MASTER THE FOURIER TRANSFORM AND ITS APPLICATIONS

Course: https://www.udemy.com/course/fourier-transform-mxc/

Instructor: Mike X Cohen: http://sincxpress.com/

Notes and code completion (student): Ron Fredericks: http://BiophysicsLab.com/

Course-work: January 30, 2020 to February 9,2020 **Notes last updated: Thursday, October 22, 2020**

TABLE OF CONTENTS

Master the Fourier transform and its applications 1

Section 1: Introduction to the Fourier transform 2
2. Nontechnical description of Fourier transform 2
3. Examples of Fourier transform applications 3

Section 2: Foundations of the Fourier transform 16
9. Euler's formula e^ik 16
10. Sine waves and complex sine waves 21
11. Dot product 25
12. Complex dot product 27

Section 3: The discrete Fourier transform 31
14. How the discrete Fourier transform works 31
15. Converting indices to frequencies 33
16. Shortcut: converting indices to frequencies 35
17. Normalized time vector 36
18. Positive and negative frequencies 37
19. Accurate scaling of Fourier coefficients 38
20. Interpreting phase values 39
21. Averaging Fourier coefficients 40
22. The DC (zero frequency) component 42
23. Amplitude spectrum vs. power spectrum 43
24. A note about terminology of Fourier features 45

Section 4: The discrete inverse Fourier transform 46
26. How and why it works 46
27. Inverse Fourier transform for bandstop filtering 48

Section 5: The fast Fourier transform 49
29. How it works, speed tests 49
30. The fast inverse Fourier transform 51
31. The perfection of the Fourier transform 52
32. Using the fft on matrices 55

Section 6: Frequency resolution and zero padding 56
34. Sampling and frequency resolution 56
35 Time-domain zero padding 59

36. Frequency-domain zero padding 61
37. Sampling rate vs. signal length 63
38. Course tangent: self-accountability in online learning 64

Section 7: Aliasing, stationarity, and violations *64*
40. Aliasing 65
41. Signal stationarity and non-stationarities 69
42. Effects of non-stationarities on the power spectrum 71
43. Solution to understanding nonstationary time series 76
44. Windowing and Welch's method 80
45. Instantaneous frequency 81

Section 8: 2D Fourier transform *82*
46. How the 2D FFT works 82

Section 9: Applications of the Fourier transform *88*
47. Rhythmicity in walking (gait) 88
48. Rhythmicity in electrical brain waves 89
49. Time series convolution 89
50. Narrowband temporal filtering 91
51. 2D image filtering 93
52. Image narrowband filtering 94
53. Real data from trends.google.com! 95

Notes 98

FFT Examples *98*
Amplitude Spectrum 99
Power Spectrum 99
Power Spectrum in Decibels 99
Inverse FFT 99

Sinc function and Whittaker-Shannon interpolation formula *100*

MATLAB commands *100*

Utility code *101*

Code to take more courses *103*
From Section 2: Fourier_foundations.m 103

References *104*

SECTION 1: INTRODUCTION TO THE FOURIER TRANSFORM

2. NONTECHNICAL DESCRIPTION OF FOURIER TRANSFORM

Two major uses for the Fourier transform

1. Spectral analysis
    a. Some signals are better understood in the frequency domain.
    b. Spectral analyses can reveal insights that cannot be seen in the time domain
2. A means to an end in signal processing
    a. Use the convolution theorem to perform operations in the frequency domain:
        i. Filtering
        ii. Autocorrelation
        iii. Etc
    b. Frequency-domain computations are often easier and faster than equivalent time-domain computations

Power spectrum

Amplitude spectrum

Mechanisms of the Fourier transform:

Use sine waves to model a signal

For a given sine wave perhaps with low and high frequency features along with noise, line it up with the signal, measure the sine wave's "similarity" to the signal. Repeat for many different sine waves. The result will be a amplitude or power spectrum.

3. EXAMPLES OF FOURIER TRANSFORM APPLICATIONS

MATLAB

Code: fourier_intro.m

Image: Lenna.png

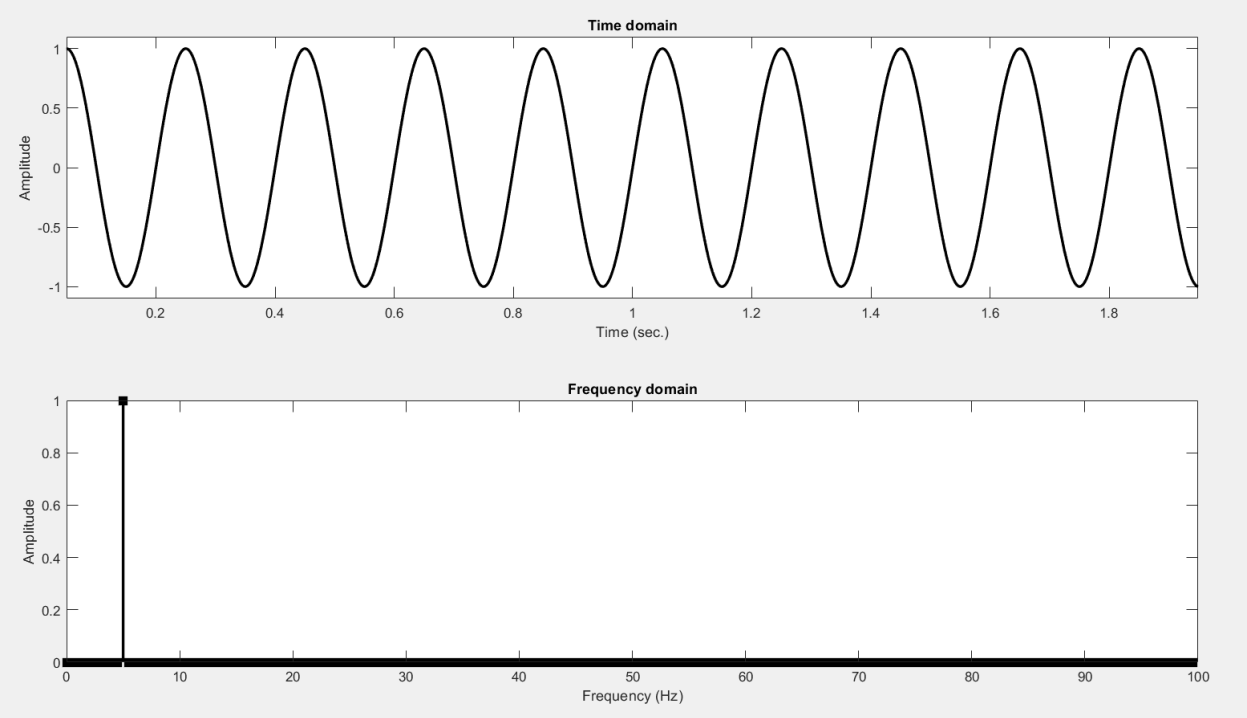
1-D EXAMPLES

Overall code from lines 39 to 100

```
srate = 1000;
time  = 0:1/srate:2-1/srate;
n     = length(time);
hz    = linspace(0,srate,n);

%%% pure sine wave
signal = sin(2*pi*5*time);

% compute amplitude spectrum
ampl = 2*abs(fft(signal)/n);

% plot in time domain
```

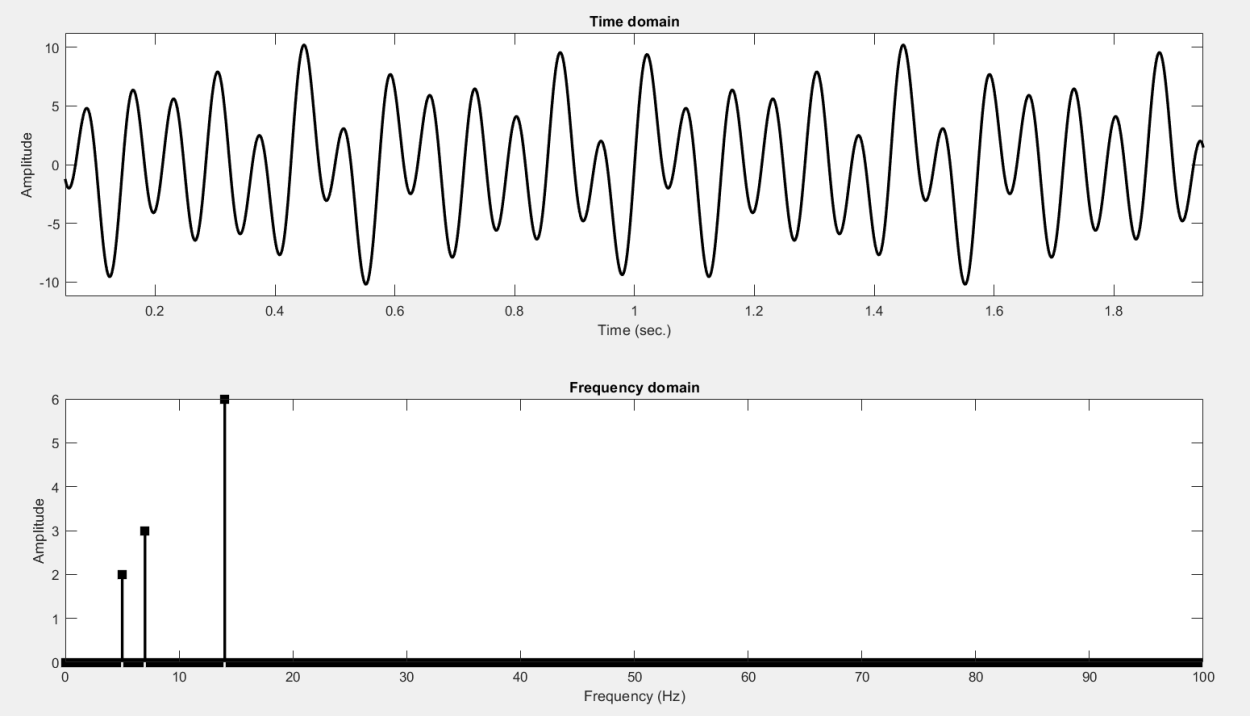
```
figure(1), clf
subplot(211)
plot(time,signal,'k','linew',2)
xlabel('Time (sec.)'), ylabel('Amplitude')
title('Time domain')
set(gca,'xlim',[time(1)+.05 time(end)-.05],'ylim',[min(signal)*1.1 max(signal)*1.1])

% plot in frequency domain
subplot(212)
stem(hz,ampl,'k-s','linew',2,'markerfacecolor','k')
xlabel('Frequency (Hz)'), ylabel('Amplitude')
title('Frequency domain')
set(gca,'xlim',[0 100])
```

PURE SINE WAVE



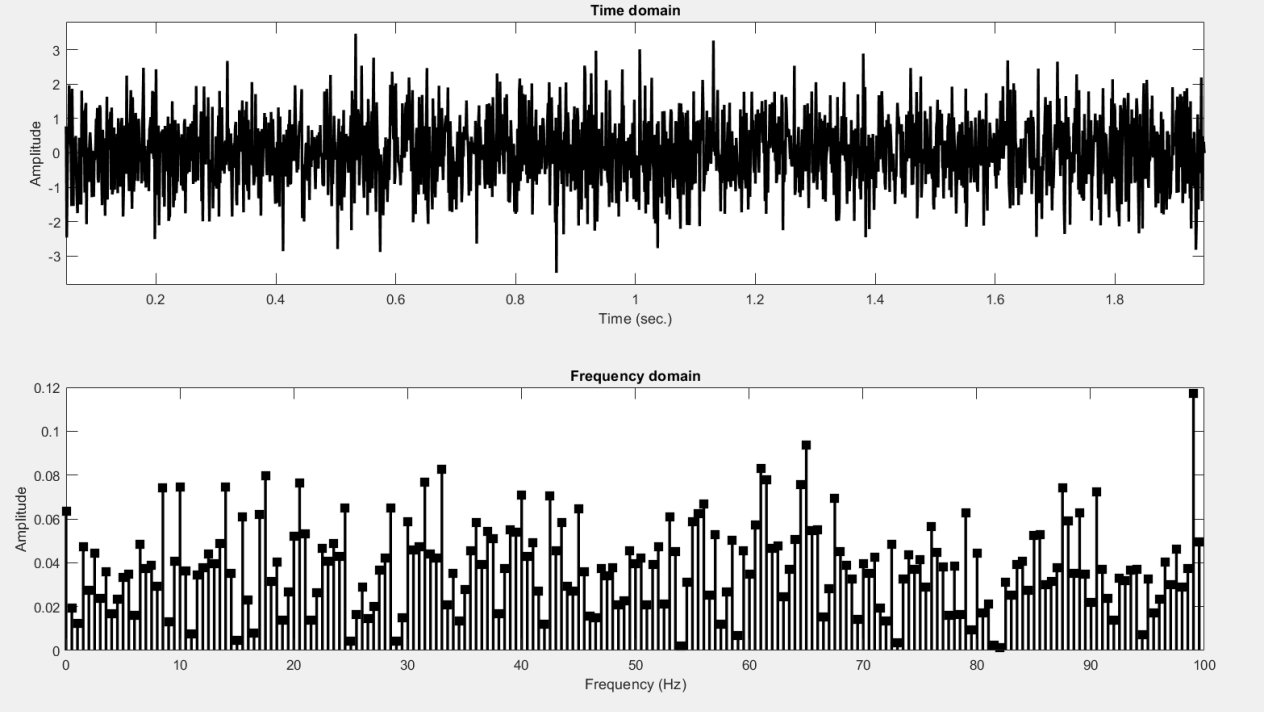

```
%%% pure sine wave
signal = sin(2*pi*5*time);
```

MULTISPECTRAL WAVE



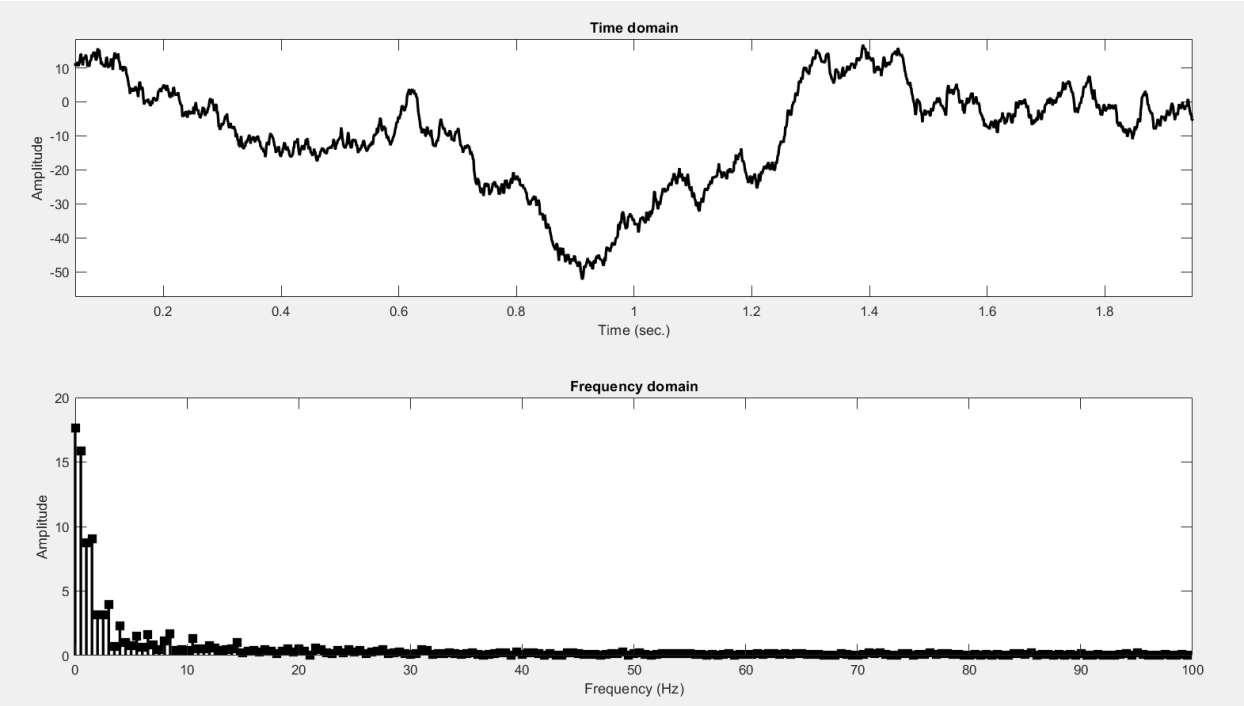

```
%%% multispectral wave
signal = 2*sin(2*pi*5*time) + 3*sin(2*pi*7*time) + 6*sin(2*pi*14*time);
```

WHITE NOISE



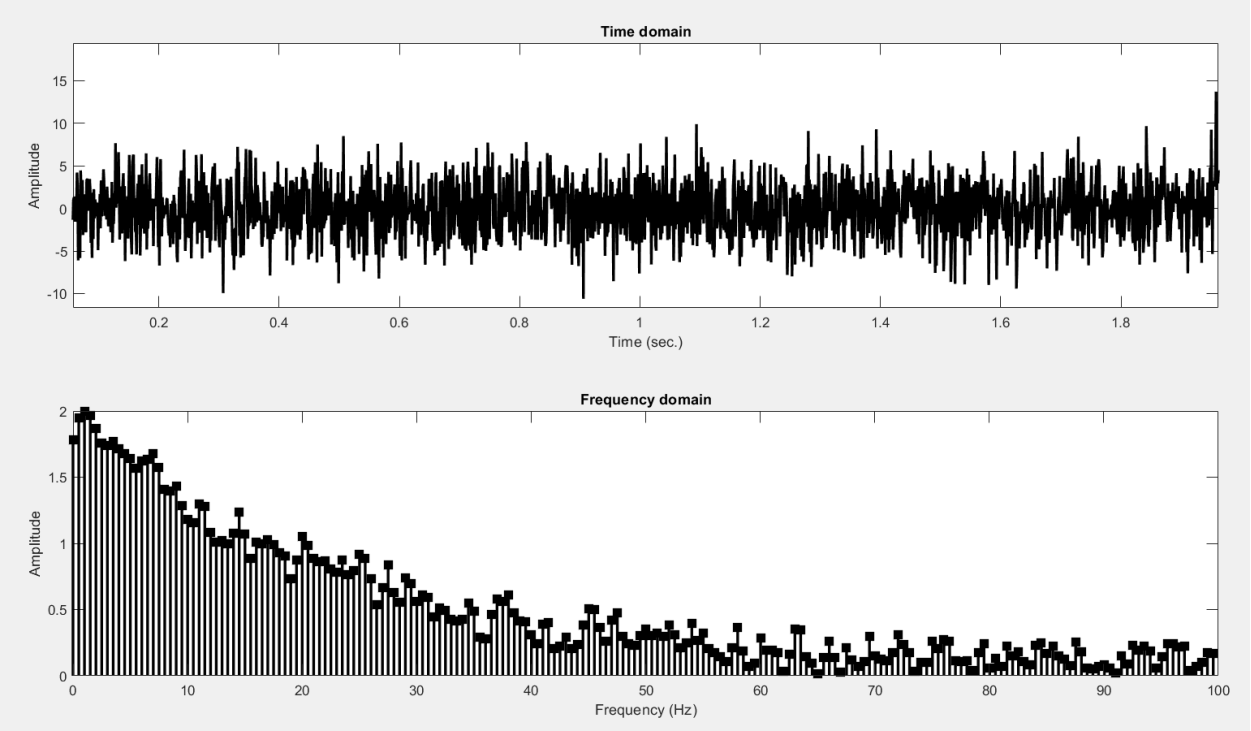
```
%%% white noise
signal = randn(size(time));
```

BROWNIAN NOISE



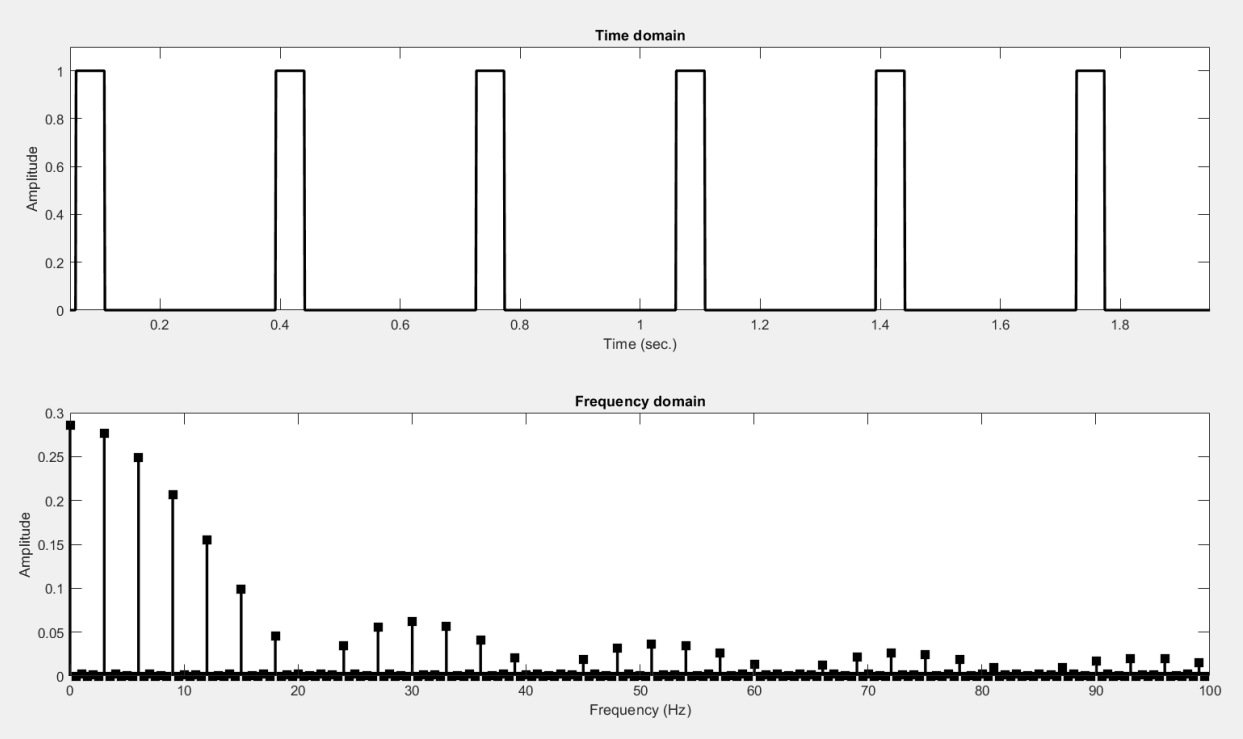

```
%%% Brownian noise (aka random walk or pink noise)
signal = cumsum(randn(size(time)));
```

1/F NOISE (PINK NOISE)



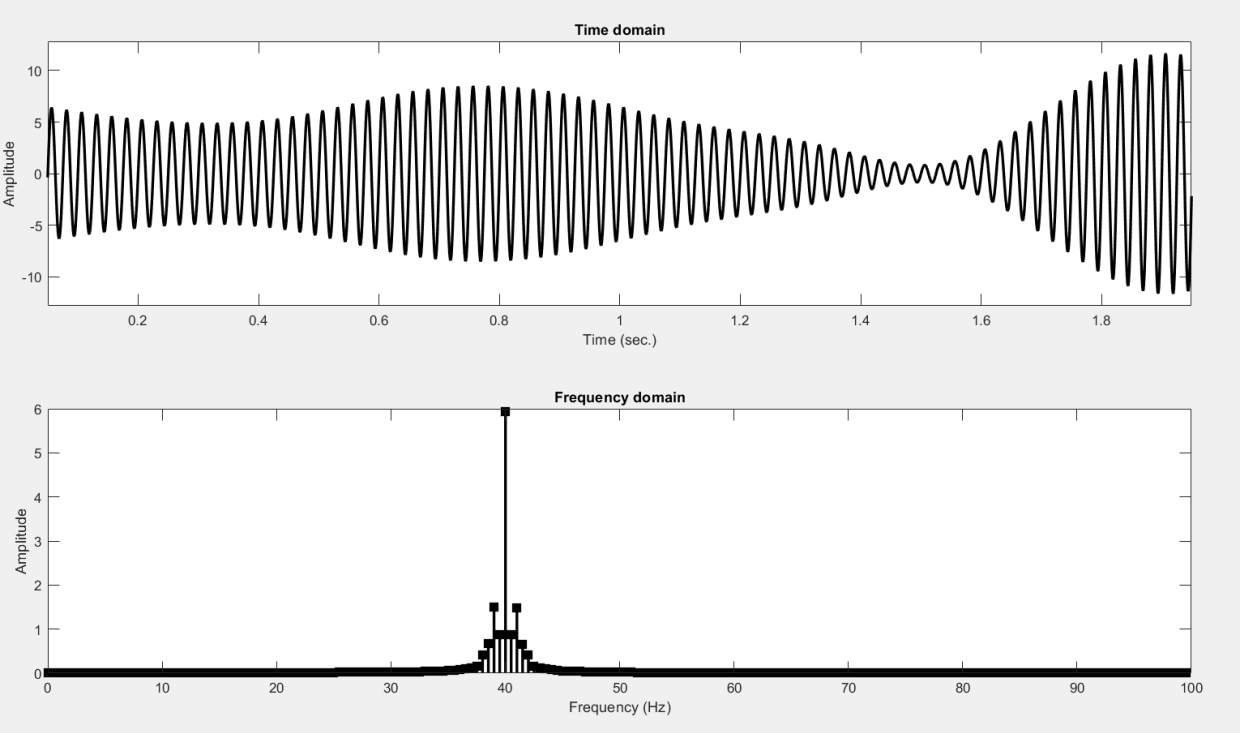

```
%%% 1/f noise a.k.a. Pink noise
ps   = exp(1i*2*pi*rand(1,n/2)) .* .1+exp(-(1:n/2)/50);
ps   = [ps ps(:,end:-1:1)];
signal = real(ifft(ps)) * n;
```

SQUARE WAVE



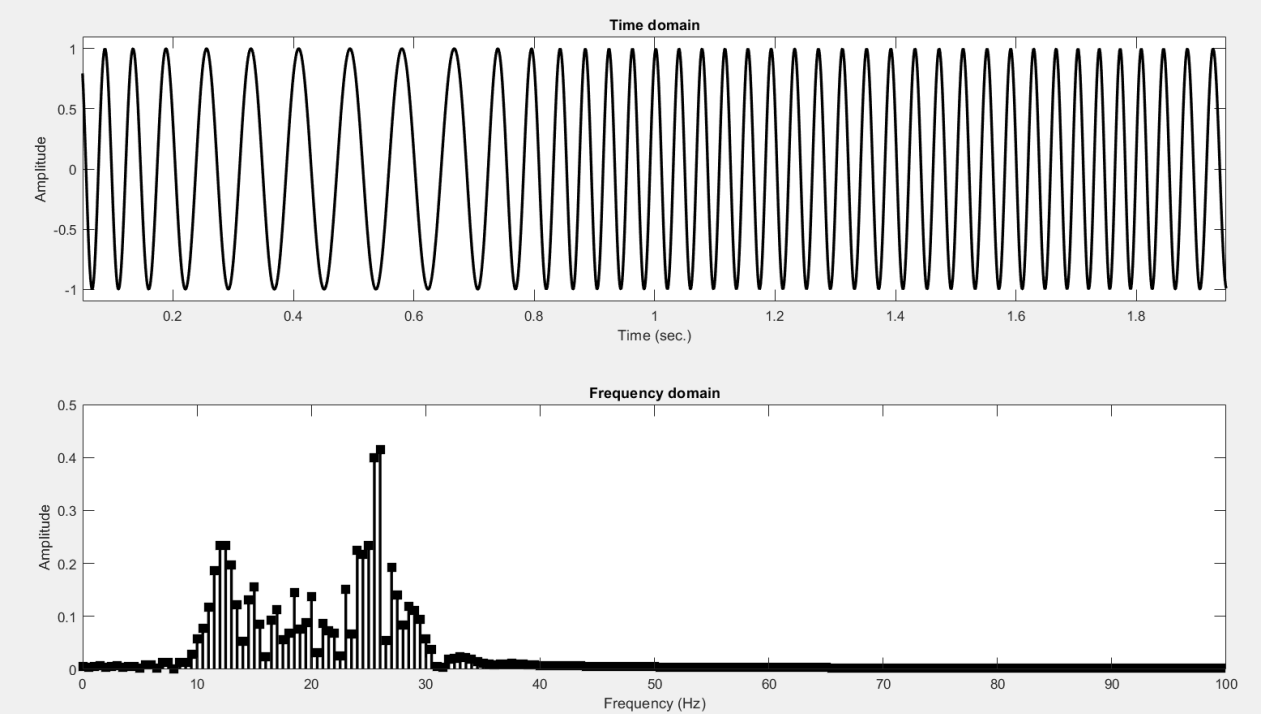

```
%%% square wave
signal = zeros(size(time));
signal(sin(2*pi*3*time)>.9) = 1;
```

AMPLITUDE MODULATION



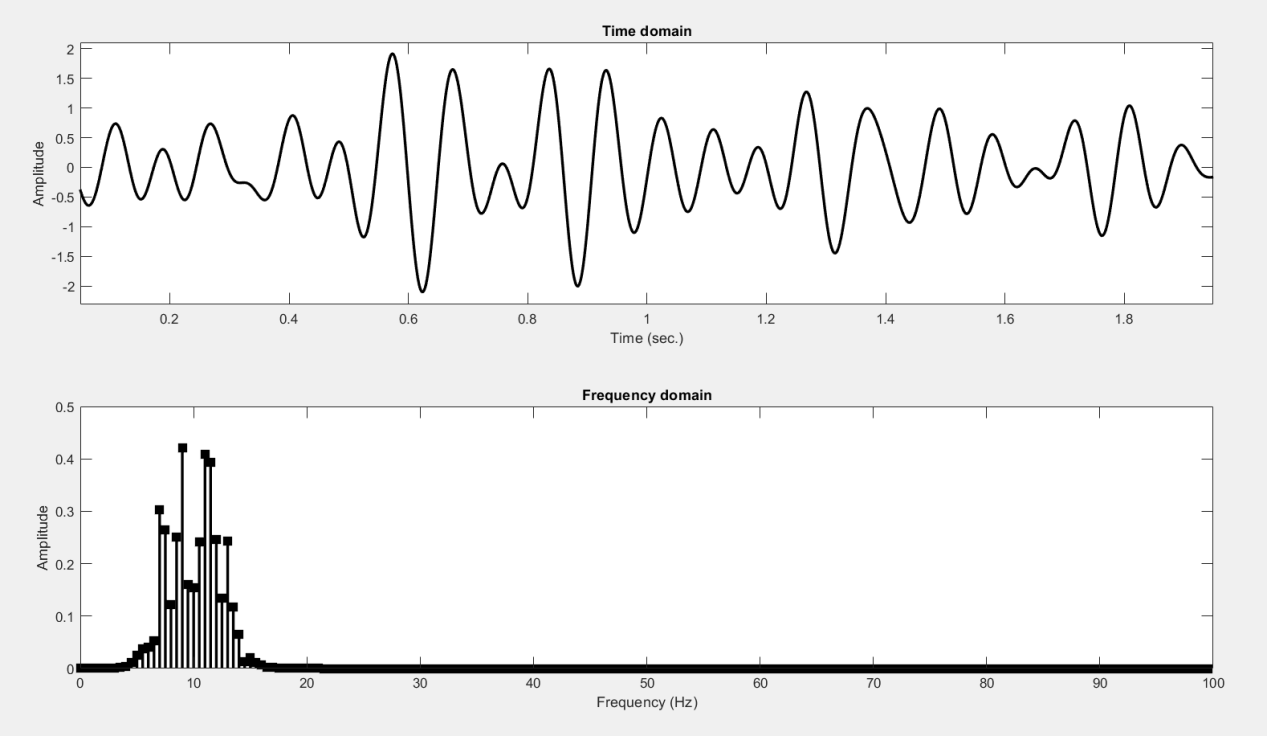

```
%%% AM (amplitude modulation)
signal = 10*interp1(rand(1,10),linspace(1,10,n),'spline') .* sin(2*pi*40*time);
```

FREQUENCY MODULATION



```
%%% FM (frequency modulation)
freqmod = 20*interp1(rand(1,10),linspace(1,10,n));
signal = sin( 2*pi * (10*time + cumsum(freqmod)/srate) );
```

FILTERED NOISE




```
%%% filtered noise
signal = randn(size(time));
s  = 5*(2*pi-1)/(4*pi);      % normalized width
fx = exp(-.5*((hz-10)/s).^2); % gaussian
fx = fx./max(fx);            % gain-normalize
signal = 20*real( ifft( fft(signal).*fx) );
```

2-D EXAMPLES

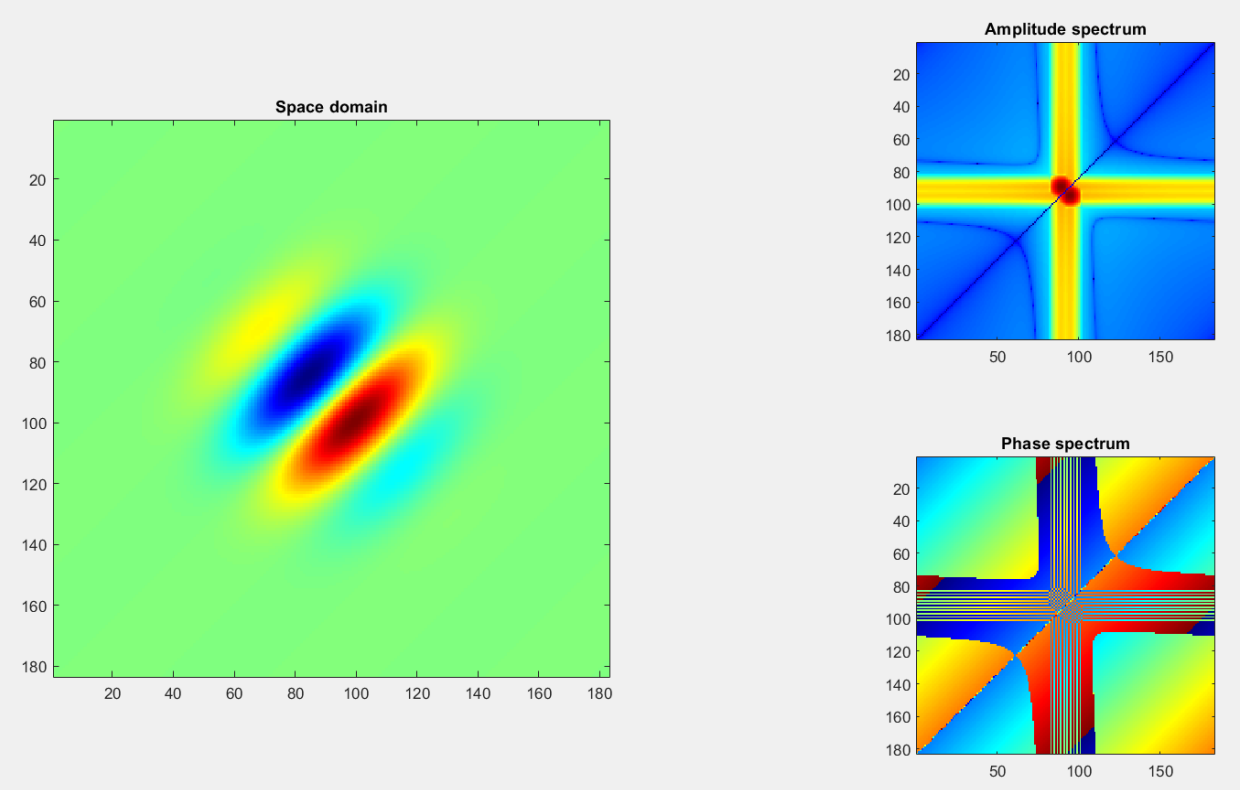
Overall code from lines 102 to 180

```
lims = [-91 91];

%%% portrait
lenna = imread('Lenna.png');
img   = double(mean(lenna,3));

% power and phase spectra
imgX  = fftshift(fft2(img));
powr2 = log(abs(imgX));
```

```
phas2 = angle(imgX);


figure(2), clf
subplot(121)
imagesc(img), axis square
title('Space domain')

subplot(222)
imagesc(powr2), axis square
nhalf = round(sum(abs(lims))/2);
title('Amplitude spectrum')

subplot(224)
imagesc(phas2), axis square
nhalf = round(sum(abs(lims))/2);
title('Phase spectrum')

colormap gray
```

GABOR FILTER

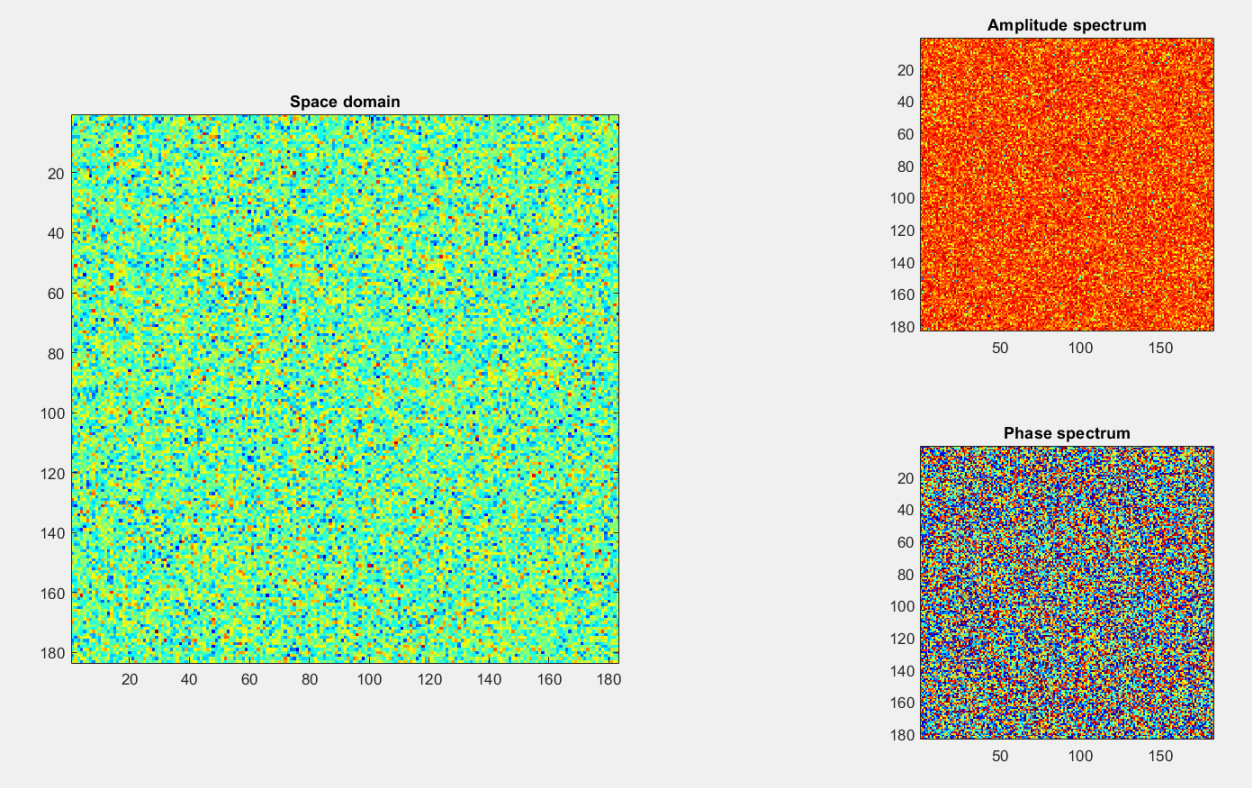
https://en.wikipedia.org/wiki/Gabor_filter




```
%%% gabor patch
```

```
width = 20;   % width of gaussian
sphs  = pi/4; % sine phase

lims = [-91 91];
[x,y] = ndgrid(lims(1):lims(2),lims(1):lims(2));
xp = x*cos(sphs) + y*sin(sphs);
yp = y*cos(sphs) - x*sin(sphs);
gaus2d = exp(-(xp.^2 + yp.^2) ./ (2*width^2));
sine2d = sin( 2*pi*.02*xp );
img = sine2d .* gaus2d;
```

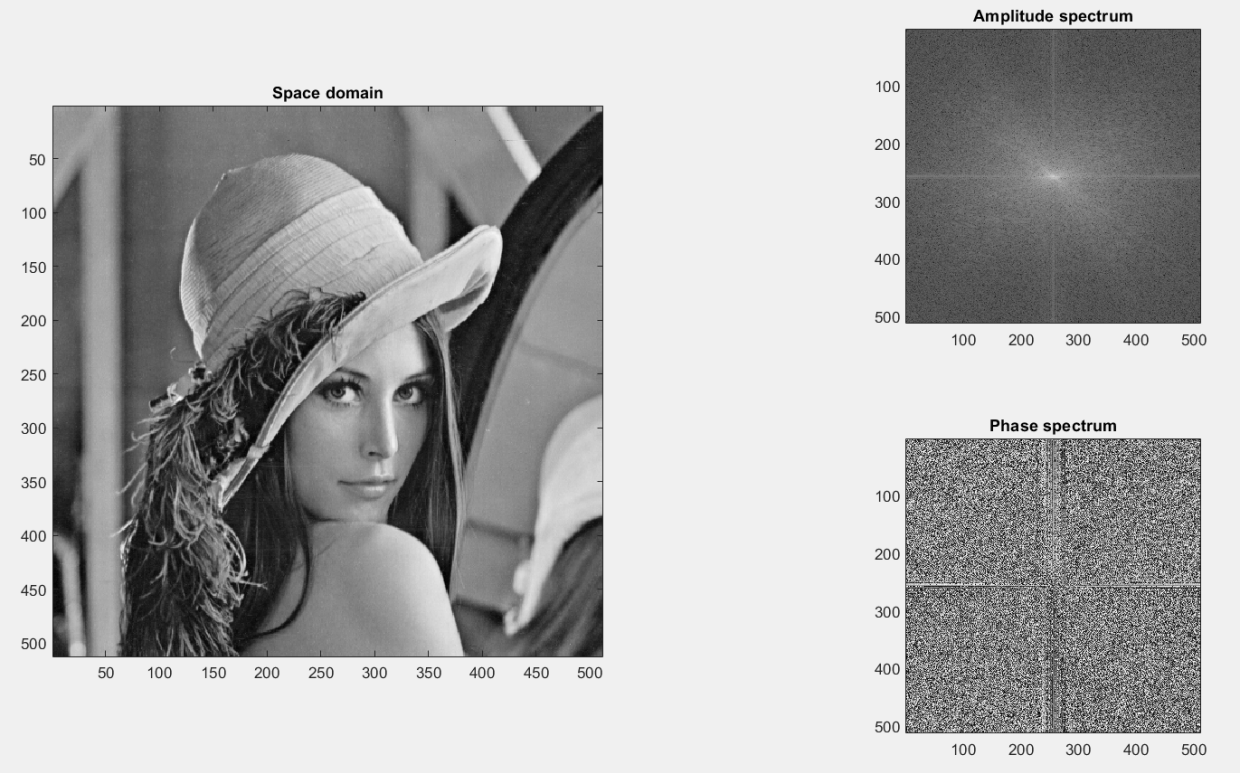
WHITE NOISE




```
%%% white noise
img = randn(size(img));
```

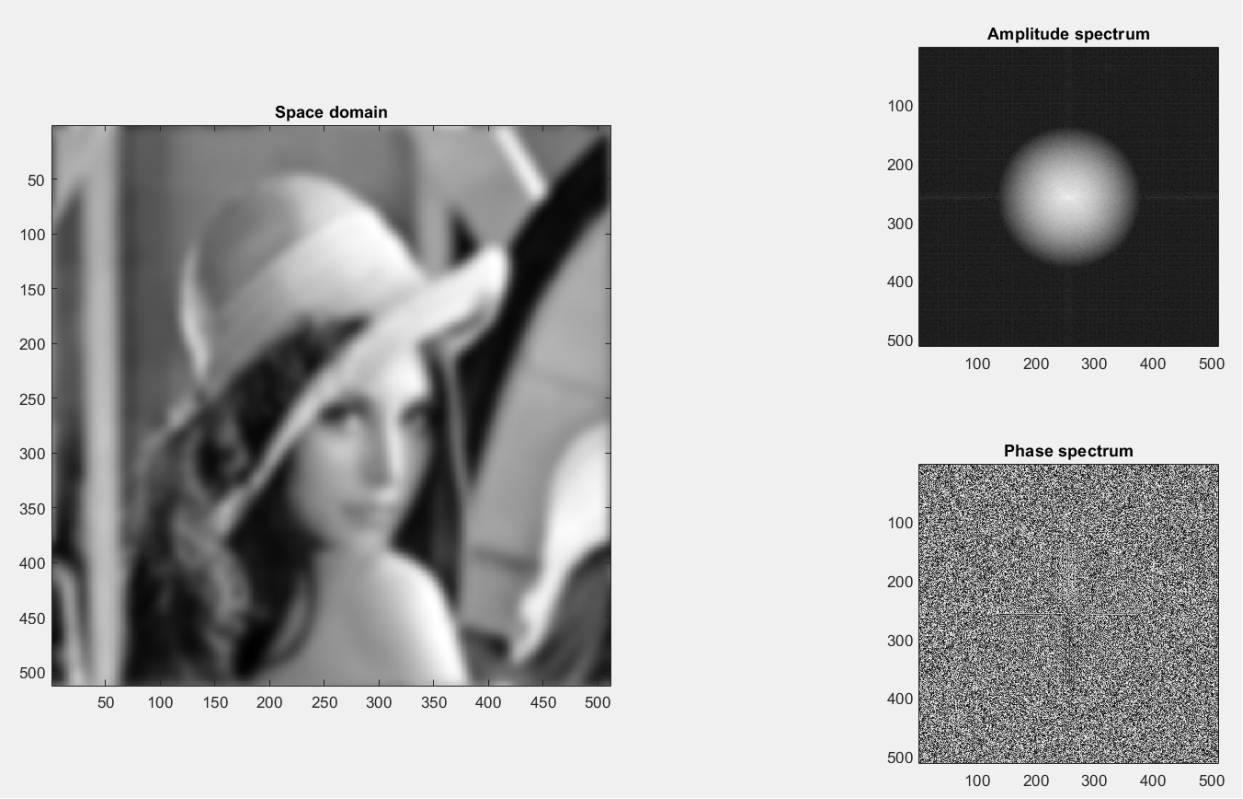
IMAGE

Use colormap gray to convert image to a gray scale as needed after plotting.



