CHAPITRE 5

LES TABLEAUX

Les tableaux dont il a déjà été question dans les chapitres précédents vont être naturellement utilisés pour tout le calcul matriciel, le programme des carrés magiques présenté comme premier exemple n'est pas difficile, celui de la triangulation par la méthode de Gauss est très classique, il est ici complété avec l'inversion des matrices carrées par la méthode de Gauss-Jordan et divers problèmes de calcul matriciel, on reverra enfin des problèmes de "backtracking".

5-1° Algorithme de Bachet de Meziriac

Dans cet exemple, on construit les éléments d'un tableau carré dans un ordre bien déterminé de façon à obtenir un carré magique. Un carré magique est un tableau de nombres tel que la somme des éléments de toutes les lignes, colonnes et des deux diagonales soit la même. L'algorithme présenté ici permet, pour un tableau carré de côté impair, de le remplir avec des entiers consécutifs.
1 est placé juste sous le milieu, les suivants se placent en descendant vers la droite, quand on arrive à un bord on va à l'extrémité opposée, et si une case est déjà remplie, le suivant est dans la même colonne mais 2 lignes en dessous.
Pour N=5, on peut facilement suivre 1, 2, puis le 3 devant se trouver à la sixième ligne est placé à la première ligne, le 4 qui devrait être à la sixième colonne est placé dans la première colonne, le 6 ayant déjà sa case prise est placé deux cases en dessous du 5, etc.

11	24	7	20	3
4	12	25	8	16
17	5	13	21	9
10	18	1	14	22
23	6	19	2	15

```
program carre_magique;
        type mat = array [1..19,1..19] of integer ;
{ Remarquer qu'une indexation de 0 à 18 rendrait possible l'utilisation de la fonction "mod" au lieu de la fonction F définie plus loin. }
        var N, D : integer ; A : mat;

procedure lecture (var N : integer ; var P : integer );
        begin write ('Carrés magiques de coté impair avec des entiers consécutifs positifs.');
        write ('Nombre de lignes et de colonnes: ');
        repeat readln (N) until odd (N); {seule une valeur impaire est acceptée}
        write ('Valeur de départ: '); readln (P) end ;

procedure ecriture (M : mat ; N, P : integer );
        { Affichage d'une matrice de N lignes et P colonnes }
        var I, J : integer ;
        begin for I := 1 to N do        begin for J := 1 to P do write (M[I, J] : 4 );
                                        writeln { passage à la ligne} end end ;
{ Il est tout à fait possible de se dispenser de cette procédure d'affichage, en plaçant à l'écran les valeurs au fur et à mesure de leur construction, grâce à l'instruction "gotoxy" (colonne , ligne) }
```

```
function f (X, N : integer ) : integer ;
{F est construite pour donner un résultat entre 1 et N dans tous les cas, on peut bien sûr se contenter de : "si X<N alors X sinon X-N ", mais indexer le tableau de 0 à n-1 permet d'utiliser "mod" et donc de se dispenser de F
        begin if X < 1   then F := F(X + N ,N)
                         else if X  > N    then F := F(X - N,N)
                                           else F := X
        end;

procedure construc (N, D : integer ; var A  : mat );      {D est la valeur initiale, N la dimension du carré}
        var I, J, X : integer ;                           {I , J sont les coordonnées de la case courante }
        begin for I := 1 to N do for J := 1 to N do A [I, J] := 0; {Initialisation nécessaire}
        I := (N div 2) + 1 ; J := I-1;
        for X := 0 to (N*N) -1 do
                begin if A[F(I+1, N), F (J+1, N)] = 0
                        then begin I := F (I+1, N); J := F (J+1, N) end
                        else repeat I := F (I+2, N) until A[I, J] = 0;
                        { Si la case suivante est vide, alors la case courante devient cette case-là.}
                A[I, J] := X + D
                end; { D est la valeur de départ }
        end;

begin
lecture (N, D); construc (N, D, A); ecriture (A, N, N); writeln ('La somme vaut : ', (N*(N*N-1) div 2) + N*D )
end.
```

5-2° Triangulation des matrices par la méthode de Gauss

L'algorithme de triangulation pour une matrice de N lignes et P colonnes (ici on a P > N et P = N+1 pour un système ayant autant d'équations que d'inconnues, voir le schéma ci-dessous) consiste pour chaque colonne $J \leq N-1$ à :

a) Chercher à partir de la diagonale pour $J \leq I \leq N$ la première ligne I où le coefficient M = A[I, J] est non nul (pivot).
ecrire une fonction "rgpivot" donnant ce numéro de ligne.

b) S'ils sont tous nuls, on passe à la colonne suivante.

c) Si I est différent de J, on échange les lignes I et J. (procédure "ech")

d) Diviser toute la ligne J par ce pivot (on a donc 1 sur la diagonale) à l'aide d'une procédure "unit" .

e) Remplacer chaque ligne au-dessous par une combinaison linéaire de cette ligne et de la ligne de référence J de façon à obtenir 0 dans la colonne J (procédure "comb").

```
program matrice;
        type mat = array [1..20,1..20] of real ;
        var A, B, C, X : mat ; N, P, Q, K, R : integer ; D : real ;
```

```
procedure lecture (var M : mat ; var N, P : integer );
{ Demande au clavier les éléments d'une matrice M de N lignes et P colonnes. }
        var I, J : integer ;
        begin write ('Nombre de lignes '); readln (N) ; write ('Nombre de colonnes '); readln (P);
        for I := 1 to N do
                begin   write ('Donnez les éléments de la ligne ', I, ': ');
                for J := 1 to P do begin read (M[I,J]) ; write (' ') end ; writeln end end ;

procedure ecriture (M : mat ; N, P : integer );
        { Edition à l'écran d'une matrice M de N lignes et P colonnes.}
        var I, J : integer ;
        begin for I := 1 to N do begin for J := 1 to P do write (M [I, J], ' '); writeln end end ;

procedure ech (var M : mat ; I, J, P : integer);
        { Réalise l'échange des lignes I et J dans la matrice M de P colonnes , M est donc modifiée }
        var  K : integer ; X : real;
        begin for K := 1 to P do begin X := M [I, K] ; M [I, K] := M [J, K] ; M [J, K] := X end end;

procedure unit (var A : mat ; L, P : integer ; M : real );
{A pour effet de diviser tous les termes de la ligne L sur la matrice A, par le même réel M }
        var  K : integer ;
        begin for K := 1 to P do A [ L,K ] := A [ L,K ] / M  end ;

procedure comb ( var  A : mat ; L, J, P : integer; M : real ); {A pour effet de remplacer dans la matrice A, la
ligne L, par cette ligne moins M fois la ligne J (combinaison linéaire ) }
        var K : integer ;
        begin for K := 1 to P do A[L,K] := A[L,K] - M*A[J,K]    end;

function rgpivot (A : mat ; J , N : integer ) : integer ; { Est la fonction qui donne le rang (n° de la ligne) où se
trouve le premier élément non nul de la colonne J à partir de la ligne J, dans la matrice A.  Par convention, ce
rang sera 0 au cas où tous sont nuls. }
        var I : integer ;
        begin  I := J;
        while ( (I < N+1) and (B[I , J] = 0) ) do I := I+1;
        if A[ I ,  J ] = 0 then rgpivot := 0 else rgpivot := I end ;

procedure trig (A : mat ; var B : mat; N,P : integer ; var R : integer ; var D : real);
{Réalise la triangulation B de la matrice A de N lignes et P colonnes, et calcule le rang R et le déterminant D du
carré N*N . Trig sera utilisé avec P = N+1 ici.}
        var I, J, K : integer ; M : real ;
        begin D := 1;              { D sera le déterminant }
        R := min (N, P);           { rang valable pour les matrices carrées seulement ! }
        for I := 1 to N do for J := 1 to P do B[I, J] := A[I, J]; { B est une copie de A au départ }
        for J := 1 to N-1 do
        begin I := rgpivot (B, J, N );
                if    I = 0      then     begin D := 0 ; R := R-1 end
                                 else     begin if  I <> J then begin ech (B, I ,J ,P); D:= -D end ;
                                          unit (B, J, P, B[J, J] ) ; D := D*B[ J, J ];
                                          for I := J+1 to N do comb (B, I, J, P, B[ I, J ]);
                                          end
        end;
        D := D*B[ N , N]
        end;
```

5-3° Résolution d'un système linéaire de N équations et N inconnues, la méthode consiste, après triangulation, à calculer les inconnues en remontant de la dernière à la première. En ce qui concerne l'inversion d'une matrice N*N, elle s'obtient ici en résolvant N systèmes: en effet si E_i désigne le vecteur colonne nul sauf 1 à la i-ième ligne, la i-ième colonne C_i de A^{-1} est le produit $A^{-1}E_i$, il faut donc la trouver en résolvant le système $AC_i = E_i$, c'est ce que l'on fait pour $1 \le i \le N$. Terminer le programme précédent par la multiplication des matrices (afin de vérifier l'inversion) et un menu.

```
procedure reso (A : mat; var X : mat ; N : integer );  { X est la colonne des inconnues construite à partir d'une
matrice A déjà triangulaire de N lignes et N+1 colonnes. }
        var  I, K : integer ;
        begin  X [N, 1] := A [N, N+1] / A[N, N];
        for I:=N-1 downto 1 do begin X[I, 1] := A[I, N+1] ;
                    for K := I+1 to N do X[I, 1] := X[I, 1] - A[I, K]*X[K, 1] end end ;

procedure concat (A, B : mat ; var C : mat ; N, P, Q : integer );
{Construit une matrice C de N lignes et P+Q colonnes en "concaténant" celles de A et de B. }
        var  I, J : integer ;
        begin for  I := 1 to N  do      begin for J := 1 to P do C[I, J] := A[I, J];
                                        for J := 1 to Q do C[I, P+J] := B[I, J] end end ;

procedure jordan (var  A : mat; var  IA : mat; N : integer ; var  R : integer );
{ A est la matrice donnée au départ, IA son inverse, N sa dimension, à A on ajoute tour à tour une dernière
colonne : les colonnes successives de la matrice identité, cette nouvelle matrice est triangulée en B, et le système
ainsi formé admet la solution E; IA est constitué de la concaténation des colonnes E successives. }
        var  I : integer ; D : real ; B, E : mat ;
        begin  A[1, N+1] := 1; for I := 2 to N  do A[I, N+1] := 0; trig (A, B, N, N+1, R, D);
        if   D = 0      then writeln ('Déterminant nul,  matrice non inversible.')
                        else    begin writeln ('Dét = ', D);
                                reso (B, IA, N) ; A[1, N+1] := 0;
                                for I := 2 to N do
                                        begin A[I, N+1] := 1;
                                        trig (A, B, N, N+1, R, D);
                                        reso (B, E, N);
                                        concat (IA, E, IA, N, I-1, 1);
                                        A[I, N+1] := 0 end;
                                ecriture (IA, N, N);
                                end;
        end;

procedure mult (A, B : mat ; N, P, K : integer ; var C : mat); { Calcule le produit matriciel C de A par B }
        var I, J, Q : integer ;
        begin for I := 1 to N do for J := 1 to K do
                begin C[I, J] = 0; for Q := 1 TO P do C[I, J] := C[I, J] + A[I, Q]*B[Q, J] end end;

begin { Début du programme. } writeln ('1: Multiplication de deux matrices ');
writeln ('2: Déterminant et inversion'); writeln ('3: Résolution de système '); readln (N);
case N of       1       : begin writeln ('Matrice de gauche'); lecture (A, N, P); writeln ('Matrice de droite');
                        lecture (B, Q, K);
                        if P <> Q       then    write ('Impossible')
                                        else    begin mult (A, B, N, P, K, C); ecriture (C, N, K) end end ;
                2       : begin lecture (A, N, P);
                        if N <> P       then writeln ('un carré !')
                                        else begin jordan (A, B, N, R); writeln ('Rang =', R) end end ;
                3       : begin lecture (A, N, P);
                        if  P  =  N+1   then    begin trig (A, B, N, P, R, D);
                                                if  D  <>  0    then    begin  reso (B, X, N);
                                                                        ecriture (X, N, 1);
                                                                        writeln ('Dét = ', D)  end
                                                                else write ('Dét. nul  Rang =', R) end
                                        else write ('Non de Cramer') end end {du "case"}
end.
```

Exemple à tester : 1 2 3 4 5 6 7 sont les solutions de :

$$\begin{array}{rrrrrrrcr}
-2x & +3y & -z & +4t & -u & -v & +5w & = & 41 \\
3x & & -z & +5t & -3u & +v & -7w & = & -38 \\
4x & -y & +2z & & -u & +3v & +w & = & 28 \\
x & -y & +3z & -t & +u & +v & +w & = & 22 \\
x & -2y & +5z & -7t & +2u & -3v & -w & = & -31 \\
-x & -y & -z & & +3u & & +w & = & 16 \\
-7x & -5y & +z & -3t & +4u & -v & +2w & = & 2
\end{array}$$

5-4° Placer dans un tableau les éléments successifs d'un développement en fraction continue et renvoyer la valeur.(exemple 1 2 2 2 2 ... pour √3, 2 1 2 1 1 4 1 1 6 1 1 8 ... est e, et 1 1 1 2 1 2 1 2... est (1+√5)/2 le nombre d'or).

