Fourier Transform theorems and properties

Xavier Serra
Music Technology Group
Universitat Pompeu Fabra, Barcelona
http://mtg.upf.edu

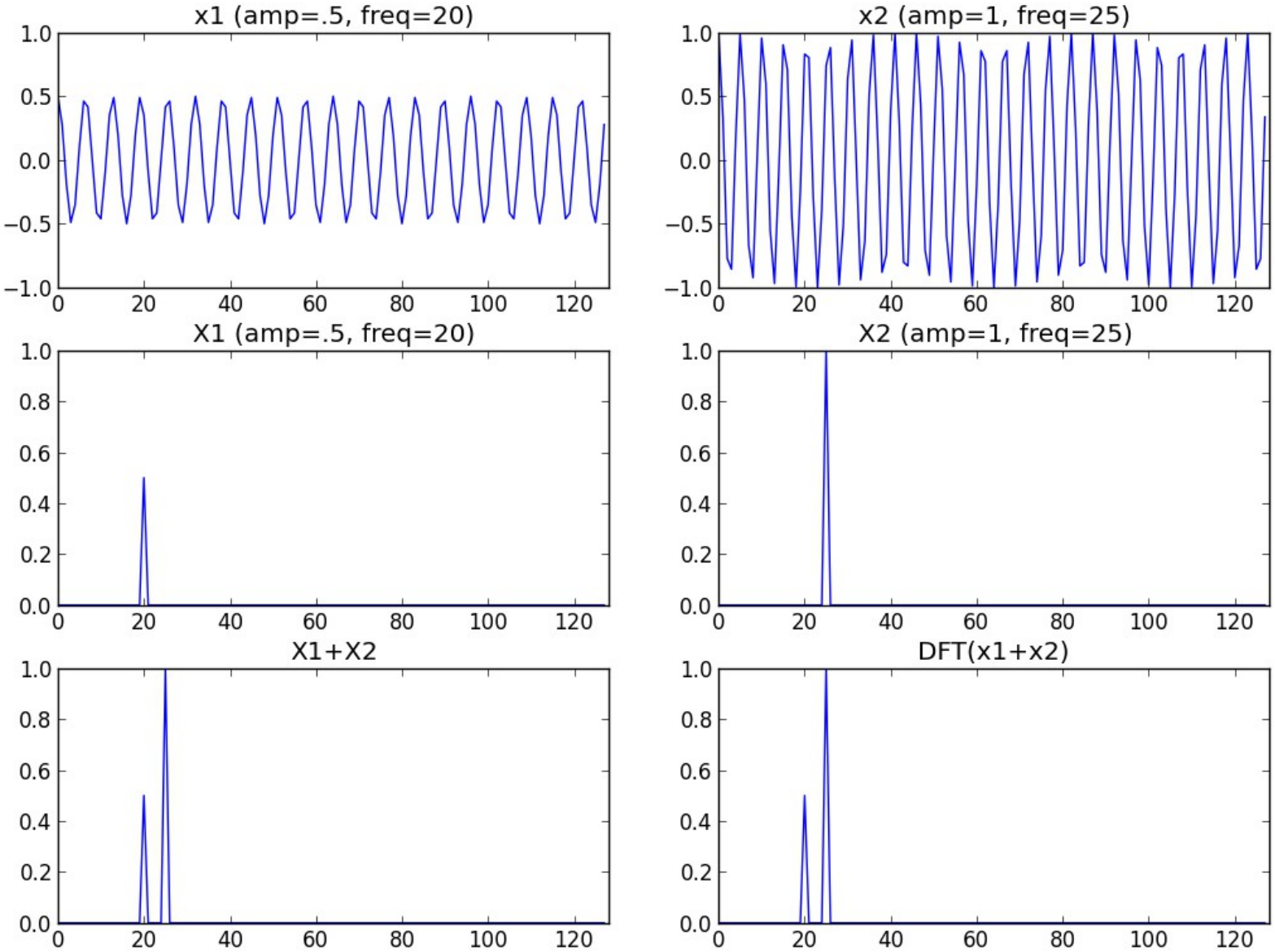
Index

- Linearity
- Convolution
- Shift
- Evenness & phase unwrapping
- Zero-padding
- Power & amplitude in dB
- Fast Fourier Transform (FFT)

Linearity

$$ax_1 + bx_2 \leftrightarrow aX_1 + bX_2$$

$$\begin{aligned}
&DFT\left(a\,x_1[n]+b\,x_2[n]\right) \\
&\quad=\sum_{n=0}^{N-1}\left(a\,x_1[n]+b\,x_2[n]\right)e^{-j\,2\pi\,kn/N} \\
&\quad=\sum_{n=0}^{N-1}a\,x_1[n]e^{-j\,2\pi\,kn/N}+\sum_{n=0}^{N-1}b\,x_2[n]e^{-j\,2\pi\,kn/N} \\
&\quad=a\sum_{n=0}^{N-1}x_1[n]e^{-j\,2\pi\,kn/N}+b\sum_{n=0}^{N-1}x_2[n]e^{-j\,2\pi\,kn/N} \\
&\quad=a\,X_1[k]+b\,X_2[k]
\end{aligned}$$

