Questions

Any questions to exercises and homeworks from last time?

Table of Content

The Fast Fourier Transform

- The FFT is an algorithm to compute the discrete time Fourier transform (DTFT) efficiently.
- The DTFT is the Fourier transform for equally spaced sampled complex data (in time and frequency domain)
- The DTFT is often used to find an approximation to the power spectral density

$$P(\psi) = \left| \frac{1}{N} \sum_{n=0}^{N-1} x[n] \exp(-\mathrm{j}2\pi\psi n) \right|^2$$

at discrete sampling points k in the frequecy domain

$$P[k] = \left| \frac{1}{N} \sum_{n=0}^{N-1} x[n] \exp\left(-\mathrm{j}2\pi \frac{k\, n}{N}\right) \right|^2 \quad \text{for} \quad k = 0, \ldots, N-1$$

The Fast Fourier Transform in Python

- Implementation of the FFT in libraries may differ in pre-factor (1, $\frac{1}{N}$, $\frac{1}{2\pi}$, or square-roots of them), exponent ($\pm 2\pi$ or ω), and start index (0 or $-\frac{N}{2}$), and the lacking of the absolute value.
- In Python, the FFT is usually calculated as

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\left(-2\pi \mathrm{j} \frac{n\,k}{N}\right) \qquad k = 0, \ldots, N-1$$

  which is just the sum-term of the power spectrum $P[k]$.
- Symmetric Fourier transform (start index at $-N/2$):

$$X_{\text{sym}} = X[k] \exp\left(-2\pi \mathrm{j} \frac{k}{N} \frac{N-1}{2}\right)$$

- The FFT is not actually implemented in Python due to speed.
- FFT modules in Python just link to a compiled library.

FFT Libraries in Python

- Most common wrapper modules for the FFT are:
  - `numpy.fft` https://docs.scipy.org/doc/numpy-1.15.0/reference/routines.fft.html
  - `scipy.fftpack` https://docs.scipy.org/doc/scipy/reference/fftpack.html
  - `pyfftw` https://github.com/pyFFTW/pyFFTW
- Functions provided by those modules have nearly the same arguments, thus modules are interchangeable.
- Speed of the FFT can vary heavily with
  - utilized library (optimizations and multithreading),
  - data type (single, or double precision), and
  - number of samples (different algorithms perform differently)
- `scipy` and `numpy` work out of the box and try to use Intel's Math Kernel Library (MKL).
- FFTW is usually the fastest but needs extra steps on Windows, as Windows does not ship with a C compiler.

Partial Spectrum DFT Algorithms in Python

Goertzel-Algorithm

- To calculate the DFT for an arbitrary list of frequencies.
- Faster than FFT for low number of desired frequencies.
- No module in python but implementation is trivial.

Chirp-z Transform

- Also called chirp-transform algorithm (CTA), or zoom-FFT
- Based on Bluestein FFT (for arbitrary data length)
- CZT didn't made it into scipy/numpy due to numerical problems with corner cases.

  `https://github.com/scipy/scipy/issues/4288`
- Available at `https://gist.github.com/ewmoore/6d321c1dbb5cea9bfaeb`

Windowing

- There is no unwindowed FFT!

$$P_{\mathrm{rect}}[k] = \left| \frac{1}{N} \sum_{n=0}^{N-1} x[n] \exp\left(-\mathrm{j}2\pi \frac{n\,k}{N} \right) \right|^2$$

$$P_{\mathrm{wind}}[k] = \left| \frac{1}{\sum_{n=0}^{N-1} a[n]} \sum_{n=0}^{N-1} a[n] x[n] \exp\left(-\mathrm{j}2\pi \frac{n\,k}{N} \right) \right|^2$$

- Different windows produce different the PSF.
- Allows to trade beam-width for side-lobe level, e.g., increases dynamic range but reduces resolution.
- Note that the sum can still be evaluated using an FFT library!

Windowing using Python

- Three modules provide window functions:
  - `numpy`
  - `scipy.signal.windows`
  - `spectrum.window`

- `numpy` provides some basic window functions with

  `np.bartlett`, `np.blackman`, `np.hamming`, `np.hanning`, `np.kaiser`

```
import numpy as np
wind=np.hanning(128)
```

- `scipy.signal.windows` provide most commonly used window functions with the option for symmetric windows.

