北京理工大學
Beijing Institute of Technology

本科实验报告

实验名称：RLS 算法在自适应滤波器中的性能分析

课程名称：	自适应信号处理	实验时间：	2019.04.28
任课教师：	许文龙	实验地点：	
实验教师：	许文龙	实验类型：	■原理验证 □综合设计 □自主创新
学生姓名：	刘仕聪		
学号/班级：	1120161380	第一组：	
学　院　：	信息与电子学院	同组搭档：	
专　业　：	电子信息类（实验班）	成　绩　：	

信息与电子学院
SCHOOL OF INFORMATION AND ELECTRONICS

RLS 算法在自适应滤波器中的性能分析

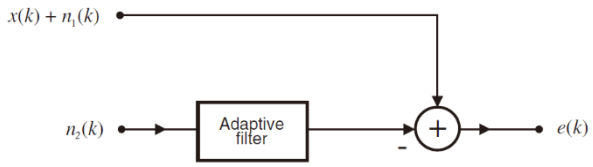
一、 实验目的

1. 完成 RLS 算法的仿真（信号增强）
2. RLS 算法性能分析（包括时变）

二、 实验原理

1. 自适应滤波器信号增强

信号增强是自适应滤波器的一个重要应用。典型原理框图如下



如图所示，一路信号输入加噪信号，另一路输入一个相关噪声。相关噪声是与信号的噪声相关的噪声，我们通过修改滤波器的值，来得到输入信号中含有的噪声的形式，两者相减保留原信号而除去噪声。我们的误差信号可以表示为

$$\begin{aligned} e(k) &= d(k) - y(k) \\ &= x(k) + n_1(k) - \sum_{l=0}^{N} w_l(k) n_2(k-l) \end{aligned}$$

此时均方误差为

$$\xi = E[e^2(k)] = E[x^2(k)] + E\{[n_1(k) - y(k)]^2\}$$

最小均方误差就是输入信号的功率，和其他滤波器不同，这种自适应滤波器的误差信号最优结果就是输入信号。因此最小化的过程也是在求解原信号的过程。

2. RLS 算法

RLS 也是自适应滤波器的一种实现方式。该方法是递归最小二乘方法，目的在于通过选择自适应滤波器的系数，使得观测器件的输出信号在最小二乘意义下与期望信号匹配。其最小化过程需要利用到此刻为止的所有信号，而且在算法后期对于信号的变化可能不敏感。对于某个指定时刻，我们的输入仍然和前面一样，但是目标函数固定为

$$\xi^d(k)=\sum_{i=0}^{k}\lambda^{k-i}\varepsilon^2(i)=\sum_{i=0}^{k}\lambda^{k-i}[d(i)-\boldsymbol{x}^{\boldsymbol{T}}(i)\boldsymbol{w}(k)]^2$$

注意每一次迭代过程中的所有系数都是相同的，此处误差采用的是后验误差。由于实现也是基于 FIR 的，但是估计器结构稍显复杂，这里不单独列出框图。

RLS 算法的一个优势在于，采用简化算法求解矩阵的逆，这样可以有效加快运算。其系数更新有两种方法，但是性能是相同的。

3. 有色噪声

一般认为，白噪声是在频谱中呈现为均匀分布的噪声，与时域分布无关。色噪声在频域就不是均匀分布的，我们认为有色噪声可以通过白噪声通过一阶全极点滤波器得到，例如传递函数为

$$H(z)=\frac{z}{z-a}$$

色噪声可以看做是一个 AR 信号，是白噪声信号通过自回归模型得到的。在本次的仿真中，我们令白噪声通过一个二阶全极点滤波器，这时结果并不是色噪声，相比之下更为复杂一些。我们可以将随机信号看作是这样类型的模型。

4. 非平稳问题性能分析

实际问题往往都是非平稳问题，对于可变的目标响应，此时滤波器系数的滞后会带来之后超量误差，为了计算超量误差，我们假设最优系数向量的每一个元素都可以用一个马尔科夫过程来描述，这种非平稳情形可以看做是一种简化。

$$\boldsymbol{w}(k)=\lambda_{\boldsymbol{w}}\boldsymbol{w}(k-1)+\boldsymbol{n}_{\boldsymbol{w}}(k)$$

