TP d'Informatique : corrigé

PTSI A et B Lycée Eiffel

29 mai 2015

Première partie : ordre 1.

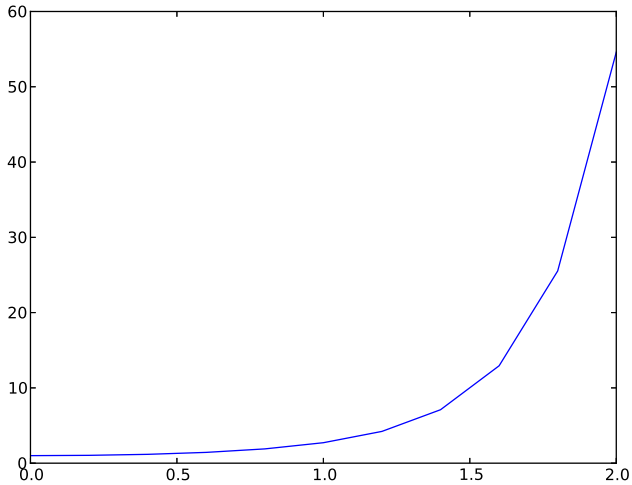
1. Une des multiples possibilités pour cette première question :

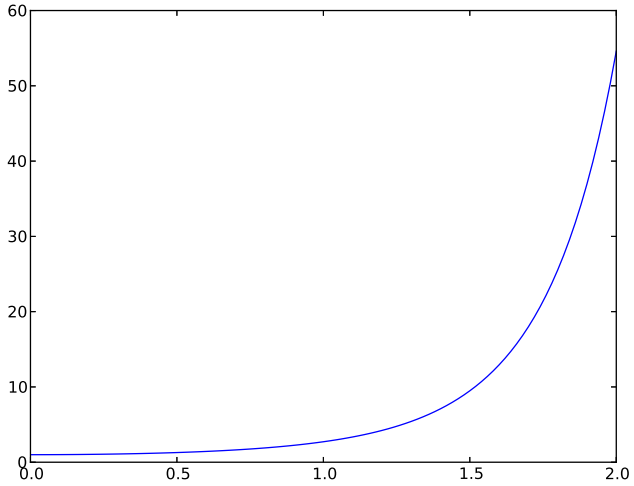
```
def découpage(t0,tf,n) :
    h=(tf-t0)/float(n)
    return [t0+i*h for i in range(n+1)]
```

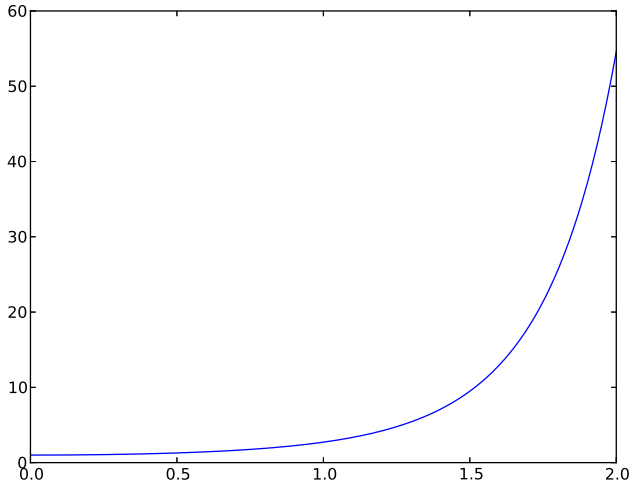
2. On peut taper le code suivant pour obtenir les trois courbes :

```
import matplotlib.pyplot as plt
t1=decoupage(0,2,10)
y1=[exp(i*i) for i in t1]
t2=decoupage(0,2,100)
y2=[exp(i*i) for i in t2]
t3=decoupage(0,2,1000)
y3=[exp(i*i) for i in t3]
plt.plot(t1,y1)
plt.plot(t2,y2)
plt.plot(t3,y3)
```

