L1 SPI – Algorithmique

Projet C01 – Dérivation numérique

Cyril Desjouy

15 juillet 2024

1 Introduction

Le but de ce projet est d'étudier la méthode des différences finies. Il s'agit d'une méthode simple et largement utilisée pour l'approximation de la dérivée $f'(x)$ d'une fonction $f(x)$ dérivable suivant x.

2 La méthode des différences finies

2.1 Principe

Il est possible d'approcher la fonction $f(x+h)$ par un polynôme qu'on appelle communément développement limité (ici à l'ordre 2) autour de la valeur $x+h$:

$$f(x+h) = f(x) + h \cdot f'(x) + \frac{h^2}{2} f''(x) + \mathcal{O}(h^2). \tag{1}$$

La manipulation de cette équation conduit à l'expression de la dérivée $f'(x)$ de la fonction $f(x)$ sous la forme suivante :

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h). \tag{2}$$

Afin de discrétiser le problème, un maillage suivant x peut être défini sous la forme :

$$x_i = i \cdot h, \text{ avec } h = \frac{x_{max}}{N_x}, \tag{3}$$

où x_{max} est la valeur maximale admise par la variable x et N_x est le nombre de points suivant x. Aux nœuds de ce maillage la fonction $f(x)$ est approchée par le scalaire $f_i = f(x_i)$. Cette discrétisation permet alors d'écrire :

$$f'(x_i) = f'_i = \frac{f_{i+1} - f_i}{h} + \mathcal{O}(h) \tag{4}$$

La dérivée discrète $f'(x_i)$ s'exprime alors en fonction de la valeur de la fonction f au point d'indice i et de celle au point suivant, d'indice $i+1$, ce qui lui vaut l'appellation de formulation *décentrée avancée*. Suivant la même procédure il est possible d'écrire le développement limité de $f(x-h)$ qui conduit alors à une formulation de la dérivée discrète $f'(x_i)$ en fonction de la valeur de la fonction f au point i et de celle au point précédent, d'indice $i-1$:

$$f'(x_i) = f'_i = \frac{f_i - f_{i-1}}{h} + \mathcal{O}(h) \tag{5}$$

Cette formulation est classiquement appelée formulation *décentrée retardée*.

2.2 Implémentation

1. Créer un module `derivatives` proposant deux fonctions prenant toutes deux en arguments d'entrée un signal $s(x)$ et un vecteur x et retournant sa dérivée $s'(x)$ suivant x. Ces deux fonctions seront :
   - la fonction `ffdm01` utilisant des boucles `for` pour faire le calcul de la dérivée
   - la fonction `sfdm01` utilisant des slices pour faire le calcul de la dérivée
2. Tester la dérivation sur la fonction mathématique *cosinus*. Ce test ne s'exécutera que si le module est lu en tant que script, pas lorsque celui-ci est simplement importé. Comparer les résultats théoriques à ceux obtenus par dérivation numérique.
3. Évaluer dans un interpréteur Python les temps d'exécution de ces deux fonctions à l'aide de la fonction `timeit` [a] comme suit : `timeit ffdm01(x, y)`
4. En utilisant les fonctions précédemment développées, créer une fonction nommée `nderivative` prenant en arguments d'entrée un signal $s(x)$, un vecteur x, et un entier n, et retournant la dérivée n-ième d'un objet de type `ndarray` en notant que :
$$f''(x) = (f'(x))' \tag{6a}$$
$$f^{(3)}(x) = (f''(x))' \tag{6b}$$
$$f^{(4)}(x) = (f^{(3)}(x))' \tag{6c}$$
$$\dots$$
5. Tester cette fonction avec un cosinus défini entre 0 et 2π. Calculer et tracer la dérivée 3 de ce cosinus. Comparer les résultats numériques aux résultats théoriques. Conclure sur la précision des formulations décentrées d'ordre 1.

a. La fonction `timeit` est une fonction fournie par IPython (Jupyter). Elle est utilisable uniquement si elle est la seule instruction dans une cellule Jupyter.

2.3 Schémas décentrés à l'ordre 2

Les résultats précédent ont montrés que l'utilisation de schémas à l'ordre 1 ne suffit pas pour calculer correctement les dérivées secondes et supérieures d'une fonction. Le but de cette partie est donc d'améliorer les schémas différences finies utilisés dans la partie précédente pour qu'ils soient précis à l'ordre 2. La combinaison des équations 4 et 5 permet d'écrire la dérivée discrète $f'(x_i)$ en fonction de la valeur de la fonction f au point précédent (point d'indice $i-1$) et au point suivant (point d'indice $i+1$) :

$$f'(x_i) = f'_i \simeq \frac{f_{i+1} - f_{i-1}}{2h} + \mathcal{O}(h^2) \tag{7}$$

Ce type de formulation est appelée formulation *centrée*. En écrivant les développements limités adaptés, il est aussi possible d'exprimer les formulations décentrées avancées et retardées à l'ordre 2 comme suit :

$$f'(x_i) = f'_i \simeq \begin{cases} \dfrac{-3f_i + 4f_{i+1} - f_{i+2}}{2h} & \text{formulation avancée} \\ \dfrac{3f_i - 4f_{i-1} + f_{i-2}}{2h} & \text{formulation retardée} \end{cases} \tag{8}$$

Implémentez :

1. une nouvelle fonction `sfdm02` dont le fonctionnement sera similaire à la fonction `sfdm01` mais implémentant les schémas décentrés à l'ordre 2,
2. modifier la fonction `nderivate` afin de permettre de choisir l'ordre des schémas (argument `order` par défaut égal à 1),
3. tester à nouveau cette fonction avec un cosinus défini entre 0 et 2π. Calculer et afficher la dérivée 3 de ce cosinus en utilisant les schémas décentrées d'ordre 2. Afficher sur la même courbe les résultats théoriques.

3 Applications

3.1 Application : Photo à la dérive

Le patron de Brian lui a demandé de numériser quelques vieilles photos avant de les jeter dans le broyeur à papier. Jusque là tout allait pour le mieux, mais après avoir effectué ce travail, Brian a voulu faire du zèle et s'est dit que son patron serait probablement très content de lui s'il retouchait un peu ces vieilles photos. Le drame ! Lors de ses essais de retouche, Brian a maladroitement appliqué un filtre intégrateur à l'une des photos et écrasé les données originales. Mais comme Brian est malin et qu'il voulait réparer son erreur, il a essayé d'appliquer un filtre dérivateur aux données numérisées. Tout ce qu'il a réussi à faire est d'empirer les choses en appuyant deux fois de plus sur le bouton **intégration** ! Ah ce Brian !

Maintenant Brian a besoin de votre aide pour récupérer les données originales de cette photo, sans quoi le pauvre risque de perdre son emploi !

1. Créer un nouveau script permettant de charger les données de la photo abimée regroupées dans le fichier binaire *image.npy*
2. Afficher cette image à l'aide de la méthode `imshow()` héritée par les objets `Axes` de matplotlib. Ci-après un exemple permettant d'afficher sous forme d'image l'objet de type `ndarray` `data` :

```
fig, ax = plt.subplots(figsize=(12, 12))
ax.imshow(data, cmap="gray")
plt.show()
```

3. Utiliser vos développements précédents pour dériver 3 fois chaque ligne de l'image afin de réparer les erreurs de Brian. Tester avec les schémas décentrés à l'ordre 1, puis ceux à l'ordre 2.
4. Afficher sur deux sous-figures l'image avant et après la triple dérivation.