# T.P. d'informatique n° 7-4

## Différentes méthodes d'approximation des solutions d'équations différentielles

## Méthodes d'Euler, de Heun, de Runge-Kutta

On cherche à programmer trois méthodes d'approximation des solutions d'une équation différentielle du premier ordre. Pour cela, on s'intéresse au problème de Cauchy

$$\begin{cases} y'(t) = F(t, y(t)) \\ y(0) = y_0 \end{cases}$$

où $F : [0, T] \times \mathbb{R} \to \mathbb{R}$ est une fonction de classe $\mathscr{C}^1$ et $y_0 \in \mathbb{R}$.
Pour l'approximation numérique, on note $h$ le pas de la subdivision uniforme $(t_k)_{0 \leqslant k \leqslant N}$ de l'intervalle $[0, T]$. Ainsi $t_k = k\,h$ et $y_k$ désigne une approximation de $y(t_k)$ pour tout $0 \leqslant k \leqslant N$.

Pour toute la suite, on s'intéresse à l'équation différentielle :

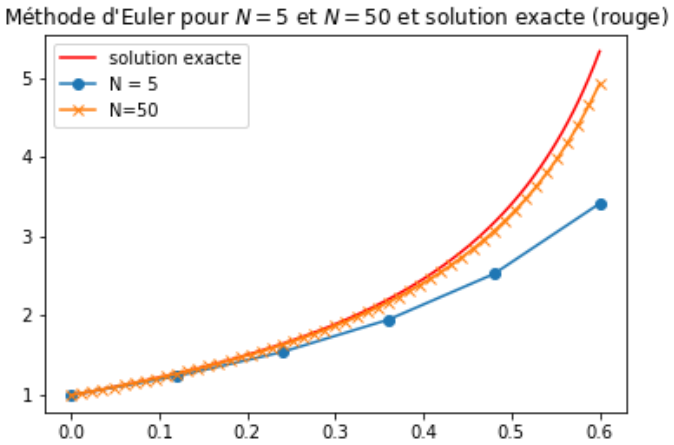
$$y' = 1 + y^2 \qquad (1)$$

avec la condition initiale $y(0) = 1$. On prendra $T = 0.6$. Il est assez remarquable (et peu "fréquent") pour des équations différentielles non linéaires de pouvoir les résoudre explicitement. Mais c'est le cas ici : la solution de ce problème de Cauchy est :

$$y : t \to \tan\left(t + \frac{\pi}{4}\right)$$

Ceci nous permettra de comparer la solution approchée donnée par la méthode d'Euler (ou d'autres méthodes que l'on rencontrera un peu plus tard) avec la solution exacte

1. Définir la fonction $(t, y) \to F(t, y)$ correspondant à l'équation (1).
2. Ecrire une fonction **Euler(F, y0, T, N)** qui renvoie la liste des approximations de la solution aux points $(t_k)_{0 \leqslant k \leqslant N}$ avec $h = \dfrac{T}{N}$.
3. Obtenir le graphique suivant, où la solution exacte sera dessinée en rouge et où figureront les "marques" des points pour les solutions approchées




4. On s'intéresse désormais à deux autres méthodes d'approximation : la méthode de Heun et la méthode de Runge-Kutta (RK4) dont les shémas d'approximation sont donnés ci-dessous.

**Méthode de Heun** $\qquad y_{n+1} = y_n + \dfrac{h}{2}\left[F\left(t_n, y_n\right) + F\left(t_{n+1}, y_n + hF\left(t_n, y_n\right)\right)\right]$

