EE102B Spring 2018-19 Pauly
Signal Processing and Linear Systems II

Homework 3

Due Apr 26, 2019

1. *DFT's of Simple Sequences*
   Find the DFT's of these 8-sample signals for $n = 0..7$. Only minimal computation should be required.

   (a) $\{1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\}$
   (b) $\{1\ -1\ \ 1\ -1\ \ 1\ -1\ \ 1\ -1\}$
   (c) $\{e^{i\pi n/2}; n = 0..7\}$
   (d) $\{\sin(\pi n/2); n = 0..7\}$
   (e) $\{1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\}$
   (f) $\{0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\}$

2. *Circular Verses Linear Convolution*
   Let $x[n] = \{1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\}$ for $n = 0..7$ and $y[n] = \{1\ 2\ 3\ 4\ 3\ 2\ 1\ 0\}$ also for $n = 0..7$.

   (a) Find the convolution of these two signals, and sketch the result.
   (b) Find the 8-point circular convolution of these two signals, and sketch the result.
   (c) Assume that each of the signals has been zero padded up to a length 16. Find the 16-point circular convolution of these two zero-padded signals, and sketch the result.

3. *Symmetries of the DFT*

   Let $f[n]$ be an 8-sample sequence with the $DFT\{f[n]\} = F[k]$. We know some of the values of $f[n]$:

   $$f[n] = \{1\ \ i\ \ -1\ \ -i\ \ 1\ \ ?\ \ ?\ \ ?\}$$

   Other values, indicated by "?", are unknown. We want to find these unknown values based on the what we know about the symmetry of the signal or the symmetry of its DFT.

   Determine the rest of $f[n]$ for each of the following cases. If $F[k]$ cannot possess the given property, explain.

   (a) $F[k]$ is even (i.e. $F[k] = F[((-k))_8]$).
   (b) $F[k]$ is odd (i.e. $F[k] = -F[((-k))_8]$).
   (c) $F[k]$ is imaginary-valued.
   (d) $F[k]$ is real-valued.

4. *Faster DFT's?*

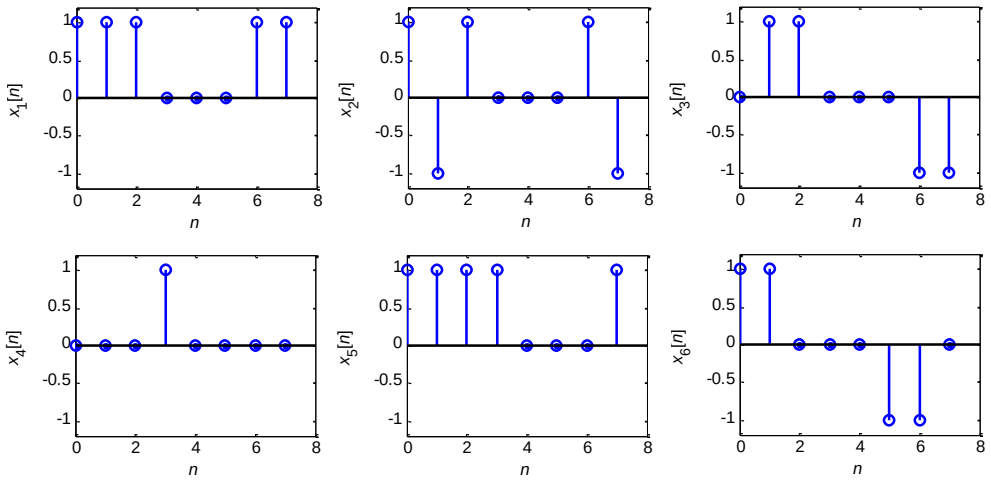
   Let $f[n]$ and $g[n]$ be N-point real-valued sequences with DFT's of $F[k]$ and $G[k]$ respectively. Consider the possibility of computing both of their DFT's simultaneously by using an N-point complex DFT.

   One idea is to construct $h[n] = f[n] + jg[n]$. If $DFT\{h[n]\} = H[k]$, find separate expressions (if possible) for $F[k]$ and $G[k]$ in terms of $H[k]$. If it is not possible to separate $F[k]$ and $G[k]$ from $H[k]$, explain why.