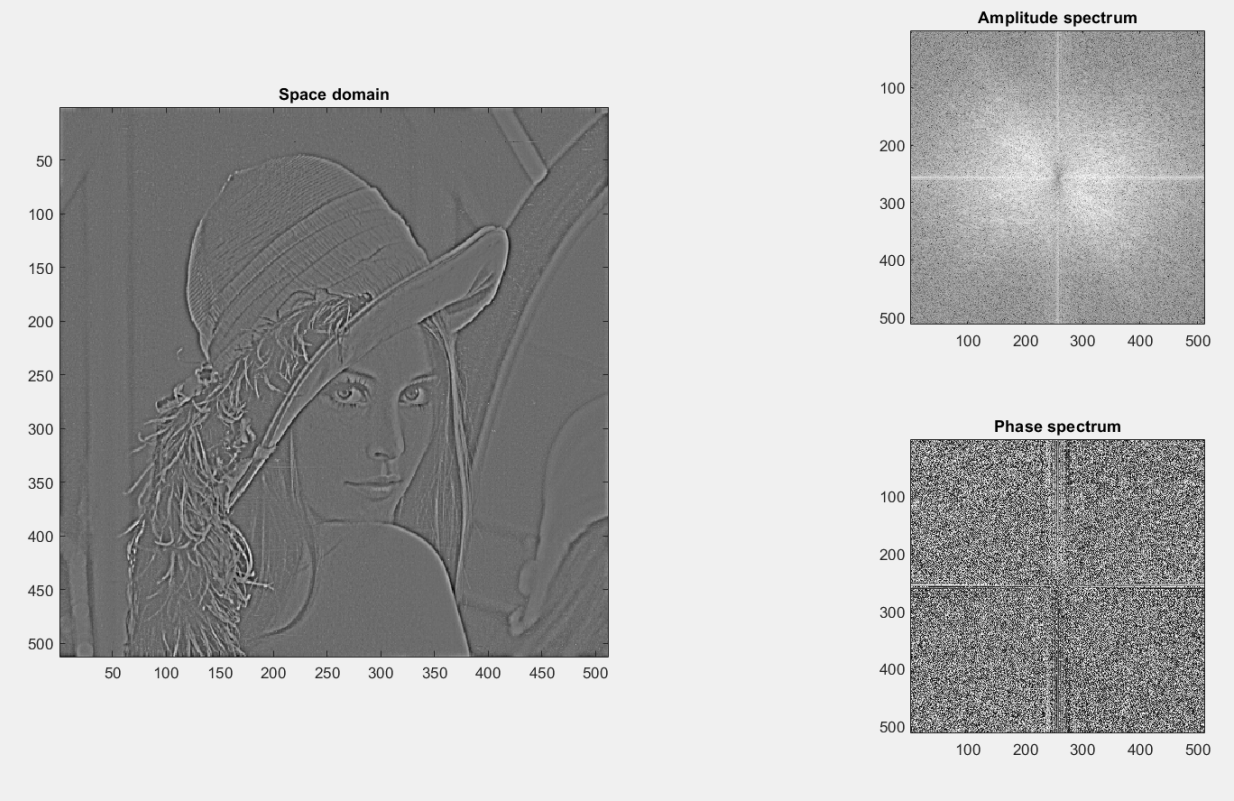

```
%%% portrait
lenna = imread('Lenna.png');
img   = double(mean(lenna,3));
```

LOW-PASS FILTERED IMAGE



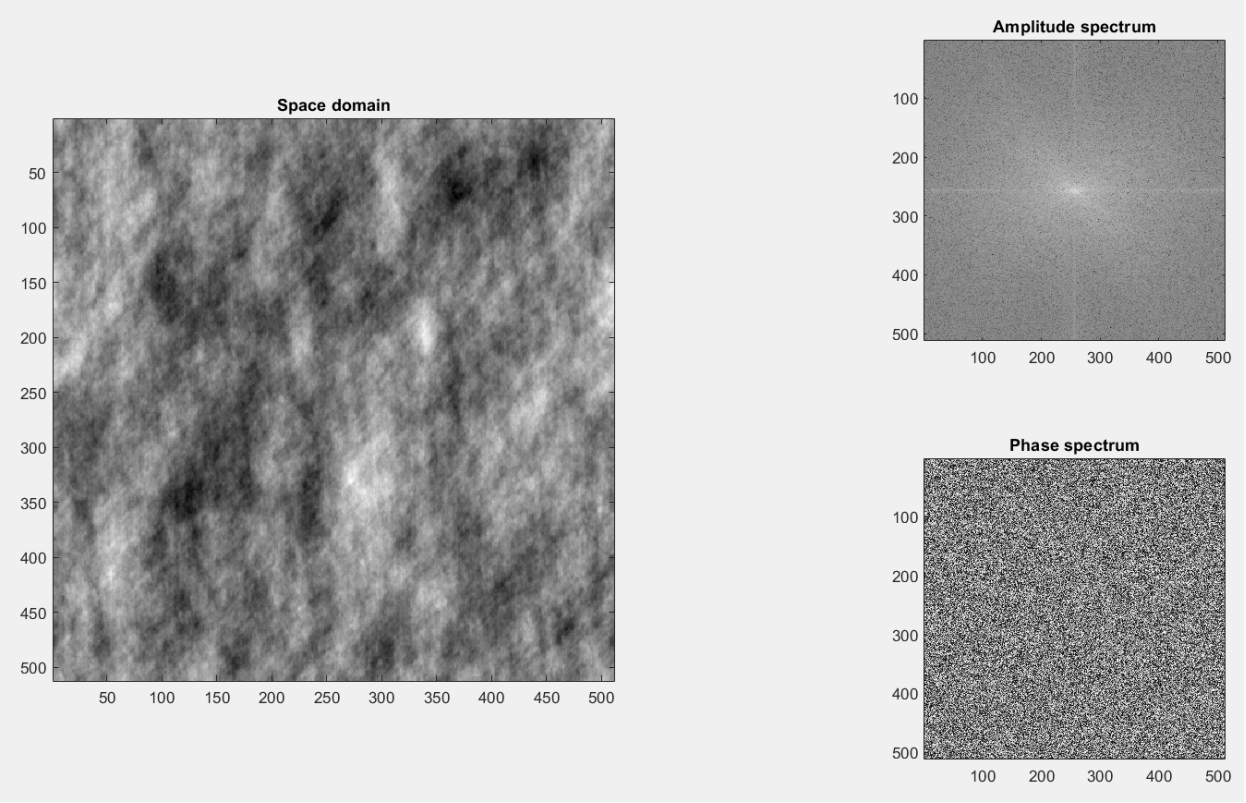

```
%%% portrait
lenna = imread('Lenna.png');
img   = double(mean(lenna,3));
%
%
% %%% low-pass filtered Lenna
imgL = img;
width = .1;   % width of gaussian (normalized Z units)
[x,y] = ndgrid(zscore(1:size(imgL,1)),zscore(1:size(imgL,2)));
gaus2d= exp(-(x.^2 + y.^2) ./ (2*width^2)); % add 1- at beginning to invert filter
imgX  = fftshift(fft2(imgL));
img   = real(ifft2(fftshift(imgX.*gaus2d)));
```

HIGH-PASS FILTERED IMAGE




```
%%% portrait
lenna = imread('Lenna.png');
img   = double(mean(lenna,3));
%
%%% high-pass filtered Lenna
imgL = img;
width = .3;   % width of gaussian (normalized Z units)
[x,y] = ndgrid(zscore(1:size(imgL,1)),zscore(1:size(imgL,2)));
gaus2d= 1-(exp(-(x.^2 + y.^2) ./ (2*width^2))); % add 1- at beginning to invert filter
imgX  = fftshift(fft2(imgL));
img   = real(ifft2(fftshift(imgX.*gaus2d)));
```

PHASE-SCRAMBLED IMAGE




```
%%% portrait
lenna = imread('Lenna.png');
img   = double(mean(lenna,3));
%
%%% phase-scrambled Lenna
imgL = img;
imgX  = fftshift(fft2(imgL));
powr2 = abs(imgX);
phas2 = angle(imgX);
img   = real(ifft2(fftshift( powr2.*exp(1i*reshape(phas2(randperm(numel(phas2))),size(phas2))) )));
```

SECTION 2: FOUNDATIONS OF THE FOURIER TRANSFORM

9. EULER'S FORMULA E^IK

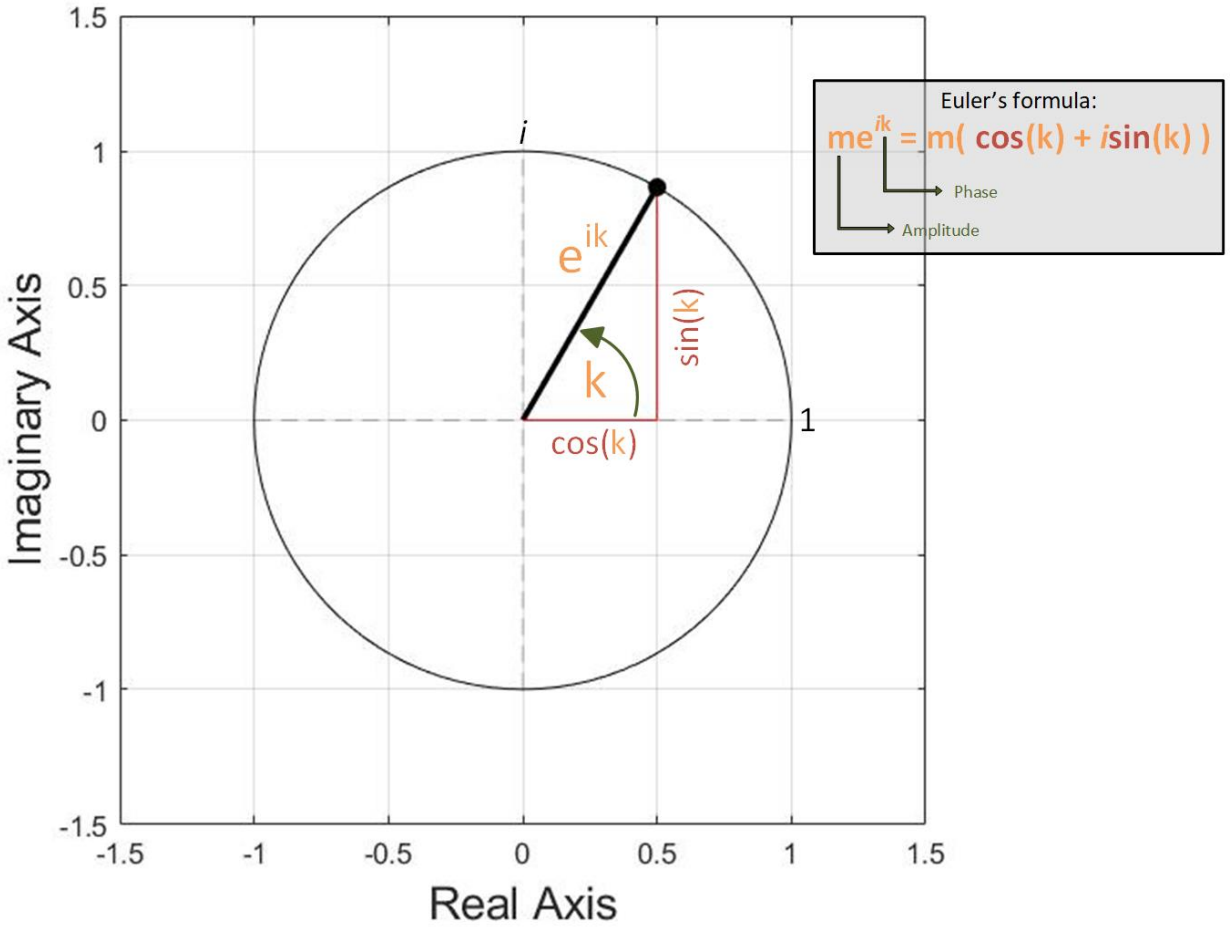


Figure showing relationship between phase angle and complex plane using Euler's formula.

MATLAB

Code: Fourier_foundations.m

```
% several ways to create a complex number
z = 4 + 3i;
z = 4 + 3*1i;
z = 4 + 3*sqrt(-1);
z = complex(4,3);

disp([ 'Real part is ' num2str(real(z)) ' and imaginary part is ' num2str(imag(z)) '.' ])
```

Real part is 4 and imaginary part is 3.

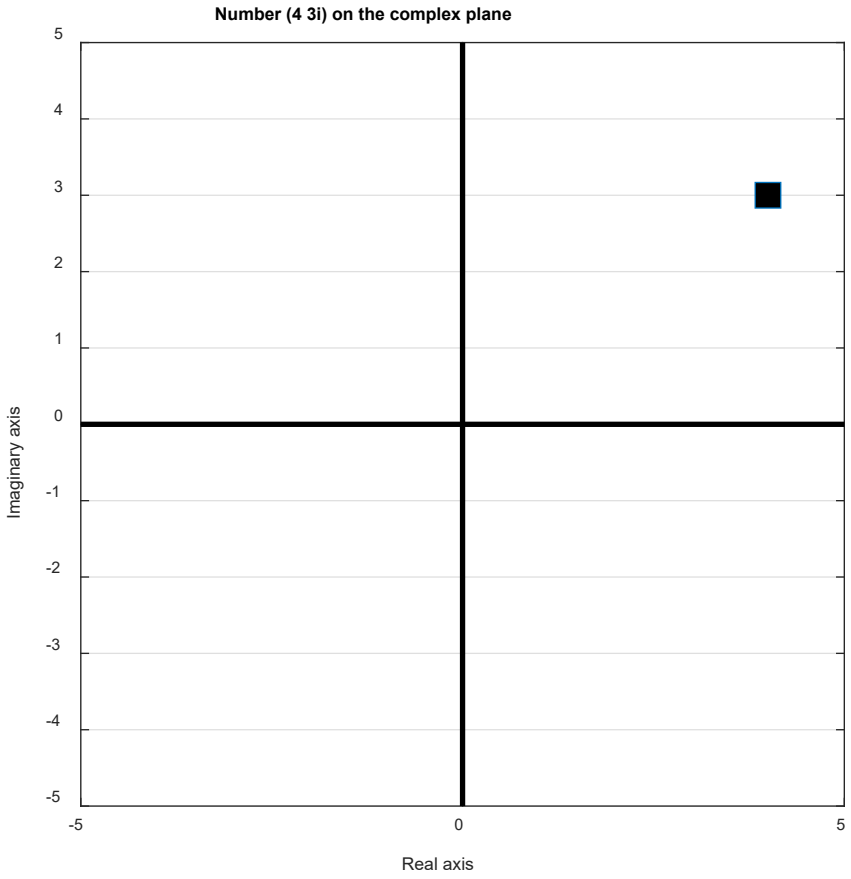


Figure 1: Lines of code from 23 to 34

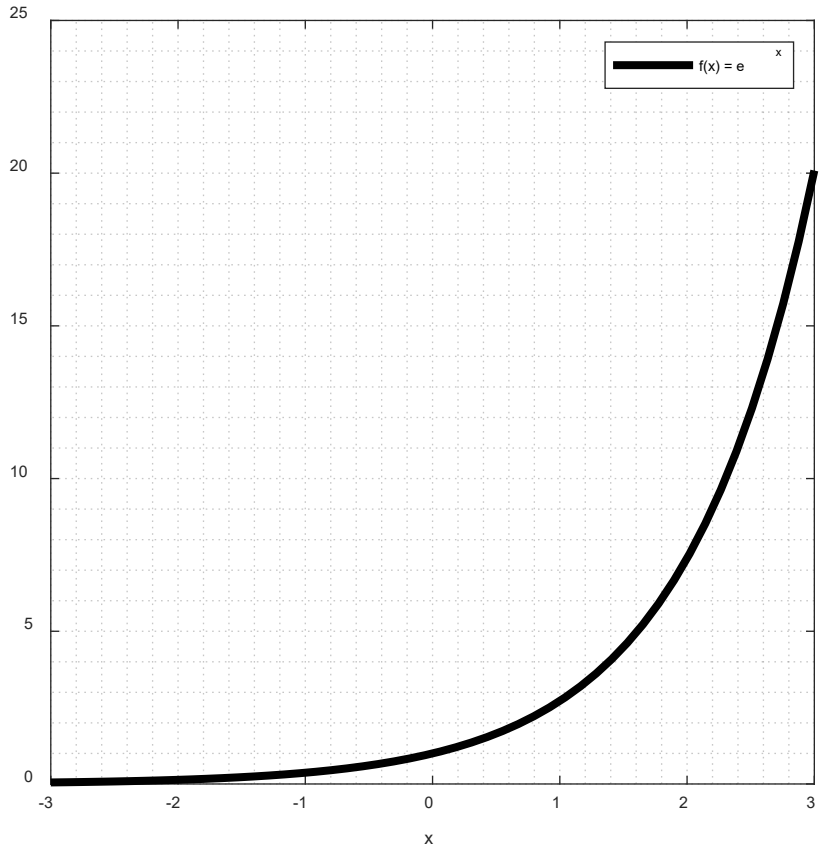


Figure 2: Lines of code from 53 to 62

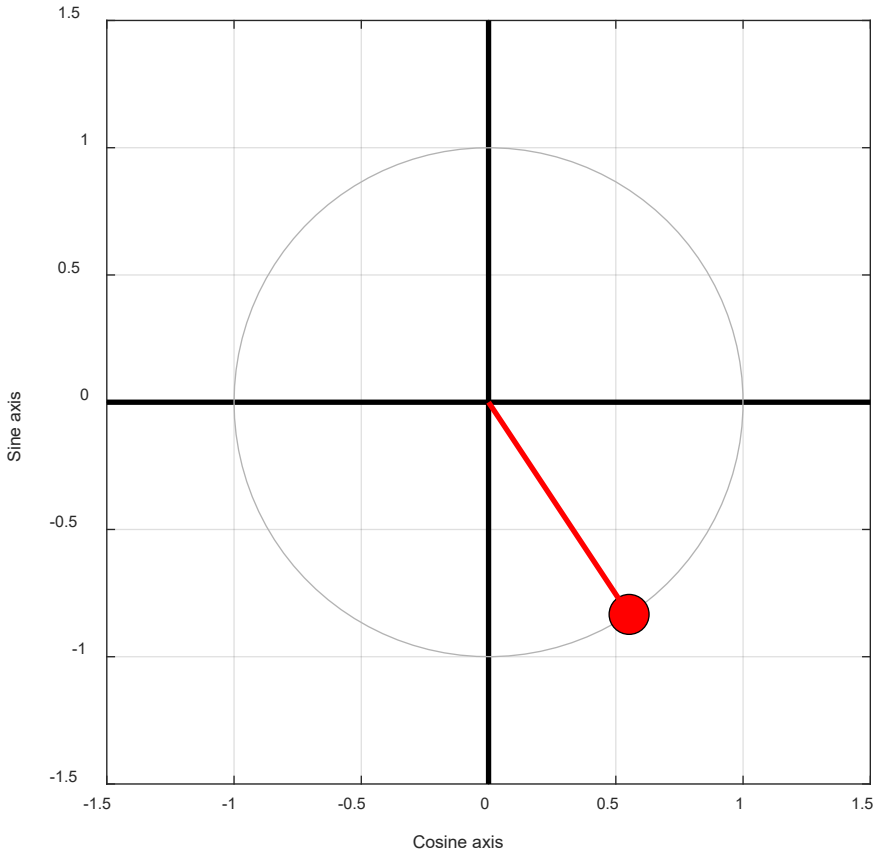


Figure 3: Lines of code from 63 to 94

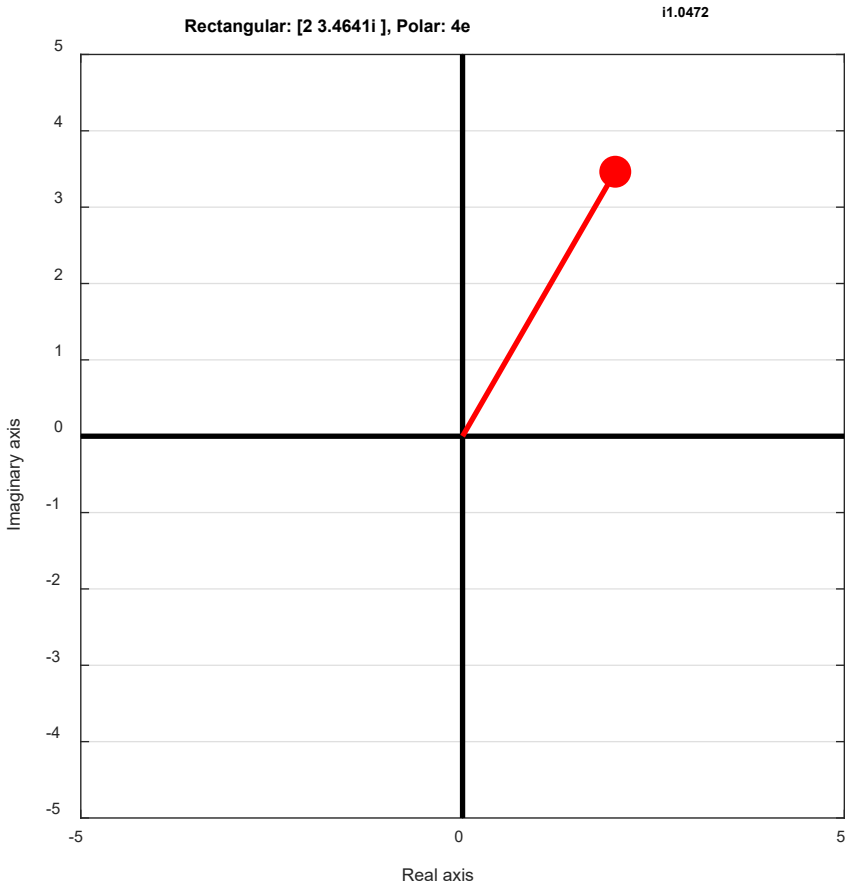


Figure 4: Lines of code from 95 to 116

10. SINE WAVES AND COMPLEX SINE WAVES

Sine Equation:

$$a\sin(2\pi ft + \theta)$$

Where

a = amplitude

f = frequency

theta = phase

Complex sine wave: Sine wave + Euler's formula

$$e^{ik} = \cos(k) + i\sin(k)$$

$$k = 2\pi ft + \theta$$

$$e^{i2\pi ft}$$

MATLAB

Code: Fourier_foundations.m

GUI: sinewave_from_params.m

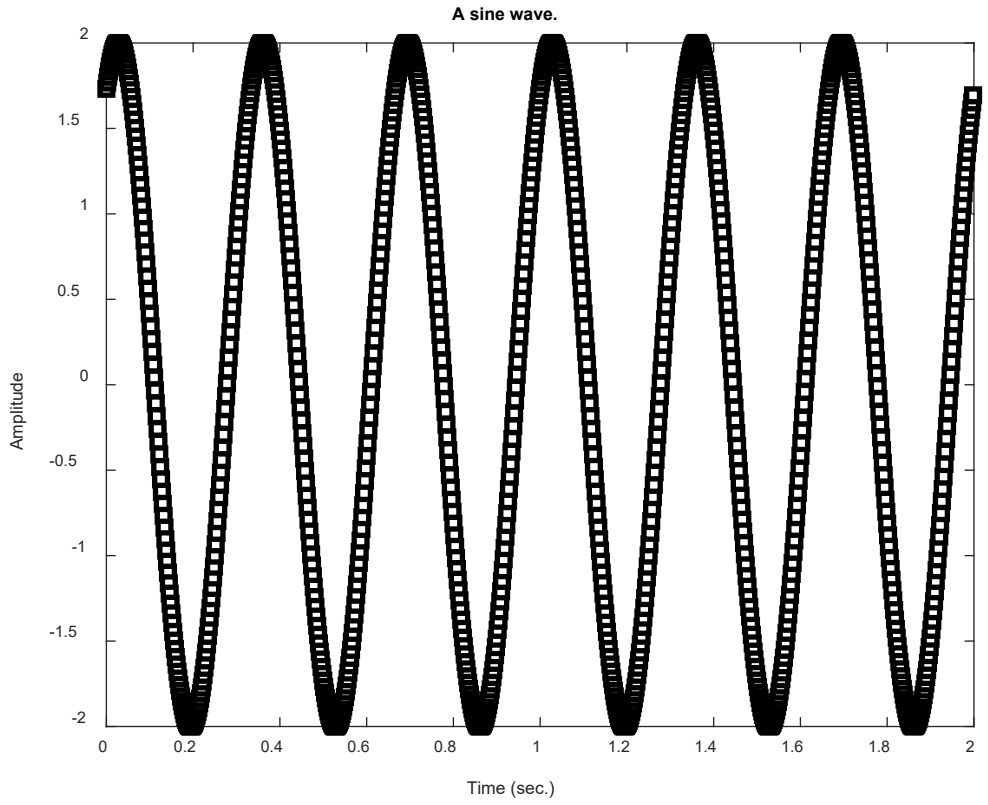


Figure 1: Lines of code from 134 to 151

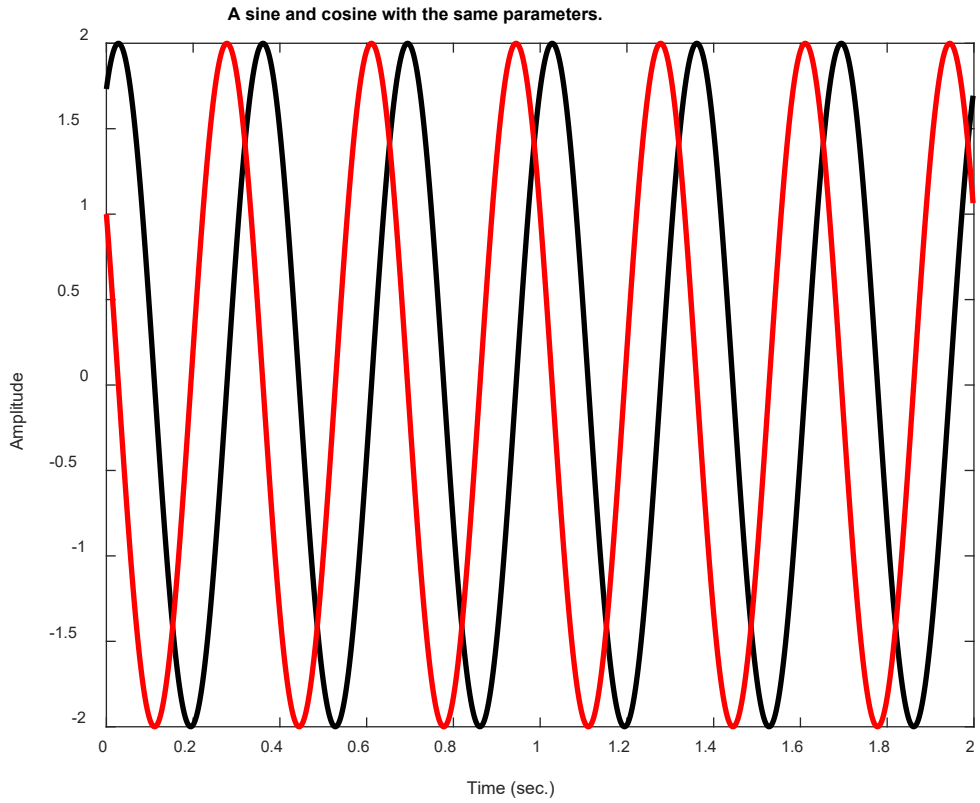


Figure 2: Lines of code from 153 to 165

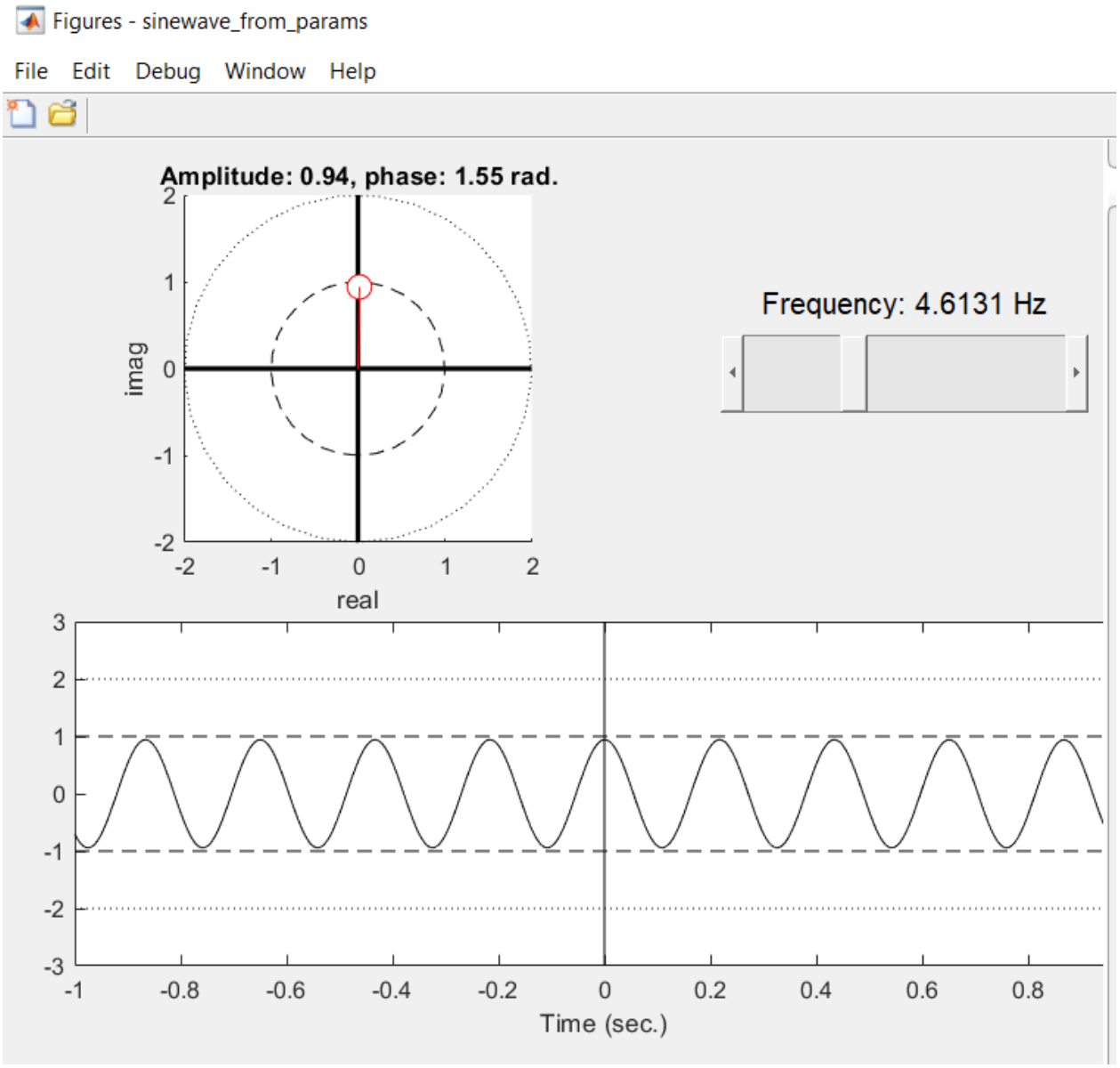
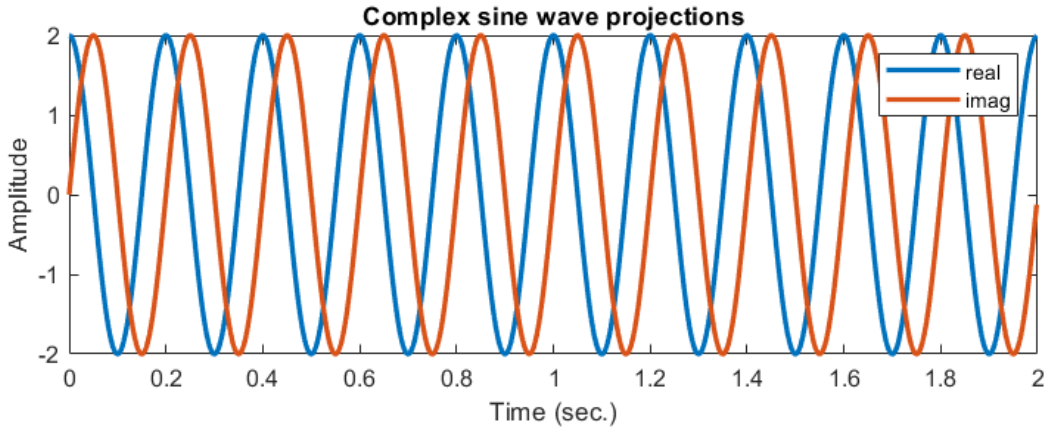


Figure 3: Lines of code from 166 to 171



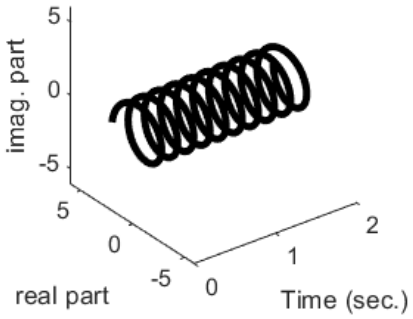


Figure 3 (again): Lines of code: 173 to 199

11. DOT PRODUCT

A single number that tells you about the relationship between two vectors of equal size (same number of elements).

$$\begin{bmatrix}1\\3\\0\\5\end{bmatrix} \cdot \begin{bmatrix}0\\8\\1\\6\end{bmatrix} = 1*0+3*8+0*1+5*6 = 54$$

Examples of dot product:

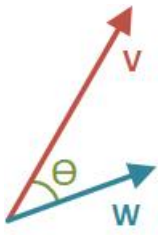
V1 = [1 3 0 5];

V2 = [0 8 1 6];

Dp = sum(v1.*v2);

Dp = dot(v1, v2);

Definition of Dot Product: The product of vector magnitudes scaled by angle between them:



$$d = |\mathbf{V}||\mathbf{W}| \cos(\theta_{vw})$$

Dot product between two vectors:

- Product of two vectors less than pi/2 or 90 degrees is a positive number.
- Product of two vectors that are pi/2 or 90 degrees is zero.
- Product of two vectors greater than pi/2 or 90 degrees is a negative number.

Dot product between sine waves:

- Product between two identical sine waves (dp=dot(sinewave1,sinewave2)) is not zero.
- Product between two sine waves with different frequencies in steps of 1 or 0.5 will be zero.
- Product between two sine waves with different frequencies in steps other than 1 or 0.5 will be a number less than the product between two identical sine waves. This effect is due to the sampling rate in digital computation.
- Product between two sine waves with phase angles that differ by pi/2 or 90 degrees (a sine wave and a cosine wave) will be zero.

When normalized, the dot product is the correlation coefficient:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\ \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Reference: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient#For_a_sample

MATLAB

Code: Fourier_foundations.m

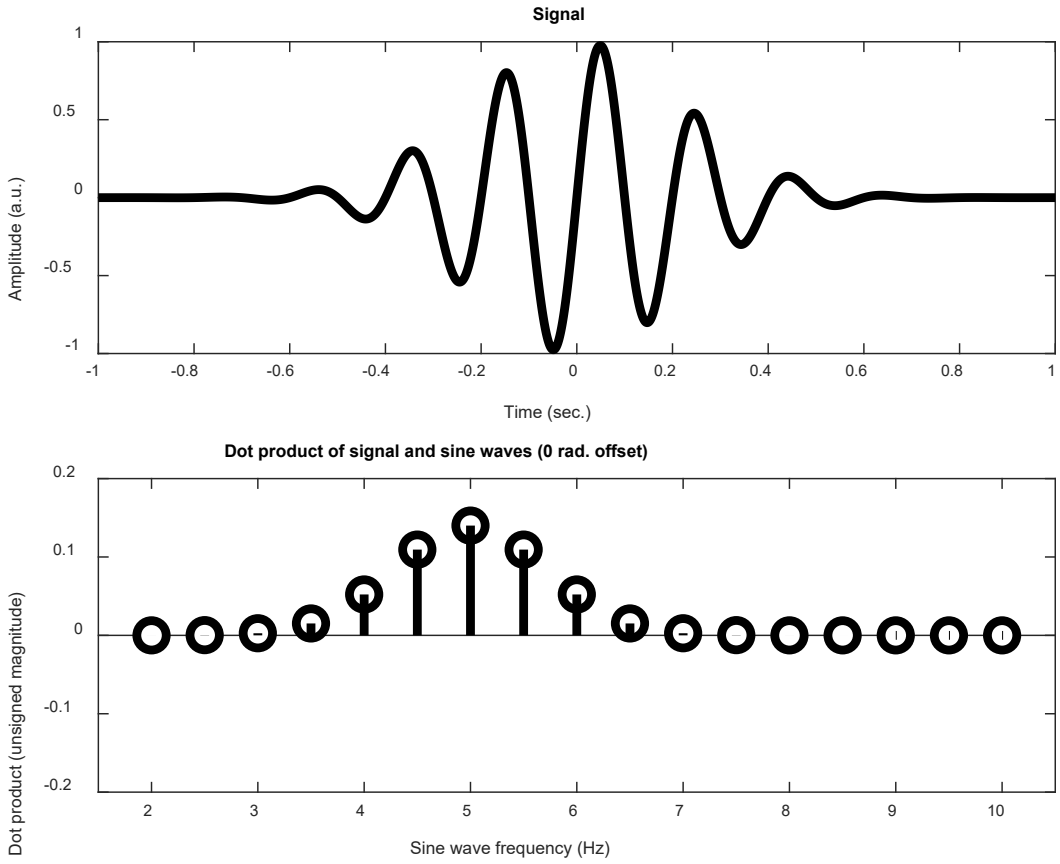
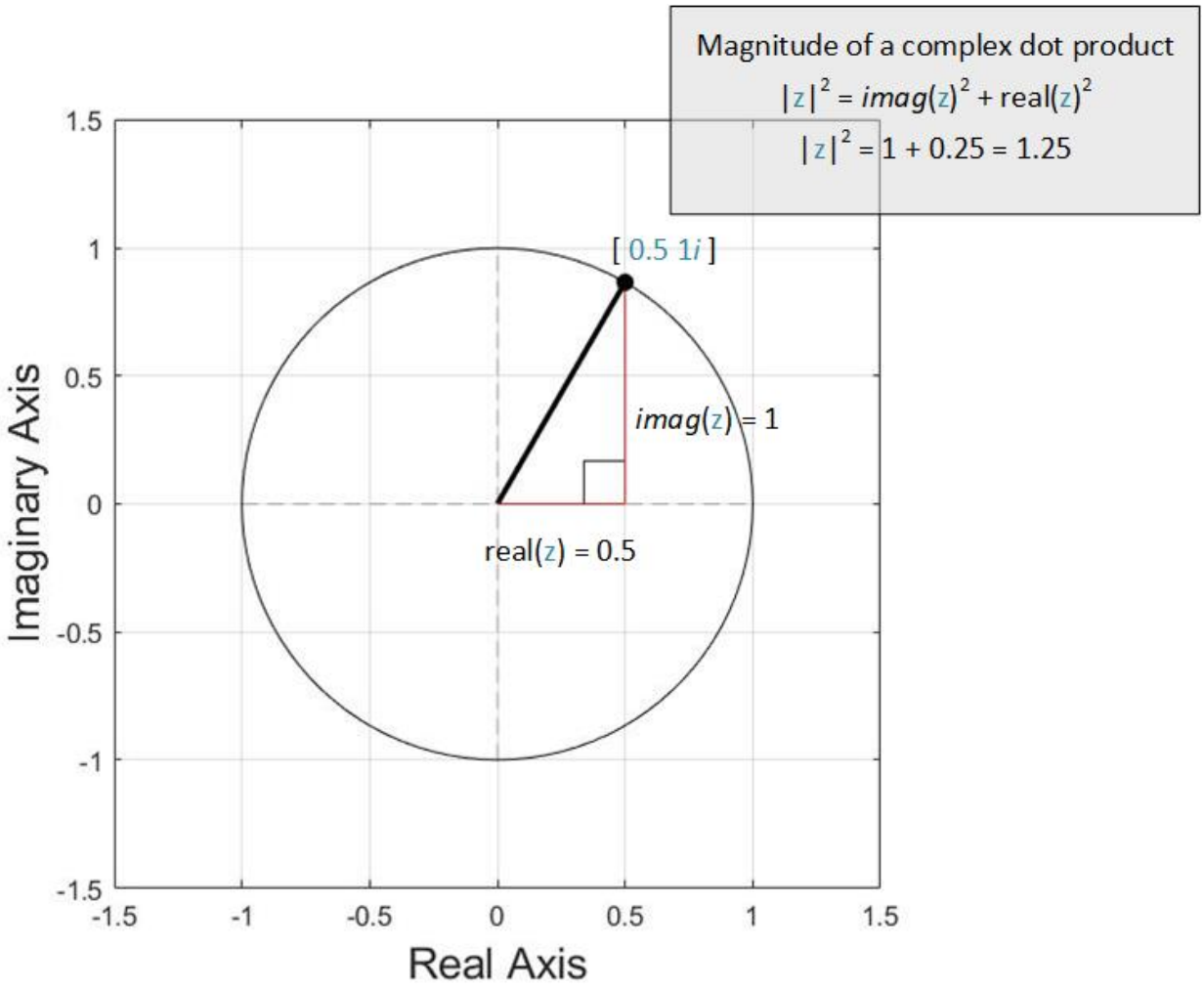


Figure: Lines of code from 219 to 285. The dot product between a signal (Morlet wavelet) and sine waves at frequencies ranging before, at, and after the signal frequency of 5 Hz.

