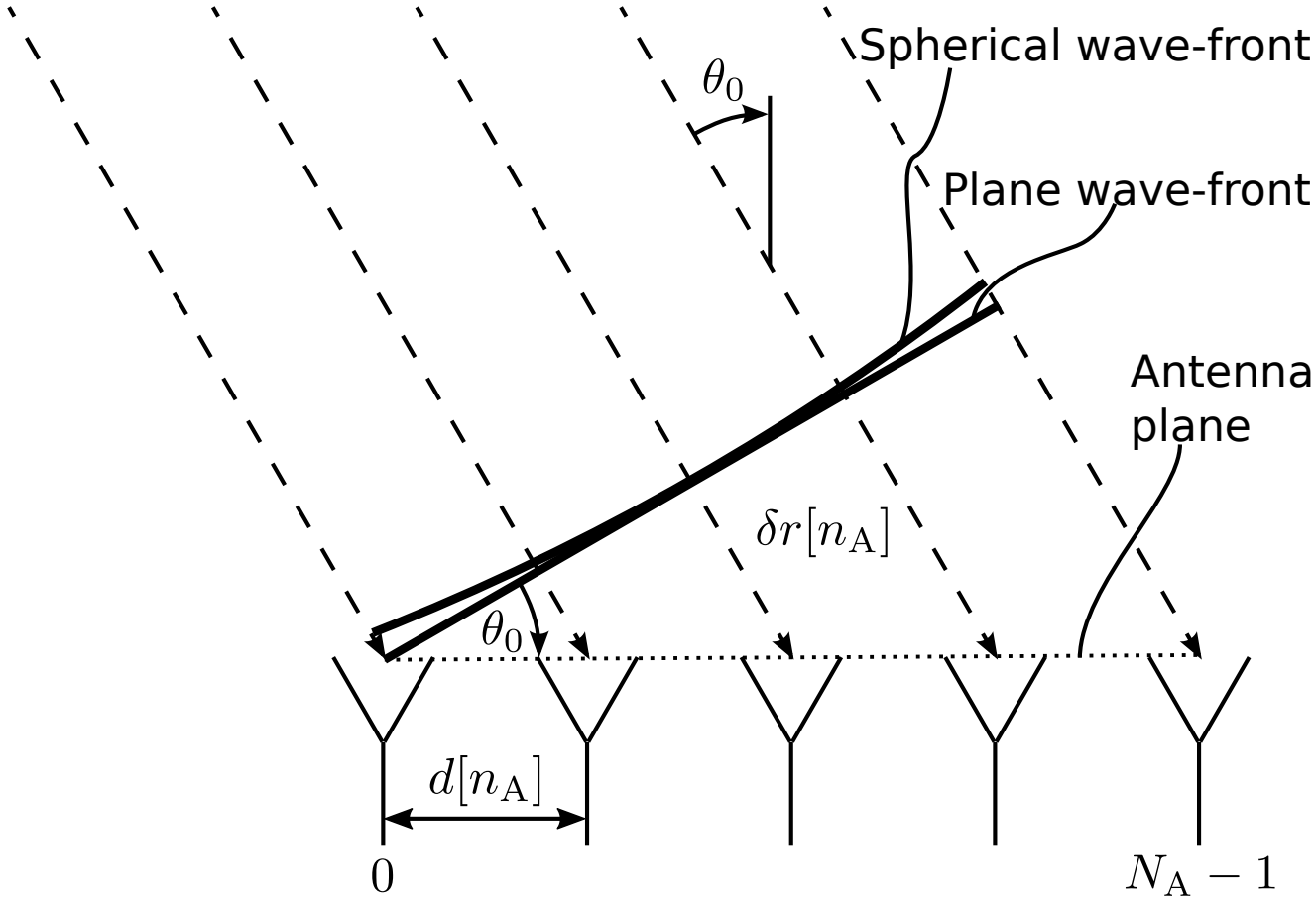
Questions

Any questions to exercises and homeworks from last time?

Table of Content

Linear Receive Array



Single Chirp, Multiple Antennas

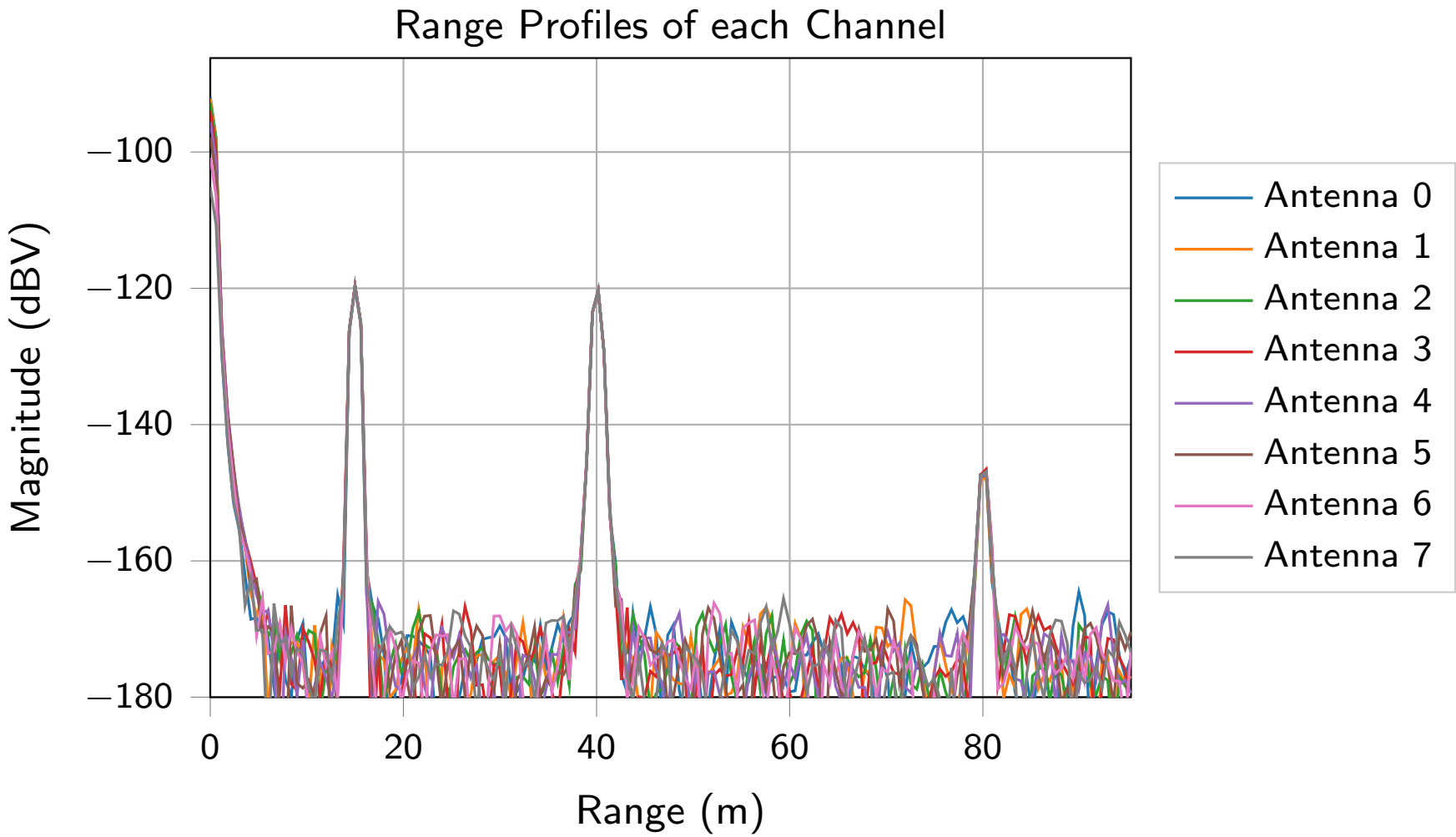
- Simplified signal model for a single chirp and a uniform linear array (ULA) with a spacing of $d = \lambda/2$

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

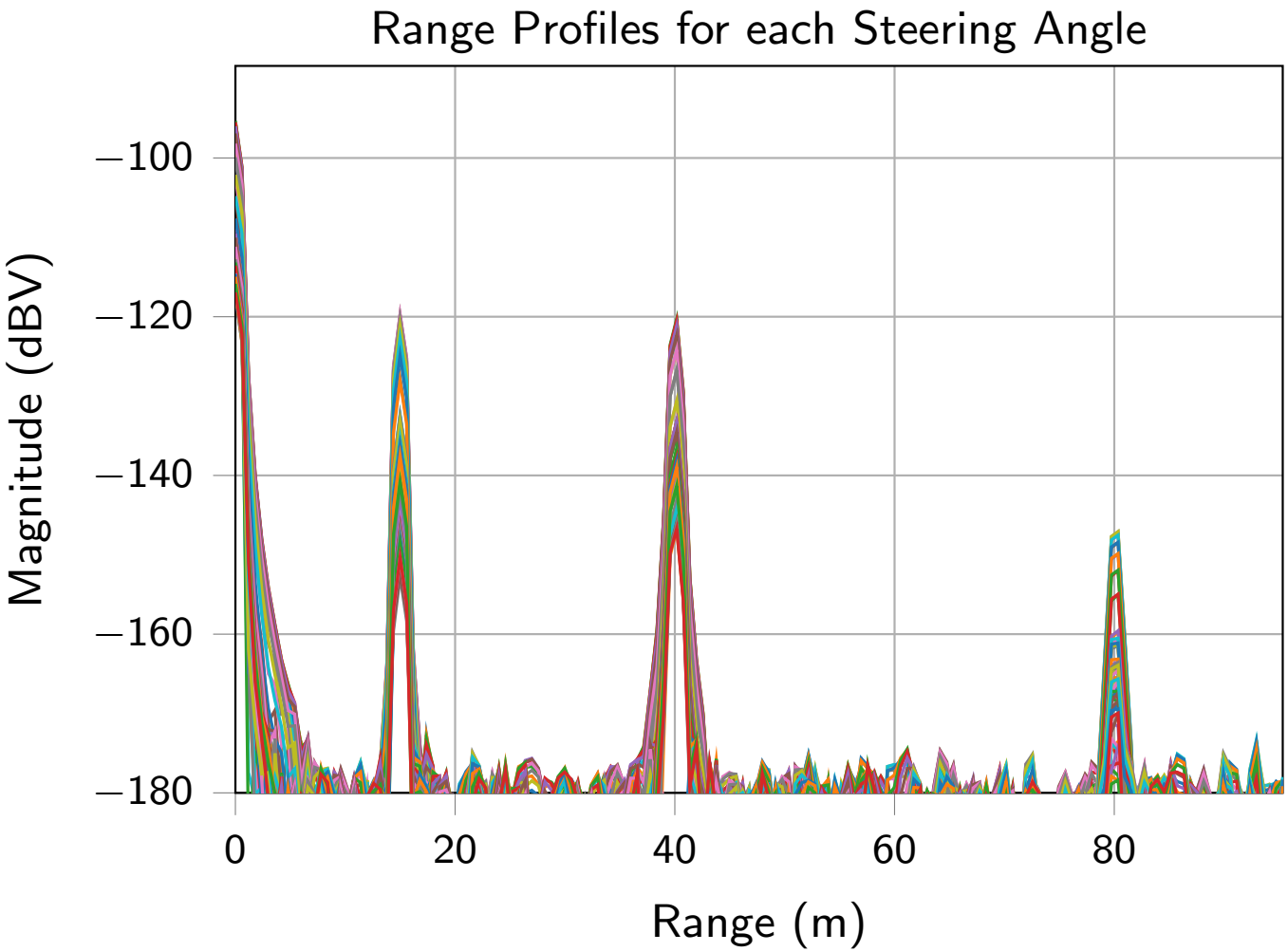
$$\phi_m[n, n_{\mathrm{A}}] = \frac{2f_c r_{0,m}}{c_0} + \underbrace{T_{\mathrm{S}}\frac{2kr_{0,m}}{c_0}}_{\psi_N} n + \underbrace{\frac{1}{2}\sin\left(\theta_{0,m}\right)}_{\psi_{\mathrm{A}},u} n_{\mathrm{A}}$$

