FAIRE DES MATHÉMATIQUES AU LYCÉE EN PROGRAMMANT

Quelques exercices de programmation avec XCAS et CAML

Guillaume Connan - IREM de Nantes

Courriel : gconnan - at - free.fr

Stage PAF du 25 mars 2010

1 - Quelques exemples pour observer

A Partie entière

A 1 Algorithme récursif

On part du fait que la partie entière d'un nombre appartenant à $[0;1[$ est nulle.
Ensuite, on « descend » de x vers 0 par pas de 1 si le nombre est positif en montant que $\lfloor x \rfloor = 1 + \lfloor x - 1 \rfloor$.
Si le nombre est négatif, on « monte » vers 0 en montrant que $\lfloor x \rfloor = -1 + \lfloor x + 1 \rfloor$.

A 1 a Avec XCAS

```
perx(r):={
  si(r>=0) et (r<1)
      alors 0
      sinon si r>0
            alors 1+perx(r-1)
            sinon -1+perx(r+1)
            fsi
  fsi
};;
```

Programme 1 – partie entier en récursif

A 1 b Avec CAML

```
# let perc (r)
       =
       if r >= 0. && r < 1.
        then 0.
        else if r > 0.
              then   1. +. perc (r -. 1.)
              else -.1. +. perc (r +. 1.)
```

Programme 2 – partie entière en récursif (CAML)

A 2 Algorithme impératif

On peut commencer par se restreindre aux nombres positifs. On part de 0. Tant qu'on n'a pas dépassé x, on avance d'un pas. La boucle s'arrête dès que k est strictement supérieur à x. La partie entière vaut donc $k-1$.

Entrées : x(réel positif)
Initialisation : k ⟵ 0
début
  tant que $k<=x$ **faire**
    k⟵ k+1
  retourner $k-1$
fin

Algorithme 1 : partie entière d'un nombre positif

Ce qui donne en XCAS

```
pe(x):={
local k;
k:=0;
   tantque k<=x faire
      k:=k+1;
   ftantque;
retourne(k-1);
};;
```

Programme 3 – partie entière en impératif (cas x positif)

B Les réels et le processeur

Ayez toujours en tête qu'un ordinateur ne peut pas travailler avec des réels ! Par essence, il a un nombre fini de bits sur lesquels il peut coder des nombres...
Les nombres *flottants* qu'il manipule sont donc des classes qui représentent chacune une infinité de réels (mais tout en constituant un intervalle cohérent).
Si on n'y fait pas attention, cela peut créer des surprises...
Entrons trois nombres sur XCAS :

```
x:=10;
y:=-5;
z:=5.00000001;
```

Alors

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

renvoie :

```
1.000000012e-07
```

Mais

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

renvoie :

```
9.999999939e-08
```

Le problème est le même avec CAML :

```
# let x,y,z=10.,-5.,5.00000001;;
val x : float = 10.
val y : float = -5.
val z : float = 5.00000001

# (x*.y)+.(x*.z);;
- : float = 1.00000001168609742e-07

# x*.(y+.z);;
- : float = 9.99999993922529e-08
```

Ainsi l'addition n'est pas toujours distributive sur la multiplication quand on travaille sur les flottants !
Ces petites erreurs qui s'accumulent peuvent en créer de grandes !
Créons une fonction qui renvoie l'approximation de la dérivée d'une fonction en un nombre donné avec un « dx » valant 10^{-10} :

```
# let dernum(f)=function
          x->(f(x+.0.0000000001)-.f(x))/.0.0000000001;;
```

Programme 4 – dérivation numérique

Dérivons le sinus cinq fois de suite et évaluons la fonction obtenue en 0 :

```
# dernum(dernum(dernum(dernum(dernum(sin)))))(0.);;
- : float = -1.33226762955018773e+25
```

Oups ! On obtient $\cos(0) \approx -10^{25}$...Ce n'est plus de l'ordre de l'erreur négligeable !

C Méthode d'Euler

Supposons qu'on connaisse $f'(x)$ en fonction de x et de $f(x)$ sous la forme $f'(x) = u\Big(f(x), x\Big)$.
On utilise le fait que $f'(x) \approx \frac{f(x+h)-f(x)}{h}$. Alors $f(x+h) \approx h \cdot u\Big(f(x), x\Big) + f(x)$
La traduction algorithmique est alors directe.