The dot product of signal and sine waves is highly dependent on their phase relationship. At some phase angles the resulting dot product for all sine waves will be zero.

12. COMPLEX DOT PRODUCT

Use a complex dot product to encode magnitude and angle information within one calculation.



MATLAB

Code: Fourier_foundations.m

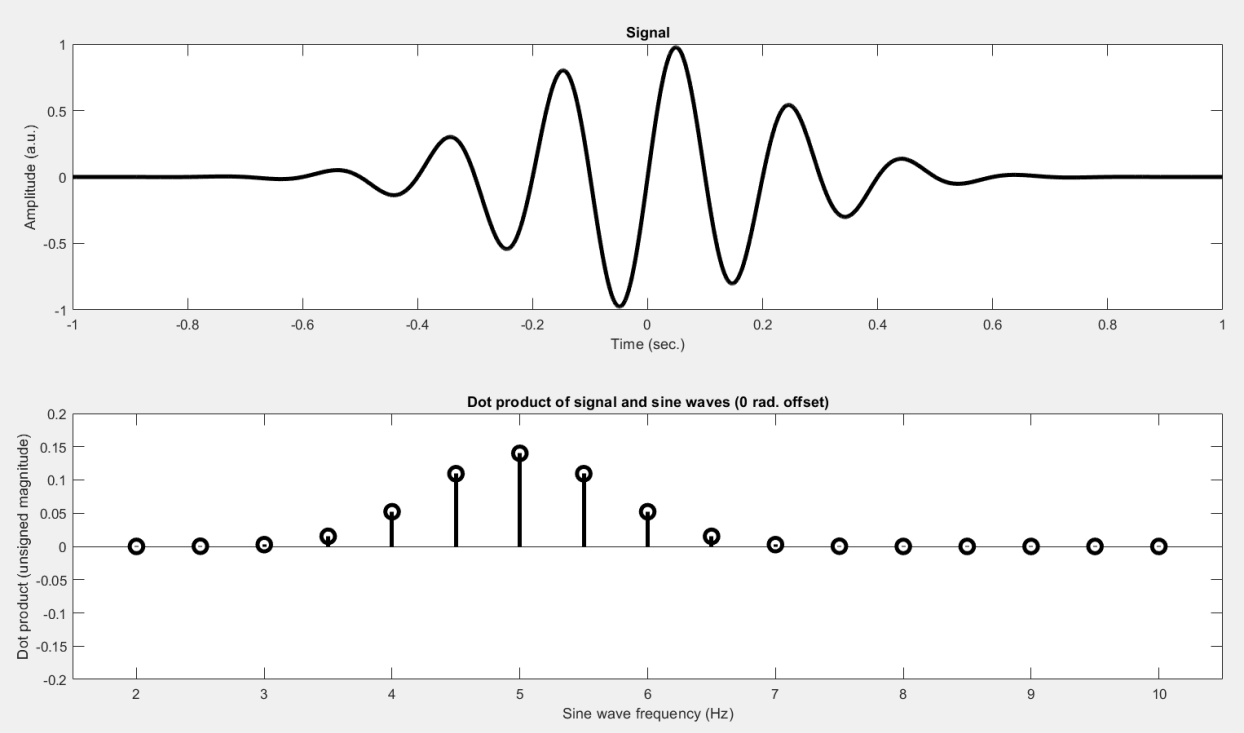


Figure: Lines of code from 296 to 336.

Same plot as before but with complex signal. Now phase angle does not affect the Dot product of signal and sine waves.

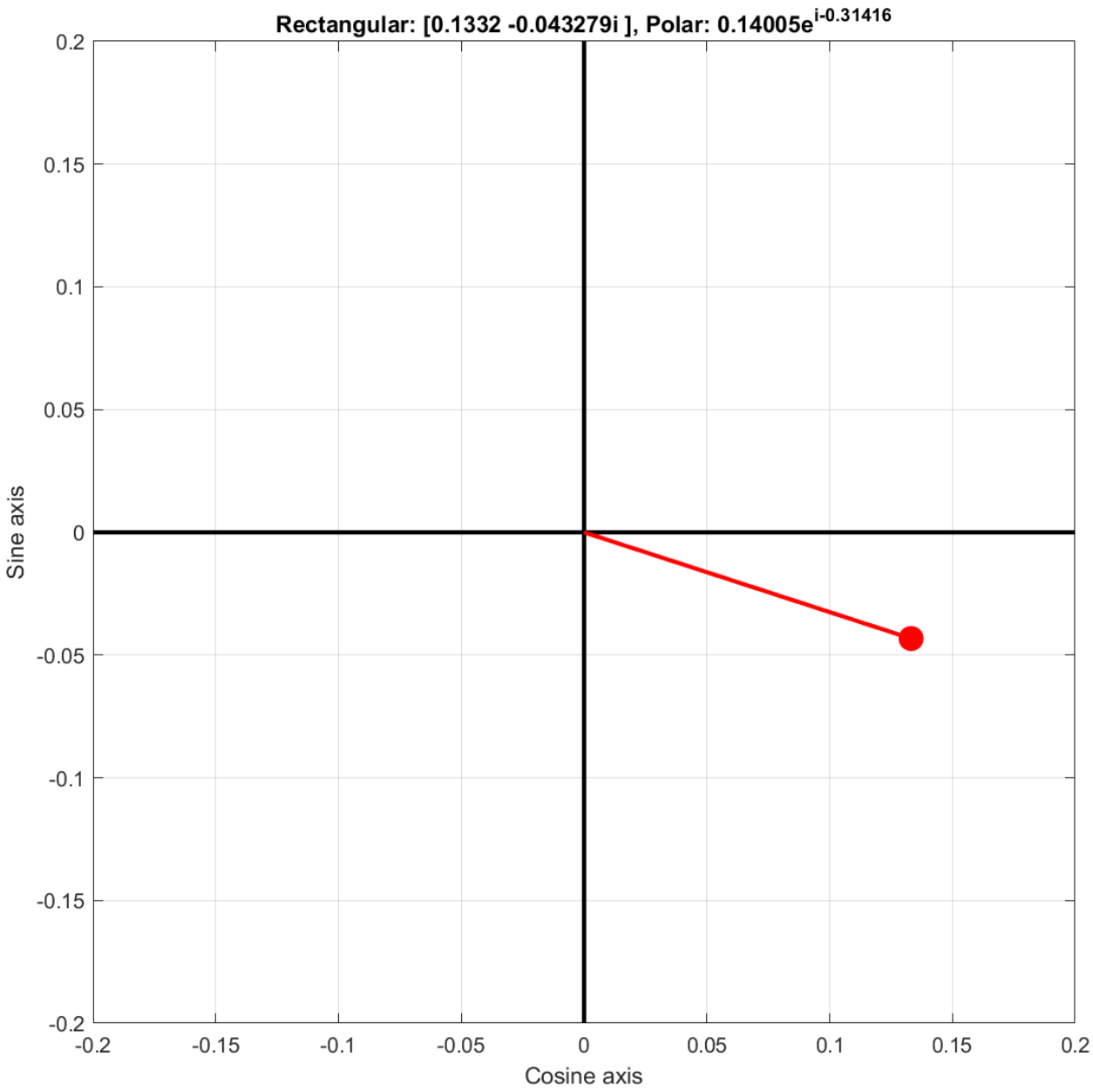


Figure: Lines of code from 337 to 377. Magnitude and phase components of dot product between signal and Gaussian for a phase of theta = 2.4*pi/4.

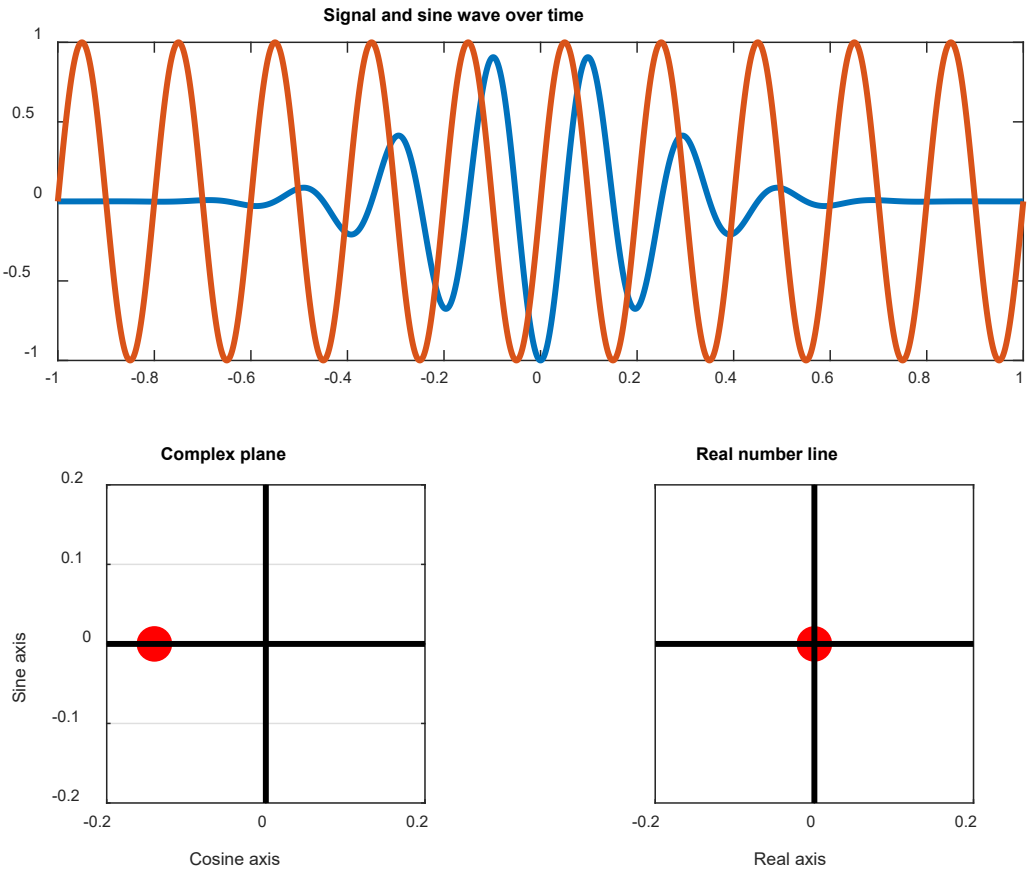


Figure: Lines of code from 378 to 438 (end). A snapshot during movie showing changes in signal phase in real and complex plane.

SECTION 3: THE DISCRETE FOURIER TRANSFORM

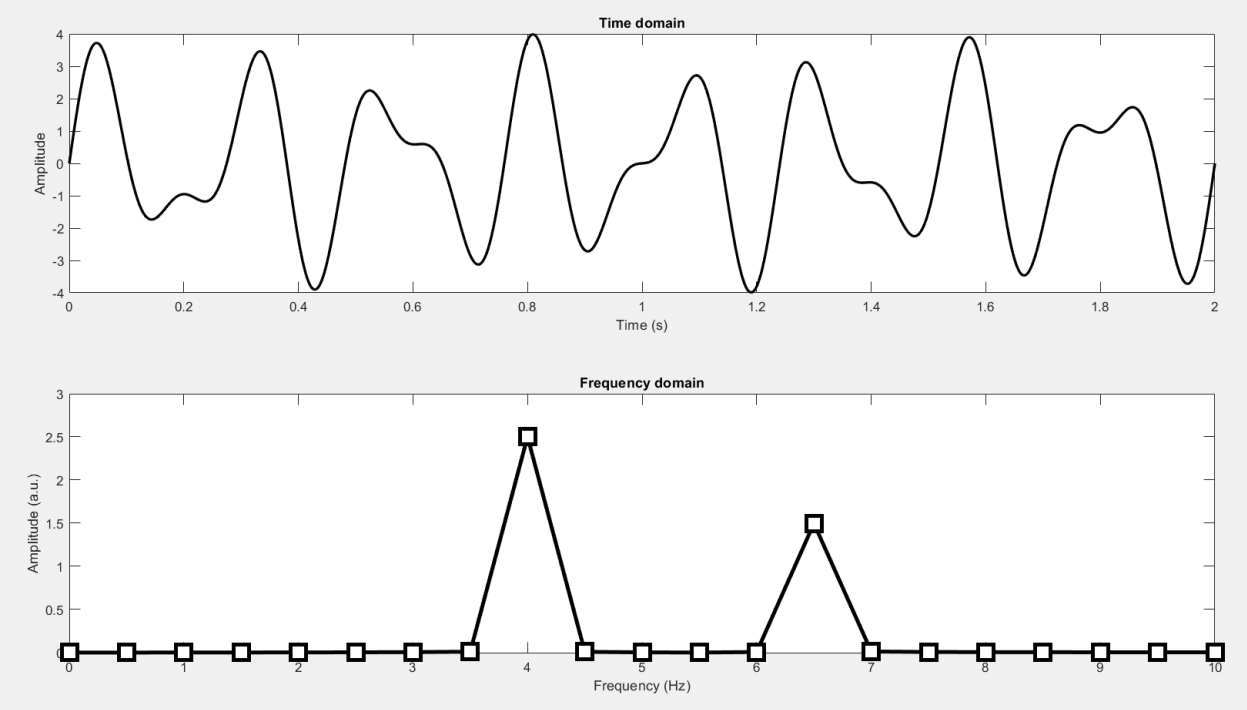
14. HOW THE DISCRETE FOURIER TRANSFORM WORKS

DTFT method:

- Loop over time points
  - Create complex sine wave with the same number of time points as the signal and a frequency defined by point-1
  - Compute dot product between complex sine wave and the signal
- Signal amplitude spectrum is magnitude of Fourier coefficients
  - Square amplitude to get power
- Signal phase spectrum is angle of Fourier coefficients

MATLAB

Code: Fourier_DTFT.m



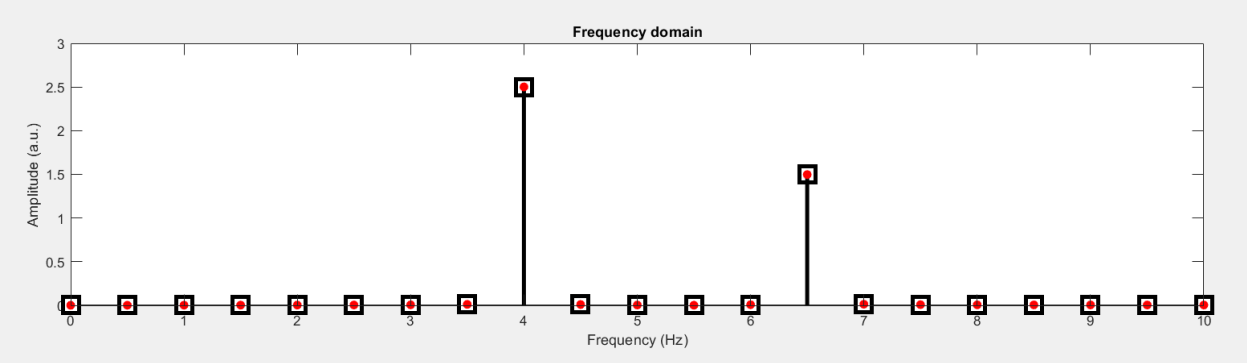


Figure: Code lines from 1 to 60 in the script

Comparison between dot product of signal with individual complex sine waves (black), and FFT function (red dots).

Comparison between line plot and stem plot for Fourier transform. Stem plot is usually preferred because it does not hide the limitations imposed by the sampling rate.

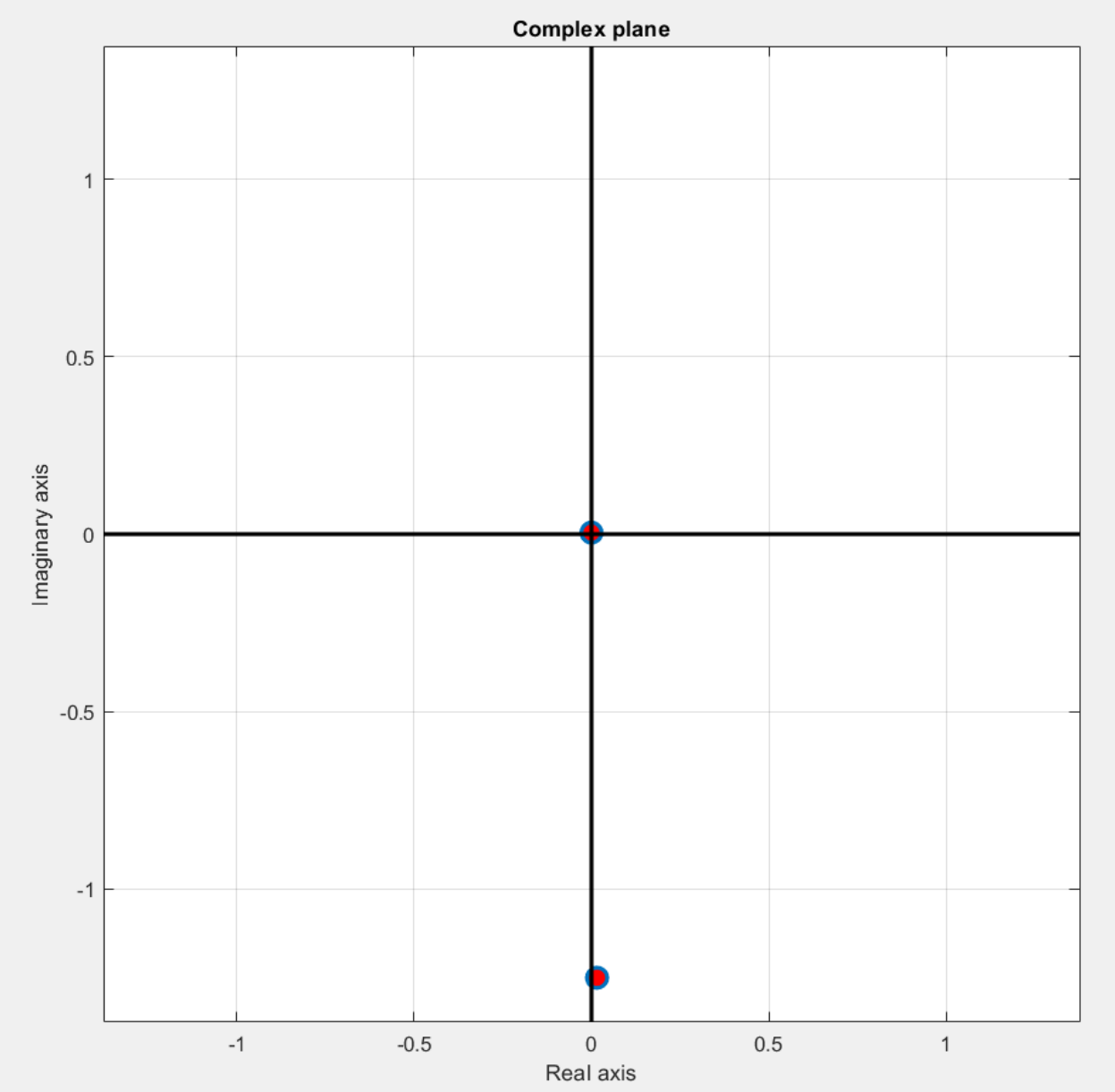


Figure: Code lines from 61 to 80 in the script.

Plot shows two Fourier coefficients in signal above, 4 and 4.5 Hz. The larger the magnitude (distance away from origin), the more similar the sine wave is to the signal (4 Hz in this case). The phase is also shown for the two data points.

15. CONVERTING INDICES TO FREQUENCIES

Prepare a Fourier Transform:

- Create the signal
- Prepare the Fourier transform
- Compute the frequencies vector

Nyquist frequency: ½ sample rate, where two data points per cycle are available

MATLAB

Code: Fourier_DTFT.m

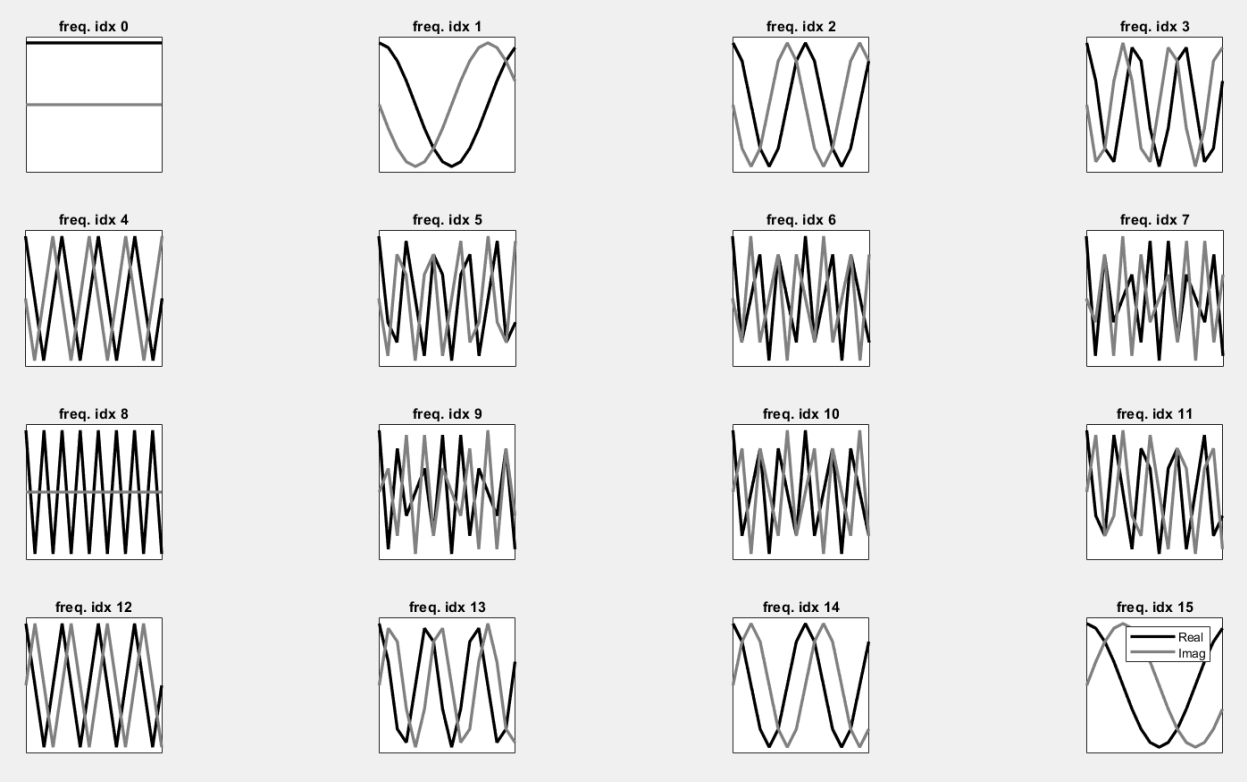


Figure: code lines 83 to 112 in the script. Black lines are real and grey lines are imaginary.

- idx=0 corresponds to a frequency of zero, or the DC component in a signal.
- idx=1 to 8 corresponds to frequencies up to the Nyquist frequency.
- idx=9 to 15 correspond to frequencies that are faster than the Nyquist frequency. So, the plots show a signal that is aliased into slower frequencies than the actual signal. These plots can also be called negative frequencies.

16. SHORTCUT: CONVERTING INDICES TO FREQUENCIES

MATLAB

Code: Fourier_DTFT.m

Typical Fourier transform development:

```
% Create the signal
srate = 1000;
time  = (0:srate)/srate;
npnts = length(time);

% Notice: A simple 15-Hz sine wave!
signal = sin(15*2*pi*time);
```

```
% Prepare the Fourier transform
fourTime = (0:pnts-1)/pnts;
```

```
% Compute frequencies vector
hz = linspace(0,srate/2,floor(pnts/2)+1);
```

Shortcut:

```
% Frequencies vector with odd number of data points
hz = linspace(0,srate,npnts);
```

```
% Frequencies vector with even number of data points
hz = linspace(0,srate,npnts+1);
```

```
% Frequencies vector for long signal: odd and even shortcuts converge
hz1 = linspace(0,srate,npnts+1);
hz2 = linspace(0,srate,npnts);
```

```
% Frequencies vector N/2+1 is always correct (ie no shortcut)
hz = linspace(0,srate/2,floor(npnts/2)+1);
```

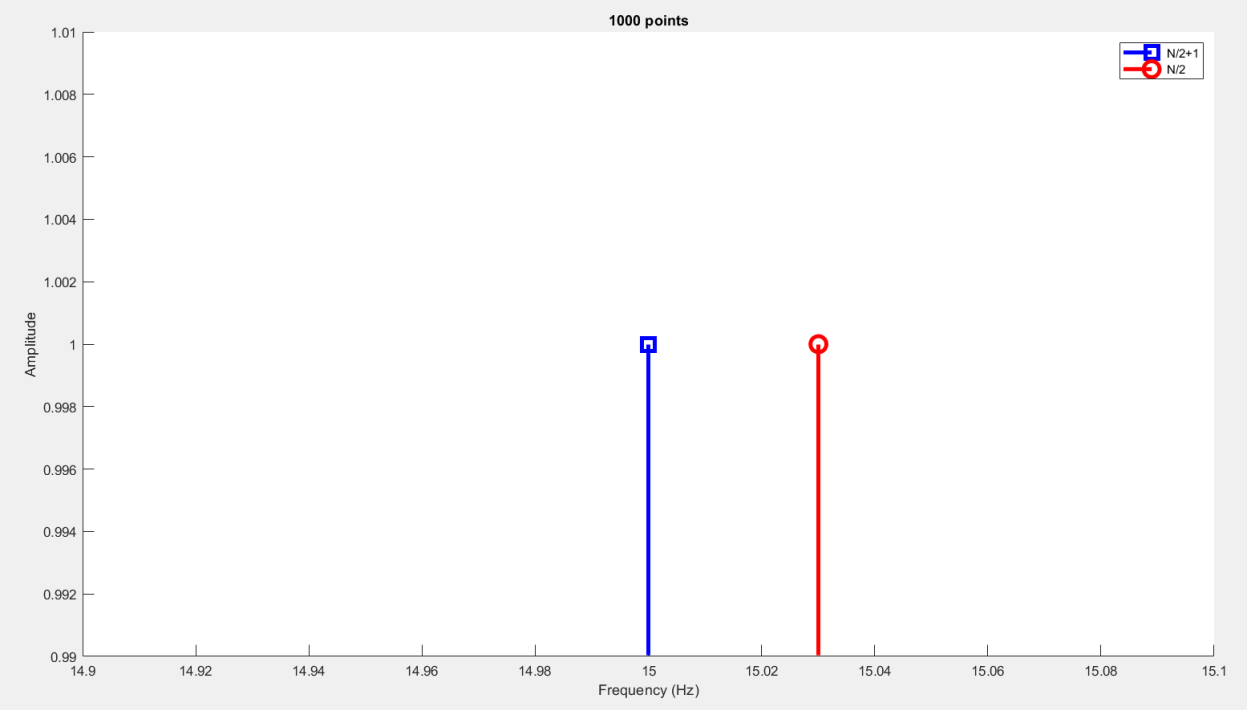


Figure: Code from 116 to 236

Demonstrate the effect of different shortcut methods.

17. NORMALIZED TIME VECTOR

Plot reconstructed time series on top of original time series with ifft

MATLAB

Code: Fourier_DTFT.m

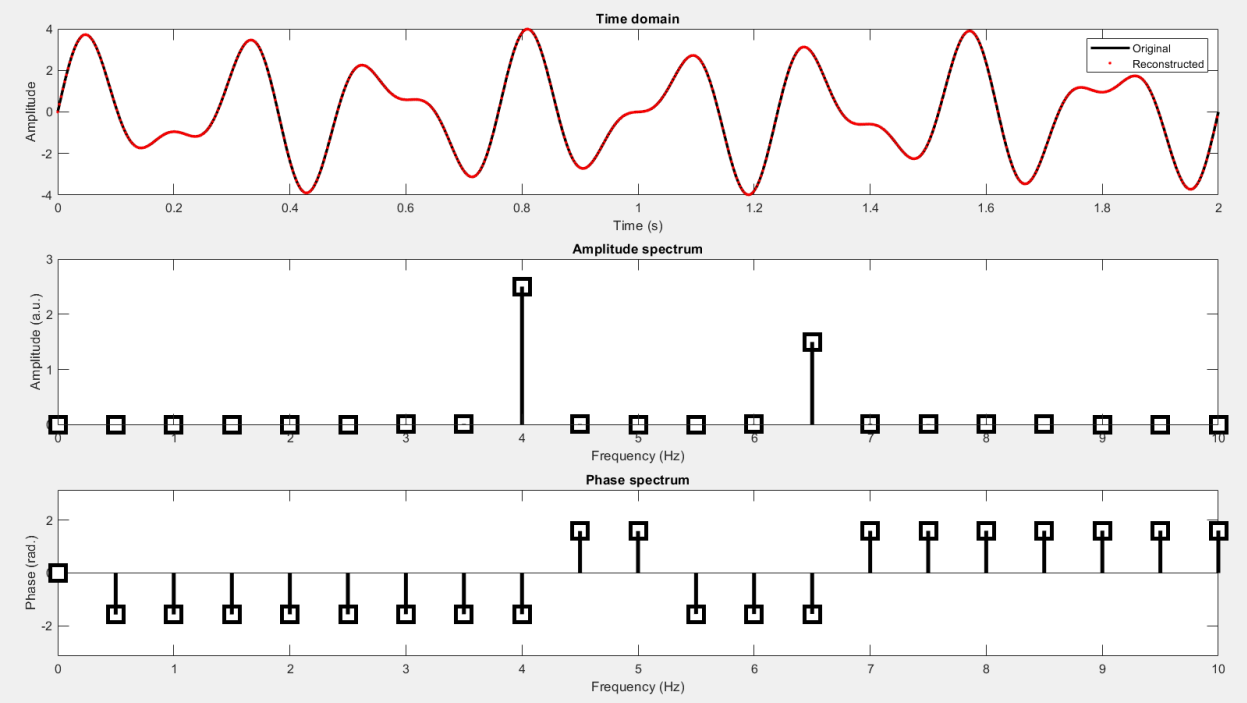


Figure: Plot the angles: code lines 237 to 307

18. POSITIVE AND NEGATIVE FREQUENCIES

The term negative comes from the phase offset from the complex sine waves above the Nyquist frequency.

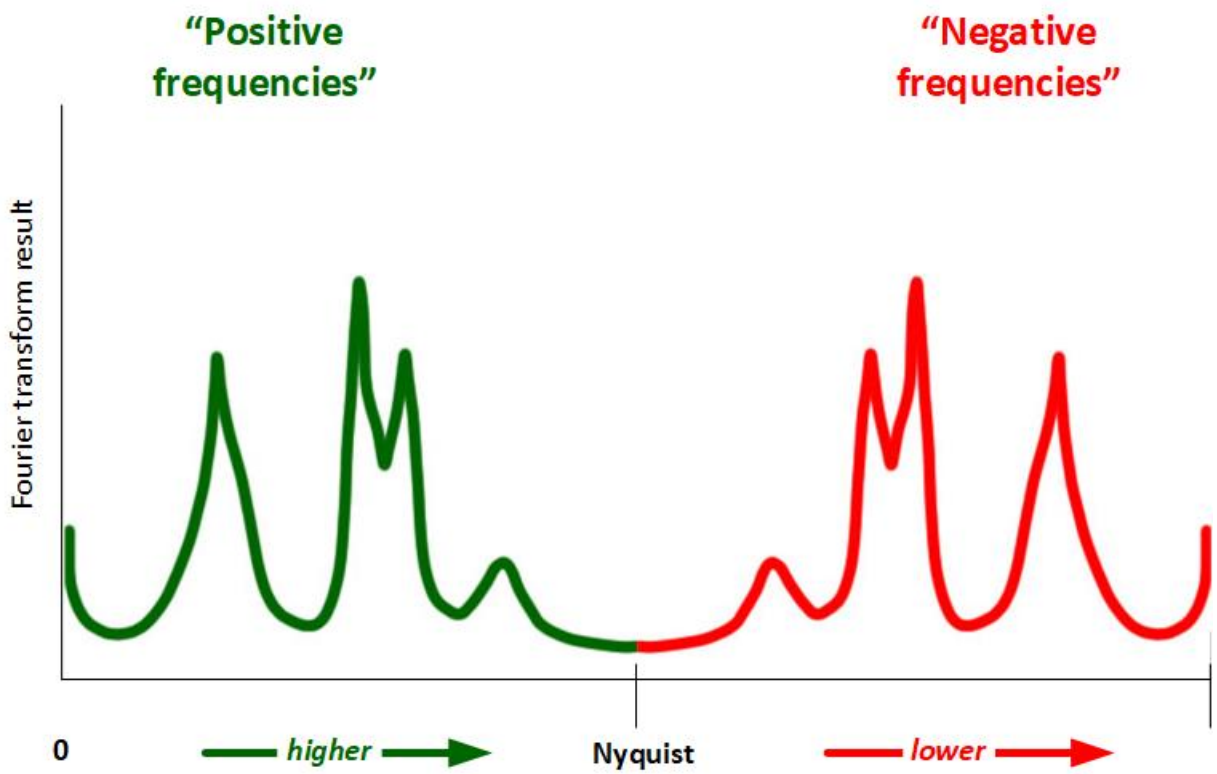


Figure shows symmetry between positive and negative frequencies in a Fourier transform

Equation for positive and negative frequencies (see Euler's equation)

$$\cos k = \frac{e^{ik} + e^{-ik}}{2}$$

or

$$\cos(2\pi f t) = \frac{e^{2\pi f t} + e^{-i2\pi f t}}{2}$$

Equation shows origin of positive and negative frequencies in Fourier transform, going back to Euler's equation

19. ACCURATE SCALING OF FOURIER COEFFICIENTS

The "raw" Fourier coefficients are not in the same units as the signal.

Two normalization factors to scale the Fourier coefficients accurately.

- Normalize dot product by dividing by number of points.
    - This is because the dot product is the summation of signal with many complex sine waves.
    - By dividing by the number of points, we get the average value. This in-turn is exactly one half the correct amplitude for each frequency peak.
    - Without dividing by the number of points, the amplitude will get larger with more points included in the summation.
- Double the absolute value of the dot product to remove complex negative frequencies while showing full amplitude.
    - Amplitude doubling should not include DC component (zero point) or Nyquist frequency.

MATLAB

Code: Fourier_DTFT.m

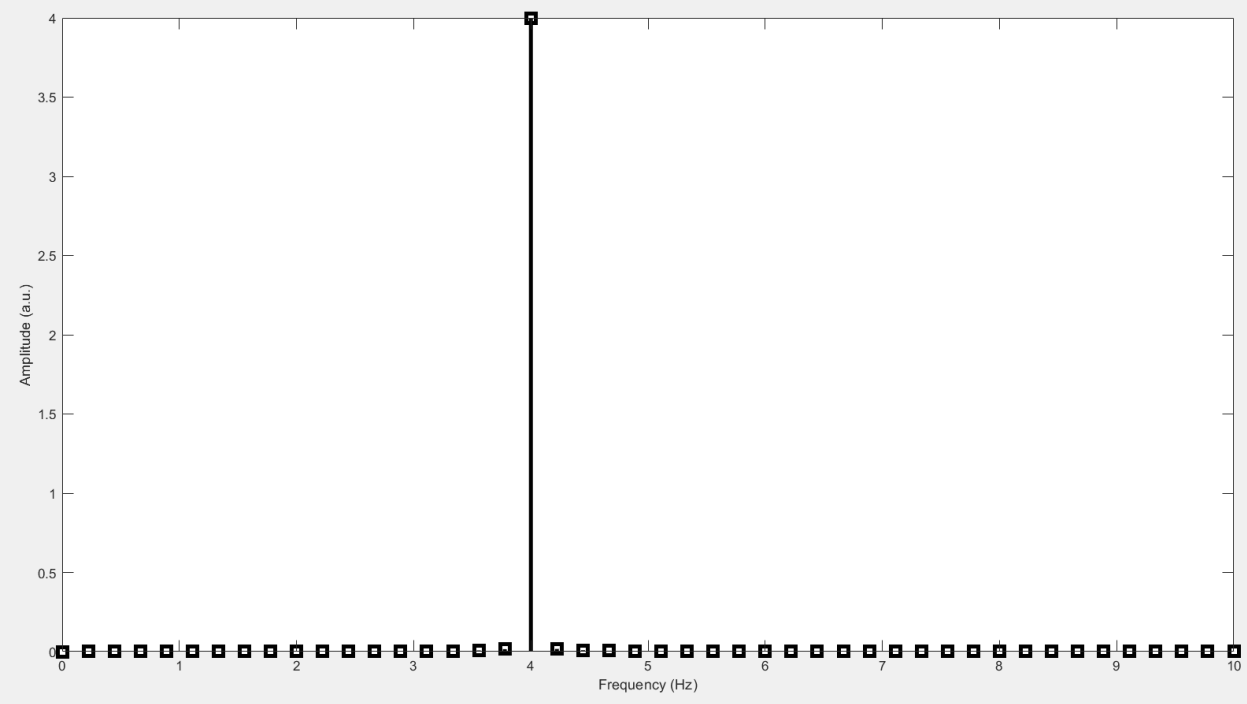


Figure: Lines of code from 308 to 355. With correct normalization the Fourier transform amplitude matches original signal

20. INTERPRETING PHASE VALUES

Phase is defined as the angle relative to the positive real axis of the vector that goes from the origin of the complex plane to the to the complex coefficient. See section 2 lesson 9 figure.

Phase independent of amplitude.

Phase is crucial when reconstructing a signal.

Normalization is not required when calculating and plotting phase.

Phase value is uncertain when amplitude is near zero and undefined when amplitude is zero.

MATLAB

Code: Fourier_DTFT.m

Code lines 356 to 433

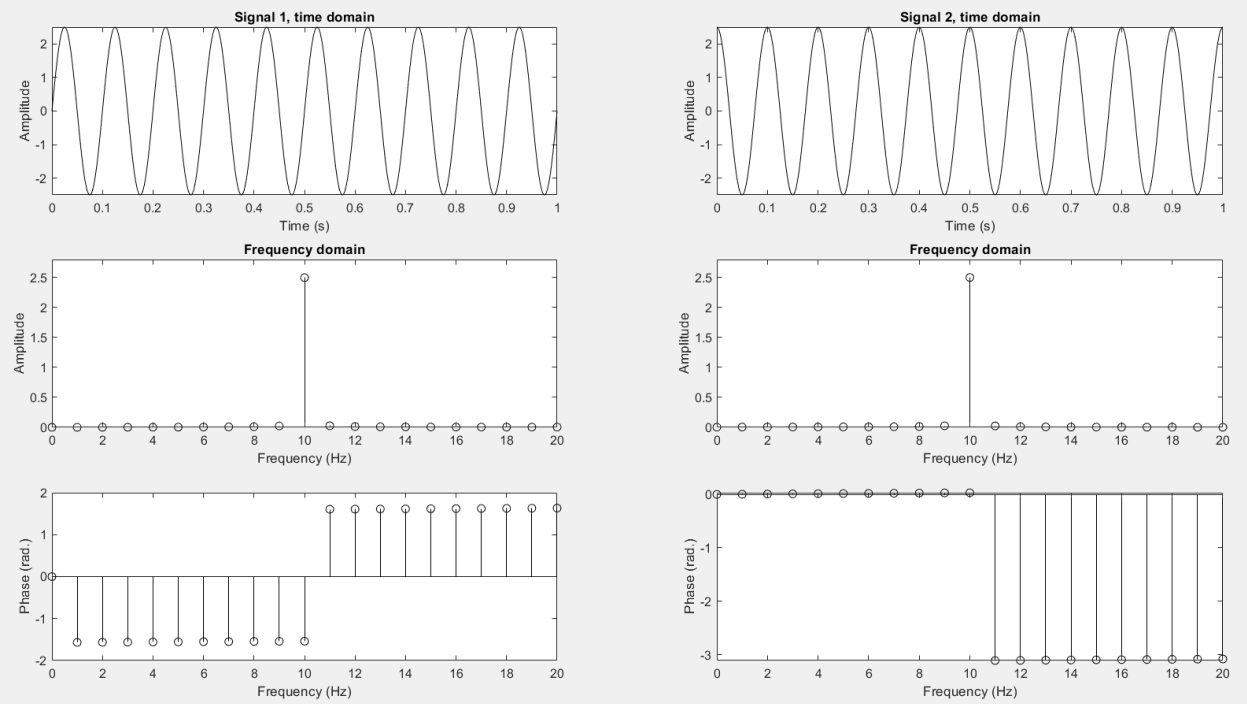


Figure shows that phase information is not available in amplitude vs frequency domain. But phase is needed to differentiate between signal 1 and signal 2.

21. AVERAGING FOURIER COEFFICIENTS

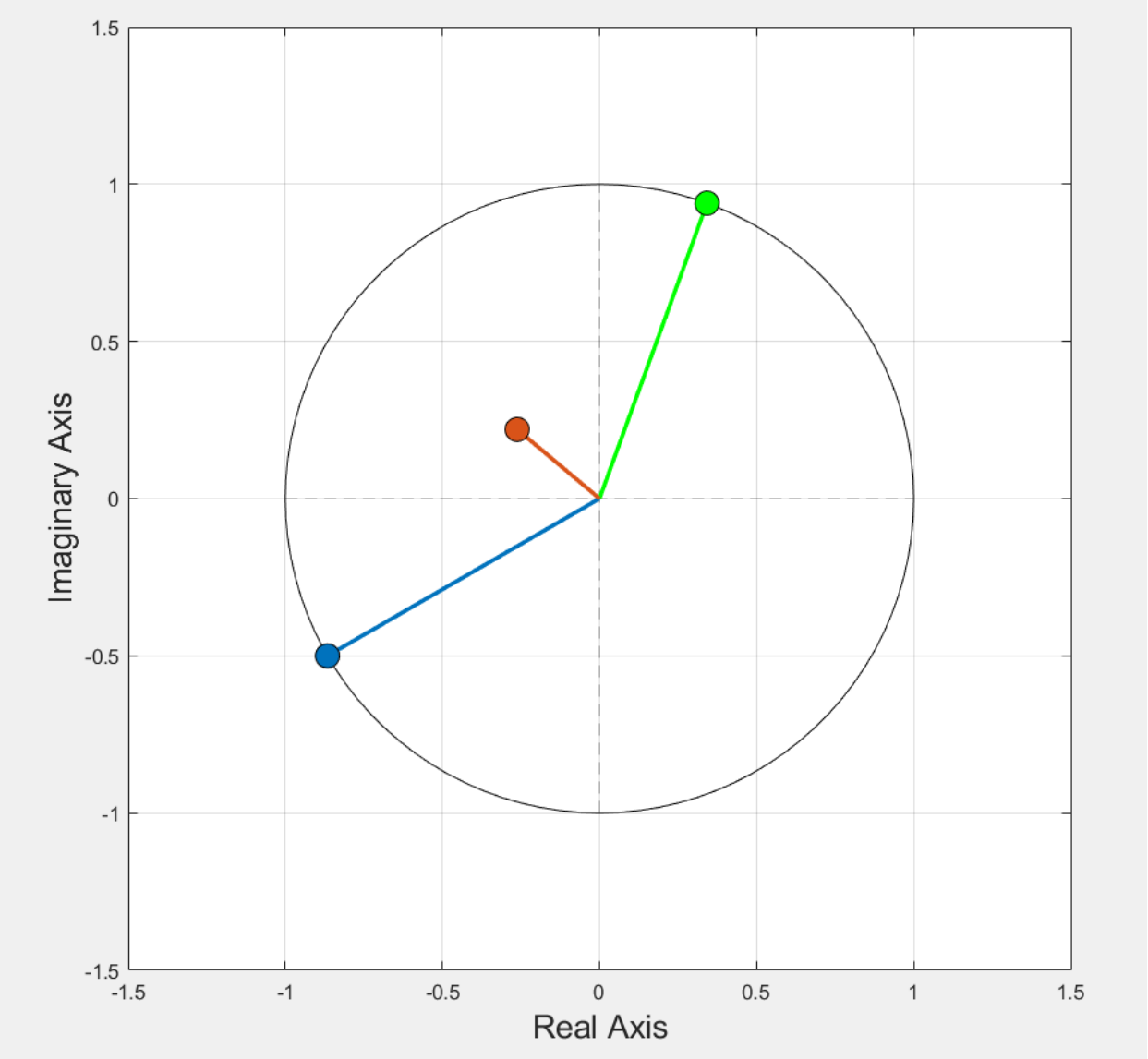


Figure: Averaging Fourier coefficients is like averaging two vectors. The average is not just the average of the lengths of the blue and green lines, but the average of the two vectors producing the orange resulting vector.

