UECM1703 INTRODUCTION TO SCIENTIFIC COMPUTING Dec 2019 Marking Guide

PART A (Answer **ONE** question only)

Q1. (a) Given

$$\mathbf{A} = \begin{bmatrix} 4 & 9 & 8 & 0 & 2 & 9 \\ 3 & 1 & 9 & 1 & 2 & 8 \\ 7 & 2 & 2 & 3 & 7 & 8 \\ 3 & 5 & 0 & 7 & 9 & 7 \\ 9 & 4 & 6 & 7 & 9 & 8 \end{bmatrix}.$$

Use the above information to **execute** the following Python commands for item (i) to item (iv) and write down the output of the execution.

(i) **`print(A[:,1])`** (2 marks)
Ans. `[9 1 2 5 4]` .... [2 marks]

(ii) **`print(A[1:4,[1,2,3]])`** (3 marks)
Ans.
```
[[1 9 1]
 [2 2 3]
 [5 0 7]]
```
.... [3 marks]

(iii) **`print(A[A<5].sum())`** (4 marks)
Ans. $(4+0+2)+(3+1+1+2)+(2+2+3)+(3+0)+4 = 27$ ...[4 marks]

(iv) **`print(A[:4,:3].sum(axis=0))`** (3 marks)
Ans. $[4+3+7+3, 9+1+2+5, 8+9+2+0] = [17, 17, 19]$ ..... [3 marks]

(v) Write down the Python command which gives the mean of rows in **A** after **execution**. (2 marks)
Ans. `print(A.mean(axis=1))` .... [2 marks]

(vi) Write down the warning message that the command

`print(A[1,:]/A[0,:].astype(np.float64))`

will raise when it is executed. (2 marks)
Ans. Since `A[0,3]` is zero, a division by zero error will be produced. [2 marks]

(b) Use Numpy array operations such as `np.arange`, etc. to write a computer program in no more than 3 lines and without using any semicolon to print the following output:

```
 1    1    1     1
 2    4    8    16
 3    9   27    81
 4   16   64   256
 5   25  125   625
 6   36  216  1296
 7   49  343  2401
 8   64  512  4096
 9   81  729  6561
10  100 1000 10000
```

(4 marks)

Ans. A sample answer is given below. If a student uses ';' or a wrong syntax, marks will be deducted.

```
col1 = np.arange(1,11).reshape(10,1)            # [1.5 marks]
B = np.hstack((col1, col1**2, col1**3, col1**4)) # [2   marks]
print(B)                                         # [0.5 mark ]
```

[Total : 20 marks]

Q2. Given the matrix

$$\mathbf{M} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}.$$

(a) **Execute** the following Python commands for item (i) to item (v) and write down the output of the execution.

(i) `print(M * M)` (3 marks)

Ans. $\begin{bmatrix} 1 & 4 \\ 9 & 16 \\ 25 & 36 \end{bmatrix}$ .... [3 marks]

(ii) `print(M @ M.T)` (3 marks)

Ans. $\begin{bmatrix} 5 & 11 & 17 \\ 11 & 25 & 39 \\ 17 & 39 & 61 \end{bmatrix}$ .... [3 marks]

(iii) `print(M[[2,1,0,1,2],:][:,[1,0,0,1]])` (4 marks)

Ans. $\begin{bmatrix} 5 & 6 \\ 3 & 4 \\ 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 6 & 5 & 5 & 6 \\ 4 & 3 & 3 & 4 \\ 2 & 1 & 1 & 2 \\ 4 & 3 & 3 & 4 \\ 6 & 5 & 5 & 6 \end{bmatrix}$ .... [4 marks]

(iv) `print(M[:2,:]==M[[2,1],:])` (2 marks)

Ans. $\begin{bmatrix} \mathit{False} & \mathit{False} \\ \mathit{True} & \mathit{True} \end{bmatrix}$ .... [2 marks]

(v) `print((M<3)|(M>4))` (3 marks)

Ans. $\begin{bmatrix} \mathit{True} & \mathit{True} \\ \mathit{False} & \mathit{False} \\ \mathit{True} & \mathit{True} \end{bmatrix}$ .... [3 marks]

