Méthodes numériques

Méthodes d'Euler de de Runge-Kutta d'ordre 4 pour des équations du premier ordre ou du deuxième

Table des matières

1 Introduction **2**

2 Equation du premier ordre : étude de la vitesse lors d'une chute avec frottements quadratiques **2**

2.1 Méthode l'Euler explicite . . . 2
2.1.1 Schéma Global . . . 2
2.1.2 Mise en oeuvre sur l'exemple . . . 3
2.1.2.1 Calcul des premiers termes à la main . . . 4
2.1.2.2 Utilisation d'un tableur . . . 4
2.1.2.3 Utilisation de Python . . . 6
2.2 Méthode de Runge-Kutta d'ordre 4 . . . 6
2.2.1 Schéma global . . . 6
2.2.2 Mise en oeuvre sur l'exemple . . . 8
2.2.2.1 Calcul des premiers termes à la main . . . 8
2.2.2.2 Utilisation d'un tableur . . . 9
2.2.2.3 Utilisation de Python . . . 9

3 Equation du deuxième ordre : pendule simple **10**

3.1 Méthode d'Euler explicite . . . 11
3.1.1 Calculs manuels . . . 11
3.1.2 Utilisation d'un tableur . . . 11
3.2 Méthode d'Euler implicite . . . 12
3.2.1 Principe . . . 12
3.2.2 Calculs manuels . . . 12
3.2.3 Utilisation d'un tableur . . . 12
3.2.4 Utilisation de python . . . 13
3.3 Méthode de Runge-Kutta d'ordre 4 . . . 14
3.3.1 Calculs manuels . . . 15
3.3.2 Utilisation du tableur . . . 15
3.3.3 Utilisation de Python . . . 16
3.3.3.1 Animation . . . 17

4 Conclusion **20**

1 Introduction

En physique, nous recherchons souvent l'évolution temporelle d'une grandeur caractéristique du système étudié. L'étude théorique nous amène à une équation différentielle souvent non linéaire lorsque le système n'a pas été trop modélisé. La résolution exacte de ce type d'équation est rare, on utilise alors une méthode numérique pour approcher la solution graphiquement.
L'idée de ce document est de montrer l'utilisation quelques méthodes numériques classiques. Pour cela et afin d'éviter trop de formalisme mathématique, on se basera sur deux exemples classiques de la physique donnant :

— pour l'un une équation différentielle du premier ordre, la chute avec frottements quadratiques ;

— pour l'autre une équation différentielle du deuxième ordre, le pendule simple.

Après la mise en équation, nous proposerons de mettre en oeuvre la méthode numérique à l'aide d'un tableur ainsi qu'à l'aide d'un script Python.

2 Equation du premier ordre : étude de la vitesse lors d'une chute avec frottements quadratiques

L'application du principe fondamental de la dynamique conduit à l'équation différentielle suivante :

$$\frac{\mathrm{d}v}{\mathrm{d}t} = A\,v^2 + B \tag{1}$$

Le sauteur possède une vitesse initiale nulle, $A = -2.03 \times 10^{-3}\,\mathrm{SI}$ et $B = 9{,}81\,\mathrm{SI}$.

2.1 Méthode l'Euler explicite

2.1.1 Schéma Global

Cette méthode est généralement la première méthode numérique enseignée malgré des résultats peu fiables. Son schéma et sa compréhension sont simples.
Pour la mettre en oeuvre, nous avons besoin des informations suivantes :

$$\begin{cases} \text{Condition initiale : } y(t=0) = y_0 \\ \text{Equation différentielle : } \dfrac{\mathrm{d}y}{\mathrm{d}t} = f(t,y) \end{cases} \tag{2}$$

Après avoir choisi un **pas** $\mathrm{d}t$ **de calcul**, on établit le schéma suivant :

$$\begin{cases} y(t=0) = y_0 \\ y_1 = y_0 + f(t_0,y_0) \times \mathrm{d}t \\ y_2 = y_1 + f(t_1,y_1) \times \mathrm{d}t \\ \cdots \\ y_{n+1} = y_n + f(t_n,y_n) \times \mathrm{d}t \end{cases} \tag{3}$$

Cela signifie que l'on évalue la valeur de la fonction y à l'instant $n+1$ en calculant la pente de celle-ci $\left(\dfrac{\mathrm{d}y}{\mathrm{d}t}\right)$ à l'instant n :

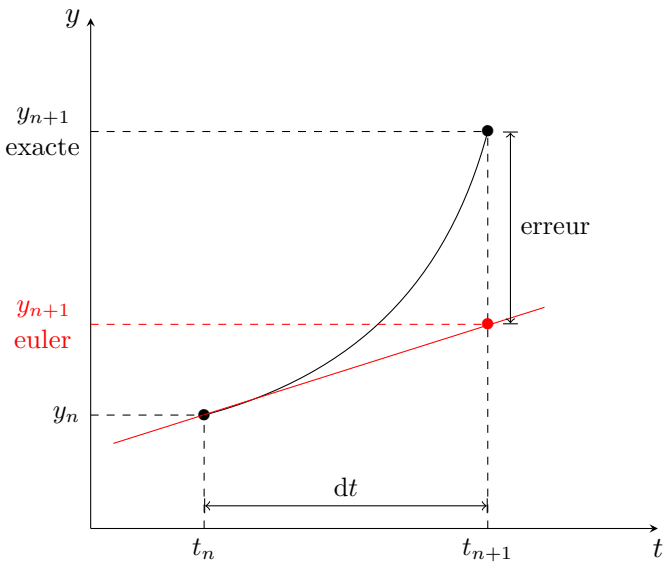

FIGURE 1 – Méthode d'Euler explicite avec un pas égal à dt et erreur

Les méthodes numériques telle que la méthode d'Euler sont très sensibles au **choix du pas de calcul** : celui-ci doit être pris suffisamment petit pour que les résultats soit cohérents.

On peut également noter que l'erreur diminue avec le pas de calcul. Pour la méthode d'Euler explicite, **un pas divisé par deux divise l'erreur par deux** (schéma ci-dessous).

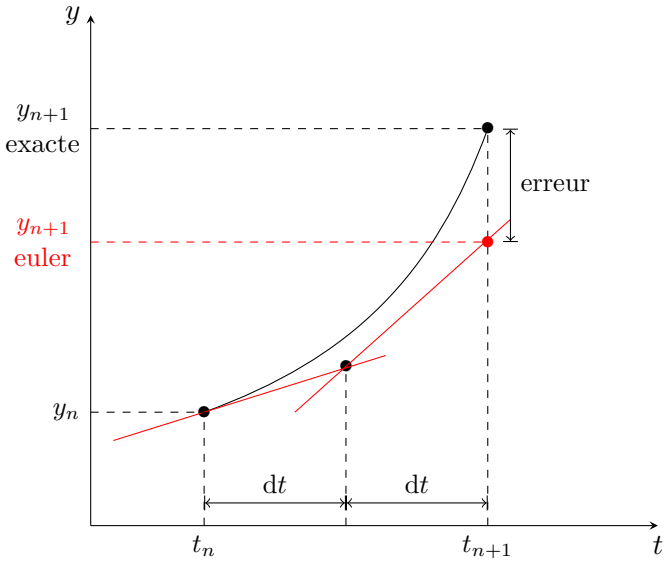

FIGURE 2 – Méthode d'Euler explicite avec un pas égal à d$t/2$ et erreur

Attention !
Ne pas croire que diminuer le pas de calcul améliore systématiquement les résultats de la modélisation. En effet, les erreurs (appelées erreurs d'arrondi et de troncature, voir dans les référence section 4) s'ajoutent avec le principe d'itération (et plus le pas diminue plus il y a d'étapes de calculs). Il faut donc trouver un compromis.

