Université Pierre et Marie Curie
LM383
Année universitaire 2007-2008

Equations Differentielles, Méthodes de résolution

T. Capricelli et **D. Yakoubi**

TP 2 : Intégration numérique et résolution d'équations non linéaires

1. Intégration numérique.

- Ecrire un programme Scilab permettant d'intégrer la fonction sin sur $[0, \pi]$ par la méthode des trapèzes en fonction d'un pas de discrétisation $h = \pi/n$. Même question avec la méthode de Simpson.

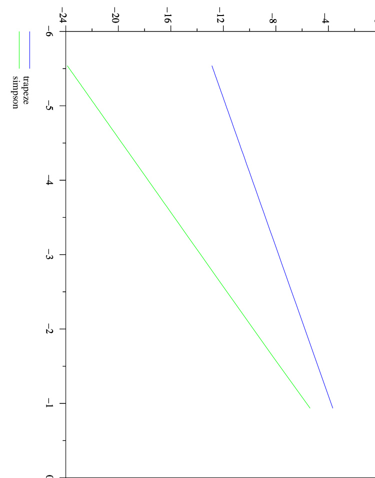
```
function v=trapeze(f,a,b,n)
  dx=(b-a)/n
  v=0
  for i=1:n-1
    v=v+f(a+i*dx)
  end
  v=(v+(f(a)+f(b))/2)*dx
  // v=dx*((f(a)+f(b))/2 + sum(f([1:n-1]*dx)) );
  // Version optimisee par vectorisation
endfunction

function v=simpson(f,a,b,n)
  n=2*n;
  // Surdiscretisation pour pouvoir considere les points
  // milieux comme etant a des coordonnees entieres
  dx=(b-a)/n;
  res1=0; // points impairs
  res2=0; // points pairs
  for i=1:2:n-1
    res1 = res1+f(a+i*dx);
  end
  for i=2:2:n-1
    res2 = res2+f(a+i*dx);
  end
  v=(f(a) + f(b) + 4*res1 + 2*res2)*dx/3;
  // v=(f(a)+f(b) + 4*sum(f([1:2:n-1]*dx)) + 2*sum(f([2:2:n-1]*dx)) )*dx/3;
  // Version optimisee par vectorisation
endfunction
```

- Comparer les résultats avec la valeur exacte pour différentes valeurs de h
- Estimer numériquement, en représentant graphiquement le logarithme de l'erreur pour différentes valeurs de $\log(h)$, les ordres de convergence des deux méthodes. Vérifier qu'on retrouve bien les ordres théoriques de convergence.

```
deff('y=f(x)','y=sin(x)');
a=0;
b=%pi;
n=100;
xd=zeros(n,1);
td=zeros(n,1);
```

```
sd=zeros(n,1);
for i=1:n
  ni=8*i;
  dh=(b-a)/ni;
  // nombre et taille des segments d'integration
  // ni augmente de 8 points a chaque iteration
  td(i)=log(norm(2-trapeze(f,0,b,ni)));
  sd(i)=log(norm(2-simpson(f,a,b,ni)));
  xd(i)=log(dh);
end
xbasc();
plot2d(xd,td,style=[2,2],leg="trapeze")
plot2d(xd,sd,style=[3,3],leg="simpson")
```



L'estimation des coefficients directeurs des droites permet d'obtenir une approximation de l'ordre des méthodes.

2. Nous allons étudier le comportement de la suite $x_n = f(x_{n-1})$ avec $f(x) = x^2 + c$

- Écrire une fonction *Scilab représentant la courbe* $(n; x_n)$.

```
//Voici une fonction elementaire qui stocke dans un vecteur x
//les differentes valeurs de xn pour n variant de 1 a nmax.
function [x]=iter(c,nmax,x0)
  x=zeros(nmax,1)
  x(1)=x0
  for i=2:nmax
    x(i)=x(i-1)^2+c
  end
endfunction
```

```
// Trace de la courbe pour un jeu de parametre c,nmax,x0 donne
plot2d([1:nmax],iter(c,nmax,x0))
```

