# **CHAPITRE 3**

**Equations différentielles ordinaires**

1. **Présentation des différentes méthodes**
   1. **Schéma à un pas**

On considère la fonction solution d’un problème de Cauchy, c’est-à-dire une équation différentielles munie d’une condition initiale en t = 0 :

On suppose que f vérifie les conditions du théorème de Cauchy-Lipschitz, ce qui garantit l’existence et l’unicité de la solution de . Pour on considère une discrétisation uniforme de l’intervalle sous la forme d’une suite avec définie par où . Un schéma à un pas a pour but de construire une suite avec approchant les valeurs de la solution aux points , c’est-à-dire ( avec , en considérant l’équation de récurrence :

Nous nous intéresserons principalement à quatre schémas :

1. Schéma d’Euler

(

1. Schéma d’Euler-Cauchy
2. Schéma du point milieu
3. Schéma du Runge et Kutta d’ordre 4
   1. **Les méthodes de résolutions utilisées en pratique**

On va comparer les schémas que on présenter au-dessus. On considère l’équation différentielle suivante :

La solution exacte de cette équation est

Voici comment organiser le travail :

1 . Ecrire une fonction Scilab calculant le second membre de l’équation différentielle .

function **yprime** = f1(**t**, **y**)

**yprime** = -**t** \* **y** + **t**;

endfunction

2 . Ecrire une fonction Scilab acceptant comme argument , un vecteur contenant avec et l’identificateur de la fonction Scilab calculant , et renvoyant dans un vecteur pour données par chaque Schéma :

1. Schéma d’Euler

function **y** = euler(**y0**, **t**, **f**)

n = length(**t**);

**y** = zeros(1, n);

h = **t**(2) - **t**(1);

**y**(1) = **y0**;

for i = 1 : n-1

**y**(i+1) = **y**(i) + h \* **f**(**t**(i), **y**(i));

end

endfunction

t = linspace(0, 4, 10);

y0 = 0;

y = euler(y0,t,f1);

1. Schéma d’Euler – Cauchy

function **y**=eulerCauchy(**y0**, **t**, **f**)

n = length(**t**);

**y** = zeros(1, n);

h = **t**(2) - **t**(1);

**y**(1) = **y0**;

for i = 1 : n-1

k1 = **f**(**t**(i), **y**(i));

k2 = **f**(**t**(i) + h, **y**(i)+h \* k1);

**y**(i+1) = **y**(i) + h/2 \* (k1+k2);

end

endfunction

t = linspace(0, 4, 10);

y0 = 0;

plot(t, eulerCauchy(y0, t, f1));

1. Schéma du point milieu

function **y** = pointMilieu(**y0**, **t**, **f**)

n = length(**t**);

**y** = zeros(1, n);

h = **t**(2) - **t**(1);

**y**(1) = **y0**;

for i = 1 : n-1

k1 = **f**(**t**(i), **y**(i));

k2 = **f**(**t**(i) + h/2,**y**(i) + h \* k1/2);

**y**(i+1) = **y**(i) + h \* k2;

end

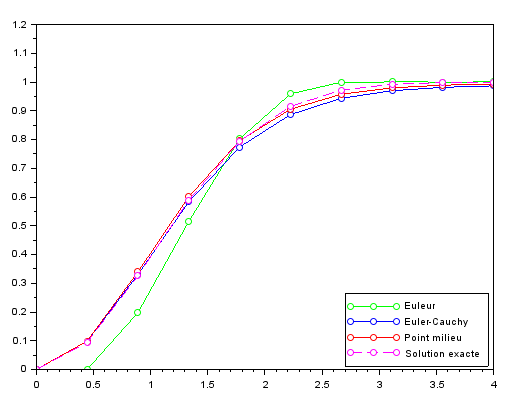
endfunction

t = linspace(0, 4, 10);

y0 = 0;

plot(t,pointMilieu(y0,t,f1));

3 . Représenter graphiquement les approximations superposées à la solution exacte pour des valeurs croissantes de N :



*Différentes solutions obtenues*

Remarque : Visiblement, la solution approchée par schéma du point milieu est plus approximative que la solution exacte.

