MT94 - Introduction aux mathématiques appliquées

CAHIER D'INTÉGRATION

NGUYEN Thi Thu Uyen

Printemps - 2019

Introduction & Remerciements

Ce rapport est une compilation de l'essentiel des notions que j'ai vu en cours et en TD de MT94 et il s'organise en 6 chapitres. Chaque chapitre contient un court rappel de cours, les présentations de mes travaux accompagnés de quelques codes sources, de quelques graphiques et d'explications.

Je tiens à remercier M. Stéphane Mottelet qui a nous avons accompagné durant tout le semestre dans le cadre de l'UV MT94, et m'a beaucoup aidé à m'accomplir totalement dans mes missions et dans la rédaction de ce rapport.

Table des matières

CHAPITRE 1

Visualisation, Fractales

1. Généralités

1.1. Définition

Une figure fractale est une courbe ou surface de forme irrégulière ou morcelée qui se crée en suivant des règles déterministes ou stochastiques impliquant une homothétie interne. Le terme fractal est un néologisme créé par Benoît Mandelbrot en 1974 à partir de la racine latine « fractus », qui signifie brisé, irrégulier. Dans la théorie de la rugosité développée par Mandelbrot, une fractale désignes des objets dont la structure est invariante par changement d'échelle.

Des formes fractales approximatives sont facilement observables dans la nature. Ces objets ont une structure autosimilaire sur une échelle étendue, mais finie : les nuages, les flocons de neige, les montagnes, les réseaux de rivières, le chou-fleur ou le brocoli, et les vaisseaux sanguins, etc.

Les domaines d'application des fractales sont très nombreux, on peut citer en particulier :

- En biologie, répartition des structures des plantes, bactéries, feuilles, branches d'arbres…
- En médecine, structure des poumons, intestins, battements du cœur
- En électronique, antennes larges bandes des téléphones portables
- En météorologie, nuages, vortex, banquise, vagues scélérates, turbulences, structure de la foudre

En astronomie avec la description des structures de l'univers, cratères sur la Lune, répartition des exoplanètes et des galaxies…

1.2. Quelques propriétés

Propriété 1.2.1 *Quand la fractale est formée de répliques d'elle-même en plus petit, sa dimension fractale peut se calculer comme suit :*

$$D = \frac{ln(n)}{ln(h)}$$

où la fractale de départ est formée de n exemplaires dont la taille a été réduite d'un facteur h (pour homothétie).

Quelques exemples :

- Un côté du flocon de Koch est formé de $n = 4$ exemplaires de lui-même réduit d'un facteur $h = 3$. Sa dimension fractale vaut :

$$D = \frac{\ln(4)}{\ln(3)}$$

- Le triangle de Sierpinski est formé de $n = 3$ exemplaires de lui-même réduit d'un facteur $h = 2$. Sa dimension fractale vaut :

$$D = \frac{\ln(3)}{\ln(2)}$$

Propriété 1.2.2 *Si on obtient une fractale à partir d'une figure vivant dans un espace ambiant de dimension N, on a : $N - 1 < D < N$, avec D étant la dimension de la figure obtenue.*

Définition 1.2.3 *Un IFS est une famille S de N fonctions contractantes $T_i : M \to M$ dans un espace métrique complet M. On définit à partir des T_i une nouvelle fonction T, elle aussi contractante sur l'ensemble des parties compactes de M muni de la distance de Hausdorff, par l'expression :*

$$T(A) = \bigcup_{i=1}^{N} T_i(A).$$

- Les transformations affines du plan, sont de la forme

$$\begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

La matrice $\begin{pmatrix} e \\ f \end{pmatrix}$ est une translation. La matrice $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ est une application linéaire.

- Il existe plusieurs types d'applications linéaires :

  - Lorsque $\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} r\cos\theta & -r\sin\theta \\ r\sin\theta & r\cos\theta \end{pmatrix}$ il s'agit d'une rotation.

  - Lorsque $\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} \lambda 1 & 0 \\ 0 & \lambda 2 \end{pmatrix}$ il s'agit d'une homothétie.

2. Fractales, IFS et « Chaos game »

Dans cette partie, pour les objets fractals que je vais construire, il existe à chaque fois deux manières pratiques de procéder. Elles ne donneront pas les mêmes résultats visuels mais sont toutes les deux intéressantes à expérimenter.

2.1. Méthode déterministe

Dans cette méthode, on va faire appel à la récursivité, on la qualifiera de déterministe par rapport à l'autre.

2.1.1. Triangle de Sierpinski

Algorithme : Un algorithme pour obtenir des approximations arbitrairement proches du triangle de Sierpinski peut s'écrire de la manière récurrente suivante :

1. Commencer à partir d'un triangle quelconque du plan. Le triangle canonique de Sierpinski se construit à partir d'un triangle équilatéral ayant une base parallèle à l'axe des abscisses.
2. Tracer les trois segments qui joignent deux à deux les milieux des côtés du triangle, ce qui délimite 4 nouveaux triangles.
3. Enlever le petit triangle central. Il y a maintenant trois petits triangles qui se touchent deux à deux par un sommet, dont les longueurs des côtés sont la moitié de celles du triangle de départ (obtenue par une homothétie de rapport 1/2), et dont l'aire est divisée par 4.
4. Recommencer à la deuxième étape avec chacun des petits triangles obtenus.

Démarche : Pour cette fractale, j'ai écrit une fonction récursive acceptant en entrée les coordonnées de trois points dans les vecteurs colonnes A, B, C et renvoyant dans triangles une matrice à deux lignes contenant sur chaque colonne les coordonnées (x, y) des points successifs de tous les triangles pleins de domaine intérieur du triangle (A, B, C).

Par exemple, en prenant le triangle (A, B, C) où $A(0,0), B(1,0), C(0.5, \sqrt{3}/2)$, On réalise le programme suivant :

```
function triangles=sierpinski(A, B, C, n)
   if n == 0 then
      triangles = [A,B,C];
   else
      D  =  (A + B) / 2;
      E  =  (A + C) / 2;
      F  =  (B + C) / 2;
      Triangles  =  [sierpinski(C,E,F,n-1), sierpinski(A,E,D,n-1), sierpinski(F,D,B,n-1)];
      end
endfunction

A  =  [0;0];
B  =  [1;0];
C  =  [0.5;sqrt(3)/2];
tri  =  sierpinski(A,B,C,8)

clf
x = matrix(tri(1,:),3,-1);
y = matrix(tri(2,:),3,-1);
plot3d(x,y,0*x);
set(gca(),"view","2d")
```

CODE SOURCE 1.2.1.1 – Triangle de Sierpinski

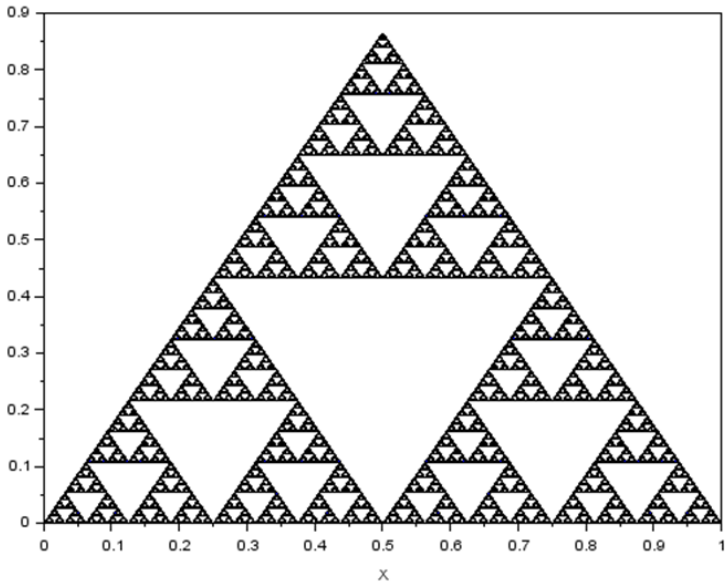


FIGURE 1.2.1.1 – Triangle de Sierpinski

2.1.2. Flocon de Von Kock

On considère le segment $K_0 = AE$ et on définit K_1 par la courbe union des segments AB, BC, CD, DE, où les nouveaux points sont définis par leur vecteur de coordonnées obtenus par :

$$B = A + \vec{u}\ , \qquad C = B + R\vec{u}\ , \qquad D = E - \vec{u}$$

avec $\vec{u} = \frac{1}{3}\overrightarrow{AB}$ et $R = \begin{pmatrix} \cos\frac{\pi}{3} & -\sin\frac{\pi}{3} \\ \sin\frac{\pi}{3} & \cos\frac{\pi}{3} \end{pmatrix}$ la matrice de rotation d'angle $\frac{\pi}{3}$.

On note G la transformation telle que $K_1 = G(K_0)$. Ensuite, pout tout n, K_{n+1} est la courbe obtenue en appliquant G à des segments de K_n.

Un code Scilab associé peut être le suivant :

```
function points=G(A, E, n)
  if (n == 0) then
      points = [A, E];
  else
      U = 1/3 * (E - A);
      B = A + U;
      R = [cos(%pi/3) -sin(%pi/3); sin(%pi/3) cos(%pi/3)];
      C = B + R * U;
      D = E - U;
      points = [G(A,B,n-1) G(B,C,n-1) G(C,D,n-1) G(D,E,n-1)];
  end;
endfunction

A = [0; 0];
E = [1; 0];
result = G(A, E, 5)
clf
plot(result(1, :), result(2, :))
set(gca(), "view", "2d")
```

CODE SOURCE 1.2.1.2 – Flocon de Von Kock

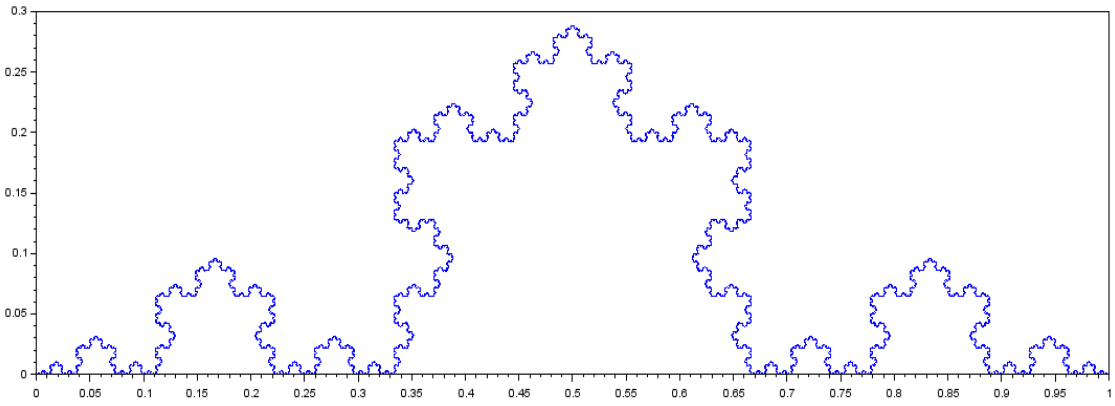


FIGURE 1.2.1.2 – Courbe de Von Kock après 7 itérations

2.2. Méthode utilisant un IFS et le 'Chaos game'

IFS est l'acronyme anglo-saxon pour «Système de Fonctions itérées». Un IFS aléatoire est un IFS où chaque transformation affine w_i est pondérée par une probabilité $0 < p_i < 1$, telle que $\sum_{i=1}^{N} p_i = 1$. On se donne un point $(x_0; y_0)$, on choisit au hasard et avec une probabilité p_k la transformation w_k et on construit $(x_1; y_1) = w_k(x_0; y_0)$ et on recommence. Les itérations successives conduisent au même attracteur K obtenu avec l'IFS déterministe.

2.2.1. Le 'Chaos game'

Le Chaos game ou jeu du chaos a été introduit en 1993 par Michael Barnsley. À l'origine, il s'agissait d'une méthode simple et rapide de création de fractales utilisant un polygone et un point initial choisi au hasard dans ce polygone. La fractale est réalisée, en créant, par itérations successives une séquence de points, partant d'un point initial choisi aléatoirement, pour lesquels chaque point de la séquence est positionné à une fraction donnée de la distance qui sépare le point précédent d'un des sommets du polygone. Ce sommet est choisi aléatoirement à chaque itération. En répétant ce processus un nombre de fois important, et en ignorant les premiers points de la suite, un motif fractal apparait dans la plupart des cas.

On se donne n transformations affines contractantes définies par :

$$T_i(\vec{x}) = M_i \vec{x} + b_i, \quad i = 1, \dots, n$$

telles que l'ensemble fractal étudié $\mathcal{E}$ vérifie la propriété :

$$\mathcal{E} = \bigcup_{i=1}^{n} T_i(\mathcal{E})$$

Puis on considère le processus aléatoire défini par un sous ensemble compact T_i (par exemple un point) et on définit pour n donné X_{n+1} à partir de X_n tel que

$$Prob(X_n + 1 = T_i(X_n)) = T_i$$

2.2.2. Le Triangle de Sierpinski

L'IFS du Triangle de Sierpinski est :

$$M_1 = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad b_1 = \vec{0}$$

$$M_2 = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad b_2 = \begin{pmatrix} 1/2 \\ 0 \end{pmatrix}^T$$

$$M_3 = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad b_3 = \begin{pmatrix} 1/4 \\ \sqrt{3}/4 \end{pmatrix}^T$$

Où $T_n(\vec{x}) = M_n\,\vec{x} + b_n\,,\ \ n = 1, \dots, N$

Voici un code Scilab calculant les termes successifs de la suite (T_n) pour $n = 1..N$ en prenant $T_0 = (0,0)$:

```
N=10000;
x = zeros(2,N);
p = [1 1 1]/3;
P = cumsum(p);
M1 = 0.5 * eye(2,2);
M2 = 0.5 * eye(2,2);
M3 = 0.5 * eye(2,2);
b1 = zeros(2,1); b2 = [0.5;0]; b3 = [1;sqrt(3)]/4;

for i=1:N-1
    r = rand();
    if r < P(1)
        x(:,i+1) = M1 * x(:,i) + b1;
    elseif r < P(2)
        x(:,i+1) = M2 * x(:,i) + b2;
    else
        x(:,i+1) = M3 * x(:,i) + b3;
    end
end
plot(x(1,:),x(2,:),'.','markersize',1)
```

CODE SOURCE 1.2.2.2 – Triangle de Sierpinski

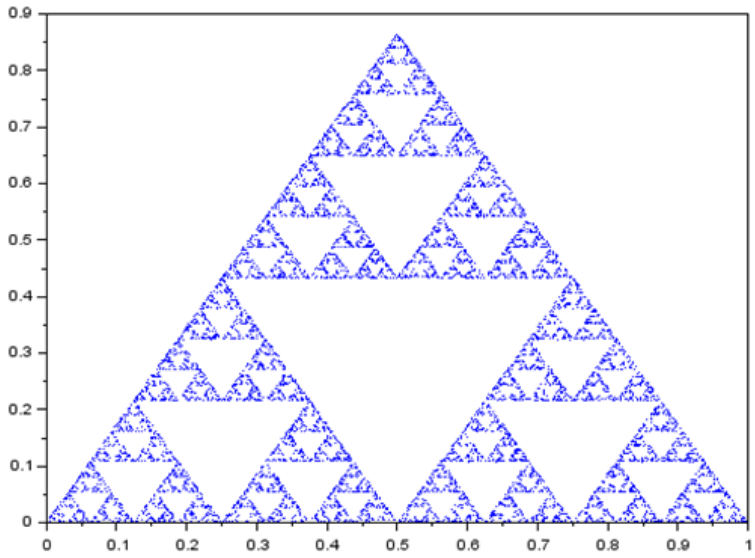


FIGURE 1.2.2.2 - Le Triangle de Sierpinski avec $N = 10\,000$

2.2.3. La courbe de Von Kock

L'IFS de la courbe Von Kock est :

$$M_1 = \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \qquad b_1 = \vec{0}$$

$$M_2 = \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \begin{pmatrix} \cos\frac{\pi}{3} & -\sin\frac{\pi}{3} \\ \sin\frac{\pi}{3} & \cos\frac{\pi}{3} \end{pmatrix} \qquad b_2 = \begin{pmatrix} 1/3 \\ 0 \end{pmatrix}^T$$

$$M_3 = \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \begin{pmatrix} \cos\frac{-\pi}{3} & -\sin\frac{-\pi}{3} \\ \sin\frac{-\pi}{3} & \cos\frac{-\pi}{3} \end{pmatrix} \qquad b_3 = \begin{pmatrix} 1/2 \\ \sqrt{3}/6 \end{pmatrix}^T$$

$$M_4 = \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \qquad b_4 = \begin{pmatrix} 2/3 \\ 0 \end{pmatrix}^T$$

Où $T_n(\vec{x}) = M_n\,\vec{x} + b_n\,,\ \ n = 1, \dots, N$

Voici un code Scilab calculant les termes successifs de la suite (T_n) pour $n = 1..N$:

```
N=5000;
x = zeros(2,N);
p = [1 1 1 1]/4;
P = cumsum(p);

M1 = 1/3 * eye(2,2);
M2 = 1/3 * eye(2,2) * [cos(%pi/3) -sin(%pi/3); sin(%pi/3) cos(%pi/3)];
M3 = 1/3 * eye(2,2) * [cos(-%pi/3) -sin(-%pi/3); sin(-%pi/3) cos(-%pi/3)];
M4 = 1/3 * eye(2,2);

b1 = zeros(2,1);   b2 = [1/3;0];   b3 = [1/2;sqrt(3)/6];   b4 = [2/3;0];

for i=1 : N-1
    r = rand();
    if r < P(1)
        x(:,i+1) = M1 * x(:,i) + b1;
    elseif r < P(2)
        x(:,i+1) = M2 * x(:,i) + b2;
    elseif r < P(3)
        x(:,i+1) = M3 * x(:,i) + b3;
    else
        x(:,i+1) = M4 * x(:,i) + b4;
    end
end
```

CODE SOURCE 1.2.2.3 – La courbe de Von Kock

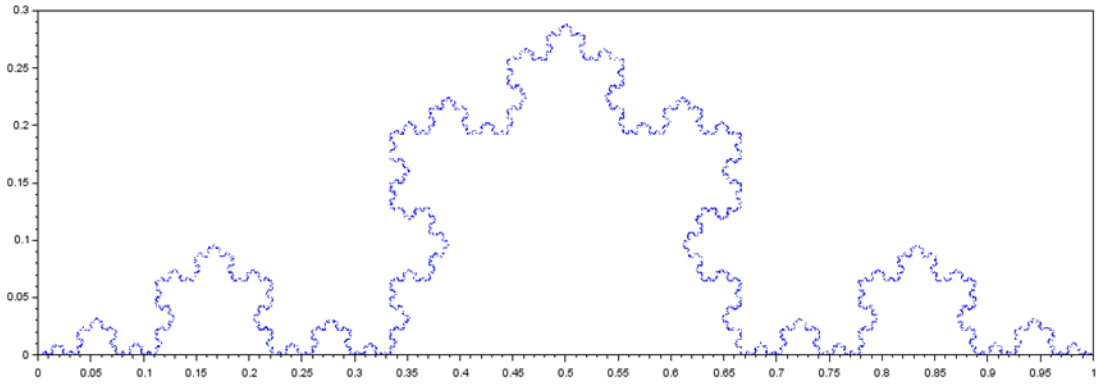


FIGURE 1.2.2.3 – La Courbe de Von Kock

2.2.4. La fougère de Barnsley

Barnsley propose, pour dessiner une fougère, l'IFS suivant :

$$M_1 = \begin{pmatrix} 0.00 & 0.00 \\ 0.00 & 0.16 \end{pmatrix} \qquad b_1 = \begin{pmatrix} 0.00 \\ 0.00 \end{pmatrix}^T$$

$$M_2 = \begin{pmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{pmatrix} \qquad b_2 = \begin{pmatrix} 0.00 \\ 1.6 \end{pmatrix}^T$$

$$M_3 = \begin{pmatrix} 0.20 & -0.26 \\ 0.23 & 0.22 \end{pmatrix} \qquad b_3 = \begin{pmatrix} 0.00 \\ 1.6 \end{pmatrix}^T$$

$$M_4 = \begin{pmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{pmatrix} \qquad b_4 = \begin{pmatrix} 0.00 \\ 0.44 \end{pmatrix}^T$$

Où $T_n(\vec{x}) = M_n \vec{x} + b_n$, $n = 1, \dots, N$, avec les probabilités suivantes :

$$P_1 = 0.01$$

$$P_2 = 0.85$$

$$P_3 = 0.07$$

$$P_4 = 0.07$$

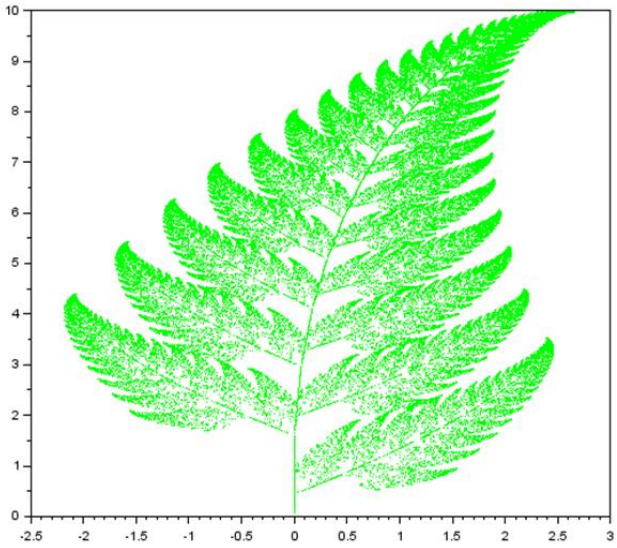


FIGURE 1.2.2.4 - La fougère de Barnsley via un programme récursif à l'itération $n = 10\,000$

3. Ensembles de Mandelbrot et de Julia

La suite (z_n) de $\mathbb{C}$ définie par : $z_{n+1} = z_n^2 + c$, Où z_0 et c sont donnés dans $\mathbb{C}$. On définit les deux ensembles suivants, tout d'abord l'ensemble de Mandelbrot, pour lequel on prend z_0 et

$$\boldsymbol{M} = \{c \in \mathbb{C}, \exists M > 0, \forall n \geq 0, |z_n| < M\},$$

C'est-à-dire que $\boldsymbol{M}$ est l'ensemble des nombres complexes c pour lesquels la suite (z_n) est bornée. De manière duale, on définit l'ensemble de Julia associé à un nombre complexe c par

$$\boldsymbol{J}\,(c) = \{z_0 \in \mathbb{C}, \exists M > 0, \forall n \geq 0, |z_n| < M\},$$

C'est-à-dire que pour c donné, $\boldsymbol{J}\,(c)$ est l'ensemble des valeurs initiales z_0 pour lesquels la suite (z_n) est bornée.

