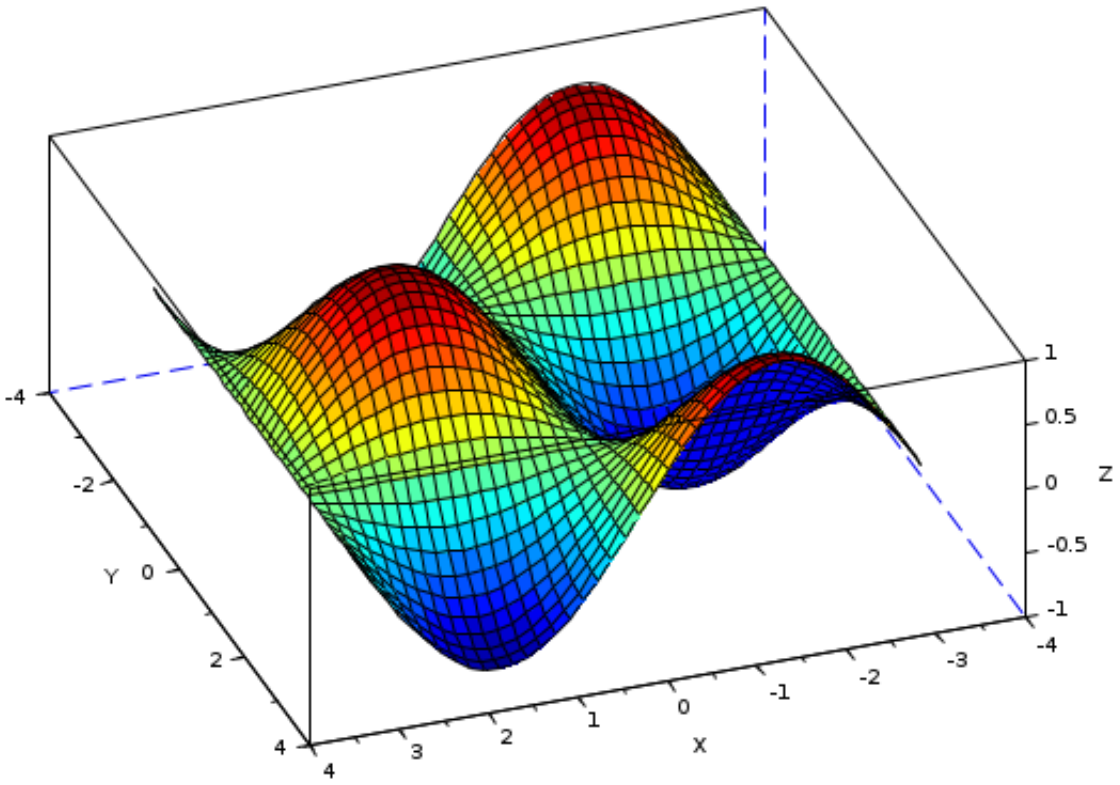
DUPARCHY Emilien, TC04

Cahier d'integration MT94

-Mathématiques appliquées-



Introduction :

Dans ce cahier d'intégration, je présenterai les notions de l'UV MT94 vues en cours, les applications réalisées en TD ainsi que mes recherches personnelles et les travaux complémentaires que j'ai réalisés. De nature pragmatique, j'ai beaucoup aimé appliquer des notions théoriques de mathématiques pour résoudre des problèmes concrets et cela m'a aussi permis de mieux comprendre certains concepts. L'UV n'étant qu'une introduction aux mathématiques appliquées, je n'ai pas forcément traité tous les aspects de chaque notion, même si j'aurais aimé avoir plus de temps pour le faire.

Remerciements :

Je souhaite remercie M.Stéphane Mottelet, qui nous a enseigné l'UV MT94 avec beaucoup de passion et de pédagogie et a répondu à toutes nos questions aussi bien en présentiel qu'à distance.

Table des matières

Chapitre 1

Codage et dérivation

Dans ce chapitre seront présentés le codage utilisé par scilab et les problématiques qu'il impose sur le calcul numérique ainsi que les deux méthodes numériques vues en cours pour approcher une dérivée.

1.1 Codage des nombres réels

Dans scilab, comme souvent en informatique, les nombres sont codés en base 2 suivant la norme IEEE 754 . C'est à dire que les nombres sont des successions de 0 et de 1 chacun multipliés par une puissance de 2. Formellement cela se traduit par la formule suivante avec l'indice 2 indiquant que le nombre est en base 2 :

$$(e_k, m_k) \in [0,1], (e_n e_{n-1} ... e_0 . m_1 m_2 ...)_2 = \sum_{k=0}^{n} e_k 2^k + \sum_{k>0} m_k 2^{-k} (1.1)$$

Par exemple 1101_2 correspond à 13 en base 10 puisque $1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 = 13$

Sur scilab, le codage des nombres se fait sur 64 bits (l'ordinateur stocke des chaînes de 0 et de 1 constituées de 64 éléments) et le codage en base 2 se fait à virgule flottante (les nombres stockés en base 2 sont compris entre 1 et 2 et sont multipliés par une puissance de 2, ce qui se rapproche de l'écriture scientifique). Un exemple illustrant le codage en virgule flottante est donné ci-contre : $12.1 = 1.5125 \times 2^3$ est sera donc codé comme la multiplication de 1.5125 par 2^3.

Le premier bit des 64 bits utilisés donne le signe du nombre : le nombre est positif si ce bit vaut 0 et il est négatif sinon.
Les 11 bits suivants donnent l'exposant du nombre : ces 11 bits codent un nombre auquel on soustrait 1023 et le résultat correspond à la puissance de 2 multipliant le nombre codé.
Enfin les 52 derniers bits appelés mantisse codent le nombre compris entre 1 et 2 qui doit être multiplié par l'exposant. 0 est une exception puisqu'il est codé par une mantisse et un exposant uniquement composés de zéros.

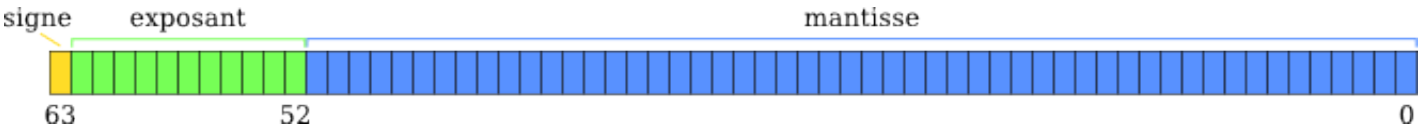


FIGURE 1.1 – schéma des nombres à virgule flottante codés sur 64 bits

Formellement, le codage se traduit par la formule suivante :

$$x = (-1)^s(1, m_1 m_2 ... m_{52})2^{e-1023} \tag{1.2}$$

Puisque les nombres sont codés par un nombre finis de bits , il existe un nombre maximal et un autre minimal pouvant étre codés. Ainsi, le premier défaut du codage est qu'il est impossible de coder une infinité de nombre puisque par définition le nombre de bits est fini. Ce nombre fini de bits engendre donc aussi des problèmes de codage des nombres réels avec des parties décimales infinies. De tels nombres sont donc approchés, c'est ce que l'on appelle la précision relative ϵ qui formellement correspond au plus petit nombre tel que $x + \epsilon \times x \neq x$ Sur Scilab, $\epsilon = 2^{-52}$.
Cette précision relative entraîne un autre phénomène appelé Perte de signification ou catastrophic cancellation qui s'exprime lorsqu'une opération entre deux nombres très proche diminue considérablement la précision. Ce phénomène s'exprime surtout durant les soustractions comme on peut l'observer dans l'exemple suivant.
Ainsi, l'utilisation du codage entraine plusieurs problèmes impactant directement les calculs numériques : les nombres pouvant êtres codés

sont bornés et non infinis, les nombres avec des parties décimales infinies sont approchés et cette approximation entraîne des pertes de signification au cours de certaines opérations mathématiques réalisées avec les nombres codés.

1.2 dérivation

1.2.1 approximation d'ordre 1

Théoriquement, la dérivée $f'(x_0)$ de $f(x_0)$ correspond à la valeur suivante :

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h} \tag{1.3}$$

Si la fonction étudiée est continûment dérivable deux fois, le développement de Taylor avec reste de Lagrange à l'ordre 2 donne :

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} + \frac{h}{2} f''(\xi), \xi \in Int(x, x + h) \tag{1.4}$$

C'est pourquoi, pour approcher une dérivée numériquement on calcule $D(h) = \frac{f(x_0+h)-f(x_0)}{h} + \delta$ où $|\delta| \leq 2\epsilon \left|\frac{f(x)}{h}\right|$.

Ainsi, pour h il existe $\xi \in Int(x, x + h)$ tel que

$$|f'(x_0) - D(h)| \leq 2\epsilon \left|\frac{f(x)}{h}\right| + \left|\frac{h}{2} f''(\xi)\right| \tag{1.5}$$

On remarque alors qu'il existe un h optimal que l'on nomme h^* tel que l'erreur dans le calcul numérique de la dérivée soit minimale. h optimal peut être approché par

$$h^* \leq 2 \left|\epsilon \frac{f(x)}{M_2}\right|^{\frac{1}{2}} \tag{1.6}$$

1.2.2 Exemple

Sur scilab, nous allons approcher la dérivée de la fonction

$$f(x) = \frac{exp(x)}{\cos^3 x + sin^3 x}$$

lorsque $x = \frac{\pi}{4}$ et déterminer l'évolution de l'erreur suivant les valeurs de h pour déterminer h^* .

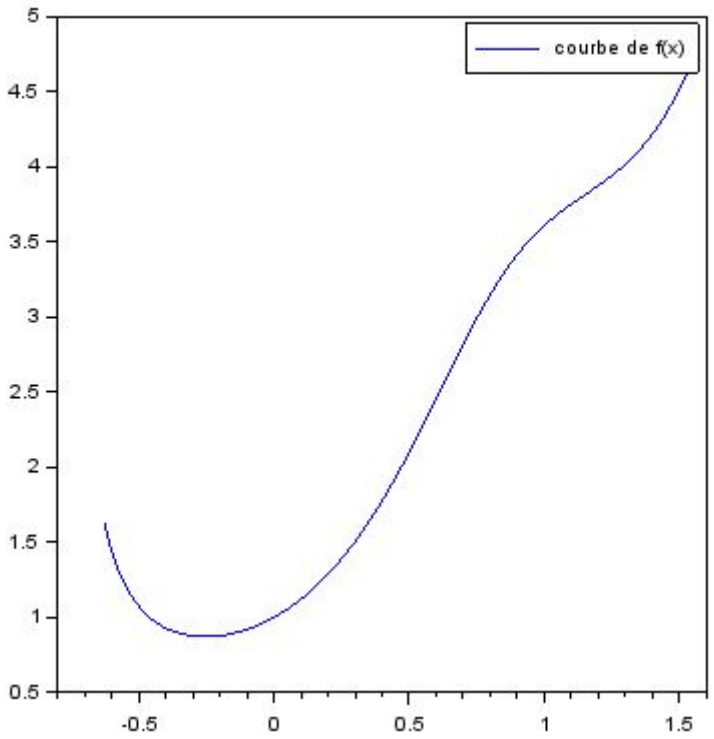


FIGURE 1.2 – courbe de la fonction f(x)

Tout d'abord par le calcul on trouve :

$$f'(x) = \frac{exp(x)(cos(x)^3 + sin(x)^3) + 3sin(x)cos(x)^2 - 3sin(x)^2cos(x)}{(cos(x)^3 + sin(x)^3)^2}$$

Sur scilab, à l'aide du code ci-dessous, on approche à l'odre 1 la dérivée de f(x) lorsque $x = \frac{\pi}{4}$:

```
function y=f(x)
    y=exp(x)./(cos(x)^3 + sin(x)^3);
endfunction

h=2^(0:-1:-70)
D_1 = (f(%pi/4+h)-f(%pi/4))./h;
```

On calcule ensuite l'erreur entre l'approximation numérique et la dérivée réelle calculée précédemment en faisant varier la valeur de h avec le code en annexe . On peut ensuite tracer la courbe de l'évolution de l'erreur en fonction de la valeur de h.

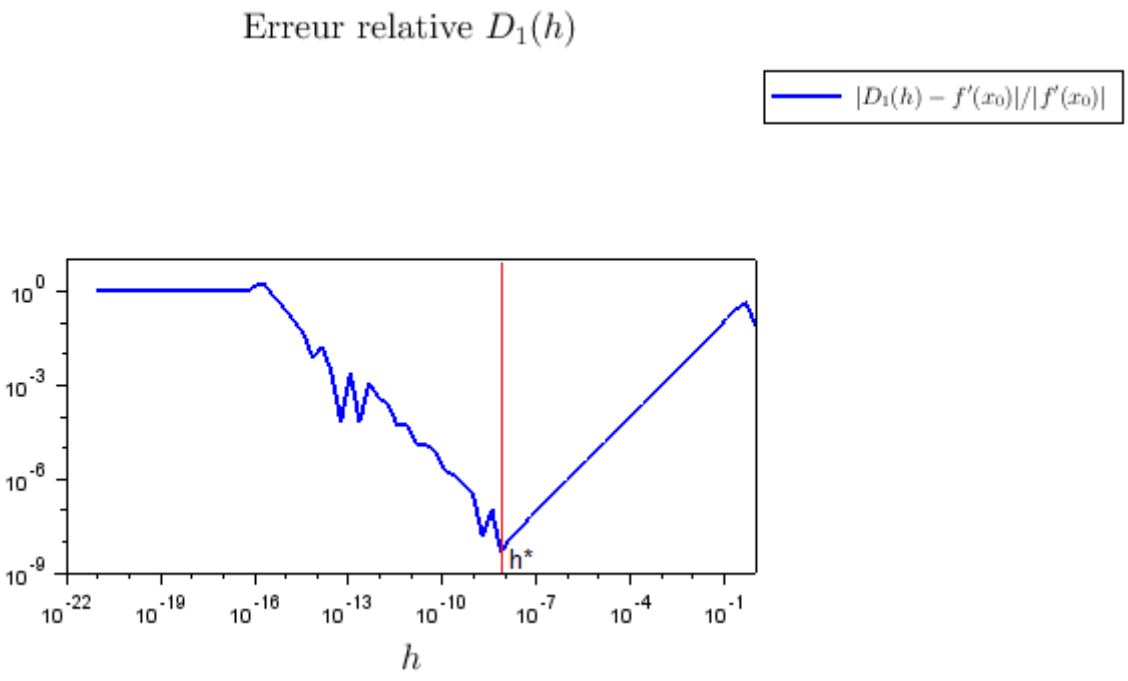


FIGURE 1.3 – évolution de l'erreur en fonction de h

On obtient $h^* = 10^{-8}$ pour une erreur de 10^{-9}

1.2.3 approximation d'ordre 2

On peut aussi approcher une dérivée par la formule d'ordre 2 suivante :

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} + O(h^2) \tag{1.7}$$

Sur scilab, pour approcher la dérivée à l'ordre 2 lorsque $x = \frac{\pi}{4}$ on utilise le code ci-dessous :

```
function y=f(x)
    y=exp(x)./(cos(x)^3 + sin(x)^3);
endfunction

h=2^(0:-1:-70)
D_2 = (f(%pi/4+h)-f(%pi/4-h))./h/2;
```

On calcule ensuite l'erreur entre l'approximation numérique à l'ordre 2 et la dérivée réelle en faisant varier la valeur de h avec le code en annexe. On peut ensuite tracer la courbe de l'évolution de l'erreur en fonction de la valeur de h.

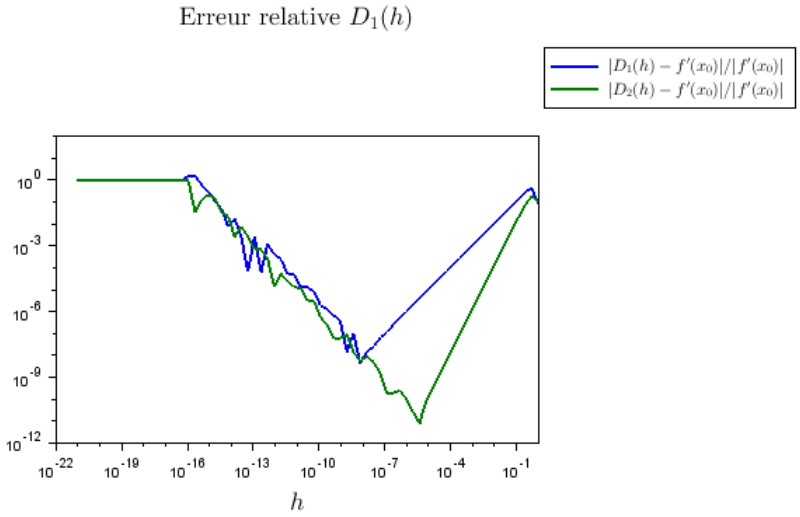


FIGURE 1.4 – évolution de l'erreur en fonction de h

$h^* = 10^{-5}$ pour une erreur de 10^{-13} ce qui est mieux qu'à l'ordre 1

1.2.4 approximation dans le plan complexe

On peut aussi approcher une dérivée à l'aide de la formule d'ordre 2 suivante :

$$f'(x_0) = \frac{Im(f(x_0 + ih))}{h} + O(h^2) \tag{1.8}$$

on constate qu'il n'y a plus de soustractions donc aucun phénomène de catastrophic cancellation. Ainsi, la précision est bien meilleure comme on peut l'observer sur scilab en utilisant le code suivant :

```
function y=f(x)
    y=exp(x)./(cos(x)^3 + sin(x)^3);
endfunction

h=2^(0:-1:-70)
D_2 = (f(%pi/4+h)-f(%pi/4-h))./h/2;
```

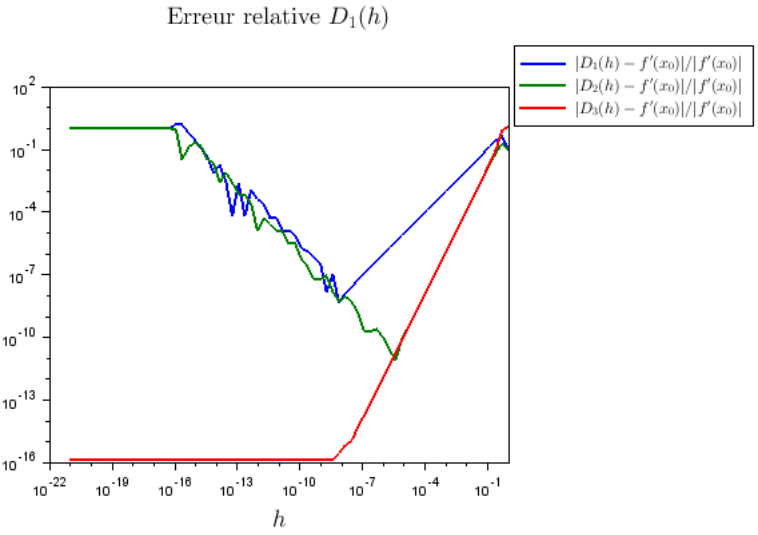


FIGURE 1.5 – évolution de l'erreur en fonction de h

On constate que pour tout h inférieur à 10^{-9} l'erreur correspond au ϵ machine soit 2^{-52} ce qui est optimal et bien meilleur que les approximations précédentes.

Chapitre 2

Problèmes non linéaires I : zéros d'une fonction

Certains problèmes ne sont pas linéaires, c'est à dire que le système étudié ne peut pas s'écrire sous la forme $Ax+b=0$ avec A une matrice inversible.
Ce type de problème est généralement difficile ou long à résoudre analytiquement. Il est alors plus simple d'approcher ses solutions numériquement. C'est notamment le cas pour la détermination des zéros de fonctions non affines ou non usuelles que nous étudierons dans ce chapitre.

2.1 Dichotomie

Lorsque l'on cherche la solution à un problème non linéaire, on peut mettre le problèmes sous la forme $f(x^*) = 0$ ce qui correspond à la recherche des zéros d'une fonction.

En admettant que $f : \mathbb{R} \to \mathbb{R}$ est continue et qu'il existe $I = [a, b]$ tel que $f(a)f(b) < 0$ alors d'après le théorème des valeur intermédiaire $\exists x^* \in [a, b], f(x^*) = 0$.

Pour approcher numériquement x^* on peut construire deux suites $(a_k), (b_k)$ telles que $\forall k, f(a_k)f(b_k) < 0$ et $\lim_{k\to\infty}(b_k - a_k) = 0$

Puisque $a_k < x^* < b_k$, $\lim_{k\to\infty}(a_k) = \lim_{k\to\infty}(b_k) = x^*$

La méthode fonctionnant selon ce principe est appelée dichotomie ou bissection. Concrètement, on définit deux suites $(a_k), (b_k)$ avec $k \geq 0$, $a_0 = a$ et $b_0 = b$.
On pose ensuite $x_k = \frac{a_k + b_k}{2}$ et on choisit une incertitude ϵ selon la précision attendue. On applique alors l'algorithme suivant :

Tant que $|f(x_k)| > \epsilon$
  si $f(x_k)f(a_k) > 0$
    $a_{k+1} = x_k$
  sinon
    $b_{k+1} = x_k$
  fin
fin

Etude de la convergence :

On peut remarquer qu'à chaque nouvelle itération, la taille de l'intervalle est divisé par deux. On a donc :

$$(b_n - a_n) = \frac{1}{2}(b_{n-1} - a_{n-1}) = \left(\frac{1}{2}\right)^n (b_0 - a_0)$$

Et comme

$$|x_n - x^*| \leq \frac{1}{2}(b_n - a_n)$$

alors

$$|x_n - x^*| \leq \left(\frac{1}{2}\right)^{n+1} (b_0 - a_0)$$

et donc on a bien

$$\lim_{n \to \infty} |x_n - x^*| = 0$$

Ainsi, la méthode de la dichotomie converge bien vers le zéro de la fonction étudiée.

Sur scilab, on peut créer une petite fonction qui applique la méthode de la dichotomie :

```
function x=dichotomie(a, b, F, TOL, ITMAX)
for k=1:ITMAX
x = (a + b)/2
if(abs(F(x))<TOL) then
break
end
if(F(x)*F(a) > 0) then
a = x
else
b = x
end
end
endfunction
```

Exemple : On cherche à résoudre l'équation $x^2 = 2$, ce qui revient à trouver les zéros de la fonction $f(x) = x^2 - 2$. Sur scilab on utilise la fonction dichotomie crée précédemment avec la fonction f(x) et l'intervalle [1,2].