C 1 Version récursive

Pour la liste des coordonnées de points :

```
Euler(u,x,xmax,y,h,liste):={
 if(x>=xmax)
   then{liste}
   else{ Euler(u,x+h,xmax,y+u(y,x)*h,h,[op(liste),[x,y]])}
}:;
```

Programme 5 – méthode d'Euler générale en récursif

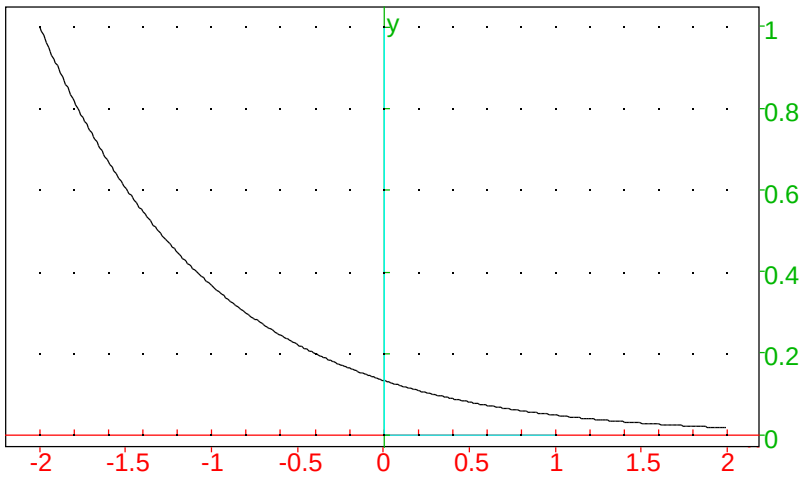
Puis pour le tracé :

```
trace_euler(u,xo,xmax,yo,h):={
polygonplot([Euler(u,xo,xmax,yo,h,[])])
}:;
```

Alors, pour résoudre graphiquement $y' = -y$ avec $y(0) = 1$, sur $[-2;2]$ avec un pas de $0,01$ on entre :

```
trace_euler((y,x)->-y,-2,2,1,0.01)
```

et on obtient :



C2 Version impérative

On effectue une division régulière en découpant notre intervalle d'étude $[a;b]$ en N points :

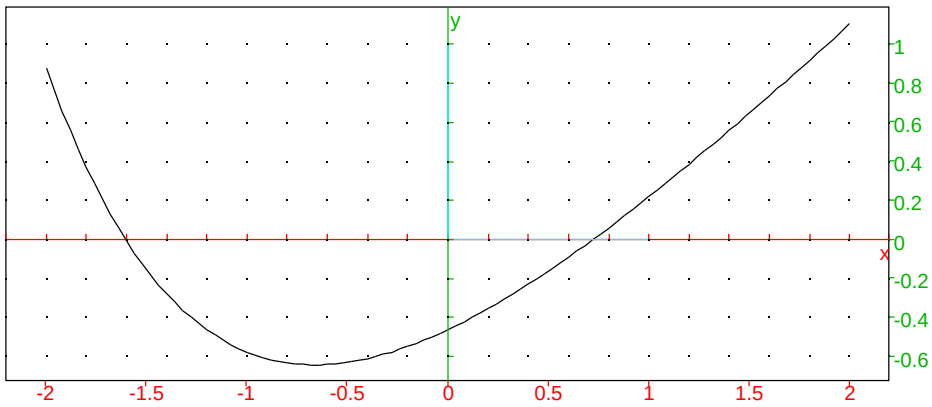
```
EulerImp(u,N,a,b,yo):={
S:=NULL;
X:=a;
Y:=yo;
h:=(b-a)/N
 pour k de 0 jusque N pas 1 faire
  X:=a+k*h;
  Y:=Y+u(Y,X)*h;
  S:=S,[X,Y];
 fpour
polygonplot([S]);
};;
```

Programme 6 – méthode d'Euler générale en impératif

Ensuite, pour avoir la solution de $y' = -y + x$ sur $[-2;2]$ avec $y(0) = 1$ on entre :

```
EulerExpo((y,x)->-y+x,100,-2,2,1)
```

et on obtient :



D Méthode des rectangles

On veut calculer une intégrale d'une fonction par la méthode des rectangles.

