I Résolutions numériques approchées

Le problème que l'on cherche à résoudre ici est une équation ou un système d'équations différentielles, éventuellement faisant intervenir des dérivées partielles. Les inconnues sont donc en général des fonctions d'une (ou plusieurs) variable(s) réelle(s) et à valeurs réelles.

I.1 Principes généraux

Le principe fondamental est qu'une dérivée peut être approchée par un taux de variation, et inversement qu'une évolution d'une quantité peut être approchée à l'aide de la dérivée :

$$\boxed{y'(x) \simeq \frac{y(x+\Delta x)-y(x)}{\Delta x} \iff y(x+\Delta x) \simeq y(x) + \Delta x \times y'(x)} \text{ ou bien } y'(x) \simeq \frac{y(x)-y(x-\Delta x)}{\Delta x}$$

Pour une dérivée partielle, qui est la dérivée selon une seule variable, les autres étant fixées, on a de même :

$$\boxed{\frac{\partial y}{\partial x}(x,t) \simeq \frac{y(x+\Delta x,t)-y(x,t)}{\Delta x} \iff y(x+\Delta x,t) \simeq y(x,t) + \Delta x \times \frac{\partial y}{\partial x}(x,t).}$$

Pour approcher les dérivées d'ordre supérieur, on itère l'approximation précédente ; par exemple :

$$y''(x) \simeq \frac{y'(x+\Delta x)-y'(x)}{\Delta x} \simeq \frac{\frac{y(x+2\Delta x)-y(x+\Delta x)}{\Delta x} - \frac{y(x+\Delta x)-y(x)}{\Delta x}}{\Delta x} = \frac{y(x+2\Delta x)-2y(x+\Delta x)+y(x)}{(\Delta x)^2}.$$

Comme on l'a déjà vu pour le tracé de fonctions, la représentation informatique d'une fonction y d'une variable réelle se fait sous la forme d'une liste finie $x_0, \dots, x_n$ d'abscisses et d'ordonnées $y_0, \dots, y_n$ correspondantes. Pour une fonction de deux variables on utilisera un tableau fini $y_{0,0}, \dots, y_{n,m}$ de valeurs de y correspondant à un tableau fini à deux dimensions de valeurs, de variables x et t.

Discrétiser le problème posé revient à traduire les équations de départ sous forme approchée par des équations vérifiées par les éléments de ces listes finies.

Un schéma numérique est une présentation algorithmique de ces équations discrètes, i.e. sous une forme qui permet de calculer successivement les listes finies de valeurs y_i ou $y_{i,j}$ recherchées.

I.2 Rappels sur la méthode d'Euler explicite

Elle permet de résoudre de manière approchée un problème de Cauchy, par exemple du type :

$$\begin{cases} y'(t) = f(y(t),t) \\ y(t_0) = a \end{cases}$$

1. *Discrétisation* : on effectue une résolution sur l'intervalle $[t_0, t_0+T]$ que l'on découpe régulièrement en n parties bornées par $t_0, ..., t_n = t_0 + T$. On cherche donc les valeurs approchées correspondantes $y_0, ..., y_n$ de la solution du problème de Cauchy, i.e. en posant $\Delta t = \frac{T}{n}$:
$$\forall i \in [\![0, n[\![, \ \frac{y_{i+1}-y_i}{\Delta t} = f(y_i, t_i) \text{ et } y_0 = a$$
2. *Schéma numérique :* on part de $y_0 = a$ et on calcule successivement $y_1, \dots, y_n$ par :
$$y_{i+1} = y_i + \Delta t \times f(y_i, t_i).$$

Remarque : la méthode d'Euler implicite, plus délicate à mettre en œuvre, repose sur :

$$\frac{y_{i+1}-y_i}{\Delta t} = f(y_{i+1}, t_{i+1}).$$

II Exercices sur la résolution d'équations différentielles

Q1 Écrire une procédure `Euler1(f, a, t0, T, n)`, où `f` est une fonction de deux variables, où `a, t0, T` sont des réels et où `n` un entier supérieur à 2. Cette procédure affiche le graphique d'une approximation formée de $n+1$ points de la solution y définie sur le segment $[t_0, t_0+T]$ du problème de CAUCHY suivant :

$$\begin{cases} y'(t) = f(y(t), t) \\ y(t_0) = a \end{cases}$$

Résolution d'un système différentiel

Un système différentiel linéaire à coefficients et second membre constants de la forme :

$$(\mathcal{E}) : \begin{cases} x_1'(t) = a_{1,1}x_1(t) + \cdots + a_{1,n}x_n(t) + b_1 \\ \vdots \\ x_n'(t) = a_{n,1}x_1(t) + \cdots + a_{n,n}x_n(t) + b_n \end{cases}$$

se met sous forme matricielle $X'(t) = AX(t) + B$, avec $A = (a_{i,j})$ la matrice du système, $X(t) = \begin{pmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{pmatrix}$ la fonction vectorielle inconnue, et $B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$ le second membre du système.

