Radarsignalprocessing Exercise

LVA number: 335.045

Thomas Wagner
thomas.wagner@jku.at

WS 2018

Table of Content I

Exercise Mode

Time and Date

- Blocked and interleaved with lecture of Prof. Stelzer
  - No dates for exercises in KUSSS
  - Lecture block is an hour longer

Content

- Calculation and programming exercises
  - Closely tied to the lecture's content
  - Programming in python

Grading

- Exercises (marked in special blocks) during the course

Exercise: Title

Work description.

- Homework (marked in special blocks) during course or after course

Homework: Title

Work description.

- Submit exercises and homeworks with sufficient documentation via mail.

Python

Why Python?

- for free (no license hassles)
- general purpose interpreted language
- easy to learn
- many libraries available (with installation systems)
- reasonable fast

Quickstart

- `https://docs.python.org/3.7/tutorial/`
- Windows: `https://anaconda.org/`
- Linux: Use system's package management system or `https://github.com/pyenv/pyenv`

Script Framework

```
import time

def my_cool_function(n):
    """
    Function call test

    Parameters
    ----------
    n: float
        Wait time in seconds.
    """
    print("Function called with parameter ", n)
    time.sleep(n)

if __name__=="__main__":
    N=1
    t=time.time()
    # call a function
    my_cool_function(N)
    print('Script took %d ms' % (1000*(time.time()-t)))
```

Copy-Paste Source Code

Problem: Leading white-spaces disappear when coping source code from PDFs

- Most PDF readers will gobble leading white-spaces during copy-paste from the PDF to a text processor.
- This breaks indention of source code listings and the copy-pastes python code might not run.

Solution

- In this PDF, leading invisible spaces are typeset as visible spaces using the background color.
- With this measure, at least the free of charge tools “Adobe Acrobat Reader DC” and “PDF-XChange Viewer” can copy the source code properly.

Excursion to Spyder

- Start a script
- Ipython console
- Debug line-wise
- Show variables
- Show help
- Code completion

Excursion to Anaconda and conda

- Install packages via GUI
- Anaconda prompt
- Install packages via conda CLI

Basic Flow Control

- for-loop (try yourself in ipython console)

```
for i in range(10):
    print(i)
```

- if condition (elif and else are optional)

```
a=4.0; b=3.0; c=1;
if a>=b and c>=0:
    print(a)
elif c==0:
    pass
else:
    print(c)
```

- function definition

```
def my_fun(a,b,c=None):
    if c is None
        return a+b
    else:
        return c
```

Working With Numbers

- Built in mathematical operations

`2*2`	multiplication
`17/3`	classic division returning floats
`17//3`	floor division
`2+3*2`	multiplication before addition
`2**3`	exponent
`1e6`	exponential notation with power 10
`1j`	imaginary unit
`min(a,b) max(a,b)`	minimum and maximum
`abs(a,b)`	absolute value
`math.*`	the math standard library (similar to C)

- More complex math available in the numpy and scipy module

Working With Lists and Indices

- Creating lists

```
a = []    # empty list
b = [1,] # list with a single element
squares = [1, 4, 9, 16, 25] # list with initial values
```

- Using round brackets will create tuples, i.e. immutable lists.
- List operations:

```
squares.append(36)
last=squares.pop()
squares.extend([49, 64])
l=len(squares)
squares[0]=3
```

- Accessing elements in lists

```
squares[0]    # 1
squares[-1]   # 25
squares[2:3] # [9]
squares[3:]   # [16,25]
```

For-loop Considerations

- A for-loop runs over iterables, e.g., lists and tuples, ranges, and enumerations.

```
# integers from 3 to 9
for i in range(3,10):
    print(i)

# loop over list entries
squares = [1, 4, 9, 16, 25]
for val in squares:
    print(val)

# loop with index and list entries
for i,val in enumerate(squares):
    print('square[%d]=%d' %(i,val))

# loop over multiple lists at once
for a,b,c in zip(a_list,b_list,c_list):
    print(a,b,c)
```

- A for-loop can have an `else` clause
- `continue` aborts iteration, `break` aborts loop

Programming Example

Exercise: Fibonacci sequence

Make a script to generate and print a list of the first 10 numbers of the Fibonacci sequence.

