Lecture 5. Fourier transform

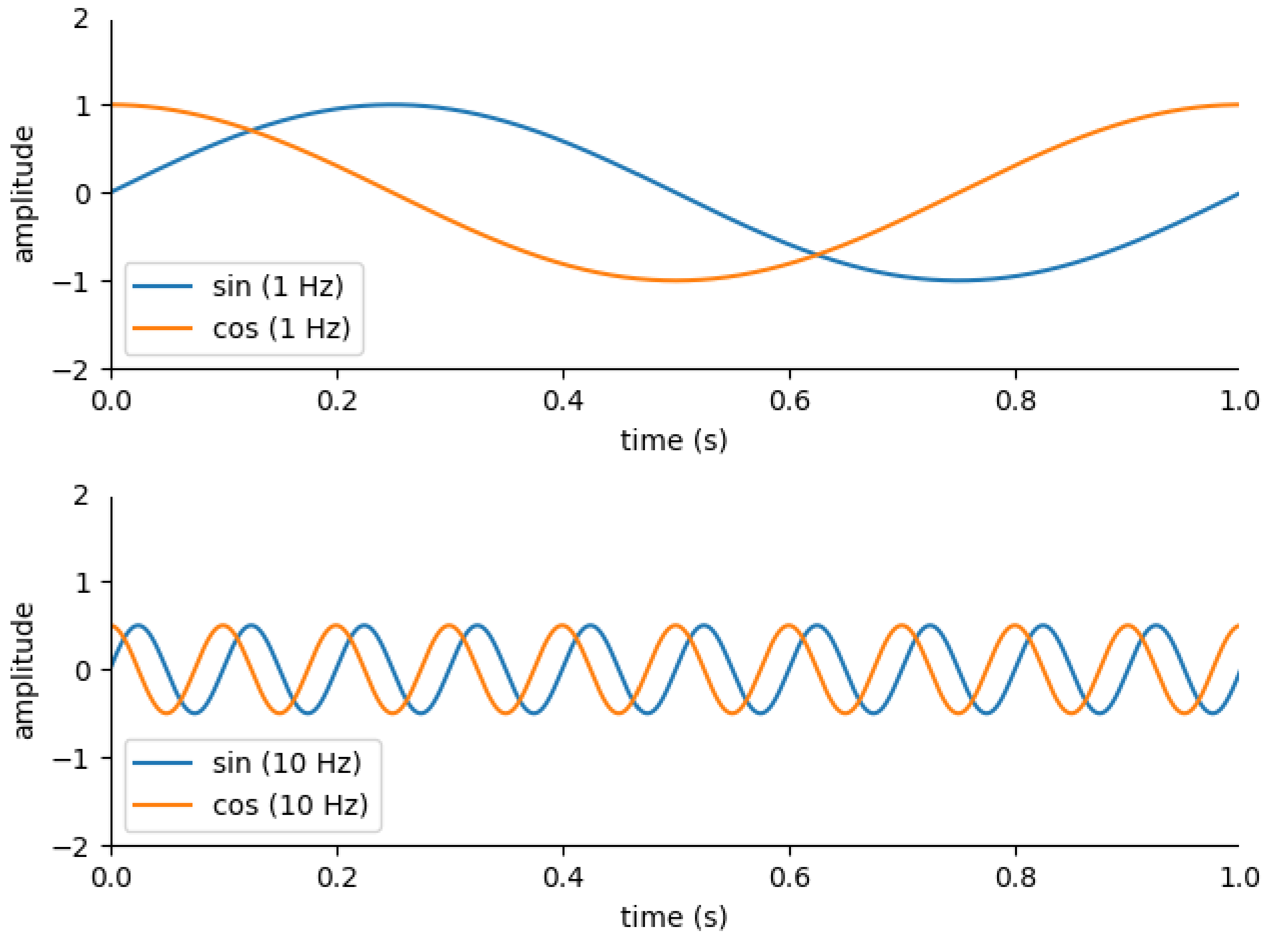
Alexander Zhigalov / Dept. of CS, University of Helsinki and Dept. of NBE, Aalto University

Outline / overview

Section 1. Periodic functions

Periodic functions

```
# sampling parameters
fs = 1000   # sampling rate, in Hz
T  = 1      # duration, in seconds
N  = T * fs # duration, in samples

# time variable
t = np.linspace(0, T, N)

# signal parameters
A = 1 # amplitude
f = 1 # frequency

# sin and cos functions
x = A * np.sin(2 * np.pi * f * t)
y = A * np.cos(2 * np.pi * f * t)
```

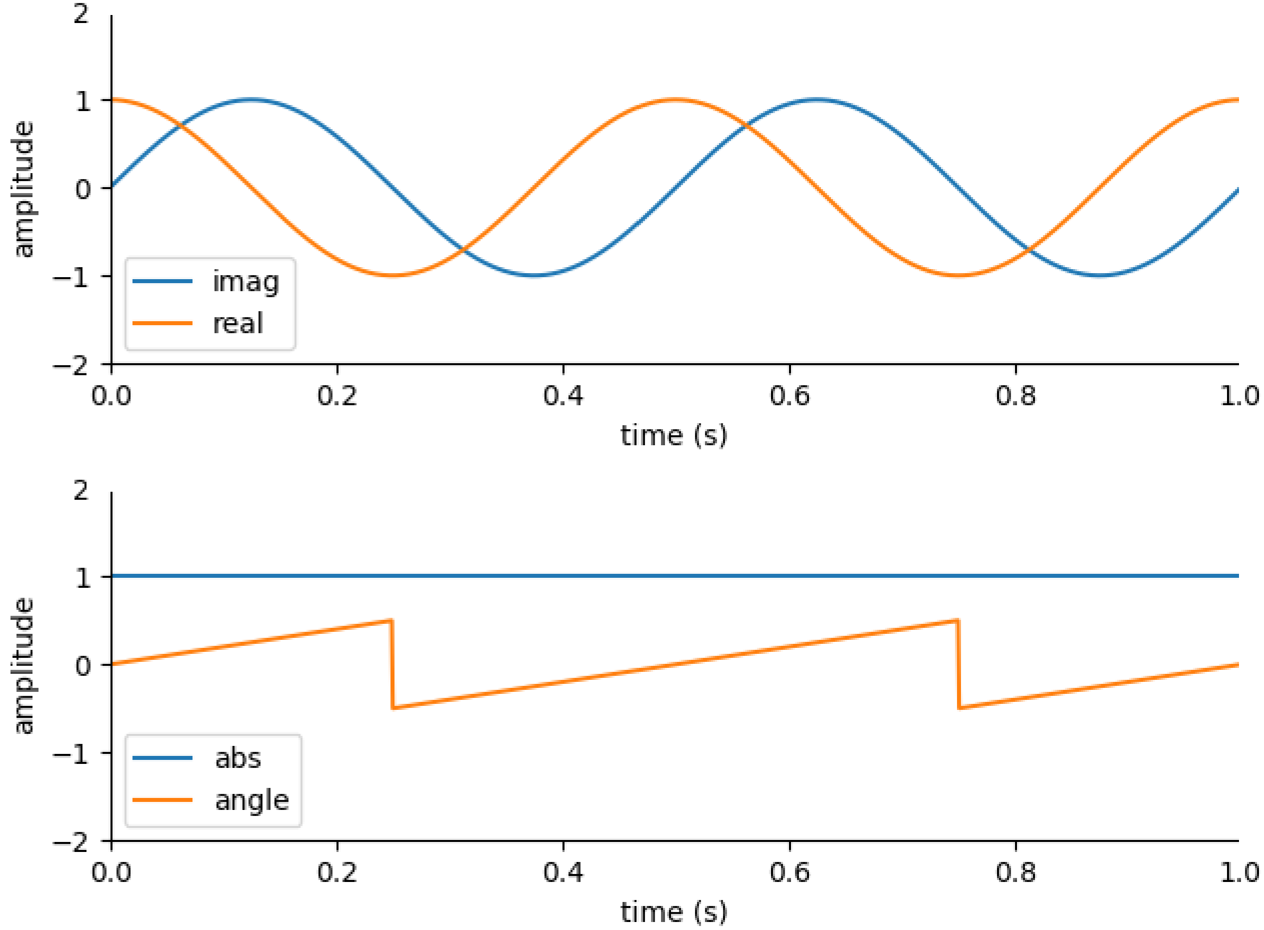


See, “L05_periodic_functions.py”

Complex exponent

exp(**1j** *f * t) = **cos**(f * t) + **1j** * **sin**(f * t)

```
# sampling parameters
fs = 1000   # sampling rate, in Hz
T  = 1      # duration, in seconds
N  = T * fs # duration, in samples

# signal parameters
A = 1 # signal amplitude
f = 2 # signal frequency, in Hz

# time variable
t = np.linspace(0, T, N)

# signal
z = A * np.exp(1j * 2 * np.pi * f * t)

x = np.imag(z)
y = np.real(z)
a = np.abs(z)
p = np.angle(z) / (2 * np.pi)
```

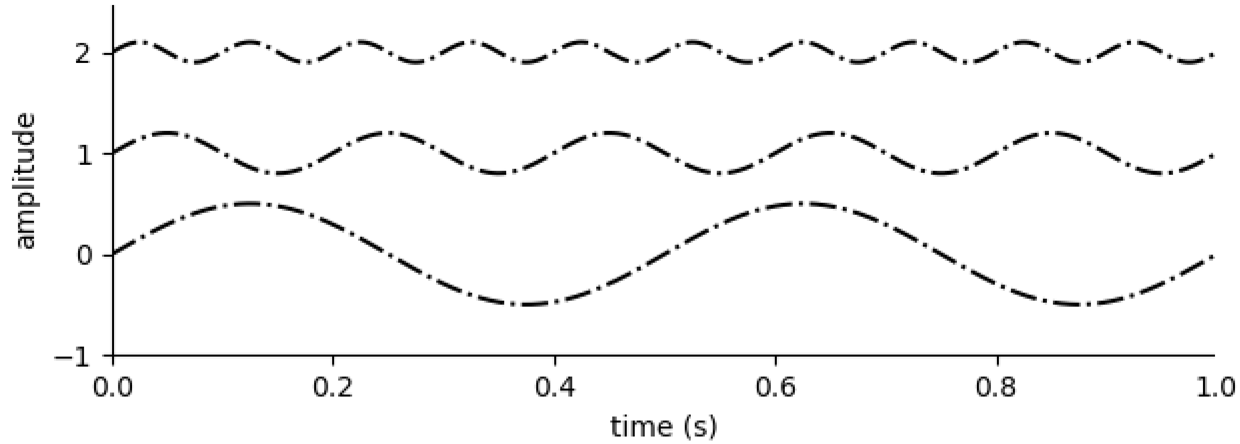


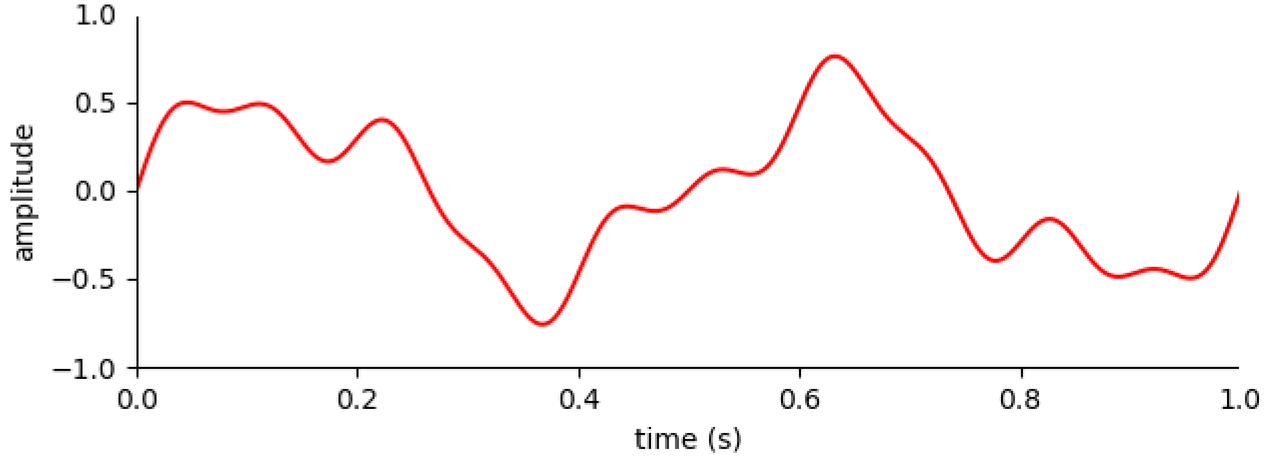
See, “L05_complex_exponent.py”

Sum of periodic signals

Any periodic signal can be represented as a sum of set of oscillating functions, namely sines and cosines.

```
# sampling parameters
fs = 1000   # sampling rate, in Hz
T  = 1      # duration, in seconds
N  = T * fs # duration, in samples

# time variable
t = np.linspace(0, T, N)

# sum of periodic signals
x1 = 0.5 * np.sin(2 * np.pi * 2 * t)
x2 = 0.2 * np.sin(2 * np.pi * 5 * t)
x3 = 0.1 * np.sin(2 * np.pi * 10 * t)
x = x1 + x2 + x3
```



