Informatique en CPGE (2018-2019)
TD 13 : équations différentielles

Exercice 1 Corrigé

1.

```
def integRec(f,a,b,h):
    x,s=a,0
    while x<b:
        s+=h*f(x)
        x+=h
    return s

def f(x):
    return 1/x

print(integRec(f,1,2,0.01))
```

La valeur obtenue est 0.695653430481824.

2.

```
def euler(f,a,b,y0,h):
    x,y=a,y0
    liste_x=[x]
    liste_y=[y]
    while x<b:
        y+=h*f(x)
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return liste_x,liste_y

a,b=euler(f,1,2,0,0.01)
print(b[-1])
```

La valeur obtenue est encore 0.695653430481824. C'est normal puisque les deux méthodes sont "identiques".

La valeur exacte est $\ln(2) \simeq 0.6931471805599453$.

3.

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

plt.plot(a,b,"*") # solution approchée

a=np.linspace(1,2,101)
b=np.log(a)
plt.plot(a,b) # solution exacte

plt.show()
```

Exercice 2 Corrigé

1.

```
def integTrap(f,a,b,h):
    s=f(a)+f(b)
    x=a+h
    while x<b:
        s+=2*f(x)
        x+=h
    return s*h/2

from math import exp,pi
def f(x):
    return exp(-x**2)

I=integTrap(f,0,8,0.01)
```

Nous obtenons la valeur approchée $I \simeq 0.8862269254527576$.

2.

```
def euler(f,a,b,y0,h):
    x,y=a,y0
    liste_x=[x]
    liste_y=[y]
    while x<b:
        y+=h*f(x+h/2)
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return liste_x,liste_y

a,b=euler(f,0,8,0,0.01)
print(b[-1],I,pi**0.5/2)
```

Nous obtenons $y(2) \simeq 0.886226925452757$.

Le nombre $\sqrt{\pi}/2$ est une valeur approchée de $\displaystyle\int_0^8 \mathrm{e}^{-t^2}\mathrm{d}t$, soit environ 0.8862269254527579.

Note : avec la méthode d'Euler classique, (exercice 1), la valeur obtenue est légèrement supérieure ($\simeq 0.891226925452757$).

3.

```
import matplotlib.pyplot as plt
plt.plot(a,b)
plt.show()
```

La courbe se rapproche très vite de son asymptote d'équation $y = 1$.

Exercice 3 Corrigé

```
def taylor(f,df,a,b,y0,h):
    x,y=a,y0
    liste_x=[x]
    liste_y=[y]
    while x<b:
        y+=h*f(x)+0.5*h**2*df(x)
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return liste_x,liste_y

def f(x):
    return 1/x

def df(x):
    return -1/(x**2)

a1,b1=taylor(f,df,1,10,0,0.1)
a2,b2=euler(f,1,10,0,0.1) # voir exercice 1

import matplotlib.pyplot as plt
import numpy as np

lx=np.linspace(1,10,91)
ly=np.log(lx)
plt.plot(a1,b1)
plt.plot(a2,b2)
plt.plot(lx,ly)
plt.show()
```

Nous pouvons constater sur la figure obtenue que la méthode de Taylor est plus performante que la méthode d'Euler.

Exercice 4 Corrigé

1. Schéma : $y_{i+1} = y_i + hay_i(1 - y_i/k) = y_i + hf(y_i)$

```
a=0.15
p0=20 # population initiale

def k1(x):
    return 1000

def k2(x):
    return 700 if x<60 else 300

def k3(x):
    r=x%40
    if r<10: return 200
    else: return 100

def f1(y,x):
    return a*y*(1-y/k1(x))

def f2(y,x):
```

```
    return a*y*(1-y/k2(x))

def f3(y,x):
    return a*y*(1-y/k3(x))

def euler(f,a,b,y0,h):
    x,y=a,y0
    liste_x=[x]
    liste_y=[y]
    while x<b:
        y+=h*f(y,x)
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return liste_x,liste_y

import matplotlib.pyplot as plt

lx,ly=euler(f1,0,200,p0,0.01)
plt.plot(lx,ly)

lx,ly=euler(f2,0,200,p0,0.01)
plt.plot(lx,ly)

lx,ly=euler(f3,0,200,p0,0.01)
plt.plot(lx,ly)

plt.show()
```

2. Méthode de Heun

Le schéma est le suivant :

$$x_{k+1} = x_k + h$$
$$v = y_k + hf(x_k, y_k)$$
$$y_{k+1} = y_k + \frac{1}{2}hf(x_k, y_k) + \frac{1}{2}hf(x_{k+1}, v)$$