D 1 Version récursive

La définition récursive est assez naturelle :

```
# let rec integ(f,a,b,dx)=
      if a>b then 0.
      else f(a)*.dx+.integ(f,a+.dx,b,dx);;
```

Programme 7 – méthode des rectangles en récursif

Par exemple :

```
# let g(x)=x;;

# integ(g,0.,1.,0.0001);;
```

donne :

```
- : float = 0.500049999999960249
```

C'est-à-dire $\int_0^1 x\,\mathrm{d}x \approx 0.5$

Mais la récursion n'est pas terminale. Introduisons une fonction intermédiaire :

```
# let rec integ_temp(f,a,b,dx,res)=
     if a>b then res
     else integ_temp(f,a+.dx,b,dx,res+.f(a)*.dx);;
```

Programme 8 – méthode des rectangle en récursif terminal étape 1

Puis :

```
# let integrale(f,a,b,dx)=integ_temp(f,a,b,dx,0.);;
```

Programme 9 – méthode des rectangles en récursif terminal étape 2

Alors on obtient :

```
# integrale(g,0.,1.,0.0000001);;
- : float = 0.500000049944005598
```

Pour améliorer l'approximation, il faudrait en fait améliorer la méthode mathématique...

D 2 Version impérative

```
integ_rectangle(f,a,b,dx):={
local aa,I;
aa:=a;
I:=0;
 while(aa<b){
 I:=I+f(aa)*dx;
 aa:=aa+dx;
 }
```

```
return(I)
}:;
```

Programme 10 – méthode des rectangles en impératif

E Les vecteurs, le renard et le lapin

Un lapin court vers le nord en ligne droite et un renard le poursuit en bondissant toujours dans sa direction. On suppose que le lapin et le renard fond des bonds de longueurs respectives bl et br, que le lapin part du point de coordonnées $(0;0)$ et le renard du point de coordonnées (x_0, y_0).
On repérera ensuite les animaux par le point $\mathrm{R}(x;y)$ et le point $\mathrm{L}(0;z)$.
Calculons la distance Renard-Lapin : comme $\overrightarrow{\mathrm{RL}}(-x;z-y)$ alors la distance d qui les sépare vaut

$$d = \sqrt{x^2 + (y-z)^2}$$

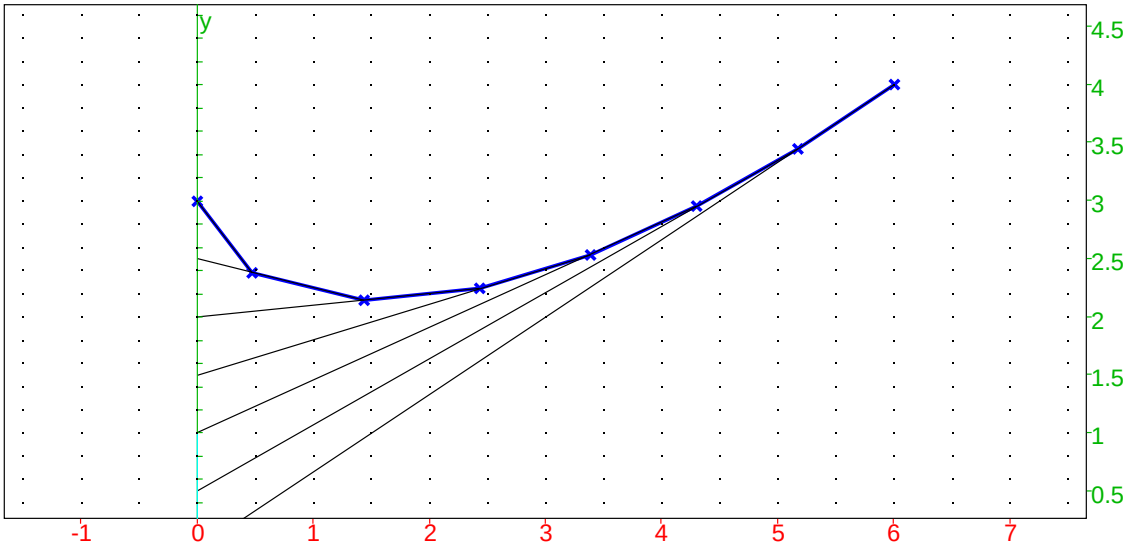
Il est alors pratique de travailler dans un espace affine.
La position suivante du renard, après un bond est $\mathrm{R} + \frac{br}{d}\overrightarrow{\mathrm{RL}}$
Celle du lapin est $\mathrm{L} + bl\vec{\jmath}$. Tout est en ordre pour pouvoir programmer...

```
poursuite1(xo,yo,br,bl):={
local R,L,d,pR,Dir,nbsaut;
R:=[xo,yo];
L:=[0,0];
pR:=NULL;
Dir:=NULL;
nbsaut:=0;
d:=evalf(sqrt(xo^2+yo^2));
 while((nbsaut<1000) and (d>0)){
 pR:=R,pR;
 Dir:=Dir,segment(point(R),point(L));
 d:=evalf(sqrt((L[0]-R[0])^2+(L[1]-R[1])^2));
  if(d>br)then{
   R:=R+(br/d)*(L-R);
   L:=L+[0,bl];
  }else{R:=L}
 nbsaut:=nbsaut+1;
 }
print(if(nbsaut<1000)
      then{"Lapin attrapé en "+nbsaut+" bonds"}
      else{"Le lapin court toujours après 1000 bonds"})
return(display(polygonscatterplot([pR]),blue+line_width_3+point_width_3),Dir)
}:;
```

