Lecture 26

- Power spectrum calculation.
- Data windowing.
- Digital filtering in the time domain

Computing the power spectrum:

Up to now we have "informally" estimated power spectra by doing an FFT and squaring the amplitude. We need to take a closer look to get things right. Power spectra are important in many applications so it is worthwhile spending some time on the details.

First an issue of conventions (which as usual, are not necessarily spelled out in many treatments of the subject). All of the following definitions are related to power.

$$\sum_{j=0}^{N-1} |c_j|^2 \equiv \text{"sum squared amplitude"}$$

$$\frac{1}{T}\int_0^T |c(t)|^2 \, dt \approx \frac{1}{N}\sum_{j=0}^{N-1} |c_j|^2 \equiv \text{"mean squared amplitude"}$$

$$\int_0^T |c(t)|^2 \, dt \approx \Delta \sum_{j=0}^{N-1} |c_j|^2 \equiv \text{"time-integral squared amplitude"}$$

Power spectral density can be defined:

- Over discrete positive, zero and negative frequencies, and its sum over these is the function mean squared amplitude.
- For zero and discrete positive frequencies only, and its sum over these is the function mean squared amplitude.
- In the Nyquist interval $-f_c$ to f_c, and its integral over this range is the function mean squared amplitude.
- Defined from 0 to f_c and its integral over this range is the mean squared amplitude.

It does not make a lot of sense to integrate the power spectral density outside of the Nyquist domain, since we know that power out there is aliased into the domain.

One usual way of estimating the PSD is through the "periodogram". Start by FFT'ing your data,

$$C_k = \sum_{j=0}^{N-1} c_j \, e^{2\pi i j k/N} \qquad k = 0, \ldots, N-1$$

$$P(0) = P(f_0) = \frac{1}{N^2} |C_0|^2 \qquad f_k \equiv \frac{k}{N\Delta} = 2 f_c \frac{k}{N}$$

$$P(f_k) = \frac{1}{N^2} \left[|C_k|^2 + |C_{N-k}|^2 \right] \qquad k = 1, 2, \ldots, \left(\frac{N}{2} - 1 \right)$$

$$P(f_c) = P(f_{N/2}) = \frac{1}{N^2} |C_{N/2}|^2$$

If one applies Parseval's theorem, one sees that we have normalized things in such a way that summing $P(f_k)$ over the N+1 points one gets the mean square amplitude of the function.

The question now arises: how good is the periodogram as an approximation to "the real thing", that is how close is $P(f_k)$ to the continuous value P(f) evaluated at f_k?

First of all, obviously one does not expect these two quantities to match perfectly, since $P(f_k)$ represents the power in a "frequency bin". One would expect it to represent some sort of average of the continuous quantity given by a function that is narrowly concentrated across the bin.

For the periodogram estimate, the window function can be calculated exactly, it is, as a function of the distance s (measured in bins),

$$W(s) = \frac{1}{N^2}\left[\frac{\sin(\pi s)}{\sin(\pi s/N)}\right]^2 \qquad \text{(proof later)}$$

(sort of the profile one gets in a diffraction grating). The function has a narrow profile with oscillating lobes that decrease as $(\pi s)^{-2}$. This implies that there is "leakage" from one bin to another.

The window function vanishes for non-zero integer s, that means that if your signal is a sinusoid with frequency exactly f_k there will be no leakage.

This is rarely the case. If one has a perfect sinusoid of frequency somewhere in between bins, one will pick non-vanishing contributions across several bins. To address this problem we will introduce a technique called *data windowing*.

A related question is: how does the error in the periodogram depend on the number of points N (either by sampling a longer stretch of data or by sampling more often). The unpleasant news is: it is *independent* of N! The variance of the periodogram is the square of the mean frequency: a 100% error! Independent of sampling number!

How could this be? Where does the additional information we are gathering go? It goes to produce more values $P(f_k)$, not to produce better values!

One way of making the situation better is to sample the data with finer resolution (say, K times finer) than one needs and then "smooth out" across a number of bins (K) to obtain a periodogram with a smaller number of bins. The variance will decrease by a factor 1/K, and the error therefore by $1/K^{1/2}$.

Another way is to use a finer resolution and chop the function into K subintervals. One FFT's each of them and averages the resulting periodograms. This is computationally slightly more efficient and requires handling smaller FFTs, which is also better memory-wise.