System Parameters		*Environment*	
f_c	carrier frequency	$r_{0,m}$	ranges of targets
B	bandwidth	$\theta_{0,m}$	directions in rad of target
T	chirp duration	$A_{0,m}$	magnitude of targets
$k = B/T$	ramp slope	$w[n, n_{\mathrm{A}}]$	additive noise
T_{S}	sampling interval		

Range Profiles



Range Profiles



Exercise: FMCW: Single Chirp with ULA

- Consider a single chirp with the following Parameters (you can reuse parts from exercise "FMCW: Single Chirp"):

```
fs=1e6; B=250e6; fc=24.125e9; T=1e-3; F_dB=12; T0=270; R=50
NA=8          # number of channels
```

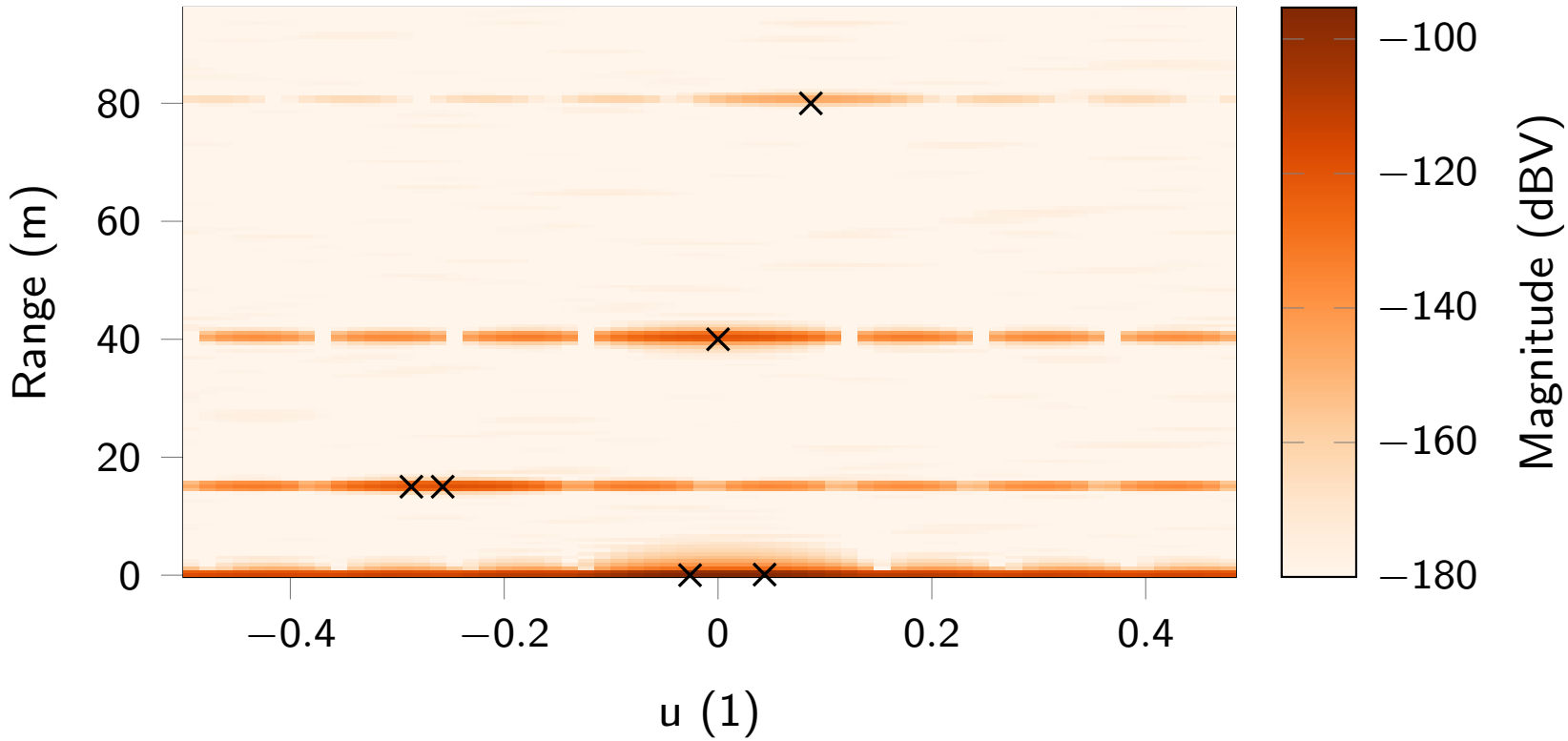
- Two close (spurious) targets and four wanted targets:

```
A0_arr=np.array([20, 18, 0.1, 2, 2, 0.1])*1e-6  # magnitude
r0_arr=np.array([0.001, 0.1, 15, 15, 40, 80])   # ranges
theta0_arr=np.array([-3, 5, -35, -31, 0, 10])   # angles (degree)
```

- Calculate the IF signal for all channels of a $\lambda/2$-spaced ULA.
- Use the 2D-FFT to estimate the spectrum
  - For hints on the 2D-FFT, see FMCW slides.
  - Use a hanning window in range-dimension and a rectangular window in direction of arrival dimension.

Exercise: FMCW: Single Chirp with ULA - FFT Spectrum

- Plot the result of the 2D-FFT for spectrum estimation
  - Set the lower plot limit to $-180\,\mathrm{dBV}$.
  - Mark the true locations.



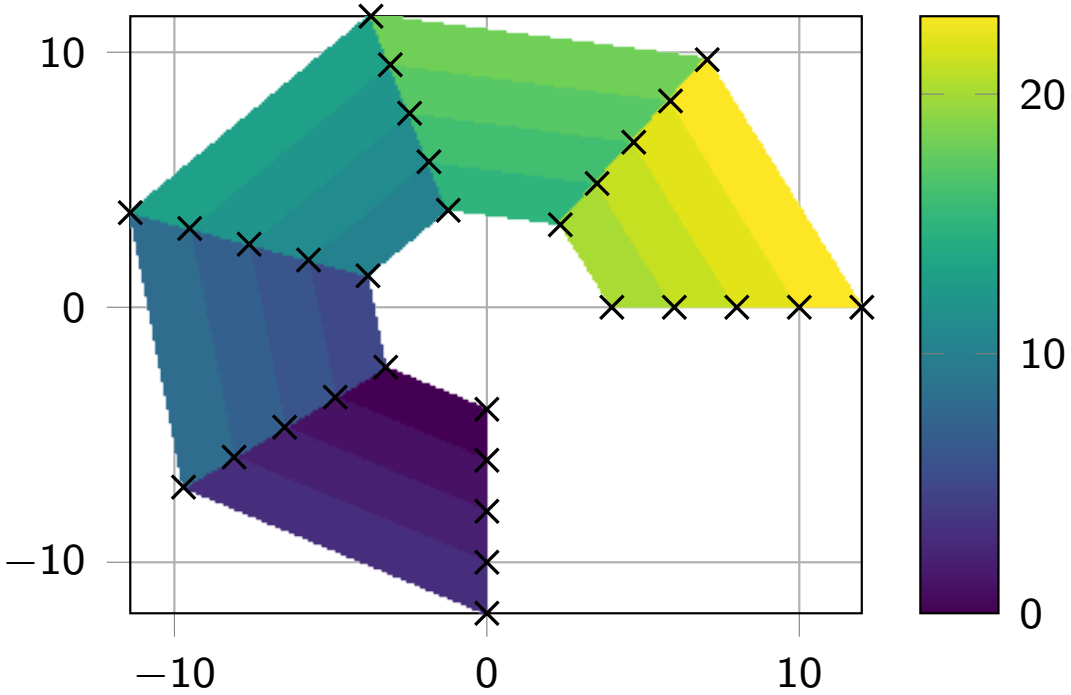
Pcolormesh Plots

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

data=np.reshape(np.arange(6*5),(6,5)) # dummy data

# map to a circular ring sector
theta=np.linspace(-np.pi, np.pi/2, data.shape[0])
theta=theta[:,np.newaxis]              # make theta a column vector
r=np.linspace(4, 12, data.shape[1])
X=r*np.sin(theta); Y=r*np.cos(theta) # list x colum vector produces a matrix
# plot
c=plt.pcolormesh(X, Y, data)           # plot it
plt.colorbar()                         # add colorbar
plt.plot(X, Y, 'kx')                   # combination with other plots possible
plt.grid()                             # common plot formatting stuff works too
# export
tikz_save('test_pcolormesh.tikz',
          tex_relative_path_to_data='python',
          override_externals=True)
plt.savefig('test_pcolormesh.png', dpi=150)
```

Relation between Coordinates and Data Array

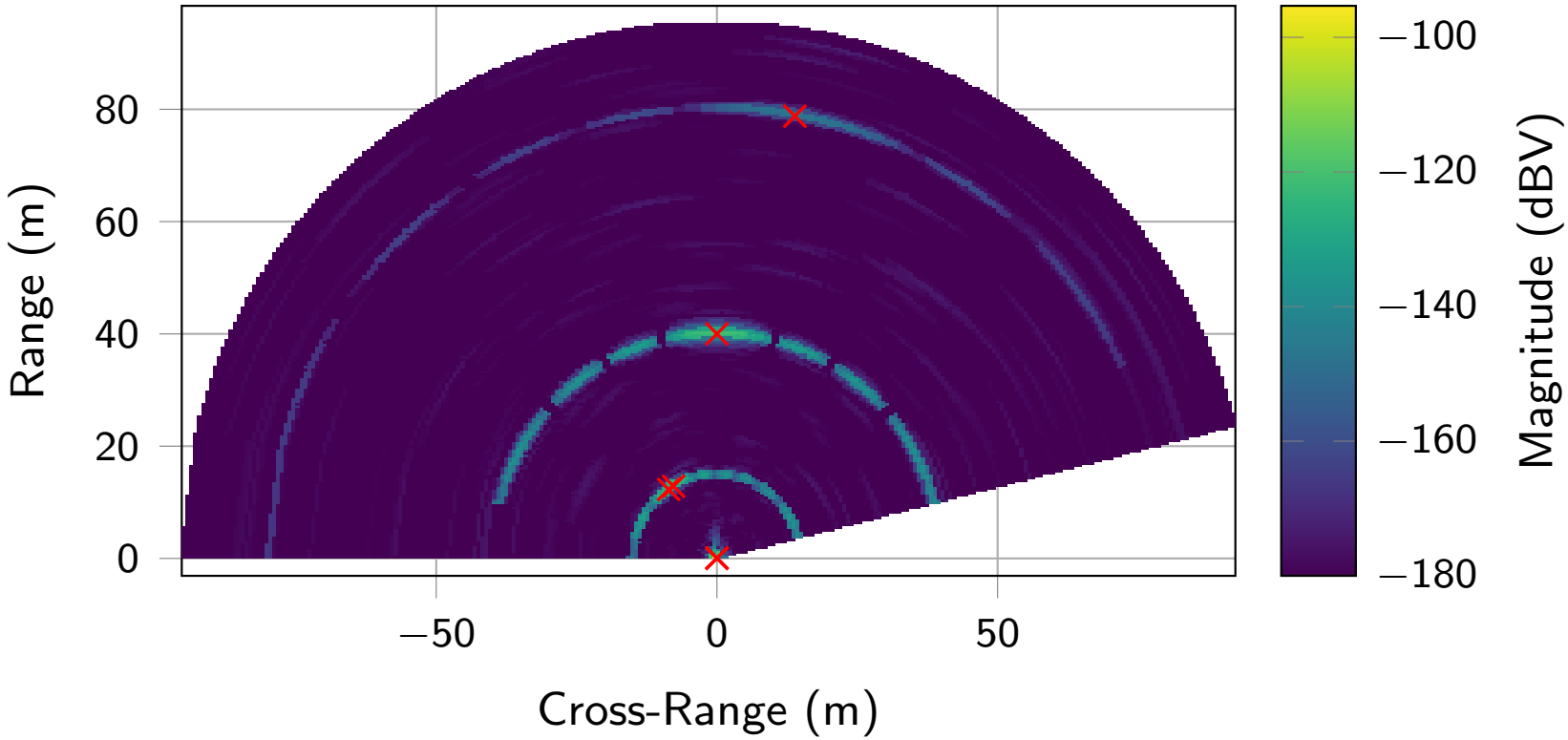


- Data was 6x5, but plot segments are 5x4!
- X and Y must be one element larger in both dimensions.

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.pcolormesh.html

Homework: FMCW: Single Chirp with ULA - DBF

- Use `pcolormesh` to map the spectrum to range/cross-range coordinates, i.e. Cartesian coordinates.
- Mark the true locations.



Pseudo Spectrum Estimation Methods

- Note the high sidelobes in DoA dimension, compared to range dimension. Windows with high sidelobe suppression have broad main lobes. A nuttall mainlobe of ± 4 bins would cover all eight channel.
- Zero-padding can increase accuracy but not resolution. Zero-padding in 2D (3D for many ramps) needs lots of memory.
- Use different spectrum estimation methods.
- Spectrum module: `https://github.com/cokelaer/spectrum`
  - Spectrum is in channel conda-forge! Don't use `--add`, but `--append` to get channels right.
- Other methods gain advantage by considering model parameters, but are not that robust and fast as the FFT.
- Use FFT to locate ranges (or range/Doppler) bins above threshold and do different spectrum estimation methods only for those bin.

Super-Resolution and the Music Algorithm

- Super-resolution algorithms: Better resolution than DFT by utilizing more signal parameters/assumptions, like number of sinusoids, or sparsity.
  - resolution gets better
  - sensitivity towards model assumption (noise color, number of sinusoids) gets worse
  - computational demand gets worse
- Music:
  - Assumption: P complex exponentials in AWGN.
  - Uses, eigenvector decomposition of the covariance matrix and separates noise based and signal based eigenvectors.
  - P-strongest one can be bogus if only less than P complex exponentials are present.
  - Music does not preserve power!

Conda Channels I

- `conda` can use packets from different sources for installing and upgrading modules.
- If no dedicated source is given upon installation, packets are located in the channels, i.e. packet repositories.
- Documentation

  https://conda.io/docs/user-guide/tasks/manage-channels.html
- Only channel `defaults` is tested by Anaconda from compatibility among all packets.
- Most prominent channel: `conda-forge`, which contains far more packets than `defaults`, but might provide broken modules.
- Show list of active channels (from an Anacond Prompt):

