# MPIS TP2
## Implémentation de lois de probabilité

*Ballot Corentin, Simon Camille* 08/11/2016

Ce compte-rendu présente l'implémentation de trois lois de probabilité : la loi binomiale, la loi exponentielle et la loi de Poisson. Chaque partie de ce compte-rendu est dédié à une loi.
L'archive MPIS-TP2-CorentinBallot-CamilleSimon.tar.gz. contient pour chaque exercice :

- Les fichiers java des classes
- Un fichier nommé *gnuplot* contenant les commandes pour générer le(s) graphique(s)
- Des fichiers textes contenant les données pour gnuplot
- Les graphiques utilisés pour ce compte-rendu

Le sujet du TP est accessible ici : Sujet TP2

## 1 La loi binomiale

On s'intéresse ici à la loi Binomiale où $n$ est le nombre de tirage, $p$ la probabilité d'un succès, elle est comprise entre 0 et 1. Enfin $q$ est la probabilité d'un échec, donc $q = 1 - p$. Une variable aléatoire $X$ donnant le nombre de succès est obtenu en effectuant le calcul suivant :

$$P(X = k) = \binom{n}{k}.p^k.q^{n-k}$$

Pour calculer la combinaison $\binom{n}{k}$, on peut utiliser sa forme factorielle :

$$\binom{n}{k} = C_n^k = \frac{n!}{k!(n-k)!}$$

En informatique, la factorielle peut être calculée à l'aide d'une méthode récursive, mais cette approche peut entraîner un dépassement de la pile d'exécution (*stack overflow*) si le nombre d'appel à la fonction est très grand.
Pour le calcul de la combinaison, l'utilisation d'une méthode récursive pour la factorielle présente plusieurs désagréments : il nécessite beaucoup de temps et il n'est pas parallélisable. C'est à dire qu'il faut attendre d'avoir calculé complètement une factorielle avant d'entamer le calcule d'une autre. La pile d'exécution étant commune à tout le programme, si on exécute en même temps le calcul de deux factorielles l'espace mémoire utilisable pour chacune d'elle est deux fois moins grand que lorsque l'on ne calcule qu'une factorielle.
Afin d'éviter les problèmes liés à la méthode récursive du calcule de la factorielle, on préfère utiliser la définition suivante pour la combinaison :

$$\binom{n}{k} = \begin{cases} 1 & \text{si } k = 0 \text{ ou } k = n \\ \binom{n-1}{k} + \binom{n-1}{k-1} & \text{sinon} \end{cases}$$

On ajoute à notre classe Binomial une méthode *compute* qui calcule $X$ pour des $n$, $p$ et $k$ donnés. La méthode *generate* appelle *compute* pour $k$ variant de 0 à $t$. L'ensemble des données générées est enregistré dans un fichier grâce à la méthode *write*.

A l'aide de *GnuPlot*, on obtient à partir des données le graphiques suivants :

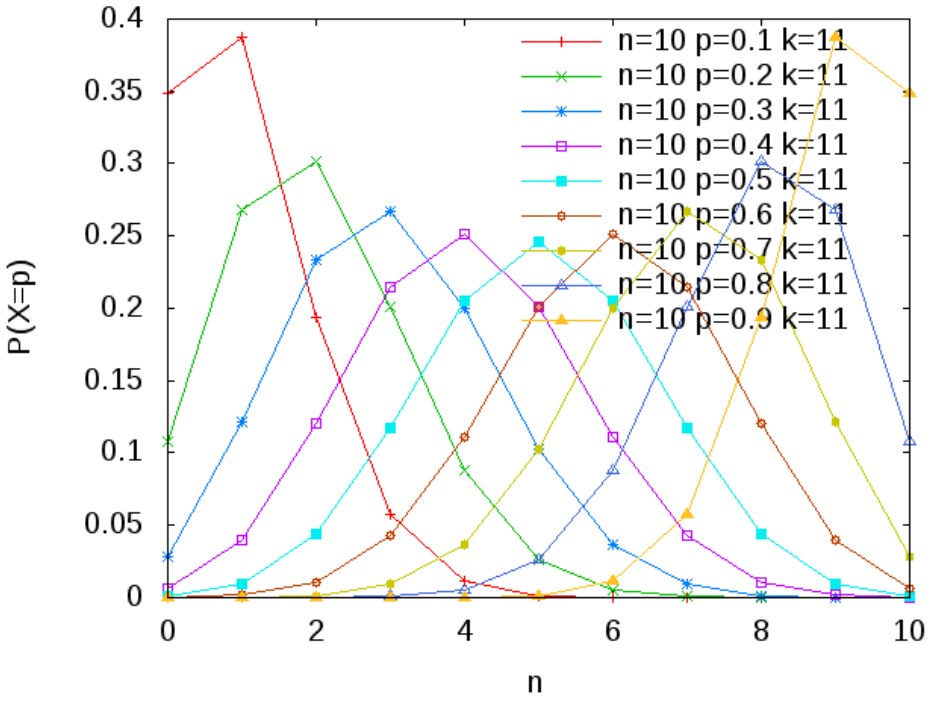


Figure 1: Courbes des lois binomiales avec n=10, k=11 et p variant de 0,1 à 0,9

## 2 La loi exponentielle

L'implémentation de la loi exponentielle est traitée en deux temps :

- Un partie pour générer la courbe p. 3 de l'énoncé, c'est à dire la représentation de la réciproque de la fonction de répartition.
- Une deuxième partie qui va modéliser les résultats de la loi exponentielle pour une valeur d'entrée aléatoire.

le traitement de la fonction de répartition puis l'implémentation de la fonction de masse.

