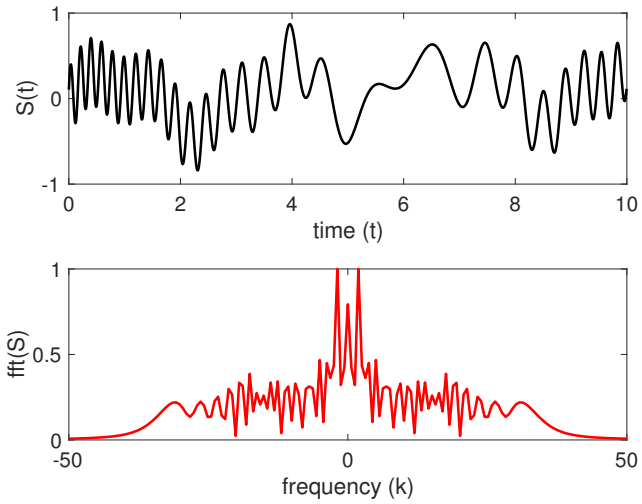
Lecture 7
Spectrograms and the Gabor Transform in MATLAB

Jason J. Bramburger

Now that we have developed some theory, let's see the Gabor transform in practice. In particular, we are going to see how we can visualize things in the time-frequency domain. Recall that the implementation is fairly straightforward - we just multiply the signal by a (discrete) window function (or filter) and then apply the FFT. We then move the window and repeat the process. For today's experiments let's use the example signal from Lecture 4. Let's start by plotting the signal and its Fourier transform.

```
L = 10; n = 2048;
t2 = linspace(0,L,n+1); t = t2(1:n);
k = (2*pi/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);

% Create signal
S = (3*sin(2*t) + 0.5*tanh(0.5*(t-3)) + 0.2*exp(-(t-4).^2) ...
    + 1.5*sin(5*t) + 4*cos(3*(t-6).^2))/10 + (t/20).^3;
St = fft(S);

% Plot signal in both time and frequency domain
figure(1)
subplot(2,1,1) % time domain
plot(t,S,'k','Linewidth',2)
set(gca,'Fontsize',16); xlabel('time (t)'); ylabel('S(t)')

subplot(2,1,2) % frequency domain
plot(ks,abs(fftshift(St))/max(abs(St)),'r','Linewidth',2); axis([-50 50 0 1])
set(gca,'Fontsize',16)
xlabel('frequency (k)'), ylabel('fft(S)')
```



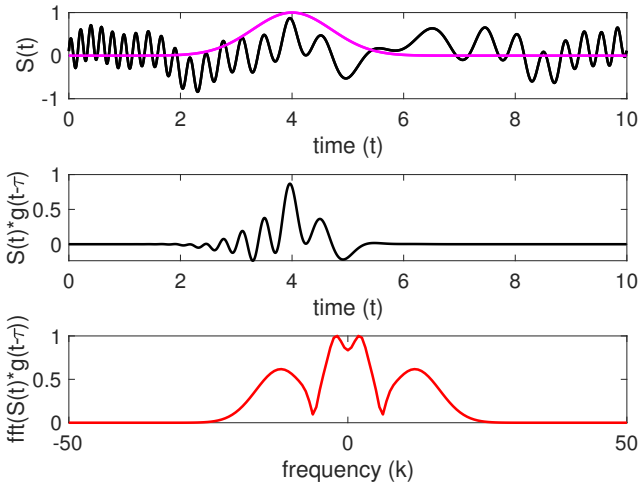
You will remember that there are many choices for the window function $g(t-\tau)$. We will keep it simple and use a Gaussian:

$$g(t-\tau) = \mathrm{e}^{-a(t-\tau)^2}.$$

We have two parameters: $a > 0$ and τ, representing the width of the window and the centre of the window,

respectively. Notice that small a means a wide window and large a means a thin window. Let's start by fixing $a = 1$ and $\tau = 4$ to consider a single window.