  ```
  conda config --show channels
  ```
- Highest priority channel is on top of list.

Conda Channels II

- If `defaults` channel was not the first entry in the channel list, auto-selected packets (during upgrade or from dependencies) might be pulled from undesired channels.
- To manipulate the channel list, to following commands are provided:

```
conda config --add channels channel_name
conda config --append channels channel_name
conda config --prepend channels channel_name
conda config --remove channels channel_name
```

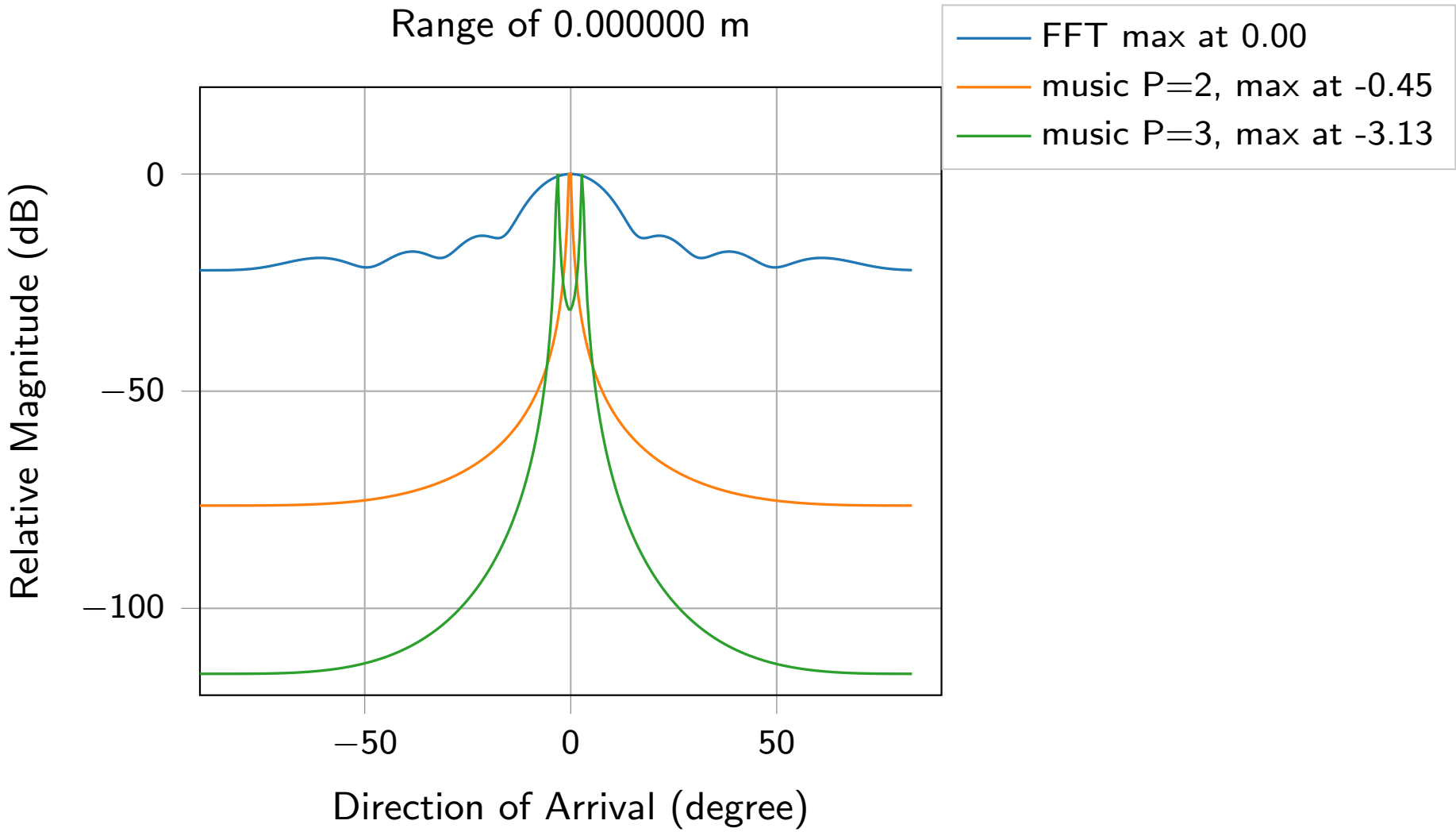
Homework: Pseudo Spectrum

- Take the IF signal from exercise "FMCW: Single Chirp with ULA"
- Use a threshold of $T_{\mathrm{dB}} = -160\,\mathrm{dBV}$ to find those range indices (in the 2D-spectrum), where at least one DoA bin exceeds the threshold.
  - Take a look at `numpy`'s functions `any`, `where`, and `flatnonzero`, which can help to avoid (slow) for-loops.