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

x=dichotomie(1, 2, f, 1e-10,100)
disp(f(x),x)
```

Scilab nous donne alors une valeur $x^* = 1.4142136$ après 29 itérations, ce qui est très proche de $\sqrt{2}$. Si on remplace l'intervalle [1,2] par [-2,-1], scilab nous donne une valeur $x^* = -1.4142136$, ce qui est très proche de $-\sqrt{2}$.

2.2 méthode du point fixe

Lorsque l'on cherche la solution à un problème non linéaire, on peut aussi mettre le problèmes sous la forme $g(x^*) = x^*$ avec x^* un point fixe de g.

Théorème : Si $g : \mathbb{R} \to \mathbb{R}$ est continûment dérivable et possède un point fixe x^* tel que $|g'(x)| < 1$.
Alors il existe $\alpha > 0$ tel que la suite (x_k) définie par x_0 vérifiant $|x_0 - x^*| < \alpha$ et $x_{k+1} = g(x_k)$, $k \geq 0$ converge vers x^* .

Etude de la convergence :

Puisque $|g'(x)| < 1$, il existe $L \in]0, 1[$ tel que $|g'(x)| < L < 1$. Comme g' est continue, il existe $\alpha > 0$ tel que

$$|x - x^*| < \alpha \Rightarrow |g'(x)| < L$$

Puisque $x_{k+1} = g(x_k)$, la formule des accroissements finis donne

$$x_{k+1} - x^* = g(x_k) - g(x^*) = g'(\xi)(x_k - x^*)$$

avec $xi \in int(x_k, x^*)$.

On peut donc en déduire que $|x_{k+1} - x^*| < L|x_k - x^*|$. Par récurrence on peut alors montrer que

$$|x_{k+1} - x^*| < L^k |x_0 - x^*|$$

et donc

$$\lim_{k \to \infty} |x_k - x^*| = 0$$

Si $|g'(x)| > 1$, la méthode ne converge pas et si $|g'(x)| = 1$, la convergence est incertaine. La méthode fonctionnant selon ce principe est appelée méthode du point fixe ou méthode des approximations successives.

Concrètement, on pose $x_{k+1} = g(x_k)$, on choisit x_0 arbitrairement ainsi qu'une incertitude ϵ suivant la précision recherchée. On réalise ensuite autant d'itérations que nécessaire pour obtenir

$$|x_k - x_{k+1}| < \epsilon$$

Si $|g'(x)| < 1$, la méthode ne converge pas et si $|g'(x)| = 1$, la convergence est incertaine. Sur scilab, on peut créer une petite fonction qui applique la méthode du point fixe.

```
function x=pointfixe(x0, G, TOL,ITMAX)
x(1)=x0
for k=2:ITMAX
x(k) = G(x(k-1))
if(abs(G(x(k))-x(k))<TOL) then
break
end
end
endfunction
```

Exemple :

On cherche à résoudre l'équation $x^2 = 2$, ce qui revient à trouver les points fixes de la fonction $g(x) = \frac{x+2}{x+1}$. Sur scilab on utilise la fonction pointfixe crée précédemment avec la fonction g(x) et $x_0 = 1.5$.

```
function y=g(x)
    y=(x+2)/(x+1)
endfunction
x=pointfixe(1.5,g,1e-10,100)
disp(f(x),x)
```

Scilab nous donne alors une valeur $x^* = 1.4142136$ après 14 itérations, ce qui est très proche de $\sqrt{2}$. Il est par contre impossible de trouver le deuxième point fixe $x_2^* = -\sqrt{2}$ car $g'(x_2^*) > 1$

2.3 méthode de Newton

Si la fonction f(x) est dérivable, l'équation de sa tangente en un point x_k s'écrit $y = f'(x_k)(x - x_k) + f(x_k)$. Ainsi, l'intersection entre l'axe des abscisse et la tangente à f(x) en x_k est solution de l'équation :

$$f'(x_k)(x - x_k) + f(x_k) = 0$$
$$\Leftrightarrow x = x_k - \frac{f(x_k)}{f'(x_k)}$$

Donc le zéro x^* de f(x) correspond au point fixe de la fonction :

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

Il peut donc être obtenu en appliquant la méthode du point fixe à la fonction g(x), à condition que f(x) soit deux fois continûment dérivable. Il s'agit de la méthode de Newton.

Théorème : Si f est dérivable 3 fois et qu'il existe C > 0 tel que $\forall x, \frac{1}{2}|g''(x)| \leq C$ alors pour tout x tel que $|x_0 - x^*| \leq \frac{1}{C}$ alors la méthode de newton converge.

Etude de la convergence :
Si f'(x) est dérivable deux fois et $f'(x^*) \neq 0$ alors

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

$$g'(x^*) = 1 - \frac{f'(x)^2}{f'(x)^2} = 0$$

Si f est dérivable 3 fois et qu'il existe C > 0 tel que $\forall x, \frac{1}{2}|g''(x)| \leq C$, pour tout k, il existe $\xi \in int(x_k, x^*)$ tel que

$$x_{k+1} - x^* = g(x_k) - g(x^*) = g'(x^*)(x_k - x^*) + \frac{1}{2}g''(\xi)(x_k - x^*)^2$$

$$\Leftrightarrow x_{k+1} - x^* = \frac{1}{2}g''(\xi)(x_k - x^*)^2$$

Ainsi, $|x_{k+1} - x^*| < C|x_k - x^*|^2$ et par récurrence on peut montrer que $C|x_k - x^*| < C|x_0 - x^*|^{2^k}$ et donc $x_k \to x^*$ si $C|x_k - x^*| < 1$.

Sur scilab, j'ai créé une fonction appliquant la méthode de Newton :

```
function x=Newton(X0,F,DF,TOL, ITMAX)
x(1)=X0
for k=2:ITMAX
x(k)=x(k-1)-F(x(k-1))/DF(x(k-1));
if(abs(F(x(k)))<TOL) then
break
end
end
x=x(k)
endfunction
```

Exemple : On cherche à résoudre l'équation $x^2 = 2$, ce qui revient à trouver les zéros de la fonction $f(x) = x^2 - 2$ dont la dérivée vaut $df(x) = 2x$.

Sur scilab on utilise la fonction Newton crée précédemment avec les fonction f(x) et df(x) et $x_0 = 1.5$.

```
function y=f(x)
    y=x^2-2
endfunction
function d=df(x)
    d=2*x
endfunction
x=Newton(2,f,df,1e-10,100)
disp(f(x),x)
```

Scilab nous donne alors une valeur $x^* = 1.4142136$, ce qui est très proche de $\sqrt{2}$.

2.4 méthode de la sécante

Lorsque la dérivée de la fonction f(x) étudiée est difficile à calculer pour appliquer la méthode de newton, on peut réutiliser la formule (1.3) vue dans le chapitre précédent en prenant $h = x_k - x_{k-1}$ et $\delta = 0$.

Ce qui donne

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

En appliquant la méthode de Newton avec cette formule, on obtient

$$x_{k+1} = xk - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \tag{2.1}$$

Sur scilab, on peut créer une petite fonction qui applique la méthode de la sécante :

```
function x=secante(X0,X1,F,TOL,ITMAX)
x=zeros(ITMAX,1)
x(1)=X0
x(2)=X1
for k=2:ITMAX-1
    if(abs(F(x(k)))<TOL) then
break
end
    x(k+1)=x(k)-(F(x(k))*(x(k)-x(k-1)))/(F(x(k))-F
(x(k-1)));
end
x=x(k)
endfunction
```

Exemple : On cherche à résoudre l'équation $x^2 = 2$, ce qui revient à trouver les zéros de la fonction $f(x) = x^2 - 2$ dont la dérivée vaut $df(x) = 2x$. Sur scilab on utilise la fonction Newton crée précédemment avec la fonction f(x), $x_0 = 1$ et $x_1 = 2$.

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

x=secante(1,2,f,1e-10,100)
disp(f(x),x)
```

Scilab nous donne alors une valeur $x^* = 1.4142136$, ce qui est très proche de $\sqrt{2}$.

2.5 ordre de convergence

Pour déterminer quelle méthode est la plus efficace pour approcher les zéros d'une fonction, on peut comparer l'ordre de convergence de chaque méthode. L'ordre de convergence p est définit comme suit :

Définition : Si (x_k) converge vers x^* et s'il existe $p > 0$ et $\alpha > 0$ tels que :

$$\lim_{x \to \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} = \alpha \tag{2.2}$$

la convergence est dite d'ordre p.

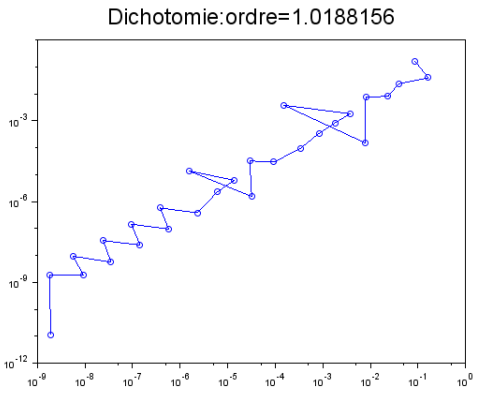
Ainsi, lorsque k est grand

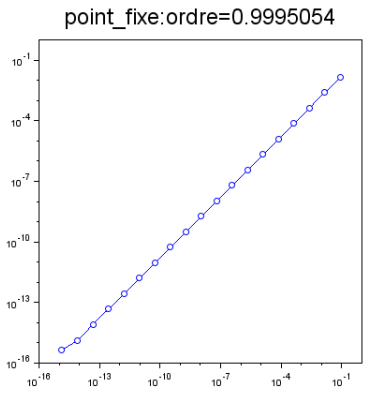
$$|x_{k+1} - x^*| \approx \alpha |x_k - x^*|^p$$

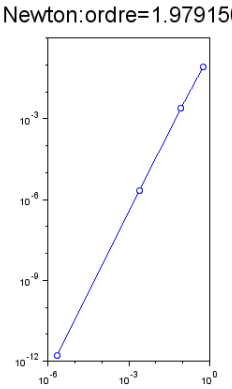
et donc

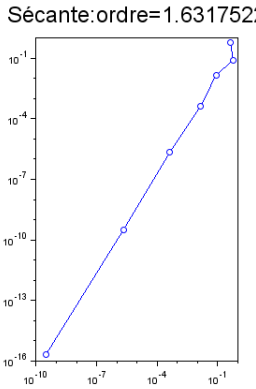
$$\log |x_{k+1} - x^*| \approx p \log |x_k - x^*| + \log \alpha \tag{2.3}$$

L'ordre de convergence p de chaque méthode correspond donc à la pente de la courbe $\log|x_{k+1} - x^*|$ en fonction de $\log|x_k - x^*|$. On peut donc approcher l'ordre de convergence des différentes méthodes sur scilab en étudiant de nouveau l'exemple précédent et en faisant des régressions linéaires avec la macro reglin. Contrairement à précédemment, on devra stocker toutes les valeurs prises par x_k pour pouvoir réaliser la régression linéaire.









Pour la dichotomie et la méthode du point fixe, la convergence est d'ordre 1. Pour la méthode de Newton,la convergence est d'ordre 2 et pour la méthode de la sécante, l'ordre de convergence correspond approximativement au nombre d'or $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$. Ainsi, la méthode de Newton a l'ordre de convergence le plus élevé et est donc la méthode la plus efficace.

2.6 macro fsolve

La macro fsolve de Scilab permet de résoudre des problèmes non-linéaires avec une méthode inspirée de la méthode de Newton mais bien plus efficacement. Il s'agit d'une fonction déjà implémentée dans le logiciel et qui nécessite comme entrées la valeur de x_0 ainsi que la fonction f(x) étudiée, il est aussi possible d'entrer la valeur de f'(x) mais cela est facultatif puisque fsolve peut approcher la dérivée avec une méthode semblable aux méthodes vues dans le chapitre précédent.

2.7 applications

2.7.1 GPS

Le GPS est un système de positionnement fonctionnant grâce à trois satellites situés a des orbites de l'ordre de 28 000 km mesurant leur distance par rapport à un récepteur dont on cherche la position.

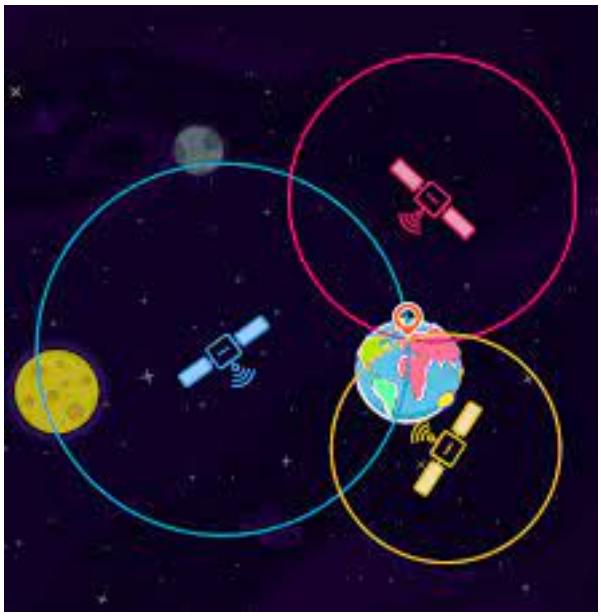


FIGURE 2.1 – image GPS

Dans cette partie on déterminera la position d'un récepteur avec fsolve tel que les positions des trois satellites au moment du calcul de distance sont les suivantes :

$$S_1 = (-11716.227778, -10118.754628, 21741.083973)$$
$$S_2 = (-12082.643974, -20428.242179, 11741.374154)$$
$$S_3 = (14373.286650, -10448.439349, 19596.404858)$$

Et les trois distances respectives du récepteur par rapport aux satellites au moment des calculs sont les suivantes :

$$(d_1,d_2,d_3) = (22163.847742,21492.777482,21492.469326)$$

On note X le vecteur colonne des coordonnées du récepteur. Les trois equations à résoudre sont donc données par :

$$||X - S_i||^2 = d_i^2, i = 1, 2, 3$$

On peut alors définir une fonction $f : \mathbb{R}^3 \to \mathbb{R}^3$ tel que

$$f(x) = \begin{pmatrix} ||X - S_1||^2 = d_1^2 \\ ||X - S_2||^2 = d_2^2 \\ ||X - S_3||^2 = d_3^2 \end{pmatrix} \tag{2.4}$$

Ce qui permet de mettre notre problème sous la forme f(x)=0, c'est à dire la recherche des zéros d'une fonction.

Il suffit ensuite d'utiliser le programme suivant sur scilab pour déterminer la position du récepteur dans un repère centré au centre de la terre.

```
function y=FGPS(X)
    y=[norm(X-S1)^2-d1^2
       norm(X-S2)^2-d2^2
       norm(X-S3)^2-d3^2]
endfunction
S1=[-11716.227778,-10118.754628,21741.083973]';
S2=[-12082.643974,-20428.242179,11741.374154]';
S3=[14373.286650,-10448.439349,19596.404858]';
```

```
d1=22163.847742;
d2=21492.777482;
d3=21492.469326;

[X,v,info]=fsolve([0;0;0],FGPS)
disp(X,v,info)

Y=norm(X)
disp(Y)
```

On obtient alors la sortie suivante :

```
--> exec('C:\Users\merinos\Desktop\latex\gps_fsolve.sce', -1)

   595.02505
  -4856.0251
   4078.33

  -0.0000002
   0.
   0.

   1.

   6369.2864
```

FIGURE 2.2 – Sortie du programme GPS

Ainsi, la position du récepteur correspond au vecteur :

$$V_{recepteur} = \begin{pmatrix} 595.02505 \\ -4856.0251 \\ 4078.33 \end{pmatrix} \tag{2.5}$$

Dont la norme vaut 6369.2864 km ce qui est très proche du rayon terrestre qui vaut 6371 km et prouve donc que le vecteur obtenu est cohérent.

2.7.2 Cinématique inverse

La cinématique inverse est un sujet très intéressant et concret puisqu'il s'agit de l'étude des positions et rotations d'un modèle articulaire afin d'obtenir une position désirée.
Résoudre un problème de cinématique inverse revient à résoudre un système d'équations non-linéaires, ce que nous pourrons faire avec la fonction fsolve.

Dans cette partie, nous étudierons un bras robot articulé dans l'espace d'origine O, avec deux segments dans un même plan contenant l'axe (o_z) et pouvant tourner autour de cet axe.
Nous considérons que le premier segment a une longueur l_1 et fait un angle θ_1 avec le plan (0,x,y), le deuxième segment a une longueur l_2 égale à $\frac{2}{3}$ m et fait un angle $(\theta_1 - \theta_2)$ avec le premier segment et le plan contenant les segments fait un angle $\theta 3$ avec (o,y,z).

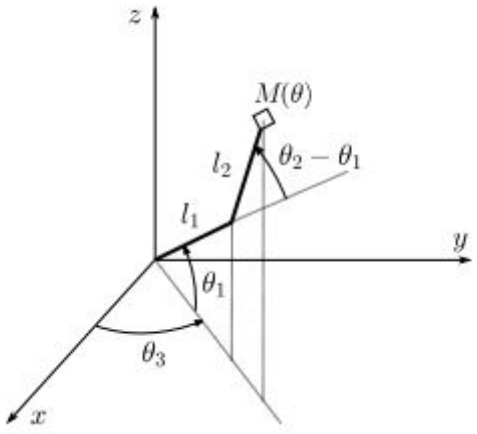


FIGURE 2.3 – bras cinématique inverse

Une étude de dynamique (que nous n'avons pas réalisée) permet de trouver que les coordonnées de l'extrémité du bras sont données par l'équation :

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

Pour commencer, nous allons déterminer les angles $\theta_1, \theta_2, \theta_3$ pour que l'extrémité du segment 2 se trouve au point

$$A = \begin{pmatrix} 1 \\ 0 \\ 0.5 \end{pmatrix} \tag{2.6}$$

Pour trouver les valeurs des angles, il suffit de résoudre le système d'équations $M(\theta) = A$ ce qui est équivalent à trouver les zéros de la fonction $f(\theta) = M(\theta) - A$. Ce qui se fait à l'aide du programme scilab ci-dessous :

```
function y = f(theta,A)
    M = [l*cos(theta(1:2))*cos(theta(3))
        l*cos(theta(1:2))*sin(theta(3))
        l*sin(theta(1:2))];
  y = M-A;
endfunction

l = [1, 2/3];
A = [1.1; 0.1; .5];

theta0=[0.1;0.1;0.1];

[theta,v,info] = fsolve(theta0,list(f,A));
visuArm(theta,l,%t)
disp(theta,v,info)
```

Ce qui nous donne :

$$\theta = \begin{pmatrix} -0.1569933 \\ 1.3946352 \\ 0.0906599 \end{pmatrix} \tag{2.7}$$

Ce qui correspond à la figure suivante :

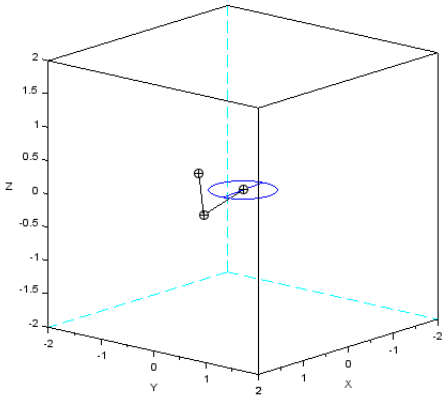


FIGURE 2.4 – extrémité en A

Ensuite, on peut faire tracer des figures au bras. Par exemple j'ai réussi à tracer un parallélépipède grâce à la macro linspace de scilab. Le code que j'ai utilisé est mis en annexe.

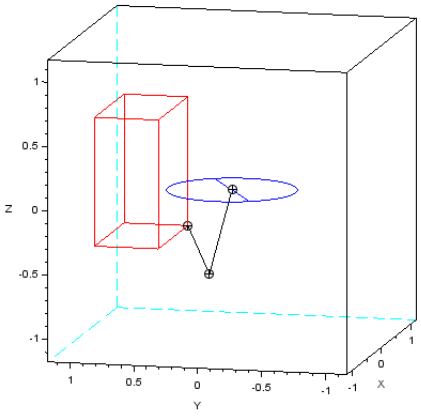


FIGURE 2.5 – parallélépipède

Chapitre 3

Équations différentielles

Résoudre des systèmes d'équations différentielles linéaires peut se faire relativement facilement à la main mais dès lors que les équations différentielles sont très complexes ou non linéaires, il est très intéressant de résoudre ce type de problèmes numériquement. Un exemple très connu d'équation différentielle non linéaire est celui de l'étude d'un pendule simple non amorti et sans frottements :

$$\begin{cases} \ddot{\theta} = -\frac{-g}{L} sin(\theta) \\ \theta(0) = \theta_0 \\ \dot{\theta} = 0 \end{cases}$$

On remarque que ce système d'équations est non linéaire à cause du sinus. Nous reviendrons dessus à la fin de ce chapitre

3.1 principe des méthodes numériques

On cherche a calculer une approximation de $t \to y(t)$ solution de

$$\begin{cases} y'(t) = f(t, y), t \in]0, t] \\ y(0) = y_0 \end{cases}$$

pour des valeurs discrètes de t données par la suite $(t_k)_{k=0...N}$ où $t_k = kh$ et $h = T/N$.

On peut remarquer alors que

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} y'(t)dt = y(t_k) + \int_{t_k}^{t_{k+1}} f(t, y(t))dt$$

La résolution de l'équation différentielle se ramène alors à un calcul numérique d'intégrales.