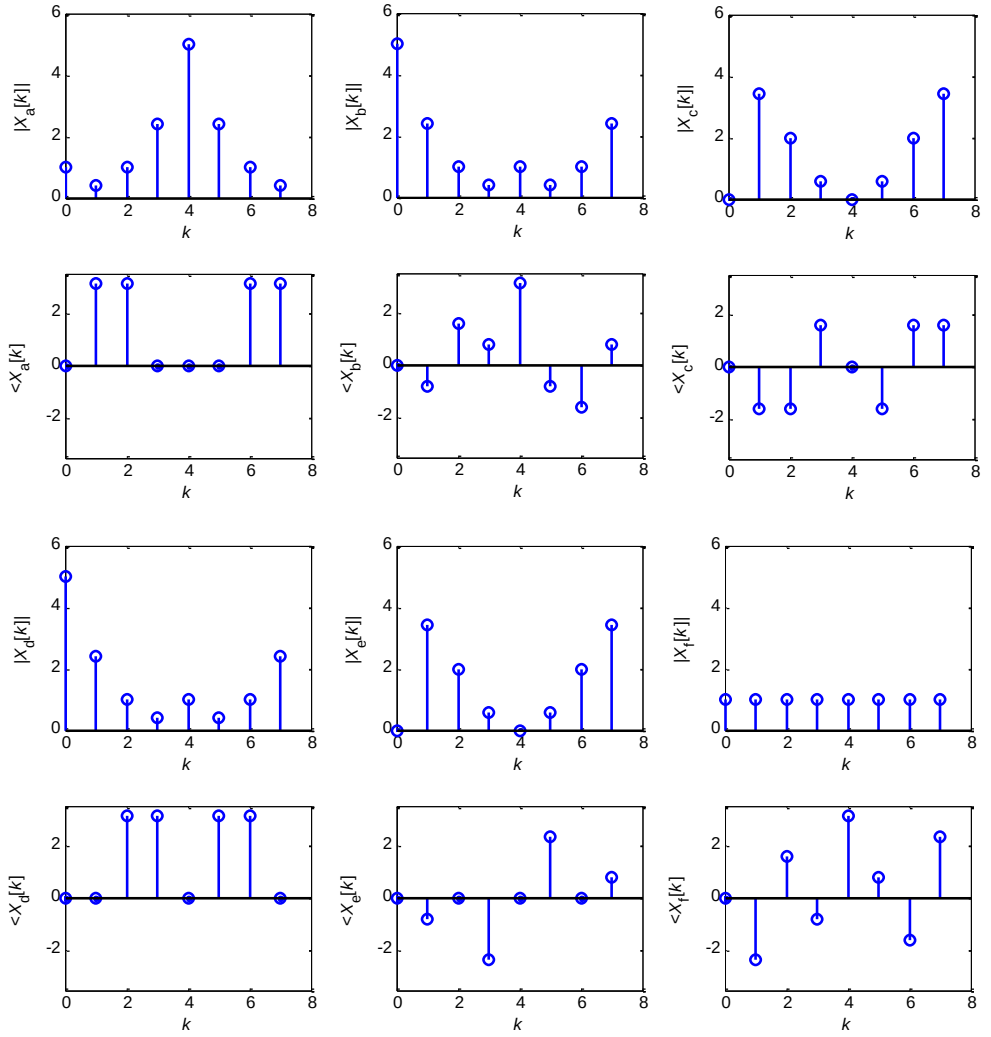
5. *Discrete Time Signals and Their DFT's*

   On the next page we have six real signals $x_m[n]$ and their DFTs $X_m[k]$ where $1 \leq m \leq 6$, with the order scrambled. Each is of length $N = 8$. Match each signal to its DFT. Provide a simple argument for your choice. Things to look for: The signals are all either impulses or rects, maybe shifted or modulated. What does that do to the spectra? Note that we are plotting magnitude and phase, so things may look different than you expect (negative numbers are a positive numbers with a phase of π). Most of these are pretty straight forward, but a couple are more subtle.

Signals



Spectra



Laboratory

This week we will look at estimating the spectrum of time-varying signals.

