TP 2

Ibrahim ALAME

11/10/2019

1 Manipulation élémentaire des tableaux

1.1 Référence contre copie

Examiner la différence entre les lignes suivantes :

```
a = np.arange(5)
b = b[2] = -1
b = a[:]
b[1] = -1
b = a.copy()
b[0] = -1
```

1.2 Tableaux bidimensionnels et découpage

Créez un tableau 4×4 avec des valeurs arbitraires, puis

1. Extraire les éléments de la deuxième ligne.
2. Extraire les éléments de la troisième colonne.
3. Attribuez une valeur de 3 à la sous-matrice 2×2 supérieure gauche.

1.3 Division et combinaison de tableaux

Continuer avec le tableau 4×4 précédent :

1. Utilisez la fonction `np.split()` pour diviser le tableau en deux nouveaux tableaux 2×4. Reconstruisez le tableau 4×4 d'origine en utilisant `np.concatenate()`.
2. Répétez l'exercice ci-dessus, mais créez maintenant des sous-tableaux 4×2, puis combinez-les.

- `np.split(ary,indices_ou_sections, axis=0)`
  - `ary` : le tableau ndarray à diviser en sous-tableaux
  - `indices_ou_sections` est un nombre entier, N, le tableau sera divisé en N tableaux égaux le long de l'axe. Si une telle division n'est pas possible, une erreur est générée. Si `indices_ou_sections` est un tableau 1D d'entiers triés, les entrées indiquent où, le long de l'axe, le tableau est divisé.
  - Axis : int, facultatif, 0 par défaut
- `np.concatenate((ary1,ary2),axis,out=None)`
  - `ary1, ary2` : les tableaux doivent avoir la même forme, sauf dans la dimension correspondant à l'axe

— `axis` : int, facultatif, l'axe le long duquel les tableaux seront joints
— `out` : la destination pour placer le résultat.

2 Analyse numérique

2.1 Différenciation numérique

Les dérivés peuvent être calculés numériquement avec la méthode des différences finies comme suit :

$$f'(x_i) = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x}$$

Construire un tableau 1D Numpy contenant les valeurs de x_i dans l'intervalle $[0, \pi/2]$ avec un pas $\Delta x = 0.1$. Évaluez numériquement la dérivée de $\sin(x)$ dans cet intervalle (à l'exclusion des bornes) à l'aide de la formule ci-dessus. Essayez d'éviter les boucles. Comparez le résultat à la fonction $\cos(x)$ dans le même intervalle.

2.2 Polynômes

Approcher un polynôme du second degré aux données de l'exercice précédent en utilisant `numpy.polyfit()`. Tracer le graphique représentant toutes les données.

2.3 Algèbre linéaire

Construire deux matrices 2×2 symétriques A et B. (indice : une matrice symétrique peut être construite facilement comme $A_{sym} = A + A^t$) Calculez le produit de la matrice $C = A * B$ en utilisant `numpy.dot()`. Calculez les valeurs propres de la matrice C avec `numpy.linalg.eigvals()`.

2.4 Graphique simple

Tracez sur le même graphe les fonctions sin et cos dans l'intervalle $[-\pi/2, -\pi/2]$. Utilisez θ comme $x-$label et insérez aussi les légendes. Enregistrez la figure au format `.png` . Tracez les mêmes fonctions sur deux graphiques séparés en utilisant 2 méthodes.

2.5 Intégration

Le module d'intégration `scipy.integrate` contient des outils d'intégration numérique. Utilisez le module pour évaluer les intégrales

$$\int_1^{3.5} (1 + x^2)\mathrm{d}x \quad et \quad \int_0^{\infty} \exp(-x^2)\mathrm{d}x$$

2.6 Optimisation

Trouver, s'il existe, le minimum de la fonction réelle définie par

$$f(x) = (x + 4)(x + 1)(x - 1)(x - 3)$$

en utilisant la fonction `minimize_scalar()` du module `scipy.optimize`.

2.7 Série de Riemann

Une méthode simple pour évaluer numériquement les intégrales est la somme de Riemann du milieu

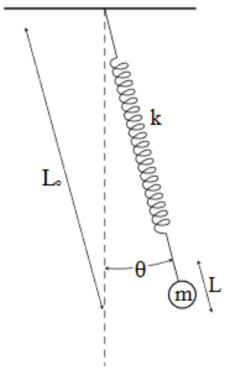
$$S = \sum_{i=1}^{n} f(x_i')\Delta x$$

où $x_i' = \frac{x_i + x_{i-1}}{2}$. Utilisez le même intervalle que dans le premier exercice et déterminez dans quelle mesure la somme de Riemann diffère de 0.1 de la valeur de l'intégrale correspondant. Évitez les boucles. Recherchez également comment les résultats changent avec le choix de Δx.

3 Mécanique du point : Pendule simple à ressort

Comme vous vous en doutez, `Scipy` a une routine `odeint()` qui résout les équations différentielles. Cette fonction est disponible sous `scipy.integrate.odeint()`. Elle utilise des tailles de pas variables et des méthodes de vérification des erreurs pour obtenir des résultats très précis, de manière efficace. Appelez cette routine avec une fonction dérivative, une valeur d'état initiale (qui peut-être un tableau, comme d'habitude) et un tableau de fois (plutôt qu'un pas de temps). La fonction `scipy.integrate.odeint()` retournera un tableau de valeurs d'état en fonction du temps.

Une masse m est attachée à un ressort de raideur k, qui est attachée à un point de support, comme indiqué dans la figure ci-dessous. La longueur du pendule résultant à un instant donné est égale à la longueur de repos du ressort ℓ_0 plus l'allongement (ou la compression) ℓ, et l'angle du pendule par rapport à la verticale est égal à θ. Voici un exemple de système oscillateur couplé : les oscillations «pendulaires» d'écart θ interagissent avec les oscillations «de ressort» d'allongement ℓ, produisant un mélange complexe des deux. Les équations différentielles pour ce système sont données par



$$\ell'' = (\ell_0 + \ell)\theta'^2 - \frac{k}{m}\ell + g\cos\theta$$

$$\theta'' = -\frac{1}{\ell_0 + \ell}\left(2\ell'\theta' + g\sin\theta\right)$$

Écrivez un programme qui trace le mouvement de la masse pour une $\theta \neq 0$ initiale.
Suivez ces étapes :

1. Définissez d'abord les variables initiales et placez-les dans un tableau de dimension = 1 et `shape = (4,)`.
   Nous avons en fait quatre paramètres à suivre (position et vitesse) :

   $$\ell, \quad \ell', \quad \theta \quad \text{et} \quad \theta'$$

   Les valeurs initiales sont $\ell_0 = 1$, $\ell = 0$, $\ell' = 0$, $\theta = 0.3$ et $\theta' = 0$.
2. Définir les pas de temps. Pour résoudre ce problème, nous devons itérer dans le temps. Pour chaque instant t, nous mettrons à jour les valeurs des quatre paramètres. Prenons par exemple $N = 1000$ (nombre de pas de temps). Définissez un tableau de pas de temps pour faire 1000 itérations. La fin de la simulation devrait être à $t = 25s$

3. Définissez une fonction qui prendra le tableau des quatre paramètres (1.) et le tableau du temps (2.). Cette fonction devrait calculer les dérivées des paramètres et les renvoyer.
4. Définissez un tableau $N \times 4$ en utilisant simplement la fonction `scipy.integrate.odeint()`.
5. Enfin, tracez les données.