Rapport Tp4 simulation

Maxence CHUTAUX

Novembre 2024

Table des matières

Première partie

Introduction

1 Introduction

Le but de ce TP est de modéliser la croissance des lapins. Au début, on la modélise à l'aide de la suite de Fibonacci, qui crée des conditions simples pour cette modélisation. La seconde grande partie s'attardera sur la modélisation de la population de lapins avec des conditions plus précises, effectuée sur plusieurs années.

Deuxième partie

Fibonnaci

2 Résultat

```
import matplotlib.pyplot as plt

def fibobotup( x):
    L= [0 for i in range(x)]
    L[0]=1
    L[1]=1
    for i in range(2, x):
        L[i] = L[i-1] + L[i-2]
    return L


def fibotab(x):
    L = fibobotup(x)
    T= [i for i in range(x)]
    print(T)
    plt.plot(T,L)
    plt.xlabel("année")
    plt.ylabel("nombre de lapin")
    plt.title("nombre de lapin généré par la suite de fibonacci")
    plt.show()


```

Les deux fonctions servent à modéliser la suite de Fibonacci. La première applique la méthode bottom-up, réduisant le temps de calcul en déterminant tous les nombres entre 2 et x, en les stockant dans un tableau. Ce tableau est ensuite envoyé à une autre fonction pour afficher les données sous forme de graphe.

```
def fibobase(x,y):
    L= [0 for i in range(x)]
    L[0]=y
    L[1]=y
    for i in range(2, x):
        L[i] = L[i-1] + L[i-2]
    return L



def fibobasetab(x,y):
    L = fibobase(x,y)
    T= [i for i in range(x)]
    print(T)
    plt.plot(T,L)
    plt.xlabel("année")
    plt.ylabel("nombre de lapin")
    plt.title("nombre de lapin généré par la suite de fibonacci")
    plt.show()

```

Ces fonctions sont similaires aux deux précédentes, à la seule différence qu'elles permettent de définir un nombre initial de lapins. La seconde fonction se distingue par l'appel à une fonction différente : elle utilise fibobase(x, y) au lieu de fibobotup(x).

Voici ce que donne fibobasetab(12, 2000) :

