```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

Ms=2e30   #masse soleil
G=6.67e-11  #constante gravitation

#données planète
e=0.917  #excentricité
coeff=np.sqrt((1+e)/(1-e))  #coefficient devant angle vrai
q=0.98 #distance périhélie soleil
a=q/(1-e) #demi grand axe UA
c=a*e
b=np.sqrt(a**2-c**2)
a1=a*1.5e11  #demi grand axe SI
T=2*np.pi*np.sqrt(a1**3/(G*Ms))  #période SI
n=np.sqrt(G*Ms/(a1**3)) #moyen mouvement
titre='Trajectoire planète autour du soleil, selon équation de Kepler'
espac=400  #espacement points tableau des temps
t=np.linspace(0,T,espac)  #temps

E=np.linspace(0,2*np.pi,100)  #tableau recherche anomalie excentrique
sol=np.empty(espac)   #tableau solution anomalie excentrique
vrai=np.empty(espac)   #tableau solution angle vrai
distance=np.empty(espac)  #tableau solution distances au soleil
Rx=np.empty(espac)  #tableau position planète coord x
Ry=np.empty(espac)  #tableau position planète coord y
j=0

for temps in t:
   M=n*temps  #calcul anomalie moyenne
   f=lambda E:  E-e*np.sin(E)-M  #équation de kepler
   f1=lambda E:  1-e*np.cos(E)  #1ère dérivée
   f2=lambda E:  e*np.sin(E) #2ème dérivée
   resol=optimize.newton(f,M,fprime=f1,fprime2=f2)  #recherche angle vrai avec équation de kepler
   sol[j]=resol   #tableau solution anomalie excentrique
   vrai[j]=2*np.arctan(coeff*np.tan(resol/2))   #tableau solution angle vrai
   distance[j]=a*(1-e*np.cos(resol))  #tableau solution distances au soleil en UA
   Rx[j]=c+distance[j]*np.cos(vrai[j]) #coordonnées x planète cart
   Ry[j]=distance[j]*np.sin(vrai[j])  #coordonnées y planète cart
   j+=1

#définition de la figure
fig = plt.figure(figsize=(10,10*b/a))
ax=fig.add_subplot(111)
ax.plot(c,0, color='r', marker='*')
ax.text(0.8*c,0,'Soleil',color='r')

#définition des données à dessiner
xdata,ydata=[],[]
point,=plt.plot([],[], 'b.', lw=1)

#partie animation
def début():
   ax.set_xlim(-1.1*a,1.1*a)
   ax.set_ylim(-1.1*b,1.1*b)
   point.set_data([],[])
   return point,

def maj(u):
   xdata.append(Rx[u])
   ydata.append(Ry[u])
```

```
    point.set_data(xdata,ydata)
    return point,

ani=FuncAnimation(fig, maj, frames=np.linspace(0,espac-1,espac,dtype=int),
                  init_func=début, blit = True, interval = 2, repeat = False)

#dessin graphique
plt.grid()
ax.set_xlabel('distances en UA')
ax.set_ylabel('distances en UA')
plt.title(titre)
ax.set_aspect('equal')

#sauvegarde fichier(optionnel)
#ani.save(titre+'.mp4',fps=30,extra_args=['-vcodec', 'libx264'])

plt.show()
```