Rapport de Projet de Synthèse

Simulation et Analyse du Modèle SEIRS

Étude comparative multi-langages et analyse de performance

Boueke Omer

Master 2 CHPS

Année universitaire 2025-2026
27 janvier 2026

Sommaire

Ce rapport présente une étude approfondie du modèle épidémiologique SEIRS (Sain, Exposé, Infecté, Rétabli, Sain) à travers une approche multi-langages et multi-méthodes. Le projet est structuré en trois parties principales, chacune apportant un éclairage différent sur la modélisation épidémiologique et le calcul haute performance.

Table des matières

Table des figures

Introduction du Projet : Simulation et Analyse du Modèle SEIRS

Ce projet porte sur l'étude et l'implémentation du modèle épidémiologique SEIRS (Sain, Exposé, Infecté, Rétabli, Sain). L'objectif est de comprendre la dynamique de propagation d'une maladie au sein d'une population à travers deux approches complémentaires.

Le projet est structuré en deux grandes parties, complétées par une analyse comparative approfondie.

Chapitre 1

Analyse comparative des méthodes numériques

1.1 Introduction

Pour cette première partie, j'ai choisi d'implémenter le modèle épidémiologique SEIRS en utilisant deux langages : Python et C. L'objectif est de comparer deux méthodes de résolution numérique : Euler (ordre 1) et Runge-Kutta 4 (RK4, ordre 4).

Comme la bibliothèque scipy en Python ne propose pas directement la méthode d'Euler, j'ai décidé de l'implémenter manuellement en Python et en C. Pour RK4, j'ai utilisé les bibliothèques standards pour plus de précision : `scipy.integrate.solve_ivp` pour Python et la GSL (GNU Scientific Library) pour le langage C.

1.2 Analyse comparative entre les langages (C vs Python)

L'idée ici était de voir si, pour une même méthode, le langage de programmation changeait le résultat.

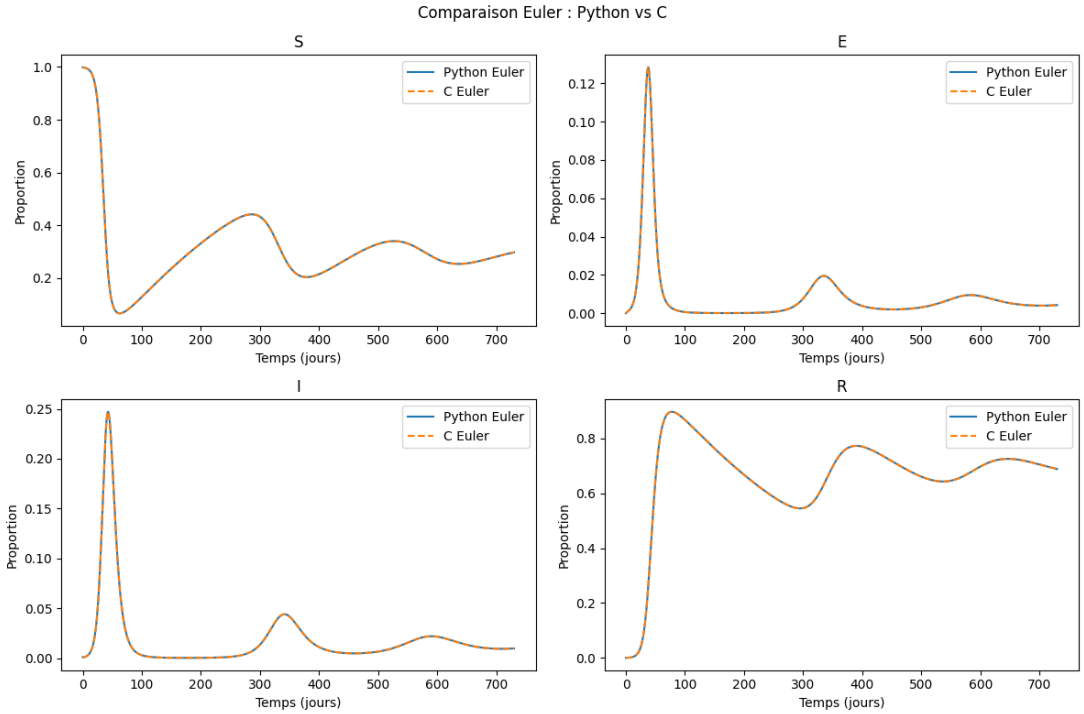

FIGURE 1.1 – Comparaison des résultats de la méthode d'Euler entre Python et C pour les quatre compartiments S(t), E(t), I(t) et R(t)

— **Méthode d'Euler :** En observant les graphiques des quatre fonctions S(t), E(t), I(t) et R(t), on remarque que les courbes sont parfaitement superposées. Les résultats produits par Python et C sont quasi identiques.