In the previous lab we used the DFT to estimate the frequency of a segment of a signal. When we do this, we are implicitly assuming that the frequency is constant over that interval. This is a reasonable assumption during the time a key on the phone is pressed,

There are many times when we'd like analyze signals whose frequency is changing over time. In fact, most signals aren't interesting unless they do change! There are many examples, including speech, music, and the sounds that surround you in daily life. In this lab we will learn how to process these signals to determine how their spectrum changes with time.

The basic problem is that we have long segment of a signal $x[n]$, where $n = 1, N$. We want to know what its frequency is as a function of time. There are two basic approaches. One is to pass the signal through a bank of bandpass filters, and plot the outputs of the filters as a function of time. This is what you did in the DTMF lab last time. The second approach is to break the signal up into short segments, and compute the spectrum of each of these separately. This is the approach we will use in this lab.

A sample signal is shown in Fig. 1. Obviously you can tell that the amplitude is changing as a function of time. When you examine the signal closely, you can also tell that the frequency is changing. Since the frequency is changing, we want to break the signal into segments over which the frequency is relatively constant. Then we will analyze each segment using the DFT. The result of analyzing the first 256 samples, and another block of 256 samples that is 1024 samples later is shown in Fig. 2. As is the convention with the DFT, zero frequency is the first sample, and the sampling rate corresponds to the last sample. Note that the frequency has dropped about 300 Hz between the first and second block.

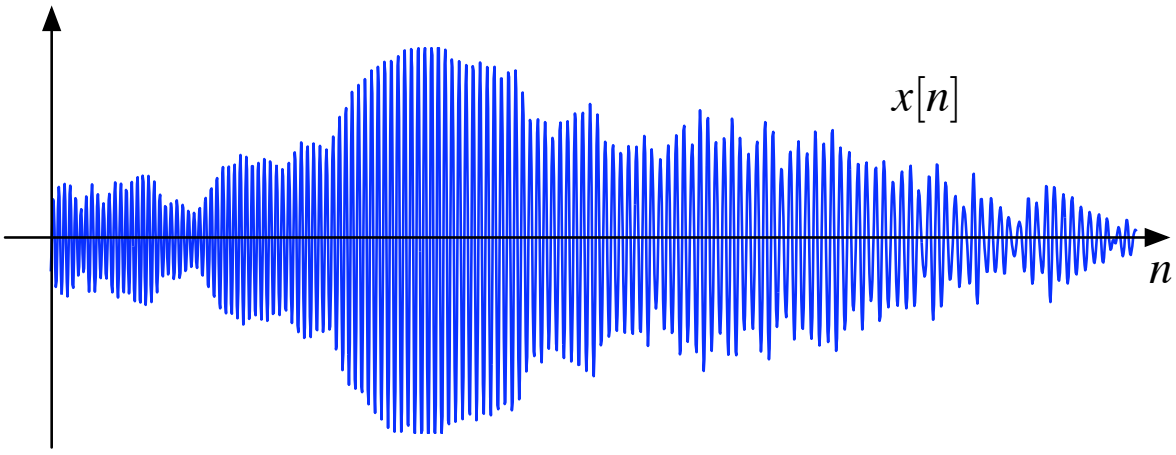


Figure 1: A segment of a sampled bird song.

For a long signal, there would be a tremendous number of plots that would have to be examined, and this doesn't convey the information very effectively. Instead, the most common presentation is to display the information in an image format, with frequency on the vertical axis, time on the horizontal axis, and the value of a particular pixel being the magnitude of the spectra at that time and frequency. Since we are assuming that the signal is real, the magnitude