  https://docs.scipy.org/doc/scipy/reference/signal.windows.html

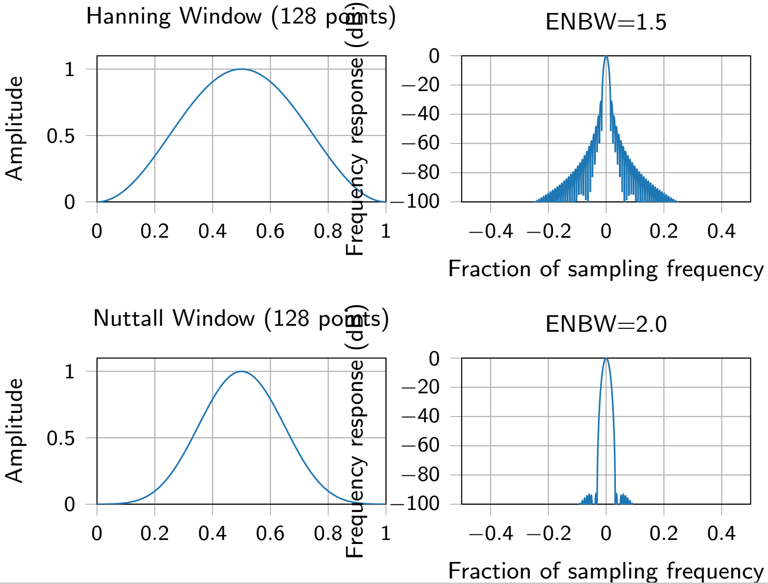
```
import scipy.signal.windows as windows
wind=windows.nuttall(N, sym=True)
```

Windowing using Python

- `spectrum.window` provides many tools for spectrum estimation, like functions for generating and analysis of a wider range of window functions. https://github.com/cokelaer/spectrum

```python
import spectrum
import matplotlib.pyplot as plt
from matplotlib2tikz import save as tikz_save

# generate the samples
wind_samples=spectrum.window.window_nuttall(128)

# generate window object
for name in ('hanning', 'nuttall'):
    wind=spectrum.Window(128,name)      # generate a window object
    wind_samples=wind.data              # get the data

    plt.figure()
    wind.plot_time_freq()               # plot the window function

    tikz_save('Window_'+name+'.tikz');
    plt.savefig('Window_'+name+'.png', dpi=150)
```

Windowing using Python



Sums of Sinusoidal Signals With Normalized Frequency

- Sum of M sinusoidal signal at sampling indicess n:

$$s_{\text{real}}[n] = \sum_{m=0}^{M} A_m \sin\left(2\pi\psi_m n + \phi_m\right)$$

$$s_{\text{analytic}}[n] = \sum_{m=0}^{M} A_m \exp(\mathrm{j}2\pi\psi_m n + \mathrm{j}\phi_m)$$

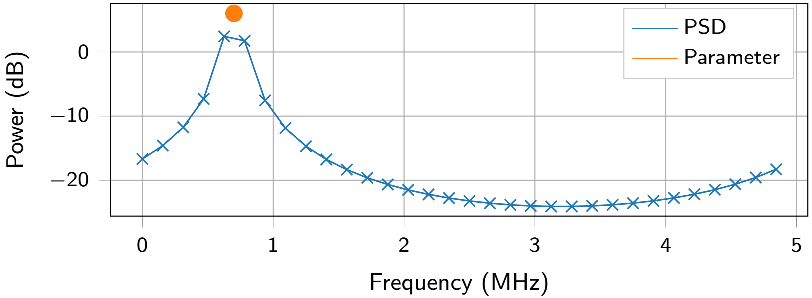
- Normalized notations to become independent of sample rate.
- Here: $\psi_m = f_m / T_\mathrm{s}$ with

ψ_m	normalized frequency
f_m	frequency in Hz
T_s	sampling frequency in Hz

- $\psi_m = 0$ is DC, $\psi_m = 0.5$ is half the sampling frequency (cf. Nyquist–Shannon sampling theorem)
- Note: $\sin x = \frac{1}{2\mathrm{j}}\left(\exp(\mathrm{j}x) - \exp(-\mathrm{j}x)\right)$

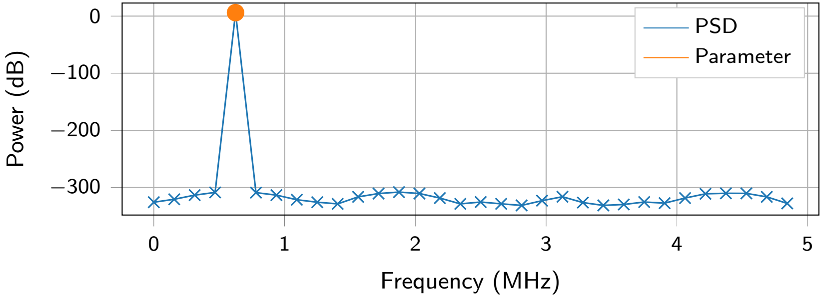
Exercise: Visualization of the FFT Spectrum

- Consider 32 samples of a complex signal taken at a sampling rate of 5 MHz with $A = 2$, $\psi = 0.14$, and $\phi = 0$.
- Estimate and plot the power spectrum using the FFT.
  - Plot in dB over MHz
  - What happens if ψ is a multiple of $1/N$, e.g, $\psi = 0.125$?
  - Hint: take a look at the keyword parameter `endpoint` if you use linspace to construct the frequency axis.