Exemple pour la suite a_1, b_1, a_2, b_2, a_3, b_3,....

$$F = a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{a_3 + \cfrac{b_3}{a_4 + \cfrac{b_4}{a_5 + \ldots}}}}$$

```
program frac;
        var A : array [1..20] of real; n, k : integer; f : real;
        begin writeln ('Donnez les éléments d'un dév. en fraction continue ');
        write ('donnez en le nombre de couples '); readln (n);
        for k := 1 to n do read (A[k], B[k]);
        f := B[n] / A[n];
        for k := n-1 downto 1 do f := A[k] + B[k]/f;
        write ('Le résultat est ', f)
        end.
```

5-5° Modifier la procédure "reso" en introduisant un nouveau type de données : les colonnes, qui seront des tableaux à une dimension.

5-6° Ecrire une procédure "cof" construisant une sous-matrice obtenue à partir d'une matrice, en lui retirant une ligne et une colonne. Utiliser cette procédure pour construire une fonction récursive à deux paramètres : une matrice A et un rang N, calculant le déterminant de A en développant par rapport à la première ligne. On rappelle que det(A) = S $(-1)^{1+k}$ det($B_{1,k}$) pour k allant de 1 à N, étant la sous-matrice de A obtenue en retirant la première ligne et la k-ième colonne.

5-7° Quelle est l'erreur dans for K := 1 to P do A[L, K] := A[L, K] / A[L, J] que l'on pourrait mettre dans la procédure "trig" afin de supprimer "unit" ?

5-8° Modifier la fonction "rgpivot" afin qu'elle donne le numéro de la ligne portant le plus grand nombre en valeur absolue. L'intérêt de ce pivot est de minimiser les erreurs dans les divisions ultérieures.

5-9° Problème du professeur A. Roze : produire le triangle de Pascal à l'écran en se servant d'une matrice-ligne uniquement. Utiliser la symétrie du triangle et la relation :

C(n, p) = C(n-1, p-1) + C(n-1, p)

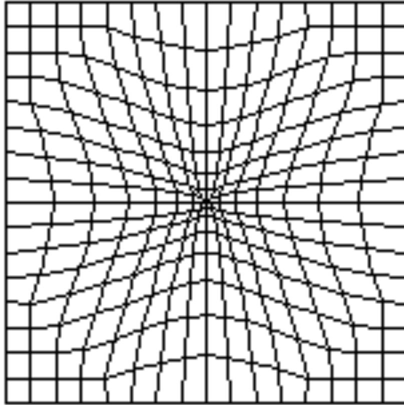
On transforme le tableau de droite à gauche en recalculant ses éléments à l'envers, et on l'affiche dans le même temps :

```
program triangle;
var p, n : integer; C : array[0..15] of integer; {on ne fait que 16 lignes}
begin C[0] := 1; writeln (C[0]:5);
for n := 1 to 15 do     begin C[n] := 1; write (C[n]:5);
                        for p := n-1 downto 1 do     begin C[p] := C[p] + C[p-1];
                                                     write (C[p]:5)
                                                     end;
                        writeln (C[0]:5) end
end.
```

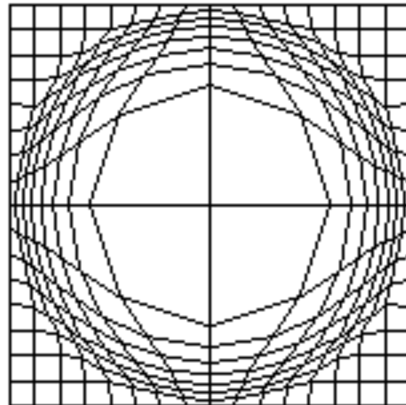
5-10° Quadrillages déformés [J.P.Delahaye 85]
On considère un quadrillage 20*20 par exemple, 441 points. Pour chacun de ces points :
a) On se ramène aux coordonnées cartésiennes dans un repère où le carré sera inscrit dans le carré [-1, 1]*[-1, 1]

b) On passe en coordonnées polaires ρ, θ

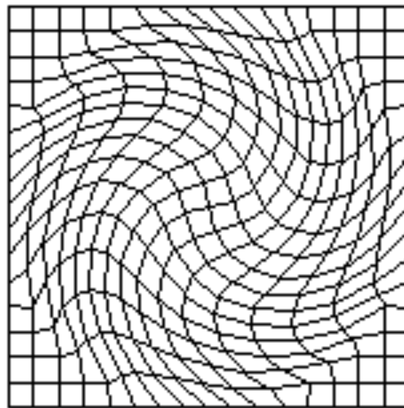
c) On exerce une transformation telle que $\rho \leftarrow \rho^2$ localisée dans le cercle unité comme ci-dessous.
d) On reconvertit en coordonnées cartésiennes, puis dans le repère de l'écran.
e) On joint chacun de ces points en lignes et en colonnes pour bien visualiser un maillage.



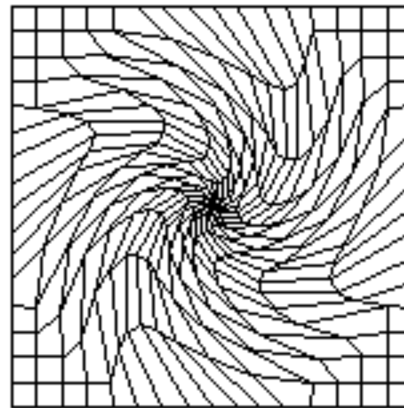
$\rho \leftarrow \rho^2$



$\rho \leftarrow \rho^{0,25}$



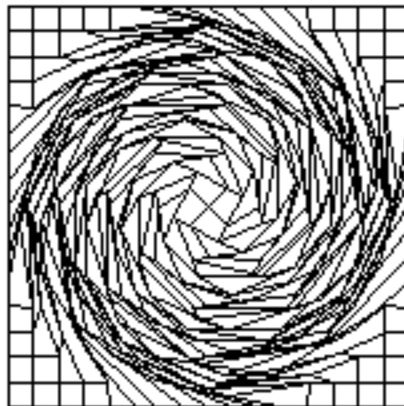
$\theta \leftarrow \theta + \pi\ (1 - \rho) / 2$



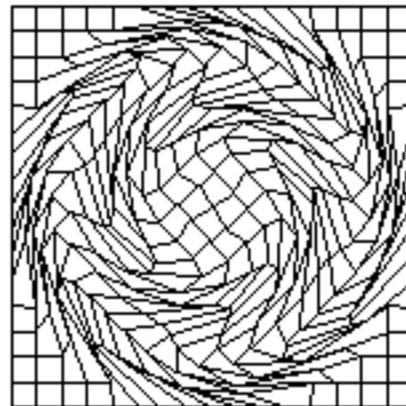
$\theta \leftarrow \theta + 3,5\ (1 - \rho)$ et $\rho \leftarrow \rho^3$

```
program tole ; { Version Macintosh }
        uses memtypes, quickdraw;
        var A, B, N, L, H, CH : integer; P, Q : array [0..20,0..20] of real;
{tableaux renfermant les modules et arguments des 441 points}

procedure rtgle (A, B : integer; var U, V : integer); {calcule U, V à partir des coord. polaires}
        begin U := round (P[A, B]*cos (Q[A, B])*H/2+L/2);
        V := round (P[A, B]* sin (Q[A, B])*H/2+H/2+50)
        end;

procedure ecran; {éxecute le dessin}
        var A, B, U, V : integer;
        begin for B := 0 to N do begin rtgle (0, B, U, V); moveto (U, V);
                        for A := 1 to N do begin rtgle (A, B, U, V); lineto (U, V) end end;
        for A := 0 to N do begin rtgle (A, 0, U, V); moveto (U, V);
                        for B := 1 to N do begin rtgle (A, B, U, V); lineto (U, V) end end
        end;
```

```
procedure polaire (X, Y : real; var M, G: real);
        begin M := sqrt ((X*X)+(Y*Y));
        if X = 0        then if Y>0     then G := PI/2
                                        else if Y<0 then G := 1.5*PI else G := 0
                        else begin G := arctan (Y/X); if X<0 then G := G+PI; end end;

procedure init; {remplit les tableaux avec les coord. polaires des noeuds du quadrillage}
        var X, Y : real;
        begin for B := 0 to N do for A := 0 to N do
                begin X := 2*(A - N / 2) / N; Y := 2*(B-N/2)/N; polaire (X, Y, P[A,B], Q[A,B])
        end end;

begin {programme} L := 250; H := 150; N := 16; {largeur, et hauteur d'écran, dim. du quadrillage}
clearscreen; write ('Initialisation '); init; clearscreen; write ('Votre choix ? (1 à 8) '); readln (CH);
for A := 0 to N do for B := 0 to N do begin if P[A, B]<1 then begin
case CH of      1: Q[A, B] := Q[A, B] + (PI*(1-P[A, B]))/2;
        2: begin Q[A, B] := Q[A,B] + 3.5*(1-P[A,B]); P[A, B] := P[A, B]*P[A, B]*P[A, B] end;
        3: Q[A, B] := Q[A, B] +2*PI* (1-P[A, B]);
        4: Q[A, B] := Q[A, B] +PI* sin (2*PI*(1-P[A, B]))/4;
        5: begin Q[A, B] := Q[A,B]+PI* sin(PI*(1-P[A, B]))/2; P[A, B] := P[A, B]*P[A, B] end;
        6: P[A, B] := sqrt (sqrt (P[A, B]));
        7: P[A, B] := P[A, B]*P[A, B];
        8: begin Q[A, B] := Q[A, B]+PI* sin (PI*(1-P[A, B]))/2; P[A, B] := sqrt (P[A, B]) end
 end end end;
clearscreen; ecran end.
```

