Chapitre[1] Résolution d'équations non-linéaires

SAMIR KENOUCHE - DÉPARTEMENT DES SCIENCES DE LA MATIÈRE - UMKB

MÉTHODES NUMÉRIQUES ET PROGRAMMATION
VERSION ACTUALISÉE LE 19/09/2019

Résumé

L'objectif des méthodes présentées dans ce chapitre est la résolution d'équations de la forme $f(x = x^*) = 0$. Nous aborderons successivement les méthodes du point fixe, de dichotomie, de Newton et de la Sécante. Bien évidemment, la liste des méthodes numériques présentées ici est loin d'être exhaustive, sont présentées les méthodes les plus couramment utilisées en graduation, particulièrement en deuxième année Licence des filières Physique et Chimie. Ces méthodes numériques seront mises en œuvre par le biais du logiciel Matlab®. Soulignons par ailleurs que l'utilité d'un algorithme se mesure au moins suivant deux critères qui sont, la rapidité de convergence vers la solution approchée et la précision par rapport aux erreurs (erreurs d'arrondi et de troncature) inhérentes au calcul numérique. A cet effet et après un texte introductif, la première section est consacrée à l'étude succincte du taux de convergence. La dernière section porte sur le travail que doivent réaliser les étudiants (es) lors des séances de travaux pratiques.

Mots clés

Point fixe, dichotomie, méthode de Newton, sécante, scripts Matlab®.

Table des matières

I. INTRODUCTION

Il existe toute une panoplie de méthodes numériques (dichotomie, point fixe, *Newton*, *Lagrange*, ..., etc) conduisant à chercher numériquement les zéros de fonction $f(x) = 0$ d'une variable réelle. La majorité de ces méthodes sont itératives. En d'autres mots, elles calculent des approximations successives $x^{(1)}, x^{(2)}, x^{(3)}, ...$ de la véritable racine x^* de l'équation $f(x = x^*) = 0$, à partir d'une valeur initiale x_0 plus au moins bien choisie. Ce qui les distingue, entre autre, c'est leur vitesse de convergence et leur robustesse. Dans certaines applications, cette vitesse de convergence devient un facteur déterminant notamment quand il s'agit de calculer les racines d'une succession de fonctions.

S. Kenouche est docteur en Physique de l'Université des Sciences et Techniques de Montpellier et docteur en Chimie de l'Université A. Mira de Béjaia.
Site web : voir http://www.sites.univ-biskra.dz/kenouche
Document corrigé, amélioré et actualisé le 19.09.2019.

A. *Convergence*

L'étude de la convergence d'une méthode numérique est conduite à travers la suite des itérés $\left\{x^{(k)}\right\}_{k\in\mathbb{N}}$ générés par l'algorithme. Ce dernier est dit convergent si pour tout $x^{(0)} \in \mathbb{R}$ nous avons :

$$\lim_{k\to+\infty} \parallel x^{(k)} - x^{*} \parallel = 0 \tag{1}$$

Avec $x^{*} \in \mathbb{R}$ est la solution approchée de la valeur exacte, déterminée avec une tolérance fixée préalablement. La condition (1) garantit, à partir d'une certaine itération, la satisfaction du critère d'arrêt pour la tolérance exigée. Nous rappelons que le taux de convergence (ou "vitesse" de convergence) et la complexité du problème traité rentrent en ligne de compte lors de l'utilisation d'un algorithme d'optimisation. La convergence ne signifie pas systématiquement l'existence d'une solution au problème. Le schéma numérique adopté doit être à la fois rapide et robuste. Ces deux derniers critères sont contrôlés par l'étude du taux de convergence qui quantifie l'erreur [1] commise à chaque itération, soit :

$$e^{(k)} = \parallel x^{(k)} - x^{*} \parallel \quad \text{tel que} \quad f(x^{*}) = 0 \tag{2}$$

L'estimation de l'erreur servira, entre autre, à comparer le taux de convergence des différentes méthodes numériques. En pratique, l'erreur est représentée graphiquement en traçant la norme de l'erreur, soit $\parallel e^{(k+1)} \parallel$ en fonction de $\parallel e^{(k)} \parallel$ avec une échelle logarithmique. Ainsi, *l'ordre* noté p, de la méthode s'obtient à partir de :

$$\parallel e^{(k+1)} \parallel \approx A \parallel e^{(k)} \parallel^{p} \Longrightarrow \log \parallel e^{(k+1)} \parallel \approx p \log \parallel e^{(k)} \parallel + \text{cste} \tag{3}$$

Ainsi l'ordre, p, est quantifié via la pente de l'équation ci-dessus. Il en ressort les conclusions suivantes :

1) Si $p = 1 \Longrightarrow x^{(k)}$ converge linéairement vers la solution approchée. De cette façon nous gagnons la même quantité de précision à chaque itération. Dans ce cas, nous avons :

$$\lim_{k\to+\infty} = \frac{\parallel x^{(k+1)} - x^{*} \parallel}{\parallel x^{(k)} - x^{*} \parallel} = \tau \quad \text{avec} \quad 0 < \tau < 1 \tag{4}$$

   Elle est dite superlinéaire si $\tau = 0$.
2) Si $p = 2 \Longrightarrow x^{(k)}$ converge quadratiquement vers la solution approchée. De cette façon, nous gagnons, le double de précision à chaque itération.
3) Si $p = 3 \Longrightarrow x^{(k)}$ converge cubiquement vers la solution approchée. Dans ce cas on gagne le triple de précision à chaque itération. Dans ce cas, nous avons :

$$\lim_{k\to+\infty} = \frac{\parallel x^{(k+1)} - x^{*} \parallel}{\parallel x^{(k)} - x^{*} \parallel^{p}} = \tau \quad \text{avec} \quad \tau \geq 0 \tag{5}$$

D'un point de vue pratique et pour un k suffisamment élevé, la vitesse de convergence d'une méthode itérative est évaluée "empiriquement" au moyen de la relation :

$$K_p(x,k) = \frac{\parallel x^{(k+2)} - x^{(k+1)} \parallel}{\parallel x^{(k+1)} - x^{(k)} \parallel^{p}} \tag{6}$$

Bien évidemment, nous visons à ce que la convergence de algorithme soit la plus élevée possible afin de tendre vers la solution en un minimum d'itérations pour la tolérance exigée. La convergence quadratique est plus rapide

1. Le terme erreur n'est pas considéré au sens de faute. Cela fait référence aux erreurs inéluctables propres aux calculs par ordinateur. Cela concerne, entre autre, les erreurs d'arrondi et de discrétisation. Ainsi, l'erreur commise est majorée par $\parallel \frac{x-x^{*}}{x} \parallel \leq \frac{b^{1-n}}{2}$. Où x est la valeur exacte, x^{*} sa représentation par l'arithmétique de l'ordinateur, b est la base du système de numération et n représente le nombre de chiffres significatifs. Notons aussi que la *stabilité* et le *conditionnement* sont les deux critères pris en compte afin d'analyser la propagation des erreurs d'arrondi dans le calcul numérique.

que celle superlinéaire. A son tour, cette dernière, est plus rapide que la convergence linéaire. Tenant compte de l'équation (6), il vient que plus $K_p(X,k)$ tend vers zéro plus le taux de convergence de la méthode est élevé.

B. Critères d'arrêt

Les méthodes numériques, dites locales, procèdent de façon itérative. Typiquement, étant donné une valeur initiale, un nouvel itéré est mis à jour afin de converger vers la solution recherchée. Ce processus est réitéré jusqu'à la satisfaction d'un ou plusieurs critères d'arrêt de l'algorithme. D'un point de vue purement théorique, le schéma itératif est infini. En pratique, la suite d'approximations successives est tronquée dés que l'on considère avoir atteint la précision requise. Étant donné une tolérance ϵ, les critères d'arrêt pouvant êtres envisagés sont :

$$\| x^{(k+1)} - x^{(k)} \| < \epsilon \tag{7}$$

$$\| f(x^{(k)}) \| < \epsilon \tag{8}$$

$$\frac{\| f(x^{(k+1)}) - f(x^{(k)}) \|}{\| f(x^{(k)}) \|} < \epsilon \tag{9}$$

Nous pouvons envisager un quatrième critère, celui du nombre d'itérations dépassant un seuil fixé préalablement. Il se peut que le critère (9) ne soit pas satisfait même si l'algorithme converge. Les erreurs d'arrondis dues à l'accumulation des opérations arithmétiques peuvent êtres du même ordre de grandeur que le gain de précision obtenu à l'itération en cours. Le critère (7) est recommandé, le schéma itératif est interrompu lorsqu'il ne produit plus de gain significatif en terme de précision. En outre, dans certaines situations la divergence d'un algorithme ne signifie pas forcément l'inexistence de la solution, il faudra juste adapter le nombre d'itérations et/ou la tolérance considérés. Pour éviter que la boucle tourne indéfiniment quand il n'y a pas de convergence. Il est indispensable d'ajouter systématiquement un critère limitant le nombre d'itérations. D'un point de vue purement pratique, une combinaison de ces critères est prise en compte.

