TP 2 - GLOBAL WADDINGTON-OT

Optimisation des marginales pour l'inférence de trajectoires

Elias Ventre
contact: elias.ventre@inria.fr

Résumé

Dans ce deuxième TP, nous étendons l'approche de Waddington-OT en introduisant une méthode d'optimisation globale sur les marginales temporelles. Cette extension, appelée *global Waddington-OT* (gWOT), permet de reconstruire des trajectoires cellulaires même lorsque les données observées sont bruitées ou échantillonnées de manière sparse. Nous implémentons un algorithme d'optimisation dans l'espace des mesures utilisant la descente de gradient et évaluons la qualité de reconstruction sur des données simulées.

1 Introduction

1.1 Rappel : Waddington-OT avec marginales fixées

Dans le TP précédent, nous avons utilisé le transport optimal entropique pour reconstruire des trajectoires cellulaires en supposant que les distributions empiriques $\hat{P}_{t_i}$ observées aux différents temps étaient des estimations fiables des vraies marginales temporelles P_{t_i}.

La méthode consistait à :

1. Calculer les couplages optimaux entre paires de snapshots consécutifs
2. Chaîner ces couplages pour construire des trajectoires
3. Interpoler via l'interpolation de McCann

Limitation : Cette approche suppose que nous avons suffisamment de cellules à chaque temps pour estimer correctement les marginales. En pratique, les contraintes expérimentales conduisent souvent à un nombre faible de cellules à certains temps, donc des distributions empiriques bruitées et un risque d'overfitting aux observations.

1.2 Motivation pour gWOT

Considérons un scénario réaliste en biologie cellulaire :
— Au temps initial t_0 : 100 cellules (population initiale)
— Aux temps intermédiaires $t_1, \ldots, t_{N-1}$: peu de cellules (échantillonnage sparse)
— Au temps final t_N : 100 cellules (population finale)

Problème : Aux temps intermédiaires, les distributions empiriques $\hat{P}_{t_i}$ sont de très mauvaises approximations des vraies marginales P_{t_i}.

Solution : Au lieu de fixer les marginales aux observations bruitées, nous allons optimiser simultanément sur les marginales elles-mêmes en cherchant un compromis entre :

1. La fidélité aux données observées
2. La régularité du processus sous-jacent

1.3 Objectifs du TP

Ce travail pratique vise à :
— Comprendre la formulation du problème d'optimisation global sur les marginales
— Implémenter les composantes de la loss gWOT
— Optimiser dans l'espace des mesures via une descente de gradient en utilisant pytorch
— Évaluer la qualité de reconstruction en comparant avec une simulation de référence
— Analyser l'effet des paramètres de régularisation

1.4 Organisation du TP

Important

Le TP est structuré autour de deux fichiers principaux :
— `src/models_gwot.py` : Fichier où vous devez implémenter les classes de perte (voir Section 4)
— `pipeline_TP2.py` : Script fourni qui simule les données, appelle votre modèle, et visualise les résultats

Vous devrez également créer un notebook ou un document PDF pour présenter vos expériences et analyses.

2 Formulation mathématique

2.1 Le problème global gWOT

Au lieu de travailler avec des couplages indépendants entre paires de temps, gWOT considère le problème global de reconstruction de la mesure sur les chemins $\mathcal{P}$.

2.1.1 Mesure sur les chemins

Une mesure sur les chemins $\mathcal{P}$ est une distribution de probabilité sur l'espace des trajectoires continues $\mathcal{C}([t_0, t_1], \mathbb{R}^d)$. Pour une EDS :

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

la mesure $\mathcal{P}$ caractérise complètement le processus stochastique.

2.1.2 Marginales temporelles

Les marginales temporelles P_t sont définies par :

$$P_t = (\text{eval}_t)_{\#}\mathcal{P} \tag{2}$$

où $\text{eval}_t : \mathcal{C}([t_0, t_1], \mathbb{R}^d) \to \mathbb{R}^d$ est l'application d'évaluation au temps t.

