RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITÉ MOHAMED BOUDIAF - M'SILA

FACULTÉ DE TECHNOLOGIE

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

POLYCOPIÉ DU TP : MÉTHODES NUMÉRIQUES

SEMESTRE (4) - 2eme ANNÉE - ST

Méthodes d'approximation numérique pour la résolution des équations non linéaires

Programmation avec MATLAB et C

Créé et modifié par : **Mr. Khatir KHETTAB**

'MCB' au département de : **GÉNIE ÉLECTRIQUE**

Spécialité : **AUTOMATIQUE**

ANNÉE UNIVERSITAIRE : 2017/2018

أعوذ بالله من الشيطان الرجيم :

" وَقُل رَّبِّ زِدْنِي عِلْمًا " الآية 114 من سورة طه

قال رسول الله صلى الله عليه وسلم :

" اللَّهُمَّ انْفَعْنِي بِمَا عَلَّمْتَنِي، وَعَلِّمْنِي مَا يَنْفَعُنِي، وَزِدْنِي عِلْمًا

وارزقني علماً تنفعني به " أخرجه الترمذي في صحيحه.

Avant-propos

Cet ouvrage s'adresse aux étudiants en 2ièm année Sciences et Techniques (ST) en formation du Master. Son objectif est de donner au lecteur un outil lui permettant de travailler de manière autonome à l'aide de questions détaillées et progressives, et d'une construction pas à pas des programmes.

Contrairement à ce qui a été le cas pour les systèmes d'équations linéaires, la résolution des systèmes non-linéaires s'avère beaucoup plus délicate. En général il n'est pas possible de garantir la convergence vers la solution correcte et la complexité des calculs croît très vite en fonction de la dimension du système.

Dans ce cas la solution du système d'équations s'obtient en résolvant une succession de systèmes récursifs et de systèmes interdépendants. Dans la suite on supposera toujours que le système à résoudre est interdépendant.

Parmi les approches des calculs récursifs constituent les techniques les plus souvent utilisées pour la résolution de systèmes d'équations non-linéaires :

- Méthode de Dichotomie (Bissection),
- Méthode du point fixe,
- Méthode de Newton-Raphson,
- Méthode de Lagrange,
- Méthode de Régula-Falsi (fausse position),
- Résolution par minimisation, ...

Ce support est dédié aux étudiants de 2ème année Sciences et Techniques (ST) toutes les options, ce support du TP s'articule autours des objectifs différents comme :

- la résolution des systèmes non linéaires par la programmation de différentes méthodes numériques sous MATLAB,
- La connaissance des autres langages de programmation comme C et le Fortran,
- La conclusion et la comparaison entre ces différentes méthodes.

Ce polycopié, a pour but de présenter un exposé de TP sur la programmation sous MATLAB des méthodes d'approximation et d'optimisation numérique pour résoudre des équations non linéaires. Il est destiné aux ingénieurs, physiciens, mathématiciens ainsi qu'aux étudiants en Troisième année Licence Electronique et Electrotechnique.

Le contenu de ce support de TP se décompose principalement en six chapitres de TP sur les méthodes d'approximation et d'optimisation numérique programmée sous Matlab.

Pour atteindre ces objectifs nous proposons une approche pédagogique pour participer à la formation des concepteurs en génie électrique, en les dotant d'outils méthodologiques, mais plus encore en les mettant dans un état d'esprit propice aux fonction de programmation, caractérisé par la volonté de maîtriser au lieu de subir.

Syllabus

(Plan du TP)

Année : 2^{eme} année Sciences et techniques (ST).

Semestre : 4

UE : Méthodologique.

Matière : TP-Méthodes numériques.

Enseignant responsable de l'UE :

Enseignant responsable de la matière : Khatir KHETTAB

Objectifs de l'enseignement :

- Constituer les banques des connaissances du MATLAB,

-Approfondir les connaissances en programmation sous MATLAB des méthodes d'approximation numériques.

Contenu de la matière :

TP1. Rappels et notions sur les calculs matriciels,

TP2. Méthode de Dichotomie (Bissection),

TP3. Méthode du Point-fixe,

TP4. Méthode de Newton,

TP5. Méthode de Lagrange,

TP6. Méthode de Régula-falsi (fausse position) ...

Mode d'évaluation : L'examen continu ainsi que les travaux pratiques contribueront chacun au quart de la note globale qui englobe :

La note de la présence et participation, la note du compte rendu et la note du test pratique.

Table des matières

Table des figures

Préambule :

L'étude générale des fonctions à variables réelles nécessite de temps à autre la résolution d'équations de type $f(x) = 0$.

Autrement dit, nous sommes amenés à trouver les zéros de fonctions non linéaires, cest-à-dire les valeurs réelles α telles que $f(\alpha) = 0$; ou, ce qui est équivalent, à résoudre une équation de type : $g(x) = x$.

Chapitre 1

TP N° 1 : Rappels et historiques sur Matlab

1.1 Introduction :

MATLAB est une abréviation de Matrix LABoratory. Ecrit à l'origine, en Fortran, par C. Moler, MATLAB était destiné à faciliter l'accès au logiciel matriciel développé dans les projets LINPACK et EISPACK. La version actuelle, écrite en C par the MathWorks Inc., existe en version professionnelle et en version étudiant. Sa disponibilité est assurée sur plusieurs plates-formes : Sun, Bull, HP, IBM, compatibles PC (DOS, Unix ou Windows), Macintoch, iMac et plusieurs machines parallèles.

MATLAB est un environnement puissant, complet et facile à utiliser destiné au calcul scientifique. Il apporte aux ingénieurs, chercheurs et à tout scientifique un système interactif intégrant calcul numérique et visualisation. C'est un environnement performant, ouvert et programmable qui permet de remarquables gains de productivité et de créativité. MATLAB est un environnement complet, ouvert et extensible pour le calcul et la visualisation. Il dispose de plusieurs centaines (voire milliers, selon les versions et les modules optionnels autour du noyau Matlab) de fonctions mathématiques, scientifiques et techniques. L'approche matricielle de MATLAB permet de traiter les données sans aucune limitation de taille et de réaliser des calculs numériques et symboliques de façon fiable et rapide. Grâce aux fonctions graphiques de MATLAB, il devient très facile de modifier interactivement les

différents paramètres des graphiques pour les adapter selon nos souhaits.

L'approche ouverte de MATLAB permet de construire un outil sur mesure. On peut inspecter le code source et les algorithmes des bibliothèques de fonctions (Toolboxes), modifier des fonctions existantes et ajouter d'autres. MATLAB possède son propre langage, intuitif et naturel qui permet des gains de temps de CPU spectaculaires par rapport à des langages comme le C, le Turbo Pascal et le Fortran. Avec MATLAB, on peut faire des liaisons de façon dynamique, à des programmes C ou Fortran, échanger des données avec d'autres applications (via la DDE : MATLAB serveur ou client) ou utiliser MATLAB comme moteur d'analyse et de visualisation.

MATLAB comprend aussi un ensemble d'outils spécifiques à des domaines, appelés Toolboxes (ou Boîtes à Outils). Indispensables à la plupart des utilisateurs, les Boîtes à Outils sont des collections de fonctions qui étendent l'environnement MATLAB pour résoudre des catégories spécifiques de problèmes. Les domaines couverts sont très variés et comprennent notamment le traitement du signal, l'automatique, l'identification de systèmes, les réseaux de neurones, la logique floue, le calcul de structure, les statistiques, etc...

MATLAB fait également partie d'un ensemble d'outils intégrés dédiés au Traitement du Signal. En complément du noyau de calcul MATLAB, l'environnement comprend des modules optionnels qui sont parfaitement intégrés à l'ensemble :

1. une vaste gamme de bibliothèques de fonctions spécialisées (Toolboxes),
2. Simulink, un environnement puissant de modélisation basée sur les schémas-blocs et de simulation de systèmes dynamiques linéaires et non linéaires,
3. Des bibliothèques de blocs Simulink spécialisés (Blocksets),
4. D'autres modules dont un Compilateur, un générateur de code C, un accélérateur,...
5. Un ensemble d'outils intégrés dédiés au Traitement du Signal : le DSP Workshop.

1.1.1 Quelles sont les particularités de MATLAB ?

MATLAB permet le travail interactif soit en mode commande, soit en mode programmation ; tout en ayant toujours la possibilité de faire des visualisations graphiques. Considéré comme un des meilleurs langages de pro-

grammations (C ou Fortran), MATLAB possède les particularités suivantes par rapport à ces langages :

- la programmation facile,
- la continuité parmi les valeurs entières, réelles et complexes,
- la gamme étendue des nombres et leurs précisions,
- la bibliothèque mathématique très compréhensive,
- l'outil graphique qui inclus les fonctions d'interface graphique et les utilitaires,
- la possibilité de liaison avec les autres langages classiques de programmations (C ou Fortran).