- *Écrire un programme Scilab réalisant les calculs suivants*
  - *Pour N_c valeurs de c décrivant l'intervalle $[-2, 1/4]$ faire*
    - *calculer les N_{init} premiers termes de la suite x_n avec $x_0 = 0$*
    - *calculer les N_{suiv} termes suivants en les sauvant dans un tableau de taille N_{suiv}.*
    - *représenter dans une fenêtre graphique $[-2, -2, 1/4, 2]$ les points de la suite x_n sur la verticale de c*
  - *faire varier N_{init}, et N_{suiv} pour avoir une figure stable.*

```
function [ ]=iter2(ninit,nsuiv,cmin,cmax,nc)
  c=[cmin:(cmax-cmin)/(nc-1):cmax]'
  //on a pris ici nc points dans l'intervalle [cmin,cmax]
  x=zeros(nsuiv,1)
  xbasc();
  for j=1:nc
    xninit=0
    //on part de x_0=0
    for i=1:ninit
      xninit=xninit^2+c(j)
    end
    // cette boucle calcule successivement les ninit premiers termes
    // de la suite et ne retient que le dernier, stocke dans xninit
    x=iter(c(j),nsuiv,xninit)
    // on utilise alors la fonction iter pour calculer et
    //stocker les termes suivants.
    plot2d(c(j)*ones(nsuiv,1),x,style=0,rect=[cmin,-2,cmax,2])
    //et on affiche le tout pour chaque valeur de c
    // (en mode point grace a 'style=0')
  end
endfunction
```

3. Algorithme de Newton

- Écrire une fonction *Scilab* ***newton***

```
function [x,iter]=newton(x0,f,jacf,eps,itermax)
  //on commence par initialiser x par la donnee intiale x0
  //et on cacule le jacobien en ce point
  x=x0
  fx=f(x)
  g=jacf(x)
  iter=0
  while norm(fx)>eps & norm(g)>eps & iter<abs(itermax)
    //a chaque etape de la boucle while, on applique l agorithme
    //de Newton, comme decrit dans le sujet
    dx=g\fx
    x=x-dx
    fx=f(x)
    g=jacf(x)
```

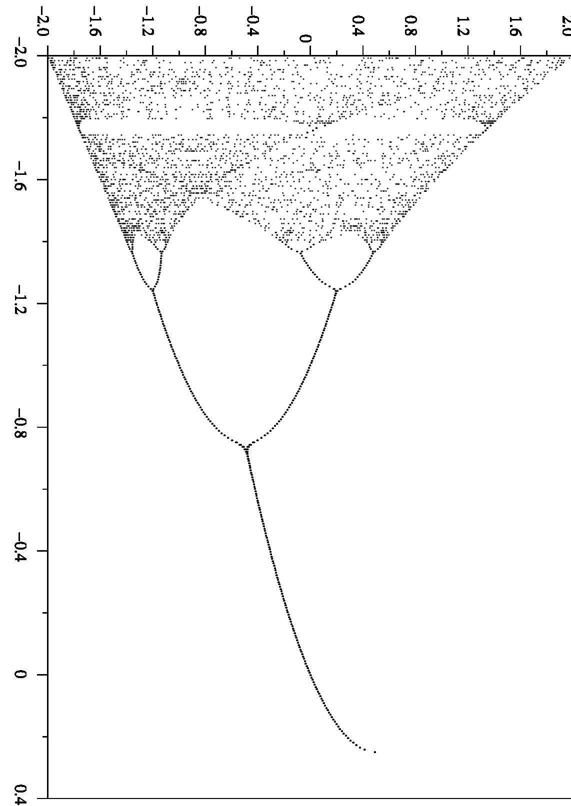


Figure 1: Exemple de tracé

```
    iter=iter+1
  end
  //message d'erreur si la methode n a pas converge avant itermax iterations
  //si itermax est negatif, on saute le message d'erreur
  if iter>= itermax & itermax > 0 & norm(f(x))>eps then
    printf("pas de convergence dans Newton")
  end
endfunction
```

- *Tester* **newton** *sur l'exemple suivant:*

$$y \ln y - x \ln x = 3$$
$$x^4 + xy + y^3 = a$$

```
function y=f(x)
```

```
  y=0*x
  y(1)=x(2)*log(x(2))-x(1)*log(x(1))-3
  y(2)=x(1)^4+x(1)*x(2)+x(2)^3-a
endfunction

function g=jacf(x)
  g=[-log(x(1))-1,1+log(x(2));4*x(1)^3+x(2),x(1)+3*x(2)^2]
endfunction

--> x0=[1;1];
-->sol=newton(x0,f,jacf,1e-8,100)
 sol  =
! - 0.5763519 + 1.1118236i !
!   1.5744907 - 0.6291920i !
-->f(sol)
 ans  =
   1.0E-14 *
!   0.0111022i             !
! - 0.0222045 - 0.0888178i !
```