Q2 Écrire un schéma numérique de résolution d'un tel système différentiel — avec la condition initiale $X(0) = X_0$ et un pas temporel h — qui exprime la valeur X_{i+1} au temps $(i+1)h$ de l'inconnue en fonction de la valeur X_i qui est sa valeur au temps ih.

Q3 Écrire une fonction `suivant(A, B, Xi, h)` où l'argument `Xi` est la valeur de X_i et qui renvoie X_{i+1}. Les matrices X_i, X_{i+1}, A et B seront réprésentées par des tableaux `numpy`.

Q4 Écrire une fonction `Euler(A, B, X0, h, N)` où `N` est le nombre d'itérations de la méthode d'EULER et qui renvoie le tableau `numpy` à n lignes et $N+1$ colonnes dont les colonnes sont $X_0, X_1, \ldots, X_N$.

Q5 Écrire les lignes de code nécessaires au calcul appproché de la solution sur $[0,1]$ du problème de CAUCHY suivant :

$$\begin{cases} x_1'(t) = 3x_1(t) + 4x_2(t) + 3 \\ x_2'(t) = 5x_1(t) - 2x_2(t) + 4 \end{cases} \text{ et } x_1(0) = 1,\ x_2(0) = -1,$$

et qui tracent sur le même graphique les deux courbes des fonctions x_1, x_2 correspondantes.

Résolution d'une équation différentielle d'ordre 2

Soit l'équation $(\mathcal{E})$: $y''(t) + a(t)\,y'(t) + b(t)\,y(t) = c(t)$ avec les conditions $y(t_0) = y0$ et $y'(t_0) = z0$.

Q6 Indiquer comment on ramène la résolution d'une telle équation à celle d'un système linéaire de la forme $X'(t) = A(t)\,X(t) + B(t)$ d'inconnue $t \mapsto X(t) = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}$ à valeur dans $\mathbb{R}^2$. En déduire un schéma numérique de résolution approchée quand t varie à partir de t_0 avec un pas h.

Q7 On suppose que la fonction a est seulement connue aux instants $t_0, t_1, \ldots, t_{N-1}$ (avec $t_i = t_0 + ih$) sous la forme de la liste `la=[`$a(t_0)$`,` $a(t_1), \ldots, a(t_{N-1})$`]`, et de même pour b et c donnée par `lb` et `lc`.
Écrire une fonction `Euler_second_ordre(y0, z0, h, la, lb, lc)` qui applique la méthode d'Euler et qui renvoie un tableau `numpy X` à deux lignes et $N+1$ colonnes tel que `X[0, i]` est une approximation de $y(t_i)$ et `X[1, i]` est une approximation de $y'(t_i)$.
Remarque : le tracé `plt.plot(X[0,:], X[1,:])` est alors une courbe plane qui s'appelle le portrait de phase de l'équation différentielle.

III Simulation numérique de la diffusion de la chaleur dans une barre

On note $T(x,t)$ la température à l'instant t et à l'abscisse x d'une barre en métal conductrice longiligne parfaitement isolée. Dans ce cas, la diffusion de la chaleur satisfait à l'équation aux dérivées partielles suivante (appelée équation de la chaleur) :

$$\frac{\partial^2 T}{\partial x^2} - \frac{1}{D}\frac{\partial T}{\partial t} = 0, \text{où } D \text{ est une constante physique}$$

.

III.1 Discrétisation spatiale et temporelle

Pour traiter numériquement le problème, la barre est subdivisée en N cellules de longueurs égales h (h, pas de longueur) et le temps est divisé en instants régulièrement répartis tous les k secondes (k, pas de temps).

Q8 Proposer une approximation de $\frac{\partial T}{\partial t}(x,t)$ qui s'exprime en fonction de $T(x,t+k)$ et $T(x,t)$.

Q9 Proposer une approximation de $\frac{\partial^2 T}{\partial x^2}(x,t)$ en fonction de $T(x-h,t)$, $T(x,t)$ et $T(x+h,t)$.

