = M[k,j]$, la combinaison :
$pivo * ligne_k - a * ligne_j$ (et ainsi $M[k,j]$ devient nul).

On écrit pour réaliser cette combinaison :
lorsque $k < j$

```
a:=M[k,j];
for (l:=0;l<j;l++){
M[k,l]:=M[k,l]*pivo-M[j,l]*a;}
```

et lorsque $k > j$

```
a:=M[k,j];
for (l:=j;l<nc;l++){
M[k,l]:=M[k,l]*pivo-M[j,l]*a;}
```

On remarquera que l part soit de 0 soit de j car pour $l < j$, on a $M[k,l] = 0$ seulement si $k > j$.
Si on ne trouve pas de pivot, on continue comme si de rien n'était : on obtiendra donc des zéros au dessus de la diagonale que si on a trouvé un pivot pour chaque colonne.

```
gaussjordan_redi(M):={
local pivo,j,k,nl,nc,temp,l,n,a;
nl:=nrows(M);
nc:=ncols(M);
n:=min(nl,nc-1);
//on met 0 sous et au dessus de la diag du carre n*n
for (j:=0;j<n;j++) {
  //choix du pivot
    k:=j;
    while (M[k,j]==0 and k<nl-1) {k:=k+1;}
    //on ne fait la suite que si on a pivo!=0
    if (M[k,j]!=0) {
      pivo:=M[k,j];
      //echange de la ligne j et de la ligne k
      for (l:=j;l<nc;l++){
       temp:=M[j,l];
       M[j,l] := M[k,l];
        M[k,l]:=temp;
       }
     //fin du choix du pivot qui est M[j,j]
     // on met des zeros au dessus de la diag
     for (k:=0;k<j;k++) {
        a:=M[k,j];
        for (l:=0;l<nc;l++){
          M[k,l]:=M[k,l]*pivo-M[j,l]*a;
         }
      }
      // on met des zeros au dessous de la diag
      for (k:=j+1;k<nl;k++) {
        a:=M[k,j];
        for (l:=j;l<nc;l++){
          M[k,l]:=M[k,l]*pivo-M[j,l]*a;
         }
       }
```

```
    }
}
return M;
}
```

De la même façon que pour la mèthode de Gauss, on va mettre des zéros sous la "fausse diagonale" et au dessus de cette "fausse diagonale" (on ne pourra pas bien sûr, mettre des zéros au dessus de cette "fausse diagonale", pour les colonnes sans pivot ! !)

```
gaussjordan_red(M):={
local pivo,jc,jl,k,nl,nc,temp,l,a;
nl:=nrows(M);
nc:=ncols(M);
//on met des 0 sous la fausse diagonale
jc:=0;
jl:=0;
//on doit traiter toutes les colonnes sauf la derniere
//on doit traiter toutes les lignes
while (jc<nc-1 and jl<nl) {
  //choix du pivot que l'on veut mettre en M[jl,jc]
    k:=jl;
    while (M[k,jc]==0 and k<nl-1) {k:=k+1;}
    //on ne fait la suite que si on a pivo!=0
    if (M[k,jc]!=0) {
      pivo:=M[k,jc];
      //echange de la ligne jl et de la ligne k
      for (l:=jc;l<nc;l++){
       temp:=M[jl,l];
       M[jl,l] := M[k,l];
        M[k,l]:=temp;
      }
     //fin du choix du pivot qui est M[jl,jc]
      //on met des 0 au dessus la fausse diagonale de
      //la colonne jc
      for (k:=0;k<jl;k++) {
        a:=M[k,jc];
        for (l:=0;l<nc;l++){
          M[k,l]:=M[k,l]*pivo-M[jl,l]*a;
         }
       }
       //on met 0 sous la fausse diag de la colonne jc
       for (k:=jl+1;k<nl;k++) {
        a:=M[k,jc];
        for (l:=jc;l<nc;l++){
          M[k,l]:=M[k,l]*pivo-M[jl,l]*a;
         }
       }
      //on a un pivot donc le numero de
```

```
      //la ligne de la fausse diag augmente de 1
      jl:=jl+1;
    }
    //ds tous les cas, le numero de
    //la colonne de la fausse diag augmente de 1
    jc:=jc+1;
}
return M;
}
```

On tape :
`M0:= [[1,2,3,6],[2,3,1,6],[3,2,1,6]]`
`gaussjordan_red(M0)`
On obtient :
`[[12,0,0,12],[0,12,0,12],[0,0,-12,-12]]`
On tape :
`M1:= [[1,2,3,4],[0,0,1,2],[0,0,5,1]]`
`gaussjordan_red(M1)`
On obtient :
`[[1,2,0,-2],[0,0,1,2],[0,0,0,-9]]`
On tape :
`M2:= [[1,2,3,4],[0,0,1,2],[0,0,5,1],[0,0,3,2],[0,0,-1,1]]`
`gaussjordan_red(M2)`
On obtient :
`[[1,2,0,-2],[0,0,1,2],[0,0,0,-9],[0,0,0,-4],[0,0,0,3]]`

$$\texttt{gaussjordan\_red}\left(\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 5 & 1 \\ 0 & 0 & 3 & 2 \\ 0 & 0 & -1 & 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 2 & 0 & -2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & -9 \\ 0 & 0 & 0 & -4 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

14.1.4 La méthode de Gauss et de Gauss-Jordan avec recherche du pivot

On peut bien sûr modifier la recherche du pivot.
Pour les méthodes numériques il est recommandé de normaliser les équations (on divise chaque ligne k par $\max_j |a_{k,j}|$) et de choisir le pivot qui a la plus grande valeur absolue.
En calcul formel, on prend l'expression exacte la plus simple possible. Ici on normalise les équations à chaque étape et on choisit le pivot qui a la plus grande valeur absolue : c'est ce que font, avec ce choix du pivot les programmes (cf le répertoire exemples), `gauss_reducni` (analogue à `gauss_redi`), `gauss_reducn` (analogue à `gauss_red`)), `gaussjordan_reducni` (analogue à `gaussjordan_redi`) et `gaussjordan_reducn` (analogue à `gaussjordan_red`) :

```
gaussjordan_reducn(M):={
local pivo,j,jc,jl,k,nl,nc,temp,l,a,piv,kpiv,maxi;
```

```
nl:=nrows(M);
nc:=ncols(M);
//on met des 0 sous la fausse diagonale
jc:=0;
jl:=0;
while (jc<nc-1 and jl<nl) {
   //on normalise les lignes
   for (jj:=0;jj<nl;jj++) {
       maxi:=max(abs(seq(M[jj,kk], kk=0..nc-1)));
       for (kk:=0;kk<nc;kk++) {
           M[jj,kk]:=M[jj,kk]/maxi;
       }
   }
   //choix du pivot que l'on veut mettre en M[jl,jc]
    kpiv:=jl;
    piv:=abs(M[kpiv,jc]);
    for (k:=jl+1;k<nl;k++){
    if (abs(M[k,jc])>piv) {piv:=abs(M[k,jc]);kpiv:=k;}
    }
   //on ne fait la suite que si on a piv!=0
    if (piv!=0) {
      pivo:=M[kpiv,jc];
      k:=kpiv;
      //echange de la ligne jl et de la ligne k
      for (l:=jc;l<nc;l++){
       temp:=M[jl,l];
       M[jl,l] := M[k,l];
        M[k,l]:=temp;
      }
     //fin du choix du pivot qui est M[jl,jc]
     //on met des 0 au dessus la fausse diagonale
     //de la colonne jc
      for (k:=0;k<jl;k++) {
        a:=M[k,jc];
        for (l:=0;l<nc;l++){
          M[k,l]:=M[k,l]*pivo-M[jl,l]*a;
         }
       }
       //on met des 0 sous la fausse dia de la colonne jc
       for (k:=jl+1;k<nl;k++) {
        a:=M[k,jc];
        for (l:=jc;l<nc;l++){
          M[k,l]:=M[k,l]*pivo-M[jl,l]*a;
         }
       }
      //on a un pivot donc,le numero de
      //la ligne de la fausse diag augmente de 1
      jl:=jl+1;
```

```
    }
    //ds tous les cas,le numero de
    //la colonne de la fausse diag augmente de 1
    jc:=jc+1;
}
return M;
}
```

On tape :
`M0:= [[1,2,3,6],[2,3,1,6],[3,2,1,6]]`
`gaussjordan_reducn(M0)`
On obtient :
`[[5/6,0,0,5/6],[0,5/6,0,5/6],[0,0,1,1]]`
On tape :
`M1:= [[1,2,3,4],[0,0,1,2],[0,0,5,1]]`
`gaussjordan_reducn(M1)`
On obtient :
`[[1/4,1/2,0,17/20],[0,0,1,1/5],[0,0,0,9/10]]`
On tape :
`M2:=[[1,2,3,4],[0,0,1,2],[0,0,5,1],[0,0,3,2],[0,0,-1,1]]`
`gaussjordan_reducn(M2)`
On obtient :
`[[1/4,1/2,0,17/20],[0,0,1,1/5],`
`[0,0,0,9/10],[0,0,0,7/15],[0,0,0,6/5]]`

On a donc :

$$\texttt{gaussjordan\_reducn}\left(\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 5 & 1 \\ 0 & 0 & 3 & 2 \\ 0 & 0 & -1 & 1 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & 0 & \frac{17}{20} \\ 0 & 0 & 1 & \frac{1}{5} \\ 0 & 0 & 0 & \frac{9}{10} \\ 0 & 0 & 0 & \frac{7}{15} \\ 0 & 0 & 0 & \frac{6}{5} \end{bmatrix}$$

14.1.5 Application : recherche du noyau grâce à Gauss-Jordan

Soit une application linéaire f de $\mathbb{R}^n$ dans $\mathbb{R}^p$, de matrice A dans la base canonique de $\mathbb{R}^n$ et de $\mathbb{R}^p$.
Trouver le noyau de f revient à résoudre $f(X) = 0$ ou encore à résoudre $AX = 0$ pour $X \in \mathbb{R}^n$.
Pour cela, on va utiliser la méthode de Gauss-Jordan en rajoutant des lignes de 0 aux endroits où l'on n'a pas trouvé de pivot de façon à ce que les pivots suivants se trouve sur la diagonale (et non sur la "fausse diagonale").
Plus précisément :

- quand on a trouvé un pivot pour une colonne, à l'aide de la méthode habituelle de réduction de Gauss-Jordan, on met sur cette colonne :
1 sur la diagonale et
0 de part et d'autre du 1.
- quand on n'a pas trouvé de pivot pour la colonne j, on rajoute une ligne de 0 : la ligne j devient une ligne de 0 et les autres lignes sont décalées.
- on rajoute éventuellement à la fin, des lignes de 0, pour traiter toutes les colonnes de A et avoir une matrice carrée.
- on enlève éventuellement à la fin, des lignes de 0, pour avoir une matrice carrée.
Remarque : on ne change pas le nombre de colonnes.
Ainsi, si la colonne C_j a un 0 sur sa diagonale, si e_j est le j-ième vecteur de la base canonique de $\mathbb{R}^n$, alors $N_j = C_j - e_j$ est un vecteur du noyau. En effet, on a $A(e_j) = C_j$; si on a $C_j = a1e_1 + ... + a_{j-1}e_{j-1}$, pour $k < j$, et pour $a_k! = 0$, on a $A(e_k) = e_k$ c'est à dire $A(C_j) = C_j = A(e_j)$, soit $A(C_j - e_j) = A(N_j) = 0$.
Voici le programme correspondant en conservant la variable `jl` afin de faciliter la lisibilité du programme :

```
gaussjordan_noyau(M):={
local pivo,jc,jl,k,j,nl,nc,temp,l,a,noyau;
nl:=nrows(M);
nc:=ncols(M);
//on met des 0 sous la diagonale
jc:=0;
jl:=0;
// on traite toutes les colonnes
while (jc<nc and jl<nl) {
  //choix du pivot que l'on veut mettre en M[jl,jc]
    k:=jl;
    while (M[k,jc]==0 and k<nl-1) {k:=k+1;}
    //on ne fait la suite que si on a pivo!=0
    if (M[k,jc]!=0) {
      pivo:=M[k,jc];
      //echange de la ligne jl et de la ligne k
      for (l:=jc;l<nc;l++){
       temp:=M[jl,l];
       M[jl,l] := M[k,l];
        M[k,l]:=temp;
      }
     //fin du choix du pivot qui est M[jl,jc]
      //on met 1 sur la diagonale de la colonne jc
      for (l:=0;l<nc;l++) {
      M[jl,l]:=M[jl,l]/pivo;
      }
      //on met des 0 au dessus de la diagonale
      // de la colonne jc
      for (k:=0;k<jl;k++) {
        a:=M[k,jc];
        for (l:=0;l<nc;l++){
          M[k,l]:=M[k,l]-M[jl,l]*a;
```

```
          }
        }
        //on met des 0 sous la diag de la colonne jc
        for (k:=jl+1;k<nl;k++) {
         a:=M[k,jc];
         for (l:=jc;l<nc;l++){
           M[k,l]:=M[k,l]-M[jl,l]*a;
          }
        }
     }
     else{
     //on ajoute une ligne de 0 si ce n'est pas le dernier 0
     if (jl<nc-1){
      for (j:=nl;j>jl;j--){
       M[j]:=M[j-1];
       }
     M[jl]:=makelist(0,1,nc);
     nl:=nl+1;
     }
    }
     //ds tous les cas,le numero de colonne et
     //le numero de ligne augmente de 1
     jc:=jc+1;  jl:=jl+1;
     //il faut faire toutes les colonnes
     if (jl==nl and jl<nc) { M[nl]:=makelist(0,1,nc);nl:=nl+1;}
}
noyau:=[];
//on enleve les lignes en trop pour avoir
//une matrice carree de dim nc
//on retranche la matrice identite
M:=M[0..nc-1]-idn(nc);
for(j:=0;j<nc;j++){
if (M[j,j]==-1) {noyau:=append(noyau,M[0..nc-1,j]);}
}
return noyau;
}
```

Remarque

On peut écrire le même programme en supprimant la variable jl puisque $jl = jc$ (on met jc à la place de jl et on supprime $jl := 0$ et $jl := jl + 1$).

On met ce programme dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`), puis on le teste et on le valide avec `OK`.

On tape :

```
gaussjordan_noyau([[1,2,3],[1,3,6],[2,5,9]])
```

On obtient :

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

On tape :

```
gaussjordan_noyau([[1,2,3,4],[1,3,6,6],[2,5,9,10]])
```

On obtient :

```
[[-3,3,-1,0],[0,2,0,-1]]
```

14.2 Résolution d'un système linéaire

14.2.1 Résolution d'un système d'équations linéaires

L'algorithme

On associe à un système d'équations linéaires, une matrice A constituèe de la matrice du système augmentèe d'une colonne formée par l'opposé du second membre.
Par exemple au système : $[x+y=4, x-y=2]$ d'inconnues $[x,y]$ on associe la matrice :
$A=[[1,1,-4],[1,-1,-2]]$.
Puis on réduit avec la méthode de Gauss-Jordan la matrice A pour obtenir une matrice B. Pour chaque ligne de B :
- si il n'y a que des zéro on regarde la ligne suivante,
- si il y a des zéro sauf en dernière position, il n'y a pas de solution,
- dans les autres cas on obtient la valeur de la variable de même indice que le premier élément non nul rencontré sur la ligne,
- les valeurs arbitraires correspondent aux zéros de la diagonale de B et en général il y a des valeurs non nulles au dessus de ces zéros, c'est pourquoi il faut initialisé la solution au vecteur des variables.

Le programme

Voici le programme de résolution d'un système linéaire :

```
//Veq vecteur des equations
//v vecteur de variables
//renvoie le vecteur solution
linsolv(Veq,v):={
  local A,B,j,k,l,deq,d,res,ll,rep;
  d:=size(v);
  deq:=size(Veq);
  //A est la matrice du systeme +le 2nd membre
  A:=syst2mat(Veq,v);
  //B matrice reduite de Gauss-jordan
  B:=rref(A);
  res:=v;
  //ll ligne l de B
  ll:=makelist(0,0,d);
  for (l:=0; l<deq;l++){
    for (k:=0;k<d+1;k++){
      ll[k]:=B[l][k];
    }
  j:=l;
  while (ll[j]==0 && j<d){
    j:=j+1;
```

```
  }
  //si (j==d and ll[d]==0)
  //ll=ligne de zeros on ne fait rien
  if (j==d and ll[d]!=0){
    // pas de sol
    return [];
  }
  else {//la sol res[j] vaut rep/ll[j]
      if (j<d) {
      rep:=-ll[d];
      for (k:=j+1;k<d;k++) {
      rep:=rep-ll[k]*v[k];
      }
      res[j]:=rep/ll[j];
      }
    }
  }
  return res;
}
```

Autre algorithme

14.2.2 Résolution de $MX = b$ donné sous forme matricielle

L'algorithme

On transforme la résolution de $MX = b$ en $MX - b = AY = 0$ où A est la matrice constituée de la matrice M du système augmentée d'une colonne formée par l'opposé du second membre b.
Y est donc un vecteur du noyau de A ayant comme dernière composante 1.
D'après l'algorithme de recherche du noyau, seul le dernier vecteur a comme dernière composante -1. Si `-ker(A)` renvoie n vecteurs, la solution Y est donc une combinaison arbitraire des $n-1$ premiers vecteurs du noyau plus le n-ième vecteur.

Le programme

Voici le programme de résolution de $AX = b$:

```
//M*res=b res renvoie le vecteur solution
//M:=[[1,1],[1,-1]]; b:=[4,2]
//M:=[[1,1,1],[1,1,1],[1,1,1]];b:=[0,0,0]
//M:=[[1,2,1],[1,2,5],[1,2,1]];b:=[0,1,0]
linsolvm(M,b):={
  local A,B,N,n,k,d,res;
  d:=ncols(M);
  //A est la matrice du systeme +le 2nd membre
  A:=border(M,-b);
  //N contient une base du noyau
  N:=-ker(A);
  n:=size(N);
```

```
  //res a d+1 composante (la derniere=1)
  res:=makelist(0,0,d);
  //C_(k) designe les constantes arbitraires
  for (k:=0;k<n-1;k++){
    res:=res+N[k]*C_(k);
  }
  res:=res+N[n-1];
  res:=suppress(res,d);
  return res;
}
```

14.3 La décomposition LU d'une matrice

C'est l'interprétation matricielle de la méthode de Gauss.
Si A est une matrice carrée d'ordre n, il existe une matrice L triangulaire supérieure, une matrice L triangulaire inférieure, et une matrice P de permutation telles que :
$P * A = L * U$.
Supposons tout d'abord que l'on peut faire la méthode de Gauss sans échanger des lignes. Mettre des zéros sous la diagonale de la 1-ière colonne (d'ndice 0) de A, reviens à multiplier A par la matrice E_0 qui a des 1 sur sa diagonale, comme première colonne :
$[1, -A[1,0]/A[0,0], ... - A[n-1,0]/A[0,0]]$ et des zéros ailleurs.
Puis si $A1 = E1 * A$, mettre des zéros sous la diagonale de la 2-ière colonne (d'ndice 1) de $A1$, reviens à multiplier $A1$ par la matrice E_1 qui a des 1 sur sa diagonale, comme deuxième colonne :
$[0, 1, -A1[2,1]/A[1,1], ... - A[n-1,1]/A[1,1]]$ et des zéros ailleurs.
On continue ainsi jusqu'à mettre des zéros sous la diagonale de la colonne d'indice $n-2$, et à la fin la matrice $U = E_{n-2} * ... * E_1 * E_0 * A$ est triangulaire supérieure et on a $L = inv(E_{n-2} * ... * E_1 * E_0)$.
Le calcul de $inv(L)$ est simple car on a :
- $inv(E_0)$ des 1 sur sa diagonale, comme colonne d'indice 0 $[1, +A[1,0]/A[0,0], ...+ A[n-1,0]/A[0,0]]$ et des zéros ailleurs de même $inv(E_k)$ est obtenue à partir de E_k "en changeant les moins en plus".
- la colonne d'indice k de $inv(E_0) * inv(E_1)...inv(E_{n-2})$ est égale à la colonne d'indice k de E_k.
Lorsqu'il y a à faire une permutation de lignes, il faut répercuter cette permutation sur L et sur U : pour faciliter la programmation on va conserver les valeurs de L et de U dans une seule matrice R que l'on séparera à la fin : U sera la partie supérieure et la diagonale de R L sera la partie inférieure de R, avec des 1 sur sa diagonale.
Voici le programme de séparation :

```
splitmat(R):={
  local L,U,n,k,j;
  n:=size(R);
  L:=idn(n);
  U:=makemat(0,n,n);
```

```
  for (k:=0;k<n;k++){
    for (j:=k;j<n;j++){
      U[k,j]:= R[k,j];
    }
  }
  for (k:=1;k<n;k++){
    for (j:=0;j<k;j++){
      L[k,j]:= R[k,j];
    }
  }
return (L,U);
};
```

Le programme ci-dessous, `decomplu(A)`, renvoie la permutation p que l'on a fait sur les lignes, L et U et on a : $P * A = L * U$.
Voici le programme de décomposition LU qui utilise `splitmat` ci-dessus :

```
//A:=[[5,2,1],[5,2,2],[-4,2,1]]
//A:=[[5,2,1],[5,-6,2],[-4,2,1]]
// utilise splitmat
decomplu(A):={
  local B,R,L,U,n,j,k,l,temp,p;
  n:=size(A);
  p:=seq(k,k,0,n-1);
  R:=makemat(0,n,n);
  B:=A;
  l:=0;
//on traite toutes les colonnes
  while (l<n-1) {
    if (A[l,l]!=0){
    //pas de permutations
    //on recopie dans R la ligne l de A
    //a partir de la diagonale
      for (j:=l;j<n;j++){R[l,j]:=A[l,j];}
      //on met des zeros sous la diagonale
      //dans la colonne l
      for (k:=l+1;k<n;k++){
for (j:=l+1;j<n;j++){
  A[k,j]:=A[k,j]-A[l,j]*A[k,l]/A[l,l];
  R[k,j]:=A[k,j];
}
R[k,l]:=A[k,l]/A[l,l];
A[k,l]:=0;
      }
    l:=l+1;
    }
    else {
      k:=l;
      while ((k<n-1) and (A[k,l]==0)) {
```

```
        k:=k+1;
      }
      //si (A[k,l]==0) A est non inversible,
      //on passe a la colonne suivante
      if (A[k,l]==0) {
l:=l+1;
      }
      else {
      //A[k,l]!=0) on echange la ligne l et k ds A et R
       for (j:=l;j<n;j++){
        temp:=A[k,j];
        A[k,j]:=A[l,j];
        A[l,j]:=temp;
       };
       for (j:=0;j<n;j++){
        temp:=R[k,j];
        R[k,j]:=R[l,j];
        R[l,j]:=temp;
       }
       //on note cet echange dans  p
       temp:=p[k];
       p[k]:=p[l];
       p[l]:=temp;
     }//fin du if (A[k,l]==0)
    }//fin du if (A[l,l]!=0)
  }//fin du while
  L,U:=splitmat(R);
  return(p,L,U);
};
```

On tape :

```
A:=[[5,2,1],[5,2,2],[-4,2,1]]
decomplu(A)
```

On obtient :

```
[0,2,1],[[1,0,0],[-4/5,1,0],[1,0,1]],
[[5,2,1],[0,18/5,9/5],[0,0,1]]
```

On verifie, on tape (car `permu2mat(p)=P=inv(P)`) :

```
[[1,0,0],[0,0,1],[0,1,0]]*[[1,0,0],[-4/5,1,0],[1,0,1]]*
[[5,2,1],[0,18/5,9/5],[0,0,1]]
```

On obtient bien `[[5,2,1],[5,2,2],[-4,2,1]]`

On tape :

```
B:=[[5,2,1],[5,-6,2],[-4,2,1]]
decomplu(B)
```

On obtient :

```
[0,1,2]„[[1,0,0],[1,1,0],[-4/5,-9/20,1]],
[[5,2,1],[0,-8,1],[0,0,9/4]]
```

On verifie, on tape :

```
[[1,0,0],[1,1,0],[-4/5,-9/20,1]]*[[5,2,1],[0,-8,1],[0,0,9/4]]
```

On obtient bien `[[5,2,1],[5,-6,2],[-4,2,1]]`

14.4 Décomposition de Cholesky d'une matrice symétrique définie positive

14.4.1 Les méthodes

Lorsque la matrice A est la matrice associée à une forme bilinéaire définie positive, on lui associe la matrice symétrique B=1/2*(A+tran(A)) qui est la matrice de la forme quatratique associée.
Avec ses hypothèses, il existe une matrice triangulaire inférieure unique C telle que B=C*tran(C).
Pour déterminer C on présente ici deux méthodes :

La méthode utilisant la décomposition LU

Si on décompose B selon la méthode LU, on n'est pas obligé de faire des échanges de lignes car les sous-matrices principales B_k d'ordre k (obtenues en prenant les k premières lignes et colonnes de B) sont des matrices inversibles car ce sont des matrices de formes définies positives.
On a donc :
p,L,U:=decomplu(B) avec p=[0,1..n-1] si A est d'ordre n. Posons D la matrice diagonale ayant comme diagonale la racine carrée de la diagonale de U.
On a alors C=L*D et tran(C)=inv(D)*U.
Pour le montrer on utilise le théorème :
Si B=C*F avec B symétrique, C triangulaire inférieure et F triangulaire supérieure de même diagonale que C alors F=tran(C).
En effet on a :
B=tran(B) donc C*F=tran(C*F)=tran(F)*tran(C).
On en déduit l'égalité des 2 matrices :
inv(tran(F))*C=tran(C)*inv(F)
la première est triangulaire inférieure, et la deuxième est triangulaire supérieure donc ces matrices sont diagonales et leur diagonale n'a que des 1 ces 2 matrices sont donc égales à la matrice unité.