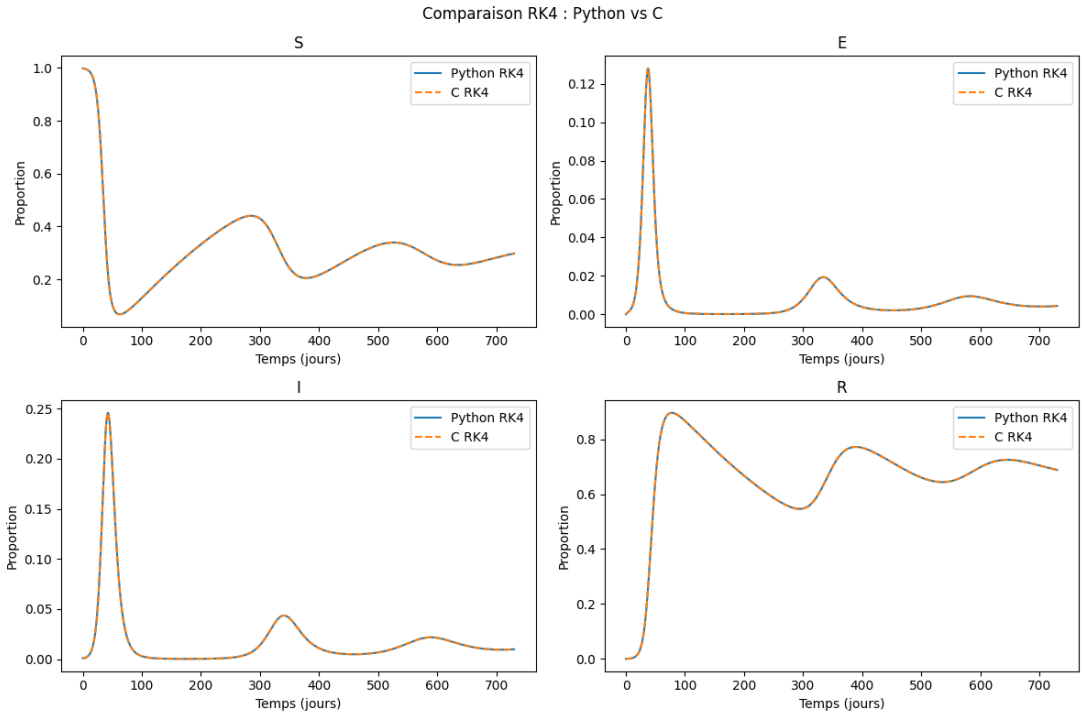

FIGURE 1.2 – Comparaison des résultats de la méthode RK4 entre Python et C pour les quatre compartiments

— **Méthode RK4 :** On fait le même constat pour RK4 ; les bibliothèques des deux langages convergent vers la même solution.

