Simulation Stochastique

Liu Jingyi - Nicolas Bonneel

6 juin 2005

1 La methode de la fonction de répartition

1.1 Loi de Bernoulli

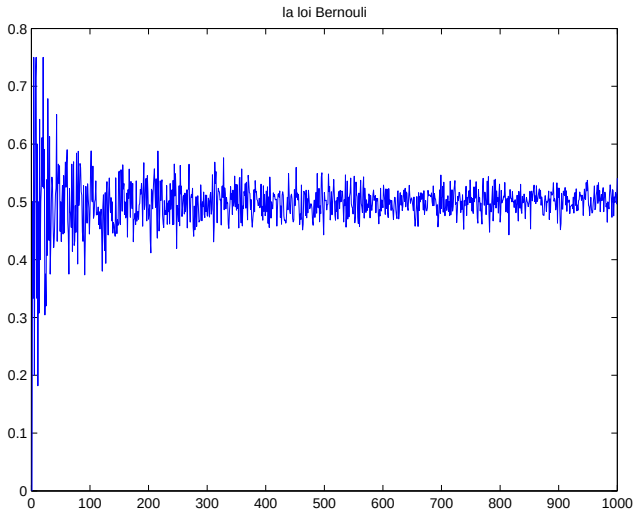
On se sert de la fonction de répartition suivante :

$$\begin{aligned} F_X^{-1}(\alpha) &= 0 \quad \text{si } \alpha \leq 1-\theta \\ &= 1 \quad \text{si } \alpha > 1-\theta \end{aligned}$$

Avec $\theta = 0.5$, on obtient :

```
for n=1:1000
 X(n)=sum(rand(1,n)<=0.5)/n;
end;
```

On obtient la simulation suivante :



et on remarque que les valeurs tendent vers 0.5.

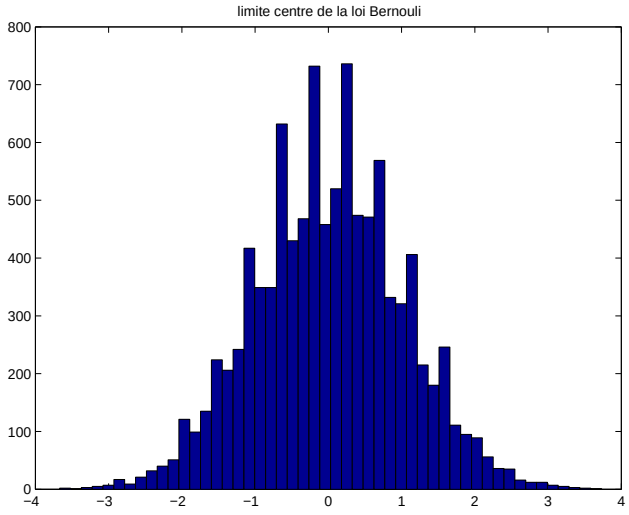
On utilise maintenant le théorème de la limite centrale pour simuler une loi Normale à partir de lois de Bernoulli :

$$2\sqrt{(n)}\left(\frac{\sum_{i=1}^{N} B(n,p)}{N} - p\right) \sim \mathcal{N}(0,1)$$

On obtient :

```
for i=1:10000
 Y(i)=(sum(rand(1,1000)<=0.5)/1000-0.5)*sqrt(1000)*2;
end;
```

dont l'histogramme nous donne la répartition d'une loi Normale :



1.2 Loi exponentielle

On utilise la fonction de répartition :

$$F_X(t) = 1 - exp(-\theta t) \quad (\theta \in \mathbb{R}_*^+ \quad \forall t \geq 0)$$

Donc :

