ANALYSE NUMÉRIQUE

T.P. N° 0

Prénom Nom1
Prénom Nom2

N° Groupe : A00

Observations :

Notes :	ANALYSE	:	**Total :**
	RÉSULTATS	:	
	PROGRAMMATION	:	
	RAPPORT	:	

EISTI, 16 avril 2013

Titre du TP

16 avril 2013

Table des matières

1 Introduction

Soit $f(x)$ une fonction réelle. On cherche ses racines, c'est-à-dire les solutions de l'équation

$$f(x) = 0 \tag{1}$$

Nous allons utiliser la *méthode de Newton* qui consiste à chercher la solution comme limite de la suite :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \; ; \; n = 1, 2, \ldots \tag{2}$$

x_0 étant choisi aléatoirement.

2 Méthodes et programme

2.1 Aspects théoriques

L'algorithme issu de la méthode de Newton est le suivant :

Choix aléatoire de x_0,

$$x_{n+1} = g(x_n)$$

où

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

Il s'agit d'un algorithme itératif qui permet de remplacer le problème initial – recherche de racines de la fonction $f(x)$ – par la recherche de point fixe de la fonction $g(x)$:

$$x = g(x)$$

Comme pour tout algorithme itératif, quelques questions se posent :

(a) Est-ce que la suite (2) converge ?

(b) Si la suite x_n converge, est-ce que sa limite est bien la solution recherchée de l'équation (1) ?

(c) Comment choisir la condition initiale pour garantir la convergence vers la solution de l'équation (1) ?

Voici un théorème (voir [1]) qui donne une réponse aux questions (a) et (b).

Théorème 2.1. *Soit ξ un point fixe de la fonction $g(x)$:*

$$\xi = g(\xi)$$

Si la dérivée $g'(x)$ est continue et si

$$|g'(\xi)| < 1$$

alors il existe un intervalle $[a, b]$ tel que $\xi \in [a, b]$ et que pour tout $x_0 \in [a, b]$ la suite récurrente

$$x_0,\ x_{n+1} = g(x_n),\ n = 1, 2, 3, \ldots$$

converge vers ξ.

Revenons à la méthode de Newton qui résout l'équation (1). Soit ξ la solution de l'équation : $f(\xi) = 0$. Alors, ξ est le point fixe de la fonction

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

On calcule la dérivée :

$$g'(x) = \frac{f(x)f''(x)}{[f'(x)]^2}$$

Quand $x = \xi$ on a :

$$g'(\xi) = 0 < 1$$

Donc on peut appliquer ici le théorème 2.1.

On remarque cependant que l'intervalle $[a, b]$ de convergence n'est pas défini explicitement. La question (c) sur le choix de la condition initiale dans l'algorithme reste donc ouverte. Voici un théorème qui donne une précision utile :

Théorème 2.2. *Si $f \in C^2[a, b]$ vérifie les conditions suivantes :*

(i) $f(a) \cdot f(b) < 0$

(ii) $\forall x \in [a,b] \quad f'(x) \neq 0$

(iii) $\forall x \in [a,b] \quad f''(x) \neq 0$

alors, en choisissant $x_0 \in [a,b]$ *tel que*

$$f(x_0)f''(x_0) > 0$$

la suite

$$x_0,\ x_{n+1} = g(x_n),\ n = 1, 2, 3, \ldots$$

converge vers l'unique solution ξ *de l'équation (1) dans* $[a,b]$.

Nous avons appliqué la méthode de Newton au calcul des racines carrées de nombres réels (voir [1]). Soit $\alpha > 0$ le nombre dont on cherche la racine carrée. Pour ce faire, on doit résoudre l'équation

$$x^2 - \alpha = 0$$

Alors $f(x) = x^2 - \alpha$.

Les approximations successives de $\sqrt{\alpha}$ par la méthode de Newton s'écrivent :

$$x_0, \quad x_{n=1} = x_n - \frac{f(x_n)}{f'(x_n)} = \frac{x_n}{2} + \frac{\alpha}{2x_n}$$

Le programme qui réalise cet algorithme permet de calculer la racine carrée d'un nombre positif arbitraire avec une précision voulue. Le source se trouve en annexe.