* 1. **La macro *ode* de Scilab**

La macro *ode* de Scilab permet de resoudre l’équation différentielle ordinaire. Voici par exemple ce qu’il faut écrire pour l’équation  :

function **yprime** = f1(**t**, **y**)

**yprime** = - **t** \* **y** + **t**;

endfunction

t = linspace(0, 4, 10);

y0 = 0;

y = ode(y0, 0, t, f1);

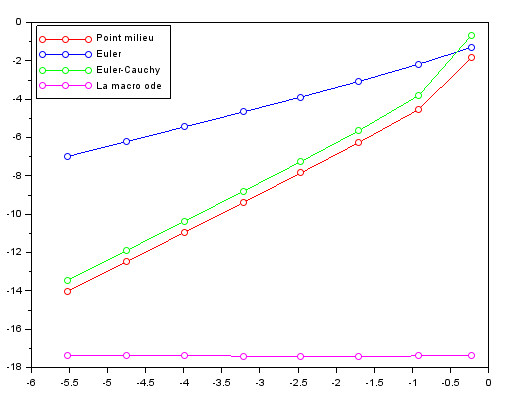
* 1. **Evaluation de l’ordre de l’erreur des schémas**

Une fonction d'erreur de convergence est définie comme suit :

avec

Pour chacun des schémas et ainsi la macro *ode*, on calcule l’erreur de convergence pour

et représente graphiquement en fonction de en utilisant une échelle logarithmique pour les abscisses et les ordonnées.



*Courbe représentation des fonctions d'erreurs pour les trois méthodes*

Puis, pour déterminer l'ordre d'erreur, on effectue une régression linéaire sur chacune

des courbes avec la macro *reglin* de Scilab pour en déterminer la pente. On trouve :

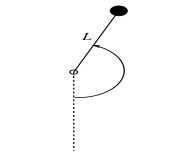
* L ' ordre de convergence du schéma d ' Euler est de : 1.0635769
* L ' ordre de convergence du schéma d ' Euler - Cauchy est de : 2.2786725
* L ' ordre de convergence du schéma du point milieu est de : 2.2034476
* L ' ordre de convergence de la macro *ode* est de : -0.0041744

Comme nous l'avons constaté avant, les schémas d'Euler et d'Euler-Cauchy rétrograde sont peu

précises, pour un calcul approximatif il faut utiliser la macro *ode*

1. **Application : Le pendule**

Le pendule considéré est représenté dans la figure. Nous supposerons que la tige reliant un point de masse M à l’axe de rotation est de masse négligeable devant . Nous choisissons de mesurer la déviation du pendule verticale d’équilibre stable par l’angle



*Le pendule considérer*

Si l’on applique les relations de la dynamique pour les solides en rotation autour d’un axe, on obtient l’équation différentielle suivante :

La valeur de donne la déviation initiale du pendule, et on a considéré que la vitesse angulaire

initiale est nulle.

Cette équation est non-linéaire à cause de , et sa résolution est difficile. Cependant, si l’écart angulaire est faible, sa mesure en radians est très peu différente de celle de . Si on peut se contenter d’une solution approchée, nous pouvons linéariser l’équation en confondant et on obtient l’équation différentielle suivante, où a été remplacé par pour bien montrer qu’il ne s’agit plus du même problème :

Il est assez élémentaire de montrer que la solution de cette équation différentielle est la

Fonction

En posant

On obtient alors

=

On va donc comparer la solution du modèle linéarisé avec l’approximation donnée par la macro

*ode* de Scilab de la solution du système d’équations différentielles pour plusieurs valeurs de

function **dydt**=fpendule(**t**, **y**)

theta = **y**(1);

thetaprime = **y**(2);

**dydt** = [thetaprime

-g/L\*sin(theta)];

endfunction

clf;

t = linspace(0, 10, 1000);