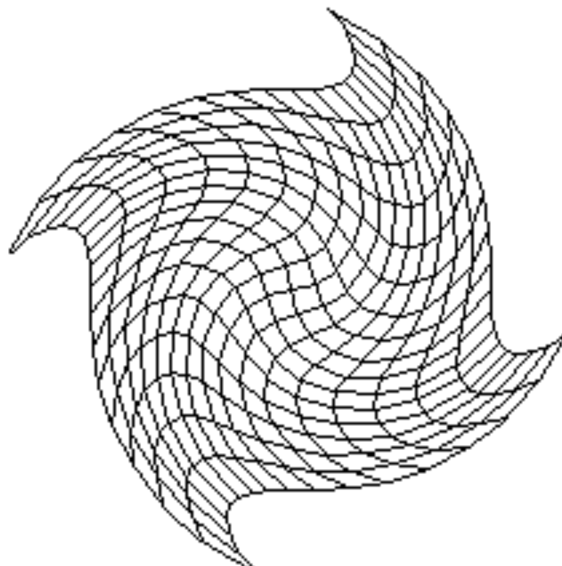


$\theta \leftarrow \theta + 2\pi(1-\rho)$



$\theta \leftarrow \theta + \pi \sin(2p(1-\rho))/4$

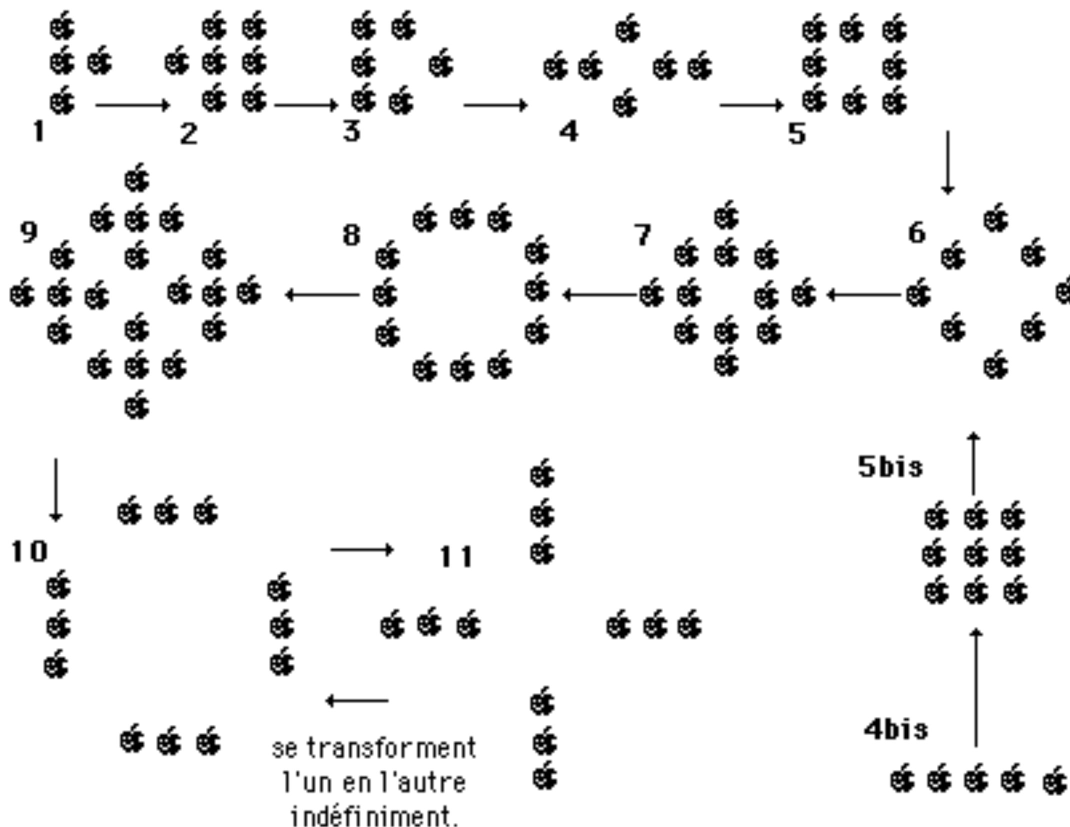
En supprimant la condition $\rho < 1$, l'effet n'est pas mal non plus :



5-11° Le jeu de la vie (Conway). Sur un quadrillage 20 lignes et 80 colonnes par exemple, que l'on considère "toroïdal" (la 21° ligne est la première et la 81° colonne est la première), chaque case est soit vivante, soit morte. Etant donné un état du tableau (un état initial sera un petit motif donné par le joueur), on a les règles suivantes pour chaque case :
Elle est vivante et elle touche au plus une voisine vivante, alors à l'étape suivante, elle meurt par isolement.
Elle est vivante et possède plus de trois voisines vivantes, alors elle meurt d'étouffement.
Dans les autres cas où elle est vivante, elle reste vivante.
Une case morte qui possède exactement trois voisines vivantes, provoque une naissance.
On peut former deux tableaux ou bien un tableau à trois dimensions où le troisième indice appartient à un type défini par (ancien , nouveau). Les deux autres indices peuvent être entiers 0..19 et 0..79 par exemple. Rappelons que les différentes générations ne sont pas mélangées. Certains motifs disparaissent, d'autres se déplacent en restant identiques, d'autres sont constants, mais la plupart sont périodiques. Ainsi un motif de 7 enzymes alignés consécutifs devient périodique en 17 étapes, pour 8, il devient constant au bout de 48 étapes, et pour 9, il devient de période 2 après 21 étapes.

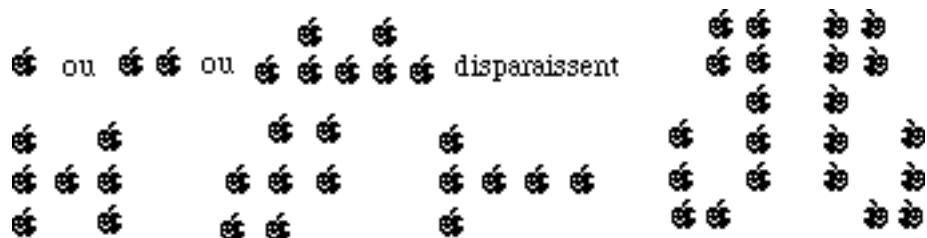
Exemples de motifs constants, sauf le quatrième qui meurt en huit générations.:

Exemples de transformations (feux de navigation) :




```
program enzymesgloutons;
var A : array[1..20,1..20,1..2] of integer; R:char; ng :integer;
        {A est une variable globale qui représente une matrice à trois dimensions}
```

```
function nbvoisins (L, C, I : integer) : integer; {donne le nombre de voisins vivants de la case L,C,I}
          function tore (X : integer) : integer; {fonction locale à nbvoisins}
          begin case X of   0   : tore := 20;
                            21 : tore := 1;
                    otherwise tore := X end end; {tore}
          begin nbvoisins := A[tore(L-1), tore(C-1), I] + A[tore(L-1), tore(C), I] + A[tore(L-1), tore(C+1), I]
                           +A[tore(L), tore(C-1), I] + A[tore(L), tore(C + 1), I] + A[tore(L+ 1), tore(C-1), I]
                           + A[tore(L + 1), tore(C), I] + A[tore(L+1), tore(C + 1), I] end;
procedure entree;          {initialise le tableau A pour I=1 avec la population initiale}
          var L, C, I, N:integer;
          begin N := 1; ng := 0; {ng est une variable globale : le numéro de génération}
          for I := 1 to 2 do for L := 1 to 20 do for C := 1 to 20 do A[L, C, 1]:= 0;
          writeln ('donnez les coordonnées des points (0 pour finir)');
          repeat write ('enzyme numero ',N ,' ligne '); read (L);
                    write ('              colonne '); readln (C); A[L, C, 1] := 1 ; N := N+1
          until (L = 0) or (C = 0) end;
procedure affiche (I : integer);      {affiche le tableau à l'écran, le premier à gauche, le second à droite sur l'écran
les lignes 2 à 21 et col. 10 à 29 pour I = 1 ou 50 à 69 pour I=2}
          var L,C, n :integer;
          begin n := 0; for L := 1 to 20 do for C := 1 to 20 do
                    begin gotoxy (C - 31 + 40*I, L+1); n := n + A[L, C, I]; write (chr(32+3*A[L, C, I])) end;
                    {si la case est 0 on affiche un blanc de code 32, sinon # de code 35}
                    gotoxy (40*I-30, 24); write ('Nouvelle population ', n);
                    gotoxy (90-40*I, 24); write ('Ancienne population ') end;
procedure cadres; {construit deux cadres afin que l'on puisse analyser chaque passage d'un état à son suivant}
          var X : integer;
          begin clearscreen;
          for X := 1 to 22 do begin gotoxy (8+X, 22); write ('_'); gotoxy (48+X,22); write ('_');
                           gotoxy (9, X); write ('|'); gotoxy (30, X); write ('|'); gotoxy(49, X); write ('|');
                           gotoxy (70, X); write('|'); gotoxy (8+X, 1); write('_'); gotoxy (48+X,1); write ('_') end
          end;
procedure transfo (I : integer); {I = 1 ou 2 suivant la trabche à transformer}
{va calculer l'autre tranche (3-I) du volume A à partir de la tranche I}
          var L, C, N : integer;
          begin for L := 1 to 20 do for C := 1 to 20 do
                    begin N := nbvoisins (L, C, I);
                    if A[L, C, I] = 1 then if (N = 2) or (N = 3)   then A[L, C, 3-I] := 1
                                                                   else A[L, C, 3-I] := 0
                                      else if N=3          then A[L, C, 3-I] := 1
                                                           else A[L, C, 3-I] := 0
                    end
          end;
procedure jeu (I : integer); {est la procedure essentielle, elle demande si on veut poursuivre, et si oui, se
rappelle elle-même.}
          var R : char;
          begin  affiche (I); transfo(I) ; gotoxy (26,25); write ('Génération ', ng, ' suite (O/N) '); ng := ng + 1;
          R := readchar ; if (R = 'O') or (R = 'o') then jeu (3 - I) end;

begin entree; cadres ; jeu (1) end. {constitue le programme }
```



Le dernier motif se répète au bout de 14 générations. Le motif en "T couché" se multiplie énormément puis amorce une décroissance vers la 80° étape, et enfin devient constant à la 114° étape en étant formé de 4 carrés.

Les deux premiers sont des "planeurs" qui sont de période 4 en se déplaçant en biais, le chat arrive en 7 étapes à un carré stable de 4.

5-12° Le corail de S.Ulam. [Heudin 94] Sur une grille 21*21, les cellules peuvent être mortes ou vivantes (jeunes ou vieilles). Les voisines d'une cellule ne sont que les 4 cellules au dessus, en dessous, à gauche et à droite. A chaque génération une cellule morte ayant une seule voisine vivante prend vie (elle est jeune), sinon elle reste morte, une cellule jeune devient vieille, une vieille devient morte (la vie ne dure que deux générations). On peut partir d'une seule cellule vivante ou d'un motif plus complexe. Trouver une structure apte à la programmation (matrice de booléens à 3 couches ou bien matrice ordinaire d'entiers).

5-13° Karel le robot doit trouver la sortie d'un labyrinthe (un tableau carré de booléens) en longeant toujours le mur gauche (ce n'est pas du tout le même problème que celui du labyrinthe, il n'y a pas de backtrack ici). On écrira une procedure
"bonne direction" (x, y ; var a)
quart de tour gauche; tant que mur face à x, y, a faire quart de tour droite
et une procedure "générale" :
quart de tour gauche;
tant que non sorti faire bonne direction (x, y, a) puis avancer d'un pas

5-14° Jeu du Tic Tac Toe Sur un carré 3*3, l'utilisateur doit jouer contre l'ordinateur. Chacun à tour de rôle, met sa marque sur une case libre. Le premier ayant réussi un alignement de 3 cases a gagné. Trouver une représentation des données adaptée à ce jeu, et programmer sans stratégie particulière.

Correction sommaire : On utilise les cases des touches du clavier, soit 7 8 9 en haut, 4 5 6 au milieu et 1 2 3 pour la ligne du bas. Le tableau t[1..9] va contenir 0 (libre) ou 1 (pour le joueur) ou 5 (l'ordinateur), et un tableau p[1..24] toujours constant avec les éléments [1 2 3 4 5 6 7 8 9 1 4 7 2 5 8 3 6 9 1 5 9 3 5 7].
Procedure joue (var gagne, perdu, nul : boolean; var c : integer);
{gagne pour l'ordinateur, perdu si c'est le joueur qui gagne, et nul : on arrête}
{c est la case à jouer par l'ordinateur}
Sur les 8 cas du tableau p :
On trouve une somme 3 dans t, alors fini le joueur a gagné,
si une somme 10 alors l'ordinateur gagne avec la case c,
si une somme 12, l'ordinateur joue sur la case libre (il est sur la défensive)
Si toutes les cases sont pleines et toutes les sommes > 7 alors match nul
Sinon l'ordinateur joue la première case libre.
Programme jeu
t est initialisé à 0
Voulez-vous commencer ? si oui case départ ? i
t[i] := 1; gagné, perdu, nul := false
tant que non (gagné ou perdu ou nul) faire
c := 0
Joue (gagne, perdu, nul, c);
Si c ≠ 0 alors t[c] := 5 puis affichage;
Jouez! ; read(i); t[i] := 1
affichage
si gagné ..., perdu ..., nul ...

