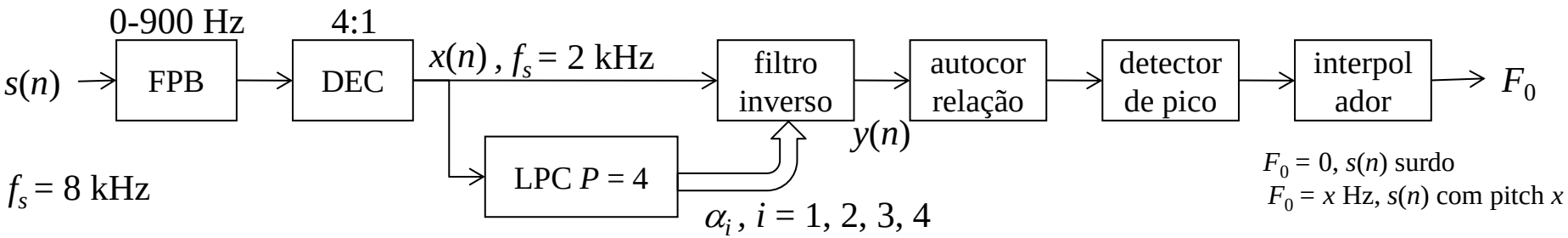
Exercício: detecção de pitch

- Implementaremos o método SIFT (*simple inverse filtering tracking*)

Implementação no MATLAB do método de detecção de pitch SIFT

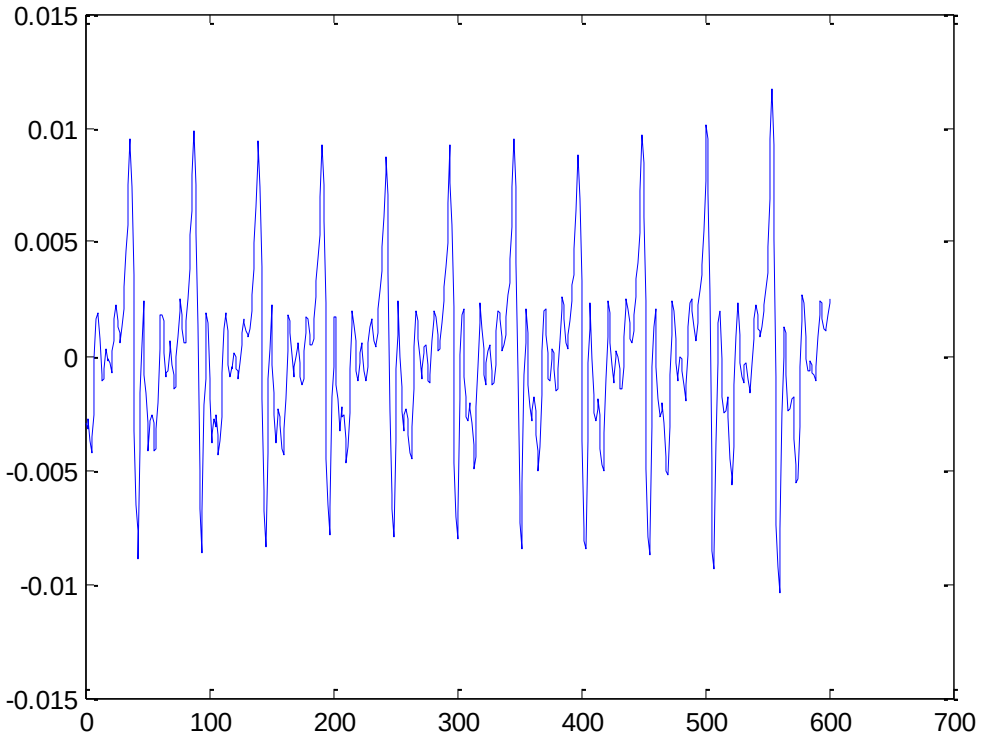


Utilizaremos o arquivo `o9.wav`, disponível no Moodle junto com este material (apenas para informação, é um pequeno trecho do som "ó" do dígito 9)

