Université de Chahid Mostfa Ben boulaid; BATNA 2
Faculté De Technologie, Département D'Hydraulique

Semestre : 2
Unité d'enseignement : UEM 1.2
Matière : Modélisation et simulation en hydraulique Responsable : KERKOURI Ali
VHS: 37h30 (TP: 2h30)
Crédits : 3
Coefficient : 2

Objectifs de l'enseignement:

Permettre à l'étudiant la résolution numérique des équations mathématiques régissant les problèmes hydrauliques et des problèmes fondamentaux d'ordre pratique par la réalisation de programmes simplifiés sur Matlab (ou autres environnements) et de simuler des cas réels (complexes) sur des logiciels appropriés.

Connaissances préalables recommandées

Bonne connaisse des bases de la mécanique des fluides, des écoulements sous pression, des écoulements à surface libre et des méthodes numériques et langages de programmation informatique.

Contenu de la matière :

Chapitre 1: **Rappel (Méthodes de résolution des équations non-linéaires et du système d'équations) (1 Semaine)**

- Méthode de Dichotomie (Bissection), Méthode de la sécante, Méthode de Regula Falsi (Fausse Position), Méthode de Newton Raphson, Méthode du point fixe
- Méthodes utilisées pour résoudre les systèmes d'équations (Méthodes directes et indirectes)

Chapitre 2 Modélisation par la méthode des différences finies (M.D.F.) des écoulements (2 Semaines)

- Discrétisation des opérateurs différentiels
- Introduction des conditions aux limites et initiales
- Différences finies du premier ordre
- Différences finies de second ordre
- Schémas de discrétisations temporelles (explicites, implicites et mixte)
- Convergence, stabilité et précision des schémas numériques.
- Exemple d'application de modélisation d'un écoulement permanent uniforme par M.D.F.

Chapitre 3 Modélisation par la méthode des éléments finis des écoulements (M.E.F) (2 Semaines)

- Maillage et éléments
- Méthodes de minimisation de l'erreur (résidus pondérés, Galerkin…)
- Approximations nodales
- Eléments de référence
- Méthodes intégrales faibles
- Calcul sur les éléments

Intégration numérique

- Exemple d'application de modélisation d'un écoulement permanent unifo

M.E.F

Chapitre 4 : **Initiation à la méthode des volumes finis(1 semaine)**

Introduction, Méthodes de discrétisation, Equation de la chaleur conduction, confusion, Exemple d'application

Chapitre 5 **Modélisation et Simulation des écoulements** (**9 semaines**)

- Modélisation d'un écoulement à surface libre non permanent dans un canal prismatique 1D
- Modélisation d'un écoulement en charge transitoire dans une conduite 1D
- Calcul des courbes de remous (utilisation de logiciels)
- Vidange d'un réservoir (barrage) vers l'atmosphère
- Ecoulement entre deux réservoirs (barrages)
- Simulation des écoulements dans les réseaux d'AEP, d'assainissement, irrig drainage
- Autres simulations…etc.

Mode d'évaluation :

Contrôle continu : 100%. :

Chapitre I :Rappel (Méthodes de résolution des équations non-linéaires et du système d'équations)

Ce document, destiné aux étudiants de MASTER 1 (HU,OH,RH) S 2

Contenu du chapitre :

1-Méthodes de résolution des équations non-linéaires

1.1.Méthode de Dichotomie (Bissection),
1.1-1.L'objectif
1.1.2. Le principe
1.1.3. L'algorithme de Dichotomie
1.1.4 TP
1.2., Méthode de Newton Raphson,
1.2.1. Le principe
1.2.2. L'algorithme de Dichotomie
1.2.3 TP
1.3 Méthode de Regula Falsi (Fausse Position),
1.3.1. Le principe
1.3.2 Le script MATLAB de la méthode Regula Falsi
1.4 Méthode de la sécante
1.3.1. Le principe
1.3.2 Le script MATLAB de la méthode
1.5.Méthode du point fixe
1.5.1. Le principe
1.5.2Illustrations graphiques

2Méthodes utilisées pour résoudre les systèmes d'équations (Méthodes directes et indirectes)

2.1 Méthodes directes
2.1.1.Méthode de Cramer
2.1.2.Méthode de la matrice inverse
2.1.3.Méthode de Gauss (pivot)
2.1.4.Méthode de Cholesky
2.1.5.Méthode de factorisation LU
2.2. Méthodes indirectes
2.2.1.Méthode de Jacobi
2.2.2.Méthode de Gauss-Seidel
2.2.3Utilisation de la relaxation