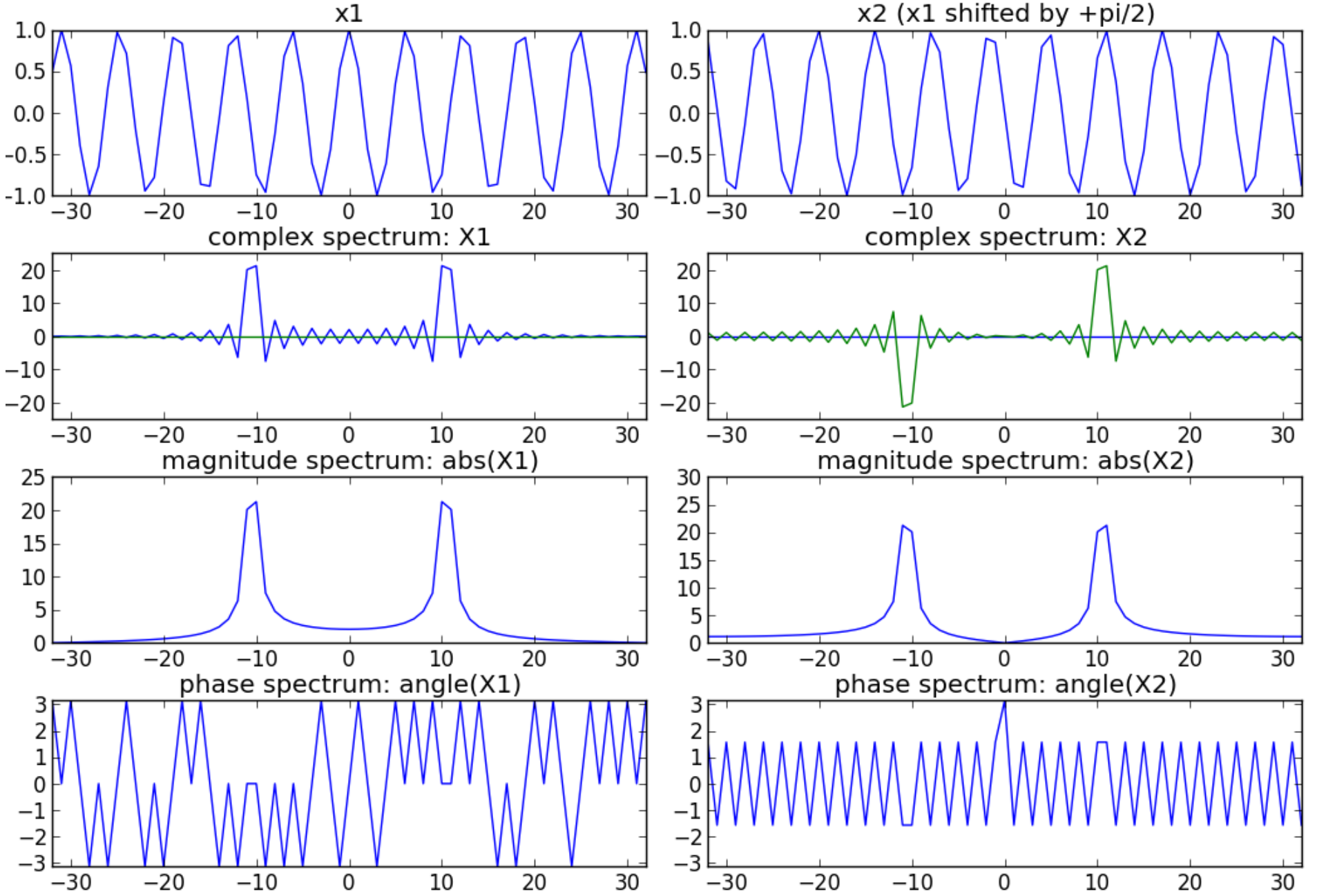

x1 (amp=.5, freq=20)
x2 (amp=1, freq=25)
X1 (amp=.5, freq=20)
X2 (amp=1, freq=25)
X1+X2
DFT(x1+x2)

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.fftpack import fft, ifft

a = 0.5
b = 1.0
k1 = 20
k2 = 25
N = 128
x1 = a*np.exp(1j*2*np.pi*k1/N*np.arange(N))
x2 = b*np.exp(1j*2*np.pi*k2/N*np.arange(N))

plt.subplot(321)
plt.plot(np.arange(0, N, 1.0), np.real(x1))

plt.subplot(322)
plt.plot(np.arange(0, N, 1.0), np.real(x2))

X1 = fft(x1)
mX1 = abs(X1)/N
plt.subplot(323)
plt.plot(np.arange(0, N, 1.0), mX1)

X2 = fft(x2)
mX2 = abs(X2)/N
plt.subplot(324)
plt.plot(np.arange(0, N, 1.0), mX2)

plt.subplot(325)
plt.plot(np.arange(0, N, 1.0), mX1+mX2)

X = fft(x1+x2)
mX= abs(X)/N
plt.subplot(326)
plt.plot(np.arange(0, N, 1.0), mX)
```

Shift

shift $\leftrightarrow$ multiplication by a complex exponential

$$
\begin{aligned}
&DFT\left(x[n-n_0]\right) \\
&\quad = \sum_{n=0}^{N-1} x[n-n_0]e^{-j2\pi kn/N} \\
&\quad = \sum_{m=-n_0}^{N-1-n_0} x[m]e^{-j2\pi k(m+n_0)/N} \qquad (m=n-n_0) \\
&\quad = \sum_{m=0}^{N-1} x[m]e^{-j2\pi km/N}e^{-j2\pi kn_0/N} \\
&\quad = e^{-j2\pi kn_0/N}\sum_{m=0}^{N-1} x[m]e^{-j2\pi km/N} \\
&\quad = e^{-j2\pi kn_0/N}X[k]
\end{aligned}
$$


x1
x2 (x1 shifted by +pi/2)
complex spectrum: X1
complex spectrum: X2
magnitude spectrum: abs(X1)
magnitude spectrum: abs(X2)
phase spectrum: angle(X1)
phase spectrum: angle(X2)

```
import numpy as np
import matplotlib.pyplot as plt
N = 64
k0 = 10.5

X = np.array([])
x = np.cos(2*np.pi*k0/(N+1)*np.arange(-N/2, N/2+1))
plt.subplot(4,2,1)
plt.plot(np.arange(-N/2, N/2+1), x)

for k in range(-N/2, N/2+1):
     s = np.exp(1j*2*np.pi*k/(N+1)*np.arange(-N/2, N/2+1))
     X = np.append(X, sum(x*np.conjugate(s)))
plt.subplot(4,2,3)
plt.plot(np.arange(-N/2, N/2+1), np.real(X))
plt.plot(np.arange(-N/2, N/2+1), np.imag(X))

plt.subplot(4,2,5)
plt.plot(np.arange(-N/2, N/2+1), abs(X))

plt.subplot(4,2,7)
plt.plot(np.arange(-N/2, N/2+1), np.angle(X))