1.3 Analyse comparative des méthodes (Euler vs RK4)

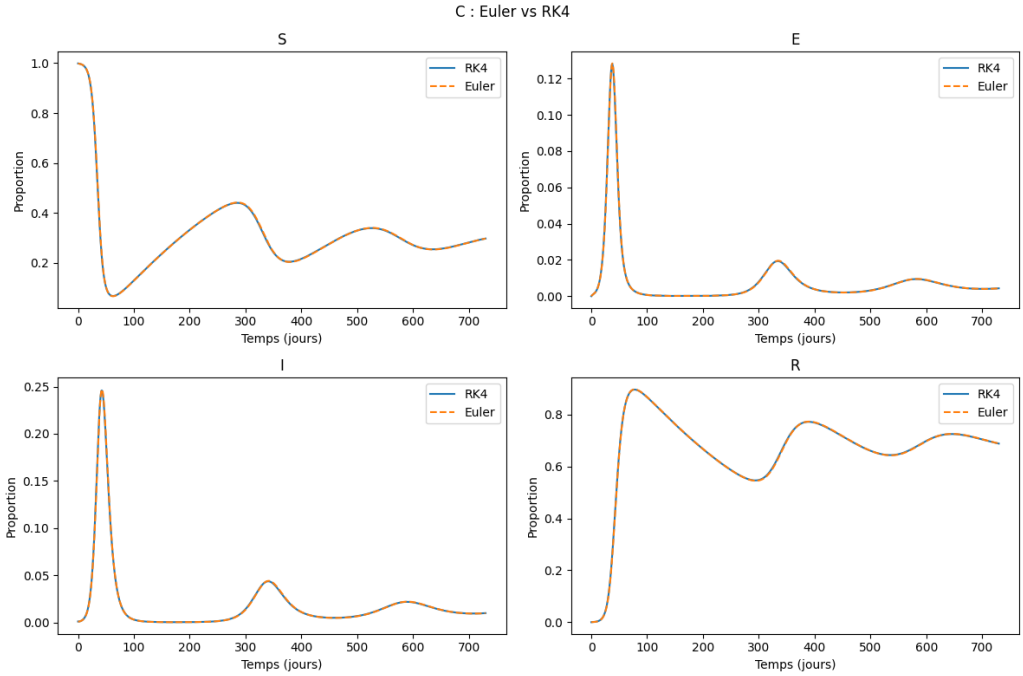

FIGURE 1.3 – Comparaison entre Euler et RK4 en C

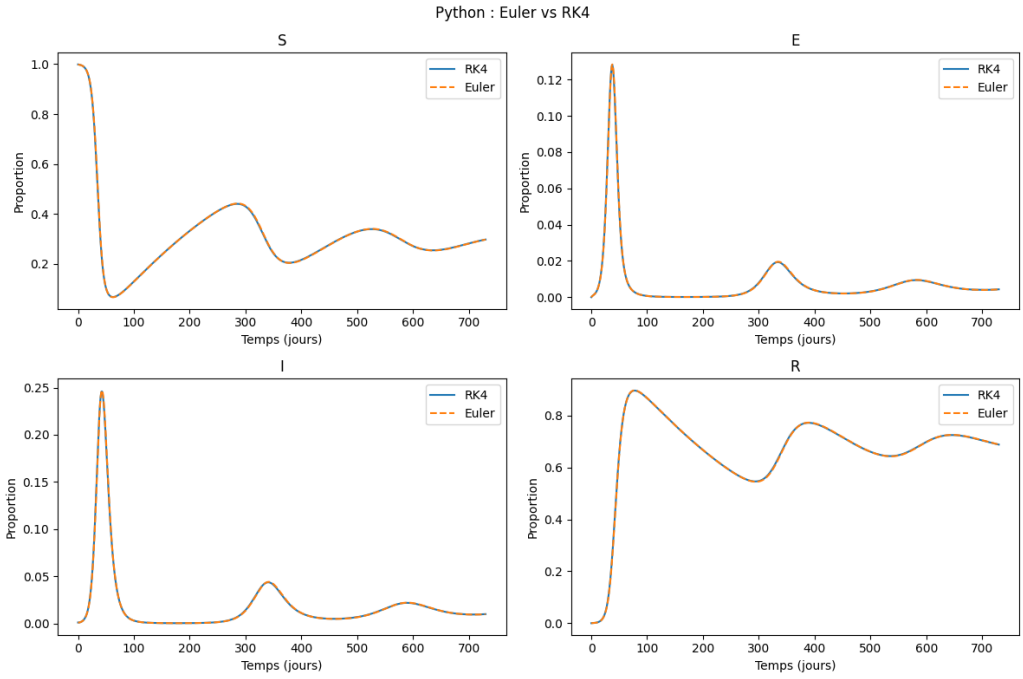

FIGURE 1.4 – Comparaison entre Euler et RK4 en Python

J'ai ensuite comparé les deux méthodes de calcul au sein d'un même langage.

— **En Python et en C :** On remarque que les courbes d'Euler et de RK4 se superposent très bien. C'est un résultat logique : même si mathématiquement RK4 est bien plus précis qu'Euler, notre problème est ici « bien posé » et pas trop complexe. La méthode d'Euler est donc suffisante pour obtenir un résultat très proche de la réalité sur cette simulation.

1.4 Analyse des erreurs numériques

Pour aller plus loin que l'observation visuelle, j'ai calculé l'erreur maximale (la différence la plus grande) entre les méthodes et les langages.