3.1.1 Schéma explicite d'Euler

La première méthode numérique pour approcher une intégrale et le schéma d'Euler explicite. Il consiste à approcher une intégrale par l'aire du rectangle à gauche c'est à dire le rectangle de hauteur $f(t_k, y(t_k) =$ et de largeur $(t_{k+1} - t_k)$:

$$\int_{t_k}^{t_{k+1}} f(t, y(t))dt \approx (t_{k+1} - t_k)f(t_k, y(t_k), k \geq 0$$

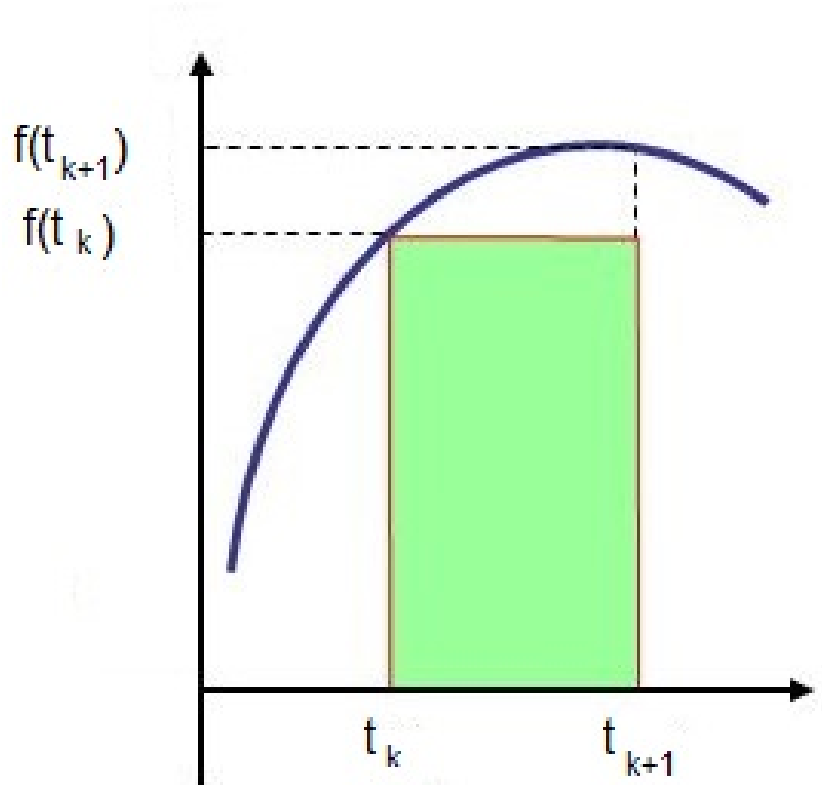


FIGURE 3.1 – approximation par la méthode des rectangles à gauche

Bien entendu, la figure 3.1 est une représentation zoomée de l'approximation d'Euler. Sur un intervalle de temps très faible, l'approximation est très bonne. En posant $h = (t_{k+1} - t_k)$, le schéma explicite d'Euler est donc :

$$y_{k+1} = y_k + hf(t_k, y(t_k)), k \geq 0$$

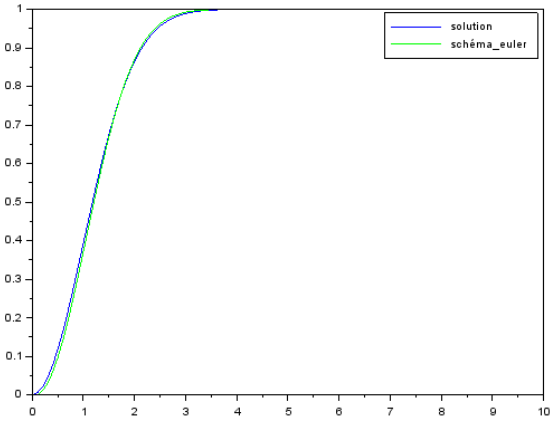
Sur scilab, on peut créer une fonction qui applique le schéma explicite d'Euler :

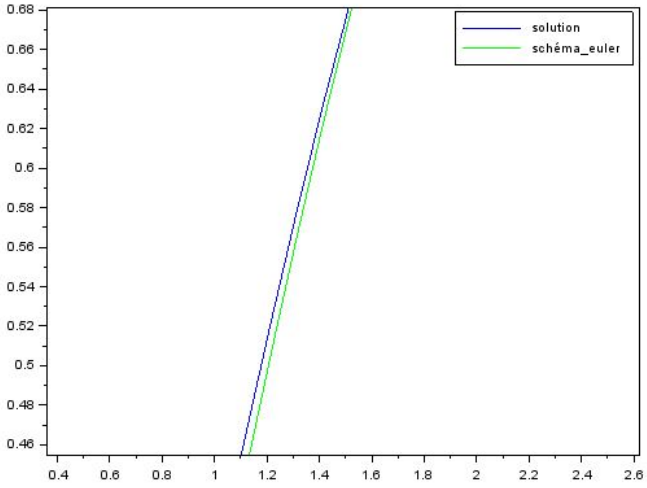
```
function y=euler(y0,t,f)
n=length(t);
h=t(2)-t(1);
y=zeros (1,n);
y(1)=y0;
for k=1:n-1
    y(k+1)=y(k) + h*f(t(k),y(k));
end
endfunction
```

Par exemple si on essaye de déterminer à l'aide de la fonction scilab, la solution de l'équation différentielle :

$$\begin{cases} y' = -ty + t, t \in [0, 4] \\ y(0) = 0; \end{cases}$$

dont la vraie solution est $y = 1 - exp(-t^2/2)$, on obtient :





Ainsi, le schéma explicite d'Euler approche très bien la solution réelle mais si on zoome sur le graphique (image de droite) on constate que le schéma d'Euler a tout de même une erreur.

3.1.2 Schéma implicite d'Euler

Une autre méthode numérique pour approcher une intégrale est le schéma d'Euler implicite. Il consiste à approcher une intégrale par l'aire du rectangle à droite c'est à dire le rectangle de hauteur $f(t_{k+1}, y(t_{k+1}))$ et de largeur $(t_{k+1} - t_k)$:

$$\int_{t_k}^{t_{k+1}} f(t, y(t))dt \approx (t_{k+1} - t_k)f(t_{k+1}, y(t_{k+1})), k \geq 0$$

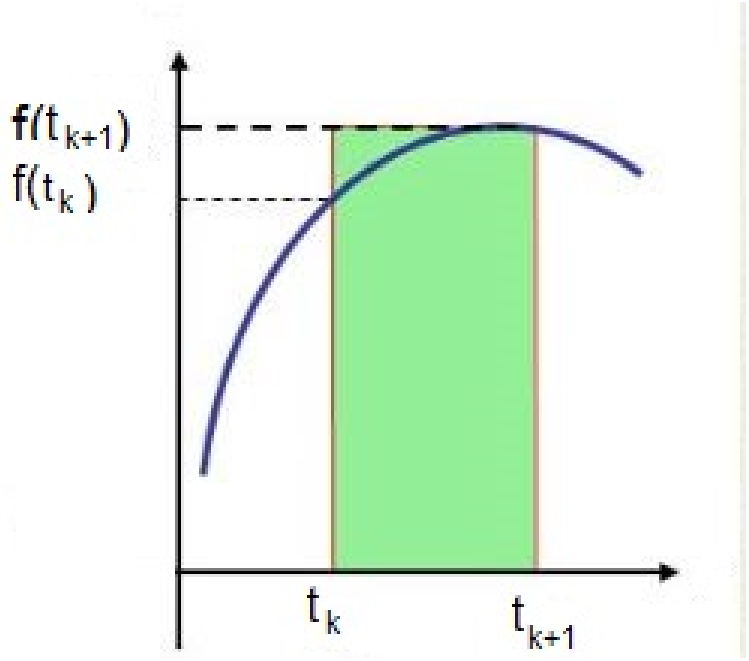


FIGURE 3.2 – approximation par la méthode des rectangles à droite

La figure 3.2 est une représentation zoomée de l'approximation d'Euler, sur un intervalle de temps très faible, l'approximation est très bonne et En posant $h = (t_{k+1} - t_k)$,le schéma implicite d'Euler est donc :

$$y_{k+1} = y_k + hf(t_{k+1}, y(t_{k+1})), k \geq 0$$

Ce schéma n'est pas à un pas et nécessite donc de résoudre pour tout k une équation dont y_{k+1} est la solution. Ce schéma n'est donc pas très intéressant mais permet néanmoins de résoudre certains problèmes raides.

3.1.3 Schéma du point milieu

Une autre méthode numérique pour approcher une intégrale est le schéma du point milieu. Il consiste toujours à approcher une intégrale par l'aire d'un rectangle mais ce rectangle est symétrique par rapport à la droite d'équation $y = t_{k+\frac{1}{2}}$. C'est à dire le rectangle de hauteur $f(t_{k+\frac{1}{2}}, y(t_{k+\frac{1}{2}}))$ et de largeur $(t_{k+1} - t_k)$:

$$\int_{t_k}^{t_{k+1}} f(t, y(t))dt \approx (t_{k+1} - t_k) f(t_{k+\frac{1}{2}}, y(t_{k+\frac{1}{2}})), k \geq 0$$

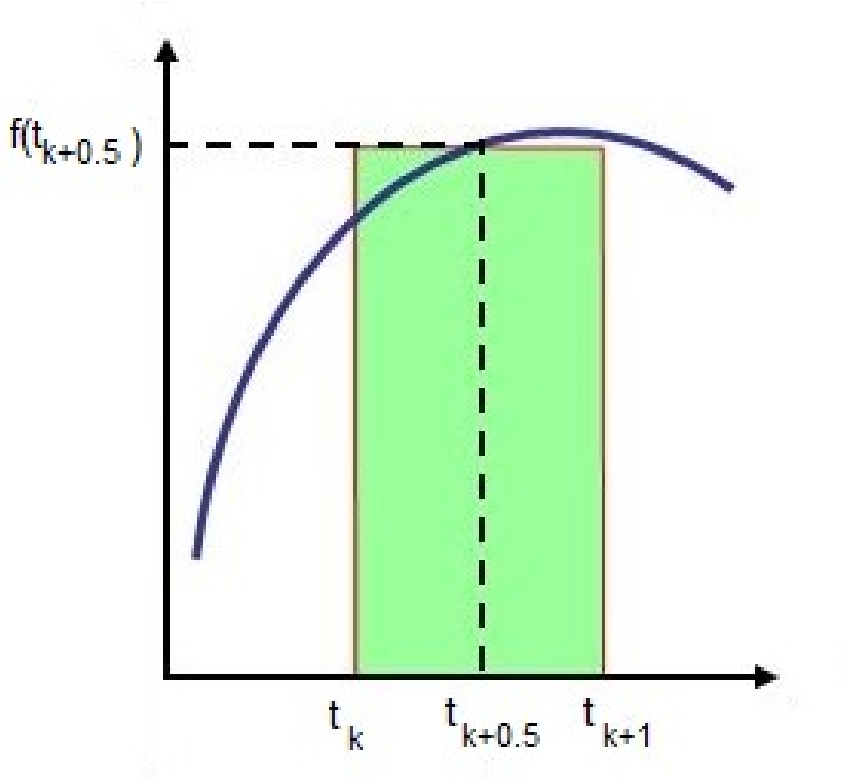


FIGURE 3.3 – approximation par la méthode du point milieu

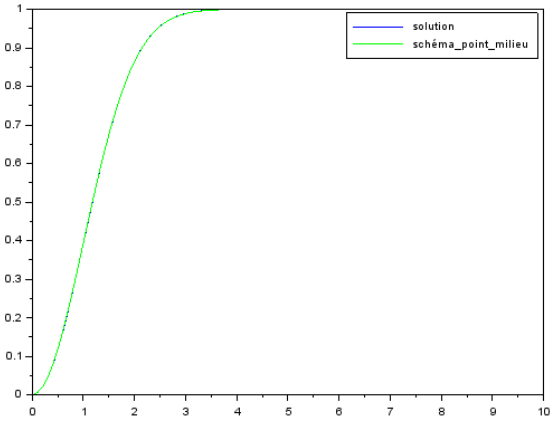
Bien entendu, la figure 3.1 est une représentation zoomée de l'approximation d'Euler. Sur un intervalle de temps très faible, l'approximation est très bonne. Si l'on approche $y(t_{k+\frac{1}{2}})$ par le schéma explicite d'Euler et que l'on pose $h = (t_{k+1} - t_k)$, on obtient le schéma du point milieu :

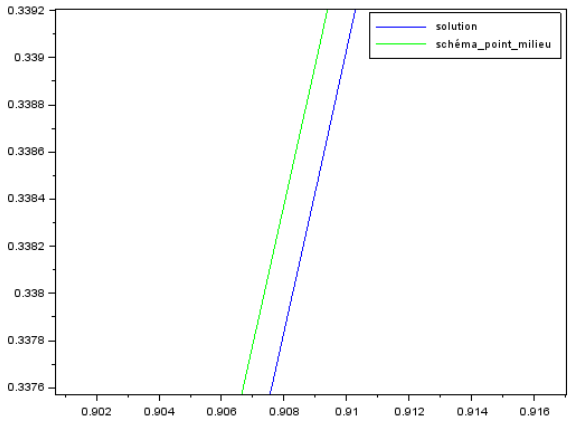
$$y_{k+1} = y_k + hf(t_k + \frac{h}{2}, y_k + \frac{h}{2} f(t_k, y(t_k))), k \geq 0$$

Sur scilab, on peut créer une fonction qui applique le schéma du point milieu :

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

Par exemple si on essaye de déterminer à l'aide de la fonction scilab, la solution de l'équation différentielle $\begin{cases} y' = -ty + t, t \in [0,4] \\ y(0) = 0; \end{cases}$ dont la vraie solution est $y = 1 - exp(-t^2/2)$, on obtient :





Ainsi, le schéma du point milieu approche très bien la solution réelle mais si on zoome sur le graphique (image de droite) on constate que le schéma d'Euler a tout de même une erreur.

3.1.4 Schéma d'Euler-Cauchy

Il existe une autre méthode numérique consistant à approcher une intégrale par l'aire d'un trapèze ayant pour première base $b_1 = f(t_k, y(t_k))$, pour seconde base $b_2 = f(t_{k+1}, y(t_{k+1}))$ et de hauteur $h = (t_{k+1} - t_k)$. En utilisant la formule de calcul d'aire d'un trapèze $\frac{h(b_1+b_2)}{2}$ on obtient :

$$\int_{t_k}^{t_{k+1}} f(t, y(t))dt \approx \frac{h}{2}(f(t_k, y(t_k)) + f(t_{k+1}, y(t_{k+1})), k \geq 0$$

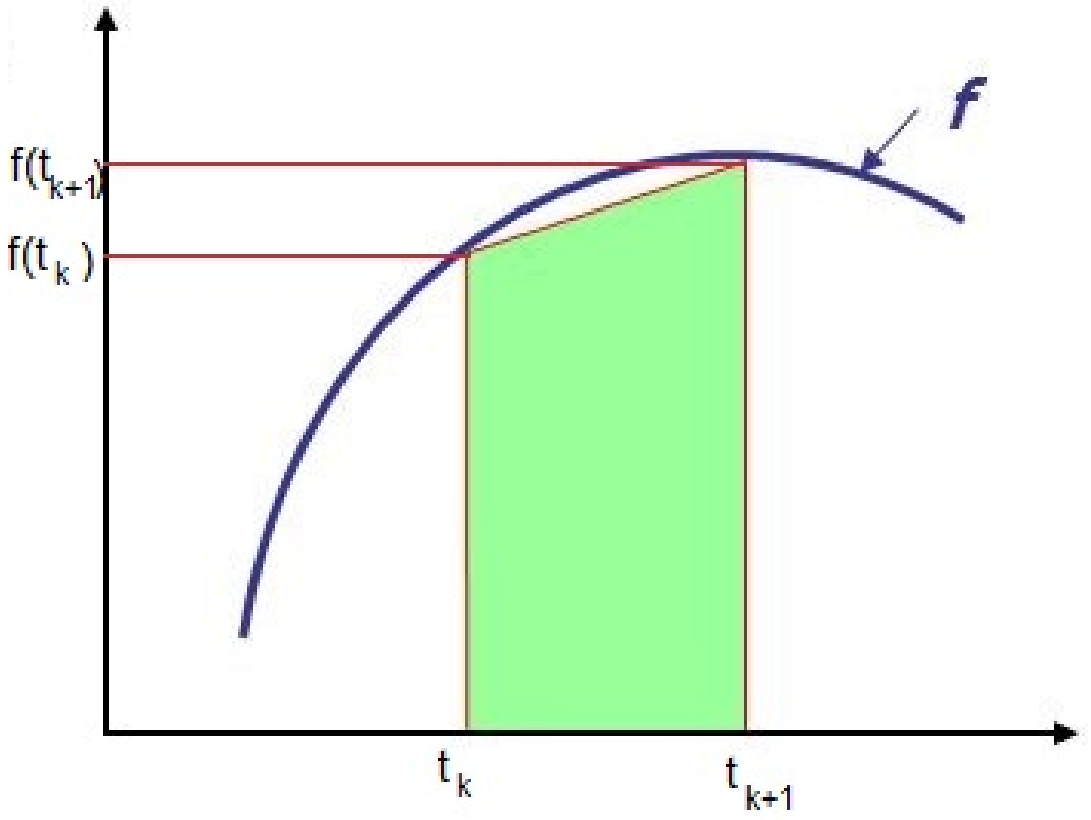


FIGURE 3.4 – approximation par la méthode des trapèzes

La figure 3.2 est une représentation zoomée de l'approximation d'Euler-Cauchy, sur un intervalle de temps très faible, l'approximation est très bonne. Si l'on approche $y(t_{k+1})$ par le schéma explicite d'Euler, on obtient le schéma d'Euler-Cauchy :

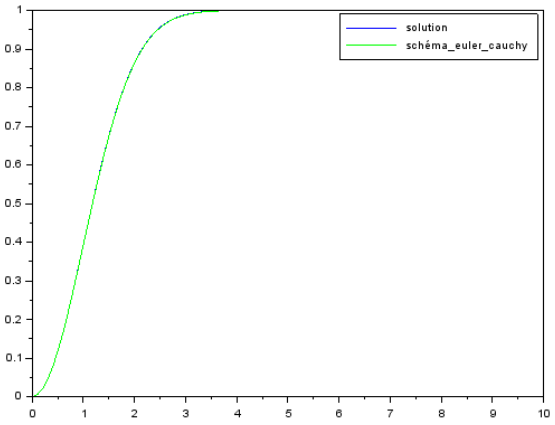
$$y_{k+1} = y_k + \frac{h}{2} f(t_{k+1}, y_k + hf(t_k, y(t_k))), k \geq 0$$

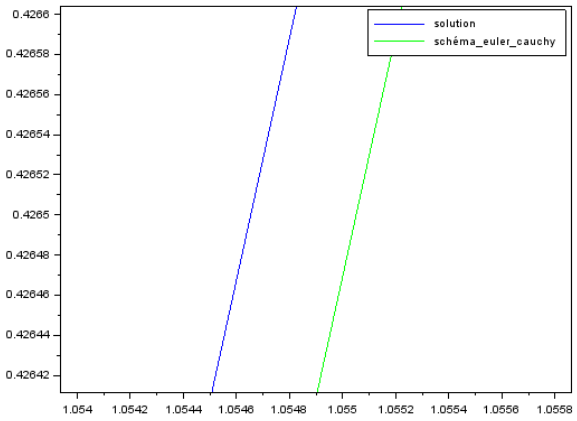
Sur scilab, on peut créer une fonction qui applique le schéma d'Euler-Cauchy :

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

Si on reprend l'exemple précédent : $\begin{cases} y' = -ty + t, t \in [0,4] \\ y(0) = 0; \end{cases}$

Et si on essaye à présent de déterminer à l'aide de la fonction scilab euler-cauchy, la solution de l'équation différentielle dont la vraie solution est $y = 1 - exp(-t^2/2)$, on obtient :





Ainsi, le schéma d'Euler approche très bien la solution réelle mais si on zoome sur le graphique (image de droite) on constate que le schéma d'Euler a tout de même une erreur.

3.1.5 Schéma de Runge et Kutta

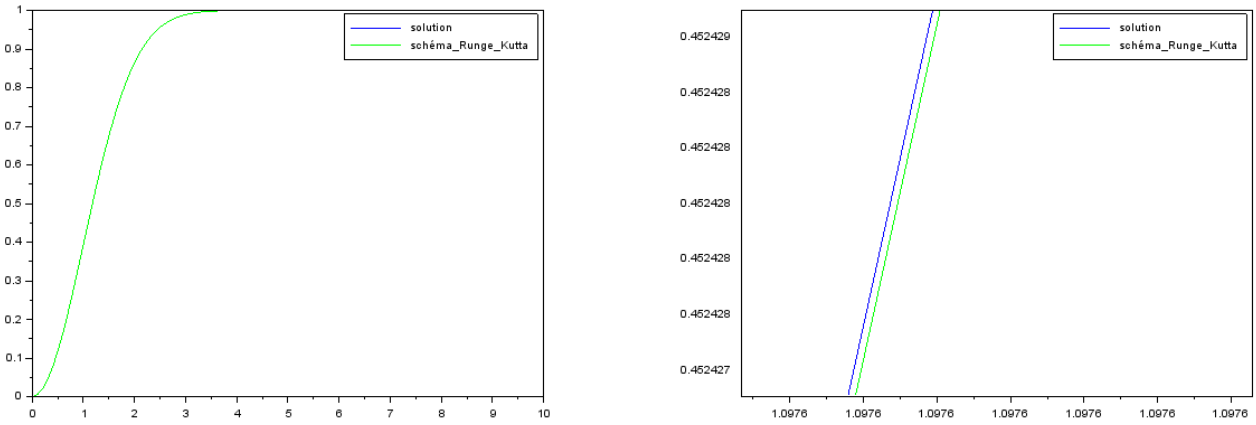
Ce schéma est bien plus complexe que les précedent et s'exprime de la manière suivante :

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\ k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\ k_4 &= f(t_n + h, y_n + hk_3) \\ y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

Sur scilab, on peut créer une fonction qui applique le schéma de Runge et Kutta :

```
function y=Runge_Kutta(y0,t,f)
n=length(t);
h=t(2)-t(1);
y=zeros (1,n);
y(1)=y0;
for k=1:n-1
    k1=f(t(k),y(k));
    k2=f(t(k)+h/2,y(k)+h/2*k1);
    k3=f(t(k)+h/2,y(k)+h/2*k2);
    k4=f(t(k)+h,y(k)+h*k3);
    y(k+1)=y(k) + h/6*(k1+2*k2+2*k3+k4);
end
endfunction
```

Par exemple si on essaye de déterminer à l'aide de la fonction scilab, la solution de l'équation différentielle $\begin{cases} y' = -ty + t, t \in [0, 4] \\ y(0) = 0; \end{cases}$ dont la vraie solution est $y = 1 - exp(-t^2/2)$, on obtient :



Ainsi, le schéma de Runge et Kutta approche très bien la solution réelle mais si on zoome sur le graphique (image de droite) on constate que le schéma de Runge et Kutta a tout de même une erreur qui est néanmoins bien plus faible que celle des schémas précédents.

