University of Applied Sciences Northwestern Switzerland
School of Engineering

Institute of Microelectronics

How to use the FFT and Matlab's pwelch function for signal and noise simulations and measurements

Hanspeter Schmid[*]

Abstract — This report describes how information on signal and noise levels can be extracted from an FFT when windowing is used. We explain in detail what the function pwelch from Matlab's Signal Processing Toolbox is doing, and how to read signal magnitudes out of pwelch-generated periodograms.

The target group of readers are engineers who want to simulate (or measure) signal-to-noise ratios using FFTs or periodograms on a captured signal, e.g., a sigma-delta bitstream.

Contents

[*] `hanspeter.schmid@fhnw.ch`, Institute of Microelectronics, University of Applied Sciences NW Switzerland.

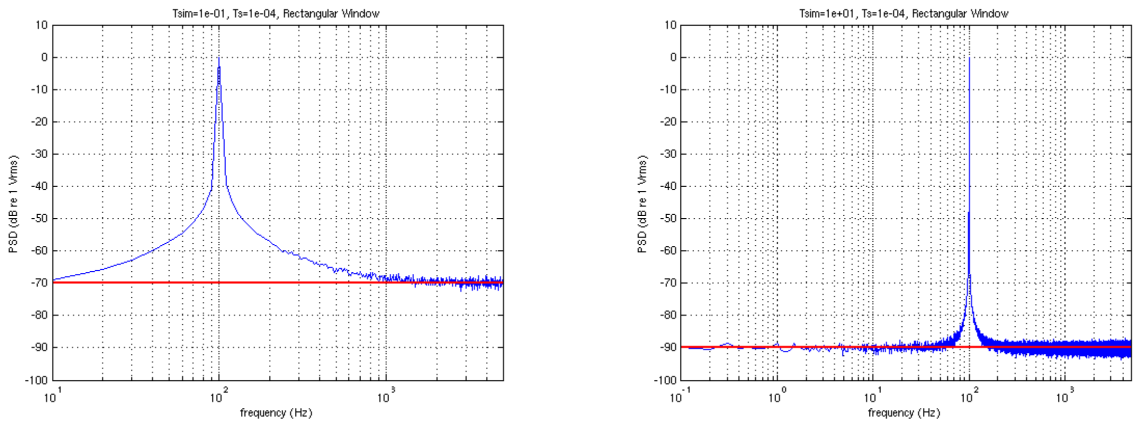


Figure 1: The same sum of white noise added to an sine function for two different simulation times.

1 Introduction

Simulating (or measuring) signals and noise with an FFT is not trivial, because signals and noise do not behave in the same way when plotted as a power spectrum using an FFT.

This phenomenon can be explained intuitively: Let us do the following thought experiment: we use a sine signal of frequency 100 Hz and amplitude $\sqrt{2}$ and a white noise source with power spectral density 10^{-8} over the whole frequency range covered by the FFT.[1] What happens if the simulation[2] time is increased by a factor of 100, but the sampling time is left the same?

The answer directly follows from Parceval's Theorem that states: the total signal power in the time domain and in the frequency domain is the same. If we just increase the simulation time, then the signal power does not change, so the amplitude of the signal stays the same. The noise power *also* does not change, but it is white noise, and occurs in all frequency bins of the FFT. We now have 100 times as many frequency bins as before, so we have to expect that the signal power within one frequency bin is diminished by a factor of 100, or 20 dB. Figure 1 shows these two simulations next to each other. The 20 dB difference is well visible.

To see this in simulation is not trivial, for two reasons: first, the FFT itself also introduces a factor N, the length of the FFT, and second, as can be seen in the left plot of Fig. 1, the signal may obfuscate the noise because it is smeared out. The latter effect can be fought with windowing.

2 FFT and windowing

Windowing means that the time series to be transformed is multiplied by a window function before the FFT is done. So instead of $x[i]$, we transform $x[i]w[i]$ for some window function which promises to produce a clearer spectral representation of the signal.

Like the FFT itself, the window also affects signals and noise in different ways. A wealth of details on windowing can be found in [1]; here we just give an intuitive explanation: every single frequency bin of the transformed signal is a linear combination of N time samples. If the signal to be transformed is a sine function, then, ideally, all these N samples add up in one bin and cancel out in all other bins, such

[1]In Matlab Simulink, this would be a "Sine Wave" block with amplitude $\sqrt{2}$ and frequency $2\pi 100$ rad/sec; and a "Band Limited White Noise" block with noise power $10^{-8}/2$. The reason for the $\cdot/2$ is that we want to have a one-sided power spectral density (PSD) of 10^{-8}, but the Simulink block "Band Limited White Noise" assumes a two-sided PSD.

