Méthode d'Euler - Principe et squelette

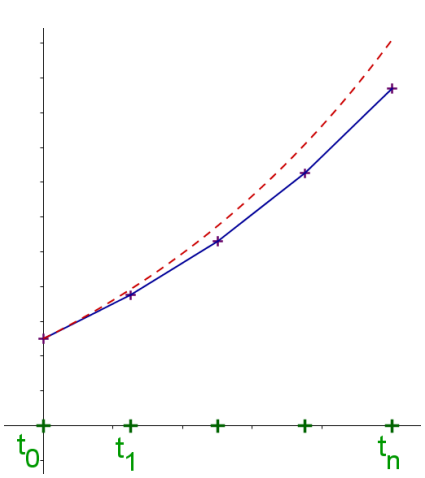
Objectif. Résoudre numériquement l'équation différentielle :

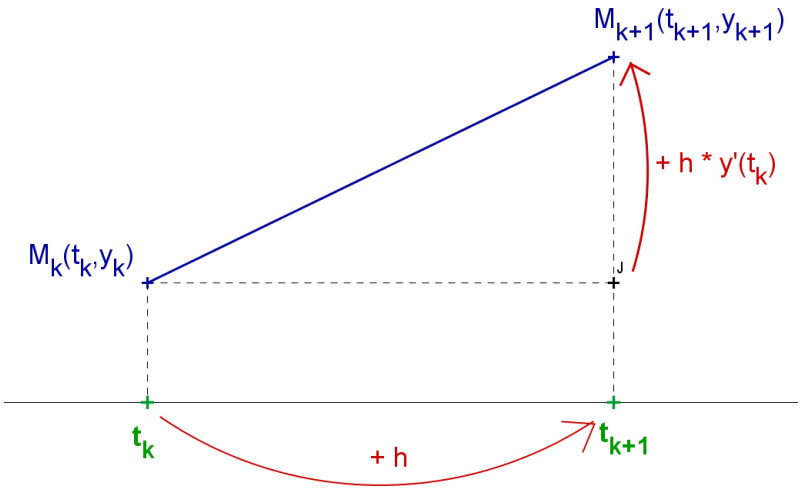
$$y'(t) = F(t, y(t)) \text{ sur l'intervalle } [a, b], \text{ avec condition initiale : } y(0) = y_0$$

Moyen. On construit une ligne polygonale $M_0M_1 \dots M_n$ approchant la courbe représentative de la solution exacte en choisissant un nombre de points (ou un pas, c'est équivalent), et en construisant les points M_k essentiellement à l'aide d'un DL à l'ordre 1, dont on néglige le "o" :

$$y(t_{k+1}) = y(t_k) + hy'(t_k) \qquad \text{soit :} \qquad y_{k+1} = y_k + hF(t_k, y_k)$$

Illustrations. Ci-dessous une illustration du type de ligne brisée que l'on souhaite obtenir (à gauche), et le zoom expliquant le passage du k-ème point de la ligne brisée au $(k+1)$-ème point (à droite).





Ce qui est très agréable : c'est que la méthode pour parvenir à nos fins (la méthode d'Euler) est très simple à décrire, et repose sur l'invariable schéma suivant.

```
def F(t,y):
        return ... # A compléter en fonction de l'équa diff

a, b =0, 1 # Bornes de l' intervalle de résolution

N =1000 # Nbre de points

h =(b - a) / N # Définition du pas

y =y0 # Initialisation de y (valeur de y_0, condition initiale )
t =a # Initialisation de t (valeur de a)

ValT =[t] # Initialisation de la liste des abscisses
ValY =[y] # Initialisation de la liste des ordonnées

for k in range(N):
    t =t +h
    y =y +h * F(t,y) # Approx affine: y(t+h) = y(t) + h * y'(t)
    ValT =ValT +[t]
    ValY =ValY +[y]
```

Et pour l'ordre 2 ?

