Compte Rendu TP4
Simulation d'une population de lapins

Jérémy ZANGLA

18 novembre 2019

Table des matières

1 Analyse

1.1 Rappel du sujet

Le but de ce TP est de simuler l'évolution d'une population de lapins au cours du temps. La population ne sera affectée par aucun élément extérieur. Les seuls facteurs influant l'évolution seront les naissances et morts naturelles. Ainsi, les données que nous implémenterons seront les suivantes :

— Un lapin devient adulte entre 5 et 8 mois après sa naissance.
— Une lapine doit attendre au moins un mois entre deux portées (temps de gestation).
— Une lapine aura de 4 à 8 portées par an.
— Un laperau aura 20% de chance de survivre avant sa maturité.
— Un lapin aura 50% de chance de survie chaque année jusqu'à ses 11 ans.
— Pour chaque année suivante, les lapins auront 10% de chance de mourir supplémentaires.
— Ainsi aucun lapin ne pourra vivre plus de 15 ans.

Enfin le travail devra être fait en programmation objet dans le langage C++.

1.2 Choix d'implémentation

1.2.1 Génération de nombres aléatoires

Comme pour les travaux précédents, nous allons utiliser l'algorithme de Mersenne-Twister pour la génération des nombres pseudos aléatoires. Nous utiliserons son implémentation dans la libraire standard du C++ (bibliothèque : random).
Cependant les nombres pseudo-aléatoires ne seront pas utilisés directement, contrairement aux travaux en C. Nous utiliserons les distributions de libraire standard. Nous parlerons par la suite de nombres aléatoires pour alléger le discours, il faut bien garder en tête que tous les nombres seront générés à partir de ce générateur pseudo-aléatoire.

1.2.2 Définition du temps

Nous ne pouvions pas ici faire une simulation en temps réel de l'évolution de la population, pour une raison simple : le but est d'étudier des résultats obtenus après plusieurs mois ou années. Nous avons donc décidé d'accélérer le temps en intégrant un intervalle de temps. Cet intervalle correspond à la durée simulée entre deux mises à jour du code. Cet intervalle a été choisi à 1 semaine. Nous obtenons donc une simulation "précise" à la semaine près. Nous avons également décidé de simplifier une année pour qu'elle se compose de 48 semaines seulement.

1.2.3 Fonctionnement des mises à jour

Au lieu de faire fonctionner notre simulation au fur et à mesure de son évolution, le choix a été fait de déterminer toute la vie d'un lapin dès sa naissance. Ainsi à la naissance d'un lapin, la semaine à laquelle il mourra sera décidée. De plus, pour une femelle, la date ainsi que les spécificités de chaque portée qu'elle aura seront fixées au même moment. Il y aura tout de même une unique vérification en "temps réel", c'est la présence ou non d'un lapin mâle pour la reproduction.

1.2.4 Déroulement des mises à jour et lancement d'une simulation

Chaque mise à jour se fera de la manière suivante, traitement de la semaine actuelle : naissances puis morts des lapins. On passera ensuite à la semaine suivante. Une fois la simulation créée, on pourra la lancer pour une durée choisie et la reprendre autant de fois que désiré.

1.2.5 Performances

Le choix de la déterminisation de la vie d'un lapin à sa naissance, devrait permettre d'améliorer les performances du simulateur en simplifant les calculs, et en évitant d'en faire certains. L'espace mémoire occupé par la simulation peut être un point sensible, cependant nous avons décidé ici de ne pas s'en occuper.

2 Implémentation

2.1 Nombres aléatoires

2.1.1 Distributions de nombres aléatoires

Une distribution est un objet implémenté dans la librairie standard (bibliothèque random), qui permet d'obtenir un nombre aléatoire qui suit une loi de probabilité. Nous en utiliserons ici 4 types en expliquant leur intérêt. Nous utilisons l'implémentation standard du générateur de Mersenne-Twister pour la génération de nombres pseudo-aléatoires.

RabbitSimulation.hpp

```
1       // Générateur de Mersenne-Twister de cette simulation.
2       std::mt19937 _generator;
```

2.1.2 Distribution uniforme

La distribution uniforme est la plus simple, elle nous permet d'obtenir un nombre aléatoire compris dans un intervalle suivant une loi de probabilité uniforme, c'est à dire que chaque tirage est équiprobable. Il existe deux versions de cette distribution : réelle et entière ; comme on pourrait le supposer la première permet la génération d'un nombre réel et la seconde d'un nombre entier.

La première utilisée permet de choisir le sexe d'un lapin à naître lors de la génération des portées d'une lapine. C'est donc une distribution uniforme de nombres entiers avec deux possibilités : 1 et 2. Ces possibilités correspondent respectivement à Mâle et Femelle.

RabbitFemale.hpp

```
1       // Distribution générant le sexe de chaque laperau
2       static std::uniform_int_distribution<short> dist_gender;
```

RabbitFemale.cpp

```
1  std::uniform_int_distribution<short> RabbitFemale::dist_gender(1, 2);
```

La seconde est une distribution de nombres réels dans l'intervalle [0 ; 1]. Le nombre obtenu est très facilement comparable à un pourcentage. On l'utilise pour décider à quel période de sa vie le lapin va mourir (avant la maturité ou non, puis avant 11 ans ou non).

Rabbit.hpp

```
1       // Distribution permettant de faire un nombre aléatoire
2       // de type pourcentage
3       static std::uniform_real_distribution<> dist_rand;
```

Rabbit.cpp

```
1  std::uniform_real_distribution<double> Rabbit::dist_rand(0, 1);
```

La dernière sert à calculer la date de maturité d'un laperau, donc entre 4 et 8 mois (converti en semaine).

