PreparationCentraleMaths2_1718

June 3, 2018

1 Lycée Henri Poincaré --- année scolaire 2017-2018

1.1 Préparation à l'oral Mathématiques 2 du concours Centrale-Supélec

1.2 PC - PC* - PSI

Édouard Lebeau, mai-juin 2018 Nous allons explorer la documentation Python fournie par le concours. Le poly correspondant peut être récupéré ici

```
In [1]: # Importons
        import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        import scipy.optimize as resol
        import scipy.integrate as integr
        import numpy.random as rd
        from numpy.polynomial import Polynomial
```

2 Exercice 1 (écriture décimale d'un entier)

```
In [2]: # Exercice 1, question a

        def nombre9(n):
            total = 0
            quotient = n
            test = True
            while test:
                reste = quotient % 10
                quotient = quotient // 10
                if reste == 9:
                    total += 1
                else:
                    test = False
            return total

        # Testons la chose
```

```
        tests = [1, 9, 99, 199, 1999999, 9991]
        objectifs = [0, 1, 2, 2, 6, 0]
        resultat = [nombre9(n) for n in tests]
        print("Objectif : {}".format(objectifs))
        print("Résultat obtenu : {}".format(resultat))
        print("Succès ? {}".format(objectifs == resultat))
```

```
Objectif : [0, 1, 2, 2, 6, 0]
Résultat obtenu : [0, 1, 2, 2, 6, 0]
Succès ? True
```

On peut aussi programmer ça récursivement.

In [3]:
```
# Version récursive

def nombre9_rec(n):
    a = n % 10
    if a < 9:
        return 0
    else:
        return 1 + nombre9_rec(n // 10)
```

In [4]:
```
tests = [1, 9, 99, 199, 1999999, 9991]
objectifs = [0, 1, 2, 2, 6, 0]
resultat = [nombre9_rec(n) for n in tests]
print("Objectif : {}".format(objectifs))
print("Résultat obtenu : {}".format(resultat))
print("Succès ? {}".format(objectifs == resultat))
```

```
Objectif : [0, 1, 2, 2, 6, 0]
Résultat obtenu : [0, 1, 2, 2, 6, 0]
Succès ? True
```

In [5]:
```
# Exercice 1, question b

def avant9(n):
    quotient = n
    reste = 9
    while reste == 9:
        reste = quotient % 10
        quotient = quotient // 10
    return reste

# Testons la chose

tests = [1, 9, 929, 199, 1999999, 9991]
objectifs = [1, 0, 2, 1, 1, 1]
```

```
        resultat = [avant9(n) for n in tests]
        print("Objectif : {}".format(objectifs))
        print("Résultat obtenu : {}".format(resultat))
        print("Succès ? {}".format(objectifs == resultat))
```

```
Objectif : [1, 0, 2, 1, 1, 1]
Résultat obtenu : [1, 0, 2, 1, 1, 1]
Succès ? True
```

3 Exercice 2 (suite récurrente)

La suite $(a_n)_{n\in\mathbb{N}}$ est définie par les conditions initiales $(a_0, a_1) = (1, 1)$ et les relations de récurrence $a_{2n+1} = a_n$ et $a_{2n} = a_n + a_{n-1}$.

In [6]:
```
# Exercice 2, question a

def a(n):
    if n < 2:
        return 1
    else:
        quotient = n // 2
        reste = n % 2
        if reste == 1:
            return a(quotient)
        else:
            return a(quotient) + a(quotient-1)

print(a(534))  # 39 ?
```

```
39
```

In [7]:
```
# Juste histoire de voir un peu
print(a(12345))
```

```
95
```

Pour la deuxième question, on demande d'écrire une fonction qui renvoie la liste $[a_0, \ldots, a_n]$. Une première possibilité est bien sûr d'engendrer les $n + 1$ termes de cette liste en appelant la fonction de la première question. Cela implique néanmoins d'effectuer des calculs redondants, alors que l'application des relations de récurrence permet de remplir la liste directement avec une boucle for de complexité $\mathcal{O}(n)$.

In [8]:
```
# Exercice 2, question b
# Première version
```

```
        def tab_a(n):
            return [a(k) for k in range(n+1)]

        # Test
        tab100 = tab_a(100)
        print(tab100)
```

```
[1, 1, 2, 1, 3, 2, 3, 1, 4, 3, 5, 2, 5, 3, 4, 1, 5, 4, 7, 3, 8, 5, 7, 2, 7, 5, 8, 3, 7, 4, 5, 1,
```

```
In [9]: # Deuxième version
        # Pour gérer les questions de parité, on remplit un tableau susceptible
        # d'avoir un terme en trop et on le retire si besoin est


        def tab_a_bis(n):
            n2 = n // 2
            resultat = [1] * (2*n2 + 2)
            for k in range(1, n2 + 1):
                resultat[2*k] = resultat[k] + resultat[k-1]
                resultat[2*k+1] = resultat[k]
            return resultat[:n+1]

        # Test
        retab100 = tab_a_bis(100)
        print(retab100)
        print(retab100 == tab100)
```

```
[1, 1, 2, 1, 3, 2, 3, 1, 4, 3, 5, 2, 5, 3, 4, 1, 5, 4, 7, 3, 8, 5, 7, 2, 7, 5, 8, 3, 7, 4, 5, 1,
True
```

```
In [10]: tab_a(101) == tab_a_bis(101)  # Testons l'autre parité

Out[10]: True
```

Pour l'anecdote, le nombre a_n compte les polynômes à coefficients dans $\{0,1,2\}$ tels que $P(2) = n$. Par exemple, l'égalité $a_4 = 3$ est associée aux trois polynômes X^2, $X+2$, $2X$, qui vérifient l'égalité $P(2) = 4$. Je laisse en exercice la démonstration de ce fait.