2.1.2 Mise en oeuvre sur l'exemple

On choisira un pas de calcul de 1 seconde, un temps total de chute de 30 secondes.

2.1.2.1 Calcul des premiers termes à la main

$$v_0 = 0 \tag{4}$$

$$v_1 = v_0 + \left(\frac{\mathrm{d}v}{\mathrm{d}t}\right)_{t=0} \times \mathrm{d}t = v_0 + (A \times v_0^2 + B) \times \mathrm{d}t = 9{,}81 \tag{5}$$

$$v_2 = v_1 + (A \times v_0^2 + B) \times \mathrm{d}t = 19{,}42 \tag{6}$$

...

2.1.2.2 Utilisation d'un tableur

Pour se faire, on créé une **colonne de temps** : le temps initial étant nul, à chaque ligne suivante on ajoute le pas choisi jusqu'au temps final voulu.

Dans la **deuxième colonne, on calcule les valeurs de vitesse**. On renseigne tout d'abord la condition initiale dans la première ligne, puis dans la deuxième ligne la formule de calcul issue de la méthode d'Euler que l'on recopiera dans les lignes suivantes jusqu'à la ligne correspondant au temps final.

Voici les premières cellules du tableau :

	A	B
1	t	v
2	0	0
3	$= \mathrm{A2} + 1$	$= \mathrm{B2} + (-0{,}00203 \times \mathrm{B2}^2 + 9{,}81) \times \$\mathrm{A}\$2$
4	$= \mathrm{A3} + 1$	$= \mathrm{B3} + (-0{,}00203 \times \mathrm{B3}^2 + 9{,}81) \times \$\mathrm{A}\$2$

TABLE 1 – Premières cellules de la feuille de calculs "Méthode d'Euler explicite et chute avec frottements quadratiques"

Remarque

Les signes $ permettent de bloquer le nom de la référence à la cellule A2 (là où est indiqué le pas de calcul) lors de la recopie de la formule.

Résultats et comparaison avec la solution exacte

Les résultats graphiques de la méthode sont représentés sur la figure 3.

Nous avons utilisé un pas de 1 seconde, 0,5 seconde ou bien 5 secondes.
Notons qu'il n'y a pas de différence notoire entre les deux premières courbes. Par contre, le pas de 5 secondes donnent un résultat incohérent.

Enfin, on peut comparer les résultats donnés par cette méthode d'Euler par rapport à la solution exacte pour se rendre compte que sur cet exemple, la méthode numérique donne des résultats très acceptables (voir figure 4).

Télécharger le fichier de calcul

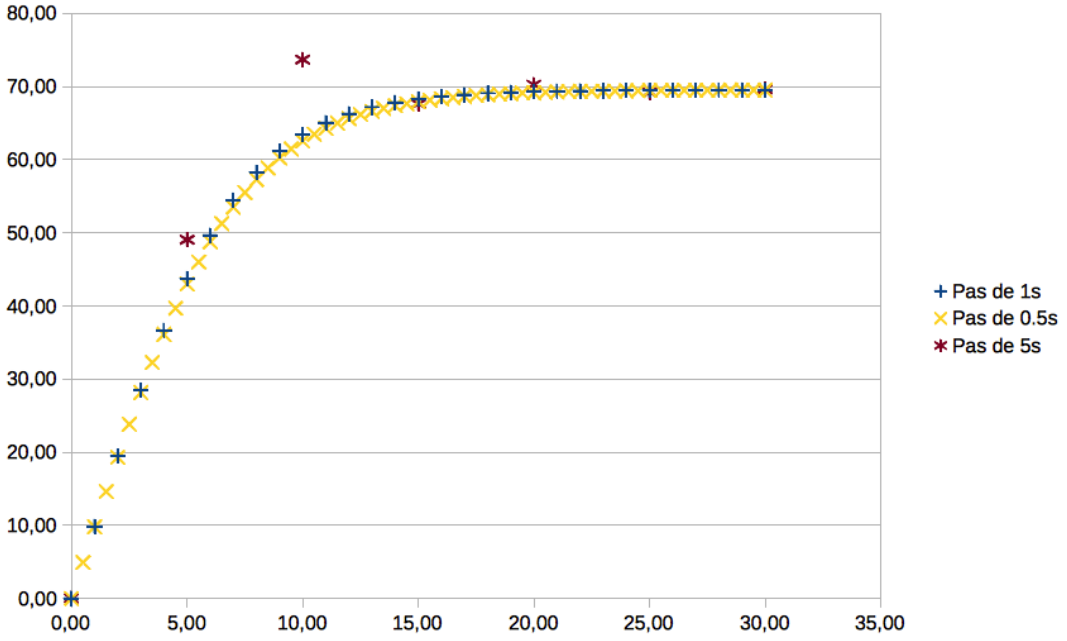

FIGURE 3 – Méthode d'Euler explicite appliquée à la chute avec frottements quadratiques

