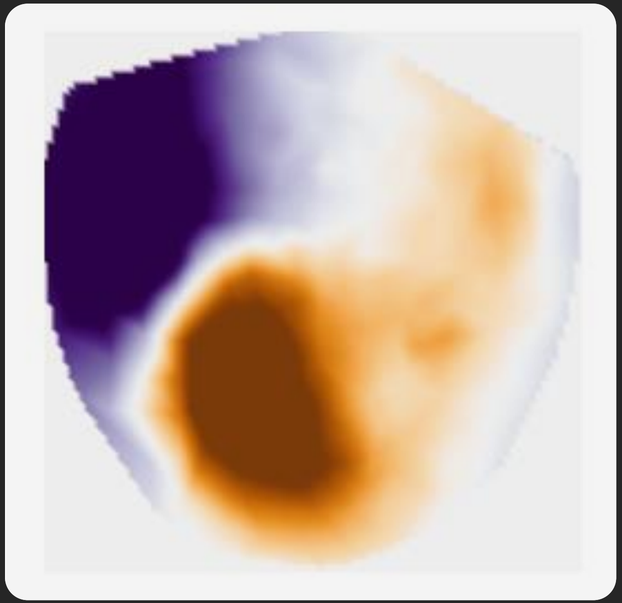
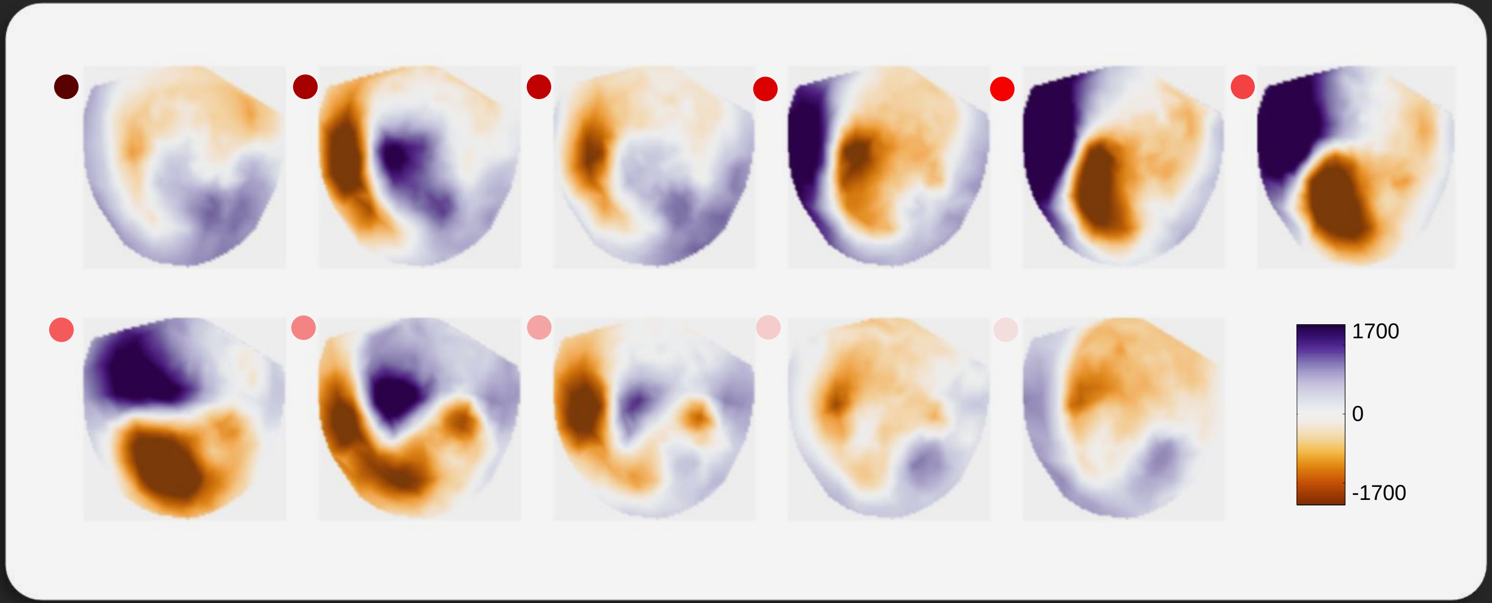
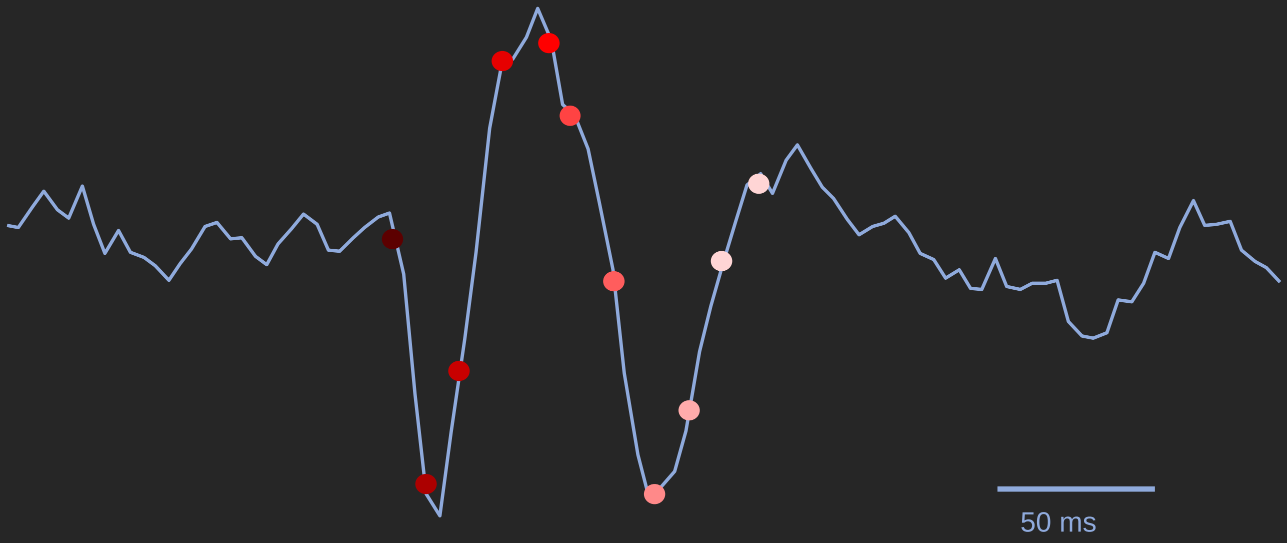
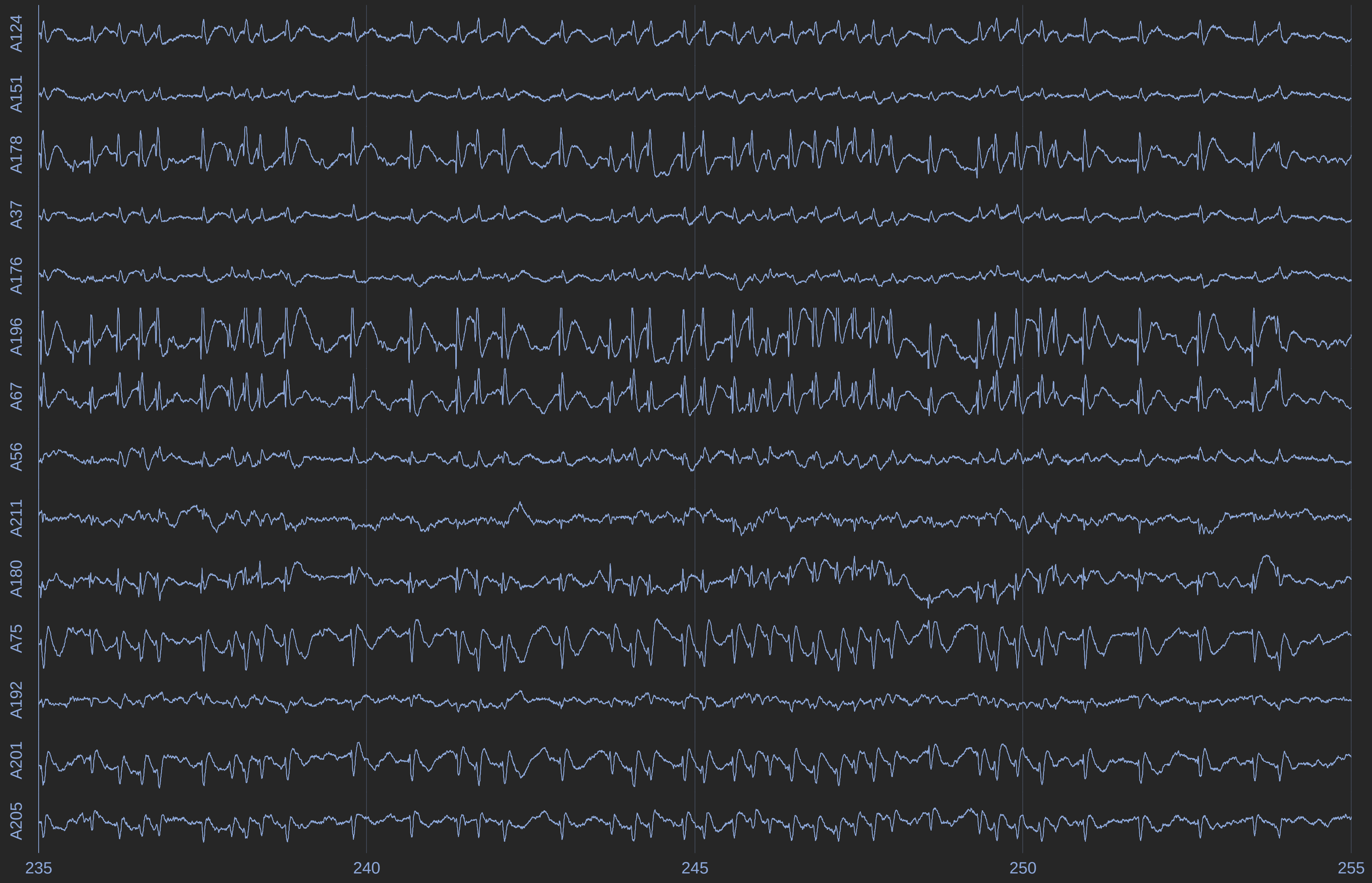
Filtering & Oscillations