II. MÉTHODE DU POINT FIXE

Le principe de la méthode du *point fixe* consiste à transformer la fonction $f(x) = 0$ tel que $f : [a \quad b] \longrightarrow R$, en une fonction $\varphi(x) = x$. La fonction $\varphi : [a \quad b] \longrightarrow R$, est construite de façon à ce que $\varphi(x^*) = x^*$ quand $f(x^*) = 0$. Trouver la racine de $f(x)$ se résume donc à déterminer un $x^* \in [a \quad b]$ tel que $x^* = \varphi(x^*)$. Dans le cas où un tel point existe, il sera qualifié de *point fixe* de φ et cette dernière est dite *fonction d'itération*. Le schéma numérique de cette méthode est donné par :

$$x^{(k+1)} = \varphi(x^{(k)}) \quad \forall \quad k \geq 0 \tag{10}$$

a) ***Définition****:* Une application f est k-contractante alors $\exists\, q \in [0\,,\,1[$ tel que $\forall (x,y)\, \mathcal{D}^2$ nous avons :

$$\| f(x) - f(y) \| \leq q \ \| x - y \| \tag{11}$$

b) ***Théorème****:* $f : \mathcal{D} \subset \mathbb{R} \longmapsto \mathbb{R}$ est une application k-contractante alors f possède un point fixe unique $\forall x^{(0)} \in \mathcal{D}$ et la suite des itérés $\left\{x^{(k)}\right\}_{k\in\mathbb{N}}$ converge vers ce point fixe.

Il est possible de démontrer ce théorème en procédant par l'absurde. Soient x et y deux points fixes de f. Puisque cette dernière est k-contractante alors $\| f(x) - f(y) \| \leq q \ \| x - y \|$, x et y sont deux points fixes de f alors $f(x) = x$ et $f(y) = y$ il s'en suit :

$$\| f(x) - f(y) \| \leq q \ \| x - y \| \Rightarrow \| x - y \| - q \ \| x - y \| \leq 0$$

$$\Rightarrow \| x - y \| (1 - q) \leq 0$$

Avec $q \in [0\,,\,1[$ il en découle

$$\| x - y \| = 0 \Leftrightarrow x = y$$

Ainsi si f est k-contractante alors f admet un point fixe unique. Une autre façon de démontrer le théorème consiste à démontrer que $\left\{x^{(k)}\right\}_{k\in\mathbb{N}}$ est une suite de Cauchy [2]. Soit $k \in \mathbb{N}$, nous avons :

$$\| f(x^{(k)}) - f(x^{(k-1)}) \| \leq q \ \| x^{(k)} - x^{(k-1)} \|$$

$$\Rightarrow \quad \| x^{(k+1)} - x^{(k)} \| \leq q \ \| x^{(k)} - x^{(k-1)} \| \tag{12}$$

D'un autre côté, nous pouvons écrire

$$\| x^{(k)} - x^{(k-1)} \| \leq q \ \| x^{(k-1)} - x^{(k-2)} \| \tag{13}$$

En substituant (13) dans (14) nous obtenons :

$$\Rightarrow \quad \| x^{(k+1)} - x^{(k)} \| \leq q^2 \ \| x^{(k-1)} - x^{(k-2)} \| \tag{14}$$

En procédant par récurrence nous démontrons [3] :

$$\Rightarrow \quad \| x^{(k+1)} - x^{(k)} \| \leq q^k \ \| x^{(1)} - x^{(0)} \| \tag{15}$$

Avec $q \in [0\,,\,1[$, la distance $\| x^{(k+1)} - x^{(k)} \|$ tend vers zéro pour k assez grand alors $\left\{x^{(k)}\right\}_{k\in\mathbb{N}}$ est une suite de Cauchy donc elle est convergente. Comme exercice d'application, montrer que la fonction $f(x) = \dfrac{x+1}{x+2}$ est k-contractante dans l'intervalle $[0\,,\,+\infty[$. Cet exercice sera résolu lors de la séance de cours.

Par ailleurs, il est aussi possible de démontrer que f est k-contractante en se servant de *l'inégalité des accroissements finies* qui stipule : si $\forall x \in \mathcal{D} \subset \mathbb{R}, \| f'(x) \| < 1 \Rightarrow \forall (x,y) \in \mathcal{D}^2$ alors $\| f(x) - f(y) \| \leq q \ \| x - y \|$. Pour que f soit k-contractante il suffit de démontrer que sa dérivée $\| f'(x) \|$ est majorée. En utilisant cette définition, montrer que $f(x) = \sin(x)$ est k-contractante. Cet exercice sera également résolu lors de la séance de cours

Exercice ❶ ☞ ®

A partir du formalisme de la physique statistique et dans l'objectif de compléter l'équation d'état des gaz parfaits, **Johannes Diderik van der waals** en 1873 a développé une équation d'état décrivant des gaz réels. Cette dernière prend en compte l'interaction mutuelle entre molécules du gaz dont la formule est :

$$\left(P + a\left(\frac{N}{V}\right)^2\right)(V - Nb) = k_B\, N\, T \tag{16}$$

- Évaluer numériquement par la méthode du *point fixe*, le volume V occupé par $N = 10^3$ molécules de CO_2 dans les conditions de température $T = 300\,K$ et de pression $P = 3.5\,10^7\,Pa$. Pour ce gaz nous avons les constantes suivantes : $a = 0.40\,Pa\,m^3$, $b = 42.70\,10^{-6}\,m^3$ et $k_B = 1.38\,10^{-23}\,J/K$. Prendre une valeur initiale $V_0 = 0.2\,m^3$. Il faudra donc résoudre l'équation non linéaire $\varphi(V) = 0$ dont la racine est V^*.

- Réaliser un test d'arrêt pour une tolérance $\epsilon = 10^{-3}$.
- Donner le nombre d'itérations permettant la satisfaction du test d'arrêt.
- Écrire le script Matlab® correspondant.

2. Rappelons que toute suite convergente dans $\mathbb{R}$ est suite de Cauchy.
3. Pour les puristes, nous opterons plutôt pour $\| x^{(k+1)} - x^{(k)} \| \leq \dfrac{q^k}{1-q} \ \| x^{(1)} - x^{(0)} \|$

*c) **Solution**:* nous cherchons V^* tel que $\varphi(V = V^*) = 0$. Le schéma numérique de la méthode du *point fixe* impose :

$$V^{(k+1)} = \varphi(V^{(k)})$$

Cherchons $\varphi(V)$:

$$P\,V + a\,\frac{N^2}{V} - \frac{a\,b\,N^3}{V^2} - \underbrace{P\,N\,b - k_B\,N\,T}_{\text{const}} = 0$$

$$V = -\frac{a\,\frac{N^2}{V} - \frac{a\,b\,N^3}{V^2} - \text{const}}{P} \Rightarrow \varphi(V) = -\frac{a\,\frac{N^2}{V} - \frac{a\,b\,N^3}{V^2} - \text{const}}{P}$$

$$\begin{aligned} V^{(1)} &= \varphi(0.2000) = -0.002 \\ V^{(2)} &= \varphi(-0.002) = 102.14 \\ V^{(3)} &= \varphi(102.14) = 0.0426 \end{aligned}$$

Le test d'arrêt pour une tolérance $\epsilon = 10^{-3}$.

$$\begin{aligned} Err^{(1)} &= |V^{(1)} - \varphi(V^{(1)})| = 102.15 \\ Err^{(2)} &= |V^{(2)} - \varphi(V^{(2)})| = 102.09 \\ Err^{(3)} &= |V^{(3)} - \varphi(V^{(3)})| = 8.16 \times 10^{-4} < \epsilon \end{aligned}$$

La racine recherchée $\Rightarrow V^* = V^{(3)} = 0.0426\,cm^3$. Le nombre d'itérations permettant la satisfaction du test d'arrêt est égale à **3** itérations, c'est le dernier itéré.

Voici le script Matlab® :

```
clc ; clear all ;
% Samir KENOUCHE - Le 05/09/2019
% METHODE DU POINT FIXE APPLIQUEE A L'EQUATION D'ETAT DES GAZ REELS

p = 3.5*10.^7 ; N = 1000 ; T = 300 ; a = 0.40 ; b = 42.7*10^(-6) ;
kb = 1.38*10.^(-23) ; const = p*N*b - kb*N*T ;
x0 = 0.20 ; % VALEUR INITIALE

phi_V = @(V) -((a*N.^2)./V - (a*b*N.^3)./V.^2 - const )./p ;
it = 0 ; err = 1e-03 ; Nbrit = 10 ;

while it < Nbrit

    xk = phi_V(x0) ;

     if abs(xk - phi_V(xk)) < err  % TEST D'ARRET
```

```
        volume = xk ;                % LA RACINE TROUVEE
        iteration_max = it ;         % MAXIMUM ITERATION
        break
     end
   x0 = xk ; it = it +1 ;            % MISE A JOUR
end

str1 = strcat('LE VOLUME DU CO2 VAUT (EN m3) : ', num2str(volume))
str2 = strcat(' NOMBRE D''ITERATIONS PERMETTANT LA CV VAUT : ', ...
    num2str(iteration_max))
```

Exercice ❷ ☞ ®

Avant les travaux théoriques de Max Planck sur le rayonnement du corps noir, Wien a démontré empiriquement la formule :

$$\lambda_{\max} T = 2.90 \times 10^{-3} \, [m\, K] \tag{17}$$

Avec $\lambda_{\max}$ correspond au maximum du spectre d'émission du corps noir à la température T. C'est la loi de déplacement de Wien. Par la suite, Max Planck établit la loi du rayonnement thermique sous la forme :

$$d\rho(\lambda, T) = \rho_\lambda(T)\, d\lambda = \frac{8\,\pi\,h\,c}{\lambda^5} \, \frac{d\lambda}{exp\left(\dfrac{h\,c}{k_b\,T\,\lambda}\right) - 1} \tag{18}$$

1) Déduire la formule de Wien (17) à partir de celle de Max Planck (18). On donne $h = 6.62\,10^{-34}\,J\,s$, $k_b = 1.38\,10^{-23}\,J\,K^{-1}$ et $c = 2.99\,10^{8}\,m\,s^{-1}$.