Rabbit.hpp

```
1       // Constante du nombre de semaines par an.
2       static constexpr unsigned short WEEK_PER_YEAR = 48;
3       // Constante de la durée de vie maximum d'un lapin en semaine
4       static constexpr unsigned int MAX_LIFESPAN = 15 * WEEK_PER_YEAR;
5       // Age en semaine à partir duquel un lapin peut devenir mature
6       static constexpr unsigned int MATURITY_MIN = 5 * 4;
7       // Age en semaine jusqu'auquel un lapin peut devenir mature
8       static constexpr unsigned int MATURITY_MAX = 8 * 4;
```

Rabbit.hpp

```
1       // Distribution générant la date de maturité du lapin
2       static std::uniform_int_distribution<> dist_maturity;
```

Rabbit.cpp

```
1  std::uniform_int_distribution<>
2       Rabbit::dist_maturity(MATURITY_MIN, MATURITY_MAX);
```

2.1.3 Distribution linéaire

Une distribution linéaire, permet d'avoir une densité de probabilité qui suit une fonction linéaire (ou un ensemble de fonctions linéaires). Nous n'en utilisons qu'une seule, qui n'utilise qu'une seule pente. L'implémentation dans la librairie standard utilise la méthode de Piecewise et a besoin de 2 tableaux pour le paramétrage. Le premier contient les intervalles des pentes, et le second contient les poids de chaque points, pour définir l'inclinaison de la pente.

Nous avons décidé d'utiliser cette distribution pour simuler l'accroissement de la mortalité à partir de 11 ans. Cette accroissement étant de 10% par an, il est linéaire.

Rabbit.hpp

```
1       // Variable d'initialisation d'une distribution
2       // linéaire de piecewise
3       static std::array<double,3> intervals;
4       static std::array<double,3> weights;
```

Rabbit.hpp

```
1       // Distribution générant la date de mort d'un papy lapin
2       static std::piecewise_linear_distribution<double> dist_death_end;
```

Rabbit.cpp

```
1  std::array<double, 3> Rabbit::intervals
2       {14 * WEEK_PER_YEAR, 14 * WEEK_PER_YEAR, MAX_LIFESPAN};
3  std::array<double, 3> Rabbit::weights {0, 0, 1};
```

Rabbit.cpp

```
1  std::piecewise_linear_distribution<double> Rabbit::dist_death_end
2       (Rabbit::intervals.begin(),
3       Rabbit::intervals.end(),
4       Rabbit::weights.begin());
```

2.1.4 Distribution géométrique

Cette loi représente le nombre d'essais à faire avant d'obtenir un succés. Le succés ici est la mort du lapin. En paramétrant correctement cette loi, on peut simuler une probabilité de mort qui réduit lorsque l'on devient plus vieux. On l'utilise pour choisir la date de la mort d'un laperau ou d'un lapin avant 11 ans. Nous aurions pu utiliser une loi uniforme mais la simulation aurait été moins réaliste.

Rabbit.hpp

```
1       // Distribution générant la date de mort d'un lapin adulte
2       static std::geometric_distribution<int> dist_maturity_death;
3       // Distribution générant la date de mort d'un laperau
4       static std::geometric_distribution<int> dist_baby_death;
```

Rabbit.cpp

```
1  std::geometric_distribution<int> Rabbit::dist_baby_death(0.1);
2  std::geometric_distribution<int> Rabbit::dist_maturity_death(0.003);
```

2.1.5 Distribution normale

La loi normale est utilisée pour le choix de la date de la prochaine portée d'une lapine et également pour le nombre de laperaux de cette portée. En centrant correctement la loi, on rend la distribution parfaite pour ces situations, car la distribution est symétrique.

RabbitFemale.hpp

```
1      // Distribution générant la date de la prochaine portée
2      static std::normal_distribution<> dist_birth;
3      // Distribution générant le nombre de laperaux de la prochaine portée
4      static std::normal_distribution<> dist_nb_litters;
```

RabbitFemale.cpp

```
1  std::normal_distribution<> RabbitFemale::dist_birth(9, 4);
2  std::normal_distribution<> RabbitFemale::dist_nb_litters(6.5, 2);
```

2.2 Algorithmes d'évolution de la population

Pour rappel, nous utilisons pour nos algorithmes une vision déterministe de la vie d'un lapin. Ainsi nous calculons toutes les informations dès la naissance du lapin.

2.2.1 Algortithme de choix des naissances

Pour les naissances, il nous faut déterminer plusieurs choses :
— la date de naissance de chaque portée.
— le nombre d'enfants de chaque portée.
— le sexe de chacun des enfants.
Nous avons rassemblé tous ces éléments dans une structure qui n'est accessible que depuis le RabbitFemale.

RabbitFemale.hpp

```
1      // Structure d'une portée
2      struct litter{
3          // Semaine de naissance de la portée
4          unsigned int week;
5          // Nombre de mâles de la portée
6          short males;
7          // Nombre de femelles de la portée
8          short females;
9      };
```

Pour stocker les portées nous avons décidé d'utiliser une structure de type FIFO (First In First Out). Cette structure permet de trier les naissances par ordre chronologique, et donc évite de parcourir l'ensemble des portées chaque semaine pour faire les naissances.