La méthode par identification

On peut aussi déterminer C par identification en utilisant l'égalité B=C*tran(C) et C triangulaire inférieure.
On trouve pour tout entier j compris entre 0 et $n-1$:
- pour la diagonale :
$(\mathtt{C[j,j]})^2 = \mathtt{B[j,j]} - \sum_{\mathtt{k=0}}^{\mathtt{j-1}} (\mathtt{C[j,k]})^2$
- pour les termes subdiagonaux c'est à dire pour $j+1 \leq l \leq n-1$:
$\mathtt{C[l,j]} = \mathtt{1/C[j,j]} * (\mathtt{B[l,j]} - \sum_{\mathtt{k=0}}^{\mathtt{j-1}} \mathtt{C[l,k]} * \mathtt{C[j,k]})$
On peut donc calculer les coefficients de C par colonnes, en effet, pour avoir la colonne j :
- on calcule le terme diagonal C[j,j] qui fait intervenir les termes de la ligne j des colonnes précédentes (avec un test vérifiant que le terme $\mathtt{B[j,j]} - \sum_{\mathtt{k=0}}^{\mathtt{j-1}} (\mathtt{C[j,k]})^2$ est positif), puis,
- on calcule les termes subdiagonaux C[l,j] pour $j+1 \leq l \leq n-1$ qui font

intervenir les termes des colonnes précédentes des lignes précédentes. Mais cet algorithme n'est pas très bon car les termes de la matrice C sont obtenus à chaque étape avec une multiplication ou une division de racine carrée : il serait préférable d'introduire les racines carrées qu'à la fin du calcul comme dans la méthode LU. On va donc calculer une matrice CC (sans utiliser de racines carrées) et une matrice diagonale D (qui aura des racines carrées sur sa diagonale) de tel sorte que C=CC*D (la colonne k de CC*D est la colonne k de CC multiplié par D[k,k]).
Par exemple : C[0,0]* C[0,0]= B[0,0] on pose :
CC[0,0]= B[0,0]=a et D[0,0]=1/sqrt(a) ainsi
C[0,0]= CC[0,0]/sqrt(a)=sqrt(a) et on a bien :
C[0,0]* C[0,0]=a=B[0,0]
On aura donc C[l,0]=CC[l,0]/sqrt(a) c'est à dire :
CC[l,0]=B[l,0] pour $0 \leq l < n$
Puis :
C[1,1]*C[1,1]=B[1,1]-C[1,0]*C[1,0]=
B[1,1]-CC[1,0]*CC[1,0]/a=b
on pose :
CC[1,1]= B[1,1]-CC[1,0]*CC[1,0]/a=b et
D[1,1]=1/sqrt(b)
On aura donc pour $2 \leq l < n$;
C[l,1]=CC[l,1]/sqrt(b)=1/sqrt(b)*(B[l,1]-C[l,0]*C[1,0])
c'est à dire pour $2 \leq l < n$:
CC[l,1]=B[l,1]-C[l,0]*C[1,0]=B[l,1]-CC[l,0]*CC[1,0]/a
etc.....
Les formules de récurences pour le calcul de CC sont :
- pour la diagonale :
$\texttt{CC}[\texttt{j},\texttt{j}] = \texttt{B}[\texttt{j},\texttt{j}] - \sum_{\texttt{k}=0}^{\texttt{j}-1}(\texttt{CC}[\texttt{j},\texttt{k}])^2/\texttt{CC}[\texttt{k},\texttt{k}]$
- pour les termes subdiagonaux c'est à dire pour $j+1 \leq l \leq n-1$:
$\texttt{CC}[\texttt{l},\texttt{j}] = \texttt{B}[\texttt{l},\texttt{j}] - \sum_{\texttt{k}=0}^{\texttt{j}-1} \texttt{CC}[\texttt{l},\texttt{k}] * \texttt{CC}[\texttt{j},\texttt{k}]/\texttt{CC}[\texttt{k},\texttt{k}]$
avec $\texttt{D}[\texttt{j},\texttt{j}] = 1/\texttt{sqrt}(\texttt{CC}[\texttt{j},\texttt{j}])$.

14.4.2 Le programme de factorisation de Cholesky avec LU

```
//utilise decomplu et splitmat ci-dessus
//A:=[[1,0,-1],[0,2,4],[-1,4,11]]
//A:=[[1,1,1],[1,2,4],[1,4,11]]
//A:=[[1,0,-2],[0,2,6],[0,2,11]]
//A:=[[1,-2,4],[-2,13,-11],[4,-11,21]]
//A:=[[-1,-2,4],[-2,13,-11],[4,-11,21]] (pas def pos)
//A:=[[24,66,13],[66,230,-11],[13,-11,210]]
choles(A):={
  local j,n,p,L,U,D,p0;
  n:=size(A);
  A:=1/2*(A+tran(A));
 (p,L,U):=decomplu(A);
  p0:=makelist(x->x,0,n-1);
  if (p!=p0) {return "pas definie positive ";}
  D:=makemat(0,n,n);
```

```
  for (j:=0;j<n;j++) {
    //if (U[j,j]<0) {return "pas def positive";}
    D[j,j]:=sqrt(U[j,j]);
  }
  return normal(L*D);
}
```

14.4.3 Le programme de factorisation de Cholesky par identification

```
//A:=[[1,0,-1],[0,2,4],[-1,4,11]]
//A:=[[1,1,1],[1,2,4],[1,4,11]]
//A:=[[1,0,-2],[0,2,6],[0,2,11]]
//A:=[[1,-2,4],[-2,13,-11],[4,-11,21]]
//A:=[[-1,-2,4],[-2,13,-11],[4,-11,21]] (pas def pos)
//A:=[[24,66,13],[66,230,-11],[13,-11,210]]
cholesi(A):={
  local j,n,l,k,C,c2,s;
  n:=size(A);
  A:=1/2*(A+tran(A));
  C:=makemat(0,n,n);;
  for (j:=0;j<n;j++) {
    s:=0;
    for (k:=0;k<j;k++) {
      s:=s+(C[j,k])^2;
    }
    c2:=A[j,j]-s;
    if (c2<=0) {return "pas definie positive ";}
    C[j,j]:=normal(sqrt(c2));
    for (l:=j+1;l<n;l++) {
      s:=0;
      for (k:=0;k<j;k++) {
s:=s+C[l,k]*C[j,k];
      }
      C[l,j]:=normal(1/sqrt(c2)*(A[l,j]-s));
    }
  }
return C;
}
```

Ce programme n'est pas très bon car les termes de la matrice `C` sont obtenus avec une multiplication ou une division de racine carrée...
On se pourra se reporter ci-dessous pour avoir le programme choleski optimisé.

14.4.4 Le programme optimisé de factorisation de Cholesky par identification

```
//A:=[[1,0,-1],[0,2,4],[-1,4,11]]
//A:=[[1,1,1],[1,2,4],[1,4,11]]
//A:=[[1,0,-2],[0,2,6],[0,2,11]]
```

```
//A:=[[1,-2,4],[-2,13,-11],[4,-11,21]]
//A:=[[-1,-2,4],[-2,13,-11],[4,-11,21]] (pas def pos)
//A:=[[24,66,13],[66,230,-11],[13,-11,210]]
choleski(A):={
  local j,n,l,k,CC,D,c2,s;
  n:=size(A);
  A:=1/2*(A+tran(A));
  CC:=makemat(0,n,n);
  D:=makemat(0,n,n);
  for (j:=0;j<n;j++) {
    s:=0;
    for (k:=0;k<j;k++) {
      s:=s+(CC[j,k])^2/CC[k,k];
    }
    c2:=normal(A[j,j]-s);
    //if (c2<=0) {return "pas definie positive ";}
    CC[j,j]:=c2;
    D[j,j]:=normal(1/sqrt(c2));
    for (l:=j+1;l<n;l++) {
      s:=0;
      for (k:=0;k<j;k++) {
s:=s+CC[l,k]*CC[j,k]/CC[k,k];
      }
      CC[l,j]:=normal(A[l,j]-s);
    }
  }
return normal(CC*D);
}
```

Avec cette méthode, pour obtenir les coefficients diagonaux ont utilise la même relation de récurrence que pour les autres coefficients. De plus on peut effectuer directement et facilement la multiplication par `D` sans avoir à définir `D` en multipliant les colonnes de `CC` par la même valeur `1/sqrt(c)` avec `c=CC[k,k]`... donc on écrit :

```
choleskii(A):={
  local j,n,l,k,CC,c,s;
  n:=size(A);
  A:=1/2*(A+tran(A));
  CC:=makemat(0,n,n);
  for (j:=0;j<n;j:=j+1) {
     for (l:=j;l<n;l++) {
      s:=0;
      for (k:=0;k<j;k++) {
//if (CC[k,k]<=0) {return "pas definie positive ";}
if (CC[k,k]==0) {return "pas definie";}
s:=s+CC[l,k]*CC[j,k]/CC[k,k];
      }
      CC[l,j]:=A[l,j]-s;
```

```
    }
  }
  for (k:=0;k<n;k++) {
    c:=CC[k,k];
    for (j:=k;j<n;j++) {
      CC[j,k]:=normal(CC[j,k]/sqrt(c));
    }
  }
  return CC;
}
```

Avec la traduction pour avoir une fonction interne à Xcas :

```
cholesky(_args)={
gen args=(_args+mtran(_args))/2;
matrice &A=*args._VECTptr;
int n=A.size(),j,k,l;
vector<vecteur> C(n,vecteur(n)), D(n,vecteur(n));
for (j=0;j<n;j++) {
gen s;
for (k=0;k<j;k++)
s=s+pow(C[j][k],2)/C[k][k];
gen c2=A[j][j]-s;
if (is_strictly_positive(-c2))
setsizeerr("Not a positive define matrice");
C[j][j]=c2;
D[j][j]=normal(1/sqrt(c2));
for (l=j+1;l<n;l++) {
s=0;
for (k=0;k<j;k++)
s=s+C[l][k]*C[j][k]/C[k][k];
C[l][j]=A[l][j]-s;
}
}
matrice Cmat,Dmat;
vector_of_vecteur2matrice(C,Cmat);
vector_of_vecteur2matrice(D,Dmat);
return Cmat*Dmat;
}
```

14.5 Réduction de Hessenberg

14.5.1 La méthode

Une matrice de Hessenberg est une matrice qui a des zéros sous la "deuxième diagonale inférieure".
Soit A une matrice. On va chercher B une matrice de Hessenberg semblable à A. Pour cela on va mettre des zéros sous cette diagonale en utilisant la méthode de Gauss mais en prenant les pivots sur la "deuxième diagonale inférieure" encore

appelée "sous-diagonale" : cela revient à multiplier A par $Q = R^{-1}$ et cela permet de conserver les zéros lorsque l'on multiplie à chaque étape le résultat par R pour obtenir une matrice semblable à A. Si on est obligé de faire un échange de lignes (correspondant à la multiplication à droite par E) il faudra faire aussi un échange de colonnes (correspondant à la multiplication à gauche par E.
Par exemple si on a :

$$A := \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Pour mettre des zéros dans la première colonne en dessous de a_{10}, on va multiplier A à gauche par Q et à droite par $R = Q^{-1}$ avec si on suppose $a_{10}! = 0$ c'est à dire si on peut choisir comme pivot a_{10} : $Q := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -a_{20}/a_{10} & 1 & 0 & 0 \\ 0 & -a_{30}/a_{10} & 0 & 1 & 0 \\ 0 & -a_{40}/a_{10} & 0 & 0 & 1 \end{bmatrix}$

$$R := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & +a_{20}/a_{10} & 1 & 0 & 0 \\ 0 & +a_{30}/a_{10} & 0 & 1 & 0 \\ 0 & +a_{40}/a_{10} & 0 & 0 & 1 \end{bmatrix}$$

On tape alors :
$\mathtt{A} := [[\mathtt{a_{00}}, \mathtt{a_{01}}, \mathtt{a_{02}}, \mathtt{a_{03}}, \mathtt{a_{04}}], [\mathtt{a_{10}}, \mathtt{a_{11}}, \mathtt{a_{12}}, \mathtt{a_{13}}, \mathtt{a_{14}}],$
$[\mathtt{a_{20}}, \mathtt{a_{21}}, \mathtt{a_{22}}, \mathtt{a_{23}}, \mathtt{a_{24}}], [\mathtt{a_{30}}, \mathtt{a_{31}}, \mathtt{a_{32}}, \mathtt{a_{33}}, \mathtt{a_{34}}], [\mathtt{a_{40}}, \mathtt{a_{41}}, \mathtt{a_{42}}, \mathtt{a_{43}}, \mathtt{a_{44}}]]$
$\mathtt{Q} := [[\mathtt{1}, \mathtt{0}, \mathtt{0}, \mathtt{0}, \mathtt{0}], [\mathtt{0}, \mathtt{1}, \mathtt{0}, \mathtt{0}, \mathtt{0}], [\mathtt{0}, (-\mathtt{a_{20}})/\mathtt{a_{10}}, \mathtt{1}, \mathtt{0}, \mathtt{0}],$
$[\mathtt{0}, (-\mathtt{a_{30}})/\mathtt{a_{10}}, \mathtt{0}, \mathtt{1}, \mathtt{0}], [\mathtt{0}, (-\mathtt{a_{40}})/\mathtt{a_{10}}, \mathtt{0}, \mathtt{0}, \mathtt{1}]]$
$\mathtt{R} := [[\mathtt{1}, \mathtt{0}, \mathtt{0}, \mathtt{0}, \mathtt{0}], [\mathtt{0}, \mathtt{1}, \mathtt{0}, \mathtt{0}, \mathtt{0}], [\mathtt{0}, (\mathtt{a_{20}})/\mathtt{a_{10}}, \mathtt{1}, \mathtt{0}, \mathtt{0}],$
$[\mathtt{0}, (\mathtt{a_{30}})/\mathtt{a_{10}}, \mathtt{0}, \mathtt{1}, \mathtt{0}], [\mathtt{0}, (\mathtt{a_{40}})/\mathtt{a_{10}}, \mathtt{0}, \mathtt{0}, \mathtt{1}]]$
On obtient la matrice $B1$:

$$B1 = Q * A * R = R^{-1} * A * R = \begin{bmatrix} a_{00} & ... & a_{02} & a_{03} & a_{04} \\ a_{10} & ... & a_{12} & a_{13} & a_{14} \\ 0 & ... & ... & ... & ... \\ 0 & ... & ... & ... & ... \\ 0 & ... & ... & ... & ... \end{bmatrix}$$ où les ... sont

des expressions des coefficients de A.
On va faire cette transformation successivement sur la deuxième colonne de la matrice $B1$ pour obtenir $B2$ etc...
On appellera B toutes les matrices obtenues successivement.
Si on doit échanger les deux lignes k et j pour avoir un pivot, cela revient à multiplier à gauche la matrice B par une matrice E égale à la matrice identité ayant subie l'échange des deux lignes k et j. Il faudra alors, aussi multiplier à droite la matrice B par E c'est à dire échanger les deux colonnes k et j de B

14.5.2 Le programme de réduction de Hessenberg

```
//A est mise sous forme de hessenberg B avec
//P matrice de passage
```

```
//p indique si il y a eu une permutation de lignes
//a chaque etape la matrice inv(R)
// met des zeros sous la "sous diagonale"
//B=P^-1AP ex A:=[[5,2,1],[5,2,2],[-4,2,1]]
//A:=[[5,2,1,1],[5,2,2,1],[5,2,2,1],[-4,2,1,1]]
hessenbg(A):={
  local B,R,P,n,j,k,l,temp,p;
  n:=size(A);
  P:=idn(n);
  p:=seq(k,k,0,n-1);
  B:=A;
  l:=1;
  while (l<n) {
     R:=idn(n);
     if (B[l,l-1]!=0){
     for (k:=l+1;k<n;k++){
R[k,l]:=B[k,l-1]/B[l,l-1];
//on multiplie B a droite par inv(R)
        for (j:=l;j<n;j++){
  B[k,j]:=B[k,j]-B[l,j]*B[k,l-1]/B[l,l-1];
}
B[k,l-1]:=0;
      }
     //on multiplie B et P a gauche par R
     B:=B*R;
     P:=P*R;
    l:=l+1;
    }
  else {
    k:=l;
    while ((k<n-1) and (B[k,l-1]==0)) {
    k:=k+1;
    }

    if (B[k,l-1]==0) {l:=l+1;}
    else{
    //B[k,l]!=0) on echange ligne l et k ds B
    for (j:=l-1;j<n;j++){
    temp:=B[k,j];
    B[k,j]:=B[l,j];
    B[l,j]:=temp;
    };
    //A[k,l]!=0) on echange colonne l et k ds B et P
    for (j:=0;j<n;j++){
    temp:=B[j,k];
    B[j,k]:=B[j,l];
    B[j,l]:=temp;
    };
```

```
    for (j:=0;j<n;j++){
    temp:=P[j,k];
    P[j,k]:=P[j,l];
    P[j,l]:=temp;
    };
    temp:=p[k];
    p[k]:=p[l];
    p[l]:=temp;
  }
  }
  }
  return(p,P,B);
};
```

p nous dit les échanges effectués et on a $B = P^{-1} * A * P$.
Attention ! ! si A est symétrique, cette transformation détruit la symétrie et on n'a donc pas une tridiagonalistion d'une matrice symtrisue par cette méthode.

14.6 Tridiagonalisation des matrices symétriques avec des rotations

On a le théorème : Pour toute matrice symétrique A d'ordre n, il existe une matrice P, produit de $n-2$ matrices de rotations, telle que $B = {}^tP * A * P$ soit tridiagonale : c'est la réduction de Givens.

14.6.1 Matrice de rotation associée à e_p, e_q

Dans $\mathbb{R}^n$, on appelle rotation associée à e_p, e_q, une rotation d'angle t du plan dirigé par e_p, e_q où e_k désigne le $k+1$-ième vecteur de la base canonique de $\mathbb{R}^n$ (la base canonique est $e_0 = [1, 0.., 0], e_1 = [0, 1, 0.., 0]$ etc...).
Si $\mathbb{R}^n$ est rapporté à la base canonique, à cette rotation est associée une matrice $G(n, p, q, t)$ dont le terme général est :
si $k \notin \{p, q\}$, $g_{k,k} = 1$
$g_{p,p} = g_{q,q} = \cos(t)$
$g_{p,q} = -g_{q,p} = -\sin(t)$
Voici un matrice de rotation associée à e_1, e_3 (deuxième et quatrième vecteur de base) de $\mathbb{R}^5$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos(t) & 0 & -\sin(t) & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & \sin(t) & 0 & \cos(t) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

et le programme qui construit une telle matrice :

```
rota(n,p,q,t):={
local G;
G:=idn(n);
```

```
G[p,p]:=cos(t);
G[q,q]:=cos(t);
G[p,q]:=-sin(t);
G[q,p]:=sin(t);
return G;
}
```

14.6.2 Réduction de Givens

Soit A une matrice symétrique. On va chercher $G1$ une matrice de rotation associée à e_1, e_2 pour annuler les coefficients $2,0$ et $0,2$ de ${}^tG1 * A * G1$.
Regardons un exemple :

```
G1:=[[1,0,0,0,0],[0,cos(t),-sin(t),0,0],[0,sin(t),cos(t),0,0],
[0,0,0,1,0],[0,0,0,0,1]]
A:=[[a,b,c,d,e],[b,f,g,h,j],[c,g,k,l,m],[d,h,l,n,o],[e,j,m,o,r]]
```

On obtient :

$${}^tG1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos(t) & \sin(t) & 0 & 0 \\ 0 & -\sin(t) & \cos(t) & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, A = \begin{bmatrix} a & b & c & d & e \\ b & f & g & h & j \\ c & g & k & l & m \\ d & h & l & n & o \\ e & j & m & o & r \end{bmatrix}$$

On a :

$${}^tG1 * A = \begin{bmatrix} a & b & c & d & e \\ b\cos(t) + c\sin(t) & f\cos(t) + g\sin(t) & .. & .. & .. \\ -b\sin(t) + c\cos(t) & -f\sin(t) + g\cos(t) & .. & .. & .. \\ d & h & l & n & o \\ e & j & m & o & r \end{bmatrix}$$

On choisit t pour que $-b\sin(t) + c\cos(t) = 0$ par exemple :
$\cos(t) = b/\sqrt{(b^2 + c^2)}$ et $\sin(t) = c/\sqrt{(b^2 + c^2)}$.
On a :

$$G1 := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos(t) & -\sin(t) & 0 & 0 \\ 0 & \sin(t) & \cos(t) & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

et donc puisqu'on a choisit $-b\sin(t) + c\cos(t) = 0$, on a bien annuler les coefficients $2,0$ et $0,2$ de :

$${}^tG1*A*G1 = \begin{bmatrix} a & b\cos(t) + c\sin(t) & -b\sin(t) + c\cos(t) & d & e \\ b\cos(t) + c\sin(t) & .. & .. & .. & .. \\ -b\sin(t) + c\cos(t) & .. & .. & .. & .. \\ d & .. & .. & n & o \\ e & .. & .. & o & r \end{bmatrix}$$

Puis on annule les coefficients $0,3$ et $3,0$ de l la matrice $A1$ obtenue en formant ${}^tG2 * A1 * G2$ avec :

$$G2 := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos(t) & 0 & -\sin(t) & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & \sin(t) & 0 & \cos(t) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

en choissant correctement t etc...

14.6.3 Le programme de tridiagonalisation par la méthode de Givens

```
//A:=[[1,-1,2,1],[-1,1,-1,1],[2,-1,1,-1],[1,1,-1,-1]]
//tran(R)*A*R=B si tridiagivens(A)=R,B
//pour annuler le terme 2,0 on multiplie A par TG
//TG[1,1]=cos(t)=TG[2,2],TG[1,2]=sin(t)=-TG[2,1],
//avec -sin(t)A[1,0]+cos(t)A[2,0]=0
//TG*A multiplie par tran(TG) annule les termes 2,0 et 0,2
//pour annuler le terme q,p (q>p+1) on multiplie A par TG
//TG[p+1,p+1]=cos(t)=TG[q,q],TG[p+1,q]=sin(t)=-TG[q,p+1],
//avec sin(t)A[p+1,p]=cos(t)A[q,p]
//donc sin(t)=A[q,p]/r et cos(t)=A[p+1,p]/r
//avec r:=sqrt((A[p+1,p])^2+(A[q,p])^2)
tridiagivens(A):={
  local n,p,q,r,TG,R,c,s;
  n:=size(A);
  R:=idn(n);
  for (p:=0;p<n-2;p++) {
    for (q:=p+2;q<n;q++) {
    r:=sqrt((A[p+1,p])^2+(A[q,p])^2);
    if (r!=0) {
    c:=normal(A[p+1,p]/r);
    s:=normal(A[q,p]/r);
    TG:=idn(n);
    TG[p+1,p+1]:=c;
    TG[q,q]:=c;
    TG[p+1,q]:=s;
    TG[q,p+1]:=-s;
    TG:=normal(TG);
    A:=normal(TG*A);
    A:=normal(A*tran(TG));
    A:=normal(A);
    R:=normal(R*tran(TG));
    }
    }
  }
  return (R,A);
}
```

On tape :

```
A:=[[1,-1,2,1],[-1,1,-1,1],[2,-1,1,-1],[1,1,-1,-1]]
tridiagivens(A)
```

On obtient :

```
[[1,0,0,0],[0,(-(sqrt(6)))/6,
(-(sqrt(4838400000)))/201600,
(-(sqrt(2962400000)))/64400],
[0,sqrt(6)/3,sqrt(4838400000)/252000,
(-(sqrt(26661600000)))/322000],
```

```
[0,sqrt(6)/6,(-(26*sqrt(210)))/420,12*sqrt(35)/420]],
[[1,sqrt(6),0,0],
[sqrt(6),1/3,sqrt(275990400000)/266400,0],
[0,sqrt(209026944000)/231840,73/105,8*sqrt(6)/35],
[0,0,8*sqrt(6)/35,1/-35]]
```

On tape :
`A:=[[1,-1,2,0],[-1,1,-1,1],[2,-1,1,-1],[1,1,-1,-1]]`
`tridiagivens(A)`
On obtient :

```
[[1,0,0,0],[0,(-(sqrt(6)))/6,
(-(sqrt(4838400000)))/201600,(-(sqrt(2962400000)))/64400],
[0,sqrt(6)/3,sqrt(4838400000)/252000,
(-(sqrt(26661600000)))/322000],
[0,sqrt(6)/6,(-(26*sqrt(210)))/420,12*sqrt(35)/420]],
[[1,5*sqrt(6)/6,13*sqrt(210)/210,(-(sqrt(35)))/35],
[sqrt(6),1/3,sqrt(275990400000)/266400,0],
[0,sqrt(209026944000)/231840,73/105,8*sqrt(6)/35],
[0,0,8*sqrt(6)/35,1/-35]]
```

14.7 Tridiagonalisation des matrices symétriques avec Householder

On a le théorème : Pour toute matrice symétrique A d'ordre n, il existe une matrice P, produit de $n-2$ matrice de Householder (donc $P^{-1} = {}^tP$), telle que $B = {}^tP * A * P$ soit tridiagonale.
Si A n'est pas symétrique, il existe une matrice P, produit de $n-2$ matrice de Householder, telle que $B = {}^tP * A * P$ soit un matrice de Hessenberg c'est à dire une matrice dont les coefficient sous-tridiagonaux sonts nuls : c'est la réduction de Householder.
On va dans cette section écrire un programme qui renverra P et B.

14.7.1 Matrice de Householder associée à `v`

Soit un hyperplan défini par son vecteur normal `v`.
On appelle matrice de Householder, la matrice d'une symétrie orthogonale `hv` par rapport à cet hyperplan. On a :
`hv(u):=u-2(dot(v,u))/norm(v)^2*v`
et la matrice de Householder associée est :
`Hv=idn(n)-2*tran(v)*[v]/(v*v).`
On a :
`Hv=tran(Hv)=inv(Hv)` Par convention la matrice unité est une matrice de Householder.
On écrit le programme houdeholder qui à un vecteur `v` renvoie la matrice de Householder associé à `v`.

```
householder(v):={
  local n,w,nv;
```

```
  n:=size(v);
  nv:=v*v;
  w:=tran(v);
  return normal(idn(n)-2*w*[v]/nv);
};
```

14.7.2 Matrice de Householder annulant les dernières composantes de a

Soit Hv la matrice de Householder associée à v.
Etant donné un vecteur a non nul, il existe deux vecteurs v tels que le vecteur b=Hv*tran(a) a ses $n-1$ dernières composantes nulles (Hv étant la matrice de Householder associé à v).
Plus précisement, si e0=[1,0,..0],e1=[0,1,0,..0] etc..., les vecteurs v sont :
v1=a+norm(a)*e0 et v2=a-norm(a)*e0.
Plus généralement, il existe deux vecteurs v tels que le vecteur b=Hv*tran(a) a ses composantes à partir de la (l+1)-ième nulles.
Plus précisement, si el=[0,..0,1,0,..0] (ou toutes les composantes sont nulles sauf el[l]=1), et si al=[0,...,0,a[l],..,a[n-1]], les vecteurs v sont :
v1=al+norm(al)*el et v2=al-norm(al)*el.
On ècrit le programme qui étant donné a et l renvoie si a est nul renvoie la matrice identité et si a est non nul renvoie la matrice de Householder associée à v=al+eps*norm(al)*el en choisisant eps égal au signe de al si al est réelle et eps=1 sinon.

```
//renvoie une matrice hh= H tel que b:=H*tran(a)
//a ses n-l-1 dernieres comp=0
//ou encore b[l+1]=...b[n-1]=0
make_house(a,l):={
  local n,na,el,v;
  n:=size(a);
  for (k:=0;k<l;k++) a[k]:=0;
  na:=normal(norm(a)^2);
  na:=sqrt(na);
  if (na==0) return idn(n);
  el:=makelist(0,1,n);
  el[l]:=1;
  if (im(a[l])==0 and a[l]<0) v:=normal(a-na*el);
    else v:=normal(a+na*el);
  return householder(v);
};
```

14.7.3 Réduction de Householder

Le programme tridiaghouse(A) renvoie H,B tel que normal(H*A*tran(H))=B (ou encore normal(tran(H)*B*H)=A) avec H est le produit de matrice de

householder (tran(H)=inv(H)) et `B` est une matrice de hessenberg (ou bien `B` est une matrice tridiagonale si `tran(A)=A`).

```
tridiaghouse(A):={
  local n, a, H,Ha;
  B:=A;
  n:=size(A);
  H:=idn(n);
  for (l:=0;l<n-2;l++) {
  a:=col(B,l);
  Ha:=make_house(a,l+1);
  B:=normal(Ha*B*Ha);
  H:=normal(Ha*H);
  }
  //normal(H*A*tran(H))=B et B de hessenberg
  //(ou tridiagonale si tran(A)=A)
  return(H,B);
};
```

On tape :

```
A:=[[3,2,2,2,2],[2,1,2,-1,-1],[2,2,1,-1,1],
[2,-1,-1,3,1],[2,-1,1,1,2]]
H,B:tridiaghouse(A)
```

On obtient :

```
[[1,0,0,0,0],[0,1/-2,1/-2,1/-2,1/-2],[0,5*sqrt(11)/22,
(-(3*sqrt(11)))/22,sqrt(11)/22,(-(3*sqrt(11)))/22],
[0,0,22*sqrt(2)/44,0,(-(22*sqrt(2)))/44],[0,22*sqrt(22)/242,
22*sqrt(22)/484,(-(22*sqrt(22)))/121,22*sqrt(22)/484]],
[[3,-4,0,0,0],[-4,9/4,sqrt(11)/4,0,0],[0,sqrt(11)/4,3/4,
44*sqrt(22)/121,0],[0,0,44*sqrt(22)/121,1/2,286*sqrt(11)/484],
[0,0,0,286*sqrt(11)/484,7/2]]
```

On vérifie et on tape :

```
normal(H*A*tran(H))
```

On tape :

```
A:=[[1,2,3],[2,3,4],[3,4,5]]
H,B:=tridiaghouse(A)
```

On obtient :

```
[[1,0,0],[0,(-(2*sqrt(13)))/13,(-(3*sqrt(13)))/13],
[0,(-(3*sqrt(13)))/13,(2*sqrt(13))/13]],
[[1,-(sqrt(13)),0],[-(sqrt(13)),105/13,8/13],[0,8/13,(-1)/13]]
```

On vérifie et on tape :

```
normal(H*A*tran(H))
```

Chapitre 15

Le calcul intégral et les équations différentielles

15.1 La méthode des trapèzes et du point milieu pour calculer une aire

Dans Xcas, il existe déjà une fonction qui calcule la valeur approchée d'une intégrale (en accélérant la méthode des trapèzes par l'algorithme de Romberg) qui est : romberg
Soit une fonction définie et continue sur l'intervalle $[a\ ;\ b]$.
On sait que $\int_a^b f(t)dt$ peut être approchée par l'aire, soit :
- du trapèze de sommets $a, b, b+i*f(b), a+i*f(a)$ soit,
- du rectangle de sommets $a, b, b+i*f((a+b)/2), a+i*f((a+b)/2)$.
La méthode des trapèzes (resp point-milieu) consiste à partager l'intervalle $[a\ ;\ b]$ en n parties égales et à faire l'approximation de l'intégrale sur chaque sous-intervalle par l'aire des trapèzes (resp rectangles) ainsi définis.