(b) Write a Python program with no more than 3 lines to produce the following matrices from **M**:

$$M_1 = \begin{bmatrix} -2.5 & -1.5 \\ -0.5 & 0.5 \\ 1.5 & 2.5 \end{bmatrix}, \quad M_2 = \begin{bmatrix} -2 & -2 \\ 0 & 0 \\ 2 & 2 \end{bmatrix}, \quad M_3 = \begin{bmatrix} -0.5 & 0.5 \\ -0.5 & 0.5 \\ -0.5 & 0.5 \end{bmatrix}$$

by using the Numpy vector operation in the Python computer software. Note that M_1 is **M** subtracted by the mean of all values in **M**, M_2 is a matrix such that each column in **M** being subtracted by the mean of corresponding column, M_3 is a matrix such that each row in **M** being subtracted by the mean of corresponding row. Note that your program must work when **M** is changed to an arbitrary $m \times n$ matrix. (5 marks)

Ans.

```
M1 = M - M.mean()                                   # [1 mark ]
M2 = M - M.mean(axis=0))                            # [2 marks]
M3 = M - M.mean(axis=1)[:,None])                    # [2 marks]
```

[Total : 20 marks]

PART B (Answer **ALL** questions)

Q3. (a) **Demonstrate** the working of the following Python program script by stepping through it and write down the output.

```
P=7
print("   |",end="")
for i in range(P):
    print(f"{i:5d}",end="")
print("\n" + "-"*(3+5*P))
for i in range(P):
    print(f"{i} |", end="")
    for j in range(P):
        v = i*j % P
        print(f"{v:5d}",end="")
    print()
```

(8 marks)

Ans. The following table will be generated:

```
   |    0    1    2    3    4    5    6
--------------------------------------
0  |    0    0    0    0    0    0    0
1  |    0    1    2    3    4    5    6
2  |    0    2    4    6    1    3    5
3  |    0    3    6    2    5    1    4
4  |    0    4    1    5    2    6    3
5  |    0    5    3    1    6    4    2
6  |    0    6    5    4    3    2    1
```

Students need to perform modulus calculations with 7. .... [8 marks]

(b) The *Horner's method* is a polynomial evaluation method expressed by

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + xa_n)))).$$

Implement the Horner's method as a Python function `horner(coeffs,x)` which takes in two parameters, i.e. the coefficients $a_0, a_1, \cdots, a_n$ of the polynomial $p(x)$ as `coeffs` and a value `x` and returns the value of the polynomial $p(x)$ at `x`. (7 marks)

Ans. The marking for the implementation is as follows:

```
def horner(coeffs,x):                  # [1 mark]
    deg = len(coeffs)                  # [1 mark]
    y = coeffs[deg-1]                  # [1 mark]
    for j in range(N-2,-1,-1):         # [1 mark]
        y = coeffs[j] + x*y            # [2 marks]
    return y                           # [1 mark]
```

(c) In the late 1960's, book publishers realised that they needed a uniform way to identify all the different books that were being published throughout the world. In 1970 they came up with the International Standard Book Number system. Every book, including new editions of older books, was to be given a special number, called an ISBN, which is not given to any other book. The check digit x_{10} of a 10-digit ISBN $x_1 x_2 \cdots x_9 x_{10}$ is given by the formula:

$$x_{10} = (11 - (10x_1 + 9x_2 + 8x_3 + 7x_4 + 6x_5 + 5x_6 + 4x_7 + 3x_8 + 2x_9 \bmod 11)) \bmod 11.$$

If $x_{10} = 10$, it is represented as '`X`'. Write a Python **program script** to implement the function `checkdigit10(isbn)` which takes an ISBN of the form $x_1 x_2 \cdots x_9$ as a string of length 9 and returns a digit x_{10} as a string of length 1. (5 marks)

Ans. A sample implementation is shown below.

```python
def check_digit_10(isbn):
    isbn = list(isbn.replace('-','').replace('?',''))
    assert len(isbn) == 9
    sum = 0
    for i in range(len(isbn)):
        sum += (i+1)*int(isbn[i])
    r = sum % 11
    return str(r) if r != 10 else 'X'

print(check_digit_10('0-8044-2957'))
print(check_digit_10('0-85131-041'))
```

- Correct definition of function . . . . [1 mark]
- Proper use of for loop . . . . [1 mark]
- Demonstrate correct translation of mathematical formula to computer program . . . . [3 marks]