Programme 11 – poursuite renard/lapin (1)

Par exemple, avec des bonds du renard deux fois plus longs que ceux du lapin :

```
poursuite1(6,4,1,0.5,7)
```



F Tirage de boules

On dispose de trois urnes, la première contenant 7 boules blanches et 4 noires, la deuxième 5 blanches et 2 noires, la troisième 6 blanches et 3 noires.
On tire une boule dans chaque urne et on note le nombre de boules blanches obtenues.
On adapte l'algorithme précédent :

```
boules(n):={
T:=NULL;
for(k:=1;k<=n;k:=k+1){
X:=0;
if(rand(11)<7){X:=X+1}
if(rand(7)<5){X:=X+1}
if(rand(9)<6){X:=X+1}
T:=T,X;
}
return(evalf([count_eq(0,[T])/n,count_eq(1,[T])/n,count_eq(2,[T])/n,count_eq(3,[T])/n])
)}:;
```

Programme 12 – exemple de tirage de boules

On obtient `[0.0358,0.2253818182,0.4453272727,0.2934909091]`
Or, par exemple, $\frac{7}{11}\times\frac{5}{7}\times\frac{6}{9}\approx 0,303$ et $\frac{4}{11}\times\frac{2}{7}\times\frac{3}{9}\approx 0,034$.

G Polygones de Sierpinski

Waclaw Sierpinski (1882 - 1969) fut un mathématicien polonais qui travailla sur des domaines mathématiques très ardus : fondements des mathématiques, construction axiomatique des ensembles, hypothèse du continu, topologie, théorie des nombres...
Il a également travaillé sur les premiers objets fractals qu'étudiera plus tard Benoît Mandelbrot, mathématicien français d'origine polonaise connu pour ses travaux dans ce domaine.

G 1 Jouons aux dés

Prenez un dé à 6 faces, un triangle ABC et un point G quelconque à l'intérieur du triangle.
Vous lancez le dé :

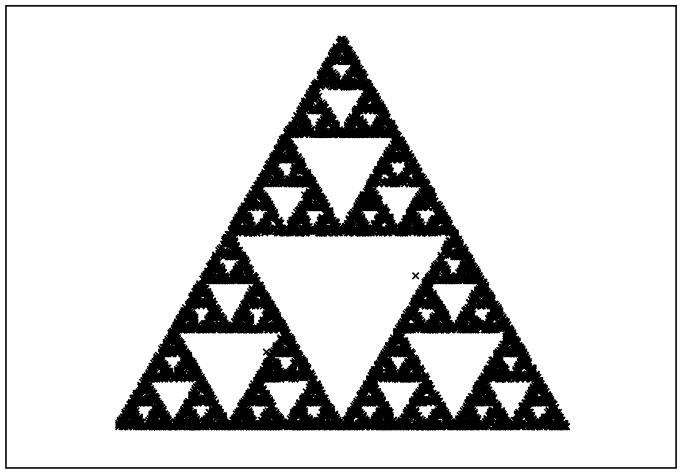
– si la face supérieure est 1 ou 2, vous faites une petite croix au niveau du milieu de G et de A ;
– si la face supérieure est 3 ou 4, vous faites une petite croix au niveau du milieu de G et de B ;
– si la face supérieure est 5 ou 6, vous faites une petite croix au niveau du milieu de G et de C ;

Ah, zut, on n'a pas de dé dans la salle d'info...mais on a XCAS.