X = np.array([])
x = np.cos(2*np.pi*k0/(N+1)*np.arange(-N/2, N/2+1)+np.pi/2)
plt.subplot(4,2,2)
plt.plot(np.arange(-N/2, N/2+1), x)

for k in range(-N/2, N/2+1):
     s = np.exp(1j*2*np.pi*k/(N+1)*np.arange(-N/2, N/2+1))
     X = np.append(X, sum(x*np.conjugate(s)))
plt.subplot(4,2,4)
plt.plot(np.arange(-N/2, N/2+1), np.real(X))
plt.plot(np.arange(-N/2, N/2+1), np.imag(X))

plt.subplot(4,2,6)
plt.plot(np.arange(-N/2, N/2+1), abs(X))

plt.subplot(4,2,8)
plt.plot(np.arange(-N/2, N/2+1), np.angle(X))
```

Evenness

even $\leftrightarrow$ real - valued

$$x_0[n] = A_0 \cos\left(2\pi k_0 n / N + \varphi_0\right)$$

$$= \frac{1}{2} A_0 e^{j\varphi_0} e^{j2\pi k_0 n/N} + \frac{1}{2} A_0 e^{-j\varphi_0} e^{-j2\pi k_0 n/N}$$

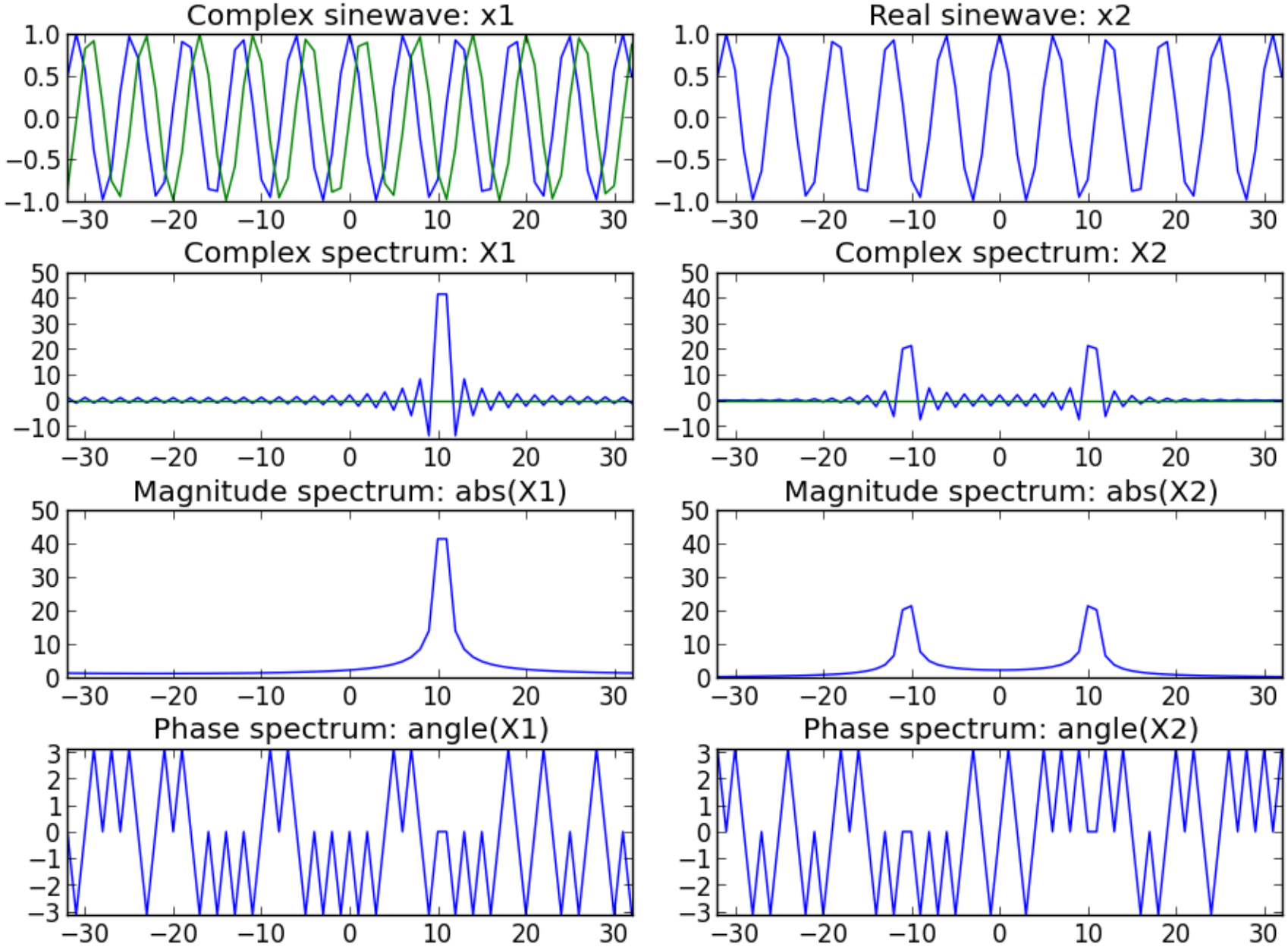
$$DFT\left(x[n]\right)$$

$$= \sum_{k=0}^{N-1} \left(\frac{1}{2} A_0 e^{j\varphi_0} e^{j2\pi k_0 n/N} + \frac{1}{2} A_0 e^{-j\varphi_0} e^{-j2\pi k_0 n/N}\right) e^{-j2\pi k n/N}$$

$$= \sum_{k=0}^{N-1} \frac{1}{2} A_0 e^{j\varphi_0} e^{j2\pi k_0 n/N} e^{-j2\pi k n/N}$$

$$+ \sum_{k=0}^{N-1} \frac{1}{2} A_0 e^{-j\varphi_0} e^{-j2\pi k_0 n/N} e^{-j2\pi k n/N}$$

$$= \frac{1}{2} A_0 e^{j\varphi_0} X_0[k] + \frac{1}{2} A_0 e^{-j\varphi_0} \bar{X}_0[k]$$

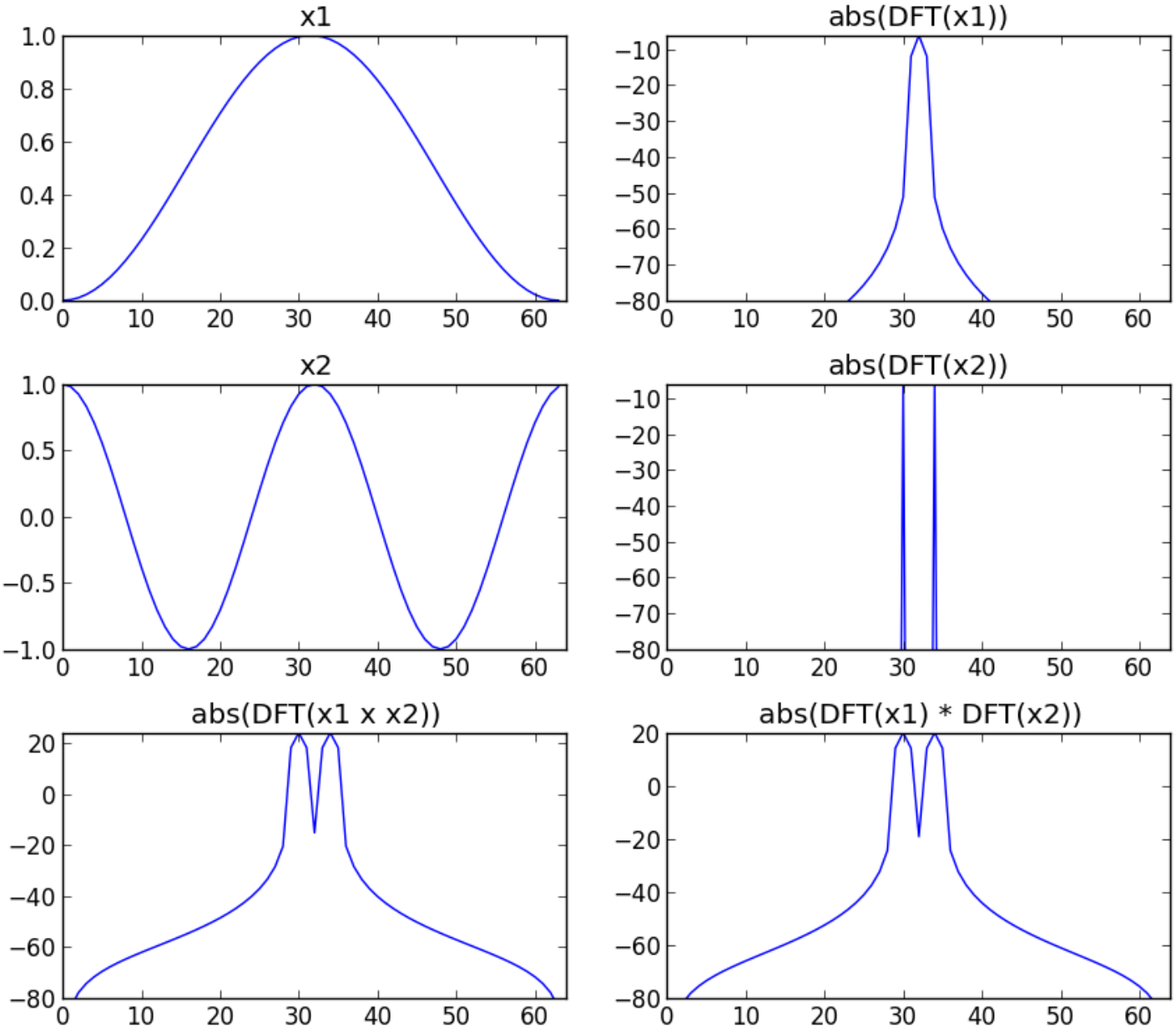

Complex sinewave: x1
Real sinewave: x2
Complex spectrum: X1
Complex spectrum: X2
Magnitude spectrum: abs(X1)
Magnitude spectrum: abs(X2)
Phase spectrum: angle(X1)
Phase spectrum: angle(X2)