Two ways to average Fourier transforms:

- Average magnitudes
- Average the complex transforms as vectors

MATLAB

Code: Fourier_DTFT.m

Code lines 434 to 482

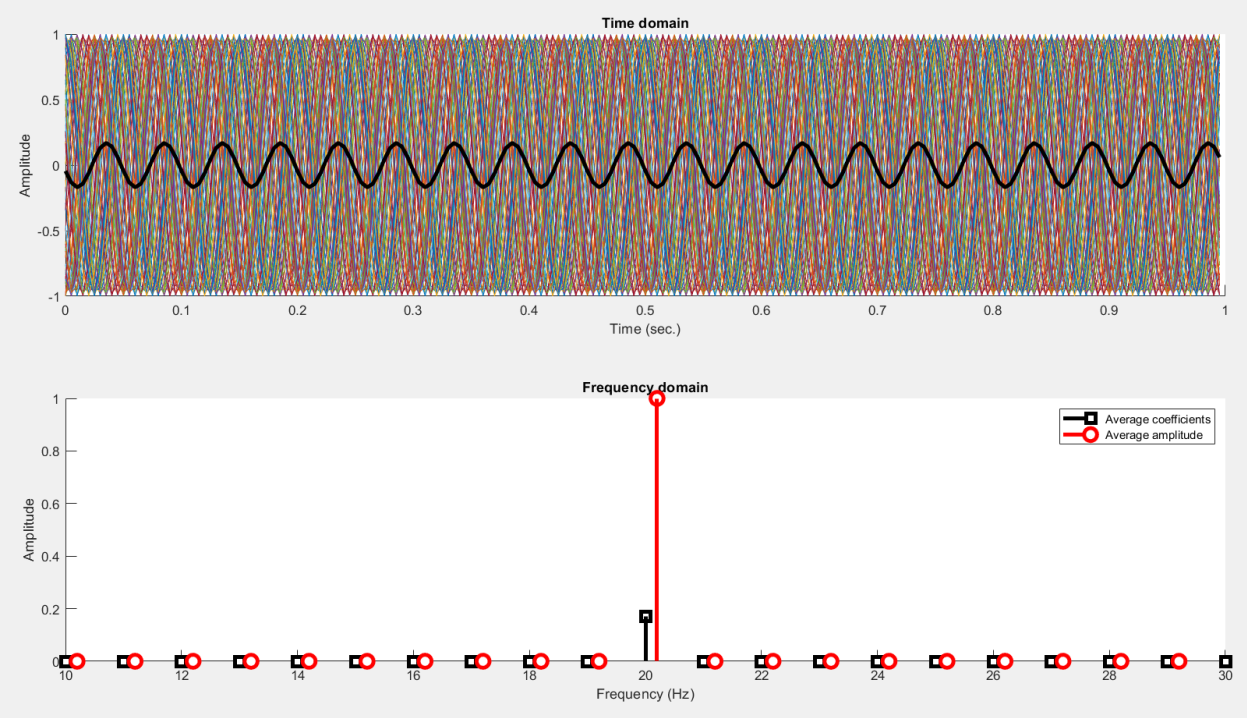


Figure above: all 100 20 Hz sine waves each with a different random phase. The black wave is the average.

Figure below: In black, take the mean after an FFT, then take the magnitude (2*abs) – the average of the coefficients. In red, take the magnitude of each coefficient, then take the mean – the average of the magnitudes. Note: the slight shift between red and black is just to visualize. When phase difference is near zero, the red and black lines are almost identical.

22. THE DC (ZERO FREQUENCY) COMPONENT

The correct way to normalize the convolution coefficients is to multiply by two for all values from zero to Nyquist, but not including zero itself.

The DC component is the average of all the datapoints.

The DC component does not affect the results of other points in the frequency domain. Both the frequency and the amplitude will read correctly even with a DC component showing at 0 Hz.

MATLAB

Code: Fourier_DTFT.m

Lines of code: 483 to 527

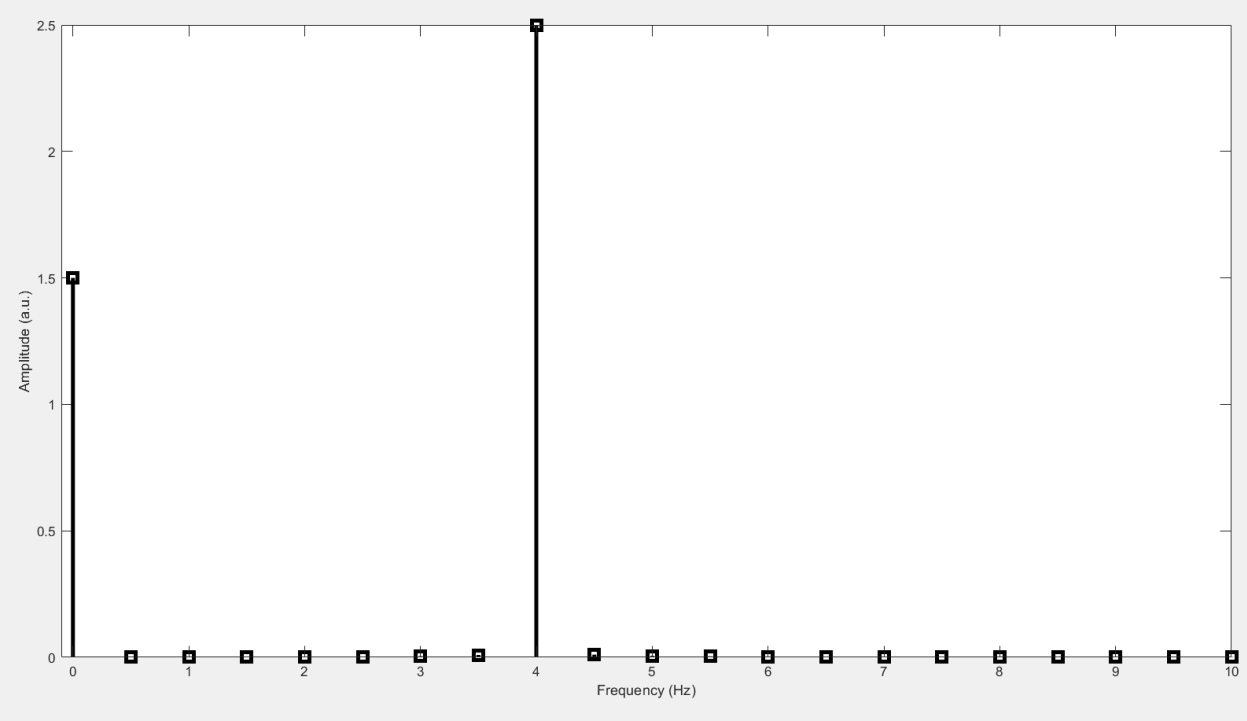


Figure shows DC amplitude at 0, and the frequency/amplitude pair at 4 Hz.

```
% extract amplitudes with normalizing point 1 (the zero value)
ampls = abs(fCoefs/pnts);
ampls(2:length(hz)) = 2*ampls(2:length(hz));
```

23. AMPLITUDE SPECTRUM VS. POWER SPECTRUM

Amplitude vs. power:

$$power = (amplitude)^2$$

$$amplitude = +\sqrt{power}$$

Power spectrum gets magnified when amplitude is greater than 1

Power spectrum gets a bit larger when amplitude is only a bit larger than 1

Power spectrum gets smaller than amplitude when amplitude is less than 1

The power spectrum often looks “cleaner” since small values get even smaller, and large values get larger.

Parseval’s theorem: the sum of all the power values squared equals the sum of all the time points squared. There is a conservation of energy in the time and frequency domains. The total energy in the time domain is the same as the total energy in the power spectrum.

Parseval’s theorem:

$$\sum_{t=1}^{n} |x_t|^2 = \frac{1}{n} \sum_{f=1}^{n} |X_f|^2$$

```
% MATLAB version of Parseval’s theorem
sum( x.^2 ) == sum( abs(fft(x).^2) / length(x)
```

MATLAB

Code: Fourier_DTFT.m

Lines of code: 529 to 592

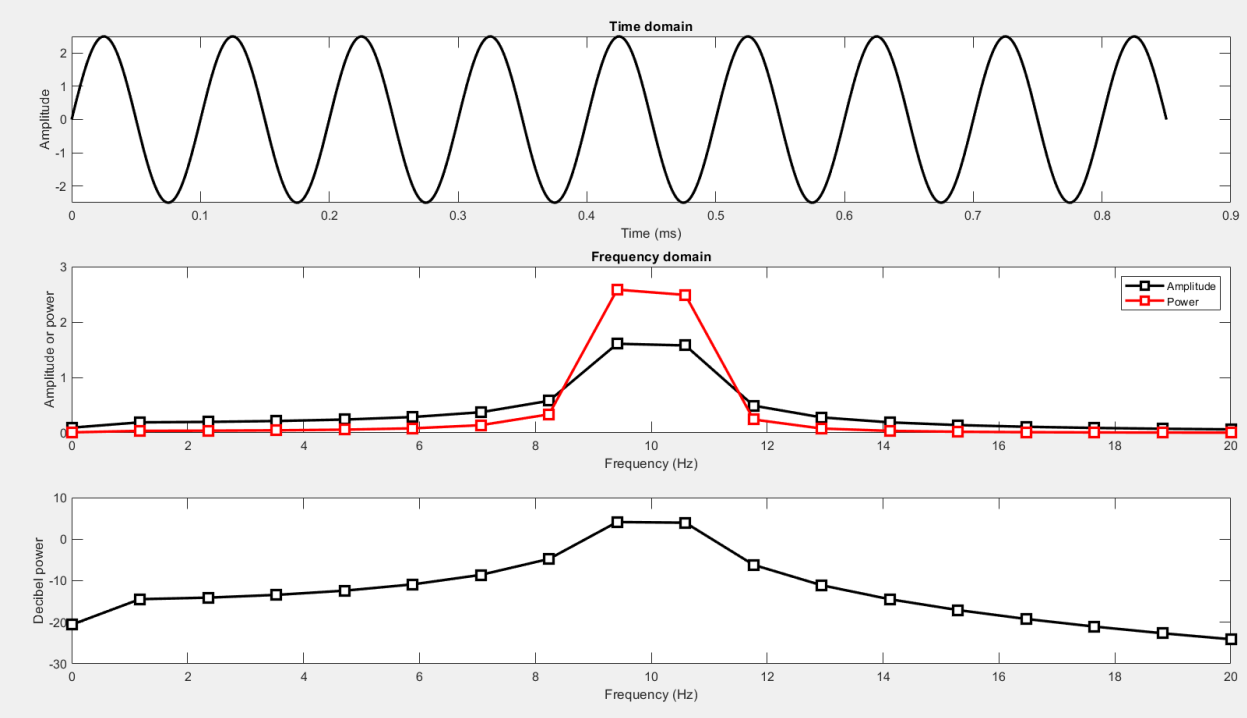


Figure top: 10 Hz signal.

Figure middle: Amplitude spectrum and Power spectrum.

Figure bottom: Decibel power spectrum.

Note: Time domain frequency is 10 Hz. But 10 Hz is not an available bucket in frequency domain so there is signal on either size of 10 in the frequency domain.

```
% extract correctly-normalized amplitude
signalAmp = abs(signalX(1:length(hz))/npnts);
signalAmp(2:end) = 2*signalAmp(2:end);

% and power
signalPow = signalAmp.^2;

% dB Power
signaldB = 10*log10(signalPow)
```

Code fragment showing how to convert from Fourier coefficients to amplitude, then power, then decibels

24. A NOTE ABOUT TERMINOLOGY OF FOURIER FEATURES

Article Title: Five methodological challenges in cognitive electrophysiology

Journal Title: NeuroImage (Elsevier), Volume 85, Part 2, 15 January 2014, Pages 702-710

Abstract

Here we discuss five methodological challenges facing the current cognitive electrophysiology literature that address the roles of brain oscillations in cognition. The challenges focus on (1) unambiguous and consistent terminology, (2) neurophysiologically meaningful interpretations of results, (3) evaluation and comparison of different spatial filters often used in M/EEG research, (4) the role of multiscale interactions in brain and cognitive function, and (5) development of biophysically plausible cognitive models. We also suggest research directions that will help address these challenges. We hope that this paper will help foster discussions and debates about important themes in the study of how the brain's rhythmic patterns of spatiotemporal electrophysiological activity support cognition.

See PDF saved in this section

Amplitude vs Magnitude

Amplitude per time point could be positive or negative value with reference to a common point vs Amplitude per sine wave in frequency domain (distance from origin in complex plane) aka complex dot product Non-negative value

SECTION 4: THE DISCRETE INVERSE FOURIER TRANSFORM

26. HOW AND WHY IT WORKS

- Loop over frequencies corresponding to the number of time points in the signal:
    - Create complex sine wave with the same number of time points as the signal and a frequency defined by point -1.
    - Multiply the sine wave by the complex Fourier coefficient.
    - Sum the modulated complex sine waves together.
- Divide the result by N (because the loop involved summing over N)

Inverse Fourier transforms are useful in algorithms for signal processing where calculations are faster in the frequency domain. Examples include:

- Convolution
- Filtering
- Cross-correlation

MATLAB

Code: Fourier_inverse.m

Code lines 1 to 134

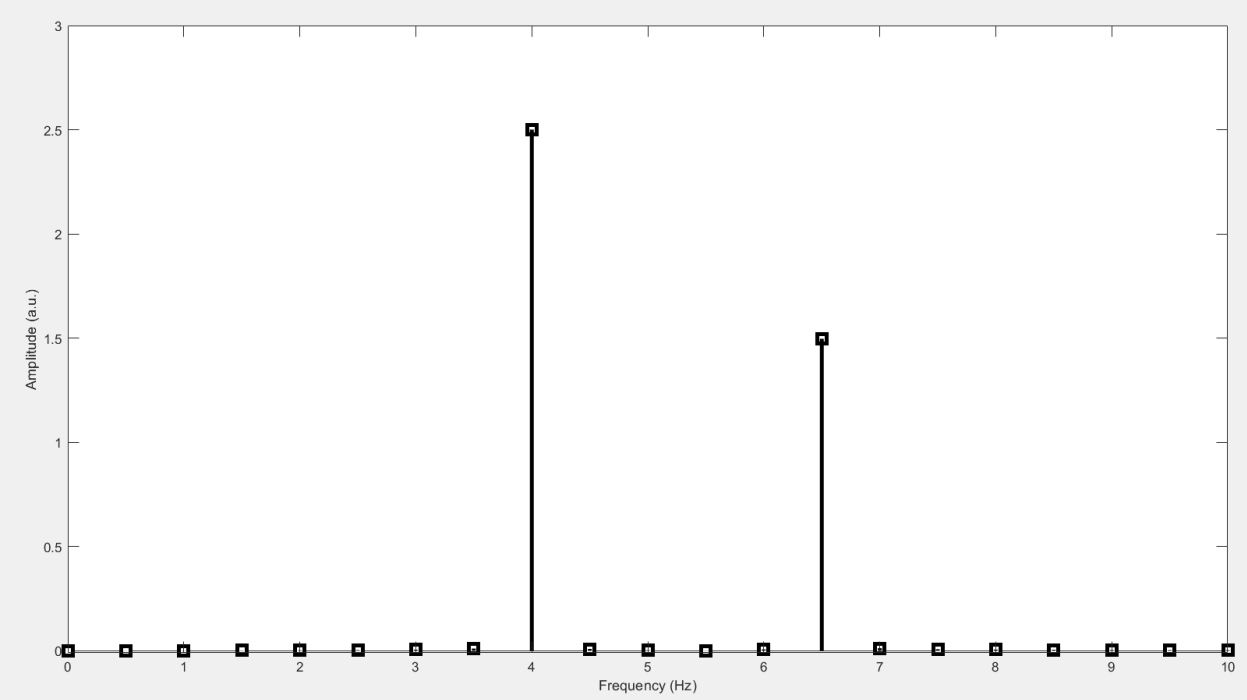


Figure: Fourier transform of a signal with two sinusoidal components at 4 and 6.5 Hz.

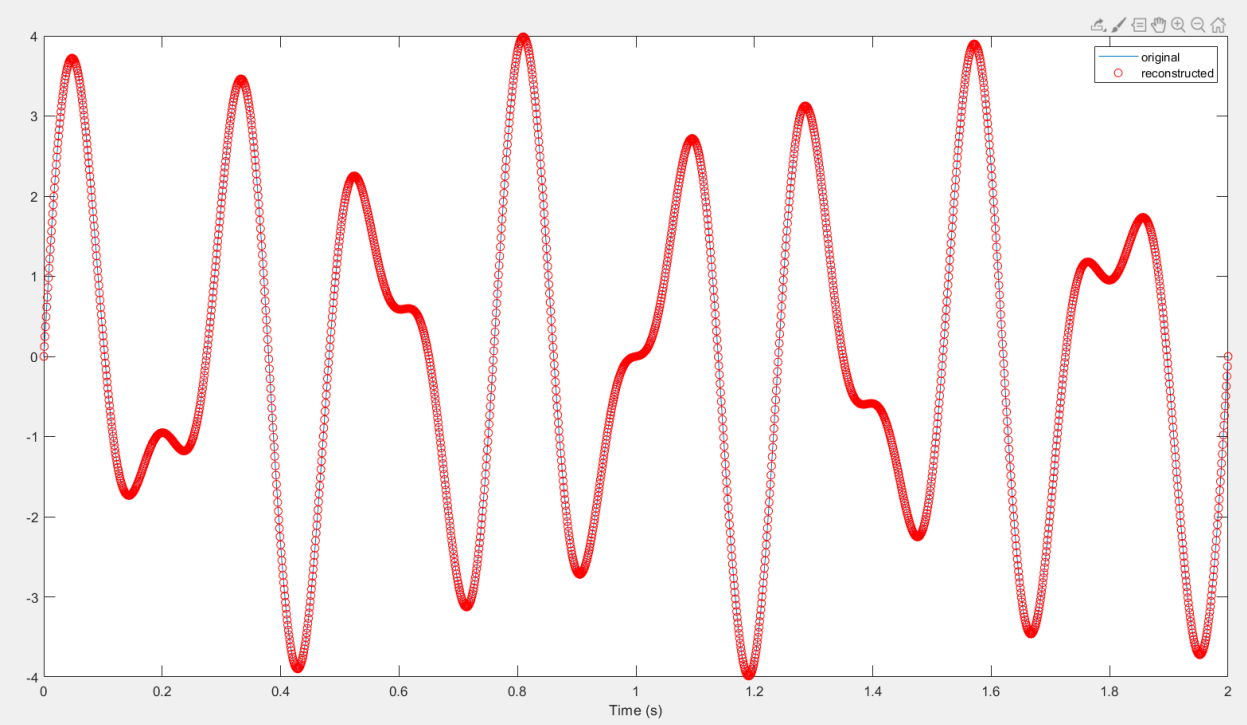
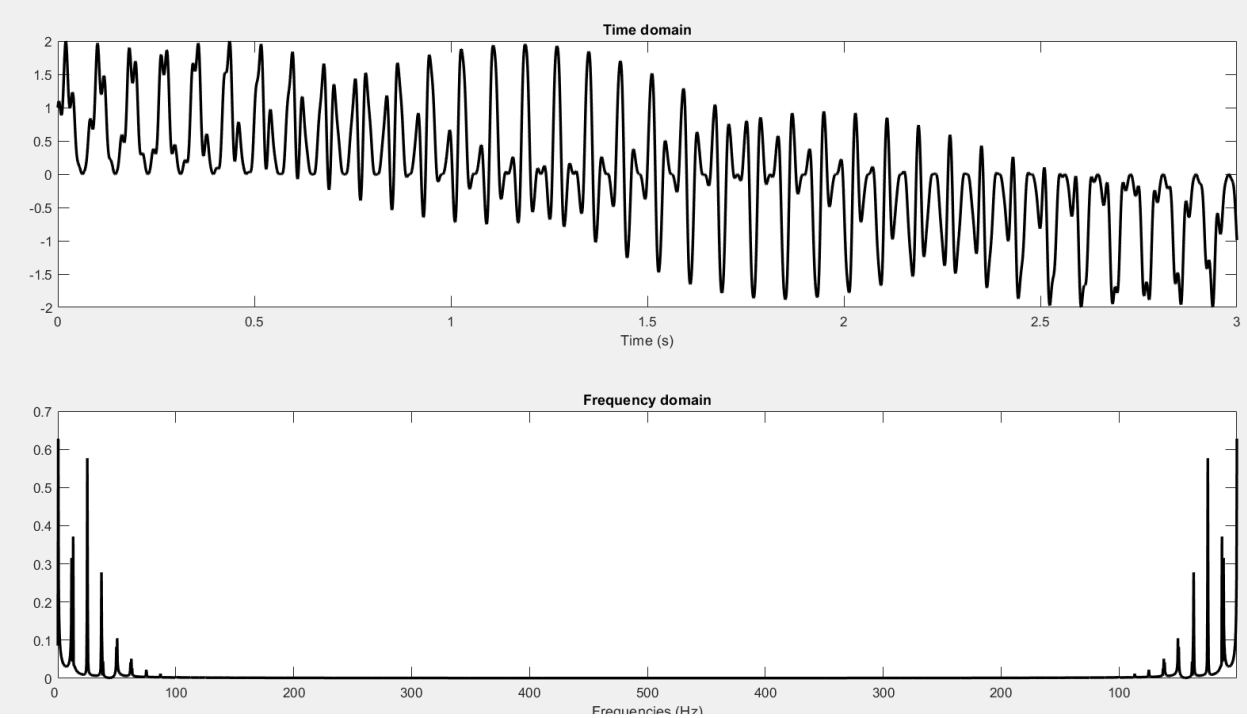


Figure: Inverse Fourier transform overlaid on the original signal showing a perfect match.



Simulation showing the reconstruction of original signal. Image here shows the end point.

27. INVERSE FOURIER TRANSFORM FOR BANDSTOP FILTERING

Procedure:

- Compute the Fourier transform of a signal.
- Change the frequencies or phases in the frequency domain (both positive and negative frequencies).
- Use the inverse Fourier transform to get back to the time domain.
- This procedure only works when there are clearly separable spectral components in the signal.

MATLAB

Code: Fourier_inverse.m

Lines of code from 135 to 208

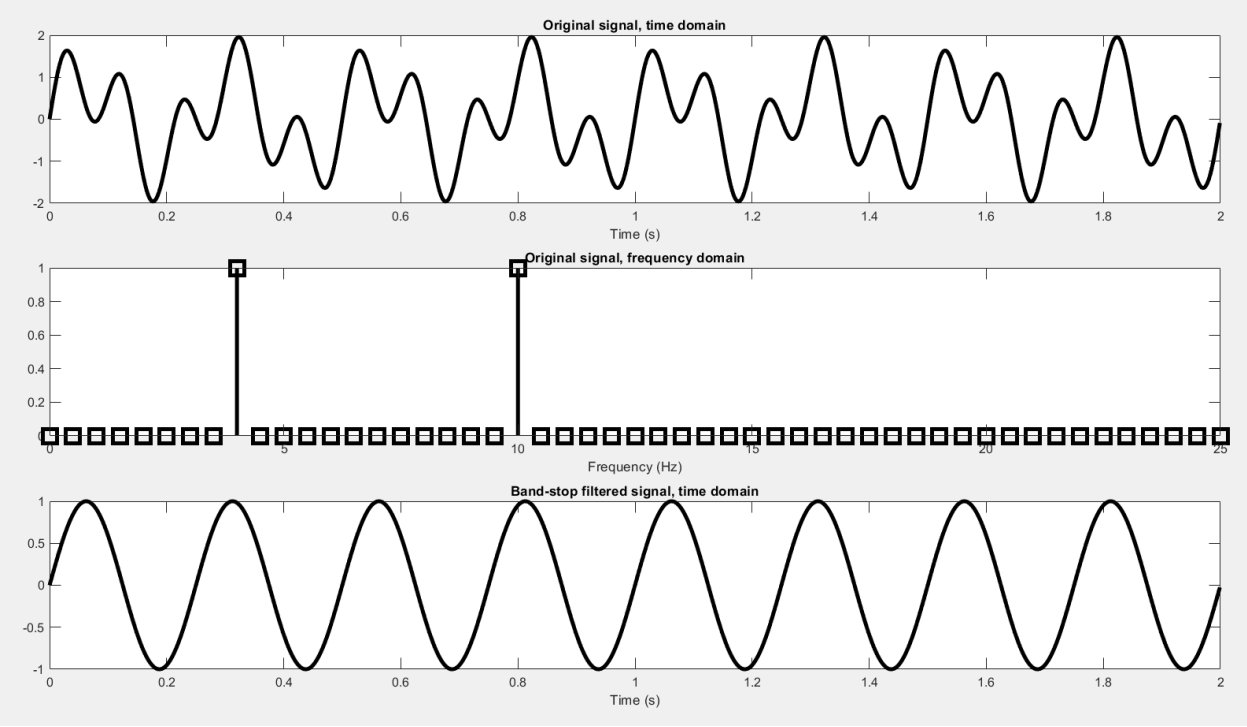


Figure top: original signal.

Figure middle: original signal frequency domain.

Figure bottom: final signal after 10 Hz coefficient was removed from both positive and negative frequencies in frequency domain.

SECTION 5: THE FAST FOURIER TRANSFORM

29. HOW IT WORKS, SPEED TESTS

Two steps:

- Step 1
  - Put all of the complex sine waves into a matrix instead of using a loop.
  - Put the signal into a column vector.
  - Now implement the Fourier transform as a matrix multiplication.
- Step 2
  - Use a series of sparse matrices to get final solution (Required: Matrix decomposition techniques).

Use the FFT as it produces the same results as a DFT loop, but significantly faster.

MATLAB

Code: Fourier_FFT.m

Lines of code from 1 to 64

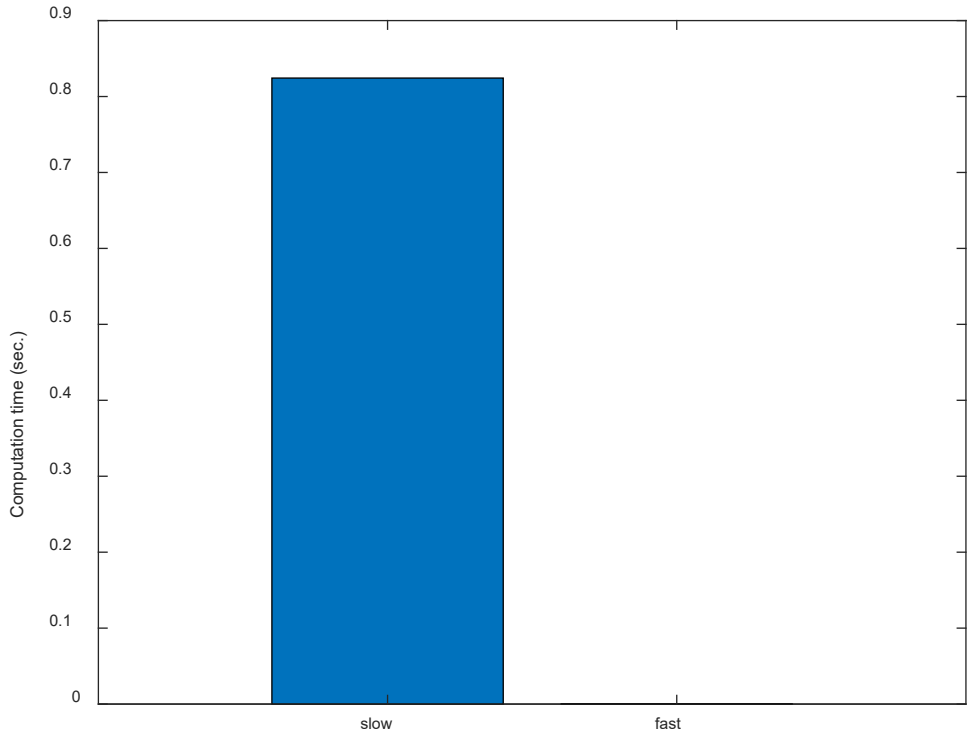


Figure shows tic/toc times for DFT loop (slow) and FFT (fast). FFT was so much faster that it is not seen.

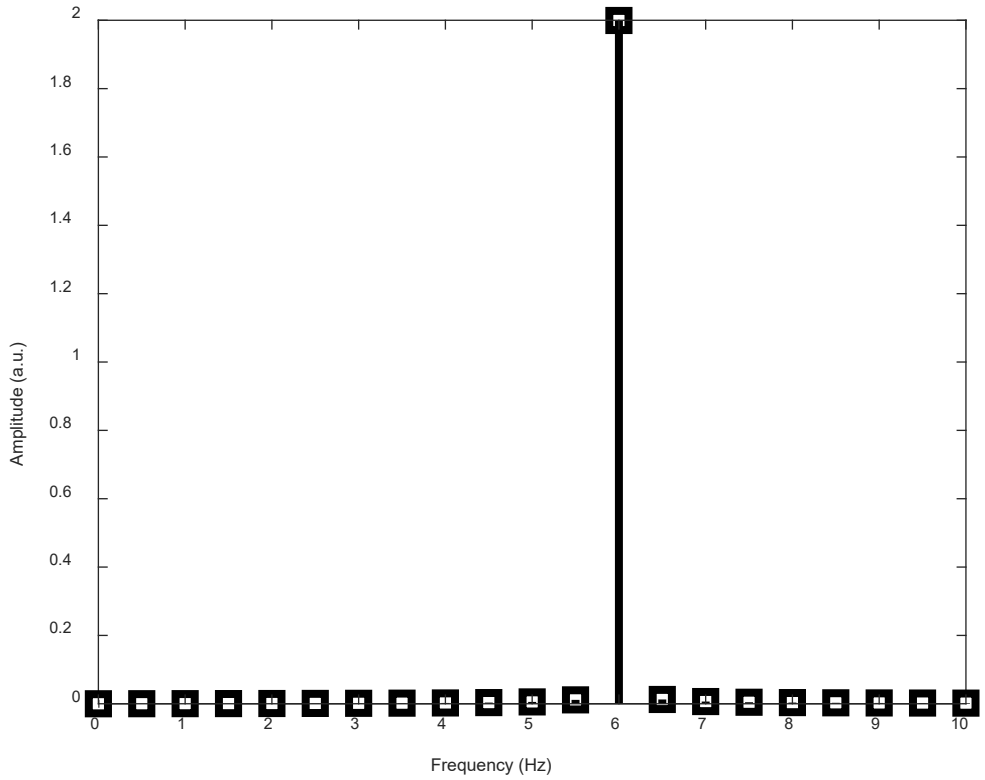


Figure demonstrates that FFT works

30. THE FAST INVERSE FOURIER TRANSFORM

MATLAB

Code: Fourier_FFT.m

Lines 65 to 97

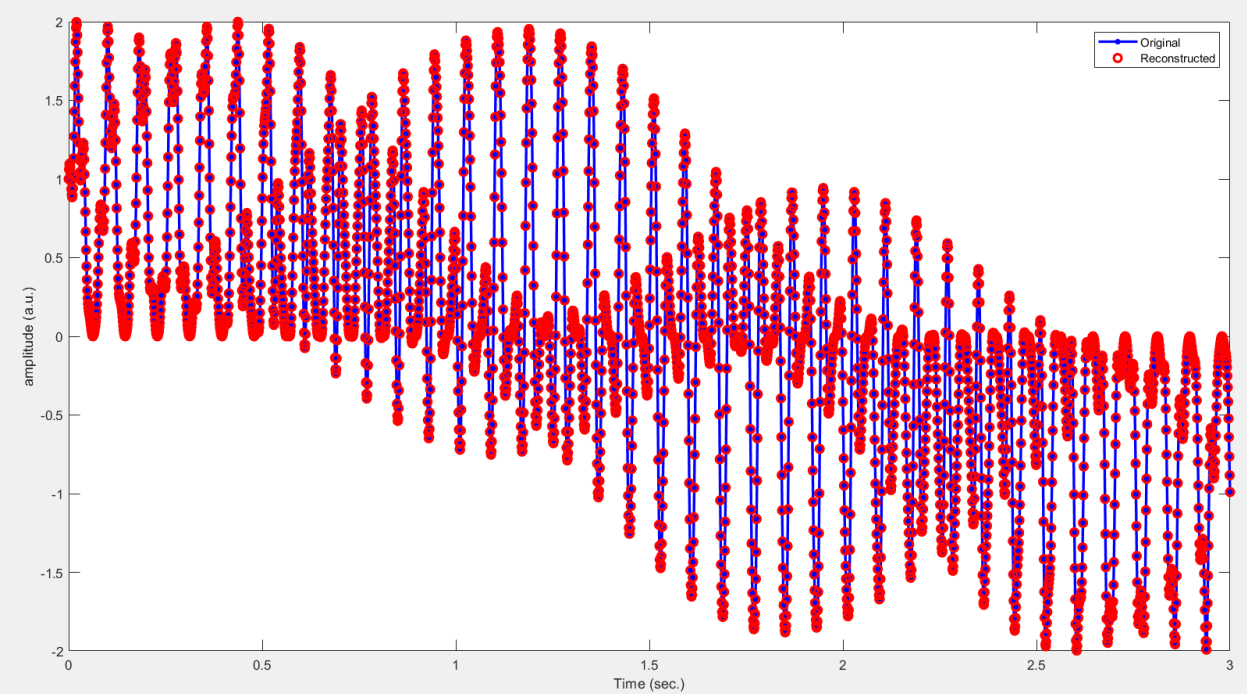


Figure verifies that the ifft(fft(signal)) produces the same original signal.

31. THE PERFECTION OF THE FOURIER TRANSFORM

No information is lost when using the Fourier transform. Two explanations:

1. Multiple regression (statistics)
2. Vector full-rank matrix multiplication (linear algebra)

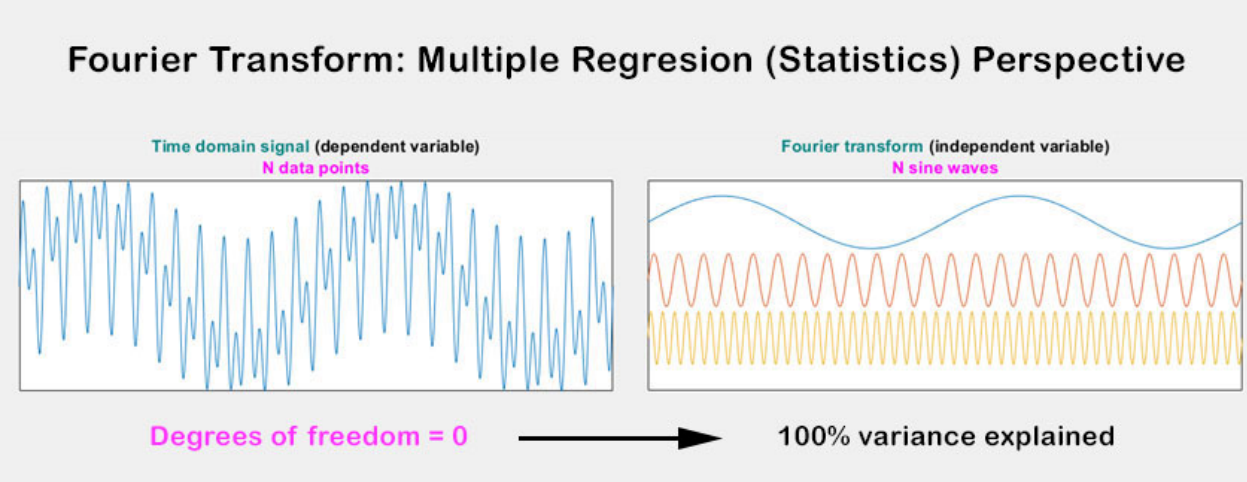


Figure: demonstration of multiple regression explanation. The sum of the sine waves on the right produce the sinusoid on the left. Note: phase is not shown in this diagram.

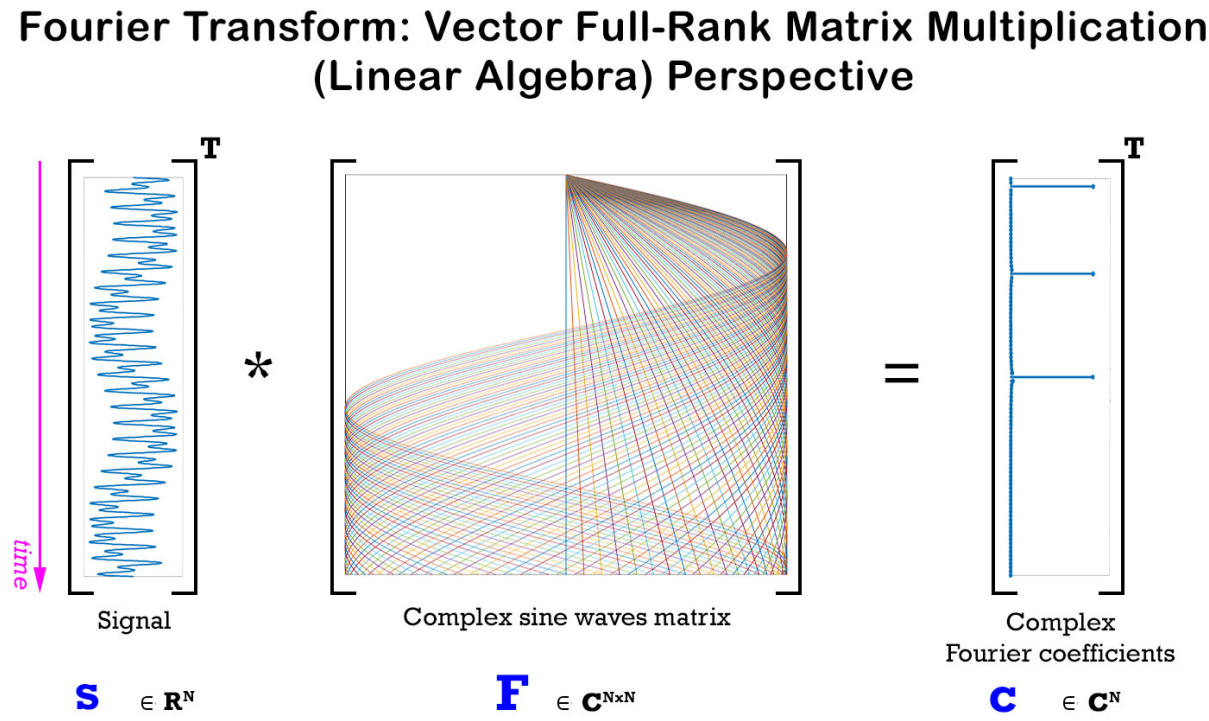


Figure: demonstration of linear algebra explanation. Use a square full-rank matrix of all complex sine waves to directly convert signal to Fourier coefficients. Note: phase is not shown in this diagram.

Fourier Transform Linear Algebra Discussion	
$\mathbf{s\,F = c}$	Always a unique solution. s is a vector of signal, F is the Fourier matrix, c is the vector of coefficients.
$\mathbf{s = c\,F^{-1}}$	Reconstruct original signal with inverse Fourier transform. Shows why there is a sign flip of the imaginary complex sine wave in the negative frequencies.
$\mathbf{F\,F^{-1} = I}$	I is the identity matrix. The imaginary parts of F must cancel with the imaginary parts of F^{-1}.

MATLAB

Code: Fourier_FFT.m

Lines of code: 98 to 139

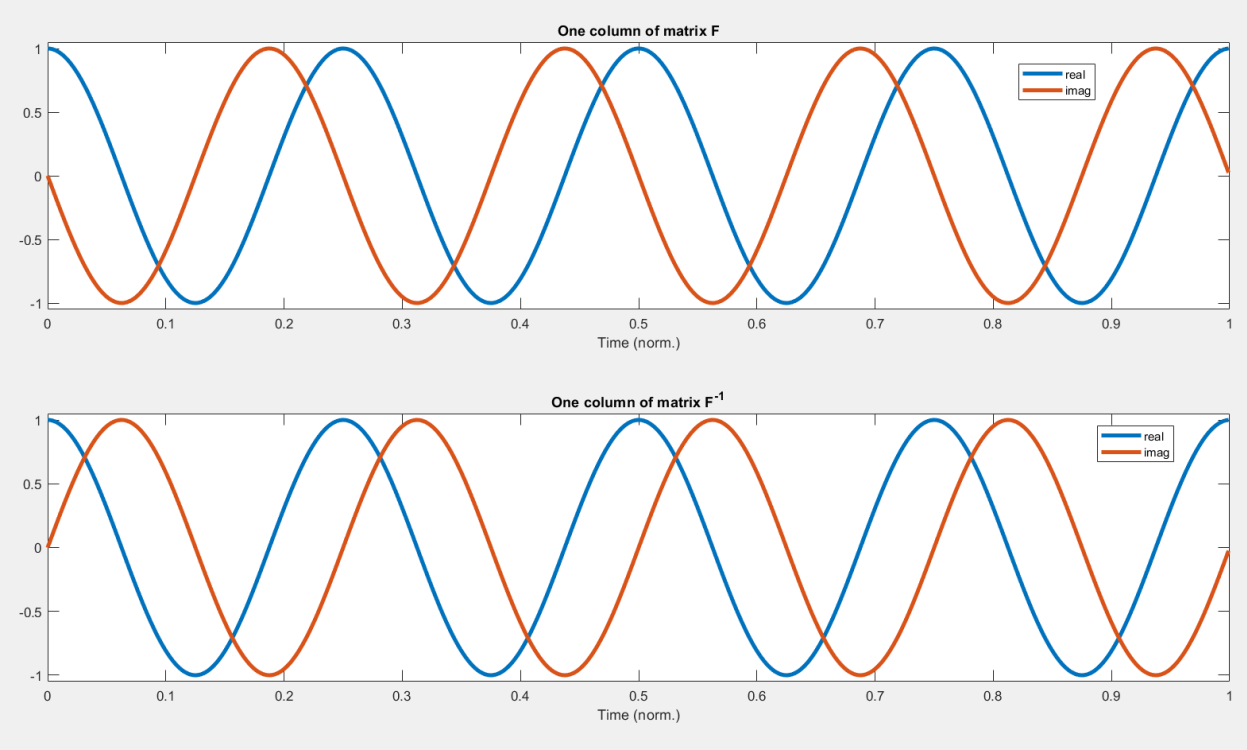
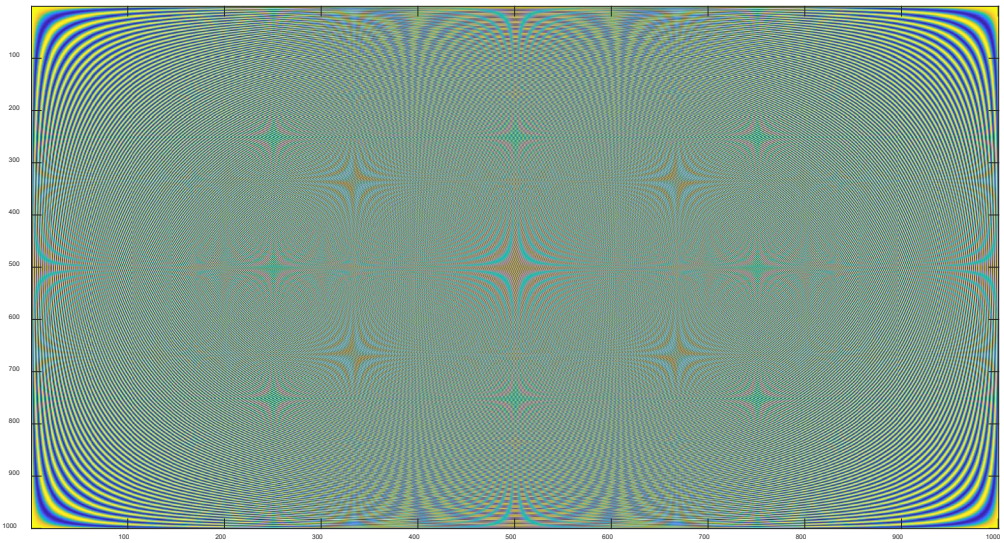


Figure demonstrates that the imaginary parts in both plots would cancel, leaving only the real signal. The original signal is a matrix create from complex sine waves.



