Table of Contents

Filtragem de Overshoot de Chaveamento

Amplificadores Ópticos a Semicondutores apresentam *overshoot* e oscilações transientes quando utilizados como Chaves eletro-ópticas. Este projeto analiza a influência de filtros ótimos e adaptativos no tempo de subida e estabilização de SOAs. Caso haja sucesso, a filtragem irá diminuir o tempo de chaveamento trazendo maior estabilidade, maior eficiência energética, maior velocidade e desempenho.

```
clear all; clc; close all; tic
```

Configurando os Parâmetros e Arquivos

```
tim = 0.32; % pre-impulse duration (ns) / pisic 4
imp = 1;    % pre-impulse voltage
deg = 1.25; % step voltage
cur = 0.07; % bias current

root_dir = 'C:/Users/Bruno/Documents/Projetos Colaborativos/';
% root_dir = uigetdir;
addpath([root_dir 'MatLab/']);
addpath([root_dir 'SOAs']);
addpath('functions');

strsoa = [root_dir 'SOAs/InPhenix-1503/optical/'];  % Diretório de Origem
strend = '-mod500mV-pinpd-12dbm-pinsoa-9dbm';       % Nome dos arqs.
strpath = [strsoa 'pisic-4/'];                      % Diretório de Destino
strsave = 'results-pisic-';                         % Técnica
str_aux = 'i%1.3fA-t%1.2fns-deg%1.2fV-imp%1.2fV';   % Medida

warning off MATLAB:MKDIR:DirectoryExists
mkdir([strpath 'script/csv/' sprintf('%2.0fmA',cur*1e3) '/']);
mkdir([strpath 'script/figs/' sprintf('%2.0fmA',cur*1e3) '/']);

aux1 = sprintf(str_aux,cur,tim,deg,imp);
aux2 = [strpath 'dados/' sprintf('%2.0fmA',cur*1e3) '/'];
strcall = ['pisic-' aux1 strend];
```

Leitura da Medição

O osciloscópio salva os arquivos **.h5** em valores normalizados, registrando o coeficiente de multiplicação no cabeçalho. xInc e xOrg são os valores de incremento e offset, respectivamente, que determinam os valores absolutos das medidas.

```
nome_arq = [aux2 strcall '.h5'];
end_arq = '/Waveforms/Channel 1/';
y = (double(h5read(nome_arq,[end_arq 'Channel 1Data'])) * ...
    h5readatt(nome_arq,end_arq,'YInc')+h5readatt(nome_arq,end_arq,'YOrg'));
t = h5readatt(nome_arq,end_arq,'XOrg'):h5readatt(nome_arq,end_arq,'XInc'):(length(
    h5readatt(nome_arq,end_arq,'XInc')-abs(h5readatt(nome_arq,end_arq,'XOrg'));
t = (t-t(1))';          % Desloca o tempo para iniciar em zero
med = mean(y);          % Ponto médio
[~, maxy] = max(y);     % Ponto de máximo global
difc = [0; diff(y)];                % Derivada de y
diff_scale = (max(y)-min(y))/10;    % Escala para o redimens. da derivada
difc = difc/max(difc)*diff_scale/2-diff_scale;  % Redimens. da derivada
```

Curva de Chaveamento

Primeiramente, definem-se as características do chaveamento, como frequência, número de amostras por ciclo (*samples*), entre outros. Logo após, aplica-se uma filtragem no sinal para que posteriormente a definição dos momentos de abertura e fechamento da chave sejam mais simples.