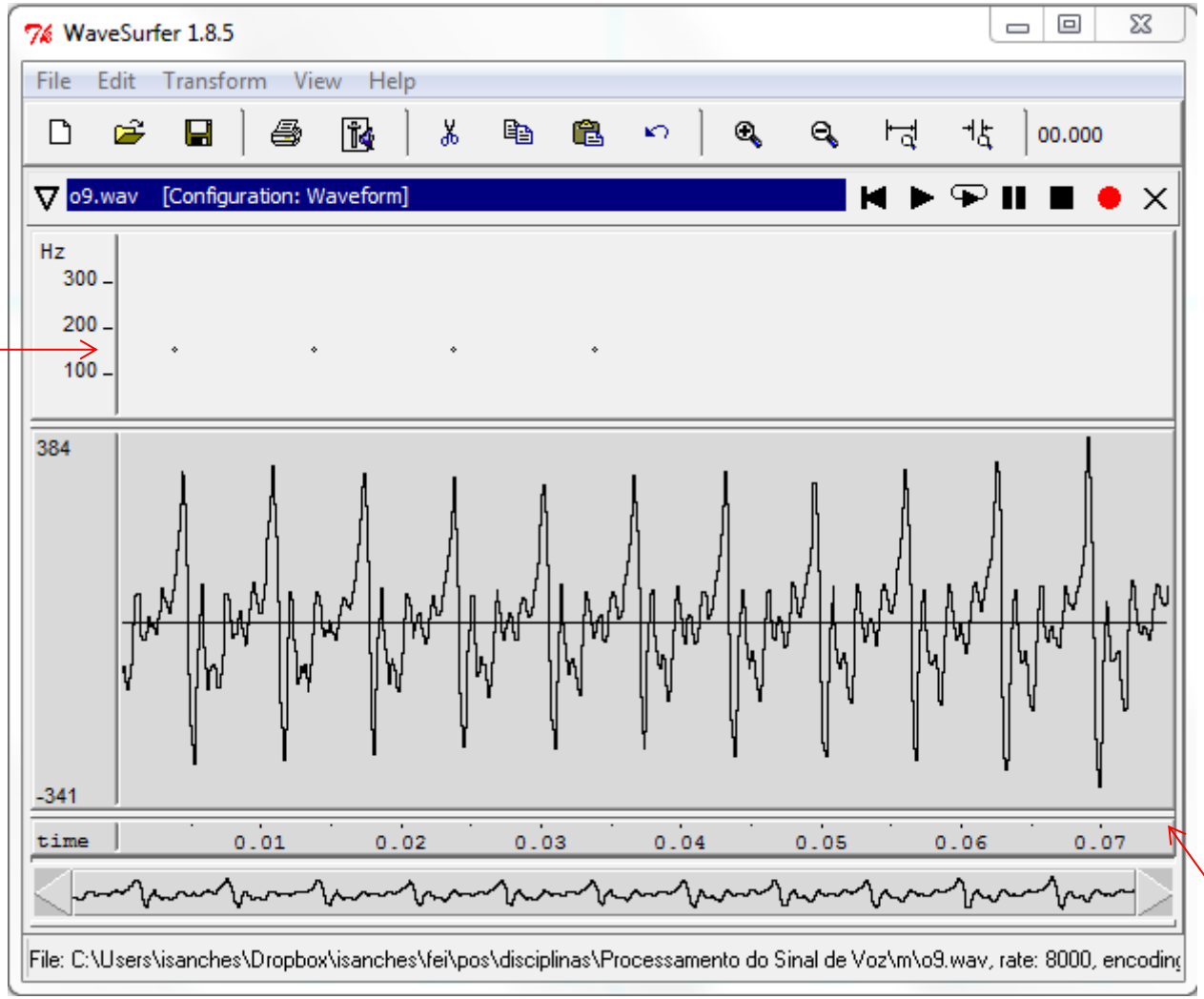
```
>> [s,fs]=audioread('o9.wav'); % antes: [s,fs,nbits]=wavread('o9');
>> fs

fs =

        8000
```