d) ***Solution****:* commençons par poser $x = \dfrac{h\,c}{\lambda\,k_B\,T}$ par conséquent le formule de Planck prend la forme :

$$\rho_\lambda(T) = \frac{8\,\pi\,h\,c}{\lambda^5} \, \frac{d\lambda}{exp\,(x) - 1} \tag{19}$$

Le maximum du profil théorique de la formule de Planck est atteint pour $\dfrac{d\rho_\lambda(T)}{d\lambda}(\lambda = \lambda_{max}) = 0$, c'est un point stationnaire. Ainsi, après dérivation par rapport à la variable λ nous obtenons :

$$\frac{d\rho_\lambda(T)}{d\lambda} = 8\,\pi\,h\,c \left[\frac{-5}{\lambda_{max}^6\,(\exp(x) - 1)} + \frac{x\,\exp(x)}{\lambda_{max}\,(\exp(x) - 1)^2}\right] = 0$$

$$\Rightarrow x\,\exp(x) = 5\,(\exp(x) - 1) \quad \text{avec} \quad x = \frac{h\,c}{\lambda_{max}\,k_B\,T} \tag{20}$$

A ce stade nous utiliserons la méthode du *point fixe* pour trouver x. L'équation (20) peut s'écrire selon le schéma numérique de la méthode en question, soit :

$$x = \underbrace{\frac{5\,(\exp(x) - 1)}{\exp(x)}}_{\varphi(x)} \quad \text{avec} \quad x = \frac{h\,c}{\lambda_{max}\,k_B\,T} \tag{21}$$

$$x^{(1)} = \varphi(2.0000) = 4.3233$$
$$x^{(2)} = \varphi(4.3233) = 4.9337$$
$$x^{(3)} = \varphi(4.9337) = 4.9640$$
$$x^{(4)} = \varphi(4.9640) = 4.9651$$

Le test d'arrêt est satisfait au bout de la quatrième itération. Par conséquent la racine recherchée est $x^* = 4.9651$, il faut juste prendre le dernier itéré, en l'occurrence $x^{(4)}$, vérifiant le critère d'arrêt. Compte tenu de ce dernier résultat, nous en déduisons ce qui suit :

$$\lambda_{max}\, T = \frac{h\, c}{k_B\, 4.9651} = \frac{6.62\, 10^{-34}\, [J\, s]\, 2.99\, 10^{8}\, [m\, s^{-1}]}{1.38\, 10^{-23}\, [J\, K^{-1}]\, 4.9651}$$

D'où la formule de Wien :

$$\Rightarrow \lambda_{max}\, T = 2.89\, 10^{-3}\, [m\, K]$$

Script Matlab® :

```
clc ; clear all ;
% Samir KENOUCHE - le 05/09/2019
% APPLICATION DE LA METHODE DU POINT FIXE

phi_x = @(x) (5.*(exp(x) - 1))./exp(x) ; x0 = 2 ; err = 1e-06 ;
it = 0 ;

while abs(x0  - phi_x(x0)) > err

    xk = phi_x(x0) ;

   iteration_max = it ;          % NOMBRE D'ITERATIONS PERMETTANT LA CV
   x0 = xk  ;  it = it + 1 ;     % MISE A JOUR
end

str1 = strcat('LA SOLUTION VAUT : ', num2str(xk))
str2 = strcat(' NOMBRE D''ITERATIONS PERMETTANT LA CV VAUT : ', ...
    num2str(iteration_max))
```

Exercice ❸ ☞ ®

Dans cet exercice, il est demandé de trouver la racine de la fonction $f_1(x) = x - cos(x)$, en utilisant la méthode du *point fixe*

1) Tracer la fonction $f_1(x)$ sur l'intervalle $[-1/2 \quad 3]$.
2) Écrire un script Matlab® permettant l'implémentation du schéma numérique de cette méthode.
3) Afficher, sur le même graphe, la fonction $f_1(x)$ et la solution approchée.
4) Afficher le graphe représentant le nombre d'approximations successives $x^{(k+1)}$ en fonction du nombre d'itérations.
5) Tracer l'évolution de l'erreur en fonction du nombre d'itérations.
6) Tracer l'erreur $\ln|e_{n+1}|$ en fonction de $\ln|e_n|$ et déterminer l'ordre de la méthode.
7) Calculer sa vitesse de convergence.

Appliquez le même algorithme pour résoudre l'équation : $f_2(x) = x + exp(x) + 1$ avec $x \in [-2 \quad 1/2]$.

On donne : tolérance $= 10^{-6}$, les valeurs initiales $x_0 = 0.8$ pour $f_1(x)$ et $x_0 = -1/5$ pour $f_2(x)$.

Script Matlab®

```
clear all ; close all ; clc ;
%  05/06/2019 Samir KENOUCHE : ALGORITHME PERMETTANT
% L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DU POINT FIXE
x0 = 0.8 ; it = 0 ; tol = 1e-05 ; Nbrit = 30 ;

while it < Nbrit

    it = it + 1 ; x = cos(x0) ;
    xn(it) = x ;   % RECUPERATION DES ITERES

      if abs(x - cos(x)) < tol
         sol = x ;  % LA RACINE
        break
      end
   x0 = x ;        % MISE A JOUR
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE GRAPHIQUE %%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = 0 ; b = 3 ; n = 1000 ; dx = (b - a)/n ; x1 = a:dx:b ;
y = inline('cos(x)') ;

figure('color', [1 1 1]) ;
plot(x1, x1,'k') ; hold on ; plot(x1, y(x1),'LineWidth',2) ;
plot(sol,y(sol),'ro','MarkerSize',12,'LineWidth',2) ;
plot(sol,y(sol) ,'rx','MarkerSize',12,'LineWidth',2) ;

plot(sol,0,'ko','MarkerSize',12,'LineWidth',2) ;
plot(sol,0 ,'kx','MarkerSize',12,'LineWidth',2) ;
line(x1, zeros(1, length(x1)),'LineStyle','-.','Color','k','LineWidth',1) ;
xlabel('x') ; ylabel('f(x)') ;

line(sol.*ones(1, length(x1)), y(x1),'LineStyle','-.','Color','k', ...
    'LineWidth',1) ; axis([0 3 -1 1.5]) ;

%%%%%%%%%%%%%%%%%%%%%%%%%% ECRITURE DANS LA FIGURE %%%%%%%%%%%%%%%%%%%%%%%%%%
text('Interpreter','latex',...
 'String','$ x = cos(x) $','Position',[2 -1/3],'FontSize',15)
text('Interpreter','latex','String','$ y = x $',...
 'Position',[1.2 1],'FontSize',15) ; text(sol,2*sol, ...
 ['\alpha = ',num2str(sol)],'Position',[0.8 -1/4],'BackgroundColor',[1 1 1]);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% ORDRE DE LA METHODE %%%%%%%%%%%%%%%%%%%%%%%%%%%%
err = abs(xn - sol); error_1 = err(1:end-1) ; error_2 = err(2: end);

figure('color', [1 1 1]) ; loglog(error_1(1:end-4),error_2(1:end-4),'+')
ylabel('Ln |e_{n+1}|') ;  xlabel('Ln |e_{n}|')

cutoff = 7 ;  % ATTENTION AU CHOIX DE LA NIEME ITERATION
```

```
pente = (log(error_2(end-cutoff)) - log(error_2(end-(cutoff+1))))/(log(error_1
   (end-cutoff)) - log(error_1(end-(cutoff+1)))); % VITESSE DE CONVERGENCE log
   |en+1| = log|en|
pente = round(pente);
```

```
%%%%%%%%%%%%%%%%%%%%% VITESSE DE CONVERGENCE (CV) %%%%%%%%%%%%%%%%%%%%%%%%%
n = 20;  % ATTENTION AU CHOIX DE LA NIEME ITERATION
vitesse_CV = (xn(n+2) - xn(n+1))/(xn(n+1) - xn(n));

msg1 = gtext(strcat('CETTE METHODE EST D''ORDRE : ', num2str(pente)));
% CLIQUER SUR LA FIGURE POUR AFFICHER msg1

msg2 = gtext(strcat('LA VITESSE DE CONVERGENCE VAUT : ',num2str(vitesse_CV)));
% CLIQUER SUR LA FIGURE POUR AFFICHER msg2
```

Les résultats des différents calculs, renvoyés par le script, sont portés sur les figures ci-dessous