Exercise: Visualization of the FFT Spectrum

- Consider 32 samples of a complex signal taken at a sampling rate of 5 MHz with $A = 2$, $\psi = 0.14$, and $\phi = 0$.
- Estimate and plot the power spectrum using the FFT.
  - Plot in dB over MHz
  - What happens if ψ is a multiple of $1/N$, e.g, $\psi = 0.125$?
  - Hint: take a look at the keyword parameter `endpoint` if you use linspace to construct the frequency axis.



Homework: Visualization of the FFT Spectrum

- Consider 128 samples of a real signal taken at a sampling rate of 2 MHz, containing three sinusoidals with

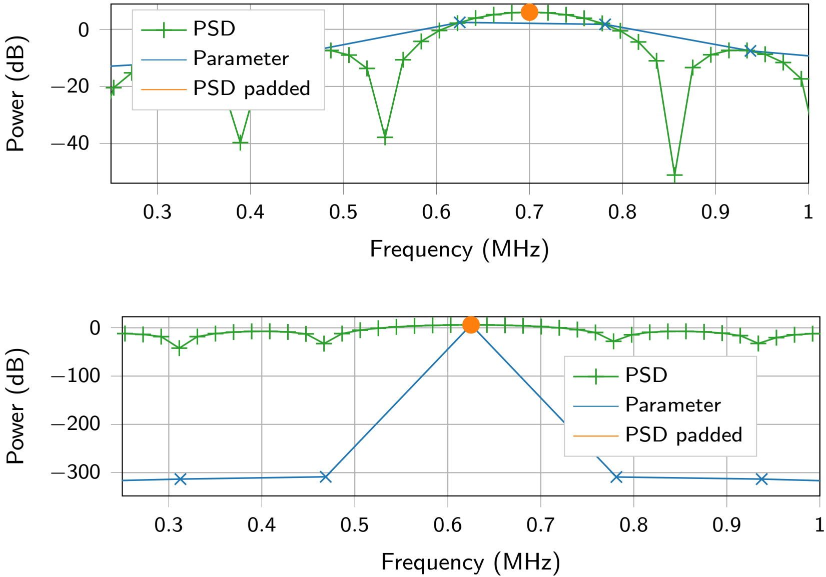
```
Am   = [1.0, 0.5, 1e-4, 0.001];
psim = [0.1, 0.1+2.5/N, 0.17, 0.21];
phim = [0.0, 0.0, 2.0, 0.0];
```

- Estimate the power spectrum by using the FFT.
  - Plot it in dB (normalized to `Am[0]`) over frequency in MHz.
  - Use `xlim` and `ylim` to zoom the plot to the interesting region 0.1 to 0.5 MHz and -100 to 10 dB.
  - Mark the magnitudes and frequencies in the spectrum plot.
- Plot the power spectrum for three different window functions: rect, hanning and nuttall.
- The ideal Fourier transform would produce 4 peaks. Explain:
  - Why (in terms of shape of the main- and sidelobes) peaks are not visible/distinct for certain window functions?
  - Why do the tips of the DFT/FFT peaks not meet the sinusoidal parameters?

Sinusoidal Parameter Estimation Using the FFT

- Idea: Location of the maximum in the power spectrum corresponds to the parameters of the strongest sinusoid.
- Zero padding
  - Padding the signal with zeros prior the FFT produces a finer spectral grid.
  - The function call to `FFT` has a parameter for zero padding. NO manual concatenate needed.
  - The pulse-spread-function does not change!
  - Thus, discretization errors are reduced, resolution is not increased, however memory consumption is increased.
- AWGN introduces additional errors. Similar to the AWGN peak detection from the correlation exercise.

Zero-Padding Example



Parabola Fit

- The frequency discretization of the DTFT is—one of many—sources for an inaccurate frequency estimation using the argmax method.
- The main lobe of the PSD (squared magnitude spectrum) is well approximated by a 2nd-order polynomial for most window functions.

$$P(\psi) = a_0 + a_1\psi + a_2\psi^2$$

- The location of the maximum of the 2nd polynomial can be found analytical without discretization.

$$0 = \left.\frac{\mathrm{d}P(\psi)}{\mathrm{d}\psi}\right|_{\psi=\psi_{\mathrm{max}}} \quad \Rightarrow \quad \psi_{\mathrm{max}} = -\frac{a_1}{2a_2} \quad F_{\mathrm{max}} = a_0 - \frac{a_1^2}{4a_2}$$

- No parabola/linear fit for phase due to wrapping on 2π. Use DFT/goertzel ones ψ_{max} found.