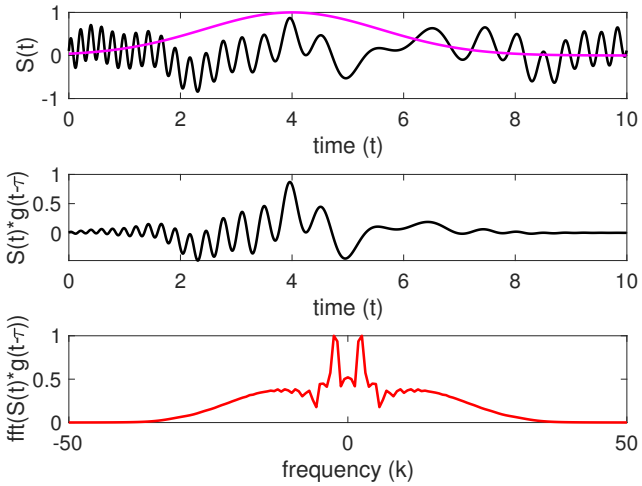
```
% Window function specs
tau = 4;
a = 1;
g = exp(-a*(t - tau).^2);

% Windowed signal
Sg = g.*S;
Sgt = fft(Sg);

figure(2)
subplot(3,1,1) % Time domain
plot(t,S,'k','Linewidth',2)
hold on
plot(t,g,'m','Linewidth',2)
set(gca,'Fontsize',16), xlabel('time (t)'), ylabel('S(t)')

subplot(3,1,2) % Time domain
plot(t,Sg,'k','Linewidth',2)
set(gca,'Fontsize',16), xlabel('time (t)'), ylabel('S(t)*g(t-\tau)')

subplot(3,1,3) % Fourier domain
plot(ks,abs(fftshift(Sgt))/max(abs(Sgt)),'r','Linewidth',2); axis([-50 50 0 1])
set(gca,'Fontsize',16), xlabel('frequency (k)'), ylabel('fft(S(t)*g(t-\tau))')
```



Now, let's see what happens as we change the width of the window. With a wider window we can get better frequency resolution, but we won't be able to localized the signal in time.

```
% Window function specs
tau = 4;
a = 0.2;
g = exp(-a*(t - tau).^2);

% Windowed signal
Sg = g.*S;
Sgt = fft(Sg);

figure(3)
subplot(3,1,1) % Time domain
plot(t,S,'k','Linewidth',2)
```

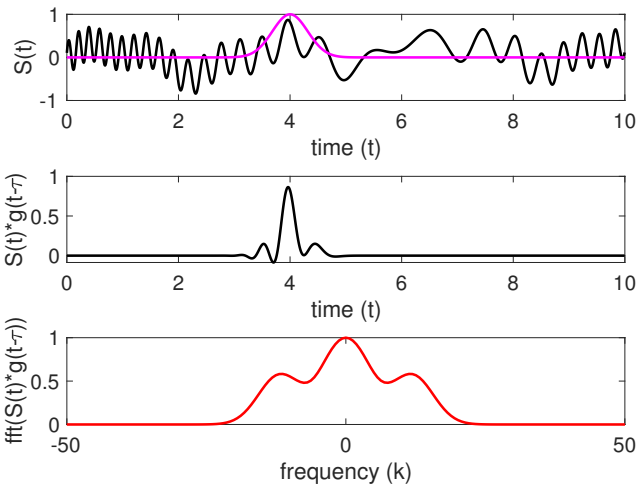
```
hold on
plot(t,g,'m','Linewidth',2)
set(gca,'Fontsize',16), xlabel('time (t)'), ylabel('S(t)')

subplot(3,1,2) % Time domain
plot(t,Sg,'k','Linewidth',2)
set(gca,'Fontsize',16), xlabel('time (t)'), ylabel('S(t)*g(t-\tau)')

subplot(3,1,3) % Fourier domain
plot(ks,abs(fftshift(Sgt))/max(abs(Sgt)),'r','Linewidth',2); axis([-50 50 0 1])
set(gca,'Fontsize',16), xlabel('frequency (k)'), ylabel('fft(S(t)*g(t-\tau))')
```