À l'oeil nu, on ne fait pas vraiment de différences entre les deux dernières courbes :







3. Il suffit de poser $F(t, y) = 2 * t * y$, soit en Python

```
def F(t,y) :
    return 2*t*y
```

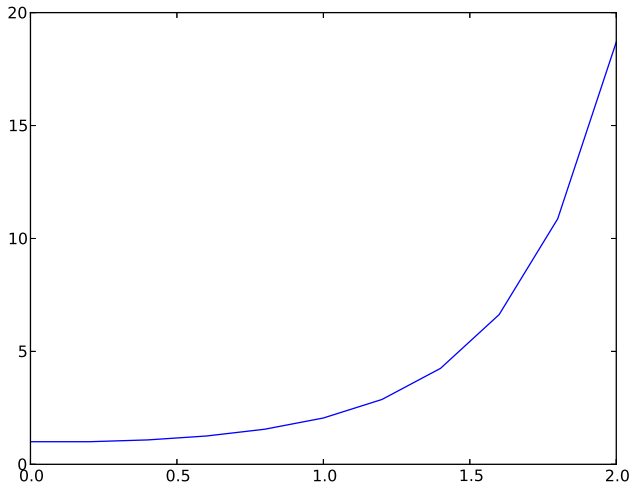
4.
```
def Euler(F,y0,t0,tf,n) :
    y,t=y0,t0
    temps,ordonnee=[t0],[y0]
    h=(tf-t0)/float(n)
    for i in range(n) :
        y=y+h*F(t,y)
        t=t+h
        temps.append[t]
        ordonnee.append[y]
    return ordonnee
```

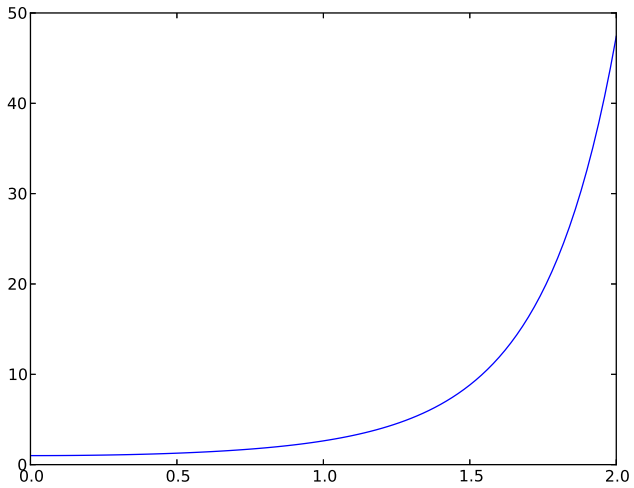
5. En reprenant les listes déjà définies précédemment, on peut taper le code suivant :

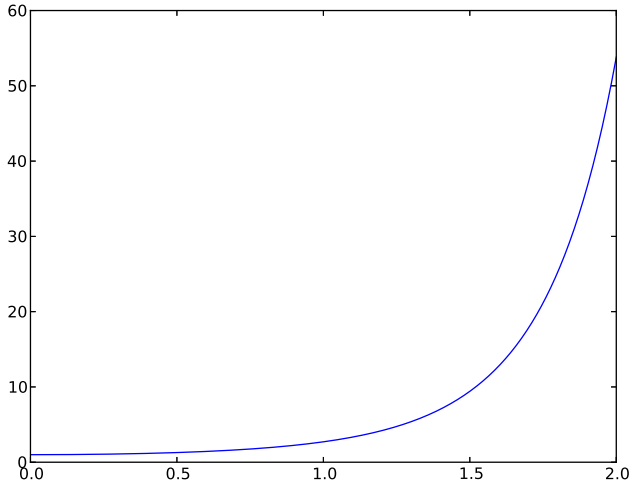
```
euler1=Euler(F,1,0,2,10)
plt.plot(t1,euler1)
```