A filtragem apresenta um atraso em relação ao sinal original, portanto faz-se a correção desse atraso utilizando a correlação cruzada. Por fim, são definidos como 70% e 30%, respectivamente, os níveis que definem chave fechada e aberta.

```
pulselength = 8e-9;
ts = t(2) - t(1);      % Tempo de amostragem
fs = 1/ts;              % Freq. de amostragem
fmod = 6.985e9;     % Freq. modulação (est. inicial)
samples = round(pulselength*2/ts);
% _Samples_ representa número de amostras durante um ciclo de chaveamento.
spbit = round(1/fmod/(t(2)-t(1)));
% _Samples per bit_. Número de amostras por símbolo modulado.

fp1 = 1e9;       % Borda da Banda de Passagem
fp2 = 2e9;       % Borda da Banda de Rejeição
Rp = 0.5;         % Ripple na Banda de Passagem
As = 1000;      % Att na Banda de Rejeição
[B,~] = filter_lp(2*pi*fp1/(fs),...
    2*pi*fp2/(fs),Rp,As,'Bar');
yf = conv(B,y);
yfxc = xcorr(y,yf); % Correlação cruzada
[~,i] = max(yfxc); lag = length(y)-i;
yf = yf(1-lag:end+lag);
y_unb = y - yf; % Sinal y subtraido de yf
% Resultado é y ao redor de zero, "unbiased".
clear fp1 fp2 Rp As B yfxc i lag aux1 aux2
```

```
% [y, t, yf, y_unb] = switch_file_import(tim, imp, deg, cur);

[~,p_maxyf] = max(yf); [~,p_minyf] = min(yf);
rise_thresh = (max(yf) - min(yf)) / 10*7 + min(yf);   % 70% da subida (em y)
fall_thresh = (max(yf) - min(yf)) / 10*3 + min(yf);   % 30% da subida
```

Segmentação e amostragem dos Ciclos de Chaveamento

Cada ciclo de chaveamento deve ser isolado para processamento individual, para que se evite a acumulação de estatísticas inválidas para o chaveamento usual.

O ponto *riseedge* representa a borda de subida da chave, e é encontrado utilizando o nível de subida definido anteriormente em *rise thresh*. A borda de descida é determinada de forma similar. No gráfico, a seção da curva mais escura denota o período do qual o sinal encontra-se chaveado.

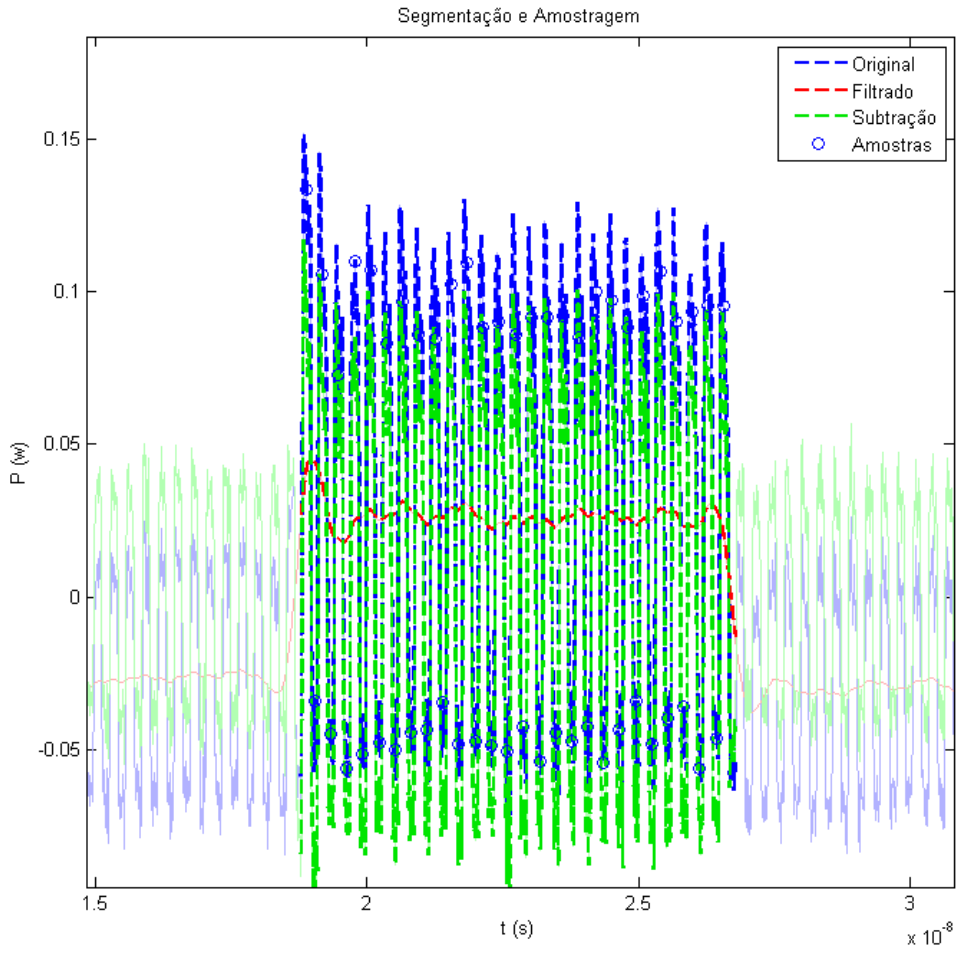
Antes de entrar no *loop* de iterações, define-se o número total de ciclos de chaveamento, segmenta-se o primeiro intervalo e prepara-se a representação gráfica para sua atualização iterativa.