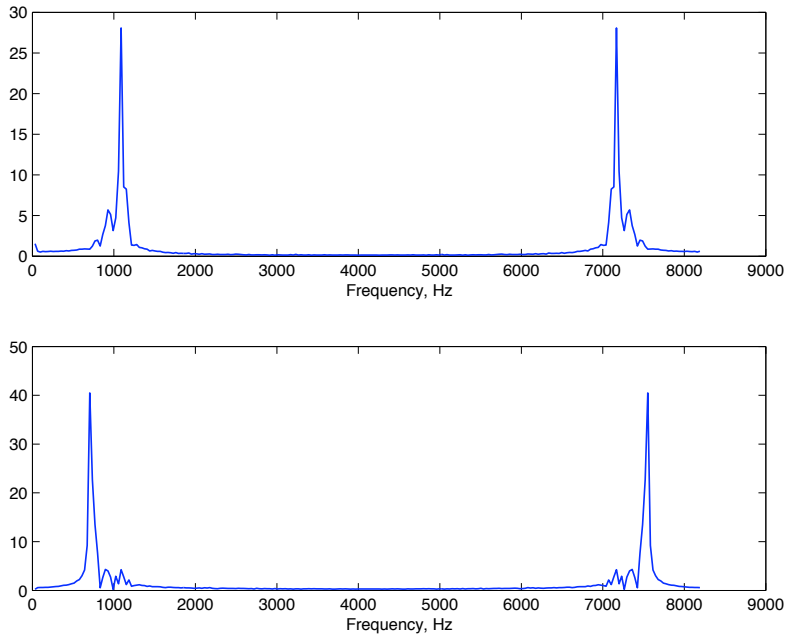


Figure 2: The spectra from two 256 sample blocks of the signal from Fig. 1

of the spectrum is symmetric about the origin. We only display the positive frequencies. This presentation is known as a spectrogram.

An example of this type of plot for the signal that Fig. 1 was taken from is shown in Fig. 1. A matlab routine that takes a two-dimensional array `y` and makes an image of the log-magnitude is given below:

```
function sg_plot(tl,fl,y)
%
%  function sg_plot(tl,fl,y)
%
%  Plot an image of the spectrogram y, with the axis labeled with time tl,
%  and frequency fl
%
%    tl -- time axis label, nt samples
%    fl -- frequency axis label, nf samples
%    y  -- spectrogram, nf by nt array
%

% find the maximum
mx = max(max(abs(y)));

% compute 20*log magnitude, scaled to the max
yl = 20*log10(abs(y/mx)+eps);

% show 60 dB of dynamic range
dbf = 60;
```

```
image(tl,fl,64*(yl+dbf)/dbf); axis xy; colormap('gray');

xlabel('Time, s');
ylabel('Frequency, Hz');
```

It is available on the class web site as `sg_plot.m`. The only parameter of note is `dbf` which determines the dynamic range in dB that will be presented. It is hard-coded to 60 dB, or a factor of 1000. This allows you to see small spectral components without seeing all of the background noise, and is reasonably suitable for our purposes here. For other applications with higher or lower noise levels, you may want to change this.

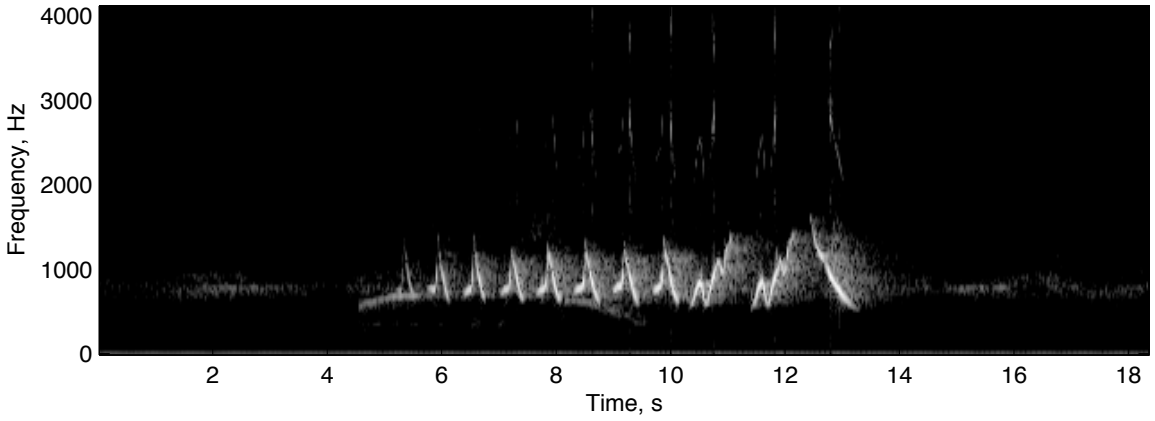


Figure 3: A segment of a sampled bird song.