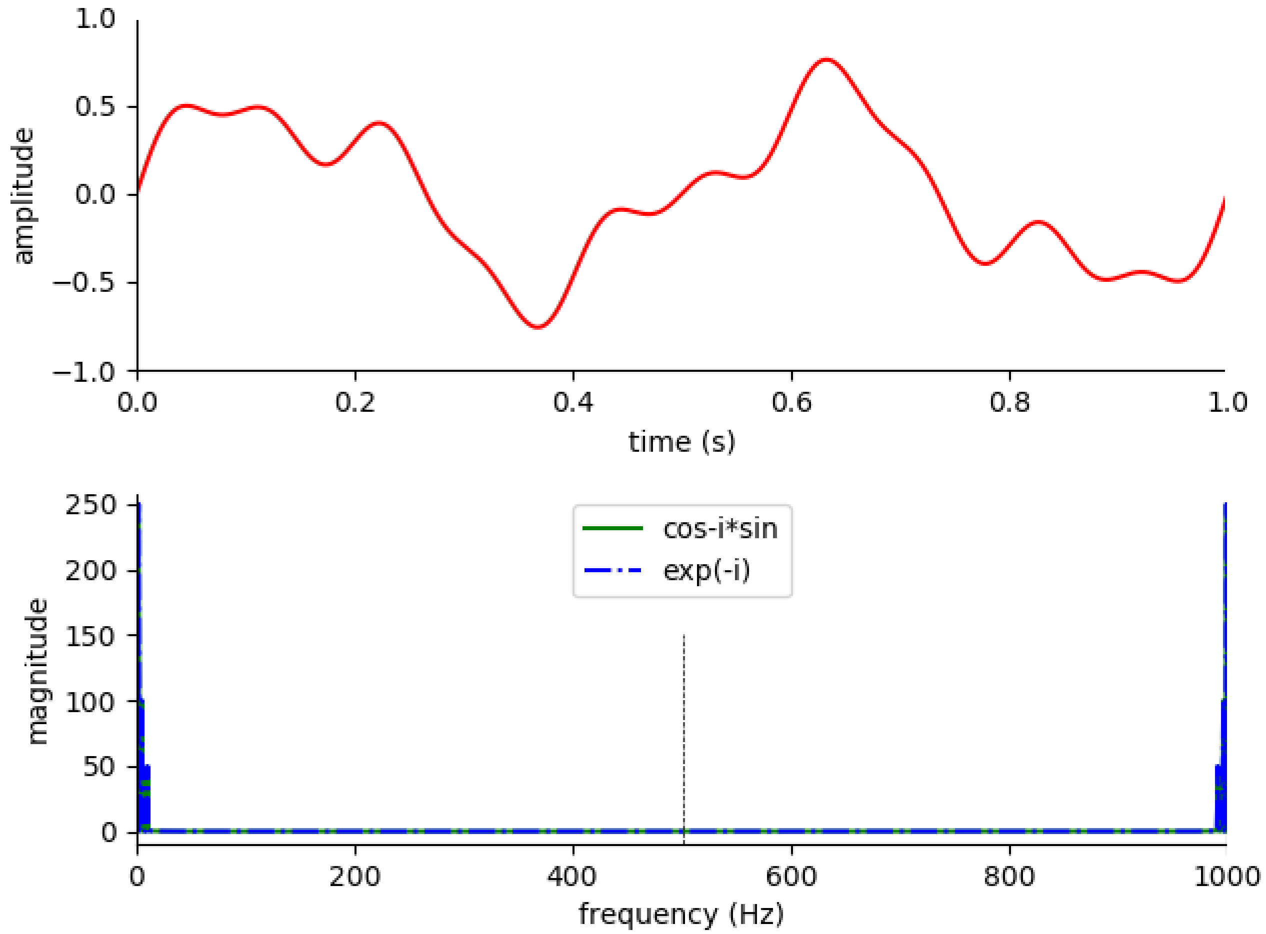


See, “L05_sum_of_sins.py”

Section 2. Discrete Fourier transform

Fourier transform (1/2)

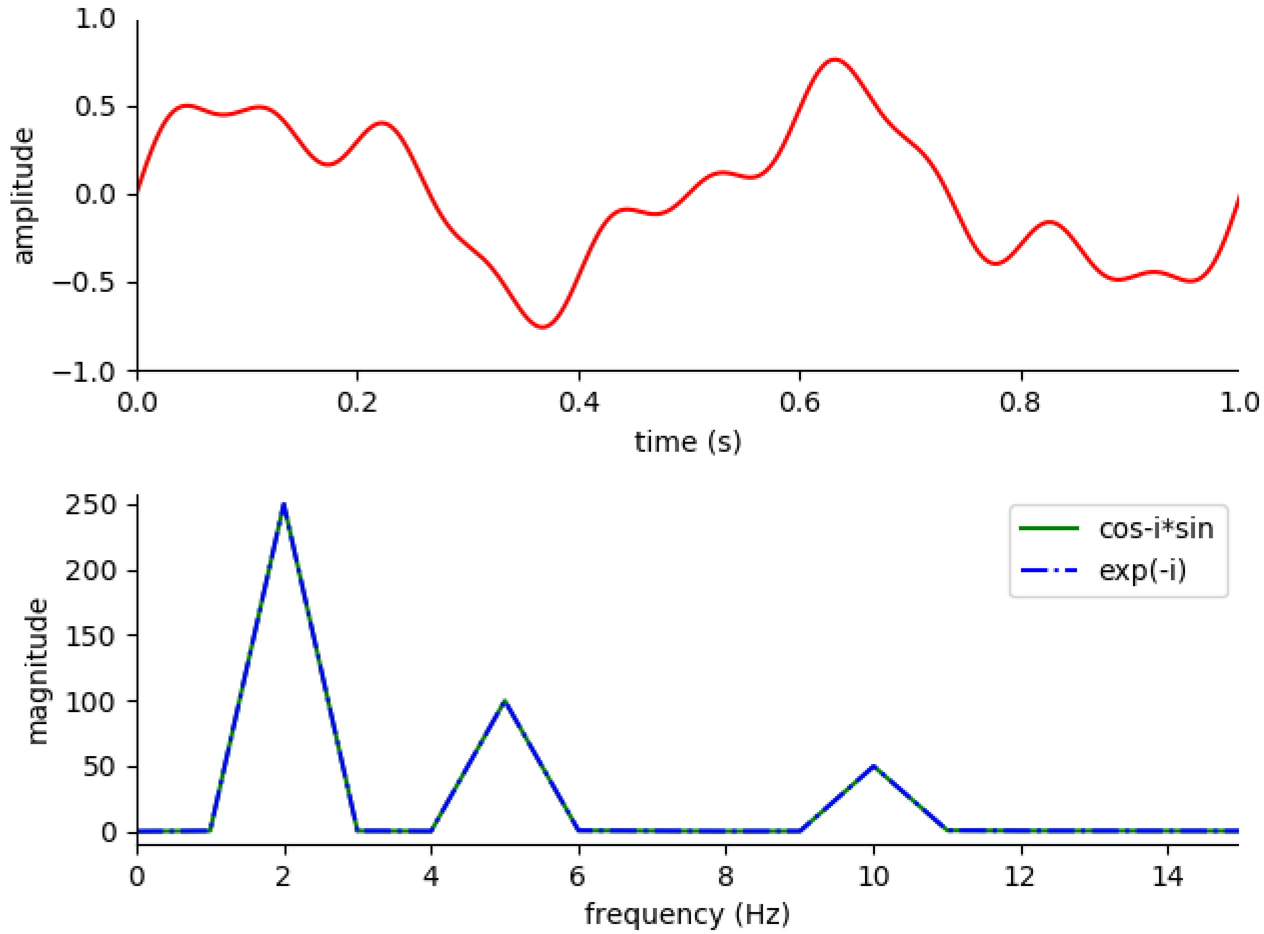
```
# frequency resolution
nFFT = fs # fs / nFFT, in Hz

# time variable
t = np.arange(0, N)

# over frequencies
for k in range(0, nFFT):

  # relative frequency
  f = k / nFFT

  # exp
  y[k] = np.sum(np.exp(-1j * 2 * np.pi * t * f)
         * x)

  # cos + 1i * sin
  u[k] = np.sum(np.cos(2 * np.pi * t * f) * x -
           1j * np.sin(2 * np.pi * t * f) * x)
```



See, “L05_fourier_transform.py”

Fourier transform (2/2)

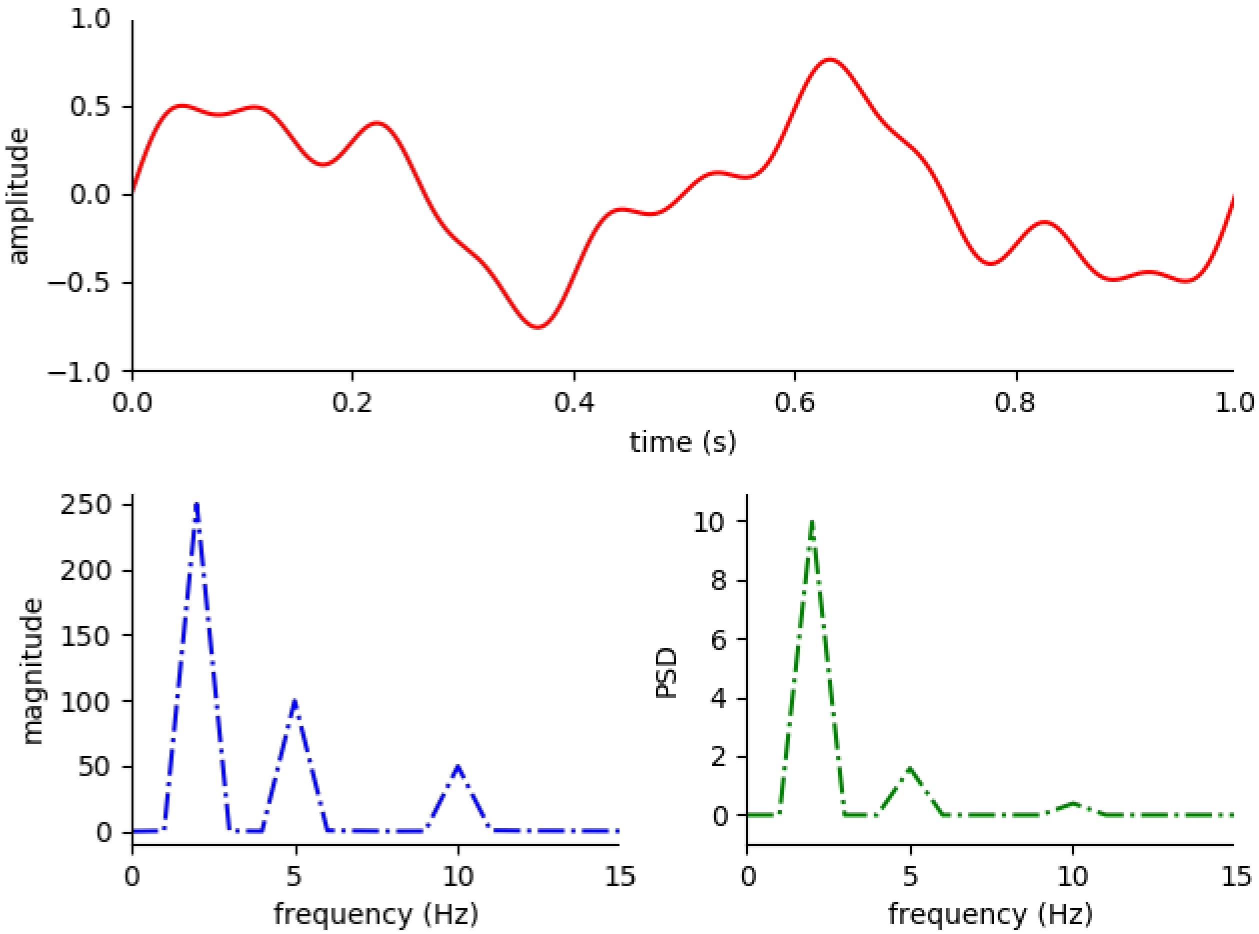
```
# frequency resolution
nFFT = fs # fs / nFFT, in Hz

# time variable
t = np.arange(0, N)

# over frequencies
for k in range(0, nFFT):

  # relative frequency
  f = k / nFFT

  # exp
  y[k] = np.sum(np.exp(-1j * 2 * np.pi * t * f)
         * x)

  # cos + 1i * sin
  u[k] = np.sum(np.cos(2 * np.pi * t * f) * x -
             1j * np.sin(2 * np.pi * t * f) * x)
```



See, “L05_fourier_transform.py”

Power spectral density

```
# signal
x1 = 0.5 * np.sin(2 * np.pi * 2 * t)
x2 = 0.2 * np.sin(2 * np.pi * 5 * t)
x3 = 0.1 * np.sin(2 * np.pi * 10 * t)
x = x1 + x2 + x3

# fourier transform
y = fourier_tansform(x, nFFT)

# magnitude
Y = np.abs(y)

# power spectral density (PSD)
U = (1 / (2 * np.pi * N)) * np.abs(y) ** 2
```