Dans MATLAB, aucune déclaration n'est à effectuer sur les nombres. En effet, il n'existe pas de distinction entre les nombres entiers, les nombres réels, les nombres complexes et la simple ou double précision. Cette caractéristique rend le mode de programmation très facile et très rapide. En Fortran par exemple, une subroutine est presque nécessaire pour chaque variable simple ou double précision, entière, réelle ou complexe. Dans MATLAB, aucune nécessité n'est demandée pour la séparation de ces variables.

La bibliothèque des fonctions mathématiques dans MATLAB donne des analyses mathématiques très simples. En effet, l'utilisateur peut exécuter dans le mode commande n'importe quelle fonction mathématique se trouvant dans la bibliothèque sans avoir à recourir à la programmation.
Pour l'interface graphique, des représentations scientifiques et même artistiques des objets peuvent être créées sur l'écran en utilisant les expressions mathématiques. Les graphiques sur MATLAB sont simples et attirent l'attention des utilisateurs, vu les possibilités importantes offertes par ce logiciel.

1.1.2 MATLAB peut-il s'en passer de la nécessité de Fortran ou du C ?

La reponse est non. En effet, le Fortran ou le C sont des langages importants pour les calculs de haute performance qui nécessitent une grande mémoire et un temps de calcul très long. Sans compilateur, les calculs sur MATLAB sont relativement lents par rapport au Fortran ou au C si les programmes comportent des boucles. Il est donc conseillé d'éviter les boucles, surtout si celles-ci est grande.

1.1.3 Ecriture d'un programme m-file par MATLAB :

En MATLAB, les programmes se terminent par une extension '.m' dans le nom du fichier programme. Aucune compilation n'est à faire avant l'exécution du programme. Au cours de l'exécution, un message d'erreur apparaît et indique les lieux où se trouvent les erreurs. Pour lancer l'exécution du programme, il faut se mettre toujours dans le même répertoire où se trouve ce programme.

1.2 Commandes structurées dans Matlab (Boucles et structures)

1.2.1 Manipulation 01 : Multiplication de matrices

Le produit matriciel s'en déduit : le produit de la matrice $A(n \times m)$ par la matrice $B(m \times p)$ est la matrice $C(n \times p)$ telle que l'élément c_{ij} est égal au produit scalaire de la ligne i de la matrice A par la colonne j de la matrice B.

$$c_{ij} = \sum_{k=1}^{m} a_{ik} * b_{kj}; \text{ avec } i = 1 \cdots n \text{ et } j = 1 \cdots p$$

Programme 01 : Ecrire Un programme en Matlab (m-file : sur le nom tp1prog1.m) qui permet de faire le produit de deux matrices en utilisant la boucle **for**. Refaire le programme en utilisant la boucle **while**.

1.2.2 Manipulation 02 : somme des n termes d'une série

On donne la série des n termes définie comme suit :

$$S1 = \sum_{k=1}^{n} \frac{(-1)^k k}{2^k}$$

Programme 02 : Ecrire un programme en Matlab (m-file : sur le nom *tp1prog2.m*) qui permet de faire la somme des n termes de la série précédente

$S1$ en utilisant la boucle **while**. On exécute le programme pour $n = 4, n = 20$ et $n = 100$. Refaire le programme en utilisant la boucle **for**.

1.2.3 Manipulation 03 : Lire des variables et faire la somme des n termes d'une série

Pour lire des variable par Matlab en utilisant la commande *input* et pour afficher un résultat on utilise la commande *fprintf* ou *disp*.
Programme 03 : Ecrivez un programme en Matlab (m-file : sur le nom *tp1prog3.m*) qui permet de demander deux entiers a et b qui affiche le résultat de la somme $S2$ avec : $S2 = \sum_{k=1}^{b} k^a$ en utilisant la boucle **for**. Refaire le programme en utilisant la boucle **while**.

Exercice : Dans un script Matlab, écrire un simple programme (sur le nom : *tp1exercice.m*) qui fait un test sur un nombre : pair ou impaire ?

1.3 Solutions

1.3.1 Programmes sous Matlab

1- manipulation 01 : Multiplication de matrices

```
A=input('donner la matrice A :') ;
B=input('donner la matrice B : ') ;
n=input('donner la dimension n : ') ;
for i = 1 :n
for j = 1 :n
x = 0
for k = 1 :n
x = x + A(i,k)*B(k,j) ;
end
C(i,j) = x
end
end
```

2- Manipulation 02 : somme des n termes d'une série
a- boucle for–end :

```
n=input('nombr de term n ?')
```

s=0 ;
for k=1 :n
$s = s + (-1)^k * k/2^k$;
end
fprintf('la somme des %g termes de la série sont %f \n',n,s) ;

b- boucle while–end :
n=input('nombr de term n ?')
s=0 ;
k=1 ;
while (k<=n)
$s = s + (-1)^k * k/2^k$;
k=k+1 ;
end
fprintf('la somme des %g termes de la série sont %f \n',n,s)

3- Manipulation 03 : Lire des variables et faire la somme des n termes d'une série
a- boucle for–end :
a=input('entrer a : ') ;
b=input('entrer b : ') ;
s=0 ;
for k=1 :b
$s = s + k^a$;
end
fprintf('le résultat est s= %f \n',s)
b- boucle while–end :
a=input('entrer a : ') ;
b=input('entrer b : ') ;
s=0 ; k=1 ;
while (k<=b)
$s = s + k^a$;
k=k+1 ;
end
fprintf('le résultat est s= %f \n',s)

Chapitre 2

TP N° 2 : Méthode de Dichotomie (Bissection)

2.1 But

Le problème est de trouver (par la programmation sous Matlab de la méthode de Dichotomie) des valeurs approchées des solutions d'une équation $f(x) = 0$ où f est une fonction non linéaire, le plus souvent continue et dérivable, sur un intervalle I. dans le cas général, en utilisant des méthodes itératives, qui donnent une suite d'approximations successives s'approchant de la solution exacte.

2.2 Principales méthodes de résolutions approchées de $f(x) = 0$

On considère un intervalle $[a, b]$ et une fonction f continue de $[a, b]$ dans $\mathbb{R}$. On suppose que $f(a) * f(b) < 0$ et que l'équation $f(x) = 0$ admet une unique solution α sur l'intervalle $]a, b[$.

2.2.1 Méthode de Dichotomie

La méthode de Dichotomie consiste à construire une suite (x_n) qui converge vers la racine α de la manière suivante :

Principe : on prend pour x_0 le milieu de l'intervalle $[a, b], (x_0 = \frac{a+b}{2})$. La racine se trouve alors dans l'un des deux intervalles $]a, x_0[$ ou $]x_0, b[$ ou bien elle est égale à x_0.
1. Si $f(x_0) = 0$, c'est la racine de f et le problème est résolu.
2. Si $f(x_0) \neq 0$, nous regardons le signe de $f(a) * f(x_0)$
a). Si $f(a) * f(x_0) < 0$, alors $\alpha \in]a, x_0[$
b). Si $f(x_0) * f(b) < 0$, alors $\alpha \in]x_0, b[$
On recommence le processus en prenant l'intervalle $[a, x_0]$ au lieu de $[a, b]$ dans le premier cas, et l'intervalle $[x_0, b]$ au lieu de $[a, b]$ dans le second cas. De cette manière, on construit par récurrence sur n trois suites $(a_n), (b_n)$ et (x_{0n}) telles que $a_1 = a$, $b_1 = b$ et telles que pour tout $n \geq 0$,
- si $f(a) * f(x_0) < 0$, alors $\alpha \in]a, x_0[$. On pose $a_1 = a$, $b_1 = x_0$.
- si $f(a) * f(x_0) = 0$, alors $\alpha = x_0$.

- si $f(a) * f(x_0) > 0$, alors $\alpha \in]x_0, b[$. On pose $a_1 = x_0$, $b_1 = b$.
On prend alors pour x_1 le milieu de $[a_1, b_1]$.
On construit ainsi une suite $x_0 = (a + b)/2, x_1 = (a_1 + b_1)/2, ..., x_0 = (a_n + b_n)/2$; telle que $|\alpha - x_n| \leq \frac{(b-a)}{2n+1}$. Etant donné une précision ϵ, cette méthode permet d'approcher α à un nombre prévisible d'itérations. L'algorithme ci-dessus s'appelle l'algorithme de Dichotomie.

