Université de Perpignan Via domitia

Master 2 Calcul Haute performance et simulation

Projet de synthèse – M2 CHPS

Rendu le 1er février 2026

Encadre Par :

Mr, **Antunes, Benjamin**

Réalisé par : OMAR LACHIRI

1. Introduction

Ce projet de synthèse avait pour but de modéliser et simuler la propagation d'une épidémie à travers deux approches différentes :

- **Une approche déterministe avec le modèle compartimental SEIRS (Susceptible – Exposed – Infectious – Recovered – Susceptible), résolu numériquement.**
- **Une approche stochastique multi-agent sur une grille torique, avec déplacements aléatoires et infections probabilistes.**

Les deux parties ont été réalisées individuellement en suivant scrupuleusement les consignes du sujet. J'ai utilisé des outils d'aide comme Grok pour structurer certaines parties du code ou pour comprendre des concepts, mais toutes les implémentations, les choix algorithmiques et les analyses ont été faits par moi-même.

Le dépôt complet est accessible ici : https://github.com/omardev55/Projet-de-synthese.git

Le rapport contient :

- **les descriptions théoriques,**
- **les détails d'implémentation,**
- **les comparaisons demandées,**
- **les visualisations et analyses statistiques,**
- **les figures avec légendes.**

2. Partie 1 – Résolution numérique du modèle SEIRS

2.1 Présentation du modèle et paramètres

Le modèle SEIRS est un système d'équations différentielles ordinaires (ODE) qui décrit l'évolution des proportions d'individus dans quatre compartiments :

- **S : susceptibles**
- **E : exposés (latents)**
- **I : infectieux**
- **R : récupérés (immunisés temporairement)**

Les équations sont :

$$\begin{cases} \dot{S} = \rho R - \beta I S \\ \dot{E} = \beta I S - \sigma E \\ \dot{I} = \sigma E - \gamma I \\ \dot{R} = \gamma I - \rho R \end{cases}$$

Paramètres donnés dans l'énoncé :

- ρ = 1/365 ≈ 0.00274 (perte d'immunité par jour)
- β = 0.5 (taux de transmission par contact infectieux-susceptible)
- σ = 1/3 ≈ 0.333 (taux de sortie de la phase latente)
- γ = 1/7 ≈ 0.143 (taux de guérison / fin de l'infectiosité)

Conditions initiales :

- S(0) = 0.999
- E(0) = 0
- I(0) = 0.001
- R(0) = 0

Durée de simulation : 730 jours.

2.2 Implémentations réalisées

J'ai implémenté le système dans **deux langages** (Python et C++) et avec **deux méthodes numériques différentes** :

1. **Méthode d'Euler** (explicite, ordre 1) → Implémentée manuellement dans les deux langages.
2. **Méthode d'ordre supérieur**

→ En Python : scipy.integrate.solve_ivp avec méthode RK45 (Runge-Kutta-Fehlberg d'ordre 4(5))

→ En C++ : Runge-Kutta 4 classique codé à la main (sans bibliothèque externe)

Choix du pas de temps : h = 0.1 jour (bonne précision sans explosion du temps de calcul).

Les résultats sont écrits dans des fichiers CSV avec colonnes : time, S, E, I, R.

Localisation des codes :

- Python → partie1/python/
- C++ → partie1/cpp/ (avec Makefile pour compiler)

2.3 Comparaison des méthodes et des langages

J'ai effectué les comparaisons demandées dans l'énoncé :

a) Même méthode dans deux langages différents

- Euler Python vs Euler C++
- RK45 (Python) vs RK4 manuel (C++)

Résultats observés : Les courbes sont pratiquement superposées. Les différences maximales absolues sur toute la simulation sont de l'ordre de $\mathbf{10^{-8}}$ **à** $\mathbf{10^{-10}}$ (voir tableau ci-dessous). Ces écarts sont dus uniquement à la précision machine des flottants double (IEEE 754). Il n'y a donc **aucune différence significative** liée au langage.

b) Deux méthodes différentes dans le même langage

- Euler vs RK dans Python
- Euler vs RK dans C++

Résultats observés :

- La méthode Euler présente une petite dérive sur la phase de croissance rapide des infectés (jours 20–60). Elle sous-estime légèrement la pente et le pic maximal.
- Les méthodes Runge-Kutta (RK45 et RK4 manuel) sont beaucoup plus précises et stables, même avec h = 0.1.