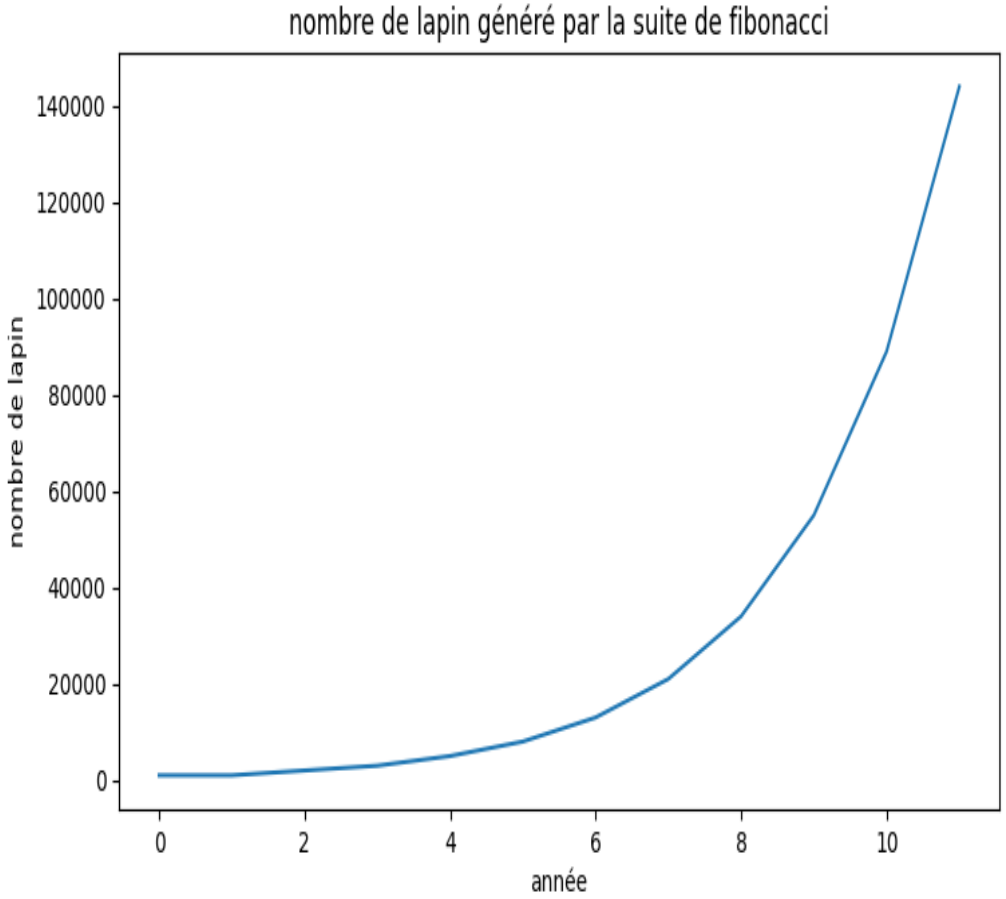


Figure 1 – génération de lapin sur 12 année par la suite de fibonacci

On remarque que, si l'on part de 2000 lapins, on obtient, avec la suite de Fibonacci, plus de 140 000 lapins, selon un retour du code de 144 000. Ce résultat offre une approximation grossière du nombre de lapins au bout de 12 ans. Par ailleurs, on observe que la suite de Fibonacci ressemble à une croissance exponentielle.

Troisième partie

Théorisation

3 hypothèse de début

Pour la réalsation du Tp, j'ai choisit plusieurs conditions au début :

1ère condition : Tout d'abord, étant donné qu'environ 50 pourcent des lapins sont des mâles et 50 pourcent des femelles, nous considérons uniquement les lapins femelles dans le cadre de la simulation. Cette simplification présente un avantage : elle réduit par deux la taille des calculs ainsi que la mémoire nécessaire pour la simulation.

Cependant, cette hypothèse introduit plusieurs problèmes. En effet, elle ne prend pas en compte les éventuelles situations de surpopulation ou de pénurie de mâles dans une population donnée, ce qui peut avoir un impact significatif sur les résultats de la simulation. En ne modélisant pas ces cas, les résultats obtenus peuvent s'éloigner de la réalité. En d'autres termes, cette hypothèse sous-entend que la simulation ne présentera pas de baisses significatives de population, car elle suppose que la population de lapins est toujours équilibrée entre mâles et femelles.

2ème condition : La simulation fonctionne à une échelle annuelle. Cette hypothèse simplifie le calcul du nombre de lapins en considérant leur évolution année par année.

3ème condition : Les décès dans la population sont simulés au début de chaque année. Cette hypothèse permet d'évaluer combien de lapins nés meurent au cours des premiers moments de l'année. Elle offre également une opportunité d'améliorer le modèle en affinant cette étape pour refléter des dynamiques plus réalistes.

Quatrième partie

Mise en oeuvre

4 Implémentation

```
1
2  import java.util.Arrays;
3
4  public class Central{
5      private static rabfemale[] rab;
6      private static int compteur = 0 ;
7      private static int n = 1000000;
8      private static rabfemale[][] tabyy;
9
10     public static void main(String[] argv){
11         int x = 2;
12         System.out.println("début de l'initialisation avec " + x +" lapin");
13         compteur = x;
14         int f = 0;
15         int h = 12;
16         tabyy = new rabfemale[h][n];
17         rab = new rabfemale[n];
18         for(int i =0;i<compteur;i++){
19             rab[i] = new rabfemale() ;
20             rab[i].upAge();
21         }
22         System.out.println("début de la simulation :");
23         while (f<h){
24             System.out.println("année : " + f);
25
26             System.out.println("début de la selection naturelle");
27             killrabbit(compteur);
28
29
30             System.out.println("début de la reproduction");
31             for(int i = 0;i<compteur;i++){
32                 rab[i].genelitter();
33             }
34             System.out.println("fin de la reproduction");
35             System.out.println("veillissement");
36             update(compteur);
37
38             System.out.println("début de l'ajout des nouveau nés");
39             for(int i = 0;i<compteur;i++){
40                 int g = rab[i].getkittens();
41                 rab[i].setkittens(0);
42                 maj(g);
43                 compteur = compteur+g;
44             }
45
46             tabyy[f] = Arrays.copyOf(rab,compteur + 1);
47
48             f=f+1;
49     }
50     affichertab(h,tabyy);
51     }
52
```

La fonction principale est divisée en trois grandes parties :

Initialisation :

On choisit le nombre initial de lapines à simuler, représenté par x. On définit le nombre total d'années de simulation, noté h. On crée deux tableaux : l'un pour représenter l'année en cours et l'autre pour stocker les

données de cette année. On initialise un compteur d'années, f.

Le cœur du programme :

Cette partie simule les différentes actions d'une année. On commence par simuler la mortalité des lapines, correspondant à la fin de l'hiver. Ensuite, on vérifie si les lapines restantes se reproduisent.