[2]All of this is also valid for measurements.

Window	CG	NG	Scallop Loss
Rectangular	1.0000	1.0000	3.92 dB
Hamming	0.5400	0.3974	1.78 dB
Hanning	0.5000	0.3750	1.42 dB
Bartlett	0.5000	0.3333	—
Blackman-Harris	0.3587	0.2580	0.83 dB
Flat Top	0.2156	0.1752	—

Table 1: Correction factors and maximum scallop loss for different window types (only exact for large window lengths N).

that the sine will result in a single peak in the spectrum. For $x[i]w[i]$, the average value in the bin where the time signals add up will therefore be multiplied by

$$\mathrm{CG} = \frac{1}{N}\sum_{i=0}^{N-1} w[i] \tag{1}$$

compared to what happens when a rectangular window ($w[i] = 1$ for all i) is used. CG is the *coherent gain* of the window. If the signal is white noise, however, then the N time samples are uncorrelated. This means that in every bin, the noise *power* of N input values will add up, and the average value in the bin will be

$$\mathrm{NG} = \frac{1}{N}\sum_{i=0}^{N-1} w[i]^2 \tag{2}$$

compared to what happens when a rectangular window is used. We call NG the noise gain. For a rectangular window, CG $=$ NG $= 1$. For reference purposes, the correction factors and the scallop loss (see Sec. 3) of a few common windows are listed in Table 1.

There are three ways to normalise the resulting spectrum, depending on how one wants to use the PSD: first, to read signal values directly off the plot; second, to read the noise power spectral density directly off the plot; and third, to quantitatively determine the power in any frequency band by adding the values of all bins in that band.

Note that the third method will *always* give the most precise numbers; the first two methods are only useful for documentation! Unfortunately, Matlab's pwelch function returns a spectrum of the second type, as described below.

2.1 Normalisation for reading signal RMS values

If we want to be able to read the RMS value of deterministic signals from an FFT plot, we have to divide the FFT by N times the coherent gain and then calculate the power spectral density. So for an input signal x, we get for the one-sided power spectral density:

$$Y[i] = \frac{\mathrm{FFT}\left\{x[i]w[i]\right\}}{N \cdot \mathrm{CG}}\ , \tag{3}$$

$$P_{yy}[0] = Y[0] \cdot Y[0]^* \quad \text{and} \quad P_{yy}[i] = 2 \cdot Y[i] \cdot Y[i]^* \quad \text{for } i > 0. \tag{4}$$

Using this scaling, we can read the RMS-value of a deterministic signal directly off the plot. Note that this reading will not necessarily be precise; depending on the frequency of the signal relative to the centre frequency of the bin showing the highest value, and depending on the window used, there can be read-off errors up to several dB. This effect is called *scallop loss* [1] and sometimes also *picket fence effect*. We chose the signal frequencies in this paper such that this does not happen (except in Sec. 6, where we want to demonstrate scallop loss), so the 0 dB visible in Fig. 1 correspond to 1 $\mathrm{V_{rms}}$, which was the value used in the simulation. Note, however, that in a practical measurement one can not force this situation unless the clock of the data acquisition is synchronised to the signal to be measured.

How can we now read the power spectral density of the noise signal from the plot? When white noise with the power spectral density $\bar{x}_n^2$ is fed into the FFT, then the noise represented by one point of the FFT will be the noise integrated over a frequency range

$$f_{\text{bin}} = \frac{1}{T_{\text{sim}}} = \frac{1}{NT_{\text{samp}}} , \tag{5}$$

(where T_{sim} is the simulation or measurement time and T_{samp} is the sampling period) and then multiplied by the noise gain. Since we previously divided the result of the FFT by CG, what we plot actually is

$$P_{yy}[i] = \bar{x}_n^2 \cdot \frac{\text{NG} \cdot f_{\text{bin}}}{\text{CG}^2} . \tag{6}$$

Therefore the power spectral density calculated from the value in the FFT is:

$$\bar{x}_n^2 = \frac{P_{yy}[i]\text{CG}^2}{\text{NG}f_{\text{bin}}} , \tag{7}$$

or, in other words, if we want to plot a known power spectral density into a plot generated using (3) and (4), we first need to scale it by the factor

$$s_n = \frac{\text{NG}f_{\text{bin}}}{\text{CG}^2} . \tag{8}$$

The red lines in Figure 1 have been obtained like this. For a rectangular window, the calculation is trivial: since $\text{CG} = \text{NG} = 1$, we get $s_n = f_{\text{bin}}$.

