Équations différentielles

Cours de L3 par Frédéric Hélein [1], janvier–avril 2021

Mardi 4 mai 2021

6.3 Algorithme du point milieu

L'idée à la base de cette méthode est de chercher à améliorer l'algorithme d'Euler, fondé sur l'approximation $z(t+h) \simeq z(t) + hz'(t)$, en utilisant plutôt l'approximation $z(t+h) \simeq z(t) + hz'(t+\frac{h}{2})$. Car en effet cette approximation s'avère être meilleure. Par exemple, dans le cas où z est une fonction polynomiale de la forme $z(t) = az^2 + bz + c$, on a

$$z(t_0) + z'(t_0)(t_1 - t_0) = z(t_1) - a(t_1 - t_0)^2 \neq z(t_1) \text{ (sauf si } a = 0)$$

mais

$$z(t_0) + z'\left(\frac{t_0 + t_1}{2}\right)(t_1 - t_0) = z(t_1)$$

À présent essayons de mettre en œuvre cette idée pour un champ de vecteur X de classe $\mathcal{C}^2$. Soit z une solution de $\frac{dz}{dt} = X(t,z)$. Ecrivons les développements de Taylor

$$\begin{array}{r|rcll} 1 & z(t+h) & = & z(t) + & hz'(t) + \frac{h^2}{2}z''(t) + \frac{h^3}{6}z'''(\tau_1), \quad \text{où } \tau_1 \in]t, t+h[\\ -h & z'(t+\frac{h}{2}) & = & & z'(t) + \frac{h}{2}z'(t) + \frac{1}{2}\frac{h^2}{4}z'''(\tau_2), \quad \text{où } \tau_2 \in]t, t+h/2[\end{array}$$

En sommant les deux côtés, avec les coefficients 1 et $-h$ comme indiqué, on obtient

$$z(t+h) - hz'(t+\frac{h}{2}) = z(t) + \frac{h^3}{6}z'''(\tau_1) - \frac{h^3}{8}z'''(\tau_2)$$

Comme X est $\mathcal{C}^2$, z est $\mathcal{C}^3$ et donc $z'''(\tau_2) = z'''(\tau_1) + o(|\tau_1 - \tau_2|^0) = z'''(\tau_1) + o(1)$, donc $z(t+h) = z(t) + hz'(t+\frac{h}{2}) + \left(\frac{h^3}{6} - \frac{h^3}{8}\right)z'''(\tau_1) + o(h^3)$ et finalement

$$z(t+h) = z(t) + hz'\left(t + \frac{h}{2}\right) + \frac{h^3}{24}z'''(\tau_1) + o(h^3)$$

En utilisant le fait que $z' = X(t,z)$, on a donc

$$z(t+h) = \underbrace{z(t) + hX\left(t + \frac{h}{2}, z\left(t + \frac{h}{2}\right)\right)}_{\text{approximation}} + \frac{h^3}{24}z'''(\tau_1) + o(h^3) \tag{1}$$

La bonne nouvelle est qu'il n'y a plus de terme en h^2 et qu'on a une chance d'obtenir une méthode d'ordre 2.

1. Université de Paris, Licence 3 de Mathématiques, helein@math.univ-paris-diderot.fr

6.3.1 Estimation de $z(t+\frac{h}{2})$

Une difficulté apparaît cependant, car on a besoin d'estimer $z(t+\frac{h}{2})$ dans $X\left(t+\frac{h}{2}, z(t+\frac{h}{2})\right)$. Cela nécessite des développements de Taylor supplémentaires :

$$z\left(t+\frac{h}{2}\right) = z(t) + \frac{h}{2}z'(t) + \frac{1}{2}\left(\frac{h}{2}\right)^2 z''(\tau_3), \qquad \text{où } \tau_3 \in]t, t+h/2[$$

et donc, en utilisant l'équation $z' = X(t,z)$,

$$z\left(t+\frac{h}{2}\right) = z(t) + \frac{h}{2}X(t,z(t)) + \frac{h^2}{8}z''(\tau_3)$$

D'où

$$X\left(t+\frac{h}{2}, z\left(t+\frac{h}{2}\right)\right) = X\left(t+\frac{h}{2}, z(t)+\frac{h}{2}X(t,z(t))\right) + \sum_{i=1}^{n}\frac{\partial X}{\partial x_i}\left(t+\frac{h}{2}, z_\theta(t)\right)\frac{h^2}{8}z_i''(\tau_3)$$

où $z_\theta(t) := \theta z(t+\frac{h}{2}) + (1-\theta)\left(z(t)+\frac{h}{2}X(t,z(t))\right)$, avec $\theta \in]0,1[$. Nous écrivons la dernière identité

$$X\left(t+\frac{h}{2}, z\left(t+\frac{h}{2}\right)\right) = X\left(t+\frac{h}{2}, z(t)+\frac{h}{2}X(t,z(t))\right) + \frac{h^2}{8}\langle dX_\theta, z''(\tau_3)\rangle$$

où $dX_\theta := dX_{\left(t+\frac{h}{2}, z_\theta(t)\right)}$.