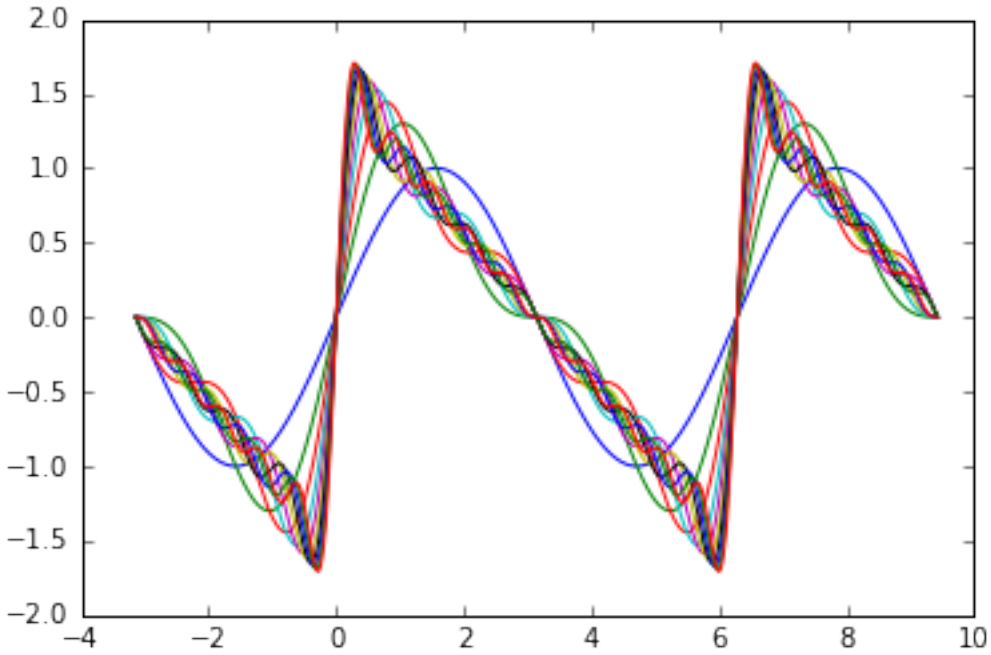
Cette suite a également la particularité que la suite $(a_{n+1}/a_n)_{n\in\mathbb{N}}$ décrit bijectivement l'ensemble des nombres rationnels strictement positifs, écrits sous forme de fractions irréductibles.

Pour en savoir plus

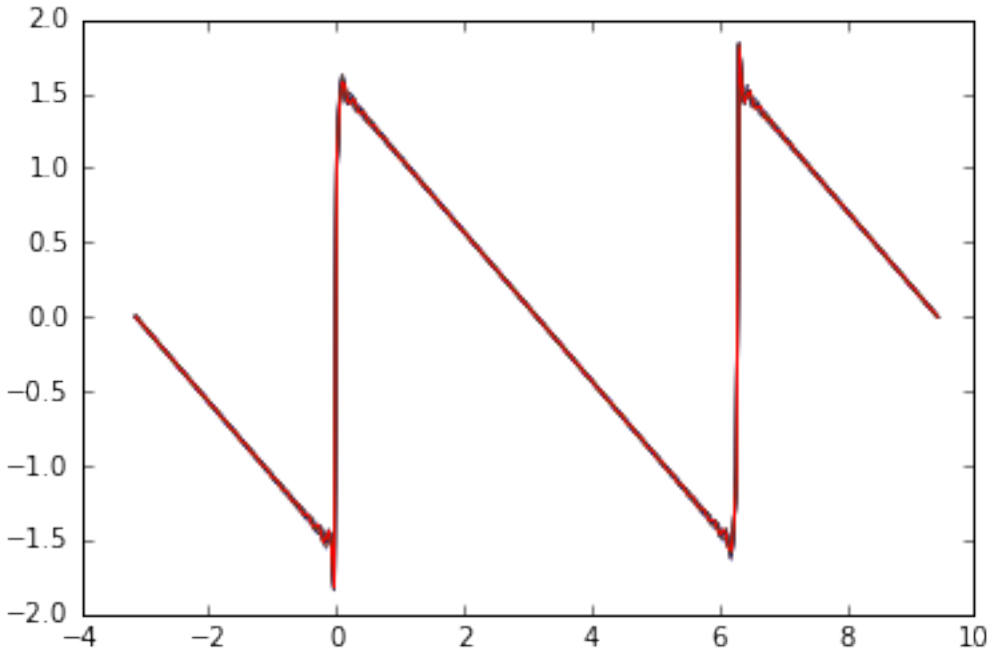
4 Exercice 3 (représentations graphiques de fonctions)

```
In [11]: theta = np.linspace(-np.pi, 3*np.pi, 500)
         theta1 = 0*theta
         for n in range(1, 11):
```

```
        theta1 = theta1 + np.sin(n*theta) / n
        plt.plot(theta, theta1)
    plt.show()
```



In [12]:
```
theta = np.linspace(-np.pi, 3*np.pi, 300)
theta1 = 0*theta
for n in range(1, 101):
    theta1 = theta1 + np.sin(n*theta) / n
for n in range(101, 111):
    theta1 = theta1 + np.sin(n*theta) / n
    plt.plot(theta, theta1)
plt.show()
```



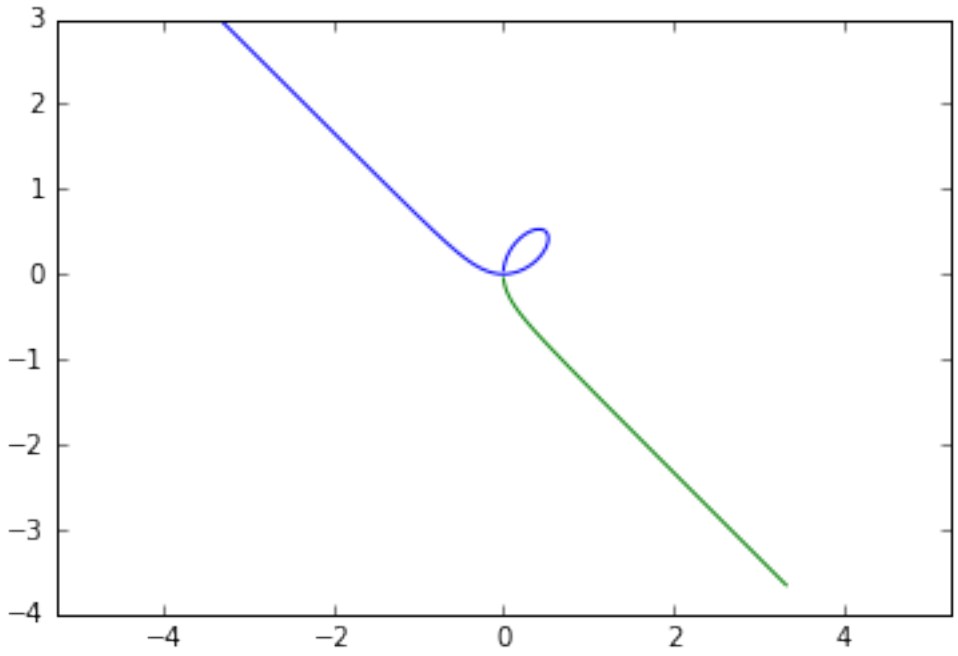
On peut prouver la formule

$$\forall \theta \in]0, \pi[, \qquad \sum_{k=1}^{+\infty} \frac{\sin(k\theta)}{k} = \frac{\pi - \theta}{2}.$$

Ce signal périodique s'appelle *dents-de-scie*. Il est très utilisé en synthèse sonore. On observe la convergence sur le schéma précédent, mais on observe aussi un saut anormalement grand au niveau des discontinuités. Ce saut peut être quantifié (c'était d'ailleurs l'objectif de l'exercice d'origine). On l'appelle *phénomène de Gibbs*.

