# Rapport TpH

Oxana USHAKOVA

December 6, 2016

# Notation

- $S$ - Stock price, also called Spot price (or any underlying asset)
- $X$ - value of an option, depending on time and spot price
- $K$ - Strike price
- $r$ - risk-free rate
- $d$ - dividend yield
- $\mu$ - drift rate of $S$ - the rate at which the average of $S$ changes
- $\sigma$ - volatility of the stock, standard deviation of $log(S)$ - return on stock
- $T_0$,$T$ - initial and final time
- $\theta$ - long variance : as $t$ tends to infinity, the expected value of $\nu$ tends to $\theta$
- $\kappa$ - the rate at which $\nu$ reverts to $\theta$
- $\xi$ - the volatility of volatility

## Remarque historique

Avant Black-Scholes il y avait plusieur tentatives de trouver un bon modèle pour option pricing. Parmi d'autres on s'intéresse au modèle de Louis Bachelier décrit en "Théorie de la spéculation"(1900):

$$X_0 = x_0$$
$$X_t = \mu + rt + \sigma W_t$$

On voit tout de suite que ce modèle est additif et donc admet le résultat **négatif**. Vu que en réalité les prix sont toujours positifs la nouvelle génération des modèles (Samuelson, Black-Scholes, etc) sont multiplicatifs.

## Diffusion de Black-Scholes et Chain Rule

Voici la diffusion de Black-Scholes:

$$dX_t = \mu X_t dt + \sigma X_t dW_t, X_0 > 0 \tag{1}$$

Soit $f(t,x) = \ln x, f \in C^2$ et $Y_t = \ln X_t$. Alors:

$$dY_t = \frac{1}{X_t} dX_t + \frac{1}{2}(-\frac{1}{X_t^2})(dX_t)^2 = \mu dt + \sigma dW_t - \frac{1}{2}\frac{1}{X_t^2}\sigma^2 X_t^2 dt = (\mu - \frac{1}{2}\sigma^2)dt + \sigma dW_t. \tag{2}$$

On applique la lemme d'Ito:

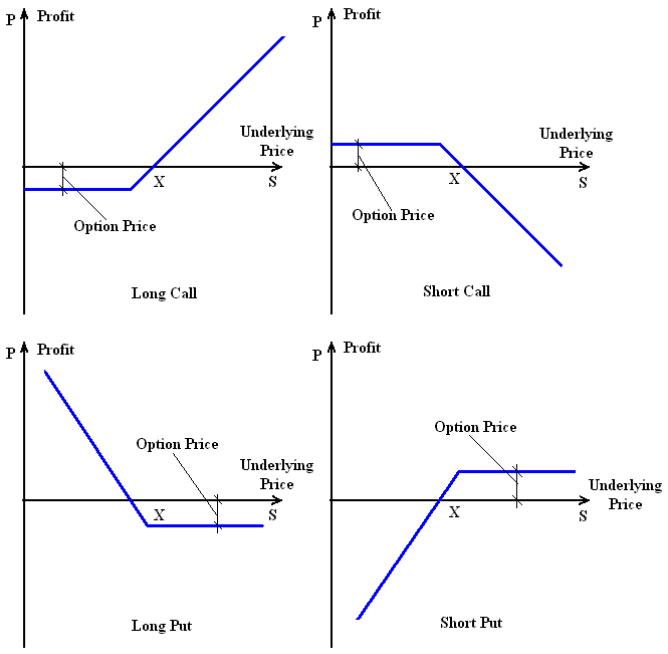
$$Y_t - Y_0 = \int_0^T dY_t = (\mu - \frac{1}{2}\sigma^2)\int_0^T dt + \sigma \int_0^t dW_t = (\mu - \frac{1}{2}\sigma^2)T + \sigma W_t \tag{3}$$

Et voila la solution:

$$X_t = e^{Y_T} = e^{Y_0 + (\mu - \frac{1}{2}\sigma^2)T + \sigma dW_t} = S_0 e^{(\mu - \frac{1}{2}\sigma^2)T + \sigma dW_t} \tag{4}$$

## Contrats avec Toto

Avant de trouver les prix demandés, regardons sur les payoffs de call et de put de notre point de vue (acheteur = "long") et de point de vue de Toto (vendeur = "short").