Chapitre I:

Méthodes de résolution des équations non-linéaires et du système d'équations

1-Méthodes de résolution des équations non-linéaires

1.1-Méthode de Dichotomie(Bissection):

1.1-1.L'objectif : recherche d'un point unique d'abscisse x dans une intervalle qui satisfait f(x)=0

1.1.2. Le principe:

Cette méthode repose sur les valeurs intermédiaires C'est à dire pour toute fonction monotone sur l'intervalle [a, b];

Si f(a)*f(b)<0 on fait:

Calculer M=f((a+b)/2)

Si M est de même signe que f(a); alors la racine se trouve dans l'intervalle $[\frac{a+b}{2}, b]$; si non elle se trouve dans l'intervalle$[a, \frac{a+b}{2}]$

L'étude de convergence est le critère d'arrêt

Si le processus de dichotomie arrive jusqu'à l'étape n alors on a l'estimation :

$|\alpha_{solution} - ((a+b)/2)^n| \leq \frac{b-a}{2^{n+1}}$ Avec la suite $C_n = ((a+b)/2)^n$

Par conséquent, la suite $C_n$converge vers $\alpha_{solution}$

C'est aussi vrai si $C_n = \alpha_{solution}$

1.1.3. L'algorithme de Dichotomie:

Algorithme 1: Méthode de dichotomie

Paramètre d'entrées: f,a,b

F fonction continue sur l'intervalle I

Deux réels a et b tel que f(a).f(b)<0

Paramètres de sorties: x, n

X approximation du zéro de la fonction f

N nombre d'itérations nécessaires pour atteindre x

$m \leftarrow \frac{a+b}{2}$

Tant que (m<0) or (abs(b-a)>epsilon) faire

Si f(a)*f(m)>0 alors

$a \leftarrow m$

Si non

$b \leftarrow m.$

Fin si

$m \leftarrow \frac{a+b}{2}$

Fin tant que

Afficher la racine de l'équation f est r=; m

Fin Méthode de dichotomie

1.1.4 TP :

Dans ce TP, il est demandé de trouver la racine de $F(x) = 0$;

$$F(x) = x + \exp(x) + \frac{10}{1 + x^2} - 5$$

En utilisant la méthode de dichotomie

1-écrire un programme MATLAB permettant l'implémentation de l'algorithme ou le principe numérique de cette méthode.

2-afficher les résultats

3-calculer l'ordre et la vitesse de convergence de la méthode numérique sachant que la tolérance $=10^{-3}$ a=-1.30 et b=1.5

Script MATLAB

```
Clear all; close all; clc
a=-1.30: b=1.50; itmax=100; tol=10^-6;
it=0; m=(b+a)/2; x=[a, m, b];
fun=x+exp(x)+10./(1+x.^2)-5;
while it<itmax
if fun(a)*fun(m)<0
b=x(2); a=x(1); m=(b+a)/2; x=[a, m, b];
elseif fun(m)*fun(b)<0
b=x(3); a=x(2); m=(b+a)/2; x=[a, m, b];
end

if abs(fun(m))<tol
sol=m;
break
end
if==itmax  & abs(fun(m))> tol
disp(' PAS DE CONVERGENCE POUR LA TOLERANCE CHOISIE')
end
it=it+1
xit(it)=m
itNumber=it;
end
disp(strcat( ' la racine approche vaut :',num2str(sol)))

%%%%%%%%%%%%%%%% Affichage graphique %%%%%%%%%%%%%%%%
ainit=1.3; binit=3/2; n=500; dx=(binit-ainit)/n;
xn=ainit: dx:binit;
figure('color', [1 1 1]; plot(xn;fun(xn),'line Width',1); hold on
plot(sol,fun(sol),'ro','MarkerSize',12,'lineWidth',2)
plot(sol,fun(sol),'rx','MarkerSize',12,'lineWidth',2)
```

```
plot(xit,fun(xit),'kx','MarkerSize',10,'lineWidth',1.5)
plot(xit,fun(xit),'k0','MarkerSize',10,'lineWidth',1.5)
line(xn, zero(1,length(xn)),'LineStyle','-.','Color','k';…
''LineWidth',1); xlabel('f(x)')
title('RACINE DE F(x)= PAR LA METHODE DE DICHOTOMIE')
```

Résultat:
Solution de l'équation non linéaire
Les sorties renvoyées par ce script MATLAB sont:
La racine approche vaut:-0.92045, itNumber=21

1.2 .Méthode de Newton Raphson

1.2.1 Le principe:

Lorsque la fonction f est différentiable, on peut établir la méthode plus efficace en utilisant la méthode de Newton Raphson qui permet d'approcher par itérations la valeur x au moyen de la relation suivante:

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

Le principe de la méthode est de trouver la valeur de x qui annule la fonction f(x).