Data windowing:

The purpose of data windowing is to control the leakage of the periodogram.

$$W(s) = \frac{1}{N^2}\left[\frac{\sin(\pi s)}{\sin(\pi s/N)}\right]^2$$

How did we end with the leakage? Every time we do a discrete sampling with a finite number of points it is equivalent to multiplying an infinite set of samplings times a window function in the time domain of length NΔ that is unity during that interval and zero outside of it.

The resulting function is the convolution of the function times the window sampling. The resulting transform is therefore the product of the function times the window function transform,

$$W(s) = \frac{1}{N^2}\left[\frac{\sin(\pi s)}{\sin(\pi s/N)}\right]^2 = \frac{1}{N^2}\left|\sum_{k=0}^{N-1} e^{2\pi i s k/N}\right|^2$$

The responsibility for the leakage lies on the fact that the window function we are using drops off very fast, therefore creating lots of high frequency components for the window transform.

The solution is therefore obvious: use a window function that rises and falls more gradually. The periodogram for such a less-trivial window function reads,

$$D_k \equiv \sum_{j=0}^{N-1} c_j w_j \, e^{2\pi i j k/N} \qquad k = 0, \ldots, N-1$$

$$W_{ss} \equiv N \sum_{j=0}^{N-1} w_j^2$$

$$P(0) = P(f_0) = \frac{1}{W_{ss}} \, |D_0|^2$$

$$P(f_k) = \frac{1}{W_{ss}} \left[|D_k|^2 + |D_{N-k}|^2 \right] \qquad k = 1, 2, \ldots, \left(\frac{N}{2} - 1 \right)$$

$$P(f_c) = P(f_{N/2}) = \frac{1}{W_{ss}} \, |D_{N/2}|^2$$

Leakage is determined by the transform of the window function,

$$W(s) = \frac{1}{W_{ss}} \left| \sum_{k=0}^{N-1} e^{2\pi i s k/N} w_k \right|^2$$

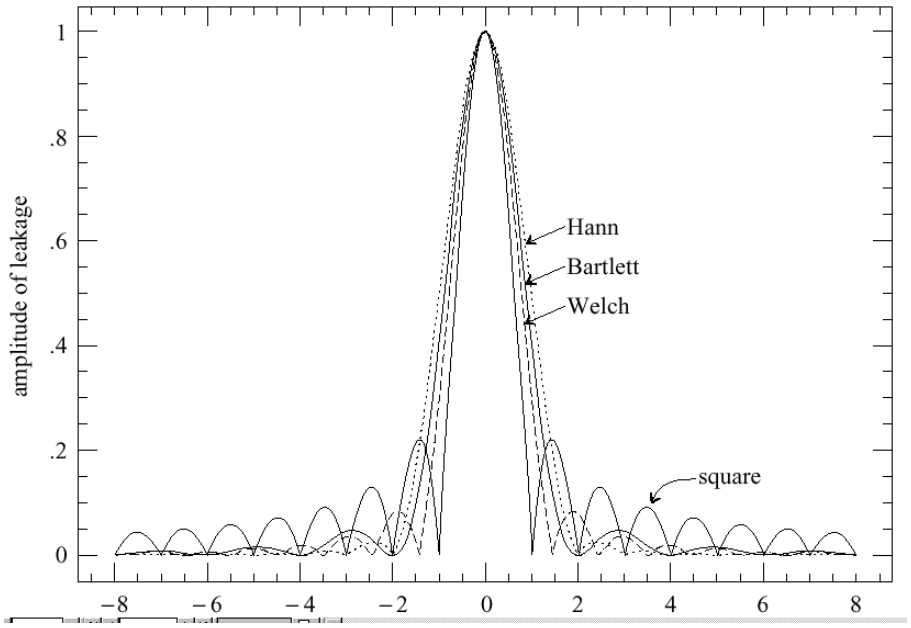
$$\approx \frac{1}{W_{ss}} \left| \int_{-N/2}^{N/2} \cos(2\pi s k/N) w(k - N/2) \, dk \right|^2$$