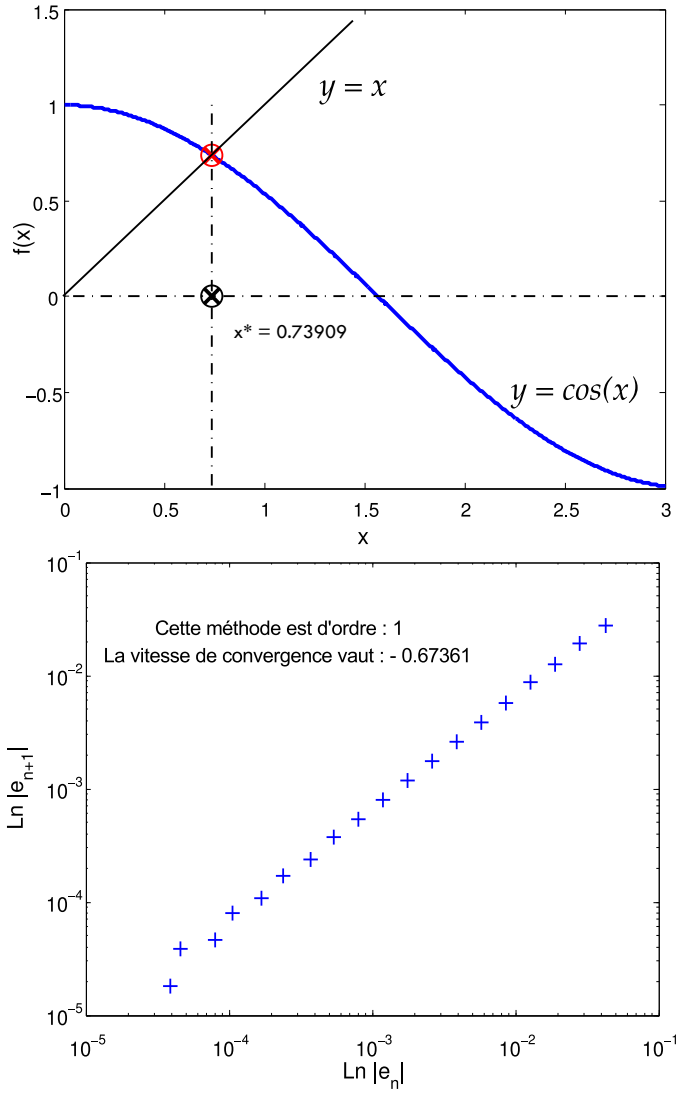

FIGURE 1: Racine de f obtenue par la méthode du *point fixe*

Notons que l'inconvénient de la méthode du *point fixe*, est qu'elle converge trop lentement. Afin d'améliorer sa convergence on peut la transformer en une nouvelle suite par le biais de l'algorithme d'accélération de convergence. Parmi ces méthodes d'accélération de la convergence, on abordera les méthodes de *relaxation* et d'*Aitken-Shanks*.

A. *Méthode de relaxation*

L'accélération de la convergence consiste à remplacer la suite originale $\{x^{(k)}\}_{k\in\mathbb{N}}$ par une nouvelle suite $\{\tilde{x}^{(k)}\}_{k\in\mathbb{N}}$ convergeant plus rapidement vers la solution x^*. Dans le cas de la méthode du point fixe, la méthode de *relaxation* impose l'écriture de (10) sous la forme :

$$\alpha\, x + x = \varphi(x) + \alpha\, x \quad \forall \alpha \in \mathbb{R} \tag{22}$$

$$\Rightarrow \quad (\alpha+1)\, x = \varphi(x) + \alpha\, x \;\Rightarrow\; x = \underbrace{\frac{\varphi(x) + \alpha\, x}{1+\alpha}}_{G(x)} \tag{23}$$

D'où la nouvelle suite d'itérés :

$$x^{(k+1)} = \underbrace{\frac{\varphi(x^{(k)}) + \alpha\, x^{(k)}}{1+\alpha}}_{G(x^{(k)})} \tag{24}$$

Où α est le paramètre de relaxation, ce dernier est choisi le près possible de $-\varphi^{'}(x^*)$. Cette suite convergera dès-que :

$$\parallel G^{'}(x^*) \parallel = \parallel \frac{\varphi^{'}(x) + \alpha}{\alpha + 1} \parallel < 1 \tag{25}$$

En outre, elle convergera d'autant plus rapidement que $\parallel G^{'}(x^*) \parallel$ tend vers zéro. Ceci peut être démontré comme suit, développons en série de Taylor au premier ordre la fonction $\varphi(x)$ et posons $\epsilon^{(k)} = x^{(k)} - x^*$:

$$x^{(k+1)} = \varphi(x^{(k)}) = \varphi(\epsilon^{(k)} + x^*) = \varphi(x^*) + \varphi^{'}(x^*)\, \epsilon^{(k)} + O(\epsilon^{(n)}) \tag{26}$$

$$\Rightarrow \quad x^{(k+1)} - \varphi(x^*) = \underbrace{x^{(k+1)} - x^*}_{\epsilon^{(k+1)}} = \varphi^{'}(x^*)\, \epsilon^{(k)} \tag{27}$$

$$\Rightarrow \quad \epsilon^{(k+1)} = \varphi^{'}(x^*)\; \epsilon^{(k)} \tag{28}$$

D'après (28) plus $\varphi^{'}(x^*)$ est faible, ie : $0 \leq \varphi^{'}(x^*) < 1$ plus la convergence est rapide. Comme *exercice d'application*, trouver la racine de la fonction :

$$f(x) = \exp(-x) - x \tag{29}$$

Prendre une valeur initiale $x^{(0)} = 1/2$. Écrivons (29) selon le schéma itératif de la méthode du point fixe, soit :

$$x^{(k+1)} = \varphi(x^{(k)}) = \exp(-\,x^{(k)}) \tag{30}$$

$$\Rightarrow \quad x^{(1)} = \varphi(1/2) = 0.606 \simeq 0.61$$
$$\Rightarrow \quad x^{(2)} = \varphi(0.61) = 0.543 \simeq 0.54$$
$$\Rightarrow \quad x^{(3)} = \varphi(0.54) = 0.5827 \simeq 0.58$$

Les erreurs correspondantes,

$$\Rightarrow \quad \epsilon^{(1)} = \| \, 0.61 - 0.50 \, \| = 0.11$$

$$\Rightarrow \quad \epsilon^{(2)} = \| \, 0.54 - 0.61 \, \| = 0.07$$

$$\Rightarrow \quad \epsilon^{(3)} = \| \, 0.58 - 0.54 \, \| = 0.04$$

Déjà à ce stade, nous constatons une convergence lente de l'algorithme. Dans ce qui suit, nous utiliserons la méthode de *relaxation* afin d'accélérer la convergence. Pour cela, on générera une nouvelle suite $\{\tilde{x}^{(k)}\}_{k\in\mathbb{N}}$ tel que $\tilde{x}^{(k+1)} = G(\tilde{x}^{(k)})$. Évaluons d'abord la valeur de α, nous avons déjà écrit $\alpha \simeq \| \, \varphi^{'}(x^*) \, \| = \| \, \exp(-0.54) \, \| \simeq 0.58$. Comme on est libre de choisir le paramètre α, nous l'avons pris proche de x^*. Ce qui donne la nouvelle suite :

$$\tilde{x}^{(k+1)} = G(\tilde{x}^{(k)}) = \frac{\exp(-\tilde{x}^{(k)}) + 0.58\,\tilde{x}^{(k)}}{1.58} \tag{31}$$

Désormais nous souhaitons calculer la racine en question avec six chiffres significatifs, avec $\tilde{x}^{(0)} = 1/2$:

$$\tilde{x}^{(1)} = G(\tilde{x}^{(0)}) = \frac{\exp(-\tilde{x}^{(0)}) + 0.58\,\tilde{x}^{(0)}}{1.58} = 0.5670567$$

$$\tilde{x}^{(2)} = G(\tilde{x}^{(1)}) = \frac{\exp(-\tilde{x}^{(1)}) + 0.58\,\tilde{x}^{(1)}}{1.58} = 0.5671426$$

$$\tilde{x}^{(3)} = G(\tilde{x}^{(2)}) = \frac{\exp(-\tilde{x}^{(2)}) + 0.58\,\tilde{x}^{(2)}}{1.58} = 0.5671433$$

$$\tilde{x}^{(4)} = G(\tilde{x}^{(3)}) = \frac{\exp(-\tilde{x}^{(3)}) + 0.58\,\tilde{x}^{(3)}}{1.58} = 0.5671435$$

B. Aitken-Shanks

Ce procédé d'accélération de la convergence est une variante de la méthode de *Romberg-Richardson*. Une suite $\{x^{(k)}\}_{k\in\mathbb{N}}$ convergente vers x^*, vérifie :

$$\| \, x^{(k+1)} - x^* \, \| = k \, \| \, x^{(k)} - x^* \, \| \tag{32}$$

Ainsi il est possible d'écrire

$$\| \, x^{(k+1)} - x^* \, \| = k \, \| \, x^{(k)} - x^* \, \| \tag{33}$$

$$\| \, x^{(k+2)} - x^* \, \| = k \, \| \, x^{(k+1)} - x^* \, \| \tag{34}$$

Par soustraction des équations (33) et (34), nous obtenons :

$$\| \, x^{(k+2)} - x^{(k+1)} \, \| = k \, \| \, x^{(k+1)} - x^{(k)} \, \| \tag{35}$$

De (33) il vient :

$$x^* = \frac{x^{(k+1)} - k\,x^{(k)}}{1-k} = x^{(k)} + \frac{x^{(k+1)} - x^{(k)}}{1-k} \tag{36}$$

Substituant (35) dans (36),

$$x^* = x^{(k)} + \frac{x^{(k+1)} - x^{(k)}}{\dfrac{x^{(k+1)} - x^{(k)} - x^{(k+2)} + x^{(k+1)}}{x^{(k+1)} - x^{(k)}}} \tag{37}$$

$$x^* = x^{(k)} + \frac{(x^{(k+1)} - x^{(k)})^2}{2\,x^{(k+1)} - x^{(k+2)} - x^{(k)}} \tag{38}$$

En notation des différences finies :

$$\Delta x^{(k)} = \left[x^{(k+1)} - x^{(k)}\right] \tag{39}$$

$$\Delta^2 x^{(k)} = -(x^{(k+1)} - x^{(k+2)} - x^{(k)}) = \underbrace{\left[x^{(k+2)} - x^{(k+1)}\right]}_{\Delta x^{(k+1)}} - \underbrace{\left[x^{(k+1)} - x^{(k)}\right]}_{\Delta x^{(k)}} \tag{40}$$

Finalement le schéma itératif de l'accélération de la convergence est donné par :

$$x^* = x^{(k)} - \frac{\left[\Delta x^{(k+1)}\right]^2}{\Delta^2 x^{(k)}} \tag{41}$$

