Analyse Numérique
Corrigés des Travaux Pratiques 2024 – 2025
Séances 7-8

Les codes des méthodes utilisées (`dico`, `regfal`, `newton`, `newton_rl`) sont en annexe. Notons que des diverses vérifications des paramètres d'entrée dans ces codes sont utiles mais pas indispensables.

1.a) Comme aucune dérivée de $J_0(x)$ n'est connue a priori, les méthodes de dichotomie et de la fausse position sont les seules applicables ici.

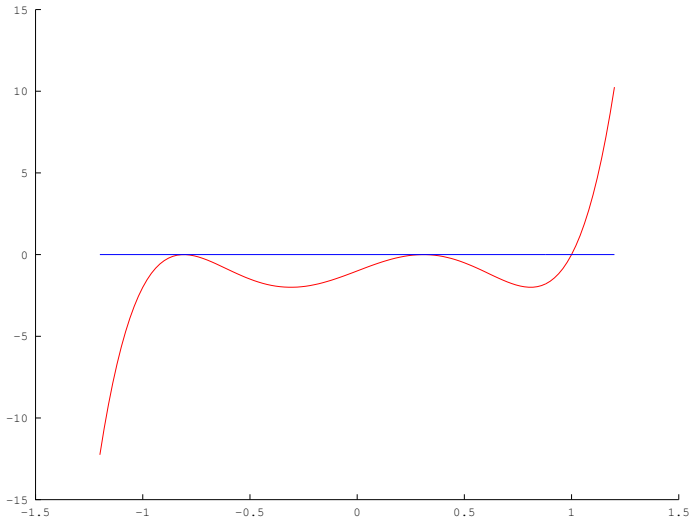
```
clear; close;
x = 0:0.1:3; % zéro entre 0 et 3
f = @(x)(besselj(0,x));
plot(x,f(x),'r'); hold on
plot(x,zeros(length(x),1),'b'); hold off
tol = 1e-12;
res = 1e-12;
maxit = 100;
[x it] = dico(f, 0, 3, tol)
[x it] = regfal(f, 0, 3, res, maxit)
```

Dans les deux cas on obtient $x \approx 2.40482555769$ (en 42 itérations pour la méthode de dichotomie, en 7 pour la fausse position).

1.b) Comme la dérivée de $f(x) = e^{-x^2+22x-120} - 0.5$ vaut $f'(x) = (-2x+22)e^{-x^2+22x-120}$, la méthode de Newton peut être utilisée et est assez efficace, si elle converge. Dans le code qui suit seule la version avec recherche linéaire converge, mais il suffit de choisir un point de départ plus proche du zéro $x \approx 9.69879010895$ pour que la version sans recherche linéaire convergence également.

```
clear; close;
x = 8:0.1:10; % zéro entre 8 et 10
f = @(x)(e.^(-x.*x+22.*x-120)-0.5);
% la dérivee est connue
fp = @(x)((-2.*x+22).*e.^(-x.*x+22.*x-120));
plot(x,f(x),'r'); hold on
plot(x,zeros(length(x),1),'b'); hold off
%
x0 = 8;
maxit = 100;
res = 1e-12;
[x it r] = newton_rl(f, fp, x0, res, maxit)
%
x0 = 8; % mais 9.1 est OK
maxit = 100;
res = 1e-12;
[x it r] = newton(f, fp, x0, res, maxit)
%
```

1.c) Notons pour commencer que la fonction a au plus 5 zéros réels (pourquoi ?). Essayons de localiser un maximum d'entre eux avec la commande `plot`. Les 5 zéros, dont 2 paires de zéros potentiellement doubles (pourquoi ?), peuvent se voir sur le graphique suivant :