Several different sound files are available on the class web site to for you to work with. They are all in the matlab file `sounds.mat`, in the matlab variables `s1`, `s2`, `s3`, and `s4`. These are

- `s1` A bird call.
- `s2` A squeaking door.
- `s3` An orca whale call.
- `s4` A sound effect.

Play each of these with the matlab `sound()` function, and try to visualize what you would expect the spectrum to look like as a function of time. In each case, the sampling rate is 8192 Hz.

Task 1: Computing a simple spectrogram.

For our first attempt at making a spectrogram, we will simply take the original signal of length N, and split it into blocks of length M. We then compute the DFT of each block. This corresponds to using a rectangular window to extract each block from the signal, as is illustrated in Fig. 4.

Write a matlab mfile that computes the spectrogram for a signal. If the original signal is a vector `x`, It should

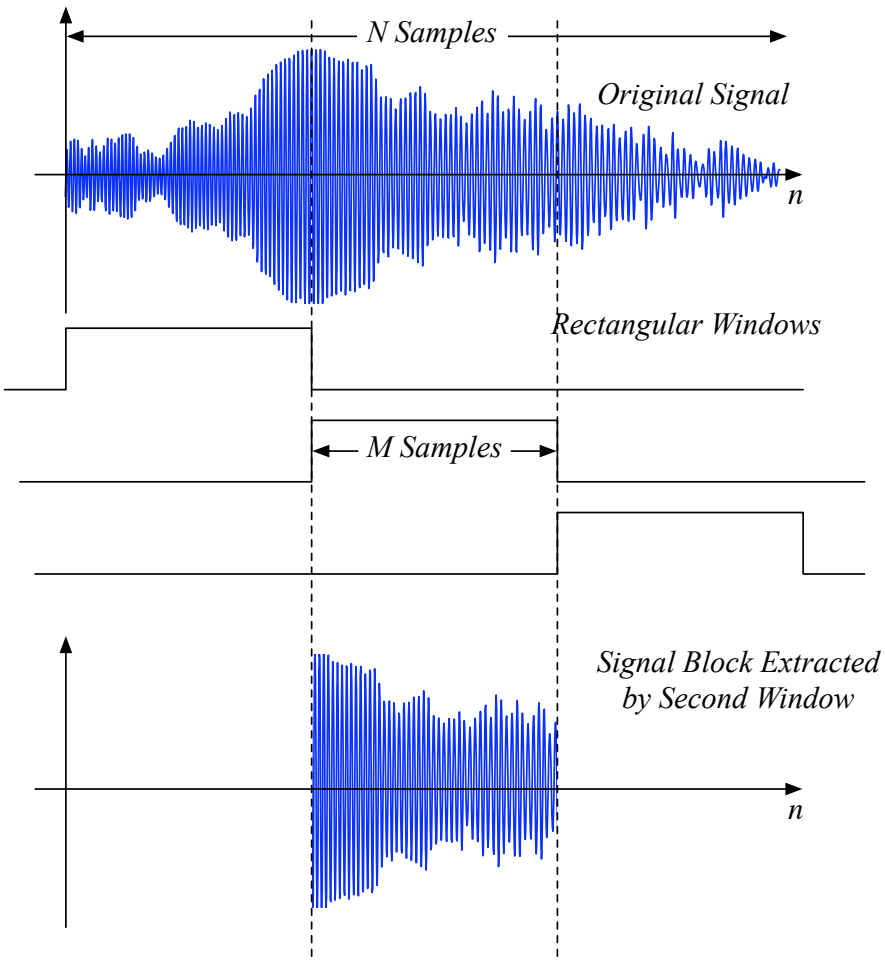


Figure 4: Extracting a segment of a long signal with a rectangular window

- Break the signal up into M-sample blocks, stored in the columns of a 2D matrix `xm`. This may require padding the signal with zeros, so that the length is a multiple of the block size.
- Apply the fft to the matrix `xmf = fft(xm)`. The `fft()` operates on each column of the matrix. It does not do a 2D FFT.
- Compute vectors for the time `tl` and frequency `fl` labels for each row and column.
- Call the `sg_plot(tl,fl,xmf(1:m/2,:))`, where we are only plotting the positive frequencies.