6.3.2 Formule finale

En injectant l'expression obtenue précédemment dans (1), on obtient

$$z(t+h) = z(t) + hX\left(t+\frac{h}{2}, z(t)+\frac{h}{2}X(t,z(t))\right) + \frac{h^3}{8}\langle dX_\theta, z''(\tau_3)\rangle + \frac{h^3}{24}z'''(\tau_1) + o(h^3) \tag{2}$$

Le reste $\frac{h^3}{8}\langle dX_\theta, z''(\tau_3)\rangle + \frac{h^3}{24}z'''(\tau_1) + o(h^3)$ peut être estimé comme suit :

(i) l'équation $z' = X(t,z)$ entraîne par dérivation $z'' = X^{(1)}(t,z)$ (où $X^{(1)}(t,z) = \frac{\partial X}{\partial t}(t,z) + \sum_{i=1}^{n}\frac{\partial X}{\partial x^i}(t,z)\frac{dz^i}{dt}(t)$), puis $z''' = X^{(2)}(t,z)$;

(ii) en supposant que X est bornée dans $\mathcal{C}^2(I\times\Omega, \mathbb{R}^n)$, on a $\|X^{(1)}(t,x)\| \le M^{(1)}$ et $\|X^{(2)}(t,x)\| \le M^{(2)}$, $\forall (t,x) \in I\times\Omega$;

(iii) donc $\|\langle dX_\theta, z''(\tau_3)\rangle\| = \|\langle dX_\theta, X^{(1)}(\tau_3, z(\tau_3))\rangle\| \le \|dX\|M^{(1)}$ et $\|z'''(\tau_1)\| = \|X^{(2)}(\tau_1, z(\tau_1))\| \le M^{(2)}$;

on en déduit

$$\left\|\frac{h^3}{8}\langle dX_\theta, z''(\tau_3)\rangle + \frac{h^3}{24}z'''(\tau_1) + o(h^3)\right\| \le h^3\left(\frac{\|dX\|M^{(1)}}{8} + \frac{M^{(2)}}{24} + o(1)\right) = M^{[2]}h^3, \tag{3}$$

où $M^{[2]} := \frac{\|dX\|M^{(1)}}{8} + \frac{M^{(2)}}{24} + o(1)$.

Récapitulons : nous avons obtenu :

$$z(t+h) = z(t) + hX\left(t + \frac{h}{2}, z(t) + \frac{h}{2}X(t, z(t))\right) + \mathcal{O}(h^3), \quad \text{où } \|\mathcal{O}(h^3)\| \leq M^{[2]}h^3 \quad (4)$$

Nous sommes en mesure d'en déduire l'**algorithme de point milieu**, en utilisant le début de ce développement : en partant d'une valeur y_n, censée être proche de $z(t_n)$, où z est la solution exacte, nous approchons $z(t_{n+1})$ par

$$y_{n+1} := y_n + h_n X\left(t_n + \frac{h_n}{2}, y_n + \frac{h_n}{2}X(t_n, y_n)\right)$$

et donc

$$\Phi(t, x, h) = X\left(t + \frac{h}{2}, x + \frac{h}{2}X(t, x)\right).$$

On peut décomposer cet étape d'algorithme en introduisant les variables auxiliaires $y_{n+\frac{1}{2}} := y_n + \frac{h_n}{2}X(t_n, y_n)$ et la « pente » $p_n := X\left(t_n + \frac{h_n}{2}, y_{n+\frac{1}{2}}\right)$. Cela nous donne

$$\begin{cases} y_{n+\frac{1}{2}} &= y_n + \dfrac{h_n}{2}X(t_n, y_n) \\ p_n &= X\left(t_n + \dfrac{h_n}{2}, y_{n+\frac{1}{2}}\right) \\ y_{n+1} &= y_n + h_n p_n \\ t_{n+1} &= t_n + h_n \end{cases}$$

6.3.3 Erreur de consistance

Il s'agit de la norme de

$$e_n := z(t_n + h_n) - y_n - h_n X\left(t_n + \frac{h_n}{2}, y_n + \frac{h_n}{2}X(t_n, y_n)\right),$$

où z est la solution exacte telle que $z(t_n) = y_n$. En utilisant le développement (2), avec l'estimation du reste (3), on obtient

$$\|e_n\| \leq M^{[2]}h_n^3$$

La méthode du point du milieu est donc d'ordre 2.