2.2 Fonction objectif de gWOT

L'objectif de gWOT est de trouver la mesure sur les chemins $\mathcal{P}$ qui minimise :

$$\mathcal{F}(\mathcal{P}) = \underbrace{\sigma H(\mathcal{P}|W_\sigma)}_{\text{Terme de régularisation}} + \underbrace{\frac{1}{\lambda}\sum_{i=1}^{N} \Delta t_i H(\hat{P}_{t_i}|P_{t_i})}_{\text{Terme de fidélité aux données}} \tag{3}$$

où :

— $H(\mathcal{P}|W_\sigma)$: entropie relative par rapport au mouvement brownien de diffusion σ
— $H(\hat{P}_{t_i}|P_{t_i})$: entropie relative (divergence de Kullback-Leibler) entre l'observation et la marginale
— λ : paramètre de régularisation contrôlant le compromis
— $\Delta t_i = t_{i+1} - t_i$: pas de temps

Proposition 1 (Lavenant et al. 2021)**.** *Sous des hypothèses techniques appropriées :*

$$\lim_{\lambda \to 0, N \to \infty} \mathcal{P}^{\lambda,N} = \mathcal{P} \tag{4}$$

La convergence a lieu indépendamment du nombre de cellules dans les observations $\hat{P}_{t_i}$!

2.3 Décomposition en marginales optimales

En pratique, on ne travaille pas directement avec $\mathcal{P}$ mais on cherche la séquence de marginales optimales $(P^*_{t_1}, \ldots, P^*_{t_N})$.

Le problème peut se réécrire comme :

$$\min_{(P_{t_1},\ldots,P_{t_N})} \left\{ \sum_{i=1}^{N-1} \sigma \Delta t_i \, \mathrm{OT}_{\sigma \Delta t_i}(P_{t_i}, P_{t_{i+1}}) + \frac{1}{\lambda} \sum_{i=1}^{N} \Delta t_i \, \mathrm{KL}(\hat{P}_{t_i} \| \Phi_h * P_{t_i}) \right\} \tag{5}$$

où $\varepsilon = \sigma \Delta t_i$ et OT_ε désigne le coût de transport optimal entropique et Φ_h un noyau Gaussien de variance h.

2.4 Interprétation des termes

Terme de régularisation $\sum_{i=1}^{N-1} \sigma \Delta t_i \, \mathrm{OT}_{\sigma \Delta t_i}(P_{t_i}, P_{t_{i+1}})$:
— Pénalise les trajectoires qui s'éloignent du mouvement brownien
— Force les marginales consécutives à être proches au sens du transport optimal
— Assure la régularité temporelle du processus

Terme de fidélité $\frac{1}{\lambda} \sum_{i=1}^{N} \Delta t_i \, \mathrm{KL}(\hat{P}_{t_i} \| \Phi_h * P_{t_i})$:
— Force les marginales optimisées à rester proches des observations
— Évite que le modèle ne s'éloigne trop des données
— Contrôlé par le paramètre λ et la variance h pour lisser les données et pouvoir les estimer sur le même support que les marginales à optimiser.

Rôle du paramètre λ :
— λ petit : forte fidélité aux données (risque d'overfitting)
— λ grand : forte régularisation (peut sous-ajuster)
— Trade-off classique biais-variance

3 Optimisation dans l'espace des mesures

3.1 Représentation particulaire des mesures

Pour optimiser sur les mesures de probabilité, nous utilisons une représentation empirique par particules :

$$P_t \approx P_t^m = \frac{1}{m} \sum_{j=1}^{m} \delta_{X_j^t} \tag{6}$$

où $\{X_j^t\}_{j=1}^m$ est un ensemble de m particules au temps t.

3.2 Descente de gradient en champ moyen

L'optimisation dans l'espace des mesures peut être réalisée via une descente de gradient sur les positions des particules.

Pour un fonctionnel $F : \mathcal{P}(\mathbb{R}^d) \to \mathbb{R}$, la dérivée fonctionnelle $\frac{\delta F}{\delta \mu}(x)$ définit le gradient au point x.

La descente de gradient en champ moyen s'écrit :

$$X_j^{k+1} = X_j^k - \eta \nabla \frac{\delta F}{\delta \mu}(\mu_k)(X_j^k) + \sqrt{2\eta}\xi_j^k \tag{7}$$

où :
— η : taux d'apprentissage
— $\xi_j^k \sim \mathcal{N}(0, I_d)$: bruit stochastique

3.3 Calcul des gradients avec PyTorch

En pratique, nous utilisons PyTorch et l'optimiseur Adam pour calculer automatiquement les gradients :

Algorithm 1 Optimisation gWOT avec Adam

Entrée : Observations $\{\hat{P}_{t_i}\}_{i=1}^N$, paramètres λ, σ, ε
Initialisation : Pour chaque temps t_i, initialiser m particules près de $\hat{P}_{t_i}$
for $k = 1$ to n_{epochs} **do**
    Calculer $\mathcal{L} = \mathcal{L}_{\text{reg}} + \mathcal{L}_{\text{fit}}$
    Calculer $\nabla_{\{X_j^{t_i}\}} \mathcal{L}$ via autograd
    Mettre à jour les particules avec Adam
end for
Retour : Particules optimisées $\{X_j^{t_i}\}$

3.4 Différentiabilité du transport optimal entropique

Un aspect clé est que le transport optimal entropique (Sinkhorn) est différentiable par rapport aux distributions d'entrée. Cela permet de propager les gradients à travers les calculs de OT.

La bibliothèque `geomloss` implémente efficacement cette différentiabilité, permettant :
— Calcul du coût OT entropique
— Rétropropagation automatique des gradients
— Scalabilité à de grandes distributions

4 Implémentations requises

Dans cette section, nous détaillons les classes que vous devez implémenter dans le fichier `src/models_gwot.py`.

4.1 Classe 1 : PathsLoss

À implémenter

Classe : `PathsLoss`
Description : Calculer le terme de régularisation basé sur le transport optimal entre marginales consécutives.
Formule :

$$\mathcal{L}_{\text{reg}} = \sum_{i=1}^{N-1} \varepsilon \cdot \text{Sinkhorn}(P_{t_i}, P_{t_{i+1}})) \tag{8}$$

avec $\varepsilon = \sigma \Delta t_i$.

Structure de la classe :

```
class PathsLoss(nn.Module):
    def __init__(self, x_list, eps, device="cpu"):
        """
        Initialize path regularization loss.

        Parameters
        ----------
        x_list : list of torch.Tensor
            List of particle positions at each time
            x_list[i] has shape (n_particles, dim)
        eps : float
            Entropic regularization parameter (sigma * dt)
        device : str
            Computation device
        """

    def forward(self):
        """
        Compute the path regularization loss.

        Returns
        -------
        loss : torch.Tensor
            Total regularization loss (scalar)
        """
```

Indices d'implémentation :
— Utilisez `SamplesLoss` de `geomloss` :
`SamplesLoss(loss="sinkhorn", p=2, blur=` $\sqrt{\sigma \Delta_{t_i}}$`, debias=False, scaling=0.95)`
— Bouchez sur les temps consécutifs et sommez les coûts OT
— Multipliez la somme finale par `eps=`$\sigma \Delta_{t_i}$

4.2 Classe 2 : FitLoss

À implémenter

Classe : `FitLoss`
Description : Calculer le terme de fidélité aux données pour un temps donné.
Formule : Pour un temps t, le terme de fidélité utilise une structure de type log-vraisemblance gaussienne :

$$\mathcal{L}_{\text{fit}}^{(t)} = \text{KL}(\hat{P}_{t_i} \| \Phi_{\sigma_{\text{fit}}} * P_{t_i}) := -\sum_{j=1}^{n_{\text{obs}}} \log \left(\frac{n_{\text{obs}}}{m} \sum_{i=1}^{m} \cdot \mathcal{N}(y_j | x_i, \sigma_{\text{fit}}^2 I) \right) \cdot \frac{1}{n_{\text{obs}}} \tag{9}$$

où :
- $\{x_i\}$: particules du modèle
- $\{y_j\}$: observations
- σ_{fit} : écart-type du terme de fidélité

En utilisant la formulation log-sum-exp stable :

$$C_{ij} = -\frac{\|x_i - y_j\|^2}{2\sigma_{\text{fit}}^2} - \log(\sum_{j=1}^{n_{\text{obs}}} \exp(-\frac{\|x_i - y_j\|^2}{2\sigma_{\text{fit}}^2})) \tag{10}$$

$$\mathcal{L}_{\text{fit}}^{(t)} = -\log(\frac{n_{\text{obs}}}{m}) - \frac{1}{n_{\text{obs}}} \sum_{j=1}^{n_{\text{obs}}} \log \sum_{i=1}^{m} \exp(C_{ij}) \tag{11}$$

Structure de la classe :

```
class FitLoss(nn.Module):
    def __init__(self, x, y, sigma, device="cpu"):
        """
        Initialize data fitting loss for one timepoint.

        Parameters
        ----------
        x : torch.Tensor, shape (n_model, dim)
            Model particle positions
        y : torch.Tensor, shape (n_obs, dim)
            Observed particle positions
        sigma : float
            Standard deviation for Gaussian likelihood
        device : str
            Computation device
        """

    def forward(self):
        """
        Compute data fitting loss.

        Returns
        -------
        loss : torch.Tensor
            Negative log-likelihood (scalar)
        """
```

Indices d'implémentation :
— Calculez la matrice de distances carrées avec `torch.cdist(x, y, p=2)**2`
— Formez $C_{ij} = -\frac{C_{\text{dist}}}{2\sigma^2}$ + facteur d'échelle (attention à la shape)
— Utilisez `torch.logsumexp(C, dim=0)` pour stabilité numérique
— Prenez la moyenne sur les observations
— N'oubliez pas le signe négatif !

4.3 Classe 3 : TrajLoss

À implémenter

Classe : `TrajLoss`
Description : Combinaison de `PathsLoss` et `FitLoss` pour tous les temps.
Formule :

$$\mathcal{L}_{\text{total}} = \lambda_{\text{reg}} \cdot \mathcal{L}_{\text{reg}} + \lambda_{\text{fit}} \cdot \sum_{i=1}^{N} \mathcal{L}_{\text{fit}}^{(t_i)} \tag{12}$$

Structure de la classe :

```
class TrajLoss(nn.Module):
    def __init__(
        self,
        x0_list,
        obs_list,
        lam_reg,
        lam_fit,
        eps,
        sigma_fit=1.0,
        device="cpu",
    ):
        """
        Initialize global trajectory loss (gWOT).

        Parameters
        ----------
        x0_list : list of torch.Tensor
            Initial particle positions for each timepoint
        obs_list : list of torch.Tensor
            Observed particles at each timepoint
        lam_reg : float
            Regularization weight
        lam_fit : float
            Data fitting weight
        eps : float
            Entropic regularization (sigma * dt)
        sigma_fit : float
            Std for data fitting term
        device : str
            Computation device
        """

    def forward(self):
```

```
"""
Compute total loss.

Returns
-------
loss : torch.Tensor
    Total loss (scalar)
"""
```

Indices d'implémentation :
— Stockez les positions comme `nn.ParameterList` pour l'optimisation
— Dans `forward` :

1. Calculez `PathsLoss` sur toutes les particules
2. Pour chaque temps, calculez `FitLoss`
3. Sommez avec les poids appropriés

— Attention : les poids des particules doivent être uniformes ($1/m$)
— Vous pouvez ajouter un facteur d'échelle pour équilibrer les termes

5 Travail attendu

5.1 Implémentation ($\sim$ 2h)

Fichier `src/models_gwot.py` :

1. Implémenter la classe `PathsLoss`
2. Implémenter la classe `FitLoss`
3. Implémenter la classe `TrajLoss`

La fonction `optimize_model` vous est fournie et ne nécessite pas de modification.

5.2 Expériences et analyse (à rendre)

Créez un notebook Jupyter ou un document PDF présentant quelques éléments d'analyse des résultats :

5.2.1 Validations de base (tous)

1. Exécutez `pipeline_TP2.py` avec les paramètres par défaut
2. Vérifiez que l'optimisation converge (perte décroissante)
3. Visualisez les trois panels :
   — Données sparse (input)
   — Données inférées par gWOT
   — Données de référence (full simulation)
4. Commentez les distances W2 entre :
   — Inferred vs Full
   — Sparse vs Full

5.2.2 Étude paramétrique

1. Influence de λ_{reg}
Testez différentes valeurs : `lam_reg` $\in \{0.01, 0.1, 1.0, 10.0\}$
— Observez l'effet sur la régularité des trajectoires

— Mesurez la distance W2 moyenne
— Quelle valeur donne le meilleur compromis ?

2. Influence du nombre de temps N

Testez : `n_times` $\in \{5, 10, 20\}$
— Comment la qualité de reconstruction évolue-t-elle ?
— Quel est l'impact sur le temps de calcul ?

3. Influence de la sparsité des données

Modifiez le nombre de cellules aux temps intermédiaires :
— `n_sparse[1:-1]` $\in \{5, 10, 20, 50\}$
— Quelle est la sparsité minimale pour une bonne reconstruction ?
— Vérifiez que gWOT est robuste au bruit d'échantillonnage

5.2.3 Mise en relation avec le premier TP

Regarder l'influence des l'inférence des marginales sur la qualité des trajectoires et interpolations de McCann réalisées à partir des séries temporelles. Illustrer par exemple avec les fonctions codées lors du premier TP.

6 Références

1. Lavenant, H., Zhang, S., Kim, Y. H., & Schiebinger, G. (2021). Towards a mathematical theory of trajectory inference. *arXiv preprint arXiv :2102.09204*.
2. Chizat, L., Zhang, S., Heitz, M., & Schiebinger, G. (2022). Trajectory inference via mean-field Langevin in path space. *Advances in Neural Information Processing Systems*, 35, 27276-27289.
3. Schiebinger, G., Shu, J., Tabaka, M., et al. (2019). Optimal-transport analysis of single-cell gene expression identifies developmental trajectories in reprogramming. *Cell*, 176(4), 928-943.