

【適応フィルタ (ニュートン法) Adaptive Filter (Newton's Method)】

フィルタ係数 \mathbf{w} (ベクトル) を $\mathbf{w} = \mathbf{w}_{k-1}$ 近傍で、2次まで Taylor 展開すると、

$$J(\mathbf{w}) \approx J(\mathbf{w}_{k-1}) + (\mathbf{w} - \mathbf{w}_{k-1})^T \nabla J(\mathbf{w}_{k-1}) + \frac{1}{2!} (\mathbf{w} - \mathbf{w}_{k-1})^T \nabla^2 J(\mathbf{w}_{k-1}) (\mathbf{w} - \mathbf{w}_{k-1})$$

ここから、

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \nabla J(\mathbf{w}_{k-1}) + \frac{1}{2!} \nabla^2 J(\mathbf{w}_{k-1}) (\mathbf{w} - \mathbf{w}_{k-1}) = \mathbf{0}$$

とすると、フィルタ係数の更新分は、

$$\mathbf{w} - \mathbf{w}_{k-1} = -2[\nabla^2 J(\mathbf{w}_{k-1})]^{-1} \nabla J(\mathbf{w}_{k-1})$$

と書ける。ここで、評価関数が

$$J(\mathbf{w}) = \sigma_d^2 - 2\mathbf{w}^T \sigma_{dx} + \mathbf{w}^T \sigma_x \mathbf{w}$$

で与えられているとする。但し、 $\sigma_d^2 = \mathbb{E}[d_k d_k^T]$, $\sigma_{dx} = \mathbb{E}[d_k \mathbf{x}_k]$, $\sigma_x = \mathbb{E}[\mathbf{x}_k \mathbf{x}_k^T]$ である。
この時、Hesse 行列は、

$$\nabla^2 J(\mathbf{w}_{k-1}) = \sigma_x$$

以上より、フィルタ係数更新アルゴリズムは、次のようになる。

$$\mathbf{w}_k = \mathbf{w}_{k-1} + 2\mu \sigma_x^{-1} (\sigma_{dx} - \sigma_x \mathbf{w}_{k-1}) = (1 - 2\mu) \mathbf{w}_{k-1} + 2\mu (\sigma_x^{-1} \cdot \sigma_{dx}) \quad (\mu: \text{ステップサイズパラメーター})$$

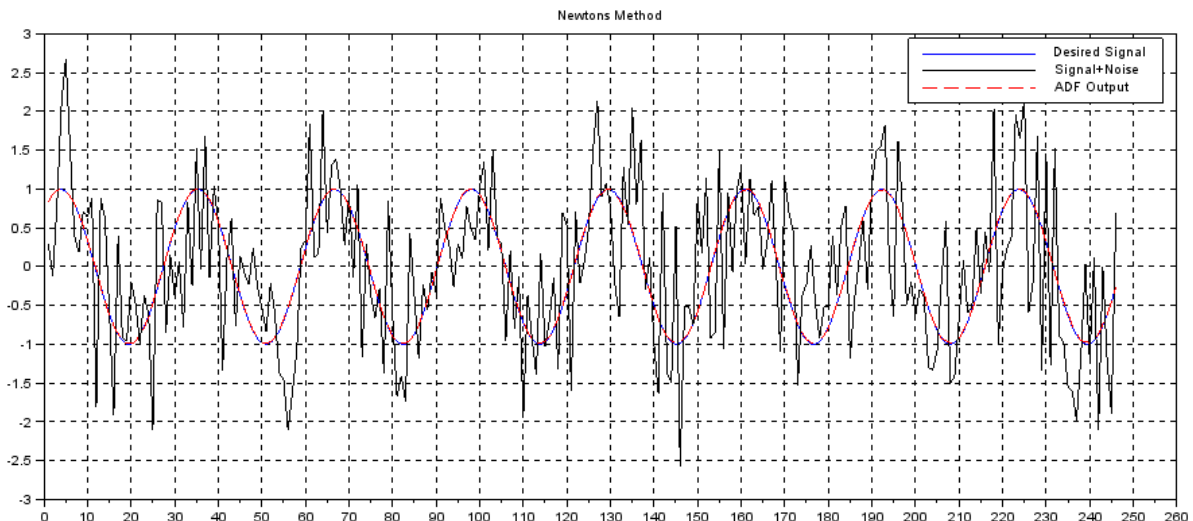


Figure 1: Scilab 実行結果

Source Code 1: Scilab

```

////////////////////////////////////
//      適応フィルタ(ニュートン法)
//      Adaptive Filter(Newton's Method)
//
//                                     M.Tsutsui
////////////////////////////////////

clear all;