**Méthode de Runge-Kutta (RK4)**

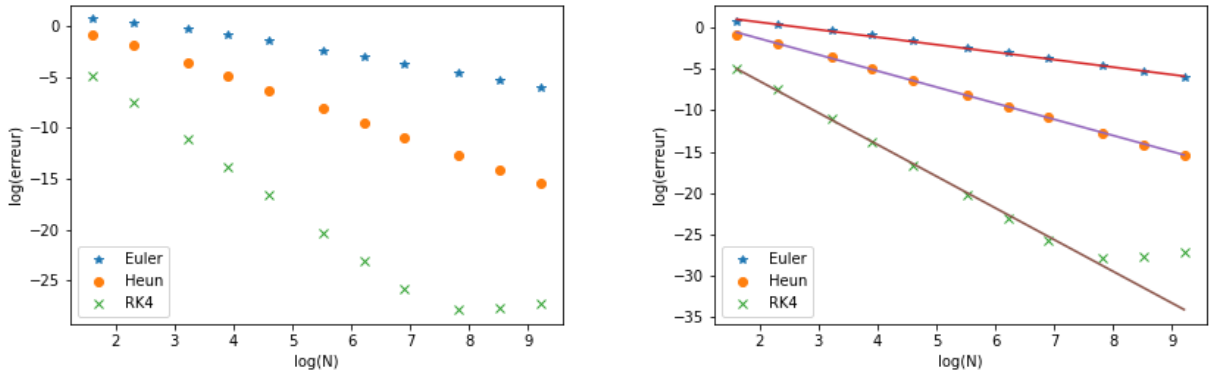
$$\begin{cases} k_1^n & = F\left(t_n, y_n\right) \\ k_2^n & = F\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1^n\right) \\ k_3^n & = F\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2^n\right) \\ k_4^n & = F\left(t_n + h, y_n + hk_3^n\right) \\ y_{n+1} & = y_n + \frac{h}{6}\left(k_1^n + 2k_2^n + 2k_3^n + k_4^n\right) \end{cases}$$

Ecrire deux fonctions **Heun(F, y0, T, N)** et **RK4(F, y0, T, N)** qui renvoient la liste des approximations de la solution aux points $(t_k)_{0\leqslant k\leqslant N}$ avec $h = \dfrac{T}{N}$ par les méthodes de Heun et de Runge-Kutta.

5. Tracer sur un même graphique pour $N = 5$ la solution exacte et les solutions approchées par les trois méthodes proposées : Euler, Heun et RK4
6. **optionnelle** Pour chacune des trois méthodes proposées, une étude théorique de l'erreur permettrait de montrer l'estimation suivante

$$\max_{0\leqslant n\leqslant N} |y_n - y\left(t_n\right)| \leqslant C.h^p.\ \sup_{t\in[0,T]} \left|y^{(p)}(t)\right|$$

où $p$ est un entier appelé ordre de la méthode. Pour la méthode d'Euler, on trouve $p = 1$, pour Heun, $p = 2$ et enfin $p = 4$ pour RK4
On choisit pour $N$ les valeurs 5, 10, 25, 50, 100, 250, 500, 1000, 2500, 5000, 10000.
Obtenir alors les deux graphes suivants




*Remarques :*

- La fonction log désigne ici le logarithme népérien (comme en Python)
- La fonction `polyfit` de `numpy` permet d'effectuer une régression linéaire. Si $x$ et $y$ sont deux listes de flottants (de même taille), la fonction `np.polyfit(x,y,1)` renverra un tableau `numpy [a,b]` de sorte que la fonction polynomiale $P$ de degré inférieur ou égal à 1 définie par $P : x \to ax + b$ minimise le carré de l'erreur, c'est-à-dire $\sum_j |P\left(x_j\right) - y_j|^2$. Cette fonction a permis de tracer les droites du second graphique
- On remarquera que pour la méthode RK4, les deux derniers points sont abérrants (l'erreur étant proche de la précision machine) : on n'en a pas tenu compte dans le calcul de la régression linéaire

7. Constater que, expérimentalement sur notre exemple, les méthodes d'Euler, de Heun et de RK4 correspondent bien à des méthodes d'ordre respectif 1, 2 et 4.

## Le modèle proie-prédateur de Volterra-Lotka

Le modèle formel de Lotka fut repris en 1931 par Volterra pour expliquer les évolutions des populations de sardines et de requins en mer Adriatique. Si $a, b, c, d$ sont quatre réels strictement positifs, on appelle système proie-prédateur de Volterra-Lotka, le système différentiel

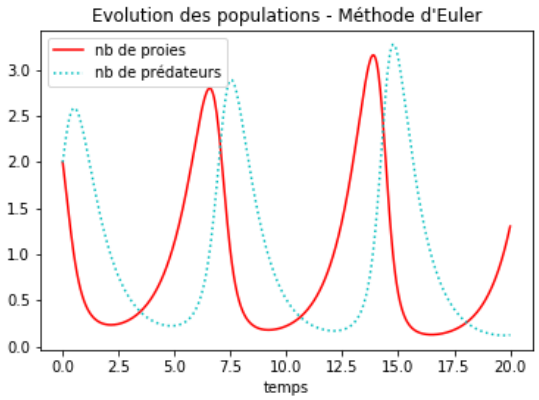
$$\begin{cases} x' & = (a - cy)x \\ y' & = (-c + dx)y \end{cases}$$