Visualização do sinal a ser usado na implementação

```
>>
>> plot(s)
```



Visualização do sinal via wavesurfer e painel de pitch



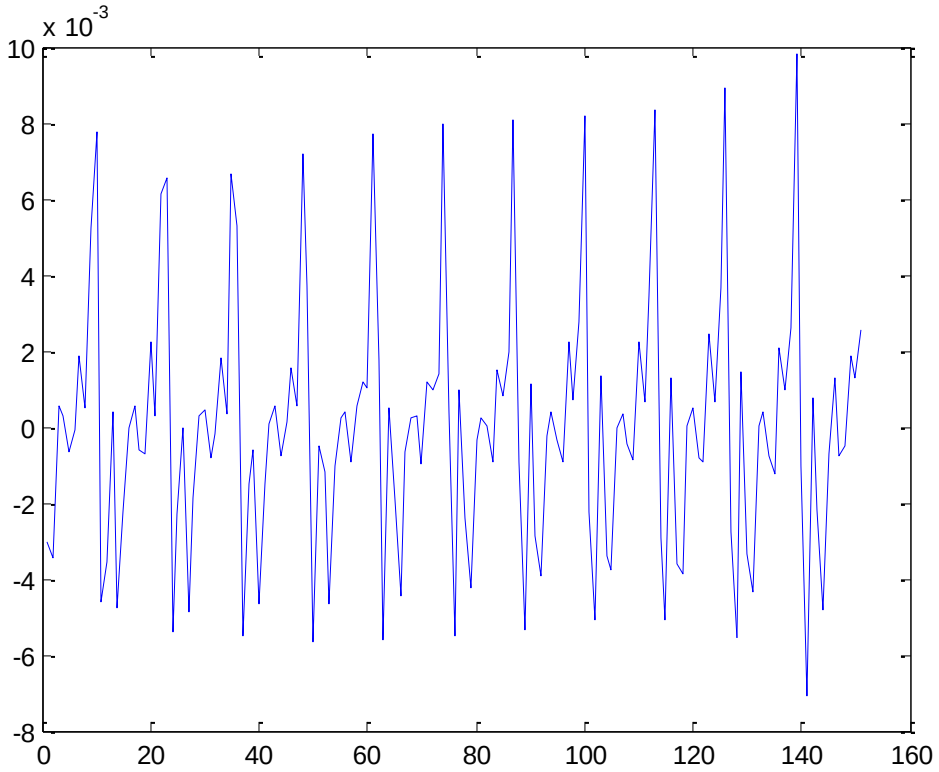
~155 Hz

No trecho de 75ms , apenas 4 valores estimados a cada 10ms, aparentemente em segmentos de 45ms, pois:
*45 ms + 3 * 10 ms = 75ms*
*(Note nessa expressão que deve ser 3 * 10ms e não 4 * 10ms)*

~75 ms

Primeira etapa: redução da frequência de amostragem do sinal para 2 kHz

```
>>
>> x = decimate(s, 4); % no MATLAB: >> help decimate
>> plot(x)
```