Q10 Pour simplifier les calculs, on suppose que les pas de longueur et de temps h et k sont choisis de manière que $h^2 = 6Dk$. Montrer que remplacer les dérivées partielles par les approximations précédentes permet d'obtenir une équation approchée de la forme :

$$T(x,t+k) \simeq aT(x-h,t) + bT(x,t) + cT(x+h,t) \quad (\mathcal{R}_{int}).$$

Q11 Par définition, sur les extrémités d'abscisses x_0 et x_{N-1} parfaitement isolées, le flux thermique est nul. Il en serait de même si chaque extrémité était prolongée par une seconde barre possédant une répartition de température symétrique de la première par rapport à cette extrémité. Justifier que cela revient à considérer que :

$$T(x_0,t+k) \simeq \frac{4T(x_0,t) + 2T(x_0+h,t)}{6} \text{ et } T(x_{N-1},t+k) \simeq \frac{2T(x_{N-1}-h,t) + 4T(x_{N-1},t)}{6} \quad (\mathcal{R}_{ext})$$

III.2 Mise en oeuvre informatique

On suppose que la barre est composée de $N = 2p+1$ cellules numérotées de 0 à $2p$ d'abscisses $x_j = x_0 + jh$, avec $j \in [\![0, N-1]\!]$ et l'on s'intéresse à la température de la barre lors de n instants successifs $t_i = t_0 + ik$ avec $i \in [\![0, n-1]\!]$. On applique le schéma numérique donné par les relations $(\mathcal{R}_{int})$ et $(\mathcal{R}_{ext})$. On utilise à cet effet un tableau `numpy T` de type à deux dimensions comportant n lignes et N colonnes, où `T[i,j]` contiendra la température $T(x_j, t_i)$ — attention à l'ordre inverse de i et j — de la cellule située à l'abscisse x_j à l'instant t_i.

Q12 On suppose connue la température de la barre à l'instant t_0 au moyen d'une liste `L` qui vaut $[T(x_0,t_0),\dots,T(x_N,t_0)]$. Écrire une fonction `initialise_tableau(L,n)` qui renvoie un tableau `T` de la bonne taille et dont les coefficients sont nuls exceptés ceux de la ligne 0 qui contient les températures initiales.

Q13 Écrire une procédure `ligne_suivante(T,i)` qui suppose que `T` contient les valeurs des températures simulées jusqu'à l'instant t_i (les autres valeurs étant nulles) et qui modifie le tableau `T` de manière qu'il contienne les valeurs des températures simulées jusqu'à l'instant t_{i+1}.

Q14 Écrire une fonction `simulation(L,n)` qui renvoie le tableau complet des températures simulées.

Q15 On suppose qu'au temps $t=0$ la température de la barre est homogène, sauf la cellule centrale dont on élève la température jusqu'à une certaine valeur.

1. Comment modifier le code précédent pour que la température soit représentée par un entier de type `np.uint16` ? Dans ce cas comment faut-il initialiser la simulation pour avoir la meilleure précision possible des résultats ?
2. Quelles commandes faut-il exécuter pour définir un tableau `numpy T` formés d'entiers de type `np.uint16` qui représente les températures simulées, avec la meilleure précision possible, pendant 1000 instants successifs, lorsque la barre est décomposée en 10 001 cellules. Quelle est alors la taille mémoire du tableau T ? Comparer avec l'espace en mémoire qui aurait été nécessaire avec des températures représentées par un `float`.

IV Exercices posé à l'oral du concours Centrale

Exercice 1. *(Centrale Python, PSI 2016)*
On considère l'équation différentielle $(1-x)\,y''=y$ *sur* $]-\infty,1[$.

1. *Montrer qu'il existe une et une seule solution* f *de cette équation vérifiant* $f(0)=0$ *et* $f'(0)=1$.
2. *Représenter la fonction* f *sur l'intervalle* $[-2;0,95]$.
3. *Soit* (a_n) *la suite définie par* $a_0=0$, $a_1=1$ *et :*

$$\forall n\in\mathbb{N}\backslash\{0,1\},\ a_n=\frac{n-2}{n}a_{n-1}+\frac{1}{n(n-1)}a_{n-2}.$$

   Représenter graphiquement a_n *pour* $n\in[0,100]$.
4. *Montrer que le rayon de convergence de* $\sum a_n x^n$ *est supérieur ou égal à* 1.
5. *Représenter graphiquement la fonction* $x\mapsto\sum_{n=0}^{100}a_k x^k$ *sur* $[-2;0,95]$.
6. *Que constate-on ? Démontrer le résultat.*