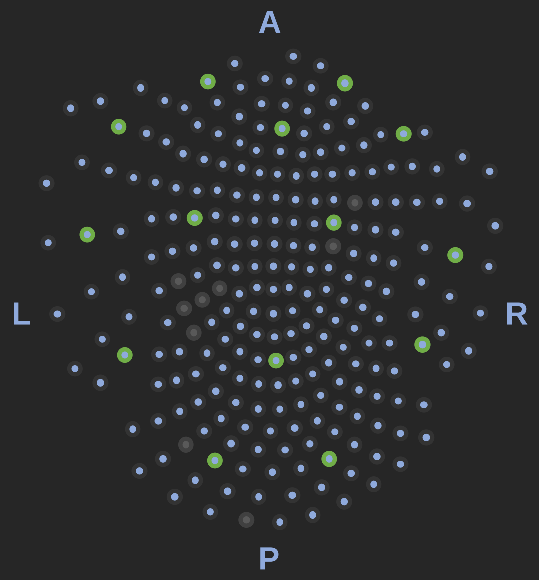
Tallie Adams

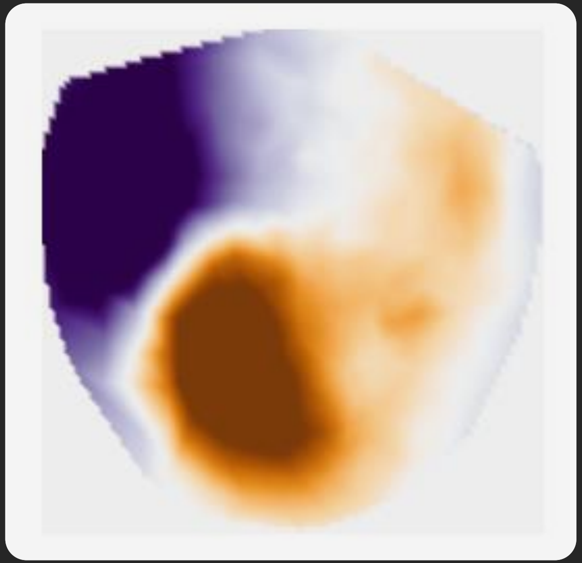



1700
0
-1700

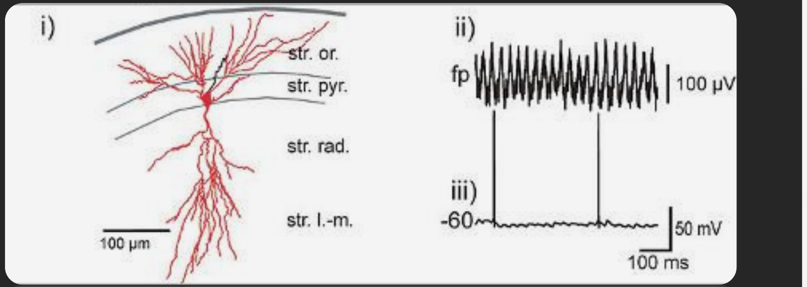

50 ms

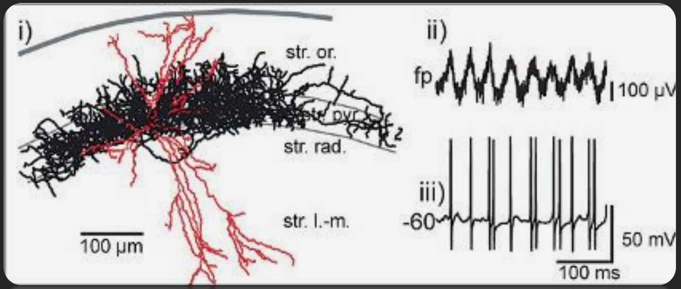

A124
A151
A178
A37
A176
A196
A67
A56
A211
A180
A75
A192
A201
A205
235
240
245
250
255

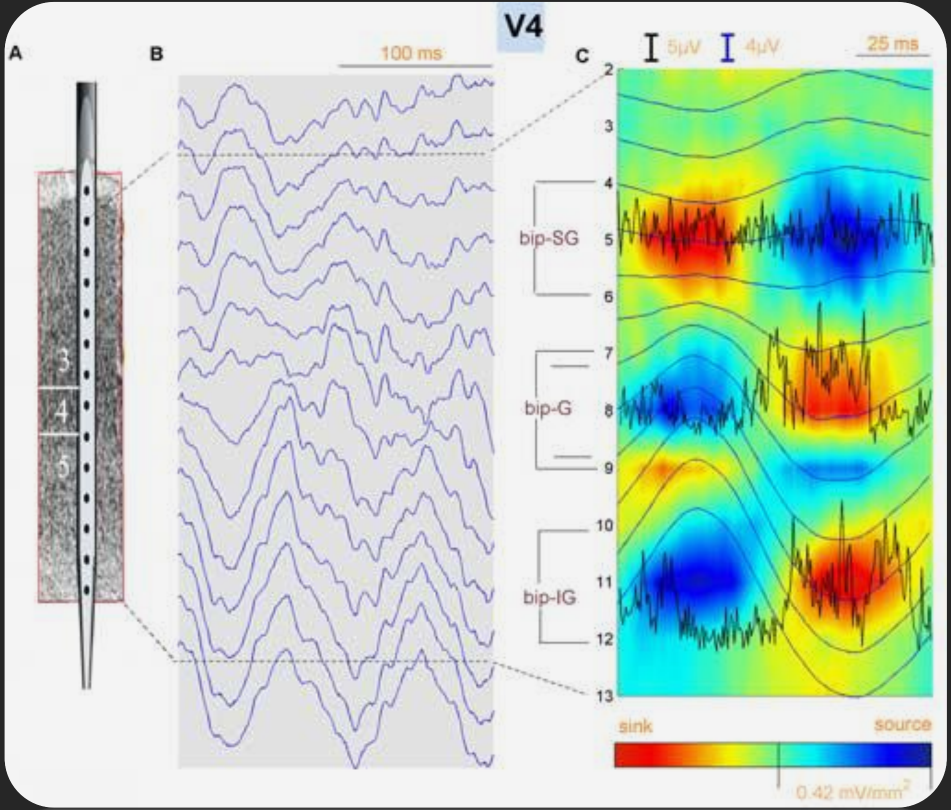

A
L
R
P

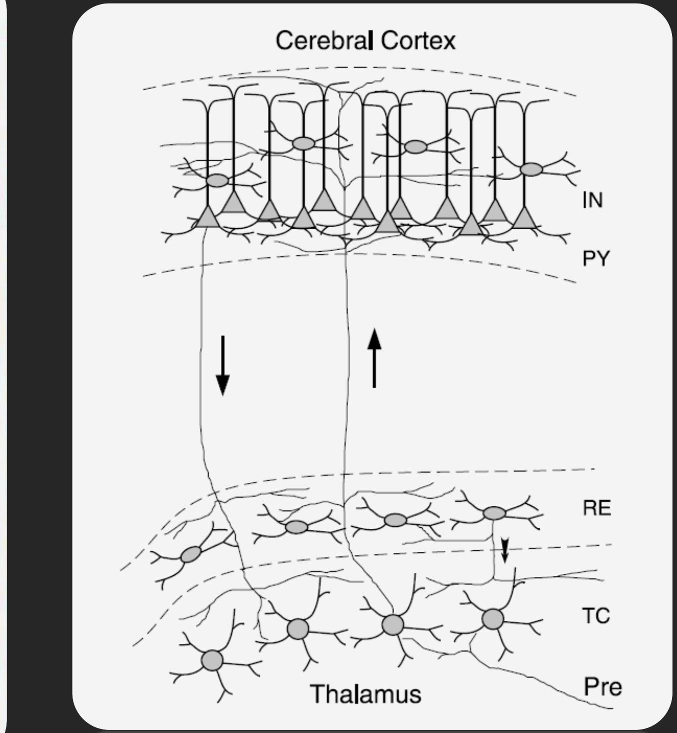


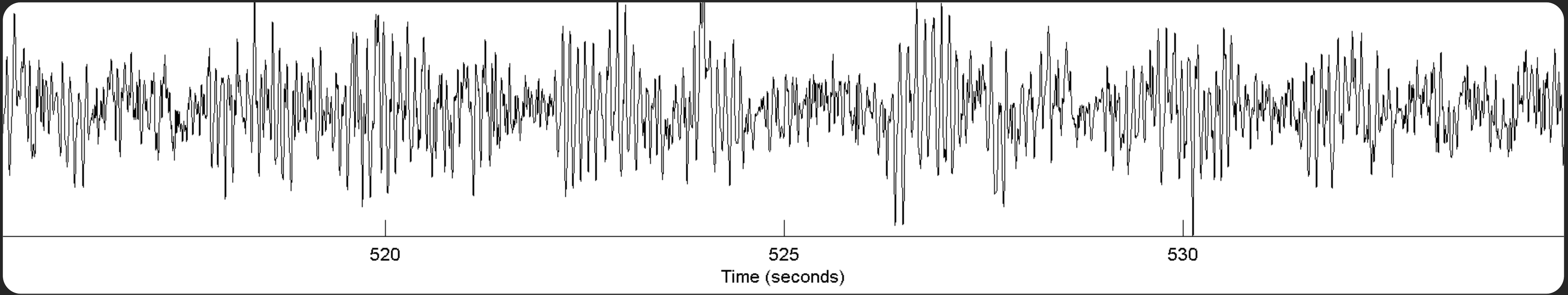
What is an oscillation

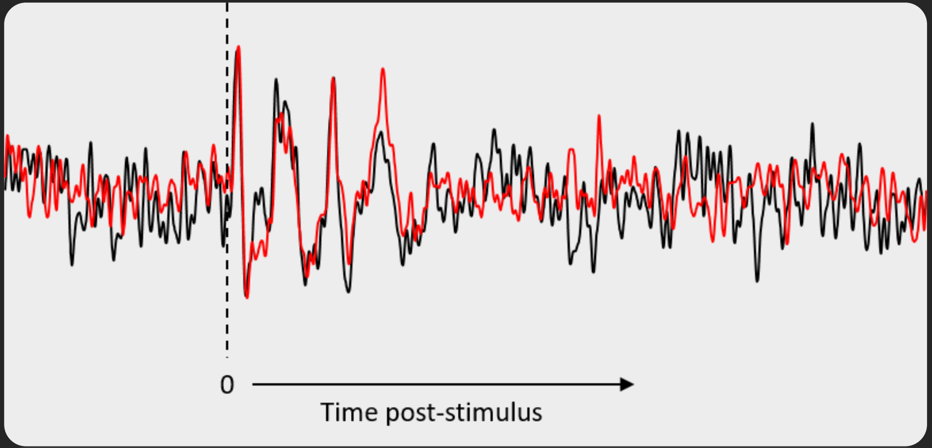









520
525
530
Time (seconds)

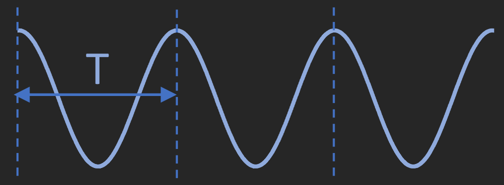

0
Time post-stimulus

- Oscillations are periodic signals. For an oscillation with period ‘T’:

$$f(t+T) = f(t)$$

‘f’ is the function that describes our oscillation

‘t’ is the dependent variable (time).



