NUMERICAL SOLUTION AND SIMULATION IN ENGINEERING WITH PYTHON

By:

Ayoob Salari

Chapter 2

Table of Contents

SECTION 1

Getting started with Python

1.1 Spyder Desktop

Once you run the Spyder, it appears with a number of windows. Spyder default view shows the following windows:

Editor This is similar to m-file in Matlab where we will write/edit our code and store.

Console This is similar to command window of Matlab where we can enter commands

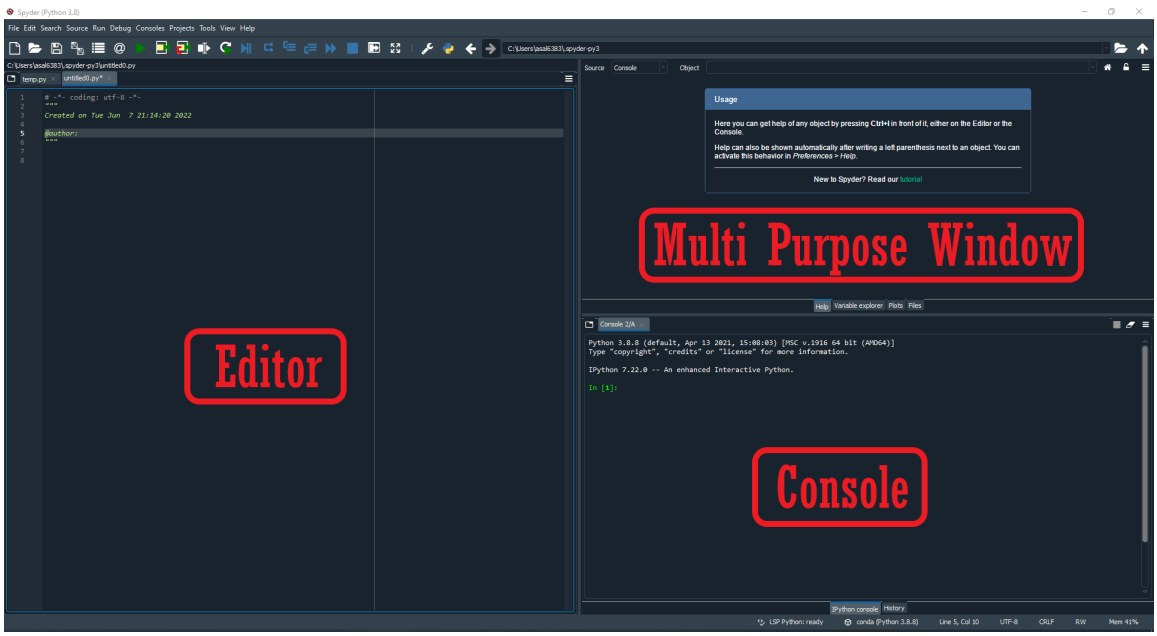
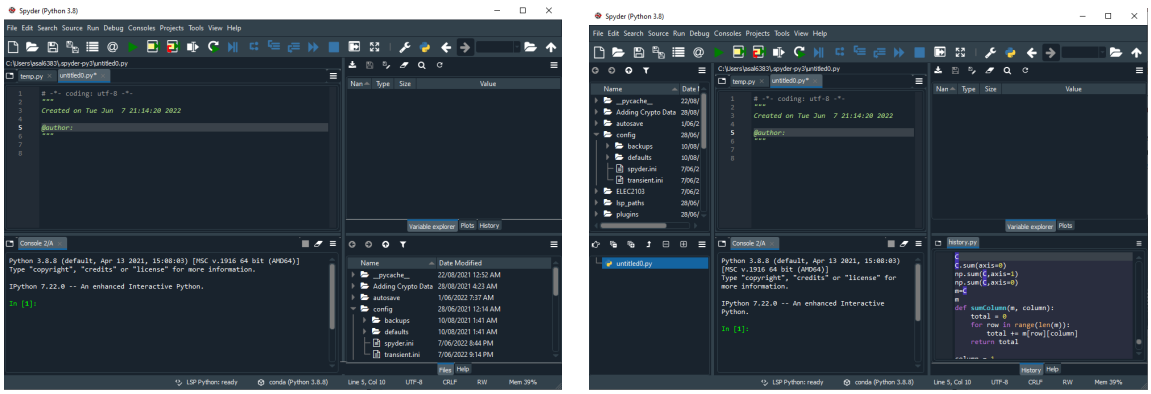


Figure 1: Spyder Default Layout

for processing and also see the outputs/errors of our code.