Após a definição dos pontos de fechamento e abertura da chave, é feita a amostragem do sinal, guardando os pontos amostrados em um novo vetor, que será utilizado nas seções seguintes. No gráfico, os pontos amostrados são apresentados em forma de pequenos círculos.

```
if(yf(samples)<rise_thresh)     % Condição de chave aberta no ponto _samples_.
    riseedge = find(yf(samples:samples*10)>rise_thresh,1,'first')+samples;
    falledge = find(yf(riseedge:samples*10)<fall_thresh,1,'first')+riseedge;
else    % Condição de chave fechada no ponto _samples_.
    falledge = find(yf(samples:samples*10)<fall_thresh,1,'first')+samples;
%     riseedge = find(yf(falledge:samples*10)>rise_thresh,1,'first')+falledge;
end

% Intervalo do primeiro ciclo de chaveamento
interv1 = falledge-round(3*samples/4); interv2 = interv1 + samples - 1; i12 = inte
N_cycles = floor(length(y(interv1:end)) / samples/2); % Numero de ciclos de chavea
Ns_cy = zeros(N_cycles,1);

% Alocação de memória
y_cycle = cell(N_cycles,1); yf_cycle = cell(N_cycles,1);
yub_cycle = cell(N_cycles,1); ys_c = cell(N_cycles, 1);

y_cycle{1} = y(i12); y_cur = y_cycle{1};
yf_cycle{1} = yf(i12); yf_cur = yf_cycle{1};
yub_cycle{1} = y_unb(i12);
yub_cur = yub_cycle{1};
tcrop = t(i12);
hsm = zeros(length(i12),1);
riseedge = find(yf_cur>rise_thresh,1,'first');
falledge = find(yf_cur(riseedge:end) < fall_thresh,1,'first')+riseedge;
sw_t = riseedge:falledge;
y_cur = y_cur(sw_t); yf_cur = yf_cur(sw_t);
yub_cur = yub_cur(sw_t);
```

```
set(0,'DefaultFigureWindowStyle','docked')
fig(1) = figure('name','Amostragem'); h1 = plot(tcrop,y_cycle{1},'Color',[.7,.7,1]
hold on, h2 = plot(tcrop,yf_cycle{1},'Color',[1,.7,.7]);
h3 = plot(tcrop,yub_cycle{1},'Color',[.7,1,.7]);
h4 = plot(tcrop(sw_t),y_cur,'--'); set(h4,'Color','b','LineWidth',2)
h5 = plot(tcrop(sw_t),yf_cur,'--'); set(h5,'Color','r','LineWidth',2)
h6 = plot(tcrop(sw_t),yub_cur,'--'); set(h6,'Color',[0, .9, 0],'LineWidth',2)
h7 = plot(1:length(hsm),hsm,'o');
xlim([t(interv1) t(interv2)]), ylim([min(y) max(y)]); xlabel('t (s)'), ylabel('P (
legend([h4 h5 h6 h7],'Original','Filtrado','Subtração','Amostras')
title('Segmentação e Amostragem');
% _Loop_ de iterações para segmentação e apresentação.
for i=1:N_cycles
    start_interv = interv1 + (i-1)*samples;
    end_interv = start_interv + samples - 1;
    interval = start_interv:end_interv;
    y_cur = y(interval); yf_cur = yf(interval);
    yub_cur = y_unb(interval);

    riseedge = find(yf_cur>rise_thresh,1,'first');
    falledge = find(yf_cur(riseedge:end) < fall_thresh,1,'first')+riseedge;
    sw_t = riseedge:falledge;

    tcrop_cur = tcrop(sw_t);
    y_cycle{i} = y_cur(sw_t);
    yf_cycle{i} = yf_cur(sw_t);
    yub_cycle{i} = yub_cur(sw_t);

    p_hsm = cycle_samp(y_cycle{i},tcrop_cur);
    p_hsm = riseedge + p_hsm - 1;
    y_s = y_cur(p_hsm); t_s = tcrop(p_hsm);
    ys_c{i} = [t_s, y_s];
    Ns_cy(i) = length(y_s);

    plot_update(h1,y_cur,h2,yf_cur,h3,yub_cur,...
        h4, [tcrop_cur, y_cycle{i}], h5, [tcrop_cur, yf_cycle{i}],...
        h6, [tcrop_cur yub_cycle{i}], h7, [t_s, y_s])
    drawnow
end

clear interv1 start_interv end_interv interval interv2 interv12 handle1 handle2 fa
    riseedge i12 tcrop tcrop_cur y_cur yf_cur yub_cur sw_t hsm p_hsm y_s t_s sw_t

toc
```