- For sine and cosine functions:

$$\sin(t+360) = \sin(t)$$

$$\cos(t+360) = \cos(t)$$

- MATLAB works in radians by default, so…

$$\sin(t+2\pi) = \sin(t)$$

$$\cos(t+2\pi) = \cos(t)$$

- To convert between radians and degrees in MATLAB:

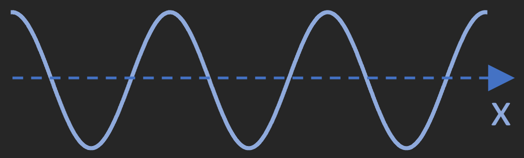
360° is equivalent to 2*π

$$x(rads) = x * \frac{360}{2\pi} = x * \frac{180}{\pi}$$

f(x) = a*sin(b*x + c)

f(x) = a*cos(b*x + c)

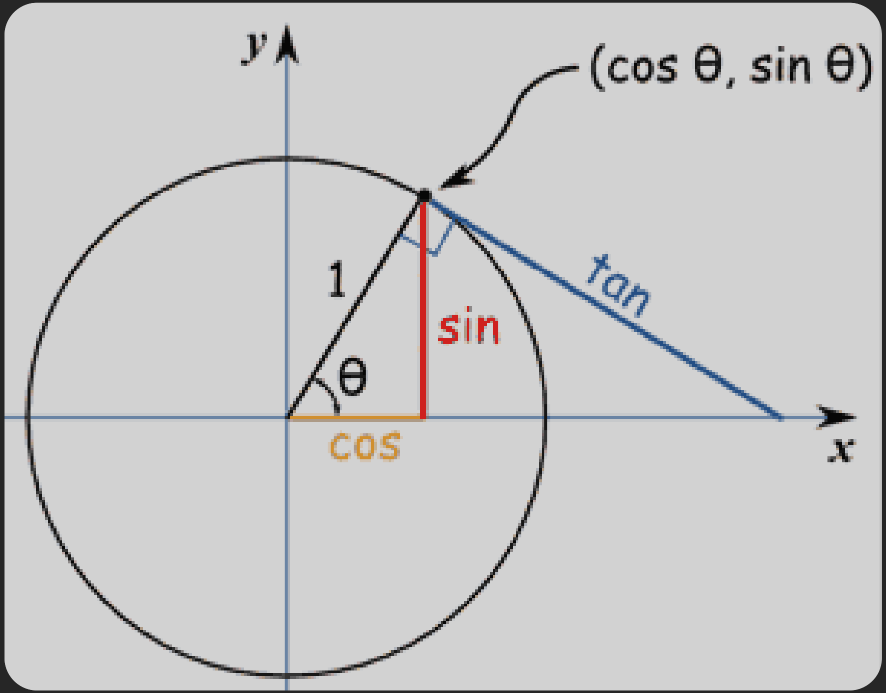
tan(x) = sin(x)/cos(x)