Voici les justes prix pour call/put options dans les cas où l'argent est placé/pas placé. On appele $e^{-rT}$ - "discounting factor".

| | Argent pas placé | Argent placé |
|---|---|---|
| Call | X= E[max{S(T)-K,0}] | X= exp(-rT) * E[max{S(T)-K,0}] |
| Put | X= E[max{K-S(T),0}] | X= exp(-rT) * E[max{K-S(T),0}] |

# Simulations

Les EDP stochastiques dont on parle en ce TpH sont particulièrement intéressantes pour les marchés financiers. Comme c'était dit avant, le modèle de Black-Scholes n'est pas le seul :

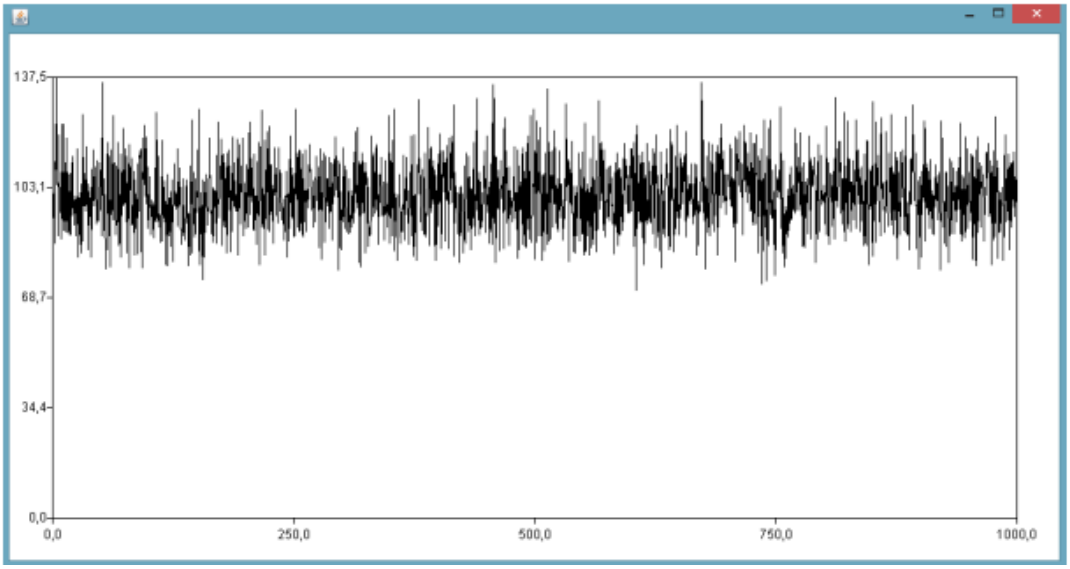
| | Diffusion | Jump Element | Stochastic Element | Pure Jump |
|---|---|---|---|---|
| Black-Scholes | yes | no | no | no |
| Heston | yes | no | yes (volatility) | no |
| Vasicek | yes | no | yes (interest rate) | no |
| Merton | yes | yes | no | no |
| Bates | yes | yes | yes (volatility) | no |
| CGMY,etc | no | no | no | yes |

En gros, tous les modlès peuvent être divisés en trois types: modèles à diffusion, à diffusion et sauts, à sauts. C'est également intéressant pour nous de voir le soustype "avec un élément stochastique" - par exemple, le processus de Ornstein-Uhlenbeck est en fait le modèle de Vasicek : modèle à diffusion à un élément stochastique (interest rate). De même on a le modèle de Heston : modèle à diffusion à un élément stochastique (volatility).

## Black-Scholes et Merton

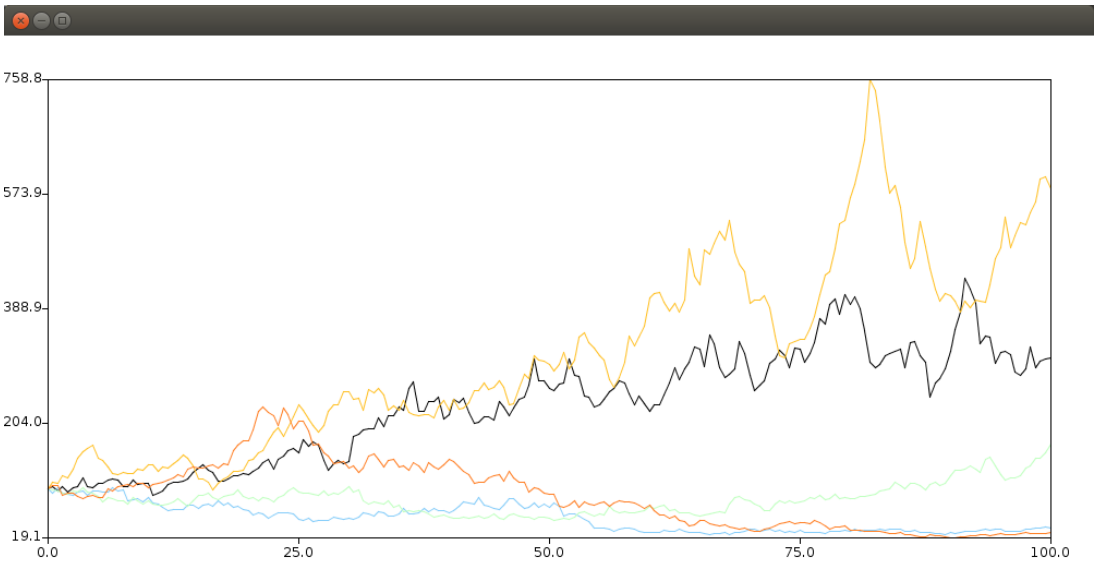
D'abord, le bruit:

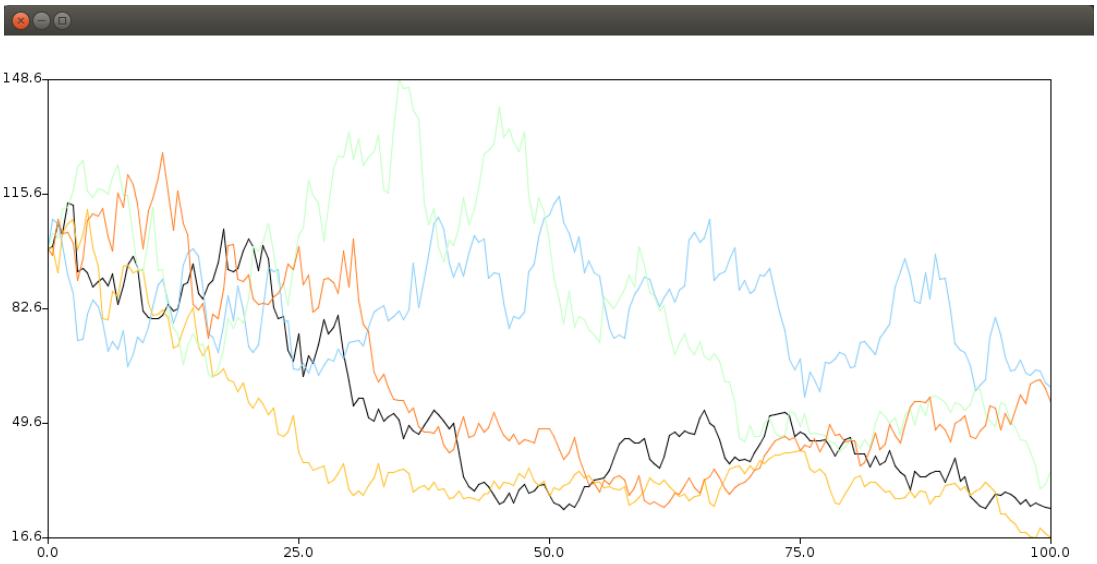
```
public BS(double droite, double S, double X, double r,double d, double
sigma, double T) {

    Xs.add(0.);
    Ys.add(0.);

    int i = 0;
    while (Xs.get(i) < droite) {
        Xs.add(Xs.get(i) + T);
        Ys.add( S * Math.exp((r - d) * T - 0.5 * sigma * sigma * T + sigma
* rd.nextGaussian() * Math.sqrt(T)));
        i++;
    }
}
```




Les marches aléatoires des modèles Black-Scholes (diffusion) et Merton (jump-diffusion).

```
public BS(double droite, double S, double X, double r,double d, double sigma, double T,double l,double k, double theta) {

    Xs.add(0.);
    Ys.add(S);
    //note that mu=r-d
    int i = 0;
    while (Xs.get(i) < droite) {
        Xs.add(Xs.get(i) + T);
        //BS model
        Ys.add(Ys.get(i) + Ys.get(i) * (r - d ) * T + Ys.get(i) * sigma * Math.sqrt(T) * rd.nextGaussian());
        //Merton model: l = nb des sauts par an, k = taille de sauts (% de prix)
        Ys.add(Ys.get(i) + Ys.get(i) * (r - d - l * k) * T + Ys.get(i) * sigma * Math.sqrt(T) * rd.nextGaussian()+Math.sqrt(T)*mrd.nextPoisson(l));
        i++;
    }
}
```

