Numerical Method

National Cheng Kung University Department of Engineering Science

Instructor: Chi-Hua Yu

Name:__________________________________ **2024/04/23**

Student ID:______________________________

Midterm Mock Exam

注意事項

1. 期中考試時間為 **14:10~17:00**。
2. 本次考試可以 **open book**，使用電子書者可以攜帶 ipad。
3. 考試時皆**不可使用網路查找答案**，發現有使用網路者一律視為作弊，以零分計算。
4. 程式題部分，請繳交 **ipynb 的檔案形式**，並輸入正確的檔名。
5. 請用**學號_Midterm** 為檔名做一個資料夾**(e.g., N96091350_Midterm)**，並將程式題之**.ipynb 檔案**放入資料夾中，壓縮後上傳至課程網站**(e.g., N96091350_Midterm.zip)**。
6. 如未依照上述規則繳交作業、繳交錯誤檔案，則以零分計算，不允許要分。
7. 手寫題可跳題作答，但必須標示清楚題號，若題號標示錯，該題也會視為零分，不允許要分。如字跡潦草至助教難以辨別，則會以助教辨視為主。
8. 程式題請依照題目規定作答，若無依照題目則將該題視為零分，不允許要分。
9. 請注意作答時不要抄襲網路或是同學的答案，助教會將程式碼放入自動比對程式，只要超過 70%相似度，以抄襲處置，抄襲者與被抄襲者都以零分計算。
10. 本次閱卷將採用自動批改，命名錯誤或是無法執行將被自動判定為失敗，失去該題的分數。成功通過自動閱卷的程式碼，助教會再進行人工判讀，確定程式邏輯是否恰當，因此添加最低限度的註解可以保障作答時候的分數。

請勿抄襲，抄襲者與被抄襲者本次考試皆 0 分計算

Numerical Method

National Cheng Kung University Department of Engineering Science

Instructor: Chi-Hua Yu

Total (120%)

Part I (20%) Conceptual Problems: answer the following questions **briefly.**

1. **(8%) (a) (4%)** What is the local truncation error (error complexity) of **Riemann** integral approximation method?

   (b) (4%) What is the local truncation error (error complexity) of **Simpson's rule** integral approximation method?

2. **(6%)** In numerical differentiation and integration, what can we do if we want to improve the accuracy?

3. **(6%) (a) (3%)** What is data type of `a = {-1, 54, 11, 6}`

   (b) (3%) What is data type of `a = [2, 3, 0 ,-7]`

Numerical Method

National Cheng Kung University Department of Engineering Science

Instructor: Chi-Hua Yu

Part II (30%) Derivation Problems: give detailed derivations of the following questions.

1. **(10%)** Use the power method to obtain the largest eigenvalue and eigenvector for the matrix $A = \begin{bmatrix} 2 & 4 & 1 & 2 \\ 1 & 8 & -2 & 4 \\ 5 & -3 & 2 & -1 \\ 0 & 8 & -6 & 3 \end{bmatrix}$ Start with initial vector $[1 \quad 1 \quad 1 \quad 1]$ and see the results after 5 iterations.

2. **(10%)** Use Taylor series to show the following approximations and their accuracy $O(h^*)$:

$$f''(x_j) = \frac{-f(x_{j+3}) + 4f(x_{j+2}) - 5f(x_{j+1}) + 2f(x_j)}{h^2} + O(h^2)$$

3. **(10%)** A flat aluminum bar has constant thickness of 10 mm and a variable profile as shown in the Figure 1 below.

   (a) **(5%)** Write down each Lagrange interpolation.

   (b) **(5%)** Use Lagrange interpolation N_i to determine an expression for its area of cross-section.

$$A = \sum_{i=i}^{7} A_i N_i$$

Where A_i is the cross section of each given point, N_i is the Lagrange interpolation function of each point.

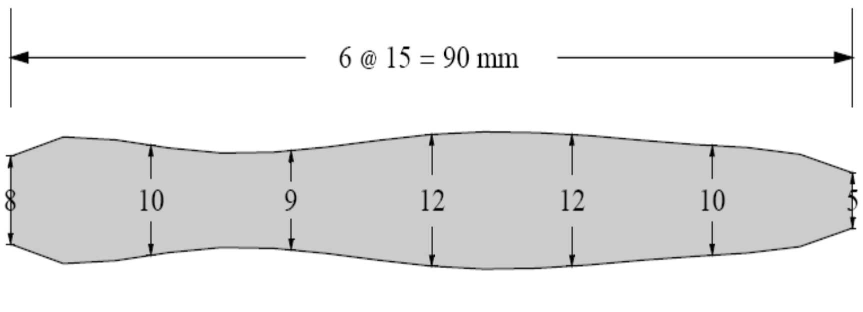


Figure 1

Part III (70%) Programming Problems.

