TP - Méthodes numériques

Un premier exemple : la méthode d'Euler explicite

On va montrer comment mettre en oeuvre efficacement la méthode d'Euler pour la résolution d'équations différentielles. Pour une équation du type :

$$y'(t) = f(t, y(t)),$$

elle consiste, pour un pas h donné, à construire une suite d'approximations (y_n) de la solution y aux temps $t_n = nh$, par la formule

$$y_{n+1} = y_n + hf(t_n, y_n).$$

L'initialisation est fournie par la donnée de la condition initiale $y(0) = y_0$. Par la suite, on utilisera l'exemple modèle $y' = t - ty$ avec la condition initiale $y(0) = 2$, dont la solution exacte est donnée par $y(t) = 1 + e^{-\frac{t^2}{2}}$.

Pour définir cette fonction en Scilab, on écrit :

```
function yp=f(t,y)
yp=t-t*y;
endfunction
```

1. Le coeur du programme

   L'itération de la méthode d'Euler s'écrit simplement $y = y + h * f(t, y)$ si bien qu'une première version de la méthode d'Euler est la suivante :

```
function y=Euler(y0,N,T)
// Version 1
y=y0;t=0;
h=T/N;
for i=1:N
  y=y+h*f(t,y);
  t=t+h;
end
endfunction
```

2. Renvoi des arguments

   Bien sûr, l'appel de la fonction précédente $y = Euler(y0, N, T)$ ne fournit que l'approximation finale de $y(T)$, sans les valeurs intermédiaires... On y remédie en renvoyant la liste complète plutôt que la dernière valeur :

```
function liste_y=Euler(y0,N,T)
// Version 2
y=y0;liste_y=[y0];
t=0;
h=T/N;
for i=1:N
  y=y+h*f(t,y);
  t=t+h;
  liste_y=[liste_y,y];
end
endfunction
```

Il peut être aussi commode que la fonction renvoie les temps successifs où sont effectuées les approximations :

```
function [liste_y,liste_t]=Euler(y0,N,T)
// Version 3
y=y0;liste_y=[y0];
t=0;liste_t=[0];
h=T/N;
for i=1:N
  y=y+h*f(t,y);
  t=t+h;
  liste_y=[liste_y,y];
  liste_t=[liste_t,t];
end
endfunction
```

3. Programme principal - appel de la fonction

La possibilité de renvoyer à la fois les temps et les valeurs des approximations permet une utilisation très simple de la fonction :

```
// Parametres
T=2;
N=50;
y0=2;
// Calcul
[sol,tps]=Euler(y0,N,T);
// Graphique
plot(tps,sol)
```

Si l'on souhaite comparer l'approximation obtenue avec la solution exacte, on peut définir une fonction :

```
function y=yex(t)
y=1+exp(-t.^2/2);
endfunction
```

Noter l'utilisation de l'opérateur $t.^2$ afin d'élever terme-à-terme un vecteur au carré. Elle permet un appel unique pour l'évaluation de la solution exacte sur la subdivision :

```
plot(tps,sol)
plot(tps,yex(tps),'r')
```

Par ailleurs la méthode d'Euler programmée est indépendante de la dimension, pour peu qu'on impose aux vecteurs d'être écrits en colonne. Par exemple, pour résoudre le système différentiel suivant :

$$x'(t) = x(t)(3 - y(t))$$
$$y'(t) = y(t)(-2 + 2x(t))$$

il suffit de redéfinir la fonction comme suit

```
function yp=f(t,y)
yp=[y(1)*(3-y(2));y(2)*(-2+2*y(1))];
endfunction
```

et le programme principal suivant les trajectoires en fonction du temps, ainsi que le plan de phase :

```
// Parametres
T=10;
N=5000;
y0=[1;1];
// Calcul
[sol,tps]=Euler(y0,N,T);
// Graphique
subplot(2,1,1)
plot(tps,sol(1,:),tps,sol(2,:))
title('Solutions x(t) et y(t)')
subplot(2,1,2)
plot(sol(1,:),sol(2,:))
title('Plan de phase')
```

4. Erreur d'approximation - Ordre de convergence

   Si l'on souhaite mettre en évidence la convergence d'ordre 1 de la méthode d'Euler, on doit effectuer des simulations pour différentes valeurs du pas h. Il suffit d'ajouter une boucle extérieure à notre programme principal :

```
// Parametres
T=2;
y0=2;
liste_N=10:10:1000;
// Boucle sur N
liste_Err=[];
for N=liste_N
    // Calcul
    [sol,tps]=Euler(y0,N,T);
    // Evaluation de l'erreur
    erreur=norm(sol-yex(tps),'inf');
    // Mise a jour du tableau d'erreurs
    liste_Err=[liste_Err,erreur];
end
// Trace
close()
plot(liste_N,liste_Err,'o-')
title('Erreur en fonction de N')
xlabel('Nombre de points N')
ylabel('Erreur uniforme')

xclick();

clf
plot2d(liste_N,liste_Err,logflag='ll')
title('Erreur en fonction de N (log-log)')
xlabel('Nombre de points N')
ylabel('Erreur uniforme')

xclick();

[a,b,sig]=reglin(log(liste_N),log(liste_Err))
chaine=['Pente=',string(r(1))];
xstring(100,1e-3,chaine)
```