2.2.2 Algorithme (Organigramme) de la méthode de Dichotomie

Entrées
Saisir a, la borne de gauche de l'intervalle ;
Saisir b, la borne de droite de l'intervalle $(a < b)$;
Saisir la précision ϵ souhaitée ;
Traitement et sorties
Si $f(a)$ et $f(b)$ sont du même signe alors
| Afficher "On ne peut pas faire de Dichotomie!"
Sinon
| Tant que l'écart entre a et b est supérieur à la précision ϵ répéter
| | c prend comme valeur la moyenne des nombres a et b
| | si $f(a)$ et $f(c)$ sont de signes contraires alors
| | | Affecter la valeur c à b
| | sinon

| | | Affecter la valeur c à a
| | Fin de si
| Fin de Tant que
| | Afficher "La solution est comprise entre a et b"
Fin de si

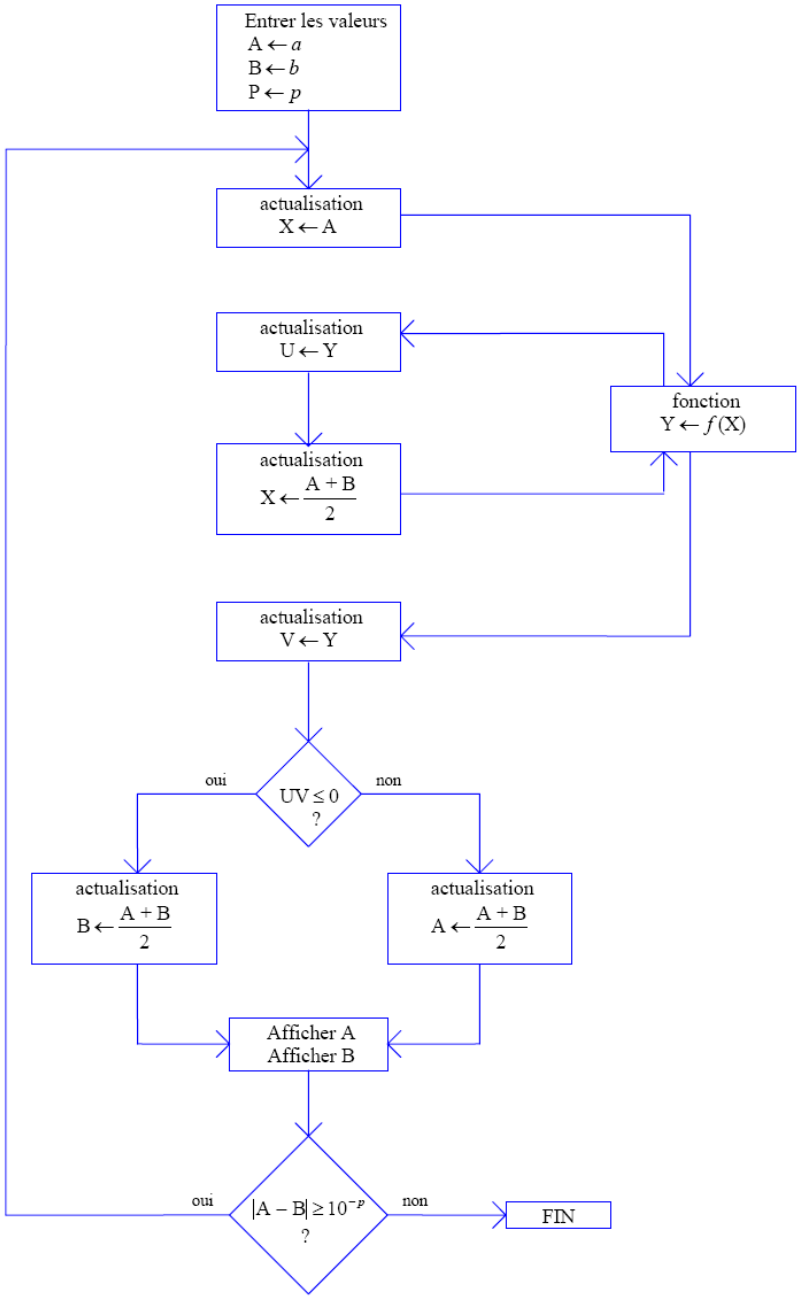

FIGURE 2.1: Organigramme de la méthode de Dichotomie.

2.3 Jeux de données

On travaillera avec les fonctions et intervalles suivants :

$$\begin{cases} f_1(x) = x - e^{\sin(x)} \\ [a,\ b] = [1,\ 10] \end{cases} \tag{2.1}$$

$$\begin{cases} f_2(x) = x^3 - 12x^2 - 60x + 46 \\ [a,\ b] = [0,\ 1] \end{cases} \tag{2.2}$$

$$\begin{cases} f_3(x) = x^3 - 12x^2 + 1 \\ [a,\ b] = [0,\ 1] \end{cases} \tag{2.3}$$

$$\begin{cases} f_4(x) = \cos(x) - x^3 \\ [a\ b] = [0\ 1] \end{cases} \tag{2.4}$$

2.4 Travail à réaliser

Programmation de la méthode de Dichotomie :
Écrire les programmes sous MATLAB permettant d'appliquer la méthode en question aux fonctions précédemment définies.
On prendra un test d'arrêt de la forme $|x_{n+1} - x_n| < \epsilon$ et on prendra soin de prévoir un compteur d'itérations qui permettra d'interrompre le traitement dès que N_{max} d'itérations sont effectuées sans que la précision ϵ ne soit atteinte. On pourra prendre par exemple $N_{max} = 100$.
- Les données seront a, b et ϵ, N_{max} et la fonction utilisée ; Les résultats seront la racine obtenue ainsi que son image par la fonction utilisée, le nombre d'itérations effectuées, et l'erreur de calcul. - Tester sur les fonctions f_1, f_2, f_3 et f_4 pour $\epsilon = 10^{-3}, 10^{-6}, 10^{-9}$ et 10^{-12}.

2.5 Solutions

2.5.1 Programme sous Matlab

1- Avec la boucle 'for ... end'
On prend l'exemple de la solution d'une fonction : $f(x) = x - e^{\sin(x)}$:

```
clc
clear all
x=1 :0.1 :10;
f=inline('x-exp(sin(x))');
plot(x,f(x)), grid;
a=1; fa=f(a);
b=10; fb=f(b);
i=0; Nmax=20;
eps=1.0e-3;
if ((fa*fb)<0)
      for (i=0 :Nmax)
        x=(a+b)/2;
        i=i+1;
          if (sign(f(x)) == sign(fa))
            a=x; fa=f(x);
          else
            b=x; fb=f(b);
          end
           fprintf('dans l''ititation i=%d \t la solution est x0=f \t f(x0)= %f
\n',i,x,f(x))
       end;
      fprintf('La solution finale est x0 = %f \n',x)
else
   disp('On ne peut pas faire de Dichotomie dans cet intervalle!!')
end
```

2- Avec la boucle 'while ... end'

On prend le même exemple $f(x) = x - e^{\sin(x)}$:

```
clc
clear all
x=1 :0.1 :10;
f=inline('x-exp(sin(x))');
plot(x,f(x)), grid
a=1; fa=f(a);
b=10; fb=f(b);
i=0; eps=1.0e-3;
if ((fa*fb)<0)
      while (b-a)>eps
        x=(a+b)/2;
```

```
        i=i+1 ;
         if (sign(f(x)) == sign(fa))
           a=x ; fa=f(x) ;
         else
           b=x ; fb=f(x) ;
          end
          fprintf('dans l'ititation i=%d \t la solution est x0=f \t f(x0)= f
\n',i,x,f(x))
       end
   fprintf('La solution finale est x0 = %f \n ',x)
else
   disp('On ne peut pas faire de Dichotomie dans cet intervalle!!')
end
```

2.5.2 Programme en C

```
   #include <stdio.h>
#include <math.h>
#define g(x) (pow(x,3) - 12 * x + 1)
int main()
{
float a,b,x0,eps,e,p,n ;
int i ;
printf("donner a,b et eps \n") ;
scanf("%f%f%f",&a,&b,&eps) ;
i=0 ;
if (g(a)*g(b)<0)
{
do
{
i++ ;
x0=(a+b)/2.0 ;
if(g(a)*g(x0)<0)
{
b=x0 ;
e=fabs(a-x0) ;
}
```

```
else
{
a=x0;
e=fabs(b-x0);
} printf("\n %d   x0= %f   g(x0)= %f   e= %f",i,x0,g(x0),e);
}
while (e>eps);
}
else
{printf("pas de solution, SVP tapey un autre intervalle");}
getch();
}
```

Chapitre 3

TP N° 3 : Méthode du Point-fixe

3.1 But

Dans ce TP, nous nous intéressons à la résolution numérique des équations non linéaires de type $g(x) = x$, où g est une fonction non linéaire. Pour résoudre ce type de problème, on utilise la méthode du point fixe (résolution d'équations du type $g(x) = x$). Cette méthode numérique est une méthode itérative : à partir d'une donnée x_0, on construit x_1 puis x_2, puis, pas à pas, les premiers termes de la suite $x_k, k \in \mathbb{N}$.