1.4.1 Comparaison entre langages (Même méthode)

On voit que les différences sont minuscules, ce qui confirme que le langage n'influence pas le calcul :

— **RK4 (Python vs C) :** L'erreur maximale est d'environ 0.00015 (pour le compartiment E).

— **Euler (Python vs C) :** L'erreur est encore plus petite, de l'ordre de 4.9×10^{-7}.

1.4.2 Comparaison entre méthodes (Même langage)

Ici, on voit que l'écart est un peu plus grand, ce qui montre la différence de précision entre les deux algorithmes :

— **En C (Euler vs RK4) :** L'écart maximal est de 0.0109 (soit environ 1%).

— **En Python (Euler vs RK4) :** On obtient un résultat similaire avec un écart maximal de 0.0109.

Voici le tableau complet des résultats :

TABLE 1.1 – Erreurs maximales entre les différentes implémentations

Comparaison	**Erreur Max (S)**	**Erreur Max (E)**	**Erreur Max (I)**	**Erreur Max (**
RK4 Python vs C	0.000071	0.000151	0.000129	0.000040
Euler Python vs C	4.99×10^{-7}	4.99×10^{-7}	4.99×10^{-7}	4.99×10^{-7}
Euler vs RK4 (C)	0.010912	0.002175	0.003677	0.008575
Euler vs RK4 (Python)	0.010938	0.002262	0.003683	0.008611

1.4.3 Analyse du tableau

— Les erreurs entre Python et C (pour Euler) sont extrêmement faibles ($\approx 10^{-7}$), ce qui prouve que mon code manuel est identique dans les deux langages.

— Les erreurs entre Euler et RK4 (environ 0.01) montrent que RK4 apporte une petite correction de précision par rapport à Euler, mais l'écart reste très faible (environ 1%), ce qui confirme que les deux méthodes sont adaptées pour ce projet.

1.5 Conclusion

Cette étude montre que pour le modèle SEIRS, le choix du langage n'a pas d'impact sur la précision. De plus, bien que RK4 soit théoriquement meilleure, la méthode d'Euler donne des résultats très satisfaisants pour ce système.

Chapitre 2

Implémentation multi-langages et analyse des résultats

Dans cette partie, j'ai implémenté le modèle multi-agent SEIRS dans trois langages de programmation utilisés en calcul haute performance (HPC) : Python, C et C++.

Pour la génération des nombres aléatoires, j'ai utilisé le générateur pseudo-aléatoire Mersenne Twister (MT19937), afin d'assurer une bonne qualité statistique des simulations.

Chaque simulation a été répétée 30 fois pour chaque langage, ce qui permet de réduire l'influence du hasard et d'obtenir des résultats plus fiables.

2.1 Graphe 1 : Moyennes des compartiments S, E, I et R

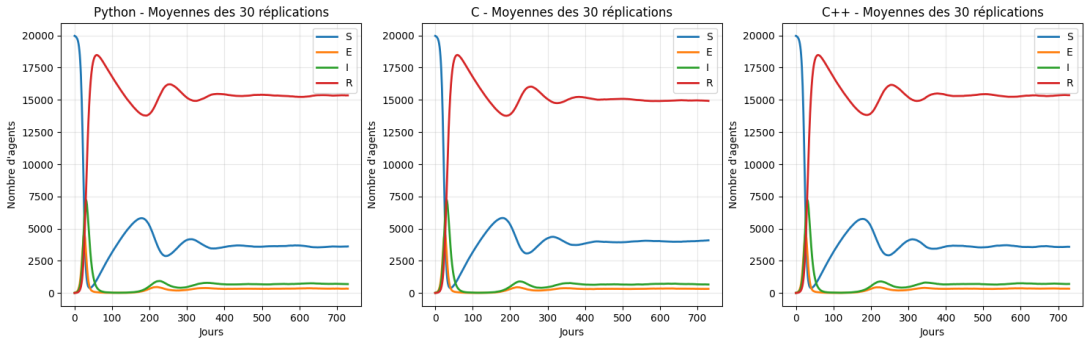

FIGURE 2.1 – Courbes moyennes des compartiments S, E, I et R pour Python, C et C++ (moyenne de 30 réplications)

Dans cette partie, j'ai tracé les courbes moyennes des compartiments S (Susceptibles), E (Exposés), I (Infectieux) et R (Rétablis) pour les trois langages : Python, C et C++.