3.1. Ensemble de Mandelbrot

3.1.1. Un résultat important

Dans cette partie, on va clarifier les choses concernant le majorant M et en déduire es informations utiles pour déterminer ces ensembles :

1 . S'il existe p tel que $|z_p| > 2$ et $|z_p| > |c|$ alors $\lim_{n\to\infty} |z_n| = \infty$

2 . En déduire que

(a) Si $|c| > 2$, alors $c \notin \boldsymbol{M}$
(b) Si $|c| \leq 2$, alors s'il existe p tel que $|z_p| > 2$ et $c \notin \boldsymbol{M}$

3.1.2. Visualisation

On a vu plus haut que dès que le module d'un z_n est strictement plus grand que 2, la suite diverge vers l'infini, et donc c est en dehors de l'ensemble de Mandelbrot. Cela nous permet d'arrêter le calcul pour les points ayant un module strictement supérieur à 2 et qui sont donc en dehors de l'ensemble de Mandelbrot. Pour les points de l'ensemble de Mandelbrot, le calcul n'arrivera jamais à terme, donc il doit être arrêté après un certain nombre d'itérations déterminé par le programme. Il en résulte que l'image affichée n'est qu'une approximation du vrai ensemble. Voici le code source pour représenter l'ensemble de Mandelbrot :

```
x = linspace(-2, 1, 400);
y = linspace(-1.5, 1.5, 400);
[X, Y] = meshgrid(x, y);

C = complex(X, Y);
Z = C
G = ones(Z);

for i = 1 : 200
      Z = Z.^2 + C;
      G(abs(Z) < 2) = G(abs(Z) < 2) + 1;
end

c = jetcolormap(201)
set(gcf(),"color_map",c)
plot(x(1), y(1), x($), y($))
Matplot1(G,[x(1) y($) x($) y(1)])
```

CODE SOURCE 1.3.1.2 – Ensemble de Mandelbrot

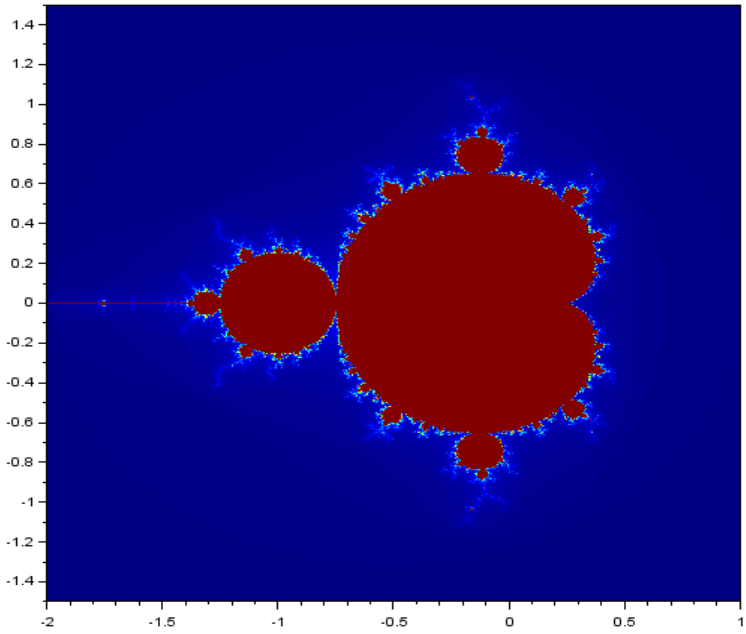


FIGURE 1.3.1.2(1) – Ensemble de Mandelbrot

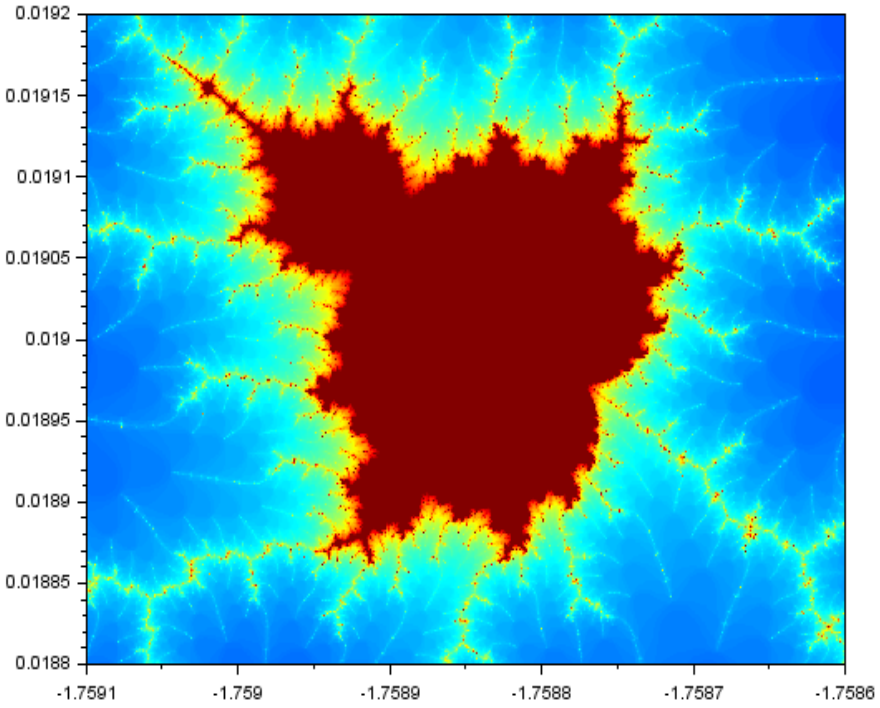


FIGURE 1.3.2.1(2) - Une partie de l'ensemble Mandelbrot se trouvant dans [-1.7591,-1.7586][0.0188,0.0192]*

3.2. Ensemble de Julia

Étant donnés deux nombres complexes, c et z_0, définissons la suite (z_n) par la relation de récurrence:

$$z_n + 1 = z_n^2 + c.$$

Pour une valeur donnée de c, l'ensemble de Julia correspondant est la frontière de l'ensemble des valeurs initiales z_0 pour lesquelles la suite est bornée. La définition des ensembles de Julia est relativement proche de celle de l'ensemble de Mandelbrot qui est l'ensemble de toutes les valeurs de c pour lesquelles la suite (z_n) est bornée, en prenant $z_0 = 0$.

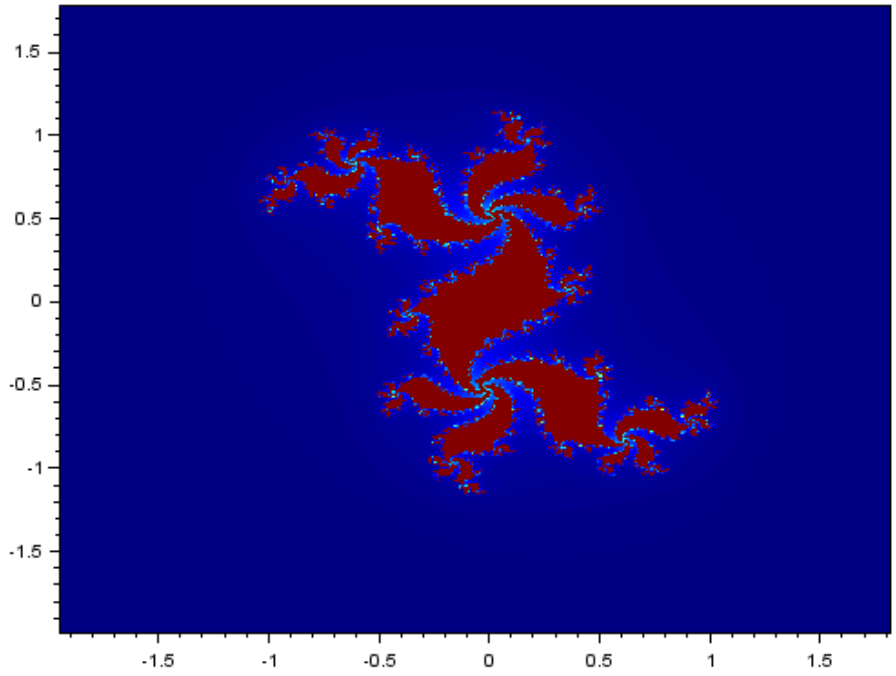


FIGURE 1.3.2(1) - L'ensemble de Julia avec $c = 0.3 + 0.5i$

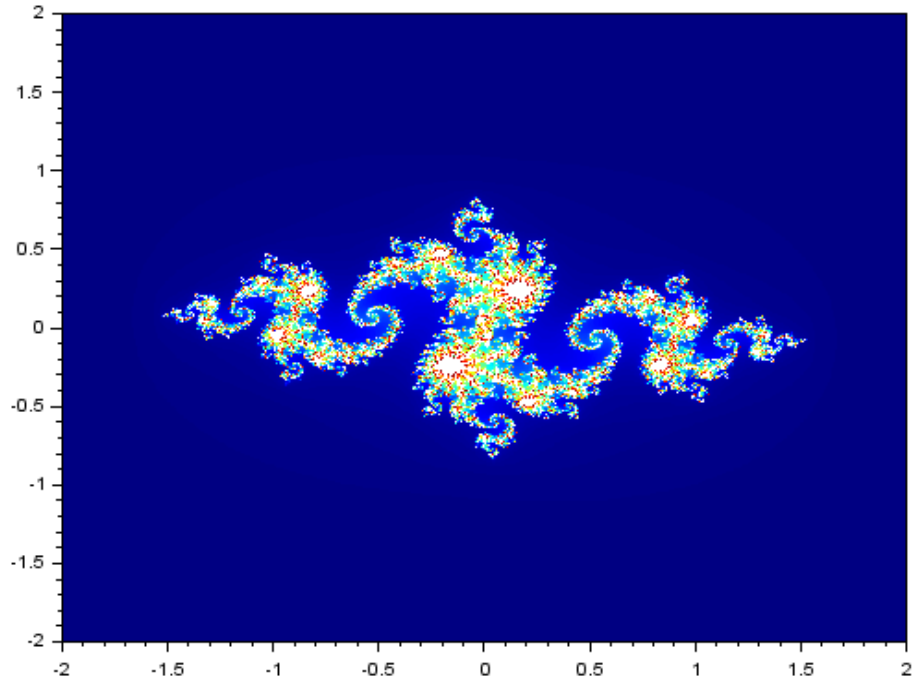


FIGURE 1.3.2(2) - L'ensemble de Julia avec $c = -0.8 + 0.156i$

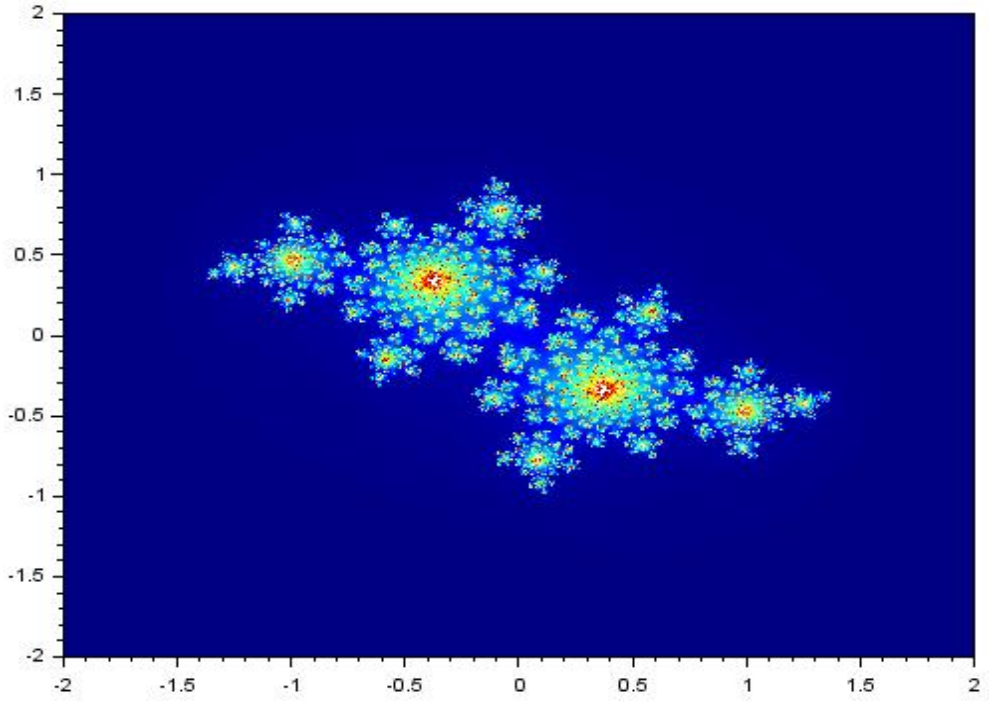


FIGURE 1.3.2(3) - L'ensemble de Julia avec $c = -0.4 + 0.6i$

3.3. Lien entre l'ensemble de Mandelbrot et de Julia

L'ensemble de Mandelbrot $\boldsymbol{M}$ peut être défini comme l'ensemble des complexes c pour lesquels l'ensemble de Julia correspondant, $\boldsymbol{J}(\mathrm{c})$, est connexe. Mais, plus précisément, on a (à la limite) identité entre $\boldsymbol{J}(\mathrm{c})$ et $\boldsymbol{M}$ au voisinage de c, lorsque c appartient à la frontière de $\boldsymbol{M}$. Ainsi, il y a une correspondance proche entre la géométrie de l'ensemble de Mandelbrot à un point donné et la structure de l'ensemble de Julia lui correspondant. Par exemple, un point est dans l'ensemble de Mandelbrot exactement là où l'ensemble de Julia correspondant est connexe.

CHAPITRE 2

Problèmes non linéaires (I)

1. Rappels de cours

1.1. Objectif

Soit $f : \mathbb{R}_n \rightarrow \mathbb{R}_n$. Trouver $x^* \in \mathbb{R}_n$ tel que $f(x*) = 0$.

Toutes les méthodes que nous exposerons dans cette sous-section construisent une suite x_n qui vérifie :

$$\lim_{n\to\infty}(x_n) = x^* \text{ où } f(x*) = 0.$$

1.2. Méthodes numériques

1.2.1. Le cas où $n = 1$

On suppose que $f : \mathbb{R} \rightarrow \mathbb{R}$ est continue et qu'il existe x^* tel que $f(x*) = 0$, et un intervalle $[a, b]$ tel que $f(a).f(b) < 0$.

1.2.1.1. Dichotomie ou 'bissection'

Idée : Construire deux suite (a_n) et (b_n) telles que :

$$f(a_n).f(b_n) < 0 \text{ et } \lim_{n\to\infty}(a_n - b_n) = 0.$$

Algorithme de Dichotomie : Soient (a_0) et (b_0) telles que $f(a_0).f(b_0) < 0$, $x_0 = \frac{1}{2}(a_0 + b_0)$

Tant que $|f(x_n)| > \varepsilon$

Si $f(a_n).f(x_n) > 0$

$a_{n+1} = x_n$

Sinon

$b_{n+1} = x_n$

Fin

Fin tant que

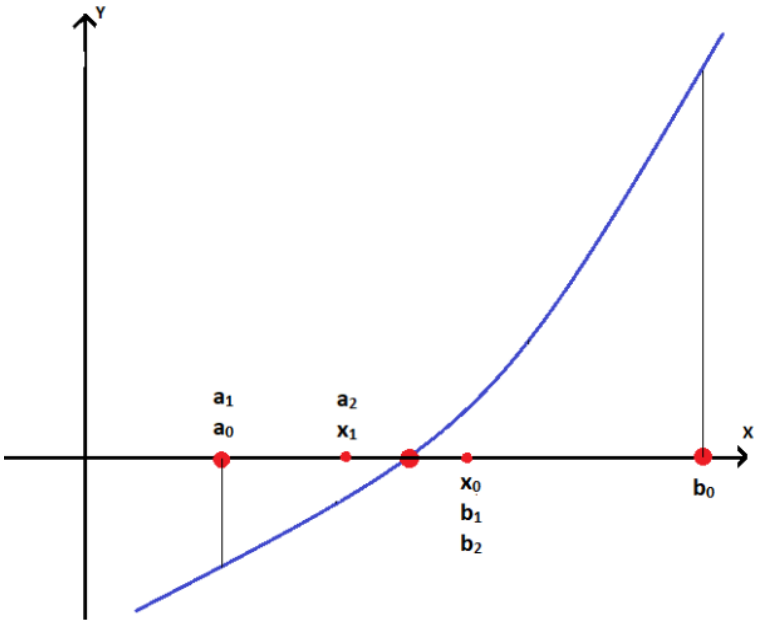


FIGURE 1.2.1.1 - Illustration de la méthode de Dichotomie

<u>Convergence de la méthode</u> :

Comme on a $(b_{n+1} - a_{n+1}) = \frac{1}{2}(b_n - a_n)$, alors $(b_{n+1} - a_{n+1}) = \frac{1}{2^n}(b_0 - a_0)$.

Et comme $|x_n - x^*| \leq \frac{1}{2}(b_n - a_n)$ on a $\forall n \in \mathbb{N}$, $|x_n - x^*| \leq \frac{1}{2^n}(b_0 - a_0)$.

1.2.1.2. Méthode du point fixe

Idée : Trouver $g : \mathbb{R} \to \mathbb{R}$ telle que :

$$f(x) = 0 \Leftrightarrow x = g(x).$$

Théorème : Soit $g : \mathbb{R} \to \mathbb{R}$ dérivable et possédant un point fixe x^* vérifiant $|g(x^*)| < 1$. Alors il existe $[a, b]$ tel que si $x_0 \in [a, b]$ alors la suite (x_n) définie par $x_{n+1} = g(x_n)$ converge vers x^*.

1.2.1.3. Méthode de Newton

On suppose que f est deux fois continûment dérivable et on se place au voisinage de x_0, on peut écrire :

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0 + \partial(x - x_0)) \ , \ \partial \in]0,1[\ .$$

Idée : Définir x_1 comme la solution de l'équation linéaire :

$$f(x_0) + (x - x_0)f'(x_0) = 0,$$

c'est-à-dire, si $f'(x_0) \neq 0$, $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$, et de construire une suite (x_n).

Algorithme de Newton :

x_0 donné

Tant que $|f(x_n)| > \varepsilon$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \ (\Leftrightarrow \ x_{n+1} = g(x_n))$$

Fin tant que

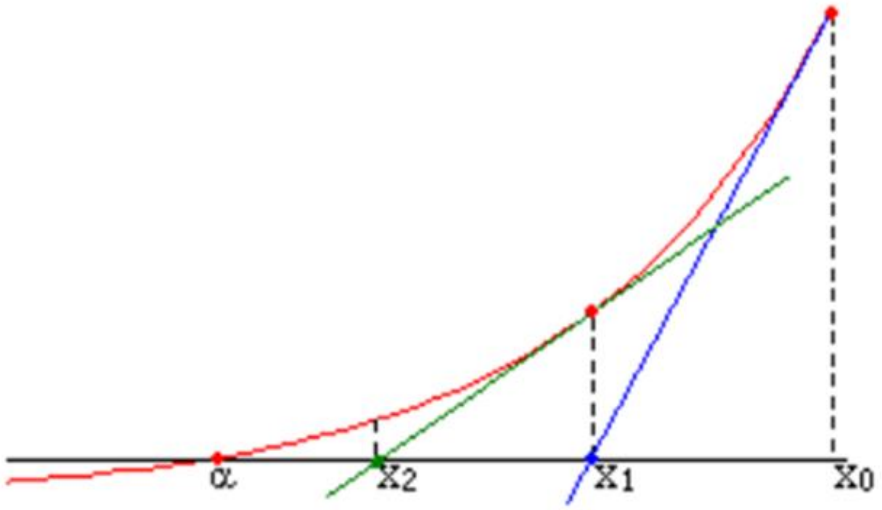


FIGURE 1.2.1.3 - Illustration de la méthode de Newton

Remarque : Comment choisit-on x_0 ?

Mais avant de répondre, voyons que la méthode est une méthode de point fixe très particulière, on a ici

$$g(x) = x - \frac{f(x)}{f'(x)},$$

$$g'(x) = 1 - \frac{\left(f'(x)\right)^2 - f(x)f'(x)}{\left(f'(x)\right)^2},$$

d'où $g'(x^*) = 1 - \frac{\left(f'(x*)\right)^2}{\left(f'(x*)\right)^2} = 0$, donc il existe forcément $[a, b]$ tel que $\forall x[a, b], |g'(x)| < 1$.

Convergence de la méthode : la convergence est quadratique :

$$\exists n \geq 0\ , |x_{n+1} - x*| \leq c|x_n - x*|^2.$$

Si $|x_0 - x*| < \frac{1}{c}$ alors la méthode de Newton converge, et quadratiquement.

1.2.2. Le cas où $n > 1$ (La méthode de Newton)

Idée : est la même que pour $n = 1$. On se donne $x_0 \in \mathbb{R}_n$ on a :

$$f(x_0 + h) = f(x_0) + h\,.J_f(x_0) + |h|\varepsilon(h),$$

et on en déduit l'approximation affine de f en $x_0, i.e.$ la fonction $T(x)$ définie par :

$$T(x) = f(x_0) + (x - x_0)\,.J_f(x_0).$$

Si la matrice $J_f(x_0)$ est inversible, on peut déduire x_1 tel que $T(x_1) = 0$, soit :

$$f(x_0) + (x_1 - x_0)\,.J_f(x_0) = 0\text{ , soit } x_1 = x_0 - (\,J_f(x_0))^{-1}\,f(x_0).$$

Algorithme de la méthode :

$x_0 \in \mathbb{R}_n$ donné

Tant que $|f(x_n)| > \varepsilon$

$x_{n+1} = x_n - (\,J_f(x_n))^{-1}\,f(x_n)$

Fin

2. Résolution d'une équation à une inconnue

Dans cette partie, on cherche à résoudre l'équation :

$$x^2 = 2 \quad (1)$$

Pour y parvenir, on va tour d'abord choisir un intervalle $[a, b]$ dans lequel se situe x^* une seule solution de (1). En l'occurrence, on choisit $a = 1\ et\ b = 2$: on a donc $x^* = \sqrt{2}$. L'équation (1) peut s'écrire de façon équivalente $f(x) = 0$ ou $g(x) = 0$ avec :

$$f(x) = x^2 - 2\,,g(x) = \frac{x+2}{x+1}$$

On cherche à comparer plusieurs méthodes pour trouver x^* :

2.1. Méthode de dichotomie

On choisit $a = 1,\ b = 2$, on a :

$$f(1) = 1^2 - 2 = -1 < 0,$$
$$f(2) = 2^2 - 2 = 2 > 0,$$
$$f(1).f(2) < 0.$$

Sur l'intervalle $[1,2]$, on construit 3 suites $(a_n), (b_n), (x_n)$ de la manière suivante :

```
function y=f(x)
        y = x^2 - 2;
endfunction

a = 1; b = 2;
for k = 1 : 100
      x(k) = (a + b) / 2;
      if f(a) * f(x(k)) < 0
            b = x(k);
      else
            a = x(k);
      end
      if abs(f(x(k))) < 1e-10
           break;
      end
end
```

CODE SOURCE 2.1 – Méthode de dichotomie

2.2. Méthode du point fixe

Dans cette méthode, on trouve une fonction $g(x)$ telle que :

$$f(x) = 0 \Leftrightarrow x = g(x).$$

Par exemple, on prend $g(x) = \frac{x+2}{x+1}$, on a donc :

$$x = g(x) \Leftrightarrow x^2 + x = x + 2 \Leftrightarrow x^2 - 2 = 0 \Leftrightarrow x = \sqrt{2}\ ou\ x = -\sqrt{2}$$

```
function y=f(x)
    y = x^2 - 2;
endfunction

function y=g(x)
    y = (x+2)/(x+1);
endfunction

x = 1.5;
for k = 1 : 100
    if abs(f(x(k))) < 1e-10
        break;
    end
    x(k+1) = g(x(k));
end
```

CODE SOURCE 2.2 – Méthode du point fixe

2.3. Méthode de Newton

On prend $x_1 = 1.5$, on contruit une suite (x_n) avec :

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

```
function y=f(x)
    y = x^2 - 2;
endfunction

function y=fPrime(x)
    y = 2*x;
endfunction

x = 1.5;
for k = 1 : 100
    if abs(f(x(k))) < 1e-10
        break;
    end
    x(k+1) = x(k) -  f(x(k)) / fPrime(x(k));
end
```

CODE SOURCE 2.3 – Méthode de Newton

2.4. Méthode de la sécante

On prend $x_1 = 1.4$ et $x_2 = 1.5$, on contruit une suite (x_n) avec :

$$x_{k+1} = x_k - \frac{f(x_k)}{f(x_k) - f(x_{k-1})}(x_k - x_{k-1})$$

```
function y=f(x)
    y = x^2 - 2;
endfunction

x(1) = 1.4; x(2) = 1.5;
for k = 2 : 100
    if abs(f(x(k))) < 1e-10
        break ;
    end
    tmp = ( f(x(k)) - f(x(k-1)) ) / (x(k) - x(k-1)) ;
    x(k+1) = x(k) - f(x(k)) / tmp;
end
```

CODE SOURCE 2.4 – Méthode de la sécante

2.5. Comparaison des convergences des différentes méthodes

L'ordre de convergence $p \geq 1$ d'une méthode est défini par :

$$\lim_{k \to \infty} \frac{|x_{k+1} - x*|}{|x_k - x*|} = \alpha > 0$$

En utilisant Scilab, nous trouvons expérimentalement que méthode de dichotomie et de point fixe sont d'ordre 1, méthode de Newton est d'ordre 2 et méthode de la sécante est d'ordre de $\frac{1+\sqrt{5}}{2}$. A l'aide de Scilab, j'ai réalisé un graphique consultable permettant de comparer les convergences des différentes méthodes vers la solution de l'équation. Cette comparaison a été effectuée pour un nombre d'itération $n = 10$.

