Questions

Any questions to exercises and homeworks from last time?

Table of Content

Single Chirp LFMCW

- Approximate IF signal for an LFMCW chirp (from lecture):

$$s_{\mathrm{IF}}[n] \approx \sum_{m=0}^{M-1} A_m \cos\left(2\pi\phi_m[n]\right) + w[n]$$

$$\phi_m[n] = \frac{2f_c r_0[m]}{c_0} + nT_{\mathrm{S}}\frac{2kr_0[m] + 2f_c v_0[m]}{c_0} + n^2 T_{\mathrm{S}}^2 \frac{2kv_0[m]}{c_0}$$

System Parameters		*Environment*	
f_c	carrier frequency	$r_{0,m}$	ranges of targets
B	bandwidth	$v_{0,m}$	range-rates of targets
T	chirp duration	$A_{0,m}$	magnitude of targets
$k = B/T$	ramp slope	$w[n]$	additive noise
T_{S}	sampling interval		

- The range-profile can be obtained as PSD of $s_{\mathrm{IF}}[n]$ using the FFT and windowing.

Noise in the Radar System

- The noise in a radar system is often given as a *system's noise figure* F in dB, e.g. $F_{\mathrm{dB}} = 12\,\mathrm{dB}$.

$$F = 10^{F_{\mathrm{dB}}/10} = 1 + \frac{T_{\mathrm{e}}}{T_0}$$

- Noise power can be calculated with the equivalent noise temperature.

$$P_{\mathrm{N}} = k_{\mathrm{B}}\,T_{\mathrm{e}}/\,T_{\mathrm{obs}}$$

- Noise process in signal model represents a voltage.

$$V_{\mathrm{N}} = \sqrt{P_{\mathrm{N}} R} \qquad w \sim \mathcal{N}(0, \sigma^2) \quad \sigma = V_{\mathrm{N}}$$

F	noise figure	T_{obs}	observation time
T_0	environmental temperature	P_{N}	noise power
T_{e}	equivalent noise temperature	R	reference impedance

Exercise: FMCW: Single Chirp

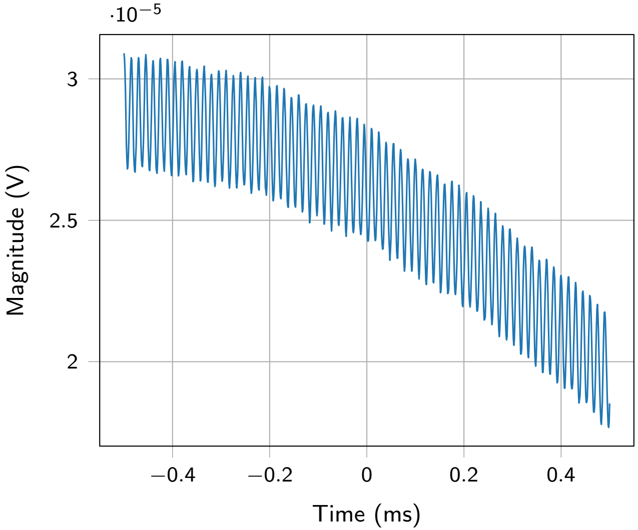
- A single chirp with the following Parameters:

```
fs=1e6          # IF sample frequency
B=250e6         # RF bandwidth
fc=24.125e9     # carrier frequency
T=1e-3          # chirp duration
F_dB=12         # noise figure of the system
T0=270          # system temperature
R=50            # reference impedance
```