通过一些巧妙的数学方法，我们能推导出超量误差等参数的近似表达式。

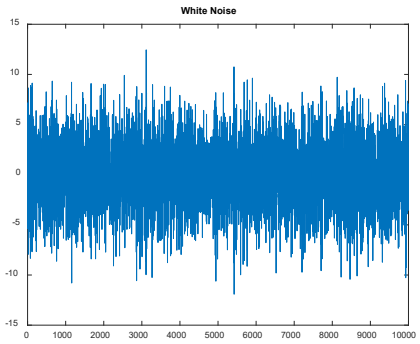
三、 实验内容

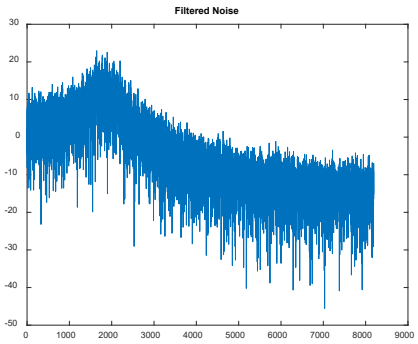
a. 噪声信号的生成

例题中假设信号是白噪声通过滤波器。其中滤波器传递函数为

$$H(z)=\frac{0.4}{z^2-1.36z+0.79}$$

此处首先给出噪声的功率谱。白噪声与通过滤波器后的噪声功率谱如下：

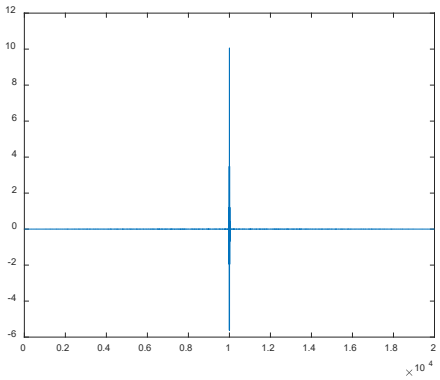




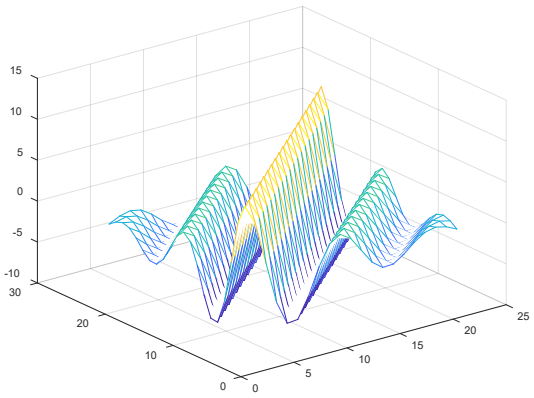
根据图中的结果，我们容易知道，噪声分布不是均匀的。我们简单看做噪声通过滤波器后，某一频率的噪声显著增强，以此为中心噪声功率随频率下降。

则此时信号的相关矩阵不能再简单地认为是理论结果可以得到的了，我们此时应该计算相关矩阵。对于一个信号增强的系统而言，此时的输入信号是经过滤波器后的噪声，而被污染信号是白噪声污染的信号。对于输入信号而言，我们可以通过计算的方法得到。

首先绘制，通过 10000 次仿真的平均结果，我们得到相关函数的图像如下。

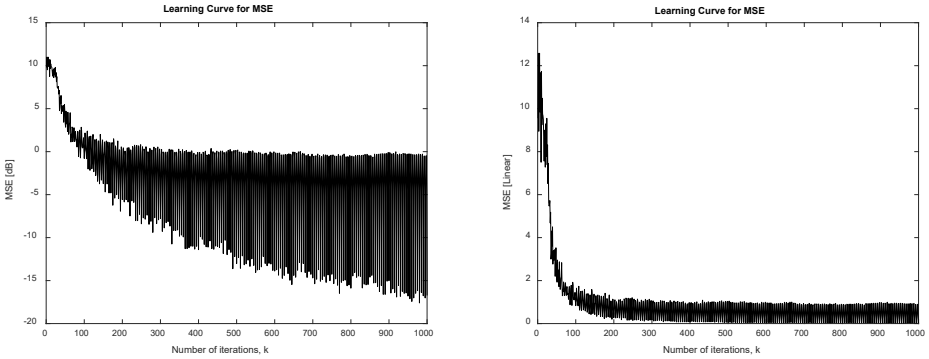


根据这个结果，根据题目要求 20 阶滤波器，我们可以计算相关矩阵。由于写出较为复杂，因此此处采用 meshgrid 方法显示如下



如上所示为相关矩阵。对该相关矩阵进行特征值分解得到，最大特征值约为 10，特征值扩展约为 345。若采用 LMS 算法，此时的收敛速度将十分缓慢，但是此处采用了 RLS 算法，可以看出**收敛速度明显加快**。其运算结果如图所示。