Elapsed time is 14.467153 seconds.



Filtragem de Wiener

Nesta seção, utilizam-se os pontos amostrados na seção anterior numa abordagem clássica de filtragem de sinais digitais. As variáveis utilizadas são o sinal amostrado original e a decisão (valor esperado). Com isso, é feita a filtragem de Wiener utilizando a estatística de cada semicíclo de chaveamento.

O primeiro gráfico apresentado mostra os valores discretos das amostras originais, a decisão, e o valor após a filtragem. O segundo gráfico encontra o erro sem, e com a utilização do filtro de Wiener.

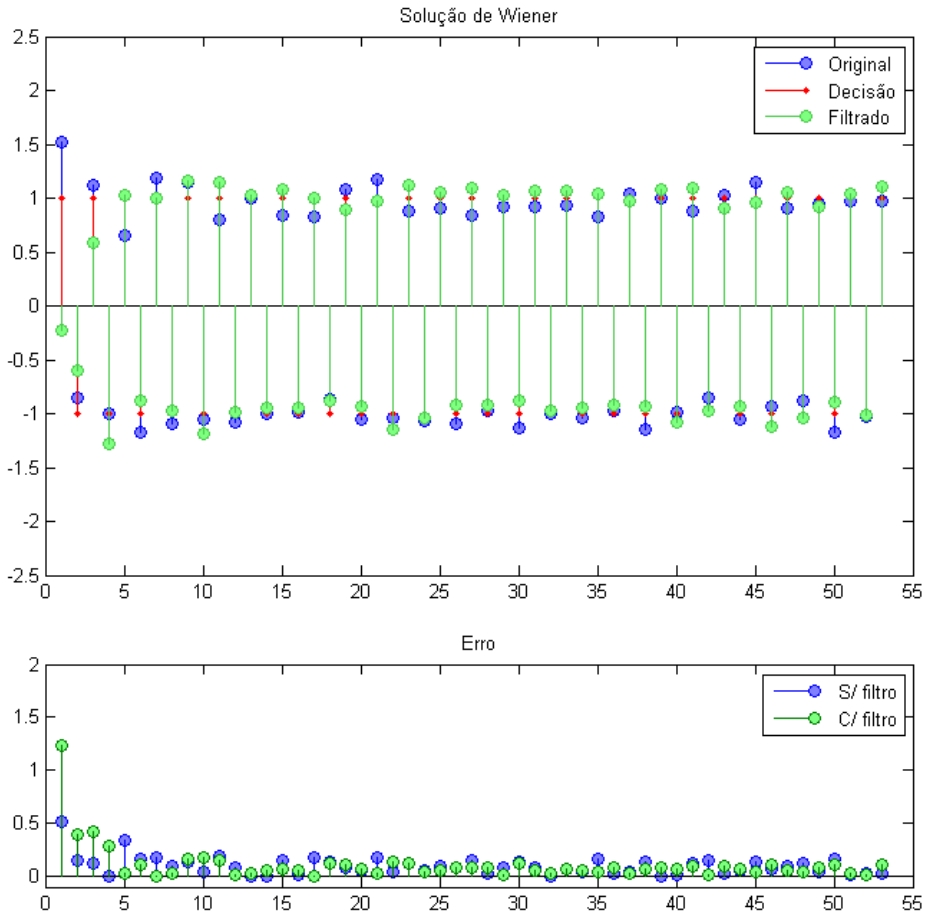
É possível observar que, após a estabilização do *overshoot*, o erro utilizando o filtro ótimo é aceitável. Entretanto, durante as oscilações do *overshoot* o filtro provoca erro ainda maior.

O filtro de Wiener utiliza as estatísticas presentes no sinal para compensar efeitos lineares do canal teórico, e ruído gaussiano. Porém, o *overshoot* não pode ser representado nem como uma convolução linear, nem como ruído gaussiano, portanto o resultado do filtro não é satisfatório.

```
tic
x = cell(N_cycles,1);
yw = cell(N_cycles,1); yw{1} = zeros(length(x{1}),1);
```

```
x_c = ys_c{1}; x{1} = x_c(:,2);
d = cell(N_cycles,1); d{1} = ones(length(x{1}),1);
e = cell(N_cycles,1); e{1} = zeros(length(ys_c{1}),1);