```
mport numpy as np
import matplotlib.pyplot as plt

N = 64
k0 = 10.5
X = np.array([])
x = np.exp(1j*2*np.pi*k0/(N+1)*np.arange(-N/2, N/2+1))

plt.subplot(4,2,1)
plt.plot(np.arange(-N/2, N/2+1), np.real(x))
plt.plot(np.arange(-N/2, N/2+1), np.imag(x))

for k in range(-N/2, N/2+1):
     s = np.exp(1j*2*np.pi*k/(N+1)*np.arange(-N/2, N/2+1))
     X = np.append(X, sum(x*np.conjugate(s)))
plt.subplot(4,2,3)
plt.plot(np.arange(-N/2, N/2+1), np.real(X))
plt.plot(np.arange(-N/2, N/2+1), np.imag(X))

plt.subplot(4,2,5)
plt.plot(np.arange(-N/2, N/2+1), abs(X))

plt.subplot(4,2,7)
plt.plot(np.arange(-N/2, N/2+1), np.angle(X))

X = np.array([])
x = np.cos(2*np.pi*k0/(N+1)*np.arange(-N/2, N/2+1))
plt.subplot(4,2,2)
plt.plot(np.arange(-N/2, N/2+1), x)

for k in range(-N/2, N/2+1):
     s = np.exp(1j*2*np.pi*k/(N+1)*np.arange(-N/2, N/2+1))
     X = np.append(X, sum(x*np.conjugate(s)))
plt.subplot(4,2,4)
plt.plot(np.arange(-N/2, N/2+1), np.real(X))
plt.plot(np.arange(-N/2, N/2+1), np.imag(X))

plt.subplot(4,2,6)
plt.plot(np.arange(-N/2, N/2+1), abs(X))