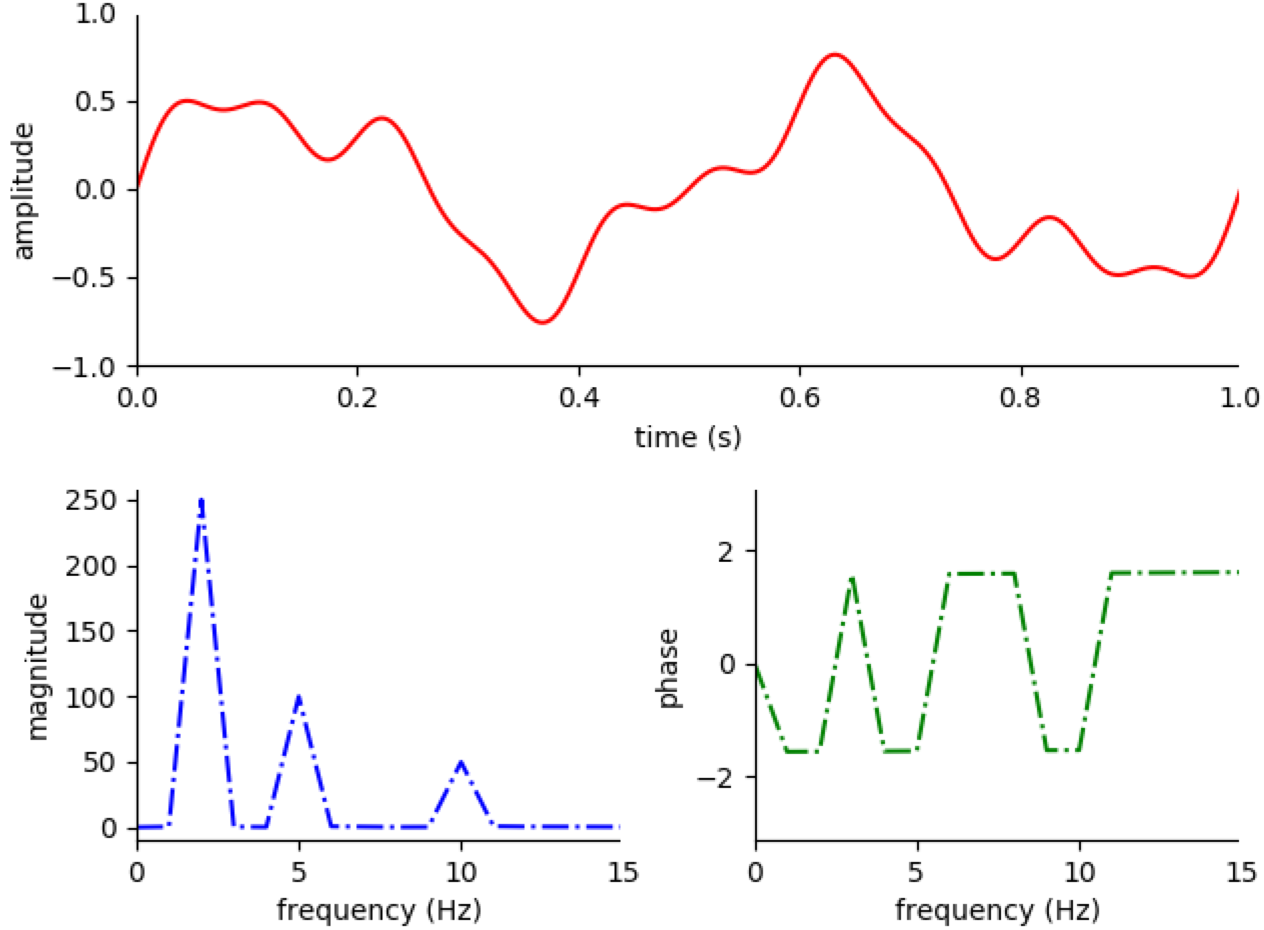


See, “L05_fourier_transform_psd.py”

Fourier transform: Amplitude and Phase spectra (1/3)

What is the outcome of Fourier transform?

```
# signal
x1 = 0.5 * np.sin(2 * np.pi * 2 * t)
x2 = 0.2 * np.sin(2 * np.pi * 5 * t)
x3 = 0.1 * np.sin(2 * np.pi * 10 * t)
x = x1 + x2 + x3

# amplitude spectrum, np.abs(y)
Y = np.sqrt(np.real(y) ** 2 + np.imag(y) ** 2)

# phase spectrum, np.angle(y)
P = np.arctan2(np.imag(y), np.real(y))
```

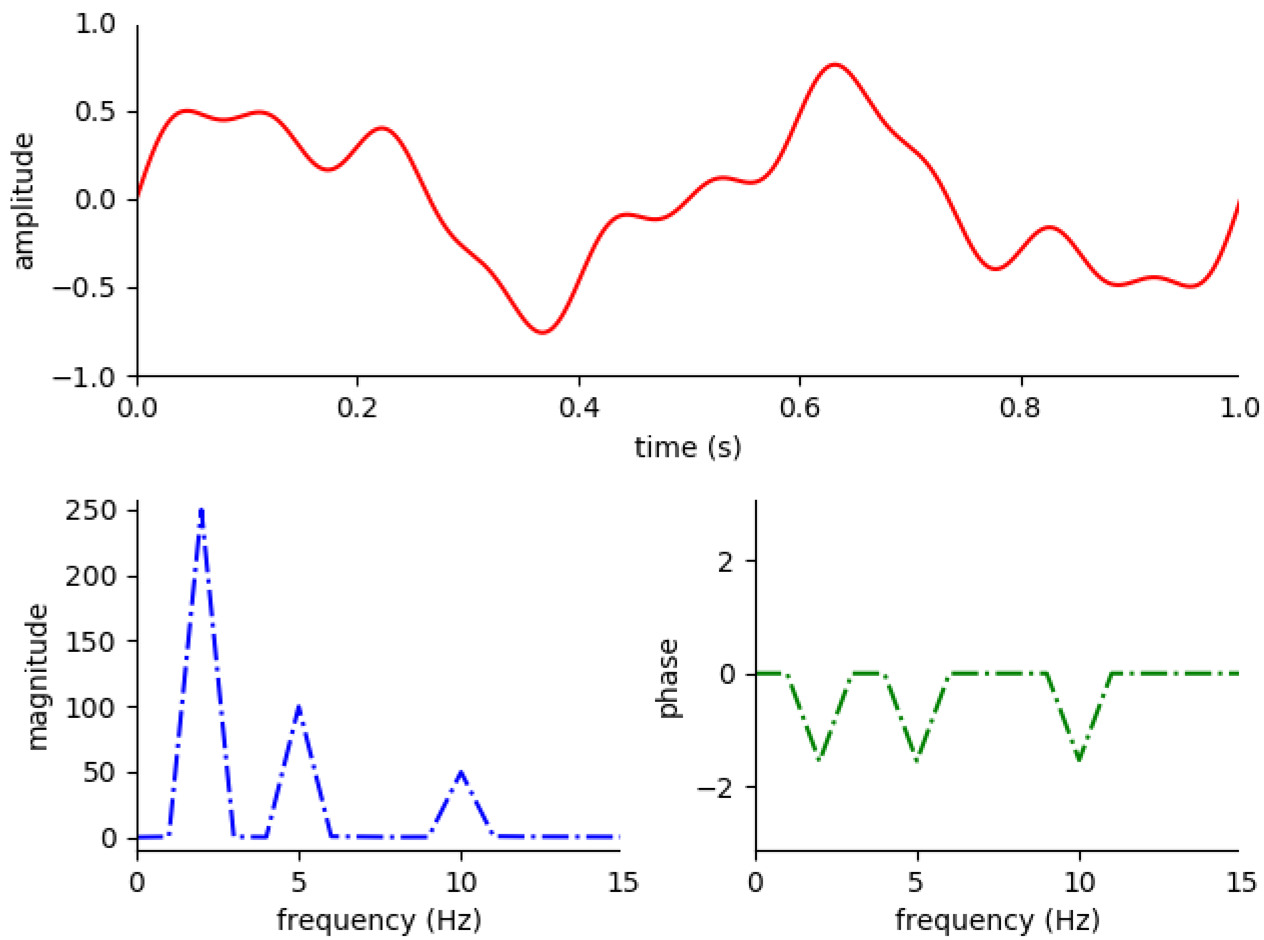


See, “L05_fourier_transform_spectra.py”

Fourier transform: Amplitude and Phase spectra (2/3)

Even a small floating rounding off error amplifies the result and manifest incorrectly as useful phase information.

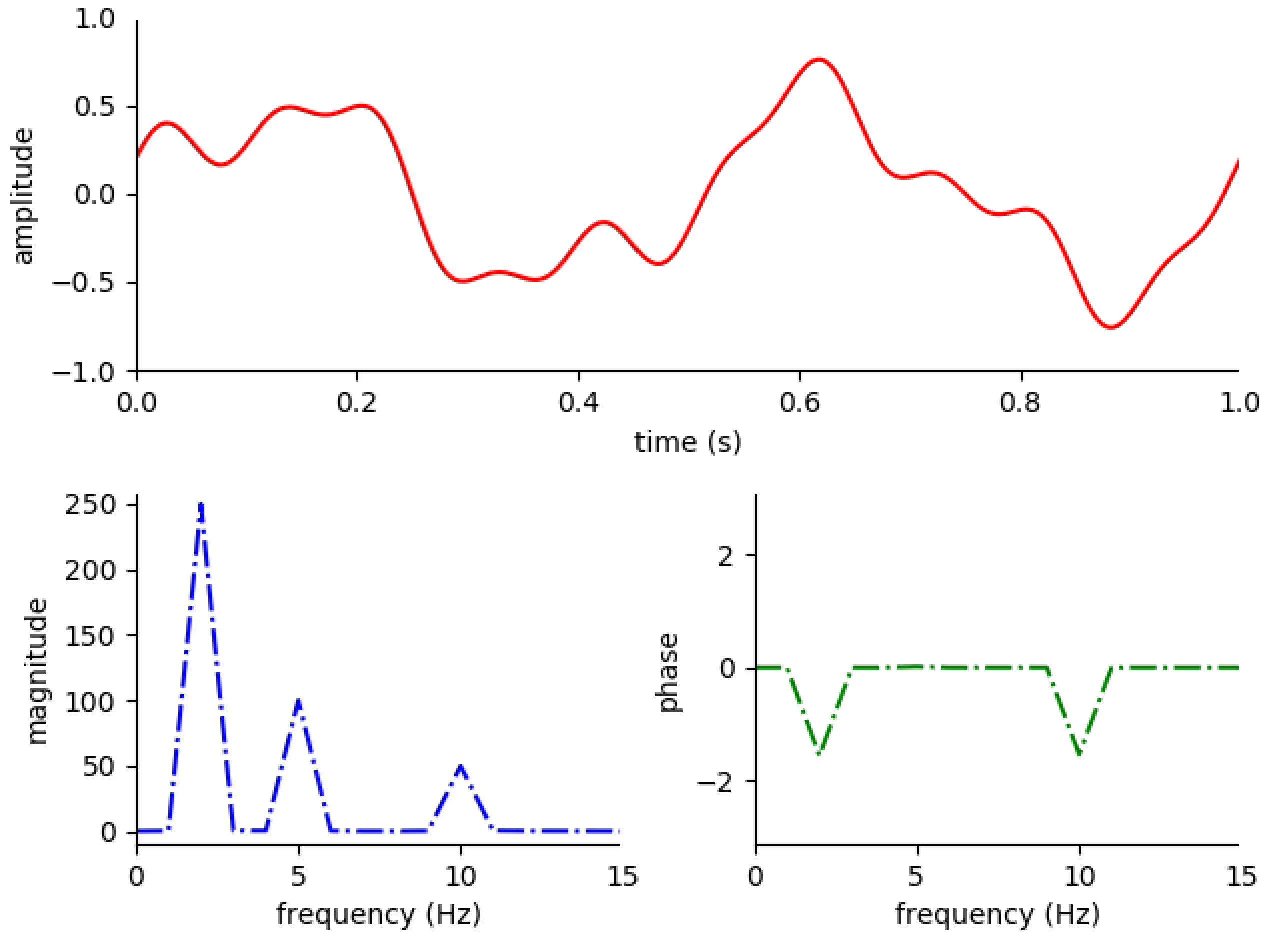
```
# signal
x1 = 0.5 * np.sin(2 * np.pi * 2 * t)
x2 = 0.2 * np.sin(2 * np.pi * 5 * t)
x3 = 0.1 * np.sin(2 * np.pi * 10 * t)
x = x1 + x2 + x3

# amplitude spectrum, np.abs(y)
Y = np.sqrt(np.real(y) ** 2 + np.imag(y) ** 2)

# phase spectrum, np.angle(y)
y[np.abs(y) < 0.9] = 0
P = np.arctan2(np.imag(y), np.real(y))
```



See, "L05_fourier_transform_spectra.py"

Fourier transform: Amplitude and Phase spectra (3/3)

Sine and cosine components have pi/2 shift.



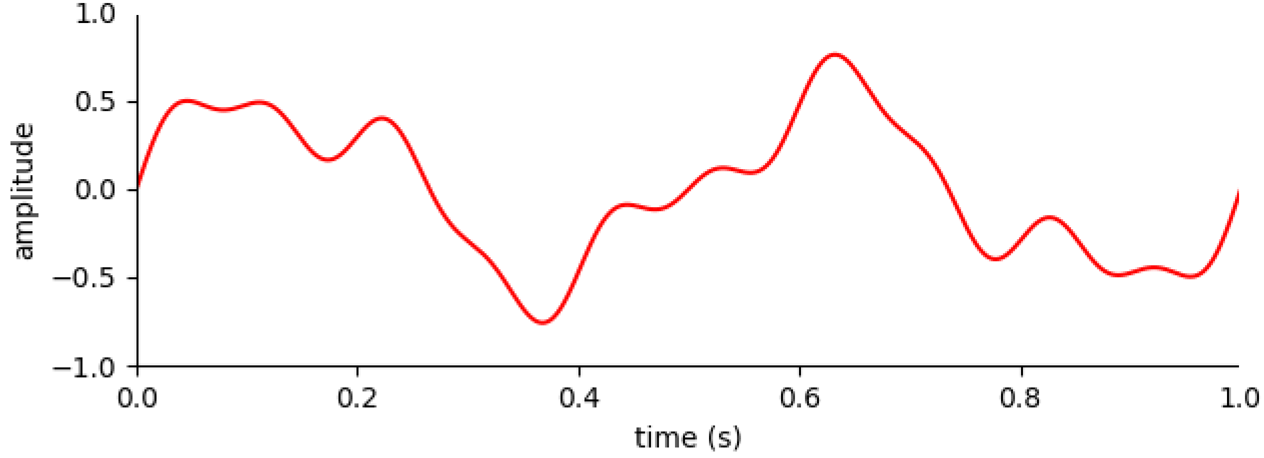

```
# signal
x1 = 0.5 * np.sin(2 * np.pi * 2 * t)
x2 = 0.2 * np.cos(2 * np.pi * 5 * t)
x3 = 0.1 * np.sin(2 * np.pi * 10 * t)
x = x1 + x2 + x3

# amplitude spectrum, np.abs(y)
Y = np.sqrt(np.real(y) ** 2 + np.imag(y) ** 2)

# phase spectrum, np.angle(y)
y[np.abs(y) < 0.9] = 0
P = np.arctan2(np.imag(y), np.real(y))
```

