Université Moulay Ismaïl
Département de Mathématiques - FST Errachidia

Filière : LST Mathématiques Appliquées S6
(Analyse Numérique)

Travaux pratiques

Module : M617
Analyse numérique des équations aux dérivées partielles

Jawad SALHI

Année Universitaire : 2022-2023

Table des matières

CHAPITRE 1

Série de Travaux-Pratiques 1 : Boucles et listes

Exercice 1 (Parcours de listes)
On travaille uniquement sur des listes d'entiers. On suppose la variable L affectée, et que celle-ci contient une liste de nombres non vide. Vous pouvez chosir n'importe quoi, comme par exemple la liste $L = [0, -1, 3, 5, 9, 11, 2, 1, 5]$.

1. Écrire une suite d'instructions permettant d'affecter à la variable somme_liste la somme des éléments de la liste.
2. Écrire une suite d'instructions permettant d'affecter à la variable max_liste l'élément maximal se trouvant dans la liste.
3. Écrire une suite d'instructions permettant d'affecter à la variable indice_max_liste l'indice de l'élément maximal se trouvant dans la liste (c'est-à-dire l'élément i tel que $L[i]$ est maximal. Si l'élément maximal se trouve à plusieurs endroits, indice_max_liste devra contenir le premier indice où ce maximum est rencontré).
4. On dit qu'une liste L de longueur n est croissante si pour tout i entre 0 et $n-2$, on a $L[i+1] \geq L[i]$. Écrire une suite d'instructions permettant d'affecter à la variable est_croissante un booléen (True ou False), ce booléen valant True si et seulement si la liste est croissante.

Testez vos codes, y compris avec d'autres listes !

Exercice 2 (Création de listes)

1. De deux manières différentes, créer une liste L0 de taille 100 ne contenant que des zéros ;
2. De même, créer la liste L100 contenant tous les entiers entre 0 et 99, dans cet ordre, de plusieurs manières :
   — avec list ;
   — avec une boucle for et append.
3. Créer la liste L300 contenant tous les entiers de 0 à 99, trois fois : le début est $[0, 0, 0, 1, 1, 1, 2, \cdots]$. On utilisera deux boucles for imbriquées et append.

4. Créer une liste qui contient tous les entiers de 0 à 99, puis ceux de 98 à 0 (elle a donc 199 éléments).
5. Créer par compréhension la liste des puissances de 2, de 0 à 20.
6. Créer par compréhension la liste des entiers inférieurs à 100 divisibles par 3 ou par 5, mais pas par 15.

Exercice 3 (De la base 10 à une base quelconque)
Écrire en Python un algorithme, permettant de calculer les chiffres d'un entier N dans la base b par divisions successives. On rappelle que // permet d'obtenir le quotient dans une division euclidienne, et % le reste. Avec $N = 158$ et $b = 7$, vous devez obtenir $[4, 1, 3]$, alors qu'en base 2 vous devez avoir $[0, 1, 1, 1, 1, 0, 0, 1]$: les chiffres sont stockés du poids faible au poids fort.

Exercice 4
Inversement, écrire un code permettant de passer d'une liste L représentant un entier dans la base b à son interprétation en base 10. Testez-le !

Exercice 5
Construire pour l'exemple une liste L contenant uniquement des entiers (pas forcément tous positifs, comme $L = [0, -1, 3, 5, -9, 11, 2, 1, 5]$).

1. À l'aide d'une boucle for, écrire une séquence d'instructions affichant à l'écran le nombre de zéros et de uns. On utilisera print("le nombre de zéros est : ", nb_zeros) pour l'affichage, avec nb_zeros une variable contenant le nombre de zéros, de même pour le nombre de uns. Modifier L si besoin pour vérifier.
2. L'expression Python x in L s'évalue un booléen (True ou False) indiquant si x est dans L. Écrire un morceau de code à l'aide d'une boucle while permettant de calculer le plus petit entier naturel non présent dans L. Modifier L si besoin pour vérifier.

Exercice 6
Écrire un code permettant de tester si un entier p stocké dans la variable du même nom est premier : il suffit de tester la divisibilité de p par tous les entiers entre 2 et $p - 1$. Affecter le résultat (True ou False dans une variable booléenne b). Tester avec 65521 (premier) et 96709 (non premier).