3.1.6 Macro ode

Dans les logiciels de calcul numérique, les macros déjà intégrées utilisent un pas variable pour pouvoir assurer que l'erreur locale reste inférieure à une tolérance donnée. Dans ce cas, des pas intermédiaires sont réalisés entre les instants différents instants $(t_k)_{k=0...N}$ imposés par l'utilisateur de manière à respecter la tolérance. C'est notamment le cas de la macro ODE de scilab.

3.2 propriétés

schéma à un pas : On appelle schéma à un pas la construction de la suite récurrente (y_k) définie par $y_{k+1} = y_k + h\phi(t_k, y_k, h), k \geq 0$

Les schémas d'Euler explicite, d'Euler Cauchy, de Runge et Kutta et du point milieu sont donc des schéma à un pas alors que le schéma d'Euler implicite n'est pas à un pas.

Convergence : On dit qu'un schéma à un pas est convergent si $\lim_{h\to 0} \max_{\{k=0...N\}} |y(t_k) - y_k| = 0$

La convergence est résulte de deux propriétés complémentaires :

Consistance : Soit $e_k = y(t_{k+1}) - y(t_k) - h\phi(t_k, y(t_k), h)$
On dit qu'un schéma est consistant si $\lim_{h\to 0} \Sigma_{k=0}^{N-1} |e_k| = 0$

S'il existe C_1 tel que $|e_k| < C_1 h^{p+1}$, on dit que le schéma est d'ordre p.

Le schéma d'Euler est d'ordre 1, le schéma d'Euler cauchy et celui du point milieu sont d'ordre 2, le schéma de Runge et Kutta est d'ordre 4.

Stabilité : Soient les suites (y_k) et $(\tilde{y}_k)$ définies par $y_0, \tilde{y}_0$, et pour $k \geq 0$ par :

$$y_{k+1} = y_k + \phi(t_k, yk, h)$$

$$\tilde{y}_{k+1} = \tilde{y}_k + \phi(t_k, \tilde{y}_k, h) + \epsilon_k$$

On dit qu'un schéma est stable s'il existe S>0 tel que :

$$\max_{\{k=0...N\}} |\tilde{y}_k - y_k| \leq S \left(|\tilde{y}_0 - y_0| + \sum_{k=0}^{N-1} |\epsilon_k| \right)$$

Le calcul de la constante S fait intervenir la formule du schéma, le temps final T et L la constante de Lipschitz de f. Si f est Lipschitzienne les

schéma explicite d'Euler, d'Euler Cauchy, du point milieu et de Runge et Kutta sont tous stables.

Théorème : Si le schéma est stable, consistant et d'ordre p, alors il est convergent et il existe C_2 tel que

$$\max_{\{k=0...N\}} \left| y_k - y_{(t_k)} \right| \leq C_2 h^p$$

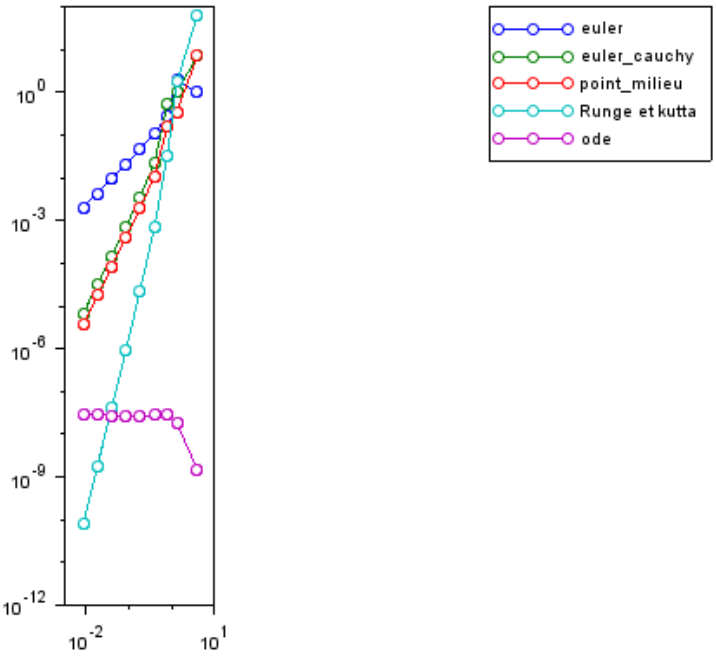
- La constante C_2 dépend du schéma, des propriétés de la solution y (donc aussi de f), ainsi que de T. Le choix d'un schéma adapté au problème étudié nécessite donc des informations sur la solution.

-L'étude de la stabilité d'un schéma nous montre l'influence de l'accumulation des erreurs d'arrondi : pour atteindre une précision voulue avec un schéma d'ordre insuffisant, augmenter le nombre de pas N ne fera qu'augmenter les erreurs d'arrondi et ne permettra donc pas forcément de gagner en précision contrairement à l'utilisation d'un schéma d'ordre plus élevé. C'est cette notion d'ordre qui permet de déterminer l'efficacité d'un modèle : plus l'ordre sera élevé, plus le modèle convergera rapidement.

Nous allons à présent essayer de comparer les différents modèles présentés précédemment en reprenant l'exemple de la résolution de l'équation différentielle $\begin{cases} y' = -ty + t, t \in [0, 4] \\ y(0) = 0; \end{cases}$
. Pour comparer les modèles, on va tracer l'évolution de l'erreur en fonction de la valeur de h pour chacun des modèles à l'aide du code ci-dessous :

```
err= zeros(5,length(N));
for i=1:length(N)
   h=T/N(i);
   t=0:h:T;
   err(1,i)=max(abs(sol(t)-euler(0,t,f1)));
   err(2,i)=max(abs(sol(t)-euler_cauchy(0,t,f1)));
   err(3,i)=max(abs(sol(t)-point_milieu(0,t,f1)));
   err(4,i)=max(abs(sol(t)-Runge_Kutta(0,t,f1)));
   err(5,i)=max(abs(sol(t)-ode(0,0,t,f1)));
   end
h=T./N;
plot(h,err,"-o");
gca().log_flags="ll";
legend("euler","euler_cauchy","point_milieu","
Runge␣et␣kutta","ode",-1);
isoview on
```

on obtient alors la figure ci-dessous :



Pour déterminer l'ordre des schémas, il suffit alors d'utiliser la fonction reglin :

```
reglin(log(h),log(err))
```

On obtient alors :

ordre(**Euler**)= 1.1425658
ordre(**Euler-Cauchy**)= 2.364994
ordre(**Point milieu**)= 2.3591619
ordre(**Runge et Kutta**)= 4.5592891
ordre(**ODE**)= 0.2953728

Ainsi, le schéma d'Euler explicite est bien d'ordre 1, les schéma d'Euler-Cauchy et du point milieu sont bien d'ordre 2 et le schéma de Runge et Kutta est bien d'ordre 4.

Par contre, pour la fonction ode, l'ordre tend vers 0, cela s'explique par le fait que l'erreur de cette fonction reste pratiquement constante peu importe la valeur de h ce qui rend la fonction très robuste.

On constate aussi que l'erreur générée par le modèle du point milieu est un peu plus faible que celle du modèle d'Euler-Cauchy bien que leur ordre soit identique.

3.3 Applications

Pendule

La première application de la résolution numérique des équations différentielles que nous allons réaliser est l'étude du déplacement d'un pendule.

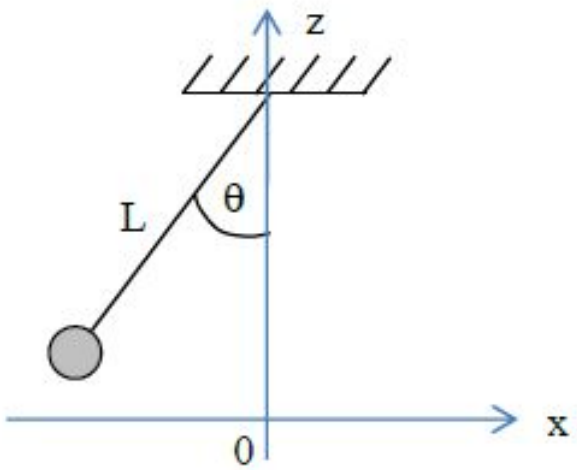


FIGURE 3.5 – schéma d'un pendule simple

En PS21, on apprend que l'étude du mouvement d'un pendule correspond à la résolution du système d'équations différentielles suivant :

$$\begin{cases} \ddot{\theta}(t) = -\frac{g}{L}\sin(\theta(t)) \\ \theta(0) = \theta_0 \\ \dot{\theta(0)} = 0 \end{cases}$$

Il s'agit d'une équation différentielle non linéaire qui est donc très complexe à résoudre à la main. Pour la résoudre on se contentait de considérer la situation où $\theta(t)$ est faible et donc pratiquement égale à $\sin(\theta(t))$ d'où le résultat $\tilde{\theta} = \theta_0 \cos(\omega t), \omega = \sqrt{\frac{g}{L}}$ avec $\tilde{\theta}$ la solution du système en considérant $\tilde{\theta} = \sin(\tilde{\theta})$

Grâce à ce que l'on a vu dans ce chapitre, on peut approcher la véritable solution grâce à la fonction ode par exemple pour ensuite tester la validité de $\tilde{\theta}$.

Avant de passer à la programmation, il faut d'abord transformer cette équation différentielle du second ordre en un système d'équations différentielles d'ordre 1.

Pour y parvenir, on pose

$$y = \begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix}$$

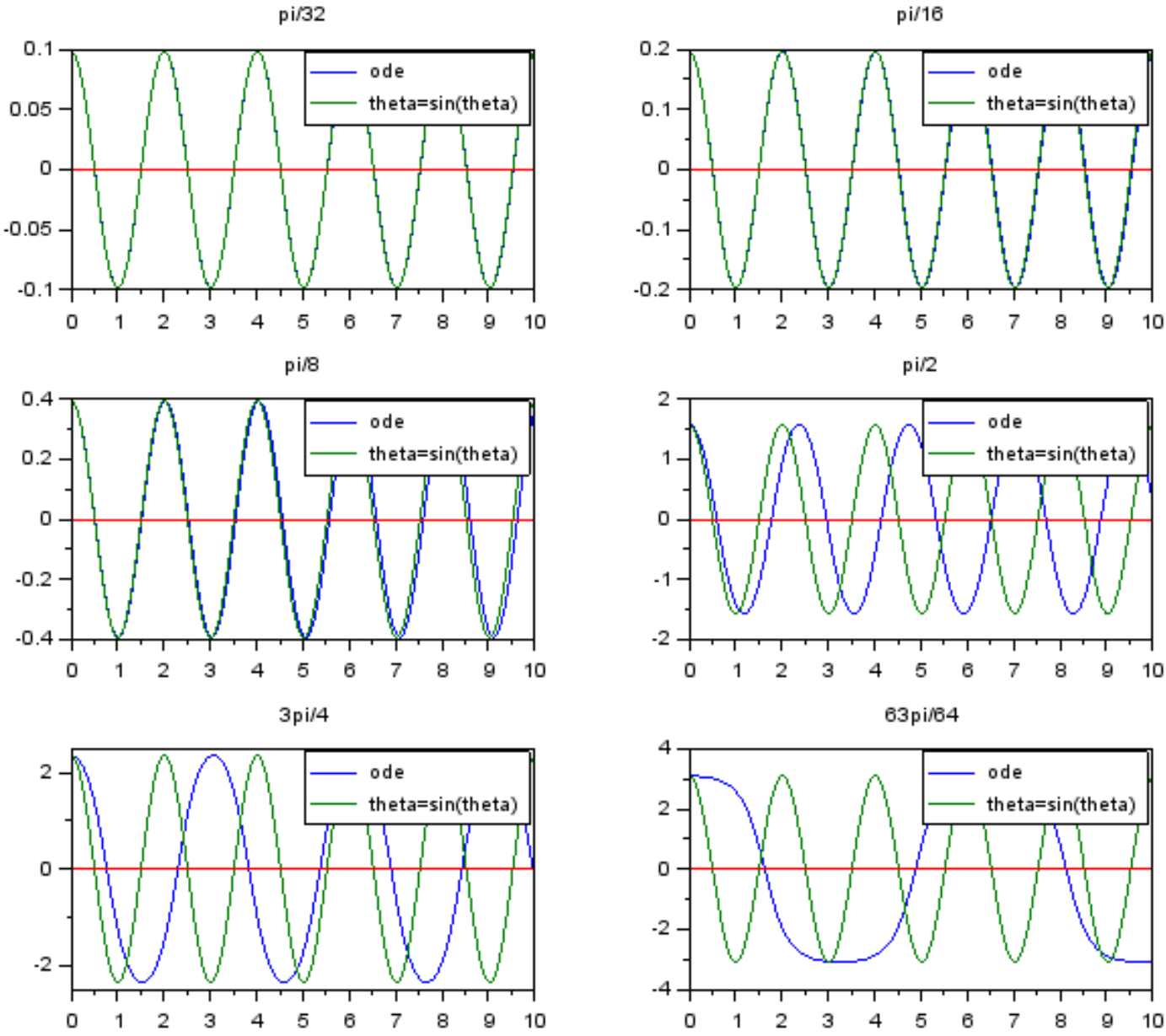
Ce qui implique : $\dot{y} = \begin{pmatrix} y_2 \\ -\frac{g}{L}\sin(y_1) \end{pmatrix}$ et $y_0 = \begin{pmatrix} \theta_0 \\ 0 \end{pmatrix}$.

Il s'agit à présent un système d'équations différentielles d'ordre 1 que l'on peut résoudre sur scilab.

Pour résoudre ce problème on utilise le programme ci-dessous :

```
function dydt=f(t,y)
theta=y(1);
thetaprime=y(2);
g=9.81;
L=1;
dydt=[thetaprime ; -g/L*sin(theta)]'
endfunction
g=9.81;
L=1;
t=linspace(0,10,1000);
y0=[%pi/32;0];
y=ode(y0,0,t,f);
theta=y(1,:);
plot(t,theta,t,y0*cos(sqrt(g/L)*t));
```

En prenant différentes valeurs pour θ_0 on obtient la figure ci-dessous pour le déplacement du pendule :



On remarque que plus θ_0 est grand, plus l'erreur faite par l'approximation faites en PS21 devient grande. Ainsi, il est bien plus intéressant d'utiliser la fonction ode pour résoudre ce problème.

Equation de Duffing

L'équation de Duffing est la suivante :

$$x'' + 0.15 * x' - x + x^3 = 0$$

. Comme pour le pendule, cela peut se réécrire sous forme matricielle en posant $X = \begin{pmatrix} x \\ x' \end{pmatrix}$, et on obtient

$$X' = \begin{pmatrix} X_2 \\ X_1 - X_1^3 - kX_2 \end{pmatrix} = f(X)$$

Nous allons étudier les points d'équilibre de ce système. Il convient de rappeler la définition d'un point d'équilibre.

Point d'équilibre :
Le vecteur X^* est un point d'équilibre du système (S) si $X(0) = X^* \Rightarrow \forall t > 0, X(t) = X^*$

Le points d'équilibre de (S) sont caractérisés par $f(X^*) = 0$

Les points d'équilibre sont donc définis par $f(X) = 0$
Soit $X_2 = 0$ et $X_1 - X_1^3 = 0$

On a donc trois points d'équilibres possibles :

$$X^* = \begin{pmatrix} -1 \\ 0 \end{pmatrix} \text{ ou } X^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \text{ ou } X^* = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

On souhaite étudier la stabilité de ces points d'équilibre. Pour y parvenir il convient de définir la stabilité d'un point d'équilibre.

Stabilité :
Soit A une matrice $n \times n$ et le système d'équations différentielles linéaires

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

Si on note $(\lambda_k)_{k=1...n}$ les valeurs propres de A, alors
- $\exists k \in \{1, ..., n\}, Re(\lambda_k) > 0 \Rightarrow$ point fixe instable
- $\forall k \in \{1, ..., n\}, Re(\lambda_k) \leq 0 \Rightarrow$ point fixe stable
- $\forall k \in \{1, ..., n\}, Re(\lambda_k) \leq 0 \Rightarrow$ point fixe stable
- $\forall k \in \{1, ..., n\}, Re(\lambda_k) < 0 \Rightarrow$ point fixe asymptotiquement stable

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

- Si $X^* = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$ ou $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ on a $f'(X) = \begin{pmatrix} 0 & 1 \\ -2 & -k \end{pmatrix}$ qui a pour polynôme caractéristique $\lambda^2 + k\lambda + 2$.
  - si $k^2 < 8$, ses racines sont complexes conjuguées avec une partie réelle strictement négative $-\frac{k}{2}$
  - si $k^2 \geq 8$, les deux racines sont réelles strictement négatives.

  Dans les deux cas, X^* est **asymptotiquement stable**.

- Si $X^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ on a $f'(X) = \begin{pmatrix} 0 & 1 \\ 1 & -k \end{pmatrix}$ qui a pour polynôme caractéristique $\lambda^2 + k\lambda - 1$. Le produit des racines -1 est négatif et les racines sont donc réelles et de signe opposé, une des deux racines est donc strictement positive. Donc X^* **est instable**.

Sur scilab on réalise l'étude de la stabilité des points d'équilibre en traçant les portraits de phase à l'aide du programme ci-dessous :

```
function DXdt=duffing(t,X)
    k=0.1;
    DXdt=[X(2);-k*X(2)+X(1)-X(1)^3]
endfunction

X1=linspace(-0.1,0.1,10);
X2=linspace(-0.1,0.1,10);

fchamp(duffing,0,X1,X2);

t=linspace(0,5,200)
plot(0,0,"xr")

for X10=X1
    for X20=X2
        X=ode([X10;X20],0,t,duffing)
        plot(X(1,:),X(2,:))
        end
end

gca().data_bounds=[X1(1) X1($) X2(1) X2($)]
```

Pour le point fixe $X^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ on obtient le portrait de phase suivant sur lequel les trajectoires sont repoussées par le point d'équilibre qui est donc instable.

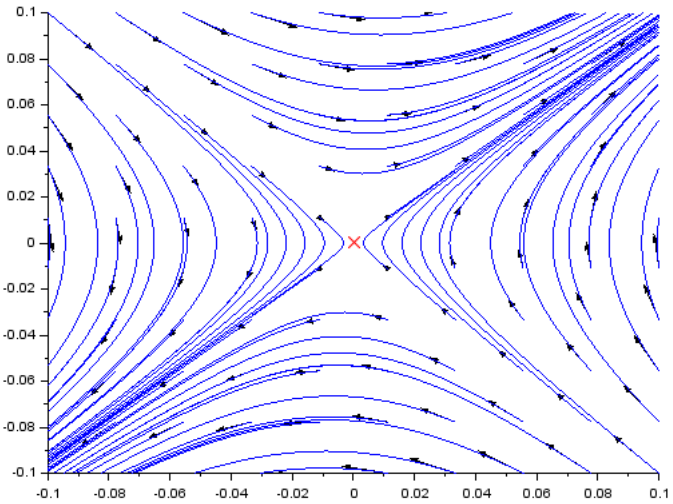


FIGURE 3.6 – portrait de phase de $X^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

Pour les points fixes $X^* = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$ et $X^* = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ on obtient les portraits de phase suivant pour lesquels les trajectoires sont attirées par les points d'équilibres qui sont donc stables :

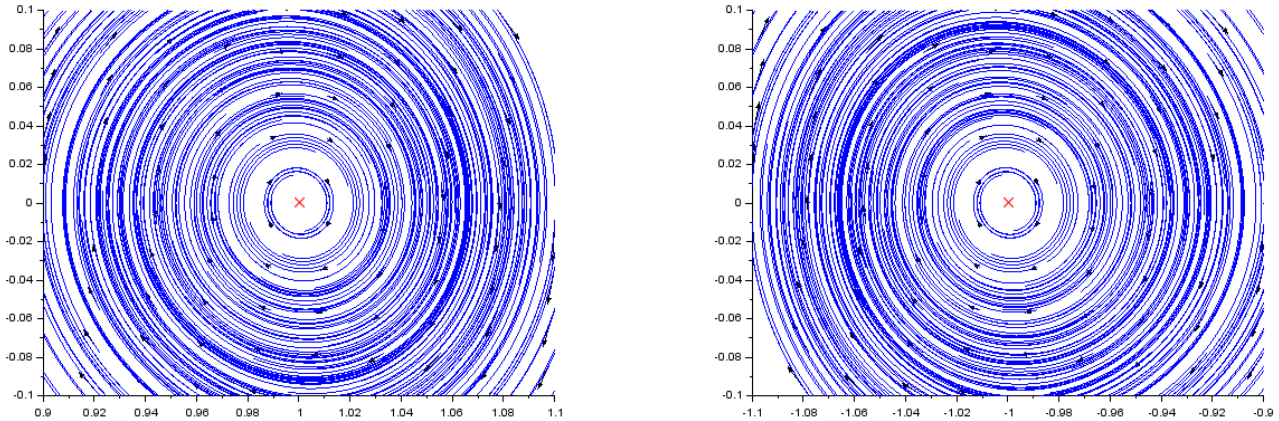


FIGURE 3.7 – portraits de phase de $X^* = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$ et $X^* = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

3.4 intégration

En cours, nous n'avons pas vu de méthode numérique pour calculer des intégrales mais en réutilisant le schéma d'Euler, on peut facilement calculer des intégrales en réalisant des sommes.

En effet, l'intégrale d'une fonction sur un intervalle [a,b] correspond à l'aire sous cette courbe. Ainsi, si on subdivise l'intervalle [a,b] en n sous intervalles qui ont donc pour taille $\frac{b-a}{n}$, que l'on réalise un schéma d'Euler sur chaque sous-intervalle et que l'on somme tous ces schémas d'Euler, on approche l'aire sous la courbe et donc l'intégrale. C'est exactement la même méthodologie que pour les sommes de Riemann. En réalisant suffisamment de subdivisions, on pourra approcher l'intégrale avec une faible erreur.