实验中采用 100 次过程进行平均。

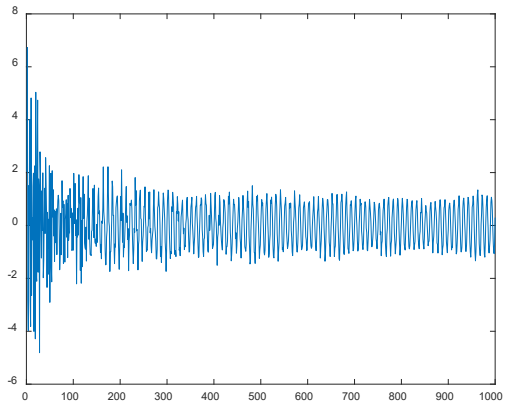


图中所示的结果分别是对数标尺下的 MSE 和线性标尺下的 MSE。可以看出收敛很快，而且 MSE 达到了较低水平。这里采用 MSE 的说法实际上不准确，应当是代价函数的取值。由于误差函数在该问题上就是原信号，因此会出现一定的抖动。

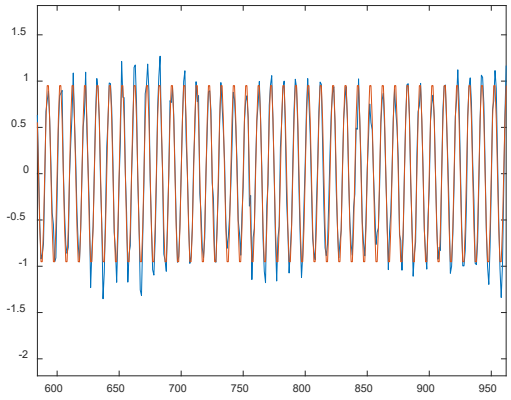
在 LMS 算法的仿真中我们知道，特征值扩展的取值越大，就会导致收敛速度的下降，但是此处采用的 RLS 算法显然没有产生明显的影响。

b. 信号增强结果分析

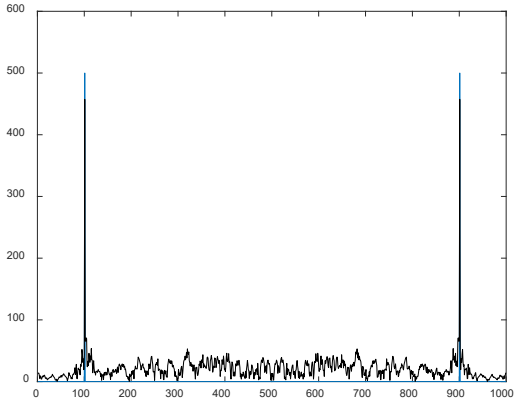
信号增强时，误差函数就是原信号。我们绘制误差信号得到



这时看似右侧已经成为了我们期望的结果。我们再将原信号绘制得到



那么显然我们已经成功增强了信号。通过图形我们知道，随着收敛的进行，增强效果变得更好。最后使用傅里叶变换分析结果，得到



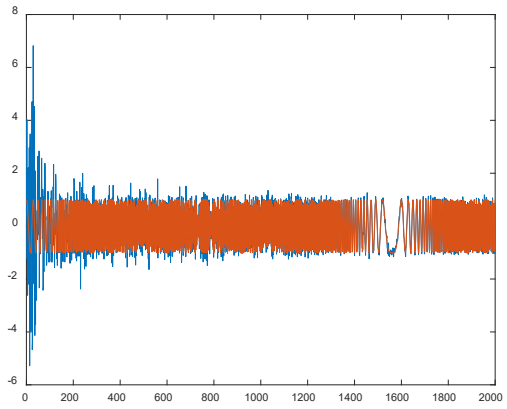
图中蓝色部分是原信号的傅里叶变换，黑色部分是误差信号的傅里叶变换，可见信号增强效果很好，显然可以从噪声中检测出该信号。通过对增强结果求解方差得到，$\sigma_e^2 = 0.9602$，该结果与原信号功率也十分接近。

c. 假设信号时变。

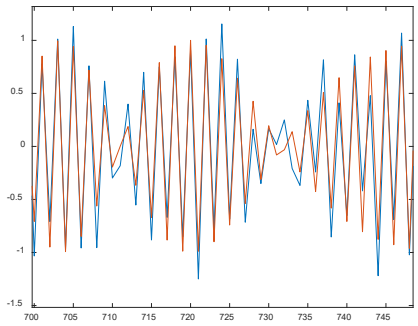
我们的假设是，待增强信号是一个变化的信号，例如 chirp 信号。此时可以表示为