C'est pareil ! Explicitement, on souhaite résoudre numériquement un **problème de Cauchy** de la forme :

$$\textbf{(P)}: \quad \forall\, t \in [0;\ T]\,, \quad \left\{ \begin{array}{l} x'(t) = f\left(t, x(t), y\left(t\right)\right) \\ y'(t) = g\left(t, x(t), y\left(t\right)\right) \end{array} \right. \quad \text{et} \quad \left\{ \begin{array}{l} x\left(0\right) = x_0 \\ y\left(0\right) = y_0 \end{array} \right.$$

où x_0 et y_0 sont deux réels fixés. La seule différence (de taille !) étant qu'il s'agit d'un problème de Cauchy pour deux fonctions inconnues.

L'idée est la même que pour les équations différentielles "simples" ; on cherche à construire une suite de points $M_k\left(x_k, y_k\right)$ approchant la solution du problème. Le changement provient seulement des formules permettant de passer du point M_k au point M_{k+1}.

On utilise cette fois-ci l'approximation :

$$\left(\begin{array}{c} x\left(t+h\right) \\ y\left(t+h\right) \end{array} \right) = \left(\begin{array}{c} x\left(t\right) \\ y\left(t\right) \end{array} \right) + h \left(\begin{array}{c} x'\left(t\right) \\ y'\left(t\right) \end{array} \right)$$

Ces remarques faites, il ne reste plus qu'à adapter la méthode d'Euler à ce cadre.

1) On choisit un réel T, et un entier n.

2) On considère dans l'intervalle $[0;T]$ les $(n+1)$ valeurs t_k données par $t_k = \dfrac{kT}{n}$.

3) On construit les valeurs x_0, x_1,..., x_n et y_0, y_1,..., y_n de la façon suivante :

   ☞ x_0 et y_0 sont données par l'énoncé (conditions initiales) ;

   ☞ pour tout $k \in [0; n-1]$, on pose

$$\left\{ \begin{array}{l} x_{k+1} = x_k + \dfrac{T}{n} f\left(t_k, x_k, y_k\right) \\ \\ y_{k+1} = y_k + \dfrac{T}{n} g\left(t_k, x_k, y_k\right) \end{array} \right.$$

4) Les suites des points (t_k, x_k) et (t_k, y_k) ainsi obtenues permettent d'obtenir deux lignes polygonales approchant les fonctions x et y solutions du problème **(P)**.

5) La suite des points (x_k, y_k) permet de représenter y en fonction de x : c'est ce que l'on appelle le **portrait de phase** de la solution.

```
##################################################################

# METHODE D'EULER D'ORDRE 2 - EQUATION DU PENDULE SIMPLE

##################################################################

# But: résoudre y''(t) = (-g/L)*sin(y(t))
# Dans ce programme, y désignera l'angle, et z = y' la vitesse angulaire

##########################
# PACKAGES
import matplotlib.pyplot as pypl
import numpy as np
from math import *
##########################

##########################
# CONSTANTES, CONDITIONS INITIALES
g =9.81
L =0.5
a, b =0, 10 # Bornes de l'intervalle de résolution
N =1000 # Nbre de points
h =(b - a) / N # Définition du pas

y =pi/2 # Position initiale
z =0 # Vitesse initiale
t =a # Initialisation de t ("origine des temps")

ValT =[t] # Initialisation de la liste des temps
ValY =[y] # Initialisation de la liste des positions angulaires
ValZ =[z] # Initialisation de la liste des vitesses angulaires

##########################
# PROGRAMME PRINCIPAL

for k in range(1,N+1):
    t =t +h
    y, z =y +h * z, z +h * ((-g/L)*sin(y)) # Approx affine: f(t+h) = f(t) + h * f'(t) (avec f=y et f=z)
    ValT =ValT +[t]
    ValY =ValY +[y]
    ValZ =ValZ +[z]

##########################
# REPRESENTATIONS GRAPHIQUES

#pypl.plot(ValT,ValY) # Position angulaire en fonction du tps
#pypl.plot(ValT,ValZ) # Vitesse angulaire en fonction du tps
pypl.plot(ValY,ValZ) # Portrait de phase
pypl.show()
```