Si $|\mathrm{x_n} - \mathrm{x_{n-1}}| < \varepsilon$ alors $\mathrm{x_n}$ est le résultat de l'estimation de la racine.

Où: ε représente l'erreur d'approximation caractérisant la qualité de la solution numérique.

Dans toutes les méthodes itératives, il est nécessaire, pour éviter une divergence de la solution de bien choisir la valeur initiale. Celle-ci peut être obtenue graphiquement.

1.2.2 Algorithme de la méthode de Newton_Raphson

Debut Newton_Raphson
Paramètres d'entrées: x0, Epsilon, N
Paramètres de sorties : x
x valeur approchée où un message d'échec
Trouve← false
$N \leftarrow 1$
Tant que N ≤ N0 and trouve=false faire

$$x \leftarrow x0 - \frac{\mathrm{f(x_0)}}{\mathrm{f'(x_0)}}$$

Si $|x - x0| > Epsilon\ alors$
N ← N + 1
X0=x
Sinon
Trouve =true
Imprimer 'approximation est', Ix
Fin Si
FinTant que

Si trouve= false alors
    Imprimer ‘la méthode échoué après N itérations
Fin Si
Fin Newton_Raphson

1.2.4 TP :

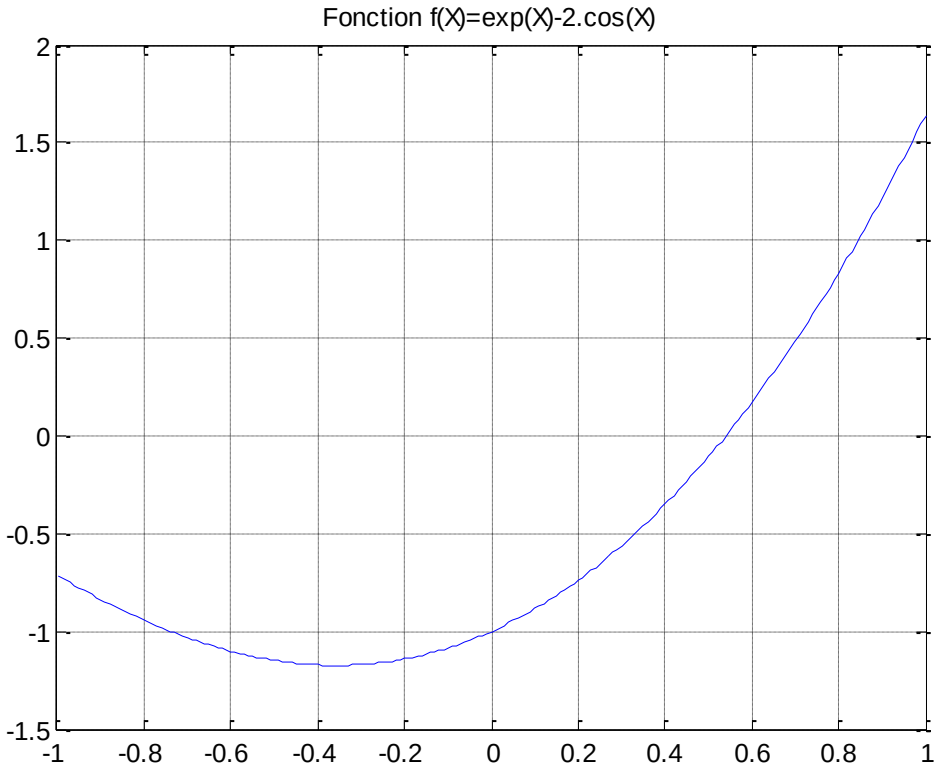
Il est demandé d’appliquer la méthode de Newton_Raphson pour la recherche des racines de la fonction non linéaire d’expression:

$$f(x) = e^x - 2.cosx$$

Les phases à suivre:

1) On se propose de tracer la courbe représentative de cette fonction par MATLAB

```
X=-1:0.01:1;
f=exp(x)-2*cos(x);
plot(X,f), grid
Title(‘fonction f(x)= exp(x)-2cos(x)’)
```



Au vu la courbe de f , il est judicieux de choisir X0=0.5 . la fonction dérivée f’ a pour expression :

$$f'(x) = e^x + 2.sinx$$

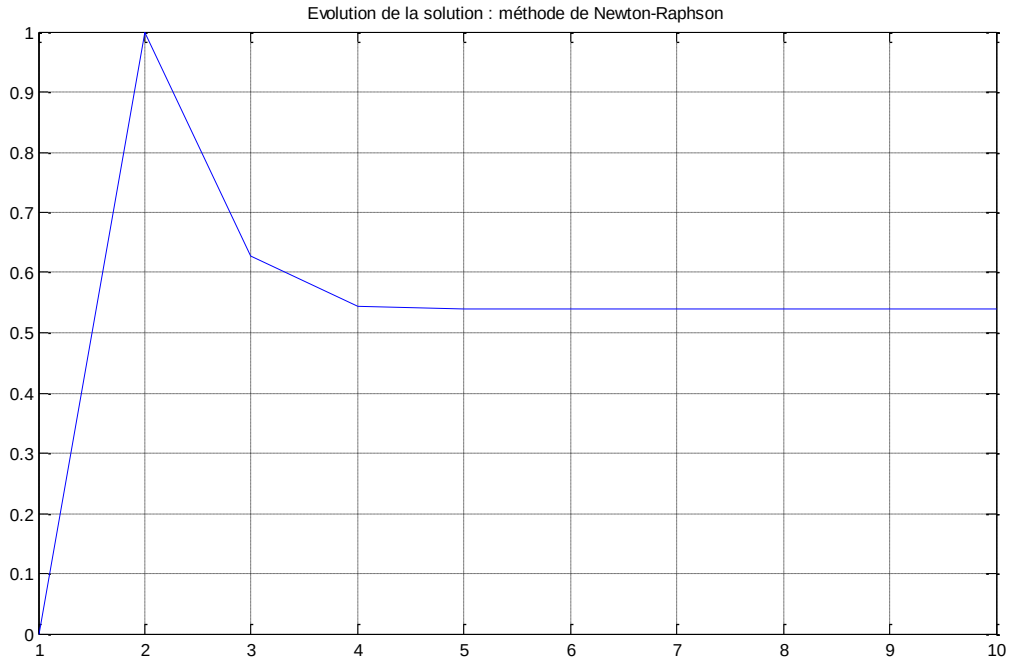
2) Pour chercher la solution de l’équation on rajoute le programme suivant:

```
clear X
X(1)=0;
for i=2:10
```

```
    f=exp(X(i-1))-2*cos(X(i-1));
    diff=exp(X(i-1))+2*sin(X(i-1));
    X(i)=X(i-1)-f/diff;
end
plot(X),grid, hold on

Title('Evolution de la solution : méthode de Newton-Raphson ')
disp('les valeurs successives sont: :')
```

X= 0 1 0,627904125793578 0,544206631445441 0,539797325698085 0,539785160901820 0,539785160809281 0,539785160809281 0,539785160809281 0,539785160809281



On remarque qu'il ya convergence au bout de 4 itérations. La solution peut etre obtenue directement par la fonction fzero. Il faut créer le fichier M dans lequel sera programmée f(x).

fichier f.m

```
function f=f(X)
% fichier M représentant la fonction
f=exp(X)-2*cos(X)
```

pour obtenir la solution de l'équation f(X)=0, au voisinage de X=0.5, on applique fzero(f, 0.5). Cette fonction affiche les valeurs obtenues à chaque itération pour enfin donner la solution.

```
>>d= fzero('f', 0.5)
```

La fonction find retourne les indices du tableau f qui satisfont à une condition particulière. Dans l'exemple suivant, on cherche ceux pour lesquels f(X) s'annule

```
>> j=find(f == 0)
J=[]
```

```
>> solutions = x(j)
```

```
solutions=[]
```

Au lieu d'indiquer f==0 comme codition pour la commande find, il faut spécifier la condition abs(f)<epsilon où epsilon est un réel très petit, fixé a priori.
En effet avec les approximations des calculs, il est difficile d'aboutir à la racine exacte.

```
>> epsilon = 0.01;
>> j=find(abs(f)<epsilon)
```

Les solutions approchées sont alors :

```
>> solutions = X(j)
Solution = 0.5398
```

1.3. Méthode de Regula Falsi (Fausse Position)

1.3.1 Le principe de la méthode:

On peut substituer à la méthode de dichotomie une méthode plus efficace qui en découle directement. L'idée est de tenir compte des valeurs de la fonction aux extrémités de l'intervalle de travail et non seulement des signes qu'elle y prend.