Multi Purpose Window Displays variables, plots, files and help section.
You can change the background theme by going to *Tools* > *Preferences* > *Appearance* > *Interface theme* . Spyder provides multiple layouts to enable programmers who have expertise with other programming languages feel at ease with Python. To change Spyder layout, go to *View* > *Window Layout* (Fig. 2).



(a) RStudio Layout (b) Matlab Layout

Figure 2: Spyder Layouts

1.2 IPython Console

The IPython console is one of the most useful windows, since it enables you to run commands and interact with data inside IPython interpreters. To start a new console, from the Consoles menu click New console (default settings), or press Ctrl-T (Command-T on macOS) while the console is active [1]. To clear only the console (not the variables), you can press Ctrl-L and to remove all the variables (not to clear shell) you can use the shortcut Ctrl-Alt-R. Using the Consoles menu is another way for doing these tasks.
To enter a command, type the command (followed by Enter) at the prompt In [1]: . For instance:

```
In [1]:  3 + 4
```

```
Out [1]:  7
```

We can also assign these value to a variable as follow

```
In [2]:  x = 3 + 4
```

To see the value of any variable, enter its name in the console. For instance,

```
In [3]:  x
```

```
Out [3]:  7
```

[1] The remainder of this book defines keyboard shortcuts for the Windows operating system. Users of MacOS should replace Ctrl with Command and Alt with Option

Once created, the variable will remain in the variable explorer (and can be viewed in the Multi-Purpose window) until they are cleared. A variable's name must begin with either a letter or an underscore. A variable cannot begin with a numeral. A variable name is limited to alpha-numeric and underscore characters (A-z, 0-9, and _).
NB: From here on, we will drop the use of In []: , and you can intepret all code in boxes as being either input into the command prompt or written in a script or function depending on context.
Pressing Ctrl-I while the cursor is on an object opens documentation for that object in the **help** pane. You may find the Ctrl-U, `Home`, `End`, ←, ↑, → and ↓ keys useful when entering commands. Up and down arrow keys go backward and forward in command history, respectively. Ctrl-U can be used to delete text on the current line, the left and right arrow keys to navigate between characters, and the `Home` and `End` keys to get to the beginning and end of a line.

1.3 Numbers, Variables and Operations

By default, Python treats every number with a decimal point as a double precision floating point number. Double precision numbers have 53 bits (16 digits) of accuracy. To see the minimum, maximum, epsilon and other system specific parameters in Python, we need to use the `sys` module. This module is included in the standard library, thus there is no need for further installation.

```
import sys
sys.float_info
```

```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16,
radix=2, rounds=1)
```

For complex numbers, Python utilises the "cmath" module.

```
 # importing "cmath" for complex number operations
import cmath
 # Initializing real numbers
x = 2
y = 3
 # converting x and y into complex number
c = complex(x,y);
print(c)
```

```
(2+3j)
```

The real, imaginary, magnitude, and phase components of a complex number "c" can be found using `c.real`, `c.imag`, `abs(c)`, and `cmath.phase(c)`, respectively. Note that

angles are in radians. Furthermore, we can use `cmath.polar(c)` and `cmath.rect(r, phi)` to convert complex numbers from rectangular to polar form and vice versa. Common mathematical operations are performed with the help of arithmetic operators. Table 1 is a quick reference of basic math operators in Python. When comparing

Operator	**Name**	**Description**	**Example**
+	Addition	Add numbers	1+2+3
-	Subtraction	Subtract a numbers	8-3-2
*	Multiplication	Multiply numbers	2*3
/	Division	Divide numbers	7/4
%	Modulus	Get the remainer of the left hand operand divided by the right hand operand	10%3
**	Exponent	Performs exponential power operations	3**3
//	Floor Division	Division with the decimal point removed	6//5

Table 1: Arithmetic operators

values, relational operators are utilised. Depending on the criteria, it returns True or False. Comparison operators are another name for these operators. Python's relational operators are included in Table 2.