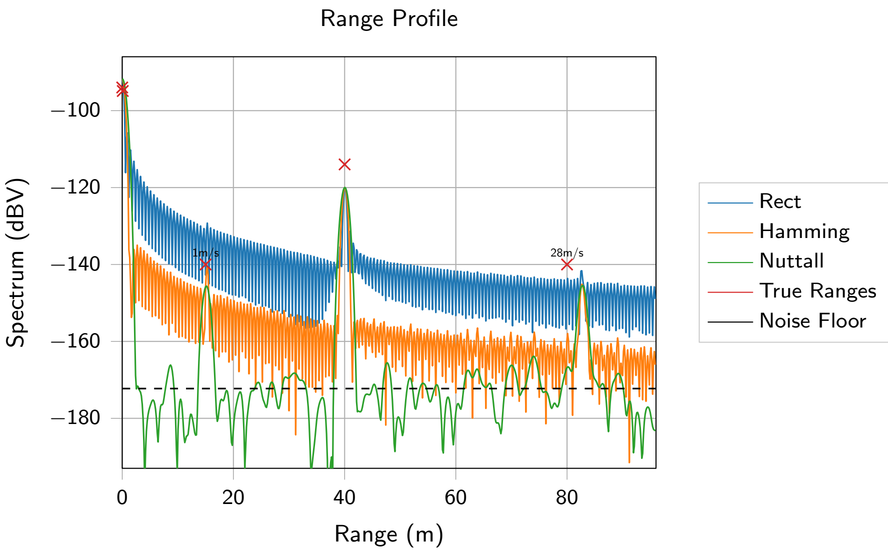
- Two close (spurious) targets and three wanted targets:

```
A0_arr=np.array([20, 18, 0.1, 2, 0.1])*1e-6  # magnitudes
r0_arr=np.array([0.001, 0.1, 15, 40, 80])    # ranges
v0_arr=np.array([0, 0, 1, 0, 28])            # range-rates
```

- Calculate and plot the sampled IF signal. Where is the noise and where are the targets visible?
- Plot the spectrum using a rect, a hamming and a nuttal window and place marks for the true range/magnitude pairs. Why do the marks not match the peaks?



$\cdot 10^{-5}$
3
2.5
2
Magnitude (V)
−0.4
−0.2
0
0.2
0.4
Time (ms)



Range Profile
Spectrum (dBV)
Range (m)
−100
−120
−140
−160
−180
0
20
40
60
80
1m/s
28m/s
Rect
Hamming
Nuttall
True Ranges
Noise Floor

Multidimensional Numpy Arrays

- Each `ndarray` has a `shape` property (tuple of length `ndmin`)
- The built-in function `len` returns length of first dimension.
- Lists, row, and column vectors:

```
a=np.array(range(3)) # a.shape==(3,);  len(a)==3; a.size==3; a.ndim==1
b=np.zeros((3,))     # b.shape==(3,);  len(b)==3; b.size==3; b.ndim==1
c=np.zeros((3,1))    # c.shape==(3,1); len(c)==3; c.size==3; c.ndim==2
d=np.zeros((1,3))    # d.shape==(1,3); len(d)==1; d.size==3; d.ndim==2
```

- Slicing and indexing returns lists instead of vectors

```
a=np.random.rand(4,4)
a[:,3].shape  # returns (4,)
```

- Slices and elements can be assigned individually with numbers, lists or other arrays:

```
a[0,0]=1.0
a[:,0]=[3,2,1]
```

- See https://docs.scipy.org/doc/numpy/user/basics.indexing.html for all indexing options.

Changing Shape and Size of an Array

- Change shape of an array
  - `a.T` (short for `a.transpose()` returns the view for the transpose
  - Transpose on list does nothing, but also produce no error
  - `np.reshape(...)` can be used to reshape
  - `a.resize(...)` can be used to reshape in-place
  - `np.newaxis` can be used to add a dimension

```
a=np.arange(3)     # produces a 0D-array
b=a.T              # b is of same shape as a
c=a[np.newaxis,:] # c is shape (1,3)
d=a[:,np.newaxis] # d is shape (3,1)
e=d.T              # e is shape (1,3)
```

- Combine multiple arrays to a bigger one:
  - `np.concatenate(...)` joins arrays along an existing axis
  - `np.stack(...)` joins arrays and adds a new axis
  - `np.vstack(...)`, `np.hstack` joins arrays horizontally (axis 1) or vertically (axis 0)