plt.subplot(4,2,8)
plt.plot(np.arange(-N/2, N/2+1), np.angle(X))
```

Convolution

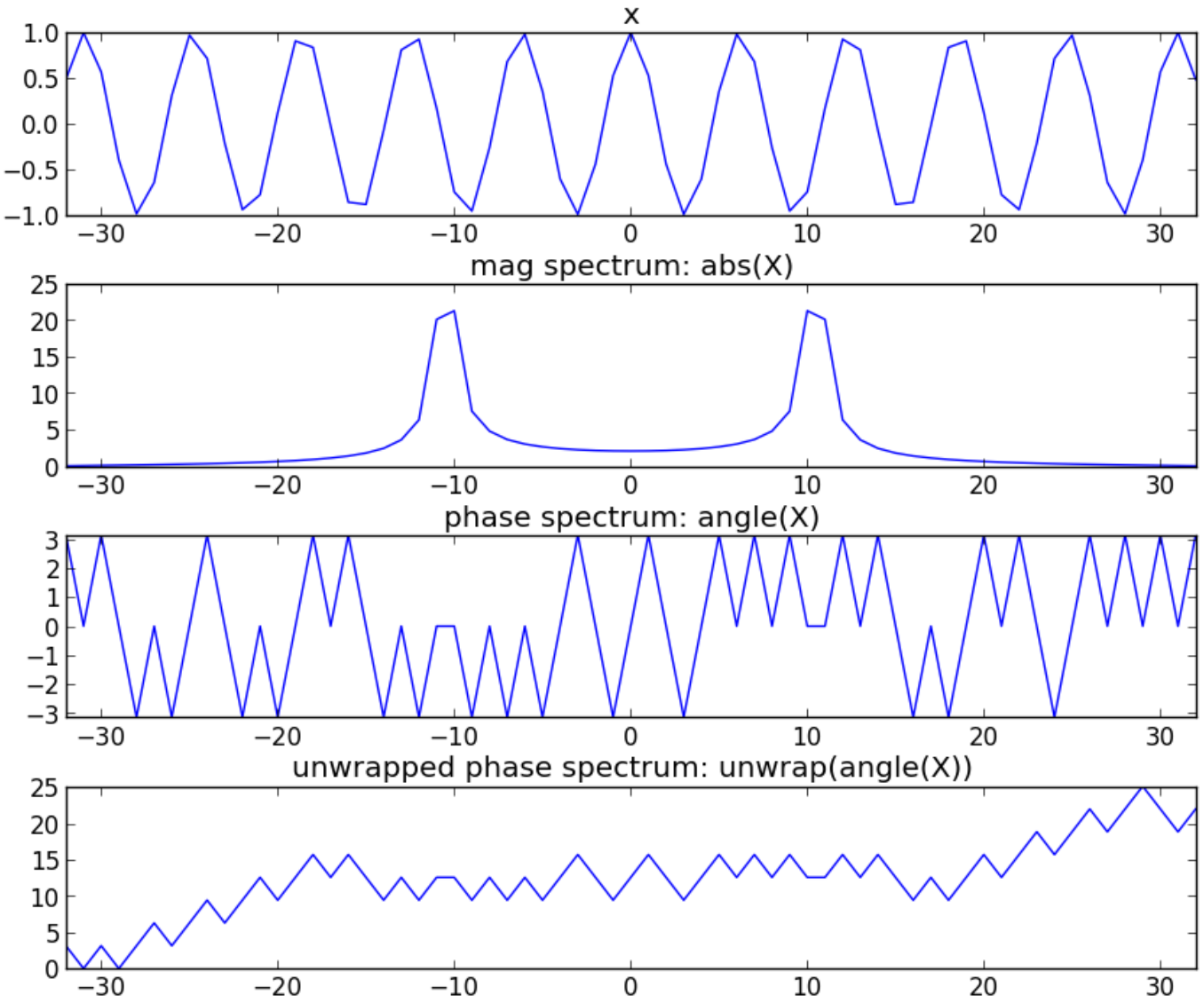
convolution $\leftrightarrow$ point - by - point multiplication

$$
\begin{aligned}
& DFT\left(x_1[n] * x_2[n]\right) \\
&= \sum_{n=0}^{N-1} \left(x_1[n] * x_2[n]\right) e^{-j 2\pi kn/N} \\
&= \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x_1[m] x_2[n-m] e^{-j 2\pi kn/N} \\
&= \sum_{m=0}^{N-1} x_1[m] \sum_{n=0}^{N-1} x_2[n-m] e^{-j 2\pi kn/N} \\
&= \left(\sum_{m=0}^{N-1} x_1[m] e^{-j 2\pi km/N}\right) X_2[k] \\
&= X_1[k] X_2[k]
\end{aligned}
$$


x1
abs(DFT(x1))
x2
abs(DFT(x2))
abs(DFT(x1 x x2))
abs(DFT(x1) * DFT(x2))

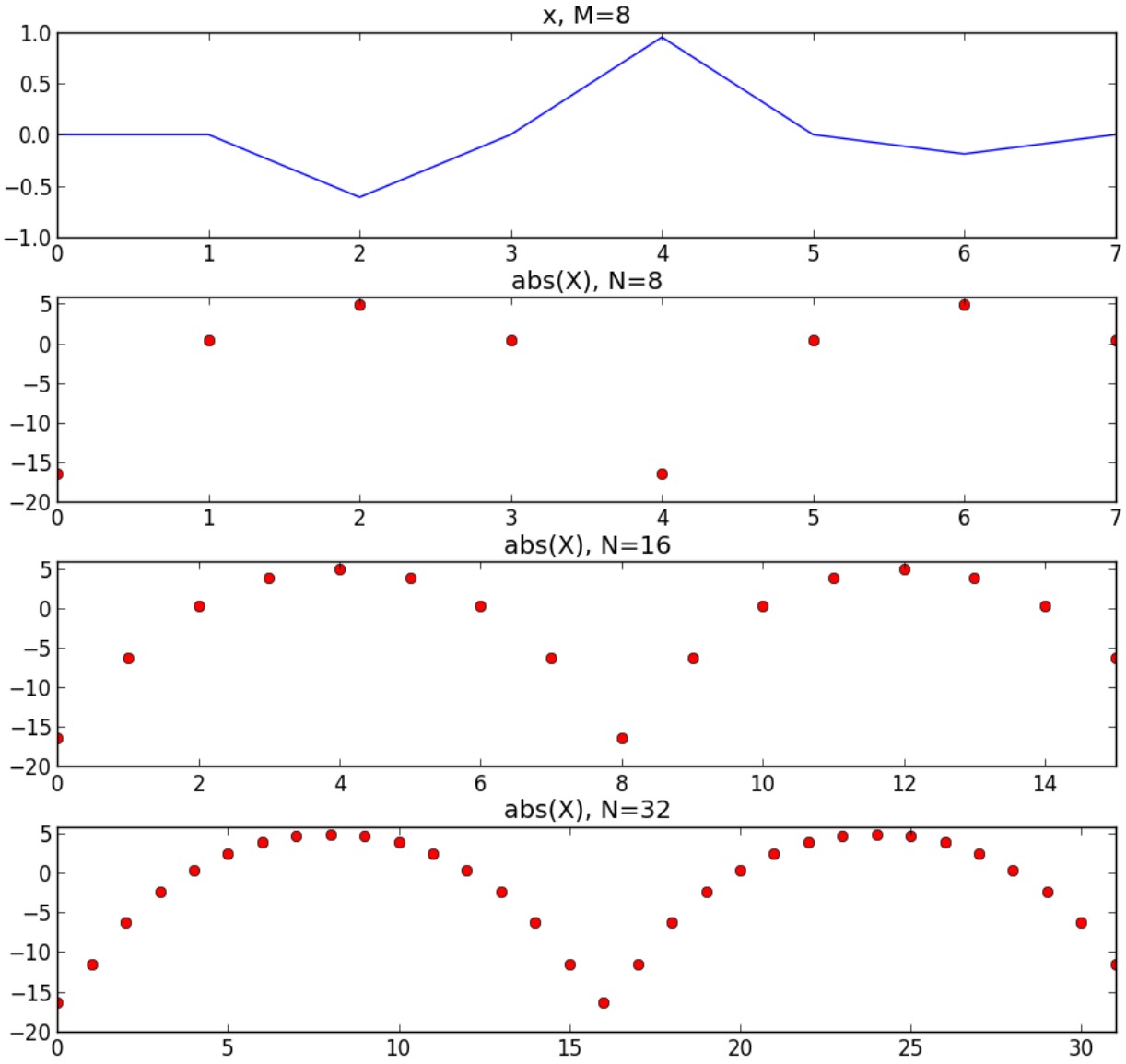
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.fftpack import fft, fftshift

M = 64
N = 512
x1 = np.hanning(M)
x2 = np.cos(2*np.pi*2/M*np.arange(M))
y1 = x1*x2
mY1 = 20 * np.log10(np.abs(fftshift(fft(y1, M))))
plt.subplot(3,2,1)
plt.plot(x1)
plt.subplot(3,2,3)
plt.plot(x2)
plt.subplot(3,2,5)
plt.plot(mY)

Y2 = np.convolve(fftshift(fft(x1, M)), fftshift(fft(x2, M)))
mY2 = 20 * np.log10(np.abs(Y2))
mX1 = 20 * np.log10(np.abs(fftshift(fft(x1, M)))/M)
mX2 = 20 * np.log10(np.abs(fftshift(fft(x2, M)))/M)
plt.subplot(3,2,2)
plt.plot(mX1)
plt.subplot(3,2,4)
plt.plot(mX2)
plt.subplot(3,2,6)
plt.plot(mY2[M/2:M+M/2])
```