5 Exercice 4 (tracé d'une courbe paramétrée)

```
In [13]: def x(t):
             return t / (1 + t**3)

         def y(t):
             return t**2 / (1 + t**3)

In [14]: plt.axis('equal')  # Pour avoir un repère orthonormé
         t = np.linspace(-0.9, 20, 300)
         plt.plot(x(t), y(t))
         t = np.linspace(-20, -1.1, 300)
         plt.plot(x(t), y(t))
         plt.show()
```



La boucle est décrite sur l'intervalle $[0, +\infty[$. La longueur de cette boucle est donnée par la formule cinématique suivante (qui n'est pas au programme du cours de mathématiques)

$$\int_0^{+\infty} \sqrt{x'(t)^2 + y'(t)^2}\, \mathrm{d}t.$$

L'intégrande doit être calculé à la main pour faire effectuer ensuite le calcul numérique par Scipy.

```
In [15]: def long(t):
             return np.sqrt((1-2*t**3)**2 + (2*t-t**4)**2)/(1+t**3)**2

         print(integr.quad(long, 0, np.inf))
```

```
(1.6391629072271778, 4.92589510655565e-10)
```

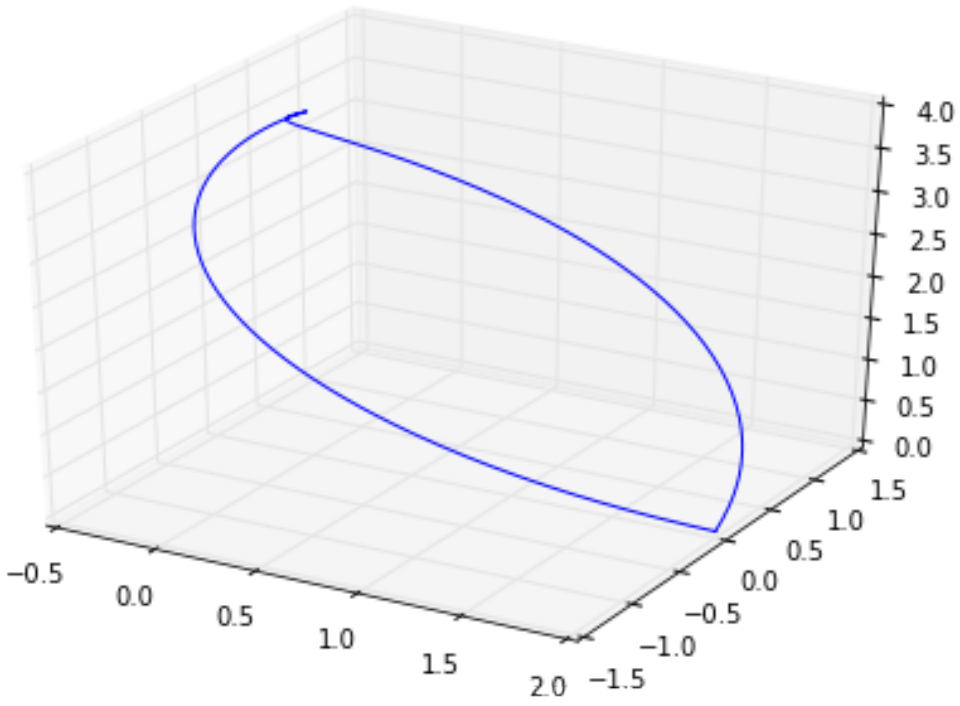
Cette courbe s'appelle le *folium de Descartes*. Pour en savoir plus

6 Exercice 5

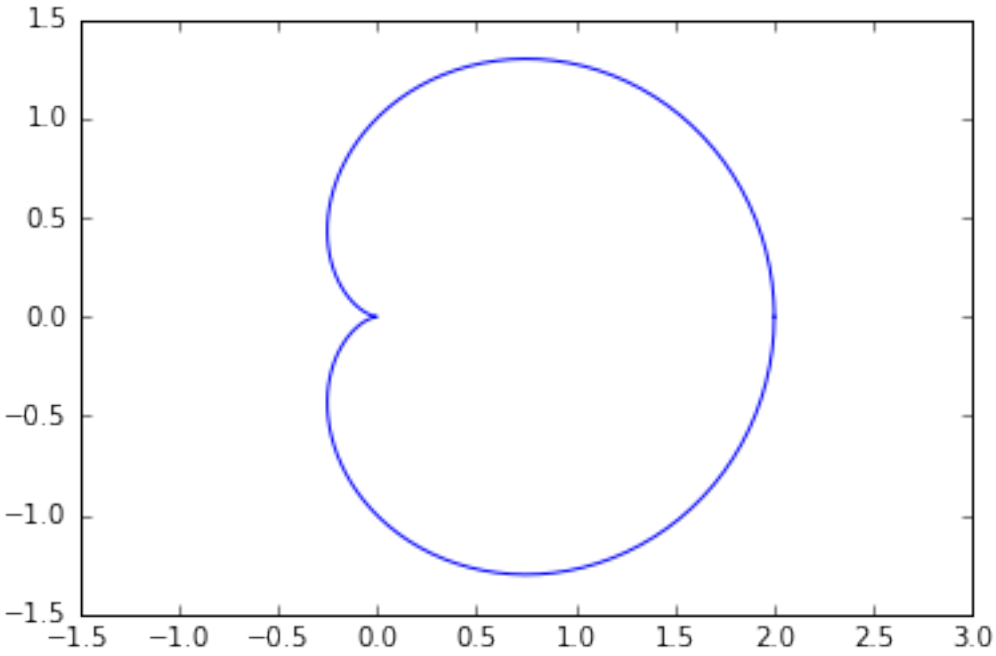
```
In [16]: def x(t):
             return (1+np.cos(t))*np.cos(t)

         def y(t):
             return (1+np.cos(t))*np.sin(t)
```