Hint: The Fibonacci sequence starts with two 1 and the next numbers are the sum of the two preceding ones.

Function and Parameters

```
def add(a,b,squared=None):
    """
    Example for optional parameter
    """
    if squared:
        return (a+b)**2
    else:
        return a+b

def add_hidden(a):
    """
    Example for scope considerations (b is not defined but used)
    """
    return a+b

if __name__=="__main__":
    print(add(2,3))                 # 5
    print(add(2,3,True))            # 25
    print(add(2,3,squared=True))    # 25

    b=33
    print(add_hidden(2))            # 35 instead of an error
```

Function with Multiple Return Values

- Function do have only one return parameter, but lists can be returned.
- Lists can be expanded automatically upon assignment to emulate multiple return values.

```
def add_sub(a,b):
    """
    Function returning a list
    """
    return [a+b,a-b]

if __name__=="__main__":
    print(add_sub(4, 10))     # prints the list [14,-6]

    res=add_sub(9.2, 34.1)
    print(res[0])             # prints the float 43.3

    plus, minus = add_sub(10,5)
    print(plus,minus)         # prints two integers 15 5
```

Docstring

- A string literal as first statement of a function/class/module is considered as its documentation.
- At least it should have
  - a sentence what it does
  - description of all parameters (functionality and datatype)
  - description of all output parameters (functionality and datatype)
- There are a few different convention for writing the doc string.
- If written in a certain convention, the help-system of IDEs can display them nicely (STRG+I in Spyder).
- numpy's convention suits best
- `https://numpydoc.readthedocs.io/en/latest/format.html#docstring-standard`

Docstring Example

```
def wmean(a,w=None):
    """
    Calculates the weighted mean of a list of elements.

    Parameters
    ----------
    a: list of floats
        List of elements

    w: list of weights
        Optional parameter, if given, its elements are
        used as weight and it must be the same size as a.

    Returns
    -------
    float
        Weighted mean of elements in a
    """
```

Modules and Imports

- A *.py file is called a module
- A module can have submodules and definitions, which are both accessed with a “.”
- Modules, submodules and definitions can be imported and renamed

```
# import modules
import numpy                        # numpy.sin(...)
import numpy as np                  # np.sin(...)
import matplotlib.pyplot as plt # import and rename

# import functions
import numpy.sin as sin             # error, it's a function
from numpy import sin               # sin(...)
from numpy import sin as cos        # to confuse people
```

Working with Numpy

Lists

- Lists are very general and often used.
- Loops over lists are not too fast.
- Python provides only a limited set of mathematical operations and none for lists.

Numpy Arrays

- Numpy provides functions similar to Matlab/Octave.
- Google “numpy matlab cheat sheet” and pick one.
- Most important consideration for Matlab programmers: indexing
  - first index 1 or 0?
  - last index N or N-1?

Basic Numpy

Useful Documentation

- https://docs.scipy.org/doc/numpy/user/quickstart.html
- https://docs.scipy.org/doc/numpy/user/basics.indexing.html

Creating Numpy Arrays

```
import numpy as np

# create empty array
arr=np.empty((3,4))                    # uninitialized array
arr=np.zeros((3,4),dtype=np.complex)  # complex numbers

# create from list
arr=np.array([0,1,2],dtype=np.float)    # automatic data conversion

# create a range
arr=np.arange(7)                       # similiar to range
```

Working with Numpy Arrays

Functions required for this lecture

- `x=np.sin(arr), x=np.cos(arr)`
- `x=np.log10(arr), x=np.exp(arr)`
- `x=np.abs(arr), x=np.angle(arr)`
- `np.max, np.argmax`
- `np.polyval`
- `np.arange, np.linspace`
- `np.power`
- `np.fft.*`
- `np.random.randn`