Exercice 7
La méthode du crible d'Eratosthène permet de calculer efficacement tous les entiers premiers strictement inférieurs à un entier $N > 2$. Pour ce faire :

- on crée une liste L de booléens de taille N, initialement tous True.
- on décoche $L[0]$ et $L[1]$ (c'est-à-dire qu'on affecte False à ces positions dans la liste) : ils ne sont pas premiers.
- on effectue ensuite une boucle sur les indices entre 2 et $N - 1$:
  - $L[2]$ est True, donc 2 est premier. On peut « décocher » tous les multiples non triviaux de 2 (4, 6, 8, etc...) : ils ne sont pas premiers.
  - $L[3]$ est True, donc 3 est premier. On peut « décocher » tous les multiples non triviaux de 3 (6, 9, 12, etc...) : ils ne sont pas premiers.
  - $L[4]$ est False, donc 4 n'est pas premier.
  - $L[5]$ est True, donc 5 est premier. On peut « décocher » tous les multiples non triviaux de 5 (10, 15, 20, etc...) : ils ne sont pas premiers.

— etc...

À l'aide de cette méthode, construire la liste de tous les entiers premiers inférieurs à 103 (vous devez en trouver 168). Combien y a-t-il d'entiers premiers inférieurs à 10^6 ?

Exercice 8

Une sous-séquence croissante d'une liste L est une portion d'éléments contigus, qui sont dans l'ordre croissant. Par exemple dans la liste $[0, 2, 8, 1, 5, 7, 9, 11, 2, 4, 6, 0, 1]$, la portion $[0, 2, 8]$ est une sous-séquence croissante. Écrire un morceau de code permettant de calculer la taille d'une sous-séquence croissante maximale (dans cet exemple, il s'agit de $[1, 5, 7, 9, 11]$, de taille 5).

CHAPITRE 2

Série de Travaux-Pratiques 2 : La notion de fonction

Exercice 1 (Premières fonctions)
Sans utiliser le module math, la fonction abs, la fonction pow ou encore l'opérateur $**$:

1. définir une fonction **absolue** prenant en entrée un entier et retournant sa valeur absolue (on ne demande pas de vérifier que le paramètre passé à la fonction est un entier).
2. définir une fonction **fact** prenant en entrée un entier positif et retournant sa factorielle définie par $n! = \prod_{i=1}^{n} i$.
3. définir une fonction **puissance** prenant deux arguments x et n et retournant x^n. (On supposera que n est un entier positif, et on posera $x^0 = 1$ pour tout réel x).

Exercice 2 (Utilisation d'une fonction dans une autre fonction.)

1. Écrire une fonction **max**2(a, b) prenant en paramètre deux entiers et retournant le maximum des deux. Remarque : cette fonction existe déja en Python sous le nom **max**, on ne l'utilisera pas ici, on comparera simplement a et b.
2. En déduire une fonction **max**3(a, b, c) retournant le maximum de 3 entiers, et utilisant **max**2. Remarque : la fonction **max** de Python peut également être utilisée à la place de **max**3.
3. Écrire une fonction **max_liste**(L) prenant en entrée une liste non vide et retournant son maximum, à l'aide de **max**2. Remarque : la fonction **max** de Python fonctionne également sur les listes.

Exercice 3 (Somme des chiffres d'un entier)
Écrire une fonction **sdc**(n) calculant la somme des chiffres de l'entier naturel n passé en paramètre. On utilisera des divisions euclidiennes par 10 pour obtenir les chiffres un par un.

Exercice 4 (Création de listes)

1. Écrire une fonction **carres**(n) qui renvoie la liste des n premiers carrés (de 0^2 à $(n-1)^2$).
2. Écrire une fonction **non_div_3**(n) qui renvoie la liste des entiers de qui ne sont pas divisibles par 3.
3. Écrire une fonction **addition**(L1,L2) qui prend en paramètres deux listes supposées de même taille, et qui renvoie la liste obtenue par addition terme à terme des éléments de $L1$ et $L2$.
   Attention, votre fonction ne doit pas modifier les listes passées en paramètre, mais créer une nouvelle liste.

Exercice 5 (Le juste prix)
On va créer un jeu du juste prix, aussi connu sous le nom de « C'est plus, c'est moins ». Le principe est le suivant : l'ordinateur choisi un nombre au hasard entre 1 et 1000. Vous essayez de deviner ce nombre, et pour ce faire, vous proposez des entiers au clavier. À chaque proposition, l'ordinateur vous dit si l'entier qu'il a choisi est plus grand ou plus petit que votre proposition. Le jeu s'arrête lorsque vous donnez l'entier choisi, avec un message de félicitations.