Stockage et affichage :

Les nouvelles lapines sont stockées dans le tableau, à la ligne correspondant à l'année en cours. À la fin de la simulation, le tableau complet est affiché pour visualiser les données.

```
1   /*
2     Debut Fonction d'affichage
3    */
4
5   private static void affichertab(int h, rabfemale[][] tabyg){
6       int j = 0;
7       for(int i =0; i<h;i++){
8           while (tabyg[i][j]!=null){
9               j=j+1;
10          }
11          System.out.println("nombre de lapin cette année " + i + " : " + j);
12          j=0;
13      }
14  }
15
16  private static void afficher(rabfemale rab){
17      if(rab!=null){
18          if (rab.isAlive()){
19              System.out.println("Il est vivant");
20          }
21          else{
22              System.out.println("Il est mort");
23          }
24      }
25      else{
26          System.out.println("inexistant");
27      }
28  }
29
30  /*
31    Fin Fonction d'affichage
32   */
33
```

Ce sont des fonctions d'affichage.

La première, affichertab(), sert à afficher les résultats de la simulation année par année. La seconde fonction était principalement utilisée pour le débogage du programme, permettant de vérifier s'il y avait des erreurs ou des anomalies dans le fonctionnement.

```
1
2
3       /*
4        Début des fonction de mise a jour de la population
5       */
6
7
8       private static void majrab(int x,int y){
9           for(int i = y; i < y + x; i++){
10              rab[i] = new rabfemale();
11          }
12      }
13      private static void maj(int x){
14          majrab(x,compteur);
15      }
16
17      private static void killrabbit(int x){
18          for (int i =0; i<x;i++){
19              double r = Math.random();
20              if (rab[i].getrateliv() > r){
21                  rab[i].setAlive(true);
22              }
23              else{
24                  rab[i].setAlive(false);
25              }
26          }
27      }
28
29      private static void update(int x){
30          int i = 0;
31          while (i<x){
32              if(rab[i].isAlive()){
33                  rab[i].upAge();
34                  i=i+1;
35              }
36              else{
37                  rabfree(i);
38                  compteur = compteur - 1;
39                  x=x-1;
40              }
41          }
42      }
43
44      /*
45        Fin des fonction de mise a jour de la population
46       */
```

Ces fonctions permettent de mettre à jour le tableau et les données concernant les lapins :

Ajout des nouveaux lapins :

La première fonction ajoute les nouveaux lapins au tableau existant. La seconde fonction sert principalement à appeler la première pour effectuer cette mise à jour.

Survie des lapines :

La troisième fonction détermine quelles lapines ont survécu à l'année en cours. Cette survie est calculée en fonction d'un taux qui dépend de l'âge de chaque lapine. Mise à jour de l'âge et gestion des décès :

La dernière fonction met à jour l'âge des lapines encore vivantes. Si une lapine est morte, elle appelle des fonctions de libération pour retirer les données correspondantes.

```
1
2       /*
3        Début des fonction de libération de la population
4       */
5
6
7       private static void rabfree(int g){
8           decal_gauche(g);
9       }
10
11      private static void decal_gauche(int g){
12          for(int i =g; i<compteur ; i++){
13              while ( rab[i + 1 ]!=null){
14                  rab[i] = rab[i + 1];
15                  i=i+1;
16              }
17              if(rab[i + 1 ]==null){
18                  rab[i] = null;
19              }
20          }
21      }
22
23      /*
24        Fin des fonction de libération de la population
25       */
26
27  }
28
29
```

La première fonction appelle la seconde. Elle a été créée pour permettre des améliorations futures, notamment en ce qui concerne le décalage à gauche dans le tableau.

La seconde fonction supprime les données des lapines mortes en décalant tous les éléments restants d'une position vers la gauche. Ce choix est motivé par une considération d'optimisation : au lieu de stocker directement la position exacte de chaque lapine dans le tableau, ce qui augmenterait considérablement la consommation de mémoire, j'ai privilégié une approche qui consomme moins de mémoire, même si cela entraîne un léger coût supplémentaire en termes de temps de calcul.

```
1   public class rabfemale{
2       private int litterThisYear;
3       private int kittens = 0;
4       private int age ;
5       private boolean alive;
6       private double rateofliving;
7
8       public rabfemale(){
9           age = 0;
10          alive = true;
11          setrate(age);
12          litterThisYear = 0;
13          kittens = 0;
14      }
```

Voici les attributs de la classe, ainsi que le constructeur.

kittens et litterThisYear sont liés à la naissance des nouveaux lapins au cours de l'année. Age représente l'âge de la lapine. rateofliving correspond au taux de survie, soit la probabilité que la lapine survive durant l'année. alive est un attribut booléen qui indique si la lapine est vivante ou morte.

```
1
2     public void setrate(int age1){
3         if (age < 1){
4             rateofliving = 0.35;
5         }
6         else{
7             if (age < 10){
8                 rateofliving = 0.60;
9             }
10            else{
11                rateofliving = rateofliving - 0.10;
12            }
13        }
14    }
```

setrate() permet de définir les chances de survie de la lapine en fonction de son âge.

```
1
2     public int uniform(int a , int b){
3         int x = (int) Math.round( Math.random() * (b - a)  + a);
4         return x;
5     }
6     public int uniformlitter(int a, int b){
7         int x =  (int)Math.round( (Math.random() * (b - a)  + a ) * 0.5 );
8         return x;
9     }
```