Supposons que les conditions d'application de la méthode de dichotomie soient remplies.

Au lieu de diviser l'intervalle de travail en deux parties égales, on procède différemment. Sur l'intervalle en question on assimile le graphe de f(x) à la droite qui joint les points (a,f(a)) et (b, f(b)). Une approximation de la racine cherchée est donc l'abscisse du point ou cette droite coupe l'axe des x. on prend pour w l'abscisse de ce point et il est facile de montrer que:

$$w = b - f(b)\frac{b-a}{f(b)-f(a)} = a - f(a)\frac{b-a}{f(b)-f(a)}$$

La nouvelle intervalle de travail sera soit [a, w] soit [w, b]selon les signes pris par f(a), f(b), f(w).

1.3.2 Le script MATLAB de la méthode Regula Falsi

```
Function [w,err;yw]=regula(f,a,b,delta,epsilon,maxi)
% In put –f is the function in put as a string 'f'
% a and b are the left end reight end points
% delta is the tolerance for the zero
% epsilon is the tolerance for the value of f at the zero
% max1 is the maximum number of iterations
% Out put-wis the zero
% yw=f(w)
% -err is the error estimate for w
ya=feval(f,a); yb=feval(f,b);
if ya*yb>0
disp('note: f(a)*f(b)>0'), break,
end
for k=1:max1
dx=yb*(b-a)/(yb-ya);
w=b-dx; aw=w-a;
```

```
yw=feval(f,w);
if yw==0, break;
elseif yb*yw>0
b=w;  yb=yw;
else
a=w; ya=yw
end
c;
err=abs(b-a)/2
yc=feval(f,c)
```

1.3. Méthode de la sécante

1.3.1 Le principe de la méthode:

La méthode de la sécante est apparentée à la méthode Regula Falsi. Ici aussi on assimile le graphe de f(x) à une droite. Toutefois, on se libère des contraintes des deux méthodes précédentes en n'exigeant plus la détermination d'un intervalle qui encadre la racine cherchée et sur lequel f(x) change de signe.

Autrement dit la méthode de de Newton Raphson suppose le calcul de f'(p). On remplace dans la méthode de Newton Raphson f'(pn) par:

$$\frac{f(p_n) - f(p_{n-1})}{p_n - p_{n-1}}$$

L'équation de la sécante s'écrit:

$$x(n) = f(p_n) + (x - p_n)\frac{f(p_n)-f(p_{n-1})}{p_n-p_{n-1}}$$

Si $f(p_{n+1}) = 0$, on en déduit

$$p_{n+1} = p_n - f(p_n)\frac{p_n - p_{n-1}}{f(p_n) - f(p_{n-1})}$$

1.3.3 Algorithme de la méthode de la sécante

Debut Méthode de la sécante

Paramètres d'entrées: p0, p1: deux approximations initiales
Epsilon : la précision désirée
NO: le nombre maximum d'itérations

Paramètres de sorties : x
p valeur approchée où un message d'échec

(1) poser N=1, q0=f(p0), q1=f(p1)

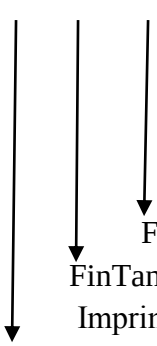
Tant que $N\_N0 + 1$, faire les étapes 3 à 6

Poser $p = p1 - q1\frac{p1-p0}{q1-q0}$

Si $|p - p1| < epsilon\ alors$

Imprimer p, aller ver fin

Sinon



N ← N + 1
p0 ← p1
q0 ← q1
p1=p
q1=f(p)
Fin Si
FinTant que
Imprimer 'la méthode a échoué après NO itérations
Fin Méthode de la sécante

