Filtre de Kalman

1 Introduction

Dans un système de mesure avec un ou plusieurs capteurs, il y a nécessairement du bruit, qui fluctue aléatoirement et brouille plus ou moins le signal. Un filtre de Kalman a pour but de réduire l'imprécision des mesures(mais à condition que le bruit soit de distribution gaussienne).

2 Filtre à moyenne mobile

2.1 Explications théoriques

On considère une expérience très simple : on mesure la masse d'un objet un grand nombre de fois avec la même balance. La balance ne fournit pas toujours le même résultat bien évidemment, elle est un capteur par nature bruité.

Comment gérer cela ? Première idée, nous allons calculer la moyenne car l'on sait que la moyenne est une meilleure estimation de la valeur cherchée que chaque mesure isolée. On utilise les notations suivantes :

- N : nombre total de mesures
- $\widehat{x}_N$: moyenne des N mesures
- z_n : mesure numéro n

Ainsi, la moyenne des N mesures vaut :

$$\widehat{x}_N = \frac{1}{N}\sum_{n=1}^{N} z_n$$

On peut par quelques lignes de calcul écrire :

$$\widehat{x}_N = \frac{1}{N}(\sum_{n=1}^{N-1} z_n + z_N) = \frac{1}{N}\sum_{n=1}^{N-1} z_n + \frac{z_N}{N}$$

$$\widehat{x}_N = \frac{1}{N}\frac{N-1}{N-1}\sum_{n=1}^{N-1} z_n + \frac{z_N}{N}$$

$$\widehat{x}_N = \frac{N-1}{N}(\frac{1}{N-1}\sum_{n=1}^{N-1} z_n) + \frac{z_N}{N}$$

$$\widehat{x}_N = \frac{N-1}{N}\widehat{x}_{N-1} + \frac{z_N}{N}$$

$$\widehat{x}_N = \widehat{x}_{N-1} - \frac{1}{N}\widehat{x}_{N-1} + \frac{z_N}{N}$$

$$\boxed{\widehat{x}_N = \widehat{x}_{N-1} + \frac{1}{N}(z_N - \widehat{x}_{N-1})}$$

On en déduit une équation formelle :

$$\text{Estimation} = \text{Prédiction} + \text{Correction}$$

$$Estimation = \widehat{x}_N$$

$$Prediction = \widehat{x}_{N-1}$$

$$Correction = \frac{1}{N}(z_N - \widehat{x}_{N-1})$$

Il s'agit d'un filtre à moyenne mobile, la moyenne est calculée au fur et à mesure, par récurrence.

L'estimation (nouvelle estimation) est égale à la prédiction (ancienne estimation) plus un terme qui tient compte de la dernière mesure pour faire une correction à la prédiction. Le terme $\frac{1}{N}$ est alors appelé le gain.

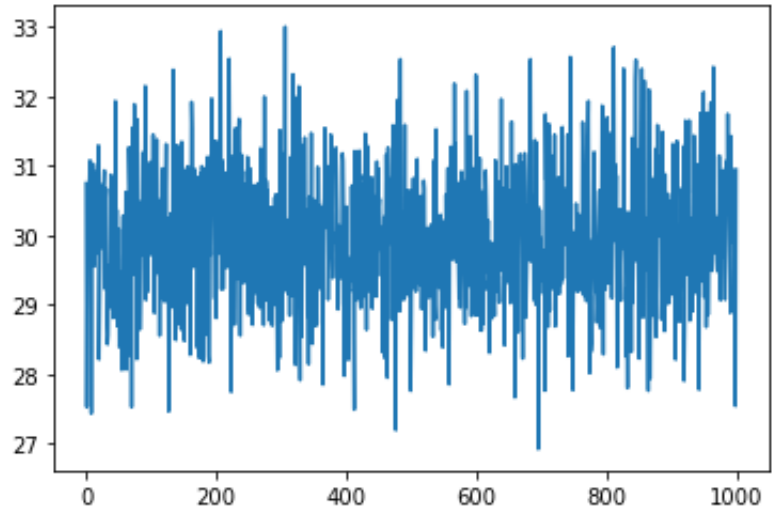
Lorsque N augmente, la dernière mesure finit par être négligeable.
Cette relation encadrée est reliée à un filtre de Kalman, comme on verra plus loin.