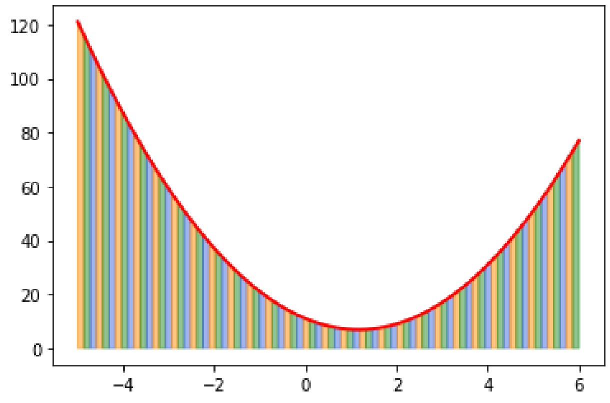
1. **(15%)** Name your Jupyter notebook `Riemann_Integral.ipynb`. Write a python program to approximate $\int_{-5}^{6} 3x^2 - 7x + 11\, dx$ with **81** evenly spaced grid points over the whole interval with the trapezoid rule. Compare this value to the exact value of **423.5**. Please plot the trapezoid integral procedure. The area below the curve is approximated by an areas of trapezoids that approximate the function.

Use the below code to plot the figure:

```
def function(x):
    f = 3*(x**2)+(-7)*x+11
    return f


…


print('Integral_riemann:', Integral_riemann)
print('error:', error)

plt.plot(x , function(x), linewidth=2, c = 'r')
for n in range(0, n-1):
    c = ['darkorange', 'forestgreen', 'royalblue']
    x_range = [x[n], x[n], x[n+1], x[n+1]]
    y_range = [0, f[n], f[n+1], 0]
    plt.fill(x_range, y_range, color=c[n%3], alpha = 0.5)

plt.show()
```

Below is the running example: (The numeric precision of the result is to three decimal places.)

```
Integral_riemann: 426.628984375
error: -3.128984375000016
```



2. **(15%)** Name your Jupyter notebook `Power_Method.ipynb`. Write a Python program to find the largest eigenvalue and the associated eigenvector by using the power method. You can use `[1, 1, 1]` as the initial vector x to start the iteration. In each iteration is usually normalized, which will make the largest element in the absolute vector equal to 1. Normalization will provide the largest eigenvalue and its corresponding eigenvector at the same time. Use the Power method to find the largest eigenvalue and the associated eigenvector of following equations:

$$a = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 5 & 7 \\ 2 & 0 & -4 \end{bmatrix}$$

Below is the running example:

(The numeric precision of the result is to three decimal places.)

```
a = np.array([[1, 3, 2], [2, 5, 7], [2, 0, -4]])

x = np.array([1, 1, 1])



Eigenvalue, Eigenvector = Power_Method(a, x)

print("The Maximum Eigenvalue:", Eigenvalue)

print("Eigenvector:", Eigenvector)
```

3. **(10%)** Name your Jupyter notebook `Gauss_Seidel.ipynb`. Write a Python program to solve the equations by using the Gauss–Seidel method.

$$\begin{bmatrix} -8 & 2 & -3 \\ 2 & 10 & 4 \\ 4 & -1 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ -5 \end{bmatrix}$$

We can assume all values of $x^{(0)}$ as 0 and using Gauss–Seidel method to substitute $x^{(1)}$ in the first iteration. After obtaining $x^{(1)}$, we continue to iterate until the difference between $x^{(k)}$ and $x^{(k-1)}$ is smaller than a predefined threshold ε = 0.0001.

Below is the running example:

(The numeric precision of the result is to three decimal places.)

```
a = np.array([[-8, 2, -3], [2, 10, 4], [4, -1, -6]])

x = np.zeros(a.shape[0])

y = np.array([2, 7, -5])



answer = Gauss_Seidel(a, x, y, it=100, epsilon=0.0001)
print("The Solve:", answer)
```