Voici les deux fonctions permettant de générer un nombre aléatoire. Étant donné que nous ne générons que des lapines, nous ne prenons que 50 pourcent des valeurs générées globalement.

```
1
2     private void newlitter(){
3         litterThisYear = litterThisYear + 1;
4         int v = uniformlitter(3,6 );
5         kittens = kittens +  v;
6     }
7
8     public void genelitter(){
9         boolean verif = true;
10        while( (isAlive()) && (verif) ){
11            if (getAge() >= 1){
12                int nblitter = uniform(3,9);
13                for (int i =0; i<nblitter; i++){
14                    newlitter();
15                }
16                verif = false;
17            }
18            else{
19                verif = false;
20            }
21        }
22    }
```

Voici les fonctions pour générer les différents accouchements. Je les ai séparées en deux fonctions : l'une qui génère les accouchements eux-mêmes et l'autre qui génère les petites lapines nées lors de l'accouchement. Les lapines ne peuvent accoucher qu'à partir de leur majorité sexuelle, atteinte entre 6 et 8 mois. Cependant, dans notre modèle, elles peuvent se reproduire au bout d'un an.

```
    public void upAge(){
        age=age+1;
    }
```

Ceci la fonction qui permet de vieillir les lapines.

```
    public int getLitterTY(){
        return litterThisYear;
    }
    public int getkittens(){
        return kittens;
    }
    public int getAge(){
        return age;
    }
    public boolean isAlive(){
        return alive;
    }
    public double getrateliv(){
        return rateofliving;
    }
```

Voici les getters. Ils nous permettent d'obtenir les valeurs des différents attributs de RabbitFemale sans les exposer directement dans d'autres fonctions.

```

    public void setkittens(int i){
        kittens = i;
    }
    public void setAlive(boolean a){
        alive = a;
    }


}
```

Ce sont les fonctions setters. On en as besoin de deux : l'une pour modifier le nombre de petites lapines, et l'autre pour indiquer si le lapine est mort ou vivant.

Cinquième partie

résultat

5 résultat

J'ai choisi de modéliser avec 1000 lapines, car avec un nombre faible de lapines (¡ 100), il est possible qu'elles meurent rapidement et fréquemment, ce qui pourrait entraîner la fin de la simulation dès les premières années. De plus, 1000 lapines semblait être un bon échantillon pour tester les améliorations, car les résultats variaient beaucoup plus.

Voici les résultats obtenue :