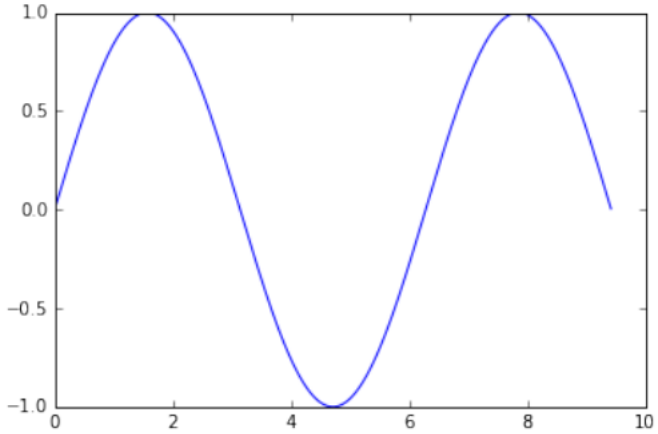
Extraits de la documentation fournies à l'oral du concours Centrale

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

Tracés de fonction

On définit une liste d'abscisses puis on construit la liste des ordonnées correspondantes. L'exemple ci-dessous trace $x \mapsto \sin x$ sur $[0, 3\pi]$.

```
def f(x) :
    return math.sin(x)

X = np.arange(0, 3*np.pi, 0.01)
Y = [ f(x) for x in X ]
plt.plot(X, Y)
plt.show()
```




```
import scipy.integrate as integr
```

Résolution approchées d'équations différentielles

Pour résoudre une équation différentielle $x' = f(x, t)$, on peut utiliser la fonction `odeint` du module `scipy.integrate`. Cette fonction nécessite une liste de valeurs de t, commençant en t_0, et une condition initiale x_0. La fonction renvoie des valeurs approchées (aux points contenus dans la liste des valeurs de t) de la solution x de l'équation différentielle qui vérifie $x(t_0) = x_0$. Pour trouver des valeurs approchées sur $[0, 1]$ de la solution $x'(t) = tx(t)$ qui vérifie $x(0) = 1$, on peut employer le code suivant.

```
def f(x, t) :
    return t*x

>>> T = np.arange(0, 1.01, 0.01)
>>> X = integr.odeint(f, 1, T)
>>> X[0]
array([ 1.])
>>> X[-1]
array([ 1.64872143])
>>> plt.plot(T,X)
>>> plt.show()
```

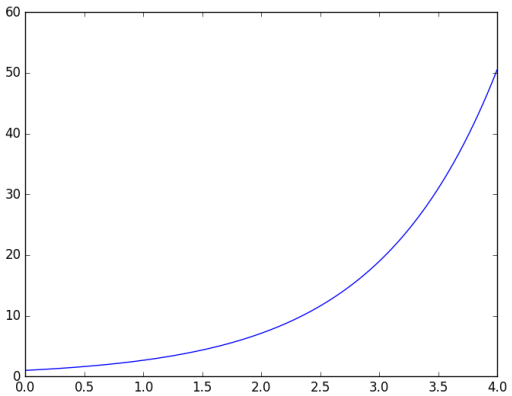
Corrigé

Q1

```
import matplotlib.pyplot as plt

def f(y, t):
    '''Fonction des deux variables y et t'''
    return y

def Euler1(f,a,t0,T,n):
    '''dans cette version, t est la liste des temps t[i] et y est la liste des ordonnées y[i]'''
    dt = T/n
    t = [t0 + i*dt for i in range(n+1)]
# NB : l'alimentation de t par t += dt dans une boucle for entraînerait des erreurs d'arrondis successifs
    y = [a]
    for i in range(n):
        y.append(y[i] + dt * f(y[i], t[i]))
# NB : on peut aussi écrire "for i in range(1,n+1):" suivi de "y.append(y[i-1]+dt*f(y[i-1],t[i-1]))"
    plt.plot(t,y)
    plt.show()

# ou

def Euler1B(f,a,t0,T,n):
    '''dans cette version, "temps" est la liste des temps t et "ordonnees" est la liste des ordonnées y'''
    dt = T/n
    temps = [t0 + i*dt for i in range(n+1)]
    y = a; ordonnees = [y]
# NB : le calcul qui suit nécessite une première valeur pour y
    for t in temps[:-1]:
# NB: il faut exclure la dernière valeur de temps, non utilisée pour le calcul de yn
        y = y + dt * f(y,t); ordonnees.append(y)
    plt.plot(temps,ordonnees)
    plt.show()
```

Exemple (où $f(y,t) = y$; la solution de $y' = y$ telle que $y(0) = 1$ est exp) :

`Euler1(f,1,0,4,100)` $\rightarrow$



Q2 Schéma numérique :
$X_{i+1} = X_i + h \times (AX_i + B)$.