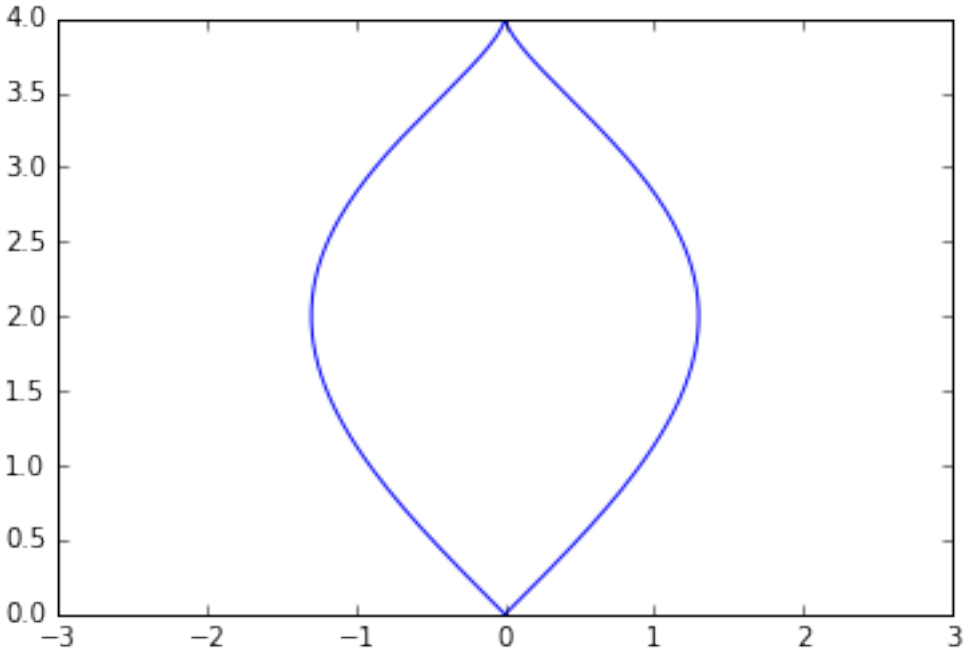
```
        def z(t):
            return 4*np.sin(t/2)
```

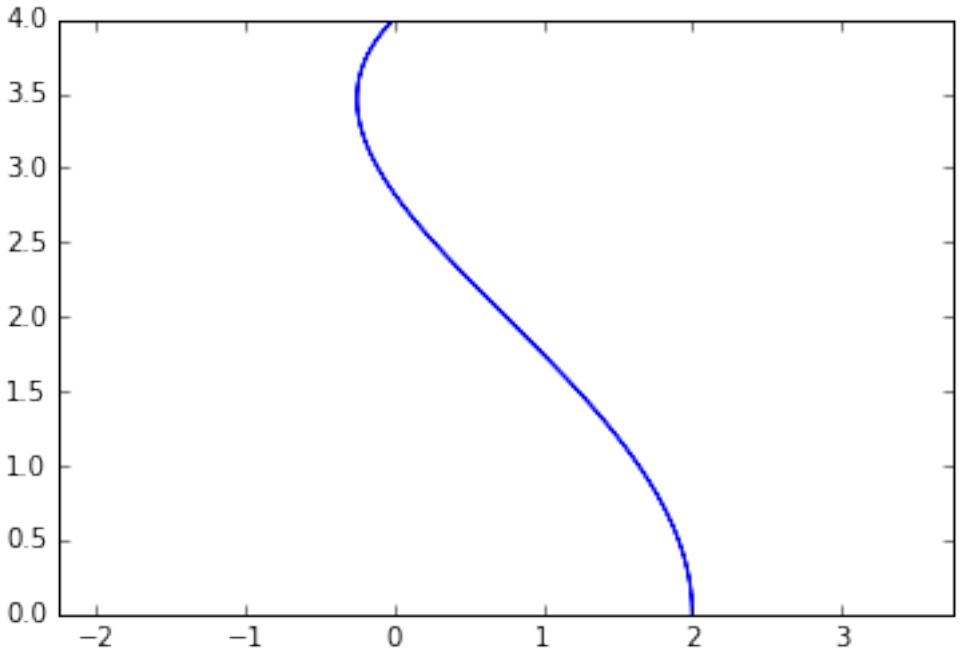
In [17]:
```
ax = Axes3D(plt.figure())
t = np.linspace(0, 2*np.pi, 300)
ax.plot(x(t), y(t), z(t))
plt.show()
```