15.1.1 La méthode des trapèzes

On partage $[a\ ;\ b]$ en n parties égales et trapeze renvoie la somme des aires des n trapèzes déterminés par la courbe représentative de f et la subdivision de $[a\ ;\ b]$.

```
trapeze(f,a,b,n):={
local s,k;
s:=evalf((f(a)+f(b))/2);
for (k:=1;k<n;k++) {
s:=s+f(a+k*(b-a)/n);
}
return s/n*(b-a);
}
```

On partage $[a\ ;\ b]$ en 2^p parties égales et trapezel renvoie la liste de la somme des aires des $n = 2^k$ trapèzes déterminés par la courbe représentative de f et la subdivision de $[a\ ;\ b]$, liste de longueur $p+1$, obtenue en faisant varier k de 0 à p.
On pourra ensuite, appliquer à cette liste une accélération de convergence.

On remarquera qu'à chaque étape, on ajoute des "nouveaux" points à la subdivision, et que le calcul utilise la somme s précédente en lui ajoutant la contribution s_1 des "nouveaux" points de la subdivision.

```
trapezel(f,a,b,p):={
local s,n,k,lt,s1,j;
s:=evalf((f(a)+f(b))/2);
n:=1;
lt:=[s*(b-a)];
for (k:=1;k<=p;k++) {
s1:=0;
for (j:=0;j<n;j++) {
s1:=s1+f(a+(2*j+1)*(b-a)/(2*n));
}
s:=s+s1;
n:=2*n;
lt:=concat(lt,s/n*(b-a));
}
return lt;
}
```

On met ce programme dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`), puis on le teste et on le valide avec `OK`.
On tape (on partage [0 ;1] en 2^6=64 parties égales) :
`trapezel(x->x^2+1,0,1,6)`
On obtient :
`[1.5,1.375,1.34375,1.3359375,1.333984375,1.33349609375,`
`1.33337402344]`
On sait que `int(x^2+1,x,0,1)`=$\frac{4}{3}$=`1.3333333333`
On tape (on partage [0 ;1] en 2^6=64 parties égales) :
`trapezel(exp,0,1,6)`
On obtient :
`[1.85914091423,1.75393109246,1.72722190456,1.72051859216,`
`1.71884112858,1.71842166032,1.71831678685]`
On sait que `int(exp(x),x,0,1)=e-1=1.71828182846`

15.1.2 La méthode du point milieu

On partage $[a\ ;\ b]$ en n parties égales et `ptmilieu` renvoie la somme des aires des n rectangles déterminés par la courbe représentative de f et les milieux des segments de la subdivision de $[a\ ;\ b]$.

```
ptmilieu(f,a,b,n):={
local s,k;
s:=0.0;
for (k:=0;k<n;k++) {
s:=s+f(a+(b-a)/(2*n)+k*(b-a)/n);
}
return s/n*(b-a);
}
```

On partage $[a\ ;\ b]$ en 3^p parties égales et `ptmilieul` renvoie la liste de la somme des aires des $n = 3^k$ rectangles déterminés par la courbe représentative de f et les milieux de la subdivision de $[a\ ;\ b]$, liste de longueur $p+1$, obtenue en faisant varier k de 0 à p.
On peut ensuite appliquer à cette liste une accélération de convergence.
On remarquera qu'à chaque étape, on ajoute des "nouveaux" points à la subdivision, et, que le calcul utilise la somme s précédente en lui ajoutant la contribution s_1 des "nouveaux" points de la subdivision.

```
ptmilieul(f,a,b,p):={
local s,n,k,lt,s1,j;
s:=evalf(f((a+b)/2));
n:=1;
lt:=[s*(b-a)];
for (k:=1;k<=p;k++) {
s1:=0.0;
for (j:=0;j<n;j++) {
s1:=s1+f(a+(6*j+1)*(b-a)/(6*n))+f(a+(6*j+5)*(b-a)/(6*n));
}
s:=s+s1;
n:=3*n;
lt:=concat(lt,s*(b-a)/n);
}
return lt;
}
```

On met ce programme dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`), puis on le teste et on le valide avec `OK`.
On tape (on partage [0 ;1] en 3^4=81 parties égales) :
`ptmilieul(x->x^2+1,0,1,4)`
On obtient :
`[1.25,1.32407407407,1.33230452675,1.33321902149,1.33332063202]`
On sait que `int(x^2+1,x,0,1)`=$\frac{4}{3}$=`1.3333333333`
On tape (on partage [0 ;1] en 3^4=81 parties égales) :
`ptmilieul(exp,0,1,4)`
On obtient :
`[1.6487212707,1.71035252482,1.7173982568,`
`1.71818362241,1.71827091629]`
On sait que `int(exp(x),x,0,1)=e-1=1.71828182846`

15.2 Accélération de convergence : méthode de Richardson et Romberg

Hypothèse : soit v une fonction qui admet un développement limité au voisinage de zéro à l'ordre n :
$v(h) = v(0) + c_1 \cdot h + c_2 \cdot h + ... + c_n \cdot h^n + O(h^{n+1})$
On veut accélérer la convergence de v vers $v(0)$ quand h tend vers 0.

15.2.1 La méthode de Richardson

Le principe

La méthode de Richardson consiste à faire dispataitre le terme en $c_1 \cdot h$ en faisant par exemple la combinaison $2 \cdot v(h/2) - v(h)$.
On a en effet :
$v(h) = v(0) + c_1 \cdot h + c_2 \cdot h^2 + ... + c_n \cdot h^n + O(h^{n+1})$
$v(\frac{h}{2}) = v(0) + c_1 \cdot \frac{h}{2} + c_2 \cdot \frac{h^2}{4} + ... + c_n \cdot \frac{h^2}{2^n} + O(h^{n+1})$
Posons :
$u(h) = 2 \cdot v(h/2) - v(h)$
On a donc $u(h) = v(0) - c_2 \cdot \frac{h^2}{2} + ... - c_n \cdot h^n \cdot \frac{2^{n-1}-1}{2^{n-1}} + O(h^{n+1})$
u tend donc plus vite que v vers $v(0)$ quand h tend vers 0.
On peut continuer et faire subir la même chose à u.

L'algorithme général

On considère $r \in]0;1[$ (on peut prendre $r = \frac{1}{2}$) et on pose :
$v_{0,0}(h) = v(h) = v(0) + c_1 \cdot h + c_2 \cdot h^2 + ... + c_n \cdot h^n + O(h^{n+1})$
$v_{1,0}(h) = v(r \cdot h) = v(0) + c_1 \cdot r \cdot h + c_2 \cdot r^2 \cdot h^2 + ... + c_n \cdot r^n \cdot h^n + O(h^{n+1})$
$v_{2,0}(h) = v(r^2 \cdot h) = v(0) + c_1 \cdot r^2 \cdot h + c_2 \cdot r^4 \cdot h^2 + ... + c_n \cdot r^{2n} \cdot h^n + O(h^{n+1})$
Soit :
$v_{1,1}(h) = \frac{1}{1-r}(v(r \cdot h) - r \cdot v(h)) = \frac{1}{1-r}(v_{1,0}(h) - r \cdot v_{0,0}(h))$
$v_{1,1}(h) = v(0) + c_2 \cdot r \cdot h^2 + ... + c_n \cdot r \cdot (r^n - 1)/(r-1) \cdot h^n + O(h^{n+1})$
on on n'a pas de terme en h dans $v_{1,1}(h)$ et de la même façon en posant :
$v_{2,1}(h) = v_{1,1}(r \cdot h) = \frac{1}{1-r}(v(r^2 \cdot h) - r \cdot v_{1,0}(h)) = \frac{1}{1-r}(v_{2,0}(h) - r \cdot v_{1,0}(h))$
$v_{2,1}(h) = v(0) + c_2 \cdot r^3 \cdot h^2 + ... + O(h^{n+1})$
on n'a pas de terme en h dans $v_{2,1}(h)$
On obtient la suite des fonctions $v_{k,1} = v_{1,1}(r^{k-1} \cdot h)$ $(k > 1)$ qui n'ont pas de terme en h dans leur développements limités et qui convergent vers v(0).
On peut faire subir à la suite $v_{k,1}$ le même sort qu'à la suite $v_{k,0}$ et obtenir la suite $v_{k,2}$ $(k > 2)$:
on pose $v_{2,2}(h) = \frac{1}{1-r^2}(v_{2,1}(h) - r^2 \cdot v_{1,1}(h))$
et on n'a pas de terme en h^2 dans $v_{2,2}(h)$ etc....
On obtient ainsi les formules de récurrence :

$$v_{k,0}(h) = v(r^k \cdot h)$$

$$v_{k,p}(h) = \frac{1}{1-r^p}(v_{k,p-1}(h) - r^p \cdot v_{k-1,p-1}(h))$$

On a alors :

$$v_{k,p}(h) = v(0) + O(h^{p+1})$$

15.2.2 Application au calcul de S=$\sum_{k=1}^{\infty} \frac{1}{k^2}$

On a :

$$R_p = S - \sum_{k=1}^{p} \frac{1}{k^2} = \sum_{k=p+1}^{\infty} \frac{1}{k)2}$$

D'après la comparaison avec une intégrale on a :

$$\int_{p+1}^{\infty} \frac{1}{x^2} dx < R_p < \int_{p}^{\infty} \frac{1}{x^2} dx$$

Donc

$$\frac{1}{p+1} < R_p < \frac{1}{p}$$

donc $S_{0,p} = S_p = S - R_p = S - \frac{1}{p} + O(\frac{1}{p^2})$
On forme pour $p = 1..2^{n-1}$:
$S_{1,p} = S1_p = 2 * S_{2p} - S_p$ puis,
$S_{2,p} = S2_p = (2^2 * S1_{2p} - S1_p)/(2^2 - 1)$ puis,
$S_{k,p} = (2^k * S_{k-1,2p} - S_{k-1,p})/(2^k - 1)$
À la main :
$S_{0,1} = 1$,
$S_{0,2} = 5/4 = 1.25$,
$S_{0,3} = 49/36 = 1.36111111$,
$S_{0,4} = 205/144 = 1.42361111111$
$S_{1,1} = 2 * S_{0,2} - S_{0,1} = 3/2 = 1.5$
$S_{1,2} = 2 * S_{0,4} - S_{0,2} = 1.59722222222$
$S_{2,2} = (4 * S_{1,2} - S_{1,1})/3 = 1.62962962963$
Dans la liste S on met $1, 1 + 1/4, 1 + 1/4 + 1/9, ...1 + 1/4 + ... + 1/2^{2n}$,
S a 2^n termes d'indices 0 à $2^n - 1$,
puis on forme
$S1 = 1.5, 1 + 1/4 + 2/9 + 2/16, ...S[2^n - 1] - S[2^{n-1} - 1]$
(on doit mettre -1 dans $S[2^n - 1] - S[2^{n-1} - 1]$ car les indices de S commencent à 0)
$S1$ a 2^{n-1} termes d'indices 0 à $2^{n-1} - 1$
puis on continue avec une suite $S2$ de termes pour $k = 1..2^{n-2}$:
$(4 * S1[2 * k - 1] - S1[k - 1])/3$ etc...
On écrit le programme suivant :

```
richardson(n):={
local s0,s1,k,j,st,S,puiss;
s0:=[1.];
st:=1.0;
for (k:=2;k<=2^n;k++) {
st:=st+1/k^2;
s0:=concat(s0,st);
}
//attention s0=S a 2^n termes d'indices 0 (2^n)-1
S:=s0;
for (j:=1;j<=n;j++){
  s1:=[];
  puiss:=2^j;
  //j-ieme acceleration s1 a 2^(n-j) termes d'indices
  // allant de 0 a 2^(n-j)-1
  for (k:=1;k<=2^(n-j);k++) {
     st:=(puiss*s0[2*k-1]-s0[k-1])/(puiss-1);
```

```
      s1:=concat(s1,st);
   }
   s0:=s1;
}
return S[2^n-1],s1[0];
}:;
```

La première valeur est la somme des 2^n premiers termes, calculée sans accélération , la deuxième valeur a été obtenue après avoir accéléré cette somme n fois.
On tape :
`richardson(6)`
On obtient :
`1.62943050141, 1.64493406732`
On a du calculer $2^6 = 64$ termes (1 et 8 décimales exactes).
On tape :
`richardson(8)`
On obtient avec plus de décimales :
`1.6410354363087,1.6449340668481` (2 et 12 décimales exactes)
On a du calculer $2^8 = 256$ termes.
On sait que $S = \sum_{k=1}^{\infty} \frac{1}{k^2} = \pi^2/6 \simeq 1.6449340668482$

15.2.3 Application au calcul de la constante d'Euler

Par définition la constante d'Euler γ est la limite quand n tend vers l'infini de :

$$u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$$

(voir aussi 13.6.5 et 13.8.3)
On a donc : $\gamma = S = 1 + \sum_{k=2}^{+\infty} \frac{1}{k} + \ln(1 - \frac{1}{k})$ On a :

$$R_p = S - 1 - \sum_{k=2}^{p} \frac{1}{k} + \ln(\frac{k-1}{k}) = \sum_{k=p+1}^{\infty} \frac{1}{k} + \ln(1 - \frac{1}{k})$$

D'après la comparaison avec l'intégrale de la fonction décroissante $f(x) = \ln(x) - \ln(x-1) - 1/x$ pour $x \geq 1$ on a :

$$\int_{p+1}^{\infty} f(x)dx < -R_p < \int_{p}^{\infty} f(x)dx$$

Donc

$$p\ln(1 - \frac{1}{p+1}) + 1 < -R_p < (p-1)\ln(1 - \frac{1}{p}) + 1$$

donc $S_{0,p} = S_p = S - R_p = S - p\ln(1 - \frac{1}{p+1}) + 1 + O(\frac{1}{p^2}) = \frac{1}{2p} + O(\frac{1}{p^2})$
On écrit le programme suivant :

```
richard(n):={
local s0,s1,k,j,st,S,puiss;
s0:=[1.0];
st:=1.0;
```

```
for (k:=2;k<=2^n;k++) {
st:=st+1/k+evalf(ln(1-1/k),24);
s0:=concat(s0,st);
}
//attention s0=S a 2^n termes d'indices 0 (2^n)-1
S:=s0;puiss:=1;
for (j:=1;j<=n;j++){
  s1:=[];
  puiss:=2*puiss;
  //j-ieme acceleration s1 a 2^(n-j) termes d'indices
  // allant de 0 a 2^(n-j)-1
  for (k:=1;k<=2^(n-j);k++) {
     st:=(puiss*s0[2*k-1]-s0[k-1])/(puiss-1);
     s1:=concat(s1,st);
  }
  s0:=s1;
}
return S[puiss-1],s1[0];
}:;
```

La première valeur est la somme des 2^n premiers termes, calculée sans accélération , la deuxième valeur a été obtenue après avoir accéléré cette somme n fois.
Avec 24 digits, on tape :
`richard(8)`
On obtient :
`0.57916751833771789353914 64, 0.57721566490150504092605 31`
On a du calculer $2^6 = 64$ termes (1 et 8 décimales exactes).
On tape :
`richard(12)`
On obtient plus de décimales :
`0.57733773024697915893000 24, 0.57721566490153286060675 01`
(3 décimales exactes sans accélération et 21 décimales exactes avec accélération)
On a du calculer $2^{12} = 4096$ termes.
On tape : `Digits:=24;`
`evalf(euler_gamma)`
On obtient : `0.57721566490153286060651 19`

15.2.4 La constante d'Euler à epsilon près et la méthode de Richardson

La constante d'Euler est la limite dela suite u définit par :

```
u(n):={
  local res,j;
  res:=0;
  pour j de n jusque 1 pas -1 faire
    res:=res+1./j;
  fpour;
  return res-evalf(ln(n),24);
```

```
}:;
```

On va écrire la fonction `Richardson` qui va calculer la limite de la suite u à epsilon près avec 24 digits.
On ècrit la fonction en travaillant sur les lignes :

```
Richardson(u,eps):={
  local Lprec,Lnouv,n,j;
  n:=8;
  Lprec:=[u(n)];
  repeter
    n:=2*n;
    Lnouv:=[u(n)];
    pour j de 1 jusque size(Lprec) faire
      Lnouv[j]:=(2^j*Lnouv[j-1]-Lprec[j-1])/(2^j-1);
    fpour;
    Lprec:=Lnouv;
    afficher(Lprec);
  jusqua abs(Lnouv[size(Lnouv)-1]-Lnouv[size(Lnouv)-2])<eps;
  return Lnouv[size(Lnouv)-1];
}:;
```

On tape :
`Richardson(u,1e-9)`
On obtient :
`0.57721566490197573685976 92`
On tape :
`Richardson(u,1e-21)`
On obtient :
`0.57721566490153286060664 75`

15.2.5 La méthode de Romberg

C'est l'application de la méthode de Richardson à la formule des trapèzes dans le calcul d'une intégrale.

La formule d'Euler Mac Laurin

La formule à l'ordre 4

On a :
$\int_0^1 g(t)dt = \frac{1}{2}(g(0)+g(1))+bernoulli(2)(g'(0)-g'(1))+\frac{1}{4!}bernoulli(4)(g^{(3)}(0)-g^{(3)}(1)) + \int_0^1 P_4(t)g^{(4)}(t)dt$
où $P_4(t) = \frac{1}{4!}B_4(t)$ avec
$B_n(0) = bernoulli(n)$ et où B_n est le n-ième polynôme de Bernoulli.
La suite P_n est définie par :
$P_0 = 1$,et pour $k \geq 1$ $P_k' = P_{k-1}$ et $\int_0^1 P_k(u)du = 0$.
Pour la démonstration voir le fasicule Tableur, statistique à la section :
Suites adjacentes et convergence de $\displaystyle\sum_{k=0}^{n} \frac{(-1)^k}{2k+1}$.

Théorème pour la formule des trapèzes
Si $f \in \mathcal{C}^{2p+2}([a;b])$, il existe des constantes c_{2k} pour $k = 0..p$ telles que :

$$I = \int_a^b f(x)dx = T(h) + c_1h^2 + ..c_ph^{2p} + O(h^{2p+2})$$

avec

$$h = \frac{b-a}{n} \text{ et } T(h) = \frac{h}{2}(f(a) + f(b)) + h\sum_{k=1}^{n-1} f(a + k \cdot h)$$

On aura reconnu que $T(h)$ est la formule des trapèzes pour le calcul de $\int_a^b f(x)dx$ avec $h = \frac{b-a}{n}$.
Application de ce théorème pour la formule du point milieu
Soit un intervalle $[a; a+h]$ de longueur h si on applique la formule du point milieu à $I = \int_a^{a+h} f(t)dt$ on a $m(h) = h * f((a + a + h)/2)$ et si on applique la formule des trapèzes aux intervalles $[a; a + h/2]$ et $[a + h/2, a + h]$ on a $t(h/2) = h*(f(a)+f(a+h))/4+h*f((a+a+h)/2)/2 = t(h)/2+m(h)/2$ donc quand on coupe $[a; b]$ en n intervalles de longueur h si on note $M(h)$ la formule obtenue pour le point milieu et $T(h)$ celle obtenue pour les trapèzes, on a :
$M(h) = 2 * T(h/2) - T(h)$
D'après la formule d'Euler Mac Laurin :
$T(h) = I - c_1h^2 - c_2 * h^4 - .....c_p * h^{2p} + O(h^{2p+2})$ et donc
$M(h) = I + c_1/2 * h^2 + c_2 * h^4 * (2^3 - 1)/2^3 + ...c_p * h^{2p} * (2^{2p-1} - 1)/2^{2p-1} + O(h^{2p+2})$
On en déduit que les termes de ces deux développements sont de signes contraires et donc si on fait le même nombre d'accélération de convergence à $T(h)$ et à $M(h)$ on obtiendra un encadrement de I.

L'algorithme de Romberg

On applque l'algorithme de Richardson à $T(h)$ avec $r = \frac{1}{2}$.
On pose :

$$T_{n,0} = T(\frac{b-a}{2^n})$$

$$T_{n,1} = \frac{4T_{n,0} - T_{n-1,0}}{3}$$

$$T_{n,k} = \frac{2^{2k}T_{n,k-1} - T_{n-1,k-1}}{2^{2k} - 1}$$

Théorème :
On a :

$$T_{n,p} = I + O(\frac{1}{2^{2np}})$$

On partage successivement l'intervalle $[a; b]$ en $1 = 2^0, 2 = 2^1, 4 = 2^2, ..., 2^n$ et on calcule la formule des trapèzes correspondante : $T_{0,0}, T_{1,0}, .., T_{n,0}$ c'est ce que fait le programme `trapezel` fait en 15.1.1 que je recopie ci-dessous.

```
trapezel(f,a,b,n):={
local s,puiss2,k,lt,s1,j;
s:=evalf((f(a)+f(b))/2);
```

```
puiss2:=1;
lt:=[s*(b-a)];
for (k:=1;k<=n;k++) {
    s1:=0;
    for (j:=0;j<puiss2;j++) {
        s1:=s1+f(a+(2*j+1)*(b-a)/(2*puiss2));
    }
    s:=s+s1;
    puiss2:=2*puiss2;
    lt:=concat(lt,s*(b-a)/puiss2);
}
return lt;
}
```

On va travailler tout d'abord avec deux listes : $l0$ et $l1$ au début $l0 = [T_{0,0}]$ et $l1 = [T_{1,0}]$, on calcule $T_{1,1}$ et $l1 = [T_{1,0}, T_{1,1}]$, puis on n'a plus besoin de $l0$ donc on met $l1$ dans $l0$ et on recommence avec $l1 = [T_{2,0}]$, on calcule $T_{2,1}$ et $T_{2,2}$, et $l1 = [T_{2,0}, T_{2,1}, T_{2,2}]$ puis on met $l1$ dans $l0$ et on recommence avec.....pour enfin avoir $T_{n,0}$, $T_{n,1}$,... $T_{n,n}$ dans $l1$.

Les programmes

On applque l'algorithme de Romberg à $\mathtt{T_{k,0}} = \mathtt{l[k]}$ où $\mathtt{l} = \mathtt{trapezel(f,a,b,n)}$.

```
intt_romberg(f,a,b,n):={
local l,l0,l1,puis,k,j;
l:=trapezel(f,a,b,n);
//debut de l'acceleration de Romberg
l0:=[l[0]];
//on fait n accelerations
for (k:=1;k<=n;k++) {
   l1:=[l[k]];
   //calcul des T_{k,j} (j=1..k) dans l1
   for (j:=1;j<=k;j++) {
      puis:=2^(2*j);
      l1[j]:=(puis*l1[j-1]-l0[j-1])/(puis-1);
    }
    l0:=l1;
}
return l1;
}
```

On applque l'algorithme de Romberg à $\mathtt{M_{k,0}} = \mathtt{l[k]}$ où $\mathtt{l} = \mathtt{ptmilieul(f,a,b,n)}$ que je rappelle ci-dessous.

```
ptmilieul(f,a,b,n):={
local s,puiss3,k,lt,s1,j;
s:=evalf(f((a+b)/2));
puiss3:=1;
lt:=[s*(b-a)];
```

```
for (k:=1;k<=n;k++) {
s1:=0.0;
for (j:=0;j<puiss3;j++) {
s1:=s1+f(a+(6*j+1)*(b-a)/(6*puiss3))+f(a+(6*j+5)*(b-a)/(6*puiss3));
}
s:=s+s1;
puiss3:=3*puiss3;
lt:=concat(lt,s*(b-a)/puiss3);
}
return lt;
}
```

On procède comme précédemment en remplacant les puissances de 2 par des puissances de 3 et le calcul de $T(h)$ par celui de $M(h)$.

```
intm_romberg(f,a,b,n):={
local l,l0,l1,puis,k,j;
l:=milieul(f,a,b,n);
//debut de l'acceleration de Romberg
l0:=[l[0]];
//on fait n accelerations
for (k:=1;k<=n;k++) {
   l1:=[l[k]];
   //calcul des M_{k,j} (j=1..k) dans l1
   for (j:=1;j<=k;j++) {
      puis:=3^(2*j);
      l1[j]:=(puis*l1[j-1]-l0[j-1])/(puis-1);
    }
    l0:=l1;
}
return l1;
}
```

On peut raffiner en calculant `puis` $= 2^{(}2 * \mathtt{j})$ dans la boucle (en rajoutant `puis:=1;` avant `for (j:=1;j<=k;j++)..` et en remplacant `puis` $:= 2^{(}2 * \mathtt{j})$; par `puis:=puis*4;`) et aussi en n'utilisant qu'une seule liste...
On met ces programmes successivement dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`), puis on les teste et on les valide avec `OK`.
On tape (on partage [0 ;1] en 2^3=8 parties égales) :
`intt_romberg(x->x^2+1,0,1,3)`
On obtient :
`[[1.3359375,1.33333333333,1.33333333333,1.33333333333]`
On sait que `int(x^2+1,x,0,1)`=$\frac{4}{3}$=`1.3333333333`
On tape (on partage [0 ;1] en 2^4=16 parties égales) :
`intt_romberg(exp,0,1,4)`
On obtient :
`1.71884112858,1.71828197405,1.71828182868,`
`1.71828182846,1.71828182846]`
On sait que `int(exp(x),x,0,1)`=`e-1=1.71828182846`.