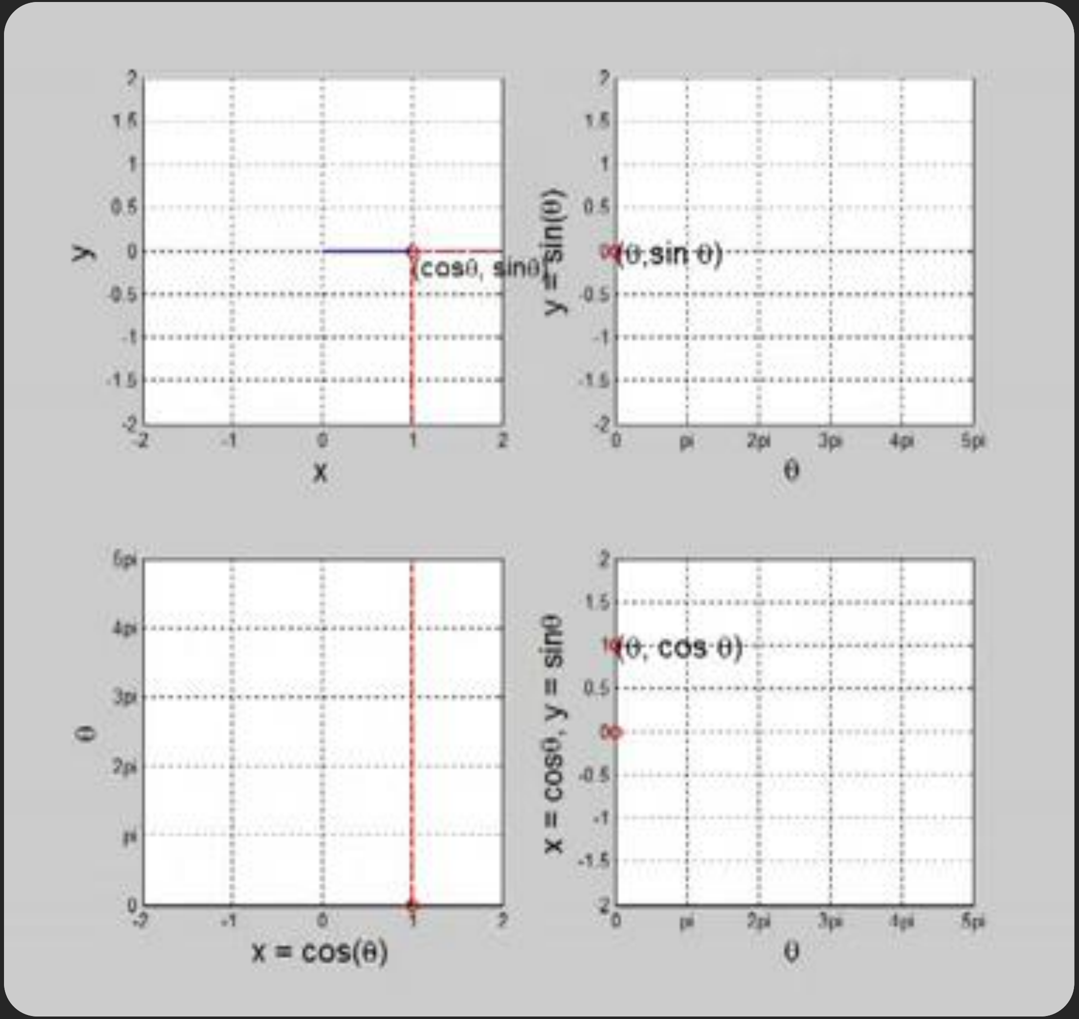


a = amplitude

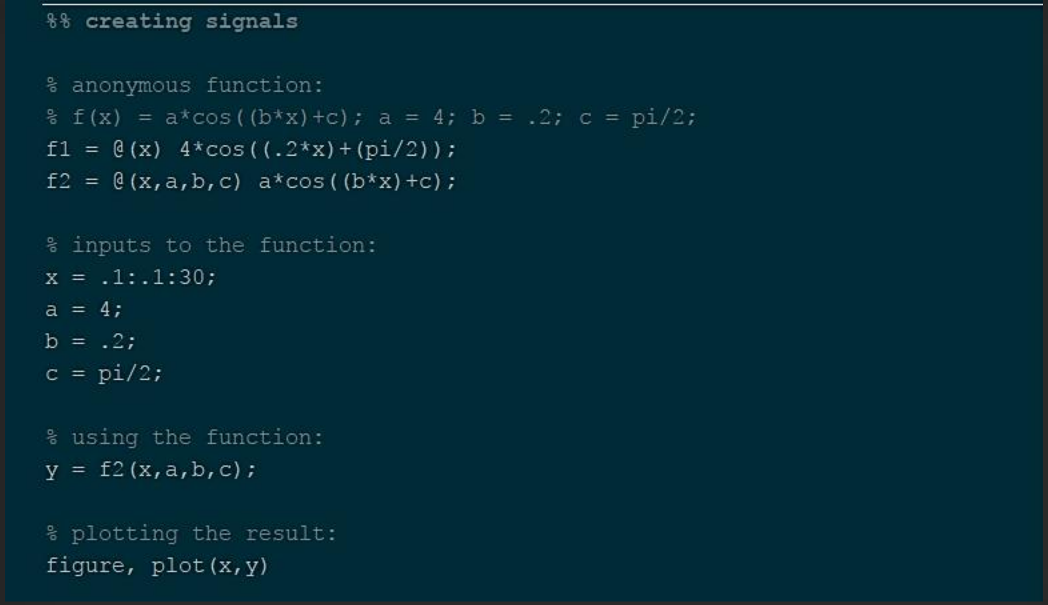
b = frequency

c = phase

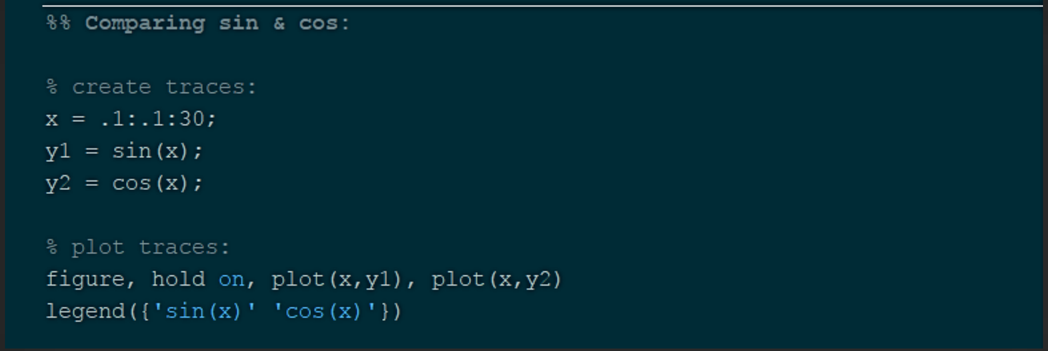




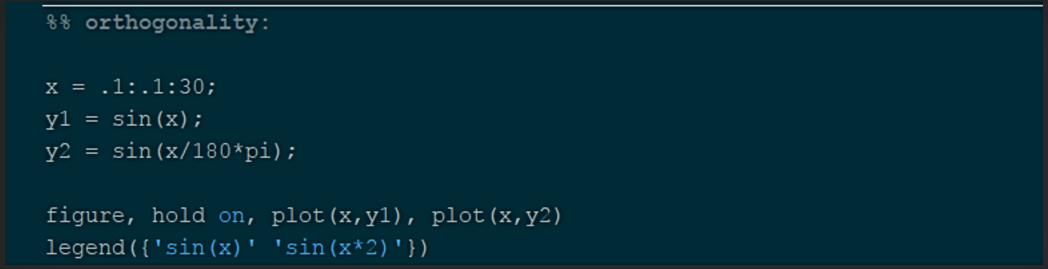
Matlab



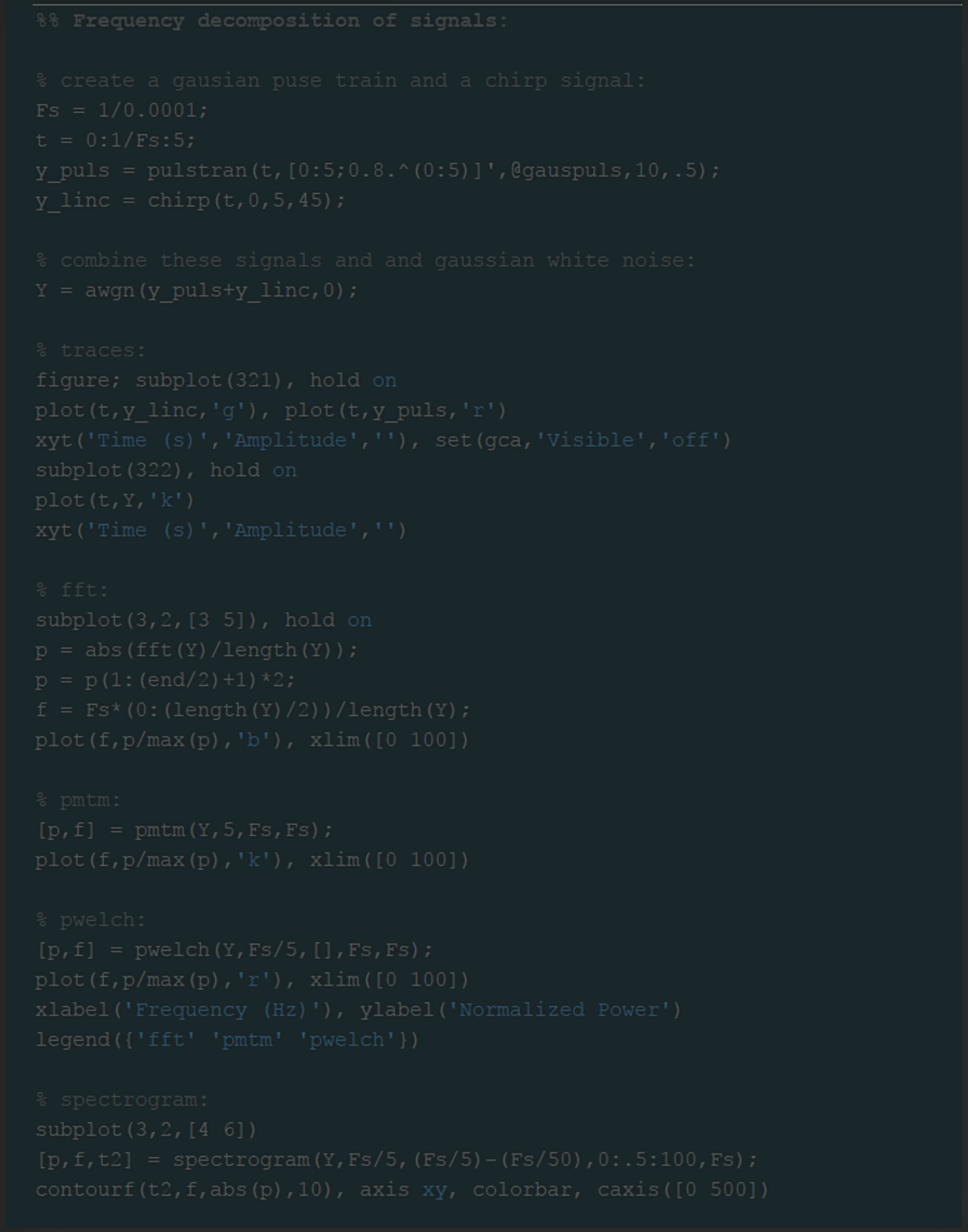
```
%% creating signals

% anonymous function:
% f(x) = a*cos((b*x)+c); a = 4; b = .2; c = pi/2;
f1 = @(x) 4*cos((.2*x)+(pi/2));
f2 = @(x,a,b,c) a*cos((b*x)+c);

% inputs to the function:
x = .1:.1:30;
a = 4;
b = .2;
c = pi/2;

% using the function:
y = f2(x,a,b,c);

% plotting the result:
figure, plot(x,y)
```



```
%% Comparing sin & cos:

% create traces:
x = .1:.1:30;
y1 = sin(x);
y2 = cos(x);

% plot traces:
figure, hold on, plot(x,y1), plot(x,y2)
legend({'sin(x)' 'cos(x)'})
```



```
%% orthogonality:

x = .1:.1:30;
y1 = sin(x);
y2 = sin(x/180*pi);

figure, hold on, plot(x,y1), plot(x,y2)
legend({'sin(x)' 'sin(x*2)'})
```



```
%% Frequency decomposition of signals:

% create a gausian puse train and a chirp signal:
Fs = 1/0.0001;
t = 0:1/Fs:5;
y_puls = pulstran(t,[0:5;0.8.^(0:5)]',@gauspuls,10,.5);
y_linc = chirp(t,0,5,45);

% combine these signals and and gaussian white noise:
Y = awgn(y_puls+y_linc,0);

% traces:
figure; subplot(321), hold on
plot(t,y_linc,'g'), plot(t,y_puls,'r')
xyt('Time (s)','Amplitude',''), set(gca,'Visible','off')
subplot(322), hold on
plot(t,Y,'k')
xyt('Time (s)','Amplitude','')

% fft:
subplot(3,2,[3 5]), hold on
p = abs(fft(Y)/length(Y));
p = p(1:(end/2)+1)*2;
f = Fs*(0:(length(Y)/2))/length(Y);
plot(f,p/max(p),'b'), xlim([0 100])

% pmtm:
[p,f] = pmtm(Y,5,Fs,Fs);
plot(f,p/max(p),'k'), xlim([0 100])

% pwelch:
[p,f] = pwelch(Y,Fs/5,[],Fs,Fs);
plot(f,p/max(p),'r'), xlim([0 100])
xlabel('Frequency (Hz)'), ylabel('Normalized Power')
legend({'fft' 'pmtm' 'pwelch'})

% spectrogram:
subplot(3,2,[4 6])
[p,f,t2] = spectrogram(Y,Fs/5,(Fs/5)-(Fs/50),0:.5:100,Fs);
contourf(t2,f,abs(p),10), axis xy, colorbar, caxis([0 500])
```

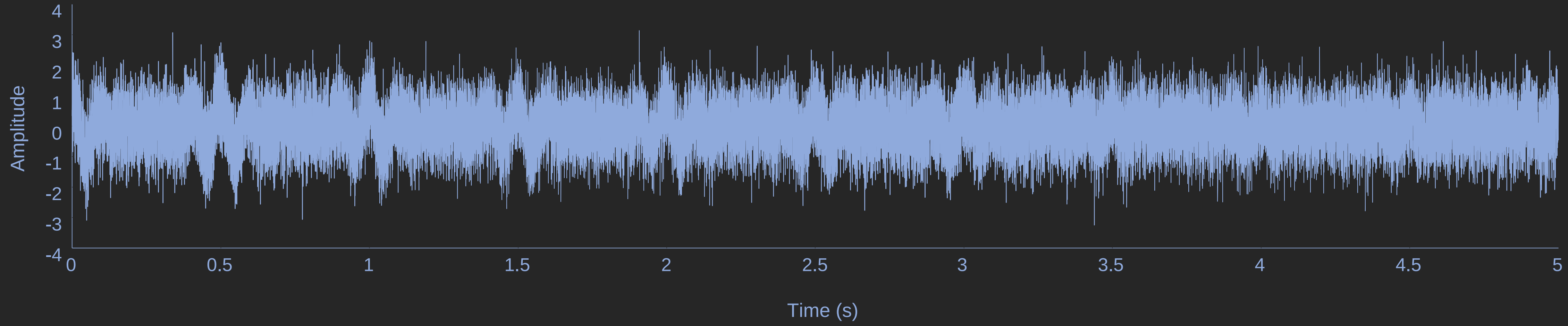
(Fast) Fourier Transform in Words

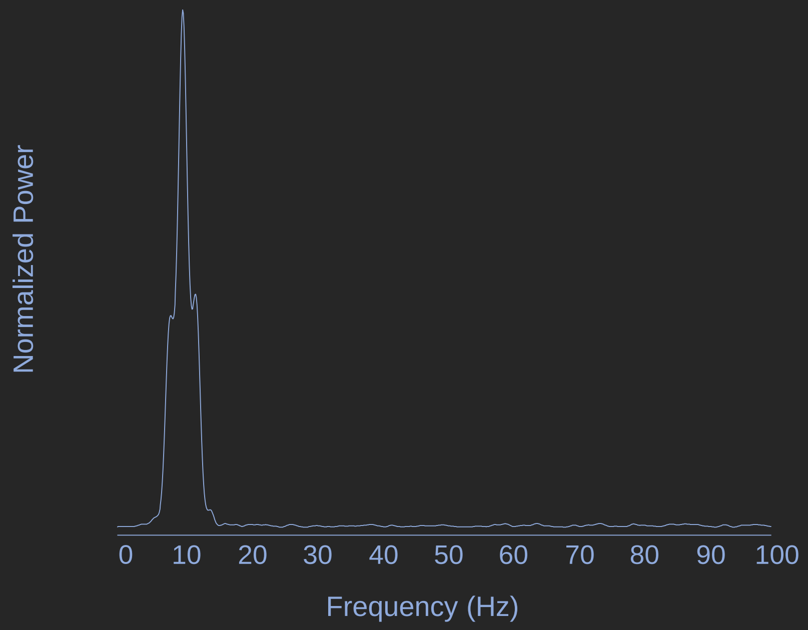
- You’ve got a signal consisting of N sample points (equidistant).
- You want to know which frequencies contribute to the signal, and how much.

- With N samples, you can estimate at most N independent parameters.
- You cannot estimate frequencies above half of the sampling frequency (the Nyquist limit).

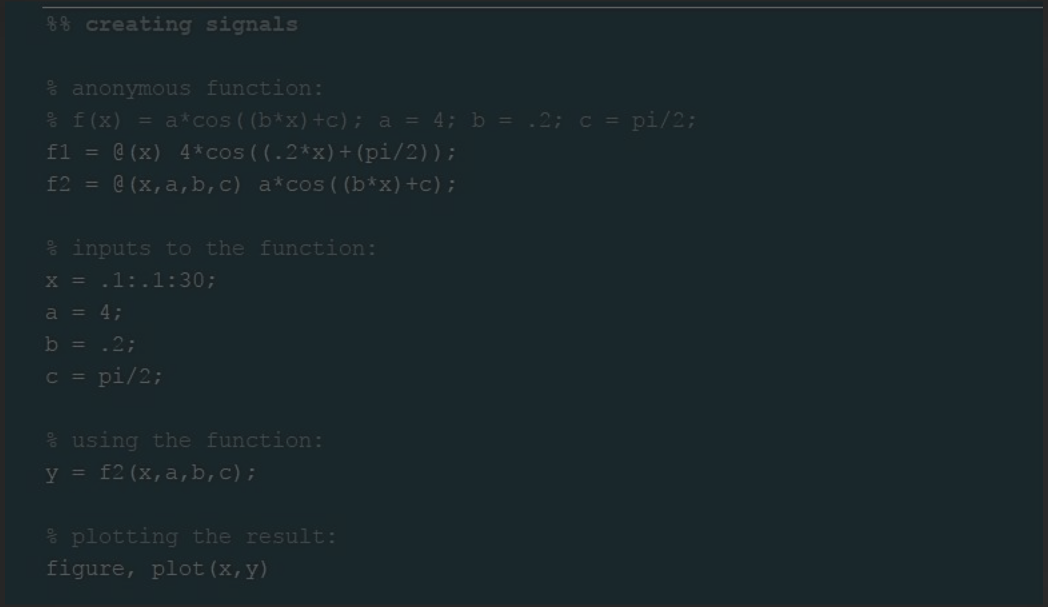
- To do this you will want to describe your signal as a linear combination of sines and cosines
- For a given frequency, sine and cosine are orthogonal, i.e. 2 basis functions per frequency.