Commands to produce above figure:

```
>> figure
```

```
>> imagesc(real(F))
```

32. USING THE FFT ON MATRICES

Apply fft to a row or a column

Input the entire matrix, instead of looping over channels

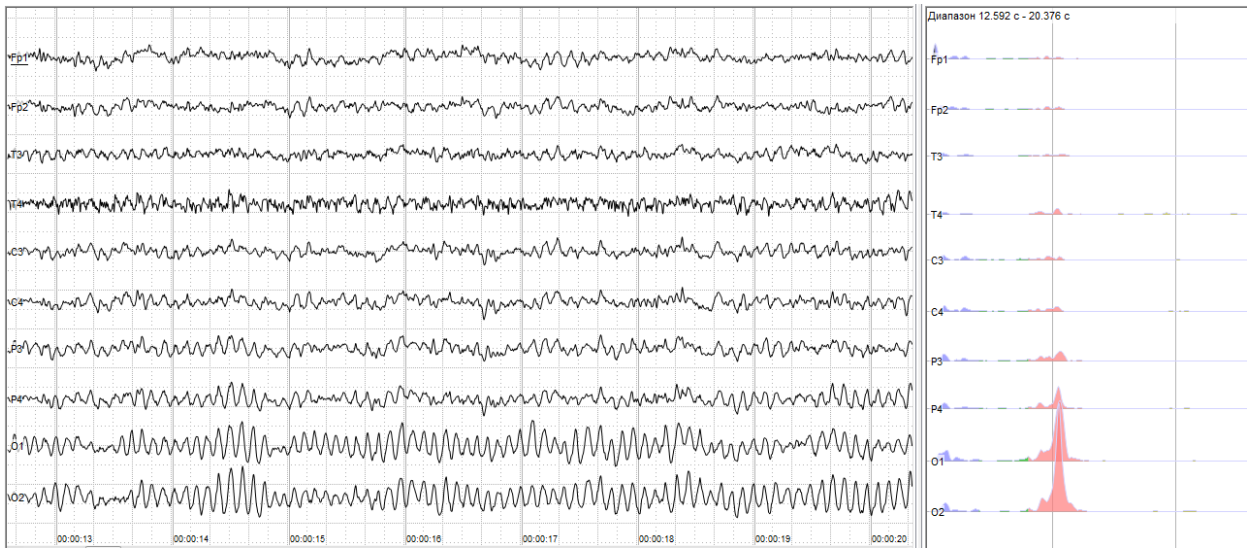


Figure: Show EEG signals with fft results per row. (Andrii Cherninskyi / CC BY-SA (https://creativecommons.org/licenses/by-sa/4.0))

MATLAB

Code: Fourier_FFT.m

Lines of code from 140 to 183

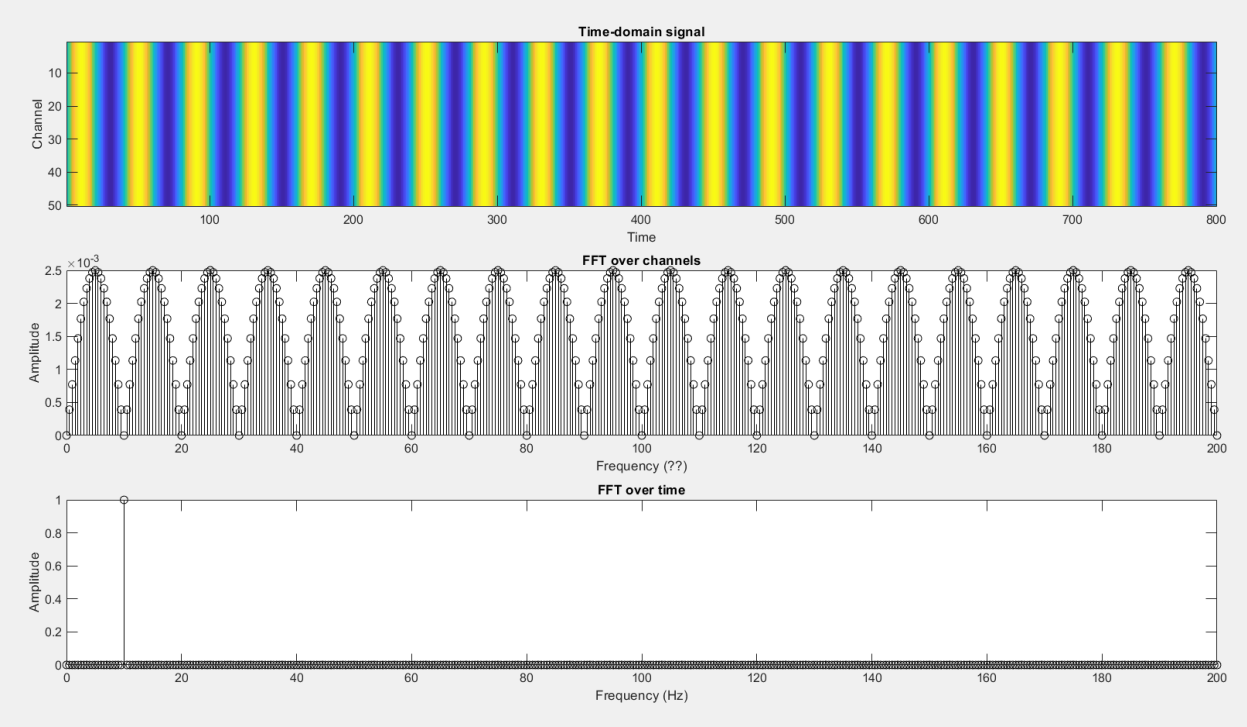


Figure: Demonstrate a matrix fft. One problem is determining which channel (row vs column) to use for the fft. In this case the middle graph shows a row fft which is nearly zero everywhere. While the bottom graph shows the correct dimension to plot (where columns represent time) with a 10 Hz peak.

SECTION 6: FREQUENCY RESOLUTION AND ZERO PADDING

34. SAMPLING AND FREQUENCY RESOLUTION

Sample rate:

- Sample at rate higher than twice the frequency of interest.
- Frequency (spectral) resolution is the distance between two successive frequency points:
- Frequency resolution = Sampling rate / number of points in signal.
- Analog (continuous) signal Vs. Measured (digital) signal. If sampling rate is too low, then important aspects of signal are lost.
- Use stem or bar plots to display fft results. Line plots indicate that there is more information than is actually present.
- Use a line plot to show spectra from multiple sources.
- **It is not possible to measure any signal faster than the Nyquist frequency – which is half the data sampling rate.**
- Any information in the analog continuous signal that is faster than the Nyquist frequency is not only lost, but also aliased into what looks like lower frequency information (artifacts).

- Use a sampling frequency higher than the Nyquist frequency to increase signal to noise and higher quality estimates of the dynamics at higher frequencies that approach Nyquist.

Discrete temporal sampling – frequency (spectral) resolution:

- Frequency resolution is the distance between two successive frequency points.
- **Frequency resolution = sample rate / number of points in signal (srate/N).**
- Example:
    - Sampling rate = 1 kHz
    - Signal length = 2000 (2 seconds)
    - Frequency resolution = 1000/2000 = ½ Hz

Nyquist frequency is ½ the Sampling rate.

MATLAB

Code: Fourier_resolution0.m

Lines of code from 1 to 58

```
% compute frequencies vector
hz = linspace(0,srate/2,floor(pnts/2)+1);

% compute empirical frequency resolution as spacing between
frequencies
freqres = mean(diff(hz));

% ... or directly from sampling rate and N
freqres = srate/pnts;
```

Figure showing how to calculate frequency resolution: Lines of code: 1 to 31

Frequency resolution is 1 Hz

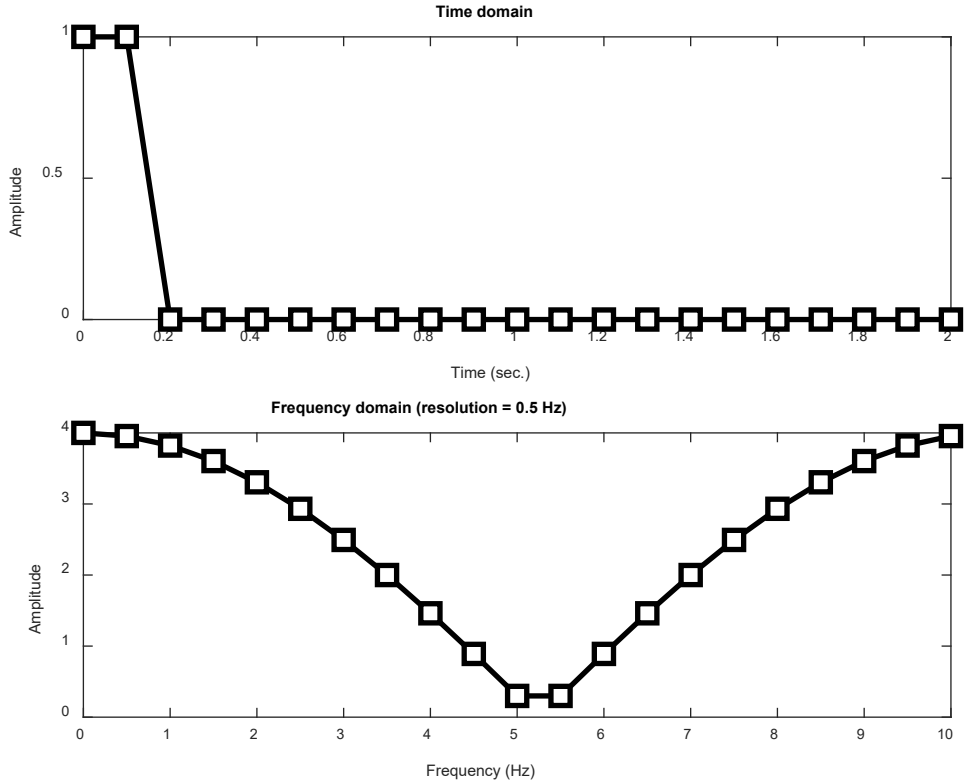


Figure showing sampling rate and frequency resolution: Lines of code from 32 to 58. Upper plot shows time domain of signal a short pulse. Lower plot shows frequency domain for the same pulse valid only from 0 to 5 Hz. The effect is based on a sample rate of 10 Hz.

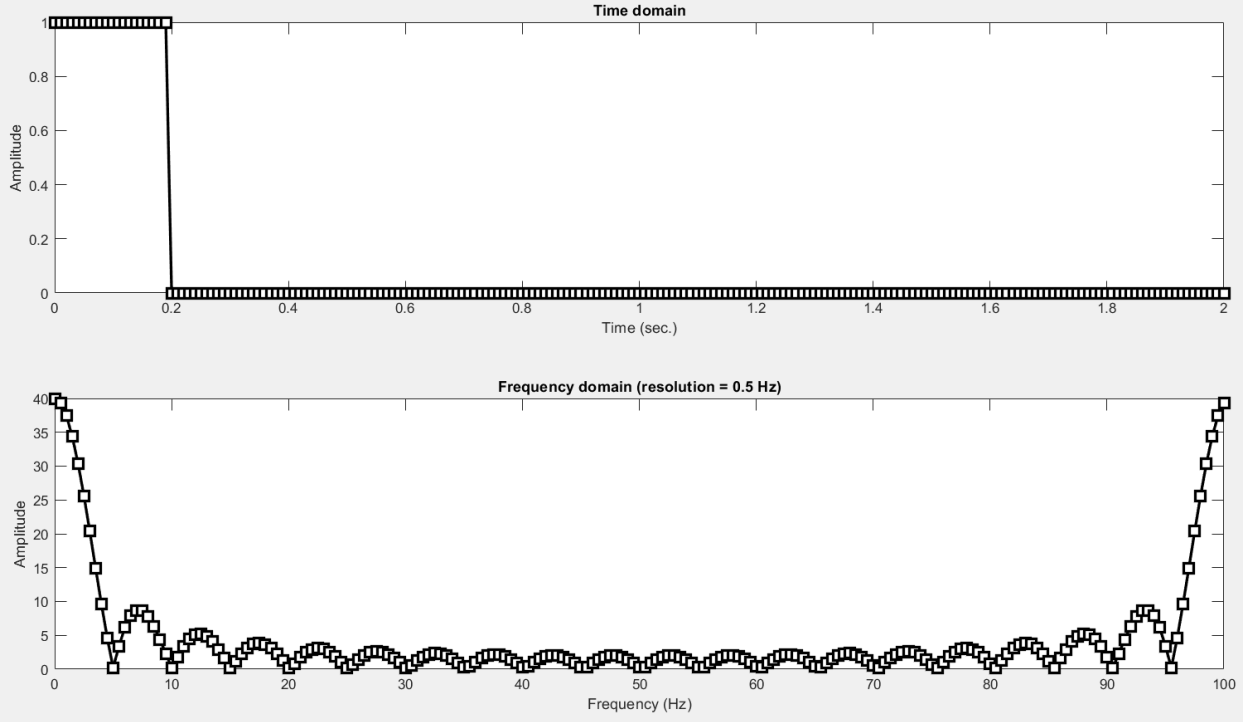


Figure shows same information as plot above, except sample rate was increased to 100 Hz.

35 TIME-DOMAIN ZERO PADDING

Add zeros to end of signal in time domain to increase frequency resolution: Sinc Interpolation.

Normalize amplitude after fft by the length of the original signal, not the signal + zero pad.

Definitions:

- **Resolution** is the fineness to which an instrument can be read.
- **Precision** is the fineness to which an instrument can be read *repeatably and reliably*.

MATLAB

Code: Fourier_resolution0.m

Lines of code: 59 to 161

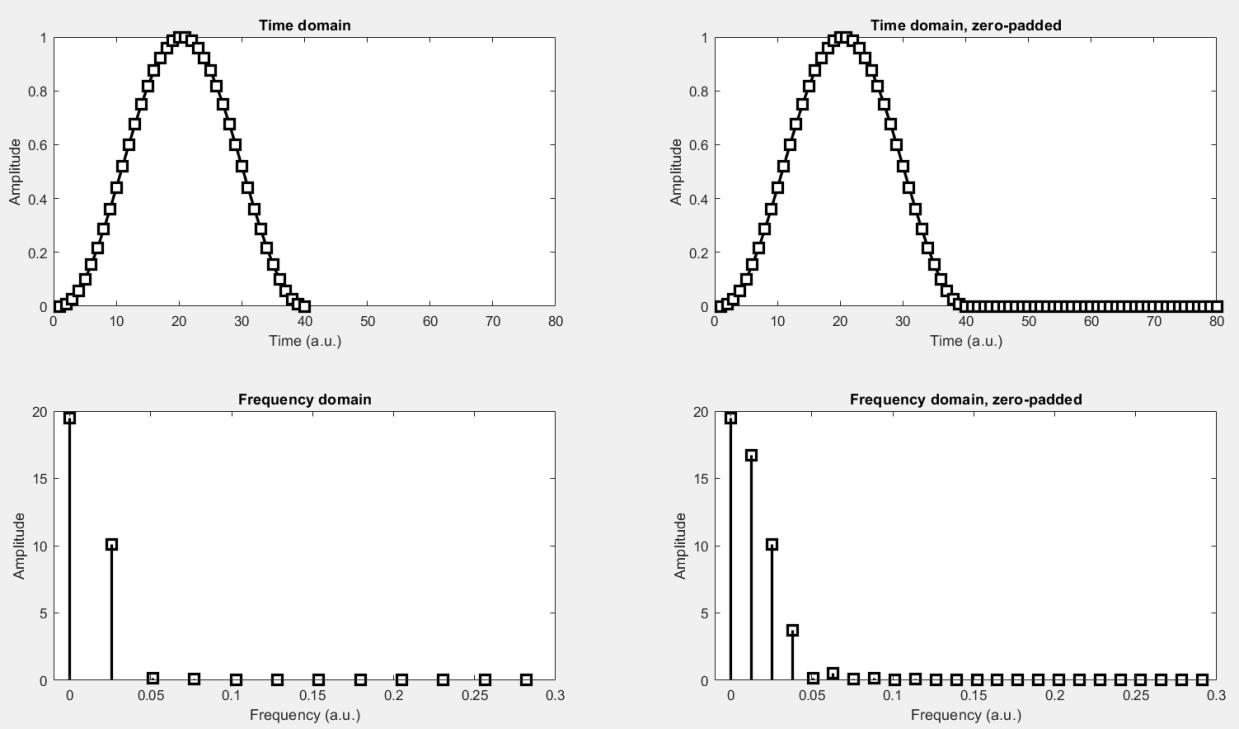


Figure: Upper left is ½ of a sine wave squared, called a “Hann (Hanning) window or “Hann taper”,

Figure: Lower left is the frequency response for the Hann window.

Figure: Upper right is the Hann window with 40 zeros added (concatenated) to the right of the signal,

Figure: Lower right is the frequency response for the Hann window with zero padding. The added frequencies are an interpolation of frequencies shown in the original signal frequency response. aka sinc interpolation.

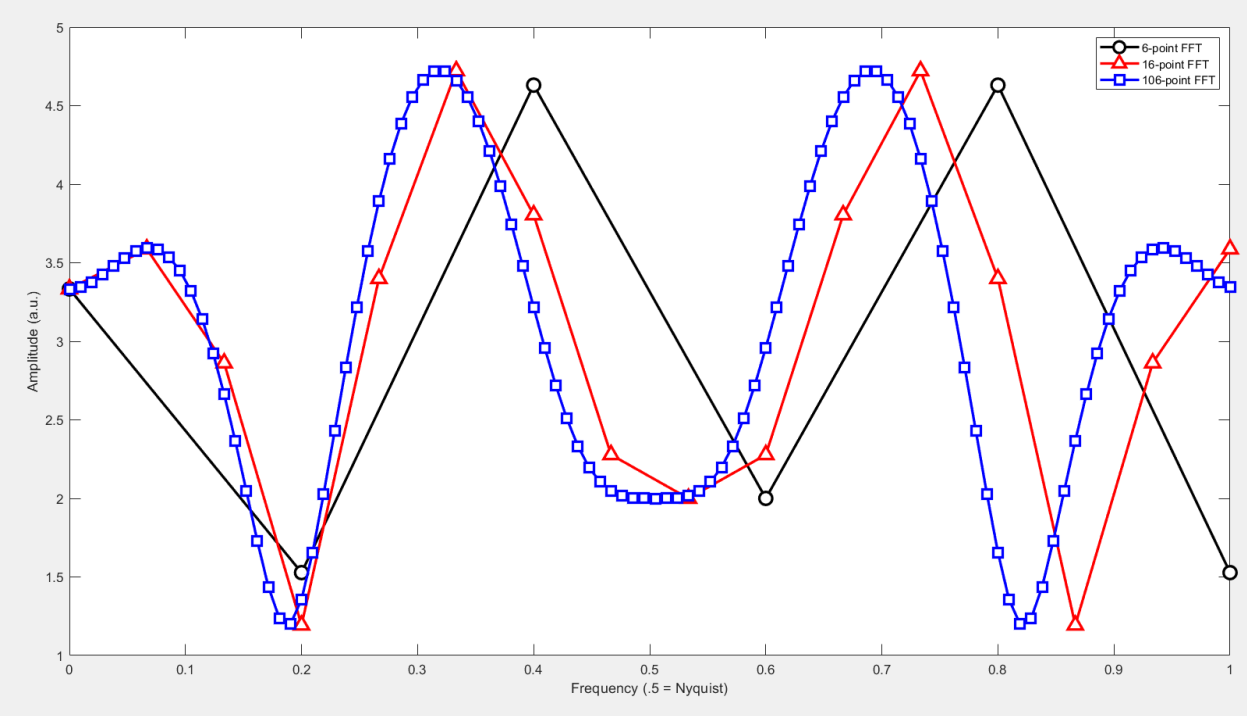


Figure: zero-padding increases frequency resolution. Original signal has 6 points (black color), while the smother plots have 10 (red) and 100 (blue) zeros added.

36. FREQUENCY-DOMAIN ZERO PADDING

Zero padding in frequency domain will increase temporal resolution after ifft, sinc interpolated signal.

Zero padding in the frequency domain corresponds to a sinc interpolated version of the original signal.

Zero padding in the frequency domain is a way to up-sample a signal in the time domain.

MATLAB

Code: Fourier_resolution0.m

Lines of code: 162 to 266

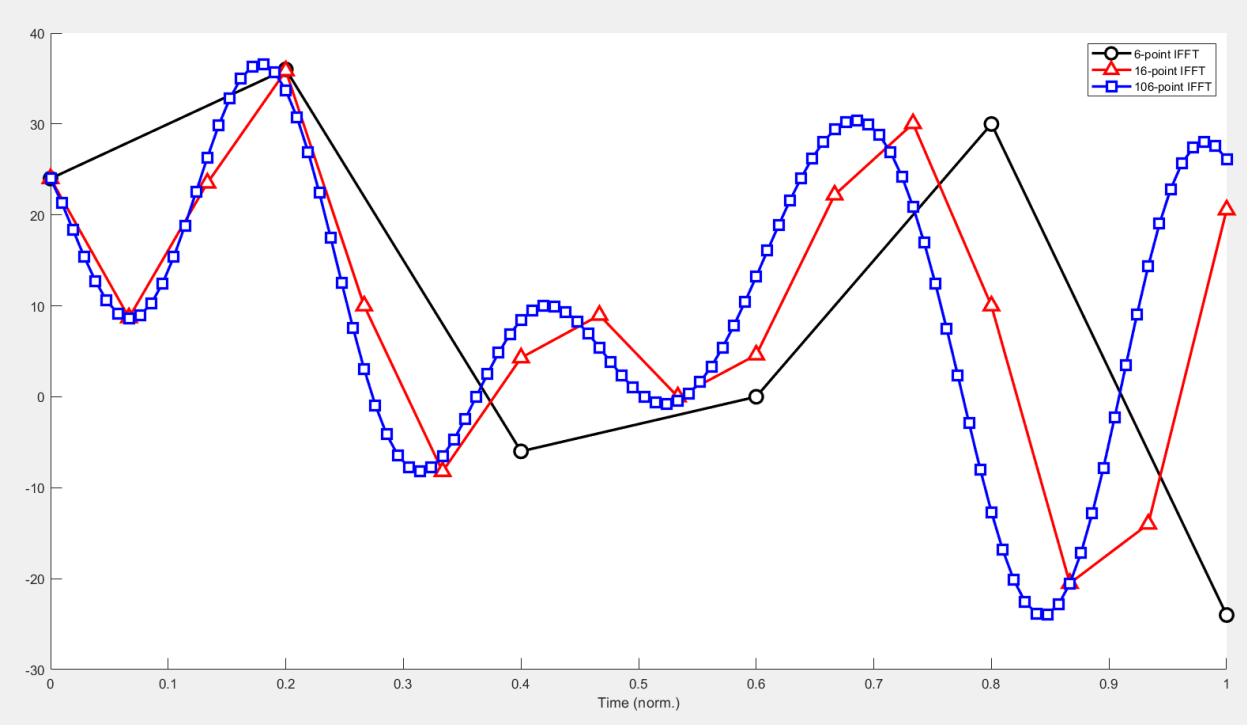


Figure: Effect of zero padding of IFFT to regenerate a sinc interpolated up-sampling of signal.

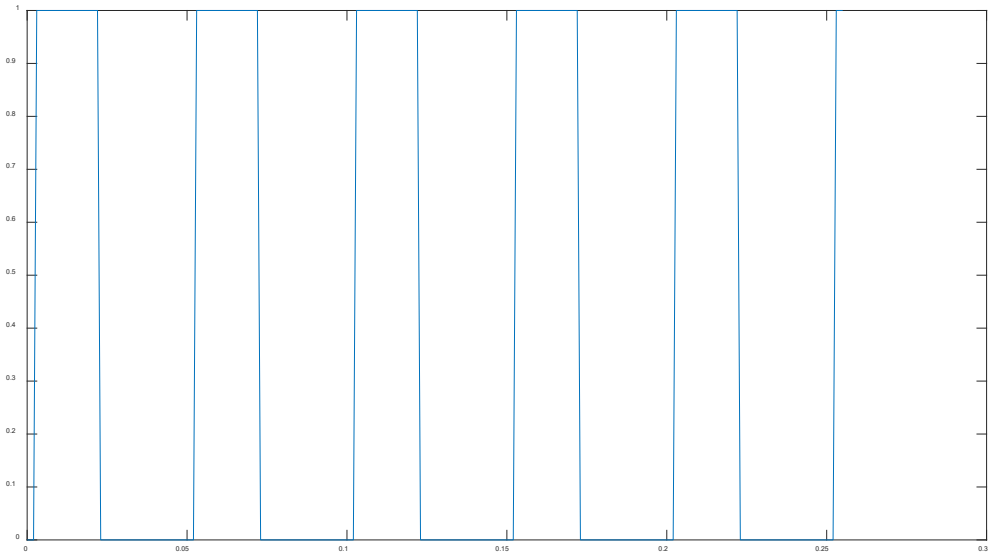


Figure: Create a square-like wave with edges and singularities

```
% create a square-like signal
srate  = 1000;
```

```
x      = (0:255)/srate;
signal = sin(2*pi*20*x)>.3;
```

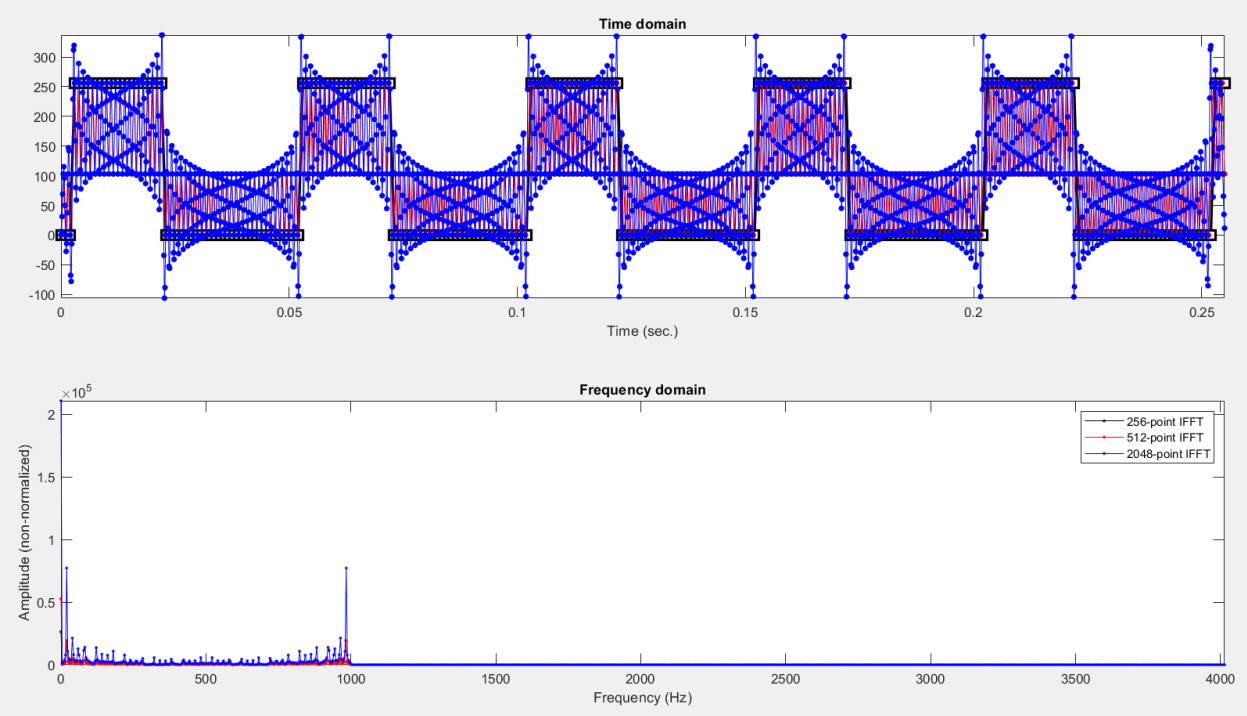


Figure: Showing the edge artifacts and singularities generated by sharp edges making the zero-padding interpolation less useful. Sharp edges have ringing.

37. SAMPLING RATE VS. SIGNAL LENGTH

Sampling rate vs. the signal length affects the time and frequency-domain representations of the signal.

Avoid the confusion between frequency resolution and temporal resolution.

Sampling rate is the number of sample points per second (or per unit of time) that you acquire from the measurement device.

The sampling rate defines the temporal resolution entirely.

In the frequency domain, the frequency resolution is not determined solely by the data sampling rate.

The frequency resolution is determined by a combination of the sampling rate and (crucially) the number of time points.

The length of the signal has nothing to do with the sampling rate.

Given a fixed sampling rate, the length of the signal determines the frequency resolution.

Because the zero point and the Nyquist point are always fixed, only the number of data points between this span will affect the frequency resolution.

MATLAB

Code: Fourier_resolution0.m

Lines of code: 267 to 327

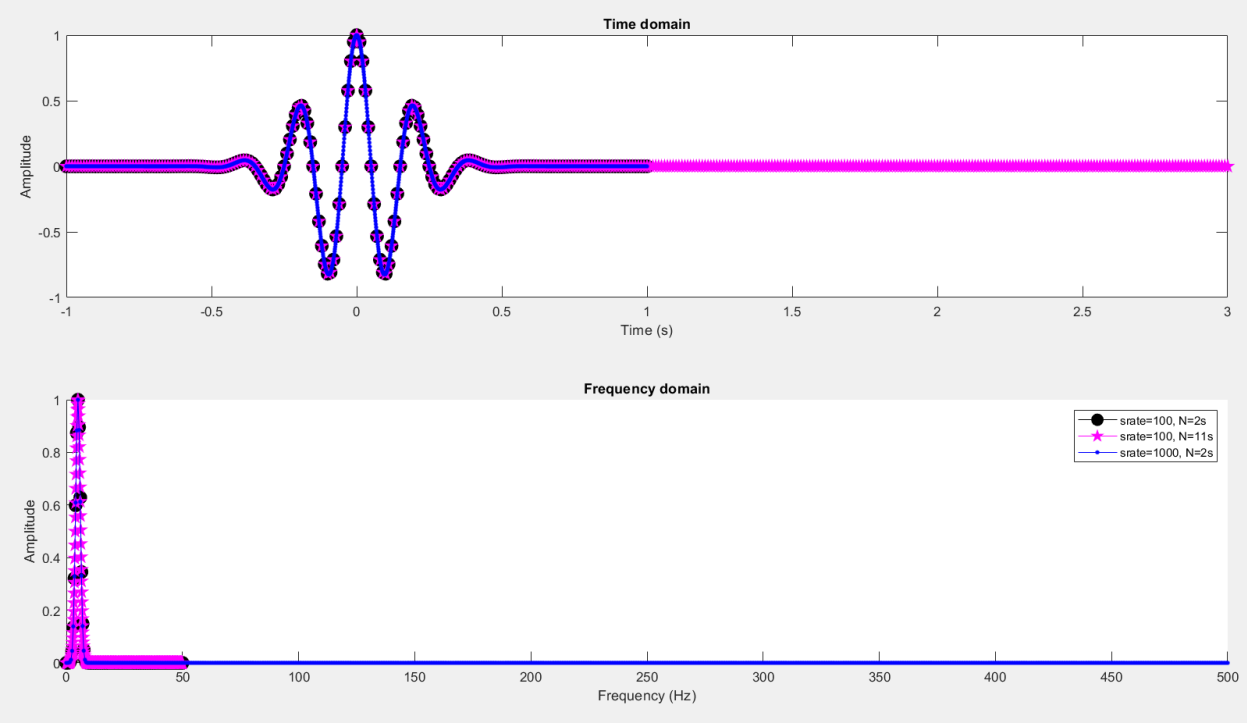


Figure: Upper shows 3 Morlet wavelet signals using two different sampling rates [100 100 1000]. The blue points has the higher sampling rate.

Figure: Lower show how number of samples and sampling rate affect the frequency domain. The blue dots go for a longer frequency since the Nyquist frequency is 500 instead of only 50 Hz. Yet the magenta stars show the highest spectral resolution because the number of samples are larger (11 seconds vs 2 seconds).

38. COURSE TANGENT: SELF-ACCOUNTABILITY IN ONLINE LEARNING

Motivation, quality of learning, accountability.

SECTION 7: ALIASING, STATIONARITY, AND VIOLATIONS

40. ALIASING

To avoid aliasing sample signals at 5x the highest frequency of interest.

To capture digital signal data: take finite measurements at regular intervals using digital devices such as amplifiers and ADCs (Analog to Digital Controllers). If the measurements are higher than the fluctuations in the signal then the measurements will be an accurate depiction of the signal.

However, if the signal fluctuations are fast relative to the sampling rate then information can be lost in the measurements. This can lead to a phenomenon called aliasing.

Features of the signal that are higher than the Nyquist frequency appear as artifacts in lower frequencies.

Subsampling causes loss of information.

Sampling at the Nyquist frequency produces incorrect results

Sampling at 2x Nyquist is the minimum recommended sampling rate but results are less then accurate even at 5x Nyquist. 20x Nyquist gets a clean sampling rate in examples shown here. Sample the analog signal well above the highest frequency in the signal.

Downsampling pro tip:

- Low-pass filter at the Nyquist frequency of the new sampling rate
- Then downsample the signal
- Example:
    - If original recording was done at 2000 Hz
    - And if you want to downsample the signal to 500 Hz
    - Then the new Nyquist frequency is going to be 250 Hz
    - Any feature in the downsampled signal above 250 Hz would be aliased
    - Low-pass filter the signal to 250 Hz before downsampling to 500 Hz.

MATLAB

Code: Fourier_aliasStation.m

Lines of code: 1 to 113

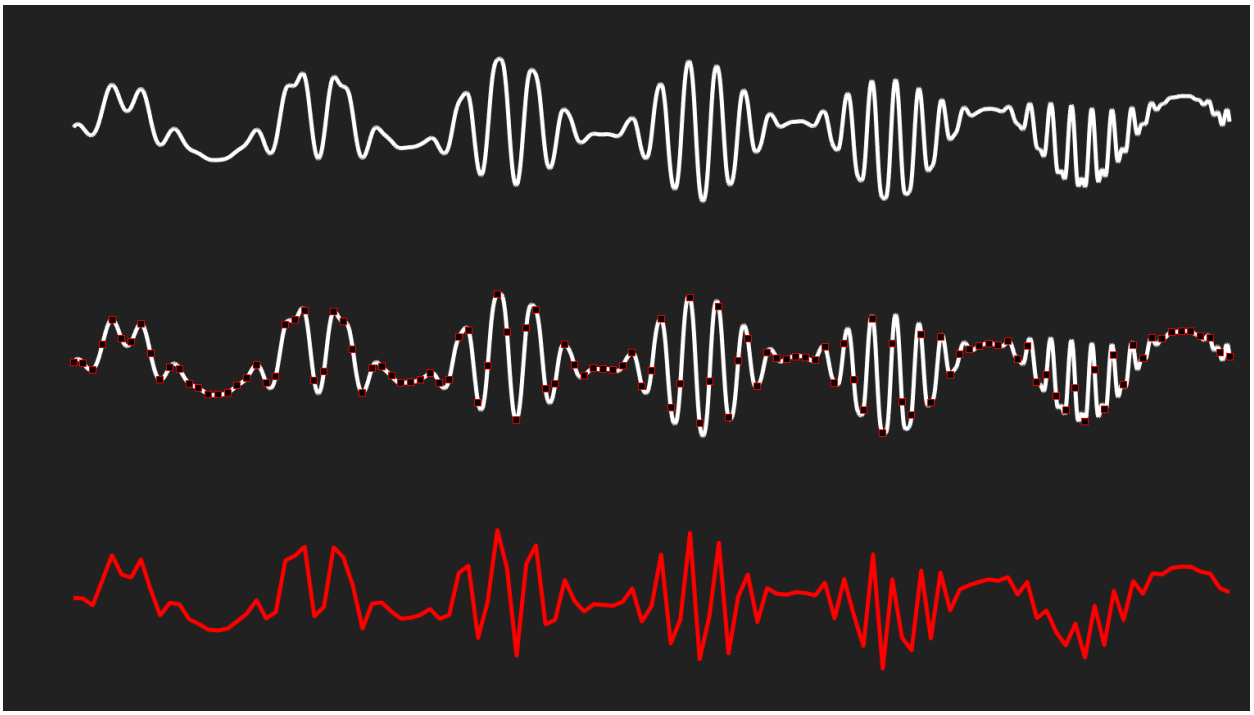


Figure: Use code lines 1 to 33 to reproduce slide showing digital sampling of analog signal, using only 1 in every 25 points. Result in red (3rd plot) shows poor signal reproduction compared to original white signal.

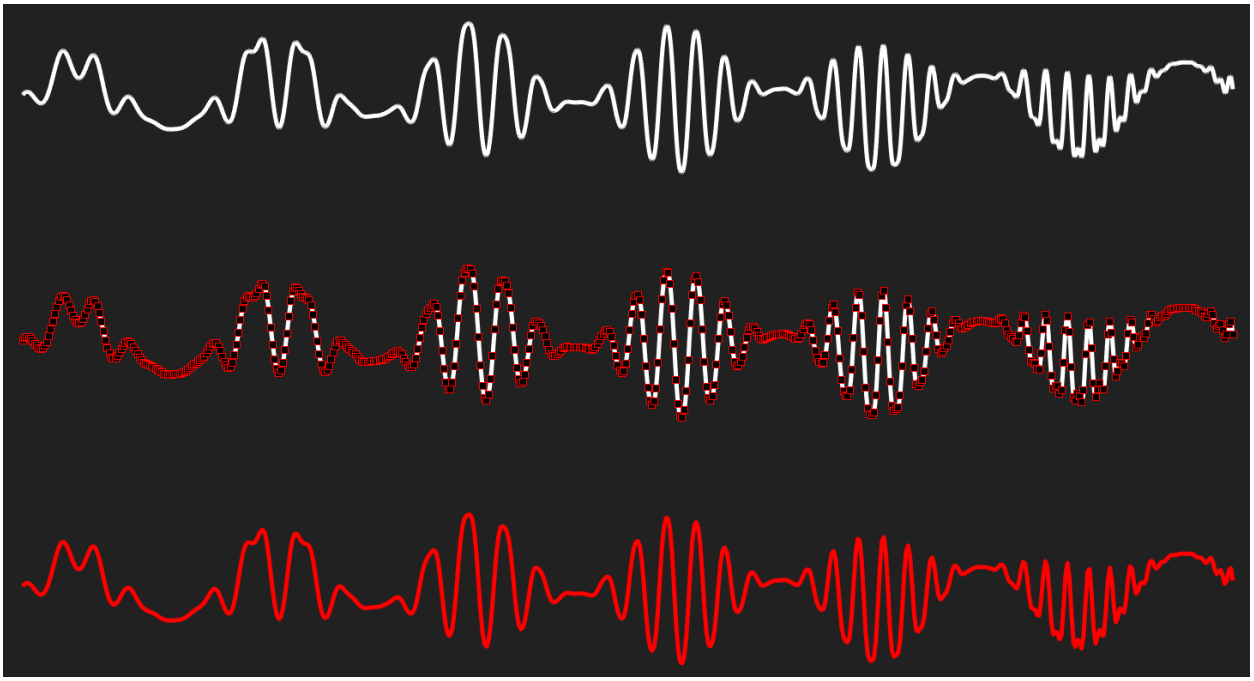


Figure: Use code lines 1 to 33 to reproduce slide showing digital sampling of analog signal, using 1 in every 5 points. Result in red (3rd plot) shows good signal reproduction compared to original white signal.

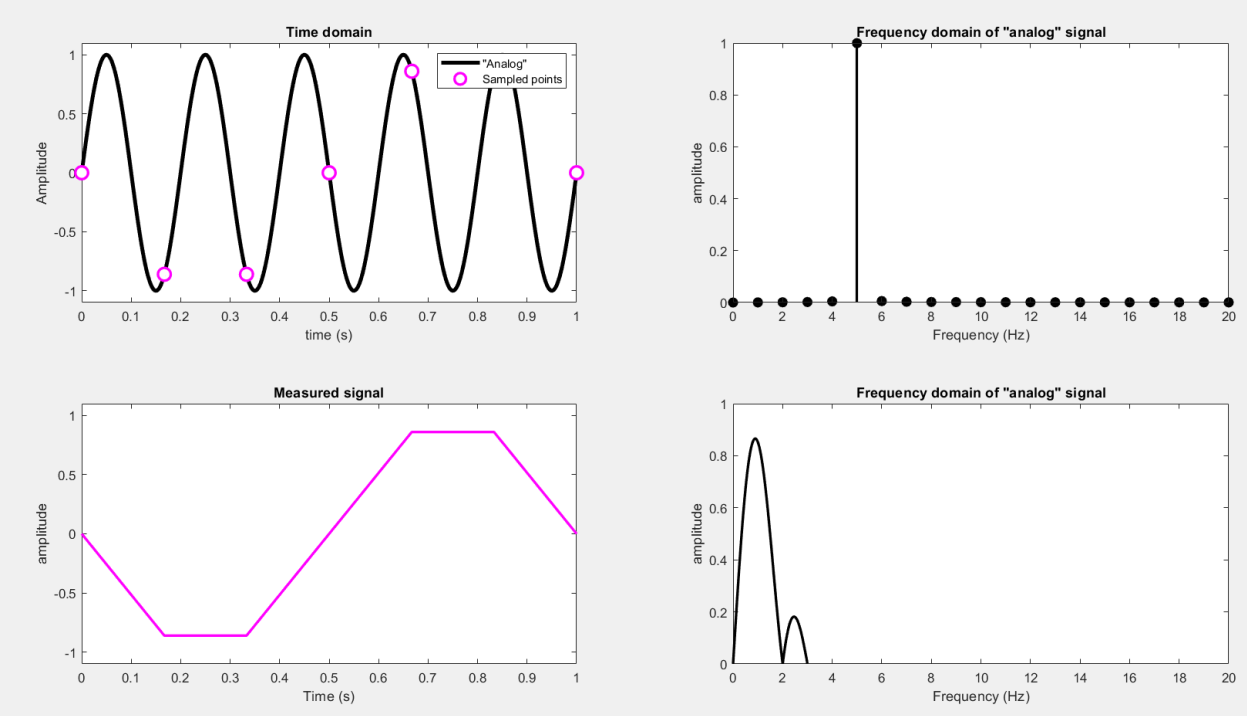


Figure: Lines of code 34 to 86 show alias problem associated with subsampling an analog signal below Nyquist frequency: signal is 5 Hz, sample rate 6 Hz, Nyquist is 10 Hz (twice the signal frequency). Upper left box shows simulated analog signal in black along with sampled points as pink dots at 6 Hz. The upper right box shows a very good amplitude spectrum verifying our 5 Hz signal. Lower left box shows the measured signal in the time domain which is not representative of the analog signal at all. The amplitude spectrum in lower right verifies that the measured signal sampled at 6 Hz is aliased and is reporting lower frequencies than the original analog signal.