- For found ranges, plot the angular FFT spectrum and the music spectrum for `P=2` and `P=3` and `ZA_fine=256`

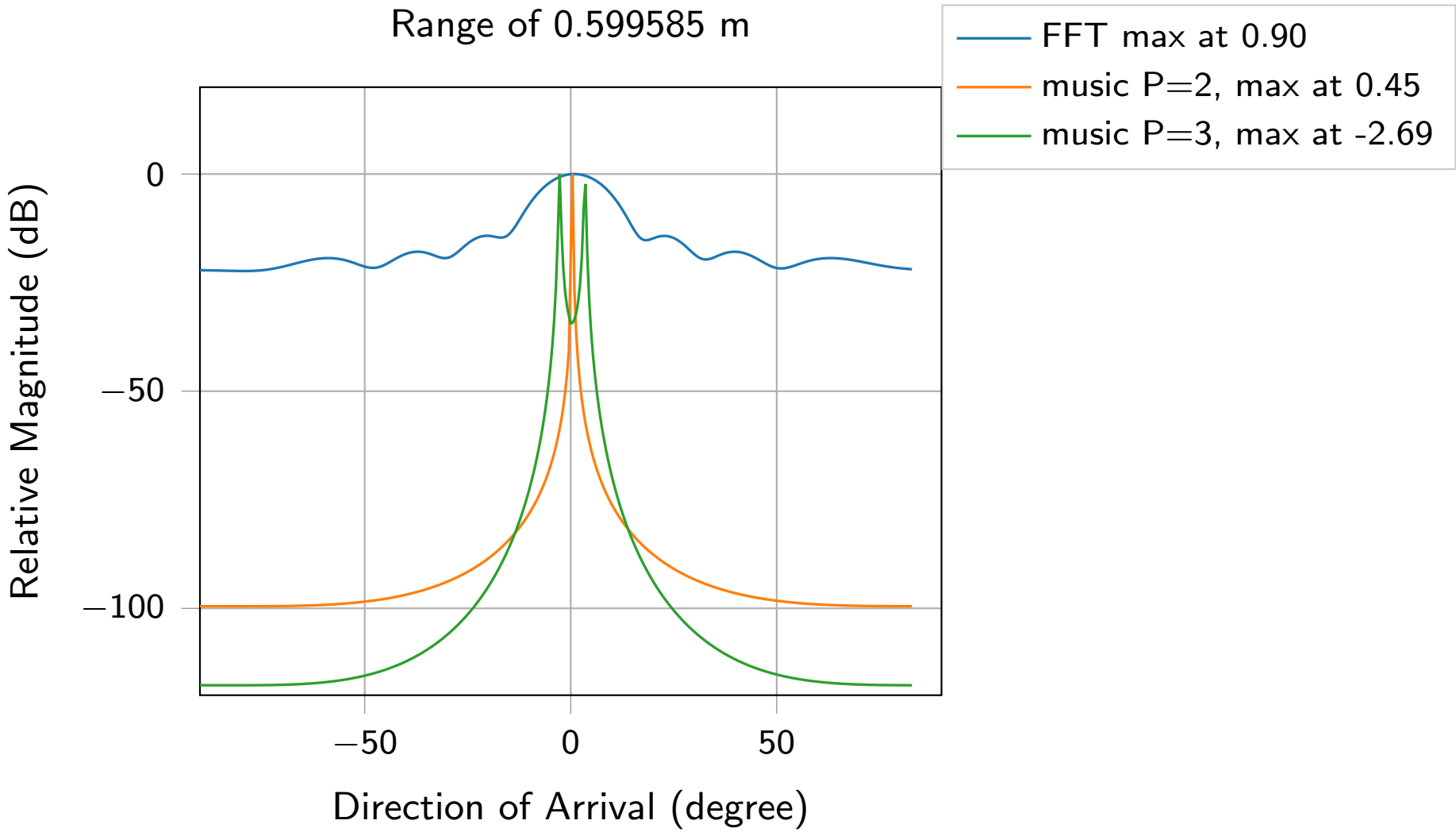
```
import spectrum as sp
res=sp.music(data, P, NFFT=ZA_fine) # data is in time domain!
res[0]                              # contains spectral data
```

- Note: as the mainlobe in range has a width of many FFT bins, multiple detection per targets are made.