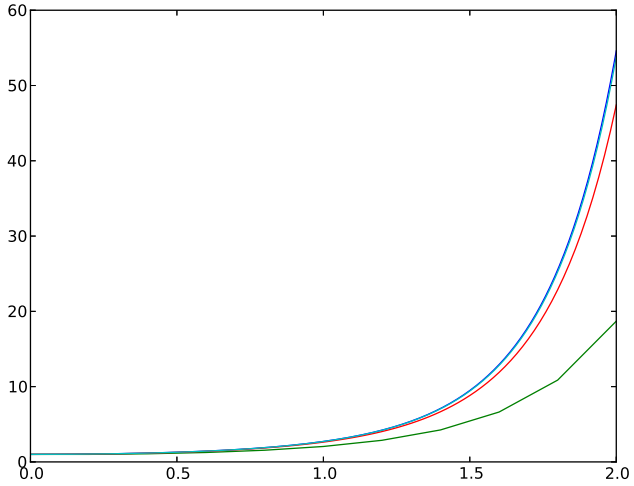
```
euler2=Euler(F,1,0,2,100)
plt.plot(t2,euler2)
euler3=Euler(F,1,0,2,1000)
plt.plot(t3,euler3)
```

Voici les trois courbes obtenues (séparément), puis un graphique regroupant ces trois courbes et la dernière courbe obtenue à la question 2 pour la solution exacte (la dernière courbe est pratiquement confondue avec la courbe théorique) :









On obtient comme valeur approchée de e^4 : pour $n = 10$, $e^4 \simeq 18.7$, pour $n = 100$, $e^4 \simeq 47.4$ et pour $n = 1\ 000$, $e^4 \simeq 53.81$. La valeur exacte donnée par Python est $e^4 \simeq 54.6$. Soit une erreur relative de 66% pour $n = 10$ (c'est évidemment horrible), de 13.2% pour $n = 100$, et de 1.5% pour $n = 1\ 000$ (ce qui commence à être très raisonnable).

6. C'est vraiment du simple copier-coller avec modification des relations de récurrence :

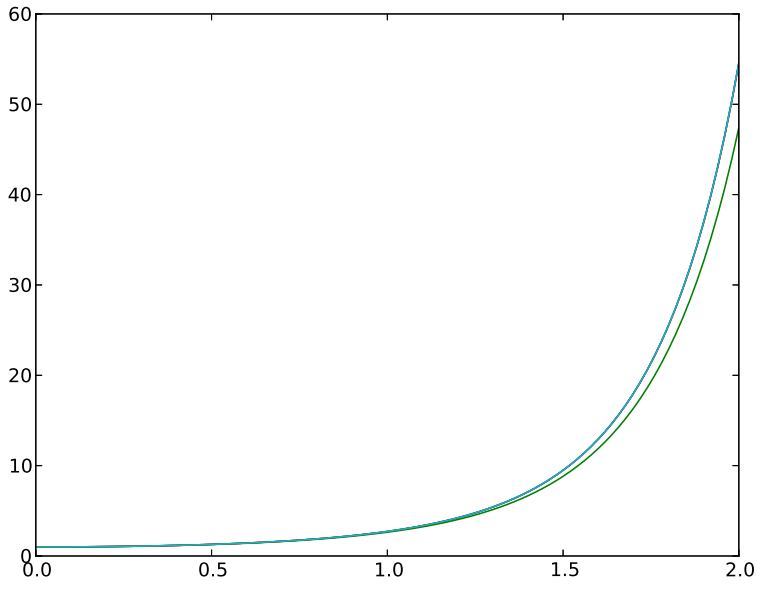
```
def Heun(F,y0,t0,tf,n) :
    y,t=y0,t0
    temps,ordonnee=[t0],[y0]
    h=(tf-t0)/float(n)
    for i in range(n) :
        y=y+h/2*(F(t,y)+F(t+h,y+h*F(t,y)))
        t=t+h
        temps.append[t]
        ordonnee.append[y]
    return ordonnee

def RK4(F,y0,t0,tf,n) :
    y,t=y0,t0
```

```
    temps,ordonnee=[t0],[y0]
    h=(tf-t0)/float(n)
    for i in range(n) :
        a=y+h/2*F(t,y)
        b=y+h/2*F(t+h/2,a)
        c=y+h*F(t+h/2,b)
        y=y+h/6*(F(t,y)+2*F(t+h/2,a)+2*F(t+h/2,b)+F(t+h,c))
        t=t+h
        temps.append[t]
        ordonnee.append[y]
    return ordonnee
```