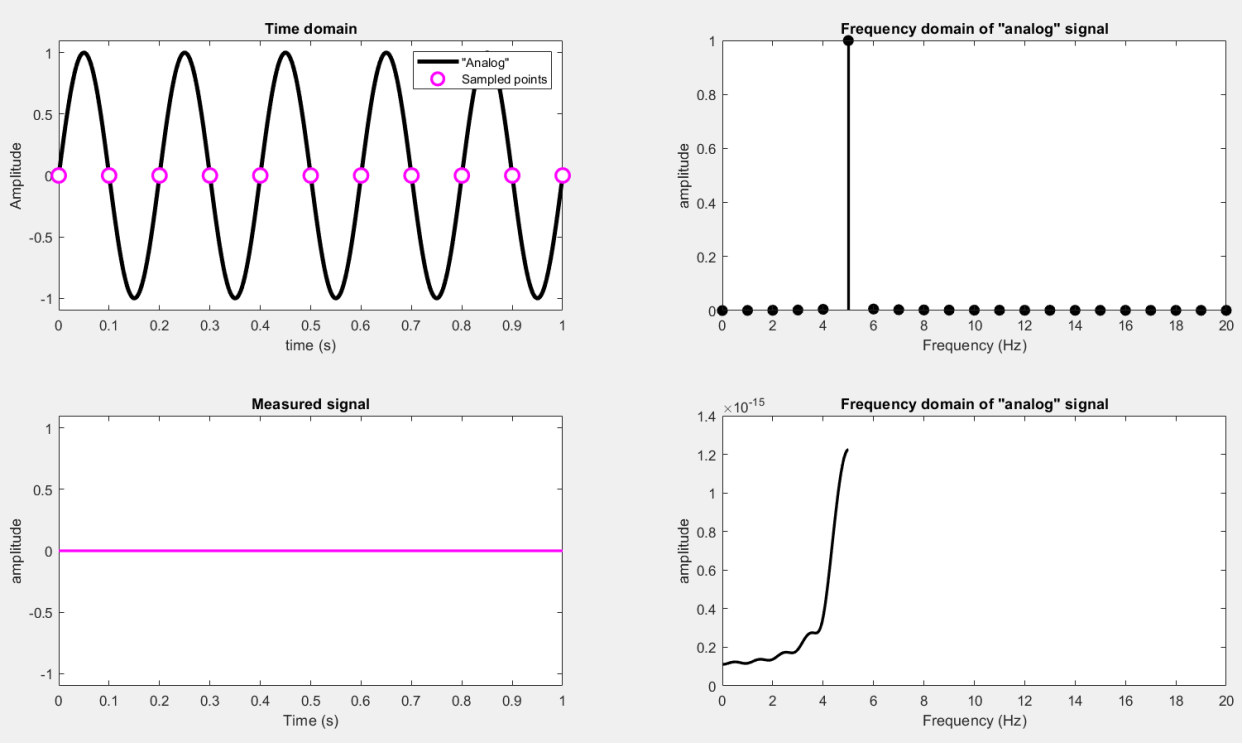


Figure: Lines of code 34 to 86 show alias problem associated with sampling at the Nyquist frequency of 10 Hz.

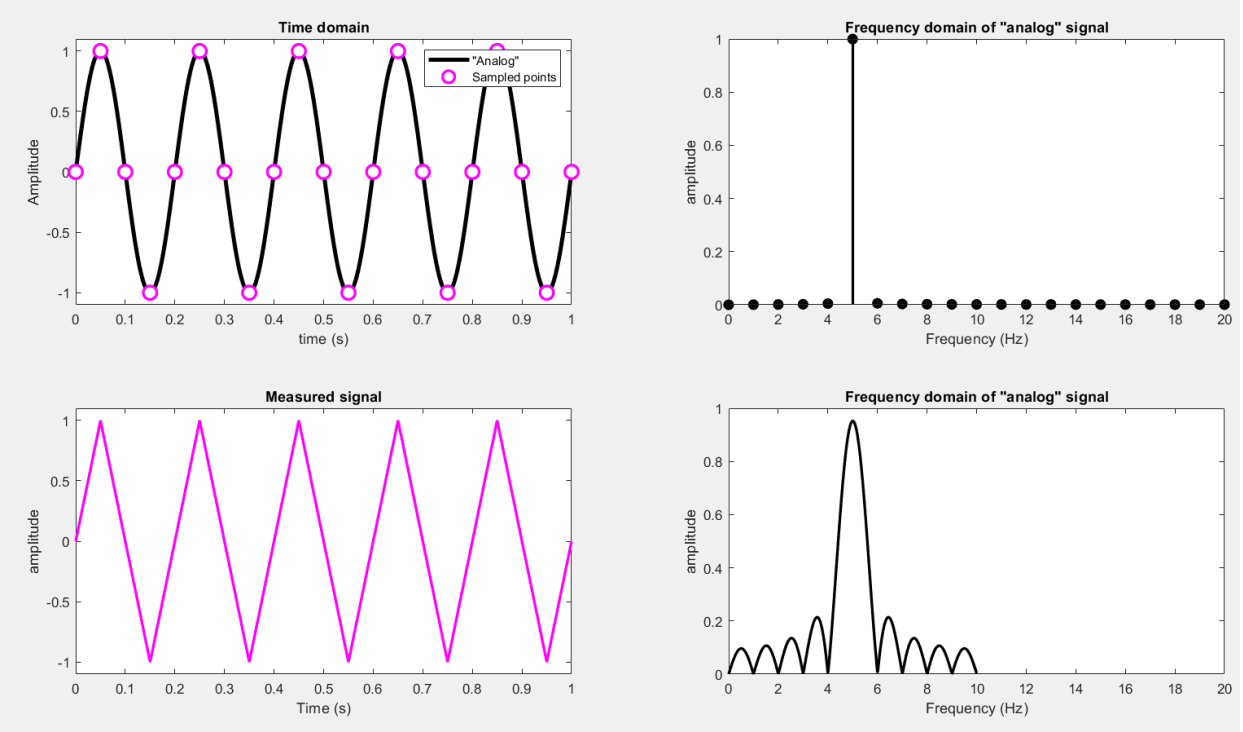


Figure: Lines of code 34 to 86 show sampling at 2x Nyquist frequency of 20 Hz.

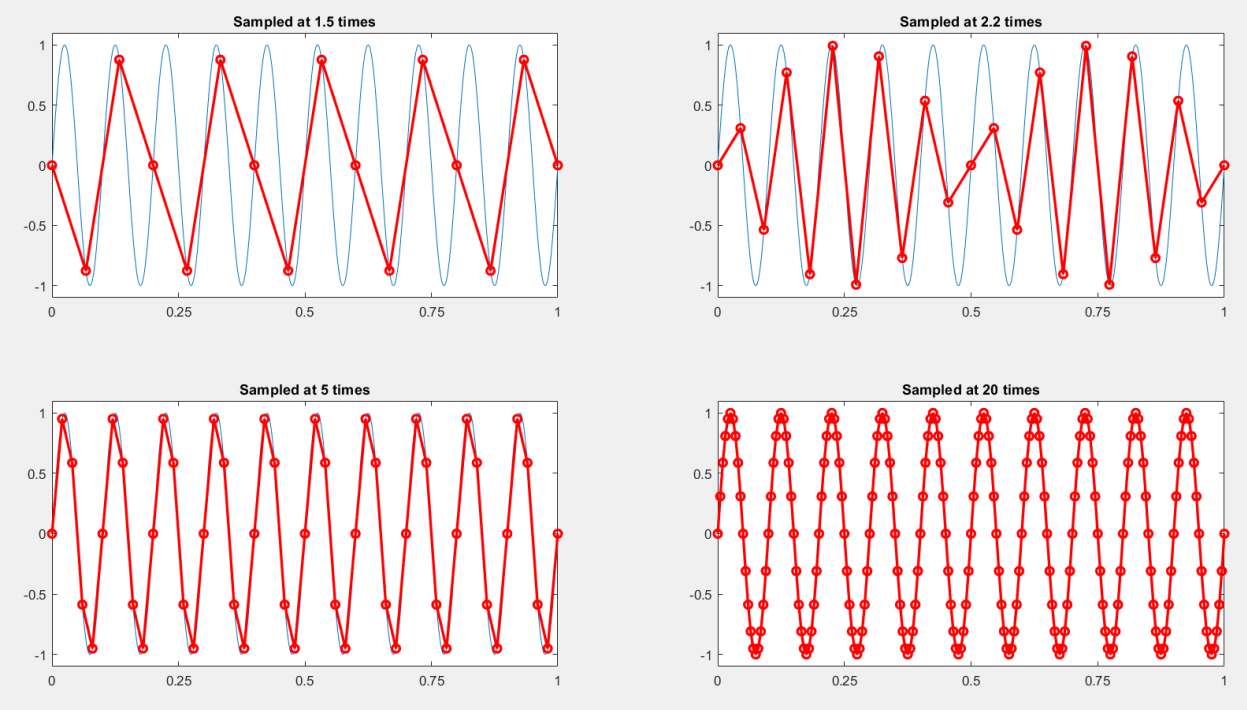


Figure: Lines of code from 87 to 113. Sampling at close to the Nyquist frequency. Plot in upper left shows aliasing with blue signal being the analog signal and the red dots/plot sampled at 1.5 times the sample rate. Plot in upper right is sampled at 2.2 times the highest frequency of the signal (the blue line). Although there is no aliasing, the reconstructed signal is not a great representation. Plot in lower left shows the result of sampling at 5 times the highest frequency of interest (again the blue line). Sampling at 5 times the highest frequency of interest is a good minimum to use in general.

41. SIGNAL STATIONARITY AND NON-STATIONARITIES

Some descriptive statistics of a signal:

- Mean (average or expected value)
- Variance (dispersion around the mean)
- Kurtosis (shape of the distribution)
- Heteroscedasticity (variance of deviances over time)
- Mode (most frequent value)
- Dominant frequency (the one that carries more energy)
- Spectral shape (features a filter function (based on a spectral reference library) that accelerates automatic recognition, search and classification processes)
- Phase stability (vs frequency stability)
- Amplitude (displacement of a wave from its mean value)
- Covariance patterns (?)
- Etc.

These statistics can be captured for the full data set or just a window of the data set. Mean stationarity indicates the mean is the same using different time windows. Mean non-stationarity indicates the mean is different at different parts of the signal.

Definition:

- **Stationarity: A signal is stationary when its statistical characteristics do not significantly change over time.**
- Non-stationarity: A signal is non-stationary when the stationarity is violated.

Sources of definitional ambiguities:

- Many features of a signal
    - Example 1: Amplitude non-stationary but frequency stationary.
    - Example 2: Frequency non-stationary but amplitude stationary.
- "Significantly different is threshold-dependent, and the appropriate threshold may be unknown or arbitrary.
- The size and placement of time windows can be subjective and uncertain.

Non-Stationarities affect the results of a Fourier transform.

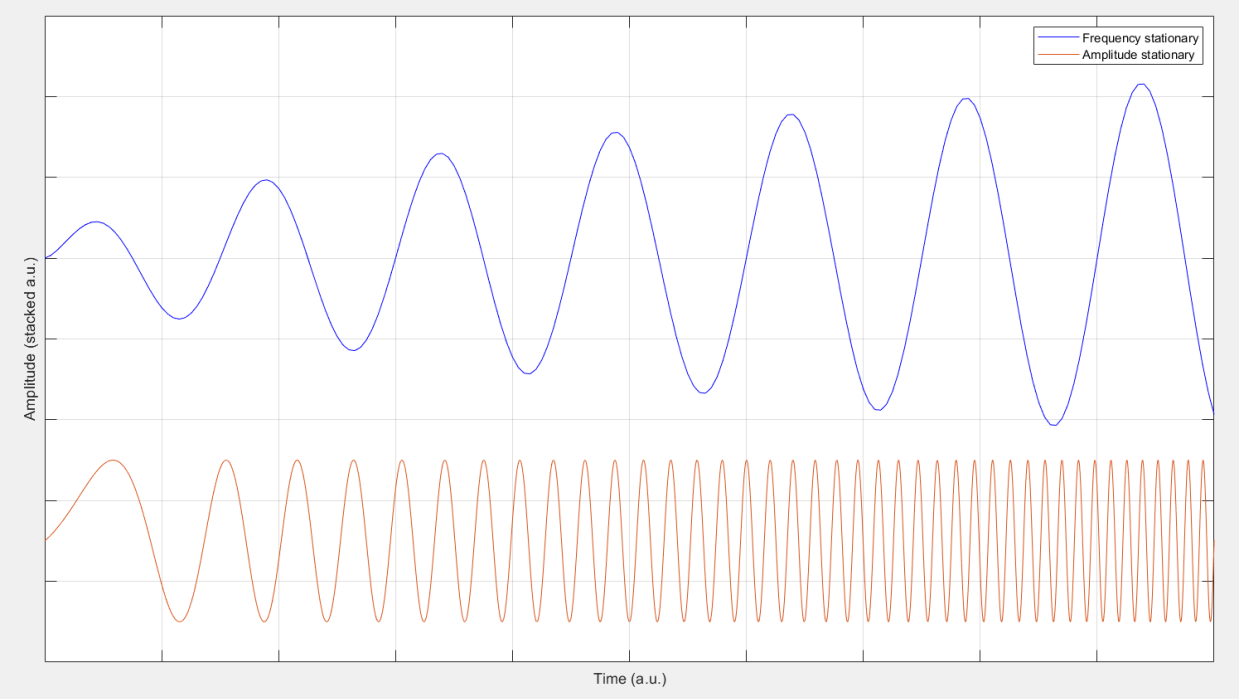


Figure: Demonstration of amplitude and frequency stationarity.

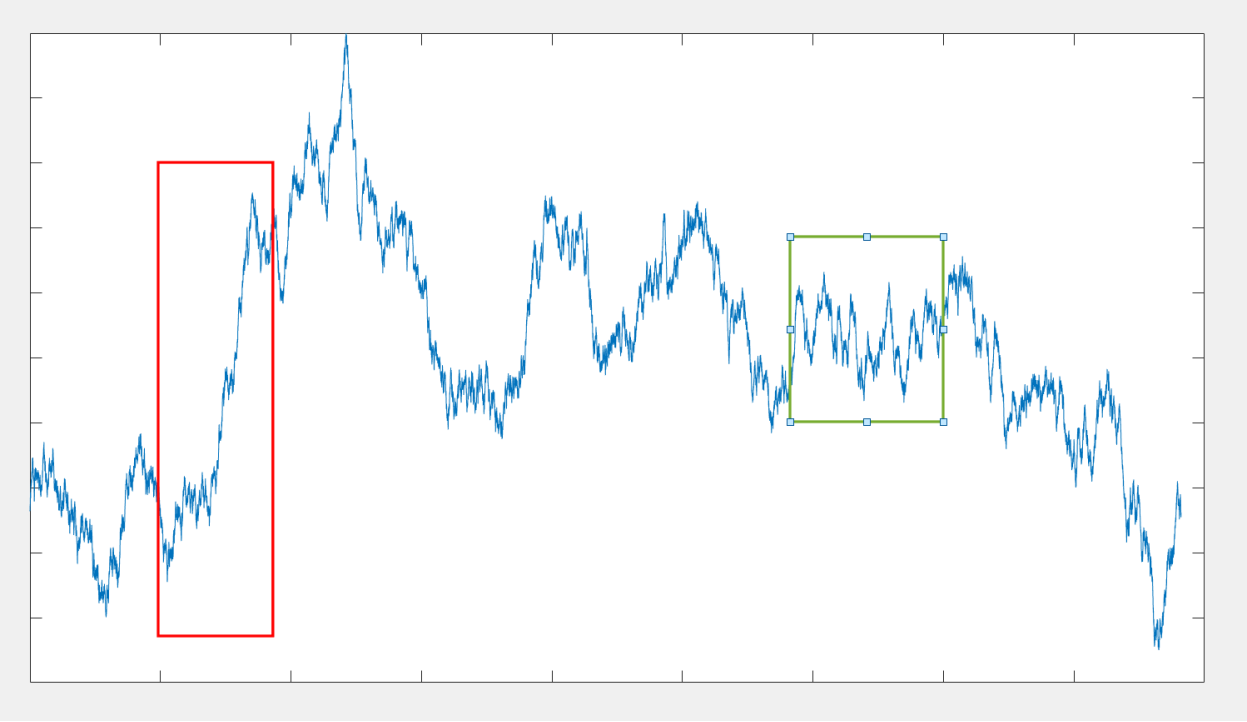


Figure: the placement and size of time windows can be subjective and uncertain. In this gray noise signal, red box shows a segment of mean non-stationarity, and green box shows a segment wo what appears to be mean stationary behavior.

42. EFFECTS OF NON-STATIONARITIES ON THE POWER SPECTRUM

Two major uses of the Fourier transform:

- Spectra analysis:
    - Some signals are better understood in the frequency domain - particularly signals with a strong rhythmicity.
    - Spectral analyses can reveal insights that cannot be seen in the time domain.
- A means to an end in signal processing:
    - Use the convolution theorem to perform operations in the frequency domain, including
        - filtering,
        - autocorrelation,
        - feature selection,
        - etc.
    - Frequency-domain computations are often easier and faster than equivalent time-domain computations.

Observe the effects of non-stationarities on amplitude spectra.

Learn about edge effects in the time and frequency domains.

Non-stationarity can make the results of a Fourier transform difficult to interpret visually.

MATLAB

Fourier_aliasStation.m

Lines of code from 114 to 290

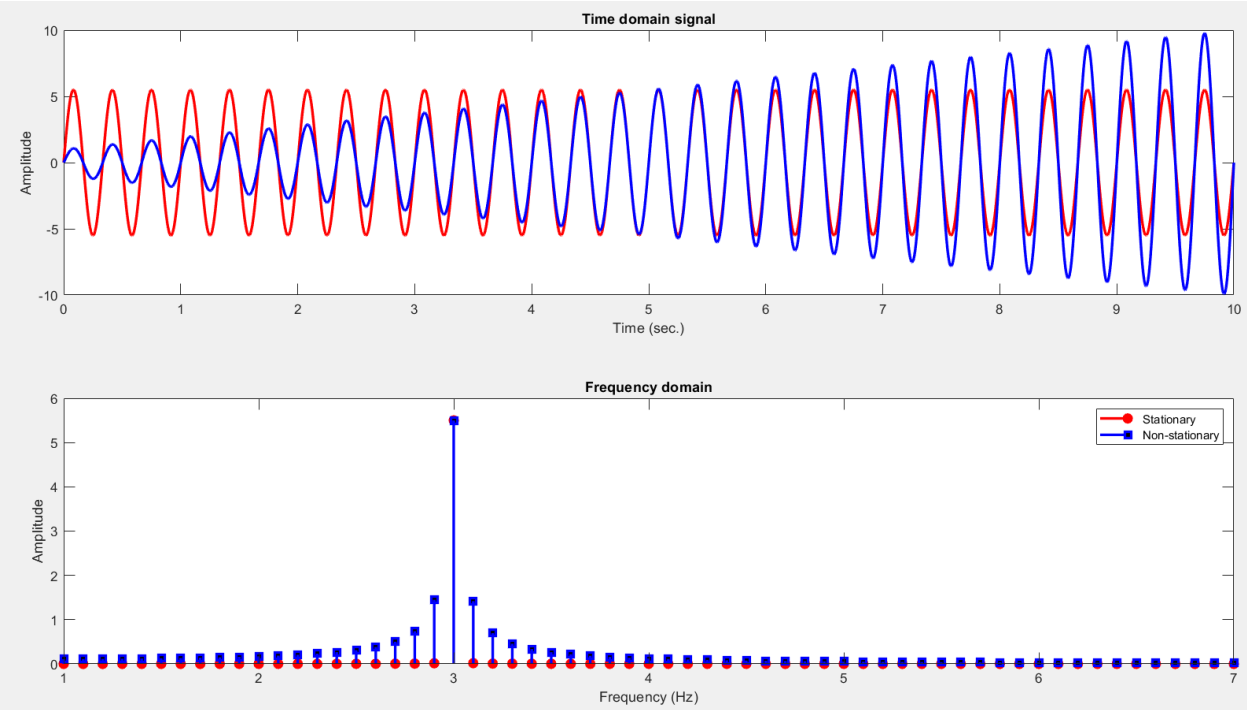


Figure shows effect of amplitude non-stationarit. Lines of code from 123 to 161. Demonstrate frequency lobes for fft when amplitude is non-stationary. Lobes are needed to describe the changes in amplitude of the blue amplitude non-stationary wave.

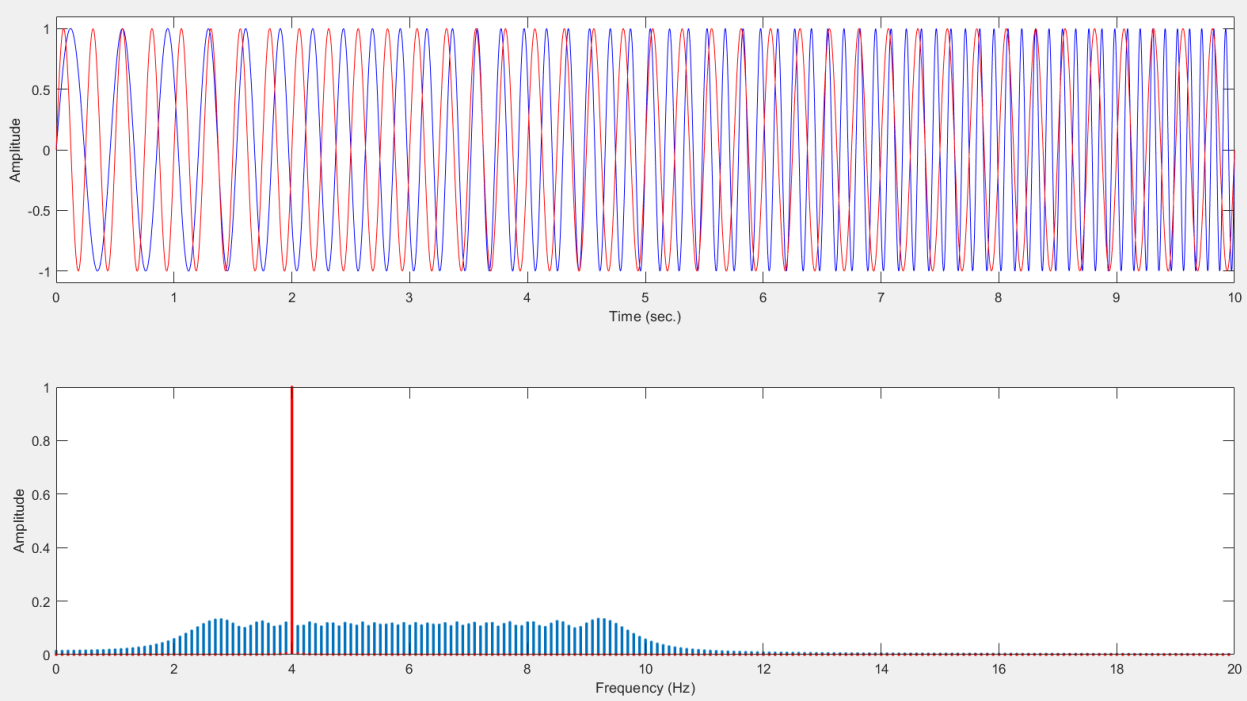


Figure shows effect of frequency non-stationarity. Lines of code from 162 to 190. Red signal is both constant amplitude and constant frequency and has only one spike in fft showing frequency. Blue line is constant amplitude but with a variable frequency like a “chirp”. The fft for blue line includes energy at many frequencies as a result of frequency non-stationarity.

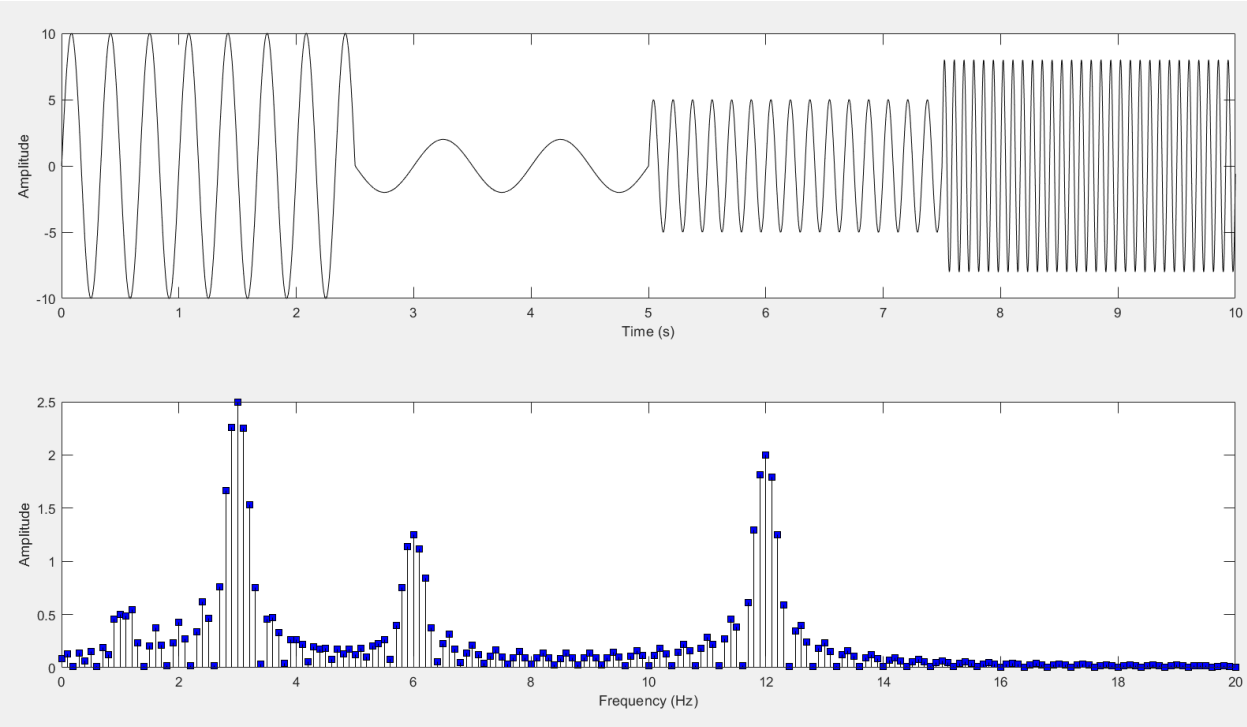


Figure shows the effect of sharp transitions. Lines of code from 191 to 215. Four non-stationarities in the time domain with resultant fft. Amplitude does not match and there is more than one frequency in support of the transitions from one stationarity to another. The 3 Hz peak in frequency domain is ¼ the amplitude in the time domain. Which makes sense as the 3 Hz signal is ¼ of the total time.

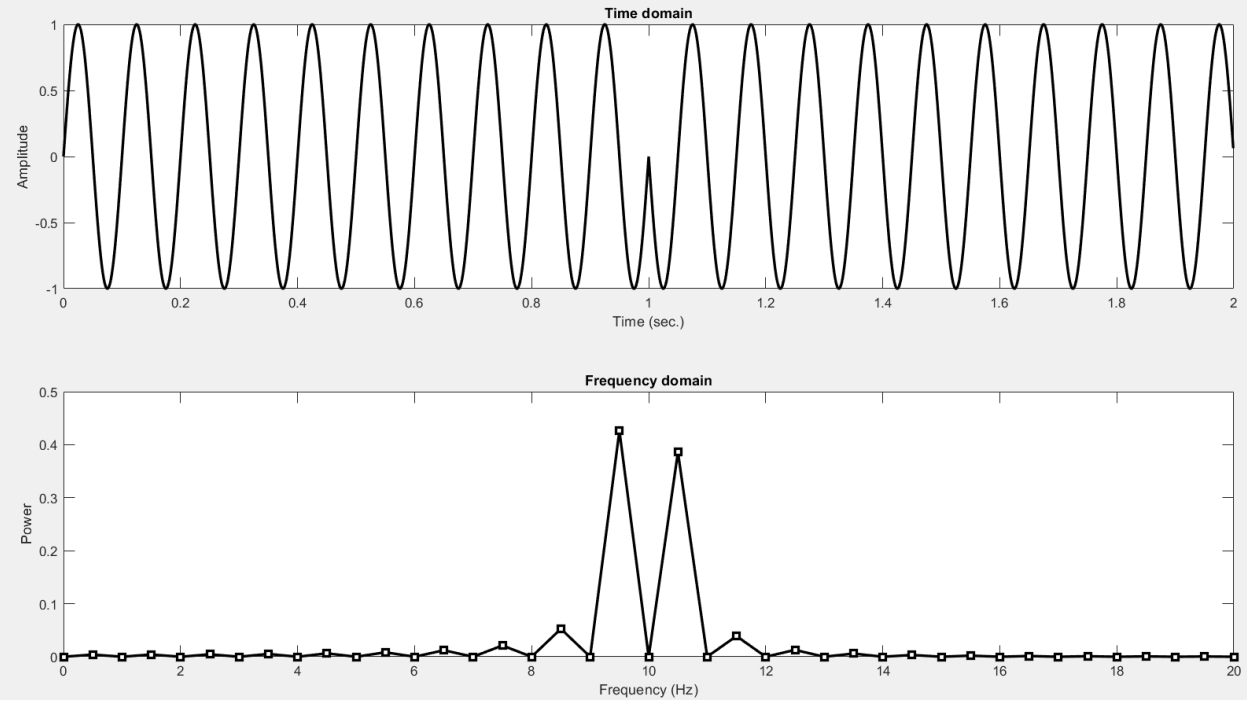


Figure shows the effect of a phase reversal. Lines of code from 216 to 240. Two 10 Hz sine waves placed back to back with opposite phase. Now the 10 Hz frequency shows as 0 with two side lobes. A windowing method or Welches method could be used to recover the expected fft.

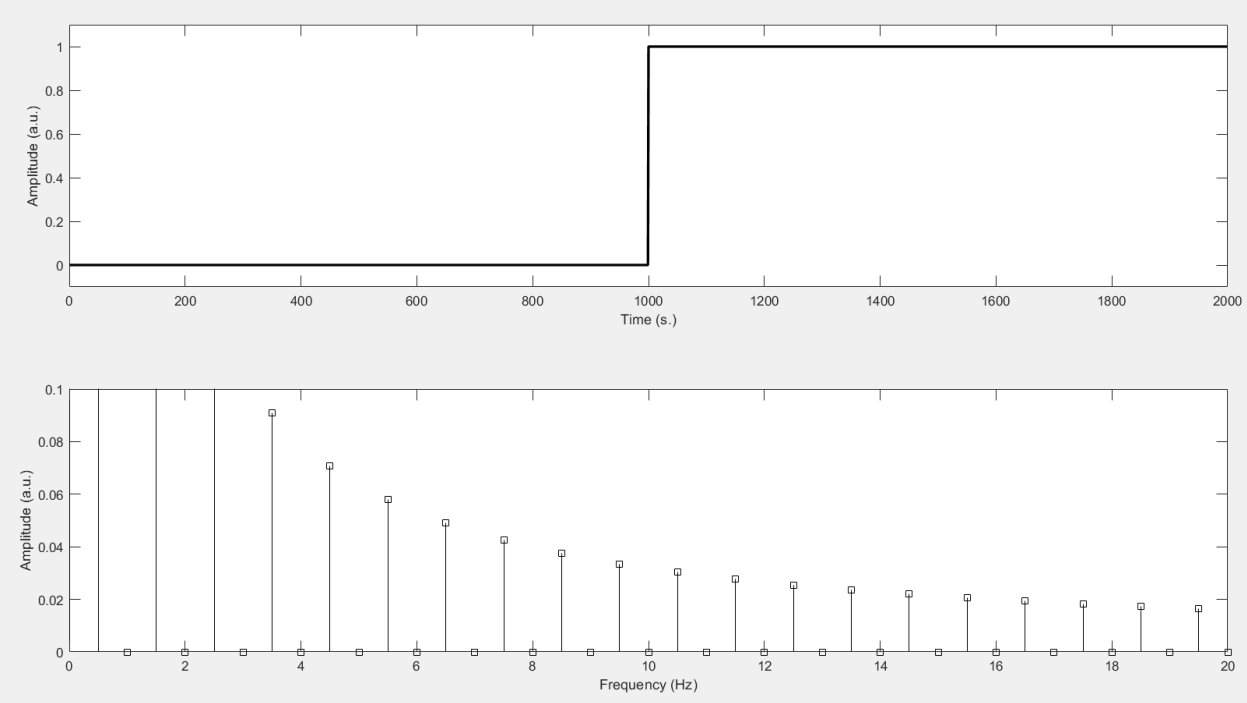


Figure shows the effect of edges and edge artifacts. Lines of code from 241 to 263: It takes many sine waves to represent one sharp edge.

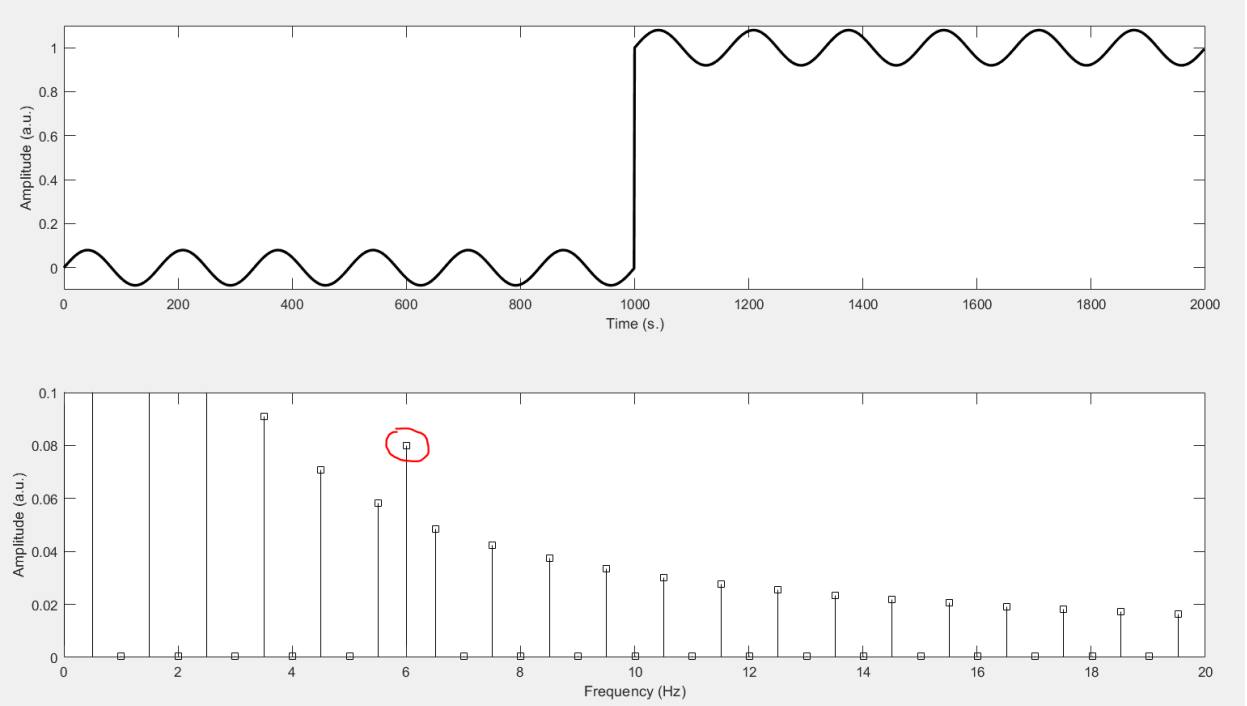


Figure shows same code as above but with 6 Hz sine wave added (by uncommenting line 248 in code) . Now the 6Hz fft line is shown as well as some of the sine waves to reproduce the edge itself. So a stationary signal can be embedded into a non-stationary signal and still be resolved.

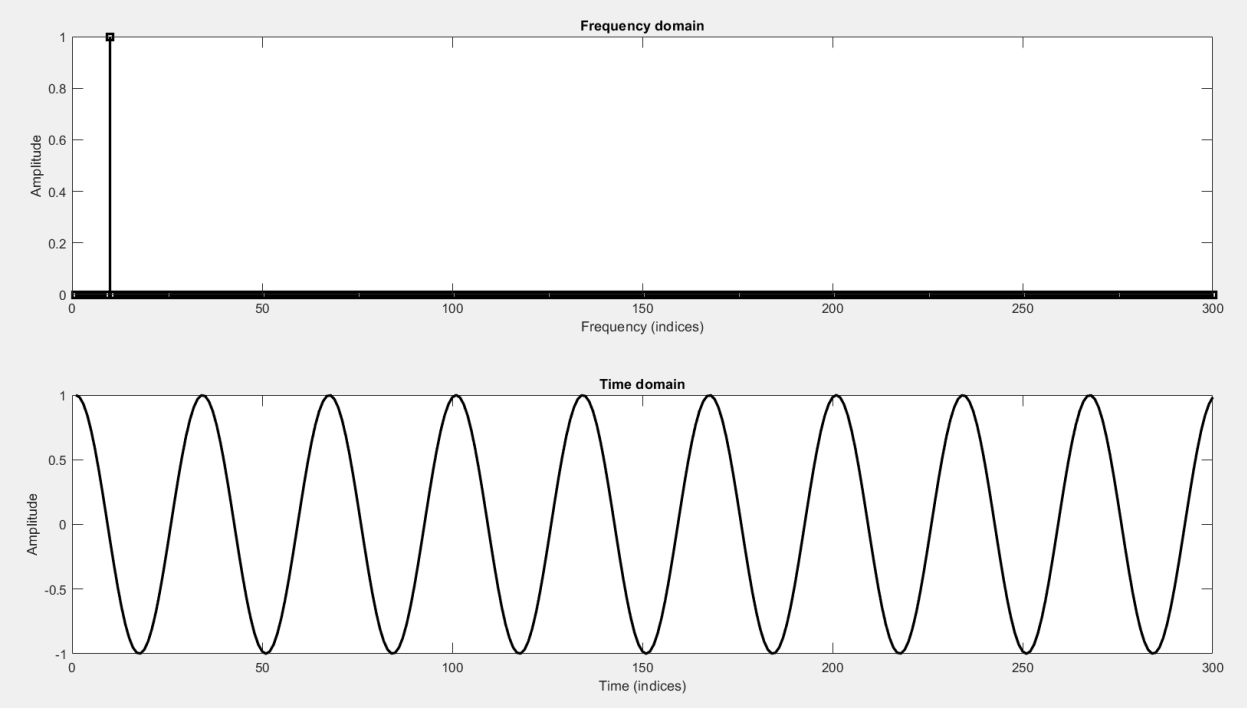


Figure shows a spike in the frequency domain. Lines of code from 264 to 290. A non-stationarity in the frequency domain produces a pure sine wave in the time domain. A spike in the frequency domain corresponds to a sine wave in the time domain. A sign wave can be thought of as an artifact of a spike in the frequency domain. An impulse in the frequency domain produces a ringing effect in the time domain.

Music is an example of a collection of non-stationary waves. The brain needs non-stationarities to process stimuli and produce thought and behavior.

43. SOLUTION TO UNDERSTANDING NONSTATIONARY TIME SERIES

Time-frequency analysis (STFFT and wavelet convolution):

The Goal of STFFT is to understand how spectral features are changing over time.

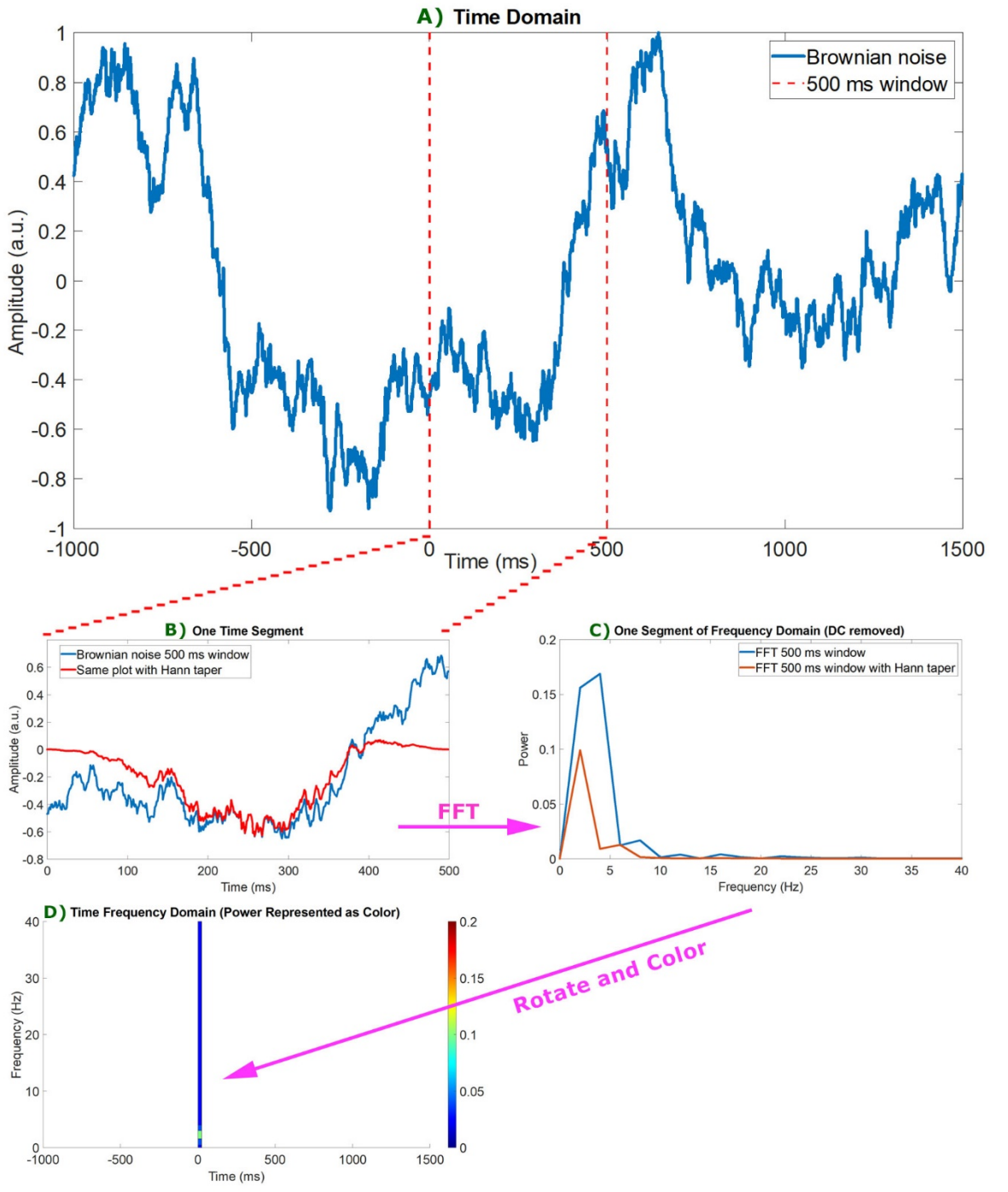


Figure: Short-time FFT example. A) Upper plot shows a signal with singularities. B) Middle left plot shows one 500 ms segment of the signal along with result of dot product with Hann taper. C) Middle right shows the FFT for just this one-time segment. D) Lower left shows the first 500 ms FFT segment rotated and colored to show time,

frequency, and power (using colorbar). The process would be repeated moving the 500 ms slice one 25 ms increment to the right at a time. The resulting color-coded signal can show how FFT changes over time. See MATLAB results below for an example.