2.2 Normalisation for reading noise values

Sometimes reading the noise level off a plot is more important than being able to read a signal. The same reasoning as above shows that in this case, the proper way to normalise the FFT is

$$Y[i] = \frac{1}{N}\text{FFT}\left\{x[i]w[i]\right\} , \tag{9}$$

$$P_{yy}[0] = \frac{Y[0] \cdot Y[0]^*}{\text{NG}f_{\text{bin}}} \quad \text{and} \quad P_{yy}[i] = \frac{2 \cdot Y[i] \cdot Y[i]^*}{\text{NG}f_{\text{bin}}} \quad \text{for } i > 0. \tag{10}$$

This is what Matlab's pwelch function is doing.

If we then know that the power at index i comes from a deterministic signal, the power of that signal is

$$P_{\text{sig}} = P_{yy}[i] \cdot \frac{\text{NG}f_{\text{bin}}}{\text{CG}^2} . \tag{11}$$

2.3 Normalisation for integrating spectral power

This case is closely related to the normalisation for reading noise values, only now we want each spectrum value to be the power integrated over the bin. This is the same as above, but without the division by f_{bin}:

$$Y[i] = \frac{1}{N}\text{FFT}\left\{x[i]w[i]\right\} \ , \tag{12}$$

$$P_{yy}[0] = \frac{Y[0]\cdot Y[0]^*}{\text{NG}} \quad \text{and} \quad P_{yy}[i] = \frac{2\cdot Y[i]\cdot Y[i]^*}{\text{NG}} \quad \text{for } i>0. \tag{13}$$

Then one can calculate the total power over some frequency range $f_1 \dots f_2$ as

$$P_{1,2} = \int_{f_1}^{f_2} P'_{yy}(f)df \approx \sum_{i_1}^{i_2} P_{yy}[i] \ , \tag{14}$$

where $P'_{yy}(f)$ is the true spectrum of the signal observed and i_1, i_2 are the indices of the bins that contain the frequencies f_1, f_2. This means that if one wants to integrate over the values returned by Matlab's pwelch function to calculate the power within a frequency range, then the pwelch spectrum must first be multiplied by f_{bin}.

3 On window functions

Much has been written about window functions, and using the best window function for a certain application requires a lot of specialised knowledge. Without going into the detail, we recommend to use the Hanning window for most applications, except when very low noise has to be observed in the presence of a signal, then the Blackman-Harris window gives better results.

Figures 2 and 3 show this. It can be seen in Fig. 3 that the Hanning window concentrates the signal in a narrower peak, but below some value, the signal is smeared out into so-called side lobes. The Blackman-Harris window creates a wider peak to start with, but has much lower side lobes.

Fig. 2 shows well that using windows is not for free, compared to the Rectangular window, they raise the visible noise floor. In Fig. 2, the rectangular window plots the white noise at -70 dB, the Hanning window at -68.24 dB, and the Blackman-Harris window at -66.98 dB. This means that the latter is least suitable when one wants to see small distortion peaks in the noise floor: distortion peaks are deterministic signals and will not change their magnitude in P_{yy} if we apply a different window.

However, remember the beginning of this report: if you want to see smaller peaks in the noise floor, then you need not optimise the FFT window. You just need to measure or simulate a longer data sequence!

4 Averaging

One problem of looking at noise with an FFT is that the obtained spectrum is inherently noisy. In fact, due to Parseval's theorem, the variance of the spectrum is simply the variance of the noise, or, in other words, $\sigma_n^2 = \bar{x}_n^2$.

Looking at any kind of scattered plots, there is a rule of thumb that states: the outer edges of the scattered field extend approximately to 3σ. This is shown in Fig. 4. It means that the upper visual edge of a noise spectrum is always at $(\bar{x}_n + 3\sigma_n)^2 = (4\bar{x}_n)^2$, which are 6 dB above $\bar{x}_n^2$ in the power spectrum.