7. On complète notre code avec les lignes suivantes :

```
heun=Heun(F,1,0,2,100)
plt.plot(t2,heun)
rk=RK4(F,1,0,2,100)
plt.plot(rk,heun)
```

On obtient les quatre courbes suivantes (de bas en haut, Euler, Heun, RK4 et la courbe exacte). Sans surprise, les méthodes plus pointues sont plus rapidement efficaces. Si on regarde les listes de valeurs ressorties par les différents programmes, Heun pour $n = 10$ est à peu près aussi efficace qu'Euler pour $n = 100$ et RK4 pour $n = 10$ est lui-même aussi efficace que Heun pour $n = 100$ et qu'Euler pour $n = 1000$.



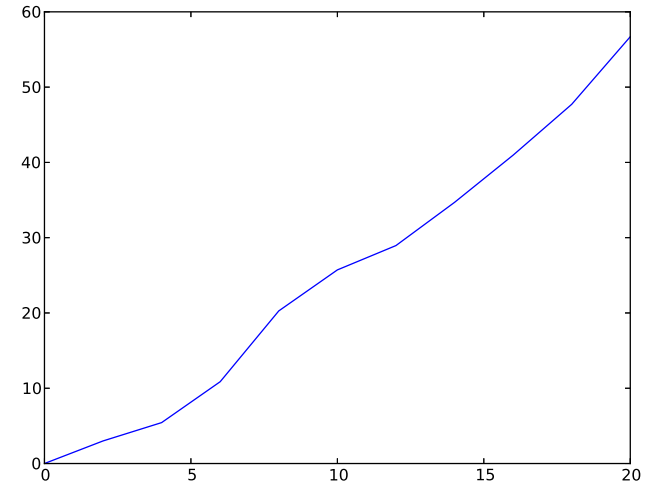
Deuxième partie : ordre 2.

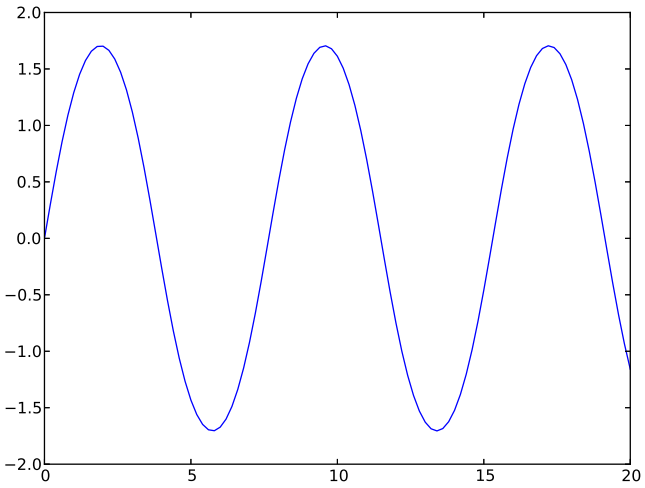
1. Il suffit d'adapter ce qu'on a fait plus haut avec les relations de récurrence de l'énoncé :