En ce qui concerne le choix de l'intervalle l'application du théorème 2.2, dans le cas particulier de la fonction $f(x) = x^2 - \alpha$, permet de déduire les conditions suivantes que doivent être vérifiées par l'intervalle $[a,b]$ et le point initial x_0 :

$$\begin{aligned} a^2 < \alpha < b^2 \\ 0 \notin [a,b] \\ x_0^2 > \alpha \end{aligned} \tag{1}$$

Remarquons que, contrairement au cas général, la vérification de ces conditions peut être facilement programmée. Cependant, nous avons décidé de ne pas programmer la vérification de ces conditions du théorème pour permettre la réutilisation de notre programme pour la recherche des racines d'autres fonctions. En effet, nous avons défini la fonction $f(x)$ et ses dérivées sous forme de fonctions indépendantes. Le programme principal qui réalise la méthode fait appel à ces fonctions. Pour l'utiliser avec d'autres fonctions il suffit de changer les définitions dans les fichiers correspondants. Cela offre une plus grande généralité, mais au prix de choix d'intervalle presque manuel.

Le programme propose à l'utilisateur de choisir un intervalle et un point initial et vérifie juste si une racine peut être trouvée dans l'intervalle choisi :

$$f(a)f(b) < 0$$

Dans le cas de réponse négative l'utilisateur peut modifier son choix ou abandonner.

2.2 Vitesse de convergence

Une fois que la méthode est définie et sa convergence est assurée nous pouvons nous poser une nouvelle question : quel est le nombre d'itérations à faire pour obtenir une précision de calcul voulue ? La réponse à cette question dans le cas de la méthode de Newton est donnée par le théorème suivant :

Théorème 2.3. *Soit ξ une racine simple de $f(x)$:*

$$0 = f(\xi), \quad f'(\xi) \neq 0, \quad f''(\xi) \neq 0$$

Soit $[a, b]$ l'intervalle de convergence établi dans le théorème 2.3. Soit

$$x_0 \in [a, b], \quad x_{n+1} = g(x_n)$$

la suite qui converge vers ξ. Si on note

$$e_n = x_n - \xi$$

l'erreur d'approximation, alors on a pour cette erreur l'estimation suivante

$$|e_{n+1}| \cong \left| \frac{f''(\xi)}{f'(\xi)} \right| \cdot \frac{|e_n|^2}{2}$$

Le dernier théorème montre que la convergence est quadratique. On dit dans ces cas que la méthode récurrente est d'ordre 2.

La difficulté principale de la méthode de Newton est le choix de l'intervalle dans lequel se trouve la racine. L'application du théorème 2.3 à chaque fonction particulière aboutit à un ensemble des conditions dont il faut vérifier la satisfaction. Dans le cas général c'est une tâche difficile, parce qu'il faut écrire – pour chaque fonction – un programme particulier qui vérifier numériquement ces conditions. En particulier c'est la condition 2 du théorème qui est la plus difficile à vérifier. En effet, nous ne pouvons pas parcourir numériquement tous les points d'un intervalle sur l'axe réel. Donc cette condition peut être vérifiée seulement en un nombre fini de points.

On peut choisir un réseau de points très dense en supposant que la fonction $f(x)$ ne présente pas de variations importantes sur de petits intervalles (déjà une restriction !). Mais cela impose un temps de calcul important pour effectuer le contrôle.

2.3 Programmes

Le programme `racine.sci` est divisé en trois parties :

Initialisations diverses.- En particulier nous définissons la forme de la fonction f et de ses deux premières dérivées. Le programme demande aussi à l'utilisateur de fournir les valeurs :
- des paramètres de la fonction (ici il s'agit du paramètre α) ;
- de la valeur du pas de discrétisation `dt` pour le calcul de la fonction (un choix de `dt = 0.001` est, en général bon) ;
- du nombre maximal `maxIter` d'itérations à effectuer (en fonction de la nature de la fonction on choisit `maxIter` entre 50 et 1000, voire plus, si nécessaire)

1e partie.- Recherche d'un intervalle dans lequel se trouve un nombre impair de racines, en utilisant le test $f(a)\dot{f}(b) < 0$ ou a et b sont les extrémités de l'intervalle. Ces extrémités sont fournies par l'utilisateur et si le test n'est pas vérifié, alors l'utilisateur est invité à donner un nouvel intervalle, ou, éventuellement, de s'arrêter.