Operator	**Name**	**Description**	**Example**
>	Greater than	True if the left operand is greater than the right	x > y
<	Less than	True if the left operand is less than the right	x < y
==	Equal to	True if both operands are equal	x == y
!=	Not equal to	True if operands are not equal	x != y
>=	Greater than or equal to	True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to	True if left operand is less than or equal to the right	x <= y

Table 2: Relational operators

Conditional statements can be combined using logical operators. Table 3 includes the Python conditional operators.

Operator	**Name**	**Description**	**Example**
and	Logical AND	Returns True if both statements are true	x > 3 and x < 7
or	Logical OR	Returns True if one of the statements is true	x > 3 or x < 7
not	Logical NOT	Reverse the result, returns False if the result is true	not (x > 3 and x < 7)

Table 3: Logical operators

Example 1.1 Arithmetic Operations

1. What are the values of $8/2$ and $8\backslash 2$?
2. Find the magnitude and phase in radians of $3 + j6$.
3. What are the real and imaginary parts of $5\angle 1.2$, where the angle is in radians?

Solution:

Example 1.2 Arithmetic Operations

In a steady-state circuit problem involving phasors, a voltage V is given by

$$V_s = E + ZI_L$$

If $E = 100 + 20j$V, $I_L = 5 + 2j$A and $Z = 1.6\angle 30°$ show (i.e. write the code) how to determine the magnitude and phase (in degrees) of V_s.
Hint: Remember to convert between degrees and radians!

Solution:

1.4 Arrays

NumPy is a Python-based array-processing library. It includes a high-performance multidimensional array object as well as utilities for manipulating these arrays. It is the core Python library for scientific computing. Numpy is mostly used to create arrays with n dimensions.
A vector can be viewed as a list of numbers, and vector algebra as operations done on the list's numbers. In other words, vector is numpy's 1-dimensional array. We can define a vector as

```
 # importing numpy
import numpy as np
 # creating a horizontal list
list_horizontal = [4, 1, 3]
 # creating a vertical list
list_vertical = [[8],
                 [2],
                 [5]]
 # row vector
row_vector = np.array(list_horizontal)
 # column vector
```

```
column_vector = np.array(list_vertical)
 # showing vectors
print("Row-Vector:", row_vector)
print("Column-Vector:", column_vector)
```

```
Row-Vector:  [4 1 3]
Column-Vector:  [[8]
                 [2]
                 [5]]
```

Basic vector arithmetic can be viewed as an element-wise operation between two vectors of equal length that results in a new vector of same length.

```
import numpy as np
x = [4, 6, 8]
y = [1, 2, 4]
v1 = np.array(x)
v2 = np.array(y)
 # addition: x + y = (x1 + y1, x2 + y2, x3 + y3)
addition = v1 + v2
print("Addition:" + str(addition))
 # multiplication: x * y = (x1 * y1, x2 * y2, x3 * y3)
multiplication = v1 * v2
print("Multiplication:" + str(multiplication))
 # division: x / y = (x1 / y1, x2 / y2, x3 / y3)
division = v1 / v2
print("Division:" + str(division))
```

```
Addition:[ 5 8 12]
Multiplication:[ 4 12 32]
Division:[4. 3. 2.]
```

The dot product, also known as the scalar product, is an algebraic operation that takes two equal-length numerical sequences and outputs a single integer.

```
import numpy as np
x = [4, 6, 8]
y = [1, 2, 4]
v1 = np.array(x)
v2 = np.array(y)
 # dot product: x . y = (x1 * y1 + x2 * y2 + x3 * y3)
dot_product = v1.dot(v2)
 # Alternatively we can use np.dot(v1,v2)
print("Dot-Product:" + str(dot_product))
```

```
Dot-Product:48
```

Similarly, we can define 2-D list and use numpy to create a matrix.

```
import numpy as np
 # creating a 2-D list
 # Matrix = [[Row 1],[Row 2],...,[Row N]]
A = [[1, 2], [3,4]]
B = [[5, 6], [7,8]]
X = np.array(A)
Y = np.array(B)
 # multiplication:
multiplication = X * Y
 # Alternatively we can use np.multiply(X,Y)
print("Multiplication:" + str(multiplication))
 # dot product:
dot_product = np.dot(X,Y)
 # Alternatively we can use np.matmul(X,Y)
print("Dot-Product::" + str(dot_product))
```

```
Multiplication:[[ 5 12]
                [21 32]]
Dot-Product::[[19 22]
              [43 50]]
```

Example 1.3 Array Manipulation

Write down explicitly the calculations involved in computing $X*Y$ and $X.Y$ shown above, i.e. element-by-element.

