CHAPITRE 3

EXEMPLES GRAPHIQUES

Représentation de courbes

Les exemples donnés ici ne posent pas de grand problème de programmation, il faut savoir, pour les lire qu'on accède aux bibliothèques de fonctions graphiques avec "uses graph ;" sur PC et "uses memtypes, quickdraw ;" sur MAC.

Les différents modes graphiques sont tellement variés qu'il vaut mieux toujours poser d'emblée des constantes L et H signifiant largeur et hauteur d'écran. Ainsi, le programme peut commencer par l'attribution de deux valeurs, ce qui permet d'avoir une grille, par exemple, de L = 640 colonnes numérotées de 0 à 639 et H = 200 lignes numérotées de 0 à 199 de haut en bas.
Les instructions utilisées ici sont "moveto (u, v)" pour se placer à un point quelconque (colonne u, ligne v) de l'écran, et "lineto (u, v)" pour tracer un segment dans la couleur considérée, depuis le dernier point placé jusqu'à celui (u, v) qui est donné en paramètre.
Rappelons, en outre, que "trunc" est la fonction de troncature, trunc (45.123) = 45 par exemple, et que "div" est la division entière 45 div 6 = 7.

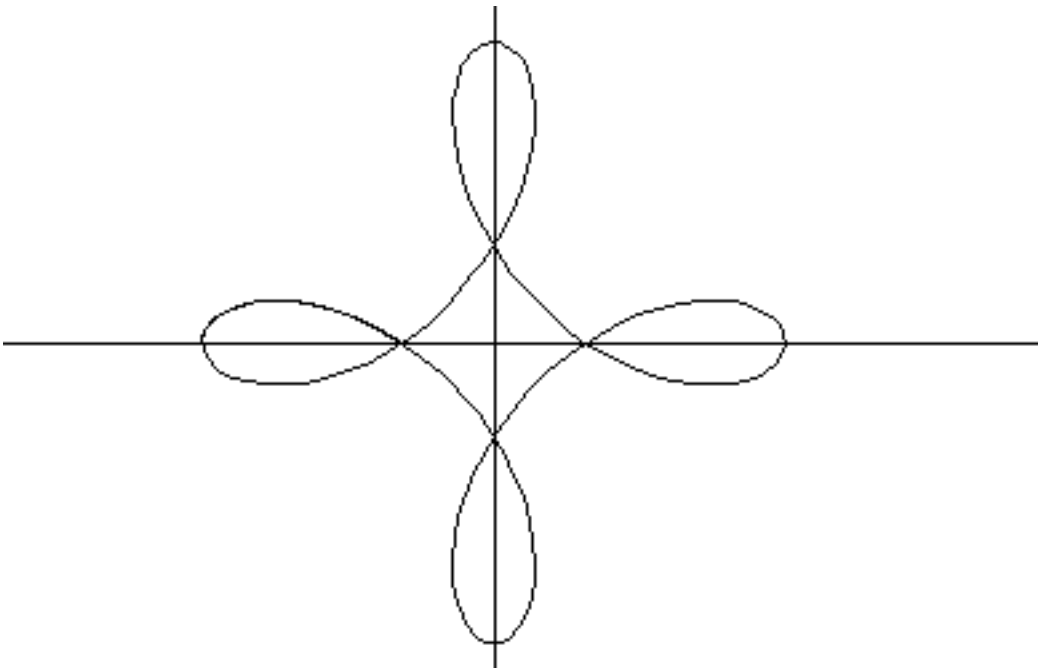
Afin d'éviter les confusions nous écrirons systématiquement les coordonnées d'écran u et v ou bien c (colonne) et l (ligne), et nous réserverons x, y pour le point de vue du dessin réel sur le papier. C'est sur ce dessin que l'on doit définir la fenêtre A < x < B et C < y < D.

Dans ce premier programme, la fenêtre est fixée par le programme proprement dit, il faut y remarquer la séparation très nette entre les déclarations : les deux fonctions f et g, et les deux sous-programmes de tracé d'axes et de tracé de courbes. Le programme proprement dit n'est constitué que par la dernière ligne fournissant des arguments à la procédure de trace. Ce point sera examiné plus en détail au chapitre 4 .

```
program courbes ;  {Courbes paramétrées X=f(t) et Y=g(t) l'exemple donné est une épicycloïde "rallongée" ou trochoïde }
const L = 640; H = 200;          {dimensions de l'écran}

function f (t : real) : real ;
                begin f := -3*cos (t ) - 2*cos (3*t ) end ;

function g (t : real) : real ;
                begin g := -3*sin (t ) + 2*sin (3*t ) end ;

procedure axes (A, B, C, D : real);
{ A, B, C, D seront des bornes données. Si on donne x entre A et B, et y entre C et D alors les coordonnées sur l'écran sont U = L*(X-A) / (B-A) et V = H*(Y - D) / (C - D). On élargira cette question dans les programmes graphiques suivants }
begin           if A*B < 0 then begin moveto ( L*A div (A - B) , 0); lineto (L*A div (A - B), H) end;
                if C*D < 0 then begin moveto (0, H*D div (D - C)); lineto (L, H*D div (D - C)) end end;
```

```
procedure trace (A, B, C, D, t1, t2, h : real );
{ Réalise le tracé de la courbe pour A < X < B et C < Y < D et t1 < t < t2 avec t allant de h en H ( le pas de calcul ).}
                var t : real ;
                begin  axes (A, B, C, D) ; t := t1; moveto (L*(f(t) - A) div (B-A) , H*(g(t) - D) div (C-D));
                repeat lineto (L*(f(t) - A) div (B-A) , H*(g(t) - D) div (C-D) ); t := t+h
                until t > t2 ;
                end ; { Le point (f(t), g(t)) décrit la courbe quand t va se 0 à 2π }

begin trace (-6, 6, -6, 6, 0, 6.28, 0.01) { t va de 0 à 2π } end.
```

Exécution



Remarques

Les multiples versions de Pascal et les multiples cartes grahiques obligent au maximum de souplesse, c'est à dire qu'il est préférable de conserver les vraies coordonnées x, y dans l'utilisation de procédures que l'on appelera "place (x, y)" et "trace(jusqu'à) (x, y)" lesquelles seront fonction d'une fenêtre [A, B]*[C, D] que l'on définira en constantes, variables globales ou paramètres suivant la souplesse désirée.

Par exemple pour Macintosh :

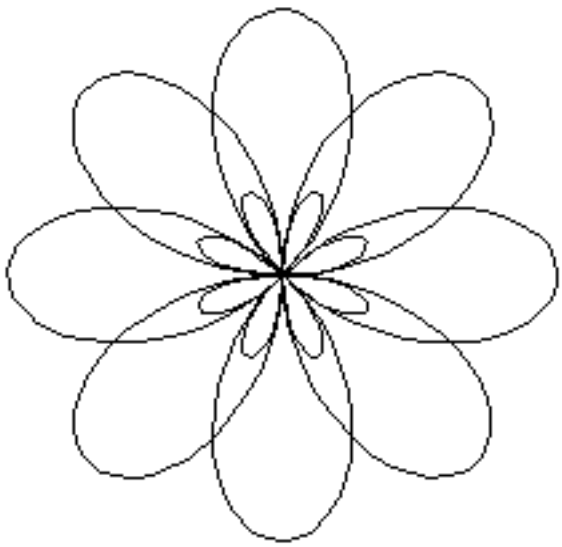
```
uses memtypes, quickdraw;
const A = -2; B = 2; C = -1; D = 1.5; L = 480; H = 270;
{Pour x entre A et B, et y entre C et D les coordonnées d'écran sont U = L*(X-A)/(B-A) et V = H*(Y-D)/(C-D)}

procedure place (x, y : real); {place le point de vraies coord x, y}
    begin moveto ( (L*(x - A)) div (B - A), (H*(y - D)) div (C - D) ) end;
procedure trace (x, y : real); {trace le segment vers le point de vraies coord x, y}
    begin lineto ( (L*(x - A)) div (B - A), (H*(y - D)) div (C - D) ) end;
```

On pourra utiliser aussi une procedure "grospoint" placant cinq ou neuf point autour de (x, y). Dans l'ancienne version de Turbo-Pascal, on accède au mode graphique par "hires", un point est placé par plot (U, V, couleur) et un segment de droite est décrit par draw (U1, V1, U2, V2, couleur).

3-1° Compléter le programme de tracé de courbes pour des courbes paramétrées de type x = f(t) et y = f(t) ou ρ = f(θ) ou des tracés aléatoires de courbes de Lissajoux x = cos(nt) y = sin(mt), de rosaces ρ = a+cos(bθ/c), de cycloïdes x = (n+1)cos(t)-mcos(n+1)t et y = (n+1)sin(t)-msin(n+1)t, en prenant des petites valeurs entières de m, n, a, b, c au hasard.

```
program courbes;{ Courbes paramétrées X=f(t) et Y=g(t) turbo-pascal sur mac}
        uses memtypes, quickdraw;
        const A = 0; B = 20; C = -2; D = 2 ; L = 480; H = 270;
procedure place (x, y : real);{place le point de vraies coord x,y}
        begin moveto (L*(x-A)) div (B-A), (H*(y-D)) div (C-D)) end;
procedure trace (x, y : real);{trace le segment vers le point de vraies coord x,y}
        begin lineto (L*(x-A) div (B-A), (H*(y-D)) div (C-D)) end;

function f (t : real) : real; begin F := t {ou autre définition} end;

function g (t : real; m : integer) : real;
        var i, s : integer; v : real; {ici m est un paramètre servant au problème suivant }
        begin v := 0; s := 1;
        for i := 1 to m do begin v := v + s* sin(i*t) / i; s := -s end;
        G:= v end;

function ro (t : real) : real; {cas de la spirale d'Archimède }
        begin ro := t end;

procedure axes;         begin if A*B <0 then begin place (0,D); trace (0,C) end;
                        if C*D <0 then begin place (A,0); trace (B,0) end end ;

procedure courbe ( x1, x2, h : real; m : integer);
        var x : real;
        begin axes ; x := x1; place (x, g(x, m)); repeat trace(x , g (x, m)); x:= x+h until x > x2 end;

procedure polaire (t1, t2, h : real) ;
        var t : real;
        begin t := t1; place ( ro(t)*cos(t), ro(t)*sin(t));
        repeat trace (ro(t)*cos(t), ro(t)*sin(t)) ; t := t + h until t > t2 end ;
```