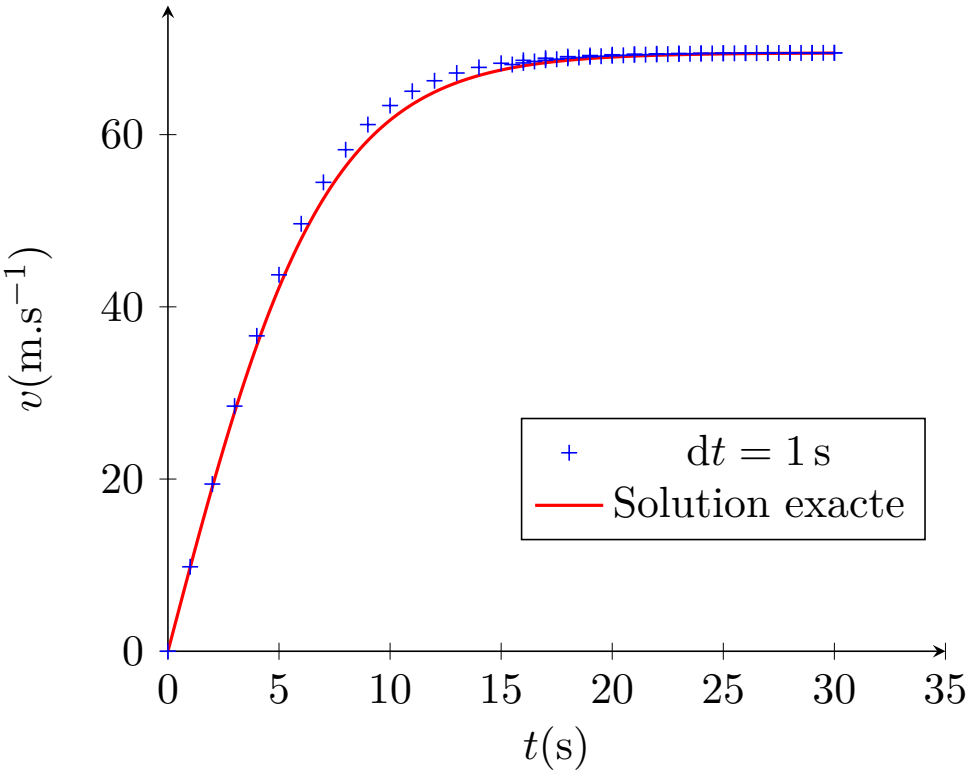

FIGURE 4 – Comparaison entre la méthode d'Euler explicite et la solution exacte

2.1.2.3 Utilisation de Python

```
# -*- coding : utf8 -*- #le fichier est encode en utf8
#Importations de librairies utiles (calculs, fonctions, graphiques) :
import numpy, math, matplotlib
from pylab import * #Utilisation des librairies precedentes

t, dt, tmax = 0 , 1, 30 #initialisation des parametres de temps
x = [0]          #initialisation d une liste avec les dates
y0 = 0        #initialisation de la vitesse
y=[0]             #initialisation d une liste avec les vitesses
#ouverture du fichier pour stocker liste de points :
fichier=open("euler-chute.txt", w )
while t < tmax : #Tant que la date est inferieure a la date max
       t = t + dt    #on incremente la date avec le pas de calcul
       #on ajoute cette nouvelle date a la fin de la liste des dates
       #(round permet l affichage de deux decimales) :
       x.append(round(t,2))
       y1 = y0 + (-0.00203*y0*y0+9.8)*dt #on calcule la vitesse a l instant n+1
       #on affecte a l ancienne vitesse la nouvelle pour la boucle suivante :
       y0 = y1
       #on ajoute la nouvelle vitesse a la fin de la liste des vitesses :
       y.append(round(y1,2))
       #on ecrit la date et la vitesse a l instant n+1 dans le fichier :
       fichier.write(str(round(t,2))+"\t"+str(round(y1,2))+"\n")
       fichier.close()   #on ferme le fichier de stockage
       print (x,y)  #on affiche les valeurs de temps et de vitesses calculees

# Tracer courbes : b pour bleu  ; - pour ligne continue  ; #x pour les marqueurs en
    croix :
plt.plot(x,y,"b-x", label="pas de 1s")
plt.xlabel("temps (s)") #nom de l axe des abscisses
plt.ylabel("vitesse (m/s)") #nom de l axe des abscisses
plt.legend(loc= lower right ) # position de la legende

plt.show()              #on montre le graphique
```

Télécharger le source de ce code

L'intérêt de ce type de code est de pouvoir changer le pas de calcul aisément, sans avoir besoin de recopier des formules.

Sur la figure 5 on oberve la fenêtre graphique obtenue.

On peut également tracer plusieurs courbes avec différents tests de pas (le script demande les deux pas à tester), ainsi que montrer le tracé de la solution exacte : voici un autre code.

2.2 Méthode de Runge-Kutta d'ordre 4

2.2.1 Schéma global

Cette méthode plus complexe fait intervenir quatre fois plus de calculs que la méthode d'Euler (pour une équation différentielle du premier ordre), mais est beaucoup plus fiable.

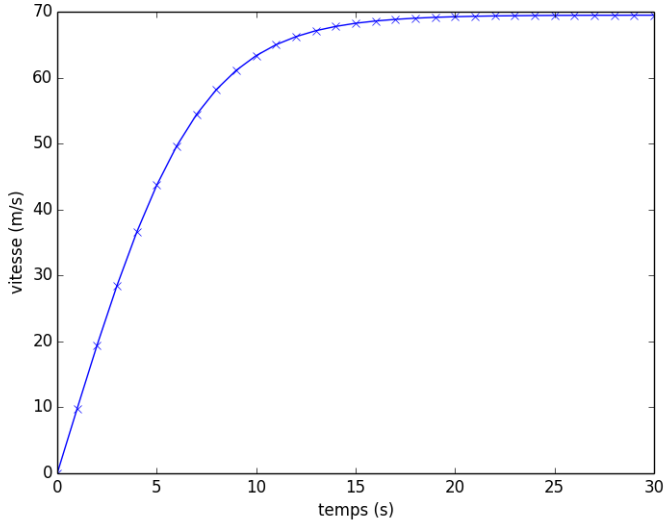

FIGURE 5 – Mise en oeuvre de la méthode d'Euler explicite avec Python : graphique

Voici les étapes de calculs :

$$\begin{cases} y(t=0) = y_0 \\ k_{10} = f(t_0,y_0) \times \mathrm{d}t \\ k_{20} = f\left(t_0 + \dfrac{\mathrm{d}t}{2}, y_0 + \dfrac{k_1}{2}\right) \times \mathrm{d}t \\ k_{30} = f\left(t_0 + \dfrac{\mathrm{d}t}{2}, y_0 + \dfrac{k_2}{2}\right) \times \mathrm{d}t \\ k_{40} = f\left(t_0 + \mathrm{d}t, y_0 + k_3\right) \times \mathrm{d}t \\ y_1 = y_0 + \dfrac{1}{6}\left(k_{10} + 2\,k_{20} + 2\,k_{30} + k_{40}\right) \\ \ldots \end{cases} \tag{7}$$

Il y a donc quatre coefficients à calculer (méthode d'ordre 4, il existe également la méthode RK2) qui correspondent à l'évaluation de 4 pentes à des instants différents entre t et $t + \mathrm{d}t$ (une évaluation en t, deux évaluations en $t + \dfrac{\mathrm{d}t}{2}$, une évaluation en $t + \mathrm{d}t$). Voici un graphique qui montre le lien entre les différents coefficients k :

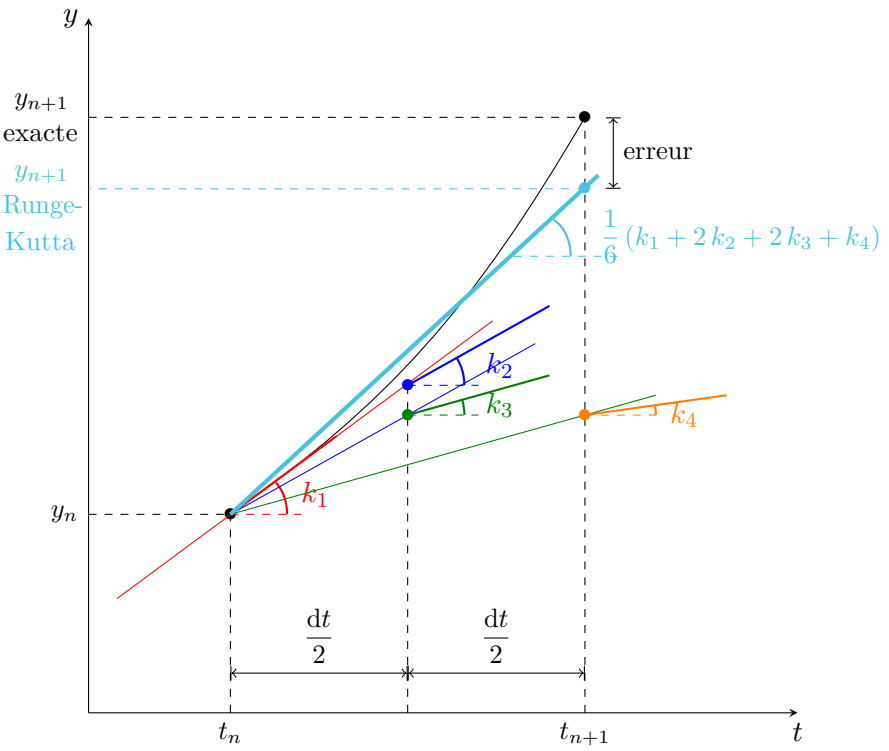

FIGURE 6 – Evaluation des pentes pour la méthode de Runge-Kutta d'ordre 4

On calcule le coefficient k_1 (qui correspond à l'unique coefficient de la méthode d'Euler), celui nous permet de trouver l'ordonnée du point où l'on évalue k_2 (tracé rouge), son abscisse étant égale à $t_n + \frac{dt}{2}$. Le coefficient k_2 permet de trouver l'ordonnée du point où l'on évalue k_3 (tracé bleu), l'abscisse restant inchangée.
On évalue une dernière fois la pente, coefficient k_4, en un point d'abscisse t_{n+1} et dont l'ordonnée est donnée par le coefficient k_3 (tracé vert).

La pente retenue pour l'évaluation de y_{n+1} est une pondération des pentes évaluées précédemment (tracé bleu ciel).

2.2.2 Mise en oeuvre sur l'exemple

De la même manière que précédemment et pour pouvoir comparer les méthodes, on choisit un pas de calcul de référence de 1s, et un temps total de 30s.