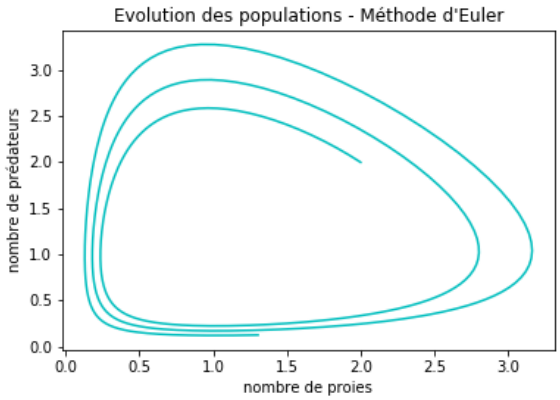
La fonction $x$ représente le nombre de proies (par exemple les sardines) et la fonction $y$ représente le nombre de prédateurs (par ex. les requins). La constante $a$ représente le taux de natalité des proies, $b$ le taux de mortalité

des proies dû aux prédateurs rencontrés, $c$ le taux de mortalité des prédateurs, et $d$ le taux de reproduction des prédateurs en fonction des proies rencontrées et mangées.
Pour la suite, on prendra arbitrairement $a = b = c = d = 1$. On cherche des approximations de $x, y$ sur un intervalle de temps $[0, T]$ par la méthode d'Euler et par la méthode RK4 pour les conditions initiales

$$\begin{cases} x(0) = x_0 \\ y(0) = y_0 \end{cases}$$

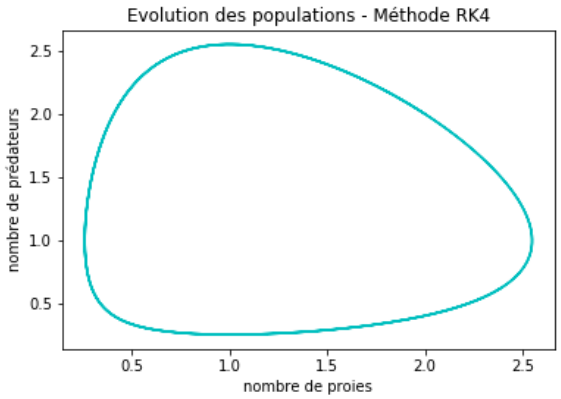
1. En posant $X(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$, transformer le système différentiel en une équation différentielle de la forme : $X' = F(X)$ où $F$ est une fonction de $\mathbb{R}^2$ vers $\mathbb{R}^2$ que l'on explicitera.
   Définir en Python cette $F$ (on utilisera des tableaux numpy pour les couples)
2. Ecrire une fonction **Euler(F, X0, T, N)** où X0 est un tableau numpy contenant les conditions initiales $(x_0, y_0)$. Cette fonction renverra une liste de tableaux numpy contenant les approximations de $x$ et $y$ aux temps $(t_k)_{0 \leqslant k \leqslant N}$.
3. Pour $(x_0, y_0) = (2, 2)$, $T = 20$ et $N = 500$ obtenir les tracés suivants :




4. En fait, une étude théorique montrerait que les fonctions $x$ et $y$ sont périodiques (de même période). La figure précédente montre que l'approximation des solutions par la méthode d4euler ne fait pas apparaitre ce caractère périodique.

   On va désormais chercher à obtenir une approximation de $x$ et $y$ par la méthode RK4.
   Ecrire une fonction **RK4(F, X0, T, N)** où X0 est un tableau numpy contenant les conditions initiales $(x_0, y_0)$. Cette fonction renverra une liste de tableaux numpy contenant les approximations de $x$ et $y$ aux temps $(t_k)_{0 \leqslant k \leqslant N}$.
   En reprenant les valeurs de $(x_0, y_0)$, $T$ et $N$, obtenir le tracé ci-contre.