Similarly, if we make the window thinner, we are localizing in time, but then get poor resolution in frequency space.

```
% Window function specs
tau = 4;
a = 5;
g = exp(-a*(t - tau).^2);

% Windowed signal
Sg = g.*S;
Sgt = fft(Sg);

figure(4)
subplot(3,1,1) % Time domain
plot(t,S,'k','Linewidth',2)
hold on
plot(t,g,'m','Linewidth',2)
set(gca,'Fontsize',16), xlabel('time (t)'), ylabel('S(t)')

subplot(3,1,2) % Time domain
plot(t,Sg,'k','Linewidth',2)
set(gca,'Fontsize',16), xlabel('time (t)'), ylabel('S(t)*g(t-\tau)')

subplot(3,1,3) % Fourier domain
plot(ks,abs(fftshift(Sgt))/max(abs(Sgt)),'r','Linewidth',2); axis([-50 50 0 1])
set(gca,'Fontsize',16), xlabel('frequency (k)'), ylabel('fft(S(t)*g(t-\tau))')
```

In order to proceed, we need to fix a width. Out of the three that we looked at, $a = 1$ struck the best balance, so let's try use that window size. We are going to slide the window across the domain and calculate the FFT at each shift. We will start with the window on the left side of the domain and slide it across by $t = 0.1$ each time.

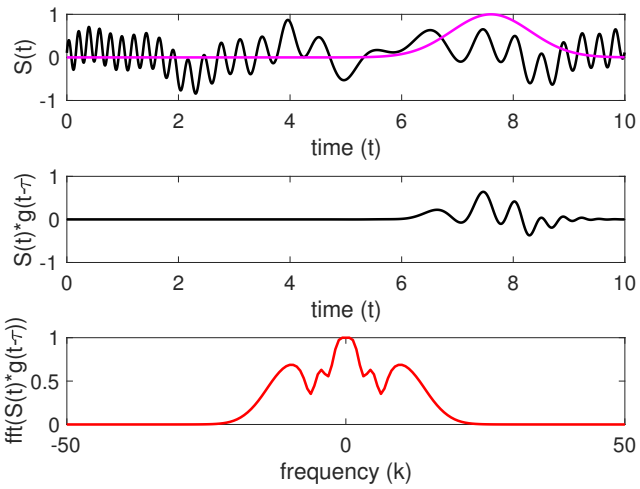



```
figure(5)
a = 1;
tau = 0:0.1:10;

for j = 1:length(tau)
    g = exp(-a*(t - tau(j)).^2); % Window function
    Sg = g.*S;
    Sgt = fft(Sg);

    subplot(3,1,1) % Time domain
     plot(t,S,'k','Linewidth',2)
     hold on
     plot(t,g,'m','Linewidth',2)
     set(gca,'Fontsize',16), xlabel('time (t)'), ylabel('S(t)')

     subplot(3,1,2) % Time domain
     plot(t,Sg,'k','Linewidth',2)
     set(gca,'Fontsize',16,'ylim',[-1 1]), xlabel('time (t)'), ylabel('S(t)*g(t-\tau)')

     subplot(3,1,3) % Fourier domain
     plot(ks,abs(fftshift(Sgt))/max(abs(Sgt)),'r','Linewidth',2); axis([-50 50 0 1])
     set(gca,'Fontsize',16), xlabel('frequency (k)'), ylabel('fft(S(t)*g(t-\tau))')
     drawnow
     pause(0.1)
     clf
end
```