Puis un programme appelle "polaire", par exemple : ρ = 0.5 + cos(8θ/3)

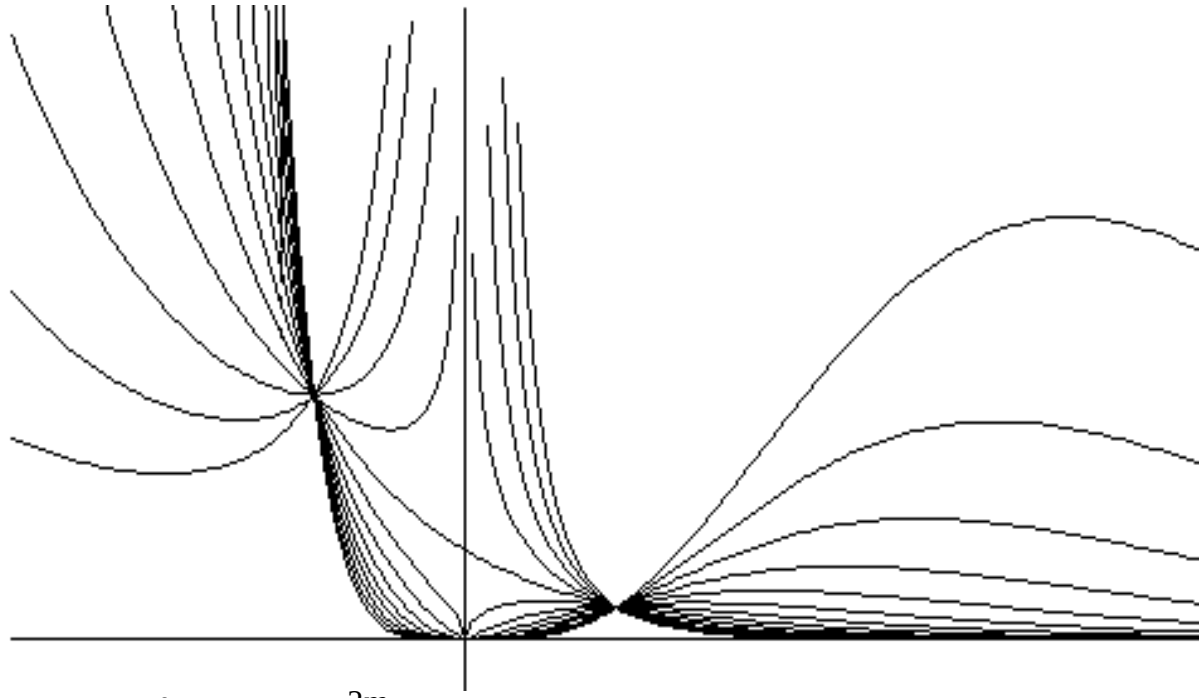


Essayer x = cos 3t - 2sin3t, y = ln|sin2t - cos5t| pour une belle architecture, ρ = √(2|cos2θ) - 1 pour avoir les pétales avec la fleur et 1/ρ = √(1+sin2θ) + √(1-sin2θ) vous aurez une surprise!

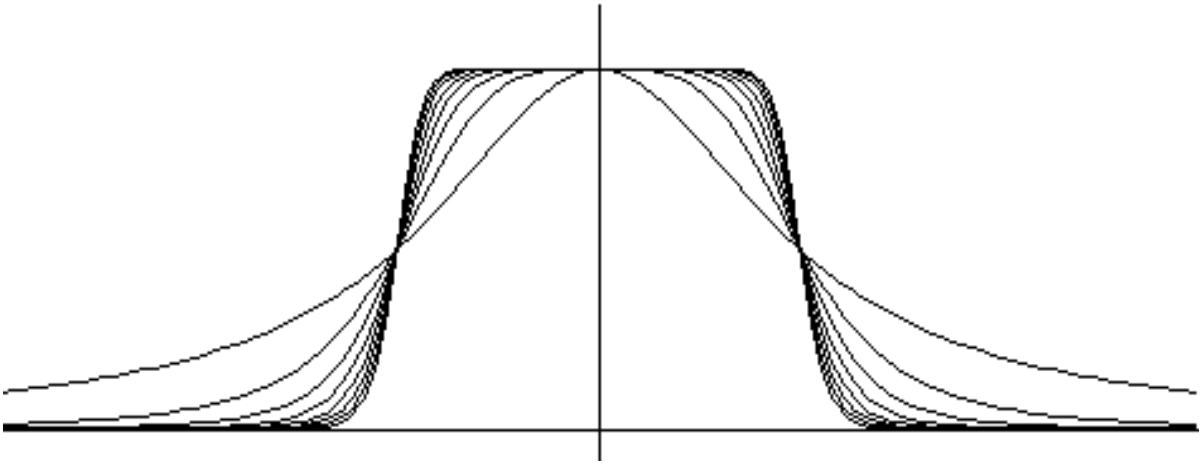
3-2° Tracer une famille de courbes sur un même repère, exemple : $f(x) = e^{-x}|x|^m$ pour m allant de -2 à 4 de 1/2 en 1/2

On écrit , en plus de f, g et "courbe" du programme précédent :

```
procedure famille (deb, fin, pas : integer);
        var m : integer; {trace les courbes Γm pour m de "deb" à "fin" avec le "pas" }
begin m := deb; repeat courbe (0, 20, 0.1, m); m := m + pas until m > fin end;

begin famille (-2, 4, 0.5) end. {©ette ligne constituant le programme pour les 13 courbes ci-dessous}
```



Ou bien, si $f(x) = 1/(1 + x^{2m})$ pour $1 < m < 9$, on peut voir la convergence de cette suite de fonction vers une fonction discontinue.



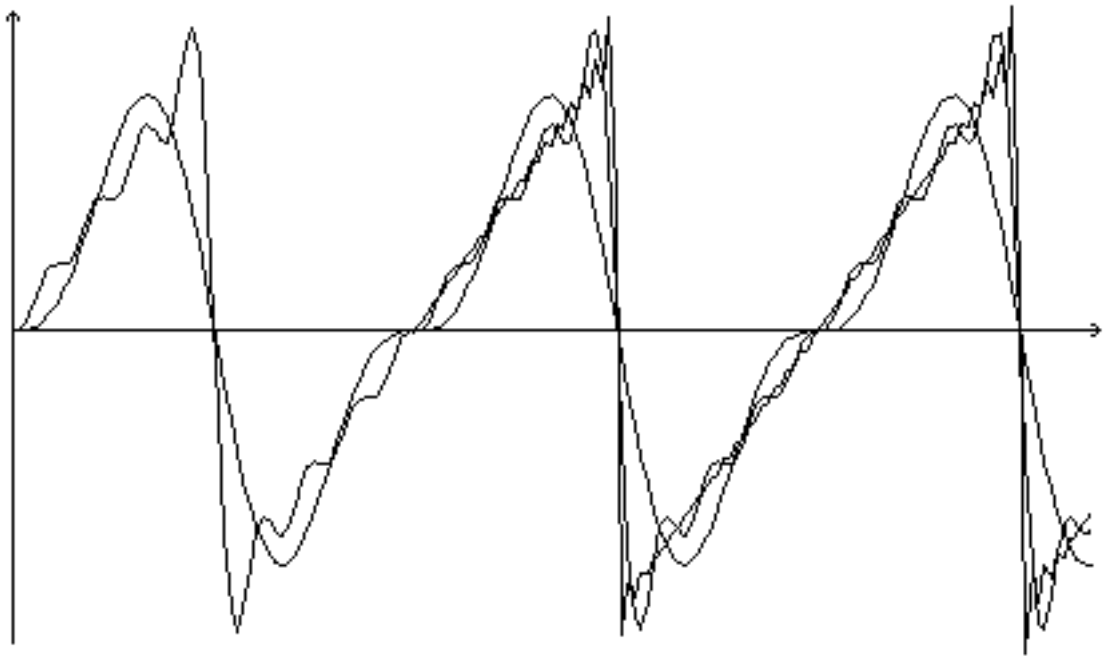
3-3° Dans un cadre carré où chaque côté est régulièrement subdivisé en 10, tracer des segments joignant des points de deux côtés adjacents correspondant aux mêmes subdivisions, à la façon des tableaux de fils tendus par des clous.

3-4° Produire une série de carrés emboîtés, chacun ayant ses sommets en des points divisant dans le même rapport les côtés du carré dans lequel il s'inscrit.

3-5° Dessiner une toile d'araignée (une sorte de spirale octogonale).

3-6° **Développement de Fourier.** Etudier graphiquement la convergence d'un développement, par exemple $f_n(x) = \sin(x) - \sin(2x)/2 + \ldots + (-1)^n \sin(x)/n$. La série de Fourier converge vers la fonction de période π, nulle pour $k\pi$ (k entier) et valant x sur $]-\pi, \pi[$.

On a représenté les tracés superposés de f_2, f_8 et f_{30} sur les deux dernières arches.