**Solution**

```
1
2
3
# -*- coding: utf-8 -*-
"""
Created on Fri Feb  9 05:56:17 2018

@author: FranÃ§ois
"""
import numpy as np
import matplotlib.pyplot as plt
from os import chdir

chdir('D:/Latex/exoperso/feuilleMP20172018')

def Euler(F, y0, T, N):
    h = T/N
    t, y, Liste_t, Liste_y = 0, y0, [0], [y0]
    for k in range(N):
        y = y + h * F(t,y)
        Liste_y.append(y)
        t += h
        Liste_t.append(t)
    return(Liste_t,Liste_y)


F = lambda t,y : 1+y**2

solexact = lambda t:np.tan(t+np.pi/4)

T = 0.6
y0 = 1
x = np.linspace(0,T,100)
y = solexact(x)
plt.plot(x,y,'r')

abscisses, ordonnees = Euler(F,1.,0.6,5)
plt.plot(abscisses,ordonnees,marker = 'o', linestyle='-')

abscisses, ordonnees = Euler(F,1.,0.6,50)
plt.plot(abscisses,ordonnees,marker = 'x',linestyle='-')

plt.legend(['solution exacte', 'N = 5', 'N=50'])
plt.title("MÃ©thode d'Euler pour $N=5$ et $N = 50$ et solution exacte (rouge)")
plt.savefig('Euler1.png')
plt.show()


def Euler(F, y0, T, N):
    h = T/N
    t, y, Liste_t, Liste_y = 0, y0, [0], [y0]
    for k in range(N):
        y = y + h * F(t,y)
        Liste_y.append(y)
        t += h
        Liste_t.append(t)
    return(Liste_t,Liste_y)



def Heun(F, y0, T, N):
    h = T/N
```

```
    t, y, Liste_t, Liste_y = 0, y0, [0], [y0]
    for k in range(N):
        y = y + (h/2) * ( F(t,y) + F(t+h, y + h*F(t,y)))
        Liste_y.append(y)
        t += h
        Liste_t.append(t)
    return(Liste_t,Liste_y)


F = lambda t,y : 1+y**2
solexact = lambda t:np.tan(t+np.pi/4)
T = 0.6
y0 = 1
x = np.linspace(0,T,100)
y = solexact(x)
plt.plot(x,y,'r')

abscisses, ordonnees = Euler(F,1.,0.6,5)
plt.plot(abscisses,ordonnees,marker = 'o', linestyle='-')

abscisses, ordonnees = Heun(F,1.,0.6,5)
plt.plot(abscisses,ordonnees,marker = 'x',linestyle='-')


plt.legend(['solution exacte', 'Euler', 'Heun'])
plt.title("MÃ©thodes d'Euler et Heun pour $N=5$ et solution exacte (en rouge)")
plt.savefig('EulerHeun.png')
plt.show()









def Euler(F, y0, T, N):
    h = T/N
    t, y, Liste_t, Liste_y = 0, y0, [0], [y0]
    for k in range(N):
        y = y + h * F(t,y)
        Liste_y.append(y)
        t += h
        Liste_t.append(t)
    return(Liste_t,Liste_y)



def Heun(F, y0, T, N):
    h = T/N
    t, y, Liste_t, Liste_y = 0, y0, [0], [y0]
    for k in range(N):
        y = y + (h/2) * ( F(t,y) + F(t+h, y + h*F(t,y)))
        Liste_y.append(y)
        t += h
        Liste_t.append(t)
    return(Liste_t,Liste_y)



def RK4(F, y0, T, N):
    h = T/N
    t, y, Liste_t, Liste_y = 0, y0, [0], [y0]
    for k in range(N):
        k1 = F(t,y)
        k2 = F(t+h/2, y + h*k1/2)
```

```
        k3 = F(t+h/2, y + h*k2/2)
        k4 = F(t+h, y + h*k3)
        y = y + (h/6) * ( k1 + 2*k2 + 2*k3 + k4)
        Liste_y.append(y)
        t += h
        Liste_t.append(t)
    return(Liste_t,Liste_y)
```

```
F = lambda t,y : 1+y**2
solexact = lambda t:np.tan(t+np.pi/4)
T = 0.6
y0 = 1
x = np.linspace(0,T,100)
y = solexact(x)
plt.plot(x,y,'r')

abscisses, ordonnees = Euler(F,1.,0.6,5)
plt.plot(abscisses,ordonnees,marker = 'o', linestyle='-')

abscisses, ordonnees = Heun(F,1.,0.6,5)
plt.plot(abscisses,ordonnees,marker = 'x',linestyle='-')

abscisses, ordonnees = RK4(F,1.,0.6,5)
plt.plot(abscisses,ordonnees,marker = '+',linestyle='-')