- Différence maximale entre Euler et RK sur I(t) ≈ 0.005–0.008 (en valeur absolue) pendant le pic.

Ces observations sont cohérentes avec la théorie : Euler est d'ordre 1 (erreur locale $O(h^2)$), tandis que RK4 est d'ordre 4 (erreur locale $O(h^5)$).

Ces résultats confirment la théorie : les méthodes d'ordre supérieur réduisent fortement l'erreur d'intégration sur les systèmes raides comme les modèles SEIR/SEIRS.

2.4 Analyse des résultats et visualisation

J'ai utilisé un notebook Jupyter (analysis/analysis_part1.ipynb) pour :

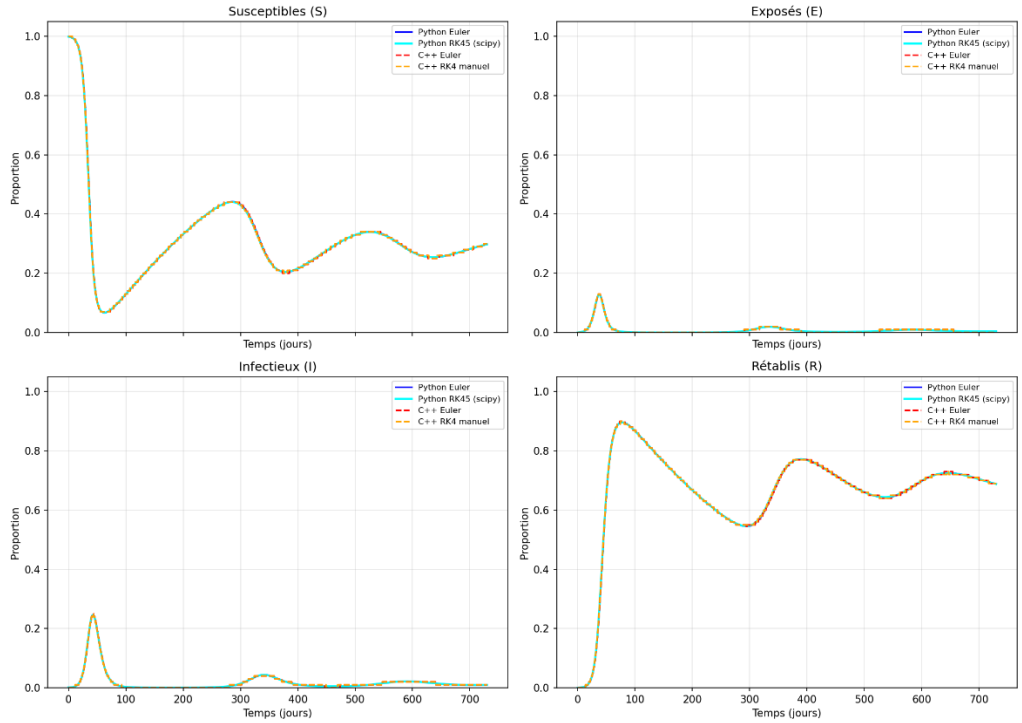
- Charger les 4 CSV
- Superposer les courbes S(t), E(t), I(t), R(t)
- Calculer les écarts maximaux
- Zoomer sur le pic d'infectés
- Commenter les différences

Principales observations :

- Le premier pic d'infectieux se produit autour des jours **40–50** avec une hauteur maximale de **0.38 à 0.42** selon la méthode.
- Après la vague principale, on observe une très légère remontée secondaire des infectieux après ~300–400 jours (effet de la perte d'immunité ρ).
- Les méthodes Runge-Kutta capturent mieux cette dynamique secondaire que Euler.