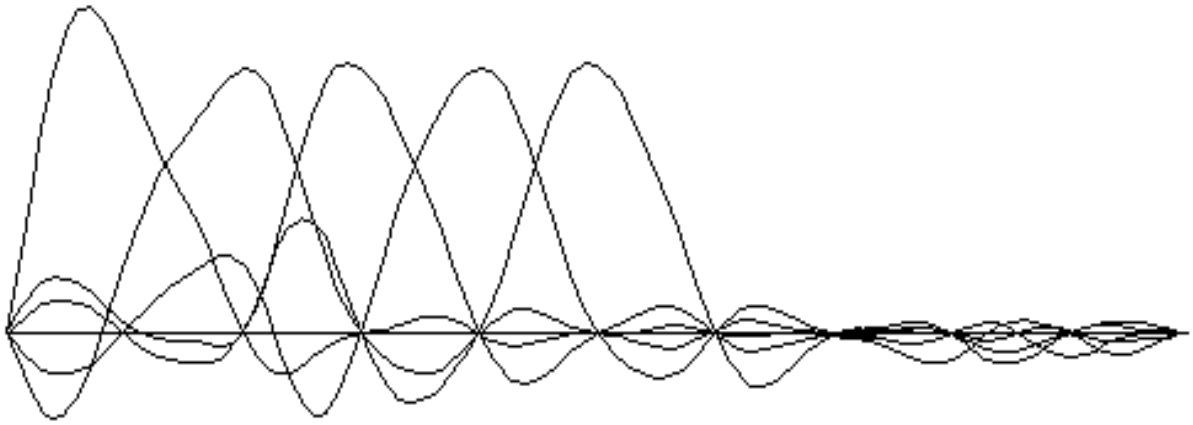


3-7° **Tracé de la clothoïde** $x = a \int_{0,t} \cos \pi u^2/2 \, du \quad y = a \int_{0,t} \sin \pi u^2/2 \, du$ (Intégrales de Fresnel, u est la variable d'intégration). Cette courbe a un rayon de courbure proportionnel à la distance parcourue sur la courbe (son équation intrinsèque $R = \pi s/a^2$ est directement utilisable si l'on dispose des instructions de manipulations de "tortues" au chapitre suivant.)

On se servira de :

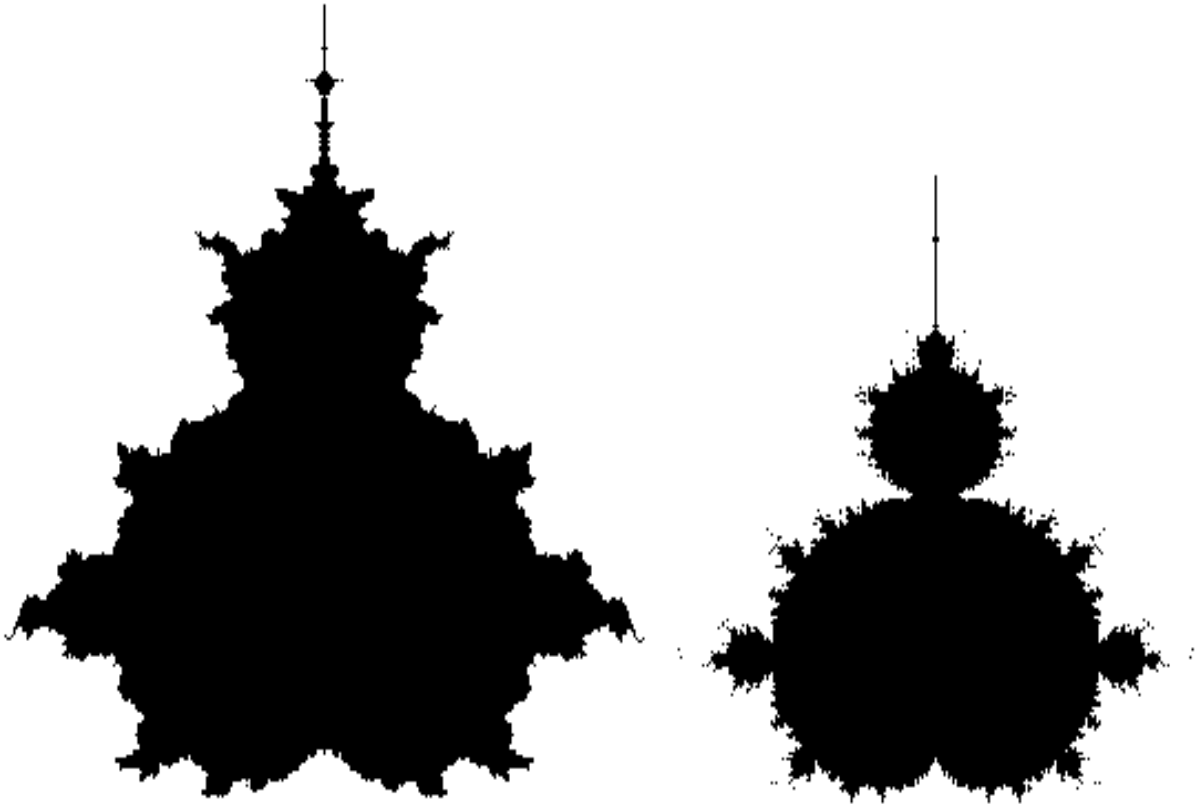
```
function integrale (a, b : real; n: integer) : real; {donne l'intégrale de F sur [a, b]}
        var s, h : real; i : integer;
        begin h := (b-a)/n/2; s := f(a) + f(b) + 4*f(a + h);
        for i := 1 to n-1 do s := s + 2*f(a + 2*i*h) + 4*f(a + (2*i+1)*h);
        integ := h*s/3 end;
```

3-8° Tracé des $fn(x) = 1/\pi \int_{[0,\pi]} \cos|nu - \sin xu| \, du$, pour les entiers $1 \le n \le 5$ et $0 \le x \le 10$ avec Simpson.



3-9° Faire rebondir une balle à l'intérieur d'un cadre rectangulaire.

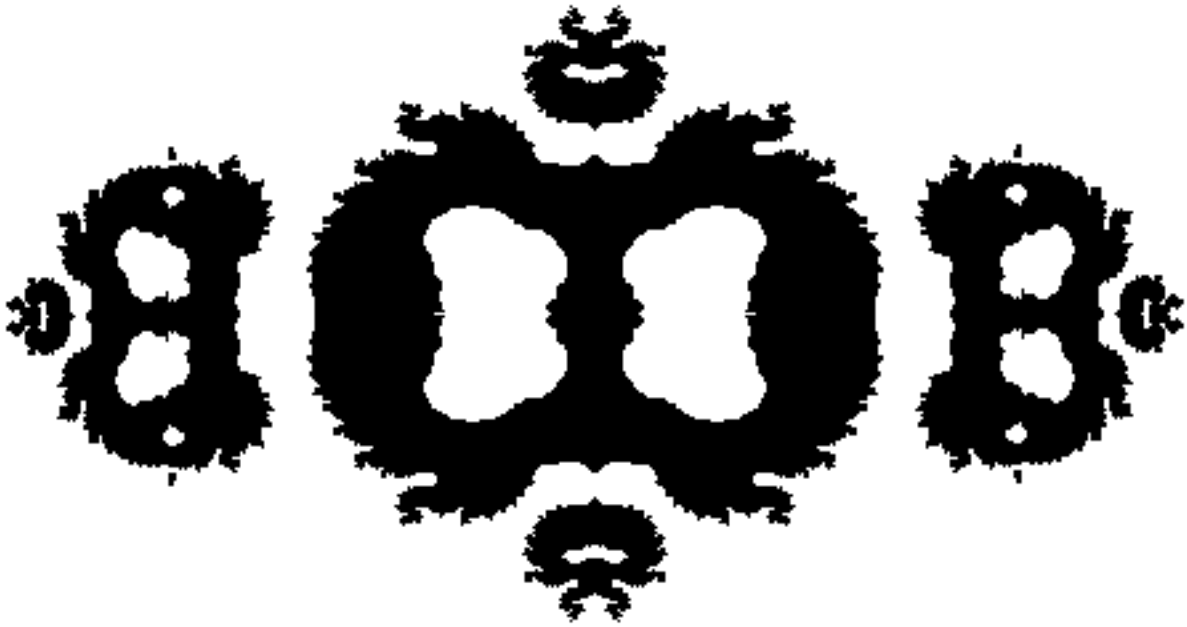
3-10° **Ensemble de Mandelbrot :** On cherche l'ensemble des complexes c de $[-2, 1]^2$ tel que l'itération de la suite déterminée par $z_0 = 0$ et $z_{n+1} = f(z_n) = z_n^2 + c$ ait ses 10, 30, ou 60 premiers termes (différence entre les deux figures) dans un cercle de centre O et de rayon 3 (ce qui ne veut pas dire qu'elle converge, ni que cette "convergence" ait lieu pour les autres complexes que 0, c'est l'ensemble des c tels que Kc ensemble de Julia, soit connexe). Pour chaque point c d'une fenêtre de l'écran. (L'axe des x est dirigé ici vers le bas, l'axe des y vers la droite. Par ailleurs la borne 2 peut suffire.)



```
Program mandelbrot; uses memtypes, quickdraw; {Accès aux fonctions graphiques sur Macintosh }
        const A = - 3; B = 2; C = - 1.2; D = 1.2 ; L = 480; H = 270; { On définit une fenêtre }
procedure point(x, y : real);
        var u, v : integer;
{Pour x entre A et B, et y entre C et D les coordonnées à l'écran sont U=L*(X-A)/(B-A) et V=H*(Y-D)/(C-D)}
        begin u := round(L*(x-A)/(B-A)); v := round(H*(y-D)/(C-D)); moveto(u, v); lineto(u, v) end;

function conv (x, y : real) : boolean;  {dit si la suite zn+1 = zn2+x+iy converge, le départ étant toujours 0}
        var x0, y0, x1, y1 : real; it : integer;
        begin x1 := x; y1 := y; it := 1;
        repeat  it := it + 1; x0 := x1; y0 := y1; x1:=x+x0*x0-y0*y0; y1 := y + 2*x0*y0
        until ( abs(x0)+abs(y1) > 3) or  (it > 30);
        conv := (it > 30) end;

procedure cardio (p : real); { p est simplement un pas de calcul pour x et pour y }
        var x, y : real;
        begin  x := -2;
        repeat   y := 0;
                repeat  if conv(x, y) then begin point (x, y); point(x,-y) end; y := y + p until y > 2;
                x := x + p
        until x > 1 end;

begin cardio (0.009) end.
```

Un résultat récent de Mitsuhiro Shishikura montre que la frontière de M est un ensemble fractal de dimension 2.

