Cours 3 - Intégration et équations différentielles

Alain Le Gall - alain.legall@ece.fr

2022-2023

Table des matières

Chapitre 1

Calcul d'intégrales

1.1 Présentation

Il existe de nombreuses méthode pour calculer la valeur numérique d'une intégrale définie, de la forme classique :

$$I = \int_a^b f(x)dx$$

Toutes les méthodes que l'on envisage peuvent s'écrire sous la forme :

$$I = \int_a^b f(x)dx = \sum_{i=1}^{n} \omega_i f(x_i) + E$$

avec E : erreur de troncation, les x_i sont les abscisses (pivots ou nœuds) et les ω_i sont des poids. On peut ainsi construire une méthode des rectangles, des trapèzes, des paraboles (Simpson) etc...Toutes ces méthodes ont une fonction E différente. On peut aussi généraliser cette technique et obtenir alors une meilleure précision, la méthode de Romberg.

1.2 La méthode de Romberg

La méthode de Romberg permet d'évaluer la valeur d'une intégrale et ce, avec la précision voulue. On calcule cette intégrale à l'aide d'un schéma récursif, en créant un tableau de Romberg.

1.2.1 Résultats mathématiques utiles

L'intégrale à calculer est mise sous la forme :

$$I_n = \int_{x_0}^{x_0+nh=x_n} f(x)dx = T_k(h) + o(h^{2k+2})$$

avec :

$$T_0(h) = h(\frac{f_0}{2} + f_1 + ... + f_{n-1} + \frac{f_n}{2})$$

$$f_k = f(x_0 + kh)$$

$$h = \frac{x_n - x_0}{n}$$

On montre alors que :

$$T_k(h) = \frac{4^k T_{k-1}(\frac{h}{2}) - T_{k-1}(h)}{4^k - 1}$$

ou encore :

$$T_k(h) = T_{k-1}(\frac{h}{2}) + \frac{T_{k-1}(\frac{h}{2}) - T_{k-1}(h)}{4^k - 1}$$

D'après la formule de I_n, on voit bien que l'erreur commise sur la valeur exacte de l'intégrale est en $o(h^{2k+2})$. Il suffit donc de calculer $T_k(h)$ avec une valeur de k suffisante pour obtenir la précision voulue.

1.2.2 Analyse et algorithme

On calcule, grâce à la relation de récurrence sur les $T_k(h)$, la valeur de T_0, T_1, ..., T_k jusqu'à ce que $|T_k(h) - T_{k-1}(h)| < \epsilon$ où ϵ est la précision voulue sur l'intégrale.

On note :

$$T_k^m = T_k(\frac{h}{2^m})$$

Ainsi, on aura la récurrence : (1)

$$T_k^m = T_{k-1}^{m+1} + \frac{T_{k-1}^{m+1} - T_{k-1}^m}{4^k - 1}$$

Pour évaluer T_m^0, il faut donc avoir initialement calculé $(T_0^0, ..., T_0^m)$ avec :

$$T_0^0 = \frac{h}{2}(f(a) + f(b))$$

car $h = b - a$, $x_0 = a$ et $x_n = b$.

On peut montrer rapidement que l'on a : (2)

$$T_0^m = \frac{T_0^{m-1}}{2} + \frac{h}{2^m} \sum_{i=1}^{2^{m-1}} f(a + (2i-1)\frac{h}{2^m})$$

L'algorithme se déroule ainsi : (étape 1) on calcule d'abord T_0^m par la formule (2) puis son calcule (étape 2) T_1^{m-1}, T_2^{m-2},...,T_m^0 par la relation (1) de récurrence. Enfin, on compare la valeur de T_m^0 avec T_{m-1}^0 précédemment sauvée afin de voir si il y a convergence du processus.