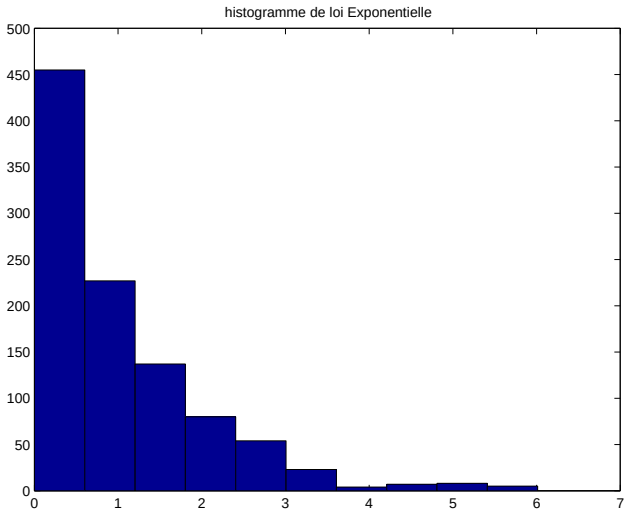
$$F_X^{-1}(t) = -\theta^{-1}\, log(1-\alpha) = -\theta^{-1}\, log(u)$$

avec $u = 1 - \alpha \quad \in]0,1[$ uniforme.

On obtient le code :

```
lamda = 1;
z     = (-log(rand(1,1000))/lamda);
```

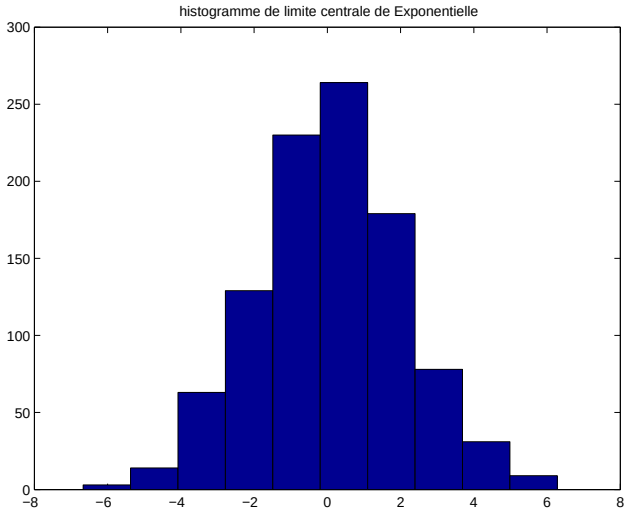
qui nous donne l'histogramme suivant :



De même on utilise le théorème de la limite centrale pour simuler une loi normale à partir de lois exponentielles :

$$2\sqrt{n}.\left(\frac{\sum_{i=1}^{n}\epsilon(\lambda)}{n}-\frac{1}{\lambda}\right)\sim\mathcal{N}(0,1)$$

On obtient la répartition d'une loi normale :



1.3 Applications

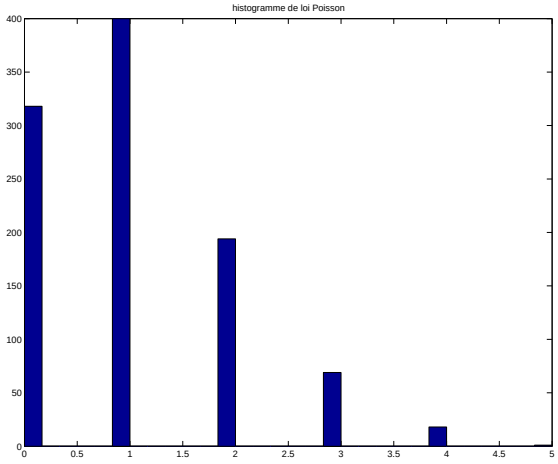
1.3.1 Loi de Poisson

On cherche à simuler une loi de Poisson de paramètre λ à partir de loi exponentielles de paramètre 1.

On calcule la somme : $s_k = \sum_{j=1}^{k} x_j$ puis $N = \sum_{k\geq 1} 1_{\{s_k \leq \lambda\}}$ pour $\lambda > 0$. On aura alors $N \sim \mathcal{P}(\lambda)$.