ew = cell(N_cycles,1); ew{1} = zeros(length(ys_c{1}),1);
e_avg = zeros(N_cycles,1); ew_avg = zeros(N_cycles,1);

fig(2) = figure('name','Filtragem de Wiener'); subplot(3,1,[1 2]);
h1 = stem(x{1},'b'); hold all; set(h1,'MarkerFaceColor',[0.5,0.5,1]);
h2 = stem(d{1},'r','.');
h3 = stem(yw{1},'Color',[0.3, 0.8, 0.3]); set(h3,'MarkerFaceColor',[0.5,1,0.5]);
legend([h1, h2, h3], 'Original', 'Decisão', 'Filtrado');
xlim([0, 55]), ylim([-2.5, 2.5]), title('Solução de Wiener'); hold off;

subplot(3,1,3), h4 = stem(e{1}); hold all; set(h4,'MarkerFaceColor',[0.5,0.5,1]);
h5 = stem(ew{1}); set(h5,'MarkerFaceColor',[0.5,1,0.5]);
legend([h4 h5], 'S/ filtro','C/ filtro');
xlim([0 55]), ylim([-0.1, 2]); title('Erro');

M = 4;
Wo = zeros(N_cycles,M);
for k = 1:N_cycles
    x_c = ys_c{k}; x_c = x_c(:,2) - mean(x_c(:,2));
    coef = mean(abs(x_c)); x{k} = x_c/coef;
    d_c = ones(length(x{k}),1);
    if x_c(1) < 0, d_c(1:2:end) = -d_c(1:2:end); d{k} = d_c;
    else d_c(2:2:end) = -d_c(2:2:end); d{k} = d_c; end

    Wo(k,:) = WStoc(x{k},d{k},M); yw{k} = filter(Wo(k,:),1,x{k});
    e{k} = abs(d{k} - x{k}); e_avg(k) = sum(e{k})/Ns_cy(k);
    ew{k} = abs(d{k} - yw{k}); ew_avg(k) = sum(ew{k})/Ns_cy(k);
    plot_update(h1,x{k},h2,d{k},h3,yw{k},h4,e{k},h5,ew{k}); drawnow;
end
clear x_c coef d_c
toc
```