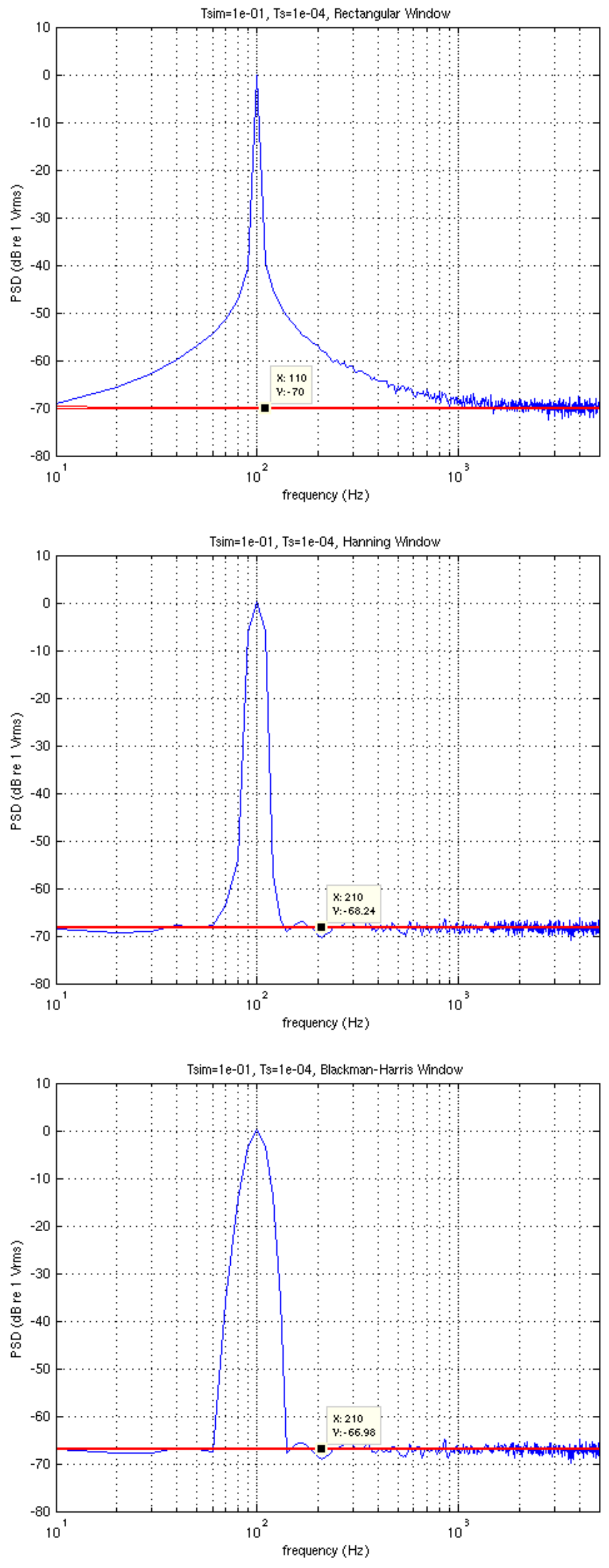
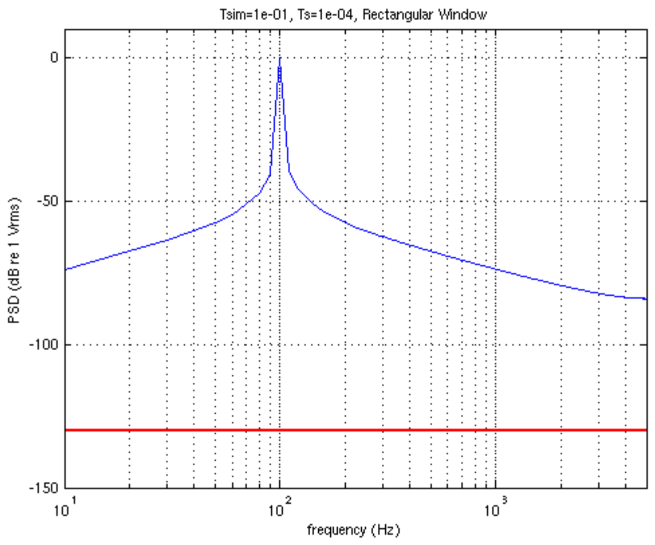
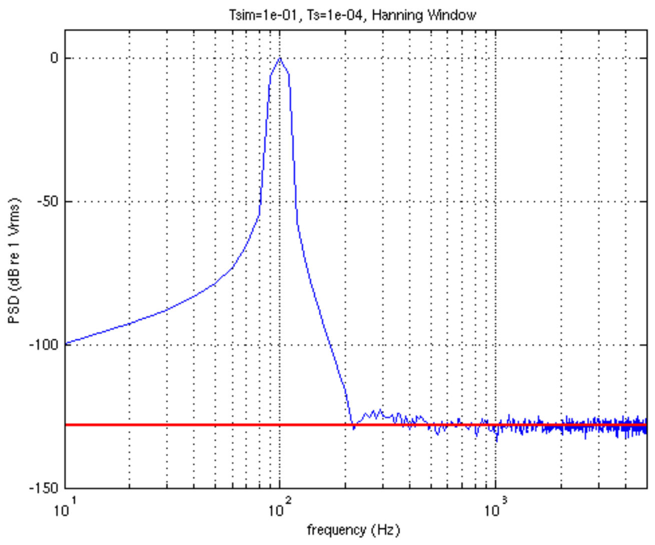