Views and Copies

- numpy avoids copying data in memory for performance reasons.
- Many operations that do not change content return a "view" instead of a copy, e.g., shape changes or slicing with ":".

```
a=np.random.rand(3,3) # generate a (3,3) array
b=a.T                 # generate a transposed view
b[2,1]=np.nan         # change one element in the view
a[:,1]=np.inf         # change a column in the original array
print('a =', a); print('b =', b)    # compare arrays
```

```
a = [[0.47433699    inf  0.76066986]
     [0.3603668     inf         nan]
     [0.68291021    inf  0.8093564 ]]
b = [[0.47433699  0.3603668  0.68291021]
     [       inf        inf         inf]
     [0.76066986        nan  0.8093564 ]]
```

- Best: Consult docstrings for return values of numpy functions.
- Or: Check if a view was generated:

```
b.base is a          # True if b is a view of a (a must be known)
b.flags['OWNDATA'] # False if b is a view
```

Broadcasting

- Broadcasting is auto-fillup of arrays if dimensions do not match to a given operation.
- https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html

```
>>> np.full((3,1),1.0) + np.full((1,3),1.0)+2.0
array([[4., 4., 4.],
       [4., 4., 4.],
       [4., 4., 4.]])
```

- Numpy's broadcast is quite flexible, i.e. seldom produces errors but result may be unexpected, e.g. when using lists

```
>>> np.full((3,1),1.0) + np.full((3,),1.0)
array([[2., 2., 2.],
       [2., 2., 2.],
       [2., 2., 2.]])
```

- Most operations (addition, multiplication, abs,...) are element-wise.
  - True matrix multiplication provided by `@`.
  - Other matrix operations are in `numpy.linalg`.

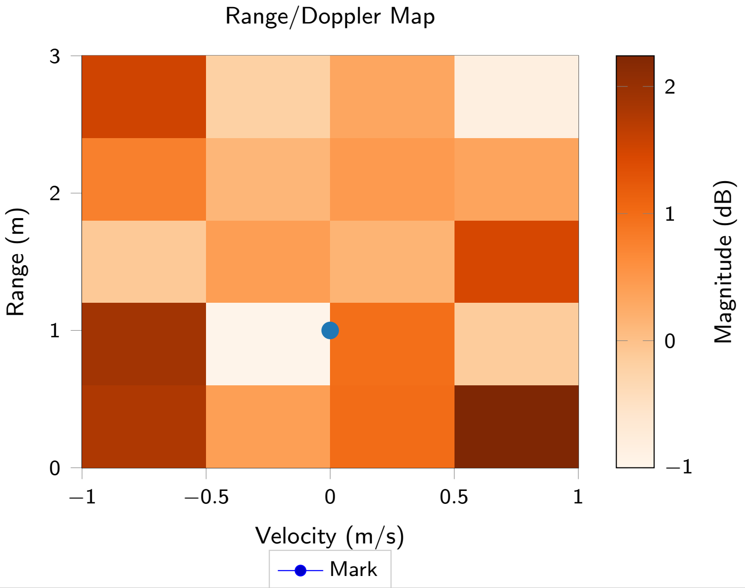
Plot 2D Array Using Matplotlib's imshow

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib2tikz import save as tikz_save

np.random.seed(0)
Z=np.random.randn(5,4)
plt.imshow(Z,
           cmap='Oranges',
           vmin=-1,
           origin='lower', aspect='auto',
           extent=(-1,1,0,3))
plt.xlabel('Velocity (m/s)')
plt.ylabel('Range (m)')
plt.colorbar(label='Magnitude (dB)')
plt.title('Range/Doppler Map')
plt.plot(0, 1, 'o', label='Mark')
plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.3))

tikz_save('test_imshow.tikz',
          tex_relative_path_to_data='python',
          override_externals=True)
plt.savefig('test_imshow.png', dpi=150)
```

Plot 2D Array Using Matplotlib's imshow



Surface Plots with Matplotlib

- 3D plots are an extension to `matplotlib`. Documentation:
  https://matplotlib.org/mpl_toolkits/mplot3d/api.html
- using `%matplotlib qt` from spyder provides interaction

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

# generate data
x=np.arange(-15,15); y=np.arange(-20,20)                            # generate axis
X, Y = np.meshgrid(x, y)                                             # generate grid
A=np.cos(2*np.pi*1/x.size*X)*np.cos(2*np.pi*0.25/y.size*Y)**2 # generate data

# init figure
fig=plt.figure(1, figsize=[9,6])   # default figure size is too small for 3D
fig.clear()                        # reusing same figure with %matplotlib qt
ax=fig.gca(projection='3d')        # set axes to a 3D axis
ax.view_init(elev=27, azim=-66)    # set view
plt.tight_layout(pad=0.0,rect=(-0.1,0,1,1)) # adjust boarders around axes
```