Solution:

The function of `numpy.sum(X)` can be used to add up all of the matrix's elements. `numpy.sum(X,axis)` provides either the column sum of the matrix X or the row sum of the matrix X when axis is 0 and 1 respectively. Furthermore, functions such as `numpy.sqrt(X)`, `numpy.sin(X)`, and `numpy.exp(X)` operate on a matrix, element by element, to produce a matrix of the same dimension.

The transpose of a matrix X can be obtained using `X.T` or `numpy.transpose(X)`. When we define matrix using `ndarray` class, the `X.conj().T` returns the conjugate transpose of matrix X (Hermitian of matrix X). Alternatively, we can use Numpy's matrix class and use `X.H` to obtain the conjugate transpose.

```
import numpy as np
X = np.array([[1-1j, 2+2j], [4-3j,4]])
print("X=",X)
```

```
sum_all = np.sum(X)
print ("Sum-All:",sum_all)
print ("Column-Sum:",np.sum(X,axis=0) )
print ("Row-Sum:",np.sum(X,axis=1) )
print ("Transpose:",X.T )
print ("Conjugate-Transpose:",X.conj().T )
 # Using matrix class for matrix manipulation
Z = np.matrix([[1-1j, 2+2j], [4-3j,4]])
print ("Conjugate-Transpose:",Z.H )
```

```
X= [[1.-1.j 2.+2.j]
    [4.-3.j 4.+0.j]]
Sum-All: (11-2j)
Column-Sum: [5.-4.j 6.+2.j]
Row-Sum: [3.+1.j 8.-3.j]
Transpose: [[1.-1.j 4.-3.j]
            [2.+2.j 4.+0.j]]
Conjugate-Transpose: [[1.+1.j 4.+3.j]
                      [2.-2.j 4.-0.j]]
Conjugate-Transpose: [[1.+1.j 4.+3.j]
                      [2.-2.j 4.-0.j]]
```

To get a sequence of the specified number falling inside a specified range, one can make use of function `range(start, stop, step)`. If the `start, step` parameters are removed, it will automatically start at 0 and increment by 1 till terminating before the specified number. It should be noted that when using `range(start, stop, step)`, all parameters should be integers.

To pass float numbers to the `start, stop, step` arguments, function `numpy.arange(start, stop, step)` can be utilized. Alternatively, we can use `numpy.linspace(start, stop, n, endpoint)` to have linearly spaced vector of n points, where `start` is the beginning position of the range, which defaults to 0, `stop` is the end of the interval range, `n` is the number of samples to create, which defaults to 50, and if the `endpoint` is set to `False`, the stop value will not be included in the output.

```
import numpy as np
a = range(1,10,2)
b = np.arange(2.5,12.5,2.5)
c = np.linspace(2.5, 12.5, num=5)
d = np.linspace(2.5, 12.5, num=5, endpoint=False)
print("a="+str(a), "b="+str(b), "c="+str(c), "d="+str(d),
   sep='\n')
 # \n for printing multiple variables in separate lines
```

```
a=range(1, 10, 2)
b=[ 2.5  5.   7.5  10. ]
c=[ 2.5  5.   7.5  10.  12.5]
d=[ 2.5  4.5  6.5  8.5  10.5]
```

There are a number of functions for generating special matrices. Observe, for example, the results of the following commands:

```
import numpy as np
a = np.zeros((2,3))
b = np.ones((2,2))
c = np.eye(2)
d = np.identity(2)
print("a="+str(a), "b="+str(b), "c="+str(c), "d="+str(d),
   sep='\n')
```

```
a=[[0. 0. 0.]
   [0. 0. 0.]]
b=[[1. 1.]
   [1. 1.]]
c=[[1. 0.]
   [0. 1.]]
d=[[1. 0.]
   [0. 1.]]
```

The size of a matrix or vector can be determined using the functions `len()` or `numpy.shape()` as is shown below

```
import numpy as np
M=[[1,4,3,7],[2,5,4,9],[3,0,6,6]]
rows = len(M)
columns = len(M[0])
print("Rows:"+ str(rows), "Columns:"+str(columns))
size=np.shape(M)
print("Size:",size)
```

```
Rows:3 Columns:4
Size: (3, 4)
```