Pour chaque langage, les résultats correspondent à la moyenne de 30 réplications, afin de limiter l'effet de l'aléa lié au caractère stochastique du modèle.

J'observe que les quatre compartiments présentent une dynamique très similaire dans les trois implémentations. Les phases d'augmentation et de diminution des populations S, E, I et R sont quasiment identiques.

Les pics d'infection apparaissent aux mêmes périodes et avec des amplitudes proches. De même, les phases de stabilisation à long terme sont comparables entre les trois langages.

Ces résultats montrent que les implémentations en Python, C et C++ produisent des comportements équivalents du modèle SEIRS. Les faibles différences observées peuvent être attribuées aux générateurs aléatoires et aux erreurs numériques propres à chaque langage.

2.2 Graphe 2 : Validation inter-langages – Analyse du compartiment I

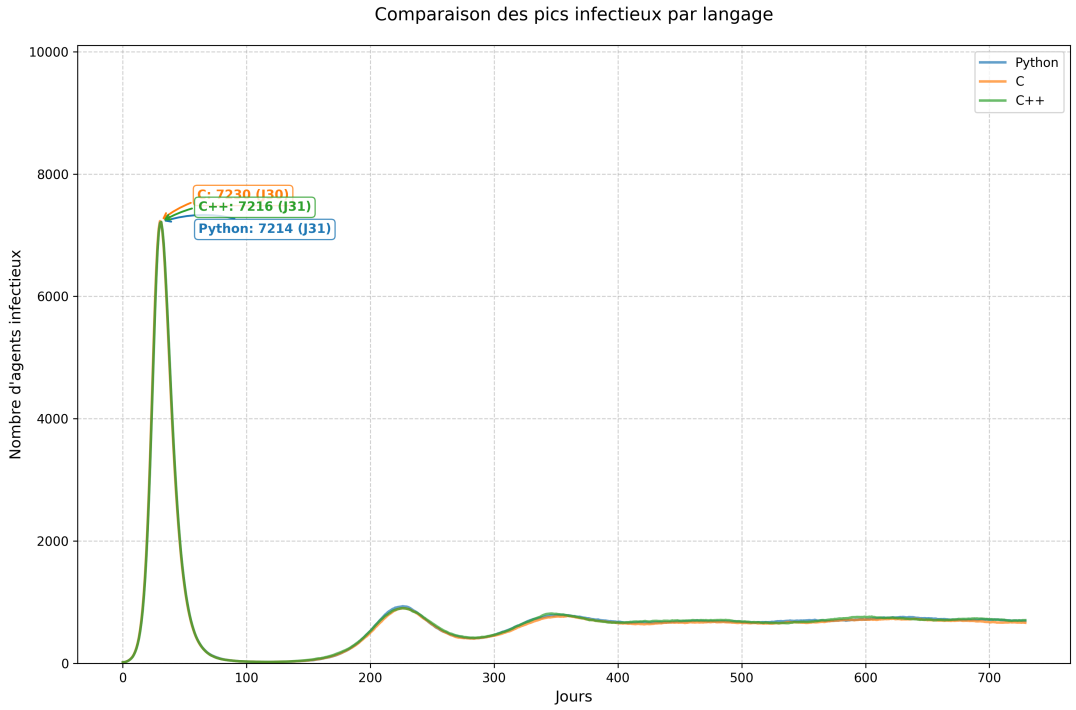

FIGURE 2.2 – Comparaison du compartiment des infectieux (I) pour Python, C et C++

Dans cette partie, j'ai tracé la courbe du compartiment des infectieux (I) pour les trois langages : Python, C et C++.

Le compartiment I est le plus important dans l'étude de l'épidémie, car il représente le nombre d'individus infectieux à chaque instant et permet d'évaluer l'intensité de la propagation.

Les résultats montrent que la hauteur du premier pic de I est comprise entre 7214 et 7230 agents, et que sa date d'apparition se situe autour des jours 30 et 31 pour les trois langages.

Cette figure met en évidence une forte similarité entre les courbes obtenues avec Python, C et C++. Les dynamiques sont visuellement très proches, tant au niveau de l'amplitude que de la temporalité des pics.

NB : Cette similarité visuelle suggère que les trois implémentations produisent des résultats cohérents. Elle sera confirmée par l'analyse statistique présentée dans la section suivante.