On obtient alors le code :

```
lamda=1;
for i=1:1000
  z =(-log(rand(1,1000))/lamda);
  for k=1:1000
    sk(k)=sum(z(1:k));
  end;
  N(i) = sum(sk<=lamda);
end;
```

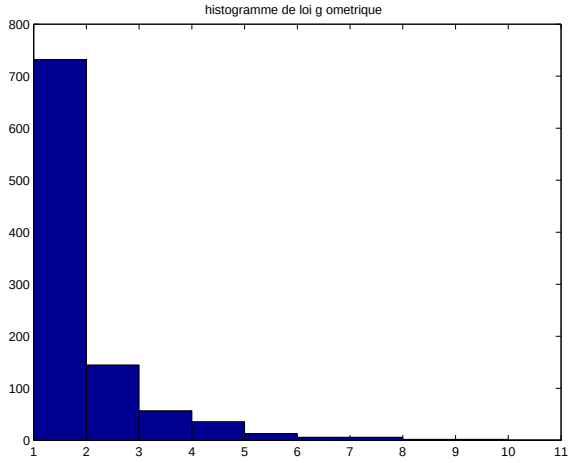


1.3.2 Loi de Géometrique

– première méthode On génère une loi de bernoulli de paramètre 0.5, et on regarde l'indice du premier 1. Cette variable est une loi géométrique. On obtient :

```
P=0;
for i = 1:1000
  for n=1:30
    S(n)=(rand<=0.5);
  end;
  [toto,P] = min(abs(S-1));
  R(i)=P(1);
end;
```

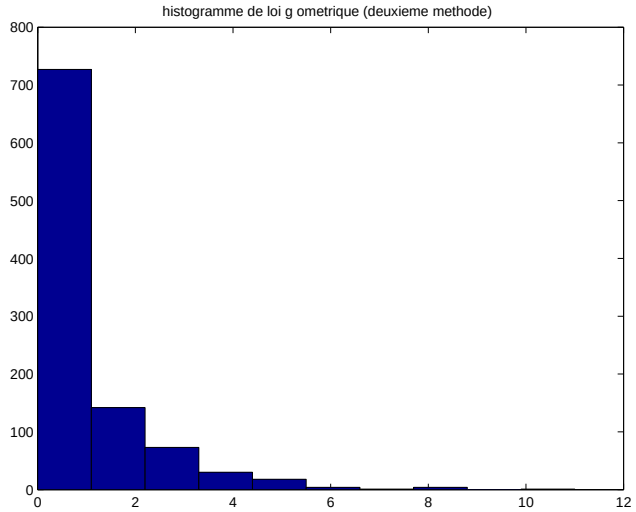
dont l'histogramme est :



– deuxième méthode On cherche à simuler une loi de géometrique à partir de lois exponentielles de paramètre λ .
Pour cela, on garde la partie entière d'une loi exponentielle de paramètre $-log(p)$. :

```
p      = 0.5;
lambda = -log(p);
X      = fix(-log(rand(1,1000))/lambda);
```

on obtient l'histogramme :



1.3.3 Méthode de Box-Muller

– Loi de normale On simule un loi normale grâce à la méthode de Box-Muller : à partir de 2 variables U et V suivant des lois uniformes indépendantes, on obtient un couple de variables X,Y indépendantes suivant des

lois normales :

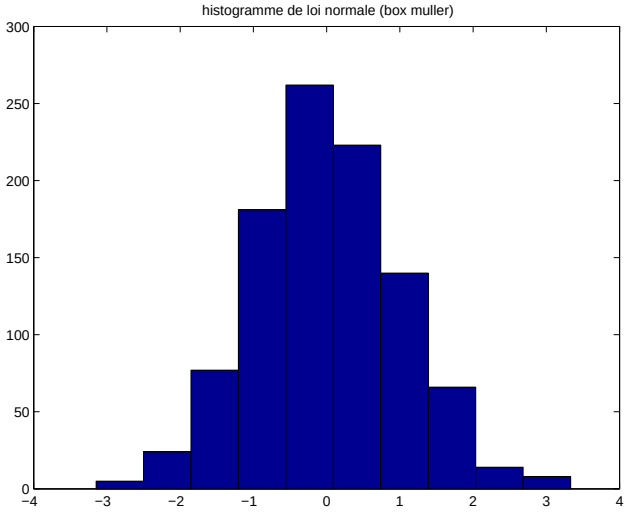
$$X = \sqrt{-2\,log\,U}cos2\pi V$$

$$Y = \sqrt{-2\,log\,U}sin2\pi V$$

Dans notre cas, on ne souhaite simuler qu'une seule variable, donc on peut ne garder que X :

```
X = sqrt(-2*log(rand(1,1000))).*cos(2*pi*rand(1,1000));
```