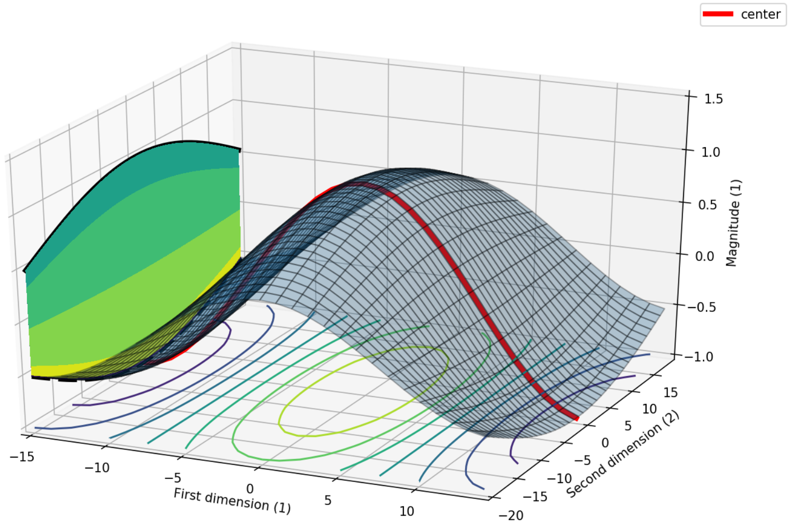
Surface Plots with Matplotlib

```
# plot objects
ax.plot_surface(X, Y, A, alpha=0.3, edgecolor='k')  # surface plot
ax.contour(     X, Y, A, zdir='z', offset=-1)       # contour at bottom
ax.contourf(    X, Y, A, zdir='x', offset=x[0])     # filled contour side

# plot lines
ax.plot3D(x, np.full(x.shape,np.mean(y)), zs=A[A.shape[0]//2,:],
          color='r', linewidth=4, label='center')
ax.plot3D(np.full(y.shape,x[0]), y, zs=np.max(A,axis=1),
          color='k', linewidth=4)
ax.plot3D(np.full(y.shape,x[0]), y, zs=np.min(A,axis=1),
          linestyle='--', color='k', linewidth=4)

# annotate plot
plt.xlim(x[0],x[-1]); plt.ylim(y[0],y[-1]) # to place contours being in planes
ax.set_zlim(bottom=-1.0, top=1.5*np.max(A))# zlim is only available in axis
ax.set_zlabel('Magnitude (1)')             # same for zlabel
plt.xlabel('First dimension (1)'); plt.ylabel('Second dimension (2)')
plt.legend()                               # legend not avail. for all items
plt.savefig('test_3D_plot.png', dpi=150)
```

Surface Plot Result



Generating a 2D Signal

$$s[n_A, n_B] = A \sin(2\pi(\psi_A n_A, + \psi_B n_B) + \phi)$$

- Define sample vectors as numpy arrays

```
nA=np.array(range(NA))            # 0-D array of integers, 0 to NA-1
nA=np.arange(NA))                 # 0-D array of integers, 0 to NA-1
nA=np.linspace(0, NA-1, num=NA,
               endpoint=True)     # 0-D array of floats, 0 to NA-1
```

- Using meshgrids as in the 3D plot example.
- Assemble 2D signal using loops

```
s=np.zeros(NA,NB)
for na in range(NA):
  for nb in range(NB):
    s[na,nb]=A*np.sin(2*np.pi*(psia*na + psib*nb + phi))
```

- Assemble using broadcasting

```
na=na[:,np.newaxis] # make a column vector
nb=nb.reshape(1,NB) # make a row vector (not strictly required)
s=A*np.sin(2*np.pi*(psia*na + psib*nb + phi))
```