1. Tout d'abord, votre programme doit choisir un entier au hasard. On va pour cela importer la fonction randint du module random.
   Écrire une fonction **nombre_au_hasard**() (ne prenant pas d'argument) qui renvoie un entier aléatoire entre 1 et 1000.
2. Programmez maintenant le jeu proprement dit, sous la forme d'une fonction **juste_prix**() (ne prenant pas d'argument). Pour demander à l'utilisateur de donner un nombre, vous pouvez utiliser la fonction **input**. Cette fonction renvoie ce que l'utilisateur tape au clavier, sous forme de chaîne de caractères. Pour obtenir un entier, il faut convertir cette chaîne via la fonction **int**(). On peut placer un message comme paramètre à **input**(), sous forme de chaîne de caractères. Le code suivant convertit donc ce que l'utilisateur entre au clavier en entier et le stocke dans la variable n :

   $$n = \text{int}(\text{input}(\text{"Un nombre entre 1 et 1000 ? "}))$$

3. Rajouter la fonctionnalité « Rejouer ? », une fois que l'utilisateur a trouvé le nombre. S'il répond « oui » ou « Oui », on recommencera, sinon on arrêtera.

Exercice 6 (Un algorithme de tri)
On s'attaque maintenant à un problème un peu plus conséquent : le problème du tri d'une liste. On dit qu'une liste est triée si ses éléments sont dans l'ordre croissant. Écrire une fonction **tri_selection**(L) permettant de trier la liste L. Votre fonction ne renverra rien, mais modifiera la liste passée en paramètre. Remarque : la méthode **sort** de Python permet de trier une liste, elle s'utilise comme suit : **L.sort**().

Exercice 7 (Plus longue section croissante)
Écrire une fonction **plus_longue_section_croissante** ayant pour argument une liste d'entiers et qui renvoie la longueur d'une plus grande section croissante (c'est-à-dire d'éléments successifs) que l'on peut extraire de la liste ainsi qu'une telle section.

Remarque : si a et b sont deux objets, (a, b) est le couple formé de ces deux objets, et **return (a,b)** permet de retourner un tel couple dans une fonction. Les parenthèses sont en fait facultatives et **return a,b** produit le même résultat. Les couples (et plus généralement les n-uplets) sont très similaires aux listes mais sont immuables : on ne peut modifier une composante ou rajouter un élément ; on doit créer un nouveau couple.

Exercice 8 (Plus longue sous-séquence croissante)
Reprendre l'exercice 7 avec la plus longue sous-séquence croissante. Le principe est le même, mais on n'impose plus que les éléments soient contigus. On cherchera une solution la plus efficace possible.

CHAPITRE 3

Série de Travaux-Pratiques 3 : Les modules scientifiques

3.1 Numpy

Rappels :

- array : le type (optimisé) de tableaux/matrice. Un peu mieux (plus rapide) que des listes de listes... mais mono-type.
- numpy.linalg : opérations courantes en algèbre linéaire.
  - Création d'un tableau : $A = \text{array}([[1,2],[3,4]])$
  - Produit matriciel : $C = \text{linalg.dot}(A, B)$ ou bien $C = A.\text{dot}(B)$
  - Inversion : $D = \text{linalg.inv}(C)$
  - Résolution de $AX = Y$: $X = \text{linalg.solve}(A, Y)$
  - Transposition, éléments propres,... : voir cours python !

Exercice 1

Soit H_n la matrice de Hilbert d'ordre n, de coefficients $h_{ij} = \frac{1}{i+j-1}$ pour $1 \leq i, j \leq n$.

1. Créer une fonction hilbert d'argument n qui renvoie la matrice de Hilbert d'ordre n sous forme de tableau numpy.
2. Calculer le déterminant et l'inverse de la matrice de Hilbert H_n pour $2 \leq n \leq 5$. Donner également les valeurs et vecteurs propres de ces matrices.

3.2 Scipy

Rappels :

- Pour intégrer, on pourra utiliser scipy.integrate.quad
- Pour résoudre $f(x) = 0$, on dispose (entre autres) de scipy.optimize.newton, scipy.optimize.fsolve
- Les équations différentiellles seront résolues numériquement via scipy.integrate.odeint

Exercice 2

Évaluer $\int_0^1 \frac{\sqrt{\cos(t)}}{1+t^2}\, dt$.