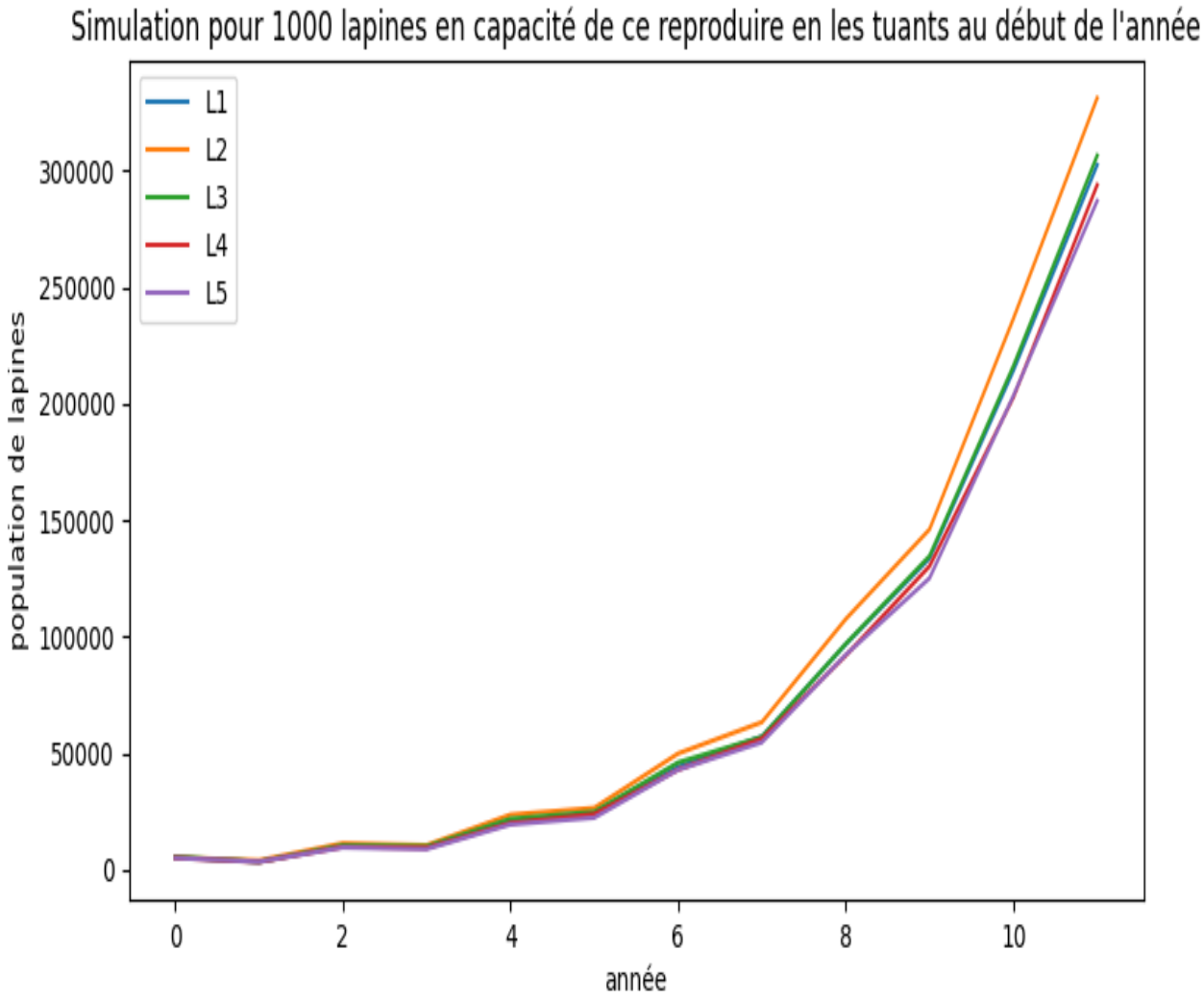


Figure 2 – nombre de lapin par année sur plusieurs simulation

Voici les différents tableau dont sont issue les données :
$L1 = [5321, 3524, 10289, 9654, 20551, 23835, 44165, 57117, 96703, 133602, 214092, 302384]$
$L2 = [5323, 4014, 11365, 10561, 23622, 26528, 49864, 63317, 107522, 146032, 236365, 331049]$
$L3 = [5505, 3485, 10455, 10013, 21832, 24872, 46010, 56986, 96911, 134592, 216080, 306359]$
$L4 = [5156, 3397, 9824, 9372, 20052, 23892, 42972, 56208, 91882, 130219, 202626, 293747]$
$L5 = [5078, 3486, 9585, 8914, 19465, 22371, 42910, 54709, 92216, 125094, 203282, 286929]$

Ces tableaux correspondent aux données issues des différentes simulations du modèle. Comme on peut le voir, nous avons à la fin de la première année un grand nombre de nouveaux-nés, puis beaucoup d'entre eux meurent au début de l'année 1, ce qui explique la chute de population par rapport à l'année 0 et à l'année 1. Ce schéma se répète jusqu'à l'année 4, avec une diminution de son impact sur la population. Après l'année 4, le nombre de nouveaux-nés est beaucoup plus important que le nombre de lapins morts, ce qui permet aux lapins de se reproduire en masse. Ce n'est qu'à partir de l'année 7 que cette chute de population n'impacte plus la croissance des lapins. Avant l'année 7, cette chute de population diminuait la croissance, mais après l'année 7, cette chute de population n'impacte plus la croissance de la population.

Du fait du niveau initial de population, cet impact se fait moins ressentir sur la durée. Il est plus présent pour les plus petits taux de départ, comme avec 100 ou 1 lapines.

6 point d'amélioration possible

Certaines de nos conditions peuvent être modifiés afin de mieux représenter la réalité. La première condition peut laisser à désirer, car elle limite la simulation en ne prenant en compte que les lapines femelles.

La deuxième condition peut également poser problème, car elle simplifie excessivement la simulation en termes de mois. En raison de cette condition, nous ne prenons pas en compte la période de gestation des lapines, les saisons de reproduction et les variations saisonnières. Cela affecte particulièrement le taux de mortalité, car les lapins font face aux mêmes risques de mort, qu'il s'agisse de l'été ou de l'hiver. Or, ces saisons peuvent présenter des climats très différents, et si l'on les considère, on pourrait observer davantage de morts en hiver qu'en été, ce qui pourrait potentiellement réduire le nombre de morts par an.

La troisième condition, celle concernant la mortalité de la population de lapins, est relativement simplifiée. En effet, nous ne prenons pas en compte toutes les causes de mortalité possibles au cours de l'année, comme le stress lié à l'accouchement ou les risques de mort pendant celui-ci dus à des causes externes.