Sum up Many 2D Signals

$$s[n_\mathrm{A}, n_\mathrm{B}] = \sum_{m=0}^{M-1} A_m \sin(2\pi(\psi_{\mathrm{A},m} n_\mathrm{A}, + \psi_{\mathrm{B},m} n_\mathrm{B}) + \phi_m)$$

- Assemble using broadcasting (memory consuming)

```
na=na[:,np.newaxis] # make a column vector
nb=nb.reshape(1,NB) # make a row vector (not strictly required)
A=A.reshape(1,1,A.size)               # make array in third dimension
psia=psia.reshape((1,1,psia.size)) # make array in third dimension
psib=psib.reshape((1,1,psib.size)) # make array in third dimension
phi=phi.reshape((1,1,phi.size))    # make array in third dimension
s=np.sum(A*np.sin(2*np.pi*(psia*na + psib*nb + phi)), axis=3)
```

- Combine loops and broadcasting (most efficient)

```
na=na[:,np.newaxis] # make a column vector
nb=nb.reshape(1,NB) # make a row vector (not strictly required)
s=np.zeros(NA,NB)   # initialize s to make first addition work
for i in range(M):
    s += A[m]*np.sin(2*np.pi*(psia[m]*na + psib[m]*nb + phi[m]))
```

The Two-Dimensional FFT

- A two-dimensional sinusoid

$$s[n, n_\mathrm{P}] = A\cos\left(2\pi\left(\psi_0 n + \psi_1 n_\mathrm{P} + \phi_0\right)\right)$$

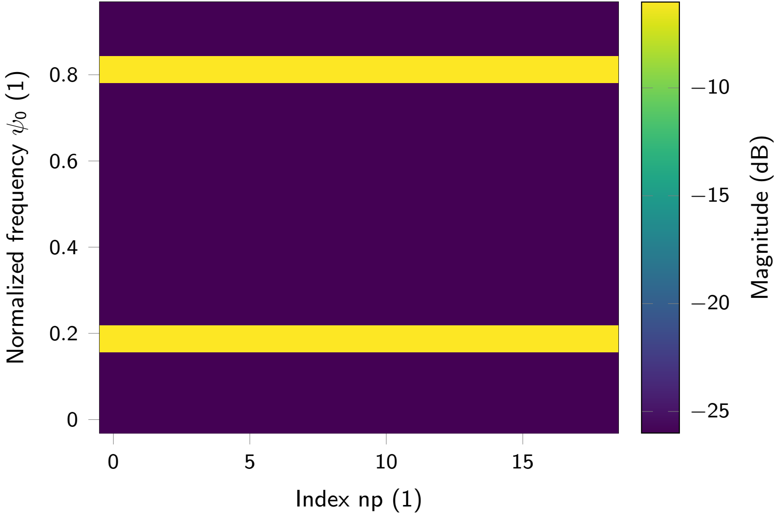
- Correlation with exponentials can be done in each dimension separately. Fourier Transform and maximum search can be applied in each dimension separately.

$$P(\psi_0, \psi_1) = \left| \frac{\sum_{n=0}^{N-1} \sum_{n_\mathrm{P}=0}^{N_\mathrm{P}-1} a[n, n_\mathrm{P}] s[n, n_\mathrm{P}] \mathrm{e}^{-\mathrm{j}2\pi\psi_0 n} \mathrm{e}^{-\mathrm{j}2\pi\psi_1 n_\mathrm{P}}}{\sum_{n=0}^{N-1} \sum_{n_\mathrm{P}=0}^{N_\mathrm{P}-1} a[n, n_\mathrm{P}]} \right|^2$$

- As FFT is linear, i.e., order (of sums) does not matter.
- Most of Python's FFT modules provide an `fft2` or `fftn`.
- Zero-padding and windowing can be applied to both FFTs individually.
- `fftshift(x, axes=None)` has an axes options.