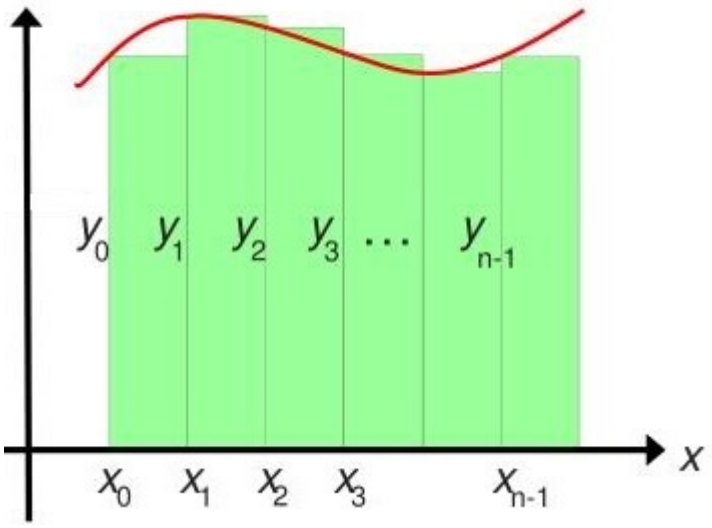


FIGURE 3.8 – schéma de l'approche d'un intégrale par des schémas d'Euler

Cela donne :

$$\int_a^b f(x)dx \approx \frac{b-a}{n} \sum_{k=1}^{n} f\left(a + k\frac{b-a}{n}\right)$$

On peut alors écrire le programme scilab ci dessous qui calcule l'intégrale de la fonction $f(x) = x^2$ sur l'intervalle $[0, 10]$ et détermine l'erreur d'approximation :

```
function y=f(x)
    y=x^2
endfunction
clf
t=linspace(0,10,100),
plot(t,f(t))
n=10;
a=0;
b=10;
h=(b-a)/n;
z=0;

for i=1:n
z=z+h*f(a+i*h)
end

integ=b^3/3-a^3/3;
erreur=norm(integ-z)
disp (integ);
disp(z);
disp(erreur);
```

Pour plus de lisibilité, j'ai décidé d'ajouter dans mon programme une fonctionnalité permettant d'afficher les rectangles utilisés dans l'approximation de l'intégrale. Il s'agit d'une commande intégrée dans la boucle calculant la somme des schémas d'Euler pour pouvoir ainsi tracer à chaque itération le rectangle dont l'aire est mesurée.

```
for i=1:n
z=z+h*f(a+i*h)
if i==n then
    break
    else
for j=1:10
v(j)=a+i*h
s(j)=f(a+i*h)
end
u=linspace(0,f(a+i*h),10)
w=linspace(a+i*h,a+i*h+h,10)
plot(v,u)
plot(w,s)
end
```

Ce que j'ai obtenu pour $n = 5; n = 10; n = 20; n = 100$ est présenté ci-dessous :

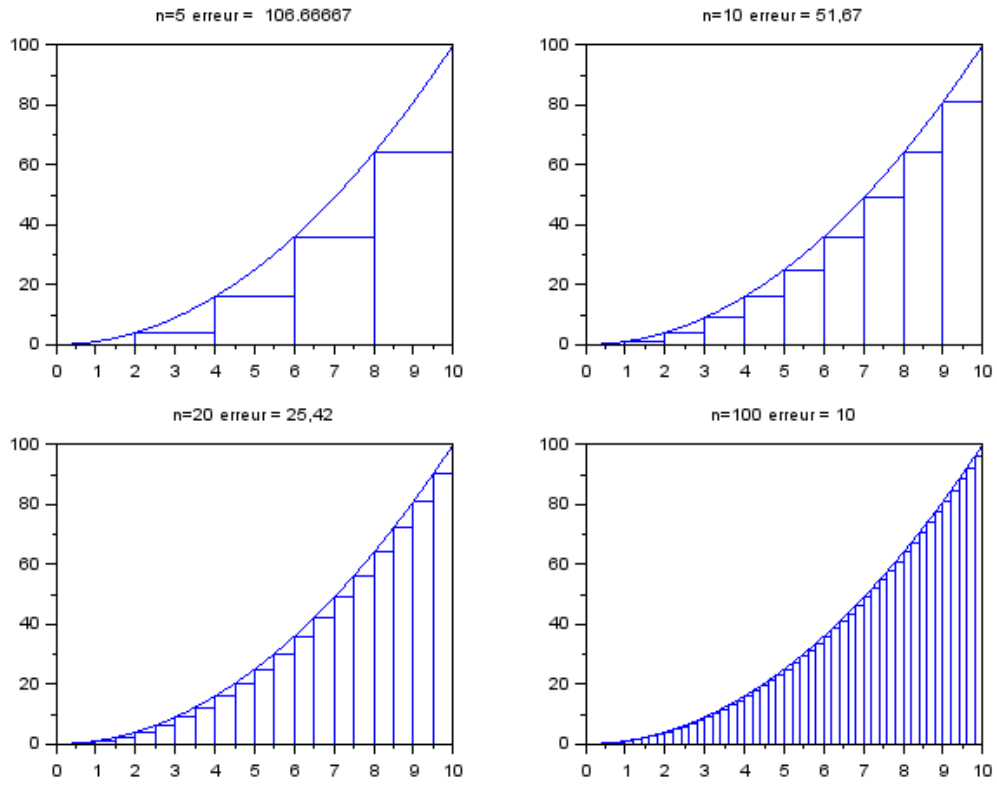


FIGURE 3.9 – Approche de l'intégrale de $f(x) = x^2$ par des schémas d'Euler

J'ai aussi testé mon programme avec la fonction logarithmique sur l'intervalle [1,10] pour laquelle les erreurs sont très faibles.

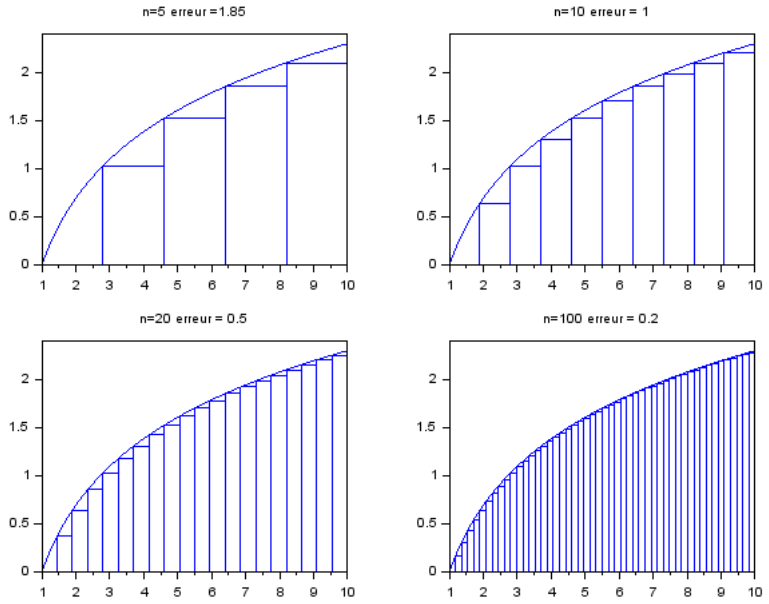


FIGURE 3.10 – Approche de l'intégrale de $f(x) = \log(x)$ par des schémas d'Euler

J'ai enfin testé mon programme avec la fonction exponentielle sur l'intervalle [0,10], les erreurs sont très élevées mais cela est liée aux valeur prise par la fonction exponentielle qui sont colossales et donc par rapport aux valeurs prises par la fonction exponentielle, les erreurs sont convenables.

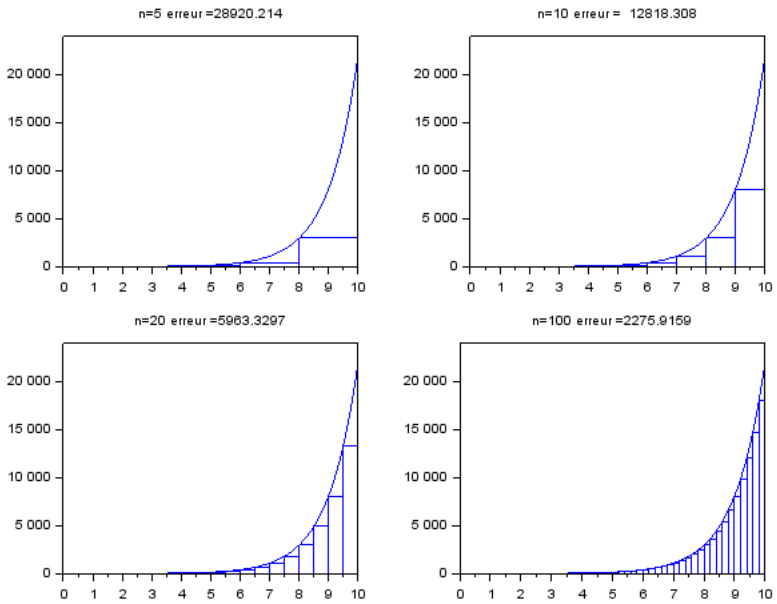


FIGURE 3.11 – Approche de l'intégrale de $f(x) = \exp(x)$ par des schémas d'Euler

Chapitre 4

Problèmes de moindres carrés et régression

Lorsque l'on réalise des études expérimentales, on obtient des données que l'on peut représenter graphiquement. Généralement, on obtient un certain nombre de points plus ou moins éparpillés avec lesquels on souhaite établir un modèle.

L'exemple typique est celui de la régression linéaire simple qui correspond à tracer la droite passant "au mieux" par tous les points tracés lorsque l'on étudie un phénomène avec des relations de proportionnalité entre les différentes données.

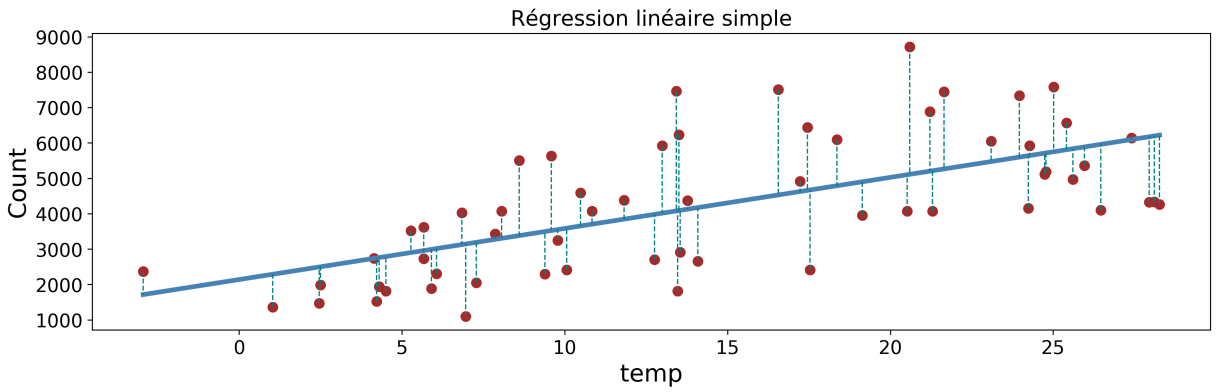


FIGURE 4.1 – exemple de régression linéaire simple

Dans ce chapitre nous verrons qu'il est possible d'établir un modèle pour une très grande variété de situations linéaires ou non linéaires.

4.1 méthode des moindres carrés

La notion d'approximation "au mieux" d'un ensemble de points par une courbe peut être interprétée de différentes manières mais la plus courante et l'approximation par la méthode des moindres carrés. Pour comprendre cette méthode il faut d'abord comprendre ce qu'est un vecteur résidu.

Vecteur résidu :

Un vecteur résidu r_i correspond à l'erreur entre la valeur d'un point y_i appartenant à l'ensemble des points que l'on cherche à approcher par une fonction f, et le point $f(x_i, \theta)$ avec θ les différents paramètres impliqués dans la fonction f. Ainsi :

$$r_i = y_i - f(x_i, \theta)$$

La méthode des moindres correspond à la minimisation de la somme de tous les vecteurs résidus élevés au carré pour éviter que les résidus négatifs et positifs ne s'annulent. Cette somme est généralement appelée somme des carrés des résidus.

Somme des carrés des résidus :

La somme des carrés des résidus $S(\theta)$ correspond à la formule :

$$S(\theta) = \sum_{i=1}^{n} r_i^2(\theta)$$

Dans la pratique, on commence par déterminer par quel type de fonction (f) on veut approcher le modèle (polynomiale, exponentielle...). On exprime ensuite (f) en fonction de différents paramètres inconnus (mis sous la forme d'un vecteur θ) que l'on cherche à déterminer. Ensuite, on cherche un vecteur $\hat{\theta}$ tel que la somme des carrés des résidus soit minimale. C'est à dire :

$$\forall \theta \in \mathbb{R}^p, S(\hat{\theta}) \leq S(\theta) \Leftrightarrow \hat{\theta}) = arg\min_{\{\theta \in \mathbb{R}^p\}} S(\theta)$$

4.2 moindres carrés linéaires

Une approximation par la méthode des moindres carrés est linéaire par rapport à θ si et seulement si :

$$y = \sum_{j=1}^{p} \theta_j \phi_j(x), \phi_k : \mathbb{R} \to \mathbb{R}$$

Ainsi, la linéarité d'un problème de moindre carrés n'est pas du tout liée à la linéarité de la fonction ϕ mais simplement au fait que ϕ ne dépende que de x et soit donc indépendante de θ

4.2.1 exemple : régression linéaire simple

Par exemple pour la régression linéaire simple,

$$S(\theta) = \sum_{i=1}^{n} (\theta_1 + \theta_2 x_i - y_i)^2 = ||r(\theta)||^2$$

La composante i du vecteur résidu s'écrit alors

$$r_i(\theta) = [1, x_i] \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} - y_i$$

Ainsi, $r(\theta) = A\theta - y$, avec $y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$ et $A = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$

On comprend alors facilement pourquoi ce type de problème de moindres carrés est linéaire : le vecteur résidu est une combinaison linéaire de θ et de y.

4.2.2 cas général

Dans le cas général, la somme des carrés des résidus d'un problème de moindres carrés linéaire s'écrit sous cette forme :

$$S(\theta) = \sum_{i=1}^{n} (\sum_{j=1}^{p} \theta_j \phi_j(x_i) - y_i)^2 = ||r(\theta)||^2$$

La composante i du vecteur résidu s'écrit alors

$$r_i(\theta) = [\phi_1(x_i), ..., \phi_p(x_i)] \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_p \end{bmatrix} - y_i$$

Et donc le vecteur résidu s'écrit $r(\theta) = A\theta - y$ avec A de dimension $n \times p$ et $a_{ij} = \phi_j(x_i)$.

Pour résoudre le problème de moindres carrés il faut chercher

$$\hat{\theta} = arg \min_{\{\theta \in \mathbb{R}^p\}} S(\theta) = ||A\theta - y||^2$$

Une condition nécessaire est $\nabla S(\hat{\theta}) = 0$
Pour calculer $\nabla S(\theta)$, on développe $S(\theta + h)$:

$$\begin{aligned} S(\theta + h) = ||A(\theta + h) - y||^2 &= ||A\theta - y + Ah||^2 \\ &= (A\theta - y + Ah)^T (A\theta - y + Ah) \\ &= ||A\theta - y||^2 + 2(A\theta - y)^T Ah + ||Ah||^2 \\ &= S(\theta) + \nabla S(\theta)^T h + ||Ah||^2 \end{aligned}$$

Par identification on obtient $\nabla S(\theta) = 2A^T(A\theta - y)$.

Ainsi, $\nabla S(\hat{\theta}) = 0$ implique $A^T A\hat{\theta} = A^T y$.

Ainsi, $\hat{\theta}$ est la solution de l'équation $A^T A\hat{\theta} = A^T y$.

Preuve :

$$\begin{aligned} S(\theta) &= S(\hat{\theta} + \theta - \hat{\theta}) \\ &= S(\hat{\theta}) + \nabla S(\hat{\theta})^T(\theta - \hat{\theta}) + ||A(\theta - \hat{\theta})||^2 \\ &= S(\hat{\theta}) + ||A(\theta - \hat{\theta})||^2 \\ &\geq S(\hat{\theta}) \end{aligned}$$

Si le rang de A est égal à p alors $\hat{\theta}$ est unique.

Unicité :

$$\begin{aligned} S(\hat{\theta}) = S(\theta) &\Leftrightarrow ||A(\theta - \hat{\theta})||^2 = 0 \\ &\Leftrightarrow A(\theta - \hat{\theta}) = 0 \\ &\Leftrightarrow \theta = \hat{\theta} \end{aligned}$$

Sur scilab, pour approcher la solution de l'équation $A^T A\hat{\theta} = A^T y$ il faut utiliser la division à gauche :

```
theta=A\y
```

qui détermine l'unique solution de l'équation si A est carré et inversible ou alors la solution telle que $||Ax - y||$ soit minimale. Même si mathématiquement écrire sur scilab :

```
theta=(A'*A)\ (A'*y)
```

est équivalent à la commande

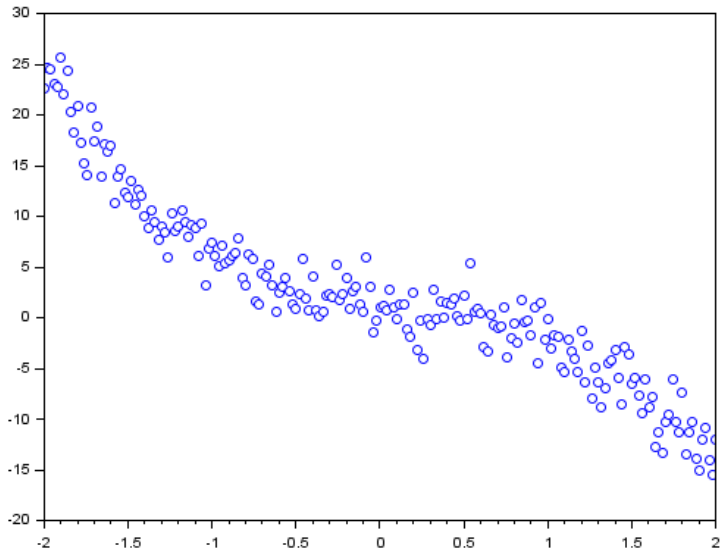
```
theta=A\y
```

, il est plus intéressant d'utiliser la deuxième option qui est bien plus stable, précise et compacte.

4.2.3 exemples

fonction polynomiale :

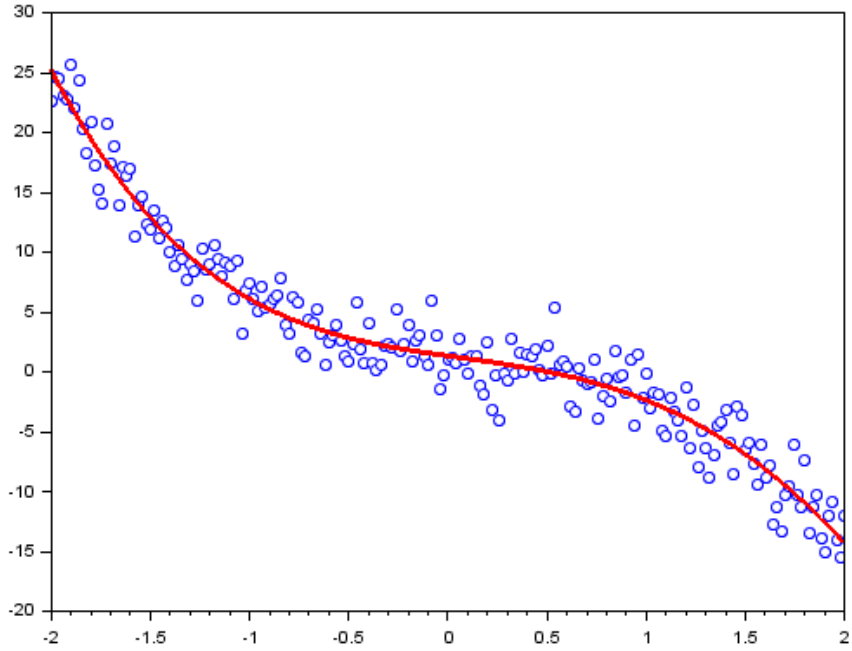
Nous allons ici tenter d'approcher les points ci-dessous par une fonction polynomiale de degré 3.



Pour y parvenir nous avons programmé le petit programme ci-dessous.

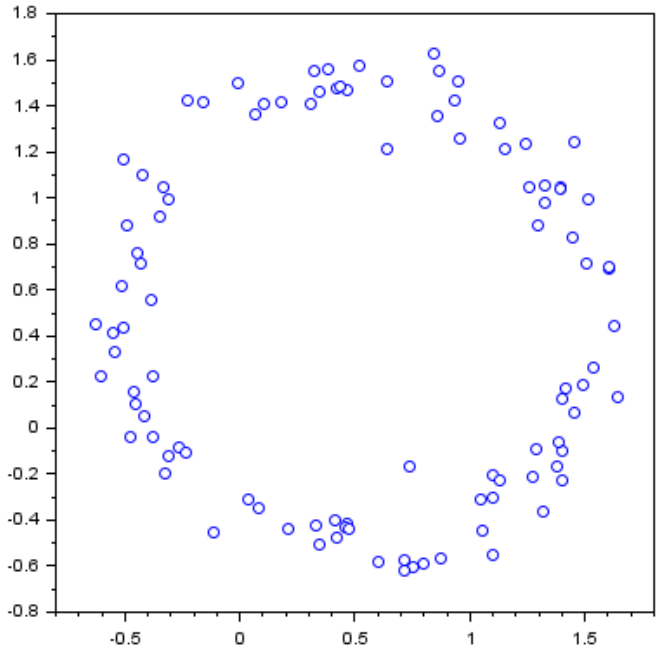
```
load data1.sod
plot(t,y,'o')
n=length(y);
p=3;
A=zeros(n,p+1);
for k=0:p
    A(:,k+1)=t.^k;
end
theta=A\y;
plot (t,A*theta,'r',"thickness",3)
```

Ce qui nous donne le résultat ci-dessous qui est très satisfaisant.



Cercle :

Un autre exemple très intéressant et celui de l'approximation de points par un cercle. Nous allons ici approcher les points ci-dessous par un cercle.



L'objectif va être de minimiser la distance algébrique suivante :

$$d(a,b,R) = \sum_{i=1}^{n}((x_i - a)^2 + (y_i - b)^2 - R^2)^2 = ||r||^2$$

qui correspond à l'équation d'un cercle de rayon R et de centre(a,b), paramètres que l'on cherche à déterminer grâce à une régression linéaire.