Nous pouvons améliorer la simulation en modifiant trois points :

Prendre en compte une population inégale de mâles et de femelles : Cela pourrait mener à un taux de mortalité plus élevé ou à une croissance plus lente des lapins.

Complexifier la simulation en la simulant mois par mois ou saison par saison : En ajustant le taux de mortalité en fonction du mois ou de la saison, nous obtiendrions une simulation plus précise. Cependant, cela augmenterait considérablement les calculs, multipliant la complexité par 4 (saison) ou 12 (mois).

Affiner les conditions de mortalité des lapins : En tenant compte des morts entre les différents accouchements et à la fin de l'année, nous pourrions affiner la simulation, notamment dans les environnements non affectés par l'activité humaine.

Nous avons choisi la dernière option d'amélioration pour affiner le modèle. Néanmoins, nous considérons que la mortalité en début d'année est similaire à celle en fin d'année et correspond à la période hivernale.

7 résultat avec amélioration

Nous avons choisi de modéliser la mortalité après l'accouchement avec un taux de 0,9 . Cela est dû à la rareté d'une lapine mourant après son accouchement. Les causes de cette mortalité peuvent inclure le stress, des complications pendant l'accouchement, ou d'autres problèmes de santé. Ce taux de survie pourrait être plus faible si la lapine n'a pas eu beaucoup d'accouchements au cours des années précédentes. Pour affiner ce taux de survie, nous pourrions introduire un facteur aléatoire pour déterminer si la lapine a un accouchement ou non.

Cependant, j'ai effectué d'autres tests en modifiant le taux de mortalité, comme en le réduisant à 0,6 . Ce test a conduit à une baisse drastique du nombre de lapines et ne permettait pas de voir leur population survivre. Ce scénario pourrait modéliser un habitat où la population de prédateurs est trop élevée.

```
    private void killrabbit2(){
          double r = Math.random();
            if (0.9 > r){
                   setAlive(true);
            }
            else{
                setAlive(false);
            }

    }


```

Cette fonction sert à modéliser la mortalité des lapines pendant l'accouchement ou en raison des conséquences de celui-ci. En raison de la rareté de ces décès, nous comparons le nombre généré avec un seuil de 0,9 pour déterminer si la lapine meurt.

```


public void genelitter(){
        boolean verif = true;
        while( (isAlive()) && (verif) ){
            if (getAge() >= 1){
                int nblitter = uniform(3,9);
                for (int i =0; i<nblitter; i++){
                    newlitter();
                    killrabbit2();

                }
                verif = false;
            }
            else{
                verif = false;
            }
        }
    }
```

J'ai ajouté la fonction killrabbit2() dans la fonction genelitter() afin de calculer le nombre de lapins morts lors de l'accouchement.

voici le résultat :