– `hasard(0,1)` renvoie un nombre aléatoirement choisi entre 0 et 1 ;
– `hasard(n)` renvoie un entier aléatoirement choisi entre 0 inclus et n exclus.

```
dede(n):={
local P,r,t,d,G,L,k;
 P:=[[0,0],[1,0],[0.5,0.5*sqrt(3)]];
 t:=hasard(0,1);
 G:=t^2*P[0]+2*t*(1-t)*P[1]+(1-t)^2*P[2];
 L:=G;
  pour k de 1 jusque n faire
      G:=0.5*(G+P[hasard(3)]);
      L:=L,G;
  fpour;
 nuage_points([L]);
};;
```

ce qui donne en lançant `dede(10000)` :



2 - À vous de jouer

Exercice 1 Partie entière

Écrivez un programme dans le cas général (x de signe quelconque).

Exercice 2 Taux de remise variable

On entre un montant HT ht. On effectue une remise sur ht selon la régle suivante :
– si $ht < 2500$€ alors il n'y a pas de remise ;
– si $2500 \leqslant ht < 4000$€ alors la remise est de 5% ;
– dans les autres cas la remise est de 8%.
La TVA est de 19,6%. On demande le prix TTC en fonction du prix HT.
On peut construire une seule fonction ou une seule procédure ou couper en deux : le calcul du taux puis le calcul du prix. Essayez de construire une procédure avec affectation et une fonction sans affectation.

Exercice 3 Opérations sur les fractions

Quelle différence existe-t-il entre $\frac{7}{3}$ et $\frac{21}{9}$?
Quel rôle peut jouer le PGCD dans cette histoire ?
On va créer un programme simplifiant les fractions.
Au lieu d'écrire un nombre rationnel sous la forme $\dfrac{\text{numrateur}}{\text{dnominateur}}$, on va l'entrer dans la machine sous la forme `[numérateur,dénominateur]`.
Ainsi $\frac{21}{9}$ s'écrira `[21,9]`.
On peut alors créer un programme qui simplifiera une fraction donnée.
On utilisera une fonction donnant le pgcd de deux naturels :

```
let rec pgcd (a,b) =
        if b=0
        then a
        else pgcd(b,a mod b);;
```

Programme 13 – calcul du PGCD en récursif

Avec CAML, pour prévenir toute division par zéro on va créer un opérateur traitant les exceptions qu'on nommera `Division_par_zero` et qui nous servira de garde-fou :

```
# exception Division_par_zero;;
```

On invoquera cette exception avec la commande `raise` : étant donné votre niveau d'anglais, cette commande est naturelle !...
On peut créer un opérateur qui simplifie les fractions :

```
let simp([a;b]) =
 if b=0  then raise Division_par_zero
```

```
 else [a/pgcd(a,b);b/pgcd(a,b)];;
```

On crée ensuite une somme simplifiée de fractions :

```
let som([a;b],[c;d]) =
  if b=0 or d=0 then raise Division_par_zero else
  simp([a*d+b*c;b*d]);;
```

Par exemple :

```
# som([2;3],[1;6]);;
- : int list = [5; 6]
```

Bon : occupez-vous de la multiplication, du calcul de l'inverse et de la division.
Quand tout sera prêt, comment entrer

$$1+\frac{2+\frac{3}{4}}{1-\frac{5}{6}}$$

et

$$3+\frac{1}{2+\frac{1}{1+\frac{1}{2+\frac{1}{6}}}}$$

Exercice 4 Dichotomie

Avec XCAS, l'approximation d'un nombre formel est donnée par `evalf(nombre)`.
Par exemple :

```
evalf(pi)
```

Pour résoudre une équation du type $f(x)=0$, on recherche graphiquement un intervalle $[a,b]$ où la fonction semble changer de signe.
On note ensuite m le milieu du segment $[a,b]$. On évalue le signe de $f(m)$.
Si c'est le même que celui de $f(a)$ on remplace a par m et on recommence. Sinon, c'est b qu'on remplace et on recommence jusqu'à obtenir la précision voulue.
Donnez un programme récursif en 4 lignes qui prend en entrée la fonction f, a, b et la précision et qui renvoie l'approximation de la solution de $f(x)=0$ sur $[a;b]$.
En impératif, l'algorithme peutr être le suivant :