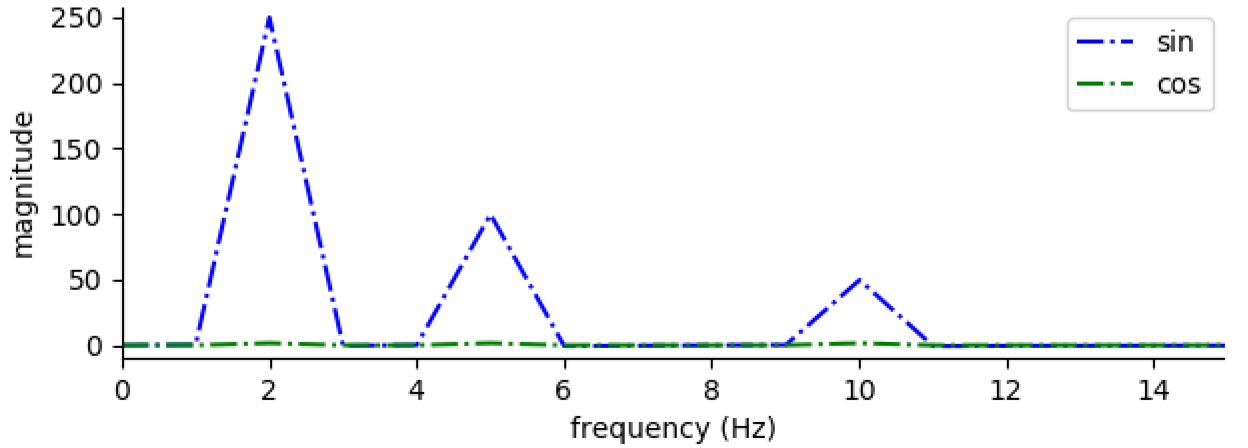
See, “L05_fourier_transform_spectra.py”

Fourier transform sin and cos components (1/2)

What the difference between sine and cosine components?

```
# signal
x1 = 0.5 * np.sin(2 * np.pi * 2 * t)
x2 = 0.2 * np.sin(2 * np.pi * 5 * t)
x3 = 0.1 * np.sin(2 * np.pi * 10 * t)
x = x1 + x2 + x3

# fourier transform
y = fourier_tansform(x, nFFT)

# power spectrum
Y_sin = (-1) * np.imag(y)
Y_cos = np.real(y)
```

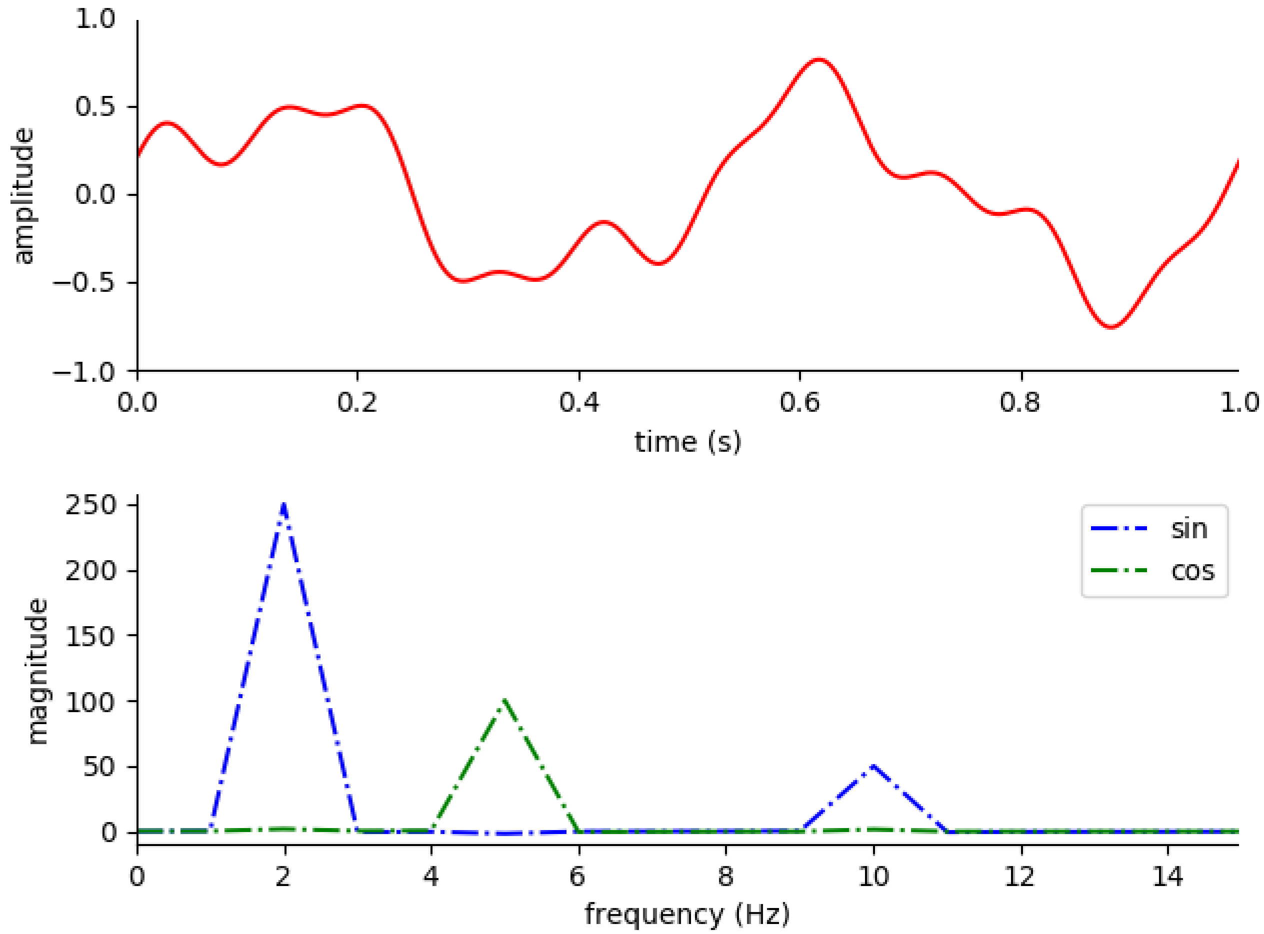




See, “L05_fourier_transform_sin_and_cos.py”

Fourier transform sin and cos components (2/2)

```
# signal
x1 = 0.5 * np.sin(2 * np.pi * 2 * t)
x2 = 0.2 * np.cos(2 * np.pi * 5 * t)
x3 = 0.1 * np.sin(2 * np.pi * 10 * t)
x = x1 + x2 + x3

# fourier transform
y = fourier_tansform(x, nFFT)

# power spectrum
Y_sin = (-1) * np.imag(y)
Y_cos = np.real(y)
```

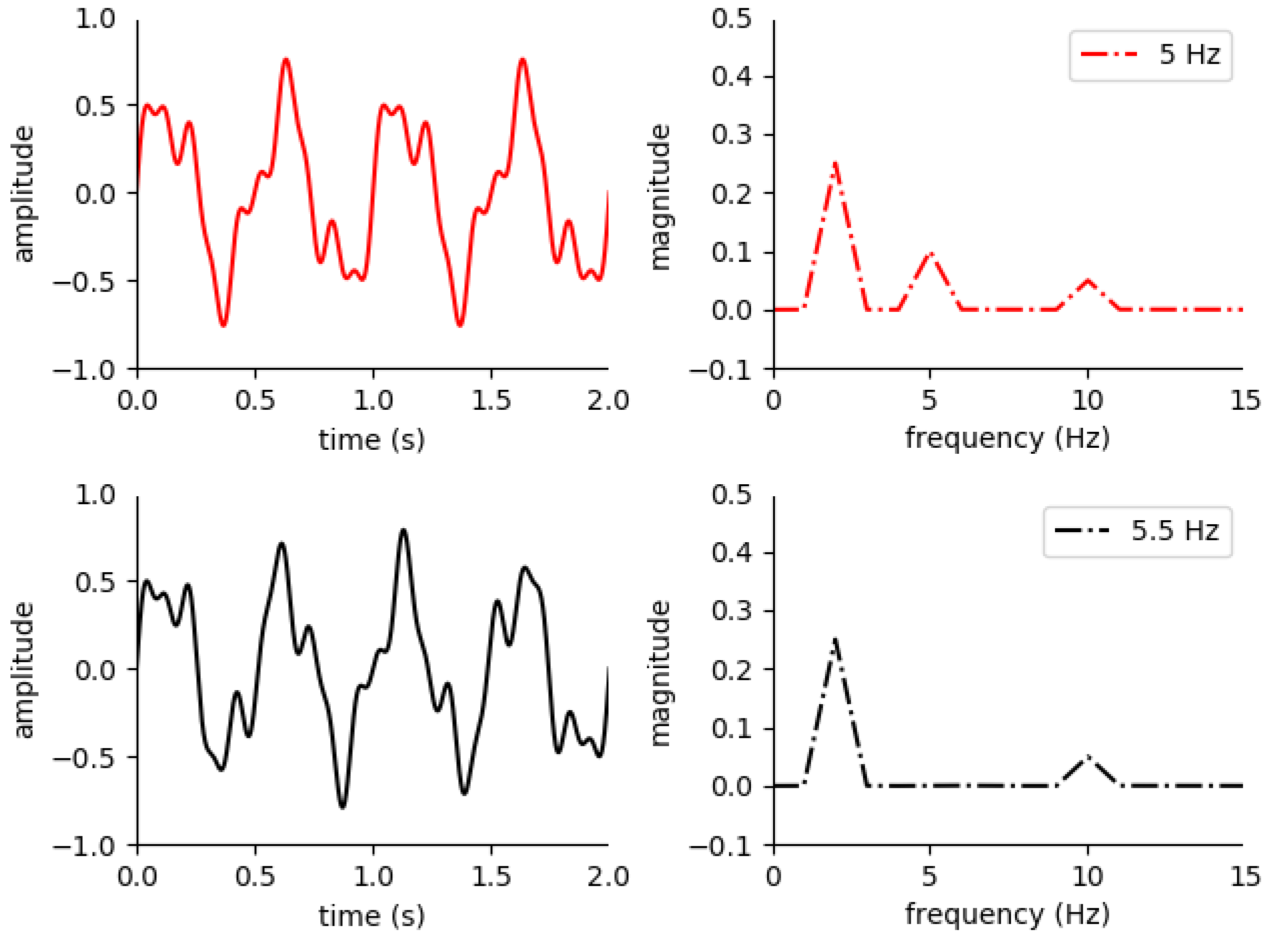


See, “L05_fourier_transform_sin_and_cos.py”

Frequency resolution (1/2)

How to define precision/resolution in frequency domain?

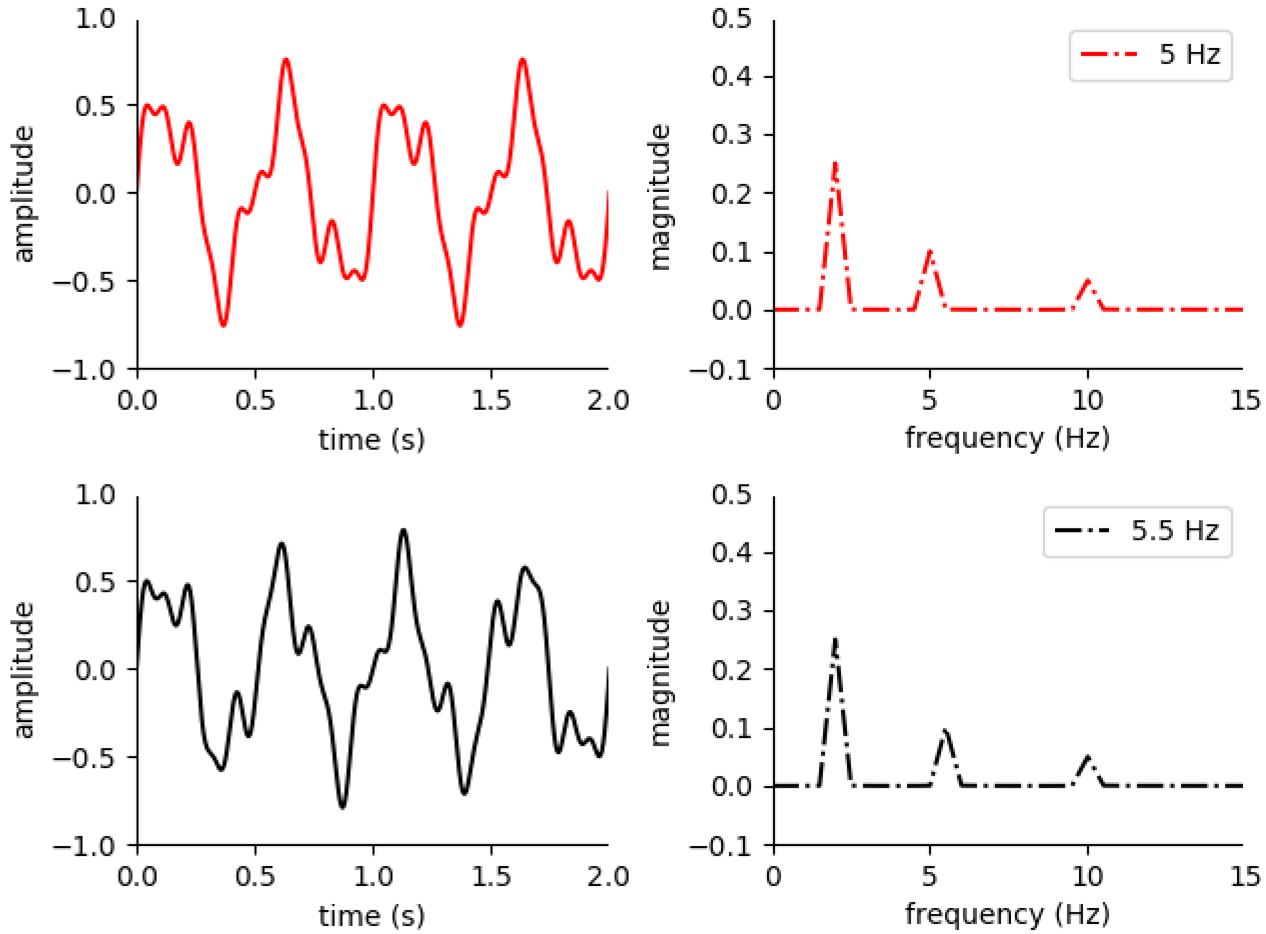
```
nFFT = fs # resolution = fs / nFFT

# signal
x1  = 0.5 * np.sin(2 * np.pi * 2 * t)
x2a = 0.2 * np.sin(2 * np.pi * 5 * t)
x2b = 0.2 * np.sin(2 * np.pi * 5.5 * t)
x3  = 0.1 * np.sin(2 * np.pi * 10 * t)

xA = x1 + x2a + x3
xB = x1 + x2b + x3

# fourier transform
yA = fourier_tansform(xA, nFFT)
yB = fourier_tansform(xB, nFFT)
```



See, "L05_frequency_resolution.py"

Frequency resolution (2/2)

```
nFFT = 2 * fs # resolution = fs / nFFT

# signal
x1  = 0.5 * np.sin(2 * np.pi * 2 * t)
x2a = 0.2 * np.sin(2 * np.pi * 5 * t)
x2b = 0.2 * np.sin(2 * np.pi * 5.5 * t)
x3  = 0.1 * np.sin(2 * np.pi * 10 * t)

xA = x1 + x2a + x3
xB = x1 + x2b + x3

# fourier transform
yA = fourier_tansform(xA, nFFT)
yB = fourier_tansform(xB, nFFT)
```



See, “L05_frequency_resolution.py”

Fast Fourier Transform

A fast Fourier transform (FFT) algorithm computes the discrete Fourier transform (DFT) in computationally efficient manner.