5-15° Méthode de Bairstow pour la résolution des équations algébriques. On divise le polynôme par un trinôme du second degré et on cherche en modifiant ce diviseur par petites retouches, à annuler le reste. Lorsque x^2-sx+p divise le polynôme, on peut donc en donner deux racines, et reprendre la méthode pour le quotient. La première étape est donc :
Calcul du quotient : pour un polynôme A de degré n, on pose :

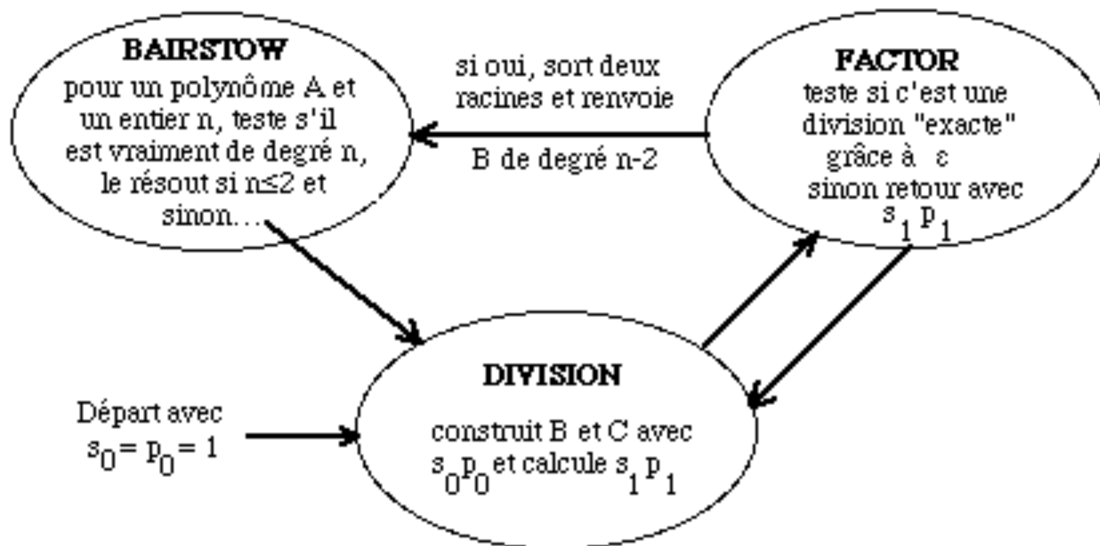
$A(x) = a_0x^n+a_1x^{n-1}+a_2x^{n-2}+....+a_0$

$= (x^2-sx+p)(b_0x^{n-2}+...+b_{n-2})+ b_{n-1}(x-s)+b_n$

ce qui conduit à $b_0 = a_0$ $b_1 = a_1+ sb_0$ $b_i = a_i + sb_{i-1}-pb_{i-2}$ si on pose $b_{n-1} = f(s, p)$ et $b_n = g(s, p)$, on souhaite donc résoudre le système f(s, p) = 0 g(s, p) = 0, on utilise pour cela la formule de Taylor à l'ordre 1 pour deux variables, qui peut s'écrire :
$f(s+h, p+k) = f(s, p) + hf'_x(s, p) + kf'_y(s, p)$; le système permet alors de déduire
$h = (fg'_y- gf'_y) / D$ et $k = (gf'_x - fg'_x) / D$ où $D = f'_yg'_x - f'_xg'_y$
Calcul des dérivées partielles en s et p, par récurrence :
En notant $c_i = db_i / ds$ et $d_i = db_i / dp = - c_{i-1}$ on a : $c_0 = 0$, $c_1 = b_0 + sc_0$ et $c_i = b_{i-1} + sc_{i-1} - pc_{i-2}$ donc partant de s_0 et p_0 il est possible de calculer les b_i et les c_i
puis $D= c^2_{n-1} - c_nc_{n-2}$ et enfin les candidats suivants :
$s_1 = s_0 + (b_nc_{n-2} - b_{n-1}c_{n-1}) / D$ et $p_1 = p_0 + (b_nc_{n-1} - b_{n-1}c_n) / D$ que l'on teste à nouveau.

Solution : cet exemple assez long, montre qu'une bonne division du travail permet de s'y retrouver. Il utilise le type de données le plus commode, à savoir les tableaux, pour représenter les polynômes. On testera l'arrêt sur $(|h|+|k|) / (|s|+|p|) < \varepsilon$ où par exemple ici $\varepsilon = 10^{-6}$

Schéma général du programme



Rappels, "sqr"(x) est le carré de x, et "sqrt"(x) en est la racine carrée. Le Pascal impose un ordre impératif dans les déclarations de constantes, types, variables, procédures et fonctions (le Turbo-Pascal, imposant d'ailleurs, moins de contraintes que les autres versions).
Pour ces dernières, elles ne peuvent qu'en appeler d'autres définies antérieurement. En cas d'appels mutuels, comme ici, on prévient l'analyseur par le mot réservé "forward".

```
program equation ; { Ce programme résout les équations algébriques }
        type pol = array [0..20] of real ; { Un polynôme est une suite de 21 réels }
        var A : pol ; N : integer ;
{ A sera le polynôme de degré N, ce sont les seules variables globales. }
```

```
procedure lecture  (var A : pol; Q, N : integer);
{ Q est N moins le degré du terme . Cette procédure est récursive et elle écrit au fur et à mesure le polynôme,
tout en demandant les coefficients. }
        begin
        if  Q < N       then begin read (A[Q]); write ('X', N - Q,'+'); lecture (A, Q+1, N) end
                        else begin read (A[N]); writeln ('=0')  end
        end;

procedure racine (S, P : real) ; { Cette procédure résout l'équation x2 - Sx + P = 0 en affichant les résultats, elle
ne renvoie rien et ne modifie aucune variable.}
        var D, X : real;
        begin D :=  S*S - 4*P;
        if   D = 0      then writeln (S/2,' (double)')
                        else if D > 0      then    begin X := (S + sqrt (D)) /2;
                                                   writeln (X); writeln (S-X) end
                                           else writeln (S/2, '(+ -)i', sqrt (-D) / 2, ' complexes conjugués')
        end;

procedure factor  (A, B, C : pol ; N : integer ; S0, P0, S1, P1, EPS : real);
        forward ; { déclaration nécessaire lors de récursivité croisée.}

procedure division (A : pol ; var B, C : pol ; N : integer ; S0, P0 : real);
        var D : real ; Q : integer ;
        begin B[0] := A[0]; B[1] := A[1] + S0*A[0]; C[0] := 0; C[1] := B[0];
        for Q :=  2 to N  do      begin B[Q] := A[Q] + S0*B[Q-1] - P0*B[Q-2] ;
                                  C[Q] := B[Q-1] + S0*C[Q-1] - P0*C[Q-2]
                                  end ; { On construit itérativement B et C }
        D := sqr (C[N-1]) - C[N]*C[N-2] ; if D = 0 then D := 0.1;
        { On produit S1 P1 pour l'envoyer à factor: }
        factor (A, B, C, N, S0, P0, S0 + (B[N]*C[N-2] - B[N-1]*C[N-1]) / D,
                                    P0+ (B[N]*C[N-1]- B[N-1]*C[N]) / D, 0.000001)
        end ;

procedure bairstow (var A : pol ; N : integer) ; forward ;

procedure factor ;
{la liste des paramètres (A, B, C : pol ; N : integer ; S0, P0, S1, P1, EPS : real) a déjà été donnée}
        begin if (abs (S0-S1) + abs (P0-P1)) / (abs (S0) + abs (P0)) > EPS
            then division (A, B, C, N, S1, P1)
            else begin racine (S1, P1) ; bairstow (B, N - 2) end ;
        end ;

procedure bairstow ; { (var A : pol; N : integer ) déjà dit }
        var B,C : pol ; Q : integer ;
        begin while A[0] = 0 do begin for Q := 0 to N - 1 do A[Q] := A[Q+1] ; N := N - 1 end ;
        case   N  of      1        : writeln (-A[1] / A[0]);
                          2        : racine ( -A[1] / A[0], A[2] / A[0]);
                          else division (A, B, C, N, 1, 1) end ; {else sur PC, otherwise sur MAC}
        end ;

begin           { début du programme }
writeln ('Résolution d ''équations algébriques de degré < 20 '); {la double apostrophe signale à la procédure
"write" que le message n'est pas fini, et qu'il faut afficher une apostrophe}
write ( 'Degré du polynome ? ' ) ; readln (N);
writeln ('Donnez les coef. du polynome dans l''ordre décroissant.'); lecture (A, 0, N); bairstow (A, N)
end.
```

5-16° Trouver des exemples pour vérifier le programme de résolutions d'équations

-5 -2 1 3 4 6 sont les racines de $x^6 -7x^5 -21x^4 +204x^3 -140x^2 -756x +720 = 0$

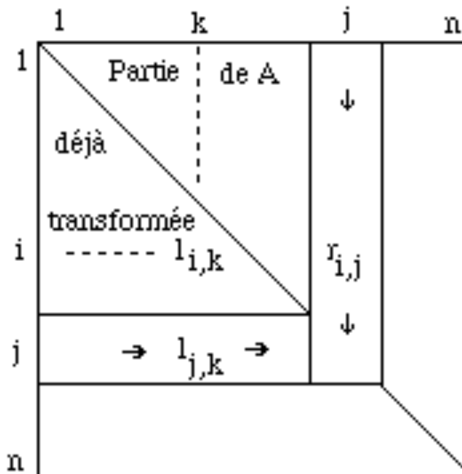
-2, 1 et 3 doubles, sont les racines de $x^6 -4x^5 -6x^4 +32x^3 +x^2 -60x +36 = 0$

$2x^7 - 13x^6 - 49x^5 + 385x^4 - 77x^3 -1652x^2 + 684x + 720 = 0$ admet -5 -2 -1/2 1 3 4 6

5-17° Décomposition LR par l'algorithme de Crout Etant donné une matrice carrée A de dimension n*n, il est possible sous certaines conditions d'obtenir A = L*R où L est une matrice triangulaire inférieure dont la diagonale est remplie de 1 et R, une matrice triangulaire supérieure.
a) Ecrire les équations auxquelles donne lieu cette décomposition, en déduire la première ligne de R et la première colonne de L.
b) Prouver que l'on peut déterminer de manière unique L et R en calculant au fur et à mesure la ligne j de L et la colonne j de R pour j allant de 2 à n.
c) En déduire une procédure transformant la matrice A de telle sorte qu'elle devienne la juxtaposition de R avec L sans sa diagonale (aucune autre matrice n'intervient).
d) Déduire de cette décomposition une méthode de résolution de système AX = Y

Solution :
a) A = LR impose pour $j \geq i$, $a_{i,j} = r_{i,j} + \Sigma l_{i,k} r_{k,j}$ avec $1 \leq k \leq i-1$ et pour $j \leq i$, $a_{i,j} = l_{i,j} r_{j,j} + \Sigma l_{i,k} r_{k,j}$ avec $1 \leq k \leq j-1$. En particulier $r_{1,j} = a_{1,j}$ et $l_{i,1} = a_{i,1} / r_{1,1}$

b) Des deux formules générales on tire :
$l_{i,j} = [\ a_{i,j} - \Sigma\ (1 \leq k \leq j-1)\ l_{i,k}\ r_{k,j}\ ] / r_{j,j}$ qui est donc calculé en fonction des coefficients du bloc des lignes et colonnes d'indice inférieurs à i, et en second lieu :
$r_{i,j} = a_{i,j} - \Sigma\ (1 \leq k \leq i-1)\ l_{i,k}\ r_{k,j}$
Ces coefficients n'existent donc qu'à la condition qu'aucun des $r_{i,i}$ ne soit nul.
Ainsi, en changeant les indices de L, on va pouvoir calculer les coefficients $l_{j,i}$ avec la contrainte $2 \leq i \leq j-1$ puis les $r_{i,j}$ pour i allant de 2 à j, et ceci pour j allant de 2 à n, on peut mettre ces coefficients à la place de ceux de A ce qui permet de n'utiliser qu'une seule matrice (figure ci-dessous où les flèches indiquent l'ordre des calculs).