2.2 Résultats numériques

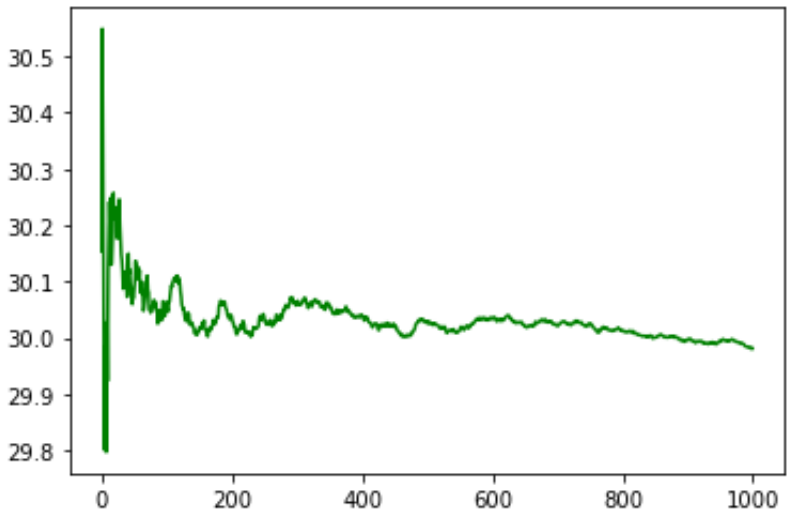
Voir le code python "moving average filter" fourni.

```python
import numpy as np
import matplotlib.pyplot as plt

noise=np.random.normal(0,1,1000)
#plt.plot(noise)

signal=np.full((1,1000),30)+noise
signal=np.squeeze(signal)
#plt.plot(signal)

def moving_avg(signal):
    old_avg=0
    avg=np.zeros((len(signal),1))
    n=1
    for i in range(len(signal)):
        avg[i]=old_avg+(1/n)*(signal[i]-old_avg)
        n=n+1
        old_avg=avg[i]
    return avg

avg=moving_avg(signal)
plt.plot(avg,'g')

```

On suppose connue la valeur recherchée pour tester notre filtre. La valeur est de 30. Ensuite on ajoute un bruit gaussien (distribution normale de moyenne 0 et de variance 1, sur 1000 points) à cette valeur, on obtient ainsi un signal bruité de moyenne 30 dont le bruit est gaussien de variance 1.



On applique ensuite le filtre à moyenne mobile et on retrouve la valeur de 30.



Bien-sûr, le bruit, fort au départ, disparait au fur et à mesure.

3 Filtre g

3.1 Explications théoriques

Ce filtre poursuit la démarche précédente du filtre à moyenne mobile, mais il est moins complexe que le filtre de Kalman aussi. Cependant, il est important à comprendre car il contient les mêmes principes.
On a observé que le filtre à moyenne mobile est un "smoothing filter" ! L'idée pour le filtre g est de changer le facteur $\frac{1}{N}$ dans la formule par un gain noté α. On obtient alors le filtre g.

$$\boxed{\widehat{x}_N = \widehat{x}_{N-1} + \alpha(z_N - \widehat{x}_{N-1})}$$

Ce gain α nous renseigne sur combien l'estimation dépend de la mesure et de combien elle dépend de la prédiction. Comme la prédiction n'est pas parfaite, on a besoin de la mesure pour améliorer l'estimation, et le gain est là pour jouer ce rôle. Dans le filtre g, on utilise la prédiction et les mesures pour obtenir la valeur réelle. Si on sait que la prédiction est meilleure que les mesures on se fie plus à elle, et vice-versa. On a l'équation formelle :