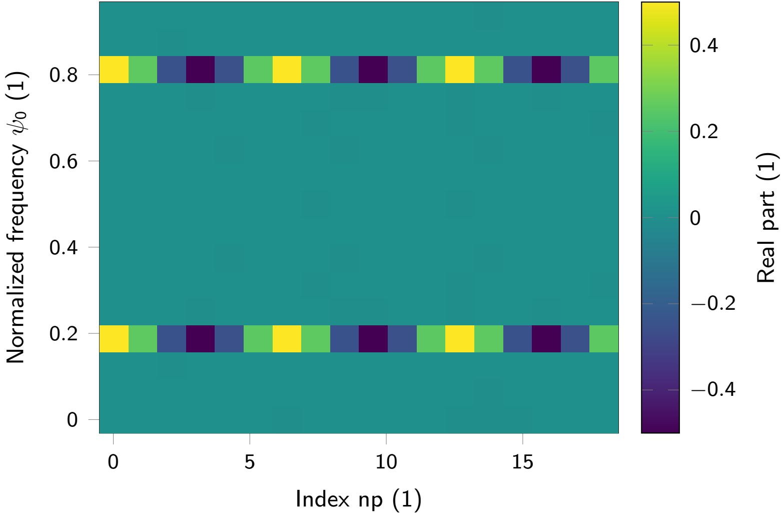
Visualized Steps of 2D FFT

Power spectrum after applying the FFT along the first dimension for all n_P.



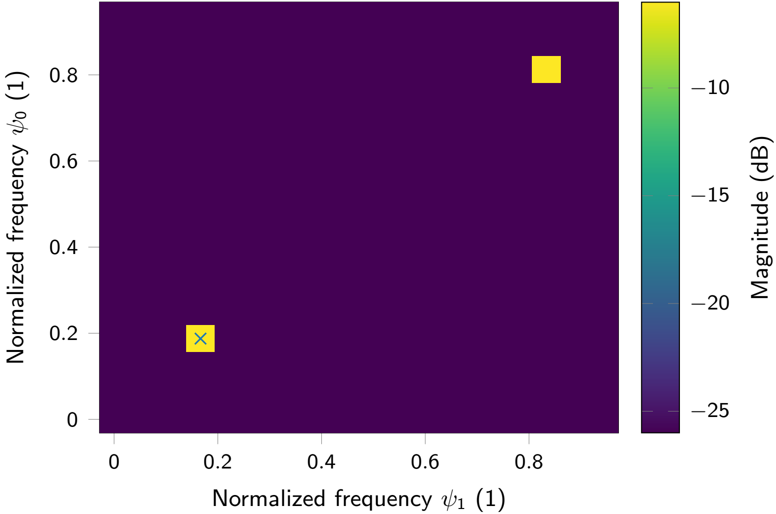
Visualized Steps of 2D FFT

Real values of the spectrum after applying the FFT along the first dimension for all n_P.



Visualized Steps of 2D FFT

Power spectrum after applying the FFT on both dimensions.



Range/Doppler (RD) processing

- The two-dimensional approximated IF signal for RD processing:

$$s_{\mathrm{IF}}[n, n_{\mathrm{P}}] \approx \sum_{m=0}^{M-1} A_m \cos\left(2\pi\phi_m[n, n_{\mathrm{P}}]\right) + w[n, n_{\mathrm{P}}]$$

$$\begin{aligned}\phi_m[n, n_{\mathrm{P}}] = & n_{\mathrm{P}} \overbrace{T_{\mathrm{P}} \frac{2 f_c v_0[m]}{c_0}}^{\psi_P} + n \overbrace{T_{\mathrm{S}} \left(\frac{2 k r_0[m]}{c_0} + \frac{2 f_c v_0[m]}{c_0}\right)}^{\psi_N} + \\ & + \frac{2 f_c r_0[m]}{c_0} + n^2 T_{\mathrm{S}}^2 \frac{2 k v_0[m]}{c_0} + n n_{\mathrm{P}} T_{\mathrm{S}} T_{\mathrm{P}} \frac{2 k v_0[m]}{c_0}\end{aligned}$$

n_{P}	index of a pulse
$T_{\mathrm{P}} > T$	pulse repetition interval

- Range/Doppler map is the PSD of $s_{\mathrm{IF}}[n, n_{\mathrm{P}}]$, obtainable via 2D-FFT using a two-dimensional window function.