Exemple :

$$\text{Pour A} = \begin{bmatrix} 2 & 1 & 0 & 1 \\ -1 & 2 & -2 & 1 \\ 1 & 1 & -2 & 0 \\ 2 & -2 & -1 & 2 \end{bmatrix}, \text{ A devient } \begin{bmatrix} 2 & 1 & 0 & 1 \\ -0,5 & 2,5 & -2 & 1,5 \\ 0,5 & 0,2 & -1,6 & -0,8 \\ 1 & -1,2 & 2,125 & 4,5 \end{bmatrix}$$

c) à condition d'avoir déclaré : const max = 20; type mat = array [1..max, 1..max] of real;

```
procedure LR (var A : mat; n : integer); {transforme la matrice A n*n en L, R}
        var i, j, k : integer; s : real;
        begin for j := 2 to n do A[j, 1] := A[j, 1] / A[1, 1];
            for j := 2 to n do for i := 2 to j do
                begin s := A[j, i]; {calcul de la ligne j jusqu'à j-1}
                if i <> j then begin for k := 1 to i-1 do s := s - A[j, k]*A[k, i];
                        A[j, i] := s / A[i, i] end; {puis de la colonne j jusqu'à j :}
                s := A[i, j]; for k := 1 to i-1 do s := s - A[i, k]*A[k, j]; A[i, j] := s end end;
```

On pourra faire une vérification grâce à :

```
procedure trianginf (L : mat; n : integer; var A : mat);
            var i, j : integer; {reconstruit A tri. inférieure à partir d'une "moitié" de A}
begin
for i := 1 to n do begin for j := 1 to i-1 do A[i, j] := L[i, j]; A[i, i] := 1; for j := i+1 to n do A[i, j] := 0 end end;

procedure triangsup (R : mat; n : integer; var A : mat);
            var i, j : integer;
begin for i := 1 to n do begin for j := 1 to i-1 do A[i, j] := 0; for j := i to n do A[i, j] := R[i, j] end end;

procedure entree (var A : mat; var n, p : integer);
            var i, j : integer;
begin write ('Combien de lignes '); readln (n); write ('Combien de colonnes '); readln (p);
for i := 1 to n do begin write (' Ligne ', i,'  ');
            for j := 1 to p do begin write (' A (', i:3, ',', j:3, ') = ' ); read (A [i, j]) end; writeln end end;

procedure sortie (A: mat; n, p : integer);
            var i, j : integer;
begin for i := 1 to n do begin for j := 1 to p do write (A[i, j] : 8 : 3); writeln end end;
```

d) Pour un système AX = Y, dans le cas où A se décompose en LR et si A n'est plus utilisée après, en posant Z = RX, on va d'abord résoudre le système triangulaire LZ = Y, puis résoudre très simplement l'autre système triangulaire RX = Z.

5-18° Décomposition QR de Householder et tridiagonalisation

a) Si V est un vecteur colonne de dimension n, on note A^t, la transposée de A et on pose H(V) = $I - 2VV^t / \|V\|^2$ la matrice n*n dite de Householder de V. Construire les procédures nécessaires.

b) Pour toute matrice carrée A, on peut trouver une décomposition A = QR où R est triangulaire supérieure et Q orthogonale ($Q^{-1} = Q^t$) en utilisant l'algorithme suivant :

Si A_{k-1} a déjà ses termes sous la diagonale nuls pour les colones < k, on pose V_k le vecteur colonne k obtenu en prenant les n-k+1 dernières composantes.

Si V_k est nul on pose Q_k la matrice identité d'ordre n-k+1, sinon on pose Q_k la matrice de Householder de Vk diminué de $\|V_k\|$ dans sa première composante.

On borde alors Q_k avec un bloc I_{k-1} en haut à gauche et deux blocs nuls en haut et sur la gauche de façon que Qk soit une matrice symétrique orthogonale d'ordre n.

En posant $A_k = Q_k A_{k-1}$ on peut montrer que A_k possède des zéros sous sa diagonale jusqu'à la colonne k, et on recommence.

Si $S = Q_{n-1}...Q_1$ et R = SA alors R, triangulaire et $Q = S^t$ répondent à la question. Construire les procédures nécessaires.

c) Quelle application peut-on en tirer pour la résolution des systèmes linéaires ?

d) Pour une matrice symétrique A de dimension n*n, on pose V le vecteur de la première colonne privé de son premier élément, et H la matrice de householder de V diminué de $\|V\|$ dans sa première composante. En bordant H en haut à gauche par un 1 et des zéros ailleurs on obtient une matrice U telle que l'on peut vérifier que UA possède des zéros dans sa première colonne à partir de la ligne 2. De plus UAU^t a la même propriété et possède en outre des zéros sur sa première ligne à partir de la colonne 2. En réitérant pour le bloc des n-1 dernières lignes et colonnes de la matrice obtenue, on acquiert une matrice T tridiagonale. Ecrire cette procédure de construction de T.

Solution a, b)

```
procedure transpo (M : mat; n, p : integer; var TM : mat); {TM transposée de M}
            var i, j : integer;
            begin for i := 1 to n do for j := 1 to p do TM[j, i] := M[i, j] end;
procedure muscal  (M : mat; n, p : integer; x : real; var R : mat); {R = xM où x est réel}
            var i, j : integer;
            begin for i := 1 to n do for j := 1 to p do R[i, j] := x * M[i, j] end;
```

```
procedure Householder (V : mat; n : integer; var H : mat);
{construit pour le vecteur V de dimension n, sa matrice H n*n, égale à I-2VtV / |V|2 }
        var i : integer; x : real; W : mat;
        begin x := 0; for i := 1 to n do x := x + sqr(V[i, 1]); x := - 2 / x;
        transpo (V, n, 1, W); mult ( V, W, n, 1, n, H); muscal(H, n, n, x, H);
        for i := 1 to n do H[i, i] := H[i, i] + 1 end;

procedure mult (A, B : mat; n, p, q : integer; var C : mat);
{multiplication de A n*p avec B p*q rendant le résultat C n*q}
        var i, j, k : integer; s : real;
        begin for i := 1 to n do for j := 1 to q do
                begin s := 0; for k := 1 to p do s := s + A[i, k]*B[k, j]; C[i, j] := s end
        end;

function norme (V : mat; n : integer) : real;
        var i : integer; s : real;
        begin s := 0; for i := 1 to n do s := s + sqr(V[i, 1]); norme := sqrt(s) end;

procedure QR (A : mat; var Q, R : mat; n : integer);
        var S, V, H : mat; vnul : boolean;  i, j, k : integer;
        begin for i := 1 to n do for j := 1 to n do begin R[i, j] := A[i, j]; S[i, j] := 0 end;
        for i := 1 to n do S[i, i] := 1;{S est initialisé à Identité}
        for k := 1 to n-1 do
                begin for i := 1 to n do for j := 1 to n do Q[i, j] := 0;
                for i := 1 to n do Q[i, i] := 1; {on prépare la matrice Qk}
                vnul := true; for i := 1 to n-k+1 do
                begin V[i, 1] := R[i+k-1, k]; if V[i, 1] <> 0 then vnul := false end;
                if not (vnul) then begin V[1, 1] := V[1, 1] - norme(V, n-k+1);
                        householder (V, n-k+1, H);
                        for i := k to n do for j := k to n do Q[i, j] := H[i-k+1, j-k+1] end;
                mult (Q, R, n, n, n, R); mult (Q, S, n, n, n, S)
                end;
        transpo (S, n, n, Q) end;
```

c) Pour un système d'inconnue X, AX = Y, en décomposant A = QR, on peut d'abord effectuer le produit $Z = {}^tQY$ puis résoudre simplement de bas en haut le système RX = Z au cas où R n'a aucun zéro sur sa diagonale.
d) On va construire T à partir de A, ainsi que la matrice orthogonale Q vérifiant $A = {}^tQAQ$

```
procedure tridiag (A : mat; n : integer; var T, Q : mat);{A doit être symétrique}
        var U, TU, H, V : mat; i, j, k : integer;
        begin for i := 1 to n do for j := 1 to n do begin Q[i, j] := 0; T[i, j] := A[i, j] end; {copie de A}
        for i := 1 to n do Q[i, i] := 1;
        for k := 1 to n-1 do
                begin for i := 1 to n do for j := 1 to n do U[i, j] := 0;
                for i := 1 to n do U[i, i] := 1; {on prépare la matrice U}
                        for i := 1 to n-k do V[i, 1] := T[i+k, k];
                        V[1, 1] := V[1, 1] + norme(V, n-k); householder (V, n-k, H);
                        for i := 1 to n-k do for j := 1 to n-k do U[i+k, j+k] := H[i, j];
                        transpo (U, n, n, TU); mult (U, T, n, n, n, T);
                        mult (T, TU, n, n, n,T); mult (U, Q, n, n, n, Q) end
        end;
```

Exemple :

$$A = \begin{bmatrix} 1 & 2 & -1 & 3 & 4 \\ 2 & 0 & -2 & 5 & 2 \\ -1 & -2 & 1 & 3 & -1 \\ 3 & 5 & 3 & 4 & 0 \\ 4 & 2 & -1 & 0 & 5 \end{bmatrix} \quad \text{semblable à} \quad T = \begin{bmatrix} 1 & -5{,}477 & 0 & 0 & 0 \\ -5{,}477 & 6{,}9 & -2{,}468 & 0 & 0 \\ 0 & -2{,}468 & -3{,}052 & -4{,}158 & 0 \\ 0 & 0 & -4{,}158 & 4{,}605 & -0{,}601 \\ 0 & 0 & 0 & -0{,}601 & 1{,}547 \end{bmatrix}$$

Cette méthode permet ensuite de calculer les valeurs propres de A comme celles de T qui lui est semblable.

5-19° Résolution de systèmes linéaires par la méthode de Jacobi : il s'agit, pour résoudre AX = B, d'une méthode itérative où A est décomposé en A = D - L - U. Dans cette formule, D, -L, -U étant respectivement les matrices diagonale, triangulaire inférieure et triangulaire supérieure formées avec les parties respectivement diagonale, inférieure et supérieure de A. Initialement on part de X_0 quelconque et on calcule $X_{n+1} = D^{-1}[(L + U)X_n + B]$.
Cette suite converge vers la solution si et seulement si le rayon spectral (max des valeurs propres en valeur absolue) de $I - D^{-1}A$ est inférieur strictement à 1. Une condition suffisante de convergence est que A soit à diagonale strictement dominante c'est à dire que chaque $|a_{ii}| > \sum |a_{ij}|$ pour les $j \neq i$.
Développer à la main le calcul des 4 premiers termes dérivant (0, 0) pour le système :
2x + y = 3 et x + 3y = 4 de solution (1, 1).
Concrètement on stoppera le calcul pour une stabilisation des composantes de X à un ε près.

5-20° Recherche du polynôme caractéristique par la méthode de Krylov : si A est une matrice carrée d'ordre n, son polynôme caractéristique P est le polynôme det (A - xI) dont les racines x sont les valeurs propres de A.
Le théorème de Cayley-Hamilton indique que P(A) = 0, en le multipliant par $(-1)^n$, il devient normalisé, ce qui fait que l'équation caractéristique s'écrit : $A^n + \sum c_i A^{n-i} = 0$ pour i allant de 1 à n, les c_i fourniront le polynôme cherché.
La méthode consiste alors, en partant d'un vecteur quelconque V de n composantes, à calculer $V_0 = A^nV$, $V_1 = A^{n-1}V$, ... , $V_{n-1} = AV$, $V_n = V$, puis l'équation ci-dessus appliquée à V entraîne : $c_1V_1 + c_2V_2 + ... + c_n V_n = -V_0$
Si C est la matrice dont les colonnes sont V_1, ... , V_n, les c_i seront rangés dans la solution X du système $CX = -V_0$. Programmer la procédure construisant cet X à partir de A et de n.

5-21° Recherche du polynôme caractéristique par la méthode de Souriau : si A est une matrice carrée d'ordre n, et tr(A) sa trace, c'est à dire la somme de ses éléments diagonaux, on calcule la suite des matrices et de leurs traces $A_1 = A$, $c_1 = -tr(A_1)$ puis $A_i = (A_{i-1} + c_{i-1} I)A$ et $c_i = -tr(A_i) / i$ jusqu'à $A_n = (A_{n-1} + c_{n-1} I)A$ et $c_n = -tr(A_n) / n$
On peut montrer que les ci sont les coefficients du polynôme caractéristique suivant les puissances décroissantes (le premier étant 1).