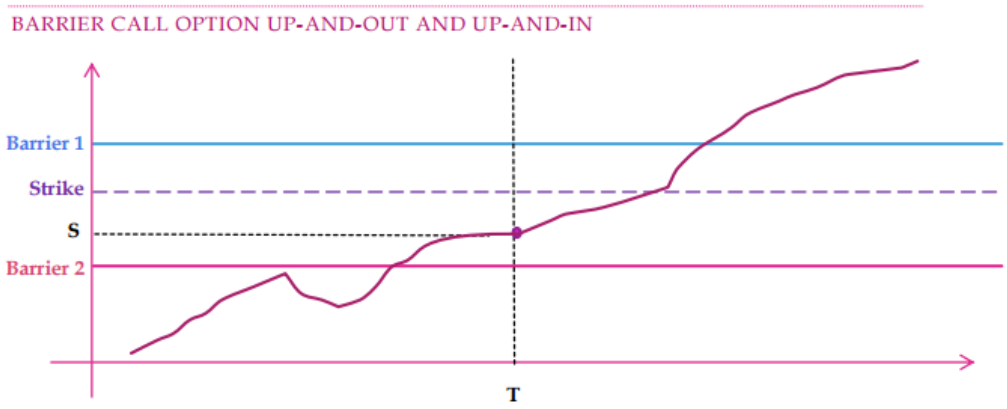





## Barrier option avec Monte Carlo

Le contrat de Toto s'appele "vanilla" option, un contrat très basic. Actuellement, il y a dizaines de types des options qui forment la classe des options dites "exotics". Par exemple, une "barrier" option - cette option peut être exécutée ssi le prix d'actif sousjacent touche/ne touche pas un barrier donné.
Comparons les prix de "barrier" ooption trouvés par Monte Carlo et de "vanilla" option trouvés par les formules analytiques (cf. Annexe).

Payoff d'un Barrier Option:




$$Si\ B < Strike: \begin{cases} C_{uo} = 0; \\ C_{ui} = C. \end{cases}$$

$$Si\ B > Strike: \begin{cases} C_{uo} = C - C_{ui}\ ; \\ C_{ui} = S_0\mathcal{N}(x_1)e^{-qT} - Ke^{-rT}\mathcal{N}\left(x_1 - \sigma\sqrt{T}\right) - S_0e^{-qT}\left(\frac{H}{S_0}\right)^{2\lambda}[\mathcal{N}(-y) - \mathcal{N}(-y_1)] + \\ Ke^{-rT}\left(\frac{H}{S_0}\right)^{2\lambda-2}\left[\mathcal{N}\left(-y + \sigma\sqrt{T}\right) - \mathcal{N}\left(-y_1 + \sigma\sqrt{T}\right)\right]. \end{cases}$$

Implémentation en Java:

```java
// estimate a barrier option by Monte Carlo simulation
public static double call(double S, double X, double r,double d, double sigma, double T, double B )
{
    int N = 10000;
    double sum = 0.0;
    for (int i = 0; i < N; i++) {

        Random rd = new Random();
        double eps = rd.nextGaussian();
        double price = S * Math.exp((r-d) * T - 0.5 * sigma * sigma  * T + sigma * eps * Math.sqrt(T));
        if (price < B){
            double value = Math.max(price - X, 0);
            sum += value;}
        }
    double mean = sum / N;
    return Math.exp(-r * T) * mean;
}
```

```java
// Black-Scholes formula
public static double callPrice(double S, double X, double r,double d,  double sigma, double T)
{
    double d1 = (Math.log(S / X) + (r + sigma * sigma / 2) * T) / (sigma * Math.sqrt(T));
    double d2 = d1 - sigma * Math.sqrt(T);
    return  Math.exp(-d * T)*S * MyRandom.Phi(d1) - X * Math.exp(-r * T) * MyRandom.Phi(d2);
}
```

*Phi(x) – revoie l'approximation de Taylor de la fonction de répartition du loi Normale.*

Voici le résultat pour "up and out" barrier option et vanilla option: au premier cas la valeur du barrier est inférieur à Strike, donc l'option vaut toujours zero.

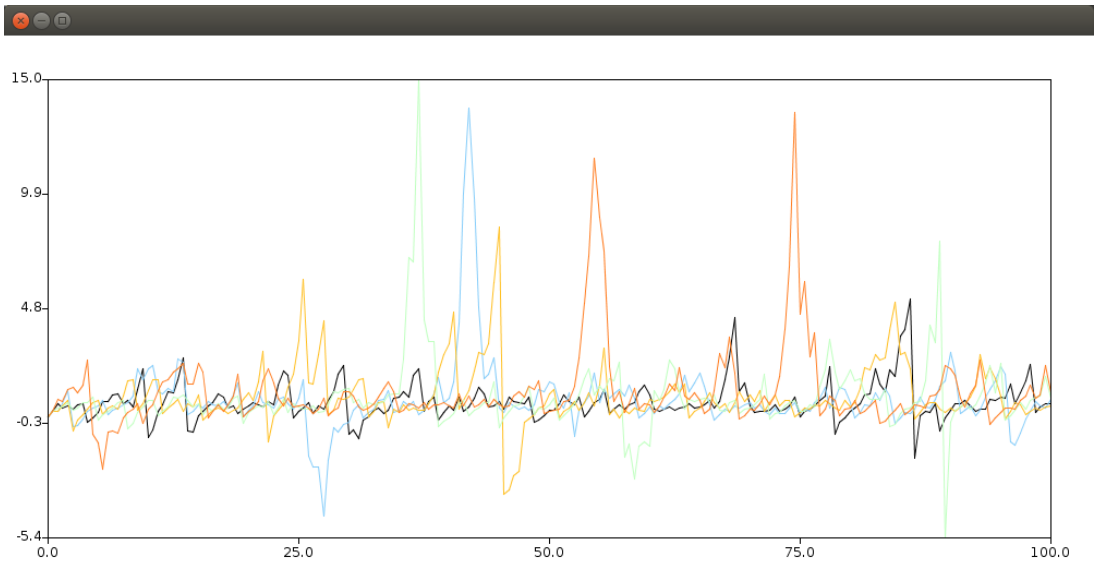
```
S0 = 100.0, Strike = 110.0, barrier 1 = 105.0, barrier 2 = 155.0
BS 0.3353001289550601
MC à barrier 1 up and in  0.0
MC à barrier 2 up and in  0.34768741304018536
```