RabbitFemale.hpp

```
1      // File contenant les portées que la mère aura
2      std::queue<litter*> litters;
```

Enfin pour la génération des portées nous suivons l'algorithme suivant. Le passage de portée en portée, nous permet de ne pas faire de tours de boucles inutiles où il ne pourrait rien être fait. On évite donc aussi de faire plusieurs tests par tour de boucle (attente minimum de 1 mois entre 2 portées).

Le choix de la date de la prochaine portée se fait grâce à une distribution aléatoire. La création des détails sur la portée génère tout d'abord le nombre de laperaux puis le sexe de chacun.

```
pour chaque semaine entre la maturité et la mort faire
    Choix de la date de la prochaine portée
    Création des détails sur cette portée
    Passage direct à la date de cette portée pour calculer les suivantes
fin
```

RabbitFemale.cpp

```cpp
void RabbitFemale::init_birth()
{
    // Pour chaque semaine entre la maturitée et la mort de la belle lapine
    for (unsigned int i = _maturity_week; i <= _death_week;)
    {
        // Génération de la date de la prochaine portée
        short w;
        do
        {
            w = std::round(dist_birth(generator));
        } while (w < 0 || w > 18);
        // On passe directement à la date de cette portée
        // et on ajoute le temps de gestation
        i += w;
        // Si la lapine n'est pas morte avant
        if (i <= _death_week)
        {
            // Création de la portée
            litter * l = new litter;
            l->males = 0;
            l->females = 0;
            l->week = i + _week_offset;
            // Génération des laperaux
            short nb_baby;
            do
            {
                nb_baby = dist_nb_litters(generator);
            } while (nb_baby < 4 || nb_baby > 8);
            for (short i = 0; i < nb_baby; i++)
                // Choix de leur sexe
                if (dist_gender(generator) == 1)
                    (l->males)++;
                else
                    (l->females)++;
            // Ajout de la nouvelle portée à la liste
            litters.push(l);
        }
    }
}
```

2.2.2 Algorithme de naissance des lapins

Pour la naissance des lapins, il ne reste plus qu'à parcourir la liste des lapins femelles en vie. Avant de parcourir la liste, on vérifie qu'il y a bien des mâles encore en vie. On fait ici une approximation, il aurait fallu vérifier si un mâle était présent 4 semaines avant (pour la fécondation). Sur chacun de ces lapins, on vérifiera si il doit mettre des lapins au monde. Si c'est le cas, on ajoutera les lapins en suivant les caractéristiques de la portée.

RabbitFemale.cpp

```cpp
void RabbitFemale::give_birth(unsigned int week, std::list<Rabbit*>& males,
    std::list<RabbitFemale*>& females)
{
    // Si la lapine attend encore des portées
    // et que la prochaine tombe cette semaine
    if (!litters.empty() && litters.front()->week == week)
```

```
7       {
8           // Naissance des mâles
9           for (short i = 0; i < litters.front()->males; i++)
10              males.push_back(new Rabbit(generator, _death_histo,
11              _death_period_histo, _maturity_histo, week));
12          // Naissance des femelles
13          for (short i = 0; i < litters.front()->females; i++)
14          {
15              RabbitFemale * rabbit = new RabbitFemale(generator, _death_histo,
16               _death_period_histo, _maturity_histo,
17               _gender_histo, _baby_histo, _delay_histo, week);
18              females.push_back(rabbit);
19          }
20
21          // Suppression de la portée de la file
22          delete litters.front();
23          litters.pop();
24      }
25  }
```

2.2.3 Algorithme de maturité

Un lapin atteint sa maturité entre 5 et 8 mois après sa naissance. On utilise donc simplement une distribution uniforme entre ces deux bornes.

2.2.4 Algorithme de gestion des morts

Le choix de la date de la mort d'un lapin se fait en 2 étapes. La première décide à quelle période de sa vie le lapin va mourir, cette décision déterminera quelle distribution sera utilisée pour choisir la date de la mort. La seconde étape est donc de choisir la date de la mort. On applique la distribution correspondant à la période de la vie, et on vérifie que le résultat est valide.

Rabbit.cpp

```
1   unsigned int Rabbit::init_death()
2   {
3       // Chances de mourir avant la maturité
4       constexpr double _mortality_before_maturity = 0.8;
5       // Chances de mourir entre la maturité et 11 ans
6       constexpr double _mortality_before_increments = 1-0.000976562;
7
8       // Date de la mort
9       unsigned int death_week = 0;
10
11      // dist_rand nous donne un nombre aléatoire entre 0 et 1 ainsi si il est
12      // inférieur à 0.8, le lapins va mourir avant la maturité (80%)
13      if (dist_rand(generator) < _mortality_before_maturity)
14      {
15          // Statistiques
16          ++_death_period_histo[1];
17          // On génère une date de mort suivant une loi géométrique.
18          // Comme la loi géométrique n'est pas bornée, nous répétons le tirage
19          // jusqu'à obtention d'une valeur valide
20          do
21          {
22              death_week = dist_baby_death(generator);
23          }
24          while(death_week > _maturity_week);
25      }
26      // Même principe que précédemment mais avec 50%
27      else if (dist_rand(generator) < _mortality_before_increments)
28      {   // Statistiques
29          ++_death_period_histo[2];
30          // Même principe que précédemment une autre loi
31          do
```

```
32          {
33              // La nombre donné par la distribution appartient à [0; +inf[,
34              // On ajoute la date de maturité pour bien placer la date de la mort
35              death_week = dist_maturity_death(generator) + _maturity_week;
36          // On répète tant que l'on n'a pas une date avant les 11 ans
37          } while (death_week > 14 * WEEK_PER_YEAR);
38      }
39      // Si le lapin n'est pas mort avant,
40      // on le tue suivant une loi linéaire entre ses 11 et 15 ans
41      else
42      {
43          // Statistiques
44          ++_death_period_histo[3];
45          death_week = dist_death_end(generator);
46      }
47
48      // On renvoie la date de sa mort
49      return death_week;
50  }
```