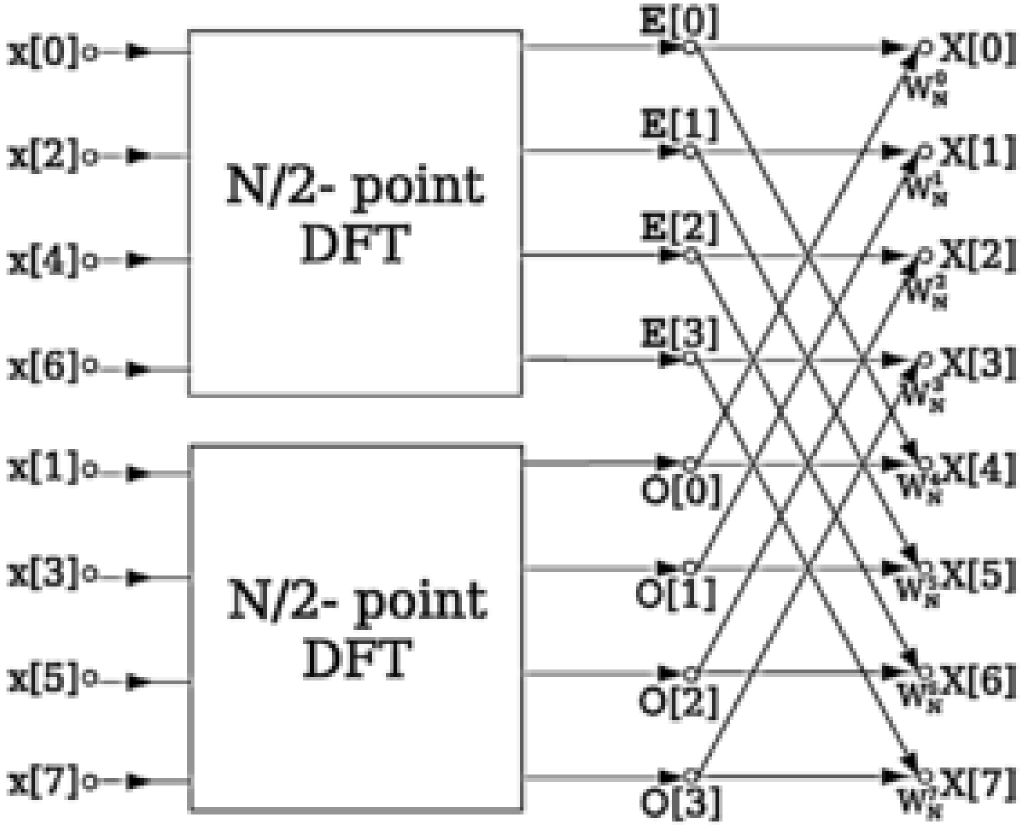
```
from scipy.fftpack import fft

# fourier transform
y = fourier_tansform(x, nFFT)
u = fft(x, nFFT)
```

$$W^{0}{}_{8} = cos\left(\frac{2\pi \times 0}{8}\right) - i \times sin\left(\frac{2\pi \times 0}{8}\right) = 1$$

$$W^{1}{}_{8} = cos\left(\frac{2\pi \times 1}{8}\right) - i \times sin\left(\frac{2\pi \times 1}{8}\right) = 0.7071 - i0.7071$$

$$W^{2}{}_{8} = cos\left(\frac{2\pi \times 2}{8}\right) - i \times sin\left(\frac{2\pi \times 2}{8}\right) = -i$$

$$W^{3}{}_{8} = cos\left(\frac{2\pi \times 3}{8}\right) - i \times sin\left(\frac{2\pi \times 3}{8}\right) = -0.7071 - i0.7071$$



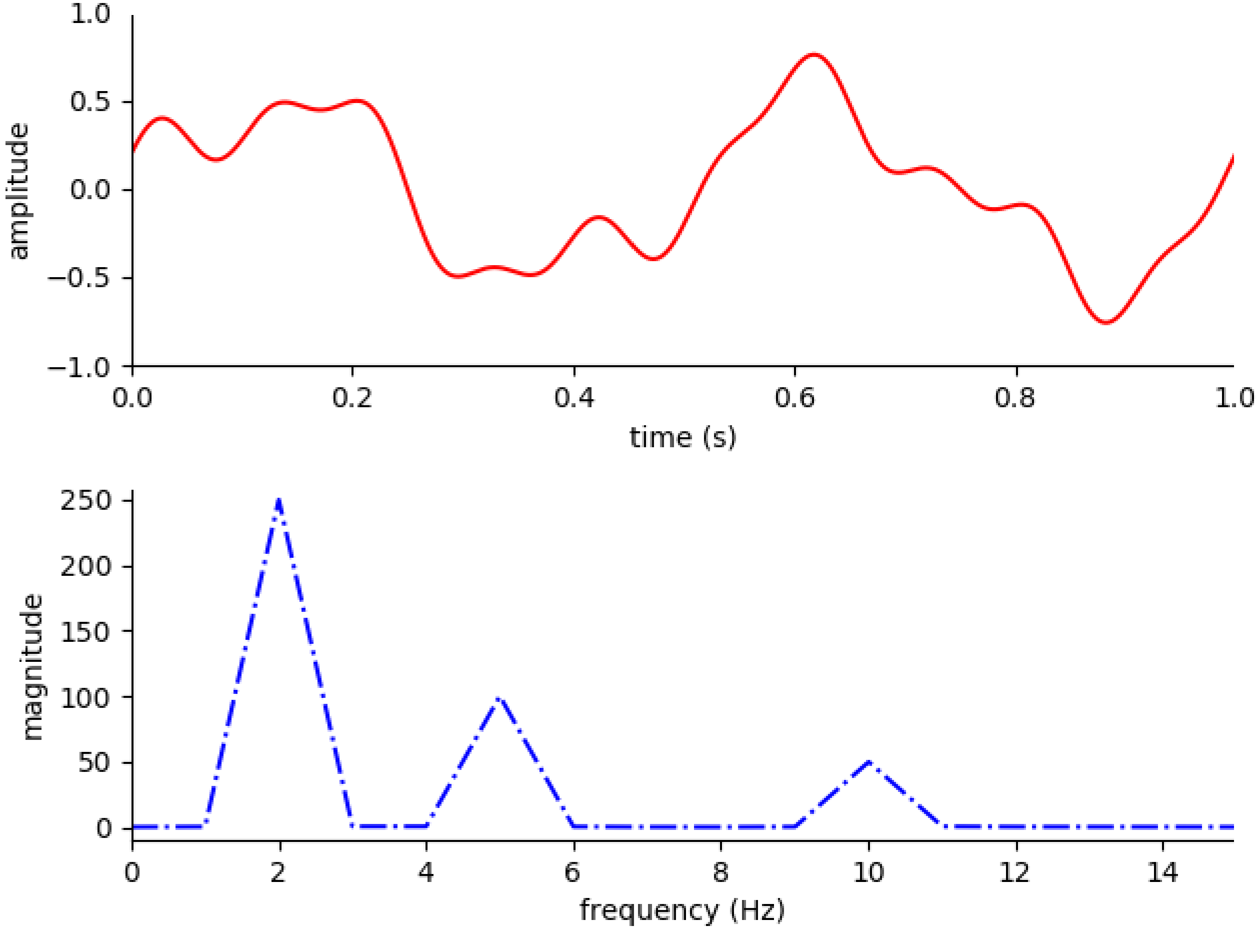
wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

Section 3. Inverse Fourier transform

Inverse discrete Fourier transform (1/2)

Is it possible to restore signal from its spectrum?

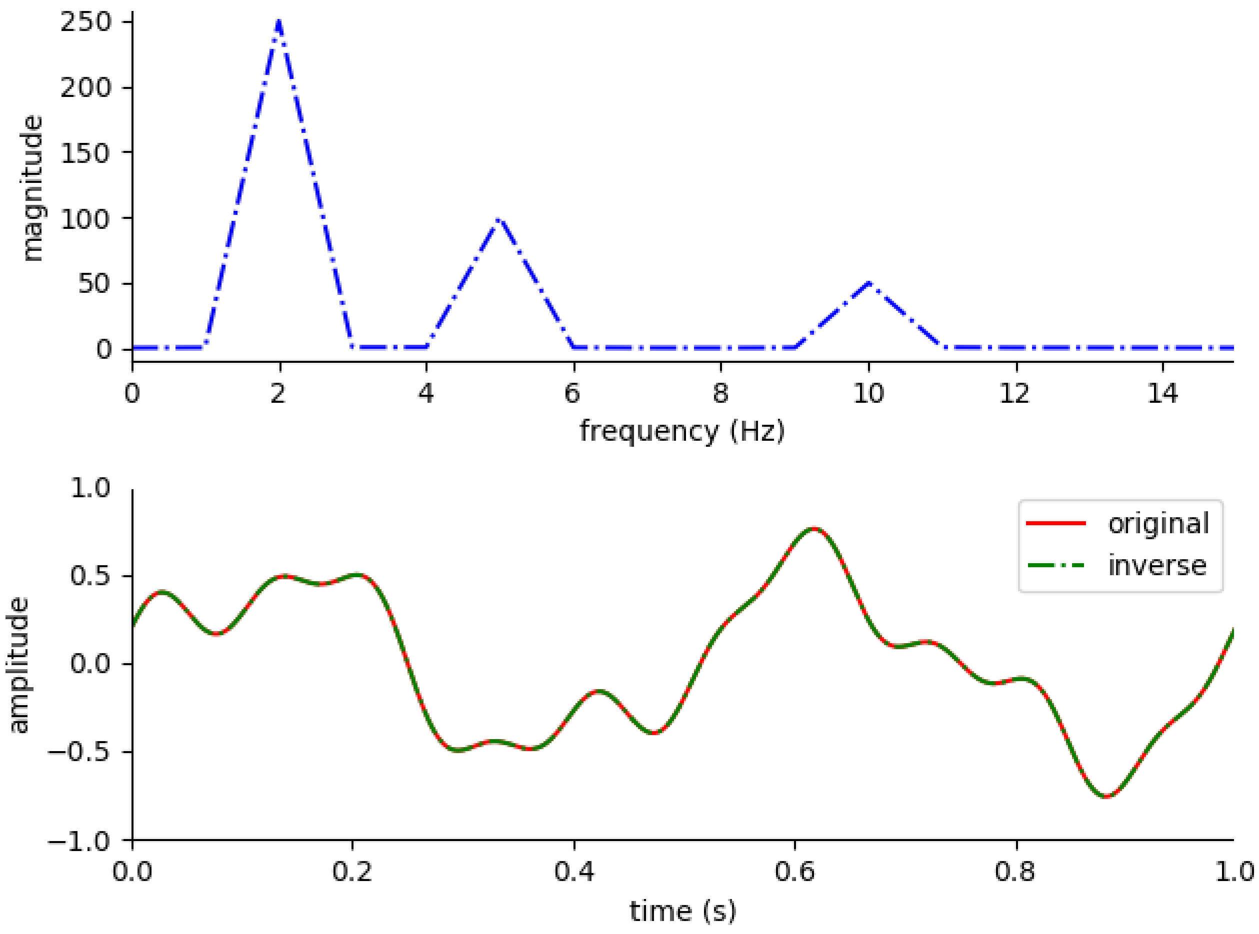
```
# fourier transform
y = np.zeros(nFFT, 'complex')
t = np.arange(0, N)
for k in range(0, nFFT):
  # relative frequency
  f = k / nFFT
  # complex exponent
  y[k] = np.sum(np.exp(-1j * 2 * np.pi * t * f)
         * x)
```



See, "L05_inverse_fourier_transform.py"

Inverse discrete Fourier transform (2/2)