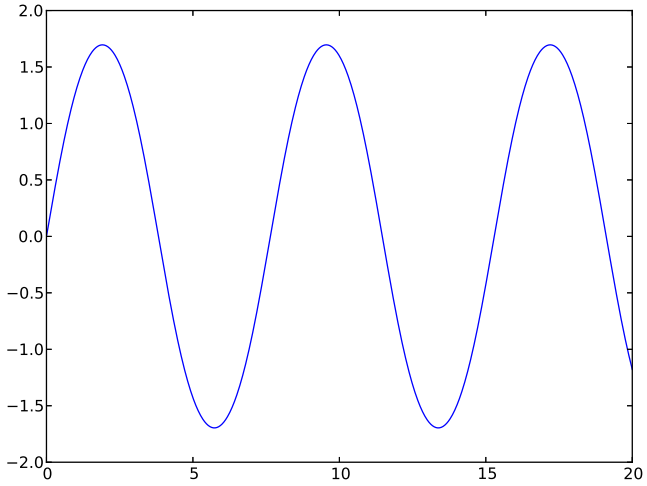
```
def pendule(y0,z0,t0,tf,n) :
    y,z,t=y0,z0,t0
    temps,ordonnee,derivee=[t0],[y0],[z0]
```

```
h=(tf-t0)/float(n)
for i in range(n) :
        y=y+h*z
        z=z-h*sin(y)
        t=t+h
        temps.append(t)
        ordonnee.append(y)
        derivee.append(z)
plt.plot(temps,ordonnee)
return ordonnee
```

2. On obtient les trois courbes suivantes.





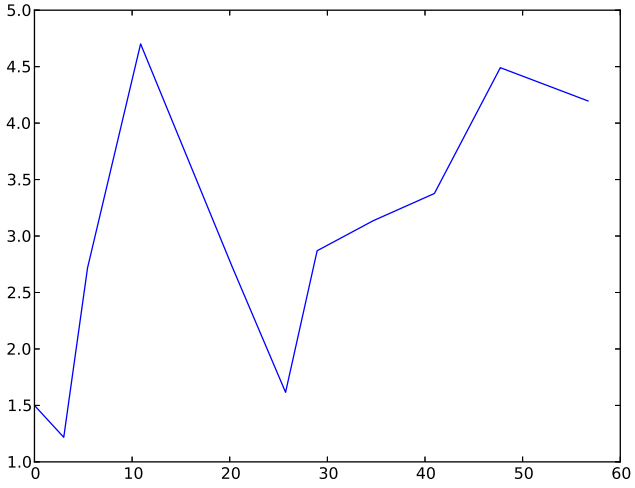


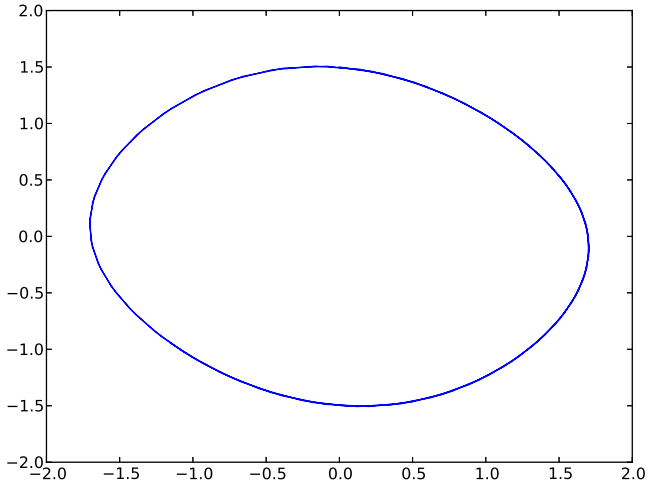
On voit bien que pour $n = 10$, on obtient absolument n'importe quoi. Les deux dernières courbes s'approchent de quelque chose de périodique, c'est nettement plus convaincant.