2.3 Graphe 3 : Analyse statistique des résultats

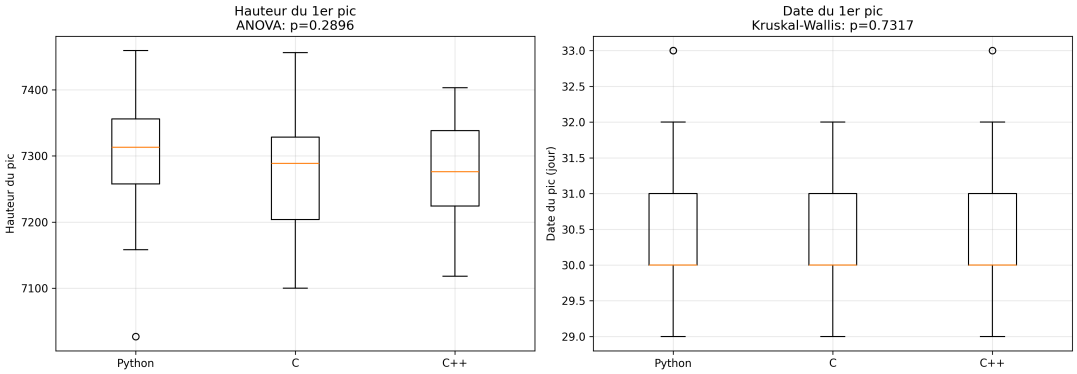

FIGURE 2.3 – Analyse statistique comparative des trois langages

2.3.1 Question étudiée

Les 30 réplications en C donnent-elles les mêmes résultats que les 30 réplications en C++ et Python ?

2.3.2 Réponse

Oui, les trois langages produisent des résultats statistiquement identiques.

2.3.3 Justification

Hauteur du premier pic

- **Moyennes très proches :**
  - Python : 7302.63
  - C : 7273.97
  - C++ : 7270.90
- **Test de normalité (Shapiro-Wilk) :**
  - Toutes les p-values > 0.05

  $\rightarrow$ Les données suivent une distribution normale.
- **Test ANOVA :**
  - p-value $= 0.2896$ (> 0.05)
- **Conclusion :**
  - Il n'existe pas de différence significative entre les langages pour la hauteur du pic.

Date du premier pic

- **Moyennes très proches :**
  - Python : 30.53 jours
  - C : 30.27 jours
  - C++ : 30.40 jours
- **Test de normalité :**

  - Toutes les p-values < 0.05
  - $\rightarrow$ Les données ne suivent pas une distribution normale.
- **Test de Kruskal-Wallis :**
  - p-value $= 0.7317$ (> 0.05)
- **Conclusion :**
  - Il n'existe pas de différence significative entre les langages pour la date du pic.

```
ANALYSE STATISTIQUE

Python - Hauteur du pic: moyenne=7302.63, std=90.30
C      - Hauteur du pic: moyenne=7273.97, std=84.47
C++    - Hauteur du pic: moyenne=7270.90, std=76.91

Python - Date du pic: moyenne=30.53, std=1.06
C      - Date du pic: moyenne=30.27, std=0.73
C++    - Date du pic: moyenne=30.40, std=0.92

------------------------------------------------------------
TEST DE NORMALITÉ (Shapiro-Wilk)
------------------------------------------------------------

Hauteur du pic:
  Python: p-value = 0.3105
  C:      p-value = 0.7670
  C++:    p-value = 0.2612

Date du pic:
  Python: p-value = 0.0030
  C:      p-value = 0.0005
  C++:    p-value = 0.0018

Distribution normale (alpha=0.05):
...
Test de Kruskal-Wallis
Statistique: 0.6248
P-value: 0.7317
Conclusion: PAS de différence significative (p >= 0.05)
```

FIGURE 2.4 – Résultats des tests statistiques pour la hauteur et la date du premier pic

2.3.4 Interprétation globale

Les deux tests statistiques (ANOVA et Kruskal-Wallis) montrent que Python, C et C++ produisent des résultats équivalents sur le plan statistique.

Les légères différences observées dans les moyennes sont dues à la stochasticité du modèle, c'est-à-dire à la variabilité aléatoire inhérente aux simulations, et non à des erreurs d'implémentation.

Ces résultats valident la cohérence, la fiabilité et la correction des trois implémentations du modèle multi-agent SEIRS.

2.4 Conclusion de la partie 2

Cette partie montre que le modèle SEIRS a été correctement implémenté dans les trois langages étudiés. Les résultats obtenus sont cohérents aussi bien visuellement que statistiquement.