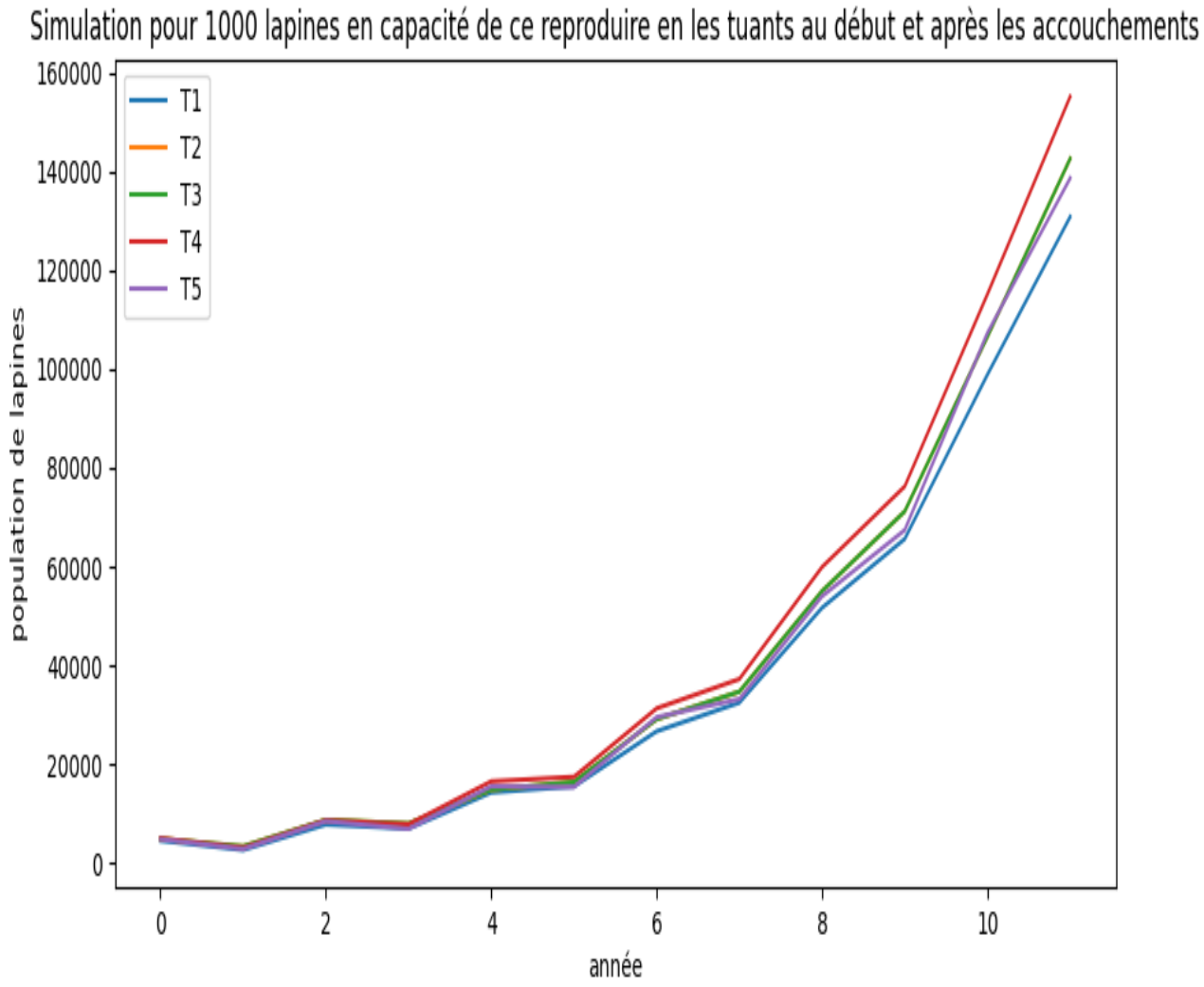


Figure 3 – nombre de lapin par année sur plusieurs simulation avec afinnement du modèle

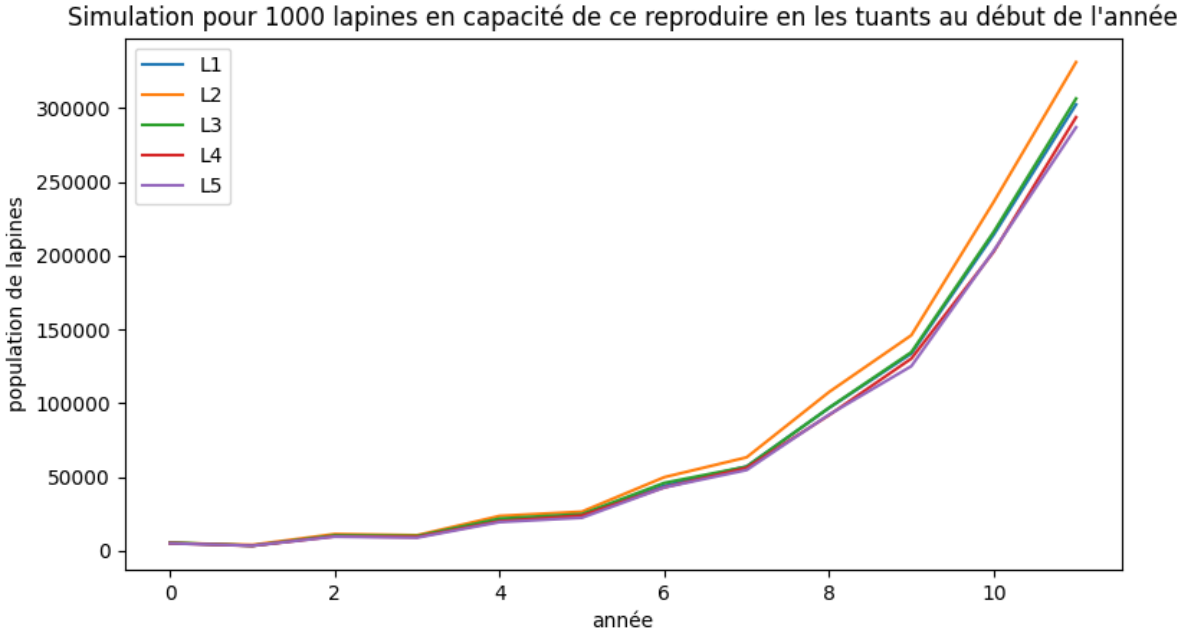
voici les tableaux de valeurs correspondant aux courbes :
$T1 = [4514, 2635, 7787, 6956, 14273, 15490, 26650, 32458, 51654, 65593, 98875, 130814]$
$T2 = [4891, 3341, 8689, 7956, 14840, 16397, 29097, 34673, 55032, 71175, 106567, 142583]$
$T3 = [4891, 3341, 8689, 7956, 14840, 16397, 29097, 34673, 55032, 71175, 106567, 142583]$
$T4 = [4992, 3058, 8613, 7802, 16509, 17427, 31293, 37224, 59905, 76235, 115166, 155149]$
$T5 = [4800, 2843, 8454, 6910, 15501, 15374, 29510, 33177, 53986, 67399, 107321, 138636]$