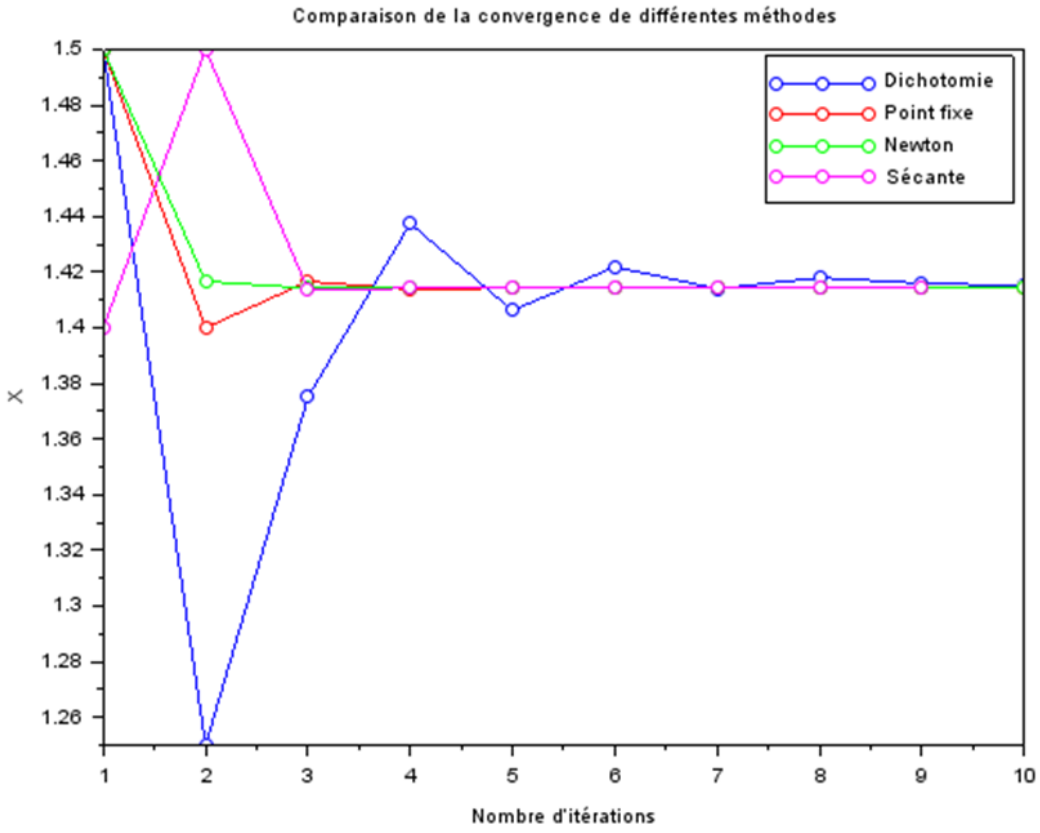


FIGURE 2.5 - Comparaison des convergences des différentes méthodes pour $n = 10$

2.6. La macro *fsolve*

La macro *fsolve* de Scilab permet de résoudre des systèmes d'équations non-linéaire avec une méthode inspirée de la méthode de Newton. On résout $x^2 = 2$ en utilisant la macro fsolve de la manière suivante :

```
function y=f(x)
  y=x^2-2;
endfunction

function d=fprime(x)
  d=2*x;
endfunction

x0 = 1.5;
x = fsolve(x0,f,fprime);
```

CODE SOURCE 2.6 – La macro fsolve

3. Applications

3.1. GPS

Le GPS est un système de positionnement basé sur la connaissance de la distance du récepteur R à trois satellites (situés à des orbites de l'ordre de 28000km).

On suppose que les trois satellites au moment du calcul de distance ont les positions suivantes dans un repère cartésien d'origine le centre de la terre (unité km) :

$$S1 = (-11716.227778, -10118.754628, 21741.083973)$$

$$S2 = (-12082.643974, -20428.242179, 11741.374154)$$

$$S3 = (14373.286650, -10448.439349, 19596.404858)$$

Sachant que les trois distances respectives au récepteur ont été calculées et valent :

$$(d1, d2, d3) = (22163.847742, 21492.777482, 21492.469326)$$

On pose :

$$f(X) = \begin{pmatrix} ||X - S1||^2 - d1 \\ ||X - S2||^2 - d2 \\ ||X - S3||^2 - d3 \end{pmatrix}$$

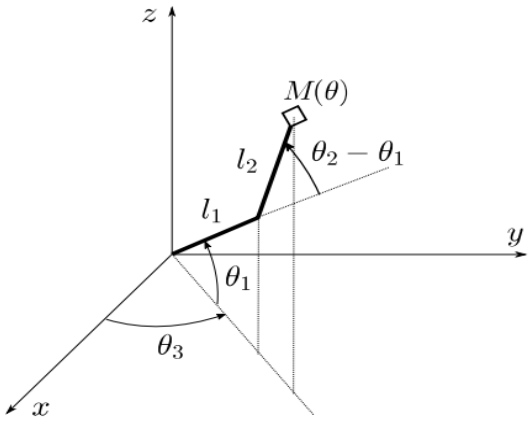
Pour X^* (la position du récepteur R dans le repère associé au centre de la Terre), on a bien : $f(X^*) = 0$.

```
function out=f(R)
  out = [norm(R-S1)^2; norm(R-S2)^2; norm(R-S3)^2] - d.^2;
endfunction

S1 = [-11716.227778;-10118.54628;21741.083973];
S2 = [-12082.643974;-20428.242179;11741.374154];
S3 = [14373.286650;-10448.439349;19596.404858];
d  = [22163.847742;21492.777482;21492.469326];

R0 = zeros(3,1);
[R,v,info] = fsolve(R0,f)
```

CODE SOURCE 3.1 – GPS

Le récepteur R se trouve expérimentalement à 6369.2864 kms du centre de la Terre.

3.2. Cinématique inverse



On considère sur la figure au-dessus un bras robot articulé dans l'espace d'origine O, avec deux segments dans un même plan contenant l'axe (Oz) et pouvant tourner autour de cet axe. Le premier segment de longueur l_1 faisant un angle θ_1 avec $(0, x, y)$ et un deuxieme segment de longueur l_2 faisant un angle $\theta_2 - \theta_1$ avec le premier segment. Le plan contenant le bras fait un angle θ3 avec (O, y, z). Les coordonnées de l'extrémité du bras sont données par :

$$M(\theta) = (cos\theta_3(l_1 cos\theta_1 + l_2 cos\theta_2), sin\theta_3(l_1 sin\theta_1 + l_2 sin\theta_2), l_1 sin\theta_1 + l_2 sin\theta_2).$$

On cherche à déterminant θ tel que $M(\theta) = A$ où A est un point du plan, on a donc les relations suivantes :

$$cos\theta_3(l_1 cos\theta_1 + l_2 cos\theta_2) - x_A = 0$$

$$sin\theta_3(l_1 sin\theta_1 + l_2 sin\theta_2) - y_A = 0$$

$$l_1 sin\theta_1 + l_2 sin\theta_2 - z_A = 0$$

On peut ensuite résoudre la fonction Scilab fsolve :

```
function out=f(t)
    M = [ cos(t(3)) * (l(1) * cos(t(1)) + l(2) * cos(t(2)))
          sin(t(3)) * (l(1) * sin(t(1)) + l(2) * sin(t(2)))
          l(1) * sin(t(1)) + l(2) * sin(t(2)) ];
    out = M - A;
endfunction
```

CODE SOURCE 3.2 – Cinématique inverse

On prendre $l_1 = 1, l_2 = \frac{2}{3}$. Lorsque le point $A(t)$ est défini par une courbe, par exemple lorsqu'il décrit les arêtes d'un cube $A(t) = (0, 1, t)$ avec $t \in [0, 1]$, on utilise la fonction Scilab au-dessus pour représenter les positions successives du bras robot :

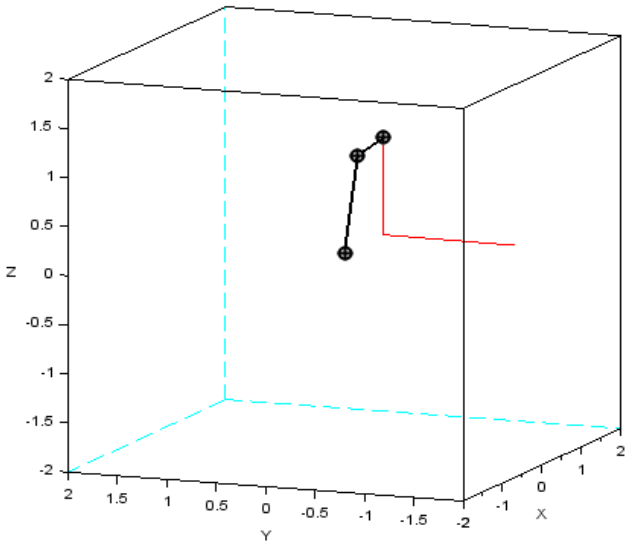


FIGURE 3.2 – Exemple du bras robot

3.3. La fractale de Newton

On cherche donc à résoudre dans $\mathbb{C}$ l'équation :

$$z^3 - 1 = 0$$

Les solutions de cette équation sont bien évidemment :

$$z_1 = 1,$$

$$z_2 = -\frac{1}{2} + \frac{\sqrt{3}}{2} i\,,$$

$$z_3 = -\frac{1}{2} - \frac{\sqrt{3}}{2} i,$$

En appliquant la méthode de Newton, la suite complexe récurrente (z_n) définie par z_0 et

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)} = z_n - \frac{z_n^3 - 1}{3z_n^2} = \frac{2z_n^3 + 1}{3z_n^2} = \frac{2}{3}z_n + \frac{1}{3z_n^2}$$

permet d'approcher les solutions de l'équation. Lorsque l'on choisit un z_0 donné, la suite (z_n) peut converger vers l'une des trois solutions ou ne converger pas du tout. Dans ce dernier cas, z_0 appartient à la fractale de Newton. On considére une discrétisation à N^2 points du rectangle [−1, 1] × [−1, 1], c'est à dire deux matrices X et Y où chaque point i, j du rectangle a pour coordonnées (x_{ij}, y_{ij}) et on considère les nombres complexes correspondants dans la matrice Z où :

$$\forall i, j \in \{1; \ldots; N\},\ z_{ij} = x_{ij} + i * y_{ij}$$

Le but est de construire une matrice M que l'on représentera ensuite avec Matplot. Le terme m_{ij} est défini ainsi : le nombre complexe z_{ij} est utilisé comme condition initiale z_0 pour la méthode de Newton. Si la méthode converge (après un nombre d'itérations fixé) à la précision machine vers l'une des racines on attribue le nombre 1,2 ou 3 à m_{ij}.

```
N = 500;
Eps = 1e-15;
plot([-1 1] , [-1 1] , ".");
[X,Y] = meshgrid(linspace(-1,1,N));
Z = X + %i * Y;
s1 = 1;
s2 = (-1 + %i * sqrt(3)) / 2;
s3 = (-1 - %i * sqrt(3)) / 2;
for  i = 1 : N
    Z = 2/3 * Z + (1/3) * Z.^(-2);
    C = ones(N,N) * color("white");
    C(abs(Z-s1) < eps) = color("green");
    C(abs(Z-s2) < eps) = color("red");
    C(abs(Z-s3) < eps) = color("blue");
    Matplot(C,rect = [-1 -1 1 1]);
end
```

CODE SOURCE 4 – La fractale de Newton

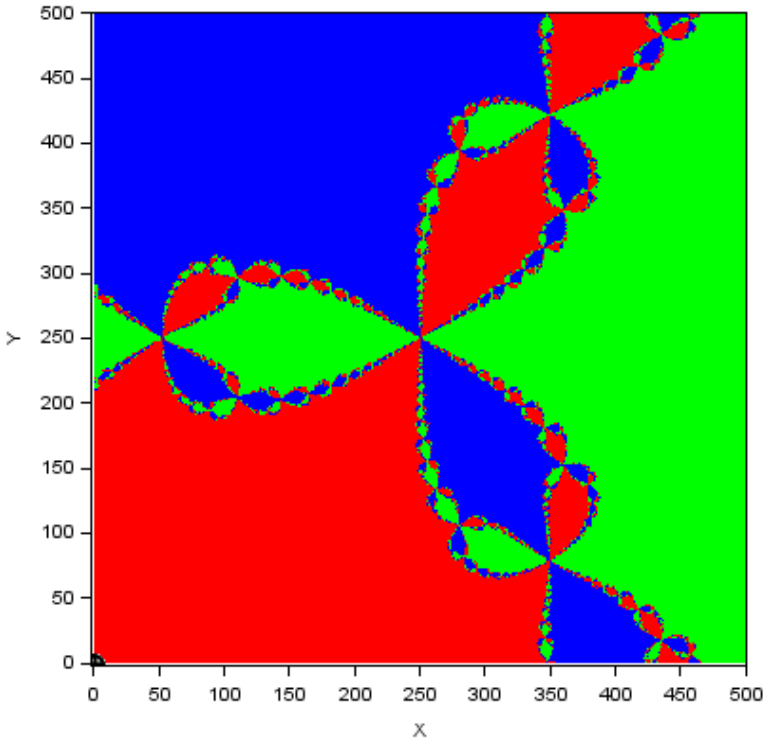


FIGURE 4 – La fractale de Newton

CHAPITRE 3

Equations différentielles ordinaires, systèmes dynamiques

1. Equation différentielles ordinaires

1.1. Présentation des différentes méthodes

1.1.1. Schéma à un pas

On considère la fonction $y : [0,T] \rightarrow \mathbb{R}$ solution d'un problème de Cauchy, c'est-à-dire une équation différentielles munie d'une condition initiale en t = 0 :

$$y' = f(t,y)\,, t \in [0,T] \qquad (1)$$

$$y(0) = y_0 \qquad (2)$$

On suppose que f vérifie les conditions du théorème de Cauchy-Lipschitz, ce qui garantit l'existence et l'unicité de la solution de $(1),(2)$. Pour $N \in \mathbb{N}^*$ on considère une discrétisation uniforme de l'intervalle $[0,T]$ sous la forme d'une suite (t_n) avec $n = 0\,..\,N$ définie par $t_n = nh$ où $h = T/N$. Un schéma à un pas a pour but de construire une suite (y_n) avec $n = 0\,..\,N$ approchant les valeurs de la solution y aux points t_n, c'est-à-dire $(y(t_n))$ avec $n = 0..N$, en considérant l'équation de récurrence :

$$y_{n+1} = y_n + h\phi(t_n, y_n; h)\,, n \in \{0..N-1\}$$

Nous nous intéresserons principalement à quatre schémas :

1. Schéma d'Euler

$$y_{n+1} = y_n + hf(t_n, y_n)$$

2. Schéma d'Euler-Cauchy

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f(t_n + h, y_n + hk_1) \\ y_{n+1} = y_n + \dfrac{h}{2}(k_1 + k_2) \end{cases}$$

3. Schéma du point milieu

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}k_1\right) \\ y_{n+1} = y_n + hk_2 \end{cases}$$

4. Schéma du Runge et Kutta d'ordre 4

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}k_1\right) \\ k_3 = f\left(t_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}k_2\right) \\ k_4 = f\left(t_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}k_3\right) \\ y_{n+1} = y_n + \dfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases}$$

1.1.2. Les méthodes de résolutions utilisées en pratique

On va comparer les schémas que on présenter au-dessus. On considère l'équation différentielle suivante :

$$y' = -ty + t, t \in [0,4] \qquad (4)$$

$$y(0) = 0$$

La solution exacte de cette équation est

$$y = 1 - \exp\left(-\frac{t^2}{2}\right).$$

Voici comment organiser le travail :

- Etape 1 : Ecrire une fonction Scilab calculant le second membre de l'équation différentielle (4).

```
function yprime = f1(t, y)
        yprime = -t * y + t;
endfunction
```

- Etape 2 : Ecrire une fonction Scilab acceptant comme argument y_0, un vecteur contenant (t_n) avec $n = 0\,..\,N$ et l'identificateur de la fonction Scilab calculant f, et renvoyant dans un vecteur y_n pour $n = 0\,..\,N$ données par chaque Schéma :

1. Schéma d'Euler

```
function y = euler(y0, t, f)
    n = length(t);
    y = zeros(1, n);
    h = t(2) - t(1);
    y(1) = y0;
    for i = 1 : n-1
        y(i+1) = y(i) + h * f(t(i), y(i));
    end
endfunction

t = linspace(0, 4, 10);
y0 = 0;
y = euler(y0,t,f1);
```

CODE SOURCE 1.1.2(1) – Schéma d'Euler

2. Schéma d'Euler-Cauchy

```
function y=eulerCauchy(y0, t, f)
    n = length(t);
    y = zeros(1, n);
    h = t(2) - t(1);
    y(1) = y0;
    for i = 1 : n-1
        k1 = f(t(i), y(i));
        k2 = f(t(i) + h, y(i)+h * k1);
        y(i+1) = y(i) + h/2 * (k1+k2);
    end
endfunction


t = linspace(0, 4, 10);
y0 = 0;
plot(t, eulerCauchy(y0, t, f1));
```

CODE SOURCE 1.1.2(2) – Schéma d'Euler-Cauchy

3. Schéma du point milieu

```
function y = pointMilieu(y0, t, f)
    n = length(t);
    y = zeros(1, n);
    h = t(2) - t(1);
    y(1) = y0;
    for i = 1 : n-1
      k1 = f(t(i), y(i));
      k2 = f(t(i) + h/2,y(i) + h * k1/2);
      y(i+1) = y(i) + h * k2;
    end
endfunction


t = linspace(0, 4, 10);
y0 = 0;
plot(t,pointMilieu(y0,t,f1));
```

CODE SOURCE 1.1.2(3) – Schéma du point milieu

- Etape 3 : Représenter graphiquement les approximations y_n superposées à la solution exacte pour des valeurs croissantes de N :

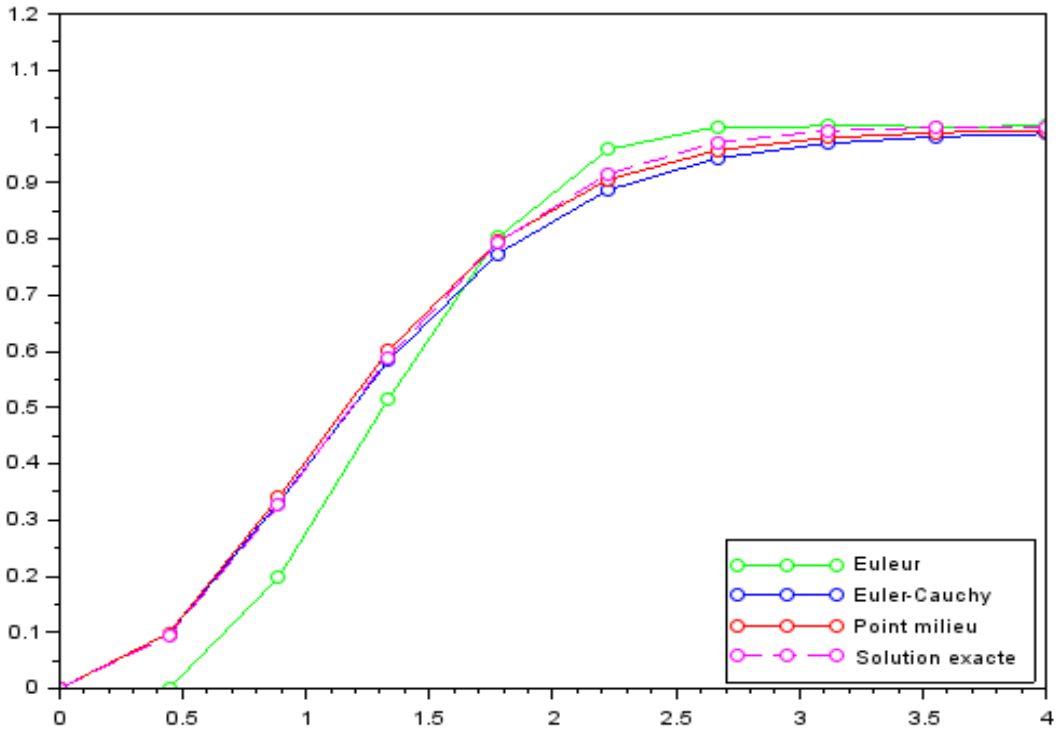


FIGURE 1.1.2 – Comparaison des différentes solutions obtenues

Visiblement, la solution approchée par schéma du point milieu est plus approximative que la solution exacte.

1.1.3. La macro *ode* de Scilab

La macro *ode* de Scilab permet de résoudre l'équation différentielle ordinaire. Voici par exemple ce qu'il faut écrire pour l'équation (4) :

```
function yprime = f1(t, y)
   yprime = - t * y + t;
endfunction

t = linspace(0, 4, 10);
y0 = 0;
y = ode(y0, 0, t, f1);
```

CODE SOURCE 1.1.3 – La macro ode

1.1.4. Evaluation de l'ordre de l'erreur des schémas

Une fonction d'erreur de convergence est définie comme suit :

$$E(h) = \max\ |y(t_n) - \ y_k\ | \text{ avec } n = 0\,..N.$$

Pour chacun des schémas et ainsi la macro *ode*, on calcule l'erreur de convergence $E(h)$ pour

$$N\ \in \{1, 3, 5, 10, 22, 47, 100, 216, 465, 1000\}$$

et représente graphiquement $E(h)$ en fonction de h en utilisant une échelle logarithmique pour les abscisses et les ordonnées.

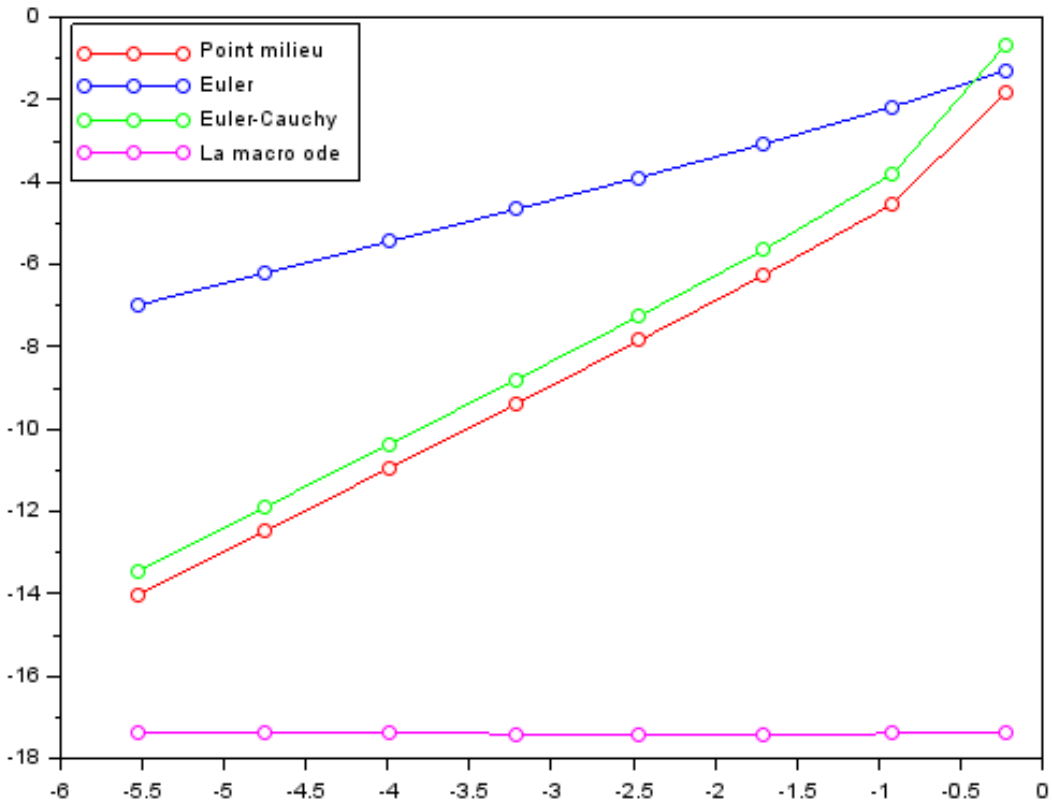


FIGURE 1.1.4 - Courbe représentation des fonctions d'erreurs pour les trois méthodes

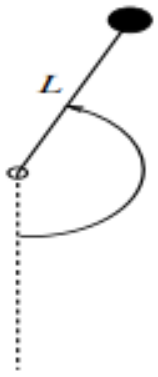
Puis, pour déterminer l'ordre d'erreur, on effectue une régression linéaire sur chacune des courbes avec la macro *reglin* de Scilab pour en déterminer la pente. On trouve :

- L ' ordre de convergence du schéma d ' Euler est de : 1.0635769
- L ' ordre de convergence du schéma d ' Euler - Cauchy est de : 2.2786725
- L ' ordre de convergence du schéma du point milieu est de : 2.2034476
- L ' ordre de convergence de la macro *ode* est de : -0.0041744

Comme nous l'avons constaté avant, les schémas d'Euler et d'Euler-Cauchy rétrograde sont peu précises, pour un calcul approximatif il faut utiliser la macro *ode.*

1.2. Application : Le pendule

Le pendule considéré est représenté dans la figure. Nous supposerons que la tige reliant un point de masse M à l'axe de rotation est de masse négligeable devant M. Nous choisissons de mesurer la déviation du pendule vertical d'équilibre stable par l'angle $\theta(t)$.