70 Hz	70*2 = 140 Hz
	70*5 = 350 Hz

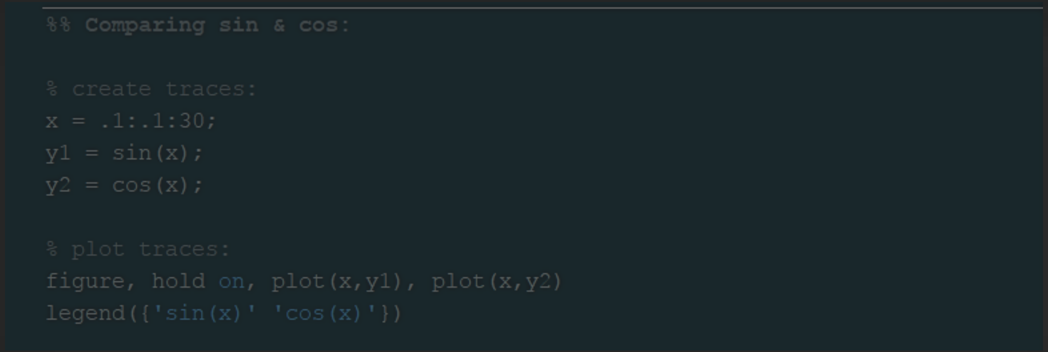

Amplitude
4
3
2
1
0
-1
-2
-3
-4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Time (s)


Normalized Power
0
10
20
30
40
50
60
70
80
90
100
Frequency (Hz)

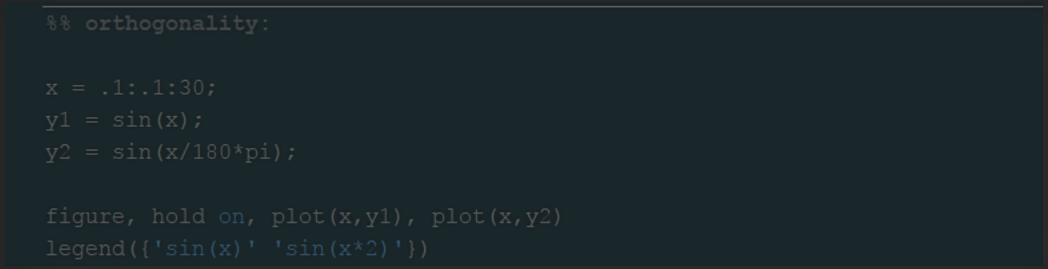
Matlab



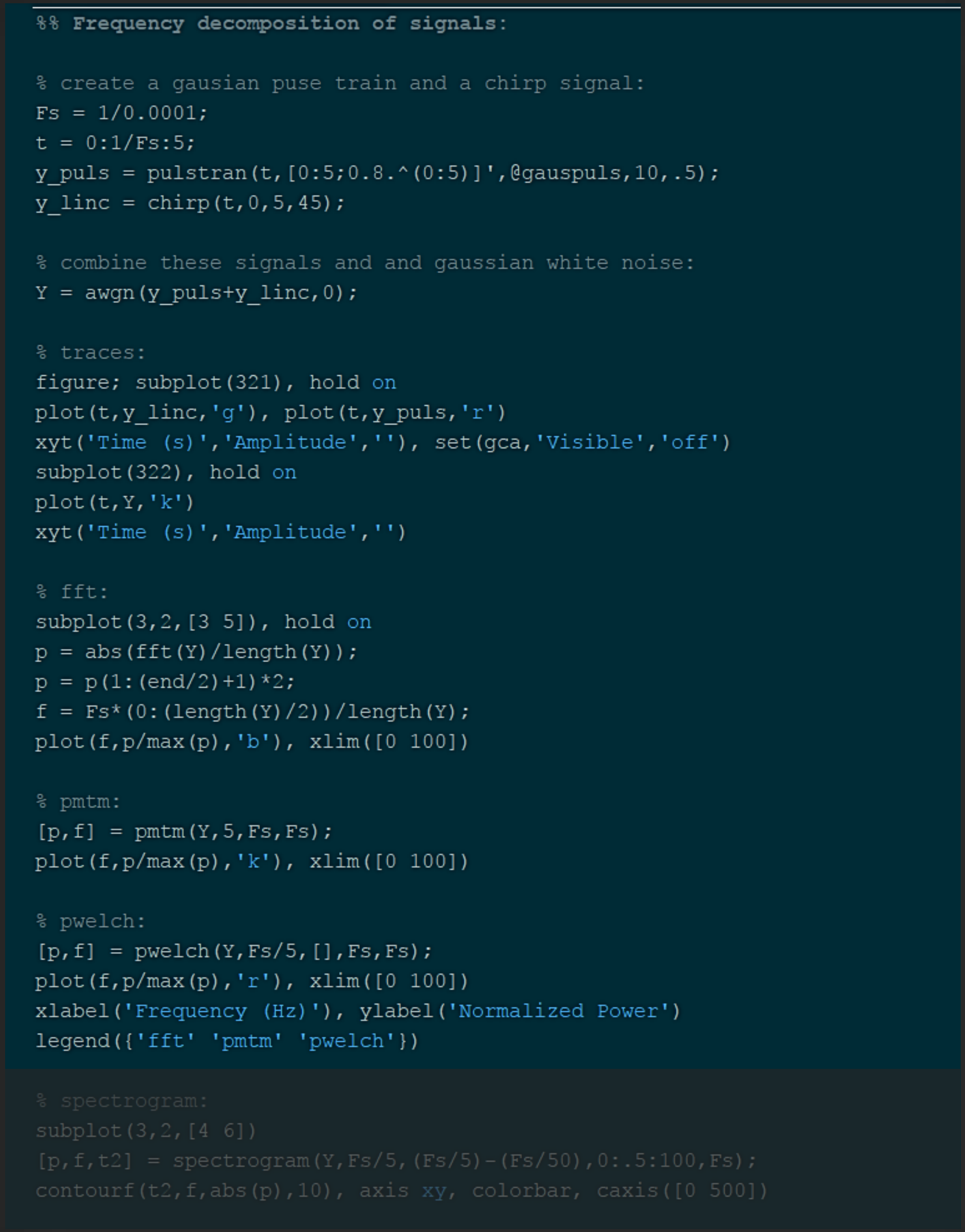
```
%% creating signals

% anonymous function:
% f(x) = a*cos((b*x)+c); a = 4; b = .2; c = pi/2;
f1 = @(x) 4*cos((.2*x)+(pi/2));
f2 = @(x,a,b,c) a*cos((b*x)+c);

% inputs to the function:
x = .1:.1:30;
a = 4;
b = .2;
c = pi/2;

% using the function:
y = f2(x,a,b,c);

% plotting the result:
figure, plot(x,y)
```



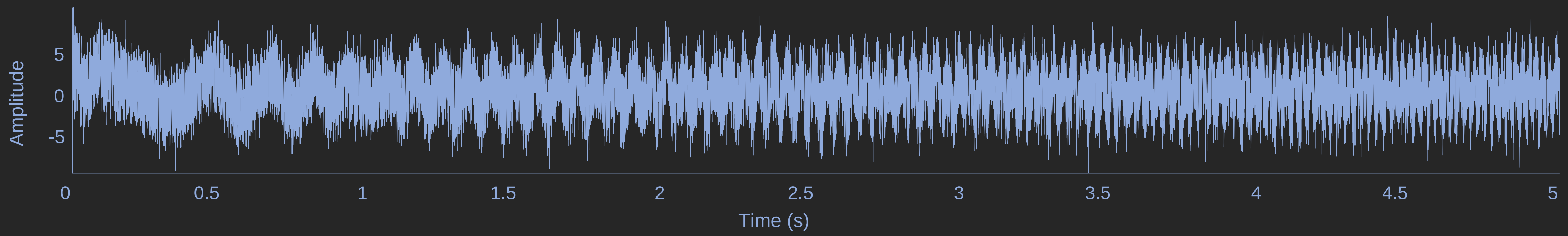
```
%% Comparing sin & cos:

% create traces:
x = .1:.1:30;
y1 = sin(x);
y2 = cos(x);

% plot traces:
figure, hold on, plot(x,y1), plot(x,y2)
legend({'sin(x)' 'cos(x)'})
```