Dans la pratique, l'utilisateur ne connait pas la valeur de n qui donnera un résultat correct sans faire trop de calculs. C'est pourquoi, on va faire le calcul de la méthode des trapèzes au fur et à mesure et changer le test d'arrêt : on s'arrête quand la différence de deux termes consécutifs est en valeur absolue plus petit que $epsi$.

```
intrap_romberg(f,a,b,epsi):={
local l0,l1,puis,puiss2,k,j,s,s1,test;
//initialisation on a 1 intervalle
s:=evalf((f(a)+f(b))/2);
puiss2:=1;
l0:=[s*(b-a)];
k:=1;
test:=1;
while (test>epsi) {
   //calcul de la methode des trapezes avec 2^k intervalles
   s1:=0;
   for (j:=0;j<puiss2;j++) {
        s1:=s1+f(a+(2*j+1)*(b-a)/(2*puiss2));
   }
   s:=s+s1;
   puiss2:=2*puiss2;
   l1:=[s*(b-a)/puiss2];
   //debut de l'acceleration de Romberg
   //calcul des T_{k,j} (j=1..k) dans l1
   j:=1;
   while ((test>epsi) and (j<=k)) {
      puis:=2^(2*j);
      l1[j]:=(puis*l1[j-1]-l0[j-1])/(puis-1);
      test:=abs(l1[j]-l1[j-1]);
      j:=j+1;
    }
    l0:=l1;
    k:=k+1;
}
return [k-1,j-1,l1[j-1]];
}
```

On renvoie la liste $[p, q, val]$ où $val = T_{q,p}$ (p=le nombre d'accélérations).
On tape :
`intrap_romberg(sq,0,1,1e-12)`
On obtient :
`[2,2,0.333333333333]`
(on a donc dû partager [0 ;1] en 2^2=4 parties égales et on a fait 2 accélérations)
On tape :
`intrap_romberg(exp,0,1,1e-12)`
On obtient :
`[5,4,1.71828182846]`
(on a donc dû partager [0 ;1] en 2^5=32 parties égales et on a fait 4 accélérations)

On peut aussi appliquer l'algorithme de Romberg à $M(h)$ on le même programme en remplacant les puissances de 2 par des puissances de 3 et le calcul de $T(h)$ par celui de $M(h)$.

```
intmili_romberg(f,a,b,epsi):={
local l0,l1,puis,puiss3,k,j,s,s1,test;
//initialisation on a 1 intervalle
s:=evalf(f((a+b)/2));
puiss3:=1;
l0:=[s*(b-a)];
k:=1;
test:=1;
while (test>epsi) {
   //calcul de la methode du point milieu avec 3^k intervalles
   s1:=0;
   for (j:=0;j<puiss3;j++) {
      s1:=s1+f(a+(6*j+1)*(b-a)/(6*puiss3))+
          f(a+(6*j+5)*(b-a)/(6*puiss3));
   }
   s:=s+s1;
   puiss3:=3*puiss3;
   l1:=[s*(b-a)/puiss3];
   //debut de l'acceleration de Romberg
   //calcul des T_{k,j} (j=1..k) dans l1
   j:=1;
   while ((test>epsi) and (j<=k)) {
      puis:=3^(2*j);
      l1[j]:=(puis*l1[j-1]-l0[j-1])/(puis-1);
      test:=abs(l1[j]-l1[j-1]);
      j:=j+1;
    }
    l0:=l1;
    k:=k+1;
}
return [k-1,j-1,l1[j-1]];
}
```

On tape :
`inmili_romberg(sq,0,1,1e-12)`
On obtient :
`[2,2,0.333333333333]`
(on a donc dû partager [0 ;1] en 3^2=9 parties égales et on a fait 2 accélérations)
On tape :
`intmili_romberg(exp,0,1,1e-12)`
On obtient :
`[4,3,1.71828182846]`
(on a donc dû partager [0 ;1] en 3^4=81 parties égales et on a fait 3 accélérations)

Application au calcul de $\sum_{k=1}^{\infty} f(k)$

On peut aussi reprendre le programme sur les séries $\sum_{k=1}^{\infty} f(k)$ et écrire pour faire n accélérations :

```
serie_romberg(f,n):={
local l,l0,l1,puis,k,j,t,p;
// calcul des sommes s1,s2,s4,s8,...s2^n que l'on met ds l
 l:=[f(1)];
p:=1;
for (k:=1;k<=n;k++) {
t:=0.0
for (j:=p+1;j<=2*p;j++){
t:=t+f(j)
}
t:=l[k-1]+t;
l:= concat(l,t);
p:=2*p;
}
//debut de l'acceleration de Richardson
l0:=[l[0]];
for (k:=1;k<=n;k++) {
   l1:=[l[k]];
   //calcul des S_{k,j} (j=1..k) dans l1
   for (j:=1;j<=k;j++) {
      puis:=2^(j);
      l1[j]:=(puis*l1[j-1]-l0[j-1])/(puis-1);
    }
    l0:=l1;
}
return l1;
}
```

On met ce programme dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`), puis on le teste et on le valide avec `OK`.
On définit f :
On tape :
`f(x):=1/x^2`
On tape (on calcule $\sum_{k=1}^{6} 4f(k)$) :
`serie_romberg(f,6)`
On obtient cette somme après avoir subit 0,1,..6 accélérations :
`[1.62943050141,1.64469373999,1.64492898925,1.64493403752,`
`1.64493409536,1.64493407424,1.64493406732]`
Avec le programme `richardson` on avait :
`1.62943050141, 1.64493406732`
On tape (on calcule $\sum_{k=1}^{2} 56f(k)$) :
`serie_romberg(f,8)`
On obtient cette somme après avoir subit 0,1,..8 accélérations avec plus de déciamles :

```
[1.6410354363087,1.6449188676629,1.6449339873839,1.6449340668192,
1.6449340668791,1.6449340668489,1.6449340668477,1.644934066848,
1.6449340668481]
```

On sait que $\pi^2/6 = 1.6449340668482$

15.2.6 Deux approximations de l'intégrale

On calcule en même temps l'accélération de convergence pour la méthode des trapèzes et pour la méthode du point milieu.
On remarquera qu'ici on découpe l'intervalle $[a;b]$ en 2 puis en 2^2... 2^k morceaux et que le calcul de sm (somme des valeurs de f aux " points milieux") sert dans le calcul du st suivant (somme des valeurs de f aux "points de subdivision"+(f(a)+f(b))/2) comme contribution des nouveaux points.

```
inttm_romberg(f,a,b,epsi):={
local lt0,lt1,lm0,lm1,puis,puiss2,k,j,st,sm,s1,test;
//initialisation on a 1 intervalle
st:=evalf((f(a)+f(b))/2);
sm:=evalf(f((a+b)/2));
puiss2:=1;
lt0:=[st*(b-a)];
lm0:=[sm*(b-a)];
k:=1;
test:=1;
while (test>epsi) {
   //calcul de la methode des trapezes avec 2^k intervalles
    st:=st+sm;
   //calcul de la methode des milieux avec 2^k intervalles
   puiss2:=2*puiss2;
   s1:=0.0;
   for (j:=0;j<puiss2;j++) {
        s1:=s1+f(a+(2*j+1)*(b-a)/(2*puiss2));
   }

   sm:=s1;
   lm1:=[sm*(b-a)/puiss2];
   lt1:=[st*(b-a)/puiss2];
   //debut de l'acceleration de Romberg
   //calcul des T_{k,j} (j=1..k) dans lt1
   //calcul des M_{k,j} (j=1..k) dans lm1
   j:=1;
   while ((test>epsi) and (j<=k)) {
      puis:=2^(2*j);
      lt1[j]:=(puis*lt1[j-1]-lt0[j-1])/(puis-1);
      lm1[j]:=(puis*lm1[j-1]-lm0[j-1])/(puis-1);
      test:=abs(lt1[j]-lm1[j]);
      j:=j+1;
    }
    lt0:=lt1;
```

```
    lm0:=lm1;
    k:=k+1;
}
return [k-1,j-1,lt1[j-1],lm1[j-1]];
}
```

15.3 Les méthodes numériques pour résoudre $y' = f(x, y)$

Dans Xcas, il existe déjà les fonctions qui tracent les solutions de $y' = f(x, y)$, ce sont : `plotode, interactive_plotode` et une fonction `odesolve` qui calcule la valeur numérique en un point d'une solution de $y' = f(x, y)$ et $y(t_0) = y_0$ Soit f une fonction continue de $[a; b] \times \mathbb{R}$ dans $\mathbb{R}$.
On considère l'équation différentielle :

$$y(t_0) = y_0$$

$$y'(t) = f(t, y(t))$$

Problème de Cauchy Soit U un ouvert de $\mathbb{R}^2$ et f une application continue de U dans $\mathbb{R}$. On appelle solution du problème de Cauchy (E) :

$$y(t_0) = y_0$$

$$y'(t) = f(t, y(t))$$

tout couple (I, g) où I est un intervalle contenant t_0 et g est une I-solution de (E) c'est à dire une fonction de classe $C^1(I, \mathbb{R})$ vérifiant $g(t_0) = y_0$ et $g'(t) = f(t, g(t))$ pour $t \in I$.
Théorème de Cauchy-Lipschitz faible Soit f, une fonction continue de $[a; b] \times \mathbb{R}$ dans $\mathbb{R}$, lipschitzienne par rapport à la seconde variable c'est à dire :
il existe $K > 0$ tel que pour tout $t \in [a; b]$ et pour tout $(y_1, y_2) \in \mathbb{R}^2$, $|f(t, y_1) - f(t, y_2)| \leq K|y_1 - y_2|$.
Alors, quel que soit $(t_0, y_0) \in [a; b] \times \mathbb{R}$, il existe une $[a; b]$-solution unique au problème de Cauchy (E) que l'on appellera y.

15.3.1 La méthode d'Euler

La méthode d'Euler consiste à approcher la solution de cette équation différentielle au voisinage de t_0 par sa tangente en ce point. Cette tangente a comme pente $y'(t_0) = f(t_0, y(t_0)) = f(t_0, y_0)$.
On a donc pour t proche de t_0 par exemple pour $t \in [t_0 - h; t_0 + h]$:
$y(t) \simeq y_0 + f(t_0, y_0) \cdot (t - t_0)$: on dit que h est le pas de l'approximation.
En principe plus h est petit, meilleure est l'approximation.
Soit h un pas, on pose $t_1 = t_0 + h$, et $y_1 = y_0 + f(t_0, y_0) \cdot (t1 - t_0) = y_0 + f(t_0, y_0) \cdot h$
on a l'approximation :
$y(t_1) = y(t_0 + h) \simeq y_1$
et on réitère cette approximation, donc si $t_2 = t_1 + h$:
$y(t_2) = y(t_1 + h) \simeq y_1 + f(t_1, y_1) \cdot (t_2 - t_1) = \simeq y_1 + f(t_1, y_1) \cdot h$.
On écrit donc la fonction `euler_f`, qui réalise une seule étape de cette méthode et qui permet de passer d'un point d'abscisse t_0 au point d'abscisse $t_0 + h$, ces points étant situés sur le graphe de la solution approchée par cette méthode.

```
euler_f(f,t0,y0,h):={
local t1,y1;
t1:=t0+h;
y1:=y0+h*f(t0,y0);
return (t1,y1);
}
```

Pour tracer le graphe de la solution approchée sur $[t_0; t_1]$, on écrit :
`segment(point((t0,y0)),point(euler_f(f,t0,y0,h)))`

Pour trouver une solution approchée sur $[a; b]$ de l'équation différentielle :
$y'(t) = f(t, y(t)),\ \ y(t_0) = y_0$, avec $t_0 \in [a; b]$,
il suffit de partager $[a; b]$ en parties égales de longueur h et d'appliquer plusieurs fois la fonction `euler_f` avec le pas h puis avec le pas $-h$.
Voici le programme qui trace la solution sur $[a; b]$ dans l'écran `DispG` et qui utilise la fonction `euler_f`.
Les derniers segments servent à s'arrêter exactement en a et en b.

```
trace_sol(f,t0,y0,h,a,b):={
local t1,y1,td0,yd0,l1,j;
td0:=t0;
yd0:=y0;
h:=abs(h);
while (t0<b-h){
    l1:=euler_f(f,t0,y0,h);
    t1:=l1[0];
    y1:=l1[1];
    segment(t0+i*y0,t1+i*y1);
    t0:= t1;
    y0:=y1;
}
segment(t0+i*y0,b+i*(y0+(b-t0)*f(t0,y0)));
//on trace avec -h
t0:=td0;
y0:=yd0;
while ( t0>a+h){
    l1:=euler_f(f,t0,y0,-h);
    t1:=l1[0];
    y1:=l1[1];
    segment(t0+i*y0,t1+i*y1);
    t0:= t1;
    y0:=y1;
}
segment(t0+i*y0,a+i*(y0+(t0-a)*f(t0,y0)));
}
```

ou encore pour avoir le dessin dans l'écran graphique obtenu comme réponse, on écrit une fonction qui renvoie la séquencee des segments à dessiner :

```
trace_euler(f,t0,y0,h,a,b):={
local td0,yd0,l1,j,ls;
td0:=t0;
yd0:=y0;
h:=abs(h);
ls:=[];
while (t0<b-h){
    l1:=euler_f(f,t0,y0,h);
    ls:=ls,segment(t0+i*y0,point(l1));
    (t0,y0):=l1;
}
ls:=ls,segment(t0+i*y0,point(euler_f(f,t0,y0,b-t0)));
//on trace avec -h en partant de td0 et de yd0
//(td0 et  yd0 sont les valeurs du debut)
t0:=td0;
y0:=yd0;
while ( t0>a+h){
    l1:=euler_f(f,t0,y0,-h);
    ls:=ls,segment(t0+i*y0,point(l1));
    (t0,y0):=l1;
}
ls:=ls,segment(t0+i*y0,point(euler_f(f,t0,y0,a-t0)));
return ls;
}
```

15.3.2 La méthode du point milieu

La méthode du point milieu consiste à approcher, au voisinage de t_0, la solution de l'équation différentielle $y'(t) = f(t, y(t)),\ \ y(t_0) = y_0\ t_0 \in [a; b]$, par une paralléle à la tangente au "point milieu de la solution obtenue par la méthode d'Euler" :
Le "point milieu de la solution obtenue par la méthode d'Euler" est le point de coordonnées :
$(t_0 + h/2, y_0 + h/2f(t_0, y_0))$
et sa tangente a donc comme pente :
$\alpha = f(t_0 + h/2, y_0 + h/2f(t_0, y_0))$
La méthode du point milieu fait passer du point (t_0, y_0) au point (t_1, y_1) avec :
$t_1 = t_0 + h$ et $y_1 = y_0 + h * \alpha = y_0 + h * f(t_0 + h/2, y_0 + h/2 * f(t_0, y_0))$.
On remarquera que :
`euler_f(f,t0,y0,h/2)=(t0+h/2,y0+h/2*f(t0,y0))`
On va donc écrire la fonction :
`ptmilieu_f(f,t0,y0,h)=(t0+h,y0+h*f(t0+h/2,y0+h/2*f(t0,y0)))`
qui réalise une seule étape de cette méthode et qui permet de passer d'un point d'abscisse t_0 au point d'abscisse $t_0 + h$, ces points étant situés sur le graphe de la solution approchée par cette méthode.

```
ptmilieu_f(f,t0,y0,h):={
local t1,y1;
t1:=t0+h;
```

```
y1:=y0+h*f(t0+h/2,y0+h/2*f(t0,y0));
return (t1,y1);
}
```

Il reste à écrire une fonction qui renvoie la séquence des segments obtenus par cette méthode, pour avoir le dessin dans l'écran graphique obtenu comme réponse.
On peut écrire la même fonction que précédemment en remplacant simplement tous les appels à `euler_f` par `ptmilieu_f`.
Mais on va plutôt rajouter un paramètre supplémentaire `methode` qui sera soit la fonction `euler_f` soit la fonction `ptmilieu_f`. On écrit :
`trace_methode(methode,f,t0,y0,h,a,b)`
où `methode` qui est une fonction des variables `f,t0,y0,h`

```
trace_methode(methode,f,t0,y0,h,a,b):={
local td0,yd0,l1,j,ls;
td0:=t0;
yd0:=y0;
h:=abs(h);
ls:=[];
while (t0<b-h){
    l1:=methode(f,t0,y0,h);
    ls:=ls,segment(t0++i*y0,point(l1));
    (t0,y0):=l1;
}
ls:=ls,segment(t0+i*y0,point(methode(f,t0,y0,b-t0)));
//on trace avec -h en partant de td0 et de yd0
//(td0 et yd0 sont les valeurs du debut)
t0:=td0;
y0:=yd0;
while ( t0>a+h){
    l1:=methode(f,t0,y0,-h);
    ls:=ls,segment(t0+i*y0,point(l1));
    (t0,y0):=l1;
}
ls:=ls,segment(t0+i*y0,point(methode(f,t0,y0,a-t0)));
return ls;
}
```

15.3.3 La méthode de Heun

La méthode de Heun consiste à approcher, au voisinage de t_0, la solution de l'équation différentielle $y'(t) = f(t, y(t)),\ \ y(t_0) = y_0\ t_0 \in [a; b]$, par une parallèle à la droite de pente $1/2 * (f(t_0, y_0) + f(t_0 + h, y_0 + h * f(t_0, y_0)))$ c'est à dire, par une pente égale à la moyenne des pentes des tangentes :
- de la solution au point (t_0, y_0) (la pente de la tangente en ce point vaut $f(t_0, y_0)$) et
- de la solution obtenue par la méthode d'Euler au point d'abscisse $t_0 + h$ (point de coordonnées $(t_0 + h, y_0 + h * f(t_0, y_0))$ et la pente de la tangente en ce point vaut $f(t_0 + h, y_0 + h * f(t_0, y_0))$).

Donc, la méthode de Heun fait passer du point (t_0, y_0) au point (t_1, y_1) avec :
$t_1 = t_0 + h$ et $y_1 = y_0 + h/2 * (f(t_0, y_0) + f(t_0 + h, y_0 + h * f(t_0, y_0)))$.
On remarquera que :
`euler_f(f,t0,y0,h)=(t0+h,y0+h*f(t0,y0))`
On va donc écrire la fonction :
`heun_f(f,t0,y0,h)=`
`(t0+h,y0+h/2*(f(t0,y0)+f(t0+h,y0+h*f(t0,y0))))`
qui réalise une seule étape de cette méthode et qui permet de passer d'un point d'abscisse t_0 au point d'abscisse $t_0 + h$, ces points étant situés sur le graphe de la solution approchée par cette méthode.

```
heun_f(f,t0,y0,h):={
local t1,y1;
t1:=t0+h;
y1:=y0+h/2*(f(t0,y0)+f(t0+h,y0+h*f(t0,y0)));
return (t1,y1);
}
```

Il reste à écrire une fonction qui renvoie la séquence des segments obtenus par cette méthode, pour avoir le dessin dans l'écran graphique obtenu comme réponse.
On peut écrit la même fonction que précédemment en remplacant simplement tous les appels à `euler_f` par `heun_f` ou encore utiliser la fonction `trace_methode` avec comme valeur de `methode` le fonction `heun_f`.
On valide les differentes fonctions :
`euler_f`, `ptmilieu_f`, `heun_f` et `trace_methode`.
On tape par exemple pour avoir le tracé de la solution sur [-1 ;1] de :
$y' = y$ vérifiant $y(0) = 1$ par les 3 méthodes avec un pas de 0.1 :
`f(t,y):=y trace_methode(euler_f,f,0,1,0.1,-1,1)` et
`trace_methode(ptmilieu_f,f,0,1,0.1,-1,1)` ou
`trace_methode(heun_f,f,0,1,0.1,-1,1)` ou

Chapitre 16

Exercice : Les courses poursuites

16.1 Le chien qui va en direction de son maitre

On chien se trouve en C de coordonnées (0,1) et son maitre se trouve en M de coordonnées (0,0).
Le maitre se déplace sur l'axe des x et le chien se dirige à la même vitesse en direction de son maitre.
On veut dessiner la trajectoire du chien.

```
chien():={
local X,Y,L,xk,yk,p,j,k,d,a,a1,b,b1;
DispG;
ClrGraph();
p:=0.1;
X:=0,0,3;
Y:=1,0,0;
L:=NULL;
pour k de 0 jusque 1 faire
L:=L,point(X[k]+i*Y[k],affichage=1+epaisseur_point_2+point_point);
fpour;
tantque abs(X[0]+i*Y[0]-X[1]-i*Y[1])>0.1 faire
pour k de 0 jusque 1 faire
a:=X[k];a1:=X[k+1]-a;
b:=Y[k];b1:=Y[k+1]-b;
d:=sqrt(a1^2+b1^2);
xk:=a+p*a1/d;
yk:=b+p*b1/d;
L:=L,segment(a+i*b,xk+i*yk,affichage=k);
X[k]:=xk;
Y[k]:=yk;
fpour;
ftantque;
return L;
}
:;
```

16.2 Avec les sommets d'isopolygônes

Des personnes $A, B, C..$ sont situées aux sommets $A, B, C..$ d'un isoploygone. Á l'instant t, A se dirige vers B, B se dirige vers C, C se dirige vers D,...et le dernier sommet se dirige vers A.
On veut dessiner les trajectoires de $A, B, C....$

```
isopoly(n,m):={
local X,Y,L,xk,yk,p,j,k,d,a,a1,b,b1,P,SP;
p:=0.1;
L:=NULL;
P:=isopolygone(0,1,n);
SP:=op(sommets(P));
X:=evalf(abscisse(SP));
Y:=evalf(ordonnee(SP));
pour k de 0 jusque n-1 faire
L:=L,point(X[k]+i*Y[k],affichage=1+epaisseur_point_2+point_point);
fpour;
pour j de 0 jusque m-1 faire
X[n]:=X[0];
Y[n]:=Y[0];
pour k de 0 jusque n-1 faire
a:=X[k];a1:=X[k+1]-a;
b:=Y[k];b1:=Y[k+1]-b;
d:=sqrt(a1^2+b1^2);
xk:=a+p*a1/d;
yk:=b+p*b1/d;
L:=L,segment(a+i*b,xk+i*yk,affichage=k+89);
X[k]:=xk;
Y[k]:=yk;
fpour;
fpour;
return L;
}
:;
```

16.3 Avec les sommets de polygônes quelconques

Des personnes $A, B, C...$ sont situées aux sommets d'un ploygone $A, B, C...$ dont les affixes sont les éléments de la liste Z.
Á l'instant t, A se dirige vers B, B se dirige vers C, C se dirige vers D,...et le dernier sommet se dirige vers A.
On veut dessiner les trajectoires de $A, B, C....$

```
poly(Z):={
local X,Y,L,xk,yk,p,j,k,d,a,a1,b,b1,n,m;
p:=0.1;
L:=NULL;
```

```
m:=100;
n:=size(Z);
X:=re(Z);
Y:=im(Z);
pour k de 0 jusque n-1 faire
L:=L,point(X[k]+i*Y[k],affichage=1+epaisseur_point_2+point_point);
fpour;
pour j de 0 jusque m-1 faire
X[n]:=X[0];
Y[n]:=Y[0];
pour k de 0 jusque n-1 faire
a:=X[k];a1:=X[k+1]-a;
b:=Y[k];b1:=Y[k+1]-b;
d:=sqrt(a1^2+b1^2);
xk:=a+p*a1/d;
yk:=b+p*b1/d;
L:=L,segment(a+i*b,xk+i*yk,affichage=k+89);
X[k]:=xk;
Y[k]:=yk;
fpour;
fpour;
return L;
}
:;
```

16.4 Avec des points aléatoires

Des souris $A, B, C...$ sont situées aux points $A, B, C...$ dont les coordonnées sont les éléments des listes X et Y définies aléatoirement entre -1 et 1.
Á l'instant t, A se dirige vers B, B se dirige vers C, C se dirige vers D,...et le dernier sommet se dirige vers A.
On veut dessiner les trajectoires de $A, B, C....$

```
courbot0():={
local X,Y,L,xk,yk,p,m,n,j,k,d,a,a1,b,b1;
DispG;
ClrGraph();
p:=0.1;
m:=10;
n:=50;
X:=NULL;
Y:=NULL;
L:=NULL;
pour k de 0 jusque n-1 faire
xk:=2*alea(0,1)-1;
yk:=2*alea(0,1)-1;
X:=X,xk;
Y:=Y,yk;
```

```
L:=L,point(xk,yk,affichage=1+epaisseur_point_2+point_point);
fpour;
Pause 1;
pour j de 0 jusque m-1 faire
X[n]:=X[0];
Y[n]:=Y[0];
pour k de 0 jusque n-1 faire
a:=X[k];a1:=X[k+1]-a;
b:=Y[k];b1:=Y[k+1]-b;
d:=sqrt(a1^2+b1^2);
xk:=a+p*a1/d;
yk:=b+p*b1/d;
L:=L,segment(a+i*b,xk+i*yk,affichage=k+90);
X[k]:=xk;
Y[k]:=yk;
fpour;
Pause 1;
fpour;
return L;
}
:;
courbot():={
local xt,yt,X,Y,L,xk,yk,p,m,n,j,k,d,a,a1,b,b1;
ClrGraph();
p:=0.01;
m:=100;
n:=50;
X:=NULL;
Y:=NULL;
L:=NULL;
pour k de 0 jusque n-1 faire
xk:=2*alea(0,1)-1;
yk:=2*alea(0,1)-1;
X:=X,xk;
Y:=Y,yk;
L:=L,point(xk,yk,affichage=1+epaisseur_point_2+point_point);
fpour;
pour j de 0 jusque m-1 faire
X[n]:=X[0];
Y[n]:=Y[0];
pour k de 0 jusque n-1 faire
a:=X[k];a1:=X[k+1]-a;
b:=Y[k];b1:=Y[k+1]-b;
d:=sqrt(a1^2+b1^2);
xt:=a+p*a1/d ;
yt:=b+p*b1/d;
L:=L,segment(a+i*b,xt+i*yt,affichage=k+90);
X[k]:=xt;
```

```
Y[k]:=yt;
fpour;
fpour;
return L;
}:;
```

Chapitre 17

Les quadriques

17.1 Équation d'une quadrique

Une forme quadratique de 4 variables `X,Y,Z,T` peut s'interpréter dans l'espace projectif `Oxyz` comme le premier membre de l'équation d'une quadrique.
Par exemple soit :
$q(x,y,z) := 4*x^2+y^2+z^2-4*x*y+4*x*z-2*y*z+8*x-4*y+4*z+2$
`q(x,y,z)=0` est l'équation d'une quadrique.
On associe à `q` la forme quadradique `Q` :
`Q(x,y,z,t):=normal(t^2*normal(q(x/t,y/t,z/t)))`
et la matrice carrée `A` d'ordre 4 :
`A:=q2a(Q(x,y,z,t),[x,y,z,t])`
On obtient :
`A:=[[4,-2,2,4],[-2,1,-1,-2],[2,-1,1,2],[4,-2,2,2]]`
on retrouve la forme `Q` à partir de `A` si :
`v:=[x,y,z,t]`
la forme quadratique est :
`Q(op(v)):=normal((v*A*tran(v))[0])` ou plus simplement :
`Q(op(v)):=normal(v*A*v)`
on retrouve la forme `Q` à partir de `A` si :
`v1:=[x,y,z,1]`
`q(op(v1)):=normal(v1*A*v1)`
et `Q(v)=0` représente une quadrique `Q` de l'espace projectif.
Les points doubles de `Q` vérifie `A*tran(v)=[0,0,0,0]` ie `tran(v)` est un élément du noyau de `A`.
Discussion selon le rang `r` de `A` : `r:=rank(A)`

- Si `r=4`, `Q` est une quadrique propre ou non dégénérée sans point double.
- Si `r=3`, `Q` est un cône et a un point double `S` qui est le sommet du cône.
- Si `r=2`, `q` est le produit de 2 formes linéaires indépendantes et `Q` est formée de 2 plans distincts et secants selon la droite lieu des points doubles de `Q`.
- Si `r=1`, `q` est le carré d'une forme linéaire non nulle et `Q` est un plan de points doubles.