Basic Classes

Caution

- In python most in-memory objects are classes
- Assigning classes and passing them as argument are pointer operations

Passing a numpy pointer to a function

```
import␣numpy␣as␣np

def␣modify(array):
␣␣␣␣array[0]=100.0

if␣__name__=="__main__":
␣␣␣␣arr=np.arange(4)
␣␣␣␣print(arr)␣␣# [0␣1␣2␣3]
␣␣␣␣modify(arr)
␣␣␣␣print(arr)␣␣# [100␣␣1␣␣2␣␣3]
```

Copy Only When Needed

In some cases, a dedicated copy is needed, e.g.

- when two different operations are be applied to an object, or
- the input values should not be modified.

Copy an object

```
import numpy as np

def modify(array):
    result=array.copy()
    result[0]=100
    return result

if __name__=="__main__":
    arr=np.arange(4)
    print(arr)      # [0 1 2 3]
    res=modify(arr)
    print(arr)      # [0 1 2 3]
    print(res)      # [100 1 2 3]
```

When is a Copy Generated?

Numpy stores data internally as chunks of data. Some numpy functions return a copy, some return a view. Consult manual (CTRL+I in spyder) if in doubt!

Usually a copy is returned by

- basic math functions like +, -, ...
- basic math library functions like `sin`, `abs`, `log10`,
- some forms of indexing