- *Étude du comportement de l'algorithme de Newton pour résoudre $x^3 = 1$ dans $\mathbb{C}$.*
  - *Écrire la fonction $f(x) = x^3 - 1$ et son jacobien comme fonctions de $\mathbb{R}^2 \to \mathbb{R}^2$ et $\mathbb{R}^2 \to \mathbb{R}^{2\times 2}$.*

    ```
    function y=f2(x)
      //z = (x(1)+%i*x(2))^3-1;
      //y = [real(z);imag(z)];
      y = [x(1)*(x(1)^2-3*x(2)^2)-1;x(2)*(3*x(1)^2-x(2)^2)];
    endfunction

    function g=jacf2(x)
      g=[3*(x(1)^2-x(2)^2),-6*x(1)*x(2);6*x(1)*x(2),3*(x(1)^2-x(2)^2)]
    endfunction
    ```

  - *Pour x_0 parcourant le carré $[-1, 1] \times [-1, 1]$ par nx increments $d_x = 2/n_x$ sur l'axe réel et n_y incréments $d_y = 2/n_y$ sur l'axe imaginaire, calculer la solution de $x^3 = 1$ par l'algorithme de Newton.*
  - *Utiliser la fonction $Matplot$ pour représenter la matrice $racine$*

    ```
    function [M,Miter]=racine(nx,ny,eps,itermax)
      // Retourne aussi une matrice contenant le nombre d'iterations necessaire
      for i=1:nx
        for j=1:ny
          //on prend comme donnee initiale chaque point du maillage du
          //carre [-1,1]x[-1,1]:
          x0=[-1+i*2/nx;-1+j*2/ny]
          //on calcule l'iteree par la methode de Newton
          [x,iter]=newton(x0,f2,jacf2,eps,-itermax)
          //et on determine vers quelle racine la methode a converge
          if norm(x-[1;0])<eps then M(i,j)=1
          elseif norm(x-[-0.5;sqrt(3)/2])<eps then M(i,j)=2
          elseif norm(x-[-0.5;-sqrt(3)/2])<eps then M(i,j)=3
    ```

```
        else M(i,j)=4
        end
        Miter(i,j)=iter;
      end
    end
  endfunction
```

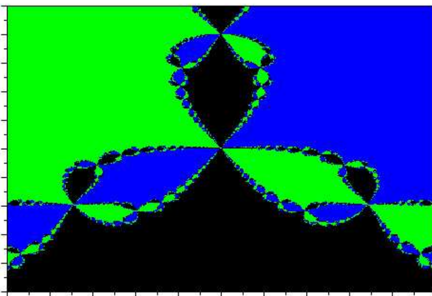


Figure 2: Zone de convergence

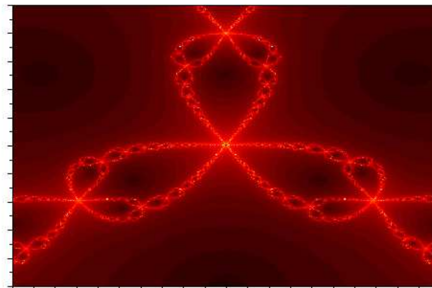


Figure 3: Nombre d'itérations

```
ndiv=400;
[M,M2]=racine(ndiv,ndiv,1e-8,100);
m=max(M2);
n= fix(3/8*m);
r=[(1:n)'/n; ones(m-n,1)];
g=[zeros(n,1); (1:n)'/n; ones(m-2*n,1)];
b=[zeros(2*n,1); (1:m-2*n)'/(m-2*n)];
h=[r g b];
xset("colormap",h)
xbasc(0);
xset("window",0);
Matplot(M2,rect=[0,0,ndiv,ndiv]);
xbasc(1);
xset("window",1);
```

```
Matplot(M,rect=[0,0,ndiv,ndiv])
```