2.3 Fonctionnement de la simulation

A chaque exécution du programme, les fichiers de sorties sont supprimés. Et les informations nécessaires à l'étude des résultats sont stockées. La création d'une simulation, initialise le générateur de Mersenne-Twister avec la valeur souhaitée. Nous pouvons également choisir le nombre de lapins mâles et femelles présents au départ. Ces lapins viendront de naître. On peut ensuite lancer la simulation pour une durée fixée en semaine. La simulation peut être prolongée plus tard, en relançant la fonction "run". Ici nous lançons 5 fois la simulation pour une durée de 300 semaines.

main.cpp

```
1  int main(int, char**)
2  {
3      std::cout << "Début des Simulations" << std::endl;
4      RabbitSimulation simu1(50, 1000, 1000);
5      RabbitSimulation simu2(100, 1000, 1000);
6      RabbitSimulation simu3(185, 1000, 1000);
7      RabbitSimulation simu4(5, 1000, 1000);
8      RabbitSimulation simu5(20, 1000, 1000);
9
10     simu1.run(300);
11     simu2.run(300);
12     simu3.run(300);
13     simu4.run(300);
14     simu5.run(300);
15
16     std::cout <<std::endl << "Fin des Simulations" << std::endl;
17     return 0;
18 }
```

2.3.1 Données exportées

Nous exportons une multitude d'informations dans des fichiers différents. Chaque résultat de simulation sera dans un dossier séparé : "../Rapport/x/" où x est le numéro de la simulation.

Variable	Nom du fichier	Informations contenues
_file1	total_rabbits.rab	Nombre de lapins en vie à la semaine x (toutes les 16 semaines)
_file2	total_females.rab	Nombre de femelles en vie à la semaine x (toutes les 16 semaines)
_file3	total_males.rab	Nombre de mâles en vie à la semaine x (toutes les 16 semaines)
_file4	nb_litters.rab	Temps entre chaque portée de chaque lapine
_file5	nb_baby_litters.rab	Nombre de laperaux par portée
_file6	gender_baby.rab	Nombre de laperaux de chaque sexe
_file7	death_dates.rab	Date de mort des lapins
_file8	death_periodes.rab	Période de mort du lapin
_file9	maturity.rab	Date à laquelle le lapin deviendra adulte
_file10	nb_litters_norm.rab	Normalisation du "_file4"

2.4 Compilation et exécution

Pour compiler et exécuter le programme, il faut garder l'organisation des fichiers sinon il y a un risque que les fichiers de résultats ne soient pas créés. Pour compiler, il faut utiliser la commande "make". Puis il faut lancer le programme avec "./out".

3 Analyse des résultats

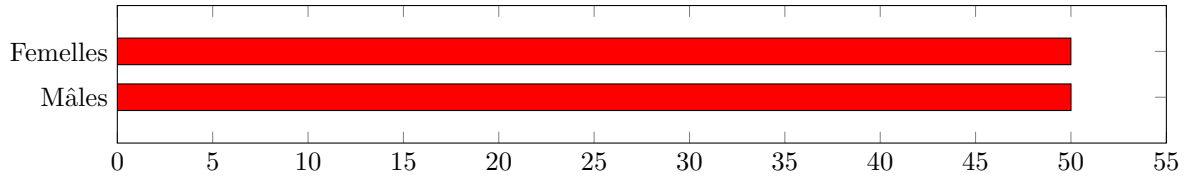
Chaque résultat de simulation a été obtenu à partir d'une initialisation de Mersenne-Twister, ils sont donc reproductibles.

3.1 Naissances

3.1.1 Sexe des laperaux

Le programme donne en sortie le fichier "gender_baby.rab" qui nous donne le pourcentage de mâles et de femelles. Toutes les simulations lancées donnent des résultats similaires, on a calculé l'intervalle de confiance en utilisant une approximation du nombre de lapins en fin de simulation (4 millions).

Répartition des sexes chez les laperaux de la simulation 1



Répartition des sexes chez les laperaux au cours de plusieurs simulations

Simulation	Ratio de mâles	Ratio de femelles
1	50,002 9	49,997 1
2	49,987 8	50,012 2
3	49,989 8	50,010 2
4	50,016 8	49,983 2
5	49,991 3	50,008 7
Moyenne	49,997 72	50,002 28
Intervalle de confiance à 95%	$\pm 0,02498$	± 0.024500

On peut conclure que la répartition est très bonne, le sexe des lapins est bel et bien uniformément réparti, comme demandé dans l'énoncé.