Negligible Terms

$$\phi_{\mathrm{IF,CV}} = 2f_{\mathrm{c}}\frac{r_0}{c_0} + \underbrace{2f_{\mathrm{c}}\frac{v_0 T_{\mathrm{P}}}{c_0}}_{\psi_{\mathrm{P}}} n_{\mathrm{P}} + \underbrace{\left(\underbrace{2f_{\mathrm{c}}\frac{v_0 T_{\mathrm{S}}}{c_0}}_{\psi_{\mathrm{N,v}}} + \underbrace{2k\frac{r_0 T_{\mathrm{S}}}{c_0}}_{\psi_{\mathrm{N,r}}}\right)}_{\psi_{\mathrm{N}}} n_{\mathrm{S}} +$$

$$\underbrace{2k\frac{v_0 T_{\mathrm{P}} T_{\mathrm{S}}}{c_0}}_{\varpi_{\mathrm{N,P}}} n_{\mathrm{P}} n_{\mathrm{S}} + \underbrace{2k\frac{v_0 T_{\mathrm{S}}^2}{c_0}}_{\varpi_{\mathrm{N,N}}} n_{\mathrm{S}}^2$$

- Typical values are $f_{\mathrm{c}} = 75\,\mathrm{GHz}$, $k = 10\,\mathrm{GHz/ms}$, $T_{\mathrm{P}} = 0.1\,\mathrm{ms}$, $T_{\mathrm{S}} = 0.1\,\mu\mathrm{s}$, $v_0 = 1\,\mathrm{m/s}$, $r_0 = 10\,\mathrm{m}$.
- With $c_0 = 3 \cdot 10^8\,\mathrm{m/s}$, this result in
  - $\psi_{\mathrm{P}} = 0.05$, $\psi_{\mathrm{N,v}} = 50 \cdot 10^{-6}$, $\psi_{\mathrm{N,r}} = 0.0\dot{6}$,
  - $\varpi_{\mathrm{N,P}} = 0.\dot{6} \cdot 10^{-6}$, $\varpi_{\mathrm{N,N}} = 0.\dot{6} \cdot 10^{-9}$.

Producing a Range/Doppler Map

- The range/Doppler map is a scaled (to range and range-rate) cutout of the 2D spectrum.
  - `np.fft.fft2(s_if, s=(Z0,Z1))` or, equivalently,
  - `np.fft.fft(np.fft.fft(s_if, n=Z0, axis=-2), n=Z1, axis=-1)`
- Magnitude scaling to dBV according to windowing.
- Negative frequencies in fast-time correspond to negative ranges, i.e. targets behind the radar.
  - Negative ranges are mostly of no interest and not plotted.
  - Crop data prior calculation of `abs`, `log`, or even prior FFT in slow-time dimension for computational efficiency.
- Frequencies in slow-time correspond to range-rates.
  - Positive and negative range-rates are of interest.
  - First element of FFT result is DC, i.e. zero range-rate.
  - Use `np.fft.fftshift(..., axis=X)` to bring zero to the middle of the array.

Range/Doppler Example

An FMCW Radar System

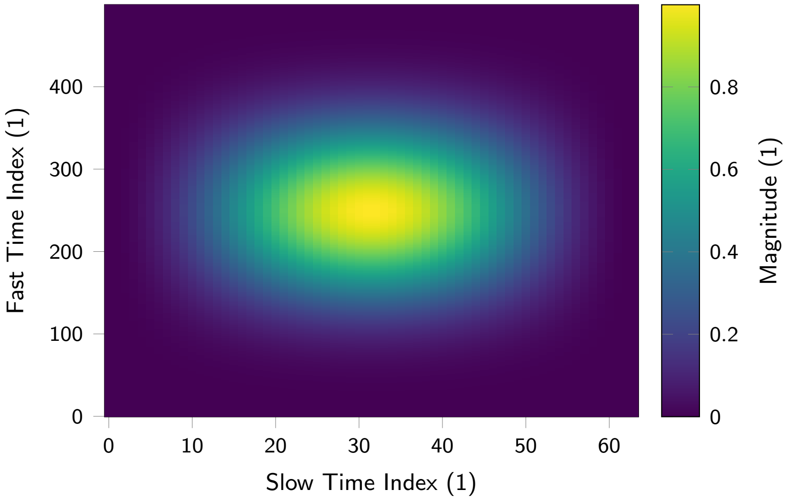
```
    fs=10e6          # IF sample frequency
    B=250e6          # RF bandwidth
    fc=24.125e9      # carrier frequency
    T=50e-6          # ramp duration
    Tp=100e-6        # chirp repetition rate
    Np=64            # number of pulses
    Temp_sys=9e5     # equivalent noise temperature
    R=50             # reference impedance
```