Figure 2: Rectangular, Hanning and Blackman-Harris Window applied to a signal of magnitude $\sqrt{2}$ and white noise with a one-sided power spectral density of 10^{-8}.





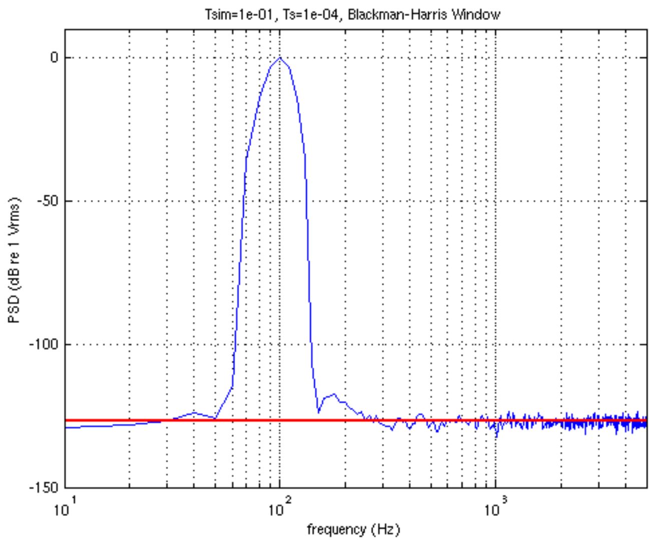


Figure 3: Rectangular, Hanning and Blackman-Harris Window applied to a signal of magnitude $\sqrt{2}$ and white noise with a one-sided power spectral density of 10^{-14}.