In [18]:
```
# Projections sur les plans de coordonnées

# (Oxy)
plt.axis('equal')
plt.plot(x(t), y(t))
plt.show()

# (Oyz)
plt.axis('equal')
plt.plot(y(t), z(t))
plt.show()

# (Oxz)
plt.axis('equal')
plt.plot(x(t), z(t))
plt.show()
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

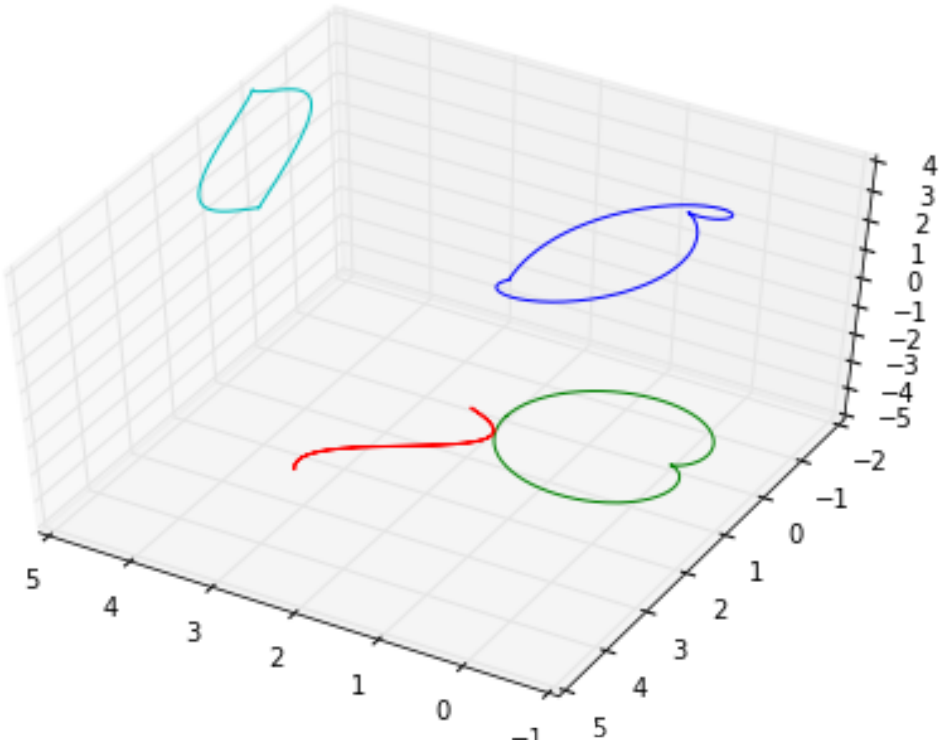

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−3
−2
−1
0
1
2
3



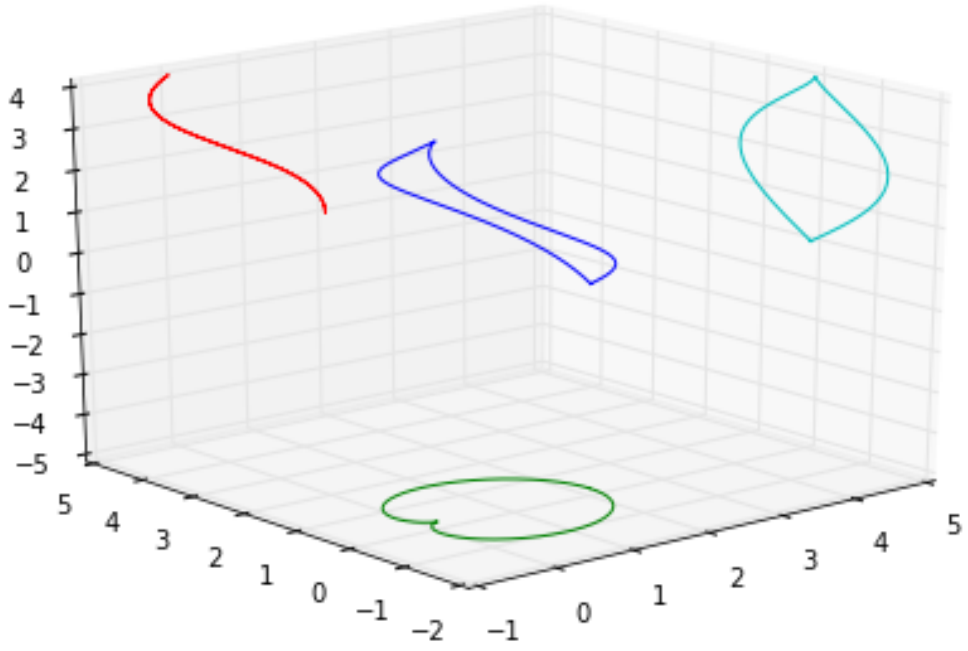
La première des trois courbes planes obtenues par projection est une *cardioïde*. Pour en savoir plus

6.0.1 Tentative de tout superposer

```
In [19]: ax = Axes3D(plt.figure())
         t = np.linspace(0, 2*np.pi, 300)
         x1 = x(t)
         y1 = y(t)
         z1 = z(t)
         t5 = 5*np.ones_like(t)
         ax.plot(x1, y1, z1)
         ax.plot(x1, y1, -t5)  # Projection sur (Oxy)
         ax.plot(x1, t5, z1)  # Projection sur (Oxz)
         ax.plot(t5, y1, z1)  # Projection sur (Oyz)
         ax.view_init(50, 120)
         plt.show()
```



In [20]:
```
ax = Axes3D(plt.figure())
t = np.linspace(0, 2*np.pi, 300)
x1 = x(t)
y1 = y(t)
z1 = z(t)
t5 = 5*np.ones_like(t)
ax.plot(x1, y1, z1)
ax.plot(x1, y1, -t5)  # Projection sur (Oxy)
ax.plot(x1, t5, z1)  # Projection sur (Oxz)
ax.plot(t5, y1, z1)  # Projection sur (Oyz)
ax.view_init(20, 230)
plt.show()
```



7 Exercice 6 (une matrice et ses valeurs propres)

```
In [21]: def mat_an(n):
             mat = np.diag(range(1, n+1))
             mat[0, :] = 1
             mat[:, 0] = 1
             return mat

In [22]: # Test
         print(mat_an(9))
```

```
[[1 1 1 1 1 1 1 1 1]
 [1 2 0 0 0 0 0 0 0]
 [1 0 3 0 0 0 0 0 0]
 [1 0 0 4 0 0 0 0 0]
 [1 0 0 0 5 0 0 0 0]
 [1 0 0 0 0 6 0 0 0]
 [1 0 0 0 0 0 7 0 0]
 [1 0 0 0 0 0 0 8 0]
 [1 0 0 0 0 0 0 0 9]]
```

Observons qu'il s'agit d'une matrice symétrique réelle. Le théorème spectral garantit en particulier que ses valeurs propres sont toutes réelles. Cela a donc un sens de parler de sa plus grande valeur propre.

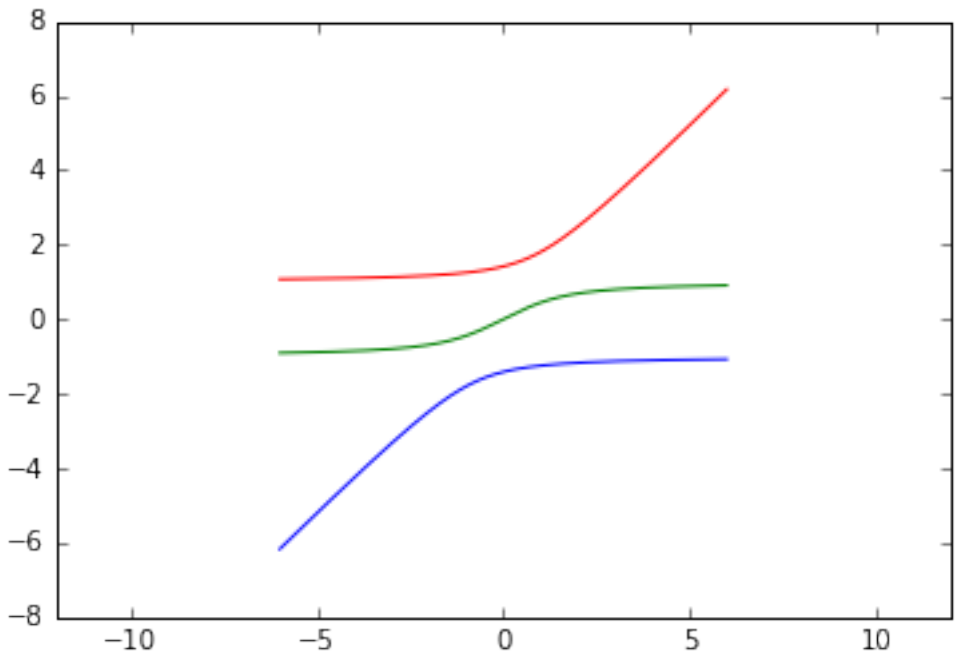
```
In [23]: def max_vp(n):
             mat = mat_an(n)
             spectre = np.linalg.eigvals(mat)
             return max(spectre)

