Questions

Any questions to exercises and homeworks from last time?

Table of Content

10 Simple Radar Simulator

- Generate Videos from Plots using Matplotlib
- Dictionaries in Python
- Dictionaries and Variable List of Arguments
- Implementing a Radar Simulator

Generate Videos from Plot using Matplotlib

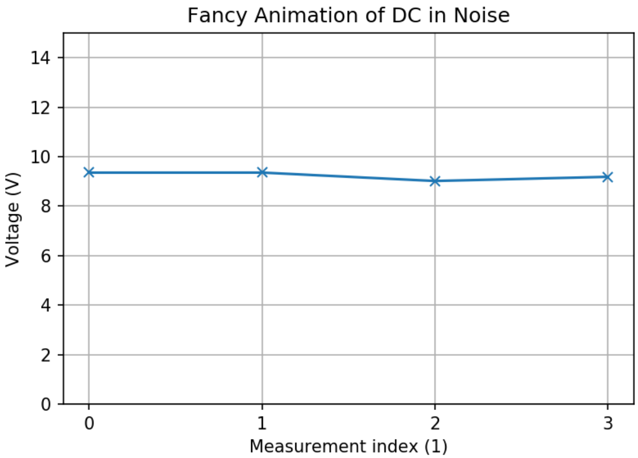
- `matplotlib.animation` provides `MovieWriter` classes to consecutively render (using `savefig`) figures to movie.
- Documentation: https://matplotlib.org/api/animation_api.html
- Generating rasterized plot and append it to a video is slow.
  - Avoid spawning new figures or axes, but change data only.
  - Parts of a figure, e.g. axes labels, might not even have changed, but checks for changes, e.g., autoscaling, or check if something was clicked or resized, are still applied.
  - `FuncAnimation` or `ArtistAnimation` can help to avoid unnecessary steps.
- Using a `MovieWriter` is sufficient for the exercises.
- FFMPEG (a powerful video creation/conversion tool) is used as backend. It is bundled with many programs and might already be installed on your system. If not:

```
conda install --channel conda-forge ffmpeg
```

With-statement and MovieWriter

```python
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FFMpegWriter as MovieWriter # ffmpeg needed

# make figure as usual
fig=plt.figure()                          # make a new figure (only once!)
np.random.seed(0)                         # reset PRNG to have reproducible a result
lines=plt.plot(np.zeros(4), 'x-')  # dummy data + store handles to modify later
plt.ylim(0,15)                            # fix ylim to avoid jumpy frames
plt.xticks(np.arange(4))                  # manually set ticks on x-axis
plt.grid()
plt.title('Fancy Animation of DC in Noise')
plt.xlabel('Measurement index (1)')
plt.ylabel('Voltage (V)')

moviewriter = MovieWriter(fps=15)                   # instanciate moviewriter
with moviewriter.saving(fig, 'test_moviewriter.mp4', dpi=150): # open file
    for j in range(100):                            # loop over frames
        new_data=np.random.rand(4)+9                # generate new data
        lines[0].set_ydata(new_data)                # update plot
        moviewriter.grab_frame()                    # append frame

plt.savefig('test_moviewriter.png', dpi=150) # after loop, thus no dummy data
```

Resulting Movie



Handling Multiple Files using `with`

- Nested `with`

```
with open_stmt1(...) as file1:
    with open_stmt2(...) as file2:
        ...
```

- Multiple opening statements can be separated with a comma

```
with open_stmt1(...) as file1, open_stmt2(...) as file2:
    ...
```

- Breaking long lines after comma for `with` does not work as it's no argument list. Use a backslash for manual line breaks.

```
with open_stmt1(...) as file1, \
     open_stmt2(...) as file2:
    ...
```

- The `as`-part is optional, e.g. movie writers don't need it while `open` for text files will provide the file handle via `as`.

Plot Update Examples

- line plots

```
# lp=plt.plot(...)          # always return tuple, even for one line
line=lp[0]
line.set_data(new_data)     # set x/y values (via 2D array or 2 arguments)
line.set_xdata(new_xdata)   # set only x values
line.set_ydata(new_yata)    # set only y values
line.axes.relim()           # optional: calculate new axes limit
line.axes.autoscale_view()  # optional: apply new axes limits
```

- imshow plots

```
# i=plt.imshow(...)
i.set_data(new_data)  # set new data (caution: no size check)
i.autoscale()         # optional: adjust colorbar limits
```

- pcolormesh plots

```
# c=pcolormesh(...)
c.set_array(new_data.ravel()) # set new data (caution: size check is done
                              # during next plotting not on setting)
c.autoscale()                 # optional: adjust colorbar limits
```

Dictionaries

- A `dict` is similar to an unordered hash-table.

```
a={}          # create empty dict, short for a=dict()
a={3: 'test', 'ha': 4.3} # create prefilled dict
a[3]='3'      # change an item
a['ha']=max   # store a function handle identified by a string
a[3]          # access selected item
a[3]='6'      # add new item (store a string identified by a number)
print(a)      # {3: '6', 'ha': <built-in function max>}
```