Pour le programme, inutile donc de créer un tableau à deux dimensions pour sauver tous les T_k^m calculés. Un seul tableau T suffit pour sauver à la fois les éléments de l'étape 1 et ceux de l'étape 2. On a alors l'algorithme suivant :

Algorithm 1: Algorithme de Romberg

Data: a, b, ϵ (précision), n_{max}
$h = b - a$
$T_1 = \frac{h}{2}(f(a) + f(b))$
$k = 0$
$k = 0$
while $|T(1) - T_d| > \epsilon$ **do**
  $k = k + 1$
  calcul de T_0^m :
  $if = 2^{k-1}$
  $s = 0$
  for *i=1,...,if* **do**
    $s = s + f(a + (i - 0,5) * \frac{h}{if})$
  end
  $T(k+1) = \frac{T(k)}{2} + \frac{h}{2if} * s$
  $T_d = T(1)$ (Td=T_{m-1}^0 qu'on sauve)
  calcul de T_m^0 :
  $q = 1$
  for *i=k,...,1 par -1* **do**
    $q = 4 * q$
    $T(i) = T(i+1) + \frac{T(i+1) - T(i)}{q-1}$
  end
end
Result: $T(1)$: approximation de l'intégrale cherchée

On peut aussi introduire la variable $rh = \frac{h}{2^{k-1}} = \frac{h}{if}$, ce qui simplifie l'algorithme.

1.2.3 Code Arduino DUE

Voir démonstration.

1.2.4 Exemples d'application

On prend par exemple $\epsilon = 10^{-13}$ à chaque fois.

1. $$\int_0^1 \frac{dx}{1+x} = ln(2)$$
2. $$\int_0^1 exp(-x)dx = 1 - e^{-1}$$
3. $$\int_0^{\frac{1}{2}} \frac{6dx}{\sqrt{1-x^2}} = \pi$$

1.2.5 Conclusion

On a bien vu que l'algorithme de Romberg est très performant et permet en théorie de calculer une intégrale avec la précision arbitraire voulue (dans les limites de la préci-

sion des microprocesseurs bien entendu). Cette méthode est de plus très rapide et peu consommatrice de mémoire : pour $\epsilon = 10^{-13}$ on a un tableau T de dimension 10 environ).

1.2.6 Applications

On verra en TD et TP des applications multiples (Formule de la période d'un pendule, approximation de Borda etc..).

Chapitre 2

Résolution d'équations différentielles

2.1 Présentation

Un très grand nombre de questions scientifiques ou techniques admettent des modèles mathématiques qui impliquent des équations différentielles. Songez à la mécanique ou à la cinétique des équations chimiques, ou l'évolution des tensions et intensités dans les circuits électroniques. La plupart de ces équations n'ont pas de solution analytique, il faut les résoudre numériquement !

2.2 Fil conducteur : l'exemple du pendule simple

Le pendule simple sans frottements est régi par l'équation différentielle :

$$y'' + k^2 sin(y) = 0$$

Si on introduit la variable

$$z = y'$$

, il vient :

$$z' = -k^2 sin(y)$$

donc on peut en général ramener un problème du second ordre à un système d'équations différentielles du premier ordre. On n'étudiera donc que les équations différentielles du premier ordre.

2.3 Méthode d'Euler

2.3.1 Présentation de la méthode d'Euler

Cette méthode d'Euler n'est jamais employée en pratique ! Mais c'est un excellent exemple introductif, généralisable. La méthode proposée par Euler pour résoudre une équation du type :

$$y' = f(x, y(x))$$

$$y(a) = A$$

$$x \geq a$$

s'écrit :

$$y_{n+1} = y_n + hf(x_n, y_n) = y_n + hf_n$$

Pour cela, on a fait l'approximation de la dérivée :

$$y'(x_n) \simeq \frac{y_{n+1} - y_n}{h}$$

2.3.2 Caractéristiques

Pour calculer y_{n+1}, nous avons utilisé la valeur de y_n et pas celles de y_{n-1}, y_{n-2}, etc... Il s'agit donc d'une méthode à un pas.Pour passer de y_n à y_{n+1}, nous ne calculons f qu'une seule fois (les méthodes de Runge-Kutta que l'on verra ensuite utilisent plusieurs valeurs de f pour améliorer la précision sur y'). Enfin le calcul de y_{n+1} s'accompagne d'une erreur de troncation proportionnelle à h^2 et c'est pourquoi ce schéma est dit d'ordre un.