### 2.1 Réciproque de la fonction de répartition

Dans un premier temps, on s'intéresse à la fonction de répartition de la loi exponentielle $y = 1 - e^{\lambda x}$. Mais nous voulons les valeurs de $x$ pour $y$ variant de 0 à 1. Il est donc nécessaire de transformer la fonction de répartition pour obtenir une fonction de $y$, la fonction réciproque de la fonction de répartition :

$$y = 1 - e^{-\lambda x} \Leftrightarrow x = -\frac{ln(1-y)}{\lambda}$$

On crée une méthode *compute* qui utilise la formule ci-dessus pour des paramètres $\lambda$ et $y$ fixés. La méthode *generate* quant à elle, va générer l'ensemble des $x$ pour $y$ variant de 0 à 0,999. On remarque qu'il n'est pas possible de choisir $y = 1$, car $ln(0) = -\infty$. Essayons maintenant de déterminer la valeur maximale de $x$ :

$$\begin{array}{lccc}
\Leftrightarrow & y & \in & [0;1[ \\
\Leftrightarrow & 1-y & \in & ]0;1] \\
\Leftrightarrow & ln(1-y) & \in & ]-\infty;0] \\
\Leftrightarrow & -ln(1-y) & \in & [0;+\infty[ \\
\Leftrightarrow & -\frac{ln(1-y)}{\lambda} = x & \in & [0;+\infty[
\end{array}$$

Comme $x \in$ [0;+$\infty$[, $x$ n'a pas de valeur maximale.
On trace maintenant les courbes pour $y$ allant de 0 à 1 et pour des $\lambda$ valant 0,5, 1,0 et 1,5.

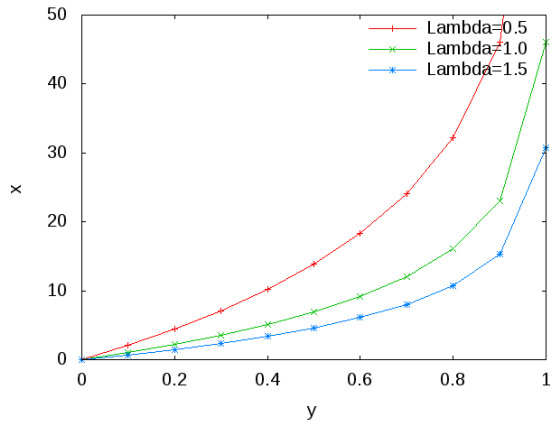


Figure 2: Histogramme de l'exponentielle pour $y$ variant de 0 à 1 et $\lambda_1 = 0.5$, $\lambda_2 = 1.0$ et $\lambda_3 = 1.5$

## 2.2 Densité de probabilité

On s'intéresse maintenant à la fonction de masse de la loi exponentielle. Il s'agit ici de montrer les valeurs les plus fréquentes de $x$ pour un $\lambda$ donné et un $y$ généré aléatoirement. La fonction *generate* dans cette partie utilise la fonction de la partie précédente avec un $y$ généré grâce au RANDOM de Java. Pour une meilleur lisibilité du graphique, on multiplie le résultat ce calcul par 10. On obtient le graphique suivant :