$$\text{Estimation} = \text{Prédiction} + \alpha(\text{Mesure-Prédiction})$$

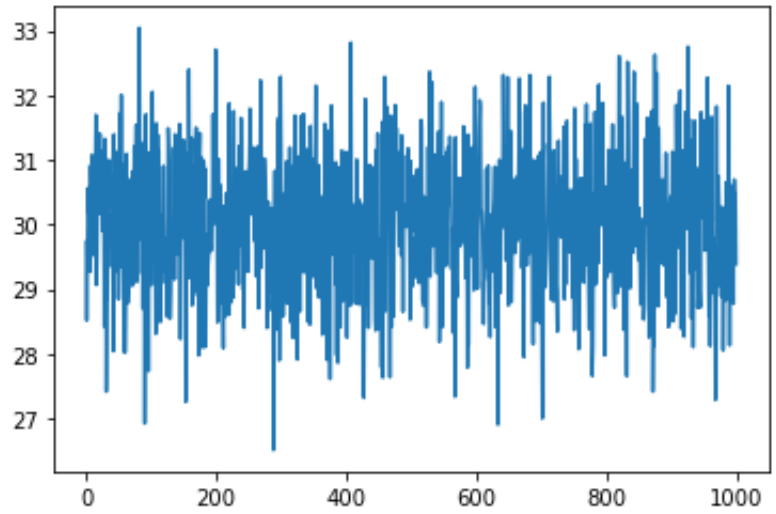
Avec $\alpha \in [0, 1]$
Pour $\alpha = 0$: aucune mesure n'intervient, que la prédiction !
Pour $\alpha = 1$: aucune prédiction n'intervient, que la mesure !

3.2 Résultats numériques

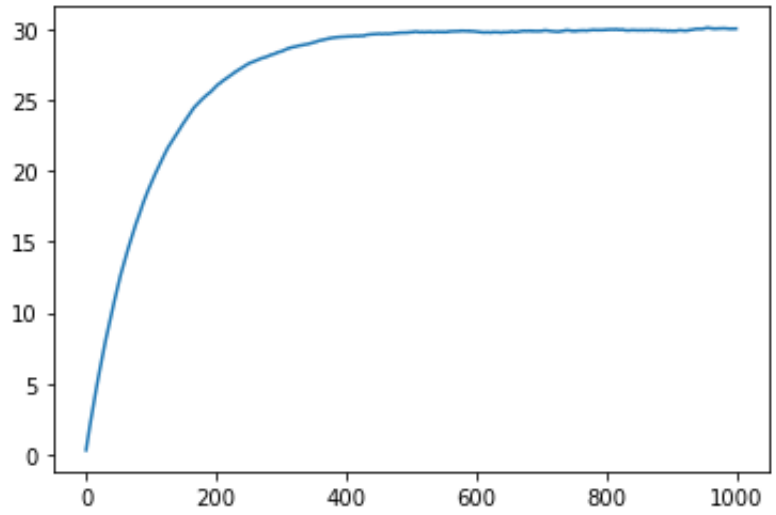
Voir le code python "g filter" fourni.

```python
import numpy as np
import matplotlib.pyplot as plt
noise=np.random.normal(0,1,1000)
signal=np.full((1,1000),30.0)+noise
signal=np.squeeze(signal)
#plt.plot(signal)

def gFilter(signal,alpha):
    prediction=0.0
    estimate=0.0
    out=np.zeros((len(signal),1))
    for i in range(len(signal)):
        prediction=estimate
        estimate=prediction+alpha*(signal[i]-prediction)
        out[i]=estimate
    return out

out=gFilter(signal,0.01)
plt.plot(out)