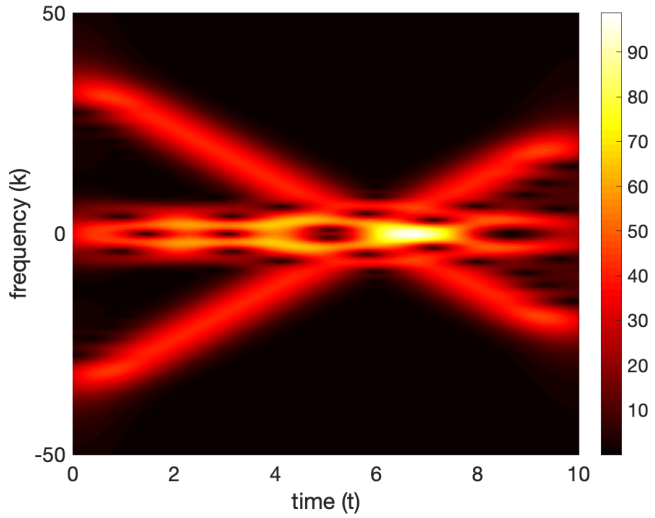
Spectrogram

Above we used a video to illustrate the sliding window over the domain, but this isn't a great way to convey information, especially when it comes to writing up a report on your findings. Instead we could stack all the Fourier transforms next to each other to make a plot with the horizontal direction being the value of τ (window centre) and the vertical direction the frequency. This is called a **spectrogram**. This will give us a still that represents the changing of the Fourier transform as the window slides over the domain. This is also quite easy to create in MATLAB since we just need to create a 2D array.

```
a = 1;
tau = 0:0.1:10;

for j = 1:length(tau)
    g = exp(-a*(t - tau(j)).^2); % Window function
    Sg = g.*S;
    Sgt = fft(Sg);
```



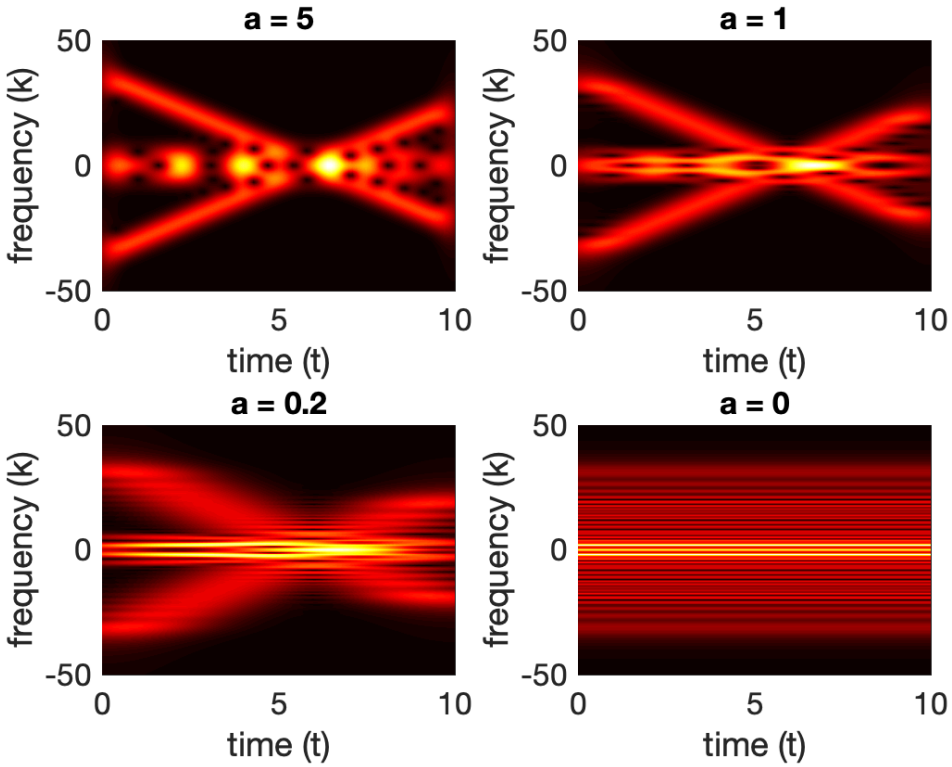

```
    Sgt_spec(:,j) = fftshift(abs(Sgt)); % We don't want to scale it
end

figure(6)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[-50 50],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (k)')
```



Taking $a = 1$ was a bit of an arbitrary choice, and now with spectrograms we can easily compare across different window sizes. We will look at the values $a = 0.2, 1, 5$, just like earlier. Let's also compare with the usual Fourier transform of the whole signal, representing an infinite width window ($a = 0$).

```
figure(7)

a = [5 1 0.2];
tau = 0:0.1:10;
for jj = 1:length(a)
    Sgt_spec = []; % Clear at each loop iteration
    for j = 1:length(tau)
        g = exp(-a(jj)*(t - tau(j)).^2); % Window function
```

```
        Sg = g.*S;
        Sgt = fft(Sg);
        Sgt_spec(:,j) = fftshift(abs(Sgt));
    end

    subplot(2,2,jj)
    pcolor(tau,ks,Sgt_spec)
    shading interp
    set(gca,'ylim',[-50 50],'Fontsize',16)
    colormap(hot)
    %colorbar
    xlabel('time (t)'), ylabel('frequency (k)')
    title(['a = ',num2str(a(jj))],'Fontsize',16)
end

% Add in the Fourier transform of the whole signal
Sgt_spec = [];
Sgt_spec = repmat(fftshift(abs(St)),length(tau),1);
subplot(2,2,4)
pcolor(tau,ks,Sgt_spec'),
shading interp
set(gca,'ylim',[-50 50],'Fontsize',16)
colormap(hot)
%colorbar
xlabel('time (t)'), ylabel('frequency (k)')
title('a = 0','Fontsize',16)
```