3.2 Principales méthodes de résolutions approchées de $f(x) = 0$

3.2.1 Critère d'arrêt de l'algorithme

Numériquement il faut pouvoir arrêter l'algorithme une fois que celui-ci a convergé, c'est-à- dire une fois que x_n est suffisamment proche de la solution exacte (notée $\overline{x}$). Nous allons pour cela définir des critères d'arrêt. Pour une tolérance donnée, il faudrait pouvoir vérifier que le module de la différence entre la solution exacte et la solution approchée est inférieur à ϵ , c'est à dire que :

$$|x_n - \overline{x}| < \epsilon$$

Cependant, comme, en général, la solution exacte $\overline{x}$ n'est pas connue, ce critère n'est pas exploitable numériquement. Il existe quand même deux types de critères d'arrêt utilisables en pratique :
1. Contrôle du résidu : les itérations s'achèvent dès que :

$$|f(x)| < \epsilon \quad (\text{ou} \quad |g(x_n) - \overline{x}| < \epsilon)$$

2. Contrôle de l'incrément : les itérations s'achèvent dès que :

$$|x_{n+1} - x_n| < \epsilon$$

3.2.2 Nombre maximal d'itérations

En plus d'un de ces critères d'arrêt, il faut pouvoir arrêter l'algorithme lorsque celui-ci diverge ou lorsque la convergence est trop lente. On va alors définir le nombre d'itérations maximal N_{max}. L'algorithme s'arrêtera dès que $n > N_{max}$.

3.2.3 Calcul de l'erreur

Le calcul de l'erreur sert d'une part à vérifier les estimations théoriques, et, d'autre part à comparer la vitesse de convergence des différentes méthodes pour approcher la solution exacte $\overline{x}$. Ainsi, en plus du calcul d'une approximation de $\overline{x}$ on a parfois besoin de calculer l'erreur à chaque étape de l'algorithme $e_n = |x_n - \overline{x}|$.
De nouveau, en général, $\overline{x}$ n'est pas connue (même si $\overline{x}$ est connue sa valeur sera approchée par l'ordinateur (par exemple si $\overline{x} = \pi$)). En pratique, on calculera un vecteur e tel que $e_n = |x_n - \overline{x}_a|$.
1. x_a vaut $\overline{x}$ si $\overline{x}$ est connue.
2. $\overline{x}_a$ est une valeur approchée de $\overline{x}$ aussi précise que possible (calculée avec une tolérance ϵ très petite) sinon.

3.3 Algorithmique

On souhaite calculer une valeur approchée d'un point fixe d'une fonction g à partir de choix d'un point x_0 donné. Le point fixe doit être calculé avec une précision de ϵ donnée.
Quels sont les paramètres d'entrée ?
- Quels sont les paramètres de sortie ?

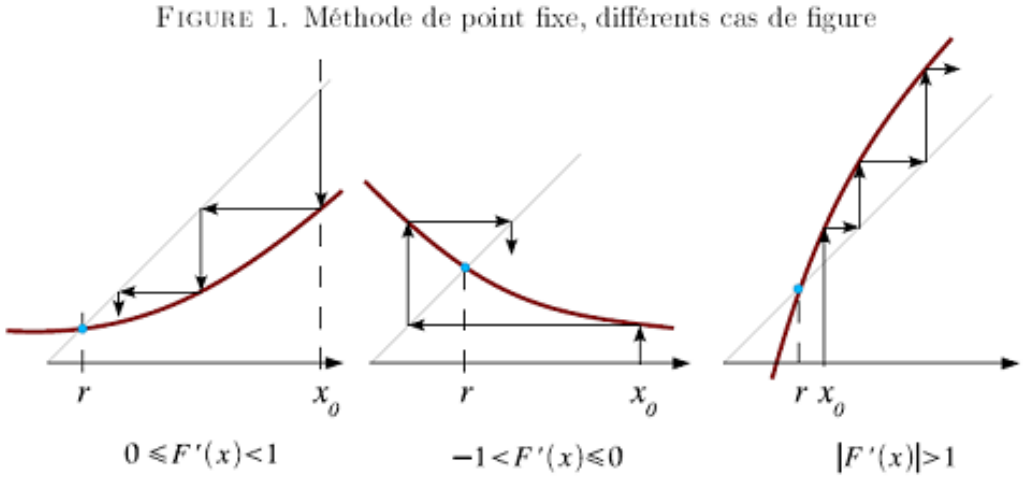

FIGURE 3.1: Exemple 1 'Méthode du Point-fixe'

Définir un critère d'arrêt pour cet algorithme,
Ecrire l'algorithme complet de la fonction PointFixe.m,
Ecrire l'algorithme d'une fonction supplémentaire ayant un paramètre d'entrée supplémentaire x_a et un paramètre de sortie supplémentaire le vecteur e.

3.4 Travail à réaliser

Ecrire la fonction pintfixe.m sous Matlab en se basant sur :

3.4.1 Entrées et sorties :

g : la fonction étudiée,
$zero$: la racine trouvée par la méthode,
x_0 : le point initial,
$erreur$: l'erreur estimée,
N_{max} : le nombre maximal d'itérations,
n_{iter} : le nombre d'itérations effectuées,
ϵ : le critère d'arrêt (erreur tolérée).

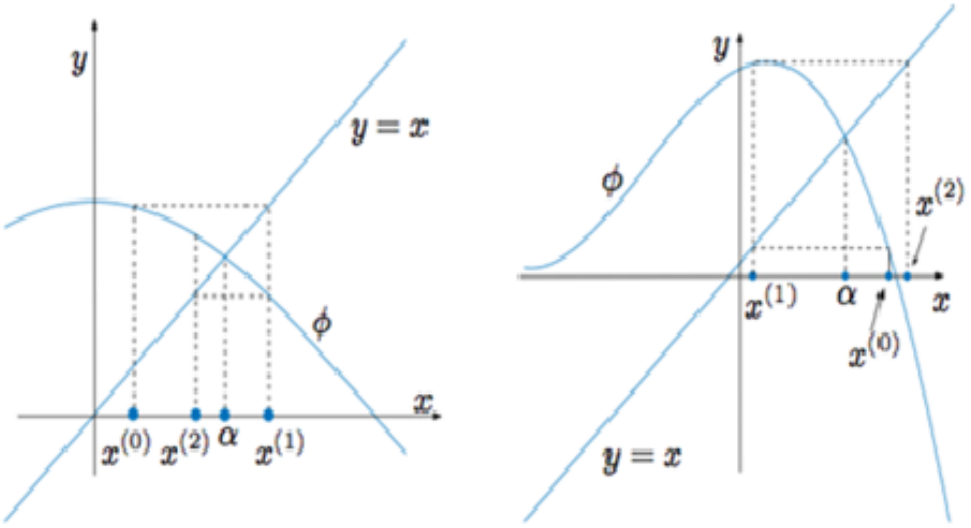

FIGURE 3.2: Exemple 2 'Méthode du Point-fixe'

3.4.2 Algorithme :

1. On commence par choisir le point initial x_0 $(n = 0)$;
2. On calcule $x_{n+1} = g(x_n)$;
3. Si $|x_{n+1} - x_n| <$ alors la méthode a convergé, et on s'arrête ;
4. Si n_{iter} atteint N_{max} alors la méthode a divergé, ou elle n'a pas pu converger avec N_{max} itérations et on s'arrête ;
5. Sinon, on passe à l'étape 2 pour une nouvelle itération $n + 1$ (n devient $n + 1$).

3.4.3 Application :

a). Considérons l'équation non linéaire : $f(x) = x^3 + 4x^2 - 10 = 0$, qui admet une racine x_0 dans l'intervalle $[1,\ 2]$.
Trouvez trois façons d'écrire $f = 0$ sous la forme d'un point-fixe (g_1, g_2 et g_3) ?
1. on fait un plot des (g_1 avec x) , (g_2 avec x) et (g_3 avec x).
2. Appliquez la fonction Matlab 'pointfixe.m' que vous avez créé sur $g_1(x), g_2(x)$ *et* $g_3(x)$, en mettant : $x_0 = 1.5$,ϵ=0.001, $N_{max} = 50$.
3. Quelle est la fonction (g_1, g_2 ou g_3) qui donne la convergence la plus rapide.

b). Toujours en Matlab, coder la méthode Pointfixe.m ;

1. Appliquer l'algorithme du point fixe pour trouver les zéros de $h(x) = 2\sin(x) - x$ en prenant $x_0 = [1.5,\ 2]$. Après le plot de $g(x)$ *hold on* avec le plot de x.
2. Remarquer que 0 est un point fixe de h. Appliquer l'algorithme du point fixe à h en prenant $x_0 = 0.01$ puis $x_0 = -0.01$. Que se passe-t-il ? Expliquer.

3.5 Exercice

Sous Matlab, appliquer la méthode du point fixe pour résoudre l'équation suivante :
$f(x) = x - \cos(x)$, avec [a, b]=[0, 1], ϵ=10^{-3}.