```

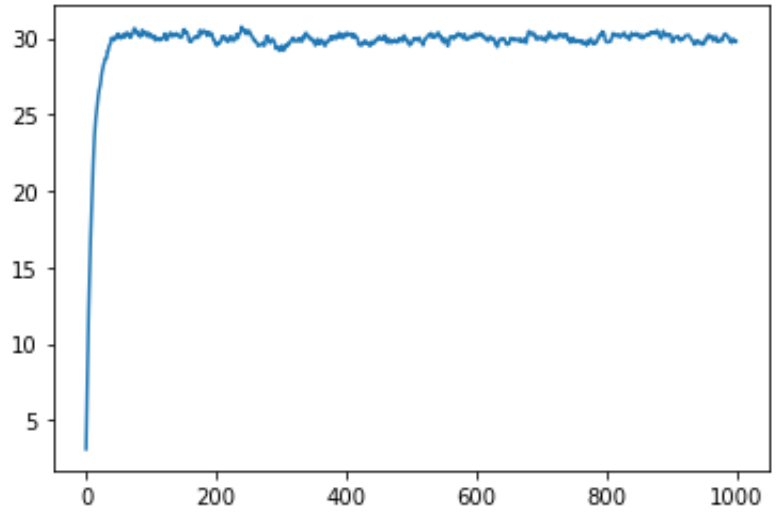
Le signal de départ est du même genre que précédemment :



Et on obtient grâce au filtre g le résultat suivant ($\alpha = 0.01$) :



Ob observe que l'on retrouve bien la valeur de 30 : le but a bien été atteint ! Et surtout, il y a une forte réduction du bruit, la courbe est bien meilleure ! Si on augmente un peu α (par exemple $\alpha = 0.1$), c'est un peu moins bon :



4 Filtre g-h

4.1 Explications théoriques

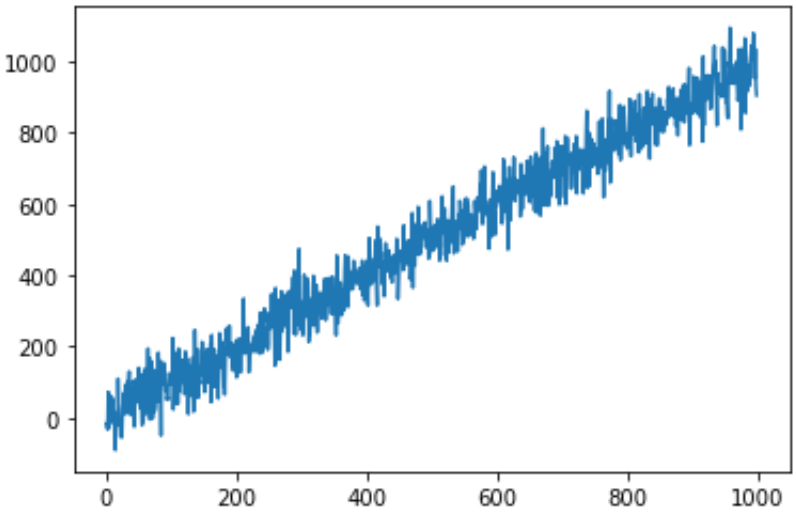
On suppose maintenant que la valeur recherchée peut varier ! Mais, elle est bien entendu toujours bruitée... !

4.2 Résultats numériques

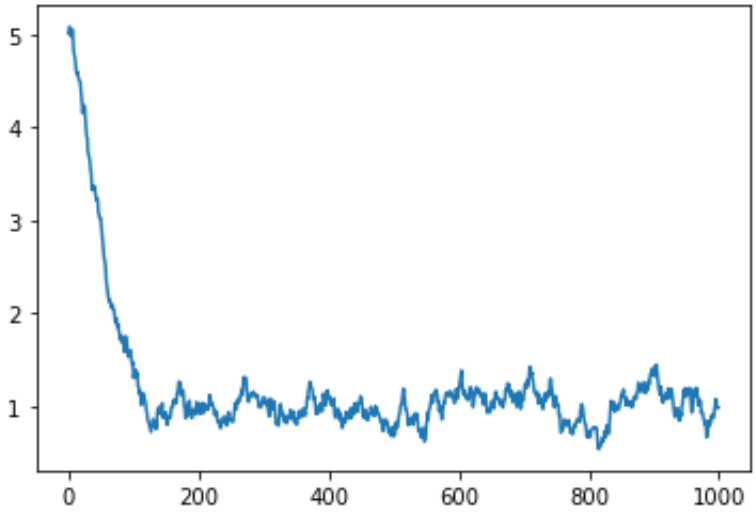
Voici le code python "gh filter" fourni :

```python
import numpy as np
import matplotlib.pyplot as plt
import math as mama
noise=np.random.normal(0,50,1000)
signal=np.linspace(0.0,1000.0,num=1000)

#for i in range(len(signal)):
   #  signal[i]=500*mama.sin(0.05*i)+i


signal=signal+noise
#plt.plot(signal)

def ghFilter(signal,alpha,beta):
    prediction=0.0
    estimate=0.0
    sample_time=1.0
    rate=5
    out=np.zeros((len(signal),1))
    velocity=np.zeros((len(signal),1))
    for i in range(len(signal)):
        #prediction
        rate=rate
        prediction=estimate+rate*sample_time
        #estimation
        rate=rate+beta*(signal[i]-prediction)/sample_
        estimate=prediction+alpha*(signal[i]-predicti
        out[i]=estimate
        velocity[i]=rate
    return out,velocity

out,rate=ghFilter(signal,0.05,0.001)
#plt.plot(rate)
plt.plot(out)
```