1.3.3 Le script MATLAB de la méthode de la sécante

TP : développer un programme MATLAB pour résoudre l'équation: **x-0.2 sin(x)-0.5=0** par la méthode de la sécante:

```
Function [x, fx,xx] = secant (f,x0,Tolx, MaxIter, varargin)
% solve f(x)=0 by using the secant method.
% input: f= ftn to be given as a string 'f' if defined in an M-file
% x0 = the initial quess of the solution
%Tolx = the upper limit of |x(k) − x(k − 1)|
% MaxIter = the maximum # of iterations
% out put : x = the point which the algorithm has reached
% fx = f(x(last)), xx = the history of x
h= 1e-4; h2= 2*h; TolFun= eps;
xx(1) = x0; f(x)= feval(f,x0, varargin{:});
for k=1: MaxIter
if k <= 1, dfdx= (feval(f,xx(k) +h, varargin{:})- feval(f,xx(k) -h, varargin{:}))/2;
else dfdx= (fx-fx0)/dx;
end
dx=-fx/dfdx;
xx(k+1) = xx(k)+ dx;
fx0=fx;
fx=feval(f;xx(k+1));
if abs(fx)< TolFun ı abs(dx)< Tolx, break; end
end
x= xx(k+1);
if k== MaxIter, fprint(' The best in % d iterations\n', MaxIter), end
```

1.4 .Méthode du point fixe

1.4.1 Le principe:

Le principe de la méthode du point fixe consiste à transformer, la fonction f(x)=0, f:[a, b]→R, en une fonction g(x)=x. La fonction g: :[a, b]→R, est construite de facon que g(α)= α quand f(α)=0. Trouver la racine de f(x), se résume donc à déterminer un α ∈:[a, b] tel que g(α)= α.
Dans le cas ou un tel point existe, il sera qualifié de point fixe de g et cette dernière est dite fonction d'itération. Le schéma numérique de cette méthode est donnée par:
x(k+1)= g(x(k)) pour k ≥ 0

1.4.2. Illustrations graphiques

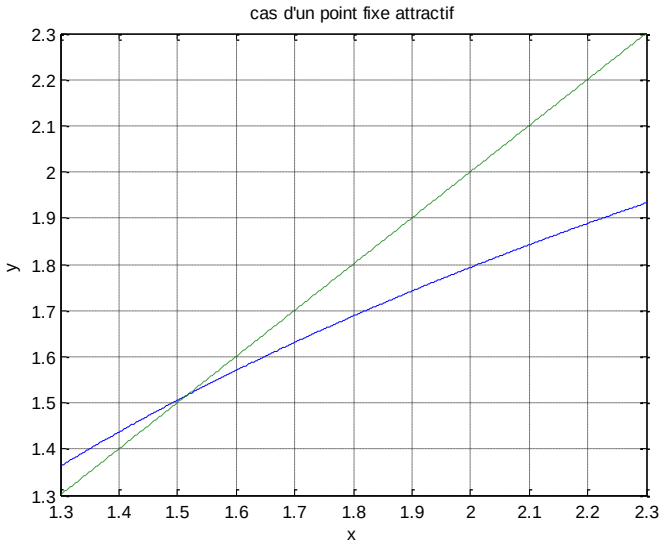
a)point attractif

Exemple

Illustration graphique

Code MATLAB.

```
>> x=1.3:0.001:2.3;
>> plot(x,log(x)+1.1, '-', x, x,'--')
>> grid on;
```



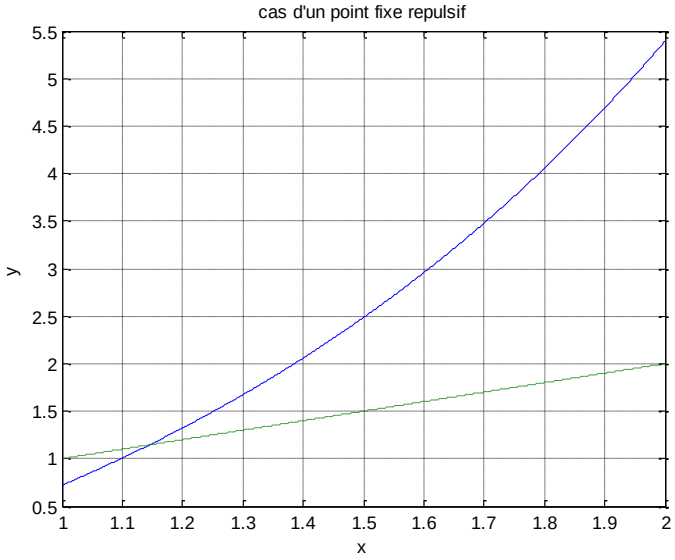
b) point répulsif

Exemple

Illustration graphique

Code MATLAB

```
x=1:0.0001:2;
>> plot(x, exp(x)-2, '-', x, x, '--')
>> grid on;
>> ylabel('y');
>> xlabel('x');
>> title('Cas d'un point fixe répulsif')
```



c) Point douteux:

Exemple : Soit la fonction g définie par g(x)=sinx, x∈:[0, π/2]. On a ∀ x∈:[0, π/2]., sinx< x et pour tout x0 x∈:[0, π/2]., la suite itérée (xn) définie par xn+1=g(xn) et strictement décroissante minorée par 0 donc concergent vers une limite α . comme g est conyinue et que α=g(α), α=0 est l'unique point fixe de g sur ∈:[0, π/2]

Illustration graphique

Code MATLAB

```
x= -pi/2:0.0001:pi/2 ;
g= inline('sin(x)');
plot(x,g(x), '—', x, x, '—')
grid on;
ylabel('g(x)');
xlabel('x');
axis on;
title (graphe de g');
```

1.2. Méthodes de résolution des systèmes d'équations

1.2.1. Méthodes directes:

Une méthode de résolution d'un système d'équation est dite directe si on ibtient les valeurs finis après un nombre d'opérations ; les méthodes connues sont la méthode de Cramer ; méthode de matrice inverce, méthode de Gauss , méthode de cholesky et la méthode de factorisation LU

a) **Méthode de Cramer:** soit l'équation :

$$\begin{pmatrix} a_{11} & a_{12} \cdots & a_{1n} \\ \vdots & \ldots\ldots & \vdots \\ a_{n1} & a_{n2} \cdots & a_{nn} \end{pmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \ddots \\ \ddots \\ \ddots \\ x_n \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ \ddots \\ \ddots \\ \ddots \\ b_n \end{Bmatrix} \text{ ou AX=B}$$

La méthode de Cramer est basé sur le calcul du déterminant de la matrice A et les déterminants associés aux inconnues x_i (i=1……n)

$$x_i = \frac{Det_i}{Det\ A}$$

Det A: Déterminant de la matrice A

Det_i: Déterminant associé à l'inconnue x_i

b) Méthode de la matrice inverse:

Soit le système d'équations: AX=B; multiplions les deux termes par A^{-1}(l'inverse de A):

$X=A^{-1}B$

$$A^{-1} = \frac{1}{Det\ A} C^T$$

C^T: Cofacteur total sachant que $C_{ij} = (-1)^{i+j}(cofacteur)_{ij}$

c) **Méthode de Gauss (pivot)**:

Soit le système d'équations AX=B , la méthode de Gauss est basé sur l'idée de trianguler la matrice A c.-à-d.

$\begin{pmatrix} a_{11} & a_{12} \cdots & a_{1n} \\ \vdots & \ldots\ldots & \vdots \\ a_{n1} & a_{n2} \cdots & a_{nn} \end{pmatrix}$ se transforme en $\begin{pmatrix} A_{11} & A_{12} \cdots & A_{1n} \\ 0 & A_{22} & A_{2n} \\ 0 & 0 \cdots & A_{nn} \end{pmatrix}$ et de transformer le vecteur: $\begin{Bmatrix} b_1 \\ b_2 \\ \ddots \\ \ddots \\ \ddots \\ b_n \end{Bmatrix}$

en $\begin{Bmatrix} B_1 \\ B_2 \\ \ddots \\ \ddots \\ \ddots \\ B_n \end{Bmatrix}$

d) Méthode de Cholesky:

Soit le système d'équations AX=B, avec A une matrice symétrique définie positive, il existe une matrice triangulaire inferieure L tel que: $A=LL^T$; alors

Le système sera remplacé par: $L^TX=Y$ et y se détermine par la résolution de l'équation: LY=B.

e) **Méthode de factorisation LU**:

Théorème:

La factorisation LU de A existe et unique si et seulement si les sous matrices principales de A sont inversibles.