Calculate Parameters of the Polynomial

- Tricks for derivation:
  - center the system of coordinates around 0 frequency
  - FFT bins are all equally spaced by d_f, i.e $\psi \in \{0, -d_\mathrm{f}, d_\mathrm{f}\}$

$$\begin{pmatrix} P_\mathrm{L} \\ P_\mathrm{C} \\ P_\mathrm{R} \end{pmatrix} = \begin{pmatrix} 1 & -d_\mathrm{f} & d_\mathrm{f}^2 \\ 1 & 0 & 0 \\ 1 & d_\mathrm{f} & d_\mathrm{f}^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}$$

Calculate Parameters of the Polynomial

- Tricks for derivation:
  - center the system of coordinates around 0 frequency
  - FFT bins are all equally spaced by d_f, i.e $\psi \in \{0, -d_\mathrm{f}, d_\mathrm{f}\}$

$$\begin{pmatrix} P_\mathrm{L} \\ P_\mathrm{C} \\ P_\mathrm{R} \end{pmatrix} = \begin{pmatrix} 1 & -d_\mathrm{f} & d_\mathrm{f}^2 \\ 1 & 0 & 0 \\ 1 & d_\mathrm{f} & d_\mathrm{f}^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}$$

- Polynomial Parameters:

$$a_0 = P_\mathrm{C} \qquad a_1 = \frac{P_\mathrm{R} - P_\mathrm{L}}{2d_\mathrm{f}} \qquad a_2 = \frac{P_\mathrm{L} - 2P_\mathrm{C} + P_\mathrm{R}}{2d_\mathrm{f}^2}$$

Calculate Parameters of the Polynomial

- Tricks for derivation:
  - center the system of coordinates around 0 frequency
  - FFT bins are all equally spaced by d_f, i.e $\psi \in \{0, -d_\mathrm{f}, d_\mathrm{f}\}$

$$\begin{pmatrix} P_\mathrm{L} \\ P_\mathrm{C} \\ P_\mathrm{R} \end{pmatrix} = \begin{pmatrix} 1 & -d_\mathrm{f} & d_\mathrm{f}^2 \\ 1 & 0 & 0 \\ 1 & d_\mathrm{f} & d_\mathrm{f}^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}$$

- Polynomial Parameters:

$$a_0 = P_\mathrm{C} \qquad a_1 = \frac{P_\mathrm{R} - P_\mathrm{L}}{2d_\mathrm{f}} \qquad a_2 = \frac{P_\mathrm{L} - 2P_\mathrm{C} + P_\mathrm{R}}{2d_\mathrm{f}^2}$$

- Result:

$$\psi_\text{offset} = -\frac{a_1}{2a_2} = \frac{d_\mathrm{f}(P_\mathrm{R} - P_\mathrm{L})}{2\,(P_\mathrm{L} - 2P_\mathrm{C} + P_\mathrm{L})}$$

$$F_\text{max} = \frac{-2P_\mathrm{R}(4P_\mathrm{C} + P_\mathrm{L}) + (P_\mathrm{L} - 4P_\mathrm{C})^2 + P_\mathrm{R}^2}{16P_\mathrm{C} - 8(P_\mathrm{L} + P_\mathrm{R})}$$

Calculate Parameters of the Polynomial

- Tricks for derivation:
  - center the system of coordinates around 0 frequency
  - FFT bins are all equally spaced by d_f, i.e $\psi \in \{0, -d_\mathrm{f}, d_\mathrm{f}\}$

$$\begin{pmatrix} P_\mathrm{L} \\ P_\mathrm{C} \\ P_\mathrm{R} \end{pmatrix} = \begin{pmatrix} 1 & -d_\mathrm{f} & d_\mathrm{f}^2 \\ 1 & 0 & 0 \\ 1 & d_\mathrm{f} & d_\mathrm{f}^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}$$

- Polynomial Parameters:

$$a_0 = P_\mathrm{C} \qquad a_1 = \frac{P_\mathrm{R} - P_\mathrm{L}}{2d_\mathrm{f}} \qquad a_2 = \frac{P_\mathrm{L} - 2P_\mathrm{C} + P_\mathrm{R}}{2d_\mathrm{f}^2}$$

- Result:

$$\psi_\mathrm{offset} = -\frac{a_1}{2a_2} = \frac{d_\mathrm{f}(P_\mathrm{R} - P_\mathrm{L})}{2\left(P_\mathrm{L} - 2P_\mathrm{C} + P_\mathrm{L}\right)}$$

$$F_\mathrm{max} = \frac{-2P_\mathrm{R}(4P_\mathrm{C} + P_\mathrm{L}) + (P_\mathrm{L} - 4P_\mathrm{C})^2 + P_\mathrm{R}^2}{16P_\mathrm{C} - 8(P_\mathrm{L} + P_\mathrm{R})}$$

- Zero-padding is still required but to a less extent.