Ce schéma itératif est appliqué selon l'algorithme de *Steffensen*, pour la méthode du point fixe, selon :

$$x^{(k+1)} = x^{(k)} - \frac{(a^{(k)} - x^{(k)})^2}{b^{(k)} - 2\,a^{(k)} + x^{(k)}} \quad \text{avec} \quad a^{(k)} = \varphi(x^{(k)}) \quad \text{et} \quad b^{(k)} = \varphi(a^{(k)}) \tag{42}$$

Nous avons appliqué cette méthode d'accélération au problème ❸ précédent. Ci-dessous le script Matlab® :

```
clear all ; close all ; clc ;

% 05/09/2019 Samir KENOUCHE : ALGORITHME PERMETTANT
% L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE Aitken-Shanks POUR LA
% FONCTION F(X) = X - COS(X)


xinit = 0.8 ; it = 1 ; tol = 1e-05 ; Nbrit = 30 ;


while it < Nbrit

      ak = cos(xinit) ; bk = cos(ak) ;
      x = xinit - ((ak - xinit).^2)/(bk - 2*ak + xinit) ;

      if abs(x - cos(x)) < tol

         solution = x ;           % LA RACINE
         iteration_max = it ;
         break

      end

      xinit = x ; it = it + 1 ;  % MISE A JOUR
end
```

À titre comparatif et tenant compte des deux scripts Matlab® du problème ❸, la méthode du point fixe converge vers la racine approchée `sol = 7.390850858357935e-01`, au bout de 24 itérations. Alors que celle de *Aitken-Shanks* converge vers la même solution au bout de 2 itérations seulement.

Exercice ❹ ☞ Ⓢ

Soit la fonction :

$$f(x) = e^x - x \tag{43}$$

1) En utilisant la méthode du *point fixe*, trouver numériquement le zéro de la fonction $f(x)$, l'ordre de la méthode ainsi que la vitesse convergence de la suite $x^{(k+1)} = e^{x^{(k)}}$. A cet effet, prendre $x_0 = 2.50$ et $\epsilon = 10^{-3}$.
2) Écrire le script Matlab® correspondant en déterminant le nombre d'itérations permettant la convergence de l'algorithme.

Exercice ❺ ☞ Ⓢ

1) En utilisant les méthodes du *point fixe* et de *Aitken-Shanks*, trouver la racine des fonctions :

$$\begin{cases} f_1(x) = x - x^3, & x \in [0, 1] \\ f_2(x) = x - \exp(-x), & x \in [-1, 1] \\ f_3(x) = x - x^{4/5} + 2, & x \in [-2, 2] \\ f_2(x) = x - \exp(-x) - 4, & x \in [0, 5] \end{cases} \tag{44}$$

2) Tracer, sur la même figure, la fonction et sa racine calculée.
3) Afficher le graphe représentant le nombre d'approximations successives $x^{(k+1)}$ en fonction du nombre d'itérations.
4) Tracer l'erreur $ln|e_{n+1}|$ en fonction de $ln|e_n|$ et déterminer l'ordre de la méthode.
5) Calculer la vitesse de convergence des deux méthodes. Conclure.

III. MÉTHODE DE DICHOTOMIE

Le principe de la méthode de *dichotomie*, encore appelée méthode de *bissection*, est basé sur le théorème de la valeur intermédiaire. Son fondement est régit par le théorème de *Bolzano* stipulant :

e) ***Théorème****:* soit $f : \mathcal{D} \subset \mathbb{R} \longmapsto \mathbb{R}$ une fonction continue sur $\mathcal{D} = [a\,;\, b]$ est change de signe $f(a) \times f(b) < 0 \Rightarrow f$ admet au moins une racine $x^* \subset \mathcal{D}$ telle que $f(x = x^*) = 0$.

La méthode est décrite comme suit : soit, $f : [a \quad b] \longrightarrow R$, une fonction continue sur l'intervalle $[a \quad b]$. Si $f(a) \times f(b) < 0 \longrightarrow$ il existe donc au moins une racine de $f(x)$ appartenant à l'intervalle $[a \quad b]$. On prend $c = \dfrac{a+b}{2}$ la moitié de l'intervalle $[a \quad b]$ tel que :

1) Si $f(c) = 0 \longrightarrow c$ est la racine de $f(x)$.
2) Sinon, nous testons le signe de $f(a) \times f(c)$ (et de $f(c) \times f(b)$).
3) Si $f(a) \times f(c) < 0 \longrightarrow$ la racine se trouve dans l'intervalle $[a \quad c]$ qui est la moitié de $[a \quad b]$.
4) Si $f(c) \times f(b) < 0 \longrightarrow$ la racine se trouve dans l'intervalle $[c \quad b]$ qui est la moitié de $[a \quad b]$.

Ce processus de division, par deux, de l'intervalle (à chaque itération on divise l'intervalle par deux) de la fonction est réitéré jusqu'à la convergence pour la tolérance considérée. Ainsi, pour la nième itération, on divise : $[a_n \quad b_n]$ en $[a_n \quad c_n]$ et $[c_n \quad b_n]$, avec à chaque fois $c_n = \dfrac{a_n + b_n}{2}$.

f) ***Erreur de la méthode****:* $\forall k \in \mathbb{N}$, nous avons :

$$b_k - a_k = \frac{b_{k-1} - a_{k-1}}{2} = \frac{b_{k-2} - a_{k-2}}{2^2} = \frac{b_{k-3} - a_{k-3}}{2^3} = \cdots\cdots = \frac{b_0 - a_0}{2^k} \tag{45}$$

L'algorithme de dichotomie impose $a_k \leq x^* \leq b_k$ ce qui donne :

$$0 \leq x^* - a_k \leq b_k - a_k \quad \Rightarrow \quad x^* - a_k \leq \frac{b_0 - a_0}{2^k} \tag{46}$$

Le résultat (46) prouve la convergence de la suite $\left\{x^{(k)}\right\}_{k\in\mathbb{N}}$ vers la solution x^*. Toutefois, afin de disposer d'une précision 10^{-n} soit $x^* - a_k \leq 10^{-n}$ il suffit de choisir $\dfrac{b_0 - a_0}{2^k} \leq 10^{-n}$. A titre d'exemple, pour une tolérance $\epsilon = 10^{-10}$ et $(b_0 - a_0) = 1$ nous obtenons $k = 34$ itérations.

Exercice ❻ ☞ ®

Dans cet exercice, il est demandé de trouver la racine de $f(x) = x + exp(x) + \dfrac{10}{1+x^2} - 5$, en utilisant la méthode de *dichotomie*

1) Écrire un script Matlab® permettant l'implémentation du schéma numérique de cette méthode.
2) Afficher, sur le même graphe, la fonction $f(x)$, la solution approchée et les approximations successives.
3) Calculer l'ordre et la vitesse de convergence de la méthode.

On donne : tolérance $= 10^{-8}$, $a_0 = -1.30$ *et* $b_0 = 3/2$

Script Matlab®

```
clear all ; close all ; clc ;
%  05/09/2019 Samir KENOUCHE : ALGORITHME PERMETTANT
% L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DE DICHOTOMIE

a = -1.3 ;   b = 3/2 ; itmax = 100 ; tol = 1e-6 ;
it = 0 ; center = (b + a)/2 ;  x = [a, center, b] ;
fun = @(x) x + exp(x) + 10./(1 + x.^2) - 5 ;

while it < itmax

    if fun(a)*fun(center) < 0

     b = x(2) ; a = x(1) ; center = (a + b)/2 ; x = [a, center, b] ;

    elseif fun(center)*fun(b) < 0

     b = x(3) ; a = x(2) ; center = (a + b)/2 ; x = [a, center, b] ;

    end

    if abs(fun(center)) < tol

     sol = center ;
       break
    end

    if it == itmax & abs(fun(center)) > tol

     disp('PAS DE CONVERGENCE POUR LA TOLERANCE CONSIDEREE')

    end

     it = it + 1;
     xit(it) = center ;
```

```
     itNumber = it ;
end

disp(strcat('LA RACINE APPROCHEE 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),'LineWidth',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),'ko','MarkerSize',10,'LineWidth',1.5)

line(xn, zeros(1, length(xn)),'LineStyle','-.','Color','k',...
'LineWidth',1) ; xlabel('x') ; ylabel('f(x)')
title('RACINE DE f(x) = 0 PAR LA METHODE DE DICHOTOMIE')
```

Les sorties renvoyées par ce script sont :

```
LA RACINE APPROCHEE VAUT :-0.92045
>> itNumber =
             21
```

Sur ce graphique, les étiquettes noires représentent les approximations successives et l'étiquette rouge c'est la racine calculée par *dichotomie*.