```
def heun(f,a,b,y0,h):
    x,y=a,y0
    liste_x=[x]
    liste_y=[y]
    while x<b:
        fx=f(y,x) # avec x au rang k
        v=y+h*fx
        x+=h
        liste_x.append(x)
        y+=0.5*h*fx+0.5*h*f(v,x) # avec x au rang k+1
        liste_y.append(y)
    return liste_x,liste_y
```

Il est nécessaire de calculer $f(y, x)$ avant l'incrémentation de x. Ensuite, pour le calcul de y, $f(v, x)$ est calculé avec la valeur de x incrémentée.

Exercice 5 Corrigé

1.

```
def euler(f,a,b,y0,h):
    x,y=a,y0
    liste_x=[x]
    liste_y=[y]
    while x<b:
        y+=h*f(y,x)
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return liste_x,liste_y

# s'=(1+f'(x)^2)^0.5 si y=f(x) avec s0=0
# parabole
def f(y,x):
    return (1+4*x**2)**0.5
s0=0
lx,ly=euler(f,0,1,s0,0.001)
print(ly[-1])
# la valeur exacte est (5**0.5)/2+log(2+5**0.5)/4
```

Nous obtenons la valeur 1.4783249726270449, soit 1,48 à 10^{-2} près, ce qui correspond bien avec la valeur exacte.

2.

```
# s'=(x'(t)^2+y'(t)^2 )^0.5 si x(t) et y(t)
from math import sin,cos,pi

# ellipse x=acos(t), y=bsin(t)
a=2
b=3

def f(y,x):
    return ((-a*sin(x))**2+(b*cos(x))**2)**0.5

s0=0
lx,ly=euler(f,0,2*pi,s0,0.001)
print(ly[-1],pi*(3*(a+b)-((3*a+b)*(a+3*b))**0.5))
```

Une valeur approchée du périmètre de l'ellipse est $\pi\left(3(a+b)-\sqrt{(3a+b)(a+3b)}\right)$. Si $a=b$, nous retrouvons le périmètre du cercle de rayon a.

3.

```
# cycloide
# x(t)=r(t-sin(t))
# y(t)=r(1-cos(t))
r=4
def f(y,x):
    return ((r*(1-cos(x)))**2+(r*sin(x))**2)**0.5
```

```
s0=0
lx,ly=euler(f,0,2*pi,s0,0.01)
print(ly[-1],8*r)
```

La valeur exacte de la longueur de la cycloïde est $8r$.

Exercice 6 Corrigé

1.

```
def c(x):
    return 1/(1+x**2)
```

2.

```
import numpy as np

def f(y,x): # y est un couple
    return np.array([y[1],c(x)-0.5*y[1]-8*np.sin(y[0])])
```

3.

```
def newton(h,dh,a):
    x=a
    for i in range(10):
        x=x-h(x)/dh(x)
    return x
```

4.

```
def backEuler(a,b,y0,n,f):
    h=(b-a)/n
    x=np.zeros(n+1)
    y=np.zeros((n+1,2))
    x[0]=0
    y[0]=y0
    for i in range(n):
        def F(u):
            return u-h*f(u,x[i+1])-y[i]
        def dF(t):
            e=10**(-5)
            return (F(t+e)-F(t-e))/(2*e)
        x[i+1]=x[i]+h
        st=y[i]+h*f(y[i],x[i]) # euler directe
        y[i+1]=newton(F,dF,st)
    return x,y
```

5.