Morlet wavelet = Sine wave multiplied by a Gaussian

For time-frequency analysis, create many Morlet wavelets with many different frequencies. The wavelets are convolved with the signal produces a time series of complex dot products. This produces one wavelet coefficient per frequency and per time point.

Magnitude of dot product is the amplitude (square for power)

Angle relative to positive real axis to get phase

The resulting time-frequency plot looks very similar to the FFT

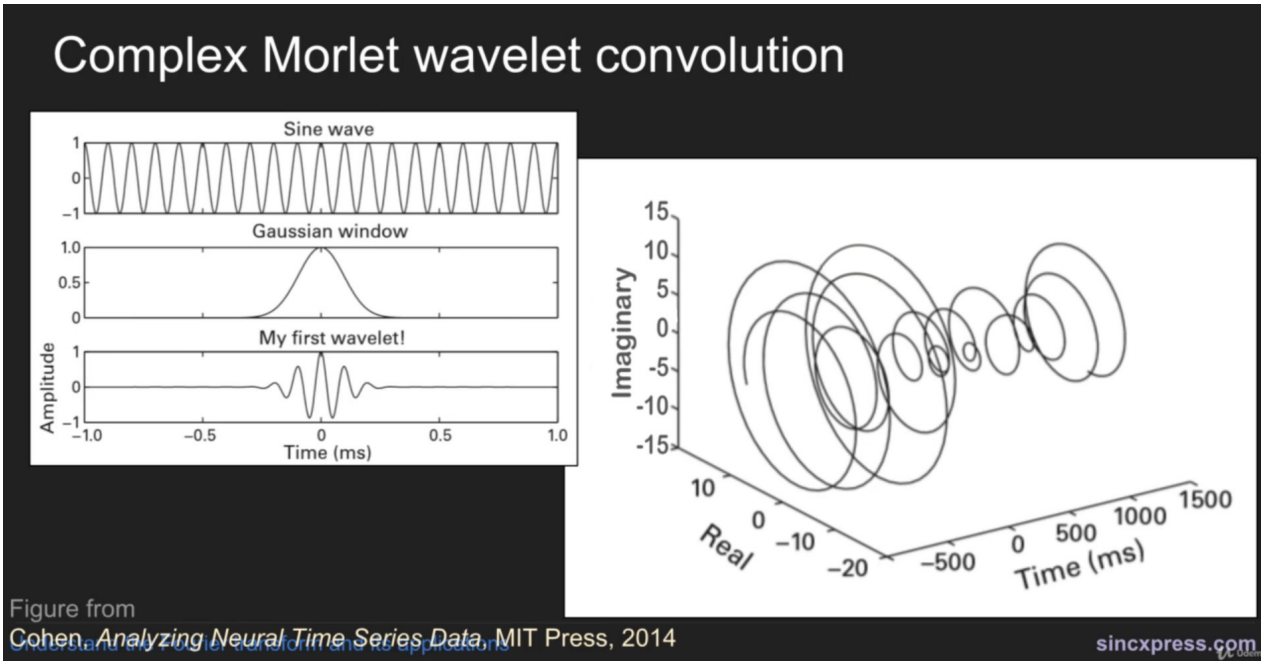


Figure: Complex Morlet wavelet convolution

MATLAB

Fourier_aliasStation.m

Lines of code: 299 to 408

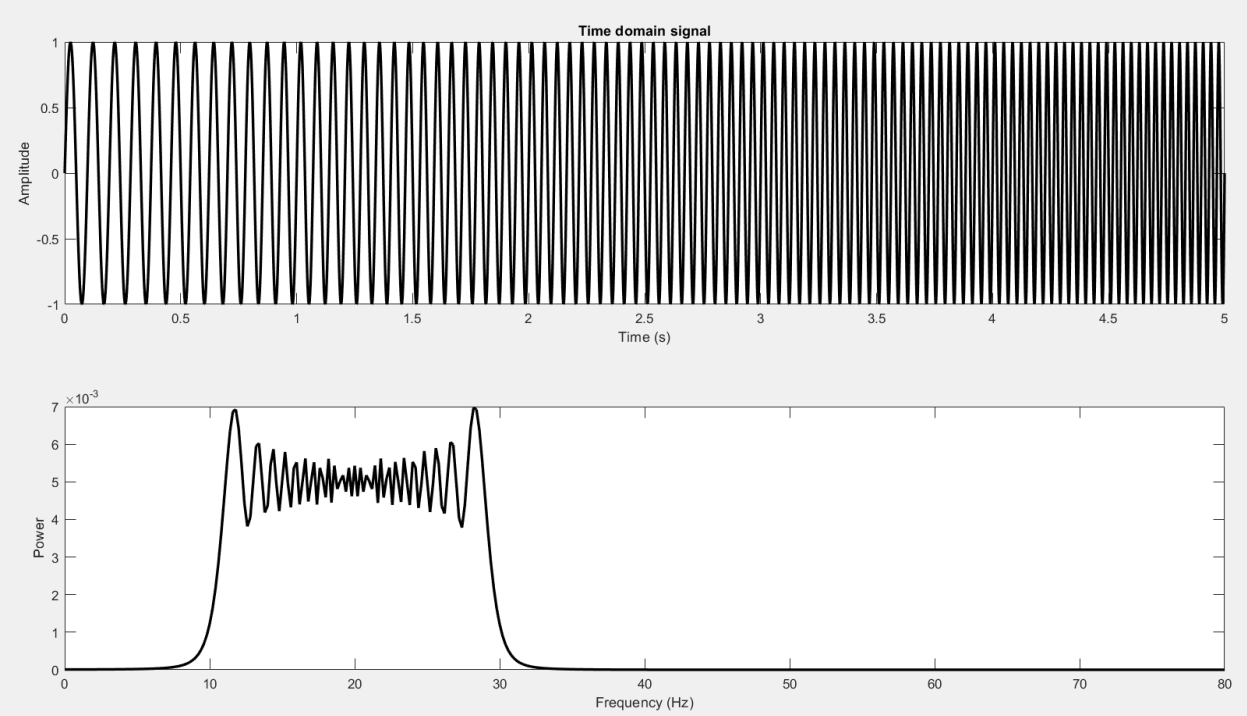


Figure: Create signal (chirp) from 10 Hz to 30 Hz to be used in the following examples. Lines of code from 299 to 328.

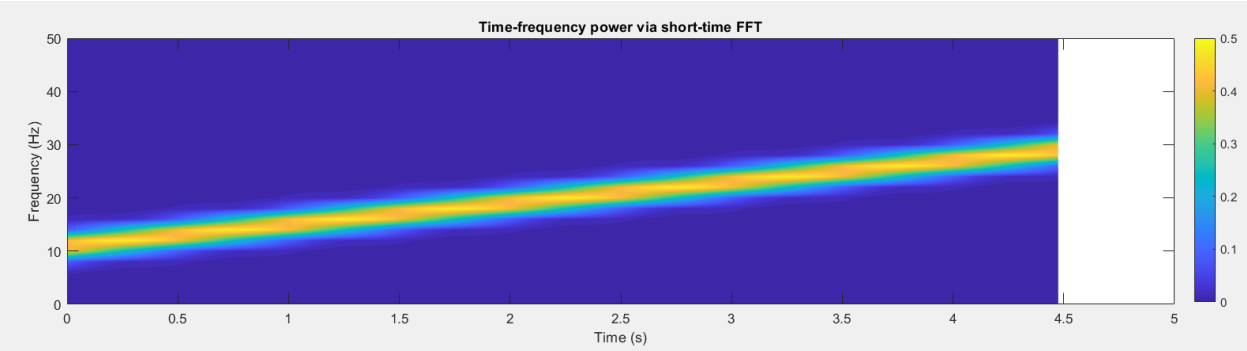


Figure FFT going from 10 Hz to 30 Hz using Short-Time FFT technique. Lines of code from 330 to 366

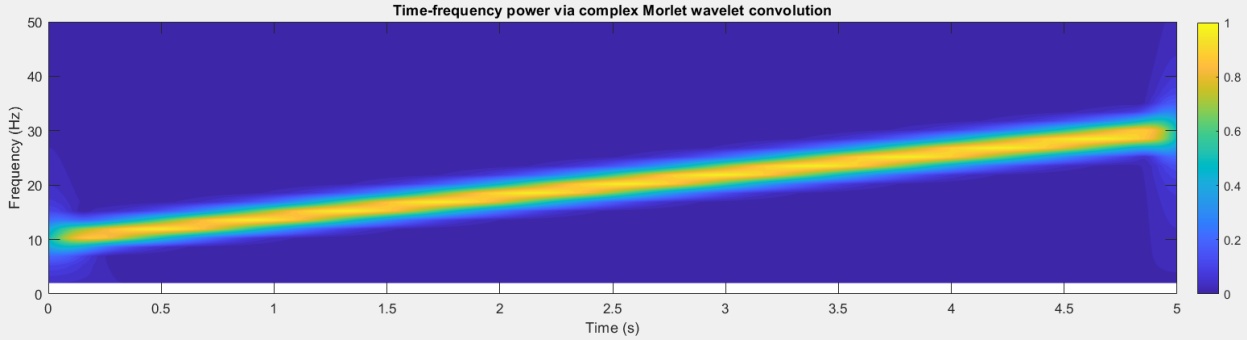


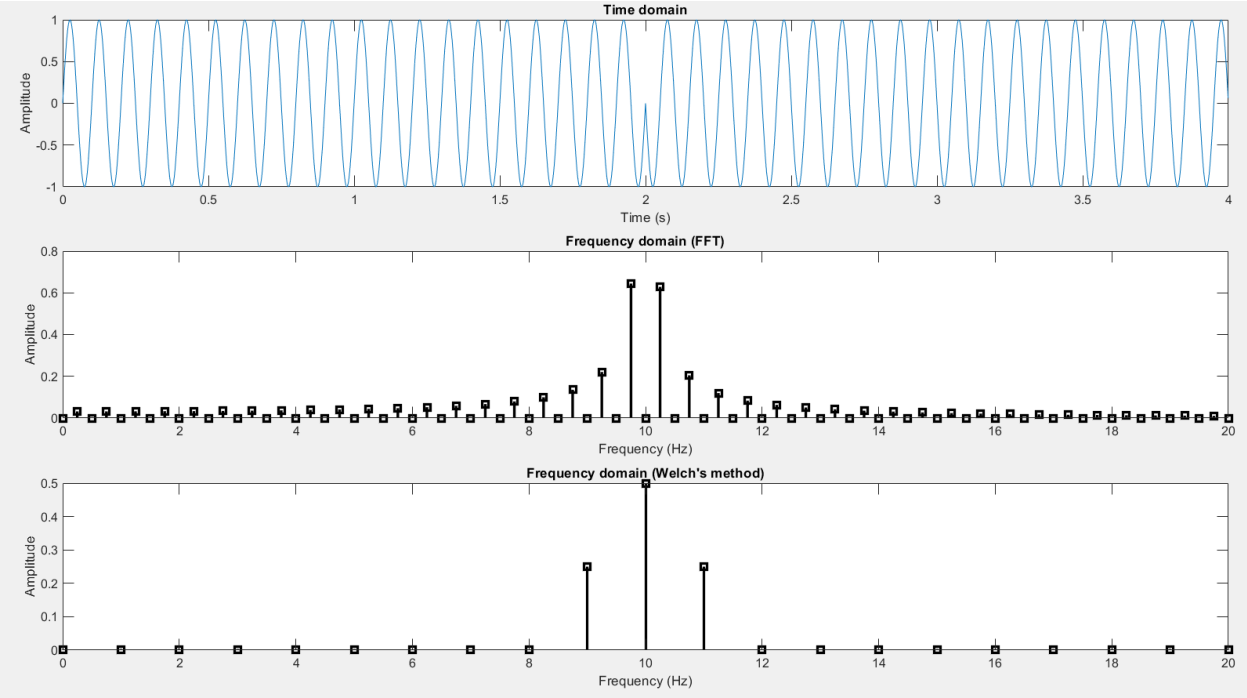
Figure FFT going from 10 Hz to 30 Hz using complex Morlet wavelet convolution technique. Lines of code from 367 to 408.

44. WINDOWING AND WELCH'S METHOD

MATLAB

Fourier_aliasStation.m

Lines of code from 417 to 484



45. INSTANTANEOUS FREQUENCY

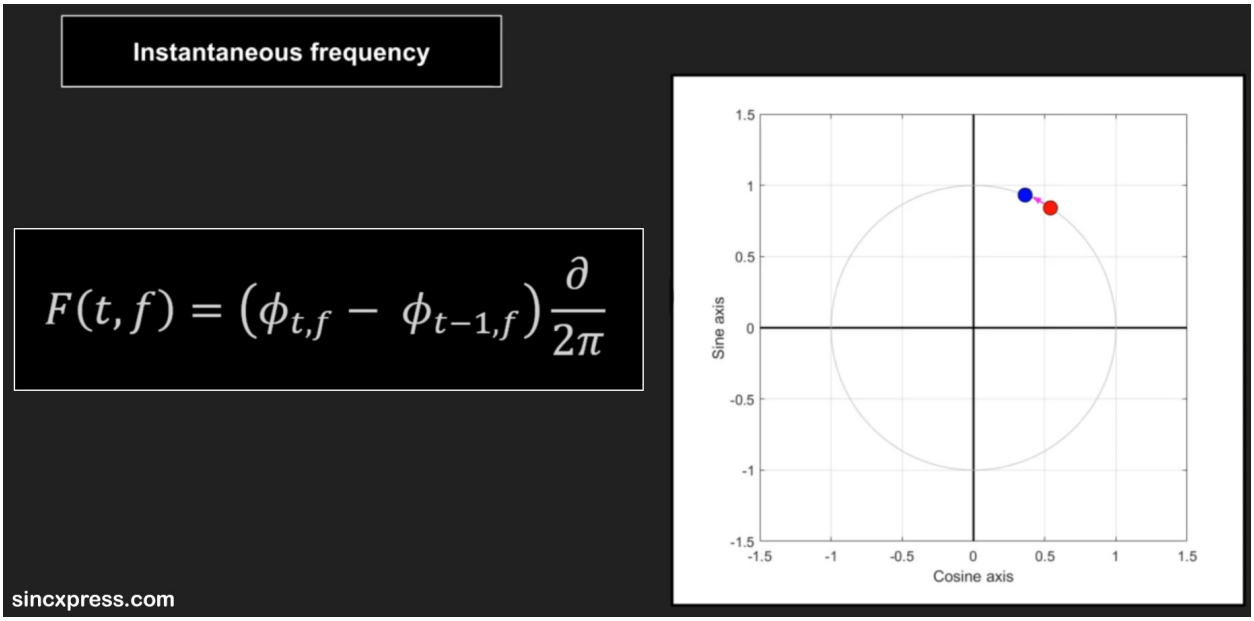
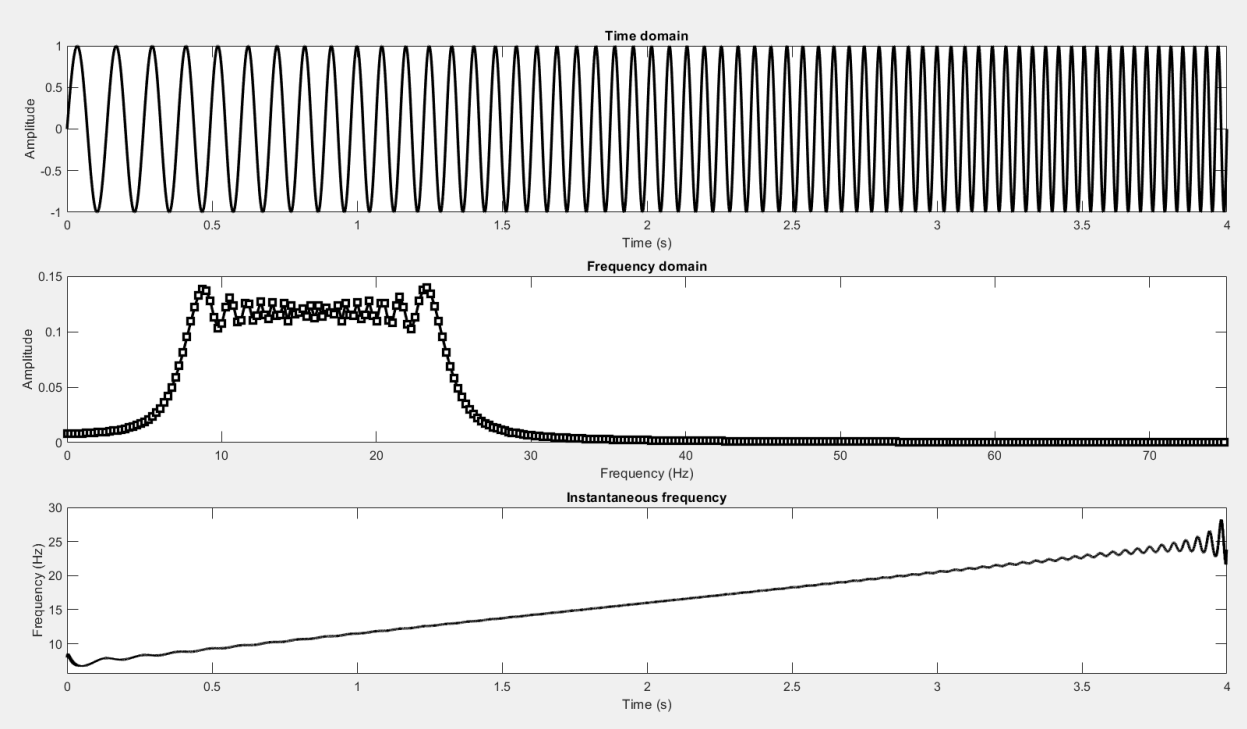


Figure: Where phi is phase and delta is sampling rate

MATLAB

Fourier_aliasStation.m

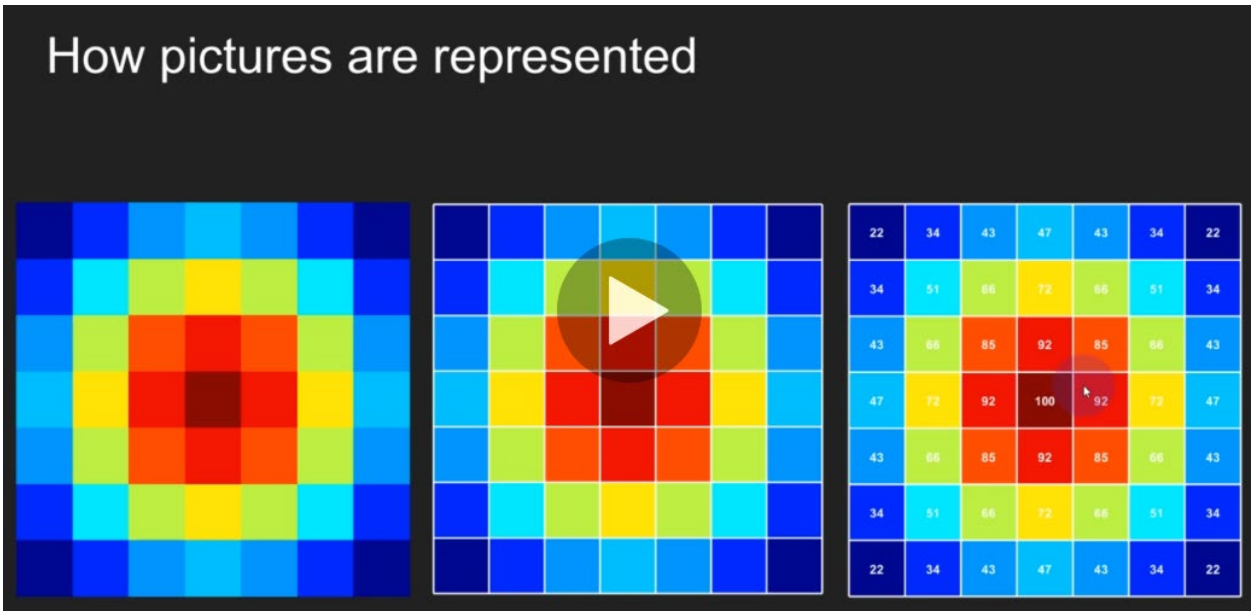
Lines of code from 493 to 530

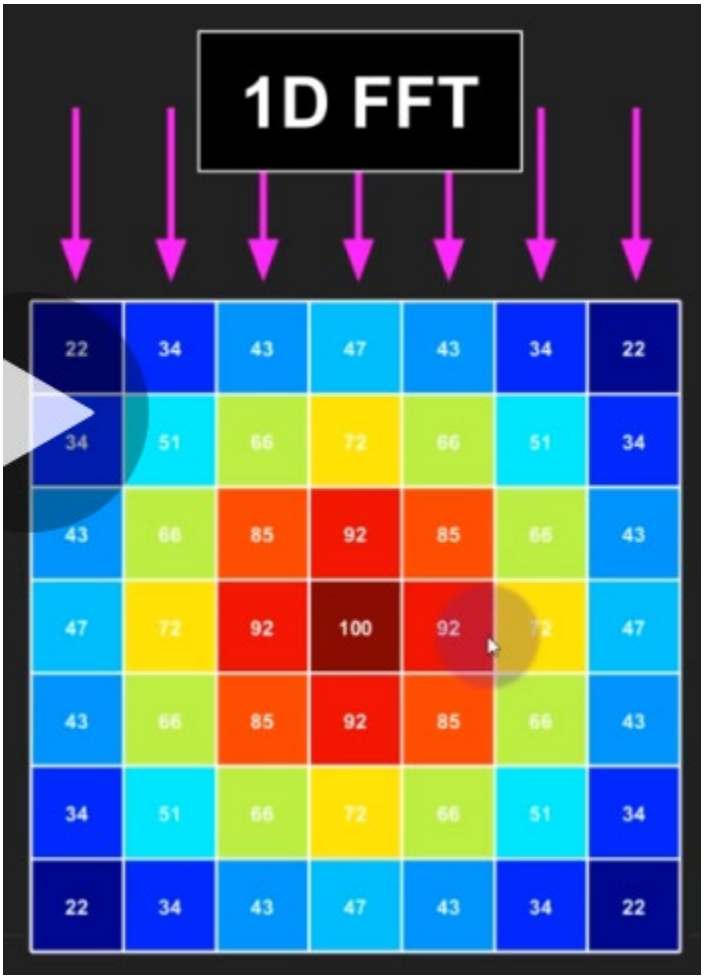


SECTION 8: 2D FOURIER TRANSFORM

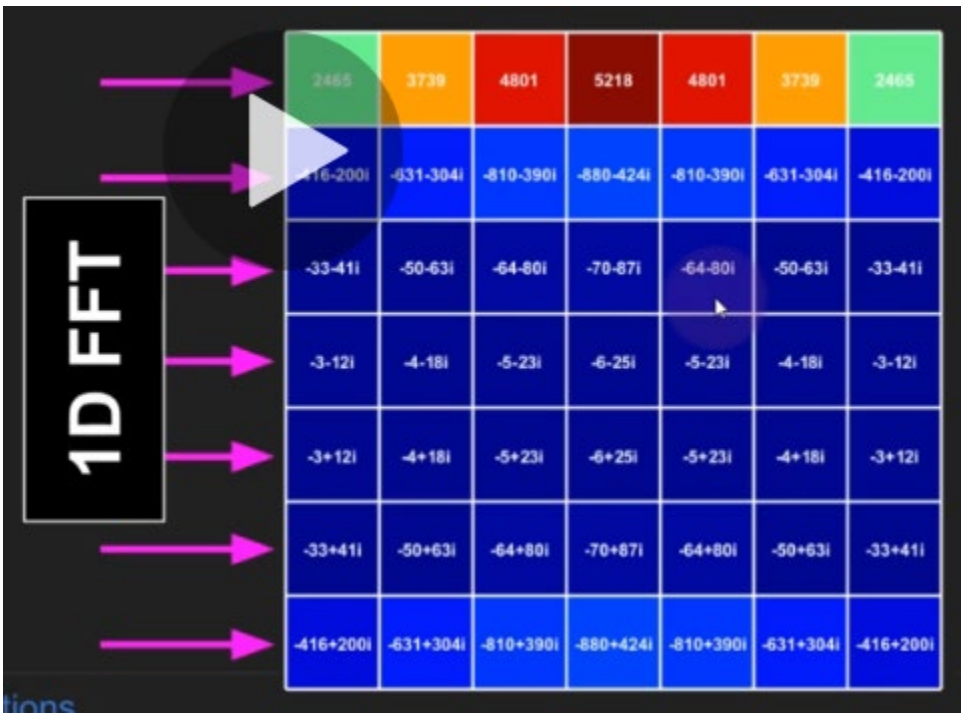
46. HOW THE 2D FFT WORKS

2D Fourier transform for images

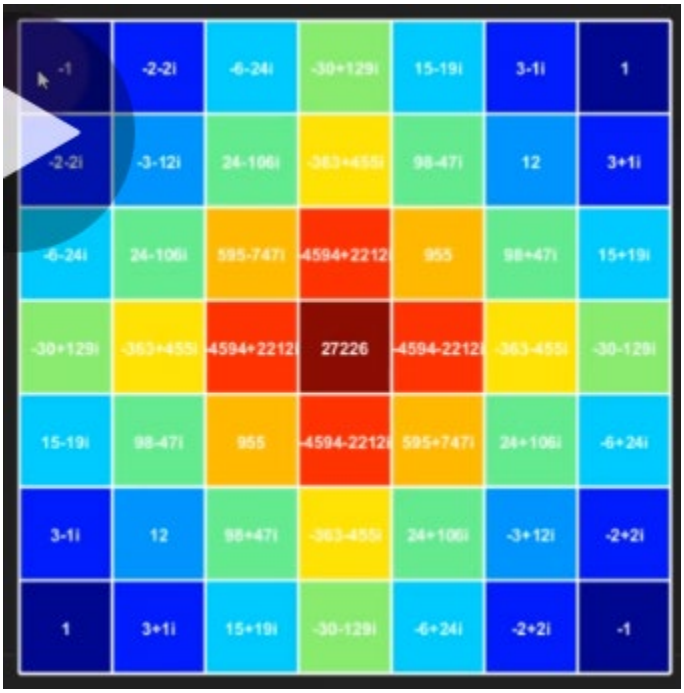

How pictures are represented
22 34 43 47 43 34 22
34 51 66 72 66 51 34
43 66 85 92 85 66 43
47 72 92 100 92 72 47
43 66 85 92 85 66 43
34 51 66 72 66 51 34
22 34 43 47 43 34 22


1D FFT
22 34 43 47 43 34 22
34 51 66 72 66 51 34
43 66 85 92 85 66 43
47 72 92 100 92 72 47
43 66 85 92 85 66 43
34 51 66 72 66 51 34
22 34 43 47 43 34 22


2465 3739 4801 5218 4801 3739 2465
-631-304i -810-390i -880-424i -810-390i -631-304i -416-200i
-33-41i -50-63i -64-80i -70-87i -64-80i -50-63i -33-41i
-3-12i -4-18i -5-23i -6-25i -5-23i -4-18i -3-12i
-3+12i -4+18i -5+23i -6+25i -5+23i -4+18i -3+12i
-33+41i -50+63i -64+80i -70+87i -64+80i -50+63i -33+41i
-416+200i -631+304i -810+390i -880+424i -810+390i -631+304i -416+200i


1D FFT
2465 3739 4801 5218 4801 3739 2465
-631-304i -810-390i -880-424i -810-390i -631-304i -416-200i
-33-41i -50-63i -64-80i -70-87i -64-80i -50-63i -33-41i
-3-12i -4-18i -5-23i -6-25i -5-23i -4-18i -3-12i
-3+12i -4+18i -5+23i -6+25i -5+23i -4+18i -3+12i
-33+41i -50+63i -64+80i -70+87i -64+80i -50+63i -33+41i
-416+200i -631+304i -810+390i -880+424i -810+390i -631+304i -416+200i

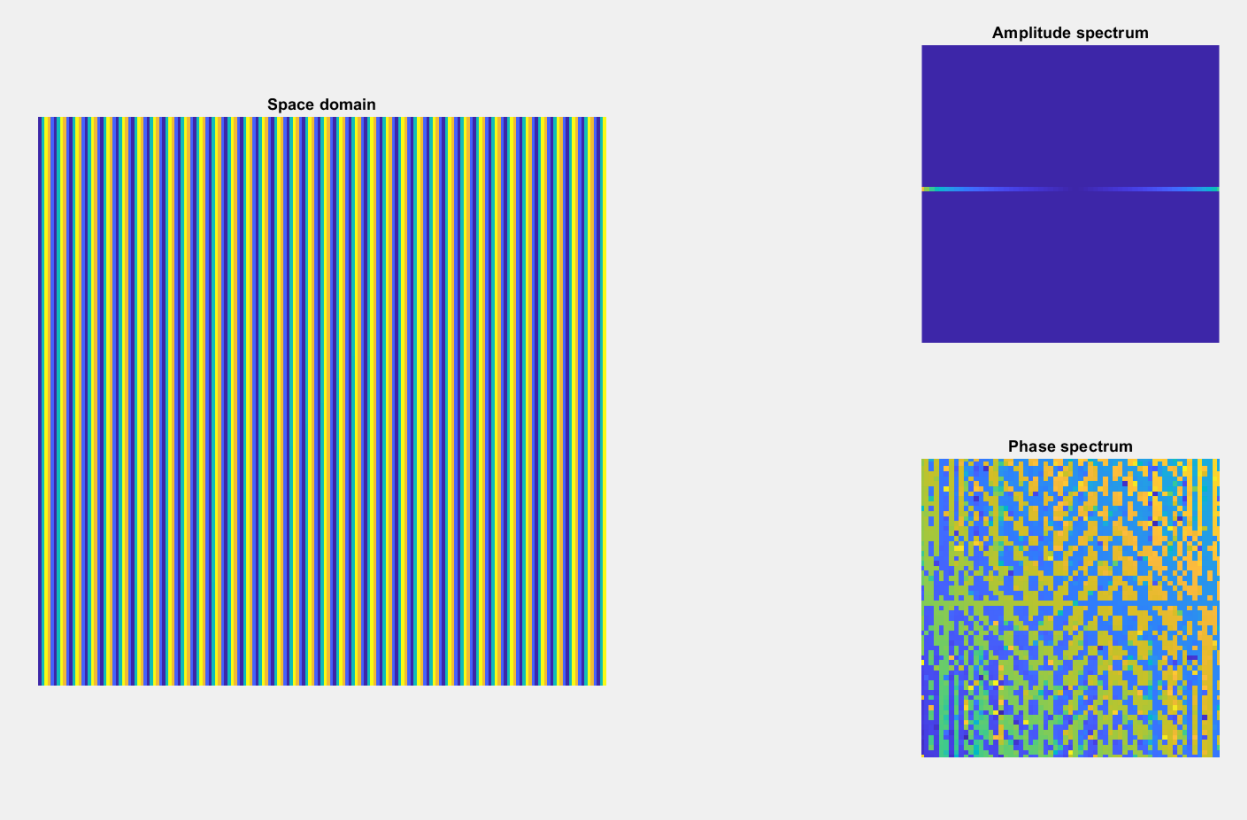
The Nyquist frequency is in the middle



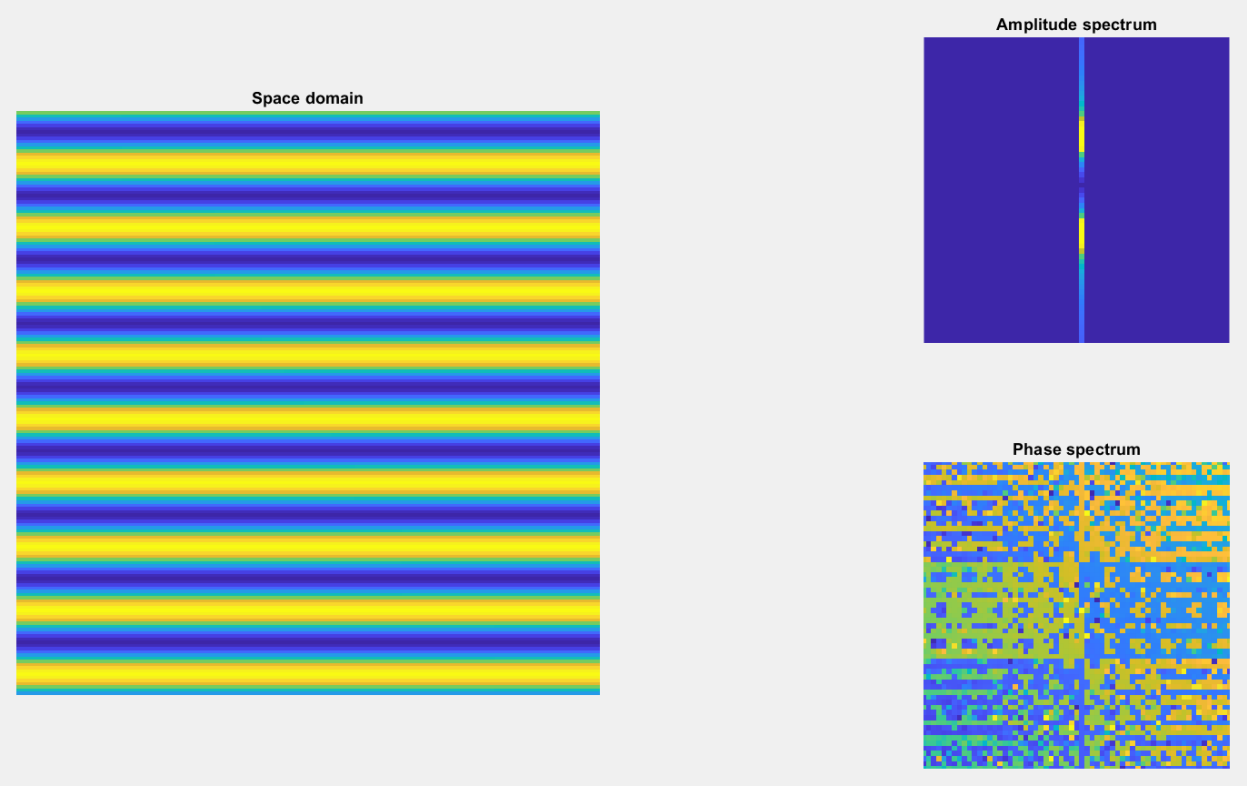
Rearrange so Nyquist frequency is in the outer corners

MATLAB

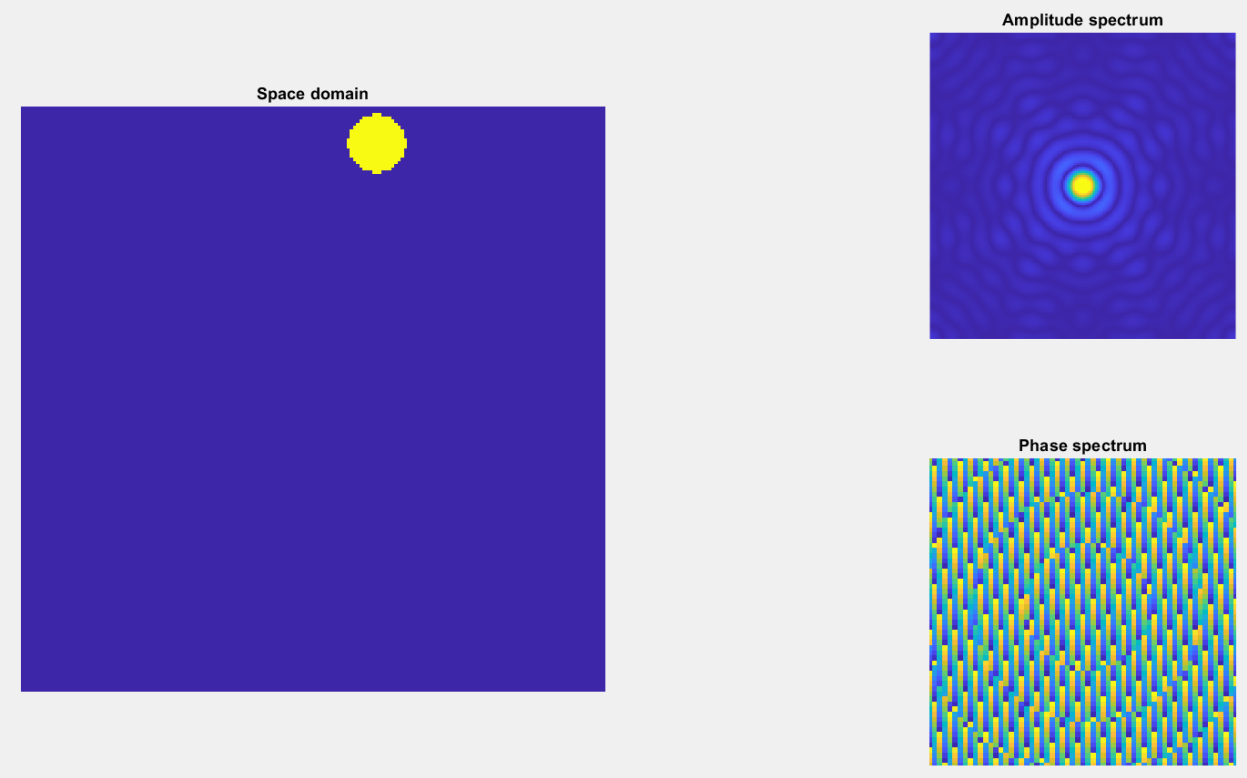
Code: Fourier_2D.m



Watch the movie as frequency changes in code lines 9 to 66



Watch the movie as phase changes in code lines 67 to 122



Watch the movie as ball moves in code lines 123 to 175. Amplitude spectrum remains constant while phase spectrum changes as ball moves.

SECTION 9: APPLICATIONS OF THE FOURIER TRANSFORM

47. RHYTHMICITY IN WALKING (GAIT)

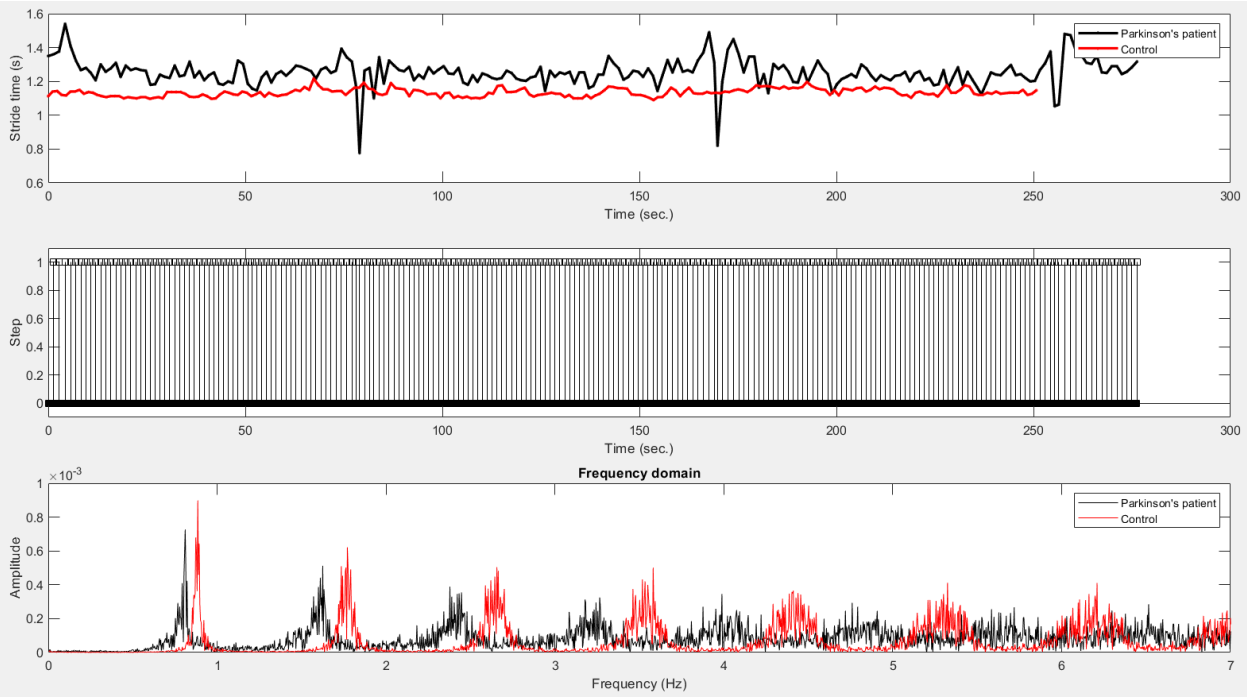
MATLAB

Code: Fourier_apps.m

Lines of code from 1 to 79

Data: gait.mat

```
% SOURCES:
%  Data downloaded from https://physionet.org/physiobank/database/gaitdb/
%   Parkinson's patient data is pd1-si.txt
%   Young control data is y1-23.si.txt
```



48. RHYTHMICITY IN ELECTRICAL BRAIN WAVES

10 Hz alpha wave demonstration

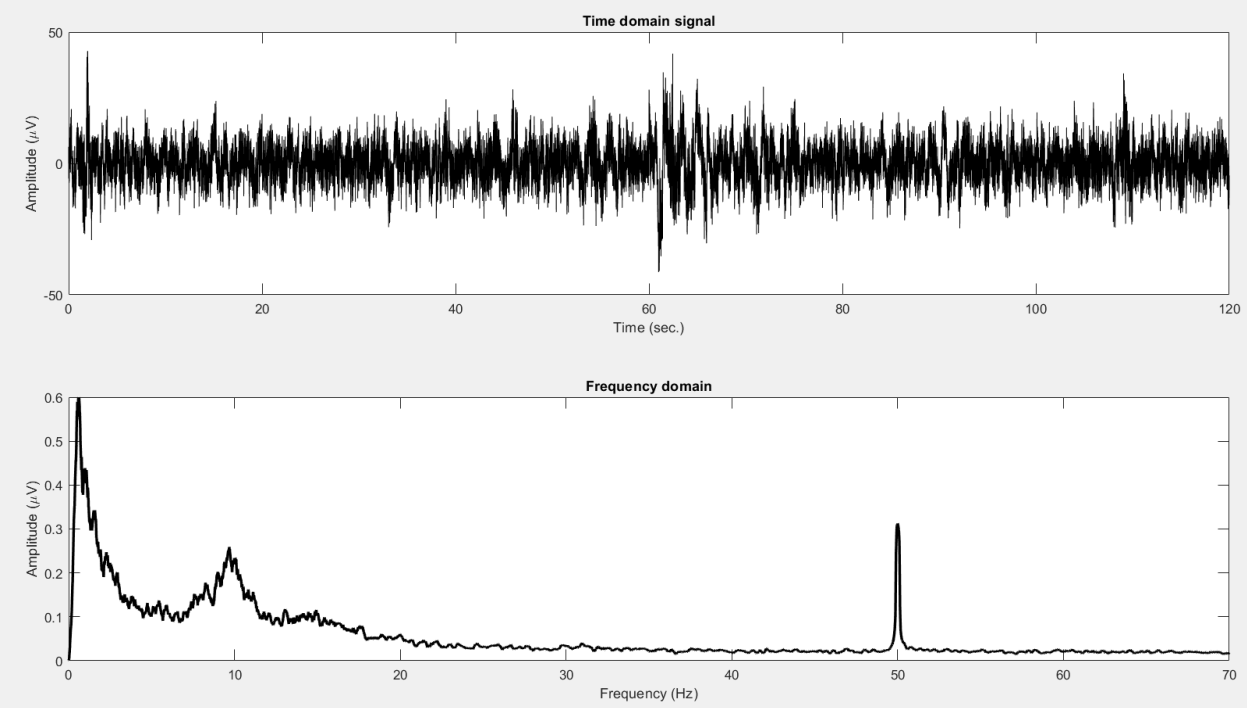
1/f decrease in power / amplitude as frequency goes higher – a fractal type of biological feature.