2.2.2.1 Calcul des premiers termes à la main

On a $v_0 = 0$. On calcule :

- $k_{10} = (A \times v_0^2 + B) \times dt = (-2{,}03 \times 10^{-3} \times 0^2 + 9{,}81) \times 1 = 9{,}81$
- $k_{20} = \left(A \times \left(v_0 + \frac{k_{10}}{2}\right)^2 + B\right) \times dt = \left(-2{,}03 \times 10^{-3} \times \left(0 + \frac{9{,}81}{2}\right)^2 + 9{,}81\right) \times 1 = 9{,}76$
- $k_{30} = \left(A \times \left(v_0 + \frac{k_{20}}{2}\right)^2 + B\right) \times dt = \left(-2{,}03 \times 10^{-3} \times \left(0 + \frac{9{,}76}{2}\right)^2 + 9{,}81\right) \times 1 = 9{,}76$
- $k_{40} = \left(A \times (v_0 + k_{30})^2 + B\right) \times dt = \left(-2{,}03 \times 10^{-3} \times \left(0 + \frac{9{,}76}{2}\right)^2 + 9{,}81\right) \times 1 = 9{,}62$

Donc on obtient pour la première valeur de vitesse :

$$v_1 = v_0 + \frac{1}{6}\left(k_{10} + 2\,k_{20} + 2\,k_{30} + k_{40}\right) \tag{8}$$
$$= 0 + \frac{1}{6}(9{,}81 + 2 * 9{,}76 + 2 * 9{,}76 + 9{,}62) \tag{9}$$
$$= 9{,}75 \tag{10}$$

Voici les calculs permettant d'obtenir v_2 :

- $k_{11} = (A \times v_1^2 + B) \times \mathrm{d}t = (-2{,}03 \times 10^{-3} \times 9{,}75^2 + 9{,}81) \times 1 = 9{,}62$
- $k_{21} = \left(A \times \left(v_1 + \frac{k_{10}}{2}\right)^2 + B\right) \times \mathrm{d}t = \left(-2{,}03 \times 10^{-3} \times \left(9{,}75 + \frac{9{,}81}{2}\right)^2 + 9{,}81\right) \times 1 = 9{,}38$
- $k_{31} = \left(A \times \left(v_1 + \frac{k_{20}}{2}\right)^2 + B\right) \times \mathrm{d}t = \left(-2{,}03 \times 10^{-3} \times \left(9{,}75 + \frac{9{,}76}{2}\right)^2 + 9{,}81\right) \times 1 = 9{,}39$
- $k_{41} = \left(A \times (v_1 + k_{30})^2 + B\right) \times \mathrm{d}t = \left(-2{,}03 \times 10^{-3} \times \left(9{,}75 + \frac{9{,}76}{2}\right)^2 + 9{,}81\right) \times 1 = 9{,}07$

$$v_2 = v_1 + \frac{1}{6}\left(k_{11} + 2\,k_{21} + 2\,k_{31} + k_{41}\right) \tag{11}$$
$$= 9{,}75 + \frac{1}{6}(9{,}62 + 2 * 9{,}38 + 2 * 9{,}39 + 9{,}07) \tag{12}$$
$$= 19{,}12 \tag{13}$$

2.2.2.2 Utilisation d'un tableur

Dans le fichier de calcul, il faudra créer 6 colonnes : une pour le temps, une pour la vitesse et une pour chaque coefficient k.
Une fois les formules correctement entrées, il suffit de les recopier vers le bas jusqu'au temps final voulu.

Voici les premières cellules du tableau :

	A	B	C
1	t1	v_RK4	k1
2	0,00	0,00	=(-0,00203*B2^2+9,81)*$A$3
3	=A2+1	=B2+(1/6)*(C2+2*D2+2*E2+F2)	=(-0,00203*B3^2+9,81)*$A$3
4	=A3+1	=B3+(1/6)*(C3+2*D3+2*E3+F3)	=(-0,00203*B4^2+9,81)*$A$3

D	E	F
k2	k3	k4
=(-0,00203*(B2+C2/2)^2+9,81)*$A$3	=(-0,00203*(B2+D2/2)^2+9,81)*$A$3	=(-0,00203*(B2+E2)^2+9,81)*$A$3
=(-0,00203*(B3+C3/2)^2+9,81)*$A$3	=(-0,00203*(B3+D3/2)^2+9,81)*$A$3	=(-0,00203*(B3+E3)^2+9,81)*$A$3
=(-0,00203*(B4+C4/2)^2+9,81)*$A$3	=(-0,00203*(B4+D4/2)^2+9,81)*$A$3	=(-0,00203*(B4+E4)^2+9,81)*$A$3

FIGURE 7 – Premières cellules de la feuille de calculs utilisée pour la méthode de Runge-Kutta

Télécharger le fichier de calcul

2.2.2.3 Utilisation de Python

```
import numpy, math, matplotlib
from pylab import *
t, dt, tmax = 0 , 1 , 30 #initialisation du temps
x = [0] #creation liste des temps
y0 = 0 #condition initiale de vitesse
```

```
y=[0]  #creation liste des vitesses

while t < tmax : #tant que t est inferieur a tmax
    t = t + dt #on incremente le temps
    x.append(round(t,2)) # on rentre ce temps dans la liste
    #calcul des coefficients RK4
    k1 = (-0.00203*y0**2+9.81)*dt
    k2 = (-0.00203*(y0+0.5*k1)**2+9.81)*dt
    k3 = (-0.00203*(y0+0.5*k2)**2+9.81)*dt
    k4 = (-0.00203*(y0+k3)**2+9.81)*dt
    # calcul de la vitesse a n+1
    y1 = y0 + (1/6.0)*(k1 + 2*k2 + 2*k3 + k4)*dt
    y0 = y1 # on remplace la valeur de la vitesse a n par celle a n+1
    y.append(round(y1,2)) #on rentre la vitesse dans la liste

print (x,y) #on affiche les valeurs de temps et de vitesse dans la console

plt.plot(x,y,"b-x") #b pour bleu  ; - pour ligne continue  ; x pour les marqueurs en
    croix
plt.ylabel("$v$ ($\mathrm{m.s^{-1}}$)") #legende de l axe des y
plt.ylim(ymin = 0, ymax = 80) # on peut fixer les limites des axes
plt.xlabel("$t$ ($s$)") #legende de l axe des x
plt.title("Chute 1D avec frottements quadratiques methode RK4") # titre du graphique
plt.show() # on montre le graphique
```

Télécharger le source de ce code

3 Equation du deuxième ordre : étude de la position et de la vitesse (angulaire) d'un pendule simple

On étudie l'équation différentielle suivante :

$$\frac{\mathrm{d}^2\theta}{\mathrm{dt}^2} + \omega_0^2 \sin\theta = 0 \tag{14}$$

θ est donc la position angulaire, ω_0, la pulsation propre des oscillations. On notera également par la suite Ω la vitesse angulaire du pendule.

On prendra une pulsation propre de façon à ce que $\omega_0^2 = 40\,\mathrm{rad.s^{-1}}$, l'oscillateur partira de la position angulaire $\theta_0 = 0$ avec une vitesse angulaire initiale non nulle $\Omega_0 = \dot{\theta}_0 = 2\,\mathrm{rad.s^{-1}}$.

Ce type d'équation différentielle est traitée numériquement sous forme matricielle :

$$\begin{cases} \Omega = \dfrac{\mathrm{d}\theta}{\mathrm{d}t} & \Longrightarrow \quad \mathrm{d}\theta = \Omega\,\mathrm{d}t \\ \dot{\Omega} = \dfrac{\mathrm{d}\Omega}{\mathrm{d}t} = -\omega_0^2 \sin\theta & \Longrightarrow \quad \mathrm{d}\Omega = -\omega_0^2 \sin\theta\,\mathrm{d}t \end{cases} \tag{15}$$

Ainsi le calcul des θ et des Ω sont liés, on a besoin d'Ω pour calculer θ et de θ pour calculer Ω.