Exercice 3
Évaluer les deux plus petites solutions dans $\mathbb{R}^+$ de l'équation :

$$e^{x/5} - 2\cos(x) = 2.$$

3.3 Matplotlib

Rappels :
- En général, on commence par : **import** matplotlib.pyplot **as** plt
- Principe : plot($\text{les}_\text{x}, \text{les}_\text{y}$). Les arguments sont des listes/array d'abscisses et d'ordonnées
- Il y a des tas de petites choses optionnelles : voir cours python.
- Prmière chose à apprendre : créer des listes/array d'abscisses et d'ordonnées : **arange, linspace,** [$\cdots$ **for** $\cdots$ **in** $\cdots$]

Exercice 4
Représenter la spirale d'équation paramétrique polaire $r = e^{\lambda t}$ avec $\lambda = \dfrac{\ln(2)}{2\pi}$.

Exercice 5
Trouver les coefficients de l'unique polynôme de degré au plus 3 interpolant $f : x \mapsto e^{x/5} - 2\cos(x)$ aux points d'abscisses $0, 2, 4, 6$. Représenter ensuite ce polynôme et f sur $[-2, 8]$.

Exercice Complémentaire

1. Implémenter l'algorithme du pivot de Gauss pour résoudre $AX = Y$, ou pour inverser A, ou encore calculer le déterminant d'une matrice...
2. Écrire un programme réalisant la décomposition LU d'une matrice dont les mineurs principaux sont non nuls.
3. Faire programmer la méthode de Newton (la fonction et sa dérivée étant fournies).
4. Faire programmer la méthode de dichotomie pour trouver une solution de $f(x) = 0$ sur $[a, b]$, lorsque f est continue sur $[a, b]$ et vérifie $f(a)f(b) \leq 0$.

CHAPITRE 4

Série de Travaux-Pratiques 4 : Résolution numérique des EDO

Exercice 1 (Schéma d'Euler pour la résolution numérique d'une EDO)
On se propose d'approcher numériquement les solutions des EDO s'écrivant sous la forme

$$\frac{dy(t)}{dt} = f(t, y) \text{ avec } y(0) = y_0, \tag{4.1}$$

à l'aide de la méthode d'Euler explicite (étudiée en cours) qui s'écrit sous la forme

$$y_{n+1} = y_n + h_n f(t_n, y_n) \text{ pour } 0 \leq n \leq N$$

où y_n est une approximation de $y(t_n)$, la suite $t_0, t_1, \cdots, t_N$ est une subdivision de l'intervalle $[t_0, t_f]$ et $h_n = t_{n+1} - t_n$ est le pas de temps.

1. On va mettre en oeuvre le schéma à l'aide d'une fonction PYTHON dont les arguments d'entrée sont quasi-identiques à ceux d'*odeint* et les arguments de sortie identiques. Écrire une fonction *euler* dont les 3 arguments en entrée sont f, $y0$ et vt et dont l'argument en sortie est vy où
   — f est une fonction de t et y définissant le système différentiel par un scalaire si problème scalaire, par un tableau de type *array* si problème vectoriel. **Remarque** : L'ordre des arguments de f n'est donc pas le même que celui d'*odeint*.
   — $y0$ contient la condition initiale (scalaire si problème scalaire, tableau de type *array* si problème vectoriel),
   — vt est un vecteur de type *array* contenant la subdivision $t_0, t_1, \cdots, t_N$,
   — vy est un tableau de type *array* dont le nombre de colonnes est la dimension du problème : chaque colonne correspond à une composante des solutions approchées aux points de la subdivision, chaque ligne correspond à un point de la subdivision.

On rappelle que l'algorithme est le suivant :

Poser $y = y0$

Stocker y dans vy

Puis pour chaque valeur t de vt à l'exception de la derniere

faire $y = y + hf(t, y)$

puis stocker y dans vy

Retourner le tableau de type array correspondant à vy

Le stockage dans vy se fera sous forme de liste à l'aide de *append.vy* et sera transformé en tableau à l'aide de *array* au moment du retour de la variable.

2. On considère le modèle logistique décrivant la dynamique d'une population de la forme (4.1) avec $f(t, y) = ry(1 - \frac{y}{K})$. On rappelle que la solution exacte est donnée par :

$$y(t) = \frac{y_0 K}{y_0 - e^{-rt}(-K + y_0)}.$$

On prendra $r = 1$ et $K = 1$ pour faire les calculs.

Ecrire un script qui :
- définit et calcule avec $y_0 = 0,1$, $t_0 = 0$ et $t_f = 2$ et pour $h = 0,25$ la solution approchée par la méthode d'Euler,
- définit la fonction qui à (t, y_0) calcule en t la solution exacte,
- trace sur une même figure le graphe de la solution exacte et de la solution approchée ainsi obtenue (pour cette dernière on utilisera un marqueur).