plt.legend(['solution exacte', 'Euler', 'Heun', 'RK4'])
plt.title("MÃ©thodes d'Euler, Heun et RK4 pour $N=5$ et solution exacte ")
plt.savefig('EulerHeunRK4.png')
plt.show()

def maxliste(L):
    res = -1 * float("inf")
    for x in L:
        if x>res:
            res = x
    return(res)
def erreurmax(methode,N):
    abscis, ordon = methode(F,1.,0.6,N)
    ordonexactes = [solexact(t) for t in abscis]
    erreursinstantannees = [abs(ordon[k]-ordonexactes[k]) for k in range(N+1)]
    return(maxliste(erreursinstantannees))


valtest = [5, 10, 25, 50, 100, 250, 500, 1000, 2500, 5000, 10000]

erreurEuler = [erreurmax(Euler,N) for N in valtest]
erreurHeun = [erreurmax(Heun,N) for N in valtest]
erreurRK4 = [erreurmax(RK4,N) for N in valtest]

abscisses = np.log(valtest)
ordonnees = np.log(erreurEuler)
plt.plot(abscisses,ordonnees,'*')
ordonnees = np.log(erreurHeun)
plt.plot(abscisses,ordonnees,'o')
ordonnees = np.log(erreurRK4)
plt.plot(abscisses,ordonnees,'x')
plt.xlabel('log(N)')
plt.ylabel('log(erreur)')
plt.legend(['Euler', 'Heun', 'RK4'])
plt.savefig('EulerHeunRK4Erreur.png')
```

```
plt.show()


abscisses = np.log(valtest)
ordonneesE = np.log(erreurEuler)
plt.plot(abscisses,ordonneesE,'*')

ordonneesH = np.log(erreurHeun)
plt.plot(abscisses,ordonneesH,'o')
ordonneesR = np.log(erreurRK4)
plt.plot(abscisses,ordonneesR,'x')
aE,b = np.polyfit(abscisses,ordonneesE,1)
reg = aE*abscisses + b
plt.plot(abscisses,reg)
aH,b = np.polyfit(abscisses,ordonneesH,1)
reg = aH*abscisses + b
plt.plot(abscisses,reg)
aR,b = np.polyfit(abscisses[:-2],ordonneesR[:-2],1)
reg = aR*abscisses + b
plt.plot(abscisses,reg)

plt.xlabel('log(N)')
plt.ylabel('log(erreur)')
plt.legend(['Euler', 'Heun', 'RK4'])
plt.savefig('EulerHeunRK4regressionlineaire.png')
plt.show()

print(aE, aH, aR)

a, b, c, d = 1,1,1,1
F = lambda t,Y: np.array([Y[0]*(a - b * Y[1]),
                          -Y[1]*(c - d * Y[0])])

T, N = 20, 500

resEuler = Euler(F,np.array([2.,2.]),T, N)
abscisses = resEuler[0]
n = len(abscisses)
ordonneesProie = [resEuler[1][k][0] for k in range(n)]
ordonneesPreda = [resEuler[1][k][1] for k in range(n)]
plt.plot(abscisses,ordonneesProie, 'r-')
plt.plot(abscisses,ordonneesPreda, 'c:')
plt.xlabel('temps')
plt.title("Evolution des populations - MÃ©thode d'Euler")
plt.legend(['nb de proies', 'nb de prÃ©dateurs'])
plt.savefig("predateurproieEuler.png")
plt.show()

plt.plot(ordonneesProie,ordonneesPreda, 'c-')
plt.xlabel('nombre de proies')
plt.ylabel('nombre de prÃ©dateurs')
plt.title("Evolution des populations - MÃ©thode d'Euler")
plt.savefig("predateurproieportraitdephaseEuler.png")
plt.show()



T, N = 20, 500
resRK4 = RK4(F,np.array([2.,2.]),T, N)
abscisses = resEuler[0]
n = len(abscisses)
ordonneesProie = [resRK4[1][k][0] for k in range(n)]
ordonneesPreda = [resRK4[1][k][1] for k in range(n)]
plt.plot(abscisses,ordonneesProie, 'r-')
plt.plot(abscisses,ordonneesPreda, 'c:')
```

```
plt.xlabel('temps')
plt.title("Evolution des populations - MÃ©thode RK4")
plt.legend(['nb de proies', 'nb de prÃ©dateurs'])
plt.savefig("predateurproieRK4.png")
plt.show()

plt.plot(ordonneesProie,ordonneesPreda, 'c-')
plt.xlabel('nombre de proies')
plt.ylabel('nombre de prÃ©dateurs')
plt.title("Evolution des populations - MÃ©thode RK4")
plt.savefig("predateurproieportraitdephaseRK4.png")
plt.show()
```