```
import matplotlib.pyplot as plt
from scipy.integrate import odeint

u0,v0=1,2
y0=np.array([u0,v0])
x,y=backEuler(0,10,y0,1000,f)
u=[v[0] for v in y]
# ou u=y[:,0]
plt.plot(x,u,"r") # en rouge

# avec odeint
x=np.linspace(0,10,201)
sol=odeint(f,y0,x)
y=sol[:,0]
plt.plot(x,y)
plt.show()
```

Avec $n = 1000$ intervalles, la précision n'est pas très bonne. Elle devient corecte à partir de $n = 4000$ intervalles mais le temps de calcul est allongé.

Exercice 7 Corrigé

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

g=9.81
k=0.2
v0=6
a0=np.pi/4

def f1(u,t): # sans résistance de l'air
    return np.array([u[1],0,u[3],-g])

def f2(u,t): # avec résistance de l'air
    return np.array([u[1],-k*u[1],u[3],-k*u[3]-g])

def euler(a,b,u0,h,f):
    n=int(round((b-a)/h))
    t=np.zeros(n+1)
    u=np.zeros((n+1,4))
    t[0]=0
    u[0]=u0
    lim=n+1
    for i in range(n):
        t[i+1]=t[i]+h
        u[i+1]=u[i]+h*f(u[i],t[i])
        if u[i+1][2]<0:
            lim=i+1
            break
    return t,u[:lim]

vx0,vy0=v0*np.cos(a0),v0*np.sin(a0)
y0=np.array([0,vx0,0,vy0])
x,y=euler(0,1.2,y0,0.01,f1)

lx=y[:,0]
ly=y[:,2]
```

```
plt.plot(lx,ly)

x,y=euler(0,1.2,y0,0.01,f2)
lx=y[:,0]
ly=y[:,2]
plt.plot(lx,ly)
plt.show()
```

Le tracé des courbes montre bien la différence de trajectoire obtenue dans chacun des deux cas.

Exercice 8 Corrigé

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

L=0.5
g=9.81
a0,v0=np.pi/3,0

def f(y,x): # y est un couple
    return np.array([y[1],-(g/L)*np.sin(y[0])])

def RK2(a,b,y0,n,f):
    h=(b-a)/n
    x=np.zeros(n+1)
    y=np.zeros((n+1,2))
    x[0]=0
    y[0]=y0
    for i in range(n):
        k1=h*f(y[i],x[i])
        k2=h*f(y[i]+k1/2,x[i]+h/2)
        x[i+1]=x[i]+h
        y[i+1]=y[i]+k2
    return x,y

u0=np.array([a0,v0])
t,u=RK2(0,10,u0,1000,f)
alpha=u[:,0]
plt.plot(t,alpha)

# avec odeint
t=np.linspace(0,10,101)
sol=odeint(f,u0,t)
alpha=sol[:,0]
plt.plot(t,alpha)
plt.show()

v=u[:,1]
plt.plot(alpha,v)
plt.show()
```

Pour le diagramme de phase, nous obtenons un cercle.

Exercice 9 Corrigé

1. Méthode RK4.

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

def f(y,x):
    return np.array([y[1],-0.5*y[1]-10*np.sin(y[0])])

def RK4(a,b,y0,n,f):
    h=(b-a)/n
    x=np.zeros(n+1)
    y=np.zeros((n+1,2))
    x[0]=0
    y[0]=y0
    for i in range(n):
        k1=h*f(y[i],x[i])
        k2=h*f(y[i]+k1/2,x[i]+h/2)
        k3=h*f(y[i]+k2/2,x[i]+h/2)
        k4=h*f(y[i]+k3,x[i]+h)
        x[i+1]=x[i]+h
        y[i+1]=y[i]+(k1+2*k2+2*k3+k4)/6
    return x,y
```

2.

```
a0,v0=np.pi/3,0
u0=np.array([a0,v0])
t,u=RK4(0,20,u0,500,f)
alpha=u[:,0]
plt.plot(t,alpha)
plt.show()

v=u[:,1]
plt.plot(alpha,v)
plt.show()
```

Pour le diagramme de phase, nous obtenons une spirale.

3. L'équation différentielle du mouvement se traduit simplement par les deux équations différentielles :

$$y' = z \text{ et } z' = -0,5z - 10\sin(y)$$

4.

```
def f1(y,z):
    return z
def f2(y,z):
    return -0.5*z-10*np.sin(y)
```

5.

```
def euler(a,b,u0,n,f,g):
    h=(b-a)/n
```

```
    x=np.zeros(n+1)
    y=np.zeros(n+1)
    z=np.zeros(n+1)
    x[0]=0
    y[0]=u0[0]
    z[0]=u0[1]
    for i in range(n):
        x[i+1]=x[i]+h
        y[i+1]=y[i]+h*f(y[i],z[i])
        z[i+1]=z[i]+h*g(y[i],z[i])
    return x,y,z

u0=np.array([a0,v0])
t,alpha,v=euler(0,20,u0,20000,f1,f2)
plt.plot(t,alpha)
plt.show()

plt.plot(alpha,v)
plt.show()
```

Nous pouvons constater sur le diagramme de phase qu'il faut un pas h=0.001, soit 20000 intervalles, pour que la précision soit comparable à celle obtenue avec la méthode RK4.