Elapsed time is 19.900140 seconds.



Filtragem Adaptativa RLS

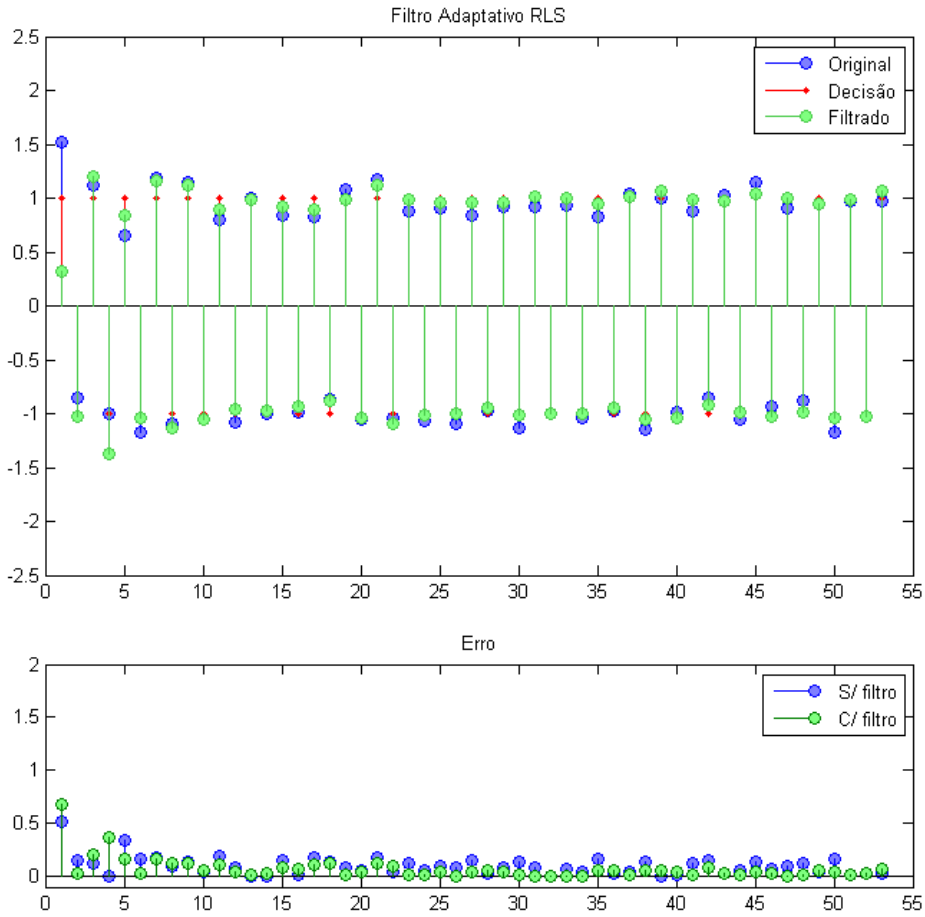
Ao contrário da filtragem de Wiener, o filtro adaptativo não utiliza as estatísticas de todas as amostras disponíveis para gerar os coeficientes do filtro. Ele considera apenas o erro instantâneo, além de seus próprios parâmetros. Consequêntemente, ele consegue se adaptar as oscilações causadas pelo *overshoot* com relativa velocidade, diminuindo o erro.

