Projet ICC-P: Anneau en chute

Introduction

Un anneau métallique est en train de tomber dans un bac de liquide. Les deux (anneau et liquide) ont une température non-uniforme. Ce projet de programmation a pour but de modéliser et simuler le transfert de chaleur entre l'anneau et liquide, et de visualiser les résultats de la simulation.

Supposons que l'anneau est à l'horizontal, et tombe verticalement et à vitesse constante dans le liquide. Cet anneau peut être modélisé comme un cercle, représenté par un nombre fini de *points* équidistants le long de son périmètre. De même, le *temps* sera représenté par un nombre fini d'instants. De ce fait, durant la chute simulée de l'anneau, sa position verticale passe de la profondeur d_i à la profondeur d_{i+1}, où $|d_{i+1} - d_i|$ est la distance parcourue par l'anneau entre deux instants, t_i et t_{i+1}.

Avoir un nombre fini de points sur l'anneau permet de modéliser les propriétés physiques de l'anneau et du liquide qu'il traverse. Au fur et à mesure que l'anneau tombe et que le temps passe (c'est à dire, au fur et à mesure que le temps et la profondeur augmentent), on suit les points de cet anneau, qui tracent la surface d'un cylindre fictif. Cette région d'espace-temps représentée par la surface du cylindre peut être "dépliée" en plan, comme illustré en Figure 1a. Ce plan peut être, à son tour, représenté par une image en deux dimensions, et visualisé comme tel.

Le plan sur lequel le cylindre peut être déplié peut être représenté en un tableau à 2 dimensions (2D) de données, indexé par la profondeur et le temps. Considérez la Figure 1b, l'abscisse (axe x, nommé p) représente les différents points le long de l'anneau à une profondeur spécifique, et l'ordonnée (axe y, nommé d) représente la profondeur. Comme l'anneau tombe à la vertical, la profondeur à la surface du liquide est de zéro.

Modélisation de l'anneau: Un tableau 2D sera utilisé pour stocké la température de chaque point de l'anneau pendant la chute de l'anneau. Une ligne de ce tableau 2D modélisera la température de l'anneau entier à la profondeur et au moment correspondant. Une colonne de ce tableau 2D modélisera la température d'un point particulier de l'anneau à différentes profondeurs pendant la chute. Ce tableau 2D peut être stocké comme une liste de listes en Python.

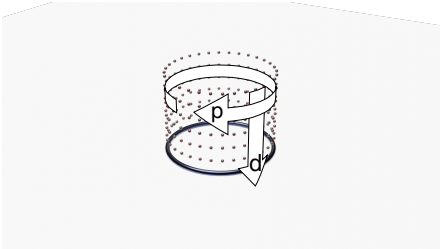
Modélisation du liquide: De la même manière, un tableau 2D modélise la température de la surface cylindrique du liquide qui entre en contact avec l'anneau tombant pour chaque pas de temps.

Ce devoir comporte trois parties principales:

1. **Partie 1 [50 pts]:** Implémenter la fonctionnalité de conversion de tableaux 2D en images colorées afin de pouvoir réaliser les premières visualisations.
2. **Partie 2 [20 pts]:** Simuler le transfert de chaleur entre l'anneau et le liquide qui l'entoure.
3. **Partie 3 [30 pts]:** Simuler le transfert de chaleur entre les points équidistants du l'anneau.

La partie 1 est divisée en plusieurs sous-problèmes. Bien que nous recommandions de commencer par la partie 1, afin que vous puissiez visualiser les résultats des simulations, les trois parties sont indépendantes et vous pouvez parfaitement travailler dessus dans n'importe quel ordre. Nous

(a) Durant sa chute, l'anneau traverse un cylindre fictif, qui peut être déplié en une surface à 2 dimensions, qui peut à son tour être représentée par une image. L'anneau est représenté comme un anneau "plein", malgré le fait que nous le modélisons comme un ensemble de points équidistants. Les points du liquide en contact avec l'anneau sont représentés par des simples points.



(b) Le lien entre la surface à 2 dimensions (image) et l'anneau tombant.

Figure 1: Une illustration de comment les points de l'anneau et du liquide peuvent être représentés par une image.

fournissons des fonctions d'aide lorsque cela est nécessaire, par exemple pour visualiser les images générées par votre code dans la partie 1.

Tester et soumettre votre code pour évaluation

Le fichier `falling_ring.py` contient des espaces réservés pour les fonctions que vous devez implémenter. **C'est le fichier que vous soumettrez sur Moodle pour une notation automatisée**.

Important: ne modifiez pas les en-têtes des fonctions, c'est-à-dire leurs noms ou leurs arguments (noms, ordre ou type). Si vous modifiez l'un de ces éléments, le score sera de zéro pour la fonction concernée. D'autre part, vous n'avez pas à vous soucier de traiter des arguments d'un type incorrect. Par exemple, si nous vous demandons une fonction qui prend une valeur réelle, le système de notation ne lui attribuera jamais autre chose qu'un float.