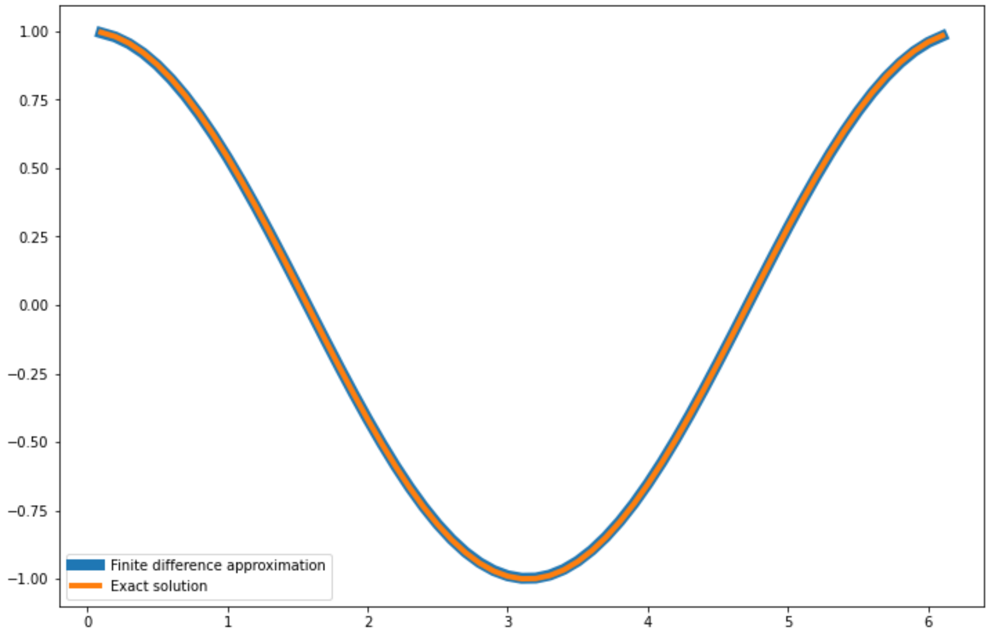
4. (15%) Name your Jupyter notebook `Central_Difference.ipynb`. Consider the function $f(x) = \sin(x)$. We know that the derivative of $\sin(x)$ is $\cos(x)$. Write a Python program to differentiate $f(x) = \sin(x)$ without using the function `np.diff`. Please write a function which name `my_central_diff` to complete the differentiation. Plot the difference and print the value of maximum error between the aforementioned numerical differentiation methods and their accuracy.

Below is the running example:

The wide of the blue line is 8

The wide of the orange line is 4

```
The maximum error is 0.001664392836042916
```

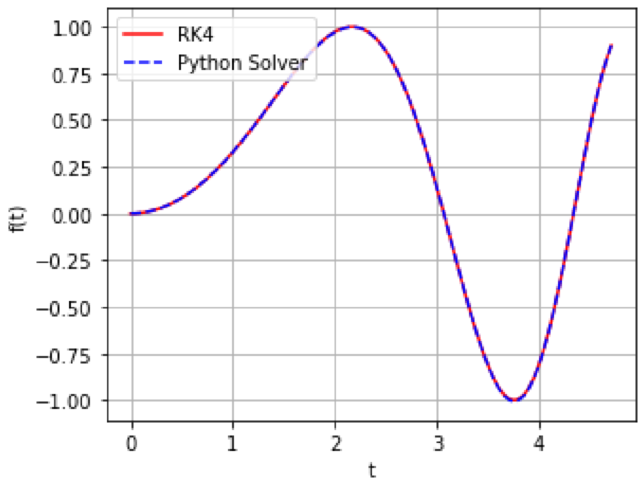


5. (15%) Name your Jupyter notebook `Runge_Kutta_4.ipynb`. Write a function `my_RK4 (ds, t_span, s0)` where `ds` is a function object, $f(t,s)$, describing a first-order differential equation, `t_span` is an array of times for which numerical solutions of the differential equation are desired, and `s0` is the initial condition of the system. Assume that the size of the state is one. The output argument should be a list of `[t,s]`, such that `t[i] = t_span[i]` for all `i`, and `s` should be the integrated values of `ds` at times `t`. Perform the integration using the **fourth-order Runge–Kutta method** and solve the following ODE.

$$\frac{dS(t)}{dt} = \frac{2}{3}t * \cos(\frac{1}{3}t^2)$$

Below is the running example:

```
from scipy.integrate import solve_ivp

def my_RK4(ds,t_span,s0):
   …
   return [t_span, y]

f = lambda t, s: 2/3*t*np.cos(1/3*t**2)
t_span = np.linspace(0, 1.5*np.pi, 100)
s0 = 0
t, s = my_RK4(f, t_span, s0)
plt.plot(t, s, "r-",label="RK4")
sol = solve_ivp(f, [0, 1.5*np.pi], [s0], t_eval=t_span)
plt.plot(sol.t, sol.y[0], "b--", label="Python Solver")
```


RK4
Python Solver
f(t)
t