Entrées : une fonction f, les bornes a et b, une précision p
Initialisation : aa⟵ a, bb⟵ b
début
  tant que *bb-aa>p* **faire**
    si *signe de f((aa+bb)/2) est le même que celui de f(bb)* **alors**
      bb⟵ (aa+bb)/2
    sinon
      aa⟵ (aa+bb)/2
  retourner *(aa+bb)/2*
fin

Algorithme 2 : dichotomie

Exercice 5

En vous inspirant du cas général traitant la méthode d'Euler, imaginez une séance de TP qui construirait la courbe de la fonction dérivable sur $\mathbb{R}$ vérifiant $f'=f$ et $f(0)=1$.

Imaginez-en une autre pour le logarithme népérien.

Exercice 6 Méthode des trapèzes

Déterminez des programmes qui calculent une intégrale sur un segment par la méthode des trapèzes.

Exercice 7 Systèmes 2x2

Imaginez un TP en 2nde qui permettrait de construire un programme résolvant de manière exacte un système de deux équations linéaires à deux inconnues.

Exercice 8 Algorithme de Héron

Héron d'Alexandrie n'avait pas attendu Newton et le calcul différentiel pour trouver une méthode permettant de déterminer une approximation de la racine carrée d'un nombre positif.
Si x_n est une approximation strictement positive par défaut de $\sqrt{a}$, alors a/x_n est une approximation par excès de $\sqrt{a}$ (pourquoi ?) et vice-versa.
La moyenne arithmétique de ces deux approximations est $\frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$ et constitue une meilleure approximation que les deux précédentes.
On peut montrer c'est une approximation par excès (en développant $\left(x_n - \sqrt{a}\right)^2$ par exemple).
On obtient naturellement un algorithme :

Entrées : le réel positif a, une première approximation par défaut strictement positive xo et une précision eps
Initialisation : X⟵ 0.5(xo+a/xo)
début
  tant que |xo-a/xo|<eps **faire**
    X⟵ 0.5(X+a/X)
  retourner X
fin

Algorithme 3 : approximation de $\sqrt{a}$

On notera que cette méthode est un cas particulier de la méthode de Newton-Raphson appliquée à une équation particulière : $x^2 - a = 0$.
Traduire cet algorithme en impératif avec XCAS et en récursif avec CAML.

Exercice 9 Lapin malin

Reprenez le problème du renard et du lapin qui fait maintenant des bonds en zigue-zagant à angle droit.

Exercice 10 Tirages de boules avec remise

Simulons le tirage successif de quatre boules avec remise dans une urne contenant 7 boules blanches et 3 boules noires. Comptons le nombre de tirages contenant
– exactement deux boules blanches ;
– au moins une boule blanche.

Exercice 11 Fonction mystérieuse

Observez ces fonctions et décrivez leur action sachant que && signifie « et » et que || signifie « ou » :

```
# let rec truc(n)   = n<>1 && ( n=0 || machin(n-1) )
    and   machin(n) = n<>0 && ( n=1 || truc(n-1) ) ;;
```

Programme 14 – fonction mystérieuse (1)

Pour vous mettre sur la voie, voici trois fonctions plus compliquées mais aux noms plus explicites...

```
# let rec multiple_3(n) = n<>1 && n<>2 && ( n=0 || reste_3_2(n-1) )
  and     reste_3_2(n)  = n<>0 && n<>1 && ( n=2 || reste_3_1(n-1) )
  and     reste_3_1(n)  = n<>0 && n<>2 && ( n=1 || multiple_3(n-1) );;
```

Programme 15 – fonction mystérieuse (2)

Exercice 12 Algorithme glouton et rendu de monnaie

Une première idée pour optimiser une certaine situation est d'effecteur, étape par étape un choix optimum local dans l'espoir que le résultat global obtenu soit optimum : l'algorithme obtenu est appelé *algorithme glouton*.
Une illustration classique de ce problème est le rendu de monnaie qui est lui-même un cas particulier du problème du sac à dos. On dispose d'au moins deux types de pièces de valeurs premières entre elles (pourquoi ?...) et on veut rendre la monnaie en utilisant le moins de pièces possibles.
Une première idée est de répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante : c'est un algorithme glouton.