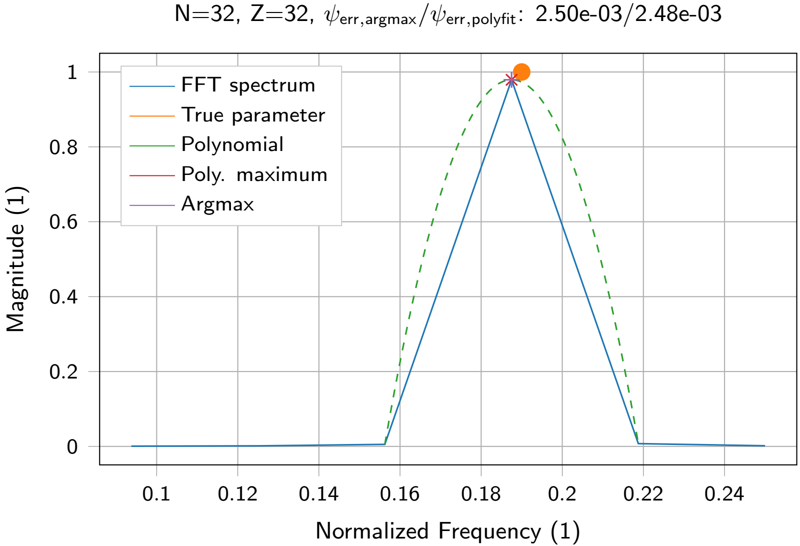
Exercise: Calculate Polynomial Fit

```python
def calc_polyfit_correction(spectrum_excert, df):
    """
    Parameters
    ----------
    spectrum_excert: list of three floats
        left, center and right magnitude value

    df: float
        frequency difference of two consecutive FFT-bins

    Returns
    -------
    offset_frequency: float
        Offset towards the location of Fc

    Fmax: magnitude at the maximum

    a: list of three floats
        Parameters of the polynomial.

    """
```

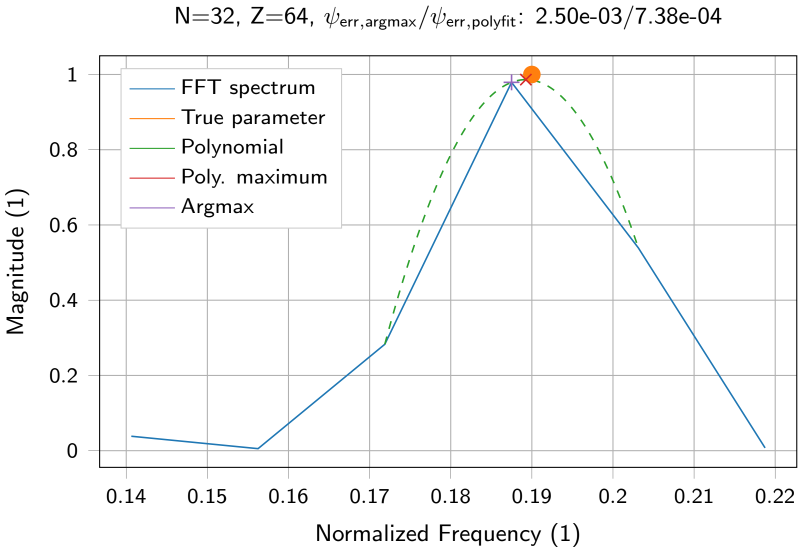
Exercise: Test Polynomial Fit

- Generate and plot the FFT based spectrum estimate for a sinusoid `A=[1]; psi=[0.19]; phi=[0.85]; N=32; Z=4*N`.
- Apply the method `calc_polynomial_fit` from the exercise before.
- Annotate the plot with
  - a marker for the sinusoidal parameters,
  - the maximum using argmax,
  - the maximum obtained by polynomial fit, and
  - the polynom in a region of ± 1 FFT bin around the maximum.
- Hints:
  - `numpy.polyval` can help drawing the polynomial curve
  - The FFT spectrum is periodic, thus getting the left and right value might need index wrapping.
    - If argmax produced index 0, negative indexing allows to get the left value with `index-1`.
    - If argmax produced index $Z-1$, obtaining the right value might produce `index is out of bounds` error. Check for this case or use the modulo operation % to avoid it.