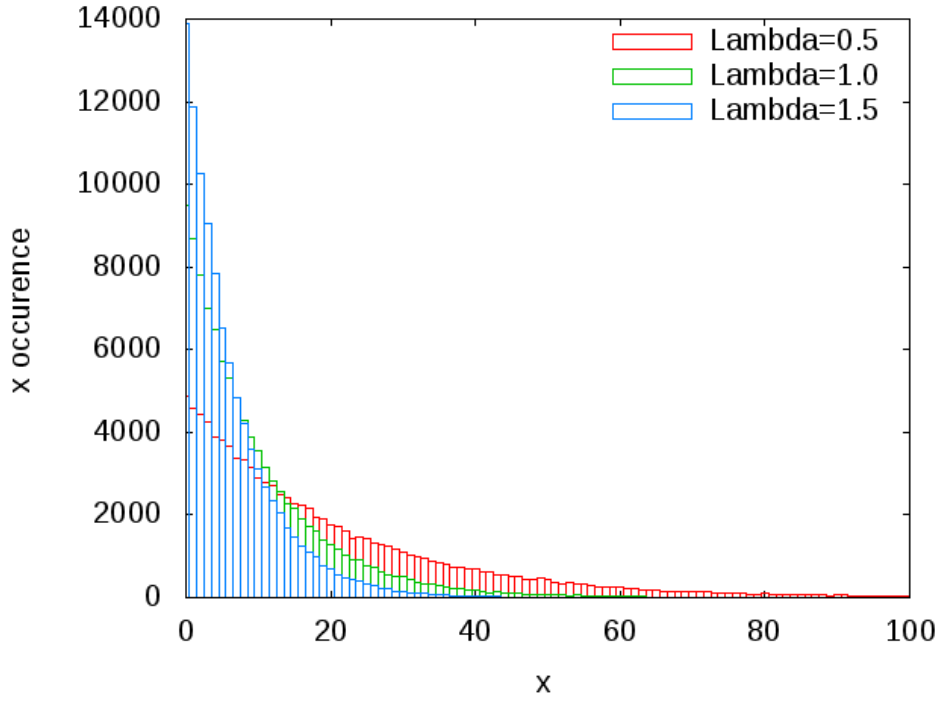


Figure 3: Courbe représentant le nombre d'occurrence $y$ en fonction de $x$ pour des $\lambda$ donnés : $\lambda_1 = 0.5$, $\lambda_2 = 1.0$ et $\lambda_3 = 1.5$

## 3 La loi de Poisson

Il nous reste maintenant à traiter de la loi de Poisson. Elle s'écrit de la façon suivante, avec $\lambda$ le paramètre de la loi :

$$P(X = k) = e^{-\lambda} \times \frac{\lambda^k}{k!}$$

Dans notre exercice nous voulons les instants d'arrivées, or nous savons que le temps entre deux arrivées suit une loi exponentielle. Si on pose $X$, la variable aléatoire entre deux arrivées, on obtient la fonction de répartition suivante :

$$F_x(D) = -\lambda D \text{ si } x \geq 0$$

A partir d'ici, on peut utiliser la classe EXPONENTIELLE de l'exercice précédent. En effet, la méthode *compute* nous donne $x$ sachant $y$ se qui s'applique très bien dans notre cas puisque nous connaissons $D$, $\lambda$ et que nous voulons trouver $X$.

La fonction *generate* va générer pour chaque intervalle de temps le nombre de naissance. On ajoute à cette fonction la notion de fiabilité ce qui va permettre de lisser la courbe.

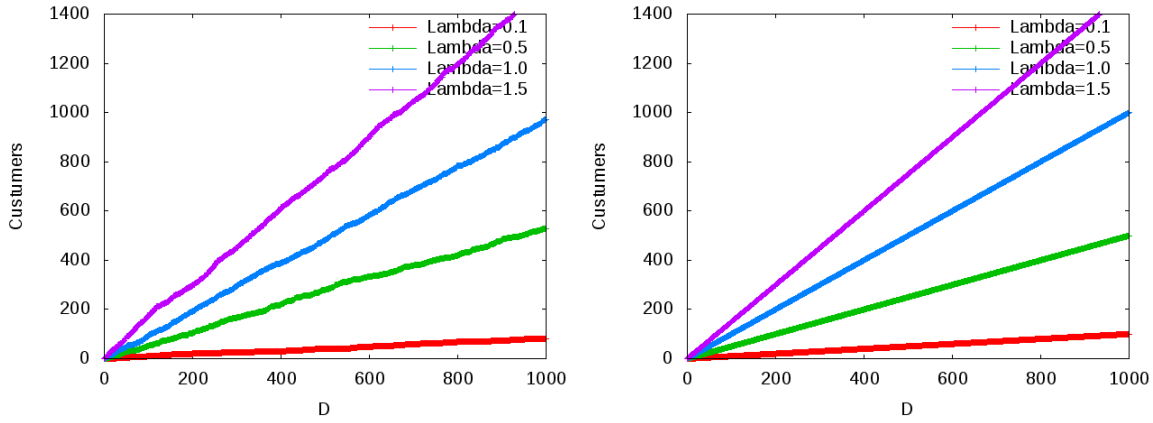


Figure 4: Courbes représentant le nombre de "clients" $y$ au cours du temps avec : $\lambda_1 = 0.1$, $\lambda_2 = 0.5$, $\lambda_3 = 1.0$ et $\lambda_4 = 1.5$. A gauche pour une fiabilité de 1 et à droite pour une fiabilité de 100000.

# 4 Conclusion

Au cours de ce TP, un seul problème a posé vraiment des difficultés, il s'agit des approximations des valeurs numériques. En effet, le langage et la machine ne permettent pas d'être très précis, ils ne peuvent pas représenter l'ensemble des nombres réels. Plusieurs réponses sont possibles face à ce problème : utiliser des librairies java adapté (BigInteger), faire attention à l'utilisation des cast et enfin, utiliser la fiabilité ce qui va permettre de limiter les effets des valeurs extrêmes.
Par la suite, on peut imaginer améliorer ce programme non seulement en y ajoutant d'autres lois de probabilités mais également en proposant d'autres méthodes de calcul des lois déjà implémenté.