On choisira un pas de calcul de 0,02 s.

3.1 Méthode d'Euler explicite

3.1.1 Calculs manuels

Comme précédemment dans ce document, montrons les premiers calculs manuellement :

$$\begin{aligned}\theta_0 &= 0\\ \theta_1 &= \theta_0 + \mathrm{d}\theta\\ &= \theta_0 + \Omega_0\,\mathrm{d}t\\ &= 0 + 2\times 0{,}02 = 0{,}04\\ \theta_2 &= \theta_1 + \mathrm{d}\theta\\ &= \theta_1 + \Omega_1\,\mathrm{d}t\\ &= 0{,}04 + 2\times 0{,}02 = 0{,}08\\ &\dots\end{aligned}$$

$$\begin{aligned}\Omega_0 &= 2\\ \Omega_1 &= \Omega_0 + \mathrm{d}\Omega\\ &= \Omega_0 - \omega_0^2\,\sin\,\theta_0\,\mathrm{d}t\\ &= 2 - 40\times\sin\,0\times 0{,}02 = 2\\ \Omega_2 &= \Omega_1 + \mathrm{d}\Omega\\ &= \Omega_1 - \omega_0^2\,\sin\,\theta_1\,\mathrm{d}t\\ &= 2 - 40\times\sin\,0{,}04\times 0{,}02 = 1{,}97\\ &\dots\end{aligned}$$

3.1.2 Utilisation d'un tableur

Voici les premières cellules du tableur :

	A	B	C	D
1	t	theta	omega	delta t
2	0,00	0,00	2,00	0,02
3	=A2+$D$2	=B2+C2*$D$2	=C2-40*SIN(B2)*$D$2	
4	=A3+$D$2	=B3+C3*$D$2	=C3-40*SIN(B3)*$D$2	

Et le graphique obtenu avec cette méthode :

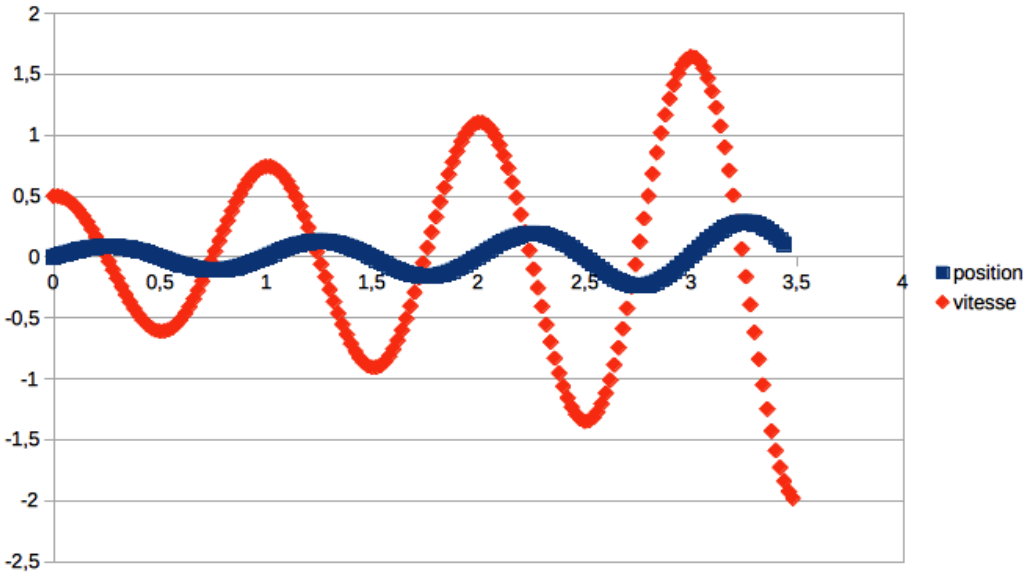

FIGURE 8 – Résultat graphique de la méthode d'Euler explicite dans le cas du pendule

Ainsi nous voyons que la **méthode d'Euler explicite est divergente**.
On peut utiliser la méthode d'Euler pour résoudre le pendule simple, mais il faut mettre en oeuvre la **méthode implicite** (voir ci-dessous).

Le fichier de calcul est téléchargeable ci-dessous, il comporte trois feuilles avec la méthode d'Euler explicite, l'implicite et la méthode Runge-Kutta d'ordre 4.

Tableur

3.2 Méthode d'Euler implicite

3.2.1 Principe

Celui-ci est simple, à la place d'évaluer la pente en t_n pour calculer y_{n+1}, on évalue cette pente en t_{n+1} :

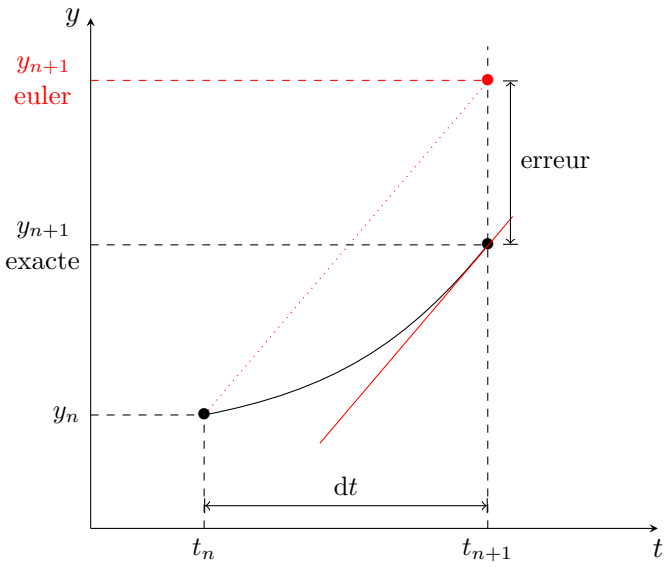

FIGURE 9 – Principe de la méthode d'Euler **explicite**

3.2.2 Calculs manuels

Il suffit donc de remplacer θ_n par θ_{n+1} dans l'expression du calcul des Ω :

$$\begin{aligned}
\theta_0 &= 0 \\
\theta_1 &= \theta_0 + \mathrm{d}\theta \\
&= \theta_0 + \Omega_0\,\mathrm{d}t \\
&= 0 + 2 \times 0{,}02 = 0{,}04 \\
\theta_2 &= \theta_1 + \mathrm{d}\theta \\
&= \theta_1 + \Omega_1\,\mathrm{d}t \\
&= 0{,}04 + 2 \times 0{,}02 = 0{,}08 \\
&\dots
\end{aligned}
\qquad
\begin{aligned}
\Omega_0 &= 2 \\
\Omega_1 &= \Omega_0 + \mathrm{d}\Omega \\
&= \Omega_0 - \omega_0^2 \sin\theta_1\,\mathrm{d}t \\
&= 2 - 40 \times \sin 0{,}04 \times 0{,}02 = 1{,}97 \\
\Omega_2 &= \Omega_1 + \mathrm{d}\Omega \\
&= \Omega_1 - \omega_0^2 \sin\theta_2\,\mathrm{d}t \\
&= 2 - 40 \times \sin 0{,}08 \times 0{,}02 = 1{,}90 \\
&\dots
\end{aligned}$$

Les premières valeurs ne sont pas bien différentes de celle de la méthode d'Euler explicite, mais ensuite tout change :

3.2.3 Utilisation d'un tableur

Voici les premières cellules du tableur :