Entrées : somme n à rendre et liste L des pièces disponibles
Initialisation :
somme partielle⟵ 0
liste des pièces⟵ vide
début
  tant que *somme partielle* < *n* **faire**
    k⟵ taille de la liste
    tant que *k>0 et pièce n° k est supérieure à la somme restante* **faire**
      k⟵ k-1 : on regarde la pièce de valeur inférieure
  retourner *liste des pièces*
fin

Algorithme 4 : algorithme glouton de rendu de monnaie

Qu'est-ce que ça donne en XCAS ?
Ainsi, pour savoir comment former 39 centimes avec nos pièces européennes, on entrera :

```
piece_glouton(39,[1,2,5,10,20,50,100,200])
```

Commentez le résultat de :

```
piece_glouton(8,[1,4,5])
```

En fait, l'algorithme glouton est optimum si la suite des valeurs des pièces est *super-croissante* (on peut s'amuser à le démontrer...), c'est-à-dire si chaque terme est supérieur à la somme des précédents.
Par exemple, $20 > 10+5+2+1$ mais $5 \leqslant 4+1$ et la deuxième liste n'est pas super-croissante.

Exercice 13 Suite de Viète

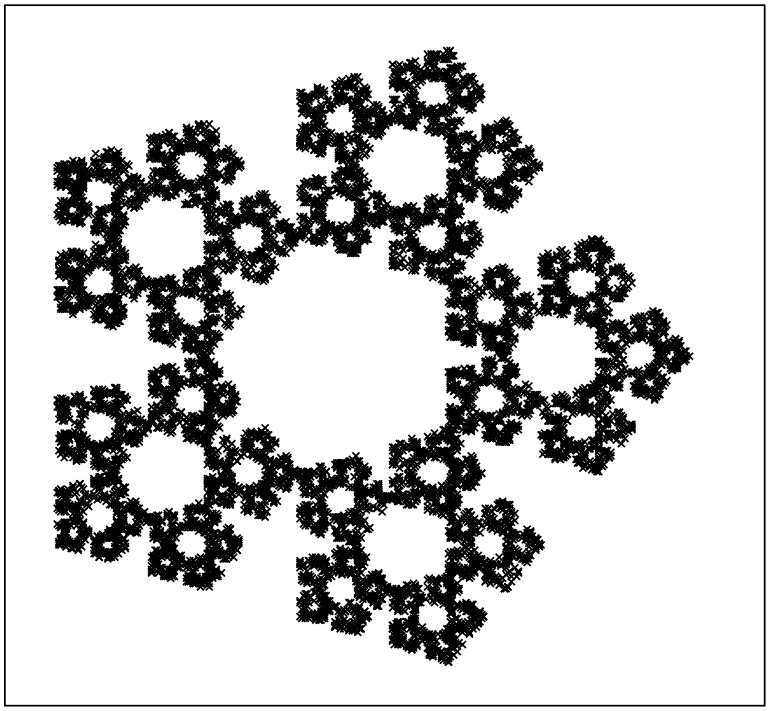
François Viète proposa au XVI^{e} siècle une approximation de π qui revient en langage moderne à étudier la suite :

$$u_0 = \frac{1}{\sqrt{2}} \qquad u_n = \sqrt{\frac{1}{2} + \frac{1}{2}u_{n-1}} \qquad \pi = \frac{2}{\prod_{n=0}^{+\infty} u_n}$$

Donnez des approximations de π à l'aide de cette méthode en programmant récursivement ou impérativement.

Exercice 14 Dentelle de Varsovie

Nous voudrions obtenir le joli napperon en dentelle suivant :



Déterminez le rapport des similitudes qui transforment le grand pentagone en un des pentagones plus petit. En vous inspirant de ce qui a été fait pour le triangle de Sierpinski, faites tracer à XCAS ce joli napperon.

Exercice 15 Végétation récursive

Analysez cet algorithme :