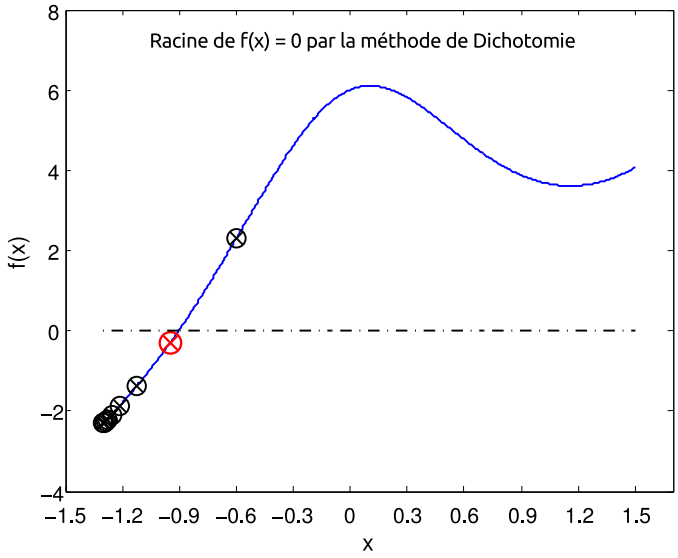

FIGURE 2: Racine de f obtenue par la méthode de *dichotomie*

La méthode de dichotomie est simple à mettre en œuvre. Néanmoins, elle souffre d'un inconvénient du fait que son seule critère d'arrêt consiste à contrôler la longueur de l'intervalle I_n à chaque itération. Ceci risque de rejeter

la racine recherchée, car elle ne tient pas suffisamment en compte du comportement effectif de la fonction. Ceci est parfaitement illustré par l'exemple ci-dessus.

IV. MÉTHODE DE NEWTON

Comme il a été montré précédemment, la méthode de dichotomie exploite uniquement le signe de la fonction f aux extrémités des sous-intervalles. Lorsque cette fonction est différentiable, on peut établir une méthode plus efficiente en exploitant les valeurs de la fonction et de ses dérivées. Le but de cette section, est la programmation sous Matlab®, de la méthode itérative de *Newton*. Afin d'appréhender cette dernière, soit la figure ci-dessous :

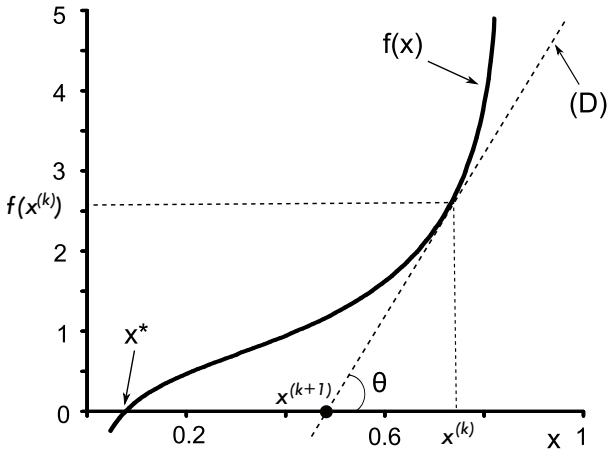

FIGURE 3: Principe de la méthode de *Newton*

Géométriquement, la solution approchée $x^{(k+1)}$ n'est autre que le point d'intersection de l'axe des abscisses et la tangente, au point $(x^{(k)},\ f(x^{(k)}))$, d'équation D : $y = f'(x^{(k)}) \times (x^{(k)} - x^{(k+1)}) + f(x^{(k)})$. Notons que x^* est la véritable racine de l'équation $f(x = x^*) = 0$, dont on cherche à approcher. À partir de la figure ci-dessus, nous en déduisons :

$$\tan\theta = \frac{f(x^{(k)})}{x^{(k)} - x^{(k+1)}} \tag{47}$$

Or on sait que :

$$f'(x^{(k)}) = \lim_{(x^{(k)} - x^{(k)}) \to 0} \frac{f(x^{(k)}) - 0}{x^{(k)} - x^{(k+1)}} = \tan\theta \tag{48}$$

À partir des Eqs. (47) et (48) on obtient le schéma numérique de la méthode de *Newton* :

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \tag{49}$$

Tenant compte de toutes les méthodes vues jusqu'à présent, on constate que la méthode de *Newton* nécessite à chaque itération l'évaluation de deux fonctions, à savoir f et de sa dérivée. Néanmoins, cet effort est compensé par une vitesse de convergence accrue, puisque cette méthode est d'ordre deux. Cet accroissement de la vitesse de convergence est conditionné par le choix de la valeur initiale qui doit être la proche possible de la racine recherchée.

g) ***Théorème****:* si $f : \mathcal{D} \subset \mathbb{R} \longmapsto \mathbb{R}$ est deux fois continument dérivable et $f(x = x^*) = 0$ alors $\exists\, \epsilon > 0$ tel que $\parallel x^{(0)} - x^* \parallel < \epsilon$. La suite des itérés $\left\{x^{(k)}\right\}_{k\in\mathbb{N}}$ donnée par (49) converge quadratiquement vers la solution x^*.

h) ***Démonstration****:* développons en série de Taylor, d'ordre deux, la fonction f autour du point $x^{(k)}$, il vient :

$$f(x) = f(x^{(k)}) + f^{'}(x^{(k)})\,(x - x^{(k)}) + \frac{f^{''}(x^{(k)})}{2}\,(x - x^{(k)})^2 \tag{50}$$

Posons $x = x*$ et divisons (50) par $f^{'}(x^{(k)})$, il vient :

$$f(x = x*) = 0 = \frac{f(x^{(k)})}{f^{'}(x^{(k)})} + \frac{f^{'}(x^{(k)})}{f^{'}(x^{(k)})}\,(x - x^{(k)}) + \frac{f^{''}(x^{(k)})}{2\,f^{'}(x^{(k)})}\,(x - x^{(k)})^2 \tag{51}$$

A partir de (49) nous avons :

$$\frac{f(x^{(k)})}{f^{'}(x^{(k)})} = x^{(k)} - x^{(k+1)} \tag{52}$$

En substituant (52) dans (51) nous obtenons :

$$\parallel x^* - x^{(k+1)} \parallel = \frac{\parallel f^{''}(x^{(k)}) \parallel}{2 \parallel f^{'}(x^{(k)}) \parallel}\,(x^* - x^{(k)})^2 \tag{53}$$

Posons :

$$c = \frac{\parallel f^{''}(x^{(k)}) \parallel}{2 \parallel f^{'}(x^{(k)}) \parallel}$$

En prenant,

$$c = \frac{\max\limits_{x\in\mathcal{D}} \parallel f^{''}(x) \parallel}{2 \min\limits_{x\in\mathcal{D}} \parallel f^{'}(x) \parallel} \tag{54}$$

Nous en déduisons finalement l'inégalité,

$$\parallel x^* - x^{(k+1)} \parallel \leq c \parallel x^* - x^{(k)} \parallel^2 \tag{55}$$

i) ***Erreur de la méthode****:* Il $\exists\, h, 0 < h < 1/c$ tel que pour tout $x^{(0)} \in [x^* - h\,;\, x^* + h]$ la suite $\left\{x^{(k)}\right\}_{k\in\mathbb{N}}$ vérifie :

$$\parallel x^{(k)} - x^* \parallel \leq \frac{1}{c}\,(c \parallel x^{(0)} - x^* \parallel)^{2^k} \tag{56}$$

L'erreur de la méthode de Newton est majorée par la quantité $\frac{1}{c}\,(c \parallel x^{(0)} - x^* \parallel)^{2^k}$. A titre illustratif prenons $c = 1$ et $\parallel x^{(0)} - x^* \parallel = 10^{-1}$, la précision sera de 10^{-2} à la première itération et de 10^{-4} à la seconde itération. En effet, chaque itération double la précision de la méthode. D'un autre côté, plus la dérivée seconde est faible (dans ce cas de figure la courbe est plus proche de sa tangente) plus la méthode est efficiente.

Exercice ❼ ☞ ®

Nous allons résoudre l'équation : $f(x) = x + exp(x) + 1$. Nous choisissons $x_0 = -1/2$ comme valeur initiale. Écrire un script matlab®, portant sur l'implémentation de la méthode de *Newton*, en suivant les étapes suivantes :

1) Faire un test si $f^{'}(x) = 0 \Longrightarrow$ arrêt du programme.
2) Le critère d'arrêt est : $|x_{n+1} - x_n| < \varepsilon$, x_n étant la solution approchée et ε, la tolérance considérée.
3) Afficher la solution approchée x_n.

4) Afficher le nombre d'itérations conduisant à la solution approchée.
5) Afficher sur le même graphe, la fonction $f(x)$, la solution approchée x_n et la droite tangente au point $(x_n, f(x_n))$.

Appliquez le même algorithme pour résoudre l'équation : $f(x) = 8x^3 - 12x^2 + 1$

Voici le script Matlab®

```
clear all ; close all ; clc ;
%  05/09/2019 Samir KENOUCHE : ALGORITHME PERMETTANT
% L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DE NEWTON

Nmax = 100 ; x = -1/2 ; it = 0 ; tol = 1e-04 ; verif = tol + 1/2 ;

 while (it < Nmax & verif >= tol)

  fx   = inline('x +exp(x) + 1') ; dfx = inline('1 + exp(x)') ;
  fx = feval(fx, x) ;  dfx = feval(dfx,x);

     if dfx ~= 0
        xn = x - (fx/dfx) ; verif = abs(fx/dfx) ; x = xn ;
     elseif dfx == 0
        disp('PAS DE SOLUTION, LA DERIVEE EST NULLE')
     end

     if (it == Nmax & verif > tol)
        disp('CETTE METHODE NE CONVERGE PAS POUR LA TOLERANCE CONSIDEREE')
     end
   it = it +1 ;
 end

 disp(strcat('CONVERGENCE, LA SOLUTION APPROCHEE EST xn = ', num2str(xn)))
 disp(strcat('LE NOMBRE D''ITERATION EST = ', num2str(it)))

 %%%%%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE GRAPHIQUE %%%%%%%%%%%%%%%%%%%%%%%%%%%%
 xi = linspace(-12/2,6/2,1000) ; fx   = inline('xi +exp(xi) + 1') ;
 fxi = feval(fx, xi) ; dfx = inline('1 + exp(x)') ;
 droite = dfx(xn)*(xi - xn) + fx(xn) ;

 figure('color',[1 1 1]) ; plot(xi,fxi,'LineWidth',2) ; hold on ;
 plot(xi,droite,'r','LineWidth',1) ;
 plot(x,fx(x),'kx','MarkerSize',12,'LineWidth',2) ;
 plot(x,fx(x),'ko','MarkerSize',12,'LineWidth',2) ;
 xlabel('x','fontweight','b','fontsize',12,'LineWidth',1)
 ylabel('f(x)','fontweight','b','fontsize',12,'LineWidth',1)

 %%%%%%%%%%%%%%%%%%%%%% ECRITURE A L'INTERIEUR DE LA FIGURE %%%%%%%%%%%%%%%%%%
  text('Interpreter','latex','String','$f(x) = x + exp(x) + 1 $', ...
 'Position',[-10/2 20/2],'FontSize', 15) ;
 text(xn,2*xn,['x_n = ',num2str(xn)],'Position',[-3 2], ...
 'BackgroundColor',[1 1 1]) ;
```