	A	B	C	D
1	t	theta	omega	delta t
2	0,00	0,00	2,00	0,02
3	=A2+$D$2	=B2+C2*$D$2	=C2-40*SIN(B3)*$D$2	
4	=A3+$D$2	=B3+C3*$D$2	=C3-40*SIN(B4)*$D$2	

Et le graphique obtenu avec cette méthode :

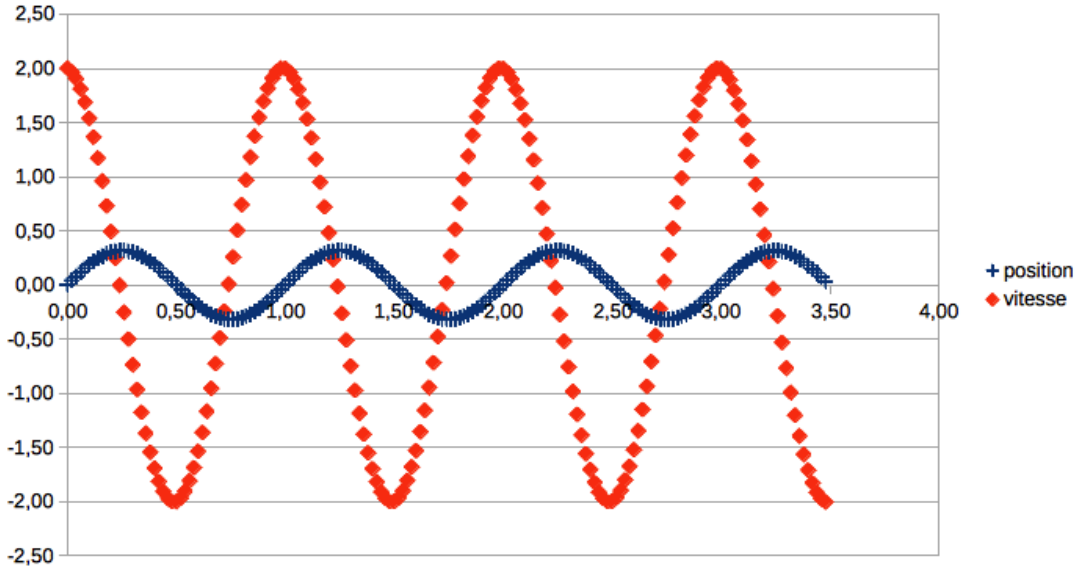

FIGURE 10 – Résultat graphique de la méthode d'Euler **implicite** dans le cas du pendule

Cette méthode converge, nous obtenons des belles oscillations ...

Rappel : le fichier ci-contre permet de retrouver les calculs pour les 3 méthodes : Tableur

3.2.4 Utilisation de python

Montrons le code pour la méthode implicite qui fonctionne :

```
# -*- coding : utf8 -*-
import numpy, math, matplotlib
from pylab import *
import matplotlib.pyplot as plt
t, dt, tmax = 0 , 0.02 , 3  #initialisation des termes de temps
T = [0]                 #initialisation d une liste avec les dates
x0 = 0                 #initialisation de la position
x=[x0]                  #initialisation d une liste avec les positions
v0 = 0.5 #initialisation de la vitesse
v=[v0] #initialisation d une liste avec les vitesse
fichier=open("euler-oscillateur.txt",'w') #ouverture du fichier pour stocker liste de
    points

while t < tmax : #tant que la date est inferieur a la date max
   t = t + dt  #on ajoute le pas a la date precedente pour creer la date t+1
   T.append(round(t,2)) # on rentre la nouvelle date a la fin de la liste des dates
   x1 = x0 + v0*dt #on calcule la position a t+1
   v1 = v0 + (-40*sin(x1)*dt) #on calcule la vitesse a t+1
   x0 = x1 #on remplace la position a t par la position a t+1
   x.append(round(x1,2)) #on rentre na nouvelle position a la la fin de la liste des
       positions
   v0 = v1 #on remplace la position a t par la position a t+1
   v.append(round(v1,2)) #on rentre na nouvelle vitesse a la la fin de la liste des
       positions
   fichier.write(str(t)+"\t"+str(round(x1,2))+"\t"+str(round(v1,2))+"\n") #on ecrit la
        date, position et vitesse a t+1 dans le fichier
```

```
fichier.close() # on ferme le fichier de stockage

# print T,x,v # on affiche les points de mesure

plt.plot(T,x,"b-x", label="position") # on trace la position en fonction du temps  : b
    pour bleu  ; - pour ligne continue  ; x pour les marqueurs en croix
plt.plot(T,v,"r-x", label="vitesse") # on trace la vitesse en fonction du temps en
   rouge
plt.xlabel("temps") #nom de l axe des abscisses
plt.legend() # on affiche les legende
plt.savefig('euler-oscillateur.pdf', format='pdf') #on enregistre le graphique dans un
    fichier pdf

plt.axis([0,3,-0.75,0.75])
plt.axhline(linewidth=1, color='k')

plt.show()
```

Télécharger le source de ce code

3.3 Méthode de Runge-Kutta d'ordre 4

Pour cette équation différentielle d'ordre 2, comme nous l'avons décomposée en deux équations du premier ordre, nous aurons non pas 4 mais 8 coefficients à calculer : 4 pour l'évaluation de θ et 4 pour l'évaluation de Ω.

Voici le principe :

$$\begin{cases} \theta_0 &= 0 \\ k_1 &= \Omega_0 \, \mathrm{d}t \\ k_2 &= \left(\Omega_0 + \dfrac{j_1}{2}\right) \mathrm{d}t \\ k_3 &= \left(\Omega_0 + \dfrac{j_2}{2}\right) \mathrm{d}t \\ k_4 &= (\Omega_0 + j_3) \, \mathrm{d}t \\ \theta_1 &= \theta_0 + \dfrac{1}{6}\left(k_1 + 2\,k_2 + 2\,k_3 + k_4\right) \end{cases} \qquad \begin{cases} \Omega_0 &= 2 \\ j_1 &= -\omega_0^2 \sin\theta_0 \, \mathrm{d}t \\ j_2 &= -\omega_0^2 \sin\left(\theta_0 + \dfrac{k_1}{2}\right) \mathrm{d}t \\ j_3 &= -\omega_0^2 \sin\left(\theta_0 + \dfrac{k_2}{2}\right) \mathrm{d}t \\ j_4 &= -\omega_0^2 \sin\left(\theta_0 + k_3\right) \mathrm{d}t \\ \Omega_1 &= \Omega_0 + \dfrac{1}{6}\left(j_1 + 2\,j_2 + 2\,j_3 + j_4\right) \end{cases}$$

3.3.1 Calculs manuels

$$\theta_0 = 0$$
$$\begin{aligned} k_1 &= \Omega_0 \, \mathrm{d}t \\ &= 2 \times 0{,}02 \\ &= 0{,}04 \end{aligned}$$
$$\begin{aligned} k_2 &= \left(\Omega_0 + \frac{j_1}{2}\right) \mathrm{d}t \\ &= \left(2 + \frac{0}{2}\right) \times 0{,}02 \\ &= 0{,}04 \end{aligned}$$
$$\begin{aligned} k_3 &= \left(\Omega_0 + \frac{j_2}{2}\right) \mathrm{d}t \\ &= (2 + \frac{-0{,}016}{2}) \times 0{,}02 \\ &= 0{,}0398 \end{aligned}$$
$$\begin{aligned} k_4 &= (\Omega_0 + j_3) \, \mathrm{d}t \\ &= (2 - 0{,}016) \times 0{,}02 \\ &= 0{,}0396 \end{aligned}$$
$$\begin{aligned} \theta_1 &= \theta_0 + \frac{1}{6}\left(k_1 + 2\,k_2 + 2\,k_3 + k_4\right) \\ &= 0 + \frac{1}{6}(0{,}04 + 2 \times 0{,}04 + 2 \times 0{,}0398 + 0{,}0396) \\ &= 0{,}0399 \end{aligned}$$

