National University of Computer and Emerging Sciences

Islamabad Campus

Num. Computing (CS2008) — Sessional-II Exam

Date: April 6th 2024

Course Instructor(s)

Mukhtar Ullah, Muhammad Ali

Imran Ashraf, Muhammad Almas Khan

Total Time (Hrs): 1

Total Marks: 50

Total Questions: 5

____________ Roll No

____________ Section

____________________ Student Signature

Do not write below this line

Attempt all the questions.

Q1: .... [5+5 marks]

Measurements of variables *x* and *y* are tabulated below.

k	0	1	2	3	4
x[*k*]	0.0	0.2	0.3	0.6	0.7
y[*k*]	1.000	0.935	0.858	0.504	0.368

You will need the following algorithms for this question.

Algorithm 19 Interpolating polynomial coefficients using divided differences

Given the data (x_i, y_i), $i = 0, 1, \ldots, N$
for $i = 0 : N$ **do**
  $a_i = y_i = f(x_i)$
end for
for $k = 1 : N$ **do**
  for $j = k : N$ **do**
    $a_j = (a_j - a_{k-1})/(x_j - x_{k-1})$
  end for
end for

Algorithm 18 Evaluation of a polynomial in its Newton's form

Given the point z and the nodes $x_0, x_1, \ldots, x_N$
Initialize $p = a_N$
for $k = 1 : N$ **do**
  $p = a_{N-k} + (z - x_{N-k})p$
end for
On return the variable p contains the value $P_N(z)$

1. Tabulate Newton's divided differences for the data and write down the interpolation polynomial.

$x[k]$	$y[k]$	1st-order	2nd-order	3rd-order	4th -order
0.0	1.000				
0.2	0.935	$\frac{0.935-1.0}{0.2-0.0}=-0.325$			
0.3	0.858	$\frac{0.858-1.0}{0.3-0.0}=-0.473$	$\frac{-0.473+0.325}{0.3-0.2}=-1.48$		
0.6	0.504	$\frac{0.504-1.0}{0.6-0.0}=-0.827$	$\frac{-0.827+0.325}{0.6-0.2}=-1.26$	$\frac{-1.26+1.48}{0.6-0.3}=0.73$	
0.7	0.368	$\frac{0.368-1.0}{0.7-0.0}=-0.903$	$\frac{-0.903+0.325}{0.7-0.2}=-1.16$	$\frac{-1.16+1.48}{0.7-0.3}=0.80$	$\frac{0.80-0.73}{0.7-0.6}=0.7$

The interpolating polynomial is constructed from the table:

$$P(x) = 1 - 0.325x - 1.48x(x-0.2) + 0.73x(x-0.2)(x-0.3) + 0.7x(x-0.2)(x-0.3)(x-0.63)$$

2. Code your polynomial in Python to interpolate the given data at np.linspace(0,4,20).

```
z = np.linspace(0, 4, 20);
x = [0.0, 0.2, 0.3, 0.6, 0.7];
a = [1, -0.325, -1.48, 0.73, 0.7];
p = a[4];
for k in range(1,4+1):
   p = a[4-k] + (z - x[4-k])*p
```

Q2: .... [6+4 marks]

For the data

x	y
0.5	0.4794
0.6	0.5646
0.7	0.442

a) Compute $y'(0.5)$ and $y''(0.6)$

y'(0.5)=$\frac{y(0.6)-y(0.5)}{0.1}$=0.852

y''(0.6)= $\frac{y(0.7)+y(0.5)-2y(0.6)}{0.1^2} = -20.78$

b) Write a python code that will evaluate $y'(0.6)$.

```
x=[0.5,0.6,0.7]
y=[0.4794,0.5646,0.442]
derivativey=(y[2]-y[0])/(2*(x[1]*x[0]))
```

Q3: .... [5 marks]

Apply the Euler's method with $h = 0.25$ to solve the following initial value problem

$$\frac{dy}{dt} = t^2y - 1.1y, \qquad y(0) = 1$$

over the interval $[0,1]$.

t_0=0,	y_0=1
t_1=0.25,	y_1=y_0+hf(t_0,y_0)=0.7250
t_2=0.5,	y_2=y_1+hf(t_1,y_1)=0.5369
t_3=0.75,	y_3=y_2+hf(t_2,y_2)=0.4429
t_4=1,	y_4=y_3+hf(t_3,y_3)=0.3660

Q4: .... [1+2+4+4+4 marks]

Simpson's quadrature rule is an extension of the trapezoidal rule in which the integrand is approximated by a polynomial. Simpson's 1/3rd rule is given by

$$\int_a^b f(x)\,dx \approx \left[\sum_{k=1}^{n} f(x_{2k-2}) + 4f(x_{2k-1}) + f(x_{2k})\right]\frac{\Delta x}{3}$$

a) What is the order of the polynomial used for 1/3rd rule?

b) Second order

c) Is this rule composite? Why?

