

Devoir 1

Directives

À déposer dans la boîte de dépôt sur le site du cours dans monportail : **deux fichiers Python (.py)**, intitulés `equipeXX.py` (remplacer XX par le numéro de votre équipe) et `suiteSn.py`, et **le rapport**.

Lorsqu'on exécute le fichier script `equipeXX.py`, tous les éléments demandés à la question 1 doivent s'afficher sur la fenêtre de commandes de votre IDE Python, et toutes les figures demandées doivent s'afficher, chacune dans une fenêtre (**5 figures au total**). Si les figures n'apparaissent pas à l'exécution du programme, les réponses correspondantes seront considérées comme nulles.

**Dans le rapport**, on mettra les **5 figures obtenues** (dans l'ordre et numérotées, avec leurs titres) et les réponses aux **questions posées**. Si les figures n'apparaissent pas ou ne coïncident pas avec celles qui apparaissent en exécutant le programme, les réponses seront considérées comme nulles.

On utilisera un éditeur de texte (Word, LaTeX, ...) pour le rapport. On ne remettra donc **pas de rapport manuscript** ou un rapport contenant en partie des notes manuscrites.

Prérequis

Vous aurez besoin des lignes suivantes au début de vos deux codes Python :

```
import numpy as np
```

De plus, vous aurez besoin de

```
from math import e
```

dans votre fichier `suiteSn.py`. Finalement, vous aurez besoin de

```
import matplotlib.pyplot as plt
from suiteSn import suiteSn
```

dans votre fichier `equipeXX.py`. Cette dernière ligne vous permettra d'utiliser dans votre fichier script la fonction que vous définirez dans la question 2.

## Questions

La première question permet de se familiariser avec les principales commandes Python qui seront utiles pour tous les devoirs à venir (incluant celui-ci). On pourra consulter le (court) guide de M. LeMesurier et M. Roberts (contributions) vers lequel un lien a été placé sur le site web du cours, dans la page d'introduction (c'est le premier guide Python suggéré dans la liste).

1. Créer un fichier script Python dont le nom contiendra le numéro de votre équipe (exemple : *equipe43.py*). Dans ce fichier, on commencera par écrire les lignes de commandes qui définiront les nombres, vecteurs et matrices suivants et qui les afficheront sur la fenêtre de commandes de l'IDE Python. Attention, sauf indication contraire, il faut 1 seule ligne de commandes pour chaque élément demandé.
  - $a$ , le vecteur colonne de longueur 6 dont toutes les composantes valent 1. Utiliser pour cela la commande Python *np.full*.
  - $b$ , le vecteur ligne de longueur 6 dont les composantes valent 1, 2, ..., 6 (en une seule ligne). Utiliser pour cela la commande Python *np.arange*.
  - $c$ , le vecteur ligne de longueur 6 dont toutes les composantes valent 1. On obtiendra  $c$  à partir de  $a$  grâce à la transposition (fonction *np.reshape* ou méthode *reshape*).
  - $d$ , le vecteur ligne de longueur 6 dont toutes les composantes valent le numéro de votre équipe. On obtiendra  $d$  en multipliant  $c$  par le numéro de votre équipe.
  - $I$ , la matrice identité de taille  $6 \times 6$ . On utilisera la commande *np.identity*.
  - $J$ , la matrice de taille  $6 \times 6$  dont toutes les composantes valent 1. Utiliser pour cela la commande *np.full*.
  - $K$ , la matrice de taille  $6 \times 6$  dont la diagonale est le vecteur  $b$ . Utiliser pour cela la commande *np.diag*.
  - $L$ , la matrice de taille  $6 \times 6$ , définie par  $L = 55I - J + 2a * c$ . On obtiendra  $L$  à partir de  $I$ ,  $J$ ,  $c$  et  $a$ , bien sûr.
  - $M$ , la matrice obtenue en remplaçant la 1ère colonne de  $K$  par  $a$ . On obtiendra  $M$  à partir de  $K$  et  $a$ , bien sûr, en commençant votre ligne de commande par :  $M=1*K; M[:,0]=...$  (à compléter, utiliser *reshape*).
  - $dd$ , le déterminant de  $M$  (utiliser la commande *np.linalg.det*).
  - $x$ , la solution de  $Mx = a$ , sans calculer l'inverse de  $L$  (utiliser la commande *np.linalg.solve*).
  - $N$ , la matrice solution de  $M * N = M'$ , où  $M'$  est la matrice transposée de  $M$  (utiliser la commande *np.linalg.solve* pour résoudre et la méthode *T* pour transposer).

Sur la fenêtre de commandes, on verra donc s'afficher (utiliser des *print*) :

$a =$

...

$b =$

...

et ainsi de suite ....

À la suite, dans ce fichier script, créer une figure (Figure 1) visualisant la matrice  $N$ . On utilisera la commande *plt.matshow*. On écrira en titre, grâce à la commande *plt.title*, qu'il s'agit de la matrice  $N$ .

Mettre la figure dans le rapport (*plt.show* pour afficher dans l'IDE Python).

Ensuite, toujours dans le fichier script, définir la fonction  $f(x) = -\frac{x^2}{2} + e^x + \sin x$  de façon à ce qu'elle permette d'être appliquée à un vecteur  $x$ , en utilisant la syntaxe (à compléter) :

```
def nom_fonction(variable_a_entrer) :
```

```
code
```

```
return variable_a_retourner
```

Rappel : il est possible de tout évaluer en une ligne en utilisant les fonctions *np.exp* et *np.sin* de Python.

