Modèle Toy case SEIR multi-agent

Stephen BOUCHARDON, Jean BAREKZAI, Saphir GOBBI

April 7, 2024

Contents

List of Figures

1 Introduction

Ce projet a pour but de réaliser une simulation d'un modèle simplifié de la propagation d'une maladie avec différents états (Susceptible, Exposed, Infected, Recovered) dans une population en utilisant le langage Java. Le générateur de nombres pseudo-aléatoires utilisé dans l'entièreté du travail est le Mersenne Twister de Makoto Matsumoto. Pour ce faire, nous avons utilisé l'adaptation en Java implémentée par David Beaumont. Pour plus d'informations sur la mise en œuvre de Mersenne Twister en Java, consultez l'élément 1 de la bibliographie.

2 Développement

2.1 Structure du code

Human.java Classe pour créer un individu, avec les états, la durée de la vie des statuts et la position dans le monde.

MTRandom.java Classe du Mersenne Twister.

Monde.java Classe pour la simulation de l'épidémie et la création d'un tableau en deux dimensions représentant le monde.

Main.java Classe principale pour l'exécution du programme.

2.2 Analyse des résultats

Avec un même paramétrage, on trouve une extrême variation des courbes les 30 premiers jours, puis elle converge vers le 100ème jour à un état fixe : environ 2000 pour les individus susceptibles, environ 1000 pour les individus contaminés, environ 500 pour les individus exposés et 16000 pour les individus qui récupèrent. On observe entre les 100 réplications un état stable et cohérent. Au début de chaque propagation, comme nous avons choisi d'avoir 19980 individus susceptibles et 20 contaminés, il y a toujours une légère variation entre le nombre de susceptibles et d'exposés. Le nombre d'individus qui sont en récupération reste stable et toujours à 0 au départ. Ainsi, au vu de la stabilité des résultats trouvés, on en a déduit que la moyenne des courbes sera plus lisse, comme on peut le voir avec le résultat ci-dessous.