Both Yes/No.

It depends upon n. Simple for n=1 and composite for n>1.

d) Utilize numpy vectorized operations to provide a python implementation of this rule as a subroutine. The prototype of this implementation should be:

```
def simpson13((f, a, b, N):
     # your implementation to approximate the integral
```

```
    return result
```

Hint: Provide only python implementation.

```
def simpson13((f, a, b, N):
    # your implementation to approximate the integral

    h = (b - a)/float(N)
    x=np.linspace(a, b, N + 1)
    s = f(x)
    s[1:N]=2*s[1:N]
    s[1:N:2]=2*s[1:N:2]
    result = h/3.0*np.sum(s)

    return result
```

Hint: Provide only python implementation.

e) Suppose we do not know how to analytically compute as:

$$\int_0^3 \sqrt{x}dx = 2\sqrt{3}$$

However, as a proud Computer Scientist we can numerically estimate the integral. Utilize Simpson's 1/3rd rule with N = 4 to approximate the following integral:

$$\int_0^3 \sqrt{x}dx$$

Hint: For this part you have to provide clear numerical steps to compute the result.
As N=2n => n = N/2 = 4/2 = 2

I[f] = h/3 * [f(x0) + 4 { f(x1) + f(x3) } + 2f(x2) + f(x4)]

h = (b-a) / N = (3-0)/4 = 0.75
x0 = 0.00; f(x0) = 0
x1 = 0.75; f(x1) = 0.866
x2 = 1.50; f(x2) = 1.225
x3 = 2.25; f(x3) = 1.5
x4 = 3.00; f(x4) = 1.732

I[f] = 0.75/3 * [0 + 4 { 0.866 + 1.5 } + 2 * 1.225 + 1.732]
I[f] = 0.75/3 * [13.646]
I[f] = 3.4115

f) What is the percentage relative error in your result?
Hint: Provide formula, clear computation steps and final result.

Relative Error = | (actual - approx) | / | actual |
=> Relative Error = | 2 * sqrt(3) – 3.4115 | / | 2 * sqrt(3) |
=> Relative Error = 0.015
=> Relative Error = 1.5 %

Q4: .... [2+2+2+2+2 marks]

Write the correct answer for each MCQ on answer sheet.

1. Simpson's rule is a numerical method used to approximate.
   A. Derivatives **B.** Integrals **C.** Limits **D.** Nothing

2. In linear spline interpolation, if the data points are (x0, y0) = (1, 3) and (x1, y1) = (2, 5), which code interpolates the function correctly ?

   A
   ```
   def linear_spline_interpolation(x0, y0, x1, y1, x):
       slope = (y1 - y0) / (x1 - x0)
       return y0 + slope * (x - x0)
   ```
   B
   ```
   def linear_spline_interpolation(x0, y0, x1, y1, x):
       slope = (y1 - y0) / (x1 / x0)
       return y0 + slope * (x /x0)
   ```
   C. Both **D** None

3. Which one is the correct implementation for trapezoidal rules.

   A.
   ```
   def trapezoidal(f, a, b, N):
       x = np.linspace(a, b, N+1)
       y = f(x)
       h = (b - a)/N
       sum = 0.0
       for i in range(1,N):
           sum += 2.0*y[i]
       sum = 0.5*h*(f(a) + sum + f(b))
       return sum
   ```
   B.
   ```
   def trapezoidal(f, a, b, N):
       x = np.linspace(a, b, N+1)
       y = f(x)
       h = (b + a)/N
       sum = 0.0
       for i in range(1,N):
           sum += 2.0/y[i]
       sum = 0.5*h*(f(a) + sum + f(b))
       return sum
   ```
   C. Both **D.** None

4. What does the `solve_ivp()` function in SciPy primarily do?
   A) Interpolations **B.** Perform numerical integration of ordinary differential equations (ODEs)
   C) Fit a curve to a set of data points **D.** Calculate the eigenvalues of a matrix

5. Which argument is used to specify the initial condition in the `solve_ivp()` function in SciPy?
   A) fun **B.** t_span **C.** y0 **D** method