17.2 Équation reduite d'une quadrique

Pour réaliser la réduction de l'équation d'une quadrique, on pose :

```
B:=A[0..2,0..2];
C:=A[0..2,3];
d:=A[3,3];
```

Les directions propres de `B` sont les directions principales de la quadrique `Q`.
Comme une matrice symétrique réelle est diagonalisable dans un repère orthonormé, il existe un repère orthonormé (O, u, v, w) dans lequel la quadrique a comme équation :
$\mathtt{s1} * \mathtt{x1}^2 + \mathtt{s2} * \mathtt{y1}^2 + \mathtt{s3} * \mathtt{z1}^2 + 2\mathtt{c1} * \mathtt{x1} + 2 * \mathtt{c2} * \mathtt{y1} + 2 * \mathtt{c3} * \mathtt{z1} + \mathtt{d1} = 0$
où `s1,s2,s3` sont les valeurs propres de `B` et `u,v,w` sont les vecteurs propres associés, choisis normés et orthogonaux.

Remarques

- Si les 3 valeurs propres de B sont différentes : les vecteurs propres sont orthogonaux et il suffira de les normer.
- Si il y a une valeur propre double non nulle il faudra rendre les vecteurs propres orthogonaux et les normer.
- Si il y a une valeur propre double nulle il faut rendre les vecteurs propres orthogonaux et les normer.

Voici les différents cas :

- 1ier cas $s1 * s2 * s3! = 0$ supposons que signe($s1$)=signe($s2$)
  l'équation s'écrit :
  $s1 * (x1 + c1/s1)^2 + s2 * (y1 + c2/s2)^2 + s3 * (z1 + c3/s3)^2 + d2 = 0$
  en faisant un changement d'origine $O1$ avec :
  $OO1 = -c1/s1 * u - c2/s2 * v - c3/s3 * w$
  l'équation s'écrit :
  $s1 * X^2 + s2 * Y^2 + s3 * Z^2 + d2 = 0$
  quitte a multiplier par -1 on obtient une équation de la forme :
  $X^2/a^2 + Y^2/b^2 + s * Z^2/c^2 + d3 = 0$
  avec $s = 1$ ou $s = -1$ et $d3 = 1$ ou $d3 = -1$ ou $d3 = 0$.
  La quadrique est donc de révolution et est engendrée par la rotation autour de l'axe des Z de la conique d'équation $Y = 0, X^2/a^2 + s * Z^2/c^2 + d3 = 0$.
  On a donc :
  - $s = 1, d3 = 1$ on a un ellipsoïde imaginaire,
  - $s = 1, d3 = 0$ on a un cône imaginaire,
  - $s = 1, d3 = -1$ on a un ellipsoïde réelle,
  - $s = -1, d3 = 1$ on a un hyperboloïde a 2 nappes, engendré par la rotation d'une hyperbole autour de son axe transverse (c'est l'axe focal, celui qui coupe l'hyperbole),
  - $s = -1, d3 = 0$ on a un cône réel,
  - $s = -1, d3 = -1$ on a un hyperboloïde a 1 nappe, engendré par la rotation d'une hyperbole autour de son axe non transverse (celui qui ne coupe pas l'hyperbole),

  $O1$ est centre de symétrie et les plans (resp les axes) de coordonnées du repère $(01, u, v, w)$ sont des plans (resp des axes) de symétrie pour `Q`.
- 2ième cas $s1 * s2! = 0$ et $s3 = 0$
  l'équation s'écrit :

$s1*(x1+c1/s1)^2+s2*(y1+c2/s2)^2+2*c3*z1+d2=0$

— Si $c3!=0$ l'équation s'écrit :
$s1*(x1+c1/s1)^2+s2*(y1+c2/s2)^2+2*c3*(z1+d2/(2*c3))=0$
en faisant un changement d'origine O1 avec :
$OO1=-c1/s1*u-c2/s2*v-d2/(2*c3)*w$
selon le signe de s1*s2 l'équation s'écrit :
si $s1*s2>0$ l'équation s'écrit :
$X^2/a^2+Y^2/b^2-2*s*Z=0$ avec $s=1$ ou $s=-1$
on a un paraboloïde elliptique
si $s1*s2<0$
l'équation s'écrit : $X^2/a^2-Y^2/b^2-2*s*Z=0$ avec $s=1$ ou $s=-1$
on a un paraboloïde hyperbolique

— Si $c3==0$ l'équation s'écrit :
$s1*(x1+c1/s1)^2+s2*(y1+c2/s2)^2+d2=0$
en faisant un changement d'origine $O1$ avec :
$OO1=-c1/s1*u-c2/s2*v$ on obtient
$s1*X^2+s2*Y^2+d2=0$
La quadrique Q est soit, un cylindre de génératrices parallèles à w, soit, formée de 2 plans parallèles :
si $s1>0, s2>0, d2>0$ on a un cylindre elliptique imaginaire de génératrices parallèles à w,
si $s1>0, s2>0, d2<0$ on a un cylindre elliptique réel de génératrices parallèles à w,
si $s1>0, s2<0, d2<0$ on a un cylindre hyperbolique de génératrices parallèles à w,
si $s1>0, s2>0, d2=0$ on a 2 plans imaginaires conjugués non parallèles qui se coupent selon la droite réelle $X=0, Y=0$ i.e. selon un cylindre réel ayant comme base un point et de génératrices parallèles à w.
si $s1>0, s2<0, d2=0$ on a 2 plans réels non parallèles.

— 3ième cas $s1=!0, s2=0$ et $s3=0$
l'équation s'écrit :
$s1*(x1+c1/s1)^2+2*c2*y1+2*c3*z1+d2=0$
Si $c2!=0$ ou $c3!=0$:
soit $v1$ le vecteur unitaire de la droite du plan $x1=0$ definie par :
$c2*y1+c3*z1=0, v1=[0,c3,-c2]/\sqrt{c2^2+c3^2}$ et soit $w1$ le vecteur unitaire de la droite du plan $x1=0$ definies par :
$c3*y1-c2*z1=0, w1=[0,c2,c3]/\sqrt{c2^2+c3^2}$
$Y*v1+Z*w1=[0,Y*c3+Z*c2,-Y*c2+Z*c3]/\sqrt{c2^2+c3^2}$
$y1=(Y*c3+Z*c2)/\sqrt{c2^2+c3^2}$
$z1=(-Y*c2+Z*c3)/\sqrt{c2^2+c3^2}$ donc
$c2*y1+c3*z1=Z*\sqrt{(c2^2+c3^2)}$
Si on ne norme pas $v1$ et $w1$ on a $v1=[0,c3,-c2]$, $w1=[0,c2,c3]$, $Y*v1+Z*w1=[0,Y*c3+Z*c2,-Y*c2+Z*c3]$, $y1=(Y*c3+Z*c2)$, $z1=(-Y*c2+Z*c3)$ et $c2*y1+c3*z1=Z$ donc le nouveau $c3==1$.
Ainsi, $2*(c2*y1+c3*z1)=2*Z*\sqrt{c2^2+c3^2}$
en faisant un changement d'origine $O1$ avec :

$OO1 = -c1/s1 * u$
Dans le repère $(O1, u, v1, w1)$ l'équation s'écrit :
$s1 * X1^2 + 2 * c4 * Z1 + d2 = 0$ avec $c4 = \sqrt{c2^2 + c3^2}! = 0$
en faisant un changement d'origine $O2$ avec :
$OO2 = -d2/(2 * c4) * w1$
l'équation s'écrit :
$s1 * X^2 + 2/c4 * Z = 0$
`Q` est un cylindre parabolique dont les génératrices sont parallèles à $u1$.
Si $c2 = 0$ et $c3 = 0$ l'équation s'écrit :
$s1 * (x1 + c1/s1)^2 + d2 = 0$
on a alors deux plans parallèles imaginaires conjugués ($s1 * d2 > 0$) deux plans parallèles réels ($s1 * d2 < 0$) ou deux plans confondus ($d2 = 0$).

Pour faciliter l'étude des différents cas le programme `jortho` va renvoyer 6 valeurs :
les 3 valeurs propres (en regroupant les valeurs propres égales et non nulles au début et en mettant les valeurs propres nulles à la fin) et les 3 vecteurs propres seront normés et orthogonaux sauf pour les vecteurs propres associés à 0.
Soit les trois valeurs propres sont : `s1,s2,s3` et les trois vecteurs propres associés sont : `u,v,w`.
On a soit les zeros déjà à la fin et les valeurs égales non nulles déjà regroupées au début, soit on peut avoir 8 cas (`a!=0,b!=0,a!=b`) :
`(0,a,b)` ou `(0,a,a)` ou `(0,a,0)` ou `(b,a,a)` ou `(a,b,a)` ou `(0,0,a)` ou `(a,0,b)` ou `(a,0,a)`.
Pour les quatre premiers cas : Si `(s1==0 ou s2==s3) et s2!=0)` on renvoie `s2,s3,s1` et `v,w,u` .
Ainsi on a maintenant :
`(a,b,0)` ou `(a,a,0)` ou `(a,0,0)` ou `(a,a,b)` ou `(a,b,a)` ou `(0,0,a)` ou `(a,0,b)` ou `(a,0,a)`.
Pour les quatre derniers cas, on a :
`(s2==0 ou s1==s3) et s3!=0)` on renvoie `s3,s1,s2` et `w,u,v`.

Il faut rendre u,v othogonaux lorsque `s1==s2` si `v*u!=0`
`v1:=-(v*u)*u+(u*u)*v` ainsi `v1*u=0` il faut rendre `v,w` othogonaux lorsque `s3==s2==0` si `v*w!=0`
puis on norme `u,v,w`.
Voici les programmes :
`jortho` renvoie les valeurs propres et les vecteurs propres orthonormés.
`quadrique` renvoie la nouvelle origine la matrice de passage et la forme reduite ainsi que la verification.

```
jortho(A):={
local P,J,u,v,w,s1,s2,s3,a,b;
(P,J):=jordan(A);
u:=P[0..2,0];
v:=P[0..2,1];
w:=P[0..2,2];
s1:=J[0,0];
s2:=J[1,1];
s3:=J[2,2];
```

```
if ((s1==0 || s2==s3) && s2!=0){
b:=u;
u:=v;
v:=w;
w:=b;
a:=s1;
s1:=s2;
s2:=s3;
s3:=a;
}
if ((s2==0|| s1==s3)&& s3!=0){
b:=w;
w:=v;
v:=u;
u:=b;
a:=s3;
s3:=s2;
s2:=s1;
s1:=a;
}
//si s1==s2 et si v*u!=0
a:=normal(u*u);
b:=normal(v*u);
if (b!=0) {
v:=-b*u+a*v;
}
//on norme u
u:=u/sqrt(a);
//si s3==s2==0 et si v*w!=0
a:=normal(v*v);
b:=normal(v*w);
if (b!=0) {
w:=-b*v+a*w;
}
//on norme w et v
w:=w/sqrt(normal(w*w));
v:=v/sqrt(a);
return (s1,s2,s3,u,v,w);
};
```

Le programme `Quadrique` renvoie le repere (la nouvelle origine et la matrice de passage) et l'équation réduite dans ce repère :

```
//q0(x,y,z):=4*x^2+y^2+z^2-4*x*y+4*x*z-2*y*z+8*x-4*y+4*z+2
//            (s2=s3=0,c2=c3=0)
//q1(x,y,z):=x^2+3*y^2-3*z^2-8*y*z+2*z*x-4*x*y-1
//            (s3=0,c3=0)
//q2(x,y,z):=5*x^2+y^2+z^2-2*x*y+2*x*z-6*y*z+2*x+4*y-6*z+1
//            (s1,s2,s3!=0)
```

```
//Quadrique(q2)=[(-1)/2,-1,1/2],[[0,2/(sqrt(6)),
//-(1/(sqrt(3)))],[1/(sqrt(2)),-(1/(sqrt(6))),-(1/(sqrt(3)))],
//[1/(sqrt(2)),1/(sqrt(6)),1/(sqrt(3))]],
//-2*X^2+6*Y^2+3*Z^2-3
//q3(x,y,z):=2*x^2+2*y^2+z^2+2*x*z-2*y*z+4*x-2*y-z+3
//            (s3=0,c3!=0)
//q4(x,y,z):=4*x^2+y^2+z^2-4*x*y+4*x*z-2*y*z+8*x-
//            6*y+4*z+2
//            (((s2=s3=0,c2!=0||c3!=0))
//q5(x,y,z):=x^2+2*y^2-z^2-4*x*y+4*x*z+4*y*z+6*x-
//            4*y+2*z+1
//            (root of)
//q6(x,y,z):=(x+y)*(y-z)+3*x-5*y
//              (s3=0,c3!=0)
//q7(x,y,z):=7*x^2+4*y^2+4*z^2+4*x*y-4*x*z-2*y*z-
//              4*x+5*y+4*z-18
//            (s1,s2,s3!=0)
//q8(x,y,z):=-x^2-3*z^2-4*x*y+4*x*z+4*y*z+6*x-4*y+2*z+1
//q9(x,y,z):=x^2-3*x*y+y^2+x-y-z^2/2+sqrt(10)/5*z
//q est une fct de 3 variables de degre 2
//Quadrique renvoie la nouvelle origine,
// la matrice de passage,
//la forme reduite avec le vecteur de variables et
//la verification par ex pour q4
//B:=[[4,-2,2],[-2,1,-1],[2,-1,1]];C:=[4,-3,2];
Quadrique(q):={
local Q,A,B,C,d,J,r,P,O1,c,c1,c2,c3,c4,s1,s2,
      s3,u,v,w,v1,w1,N,k;
Q(x,y,z,t):=normal(t^2*normal(q(x/t,y/t,z/t)));
A:=q2a(Q(x,y,z,t),[x,y,z,t]);
r:=rank(A);
B:=A[0..2,0..2];
C:=A[0..2,3];
d:=A[3,3];
//on rend P orthogonale sauf si il y a des val propres =0
(s1,s2,s3,u,v,w):=jortho(B);
P:=tran([u,v,w]);
//c:=normal(diff(C*P*[x,y,z],[x,y,z]));
c:=normal(C*P);
//les vp nulles sont a la fin
c1:=c[0];
c2:=c[1];
c3:=c[2];
if (s1*s2*s3!=0) {
O1:=normal(-c1/s1*u-c2/s2*v-c3/s3*w);
//O1:=solve(B*[x,y,z]+C,[x,y,z])[0];
d:=q(O1[0],O1[1],O1[2]);
return (O1,P,s1*X^2+s2*Y^2+s3*Z^2+d,[X,Y,Z],r,
```

```
        normal(q(op(O1+P*[X,Y,Z]))));
}
if (s1*s2!=0) {
if (c3!=0) {
O1:=normal(-c1/s1*u-c2/s2*v);
d:=q(O1[0],O1[1],O1[2]);
O1:=normal(-c1/s1*u-c2/s2*v-d/(2*c3)*w);
return (O1,P,s1*X^2+s2*Y^2+2*c3*Z,[X,Y,Z],r,
           normal(q(op(O1+P*[X,Y,Z]))));
}
if (c3==0) {
O1:=normal(P*[-c1/s1,-c2/s2,0]);
d:=q(O1[0],O1[1],O1[2]);
return (O1,P,s1*X^2+s2*Y^2+d,[X,Y,Z],r,
           normal(q(normal(op(O1+P*[X,Y,Z])))));
}
}
if (s1!=0) {
if (c2 !=0 or c3 !=0) {
c4:=sqrt(c2^2+c3^2);
v1:=normal(P*[0,c3,-c2]/c4);
w1:=normal(P*[0,c2,c3]/c4);
O1:=normal(P*[-c1/s1,0,0]);
d:=q(O1[0],O1[1],O1[2]);
P:=P[0..2,0..0];
P:=border(P,v1);
P:=border(P,w1);
//en fait le nouveau c3 ci dessous ==c4 mais
//ca ne marche pas avec c4 mystere (corrige?)
//c:=normal(diff(C*P*[x,y,z],[x,y,z]));
//c3:=c[2];
//O1:=O1+normal(P*[0,0,-d/(2*c3)]);
O1:=O1+normal(P*[0,0,-d/(2*c4)]);
d:=q(O1[0],O1[1],O1[2]);
return (O1,P,s1*X^2+2*c3*Z+d,[X,Y,Z],r,
        normal(q(normal(op(O1+P*[X,Y,Z])))));
//return (O1,P,s1*X^2+2*c4*Z+d,[X,Y,Z],r,
//          normal(q(op(O1+P*[X,Y,Z]))));
}
if (c2==0 && c3==0) {

//on rend P orthogonale
k:=normal(v*v);
w:=normal(-(w*v)*v+k*w);
w:=normal(w/sqrt(normal(w*w)));
v:=normal(v/sqrt(k));
P:=tran([u,v,w]);
O1:=normal(P*[-c1/s1,0,0]);
```

```
d:=q(O1[0],O1[1],O1[2]);

return (O1,P,s1*X^2+d,[X,Y,Z],r,
         normal(q(op(O1+P*[X,Y,Z]))));
}
}
}
```

On tape :

```
q0(x,y,z):=4*x^2+y^2+z^2-4*x*y+4*x*z-2*y*z+8*x-4*y+4*z+2
Quadrique(q0)
```

On obtient :

```
[(-2)/3,1/3,(-1)/3],[[2/(sqrt(6)),(sqrt(5))/5,(sqrt(30))/15],
[-(1/(sqrt(6))),0,(sqrt(30))/6],[1/(sqrt(6)),(-2*sqrt(5))/5,
(sqrt(30))/30]],6*X^2-2,[X,Y,Z],2,6*X^2-2
```

On tape :

```
q1(x,y,z):=x^2+3*y^2-3*z^2-8*y*z+2*z*x-4*x*y-1
Quadrique(q1)
```

On obtient :

```
[0,0,0],[[0,1/(sqrt(6)),5*(-(1/(sqrt(30))))],[1/(sqrt(5)),
2*(-(1/(sqrt(6)))),2*(-(1/(sqrt(30))))],[2/(sqrt(5)),
1/(sqrt(6)),1/(sqrt(30))]],-5*X^2+6*Y^2-1,[X,Y,Z],3,
6*Y^2-5*X^2-1
```

On tape :

```
q2(x,y,z):=5*x^2+y^2+z^2-2*x*y+2*x*z-6*y*z+2*x+4*y-6*z+1
Quadrique(q2)
```

On obtient :

```
[(-1)/2,-1,1/2],[[0,2/(sqrt(6)),-(1/(sqrt(3)))],[1/(sqrt(2)),
-(1/(sqrt(6))),-(1/(sqrt(3)))],
[1/(sqrt(2)),1/(sqrt(6)),1/(sqrt(3))]],
-2*X^2+6*Y^2+3*Z^2-3,[X,Y,Z],4
3*Z^2+6*Y^2-2*X^2-3
```

On tape :

```
q3(x,y,z):=2*x^2+2*y^2+z^2+2*x*z-2*y*z+4*x-2*y-z+3
Quadrique(q3)
```

On obtient :

```
[(-113)/144,41/144,17/72],
[[1/(sqrt(3)),-(1/(sqrt(2))),1/(sqrt(6))],
[-(1/(sqrt(3))),-(1/(sqrt(2))),
-(1/(sqrt(6)))],[1/(sqrt(3)),
0,2*(-(1/(sqrt(6))))]],
3*X^2+2*Y^2+(2*2*sqrt(6)*Z)/3,
[X,Y,Z],4,3*X^2+(4*sqrt(6)*Z)/3+2*Y^2
```

On tape :

```
q4(x,y,z):=4*x^2+y^2+z^2-4*x*y+4*x*z-2*y*z+8*x-6*y+4*z+2
Quadrique(q4)
```

On obtient :

```
[(-227)/180,(-71)/72,(-227)/360],
[[2/(sqrt(6)),(-(sqrt(5)))/5,(-(sqrt(30)))/15],
```

```
[-(1/(sqrt(6))),0,(-(sqrt(30)))/6],
[1/(sqrt(6)),(2*sqrt(5))/5,(-(sqrt(30)))/30]],
6*X^2+(2*sqrt(30)*Z)/6,[X,Y,Z],3,
(48*sqrt(30)*Z)/144+6*X^2
```

On tape :

```
q5(x,y,z):=x^2+2*y^2-z^2-4*x*y+4*x*z+4*y*z+6*x-4*y+2*z+1
Quadrique(q5)
```

On obtient :

```
[(-15)/13,5/13,(-7)/13],[[(-2)/3,(-sqrt(13)+1)/(sqrt(-8*sqrt(13)+52)),
(sqrt(13)+1)/(sqrt(8*sqrt(13)+52))],[(-1)/3,4/(sqrt(-8*sqrt(13)+52)),
4/(sqrt(8*sqrt(13)+52))],[(-2)/3,(sqrt(13)-3)/(sqrt(-8*sqrt(13)+52)),
(-sqrt(13)-3)/(sqrt(8*sqrt(13)+52))]],
2*X^2+sqrt(13)*Y^2+(-(sqrt(13)))*Z^2+(-49)/13,[X,Y,Z],4,
((-2*sqrt(13))*Z^2)/2+(2*sqrt(13)*Y^2)/2+2*X^2+(-49)/13
```

On tape :

```
q6(x,y,z):=(x+y)*(y-z)+3*x-5*y
Quadrique(q6)
```

On obtient :

```
[16/9,8/9,11/9],[[1/(sqrt(2)),1/(sqrt(6)),-(1/(sqrt(3)))],
[0,2/(sqrt(6)),1/(sqrt(3))],[1/(sqrt(2)),-(1/(sqrt(6))),
1/(sqrt(3))]],(X^2)/-2+(3*Y^2)/2+(2*(-4*sqrt(3))*Z)/3,
[X,Y,Z],4,-(X^2)/2-(8*sqrt(3)*Z)/3-(-3*Y^2)/2
```

On tape :

```
q7(x,y,z):=7*x^2+4*y^2+4*z^2+4*x*y-4*x*z-2*y*z-4*x+5*y+4*z-18
Quadrique(q7)
```

On obtient :

```
[11/27,(-26)/27,(-29)/54],[[1/(sqrt(5)),54*(-(1/(sqrt(30)*27))),
2/(sqrt(6))],[0,5/(sqrt(30)),1/(sqrt(6))],[2/(sqrt(5)),1/(sqrt(30)),
-(1/(sqrt(6)))]],3*X^2+3*Y^2+9*Z^2+(-602)/27,
[X,Y,Z],4,3*Y^2+3*X^2+9*Z^2+(-602)/27
```

On tape :

```
q8(x,y,z):=-x^2-3*z^2-4*x*y+4*x*z+4*y*z+6*x-4*y+2*z+1
Quadrique(q8)
```

On obtient (c'est tres long !) :

```
[(-11)/54,223/108,61/54],[[(-sqrt(13)+1)/(sqrt(-8*sqrt(13)+52)),
(sqrt(13)+1)/(sqrt(8*sqrt(13)+52)),(-2)/3],[4/(sqrt(-8*sqrt(13)+52)),
4/(sqrt(8*sqrt(13)+52)),(-1)/3],[(sqrt(13)-3)/(sqrt(-8*sqrt(13)+52)),
(-sqrt(13)-3)/(sqrt(8*sqrt(13)+52)),(-2)/3]],
(sqrt(13)-2)*X^2+(-sqrt(13)-2)*Y^2-4*Z,[X,Y,Z],4,
((-4+2*sqrt(13))*X^2)/2+((-4-2*sqrt(13))*Y^2)/2-4*Z
```

On tape :

```
q9(x,y,z):=x^2-3*x*y+y^2+x-y-z^2/2+sqrt(10)/5*z
Quadrique(q9)
```

On obtient un cône car `r=3` :

```
[-1/5,1/5,(sqrt(10))/5],[[1/(sqrt(2)),1/(sqrt(2)),0],[-(1/(sqrt(2))),
1/(sqrt(2)),0],[0,0,1]],(5*X^2)/2+(Y^2)/(-2)+(Z^2)/(-2),[X,Y,Z],3,
5/2*X^2+(-1)/2*Y^2+(-1)/2*Z^2
```

On tape :

```
q10(x,y,z):=x^2-3*x*y+y^2-x-y-z^2/2+sqrt(10)/5*z
Quadrique(q10)
```
On obtient :
```
[-1,-1,(sqrt(10))/5],[[1/(sqrt(2)),1/(sqrt(2)),0],
[-(1/(sqrt(2))),1/(sqrt(2)),0],[0,0,1]],
(5*X^2)/2+(Y^2)/(-2)+(Z^2)/(-2)+6/5,[X,Y,Z],4,
5/2*X^2+(-1)/2*Y^2+(-1)/2*Z^2+6/5
```

17.3 Fonction de Sudan

En calculabilité, la fonction de Sudan est un exemple de fonction récursive mais non récursive primitive. C'est aussi le cas de la fonction d'Ackermann, plus connue.
Elle fut conçue en 1927 par le mathématicien roumain Gabriel Sudan, élève de David Hilbert.
Définition $F_0(x,y) = x + y$
$F_{n+1}(x,0) = x$ si $n \geq 0$
$F_{n+1}(x,y+1) = F_n(F_{n+1}(x,y), F_{n+1}(x,y) + y + 1)$ si $n \geq 0$
Avec `Xcas` on tape :

```
fsudan(n,p,q):={
  local a;
  si n==0 alors retourne simplify(p+q); fsi;
  si q==0 alors retourne p; fsi;
  a:=simplify(fsudan(n,p,q-1));
  retourne fsudan(n-1,a,a+q);
  }:;
```

On tape :
`fsudan(1,1,1)`
On obtient :
`3`
On tape :
`fsudan(1,3,3)`
On obtient :
`35`
On tape :
`fsudan(2,1,1)`
On obtient :
`8`
On tape :
`fsudan(2,2,2)`
On obtient :
`15569256417`
Bien voir que :
`fsudan(1,x,4)` renvoie `26+16*x` MAIS si je tape : `fsudan(2,x,1)` j'obtiens :