Pour tester votre code localement (sans l'évaluation automatique de Moodle), modifiez la fonction `quick_tests` en haut de `simulator.py`. Cette fonction (avec vos propres tests) sera appelée lorsque vous lancerez `simulator.py` avec l'argument test comme suit:

```
>> python simulator.py --test
```

Tout au long de ce document, nous fournissons des exemples d'entrées et de sorties pour chaque fonction évaluée, que vous pouvez copier dans `quick_test` et utiliser pour tester votre code. Le script de `simulator.py` peut également exécuter des tests plus complexes qui génèrent des images. Les commandes nécessaires et les résultats attendus sont décrits dans les sections suivantes.

Enfin, nous fournissons également des tests automatisés similaires à ceux utilisés par notre évaluateur automatisé dans Moodle. Vous pouvez exécuter les tests comme suit:

```
>> python tests.py
```

Les résultats de chaque test seront imprimés. **Ces tests sont le meilleur moyen de vérifier votre code avant de le soumettre à l'évaluation et vous donneront un retour plus détaillé que sur Moodle**. N'hésitez pas à ajouter vos propres tests pour déboguer davantage votre code, car ce fichier n'est de toute façon pas destiné à être soumis à Moodle.

Notez que toutes les valeurs renvoyées par les fonctions sont arrondies à deux chiffres après la virgule avant d'être imprimées afin d'éliminer les erreurs d'arrondi. En pratique, pour l'ensemble de ce projet, ce qui est renvoyé par vos fonctions sera considéré comme "correct" si chaque valeur se situe dans une marge d'erreur de ± 0.005 centrée sur ce que nous attendons (c'est-à-dire $|a - b| < 0.005$ où a est votre valeur et b la valeur de référence). **Vous ne devez pas arrondir les valeurs de retour. Le correcteur se chargera de cette comparaison.**

Nous vous conseillons d'utiliser la machine virtuelle que vous utilisez dans les laboratoires pour coder ce projet. Nous recommandons d'utiliser `Python3.10`, la version de `Python` fournie dans la machine virtuelle. De plus, les bibliothèques externes nécessaires à ce projet sont *matplotlib* et *numpy* et sont déjà installées dans la machine virtuelle.

Instructions relatives à la soumission du travail

Vous soumettrez votre travail sur Moodle (https://moodle.epfl.ch/) par groupe. La notation se fera automatiquement sur Moodle. Chaque fonction sera notée séparément. Pour toutes les fonctions, vous recevrez des points en fonction du nombre total de tests réussis, chaque test ayant le même poids.

Moodle affiche votre score total et génère un fichier `report.txt`, contenant une liste des cas de test réussis et échoués. **Notez que Moodle affiche le score total de votre dernière soumission et non le score le plus élevé de toutes les soumissions.**

De plus, avant de soumettre votre code à l'évaluation sur Moodle, assurez-vous que les tests locaux que nous vous avons fournis passent avec succès. De plus, nous vous recommandons de consulter le guide de style Python suivant:
https://peps.python.org/pep-0008/#introduction

Pour plus d'informations (par exemple, les instructions sur l'accès, la date limite pour soumettre le code, le nombre maximum d'essais autorisé, etc.), veuillez consulter la page Moodle du cours.

Partie 1: Visualisation [50 pts]

Comme décrit dans l'introduction, il y a des tableaux 2D pour stocker la température de l'anneau et le liquide pendant la simulation. Les tableaux 2D doivent être traités de la même manière pour être visualisés. Pour ce faire, nous allons mettre en place un moyen de faire correspondre une couleur à une température. La manière la plus simple est d'assigner un ensemble discret de couleurs à un ensemble discret de températures. En conséquence, les couleurs peuvent être mélangées pour créer une couleur intermédiaire appropriée pour des températures intermédiaires.

Trouver l'intervalle approprié [15 pts]

Dans cette partie, vous allez écrire la fonction suivante:

```
def get_interval(
    query, # float, la température pour laquelle trouver un intervalle
    values # une liste triée de floats, les températures pour lesquelles les couleurs
        sont connues
) # renvoie un int, l'indice de la borne inférieure de l'intervalle auquel appartient
    "query"
```

Cette fonction devrait trouver et renvoyer `i`, $0 \leq$ `i` $<$ `len(values)`, tel que:
`values[i]` $\leq$ `query` $<$ `values[i+1]`.

Considérez ces cas particuliers:

- Si `query` est plus petit que `values[0]`, la fonction devrait renvoyer zéro.
- Si `query` est plus grand ou égal que `values[-1]`, la fonction devrait renvoyer `len(values)-1`.

Voici quelques tests pour la fonction:

```
get_interval(0.5, [-1, 0, 1, 2, 3])
# -> 1

get_interval(5, [-100, 0, 3, 9])
# -> 2

get_interval(2.5, [0, 1, 2])
# -> 2

get_interval(0.5, [2, 6, 50])
# -> 0

get_interval(3, [1, 2, 3, 4])
# -> 2
```

Interpolation des couleurs [15 pts]

Dans cette partie, vous allez écrire la fonction suivante:

```
def interpolate_color(
    query, # float, la température pour laquelle obtenir la couleur
    values, # une liste triée de floats, les températures pour lesquelles nous
        connaissons les couleurs correspondantes
    colors # liste de listes, chacune de ces listes contient trois flottants
        représentant les composantes rouge, verte et bleue de la couleur pour les
        températures dans "values"
) # renvoie une liste de 3 flottants, les composantes rouge/verte/bleue de la couleur
    pour la température de la requête
```

Cette fonction prend `query` correspondant à une température, `values` correspondant à une liste triée de températures, et `colors` qui déclare la couleur correspondante pour chaque température dans `values`. En conséquence, il calcule et renvoie la couleur de `query`. Les deux listes, `values` et `colors`, ont au moins deux et le même nombre d'éléments.