On observe facilement que le vecteur résidu r est non linéaire par rapport au vecteur (a,b,R). Mais si on développe l'expression d'une composante i de r on obtient le résultat suivant :

$$\begin{aligned} r_i &= R^2 - a^2 - b^2 + 2ax_i + 2by_i - (x_i^2 + y_i^2 \\ &= [2x_i, 2y_i, 1] \begin{bmatrix} a \\ b \\ R^2 - a^2 - b^2 \end{bmatrix} - (x_i^2 + y_i^2) \end{aligned}$$

Par conséquent, le vecteur résidu est linéaire par rapport au vecteur :

$$\theta = [a, b, R^2 - a^2 - b^2]$$

Le problème de régression linéaire pour approcher le cercle peut alors se mettre sous la forme $r(\theta) = A\theta - z$,

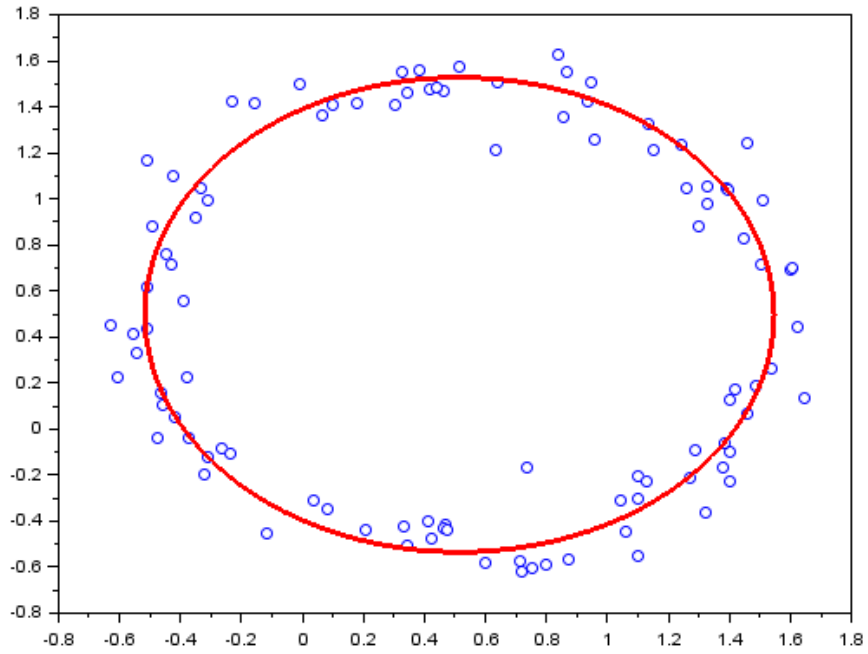
$$\text{avec } A = \begin{bmatrix} 2x_1 & 2y_1 & 1 \\ \vdots & \vdots & \vdots \\ 2x_n & 2x_n & 1 \end{bmatrix} \text{ et } z = \begin{bmatrix} x_1^2 + y_1^2 \\ \vdots \\ x_n^2 + y_n^2 \end{bmatrix}$$

On peut alors trouver le vecteur θ grâce à scilab. On peut alors en déduire les valeurs de a,b et R : $a = \theta(1)$, $b = \theta(2)$ et $R = \sqrt{\theta(3) + a^2 + b^2}$

on peut alors programmer un petit programme scilab :

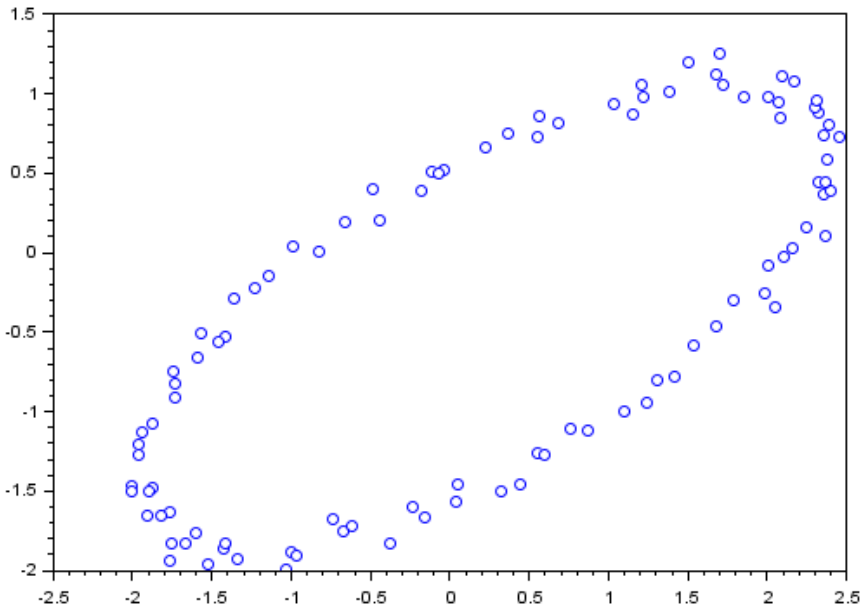
```
A=[2*x,2*y,ones(x)];
z=x.^2+y.^2;

theta=A\z;
a=theta(1);
b=theta(2);
R=sqrt(theta(3)+a^2+b^2);

t=linspace(0,2*%pi,100);

plot(x,y,"o");
plot(a+R*cos(t),b+R*sin(t),'r',"thickness",3);
```

On obtient alors le résultat ci-dessous :



Ellipse :

Dans cet exemple, nous allons généraliser ce que nous avons vu pour le cercle, pour les ellipses en approchant les points ci-dessous par une ellipse.



L'objectif va être de minimiser la distance algébrique suivante :

$$d(\alpha, \beta, \gamma, c1, c2) = \sum_{i=1}^{n} ((x_i - c)^T M (x_i - c) - \gamma^2)^2$$

avec $x = (x, y)^T$, $M = \begin{pmatrix} 1 & \alpha \\ \alpha & \beta \end{pmatrix}$ et $c = (c_1, c_2)^T$.

Cela correspond à l'équation d'une ellipse de centre $c = (c_1, c_2)^T$. On peut remarquer que dans le cas du cercle, $\alpha = 0$ et $\beta = 1$.

On cherche à déterminer la valeur des paramètres $(\alpha, \beta, \gamma, c1, c2)$ grâce à une régression linéaire. Comme pour le cercle, le vecteur résidu r est non linéaire par rapport au vecteur $(\alpha, \beta, \gamma, c1, c2)$. Mais si on développe l'expression d'une composante i de r on obtient le résultat

suivant :

$$\begin{aligned} r_i &= x_i^T M x_i - 2x_i^T M c + c^T M c - \gamma^2 \\ &= x_i^2 + 2\alpha x_i y_i + \beta y_i^2 - 2x_i(c_1 + \alpha c_2) - 2y_i(\alpha c_1 + \beta c_2) + c^T M c - \gamma^2 \\ &= x_i^2 + (2x_i y_i, y_i^2, 2x_i, 2y_i, 1)(\alpha, \beta, c_1 + \alpha c_2, \alpha c_1 + \beta c_2, c^T M c - \gamma^2)^T \end{aligned}$$

Par conséquent, le vecteur résidu est linéaire par rapport au vecteur :

$$\theta = [\alpha, \beta, c_1 + \alpha c_2, \alpha c_1 + \beta c_2, c^T M c - \gamma^2]^T$$

Le problème de régression linéaire pour approcher l'ellipse peut alors se mettre sous la forme $r(\theta) = A\theta - b$,

$$\text{avec } A = \begin{bmatrix} 2x_1y_1 & 2y_1^2 & -2x_i & -2y_i & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 2x_ny_1 & 2y_n^2 & -2x_n & -2y_n & 1 \end{bmatrix} \text{ et } b = \begin{bmatrix} -x_1^2 \\ \vdots \\ -x_n^2 \end{bmatrix}$$

On peut alors trouver le vecteur θ sur scilab.

Comme pour le cercle il faut à présent en déduire les valeurs de$\alpha, \beta, \gamma, c1, c2$.

Par identification, $\alpha = \theta(1)$, $\beta = \theta(2)$, $\gamma = \sqrt{c^T M c - \theta(5)}$.

Pour déterminer c_1 et c_2, il faut résoudre le système $Mc = \begin{pmatrix} p_3 \\ p_4 \end{pmatrix}$

Pour tracer l'ellipse, il est judicieux de passer dans la base des vecteurs propres de la matrice M : il existe une matrice orthogonale P telle que $D = P^T M P$ avec $D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$

En posant le changement de repère $x = c+Pu$, l'équation de l'ellipse s'écrit :

$$\begin{aligned}(x-c)^T M(x-c) &= u^T P^T M P u \\ &= u^T D u \\ &= \lambda_1 {u_1}^2 + \lambda_2 {u_2}^2 = \gamma^2\end{aligned}$$

On a donc le paramétrage $u_1 = \frac{\gamma}{\sqrt{\lambda_1}}\cos(t)$, $u_2 = \frac{\gamma}{\sqrt{\lambda_2}}\sin(t)$ dans le repère propre.

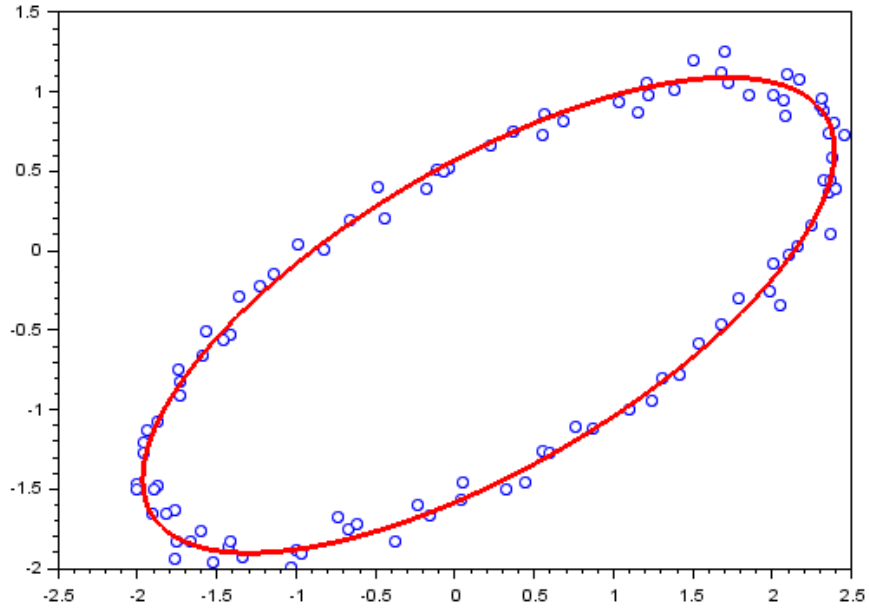
Ainsi, dans le repère initial :

$$x = c + \gamma P \begin{pmatrix} \frac{\cos(t)}{\sqrt{\lambda_1}} \\ \frac{\sin(t)}{\sqrt{\lambda_2}} \end{pmatrix}$$

On dispose donc de tous les éléments pour réaliser un petit programme scilab :

```
load data2.sod;
plot(x,y,'o');
isoview on

A = [2*x.*y y.^2 -2*x -2*y ones(x)];
p = A \ (-x.^2);
alpha = p(1);
bet=p(2);
M=[1 alpha;alpha bet];
c=M\p(3:4);
gam = sqrt (c'*M*c-p(5));

[P,D] =spec(M)
t = linspace(0,2*%pi,500);
X=gam*P*[cos(t)/sqrt(D(1,1));sin(t)/sqrt(D(2,2))];

plot(c(1)+X(1,:),c(2)+X(2,:),"r","thickness",3)
```

On obtient alors le résultat ci-dessous :



4.2.4 choix du modèle

Dans l'exemple de la fonction polynomiale, on a choisi d'approcher les points par un polynôme de degré 3 mais on aurait très bien pu utiliser un polynôme d'un autre degré. On peut donc se demander comment choisir le modèle le plus adapté pour réaliser une approximation par les moindres carrés.

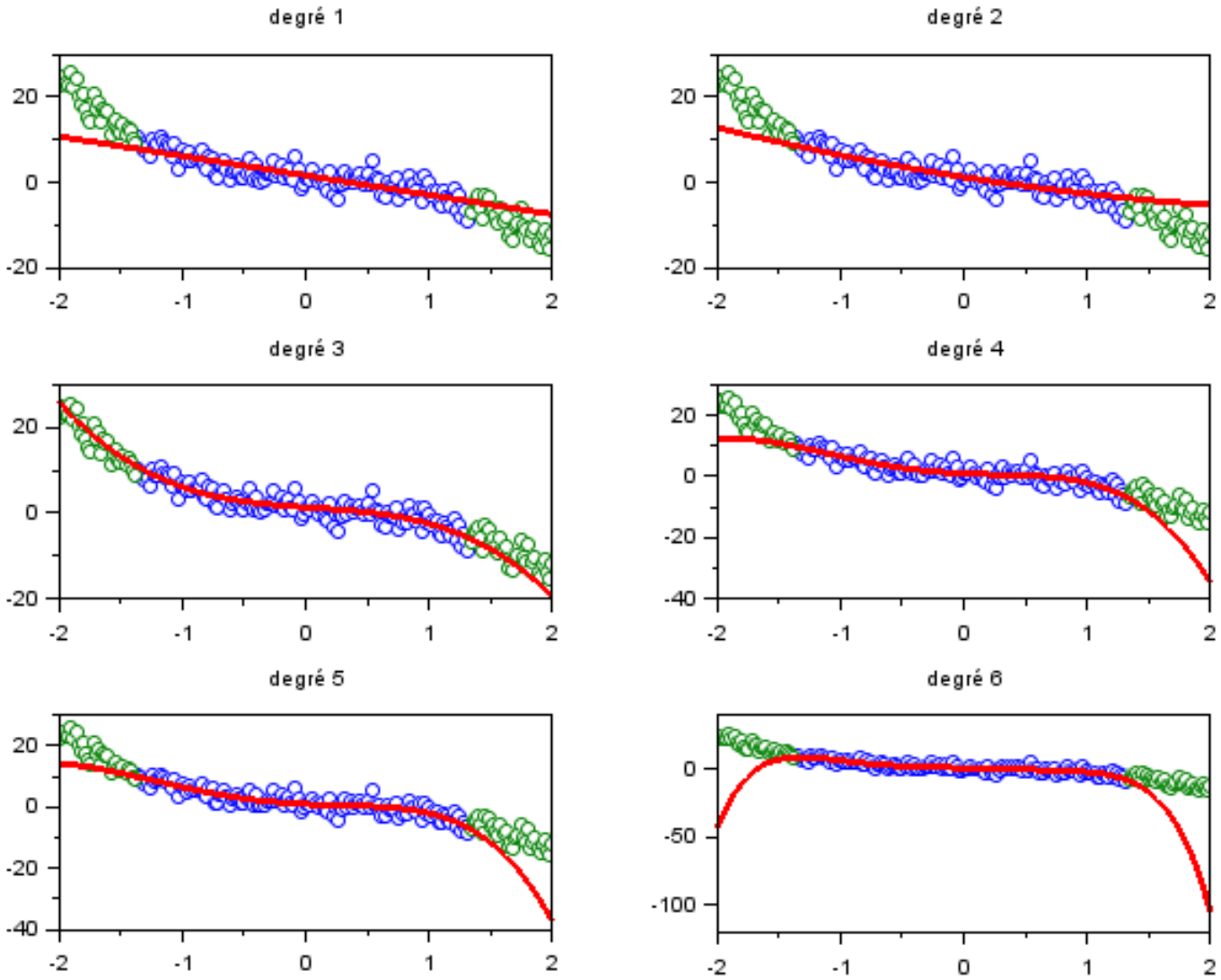
L'astuce est de séparer les données que l'on cherche à approcher en deux ensembles distincts : un ensemble de validation et un ensemble de travail.
On réalisera alors des régressions linéaires pour les différents modèles sur l'ensemble de travail et donc uniquement sur une partie des données.
L'ensemble de validation joue quant à lui le rôle de nouvelles données et nous permet de voir si le modèle utilisé correspond bien à la réalité en approchant bien toutes les données, y compris celles qui ne sont pas utilisées dans la régression.

On pourra alors comparer l'erreur résiduelle sur l'ensemble de validation et celle sur l'ensemble de travail et nous choisirons alors le modèle pour lequel l'erreur sur les deux ensembles et la plus proche.

Si on reprend l'exemple précédent, à l'aide du code scilab ci-dessous, on peut réaliser la validation pour des approximation par des polynômes de différents degrés :

```
load data1.sod
plot(t,y,'o')
n=length(y);

//Ensemble d'apprentissage
T=25:n-25;;
V=setdiff(1:n,T);
ST=zeros(9,1);
SV=zeros(9,1);

p=5;
A=zeros(n,p+1);
for k=0:p
    A(:,k+1)=t.^k;
end

//Resolution de Pb demoindres carrés sur l'
ensemble d"apprentissage uniquement
theta=A(T,:)\y(T);
clf

plot(t(T),y(T),'o',t(V),y(V),'o');
plot (t,A*theta,'r',"thickness",3)
```

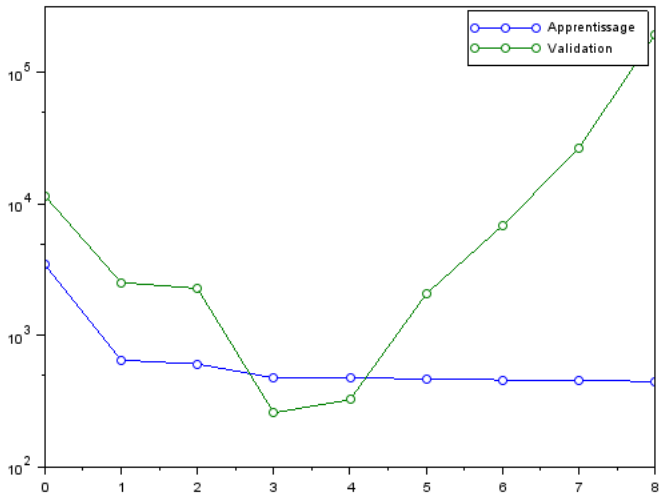
En utilisant cet algorithme avec différentes valeurs de p, on obtient la figure suivante :



Le polynome de degré 3 approche le mieux les points. Pour le prouver il suffit de comparer l'erreur sur l'ensemble de validation et celle sur l'ensemble de travail à l'aide du programme scilab ci-dessous :

```
load data1.sod
n=length(y);
T=25:n-25;
V=setdiff(1:n,T);
ST=zeros(9,1);
SV=zeros(9,1);
for p=0:8;
A=zeros(n,p+1);
for k=0:p
    A(:,k+1)=t.^k;
end
theta=A(T,:)\y(T);
ST(p+1)=norm(A(T,:)*theta-y(T))^2;
SV(p+1)=norm(A(V,:)*theta-y(V))^2;
end
plot(0:8,ST,'-o',0:8,SV,'-o')
gca().log_flags='nl';
```

On obtient alors la figure suivante qui nous prouve que le polynome de degré 3 est celui pour lequel l'erreur de validation est la plus faible.



4.3 moindres carrés non linéaires

4.3.1 méthode Levenberg-Marquardt

Si on essaye d'approcher un modèle par une fonction ϕ dépendant de x et de θ la méthode des moindres carrés n'est plus linéaire par rapport à θ :

$$y = \sum_{j=1}^{p} \theta_j \phi_j(x, \theta), \phi_k : \mathbb{R} \to \mathbb{R}$$

Ce que nous avons vu précédemment n'est donc plus valable.
Pour résoudre un problème non linéaire de moindres carrés, on va s'y prendre de manière itérative :

— on pose θ_0

— pour passer de θ_k à θ_{k+1} on considère le développement de Taylor du vecteur résidu θ_k :

$$r(\theta) = r(\theta_k) + r'(\theta_k)(\theta - \theta_k) + ||(\theta - \theta_k||\epsilon(\theta - \theta_k)$$

et on choisi θ_{k+1} tel que $||r(\theta_k)+r'(\theta_k)(\theta_{k+1}-\theta_k)||^2$ soit minimale.

Il s'agit à présent d'un problème linéaire qui peut être mis sous la forme :

$$\theta_{k+1} = \theta_k - [r'(\theta_k)^T r'(\theta_k)]^{-1} r'(\theta_k)^T r(\theta_k)$$

Cette méthode est appelée méthode de Gauss-Newton. Sur scilab cela peut s'écrire en utilisant l'opérateur de division à droite :

$$\theta_{k+1} = \theta_k - r'(\theta_k) \backslash r(\theta_k)$$

Cette méthode fonctionne mais seulement si le rang de $r'(\theta_k)$ est maximal.

Pour éviter ce problème, on applique la méthode de Levenberg-Marquardt qui consiste à utiliser un $\lambda > 0$ et prendre θ_{k+1} tel que

$$S_\lambda(\theta_{k+1} - \theta_k) = ||r(\theta_k) + r'(\theta_k)(\theta_{k+1} - \theta_k)||^2 + \lambda||(\theta_{k+1} - \theta_k)||^2$$

soit minimal.

En utilisant les matrices par bloc cela donne :

$$S_\lambda(\theta_{k+1}-\theta_k)=\left\|\begin{pmatrix}r'(\theta_k)\\ \lambda^{\frac{1}{2}}I\end{pmatrix}(\theta_{k+1}-\theta_k)+\begin{pmatrix}r(\theta_k)\\ 0\end{pmatrix}\right\|^2$$

Ce qui est toujours un problème linéaire et le vecteur résidu s'écrit donc :

$$\begin{pmatrix}r'(\theta_k)\\ \lambda^{\frac{1}{2}}I\end{pmatrix}(\theta_{k+1}-\theta_k)+\begin{pmatrix}r(\theta_k)\\ 0\end{pmatrix}$$

Et l'équation normale s'écrit alors :

$$(r'(\theta_k)^T r'(\theta_k)+\lambda I)(\theta_{k+1}-\theta_k)=r'(\theta_k)^T r(\theta_k)$$

La formule mathématique de la méthode de Levenberg-Marquardtest alors

$$\theta_{k+1}=\theta_k-[r'(\theta_k)^T r'(\theta_k+\lambda I)]^{-1}r'(\theta_k)^T r(\theta_k)$$

Sur scilab cela peut s'écrire en utilisant l'opérateur de division à droite :

$$\theta_{k+1}=\theta_k-\begin{pmatrix}r'(\theta_k)\\ \lambda^{\frac{1}{2}}I\end{pmatrix}\backslash\begin{pmatrix}r(\theta_k)\\ 0\end{pmatrix}$$

Exemple

Pour tester la méthode de Levenberg-Marquardt, on peut chercher à déterminer le minimum de la fonction

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

.

La première étape est de déterminer et de tracer les courbes iso-valeurs de f, c'est à dire la famille des courbes

$$C_\alpha=x\in\mathbb{R}^2, f(x)=\alpha$$

.