Si l'on applique les relations de la dynamique pour les solides en rotation autour d'un axe, on obtient l'équation différentielle suivante :

$$\begin{cases} \theta''(t) = -\dfrac{g}{L} sin\,\theta(t) \\ \theta(0) = \theta_0 \\ \theta'(0) = 0 \end{cases}$$

La valeur de θ_0 donne la déviation initiale du pendule, et on a considéré que la vitesse angulaire initiale est nulle. Cette équation est non-linéaire à cause de $sin\theta(t)$, et sa résolution est difficile. Cependant, si l'écart angulaire $\theta(t)$ est faible, sa mesure en radians est très peu différente de celle de $sin\theta(t)$. Si on peut se contenter d'une solution approchée, nous pouvons linéariser l'équation en confondant $\theta(t)$ et $sin\theta(t)$, on obtient l'équation différentielle suivante, où θ a été remplacé par ϕ pour bien montrer qu'il ne s'agit plus du même problème :

$$\begin{cases} \phi''(t) = -\frac{g}{L}\phi(t) \\ \phi(0) = \theta_0 \\ \phi'(0) = 0 \end{cases}$$

Il est assez élémentaire de montrer que la solution de cette équation différentielle est la fonction

$$\phi(t) = \theta_0 \cos\left(\sqrt{\frac{g}{L}}t\right).$$

En posant

$$y = \begin{pmatrix} \theta \\ \theta_0 \end{pmatrix},$$

on obtient alors

$$y = \begin{pmatrix} \theta' \\ \theta'' \end{pmatrix} = \begin{pmatrix} \theta' \\ -\frac{g}{L} * sin\theta \end{pmatrix}.$$

On va donc comparer la solution du modèle linéarisé avec l'approximation donnée par la macro *ode* de Scilab de la solution du système d'équations différentielles pour plusieurs valeurs de θ_0

```
function dydt=fpendule(t, y)
   theta = y(1);
   thetaprime = y(2);
   dydt = [thetaprime
           -g/L*sin(theta)];
endfunction

clf;
t = linspace(0, 10, 1000);
L = 1;
g = 9.81;
theta0 = %pi/32;
y = ode([theta0; 0], 0, t, fpendule);
theta = y(1, :);
plot(t, theta, t, theta0 * cos(t * sqrt(g/L)));
t = 'La solution linéarisée et l''aproximation de Scilab pour theta0 = π/32';
title(t);
legend('ode','La solution linearisée');
```

CODE SOURCE 1.2 – Le pendule

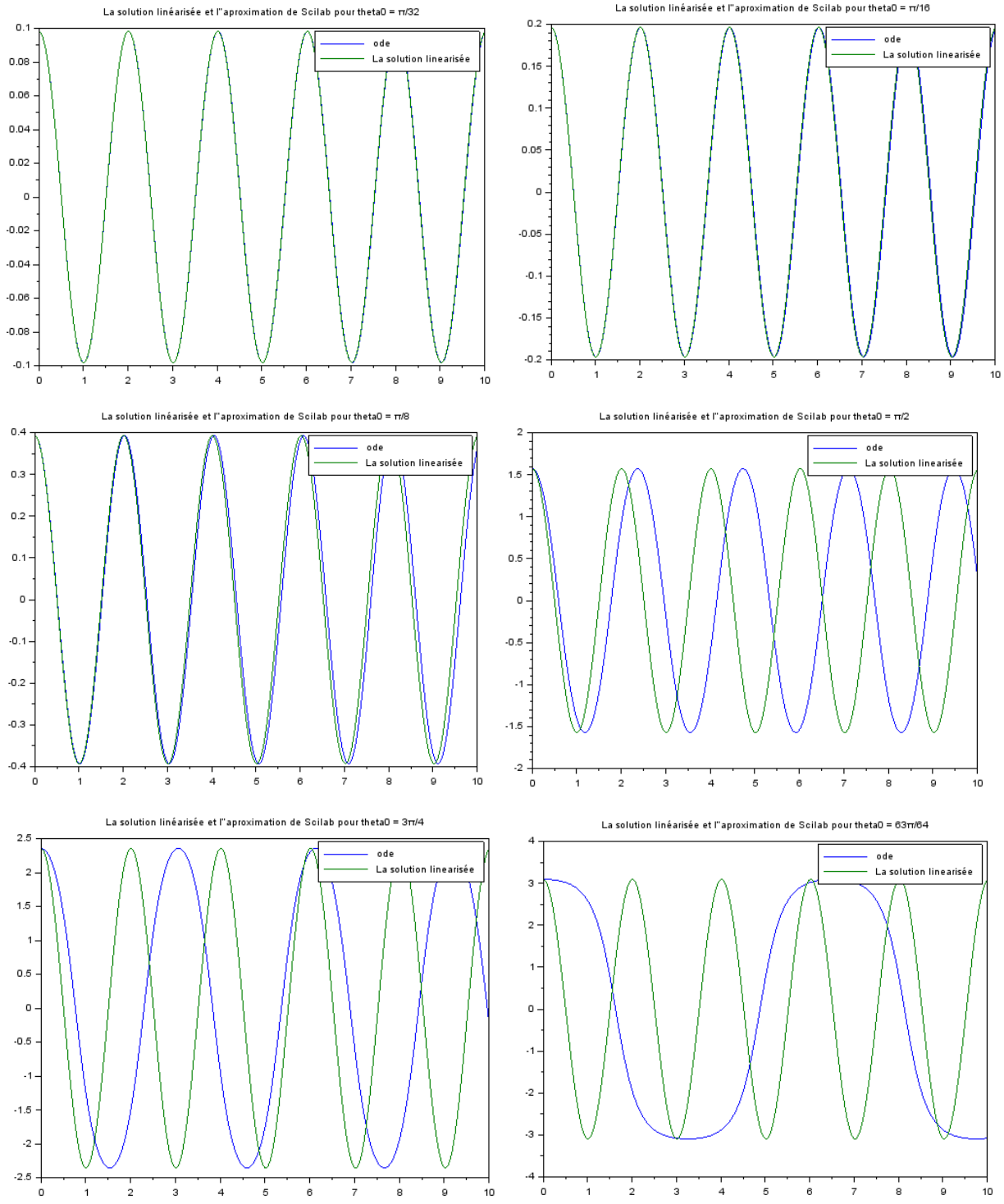


FIGURE 1.2 - Comparaison de la solution du modèle linéarisé avec l'approximation donnée par la macro ode de Scilab pour $\theta_0 = \frac{\pi}{32}, \frac{\pi}{16}, \frac{\pi}{8}, \frac{\pi}{2}, \frac{3\pi}{4}, \frac{63\pi}{64}$

Comme nous pouvons assez facilement le constater dans la figure au-dessus, plus le temps t est éloigné du temps t_0 initial, plus la fiabilité de la solution linéarisé décroit. De même, plus, l'écart angulaire initial θ_0 est élévé, moins le recours à la solution linéarisée s'avère judicieuse.

2. Systèmes dynamiques

2.1. Rappels de cours

Définition 3.1.1. Un système autonome est un cas particulier important d'un système d'équations différentielles où la variable n'apparait pas dans le système d'équations fonctionnelles. Soit $x(t) \in \mathbb{R}^n$

$$x' = f(x), \forall t > 0$$

Définition 3.1.2. Le vecteur $X^* \in \mathbb{R}^n$ est un point d'équilibre si

$$X(0) = X^* \Rightarrow \forall t > 0, X(t) = X^*$$

Les points d'équilibre de l'équation sont caractérisés par $f(X*) = 0$.
Définition 3.1.3. Le point d'équilibre $X*$ est stable si

$$\forall R > 0, \exists r > 0, \| X(0) - X^* \| < r \Rightarrow \forall t > 0, \| X(t) - X^* \| < R$$

Définition 3.1.4. Un point d'équilibre $X*$ qui n'est pas stable est dit instable et on a

$$\exists R > 0, \forall r > 0, \| X(0) - X^* \| < r \; et \; \exists t > 0, \| X(t) - X^* \| \geq R$$

Définition 3.1.5. Le point d'équilibre $X*$ est asymptotiquement instable s'il est stable et si

$$\exists r > 0, \| X(0) - X^* \| < r \rightarrow \lim_{t \to +\infty} \| X(t) - X* \| = 0$$

Propriété 2.2.1. Considérons le système

$$X' = AX, \forall t > 0$$

Le système au-dessus est :

- Instable si au moins une valeur propre de A est à partie réelle strictement positive.
- Stable si toutes les valeurs propres de A sont à parties réelle négative.
- Asymptotiquement stable si toutes les valeurs propres de A sont à parties réelle strictement négative.

2.2. Equation de Duffing

2.2.1. Système autonome

On considère l'équation différentielle :

$$x'' + kx' - x + x^3 = 0,$$

avec $k = 0.15$.

Tout d'abord, on commence par mettre cette équation différentielle sous forme d'un système d'équations différentielles du premier ordre en temps : $X' = f(t, X)$

En posant :

$$X = \begin{pmatrix} x \\ x' \end{pmatrix} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

on obtient alors :

$$X' = \begin{pmatrix} x' \\ x'' \end{pmatrix} = \begin{pmatrix} X_2 \\ -kX_2 + X_1 - X_1^3 \end{pmatrix}$$

On résout ensuite l'équation en vue d'obtenir les points d'équilibre :

$$f(X^*) = 0 \Leftrightarrow \begin{pmatrix} X_2^* \\ -kX_2^* + X_1^* + X_3^{3*} \end{pmatrix} = 0 \Rightarrow (X^*) \in \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right\}$$

On calcule ensuite la matrice Jacobienne de f et on obtient :

$$f'(X) = \begin{pmatrix} 0 & 1 \\ 1 - 3X_1^2 & k \end{pmatrix}$$

Enfin, on calcule les valeurs propres de la matrice Jacobienne aux différents points d'équilibre en vue de connaître les propriétés :

- Au point

$$(X_1^*) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

on obtient comme valeurs propres

$$\lambda_1 = \frac{-k + \sqrt{k^2 + 4}}{2}, \lambda_2 = \frac{-k - \sqrt{k^2 + 4}}{2}$$

avec $k = 0.15$ on a $\lambda_1 > 0$ and $\lambda_2 < 0$. Donc (X_1^*) est un point d'équilibre instable.

- Au point

$$(X_2^*) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, (X_3^*) = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

on obtient les mêmes valeurs propres :

$$\lambda_1 = \frac{-k + i\sqrt{-k^2 + 8}}{2}, \lambda_2 = \frac{-k - i\sqrt{k^2 + 8}}{2}$$

toutes les valeurs propres sont à partie réelle strictement négative, donc les points (X_2^*), (X_3^*) sont des points d'équilibre asymptotiquement stable.

On peut utiliser la fonction ode de Scilab pour dessiner le portrait de phase des systèmes dynamiques en superposant le comportement des solutions avec des conditions initiales $X(0)$:

```
function Xprime=f(t, X)
  Xprime = [ X(2)
             -k * X(2) + X(1) - X(1)^3];
endfunction

k = 0.15;
t = linspace(0, 200, 1000);
clf;
for X10 = 0 : 0.5 : 3
  for X20 = 0 : 0.5 : 3
    X0 = [X10; X20];
    X = ode(X0, 0, t, f);
    plot(X(1,:), X(2,:));
  end
end
fchamp(f,0,(0 : 0.5 : 3),(0 : 0.5 : 3));
```

CODE SOURCE 2.2.1 - Système autonome d'équation de Duffing

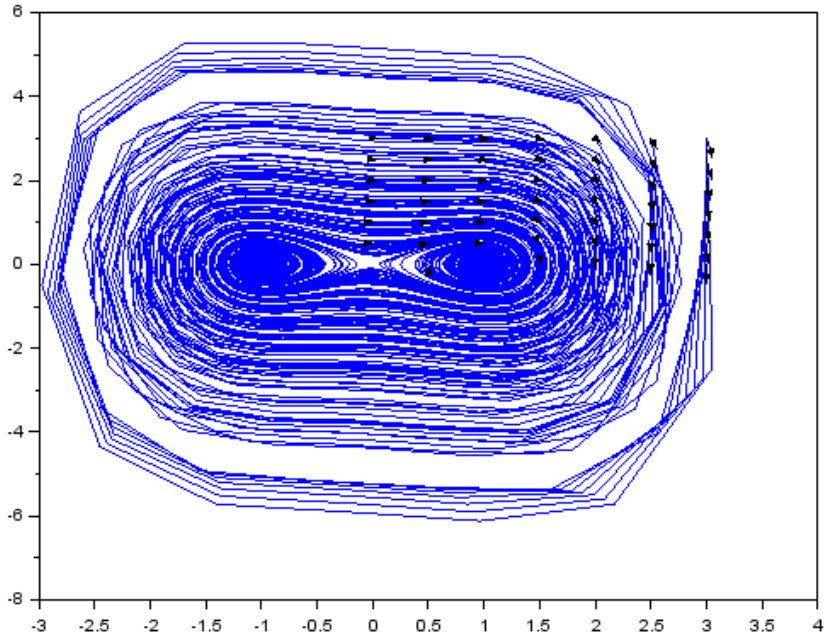


FIGURE 2.2.1 - Portrait de phase de l'équation Duffing autonome

2.2.2. Système non autonome

On considère maintenant l'équation différentielle

$$x'' + kx' - x + x^3 = bcost,$$

avec $b = 0.3$ et $k = 0.15$

Tout d'abord, on commence par mettre cette équation sous forme d'un système d'équations différentielles du premier ordre en temps de la forme: $X' = f(t, X)$.

En posant :

$$X = \begin{pmatrix} x \\ x' \end{pmatrix} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

On obtient alors :

$$X' = \begin{pmatrix} x' \\ x'' \end{pmatrix} = \begin{pmatrix} X_2 \\ -kX_2 + X_1 - X_1^3 + bcost \end{pmatrix}$$

On va calculer avec la solution de cette équation différentielle pour $t \in [0,40\pi]$ en prenant les conditions initiales :

1. $x(0) = x'(0) = 0$. La solution figure en bleu sur la figure 2.2.2.
2. $x(0) = 10^{-6}, x'(0) = 0$. La solution figure en rouge sur la figure 2.2.2.

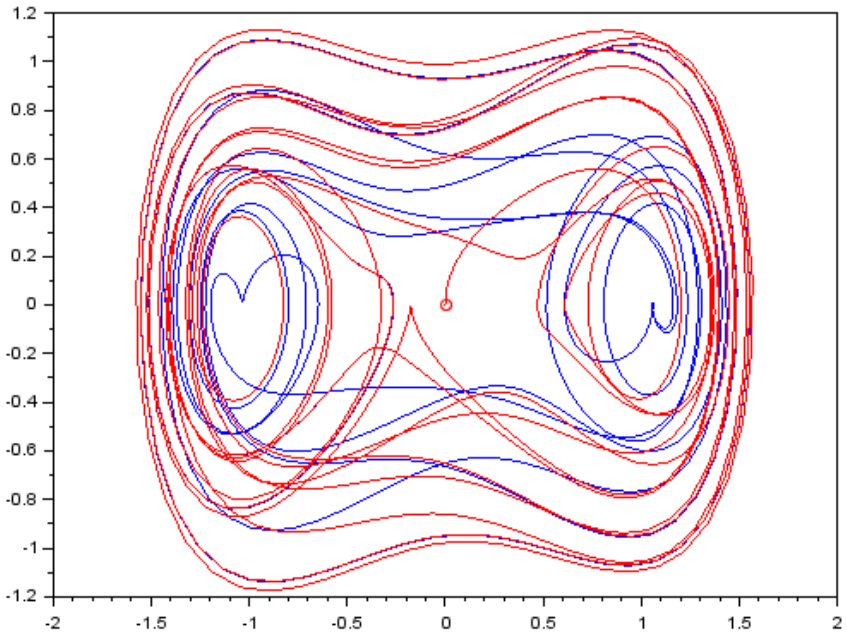


FIGURE 2.2.2 - Portrait de phase de l'équation Duffing non autonome

Les points $\binom{1}{0}, \binom{-1}{0}$ des points d'équilibre instable. En effet, les trajectoires semblent s'écarter de ces deux points. De plus, ces trajectoires semblent atteindre des cycle limite. Un cycle limite est une solution temporellement périodique qui est indépendante des conditions initiales et qui possède une fréquence intrinsèque au système indépendante des conditions initiales. Ici, c'est bien le cas, les trajectoires pour la première condition initiale et la seconde condition initiale semblent atteindre une solution temporellement périodique et possédant une fréquence intrinsèque. Cette dernière est bien indépendante des conditions initiales puisque la courbe bleue et le courbe rouge de la figure 2.2.2 semblent identiques.

2.3. Equation de Van de Pol

On considère l'équation différentielle

$$x'' - \mu(1 - x^2)x' + x = 0$$

Tout d'abord, on commence par mettre cette équation sous forme d'un système d'équations différentielles du premier ordre en temps de la forme $X' = f(t, X)$.

En posant :

$$X = \binom{x}{x'} = \binom{X_1}{X_2}$$

on obtient alors :

$$X' = \begin{pmatrix} x' \\ x'' \end{pmatrix} = \begin{pmatrix} X_2 \\ \mu(1 - X_1^2)X_2 - X_1 \end{pmatrix}$$

On résout ensuite l'équation en vue d'obtenir les points d'équilibre :

$$f(X^*) = 0 \Leftrightarrow \begin{pmatrix} X_2 \\ -\mu(1 - X_1^2)X_2 - X_1 \end{pmatrix} = 0 \Rightarrow (X^*) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

On calcule ensuite la matrice Jacobienne de f et on obtient :

$$f'(X) = \begin{pmatrix} 0 & 1 \\ -\mu 2X_1X_2 - 1 & \mu(1 - X_1^2) \end{pmatrix}$$

Enfin, on calcule la valeur propre de la matrice Jacobienne au point d'équilibre (X^*) et on obtient alors :

$$\lambda_1 = \frac{\mu + \sqrt{21}}{2}, \lambda_2 = \frac{\mu - \sqrt{21}}{2}$$

pour $\mu \in [0.01, 5]$, on a $\lambda_1 > 0$ et $\lambda_2 < 0$, donc (X^*) est un point d'équilibre instable.

Exemple de l'équation de Van der Pol pour $\mu = 1$:

```
function Xprime=f(t, X)
        Xprime = [ X(2)
                   u*(1-X(1).^2)*X(2)-X(1)];
endfunction

u = 1;
t = linspace(0,10,1000);
clf;
for X10 = -3 : 0.5 : 3
    for X20 = -3 : 0.5 : 3
        X0 = [X10; X20];
        X = ode(X0,0,t,f);
        plot(X(1,:),X(2,:));
    end
end
fchamp(f,0,(-3 : 0.5 : 3),(-3 : 0.5 : 3));
```

CODE SOURCE 2.3 – Equation de Van de Pol pour $\mu = 1$

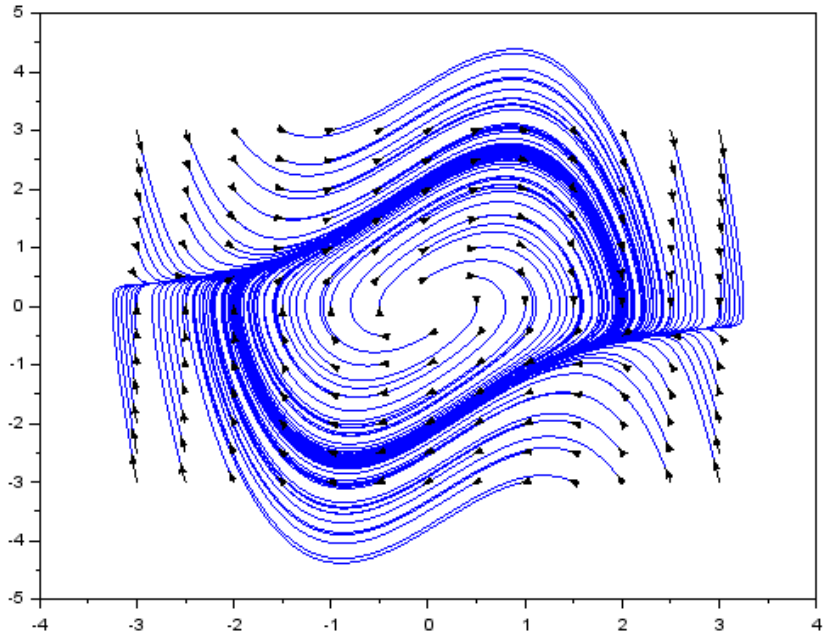


FIGURE 2.3(1) - Portrait de phase de l'équation de Van der Pol pour $\mu = 1$

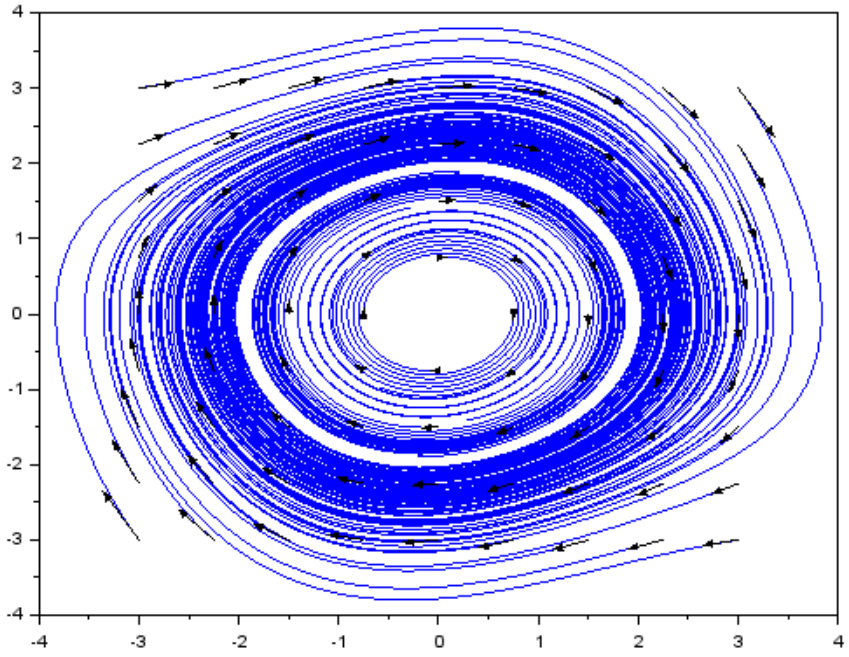


FIGURE 2.3(2) - Portrait de phase de l'équation de Van der Pol pour $\mu = 0.1$

Visiblement, ces deux figures nous confirment que (X^*) est un point d'équilibre instable. En effet, les trajectoires proches de ce point s'éloignent de celui-ci dans les deux figures. De plus, on peut observer un cycle limite pour l'oscillateur de Van der Pol. En effet, toutes les trajectoires tendent à former une figure fermée : le système de Van der Pol a tendance à maintenir les oscillations.