Phase unwrapping



Zero-padding

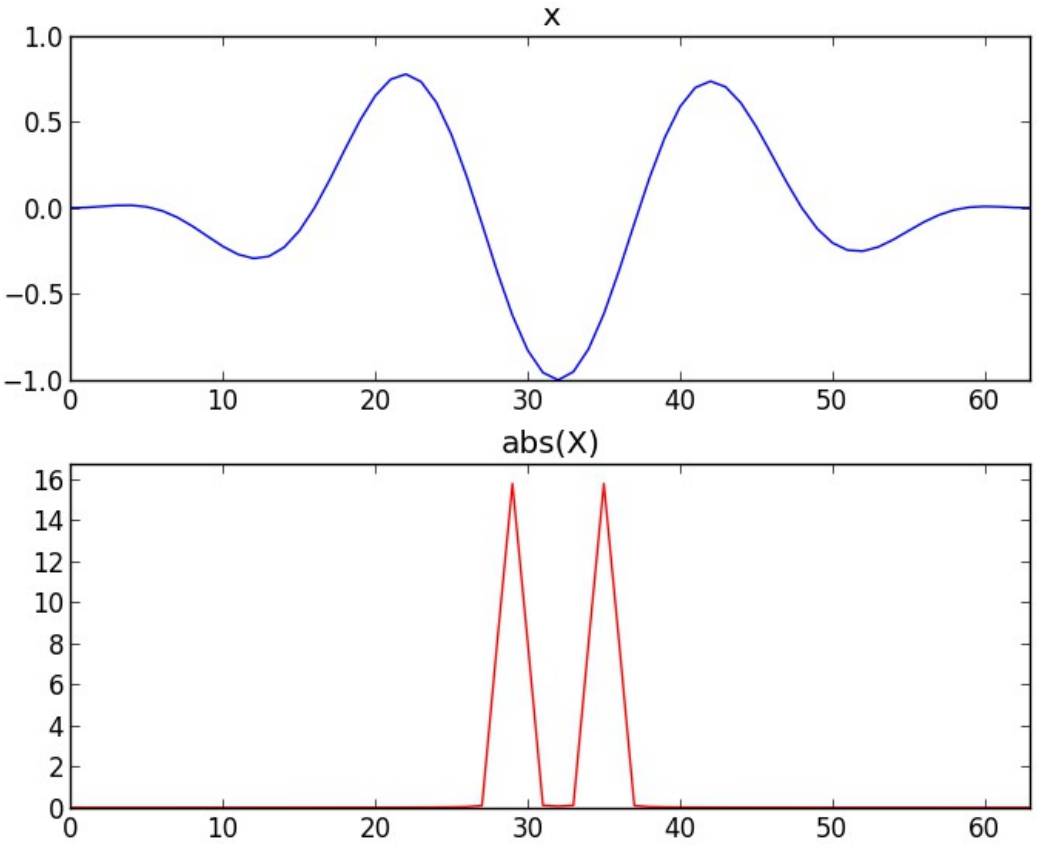
zero padding $\leftrightarrow$ interpolation



```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import hamming
from scipy.fftpack import fft, fftshift

M = 8
N1 = 8
N2 = 16
N3 = 32
x = np.cos(2*np.pi*2/M*np.arange(M)) * np.hanning(M)

plt.subplot(4,1,1)
plt.plot(x, 'b')

mX = 20 * np.log10(np.abs(fftshift(fft(x, N1))))
plt.subplot(4,1,2)
plt.plot(mX, 'ro')

mX = 20 * np.log10(np.abs(fftshift(fft(x, N2))))
plt.subplot(4,1,3)
plt.plot(mX, 'ro')

mX = 20 * np.log10(np.abs(fftshift(fft(x, N3))))
plt.subplot(4,1,4)
plt.plot(mX, 'ro')
```