On peut remarquer que

$$f(x) = \alpha \Leftrightarrow (1 - x_1)^2 + 100(x_2 - x_1^2)^2 = \alpha$$
$$\Leftrightarrow \left(\frac{1 - x_1}{\sqrt{\alpha}}\right)^2 + \left(10\frac{x_2 - x_1^2}{\sqrt{\alpha}}\right)^2 = 1$$

on peut donc poser

$$\frac{1 - x_1}{\sqrt{\alpha}} = cos(t), 10\frac{x_2 - x_1^2}{\sqrt{\alpha}} = sin(t)$$

D'où le paramétrage pour C_α :

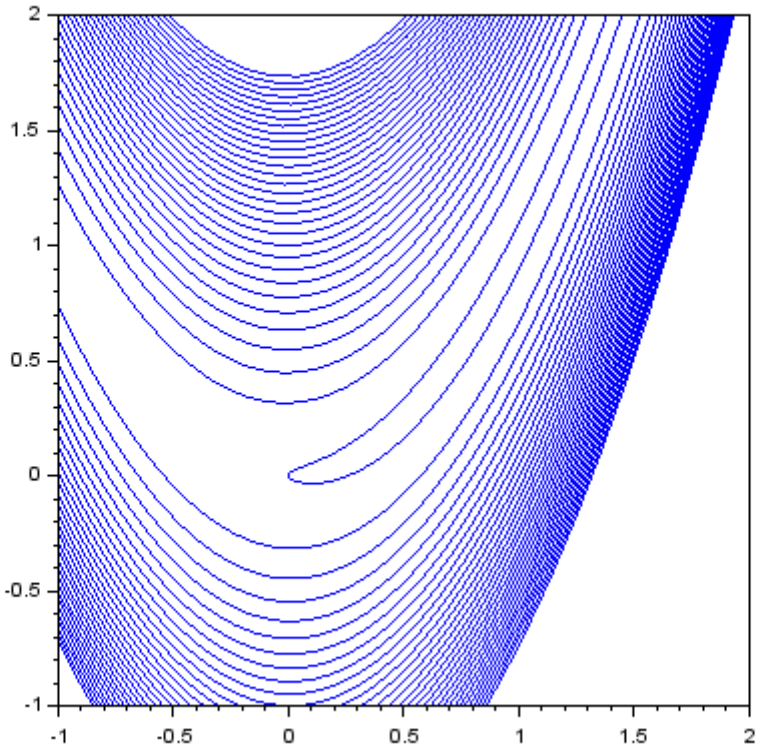
$$x_1(t) = 1 - \sqrt{\alpha}cos(t)$$
$$x_2(t) = \frac{\sqrt{\alpha}}{10}sin(t) + x_1(t)^2$$

$t \in [0, 2\pi]$

On peut aller tracer les courbes iso-valeur sur scilab à l'aide du code ci-dessous :

```
t=linspace(0,2*%pi,1000);
clf
isoview
gca().data_bounds = [-1 2 -1 2];
gca().auto_scale="off"
for alpha=1:10:301
    x1=1-sqrt(alpha)*cos(t);
    x2=sqrt(alpha)/10*sin(t)+x1.^2
    plot(x1,x2)
end
```

On obtient alors la figure ci-dessous :



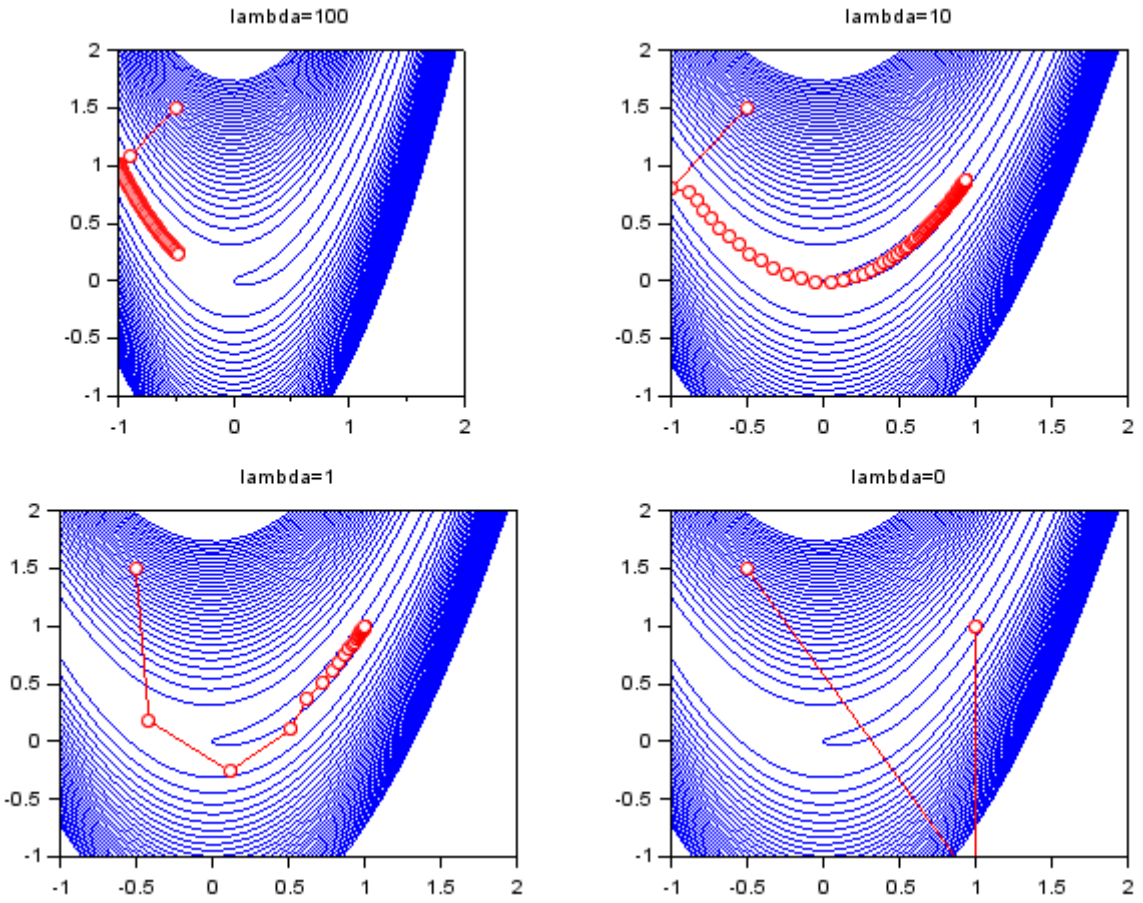
On peut ensuite calculer les résidus et la jacobienne de la fonction f à l'aide de deux fonctions scilab :

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

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

Et ensuite on peut appliquer la méthode de Levenberg-Marquardt avec le code ci-dessous :

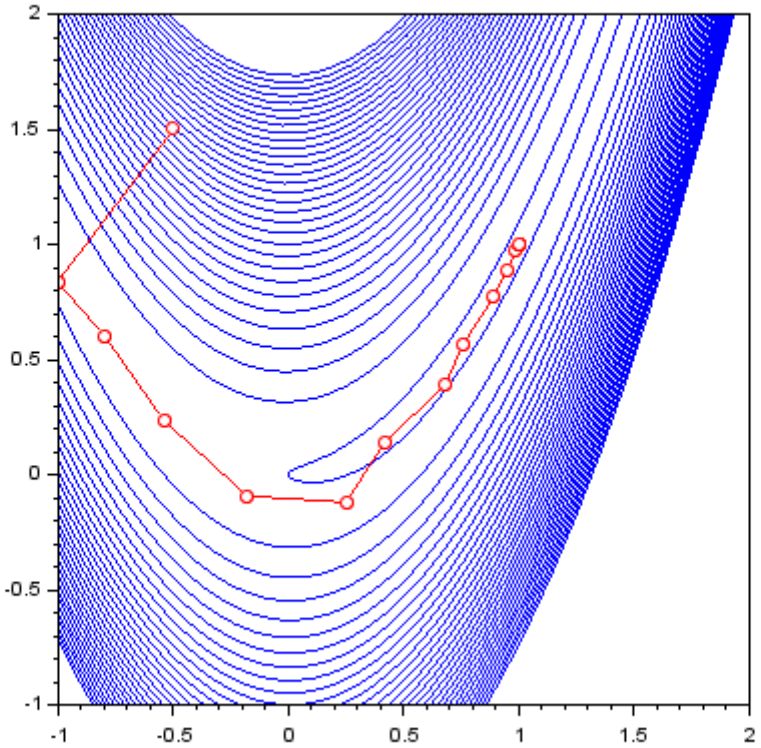
```
x=[-.5;1.5];
I=eye(2,2)
lambda=10;
for k=1:100
    r=resid(x);
    J=jac(x);
    h=[J;sqrt(lambda)*I]\ [r;0;0]
    plot([x(1) x(1)-h(1)],[x(2) x(2)-h(2)],'-or')
    x=x-h
    if norm(h)<1e-6
        break
     end
end
```

On obtient les résultats ci-dessous en faisant varier la valeur de lambda :



On constate que lorsque λ est trop grand, la convergence vers le minimum de f est très lente et nécessite énormément d'itérations alors que quand lambda est très faible voire nul, la convergence est vraiment brutale et même incohérente puisque les valeurs intermédiaires de l'algorithme sont extrêmement éloigné de la courbe des iso-valeurs.Il convient donc de choisir une valeur de λ faible mais pas trop.

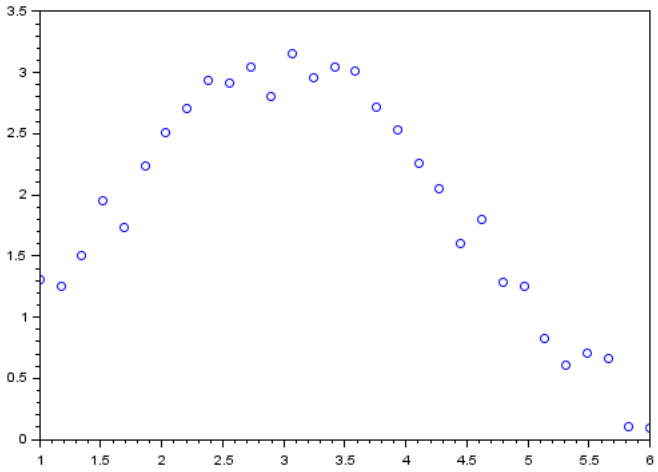
Une autre possibilité est de faire varier la valeur de λ. On peut notamment faire diminuer la valeur de λ au fur et à mesure que l'on s'approche de la solution finale. On peut par exemple affecter la valeur $||h||$ à lambda. On obtient alors la figure suivante :



La convergence dans cette situation est relativement rapide est stable et donc très intéressante.

4.3.2 macro lsqrsolve

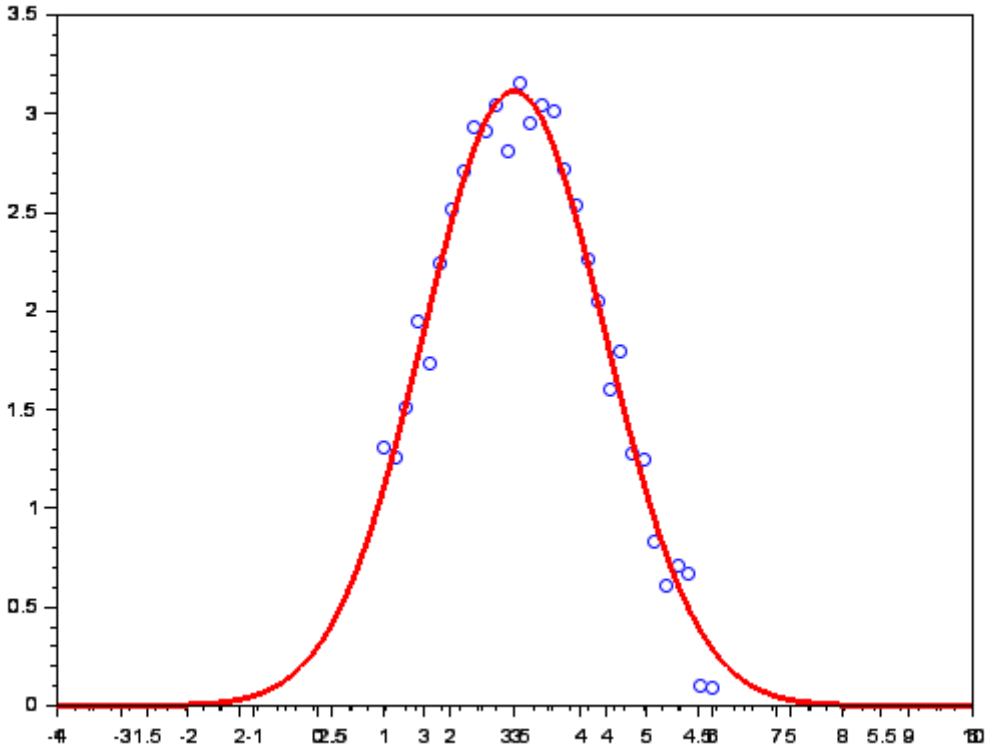
Sur scilab, il existe une fonction pré implantée dans le logiciel permettant de résoudre des problèmes de moindres carrés. Il s'agit de la fonction lsqrsolve. par exemple, nous pouvons approcher les points ci-dessous par une fonction de la forme $f(t) = aexp(-(t-\mu)^2/\sigma^2)$



Il suffit d'écrire une fonction qui calcule le résidu, de choisir une valeur de θ_0 et ensuite d'utiliser lsqrsolve, ce qui est fait dans le code ci-dessous :

```
a1 = newaxes();
plot(t,y,"o")
function z=resid(theta,m)
    z=theta(1)*exp(-(t-theta(2))^2./theta(3)^2)-y
endfunction
theta0=[1;1;1]'
Y=lsqrsolve(theta0,resid,length(y))
x=linspace(-4,10,1000)
plot(x,Y(1)*exp(-(x-Y(2))^2./Y(3)^2),"r","
thickness",3)
a1.data_bounds=[-4␣0;10␣3.5];
```

On obtient alors la figure ci-dessous qui est très satisfaisante.



Chapitre 5

Valeurs propres

En algèbre, il est très souvent nécessaire de déterminer les valeurs propres et vecteurs propres d'une matrice nitamment pour pouvoir effectuer sa diagonalisation. Déterminer les valeurs propres d'une matrice peut être long et complexe et il est donc très intéressant de pouvoir le faire numériquement.

5.1 méthode de la puissance

La première méthode pour déterminer les valeurs propres d'une matrice est appelée méthode de la puissance.

Méthode de la puissance :
Soit A une matrice réelle $n \times n$, diagonalisable et de valeurs propres $(\lambda_i)_{i=1...n}$ avec $|\lambda_1| > |\lambda_2| \geq ...|\lambda_n|$ et $y_1, ...y_n$ les vecteurs propres associés vérifant $||y_i|| = 1$
Soit $x_0 \notin \text{Vect}(y_2, ..., y_n)$. La suite (x_k) définie par

$$x_{k+1} = \frac{Ax_k}{||Ax_k||}$$

converge dans le sous espace propre associé à y_1 au sens où

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

Pour tester cet algorithme, j'ai écrit le code scilab mis en annexe dont la fonction principale est celle ci-dessous qui détermine la valeur

propre maximale d'une matrice aléatoire avec une tolérance que l'on peut faire varier. j'ai aussi mis en place un compteur pour déterminer le nombre d'itération réalisées :

```
function y=puissance(M,z)
    for i=1:100
    if norm(M*z-(z'*M*z)*z)<Tol then break
        else
    z=(M*z)/norm(M*z);
    compteur=i
    y=z'*M*z
end
end
endfunction
```

Pour le premier essai j'ai obtenu :

```
--> exec('C:\Users\merinos\puissance.sce', -1)

        column 1 to 3
  0.4217656   0.1993437   0.5980196
  0.1034854   0.7131302   0.5549106
  0.4279759   0.5208952   0.8552953
  0.7860729   0.9311723   0.3097751
  0.8568082   0.4143836   0.9446128
        column 4 to 5
  0.244268    0.0705419
  0.8760447   0.7338807
  0.4874215   0.7792181
  0.3844019   0.7827638
  0.0922345   0.2161568

  2.6762608

  7.
```

L'algorithme a réalisé 7 itérations. J'ai réalisé d'autres essais et généralement, le nombre d'itérations varie entre 7 et 12.

5.2 méthode de la puissance inverse

Méthode de la puissance inverse :
Soit $x_0 \notin \text{Vect}(y_1, ..., y_{n-1}$. Si on note z_k la solution du système linéaire $Az_k = x_k$, la suite (x_k) définie par

$$x_{k+1} = \frac{z_k}{||z_k||}$$

converge dans le sous espace propre associé à y_n au sens où

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

Pour tester cet algorithme, j'ai écrit le code scilab mis en annexe dont la fonction principale est celle ci-dessous qui détermine la valeur propre de module minimal. Le programme ressemble énormément au programme précédent mis à part l'utilisation de la division à gauche :

```
function y=puissance_inv(M,z)
    for i=1:100
    if norm(M*z-(z'*M*z)*z)<Tol then break
        else
    z=(M\z)/norm(M\z);
    compteur=i
    y=z'*M*z
end
end
endfunction
```

Pour le premier essai j'ai obtenu :

```
--> exec('C:\Users\merinos\puissance.sce', -1)

        column 1 to 3
   0.2651769   0.3051453   0.0663265
   0.0640081   0.4280301   0.4792961
   0.5873902   0.8355348   0.3171234
   0.9278804   0.208429    0.023136
   0.4108539   0.5201674   0.2121407
        column 4 to 5
   0.2952926   0.9267992
   0.0740681   0.9292422
   0.9572983   0.8083564
   0.3095793   0.2290231
   0.0761205   0.1606745

  -0.3315433

   41.

   1.9178827 + 0.i
   0.2712689 + 0.3410999i
   0.2712689 - 0.3410999i
  -0.6482929 + 0.i
  -0.3315435 + 0.i
```

L'algorithme a réalisé 41 itérations. J'ai réalisé d'autres essais et généralement, le nombre d'itérations varie entre 10 voir plus 100 itérations. Ainsi, le nombre d'itérations et généralement plus important que pour la méthode de la puissance.

5.3 macro spec

Sur scilab, une fonction permettant de déterminer les valeurs propres et même les vecteursd'une matrice est déjà implémentée dans le logiciel. Il s'agit de la macro spec. Si l'on souhaite uniquement connaitre les valeurs propres d'une matrice, il suffit de taper :

```
spec(M)
```

et scilab affichera directement les valeurs propres de la matrice M. Par contre si l'on écrit le code suivant :

```
[P,D]=spec(M)
```

Scilab affichera en P les différents vecteurs propres sous la forme d'une matrice dont la colonne i correspond au vecteur propre i ; en D, scilab affichera une matrice diagonale dont chaque composante non nulle de la colonne i correspondra à la valeur propre i associée au vecteur propre i.

5.4 Applications :

5.4.1 Résonance dans les systèmes mécaniques

La première application des valeurs propre que nous allons réaliser est l'étude de la résonance dans les systèmes mécaniques. Le système que nous allons étudier est constitué de deux ressorts r_1 et r_2 de raideur respectives k_1 et k_2. Une extrémité de r_1 est encastrée et l'autre extrémité est reliée à une masse m_1 ; cette dernière est aussi reliée à une des extrémité der_2 ; et enfin l'autre extrémité de r_2 est reliée à une masse m_2 sur laquelle on applique une force $\vec{F}$. On suppose que les masses peuvent glisser sans frottement sur un axe, (voir la figure).

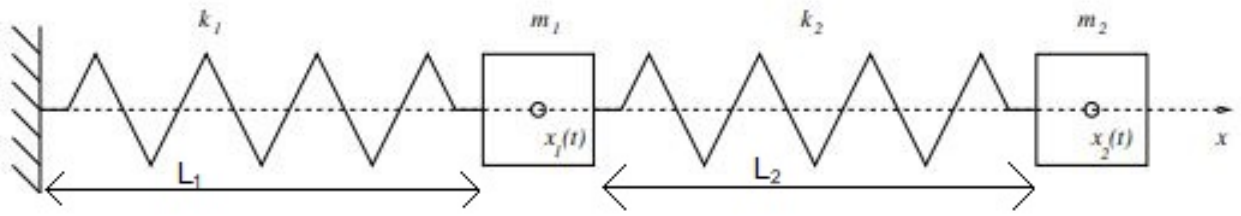


FIGURE 5.1 – Système à deux ressorts et deux masses

Grâce à une étude mécanique que nous n'avons pas réalisée on obtient les équations différentielles suivantes régissant les positions $(x_1(t))$ et $(x_2(t))$ du centre de chacune des masses.

$$\begin{cases} m_1\ddot{u}_1 = -(k_1+k_2)u_1 + k_2u_2 \\ m_2\ddot{u}_2 = k_2u_1 - k_2u_2 + f(t) \end{cases} \tag{5.1}$$

avec$f(t) = \sin(\omega t)$,$u_1 = x_1 - L_1$ et $u_2 = -L_1 - L_2$ où L_1, L_2 les longueurs au repos des ressorts 1 et 2.

Résoudre ce système d'équation différentielle est complexe puisqu'il s'agit de deux équations différentielles d'ordre 2 interdépendantes.