"Trop de niveaux de récursions Erreur : Valeur Argument Incorrecte"
en effet :
`fsudan(2,x,1)=fsudan(1,fsudan(2,x,0),fsudan(2,x,0)+1)=`
`fsudan(1,x,x+1)=fsudan(0,f(1,x,x),fsudan(1,x,x)+x+1)=2*fsudan(1,x,x)+x+1)`
et là le programme ne s'arrête pas puisque `x+1` n'est jamais nul.... Les valeurs de
`fsudan` augmentent très vite lorsque `y` augmente On tape :
`fsudan(2,1,3` On obtient :

```
70430581144274911293924132443665781679828953193120507606992938292310
32205707501679774305057283702760018257504486209030580985157737087852
58459878329347613373850614162671735803936177702653848366454984893981
57864724880917330512522358149695410980606136793910891712447130067494
43526620975760942935041533626718036993326887758474625214966139241641
66308641419171537689517638142546615351803304614846616706309934486939
09061871928176262153144988786161552551902033567924464412779006525324
81648076737790439398901503951010830175596816079671926448063917302969
64795908712145792424206130480813372398724831155767229964348090616442
76816294018500692663722771598512047416079928567873531272680546972882
60182549740971546148787588513273168471111740355466749809126344606734 5
74343805105852080535515406441877254328114097340282397486448183862387
31800609136457774376697772488022441224607047419045651816575399789265
99809734382801607685680855783679491033753441023753393567048932129357
68599729545389947046472569820401695378486299507895151881435591016120
20310816281472267832807829442893850384589099675898858687214289267044
21886605233171546279548773723367958249206185260885238126686277104855
13682840079783982409119366713084561489904438571535964183661683476132
71402500150599635546989064262071657410401942732044814439171595515116
54490381144778528193464750398898837705190486581756732150795449700211
80703777373429836593251964607657859713221615296352656664271588786359
51087907886714185396054249893875773443677281234164064016284120013075
32777671117555915379088944097932417504756891718015281700293999857066
19507205580930593219498476858980777490863597219500477827085170146768
04639005796989582498513400470793408842579395090373497242403463981068
42984463733084025698268158645979914054900030082849031249744681008133
02368131162629238738532330589669161607107464335326949181835955797630
19217720062013488188129056154232088915171179110165349040820783856654
60578835708508174562225983623694238373559655135528100033782466716475
10513688767406953734867218559677238596322296476555341808085474491390
27872159028021170415926535524774239426095842301684773904408866420164
10648313044756837782460430257073827721571365311985884420591916902909
57543674116461578707663403685910976670737792091882268830226679468321
89375110807365573321368062533276092652548025947520753583335535872065
39586696954394195524090370565318868165877600492116370931306048484703
06912918852587545668137372593156659423032213812838597997912121715574
10835734856309425448274272933846121708026874116853055558401318974282
92624953476443186032426370233001292834951827056049287080240866716093
71480216699773012002381184469101992962477562430264745347117951998279
67046629241292368040054261029928729469512460506530381041179568755853
08900893728891785655112379025122738236626597618170134596030253420745
```

```
79923221677363131209685362424379632738056778165022742100298446483974
88268745407132460619535722775316354647098182369054091375207675184756
63172999983597118586847701969854541979904796401072035939019175492022
68632765205942777599112121658779210773283830545460696791365672087378
73487190034966751738880 7
```

En effet :

```
fsudan(2,1,3) =fsudan(1,fsudan(2,1,2),fsudan(2,1,2)+3)=
fsudan(1,10228,10231)=
fsudan(0,fsudan(1,10228,10230),fsudan(1,10228,10230)+10231)=
2*fsudan(1,10228,10230)+10231).....
```

17.4 Fonction d'Ackermann

Dans les années 1920, Wilhelm Ackermann et Gabriel Sudan, alors étudiants sous la direction de David Hilbert, étudiaient les fondements de la calculabilité. Sudan est le premier à donner un exemple de fonction récursive mais non récursive primitive, appelée alors fonction de Sudan. Peu après et indépendamment, en 1928, Ackermann a publié son propre exemple de fonction récursive mais non récursive primitive1.

La fonction d'Ackermann-Péter est définie récursivement comme suit : $A(0,n) = n+1$

$A(m,0) = A(m-1,1)$ si $m > 0$

$A(m,n) = A(m-1, A(m,n-1))$ si $m > 0$ et $n > 0$

Avec `Xcas` on tape :

```
a(m,n):={
  si m==0 alors retourne n+1; fsi;
  si m>0 and n==0 alors retourne a(m-1,1); fsi;
  retourne a(m-1,a(m,n-1));
  }:;
```

On tape :

```
a(2,5)
```

On obtient :

```
13
```

On tape :

```
a(3,5)
```

On obtient :

```
253
```

Chapitre 18

Quelques compléments

18.1 Pour réutiliser le graphe d'une fonction utilisateur

Lorsque Xcas fait le graphe G d'une fonction utilisateur g, dans la réponse, il va figurer l'expression formelle de g(x) et si cette fonction contient des tests, lorsque on va reutiliser G il y aura un message d'erreur car il ne sait pas évaluer les tests contenus dans g(x). Pour ne pas avoir ce genre d'erreurs, il faut alors commencer l'écriture de la fonction g par un test sur le type de ses arguments qui est : si les arguments ne sont pas réels on renvoie la fonction quotée.
Exemple

```
g(x):={
si type(x)!=DOM_FLOAT alors
  retourne 'g'(x);
fsi;
si x<=-1 alors
  retourne -1;
fsi;
si -1<x  et x<1 alors
  retourne sin(pi*x/2);
fsi;
si 1<= x alors
retourne 1;
fsi;
}:;
f(x,y):={
si type(x)!=DOM_FLOAT et type(y)!=DOM_FLOAT alors
  retourne 'f'(x,y)
fsi;
si x^2+y^2<=2 alors
  retourne 2
fsi;
si x^2+y^2>2 alors
  retourne x^2+y^2
fsi;
}
```

```
:;
```

On tape par exempple :
`G:=plotfunc(g(x))` ou
`F:=plotfunc(f(x,y),[x=-4..4,y=-4..4]);` et pour voir le cercle $x^2+y^2=2$:

```
plotparam([x,sqrt(2-x^2),2],[x,y],affichage=1);
plotparam([x,-sqrt(2-x^2),2],[x,y],affichage=1)
```

18.2 Les programmes de quadrillage

Voici les programmes qui permettent de paire du papier quadrillé (`papierq`), du papier triangulé (`papiert`), du papier pointé (`papierp`).

`u` est le pas en `x`,
`v` est le pas en `y`,
les lignes sont parallèles à `y=x*tan(t)` avec `0<t<pi` et à `y=cste` dans le rectangle `[x1,x2]*[y1,y2]`

```
papierq(u,v,t,x1,x2,y1,y2):={
  local x3,y3,y4,L,k,q,j,j0;
  si normal(t-pi)>=0 ou normal(t)<=0  alors return "0<=t<=pi";fsi;
  si x2<x1 alors j:=x1;x1:=x2;x2:=j;fsi;
  si y2<y1 alors j:=y1;y1:=y2;y2:=j;fsi;
  L:=polygone(x1+i*y1,x1+i*y2,x2+i*y2,x2+i*y1);
  pour k de y1 jusque y2 pas v faire
    L:=L,segment(x1+i*k,x2+i*k);
  fpour;
  si normal(t-pi/2)==0 alors
    pour j de x1 jusque x2 pas u faire
      L:=L,segment(j+i*y1,j+i*y2);
    fpour;
    return L;
  fsi;
  k:=evalf((y2-y1)/tan(t));
  si normal(t-pi/2)<0 alors
    q:=floor(k/u+1e-12);
    pour j de x1-q*u jusque x1 pas u faire
      y3:=tan(t)*(x2-j)+y1;
      y4:=tan(t)*(x1-j)+y1;
      x3:=(y2-y1)/tan(t)+j;
      si y3<y2 alors L:=L,segment (x1+i*y4,x2+i*y3); fsi;
      si x3<x2 alors L:=L,segment (x1+i*y4,x3+i*y2); fsi;
    fpour;
    pour j de x1 jusque x2 pas u faire
      y3:=tan(t)*(x2-j)+y1;
      x3:=(y2-y1)/tan(t)+j;
      si y3<y2 alors L:=L,segment (j+i*y1,x2+i*y3); fsi;
      si x3<x2 alors L:=L,segment (j+i*y1,x3+i*y2); fsi;
```

```
    fpour;
    return L;
  fsi;
  si normal(t-pi/2)>0 alors
    j:=x1; tantque j<=x2 faire
      y3:=tan(t)*(x1-j)+y1;
      x3:=(y2-y1)/tan(t)+j;
      si y3<y2 alors L:=L,segment (j+i*y1,x1+i*y3); fsi;
      si x3>x1 alors L:=L,segment (j+i*y1,x3+i*y2); fsi;
      j:=j+u;
    ftantque;
    q:=ceil(k/u-1e-12);
    tantque j<=x2-q*u faire
      y3:=tan(t)*(x1-j)+y1;
      y4:=tan(t)*(x2-j)+y1;
      x3:=(y2-y1)/tan(t)+j;
      si normal(y3-y2)<0 alors L:=L,segment (x2+i*y4,x1+i*y3); fsi;
      si normal(x3-x1)>0 alors L:=L,segment (x2+i*y4,x3+i*y2); fsi;
      j:=j+u;
    ftantque;
    return L;
  fsi;
}
:;
papiert(u,v,t,x1,x2,y1,y2):={
  local x3,y3,y4,L,k,q,j,u1,t1,L;
  //si normal(t-pi/2)==0 alors retourne -atan(v/u)+pi; fsi;
  u1:=v/tan(t);
  si normal(u1-u)==0 alors t1:=pi/2; fsi;
  si normal(u1-u)>0 alors t1:=atan(v/(u1-u)); fsi;
  si normal(u1-u)<0 alors t1:=atan(v/(u1-u))+pi; fsi;
  L:=papierq(u,v,t,x1,x2,y1,y2);
  L:=L,papierq(u,v,t1,x1,x2,y1,y2);
  retourne L;
}:;
papierp(u,v,t,x1,x2,y1,y2):={
  local j,k,L,q,q2,x3;
  si normal(t-pi)>=0 ou normal(t)<=0  alors return "0<t<pi";fsi;
  L:=NULL;
  L:=L,polygone(x1+i*y1,x2+i*y1,x2+i*y2,x1+i*y2);
  si x2<x1 alors j:=x1;x1:=x2;x2:=j;fsi;
  si y2<y1 alors j:=y1;y1:=y2;y2:=j;fsi;
  pour j de y1 jusque y2 pas v faire
    x3:=evalf((j-y1)/tan(t));
    q:=floor(x3/u+1e-12);
    pour k de x1-q*u+x3 jusque x2 pas u faire
      L:=L,point(k+i*j);
    fpour;
```

```
  fpour;
  retourne L;
}:;
```

Chapitre 19

Les programmes récursifs

Certains programmes se trouvent dans `examples/recur`.

19.1 Avec des chaines de caractères

19.1.1 Une liste de mots

On veut énumérer les éléments d'une liste. Pour cela on écrit le premier élément et on énumère la liste privée de son premier élément. On s'arrête quand la liste est vide.
On écrit :

```
enumere(l):={
if (l==[]) return 0;
print(l[0]);
enumere(tail(l));
}
```

On tape :
`enumere(["jean","paul","pierre"])`
On obtient, en écriture bleue, dans la zone des résultats intermédiaires :
`"jean"`
`"paul"`
`"pierre"`

19.1.2 Les mots

Étant donné un mot de n lettres, on veut écrire n lignes :
la première ligne sera constituée par la première lettre du mot,
la deuxième ligne sera constituée par les deux premières lettres...,
la dernière ligne sera constituée par le mot tout entier.
On écrira : `mots(m)` de façon récursive. On peut se servir de la fonction `size(m)` de `Xcas` qui renvoie le nombre de lettres du mot `m`, et de la fonction `suppress(m,k)` de `Xcas` qui renvoie le mot `m` privé de sa k-ième lettre (k=`0.. size(m)-1`).
On tape :

```
saufdernier(m):={
```

```
return suppress(m,size(m)-1);
}
```

puis

```
mots(m):={
if (size(m)==0) return 0;
mots(saufdernier(m));
print(m);
}
```

Exercice
Comment modifier le programme précédent pour avoir :
Étant donné un mot de n lettres, on veut écrire n lignes :
la première ligne sera constituée par le mot tout entier,
la deuxième ligne sera constituée par le mot privé de sa première lettre...,
la dernière ligne sera constituée par la première lettre du mot.
Réponse
On peut se servir de la fonction `tail(m)` de `Xcas` qui renvoie l le mot `m` privé de sa première lettre.

```
motex(m):={
if (size(m)==0) return 0;
print(m);
motex(tail(m));
}
```

19.2 Les palindromes

19.2.1 Les phrases palindromes

Étant donné une phrase, on veut écrire cette phrase en l'écrivant de droite à gauche. On écrira :
`palindrome(s)` de façon récursive :
il faut rajouter la première lettre de la phrase à la fin du palindrome de la phrase privée de sa première lettre.
On tape :

```
palindrome(ph):={
local s;
if (s==0) return ph;
s:=size(ph)-1;
return concat(palindrome(tail(ph)),ph[0]);
}
```

ou encore :
il faut rajouter la dernière lettre de la phrase devant le palindrome de la phrase privée de sa dernière lettre.
On tape :

```
saufdernier(m):={
return suppress(m,size(m)-1);
}

palindrome(ph):={
local s;
if (s==0) return ph;
s:=size(ph)-1;
return concat(ph[s],palindrome(saufdernier(ph)));
}
```

19.2.2 Nombre et valeur palindromique d'un entier

Un entier n est un palindrome s'il est identique à son palindrome qui est le nombre obtenu en écrivant n de droite à gauche.
Par exemple 12321 est un palindrome.
Pour tout entier n , on considère l'algorithme suivant :

1. si l'entier n est un palindrome on s'arrête,
2. sinon on ajoute à l'entier son palindrome (l'entier écrit à l'envers) et on retourne à l'étape 1.

Par exemple : 687,687+786=1473,1473+3741=5214,5214+4125=9339.
On appelle nombre palindromique d'un entier n le nombre Npal(n) d'étapes nécessaire pour obtenir unpalindrome.
Par exemple : Npal(12321)=0, Npal(687)=3.
On appelle valeur palindromique d'un entier n la valeur Vpal(n) du palindrome final.
Par exemple : Vpal(12321)=12321, Vpal(687)=9339.

a) Étant donné un entier n, écrire une fonction `palind` qui renvoie le palindrome de n.

b) Étant donné un entier n, écrire une fonction `vnpalind` qui renvoie la liste constituée de n, de sa valeur palindromique et de son nombre palindromique.
Attention
L'arrêt du processus est une conjecture. Prévoir un contrôle stoppant la boucle au delà de p étapes et renvoyant alors la liste .

c) Explorer pour $p = 300$ le cas des 100 premiers entiers (non nuls).

d) Quels sont les entiers qui n'ont pas obtenu de nombre palindrome parmi les 1000 premiers entiers ?

a) Soit m le palindrome de n.
Le 1ier chiffre de m est le reste de la division de n par 10.
Le 2nd chiffre de m est le reste de `iquo(n,10)` par 10.
On utilise ici la commande `iquorem`.
On tape :

```
palind(n):={
  local r,m;
```

```
  m:=0;
  tantque n>=10 faire
    n,r:=iquorem(n,10);
    m:=10*m+r;
  ftantque;
  m:=10*m+n;
  retourne m;
  }:;
```

On tape :

```
palind(89)
```

On obtient :

```
98
```

On tape :

```
palind(123456789)
```

On obtient :

```
987654321
```

b) On tape :

```
vnpalind(n,p):={
  local j,m,n0;
  m:=palind(n);
  j:=0;
  n0:=n;
  tantque n!=m and j<p faire
    n:=n+m;
    j:=j+1;
    m:=palind(n);
  ftantque;
  si j==p and m!=palind(n+m) alors return [n0]; fsi;
  return [n0,m,j];
  }:;
```

On tape :

```
vnpalind(86,50)
```

On obtient :

```
[86,1111,3]
```

On tape :

```
vnpalind(89,50)
```

On obtient :

```
[89,8813200023188,24]
```

c) On tape :

```
lvnpalind():={
local L,n;
L:=NULL;
pour n de 1 jusque 100 faire
L:=L,vnpalind(n,300);
fpour;
retourne L;
}:;
```

On tape :

```
lvnpalind()
```

On obtient :

```
[1,1,0],[2,2,0],[3,3,0],[4,4,0],[5,5,0],[6,6,0],
[7,7,0],[8,8,0],[9,9,0],[10,11,1],[11,11,0],
[12,33,1],[13,44,1],[14,55,1],[15,66,1],[16,77,1],
[17,88,1],[18,99,1],[19,121,2],[20,22,1],[21,33,1],
[22,22,0],[23,55,1],[24,66,1],[25,77,1],[26,88,1],
[27,99,1],[28,121,2],[29,121,1],[30,33,1],[31,44,1],
[32,55,1],[33,33,0],[34,77,1],[35,88,1],[36,99,1],
[37,121,2],[38,121,1],[39,363,2],[40,44,1],[41,55,1],
[42,66,1],[43,77,1],[44,44,0],[45,99,1],[46,121,2],
[47,121,1],[48,363,2],[49,484,2],[50,55,1],[51,66,1],
[52,77,1],[53,88,1],[54,99,1],[55,55,0],[56,121,1],
[57,363,2],[58,484,2],[59,1111,3],[60,66,1],[61,77,1],
[62,88,1],[63,99,1],[64,121,2],[65,121,1],[66,66,0],
[67,484,2],[68,1111,3],[69,4884,4],[70,77,1],[71,88,1],
[72,99,1],[73,121,2],[74,121,1],[75,363,2],[76,484,2],
[77,77,0],[78,4884,4],[79,44044,6],[80,88,1],[81,99,1],
[82,121,2],[83,121,1],[84,363,2],[85,484,2],[86,1111,3],
[87,4884,4],[88,88,0],[89,8813200023188,24],[90,99,1],
[91,121,2],[92,121,1],[93,363,2],[94,484,2],[95,1111,3],
[96,4884,4],[97,44044,6],[98,8813200023188,24],[99,99,0],
[100,101,1]
```

d) On tape :

```
pbvnpalind():={
  local L,n;
  L:=NULL;
  pour n de 1 jusque 1000 faire
    si vnpalind(n,300)==[n] alors L:=L,n fsi;
  fpour;
  retourne L;
}:;
```

On tape :

```
pbvnpalind()
```

On obtient :

```
196,295,394,493,592,689,691,788,790,879,887,978,986
```

On peut aussi écrire le programme qui donne la suite des transformés de n par l'algorithme en au plus p étapes.

On tape :

```
tracevnpalind(n,p):={
  local L,k,m;
  L:=n;
  m:=palind(n);
  k:=0;
  tantque n!=m and k<p faire
    n:=n+m;
    k:=k+1;
    L:=L,n
```

```
    m:=palind(n);
  ftantque;
  return L;
  }:;
```

On tape :
`tracevnpalind(196,25)`
On obtient :

```
196,887,1675,7436,13783,52514,94039,187088,1067869,
10755470,18211171,35322452,60744805,111589511,
227574622,454050344,897100798,1794102596,
8746117567,16403234045,70446464506,130992928913,
450822227944,900544455998,1800098901007,8801197801088
```

On tape :
`tracevnpalind(986,25)`
On obtient :

```
986,1675,7436,13783,52514,94039,187088,1067869,
10755470,18211171,35322452,60744805,111589511,
227574622,454050344,897100798,1794102596,8746117567,
16403234045,70446464506,130992928913,450822227944,
900544455998,1800098901007,8801197801088,17602285712176
```

On tape :
`tracevnpalind(1585,25)`
On obtient :

```
1585,7436,13783,52514,94039,187088,1067869,10755470,
18211171,35322452,60744805,111589511,227574622,
454050344,897100798,1794102596,8746117567,
16403234045,70446464506,130992928913,450822227944,
900544455998,1800098901007,8801197801088,
17602285712176,84724043932847
```

On tape :
`tracevnpalind(1997,25)`
On obtient :

```
1997,9988,18887,97768,184547,930028,1750067,9350638,
17711177,94822948,179745797,977293768,1844686547,
9301551028,17503102067,93523232638,177146465177,
948711106948,1798312224797,9772534363768,18446168716547,
93007954881028,175026800851067,935184809471638,
1771359717953177,9484956897484948
```

19.3 Les dessins récursifs

19.3.1 Les segments

1. Une spirale
   L'utilisateur choisit un réel $a \in]0;1[$. Soit un point M_0 d'affixe un réel $z_0 > 0$ et les points M_k d'affixe $z_k = r_k \exp(ik\pi/3)$ avec $r_k = a * r_{k-1}$.
   Écrire un programme qui réalise le dessin du point M_0 et des p segments $M_k M_{k-1}$ pour $k = 1..p$.

On a donc $z_k == a*r_{k-1}\exp(i(k-1)\pi/3)\exp(i\pi/3) == a\exp(i\pi/3)z_{k-1}$
On peut faire soit un programme itératif, soit un programme récursif.
On tape pour le programme itératif :

```
segmenti(a,z0,p):={
local L,k;
point(z0);
L:=NULL;
pour k de 1 jusque p faire
L:=L,segment(z0,a*z0*exp(i*pi/3));
z0:=a*z0*exp(i*pi/3);
fpour;
retourne L;
}
:;
```

Pour le programme récursif : On peut décider d'avoir le dessin récursif seulement dans l'écran `DispG` : le programme est plus simple car toutes les instructions graphiques sont exécutées dans cet écran. On renvoie `1` pour que l'on puisse vérifier que la procédure s'est bien exécutée.
On tape :

```
segmentg(a,z0,p):={
  point(z0);
  si p>0 alors
    segment(z0,a*z0*exp(i*pi/3));
    segmentg(a,a*z0*exp(i*pi/3),p-1);
  fsi;
  retourne 1;
}
:;
```

Ou bien on met les différentes instructions graphiques à réaliser dans une liste `L`.
On tape :

```
segmentr(a,z0,p):={
  local L;
  si p==0 alors retourne point(z0); fsi;
  L:=segment(z0,a*z0*exp(i*pi/3)),segmentr(a,a*z0*exp(i*pi/3),p-1);
  retourne L;
}
:;
```

Puis on tape :
`segmenti(0.8, 20, 30)` ou
`segmentg(0.8, 20, 30)` ou
`segmentr(0.8, 20, 30)`

2. Une autre spirale
À partir d'un point M_0 de coordonnées $(x_0, 0)$ et d'un réel $0 < k < 1$, on définit les points : M_1 de coordonnées $(0, kx_0)$

M_2 de coordonnées $(-k^2x_0, 0)$
M_3 de coordonnées $(0, -k^3x_0)$
M_4 de coordonnées $(k^4x_0, 0)$
etc
Écrire un programme qui dessine n segments M_kM_{k+1}

On peut remarquer que l'affixe de M_{k+1} se déduit de celui de M_k par une multiplication par $i * k$.
On tape un programme itératif :

```
spirali(x0,k,n):={
local k,L;
pour k de 1 jusque n faire
 L:=L,segment(x0,i*k*x0);
 x0:=i*k*x0;
}
retourne L;
}:;
```

On tape un programme récursif :

```
spiralr(x0,k,n):={
 si n<=0 alors retourne point(x0); fsi;
 retourne segment(x0,i*k*x0),spiralr(i*k*x0,k,n-1);
}:;
```

3. Une maison Soient $A = (0, 0)$ et $B = (1, 0)$. À partir du vecteur AB, on construit une maison c'est à dire les murs sont un carré direct $ABEC$ et le toit est un triangle isocèle CED.
On trace les segments AC et BE et on recommence la même opération avec les vecteurs CD et DE etc...On s'arrête en tracant le segment correspondant à AB
Écrire un programme qui réalise ce dessin au bout de n fois.

On tape un programme récursif (pour éviter des calculs trop long, on travaille en mode approché en mettant `A:=evalf(A); B:=evalf(B);`):

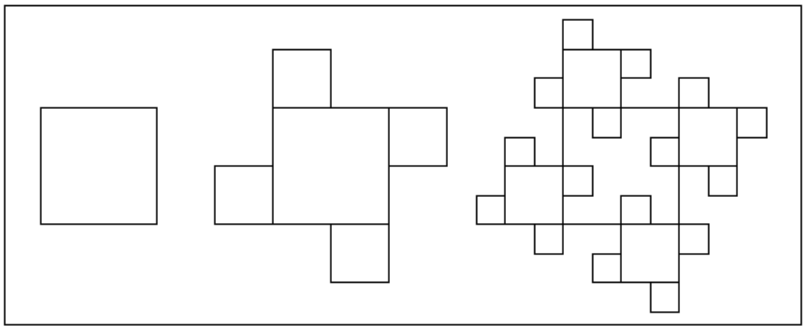
```
maison(A,B,n):={
 local C,D,E,L;
 A:=evalf(A); B:=evalf(B);
 si n<=0 alors retourne segment(A,B); fsi;
 C:=rotation(A,pi/2,B);
 E:=rotation(B,-pi/2,A);
 D:=similitude(C,sqrt(2)/2,pi/4,E);
 L:=segment(A,C),segment(B,E);
 L:=L,maison(C,D,n-1),maison(D,E,n-1);
 retourne L;
}:;
```

On peut transformer cet exercice en un exercice sur les complexes en demandant de calculer les affixes de C, D, E en fonction des affixes de A et B. On tape un programme récursif :

```
maisonee(A,B,n):={
 local C,D,E,L,za,zb,zc,zd,ze;
 si n<=0 alors retourne segment(A,B); fsi;
 za:=affixe(A);
 zb:=affixe(B);
 zc:=normal(za+i*(zb-za));
 ze:=normal(zb+i*(zb-za));
 zd:=normal(3*i*(zb-za)/2+(za+zb)/2);
 C:=point(zc);
 E:=point(ze);
 D:=point(zd);
 L:=segment(A,C),segment(B,E);
 L:=L,maisonee(C,D,n-1),maisonee(D,E,n-1);
 retourne L;
}:;
```

19.3.2 Les carrés

On veut réaliser le dessin récursif dont on a mis ci-dessous les premières étapes (profondeur 0,1 et 2) :



On peut décider d'avoir le dessin récursif seulement dans l'écran `DispG` : le programme est plus simple car toutes les instructions graphiques sont exécutées dans cet écran.
On appelle cette procédure `carresg(a,d,f)` où `a` est l'affixe du sommet en bas à gauche du grand carré, `d` est la longueur de son côté et `f` donne la longueur du côté du plus petit carré. `carresg(a,d,f)` renvoie `1` pour que l'on puisse vérifier que la procédure s'est bien exécutée.
On tape :

```
carresg(a,d,f):={
  si d>=f alors
    carre(a,a+d);
    carresg(a-d/2,d/2,f);
    carresg(a+i*d,d/2,f);
    carresg(a+d/2-i*d/2,d/2,f);
    carresg(a+d+i*d/2,d/2,f);
  fsi;
  retourne 1;
```

```
}:;
```

On tape : `carresg(0,40,2)`
L'écran `DispG` s'ouvre et l'on voit le dessin se faire....