L = 1;

g = 9.81;

theta0 = %pi/32;

y = ode([theta0; 0], 0, t, fpendule);

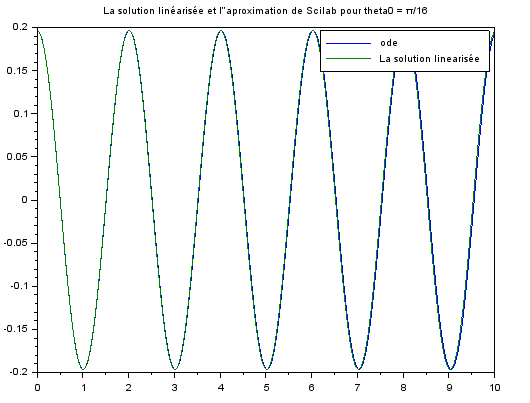
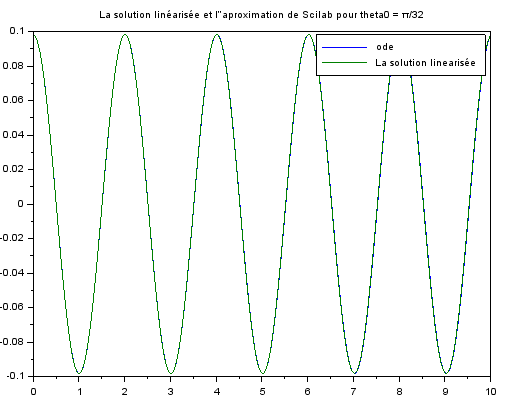
theta = y(1, :);

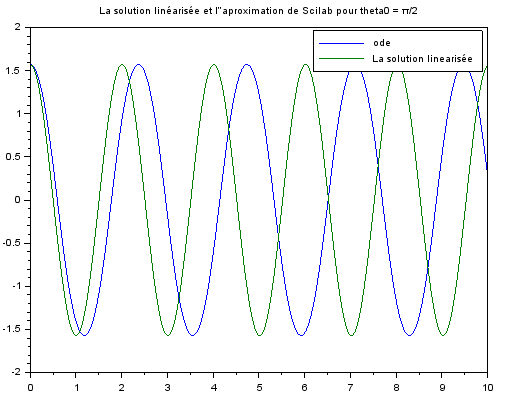
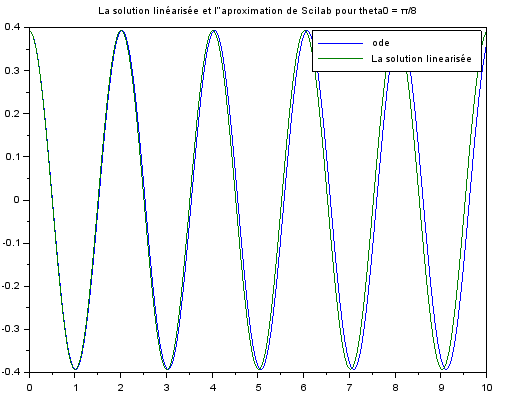
plot(t, theta, t, theta0 \* cos(t \* sqrt(g/L)));

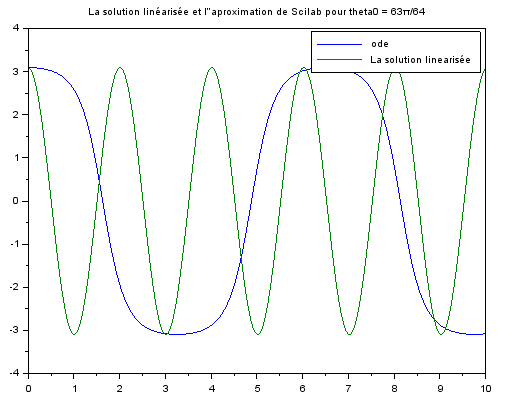
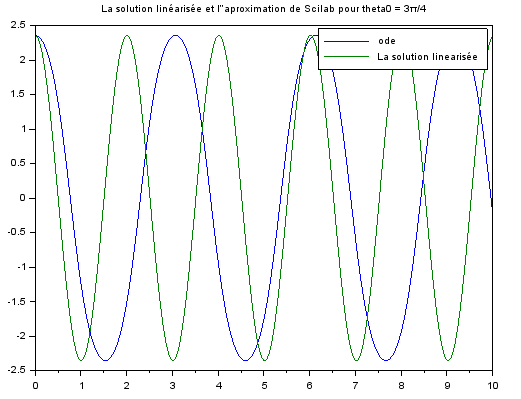
t = 'La solution linéarisée et l’’aproximation de Scilab pour theta0 = π/32';