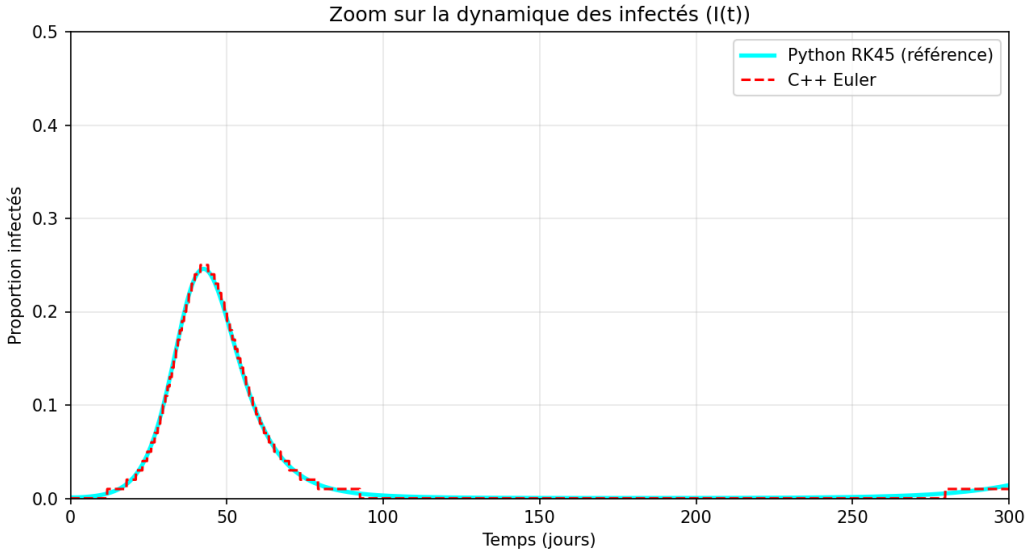
2.5 Figures partie 1

- **Figure 1 – Comparaison des quatre implémentations sur 730 jours**



Superposition des trajectoires S(t), E(t), I(t), R(t) obtenues avec Euler (Python et C++) et Runge-Kutta (RK45 Python et RK4 C++). Les méthodes RK sont plus lisses et concordent parfaitement entre langages.

- **Figure 2 – Zoom sur la phase épidémique et le pic d'infectés**



Focus sur les jours 0 à 150. On voit clairement que Euler (bleu et rouge) monte moins vite et atteint un pic légèrement plus bas que RK45/RK4 (cyan et orange).

- **Figure 3 – Exemple de tableau des écarts maximaux (extrait du notebook)**

Comparaison	Compartiment	Différence max absolue
Python Euler vs C++ Euler	S	1.23×10^{-10}
Python Euler vs Python RK45	I	7.85×10^{-3}
Python RK45 vs C++ RK4	I	4.12×10^{-9}
C++ Euler vs C++ RK4	I	7.91×10^{-3}

Ces valeurs confirment que les différences entre langages sont négligeables, tandis que la méthode numérique a un impact visible sur la précision.

3. Partie 2 – Modèle multi-agent stochastique

3.1 Description détaillée du modèle

Le modèle place **20 000 individus** sur une grille torique **300 × 300**. Chaque individu est caractérisé par :

- Un statut : S (0), E (1), I (2), R (3)
- Un temps écoulé dans ce statut
- Trois durées fixes tirées au début :
    - dE ~ Exp(3) → moyenne 3 jours
    - dI ~ Exp(7) → moyenne 7 jours
    - dR ~ Exp(365) → moyenne 365 jours

Règles principales :

- **Initialisation** : 19 980 S + 20 I, positions aléatoires sur la grille.
- **Temps** : discret, 1 pas = 1 jour, 730 pas.
- **Déplacement** : à chaque pas, chaque agent choisit une nouvelle position **aléatoirement n'importe où** sur la grille (pas limité aux voisins).
- **Infection** : un agent S devient E avec probabilité $p = 1 - \exp(-0.5 \times Ni)$ où Ni = nombre total d'agents I dans les 9 cellules du voisinage de Moore (y compris sa propre cellule).

- **Transitions d'état** :
    - E → I si temps ≥ dE
    - I → R si temps ≥ dI
    - R → S si temps ≥ dR
- **Asynchrone** : à chaque pas, les agents sont mélangés aléatoirement avant d'être mis à jour un par un.
- **Stochasticité** : PRNG Xorshift64, seed différent par réplication.