On met les différentes instructions graphiques à réaliser dans une liste `L`. On appelle cette procédure `carres(a,d,f)` où `a` est l'affixe du sommet en bas à gauche du grand carré, `d` est la longueur de son côté et `f` donne la longueur du côté du plus petit carré. `carresg(a,d,f)` renvoie la liste `L`.
On tape :

```
carres(a,d,f):={
  local L;
  si d<f alors retourne NULL fsi;
  L:=carre(a,a+d),carres(a-d/2,d/2,f),carres(a+i*d,d/2,f),
   carres(a+d/2-i*d/2,d/2,f),carres(a+d+i*d/2,d/2,f);
  retourne L;
}
:;
```

On tape : `carres(0,40,2)`

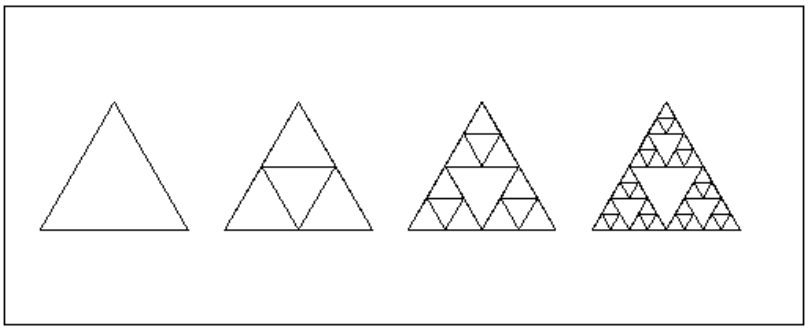
On peut aussi choisir comme paramètre la profondeur `n` du dessin récursif au lieu de `f`. On tape :

```
carren(a,d,n):={
local L;
si n=<0 alors retourne NULL fsi;
L:=carre(a,a+d),carren(a-d/2,d/2,n-1),carren(a+i*d,d/2,n-1),
   carren(a+d/2-i*d/2,d/2,n-1),carren(a+d+i*d/2,d/2,n-1);
retourne L;
}
:;
```

On tape : `carren(0,40,4)`

19.3.3 Les triangles

On veut réaliser le dessin récursif dont on a mis ci-dessous les premières étapes (profondeur 0,1,2 et 3) :



On peut décider d'avoir le dessin récursif seulement dans l'écran `DispG` : le programme est plus simple car toutes les instructions graphiques sont exécutées

dans cet écran.
On appelle cette procédure `triang(a,d,f)` où `a` est l'affixe du sommet en bas à gauche du grand triangle, `d` est la longueur de son côté et `f` donne la longueur du côté du plus petit triangle. `triang(a,d,f)` renvoie `1` pour que l'on puisse vérifier que la procédure s'est bien exécutée.
On tape :

```
triang(a,d,f):={
  si d>=f alors
    triangle_equilateral(a,a+d);
    triang(a,d/2,f);
    triang(a+d/4+i*d*sqrt(3.)/4,d/2,f);
    triang(a+d/2,d/2,f);
  fsi;
  retourne 1;
}:;
```

On tape : `triang(0,40,2)`
L'écran `DispG` s'ouvre et l'on voit le dessin se faire....On met les différentes instructions graphiques à réaliser dans une liste. On appelle cette procédure `triangles(a,d,f)` où `a` est l'affixe du sommet en bas à gauche du grand triangle, `d` est la longueur de son côté et `f` donne la longueur du côté du plus petit triangle.
On tape :

```
triangles(a,d,f):={
  local L;
  si d<f alors retourne NULL fsi;
  L:=triangle_equilateral(a,a+d),triangles(a,d/2,f),
     triangles(a+d/4+i*d*sqrt(3.)/4,d/2,f),triangles(a+d/2,d/2,f);
  retourne L;
}
:;
```

On tape : `triangles(0,40,2)`

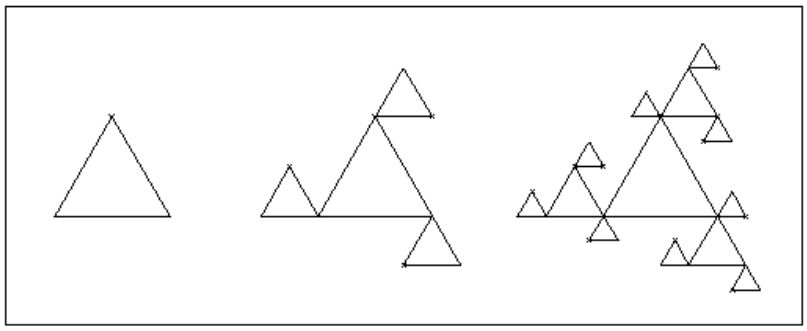
On peut aussi choisir comme paramètre la profondeur `n` du dessin récursif au lieu de `f`. On tape :

```
trianglen(a,d,n):={
local L;
si n<0 alors retourne NULL fsi;
L:=triangle_equilateral(a,a+d),trianglen(a,d/2,n-1),
trianglen(a+d/4+i*d*sqrt(3.)/4,d/2,n-1),trianglen(a+d/2,d/2,n-1);
retourne L;
}
:;
```

On tape : `trianglen(0,40,4)`

19.3.4 Exercice

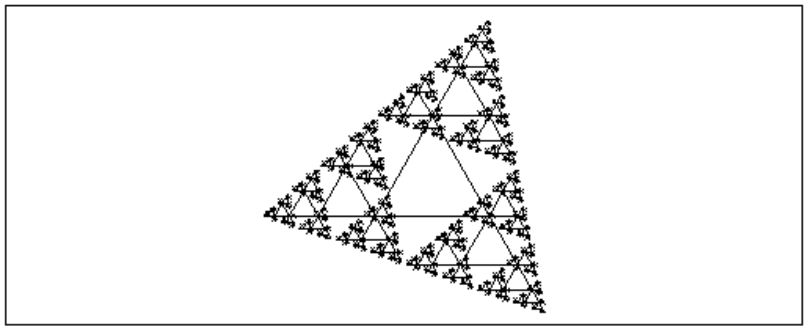
Écrire un programme qui réalise le dessin récursif dont on a mis ci-dessous les premières étapes (profondeur 0, 1 et 2) :



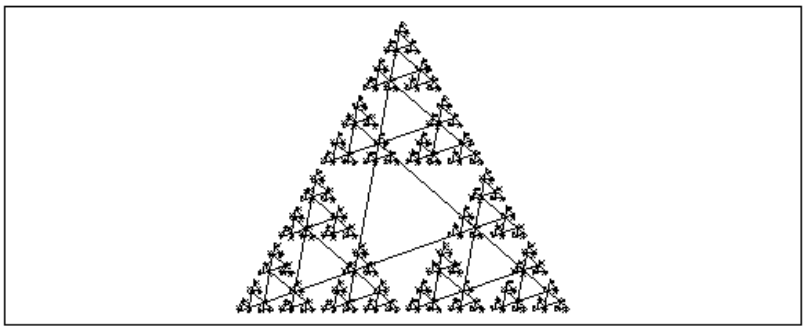
On tape :

```
triequi(A,B,n):={
local C,L,A1,B1,C1;
L:=triangle_equilateral(A,B,C);
si n>0 alors
A1:=homothetie(C,-0.5,A);
B1:=homothetie(A,-0.5,B);
C1:=homothetie(B,-0.5,C);
L:=L,triequi(A1,C,n-1);
L:=L,triequi(B1,A,n-1);
L:=L,triequi(C1,B,n-1);
fsi;
retourne L;
}:;
```

On tape : `triequi(0,1,5)`
On obtient :



On tape : `triequi(0,1+0.35*i,5)`
On obtient :



19.3.5 Des triangles équilatéraux emboiés

À partir d'un triangle équilatèral direct ABC on construit les points A_1, B_1, C_1 vérifiant :
$\overrightarrow{AA_1} = \frac{4}{3}\overrightarrow{AB}$
$\overrightarrow{BB_1} = \frac{4}{3}\overrightarrow{BC}$
$\overrightarrow{CC_1} = \frac{4}{3}\overrightarrow{CA}$
Interprétez A_1 comme le barycentre de A, a et B, b. Montrer que le triangle $A_1B_1C_1$ estéquilatèral.
On recommence la même construction à partir de $A_1B_1C_1$.
Écrire la procédure récursive qui réalise le dessin des n triangles obtenus par cette construction (en tout $n+1$ triangles ABC + les autres).
On a :
$\overrightarrow{AA_1} = \frac{4}{3}\overrightarrow{AB} - \frac{1}{3}\overrightarrow{AA}$
Donc A_1 est le barycentre de $A, -1$ et $B, 4$.
Donc B_1 est le barycentre de $B, -1$ et $C, 4$
Donc C_1 est le barycentre de $C, -1$ et $A, 4$.
La rotation r de centre O, le centre de ABC, et d'angle $2\pi/3$ transforme A en B, B en C et C en A donc r transforme le barycentre de $A, -1$ et $B, 4$ en le barycentre de $B, -1$ et $C, 4$ c'est à dire transforme A_1 en B_1 et r transforme le barycentre de $B, -1$ et $C, 4$ en le barycentre de $C, -1$ et $A, 4$ c'est à dire transforme B_1 en C_1.
Donc le triangle $A_1B_1C_1$ estéquilatèral.
On tape dans l'éditeur de programmes :

```
triangles(A,B,n):={
  local L,C;
  L:=triangle_equilateral(A,B,C);
  si n>0 alors
    A:=barycentre([A,B],[-1,4]);
    B:=barycentre([B,C],[-1,4]);
    L:=L,triangles(A,B,n-1);
  fsi;
  return L;
}:;
```

puis on compile avec `F9` et dans une ligne de commande, on tape :
`triangles(point(0),point(1),5)` On obtient :

19.3.6 Le problème des 3 insectes

Trois insectes partent, des sommets d'un triangle équilatéral A, B, C en direction de son voisin (C regarde B. B regarde C. A regarde C). À chaque étape de leur marche les 3 insectes forment un triangle équilatéral.
Dessiner les trajectoires des 3 insectes en résolvant une équation différentielle ou un système d'équations différentielles.
On peut faire une simulation de la situation en supposant que chaque insecte :
- regarde son voisin ce qui lui donne sa direction, puis,
- avance dans cette direction d'une longueur proprortionnelle au côté du triangle, puis , - regarde son voisin ce qui lui donne sa nouvelle direction etc...
Faire un programme qui dessine les triangles étapes de cete marche.
Refaire le même exercice en remplaçant le triangle équilatéral A, B, C par un triangle rectangle isocèle.

Résolution de 3 équations différentielles
Les trois insectes ont des trajectoires qui se déduisent l'une de l'autre par une rotation de centre G le centre de gravité du triangle et d'angle $-2 * \pi/3$.
Si zA est l'affixe du point A et zB celle du point B..., on a :
$zA' = zC - zA$
$zB' = zA - zB$
$zC' = zB - zC$
donc $zA' + zB' + zC' = 0$ et donc
$zA + zB + zC = cste = 1 + 1/2 + i\sqrt{3}/2 = 3 * zG$
On a :
$zC - zG = \exp(-2 * \pi/3)(zA - zG)$
$(zA-zG)' = zC-zA = (zC-zG)-(zA-zG) = (\exp(-2*\pi/3)-1)(zA-zG)$.

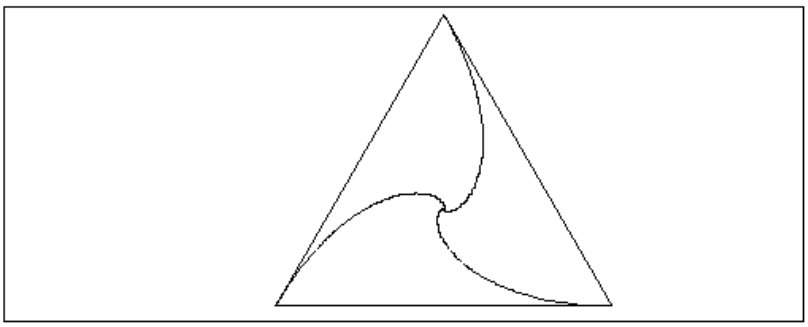
$zG = -(3 + i\sqrt{3})/6$
au temps t=0 on a : $zA = 0$, $zB = 1$, $zC = 1/2 + i\sqrt{3}/3$
On tape (on suppose que l'on a coché `complexe` dans la configuration du CAS) :

```
triangle_equlateral(0,1)
zG:=(3+i*sqrt(3))/6
SA:=simplify(desolve([diff(z(t),t)=(exp(-2*i*pi/3)-1)*(z(t)-zG),
              z(0)=0],[t,z]))
SB:=simplify(desolve([diff(z(t),t)=(exp(-2*i*pi/3)-1)*(z(t)-zG),
              z(0)=1],[t,z]))
SC:=simplify(desolve([diff(z(t),t)=(exp(-2*i*pi/3)-1)*(z(t)-zG),
              z(0)=1/2+i*sqrt(3)/2],[t,z]))
```

On obtient :

```
plotparam(SA[0],t=0..4),plotparam(SB[0],t=0..4),
plotparam(SC[0],t=0..4)
```

On obtient :



Résolution d'un système d'équations différentielles

On peut aussi résoudre le système :
$Z' = A * Z$ et au temps $t = 0$, $Z(0) = [0, 1, 1/2 + i * \sqrt{3}/2]$ avec

$$A := \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$

On tape (on suppose que `Complexe` est coché dans la configuration du CAS) :

```
A:=[[-1,1,0],[0,-1,1],[1,0,-1]]
P,B:=jordan(A)
```

On obtient pour P :

```
[[1,(-i)*sqrt(3)-1,(i)*sqrt(3)-1],[1,2,2],[1,(i)*sqrt(3)-1,
(-i)*sqrt(3)-1]]
```

On obtient pour B :

```
[[0,0,0],[0,((i)*sqrt(3)-3)/2,0],[0,0,((-i)*sqrt(3)-3)/2]]
```

On tape :

```
V0:=simplify(inv(P)*[0,1,1/2+i*sqrt(3)/2])
```

On obtient :

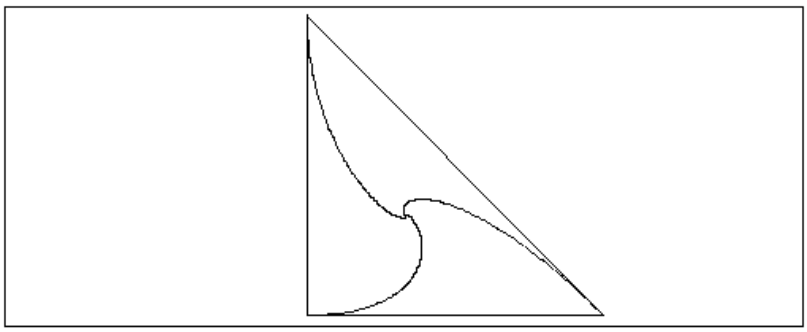
```
[((i)*sqrt(3)+3)/6,((-i)*sqrt(3)+3)/12,0]
```

On tape :

```
V:=V0*exp(B*t)
```

On obtient :
`[1/6*((i)*sqrt(3)+3),1/12*exp(((i)*sqrt(3)*t-3*t)/2)*((-i)*sqrt(3)+3),`
On tape :
```
Z:=P*V
ZA:=simplify(Z[0]);ZB:=simplify(Z[1]);ZC:=simplify(Z[2]);
plotparam(ZA,t=0..4),plotparam(ZB,t=0..4),plotparam(ZC,t=0..4),
triangle_equilateral(0,1)
```
On obtient la figure précédente.

Dans le cas du triangle ABC avec
`A:=point(0);B:=point(10);C:=point(i*10)`, le système à resoudre est le même c'est juste la condition initiale qui change (`V0:=inv(P)*[0,1,i]`) et les 3 insectes convergent vers le centre de gravité K du triangle ABC.
On tape (on suppose que `Complexe` est coché dans la configuration du CAS) :
```
A:=[[-1,1,0],[0,-1,1],[1,0,-1]]
P,B:=jordan(A)
V0:=simplify(inv(P)*[0,1,i])
```
On obtient :
```
[(1+i)/3,(sqrt(3)+2-i)/12,(-sqrt(3)+2-i)/12]
V:=V0*exp(B*t)
```
On obtient :
```
[(1+i)/3,1/12*exp(((i)*sqrt(3)*t-3*t)/2)*(sqrt(3)+2-i),
1/12*exp(((-i)*sqrt(3)*t-3*t)/2)*(-sqrt(3)+2-i)]
```
On tape :
```
Z:=P*V
ZA:=simplify(Z[0]);ZB:=simplify(Z[1]);ZC:=simplify(Z[2]);
plotparam(ZA,t=0..4),plotparam(ZB,t=0..4),plotparam(ZC,t=0..4),
triangle(0,1,i)
```
On obtient la figure :

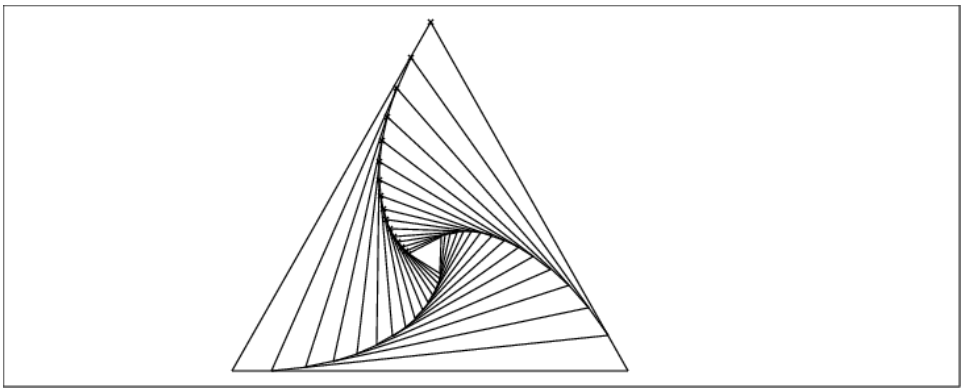


Le dessin des triangles
On dessine le triangle équilatéral ABC puis le triangle $A1B1C1$ avec :
$A1 = A + \text{evalf}((B - A)/10)$,

$B1 = B + \text{evalf}((C-B)/10)$ et
$C1 = C + \text{evalf}((A-C)/10)$.
puis on recommence le même processus avec $A1B1C1$... On tape :

```
triop0(a,b):={
local L,C,c;
L:=triangle_equilateral(point(a),point(b),C);
c:=evalf(affixe(C));
si evalf(abs(b-a))<1 alors return L; fsi;
a:=a*0.9+b*0.1;
b:=b*0.9+c*0.1;
L:=L,triop0(a,b);
return L;
}:;
```

On tape :
`triop0(0,10)`
On obtient :

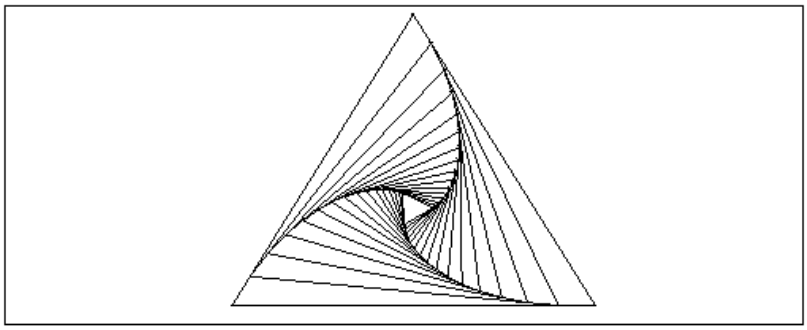


On peut tourner dans l'autre sens : on dessine le triangle équilatéral ABC puis le triangle $A1B1C1$ avec :
$A1 = A + \text{evalf}((C-A)/10)$,
$B1 = B + \text{evalf}((A-B)/10)$ et
$C1 = C + \text{evalf}((B-C)/10)$.
puis on recommence le même processus avec $A1B1C1$... On tape :

```
triop(a,b):={
local L,C,c,a0;
L:=triangle_equilateral(point(a),point(b),C);
c:=evalf(affixe(C));
si evalf(abs(b-a))<1 alors return L; fsi;
a0:=a;
a:=a*0.9+c*0.1;
b:=b*0.9+a0*0.1;
L:=L,triop(a,b);
return L;
}:;
```

On tape :
`triop(0,10)`

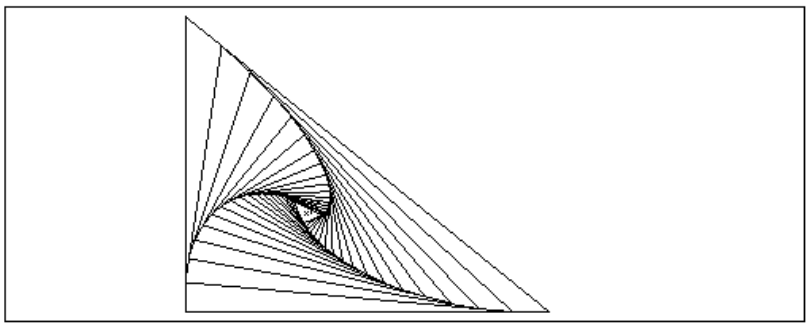
On obtient :



On tape si on choisit un triangle ABC quelconque :

```
triopa(a,b,c):={
local L,a0,b0;
L:=triangle(point(a),point(b),point(c));
si (evalf(abs(b-a))<1) alors return L; fsi;
a0:=a;
b0:=b;
a:=a+evalf((c-a)/10);
b:=b+evalf((a0-b)/10);
c:=c+evalf((b0-c)/10);triopa(0,10,i*10);K:=point((10+10*i)/3)
L:=L,triopa(a,b,c);
return L;
}
:;
```

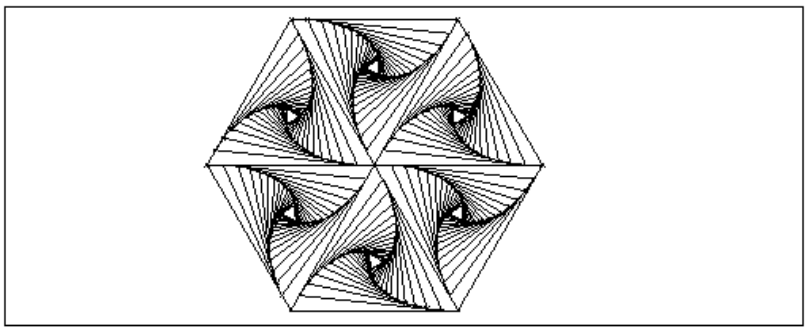
On tape :
`triopa(0,10,i*10);K:=point((10+10*i)/3)`
On obtient :



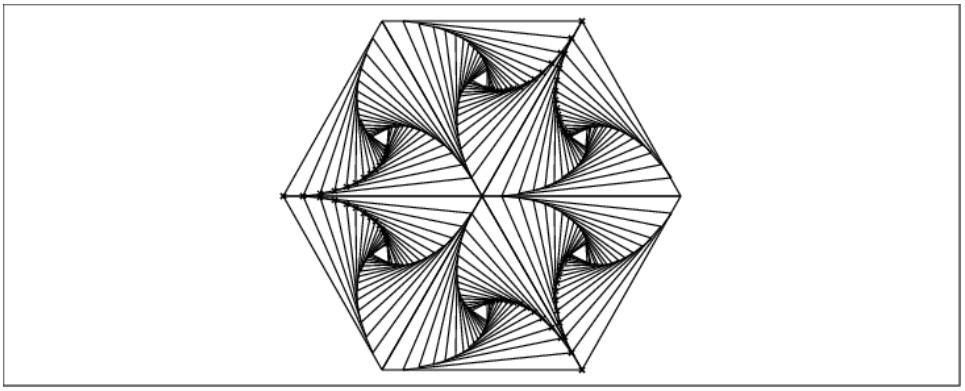
On fait faire des rotations à ces triangles
Dans triops on fait 6 rotations de `triop` alors que dans `triops0` on fait 3 rotations du losange formé par `triop0(a,b)` et `triop(b,a)`
0n tape :

```
triops(A,B):={
local L,j,a,b;
a:=affixe(A);
b:=affixe(B);
L:=triop0(a,b);
pour j de 1 jusque 5 faire
B:=rotation(A,pi/3,B);
b:=affixe(B);
L:=L,triop(a,b);
fpour;
return L;
}:;
triops0(A,B):={
local L,j,a,b;
a:=affixe(A);
b:=affixe(B);
L:=NULL;
pour j de 1 jusque 3 faire
L:=L,triop0(a,b),triop(b,a);
B:=rotation(A,pi/3,B);
b:=affixe(B);
fpour;
return L;
}:;
triops1(A,B):={
local L,j,a,b,c,C;
a:=affixe(A);
b:=affixe(B);
L:=NULL;
pour j de 1 jusque 3 faire
triangle_equilateral(B,A,C);
c:=affixe(C);
L:=L,triop0(a,b),triop(b,a),,triop0(a,c);
B:=rotation(A,2*pi/3,B);
b:=affixe(B);
fpour;
return L;
}:;
```

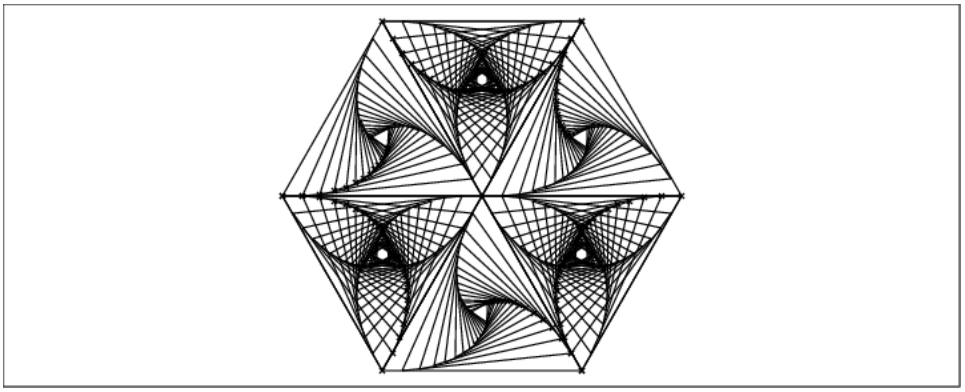
On tape :
`triops(point(0),point(10))`
On obtient :



On tape :
`triops0(point(0),point(10))`
On obtient :



On tape :
`triops1(point(0),point(10))`
On obtient :