2e partie.- Évaluation de la racine de la fonction dans l'intervalle retenu lors de la 1e partie en utilisant l'algorithme itératif exposé plus haut.

L'arrêt des itérations de cet algorithme peut se faire de deux façons :

- Soit la précision voulue pour le calcul de la racine est atteinte.
- Soit le nombre d'itérations devient plus grand qu'un nombre maximal d'itérations – `maxIter` – fourni par l'utilisateur.

La valeur de la précision pour le calcul de la racine n'est pas fournie par l'utilisateur mais calculée par le programme en utilisant le résultat du théorème 2.3 et la précision de la machine. Nous devons arrêter les itérations dès que $|e_{n+1}| \cong eps$. D'après (2.3), nous avons : $|e_n| \cong \left(2 \cdot eps \cdot \left|\frac{f'(\xi)}{f''(\xi)}\right|\right)^{-\frac{1}{2}}$. Du fait que nous ne connaissons pas les valeurs de $f'(\xi)$ et $f''(\xi)$, nous prenons la valeur du rapport égale à 1 et nous avons ainsi pour la précision la valeur

$$precision = \sqrt{2 \cdot eps}$$

Le programme à la fin des calculs, crée un fichier de compte rendu, appelé ici `newton.crd`, qui contient le détail des calculs et qui peut être repris tel quel dans un fichier TEX. De plus le programme crée un graphique qui représente la fonction f dans l'intervalle choisi et l'approximation pour le calcul de la racine.

Limitations.- 1° Il faut noter, et ceci constitue la principale limitation de la méthode de Newton et, par voie de conséquence du programme aussi, que nous ne pouvons calculer que des racines qui sont des nombres réelles.
2° Il va de soi, que les paramètres de la fonction doivent aussi être des nombres réels.

3 Résultats

Les résultats du programme pour $\alpha = 0.0025$ sont donnés par la table 1 pour différentes valeurs de la précision.

La figure 1 fournit la représentation graphique du déroulement de l'algorithme.

4 Discussion

Les résultats obtenus par l'exécution du programme, qu'on peut voir sur la table 1 sont corrects.

Afin de tester la robustesse de l'algorithme nous avons exécuté le programme en prenant $\alpha = 10^{-16}$. Les résultats donnés par la table 2 montrent que les performances de l'algorithme ne se détériorent pas.

D'autre part pour tester la validité de la méthode de Newton pour le calcul de racines carrées, nous avons utilisé le programme pour calculer la racine carrée

```
========================================================================

Méthode de Newton pour le calcul des racines de fonction y = x^2-alpha

Valeur de alpha         =  2.50000000e-003

Pas de discrétisation   =  1.00000000e-003

Nombre max d'itérations =       1000

Approximation en fonction de la valeur de la précision

     Précision     Racine x              f(x)            Nb iter

2.10734243e-008  5.00000000e-002    5.63785130e-018         5

Vitesse de convergence

Précision =  2.10734243e-008

Itération       x
     1  1.00000000e-001
     2  6.25000000e-002
     3  5.12500000e-002
     4  5.00152439e-002
     5  5.00000023e-002

========================================================================
```