In [24]: [max_vp(n) for n in range(1, 21)]

Out[24]: [1.0,
          2.6180339887498949,
          3.5320888862379567,
          4.4309078368330663,
          5.3472060613930781,
          6.2834227128619622,
          7.2356832141917806,
          8.1997080496052881,
          9.1721489357263692,
          10.150626128348826,
          11.133492941183421,
          12.119609535187349,
          13.108177540484951,
          14.098628512458406,
          15.090550448407608,
          16.083639409528324,
          17.077667366583192,
          18.072460526420571,
          19.067884489923529,
          20.063833921736141]

In [25]: max_vp(100)

Out[25]: 100.01065409091318
```

On semble s'orienter vers un développement asymptotique de la forme $n + 1/n + o(1/n)$.

8 Exercice 7 (valeurs propres d'une matrice paramétrée)

```
In [26]: def mat_m(t):
             return np.array([[t, 0, 1],
                              [0, 0, 1],
                              [1, 1, 0]])

In [27]: mat_m(3)

Out[27]: array([[3, 0, 1],
                [0, 0, 1],
                [1, 1, 0]])
```

```
In [28]: def spectre(t):
             a = np.linalg.eigvals(mat_m(t))
             a.sort()
             return a

In [29]: spectre(5)

Out[29]: array([-1.08553172,  0.88582674,  5.19970498])

In [30]: nb = 500
         t = np.linspace(-6, 6, nb)
         lambda1 = 0 * t
         lambda2 = 0 * t
         lambda3 = 0 * t
         for i in range(nb):
             lambda1[i], lambda2[i], lambda3[i] = spectre(t[i])

         plt.plot(t, lambda1)
         plt.plot(t, lambda2)
         plt.plot(t, lambda3)
         plt.axis('equal')  # Repère orthonormé
         plt.show()
```




```
In [31]: spectre(10000)

Out[31]: array([ -1.00005000e+00,   9.99949996e-01,   1.00000001e+04])
```

```
In [32]: spectre(-10000)

Out[32]: array([ -1.00000001e+04,  -9.99949996e-01,   1.00005000e+00])
```

Tout ceci permet d'émettre des conjectures quant au comportement asymptotique des trois valeurs propres.

Conjecture quand t tend vers $+\infty$

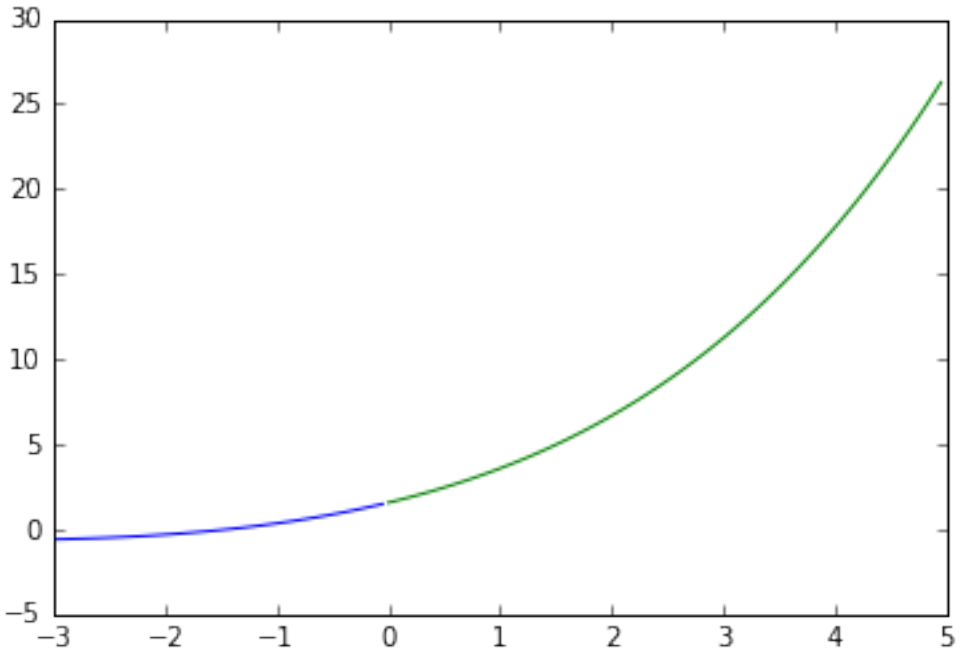
$$\lambda_1(t) = -1 + \frac{1}{2t} + o\left(\frac{1}{t}\right), \quad \lambda_2(t) = 1 - \frac{1}{2t} + o\left(\frac{1}{t}\right), \quad \lambda_3(t) = t + \frac{1}{t} + o\left(\frac{1}{t}\right).$$

Et ça se démontre (voir annexe à la fin de ce document).

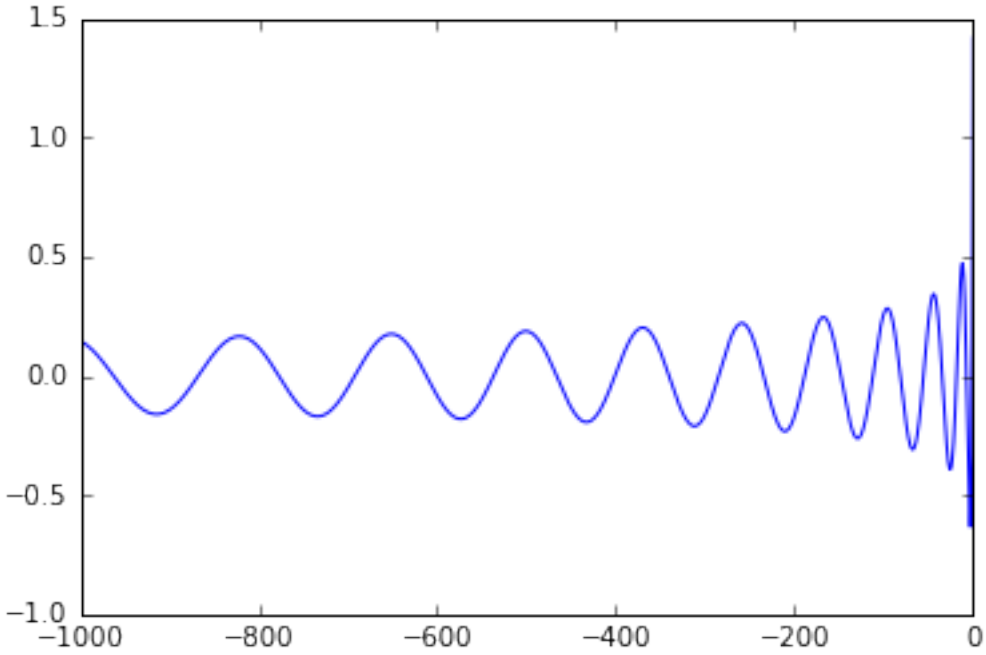
9 Exercice 8 (fonction définie par une intégrale à paramètre)

Pour les calculs, je décide de ne pas m'encombrer de la constante multiplicative $2/\pi$, dans la mesure où elle n'influe pas sur les représentations graphiques.