Power

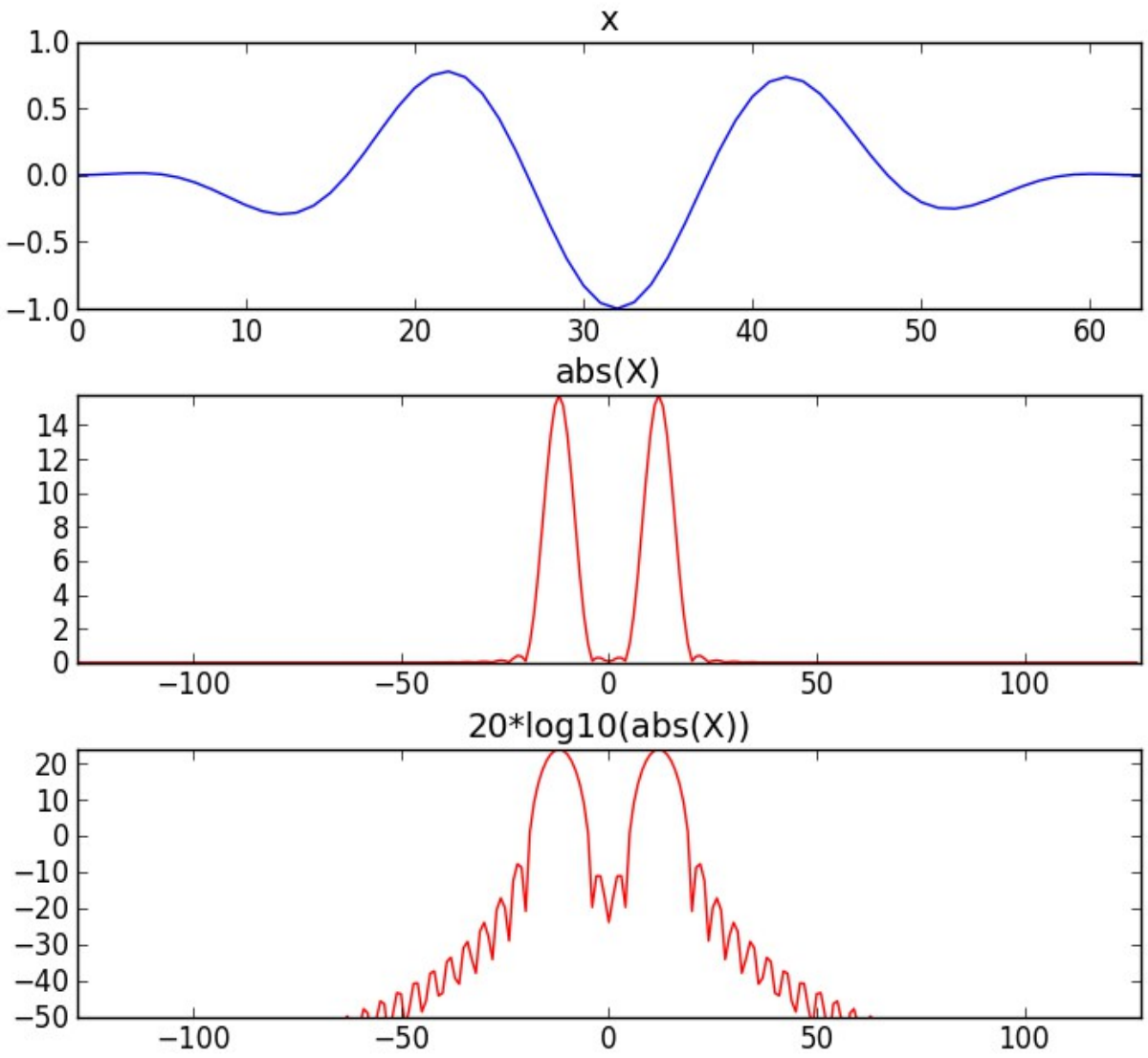
$$\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2$$