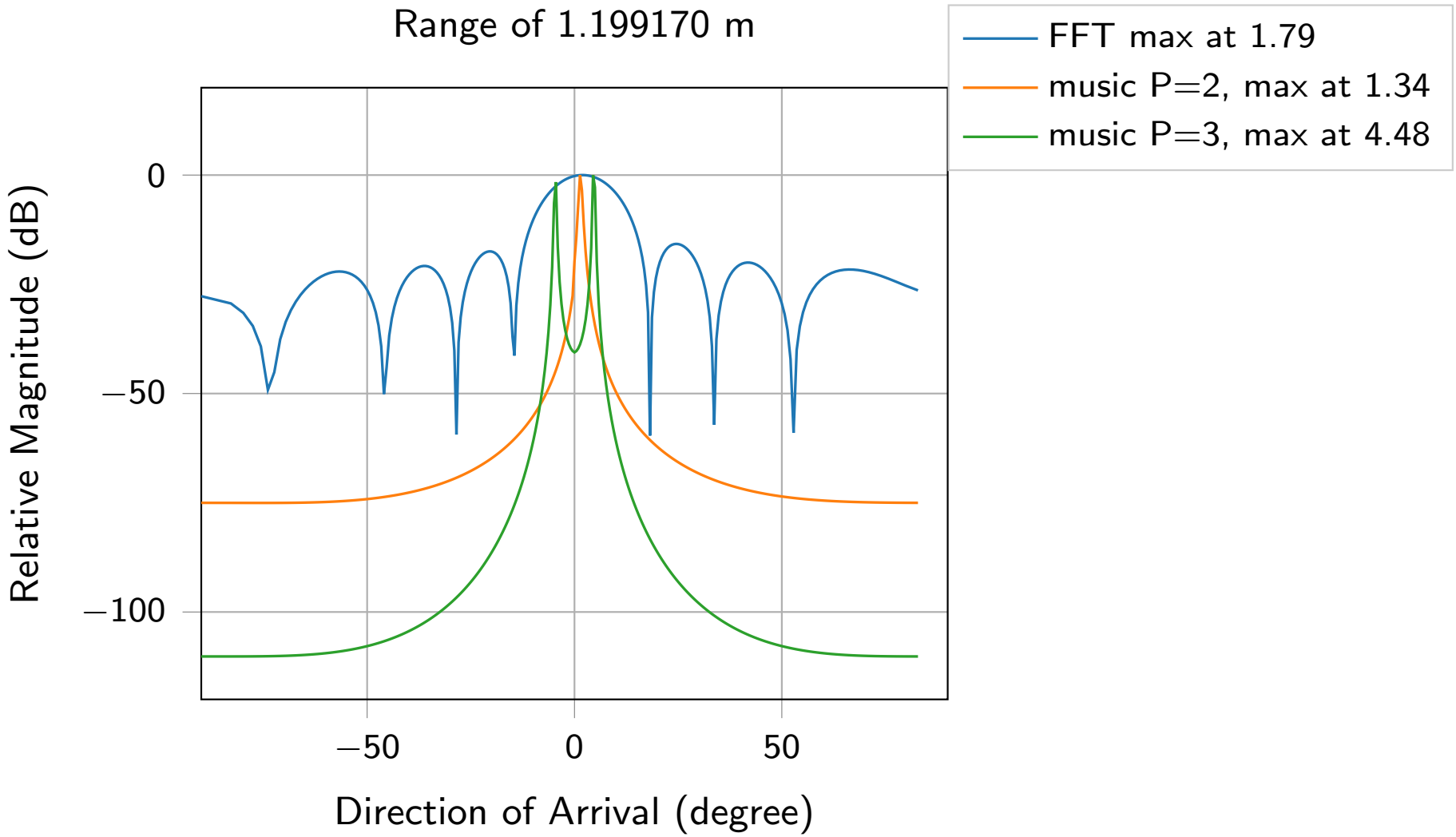
Music Pseudo Spectra vs. FFT Spectrum



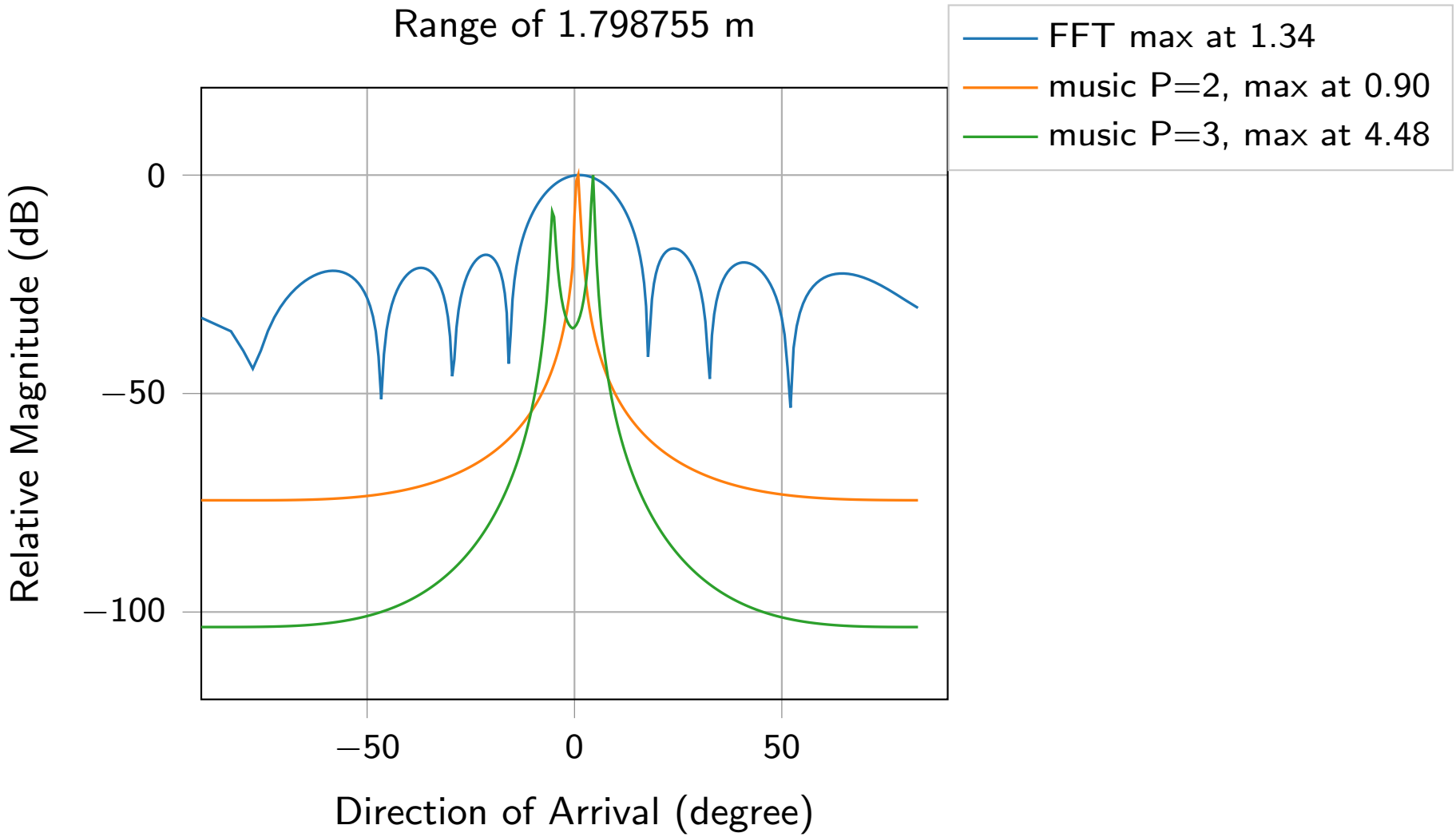
Music Pseudo Spectra vs. FFT Spectrum



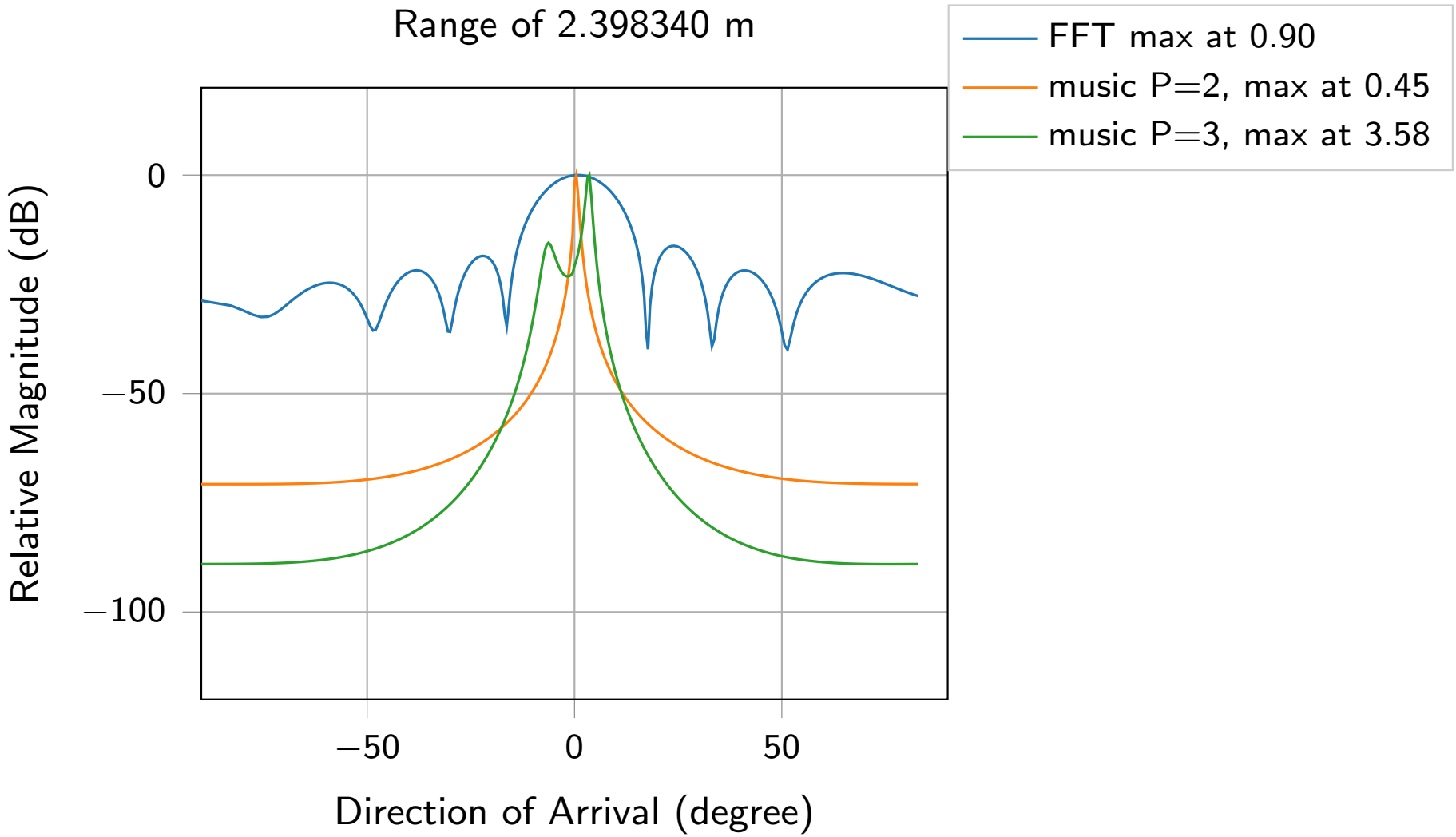
Music Pseudo Spectra vs. FFT Spectrum



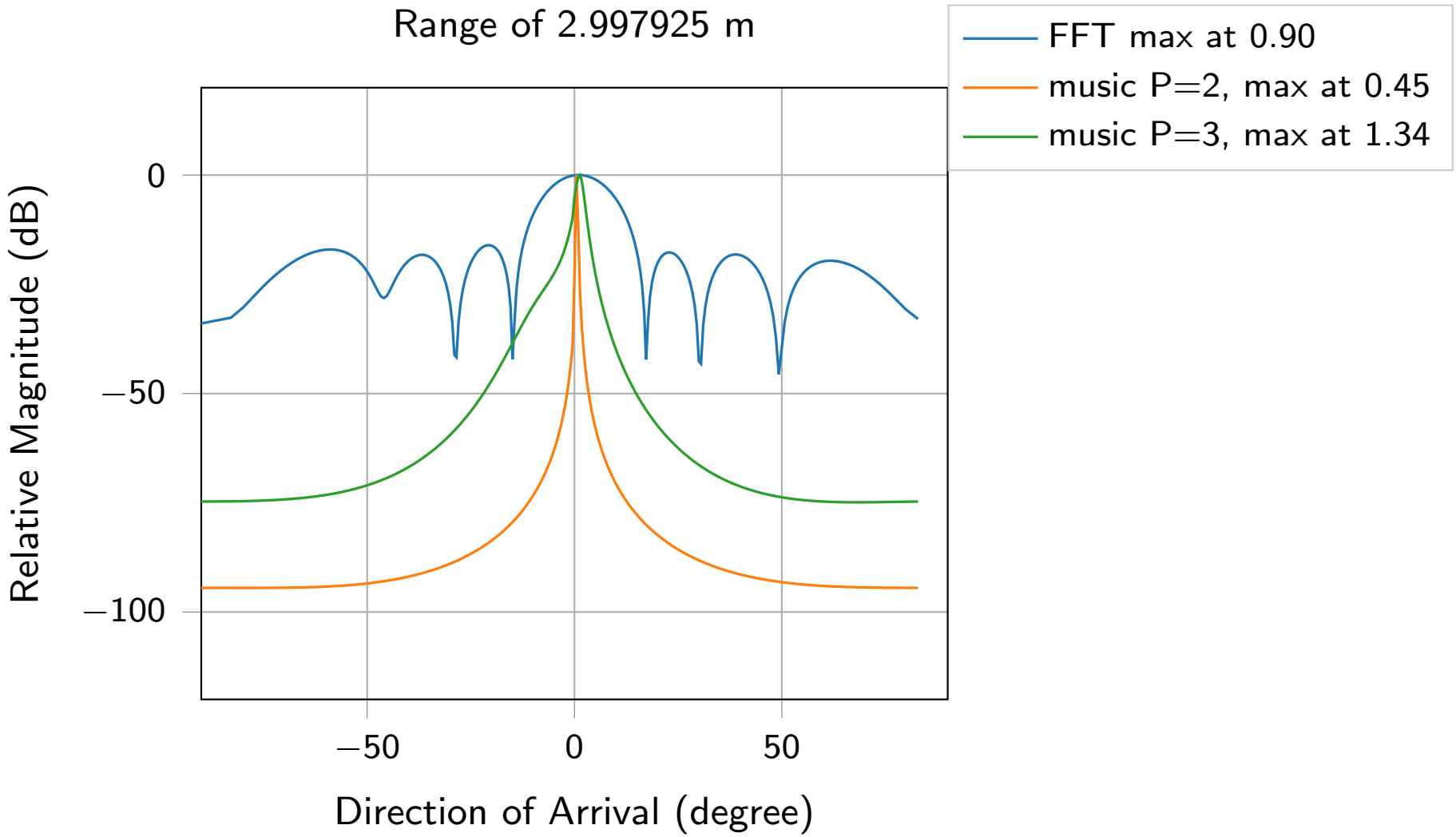
Music Pseudo Spectra vs. FFT Spectrum



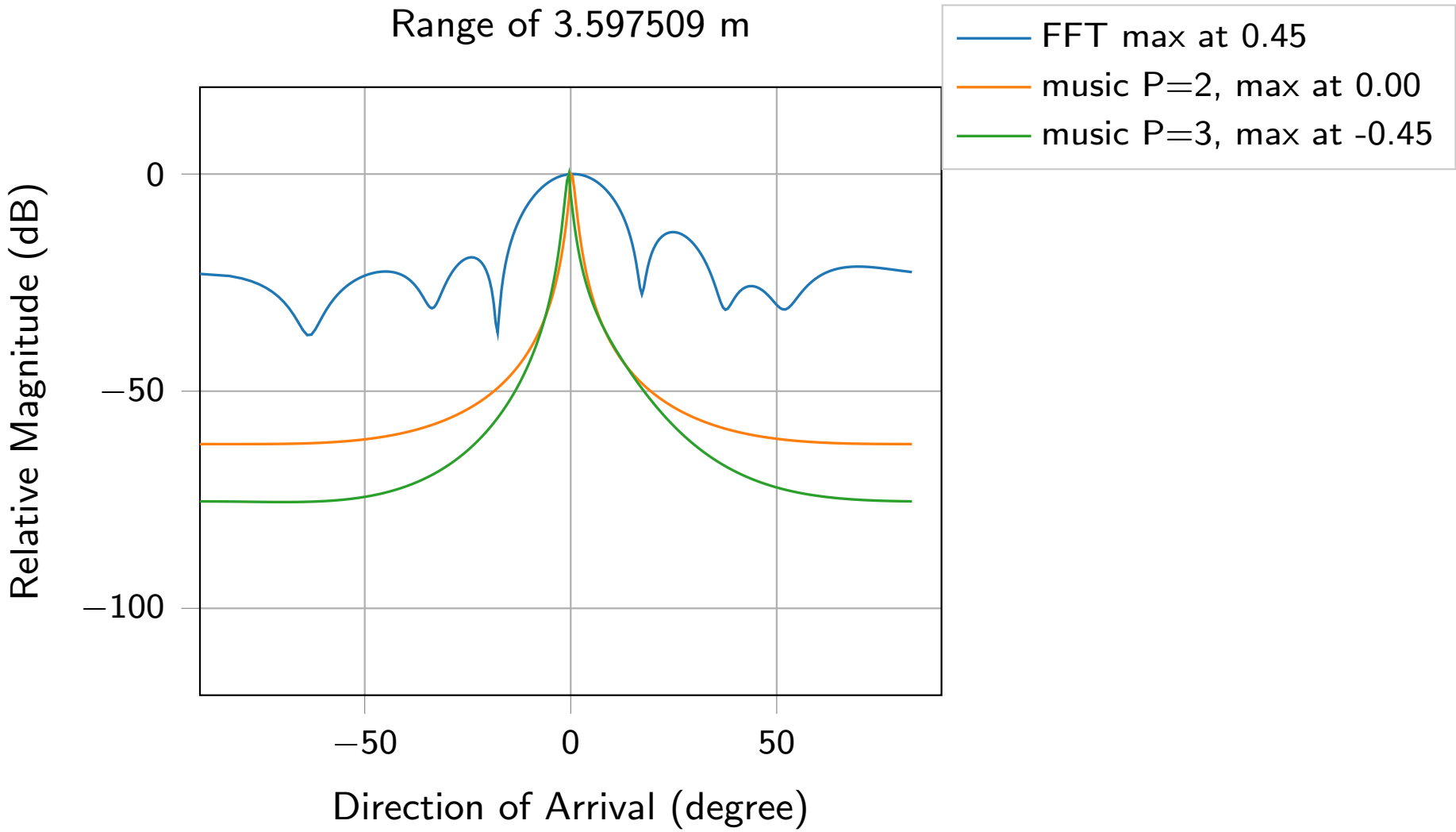
Music Pseudo Spectra vs. FFT Spectrum



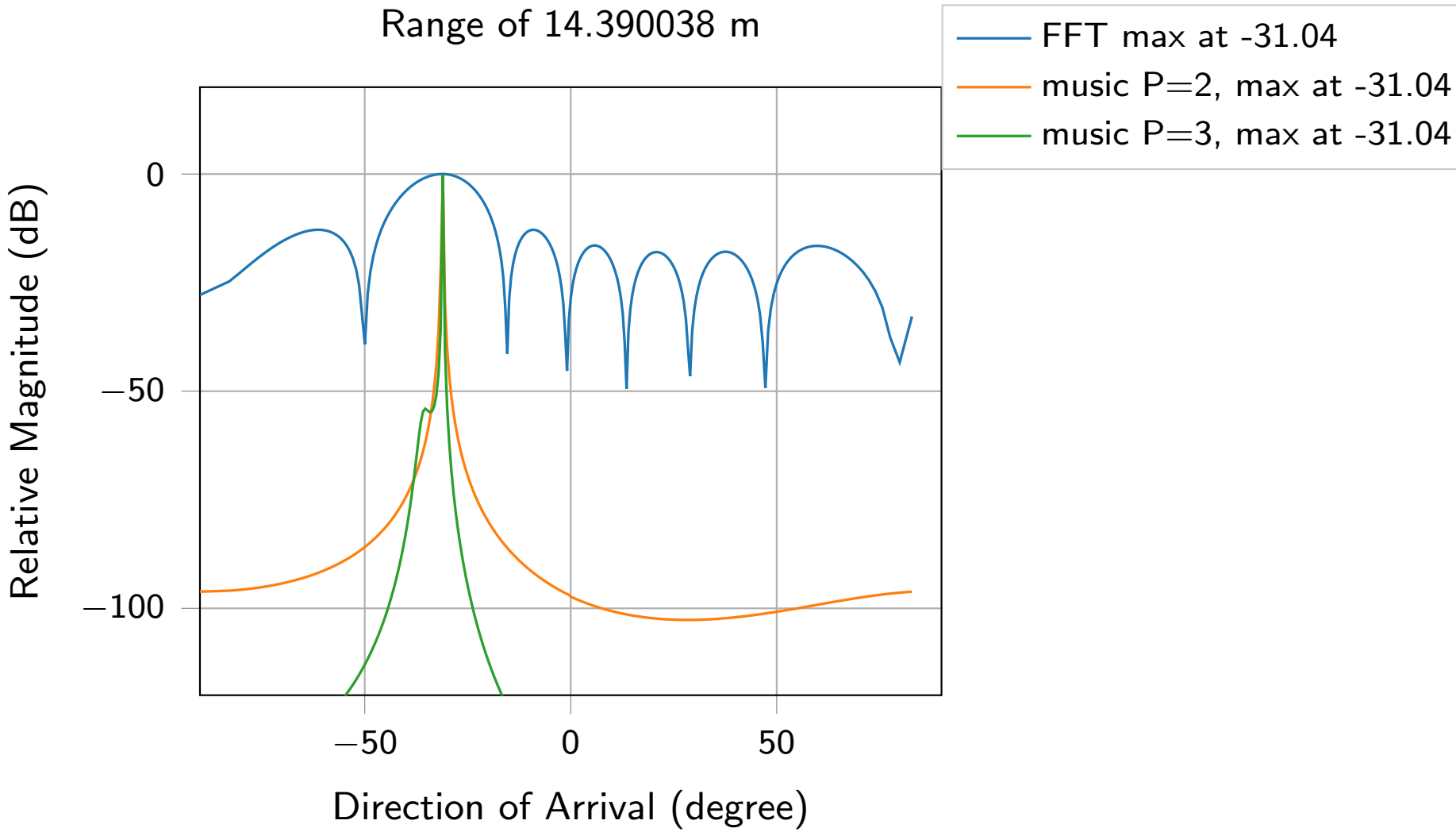