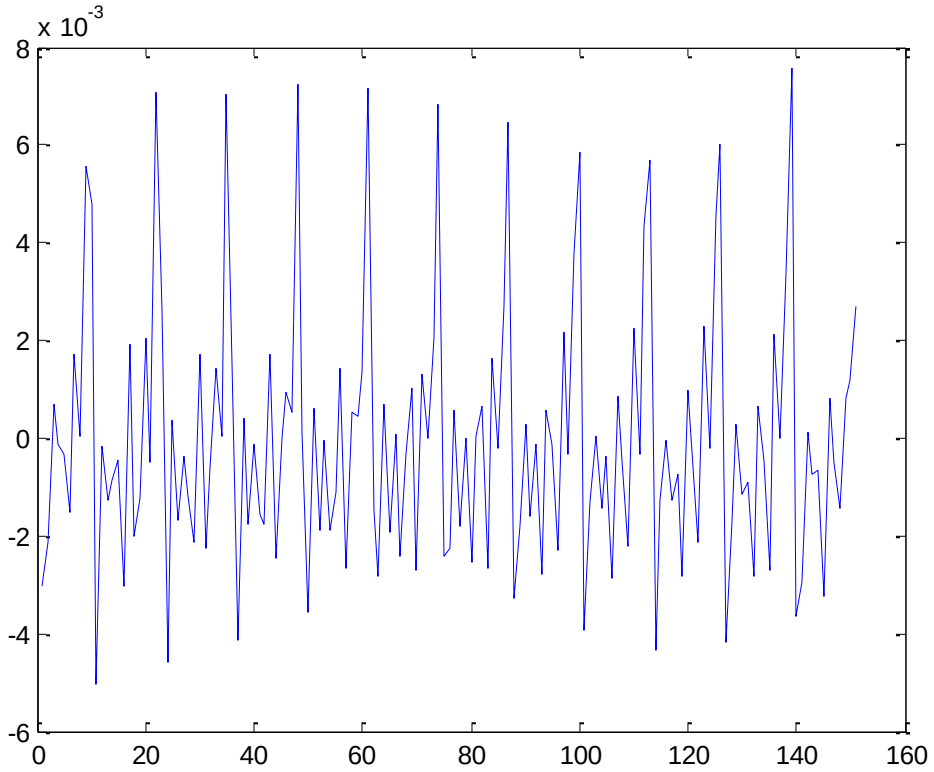
Segunda etapa: análise de predição linear de ordem 4 no sinal `x`

- método da autocorrelação
- não foi aplicada janela de Hamming

```
>> r % coeficientes de autocorrelacao

r =
   0.001626241214952
   0.000278081526932
  -0.000332437296367
   0.000348438509983
  -0.000319244537249


>> a % coeficientes de predicao

a =
   0.441916194832039
  -0.453408151410111
   0.461221866991762
  -0.462546522514809
```

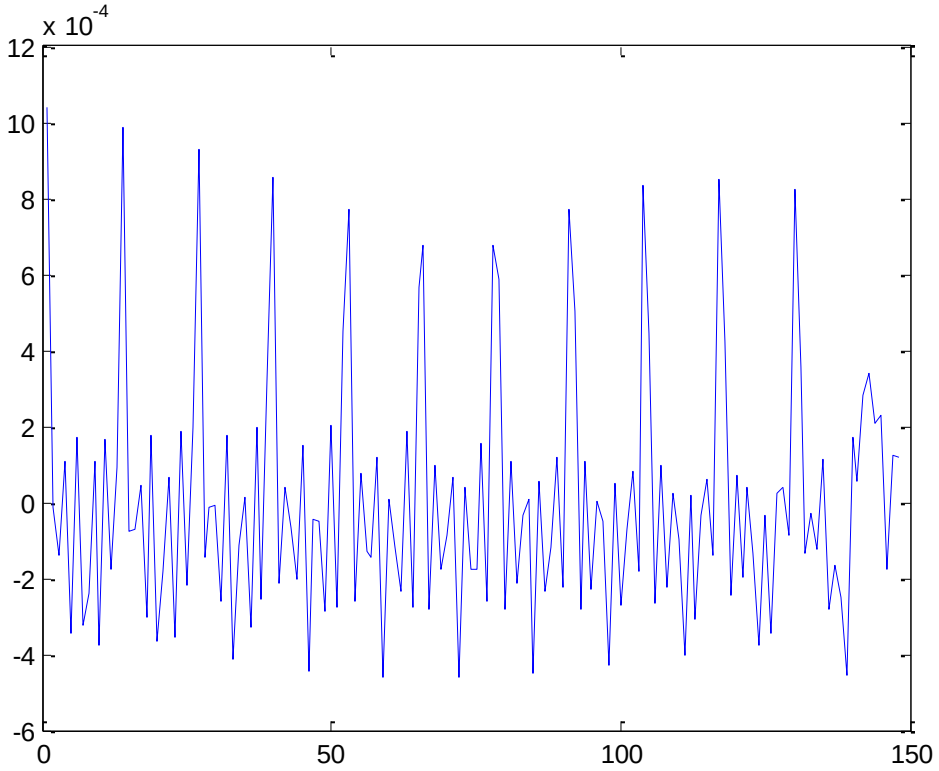
Terceira etapa: aplicação do filtro inverso

```
>> y = filter( ?? , ?? , x); % filtro inverso

>> plot(y)
```