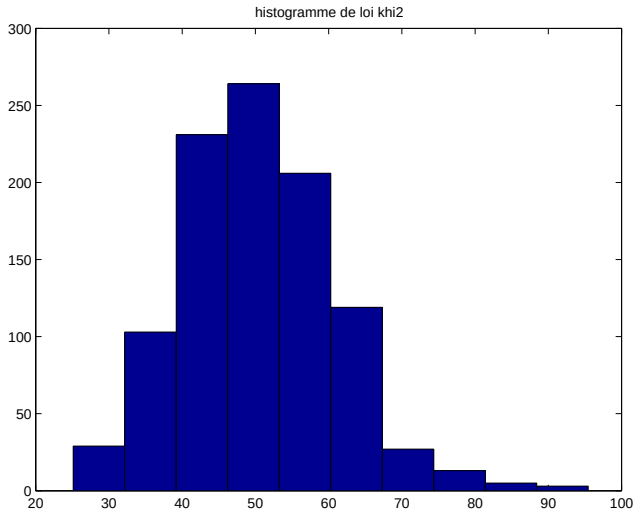
on obtient l'histogramme :



– Loi de χ^2 Une loi de χ^2 est une somme de carrés de lois Normales. On l'obtient donc par le code :

```
X=0;
for i=1:50
  X = X + (sqrt(-2*log(rand(1,1000))).*cos(2*pi*rand(1,1000))).^2;
end;
```

et on obtient l'histogramme :



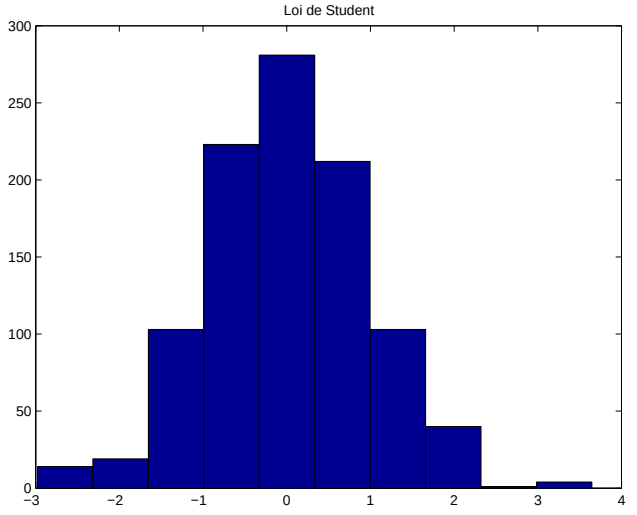
– loi de Student On l'obtient en posant :

$$T = \frac{X}{\sqrt{\frac{Y}{n}}}$$

avec $X \sim \mathcal{N}(0,1)$, $Y \sim \chi^2(n)$ et X indépendant de Y.
Le code s'écrit :

```
Y=0;
for i=1:50
  Y = Y + (sqrt(-2*log(rand(1,1000))).*cos(2*pi*rand(1,1000))).^2;
end;
X=sqrt(-2*log(rand(1,1000))).*sin(2*pi*rand(1,1000))
T = X./sqrt(Y/50);
```

On obtient l'histogramme :



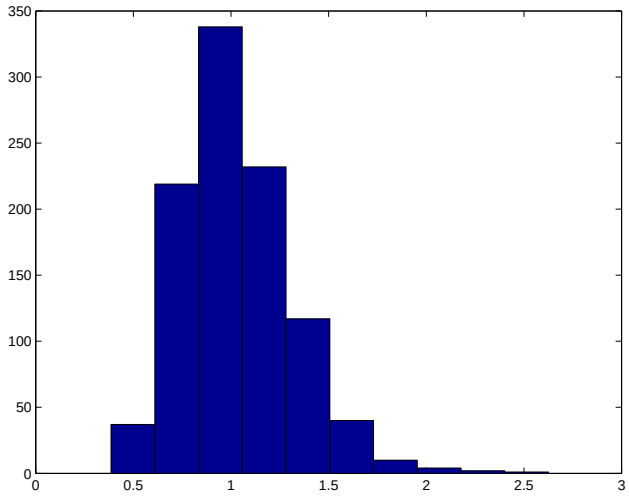
– Loi de Fisher On l'obtient en posant :