3. Avec $y_0 = 0,1$, $t_0 = 0$ et $t_f = 2$,
- calculer pour $h = 1/2^k$ avec $k = 1, 2, \cdots, 10$ l'erreur $E_k = |y(t_N) - y_N|$. **Attention** : on prendra t_N et non pas t_f (erreurs d'arrondis). En toute rigueur, il faudrait calculer $E_k = \max_i |y(t_i) - y_i|$.
- Tracer avec une échelle logarithmique cette erreur en fonction de h.
- Montrer graphiquement que la méthode est d'ordre 1 en calculant et en traçant la droite de régression linéaire. Afficher le coefficient directeur de la droite de régression linéaire.

Exercice 2 (Schémas à un pas d'ordre supérieur pour la résolution d'une EDO)
Reprendre l'exercice 1 avec :

1. La méthode du point milieu :

$$p_1 = f(t_n, y_n), \quad p_2 = f(t_n + \frac{h_n}{2}, y_n + \frac{h_n}{2} p_1), \quad y_{n+1} = y_n + h_n p_2.$$

2. La méthode de Heun :

$$p_1 = f(t_n, y_n), \quad p_2 = f(t_n + h_n, y_n + h_n p_1), \quad y_{n+1} = y_n + \frac{h_n}{2}(p_1 + p_2).$$

3. La méthode RK4 :

$$p_1 = f(t_n, y_n), \quad p_2 = f(t_n + \frac{h_n}{2}, y_n + \frac{h_n}{2} p_1), \quad p_3 = f(t_n + \frac{h_n}{2}, y_n + \frac{h_n}{2} p_2)$$

$$p_4 = f(t_n + h_n, y_n + h_n p_3), \quad y_{n+1} = y_n + \frac{h_n}{6}(p_1 + 2p_2 + 2p_3 + p_4).$$

Exercice 3 (Modèle SIR)
On considère le système du modèle SIR (Susceptibles-Infectés-Rétablis) de Kermack & McKendrick, décrivant l'évolution d'une population touchée par une maladie infectieuse,

$$\begin{cases} S'(t) = -rS(t)I(t), \\ I'(t) = rS(t)I(t) - aI(t), \quad t > 0, \\ R'(t) = aI(t), \end{cases}$$

associé à la donnée initiale

$$S(0) = S_0, \quad I(0) = I_0, \quad R(0) = R_0.$$

On rappelle que les fonctions S, I et R représentent le nombre au cours du temps de personnes respectivement sains et susceptibles de contracter la maladie, infectées et rétablis et immunisés après avoir été malades, le réel r est le taux d'infection et le réel a est le taux de guérison. On utilisera les valeurs suivantes des paramètres du problème :

$$S_0 = 762, I_0 = 1, R_0 = 0, r = 0.00218, a = 0.44036, T = 14.$$

1. Calculer une solution approchée du système sur l'intervalle de temps $[0, T]$ en utilisant la méthode d'Euler explicite (on se servira de la fonction écrite dans l'exercice 1) et $N = 10^5$ pas de discrétisation. Représenter l'évolution des valeurs des approximations obtenues des nombres $S(t), I(t)$ et $R(t)$ au cours du temps.
2. On note N le nombre de pas de discrétisation employés. Vérifier théoriquement et numériquement que l'approximation obtenue vérifie, pour tout $0 \leq n \leq N$,
$$S_n + I_n + R_n = S_0 + I_0 + R_0.$$
3. En prenant comme solution de référence celle obtenue avec $N = 105$ pas de discrétisation, représenter (avec une échelle logarithmique) les erreurs au temps T trouvées en utilisant successivement $N = 10, 10^2, \cdots, 10^4$ pas de discrétisation. Retrouver alors l'ordre de convergence de la méthode en utilisant la fonction *polyfit* de **numpy**.
4. Répondre à la question précédente en utilisant la méthode de Runge-Kutta d'ordre quatre.

Exercice 4 (Problème raide et stabilité absolue)
On cherche à approcher numériquement la solution du problème de Cauchy

$$y'(t) = -50(y(t) - \cos(t)), \quad t > 0 \quad \text{et } y(0) = 0,$$

sur l'intervalle $[0, 2]$.

1. Utiliser la méthode d'Euler explicite pour résoudre le problème avec un pas de discrétisation de longueur constante, successivement choisie :
   — strictement supérieure à $0,04$,
   — comprise entre $0,02$ et $0,04$,
   — strictement inférieure à $0,02$.
   Qu'observe-t-on et quelle explication peut-on fournir ?