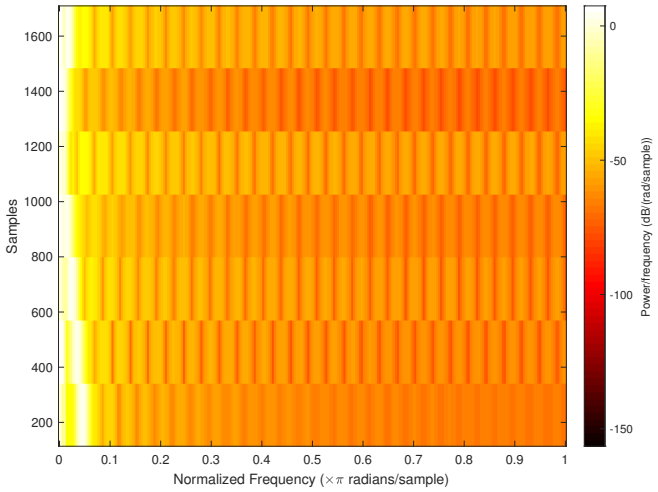
Notice that $a = 5$ has the best time resolution. This means that you can tell with the most precision where those bright, low frequency spots are located in time. You can also see how the high frequency content from the beginning of the signal decays in time. However, there is not much resolution in frequency space. You can tell by considering vertical slices. We see that for $a = 1$ we have slightly better resolution in the frequency domain, but things are blurrier in time. The trend continues for $a = 5$. For $a = 0$ we just have the Fourier transform (precisely, the FFT), meaning there is no localization in time. Hence, there is no change over time, but you can see that the frequency resolution is the best.

MATLAB's Built-In Spectrogram Function

Like most numerical routines, MATLAB has its own built-in spectrogram function that makes use of the Gabor transform. In what follows we will learn how to use this functionality. It turns out that it isn't that easier than just building your own routine as we did above - especially since the above code now functions as a sample for you to use later.

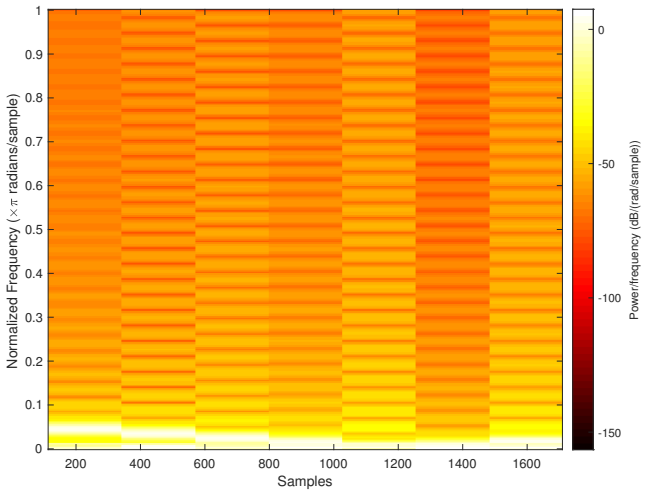
Note: On your homework you are expected to build your own spectrogram like we did above.

```
figure(8)
spectrogram(S)
colormap(hot)
```