2.4. Système proie-prédateur (Lokta-Voltera)

On considère maintenant le système :

$$\begin{cases} x' = x(\alpha - \beta y) \\ y' = -y(\gamma - \delta x) \end{cases}$$

Avec $\alpha = 2$, $\beta = 10^{-3}$, $\gamma = 10$, $\delta = 2.10^{-3}$.

Tout d'abord, on commence par mettre cette équation sous forme d'un système d'équations différentielles du premier ordre en temps de la forme $X' = f(t, X)$.

En posant :

$$X = \begin{pmatrix} x \\ x' \end{pmatrix} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

on obtient alors :

$$X' = \begin{pmatrix} x' \\ x'' \end{pmatrix} = \begin{pmatrix} X_1(\alpha - \beta X_2) \\ -X_2(\gamma - \delta X_1) \end{pmatrix}$$

On résout ensuite l'équation en vue d'obtenir les points d'équilibre :

$$f(X^*) = 0 \Leftrightarrow \begin{pmatrix} X_1(\alpha - \beta X_2) \\ -X_2(\gamma - \delta X_1) \end{pmatrix} = 0 \Rightarrow (X^*) = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \alpha/\beta \\ \gamma/\delta \end{pmatrix} \right\}$$

On calcule ensuite la matrice Jacobienne de f et on obtient

$$f'(X) = \begin{pmatrix} \alpha - \beta X_2 & -\beta X_1 \\ \delta X_2 & \delta X_1 - \gamma \end{pmatrix}$$

Enfin, on calcule les valeurs propres de la matrice Jacobienne aux différents points d'équilibre en vue de connaître les propriétés :

- Au point

$$(X_1{}^*) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

on obtient comme valeurs propres

$$\lambda_1 = \frac{\alpha - \gamma + \sqrt{-\alpha + \gamma^2 + 4\alpha\gamma}}{2}$$
$$\lambda_2 = \frac{\alpha - \gamma - \sqrt{-\alpha + \gamma^2 + 4\alpha\gamma}}{2}$$

Avec les conditions initiales, on a $\lambda_1 > 0$ and $\lambda_2 < 0$. Donc (X_1^*) est un point d'équilibre instable.

- Au point

$$(X_2^*) = \begin{pmatrix} \alpha/\beta \\ \gamma/\delta \end{pmatrix}$$

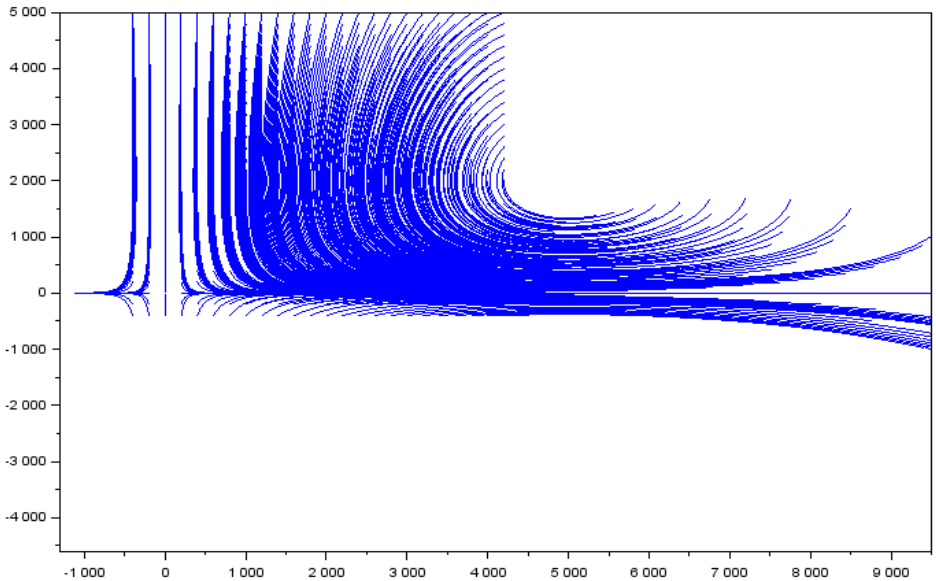
on obtient comme valeurs propres

$$\lambda_1 = i\sqrt{20}, \qquad \lambda_2 = -i\sqrt{20}$$

Toutes les valeurs propres sont à partie réelle négative, (X_2^*) est donc un point d'équilibre stable.

```
function Xprime=f(t, X)
   Xprime = [ X(1)*(a-b*X(2))
            -X(2)*(c-d*X(1))];
endfunction

a = 2;
b = 10^(-3);
c = 10;
d = 2*10^(-3);
t = linspace(0,0.5,100);
clf;
for X10 = -400 : 200 : 10000
   for X20 = -400 : 200 : 5000
      X0 = [X10; X20];
      X = ode(X0,0,t,f);
      plot(X(1,:),X(2,:));
   end
end
```

CODE SOURCE 2.4 - Système proie-prédateur



CHAPITRE 4

Problèmes non-linéaires (II)

1. Problèmes de moindres carrés linéaires

1.1. Méthode des moindres carrés

On considère le modèle $y = f(x, \theta)$ est linéaire :

$$y = \sum_{k=1}^{p} \theta_k \varphi_k(x), \quad \varphi_k : \mathbb{R} \to \mathbb{R}$$

La méthode consiste en une prescription (initialement empirique), qui est que la fonction $y = f(x, \theta)$ qui décrit « le mieux » les données est celle qui minimise la somme quadratique des déviations des mesures aux prédictions de $f(x, \theta)$. Si, par exemple, nous disposons de N mesures $(y_i)_{i=1..n}$, les paramètres θ « optimaux » au sens de la méthode des moindres carrés sont ceux qui minimisent la quantité :

$$S(\theta) = \sum_{i=1}^{n} (y_i - f(x_i, \theta))^2 = \| r(\theta) \|^2,$$

où $r(\theta)$ est le vecteur résiduel du modèle

$$r(\theta) = A\theta - y$$

On trouve $\hat{\theta}$ tel que :

$$\hat{\theta} = \arg\min S(\hat{\theta}) = \| A\theta - y \|^2$$

Condition d'optimalité nécessaire :

$$\nabla S(\hat{\theta}) = 0$$

Grâce à un développement de Taylor, on obtient :

$$\nabla S(\hat{\theta}) = 2A^T(A\theta - y)$$

Théorème 1.1.1 : La solution de problème de moindres carrés linéaires est donnée par $\hat{\theta}$, solution d'équation suivante :

$$A^T A\hat{\theta} = A^T y$$

Remarque 1.1.2 : Si $rang(A) = p$, elle admet une solution $\hat{\theta}$ unique.

1.2. Régression polynômiale avec validation

1.2.1. Régression

On considère n couples $(t_i, y_i)_{i=1..n}$. On cherche un polynôme de degré p défini par

$$P(t) = \theta_1 + \theta_2 t + \cdots + \theta_{p+1} t^p \quad (1)$$

tel que la quantité

$$S(\theta) = \sum_{i=1}^{n} (P(t_i) - y_i)^2$$

est minimale, où on a noté $\theta = (\theta_1, \dots, \theta_{p+1})^T$

D'après les méthodes des moindres carrés, on pose :

$$S(\theta) = \| A\theta - y \|^2$$

où

$$A = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^p \\ 1 & t_2 & t_2^2 & \dots & t_2^p \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & t_n & t_n^2 & \dots & t_n^p \end{bmatrix}$$

Le problème de moindres carrés admet une solution unique si et seulement si $rang(A) = p$+1. On va chercher la condition sur les $(t_i)_{i=1..n}$ pour que le problème admette une solution unique :

- Soit $p = 1$:

$$A = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_n \end{bmatrix}$$

On a bien que :

$$\begin{vmatrix} 1 & t_i \\ 1 & t_j \end{vmatrix} = t_j - t_i$$

On en déduit que

$$rang\ A = p + 1 = 2 \Leftrightarrow t_j - t_i \neq 0 \Leftrightarrow \exists i, j \in \{1..n\}, t_j \neq t_i$$

- Soit $p = 2$

$$A = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ \vdots & \vdots & \vdots \\ 1 & t_n & t_n^2 \end{bmatrix}$$

On a bien que :

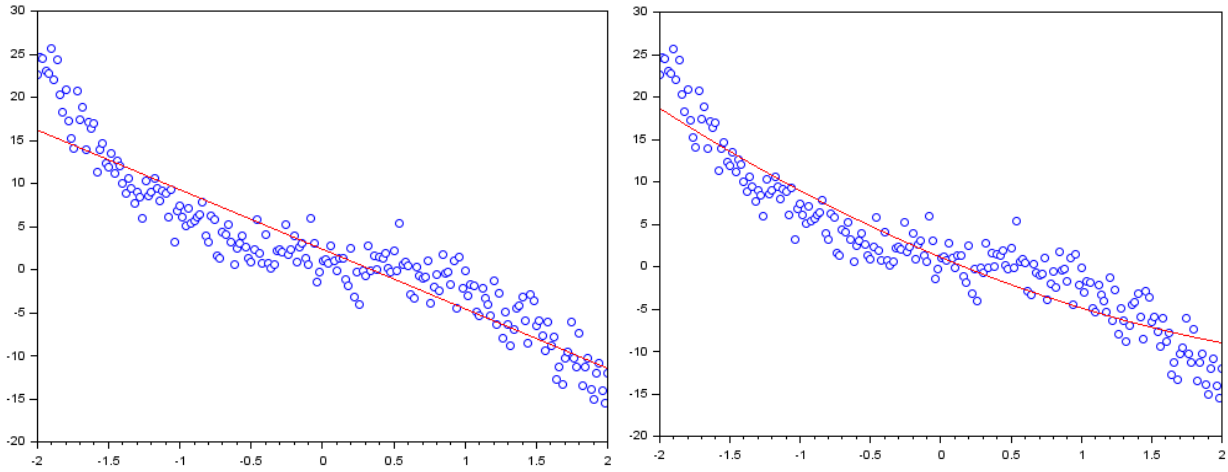
$$\begin{vmatrix} 1 & t_i & t_i^2 \\ 1 & t_j & t_j^2 \\ 1 & t_k & t_k^2 \end{vmatrix} = (t_k - t_j)(t_k - t_i)(t_j - t_i)$$

On en déduit que

$$rang\ A = p + 1 = 3 \Leftrightarrow t_i \neq t_j \neq t_k$$

Par récurrence, on en déduit que ce polynôme de degré p admet une solution unique si et seulement si $\exists\ i_1, i_2, \ldots, i_{p+1} \in \{1..n\}$, tel que $t_{i_j} \neq t_{i_j}$,

On écrit ensuite un programme Scilab permettant de résoudre le problème de moindres carrés, et représenter à l'écran les points $(t_i, y_i)_{i=1..n}$ et le polynôme $P(t)$ obtenu pour les valeurs croissantes de son degré :



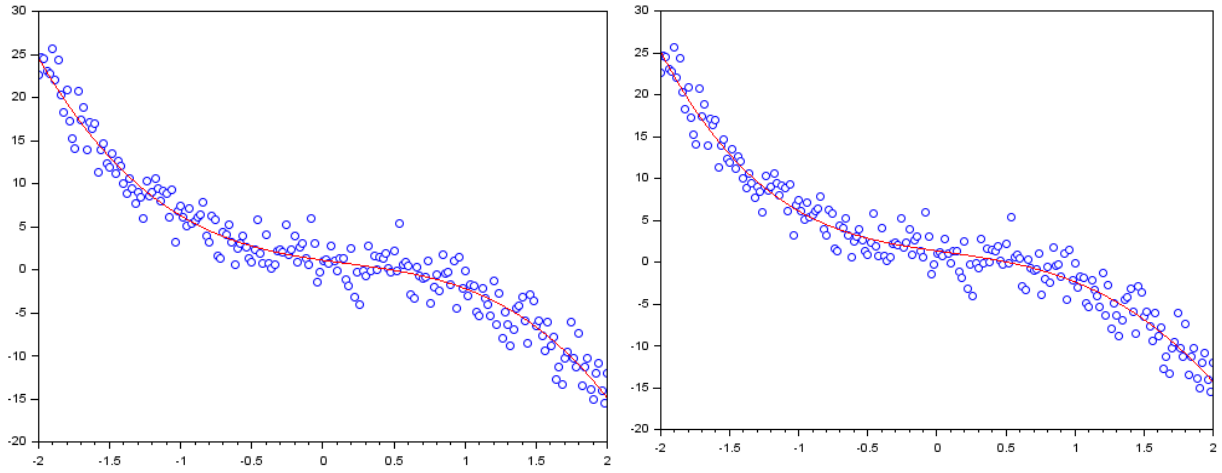
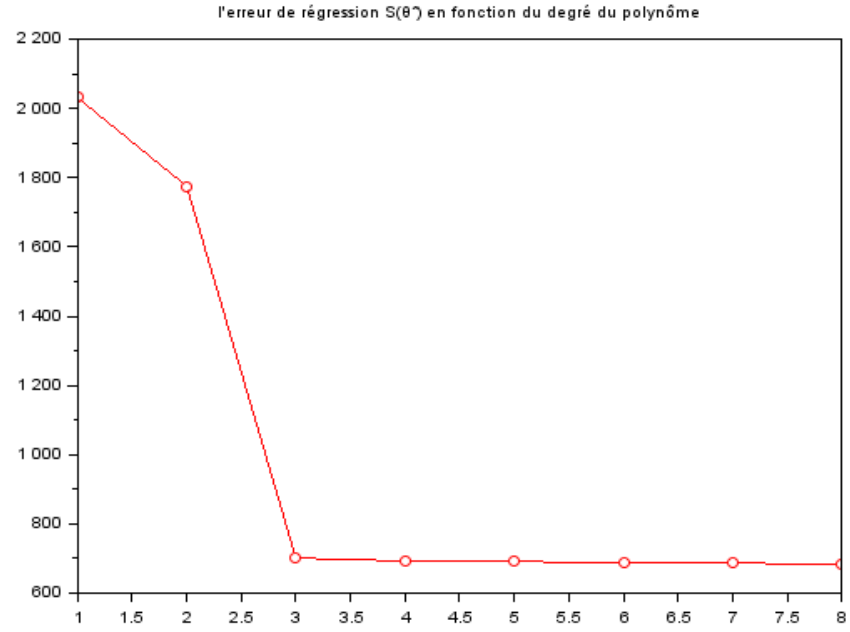


FIGURE 1.2.1 – Solution du problème de moindres carrés pour $p = 1,2,3,4$

1.2.2. Validation

Dans cette partie, on va chercher le degré p (valeurs de p entre 1 et 8) le plus adapté pour le polynôme (1). Tout d'abord, on trace l'erreur de régression $S(\hat{\theta})$ en fonction du degré du polynôme :



p	$S(\hat{\theta})$
1	2030.7641
2	1774.9428
3	701.20579
4	692.8908
5	691.68128
6	686.86014
7	686.81951
8	683.3821

FIGURE 1.2.2(1) – L'erreur de régression $S(\hat{\theta})$ en fonction du degré p du polynôme $p = 1 \dots 8$

On constate que l'erreur de régression $S(\hat{\theta})$ pour $p = 3,4,5,6,7,8$ sont proches, donc, on ne peut pas l'utiliser pour choisir le degré p le plus adapté.

On propose d'une répartition des points $(t_i, y_i)_{i=1..n}$ en un ensemble d'apprentissage *T* et un ensemble de validation *V*. J'ai choisi :

$$T = \{t_i, i = 1..n \mid t_i < -1 \; ou \; t_i > 1\}$$

$$V = \{t_i, i = 1..n \mid -1 \leq t_i \leq 1\}$$

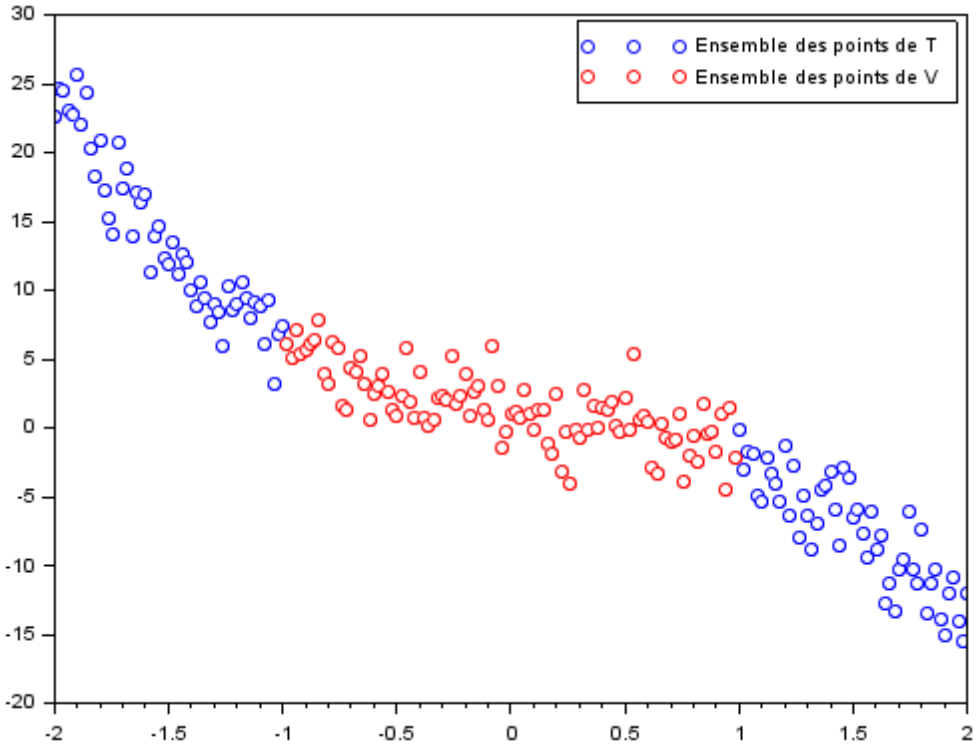


FIGURE 1.2.2(2) – Répartition des points

On définit l'erreur d'apprentissage et l'erreur de validation respectivement par :

$$S_T(\theta) = \sum_{t_i \in T} (P(t_i) - y_i)^2$$

$$S_T(\theta) = \sum_{t_i \in V} (P(t_i) - y_i)^2$$

Et $\hat{\theta}^{(p)}$ les coefficients du polynôme de degré p rendant cette erreur minimale. On écrit ensuite un programme Scilab pour calculer et trace $S_T(\hat{\theta}^{(p)})$ et $S_V(\hat{\theta}^{(p)})$ en fonction de $p = 1..8$:

```
clf;
n = size(t, 1);
V = find(t > -1 & t < 1);
T = setdiff(1:n, V);
//plot(t(T), y(T), 'bo', t(V), y(V), 'or');
A_T = [ones(t(T))];
A_V = [ones(t(V))];
sT = [];
sV = [];
p = [];
for i = 1 : 8
            A_T = [A_T, t(T).^i];
            theta = A_T\y(T);
            A_V = [A_V, t(V).^i];
            E_T = y(T) - A_T * theta;
            E_V = y(V) - A_V * theta;
            m = size(T, 1);
            sT = [sT, 0];
             for j = 1 : m
                        sT(i) = sT(i) + E_T(j,1).^2;
             end;
             sV = [sV, 0];
             l = size(V, 1);
            for h = 1 : l
                        sV(i) = sV(i) + E_V(h,1).^2;
            end;
end;
plot(sV, 'bo-', sT, 'ro-');
```

FIGURE 1.2.2 – Code source de validation

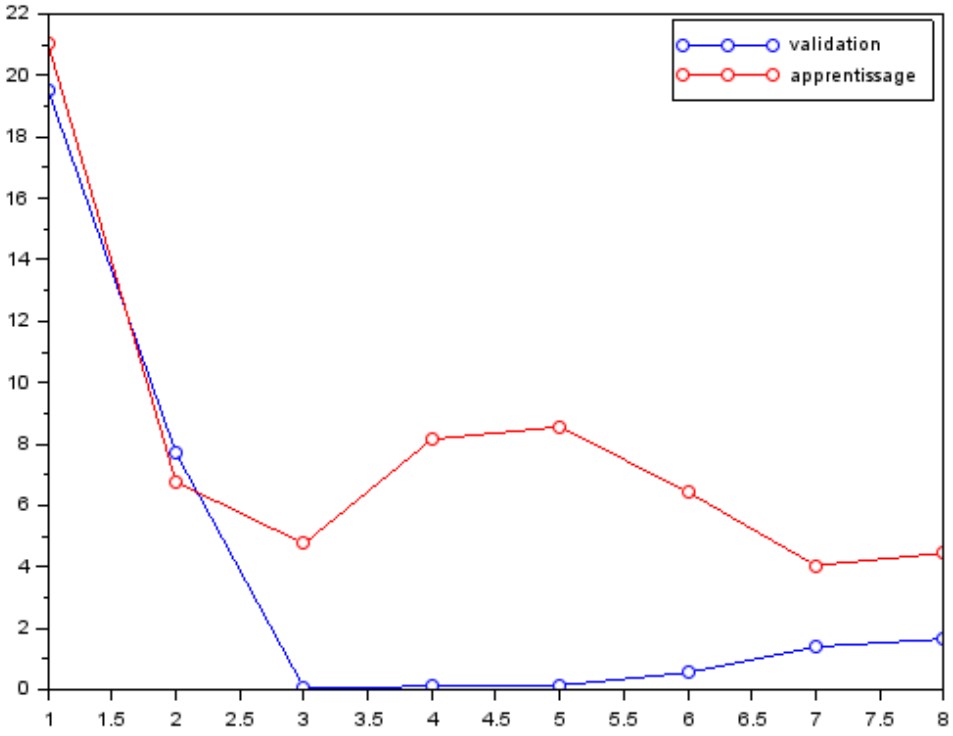


FIGURE 1.2.2(3) – Courbes de $S_T(\hat{\theta}^{(p)})$ et $S_V(\hat{\theta}^{(p)})$ en fonction de $p = 1..8$

Visiblement, pour $p = 3, S_V(\hat{\theta}^{(p)})$ obtient la valeur minimale, $p = 3$ est donc le degré le plus adapté du polynôme (1).

1.3. Restauration d'une image

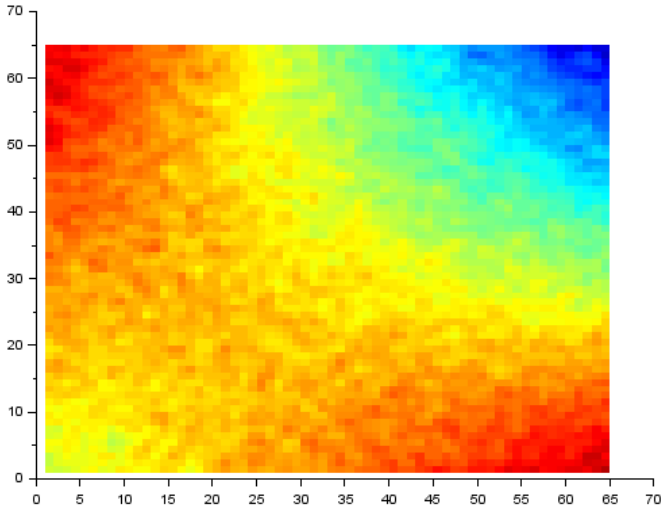


FIGURE 1.3(1) – Image avant restauration

Il s'agit d'améliorer la qualité visuelle d'une image, représentée ici sur la figure 1.3(1). Cette image est du type de celles rencontrées en astronomie, dans le domaine de l'imagerie médicale, etc. Dans ces divers domaines, et pour des raisons variées, les images brutes sont souvent de mauvaise qualité : on dit qu'elles sont "bruitées". Avant de pouvoir les exploiter il faut leur faire subir un traitement permettant d'améliorer leur qualité visuelle.