title(t);

legend('ode','La solution linearisée');







*Comparaison de la solution du modèle linéarisé avec l’approximation donnée par la macro ode de Scilab pour*

Comme nous pouvons assez facilement le constater dans la figure au-dessus, plus le temps est éloigné du temps initial, plus la fiabilité de la solution linéarisé décroit. De même, plus, l’écart angulaire initial est élévé, moins le recours à la solution linéarisé s’avère judicieuse.

1. **Systèmes dynamiques**
   1. **Rappels de cours**

**Définition 3.1.1**. Un système autonome est un cas particulier important d’un système d’équations différentielles où la variable n’apparait pas dans le système d’équations fonctionnelles. Soit

**Définition 3.1.2.** Le vecteur \* est un point d’équilibre si

\*\*

Les points d’équilibre de l’équation sont caractérisés par ∗**Définition 3.1.3.** Le point d’équilibre ∗ est stable si

\*\*

**Définition 3.1.4.** Un point d’équilibre ∗ qui n’est pas stable est dit instable et on a

*\*\**

**Définition 3.1.5.** Le point d’équilibre ∗ est asymptotiquement instable s’il est stable et si

\*

**Propriété 2.2.1.** Considérons le système

Le système au-dessus est :

* Instable si au moins une valeur propre de A est à partie réelle strictement positive.
* Stable si toutes les valeurs propres de A sont à parties réelle négative.
* Asymptotiquement stable si toutes les valeurs propres de A sont à parties réelle strictement négative.
  1. **Equation de Duffing**
     1. **Système autonome**

On considère l’équation différentielle

avec

Tout d’abord, on commence par mettre cette équation différentielle sous forme d’un système d’équations différentielles du premier ordre en temps :

En posant :

=

On obtient alors :

=

On résout ensuite l’équation en vue d’obtenir les points d’équilibre  
\*}

On calcule ensuite la matrice Jacobienne de et on obtient

Enfin, on calcule les valeurs propres de la matrice Jacobienne aux différents points d’équilibre en vue de connaître les propriétés :

* Au point

On obtient comme valeurs propres

,

Avec on a and . Donc est un point d'équilibre instable.

* Au point

On obtient les mêmes valeurs propres :

,

Toutes les valeurs propres sont à partie réelle strictement négative, Donc les points , sont des points d'équilibre asymptotiquement stable.

On peut utiliser la fonction ode de Scilab pour dessiner le portrait de phase des systèmes dynamiques en superposant le comportement des solutions avec des conditions initiales  :

function **Xprime**=f(**t**, **X**)

**Xprime** = [ **X**(2)

-k \* **X**(2) + **X**(1) - **X**(1)^3];

endfunction

k = 0.15;

t = linspace(0, 200, 1000);

clf;

for X10 = 0 : 1 : 3

for X20 = 0 : 1 : 3

X0 = [X10; X20];