3-11° Un autre programme inspiré de ce dernier produit de magnifique résultats, c'est l'étude des points z0 de convergence de la suite $z_{n+1} = f(z_n) = z_n^2 + 0{,}78 + 0{,}2i$. On calcule les parties réelle et imaginaire x, y de z_n avec n limité à 20, dès que $x^2 + y^2 > 5$, on considère qu'il y a divergence, sinon on place le point et ses symétriques par rapport aux axes.

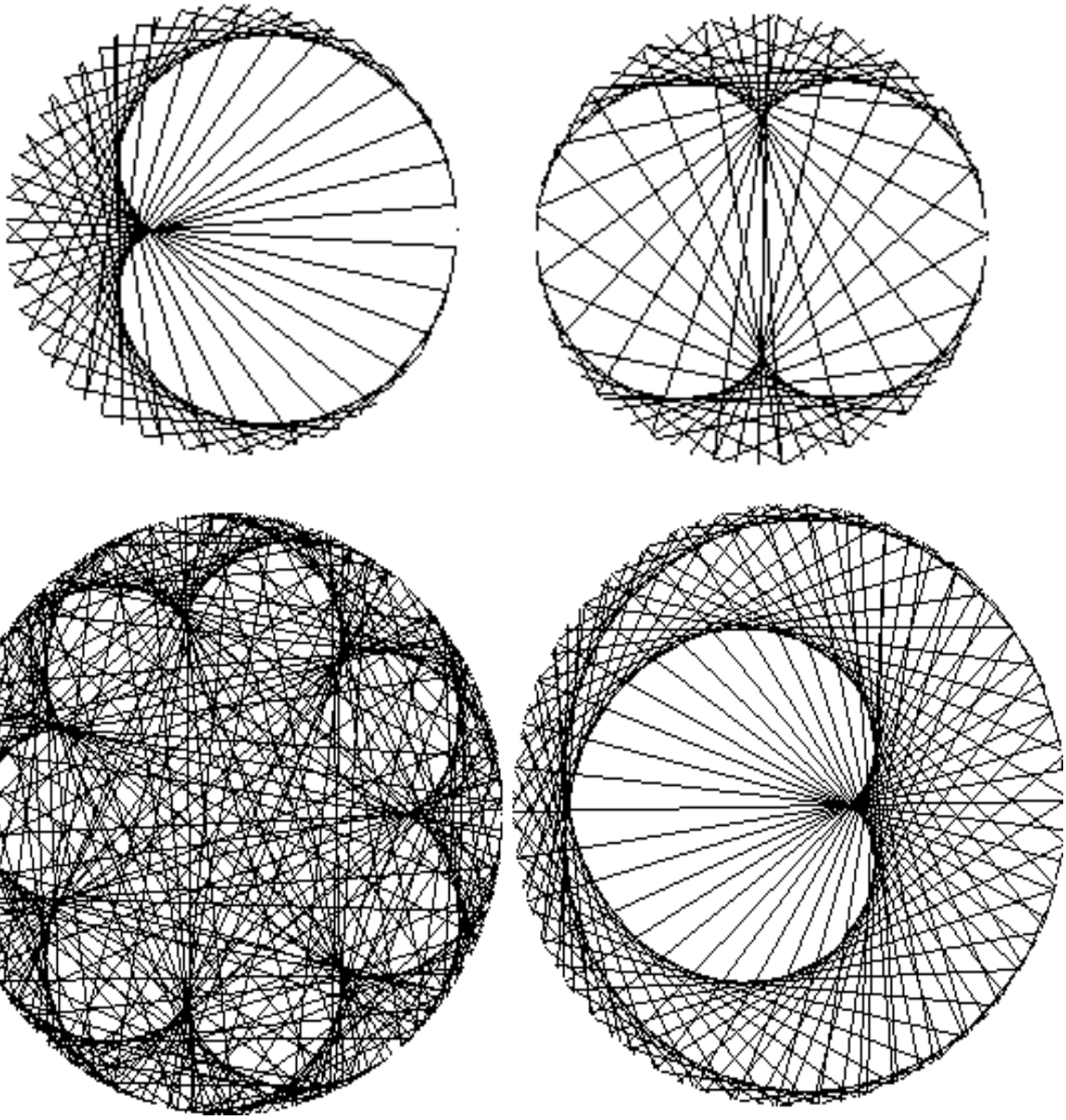


```
procedure f (x, y : real; var re, im : real); { calcule le complexe (re, im) image de (x, y) par f }
                begin  re := sqr(x) - sqr(y) - 0.8; im := 2*x*y - 0.2  end;

function converge (x, y : real): boolean;
{on calcule zn pour n < 20, dès que x2 + y2 > 10, on dit qu'il y a divergence}
                var re, im : real; it : integer;
                begin    re := x; im := y; it := 0;
                         repeat it := it + 1; f (re, im, re, im)
                until (it > 20) or (sqr(re) + sqr(im) > 5);
                if it > 10 then converge := true else converge := false
                end;

procedure dessin (p : real); var x, y : real;
 {On cherche l'ensemble des complexes z de [-1, 1]2 telle que l'itération de z:=f(z) converge et en ce cas on
place le point et son symétrique par rappor à l'origine.}
                begin  x := 0;
                repeat  y := 0;
                         repeat   if converge (x, y) then   begin point (x, y) ; point(-x, y);
                                                            point (-x, -y); point(x, -y) end;
                                  y := y + p
                         until y > 1;
                x := x + p
                until x > 2  end;
```

3-12° Pour construire un diaphragme, on trace un polygone régulier de n côtés mesurés par c. Puis en avançant sur l'un des côtés de rc (avec $0 < r < 1$) on trace un autre polygone régulier de n sommets dont chacun se trouve sur un côté du polygone précédent au même rapport r. (Elégante solution dans le langage Logo disposant d'instructions pour fixer un cap et calculer une distance parcourue.)

3-13° **Enveloppes de droite**, sur un cercle de rayon r, on joint les points de coordonnées polaires (r, θ) et (r , nθ) en faisant varier θ de p en p (par exemple p = 5°). Pour n = 2 on obtient une cardioïde, n = 3 une néphroïde etc...

```
procedure env (p : real ; n : real );       {lancée avec :  n = 1,5 ou env (0.1 , 4.5) pour le dernier dessin.}
              var a : real;
              begin a := 0;
              repeat          place (cos (a), sin (a)) ; trace (cos (n*a), sin (n*a)) ; a := a + p
              until a > 8*pi
              end;
```



3-14° (Rien à voir avec π) **Simulation d'un vol de mouettes**, [Bonabeau 94] on place aléatoirement, mais vers le coin gauche en haut de l'écran, n = 20 points. Chacun ayant une vitesse entre 2 et 20 pixels par intervalle de temps et une direction (initialement vers le coin en bas à droite). Chaque oiseau doit contrôler ses camarades dans un rayon de 10 pixels et éviter les collisions en modifiant sa direction (±5° par exemple), puis (règle moins prioritaire) atteindre leur vitesse moyenne en modifiant éventuellement la sienne de ±1, sinon il doit modifier sa direction pour s'approcher de leur centre de gravité.

3-15° Morphologie mathématique
Etant donné un ensemble E dans R^2, et B un "élément structurant" qui peut être un voisinage fermé et connexe de 0. On peut considérer par exemple $B = \{x / |x| \leq \varepsilon\}$, mais le plus simple est de prendre la norme "sup" pour laquelle B est un carré dans le plan, et B_z est donc le carré de côté 2ε centré sur z. On définit :

Dilatation de E par rapport à B : $Dil_B(E) = \{x / E \cap B_x \neq \emptyset\}$

Erosion de E : $Ero_B(E) = \{x / B_x \subset E\}$

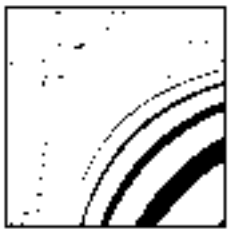
Il s'agit donc d'une interprétation tout à fait discrète des notions topologiques d'intérieur et d'adhérence.
Ouverture de E : Dil (Ero (E)) qui donne de bons résultats en traitement d'images, le but étant de rectifier des formes et d'obtenir des images schématiques. On veut une image non bruitée à partir d'une image floue. Faire un petit logiciel de démonstration avec quelques images tests, réalisant la dilatation de l'érosion floue en permettant un choix de a.

Solution : c'est un excellent exemple de manipulations de boucles.

```
program morphologie ;
        uses memtypes, quickdraw ; {sur Macintosh}
        const m = 80 ;
        type tab = array [1..m, 1..m] of boolean ; { on anticipe sur le chapitre 5 }
        var E, DE, RE, RDE, DRE : tab ;{ Ce sera l'essai du programme } eps : integer ;

procedure afficher (T : tab ; u, v : integer ) ;
{ affiche le tableau T avec le coin haut à gauche en ligne u, colonne v }
        var i , j : integer ;
        begin moveto (u, v); lineto (u+m+2, v) ; lineto (u+m+2, v+m+2);
        lineto (u, v+m+2); lineto (u, v) ;
        for i := 1 to m do for j := 1 to m do
                if T [i, j] then begin moveto (u+j+1, v+i+1); lineto (u+j+1, v+i+1) end
        end;

function max (a, b : integer) : integer ; begin if a < b then max := b else max := a end ;

function min (a, b : integer) : integer ; begin if a < b then min := a else min := b end ;

function interieur (l, c , a : integer ; T : tab) : boolean ; {teste si tous les éléments de T autour de (l = ligne, c =
colonne) dans un "rayon" de a, sont dans T }
        var i, j : integer ; test : boolean ; {test = "tout ce qui a été vu est dans T"}
        begin i := max (l - a, 1); test := true ;
        while (i <= min (l + a, m)) and test do
                begin j := max (c - a, 1);
                while  T[i, j] and (j <= min (c + a, m)) do j := j + 1;
                test := (j > min (c + a, m)) ;
                i := i+1 end ;
        interieur := (i > min (l+a, m)) end;

procedure erosion (T : tab; a : integer ; var ER : tab); {construit TR à partir de T}
        var i, j : integer ;
        begin for i := 1 to m do for j := 1 to m do ER [i, j] := interieur (i, j , a, T) end;

function adherent (l, c , a : integer ; T : tab) : boolean ; { teste si au moins un élément autour de (l = ligne, c =
colonne) dans un "rayon" de a, est dans T }
        var i, j : integer ; test : boolean ;
        begin i := max (l-a, 1);
        repeat   j := max (c-a, 1);
                repeat test := T [i, j] ; j := j + 1 until test or (j > min (c+a, m)) ; i := i + 1
        until test or (i > min (l + a, m)) ;
        adherent := test end;
```

```
procedure dilatation (T : tab; a : integer ; var DL : tab); {construit DL à partir de T}
                var i, j : integer ;
                begin for i := 1 to m do for j := 1 to m do DL[i, j] := adherent (i, j , a, T) end;

procedure creation (var T : tab) ;
                {pour le premier ensemble E, on choisit les points (i, j) où une sinusoïde en 1 / ij, est positive}
                var i, j : integer;
                begin for i := 1 to m do for j := 1 to m do T[i, j] := (0 < sin(9*m*m/(i*j)) )
                {et si création aléatoire : T[i, j] := (2 < 3*(1-abs(i/m-0.5)-abs(j/m-0.5)) + random) }
                end;

begin           eps := 1; creation (E) ;        afficher (E, 30, 70);
                erosion (E, eps, RE);           afficher (RE, 170, 20);
                dilatation (RE, eps, DRE);      afficher (DRE, 300, 20);
                dilatation (E, eps, DE);        afficher (DE, 170, 130);
                erosion (DE, eps, RDE);         afficher (RDE, 300, 130)  end.
```