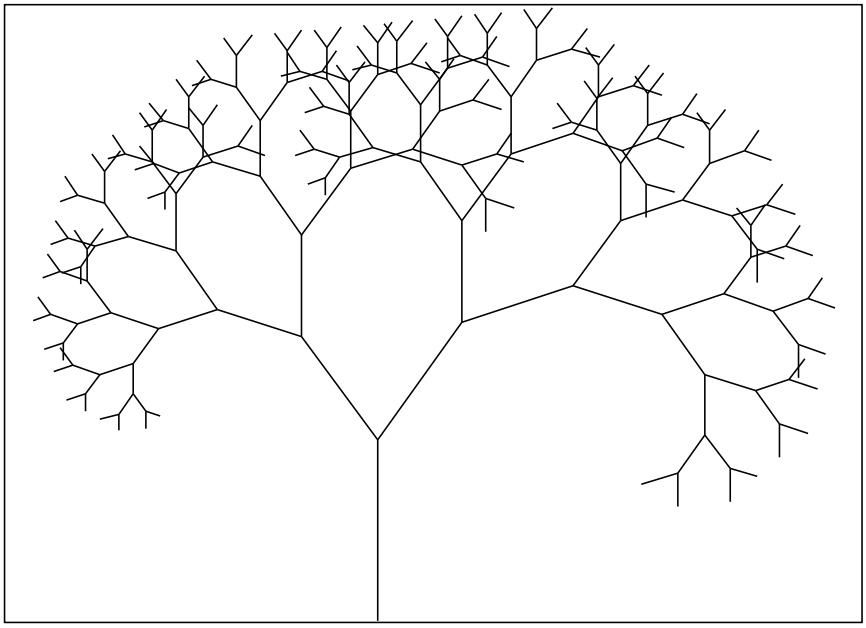
```
arbre(A,B,Rap,Ang,n):={
    si n>0 alors
      segment(point(A),point(B)),seq(arbre(B,B+Rap[q]*exp(Ang[q]*i)*(B-A),Rap,Ang,n-1),q=0..
          size(Rap)-1)
      sinon  segment(point(A),point(B))
    fsi
}:;
```

Programme 16 – L-System

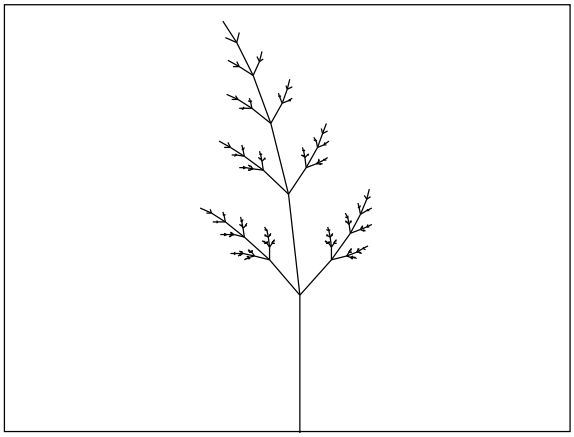
sachant que

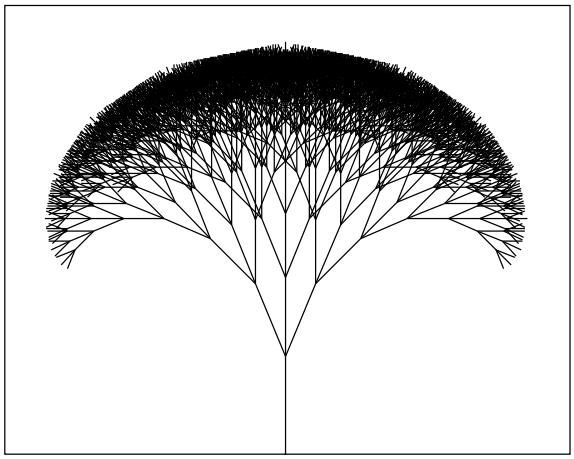
```
arbre(0,i,[0.7,0.8],[pi/5.,-pi/5.],7)
```

donne



Comment obtenir cette fougère et ce brocolis ?





Exercice 16 Étoiles et espaces

Que fait cette procédure :

```
pascal(n):={
local T,j,i;
 T:=[[0$n]$n];
 pour j de 0 jusque n-1 faire
   pour i de 0 jusque n-1 faire
     si j>i alors T[i,j]:="" ;
     fsi;
   fpour;
 fpour;
retourne(T);
}:;
```

Programme 17 – Tableau en triangle

Transformez-la un peu pour obtenir ceci avec $n = 10$ par exemple :

```
      1
      1    1
```

```
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
1    6    15   20   15   6    1
1    7    21   35   35   21   7    1
1    8    28   56   70   56   28   8    1
1    9    36   84   126  126  84   36   9    1
```

Complétez cette dernière procédure afin de remplacer chaque nombre pair par une espace ‘ ‘ et chaque nombre impair par une étoile ‘∗‘. Lancez-la pour $n = 48$: incroyable, non ?