Matrices can be used as components in constructing a larger matrix, via `numpy.concatenate((X1, X2, ...), axis)`. When axis is zero, the matrices are concatenated vertically; when axis is one, the matrices are concatenated horizontally. Arrays are flattened before usage if axis is None. The axis is set to zero by default.

```
import numpy as np
X1 = [ [1,2,3], [4,5,6] ]
X2 = [ [9,10,11], [12,13,14] ]
X_v = np.concatenate((X1, X2), axis=0)
X_h = np.concatenate((X1, X2), axis=1)
X_f  = np.concatenate((X1, X2), axis=None)
print("X1="+ str(X1), "X2="+str(X2), "Vertical-
   Concatenated="+str(X_v), "Horizontal-Concatenated="+str
   (X_h), "Flattened="+str(X_f), sep='\n')
```

```
X1=[[1, 2, 3], [4, 5, 6]]
X2=[[9, 10, 11], [12, 13, 14]]
Vertical-Concatenated = [[ 1  2  3]
                         [ 4  5  6]
                         [ 9 10 11]
                         [12 13 14]]
Horizontal-Concatenated = [[ 1  2  3  9 10 11]
                           [ 4  5  6 12 13 14]]
Flattened=[ 1  2  3  4  5  6  9 10 11 12 13 14]
```

Accessing elements of a matrix is carried out using index elements or lists in square brackets. To delete any row or column from an array, we can use `numpy.delete(X, obj, axis)` where X is the input array, `obj` is the row or column index to be removed and `axis` is the axis to be removed. Axis zero corresponds to rows, whereas axis one refers to columns.

```
import numpy as np
X = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
a = X[1,2]
R1 = X[0,:]
Sub_matrix = X[0:2,1:3]
X_rem = np.delete(X, 0, 1)
print("X="+str(X),"a="+str(a),"First-Row="+str(R1), "Sub-
   Matrix="+str(Sub_matrix), "X-without-first-column="+str
   (X_rem), sep='\n' )
```

```
X=[[1 2 3 4]
   [5 6 7 8]]
a=7
First-Row=[1 2 3 4]
Sub-Matrix=[[2 3]
            [6 7]]
X-without-first-column=[[2 3 4]
                        [6 7 8]]
```

High dimensional array So far we have discussed 1d-arrays and 2d-arrays. As is shown in Fig. 3, arrays can have multiple dimension. Assuming the array in Fig. 3 is called **A**,

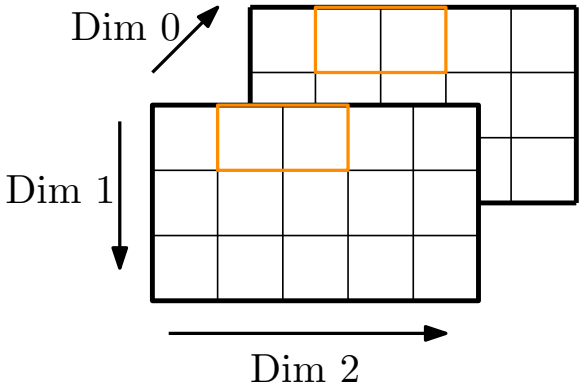


Figure 3: 3D Array

the selected elements of the array can be shown as `A[:,0,1:3]`.

Example 1.4 Relational Operators

Suppose we have a vector y of the same length as x. How could you determine the number of elements of y which are equal to the corresponding elements of x
For example, if x = [0 5 -3 7 1 8 10] and y = [1 5 3 0 0 8 -2], the answer should be 2.

Solution:

Example 1.5 Logical Operators

Generate a row vector containing 10 random numbers in the range 0 to 1.0.

1. Remove all elements less than 0.5.
2. Repeat for the case when only elements in the range 0.4 to 0.6 inclusive are to be retained.

Solution:

Example 1.6 Working with Arrays

1. Set up a 6 by 8 matrix A in which all elements on the top, bottom, left and right edges are zero and all other elements are equal to 5.
2. Set up a row vector B containing the digits of your DoB as elements. Now reverse the order of the digits in B, first using `numpy.flip()`, and then without using the Python function `numpy.flip()`.
3. Use the function `np.random.random()` to generate a 5 by 5 matrix C. Use

the function `sum` to sum all the entries, column sum and row sum. Using `numpy.trace()` and `numpy.diagonal()`, find the main diagonal and calculate the sum of diagonal elements.