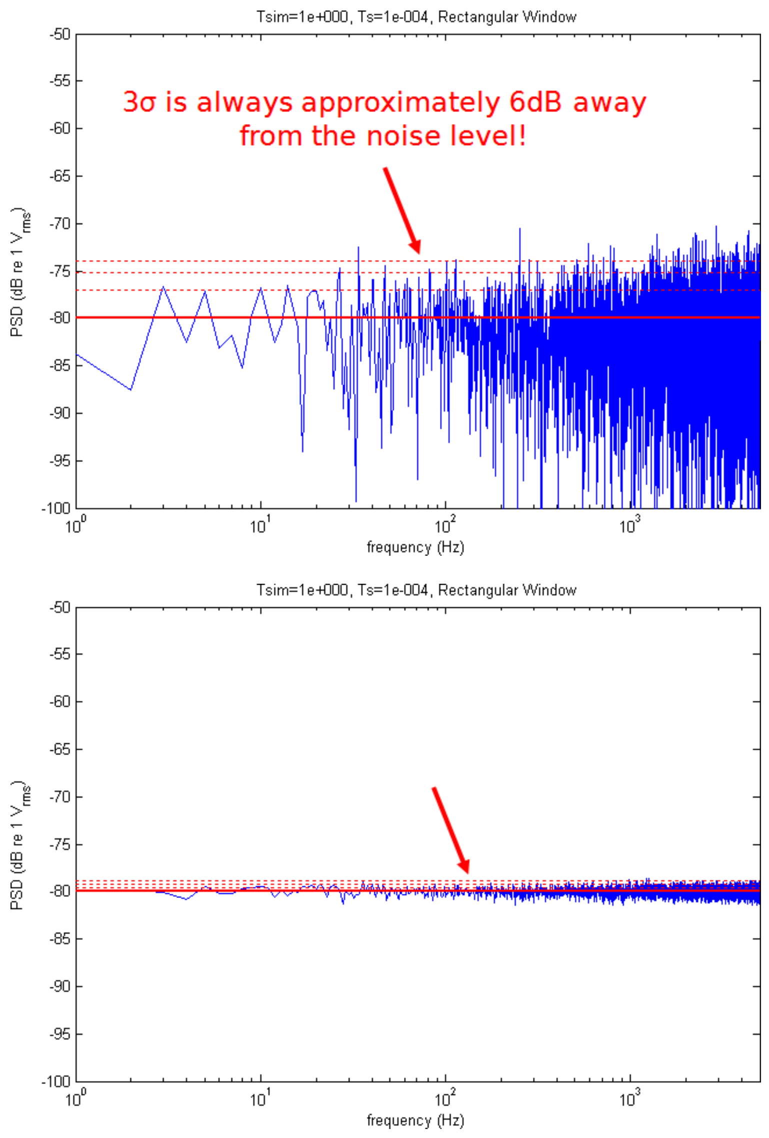


Figure 4: A demonstration of the 3σ principle without averaging (top) and with an averaging factor $n_a = 100$ (bottom).

Averaging is the process of making multiple measurements, determining the power spectrum for each measurement, and then calculating the average of these spectra. Given a number n_a of spectra to average, the standard deviation of the individual points of the spectrum is reduced from σ_n to $\sigma_n/\sqrt{n_a}$.

Therefore, the 3σ-peaks of an averaged spectrum extend to

$$(\bar{x}_n + 3\sigma_n)^2 = \left(1 + \frac{1}{\sqrt{n_a}}\right)^2 \bar{x}_n^2 \ . \tag{15}$$

For $n_a = 100$ as in Fig. 4, this factor in front of $\bar{x}_n^2$ is 1.14 dB. The Matlab function shown in Sec. 6 does averaging with $n_a = 8$.

5 Reading signal RMS values out of a Matlab pwelch periodogram

Matlab's pwelch function includes all of the above functionality, but the documentation is not so straightforward to read. We will discuss pwelch using several examples.

5.1 pwelch with standard parameters

Given a signal x, one can plot a one-sided periodogram using the command `pwelch(x)` or, if data instead of a plot are required, `[Pxx,f]=pwelch(x)`. This will cut up the signal into eight segments, with 50 % overlap between the segments. Hence, if x is a signal of length 1'000'000, then the length of the single-sided spectrum will not be 62501, as one would expect, but 131073. In practice, we obtain this number by actually executing `[Pxx,f]=pwelch(x)` and `size(f)`. So signal values are actually re-used to give better frequency resolution, but the averaging factor simply is $n_a = 8$.

The window used on the data is a Hamming window; the output is normalised as described in Sec. 2.2. Hence, to read a signal magnitude out of a plot generated with `pwelch`, one would let pwelch calculate the two vectors `[Pxx,f]` and then determine f_{bin} as `f(2)-f(1)`, and finally apply eq. 11 with the CG and NG values of the Hamming window.

5.2 pwelch with a given window length

`[Pxx,f]=pwelch(x,1000)` will use a 1000-point Hamming window on the signal. However, pwelch will use the next higher power of 2 as the length of the FFT. So the above command applied to a length 1'000'000 signal will give a 513-point spectrum with an averaging factor of $n_a = 1.5{\cdot}10^6/1000 = 1500$. Again, the best way to find out what pwelch is doing is to actually execute it and look at the length of the result vectors.

5.3 pwelch with a different window

The previous example is equivalent to `pwelch(x,hamming(1000))`. A different window can simply be used by writing `pwelch(x,hanning(1000))` or `pwelch(x,blackmanharris(1000))` to get a Hanning window or a Blackman-Harris window, respectively. The result is then still normalised properly to reading noise values off the plot, so in order to determine signal levels, eq. 11 must be applied with the CG and NG values of the respective window.

5.4 pwelch without overlap

`pwelch(x,hanning(1000),0)` will set the overlap to zero signal values. For a signal of length 1'000'000, this will reduce n_a from 1500 above to just 1000.