On considère que l'image est une matrice Z à n lignes et n colonnes, dont chaque élément z_{ij}, représentant un pixel de l'image, a une valeur donnée, proportionelle à l'intensité lumineuse. Pour débruiter cette image, on approcher chaque terme z_{ij} par une fonction polynômiale des deux variables i et j, dont le degré par rapport à l'une de ces deux variables est au plus 1, c'est à dire de la forme

$$f_c(i,j) = c_1 + c_2 i + c_3 j + c_4 ij$$

où c_1, c_2, c_3, c_4 sont les quatre coefficients qu'il va falloir déterminer.

Pour déterminer les coefficients c_1, c_2, c_3, c_4 tels que les valeurs de la fonction f_c aux points (i,j) approchent au mieux l'image, on va chercher à minimiser par rapport à c la fonction coût quadratique

$$J(c) = \sum_{i=1}^{n}\sum_{j=1}^{n}(z_{ij} - f_c(i,j))^2$$

Une fois déterminés les coefficients c_1, c_2, c_3, c_4 minimisant la fonction coût $J(c)$, on peut calculer une nouvelle image dite image restaurée représentée par une matrice $\tilde{Z}$, dont les éléments sont donnés par

$$\widetilde{z_{ij}} = f_c(i,j)$$

On définit le vecteur

$$b = \begin{pmatrix} z_{11} \\ z_{21} \\ \vdots \\ z_{n1} \\ z_{12} \\ \vdots \\ z_{n2} \\ \vdots \\ z_{1n} \\ \vdots \\ z_{nn} \end{pmatrix}, c = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}$$

On va préciser alors la structure de matrice A pour que l'on ait :

$$J(c) = \sum_{i=1}^{n}\sum_{j=1}^{n}(z_{ij} - f_c(i,j))^2 = \| Ac - b \|^2$$

Après calculer, j'ai en déduit que

$$A = \begin{pmatrix} 1 & 1 & 1 & 1*1 \\ 1 & 2 & 1 & 2*1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & i & j & i*j \\ \vdots & \vdots & \vdots & \vdots \\ 1 & n & n & n*n \end{pmatrix}$$

Ensuite, j'ai écrit un programme Scilab qui réalise dans l'ordre des opération suivantes :

- Construction de la matrice A et du vecteur b
- Résolution du problème de moindres carrés permettant d'obtenir le vecteur c.
- Calcul de l'image restaurée $\tilde{Z}$
- Affichage l'image restaurée.

```
n = size(Z, 1);
b = zeros(n*n, 1);
tmp = 0;
for i = 1 : n       //Construire vecteur b;
      for j = 1 : n
            tmp = tmp + 1;
            b(tmp, 1) = Z(j, i);
      end
end
A = zeros(n * n, 4);
tmp = 0;
for i = 1 : n       //Construire matrice A;
      for j = 1 : n
            tmp = tmp + 1;
            A(tmp,:) = [1, j, i, i*j];
      end
end
c = A\b;
res = A*c;
tmp = 0;
for i = 1 : n       //Construire matrice Z~
      for j = 1 : n
            tmp = tmp + 1;
            ZZ(j,i) = res(tmp,1);
      end
end
set(gcf(),'color_map',jetcolormap(128));
grayplot(1:65,1:65,ZZ);              //Affichage l'image restaurée
```

CODE SOURCE 1.3 – Restauration d'une image

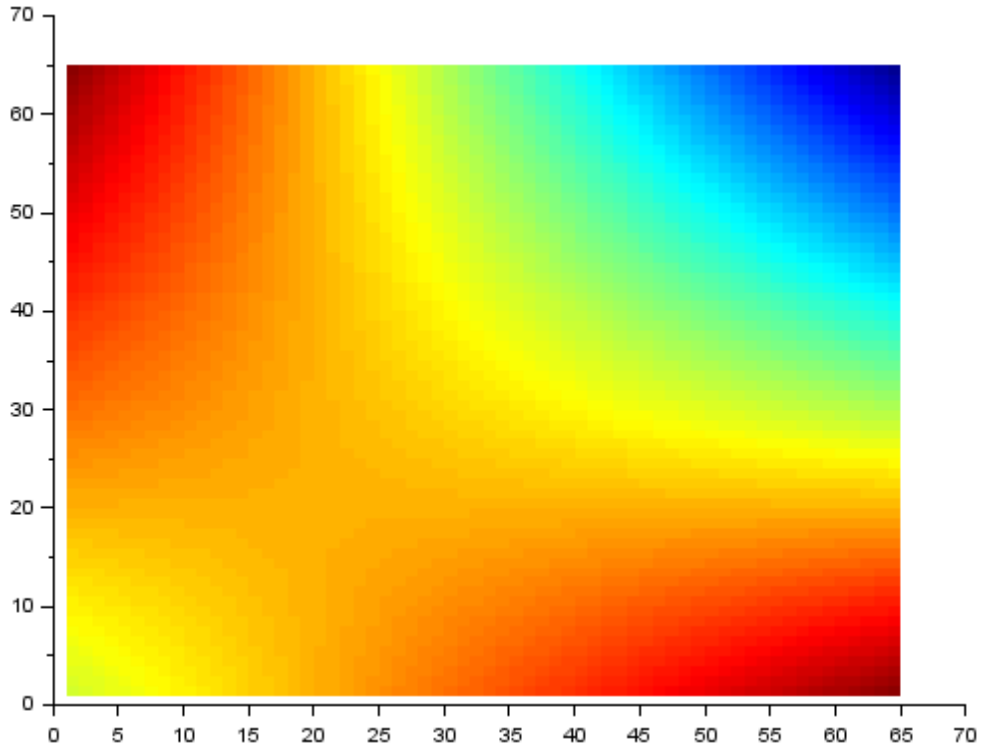


FIGURE 1.3(2) - L'image restaurée

2. Problèmes de moindres carrés non linéaires

2.1. La méthode de Levenberg-Marquardt

Cette méthode permet de déterminer le minimum d'une fonction $f : \mathbb{R}^n \to \mathbb{R}$ s'écrivant sous la forme

$$f(x) = \|g(x)\|^2,$$

où $g : \mathbb{R}^n \to \mathbb{R}^m$. Elle est définie comme la méthode itérative

$$x_{k+1} = x_k - (g'(x_k)^T g'(x_k) + \lambda I)^{-1} \nabla f(x_k)$$

où $\nabla f(x_k) = g'(x_k)^T g(x_k)$ et $\lambda > 0$. Lorsque λ est grand, la méthode est robuste car elle se comporte comme la méthode du gradient

$$x_{k+1} = x_k - \frac{1}{\lambda} \nabla f(x_k),$$

avec de surcroit un pas $1/\lambda$ très petit, par contre elle en hérite du principal défaut, sa lenteur.

Afin d'illustrer l'influence de λ, on considère la fonction de Rosenbrock, définie par

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

Qui présente un minimum global en $(1,1)$.

2.1.1. Les courbes iso-valeurs

Tout d'abord, on commence par tracer les courbes iso-valeurs de f dans le domaine $[-1,1] * [-0.5,1.5]$ en utilisant la macro *contour* de Scilab :

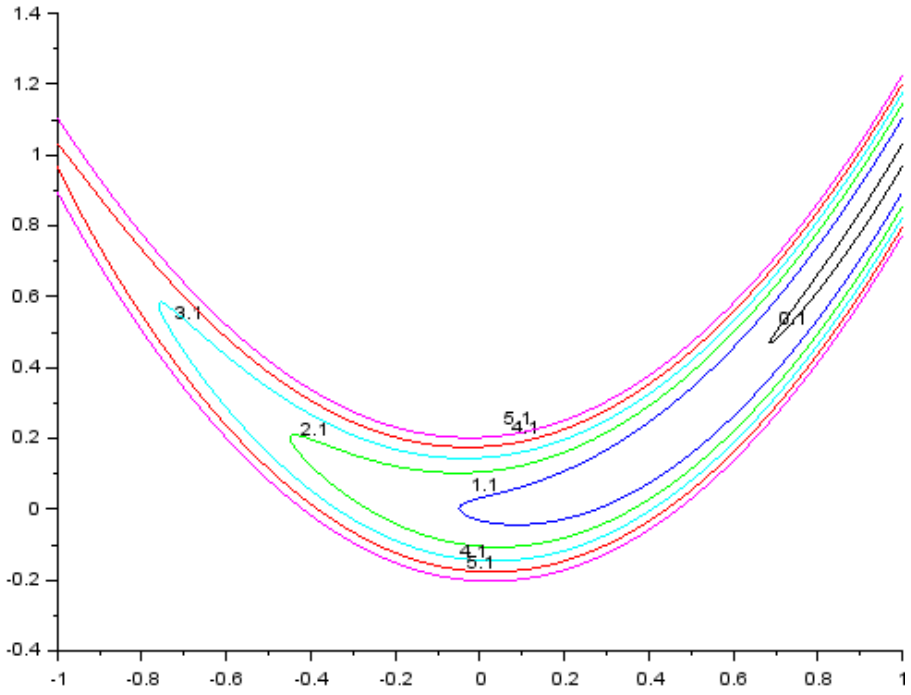


FIGURE 2.1.1 – Courbes iso-valeurs de la fonction Rosenbrock

2.1.2. La méthode du gradient

On considère la méthode du gradient

$$x_{k+1} = x_k - \frac{1}{\lambda}\nabla f(x_k),$$

Pour la fonction Rosenbrock on a

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 = \|g(x)\|^2$$

$$\Rightarrow \|g(x)\| = \begin{pmatrix} 1 - x_1 \\ 10(x_2 - x_1^2) \end{pmatrix}$$

On calcule ensuite la jacobienne de $g(x)$:

$$g'(x) = \begin{pmatrix} -1 & 0 \\ -20x_1 & 10 \end{pmatrix}$$

Il ne restes alors plus qu'à écrire un programme Scilab pour afficher les points successifs x_k et x_{k+1}, j'ai pris $x_0 = (0.5,1.5)$ et $\lambda = 200\ et\ 300$

```
function out=g(x)
    out = [1 - x(1)
           10*(x(2) - x(1)^2)];
endfunction

function jac=gprime(x)
    jac = [-1,          0
           -20*x(1), 10];
endfunction

function y=grad(x)
    y = gprime(x)'*g(x);
endfunction

x = zeros(2,4000);
x(:,1) = [0.5;1.5];
lambda = 300;
for k = 1 : 3999
    x(:,k+1) = x(:,k)-1/lambda*grad(x(:,k));
end
//Tracer les courbes iso-valeurs
x1 = -1 : .005 : 1.5;
x2 = -0.5 : .005 : 1.5;
[X1, X2] = ndgrid(x1, x2);
Z = (1 - X1).^2 + (10 * (X2 - X1.^2)).^2;
contour(x1, x2, Z,  [0.1:1:6]);
//Afficher des points  x_k,x_k+1
plot(1,1,'xr');
plot(x(1,:),x(2,:),'-o');
```

CODE SOURCE 2.1.2 – Le méthode du gradient pour $\lambda = 300$ après 4000 itérations

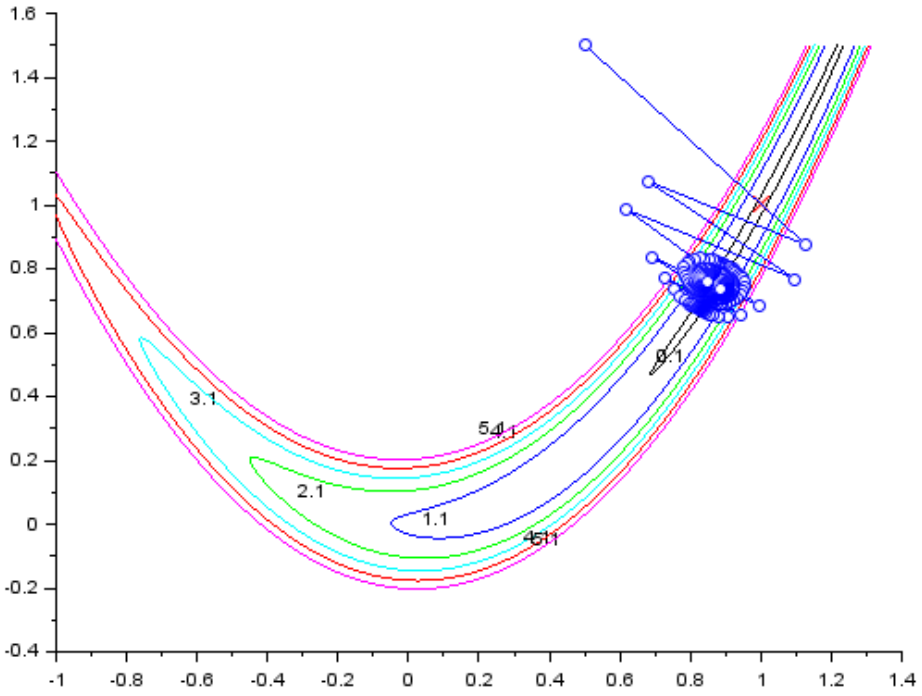


FIGURE 2.1.2(1) - Le méthode du gradient pour $\lambda = 200$ *après 4000 itérations*

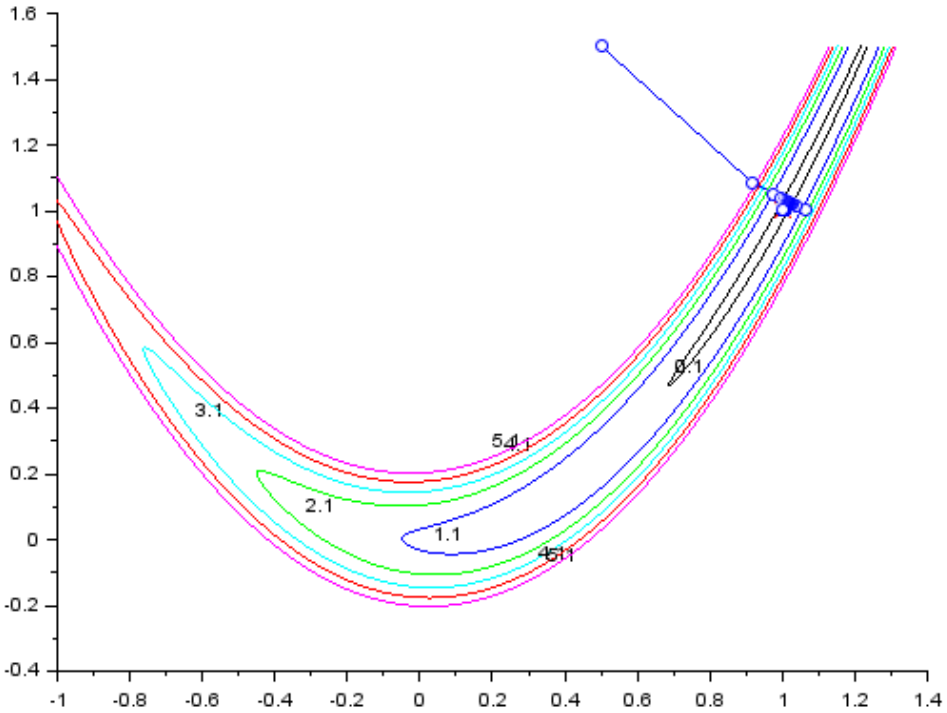


FIGURE 2.1.2(2) - Le méthode du gradient pour $\lambda = 300$ *après 4000 itérations*

Pour $\lambda_1 = 200$, et $\lambda_2 = 300$ après 4000 itérations, on obtient le point minimum global de la fonction de Rosenbrock respectivement en

$$x_{\min 1}=(0.7386408, 0.8820556) \text{ et } x_{\min 2}=(1.0017493, 1.0008725)$$

Les résultats obtenus illustrent l'influence de λ, en effet, lorsque λ est plus grand, le surcroit un pas $\frac{1}{\lambda}$ est plus petit, la méthode du gradient est alors plus précise.

2.1.3. La méthode de Levenberg-Marquardt

On considère maintenant la méthode de Levenberg-Marquardt, c'est-à-dire

$$x_{k+1} = x_k - (g'(x_k)^T g'(x_k) + \lambda I)^{-1} \nabla f(x_k)$$

J'ai écrit un programme Scilab pour afficher les points successifs x_k et x_{k+1} en essayant d'ajuster au mieux le paramètre λ. La condition initiale de x_0 est la même que dans la partie 2.1.2.

```
function out=g(x)
  out = [1 - x(1)
       10*(x(2) - x(1)^2)];
endfunction

function jac=gprime(x)
  jac = [-1,0
       -20*x(1),10];
endfunction

function out=grad(x)
  out = gprime(x)'*g(x);
endfunction

x = zeros(2,4000);
x(:,1) = [0.5;1.5];
lambda = 0.01;
I = eye(2,2);

for k = 1 : 3999
      jac = gprime(x(:,k));
      x(:,k+1) = x(:,k)-(jac'*jac+lambda*I)\ grad(x(:,k));
      if (norm(gradient)) < 1e-6
            break;
      end;
end;
```

CODE SOURCE 2.1.3 - la méthode de Levenberg-Marquardt pour $\lambda = 0.01$

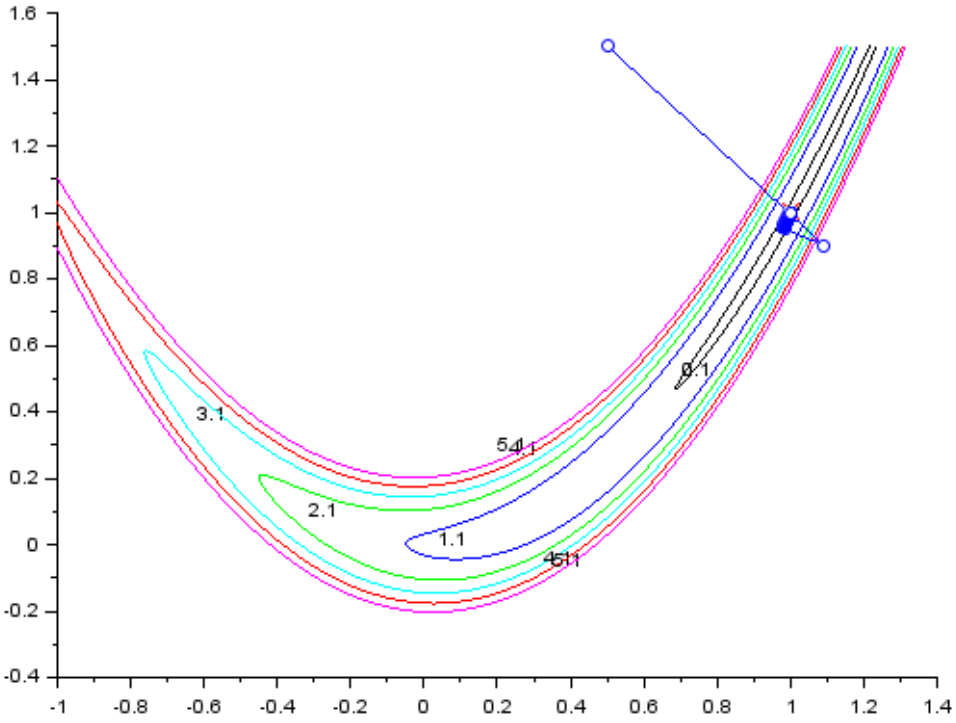


FIGURE 2.1.3(1) - la méthode de Levenberg-Marquardt pour $\lambda = 10$

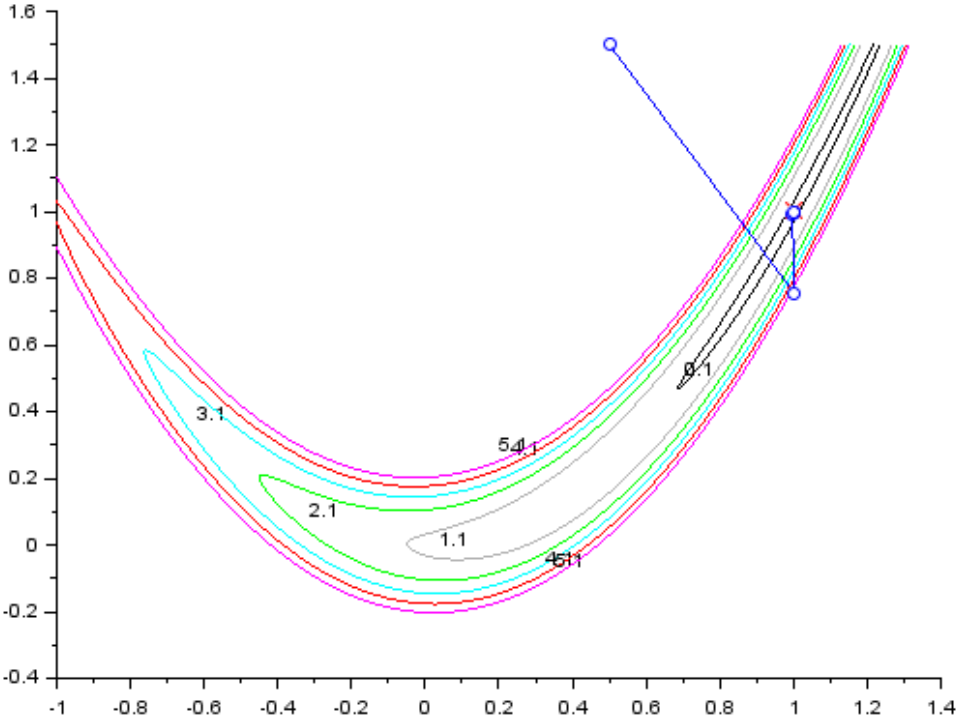


FIGURE 2.1.3(2) - la méthode de Levenberg-Marquardt pour $\lambda = 0.01$

Pour $\lambda_1 = 20$, et $\lambda_2 = 10$, et $\lambda_3 = 0.1$, il a fallu respectivement que $k_1 = 892, k_2 = 471, k_3 = 7$ itérations pour que $\|\nabla f(x_k)\| \leq 10^{-6}$. En effet, cette méthode est très efficace uniquement si on choisit bien le paramètre λ, le choix du paramètre λ est crucial. Cependant, elle est instable car le rang maximal de la matrice $g'(x_k)^T g'(x_k)$ n'est pas garanti, lorsque λ est trop petit voire nul.

2.2. Problème de régression non-linéaire

On considère m couples $(t_i, y_i)_{i=1..m}$. On cherche à approcher au mieux ces données à l'aide de la fonction

$$f(t) = \exp(a + bt + ct^2)$$

de manière à ce que la quantité

$$E(a, b, c, d) = \sum_{i=1}^{m} (f(t_i) - y_i)^2$$

soit minimale.