4. Generate a matrix on screen with 10 rows and 3 columns. The first column contains the integers 1 to 10, the second contains the square root of the number in the first column and the third contains the square.
   Hint: You may find `numpy.vstack()` and `numpy.hstack()` useful.

Solution:

Example 1.7 Solving Linear Equations

Consider the linear equations:

$$2x_1 + 6x_3 = 10$$

$$4x_1 + 8x_2 - 5x_3 = -3$$

$$-x_1 + 10x_2 + 2x_3 = 7$$

or, in matrix form, $Ax = b$, where A is a 3-by-3 matrix and x and b are 3-vectors. Set up the matrix A and the vector b and solve for x. Use the function `numpy.linalg.inv()` to obtain the inverse of A.

Solution:

Example 1.8 Matrix Manipulation

Define a square matrix that consists of real and complex number. Calculate the transpose, conjugate transpose, trace, rank, determinant, inverse, pseudo-inverse, eigen values and eigen vectors of that matrix.

Solution:

1.5 Strings

A string in Python represents a series of Unicode characters. Unicode was created to include all characters in all languages and to standardise encoding. The Python allows for the use of either single or double quotation marks around strings. Sequences in Python may well be indexed in negative order. Last item is indicated by an index of -1, next to last by an index of -2, and so on.

```
str = 'Hello-World!'
print('String:', str)
print('First-character:', str[0])
print('Last-character:', str[-1])
print('String[1:5]=', str[1:5])
 #slicing 6th to 2nd last character
print('String[5:-2]=', str[5:-2])
```

```
String: Hello-World!
First-character: H
Last-character: !
String[1:5]= ello
String[5:-2]= -Worl
```

Python's + and * operators allow you to concatenate two separate strings or the same string numerous times.

```
str1="Hello"
str2="World!"
print ("String-1:",str1)
print ("String-2:",str2)
Concat =str1+str2
print("Concatenated:",Concat)
Append = str * 3
print("Appended:",Append)
print("The␣square␣root␣of␣{}␣is␣{}␣number.".format(-1, "
   complex"))
```

```
String-1: Hello
String-2: World!
Concatenated: HelloWorld!
Appended: HelloWorld!HelloWorld!HelloWorld!
The square root of -1 is complex number.
```

1.6 Polynomials

In Python, a polynomial is represented by a row vector of the coefficient of the powers in ascending orders. Module `numpy.polynomial.polynomial` deals with polynomials.

```
from numpy.polynomial import polynomial
p1=(2,6,0,-9,24)
p2=(1,0,2)
print("P1=", p1)
print("P2=", p2)
```

```
print("Addition:", polynomial.polyadd(p1,p2) )
print("Subtract:", polynomial.polysub(p1,p2) )
print("Multiplication:", polynomial.polymul(p1,p2) )
print("Division(P1/P2):", polynomial.polydiv(p1,p2) )
print("Differentiate:", polynomial.polyder(p1) )
print("Roots:", polynomial.polyroots(p1) )
print("Value-P1(x=2):", polynomial.polyval(2,p1) )
```

```
P1= (2, 6, 0, -9, 24)
P2= (1, 0, 2)
Addition: [ 3.  6.  2. -9. 24.]
Subtract: [ 1.  6. -2. -9. 24.]
Multiplication: [  2.   6.   4.   3.  24. -18.  48.]
Division: (array([-6. , -4.5, 12. ]), array([ 8. , 10.5]))
Differentiate: [  6.   0. -27.  96.]
Roots: [-0.3428-0.1578j -0.3428+0.1578j  0.5303-0.5510j  0.5303+0.5510j]
Value-P1-x=2 : 326.0
```

1.7 Introduction to Graphics

In this section we briefly go through graphing in Python using `matplotlib` which is the most popular visualization library for Python. We make use of `plot()` function where the first parameter refers to x-axis and second parameter is the y-axis. The following example graphs $\sin(x)$:

```
import numpy as np
import matplotlib.pyplot as plt
t=np.linspace(0,5*np.pi,51)
x=np.sin(-0.5*t)
plt.plot(t,x)
plt.show()
```

First we use the function `linspace` to creates a vector of 51 values (50 intervals) linearly spread over the range specified. When plot is used, as here, with two vector arguments, it plots the first along the abscissa and the second along the ordinate. 51 points are plotted, joined by straight lines.