$$T = \frac{\frac{X}{n_1}}{fracYn_2}$$

avec $X \sim \chi^2(n_1)$, $Y \sim \chi^2(n_2)$ et X indépendant de Y.
Le code s'écrit :

```
X = 0;
Y = 0;
for i=1:50
  X = X + (sqrt(-2*log(rand(1,1000))).*cos(2*pi*rand(1,1000))).^2;
end;
for i=1:60
  Y = Y + (sqrt(-2*log(rand(1,1000))).*cos(2*pi*rand(1,1000))).^2;
end;

T = (X/50)./(Y/60);
```

On obtient l'histogramme :



2 Problème du voyageur de commerce

On cherche à résoudre le problème du voyageur de commerce par la méthode du recuit simulé.
On choisit au hasard plusieurs points que le voyageur devra visiter par un chemin de distance minimale.
Pour cela, on commence avec un chemin choisi aléatoirement que l'on cherchera à améliorer, par exemple en inversant 2 points choisis aléatoirement. On calcule ainsi la nouvelle distance que le voyageur a à parcourir (sans se servir de tous les points du graphe) et on estime plusieurs cas :
- Soit la nouvelle distance est plus petite : dans ce cas, on accepte le nouveau trajet.
- Soit la nouvelle distance est plus grande : dans ce cas, on accepte le nouveau

trajet avec une probabilité de $e^{-\beta\Delta l}$ et on le refuse avec une probabilité de $1 - e^{-\beta\Delta l}$. Dans la méthode du recuit simulé, on fait varier β à chaque itération : plus β est petit, plus l'entropie est grande et donc plus on pourra se sortir facilement des pièges et donc éviter des minimas locaux. Plus β est grand, on contraire, plus la solution se stabilisera vers la solution optimale trouvée.
On commence donc avec un β faible pour explorer grossièrement tout l'espace des solutions puis on le fera croitre pour aboutir à la solution finale.

Une solution avait été proposée pour la croissance de β en $log(iteration)$, mais cette croissance est beaucoup trop faible et la solution converge trop lentement. On choisit donc une croissance de β en $beta_{n+k} = \beta_n * (1 + \epsilon)$, avec ϵ et k à choisir en fonction du nombre de points à explorer.