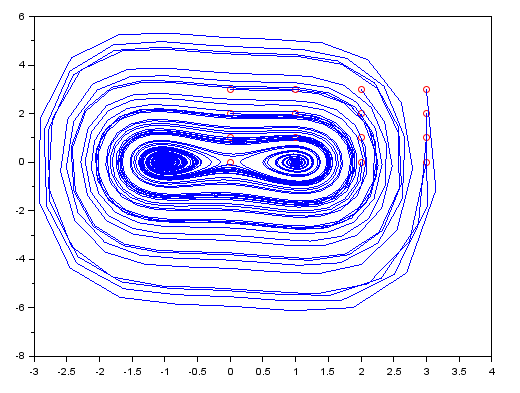
X = ode(X0, 0, t, f);

plot(X0(1), X0(2), 'or');

plot(X(1,:), X(2,:));

end

end



*Portrait de phase de l’équation Duffing autonome*

* + 1. **Système non autonome**

On considère maintenant l’équation différentielle

Avec et

Tout d’abord, on commence par mettre cette équation sous forme d’un système d’équations différentielles du premier ordre en temps de la forme

En posant :

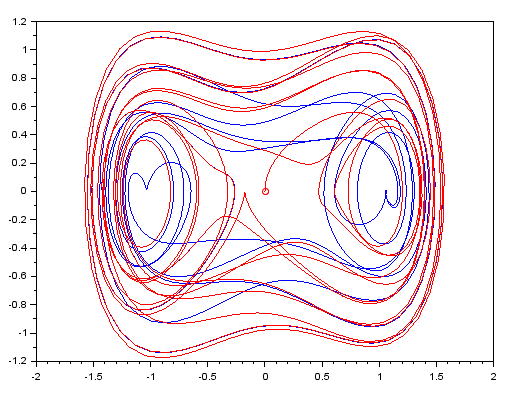
=

On obtient alors :

=

On va calculer avec la solution de cette équation différentielle pour en prenant les conditions initiales :

1. . La solution figure en bleu sur la figure au-dessous.
2. . La solution figure en rouge sur la figure au-dessous.



*Portrait de phase de l’équation Duffing non autonome*

Les points des points d’équilibre instable. En effet, les trajectoires semblent s’écarter de ces deux points. De plus, ces trajectoires semblent atteindre des cycle limite. Un cycle limite est une solution temporellement périodique qui est indépendante des conditions initiales et qui possède une fréquence intrinsèque au système indépendante des conditions initiales. Ici, c’est bien le cas, les trajectoires pour la première condition initiale et la seconde condition initiale semblent atteindre une solution temporellement périodique et possédant une fréquence intrinsèque. Cette dernière est bien indépendante des conditions initiales puisque la courbe bleue et le courbe rouge de la

figure au-dessus semblent identiques.

* 1. **Equation de Van de Pol**

On considère l’équation différentielle

Tout d’abord, on commence par mettre cette équation sous forme d’un système d’équations différentielles du premier ordre en temps de la forme

En posant :

=

On obtient alors :

=

On résout ensuite l’équation en vue d’obtenir les points d’équilibre

\*

On calcule ensuite la matrice Jacobienne de et on obtient

Enfin, on calcule le valeur propre de la matrice Jacobienne au point d’équilibre et on obtient alors :

,

pour , on a > 0 et donc est un point d'équilibre instable.

Exemple de l’équation de Van der Pol pour

function **Xprime**=f(**t**, **X**)

**Xprime** = [ **X**(2)

u\*(1-**X**(1).^2)\***X**(2)-**X**(1)];

endfunction

u = 1;

t = linspace(0,10,1000);

clf;

for X10 = -3 : 0.5 : 3

for X20 = -3 : 0.5 : 3

X0 = [X10; X20];

X = ode(X0,0,t,f);

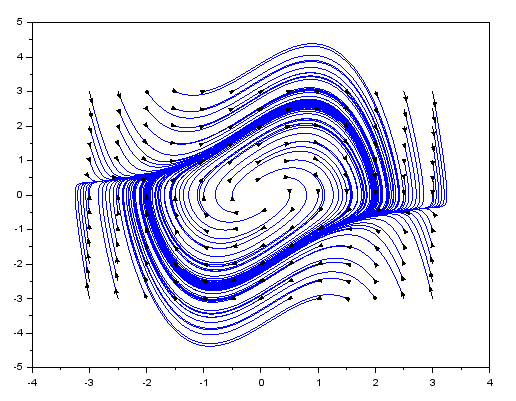
plot(X(1,:),X(2,:));