Your routine should be invoked with

```
>> myspectrogram(x,m)
```

where `x` is the signal, and m is the block size. Assume a sampling rate of 8192 Hz.

One matlab trick that can help you here is the `reshape` command. To create the `xm` matrix, first zero pad `x` to be a multiple of `m` in length,

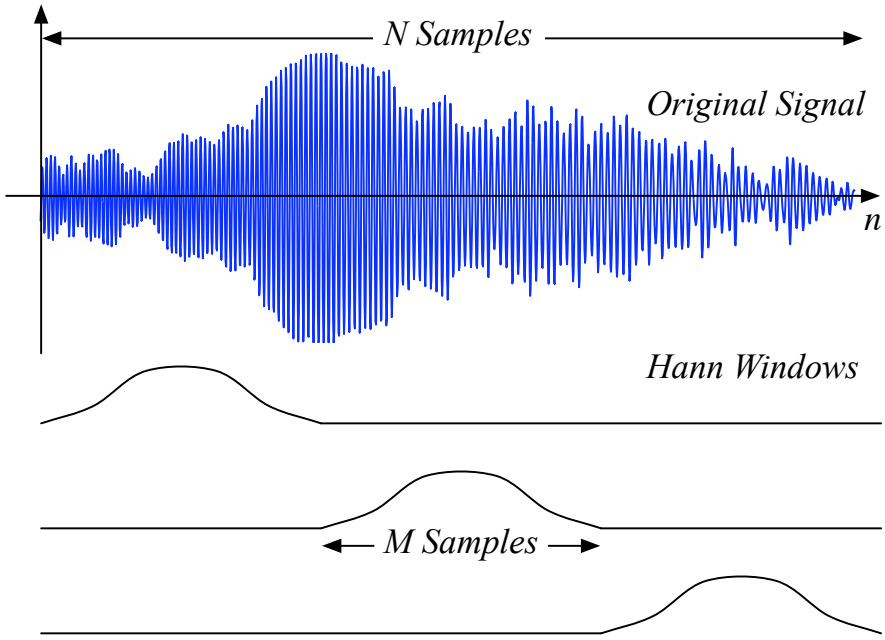


Figure 5: Extracting segments of a long signal with a Hann window

```
>> %
>> % Pad x up to a multiple of m
>> %
>> lx = length(x);
>> nt = ceil(lx/m);
>> xp = zeros(1,nt*m);
>> xp(1:lx) = x;
>> %
>> % use reshape to make it an m by nt  matrix
>> %
>> xm = reshape(xp,m,nt);
```

Each column of `xm` is one block of `x`.

To compute the time and frequency vectors, recall that the DFT frequencies go from 0 to the sampling frequency 8192 Hz in steps of 8192/M Hz. We are only going to plot the positive frequencies, so

```
>> fl = [0:(m/2-1)]*8192/m;
```

The time of a particular block is the period of one sample, 1/8192 seconds, multiplied by the number of samples in the block. If there are `nt` blocks, the time vector is

```
>> tl = [1:nt]*m/8192;
```

Try your spectrogram routine with the bird call sound file, `s1`, with a block size of 256 samples. It should look vaguely reminiscent of Fig. 3. However, there is noticeable streaking in the spectral dimension. Include a copy of this spectrogram in your report.