2. Effectuer les calculs précédents avec la méthode d'Euler implicite. Pour cela, écrire une fonction *euler_implicite* dans laquelle la méthode de Newton-Raphson est employée pour résoudre le système d'équations non linéaires (on ajoutera dans les arguments d'entrée la fonction donnant la dérivée par rapport à y de la fonction f définissant le membre de droite de l'équation différentielle, ainsi que des paramètres d'arrêt pour la méthode de Newton-Raphson). Relancer les simulations et conclure.

CHAPITRE 5

Série de Travaux-Pratiques 5 : Différences finies pour l'équation de Poisson 1D

Exercice 1 (Conditons aux limites de Dirichlet homogènes)
On s'intéresse à la résolution approchée, par la méthode des différences finies, d'un problème aux limites pour l'équation de Poisson en dimension 1 d'espace sur $\Omega =]a, b[$ $a, b \in \mathbb{R}$, avec conditions aux limites de Dirichlet homogènes

$$(P) \quad \begin{cases} -u''(x) = f(x), & \forall x \in]a, b[\\ u(a) = u(b) = 0, \end{cases} \tag{5.1}$$

où $f \in C^2(\mathbb{R})$ donnée.

Comme on l'a vu dans le cours, la méthode des différences finies pour le problème (P) consiste à, pour $N > 1$ entier positif donné, chercher des réels $u_0, \cdots, u_{N+1}$ vérifiant

$$(P_h) \quad \begin{cases} -\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = f(x_i), & i = 1, \cdots, N, \\ u_0 = u_{N+1} = 0, \end{cases} \tag{5.2}$$

où $h = \frac{b-a}{N+1}$ et $x_i = a + ih$, $i = 0, \cdots, N+1$.

On pose $U_h = (u_1, \cdots, u_N)$ et $F_h = (f(x_1), \cdots, f(x_N))$. Le problème discret (P_h) peut s'écrire de manière équivalente : pour $N > 1, N \in \mathbb{N}$, donné, trouver $(u_0, u_1, \cdots, u_N, u_{N+1}) \in \mathbb{R}^{N+2}$ tel que

$$(S) \quad \begin{cases} (S_h)\, A_h U_h = F_h, \\ u_0 = u_{N+1} = 0, \end{cases} \tag{5.3}$$

avec $A_h \in \mathcal{M}_N(\mathbb{R})$ la matrice

$$\mathrm{A}_h = -\frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix}.$$

Pour des questions pratiques on va résoudre le système (S_h) sous la forme équivalente

$$A_N U_h = h^2 F_h,$$

où $A_N \in \mathcal{M}_N(\mathbb{R})$ est la matrice $h^2 A_h$ (qui ne dépend plus explicitement de h ; rappelons que le pas h et N sont liés par la relation $h = \frac{(b-a)}{N+1}$.

On considère $a = 0$, $b = 1$, $f(x) = (2\pi)^2 \sin(2\pi x)$, de sorte que la solution du problème (P) est donnée par $u(x) = \sin(2\pi x)$.

1. Implementer en python une fonction $A(N)$ qui retourne la matrice A_N, pour un nombre de points N donné. Commandes utiles : *np.eye*$(N, k = 1)$, *np.eye*$(N, k = -1)$.
2. Soit $N = 49$. Obtenir la solution approchée U_h en résolvant le système linéaire $A_N U_h = h^2 F_h$, en utilisant la fonction de **numpy** *np.linalg.solve*. Tracer sur la même figure la solution approchée et la solution analytique.
3. On cherche à illustrer numériquement la convergence du schéma aux différences finies (P_h) : la solution $(u_0, \cdots, u_{N+1})$ du problème discret (P_h) approche la solution u du problème continu (P) au sens suivant :

   $$\lim_{h\to 0} \big(\max_{i=0,\cdots,N+1} |u(x_i) - u_i| \big) = 0.$$

   Pour $N = 2^k$, et donc pour $h = \frac{1}{2^k+1}$, $6 \le k \le 11$, calculer et représenter dans un même graphique la différence en valeur absolue entre la solution exacte aux points x_i et la solution approchée. Observer que cette différence décroit avec h.
4. Pour $h > 0$ fixé, on note

   $$e_h := \max_{i=0,\cdots,N+1} |u(x_i) - u_i|$$

   l'erreur global du schéma associée à une discrétisation de pas h. On a la convergence

   $$\lim_{h\to 0} e_h = 0,$$

   et cette convergence a lieu à l'ordre 2, dans le sens que la propriété suivante se vérifie : il existe une constante $C > 0$, indépendante de h, tel que

   $$e_h \le C h^2.$$

   Calculer, pour chaque valeur de h associée à $N = 2^k$, $2 \le k \le 11$, l'erreur globale en norme infinie entre la solution approchée et la solution exacte, définie par

   $$e_h := \max_{i=1,\cdots,N} |u(x_i) - u_i|.$$

Représenter dans une même figure e_h en fonction de h, ainsi qu'une droite de pente 1 et une droite de pente 2, en utilisant une échelle logarithmique. Les résultats obtenus doivent illustrer numériquement le fait que la méthode des différences finies est d'ordre 2 pour ce problème.