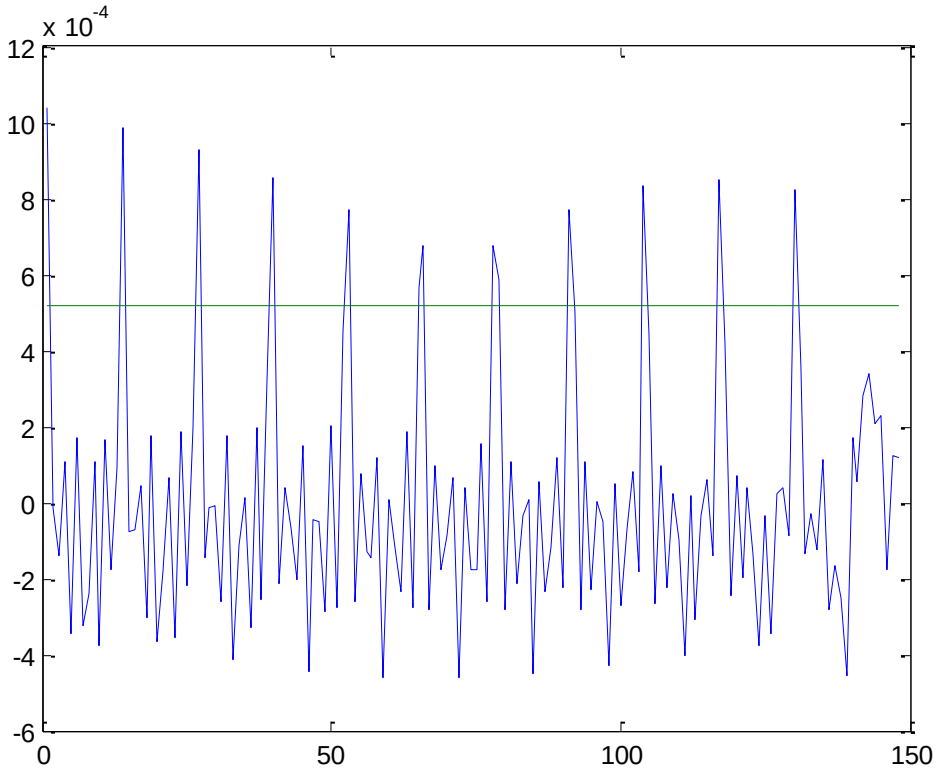
Quarta etapa: autocorrelação de `y`, com compensação pelo decaimento: `1/(1-i/length(y)), i = 0, 1,…, length(y)-1`

```
>> ry

ry =
   0.001037702519965  -0.000012595582549  -0.000140265186483 …etc…

>> plot(ry)
```



Quinta etapa: detector de pico. Assumindo limiar de `0.5` do valor máximo de `ry`