Le passage de Python à C et C++ n'a pas modifié le comportement du modèle, ce qui confirme la robustesse de l'implémentation et la reproductibilité des simulations.

Chapitre 3

Bonus – Analyse des performances et de la consommation d'énergie

Dans cette partie, je me suis intéressé aux performances des trois implémentations (Python, C et C++) en termes de temps d'exécution, de consommation mémoire et d'énergie.

3.1 Mesure des performances : temps et mémoire

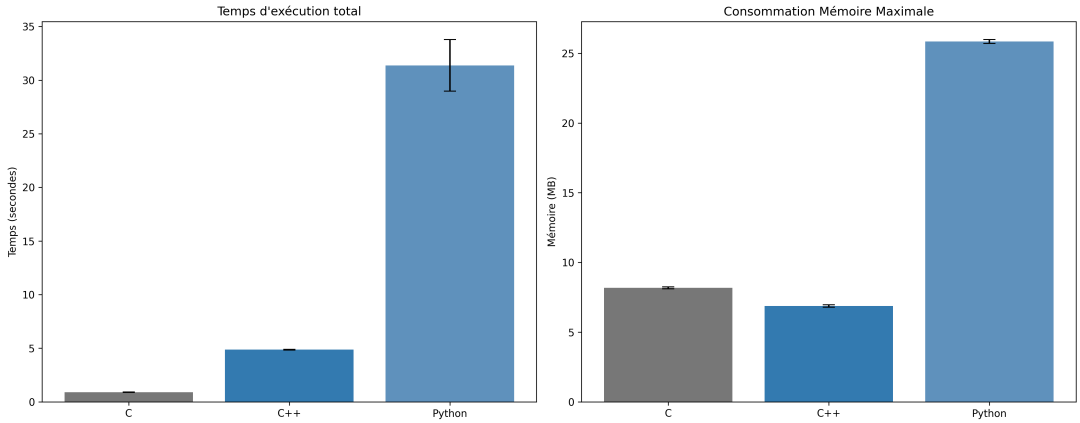

FIGURE 3.1 – Comparaison des performances : temps d'exécution et consommation mémoire

Pour réaliser ces mesures, j'ai utilisé un script bash permettant d'exécuter chaque programme cinq fois, puis de calculer une moyenne des résultats.

Pour cette partie, je n'ai utilisé qu'une seule réplication, car effectuer 30 réplications aurait rendu les temps d'exécution trop longs.

L'analyse des performances montre des résultats cohérents avec les caractéristiques des langages :

— **Temps d'exécution :**

Le langage C est le plus rapide avec un temps moyen de 0,90 s, suivi du C++ (4,87 s).

Python est beaucoup plus lent (31,38 s), ce qui s'explique par son caractère interprété et par la gestion des objets.

— **Consommation mémoire :**

Le C++ est le plus économe (6,88 MB), suivi du C (8,18 MB).

Python consomme beaucoup plus de mémoire (25,85 MB), notamment à cause de l'interpréteur et de la gestion dynamique des données.

Conclusion :

Le C est le plus rapide, le C++ offre un bon compromis entre vitesse et mémoire, et Python montre ses limites pour des simulations de grande taille.