Basé sur ce théorème la méthode de factorisation LU est analogue à celle de Cholesky mais dans ce cas A s'écrit A=LU tel que L est une matrice triangulaire inferieure avec $l_{ii}=1$ et U est une matrice triangulaire supérieure

1.2..2 Méthodes indirectes: (méthodes d'approximations):

Les méthodes approximatives ou itératives pour la résolution des systèmes d'équations linéaires de type Ax=b sont basées sur l'idée de construire une suite de vecteur x(k) qui converge vers une solution exacte en utilisant une fonction linéaire f telle que x(k+1)=f(x(k))

k est un entier naturel

Le système A.x=b peut s'écrire: x=B.x+c, par cette forme réduite le processus itératif ait lieu:

$$x(k+1)=B.x(k)+c$$

Théorème:

Soit le système réduit le processus itératif x(k+1)=B.x(k)+c, le processus converge vers une solution unique si l'une des normes canoniques de la matrice B est inférieure à 1 et la convergence ne dépend pas de la valeur initiale x(0)

Les trois normes canoniques:

1- $\|B\|_m = max_i \sum_j |a_{ij}|$ normes en lignes
2- $\|B\|_m = max_j \sum_i |a_{ij}|$ norme colonnes
3- $\|B\|_k = \sqrt{\sum_{ij} |a_{ij}|^2}$

Pour estimer l'erreur des approximations du processus itératif on utilise les formules suivantes:

$$\|x - x(k)\| \leq \frac{\|B\|}{1 - \|B\|} \|x(k) - x(k-1)\|$$

Ou

$$\|x - x(k)\| \leq \frac{\|B\|^k}{1 - \|B\|} \|x(1) - x(0)\|$$

Remarque:

En pratique si on impose une précision ε on peut estimer l'erreur par:

$$\|x(k) - x(k-1)\| \leq \varepsilon$$

Méthode de Jacobi

Soit le système d'équations suivant:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots \ldots\ldots\ldots\ldots\ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{21}x_2 + \cdots \ldots\ldots\ldots \quad \ldots + a_{21}x_n = b_2 \\ \\ a_{n1}x_1 + a_{n2}x_2 + \cdots \ldots\ldots\ldots\ldots\ldots + a_{nn}x_n = b_n \end{cases}$$

Ce système peut être écrit sous la forme:

$$\begin{cases} x_1 = [b_1 - (a_{12}x_2 + \cdots \ldots\ldots\ldots\ldots\ldots a_{1n}x_n)]/a_{11} \\ x_2 = [b_2 - (a_{21}x_1 + \cdots \ldots\ldots\ldots\ldots\ldots a_{21}x_n)]/a_{22} \\ \\ x_n = [b_n - (a_{n1}x_1 + \cdots \ldots\ldots a_{1(n-1)}x_{n-1})]/a_{nn} \end{cases}$$

Cette forme réduite du système peut s'écrire:

$$x_i = [b_i - \sum_{j=1}^{n} (a_{ji}x_j]/a_{ii}, \qquad i \neq j$$

Le processus itératif aura lieu:

$$x_i(k+1) = [b_i - \sum_{j=1}^{n} (a_{ji}x_j(k)]/a_{ii}\ ;\ i \neq j$$

Méthode de Gauss-Seidel

La méthode de Gauss-Seidel est une amélioration de la méthode de jaccobi en effet elle rend le processus itératif plus rapide.

Le système réduit reste le même sauf que la valeur nouvelle de $x_j(k+1)$ obtenue dans la ième équation sera injecté dans la suivante.

Le processus itératif se fera avec un vecteur initial et l'utilisation de la formule:

$$x_i(k+1) = [b_i - \sum_{j=1}^{i-1} (a_{ij}x_j(k+1) - \sum_{j=i+1}^{n} (a_{ij}x_j(k)]/a_{ii}$$

Utilisation de la relaxation:

Pour accélérer le processus on utilise la relaxation:

La matrice A peut s'écrire:

A=L+D+U

Où: L, D et U sont des matrices respectivement triangulaire inferieure, diagonale et triangulaire supérieure.

En remplaçant dans le système d'équations et en multipliant par ω∈[0; 2]:

$$\omega(L + D + U)x = \omega b \qquad (2)$$

En ajoutant Dx à (2):

$$(D + \omega L)x = \omega b - [\omega U + (\omega - 1)D]x$$

Le processus itératif permet d'écrire:

$$(D + \omega L)x^{(k+1)} = \omega b - [\omega U + (\omega - 1)D]x^k$$

Si $\omega = 1$, on est dans l'application de la méthode de Gauss-Seidel.

On utilise$\omega > 1$, pour accélérer la convergence et $\omega < 1$, pour forcer la convergence d'un processus divergent.