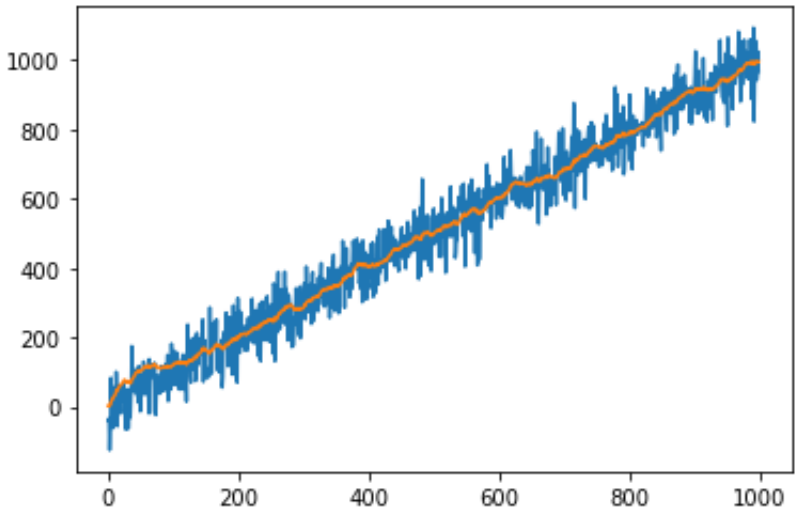
Par exemple, l'allure du signal de départ peut être linéaire de pente 1 avec ajout d'un bruit gaussien :



On obtient alors pour la vitesse (rate) :

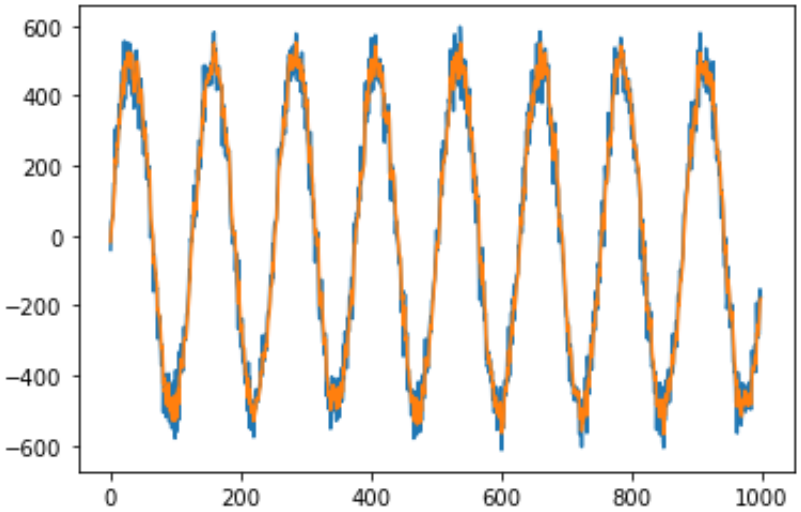


Et pour la position (en orange) supperposée au signal d'origine (en bleu) :



On obtient ainsi une bonne poursuite de trajectoire. Il suffit de faire varier les paramètres α et β pour voir s'améliorer (ou se détériorer !) l'estimation !