3.2 Mesure de la consommation d'énergie

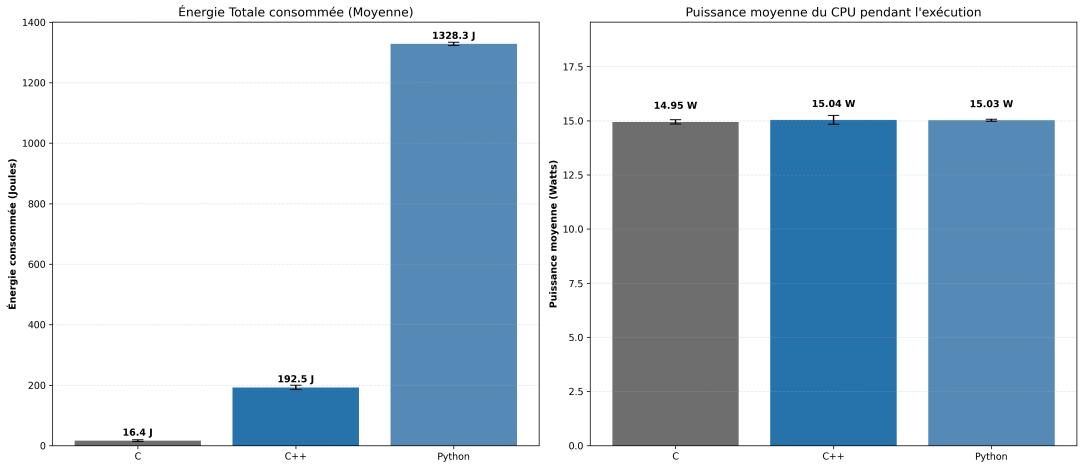

FIGURE 3.2 – Comparaison de la consommation d'énergie entre Python, C et C++

La consommation d'énergie a été mesurée de la même manière, avec cinq exécutions pour chaque langage. Une moyenne a ensuite été calculée.

Comme le programme en C s'exécute très rapidement, j'ai dû augmenter le nombre de réplications afin que l'outil PowerJoular puisse mesurer correctement la consommation. (Je suis passé de 1 réplication à 4)

Les résultats montrent que :

— Le C consomme très peu d'énergie (16,43 J),

— Le C++ consomme davantage (192,47 J),

— Python est le plus énergivore (1328,27 J).

Cette différence s'explique principalement par le temps d'exécution : la puissance consommée est similaire, mais le C termine beaucoup plus vite.

3.3 Analyse globale et choix du langage

L'analyse conjointe du temps d'exécution, de la mémoire et de l'énergie permet de comparer clairement les trois langages :

— Le C est le plus rapide et le plus économe en énergie,

— Le C++ est un bon compromis,

— Python est plus simple à utiliser mais moins performant.

Conclusion générale :

Au vu des résultats, le langage C est le plus adapté pour ce projet.

Il offre le meilleur rapport performance/consommation d'énergie.

Dans un contexte de calcul haute performance et de développement durable, le C permet de réduire les coûts de calcul et l'impact environnemental, tout en conservant des résultats fiables.

Chapitre 4

Reproductibilité des expériences

Dans le cadre de ce projet, une attention particulière a été portée à la reproductibilité des expériences, conformément aux bonnes pratiques en calcul scientifique et en HPC.

Pour les parties 1, 2 et Bonus, trois environnements virtuels Python distincts ont été créés, chacun associé à son propre dossier (`Partie_1`, `Partie_2`, `Partie_Bonus`). Ces environnements permettent d'isoler les dépendances logicielles et d'éviter les conflits entre bibliothèques.

Dans chaque dossier, un fichier `requirements.txt` a été généré à l'aide de la commande `pip freeze`. Ce fichier contient la liste complète des bibliothèques Python utilisées ainsi que leurs versions.

Grâce à ces fichiers, toute personne souhaitant reproduire les résultats peut facilement recréer l'environnement de travail en utilisant les commandes suivantes :

```
python3 -m venv env
source env/bin/activate
pip install -r requirements.txt
```

Cette démarche garantit que les simulations et les analyses pourront être exécutées dans les mêmes conditions logicielles que celles utilisées lors du développement du projet.

Ainsi, les résultats présentés dans ce rapport sont reproductibles et peuvent être vérifiés par d'autres utilisateurs.

Conclusion

Ce projet a permis d'appliquer concrètement les notions de modélisation, de calcul scientifique et de calcul haute performance (HPC) à travers l'étude d'un modèle épidémiologique SEIRS.

L'implémentation du modèle dans plusieurs langages (Python, C et C++) a permis de comparer à la fois la précision des résultats, les performances, et la consommation d'énergie. Les analyses montrent que les trois langages produisent des résultats cohérents sur le plan scientifique, ce qui valide la qualité des implémentations réalisées.

Cependant, ce projet met également en évidence les limites du langage Python pour des simulations de grande taille en HPC, notamment en termes de temps d'exécution et de consommation de ressources. À l'inverse, le langage C offre d'excellentes performances et une meilleure efficacité énergétique.

Ce travail illustre ainsi l'importance du choix du langage et des outils dans les projets de calcul scientifique.

Pour la suite, il serait intéressant d'explorer des pistes d'optimisation, comme l'utilisation du parallélisme, des bibliothèques spécialisées, ou encore l'optimisation mémoire, afin d'améliorer davantage les performances.

En conclusion, ce projet constitue une expérience formatrice en modélisation, en programmation scientifique et en HPC, et permet de mieux comprendre les enjeux liés aux simulations à grande échelle.