Tout d'abord, j'ai résolu ce problème de moindres carrés en utilisant le « log trick » vu en cours, en posant :

$$S_{log}(a, b, c) = \sum_{i=1}^{m} (\log(y_i) - (a + bt + ct^2)^2 = \| r(a, b, c) \|^2,$$

où $r(\theta)$ est le vecteur résiduel du modèle

$$r(a, b, c) = \begin{pmatrix} 1 & t_1 & t_1^2 \\ \vdots & \vdots & \vdots \\ 1 & t_m & t_m^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} - \begin{pmatrix} \log(y_1) \\ \vdots \\ \log(y_m) \end{pmatrix}$$

On trouve $(\widehat{\mathrm{a}, \mathrm{b}, \mathrm{c}})$ tel que :

$$(\widehat{\mathrm{a}, \mathrm{b}, \mathrm{c}}) = \arg\min S(\widehat{\mathrm{a}, \mathrm{b}, \mathrm{c}})$$

J'ai résolu ensuite ce problème directement grâce à la macro $lsqrsolve$ de Scilab. La macro $lsqrsolve$ nous permet de minimiser $E(a, b, c, d)$ en utilisant la méthode Levenberg-Marquardt. Enfin, j'ai comparé les résultats obtenus en représentant à l'écran les points $(t_i, y_i)_{i=1..m}$ et les graphes des fonctions obtenues.

```
load dateExp.sci;
t = data(:, 1);
y = data(:, 2);
[t, k]  =  gsort(t);
y  =  y(k);
plot(t,y,'o');

A = [ones(t)  t  t.^2];
theta0  =  A \  log(y);
plot(t, exp(A * theta0), 'r--');

function out=resid(x, m)
   a = x(1);
   b = x(2);
   c = x(3);
   out = exp(a + c * t + b * t.^2) - y;
endfunction

 [theta, v, info] = lsqrsolve(theta0, resid, size(t, 1))
plot(t, exp(theta(1) + theta(2) * t + theta(3) * t.^2), 'r');
```

CODE SOURCE 2.2 – Régression non-linéaire

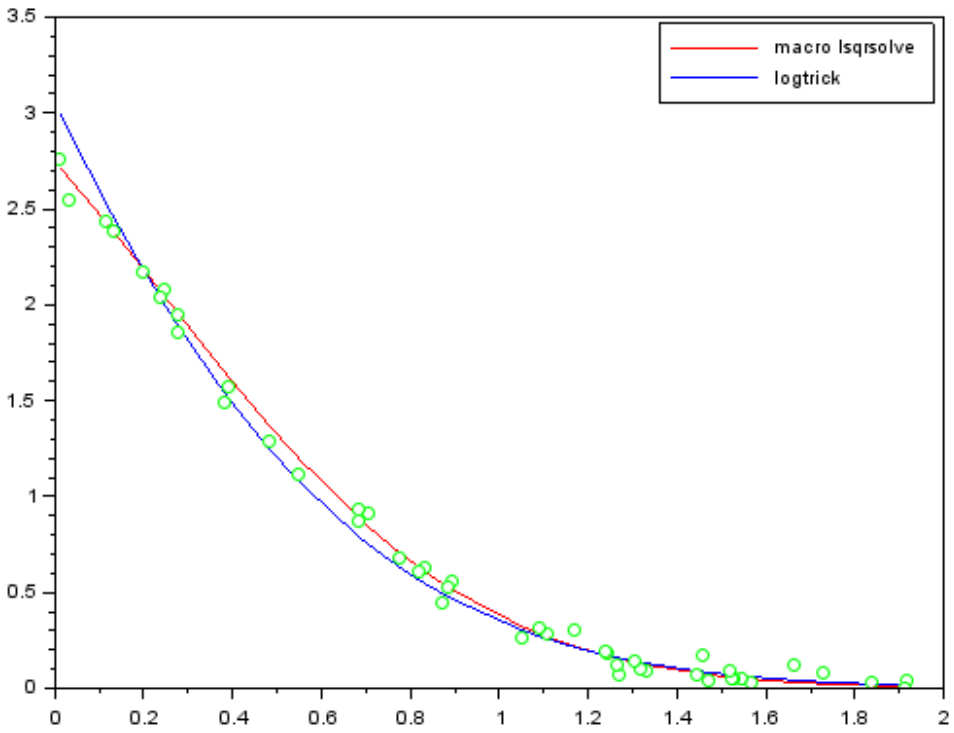


FIGURE 2.2 – Régression non-linéaire

2.3. Problème de cinématique inverse

On considère un bras robot articulé dans le plan (x_1, x_2), d'origine O, avec un premier segment de longueur l_1 et faisant un angle θ_1 avec (O, x_1), un deuxième segment de longueur l_2 faisant un angle $\theta_2 - \theta_1$, avec le premier segment et un troisième segment de longueur l_3 faisant un angle $\theta_3 - \theta_2$ avec le deuxième segment. L'extrémité du bras a donc pour coordonnées

$$M(\theta) = \begin{pmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_2) + l_3 \cos(\theta_3) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_2) + l_3 \sin(\theta_3) \end{pmatrix}$$

Soit $A = (x_A, y_A)^T$, on cherche à déterminer le vecteur $\theta = (\theta_1, \theta_2, \theta_3)^T$ tel que

$$M(\theta) = A$$

tout en essayant de minimiser la déformation du bras. Pour cela on va considérer le problème d'optimisation

$$\theta = \arg\min f(\theta)$$

où

$$f(\theta) = \|M(\theta) - A\|^2 + \lambda^2((\theta_2 - \theta_1)^2 + (\theta_3 - \theta_2)^2) = \|g(\theta)\|^2$$

On en déduit que

$$g(\theta) = \begin{pmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_2) + l_3 \cos(\theta_3) - x_A \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_2) + l_3 \sin(\theta_3) - y_A \\ \lambda(\theta_2 - \theta_1) \\ \lambda(\theta_3 - \theta_2) \end{pmatrix}$$

On prend $l_1 = l_2 = 1, \lambda = 10^{-1}$. J'ai écrit un programme Scilab calculant le vecteur θ minimisant $f(x) = \|g(\theta)\|^2$ en utilisant la fonction *lsqrsolve* et ensuite représentant les positions successives du bras lorsque le point A est défini par une courbe paramétrique :

$$A(t) = \begin{cases} x_1(t) = 1 + \frac{1}{2}\cos(t) \\ x_2(t) = 1 + \frac{1}{2}\sin(t) \end{cases}$$

```
function out=g(theta, m)
   out = [l(1) * cos(theta(1)) + l(2) * cos(theta(2)) + l(3) * cos(theta(3)) - xA
          l(1) * sin(theta(1))  + l(2) * sin(theta(2)) + l(3) * sin(theta(3))  - yA
          lambda * (theta(2) - theta(1))
          lambda * (theta(3) - theta(2))];
endfunction


l = [1 1 1];
lamda = 0.1;
x = [];
y = [];
for i = 0 :  2*%pi/500  :  2*%pi
   xA = 1 + 0.5 * cos(i);
   yA = 1 + 0.5 * sin(i);
   theta0 = [0 0 0];
   [theta, v, info] = lsqrsolve(theta0, g, 4);
   x = [x, l(1) * cos(theta(1)) + l(2) * cos(theta(2)) + l(3) * cos(theta(3))];
   y = [y, l(1) * sin(theta(1)) + l(2) * sin(theta(2)) + l(3) * sin(theta(3))];
end
plot(x, y, 'o');

M = [[0 0]];          //construire M : ensemble des nœuds du bras
   for j = 1 : 3
         M = [M; [M(j, 1) + l(j) * cos(theta(j)) , M(j, 2) + l(j) * sin(theta(j))] ];
   end;
plot(M(:, 1), M(:, 2),'r-');
```

FIGURE 2.3 – Cinématique inverse

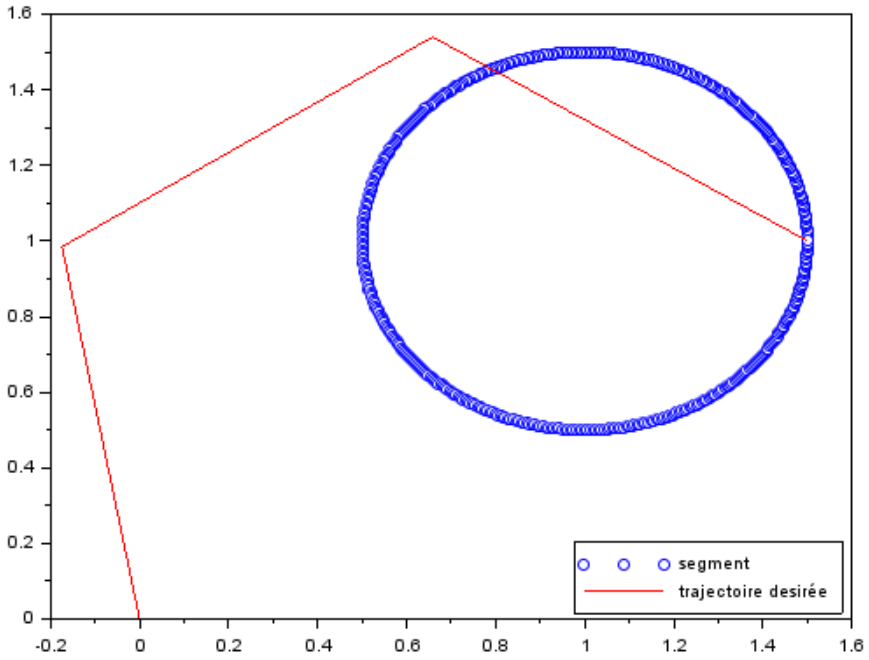


FIGURE 2.3 – Bras robot de trois articulations

CHAPITRE 5

Valeurs propres

1. Méthodes d'approximation des valeurs propres

1.1. Méthode de la puissance

Soit A une matrice carrée de dimension $n * n$, on suppose que ses valeurs propres vérifient

$$|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$$

λ_1 est nécessairement réelle. On note $\{x_1, \dots, x_n\}$ les vecteurs propres de A, avec $\|x_i\|$ = 1 et on suppose que A est diagonalisable. Cette méthode nous permet de déterminer la valeur propre de module maximal d'une matrice d'ordre n.

Algorithme de la puissance :

x_0 donné

$$y_k = A * x_{k-1}$$

$$x_k = \frac{y_k}{\|y_k\|}$$

$$\lambda_k = (x_k)^T * Ax_k$$

Propriété : Si x_0 et y_1 ne sont pas orthogonaux, alors :

$$\lim_{k \to +\infty} signe(\lambda_1)^k x_k = \pm y_1$$

1.2. Méthode de la puissance inverse

Cette méthode nous permet de déterminer la valeur propre de module maximal d'une matrice d'ordre n. Cest-à-dire on trouve $\lambda_n \neq 0$ et

$$|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$$

on suppose que λ_n réelle

Remarque : Si λ est valeur propre de A et y est le vecteur propre associé on a : $Ay = \lambda y$ et si A inversible alors

$$A^{-1}(Ay) = \lambda A^{-1}y$$

$$y = \lambda A^{-1}y$$

$$\Rightarrow A^{-1}y = \frac{1}{\lambda}y$$

Algorithme de la puissance inverse :

Remarque préliminaire : $A^{-1}x_k$ est le vecteur y_{k+1} tel que $Ay_{k+1} = x_k$ donc il suffit de résoudre un système linéaire à chaque itération

x_0 donné

$y_k = x_{k-1}/A$

$x_k = \dfrac{y_k}{\|y_k\|}$

$\lambda_k = (x_k)^T * y_k$

La valeur propre de module maximal de la matrice A est $\frac{1}{\lambda}$.

Propriété :

$$\lim_{k \to +\infty} signe(\lambda_n)^k x_k = \pm y_n$$

2. Résonance dans les systèmes mécaniques discrets

2.1. Un système 2 masses + 2 ressorts

On considère un système 2 masses + 2 ressorts, donc un ressort de raideur k_1 dont une extrémité est fixée et d'ont l'autre, portant une masse m_1 pouvant glisser sans frottement sur un

axe. On connecte un autre couple masse m_2 + ressort k_2, et on fait agir la force $F(t)$ sur la deuxième masse.

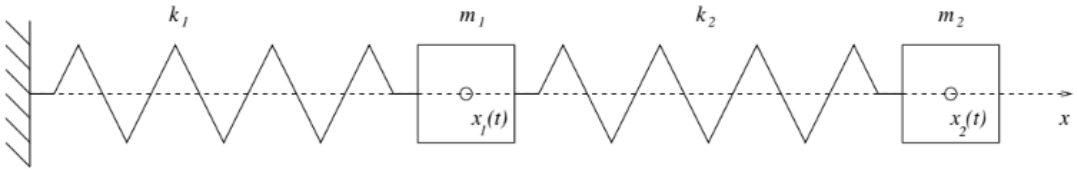


FIGURE 2 – Système mécanique à deux degrés de liberté

On peut montrer que les équations régissant les positions $x_1(t)$ et $x_2(t)$ des masses sont les suivantes

$$\begin{cases} m_1 u_1'' = -(k_1 + k_2)u_1 + k_2 u_2 \\ m_2 u_2'' = k_2 u_1 - k_2 u_2 + f(t) \end{cases} \tag{1}$$

où $f(t) = \sin \omega t$ et $u_1 = x_1 - L_1$ et $u_2 = x_2 - L_2$, L_1 et L_2 étant respectivement les longueurs au repos des ressorts 1 et 2. Si on pose $u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$, on peut alors récrire matriciellement ces deux équations sous la forme

$$u'' = -Au + \begin{pmatrix} 0 \\ \frac{1}{m_2} f \end{pmatrix} \tag{2}$$

où

$$A = \begin{pmatrix} \frac{k_1 + k_2}{m_1} & -\frac{k_2}{m_1} \\ -\frac{k_2}{m_2} & \frac{k_2}{m_2} \end{pmatrix}$$

Pour un système à deux degrés de liberté, il suffit en fait de pouvoir diagonaliser la matrice A. En effet, si on est capable de trouver une matrice P telle que

$$PAP^{-1} = D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

où λ_1 et λ_2 sont les valeurs propres de A, il suffit de pose le changement de variable

$$v = Pu$$

et dans ce cas l'équation (2) se récrit

$$v'' = -Du + g$$

où

$$g(t) = P\begin{pmatrix} 0 \\ \frac{1}{m_2} f(t) \end{pmatrix}$$

soit le système différentiel

$$\begin{cases} v_1'' + \lambda_1 u_1 = g_1(t) \\ v_2'' + \lambda_1 u_1 = g_1(t) \end{cases} \quad (3)$$

Il est clair que les deux équations du système (3) sont découplées, et donc qu'elles peuvent être considérées indépendamment l'une de l'autre (ce qui n'était pas possible avant le changement de variable). Si l'on admet que λ_1 et λ_2 sont positives, alors il est clair

$$\omega_1 = \sqrt{\lambda_1}\,, \omega_2 = \sqrt{\lambda_2}$$

2.2. Travail à réaliser

Dans cette partie on va chercher à illustrer le phénomène de résonance avec le système à deux degrés de liberté vu précédemment.

2.2.1. Détermination des valeurs propres

On prend $k_1 = 5, k_2 = 10, m_1 = m_2 = 0.1$. On note λ_1 et λ_2 les valeurs propres de A et on suppose que $|\lambda_1|>|\lambda_2|$. J'ai écrit un programme Scilab déterminant λ_1 et λ_2, ainsi que les vecteurs propres associés y_1 et y_2 à l'aide de la méthode de la puissance itérée et la méthode de la puissance itérée inverse, et puis vérifiant les résultats avec la macro *spec* de Scilab :

```
k1 = 5;
k2 = 10;
m1 = 0.1;
m2 = 0.1;

A = [ (k1+k2)/m1,  -k2/m1
      -k2/m2      ,  k2/m2];


//algorithme de la puissance
x1 = rand(2 , 1);
for k = 1 : 100
    y = A * x1;
    lambda1 = x1' * y;
```

```
    if norm(y - lambda1 * x1) < 1e-10
        x1 = y / norm(y);
        break
    end
    x1 = y / norm(y);
end

//algorithme de la puissance inverse
x2 = rand(2 , 1);
for k = 1 : 100
    y = A \ x2;
    lambda2 = x2' * y;
    if norm(y - lambda2 * x2) < 1e-10
        x2 = y / norm(y);
        break
    end
    x2 = y / norm(y);
end
lambda2 = 1 / lambda2;

disp(lambda1 , lambda2);
[d , p] = spec(A);
disp(d , p);
```

CODE SOURCE 2.2.1 – Détermination des valeurs propres

On obtient

$$\lambda_1 = 228.07764, \lambda_2 = 21.922359\ ,$$

2.2.2. Simulation

1. On va simuler le système (3) pour $t \in [0,10]$ avec la macro *ode* en prenant un pas de temps $h = 0.05$ pour $\sin(\omega_1 t)$ et puis $\sin(\omega_2 t)$ où

$$\omega_1 = \sqrt{\lambda_1} = , \omega_2 = \sqrt{\lambda_2}$$

Après visualiser l'animation les mouvements du système, j'ai constaté que les mouvements de deux ressorts sont sinusoïdes, cependant, pour ω_1, les phases des mouvements sont différentes, pour ω_2, ils sont en même phase :

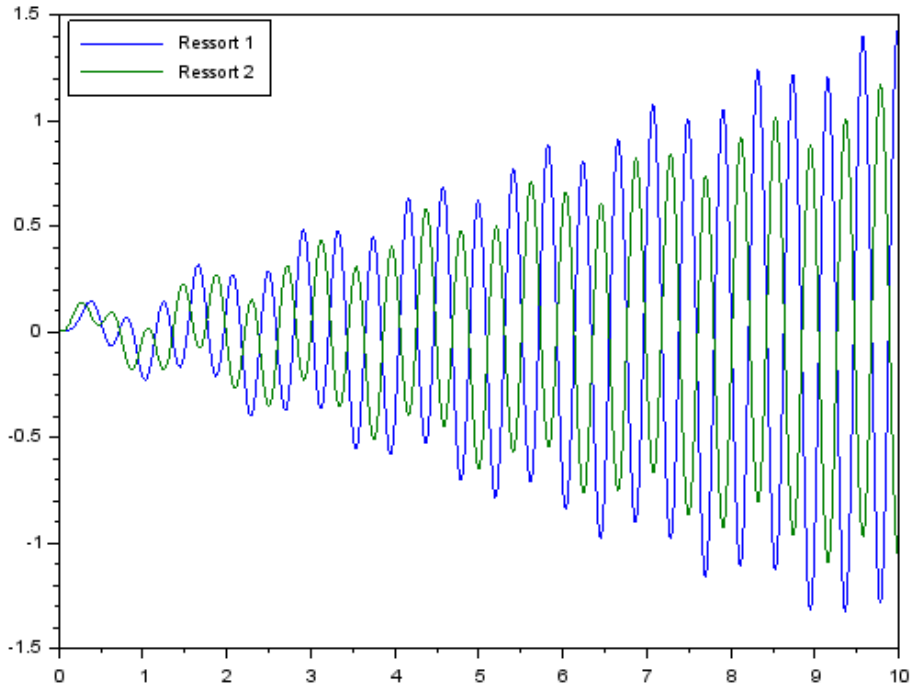


FIGURE 2.2.2(1) - Graphique des mouvements de deux ressorts en prenant ω_1.

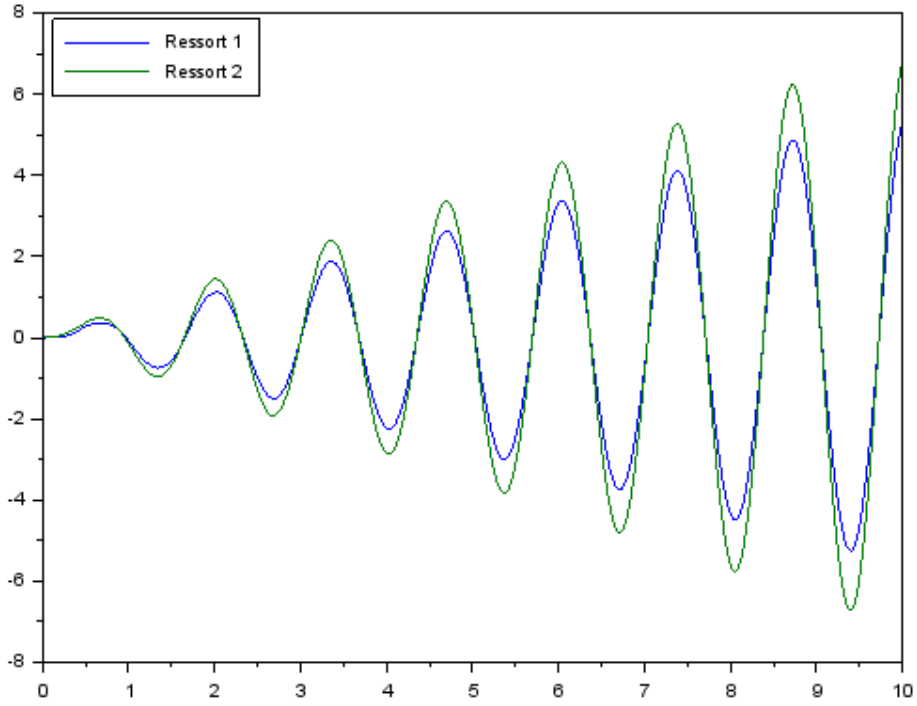


FIGURE 2.2.2(1) - Graphique des mouvements de deux ressorts en prenant ω_2.

```
function Uprime=fRessort(t, U)
   u     = U(1:2);
   uprime = U(3:4);
   Uprime = [ uprime
            -A * u + [1; 1/m2] * sin(omega * t)];
endfunction

omega = sqrt(lambda1);
t = linspace(0,10,1000);
U = ode(zeros(4,1), 0, t, fRessort);
clf
anime_os(t,U,m1,m2,5,5);
plot(t, U(1:2,:));
legend("Ressort 1","Ressort 2",2);
```

CODE SOURCE 2.2.2(1) – Code source de la simulation 1

2. On va simuler en suite le système pour $f(t)$ définie par

$$\begin{cases} f(t) = 1, t \in [0, T_i/2[\\ f(t) = 0, t \in [T_i/2, T_i[\end{cases}$$

Et $f(t + kT_i) = f(t), k \in \mathbb{N}$ avec $T_i = \frac{2\pi}{\omega_i}, i = 1,2$. Après visualiser l'animation les mouvements du système, j'ai constaté que les mouvements de deux ressorts sont sinusoïdes, cependant, pour ω_1, les phases des mouvements sont différentes, pour ω_2, ils sont en même phase, comme le système au-dessus

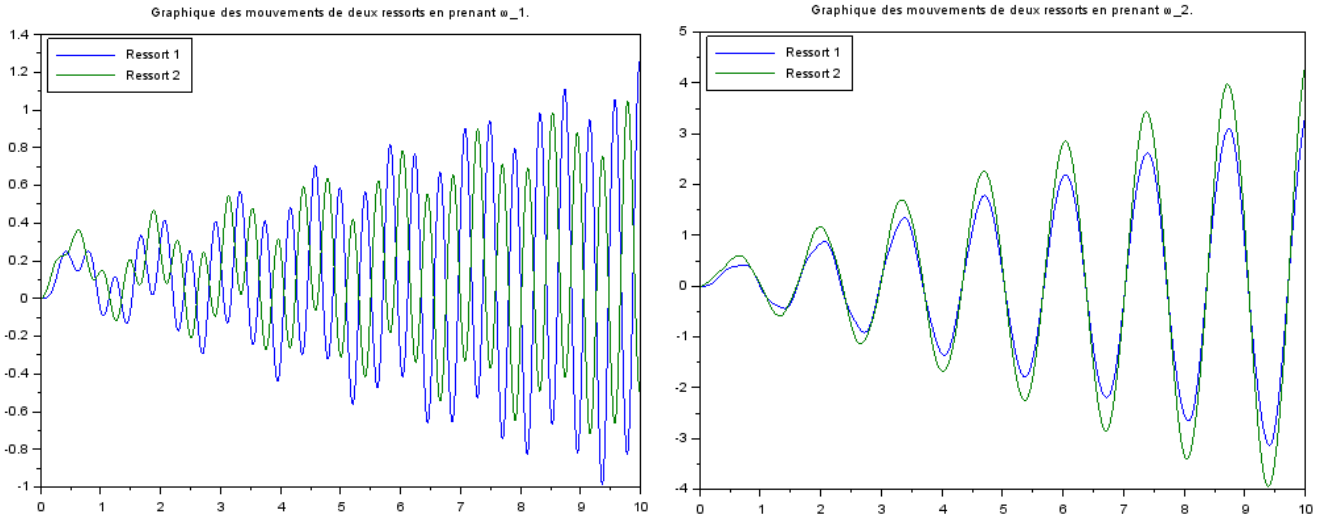


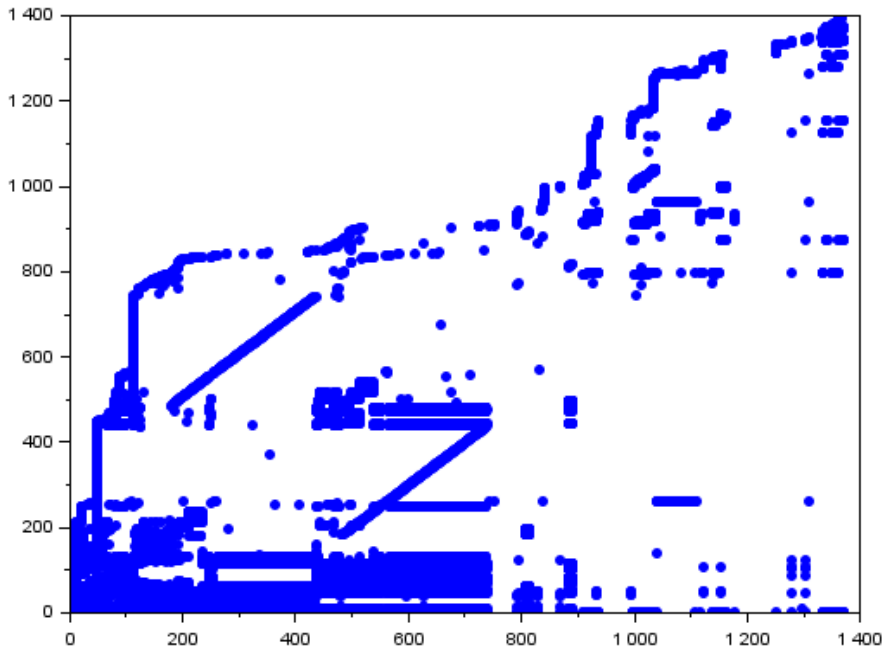
FIGURE 2.2.2(3) - Graphique des mouvements de deux ressorts en prenant ω_1, ω_1 respectivement.

```
function out=f(o, t)
        T = 2*%pi/o;
        t = modulo(t,T);
        if (0 <= t) && (t < T/2)
                out = 1
        else
                out = 0;
        end
endfunction

function Uprime=fRessort(t, U)
       u     = U(1:2);
       uprime = U(3:4);
       Uprime = [ uprime
                        -A * u + [0; 1/m2] * f(omega,t)];
endfunction

omega = sqrt(lambda1);
t = linspace(0,10,1000);
U = ode(zeros(4,1), 0, t, fRessort);
clf
anime_os(t,U,m1,m2,5,5);
plot(t, U(1:2,:));
legend("Ressort 1","Ressort 2",2);
title("Graphique des mouvements de deux ressorts en prenant ω_1.")
```