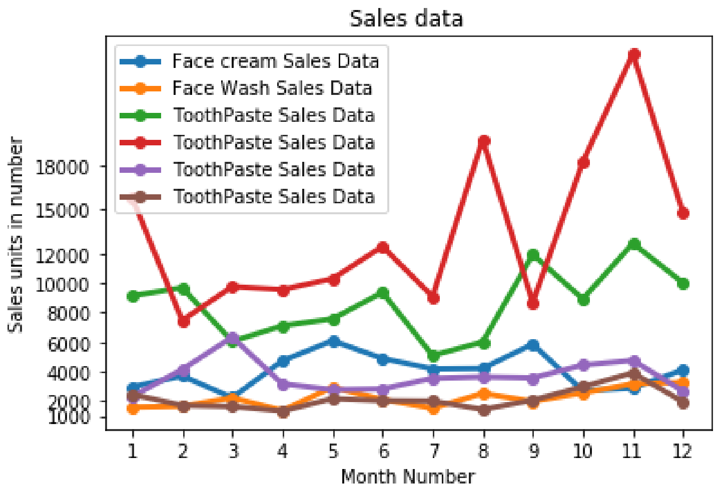
[Total : 20 marks]

Q4. (a) The following table shows the company sales data for a year (twelve months). In order to visualise the sales data, first read the the following data as a *csv* file, called "*sales_data.csv*" in the path *C:\Users\Desktop\sales_data.csv*. Then answer the following questions.

Face cream	Face wash	Toothpaste	Soap	Shampoo	Lotion	Total units	Total profit
2840	1754	9905	13681	2047	2528	29005	290050
4315	2242	8936	10557	2487	1485	27396	273960
4080	1716	8340	12641	6131	2358	33009	330090
4378	1409	6050	12076	2499	2103	38049	380490
6596	3435	7653	11138	2702	3441	32968	329680
3859	2746	5511	12251	2086	2297	31030	310300
5357	1447	7829	15010	3067	2208	31656	316560
3860	2617	6297	14793	3781	2543	39441	394410
5890	3087	6720	8240	2534	3479	36261	362610
3432	3724	9229	10660	3932	2942	43607	436070
3098	4078	10037	14627	3259	3046	51384	513840
3231	2776	10620	19016	2306	2294	51846	518460

(i) Read all product sales data and show it by using a multiline plot. Display the number of units sold per month for each product using multiline plots, i.e., Separate Plotline for each product. Note that the generated line plot must include the following properties:
X label name = Month Number
Y label name = Sales units in number

The graph should look like this. Hint: Use *marker='o', linewidth=3* in plotting.



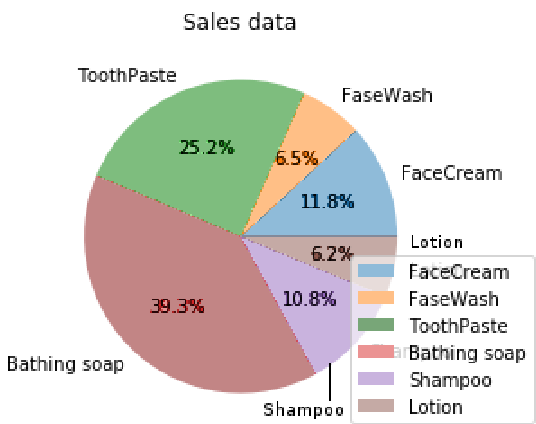
(6 marks)

Ans.

```
# For using Python packages                                # [1 mark]
import pandas as pd, numpy as np
from pylab import *
import matplotlib.pyplot as plt

filename = r"C:\Users\User\Desktop\sales_data.csv"
```

```
df = pd.read_csv(filename)                          # [1 mark]

months = np.linspace(1,12,12)                       # [1 mark]
faceCremSD    = df['Facecream'].tolist()            # [1 mark]
faceWashSD    = df['Facewash'].tolist()
toothPasteSD = df['Toothpaste'].tolist()
bathingsoapSD = df['Soap'].tolist()
shampooSD     = df['Shampoo'].tolist()
lotionSD      = df['Lotion'].tolist()

# plotting                                          # [1 mark]
plt.plot(months, faceCremSD, marker='o', linewidth=3,
    label='Face cream Sales Data')
plt.plot(months, faceWashSD, marker='o', linewidth=3,
    label='Face Wash Sales Data')
plt.plot(months, toothPasteSD, marker='o', linewidth=3,
    label='ToothPaste Sales Data')
plt.plot(months, bathingsoapSD, marker='o', linewidth=3,
    label='Bathing Soap Sales Data')
plt.plot(months, shampooSD, marker='o', linewidth=3,
    label = 'ToothPaste Sales Data')
plt.plot(months, lotionSD, marker='o', linewidth=3,
    label = 'ToothPaste Sales Data')

plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.legend(loc='upper left')
# Ticks                                             # [1 mark]
plt.xticks(monthList)
plt.yticks([1000,2000,4000,6000,8000,10000,12000,15000,18000])
plt.title('Sales data')
plt.show()
```

(ii) Calculate total sale data for last year for each product and show it using a Pie chart. Note that in Pie chart display the numbers of units sold per year for each product are in percentage.
The Pie chart should look like this.