5.5 pwelch with real frequencies

All the above examples still work with normalised frequencies, i.e., a sampling frequency of 1 Hz is assumed and the single-sided spectrum goes from 0. . . 0.5 Hz. To use a real sampling frequency of, e.g., 256 kHz, one can use `pwelch(x,hanning(1000),0,[],256e3)`.

5.6 Recommended practice

Our recommended practice for everyday use is to use pwelch with a Hanning window, without overlap, because we are never sure what the overlap will be doing (this depends on signal statistics), and by specifying the averaging factor explicitly. The following code sequence will do this to a vector `x` with $n_a = 16$:

```
nx = max(size(x));
na = 16;
w = hanning(floor(nx/na));
[Pxx,f]=pwelch(x,w,0,[],256e3);
```

6 Calculating Signal and Noise with pwelch

To demonstrate all this, we provide an example script that generates a spectrum plot (Fig. 5) and calculates signal, noise and harmonic distortion.

The output of the example script:

```
Calculations from the spectrum
==============================

1. Measuring the signal magnitude:

   Theoretical value:             0.56569 Vrms
   Signal spectrum integrated:    0.56568 Vrms
   Maximum read off spectrum:     0.51681 Vrms
   Scallop loss:                  0.78484 dB

2. Measuring the noise floor:

   Theoretical value:             4.714 uV/sqrt(Hz)
   Signal spectrum integrated:    4.741 uV/sqrt(Hz)
   pwelch output averaged:        4.716 uV/sqrt(Hz)

3. Signal, Distortion and Noise:

   Signal power:                   -4.949 dB re 1 Vrms
   Third-order distortion:        -62.190 dB re 1 Vrms
   Fifth-order distortion:        -75.955 dB re 1 Vrms
   Noise from 0 to 2500Hz:        -69.656 dB re 1 Vrms
```

The signal frequency has been set for extra high scallop loss on purpose. The 0.785 dB above are quite close to the 0.83 dB that are given for the Blackman-Harris window in Table 1. Note that in spite of scallop loss, the *integrated* signal power is still correct. Scallop loss is not a fundamental problem, it is just a problem when one wants to read values off a plotted spectrum.

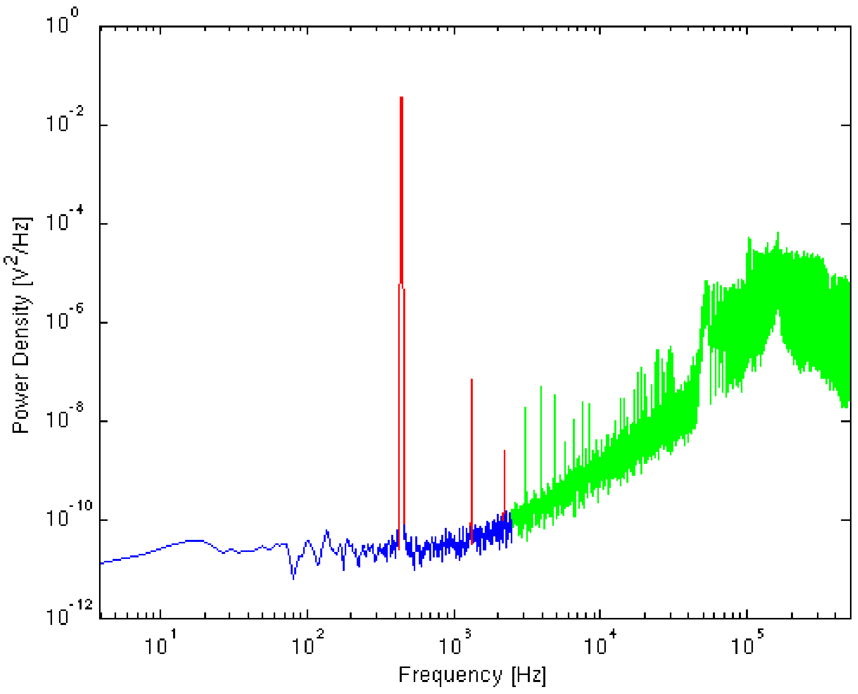


Figure 5: Example of a power spectrum generated with pwelch.