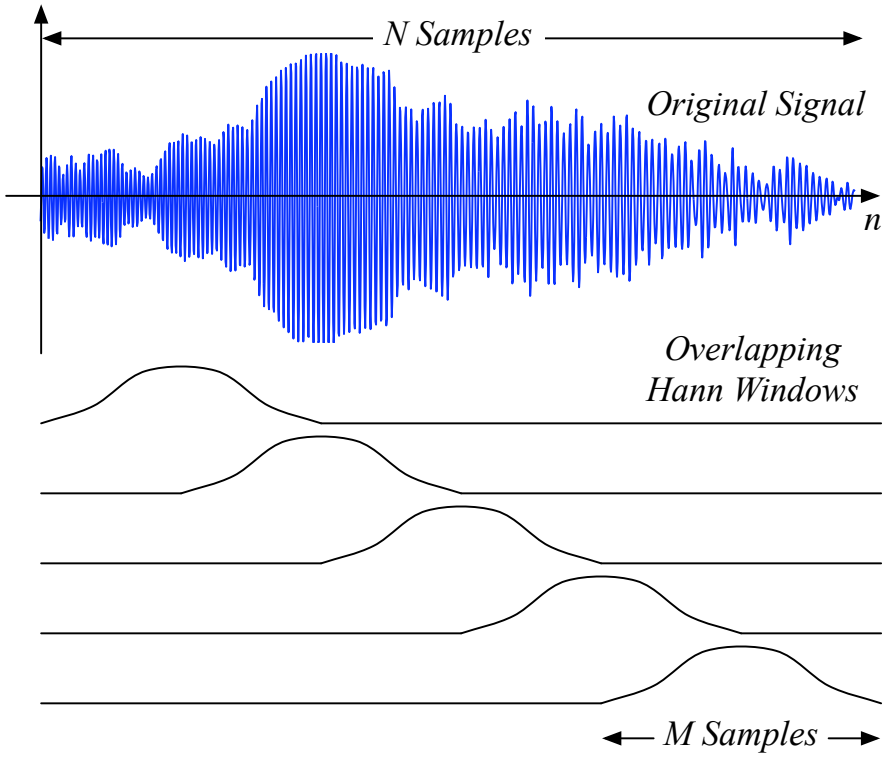


Figure 6: Extracting segments of a long signal with a Hann window using overlapping windows.

Task 2: A better spectrogram.

The problem with the spectrogram from task 1 is that we have used a square window to cut out blocks of the signal. In the spectral domain this corresponds to convolving a sinc with the spectra. The sinc sidelobes are high, and fall off slowly. We need to do something a little more gentle with the data.

To improve our spectrogram, we will extract each block of data with a Hann window. We can do this by multiplying our `xm` matrix with a matrix that has a Hann window along its columns,

```
>> xmw = xm.*(hann(m)*ones(1,nt));
```

The `.*` product is a element by element product (dot product). The `(hann(m)*ones(1,nt))` term is a matrix that has the same Hann window in each of the `nt` columns. Incorporate this into your `myspectrogram()` m-file. Note that matlab's `hann()` function returns a column vector. If you wrote your own, this may or may not be true, and you may need a transpose here.

Try your m-file on the bird song again. Include a copy of this spectrogram in your report. This time it should look very similar to the Fig. 3 figure. The location and size of the windows is shown in Fig. 5. Note that parts of the signal are effectively not used, at the ends of each window. This can result in discontinuities in the temporal direction of the spectrogram.

Task 3: An even better spectrogram!

As a final enhancement, we will overlap the Hann windows, as shown in Fig. 6. Each block of the signal, and column of the `xm` matrix overlaps the adjacent blocks by 50%. There are a number

of ways to do this in matlab. One is to replicate each half block in `xp` before the reshape. Another direct approach would be to program it in a `for` loop. Note that you also have to modify the time vector, since you have twice as many time samples.

Try this on the bird song. it should look much smoother than the previous case. Include a copy of this spectrogram in your report.

Task 4: Time and frequency resolution.

So far we've just set the block size to 256 samples, which is about 31 ms of data. The frequency resolution is then 8192/256 = 32 Hz. Ideally we'd like to have both time and frequency resolution. However, one comes at the expense of the other. For example if we wanted to improve our frequency resolution to 16 Hz, it would increase the time between samples to 62 ms.

The optimum tradeoff between the two factors depends on the nature of the spectrum you are analyzing. A good example is the orca whale call, signal `s3`. This has segments that are very rapidly changing, and segments where the changes are slow. Try different block sizes, from 128, 256, 512, and 1024 samples. Note how the frequency resolution improves, while the temporal resolution degrades.

Task 5: Choose a signal to analyze.

Choose one of the other signals, sample a CD, or find something on the web. Find a reasonable block size that shows the structure of the spectrogram well. Include the spectrogram, as well as a description of what the signal is. Note that you can load ".wav" files with the `wavread()` matlab function. Lots of .wav files are available on the web. The signals `s2` and `s3` have beautiful spectrograms. We will give up to 10 pts extra credit for finding new sounds that are at least that interesting! Happy hunting.