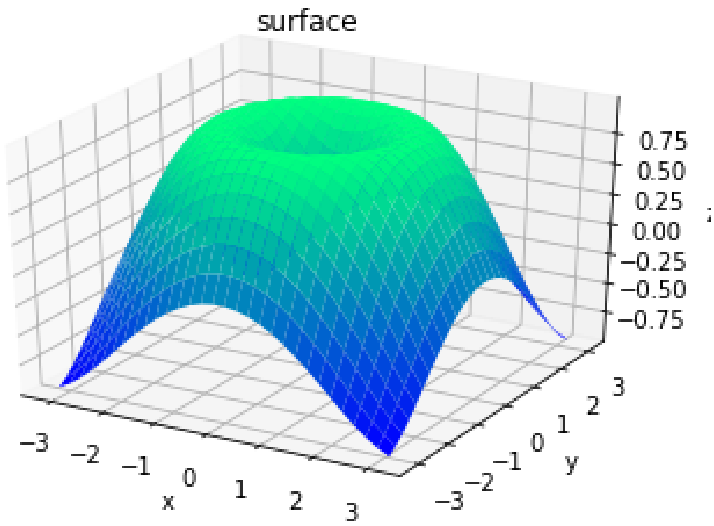


(7 marks)

Ans.

```
import pandas as pd
import matplotlib.pyplot as plt

filename = r"C:\Users\User\Desktop\sales_data.csv"
df = pd.read_csv(filename)                              # [1 mark]
monthList=np.linspace(1,12,12)                          # [1 mark]

labels = ['FaceCream','FaseWash','ToothPaste',
   'Bathing soap', 'Shampoo','Lotion']                  # [1 mark]

salesData = [df['Facecream'].sum(), df['Facewash'].sum(),
  df['Toothpaste'].sum(), df['Soap'].sum(),
  df['Shampoo'].sum(), df['Lotion'].sum()]              # [1 mark]

plt.axis("equal")                                       # [1 mark]
plt.pie(salesData, labels=labels, autopct='%1.1f%%')#[1 mark]
plt.legend(loc='lower right')                           # [1 mark]
plt.title('Sales data')
plt.show()
```

(b) Write a script to generate a 3D surface plot of the equation $z = \sin(\sqrt{x^2+y^2})$ using 3D matplotlib, where $x, y \in [-2\pi, 2\pi]$. The graph should look like this.



(7 marks)

Ans.

```
from mpl_toolkits import mplot3d                    # [1 mark]
import numpy as np
import math
import matplotlib.pyplot as plt

def z_function(x, y):                               # [1 mark]
    return np.sin(np.sqrt(x ** 2 + y ** 2))
```

```
x = np.linspace(-math.pi, math.pi, 30)          # [1 mark]
y = np.linspace(-math.pi, math.pi, 30)          # [1 mark]

X, Y = np.meshgrid(x, y)                        # [1 mark]
Z = z_function(X, Y)                            # [1 mark]
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
              cmap='winter', edgecolor='none') # [1 mark]
ax.set_title('surface');
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

plt.show()
```

[Total : 20 marks]

Q5. (a) Write a function in Python called **istriu** that receives a square matrix A as an argument and returns a Boolean True if A is an upper triangular matrix, or a Boolean False if not. Use loops in the function. (5 marks)

Ans. A sample solution is as follows.

```
def istriu(A):                          # [1 mark]
    m,n = A.shape                       # [1 mark]
    for i in range(1,m):                # [0.5 mark]
        for j in range(i):              # [0.5 mark]
            if A[i,j] != 0:             # [1 mark]
                return False            # [0.5 mark]
    return True                         # [0.5 mark]
```

(b) Consider the following system of linear equations:

$$2x_1 + 2x_2 + x_3 = 3$$
$$x_2 + 2x_3 = 1$$
$$x_1 + x_2 + 3x_3 = 2$$

(i) Write a Python code that put the above linear system in augmented matrix form. (2 marks)

Ans. `A = np.array([[2,2,1,3],[0,1,2,1],[1,1,3,2]],dtype=float)`

[2 marks]