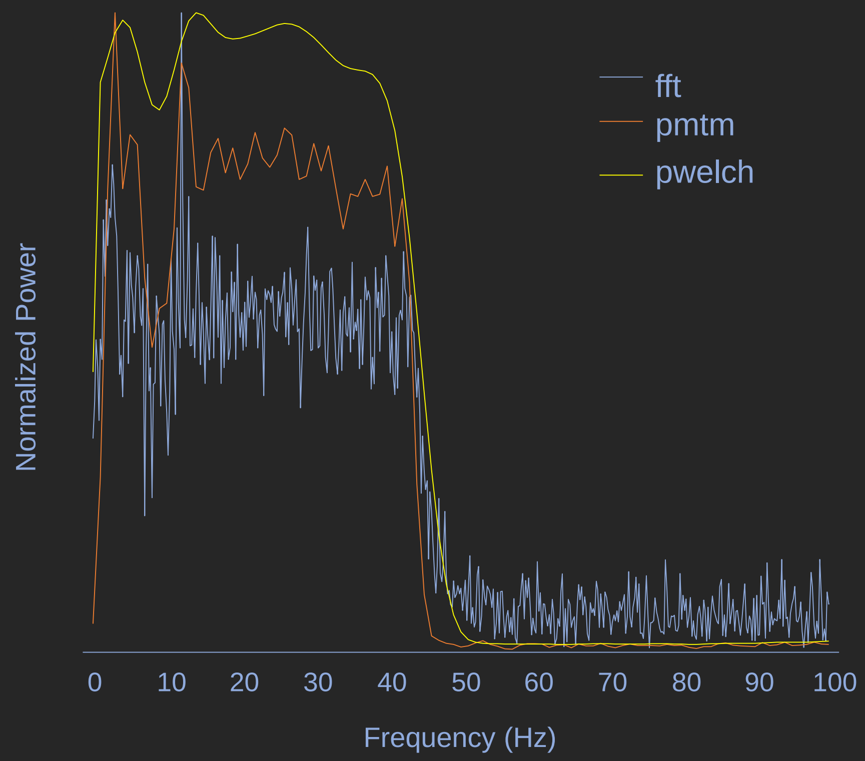


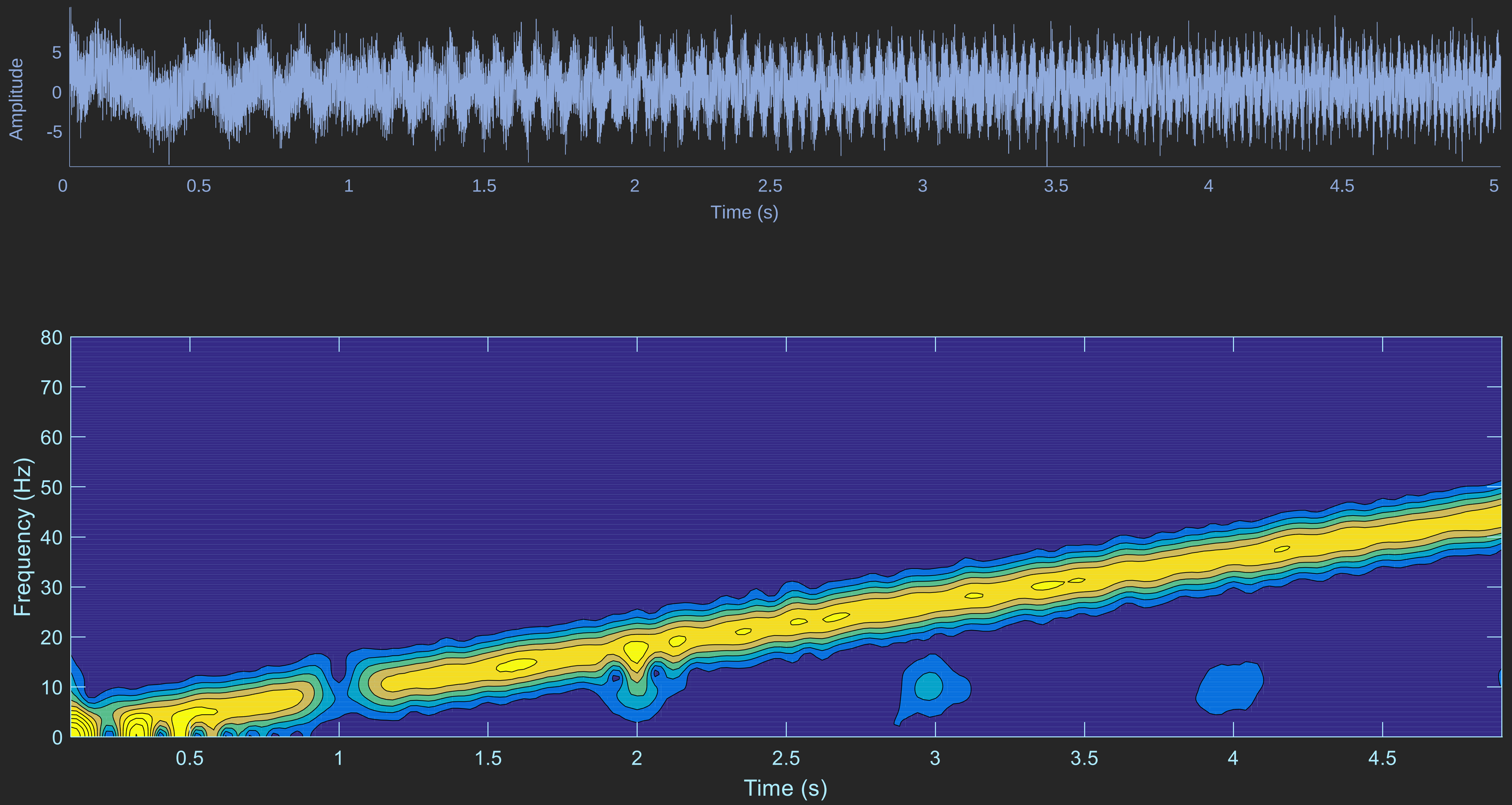
```
%% orthogonality:

x = .1:.1:30;
y1 = sin(x);
y2 = sin(x/180*pi);

figure, hold on, plot(x,y1), plot(x,y2)
legend({'sin(x)' 'sin(x*2)'})
```

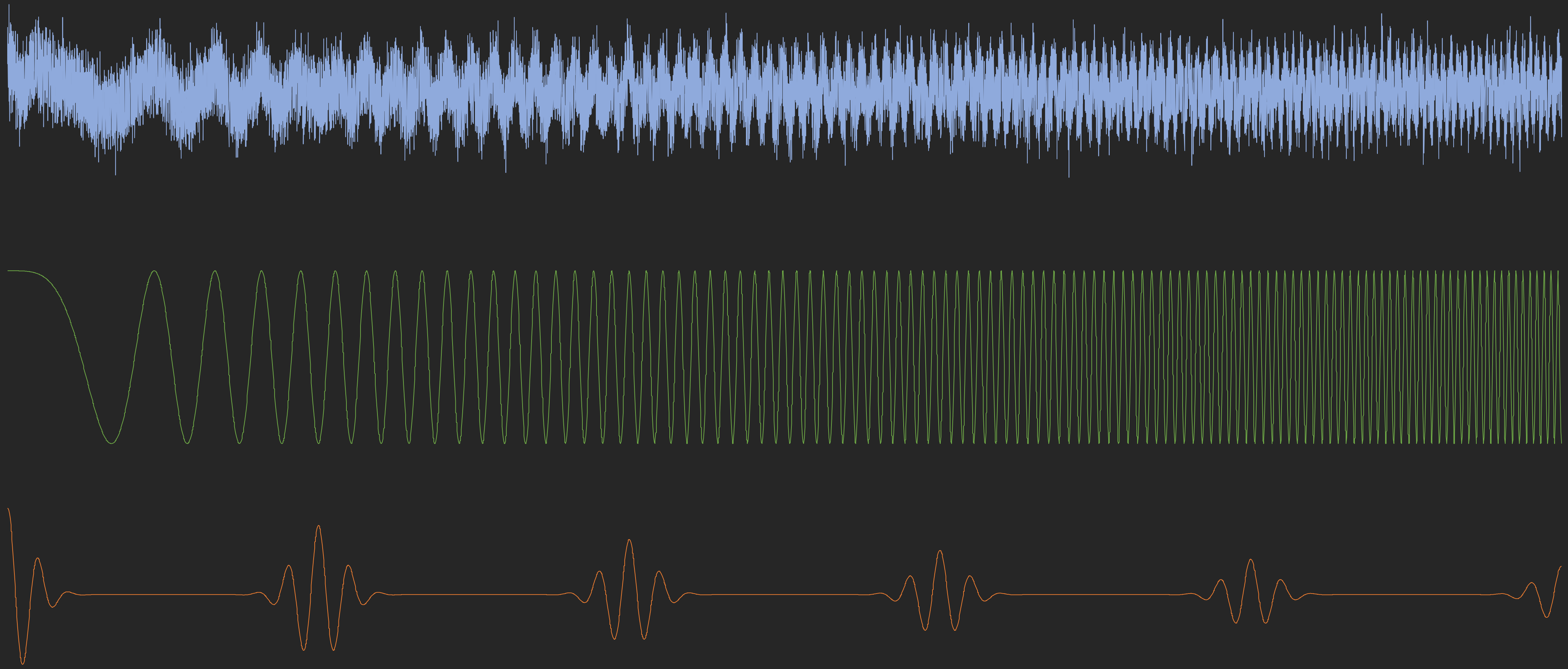


```
%% Frequency decomposition of signals:

% create a gausian puse train and a chirp signal:
Fs = 1/0.0001;
t = 0:1/Fs:5;
y_puls = pulstran(t,[0:5;0.8.^(0:5)]',@gauspuls,10,.5);
y_linc = chirp(t,0,5,45);

% combine these signals and and gaussian white noise:
Y = awgn(y_puls+y_linc,0);

% traces:
figure; subplot(321), hold on
plot(t,y_linc,'g'), plot(t,y_puls,'r')
xyt('Time (s)','Amplitude',''), set(gca,'Visible','off')
subplot(322), hold on
plot(t,Y,'k')
xyt('Time (s)','Amplitude','')

% fft:
subplot(3,2,[3 5]), hold on
p = abs(fft(Y)/length(Y));
p = p(1:(end/2)+1)*2;
f = Fs*(0:(length(Y)/2))/length(Y);
plot(f,p/max(p),'b'), xlim([0 100])

% pmtm:
[p,f] = pmtm(Y,5,Fs,Fs);
plot(f,p/max(p),'k'), xlim([0 100])

% pwelch:
[p,f] = pwelch(Y,Fs/5,[],Fs,Fs);
plot(f,p/max(p),'r'), xlim([0 100])
xlabel('Frequency (Hz)'), ylabel('Normalized Power')
legend({'fft' 'pmtm' 'pwelch'})

% spectrogram:
subplot(3,2,[4 6])
[p,f,t2] = spectrogram(Y,Fs/5,(Fs/5)-(Fs/50),0:.5:100,Fs);
contourf(t2,f,abs(p),10), axis xy, colorbar, caxis([0 500])
```

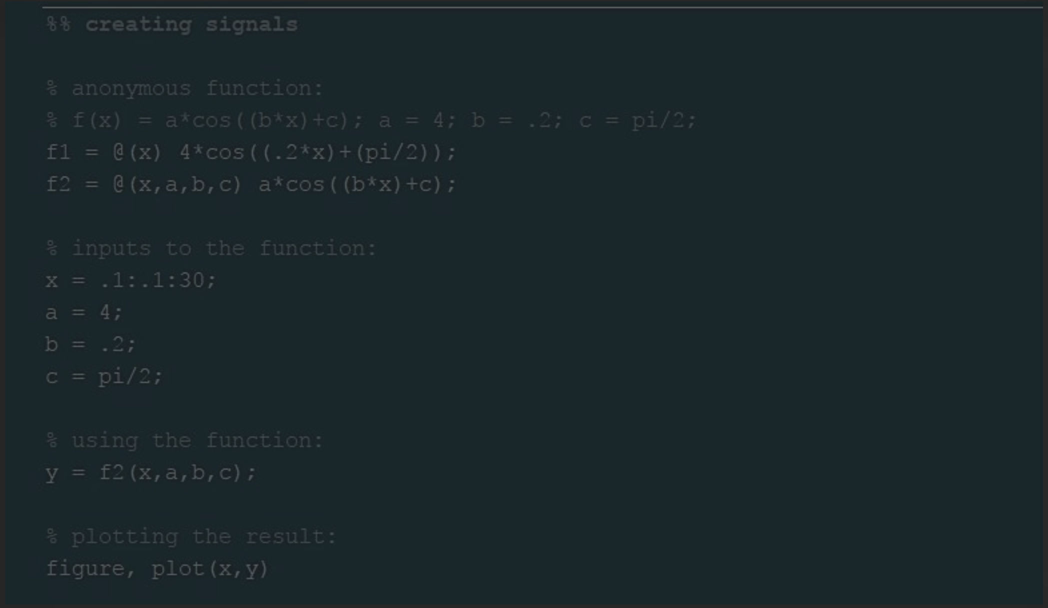

Amplitude
5
0
-5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Time (s)


Normalized Power
fft
pmtm
pwelch
0
10
20
30
40
50
60
70
80
90
100
Frequency (Hz)

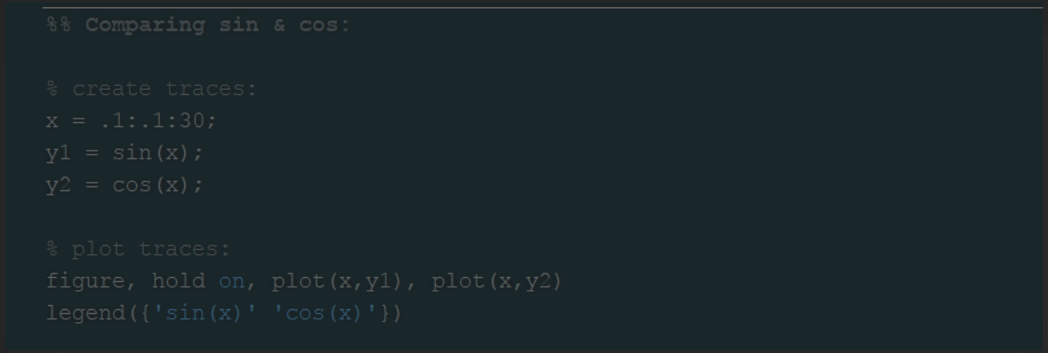

Amplitude
5
0
-5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Time (s)
Frequency (Hz)
80
70
60
50
40
30
20
10
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Time (s)