Exercice 2 (Conditions aux limites de Neumann)
On approche dans cet exercice le problème $-u'' + u = f$ dans $]0,1[$, avec conditions aux limites dites de Neumann homogènes :

$$\begin{cases} -u''(x) + u(x) = f(x), & \forall x \in]0,1[\\ u'(0) = u'(1) = 0. \end{cases}$$

On considère la même discrétisation que dans l'exercice 1 de l'intervalle $]0,1[$ et on effectue les approximations suivantes des dérivées de la solution u aux points 0 et 1 :

$$u'(0) \approx \frac{u(h) - u(0)}{h}, \quad u'(1) \approx \frac{u(1) - u(1-h)}{h}.$$

On cherche alors, pour $N > 1$ donné et $h = \frac{1}{N+1}$, des valeurs $u_0, \cdots, u_{N+1}$ approchant la solution exacte aux points $x_0, \cdots, x_{N+1}$, solutions du système linéaire

$$\begin{cases} -\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = f(x_i), & i = 1, \cdots, N, \\ \frac{u_1 - u_0}{h} = 0, \quad \frac{u_{N+1} - u_N}{h} = 0, \end{cases}$$

où $x_i = ih$ pour $0 \leq i \leq N+1$.

Le problème discret peut être écrit à nouveau sous la forme $(A_{N+2} + h^2 J_{N+2})U_h = h^2 F_h$ où A_{N+2} est une matrice de taille $(N+2) \times (N+2)$, J_{N+2} est une matrice diagonale de taille $N+2$, $U_h = (u_0, u_1, \cdots, u_N, u_{N+1}) \in \mathbb{R}^{N+2}$ et où $F_h = (0, f(x_1), \cdots, f(x_N), 0)$.

1. Donner l'expression de la matrice A_{N+2} et l'implémenter dans une fonction python.
2. En prenant $f(x) = \big((2\pi)^2 + 1\big)\cos(2\pi x)$ (et on a donc $u(x) = \cos(2\pi x)$, refaire les mêmes figures que dans l'exercice 1. Montrer que la convergence dans ce cas est d'ordre 1 mais pas d'ordre 2.
3. Proposer une discrétisation pour ce problème en utilisant une discrétisation d'ordre 2 des dérivés $u'(0)$ et $u'(1)$ et vérifier numériquement qu'on obtient effectivement une méthode d'ordre 2.

CHAPITRE 6

Série de Travaux-Pratiques 6 : Différences finies pour l'équation de la chaleur 1D

On s'intéresse à la résolution numérique du problème de valeurs initiales et aux limites pour l'équation de la chaleur, avec conditions aux limites de Dirichlet, sur un domaine $]a,b[\times]0,T[$:

$$\begin{cases} \frac{\partial u}{\partial t}(x,t) - \frac{\partial^2 u}{\partial x^2}(x,t) = f(x,t) & \text{pour } (x,t) \in]a,b[\times]0,T[, \\ u(a,t) = g_a(t),\ u(b,t) = g_b(t) & \text{pour } t \in]0,T[, \\ u(x,t=0) = u_0(x) & \text{pour } x \in]a,b[, \end{cases} \tag{6.1}$$

où $u_0 : [a,b] \to \mathbb{R}$ est une fonction de classe C^2 donnée, $g_a, g_b : [a,+\infty[\to \mathbb{R}$ sont des fonctions de classe C^1 données, vérifiant $g_a(0) = u_0(a)$, $g_b(0) = u_0(b)$.

Dans cette équation l'inconnue u représente par exemple la distribution de chaleur à l'instant t en chaque point x d'un matériau homogène conducteur de la chaleur, que l'on assimile à un domaine Ω de $\mathbb{R}^n$, $n = 1, 2$ ou 3 (ici pour simplifier on se restreindra à la dimension $n = 1$). La fonction f modélise une source de chaleur extérieure à laquelle est soumis le matériau.

On suppose en plus que la température est fixe au bord, ce qui se traduit par des conditions aux limites de Dirichlet, on aurait pu aussi supposer par exemple que le flux de chaleur au bord dans la direction normale est nul, ce qui se traduirait par une condition de Neumann.