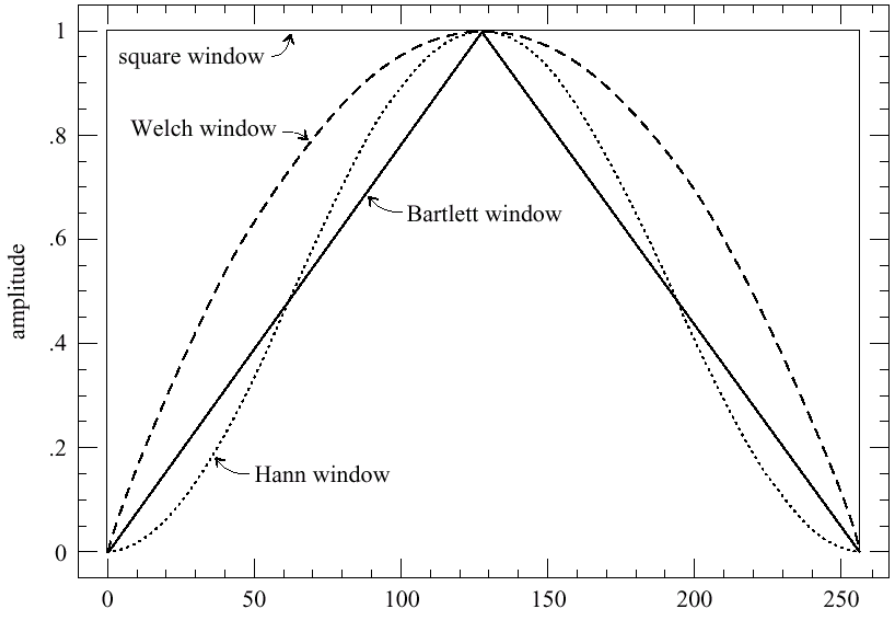
Given that the subject is a mature one, there are several proper-name window functions. There is a lot of mystique about them, but they all do the job well, since the requirements are just to have a smooth rise and fall,

$$w_j = 1 - \left| \frac{j - \frac{1}{2}N}{\frac{1}{2}N} \right| \equiv \text{“Bartlett window”}$$

$$w_j = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi j}{N}\right)\right] \equiv \text{“Hann window”}$$

$$w_j = 1 - \left(\frac{j - \frac{1}{2}N}{\frac{1}{2}N}\right)^2 \equiv \text{“Welch window”}$$





(and closely related ones called Parzen and Hamming windows).

As you saw in the picture, the main tradeoff is how narrow the central peak is at what cost of the amplitude of the side-lobes. All this has a highly developed technical nomenclature: “highest sidelobe level” (dB) “sidelobe falloff per octave” (dB/octave), “equivalent noise bandwidth” (bins), “3-dB bandwidth” (bins), “scallop loss” (dB), “worse case process loss” (dB)...

Tips on filtering:

• Remember that your filter should be defined for both positive and negative frequencies, within the Nyquist band and there is a minimum frequency given by +/-1/(NΔ).
• If the data is real, you should have your filter satisfy H(f)=H*(-f).
• You can see what your filter is doing in the time domain by $(FFT)^{-1}$.
• Remove any secular trend from the data.
• If you partition the data, use overlaying intervals to work safely away from endpoints.
• Filters in the frequency domain are in general a-causal (but who cares?)

```
      SUBROUTINE spctrm(p,m,k,ovrlap,w1,w2)
      INTEGER k,m
      REAL p(m),w1(4*m),w2(m)
      LOGICAL ovrlap                  True for overlapping segments, false otherwise.
C     USES four1
        Reads data from input unit 9 and returns as p(j) the data's power (mean square amplitude)
        at frequency (j-1)/(2*m) cycles per gridpoint, for j=1,2,...,m, based on (2*k+1)*m
        data points (if ovrlap is set .true.) or 4*k*m data points (if ovrlap is set .false.).
        The number of segments of the data is 2*k in both cases: The routine calls four1 k
        times, each call with 2 partitions each of 2*m real data points. w1(1:4*m) and w2(1:m)
        are user-supplied workspaces.
      INTEGER j,j2,joff,joffn,kk,m4,m43,m44,mm
      REAL den,facm,facp,sumw,w,window
      window(j)=(1.-abs(((j-1)-facm)*facp))     Statement function defines Bartlett window.
C     window(j)=1.                              Alternative for square window.
C     window(j)=(1.-(((j-1)-facm)*facp)**2)     Alternative for Welch window.
      mm=m+m                                    Useful factors.
      m4=mm+mm
      m44=m4+4
      m43=m4+3
      den=0.
      facm=m                                    Factors used by the window statement function.
      facp=1./m
      sumw=0.                                   Accumulate the squared sum of the weights.
      do 11 j=1,mm
        sumw=sumw+window(j)**2
      do 12 j=1,m                               Initialize the spectrum to zero.
        p(j)=0.
      enddo 12
      if(ovrlap)then                            Initialize the "save" half-buffer.
        read (9,*) (w2(j),j=1,m)
      endif
      do 18 kk=1,k                              Loop over data set segments in groups of two.
        do 15 joff=-1,0,1                       Get two complete segments into workspace.
          if (ovrlap) then
            do 13 j=1,m
              w1(joff+j+j)=w2(j)
            enddo 13
            read (9,*) (w2(j),j=1,m)
            joffn=joff+mm
            do 14 j=1,m
              w1(joffn+j+j)=w2(j)
            enddo 14
          else
            read (9,*) (w1(j),j=joff+2,m4,2)
          endif
        enddo 15
```

```
      do 16 j=1,mm                          Apply the window to the data.
          j2=j+j
          w=window(j)
          w1(j2)=w1(j2)*w
          w1(j2-1)=w1(j2-1)*w
      enddo 16
      call four1(w1,mm,1)                   Fourier transform the windowed data.
      p(1)=p(1)+w1(1)**2+w1(2)**2           Sum results into previous segments.
      do 17 j=2,m
          j2=j+j
          p(j)=p(j)+w1(j2)**2+w1(j2-1)**2
     *         +w1(m44-j2)**2+w1(m43-j2)**2
      enddo 17
      den=den+sumw
    enddo 18
    den=m4*den                              Correct normalization.


    do 19 j=1,m
        p(j)=p(j)/den                       Normalize the output.
    enddo 19
    return
    END
```

Digital filtering in the time domain:

If you want to filter data in real time, or if the data stream is too long to be FFT'd effectively, you might want to apply filters directly in the time domain.

For instance, if your signal only lies in a certain frequency band you need what is known as a *bandpass* filter.

Or if your data is contaminated by 60Hz noise from the electrical mains, you may want to run a notch filter to remove a narrow band around that frequency.

Unless one is forced by data length, it is always more convenient and efficient to filter in Fourier domain.

Linear filters:

Given a sequence of input x_k a linear filter produces an output y_k,

$$y_n = \sum_{k=0}^{M} c_k \, x_{n-k} + \sum_{j=1}^{N} d_j \, y_{n-j}$$

The output generically depends on the M sampled values of x and possibly on N of the already filtered values of y. if N=0 the filter is called "non-recursive". The Fourier transform of the filter is,

$$\mathcal{H}(f) = \frac{\sum_{k=0}^{M} c_k e^{-2\pi i k (f\Delta)}}{1 - \sum_{j=1}^{N} d_j e^{-2\pi i j (f\Delta)}}$$

To design a filter one usually wants the inverse, that is, the c's and d's as a function of H. This is an "inverse problem" and is therefore hard.

One clearly has to make compromises sin H(f) is a function and the d's and c's are a finite number of parameters to adjust. The subject of digital filter design is about those compromises. We will sketch a couple of basic techniques to get you started.

Non-recursive filters:

In this case the denominator of the formula for H(f) is unity and one is left with a Fourier transform. If one simply does the inverse transform one can solve for the c's.

This is not a good idea. The reason is that the resulting H(f) so constructed will oscillate (usually strongly) between the frequencies f_k where it gets "pinned down" by the equation. Like interpolating polynomials of high order do. This is not good for a filter.

A better idea is to take a rather large number M, perform the Fourier transform and then keep only the initial coefficients. That way one cuts out the high frequency oscillations. One then transforms back to check that the filter indeed is doing what one wants. If it does not, one can try keeping a larger number of coefficients.

A more sophisticated version of this idea is the Remes Exchange Algorithm that produces the best Chebyshev approximation to a given Response with a fixed number of coefficients.

Summary

- Power spectrum: beware of the details.
- Windowing allows to control bin leakage due to discreteness.
- Filtering in the time domain is forced on you if your data stream is too large or intense.
- Recursive filtering: avoid oscillations keeping only some coefficients.