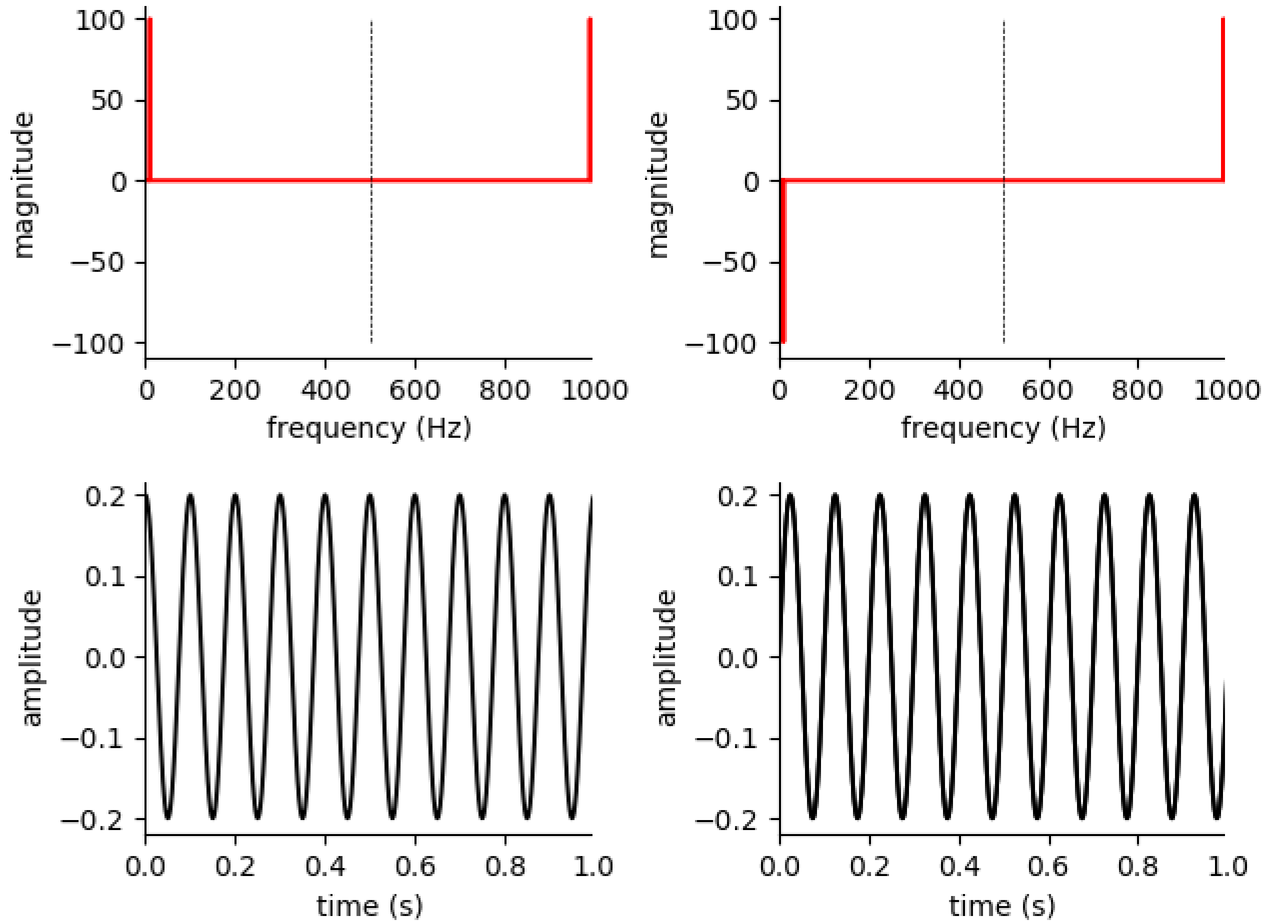
```
# inverse fourier transform
x = np.zeros(N, 'complex')
t = np.arange(0, N)
for k in range(0, N):
  # relative frequency
  f = k / nFFT
  # complex exponent
  x[k] = (1/N) *
         np.sum(np.exp(1j * 2 * np.pi * t * f)
         * y)
```



See, “L05_inverse_fourier_transform.py”

Spectra modification

```
# frequency
f0 = 10

# cos
y = np.zeros(N, 'complex')
y[f0] = 100.0
y[N-f0] = 100.0

# sin
u = np.zeros(N, 'complex')
u[f0] = -1j * 100.0
u[N-f0] = 1j * 100.0

# inverse fourier transform
x = ifft(y)
z = ifft(u)
```

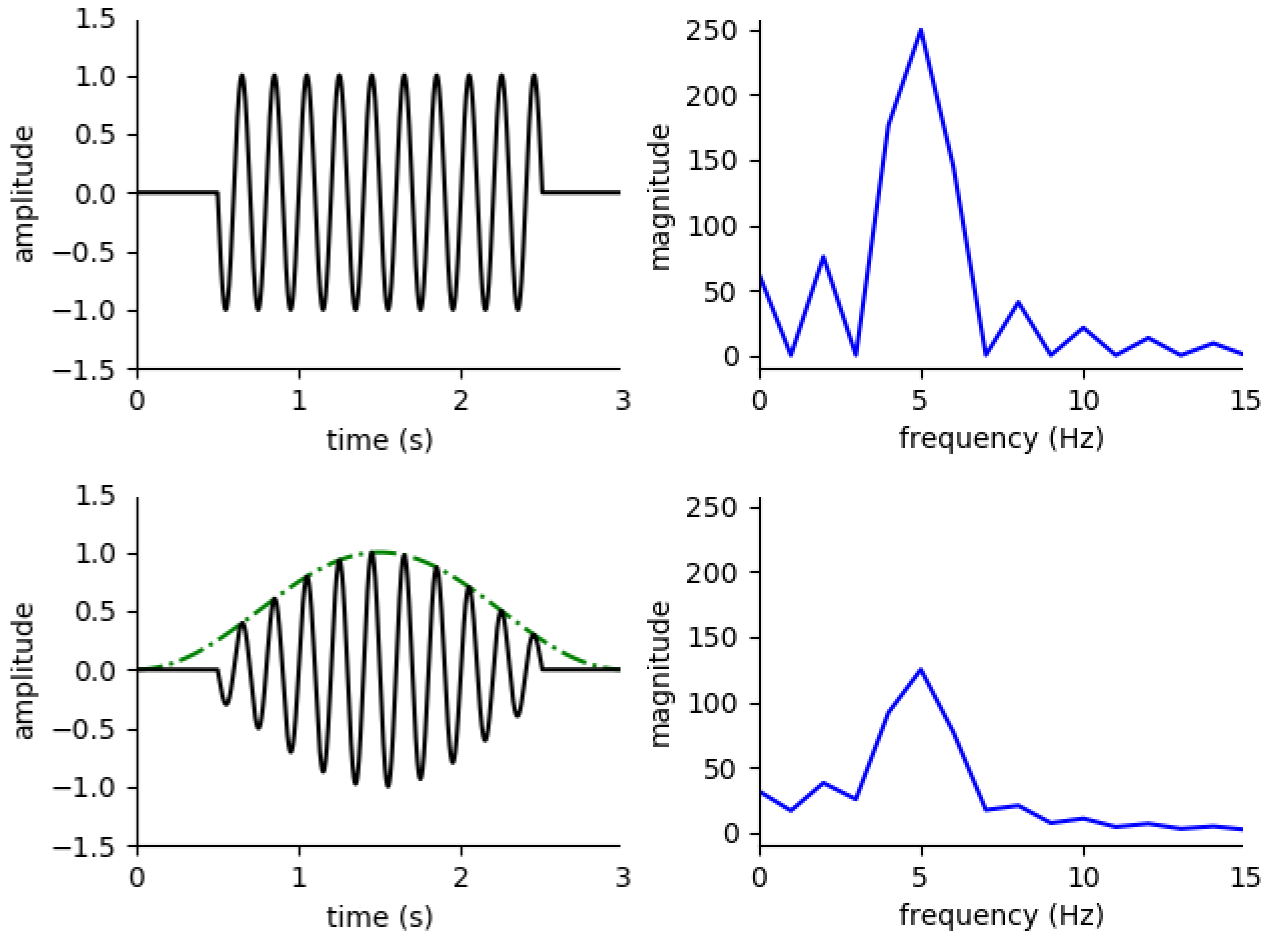


See, “L05_spectra_modification.py”

Section 4. Non-periodic signals and windowing

Fourier transform of non-periodic signals and windowing

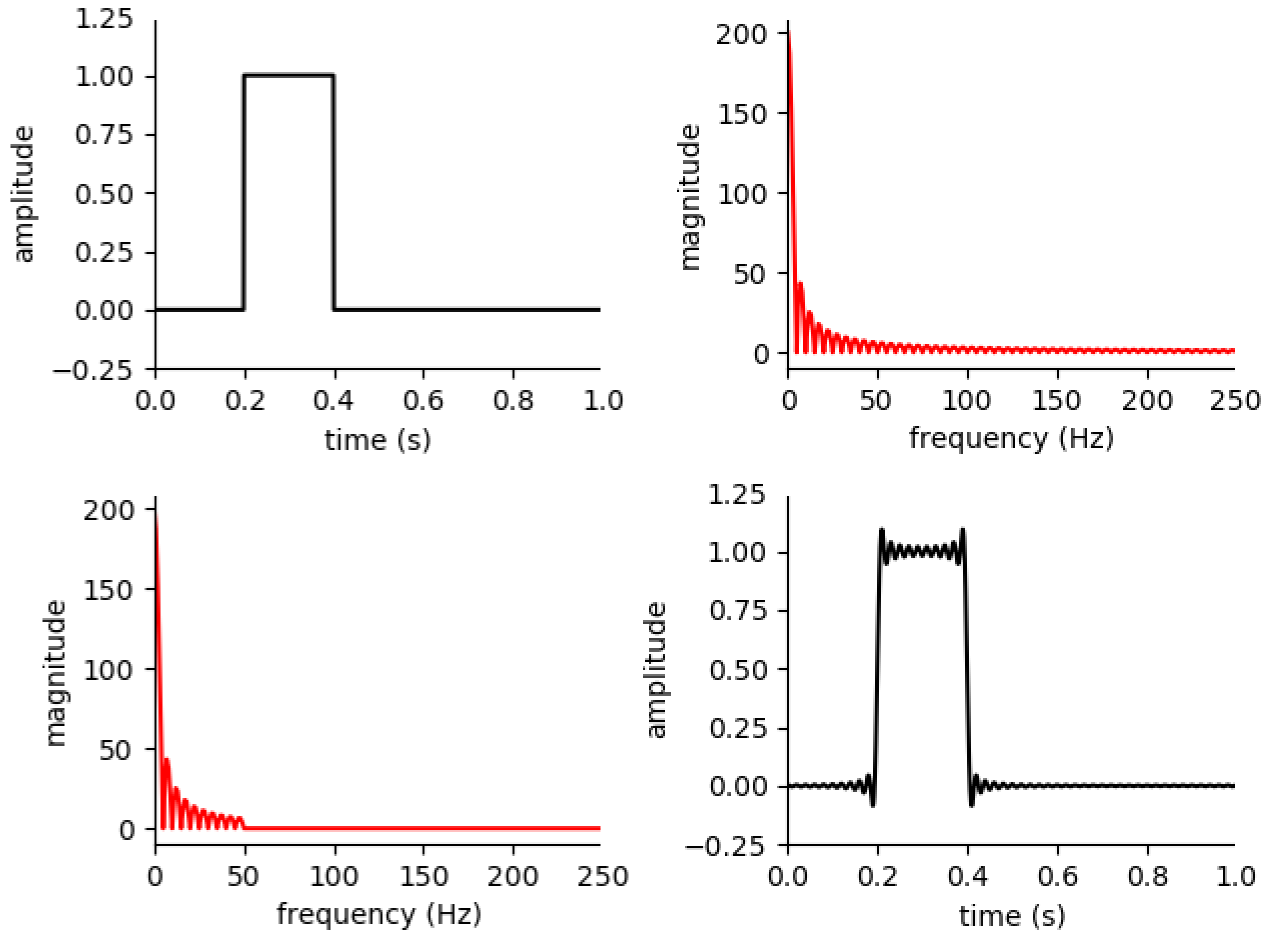
```
from scipy.fftpack import fft, ifft
from scipy import signal

# window
w = signal.hanning(N)
z = x * w

# fourier transform
y = fft(x, nFFT)
u = fft(z, nFFT)
```



See, “L05_non_periodic_signal.py”

Fourier transform of non-sinusoid signals

```
# signal
x = np.zeros(N)
x[200:400] = 1.0

# fourier transform
y = fft(x, nFFT)
Y = np.abs(y)

# cut spectrum
M = 50
y[M:(N-M)] = 0

# magnitude
U = np.abs(y)

# inverse fourier transform
z = ifft(y)
```

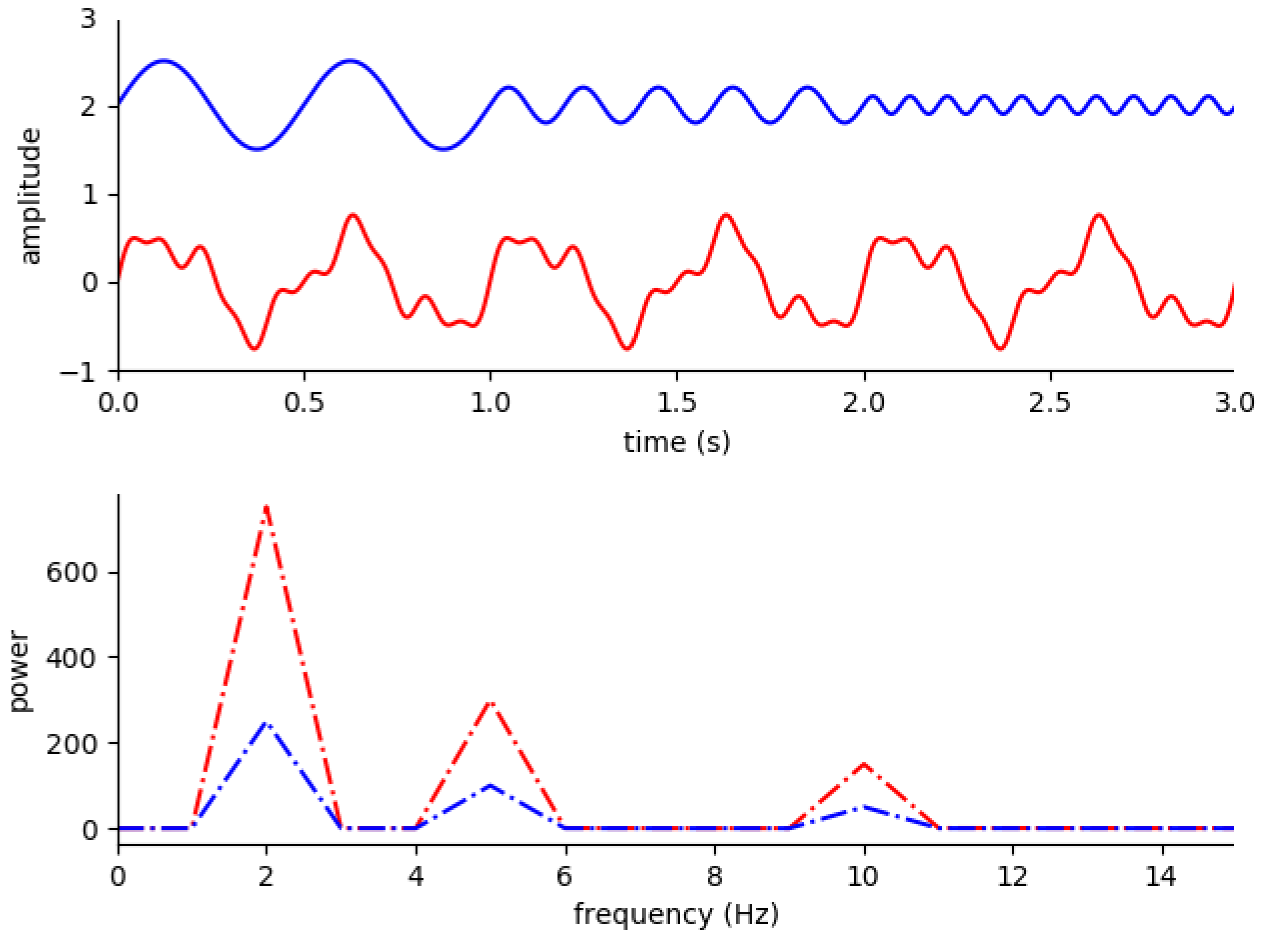


See, “L05_non_periodic_signal_pulse.py”

Section 5. Short-time Fourier transform

Short-time Fourier transform (1/3)

What is the difference in power spectra
for periodic and non-periodic signal?