TABLE 1 – Résultats numériques pour $\alpha = 0.0025$

FIGURE 1 – Représentation graphique des approximations successives. $x_0 = 0.95$, $\alpha = 0.0025$

```
======================================================================

Méthode de Newton pour le calcul des racines de fonction y = x^2-alpha

Valeur de alpha         =  1.00000000e-016

Pas de discrétisation   =  1.00000000e-005

Nombre max d'itérations =       1000

Approximation en fonction de la valeur de la précision

     Précision     Racine x              f(x)            Nb iter

2.10734243e-008  1.45941684e-008    1.12989751e-016        23

Vitesse de convergence

Précision =  2.10734243e-008

Itération       x
     1  1.00000000e-001
     2  5.00000000e-002
     3  2.50000000e-002
     4  1.25000000e-002
     5  6.25000000e-003
     6  3.12500000e-003
     7  1.56250000e-003
     8  7.81250000e-004
     9  3.90625000e-004
    10  1.95312500e-004
    11  9.76562503e-005
    12  4.88281257e-005
    13  2.44140639e-005
    14  1.22070340e-005
    15  6.10352109e-006
    16  3.05176874e-006
    17  1.52590075e-006
    18  7.62983143e-007
    19  3.81557104e-007
    20  1.90909594e-007
    21  9.57167010e-008
    22  4.83807254e-008
    23  2.52238321e-008

======================================================================
```


TABLE 2 – Résultats numériques pour $\alpha = 10^{-16}$

FIGURE 2 – Représentation graphique de l'approximation de $\sqrt{2}$. $x_0 = 10$, $\alpha = 2$

de 2. Le tableau 3 et la figure 2 fournissent les résultats. Nous pouvons constater que l'approximation de $\sqrt{2}$ est excellente.

5 Conclusions

Nous avons appliqué la méthode de Newton pour calculer les racines carrées des nombres réels positifs. Nous avons pu constater que cette méthode a de bonnes performances de convergence pour le calcul des racines dans ce cas. Il serait donc intéressent de tester la méthode de Newton à la localisation des racines des polynômes.

D'autre part il serait aussi intéressent de comparer la méthode de Newton pour le calcul de la racine carrée avec d'autres méthodes d'analyse numérique.

Références

Exemple :

Références

[1] M. SCHATZMAN, Analyse numérique, Masson, 1991

```
========================================================================

Méthode de Newton pour le calcul des racines de fonction y = x^2-alpha

Valeur de alpha         =  2.00000000e+000

Pas de discrétisation   =  1.00000000e-003

Nombre max d'itérations =       1000

Approximation en fonction de la valeur de la précision

     Précision     Racine x              f(x)            Nb iter

2.10734243e-008  1.41421356e+000    4.44089210e-016         9

Vitesse de convergence

Précision =  2.10734243e-008

Itération       x
     1  1.00000000e-001
     2  1.00500000e+001
     3  5.12450249e+000
     4  2.75739214e+000
     5  1.74135758e+000
     6  1.44494338e+000
     7  1.41454033e+000
     8  1.41421360e+000
     9  1.41421356e+000

========================================================================
```