5-22° Multiplication des matrices suivant la méthode de Strassen. Soient A et B deux matrices carrées d'ordre pair, "partitionnés" en 4 blocs de mêmes dimensions chacune, en effectuant le produit A*B, montrer qu'on éxecute 7 produits matriciels et 18 additions et en déduire que la complexité du produit matriciel de matrices carrées d'ordre 2^n est en $7^{n+1} - 6*4^n$ à comparer avec la méthode naïve. Exposer un algorithme mixte pour le produit de matrices carrées d'ordre m quelconque qui soit de complexité $m^{2,81}$ et le programmer.

5-23° Calcul de l'inverse généralisée de Moore-Penrose d'une matrice. Si A est une matrice n*p, A^+ est une matrice p*n vérifiant $AA^+A = A$ et $A^+AA^+ = A^+$ ainsi que $(A^+A)^t = A^+A$ et $(AA^+)^t = AA^+$, et bien sûr $A^+ = A^{-1}$ si A inversible (A est non nécessairement unique). Si r est le rang de A, A est semblable à la matrice J qui est nulle sauf les r premiers termes 1 de la diagonale, soit A = PJQ avec P et Q inversibles, alors $Q^{-1}JP^{-1}$ remplit la première condition. Méthode itérative de calcul de Greville : Si A est une matrice-colonne, alors $A^+ = (A^t.A)^{-1}.A^t$. Si maintenant A est une matrice rectangulaire de p lignes et n colonnes, n itérations donnent A^+. On pose a_k la k-ième colonne de A et A_k la matrice formée par les k premières colonnes de A. Si $a_1 = 0$, on pose $A_1^+ = 0$ sinon A_1^+ est la pseudo-inverse de la colonne. A chaque etape, $A_{(k-1)}^+$ étant connue, on pose $d_k = A_{(k-1)}^+a_k$, $c_k = a_k - A_{(k-1)}d_k$, et enfin b_k = si $c_k = 0$ alors $(1 + d_k^t.d_k)^{-1}d_k^t.A_{(k-1)+}$ sinon c_k^+. La matrice A_k^+ est alors formee par $A_{(k-1)}^+ - d_k.b_k$ a laquelle on rajoute en dessous la ligne b_k. A la fin $A^+ = A_n^+$

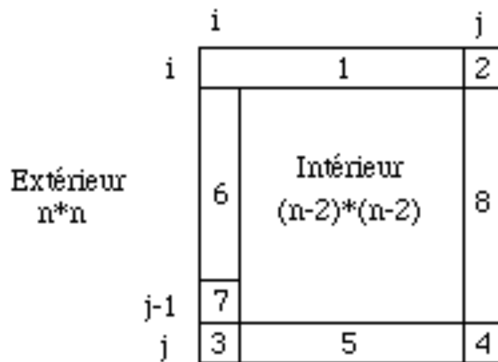
5-24° Matrices de Toeplitz. Ce sont des matrices carrées telle que A[i, j]= A[i-1, j-1] pour $2 \leq i \leq n$ et $2 \leq j \leq n$.Trouver une représentation de ces matrices et montrer qu'elle peuvent s'additionner en O(n). Construire un algorithme permettant de multiplier une telle matrice par un réel en $O(n^{1,58})$. Montrer que le produit de deux telles matrices peut s'effectuer en $O(n^{2,81})$. Construire un autre algorithme de produit en $O(n^{2,58})$ et le programmer.

5-25° Systèmes quelconques. Compléter le programme de résolution de systèmes pour N équations et P inconnues. L'algorithme est le suivant: lorsque sur la colonne J on ne peut trouver de pivot, alors on cherche le premier K tel que $J < K \leq P$ et $A[J, K] \neq 0$ (X_K sera dite inconnue principale et X_J non principale). Si ce K existe, on permute les colonnes J et K de A en prenant soin de signaler cette interversion des noms des inconnues (on pourra utiliser un tableau IND des indices qui sera [1 , 2 , ... , P] au départ).
Si un tel K n'existe pas, alors on est en présence, sur la ligne J, d'une équation du type 0 = A[J, P+1] qui est soit , impossible, soit 0 = 0, en ce dernier cas on peut la supprimer et décrémenter le rang R, puis tasser les lignes de la matrice (il n'y en a plus que N-1)
La triangulation doit se faire pour des numéros de colonnes allant de 1 à min(N,P+1), en terminant la méthode sans qu'il y ait eu d'impossibilité, on doit obtenir une matrice triangulaire de R lignes n'ayant que des 1 sur la diagonales.
IND[R+1], ... , IND[P] constituent les indices des inconnues dont on peut choisir des valeurs, pour ensuite trouver les autres en résolvant un système de Cramer de R équations.
Exemple :

$$A = \begin{bmatrix} 2 & 2 & -3 & 2 & 0 \\ 1 & -2 & 6 & 3 & -2 \\ 1 & 4 & -6 & 5 & 1 \\ 0 & 0 & 3 & 6 & 1 \end{bmatrix} \text{ de rang 3, se triangularise en } B = \begin{bmatrix} 2 & 2 & -3 & 2 & 0 \\ 0 & -3 & 15/2 & 2 & -2 \\ 0 & 0 & 3 & 6 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

5-26° Algorithme du simplexe : un problème de programmation linéaire est la recherche du minimum ou du maximum d'une fonction (linéaire) de coût $\sum c_i x_i$ avec comme contraintes que les inconnues x_i soient positives et un certain nombre d'équations ou d'inéquations linéaires $\sum a_{ij} x_j \leq b_j$. On montre que l'extremum cherché ne peut être qu'en un sommet de l'intersection de ces hyperplans (polyèdre convexe). On suivra la démarche :
a) Ramener les inéquations à la forme $\sum a_{ij} x_j \leq b_j$ puis à des équations $\sum a_{ij} x_j + y_j = b_j$ où la nouvelle variable y_j (dite d'écart) sera donc positive.
b) Ramener les équations à d'autres équations en ajoutant simplement une variable (artificielle) z_k dans le membre gauche, et en modifiant la fonction de coût $\sum c_i x_i + M z_k$ avec M très grand positif si on minimise et négatif si on maximise. On veut de cette façon avoir les variables artificielles nulles à l'optimum.
Les variables artificielles peuvent s'exprimer en fonction des variables du problème, et donc la fonction de coût peut être modifiée. Les contraintes s'expriment donc par un système dont on rangera les coefficients en tableau.
c) On part d'une solution dite "de base" où les variables du problème sont toutes nulles, on peut donc calculer les autres. La "base" est par définition l'ensemble des variables n'ayant que des 0 sauf un 1 dans leur colonne du tableau. S'il y a n inconnues et m contraintes, on aura donc m lignes et n + m colonnes plus une représentant les constantes b. De plus, avec une valeur fixée pour M on peut rajouter une ligne représentant la fonction de coût égale à 0. C'est sur ce tableau que l'on va travailler.
d) Si dans la fonction de coût à minimiser, tous les coefficients des variables "hors base" (celles qui sont nulles) sont positifs ou nuls, l'optimum est atteint.
e) Sinon, on échange la variable i dont le coefficient dans la fonction de coût est le plus fortement négatif en lui donnant la valeur x avec la première variable k de "base" telle que $x \leq \min(b_k / a_{ki})$ les x_k étant les variables de l'ancienne "base".
La ligne k qui réalise ce minimum est choisie, et la colonne i est modifiée par un jeu de combinaisons linéaires sur l'ensemble du tableau de façon à être nulle sauf un 1 à la ligne k. La variable x_i devient "de base".

5-27° Carrés magiques construits récursivement.
Ecrire un programme récursif produisant des carrés magiques formés avec des entiers pris au hasard. On considère pour cela qu'une matrice magique (n-2)*(n -2) de somme s peut-être "bordée" des 4 côtés pour former une matrice magique n * n de somme S.
Montrer que (n-2)S = ns, on choisit arbitrairement tous les éléments de la première ligne sauf le dernier, qui en est déduit, ainsi que toute la dernière ligne. On choisit alors au hasard n-1 éléments de la première colonne et on en déduit le dernier et toute la dernière colonne.



Supposons que la matrice intérieure de dimension n-2 soit déjà magique de somme s, nous allons proceder au remplissage des bords dans l'ordre indiqué pour former une matrice magique de dimension n et de somme S pour l'instant indéterminée.
1° On choisit au hasard, tous les éléments de la bande 1
2° On en déduit par soustraction avec S le dernier de la ligne.
3° On déduit le premier de la dernière ligne avec S somme de la seconde diagonale.
4° On déduit de même le dernier de la dernière ligne.
5° On déduit le reste de la dernière ligne, colonne par colonne. C'est alors qu'il faut vérifier que cette ligne fait bien une somme S : $\sum_{i \leq k \leq j} a_{j,k} = \sum(S\text{-}s\text{-}a_{i,k}) = n(S\text{-}s) - \sum a_{i,k} = n(S\text{-}s) - S$ donc pour que cette somme soit S, il faut obligatoirement S = n(S-s) - S, soit : ns = (n-2)S
6° On choisit au hasard les n-3 éléments de la bande 6
7° On en déduit le dernier inconnu de la première colonne.
8° On déduit le reste de la dernière colonne, et on vérifie de même que sa somme vaut S.
Pour arrêter les appels récursifs, il faudra aboutir à une matrice de dimension 1 si n est impair, sinon, à une matrice magique de dimension 2, qui est alors nécessairement avec ses 4 éléments égaux à s/2. D'où la procédure :

```
procedure maj (S, i, j : integer; var A : tab); {Fonctionne avec S multiple de n = j-i+1}
var k, n, s0 : integer;
begin n := j - i + 1;
if  i = j           then A[i, j] := S
                    else if j = i + 1   then    begin A[i,i] := S div 2; A[i,j]:= S div 2; A[j,i]: = S div 2;
                                                A[j,j] := S div 2 end
                                        else    begin s0 := (n-2)*S div n;
                                                for k := i to j-1 do A[i, k] := random(10);
                                                A[i, j] := S; for k := i to j-1 do A[i, j] := A[i, j] - A[i, k];
                                                A[j, i] := S - s0 - A[i, j]; A[j, i] := S - s0 - A[i, i];
                                                for k := i+1 to j-1 do A[j, k] := S - A[i, k] - s0;
                                                for k := i+1 to j-2 do A[k, i] := random(10);
                                                A[j-1, i]:=S - A[j,i];
                                                for k := i to j-2 do A[j-1, i] := A[j-1, i] - A[k, i];
                                                for k := i+1 to j-1 do A[k, j] := S - A{k, i] - s0;
                                                maj( (n-2)*S div n, i+1, j-1, A) {récursivité terminale} end
end;
```

Remarque : les carrés magiques de dimension n forment un espace vectoriel de dimension vérifiant $d_n = d_{n-2} + 2n-3$ on en déduira que si n est pair alors $d_n = n(n-1)/2$ et si n est impair $d_n = n(n-1)/2 + 1$

5-28° Valeurs propres des matrices tridiagonales Si n est un entier plus grand que 2 et Tn est une matrice symétrique nulle sauf sur ses trois diagonales où aucun bi n'est nul, on peut séparer les n valeurs propres distinctes, et les calculer par dichotomie.

$$T_n = \begin{pmatrix} a1 & b1 & & & & & \\ b1 & a2 & b2 & & & 0 & \\ & b2 & a3 & b3 & & & \\ & & \cdots & \cdots & & & \\ & & & \cdots & \cdots & & \\ & 0 & & & \cdots & \cdots & \\ & & & & \cdots & \cdots & bn-1 \\ & & & & & bn-1 & an \end{pmatrix}$$