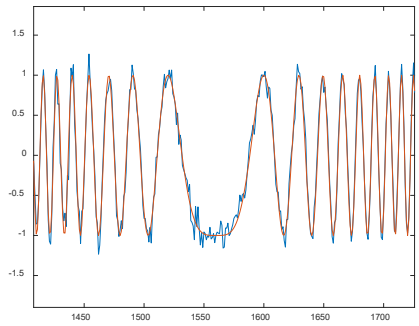
$$x(t) = e^{j2\pi\left(f_0 t + \frac{1}{2}ut^2\right)}$$

以一定的采样频率数字化以后，带入仿真得到

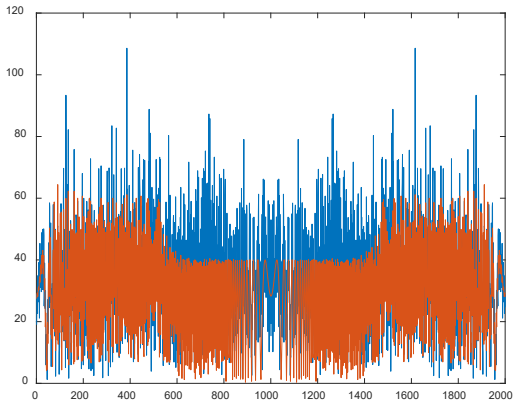


放大部分细节可以知道

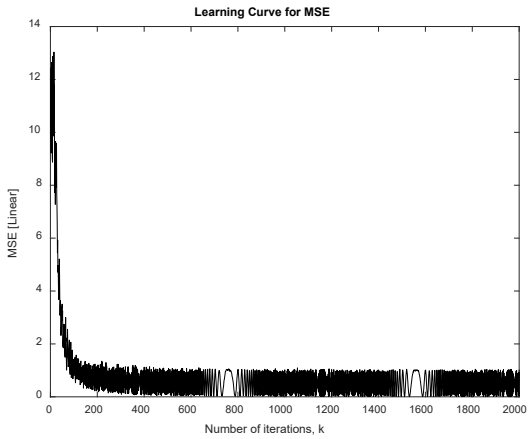




显然增强效果很好。但是从频谱分析上不能得到什么有效信息。



此处给出后验误差，以示收敛效果。对于时变信号而言，RLS 算法有着较好的效果。

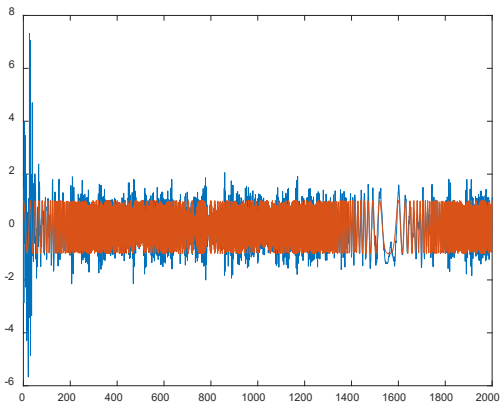


线性标尺后验误差

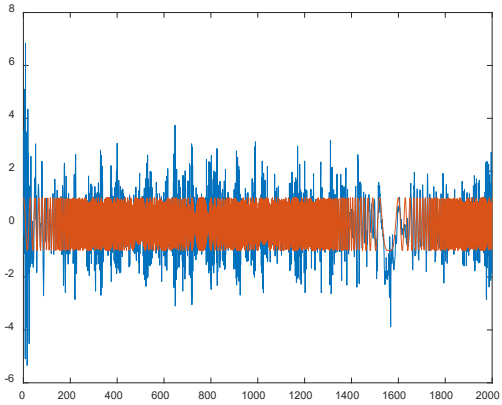
四、 问题与思考

1. 没有能够在理论上分析输入噪声的自相关函数。实际上自相关函数通过指定数字滤波器后的自相关函数应该能够推导，但是这里没能推导出该结果。实际上应是一条类似于 sinc 函数的结果。

2. 上面实际上没有探究不同的λ带来的区别。下面给出，当取非 1 值时，结果为



肉眼可见的直观结果是，增强效果变差了。此时$\lambda = 0.97$。当取更小的λ时，效果还会更差，下图中$\lambda = 0.9$。



推测是因为时变信号较为复杂，需要利用到之前的信息才能更好地进行增强。

3. 通过《Reinforcement Learning》一书的第一章，我们知道实际上 RLS 的加权因子（或者遗忘因子）是可以写成一个函数关系的，只要可以满足收敛的条件即可。例如常用的高斯加权等，都可以在此处进行尝试。由于时间与篇幅问题，在这里没有单独进行仿真。