$$\sum_{n=0}^{N-1} |x(n)|^2 = 11.81182$$

$$\frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2 = 11.81182$$

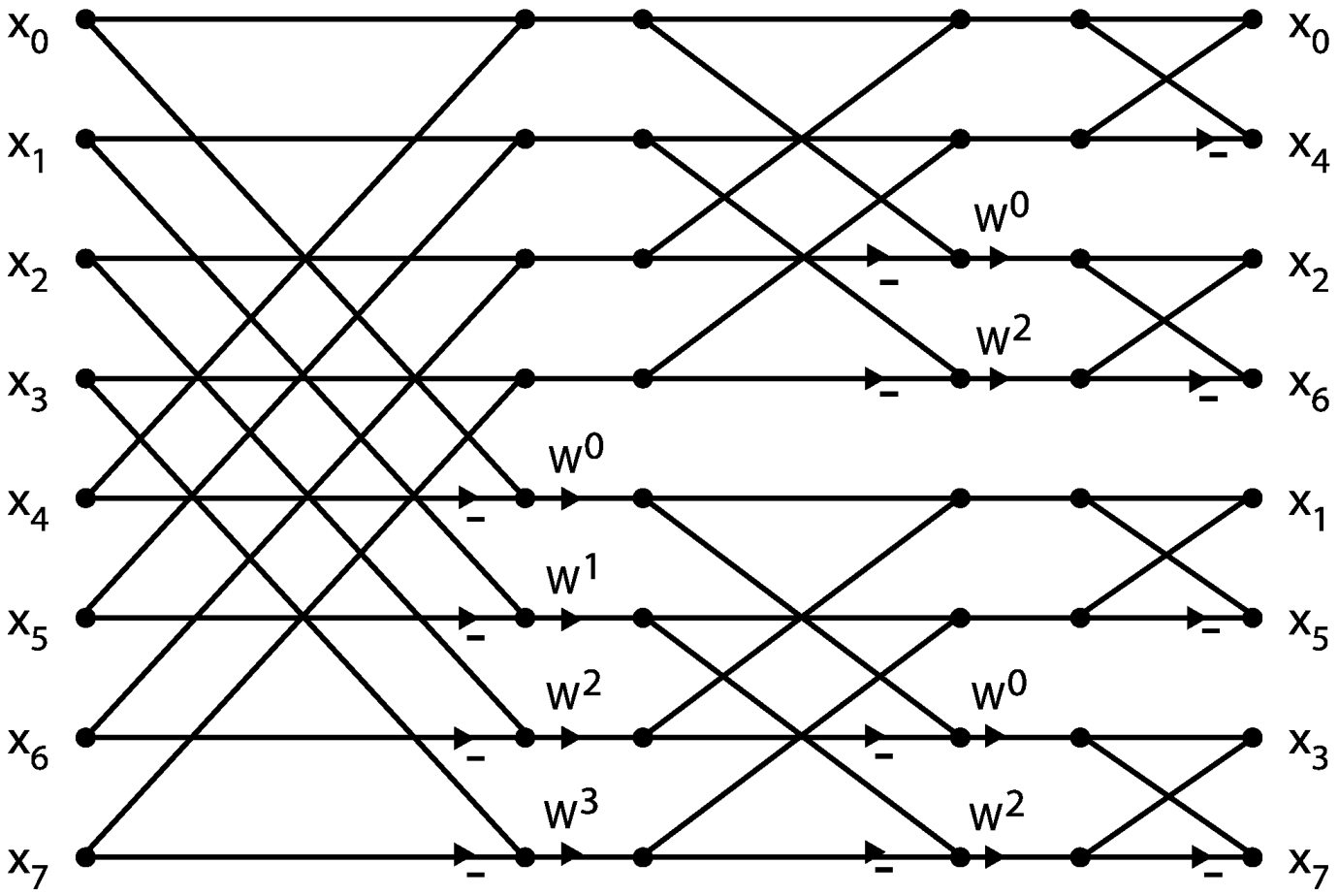
Amplitude in Decibels

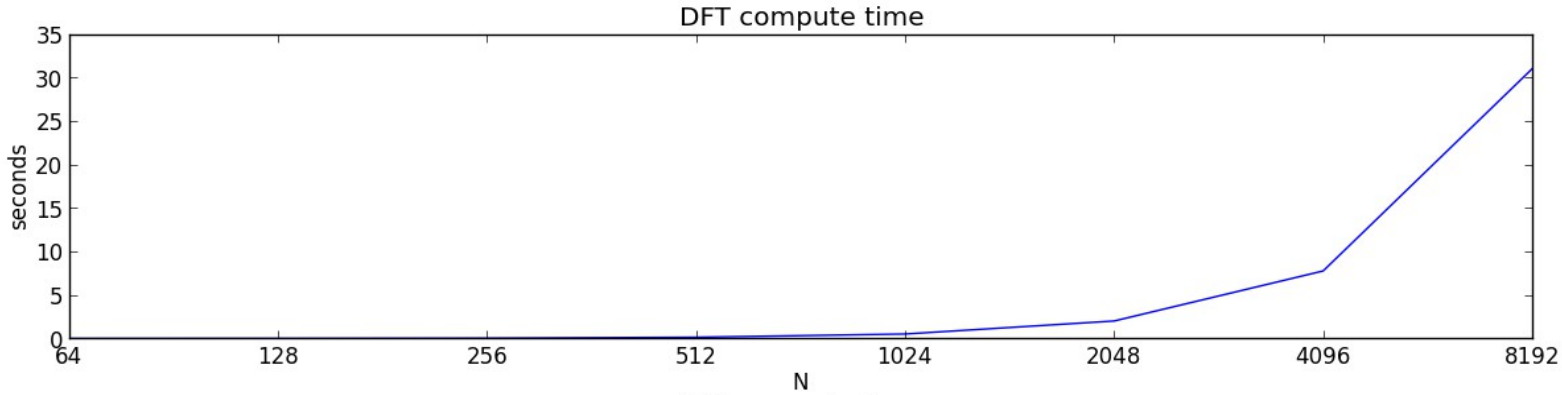


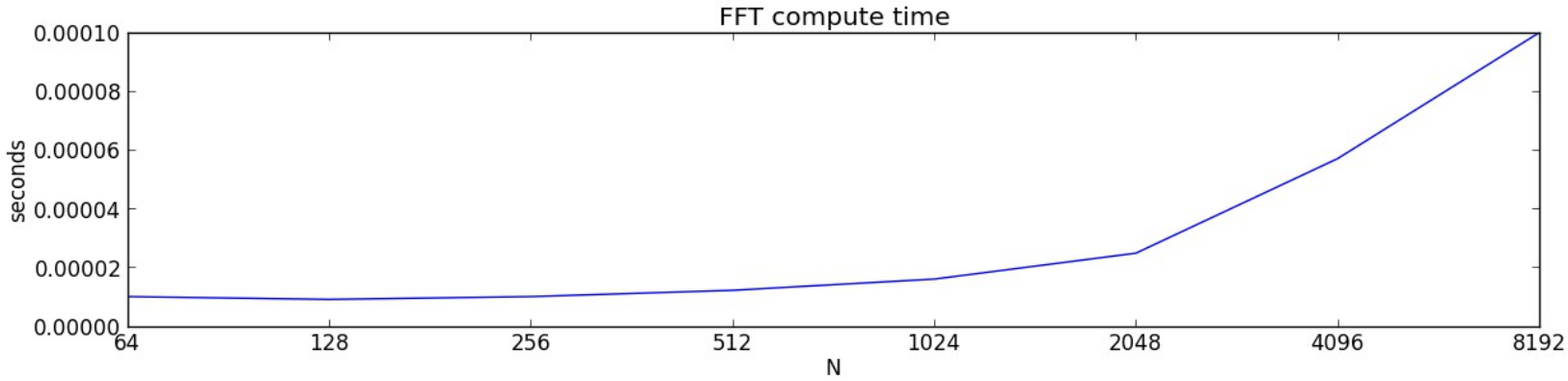
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftshift

M= 64
N = 256
x = np.cos(2*np.pi*3/M*np.arange(M)) * np.hanning(M)

plt.subplot(3,1,1)
plt.plot(x, 'b')

mX = np.abs(fftshift(fft(x, N)))
plt.subplot(3,1,2)
plt.plot(np.arange(-N/2,N/2,1), mX, 'r')

mX = 20 * np.log10(mX)
plt.subplot(3,1,3)
plt.plot(np.arange(-N/2,N/2,1), mX, 'r')
```

Fast Fourier Transform

The most common version uses the Coley-Tukey algorithm. It breaks down recursively the DFT of a power of 2 size into two pieces of size N/2.








```
Ns = 2**np.arange(6,14)
for N in Ns:
    str_time = time.time()
    for k in range(N):
        s = np.exp(1j*2*np.pi*k/N*np.arange(N))
        X = np.append(X, sum(x*np.conjugate(s)))
    timeDFT = np.append(timeDFT, time.time()-str_time)

for N in Ns:
    X = fft(x)
    str_time = time.time()
    X = fft(x)
    timeFFT = np.append(timeFFT, time.time()-str_time)
```

References

- https://ccrma.stanford.edu/~jos/mdft/
- https://en.wikipedia.org/wiki/Fast_Fourier_transform
- Full code of plots and accompanying labs available at:https://github.com/MTG/sms-tools

Credits