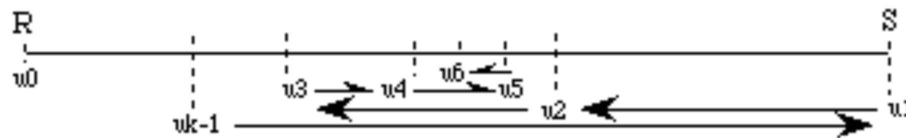
On prendra une constante n, un type vecteur = array [0..n] of real et deux variables globales A, B de type vecteur, contenant respectivement la diagonale principale avec A[0]=0, et l'autre diagonale avec B[0]=B[n]=0. Ces données sont supposées déjà en mémoire.
Soit Pn le polynôme caractéristiques de Tn pour n > 1
a) En posant $P_0(t) = 1$ et $P_1(t) = a_1 - t$, établir $P_n(t) = (a_n - t)P_{n-1}(t) - b_{n-1}^2 P_{n-2}(t)$, et prouver que P_n et P_{n-1} n'ont pas de racine commune.
b) Ecrire une fonction "carac" d'une variable réelle u, calculant la valeur $P_n(u)$ en utilisant des variables locales p0, p1, p2 contenant respectivement, à un certain stade $P_{k-1}(u)$, $P_k(u)$, $P_{k+1}(u)$
c) Si u est un réel fixé, on dit que deux polynômes consécutifs P_k et P_{k-1} présentent une concordance de signe en u si $P_k(u)$ et $P_{k-1}(u)$ ont le même signe, (si $P_k(u)$ est nul par convention son signe sera contraire à celui de $P_{k-1}(u)$)
On admettra que le nombre de valeurs propres strictement supérieures à u est égal au nombre de concordances de signe en u de la suite ($P_0(u)$ $P_1(u)$,......., $P_n(u)$)
Programmer la function "conc" (x, y) et la function "nvpsup" donnant ce nombre.
On utilisera une variable locale nv contenant ce nombre construit au fur et à mesure, et on expliquera la signification des autres variables utilisées.
d) En supposant par ailleurs que toutes les valeurs propres appartiennent à l'intervalle [R, S] tel que $R = \min\{a_i - |b_{i-1}| - |b_i| \ / \ 1 \le i \le n\}$ et $S = \max\{a_i + |b_{i-1}| + |b_i| \ / \ 1 \le i \le n\}$ écrire la procedure intervalle(var R, S : real); qui construira ces deux valeurs.
e) Ecrire la procedure separation(var U : vecteur); qui fournira n+1 valeurs croissantes dans [R, S] telles que chacun des n intervalles ainsi délimités contienne une et une seule valeur propre de Tn.
f) Ecrire la procedure calcul(eps : real; var V : vecteur); qui calcule par dichotomies les n valeurs propres de T_n à eps près.
g) Soit n = 6 et T la matrice tridiagonale dont tous les bi sont égaux à 1, vérifiant a1 = a6 = 5, a2 = a5 = 3, et a3 = a4 = 1

Si on développe par rapport à la dernière colonne, on obtient la relation. Si Pn et Pn-1 avaient une racine commune, alors elle serait commune d'après la relation de récurrence avec Pn-2, et donc avec tous les polynômes précédents donc avec P1 et P0 qui n'en a pas.

```
function conc (x, y : real) : integer; {est 1 s'ils sont de même signe, si l'un des deux est nul ce sera 0, et les deux ne sont pas nuls lors de l'utilisation de "conc" }
                    begin if x*y > 0 then conc := 1 else conc := 0 end;

function nvsup (u : real) : integer;
                    var p0, p1, p2 : real; k, nvp : integer;
                    p0 := 1; p1 := A[1] - u; nvp := conc (p0, p1);
                    for k := 2 to n do        begin p2 := (A[k] - u)*p1 - sqr (B[k-1])*p0;
                                              nvp := nvp + conc (p1, p2); p0 := p1 ; p1 := p2 end;
                    nvpsup := nvp end;
```

```
function carac (u : real) : real; {calcule Pn(u) }
        var p0, p1, p2 : real; k : integer;
        begin p0 := 1; p1 := A[1]-u;
        for k := 2 to n do      begin p2 := (A[k]-u)*p1 - sqr(B[k-1])*p0;
                                p0 := p1; p1 := p2
                                end;
        carac := p2 end;

procedure intervalle (var r, s : real);
        var p, q : real; k : integer;
        begin r := A[1] - abs(B[1]); s := A[1] + abs(B[1]);
        for k := 2 to n do      begin p := A[k]) - abs(B[k-1]) - abs(B[k]);
                                q := 2*A[k] - p;
                                if p < r then r := p;
                                if s < q then s := q end end;

procedure separation (var U : vecteur);
        var r, s : real; nb, k : integer;
        begin intervalle(r, s); U[0] := r; U[1] := s;
        for k := 1 to n-1 do    begin nb := nvpsup(U[k]);
                while nb <> n-k do   if nb < n-k     then U[k] := U[k] + (U[k]-U[k-1])/2
                                                     else U[k] := U[k] - (U[k]-U[k-1])/2;
                U[k+1] := s end end;
```

Cet algorithme assez compliqué consiste à revenir à mi-chemin (vers la gauche) s'il n'y a pas assez de valeurs propres à droite, et à repartir vers la droite de la moitié du segment de gauche au cas où il y a trop de valeurs propres à droite.
Supposons k-1 valeurs propres déjà séparées et nvpsup (Ui = n-i pour $0 \leq i < k-1$, alors si u_{k1} est en s (trop fort) u_{k2} revient à la moitié de la distance, on les note u1, u2, u3 .. sur le schéma:




```
procedure calcul (eps : real; var V : vecteur);
        var U : vecteur; k : integer; a, b, c : real;
        begin separation (U);
        for k := 0 to n-1 do    begin a := U[k]; b := U[k-1];
                                repeat c := (a+b)/2; if carac(c)*carac(a) < 0 then b :=c else a := c
                                until b - a < eps;
                                V[k+1] := c end
        end;
```

g) On trouve R = -1 et S = 7

5-29° Le plus court chemin d'une station de métro à une autre. Ce problème assez long peut recevoir beaucoup de solutions de programmation différentes, parmi toutes les idées de représentation, on pourra affecter chaque ligne du tableau de ses stations, mais les correspondances pourront figurer aussi dans un tableau à trois entrées :
C = array [1..16 , 1..16 , 1 .. 2] of integer où l'élément C[I, J, 1] est le numéro (éventuellement nul) sur la ligne I de la première station commune avec la ligne J, et C[I, J, 2] de la deuxième.

5-30° Valeurs propres par l'algorithme de Rutihauser : pour la matrice régulière A, on décompose A suivant la méthode de Crout A = LU et on pose A_1 = UL (semblable à A, donc de mêmes valeurs propres) puis si $A_n = L_nU_n$ de la même façon, on pose $A_{n+1} = U_nL_n$. Si A est symétrique définie positive, la suite converge vers une matrice triangulaire supérieure donnant sur sa diagonale toutes les valeurs propres de A.

5-31° Valeurs et vecteurs propres par itération-déflation
Supposons qu'une matrice A n*n possède exactement n valeurs propres distinctes $\lambda_1, ..., \lambda_n$ rangées par valeurs absolues décroissantes et V_1 ... V_n des vecteurs propres correspondants. On part d'un vecteur X_0 quelconque, par exemple celui dont toutes les composantes sont 1, et on considère la suite X_k des vecteurs unitaires au sens de $||X||=\sup(|x_1|,...,|x_n|)$, formée par $X_1 = AX_0/m_0$, $X_2 = AX_1/m_1$ etc... en posant $m_k = ||AX_k||$. On montre alors que : $\lambda_1 = \lim m_k$ et $V_1 = \lim X_k$ quand k tend vers $+\infty$.
Si on cherche de la même façon le premier vecteur propre W_1 (associé à λ_1) de A^t (transposée de la matrice A), alors on montre que la matrice définie par $A_1 = A - \lambda_1 V_1.W_1{}^t / W_1{}^t.V_1$ admet les vecteurs propres V_1 ... V_n pour les valeurs 0, $\lambda_2, ..., \lambda_n$, on peut donc recommencer sur A_1.
Expliquer cet algorithme sur un exemple. Expliquez comment vous organisez le travail et le répartissez en différentes procédures réalisant les opérations necéssaires à cette recherche, en particulier une procédure principale calculant les couples λ, V.

Solution partielle, exemple :

$$A = \begin{pmatrix}1 & 3\\ 4 & 2\end{pmatrix} X_0 = \begin{pmatrix}1\\1\end{pmatrix}, X_1 = \begin{pmatrix}0,66\\1\end{pmatrix}, m_1 = 4,66 \ X_2 = \begin{pmatrix}0,786\\1\end{pmatrix}, \ X_3 = \begin{pmatrix}0,735\\1\end{pmatrix},$$

$m_3 = 4,94 \ X_4 = \begin{pmatrix}0,756\\1\end{pmatrix}$, $m_4 = 5,02\ldots$ (le calcul donne une valeur propre 5, l'autre étant 2)

$$\text{puis } A_1 = \frac{1}{7}\begin{pmatrix}-8 & 6\\ 8 & -6\end{pmatrix} X_0 = \begin{pmatrix}1\\1\end{pmatrix}, X_1 = \begin{pmatrix}-1\\1\end{pmatrix}, \ X_2 = \begin{pmatrix}1\\1\end{pmatrix}, m_2 = 2$$

Les deux procedures essentielles sont les suivantes (eps est une constante) :

```
procedure propre (A : matrice; n : integer; var X : matrice; var m : real);
{calcule un vecteur propre X pour la plus petite valeur propre m de la matrice A de dim n}
                    var m0 : real; X0 : matrice; k : integer;
begin m := supabs(X);    repeat   for k := 1 to n do X[k, 1] := X0[k, 1];
                                  mult (A, n, n, 1, X, X0); m0 := m; m := supabs(X)
                         until (abs (m-m0) < eps) and (dist (X, X0, n) < eps) end;

procedure calcul (A : matrice)
                    var i, k : integer; V, W : matrice;
begin for k := 1 to n do X0[k] := 1;
for i := 1 to n do  begin for k := 1 to n do V[k, 1] := X0[k, 1];
                    propre (A, n, V, m); sortie (m, V); transpo (A, n, n, B);
                    propre (B, n, W, m); transpo (W, n , 1, W); mult (W, 1, n, 1, V, B);
                    scalaire (W, n, m / B[1, 1], W); mult (V, n, 1, n, W, B); soust (A, n, n, B, A) end end;
```

Les autres procédures presque transparentes sont à mettre au point. Application :

$$\begin{pmatrix}1&0&0&1\\0&2&1&2\\0&0&3&0\\0&0&4&0\end{pmatrix} \text{ donne } \begin{pmatrix}1/3\\1\\0\\1\end{pmatrix} \text{ propre pour la valeur 4 } \begin{pmatrix}0\\1\\1\\0\end{pmatrix} \text{ pour 2 } \begin{pmatrix}0\\1\\0\\0\end{pmatrix} \text{ pour 3 } \begin{pmatrix}1\\0\\0\\0\end{pmatrix} \text{ pour 1}$$

$$\text{Exemple, valeurs propres de } \begin{Bmatrix}-0,2 & 1,8 & -1,8 & -1,4 & 0\\ -4 & 14 & -11 & -18 & 6\\ -4,4 & 9,6 & -6,6 & -12,8 & 6\\ 0,4 & 2,4 & -2,4 & -3,2 & 0\\ 2 & -4 & 4 & 6 & -3\end{Bmatrix}$$

Ce sont plus généralement : -3, -1, 0, 2, 3.

5-32° Améliorer le parcours du cavalier sur un échiquier, en triant les coups possibles à partir de chaque position suivant le nombre de possibilités qu'ils ont eux-mêmes, suivant un ordre croissant. On arrivera ainsi plus vite aux impasses.