五、 附录

本次仿真的代码如下

```
%%
%   Definitions:
sigma_nr2=10;
B=[0.4];
A=[1 -1.36 0.79];
L=2000;
nr=sqrt(sigma_nr2)*randn(1,L);
n=filter(B,A,nr);
k=1:L;
x=sin(0.2*pi*k);
r=x+nr;
ensemble    = 100;                          % number of
realizations within the ensemble
K           = L;                           % number of iterations
% H           = [0.32+0.21*j,-0.3+0.7*j,0.5-0.8*j,0.2+0.5*j].';
% Wo          = H;                            % unknown system
% sigma_n2    = 0.04;                          % noise power
N           = 21;                           % number of
coefficients of the adaptive filter
delta       = 0.1;                          % small positive
constant (used to initialize the
```

```
                                        % estimate of the
inverse of the autocorrelation
                                        % matrix)
lambda      = 1.0;                      % forgetting factor


%   Initializing & Allocating memory:
W       = zeros(N,(K+1),ensemble);  % coefficient vector for
each iteration and realization
MSE     = zeros(K,ensemble);        % MSE for each realization
MSEPost = zeros(K,ensemble);        % MSE a posteriori for each
realization
MSEmin  = zeros(K,ensemble);        % MSE_min for each
realization


%   Computing:
for l=1:ensemble,

    X       = zeros(N,1);               % input at a certain
iteration (tapped delay line)
    d       = zeros(1,K);               % desired signal
    nr      = sqrt(sigma_nr2)*randn(L,1);

    x       = filter(B,A,nr);           % Creating the input
signal
                                                          %
(normalized)

    % sigma_x2 = var(x);                                    %
signal power = 1
    % n        =
sqrt(sigma_n2/2)*(randn(K,1)+j*randn(K,1));     % complex
noise
    k=1:L;
    t = [1:L]/40000;
    f0 = 1000;
    u = 10^6;
    yy = real(exp(1j*2*pi*(f0*t+1/2*u*t.*t)));
    dd=yy+transpose(nr);

    for k=1:K,

        X       =   [x(k,1)
```

```
                    X(1:(N-1),1)];              % input signal
(tapped delay line)

        d(k)    =   dd(k);          % desired signal

    end

    S   =   struct('filterOrderNo',(N-
1),'delta',delta,'lambda',lambda);
    [y,e,W(:,:,l),yPost,ePost]  =   RLS(d,transpose(x),S);

    MSE(:,l)    =   MSE(:,l)+(abs(e(:,1))).^2;
    MSEPost(:,l)=   MSEPost(:,l)+(abs(ePost(:,1))).^2;
    MSEmin(:,l) =   MSEmin(:,l)+(abs(n(:))).^2;

end


%   Averaging:
W_av        = sum(W,3)/ensemble;
MSE_av      = sum(MSE,2)/ensemble;
MSEPost_av  = sum(MSEPost,2)/ensemble;
MSEmin_av   = sum(MSEmin,2)/ensemble;


%   Plotting:
figure,
plot(1:K,10*log10(MSE_av),'-k');
title('Learning Curve for MSE');
xlabel('Number of iterations, k'); ylabel('MSE [dB]');

figure,
plot(1:K,MSE_av,'-k');
title('Learning Curve for MSE');
xlabel('Number of iterations, k'); ylabel('MSE [Linear]');

figure,
plot(1:K,10*log10(MSEPost_av),'-k');
title('Learning Curve for MSE(a posteriori)');
xlabel('Number of iterations, k'); ylabel('MSE(a posteriori)
[dB]');

figure,
plot(1:K,MSEPost_av,'-k');
```

```
title('Learning Curve for MSE(a posteriori)');
xlabel('Number of iterations, k'); ylabel('MSE(a posteriori)
[Linear]');

figure,
plot(1:K,10*log10(MSEmin_av),'-k');
title('Learning Curve for MSEmin');
xlabel('Number of iterations, k'); ylabel('MSEmin [dB]');

figure,
subplot 211, plot(real(W_av(1,:))),...
title('Evolution of the 1st coefficient (real part)');
xlabel('Number of iterations, k'); ylabel('Coefficient');
subplot 212, plot(imag(W_av(1,:))),...
title('Evolution of the 1st coefficient (imaginary part)');
xlabel('Number of iterations, k'); ylabel('Coefficient');

figure
plot(e)
hold on
k=1:L;
plot(yy)
```

此部分代码是时变信号的代码分析，可以稍作修改得到非时变效果。两篇实验报告的结果均在个人 GitHub 上开源，地址为
https://github.com/psycholsc/Adaptive-Filtering