Q3

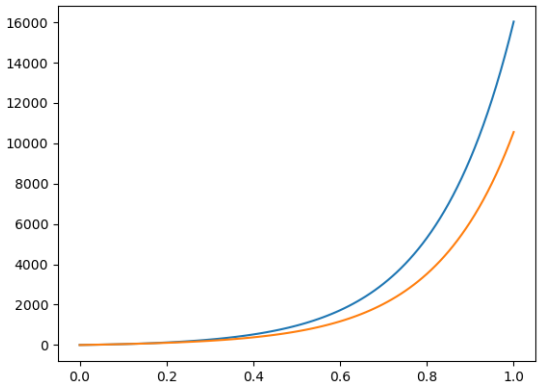
```
37 import numpy as np
38
39 def suivant(A, B, Xi, h):
40     return Xi + h * (np.dot(A, Xi) + B)
41 # * est la multiplication scalaire, + l'addition matricielle et np.dot
     la multiplication matricielle de numpy
```

Q4

```
44 def Euler(A, B, X0, h, N):
45     '''on suppose N >= 1'''
46     n = np.shape(A)[0]
47     X = np.zeros((n, N+1))
48     X[:, 0] = X0
49     for i in range(N):
50         X[:, i+1] = suivant(A, B, X[:,i], h)
51     return X
```

Q5

```
54 A = np.array([[3, 4], [5, -2]])
55 B = np.array([3, 4])
56 X0 = np.array([1, -1])
57 t0, tN = 0, 1 #bornes du temps
58 N = 100
59 h = (tN - t0)/N
60 t = [t0 + i*h for i in range(N+1)]
61 X = Euler(A, B, X0, h, N)
62 X1, X2 = X[0], X[1]
63 plt.plot(t, X1)
64 plt.plot(t, X2)
65 plt.show()
```



Q6

$$y''(t) + a(t)y'(t) + b(t)y(t) = c(t) \iff \begin{pmatrix} y'(t) \\ y''(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -b(t) & -a(t) \end{pmatrix} \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix} + \begin{pmatrix} 0 \\ c(t) \end{pmatrix}$$
$$\iff X'(t) = A(t)X(t) + B,$$
$$\text{avec } A(t) = \begin{pmatrix} 0 & 1 \\ -b(t) & -a(t) \end{pmatrix} \text{ et } B(t) = \begin{pmatrix} 0 \\ c(t) \end{pmatrix}.$$

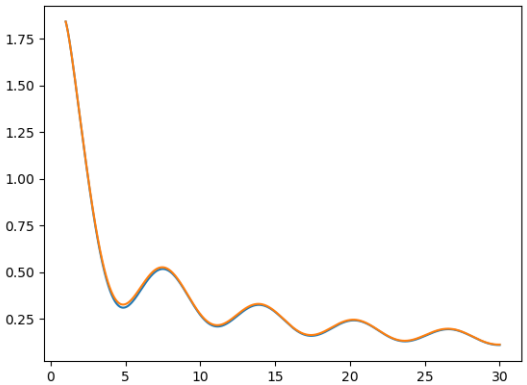
D'où le schéma numérique de résolution approchée, en notant $t_i = t_0 + ih$ et $X_i = X(t_i)$:

$$X_{i+1} = X_i + h \times (A_i X_i + B_i).$$

Q7

```
70 def suivant(Ai, Bi, Xi, h):
71     return Xi + h * (np.dot(Ai, Xi) + Bi)
72
73 def Euler_second_ordre(y0, z0, h, la, lb, lc):
74     N = len(la)
75     X = np.zeros((2, N+1))
76     X[:, 0] = np.array([y0, z0]) # il faut entrer une matrice ligne dans
       la colonne
77     for i in range(N):
78         Ai = np.array([[0, 1], [-lb[i], -la[i]]])
79         Bi = np.array([0, lc[i]])
80         X[:, i+1] = suivant(Ai, Bi, X[:, i], h)
81     return X
```

```
84 # Exemple d'équation d'Euler:
85 # (t**2)*y'' + (3*t)*y' + y = cos(t)- t * sin(t)
86 # dont la solution avec y0 = 1 + sin(1) et z0 = cos(1) - sin(1)
87 # est sin(t)/t + 1/t + ln(t)/t sur R+.
88 # On écrit : y'' +(3/t) * y' + (1/t**2) y = (cos(t)- t * sin(t)))/t**2
89 from numpy import cos, sin, log
90 t0, tN = 1, 30
91 N = 1000
92 h = (tN - t0)/N
93 temps = [t0 + i*h for i in range(N+1)]
94 la = [3/t for t in temps[:-1]]
95 lb = [1/t**2 for t in temps[:-1]]
96 lc = [(cos(t)-t*sin(t))/t**2 for t in temps[:-1]]
97 y0 = 1 + sin(1)
98 z0 = cos(1) - sin(1)
99
100 X = Euler_second_ordre(y0, z0, h, la, lb, lc)
101 plt.plot(temps, X[0])
102
103 Y = [(sin(t))/t+ (1/t) + (log(t))/t for t in temps]
104 plt.plot(temps, Y)
105
106 plt.show()
```



Q8 On approche la dérivée par le taux de variation : $\frac{\partial T}{\partial t}(x,t) \simeq \frac{1}{k}(T(x,t+k) - T(x,t))$.

Q9 De même, on a :

$$\begin{aligned}\frac{\partial^2 T}{\partial x^2}(x,t) &\simeq \frac{1}{h}\left(\frac{\partial T}{\partial x}(x,t) - \frac{\partial T}{\partial x}(x-h,t)\right)\\ &\simeq \frac{1}{h^2}\Big(T(x+h,t) - T(x,t)\Big) - \frac{1}{h^2}\Big(T(x,t) - T(x-h,t)\Big)\\ &= \boxed{\frac{1}{h^2}\Big(T(x+h,t) - 2T(x,t) + T(x-h,t)\Big).}\end{aligned}$$

Q10

$$\begin{aligned}\frac{\partial^2 T}{\partial x^2} = \frac{1}{D}\frac{\partial T}{\partial t} &\iff \frac{1}{h^2}\Big(T(x+h,t) - 2T(x,t) + T(x-h,t)\Big) \simeq \frac{6k}{h^2} \times \frac{1}{k}(T(x,t+k) - T(x,t))\\ &\iff T(x+h,t) - 2T(x,t) + T(x-h,t) \simeq 6(T(x,t+k) - T(x,t))\\ &\iff \boxed{T(x,t+k) \simeq \frac{1}{6}T(x+h,t) + \frac{2}{3}T(x,t) + \frac{1}{6}T(x-h,t).}\end{aligned}$$

Q11 Les hypothèses formulées reviennent à $T(x_0 - h, t) = T(x_0 + h, t)$ et $T(x_{N-1} - h, t) = T(x_{N-1} + h, t)$.
En remplaçant dans les relations $(\mathcal{R}_{int})$, on obtient bien les relations $(\mathcal{R}_{ext})$.

Q12

```
117 def initialise_tableau(L, n):
118     N = len(L)
119     T = np.zeros((n, N), dtype = np.uint16)
120     T[0] = L
121     return T
```

Q13

```
124 def ligne_suivante(T, i):
125     N = np.shape(T)[1]
126 
127     # relations Rext
128     T[i+1, 0] = (4*T[i, 0] + 2*T[i, 1]) / 6
129     T[i+1, N-1] = (2*T[i, N-2] + 4*T[i, N-1]) / 6
130 
131     # relations Rint
132     for j in range(1, N-1):
133         T[i+1, j] = (T[i, j-1] + 4*T[i, j] + T[i, j+1] ) / 6
134     # procédure : pas de return
```

Q14

```
137 def simulation(L, n):
138     T = initialise_tableau(L, n)
139     for i in range(n-1):
140         ligne_suivante(T, i)
141     return T
142 
143 N = 8
144 L = [0 for _ in range(N)]; L[N//2] = 2**16 - 1
145 assert (simulation(L, 10) ==
146 np.array([[    0,     0,     0,     0, 65535,     0,     0,     0],
147           [    0,     0,     0, 10922, 43690, 10922,     0,     0],
148           [    0,     0,  1820, 14563, 32767, 14563,  1820,     0],
149           [    0,   303,  3640, 15473, 26699, 15473,  3640,   606],
150           [  101,   808,  5056, 15371, 22957, 15371,  5106,  1617],
151           [  336,  1398,  6067, 14916, 20428, 14924,  6235,  2780],
152           [  690,  1999,  6763, 14359, 18592, 14393,  7107,  3931],
153           [ 1126,  2574,  7235, 13798, 17186, 13878,  7792,  4989],
154           [ 1608,  3109,  7552, 13268, 16070, 13415,  8339,  5923],
155           [ 2108,  3599,  7764, 12782, 15160, 13011,  8782,  6728]],
156       dtype=np.uint16)).all()
```