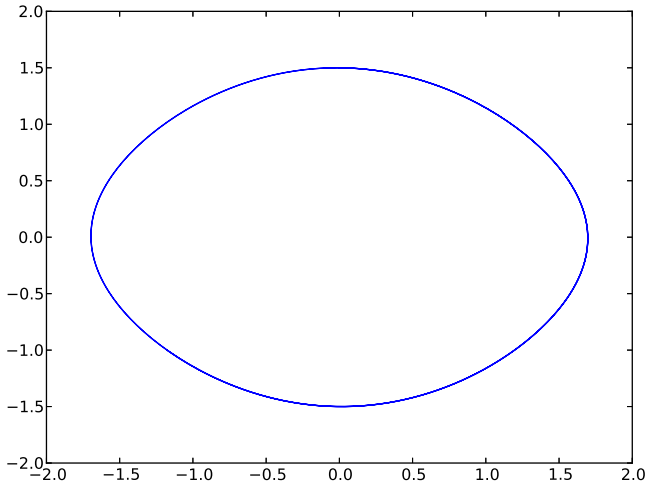
3. Il suffit évidemment de remplacer l'avant-dernière ligne par la suivante :

   plt.plot(ordonnee,derivee)

   On obtient les trois portraits de phase suivants :







On constate sans surprise que la première courbe n'a pas l'ombre d'un intérêt. La deuxième se rapproche de ce qu'on devrait obtenir, à savoir un ovale (périodique) symétrique par rapport aux deux axes. La troisième courbe est encore plus symétrique que la précédente.

4. Pour la méthode de Heun, la seule chose à laquelle il faut faire attention est de modifier simultanément les valeurs de y et de z à chaque étape (j'ai utilisé des variables intermédiaires pour le faire, histoire d'avoir quelque chose qui ressemble déjà à ce qu'on écrira pour Runge-Kutta en-dessous) :

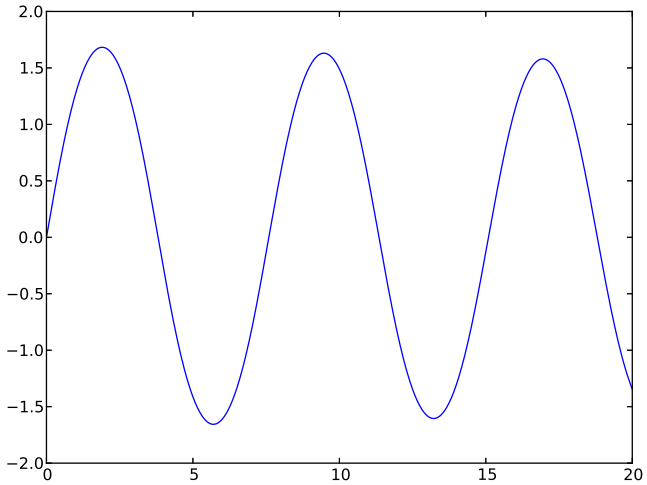
```
def penduleheun(y0,z0,t0,tf,n) :
    y,z,t=y0,z0,t0
    temps,ordonnee,derivee=[t0],[y0],[z0]
    h=(tf-t0)/float(n)
    for i in range(n) :
        ay=y+h*z
        az=z-h*sin(y)
        y=y+h/2*(z+az)
        z=z-h/2*(sin(y)+sin(ay))
        t=t+h
        temps.append(t)
```

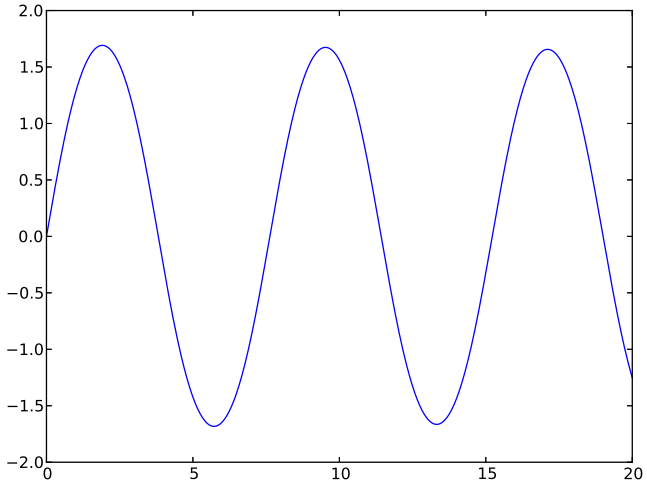
```
            ordonnee.append(y)
            derivee.append(z)
        plt.plot(temps,ordonnee)
        return ordonnee
```