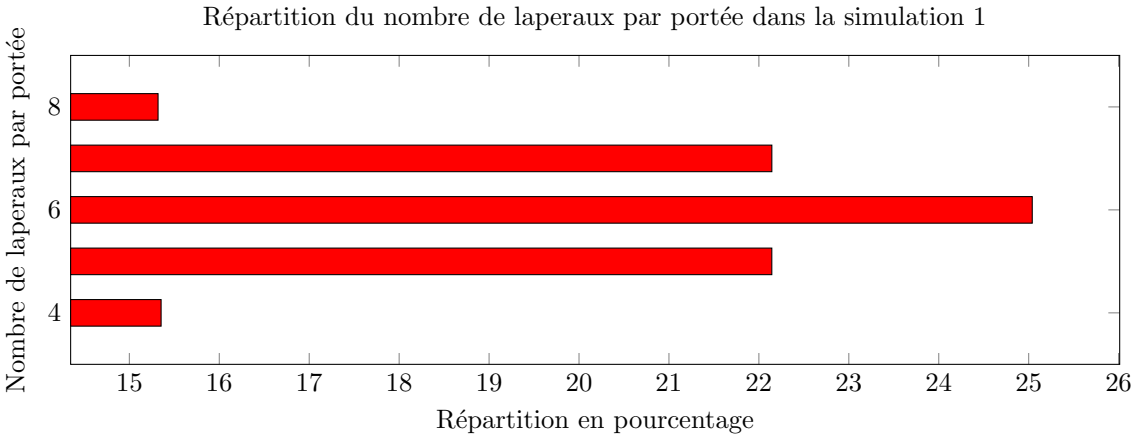
3.1.2 Nombre de laperaux dans une portée

Le programme donne en sortie le fichier "nb_baby_litters.rab" qui nous donne la répartition du nombre de laperaux dans les portées. Toutes les simulations lancées donnent des résultats similaires, on a calculé l'intervalle de confiance en utilisant une approximation du nombre de lapins en fin de simulation (4 millions). Nous avons ici un peu triché dans le code. En effet, nous prenons un nombre aléatoire tant que l'on n'est pas dans les bornes fixées dans l'énoncé.

Répartition du nombre de laperaux dans les portées au cours de plusieurs simulations

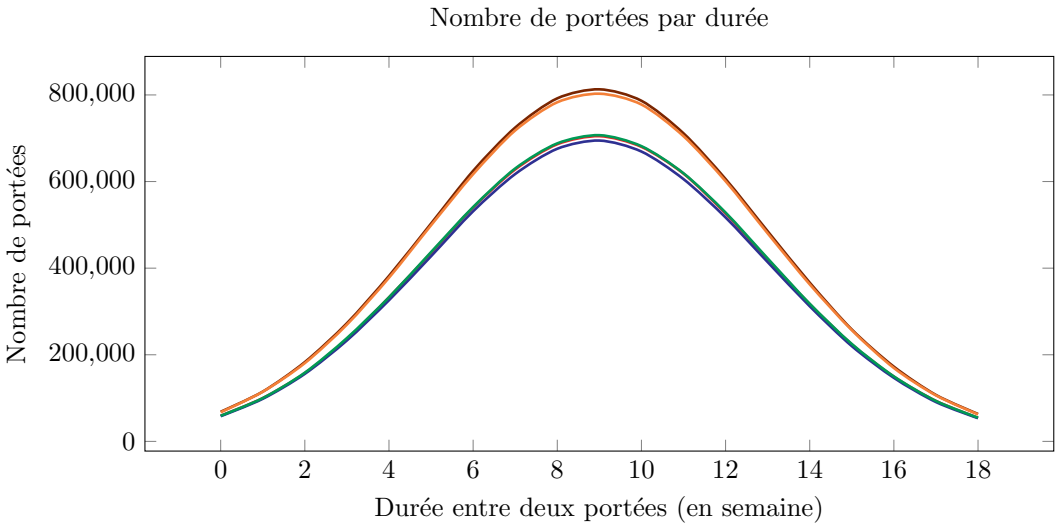
	Proportion du nombre de laperaux par portée				
Simulation	4	5	6	7	8
1	15,328 3	22,174 0	24,976 5	22,196 6	15,324 5
2	15,336 7	22,213 6	24,971 0	22,131 9	15,346 9
3	15,365 7	22,171 3	24,958 4	22,105 0	15,399 6
4	15,349 2	22,087 1	25,077 2	22,134 0	15,352 5
5	15,313 2	22,168 3	25,036 9	22,175 1	15,306 5
Moyenne	15,338 6	22,162 8	25,004 0	22,148 5	15,346 0
Intervalle de confiance à 95%	$\pm 1.10^{-5}$	$\pm 2.10^{-5}$	$\pm 2.10^{-5}$	$\pm 2.10^{-5}$	$\pm 1.10^{-5}$

La répartition du nombre de laperaux est parfaite, elle est symétrique et centrée sur 6 laperaux. Les échantillons où le nombre de lapereaux est aux extrémités (4 et 8) sont moins nombreux que pour les valeurs plus centrées sur la moyenne.

