

# TP 2 - TRANSPORT OPTIMAL ENTROPIQUE ET INFÉRENCE DE TRAJECTOIRES

Elias Ventre

contact: `elias.ventre@inria.fr`

## Résumé

Dans ce travail pratique, nous implémentons l'algorithme de Sinkhorn pour calculer le transport optimal entropique entre deux distributions empiriques. Nous utilisons ensuite ces couplages optimaux pour reconstruire des trajectoires cellulaires à partir de mesures temporelles discrètes, et évaluons la qualité de reconstruction via l'interpolation de McCann. Cette approche s'inscrit dans le contexte de l'analyse de données de biologie cellulaire, où l'on cherche à inférer les trajectoires de différenciation des cellules.

## 1 Introduction

### 1.1 Contexte biologique

La différenciation cellulaire est un processus dynamique où des cellules changent progressivement leur expression génique pour adopter des rôles spécifiques. Comprendre ces trajectoires de différenciation est fondamental en biologie du développement et en médecine régénérative.

Les technologies modernes de séquençage d'ARN sur cellule unique (scRNA-seq) permettent de mesurer l'expression de milliers de gènes dans des cellules individuelles. Cependant, la mesure détruit les cellules : on ne peut observer que des *snapshots* de populations cellulaires à différents temps, sans pouvoir suivre les trajectoires individuelles.

**Problème :** Étant données des distributions empiriques de cellules  $\hat{\mathbb{P}}_{t_1}, \dots, \hat{\mathbb{P}}_{t_N}$  à différents temps, comment reconstruire les trajectoires de différenciation ?

### 1.2 Modélisation mathématique

On modélise la différenciation cellulaire par une équation différentielle stochastique (EDS) avec dérive gradient :

$$dX_t = -\nabla\Psi(t, X_t)dt + \sqrt{2\sigma}dB_t \quad (1)$$

où  $\nabla\Psi$  représente le paysage épigénétique (dérive),  $\sigma$  est le coefficient de diffusion, et  $B_t$  est un mouvement brownien.

### 1.3 Objectifs du TP

Ce travail pratique vise à :

- Comprendre le lien entre transport optimal entropique et problème de Schrödinger
- Implémenter l'algorithme de Sinkhorn à la main,
- Reconstruire des trajectoires par chaînage de couplages optimaux,
- Calculer l'interpolation de McCann (géodésiques de Wasserstein)
- Évaluer quantitativement la qualité de reconstruction
- Observer l'effet du paramètre de régularisation entropique  $\varepsilon$

## 1.4 Organisation du TP

### Important

Le TP est organisé en deux notebooks Jupyter qui contiennent le code de visualisation et d'analyse. Votre travail consiste à **implémenter les fonctions dans le fichier `src/functions.py`**. Les notebooks appelleront vos fonctions pour effectuer les calculs et afficher les résultats.

**Structure du repository :**

- `src/functions.py` : Fichier où vous devez implémenter vos fonctions (voir Section 4)
- `src/simulation.py` : Module de simulation des EDS (fourni)
- `notebooks/01_entropic_ot.ipynb` : Notebook partie 1 (fourni)
- `notebooks/02_trajectory_inference.ipynb` : Notebook partie 2 (fourni)

## 2 Transport optimal entropique

### 2.1 Rappels théoriques

#### 2.1.1 Transport optimal classique

Étant données deux distributions de probabilité  $\mu$  et  $\nu$  sur  $\mathbb{R}^d$ , le problème de transport optimal de Kantorovich cherche le couplage  $\pi$  qui minimise le coût de transport :

$$\min_{\pi \in \Pi(\mu, \nu)} \mathbb{E}_{\pi}[c(X, Y)] \quad (2)$$

où  $\Pi(\mu, \nu)$  est l'ensemble des couplages ayant  $\mu$  et  $\nu$  comme marginales, et  $c(x, y) = |x - y|^2$  est le coût quadratique.

Pour des distributions empiriques  $\mu = \frac{1}{m} \sum_{i=1}^m \delta_{x_i}$  et  $\nu = \frac{1}{m} \sum_{j=1}^m \delta_{y_j}$ , le calcul du transport optimal exact a une complexité  $O(m^3 \log m)$  avec l'algorithme du réseau de flots.

#### 2.1.2 Régularisation entropique

Pour améliorer la stabilité numérique et réduire la complexité, on ajoute un terme d'entropie de Shannon :

$$\gamma_{\varepsilon}^* = \arg \min_{\pi \in \Pi(\mu, \nu)} \{ \mathbb{E}_{\pi}[c(X, Y)] + \varepsilon H(\pi) \} \quad (3)$$

où  $H(\pi) = - \int \pi(x, y) \log \pi(x, y) dx dy$  est l'entropie.

L'algorithme de Sinkhorn [2] résout ce problème avec une complexité  $O(m^2 K)$  où  $K$  est le nombre d'itérations (typiquement  $K \ll m$ ).

#### 2.1.3 Lien avec le problème de Schrödinger

Le transport optimal entropique peut se reformuler comme un problème d'entropie relative :

$$\min_{\pi \in \Pi(\mu, \nu)} KL \left( \pi \middle| e^{-c/\varepsilon} \right) = \min_{\pi \in \Pi(\mu, \nu)} H(\pi) + \frac{1}{\varepsilon} \mathbb{E}_{\pi}[c] \quad (4)$$

Cela correspond au *problème de Schrödinger* : trouver le processus stochastique le plus proche (au sens de l'entropie relative) du mouvement brownien, qui relie  $\mu$  à  $\nu$  en temps  $\Delta t$  [3].

**Résultat clé :** Pour  $\varepsilon = \sigma \Delta t$ , le couplage optimal  $\gamma_{\varepsilon}^*(x, \cdot)$  correspond au noyau de transition d'une EDS à dérive gradient reliant  $\mu$  à  $\nu$  !

## 2.2 Algorithme de Sinkhorn

L'algorithme de Sinkhorn exploite la structure du problème régularisé. La solution s'écrit sous la forme :

$$\gamma^* = \text{diag}(u)K\text{diag}(v) \quad (5)$$

où  $K_{ij} = e^{-c_{ij}/\varepsilon}$  est le noyau de Gibbs, et  $(u, v) \in \mathbb{R}_+^m \times \mathbb{R}_+^m$  sont les vecteurs de scaling obtenus par itération :

$$u^{(k+1)} = \frac{a}{Kv^{(k)}}, \quad v^{(k+1)} = \frac{b}{K^\top u^{(k+1)}} \quad (6)$$

où les divisions sont élément par élément. On initialise typiquement avec  $v^{(0)} = \mathbf{1}_m$  (vecteur de uns), et on itère jusqu'à convergence. L'algorithme converge vers la solution unique du problème régularisé.

---

### Algorithm 1 Algorithme de Sinkhorn

---

- 1: **Entrée** : Distributions  $a, b \in \mathbb{R}_+^m$ , matrice de coût  $C \in \mathbb{R}^{m \times m}$ ,  $\varepsilon > 0$
  - 2: **Initialisation** :  $v \leftarrow \mathbf{1}_m$
  - 3: **while** non convergence **do**
  - 4:      $u \leftarrow a \oslash (Kv)$   $\triangleright \oslash =$  division élément par élément
  - 5:      $v \leftarrow b \oslash (K^\top u)$
  - 6: **end while**
  - 7: **Retour** :  $\gamma = \text{diag}(u)K\text{diag}(v)$
- 

### Important

Dans ce TP, vous devez implémenter l'algorithme de Sinkhorn à la main (sans utiliser directement la fonction `ot.sinkhorn` de la bibliothèque POT). Cela vous permettra de mieux comprendre le fonctionnement interne de l'algorithme. Vous pourrez ensuite utiliser la bibliothèque POT pour valider votre implémentation et pour les fonctions auxiliaires (calcul de distances, etc.).

## 3 Inférence de trajectoires

### 3.1 Méthode Waddington-OT

La méthode Waddington-OT [1] utilise le transport optimal entropique pour reconstruire des trajectoires cellulaires. Entre deux snapshots consécutifs  $\hat{P}_{t_i}$  et  $\hat{P}_{t_{i+1}}$ , on calcule :

$$\gamma_{t_i, t_{i+1}}^* = \arg \min_{\pi \in \Pi(\hat{P}_{t_i}, \hat{P}_{t_{i+1}})} \{ \mathbb{E}_\pi[|X - Y|^2] + \sigma \Delta t_i H(\pi) \} \quad (7)$$

Ce couplage optimal peut ensuite être utilisé de deux façons :

1. **Chaînage stochastique** : Construire des trajectoires en échantillonnant successivement selon les couplages
2. **Interpolation de McCann** : Prédire les distributions intermédiaires

### 3.2 Interpolation de McCann

Étant données deux distributions  $\mu_0$  et  $\mu_1$ , et un plan de transport optimal  $\pi^*$ , l'*interpolation de McCann* (ou *géodésique de Wasserstein*) au temps  $t \in [0, 1]$  est définie par :

$$\mu_t := [(1 - t)X + tY]_\# \pi^* \quad (8)$$

où  $(X, Y) \sim \pi^*$  et  $f_{\#}\mu$  désigne la mesure image par  $f$ .

**Propriété géodésique** : Cette interpolation est une géodésique à vitesse constante pour la distance de Wasserstein-2 :

$$W_2(\mu_s, \mu_t) = |t - s| \cdot W_2(\mu_0, \mu_1) \quad (9)$$

Pour des distributions empiriques, l'implémentation consiste à :

1. Échantillonner  $N$  paires  $(x_i, y_j)$  selon le couplage optimal  $\gamma^*$
2. Calculer les points interpolés  $z_k = (1 - t)x_{i_k} + ty_{j_k}$  pour  $k = 1, \dots, N$
3. La distribution empirique  $\hat{\mu}_t = \frac{1}{N} \sum_{k=1}^N \delta_{z_k}$  est l'interpolation de McCann

### 3.3 Évaluation de la reconstruction

Pour évaluer la qualité de l'interpolation, on compare les distributions interpolées avec les vraies distributions intermédiaires (obtenues par simulation) en utilisant la distance de Wasserstein-2 :

$$W_2(\mu, \nu) = \left( \min_{\pi \in \Pi(\mu, \nu)} \int |x - y|^2 d\pi(x, y) \right)^{1/2} \quad (10)$$

Cette distance fournit une mesure quantitative de l'erreur de reconstruction.

## 4 Implémentations requises

Dans cette section, nous détaillons toutes les fonctions que vous devez implémenter dans le fichier `src/functions.py`.

### 4.1 Fonction 1 : Algorithme de Sinkhorn (À IMPLÉMENTER À LA MAIN)

#### À implémenter

**Fonction** : `compute_ot_coupling_manual`

**Description** : Implémenter l'algorithme de Sinkhorn pour calculer le couplage OT entropique entre deux distributions empiriques.

**IMPORTANT** : Cette fonction doit être implémentée **sans utiliser** `ot.sinkhorn`. Vous devez coder l'algorithme vous-même en suivant les équations présentées dans la Section 2.2.

**Signature** :

```
1 def compute_ot_coupling_manual(X_source, X_target, epsilon):
2     """
3     Implementer l'algorithme de Sinkhorn a la main.
4
5     Parameters
6     -----
7     X_source : ndarray, shape (n_source, d)
8         Particules sources
9     X_target : ndarray, shape (n_target, d)
10        Particules cibles
11    epsilon : float
12        Parametre de regularisation entropique
13
14    Returns
15    -----
16    gamma : ndarray, shape (n_source, n_target)
17        Plan de transport optimal (matrice de couplage)
18    """
```

#### Algorithme à suivre :

1. Créer les distributions uniformes  $a$  et  $b$  (vecteurs de poids)
2. Calculer la matrice de coût  $C$  (distances euclidiennes au carré)
3. Calculer le noyau de Gibbs  $K = \exp(-C/\varepsilon)$
4. Initialiser  $v = \mathbf{1}_m$
5. Itérer jusqu'à convergence :
  - $u = a \oslash (Kv)$
  - $v = b \oslash (K^\top u)$
  - Vérifier la convergence
6. Retourner  $\gamma = \text{diag}(u) \cdot K \cdot \text{diag}(v)$

#### Indices :

- Utilisez `scipy.spatial.distance.cdist` pour calculer  $C$

## 4.2 Fonction 2 : Validation avec POT

### À implémenter

**Fonction :** `compute_ot_coupling`

**Description :** Version utilisant la bibliothèque POT pour valider votre implémentation manuelle.

**Signature :**

```
1 def compute_ot_coupling(X_source, X_target, epsilon):
2     """
3     Calculer le couplage OT avec la bibliotheque POT.
4
5     Cette fonction sert de reference pour valider votre
6     implementation manuelle.
7
8     Parameters
9     -----
10    X_source : ndarray, shape (n_source, d)
11        Particules sources
12    X_target : ndarray, shape (n_target, d)
13        Particules cibles
14    epsilon : float
15        Parametre de regularisation entropique
16
17    Returns
18    -----
19    gamma : ndarray, shape (n_source, n_target)
20        Plan de transport optimal
21    """
```

#### Instructions :

- Utilisez `ot.sinkhorn` pour calculer le couplage
- Comparez les résultats avec votre implémentation manuelle

### 4.3 Fonction 3 : Construction de trajectoires

#### À implémenter

**Fonction :** build\_trajectories

**Description :** Construire des trajectoires en chaînant les couplages OT successifs.

**Signature :**

```
1 def build_trajectories(snapshots_dict, couplings,
2                       n_trajectories=100, seed=42):
3     """
4     Construire des trajectoires par chainage stochastique.
5
6     Parameters
7     -----
8     snapshots_dict : dict
9         Dictionnaire {temps: array de particules}
10    couplings : dict
11        Dictionnaire {(t_start, t_end): matrice de couplage}
12    n_trajectories : int
13        Nombre de trajectoires a construire
14    seed : int
15        Graine aleatoire pour la reproductibilite
16
17    Returns
18    -----
19    trajectories : list of list of ndarray
20        Liste de trajectoires, chaque trajectoire est une
21        liste de positions [pos_t0, pos_t1, ..., pos_tN]
22    times : list
23        Liste des temps correspondants
24    """
```

**Algorithme :**

1. Trier les temps dans l'ordre croissant
2. Pour chaque trajectoire :
  - (a) Échantillonner un indice de départ aléatoire au temps  $t_0$
  - (b) Pour chaque intervalle de temps  $[t_k, t_{k+1}]$  :
    - Récupérer le couplage  $\gamma_{t_k, t_{k+1}}$
    - Extraire la distribution conditionnelle  $\gamma[\text{idx\_current}, :]$
    - Normaliser pour obtenir une distribution de probabilité
    - Échantillonner le prochain indice selon cette distribution
  - (c) Stocker la séquence de positions
3. Retourner la liste des trajectoires et les temps

## 4.4 Fonction 4 : Interpolation de McCann

### À implémenter

**Fonction :** `mccann_interpolation`

**Description :** Calculer l'interpolation de McCann (géodésique de Wasserstein) au temps  $t$ .

**Signature :**

```
1 def mccann_interpolation(X_source, X_target, gamma, t,  
2                           n_samples=1000, seed=42):  
3     """  
4     Calculer l'interpolation de McCann au temps t.  
5  
6     Parameters  
7     -----  
8     X_source : ndarray, shape (n_source, d)  
9         Particules sources (au temps 0)  
10    X_target : ndarray, shape (n_target, d)  
11        Particules cibles (au temps 1)  
12    gamma : ndarray, shape (n_source, n_target)  
13        Plan de transport optimal  
14    t: float  
15        Temps d'interpolation (entre 0 et 1)  
16    n_samples : int  
17        Nombre d'échantillons à générer  
18    seed : int  
19        Graine aléatoire  
20  
21    Returns  
22    -----  
23    X_interp : ndarray, shape (n_samples, d)  
24        Particules interpolées au temps t  
25    """
```

**Algorithme :**

1. Aplatis la matrice de couplage  $\gamma$  en un vecteur
2. Normaliser pour obtenir une distribution de probabilité sur les paires  $(i, j)$
3. Échantillonner  $n$  paires  $(i_k, j_k)$  selon cette distribution
4. Pour chaque paire échantillonnée :
  - Calculer  $z_k = (1 - t) \cdot X_{\text{source}}[i_k] + t \cdot X_{\text{target}}[j_k]$
5. Retourner l'ensemble des points interpolés  $\{z_k\}$

## 4.5 Fonction 5 : Calcul de distance entre distributions

### À implémenter

**Fonction :** `distribution_distance`

**Description :** Calculer la distance de Wasserstein-2 entre deux distributions empiriques.

**Signature :**

```
1 def distribution_distance(X, Y, epsilon=0):  
2     """  
3     Calculer la distance W2 ou la divergence entropique.
```

```

4
5     Parameters
6     -----
7     X : ndarray, shape (n, d)
8         Premiere distribution empirique
9     Y : ndarray, shape (m, d)
10        Deuxieme distribution empirique
11     epsilon : float
12         Si 0: distance W2 exacte
13         Si > 0: divergence de Sinkhorn
14
15     Returns
16     -----
17     distance : float
18         Distance ou divergence entre les distributions
19     """

```

#### Instructions :

- Si  $\varepsilon = 0$  : utiliser `ot.emd2` pour calculer le transport optimal exact
- Si  $\varepsilon > 0$  : utiliser `ot.sinkhorn2` pour calculer la divergence entropique
- Retourner  $\sqrt{\text{coût}}$  pour obtenir la distance  $W_2$

**Note :** La divergence de Sinkhorn est définie par :

$$S_\varepsilon(\mu, \nu) = OT_\varepsilon(\mu, \nu) - \frac{1}{2}OT_\varepsilon(\mu, \mu) - \frac{1}{2}OT_\varepsilon(\nu, \nu)$$

## 5 Travail attendu

Le travail pratique à réaliser est structuré en deux notebooks Jupyter qui appellent vos fonctions implémentées dans `src/functions.py`.

### 5.1 Notebook 1 : Transport optimal entropique (45 minutes)

#### Objectifs :

1. **Simulation des données** : Générer des snapshots de particules suivant une EDS avec potentiel à branchement (code fourni)
2. **Implémentation et validation de Sinkhorn** :
  - Implémenter `compute_ot_coupling_manual`
  - Comparer avec `compute_ot_coupling` (version POT)
  - Vérifier que la différence est négligeable
3. **Visualisation** : Afficher la matrice de couplage et le plan de transport
4. **Étude paramétrique** : Analyser l'effet du paramètre  $\varepsilon$  sur :
  - La structure du couplage (sparsité)
  - Le coût de transport
  - L'entropie du couplage

### 5.2 Notebook 2 : Inférence de trajectoires (1h20)

#### Objectifs :

1. **Chaînage de couplages** :

- Implémenter `build_trajectories`
  - Visualiser les trajectoires reconstruites
  - Comparer avec les vraies trajectoires (si disponibles)
2. **Interpolation de McCann :**
- Implémenter `mccann_interpolation`
  - Visualiser les distributions interpolées
3. **Comparaison avec vérité terrain :**
- Calculer des snapshots denses par simulation
  - Implémenter `distribution_distance`
  - Mesurer l'erreur  $W_2$  entre interpolation et vérité terrain
  - Comparer pour différentes valeurs de  $\varepsilon$
4. **Analyse des résultats :**
- Identifier les régions temporelles difficiles (branchements)
  - Déterminer le  $\varepsilon$  optimal
  - Discuter les limitations de l'approche

## 6 Détails d'implémentation

### 6.1 Outils recommandés

- **Python 3.8+** avec les bibliothèques :
  - `numpy` : Calcul numérique
  - `matplotlib` : Visualisation
  - `scipy` : Distances et optimisation
  - **POT** (Python Optimal Transport) : Validation et fonctions auxiliaires
- **Jupyter Notebook** pour l'environnement interactif

Installation :

```
pip install numpy matplotlib scipy POT jupyter
```

### 6.2 Structure du code

Le repository contient :

- `src/functions.py` : **Fichier à compléter** avec vos implémentations
- `src/simulation.py` : Module de simulation des EDS (fourni)
- `notebooks/01_entropic_ot.ipynb` : Notebook partie 1 (fourni)
- `notebooks/02_trajectory_inference.ipynb` : Notebook partie 2 (fourni)

## 7 Évaluation et rendu

### 7.1 Livrables

À rendre avant le **12/02/2026** :

1. Fichier `src/functions.py` complété avec vos implémentations
2. Notebooks complétés (`.ipynb`) avec :
  - Toutes les cellules exécutées
  - Les visualisations générées

## 8 Pour approfondir

Pour les étudiants intéressés, voici quelques pistes d’approfondissement :

### 8.1 Optimisation sur les marginales (gWOT[4])

L’approche présentée suppose que les marginales  $\hat{P}_{t_i}$  sont fixées. En réalité, ces distributions empiriques sont bruitées (peu de cellules). La méthode gWOT [4] propose d’optimiser simultanément sur les marginales elles-mêmes :

$$\min_P \sigma H(P|W_\sigma) + \frac{1}{\lambda} \sum_{i=1}^N \Delta t_i H(\hat{P}_{t_i}|P_{t_i}) \quad (11)$$

où le terme de régularisation empêche l’overfitting.

### 8.2 Fused Gromov-Wasserstein

Pour prendre en compte la structure spatiale des tissus, on peut utiliser le transport de Gromov-Wasserstein [5] qui compare non seulement les expressions géniques mais aussi les relations de proximité entre cellules.

### 8.3 Données réelles

Appliquer la méthode à des datasets publics :

- **iPSC reprogramming** [1]
- **Mouse embryo development** [5]

### 8.4 Stabilité numérique

Étudier les problèmes de stabilité numérique de Sinkhorn :

- Implémentation en log-space (log-Sinkhorn)
- Utilisation de stabilisations numériques
- Comparaison des performances

## Références

- [1] G. Schiebinger, J. Shu, M. Tabaka, B. Cleary, V. Subramanian, A. Solomon, J. Gould, S. Liu, S. Lin, P. Berube, et al. *Optimal-transport analysis of single-cell gene expression identifies developmental trajectories in reprogramming*. Cell, 176(4) :928–943, 2019.
- [2] M. Cuturi. *Sinkhorn distances : Lightspeed computation of optimal transport*. Advances in Neural Information Processing Systems, 26, 2013.
- [3] C. Léonard. *A survey of the Schrödinger problem and some of its connections with optimal transport*. Discrete & Continuous Dynamical Systems-A, 34(4) :1533, 2013.
- [4] L. Chizat, S. Zhang, M. Heitz, G. Schiebinger. *Trajectory inference via mean-field Langevin in path space*. Advances in Neural Information Processing Systems, 35 :27276–27289, 2022.
- [5] D. Klein, G. Palla, M. Lange, C.-Y. Lin, L. Hetzel, S. Anchang, et al. *Mapping cells through time and space with moscot*. Nature, 2025.
- [6] G. Peyré, M. Cuturi. *Computational optimal transport : With applications to data science*. Foundations and Trends in Machine Learning, 11(5-6) :355–607, 2019.