3.6 Solutions

3.6.1 Programmes sous Matlab

1- Avec la boucle for–end :

```
La méthode du point fixe ;
On a l'équation f(x) = cos(x) − x = 0
clc ; clear all
x = 0.5 :0.01 :5 ;
g=inline('cos(x)') ;
plot(x,g(x),x,x) ; grid ;
x0 = input('x0 = ') ;
eps = input('eps = ') ;
nmax = input('nmax = ') ;
for n = 1 : nmax
x = x0 ;
x0 = g(x) ;
err = abs(x0-x) ;
if err <= eps
fprintf('pour i=%d   la solution x0=%f   avec erreur = %f \n',n,x0,err) ;
end
end
```

2- Avec la boucle while–end :

```
clear all ; clc ;
x = 0.5 :0.01 :5 ;
g=inline('cos(x)') ;
plot(x,g(x),x,x) ; grid ;
x0 = input('x0 = ') ;
eps = input('eps = ') ;
nmax = input('nmax = ') ;
n=1 ;
while (n<=nmax)
x = x0 ;
x0 = g(x) ;
err = abs(x0-x) ;
n=n+1 ;
if err <= eps
fprintf(' pour i=%d  la solution x0=%f   avec erreur =%f \n',n,x0,err) ;
end
end
```

3.6.2 Programmes en C

```
   #include <stdio.h>
#include <stdlib.h>
#include <math.h>
double f(double x){
return x+1 ;
}
///la fonction g(x)
double g(double x){
double y= x-f(x) ;
//3*x
return y ;}
int main(int argc, char *argv[])
{
int i,n ;
double x[100] ;
double x0,EPS ;
printf("Precision voulue : ") ;
```

```
scanf("%lf",&EPS);
printf("entrer la valeur initiale X0 : ");
scanf("%f",&x0);
x[0] = x0; // Initialisation de la valeur de départ
int k=0;
do{
x[k+1] = g(x[k]); // calcul des termes de la série x(k+1)
if(fabs(x[k+1]-x[k])<=EPS ) // critère de convergence
{
printf(" Nombre d'Iterations K=%d   La solution X =%f", k, x[k+1]);
goto fin;
}
k++;
} while(fabs(x[k+1] - x[k])> EPS );// continuer les calcul tant que le critère
n'est pas atteind fin : system("PAUSE");
return 0;
}
```

Chapitre 4

TP N° 4 : Méthode de Newton

4.1 But

Le problème est de trouver des valeurs approchées des solutions d'une équation $f(x) = 0$ où f est une fonction non linéaire, le plus souvent continue et dérivable, sur un intervalle I. Dans le cas général, on utilise des méthodes itératives, qui donnent une suite d'approximations successives s'approchant de la solution exacte.

4.2 Principales méthodes de résolutions approchées de $f(x) = 0$

4.2.1 Critère d'arrêt de l'algorithme

On considère un intervalle $[a,\ b]$ et une fonction f continue sur $[a,\ b]$ dans $\mathbb{R}$. On suppose que $f(a) \times f(b) < 0$ et que l'équation $f(x) = 0$ admet une unique solution α sur l'intervalle $]a,\ b[$.

4.2.2 Méthode de Newton

La méthode de Newton est une méthode particulière de la méthode du point fixe. Elle est basée sur l'idée de construction d'une suite (x_n) qui

converge vers α d'une manière quadratique. On construit la suite (x_n) définie par $x_0 \in [a,\ b]$ et la relation de récurrence suivante :

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

est convergente vers α de manière au moins quadratique. Où f' désigne la dérivée de la fonction f. La figure(1) montre une illustration graphique.
Nous allons annoncer un résultat de convergence globale concernant la mé-

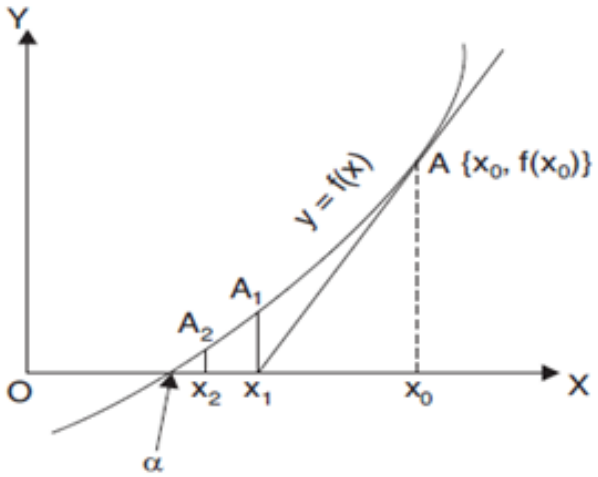

FIGURE 4.1: Méthode de Newton .

thode de Newton pour des fonctions ayant une concavité déterminée (convexe ou concave).
Soit $f : [a,\ b] \in \mathbb{R}$ de classe $\mathbb{C}^2$ vérifiant :
- $f(a) \times f(b) < 0$,
- $f'(x) \neq 0, \forall x \in [a,\ b]$,
- $f''(x) \neq 0, \forall x \in [a,\ b]$.
Alors la suite (x_n) définie par :

$$\begin{cases} x \in [a,\ b] : f(x_0) \times f''(x_0) > 0 \\ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \end{cases} \tag{4.1}$$

est convergente vers $\alpha \Rightarrow$ qu'il existe un unique solution $\alpha \in [a,\ b]$ tel que $f(\alpha) = 0$ comme f'' est de signe constant, on distingue deux cas :
(1)- Si $f''(x) > 0, \forall x \in [a,\ b]$(donc $f(x_0) > 0$), alors :

(a)- Si $f'(x) > 0, \forall x \in [a,\ b]$ on a :

$$\begin{cases} f(x) > 0, \forall x \in]\alpha,\ b] \\ f(x) < 0, \forall x \in [a,\ \alpha[\end{cases} \tag{4.2}$$

Comme $f(x_0) > 0$, alors $x_0 \in]\alpha,\ b]$. Par conséquent,
$g(x) = x - \frac{f(x)}{f'(x)} \Rightarrow g'(x) = \frac{f(x).f''(x)}{(f'(x))^2} \geq 0, \forall x \in]\alpha,\ b]$
Donc g est croissante sur $]\alpha,\ b]$, d'où
$\alpha < x_0 \Rightarrow \alpha = g(\alpha)(x_0) = x_1 \Rightarrow x_1 \in]\alpha,\ b]$
de plus,
$g(x_0) = x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} < x_0 \Rightarrow \alpha \leq x_1 \leq x_0$,
Par conséquent, on obtient :
$\alpha \leq \ldots \leq x_{n+1} \leq x_n \leq \ldots \leq x_2 \leq x_1 \leq x_0$,
donc
$|x_{n+1} - \alpha| < |x_n - \alpha|$
c'est-à-dire (x_n) est décroissante minorée par α, donc (x_n) est convergente. Comme $x_{n+1} = g(x_n)$ et que g est continue, (x_n) converge vers α l'unique point fixe de g.
(b)- Si $f'(x) < 0, \forall x \in [a,\ b]$ un croisonnement semblable au précédent implique que (x_n) est croissante majorée pas α.
(2)- Si $f''(x) < 0, \forall x \in [a,\ b]$(donc $f(x_0) < 0$), alors le raisonnement précédent avec f remplacée par $-f$, implique que la suite (x_n) est convergente vers α.

4.2.3 Test d'arrêt

Une fois construite la suite (x_n) convergeant vers α vérifiant $g(\alpha) = \alpha$, et une fois fixée la tolérance ϵ , nous cherchons le premier entier n_0 vérifiant : $|x_{n_0+1} - x_{n_0}| < \epsilon$.