Prenons un autre exemple : une trajectoire sinusoïdale bruitée et le résultat du filtre g-h supperposés (on a modifié $\alpha = 0.5$):



Le résultat est encore satisfaisant.

5 Filtre de Kalman

5.1 Explications théoriques

Comment décider quelles valeurs prendre pour α et β ? Le filtre de Kalman répond à cela et calcule automatiquement le gain de façon optimale et automatique. Comme vu déjà, il y a des imprécisions de prédiction et de mesure. L'imprécision est caractérisée par la variance du bruit (on suppose toujours les bruits comme étant gaussiens pour Kalman), qui nous renseigne sur l'amplitude de fluctuation autour de la moyenne.
Dans le filtre de Kalman, on pose :

$$Gain = K = \frac{P}{P + R}$$

avec P : variance de la prédiction et Q: variance de la mesure (ie du capteur, R peut être calculé à partir de la datasheet du capteur). Il est évident que si on ne connait rien la variance P de la prédiction, fixer au départ P assez grand comparée à R. La théorie du filtre de Kalman fournit alors :

$$X = X_P + K(Z - X_P)$$

où X est l'estimation, X_P la prédiction et Z la mesure. Ce n'est pas tout : il faut aussi supposer que P varie au cours du processus selon la loi :

$$P = (1 - K)P_{precedent}$$

ou plutôt si l'on ne veut pas que P devienne nul (car $0 < 1 - K < 1$ donc P diminue au fur et à mesure jusqu'à 0) :

$$P = (1 - K)P_{precedent} + Q$$

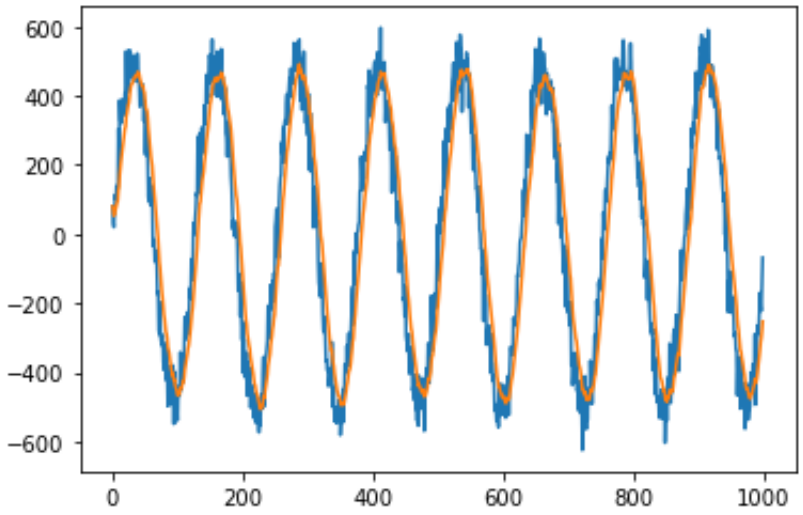
avec Q constant et positif.

5.2 Résultats numériques

Voici le code python "kalman filter" fourni :

```python
import numpy as np
import matplotlib.pyplot as plt
import math as matos
noise=np.random.normal(0,50,1000)

signal=np.linspace(0.0,1000.0,num=1000)+noise

for i in range(len(signal)):
    signal[i]=500*matos.sin(0.05*i)
signal=signal+noise

plt.plot(signal)

def Kalman(signal):
    out=np.zeros((len(signal),1))
    x=signal[0]
    P=50
    R=50
    Q=1
    for i in range(len(signal)):
        x=x
        K=P/(P+R)
        x=x+K*(signal[i]-x)
        P=(1-K)*P+Q
        out[i]=x
    return out

out=Kalman(signal)
plt.plot(out)
```

Le résultat obtenu est meilleur qu'avec le filtre g-h :


600
400
200
0
−200
−400
−600
0
200
400
600
800
1000