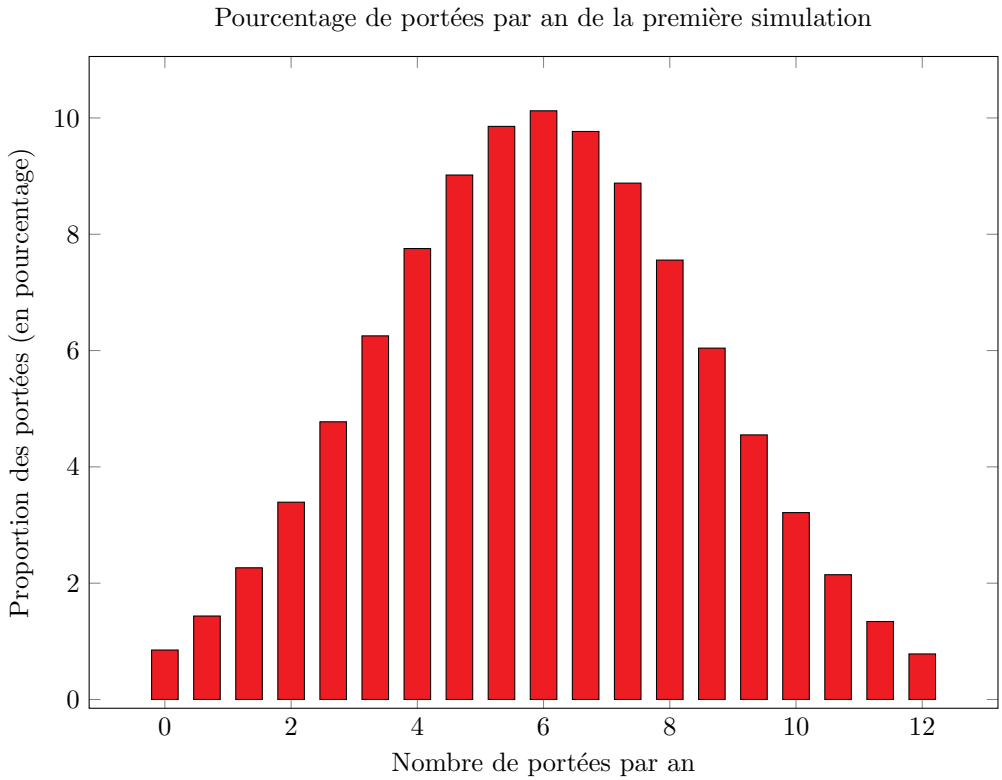


3.1.3 Nombre de portées par an d'une lapine

Le nombre de portées est borné dans le code pour ne pas être en dessous de une portée par an et pas au dessus de une par mois.¡ Le fichier contient le temps entre 2 portées qu'une lapine aura. Il faut donc faire quelques transformations pour vérifier la répartition de 4 à 8 portées par an.



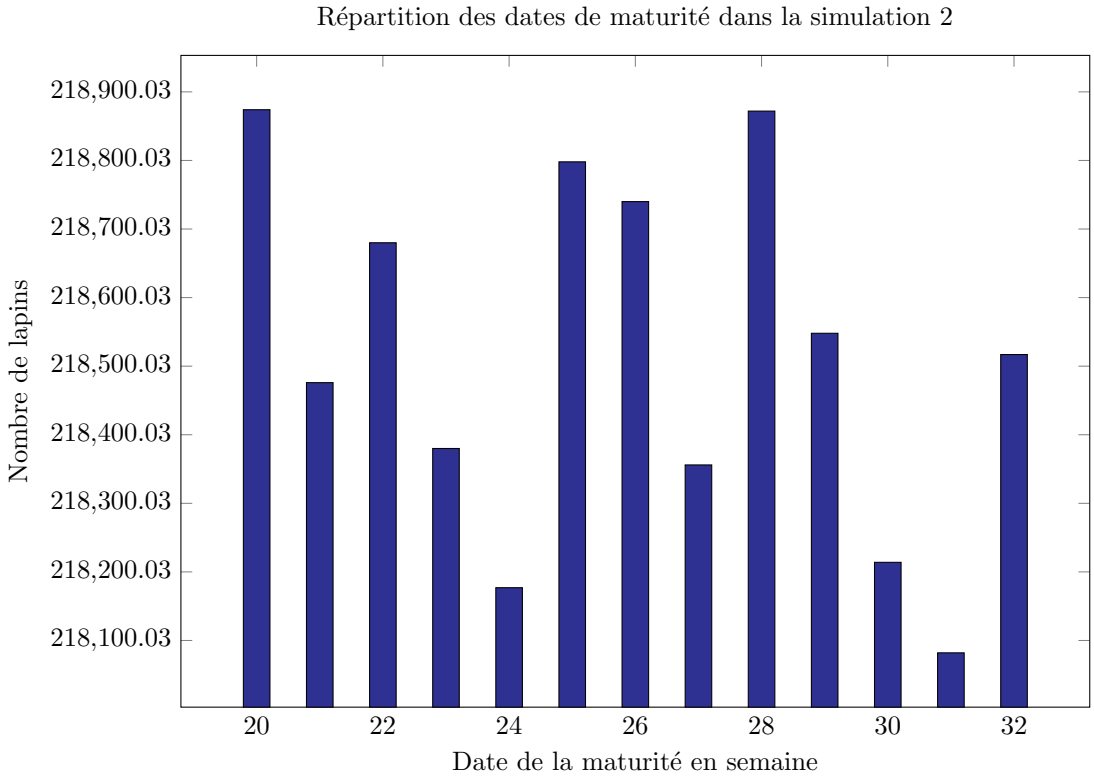
En normalisant le graphe et en changeant d'unité pour passer aux nombres de portées par an, on obtient le graphe suivant.



On obtient sur cette simulation un intervalle de confiance à 95% de ± 104 872 centré sur la moyenne de 366 471.

3.2 Maturité

L'âge de la maturité est déterminé par une loi uniforme dans l'intervalle de [4 ; 8] mois après la naissance.