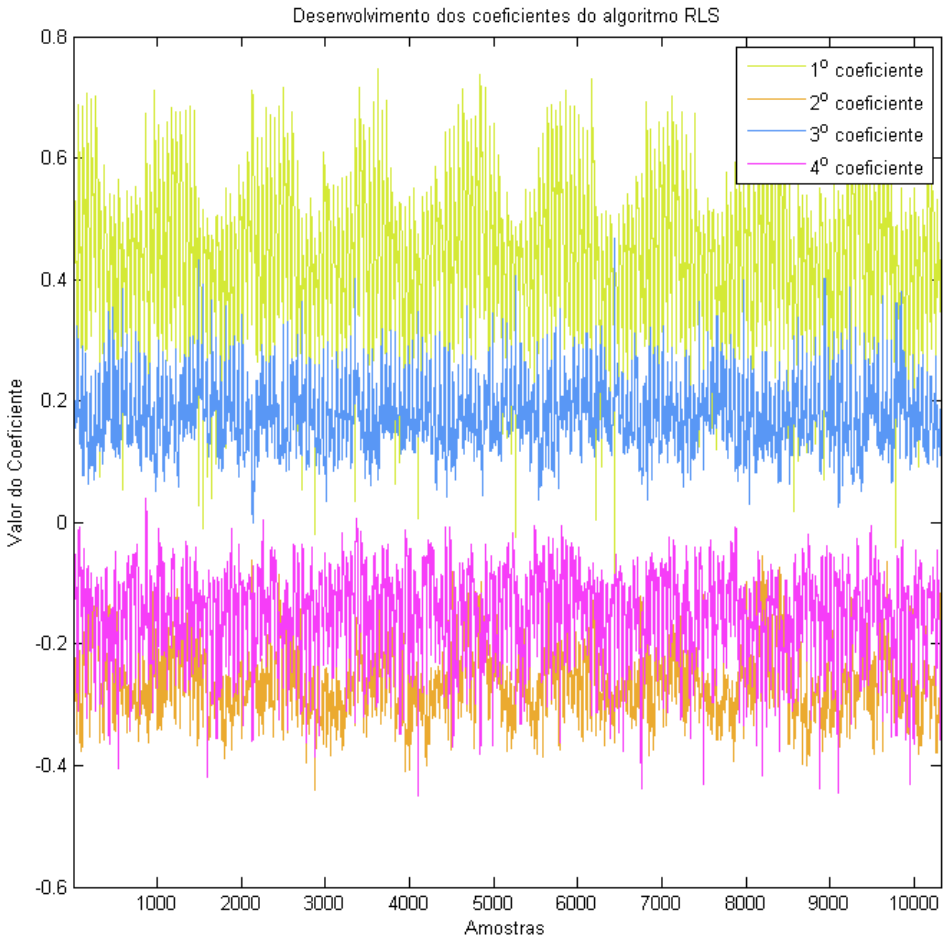
A primeira amostra, após o fechamento da chave, erra relativamente bastante, porém, a partir da segunda amostra, o erro já é diminuído expressivamente.

Ao fim da análise, é mostrado graficamente o desenvolvimento dos coeficientes do filtro ao decorrer das amostras. O parâmetro \lambda de esquecimento do algoritmo utilizado é de 0.9. É possível tirar algumas conclusões desse gráfico.