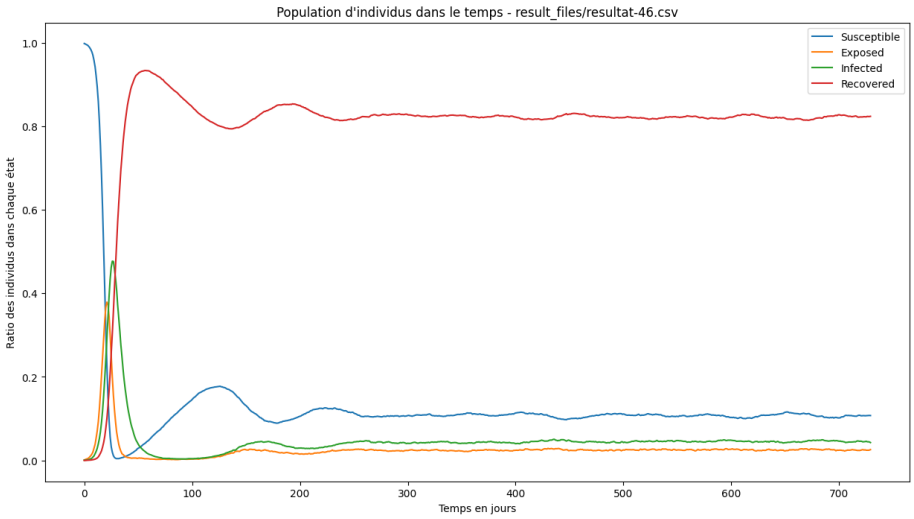

Figure 1: Représentation finale graphique d'une courbe de réplication

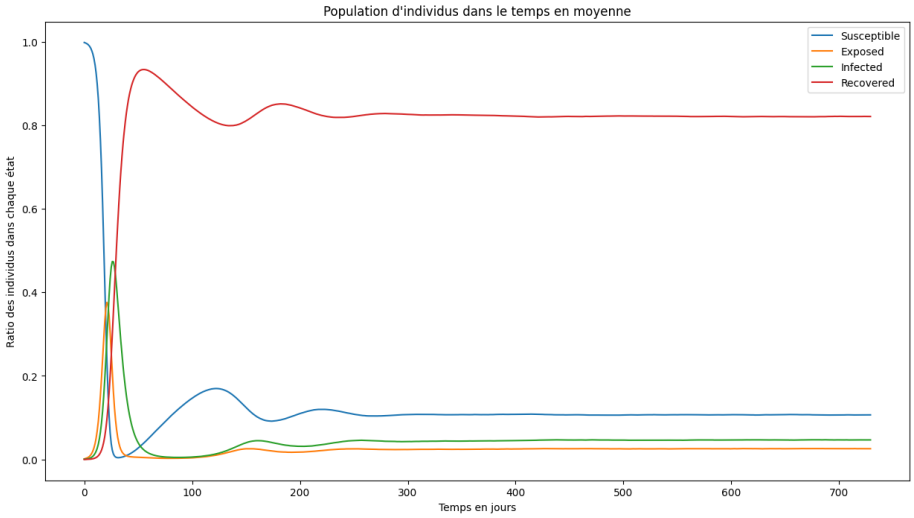

Figure 2: Représentation finale graphique de la courbe moyenne

2.3 Manière de reproduire les résultats

En premier lieu, vous devez exécuter le programme *Main.java* à l'aide des commandes suivantes dans votre terminal.

```
1.  javac -d build src/Main.java src/MTRandom.java src/Monde.java src/Human.java
2.  java -cp build Main
```

Le téléchargement des résultats prendra quelques minutes et se fera dans le répertoire $multi - agent/src/$ du projet. Ensuite, vous pourrez déplacer les 100 réplications dans un même répertoire $multi - agent/result - files/$. Il est important de réaliser ces étapes pour que le Jupyter Notebook puisse réaliser l'affichage des courbes des 100 simulations. Finalement, après avoir ouvert Jupyter, vous pourrez donc exécuter la première cellule de *resultat_jupyter_notebook.ipynb* qui donnera les 101 représentations graphiques des résultats dont la courbe moyenne.

3 Difficultés et solutions

À la fin de notre première tentative de représentations graphiques des 100 réplications, nous avons trouvé les résultats suivants avec une seed de 50.

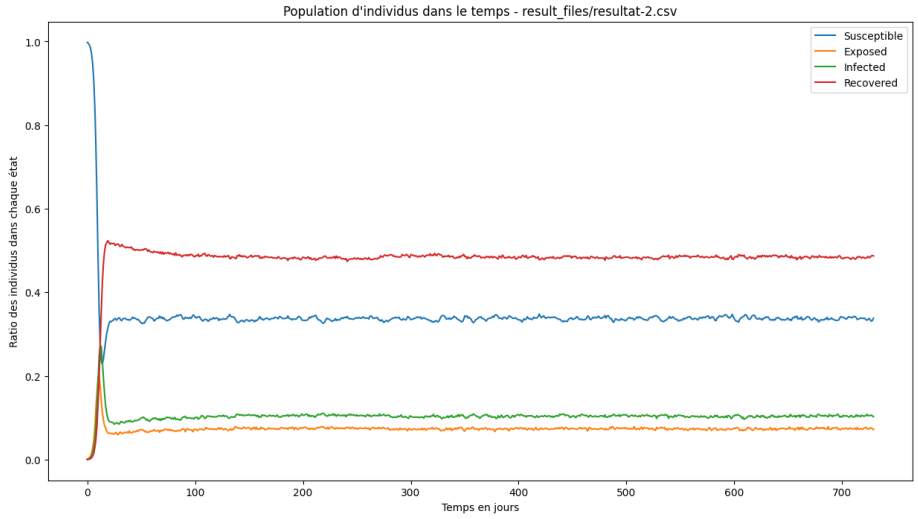

Figure 3: Premier essai de la représentation graphique d'une courbe de réplication