$$\Omega_0 = 2$$
$$\begin{aligned} j_1 &= -\omega_0^2 \sin\theta_0 \, \mathrm{dt} = \\ &= -40 \times \sin 0 \times 0{,}02 \\ &= 0 \end{aligned}$$
$$\begin{aligned} j_2 &= -\omega_0^2 \sin\left(\theta_0 + \frac{k_1}{2}\right) \mathrm{dt} \\ &= -40 \times \sin\left(0 + \frac{0{,}04}{2}\right) \times 0{,}02 \\ &= -0{,}016 \end{aligned}$$
$$\begin{aligned} j_3 &= -\omega_0^2 \sin\left(\theta_0 + \frac{k_2}{2}\right) \mathrm{dt} \\ &= -40 \times \sin\left(0 + \frac{0{,}04}{2}\right) \times 0{,}02 \\ &= -0{,}016 \end{aligned}$$
$$\begin{aligned} j_4 &= -\omega_0^2 \sin\left(\theta_0 + k_3\right) \mathrm{dt} \\ &= -40 \times \sin\left(0 + 0{,}0398\right) \times 0{,}02 \\ &= -0{,}0318 \end{aligned}$$
$$\begin{aligned} \Omega_1 &= \Omega_0 + \frac{1}{6}\left(j_1 + 2\,j_2 + 2\,j_3 + j_4\right) \\ &= 2 + \frac{1}{6}(0 + 2 \times -0{,}016 + 2 \times -0{,}016 + -0{,}0318) \\ &= 1{,}984 \end{aligned}$$

3.3.2 Utilisation du tableur

	A	B	C	D	E	F
1	t	theta	k1	k2	k3	k4
2	0,00	0	=G2*$A$3	=(G2+H2/2)*$A$3	=(G2+I2/2)*$A$3	=(G2+J2)*$A$3
3	=A2+0,02	=B2+(C2+2*D2+2*E2+F2)/6	=G3*$A$3	=(G3+H3/2)*$A$3	=(G3+I3/2)*$A$3	=(G3+J3)*$A$3

G	H	I	J	K
omega	j1	j2	j3	j4
2,000	=-40*SIN(B2)*$A$3	=(-40*SIN(B2+(C2/2))*$A$3)	=(-40*SIN(B2+(D2/2))*$A$3)	=(-40*SIN(B2+F2)*$A$3)
=G2+(H2+2*I2+2*J2+K2)/6	=-40*SIN(B3)*$A$3	=(-40*SIN(B3+(C3/2))*$A$3)	=(-40*SIN(B3+(D3/2))*$A$3)	=(-40*SIN(B3+F3)*$A$3)

Voici le résultat obtenu :

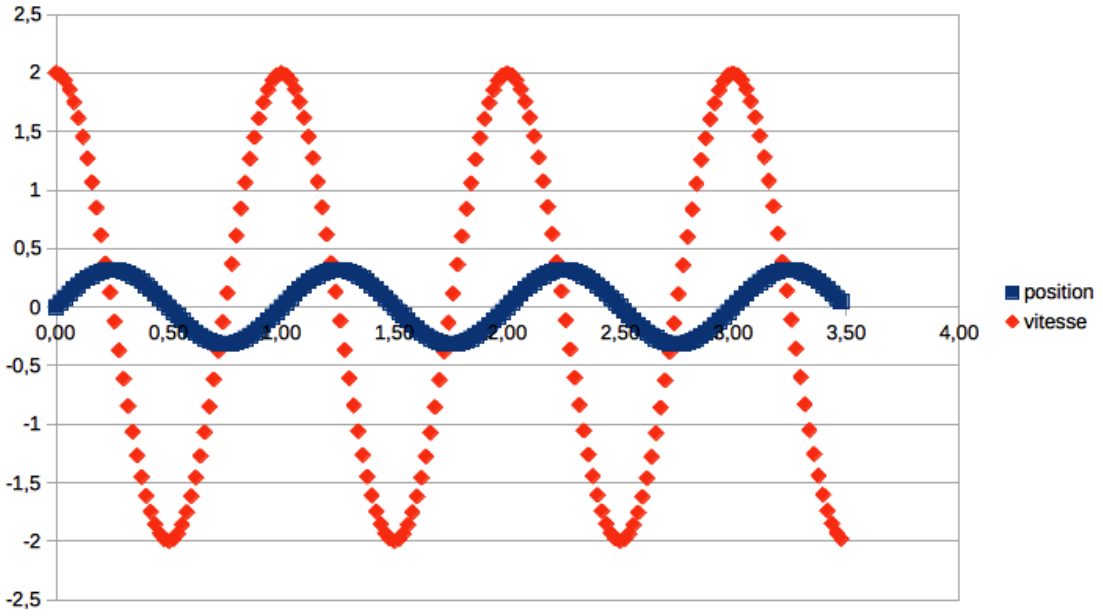

FIGURE 11 – Résultat graphique de la méthode RK4 dans le cas du pendule

Nous avons de nouveau des belles oscillations.

Rappel : le fichier ci-contre permet de retrouver les calculs pour les 3 méthodes : Tableur

3.3.3 Utilisation de Python

Voici le code du programme python, sans grande surprise :

```
import numpy, math, matplotlib
import math
from pylab import *
puls = 40
t, dt, tmax = 0 , 0.02 , 2
x = [0] #liste des temps
theta0 = 0
omega0 = 2
y = [0] #liste des theta
w = [0] #liste des omega
while t < tmax :
    x.append(round(t,2))
    k1 = omega0*dt
    j1 = -puls*math.sin(theta0)*dt
    k2 = (omega0+j1/2)*dt
    j2 = -puls*math.sin(theta0+(k1/2))*dt
    k3 = (omega0+j2/2)*dt
    j3 = -puls*math.sin(theta0+k2/2)*dt
    k4 = (omega0+j3)*dt
    j4 = -puls*math.sin(theta0+k3)*dt
    theta1 = theta0 + (1/6.0)*(k1 + 2*k2 + 2*k3 + k4)
    y.append(round(theta1,5))
```

```
    theta0 = theta1
    omega1 = omega0 + (1/6.0)*(j1 + 2*j2 + 2*j3 + j4)
    w.append(round(omega1,5))
    omega0 = omega1
    t = t + dt
print (x,y) # impression de theta en fonction de t
print (x,w) # impression de theta en fonction de t

plt.plot(x,y,"b-x") #b pour bleu  ; - pour ligne continue  ; x pour les marqueurs en
    croix
plt.plot(x,w,"r-x") #b pour bleu  ; - pour ligne continue  ; x pour les marqueurs en
    croix

plt.show()
```

Télécharger le source de ce code

3.3.3.1 Animation

Python, langage choisi pour la programmation dans l'éducation, permet de faire de "belles animations". Voici deux codes permettant d'animer un pendule simple.