4.2.3.1 Exemple

Pour illustrer la méthode, recherchons le nombre positif x vérifiant $cos(x) = x^3$. Reformulons la question pour introduire une fonction devant s'annuler : on recherche le zéro positif (la racine) de $f(x) = cos(x) - x^3$, la dérivée est $f'(x) = -sin(x) - 3x^2$.
Comme $cos(x) \leq 1$ pour tout x et $x^3 > 1$ pour $x > 1$, nous savons que notre

zéro se situe entre 0 et 1. Nous essayons une valeur de départ de $x_0 = 0,5$.

$$\begin{cases} x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0.5 - \frac{cos(0.5)-0.5^3}{-sin(0.5)-3+0.5^2} \approx 1.1121416371 \\ x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = \quad \vdots \quad \approx 0.909672693736 \\ x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = \quad \vdots \quad \approx 0.866263818209 \\ x_4 = x_3 - \frac{f(x_3)}{f'(x_3)} = \quad \vdots \quad \approx 0.865477135298 \\ x_5 = x_4 - \frac{f(x_4)}{f'(x_4)} = \quad \vdots \quad \approx 0.865474033111 \\ x_6 = x_5 - \frac{f(x_5)}{f'(x_5)} = \quad \vdots \quad \approx 0.865474033101 \\ x_7 = x_6 - \frac{f(x_6)}{f'(x_6)} = \quad \vdots \quad \approx 0.865474033102 \end{cases} \tag{4.3}$$

4.3 Jeux de Données

On travaillera avec les fonctions et intervalles suivants :

$$\begin{cases} f_1(x) = x - e^{\sin(x)} \\ [a,\ b] = [1,\ 10] \end{cases} \tag{4.4}$$

$$\begin{cases} f_2(x) = x^3 - 12x^2 - 60x + 46 \\ [a,\ b] = [0,\ 1] \end{cases} \tag{4.5}$$

4.4 Travail à réaliser

- Ecrire les programmes sous MATLAB permettant d'appliquer la méthode en question aux fonctions précédemment définies.
- On prendra un test d'arrêt de la forme $|x_{n+1} - x_n| < \epsilon$ et on prendra soin de prévoir un compteur d'itérations qui permettra d'interrompre le traitement dès que N_{max} d'itérations sont effectuées sans que la précision ϵ ne soit atteinte. On pourra prendre par exemple $N_{max} = 50$.
- Les paramètres d'entrée de la fonction Newton seront x_0, ϵ, N_{max}, la fonction f et sa dérivée f'; les résultats seront la racine obtenue ainsi que son image par la fonction f et le nombre d'itérations effectuées et l'erreur de calcul.

4.4.1 Choix de la valeur initiale

On teste cette méthode pour la fonction f_1. On testera à chaque fois pour $\epsilon = 10^{-3}, 10^{-6}, 10^{-9}$ et 10^{-12}. On prendra comme valeur initiale : $x_0 = -10, x_0 = 1, x_0 = 2$ puis $x_0 = 10$.
On teste cette méthode aussi pour la fonction f_2. On testera à chaque fois pour $\epsilon = 10^{-3}, 10^{-6}, 10^{-9}$ et 10^{-12}. On prendra comme valeur initiale : $x_0 = -1, x_0 = 0.3, x_0 = 0.5$ puis $x_0 = 3$.

4.5 Algorithme (Organigramme) de la méthode de Newton

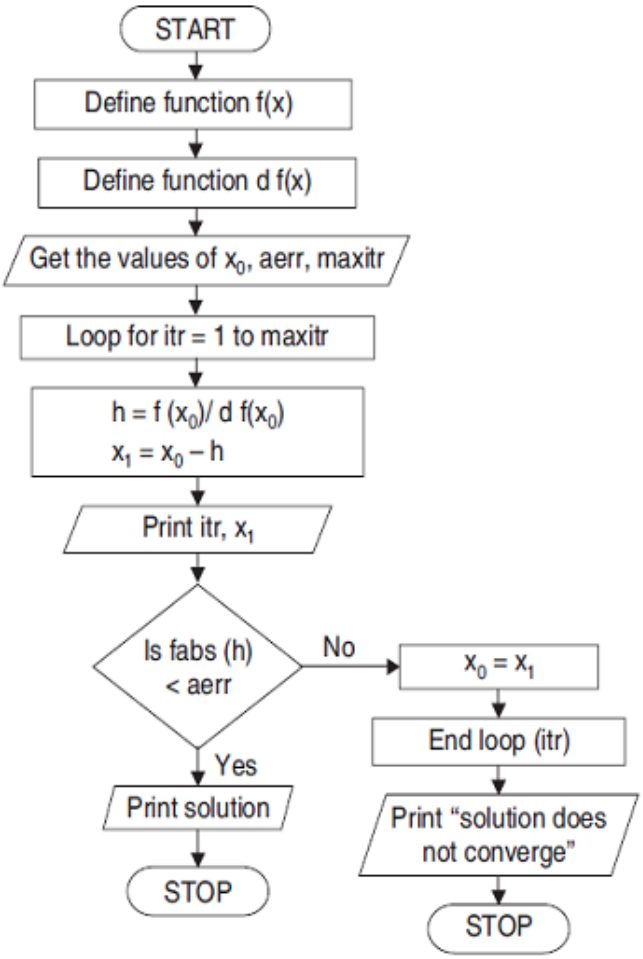

FIGURE 4.2: Algorithme de la méthode de Newton.

4.6 Exercice

On utilise le Matlab, appliquer la méthode de Newton pour résoudre l'équation suivante : $f(x) = x - cos(x)$. avec $[a,\ b] = [0,\ 1], \epsilon = 10^{-3}$.
Conclure : Comparer les résultats dans les deux cas avec la méthode du point-fixe et celle de Dichotomie.

4.7 Solutions

4.7.1 Programmes sous Matlab

1- Avec la boucle for–end :

```
% On prend par exemple la fonction f(x) = cos(x) - x^3 :
clear all ; close all ; clc ;
a=input('donner a= ') ; b=input('donner b= ') ;
eps=input('donner eps= ') ;
Nmax=input('pour la boucle for, donner Nmax= ') ;
f=inline('cos(x) - x^3') ;
df=inline('-sin(x) - 3 * x^2') ;
ddf=inline('-cos(x) - 6 * x') ;
if (df(a)== 0) && (df(b)== 0)
disp('On peut pas calculer la racine') ;
end
if (f(a)*ddf(a)>0) %initialisation de x0
x0=a ;
else x0=b ;
end
for (i=0 :Nmax)
x0 = x0 - (f(x0)/df(x0)) ;
err=abs(f(x0)/df(x0)) ;
if err>=eps
fprintf('i=%d
racine=%2.8f
f(x0)=%2.8f   err=%2.8f \n',i,x0,f(x0),err) ;
end
fplot(f,[a b]) ;
hold all ;
plot(x0,f(x0),'ro') ;
```

```
hold off; grid on;
pause(1)
end
```

2- Avec la boucle while-end :

```
clear all; close all; clc;
a=input('donner a= '); b=input('donner b= ');
eps=input('donner eps= ');
f=inline('cos(x) - x^3');
df=inline('-sin(x) - 3 * x^2');
ddf=inline('-cos(x) - 6 * x');
if (df(a)== 0) && (df(b)== 0)
disp('On peut pas calculer la racine');
end
if (f(a)*ddf(a)>0) %initialisation de x0
x0=a;
else x0=b;
end
err=1; i=0;
while (err >= eps)
i=i+1;
x0 = x0 - (f(x0)/df(x0));
err=abs(f(x0)/df(x0));
if err>=eps
fprintf('i=%d  racine=%2.8f  f(x0)=%2.8f  err=%2.8f \n',i,x0,f(x0),err);
end
fplot(f,[a b]);
hold all;
plot(x0,f(x0),'ro');
hold off; grid on;
pause(1)
end
```

4.7.2 Programmes en C

```
  #include <stdio.h>
#include <math.h>
float FONCTION (float x) return (pow((x),3.0)-12*x+1);}
```

```
#define DERIVE(x) (3*pow((x),2)-12)
#define DDERIVE(x) (6*x)
#define MSG_FONCTION "(x^3 - 12 * x + 1)"
main()
{
int i,n;
float fa,fb,dfa,dfb,ddfa;
float a,b,x0,x1,Er,EPSILON;
printf("entrer a,b EPSILON, \n");
scanf("%f%f%f",&a,&b,&EPSILON);
printf(" \n  Methode de Newton f(x)=%s=0 \n pour a=%2.3f b=%2.3f EP-
SILON =%2.6f \n",MSG_FONCTION,a,b,EPSILON);
fa=FONCTION(a);
fb=FONCTION(b);
dfb=DERIVE(b);
dfa=DERIVE(a);
ddfa=DDERIVE(a);
if((dfb==0.0) && (dfa == 0.0))
{ printf("Impossible!! "); }
if (fa*ddfa>0.0)
x0=a;
else
x0=b;
i=0;
do
{ i++;
x0=x0-(FONCTION(x0)/DERIVE(x0));
Er=fabs(FONCTION(x0)/DERIVE(x0));
printf("\n x0=%f f(x0)=%f Er=%f ", x0,FONCTION(x0),Er); }
while (Er>EPSILON);
printf("\n donc on a %d iteration, x0=%2.6f \n",i,x0);
getch(); }
```

Chapitre 5

TP N° 5 : Méthode de Lagrange

5.1 Introduction

La méthode de dichotomie est lente car elle n'utilise que partiellement l'information disponible : on ne se sert que des signes, et pas des valeurs de $f(a)$ et de $f(b)$. Aveuglément et inexorablement, le partage en deux de l'intervalle a lieu et ce, quelle que soit la fonction considérée. Or il est bien clair qu'avec $f(a) = -1$ et $f(b) = 10000$ par exemple, on a tout lieu de penser que la racine α de l'équation $f(x) = 0$ a bien des chances d'être plus proche de α que de $\frac{a+b}{2}$.
La méthode de Lagrange, ou des parties proportionnelles, remédie à ce problème : au lieu de travailler à chaque étape avec le point-milieu d'abscisse $\frac{a+b}{2}$, on fait intervenir l'abscisse c du point d'intersection de la droite joignant les points $A(a, f(a))$ et $B(b, f(b))$ avec l'axe des abscisses.
Concrètement, cela revient à remplacer la fonction f par une fonction affine et substituer à l'équation que l'on cherche à résoudre une banale équation du premier degré.