(ii) Write a Python script that solves the above linear system by performing Gauss elimination on the augmented matrix in Part (i). Use loops in the script. (10 marks)

Ans. A sample answer is shown below.

```
import numpy as np                                  # [1 mark]

A = np.array([[2.0,2,1,3],[0,1,2,1],[1,1,3,2]])
m,n = A.shape                                       # [1 mark]

#n = m+1, assuming the coefficient matrix is square.
#Forward elimination
#Going column by column from 0 to n-3.
for j in range(n-2):                                # [1 mark]
    #Going row by row from j+1 to m-1.
    for i in range(j+1,m):                          # [1 mark]
        if A[i,j] != 0 and A[j,j] != 0:             # [1 mark]
            #Do pivoting on the row.
            A[i,:] = A[i,:] - A[i,j]/A[j,j]*A[j,:]
                                                    # [1 mark]
#Back substitution
x = np.zeros([m,1]) #Initialize x
x[m-1] = A[m-1,n-1]/A[m-1,n-2] #Compute x_m.        # [1 mark]
#Going row by row from m-2 to 0.
for i in range(m-2,-1,-1):                        # [0.5 mark]
    s=0                                           # [0.5 mark]
    #Going column by column from n-2 to i-1.
    for j in range(n-2,i,-1):                     # [0.5 mark]
        #Compute A_ij*x_i for x_i that is already found.
        s += A[i,j]*x[j]                          # [0.5 mark]
    x[i] = (A[i,n-1] - s)/A[i,i] #Compute x_i.      # [1 mark]
```

```
print(A)
print(x)
```

(iii) Write a Python script that solves the above linear system using the solve routine from the *numpy.linalg* library. (3 marks)

Ans. A sample answer is shown below.

```
import numpy as np
A = np.array([[2.0,2,1],[0,1,2],[1,1,3]])     # [1 mark]
b = np.array([3,1,2],dtype=float)             # [1 mark]
x = np.linalg.solve(A,b)                      # [1 mark]
print(x)
```

[Total : 20 marks]

Q6. (a) In the following Python script snippet, the required data for this question is given:

```
f = np.poly1d([5, 1])
x = np.linspace(0, 10, 30)
y = f(x) + 6*np.random.normal(size=len(x))
plt.plot(x, y, 'or')
plt.show()
```

Write code snippets/scripts for solving the following problems.

(i) Using Numpy arrays, find the best linear fit using vertical stacking of arrays (5 marks)

Ans.

```
a = np.vstack([x, np.ones(len(x))]).T                [2 marks]
np.dot(np.linalg.inv(np.dot(a.T,a)),np.dot(a.T,y))
                                                     [3 marks]
```

(ii) Using Linear algebra module in Numpy to find the least square fit. (3 marks)

Ans.

```
       np.linalg.lstsq(a, y)[0]                     [3 marks]
```

(iii) Using built-in Polynomial fit function to find linear fit (2 marks)

Ans.

```
       m, c = np.polyfit(x, y, 1)                   [2 marks]
```

(b) The required data for this problem is given in the below code snippet. Using the optimize module and curve fitting function in Python, write a script to find the best curve fitting model. Your written script must include the plotting of the given data along with its fitting function. (10 marks)

```
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
def func(x, a, b, c):
                return a * np.exp(-b * x) + c
xdata = np.linspace(0, 4, 50)
y = func(xdata, 2.5, 1.3, 0.5)
np.random.seed(1729)
y_noise = 0.2 * np.random.normal(size=xdata.size)
ydata = y + y_noise

# remaining codes is the answer for this problem
# Initialize proper values for parameters
# Plot the given data
# Fit for the parameters a, b, c of func:
# Get the optimized output arrays
# Plot the curve fit
# Optimize the curve fit to the region of
# 0 <= a <= 3, 0 <= b <= 1 and 0 <= c <= 0.5
```

Ans.

```
p0 = [1,2,4]                                   [1 mark]
plt.plot(xdata, ydata, 'b-', label='data')     [2 mark]
popt, pcov = curve_fit(func, xdata, ydata,p0)  [2 mark]
popt                                           [0.5 mark]
```

```
plt.plot(xdata, func(xdata, *popt), 'r-')         [0.5 mark]
popt, pcov = curve_fit(func, xdata, ydata, p0, bounds=(
     0, [3., 1., 0.5]))                           [2 mark]
popt                                              [0.5 mark]
plt.plot(xdata, func(xdata, *popt), 'g--')        [0.5 mark]
plt.show()                                        [1 mark]
```

[Total : 20 marks]