```

```

funcprot(0)
function [y_opt]=Newton(myu,Ite_C); //myu:ステップサイズ ,Ite_C:更新回数

    for n=1:Ite_C;
        w_ini=(1-2*myu)*w_ini+2*myu*inv(sigma_x)*sigma_x_d; //フィルター係数更新_loop
    end

    y_opt=w_ini'*x; //係数更新後フィルタ出力

endfunction

Data=50; //データ数
d=sin(1:0.2:Data)'; //入力信号
Data_renew=size(d,1); //データサイズ更新
w_ini=rand(Data_renew,Data_renew); //適応フィルタ初期係数
v=2*rand(Data_renew,1)-2*rand(Data_renew,1); //ノイズ
x=d+v; //適応フィルタ入力

E_x=mean(x); //x平均
E_d=mean(d); //d平均
[L1,L2]=size(x); //サイズチェック
sigma_x=1/L1*(x-E_x*ones(Data_renew,1))'*(x-E_x*ones(Data_renew,1)); //x自己相関
sigma_x_d=1/L1*(x-E_x*ones(Data_renew,1))*(d-E_d*ones(Data_renew,1))'; //x,d相

plot(d); //所望信号
plot(x,'k'); //信号+ノイズ
plot(Newton(0.3,10),'r--'); //適応フィルタ出力
xgrid();
title('Newtons Method');
legend(['Desired Signal','Signal+Noise','ADF Output']);

```

Source Code 2: Python

```

#-----
# Module Name: Adaptive Filter (Newton's Method)
# Author: m_tsutsui
#-----

#Library_Import-----
from numpy import*
import math, numpy as np
import matplotlib.pyplot as plt
#-----

def Newton_m(Ite_C,myu):
    """
    Ite_C:Update count, myu:step size

    """
    global w_ini

    for Ite_C in range(1,Ite_C+1,1): #Filter_Coefficients_Renewal_loop
        w_ini=w_ini*(1-2*myu)+2*myu*1/(sigma_x)*sigma_d_x

    y_opt=w_ini.T*x #ADF_Out

    return y_opt

if __name__ == '__main__':

```

```

Data=50 #Data Number
sita=linspace(0,Data,200)

#-----desired signal-----
d=sin(sita) #Desired_Signal
d_M=matrix(d).T #array->matrix

[Data_renew,Data_renew_p]=d_M.shape #size
ones_block=matrix(np.ones((Data_renew,1)))
L=np.size(d,0) #size

#-----coefficient-----
w_ini=np.random.rand(Data_renew,Data_renew) #ADF_Initial_coefficient

#-----Noise-----
NG=5 #noise gain
v=NG*(np.random.rand(Data_renew,1)-np.random.rand(Data_renew,1)) #Noise
v_M=matrix(v)#array->matrix

x=(d_M+v_M) #ADF_In

#-----Expectation-----
E_d=np.mean(d)#E_d
E_x=np.mean(x)#E_x

#-----covariance-----
sigma_x=np.cov(x.T)#x_covariance
sigma_x_M=matrix(sigma_x)#array->matrix
sigma_d_x=1/L*(x-E_x*ones_block)*(d_M-E_d*ones_block).T #E[dx]

#plot_command-----
plt.figure(facecolor='w')
plt.plot(d)
plt.plot(array(x),'k')
plt.plot(array(Newton_m(5,0.3)),"r--")
plt.grid()
plt.legend(('Desired_Signal','ADF_Input','ADF_Output'))
plt.title('Newtons_Method')
plt.show()

```