Music Pseudo Spectra vs. FFT Spectrum



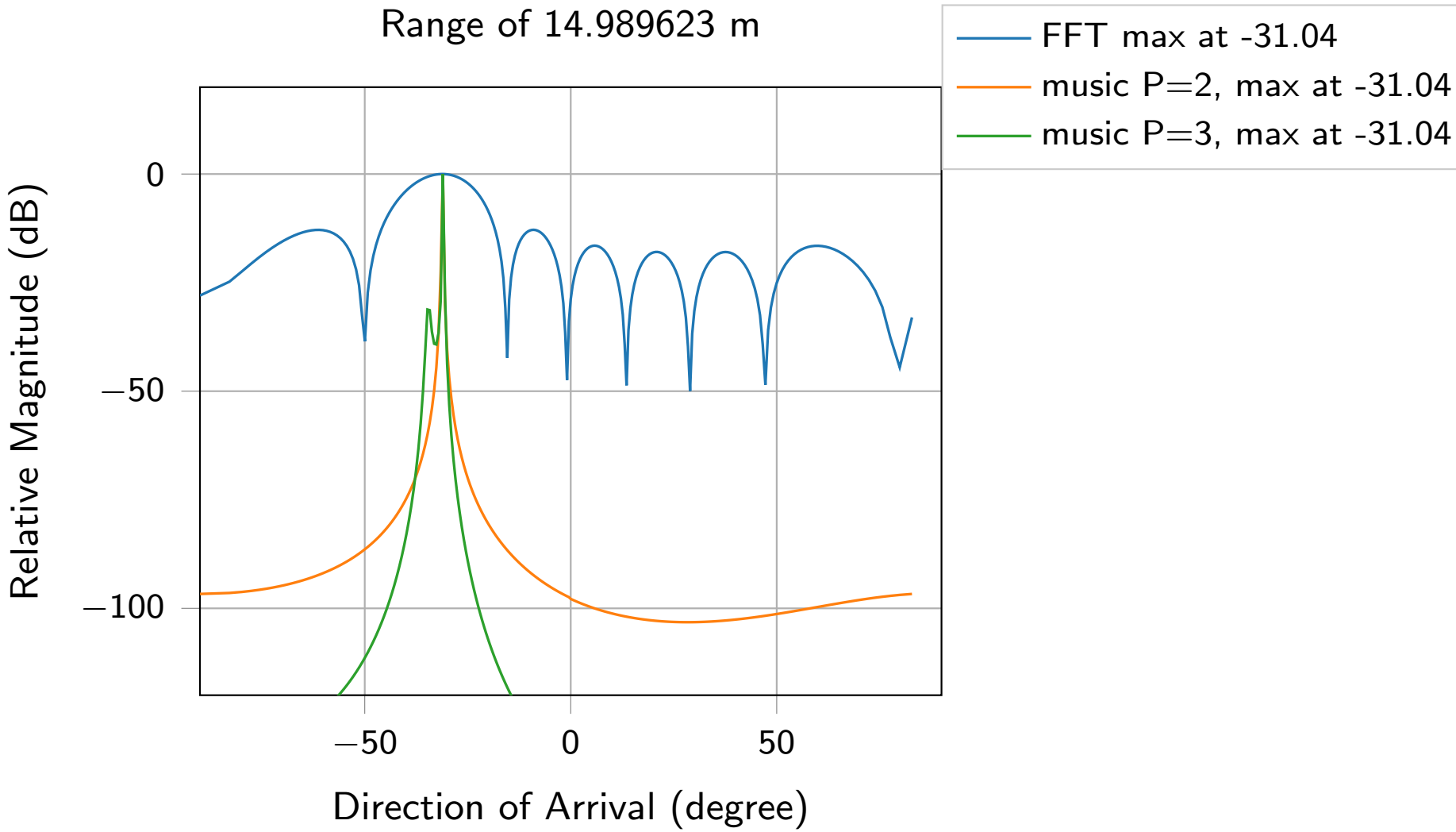
Music Pseudo Spectra vs. FFT Spectrum



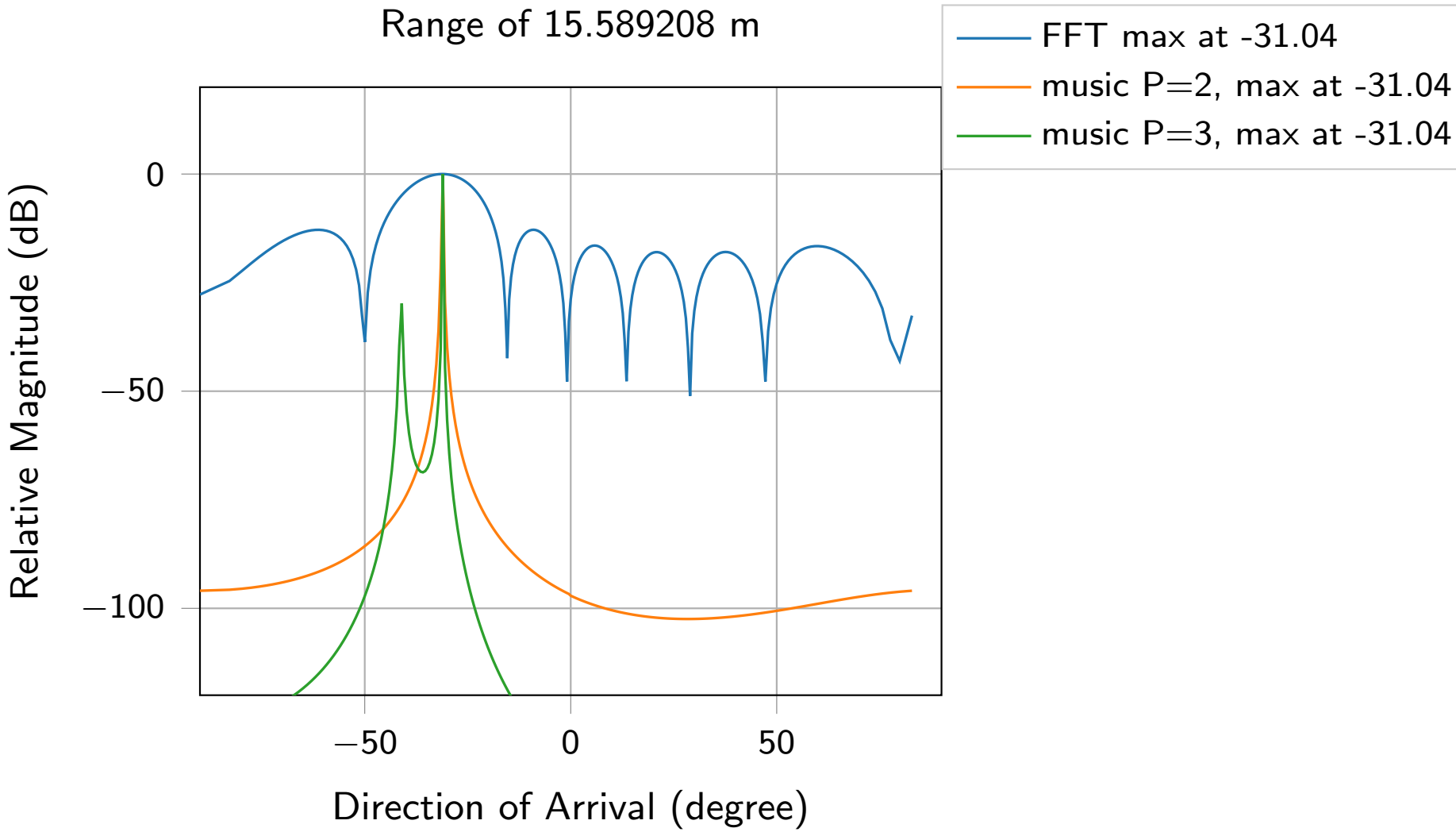
Music Pseudo Spectra vs. FFT Spectrum



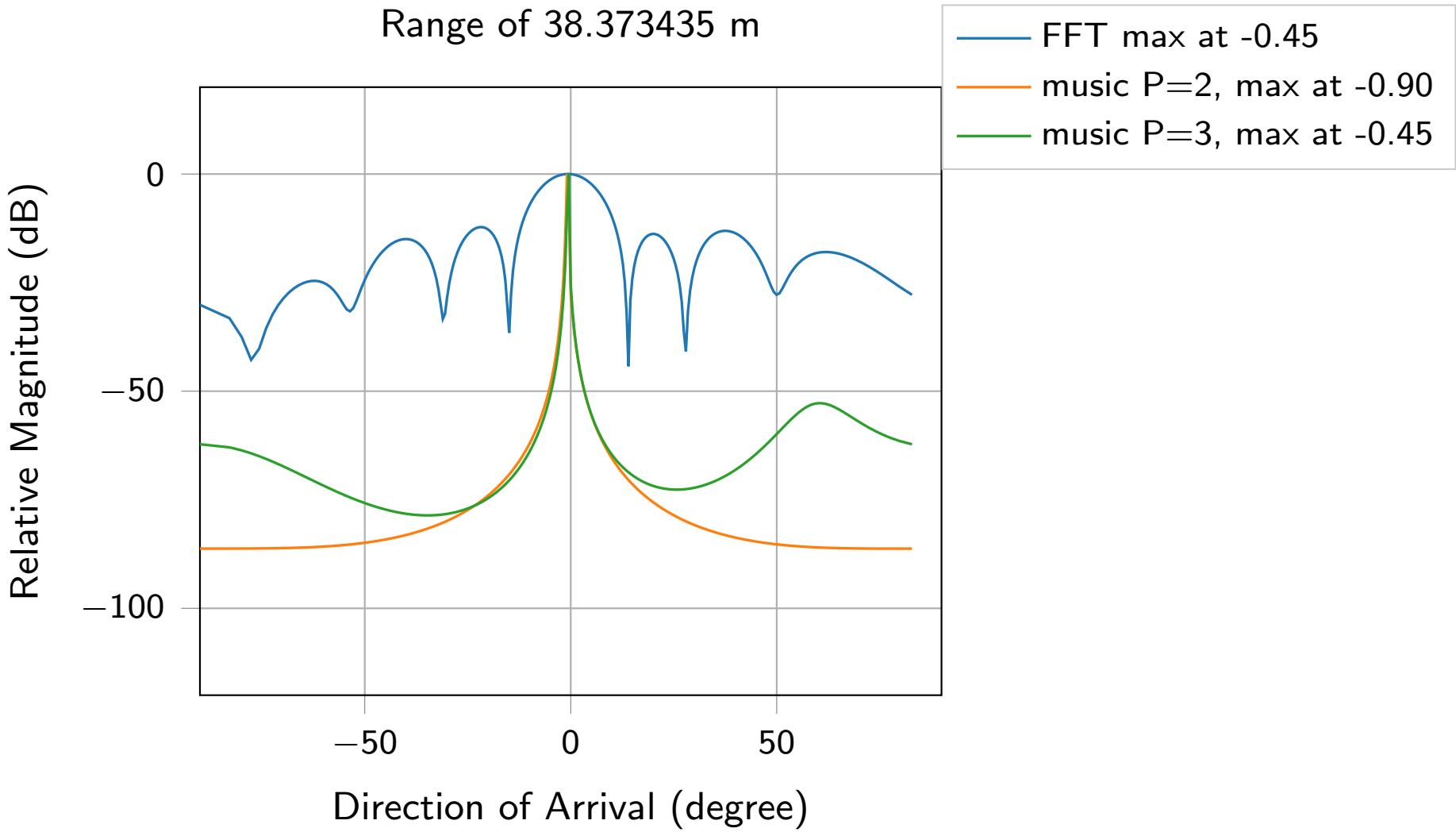
Music Pseudo Spectra vs. FFT Spectrum



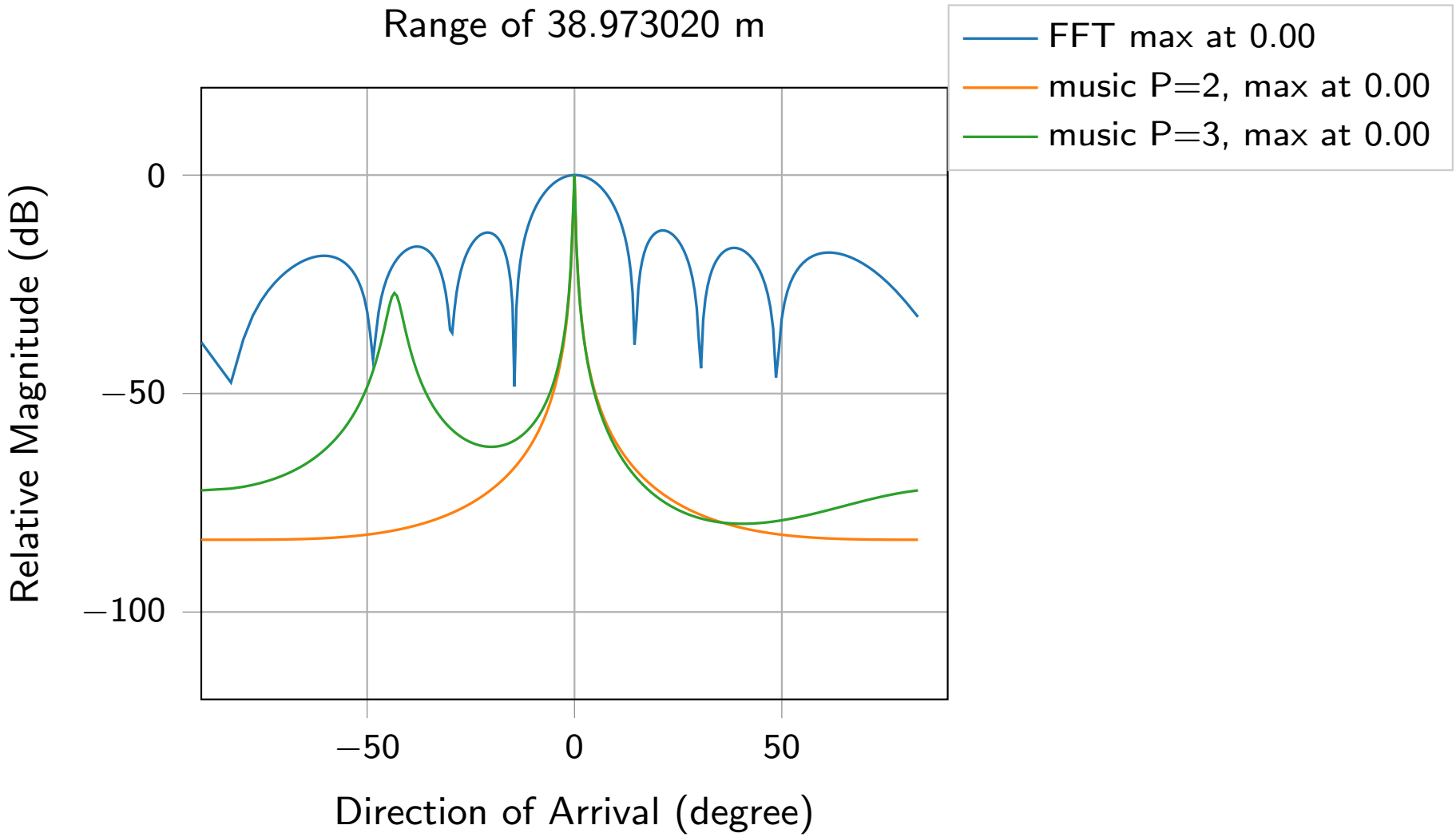
Music Pseudo Spectra vs. FFT Spectrum



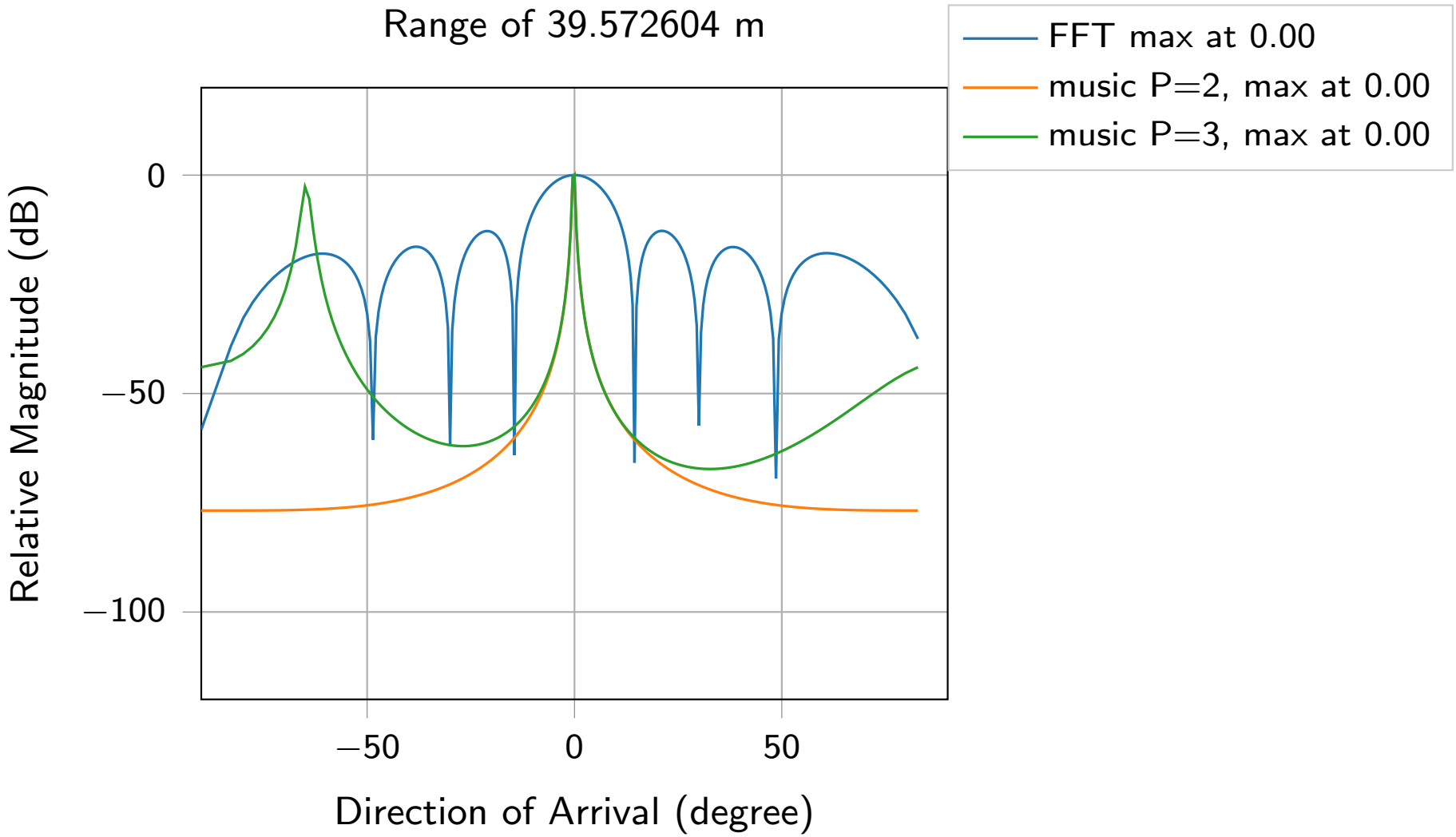
Music Pseudo Spectra vs. FFT Spectrum