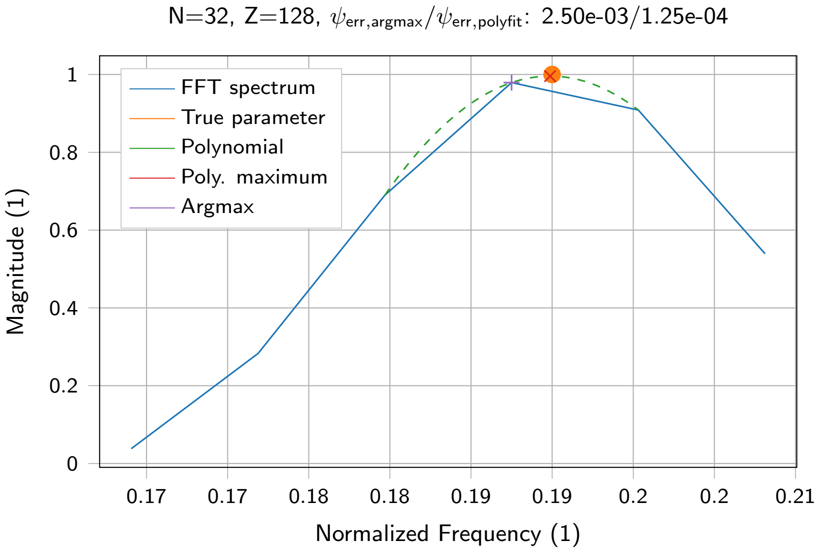
Solution to the Polyfit Excersice



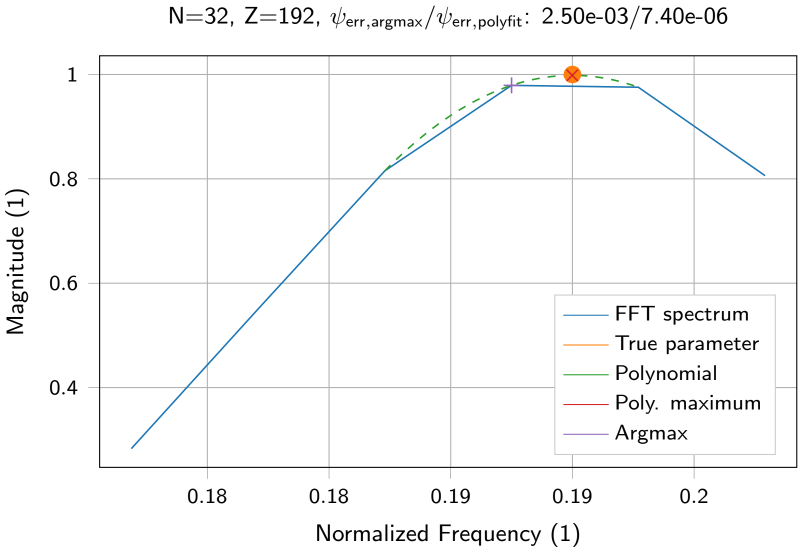
Solution to the Polyfit Excersice



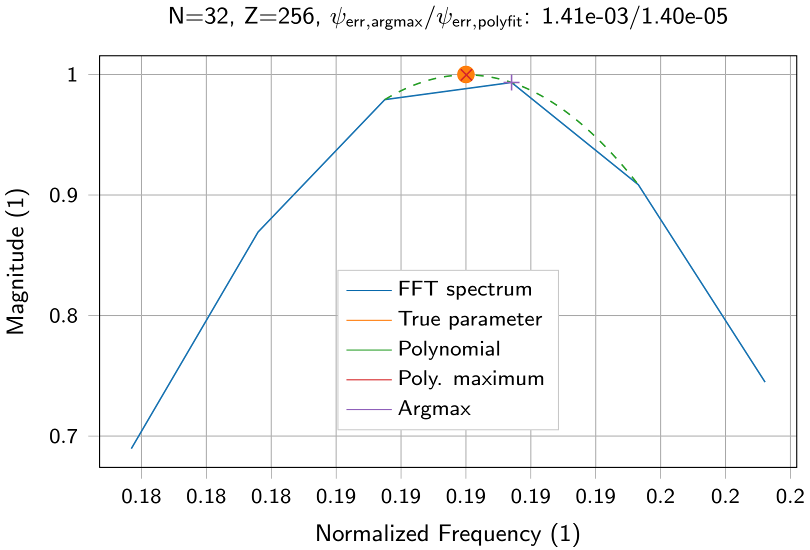
Solution to the Polyfit Excersice



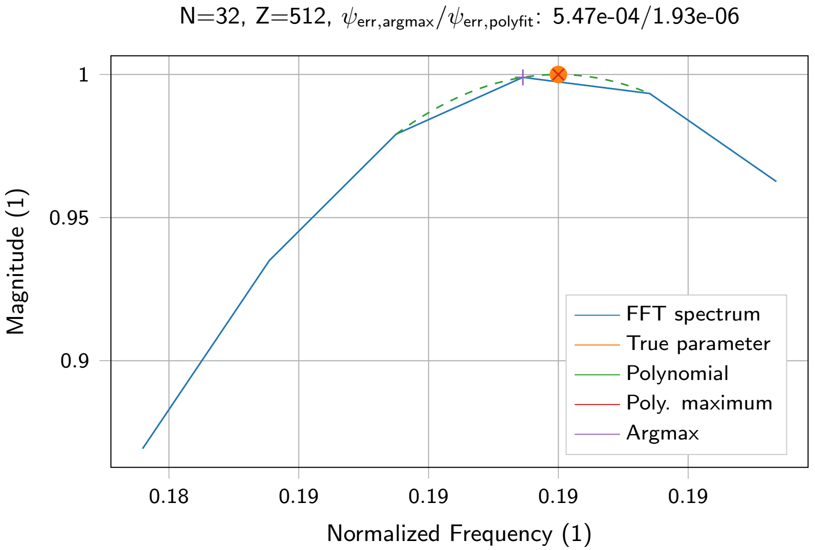
Solution to the Polyfit Excersice



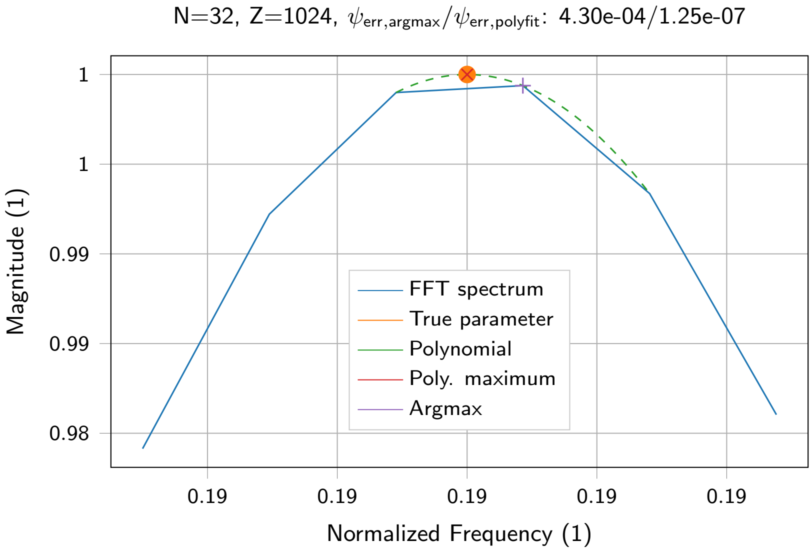
Solution to the Polyfit Excersice



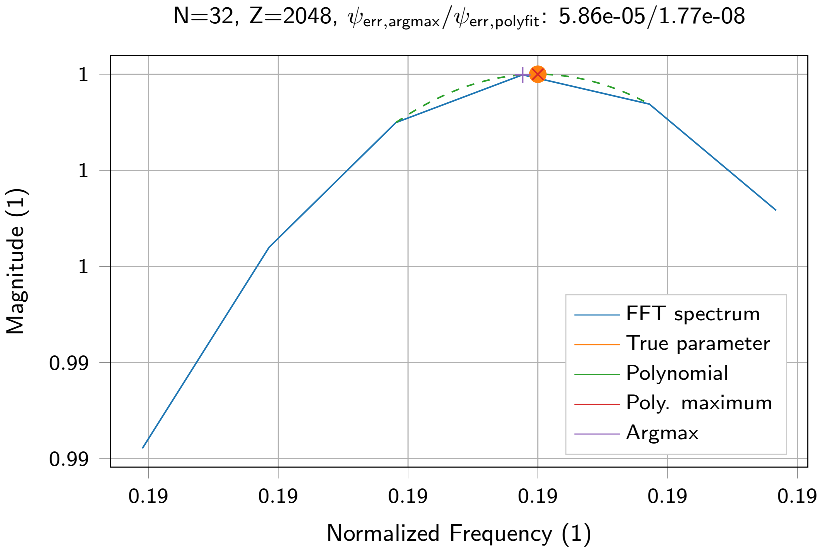
Solution to the Polyfit Excersice



Solution to the Polyfit Excersice



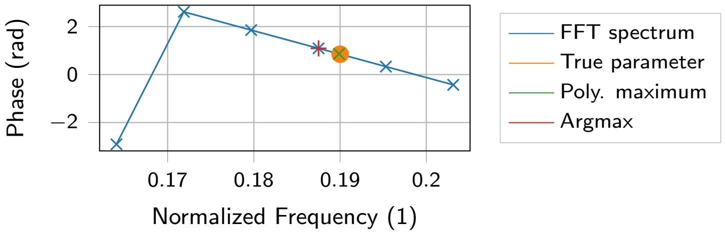
Solution to the Polyfit Excersice



Homework: Phase estimate

- Implement either the DFT sum or the Goertzel algorithm to be able to estimate a certain spectral bin.
- Use the DFT sum or the Goertzel algorithm to estimate the phase at the frequency obtained by the parabola fit.
- Plot the phase spectrum and mark the true and the estimated phase.