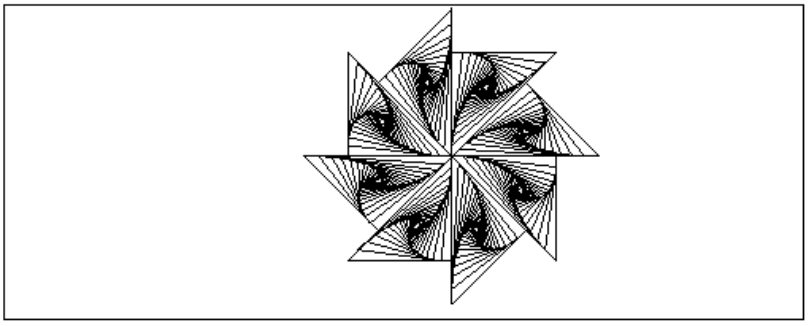


0n tape :

```
triopas(A,B,C):={
local L,j,F,a,b,c;
a:=affixe(A);
b:=affixe(B);
c:=affixe(C);
L:=triopa(a,b,c);
```

```
pour j de 1 jusque 7 faire
A:=rotation(B,pi/4,A);
C:=rotation(B,pi/4,C);
a:=affixe(A);
c:=affixe(C);
L:=L,triopa(a,b,c);
fpour;
return L;
}:;
```

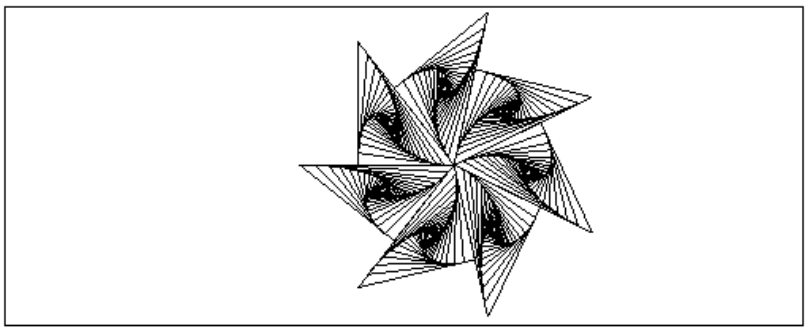
On tape :
`triopas(point(0),point(10),point(10*i))`
On obtient :



Avec le triangle `point(0),point(10),point(i*10*tan(2*pi/7))`, on tape :

```
triopas7(A,B,C):={
local L,j,F,a,b,c;
a:=affixe(A);
b:=affixe(B);
c:=affixe(C);
L:=triopa(a,b,c);
pour j de 1 jusque 6 faire
A:=rotation(B,2*pi/7,A);
C:=rotation(B,2*pi/7,C);
a:=affixe(A);
c:=affixe(C);
L:=L,triopa(a,b,c);
fpour;
return L;
}:;
```

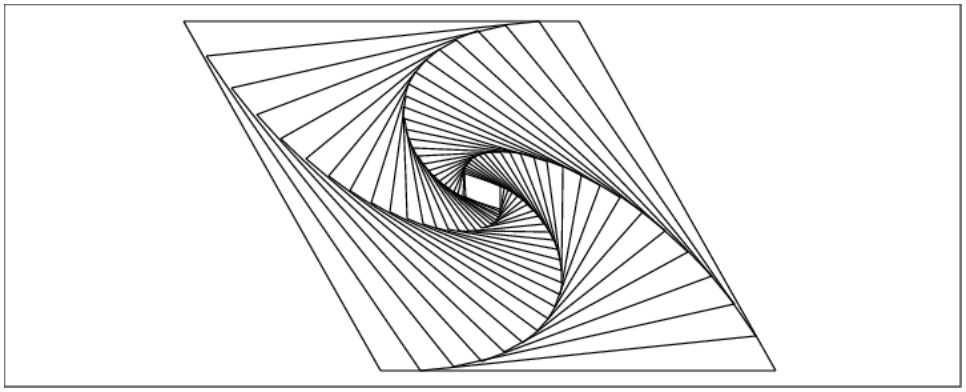
Puis, on tape :
`triopas7(point(0),point(10),point(i*10*tan(2*pi/7)));`
On obtient :



On peut aussi faire la mêmes chose ave des losanges
On tape :

```
losop(a,b,c,d):={
local L;
L:=quadrilatere(point(a),point(b),point(c),point(d));
si evalf(abs(b-a))<1 alors return L; fsi;
a:=a*0.9+b*0.1;
b:=b*0.9+c*0.1;
c:=c*0.9+d*0.1;
d:=d*0.9+a*0.1;
L:=L,losop(a,b,c,d);
return L;
}:;
```

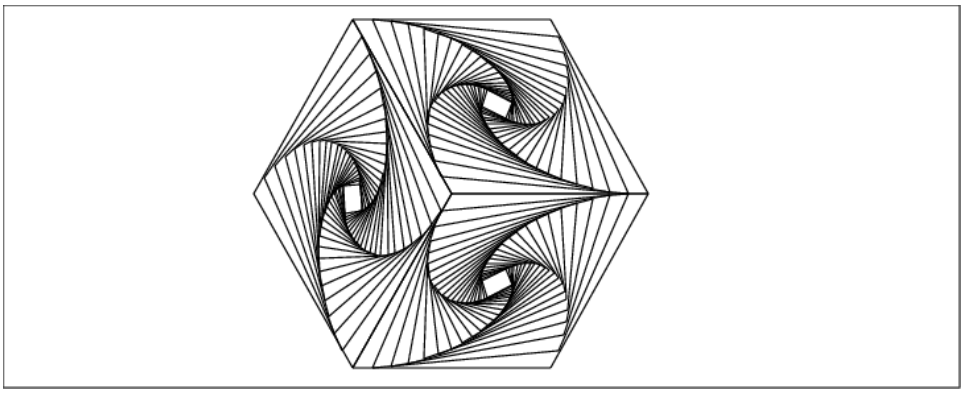
On tape :
`losop(0,10,5+i*sqrt(3)*5,-5+i*sqrt(3)*5)`
On obtient :



On tape :

```
losop(10,0,-5+i*sqrt(3)*5,5+i*sqrt(3)*5),
losop(10,0,-5-i*sqrt(3)*5,5-i*sqrt(3)*5),
losop(-5+i*sqrt(3)*5,0,-5-i*sqrt(3)*5,-10)
```

On obtient :

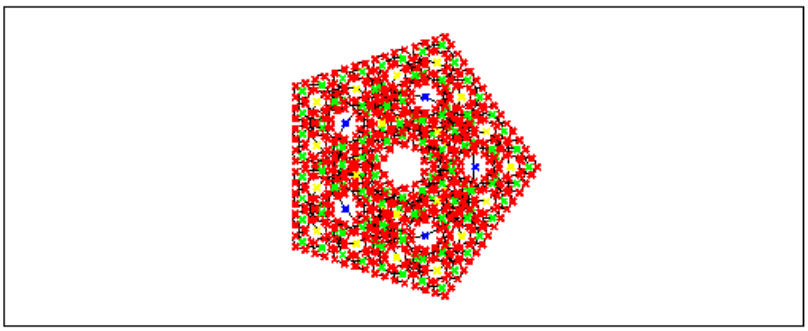


19.3.7 Les cercles

L'utilisateur choisit un entier `n`. Sur le cercle de de centre d'affixe `a` et de rayon `r` on dessine le cercle et les `n` points d'affixe `ak:=r*exp(2.*i*k*pi/n)` pour `k=0..n-1`. On recommence pour chaque `k` avec des cercles de de centre d'affixe `ak` et de rayon `r/2`. Et ainsi de suite à partir des points obtenus en divisant à chaque étape le rayon par 2. Écrire un programme qui réalise `p` étapes de ce processus. On tape :

```
cercles(a,r,n,p):={
local P,L,k,j;
P:=NULL;
si p<1 alors retourne NULL fsi;
pour k de 0 jusque n-1 faire
  P:=P,point(a+r*exp(2.*i*pi*k/n),affichage=p+epaisseur_point_2);
fpour;
L:=cercle(a,r),P;
pour j de 0 jusque n-1 faire
  L:=L,cercles(affixe(P[j]),r/2,n,p-1);
fpour;
retourne L;
}:;
```

On tape :
`cercles(0,20,5,4)`
On obtient :



19.4 Les tours de Hanoï

Une tour de Hanoï est composée de p disques de rayons différents que l'on numérote de 1 à p selon l'ordre croissant des rayons (le plus petit disque a le numéro 1 et le plus gros le numéro p). On dispose de 3 plots numérotés de 1 à 3.
Au départ les disques sont empilés selon l'ordre croissant sur le plot 1.
Le jeu consiste à reconstituer la tour sur le plot 2, en se servant du plot 3 comme plot intermédiaire, en déplacant les disques un à un, et en posant toujours un disque sur un disque plus petit que lui.
Par exemple, on peut mettre le dique 2 sur le disque 5, mais pas sur le disque 1.
On veut écrire un programme qui imprime ce qu'il faut faire comme manipulations : ce sera `tour(a,b,c,p)`, où `p` représente le nombre de disques, où `a` représente le plot de départ, où `b` représente le plot d'arrivée, et où `c` représente le plot intermédiaire.
On tapera alors par exemple :
`tour(1,2,3,4)`
si on a une tour de 4 disques sur le plot 1, et qu'on veut la reconstituer sur le plot 2 par l'intermédiaire du plot 3.
Les manipulations à faire sont récursives, en voici les étapes :

— iL faut arriver à dégager le disque p pour qu'il soit seul sur le plot 1 avec les $p-1$ disques sur le plot 3, le plot 2 étant vide et prêt à recevoir le dique p. Dans cette situation, il y une tour de $p-1$ disques sur le plot 3. Cela veut dire que l'on est arrivé à reconstituer la tour constituée des $p-1$ premiers disques sur le plot 3, en se servant du plot 2 comme plot intermédiaire.
Avec les notations ci dessus c'est que l'on a effectué :
`tour(a,c,b,p-1)`
c'est l'instruction qui permet de reconstituer une tour de $p-1$ disques sur le plot 3 en partant du plot 1 et en se servant du plot 2 comme plot intermédiaire.

— on déplace ensuite le disque p du plot 1 sur le plot 2 :
`print("deplacer le disque ",p," de ", a , " vers", b).`
- il reste à reconstituer la tour de $p-1$ disques sur le plot 2 en partant du plot 3 et en se servant du plot 1 comme plot intermédiaire.
Il faut donc effectuer :
`tour(c,b,a,p-1)`

— il reste à trouver le test d'arrêt qui est simplement `(p==0)` c'est à dire : quand on a une tour de zéro disque on ne fait rien (on renvoie 0).

On tape dans un niveau éditeur de programmes (que l'on ouvre avec `Alt+p`), puis on le teste et on le valide avec `OK` :

```
//tour(1,2,3,4) (tour de hanoi)
//depacement des p disques (de numero 1..p du plus petit au plus grand
//de a vers b en passant par c
tour(a,b,c,p) :={
  if (p==0) return 0;
  tour(a,c,b,p-1);
```

```
  print("deplacer le disque "+p+" de "+ a + " vers "+ b);
  tour(c,b,a,p-1);
  return 0;
}:;
```

On tape :
`tour(1,2,3,4)`
On obtient :

```
deplacer le disque 1 de 1 vers 3
deplacer le disque 2 de 1 vers 2
deplacer le disque 1 de 3 vers 2
deplacer le disque 3 de 1 vers 3
deplacer le disque 1 de 2 vers 1
deplacer le disque 2 de 2 vers 3
deplacer le disque 1 de 1 vers 3
deplacer le disque 4 de 1 vers 2
deplacer le disque 1 de 3 vers 2
deplacer le disque 2 de 3 vers 1
deplacer le disque 1 de 2 vers 1
deplacer le disque 3 de 3 vers 2
deplacer le disque 1 de 1 vers 3
deplacer le disque 2 de 1 vers 2
deplacer le disque 1 de 3 vers 2
```

19.5 Les permutations

19.5.1 Les permutations circulaires

La liste l est une liste de nombres tous différents.
On écrit la fonction `circulaire(l)` qui renvoie la liste obtenue à partir de `l` en renvoyant le début de la liste `l` à la fin de `l`.

```
//l:=[1,2,3]; circulaire(l)
//renvoie la liste l ou la tete est mise a la fin.
circulaire(l):={
return concat(tail(l),l[0]);
};
```

On écrit la fonction `permcir(l)` qui renvoie la liste des permutations circulaires obtenues à partir de `l`.
`permcir(l)` est donc une matrice carrée d'ordre `n=size(l)`.
On écrit cette fonction récursivement en renplacant `l` par `circulaire(l)`. Il faut un test d'arrêt pour ce parcours, pour cela on a besoin d'un paramètre supplementaire qui sera `ld` : c'est une liste de référence égale à `l` au départ et qui n'est pas modifiée. On s'arrête quand `circulaire(l)==ld`, c'est à dire quand on retrouve la liste de départ.
On utilise une variable locale `lr` égale à la liste à renvoyer.

```
// utilise circulaire, l:=[1,2,3];permcir(l,ld);
```

```
//renvoie les permutations circulaires de l
//variable locale lr la liste resultat
// ld liste reference de depart
permcir(l,ld):={
local lr;
if (circulaire(l)==ld) alors return [l];fsi;
lr:=[l];
lr:= append(lr,op(permcir(circulaire(l),ld)));
return lr;
};
```

On peut supprimer la variable locale `lr` et la fonction `circulaire`.
On écrit alors la fonction `permcc(l)` qui renvoie la liste des permutations circulaires obtenues à partir de `l`.
Ici, on utilise un autre test d'arrêt, on a toujours besoin d'un paramètre supplementaire qui sera `ld` : c'est une liste de référence égale à `l` au départ et qui est modifiée, sa taille diminue de 1 à chaque appel récursif.
On s'arrête quand `ld==[]`, c'est à dire quand on a fait autant d'appels que la taille de `l`.

```
//l:=[1,2,3];permcc(l,l);
//renvoie les permutations circulares de l
//sans variable locale, ld liste reference de depart
permcc(l,ld):={
if (ld==[]) return [];
return [l,op(permcc(concat(tail(l),l[0]),tail(ld)))];
};
```

Comme il faut 2 paramètres pour écrire la fonction récursive `permcc`, on écrit la fonction finale `permutation_circ` qui utilise `permcc` :

```
//l:=[1,2,3];permutation_circ(l);
//renvoie les permutations circulares de l
//utilise permcc
permutation_circ(l):={
return permcc(l,l);
};
```

On tape :
`permutation_circ([1,2,3])`
On obtient :
`[[1,2,3],[2,3,1],[3,1,2]]`

19.5.2 Programme donnant toutes les permutations d'une liste l

La liste l est une liste de nombres tous différents.
1/ En faisant n=`size(l)` appels récursifs.
Les fonctions que l'on va écrire vont utiliser la fonction `echange`.

```
//echange ds l les elements d'indices j et k
echange(l,j,k):={
```

```
local a;
a:=l[j];
l[j]:=l[k];
l[k]:=a;
return l;
}:;
```

On peut décrire l'arbre des permutations de la liste l :
à partir de la racine on a n=`size(l)` branches. Chaque branche commence respectivement par chacun des éléments de la liste l.
On va donc parcourir cet arbre de la racine (nœud de niveau 0) aux différentes extrémités, en renvoyant la liste des branches parcourues pour arriver à cette extrémité.
On va parcourir cet arbre en parcourant les n branches. On numérote ces n branches par $p = 1..n$ et le niveau des nœuds $q = 0..n-1$.
On aura donc n appels récursifs.
Chaque branche p ($p = 1..n$) peut être considérée à leur tour comme un arbre ayant $n-1$ branches. La branche p aboutit aux permutations qui laissent invariant le p-ième élément de l ($l[p-1]$). C'est cet élément que l'on va échanger avec $l[0]$ pour que chaque branche p laisse invariant l' élément $l[0]$.
On sait que l'on est arrivé au bout de la branche, quand on se trouve au nœud de niveau $n-1$, dans ce cas la permutation chechée est l (c'est la permutation obtenue à partir de l en laissant ces $n-1$ premiers éléments invariants).
On utilise une variable locale `lr`, égale à la liste à renvoyer et un paramètre `k`, pour que `permus(l,k)` renvoie toutes les permutations de `l` qui laissent invariant les k premiers éléments de `l`. On tape :

```
//utilise echange et la variable locale lr (liste resultat)
//permus(l,k) laisse invariant les k premiers elements de l
//permus([1,2,3,4],0); renvoie toutes les permutations de l
permus(l,k):={
local lr,j;
if (k==size(l)-1) return [l];
lr:=[];
for (j:=k;j<size(l);j++){
l:=echange(l,k,j);
lr:=[op(lr),op(permus(l,k+1))];
l:=echange(l,j,k);
}
return lr;
}:;
```

On n'est pas obligé de remettre la suite l à sa valeur de départ pour recommencer l'itération puisque le premier échange dans l'itération revient à transformer l en la liste où on a mis son j-ième élément en tête ($j = 0..n-1$). La liste résultat ne sera alors pas dans le même ordre. Si on veut avoir la liste dans l'ordre lexicographique, il ne faut pas mettre la deuxième instruction `echange`. En effet :
sans la deuxième instruction `echange`, on échange 0 et 1 pour j=1 ([1,0,2..]) puis 0 et 2 pour j=2 ([2,0,1..]) etc sans la deuxième instruction `echange`, on échange

0 et 1 ([1,0,2..]) puis, 1 et 0 ([0,1,2..]) pour j=1, puis 0 et 2 ([2,1,0..]) puis, 2 et 0 ([0,1,2..]) pour j=2 etc

```
//permuts([1,2,3,4],0) utilise echange
//la 2ieme instruction echange est inutile
permuts(l,k):={
local lr,j;
if (k==size(l)-1) return [l];
lr:=[];
for (j:=k;j<size(l);j++){
l:=echange(l,k,j);
lr:=[op(lr),op(permuts(l,k+1))];
}
return lr;
}:;
```

Comme il faut 2 paramètres pour écrire la fonction récursive `permuts`, on écrit la fonction `permutation` qui utilise `permuts` :

```
//l:=[1,2,3];permutation(l);
//renvoie toutes les permutations de l
//utilise permuts
permutation(l):={
return permuts(l,0);
};
```

On tape :
`permutation([1,2,3])`
On obtient :
`[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`
On peut aussi écrire une autre fonction récursive ayant comme paramètre `ld` et `lf`. `ld` contient les premières valeurs de `l` qui seront inchangées dans la permutation et `lf` contient les valeurs restantes de `l`, celles qui restent à permuter. On remarquera que le résultat mis dans `res` est ici une séquence.

```
//au debut ld=[] et lf=l,
//groupe_s([],l) renvoie toutes les permutations de l
groupe_s(ld,lf):={
 local j,n,res;
 n:=size(lf);
 res:=NULL;
 if (n==1)
   return concat(ld,lf);
 for (j:=0;j<n;j++){
   res:=res,groupe_s(append(ld,lf[0]),tail(lf));
   // permutation circulaire
   lf:=append(tail(lf),lf[0]);
 }
 return res;
};
```

Et la fonction `groupesym` qui utilise la fonction récursive `groupe_s` :

```
//utilise groupe_s
//groupesym(l) renvoie toutes les permutations de l
groupesym(l):=return(groupe_s([],l));
```

2/ En faisant 2 appels récursifs.
Cet algorithme est surtout fait pour des langages qui n'ont pas de boucle `for`.
Les fonctions vont utiliser la fonction `circulaire` (pour plus de clareté), puis on remplacera `circulaire(l)` par `concat(tail(l),l[0])`.

```
//l:=[1,2,3]; circulaire(l)
//renvoie la liste l ou la tete est mise a la fin.
circulaire(l):={
return concat(tail(l),l[0]);
};
```

On peut décrire l'arbre des permutations de la liste l :
à partir de la racine on a n=`size(l)` branches. Chaque branche commence par chacun des éléments de la liste l.
On va parcourir cet arbre, en parcourant la premiére branche, puis en considérant qu'il reste à parcourir un arbre de $n-1$ branches.
On aura donc 2 appels récursifs.
Pour le parcours de la première branche, il faut connaitre la liste des éléments qui nous a permis d'arriver à un nœud donné, c'est cette liste que l'on met dans `ldl`, `l` contenant les éléments qu'il faut encore permuter. On s'arrête quand `l=[]`, et le résultat est `[ldl]`.
Pour le parcours des $n-1$ branches restantes, on change pour chaque branche la liste à permuter en `circulaire(l)`.
Il faut un test d'arrêt pour ce parcours, pour cela on a besoin d'un paramètre supplementaire qui sera `ld` (liste de référence égale à `l` au départ) dans `permss` ou qui sera `n` (longueur de `l` au départ) dans `permss1`.
On écrit `permss` :

```
// utilise circulaire, l:=[1,2];permss([],l,l);
//ldl=debut de l, l=liste a permuter,
//ld=liste de reference (=l au debut)
permss(ldl,l,ld):={
if (l==[]) return [ldl];
if (ld==[]) return ld;
return [op(permss(concat(ldl,l[0]),tail(l),tail(l))),
        op(permss(ldl,circulaire(l),tail(ld)))];
};
```

On écrit `permss1` qui utilise comme paramètre `n` qui représente la longueur de la liste qui reste à permuter (`n=size(l)` au départ) :

```
//utilise circulaire, l:=[1,2,3,4];permss1([],l,size(l));
//ldl=debut de l, l=liste a permuter, n=size(l) au debut
permss1(ldl,l,n):={
```

```
if (l==[]) return [ldl];
if (n==0) return [];
return [op(permss1(concat(ldl,l[0]),tail(l),size(tail(l)))),
        op(permss1(ldl,circulaire(l),n-1))];
};
```

On a aussi écrit la fonction `permss2` contenant une variable locale `lr` qui est la liste à renvoyer et qui donne un algorithme plus lisible.

```
//l:=[1,2];permss2([],l,l);
//ldl=debut de l, l=liste a permuter,
//ld=liste de reference (=l au debut)
// lr liste a renvoyer en variable locale
permss2(ldl,l,ld):={
local lr;
if (l==[]) return [ldl];
if (ld==[]) return [];
lr:=permss2(concat(ldl,l[0]),tail(l),tail(l));
lr:=append(lr,op(permss2(ldl,concat(tail(l),l[0]),tail(ld))));
return lr
};
```

puis la fonction `permute` qui utilise `permss2` :

```
//utilise permss2,
//permute(l) renvoie toutes les permutations de l
permute(l):={
return permss2([],l,l);
};
```

On tape :
`permute([1,2,3])`
On obtient :
`[[1,2,3],[1,3,2],[2,3,1],[2,1,3],[3,1,2],[3,2,1]]`

Chapitre 20

Récupérer et installer un logiciel

La description qui suit s'applique au navigateur Netscape. Elle s'applique de manière identique pour d'autres navigateurs à quelques détails près.

Lorsqu'un lien mentionne un logiciel que vous voulez télécharger, maintenez la touche Shift appuyée et cliquez avec la souris sur ce lien. Une fenêtre s'ouvre et vous propose de sauvegarder un fichier sur votre disque dur, changez éventuellement le nom de ce fichier et notez l'emplacement du répertoire où il a été sauvegardé.

La méthode d'installation du logiciel dépend de votre système d'exploitation et du type de fichier téléchargé, que l'on détermine en général par son extension, c'est-à-dire par les lettres (en principe 3) qui suivent le caractère point dans le nom de ce fichier.

Sous Windows, vous récupérerez en général un fichier exécutable `exe` il suffit alors de cliquer sur son icône pour installer le logiciel. Ou alors il s'agira d'une archive, il vous faudra alors un logiciel de décompression pour l'installer (l'un des plus populaires s'appelle Winzip et propose une version à l'essai gratuitement). Une fois désarchivé votre logiciel, vous trouverez dans l'archive un fichier donnant des instructions complémentaires ou vous devrez vous référer au site où vous avez téléchargé l'archive.

Sous Linux, les formats les plus courants sont `rpm`, `deb` et `tgz` (ou `tar.gz`).

Les deux premiers correspondent à des distributions Linux (Red Hat, Mandrake, Suse par exemple pour `rpm`, Debian pour `deb`) ils doivent être installés par `root` (tapez `su` dans une fenêtre de commandes pour passer `root`) en utilisant respectivement les commandes :
`rpm -U nom_de_fichier.rpm` pour Red Hat, Mandrake, Suse
`apt-get install nom_de_fichier_deb` pour Debian
Regardez bien les éventuels messages d'erreurs qui apparaissent. Ils peuvent signaler que vous devez installer d'autres logiciels de votre distribution Linux pour faire fonctionner votre nouveau logiciel. En général ces logiciels non installés se trouvent sur le CD-ROM de votre distribution Linux. Ils s'installent par la même commande, mais vous devrez d'abord vous déplacer dans le répertoire du CD-ROM qui les contient. Pour cela, introduisez le CD-ROM dans le lecteur, attendez quelques secondes, tapez : `ls /mnt/cdrom`
(si rien n'apparait, tapez : `mount /mnt/cdrom` et recommencez).
Ensuite tapez `cd /mnt/cdrom` puis allez dans le répertoire correct (par exemple `cd Mandrake/RPMS` pour une distribution Mandrake) et lancez les commandes

d'installation des logiciels manquants (`rpm -U ...` ou `apt ...`). Certaines distributions Linux proposent des interfaces plus ou moins conviviales pour installer des logiciels, par exemple `rpmdrake` pour Mandrake, `dselect` pour Debian...

Les archives `tar.gz` et `tgz` s'installent sur toute distribution Linux. Commencez par en regarder la table des matières par la commande :
`tar tvfz nom_de_fichier.tgz`
afin de déterminer depuis quel répertoire il faut décompacter le fichier ou lisez la documentation du logiciel disponible en général sur le site où vous avez téléchargé l'archive. Si vous voyez apparaitre des chemins tels que `usr/local/bin` ou `usr/bin`, placez vous dans le répertoire racine (`cd /`) puis tapez :
`tar xvfz nom_du_repertoire/nom_de_fichier.tgz`
(si vous avez sauvegardé l'archive dans votre répertoire, vous pouvez utiliser :
`~votre_nom_de_login` comme `nom_de_repertoire`),
sinon, tapez simplement depuis le répertoire où se trouve l'archive :
`tar xvfz nom_de_fichier.tgz`
placez-vous dans le répertoire créé et lisez les fichiers `README` ou/et `INSTALL` qui s'y trouvent sans doute.
Remarques :

1/ N'oubliez pas de quitter le compte de `root` avant d'utiliser votre logiciel (en tapant simultanément sur les touches Ctrl et D).

2/ les archives `.tar.bz2` se traitent de manière identique, à ceci près qu'il faut enlever le `z` de `tvfz` ou `xvfz` et rajouter
`--use-compress-program bunzip2`, par exemple :
`tar tvf nom_de_fichier.tar.bz2 --use-compress-program bunzip2`

3/ les archives `.zip` se visualisent par la commande :
`unzip -v nom_de_fichier.zip` et se désarchivent par la commande :
`unzip nom_de_fichier.zip`

Sous linux et autres Unix, vous trouverez également des fichiers sources qu'il vous faudra compiler avant de pouvoir utiliser le logiciel correspondant. De nombreux fichiers sources suivent la procédure d'installation du projet GNU. Pour les compiler la procédure est la suivante (après avoir décompressé l'archive) :
`cd nom_d_archive`
`./configure`
`make`
Puis on passe `root` en tapant `su`, puis : `make install-strip`
puis quittez le compte `root` (en tapant simultanément sur les touches `Ctrl` et `D`).
Vous pourrez alors utiliser le logiciel nouvellement installé (éventuellement après avoir tapé la commande `rehash` si votre interpréteur de commandes est `tcsh`).

Si cela ne fonctionne pas vous devriez trouver un fichier `README` ou `INSTALL` qui explique la procédure à suivre.

Index

Table des matières