Pour Runge-Kutta :

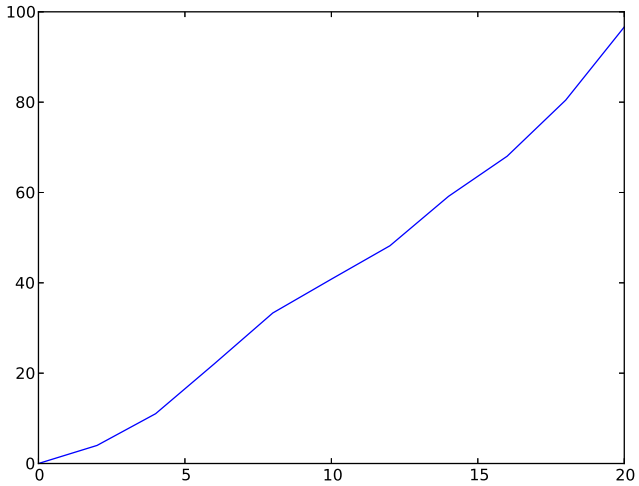
```
def pendulerk(y0,z0,t0,tf,n) :
    y,z,t=y0,z0,t0
    temps,ordonnee,derivee=[t0],[y0],[z0]
    h=(tf-t0)/float(n)
    for i in range(n) :
        ay=y+h/2*z
        az=z-h/2*sin(y)
        by=y+h/2*az
        bz=z-h/2*sin(ay)
        cy=y+h*bz
        cz=z-h*sin(by)
        y=y+h/6*(z+2*az+2*bz+cz)
        z=z-h/6*(sin(y)+2*sin(ay)+2*sin(by)+sin(cy))
        t=t+h
        temps.append(t)
        ordonnee.append(y)
        derivee.append(z)
    plt.plot(temps,ordonnee)
    return ordonnee
```

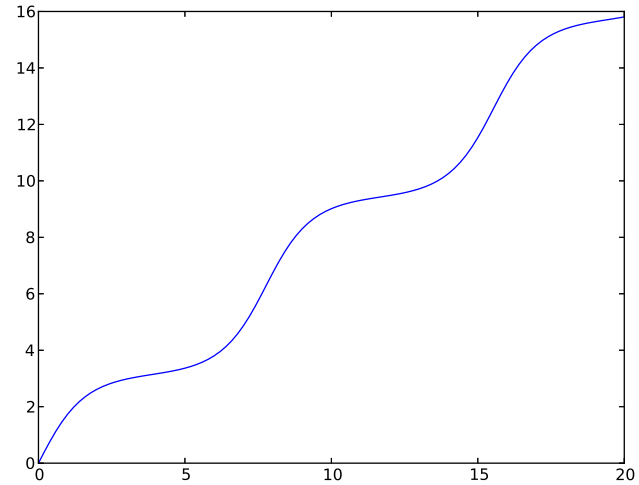
Les courbes que j'obtiens par ces deux méthodes restent pourries pour $n = 10$, mais sont curieusement amorties pour $n = 100$, ce qui a l'air de ne pas être tout à fait normal (mais je n'arrive pas à déterminer ce qui ne va pas pour l'instant, alors tant pis). Pour $n = 1\ 000$, on obtient les deux courbes suivantes (Heun en haut, Runge-Kutta en bas) :