Faire tracer le graphe de  $f$  sur l'intervalle  $[0, 1]$ , en évaluant la fonction aux abscisses  $0, 0.01, 0.02, \dots, 0.99, 1$ . On définira pour cela le vecteur  $x$ , contenant les abscisses des points, par

```
x = np.linspace(0, 1, 101)
```

Puis utiliser la commande *plt.plot(x, nom\_fonction(x))* pour tracer le graphe.

Mettre un titre à la figure avec la commande *plt.title*. Cela constituera la Figure 2.

Mettre la figure dans le rapport (*plt.show* pour afficher dans l'IDE Python).

## 2. Instabilité numérique

Soit  $S_n = \int_0^1 x^n e^x dx$ .

On aimerait calculer  $S_n$  quelle que soit la valeur de  $n$ . On voit bien que  $S_0 = e - 1$ . Mais pour des valeurs de  $n \geq 1$  il faut faire  $n$  intégrations par parties pour obtenir la valeur de  $S_n$ . Un autre moyen est d'utiliser une formule de récurrence et de programmer cet algorithme avec une boucle.

- a) On peut voir que  $S_n \geq 0$  quel que soit  $n$ . On peut également montrer que  $S_n \rightarrow 0$  lorsque  $n \rightarrow +\infty$ .

Dans le rapport, démontrer la formule de récurrence suivante en utilisant une intégration par parties :

$$S_n = e - nS_{n-1}.$$

- b) Créer une fonction Python, appelée *suiteSn*, dans un nouveau fichier *suiteSn.py*, dont la seule entrée est  $n$  et la sortie est un vecteur ligne (*np.array*)  $S$  de longueur  $n+1$ , dont les composantes sont les valeurs  $S_0, S_1, \dots, S_n$ . On utilisera une boucle *for* et la commande *np.append* pour construire les éléments de la suite  $S_n$  en utilisant la formule de récurrence précédente. Outre les *import*, cette fonction Python commencera par les 2 lignes suivantes :

```
def suiteSn(n) :  
    S = np.array([e - 1])
```

Dans le fichier script, faire appel à la fonction *suiteSn* créée précédemment pour calculer  $S_1, \dots, S_{19}$ . Dans une nouvelle figure (Figure 3), créer un graphique montrant les valeurs de  $S_n$  en fonction de  $n$ , pour  $n = 0, 1, \dots, 19$  (Figure 3). On utilisera la commande *plt.plot* pour le graphique. On utilisera également les commandes *plt.xlabel*, *plt.ylabel*, *plt.title* pour rendre la figure très explicite.

Mettre la figure dans le rapport (*plt.show* pour afficher dans l'IDE Python).

- c) Dans le rapport, expliquer pourquoi on obtient des valeurs surprenantes pour  $S_{18}$  et  $S_{19}$ . À cette fin on exprimera  $\Delta S_n$  en fonction de  $\Delta S_{n-1}$  en utilisant la formule de propagation de l'erreur vue en cours.

### 3. Opération dangereuse

Pour une fonction  $f(x)$  donnée, on sait qu'une approximation de  $f'(x)$  est donnée par la formule

$$D(x, h) = \frac{f(x+h) - f(x)}{h}$$

où  $h$  est supposé positif. Plus  $h$  est petit et plus  $D(x, h)$  est proche de  $f'(x)$ . On a d'ailleurs  $D(x, h) \rightarrow f'(x)$  lorsque  $h \rightarrow 0$ .

Dans le script Python, calculer l'erreur  $|f'(0) - D(0, h)|$ , pour la fonction  $f(x)$  de la question 1, et pour les valeurs suivantes de  $h$  :  $10^{-1}, 10^{-2}, \dots, 10^{-6}$  et faire tracer le graphique (Figure 4) de l'erreur en fonction de  $h$  avec des échelles logarithmiques (au lieu de la commande *plt.plot*, pour avoir des échelles logarithmiques à la fois en abscisse et en ordonnée, on utilisera la commande *plt.loglog*). Bien identifier les axes et mettre un titre au graphique (commandes *plt.xlabel*, *plt.ylabel*, *plt.title*).

Mettre la figure dans le rapport (*plt.show* pour afficher dans l'IDE Python).

Dans le rapport : expliquer pourquoi, alors qu'on s'attend à ce que  $D(x, h) \rightarrow f'(x)$  lorsque  $h \rightarrow 0$ , l'erreur devient bizarre pour les plus petites valeurs de  $h$ . (Indice : voir titre de cette question).

#### 4. Ordre d'un polynôme de Taylor

Dans le rapport, déterminer le polynôme de Taylor de degré 2, noté  $p_2(x)$ , de la fonction  $f(x)$  de la question 1, en  $x_0 = 0$ .

On sait d'après le cours qu'il correspond à une approximation d'ordre au moins 3 de  $f(x)$  au voisinage de  $x_0$ . Dans le rapport, expliquer en quelques lignes pourquoi il est en fait d'ordre 4.

Dans le script Python, calculer  $E(x) = |f(x) - p_2(x)|$  pour  $x = 10^{-1}, 5 \times 10^{-2}, 10^{-2}, 5 \times 10^{-3}, 10^{-3}$ , et faire tracer  $E(x)$  en fonction de  $x$  grâce à ces valeurs (Figure 5). On utilisera une échelle logarithmique grâce à la commande *plt.loglog*.

Bien identifier les axes et mettre un titre au graphique.

Mettre la figure dans le rapport (*plt.show* pour afficher dans l'IDE Python).

À la suite, dans le rapport, expliquer comment reconnaître l'ordre 4 de cette approximation. (Indice : Si  $E(x) \approx Cx^p$  alors  $\log(E(x)) \approx \log(C) + p \log(x)$ ).