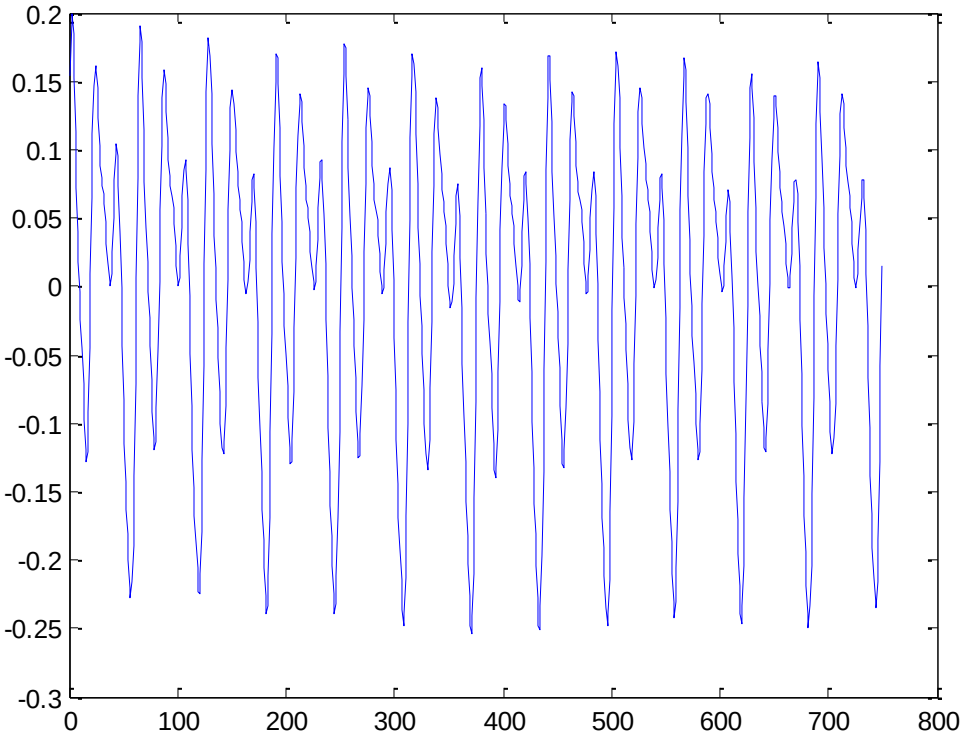
```
>> limiar = 0.5 * ry(1)

limiar =

     5.188512599823956e-04
```



Sexta etapa: interpolação (método simples a ser melhorado)

```
>> pp = find(ry > limiar)

pp =

 1    14    27    40    53    65    66    78    79    91   104   117   130

>> pd = diff(pp)

pd =

13    13    13    13    12     1    12     1    12    13    13    13

>> f0=round(2000./pd) % lembrar que fs = 2000 Hz apos decimate()

f0 =

154   154   154   154   167    2000 167    2000 167   154   154   154

>> f0=f0(f0 >= 50 & f0 <= 400) % faixa esperada para pitch: de 50 a 400 Hz

f0 =

   154   154   154   154   167   167   167   154   154   154

>> pitch = median(f0) % pitch em Hz

pitch =

   154
```

Repita agora o procedimento para o arquivo `o2.wav`, também disponível no Moodle junto com este material
(apenas para informação, é um pequeno trecho do som “ô” do dígito 2)