5. Passage de la fonction en paramètre

   On a vu plus haut qu'à chaque nouvelle équation différentielle, la fonction f devait être modifiée. Il peut être plus commode de définir une fonction par équation, et permettre un choix dans la fonction *Euler*. La fonction est alors passée en paramètre :

```
function [liste_y,liste_t]=Euler(f,y0,N,T)
// Version 4
y=y0;liste_y=[y0];
t=0;liste_t=[0];
h=T/N;
for i=1:N
  y=y+h*f(t,y);
  t=t+h;
  liste_y=[liste_y,y];
  liste_t=[liste_t,t];
end
endfunction
```

Exercices

1. Ordre de la méthode d'Euler point milieu

   On utilisera dans cet exercice l'exemple modèle précédent $y' = t - ty$ avec la condition initiale $y(0) = 2$, dont la solution exacte est donnée par $y(t) = 1 + e^{-\frac{t^2}{2}}$.

   On rappelle que la méthode d'Euler point milieu consiste à calculer la suite (y_n) d'approximations de la fonction y de la façon suivante :

$$y^{n+\frac{1}{2}} = y^n + \frac{h}{2} f(t^n, y^n)$$
$$y^{n+1} = y^n + hf\left(t^n + \frac{h}{2}, y^{n+\frac{1}{2}}\right)$$

   - Écrire une fonction $[sol, tps] = Eulerpointmilieu$ où sol sera la liste des valeurs y_n calculées avec la méthode d'Euler point milieu, et tps la liste des temps considérés.

   - Déterminer numériquement l'ordre de la méthode d'Euler point milieu.

2. Méthode de Newton

   La méthode de Newton permet d'approcher la solution d'une équation

$$F(x) = 0,$$

   où $F : \mathbb{R}^d \mapsto \mathbb{R}^d$ est différentiable. Partant d'une approximation initiale $x^0 \in \mathbb{R}^d$, la suite (x^k) est construite par récurrence

$$x^{k+1} = x^k - \left[F'(x^k)\right]^{-1} F(x^k).$$

   - Programmer une fonction scilab avec l'entête suivant,

$$function\ \ x = Newton(F, DF, x0, Tol, MaxIter)$$

     qui renvoie une approximation de la solution x. Les arguments d'entrée sont les suivants :

— F : nom de la fonction scilab définissant la fonction F,
— DF : idem pour la matrice jacobienne F',
— $x0$: vecteur initial $x^0 \in \mathbb{R}^d$,
— Tol : scalaire donnant la tolérance (critère d'arrêt sur l'incrément : $||x^{k+1} - x^k|| < Tol$)
— $MaxIter$: nombre maximal d'itérations (pour les problèmes de non-convergence).

Pour le critère d'arrêt, on utilisera une boucle *while*, selon le modèle suivant :

```
err = Tol+1;
iter = 0;
while (err>Tol)&(iter<itermax)
iter = iter +1;
 ...
end
```

- Tester cette fonction sur l'exemple monodimensionnel $F(x) = e^x - e$.

3. Méthode d'Euler implicite

A l'aide de la fonction précédente, programmer une fonction scilab qui met en oeuvre la méthode d'Euler implicite, dont l'itération s'écrit

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}).$$

- La fonction F qui définit le système à résoudre à chaque itération dépend de y_n et de h. Écrire une variante de la fonction $Newton$ précédente pour prendre en compte ces deux arguments supplémentaires.

- Comparer la méthode d'Euler explicite et la méthode d'Euler implicite sur le problème raide

$$y'(t) = -500(y(t) - cos(t)).$$

4. Méthode de Runge Kutta 4

Dans cet exercice, on cherche à résoudre numériquement le système différentiel associé à l'équation différentielle d'ordre 2 décrivant le mouvement d'un pendule pensant :

$$\begin{aligned} x'(t) &= y(t) \\ y'(t) &= -\frac{g}{l} sin(x(t)) \end{aligned}$$

On note $X = (x, y)$. On rappelle que la méthode de Runge Kutta d'ordre 4 consiste à calculer la suite (X_n) d'approximations de la fonction X de la façon suivante :

$$\begin{aligned} k_1 &= hf(t^n, X^n) \\ k_2 &= hf\left(t_n + \frac{h}{2}, X_n + \frac{k_1}{2}\right) \\ k_3 &= hf\left(t_n + \frac{h}{2}, X_n + \frac{k_2}{2}\right) \\ k_4 &= hf(t^n + h, X^n + k_3) \\ X^{n+1} &= X^n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \end{aligned}$$

- Écrire une fonction $[sol, tps] = RK4$ où sol sera la liste des valeurs X_n calculées avec la méthode de Runge Kutta 4, et tps la liste des temps considérés.

- Appliquer la méthode d'Euler explicite et la méthode de Runge Kutta 4 à l'équation du pendule pesant. Tracer dans les deux cas l'angle $y(t)$ obtenu en fonction du temps, et le portrait de phase ($x(t)$ en fonction de $y(t)$).