En reprenant le programme du chapitre 4, nous rajoutons une :

```
function fils (l, c : integer) : integer;  {nombre de cases libres dans le tableau "ech" autour de la case l, c}
                function v (b : boolean) : integer; begin if b then v := 1 else v := 0 end;
                begin fils := v(libre (l+1, c+2)) + v(libre (l+2, c+1)) + v(libre (l+2, c-1)) + v(libre (l+1, c-2))
                        + v(libre (l-1, c-2)) + v(libre (l-2, c-1)) + v(libre (l-2, c+1)) + v(libre (l-1, c+2)) end;

procedure tri (var t : tab; d : integer); {réalise un tri de t suivant la clé t[.,2] croissante, méthode bulle}
                var j, x : integer; test : boolean;
begin           repeat  test := true;
                        for j := 1 to d-1 do
                                if t[j,2]>t[j+1,2] then   begin x := t[j, 0];t[j, 0]:= t[j+1, 0]; t[j+1, 0] := x;
                                                          x := t[j, 1]; t[j, 1] := t[j+1, 1]; t[j+1, 1] := x;
                                                          x := t[j, 2]; t[j, 2] := t[j+1, 2]; t[j+1, 2] := x;
                                                          test := false end;
                until test end;

procedure voisins (l, c, n : integer; var d : integer; var tv : tab);
{construit le tableau tv des d cases libres autour de l, c qui est déjà rempli, triéés suivant leurs degrés croissants,
mais si on trouve un voisin libre en impasse, on force le backtrack grâce à d=0}
                var i, f, x, y : integer;
begin i := 1; d := 0; f := 7;
repeat          x := x+ts[i, 1]; y := c + t[i, 2]; {coordonnées d'un voisin}
                if libre(x, y) then begin d := d+1; f := fils(x, y); tv[d, 0] := x; tv[d, 1] := y; tv[d, 2] := f end;
                i := i+1
until (f=0) or (i=9);
if f = 0        then if ech[l, c]= n*n-1  then d := 1 else d := 0
                else if d > 1 then tri (tv, d) end;

function zebre (i, l, c, n : integer) : boolean; {réalise la même chose que la fonction cheval, mais en sautant
d'abord dans les cases où il y a le plus de chances de continuer,, i se trouve déjà dans la cese l, c}
                var k, d, ls, cs : integer; tv : tab; res : boolean;
begin if i = n*n   then begin aff (ech, n, n); zebre := true end
                else    begin  voisins (l, c, n, d, tv);
                        if d =0   then    zebre := false
                                  else    begin k := 1; res := false;
                                          repeat  ls := tv[k, 0]; cs := tv[k, 1]; ech[ls, cs] := i+1;
                                                  res := zebre (i+1, ls, cs, n);
                                                  if  not(res)      then      begin ech[ls, cs] := 0; k := k+1 end
                                          until res or (k > d);
                                          zebre := res end end end;
```

Le programme consistera à "if zebre (1, l0, c0, n) then write ('Voila !')"
Un autre développement consiste à trouver toutes les solutions, on utilise deux variables globales ns nombre de solutions sans retour et nr nombre de solutions avec retour.

```
procedure ecurie (i, l, c, n : integer);
                var k, ls, cs : integer;
begin if i = n*n   then    begin aff (ech, n, n); if revenu (l, c, l0, c0) then nr := nr + 1 else ns := ns + 1 end
                else    begin  for k := 1 to 8 do
                                  begin ls := l + ts[k, 1]; cs := c + ts[k, 2];
                                  if  libre (ls, cs)  then     begin ech [ls, cs] := i+1;
                                                               ecurie (i+1, ls, cs, n);
                                                               ech [ls, cs] := 0 end
                                  end
                        end end;
```

Mais attention pour une case de départ donnée, on trouve environ 300 solutions pour n = 5.
La fonction "revenu" reste à écrire.

5-33° Les mariages [knuth 73]. Etant donnés n femmes et n hommes, chacun ayant établi une liste de préférence des n personnes de l'autre sexe, on dit que le couple (x1, y1) est instable avec (x2, y2) si x1 préfère y2 à y1 et que y2 préfère x1 à x2. Chercher l'algorithme permettant de former des couples de façon à ce qu'il n'y ait aucune instabilité. Il n'y a pas de symétrie, il faudra donc vérifier que (x1, y1) stable avec (x2, y2) et que (x2, y2) stable avec (x1, y1).

Il est possible de représenter les données par un tableau pref [1..2, 1..n, 1..n] définissant :
pref (1, h, f) est le rang de la femme de numéro f dans le classement de l'homme h
pref (2, h, f) ....de l'homme h.................................. la femme f
Ce choix de structuration des données simplifie considérablement l'écriture de "stable".
Le tableau "mari" est construit petit à petit, mari(f) désignant l'homme attribué à la femme f.

```
function stable (f, h : integer) : boolean; {vrai si le fait d'accoupler f et h n'apporte pas d'instabilité sachant que
le tableau "mari" est rempli et stable jusqu'à f-1 } var i : integer;
begin i := 0;
repeat i := i+1 {On teste l'instabilité du nouveau couple avec tous les anciens dans les deux sens}
until (i >= f) or ((pref[1, mari[i], f] < pref[1, mari[i], i]) and (pref[2, mari[i], f] < pref[2, h, f]))
                  or ((pref [1, h, i] < pref [1, h, f]) and (pref [2, h, i] < pref [2, mari [i], i]))
if i = f then stable := true else stable := false
end;

function libre (h : integer) : boolean; { vrai si l'homme h n'est pas encore dans "mari"}
                  var f : integer;
                  begin f := 0;
                  repeat f := f+1 until (mari[f] = h) or (f = n);
                  if mari[f] = h then libre := false else libre := true
                  end;

function mariage (f : integer) : boolean; {vrai s'il est possible de continuer, i étant le numéro de la dernière
femme casée} var h : integer; poss : boolean;
begin if f = n then       mariage := true
                          else    begin h := 1; poss := false;
                                  repeat        if (libre(h)) and (stable (f+1, h))  then   begin mari [f+1] := h;
                                                                                             poss := mariage (f+1)
                                                                                             end;
                                        h := h+1
                                  until poss or (h > n);
                                  mariage := poss
end;
```

On appelera "if mariage (0) then afficher (mari)" avec une procedure d'affichage.
Par exemple si n = 3 et :

$$\text{Si les préférences des hommes sont : } \begin{matrix}1\\2\\3\end{matrix}\left\|\begin{matrix}3\ 2\ 1\\3\ 1\ 2\\2\ 3\ 1\end{matrix}\right| \text{ et celles des femmes : } \begin{matrix}1\\2\\3\end{matrix}\left\|\begin{matrix}2\ 1\ 3\\1\ 2\ 3\\3\ 2\ 1\end{matrix}\right|$$

Alors mariage(0) va appeler mariage(1) une première fois et tomber dans une impasse, puis une deuxième fois pour appeler mariage(2) puis mariage (3) qui terminera avec le tableau des maris [2, 1, 3] (pour les femmes respectives 1, 2, 3).

5-34° Décomposition en inverses : Tout nombre entier peut-il être décomposé en une somme de longueur quelconque d'entiers distincts, dont la somme des inverses vaut 1. Exemple 78 = 2 + 6 + 8 + 10 + 12 + 40 et 1/2 + 1/6 + 1/8 + 1/10 + 1/12 + 1/40 = 1 (En fait c'est toujours vrai à partir de 78, mais avant ?)

Solution (la fonction "possinv" est interne à la procédure "decompinv" :

```
procedure decompinv (n : integer);
                  var  L : array [0..100] of integer; eps : real;
```

```
function possinv (n, i, s : integer; si : real) : boolean;
{s'il est possible de continuer à emplir le tableau L après L[i] avec des entiers entre k=L[i] et n (il y en aura b) de
telle sorte que n=L1+L2+...Lb et 1=1/L1+1/L2+1/L3+...+1/Lb On note s=L1+L2+...+Li et si=1/L1+...+1/Li}
        var j : integer ; poss : boolean;
        begin
        if (s=n) and (abs (si-1) < eps) then    begin writeln ; write (n,' = ');
                                                for j := 1 to i-1 do write (L[j], '+');
                                                write (L[i], ' et 1 = ');
                                                for j := 1 to i-1 do write ('1/', L[j], ' + ');
                                                writeln ('1/', L[i]); possinv := true end
        else if (s>n) or (si > 1+eps) then possinv := false
        else    begin j := L[i] + 1; poss := false;
                repeat L[i+1] := j; poss := possinv (n, i+1, s+j, si+1/j); j := j+1
                until poss or (j > n-s);
                possinv := poss end
end;
begin eps := 0.00001; L[0]:=0; if possinv (n, 0, 0, 0) then write ('voila ') else write (' pas de solution pour ',n)
end;
```

Quelques résultats :

	11 = 2 + 3 + 6	24 = 2 + 4 + 6 + 12
30 = 2 + 3 + 10 + 15	31 = 2 + 4 + 5 + 20	32 = 2 + 3 + 9 + 18
38 = 3 + 4 + 5 + 6 + 20	43 = 2 + 4 + 10 + 12 + 15	45 = 2 + 4 + 9 + 12 + 18

5-35° Etude des puzzles carrés constitué uniquement de carrés :

Exemple $112^2 = 2^2 + 4^2 + 6^2 + 7^2 + 8^2 + 9^2 + 11^2 + 15^2 + 16^2 + 17^2 + 18^2 + 19^2 + 24^2 + 25^2 + 27^2 + 29^2 + 33^2 + 35^2 + 37^2 + 42^2 + 50^2$ (découvert en 1978 par A.Duijvestijn)

a) Le prouver pour 608 avec 26 carrés

b) Faire un programme déterminant une solution, le tester avec 112, 175 et 608

c) .... toutes les solutions.

```
procedure decompquadra (n : integer);
        var i, b : integer; L : array [1..100] of integer;
function possquadra (n, s, i, v : integer) : boolean; {vrai s'il est possible de continuer à remplir le tableau L avec
des entiers décroissants de [1, v-1] sachant que L[i] sera v, la somme des carrésde L[1] à L[i-1] étant s ≤ n*n}
        var j : integer ; poss : boolean ;
begin   if (v<>0) and (s<>0) then writeln ('somme=',s,' rang=',i,'->',v);
        if sqr(n) = s then begin L[i] := v; b := i; possquadra := true end
        else if (sqr(n) < s) or (v = 1) then possquadra := false
        else    begin j := min(v-1, trunc (sqrt (n*n-s)));
                repeat poss := possquadra (n, s+j*j, i+1, j); j := j-1 until (j = 0) or poss;
                if poss then begin L[i] := v; possquadra := true end end
end;
begin b := 0;
if possquadra (n,0,0,n) then    begin write (sqr (n),' = carré (', n, ') = ');
                                for i := 1 to b-1 do write ('carré (', L[i] ,') + ');
                                writeln ('carré (', L[b], ')') end end;
```

Maintenant pour avoir toutes les décompositions :

```
procedure touquadra (n, s, i: integer); {cherche à continuer de remplir le tableau t avec des entiers décroissants
de [1, ti] sachant que la somme des carrés de t[1] à t[i-1] étant s ≤ n*n}
        var j : integer;
begin t[0] := n;
if sqr (n) = s then     begin write (sqr(n),' = carré (',n,') = ');
                        for j := 1 to i-1 do write ('carré (',t[j],') + ');
                        writeln('carré (',t[i], ')') end
else for j := min (t[i]-1, trunc (sqrt (n*n-s))) downto 1 do begin t[i+1] := j; touquadra (n, s+j*j, i+1) end
end;
```

Un résultat (ils sont très nombreux):

$11^2 = 10^2 + 4^2 + 2^2 + 1^2 = 9^2 + 6^2 + 2^2 = 8^2 + 6^2 + 4^2 + 2^2 + 1^2$

5-36° Indépendamment des procédures du 34°, étant donné une solution algébrique pour un entier n donné, trouver une représentation et visualiser une solution du puzzle (elles sont très peu nombreuses), c'est à dire une disposition géométrique des carrés. Trouver toutes les solutions géométriques pour une même solution algébrique.

Indication : si b nombre sont déjà acquis dans le tableau L[1..b]. L'idée est de considérer un tableau t[0..n, 0..n] de booléens indiquant les places déjà occuppées dans le carré n*n. On construit le tableau P[1..b] formés par les points (x, y) du coin haut gauche d'un carré de côté L[i] (en ordre décroissant). On peut aussi se servir du problème du professeur Facon (chapitre précédent). On définiera :

continuer (i)

```
{vérifiant qu'il est possible de terminer avec le i-ième carré placé en P[i].x, P[i].y}
                si i = b alors demander une touche pour continuer et afficher les séparations
                sinon  x := -1;  y := 0;
                       répéter x := x + 1;
                               tant que  t[x, y]  faire y := y + 1;
                               tant que  L[i + 1] ≤ n - y   faire   emplir (x, y, L[i + 1])
                                                                    continuer (i + 1)
                                                                    vider (x, y, L[i + 1])
                       jusque n - x < L[i + 1]
```

emplir (x, y, v, drap) {drap booléen pour savoir si emplir ou vider}

```
                pour i := 0 à v - 1 faire pour j := 0 à v - 1 faire t[x + i, y + j] : = drap
                fixer la couleur à noir si drap et à blanc sinon
                placer (x + v, y); tracer (x + v, y + v); tracer (x, y + v)
```

Solution pour n = 112 côté d'un carré dont la décomposition est donnée à l'exercice précédent: