Compte rendu de stage

Garance Malnoë
Stage sous la direction de Sylvain Faure
Institut de Mathématiques d'Orsay

Mai - Juin 2024

Table des matières

1 Introduction

Une super introduction.

2 Passage de données eulériennes à lagrangiennes

[Maury(2024)] Les termes "eulérien" et "lagrangien" font référence à deux manières d'appréhender des phénomènes de transport ou de mouvement. Ils sont couramment utilisés en mécanique des fluides, mais ils peuvent plus généralement s'appliquer à toutes les entités évoluant dans un espace (un espace physique comme un espace plus abstrait). Ils permettent de qualifier à la fois la manière dont peuvent être recueillies des données de positions ou d'état des entités considérées et l'approche adoptée pour écrire des modèles visant à décrire ces phénomènes de mouvement.

2.1 Données lagrangiennes

La description lagrangienne consiste à suivre les entités tout au long de leur mouvement à partir de leur position d'origine.

Si l'on considère, par exemple, un ensemble de N particules évoluant dans l'espace physique $\mathbb{R}^d$, la donnée de leurs positions au cours du temps $t \mapsto x(t) = (x_i(t))_{i\in[\![1;N]\!]} \in \mathbb{R}^d$ est lagrangienne. De la même manière, la collection des vitesses au cours du temps des N particules $(u_i(t))_{i\in[\![1;N]\!]}$ où $u_i = x_i'(t)$ est une donnée lagrangienne.

Par la suite, nous nous intéresserons aux déplacements d'individus sur un territoire donné. Pour cela, nous pouvons, par exemple, récupérer les données GPS d'un téléphone portable au cours des déplacements de son propriétaire. Ces données sont de nature lagrangiennes : même si le droit ne permet pas de les stocker avec l'identité de la personne suivie, un identifiant d'anonymisation permet d'associer plusieurs positions successives à un seul individu.

2.2 Données eulériennes

La description eulérienne est basée sur l'observation de zones de l'espace dans lequel le mouvement s'effectue. On regarde "ce qu'il se passe" dans la zone sans prendre en compte l'identité des entités observées.

En se replaçant dans le cadre de l'étude des déplacements sur un territoire, la mesure du nombre de passages à l'entrée ou la sortie d'une gare au cours du temps ou du nombre de voitures sur un tronçon de route au cours du temps sont des données de nature eulérienne.

2.3 Passage de l'un à l'autre

Après avoir récupéré des données de nature lagrangienne ou eulérienne, il peut être pertinent de se demander s'il est possible de transformer ces données en données de l'autre nature.

Il est souvent possible de passer de la vision lagrangienne à la vision eulérienne. Faisons l'exemple avec la modélisation des trajectoires de N particules dans un espace donné. On introduit la notion de masse ponctuelle : à une particule située en un point x de l'espace, on associe la mesure δ_x qui est une application de l'ensemble des parties de l'espace dans $\mathbb{R}_+$ (ou plus simplement $\mathbb{N}$ si on se limite à compter des entités). Pour $A \subset \mathbb{R}^d$, on dit que $\delta_x(A)$ vaut 1 si $x \in A$, 0 sinon. Si deux particules se trouvent au même endroit, on applique une règle de sommation : $\delta_x + \delta_x = 2\delta_x$. On peut alors sommer ces objets pour obtenir une mesure qui encode l'ensemble des positions des particules au temps t :

$$\mu_t = \sum_{i=1}^{N} \delta_{x_i(t)}$$

Cette identité exprime que, pour tout ensemble A, $\mu_t(A)$ est le nombre d'entités qui se trouvent dans la zone A à l'instant t.

Cependant, bien que μ_t soit définie à partir des $x_i(t)$, l'information contenue dans μ est dégradée par rapport à la collection (x_i) puisque les labels ont été perdus dans la somme : on ne sait plus qui est où. Plus précisément, si les positions sont distinctes deux à deux, une même mesure μ_t correspond à un grand nombre de distributions possibles des entités. Formellement, pour toute permutation $\varphi \in S_N$, on a :

$$\sum_{i=1}^{N} \delta_{x_i(t)} = \sum_{i=1}^{N} \delta_{x_{\varphi(i)}(t)}$$

Ce manque d'information dans le point de vue eulérien empêche souvent le passage du point de vue eulérien au point de vue lagrangien. C'est un problème mal posé, plusieurs scénarios lagrangiens correspondent aux données eulériennes et il n'y a *a priori* pas un scénario qui soit plus plausible que les autres à moins de rajouter des hypothèses.

Une autre différence importante entre les deux points de vue se trouve au niveau de la récolte des données. Reprenons le cas de déplacements au sein d'un territoire donné. Pour récupérer des données eulériennes, il est possible d'installer des capteurs infrarouges à chaque porte des trains circulant sur le territoire pour compter les montées et les descentes des voyageurs et il est également possible de compter le nombre de voitures sur un tronçon de route à l'aide des caméras placées sur les routes. Pour récupérer des données lagrangiennes, on peut suivre les usagers tout au long de leur déplacement à l'aide des données GPS de leur téléphone ou bien mettre en place une enquête pour demander directement aux gens quel parcours ils ont empruntés comme cela avait été fait à Rennes en 2023 mais cela demande des dispositifs plus conséquents que pour les données eulériennes. Si bien que dans la réalité, il parfois plus simple de récupérer des données eulériennes que des données lagrangiennes.

Ainsi, dans la suite de cette section, nous nous intéresserons au passage du point de vue eulérien vers le point de vue lagrangien pour la modélisation des déplacements sur un réseau.

3 Réseau linéaire : exemple d'une ligne de bus

3.1 Cadre théorique

Nous nous intéressons ici à une ligne de bus avec $N+1$ arrêts numérotés de 0 à N et on s'intéresse seulement au sens croissant des arrêts. Nous supposons que nous avons pu récupérer des données eulériennes sur les déplacements des usagers de la ligne : à chaque arrêt $i \in [\![0; N]\!]$, on a compté le nombre de montées μ_i, le nombre de descentes ν_i et le nombre de personnes à voyager entre l'arrêt i et l'arrêt $i+1$, ρ_i. On note alors $\mu, \nu \in \mathbb{R}^{N+1}$ et $\rho \in \mathbb{R}^N$ les vecteurs associés et on considère que $\nu_0 = \mu_N = 0$. Enfin, on peut introduire la quantité $b \in \mathbb{R}^{N+1}$ où $b_i = \mu_i - \nu_i$ le bilan de montée-descente de l'arrêt i.

L'objectif est de retrouver les déplacements de chaque usager à partir de μ, ν et ρ. On souhaite savoir combien de personnes sont montées à l'arrêt i pour descendre à l'arrêt $j = \gamma_{ij}$. On note $\gamma = (\gamma_{ij})$ le plan origine-destination. On définit également le plan origine-destination normalisée $\tilde{\gamma}$ par :

$$\tilde{\gamma}_{ij} = \frac{\gamma_{ij}}{\mu_i} \qquad i < N, \quad i < j \leq N$$

On note Γ l'ensemble des plans Originie-Destination γ qui vérifient les données de montées et des descentes, μ et ν :

$$\Gamma = \{\gamma \quad | \quad \sum_{i=0}^{N} \gamma_{ij} = \nu_j \text{ et } \sum_{j=0}^{N} \gamma_{ij} = \mu_i\}$$

Ainsi défini, $\gamma_i = (\gamma_{ij})_j$ est une loi de probabilité supportée par $[\![i+1; N]\!]$ et qui décrit la distribution des destinations des voyageurs montés à l'arrêt i.

Tout d'abord, on peut s'intéresser aux liens entre les différentes quantités eulériennes : ρ, ν, μ et b.

Proposition :

$$\forall i \in [\![0; N-1]\!], \quad \rho_i = \sum_{j=0}^{i} bj$$

Démonstration :
On procède par récurrence.
- Initialisation : au départ de la première station, on a :

$$\rho_0 = \mu_0 = \mu_0 - \nu_0 = b_0$$

- Hérédité : supposons l'hypothèse vraie pour un $i \in [\![0; N-1]\!]$. Montrons qu'elle est vraie à l'étape $i+1$:
ρ_{i+1} = les voyageurs entre i et $i+1$ + les voyageurs montants en $i+1$ - les voyageurs descendants en $i+1 = \rho_i + \mu_{i+1} - \nu_{i+1} = \sum_{j=0}^{i+1} bj$.

Proposition : On fixe $\rho_{-1} = \rho_N = 0$. On a alors :

$$\forall i \in [\![0; N]\!], \quad b_i = \rho_i - \rho_{i-1}$$

Démonstration :
b_i = la différence entre les voyageurs partant de i et ceux partis de $i-1$ arrivés en $i = \rho_{i-1} - \rho_i$.

On a également une relation entre γ, μ et b sous la forme $G\mu = b$ où G est une matrice qui dépend du plan normalisé $\tilde{\gamma}$.

Proposition :

$$\begin{pmatrix} \sum_{j=1}^{N} \tilde{\gamma}_{0j} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ -\tilde{\gamma}_{01} & \sum_{j=2}^{N} \tilde{\gamma}_{1j} & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & 0 & \cdots & \cdots & \vdots \\ -\tilde{\gamma}_{0i} & \cdots & -\tilde{\gamma}_{i-1i} & \sum_{j=i+1}^{N} \tilde{\gamma}_{ij} & 0 & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \cdots & \vdots \\ -\tilde{\gamma}_{0N-1} & \cdots & \cdots & \cdots & \cdots & \tilde{\gamma}_{N-1N} & 0 \\ -\tilde{\gamma}_{0N} & \cdots & \cdots & \cdots & \cdots & -\tilde{\gamma}_{N-1N} & 0 \end{pmatrix} \begin{pmatrix} \mu_0 \\ \vdots \\ \vdots \\ \mu_i \\ \vdots \\ \mu_{N-1} \\ \mu_N \end{pmatrix} = \begin{pmatrix} b_0 \\ \vdots \\ \vdots \\ b_i \\ \vdots \\ b_{N-1} \\ b_N \end{pmatrix}$$

Démonstration :

$$b_i = \mu_i - \nu_i = \sum_{j=i+1}^{N} \gamma_{ij} - \sum_{j=0}^{i-1} \gamma_{ji} = \sum_{j=i+1}^{N} \tilde{\gamma}_{ij} \times \mu_i - \sum_{j=0}^{i-1} \tilde{\gamma}_{ji} \times \mu_j = (\sum_{j=i+1}^{N} \tilde{\gamma}_{ij}) \times \mu_i - \sum_{j=0}^{i-1} \tilde{\gamma}_{ji} \times \mu_j$$

Mais avec nos données eulériennes, on ne connait pas précisément γ ni $\tilde{\gamma}$, néanmoins nous avons plusieurs informations. On se concentrera sur γ :

- On étudie la ligne de bus dans un seul sens (ici dans le sens des indices croissants). Donc pour tout $i \in [\![0; N]\!]$ et pour tout $j < i$, $\gamma_{ij} = 0$. La partie inférieure gauche de la matrice est nulle.
- Un usager ne peut pas monter et descendre à la même station, la diagonale est donc nulle.
- Pour toute colonne d'indice j, $\sum_{i=0}^{N} \gamma_{ij} = \nu_j$. On peut en déduire que $\gamma_{01} = \nu_1$.
- Pour toute ligne d'indice i, $\sum_{j=0}^{N} \gamma_{ij} = \mu_i$. On peut en déduire que $\gamma_{N-1N} = \mu_N$.

On a donc une matrice de la forme suivante :

$$\gamma = \left(\begin{array}{cccccc|c} 0 & \nu_1 & \gamma_{02} & \cdots & \cdots & \gamma_{0N} & \sum \gamma_{0j} = \mu_0 \\ \vdots & \ddots & \gamma_{12} & \ddots & & \vdots & \sum \gamma_{1j} = \mu_1 \\ \vdots & & \ddots & \ddots & \ddots & \vdots & \sum \gamma_{ij} = \mu_j \\ \vdots & & & \ddots & \gamma_{N-2N-1} & \gamma_{N-2N} & \sum \gamma_{N-2j} = \mu_{N-2} \\ \vdots & & & & \ddots & \mu_{N-1} & \sum \gamma_{N-1j} = \mu_{N-1} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & \sum \gamma_{Nj} = \mu_N \\ \hline \sum \gamma_{i0} = \nu_0 & \sum \gamma_{i1} = \nu_1 & \sum \gamma_{i2} = \nu_2 & \sum \gamma_{ij} = \nu_j & \sum \gamma_{iN-1} = \nu_{N-1} & \sum \gamma_{iN} = \nu_N & \end{array}\right)$$

Pour trouver les valeurs des termes γ_{ij} restants, on peut mettre en place différents algorithmes, nous allons en voir deux.

3.2 Algorithme naïf

Le premier algorithme mis en place a été un algorithme "naturel", celui fait pour se faire la main sur un petit exemple avec seulement quelques arrêts et de petites valeurs. On suppose que les valeurs de μ, ν, ρ sont valides : il y a au moins une matrice γ qui correspond à ces données.

Voici un exemple pour une ligne de bus à 5 arrêts.

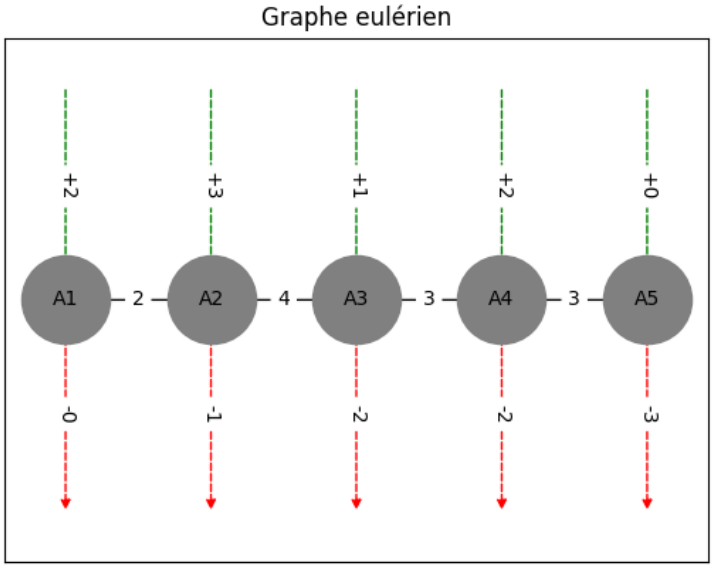

FIGURE 1 – Représentation ligne de bus à 5 arrêts à partir de données eulériennes

On a : $\mu = (2,\ 3,\ 1,\ 2,\ 0)$, $\quad \nu = (0,\ 1,\ 2,\ 2,\ 3)$, $\quad \rho = (2,\ 1,\ 2,\ 2,\ 3)$

Algorithm 1 Algorithme de remplissage de matrice

Initialiser une matrice M remplie de 0.
if la matrice est correctement remplie **then**
La matrice nulle est la seule solution, arrêt.
else
Faire une hypothèse sur γ_{02}, commencer par $\gamma_{02} = 0$.
(*) On regarde ce que cela implique : y a-t-il des cases dont on est sûr de connaitre la valeur à partir des hypothèses sur la valeur de la somme des colonnes et des lignes ? On remplit alors ces cases avec les valeurs correspondantes en faisant attention aux contradictions (ex : une case doit valoir 2 pour que la condition sur la somme de la ligne soit vérifiée mais elle doit valoir 3 pour que la condition sur la colonne soit vérifiée).
if on a une contradiction **then**
On revient à l'hypothèse précédemment faite (et on enlève toutes les implications que cette hypothèse avait entrainées) et si on le peut, on augmente de 1 la valeur de l'hypothèse et on reprend à (*). Si ça n'est pas possible, on revient encore à l'hypothèse d'avant et ainsi de suite jusqu'à ce qu'on puisse augmenter une hypothèse de 1 pour reprendre à (*) et si on ne peut pas trouver de telle hypothèse, arrêt.
else
if la matrice est correctement remplie **then**
On sauvegarde la matrice
On revient à l'hypothèse précédemment faite (et on enlève toutes les implications que cette hypothèse avait entrainées) et si on le peut, on augmente de 1 la valeur de l'hypothèse et on reprend à (*). Si ça n'est pas possible, on revient encore à l'hypothèse d'avant et ainsi de suite jusqu'à ce qu'on puisse augmenter une hypothèse de 1 pour reprendre à (*) et si on ne peut pas trouver de telle hypothèse, arrêt.
else
On fait une nouvelle hypothèse sur la prochaine case non connue (on progresse ligne par ligne et colonne par colonne) et on reprend à (*).
end if
end if
end if

La première matrice γ trouvée par l'algorithme est la suivante :

$$\gamma_1 = \left(\begin{array}{ccccc|c} 0 & 1 & 0 & 0 & 1 & 2 \\ 0 & 0 & 2 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline -- & -- & -- & -- & -- & \\ 0 & 1 & 2 & 2 & 3 & \end{array}\right)$$

— On suppose que γ_{02} vaut 0. Or, $\gamma_{02} + \gamma_{12} = 2$ donc γ_{12} vaut 2.

La prochaine case inconnue est 03.

— On suppose que γ_{03} vaut 0.
Or, $\gamma_{03} + \gamma_{04} = 1$ donc γ_{04} vaut 1.
Or, $\gamma_{04}+\gamma_{14}+\gamma_{24}+\gamma_{34} = 3$ donc $\gamma_{14} = \gamma_{24} = 0$.
Or, $\gamma_{12} + \gamma_{13} + \gamma_{14} = 3$ donc γ_{13} vaut 1.
Or, $\gamma_{03} + \gamma_{13} + \gamma_{23} = 2$ donc $\gamma_{23} = 1$

Voici les 4 autres matrices γ qui vérifient les conditions sur μ, ν et ρ :

$$\gamma_2 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \gamma_3 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\gamma_4 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \gamma_5 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

3.3 Algorithme de recherche exhaustive

Le deuxième algorithme est un algorithme de recherche exhaustive (backtracking) qui est assez proche de l'algorithme naïf. Il consiste à explorer toutes les configurations possibles et à retenir celles qui satisfont les conditions. Voici le code Python de l'algorithme, on peut retrouver le code entier en annexe du rapport ou sur GitHub.

```
def euler_to_lagrange(m, v):
    """
    :param m: une liste d'entiers, les montees a chaque arret
    :param v: une liste d'entiers, les descentes a chaque arret
    :return: une liste contenant toutes les matrices OD correspondant aux
        donnees de montees et de descentes
    """
    n = len(m)
    resultats = []

    # Fonction recursive pour trouver les grilles
    def backtrack(grille, m_rest, v_rest, ligne, col):
        # Cas 1 : On a rempli la derniere cellule. On verifie si la grille est
```

```
    valide
#       Si elle est valide, on la sauvegarde dans les resultats.
if ligne == n:
    if all(sum(grille[i]) == m[i] for i in range(n)) and all(
            sum(grille[i][j] for i in range(n)) == v[j] for j in
                range(n)):
        resultats.append([row[:] for row in grille])
    return

# Cas 2 : On a rempli la derniere colonne de la ligne actuelle. On
    passe a la ligne suivante
if col == n:
    # On appelle la fonction backtrack pour continuer a remplir la
        matrice
    # en se placant sur la ligne +1 et sur la premiere colonne
    backtrack(grille, m_rest, v_rest, ligne + 1, 0)
    return

# Cas 3 : On est sur la diagonale ou la partie inferieure gauche, la
    valeur doit etre 0.
if ligne >= col:
    grille[ligne][col] = 0 # On met le coef a 0
    backtrack(grille, m_rest, v_rest, ligne, col + 1) # On continue a
        remplir la grille.

# Autre cas : On appelle la fonction backtrack pour chaque valeur que
    peut prendre la cellule
else:
    # On definit la valeur maximale que peut prendre la cellule a
        partir de m_rest et v_rest.
    max_val = min(m_rest[ligne], v_rest[col])
    for val in range(max_val + 1):
        grille[ligne][col] = val
        # On modifie les valeurs de m_rest et v_rest pour remplir les
            cellules restantes
        # Et respecter les conditions sur les sommes de lignes et de
            colonnes
        m_rest[ligne] -= val
        v_rest[col] -= val

        # On appelle la fonction backtrack recursive pour remplir la
            cellule suivante
        backtrack(grille, m_rest, v_rest, ligne, col + 1)

        # On retablie les valeurs de m_rest et v_rest pour tester la
            nouvelle possibilite
        m_rest[ligne] += val
        v_rest[col] += val
        grille[ligne][col] = 0
```

```
    # On initialise une grille vide
    grille_vide = [[0] * n for _ in range(n)]

    # On appelle la fonction recursive backtrack pour remplir la grille et
        tester les possibilites.
    backtrack(grille_vide, m.copy(), v.copy(), 0, 0)

    return resultats
```

Pour le réseau de bus à 5 arrêts (1), on retrouve bien les mêmes cinq matrices Origine-Destination qu'avec l'algorithme naïf.

On a également pu tester l'algorithme sur d'autres petites lignes de bus. Un autre exemple avec 6 arrêts (2) et des valeurs un tout petit peu plus grandes :

$$\mu = (5,\ 4,\ 6,\ 3,\ 1,\ 0), \quad \nu = (0,\ 2,\ 4,\ 3,\ 5,\ 5)$$

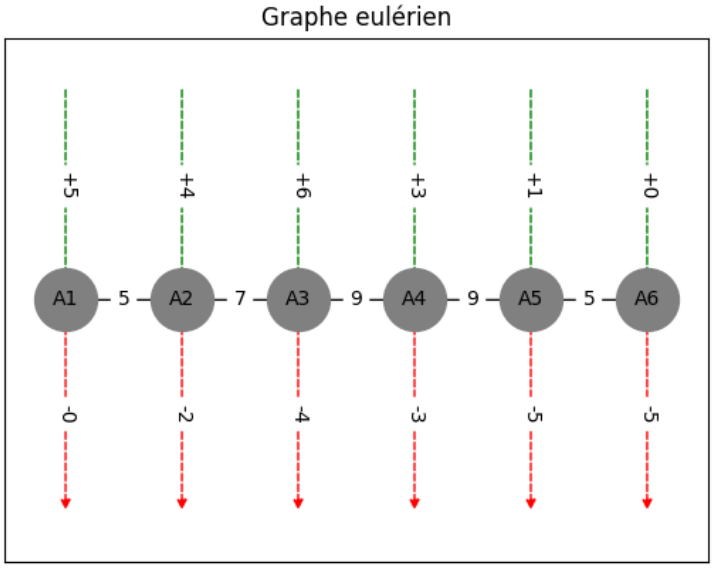

FIGURE 2 – Représentation ligne de bus à 6 arrêts à partir de données eulériennes

Avec l'algorithme exhaustif, on obtient 200 matrices Origine-Destination qui correspondent aux données de montées et descentes. On voit que le nombre de matrices augmente très rapidement dès lors que l'on augmente le nombre d'arrêts et les valeurs dans les vecteurs μ et ν.

L'augmentation des valeurs et du nombre d'arrêts entraîne également l'augmentation du nombre de possibilités de matrices et donc le temps de calcul. La complexité de l'algorithme exhaustif est exponentielle dans le pire cas. En effet :

Pour ligne à N arrêts, il faut remplir une matrice Origine-Destination de taille N^2. Pour chaque coefficient γ_{ij}, il faut tester toutes les valeurs possibles entre 0 et $min(m_i; v_j)$ sauf pour ceux qui sont fixées à zéro qui représentent environ la moitié des cellules. La complexité totale peut donc être approximée par $O(k^{N^2/2})$ où k est le nombre moyen de valeurs possibles par cellules.

Ainsi, avec cet algorithme, on est assez vite limités par les capacités de calcul de la machine. On doit se limiter à quelques arrêts avec peu de voyageurs, ce qui n'est pas très adapté à une utilisation réelle.

4 Sélection d'un plan Origine-Destination

Les deux algorithmes décrits ci-dessus, renvoient l'ensemble des matrices correspondantes, mais il est intéressant de se demander si certains scénarios sont plus plausibles que d'autres et comment les sélectionner. Pour cela, on s'intéresse à la notion d'entropie.

Définition :
On considère une variable aléatoire discrète p qui prend ses valeurs dans un ensemble de cardinal n. La loi de p est décrite par :

$$p = (p_1, p_2, ..., p_n), \quad p_i \geq 0, \quad \sum p_i = 1$$

On définit l'entropie de la variable aléatoire p comme

$$S(p) = \sum p_i log(p_i)$$

Remarque : Il existe également une définition de l'entropie avec un signe $-$.

Pour une ligne de bus donnée, on introduit $K = \sum_{i,j} \gamma_{ij}$ le nombre de voyageurs total. On peut alors voir le plan Origine-Destination γ comme une variable aléatoire dont les issues sont les trajets $i \longrightarrow j$ et dont la loi de probabilité est décrite par :

$$\mathbb{P}(i \longrightarrow j) = \bar{\gamma}_{ij} = \frac{\gamma_{ij}}{K}$$

Proposition : L'entropie d'un plan Origine-Destination γ est donc :

$$S(\gamma) = \sum_{i,j} \bar{\gamma}_{ij} log(\bar{\gamma}_{ij})$$

On peut interpréter l'entropie comme une mesure du désordre ou, dans notre cas, une mesure de la diversité des déplacements des voyageurs. Une entropie minimale correspond à une répartition relativement uniforme entre les trajets possibles, il est difficile de prédire le trajet que fera un voyageur. Au contraire, une grande entropie traduit la présence de trajets plus populaires que d'autres.

Nous faisons l'hypothèse qu'il y a une répartition uniforme des trajets et donc on cherche la matrice OD qui minimise l'entropie.

Proposition : C'est un problème de minimisation d'une fonctionnelle strictement convexe sur un compact convexe, il admet donc une unique solution. (cf. section 4.2.1)

4.1 Calcul de l'entropie sur l'ensemble des solutions

La première approche envisagée a été de calculer l'entropie de toutes les matrices Origine-Destination correspondant à des vecteurs montées et descentes donnés et de finalement renvoyer la matrice minimisant l'entropie. Pour cela, on a implémenté une fonction Python.

Pour la ligne de bus à 5 arrêts (1), c'est la matrice suivante qui minimise l'entropie :

$$\gamma_1 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

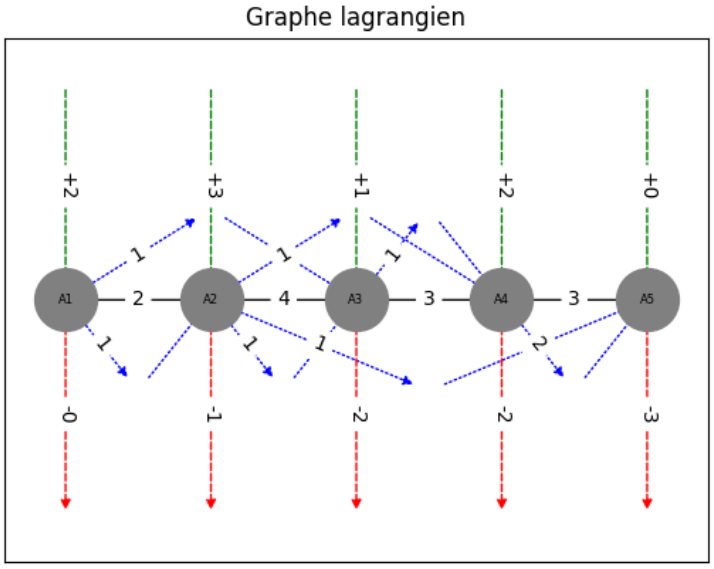

FIGURE 3 – Représentation ligne de bus à 5 arrêts minimisant l'entropie

Les valeurs et la longueur de cette ligne étant très petites, on ne peut pas en tirer grand-chose mais on voit apparaite. La figure (3) représente le graphe de la ligne de bus avec en bleu les trajets des voyageurs.

Pour la ligne à 6 arrêts (2), c'est la matrice suivante qui minimise l'entropie. La figure (4) représente le graphe de la ligne de bus avec en bleu les trajets des voyageurs.

$$\gamma = \begin{pmatrix} 0 & 2 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

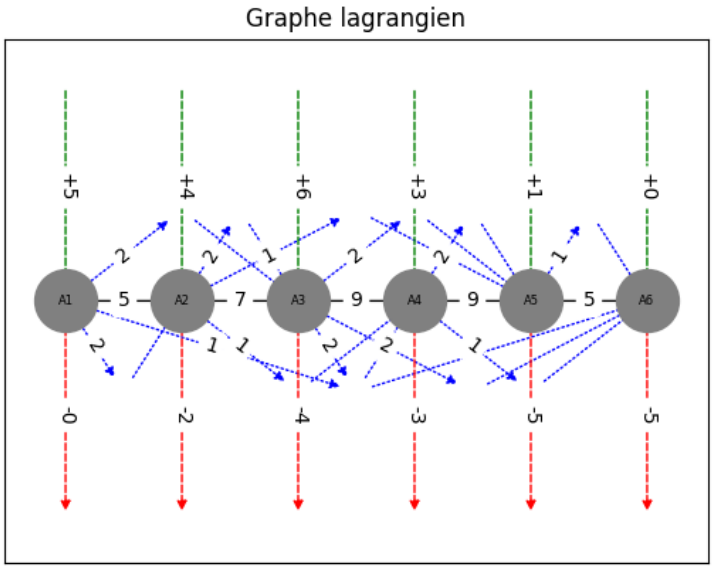

FIGURE 4 – Représentation ligne de bus à 6 arrêts minimisant l'entropie

Dans ce cas, on voit apparaitre clairement la forte répartition des trajets.

L'avantage de cette méthode est qu'elle renvoie une matrice correcte (*ie* qui respecte totalement les contraintes de montées et de descentes) et qu'elle est composée d'entiers. Un important désavantage est qu'elle nécessite d'avoir au préalable calculé l'ensemble des matrices correspondantes, cependant, nous avons vu précédemment que les méthodes pour faire cela était très lentes. Cela peut donc fonctionner pour de petites lignes avec de petites valeurs (moins d'une dizaine d'arrêts et une dizaine de passagers) mais n'est pas pertinente au-delà. Il est donc naturel de vouloir calculer directement la matrice minimisant l'entropie.

4.2 Recherche par des méthodes d'optimisation

Pour cela, nous allons utiliser des méthodes dites d'optimisation et plus précisément de minimisation que nous allons appliquer à l'entropie.

Pour une ligne de bus à N arrêts, il y a $d = \frac{N(N-1)}{2}$ données à trouver (matrice triangulaire supérieure à diagonale vide). La fonction à minimiser est donc :

$$\begin{array}{rcl} S : \mathbb{R}^d & \to & \mathbb{R} \\ x & \mapsto & \sum_{i=1}^{d} x_i log(x_i) \end{array}$$

Le problème de minimisation est donc :

$$\inf_{x \in \mathbb{R}^d} S(x) \quad (P_S)$$

Nous allons tout d'abord poser un cadre théorique de la minimisation puis nous l'appliquerons à notre problématique.

4.2.1 Théorie de l'optimisation

Cette section se base sur le cours d'optimisation de Quentin Marigot [Marigot(2020)].

Définition :
Soit $f \in C^1(\mathbb{R}^d)$,
1. On appelle minimiseur global (ou minimiseur) tout élément $x^* \in \mathbb{R}^d$ vérifiant $f(x^*) = \inf_{R^d} f$. On note $arg\min_{\mathbb{R}^d} f$ l'ensemble des minimiseurs de f (qui peut être vide) *ie* :

$$arg\min_{R^d} f = \{x \in \mathbb{R}^d | f(x) = \inf_{\mathbb{R}^d} f\}$$

2. On appelle suite minimisante pour (P) toute suite $x^{(0)}, ..., x^{(k)}$, ... d'éléments de $\mathbb{R}^d$ telle que $\lim_{k\to+\infty} f(x^{(k)}) = \inf_{x\in\mathbb{R}^d} f(x)$.

Il existe plusieurs méthodes qui permettent alors de résoudre des problèmes de minimisation pour des problèmes sans contraintes : gradient à pas fixe, à pas optimal, gradient préconditionné à rebroussement, méthode de Newton, etc. qui ne seront pas développées ici.

Cependant, dans notre cas, les matrices Origine-Destination recevables sont contraintes par les données de montées et de descentes. Nous allons donc nous intéresser aux méthodes d'optimisation sous contraintes.

Pour ce type de problème, on définit les contraintes par des inégalités : des fonctions convexes $c_1, \ldots, c_l \in C^1(\mathbb{R}^d)$. On définit alors l'ensemble $K \subseteq \mathbb{R}^d$ convexe fermé :

$$K = \{x \in \mathbb{R}^d \mid c_1(x) \leq 0, \ldots, c_l(x) \leq 0\}$$

On appelle chacune des fonctions $c_1, \ldots, c_l$ une contrainte du problème et on note alors le problème d'optimisation pour une fonction f convexe comme :

$$P := \min_{c_1(x),\ldots,c_l(x)} f(x)$$

Proposition : (Existence d'une solution)
Un problème d'optimisation sous contraintes tel que définit précédemment admet une solution si l'une des conditions suivantes est vérifiée :
1. K est compact, non vide et f est continue.
2. K est fermé, non vide, f est continue et $\lim_{\|x\|\to+\infty} f(x) = +\infty$

Proposition : (Unicité)
Si la fonction f est strictement convexe et si l'ensemble K est convexe, alors le problème d'optimisation admet au plus une solution.

Proposition : En notant $(e_i)_{1\leq i,j\leq d}$ est la base canonique de $\mathbb{R}^d$, si $\forall x \in \mathbb{R}^d, \quad D^2 f(x) = (\frac{\partial^2 f}{\partial e_i \partial e_j}(x))_{1\leq i,j\leq d}$ (héssienne de f) est définie positive, alors f est strictement convexe.

Pour faire la démonstration de cette proposition, on va utiliser les quatre lemmes suivants :

Lemme : (1) Soit $f \in C^1(\mathbb{R}^d)$. Etant donnés $x \in \Omega$ et $v \in \mathbb{R}^d$, on définit $x_t = x + tv$ et $g(t) = f(x_t)$. Alors,

$$g'(t) =< \nabla f(x_t)|v >$$

Lemme : (2) Soit $\Omega \subset \mathbb{R}^d$ ouvert, $f \in C^2(\Omega)$, $x \in \Omega$, $v \in \mathbb{R}^d$ et $g\ t \mapsto f(x_t)$ où $x_t = x + tv$. Alors,

$$g''(t) =< D^2 f(x_t)v|v >$$

Démonstration : (2) On fait le calcul en coordonnées, en notant $(e_k)_k$ la base canonique :

$$g(t) = f(x + \sum_x tv_k e_k)$$

$$g'(t) = \sum_i v_i \frac{\partial f}{\partial e_i}(x + \sum_k tv_k e_k) =< \nabla f(x_t)|v >$$

$$g''(t) = \sum_i \sum_j v_i v_i \frac{\partial^2 f}{\partial e_j \partial e_i}(x + \sum_k tv_k e_k) =< D^2 f(x_t)v|v > \quad \square$$

Lemme : (Taylor-Lagrange). Soir $f \in C^2(\mathbb{R}^d)$, $x, v \in \mathbb{R}^d$ et $x_t = x + tv$. Alors,

$$\forall t \geq 0, \exists s \in [0, t], \quad f(x_t) = f(x) + t < \nabla f(x)|v > + \frac{t^2}{2} < D^2 f(x_s)v|v > .$$

Lemme : (3) Les propositions suivantes sont équivalentes :
1. f est (strictement) convexe sur $\mathbb{R}^d$
2.

$$\forall x, y \in \mathbb{R}^d, \text{ la fonction } g : t \in [0, 1] \mapsto f((1 - t)x + ty) \text{ est (strictement) convexe.}$$

3.

$$\forall x, y \in \mathbb{R}^d,\ f(y) \geq f(x) + < y - x|\nabla f(x) > \text{ (stricetement convexe si inégalité stricte).}$$

4.

$$\forall x, y \in \mathbb{R}^d,\ < \nabla f(x) - \nabla f(y)|x - y > \geq 0 \text{ (stricetement convexe si inégalité stricte).}$$

Démonstration : (3)
1. $\Longleftrightarrow$ 2. par application directe de la définition.
2. $\Longrightarrow$ 3. Soit $x, y \in \mathbb{R}^d$ et $g : t \mapsto f(x_t)$ où $x_t = (1 - t)x + tv = x + t(y - x)$ qui est convexe par hypothèse. Par convexité, on a $g(t) \leq g(0) + tg'(0)$ soit par le lemme 1 : $f(x) + < \nabla f(x)|y - x > \leq f(y)$
3. $\Longrightarrow$ 4. Il suffit de sommer l'inégalité (3) et la même inégalité où l'on inverse les rôles de x et de y.
4. $\Longrightarrow$ 2. Soit encore $g : t \mapsto f(x_t)$. Comme $g'(t) =< \nabla f(x_t)|y - x >$ par le lemme (1), l'inégalité 4. appliquée en x_s et x_t (où t>s) nous donne

$$g'(t) - g'(s) =< \nabla f(x_t) - \nabla f(x_s)|y - x >= \frac{1}{t - s} < \nabla f(x_t) - \nabla f(x_s)|x_t - x_s > \geq 0.$$

et g' est donc croissante sur [0,1]. Ainsi, g est convexe. $\square$

Démonstration : (proposition) On considère $x, y \in \Omega$ et $g(t) = f(x_t)$ où $x_t = (1-t)x+ty$. Alors, par le lemme (2) avec $v = y - x$, on a $g''(t) =< D^2f(x_t)(y_x)|(y-x) >$ est strictement positif par hypothèse, donc par le lemme (Taylor-Lagrange) on a :

$$g(1) = g(0) + g'(0 = +\frac{s^2}{2}g''(s) > g(0) + g'(0).$$

Or, $f(y) = g(1)$ et $f(x) = g(0)$. Donc, on a, $f(y) > f(x)+ < \nabla f(x)|y - x >$ et donc d'après le lemme (3) donc f est strictement convexe. $\square$

Lemme : Si les fonctions $c_1, \ldots, c_l\, \mathbb{R}^d \to \mathbb{R}$ sont continues et convexes, alors l'ensemble $K = \{x \in \mathbb{R}^d \mid c_1(x) \leq 0, \ldots, c_l(x) \leq 0\}$ est fermé et convexe.

Démonstration :
K est fermé :
Soit (x_n) une suite d'élément de K convergent vers une limite x, montrons que $x \in K$. Par hypothèse, comme $x_n \in K$, $c_i(x_n) \leq 0$, $\forall i \in \{1, \ldots, l\}$. En passant à la limite $n \to +\infty$ en utilisant la continuité de c_i, on en déduit que $c_i(x) \leq 0, \forall i \in \{1, \ldots, l\}$. Donc x appartient à K et donc K est fermé.

K est convexe :
Soient $x, y \in K$ et $x_t = (1-t)x + ty$. Comme $x, y \in K$, on a pour tout $i \in \{1, \ldots, l\}$, $c_i(x) \leq 0$ et $c_i(y) \leq 0$. Pour tout $t \in [0, 1]$, on a donc :

$$c_i(x_t) \leq (1-t)c_i(x) + tc_i(y) \leq 0$$

Donc x_t appartient à K et donc K est convexe. $\square$

Méthode pour trouver le minimiseur :
Pour résoudre les problèmes d'optimisation sous contraintes $min_K f$ où $K = \{x \quad | \quad \forall i, c_i(x) \leq 0\}$, une idée peut être de se ramener à un problème d'optimisation sans contraintes :

$$\min_{\mathbb{R}^d} f_\epsilon$$

où est f_ϵ la somme de f et de termes qui vont exploser lorsque les conditions ne seront pas vérifiées. Précisément, pour $\epsilon > 0$, on pose :

$$P_\epsilon = \min_{x \in \mathbb{R}^d} f_\epsilon(x) \text{ où } f_\epsilon(x) = f(x) + \frac{1}{\epsilon}\sum_{i=1}^{l} max(c_i(x), 0)^2.$$

Dans cette formulation du problème, les points $x \in \mathbb{R}^d$ tels que $c_i(x) > 0$ sont pénalisés au sens où si ϵ est très petit alors $\frac{1}{\epsilon}max(c_i(x), 0)^2$ peut devenir très grand, ce qui va dissuader le choix de ce point dans le problème d'optimisation sur $\mathbb{R}^d$:

$$\forall x \in \mathbb{R}^d, \lim_{\epsilon \to 0} \frac{1}{\epsilon}\sum_{i=1}^{l} max(c_i(x), 0)^2 = \begin{cases} 0 & \text{si } c_i(x) \leq 0 \\ +\infty & \text{sinon.} \end{cases}$$

Proposition : Soit $f \in C^0(\mathbb{R}^d)$ strictement convexe et vérifiant $\lim_{\|x\|\to+\infty} f(x) = +\infty$, alors :
1. Les problèmes P_ϵ et P admettent chacun un unique minimiseur, noté x^* et x^*_ϵ.
2. $\lim_{\epsilon\to 0} x^*_\epsilon = x^*$

Démonstration :
1. L'existence d'une solution au problème d'optimisation sans contraintes P_ϵ se déduit du fait que f_ϵ est continue et que $\lim_{\|x\|\to+\infty} f_\epsilon(x) \geq \lim_{\|x\|\to+\infty} f(x) = +\infty$. Tandis que l'existence de solution au problème avec contraintes P se déduit du fait que $\lim_{\|x\|\to+\infty} f(x) = +\infty$ et que K est fermé (lemme précédent). Pour l'unicité de la solution au problème P, il suffit de remarquer que f est strictement convexe (par hypothèse) et que K est convexe (lemme précédent à nouveau). Enfin, pour l'unicité de la solution du problème P_ϵ il faut montrer que f_ϵ est strictement convexe. Soient $x, y \in \mathbb{R}^d, t \in [0,1]$ et $x_t = (1-t)x + ty$. Alors,

$$c_i(x_t) \leq (1-t)c_i(x) + tc_i(y)$$

de sorte que :

$$max(c_i(x_t), 0) \leq max((1-t)c_i(x) + tc_i(y), 0) \leq (1-t)max(c_i(x), 0) + tmax(c_i(y), 0)$$

puis en utilisant la convexité de $r \in \mathbb{R} \mapsto r^2$, on a :

$$max(c_i(x_t), 0)^2 \leq [(1-t)max(c_i(x), 0) + tmax(c_i(y), 0)]^2 \leq (1-t)max(c_i(x), 0)^2 + tmax(c_i(y), 0)^2$$

On en déduit que les fonctions $x \mapsto max(c_i(x), 0)^2$ sont convexes et f_ϵ est donc strictement convexe comme combinaison linéaire à coefficients positifs de fonctions convexes et d'une fonction strictement convexe.

2. Soit $x^* \in K$ le minimiseur de P. Alors comme $c_i(x^*) \leq 0$, on a :

$$P_\epsilon \leq f_\epsilon(x^*) = f(x^*) + \frac{1}{\epsilon}\sum_{i=1}^{l} max(c_i(x^*), 0)^2 = f(x^*) = P$$

Ainsi,

$$P_\epsilon = f(x^*_\epsilon) + \frac{1}{\epsilon}\sum_{i=1}^{l} max(c_i(x^*_\epsilon), 0)^2 \leq P$$

Comme $\lim_{\|x\|\to+\infty} f(x) = +\infty$, la fonction f est minorée par $m \in \mathbb{R}$. On en déduit donc que :

$$\forall i \{1, ..., l\}, \frac{1}{\epsilon} max(c_i(x^*_\epsilon), 0) \leq \frac{1}{\epsilon}\sum_{j=1}^{l} max(c_j(x^*_\epsilon), 0)^2 \leq P - f(x^*_\epsilon) \leq P - m$$

ou encore

$$max(c_i(x^*_\epsilon), 0)^2 \leq \epsilon(P - m).$$

Soit $\bar{x}$ une valeur d'adhérence de la suite x^*_ϵ lorsque $\epsilon \to 0$. Alors, par continuité,

$$max(c_i(\bar{x}), 0)^2 \leq 0$$

ce qui montre que $\bar{x} \in K$. De plus, pour tout $\epsilon > 0$, $f(x_\epsilon) \leq P_\epsilon \leq P$ de sorte que $f(\bar{x}) \leq P$. On en déduit que $\bar{x}$ minimise f sur K, soit $\bar{x} = x^*$. Pour conclure, on remarque que la suite (x^*_ϵ) est bornée et admet une valeur d'adhérence de sorte que $\lim_{\epsilon\to 0} x^*_\epsilon = x^*$. $\square$

4.2.2 Application à l'entropie

Vérifions maintenant que le problème de minimisation de l'entropie avec les contraintes de montées et de descentes se trouvent dans le cadre théorique décrit précédemment.

Tout d'abord, vérifions que $S : x \in \mathbb{R}^d \mapsto \sum_{i=1}^{d} x_i log(x_i)$, est C^2, strictement convexe et vérifie $\lim_{\|x\| \to +\infty} f(x) = +\infty$.

1. Continuité :
La fonction log est C^∞ sur $\mathbb{R}_*^+$.
2. Stricte convexité :
Pour cela, nous allons utiliser la première proposition de la section 4.2.1 en étudiant la hessienne de S, $\quad \forall i, j \in \{1, ..., d\}$:

$$i \neq j : \ \frac{\partial^2 S}{\partial_i \partial_j}(x) = 0$$

$$i = j : \ \frac{\partial^2 S}{\partial_i^2}(x) = \frac{1}{x_i}$$

Donc la hessienne de $D^2S(x)$ pour $x \in \mathbb{R}^d$ est une matrice diagonale, donc le spectre est $\{\frac{1}{x_1}, ..., \frac{1}{x_d}\}$ tous non-nuls donc $D^2S(x)$ est définie positive et donc S est strictement convexe.
3. Limite :

$$\forall x_i \in \mathbb{R}, \lim_{x_i \to +\infty} x_i log(x_i) = +\infty$$

donc on a $\forall x \in \mathbb{R}^d \lim_{\|x\| \to +\infty} f(x) = +\infty$

Transformons les contraintes de montées et de descente en fonctions convexes et continues $c_i : x \in \mathbb{R}^d \mapsto \mathbb{R}$.

— On a tout d'abord la contrainte de la définition du logarithme (qui correspond également à la réalité) : les x_i sont des réels positifs. Donc $\forall i \in \{1, ..., d\}$, $c_i(x) = -x_i \leq 0$.

— On a les contraintes sur les données des montées $\mu \in \mathbb{R}^N$ qui sont égales aux sommes sur les lignes. Comme ce sont des égalités, on pose deux contraintes opposées à chaque fois. Pour faciliter l'écriture, on se replace dans la vue matricielle :
$c_{d+i} = \sum_{j=1}^{N} \gamma_{ij} - \mu_i \leq 0$
$c_{d+N+i} = -(\sum_{j=1}^{N} \gamma_{ij} - \mu_i) \leq 0$

— On a les contraintes sur les données de descentes. À nouveau, ce sont des égalités, donc on pose deux inégalités pour chaque colonne.
$c_{d+2N+j} = \sum_{i=1}^{N} \gamma_{ij} - \nu_j \leq 0$
$c_{d+3N+j} = -(\sum_{i=1}^{N} \gamma_{ij} - \nu_j) \leq 0$

Toutes les fonctions de contraintes sont des fonctions polynomiales du premier degré donc sont convexes et continues alors $K = \{x \in \mathbb{R}^d \mid c_1(x) \leq 0, \ldots, c_l(x) \leq 0\}$ est un convexe fermé.

Notre problème vérifie donc bien toutes les conditions d'un problème de minimisation sous contraintes : il existe bien une unique solution que l'on va pouvoir approcher à l'aide de la méthode de pénalisation.

$$P_{S\epsilon} = \min_{\mathbb{R}^d} \sum_{i=1}^{d} x_i log(x_i) + \frac{1}{\epsilon} \sum_{j=1}^{l} max(c_i(x), 0)^2$$

4.2.3 Méthodes d'optimisation en python

La suite naturelle est d'implémenter ces méthodes en Python et de les tester sur différentes données.

Une des difficultés du code a été les passages de la vue vectorielle ($\mathbb{R}^d$) qui correspond au vecteur des fonctions d'optimisation à la vue matricielle ($\mathbb{R}^{N \times N}$) qui correspond à nos données et notamment aux contraintes. Voici un exemple de passage de l'un à l'audtre pour une matrice Origine-Destination de taille $N = 5$ et $d = 10$:

$$\gamma = \begin{pmatrix} \gamma_{00} & \gamma_{01} & \gamma_{02} & \gamma_{03} & \gamma_{04} \\ \gamma_{10} & \gamma_{11} & \gamma_{12} & \gamma_{13} & \gamma_{14} \\ \gamma_{20} & \gamma_{21} & \gamma_{22} & \gamma_{23} & \gamma_{24} \\ \gamma_{30} & \gamma_{31} & \gamma_{32} & \gamma_{33} & \gamma_{34} \\ \gamma_{40} & \gamma_{41} & \gamma_{42} & \gamma_{43} & \gamma_{44} \end{pmatrix} = \begin{pmatrix} 0 & x_1 & x_2 & x_3 & x_4 \\ 0 & 0 & x_5 & x_6 & x_7 \\ 0 & 0 & 0 & x_8 & x_9 \\ 0 & 0 & 0 & 0 & x_{10} \end{pmatrix} \rightarrow x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix}$$

Première méthode : La première méthode d'optimisation explorée est une de celles proposées par la bibliothèque Scipy, il s'agit de la fonction MINIMIZE avec la méthode TRUST-CONSTR (Trust-Region Constrained Algorithm). Pour cette méthode, il est nécessaire de définir des bornes pour les valeurs $[0, +\infty[$ ainsi qu'une matrice de contraintes. C'est une matrice A telle que :

$$Ax = \begin{pmatrix} \mu_0 \\ \vdots \\ \mu_N \\ \nu_0 \\ \vdots \\ \nu_N \end{pmatrix} \quad \text{Pour le cas } N = 4, A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Il est tout à fait possible de généraliser la matrice A, mais son écriture, notamment pour les descentes, n'est pas très pratique. Il est plus simple de la coder en repassant par la

vue matricielle. On passe en argument la fonction ainsi que son gradient et sa hessienne pour éviter que leur calcul par Scipy amènent à de trop grandes approximations.

$$S(x) = \sum_{i=1}^{n} x_i \times \log(x_i)$$

$$\nabla S(x)_i = \log(x_i) + 1$$

$$D^2(x)_{i,j} = \begin{cases} 0 & \text{si } i \neq j \\ \frac{1}{x_i} & \text{sinon.} \end{cases}$$

Cependant, la fonction log et la fonction inverse n'étant pas définies pour des valeurs nulles ou négatives, les fonctions ont été légèrement modifiées pour éviter les erreurs lors de l'exécution du code :

```
def entropie(x):
    res = 0
    # Entropie
    for x_i in x:
        if x_i > 0:
            res += x_i * np.log(x_i)
    return res

# Definition de la jacobienne associee
def jacobian_entropie(x):
    res = []
    for x_i in x:
        if x_i > 0:
            res += [np.log(x_i) + 1]
        else:
            res += [0]
    return res

# Definition de la hessienne associee
def hessian_entropie(x):
    res = []
    for i in range(len(x)):
        ligne_res = []
        for j in range(len(x)):
            if i != j:
                ligne_res.append(0)
            else:
                ligne_res.append(1 / x[i])
        res.append(ligne_res)
    return res
```

Seconde méthode : La seconde méthode explorée a été une méthode de pénalisation comme décrite dans la section précédente. Pour cela, on se base également sur la bibliothèque Scipy et la fonction MINIMIZE mais cette fois, on utilise la méthode NELDER-MEAD qui n'utilise pas de borne et de contraintes linéaires. La fonction passée en

argument est donc :

$$S_\epsilon(x) = \sum_{i=1}^{N}(x_i log(x_i) + \frac{1}{\epsilon}\max(-i,0)^2) + \sum_{i=1}^{N}\frac{1}{\epsilon}(\sum_{j \text{ sur la ligne i}} x_j - \mu_i)^2 + \sum_{j=1}^{N}\frac{1}{\epsilon}(\sum_{i \text{ sur la colonne j}} x_i - \nu_j)^2$$

Comme pour la première méthode, on a également calculé le gradient pour éviter les erreurs d'approximation. Pour cela, on va également devoir utiliser la vue matricielle. On suppose que x_i est sur la ligne k et la colonne l :

$$\nabla S_\epsilon(x)_i = \log(x_i) + 1 + \frac{2}{\epsilon}(\text{ somme des termes sur la même ligne que } x_i \text{ en vue matricielle} - \mu_k) + \frac{2}{\epsilon}(\text{ somme des termes sur la même colonne que } x_i \text{ en vue matricielle} - \mu_l)$$

Il a fallu ensuite définir une valeur pour le terme de pénalisation $\frac{1}{\epsilon}$. En effet, si on le choisit trop petit, les contraintes ne seront pas forcément respectées, mais si on le choisit trop grand, il est trop restrictif et on ne peut pas aboutir au meilleur résultat. Pour cela, on a effectué des tests sur des vecteurs de différentes longueurs, il est ressorti que prendre $\epsilon = 0.01$ ou $\frac{1}{\epsilon} = 100$ était le choix qui amenait à la meilleure qualité, notion définie (4.2.4) un peu plus tard dans la partie test des méthodes.

Enfin, pour les deux méthodes, il est nécessaire de choisir un vecteur initial $x_0 \in \mathbb{R}^d$. Pour cela, on a fait le choix du vecteur correspondant au produit des marginales (en vue matricielle) : $\gamma_{ij} = \mu_i * \nu_j$. Ce choix permet de se placer dès le début dans l'espace des vecteurs respectant les contraintes.

4.2.4 Tests des méthodes d'optimisation Python

Test 1 :
Génération de vecteurs aléatoires. Explication de la fonction qualité (respect des contraintes). Graphes comparatifs en temps et en qualités des vecteurs aléatoires et analyse des résultats.

Le respect des contraintes, c'est bien, mais est-ce que c'est bien le bon résultat de la vraie vie ? Deux moyens de comparer les matrices trouvées et les véritables matrices : entropie relative et moindres carrés.

Définition mathématique de l'entropie relative, démonstrations des quelques proposition dessus. Explique que nous, on va regarder $S_trouve(vrai)$ pour des raisons de zéros.

Définition des moindres carrés, démonstration des mêmes propositions pour les moindres

carrés.

Test 2 :
Test sur des petits vecteurs et de petites valeurs (5 et 6 arrêts). Comparaison aux résultats précédents ? Quelques images de graphes pour illustrer les résultats trouvés (dénormalisation) ?

Test 3 :
Nous avons également voulu comparer les deux algorithmes sur de vraies données. Nous avons alors pris contact avec Rennes Métropole qui nous ont donnés accès à de véritables données de déplacements issues de l'enquête réalisée au printemps 2023 où des matrices Origine-Destination avait été construites pour chaque ligne. Nous nous sommes intéressés aux données de différentes lignes avec des tailles différentes et des types de trajets différents : les deux lignes de métro pendant une journée de semaine (15 arrêts chacune), la ligne C4 (35 arrêts) et la ligne C7 (19 arrêts). À chaque fois, nous avons travaillé sur la plage horaire de 8h45-8h59.

Ligne	Temps de traitement (secondes)	Entropie relative	Moindres carrés
A JOB ϵ	26.7	0.56977	0.043473
A JOB scipy	20.3	0.6589768	0.04119
B JOB ϵ	14.6	0.63520	0.0506
B JOB Scipy	69.239	0.7512	0.0501
C4 JOB ϵ	739.4 (12min et 19.4sec)	Nan	0.0839
C4 JOB Scipy	1295.4 (21min et 35.4sec)	2.21104	0.1029
C7 JOB ϵ	64.0	0.766	0.0456
C7 JOB Scipy	73.8	0.5758	0.0423

Faire conclusion pour les tests.

5 Temps de transport

5.1 Cadre théorique

On se place maintenant dans un cadre plus large que celui de la ligne de bus : un réseau de transport d'une agglomération avec des routes, des lignes de métro, des lignes de bus, etc. On reprend la modélisation par un graphe, on pose $\mathcal{N} = (V, E)$ où :

— V est un ensemble de points qui correspondent à des zones géographiques de taille réduite : une zone de travail, d'habitation, d'étude ou un point de bifurcation (échangeur routier, gare de correspondance, etc).

— E est un ensemble d'arêtes qui correspondent à des tronçons routiers ou de voies ferrées qui relient deux points de V. On attribue un type à chaque arête : C si c'est un transport en commun et R si c'est un transport routier.

On note :

— $X \subset V$ les zones de départ (zone d'habitation).

— $Y \subset V$ les zones d'arrivée (zones d'étude ou de travail).

Remarque : On n'a pas nécessairement $X \cap Y = \emptyset$.

On se donne une mesure $\mu = (\mu_x) = \mathbb{R}_+^X$ sur X quantifiant le nombre de personnes rattachées aux différentes zones de X. Et pareillement une mesure $\nu = (\nu_y) = \mathbb{R}_+^Y$ quantifiant le nombre de personnes rattachées aux différentes zones de Y.

On peut à nouveau introduire la notion de plan Origine-Destination $\gamma = (\gamma_{xy}) \in \mathbb{R}_+^{X \times Y}$ où γ_{xy} est le nombre de personnes voulant se rendre de x à y. On notera $\Lambda_{\mu\nu}$ les plans qui satisfont les conditions :

$$\sum_{y \in Y} \gamma_{xy} = \mu_x \text{ et } \sum_{x \in X} \gamma_{xy} = \nu_y$$

On introduit également le support de (γ_{xy}) : $\Lambda = \{(x, y) \in X \times Y,\ \gamma_{xy} > 0\}$, l'ensemble des couples départ-arrivée qui sont concernés par une quantité non-nulle de voyageurs.

On introduit ensuite la notion de chemin. Pour $x \in X$ et $y \in Y$, on définit un chemin entre x et y comme une suite consécutive d'arêtes dont la première est issue de x et la dernière termine en y. On dira qu'un chemin c est admissible s'il ne contient des arêtes que de type R ou que de type C ou d'un premier tronçon d'arêtes de type R puis un deuxième tronçon d'arêtes de type C. On note C_{xy} l'ensemble des chemins admissibles de x à y.

On suppose que Y est fortement connecté à X :

$$\forall (x, y) \in \Lambda, \quad C_{xy} \neq \emptyset$$

On introduit ensuite la notion de scénario qui correspond à un choix de répartition (ou de distribution) des voyageurs entre les différents chemins admissibles que l'on note, pour tout $x, y \in \Lambda$, $(\gamma_{xy}^c)_{c \in C_{xy}} \in \mathbb{R}_+^{C_{xy}}$ où γ_{xy}^c est le nombre de personnes allant de x à y en passant par le chemin c. On a :

$$\sum_{c \in C_{xy}} \gamma_{xy}^c = \gamma_{xy}$$

5.2 Problématique

À nouveau, on pourrait se poser la question du passage des données eulériennes à lagrangiennes, le problème est relativement similaire mais la présence de plusieurs chemins pour le même trajet de x à y fait augmenter le nombre de scénarios lagrangiens amenant aux mêmes données eulériennes. À nouveau, on pourrait utiliser la maximisation ou la minimisation de l'entropie pour sélectionner un scénario.

Dans cette section, nous allons plutôt nous intéresser aux temps de trajet et aux stratégies qui leur sont associées. La difficulté d'analyse des stratégies tient au fait que le temps de transport associé à un voyageur et son itinéraire peut dépendre du choix des autres voyageurs par des phénomènes de congestions (ex : bouchons sur la rocade rennaise aux heures de pointe ou à la station de métro Gares des lignes a et b).

On note T^c_{xy} le temps de parcours associé au chemin c entre x et y. Ce temps dépend tout d'abord du chemin choisi, mais aussi *a priori* aussi du choix des autres voyageurs. En toute généralité, $T^c_{xy} = T^c_{xy}(\gamma)$ où $\gamma = (\gamma^c_{xy})$ Il s'agit donc d'un temps de parcours *a posteriori* puisqu'il dépend, pour un jour donné, de l'ensemble des choix effectués par les voyageurs qui n'est pas connu *a priori*.

On fait l'hypothèse qu'un voyageur seul ne pèse rien : son temps de parcours dépend du choix des autres voyageurs, mais sa propre décision personnelle n'affecte pas à elle seule le temps de parcours d'un chemin qui le choisisse ou pas.

Remarque : Le temps $T^c_{xy}(\gamma)$ a un sens aussi pour un chemin qui n'est emprunté par personne, i.e. tel que $\gamma^c_{xy} = 0$. Le chemin $c \in C_{xy}$, même s'il est mis de côté par les gens qui vont de x à y reste une option possible et qui dans certains cas sera même plus avantageuse que d'autres chemins empruntés.

Il existe plusieurs quantités observables associées à un plan (γ^c_{xy}) :

Si l'on suppose connue la fonction $T^c()$ d'estimation des temps de parcours, on peut associer à un plan $\gamma = (\gamma^c_{xy})$ le temps moyen de parcours défini par :

$$\bar{T} = \bar{T}(\gamma) = \frac{1}{\mu_X} \sum_{(x,y)\in X\times Y} \sum_{c\in C_{xy}} \gamma^c_{xy} T^c_{xy}(\gamma) \quad \text{où } \mu_X = \sum_{x\in X} \mu_x$$

Remarque : On prendra garde au fait que ce temps moyen n'est pas une fonction linéaire de γ, du fait de la dépendance des temps de parcours vis-à-vis de γ. On notera aussi le caractère non local des termes intervenant dans la sommation : le temps de parcours associé à un chemin c de x vers y est susceptible d'être influencé par le nombre de voyageurs se rendant de $x' \neq x$ à $y' \neq y$, si le chemin c' en question partage un tronçon avec c.

On peut également observer le temps de parcours "à vide" T^0_{xy}, le temps minimal que l'on peut mettre pour aller de x à y (sans congestion). On pourra alors s'intéresser à la différence $T^c_{xy} - T^0_{xy}$ pour un chemin c qui mesurera la "frustration" des voyageurs allant de x à y par le chemin c en un temps T^c_{xy} qui sont conscients qu'ils auraient pu aller plus vite s'ils étaient seuls (absence de congestion).

On peut associer à un plan γ d'autres quantités observables comme la quantité de gaz à effet de serre émise par l'ensemble des voyageurs que l'on pourrait chercher à minimiser.

Une question qui peut se poser en tant que voyageurs est alors "aurais-je pu aller plus vite si j'avais choisi un autre chemin ?". Pour cela, on introduit la notion d'équilibre de Nash :

Définition : Équilibre de Nash pour le temps de parcours d'un plan de mobilité.
Avec les notations précédentes, on dit qu'un plan $\gamma = (\gamma_{xy}^c)$ réalise un équilibre de Nash si :

$$T_{xy}^c \geq T_{xy}^{max} = \max_{c \in C^{xy}, \gamma_{xy}^c > 0} \quad \forall c \in C_{xy}, \text{ avec égalité si } \gamma_{xy}^c > 0$$

Cela signifie que le temps de parcours associé aux chemins $c \in C^{xy}$ est le même pour tous les chemins empruntés ($\gamma_{xy}^c > 0$) et est plus élevé pour les chemins non empruntés. Un voyageur allant de x à y ayant opté pour un certain chemin n'a pas d'intérêt à changer de chemin pour diminuer son temps de parcours par rapport à ce que font les autres voyageurs.

Le temps de parcours effectif d'un voyageur dépend de nombreux facteurs et se trouve en particulier conditionné par des phénomènes ponctuels et locaux qu'il est difficile de prévoir. Pour les parcours sur des arêtes de type R (routes), le temps de parcours d'un conducteur dépend en particulier de l'apparition de bouchons qui peuvent apparaitre puis se résorber, de sorte que pour un même parcours, un décalage du départ peut affecter de manière significative le temps de parcours. On peut néanmoins considérer que le temps effectif moyen est une fonction décroissante de la quantité de personnes présentes. Dans cette optique, on pourra considérer par exemple que le temps mis pour parcourir une arête e s'écrit comme un temps de référence T_0^e (longueur de e divisé par la vitesse limite sur ce tronçon) multiplié par un facteur correctif supérieur ou égale à 1 qui dépend du nombre de personnes qui empruntent cette arête durant leur parcours.

$$T^e = T_0^e \psi(\sum_{x,y} \sum_{c \in C_{xy} | e \in c} \gamma_{xy}^c)$$

Pour définir ψ, on peut s'inspirer des modèles classiques de trafic routier :

$$\psi(m) = \frac{1}{(1 - \frac{m}{m_{max}^e})} \in [1; +\infty[$$

où m_{max}^e est la quantité de véhicules qui conduit, sur la période de temps considérée, à une saturation complète du trafic.

Plusieurs questions se posent alors :

— Existe-t-il toujours un équilibre de Nash pour le cas du temps de parcours d'un plan de mobilité ?

— Si oui, est-il unique ? optimal ?

— Comment trouver cet équilibre ?

6 Autres méthodes

Thèse de Remy Coulon [Coulaud(2022)]

7 Conclusion

8 Références

Références

[Coulaud(2022)] R. Coulaud. *Modélisation et prévision des variables d'exploitation ferroviaire et de flux de voyageurs en zone dense*. PhD thesis, Université Paris-Saclay, 2022.

[Marigot(2020)] Q. Marigot. Algorithmes d'optimisation, 2020.

[Maury(2024)] B. Maury. Cours de modélisation mathématique, 2024.

9 Annexe