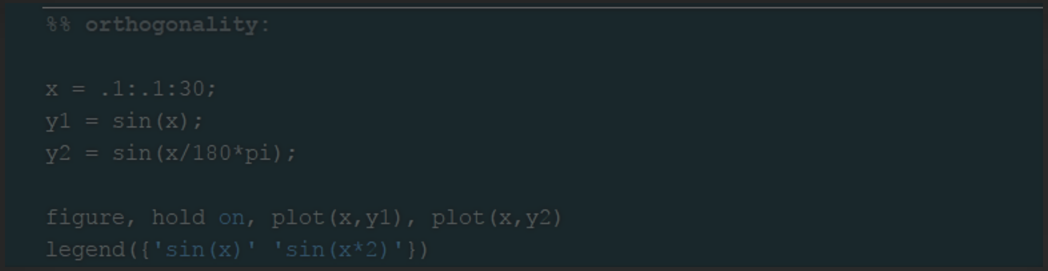
Matlab



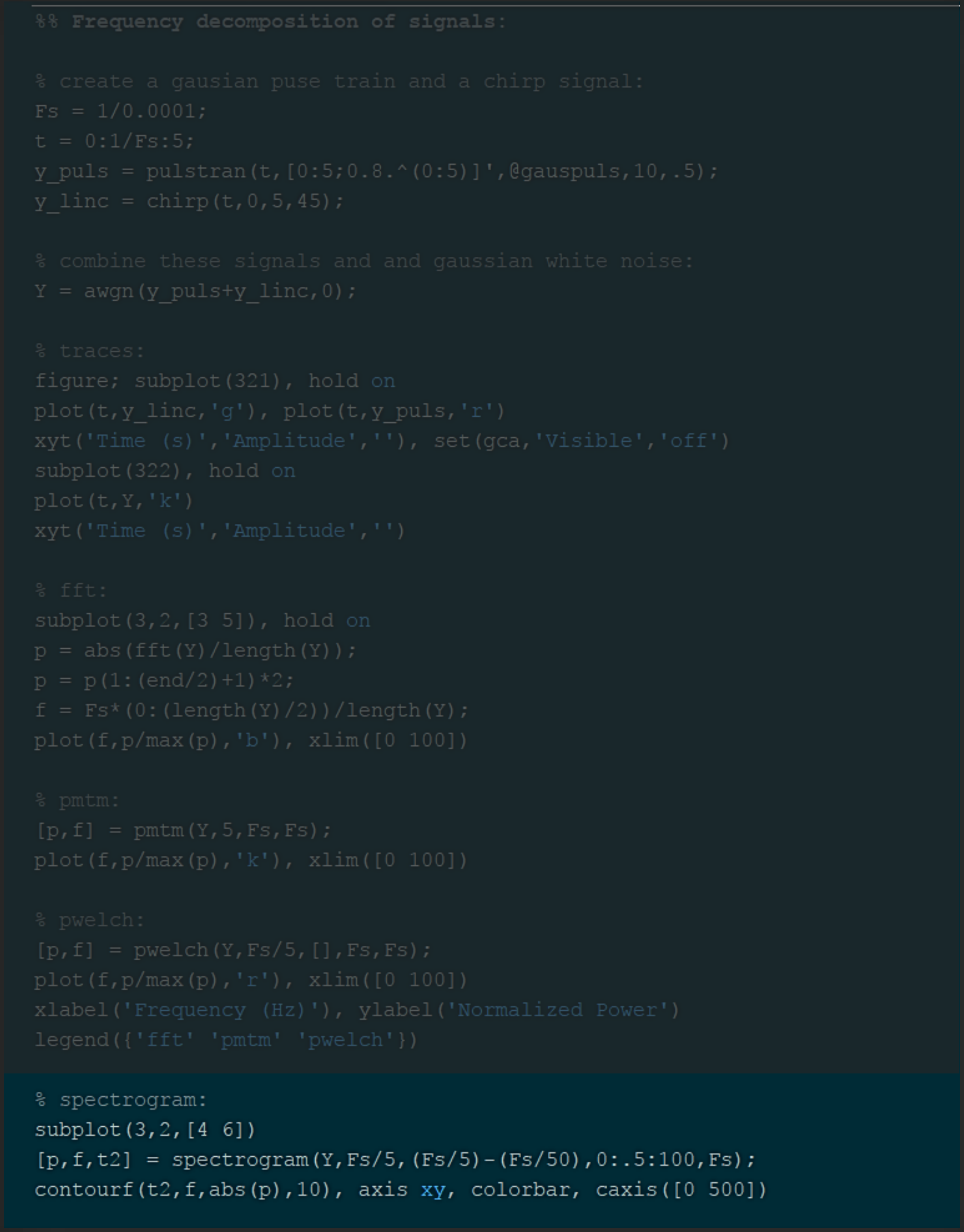
```
%% creating signals

% anonymous function:
% f(x) = a*cos((b*x)+c); a = 4; b = .2; c = pi/2;
f1 = @(x) 4*cos((.2*x)+(pi/2));
f2 = @(x,a,b,c) a*cos((b*x)+c);

% inputs to the function:
x = .1:.1:30;
a = 4;
b = .2;
c = pi/2;

% using the function:
y = f2(x,a,b,c);

% plotting the result:
figure, plot(x,y)
```



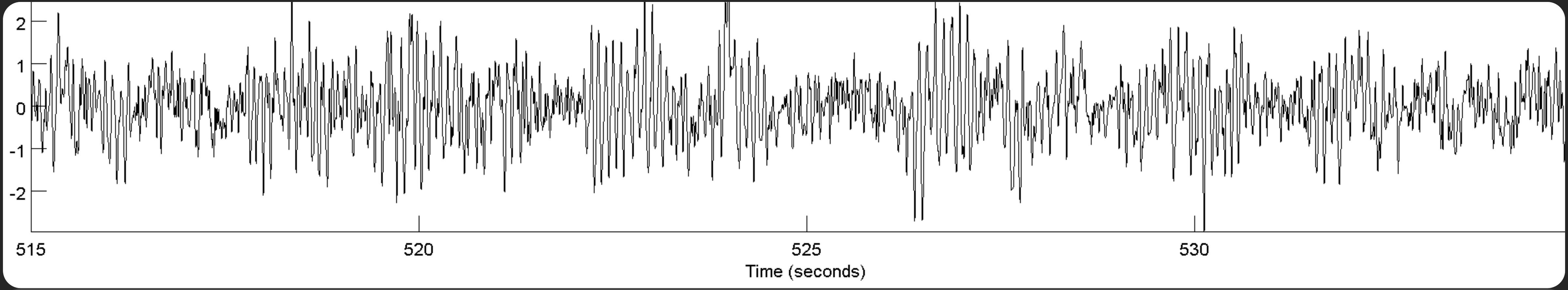
```
%% Comparing sin & cos:

% create traces:
x = .1:.1:30;
y1 = sin(x);
y2 = cos(x);

% plot traces:
figure, hold on, plot(x,y1), plot(x,y2)
legend({'sin(x)' 'cos(x)'})
```

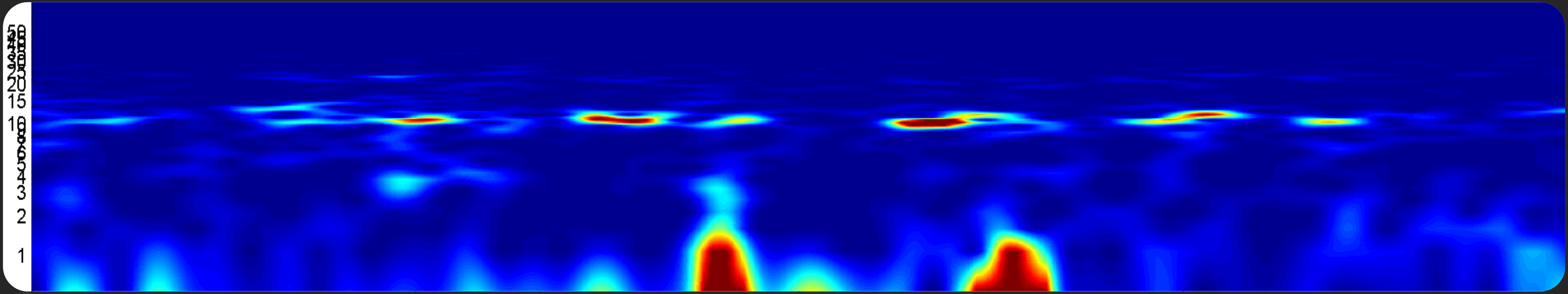


```
%% orthogonality:

x = .1:.1:30;
y1 = sin(x);
y2 = sin(x/180*pi);

figure, hold on, plot(x,y1), plot(x,y2)
legend({'sin(x)' 'sin(x*2)'})
```



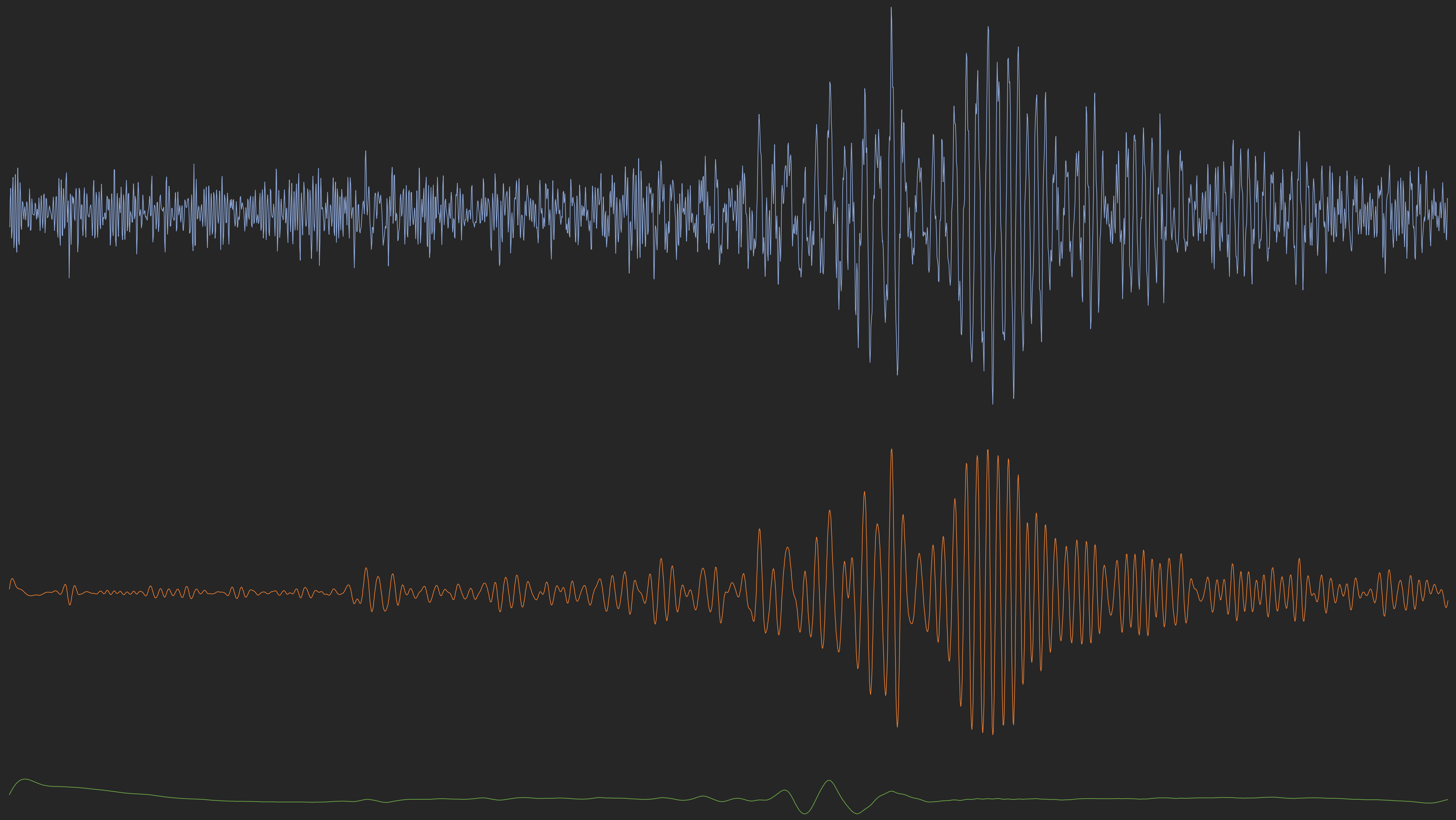
```
%% Frequency decomposition of signals:

% create a gausian puse train and a chirp signal:
Fs = 1/0.0001;
t = 0:1/Fs:5;
y_puls = pulstran(t,[0:5;0.8.^(0:5)]',@gauspuls,10,.5);
y_linc = chirp(t,0,5,45);

% combine these signals and and gaussian white noise:
Y = awgn(y_puls+y_linc,0);

% traces:
figure; subplot(321), hold on
plot(t,y_linc,'g'), plot(t,y_puls,'r')
xyt('Time (s)','Amplitude',''), set(gca,'Visible','off')
subplot(322), hold on
plot(t,Y,'k')
xyt('Time (s)','Amplitude','')

% fft:
subplot(3,2,[3 5]), hold on
p = abs(fft(Y)/length(Y));
p = p(1:(end/2)+1)*2;
f = Fs*(0:(length(Y)/2))/length(Y);
plot(f,p/max(p),'b'), xlim([0 100])

% pmtm:
[p,f] = pmtm(Y,5,Fs,Fs);
plot(f,p/max(p),'k'), xlim([0 100])

% pwelch:
[p,f] = pwelch(Y,Fs/5,[],Fs,Fs);
plot(f,p/max(p),'r'), xlim([0 100])
xlabel('Frequency (Hz)'), ylabel('Normalized Power')
legend({'fft' 'pmtm' 'pwelch'})

% spectrogram:
subplot(3,2,[4 6])
[p,f,t2] = spectrogram(Y,Fs/5,(Fs/5)-(Fs/50),0:.5:100,Fs);
contourf(t2,f,abs(p),10), axis xy, colorbar, caxis([0 500])
```