end

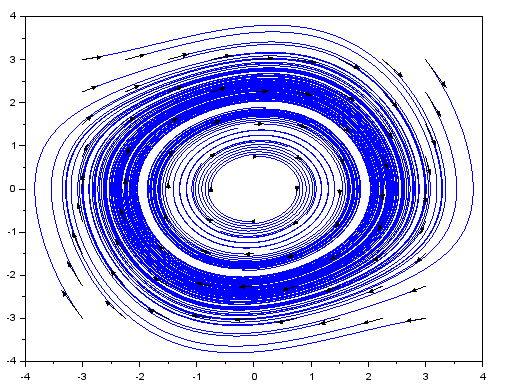
end

fchamp(f,0,(-3 : 0.

5 : 3),(-3 : 0.5 : 3));



Portrait de phase de l’équation de Van der Pol pour



Portrait de phase de l’équation de Van der Pol pour

Visiblement, ces deux figures nous confirment que est un point d’équilibre instable. En effet, les trajectoires proches de ce point s’éloignent de celui-ci dans les deux figures. De plus, on peut observer un cycle limite pour l’oscillateur de Van der Pol. En effet, toutes les trajectoires tendent à former une figure fermée : le système de Van der Pol a tendance à maintenir les oscillations.

* 1. **Système proie-prédateur (Lokta-Volterra)**

On considère maintenant le système :

Avec ,

Tout d’abord, on commence par mettre cette équation sous forme d’un système d’équations différentielles du premier ordre en temps de la forme

En posant :

=

On obtient alors :

=

On résout ensuite l’équation en vue d’obtenir les points d’équilibre

\*

On calcule ensuite la matrice Jacobienne de et on obtient

Enfin, on calcule les valeurs propres de la matrice Jacobienne aux différents points d’équilibre en vue de connaître les propriétés :

* Au point

On obtient comme valeurs propres

et

Avec les conditions initiales, on a and . Donc est un point d'équilibre instable.

* Au point

on obtient comme valeurs propres

Toutes les valeurs propres sont à partie réelle negative, est donc un point d’equilibre stable.

function **Xprime**=f(**t**, **X**)

**Xprime** = [ **X**(1)\*(a-b\***X**(2))

-**X**(2)\*(c-d\***X**(1))];

endfunction

a = 2;

b = 10^(-3);

c = 10;

d = 2\*10^(-3);

t = linspace(0,0.5,100);

clf;

for X10 = -400 : 200 : 10000

for X20 = -400 : 200 : 5000

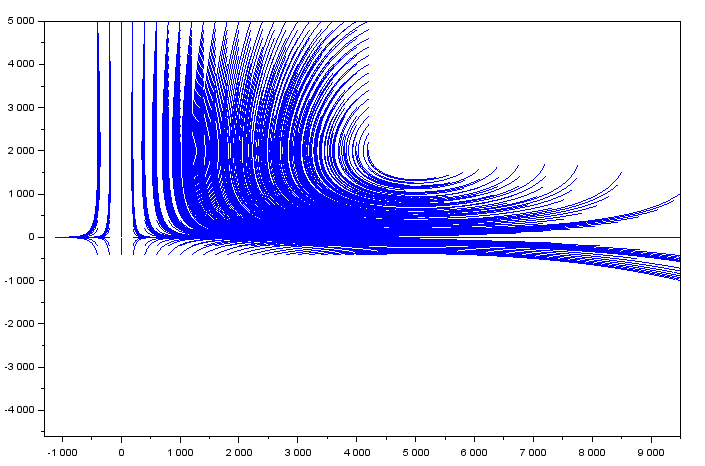
X0 = [X10; X20];

X = ode(X0,0,t,f);

plot(X(1,:),X(2,:));

end

end

****

Portrait de phase du système proie-prédateur(Lokta-Volterra)

**Système de Lorenz**

On considère le système

Où , sont des nombres positifs.