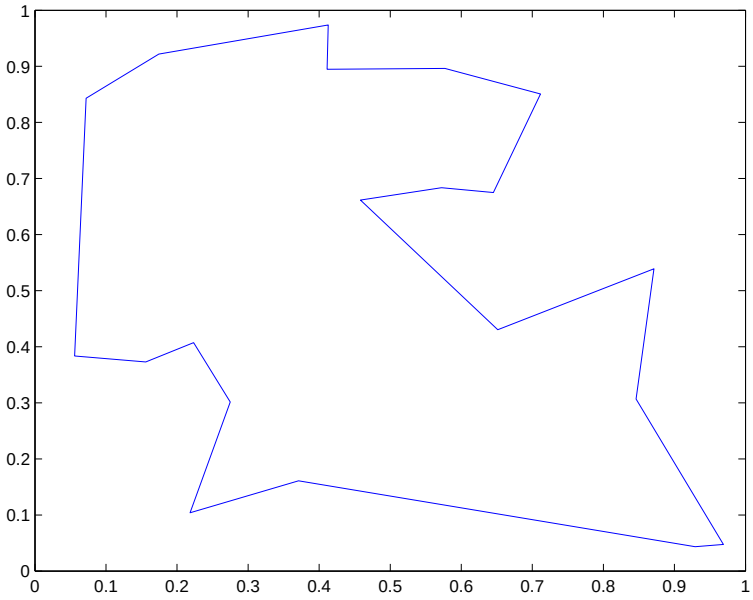
En pratique, pour 20 villes, on choisit $\beta_0 = 0.8$, $\beta_{n+100} = \beta_n * 1.07$. On obtient le code suivant :

```
N = 20;
pts = rand(2,N);
beta = 0.8;
graphe = pts;

graphe_ferme = [graphe graphe(:,1)];
l = 0;

%calcul de la longueur initiale
for i=1:N
  l = l+sqrt((graphe_ferme(1,i+1)-graphe_ferme(1,i))^2
       + (graphe_ferme(2,i+1)-graphe_ferme(2,i))^2);
end;

%pour chaque itération... :
for j=1:10000

%mise à jour de beta
  if (mod(j,100)==0)
    beta = beta*1.07;
  end;


  sauve_graphe = graphe;
  ancienl = l;

%on choisit les indices à inverser
  indice1 = 1;
  indice2 = 1;
  while (indice2==indice1)
    indice1 = floor(rand()*N+1);
    indice2 = floor(rand()*N+1);
  end
```

```
%on calcule la nouvelle distance
  if(indice1==1)
    l = l-sqrt((graphe_ferme(1,indice1+1)-graphe_ferme(1,indice1))^2 ...
               + (graphe_ferme(2,indice1+1)-graphe_ferme(2,indice1))^2)
               - sqrt((graphe_ferme(1,indice1)-graphe_ferme(1,N))^2
               + (graphe_ferme(2,indice1)-graphe_ferme(2,N))^2);
  else
    l = l-sqrt((graphe_ferme(1,indice1+1)-graphe_ferme(1,indice1))^2
               + (graphe_ferme(2,indice1+1)-graphe_ferme(2,indice1))^2)
               - sqrt((graphe_ferme(1,indice1)-graphe_ferme(1,indice1-1))^2
               + (graphe_ferme(2,indice1)-graphe_ferme(2,indice1-1))^2);
  end;


  if(indice2==1)
    l = l-sqrt((graphe_ferme(1,indice2+1)-graphe_ferme(1,indice2))^2 ...
               + (graphe_ferme(2,indice2+1)-graphe_ferme(2,indice2))^2)
               - sqrt((graphe_ferme(1,indice2)-graphe_ferme(1,N))^2
               + (graphe_ferme(2,indice2)-graphe_ferme(2,N))^2);
  else
    l = l-sqrt((graphe_ferme(1,indice2+1)-graphe_ferme(1,indice2))^2
               + (graphe_ferme(2,indice2+1)-graphe_ferme(2,indice2))^2)
               -sqrt((graphe_ferme(1,indice2)-graphe_ferme(1,indice2-1))^2
               + (graphe_ferme(2,indice2)-graphe_ferme(2,indice2-1))^2);
  end;




  %on inverse les deux points
  temp = graphe(:,indice1);
  graphe(:, indice1) = graphe(:, indice2);
  graphe(:, indice2) = temp;


%on ferme le graphe
  graphe_ferme = [graphe graphe(:,1)];


% on finit de calculer la nouvelle distance
    if(indice1==1)
    l = l+sqrt((graphe_ferme(1,indice1+1)-graphe_ferme(1,indice1))^2 ...
               + (graphe_ferme(2,indice1+1)-graphe_ferme(2,indice1))^2)
               + sqrt((graphe_ferme(1,indice1)-graphe_ferme(1,N))^2
               + (graphe_ferme(2,indice1)-graphe_ferme(2,N))^2);
  else
    l = l+sqrt((graphe_ferme(1,indice1+1)-graphe_ferme(1,indice1))^2
               + (graphe_ferme(2,indice1+1)-graphe_ferme(2,indice1))^2)
```

```
              + sqrt((graphe_ferme(1,indice1)-graphe_ferme(1,indice1-1))^2
              + (graphe_ferme(2,indice1)-graphe_ferme(2,indice1-1))^2);
  end;


  if(indice2==1)
    l = l+sqrt((graphe_ferme(1,indice2+1)-graphe_ferme(1,indice2))^2 ...
              + (graphe_ferme(2,indice2+1)-graphe_ferme(2,indice2))^2)
              + sqrt((graphe_ferme(1,indice2)-graphe_ferme(1,N))^2
              + (graphe_ferme(2,indice2)-graphe_ferme(2,N))^2);
  else
    l = l+sqrt((graphe_ferme(1,indice2+1)-graphe_ferme(1,indice2))^2
              + (graphe_ferme(2,indice2+1)-graphe_ferme(2,indice2))^2)
              + sqrt((graphe_ferme(1,indice2)-graphe_ferme(1,indice2-1))^2
              + (graphe_ferme(2,indice2)-graphe_ferme(2,indice2-1))^2);
  end;




  % on valide ou pas le nouveau chemin

  bonl = l;
  deltal = l-ancienl;
  if (deltal>=0)
    if (rand(1,1)>exp(-beta*deltal))
      graphe = sauve_graphe;
      graphe_ferme = [graphe graphe(:,1)];
      bonl = ancienl;
      l = ancienl;
    end;
  end;


  % affichage des resultats

  if (mod(j,10)==0)
    bonl
    pause(0.04);
    plot(graphe_ferme(1,:),graphe_ferme(2,:));
  end;
end
```