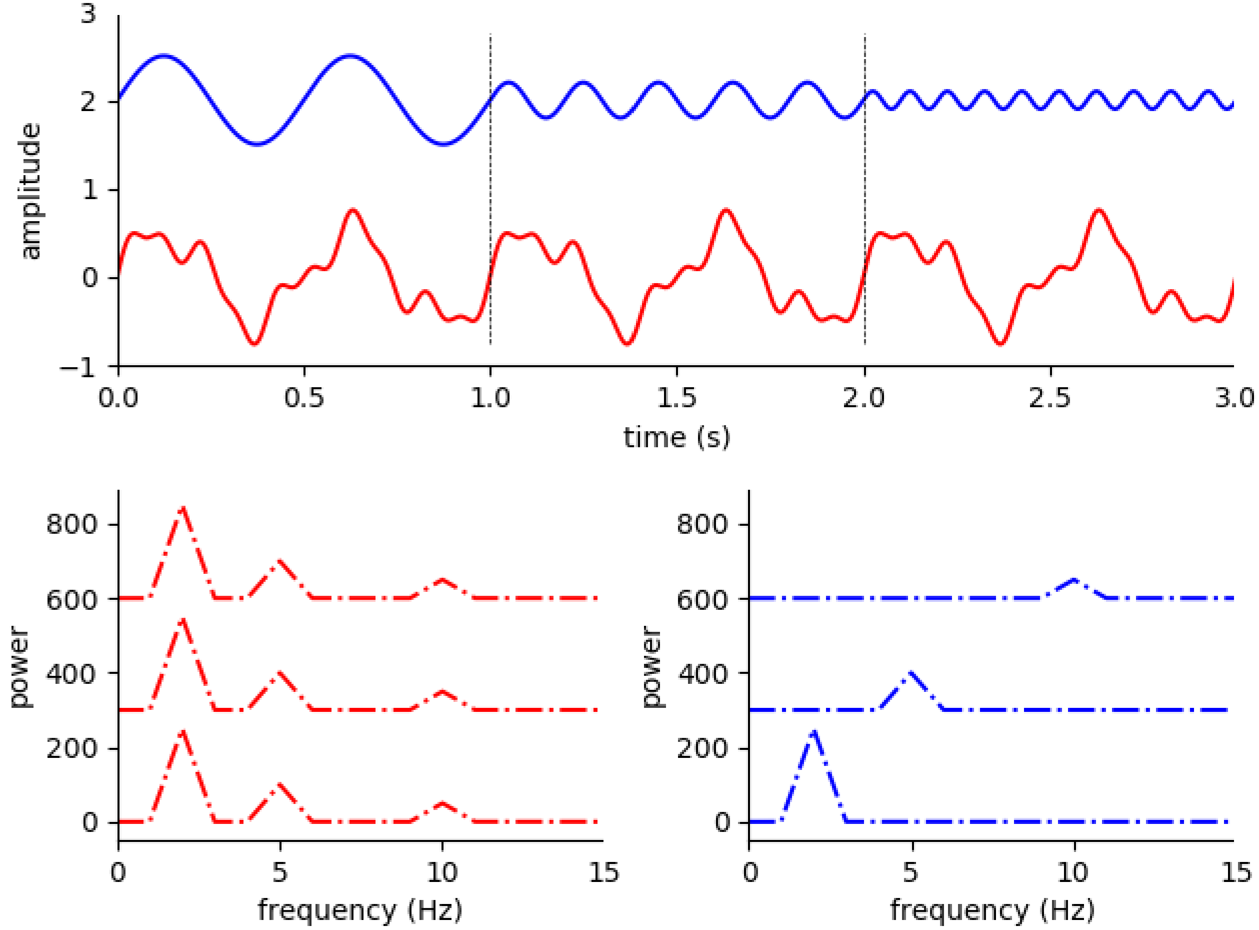

```
# signal
x1 = 0.5 * np.sin(2 * np.pi * 2 * t)
x2 = 0.2 * np.sin(2 * np.pi * 5 * t)
x3 = 0.1 * np.sin(2 * np.pi * 10 * t)
xA = x1 + x2 + x3
xB = np.concatenate((x1[:L], x2[:L], x3[:L]))
```

See, “L05_fourier_transform_short.py”

Short Fourier transform (2/3)

What if extract quasi-periodic segments?

```
# fourier transform
YA = np.zeros((nFFT, 3))
YB = np.zeros((nFFT, 3))
for i in range(0, 3):
  t1 = i * L
  t2 = (i + 1) * L
  yA = fourier_tansform(xA[t1:t2], nFFT)
  yB = fourier_tansform(xB[t1:t2], nFFT)
  YA[:, i] = np.abs(yA)
  YB[:, i] = np.abs(yB)
```

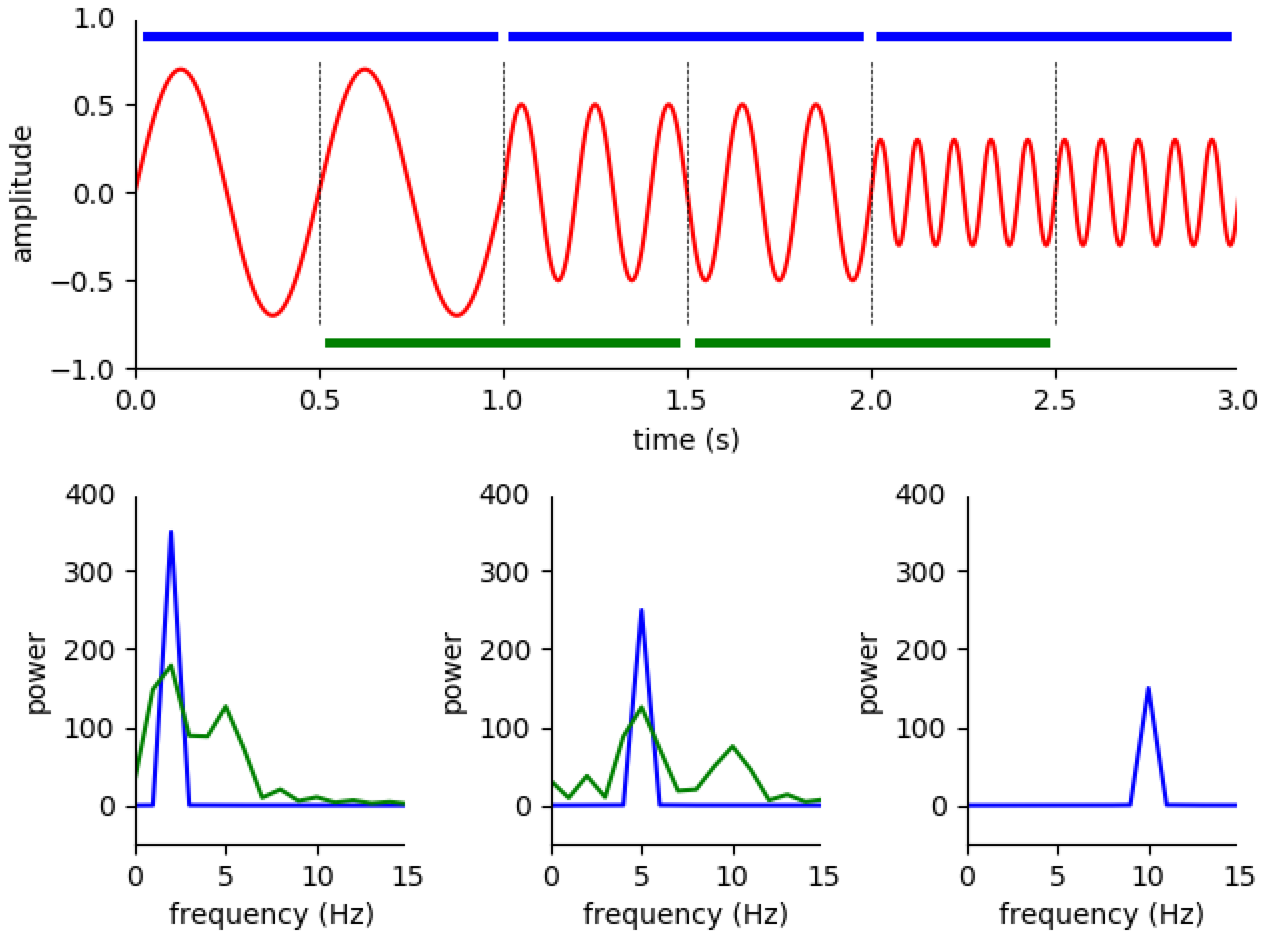


See, “L05_fourier_transform_short_segments.py”

Short Fourier transform (3/3)

How does the spectra look at the border of segment?

```
# fourier transform
M = 5
Y = np.zeros((nFFT, M))
for i in range(0, M):
  t1 = i * L//2
  t2 = t1 + L
  y = fourier_tansform(x[t1:t2], nFFT)
  Y[:, i] = np.abs(y)
```



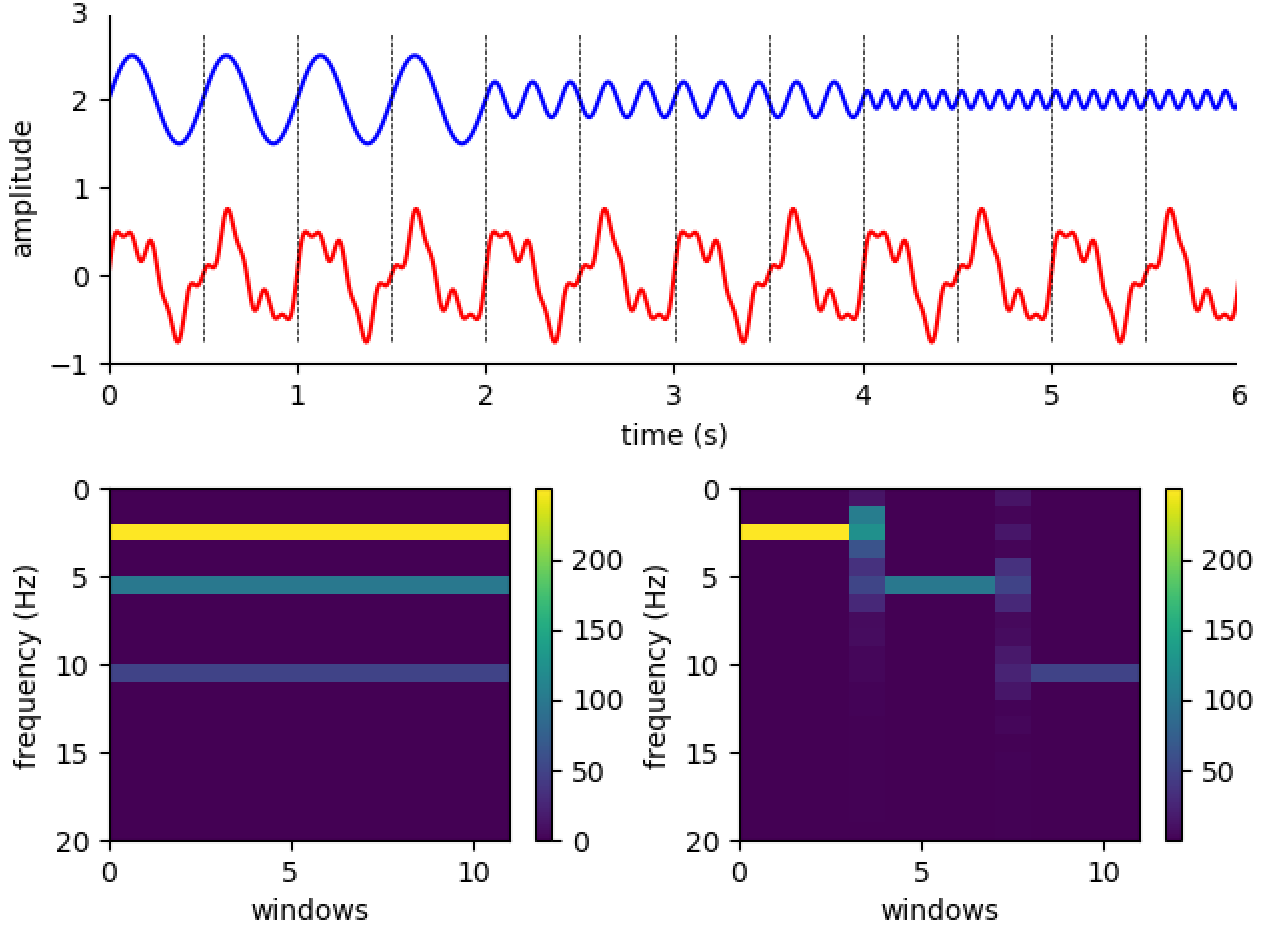
See, “L05_fourier_transform_short_segments_overlap.py”