TABLE 3 – Résultats numériques pour 2

A Programme

N.B. Le listing des programmes figure ici à titre d'exemple, afin de vous donner une idée de la manière à écrire des programmes *modulaires* et *suffisamment commentés*. Il ne doit pas faire partie de votre rapport. Par contre les programmes, prêts à fonctionner, doivent être envoyés avec votre rapport.

```
//  RACINES D'UNE FONCTION PAR LA MÉTHODE DE NEWTON

//  Ce programme permet de calculer la racine carrée d'un nombre alpha
//  en utilisant la méthode de Newton.


//************************************************************************
//
//                              Partie Initialisations
//                              ======================
//
//  Saisie des valeurs
//    - du paramètre  alpha de la fonction,
//    - du pas de discrétisation dt,
//    - du nombre d'itérations maximal maxIter,
//
//  Définition de la fonction f et de ses deux premières dérivées fdot et f2dot.
//
//************************************************************************


//  On peut définir une fonction et l'enregistrer dans un fichier.
//  Pour pouvoir faire appel à cette fonction on la déclare dans le programme principal
//  à l'aide de la commande "getf".
//  Ici on définit de cette façon la fonction f(x)=x*x-alpha

   getf('f.sci');

//  Une autre façon de définir une fonction, directement dans le corps du programme principal.
//  Ici on définit de cette façon les dérivées 1e et 2e de la fonction f(x).

   deff('y=fdot(x,alpha)','y=2*x');
   deff('y=f2dot(x,alpha)','y=2');

//  On forme la fonction g(x) qui défini le calcul récurrent des approximations successives.

   deff('y=g(x,alpha)','y=x-f(x,alpha)/fdot(x,alpha)');
   deff('y=gdot(x,alpha)','y=f2dot(x,alpha)*f(x,alpha)/(fdot(x,alpha)*fdot(x,alpha))');

//************************************************************************

//  Le programme demande à l'utilisateur de saisir certaines informations.
//  Pour cela il existe deux méthodes: en ligne de commande de Scilab ou
//  à l'aide d'une fenêtre de dialog. La fonction lecParams utilise la 1e méthode.

   getf('lecParams.sci');
```

```
   [alpha, dt, maxIter] = lecParams();

   getf('plotGraphe.sci');    //  Fonction pour le tracé de la courbe y=f(x).

//*********************************************************************
//
//                              Première Partie
//                              ===============
//
//  Recherche d'un intervalle dans lequel se trouve une racine de la fonction
//
//*********************************************************************

   write(%io(2), "Le programme va verifier les conditions d''existence d''une racine "...
                 " dans l''intervalle indiqué");

//  Le choix de l'intervalle  dans lequel le programme va chercher une racine
//  est difficile. Nous avons  laissé ce choix entièrement à l'utilisateur.
//  Le programme aide l'utilisateur en lui indiquant si la racine peut être trouvée ou non
//  dans l'intervalle qu'il a choisi.
//  Si la condition f(a)*f(b)<0 n'est pas vérifiée, l'utilisateur peut changer d'intervalle
//  autant de fois qu'il le souhaite.
//  le programme prévoit deux possibilités de sortie de cette phase :
//    soit le bon intervalle est trouvé  (indicateur "test" positionné à 0)
//    soit l'utilisateur décide d'abandonner   (indicateur "decision" positionné
//                                                    à 0 par l'utilisateur)



//  Boucle sur les différentes valeurs de la précision des calculs

    write(%io(2), "    ");
    epsilon = sqrt(2* %eps)
    printf("Précision  = %10.8e\n", epsilon);

    test=1;
    decision = 1;

    [test, decision, a, b, x0] = recInt(test, decision);

    if ~decision
        write(%io(2), "FIN par abandon ");
      else
        write(%io(2), "L''intervalle a été trouvé. ");

//*********************************************************************
//
//                        Deuxième Partie
//                        ===============
//
//  Évaluation de la racine de la fonction
//
//*********************************************************************
```

```
        [tblPrec, tblConv] = evalRac(a, b, x0, alpha, dt, epsilon, maxIter);

        write(%io(2), "C''est fini! ");


//        Tracé du graphe de la fonction

        nuEcran = 0;
        plotGraphe(a, b, alpha, dt, nuEcran);

//  Ouverture du fichier de compte rendu "newton.crd"  en écriture

        nomFicCR = 'newton.crd';
        lst = mopen(nomFicCR, 'w');

//  Écriture du compte rendu

        ficCR(tblConv, tblPrec, lst);

//  Fermeture du fichier de compte rendu "newton.crd"

        file('close',lst);
    end;

//  FIN DU PROGRAMME
//*************************************************************************