E



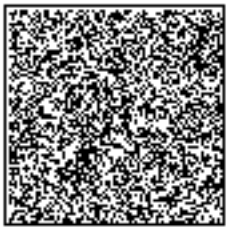
Ero (E)

Dil (Ero (E))

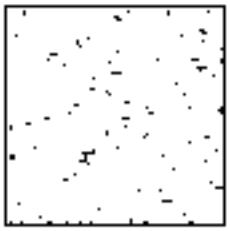
Dil (E)

Ero (Dil (E))

Pour ne pas avoir à rentrer une image point par point, on peut aussi en créer une aléatoirement.



E

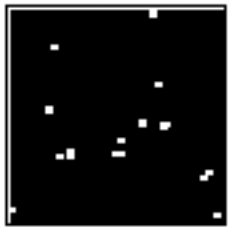


Ero (E)

Dil (Ero (E))

Dil (E)

Ero (Dil (E))



Naturellement il faut expérimenter sur une véritable image (bruitée) pour pouvoir juger. Ici epsilon était fixé à 1, ce qui signifie qu'au cours d'une transformation, pour chaque point, 9 appartenances à l'image de points voisins ont été testés.

3-16° Méthode d'Euler pour les systèmes différentiels du type :

x et y sont fonctions de la variable t, et leurs dérivées vérifient $\begin{cases} x' = f(x, y) \\ y' = g(x, y) \end{cases}$

En tout point M_0 du plan passe une solution courbe paramétrée x(t), y(t), (une trajectoire), le principe de la méthode d'Euler est de confondre cette trajectoire avec la tangente en M_0 sur une petite longueur ds. On trace donc un petit segment M_0M_1 en calculant les coordonnées du point M_1 : $x_1 = x_0 + dt*f(x_0, y_0)$ $y_1 = y_0 + dt*g(x_0, y_0)$

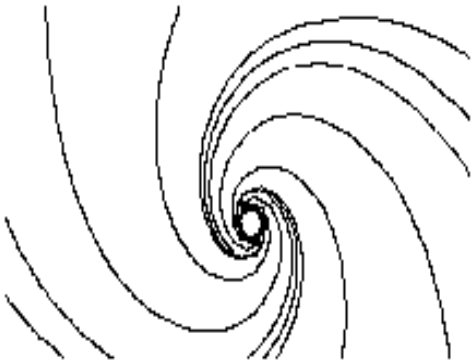
Si on tient à avoir des segments de part et d'autres, on changera le signe de dt, d'où le paramètre s (signe) dans la procédure Euler.

Si on désire que le segment M_0M_1 ait toujours la même longueur ds, alors il suffit d'après le théorème de Pythagorre d'avoir $ds^2 = dx^2 + dy^2$, soit, de prendre une variation de la variable:

$$dt = \frac{ds}{\sqrt{f^2(x_0, y_0) + g^2(x_0, y_0)}}$$

La mise au point du programme demande encore que les trajectoires soient inscrites dans une fenêtre $A \le x \le B$, $C \le y \le D$ que le nombre d'applications de la méthode d'Euler ne soit pas trop grand (on s'est fixé ici 50 itérations au maximum), et que l'on ne soit pas au voisinage d'un point singulier |x'| + |y'| < epsilon.

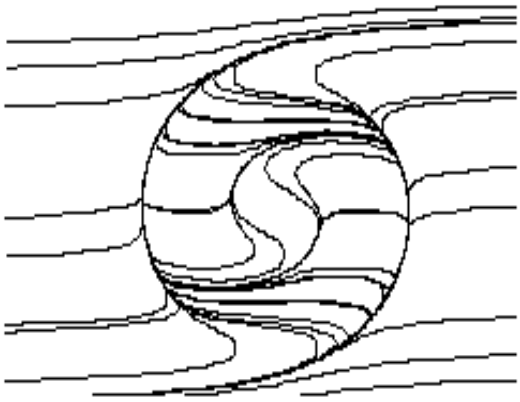
Exemples de trajectoires



x' = x + y et y' = -2x + y

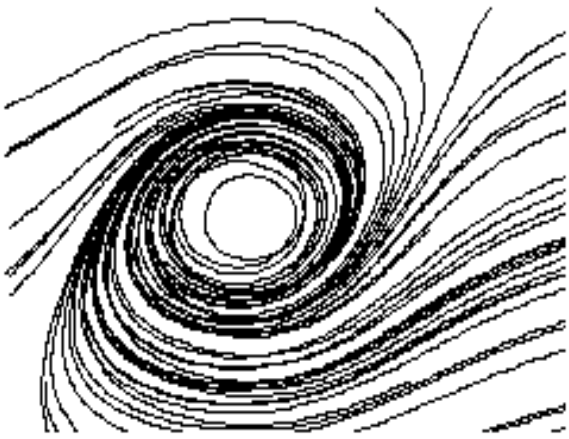
x' = y(y-1)(y+1) y' = sin(x+y)

x' = (x-cos y)(y-cos x) et y' = sin x

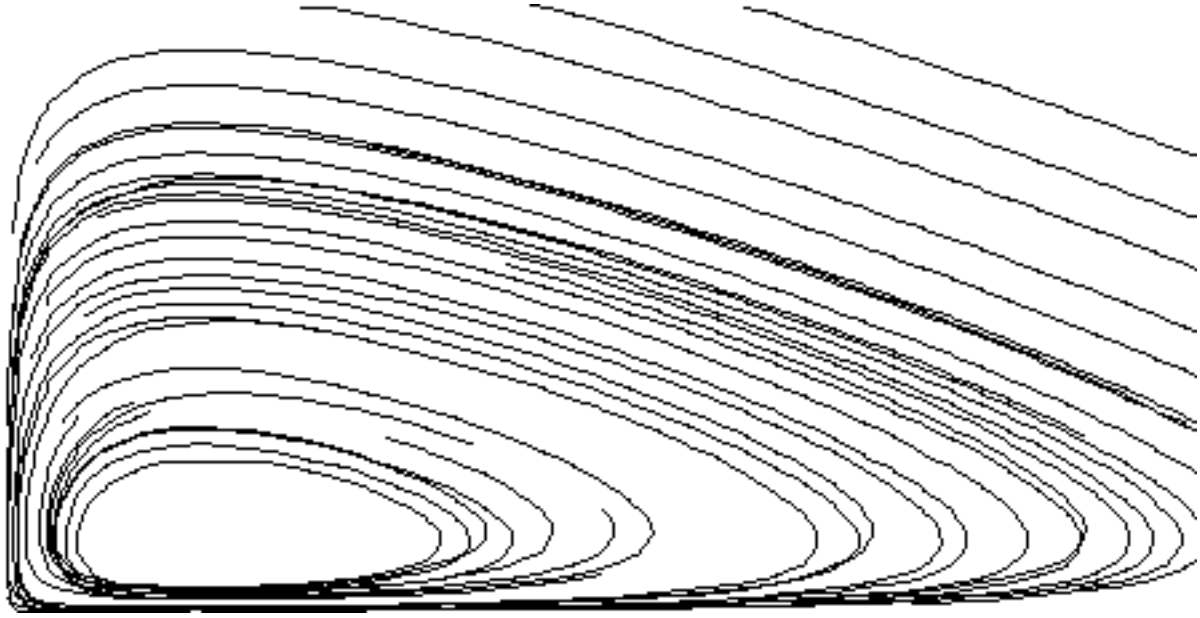


x' = $(x^2+y^2-1)(x^2+y^2-9)$ et y' = x^2+y^2-4

```
program systdiff; { turbo-pascal sur mac }
uses memtypes, quickdraw ;
const A = -2; B = 3; C = -3; D = 2; eps = 0.05;

function f (x, y : real) : real; begin f := y-x*x*x/3 end; { figure ci-dessous }

function g (x, y : real) : real; begin g := -x end;

procedure segment (x, y : real); { Si on donne x entre A et B, et y entre C et D alors les coordonnées sur l'écran
sont U = 510*(X-A)/(B-A) et V = 340*(Y-D)/(C-D)  }
                begin  lineto ( 510*(x-A) div (B-A) , 340*(y-D) div (c-d)) end;

procedure euler (S : integer; var err : boolean; x0, y0 : real;var x1, y1 : real);
{calcule un point M1 voisin du point M0 donné, S=±1 est le sens}
                var dx, dy, dt : real;
begin err := false; dx := f (x0, y0); dy := g (x0, y0);
if abs(dx) + abs(dy) < eps
                then err := true
                else begin dt := ds/sqrt(dx*dx+dy*dy); x1 := x0 + S*dt*dx; y1 := y0 + S*dt*dy end
end ;

procedure trajectoire (x,y:real);
{dessine un morceau de la trajectoire passant en x,y}
                var N, S : integer; x0, y0, x1, y1 : real; message : boolean;
begin   for S := 0 to 1 do {deux morceaux à gauche et à droite}
                begin x0 := x; y0 := y; N := 1;
                moveto ( 510*(x-A) div (B-A) , 340*(y-D) div (C-D));
                repeat   euler (2*S-1,message, x0, y0, x1, y1);
                        if not (message) then begin segment (x1, y1); N := N+1; x0 := x1; y0 := y1 end
                until message or (N>50) or (x1<A) or (x1>B) or (y1<C) or (y1>D)
                end
end;

procedure dessin ; { On distribue les points de passage de trajectoires en les disposant régulièrement sur des
cercles centrés dans l'écran, toutes sortes d'autres choix sont possibles. }
                var i, j, diviseur : integer;
begin
diviseur := 10 ;
for j:= 2 to 5 do for i:= 1 to diviseur do
                trajectoire((B+A)/2 + (B-A)*cos(2*pi*i/diviseur)/j, (C+D)/2 + (D-C)*sin(2*pi*i/diviseur)/j )
end;

var ds : real;
begin {le programme} ds := (B-A)/50; dessin end.
```