Matriciellement, en posant $u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$ cela devient

$$\ddot{u} + Au = \begin{pmatrix} 0 \\ \frac{1}{m_2} \end{pmatrix} f(t) \tag{5.2}$$

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

Pour résoudre cette équation différentielle, il est nécessaire de diagonaliser A : si on trouve une matrice P telle que $P^{-1}AP = D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$ avec λ_1, λ_2 les deux valeurs propre de A, en posant le changement de variable $u = Pv$ on peut alors réécrire l'équation (5.3) comme suit :

$$\ddot{v} + Dv = cf(t) \tag{5.3}$$

avec $c = P^{-1}\begin{pmatrix} 0 \\ \frac{1}{m_2} \end{pmatrix}$

Soit le système différentiel

$$\begin{cases} \ddot{v}_1 + \lambda_1 v_1 = c_1 f(t) \\ \ddot{v}_2 + \lambda_2 v_2 = c_2 f(t) \end{cases} \tag{5.4}$$

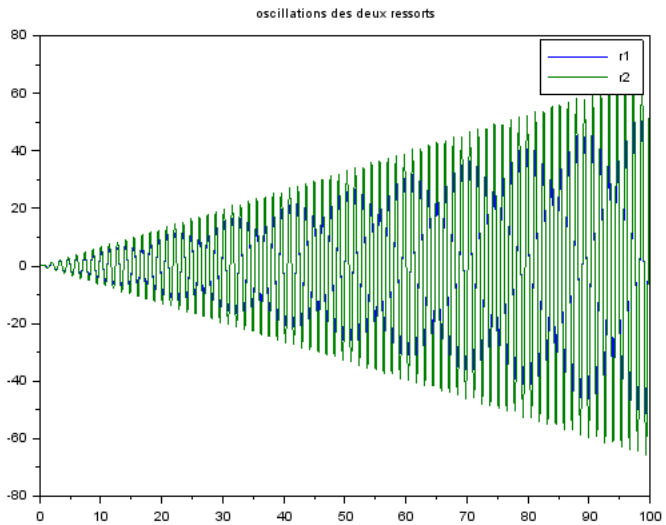
qui est bien plus simple à résoudre puisque les deux équations sont découplées et peuvent alors être traitées séparément, ce que nous savons faire grâce au chapitre 3 avec ode.

Si l'on admet que λ_1 et λ_2 sont positives, il existe deux pulsations propre $\omega_1 = \sqrt{\lambda_1}$ et $\omega_2 = \sqrt{\lambda_2}$

Sur scilab, ce problème se résout très facilement avec la fonction spec, en prenant la première valeur propre (on peut aussi utiliser la seconde). Le code que j'ai utilisé est présenté ci-dessous :

```
function vprime=F(t,v)
    u=v(1:2);
    uprime=v(3:4);
    vprime = [uprime
              -A*u+[0;1/m2]*sin(sqrt(omega(1))*t)]
endfunction
k1=5;
k2=10;
m1=0.1;
m2=0.1;
A=[(k1+k2)/m1 -k2/m1
      -k2/m2 k2/m2]
omega=spec(A);
t=linspace(0,100,1000);
v=ode(zeros(4,1),0,t,F);
clf
plot(t,v(1:2,:))
```

On obtient alors le graphique suivant qui nous montre que le système entre bien en résonance pour une valeur de $\omega = 21.92$ rad/s.



5.4.2 Page Rank

Google est le moteur de recherche le plus utilisé et permet d'ordonner les résultat d'une recherche du plus pertinent au moins pertinent. L'algorithme utilisé par google pour y parvenir est l'algorithme page rank qui permet suivant le mot clé entré dans la barre de recherche d'attribuer une note à un ensemble de page web et d'afficher ces pages web par ordre décroissant suivant leur note.

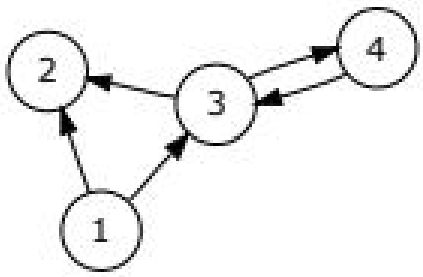


FIGURE 5.2 – graphe orienté de 4 pages web

Dans la situation ci-dessus, on a représenté le graphe orienté d'un ensemble E de 4 pages web et les liens les associant $A \subset [(x, y) \in E^2, x \neq y]$ On va alors construire une matrice carré C appelée matrice d'adjacence telle que dans le cas général

$$c(i,j) = \begin{cases} 1 \text{ si il existe un lien de } P_j \text{ vers } P_i \\ 0 \text{ sinon} \end{cases}$$

Ainsi, dans l'exemple précédent, la matrice d'adjacence est

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

Nous allons étudier l'exemple des 3404 pages web du site internet de l'UTC, dont la matrice d'adjacence est la suivante (les points bleus correspondent aux éléments non nuls :

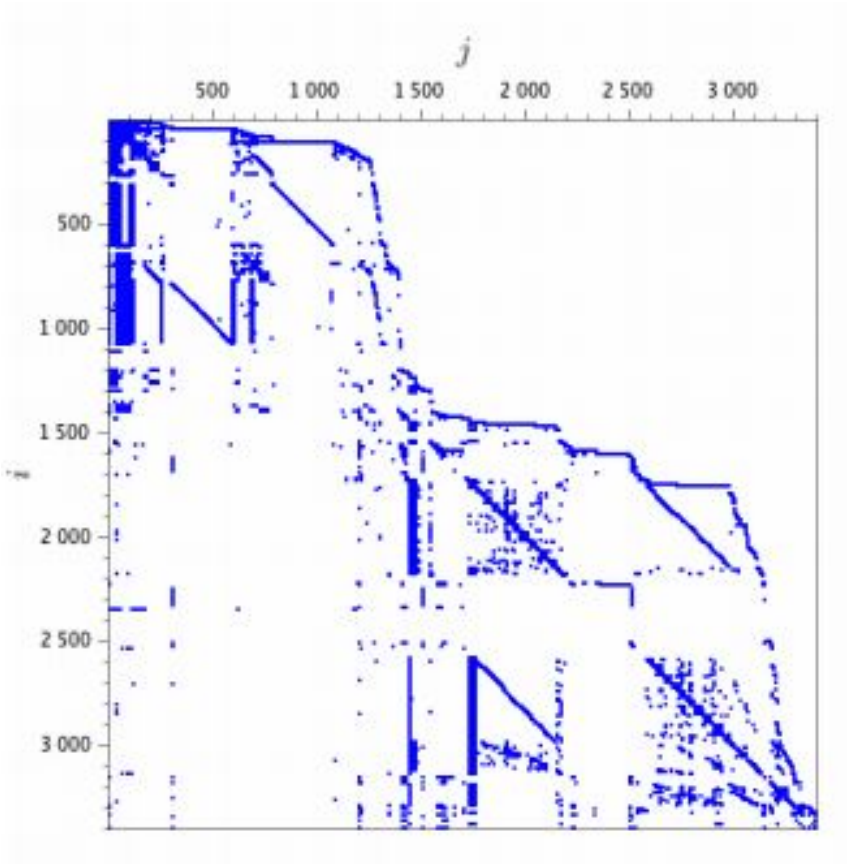


FIGURE 5.3 – matrice d'adjacence du site de l'utc

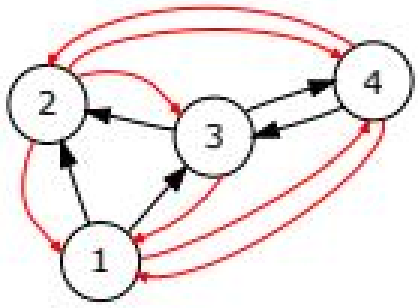


FIGURE 5.4 – graphe orienté de 4 pages web

On vient ensuite compléter ce modèle par les liens inexistants (en rouge sur la figure ci-dessus) qui correspondent au choix aléatoire d'une des pages web par un utilisateur du web. Dans ce nouveau modèle, on considère qu'a partir d'une page donnée i il y a :

- une probabilité $\frac{1-q}{n}$ pour aller aléatoirement sur une des n pages
- une probabilité $\frac{p}{\sum_{j=1}^{n} c_{ij}}$ de suivre aléatoirement un des liens de la page
- une probabilité $\frac{1}{n}$dans le cas ou la page n'a pas de liens vers d'autres page et donc l'utilisateur ne peux que choisir aléatoirement de manière équiprobable une autre page sans passer par un lien

Ainsi, la probabilité de passer de la page i a la page j est égale à :

$$p_{ij} = \begin{cases} \frac{1}{n} \text{ si } \sum_{j=1}^{n} c_{ij} = 0 \\ \frac{pc_{ij}}{\sum_{j=1}^{n} c_{ij} + \frac{1-q}{n}} \text{ sinon} \end{cases}$$

Généralement on prend q=0.85.
Dans l'exemple précédent, la matrice p vaut donc

$$P = q \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 \end{pmatrix} + \frac{1-q}{4} e e^T$$

avec $e^T = (1, 1, 1, 1)$ Mathématiquement, il s'agit d'un processus de Markov.

Processus de Markov

Soit E=1,...,n. Un processus de Markov est une suite de variables aléatoires $(X_k)_{k\in\mathbb{N}}$prenant leurs valeurs dans E et définies par

$$\forall n \geq 0, \forall (i,j) \in E^2, p_i j = Prob(X_{n+1} = j | X_n = i)$$

où P est une matrice stochastique, et par la distribution initiale $\pi_0 = (Prob(X_0 = 1), ..., Prob(X_0 = n))$

En appliquant la formule des probabilités totales on obtient :

$$\begin{aligned}(\pi_k + 1)_j = Prob(X_{k+1} = j) &= \sum_{i=1}^{n} Prob(X_{k+1} = j | X_k = i) Prob(X_k = i) \\ &= \sum_{i=1}^{n} p_{ij} (\pi_k)_i\end{aligned}$$

Ainsi, la distribution du processus de Markov à l'instant k définie par le vecteur

$$\pi_k = (Prob(X_k = 1), ..., Prob(X_k = n))$$

vérifie $\pi_{k+1} = \pi_k P$.

Si on se ramène à l'algorithme page rank, grâce à la formule ci-dessus on peut déterminer facilement la matrice pi_k à un instant k. Cette matrice indique les différentes probabilités de se retrouver sur chacune des pages du site après avoir visité $k-1$ pages.

L'objectif de l'algorithme page rank est de déterminer la distribution $\hat{\pi}$ stationnaire de la matrice pi, ce qui correspond à $\lim_{k\to\infty} \pi_k$. Cette distribution définit alors le classement de l'algorithme page rank une fois les probabilités organisées dans l'ordre décroissant. Puisque par définition $\pi_{k+1} = \pi_k P$, cette distribution vérifie nécessairement $\hat{\pi} P = \hat{\pi}$

Théorème Soit P une matrice stochastique irréductible. Alors P admet$\lambda = 1$ pour valeur propre dominante unique et il existe un vecteur propre à gauche associé $\hat{\pi}$ vérifiant

$$\hat{\pi} \geq 0, i = 1...n \text{ et } \sum_{j=1}^{n} \hat{\pi} = 1 \tag{5.5}$$

Ainsi, π converge et possède donc toujours une distribution stationnaire. En pratique, l'algorithme ne s'y prend pas vraiment de cette manière car les matrices manipulées sont gigantesques puisqu'elle correspondent à des bases de données.

Pour appliquer l'algorithme page rank sur les pages web du site internet de l'UTC, nous allons utiliser des matrices creuses : on ne stocke en mémoire que les élements non nuls.

En TD, nous avons vu que l'on peut écrire P sous la forme :

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

avec

- D une matrice diagonale définie par $d_{ii} = (\Sigma_{j=1}^{n} c_{ij})^{-1}$ si $\Sigma_{j=1}^{n} c_{ij}) \neq 0$ et $d_{ii} = 0$ sinon
- le vecteur e est défini par $e = (1, ..., 1)^T$
- le vecteur f est défini par $f_i = 0$ si $\Sigma_{j=1}^{n} c_{ij}) = 0$ et $f_i = 1$ sinon

Ce qui donne sur scilab :

```
d = full(sum(C,2))';
d(d>0=1./d(d>0)
e=ones(n,1)
f=ones(n,1)
f(d==0)=0;
```

On remarque que même si C est creuse, P ne l'est pas et il n'est donc pas judicieux d'utiliser P directement dans l'algorithme de la puissance.

Pour éviter d'utiliser P dans l'algorithme de la puissance il suffit de remarquer que pour un vecteur ligneπ donné on a :

$$\pi P = q\pi DC + \frac{1}{n}\pi(e - qf)e^T$$

D'une part le produitπDC revient a multiplier les composantes de π par les éléments diagonaux de D puis de multiplier le vecteur ligne obtenu par C.

D'autre part le deuxième terme est égal au produit scalaire $\frac{1}{n}\pi(e - qf)$ multiplie ensuite par le vecteur ligne e^T. Il suffira donc d'ajouter ce nombre aux composantes du vecteur colonne obtenu précédemment.

Ainsi,nous disposons de tous les éléments pour appliquer l'algorithme page rank sur scilab avec les données du site de l'utc. Le code permettant d'appliquer la méthode de la puissance est donné ci-dessous.

```
pi = rand(1,n);
pi = pi/sum(pi);
q=0.85;
for i = 1:1000
    pi_new=q*(pi.*d)*C+pi*(e-q*f)/n;
    if norm(pi-pi_new)<1e-10
        break
    end
    pi=pi_new;
end
```

En utilisant la macro gsort de scilab on peut classer les différentes probabilités de $\hat{\pi}$ et en déduire l'ordre d'apparition des différentes pages du site de l'utc. On obtient le classement ci-dessous pour les 20 premières pages web.

```
--> U(k(1:20))
 ans  =

  "https://interactions.utc.fr"
  "https://www.utc.fr"
  "https://interactions.utc.fr/web-tv-utc.html"
  "https://interactions.utc.fr/diplomes.html"
  "https://interactions.utc.fr/thematiques/prix-et-concours.html"
  "https://interactions.utc.fr/thematiques/vie-de-luniversite.html"
  "https://interactions.utc.fr/thematiques/regards-sur-le-monde.html"
  "https://interactions.utc.fr/thematiques/mecanique-acoustique-materiaux-electromecanique.html"
  "https://interactions.utc.fr/thematiques/entrepreneuriat-startups.html"
  "https://interactions.utc.fr/thematiques/automatique-robotique-decision-informatique-realite-virtuelle.html"
  "https://interactions.utc.fr/thematiques/genie-des-procedes-chimie-developpement-durable-agroressources.html"
  "https://interactions.utc.fr/thematiques/pluridisciplinarite.html"
  "https://interactions.utc.fr/thematiques/international.html"
  "https://interactions.utc.fr/thematiques/technologie-et-sciences-de-lhomme.html"
  "https://interactions.utc.fr/thematiques/bio-mecanique-bio-ingenierie-ingenierie-de-la-sante.html"
  "https://interactions.utc.fr/thematiques/biotechnologies-biocatalyseurs-biomimetisme.html"
  "https://interactions.utc.fr/thematiques/design-industriel.html"
  "https://interactions.utc.fr/contacter-la-redaction.html"
  "https://interactions.utc.fr/thematiques/mathematiques-appliquees.html"
  "https://interactions.utc.fr/newsletter.html"
```

FIGURE 5.5 – classement des 20 premières pages web

Grâce à la fonction search mise en annexe, on peut appliquer l'algorithme page rank avec des mots clés par exemple avec le mot branche, on obtient le résultat suivant :

```
--> search(U,pi,"branche")
Search result for "branche":

    Rank | URL
---------|--------------------------
0.000099 | https://www.utc.fr/utc/actualites-de-lutc/agenda/soiree-choisis-ta-branche.html
0.000099 | https://www.utc.fr/utc/actualites-de-lutc/agenda/soiree-tremplin-choisis-ta-branche.html
0.000099 | https://www.utc.fr/utc/actualites-de-lutc/agenda/4e-soiree-choisis-ta-branche.html
0.000098 | https://www.utc.fr/en/utc/news-utc/agenda/soiree-choisis-ta-branche.html
0.000098 | https://www.utc.fr/en/utc/news-utc/agenda/soiree-tremplin-choisis-ta-branche.html
0.000098 | https://www.utc.fr/en/utc/news-utc/agenda/choisis-ta-branche.html
```

FIGURE 5.6 – classement des 20 premières pages web

Chapitre 6

Séries de Fourier

6.1 Propriétés

Série de Fourier
On appelle polynôme trigonométrique ou série de Fourier une fonction periodique de periode T definie par

$$P_n(x) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos(k\omega x) + b_k \sin(k\omega x), \omega = \frac{2\pi}{T}$$

Soit (f_n) une suite de fonctions $f_n : E \to \mathbb{R}$ définies sur un ouvert $E \subset \mathbb{R}$.

convergence simple
On dit que (fn) converge simplement vers f si

$$\forall \epsilon > 0, \forall x \in E, \exists N \in \mathbb{N}, \forall n \in \mathbb{N}, n > N \Rightarrow |f_n(x) - f(x)| < \epsilon$$

convergence uniforme
On dit que (fn) converge uniformément vers f si

$$\forall \epsilon > 0, \exists N \in \mathbb{N}, \forall x \in E, \forall n \in \mathbb{N}, n > N \Rightarrow |f_n(x) - f(x)| < \epsilon$$

Théorème de Dirichlet

Soit f une fonction périodique de période T et telle que :

— les discontinuités de f sur [0, T[sont en nombre fini et de première espèce,

— en tout point f admet une dérivée à gauche et à droite.

Alors la série

$$S_n = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos(k\omega x) + b_k \sin(k\omega x)$$

,avec

$$\omega = \frac{2\pi}{T},$$

$$a_k = \frac{2}{T}\int_0^T f(x)cos(k\omega x)dx,$$

$$b_k = \frac{2}{T}\int_0^T f(x)sin(k\omega x)dx,$$

converge vers une limite S définie par $S(x) = \begin{cases} f(x) \text{si f est continue en x} \\ \frac{f(x^+)+f(x^-)}{2} \text{sinon} \end{cases}$ La convergence est uniforme sur tout intervalle où f est continue.

Conclusion

Pour conclure, j'ai trouvé MT94 très intéressante.
L'UV m'a permis de mettre en application certains concepts mathématiques parfois très abstraits que j'ai pu étudier durant mon cursus.

Je regrette seulement de n'avoir pas eu assez de temps pour approfondir encore plus les différentes notions vues durant cette UV notamment en réalisant des recherches personnelles et en lisant les livres que vous nous avez suggéré notamment sur la résolution des équations différentielles ou sur les nombres duals(mais ce n'est que partie remise !).

En tout les cas, je trouve que l'UV est parmi les plus intéressante pour le tronc commun et me servira grandement dans mes futures études en mécanique où les logiciels de type scilab ou mathlab sont indispensables.

Annexes

chapitre1

comparaison erreur d'aproximation des dérivées

```
function y=f(x)
    y=exp(x)./(cos(x)^3 + sin(x)^3);
endfunction

function d=fprime(x)
    d=exp(x)*(cos(x)^3 + 3*cos(x)^2*sin(x) - 3*cos(x)*sin(x)^
2 +..
     sin(x)^3)./(cos(x)^3 + sin(x)^3)^2;
endfunction

h=2^(0:-1:-70)
D_1 = (f(%pi/4+h)-f(%pi/4))./h;
D = fprime(%pi/4);
E_1=abs((D_1-D)/D);
D_2 = (f(%pi/4+h)-f(%pi/4-h))./h/2;
E_2=abs((D_2-D)/D);
D_3=imag(f(%pi/4+%i*h))./h
E_3=abs((D_3-D)/D);


plot(h,E_1,h,E_2,h,E_3,"thickness",2);
gca().log_flags="ll";

xlabel("$\LARGE␣h$");
legend("$\large|D_1(h)-f''(x_0)|/|f''(x_0)|$",...
"$\large|D_2(h)-f''(x_0)|/|f''(x_0)|$",-1)
title("$\LARGE\mbox{Erreur␣relative␣}␣D_1(h)$")
isoview on
```

chapitre2
cinématique inverse

```
function y = f(theta,A)
    M = [l*cos(theta(1:2))*cos(theta(3))
       l*cos(theta(1:2))*sin(theta(3))
       l*sin(theta(1:2))];
  y = M-A;
endfunction

l = [1, 2/3];
A = [1.1; 0.6;.5];
B = [1.1; 1.1;.5];
C = [0.1; 1.1;.5];
D = [0.1; 0.6;.5];
E=[1.1;0.6;-0.5];
F=[1.1; 1.1;-0.5];
G=[0.1; 1.1;-0.5];
H=[0.1; 0.6;-0.5];

coords = [linspace(A,B,50)..
linspace(B,C,50)..
linspace(C,D,50)..
linspace(D,A,50)..
linspace(A,E,50)..
linspace(E,F,50)..
linspace(F,B,50)..
linspace(F,G,50)..
linspace(G,C,50)..
linspace(G,H,50)..
linspace(H,D,50)..
linspace(H,E,50)];

theta=[0.1;0.1;0.1];

for X = coords
[theta,v,info] = fsolve(theta,list(f,X));
visuArm(theta,l,%t)
end
```

chapitre4
ellipse

```
load C:\Users\merinos\www_utc_fr.sod
[ij,v] = spget(C);
plot(ij(:,2),ij(:,1),".","markersize",2)
[n,n]=size (C)
d = full(sum(C,2))';
d(d>0) = 1./d(d>0)
e = ones(n,1)
f = ones(n,1)
f(d==0) = 0;
pi = rand(1,n);
pi = pi/sum(pi);
q=0.85;
for i = 1:1000
    pi_new=q*(pi.*d)*C+pi*(e-q*f)/n;
    if norm(pi-pi_new)<1e-10
        break
    end
    pi=pi_new;
end
[pis,k]=gsort (pi);
```

chapitre 5
recherche

```
// Complements Algorithme PageRank pour MT94
// S. Mottelet, UTC
// Jeu 16 mai 2020 13:17:00 CEST
function search(U,x,term) // fonction de recherche par mot-cl
és contenus dans les URL
// U : matrice des URL
// x : PageRank
// term : mot-cle recherche
ind=[]
for i=1:size(U,1)
  if ~isempty(strindex(U(i),term))
    ind=[ind i]
  end
end
if ~isempty(ind)
  xf=x(ind)
  Uf=U(ind)
  [xf,k]=gsort(xf)
  Uf=Uf(k)
  printf("Search␣result␣for␣""%s"":\n\n",term)
  printf("%8s␣|␣%-40s\n","Rank","URL")
  printf("---------|-----------------------------\n")
  for i=1:min(10,length(ind))
      printf("%8.6f␣|␣%s\n",xf(i),Uf(i))
  end
  printf("\n")
end
```

affichage des 20 premiers sites

```
load C:\Users\merinos\www_utc_fr.sod
[ij,v] = spget(C);
plot(ij(:,2),ij(:,1),".","markersize",2)
[n,n]=size (C)
d = full(sum(C,2))';
d(d>0) = 1./d(d>0)
e = ones(n,1)
f = ones(n,1)
f(d==0) = 0;
pi = rand(1,n);
pi = pi/sum(pi);
q=0.85;
for i = 1:1000
    pi_new=q*(pi.*d)*C+pi*(e-q*f)/n;
    if norm(pi-pi_new)<1e-10
        break
    end
    pi=pi_new;
end
[pis,k]=gsort (pi);
// pour afficher les 20 premiers sites
disp(U(k(1:20))
```