function y = f(x,alpha);

//*************************************************************************
//
//   Fonction y = f(x)
//
//  Variables et tableaux :
//                en entrée : x     = valeur de la variable indépendante x
//                            alpha = valeur dont on cherche la racine carrée
//                en sortie : y     = erreur de l'approximation
//
//*************************************************************************

    y=x**2-alpha;

//*************************************************************************

function [alpha, dt, maxIter] = lecParams();

//*************************************************************************
//
//  Fonction qui permet la saisie des paramètres du problème
//  en utilisant la ligne de commande Scilab
//
//  Variables et tableaux :
//                en sortie : alpha   = valeur dont on cherche la racine carrée
//                            dt      = pas de discrétisation
```

```
//                            maxIter = nombre max d'itérations autorisée
//
//*************************************************************************

//  La fonction "write" permet d'afficher un message sur l'écran
//  La fonction "read" permet de lire  des valeurs saisies au clavier

   write(%io(2), "Donnez le nombre dont vous voulez calcule la racine carré ");
   alpha = read(%io(1),1,1, '(e10.0)');

   write(%io(2), "Quelle pas de discrétisation dt souhaitez vous?");
   dt = read(%io(1),1,1, '(e10.0)');


   write(%io(2), "Entrez un nombre d''itérations à ne pas dépasser ");
   maxIter = read(%io(1),1,1, '(e10.0)');


//*************************************************************************

function[test, decision, a, b, x0] = recInt(test, decision)

//*************************************************************************
//
//  Fonction qui calcule un intervalle dans lequel se trouve
//  une racine de multiplicité impaire. Dans ce cas
//  la variable test est positionné à 1.
//  La recherche peut être abandonnée à la demande de l'utilisateur.
//  Dans ce cas la variable test est positionnée à 0.
//
//  Variables et tableaux :
//                en sortie : a, b     = les bornes de l'intervalle
//                            x0       = point initial
//      en entrée et sortie : test     = variable indiquant si la recherche
//                                       a été abandonnée ou non. (N.B. En entrée
//                                       test doit être positionné à 1)
//
//*************************************************************************

        while test&decision

//           Une autre possibilité d'interaction avec l'utilisateur :
//           une fenêtre de saisie réalisée par la commande "x_mdialog"
//           avec affichage, le cas échéant, des valeurs par défaut.

             data=x_mdialog('Choix  de l''intervalle de recherche',...
             [' Entrez a';' Entrez b';'Point initial';],['0';'1';'0.1';]);

//           Récupération des données de la fenêtre de dialogue

            D=evstr(data);
            a=D(1);
            b=D(2);
            x0=D(3);
```

```
            if f(a,alpha)*f(b,alpha)>0
                write(%io(2), "La condition n''est pas vérifiée. ");
                write(%io(2), "Pour continuer tapez 1 pour abandonner tapez 0 ");
                decision= read(%io(1),1,1, '(e10.0)');
                test=1;
              else write(%io(2), "La condition est  vérifiée. ");
                test=0;
            end;

        end;  // Boucle "while"

//***************************************************************************


function [tblPrec, tblConv] = evalRac(a, b, x0, alpha, dt, epsilon,
maxIter)

//***************************************************************************
//
//   Fonction qui calcule la valeur de la racine
//
//  Variables et tableaux :
//                en sortie : a, b     = les bornes de l'intervalle
//                            x0       = point initial
//                            alpha   = valeur dont on cherche la racine carrée
//                            dt      = pas de discrétisation
//                            epsilon = valeur de précision (dépend de la machine)
//                            maxIter = nombre max d'itérations autorisée
//                            iterPrec = numéro de l'itération.
//                en sortie : tblPrec  = tableau contenant la précision de la solution,
//                                       la solution, la valeur de la fonction et
//                                       le nombre d'itérations effectuées.
//                            tblConc  = tableau contenant la vitesse de convergence
//
//***************************************************************************