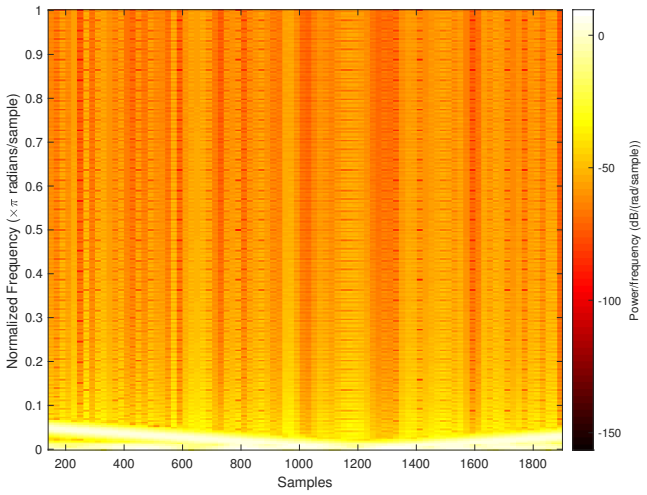
A quick visual inspection should worry you a bit since this doesn't look anything like the spectrograms we created ourselves. There are several issues that we will walk through one-by-one. First, this has frequency on the x-axis and time on the y-axis. We can switch that by specifying where we want the frequency plotted.

```
spectrogram(S,'yaxis')
colormap(hot)
```



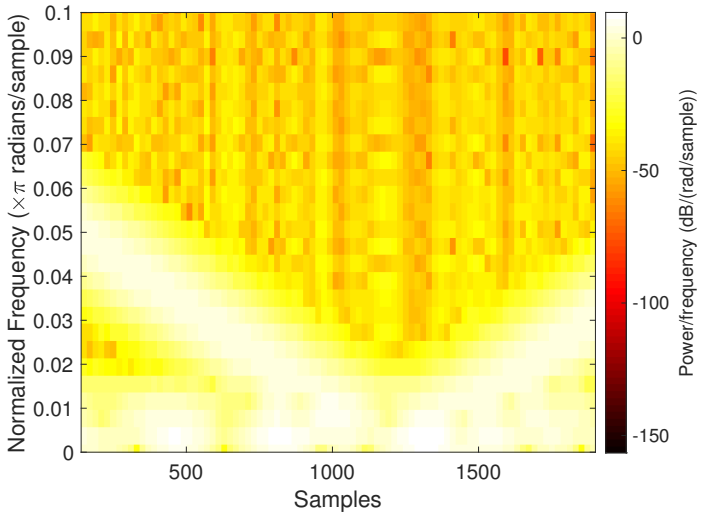
Alright, at least now the axes are arranged how I want them to be. Now if you look really hard you can see the right shape at the bottom of the plot. However, the resolution in the time domain is terrible. It even cuts off the left and right ends. That is because the filter is too wide and shifted too far each time. We can specify the size of the filter (in terms of number of points) and how much we want the filters to overlap (in numbers of points).

```
spectrogram(S,300,280,'yaxis')
colormap(hot)
```



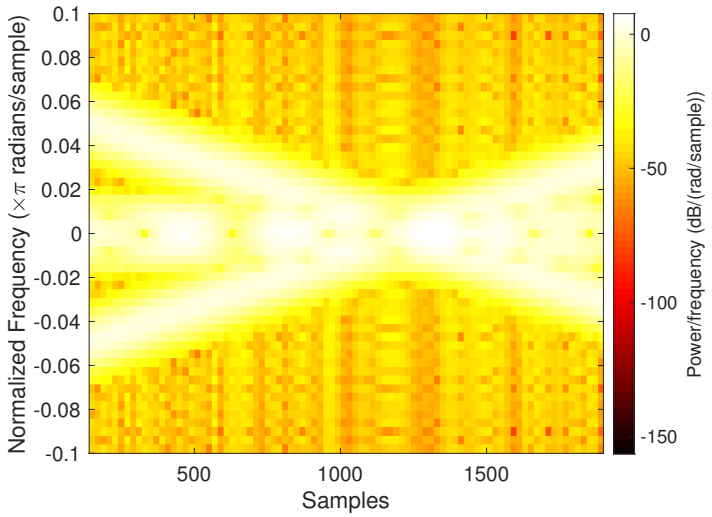
Let's zoom in a bit now.

```
spectrogram(S,300,280,'yaxis')
colormap(hot)
set(gca,'Ylim',[0 0.1],'Fontsize',16)
```