MATLAB

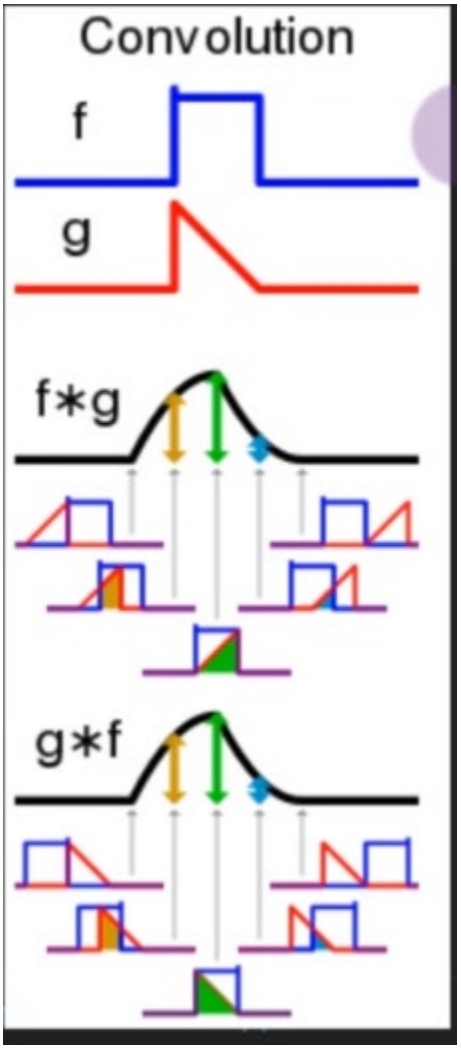
Lines of code from 80 to 122

Data: EEGrestingState.mat

Uses smooth function from the curve fitting toolbox

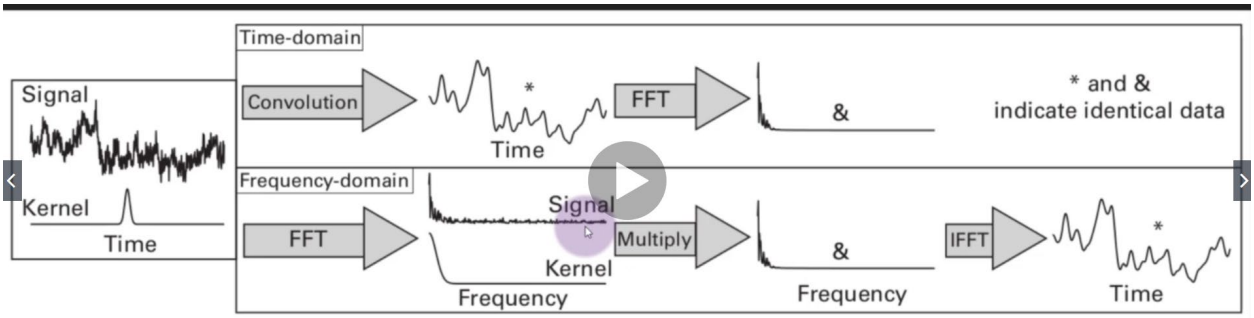


49. TIME SERIES CONVOLUTION



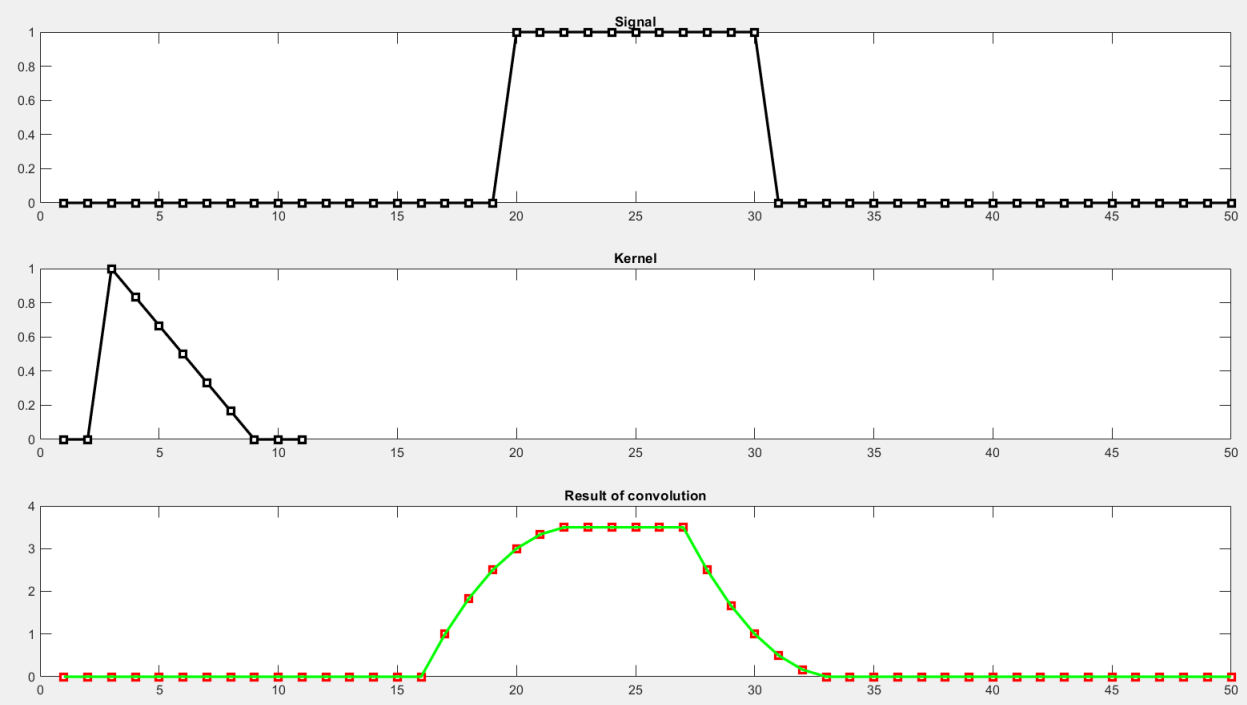
f is the signal; g is the kernel. Flip the kernel then slide the kernel along the signal. At each step compute the dot product. Reference: https://commons.wikimedia.org/w/index.php?curid=20206883

Convolution in the frequency domain is an fft multiplication of both the signal and the kernel, instead of a dot product.

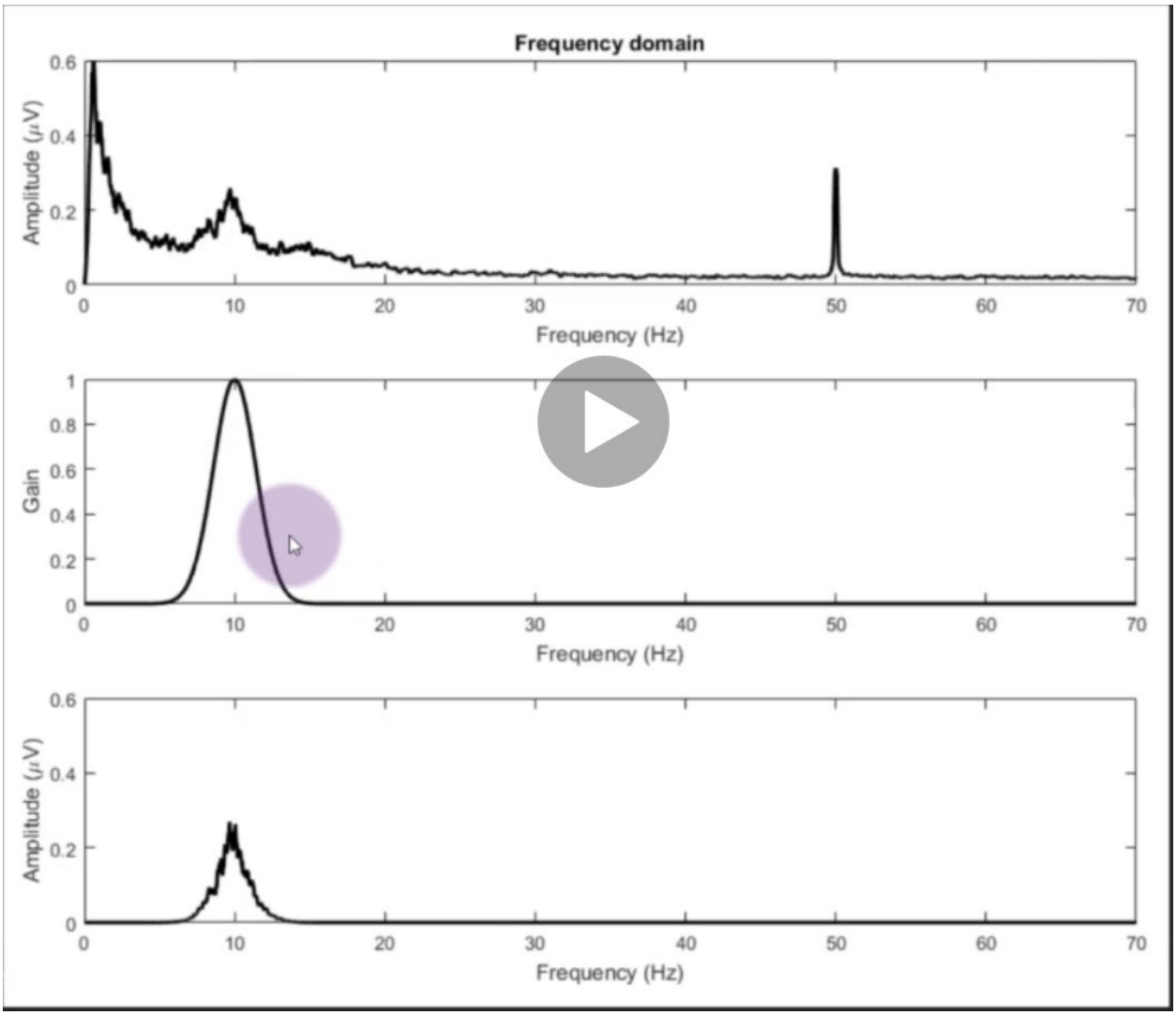


MATLAB

Lines of code: 124 to 181



50. NARROWBAND TEMPORAL FILTERING

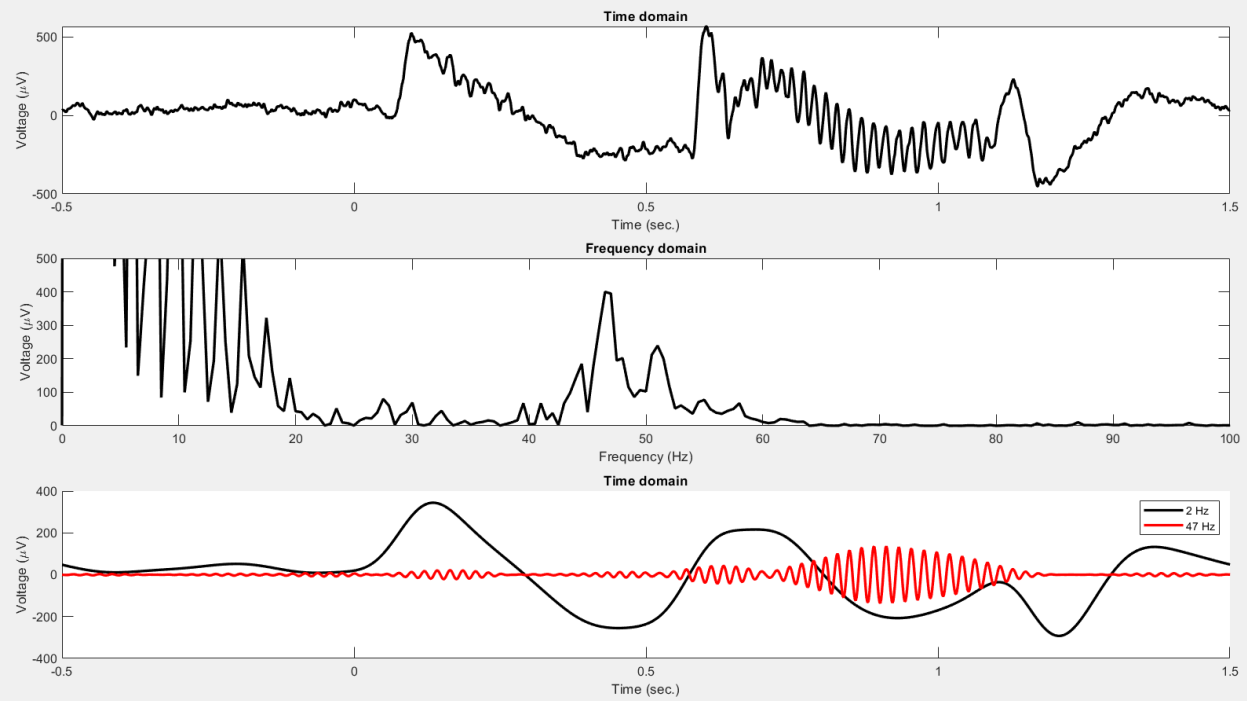


Use the smooth kernel centered around the 10 Hz frequency of interest in this case. The smooth taper to the left and right ends prevents ringing effects near singularities (edges).

MATLAB

Lines of code: from 182 to 252

Data: braindata.mat



A visual image was shown to the subject at time 0 and another image at time 0.5. Brain signal measures above the visual cortex on subject's head.

51. 2D IMAGE FILTERING

MATLAB

Lines of code from 253 to 319

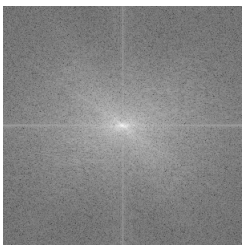
Data file: Lenna.png

Modify code to include my own zscore function to avoid using Statistics and Machine Learning Toolbox

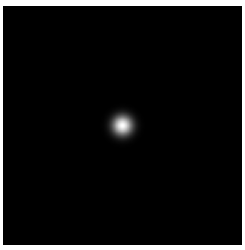
Original image

Low-pass filtered image

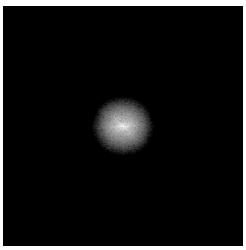
Amplitude spectrum



Gaussian (2D gain function)



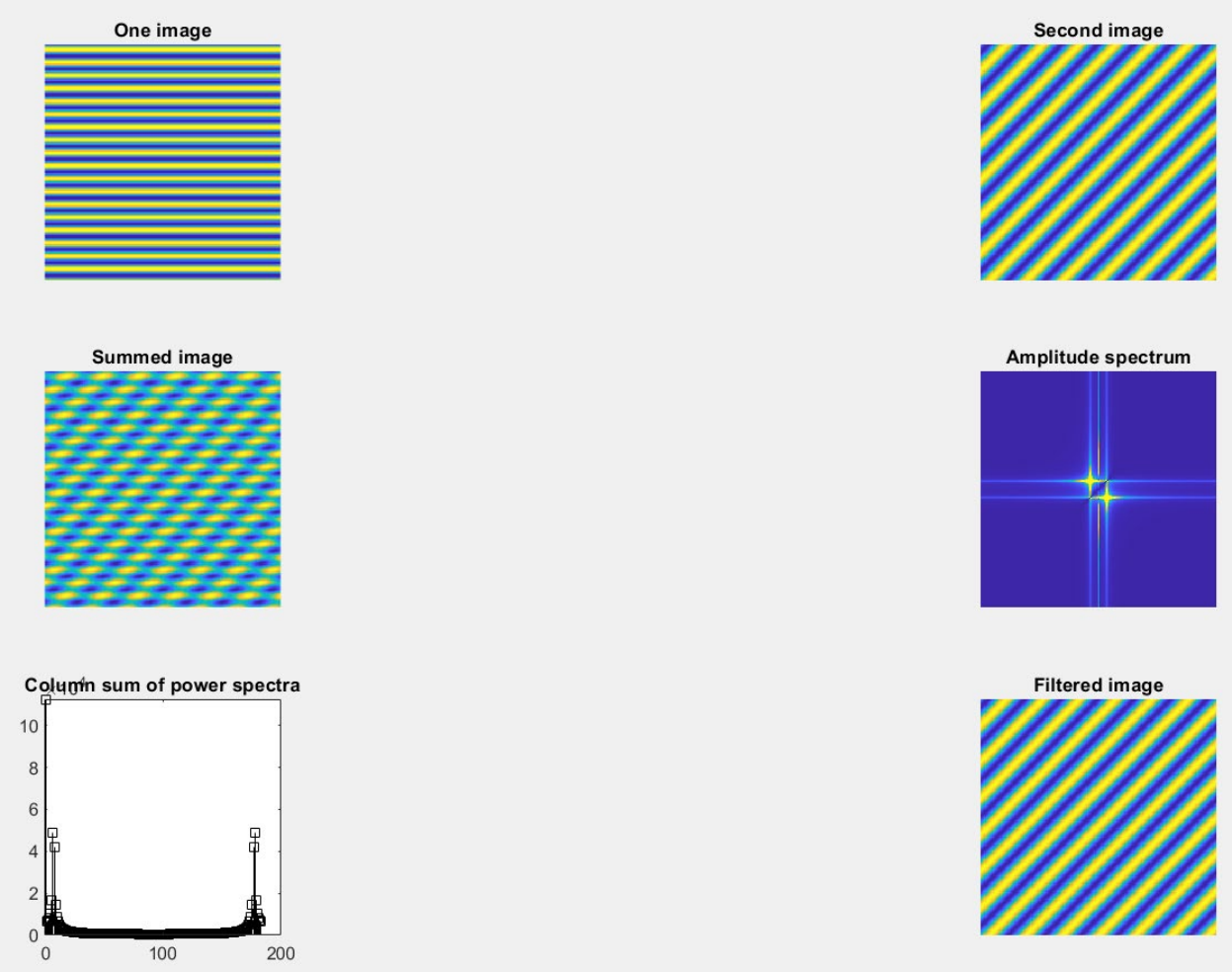
Modulated spectrum



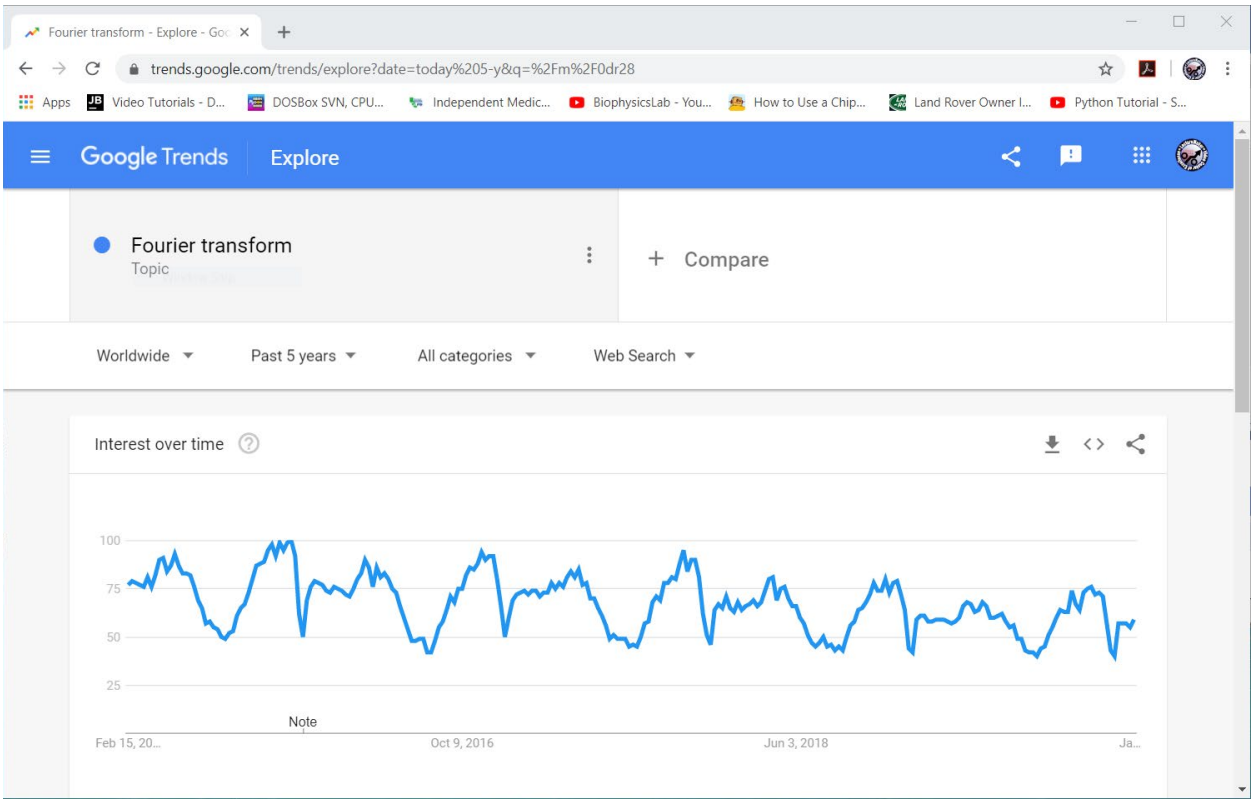
52. IMAGE NARROWBAND FILTERING

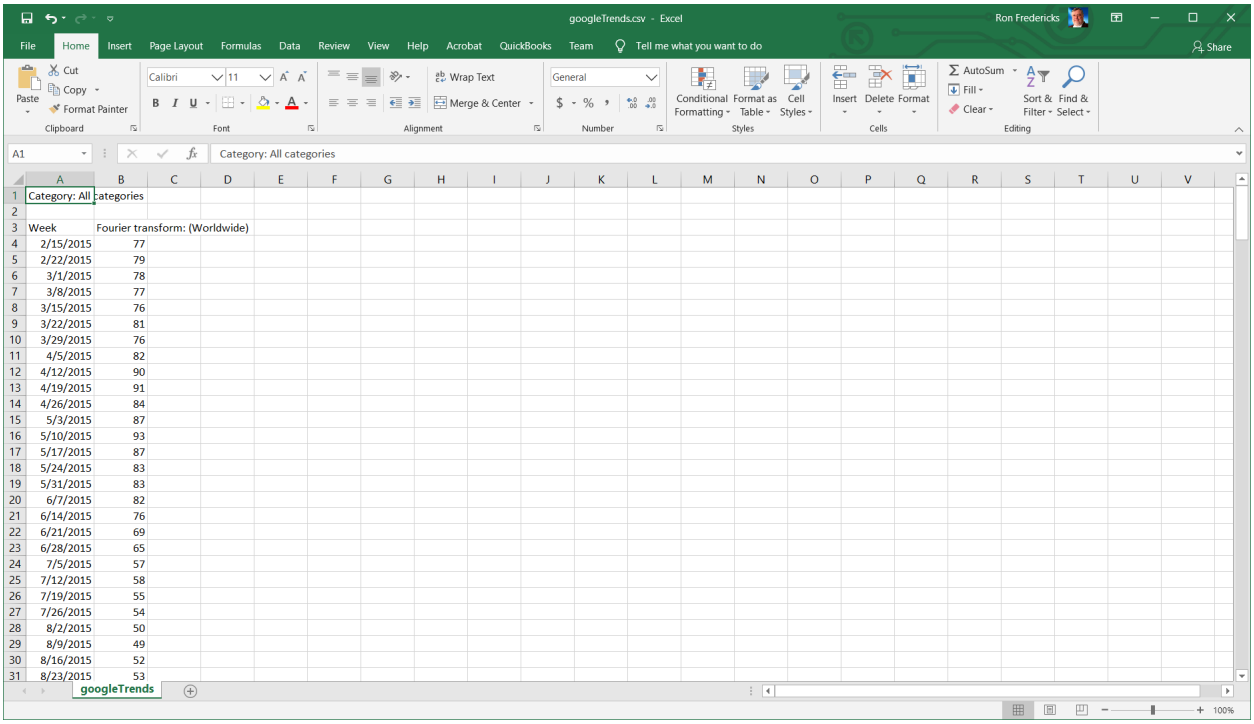
MATLAB

Lines of code: 319 to 396

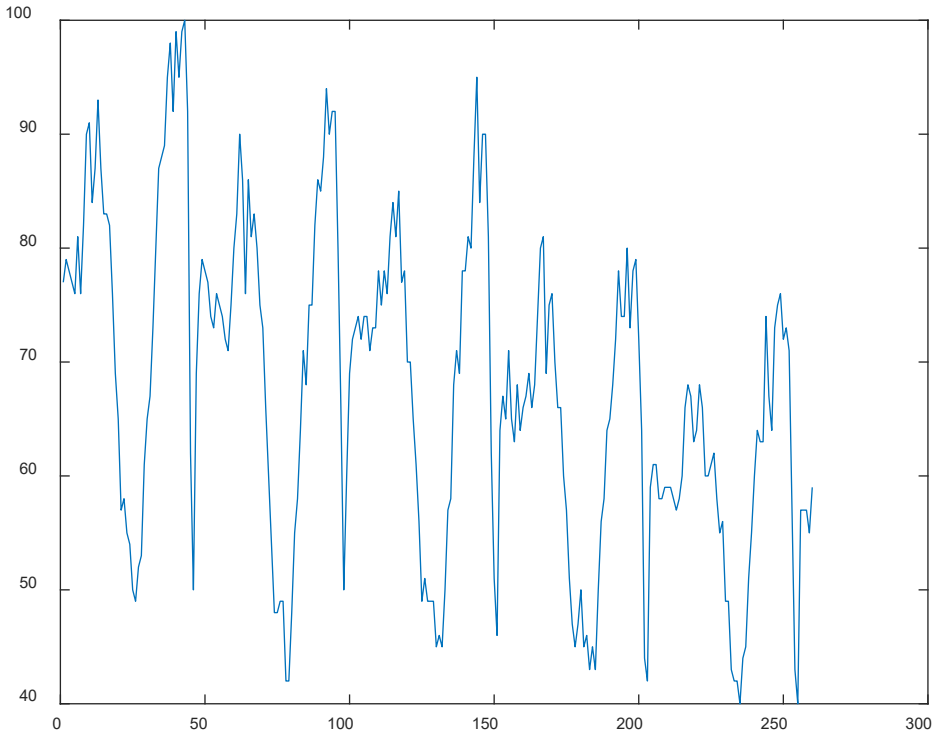


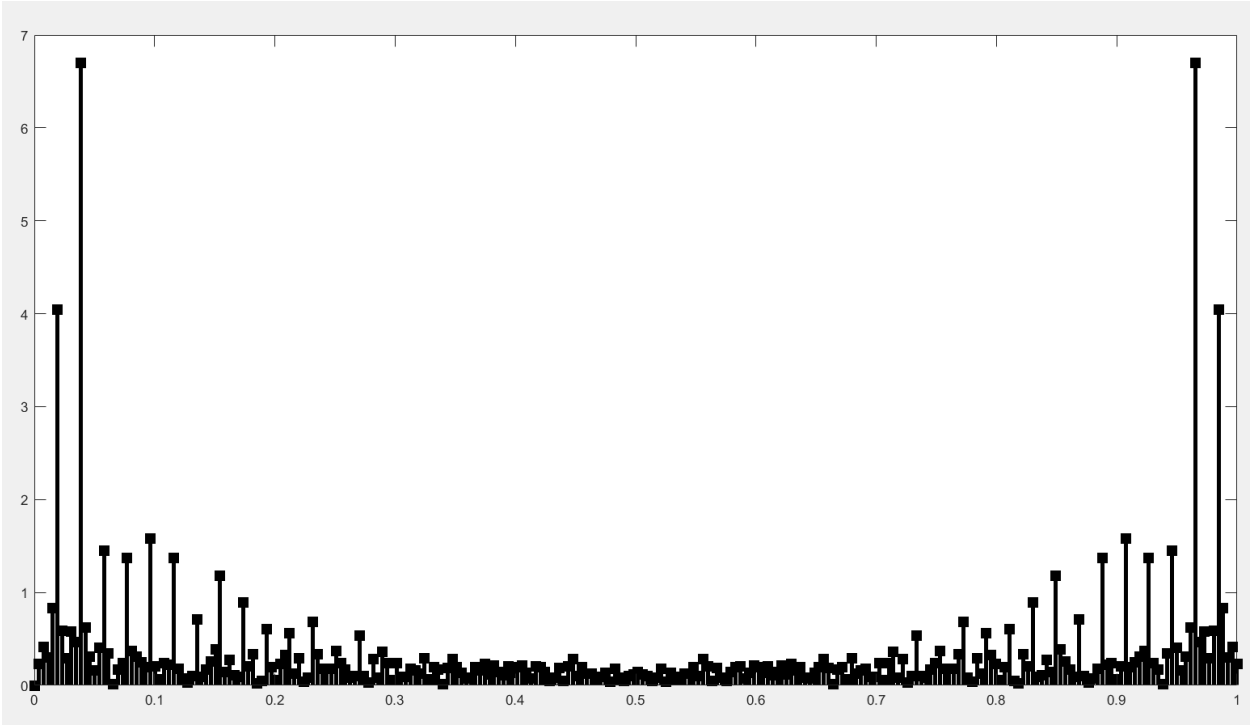
53. REAL DATA FROM TRENDS.GOOGLE.COM!

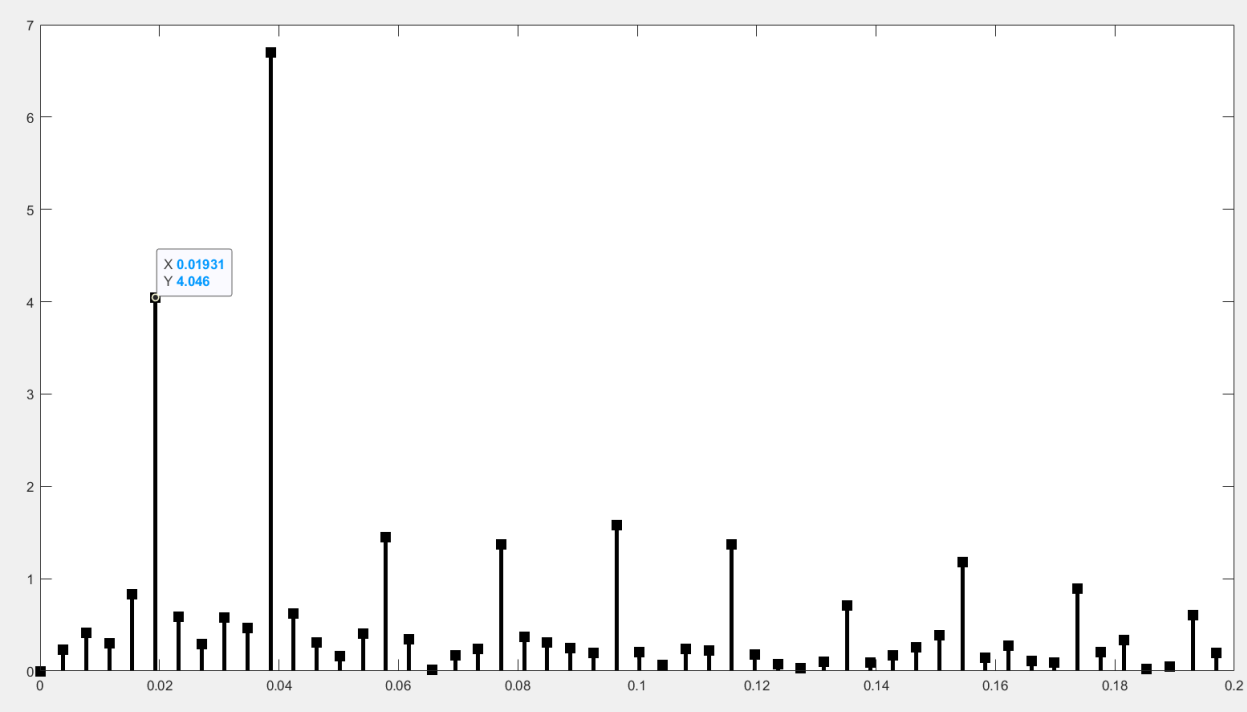




Category: All categories	
Week	Fourier transform: (Worldwide)
2/15/2015	77
2/22/2015	79
3/1/2015	78
3/8/2015	77
3/15/2015	76
3/22/2015	81
3/29/2015	76
4/5/2015	82
4/12/2015	90
4/19/2015	91
4/26/2015	84
5/3/2015	87
5/10/2015	93
5/17/2015	87
5/24/2015	83
5/31/2015	83
6/7/2015	82
6/14/2015	76
6/21/2015	69
6/28/2015	65
7/5/2015	57
7/12/2015	58
7/19/2015	55
7/26/2015	54
8/2/2015	50
8/9/2015	49
8/16/2015	52
8/23/2015	53


100
90
80
70
60
50
40
0
50
100
150
200
250
300


7
6
5
4
3
2
1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



MATLAB COMMAND WINDOW

```
>> d = [77
79
78
77
76
81
76
82
...
55
59
];
>> plot(d)
>> stem(linspace(0,1,length(d)), abs(fft(detrend(d))/length(d)), 'ks-', 'linew', 3, 'markerfacecolor', 'k')
>> set(gca,'xlim',[0 .2])
>> 1/.01931

ans =

   51.7866
```

NOTES

FFT EXAMPLES

AMPLITUDE SPECTRUM

```
srate = 1000;
time  = 0:1/srate:2;
npnts = length(time);

% signal
signal = 2*sin(2*pi*6*time);

% Fourier spectrum with normalization (/npnts)
signalX = fft(signal)/npnts;
hz = linspace(0,srate/2,floor(npnts/2)+1);

% amplitude with normalization (2*), ignoring dc normalization since it is 0
ampl = 2*abs(signalX(1:length(hz)));

% amplitude with normalization (2*), managing dc normalization
% ampl = abs(signalX(1:length(hz)));
% ampl(2:length(hz)); = 2*ampl(2:length(hz));
```

POWER SPECTRUM

```
%% Power spectrum for signal with phase reversal
srate = 1000;
time = 0:1/srate:1-1/srate; % time, for creating half the signal
n = 2*length(time)-1;

signal = [ sin(2*pi*10*time) -sin(2*pi*10*time) ]; % * (-1) has effect of phase reversal
signalAmp = (2*abs( fft(signal)/n )).^2;
hz = linspace(0,srate/2,floor(n/2)+1);

figure(1), clf
plot(hz,signalAmp(1:length(hz)),'ks-','linew',2,'markerfacecolor','w')
set(gca,'xlim',[0 20])
xlabel('Frequency (Hz)'), ylabel('Power')
title('Frequency domain')
```

POWER SPECTRUM IN DECIBELS

```
Fs=10;
 N=length(x);
 fx=fft(x)/N;
 f=(0:N-1)*Fs/N;
 figure, plot(f(1:end/2),10*log10(abs(fx(1:end/2))))
```

INVERSE FFT

```
%% frequency spectrum with a spike
% Ref: Fourier_aliasStation.m
```

```
figure(1), clf

fspect = zeros(300,1);
fspect(10) = 1;

% plot amplitude spectrum
subplot(211)
stem(fspect,'ks-','linew',2)
xlabel('Frequency (indices)')
ylabel('Amplitude')
title('Frequency domain')

% plot time domain signal

% time-domain signal via iFFT
td_sig = real(ifft(fspect)) * length(fspect);

subplot(212)
plot(td_sig,'k','linew',2)
xlabel('Time (indices)')
ylabel('Amplitude')
title('Time domain')
```

SINC FUNCTION AND WHITTAKER-SHANNON INTERPOLATION FORMULA

- https://en.wikipedia.org/wiki/Whittaker%E2%80%93Shannon_interpolation_formula
- https://en.wikipedia.org/wiki/Sinc_function

MATLAB COMMANDS

abs: Absolute value and complex magnitude. Useful in getting magnitude from a complex fft result.

angle: returns the phase angle in the interval [-π,π] for each element of a complex array.

cat(dim, A, B): concatenates B to the end of A along dimension dim when A and B have compatible sizes (the lengths of the dimensions match except for the operating dimension dim).

colorbar: displays a vertical colorbar to the right of the current axes or chart. Colorbars display the current colormap and indicate the mapping of data values into the colormap.

conv(signal, kernel, ‘same’): convolution function

diff: Differences and approximate derivatives.

double: double(X) converts the values in X to double precision.

fft(signal) or fft(signal, nConv): Fourier transform.

fft2: 2-D fast Fourier transform.

fftshift: Shift zero-frequency component to center of spectrum. Used in image fft processing after an fft2

hilbert: Discrete-time analytic signal ($x=x_r+jx_i$) using Hilbert transform from a real data sequence.

ifft: Inverse Fourier transform.

Ifft2: 2-D inverse fast Fourier transform.

imag: Imaginary part of complex number.

imagesc: Display image with scaled colors.

imread('Lenna.png'): Read image from graphics file.

linspace:

- y = linspace(x1,x2) returns a row vector of 100 evenly spaced points between x1 and x2.
- y = linspace(x1,x2,n) generates n points. The spacing between the points is (x2-x1)/(n-1).

log: Natural logarithm. Useful for display of content that has extreme values.

mean(lenna,3): returns the mean along dimension dim.

ndgrid: Rectangular grid in N-D space.

nextpow2(A): returns the exponents for the smallest powers of two that satisfy $2^p \geq |A|$.

real: Real part of complex number.

repmat: Repeat copies of array

smooth: from curve fitting toolbox, smooth(y) smooths the response data in column vector y using a moving average filter.

tic, toc: Measure performance using stopwatch timer.

unwrap: unwraps the radian phase angles in a vector.

zscore: Statistics and Machine Learning Toolbox. Standardized z-scores.

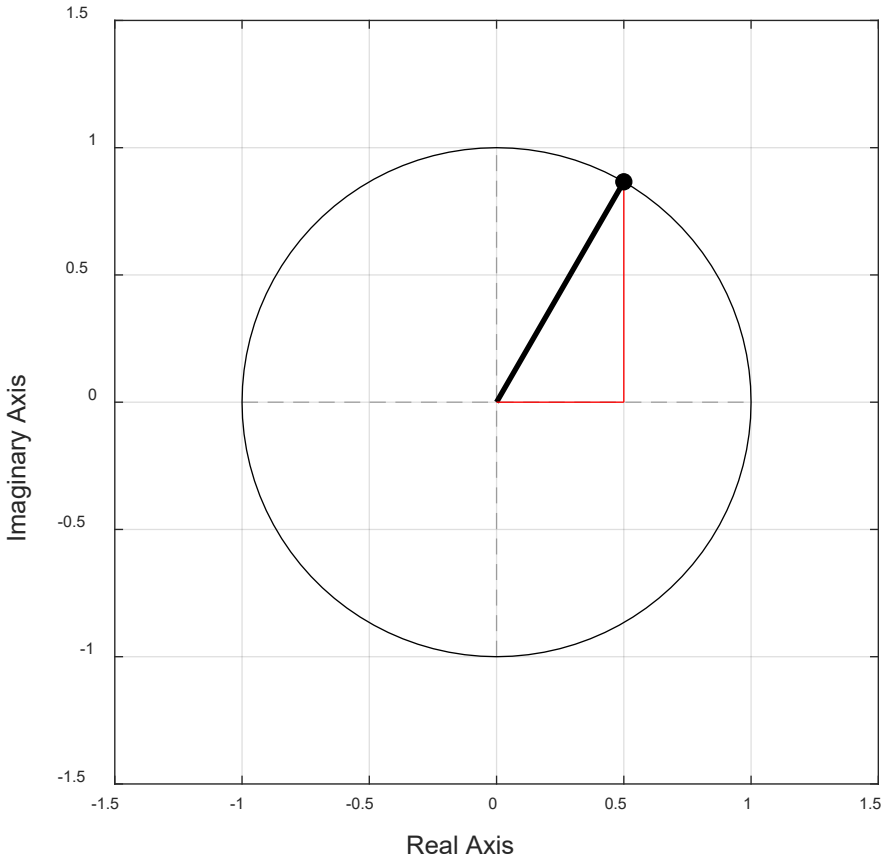
UTILITY CODE

Plot a graph for Euler Function

```
%% Create a 4-quadrant graph with line and circle from origin
```

```
gridSize = [-1 1]; % define x and y grid size
gridAlpha = .3;
figure(1), clf

% grid
h1 = plot(zeros(2,1),gridSize,'k--'); hold on
h1.Color(4) = gridAlpha; % set alpha value
h2 = plot(gridSize,zeros(2,1),'k--');
h2.Color(4) = gridAlpha; % set alpha value

% circle
cdata = circle(0,0,gridSize(2));
cdata.Color = 'k';
% line
ldata = line(0,0,gridSize(2),'degrees',60);
ldata.LineWidth = 2;
ldata.Color = 'k';
% red line from x-axis to dot
plot([ldata.XData(end) ldata.XData(end)],[0 ldata.YData(end)],'r-')
% red line from y-axis to dot
plot([0 ldata.XData(end)],zeros(2,1),'r-')
% dot at end of line
plot(ldata.XData(end),ldata.YData(end),'ko','MarkerFaceColor','k')

% details
xlabel('Real Axis', 'FontSize', 16)
ylabel('Imaginary Axis', 'FontSize', 16)
xlim(gridSize.*1.5)
ylim(gridSize.*1.5)
grid on
axis square

function h = circle(x,y,r)
   % draw a circle
   %   x,y is the start point (such as 0,0 for origin)
   %   r is the radius
   % returns plot handle

   th = 0:pi/50:2*pi;
   xunit = r*cos(th) + x;
   yunit = r*sin(th) + y;
   h = plot(xunit,yunit);
end

function h = line(x,y,L,aunits,angle)
   % draw a line
   %   x,y is the start point (such as 0,0 for origin)
   %   L is the length
   %   aunits is the angle units: 'radians' or 'degrees'
   %   angle is angle (in aunits: degrees or radians)
```

```
    % returns: plot handle

    if aunits == "degrees"
        angle = deg2rad(angle);
    end
    x2=x+(L*cos(angle));
    y2=y+(L*sin(angle));
    h = plot([x x2],[y y2]);
end
```



CODE TO TAKE MORE COURSES

FROM SECTION 2: FOURIER_FOUNDATIONS.M

See code comments for details

REFERENCES

Database and Software for research (gait in this case): https://physionet.org/physiobank/database/gaitdb/