2
1
0
-1
-2
515
520
525
530
Time (seconds)

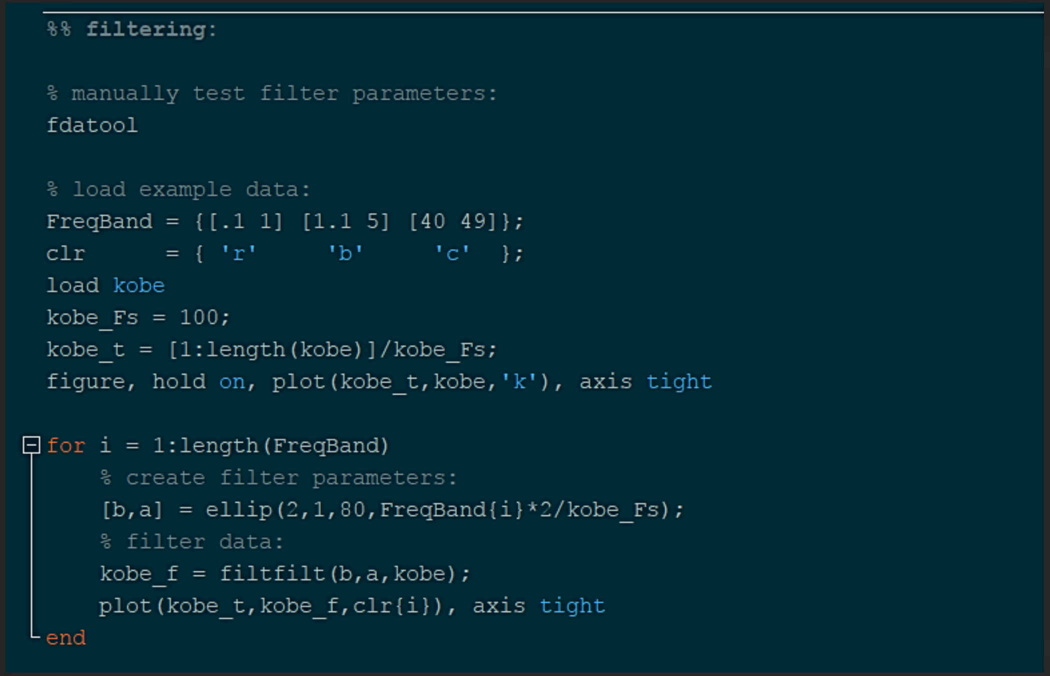

50
20
15
10
4
3
2
1

Filtering – the basic idea

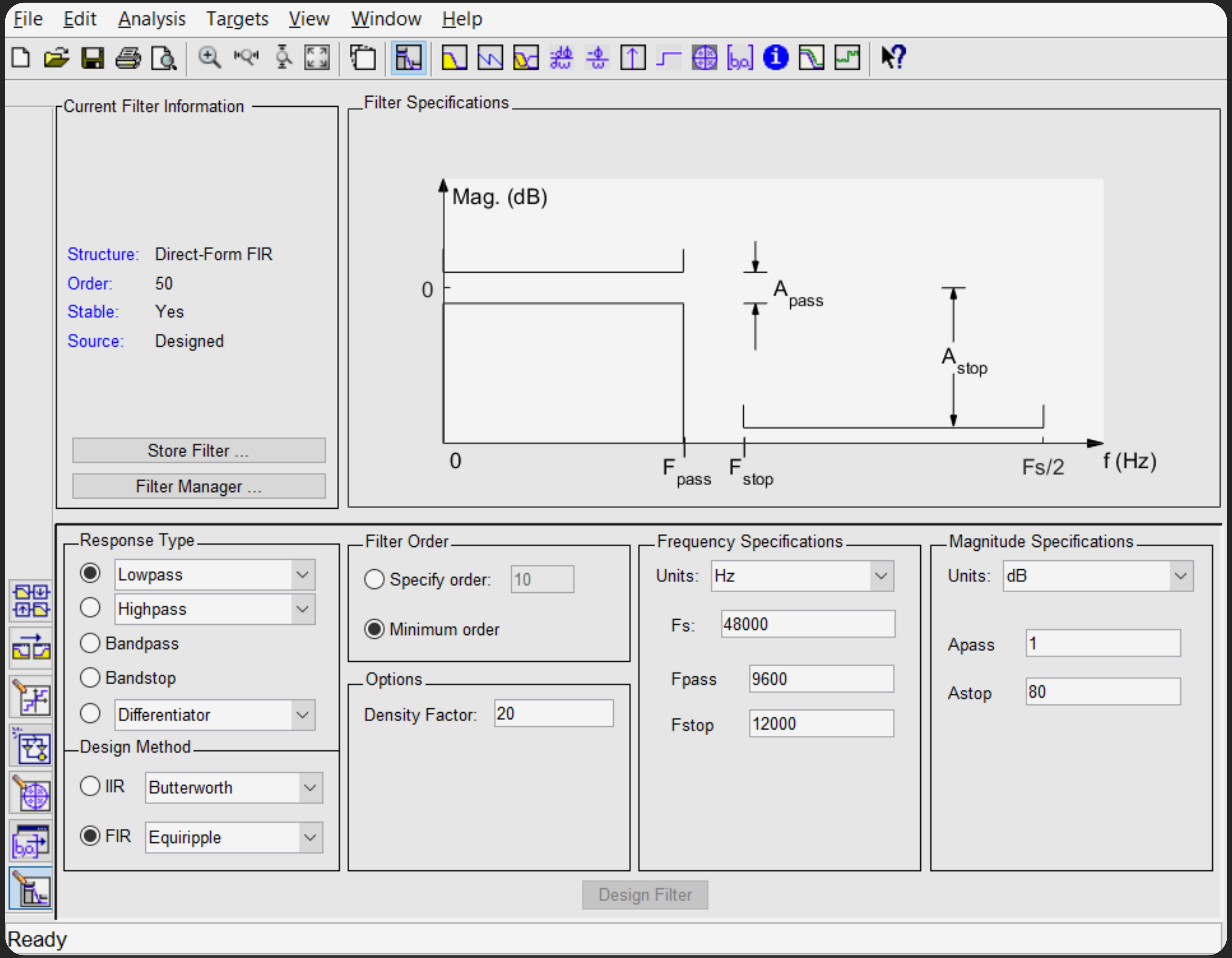
- Transform your data into the frequency domain
- Remove the frequency data you don’t want by lowering their weightings
- Reconstruct your data back into the time domain with the new weightings



Matlab



```
%% filtering:

% manually test filter parameters:
fdatool

% load example data:
FreqBand = {[.1 1] [1.1 5] [40 49]};
clr      = { 'r'     'b'     'c'  };
load kobe
kobe_Fs = 100;
kobe_t = [1:length(kobe)]/kobe_Fs;
figure, hold on, plot(kobe_t,kobe,'k'), axis tight

for i = 1:length(FreqBand)
    % create filter parameters:
    [b,a] = ellip(2,1,80,FreqBand{i}*2/kobe_Fs);
    % filter data:
    kobe_f = filtfilt(b,a,kobe);
    plot(kobe_t,kobe_f,clr{i}), axis tight
end
```



Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type:
● Lowpass
○ Highpass
○ Bandpass
○ Bandstop
○ Differentiator

Design Method:
○ IIR Butterworth
● FIR Equiripple

Filter Order:
○ Specify order: 10
● Minimum order

Options:
Density Factor: 20

Frequency Specifications:
Units: Hz
Fs: 48000
Fpass 9600
Fstop 12000

Magnitude Specifications:
Units: dB
Apass 1
Astop 80

Google™
Google Search
I'm Feeling Lucky

Example electrophysiological data is provided to try out the below. It consists of an LFP and simultaneous intracellular recordings of IPSPs from a pyramidal cell. Have some fun!

- Try plotting the LFP vs the time vector, labelling the x and y axis appropriately.
- You can play with the properties of this figure in the GUI (e.g. edit/axis properties), but I'd very much recommend playing with changing the color of the trace or the x axis limits etc via the command line.
- Plot the IPSPs and the LFP in two separate subplots on the same figure.
- Find the sampling frequency of the data using only the time vector.
- Band-pass filter (between 1 and 80 Hz) the LFP and display the result to check it has worked effectively by putting the unfiltered and filtered signals overlaid on the same plot.
- Find the frequency power spectrum of the LFP and plot the results, perhaps using different methods to see if there is a difference (lots of ways; hint: fft, pwelch, pmtm).
- Do a spectrogram to understand the dynamics of your spectral power.
- What frequencies of oscillations do you find, are they happening together, and are the pyramidal cells involved in both oscillations?
- How well correlated is the intracellular and field activity? (Hint: ask google)
- Write a matlab function that does this all for you