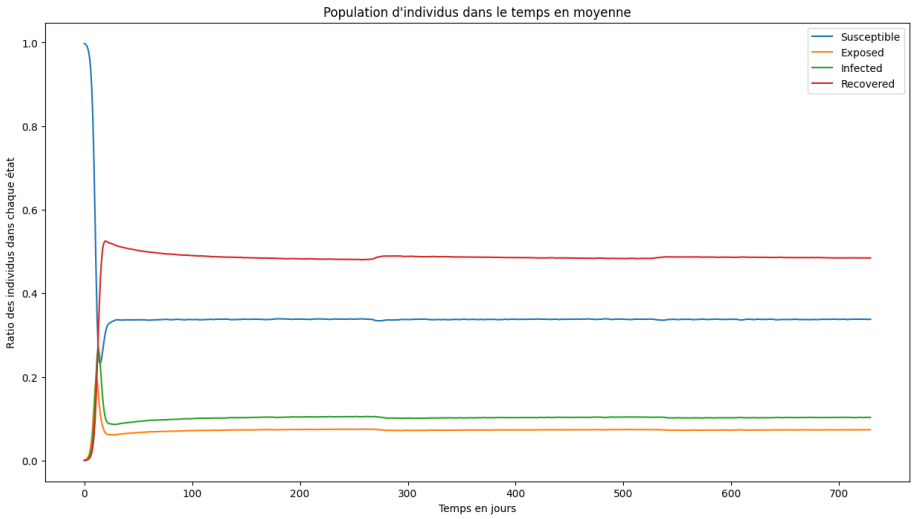

Figure 4: Premier essai de la représentation graphique de la courbe moyenne

En comparaison avec l'exemple vu sur le sujet, la courbe des Recovered de l'exemple de la feuille d'instruction du projet croît beaucoup plus que celles de nos exemples. Effectivement, les résultats sont cohérents du fait qu'ils ont tous une courbe qui converge vers le même ratio. Pour trouver une solution, nous avons essayé de remplacer la seed par 41, mais aucun changement n'a été visible. Pour commencer à trouver le problème, nous avons donc modifié le type des valeurs afin de stocker des nombres plus larges et précis. Aussi, nous avons remarqué que dans certains cas, on trouvait des valeurs négatives pour la position des individus dans le monde. Nous avons modifié les méthodes de position en conséquence pour éviter de se retrouver dans cette situation. On a aussi supposé une erreur dans l'implémentation des méthodes de changement d'état, du fait du faible ratio des individus Recovered. Finalement, après avoir réalisé les changements précédents, nous avons trouvé le problème pendant le débogage. Dans le fichier *MTRandom.java*, la méthode genRand1(), avant modification, réalisait l'opération suivante :

```
AVANT: return genRandInt32() * (1.0 / 4294967295.0);
APRÈS: return genRandInt32() * (2.0 / 4294967295.0);
```

genRandInt32() génère un entier pseudo-aléatoire de 32 bits signé. Ainsi, on le renvoie non signé en le multipliant par -1 dans sa méthode. Puis, avec la méthode genRand1() on le multiplie afin d'obtenir un double entre 0 et 1. Alors qu'auparavant, on obtenait un double entre 0 et 0.5.

4 Conclusion

Ce projet a été enrichissant du fait de la réutilisation de toutes les connaissances acquises tout au long du semestre en matière de génération de nombres pseudo-aléatoires avec l'application du Mersenne Twister en Java, différent du langage C habituellement utilisé. Les défis rencontrés tout au long de l'implémentation nous ont permis d'améliorer notre compréhension de la programmation orientée objet et de gestion des données. De plus, ce projet a constitué aussi une introduction en Jupyter Notebook. Il c'est avéré être très utile pour la visualisation des résultats. En conclusion, Les résultats obtenus offrent des perspectives intéressantes sur l'impact que peut avoir le paramètrage et les conditions initiales choisi sur la progression d'une épidémie.

5 Bibliographie

1. Mersenne Twister en Java `http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/VERSIONS/JAVA/java.html`