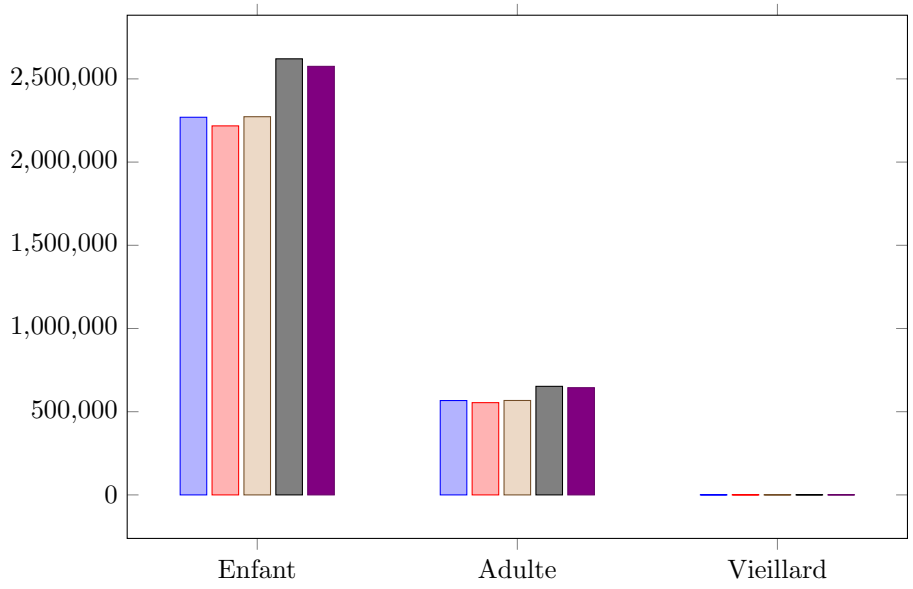
Malgrès la répartition qui semble à première vue chaotique, l'ordre de grandeur est très faible. Il y a en effet un delta maximal de 800 000 lapins, sur une échelle de 218 millions. Ce qui représente donc 3% de différence entre les extremums du graphe.

3.3 Mort

3.3.1 Période de la mort

Comme indiqué précédemment, la première étape du choix de la date de la mort est de choisir à quelle période de sa vie le lapin va mourir. Nous allons donc étudié la répartition entre ces périodes et comparer les simulations entre elles.

Répartition de la période de morts dans les simulations



On remarque sur ce premier graphique que la grande majorité des lapins meurt avant d'avoir atteint l'âge adulte. De plus une toute petite partie de la population atteint la période de durée de vie maximale. Nous allons maintenant calculer la moyenne et vérifier les proportions exactes.

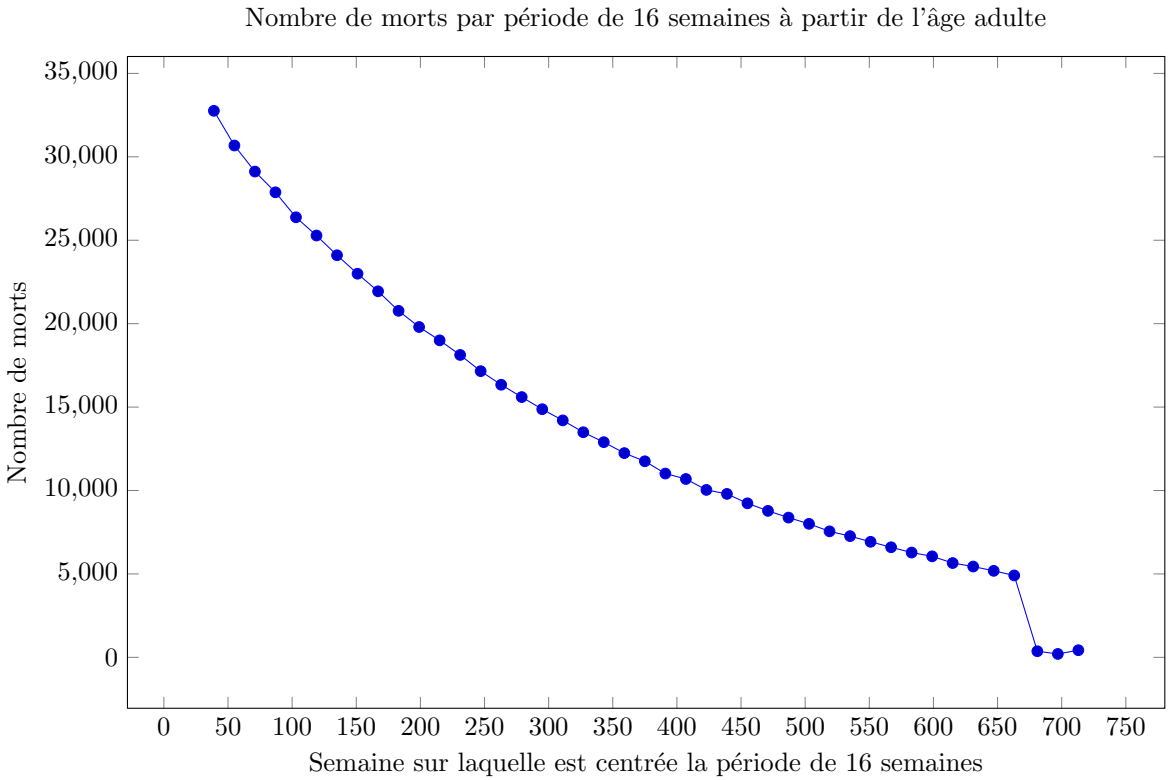
Répartition des périodes de morts dans les différentes simulations

	Période		
Simulation	Enfant	Adulte	Vieillard
1	2 269 370	567 040	521
2	2 217 600	554 427	551
3	2 272 674	567 453	587
4	2 620 560	652 463	659
5	2 575 320	644 277	610
Moyenne	2 391 104, 8	597 132	585,6
Répartition	80%	19,98%	0,02%
Intervalle de confiance à 95% (arrondi)	±108	±53	±2

On voit que la valeur de 80% de mortalité chez les jeunes lapins est respectée. On remarque également qu'une très faible partie des lapins atteint le troisième âge : 0.02%. On peut donc se questionner sur la réalisation de notre simulation. Il faudrait comparer avec une étude sur de vrais lapins pour s'assurer que notre implémentation est valide.
L'intervalle de confiance nous indique que 95% des simulations que l'on fera obtiendront les mêmes proportions, le rayon de confiance étant d'un ordre de grandeur bien plus faible que les moyennes.