`colors` est une liste où chaque couleur est représentée par une liste de trois nombres à virgule flottante allant de 0 à 1. Une couleur affichée sur un écran comprend trois composantes lumineuses: le rouge, le vert et le bleu. En conséquence, chacun des trois nombres flottants correspond à l'intensité de l'une de ces couleurs: 0 représente l'absence de lumière, tandis que 1 représente la lumière maximale. Par exemple:

- La couleur [1, 0, 0] ne contient qu'une composante rouge à l'intensité la plus élevée, sans aucune composante verte ou bleue.
- La couleur [0, 1, 0] ne contient qu'une composante verte.
- La couleur [0, 0, 1] ne contient qu'une composante bleue.
- `values=[-1, 2]` et `colors=[[1, 0, 0], [0, 0, 1]]` signifie que les températures de -1 et 2 degrés doivent être affichées respectivement en rouge et en bleu.

La fonction trouve d'abord l'intervalle de température correspondant dans `values` pour la `query` en appelant la fonction `get_interval` que vous avez implémentée précédemment. En utilisant l'index `i` obtenu, la fonction `interpolate_color` doit lire les deux couleurs correspondantes dans la liste `colors` et les mélanger linéairement. Dans les formules ci-dessous, `P` représente la distance relative de `query` par rapport au début de l'intervalle correspondant. Cette distance est utilisée pour calculer la couleur de `query` en calculant la moyenne pondérée des couleurs associées aux températures définissant l'intervalle de `query`. Notez que chaque élément dans `colors` est une liste à trois éléments, et la multiplication de la liste par un flottant signifie que le flottant met à l'échelle tous les éléments de la liste. Vous trouverez ces formules ci-dessous, applicables à `0 <= i < (len(values)-1)`:

$$\texttt{P} = \frac{\texttt{query - values[i]}}{\texttt{values[i+1] - values[i]}}$$

$$\texttt{P\_color} = \texttt{(1-P)}\cdot\texttt{colors[i] + P}\cdot\texttt{colors[i+1]}$$

Votre fonction doit mettre en œuvre les cas de figure suivants:

- Si `query` $<$ `values[0]`, sa couleur est `colors[0]`.
- Si `query` $\geq$ `values[-1]`, sa couleur est `colors[-1]`.

Voici quelques tests pour la fonction:

```
def round_list(L):
    # arrondit chaque valeur d'une liste à deux chiffres après la virgule
    return [round(x,2) for x in L]

round_list(interpolate_color(0.6, [-1,0,1],[[0,0,0],[0,1,0],[0,1,1]]))
# -> [0.0, 1.0, 0.6]

round_list(interpolate_color(5, [0,3,9], [[1,0,0],[0,1,0],[0,0,1]]))
# -> [0.0, 0.67, 0.33]
```

```
round_list(interpolate_color(2.5, [0,1,2], [[1,0,0],[0,1,1],[0,0,1]]))
# -> [0, 0, 1]

round_list(interpolate_color(0.5, [2,6,50], [[1,0,0],[0,1,1],[0,0,1]]))
# -> [1, 0, 0]
```

Application d'une carte de couleur (ColorMap) à une image [20 pts]

Maintenant que nous avons la fonction `interpolate_color` pour associer une couleur à une température, tout ce qui reste à faire pour la visualisation est d'appliquer cette fonction à chaque point du tableau 2D que nous voulons afficher.

Comme nous l'avons décrit dans l'introduction, un tableau 2D peut modéliser (1) les températures de l'anneau qui tombe (pendant l'intervalle de temps nécessaire à la chute de l'anneau) ou (2) les températures du liquide (pour chaque instant). Dans l'un ou l'autre des deux cas, nous avons un tableau 2D de températures, que nous pouvons visualiser sous forme d'une image en couleurs 2D.

Ces tableaux 2D, dans la terminologie Python, sont des listes de listes; si le nom de la liste est `L`, alors `L[d][p]` est la température de l'anneau au point `p` et à la profondeur `d` (voir Fig. 1b).