Q15

1. Dans la fonction `initialise(L,n)`, il suffit de remplacer
   `T = np.zeros([n, N])` par `T = np.zeros([n, N], dtype = np.uint16)`

   Lors de la simulation, la température d'aucune cellule ne dépassera la température de la cellule centrale, ni ne descendra en dessus de la température de départ des autres cellules (voir le cours de physique pour justifier cela rigoureusement). Pour avoir la meilleure précision possible il faut que cette température maximale soit représentée par la valeur maximale d'un entier `np.uint16`, soit $2^{16} - 1 = 65\,535$, et la température minimale par la valeur minimale d'un entier `np.uint16`, soit 0. D'où l'initialisation par `initialise(L, n)`, avec `L` de la forme $[0, 0, ..., 0, 0, 65\,535, 0, 0, ..., 0]$.

2. Le tableau `T` contient 1000 lignes et 10 001 colonnes et chaque cellule occupe 16 bits = 2 octets, d'où un espace mémoire occupé d'environ $\boxed{20 \text{ Mo.}}$

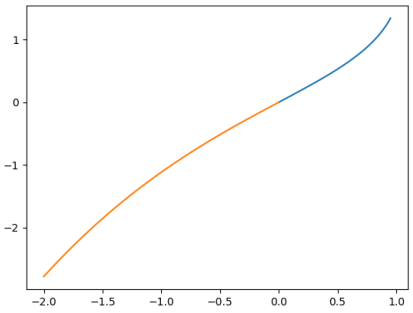
   Comme un flottant est codé sur 8 octets au lieu de 2 pour un entier `uint16`, l'utilisation de flottants nécessiterait un espace quatre fois plus important d'environ 80 Mo, d'où une économie de $\boxed{60 \text{ Mo}}$ en mémoire.

Solution de l'exercice

1. Sur $]-\infty, 1[$, l'équation s'écrit : $y'' - \dfrac{1}{(1-x)}y = 0$.
   Avec les conditions initiales données, on reconnaît donc un problème de Cauchy. D'après le théorème de Cauchy-Lipschitz il existe une unique solution.
2. Les conditions initiales sont données en 0, donc on part de 0 pour la résolution de l'équation différentielle (comme on l'a fait ci-dessus avec `Euler_second_ordre`), une fois à droite et une fois à gauche.

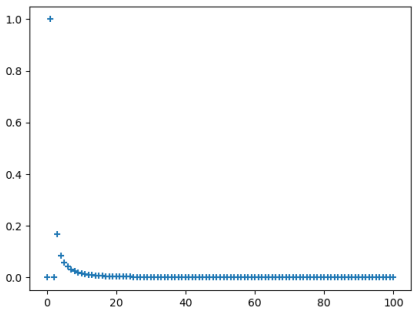
   Représentation de la solution f :

```
def f(X,t): # fonction de l'equa diff X' = f(X,t)
    '''Avec X = [y, y'], on a X' = [y', y''] = [X[1], 1/(1-t)*X[0]]'''
    return [X[1], 1/(1-t)*X[0]]

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

# Intervalle [0, 0.95] avec les conditions initiales en 0
N = 100
liste_t = np.linspace(0, 0.95, N)
X0 = np.array([0,1]) # donné dans l'énoncé
sol1 = odeint(f, X0, liste_t)

plt.figure()
plt.plot(liste_t, sol1[:, 0]) # sol est un np.array (100,2) de première
    colonnes: les valeurs de y, de seconde: les valeurs de y'

# Intervalle [0, -2] avec les conditions initiales en 0
liste_t = np.linspace(0, -2, N)
X0 = np.array([0,1]) # donné dans l'énoncé
sol2 = odeint(f, X0, liste_t)
plt.plot(liste_t, sol2[:, 0])
plt.show()
```