Exercice 10 Corrigé

```
def propage(s0,i0,r,a,n):
    s=s0
    i=i0
    liste_t=list(range(n+1))
    liste_s=[s]
    liste_i=[i]
    for k in range(n):
        s_temp=s-r*s*i
        i=i+r*s*i-a*i
        s=s_temp
        liste_s.append(s)
        liste_i.append(i)
    return liste_t,liste_s,liste_i

t,s,i=propage(500,1,2.5*10**(-3),0.32,30)

import matplotlib.pyplot as plt
plt.plot(t,s)
plt.plot(t,i)
plt.show()
```

Les individus qui ne sont plus comptabilisés parmi les "sains" ou les "infectés" sont soit rétablis et ne peuvent plus être contaminés, soit décédés.

Exercice 11 Corrigé

1.

```
a,b,c,d=0.5,0.6,0.5,0.4

def LV(x0,y0,n,dt):
```

```
    x=x0
    y=y0
    liste_t=list(range(n+1))
    liste_x=[x]
    liste_y=[y]
    for k in range(n):
        x_temp=x+dt*(a*x-b*x*y)
        y=y+dt*(-c*y+d*x*y)
        x=x_temp
        liste_x.append(x)
        liste_y.append(y)
    return liste_t,liste_x,liste_y
```

2.

```
import matplotlib.pyplot as plt

t,x,y=LV(0.85,1.25,50000,0.001)
plt.plot(x,y)
plt.show()
```

Avec $n = 500$ et $dt = 0.1$, ou avec $n = 5000$ et $dt = 0.01$, la courbe s'enroule sur elle-même.

3.

```
import numpy as np

def f(u): # u est un couple
    dx=a*u[0]-b*u[0]*u[1]
    dy=-c*u[1]+d*u[0]*u[1]
    return np.array([dx,dy])

def RK4(u0,n,dt,f):
    t=np.zeros(n+1)
    u=np.zeros((n+1,2))
    t[0]=0
    u[0]=u0
    for i in range(n):
        k1=dt*f(u[i])
        k2=dt*f(u[i]+k1/2)
        k3=dt*f(u[i]+k2/2)
        k4=dt*f(u[i]+k3)
        t[i+1]=t[i]+dt
        u[i+1]=u[i]+(k1+2*k2+2*k3+k4)/6
    return t,u

z0=np.array([0.85,1.25])
t,z=RK4(z0,500,0.1,f)
plt.plot(z[:,0],z[:,1])
plt.show()
```

4.

```
for i in range(20):
    z0=np.array([0.85+i/10,1.25+i/10])
    t,z=LVRK4(z0,200,0.1,f)
    plt.plot(z[:,0],z[:,1])
plt.show()
```

5.

```
z0=np.array([0.85,1.25])
t,z=LVRK4(z0,500,0.1,f)
plt.plot(t,z[:,0])
plt.plot(t,z[:,1])
plt.show()
```

6.

```
def f(x,y):
    return y*(-c+d*x)/x/(a-b*y)

def LV2RK4(x0,y0,n,dx,f):
    x=np.zeros(n+1)
    y=np.zeros(n+1)
    x[0]=x0
    y[0]=y0
    for i in range(n):
        k1=dx*f(x[i],y[i])
        k2=dx*f(x[i]+dx/2,y[i]+k1/2)
        k3=dx*f(x[i]+dx/2,y[i]+k2/2)
        k4=dx*f(x[i]+dx,y[i]+k3)
        x[i+1]=x[i]+dx
        y[i+1]=y[i]+(k1+2*k2+2*k3+k4)/6
    return x,y

x,y=LV2RK4(0.85,1.25,300,0.01,f)
plt.plot(x,y)
plt.show()
```