Signals and Systems Theory (COMM401)

Lab Manual

Prepared by: Eng. Maggie Shammaa
Supervised By: Prof. Dr. Ahmed El-Mahdy

GUC
German University in Cairo
الجامعة الألمانية بالقاهرة

Faculty of Information Engineering and Technology
Department of Communication Engineering

Arrays and Matrices

- Creating an array of certain elements:
  - np.array([array])

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

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Arrays and Matrices (cont'd)

- Creating an array of elements:
  - np.arange(start, stop, step, dtype=None): stop is not included [start, stop[. Default start is 0 and default step=1
- Examples:

```
>>> np.arange(3)

>>> np.arange(3.0)

>>> np.arange(3,7)

>>> np.arange(3,7,2)
```

Arrays and Matrices

- Creating an array of elements:
  - np.arange(start, stop, step, dtype=None): stop is not included [start, stop[. Default start is 0 and default step=1
- Solution:

```
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([ 0.,  1.,  2.])
>>> np.arange(3,7)
array([3, 4, 5, 6])
>>> np.arange(3,7,2)
array([3, 5])
```

Arrays and Matrices (cont'd)

- Creating an array of elements:
  - np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None): stop is included unless endpoint=False. Default value for number of elements is 50
- Examples:

```
>>> np.linspace(2.0, 3.0, num=5)

>>> np.linspace(2.0, 3.0, num=5, endpoint=False)

>>> np.linspace(2.0, 3.0, num=5, retstep=True)
```

Arrays and Matrices

- Creating an array of elements:
  - np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None): stop is included unless endpoint=False. Default value for number of elements is 50
- Solution:

```
>>> np.linspace(2.0, 3.0, num=5)
array([2.  , 2.25, 2.5 , 2.75, 3.  ])
>>> np.linspace(2.0, 3.0, num=5, endpoint=False)
array([2. ,  2.2,  2.4,  2.6,  2.8])
>>> np.linspace(2.0, 3.0, num=5, retstep=True)
(array([2.  ,  2.25,  2.5 ,  2.75,  3.  ]), 0.25)
```

Arrays and Matrices (cont'd)

- Creating an array of 1s:
  - np.ones((row,col)): Default dtype is float64
- Examples:

```
>>> np.ones(5)
```

```
>>> np.ones((5,), dtype=int)
```

```
>>> np.ones((2, 1))
```

```
>>> s = (2,2)
>>> np.ones(s)
```

Arrays and Matrices

- Creating an array of 1s:
  - np.ones((row,col)): Default dtype is float64
- Solution:

```
>>> np.ones(5)
array([1., 1., 1., 1., 1.])
```

```
>>> np.ones((5,), dtype=int)
array([1, 1, 1, 1, 1])
```

```
>>> np.ones((2, 1))
array([[1.],
       [1.]])
```

```
>>> s = (2,2)
>>> np.ones(s)
array([[1.,  1.],
       [1.,  1.]])
```

Arrays and Matrices (cont'd)

- Creating an array of 0s:
  - np.zeros((row,col)): Default dtype is float64
- Examples:

```
>>> np.zeros(5)
```

```
>>> np.zeros((5,), dtype=int)
```

```
>>> np.zeros((2, 1))
```

```
>>> s = (2,2)
>>> np.zeros(s)
```

Arrays and Matrices

- Creating an array of 0s:
  - np.zeros((row,col)): Default dtype is float64
- Solution:

```
>>> np.zeros(5)
array([ 0.,  0.,  0.,  0.,  0.])
```

```
>>> np.zeros((5,), dtype=int)
array([0, 0, 0, 0, 0])
```

```
>>> np.zeros((2, 1))
array([[ 0.],
       [ 0.]])
```

```
>>> s = (2,2)
>>> np.zeros(s)
array([[ 0.,  0.],
       [ 0.,  0.]])
```

Arrays and Matrices

- Creating an identity matrix:
  - np.eye(row): 2-D square matrix whose diagonal elements=1 and the rest of the elements=0
- Example:

```
>>> np.eye(2, dtype=int)
array([[1, 0],
       [0, 1]])
>>> np.eye(3, k=1)
array([[0.,  1.,  0.],
       [0.,  0.,  1.],
       [0.,  0.,  0.]])
```

Arrays and Matrices

- Creating a matrix full with a certain value:
  - np.full((row,col),value):
- Example:

```
>>> np.full((2, 2), np.inf)
array([[inf, inf],
       [inf, inf]])
>>> np.full((2, 2), 10)
array([[10, 10],
       [10, 10]])
```