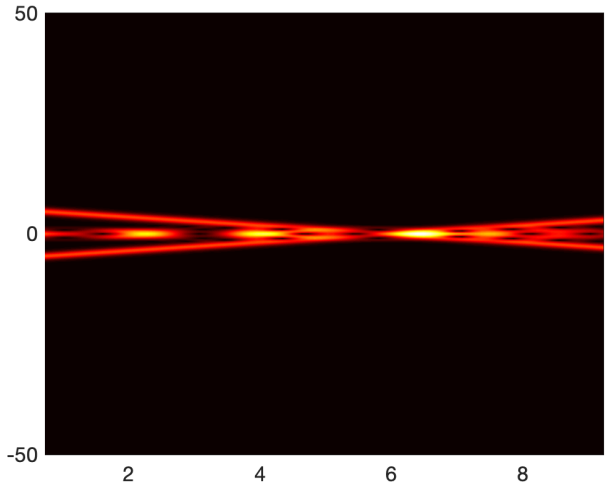
We can see another issue: we are only plotting positive frequencies. This isn't an arbitrary MATLAB decision. Since our signal is real-valued, only the positive frequencies are needed since the coefficient on the terms with negative frequencies are just the complex conjugate of the corresponding positive frequency. Of course, a complex number and its conjugate have the same absolute value, explaining the symmetry in the spectrograms above. We can use the option 'centered' to get the negative frequencies too.

```
spectrogram(S,300,280,'centered','yaxis')
colormap(hot)
set(gca,'Ylim',[-0.1 0.1],'Fontsize',16)
```



We are getting closer to our spectrogram, but the axes are off. Particularly, MATLAB automatically scales the frequencies so that the max is 1. We want to get rid of this, as well as change the time axis to have units of time instead of just counting the sample number. The 'spectrogram' function can output this information for us.

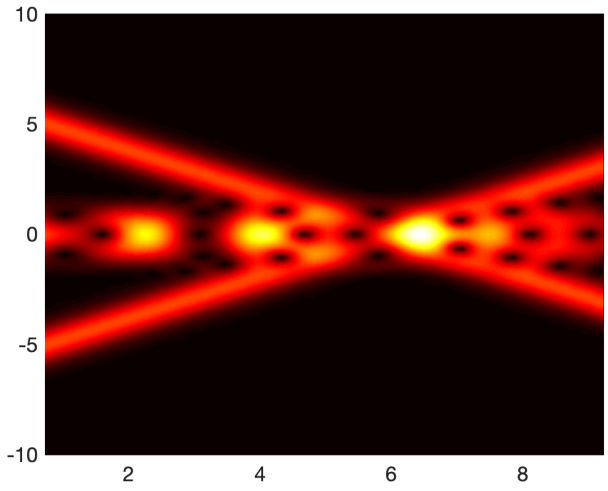
```
[Sp, w, times] = spectrogram(S,300,280,-50:0.01:50,n/L);
pcolor(times,w,abs(Sp))
shading interp
colormap(hot)
set(gca,'ylim',[-50,50],'Fontsize',16)
```



Now we can really recognize our spectrogram, except that the frequency scaling is off. If we reduce the bounds on the frequency axis, we get something that looks like our spectrogram.

```
[Sp, w, times] = spectrogram(S,300,280,-50:0.01:50,n/L);
pcolor(times,w,abs(Sp))
shading interp
colormap(hot)
```

```
set(gca,'ylim',[-10,10],'Fontsize',16)
```



Okay, this looks better but the frequency axis seems to be scaled differently in our plot. This has to do with the units we've been using for frequency. This is because MATLAB scales out the 2π in the periods of oscillation to make the periods integers. That is,