3. Représentation des a_n :

```
liste_n = [n for n in range(101)]
a, b = 0, 1

liste_a = [a, b]
for n in range(2, 101):
    a = ((n-2)/n) * b + (1/(n*(n-1)))* a
    liste_a.append(a)
    a, b = b, a

plt.scatter(liste_n,liste_a, marker = '+') # on constate 0 <= an <= 1
plt.show()
```

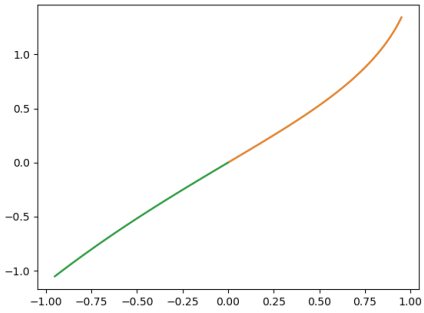


4. Montrons par récurrence sur $n \geq 1$ que $0 \leq a_n \leq 1$.

   C'est vrai pour $n \in \{0, 1\}$.
   Soit $n \geq 2$. Supposons $0 \leq a_{n-2} \leq 1$ et $0 \leq a_{n-1} \leq 1$.
   On a $a_n = \dfrac{n-2}{n} a_{n-1} + \dfrac{1}{n(n-1)} a_{n-2} \leq \dfrac{n-2}{n} + \dfrac{1}{n(n-1)} \leq \dfrac{n-2}{n} + \dfrac{1}{n} < 1$ car $n - 1 \geq 1$ et $a_n \geq 0$ donc $0 \leq a_n \leq 1$.
   Posons $b_n = 1$. La série entière $\sum b_n x^n$ est de rayon 1 et $0 \leq a_n \leq b_n$ donc le rayon de convergence de $\sum a_n x^n$ est supérieur ou égal à 1 (théorème de comparaison sur les séries entières).

5. Représentation de $S_{100}(x)$ sur $[-0,95, 0,95]$:

```python
def S100(x):
    S = 0
    for i in range(N+1):
        S = S + liste_a[i] * x**i
    return S

liste_t = np.linspace(-0.95, 0.95, N)
liste_S = [S100(x) for x in liste_t]

plt.figure()
plt.plot(liste_t, liste_S)

# Comparaison avec la solution de l'équa diff
liste_t = np.linspace(0, 0.95, N)
plt.plot(liste_t, sol1[:, 0])

# Intervalle [0, -0.95] avec les conditions initiales en 0
liste_t = np.linspace(0, -0.95, N)
X0 = np.array([0,1])
sol2 = odeint(f, X0, liste_t)
plt.plot(liste_t, sol2[:, 0])

plt.show()
```



6. On constate que les tracés se superposent sur $[-0,95;0.95]$, mais divergent ensuite l'un de l'autre rapidement quand la borne gauche est plus petite que -1.

   Soit $R \geq 1$ le rayon de convergence de la série entière $S(x) = \sum a_n x^n$.
   Montrons que, pour tout $x \in]-R;R[$, on a $f(x) = S(x)$.

   Pour tout $n \geq 0$, $n(n-1)a_n - (n-2)(n-1)a_{n-1} = a_{n-2}$ donc si $|x| < R$, on a :

$$\sum_{n=2}^{+\infty} n(n-1)a_n x^{n-2} - \sum_{n=2}^{+\infty} (n-2)(n-1)a_{n-1}x^{n-2} = \sum_{n=2}^{+\infty} a_{n-2}x^{n-2}$$

   (toutes les séries ont même rayon que $\sum a_n x^n$).
   Or $\sum\limits_{n=2}^{+\infty} n(n-1)a_n x^{n-2} = f''(x)$;
   $\sum\limits_{n=2}^{+\infty} (n-2)(n-1)a_{n-1}x^{n-2} = x\sum\limits_{n=1}^{+\infty} (n-1)na_n x^{n-1} = x\sum\limits_{n=1}^{+\infty} n(n-1)a_n x^{n-2} = xf''(x)$
   $\sum\limits_{n=2}^{+\infty} a_{n-2}x^{n-2} = f(x)$
   Ainsi, la série entière vérifie l'équation différentielle donnée sur $]-R,R[$.
   Comme $S(0) = a_0 = 0$ et $S'(0) = a_1 = 1$, on en déduit par unicité du développement en série entière, que $S = f$.
   Ceci explique la superposition des tracés constatés. Comme $R \geq 1$, on est d'ailleurs assuré de la superposition des tracés sur tout segment de $]-1,1[$.