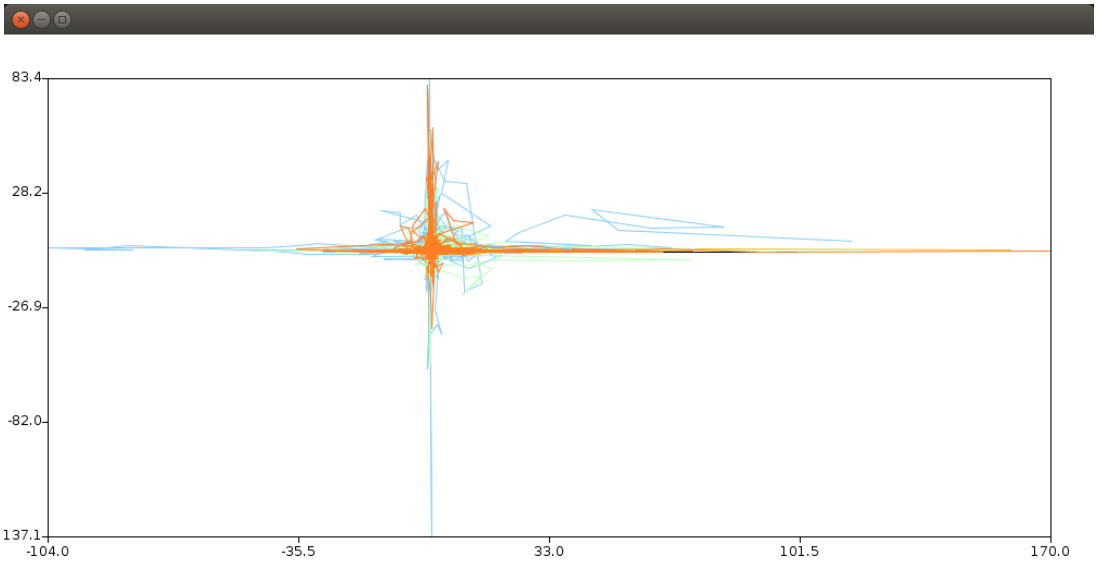
## Vasicek / Ornstein-Unlenbeck

Le processus de Ornstein-Uhlenbeck décrit le mouvement d'un particule qui revient vers le centre ($\mu$). En finance, ce processus est inclut dans le modèle de Vasicek utilisé en pricing des options dont le prix dépend de taux (Credit Derivatives) - la taux d'intérêt $r$ suit le processu d'Orstein-Unhelbeck.

```java
public OrUh(double droite, double mu, double sigma, double T, double theta) {

    Xs.add(0.);
    Ys.add(0.);

    int i = 0;
    while (Xs.get(i) < droite) {
        Xs.add(Xs.get(i) + T);
        Ys.add(Ys.get(i) + theta * (mu - Ys.get(i)) * T +  sigma* Ys.get(i) * Math.sqrt(T) * rd.nextGaussian());
        i++;
    }
}
```

Implémentation en 1D et 2D:




## Heston

Le modèle de Heston comme le modèle de Vasicek a un élément stochastique mais cette fois - la volatilité. Ici on donne que le code (no output produced).

```
public Heston(double droite, double mu, double theta, double kappa, double e, double T) {

    Xs.add(0.);
    Ys.add(0.);
    r.add(0.03);
    int i = 0;

    while (Xs.get(i) < droite) {
        Xs.add(Xs.get(i) + T);
        r.add(r.get(i) + kappa * (theta - r.get(i)) * T + e * Math.sqrt(r.get(i)) * Math.sqrt(T) * rd.nextGaussian());
        Ys.add(Ys.get(i) + Ys.get(i) * mu * T + Ys.get(i) * Math.sqrt(r.get(i)) * Math.sqrt(T) * rd.nextGaussian());
        i++;
    }

}
```