$$e^{ikt}, \quad k \in \mathbb{Z}$$

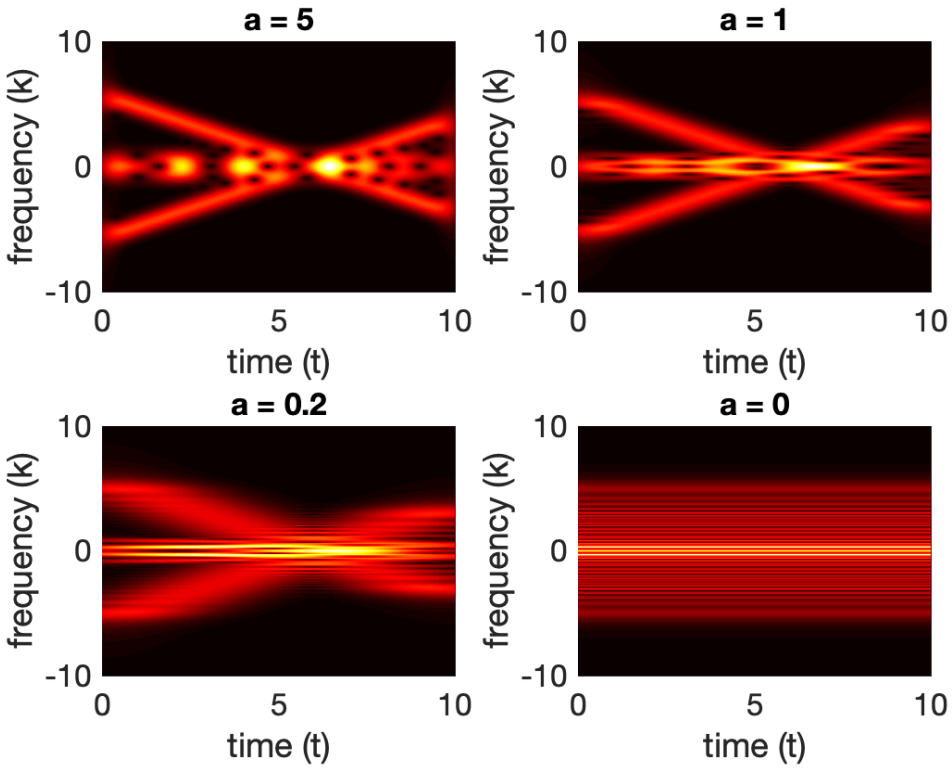
has frequency (inverse of the period) $|k|/2\pi$, while

$$e^{i2\pi ft}, \quad f \in \mathbb{Z}$$

has frequency $|f|$. MATLAB is using the latter, while we have been using the former. They both represent frequencies, but they have different units. With the notation above, f is measured in Hertz, while k is sometimes called the angular frequency. Therefore, if we go back and rescale our k in the home-made spectrograms by $1/L$ instead of $2\pi/L$ we will get the same thing.

```
figure(9)

% Different scaling on k
k = (1/L)*[0:n/2-1 -n/2:-1]; % Notice the 1/L instead of 2*pi/L
ks = fftshift(k);

a = [5 1 0.2];
tau = 0:0.1:10;
for jj = 1:length(a)
    Sgt_spec = []; % Clear at each loop iteration
    for j = 1:length(tau)
         g = exp(-a(jj)*(t - tau(j)).^2); % Window function
         Sg = g.*S;
         Sgt = fft(Sg);
         Sgt_spec(:,j) = fftshift(abs(Sgt));
    end

    subplot(2,2,jj)
    pcolor(tau,ks,Sgt_spec)
    shading interp
    set(gca,'ylim',[-10 10],'Fontsize',16)
    colormap(hot)
    %colorbar
    xlabel('time (t)'), ylabel('frequency (k)')
    title(['a = ',num2str(a(jj))],'Fontsize',16)
end
```

```
27
28 % Add in the Fourier transform of the whole signal
29 Sgt_spec = [];
30 Sgt_spec = repmat(fftshift(abs(St)),length(tau),1);
31 subplot(2,2,4)
32 pcolor(tau,ks,Sgt_spec'),
33 shading interp
34 set(gca,'ylim',[-10 10],'Fontsize',16)
35 colormap(hot)
36 %colorbar
37 xlabel('time (t)'), ylabel('frequency (k)')
38 title('a = 0','Fontsize',16)
```



Note: This discussion of scaling frequencies is important for your second homework assignment because there is a problem that asks you to do a time-frequency analysis to determine which musical notes are being played. The scale used for the frequencies of the musical notes is Hertz, so you should scale your k vector by $1/L$ and not $2\pi/L$.