```
In [33]: def phi_plus(x):
             def integrande(t):
                 return np.cosh(2*np.sqrt(x)*np.sin(t))
             return integr.quad(integrande, 0, np.pi/2)[0]

         def phi_moins(x):
             def integrande(t):
                 return np.cos(2*np.sqrt(-x)*np.sin(t))
             return integr.quad(integrande, 0, np.pi/2)[0]

In [34]: x1 = np.arange(-3, 0, 0.05)
         y1 = [phi_moins(x) for x in x1]
         x2 = np.arange(0, 5, 0.05)
         y2 = [phi_plus(x) for x in x2]
         plt.plot(x1, y1)
         plt.plot(x2, y2)
         plt.show()
```



In [35]:
```
x3 = np.arange(-1000, 0, 0.1)
y3 = [phi_moins(x) for x in x3]
plt.plot(x3, y3)
plt.show()
```

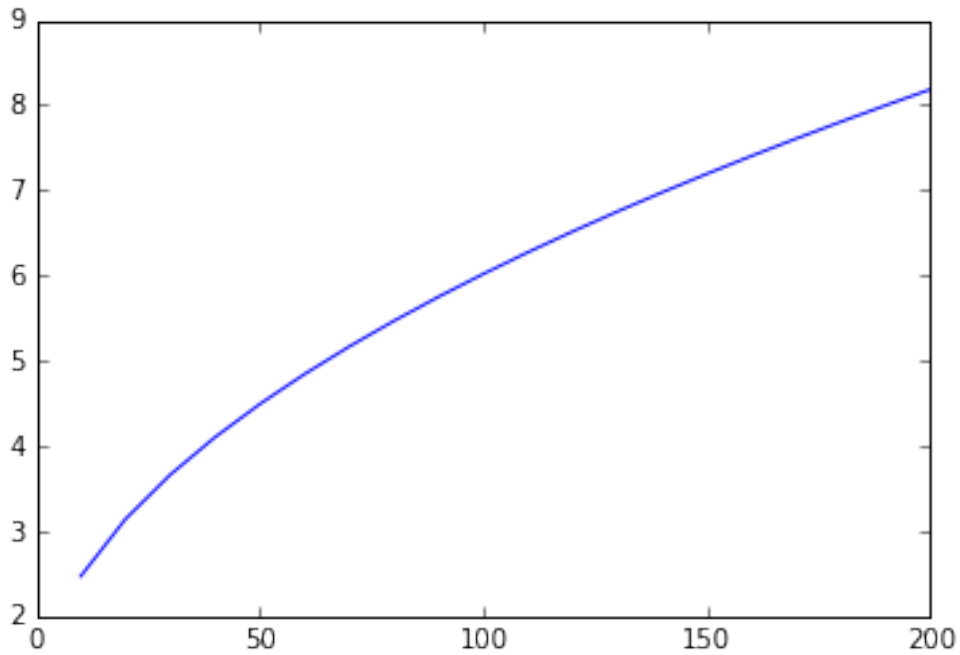


10 Exercice 9 (résolution approchée d'une équation)

In [36]:
```
# Pour réduire au maximum la complexité du calcul de la fonction
# il faut éviter de calculer les factorielles individuellement
# et profiter de leur relation de récurrence pour tout calculer
# au fur et à mesure.

def f(x, n):
    somme = 1
    terme = 1
    for k in range(1, n+1):
        terme *= x / k
        somme += terme
    return somme * np.exp(-x)
```

In [37]:
```
def resolution(n, lam):
    # Renvoie an - n
    def g(x):
        return f(x, n) - lam
    an = resol.fsolve(g, n)[0]
    return an - n
```

In [38]:
```
lam = 0.3
x = range(10, 210, 10)
y = [resolution(n, lam) for n in x]
plt.plot(x, y)
plt.show()
```



11 Exercice 10 (binomiale-Poisson)

Sachant $(N = n)$, la loi conditionnelle de Y est $\mathcal{B}(50N, 1/50)$.

```
In [39]: p = 1/12  # Autant le calculer une fois pour toutes
         def simul_Y():
             n = rd.poisson(p)
             return rd.binomial(50*n, 0.02)
```

```
In [40]: moyenne = []
         ecartype = []
         for _ in range(10):
             y = [simul_Y() for i in range(1000)]
             moyenne.append(np.mean(y))
             ecartype.append(np.std(y))
         print(moyenne, np.mean(moyenne))
         print(ecartype, np.mean(ecartype))
```

```
[0.094, 0.065000000000000002, 0.071999999999999995, 0.081000000000000003, 0.098000000000000004,
[0.42090854113453197, 0.33282878481285239, 0.34178355724054366, 0.41284258501273824, 0.473704549
```

12 Exercice 11 (marche aléatoire)

```
In [41]: def simul_T():
             compteur = 0
             disque1 = 0
             disque2 = 1
             while disque1 != disque2:
                 compteur += 1
                 move1 = rd.choice((-1, 1))
                 move2 = rd.choice((-1, 1))
                 disque1 = (disque1 + move1) % 5
                 disque2 = (disque2 + move2) % 5
             return compteur
```

```
In [42]: simulation = [simul_T() for _ in range(1000)]
```

```
In [43]: min(simulation), max(simulation)
```

```
Out[43]: (2, 96)
```

```
In [44]: np.mean(simulation)
```

```
Out[44]: 12.188000000000001
```

```
In [45]: def estimation_esperance_T(nb_essais=10000):
             somme = 0
             for _ in range(nb_essais):
                 somme += simul_T()
             return somme / nb_essais
```

```
In [46]: for _ in range(20):
             print(estimation_esperance_T())
```

```
12.1006
12.1581
11.9671
12.022
11.8745
11.8562
11.8724
11.8125
11.9305
11.9905
11.934
11.9313
12.119
11.9677
12.0885
12.099
12.075
12.0877
11.9797
12.0615
```

13 Exercice 12 (polynômes)