2.3.3 Programmation de l'algorithme d'Euler

Voici un algorithme de la méthode d'Euler :

Algorithm 2: méthode d'Euler

Data: h, x_0, y_0, x_{max}
Initialiser x à x_0 et y à y_0
while $x < x_{max}$ **do**
  $f = f(x, y)$
  $y = y + hf$
  $x = x + h$
  afficher le point solution (x, y)
end

2.3.4 Exemple à l'aide d'un tableur

Résolvons l'équation différentielle :

$$y' + y = exp(-x)$$

c'est-à-dire :

$$y' = -y + exp(-x) = f(x, y)$$

On utilise un tableur et on compare pour différentes valeurs de h la solution obtenue, sachant que la solution exacte est ici : $y = xexp(-x)$

2.3.5 Généralisation à un système différentiel

Cette généralisation est immédiate ; la fonction inconnue, sa valeur initiale et le second membre sont remplacés maintenant par des vecteurs :

Algorithm 3: méthode d'Euler pour un système différentiel

Data: h, x_0, $\mathbf{y_0}$, x_{max}
Initialiser x à x_0 et $\mathbf{y}$ à $\mathbf{y_0}$
while $x < x_{max}$ **do**
  $\mathbf{f} = \mathbf{f}(\mathbf{x}, \mathbf{y})$
  $\mathbf{y} = \mathbf{y} + h\mathbf{f}$
  $x = x + h$
  afficher le point solution $(x, \mathbf{y})$
end

2.3.6 Exemple du pendule simple

Prenons l'équation du pendule simple (avec $k = 1$) :

$$y'' + sin(y) = 0$$

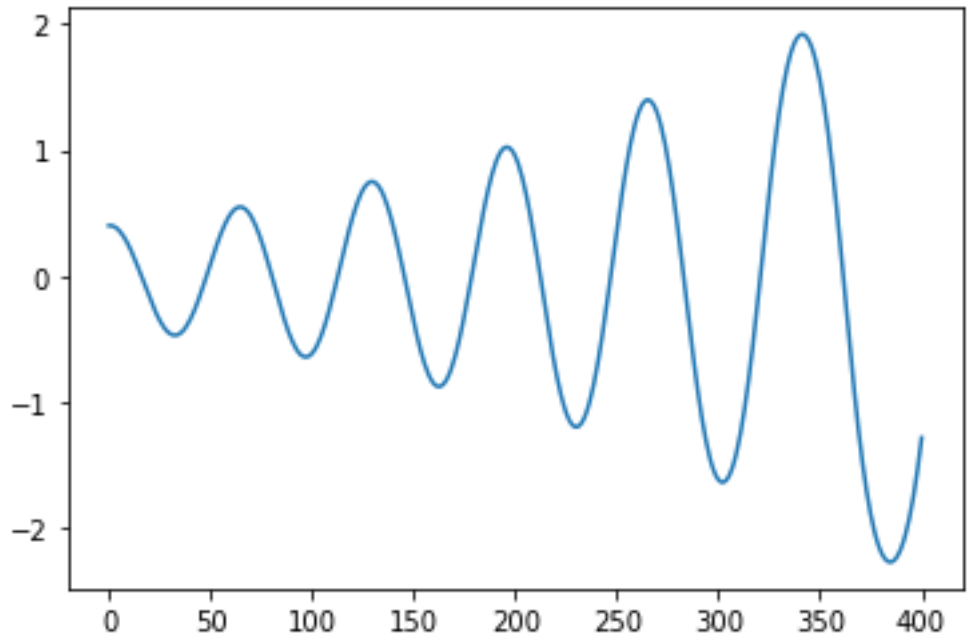
On pose :

$$y' = z$$
$$z' = -sin(y)$$

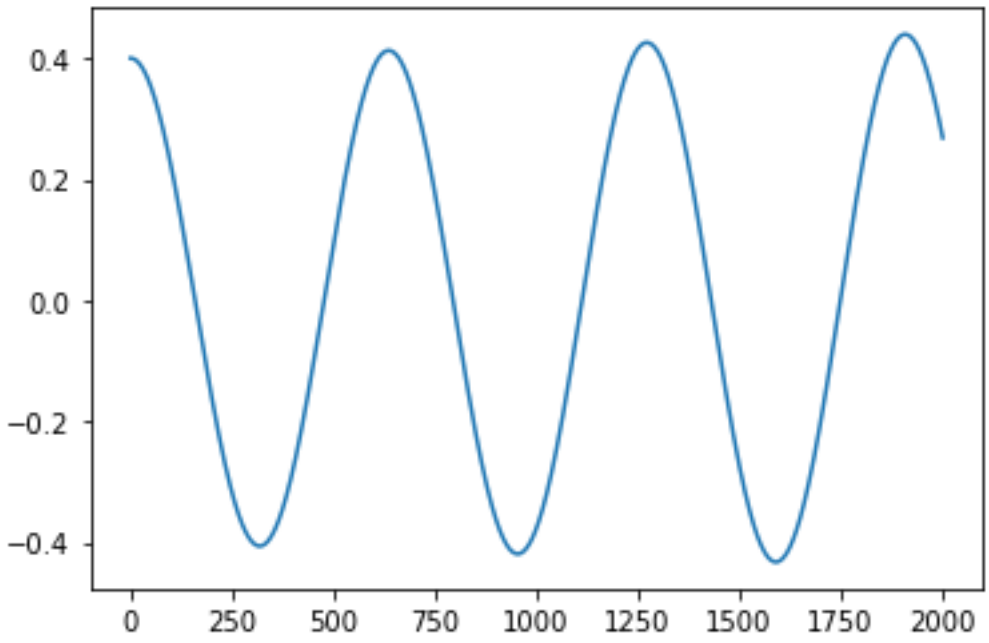
Voir le code python correspondant !

```python
import numpy as np
import matplotlib.pyplot as plt
import math

def euler_y(y,z):
    return z

def euler_z(y,z):
    return -math.sin(y)

def euler_resolution(h,y,z):
    return y+h*euler_y(y,z),z+h*euler_z(y,z)


h=0.1
y0=float(input('angle initial : '))
z0=float(input('vitesse initiale : '))

nb_points=400

t=np.linspace(0,nb_points,nb_points)
y=np.linspace(0,0,nb_points)
z=np.linspace(0,0,nb_points)

y[0]=y0
z[0]=z0


for i in range(1,nb_points):
    y[i],z[i]=euler_resolution(h,y[i-1],z[i-1])

plt.plot(t, y)
plt.show()
```

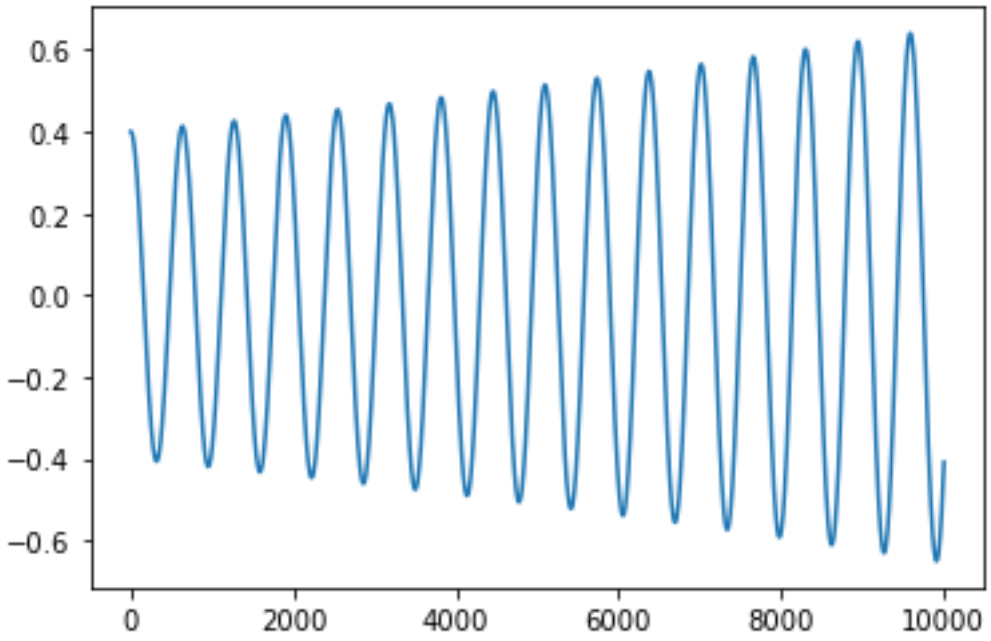
Pour $h = 0,1$, $y_0 = 0,4$ (angle initial) et $z_0 = 0$ (vitesse initiale), on obtient :



Pour $h = 0,01$, $y_0 = 0,4$ (angle initial) et $z_0 = 0$ (vitesse initiale), on obtient : c'est mieux !



mais si pour le même $h = 0,01$, on augmente le nombre de points calculés, il vient encore une divergence ! :



Comme on peut le constater, la qualité de la méthode laisse à désirer : l'amplitude augmente nettement avec le temps !

2.3.7 Conclusion

La méthode d'Euler peut donner des résultats corrects à condition de prendre un pas h très petit, et à condition que l'intervalle d'étude soit peu large ; car il apparait toujours sinon une grande divergence ! En pratique, on n'emploie jamais cette méthode, on lui préfère la méthode de Runge-Kutta que l'on va étudier ci-après.

2.4 Méthodes de Runge-Kutta

2.4.1 Présentation des méthodes de Runge-Kutta

Les méthodes que nous allons présenter ici sont à considérer comme des généralisations de la méthode d'Euler vue précédemment, où l'on va calculer f plusieurs fois par pas, pour réduire l'erreur de troncation. Ce sont aussi probablement les méthodes les plus employées dans la pratique. Nous exposerons la théorie pour une méthode d'ordre 2, bien que les applications fassent appel généralement à des méthodes d'ordre 4 ou plus.

2.4.2 Méthodes d'ordre 2

On cherche toujours la solution au problème présenté précédemment, en supposant connue la solution numérique approchée au point $x_n = x_0 + nh$. Pour cela, nous allons construire une "fonction incrément" $\Phi(x_n, y_n, h)$ telle que :

$$y_{n+1} - y_n = h\Phi(x_n, y_n, h)$$

Cette fonction dépend aussi du second membre f.

Soit d'autre part la solution exacte $e(t)$ du problème différentiel étudié :

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

$$e(x) = y$$

En termes imagés, $e(t)$ est la solution de l'équation différentielle "qui passe par le point de coordonnées x et y" ; x et y sont ici considérés comme arbitraires mais constants. L'incrément exact est défini par :

$$e(x+h) - e(x) = h\Delta(x, y, h)$$

Dans le cas du schéma d'Euler, on avait $\Phi = f$. Nous allons chercher, pour obtenir une méthode d'ordre 2, une expression de Φ qui coïncide avec Δ jusqu'aux termes en h compris (un schéma est dit d'ordre p si $\Delta - \Phi$ est $O(h^p)$). Nous supposons, comme l'ont fait Runge et Kutta il y a plus d'un siècle, que Φ est de la forme :

$$\Phi(x, y, h) = b_1 f(x, y) + b_2 f(x + p_1 h, y + p_2 h f(x, y))$$

où les quantités b_1, b_2,p_1 et p_2 sont des constantes à déterminer. Il nous suffit d'imposer que le développement de Taylor de Φ au premier ordre en h coïncide avec celui de Δ au même ordre ! Ils s'écrivent :

$$\Delta = f(x, y) + \frac{1}{2}h(f_x + f f_y) + o(h^2)$$

$$\Phi = (b_1 + b_2)f + h b_2 p_1 f_x + h b_2 p_2 f f_y + o(h^2)$$

D'où les trois conditions à remplir pour que $\Delta = \Phi$:

$$b_1 + b_2 = 1$$

$$b_2 p_1 = \frac{1}{2}$$

$$b_2 p_2 = \frac{1}{2}$$

On dispose de quatre paramètres inconnus, et il est d'usage de conserver un paramètre libre et de déterminer les trois autres à l'aide des relations précédentes, la dernière quantité sera ajustée après pour obtenir une propriété désirable de l'algorithme (bonne stabilité, erreur d'arrondi faible..). Nous décidons de conserver b_2, renommé β pour la circonstance. Le système précédent devient :

$$b_1 = 1 - \beta$$

$$b_2 = \beta$$

$$p_1 = p_2 = \frac{1}{2\beta}$$

Nous obtenons ainsi une méthode de Runge-Kutta d'ordre 2 caractérisée par la fonction incrément :

$$\Phi(x, y, h) = (1 - \beta)f(x, y) + \beta f(x + \frac{h}{2\beta}, y + \frac{h}{2\beta}f(x, y))$$