Notre objectif est de calculer des valeurs approchées de la solution exacte u de (6.1) par la méthode des différences finies. Pour cela on introduit :

- pour $N > 0$ donné, une discrétisation uniforme de l'intervalle $[a,b]$ définie par les $N+2$ points $x_i = a + ih$, $i = 0, \cdots, N+1$, avec $h = \frac{b-a}{N+1}$ le pas de la discrétisation (aussi appelé pas d'espace) ;
- le pas de temps $k > 0$ et les instants temporels $t_n = nk$, $n \in \mathbb{N}$, $n \leq M+1$, avec $(M+1)k = T$.

On cherche alors des valeurs u_i^n approchant $u(t_n, x_i)$, pour $n \in \mathbb{N}$, $n \leq M+1$ et $i = 0, \cdots, N+1$. Comme la solution exacte u de (6.1) vérifie

$$u(x_i, 0) = u_0(x_i),$$

il est naturel de prendre $u_i^0 = u_0(x_i), i = 0, \cdots, N+1$.

On considèrera deux schémas pour calculer les valeurs approchées

$$(u_i^n)_{i=0,\cdots,N+1}, \text{ pour } n = 1, \cdots, M+1.$$

— **Le schéma explicite :**

$$\begin{array}{ll} \frac{u_i^{n+1}-u_i^n}{k} - \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} = f(x_i, t_n), & i = 1, \cdots N,\ n = 0, \cdots, M, \\ u_i^0 = u_0(x_i), & i = 0, \cdots, N+1, \\ u_0^{n+1} = g_a(t_{n+1}),\ u_{N+1}^{n+1} = g_b(t_{n+1}), & n = 0, \cdots, M. \end{array} \tag{6.2}$$

— **Le schéma implicite :**

$$\begin{array}{ll} \frac{u_i^{n+1}-u_i^n}{k} - \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{h^2} = f(x_i, t_{n+1}), & i = 1, \cdots N,\ n = 0, \cdots, M, \\ u_i^0 = u_0(x_i), & i = 0, \cdots, N+1, \\ u_0^{n+1} = g_a(t_{n+1}),\ u_{N+1}^{n+1} = g_b(t_{n+1}), & n = 0, \cdots, M. \end{array} \tag{6.3}$$

Nous allons comparer les résultats des deux schémas précédents dans le cas de l'équation de la chaleur homogène ($f = 0$), avec conditions aux limites de Dirichlet homogènes ($g_a = g_b = 0$). En particulier nous allons voir comment les deux schémas sont influencés par la valeur du paramètre

$$\lambda := \frac{k}{h^2}.$$

1. Écrire une fonction python DFchaleurExp$(a, b, M, T, lambda, u_0)$ qui calcule et retourne la discrétisation en espace x et la solution approchée U de (6.1) avec $f = g_a = g_b = 0$, à un temps final T pour une valeur λ fixée, obtenue par la méthode explicite. On prendra garde à ne stocker que la solution au temps final dans le programme. On n'oubliera pas de définir la valeur approchée en $x = a$ et en $x = b$, u_0^n et u_{N+1}^n.
2. En notant $U^n = (u_1^n, \cdots, u_N^n)$, montrer que le schéma implicite peut être mis sous la forme matricielle

   $$(\mathrm{I}_N + k\mathrm{A}_h)U^{n+1} = U^n,$$

   où la matrice A_h est à déterminer. Écrire une fonction python

   $$\text{DFchaleurImp}(a, b, M, T, lambda, u_0)$$

   qui calcule et retourne la discrétisation en espace x et la solution approchée U de (6.1) avec $f = g_a = g_b = 0$, à un temps final T pour une valeur λ fixée, obtenue par la méthode implicite.
3. On considère $u_0(x) = \sin(\pi x)$ et $[a, b] = [0, 1]$. Vérifiez que la solution de (6.1) est dans ce cas $u(t, x) = e^{-\pi^2 t}\sin(\pi x)$.
   (a) Pour $N = 49$, calculer la solution approchée donnée par la méthode explicite à l'instant $T = 2$, et représenter sur un même graphe la solution approchée et la solution exacte, en utilisant $\lambda = \frac{1}{4}$.
   (b) Que se passe-t-il si on considère $\lambda > \frac{1}{2}$?
   (c) Faire de même pour la méthode implicite. Quelles différences constatez-vous ?

4. Adaptez vos programmes pour prendre en compte un second membre non nul, des conditions aux limites non homogènes, des conditions de Neumann, homogènes ou non.