Et voilà le résultat pour f(x, y) = y - x*x*x/3 et g(x, y) = -x



Exemple : système différentiel $x' = x - xy$ et $y' = xy - y$ sur $[0, 8]^2$
On pourra résoudre également des équations différentielles polaires $\rho' = f(\rho, \theta)$, ainsi $\rho' = \rho\sin(\rho\theta)$, $\rho' = \tan(8\text{atan}(\sin\theta))$ ou $\rho' = \rho\tan(5\text{atan}(\sin^2\theta))$ donnent de beaux résultats.

3-17° Méthode de Kutta-Runge : Pour le même genre de système différentiel :
x et y sont fonctions de la variable t, et leurs dérivées vérifient $\begin{cases} x' = f(x, y) \\ y' = g(x, y) \end{cases}$
En un point M_0 (x_0, y_0) correspondant à t_0, passe une trajectoire de tangente T_E sur laquelle on définit par la méthode d'Euler le point M_e $(x_0 + dt*f(x_0, y_0), y_0 + dt*g(x_0, y_0))$.
Soit P le milieu du segment $[M_0, M_e]$, en ce point passe une autre trajectoire de tangente T_P, la méthode de Runge - Kutta consiste alors à prendre le point M_r correspondant à t_0+ dt sur la parallèle T_R passant en M_0 à T_P.
Faire un schéma et calculer les coordonnées de M_r, puis construire une procédure en pascal, calculant ce point à partir de M_0 en appelant deux fois f et deux fois g : xP = x0 + dt*f(x0,y0)/2 et yP = y0 + dt*g(x0,y0)/2 puis xR = x0 + dt*f(xP,yP) et yR = y0 + dt*g(xP,yP)

3-18° Les lignes équipotentielles pour deux champs de gravitation comme la terre et la lune sont données par $U(x, y) = K(m/\sqrt{[(x - x_f)^2 + y^2]} + m'/\sqrt{[(x - x_{f'})^2 + y^2]})$ constant.
Cela donne le système différentiel $x' = -Ky\ (m/\sqrt{[(x - x_f)^2 + y^2]} + m'/\sqrt{[(x - x_{f'})^2 + y^2]})$ et $y' = K\ (m(x - xf)/\sqrt{[(x - x_f)^2 + y^2]} + m'(x - xf')/\sqrt{[(x - x_{f'})^2 + y^2]})$. En faire des tracés.

3-19° Equations du second ordre, par la même méthode, on pourra traiter des problèmes tels que:

Pendule de Foucault	x" = -kx + eps*sin(lat*y')
	y" = -ky - eps*sin(lat*x')

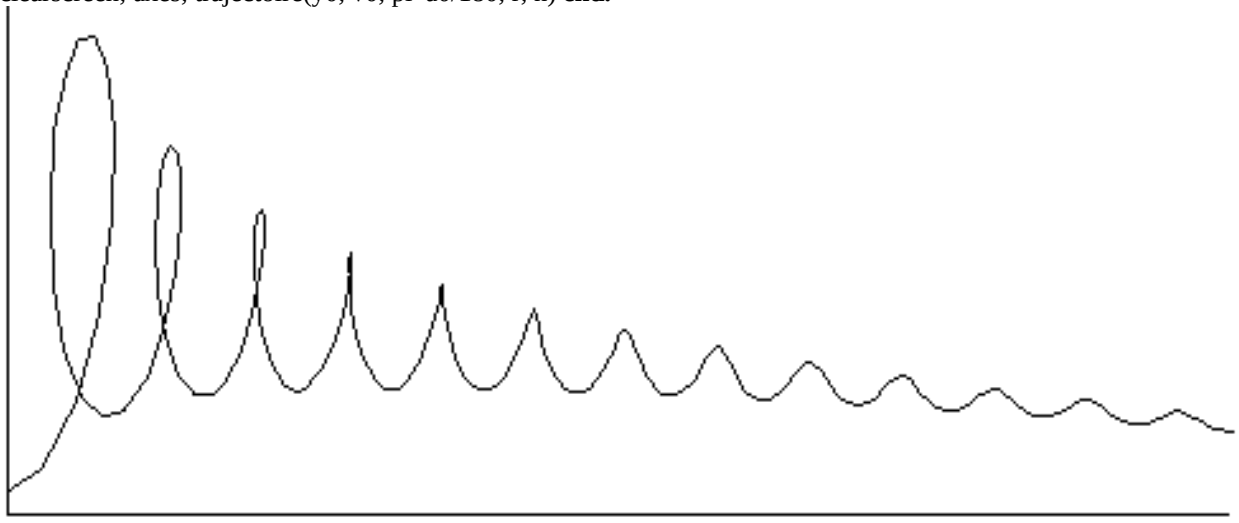
où latitude en radian = $\pi/4$, k = 1, eps = 0.05 et initialement x0 = 0,5, y0 = 0, x'0 = 0, y'0 = 1

Pendule sphérique	$x'' = -kx/\sqrt{(1-x^2-y^2)}$ + eps sin(lat*y')
	$y'' = -ky/\sqrt{(1-x^2-y^2)}$ - eps sin(lat*x')
Champ en r^2	x" = -krx - qx'
	y" = -kry - qy'
Champ newtonien	$x'' = -kx/r^3$ - qx'
	$y'' = -ky/r^3$ - qy'

3-20° Balle de tennis (problème du professeur Alba)
Pour une balle de tennis ou de golf, on dit qu'elle est "liftée" si sa vitesse de rotation N est positive en étant perpendiculaire au plan de sa trajectoire, elle est "coupée" si N < 0, elle est alors beaucoup plus "longue" et, lorsque N prend de grande valeurs (vers 100 tours à la seconde) la trajectoire peut faire en théorie de curieuses boucles (effet Magnus). Les équations différentielles du second ordre qui déterminent la trajectoire s'écrivent en fonction des caractéristiques de la balle (les deux coefficients seront pris égaux à 0,01) de g = 9,81 et de N.

$$x'' = -\frac{k}{m}M\left(\frac{d}{2}\right)^2 x'\sqrt{x'^2 + y'^2} + \frac{2aM}{m}Ny' \quad \text{et} \quad y'' = -\frac{k}{m}M\left(\frac{d}{2}\right)^2 y'\sqrt{x'^2 + y'^2} - \frac{2aM}{m}Nx' - g$$

```
program magnus; { turbo-pascal sur mac }
uses memtypes, quickdraw ;
const L = 480; H = 270;
var A, B, C, D, n, r, pas, v0, a0, y0 : real;
procedure place (x, y : real);   {place le point de vraies coord x,y}
                begin moveto (L*(x-A) div (B-A),  (H*(y-D) div (C-D)) end;
procedure trace (x, y : real);   {trace le segment vers le point de vraies coord x,y}
                begin lineto (L*(x-A) div (B-A), (H*(y-D) div (C-D)) end;

function f (dx, dy : real) : real; begin f := -0.01*sqrt (sqr (dx) + sqr (dy)) * dx + 0.01*N*dy end;
function g (dx, dy : real) : real; begin g := -0.01*sqrt (sqr (dx) + sqr (dy)) * dy - 0.01*N*dx - 9.81 end;
procedure euler ( pas : real; var x, y, dx, dy : real) ; {trace un segment et modifie le point et ses dérivées}
                begin place (x, y); x := x + pas*dx; y := y + pas*dy;
                dx := dx + pas*f(dx, dy); dy := dy + pas*g(dx, dy); trace(x,y) end;
procedure trajectoire (cote, vitesse, angle, r : real ; var n : real); {trace une courbe avec des données initiales et
toujours abscisse=0, n est la vitesse de rotation initiale, r le coefficient d'amortissement}
                var x, y, dx, dy : real;
                begin x := 0; y := cote; dx := vitesse*cos (angle); dy := vitesse*sin (angle);
                while (0<=x) and (0<= y) and (x < B) and (y < D) do begin euler (pas, x, y, dx, dy); n := n*r end
                end;
begin  writeln ('Lancer d''une balle de tennis ou de golf');
write ('Donnez la cote initiale d''où est lancée la balle en mètres '); readln (y0);
write ('Donnez la vitesse initiale (m/s) '); readln (v0);
write ('Donnez l''angle initial (degrés) '); readln (a0);
write ('Quelle est l''altitude maximale ? '); readln (D); C := 0;
write ('Quelle est la distance maximale ? '); readln (B); A := 0; pas := (B-A)/1000;
write ('Rotation initiale en nb de tours/s ? '); readln (n);
write ('Coefficient d''amortissement de cette rotation (ex. 1 ou 0,9) ? '); readln (r);
clearscreen; axes; trajectoire(y0, v0, pi*a0/180, r, n) end.
```