The script generating the above output is the following:

```
% sdex.m
% Determining the Signal-to-Noise Ratio of a signal.
% Hanspeter Schmid, May 2011 (updated August 2012)

% Generate an interesting signal (sigma-delta bitstream)
TSamp = 1e-6;   t = (0:TSamp:2^21*TSamp)';
fsig  = 440.55;                    % Constructed to show some scallop loss
asig  = 0.8;                       % Signal magnitude in V
vn    = 1/300;                     % Total integrated noise in Vrms
vin   = asig*sin(2*pi*fsig*t)+randn(size(t))*vn;
bout  = ones(size(vin));   qin = 0;   regin = vin(1);   regout = 0;
for k=2:max(size(t));
    if qin>=0
        qout=1;
    else
        qout=-1;
    end
    bout(k)=qout;
    qin = regin;
    regin = vin(k)-qout+qin;
end

% Calculate an 8-times averaged spectrum with pwelch
nx = max(size(bout));   na = 8;
w  = blackmanharris(floor(nx/na));
[Pxx,f] = pwelch(bout,w,0,[],1/TSamp);

% Calculate the spectrum parameters
fbin = f(2)-f(1);
```

```
CG   = sum(w)/(nx/na);
NG   = sum(w.^2)/(nx/na);

% Calculate the indices of the Signal, harmonics, band edge
isig = round(fsig/fbin)+1;
ih3  = round(3*fsig/fbin)+1;
ih5  = round(5*fsig/fbin)+1;
ibw  = round(2500/fbin)+1;

% Plot and mark the above
loglog(f,Pxx);
hold on
itmp = (isig-5:isig+5)'; plot(f(itmp),Pxx(itmp),'r');
itmp = (ih3-4:ih3+4)'; plot(f(itmp),Pxx(itmp),'r');
itmp = (ih5-4:ih5+4)'; plot(f(itmp),Pxx(itmp),'r');
plot(f(ibw:end),Pxx(ibw:end),'g');
hold off
a=axis(); a(1)=f(1); a(2)=f(end); axis(a);
xlabel('Frequency [Hz]');
ylabel('Power Density [V^2/Hz]');

% and now present a few calculations:

fprintf('\nCalculations from the spectrum\n')
fprintf('==============================\n\n')
fprintf('1. Measuring the signal magnitude:\n\n')
srmt = asig/sqrt(2);
fprintf('   Theoretical value:             %1.5f Vrms\n',srmt)
srms  = sqrt(sum(Pxx(isig-5:isig+5)*fbin));
fprintf('   Signal spectrum integrated:    %1.5f Vrms\n',srms)
srmsp = sqrt( Pxx(isig) * NG*fbin/CG^2 );
fprintf('   Maximum read off spectrum:     %1.5f Vrms\n',srmsp)
scl  = 20*log10(srmt/srmsp);
fprintf('   Scallop loss:                  %1.5f dB\n\n',scl)

fprintf('2. Measuring the noise floor:\n\n')
nth = vn/sqrt(max(f));
fprintf('   Theoretical value:             %2.3f uV/sqrt(Hz)\n',nth*1e6)
inmax = isig-20;
nsum = sqrt(sum(Pxx(1:inmax)*fbin)) / sqrt(f(inmax));
fprintf('   Signal spectrum integrated:    %2.3f uV/sqrt(Hz)\n',nsum*1e6)
navg = sqrt(mean(Pxx(1:inmax)));
fprintf('   pwelch output averaged:        %2.3f uV/sqrt(Hz)\n',navg*1e6)

fprintf('\n3. Signal, Distortion and Noise:\n\n')
% This calculation requires a trick with a mask such that the
% Noise summation does not use the Signal- and Distortion as well
imask = ones(size(Pxx));
imask(ibw+1:end)=0;
itmp = isig-5:isig+5; imask(itmp)=0; Ps = 10*log10(sum(Pxx(itmp)*fbin));
fprintf('   Signal power:                %8.3f dB re 1 Vrms\n',Ps)
itmp = ih3-4:ih3+4; imask(itmp)=0; P3 = 10*log10(sum(Pxx(itmp)*fbin));
fprintf('   Third-order distortion:      %8.3f dB re 1 Vrms\n',P3)
itmp = ih5-4:ih5+4; imask(itmp)=0; P5 = 10*log10(sum(Pxx(itmp)*fbin));
fprintf('   Fifth-order distortion:      %8.3f dB re 1 Vrms\n',P5)
Pn = 10*log10(sum(Pxx.*imask*fbin));
fprintf('   Noise from 0 to 2500Hz:      %8.3f dB re 1 Vrms\n',Pn)
```

References

[1] F. J. Harris, “On the use of windows for harmonic analysis with the discrete fourier transform,” *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, Jan. 1978.