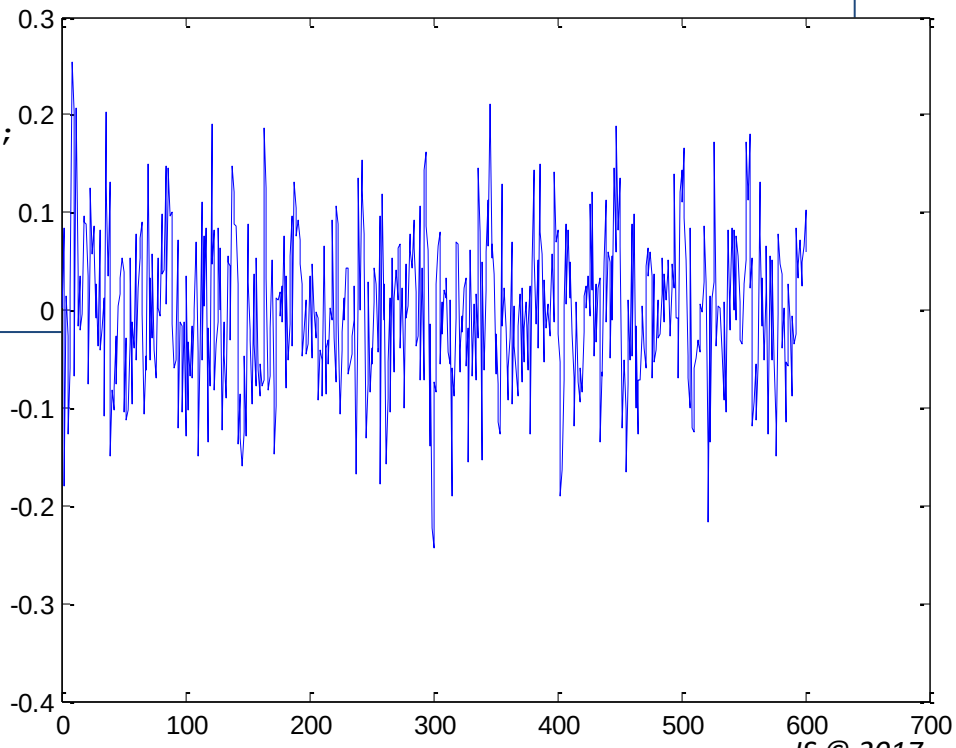


Anexo: análise do desempenho do algoritmo SIFT implementado com a adição de ruído branco gaussiano ao sinal (o9)

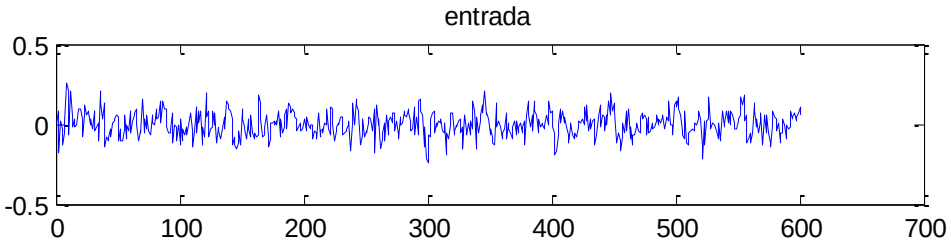
```
>> [sinal,fs,nbits]=wavread('o9');
>> ruido = randn(length(sinal),1);
>> ruido = ruido/sqrt(ruido'*ruido); % energia do ruido = 1
>> sinal = sinal/sqrt(sinal'*sinal); % energia do sinal = 1 (⇒ SNR = 0)
>> pitch = sift(sinal+ruido, fs, .5) % resultado ok para SNR = 0
pitch =
   154
>> pitch = sift(sinal+1.6*ruido, fs, .5) % aumento da intensidade do ruído
pitch =
   154
>> Es = sinal'*sinal;
>> En = (1.6*ruido')*(1.6*ruido);
>> snr = 10*log10(Es/En)
snr =
   -4.0824
>> plot(sinal+1.6*ruido)
```

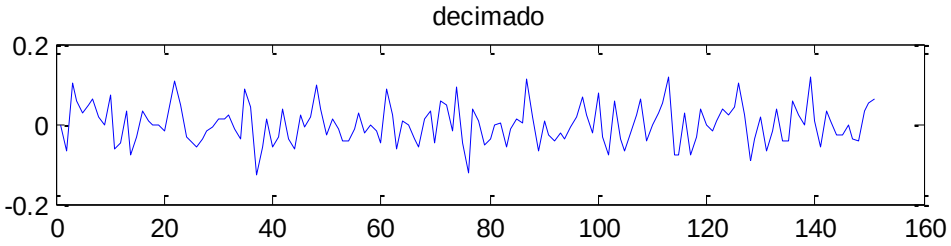
limiar da quinta etapa

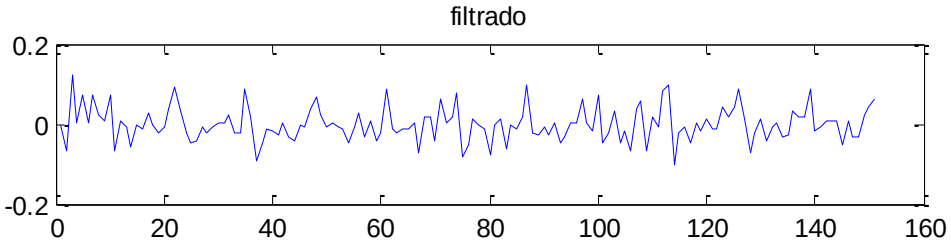
Note que o algoritmo ainda funcionou até a relação sinal-ruído de -4 dB. Veja ao lado o sinal processado. Na página seguinte apresentam-se os gráficos correspondentes.



Sinais no tempo







Autocorrelações