Primeiramente, é possível observar que nenhum coeficiente se estabiliza. Eles ficam se adaptando durante todo o processo. Provavelmente, isso se deve a compensação dos efeitos dinâmicos do chaveamento, como o *overshoot*. Em seguida, é possível observar que há um *flip flop* no sinal dos coeficientes. Os coeficientes ímpares são positivos, e os pares são negativos. Isso se deve ao fato de estarmos utilizando o sinal modulado quadrado (1, -1, 1, -1...). O sinal PRBS provavelmente demonstrará desempenho diferente.

```
tic
Wrls = cell(N_cycles,1); Wrls{1} = zeros(M,length(x{1}));
erls = cell(N_cycles,1); erls{1} = zeros(length(x{1}),1);
yrls = cell(N_cycles,1); yrls{1} = zeros(length(x{1}),1);
Wrls_cur = zeros(M,1);
erls_avg = zeros(N_cycles,1);

fig(3) = figure('name','Filtro Adaptativo RLS'); subplot(3,1,[1 2]);
h1 = stem(x{1},'b'); hold all; set(h1,'MarkerFaceColor',[0.5,0.5,1]);
h2 = stem(d{1},'r','.');
h3 = stem(yrls{1},'Color',[0.3, 0.8, 0.3]); set(h3,'MarkerFaceColor',[0.5,1,0.5]);
legend([h1, h2, h3], 'Original', 'Decisão', 'Filtrado');
xlim([0, 55]), ylim([-2.5, 2.5]), title('Filtro Adaptativo RLS'); hold off;
subplot(3,1,3), h4 = stem(e{1}); hold all; set(h4,'MarkerFaceColor',[0.5,0.5,1]);
h5 = stem(erls{1}); set(h5,'MarkerFaceColor',[0.5,1,0.5]);
legend([h4 h5], 'S/ filtro','C/ filtro');
xlim([0 55]), ylim([-0.1, 2]); title('Erro');
for k = 1:N_cycles
    [Wrls{k}, e_c, yrls{k}] = algRLS(x{k},d{k},Wrls_cur); erls{k} = abs(e_c);
    erls_avg(k) = sum(e_c)/Ns_cy(k);
    Wrls_cur = Wrls{k}; Wrls_cur = Wrls_cur(:,end);
    plot_update(h1,x{k},h2,d{k},h3,yrls{k},h4,e{k},h5,erls{k}); drawnow;
end

fig(4) = figure('name','RLS - Desenvolvimento dos Coeficientes'); leg_m = cell(M,1
Wrls_rec = zeros(M,sum(Ns_cy));
for m = 1:M
    for n = 1:N_cycles
        W_cur = Wrls{n}; W_cur = W_cur(m,:);
        t2 = sum(Ns_cy(1:n)) + 1; t1 = t2 - Ns_cy(n);
        Wrls_rec(m,t1:t2) = W_cur;
    end
    range = 1:sum(Ns_cy);
    plot(range,Wrls_rec(m,range),'LineWidth',1.2,'Color',.1+.9*rand(1,3)), hold on
    leg_m{m} = [num2str(m) '^o coeficiente'];
end
title('Desenvolvimento dos coeficientes do algoritmo RLS');
xlabel('Amostras'); ylabel('Valor do Coeficiente');
legend(leg_m),xlim([1 sum(Ns_cy)])
clear Wrls_cur leg_m e_c h1 h2 h3 h4 h5 h6 h7 hleg hleg2 t1 t2 leg_m n k i
toc

        Elapsed time is 20.052903 seconds.
```


Filtro Adaptativo RLS
Original
Decisão
Filtrado
Erro
S/ filtro
C/ filtro



Erro Resultante das Abordagens

Por fim, apresentam-se os erros resultantes sem a utilização de filtro algum, utilizando o filtro ótimo de Wiener, e com o filtro adaptativo RLS.

Observa-se que sem utilização de filtro algum, apresenta erro menor do que a utilização do filtro ótimo de Wiener. Isso está explicado na seção do mesmo. Entretanto, o filtro adaptativo RLS apresenta erro total 165 vezes menor.

```
e_t = sum([e_avg, ew_avg, erls_avg])/N_cycles;
fprintf(['\nErro médio sem filtragem:\t\t%f\n',...
    'Erro médio com filtro de Wiener:\t\t%f\n',...
    'Erro médio com filtro RLS:\t\t%f\n\n'],e_t);
```

```
Erro médio sem filtragem:  0.091716
Erro médio com filtro de Wiener:  0.144223
Erro médio com filtro RLS:  0.000400
```

Published with MATLAB® 8.0