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

II.1. Suites récurrentes

1.

```
def A(n):
    res = 2
    i = 0
    while i<n:
        res = 1-np.exp(res)
        i += 1
    return res
```

2.

```
def lstA(n):
    res = [2]
    i = 0
    while i<n:
        res.append(1-np.exp(res[-1]))
        i += 1
    return res
```

3.

```
N = 100
lst = lstA(N)
plt.plot(range(N+1),lst,'+')
```

4.

```
def BC(n):
    B = 0
    C = 1
    i = 0
    while i<n:
        D = -B + C/2
        C = B - C
        B = D
        i += 1
    return (B,C)
```

5.

```
def lstBC(n):
    res = [(0,1)]
    B = 0
    C = 1
```

```
    i = 0
    while i<n:
        D = -B + C/2
        C = B - C
        B = D
        res.append((B,C))
        i += 1
    return res
```

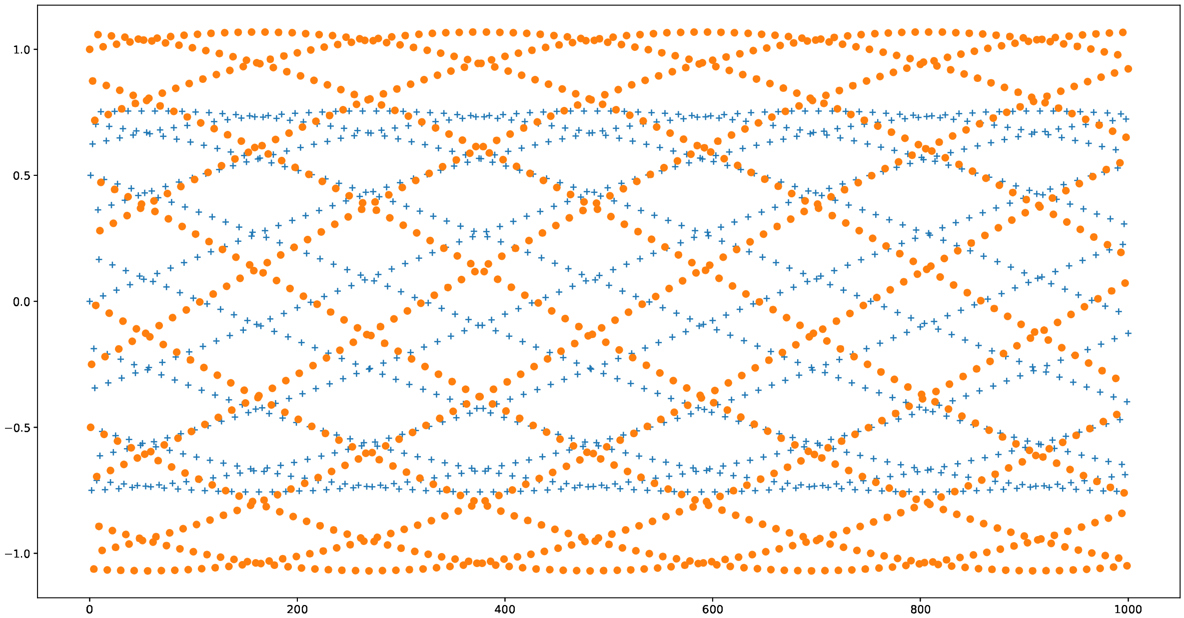
6.

```
plt.figure()
N = 10
lst = lstBC(N)
plt.plot(range(N+1),[k[0] for k in lst],'+')
plt.plot(range(N+1),[k[1] for k in lst],'o')
```

7.

```
def lstBC(n):
    res = [(0,1)]
    B = 0
    C = 1
    i = 0
    while i<n:
        B = -B + C/2
        C = B - C
        res.append((B,C))
        i += 1
    return res

plt.figure()
N = 1000
lst = lstBC(N)
plt.plot(range(N+1),[k[0] for k in lst],'+')
plt.plot(range(N+1),[k[1] for k in lst],'o')
```



II.2. Méthode d'Euler et systèmes chaotiques

1.

```
def euler(y0,T,dt=1e-2):
    y = y0
    t = 0
    res = [y0]
    while t<T:
        y = y + dt*(y**2+y-np.cos(y))
        res.append(y)
        t += dt
    return res
```

2.

```
plt.figure()
Y = euler(0.55000934993,10,dt=1e-3)
T = np.linspace(0,10,len(Y))
plt.plot(T,Y)
```