Les résultats des différents calculs, renvoyés par le script, sont portés sur la figure ci-dessous

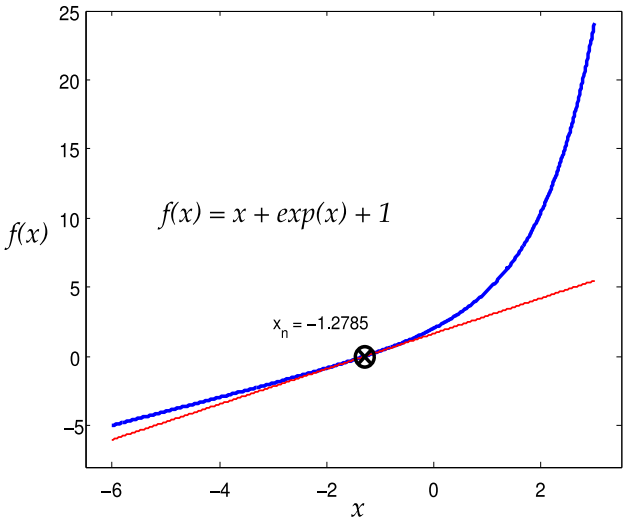

FIGURE 4: Racine de f obtenue par la méthode de *Newton*

Notons que la méthode de *Newton* converge de façon quadratique uniquement dans le cas où la racine recherchée de f est simple. Dans le cas contraire, elle converge de façon linéaire. En outre, cette méthode peut être utilisée afin de résoudre des systèmes d'équations non linéaires.

Exercice 8 ☞ Ⓢ

1) En utilisant la méthode de *Newton*, trouver les racines des fonctions :

$$\begin{cases} f_1(x) = \cos(x + 3\pi/8), & x \in [-1, 1] \\ f_2(x) = (x - \frac{1}{2})^2, & x \in [-1, 1] \\ f_3(x) = \cos(1/x^2), & x \in [-1, 1] \\ f_4(x) = \dfrac{x^2 - 3}{2}, & x \in [-5, 5] \end{cases} \tag{57}$$

2) Tracer, sur la même figure, la fonction et sa racine approchée.

A. *Méthode de la sécante*

La méthode de *Newton* est rapide et très utilisée, quand sa dérivée existe. Malheureusement dans certaines situations nous n'avons pas accès à cette dérivée. C'est pourquoi on se propose d'étudier une autre méthode, dite de la *sécante* qui une variante de la méthode de *Newton*. Cette dernière s'affranchit de la dérivée de la fonction $f(x)$ en l'approchant par l'accroissement :

$$f^{'}(x^k) \approx \frac{f^{(k)} - f^{(k-1)}}{x^{(k)} - x^{(k-1)}} \tag{58}$$

Le schéma numérique de cette méthode est donné par :

$$x^{(k+1)} = x^{(k)} - f^{(k)} \times \frac{x^{(k)} - x^{(k-1)}}{f^{(k)} - f^{(k-1)}} \tag{59}$$

En observant l'équation (59), on constate des similitudes avec celle de *Newton*. Toutefois, la méthode de la *sécante* nécessite l'initialisation de deux valeurs approchées x_0 et x_1 de la racine exacte de l'équation $f(x) = 0$. L'étude de la convergence de cette méthode est régit par le théorème ci-dessous :

j) ***Théorème:*** $f : \mathcal{D} \subset \mathbb{R} \longmapsto \mathbb{R}$ une fonction de classe $\mathcal{C}^2([\mathcal{D} = [a\,;\,b]])$ telle que les dérivées $f^{'}(x)$ et $f^{''}(x)$. La fonction change de signe $f(a) \times f(b) < 0 \Rightarrow f$ admet au moins une racine $x^* \subset \mathcal{D}$ telle que $f(x = x^*) = 0$. L'erreur de la méthode est estimée par :

$$\| x^{(k)} - x^* \| \leq \frac{c_1}{2\,c_2} (x^{(k)} - x^{(k-1)}) (b - x^{(k)}) \tag{60}$$

Avec $c_1 = \max\limits_{x \in \mathcal{D}} f^{''}(x)$ et $c_2 = \min\limits_{x \in \mathcal{D}} f^{'}(x)$. Si les valeurs initiales $x^{(0)}$ et $x^{(1)}$ sont suffisamment proches de la racine x^*, la méthode aura un ordre de convergence : $\varphi = \dfrac{1 + \sqrt{5}}{2} \simeq 1.618$.

Exercice ❾ ☞ ®

Trouver la racine de l'équation : $f(x) = x - 0.2 \times sin(4\,x) - 1/2$. On prend $x_0 = -1$ et $x_1 = 2$ comme valeurs initiales. Écrire un script Matlab®, portant sur l'implémentation de la méthode de la *sécante* en considérant une tolérance : $tol = 10^{-10}$.

1) Afficher la solution approchée x^*.
2) Afficher le nombre d'itérations conduisant à la solution approchée.
3) Afficher sur le même graphe, la fonction $f(x)$, la solution approchée et les approximations successives $x^{(k+1)}$.

Voici le script Matlab®

```
clear all ; close all ; clc ;
% LE 05/09/2019 Samir KENOUCHE : ALGORITHME PERMETTANT
% L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DE LA SECANTE

x(1) = -1 ; x(2) = 2 ; fun = x - 0.2.*sin(4.*x) - 1/2 ;
it = 0 ; tol = 1e-10 ; Nit = 20 ;

while it <  Nit

    fun = eval('fun',x) ;
    var = x(2) - fun(2)*((x(2) - x(1))/(fun(2) - fun(1))) ;
    xn = var ; x = [x(2), xn] ;

      if abs(xn - var) < tol
        sol = xn ; approx(it+1) = xn ;
        break
      end
    it = it + 1 ;
end

fun = inline('x - 0.2.*sin(4.*x) - 1/2') ; outPut = [sol, fun(sol)] ;
format long
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE GRAPHIQUE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
b = 4 ; a = -2 ; n = 1000 ; dx = (b - a)/n ; x = a:dx:b ;

figure('color', [1 1 1]) ; plot(x,fun(x),'LineWidth',1) ; hold on ;
plot(outPut(1),outPut(2),'ro','MarkerSize',12,'LineWidth',2) ;
plot(outPut(1),outPut(2) ,'rx','MarkerSize',12,'LineWidth',2) ;
plot(approx,fun(approx),'kx','MarkerSize',10,'LineWidth',1.5) ;
```

```
plot(approx,fun(approx),'ko','MarkerSize',10,'LineWidth',1.5) ;

line(x, zeros(1, length(x)),'LineStyle','-.','Color','k','LineWidth',1)
xlabel('x') ; ylabel('f(x)') ;
title('RACINE DE f(x) PAR LA METHODE DE LA SECANTE') ;

%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE A L'INTERIEUR DE LA FIGURE %%%%%%%%%%%%%%%%%%
text('Interpreter','latex','String','$x - \: 0.2.*sin(4.*x) - \: 1/2 $', ...
    'Position',[-3/2 2],'FontSize',15) ;
text(sol,2*sol,['sol = ',num2str(sol)],'Position',[-1/2 1/2],...
 'BackgroundColor',[1 1 1]) ;
```

Les résultats des différents calculs, renvoyés par ce script, sont portés sur la figure ci-dessous

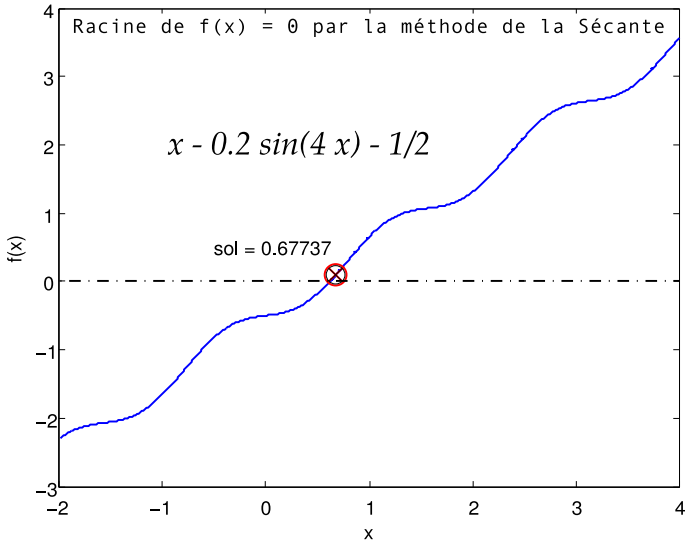

FIGURE 5: Racine de f obtenue par la méthode de la *sécante*

Rappelons aussi que la divergence d'un algorithme n'implique pas forcément l'inexistante de la solution. Très souvent, il faudra juste retenter avec une autre valeur initiale x_0, le nombre d'itérations et la tolérance considérés. Parfois, nous serons amener à changer carrément de méthode numérique.