Un résultat pour une balle tirée de 1m de haut à 10° de l'horizontale à 30 m/s et N=-150 sur une distance de 200 m. (r = 1)

3-21° L'évolution de populations telles que les pucerons et les coccinelles peut être modèlisée de la façon suivante :
Soit P l'effectif des prédateurs et V celui des victimes, partant des équations dV/dt = aV -bP et dP/dt = cP - dP où a est le taux de croissance des proies en l'absence de prédateurs, b est le nombre de proies capturées par prédateur et par unité de temps, c le taux de conversion de proies en prédateurs et d le taux de mortalité des prédateurs par manque de proies.
On dispose du modèle de Lokta-Volterra pour lequel a est constant, b = b'V, c est négatif et constant, d = b'd'V, alors pour des valeurs données de a, b', c et d' les trajectoires convergent vers un point P = a/b' et V = -c/b'd'.
Le modèle à densité et satiété dépendant est : dV/dt = r(1-V/k)V - c(1- exp(-aV/c))VP
dP/dt = -mP + bc(1- exp(-aV/c))VP

3-22° Dans la théorie générale de la dynamique des systèmes un système linéaire du premier ordre a un équilibre stable si ses valeurs propres ont leurs parties réelles toutes négatives (si au moins une est positive ou nulle : équilibre instable) sinon oscillations. Pour un organisme on définit une règle de construction (anabolisme) et une de destruction (catabolisme) proportionnelle au volume soit à la masse x. On aura donc $x' = ax^{\alpha} - bx$ où α est le degré de métabolisme (la règle de surface est la proportionnalité de l'anabolisme à la surface donc $\alpha = 2/3$). Programmer cette équation dont les solutions sont

$$x^{1-\alpha} = x_0^{1-\alpha}\, e^{-(1-\alpha)bt} + a(1-e^{-(1-\alpha)bt})/b$$

3-23° Loi de verhulst. Une population d'effectif x_n au moment n, a un taux de croissance r - cx_n, c'est à dire $x_{n+1} = (1+r)x_n - c(x_n)^2$
a) Montrer que si r < 2, il y a convergence vers L vérifiant cL = r
b) Tracer les points d'une telle suite dans ce cas, et montrer que si r > 2 il y a deux valeurs d'adhérence.

3-24° Loi de Mira. Le couple (x_n, y_n) satisfait a une relation de récurrence :
$x_{n+1} = by_n + f(x_n)$ et $y_{n+1} = -x_n + f(x_{n+1})$ avec $f(x) = ax + 2(1 - a)x^2 / (1 + x^2)$ et a = 0,31 et b = 1 représenter la suite de ces points dans [-30, 30]*[-30, 30] pour $x_0 = y_0 = 0$.

3-25° Une suite chaotique. soit une suite p_n vérifiant une relation de récurrence très simple $p_{n+1} = mp_n.(1 - p_n)$ Avec $p_0 = 0.1$ ou 0.4 par exemple, représenter p_n en abscisse et le nombre de générations en ordonnées logarithmiques. On observera alors que cette suite converge pour m < 3, puis qu'elle est alternée, puis vers m = 3.5 elle est alternée sur 4 valeurs d'adhérences, enfin vers m = 3.6, elle se met à parcourir très rapidement 4, 8, 16 ... valeurs d'adhérences, ce qui produit le dessin d'un bel arbre binaire.

```
program chaos ;
        uses memtypes, quickdraw;
        const A = 0.1; B = 1; C = - 0.1; D = 7 ; L = 480; H = 270;

procedure place (x, y :real);{place le point de vraies coord x,y}
        var u, v : integer;
        begin u := round (L*(x-A)/(B-A)); v := round (H*(y-D)/(C-D)); moveto (u, v); lineto (u, v) end;

procedure suite (ni : integer; m : real); {ni est le nombre d'itérations}
        var n : integer; p : real;
        begin p := 0.1; place (p, 0);
        for n := 1 to ni do begin p := m*p*(1-p); place (p, ln (n)) end; end;

begin suite (500 , 3.7) end.
```

3-26° Courbes de Bezier. Etant donnés des points de contrôles ou pôles P_0, P_1, ... P_n, l'idée est de construire une courbe qui soit "attirée" par ces pôles, sans nécessairement y passer. (Le problème de trouver la courbe passant vraiment par des points est résolu avec les polynôme de Lagrange) L'application évidente étant la possibilité en CAO de créér des formes à partir de points que l'utilisateur pourra déplacer à son gré dans un but technique ou artistique.
Ce problème a reçu une solution sous la forme de courbe de Bezier associée aux points P_0, P_1 .... P_n : c'est la courbe définie grâce au paramètre t par :

$\overrightarrow{OM}(t) = \sum_{k=0}^{n} B_{k,n}(t).\overrightarrow{OP_k}$ où $B_{k,n}(t)= C_n^k t^k (1-t)^{n-k}$ sont les polynômes de Bernstein

On vérifie que la somme des poids vaut 1, que cette courbe passe en M_0 et en M_n, elle est tangente en M_0 à M_0M_1, et en M_n à $M_{n-1}M_n$, par ailleurs, si n = 2, M_1 a une tangente parallèle à M_0M_2.
Construction récurrente, par un jeu de factorisations, on peut montrer que :

$\overrightarrow{OM}(t) = B_{0,n-1}(t)[(1-t)\overrightarrow{OP_0} + t\overrightarrow{OP_1}] + B_{1,n-1}(t)[(1-t)\overrightarrow{OP_1} + t\overrightarrow{OP_2}] + \ldots\ldots$

$+ B_{n-1,n-1}(t)[(1-t)\overrightarrow{OP_{n-1}} + t\overrightarrow{OP_n}]$

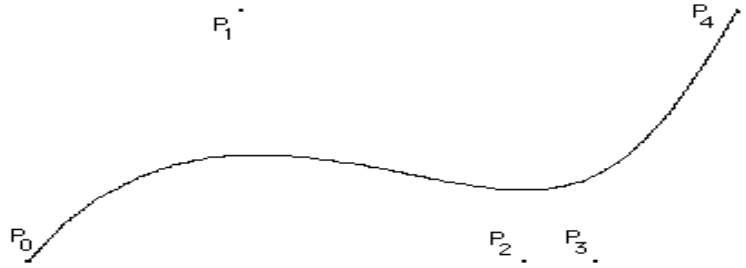
On en déduit le résultat suivant que le point M_t relatif aux n+1 points de départ, est le même que celui relatif à la courbe de Bezier déterminée par les n points P' tels que:

$\overrightarrow{OP'_k}(t) = (1-t)\overrightarrow{OP_k} + t\overrightarrow{OP_{k+1}}$

En réitérant cette propriété, on arrive à déduire M_t comme relatif à deux seuls points, puis pour finir, à un seul, en déduire un algorithme de tracé (algorithme de De Casteljau).

Entrée des points P_0, ... , P_n
Pour t allant de 0 à 1 de h en h
    Pour i allant de 0 à n faire $M_i \leftarrow P_i$ {recopie des points initiaux}
    Pour i allant de 1 à n
        Pour j allant de 0 à n-i faire $M_j \leftarrow (1-t)M_j + tM_{j+1}$
    Tracer M_0

Ainsi, en débutant, M_0, ... ,M_n, sont remplacés par d'autres points toujours apelés M_0, ..., M_{n-1}, pour i = 1, puis pour i = 2, ils le sont à leur tour par M_0, ... ,M_{n-2}. Ainsi arrivé à i = n les deux points M_0, M_1 sont remplaces par un M_0 unique qui est tracé. Cet algorithme est de complexité analogue, mais il permet d'épargner le calcul des polynômes de Bernstein.

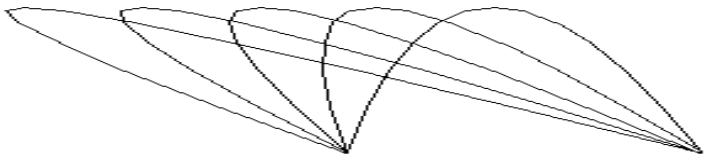



```
program bezier;
uses memtypes, quickdraw;                          {turbo-pascal sur macintosh}
const A = 0 ; B = 10 ; C = 0 ; D = 10 ; M = 10 ;   {tout à fait conventionnel}
type matrice = array [0..M, 1..2] of real;
var nb : integer ; poles : matrice;
function col (X : real) : integer ;  begin col:= 480 * (X-A) div (B-A))end;
function lgn (Y : real) : integer;  begin lgn := (270 * (Y-D) div (C-D) end;
procedure points (N : integer ; P: matrice);       {On place les N points de la matrice P sur l'écran}
        var i : integer;
        procedure grospoint (l, c : integer) ;  { trace un petit carré}
        begin moveto (l-1, c-1); lineto (l-1, c+1); lineto (l+1, c+1); lineto (l+1, c-1); lineto (l-1, c-1) end;
        begin for i := 0 to N do grospoint (col (P [i, 1]), lgn (P [i, 2])) end;
```

```
procedure trace (N : integer ; P : matrice);        {On reçoit N+1 points rangés dans la matrice P}
        var M : matrice; t, h : real; i, j : integer;
        begin t := 0; h := 1 / (6*N) ; moveto (col (P[0, 1]), lgn (P[0, 2]));
        repeat t := t + h ;
                for i := 0 to N do begin M[i, 1] := P[i, 1] ; M[i, 2] := P[i, 2] end;
                {On recopie les points initiaux dans M afin de les modifier}
                for i := 1 to N do for j := 0 to N-i do
                begin M[j, 1]:=(1-t)*M[j, 1]+ t*M[j+1, 1]; M[j, 2]:=(1-t)*M[j, 2]+ t*M[j+1, 2] end;
                lineto (col (M[0, 1]) , lgn (M[0, 2]))
        until t >= 1 end;
procedure entree (var N : integer ; var P : matrice);
        var i : integer;
        begin clearscreen; write('Combien de points allez vous donner ?  '); readln (N); N := N -1;
        for i := 0 to N do          begin write ('abscisse du point n° ', i ,' = '); read (P[i, 1]);
                                    write ('  ordonnée = '); readln (P[i, 2]) end;
        end;
begin entree (nb, poles); clearscreen; points (nb, poles); trace (nb, poles) end.
```