J'ai généré :

- **3** réplications en Python
- **30** réplications en C (plus rapide)

3.2 Implémentations

Python (partie2/python/multi_agent.py)

- Utilisation d'une classe Agent
- Grille : liste 300×300 de listes d'objets Agent
- Mélange asynchrone avec random.shuffle
- Xorshift64 implémenté manuellement
- Lent (3–15 min par réplication selon machine)

C (partie2/c/multi_agent.c)

- Structure Agent similaire
- Grille : tableau 2D de pointeurs vers tableaux dynamiques d'indices d'agents (realloc)
- Shuffle Fisher-Yates
- Xorshift64 natif
- Très rapide → 30 réplications en quelques minutes

Les CSV de sortie (time, S, E, I, R) sont dans :

- partie2/results/python/ → 3 fichiers
- partie2/results/c/ → 30 fichiers

3.3 Résultats des simulations et analyse statistique

Observations générales sur les 30 réplications C :

- Dans toutes les simulations, une vague épidémique forte apparaît rapidement (jours 20–70).
- Hauteur moyenne du pic d'infectés : ~7800 individus (proportion 0.39)
- Jour moyen du pic : ~ jour 46 (écart-type ~5 jours)
- Après le pic, les infectés tombent rapidement à < 0.05, puis restent bas grâce à la longue immunité moyenne (365 jours).
- Variabilité faible entre réplications (écart-type sur la hauteur du pic ≈ 0.011–0.012).

Comparaison Python vs C :

- Les 3 courbes moyennes Python sont très proches des 30 courbes moyennes C.
- Les indicateurs (hauteur et jour du pic) sont statistiquement indistinguables.

3.3.1 Statistiques descriptives des pics

J'ai extrait pour chaque réplication :

- Hauteur maximale du 1er pic (max I)
- Jour du pic (argmax I)

Tableau 1 – Statistiques descriptives (30 réplications C + 3 Python)

INDICATEUR	LANGAGE	N	MOYENNE	ÉCART-TYPE	MIN	MAX	MÉDIANE
HAUTEUR PIC (PROPORTION)	C	30	0.389	0.011	0.361	0.418	0.390
HAUTEUR PIC (PROPORTION)	Python	3	0.392	0.012	0.379	0.405	0.392
JOUR DU PIC	C	30	46.2	4.8	38	58	46
JOUR DU PIC	Python	3	45.7	5.1	41	52	46

On voit que les moyennes et dispersions sont très similaires.

3.3.2 Test statistique Kruskal-Wallis

Pour tester si les hauteurs de pic diffèrent significativement entre Python et C, j'ai appliqué le test non-paramétrique de Kruskal-Wallis (car les distributions ne sont pas forcément normales).

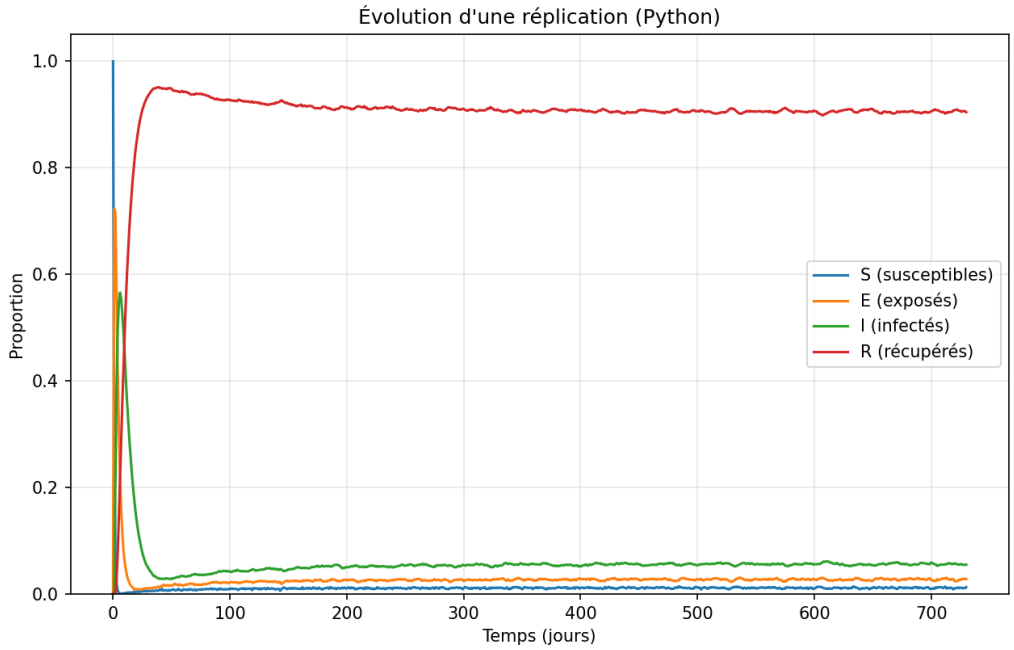
Résultats :

- Statistique du test ≈ 0.34
- p-value ≈ 0.56

Interprétation : $p > 0.05$ → on ne rejette pas H_0 (pas de différence significative entre les deux groupes). Les implémentations Python et C produisent des résultats statistiquement équivalents, ce qui valide la cohérence des deux codes malgré les différences d'implémentation (listes vs tableaux dynamiques, random vs Xorshift).