Pour ce qui est des zéros (potentiellement) doubles, on ne pourra (potentiellement) pas les localiser en utilisant les méthodes d'encadrement puisque la fonction a le même signe de part et d'autre du zéro. Il faut donc utiliser la méthode de Newton-Raphson ou ses variantes. Notons que dans ce cas la méthode de Newton-Raphson n'a pas la garantie de la convergence quadratique puisque les zéros (potentiellement) doubles sont aussi (potentiellement) les points d'extremum local, et donc leur dérivée s'annule. Ces zéros, qui valent -0.8090 et 0.3090, ne peuvent être déterminés qu'avec peu de précision (en fait, la détermination des zéros multiples est un problème intrinsèquement mal conditionné). Pour ce qui est du zéro le plus grand en module, en utilisant Newton on obtient `0.999999999999997` (ce qui vaut 1 aux erreurs d'arrondi près), avec $|f(x)|$ de l'ordre de $\mathcal{O}(u)$ en seulement 5 itérations ! Le code correspondant est

```
clear; close;
x = -1.2:0.02:1.2; % zéro entre -1.2 et 1.2
f = @(x)(16.*x.^5 - 20.*x.^3 + 5.*x - 1);
fp = @(x)(80.*x.^4 - 60.*x.^3 + 5);
plot(x,f(x),'r'); hold on
plot(x,zeros(length(x),1),'b'); hold off
%
x0 = -1;
maxit = 100000;
res = 1e-12;
[x it r] = newton(f, fp, x0, res, maxit)
%
x0 = 0.3;
maxit = 100000;
res = 1e-12;
[x it r] = newton(f, fp, x0, res, maxit)
%
x0 = 0.9;
```

```
maxit = 10;
res = 1e-12;
[x it r] = newton(f, fp, x0, res, maxit)
```

2.a) La fonction vectorielle $F(\mathbf{x})$ s'annule si et seulement si une de ses normes $\|F(\mathbf{x})\|$ s'annule. Pour $\mathbf{x} = (x, y)$ on peut visualiser $\|F(x, y)\|$ en utilisant la fonction `mesh` et voir où cette fonction croise le plan $z = 0$.

2.b,c) On peut procéder comme suit

```
clear; close;
% la fonction F(x)
%   : argument est un vecteur (x,y)
%   : retourne un vecteur 2x1
F = @(x)( [x(1).^2-x(2)-1; ...
              (x(1)-2).^2+(x(2)-0.5).^2-1] );
% la matrice Jacobienne Fp(x)
%   : argument est un vecteur (x,y)
%   : retourne une matrice 2x2
Fp = @(x)( [2.*x(1),  -1; ...
            2.*(x(1)-2), 2.*x(2)-1] );
% la fonction ||F(x,y)||
%   : argument est un vecteur (x,y)
%   : retourne un scalaire
nF = @(x)( norm( F(x) ) );
% on limite les grandes valeurs de nF
% cela facilite la visualisation
nFlim = @(x)( min( nF(x), 1 ) );
% représenter nF via mesh
X = -1:0.1:3;
Y = -1:0.1:3;
for i = 1 : length(X)
    for j = 1 : length(Y)
        Z(i,j) = nFlim( [X(i);Y(j)] );
    end
end
mesh(X, Y, Z' );
xlabel ("x"); ylabel ("y"); zlabel ("z");
% les deux points sont aux alentoures
% de (1.5,1.5), et (1,0)
maxit = 10;
res = 1e-12;
[x it r] = newton(F, Fp, [1.5;1.5], res, maxit)
[x it r] = newton(F, Fp, [1; 0], res, maxit)
```

Notons que le code de l'instruction `newton` utilisée ici est le même que celui de l'exercice précédente ; il est donnée en annexe plus bas dans ce corrigé. Un tel recyclage est possible grâce, entre autres, à une subtilité dans l'encodage de l'itération

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

qui est codée comme

```
x = x - fp(x)\f(x)
```

au lieu d'utiliser une variante plus «habituelle»

```
x = x - f(x)/fp(x)
```

Les deux instructions sont équivalentes dans le cas d'une seule équation non linéaire (car `x`, `f(x)` et `fp(x)` sont des scalaires), mais c'est bien la première variante qui correspond à l'itération vectorielle

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{f}'(\mathbf{x}^{(k)})^{-1}\,\mathbf{f}(\mathbf{x}^{(k)})$$

En effet, dans ce cas `x`, `f(x)` sont des vecteurs, `fp(x)` est une matrice (la matrice Jacobienne) et `fp(x)\f(x)` correspond à la résolution d'un système linéaire.