- https://docs.python.org/3/tutorial/datastructures.html#dictionaries
- Dictionaries are used to
  - summarize settings, e.g. `matplotlib.rcParams` stores default plot settings of `matplotlib`, or
  - fill keyword parameters for function calls.
- Use pretty print instead of `print` for nicer view of dictionaries
  - Documentation https://docs.python.org/3/library/pprint.html

```
from pprint import pprint
pprint(a)
```

Variable List of Arguments

- Dictionary `**kwargs` to fill keyword arguments.

```
def func(a, b=None, c=None):    # function with keyword arguments
    print('func:', a, b, c)

kwargs={'b': 2}                 # setup up a dictionary
func('a', **kwargs)             # fill up keyword arguments
# func: a 2 None
```

- Dictionary to accept unknown named arguments.

```
def func2(a, d=None, **kwargs):
    print('func2:', a, d)
    func(a, **kwargs)       # pass all additional arguments to sub-function

kwargs={'c': 3, 'd': 4}
func2(1, **kwargs)
# func2: 1 4
# func: 1 None 3
```

- `*args` is similar, but with a `list` for positional arguments.
- If `*args` and `**kwargs`, `*` must be placed before `**` in argument list.

Target Parameter File I

```
import numpy as np


system_parameters={
    # system parameters for a single chirp FMCW system
    'fs': 1e6,          # IF sample frequency
    'B': 250e6,         # RF bandwidth
    'fc': 24.125e9,     # carrier frequency
    'T': 1e-3,          # chirp duration
    'NF_dB': 12,        # equivalent noise temperature
    'T0': 290,          # system temperature
    'R': 50,            # reference impedance
    'NA': 8,            # number of ULA channels
    'ZA': 64,            # zero-padding to ZA FFT bins in DoA
    'T_dB': -160,       # threshold in dBV
    }

# observation timestamps, i.e. times where chirps are sent
t_arr=np.linspace(0,35,351)
```

Target Parameter File II

```python
def get_target_params(t):
    """
    Calculates targets parameters for a given point in time.

    Parameters
    ----------
    t : float
        Time in seconds for which the target parameters should be calculated.

    Returns
    -------
    Tuple containing target parameters as numpy arrays.

    A0_arr : 0D-array
        holding magnitudes in V
    r0_arr : 0D-array
        holding ranges in m
    theta0_arr : 0D-array
        holding angles of incident in rad
    """
```

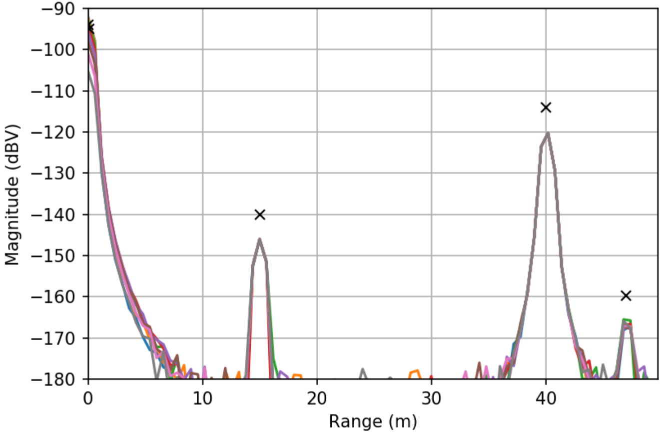
Target Parameter File III

```
# add static targets
r0_lst=[0.001, 0.1, 15]
A0_lst_uV=[20, 18, 0.1]
theta0_lst_deg=[-3, 5, -35]

# add a target moving radially
r0_lst.append(12+1*t)
A0_lst_uV.append(15**4/r0_lst[-1]**4)
theta0_lst_deg.append(31)

# add a target moving in a circle
r0_lst.append(40)
A0_lst_uV.append(2)
theta0_lst_deg.append(-25+2*t)

# convert lists to arrays and return them
A0_arr=np.array(A0_lst_uV)*1e-6
r0_arr=np.array(r0_lst)
theta0_arr=np.array(theta0_lst_deg)*np.pi/180
return A0_arr, r0_arr, theta0_arr
```

Exercise: Target Simulator

- Generate the IF signal with data from `simulator_parameters.py` (source code in previous slides and in KUSSS).
- Apply range-compression (FFT with a Hanning window), digital-beamforming (FFT with boxcar window and zero-padding) and thresholding as in the DBF exercise.
  - Hint: You can reuse most of the code from the DBF exercise.
- Produce a video of the DBF spectrum over time and mark the true positions of the targets and the detections.
  - Why does the left target vanish when the right target is approximately at the same range of 15 m?.
  - Why do the estimated position consist of multiple detections and move erratically? What could be done about it?
  - Why is the radially moving target not detected any more after it crossed with the target at 40 m?

Resulting Movie - Range Profiles



Resulting Movie - Digital Beamforming - Spectrum