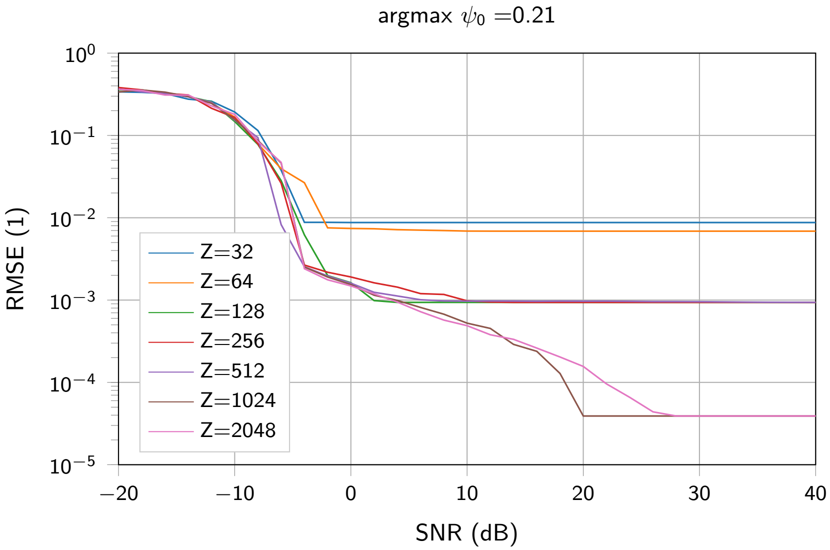
N=32, Z=128, $\phi_{\mathrm{err,argmax}}/\phi_{\mathrm{err,polyfit}}$: 2.43e-01/1.22e-02



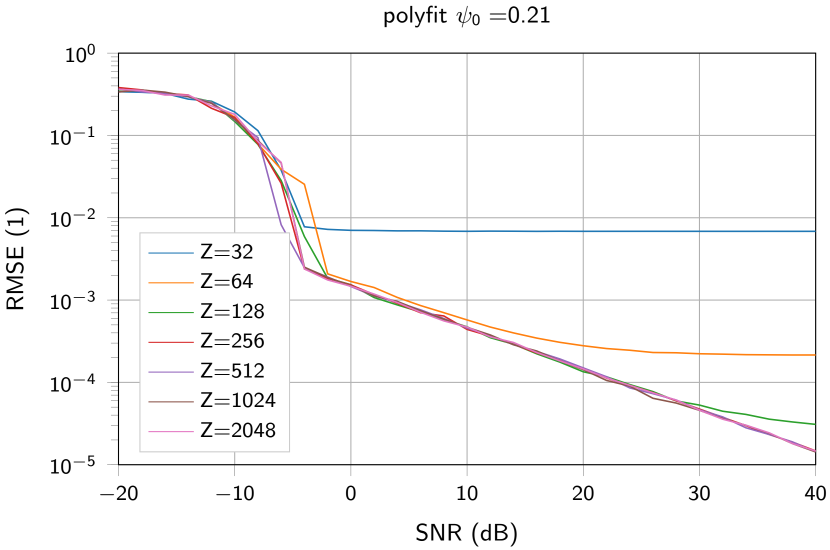
Homework: RMSE of Argmax, and Polyfit

- Consider a signal $s_{\mathrm{RX}} = s_{\mathrm{IF}} + n$ of length $N = 32$, where s_{IF} is a complex cisoid with $A = 1$, $\psi = 0.21$, $\phi = 0$ and n is zero-mean complex white Gaussian noise.
- Plot the RMSE (in logarithmic scale) of frequency estimation over SNR (-20 to 40 dB in steps of 2 dB) using the argmax method and the polyfit method for $Z_{\mathrm{f}} \in \{1, 2, 4, 8, 16, 32, 64\}$.
- Hints:
  - $\mathrm{SNR} = \frac{A^2}{2\sigma^2}$
  - Z is the zero-padding factor, i.e. a signal with N should produce $Z = Z_{\mathrm{f}} N$ FFT-bins.
  - Use at least 1000 different noise realizations for each SNR to get smooth curves for the RMSE.

Expected Plots for Homework



Expected Plots for Homework



Homework: RMSE of Argmax, and Polyfit, cntd.

- Why is a higher zero-padding factor more beneficial for the argmax method than for polyfit?
- Why does the polyfit method not decrease the RMSE (compared to argmax) for $Z < 4$?
- Why can the argmax method show the same RMSE (at high SNR) for different values of Z?
- Why does the achievable RMSE change when using the argmax method (consider $\psi \in \{0.21, 0.22, 0.23\}$)?