6.4 Implémentations en Python 3

Nous considérons un exemple d'équation différentielle pour une fonction à valeur réelle :

$$\frac{dz}{dt}(t) = t - tz(t),$$

ce qui revient à choisir $X(t, x) = t - tx$. Il n'est pas difficile de trouver la solution exacte de cette équation :

$$z(t) = 1 + (z(t_0) - 1)e^{\frac{t_0^2 - t^2}{2}}$$

Nous noterons $X = f$, de sorte que l'équation s'écrit $\frac{dz}{dt} = f(t, z)$.

6.4.1 Implémentation de la méthode d'Euler

La méthode d'Euler avec un pas constant égal à $h > 0$ nous donne l'algorithme

$$\begin{aligned} y_{n+1} &= y_n + hf(t_n, y_n) \\ t_{n+1} &= t_n + h \end{aligned}$$

Une traduction sous forme de script en Python 3 est :

```
def f(t,x)
        return t - t*x
t = eval(input("Instant initial :")
y = eval(input("Valeur initiale :")
L = eval(input("Longueur de l'intervalle de temps :")
N = eval(input("Nombre d'itérations :")
T = [t]
Y = [y]
h = L/N

for i in range(0,N) :
        y = y + h*f(t,y)
        t = t + h
        Y.append(y)
        T.append(t)

# Impression
from matplotlib.pyplot import*
plot(T,Y,'k-')
xlabel('t')
ylabel('y(t)')
show()
```

Lors de l'exécution de ce script, la fonction $f(t, x)$ est définie (def) comme associant aux variables (t, x) la valeur t-t*x, c'est à dire $t - tx$. Puis un échange a lieu entre la machine et l'utilisateur, la machine demandant d'entrer successivement un Instant initial t_0, une Valeur initiale y_0, une Longueur de l'intervalle de temps T et un Nombre d'itérations N. Pour chaque question, il suffit à l'utilisateur d'entrer un nombre, puis d'appuyer sur la touche Retour.

Alors la machine exécute le programme en calculant le pas $h = T/N$, et en créant les listes T et Y. Ces listes sont des suites finies de longueur variable qui sont initialement définies comme étant respectivement (t_0) et (y_0). À chaque l'exécution de la boucle qui suivra, chacune de ces suites s'allonge d'une unité. Ainsi par exemple la liste T prendra successivement les valeurs (t_0), puis (t_0, t_1), puis (t_0, t_1, t_2), etc.

La boucle commence par l'instruction

```
for i in range(0,N) :
```

la variable muette i sert de compteur et parcourt toutes les valeurs entières dans $[0, N[$ (ne pas oublier d'ajouter les deux points à la fin de "range(0,N) :" pour que les instructions qui suivent soient exécutées).

Cette boucle est le cœur du programme :

- L'instruction y = y + h*f(t,y) commande d'attribuer à y une nouvelle valeur et est la traduction de la relation de récurrence $y_{n+1} = y_n + hf(t_n, y_n)$.
- De même l'instruction t = t + h traduit $t_{n+1} = t_n + h$.
- L'instruction Y.append(y) a pour effet de transformer la liste Y en lui ajoutant la dernière valeur calculée de y à la fin de la liste. Autrement dit, elle transforme

$$(y_0, y_1, \cdots, y_n) \quad \text{en} \quad (y_0, y_1, \cdots, y_n, y_{n+1})$$

- L'instruction T.append(t) traduit de même la transformation $(t_0, t_1, \cdots, t_n) \longmapsto (t_0, t_1, \cdots, t_n, t_{n+1})$.

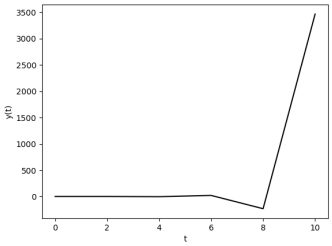
Fin de la boucle.

Les dernières commandes permettent de dessiner le graphe de la solution : from matplotlib.pyplot import* a pour effet d'importer la totalité [2] des commandes contenues dans le module matplotlib.pyplot :

- plot(T,Y,'k-') crée à partir des listes $T = (t_0, t_1, \cdots, t_N)$ et $Y = (y_0, y_1, \cdots, y_N)$ l'ensemble des points de coordonnées (t_i, y_i) et, grâce au tiret - dans 'k-', relie chaque paire de points consécutifs par des segments de droite, de manière à former une courbe continue et affine par morceau.
- La lettre k indique la couleur noire (black en anglais).
- xlabel('t') et ylabel('y(t)') sont des annotations qui apparaîtront sur les axes (respectivement celui des abscisse et celui des ordonnées).

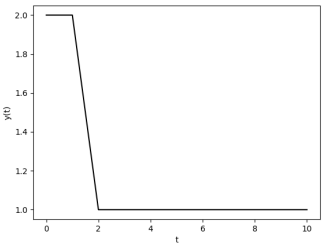
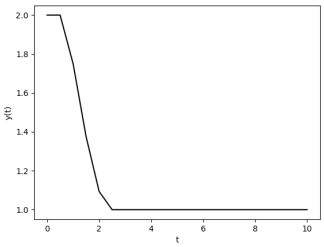
Enfin show() est la petite commande qui manquait pour faire apparaître la belle figure à l'écran.

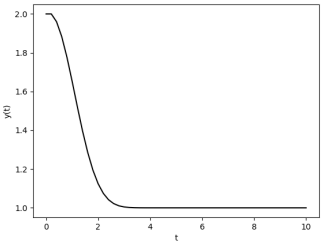
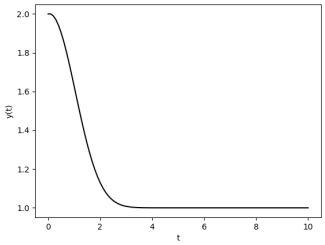
Résultats des simulations — En choisissant les données de Cauchy $(t_0, y_0, L) = (0, 2, 10)$ (c'est à dire en entrant successivement ces trois valeurs), nous obtenons les courbes suivantes en fonction du nombre N d'itérations.

Pour $N = 5$ ($h = 2$) : 

(un résultat **aberrant** car le pas est trop grand, nous reviendrons sur ce problème à propos de la méthode d'Euler *implicite*).

2. totalité à cause de l'étoile * à la fin

$N = 10$ $(h = 1)$:  $N = 20$ $(h = 0,5)$:  .

$N = 50$ $(h = 0,2)$:  $N = 100$ $(h = 0,1)$:  .

Noter que la solution exacte est $z(t) = 1 + e^{-t^2/2}$.

6.4.2 Implémentation de la méthode du point milieu

Avec un pas constant égal à $h > 0$ cette méthode conduit à

$$\begin{cases} y_{n+\frac{1}{2}} &= y_n + \frac{h}{2} f(t_n, y_n) \\ p_n &= f\left(t_n + \frac{h}{2}, y_{n+\frac{1}{2}}\right) \\ y_{n+1} &= h_n + hp_n \\ t_{n+1} &= t_n + h \end{cases}$$

La traduction sous forme de script en Python 3 s'obtient simplement à partir du script précédent en remplaçant la boucle

```
for i in range(0,N) :
    y = y + h*f(t,y)
    t = t + h
    Y.append(y)
    T.append(t)
```

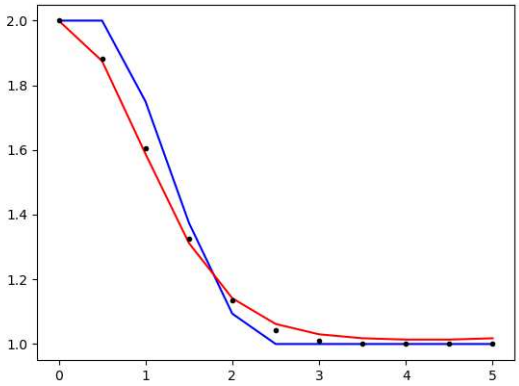
par :

```
for i in range(0,N) :
    mi = y + (h/2)*f(t,y)
    p = f(t+h/2,mi)
    y = y + h*p
    t = t + h
    Y.append(y)
    T.append(t)
```

On a donc introduit dans la boucle les variables supplémentaires mi (pour $y_{n+\frac{1}{2}}$) et p (pour p_n). Le résultat est donc :