On obtient alors le graphe suivant :



A noter que rien ne garantit l'optimalité de ce chemin.

Une deuxième méthode pour explorer l'espace des solutions est d'enlever une étape dans le trajet du voyageur de commerce et de l'inserer ailleurs au lieu d'inverser deux points.

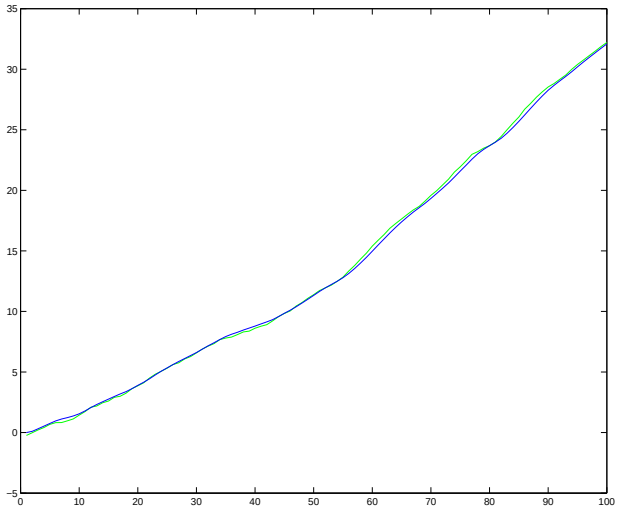
3 Filtre de Kalman

Le but est de donner la meilleure évaluation d'un paramètre (par exemple la position d'un mobile) étant donné une série de mesures bruitées par un bruit gaussien.
On procède par itérations entre des étapes de prédiction et de correction :

```
C = [0 1];
h = 0.1;
A = [1, h; 0,1];
n = 100;
sigma = 1.2;

% matrices de covariances
QW = sigma^2*[h^3/3  h^2/2 ; h^2/2  h];
QV = 0;

m = 0;
V = randn(1,1);
```

```
% Simulation du signal bruité :
X(1,:)=randn(2,1);
Y(1) = 0;
for i=2:n
        N     = randn(2,1);
        X(i,:) =A*X(i-1,:)'+sqrt(QW)*N+m;
        Y(i) =X(i,2);
end;

% Affichage du signal
plot(X(:,1),'g');


% restauration du signal
Xcm(1,:) = [0;0];
Pm=eye(2);

for i = 1:n

% correction
        K = Pm*C'*((C*Pm*C'+QV)^(-1));
        Xc(i,:)=Xcm(i,:)'+K*(Y(i)-C*Xcm(i,:)');
        Pm = (eye(2,2)-K*C)*Pm;


% prédiction :
        Xcm(i+1,:) = A*Xc(i,:)';
        Pm = A*Pm*A' + QW;
end;

% affichage du résultat
hold on;
plot(Xc(:,1));
```

On obtient deux courbes : la simulation (en vert) et le signal retrouvés (en bleu) :



On voit que le signal retrouvé est suffisament proche du signal original.