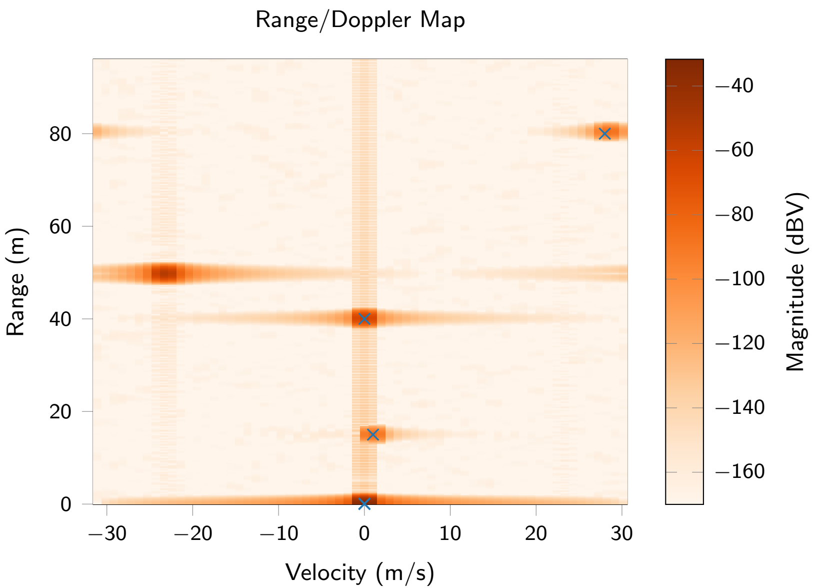
Automotive Targets

```
    A0_arr=np.array([20, 18, 0.1, 2, 0.1, 4])*1e-3   # magnitudes
    r0_arr=np.array([0.001, 0.1, 15, 40, 80, 50])    # ranges
    v0_arr=np.array([0, 0, 1, 0, 28, -84])           # velocities
```

Exercise: 2D-FFT

- Consider the radar system and targets from the previous slide.
- Calculate the IF signal $s_{\mathrm{IF}}[n, n_{\mathrm{P}}]$ for range/Doppler (RD) processing.
- Generate a 2D window using a nuttall window in range/fast-time and a hanning window in range-rate/slow-time dimension. Plot the result using `imshow` or `surface_3D`.
- Estimate the RD map using the FFT and the window function.
- Plot the RD map over range (m) and range-rate (m/s).
- Mark the true parameters in the map.
- What are the unambiguous regions in range and velocity dimension?
- Why does the fast moving target's peak appear wrongly positioned and rectangular?


Fast Time Index (1)
400
300
200
100
0
0
10
20
30
40
50
60
Slow Time Index (1)
0.8
0.6
0.4
0.2
0
Magnitude (1)


Range/Doppler Map
Range (m)
80
60
40
20
0
−30
−20
−10
0
10
20
30
Velocity (m/s)
−40
−60
−80
−100
−120
−140
−160
Magnitude (dBV)

Homework: Speed Considerations

- Calculation of the range/Doppler map in dB requires the following interchangeable steps:
  - cropping range
  - calculating the magnitude
  - calculation of the square of the magnitude
  - taking the logarithm
  - shifting velocity axis
  - applying scale for the window functions
- Comment on: How does the order of these operation might affect performance? Consider number of complex multiplication, number of real multiplications, memory accesses.
- Hint: no need to do benchmark or calculate the exact number of operations.