Création d'un gif animé du pendule simple sans frottement

```
# -*- coding : utf-8 -*-

"""
Animation d'une particule se baladant sur une ellipse
"""

from __future__ import division
from scipy import *
from pylab import *
import os

#initialisation
t = 0.0 #intitialisation du temps
tfin = 4 #temps final
listx=[]  #initialisation liste des x
listy=[]   #initialisation liste des y
puls = 40 #pulsation au carre
theta0 = 0 #angle initial
omega0 = 5 #Vitesse angulaire initiale
dt = 0.05 #pas de calcul
while t < tfin : # Tant que t n a pas atteint tfin
# calculs des coefficients :
    k1 = omega0*dt
    j1 = -puls*math.sin(theta0)*dt
    k2 = (omega0+j1/2)*dt
    j2 = -puls*math.sin(theta0+(k1/2))*dt
    k3 = (omega0+j2/2)*dt
```

```
    j3 = -puls*math.sin(theta0+k2/2)*dt
    k4 = (omega0+j3)*dt
    j4 = -puls*math.sin(theta0+k3)*dt
    theta1 = theta0 + (1/6.0)*(k1 + 2*k2 + 2*k3 + k4)
    omega1 = omega0 + (1/6.0)*(j1 + 2*j2 + 2*j3 + j4)
    #Passage en coordonnees cartesiennes :
    x = -3*math.cos(theta1)
    y = 3*math.sin(theta1)
    listx.append(x) #ajout dans la liste des x
    listy.append(y) #ajout dans la liste des y
    #pour boucler :
    theta0 = theta1
    omega0 = omega1
    t = t + dt

# Construction d une serie d images et de leur assemblage dans une animation
for i in range(0,len(listx),1) : #pour toute la liste des x (1 pour les prendre tous)
    plot([0,listy[i]],[0,listx[i]])# on trace du point (0,x) a (0,y)
    axis([-5, 5, -5, 5]) #limites des axes
    filename = 'fichierTemp'+str('%02d' %i)+'.pdf' #Fichiers temporaires pour la
        construction du gif
    savefig(filename)
    print 'Nplot = ', i #dans la console on affiche la construction des images
    clf()

# convert est une fonction d ImageMagick  : option -delay en 1/100 de seconde
#conversion des n images en gif, il faut bien definir le chemin auquel on trouve
    convert
#instructions a fournir au systeme, creation de pendule.gif :
cmd = '/opt/local/bin/convert fichierTemp*.pdf pendule.gif'
os.system(cmd) #execution
os.system('rm *.pdf')  # destruction des fichiers temporaires
print "C'est fini  !"
```

Observer le résultat

Télécharger le source de ce code

Création d'un petit film directement dans python : pendule simple avec frottement

```
# -*- coding : utf-8 -*-

"""
This short code snippet utilizes the new animation package in matplotlib 1.1.0
It's the shortest snippet that I know of that can produce an animate plot in
python. I'm hoping that the animation package can be improved so that one could
more simply animate things. What do you think ?
"""
import numpy as np
import math
from pylab import *
```

```
from matplotlib import gridspec
# import matplotlib.pyplot as plt
import matplotlib.animation as animation

##### Fonction de calcul #####
def simData() :
# this function is called as the argument for
# the simPoints function. This function contains
# (or defines) and iterator---a device that computes
# a value, passes it back to the main program, and then
# returns to exactly where it left off in the function.
# I believe that one has to use this method to animate a plot
# using the matplotlib animation package.

#initialisation
    puls = 40 #puls au carre
    lbd = 0.25 #coeff de frottement
    t_max = 10.0 #temps de l anim
    dt = 0.02 # pas de calcul
    theta0 = -1    #condition initial en theta
    omega0 = 15 #condition initial en omega
    t = 0.0 #initialisation du temps
    while t < t_max :
        #Calculs des coefficients de RK4 :
        j1 = omega0*dt
        k1 = (-2*lbd*omega0-puls*math.sin(theta0))*dt
        j2 = (omega0+k1/2)*dt
        k2 = (-2*lbd*omega0-puls*math.sin(theta0+(j1/2)))*dt
        j3 = (omega0+k2/2)*dt
        k3 = (-2*lbd*omega0-puls*math.sin(theta0+j2/2))*dt
        j4 = (omega0+k3)*dt
        k4 = (-2*lbd*omega0-puls*math.sin(theta0+j3))*dt
        theta1 = theta0 + (1/6.0)*(j1 + 2*j2 + 2*j3 + j4)
        omega1 = omega0 + (1/6.0)*(k1 + 2*k2 + 2*k3 + k4)
        # Passage en cartesien :
        x = -2*math.cos(theta1)
        y = 2*math.sin(theta1)
        #pour boucler :
        theta0 = theta1
        omega0 = omega1
        t = t + dt
        yield x, y, t, theta1 #renvoie des valeurs
 #(yield = iterateur, les valeurs sont transmises mais pas stockees)

##### Fonction graphique ####
theta = [] #liste qui contiendra tous les theta calcules
vectt = [] #liste qui contiendra les temps correspondants
def simPoints(simData) :
    #Recuperation des valeurs de la fonction simdata retourne par yield
    x, y, t, theta1 = simData[0], simData[1], simData[2], simData[3]
    #Passage de theta en degre
```

```
    theta.append(round(theta1*180/np.pi,0)) #on ajoute le theta a la liste des theta
    vectt.append(t) #on ajoute le temps a la liste des temps
    theta1 = round(theta1*180/np.pi,0) #passage de l angle en degre
    time_text.set_text(time_template%(t)) #affichage du temps
    angle_text.set_text(angle_template%(theta1)) #affichage de l angle
    x = [0,x] #permet le dessin du point (0,0)
    y = [0,y]
    line.set_data(y, x) #on tracera y en fonction de x
    linebis.set_data(vectt,theta) #et theta en fonction de t
    return line, linebis, time_text, angle_text

#####  set up figure for plotting : #####
fig = plt.figure()
# fig = plt.figure()
gs = gridspec.GridSpec(2, 1,height_ratios=[6,2]) #creation d une grille avec plusieurs
     plot
ax = plt.subplot(gs[0]) # premier plot
# ax.axis('equal') #axes premiere figure normes
ax.set_aspect('equal',adjustable='box') #axes orthonorme
ax.set_xlim(-4, 4) #limite en x
ax.set_ylim(-4.0, 4.0) #limite en y
line, = ax.plot([], [], 'bo-',lw=3, ms=10) #trace du premier plot
axbis = plt.subplot(gs[1]) # deuxieme plot
axbis.set_xlim(0, 10) #limite en x
axbis.set_ylim(0, 600) #limite en y
linebis, = axbis.plot([], [], 'bo-',lw=3, ms=1)#trace du deuxieme plot

# option graphique :
# bo  : blue ball  ; -  : trait  ; lw  : line width  ; ms  : taille  : ball

time_template = 'Time = %.1f s' # prints running simulation time arrondi a 1 decimal
angle_template = 'Angle = %.1f deg'
time_text = ax.text(0.05, 0.9, '', transform=ax.transAxes)# coordonnees du texte
angle_text = ax.text(0.05, 0.8, '', transform=ax.transAxes)

#### Now call the animation package : ####
ani = animation.FuncAnimation(fig, simPoints, simData, blit=False, interval=50, repeat
   =False)
plt.show()
```

Voici le résultat présenté dans un fichier mp4 (qu'il doit être possible de générer directement avec python ...) :

Vidéo du résultat

Télécharger le source de ce code

4 Conclusion

La méthode d'Euler permet une première approche de la résolution numérique d'équations différentielles, l'algorithme est simple et rapide (avec un bon choix de pas de calcul). Si les résultats se montrent incohérents avec la méthode d'Euler classique dit explicite, on peut essayer la méthode implicite, plus stable.

Quant à elle, la méthode de Runge-Kutta d'ordre 4 est plébiscité par sa précision et sa stabilité. A choisir, c'est elle que l'on mettra en oeuvre.
A noter que ces méthodes divergent lorsque l'on aborde des problèmes de systèmes conservatifs aux temps longs : pour plus d'informations à ce sujet ainsi qu'au sujet des erreurs introduites par les méthodes numériques utilisées ici, je vous invite à consulter les références de l'excellent site de Jimmy Roussel (voir ci-dessous).

Références

- Un excellent document de M. Schwing de l'IUFM de Loraine
- La page outils et méthodes pour la physique du site de Jimmy Roussel avec notamment :
  - Méthode d'Euler
  - Méthodes de Runge-Kutta
  - Remédiation aux méthodes classiques : méthode de Verlet
  - Erreurs numériques