Annexe

Les quelques fonctions utiles

— la méthode de la dichotomie :

```
function [x it] = dico(f,a,b,tol)
% X = dico(F,A,B,TOL)
%    détermine un zéro de la fonction F(X) d'une
%    variable dans l'intervalle [A,B] si
%    la condition suivante (condition de "bracketing")
%                    F(A)*F(B) < 0
%    est satisfaite;
% arguments:
%    F - la fonction dont on détermine un zéro
%    A,B - les extrémités de l'intervalle sur lequel
%          on cherche un zéro
%    TOL - la précision absolue avec laquelle on
%          détermine un zéro
% sortie:
%    X - l'approximation calculée du zéro
%    IT - nombre d'itérations utilisées
if(f(a)*f(b) >= 0)
    error('f(a)*f(b) doit être < 0')
else
    it = 0;
    while (abs(a-b) > tol)
        x = (a+b)/2;
        it=it+1;
        if(f(x)*f(a) < 0)
            b = x;
        elseif(f(x)*f(b) < 0)
            a = x;
        else
            return;
        end
    end
end
```

— La méthode de la fausse position :

```
function [x it] = regfal(f,a,b,res,maxit)
% X = regfal(F,A,B,RES,MAXIT)
%    détermine un zéro de la fonction F(X) d'une
%    variable dans l'intervalle [A,B] si
%    la condition suivante (condition de "bracketing")
%                    F(A)*F(B) < 0
%    est satisfaite;
% arguments:
%    F - la fonction dont on détermine un zéro
%    A,B - les extrémités de l'intervalle sur lequel
%          on cherche un zéro
%    RES - la méthode s'arrête si |F(X)| <= RES
%    MAXIT - le nombre maximal d'évaluations de F
% sortie:
%    X - l'approximation calculée du zéro
%    IT - nombre d'itérations utilisées
%    R - valeur de |F(X)|
if(f(a)*f(b) >= 0)
    error('f(a)*f(b) doit être < 0')
else
    for it=1:maxit
        x = a - (b-a)*f(a)/(f(b)-f(a));
        if(f(x)*f(a) < 0)
            b = x;
        elseif(f(x)*f(b) < 0)
            a = x;
        end
        if(abs(f(x)) < res)
            return;
        end
    end
end
```

— La méthode de Newton-Raphson :

```
function [x it r] = newton(f, fp, x0, res, maxit)
% X = newton(F,FP,X0,RES,MAXIT)
%    détermine un zéro de la fonction F(X) d'une ou de
%    plusieurs variables avec X0 comme point de départ
% arguments:
%    F - la fonction dont on détermine un zéro
%    FP - la dérivée de F
%    X0 - approximation initiale du zéro
%    RES - la méthode s'arrête si ||F(X)|| < RES
%    MAXIT - le nombre maximal d'itérations
% sortie:
%    X - l'approximation calculée du zéro
%    IT - nombre d'itérations utilisées
%    R - valeur de ||F(X)||
```

```
x = x0;
for it = 1:maxit
    r = norm(f(x));
    if(r < res)
        it--;
        return;
    elseif(fp(x) == 0)
        error(['La méthode a obtenu une dérivée nulle en x =
            ',num2str(x)]);
    end
    x = x-fp(x)\f(x);
end
```

— La méthode de Newton avec recherche linéaire :

```
function [x it r] = newton_rl(f, fp, x0, res, maxit)
% X = newton_rl(F,FP,X0,RES,MAXIT)
%    détermine un zéro de la fonction F(X) d'une ou de
%    plusieurs variables avec X0 comme point de départ
% arguments:
%    F - la fonction dont on détermine un zéro
%    FP - la dérivée de F
%    X0 - approximation initiale du zéro
%    RES - la méthode s'arrête si ||F(X)|| < RES
%    MAXIT - le nombre maximal d'itérations
% sortie:
%    X - l'approximation calculée du zéro
%    IT - nombre d'itérations utilisées
%    R - valeur de ||F(X)||
x = x0;
for it=1:maxit
    r = norm(f(x));
    if(r < res)
        it--;
        return;
    elseif(fp(x) == 0)
        error(['La méthode a obtenu une dérivée nulle en x =
            ',num2str(x)]);
    end
    d = fp(x)\f(x);
    a = 1;
    while(true)
        xn = x - d*a;
        if(norm(f(xn)) < r)
            x = xn;
            break
        end
        a = a/2;
    end
end
```