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

Arrays and Matrices

- Obtaining the dimensions of an array:
  - np.shape(array)
  - array.shape
- Example:

```
>>> np.shape(np.eye(3))

>>> np.shape([[1, 2]])

>>> np.shape([0])
```

Arrays and Matrices

- Obtaining the dimensions of an array:
  - np.shape(array)
  - array.shape
- Solution:

```
>>> np.shape(np.eye(3))
(3, 3)
>>> np.shape([[1, 2]])
(1, 2)
>>> np.shape([0])
(1,)
```

Arrays and Matrices

- Changing the dimensions of a matrix:
  - np.reshape(array,(row,col))

- Example:

```
>>> a = np.array([[1,2,3], [4,5,6]])
>>> np.reshape(a, 6)

>>> np.reshape(a, 6, order='F')
```

```
>>> np.reshape(a, (3,-1))
```

Arrays and Matrices

- Changing the dimensions of a matrix:
  - np.reshape(array,(row,col))

- Solution:

```
>>> a = np.array([[1,2,3], [4,5,6]])
>>> np.reshape(a, 6)
array([1, 2, 3, 4, 5, 6])
>>> np.reshape(a, 6, order='F')
array([1, 4, 2, 5, 3, 6])
```

```
>>> np.reshape(a, (3,-1))       # the unspecified value is inferred
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Arrays and Matrices

- Returning a 1-D array from a 2-D matrix:
  - array.flatten()

- Example:

```
>>> a = np.array([[1,2], [3,4]])
>>> a.flatten()

>>> a.flatten('F')
```

Arrays and Matrices

- Returning a 1-D array from a 2-D matrix:
  - array.flatten()

- Solution:

```
>>> a = np.array([[1,2], [3,4]])
>>> a.flatten()
array([1, 2, 3, 4])
>>> a.flatten('F')
array([1, 3, 2, 4])
```

Arrays and Matrices

- Repeating an array:
  - np.repeat(array,repeats)

- Example:

```
x = np.array([1,2,3,4])
b=np.repeat(x, [2])

c=np.repeat(x, 2, axis=1)

d=np.repeat(x, 2, axis=0)
```

Plotting

- Plotting a discrete signal:
  matplotlib.pyplot.stem(*args, linefmt=None, markerfmt=None, basefmt=None,bottom=0, label=None, use_line_collection=True, orientation='vertical')

- Linefmt → String, optional

A string defining the color and/or linestyle of the vertical lines

- Markerfmt → String, optional

A string defining the color and/or shape of the markers at the stem heads.

- Basefmt → String and default: 'C3-' ('C2-' in classic mode)

A format string defining the properties of the baseline.

- Bottomfloat → default: 0

The y/x-position of the baseline (depending on orientation).

- use_line_collection → Boolean and default: True

If True, store and plot the stem lines as a LineCollection instead of individual lines, which significantly increases performance

Function Syntax

General Syntax

```
def functionName (Input_Parameters):
    "Function_docstring"

    print('Test Function')
    # Code inside the function
    # It must be indented !

    return None# optional return
```

Example

```
def Add_3_Nums (Input_1,Input_2,Input_3):
    "This function adds the three input parameters and returns their sum"

    Addition = Input_1+Input_2+Input_3 # Code inside the function

    return Addition
```

Numpy.convolve()

numpy.convolve(a, v, mode='full') → Returns the discrete, linear convolution of two one-dimensional sequences.

Parameters

- a(N,) array_like → First one-dimensional input array.
- v(M,) array_like → Second one-dimensional input array.
- mode{'full', 'valid', 'same'}, optional

1. 'full':

By default, mode is 'full'. This returns the convolution at each point of overlap, with an output shape of (N+M-1,). At the end-points of the convolution, the signals do not overlap completely, and boundary effects may be seen.

2. 'same':

Mode 'same' returns output of length max(M, N). Boundary effects are still visible.

3. 'valid':

Mode 'valid' returns output of length max(M, N) - min(M, N) + 1. The convolution product is only given for points where the signals overlap completely. Values outside the signal boundary have no effect.

Other Important Functions

- **np.multiply(A, B)** takes 2 arrays A and B and return the multiplication of them

- **fourier_series(f, limits=None, finite=True)**

  limits : (sym, start, end)
  sym denotes the symbol the series is computed with respect to.
  start and *end* denotes the start and the end of the interval. Default range is specified as −π and π.

```
>>> np.logical_and(True, False)
False
>>> np.logical_and([True, False], [False, False])
array([False, False])
```

```
>>> x = np.arange(5)
>>> np.logical_and(x>1, x<4)
array([False, False,  True,  True, False])
```