On peut ensuite concevoir toutes sorte de procedures :

```
procedure demo1; {On fixe un point de départ et un point d'arrivée}
        var i : integer;
        begin  poles [0, 1] := 7; poles [0, 2] := 0; poles [2, 1] := 10; poles[2, 2] := 0;
        for i := 0 to 4 do {On fait varier le point intermédiaire}
                begin poles [1, 1]:=2*i; poles [1, 2] := 10; points (2, poles); trace (2, poles) end;
        end;
```



```
procedure demo2; {On fixe un point de départ et un point d'arrivée. On fait varier deux points intermédiaires,
l'un monte, tandis que celui de droite descend}
        var i : integer;
        begin  poles [0, 1] := 0; poles [0, 2] := 0; poles [3, 1] := 10; poles [3, 2] := 3;
        for i := 0 to 10 do
                begin poles [1, 1] := 2; poles [1, 2] := i; poles [2, 1] := 3+i/2; poles [2, 2] := 10-i;
        points (3, poles); trace (3, poles) end end;
```

3-27° Les courbes Splines. En deux mots, c'est une généralisation de ce qui précede, si P_0 .... P_n sont les n + 1 points de contrôle, la courbe "spline" est donnée par :

$$\overrightarrow{OM}(t) = \sum_{i=0}^{n} \overrightarrow{OP}_i N_{i,j}(t) \text{ pour } 0 \le t \le n$$

Formule dans laquelle est choisie un degré i (généralement 2 ou 3 en CAO), et une subdivision (vecteur des noeuds) $t_0 = 0 \le t_1 \le t_2 \le ...$ pour le paramètre t, dans laquelle on a des entiers consécutifs où seuls le premier 0 et le dernier peuvent être répétés m fois. Le plus souvent on prend m = j + 1, car en ce cas la courbe passera aux points extrêmes. Chaque $N_{i,j}$ est un polynôme (de Riesenfield) de degré j calculé par :

$$N_{i,0}(t) = \text{si } t_i \le t \le t_{i+1} \text{ alors 1 sinon 0}$$

$$N_{i,j}(t) = \frac{(t - t_i)N_{i,j-1}(t)}{(t_{i+j} - t_i)} + \frac{(t_{i+j+1} - t)N_{i+1,j-1}(t)}{(t_{i+j+1} - t_{i+1})} \quad (\text{avec } \frac{0}{0} = 0)$$

Les transformations sur les points de contrôle restent locales à ces points et leurs voisins, et on peut changer l'ordre j sans changer les points de contrôles. Si pour n + 1 points, le vecteur noeud est formé par n + 1 fois 0 puis n + 1 fois 1, alors on retrouve les courbes de Bezier où les polynômes $N_{i,n}$ sont les polynômes de Bernstein de degré n.
Programmer le calcul des polynômes puis le tracé des splines. Tester avec la spline carrée (ordre 2) pour les points (-1, 0), (-1, 2), (2, 2), (3, 1), (1, 0), et le vecteur noeud (0 0 0 1 2 3 3 3). Vérifier que la courbe est tangente aux milieux des côtés du polygone, sauf aux deux extrémités où elle est tangente à ces deux côtés en leur extrémité.

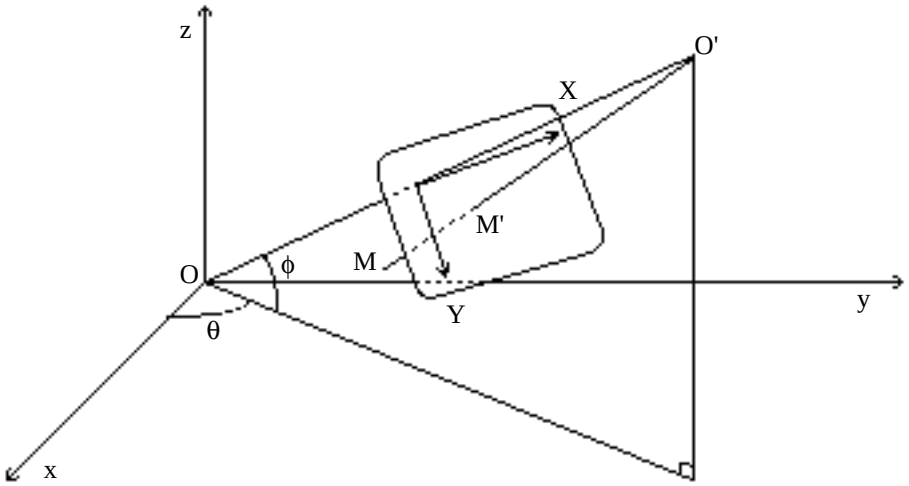
3-28° Représentation de surfaces à partir de points de contrôles, on utilise les formules suivantes pour la données d'une famille de points $P_{i,j}$ déjà en quadrillage.

Surface de Bézier : $$\overrightarrow{OM}(u, v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \overrightarrow{OP}_{i,j} B_{i,n}(u) B_{j,m}(v) \text{ pour } 0 \le u \le n \text{ et } 0 \le v \le m$$

Surface Spline : $$\overrightarrow{OM}(u, v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \overrightarrow{OP}_{i,j} N_{i,k}(u) N_{j,l}(v) \text{ pour } 0 \le u \le n \text{ et } 0 \le v \le m$$

3-29° Elimination des parties cachées par l'algorithme de Hugh
On veut dessiner des courbes d'avant en arrière (au contraire de l'algorithme du peintre) en conservant pour chaque $0 \le X \le L$ les ordonnées bas(X) et haut(X) des points d'abscisse X déjà placés dans deux tableaux.
Si par la suite, à l'abscisse X on a bas(X) < Y < haut(X), alors cela signifie que (X, Y) ne doit pas être placé mais que seule la partie visible du segment doit être dessinée.
Si (X_0, Y_0) est la position précédente alors on trace jusqu'au point de coordonnées :
$(X_0 + (X - X_0) [Y_0 - bas(X_0)] / [bas(X) - bas(X_0) - Y + Y_0], [Y_0 bas(X) - Y bas(X_0)] / [bas(X) - bas(X_0) - Y + Y_0])$ si $Y_0 < bas(X_0)$ et si haut(X0) < Y même relations avec "haut".
Si au contraire (X_0, Y_0) n'est pas tracé car $bas(X_0) < Y_0 < haut(Y_0)$ et que Y < bas(X) alors on doit tracer un segment de $(X_0 + (X - X_0) [Y_0 - bas(X_0)] / [bas(X) - bas(X_0) - Y + Y0], [Y_0 bas(X) - Y bas(X0)] / [bas(X) - bas(X_0) - Y + Y_0])$ à (X, Y) et si Y > haut(X) c'est le point analogue avec "haut".
Expérimenter cet algorithme sur le tracé d'une famille de sinusoïdes f(x) = sin (x + k) + k pour k allant de 0 à n fixé.

3-30° Représentation de surfaces, si l'objet à représenter est dans le repère Oxyz, que le point de vue est O' dont les coordonnées sphériques (r, θ, ϕ) et que la fenêtre d'écran est figurée par le plan perpendiculaire à OO' situé à la distance d de O', le but est de définir la transformation M (x, y, z) --> M' (X, Y) dans l'écran.



r permet de donner l'aspect du dessin à l'écran, si r est infini c'est une projection, sinon une perspective.
d permet de régler la taille du dessin sans changer son aspect.
Les formules sont issues de la composition d'une translation de l'origine en O', d'une rotation de - θ autour de O'z, d'une rotation de ϕ + π/2 autour de O'y et d'un passage en repère indirect. Les coordonnées homogènes en facilitent l'écriture puisqu'elles permettent d'écrire une translation en dimension 3 par un produit maticiel en dimension 4. Touver les formules de transformation. On représentera en faisant une double boucle sur x et y, des surfaces telles que z = (sin ρ) / ρ pour un bel effet ou encore z = (sin x)(sin y) / (xy) ou le paraboloïde hyperbolique $z = x^2 - y^2$

$$\begin{bmatrix} X \\ Y \\ Z \\ T \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\sin\phi & \cos\phi & 0 \\ 0 & -\cos\phi & -\sin\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sin\theta & -\cos\theta & 0 & 0 \\ \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -r\cos n\theta\cos\phi \\ 0 & 1 & 0 & -r\sin\theta\cos\phi \\ 0 & 0 & 1 & -r\sin\phi \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix}$$

Ce qui permet d'obtenir en faisant Z = d, dans le cas où r est fini :

$$X = d\,\frac{x\sin\theta - y\cos\theta}{x\cos\theta\cos\phi + y\sin\theta\cos\phi + z\sin\phi - r} \quad \text{et } Y = d\,\frac{z\cos\phi - x\cos\theta\sin\phi - y\sin\theta\sin\phi}{x\cos\theta\cos\phi + y\sin\theta\cos\phi + z\sin\phi - r}$$

et dans le cas d'une projection : $X = y\cos\theta - x\sin\theta$ et $Y = x\cos\theta\sin\phi + y\sin\theta\sin\phi - z\cos\phi$

Remarques : Si par exemple θ = ϕ = 0, l'écran ne coupe qu'un axe, c'est la perspective à un point de fuite.

Deux points de fuites : l'écran coupe deux axes pour par exemple ϕ = 0 et θ non droit.
Trois points de fuites si l'écran coupe les trois axes.
Perspective oblique si l'écran n'est pas perpendiculaire à OO', elle peut être cavalière si cet angle est 45°, les lignes fuyantes ne sont pas raccourcies, et elle peut être "cabinet" pour arccotan (0,5) = 63,4°
La projection orthogonale est dite axonométrique "isométrique" si les trois axes sont modifiés dans le même rapport θ = 45° et ϕ = 35,26°, elle est dimétrique si deux axes seulement, par exemple θ = 22,2° et ϕ = 20,7°