Usually a view is returned

- change of shape
- change of datatype
- some forms of indexing

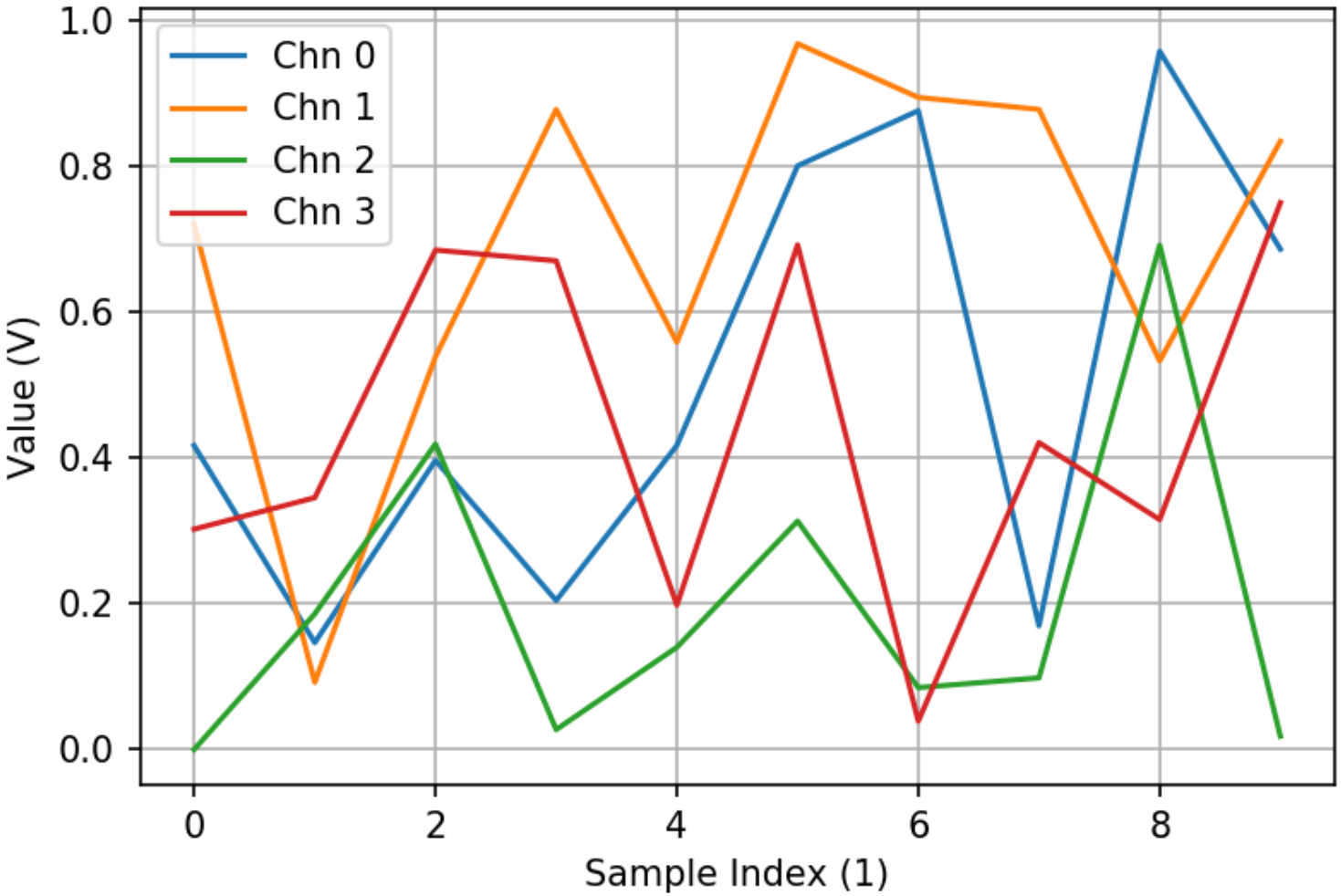
Plotting

- Matplotlib is a feature rich library for generate camera-ready plots of scientific data.
- Pyplot is a submodule to matplotlib which implements many plotting features similar to Matlab.