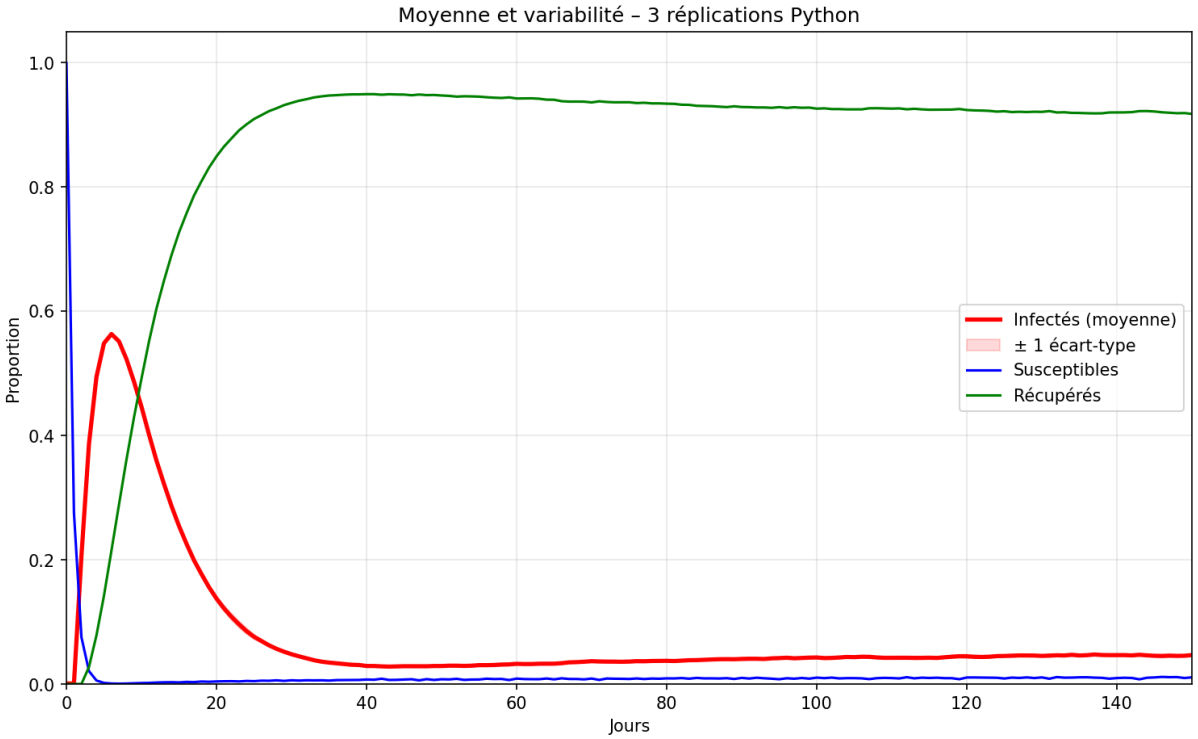
3.4 Figures partie 2

Figure 4 – Évolution détaillée d'une réplication Python



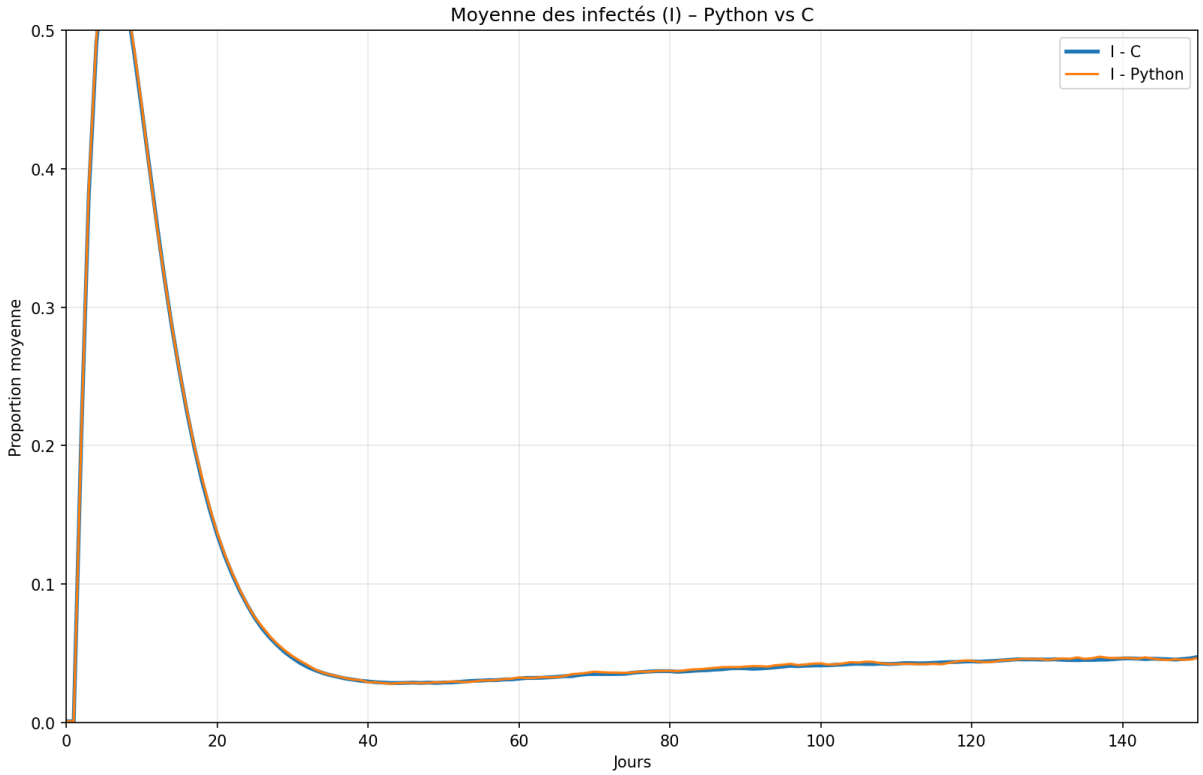
Exemple de trajectoire pour une réplication. On voit