CODE SOURCE 2.2.2(2) – Code source de la simulation 2

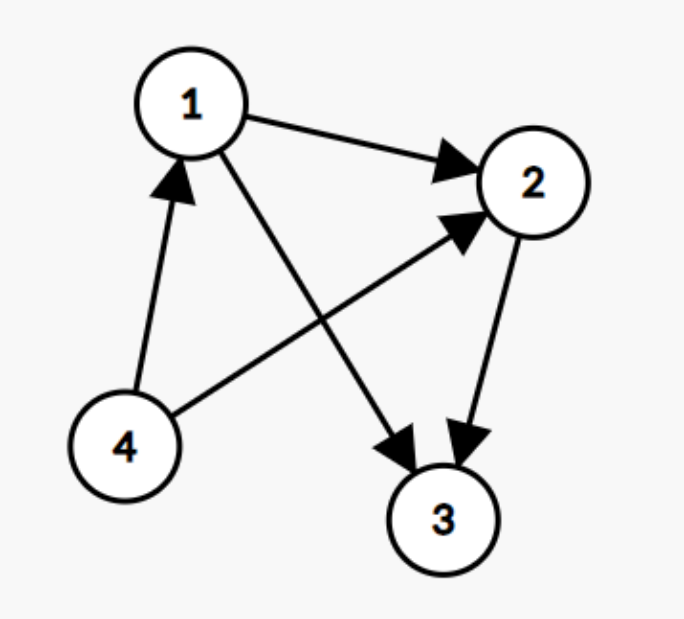
3. Algorithme PageRank

Le PageRank est l'algorithme d'analyse des liens concourant au système de classement des pages web utilisé par le moteur de recherche de Google. Il mesure quantitativement la popularité d'une page web. Il fait intervenir une énorme matrice, calculée à partir de la matrice d'adjacence C du graphe représentant les pages ainsi que les liens entre elles. Par exemple, la figure suivante représente la matrice C pour corpus des 1248 pages interconnectées obtenues à partir de l'URL *http://www.utc.fr* :



Cette matrice est creuse car il n'y a que 41876 éléments non nuls sur un total de 1557503 éléments (soit un taux de remplissage de 2,6%).

On considère le web comme un graphe orienté avec un ensemble des sommets (pages web), $E = \{x_1, x_2, .. , x_n\}$ est les arcs représentant les liens hypertextes. Par exemple :



Ici on a la matrice d'adjacence C, telle que $C_{i,j} = 1$ si la page i a un lien vers la page j :

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Algorithme d'un promeneur du web :

En une page donnée j

- S'il n'y a pas de lien sur la page, le promeneur choisit aléatoirement une autre page (donc avec la probabilité $\frac{1}{n}$)
- S'il y a des liens sur la page :
  (1) Soit le promeneur choisit l'une de ces pages aléatoirement donc avec la probabilité $1/\sum_{i=1}^{n} C_{i,j}$
  (2) Soit le promeneur choisit une page du web aléatoirement (probabilité $\frac{1}{n}$)
    (1) Avec une probabilité q proche de 1 mais inférieure
    (2) Avec une probabilité $1-q$

La matrice de Google $P = (p_{i,j})$ exprimant la probabilité $p_{i,j}$ que l'on passe à la page i de la page j est calculée à partir de C de la manière suivante :

$$p_{i,j} = q\frac{c_{i,j}}{\sum_{i=1}^{n} c_{i,j}} + \frac{1-q}{n} \; si\ la\ colonne\ C_j\ est\ non-nulle$$

$$p_{i,j} = \frac{1}{n}\ si\ la\ colonne\ C_j\ est\ nulll$$

On peut écrire cette matrice sous la forme

$$P = qC\lambda + \frac{1}{n}e(e-qf)^T$$

Où

- λ est une matrice diagonale définie par $\lambda_{i,j} = (\sum_{i=1}^{n} c_{i,j})^{-1}$ si la colonne C_j est non-nulle et $\lambda_{i,j} = 1$ sinon.
- Le vecteur $e = (1, \dots, 1)^T$ et le vecteur f avec $f_i = 0$ si la colonne C_j est non-nulle et $f_i = 1$ sinon.
- On prend $q = 0.85$.

A l'aide de l'algorithme de la puissance, on détermine la valeur propre dominante de P ainsi que le vecteur propre x associé. Il suffit de noter que :

$$Px = qC\lambda x + \frac{1}{n}e(e-qf)^T x$$

```
n = size(C, 1);
s = sum(C , 1);
s(s <> 0) = 1../s(s <> 0);
lambda = diag(s);
e = ones(n , 1);
f = ones(n , 1);
f(s == 0) = 0;
x = ones(n,1)/n;
q = 0.85;

for k = 1 : 50
   y = q * C * lambda * x + (1/n) * e * (e - q * f)' * x;
   vp = x' * y;
   x = y / norm(y);
end

x = x/sum(x);
[xs,k] = gsort(x)
```

CODE SOURCE 3 – L'algorithme PageRank

CHAPITRE 6

Séries de Fourier

1. Rappels de cours

Définition (Série de Fourier) : soit f une fonction T périodique. On pose $\omega = \frac{2\pi}{T}$, On appelle série de Fourier associée à f la série trigonométrique suivante :

$$Sf(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(n\omega t) + b_n \sin(n\omega t))$$

où l'on a :

$$\frac{a_0}{2} = \frac{1}{T}\int_{-T/2}^{T/2} f(t)dt$$

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)cos(n\omega t)dt$$

$$b_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)sin(n\omega t)dt$$

Remarque :

- Si f est paire, alors $b_n = 0$
- Si f est impaire, alors $a_n = 0$

2. Le phénomène de Gibbs

On considère la fonction $f: \mathbb{R} \to \mathbb{R}$ périodique de période 2π et vérifiant

$$\begin{cases} f(x) = -1 \ , \ x \in]-\pi, 0[\\ f(x) = 1 \ , \ x \in [0, \pi[\end{cases}$$

f est impaire sur l'intervalle $]-\pi, \pi[$, donc $a_n = 0$

$$b_n = \frac{2}{T}\int_{-\pi}^{\pi} f(t)\sin(n\,\omega t)dt$$

$$\Leftrightarrow \quad b_n = \frac{1}{\pi}\left(\int_{-\pi}^{0} \sin(n\,\omega t)dt + \int_{0}^{\pi} \sin(n\,\omega t)dt\right)$$

$$\Leftrightarrow \quad b_n = \frac{1}{\pi}(1 - \cos(-n\omega t) - \cos(n\omega t) + 1)$$

$$\Leftrightarrow \quad b_n = \frac{2}{n\pi}(1 - \cos(n\omega t)) \qquad car\ \omega = \frac{2\pi}{T} = 1$$

$$\Leftrightarrow \quad b_n = \begin{cases} 0 & si\ n\ pair \\ \dfrac{4}{n\pi} & si\ n\ impair \end{cases}$$

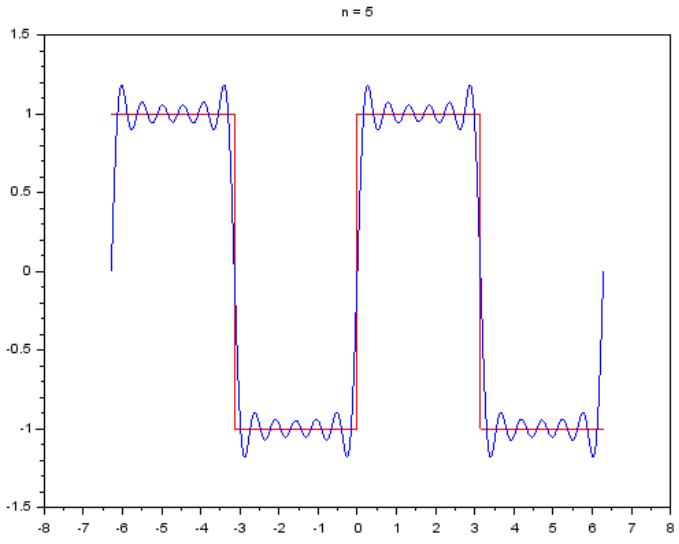
D'où

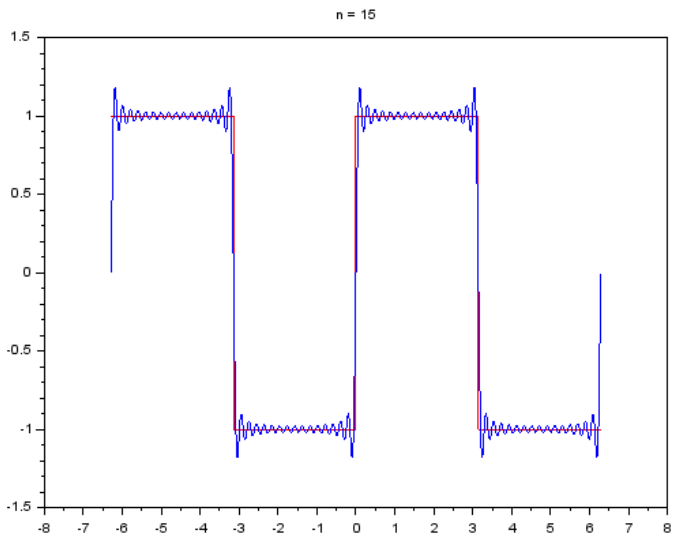
$$S_n(x) = \sum_{p\geq 0}^{2p+1\leq n} \frac{4}{(2p+1)\pi}\sin((2p+1)x)$$

Ensuite, on va écrire un programme Scilab représentant $S_n(x)$ pour des valeurs croissantes de n et observer le phénomène de Gibbs

```
// Trace f
function out = f(t, T)
   out  =  ones(t) * -1;
   tmp = modulo(t,T);
   tmp = modulo(tmp+T,T);
   out(tmp <= %pi) = 1;
endfunction

T = 2*%pi;
t = -2*%pi : 0.0001 : 2*%pi;
plot(t, f(t),'r');

//Representer S
n = 10;
S = zeros(t);
for p = 0 : n
   S = S + sin(( 2 * p + 1) * t) / (2 * p + 1);
end
S = 4 * S/%pi;
plot(t,S);
title('n = 10');
gca().data_bounds =  [-2*%pi, 2*%pi, -1.5, 1.5];
```

CODE SOURCE 2 – Phénomène de Gibbs





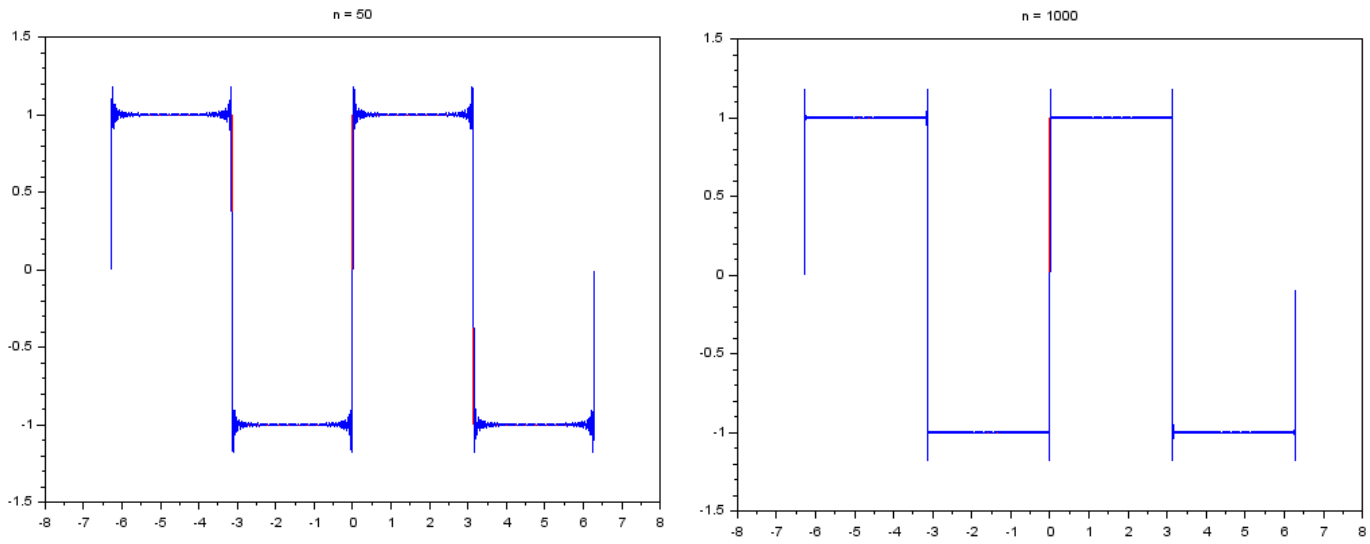


FIGURE 2 - La fonction f et la fonction S_n pour $n = 5, 15, 50, 1000$

3. Régularité et décroissance des coefficients

On considère la fonction f définie sur $[0,1/2]$ par

$$f(x) = x(x-1)$$

que l'on cherche à représenter à l'aide d'une série de Fourier. On considère les trois fonctions suivantes, qui coïncident avec f sur [0,1/2] :

- f_1 périodique de période 1/2 vérifiant : $\forall x \in [0,1/2]\,, f_1(x) = f(x)$
- f_2 paire et périodique de période 1 vérifiant: $\forall x \in [0,1/2]\,, f_2(x) = f(x)$
- f_3 impaire et périodique de période 2 vérifiant: $\forall x \in [0,1/2]\,, f_3(x) = f_2(x)$

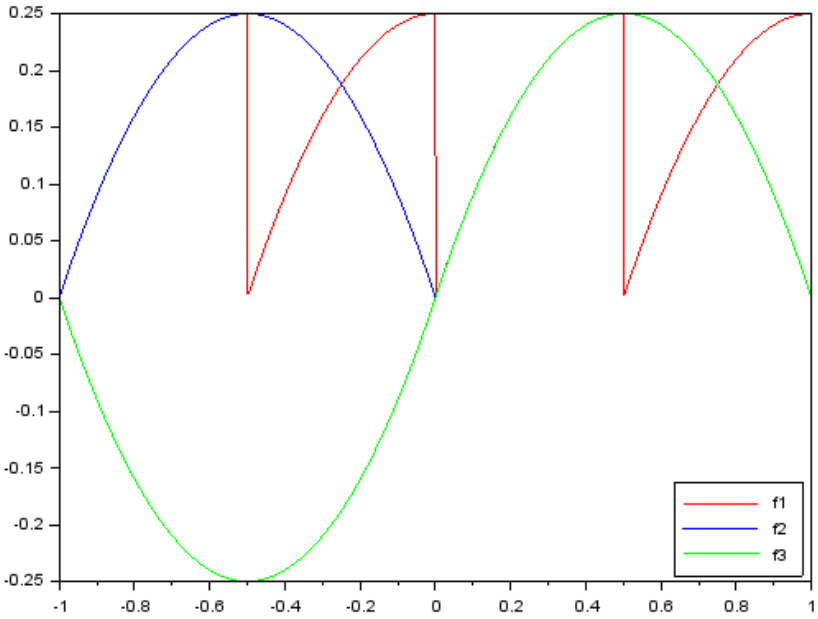
On commence par calculer les coefficients de la série de Fourier de f_1, f_2, f_3 :

	a_k	b_k	ω	a_0	T
f_1	$\frac{-1}{(2k\pi)^2}$	$\frac{-1}{4k\pi}$	4π	$\frac{1}{3}$	$\frac{1}{2}$
f_2	$\frac{-1}{(k\pi)^2}$	0	2π	$\frac{1}{3}$	1
f_3	0	$0\ si\ k\ pair$ $\frac{8}{(k\pi)^3}\ si\ k\ impair$	π	0	2

J'ai écrit ensuite un programme Scilab pour représenter simultanément les sommes partielles des séries de Fourier des fonctions f_1, f_2, f_3 sur $[0,1/2]$

```
n = 4000;
a = zeros(3,n);
b = zeros(3,n);
for p = 1 : n
    a(1,p) = ((2*%pi*p).^2)\-1;
    a(2,p) = ((p*%pi).^2)\-1;
    a(3,p) = 0;
    b(1,p) = (4*p*%pi)\-1;
    b(2,p) = 0;
    if (modulo(p,2) == 0)
       b(3,p) = 0
    else
       b(3,p) = ((%pi*p).^3)\8;
    end;
end

x = -1 : 0.001 : 1;
omega = [4*%pi, 2*%pi, %pi];
a0 = [1/3 , 1/3 , 0];

for i = 1 : 3
  S = zeros(x);
  for k = 1 : n
     S = S + a(i,k)*cos(k*omega(i)*x) + b(i,k)*sin(k*omega(i)*x);
  end
  S = S + a0(i)/2;
  if (i == 1)
     plot(x,S,'r');
  else
     if (i == 2) then
        plot(x,S,'b');
     else
        plot(x,S,'g');
     end
  end
end
```

CODE SOURCE 3 – Régularité et décroissance des coefficients



CODE SOURCE 3 – Régularité et décroissance des coefficients

4. Equation de la chaleur

On cherche à résoudre l'équation aux dérivées partielles suivantes,

$$\frac{\partial u}{\partial t}(x,t) - \frac{\partial^2 u}{\partial x^2}(x,t) = 0\ , x \in]0,2\pi[\ , t > 0$$

avec les conditions aux limites de périodicité de la solution

$$u(0,t) = u(2\pi,t)$$

$$\frac{\partial u}{\partial x}(0,t) = \frac{\partial u}{\partial x}(2\pi,t)$$

et la condition initiale

$$u(x,0) = f(x)$$

avec f est définie par

$$f(x) = \begin{cases} 1 \ , & si\ x \in [\pi - \lambda, \pi + \lambda] \\ 0 \ , & sinon \end{cases}$$

et $\lambda \in]0,\lambda[$.

La fonction f est paire sur l'intervalle $[\pi - \lambda, \pi + \lambda]$ donc $b_k = 0$.

$$a_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos(kx)\,dx = -\frac{2}{k\pi}\sin\big(k(\pi-\lambda)\big) = \frac{2\sin(k\lambda)}{k\pi}\,, \qquad k > 0$$

$$a_0 = \frac{\lambda}{\pi}$$

D'où

$$u(x,t) = \frac{a_0}{2} + \sum_{k>0} a_k \exp(-k^2 t)\cos(kx)\,dx\,, \qquad t > 0$$

J'ai commencé par calculer les coefficients de la série de Fourier de f en prenant $\lambda = 1$:

```
n = 4000;
x = -%pi : %pi/4/n : %pi;
lambda = 1;

S = lambda/%pi;

k = 1 : n;
a = -2 * sin(k*(%pi-lambda))./k/%pi;

for k = 1 : n
   S = S + a(k)*cos(k*x);
end

clf
plot(x,S);
```

CODE SOURCE 4(1) - La série de Fourier de f en prenant $\lambda = 1$

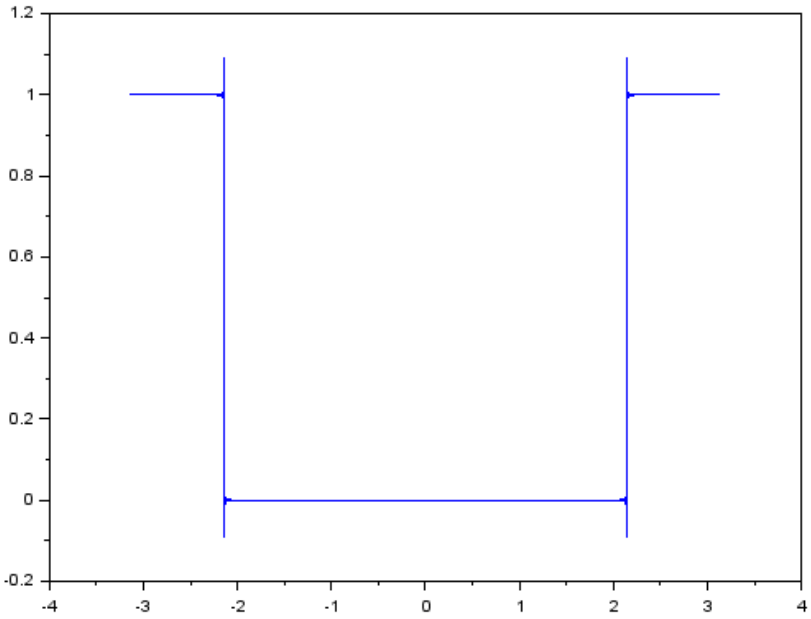


FIGURE 4(1) - la série de Fourier de f en prenant $\lambda = 1$

Ensuite, j'ai représenté $u(x,t)$ pour diverses valeurs de $\lambda \in]0,\lambda[$:

```
n = 400;
x = 0 : %pi/4/n : 2*%pi;
lambda = 1;
k = 1 : n;
a = -2 * sin(k*(%pi-lambda))./k/%pi;

for t = 0.001 : 0.001 : 5; // rand
   S = lambda/%pi;
   for k = 1 : n
      S = S + a(k)*exp(-k^2*t)*cos(k*x);
   end
   drawlater
   clf
   plot(x,S);
   gca().data_bounds(3:4) = [0,1];
   drawnow
end
```

CODE SOURCE 4(2) – Représentation $u(x,t)$

CONCLUSION

MT94 a été pour moi une unité de valeur très enrichissante. En effet, j'y ai appris de nombreuses choses concernant les applications des mathématiques, que sa soit en mécanique ou en informatique. Il a été particulièrement intéressant de voir à quel à quel point les mathématiques fondamentales que nous avons vu en MT90/91, MT22 et MT23 jouent un rôle essentiel derrière toutes les technologies numériques. De plus, cette unité de valeur a été pour moi l'occasion de découvrir un logiciel de calcul numérique Scilab.