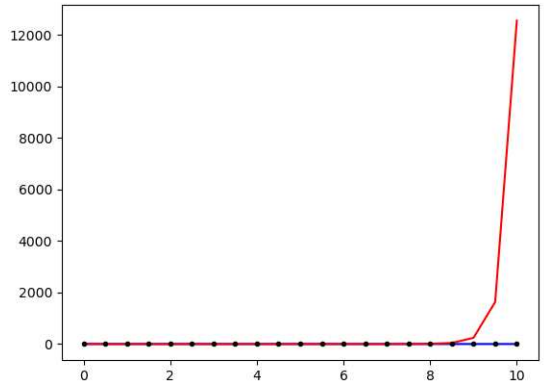
```
def f(t,x)
        return t - t*x
t = eval(input("Instant initial :")
y = eval(input("Valeur initiale :")
L = eval(input("Longueur de l'intervalle de temps :")
N = eval(input("Nombre d'itérations :")
T = [t]
Y = [y]
h = L/N
for i in range(0,N) :
        y = y + h*f(t,y)
        t = t + h
        Y.append(y)
        T.append(t)
# Impression
from matplotlib.pyplot import*
plot(T,Y,'k-')
xlabel('t')
ylabel('y(t)')
show()
```

Toujours pour $f(t,x) = t - tx$, comparons les résultats de la méthode du point milieu avec ceux de l'algorithme d'Euler. En choisissant les données de Cauchy $(t_0, y_0, L) = (0, 2, 5)$ et un nombre $N = 10$ $(h = 1/2)$, nous obtenons :



Dans ce graphique, la solution donnée par l'algorithme d'Euler est la courbe bleue, celle donnée par l'algorithme du point milieu en rouge et les points noirs indiquent les valeurs de la solution exacte $z(t) = 1 + e^{-t^2/2}$. Nous remarquons que la méthode du point milieu donne des valeurs bien plus proches de la solution exacte que celle d'Euler, surtout pour t proche de 0. Mais nous pouvons aussi observer que, pour des valeurs de t proches de 5, la courbe donnée par la méthode d'Euler donne de meilleurs résultats que celle donnée par la méthode du point milieu. Chose étrange, la courbe rouge a tendance à croître au-dessus de 1, alors la solution exacte tend exponentiellement rapidement vers 1 !

Si nous expérimentons les mêmes algorithmes sur un intervalle plus long, c'est à dire avec $(t_0, y_0, L) = (0, 2, 10)$ et $N = 20$ (donc toujours $h = 1/2$), nous obtenons le résultat suivant, cette fois-ci aberrant pour la solution donnée par l'algorithme du point milieu :



L'explication est que la méthode du point milieu conduit dans ce cas à la pente

$$p_n = \left(t_n + \frac{h}{2}\right)\left(1 - \frac{ht_n}{2}\right)(1 - y_n)$$

Dans le produit à droite, le facteur $1 - y_n$ devrait théoriquement tendre vers 0 de façon exponentiellement rapide, mais en réalité, cette convergence n'est pas exponentielle dans les calculs numériques. En revanche le facteur de ce terme est un polynôme de degré 2 en t_n, qui change de signe pour $t_n = \frac{2}{h}$.

6.5 Méthode d'Euler implicite

Nous avons vu à propos de la méthode d'Euler que celle-ci donne des résultats aberrants si le pas h est trop grand (voir la simulation précédente pour $N = 5$ et $h = 2$). La raison est que cette méthode est alors instable. Ce phénomène se comprend facilement sur l'exemple de l'équation différentielle $\frac{dz}{dt} = az$, où $a \in \mathbb{R}$ est une constante, dont la solution est $z(t) = z(t_0)e^{a(t-t_0)}$. L'algorithme d'Euler avec un pas constant h s'écrit pour cette équation

$$\begin{cases} y_{n+1} &= y_n + hay_n = (1 + ah)y_n \\ t_{n+1} &= t_n + h \end{cases}$$

On voit alors que, si $a < 0$ et si $|ah| > 1$, $(1+ah) < 0$ et donc y_{n+1} a un signe opposé à y_n, ce qui ne correspond pas du tout au comportement de la solution que l'on veut approcher. Ce type de pathologie risque de se reproduire également par exemple pour un champ de vecteur X défini sur un ouvert de $\mathbb{R} \times \mathbb{R}$ tel que $\frac{\partial X}{\partial x}(t, x) < 0$ et $|h\frac{\partial X}{\partial x}(t, x)| > 1$.