The appearance of the graph can be controlled in a number of ways. The colour, marker and line style can be specified. A title, axis labelling, a legend, grid lines and other features can be added to a graph. By simply plotting the curves one on top of the other, we can make a superimposed plot of numerous curves. Suppose t is defined above and observe the effects of the command:

```
plt.plot(t, x, "go--", linewidth=2, markersize=12, label="
   sin")
plt.plot(t, z, color="r", label="cos")
```

```
plt.xlabel("Angle")
plt.ylabel("Magnitude")
plt.title("Sine␣and␣Cosine␣functions")
plt.legend()
plt.show()
```

The string tells Python to use g the colour green, o as a marker and -- a dashed line. The default colour and line style are blue and solid respectively. If no marker is specified, none is used. If a marker is specified, but no line style, no line is drawn.
Multiple axes may be shown in a single figure window. Let's suppose we want to plot $\sin(x)$ and $\cos(x)$ vertically. Using subplot, we can break the window down into many smaller axes. Let's assume the definition of t given above. The following supplementary instructions bring about the expected outcome

```
y1 = np.sin(t)
y2 = np.cos(t)
plt.subplot(211)
plt.plot(y1)
plt.subplot(212)
plt.plot(y2)
plt.show()
```

Example 1.9 Plotting Figures

Define vectors to represent the following:

1. The values of t ranging from 0 to 10 at intervals of 0.05,
2. $x(t) = \exp(-0.5t)$,
3. $y(t) = \cos(8t)$ and
4. $z(t) = 10\, x(t)\, y(t)$.
5. Plot z versus t. Add a title "An exponentially decaying sinusoid", label the x-axis "Time", label the y-axis "Value of z", set the range of z values to be -12 to 12 and turn on the grid lines (Use the functions `title`, `xlabel`, `ylabel`, `axis`).
6. Use `subplot` to plot x, y and z of Exercise 6 on separate axes in the same *Figure window*, one above the other.

Solution:

Example 1.10 Plotting Geometric Function

Plot the geometric figure described by

$$\frac{(x-3)^2}{36}+\frac{(y+2)^2}{81}=1$$

in the x-y plane.
Hint: a parametric representation is:

$$x(t)=3+6\cos(t),\quad y(t)=-2+9\sin(t);\quad 0\le t\le 2\pi$$

Add a `grid` and use `axis` to make intervals on the x and y axes equal.

Solution:

1.7.1 3D Line Graph

The function `plot3D` is similar to `plot` except it takes triples instead of pairs and plots in a 3D space. As an example, the graph of $r(t)=(t\cos(t),t\sin(t),t)$, a parameterize space curve, can be plotted over the interval using:

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

t = np.linspace(0,4*np.pi,200)
x = t*np.cos(t)
y = t*np.sin(t)

fig = plt.figure()
ax = plt.axes(projection ='3d')

ax.plot3D(x, y, t, 'red')
ax.set_title('3D-line-plot')
plt.show()
```

1.7.2 Plotting Functions of Two Variables

Suppose we wish to display a function of two variables . The plot of z versus x and y is a surface in 3 dimensions. To plot such a function, we need to generate a matrix of x and y values. The `meshgrid` function is supplied for this purpose.

Example 1.11 **3D Surface Plot**

Plot the function $f(x, y) = \sin(x) \times \cos(y)$ for $-2\pi \leq x \leq 2\pi$ and $-2\pi \leq y \leq 2\pi$.

Solution:

Exercise 1.1

Plot the function $f(x, y) = \sin(x) \times \cos(y)$ for $-2\pi \leq x \leq 2\pi$ and $-2\pi \leq y \leq 2\pi$ as 3D line plot.

Solution:

1.7.3 Symbolic Plotting

Python `sympy` can be used to plot mathematical functions, when data points do not have to be specified exactly. The module `symplot` plots functions defined in symbolic mode.

```
import sympy as sp
from sympy import sin, cos
from sympy.plotting import plot as symplot

t = sp.symbols('t')
y = 0.05*t + 0.2/((t - 5)**2 + 2)

symplot(y)
```