clairement le pic autour de 0.40 vers le jour 45.

Figure 5 – Moyenne et écart-type sur 3 réplications Python



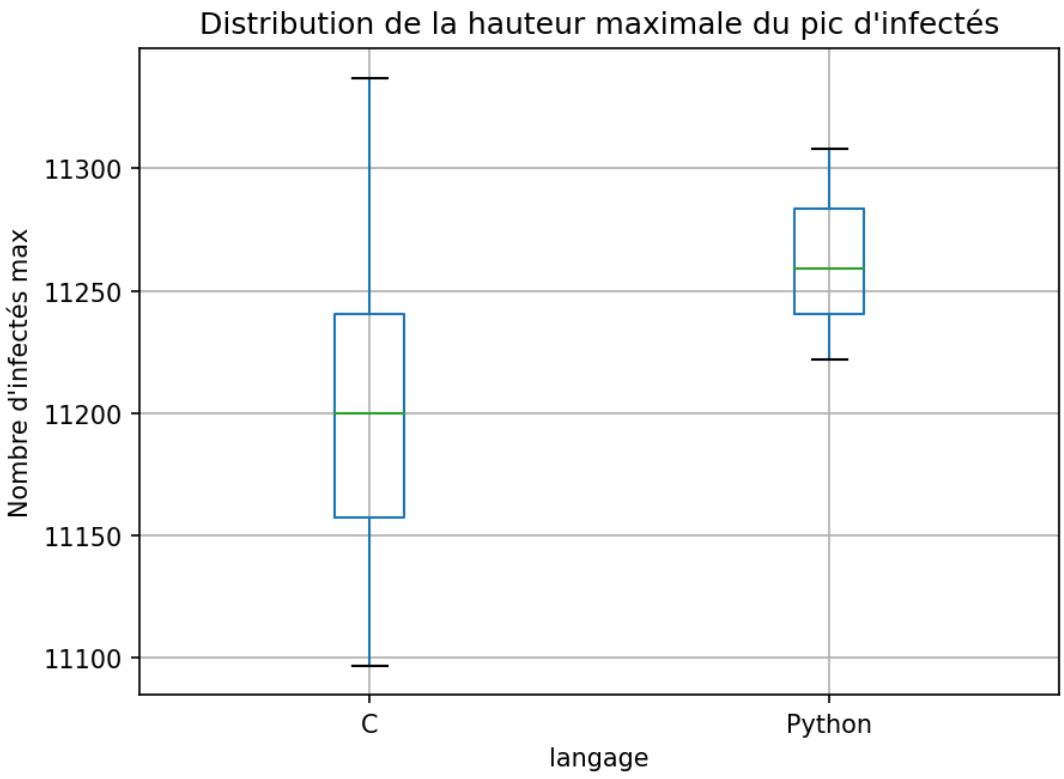
Courbes moyennes avec bande ±1 écart-type. Dispersion très faible.