Une façon d'éviter ce problème consiste à remplacer l'algorithme d'Euler par sa version *implicite*. Pour simplifier l'exposé nous ne considérerons dans la suite que des équations différentielles pour des fonctions à valeur réelle. Pour un champ de vecteur quelconque X,

cet algorithme consiste à calculer les suites $(y_n)_{0\leq n\leq N}$ et $(t_n)_{0\leq n\leq N}$ telles que

$$\left\{\begin{array}{rcl} y_{n+1} &=& y_n + h_n X(t_{n+1}, y_{n+1}) \\ t_{n+1} &=& t_n + h_n \end{array}\right.$$

On voit alors que y_{n+1} n'est plus donné par une formule mais comme étant la solution de l'équation

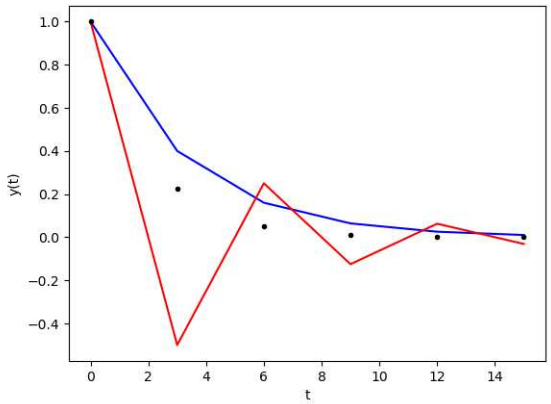
$$G(x) = 0 \quad \text{où } G(x) := x - y_n - h_n X(t_{n+1}, x) \tag{5}$$

et où x est l'inconnue.

Dans le cas simple de l'équation $\frac{dz}{dt} = az$, $X(t,x) = ax$ et, avec un pas constant, cet algorithme s'écrit

$$\left\{\begin{array}{rcl} y_{n+1} &=& y_n + hay_{n+1} \\ t_{n+1} &=& t_n + h \end{array}\right. \iff \left\{\begin{array}{rcl} y_{n+1} &=& \frac{1}{1-ah} y_n \\ t_{n+1} &=& t_n + h \end{array}\right.$$

On voit alors que, si $a < 0$, il n'y a aucune difficulté, puisque $1/(1-ah)$ est alors toujours dans $]0,1[$, même pour un pas de temps grossier. Cette méthode est donc préférable dans ce cas. Nous pouvons comparer ces deux méthodes pour l'équation $\frac{dz}{dt} = -\frac{1}{2}z$, avec les données de Cauchy $(t_0, y_0, L) = (0, 1, 15)$ et $N = 5$ (donc $h = 3$, pas grossier) :



la courbe rouge est donnée par l'algorithme d'Euler, la courbe bleue par l'algorithme d'Euler implicite et les points noirs indiquent des valeurs prises par la solution exacte.

En revanche il faut faire attention que, si $a > 0$, c'est la situation inverse qui se produit : la méthode d'Euler ne conduit pas à des incohérences, même si le pas de temps est grossier, mais la méthode d'Euler implicite peut alors conduire à des instabilités catastrophiques car alors $1 - ah$ peut être négatif ou proche de 0.

De plus l'algorithme d'Euler implicite nécessite également de résoudre l'équation (5) $G(x) = 0$ à chaque étape, parfois de façon approchée. Une méthode assez générale pour cela est la méthode de Newton : l'idée est d'approcher G par la fonction affine F définie par

$$F(x) = G(y_n) + G'(y_n)(x - y_n).$$

Cette fonction affine approche G à l'ordre 1 en y_n. Il est alors facile de résoudre l'équation $F(x) = 0$, dont la solution est

$$x = y_n - \frac{G(y_n)}{G'(y_n)} = y_n + \frac{h_n X(t_{n+1}, y_n)}{1 - h_n \frac{\partial X}{\partial x}(t_{n+1}, y_n)}$$

Si G' ne varie pas trop, cette solution est alors une approximation relativement bonne de la solution de (5). Cela conduit à remplacer l'algorithme d'Euler par la version approchée

$$\left\{ \begin{array}{rcl} y_{n+1} & = & y_n + \dfrac{h_n X(t_{n+1}, y_n)}{1 - h_n \frac{\partial X}{\partial x}(t_{n+1}, y_n)} \\ t_{n+1} & = & t_n + h_n \end{array} \right.$$