Parmis toutes les valeurs possibles de β, deux cas sont restés dans l'histoire :

$$\beta = \frac{1}{2}$$

$$y_{n+1} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_n + h, y_n + h f(x_n, y_n)))$$

et aussi le choix :

$$\beta = 1$$

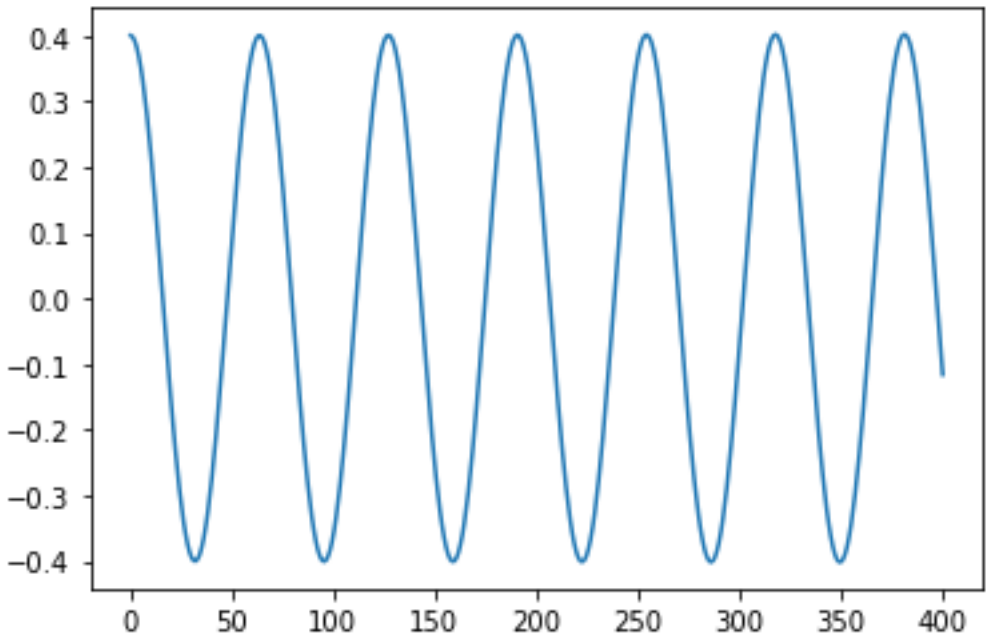
$$y_{n+1} = y_n + h f(x_n + \frac{h}{2}, y_n + \frac{h}{2}f(x_n, y_n))$$

2.4.3 Exemple du pendule simple

Il suffit de reprendre le programme précédent et de modifier la fonction euler par une nouvelle définition rk2 :

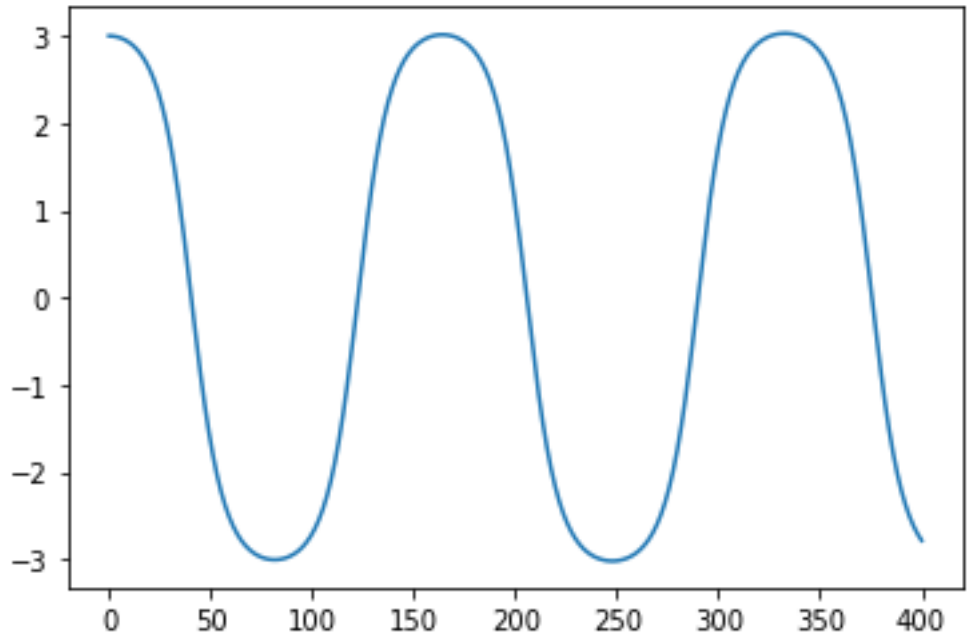
```
import numpy as np
import matplotlib.pyplot as plt
import math

def fx_y(y,z):
    return z

def fx_z(y,z):
    return -math.sin(y)

def rk2(h,y,z):
    return y+0.5*h*(fx_y(y,z)+fx_y(y+h*fx_y(y,z),z+h*fx_z(y,z))),z+0.5*h*(fx_z(y,z)+fx_z(y+h*fx_y(y,z),z+h*fx_z(y,z)))


h=0.1
y0=float(input('angle initial : '))
z0=float(input('vitesse initiale : '))

nb_points=400

t=np.linspace(0,nb_points,nb_points)
y=np.linspace(0,0,nb_points)
z=np.linspace(0,0,nb_points)

y[0]=y0
z[0]=z0


for i in range(1,nb_points):
    y[i],z[i]=rk2(h,y[i-1],z[i-1])

plt.plot(t, y)
plt.show()
```