5.2 Objectifs

Ecrire un programme sous Matlab qui recherche la racine de $f(x) = 0$ sur $[a, b]$ par la méthode de Lagrange.

5.3 Méthode de LAGRANGE

Principe : La méthode de Lagrange est une variante de la méthode de Newton.
Soit $f \in C^1([a,b],\mathbb{R})$ ayant une convexité déterminée. Rappelons que pour calculer un zéro α de f par la méthode de Newton, on considère la suite (x_n) définie par :

$$\begin{cases} x_0 \text{ proche de } \alpha \\ f'(x_n)(x_{n+1}-x_n) = -f(x_n), \forall n \geq 0 \end{cases} \tag{5.1}$$

Dans certaines situations, la dérivée de f est très compliquée voir même

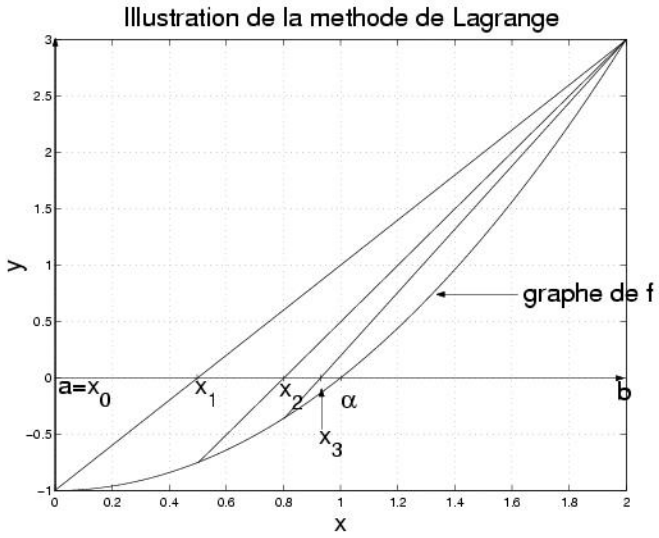

FIGURE 5.1: Illustration de la méthode de Lagrange.

impossible à calculer. Dans ce cas, nous approchons la dérivée par un quotient différentiel. Ce que nous obtenons est appelée la méthode de Lagrange :

$$\begin{cases} x_0, x_1 \text{ proche de } \alpha \\ \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}(x_{n+1}-x_n) = -f(x_n), \forall n \geq 1 \end{cases} \tag{5.2}$$

Ici, x_{n+1} dépend de x_n et de x_{n-1} : on dit que c'est une méthode à deux pas ; nous avons d'ailleurs besoin de deux itérés initiaux x_0 et x_1.
L'avantage de cette méthode est qu'elle ne nécessite pas le calcul de la dérivée f'. l'inconvénient est que nous perdons la convergence quadratique.
La fonction g correspondante vérifie :

$$x_{n+1} = g(x_n) = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}. \tag{5.3}$$

5.4 Convergence

Nous allons présenter un théorème de convergence.
Théorème : Soit $f : [a, b] \to \mathbb{R}$ de classe $\mathbb{C}^2$ telle que f' et f'' soient strictement positives sur $[a, b]$.
On suppose que $f(a) < 0; f(b) > 0$ et on appelle a l'unique solution de l'équation $f(x) = 0$. Alors : (1) La suite (x_n) telle que :

$$\begin{cases} x_0 = a \\ x_{n+1} = \frac{x_n f(b) - b f(x_n)}{f(b) - f(x_n)}, \forall n \geq 0 \end{cases} \tag{5.4}$$

est bien définie.
(2) La suite (x_n) est croissante, convergeant vers α.

5.5 Manipulation

Ecrire un programme sous Matlab qui recherche la racine de $f(x) = 0$ sur $[a, b]$ par la méthode de Lagrange, jusqu'à ce que $|x_{n+1} - x_n| < \epsilon$. On commence par $x_0 = a$ si $f(x_0) * f''(x_0) < 0$.
Tester le programme sur les deux fonctions suivantes :

1. $f(x) = x^3 - 12x^2 - 60x + 46 = 0, [a, b] = [0, 1]$ et $\epsilon = 10^{-3}$.
2. $f(x) = \cos(x) - x^3 = 0, [a, b] = [0, 1]$ et $\epsilon = 10^{-3}$.

5.6 Solutions

5.6.1 Programmes sous Matlab

1- Avec la boucle for–end :

```
clear all ; close all ; clc ;
a=input('donner a= ') ;
b=input('donner b= ') ;
eps=input('donner eps= ') ;
nmax=input('donner nmax= ') ;
f=inline('cos(x) - x^3') ;
ddf=inline('-cos(x)-6*x') ;
```

```
if (f(a)*ddf(a)<0)
x0=a;
c=b;
else x0=b;
c=a;
end
for (i=0 :nmax)
x0 = (f(x0) * c - x0 * f(c))/(f(x0) - f(c));
err = abs(x0 - (f(x0) * c - x0 * f(c))/(f(x0) - f(c)));
if (err>=eps)
fprintf('i=%d  racine=%2.8f  f(x0)=%2.8f  err=%2.8f\n',i,x0,f(x0),err);
end
fplot(f,[a b]);
hold all;
plot(x0,f(x0),'ro');
hold off; grid on;
end
```

2- Avec la boucle while–end :

```
clear all; close all; clc;
a=input('donner a= ');
b=input('donner b= ');
eps=input('donner eps= ');
nmax=input('donner nmax= ');
f=inline('cos(x) - x^3');
ddf=inline('-cos(x)-6*x');
if (f(a)*ddf(a)<0)
x0=a;
c=b;
else x0=b;
c=a;
end
err=1; i=0;
while (err>=eps)
i=i+1;
x0 = (f(x0) * c - x0 * f(c))/(f(x0) - f(c));
err = abs(x0 - (f(x0) * c - x0 * f(c))/(f(x0) - f(c)));
if (err>=eps)
fprintf('i=%d  racine=%2.8f  f(x0)=%2.8f  err=%2.8f\n',i,x0,f(x0),err);
end
```

```
fplot(f,[a b]);
hold all;
plot(x0,f(x0),'ro');
hold off; grid on;
pause(1)
end
```

5.6.2 Programmes en C

```
   #include<stdio.h>
#include<math.h>
float f(float x) return (x-exp(sin(x)));
#define df(x) (1-cos(x)*exp(sin(x)))
#define d2f(x) ((sin(x)-cos(x)*cos(x))*exp(sin(x)))
int main()
{
float a, b,c, x0,eps,e;
float fa,fb,dfa,dfb,d2fa;
int i;
printf("Donner a=");
scanf("%f",a);
printf("Donner b=");
scanf("%f",b);
printf("Donner eps=");
scanf("%f",eps);
fa=f(a); fb=f(b); dfa=df(a); dfb=df(b); d2fa=d2f(a);
if((dfa==0)  (dfb==0))
{ printf("Impossible!");}
else
{ if(fa*d2fa<0)
{ x0=a; c=b;
}
else
{ x0=b; c=a; }
e=fabs(b-a);
i=0;
while(e>eps)
{ i++;
```

```
x0=(f(x0)*c-x0*f(c))/(f(x0)-f(c)) ;
e=fabs(x0-(f(x0)*c-x0*f(c))/(f(x0)-f(c))) ;
printf("\n %d   x0=%f  (x0)= %f   e= %f",i, x0,f(x0),e) ; printf("\n") ; }}
system("pause") ;
return 0 ;
}
```

Chapitre 6

TP N° 6 : Méthode de Regula-falsi (fausse position)

6.1 But

L'objectif de ce TP est de comparer différentes méthodes de résolution d'équations non linéaires du type $f(x) = 0$.
On considère un intervalle $[a, b]$ et une fonction f continue de $[a, b] \in \mathbb{R}$. On suppose que $f(a) * f(b) < 0$ et que l'équation $f(x) = 0$ admet une unique solution α sur l'intervalle $[a, b]$.