Section 6. Time-frequency representation

Time-frequency representation

Signal can be represented simultaneously in time and frequency domain.

```
# fourier transform
step = 0.5
duration = 1.0
M = int(T / step) - 1
YA = np.zeros((nFFT, M))
YB = np.zeros((nFFT, M))
for i in range(0, M):
  t1 = i * int(step * fs)
  t2 = t1 + int(duration * fs)
  yA = fft(xA[t1:t2], nFFT)
  yB = fft(xB[t1:t2], nFFT)
  YA[:, i] = np.abs(yA)
  YB[:, i] = np.abs(yB)
```



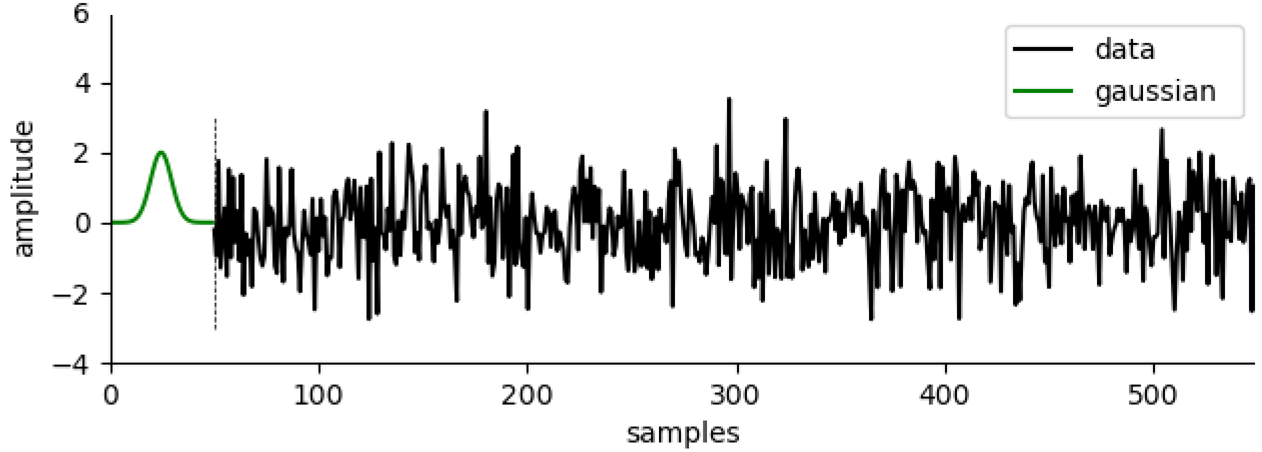
See, “L05_fourier_transform_time_frequency.py”

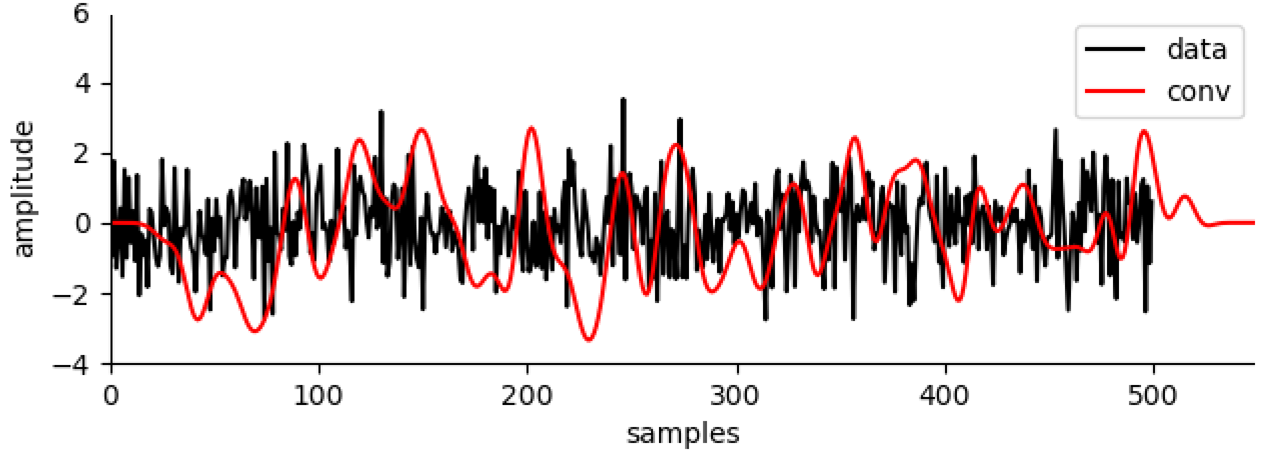
Section 7. Properties of Fourier transform

Property 1: Convolution and FFT (1/2)

Convolution in time domain equals to
Product in frequency domain.

```
# init
N = len(x)
M = len(w)

# add zeros
x = np.concatenate((np.zeros(M-1), x,
np.zeros(M-1)))
y = np.zeros(N+M-1)

# convolution
for n in range(0, (N+M-1)):
  y[n] = np.sum(x[n:(n + M)] * w[::-1])
```

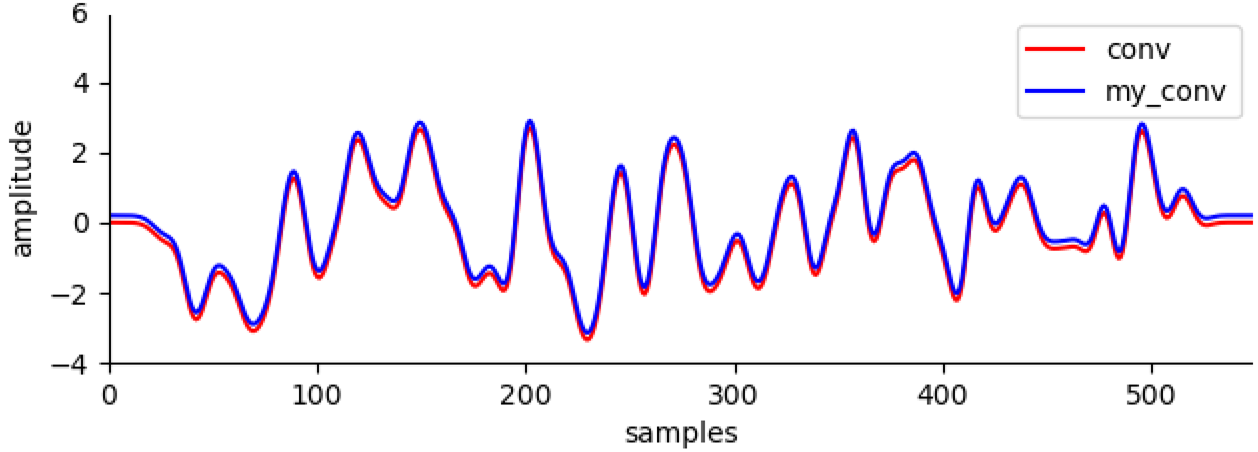


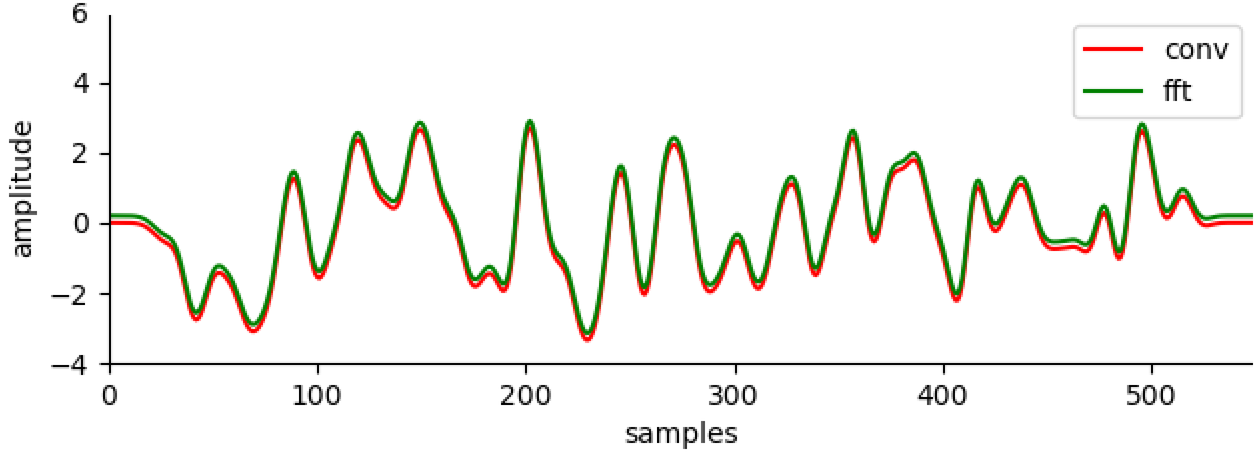


See, “L05_convolution_and_product.py”

Property 1: Convolution and FFT (2/2)

```
# signal
x = np.random.randn(N)

# window
w = signal.gaussian(M, 5)

# convolution
y = signal.convolve(x, w)

# Fourier transform
nFFT = N+M-1
u = ifft(fft(x, nFFT) * fft(w, nFFT))
```



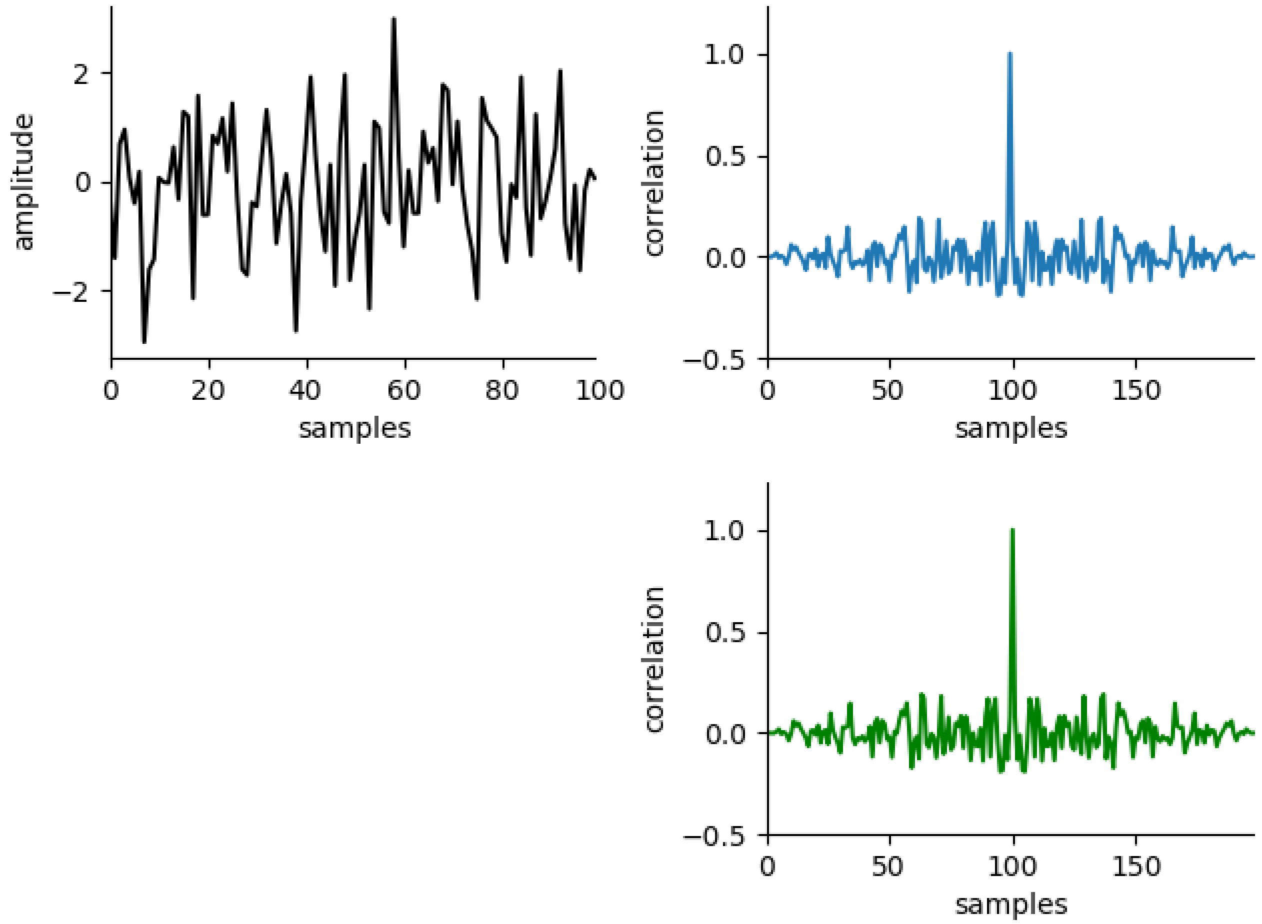


See, “L05_convolution_and_product.py”

Property 2: Autocorrelation and FFT

Autocorrelation function can be computed using FFT.

```
# compute ACF
rx = signal.correlate(x, x)
rx = rx / np.max(rx)

# compute ACF using FFT
nFFT = 2 * N
ry = np.real(ifft(fft(x, nFFT) *
          np.conj(fft(x, nFFT))))

ry = np.concatenate((ry[N::-1], ry[1:N:]))

ry = ry / np.max(ry)
```



See, “L05_acf_via_fft.py”

Literature

- **Python programming language**

- http://www.scipy-lectures.org/, see “materials/L02_ScipyLectures.pdf”

- **Data analysis**

- Downey A., “**Think DSP: Digital Signal Processing in Python**”, see materials/L05_thinkdsp.pdf
- National Instruments Inc., “Understanding FFTs and Windowing”, see materials/L05_Understanding FFTs and Windowing.pdf