Dans cette partie, vous allez écrire la fonction suivante:

```
def map_to_color(
    in_data, # une liste de lists de floats
    out_data, # une liste de listes de listes de floats
    values, # une liste de floats
    colors, # une liste de listes de floats
)  # ne renvoie rien
```

La signification des paramètres de la fonction est la suivante:

- `in_data` est une liste de listes, où chaque liste correspond à une profondeur spécifique et contient autant de nombres flottants qu'il y a de points équidistants sur l'anneau. La liste à l'index zéro correspond à la position de départ de l'anneau. Chaque valeur dans cette liste 2D représente une température, et vous devez associer ces valeurs à leurs couleurs correspondantes. Cette liste doit être accédée comme `in_data[d][p]`, où `d` et `p` représentent respectivement la profondeur et l'index d'un point sur la surface 2D (voir Figure 1.b).
- `out_data` est une liste tridimensionnelle. La première dimension correspond à la profondeur, la deuxième aux points de l'anneau, et la troisième à la couleur du point. Elle doit donc être accédée comme `out_data[d][p][color_channel]`, où `color_channel` est 0 pour le rouge, 1 pour le vert et 2 pour le bleu. Aucune autre valeur n'est acceptée pour `color_channel`.
- `values` et `colors` sont simplement les mêmes `values` et `colors` décrits dans les sections précédentes.

Vous pouvez supposer que `out_data` et `in_data` ont la même longueur, que toutes les listes dans `in_data` et `out_data` ont la même longueur, et que les listes dans les listes de `out_data` ont une longueur de trois (le nombre de canaux de couleur). De plus, il est sûr de passer directement `colors` et `values` à `interpolate_color`.

Voici quelques tests pour la fonction:

```
def round_color_image(img):
    # arrondit chaque valeur dans une liste de listes de listes, à deux chiffres après
        la virgule
    return [[[round(x,2) for x in pixel] for pixel in row] for row in img]

in_data = [[-0.6, -0.4], [-0.3, 3.1]]
out_data = [ [[0]*3 for j in range(2)] for i in range(2) ]

map_to_color(in_data, out_data, [-1,0,1], [[1,0,0],[0,1,0],[0,0,1]])
print(round_color_image(out_data))
# [ [ [0.6, 0.4, 0.0], [0.4, 0.6, 0.0] ],
#   [ [0.3, 0.7, 0.0], [0.0, 0.0, 1.0] ] ]

in_data = [[-0.6, -0.4, -0.3, -3.7, 3.4],
      [-0.3, 3.1, -3.9, -0.2, 0.7],
      [4.5,  4.8, 2.3, 4.6, 2.4],
      [-1.8, -3.1, 3.1, -0.2, 2.5],
      [-1.7, -2.0, 0.6, -3.1, 4.6]]
out_data = [ [[0]*3 for j in range(5)] for i in range(5) ]

map_to_color(in_data,out_data,[-1,0,1,2,3],
    [[0.0,0.0,0.0],[0.0,0.0,1.0],[1.0,0.0,1.0],[1.0,0.0,0.0],[1.0,1.0,1.0]])
print(round_color_image(out_data))
# [ [ [0.0, 0.0, 0.4], [0.0, 0.0, 0.6], [0.0, 0.0, 0.7], [0.0, 0.0, 0.0], [1.0, 1.0,
    1.0] ],
#   [ [0.0, 0.0, 0.7], [1.0, 1.0, 1.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.8], [0.7, 0.0,
    1.0] ],
#   [ [1.0, 1.0, 1.0], [1.0, 1.0, 1.0], [1.0, 0.3, 0.3], [1.0, 1.0, 1.0], [1.0, 0.4,
    0.4] ],
#   [ [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [1.0, 1.0, 1.0], [0.0, 0.0, 0.8], [1.0, 0.5,
    0.5] ],
#   [ [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.6, 0.0, 1.0], [0.0, 0.0, 0.0], [1.0, 1.0,
    1.0] ] ]
```

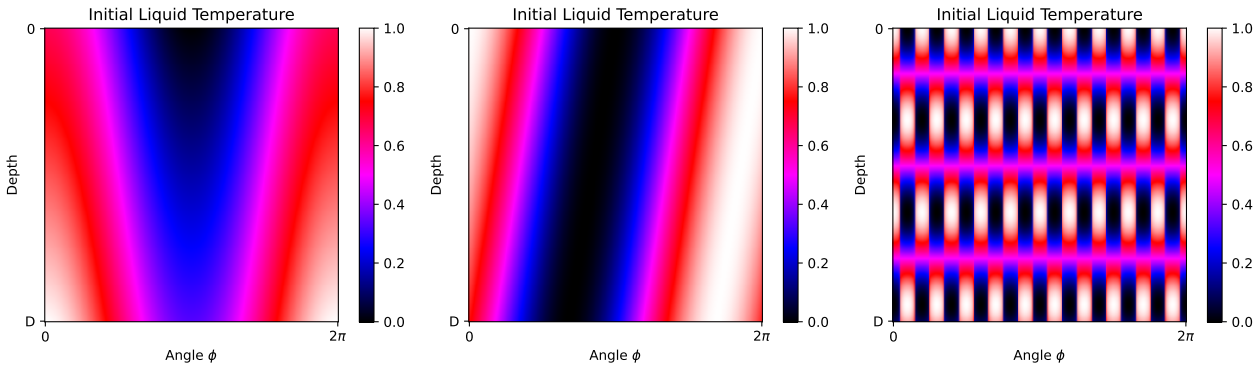
Une façon de tester votre fonction est de visualiser la température initiale du liquide en utilisant la commande suivante. Trois entrées sont disponibles, qui peuvent être testées en remplaçant N par 1, 2 ou 3:

```
>> python simulator.py --visualize N
```

Vous devriez obtenir les images ci-dessous si vous implémentez correctement vos fonctions. Notez que l'axe des ordonnées représente la profondeur, qui va de zéro (en haut, là où l'anneau commence à tomber) jusqu'à `D` (la profondeur maximale). L'axe des abscisses représente les points de l'anneau. Pour plus de commodité, au lieu de numéroter les points de zéro jusqu'au nombre total de points, nous convertissons l'indice d'un point de l'anneau p en angle radial équivalent à l'aide de la formule suivante:

$$\phi = 2\pi \frac{p}{\text{nombre\_de\_points\_dans\_l'anneau}}$$

Sur le côté droit de la figure, nous affichons la correspondance entre les températures et les couleurs.



(a) N = 1 (b) N = 2 (c) N = 3

Figure 2: Le résultat de `map_to_color` pour les entrées disponibles.

Partie 2: Échange thermique [20 pts]

Dans cette section, nous modélisons le transfert de chaleur au sein du système simulé. Nous calculons le transfert de chaleur entre deux points en déterminant la *température suivante* (un pas de temps plus tard) d'un point (T_{next}) en fonction de sa *température actuelle* (T_{current}) et de la *température actuelle* du point avec lequel il échange de la chaleur. Trois constantes reflètent les propriétés physiques impliquées dans ce transfert de chaleur spécifique. Ces constantes contiennent *implicitement* des informations sur (1) la distance entre les points et (2) la durée du pas de temps.

Tout d'abord, nous devons calculer la quantité d'énergie transférée entre les points `a` et `b` pendant le pas de temps, qui dépend du **coefficient de transfert** k_{t}:

$$E_{\text{a}\to\text{b}} = k_{\text{t}}(T_{\text{a,current}} - T_{\text{b,current}})$$

En utilisant l'énergie transférée $E_{\text{a}\to\text{b}}$ calculée entre les points, nous pouvons utiliser les **coefficients de capacité thermique** des deux points, c_{a} et c_{b}, pour déterminer l'effet du transfert d'énergie sur les températures des deux points. Notez comment la direction du transfert d'énergie affecte la seconde équation:

$$T_{\text{a,next}} = T_{\text{a,current}} - \frac{E_{\text{a}\to\text{b}}}{c_{\text{a}}}$$

$$T_{\text{b,next}} = T_{\text{b,current}} + \frac{E_{\text{a}\to\text{b}}}{c_{\text{b}}}$$

Si un point interagit avec plus qu'un autre point, tous les transferts de chaleur pour ce point doivent être calculés depuis la température actuelle, des points. Quelque chose d'intéressant à noter est que les températures de points en contact ont tendance à se rapprocher, sauf si la valeur de k_{t} (qui contient implicitement la durée entre deux instants) est trop large. Dans ce cas, le comportement des températures des points serait un non-sens, du point de vue physique. Une autre chose intéressante à noter est la *conservation de l'énergie*: la quantité totale d'énergie thermique de la simulation, qui est la somme de $c_i T_i$ pour tous les points i, est constante.

En premier lieu pour modéliser le transfert de chaleur, modélisons le transfert de chaleur entre

l'anneau et le liquide à son contact. Pour chaque point de l'anneau, nous considéreront une seule interaction: avec le point du liquide avec les mêmes coordonnées (d, p).

Dans cette partie, vous allez écrire la fonction suivante:

```
def thermal_exchange(
    liquid_temp, # liste de listes de float
    ring_temp, # liste de listes de float
    current_depth, # int, non-négatif
    transfer_coefficient=1.0, # float, positif
    capacity_coefficient_liquid=1.0, # float, positif
    capacity_coefficient_ring=1.0 # float, positif
    ) # ne renvoie rien
```

Cette fonction modifie la température des points de l'anneau et des points du liquide en contact avec l'anneau pour simuler l'échange d'énergie thermique entre eux. Notez que:

- `liquid_temp` et `ring_temp` sont deux listes 2D, l'une représentant la température *actuelle* du liquide et l'autre l'*historique* de toutes les températures de l'anneau pendant sa chute.
- `current_depth` est la profondeur actuelle de l'anneau, correspondant à l' *indice de ligne* dans la liste 2D `ring_temp` ainsi qu'au moment où l'échange thermique a lieu.

Vous pouvez supposer que `liquid_temp` et `ring_temp` ont la même taille. De plus, `current_depth` sera donné de sorte que `0 <= current_depth < len(liquid_temp)`.

Voici quelques tests pour la fonction:

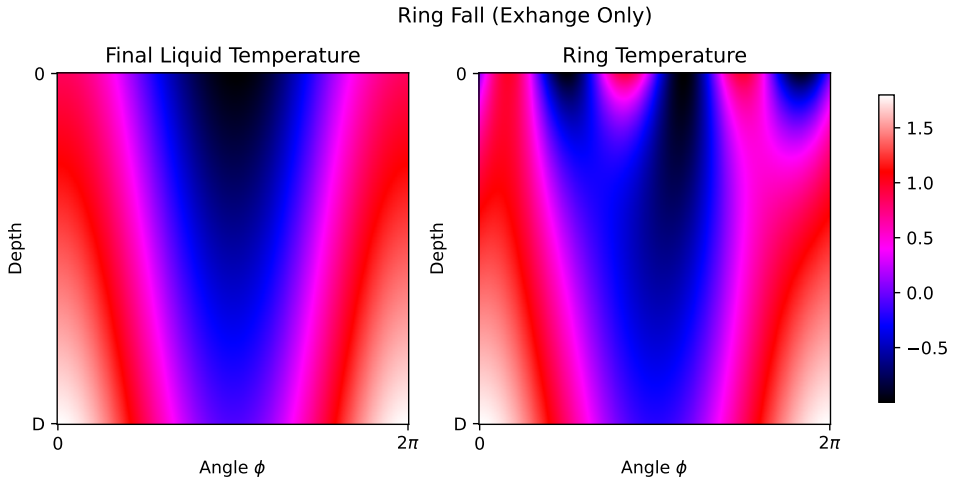
```
def round_image(img):
    # arrondit chaque valeur dans une liste de listes, à deux chiffres après la virgule
    return [[round(x,2) for x in row] for row in img]

print("example 1")
liquid_temp = [[0.0,0.1,0.2,-0.3,-0.4,0.5]]
ring_temp = [ [0]*6 ]
thermal_exchange(liquid_temp, ring_temp, 0, 0.1, 1.0, 1.0)
print(round_image(liquid_temp)) # [[0.0, 0.09, 0.18, -0.27, -0.36, 0.45]]
print(round_image(ring_temp)) # [[0.0, 0.01, 0.02, -0.03, -0.04, 0.05]]

print("example 2")
liquid_temp = [[0,0.1,0.2,0.3,0.4,0.5]]
ring_temp = [ [0]*6 ]
thermal_exchange(liquid_temp, ring_temp, 0, 0.1, 1.0, 0.25)
print(round_image(liquid_temp)) # [[0.0, 0.09, 0.18, 0.27, 0.36, 0.45]]
print(round_image(ring_temp)) # [[0.0, 0.04, 0.08, 0.12, 0.16, 0.2]]

print("example 3")
liquid_temp = [[0,0.1,0.2,0.3,0.4,0.5], [0,0.1,0.2,0.3,0.4,0.5]]
ring_temp = [[0]*6, [0]*6]
thermal_exchange(liquid_temp, ring_temp, 1, 0.1, 1.0, 0.25)
print(round_image(liquid_temp)) # [[0.0, 0.1, 0.2, 0.3, 0.4, 0.5],
             #  [0.0, 0.09, 0.18, 0.27, 0.36, 0.45]]
print(round_image(ring_temp)) # [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
           #  [0.0, 0.04, 0.08, 0.12, 0.16, 0.2]]
```

Notez que `ring_temp` a la même longueur que `liquid_temp`. Par conséquent, les valeurs dans `ring_temp` au-delà de `current_depth` sont simplement des valeurs fictives réservées pour `ring_temp` au fur et à mesure que la simulation avance. Observez comment `thermal_exchange` est utilisé dans le fichier `simulator.py`. Dans `run_thermal_exchange`, le simulateur commence avec une température initiale pour l'anneau. Ensuite, il itère sur toutes les valeurs de `current_depth`, en commençant de 0 jusqu'à la profondeur maximale. Initialement, la température actuelle de l'anneau est égale à sa température à la profondeur précédente à chaque itération. Ensuite, `thermal_exchange` est appelé pour mettre à jour la température actuelle de l'anneau à la profondeur courante. Comprendre comment `thermal_exchange` est utilisé dans la simulation peut vous aider à implémenter cette fonction correctement.

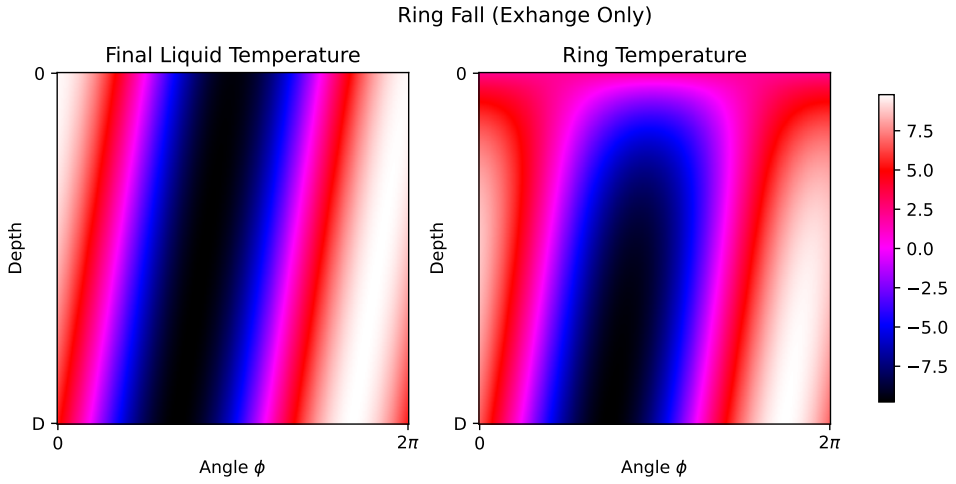
Vous pouvez utiliser le script de simulation pour tester la fonction sur des entrées de grande taille avec la commande suivante (remplacez N par 1, 2 ou 3):

```
>> python simulator.py --thermal-exchange N
```

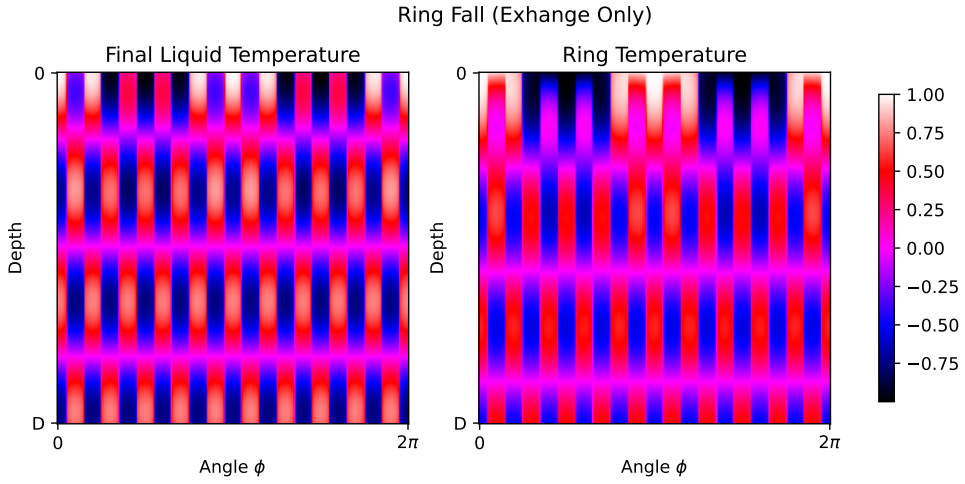
Si vous implémentez la fonction correctement, vous devriez obtenir les images ci-dessous.



(a) N = 1



(b) N = 2



(c) N = 3

Figure 3: Le résultat de l'échange thermique pour différentes entrées disponibles.

Partie 3: Conduction Thermique [30 pts]

Jusqu'à présent, nous avons considéré le transfert de chaleur entre le liquide et l'anneau. Cependant, en raison de la conduction thermique, la température tend à devenir homogène à l'intérieur de l'anneau lui-même. Modélisons maintenant le transfert de température entre les points de l'anneau, en appliquant le même mécanisme physique que précédemment.

Dans cette partie, vous allez écrire la fonction suivante:

```
thermal_conduction(
    ring_temp, # liste de listes de floats, toutes de la même longueur
    current_depth, # entier non négatif
    transfer_coefficient=1.0, # float positif
    capacity_coefficient_ring=1.0: # float positif
    )  # ne renvoie rien
```

Cette fonction simule l'échange thermique à l'intérieur de l'anneau et à la profondeur donnée. Notez que chaque point de l'anneau (`ring_temp[current_depth][p]`) interagit avec ses deux points voisins (`ring_temp[current_depth][p-1]` et `ring_temp[current_depth][p+1]`). Notez que ces deux interactions doivent lire la même température actuelle du point donné. De plus, les points à `ring_temp[current_depth][0]` et `ring_temp[current_depth][-1]` sont voisins (rappelez-vous comment les coordonnées des points de l'anneau et les températures sont transformées en une image 2D).

Vous pouvez supposer que toutes les listes dans `ring_temp` auront la même taille, et que `0 <= current_depth < len(ring_temp)`.

Voici quelques tests pour la fonction:

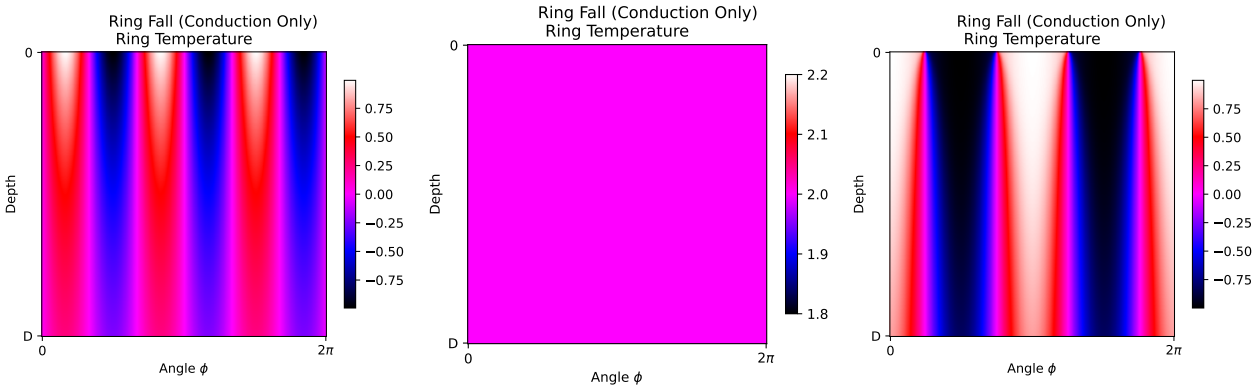
```python
def round_image(img):
    # arrondit chaque valeur dans une liste de listes, à deux décimales
    return [[round(x,2) for x in row] for row in img]

print("example 1")
ring_temp = [[0,0.1,0.2,0.4,0.2,0.1]]
print("sum before:", round( sum(ring_temp[0]), 2) ) # -> "sum before: 1.0"
thermal_conduction(ring_temp,0, 0.1,1.0)
print(round_image(ring_temp)) # [[0.02, 0.1, 0.21, 0.36, 0.21, 0.1]],
print("sum after:", round( sum(ring_temp[0]), 2) ) # -> "sum after: 1.0"
# notez comment la quantité d'énergie n'a pas changé

print("example 2")
ring_temp = [[0,0.0,0.0,0.0,0.0,0.0]]
thermal_conduction(ring_temp,0, 0.1,1.0)
print(round_image(ring_temp)) # [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]

print("example 3")
ring_temp = [[0,0,0,0,0,1]]
thermal_conduction(ring_temp,0, 0.1,1.0)
print(round_image(ring_temp)) # [[0.1, 0.0, 0.0, 0.0, 0.1, 0.8]]
# Notez comment la chaleur s'est propagée d'une extrémité de la liste à l'autre,
# les extrémités correspondant à deux points voisins sur le cercle.
```

```
print("example 4")
ring_temp = [[0,0.1,0.2,0.4,0.2,0.1]]
thermal_conduction(ring_temp,0, 0.1,1.0)
print(round_image(ring_temp)) # [[0.02, 0.1, 0.21, 0.36, 0.21, 0.1]]
```

Vous pouvez tester votre fonction sur des entrées de grande taille avec le script de simulation comme indiqué ci-dessous (remplacez N par 1, 2 ou 3):

```
>> python simulator.py --thermal-conduction N
```

Si vous implémentez correctement la fonction, vous devriez obtenir les figures ci-dessous:



(a) N = 1 (b) N = 2 (c) N = 3

Figure 4: Le résultat `thermal_conduction` pour différentes entrées disponibles.

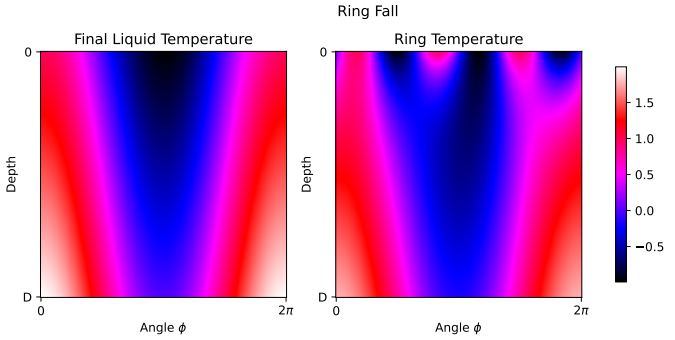
Simulation de l'Anneau en Chute

Vous pouvez tester toute votre implémentation en exécutant le script de simulation suivant (remplacez N par 1, 2 ou 3):

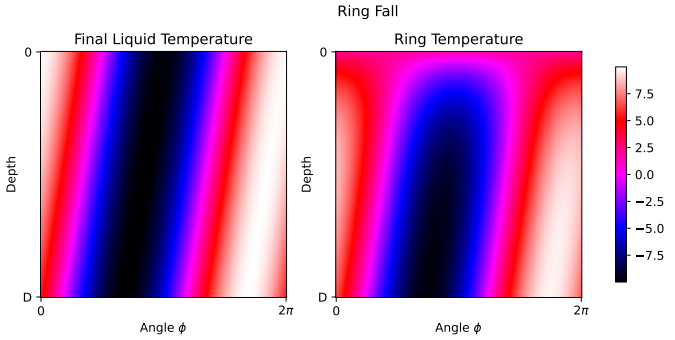
```
>> python simulator.py --run-simulation N
```

Le résultat affichera les résultats de la simulation, y compris la température finale du liquide et l'historique de la température de l'anneau pendant sa descente.

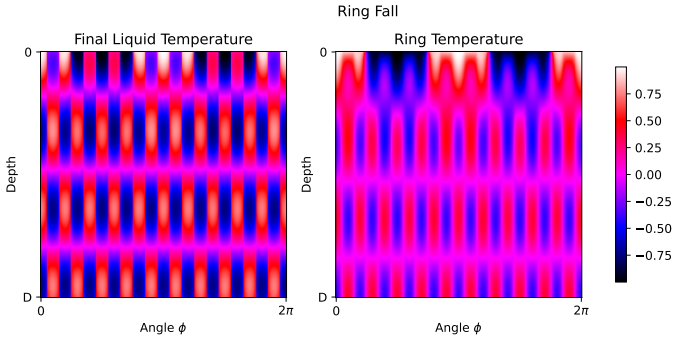
Vous devriez obtenir les figures suivantes si vous implémentez correctement toutes les fonctions.



(a) N = 1



(b) N = 2



(c) N = 3

Figure 5: Le résultat de la simulation complète pour différentes entrées disponibles.