6.2 Méthode de Regula-Falsi (fausse position)

Le principe de cette méthode est identique à celle de la méthode de Dichotomie. Au lieu d'utiliser le point médian, on utilise le point d'intersection avec l'axe des abscisses de la droite :

$$x(0) = a, \quad x(1) = b.$$

Ce point est donné par l'équation suivante qui se vérifie très simplement :

$$x(k) = a - f(a) * [(b - a)/(f(b) - f(a))]$$

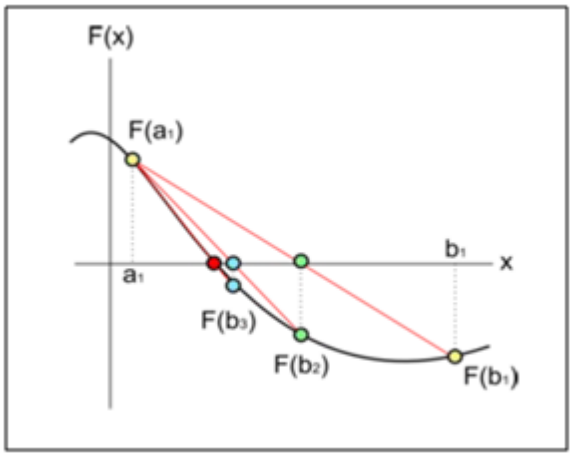

FIGURE 6.1: Recherche de la racine par la méthode de Regula-falsi.

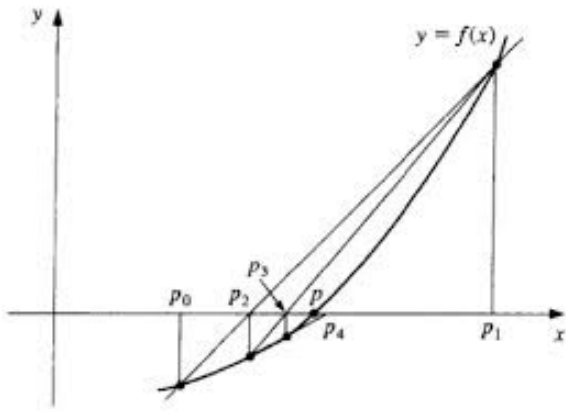

FIGURE 6.2: Exemple2 : Méthode de Regula-falsi.

6.2.1 Algorithme

entrées : $x, f(x), a$ et b ;
sorties : solution x_0 ;
1. Choisir un intervalle $[x(0) = a;\ x(1) = b]$ tel que $f(a) * f(b) < 0$:
2. Calculer la valeur de la fonction en $x(i)$
3. On retient comme nouvel intervalle $[a,\ x(i)]$ en respectant la condition de 1.
On est alors assuré de toujours encadrer la racine.
4. Répéter les étapes 2 et 3 jusqu'à l'obtention de la précision désirée, c'est à dire jusqu'à ce que $abs(f(x(i)) < \epsilon;\ \epsilon$ étant la précision désirée.
i désigne le nombre d'itérations.

6.3 Travail à réaliser

A. Écrire un programme Matlab permettant d'appliquer la méthode de Regula-falsi 'fausse position', On prendra un test d'arrêt de la forme $|f(xk)| < \epsilon$.
B. Créer, à partir du programme précédent, une fonction Matlab : $[x, f(x), k] = regulafalsi(f, a, b, \epsilon)$,
Les entrées seront f, a, b et ϵ.
Les sorties seront la racine obtenue x_0 ainsi que son image par la fonction utilisée $f(x_0)$ et le nombre ditérations effectuées i.
C. Afficher graphiquement et pas à pas la construction de cette méthode.
D. Utiliser la fonction Matlab regulafalsi.m pour résoudre les fonction f_1 et f_2 dans la section 4.

6.4 Jeux de données

On travaillera avec les fonctions et les intervalles suivants :

$$\begin{cases} f_1(x) = x - e^{sin(x)} \\ [a\ b] = [1\ 10] \end{cases} \tag{6.1}$$

$$\begin{cases} f_2(x) = x^3 - 12x^2 - 60x + 46 \\ [a\ b] = [0\ 1] \end{cases} \tag{6.2}$$

$$\begin{cases} f_3(x) = x^3 - 12x^2 + 1 \\ [a\ b] = [0\ 1] \end{cases} \tag{6.3}$$

$$\begin{cases} f_4(x) = cos(x) - x^3 \\ [a\ b] = [0\ 1] \end{cases} \tag{6.4}$$

6.5 Solutions

6.5.1 Programme sous Matlab

1- Avec la boucle 'for ... end' :

```
% la méthode de fausse position (regula falsi)
% f = x-exp(sin(x)) définie dans l'intervalle [a b].
% laboucle for--end
clear all; close all; clc;
f=inline('x-exp(sin(x))');
a=0;
b=10;
eps=10^-10;
i = 0;
Nmax=50;
x0=a;
% le programme principal
for i=1:Nmax
        x0=a-f(a)*(b-a)/(f(b)-f(a));
        %Iter = Iter + 1;
        if f(a)*f(x0)<0
            b=x0;
        else a=x0;
        end
        err=abs(f(a)*(b-a)/(f(b)-f(a)));
        % affichage des resultats
        if err<eps
             break;
        end
  fprintf(' i=%d \t x0=%2.12f \t err=%f \n',i,x0,err);
end
```

2- Avec la boucle 'while ... end' :

```
% la méthode de fausse position (regula falsi)
% f = x-exp(sin(x)) in [a b].
% la boucle while -- end.
clear all; close all; clc;
tic;
f=inline('x-exp(sin(x))');
a=0; limr=a;
b=10; liml=b;
eps=10^-10;
i=0;
x0=a;
err=abs(a-b);
% le programme principal
    while (err>eps)
        x0=a-f(a)*(b-a)/(f(b)-f(a));
        i= i + 1;
        if f(a)*f(x0)<0
            b=x0;
        else a=x0;
        end
        err=abs(f(a)*(b-a)/(f(b)-f(a)));
        % affichage des resultats
  fprintf(' i=%d \t x0=%2.12f \t err=%f \n',i,x0,err);
        % affichage graphique
        fplot(f,[limr liml]);
        hold all;
        plot(x0,f(x0),'ro');
        xlabel('x'),ylabel('f');
title(['iteration N°=',int2str(i),'  x=',num2str(x0),]);
        hold off;
        grid on;
        pause(1)
    end
toc;
```

6.5.2 Programme en C

```
   #include<stdio.h>
#include<conio.h>
#include<math.h>
float f(float x)
{
```

```
return(2*x-log10(x)-6);
}
void main()
{
float x,x1,a,b,err=0.00005;
int itr=1,n;
clrscr();
printf("enter the values of a,b and maximum iterations \n");
scanf("%f%f%d",&a,&b,&n); x=a-(((b-a)/(f(b)-f(a)))*f(a));
printf("iteration %d=%f \n",itr,x);
itr++;
while(itr<n)
{
if(f(a)*f(x)<0)
b=x;
else
a=x;
x=a-(((b-a)/(f(b)-f(a)))*f(a));
printf("iteration %d   x=%f ",itr,x);
if(fabs(x1-x)<err)
{
printf(" \n after %d iterations, the value of root is %f \n",itr,x);
break;
}
else
itr++;
x1=x;
}
if(itr==n)
printf("\n solution does not exist as iterations are not sufficient");
getch();
return 0;
}
```

Bibliographie

[1]. M. Mokhtari et A. Mesbah : 'Apprendre et Maîtriser MATLAB' Springer (1998).

[2]. Chokri Bekkey et Zouhaier Helali : 'Résolution numérique de l'équation $f(x) = 0$', Projet Tempus, Action JEP-31147 (2003).

[3]. A. Bjorck : 'Numerical Methods for Least Squares Problems'. SIAM. Philadelphia, PA, (1996).

[4]. Najed Ksouri : 'Méthodes d'approximation numérique pour le pricing des options vanilles et asiatiques dans le modèle de Heston de volatilité stochastique'. [Rapport Technique] 2007. https ://hal.inria.fr/inria-00157141v1.

[5]. Eric Goncalvès Da Silva : 'Méthodes et Analyse Numériques'. Engineering school. Institut Polytechnique de Grenoble, 2007, pp.99.
https ://cel.archives-ouvertes.fr/cel-00556967.

[6]. M. Gilli : 'Méthodes numériques'. Département d'économétrie, Université de Genève, version de l'ouvrage (Mars 2006).

[7]. S. Drapier et R. Fortunier : 'Méthodes numériques d'approximation et de résolution en mécanique' , École Nationale Supérieure des Mines de Saint-Étienne 158, cours Fauriel ; 42023 Saint-Étienne Cedex2, Janvier 2015-Version-β.

[8]. R.L. Burden and J. Douglas Faires : 'Numerical Analysis', Ninth Edition, Library of Congress Control Number : 2010922639, Boston-USA (2011).

Webographie

[9]. http ://informatique.coursgratuits.net/methodes-numeriques/recherche-des-racines.php.

[10]. http ://lmah.univ-lehavre.fr/ alaoui/livre-MN-Java.pdf

[11]. http ://www.math.sciences.univ-nantes.fr/ saad/coursanume.pdf

[12]. http ://www.fsr.ac.ma/cours/maths/bernoussi/CoursMethodesNumeriques-2011.pdf

[13]. http ://www.hach.ulg.ac.be/cms/system/files/Intro%20aux%20m%C3%A9thode