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

np.random.seed(1)
ADC_data=np.random.rand(10,4)
for i in range(ADC_data.shape[1]):
    plt.plot(ADC_data[:,i],
             label='Chn %d' % (i,))
plt.xlabel('Sample Index (1)')
plt.ylabel('Value (V)')
plt.legend()
plt.grid()
plt.savefig('test_matplotlib.png',dpi=150)
```

Plotting Result as png



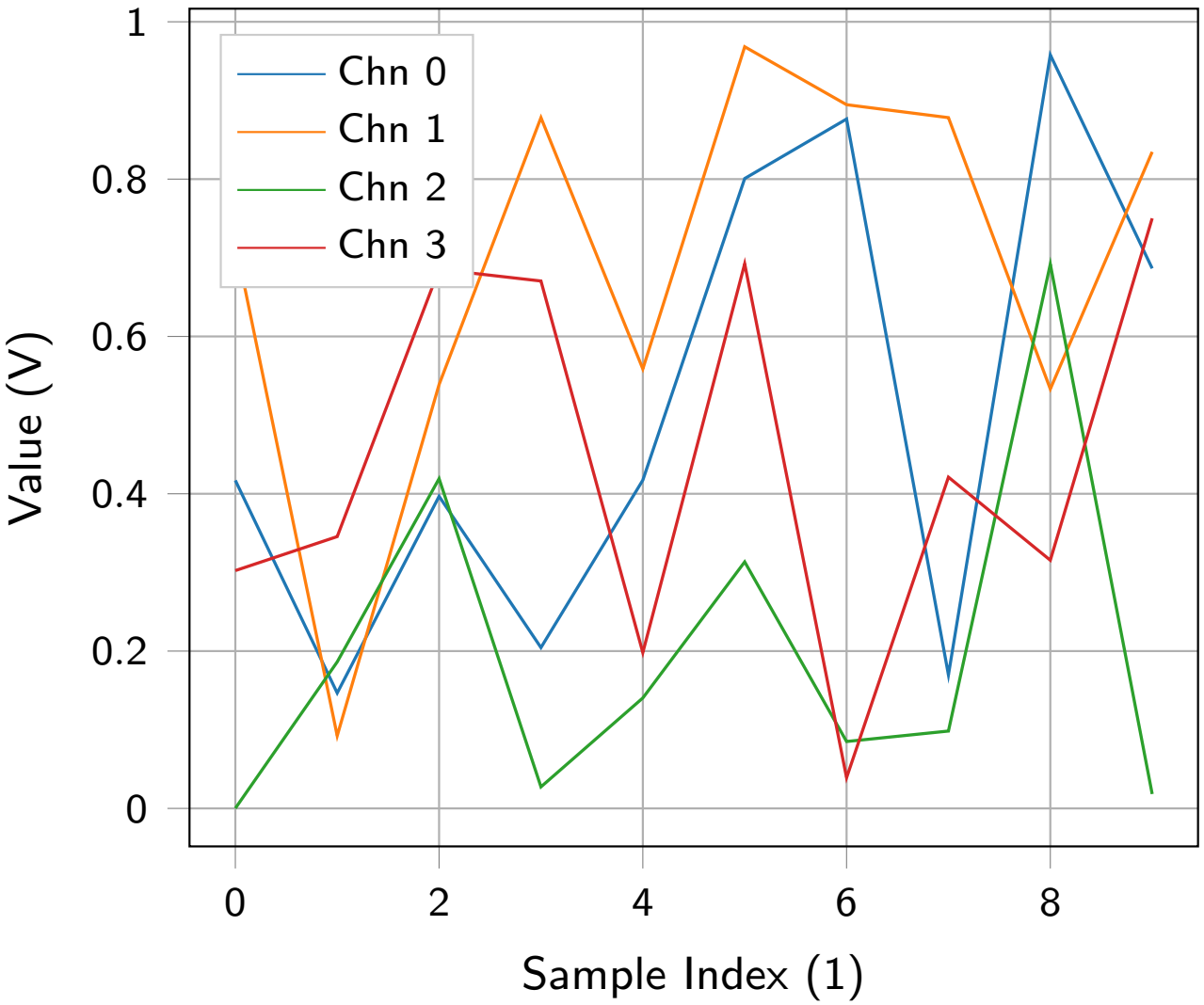
Plotting for LaTeX users

- tikz/pgfplots are powerful latex package for plotting in LaTeX
- plots look cool and can be adjusted, e.g., aspect ratio, figure width, e.t.c., after exporting
- the module matplotlib2tikz can generate

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

np.random.seed(1)
ADC_data=np.random.rand(10,4)
for i in range(ADC_data.shape[1]):
    plt.plot(ADC_data[:,i],
             label='Chn %d' % (i,))
plt.xlabel('Sample Index (1)')
plt.ylabel('Value (V)')
plt.legend()
plt.grid()
tikz_save('test_matplotlib.tikz')
```

Plotting Result as pgfplot



Function Handles and Loops

- Functions can be assigned to variables

```
plt_fct=plt.plot
plt_fct=plt.semilogx
```

- Easy to use different implementations/options without code duplication

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

R=np.logspace(1,3)
for plt_fct, name in zip((plt.plot, plt.loglog),
                         ('lin','log')):
    plt.figure()
    plt_fct(R,1/np.power(R,4))
    plt.xlabel('Range (m)')
    plt.ylabel('1/R^4 (m^-4)')
    plt.title(name)
    plt.grid()
```

Further Pyplot Considerations

- Useful Plot Functions for the Exercises (they behave similar to Matlab's plot functions)

```
plt.plot()      # linear plot
plt.semilogx() # logarithmic x-axis
plt.loglog()    # both axis logarithmic
```

- Interactive vs inline plots

```
%matplotlib qt
%matplotlib inline
```

- Bug adds extra axis at wrong place in tikz export in some cases if `savefig` was used prior `tikz_save`. `https://github.com/nschloe/matplotlib2tikz/issues/78`

Further Pyplot Considerations cntd.

MaxNLocator

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
plt.figure()
ax=plt.gca()
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.plot(np.arange(4), 3.0*np.sin(np.arange(4)*1.2))
```

Documentation

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