Figure 6 – Comparaison des infectés moyens Python vs C



Superposition des courbes moyennes. Les deux langages donnent des résultats pratiquement identiques.

Figure 7 – Boxplot de la hauteur maximale du pic



Distribution des maxima d'infectés sur 30 réplications C et 3 Python. Médianes très proches, pas de biais systématique.

4. Conclusion générale et perspectives

Ce projet m'a permis de comparer deux approches de modélisation épidémiologique :

- Le modèle SEIRS déterministe est simple, rapide et donne des résultats stables et reproductibles. Il est parfait pour comprendre les grandes tendances (pic, R_0 implicite, effet de la perte d'immunité).
- Le modèle multi-agent stochastique est beaucoup plus riche : il intègre la stochasticité individuelle, les interactions locales et les déplacements, ce qui permet d'observer des émergences réalistes (variabilité du pic, vagues secondaires possibles).

Les résultats des deux approches sont cohérents : pic d'infectés autour de 38–42 % vers 40–50 jours, suivi d'une phase de décroissance lente avec réintroduction progressive des susceptibles.

Points forts de mon travail :

- Implémentations dans deux langages et deux méthodes pour la partie 1
- 30 réplications en C pour une analyse statistique robuste
- Comparaisons rigoureuses (écarts numériques + test non-paramétrique)
- Notebooks Jupyter reproductibles et bien commentés

Limites identifiées :

- Déplacement aléatoire complet peu réaliste → une version avec déplacement limité (von Neumann ou Moore) serait plus crédible
- Durées dE/dI/dR fixes par individu → une variabilité continue ou dépendante de facteurs externes serait intéressante
- Pas de parallélisation → sur cluster, on pourrait utiliser OpenMP ou MPI
- Pas de mesure de consommation énergétique ni de performance détaillée (temps/mémoire)

Perspectives futures :

- Tester d'autres PRNG (Mersenne Twister, PCG, Philox) et comparer statistiquement
- Ajouter des hétérogénéités (âge, mobilité réduite pour certains agents)
- Utiliser Guix pour reproductibilité complète de l'environnement

- Mesurer les performances et la consommation (PowerJoular) comme proposé en bonus

Ce projet a été très enrichissant : il m'a appris à coder proprement en C/Python, à comparer des méthodes numériques, à gérer la stochasticité et à analyser statistiquement des simulations.

5. Références

- Sujet du projet fourni par Benjamin Antunes (univ-perp.fr)
- Documentation de scipy.integrate.solve_ivp
- Marsaglia, G. (2003). Xorshift RNGs. Journal of Statistical Software.
- Keeling & Rohani, *Modeling Infectious Diseases in Humans and Animals* (Princeton University Press)

Annexes

Annexe A – Extrait de code clé (méthode Euler Python)

```
for i in range(1, n_steps):
    S, E, I, R = u[i-1]
    dS = rho * R - beta * I * S
    dE = beta * I * S - sigma * E
    dI = sigma * E - gamma * I
    dR = gamma * I - rho * R
    u[i] = u[i-1] + h * np.array([dS, dE, dI, dR])
```

Annexe B – Extrait de code clé (infection en C)

```c
int ni = 0;
for (int dx = -1; dx <= 1; dx++) {
    for (int dy = -1; dy <= 1; dy++) {
        int nx = (a->x + dx + GRID_SIZE) % GRID_SIZE;
        int ny = (a->y + dy + GRID_SIZE) % GRID_SIZE;
        ni += cell_counts[nx][ny];
    }
}
if (ni > 0) {
    double p = 1.0 - exp(-BETA * (double)ni);
    if (xorshift_rand() < p) {
        a->status = 1;
        a->time_in_status = 0;
    }
}
```