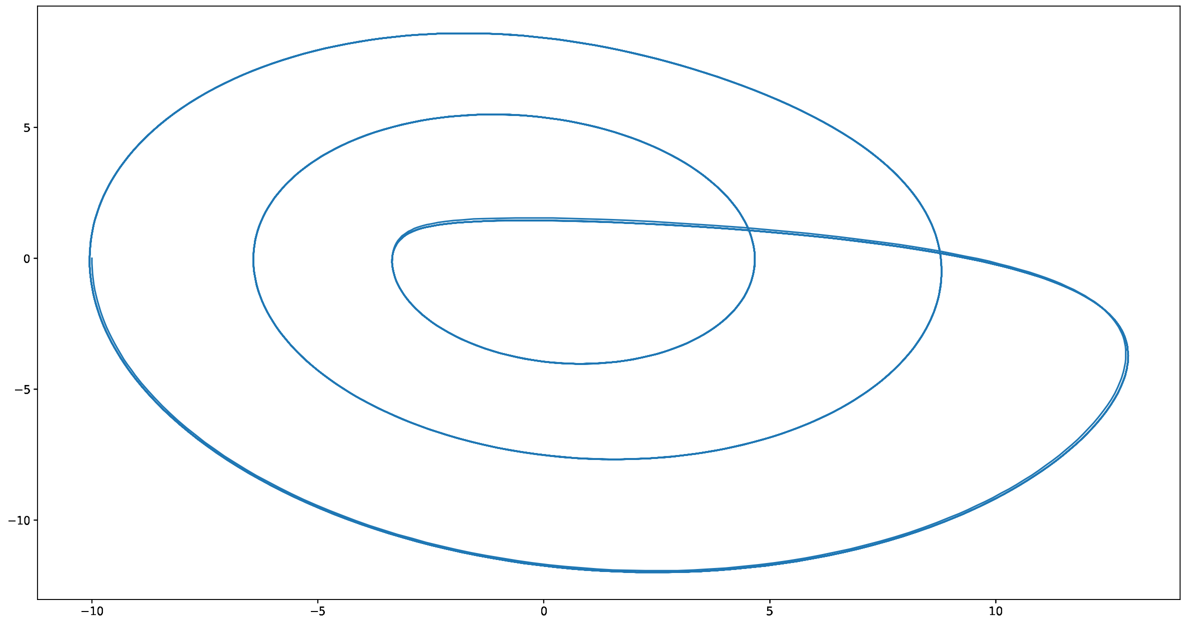
3.

```
def eulerXYZ(V0,T,dt=1e-2):
    x,y,z = V0
    t = 0
    res = [V0]
    while t<T:
        xx = x + dt*(-y-z)
        yy = y + dt*(x+0.2*y)
```

```
        z  = z + dt*(0.1+z*(x-5.7))
        x = xx
        y = yy
        res.append((x,y,z))
        t += dt
    return res
```

4.

```
plt.figure()
Sol = eulerXYZ((-10,0,0),100,dt=1e-2)
X = [k[0] for k in Sol]
Y = [k[1] for k in Sol]
plt.plot(X,Y)
```



II.3. Méthode de Newton

1. Attention à bien vérifier que $u_n \neq 0$ avant de calculer u_{n+1}.

```
def suite(u):
    n=0
    v=0
    while n<=10 and u!=0 and abs(v-u)>1e-7:
        v=u
        u=(2*u+1/u**2)/3
        n+=1
    return u,n
```

2. On utilise la fonction précédente en faisant varier v_0 sur les pixels du pavé considéré :

```
Taille=200
image=np.ndarray((Taille,Taille,3),np.uint8)
```

```
for i in range(Taille):
    for j in range(Taille):
        x=-1+2*j/(Taille-1)
        y=1-2*i/(Taille-1)
        z=x+y*1.j
        image[i,j]=[0,0,0]
        u,n=suite(z)
        if np.abs(u-1)<=0.01:
            image[i,j,0]=255-5*n
        if np.abs(u-(-0.5+0.8660254037844387j))<=0.01:
            image[i,j,1]=255-5*n
        if np.abs(u-(-0.5-0.8660254037844387j))<=0.01:
            image[i,j,2]=255-5*n

plt.imshow(image)
```

3. On modifie la suite de Newton puis on recommence la représentation graphique :

```
nbiter=50

def suite(u):
    n=0
    v=0
    while n<=nbiter and abs(v-u)>1e-7:
        v=u
        u=(3*u**3+u**2+u+1)/(4*u**2+u+1)
        n+=1
    return u,n

Taille=1000
image=np.ndarray((Taille,Taille,3),np.uint8)
xmin=-1.
xmax=1.
ymin=-1.
ymax=1.
for i in range(Taille):
    for j in range(Taille):
        x=xmin+(xmax-xmin)*j/(Taille-1)
        y=ymax-(ymax-ymin)*i/(Taille-1)
        z=x+y*1.j
        image[i,j]=[0,0,0]
        u,n=suite(z)
        if np.abs(u-1)<=0.01:
            image[i,j,0]=255-4*n
        if np.abs(u-(-0.5+0.8660254037844387j))<=0.01:
            image[i,j,1]=255-4*n
        if np.abs(u-(-0.5-0.8660254037844387j))<=0.01:
            image[i,j,2]=255-4*n

plt.imshow(image)
```

Ce qui donne (pour une image de 1000×1000 pixels et les deux dernières questions) :