//  Calcul des approximations successives de la racine

            x=x0;
            tblConv(1) = x0;
            y1=g(x,alpha);
            iter = 1;
            while (abs(y1-x) > epsilon)&(iter < maxIter)
                x = y1;
                tblConv(iter+1) = x;
//                printf("approximation: x= %10.8e\n",x);
                y1=g(x,alpha);
                iter =iter+1;
            end;
            res = f(y1,alpha);
            tblPrec(1) = epsilon;
            tblPrec(2) = y1;
```

```
            tblPrec(3) = res;
            tblPrec(4) = iter;
            printf("Solution: x= %10.8e, valeur fct = %10.8e, Nb itér = %6.0f, Prec = %10.8e\n", ...
                    y1, res, iter, epsilon);


//************************************************************************

function [] = ficCR(tblConv, tblPrec, lst);

//************************************************************************
//
//   Ecriture du fichier de Compte Rendu
//
//  Variables et tableaux :
//                en entrée : tblConv, tblPrec = tableaux de compte rendu
//                            lst              = nom du fichier où sera stocké le compte rendu
//
//************************************************************************


    disp('Ecriture du fichier de Compte Rendu. Patientez svp.');
    fprintf(lst,'\\begin{verbatim}  \n');
    fprintf(lst,'  \n');
    fprintf(lst,'========================================================================== \n');
    fprintf(lst,'  \n');
    fprintf(lst,'Méthode de Newton pour le calcul des racines de fonction y = x^2-alpha \n');
    fprintf(lst,'  \n');
    fprintf(lst,'  \n');
    fprintf(lst,'Valeur de alpha         =  %10.8e\n',alpha);
    fprintf(lst,'  \n');
    fprintf(lst,'  \n');
    fprintf(lst,'Pas de discrétisation   =  %10.8e\n',dt);
    fprintf(lst,'  \n');
    fprintf(lst,'  \n');
    fprintf(lst,'Nombre max d''itérations = %10.0f\n',maxIter);
    fprintf(lst,'  \n');
    fprintf(lst,'  \n');
    fprintf(lst,'Approximation en fonction de la valeur de la précision \n')
    fprintf(lst,'  \n');
    fprintf(lst,'  \n');
    fprintf(lst,'     Précision     Racine x              f(x)            Nb iter \n');
    fprintf(lst,'  \n');
    fprintf(lst,'%10.8e  %10.8e    %10.8e    %6.0f \n', tblPrec(1),...
                tblPrec(2), tblPrec(3), tblPrec(4));
    fprintf(lst,'  \n');
    fprintf(lst,'  \n');
    fprintf(lst,'Vitesse de convergence \n')
    fprintf(lst,'  \n');
    fprintf(lst,'Précision =  %10.8e\n',tblPrec(1));
    fprintf(lst,'  \n');
    fprintf(lst,'Itération       x  \n');
    for i = 1:tblPrec(4)
        fprintf(lst,'%6.0f  %10.8e\n', i, tblConv(i));
```

```
    end;
    fprintf(lst,'  \n');
    fprintf(lst,'============================================================================  \n');
    fprintf(lst,'  \n');

//*************************************************************************


function [] = plotGraphe(a, b, alpha, dt, nuEcran);

//*************************************************************************
//
//  Tracé d'une fonction donné par f(.) dans l'intervalle [a, b]
//
//  Variables et tableaux :
//                en entrée : alpha   = valeur dont on cherche la racine carrée
//                            dt      = pas de discrétisation
//                            a,b     = extrémités de l'intervalle de variation pour f
//                            nuEcran = numéro d'écran pour le tracé
//
//*************************************************************************

//  Préparation de  graphes

//  Évaluation de l'axe des abscises

      t=a:dt:b;
//  Calcul pour chaque point de l'axe des abscises, de l'ordonnée correspondante

      for i=1:length(t)
          f_graph(i)=f(t(i),alpha);
      end
      zero_graph=zeros(1,length(t));

//  Initialisation d'une fenêtre graphique

      xset('window',nuEcran); // crée une fenêtre et lui associe un numéro
      xbasc();
      xselect();

//  La fonction plot2D trace ici deux courbes.
//  Chaque courbe est tracée point par point
//  et chaque point est défini par deux coordonnées : (x,y).
//  Ainsi une courbe est définie par deux tableaux : X qui regroupe les coordonnes x
//                                                     de tous les points de la courbe
//                                                et Y qui regroupe toutes les ordonnées.
//  S'il faut tracer plusieurs courbes on doit indiquer
//  en premier argument de plot2D  tous les tableaux des abscises
//  et en deuxième argument tous les tableaux des ordonnées.
//  Voir l'aide de Scilab pour plus d'informations sur plot2d


      plot2d([t;t]',[zero_graph;f_graph']',[5,2],"161","zero@f(x)",[2,10,2,10])
```

```
//*****************************************************************************
}
```