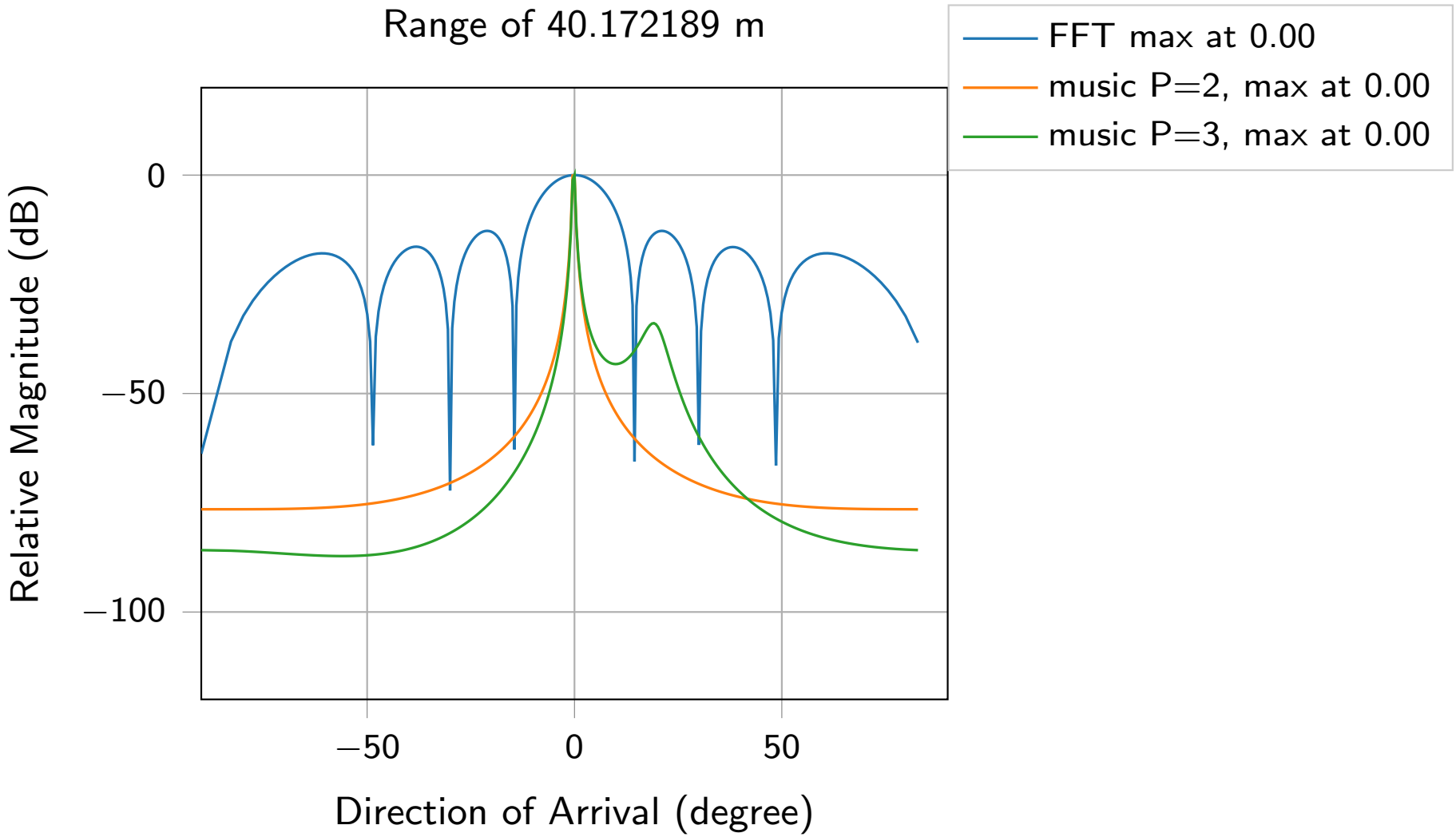
Music Pseudo Spectra vs. FFT Spectrum



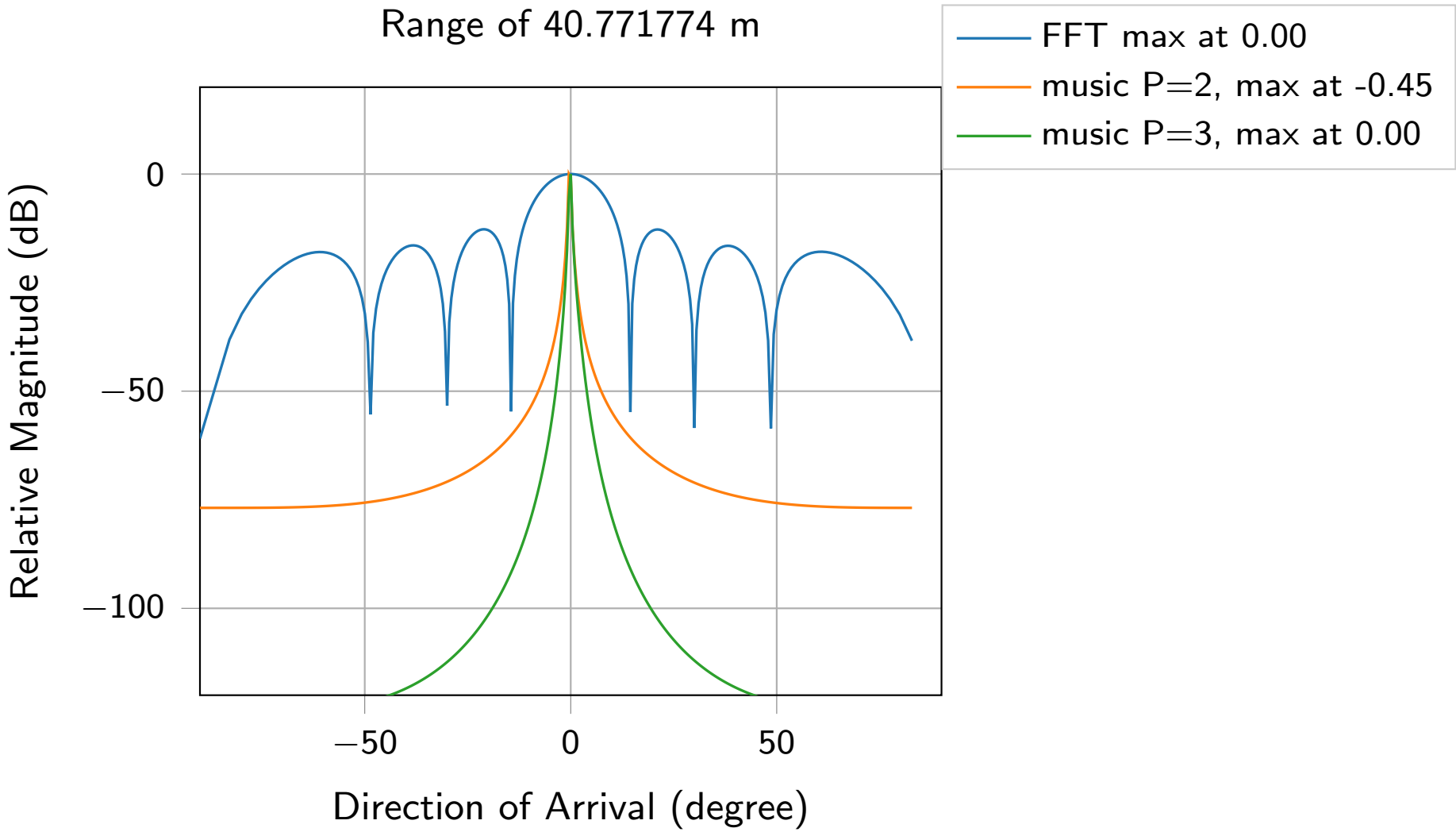
Music Pseudo Spectra vs. FFT Spectrum



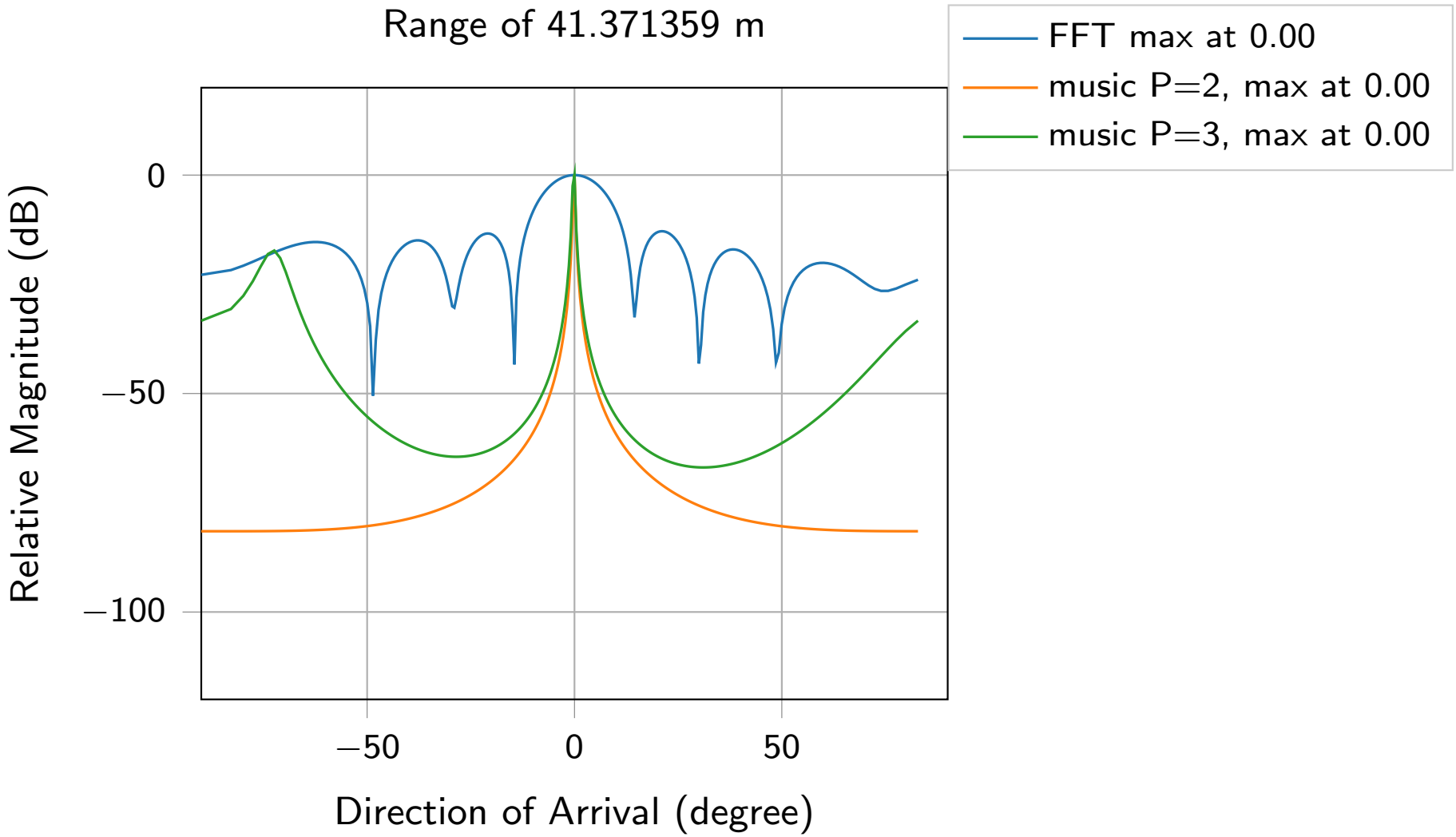
Music Pseudo Spectra vs. FFT Spectrum



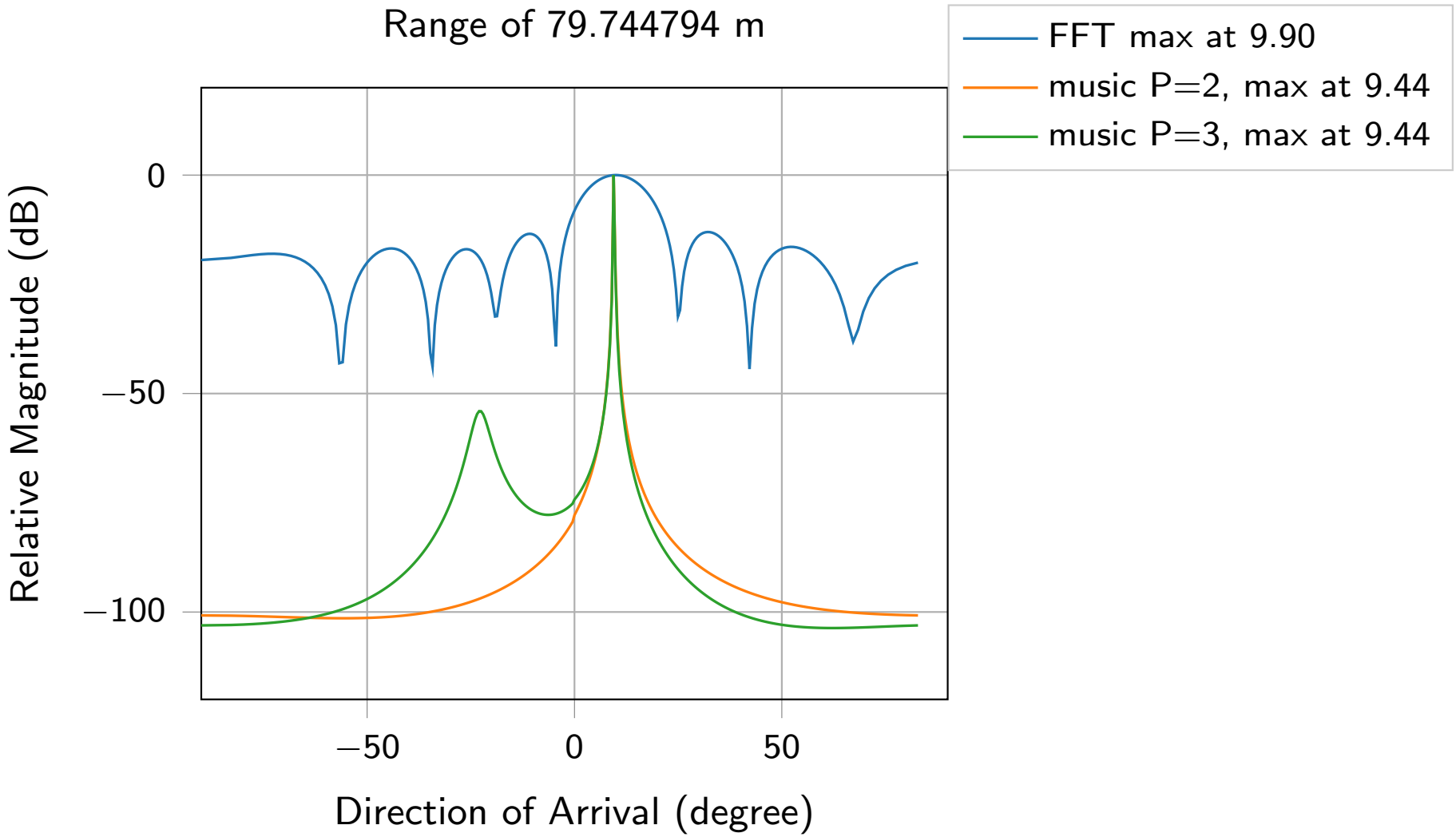
Music Pseudo Spectra vs. FFT Spectrum



Music Pseudo Spectra vs. FFT Spectrum



Music Pseudo Spectra vs. FFT Spectrum



Music Pseudo Spectra vs. FFT Spectrum