3.3.2 Date de la mort

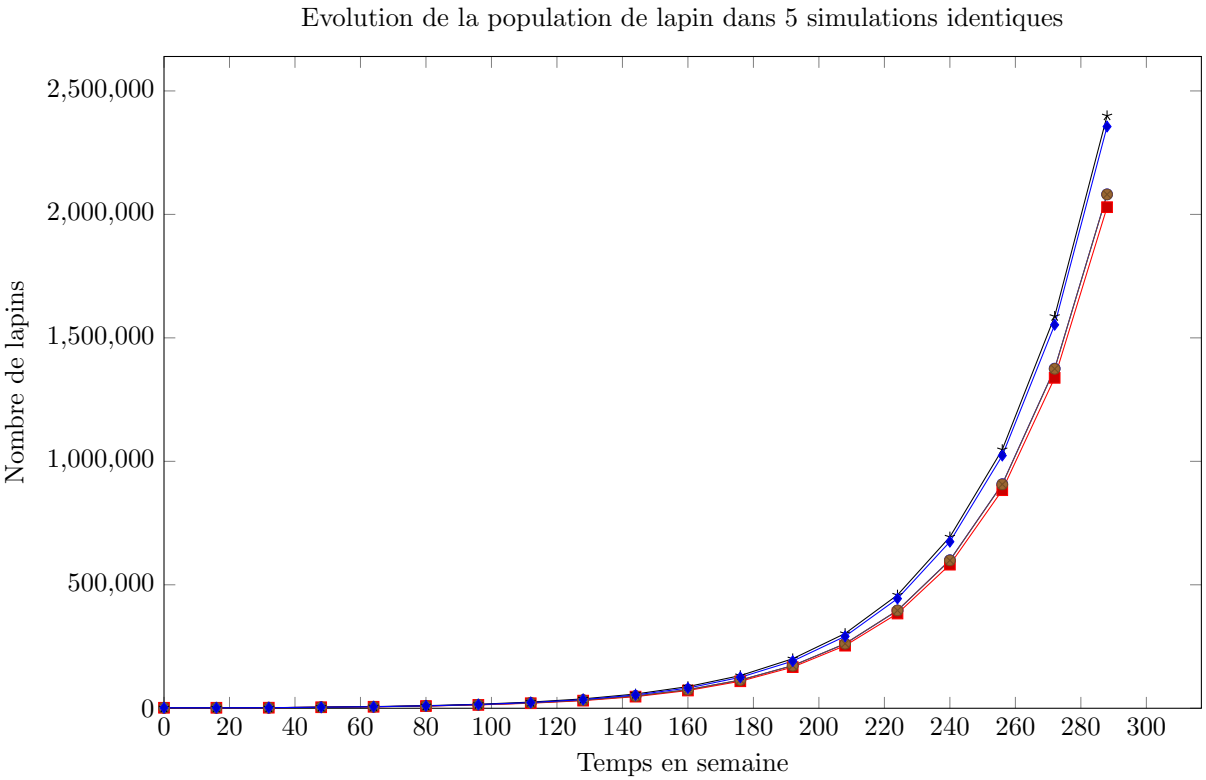
Nous avons décidé d'essayer de rendre la mortalité plus réaliste que de répartir uniformément les dates de décès. Pour ça nous réduisons la mortalité lorsque l'âge augmente. Jusqu'à atteindre la fin de la vie du lapin, où la probabilité augmentera jusqu'à atteindre la mort certaine à 15 ans.



Nous avons enlevé de ce graphe les morts des jeunes, qui rendaient le graphe trop illisible sur les dates à partir de 50 semaines. Nous avons également fait la somme des valeurs de 16 jours (en prenant la moyenne des 5 simulations), et fait un seul point avec. On remarque que la décroissance pendant l'age adulte est bien présente et semble réaliste. Cependant, le décrochement entre l'âge adulte et la fin de vie ne semble pas réaliste. De plus l'énoncé attendait une augmentation de la mortalité à partir de cet âge. Nous pouvons donc conlure que la simulation n'est pas bien réalisée sur ce point. Cependant on parle ici d'un nombre de cas assez limités de l'ordre de 10^4.

3.4 Population

Nous allons enfin étudier rapidement l'augmentation de la population. Nous voyons tout d'abord que la population de lapins augmente de manière exponentielle. De plus, un faible écart se traduit, au bout d'un an, par une différence de plusieurs centaines de milliers d'individus.



4 Conclusion

Pour conclure, nous pouvons dire que notre simulation obtient des résultats convenables sur la reproduction des lapins. Cependant l'algorithme de gestion des morts peut être revu pour obtenir des résultas plus proche de ce qui est attendu. Le calcul en amont semble également plus rapide mais il faudrait étudier plus profondément cet aspect, en comparant dans des conditions identiques les temps de calculs.