Pour $h = 0,1$, $y_0 = 0,4$ (angle initial) et $z_0 = 0$ (vitesse initiale), on obtient :

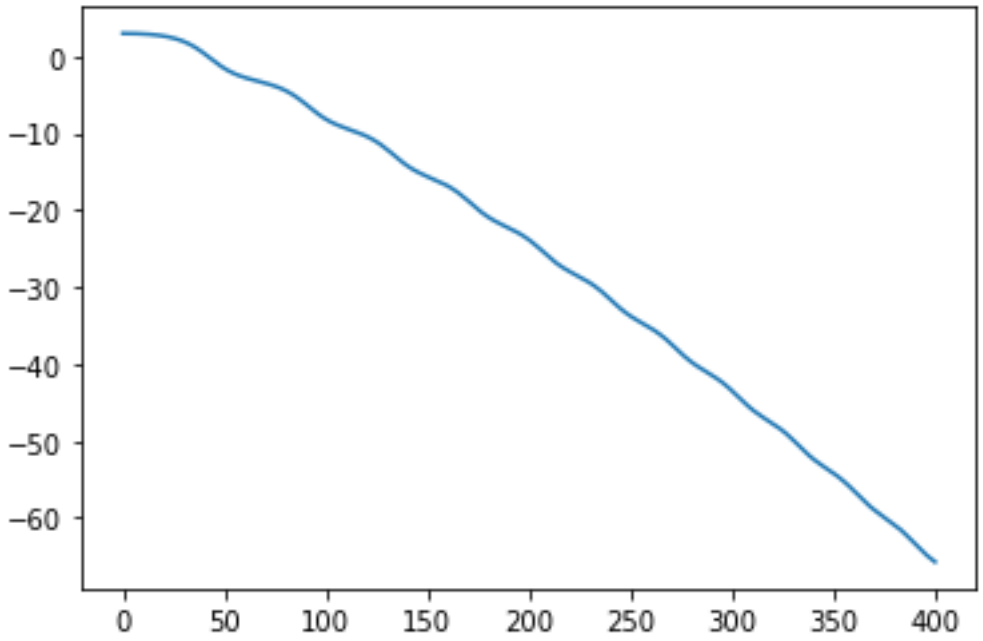


donc aucune divergence maintenant comparée aux mêmes conditions vues avec Euler.

Maintenant si on exagère la non linéarité en prenant un angle initial grand ($y_0 = \pi$), on obtient :



Le phénomène non linéaire n'est pas divergent mais bien stable et on observe bien ce phénomène périodique et non isochrone. Avec la méthode d'Euler dans les mêmes conditions, on aurait observé une solution incohérente :



2.4.4 Méthode d'ordre 4

On fournit directement le résultat ici :

$$k_1 = f(x_n, y_n)$$

$$k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$$

$$k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$$
$$k_4 = f(x_n + h, y_n + hk_3)$$
$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$x_{n+1} = x_n + h$$

2.4.5 Avantages et inconvénients des méthodes de Runge-Kutta

La méthode RK4 se distingue par sa simplicité de mise en œuvre ; nous la recommandons d'ailleurs pour tous les problèmes simples. Elle ne requiert que 4 évaluations de f par pas.

L'algorithme de RK souffre d'inconvénients. Il y a notamment des problèmes de stabilité dès que le second membre est grand ou rapidement variable. Il existe heureusement des algorithmes plus raffinés. Le prix à payer pour des améliorations est une structure plus compliquée de l'algorithme.