Nous avons le même phénomène qui ce produit comme au résultat précédent. Néanmoins le principale changement est le nombre de lapines qui est beaucoup plus faible que dans le résultat sans affinement. Ceci est du au faite que si des lapines sont tués lors de l'accouchement, elle ne peuvent plus se reproduit et donc cela a un effet papillon sur la population.

8 mise en rapport des résultats de la simulation

rappel des deux figures afin de les comparer :

[b]0.45



[b]0.45

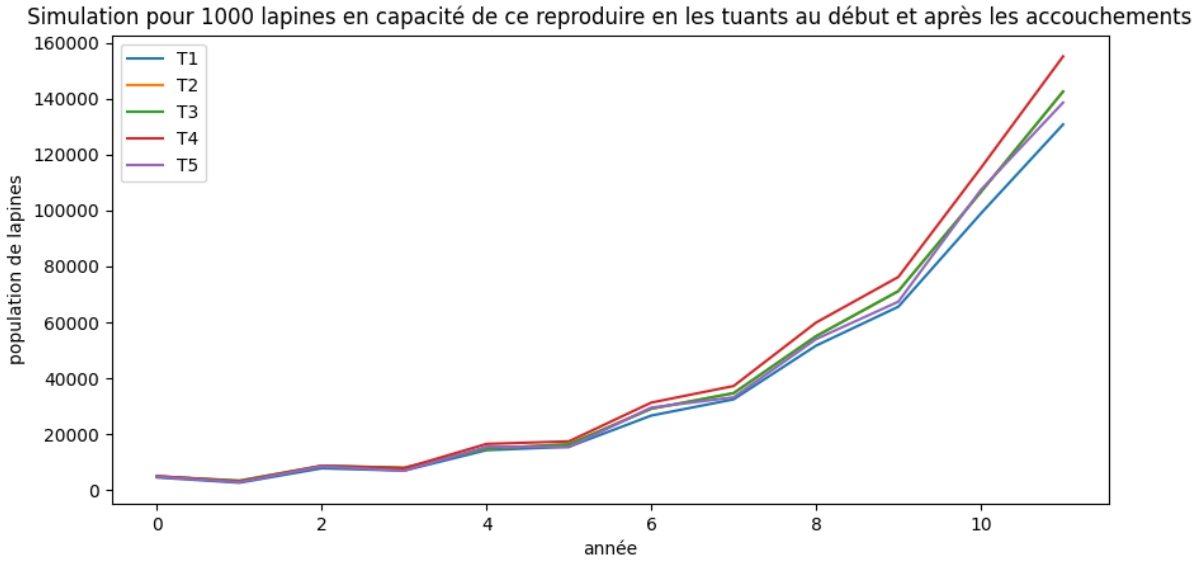


Figure 4 – Comparaison des deux résultats

On remarque qu'à part la différence de population, nous obtenons des courbes similaires. En effet, la modification au niveau des morts est moins impactante pour des nombres de départ de lapins assez élevés. Cette modification entraîne une variation plus importante des résultats, avec un nombre de lapins adultes plus faible lors de l'année 0.

Sixième partie

Conclusion

9 Conclusion

Lors de ce TP, nous avons utilisé plusieurs méthodes afin de simuler la population de lapins. Nous avons utilisé la suite de Fibonacci, puis plusieurs modèles plus précis afin de simuler la population de lapins sur plusieurs années. Néanmoins, mes modèles créés ont encore certains axes d'amélioration qui sont possibles à étudier afin d'améliorer la simulation et d'obtenir un nombre de lapins plus proche de la réalité.

Néanmoins, même si on améliorait la simulation, nous serions limités dans l'amélioration du modèle car nous ne pouvons pas tout prédire, comme les maladies ou d'autres causes externes qui peuvent apparaître et grandement affecter la population, ce qui bloquerait à un certain seuil la fiabilité par rapport à la réalité de la simulation.

10 Crédit

J'ai utilisé ChatGPT pour la correction des fautes d'orthographe dans les phrases.
J'ai utilisé Python pour le code d'affichage et Java pour le code de la simulation.

Table des figures