Exercice ⓾ ☞ Ⓢ

1) Trouver à l'aide de la méthode de la *sécante*, la racine approchée des fonctions :

$$\begin{cases} f_1(x) = \cos(x^x) - \sin(\exp(x)), \quad x \in [1/2, 3] \\ f_2(x) = x\,\log(x) - \log(x), \quad x \in [-2, 2] \\ f_3(x) = x\,\exp(x) - \exp(x), \quad x \in [-2, 2] \\ f_4(x) = x^3 - 3\,x^2 2^{-x} + 3x4^{-x} - 8^{-x}, \quad x \in [0, 1] \\ f_5(x) = x^3 + 4x^2 - 10, \quad x \in [-3, 3] \end{cases} \tag{61}$$

2) Tracer sur un graphe approprié, la fonction et la racine trouvée.

V. TRAVAUX PRATIQUES AVEC DES FONCTIONS MATLAB PRÉDÉFINIES

Matlab® dispose de fonctions prédéfinies destinées à chercher les zéros d'une fonction. On testera dans un premier temps la fonction `fzero`. Elle a pour syntaxe :

```
[x , fval, exitflag, output] = fzero(fun, x0, options)
```

Cette fonction cherche un zéro autour de la valeur initiale x_0. La racine approchée x renvoyée est prés d'un point ou la fonction `fun` change de signe. La commande renvoie NaN (Not a Number) dans le cas où la recherche de la racine a échoué. On peut aussi spécifier un intervalle dans le quelle `fzero` va chercher la racine. Cependant, il faut s'assurer que la fonction change de signe dans cet intervalle. L'argument `options` est de type `srtucture` qui recèle les options d'optimisation indiquées dans `optimset`. Voici un exemple :

```
clear all ; clc ;
% LA FONCTION fzero
fun = @(x) x - 0.2.*sin(4.*x) - 1/2 ;
lB = -2 ;  uB = 4 ; % LES BORNES MIN ET MAX

opts = optimset('Display','iter','FunValCheck','on','TolX',1e-6) ;
[racine,funEval,exitTest,output] = fzero(fun, [lB uB], opts) ;
% find the zero of fun between lowerBound and upperBound
```

Voici les résultats affichés par ce script :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE PAR DEFAUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%
 Func-count     x          f(x)
     2              -2      -2.30213
     3         0.357245     -0.340747
     4         0.730969      0.187769
     5         0.598194    -0.0379606
     6         0.620523   -0.00202224
     7         0.621766   8.20089e-06
     8         0.621761  -6.06744e-09
     9         0.621761  -6.06744e-09

Zero found in the interval [-2, 4]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
>> racine   =
            0.6218
>> funEval  =
           -6.0674e-09
>> exitTest =
            1
>> output =
            intervaliterations: 0
                    iterations: 7
                     funcCount: 9
                     algorithm: 'bisection, interpolation'
                       message: [1x34 char]
```

La sortie `exitTest = 1` signifie que l'algorithme a convergé vers la solution approchée. Une valeur de `exitTest < 0` indiquerai plutôt une divergence de l'algorithme. L'argument `funEval = -6.0674e-09` donne l'évaluation de la fonction à la dernière itération, c'est-à-dire à la racine approchée. Les sorties renvoyées par `output` sont `funcCount = 9` représentant le nombre d'évaluations de la fonction et `intervaliterations` est le nombre d'itérations pour trouver un intervalle contenant la racine approchée. Le champ `iterations = 7` exprime le nombre d'itérations pour trouver la racine approchée et `message` renvoie la chaîne de caractères : `'Zero found in the interval [-2, 4]'`.

La sortie `algorithm` renvoie l'algorithme utilisé pour résoudre cette équation qui est `'bisection'`. Le champ de la structure `opts` indiqué par `optimset('Display','iter', ...)` affiche des détails pour chaque itération. Le champ indiqué par `optimset(...., 'FunValCheck','on',...)` contrôle si les valeurs de la fonction sont réelles et affiche dans le cas contraire un avertissement quand `fzero` renvoie une valeur complexe ou NaN. Le champ indiqué par `optimset(..., 'TolX', 1e-6)` désigne la tolérance avec laquelle sera calculée la racine.

Exercice ❶ ☞ Ⓢ

1) Trouver la racine des fonctions suivantes :

$$\begin{cases} f_1(x) = \exp(x) - 2\,\cos(x), \quad x \in [-1, 1] \\ f_2(x) = \dfrac{x^3 + 2\,x - 5}{x^2 + 1}, \quad x \in [-2, 2] \end{cases} \tag{62}$$

2) Tracer, dans un graphe approprié, les fonctions et leurs racines. Annoter les graphes.

Prenez comme valeurs initiales $x_0 = 1/2$ pour $f_1(x)$ et $x_0 = 2$ pour $f_2(x)$.

Nous appliquerons désormais la fonction prédéfinie `fsolve`, pour la résolution de systèmes d'équations non-linéaires. Soit à résoudre le système d'équations suivant :

$$\begin{cases} 2\,x_1 - x_2 = \exp(-x_1) \\ -x_1 + 2\,x_2 = \exp(-x_2) \end{cases} \tag{63}$$

Pour résoudre ce système d'équations, nous commencerons d'abord par écrire un script *M-file*, selon :

```
function fun = myfunction(x)

fun = [2*x(1) - x(2) - exp(-x(1));
      -x(1) + 2*x(2) - exp(-x(2))];
return
```

Bien évidemment, comme il a été déjà signalé précédemment, ce fichier doit être absolument sauvegardé dans le répertoire courant, sous le nom de `myfunction.m`. Sinon Matlab® ne reconnaitra pas la fonction. Une fois cette étape achevée, on écrira le script comme suit :

```
clear all ; clc ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xinit = [-5 ; -3];
opts = optimset('NonlEqnAlgorithm','lm','LargeScale','off','Display','iter','
   TolX',1e-6,'TolFun',1e-6);
[x, fval, exitflag, output, jacobian] = fsolve(@myfunction, xinit, opts) ;
```

Voici les résultats affichés par ce script :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE PAR DEFAUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%
 Iteration  Func-count     Residual      derivative    Lambda
     0           3          24597.8
     1          11          38.3734      -40.9         8.07794e-28
     2          18         0.520524      -0.136        9.41035e-29
     3          25      0.000166037      -0.000798     0.421728
     4          31       3.0741e-08      -2.42e-11     0.191427
     5          37      1.38904e-12       3.13e-17     0.0948403

Optimization terminated: directional derivative along
 search direction less than TolFun and infinity-Norm of
 Gradient less than 10*(TolFun+TolX).
```

Le champ spécifié par `optimset('NonlEqnAlgorithm','lm', ....)` impose l'utilisation de l'algorithme de `Levenberg-Marquardt`. Si par exemple la chaîne de caractères `'lm'` est remplacée par `'gn'`, cela signifie qu'on utilisera l'algorithme de `Gauss-Newton`. Ce champ `optimset(....,'LargeScale','off', ....)` stipule l'utilisation de méthodes moyenne dimension. L'argument `(...,'TolFun',1e-6)` est la tolérance finale sur la valeur de la fonction. La sortie `jacobian` renvoie le Jacobien de la fonction :

```
>> jacobian =
            2.5671   -1.0000
           -1.0000    2.5671
```

La sortie `output`, de type *structure*, renvoie les champs suivants :

```
>> output =

        iterations: 6
         funcCount: 37
          stepsize: 1
         algorithm: [1x46 char] % medium-scale: Levenberg-Marquardt
           message: [1x147 char] % Optimization terminated ...
```

Les valeurs numériques renvoyées sont `iterations: 6`, `funcCount: 37` et `stepsize: 1`. La fonction a été évaluée 37 fois et l'algorithme converge vers la solution approchée au bout de la $6^{\text{ième}}$ itération. La sortie `stepsize: 1` indique le pas final de la méthode moyenne dimension.

Exercice ❷ ☞ Ⓢ

1) Trouver les racines des fonctions non-linéaires suivantes :

$$\begin{cases} f_1(x) = -x^2 + \cos^2(2x), \quad x \in [0, 2] \\ f_2(x) = x - \exp(-(1+x)), \quad x \in [-1, 1] \\ f_3(x) = x(1+\exp(x)) - \exp(x), \quad x \in [-1, 1] \\ f_4(x) = x^3 - 4x - 9, \quad x \in [1, 3] \\ f_5(x) = x - \exp(1/x), \quad x \in [1, 4] \\ f_6(x) = -5x^3 + 39x^2 - 43x - 39, \quad x \in [1, 5] \\ f_7(x) = x^3 - 6x + 1, \quad x \in [-2, 2] \\ f_8(x) = \frac{5}{2} - \sin(x) + \frac{\pi}{6} - \frac{\sqrt{3}}{2}, \quad x \in [0, \pi] \\ f_9(x) = x^2 - x - 2, \quad x \in [-3, 3] \end{cases}$$

2) Trouver les racines des systèmes d'équations suivants :

$$\begin{cases} x_1 + x_2 - 3 = 0 \\ x_1^2 + x_2^2 - 9 = 0 \end{cases}$$

$$\begin{cases} x_1^2 + x_2^2 - 1 = 0 \\ \sin(\pi\, x_1) + x_2^3 = 0 \end{cases}$$

Prendre comme valeurs initiales $x_0 = \left[\frac{1}{2}, \frac{1}{2}\right]$.