## Lotka-Volterra

Le modèle de Lotka-Volterra est un couple d'équations différentielles non-linéaires du premier ordre, et sont couramment utilisées pour décrire la dynamique de systèmes biologiques dans lesquels un prédateur et sa proie interagissent.

```java
public LV2D(double droite, double T, double eps) {
    Xs.add(0.);
    Ys.add(0.);
    int i = 0;
    while (Xs.get(i) < droite) {
        Xs.add(Xs.get(i)+Xs.get(i)*T-Xs.get(i)*Ys.get(i)*T+eps*Xs.get(i)*Math.sqrt(T)*rd.nextGaussian());
        Ys.add(Ys.get(i)-Ys.get(i)*T+Ys.get(i)*Xs.get(i)*T+eps*Ys.get(i)*Math.sqrt(T)*rd.nextGaussian());
        i++;
    }
}
```

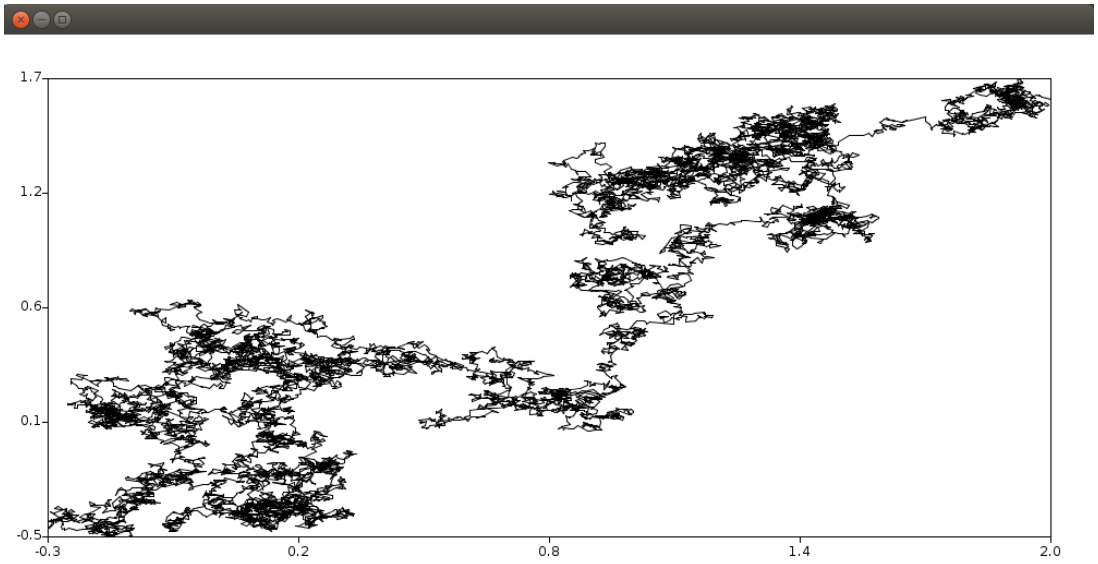
**Java heap space**

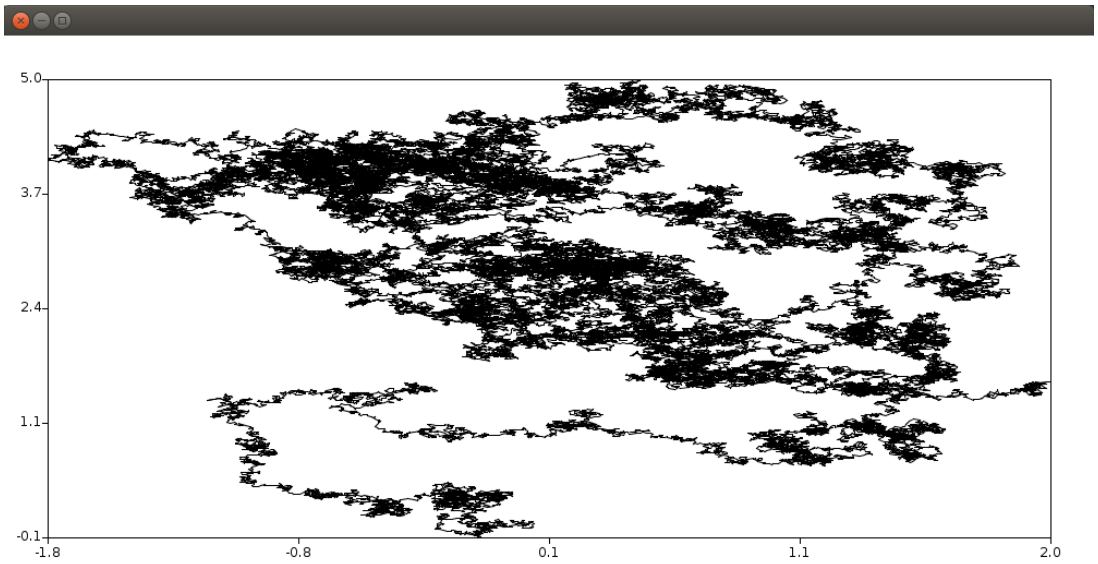
## 2D et 3D

Passont à 2D: on peut mettre Wiener en $Ox$ et $Oy$ ou on peut ajouter des des v.a., des dépendences entre l'abscisses et l'ordonnées, etc pour générer des jolies marches comme celles-ci:

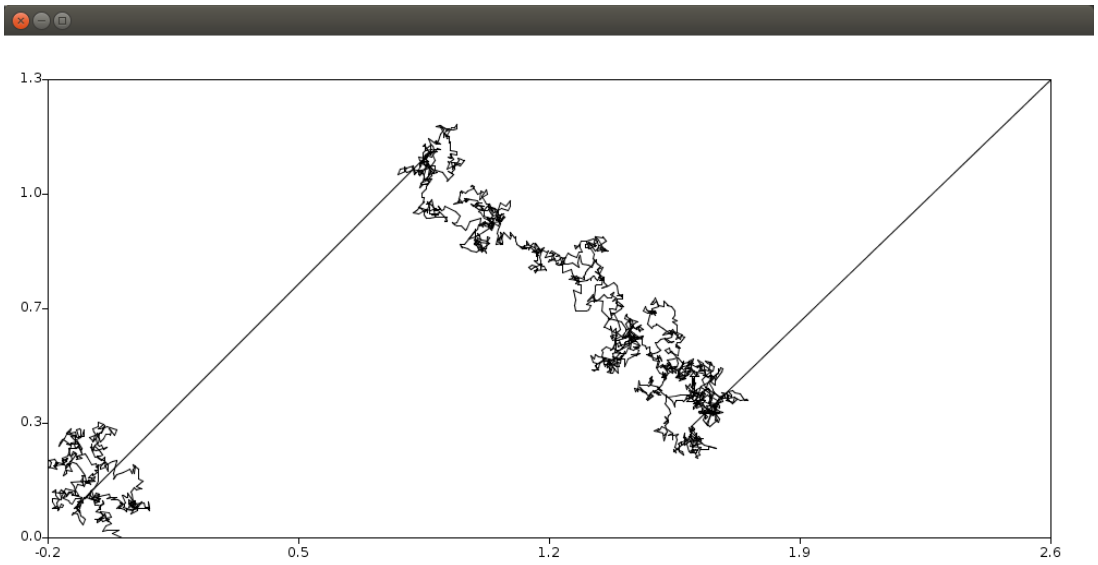
```java
    while (Xs.get(i) < droite) {
        double z=rand.nextPoisson(0.005);
        Xs.add(Xs.get(i) +  Math.sqrt(pas) * rand.nextGaussian() + z);
        Ys.add(Ys.get(i) +  Math.sqrt(pas) * rand.nextGaussian() + z);
        i++;
    }
}
```

Wiener:

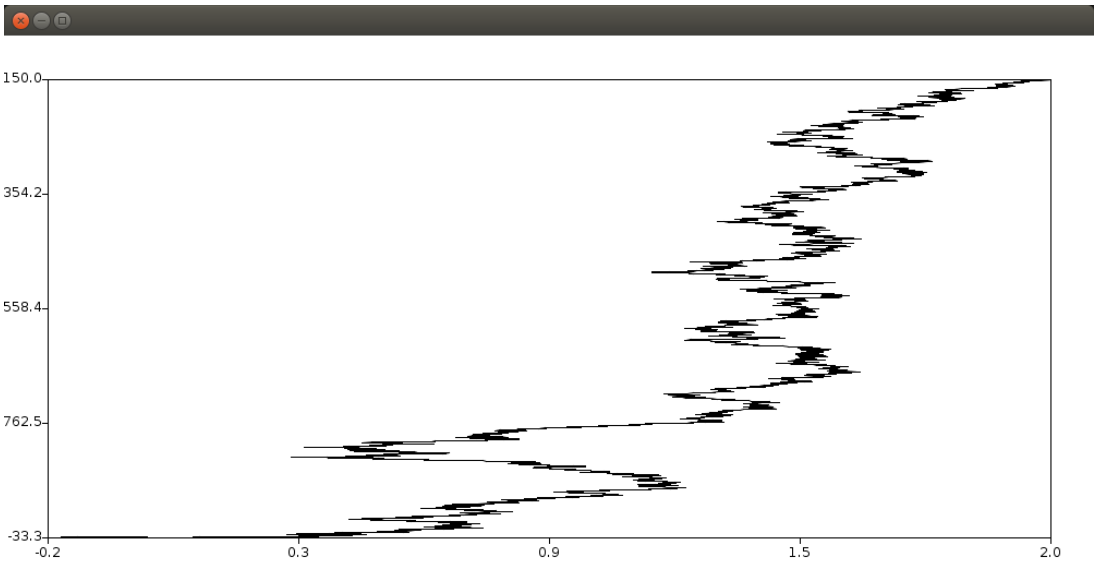




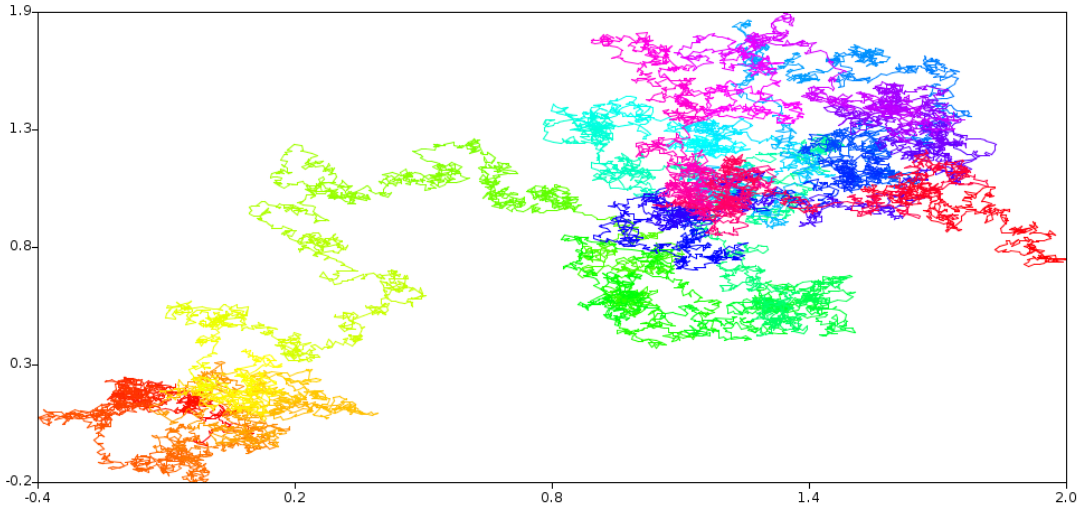

Wiener avec Poisson:




Dépendance de $Oy$ de $Ox$:




Maintenant - 3D: ici le couleur représente la troisème dimension, ça peut être la dimension en espace, la temperature, la rotation, etc.




# Annexe

## Solution Analytique

En utilisant :

- $d_1 = \frac{\ln\left(\frac{S_0}{K}\right)+\left(r+\frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$
- $d_2 = \frac{\ln\left(\frac{S_0}{K}\right)+\left(r-\frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$

$$[call] = \exp\{-rT\} * \mathbb{E}\left[\max\left(S_0 * e^{\left(r-d-\frac{\sigma^2}{2}\right)T+x*\sigma\sqrt{T}}, 0\right)\right]$$

$$= \exp\{-rT\} * \int_{-d_2}^{\infty} (S_0 * e^{\left(r-d-\frac{\sigma^2}{2}\right)T+x*\sigma\sqrt{T}} - K) * \frac{1}{\sqrt{2\pi}} * e^{\frac{x^2}{2}} dx \ (\spadesuit)$$

$$= \int_{-d_2}^{\infty} e^{-rT} * e^{rT} * e^{-dT} * e^{-\frac{\sigma^2 T}{2}} * e^{x*\sigma\sqrt{T}} * S_0 * \frac{1}{\sqrt{2\pi}} * e^{\frac{x^2}{2}} dx - \int_{-d_2}^{\infty} K * \frac{1}{\sqrt{2\pi}} * e^{\frac{x^2}{2}}$$

$$* e^{-rT} dx = e^{-dT} * S_0 * \frac{1}{\sqrt{2\pi}} \int_{-d_2}^{\infty} e^{\frac{-(x-\sigma\sqrt{T})^2}{2}} dx - e^{-rT} * K * \frac{1}{\sqrt{2\pi}} \int_{-d_2}^{\infty} e^{\frac{x^2}{2}} dx$$

$$= e^{-dT} * S_0 * \left(1 - \mathcal{N}\left(-d_2 - \sigma\sqrt{T}\right)\right) - e^{-rT} * K * (1 - \mathcal{N}(-d_2))$$

$$= e^{-dT} * S_0 * \mathcal{N}(d_1) - e^{-rT} * K * \mathcal{N}(d_2).$$

Trouvons les bornes (♣) :

$$(S_0 * e^{\left(r-d-\frac{\sigma^2}{2}\right)T+x*\sigma\sqrt{T}}) > K$$

$$e^{\left(r-d-\frac{\sigma^2}{2}\right)T+x*\sigma\sqrt{T}} > K/S_0$$

$$\left(r - d - \frac{\sigma^2}{2}\right) T + x * \sigma\sqrt{T} > \ln\left(\frac{K}{S_0}\right)$$

$$x > \frac{\ln\left(\frac{K}{S_0}\right)-\left(r-d-\frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

$$x > -\frac{\ln\left(\frac{K}{S_0}\right)+\left(r-d-\frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} = -d_2$$