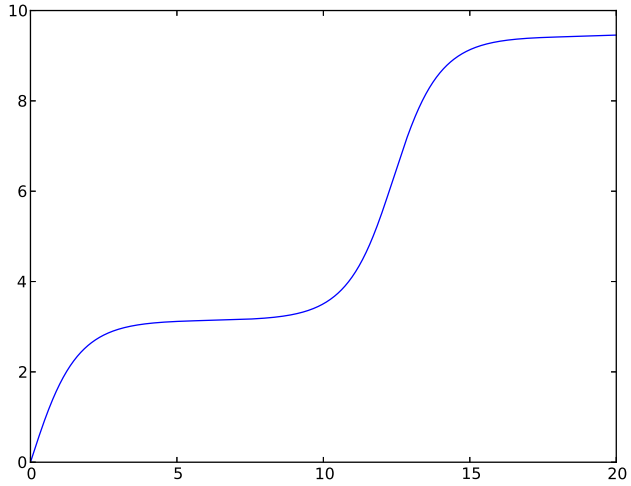


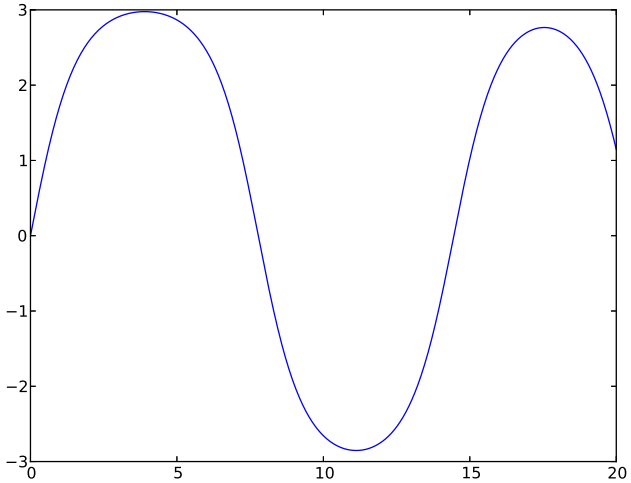


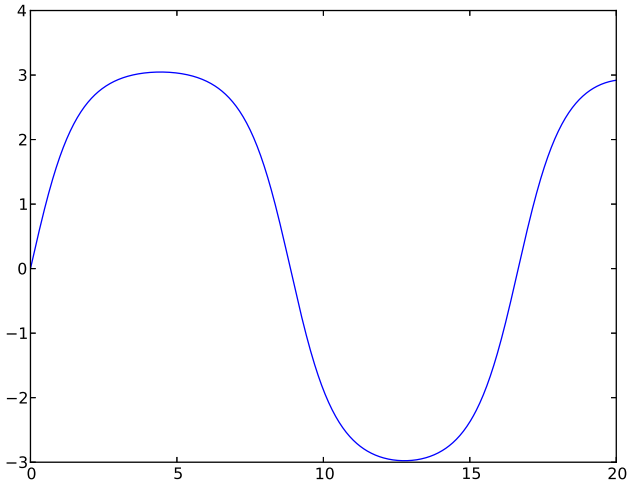
5. Je me contente de donner les courbes, dans l'ordre Euler avec $n = 10$, $n = 100$ et $n = 1\ 000$, puis Heun et Runge-Kutta avec $n = 1\ 000$ à chaque fois.











Aucune des méthodes ne donne de résultats satisfaisants, ce qui est normal : quand on s'approche d'un point d'équilibre (y prenant une valeur multiple impaire de π), les valeurs de la dérivée sont censées se rapprocher de 0, mais on n'aura jamais une approximation numérique

suffisamment bonne pour rester sur le point d'équilibre correspondant (qui est physiquement un équilibre instable, on est dans la situation où le pendule reste en équilibre dirigé vers le haut), et on repart donc vers une autre position d'équilibre. Détail curieux, avec la méthode d'Euler, ce n'est pas la même position d'équilibre qui est atteinte la seconde fois qu'avec Heun et Runge-Kutta. Si on fait des simulations avec des valeurs de n plus grandes, la courbe se stabilise plus longtemps au voisinage de l'équilibre, mais finit toujours par s'en éloigner.