```
In [47]: # Exercice 12, question a

         def legendre(n):
             p = Polynomial([-1, 0, 1])**n
             for k in range(1, n+1):
                 p = p.deriv()/(2*k)
             return p
```

```
In [48]: leg10 = legendre(10)
         leg10
```

```
Out[48]: Polynomial([ -2.46093750e-01,   0.00000000e+00,   1.35351562e+01,
                   0.00000000e+00,  -1.17304688e+02,   0.00000000e+00,
                   3.51914062e+02,   0.00000000e+00,  -4.27324219e+02,
                   0.00000000e+00,   1.80425781e+02], [-1.,  1.], [-1.,  1.])
```

```
In [49]: 256*leg10
```

```
Out[49]: Polynomial([ -6.30000000e+01,   0.00000000e+00,   3.46500000e+03,
                   0.00000000e+00,  -3.00300000e+04,   0.00000000e+00,
                   9.00900000e+04,   0.00000000e+00,  -1.09395000e+05,
                   0.00000000e+00,   4.61890000e+04], [-1.,  1.], [-1.,  1.])
```

Le but du calcul ci-dessus était de comparer la valeur trouvée avec la formule trouvée sur Wikipédia

$$L_{10} = \frac{1}{256}\left(46189X^{10} - 109395X^8 + 90090X^6 - 30030X^4 + 3465X^2 - 63\right).$$

```
In [50]: # Exercice 12, question b

         def legendre_suite(n):
             p0 = Polynomial([1])
             p1 = Polynomial([0, 1])
             x = p1
             if n == 0:
                 return p0
             else:
                 for k in range(1, n):
                     p2 = ((2*k+1)*x*p1 - k * p0) / (k+1)
                     p0, p1 = p1, p2
             return p1

In [51]: legs10 = legendre_suite(10)

In [52]: legs10

Out[52]: Polynomial([ -2.46093750e-01,   0.00000000e+00,   1.35351562e+01,
                   0.00000000e+00,  -1.17304688e+02,   0.00000000e+00,
                   3.51914062e+02,   0.00000000e+00,  -4.27324219e+02,
                   0.00000000e+00,   1.80425781e+02], [-1.,  1.], [-1.,  1.])

In [53]: leg10 == legs10

Out[53]: True
```

14 Annexe mathématique : démonstration de la conjecture de l'exercice 8

Notons χ_t le polynôme caractéristique de la matrice $M(t)$. Celui-ci est donné par le calcul suivant, où on développe selon la première colonne

$$\chi_t(\lambda) = \begin{pmatrix} \lambda - t & 0 & -1 \\ 0 & \lambda & -1 \\ -1 & -1 & \lambda \end{pmatrix} = (\lambda - t)(\lambda^2 - 1) - \lambda.$$

On obtient en particulier

$$\lim_{\lambda \to -\infty} \chi_t(\lambda) = -\infty, \quad \chi_t(-1) = 1, \quad \chi_t(1) = -1 \quad \lim_{\lambda \to +\infty} \chi_t(\lambda) = +\infty.$$

La continuité du polynôme χ_t permet d'appliquer le théorème des valeurs intermédiaires et d'affirmer que ce polynôme possède au moins une racine dans chacun des intervalles $]-\infty, -1[$,

$]-1,1[$ et $]1,+\infty[$. C'est un polynôme de degré 3 donc il n'a pas plus de trois racines. Il y a donc exactement une racine dans chacun de ces intervalles, ce qui donne

$$\lambda_1 < -1 < \lambda_2 < 1 < \lambda_3.$$

Le calcul donne $\chi_t'(-1) = 1 + 2t$. Ainsi, quand t est grand et h est petit, on trouve $\chi_t(1+h) \approx 1 + 2th$, ce qui va dans le sens de la conjecture $\lambda_1(t) \approx -1 - \frac{1}{2t}$.

L'expression exacte est

$$\chi_t(-1+h) = 1 + h(1+2t) - h^2(3+t) + h^3.$$

En particulier, on obtient

$$\chi_t\left(-1 - \frac{1}{1+2t}\right) = -\frac{3+t}{(1+2t)^2} + \frac{1}{(1+2t)^3} \overset{t\to+\infty}{\sim} -\frac{1}{4t}.$$

Ce nombre est donc strictement négatif si t est assez grand. Il existe donc t_0 tel que

$$\forall t \geq t_0, \quad \lambda_1(t) \leq -1 - \frac{1}{1+2t}.$$

De la même manière, on obtient

$$\chi_t\left(-1 - \frac{1}{2+2t}\right) = \frac{1}{2+2t} - \frac{3+t}{(2+2t)^2} + \frac{1}{(2+2t)^3} \overset{t\to+\infty}{\sim} \frac{1}{4t}$$

après un développement limité à la précision $o(\frac{1}{t})$. Ce nombre est donc strictement positif si t est assez grand. Il existe donc t_1 tel que

$$\forall t \geq t_1, \quad \lambda_1(t) \geq -1 - \frac{1}{2+2t}.$$

Le théorème des gendarmes donne alors que $(2t)(\lambda_1(t)+1)$ tend vers -1 quand t tend vers $+\infty$, ce qui se réécrit

$$\lambda_1(t) = -1 - \frac{1}{2t} + o\left(\frac{1}{t}\right).$$

Je ne détaille pas le raisonnement pour $\lambda_2(t)$ car c'est sensiblement pareil. Passons à $\lambda_3(t)$. On a vu qu'il doit être proche de t, ce qui suggère de développer $\chi_t(t+h)$.

$$\chi_t(t+h) = t^2h - t + 2th^2 - 2h + h^3.$$

On trouve en particulier

$$\chi_t\left(t + \frac{1}{t}\right) = \frac{1}{t^3}.$$

Ceci est strictement positif donc il existe t_2 tel que

$$\forall t \geq t_2, \quad \lambda_3(t) \leq t + \frac{1}{t}.$$

On trouve également

$$\chi_t\left(t + \frac{1}{t+1}\right) = -\frac{t}{1+t} - \frac{2t}{((1+t)^2} + \frac{1}{(1+t)^3}.$$

Ceci tend vers -1 quand t tend vers $+\infty$ donc c'est strictement négatif si t est assez grand. Il existe donc t_3 tel que

$$\forall t \geq t_3, \quad \lambda_3(t) \geq t + \frac{1}{t+1}.$$

Une autre application du théorème des gendarmes mène finalement, moyennant quelques étapes, au développement asymptotique

$$\lambda_3(t) = t + \frac{1}{t} + o\left(\frac{1}{t}\right).$$

In [ ]: