دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مهندسی مکانیک

تمرین تحویلی سری دوم

درس آنالیز عددی پیشرفته

توسط:

مریم سلطانی

استاد درس:

دکتر بقاپور

آذر ۱۴۰۱

پاسخنامه زیر مربوط به تمرینات مبحث ابزارهای تقریب است. کد اصلی هر تمرین در بستر google colaboratory نوشته شده است. ضمنا فایل حاضر به عنوان رفرنس جهت استفاده دیگران در github قرار گرفته است.

Question1:

Problem 1: The specific heat c_p of aluminum depends on temperature T and tabulated as follows.

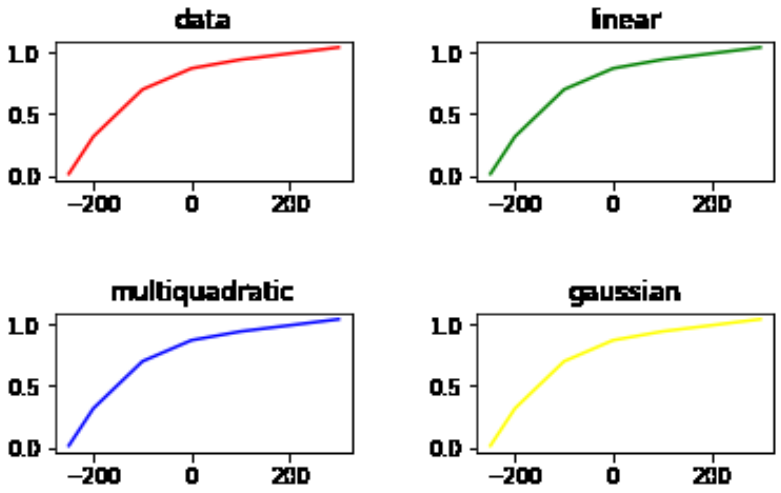
T (°C)	-250	-200	-100	0	100	300
c_p (kJ/kgK)	0.0163	0.318	0.699	0.870	0.941	1.04

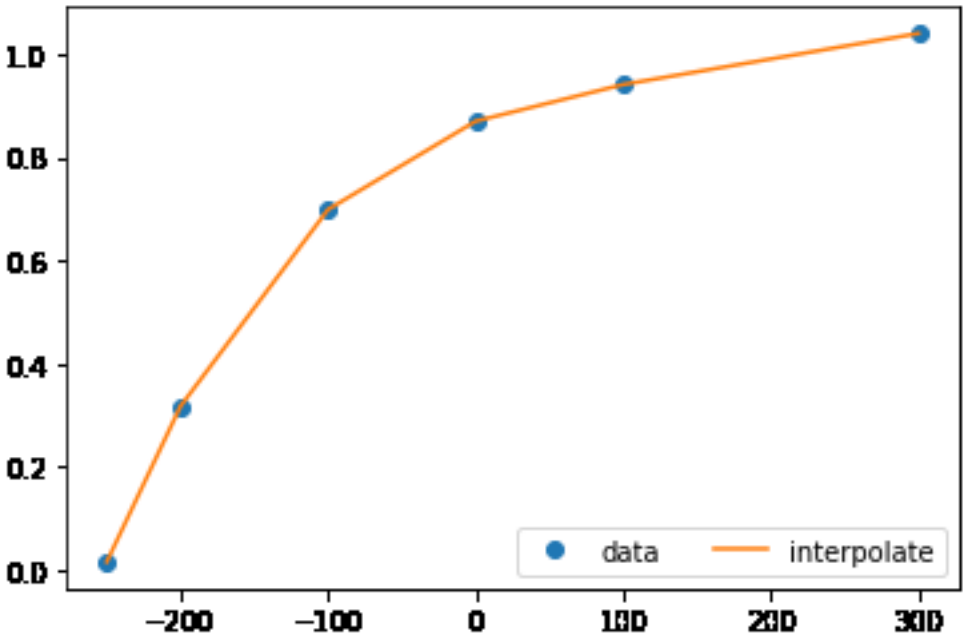
(a) Construct divide differences and find the corresponding Newton's interpolate passes throughout the points.

(b) Use radial-basis function (RBF) to build the interpolant by linear, multi-quadric, and Gaussian kernels. Take the adjustable parameter as the averaged point distance and RBF case with no additional smoothing.

(c) Plot the interpolant functions $T = -250$ °C to 300 °C and discuss the way these functions can model the behavior of the c_p changes.

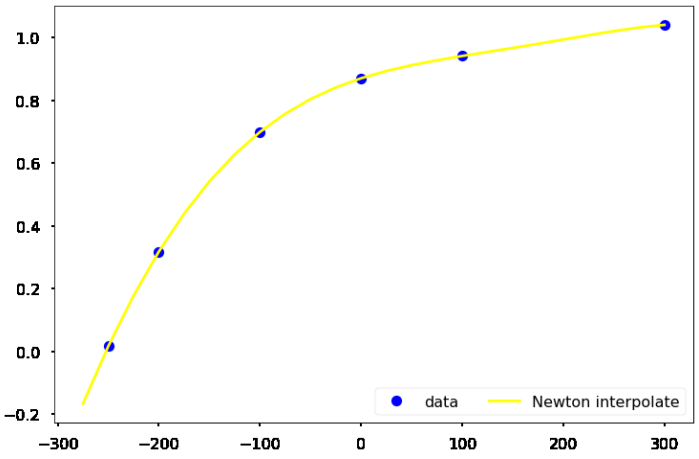
```
from scipy.interpolate import Rbf
import numpy as np
x = [-250,-200,-100,0,100,300]
y = [0.0163,0.318,0.699,0.870,0.941,1.04]
interpol_fun = Rbf(x,y,kernel='linear')
interpol_fun2 = Rbf(x,y,kernel='multiquadric')
interpol_fun3= Rbf(x,y,kernel='gaussian')
from matplotlib import pyplot

gg=interpol_fun(x)
hh=interpol_fun2(x)
ii=interpol_fun3(x)

plt.subplot(2, 2, 1)
plt.plot(x,y,'red')
pyplot.title('data')

plt.subplot(2, 2, 2)
plt.plot(x,gg,'green')
pyplot.title('linear')

plt.subplot(2, 2, 3)
plt.plot(x,hh,'blue')
pyplot.title('multiquadratic')

plt.subplot(2, 2, 4)
plt.tight_layout(pad=3.0)
plt.plot(x,ii,'yellow')
pyplot.title('gaussian')
```






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

plt.style.use('seaborn-poster')

%matplotlib inline
def divided_diff(x, y):
    '''
    function to calculate the divided
    differences table
    '''
    n = len(y)
    coef = np.zeros([n, n])
    # the first column is y
```

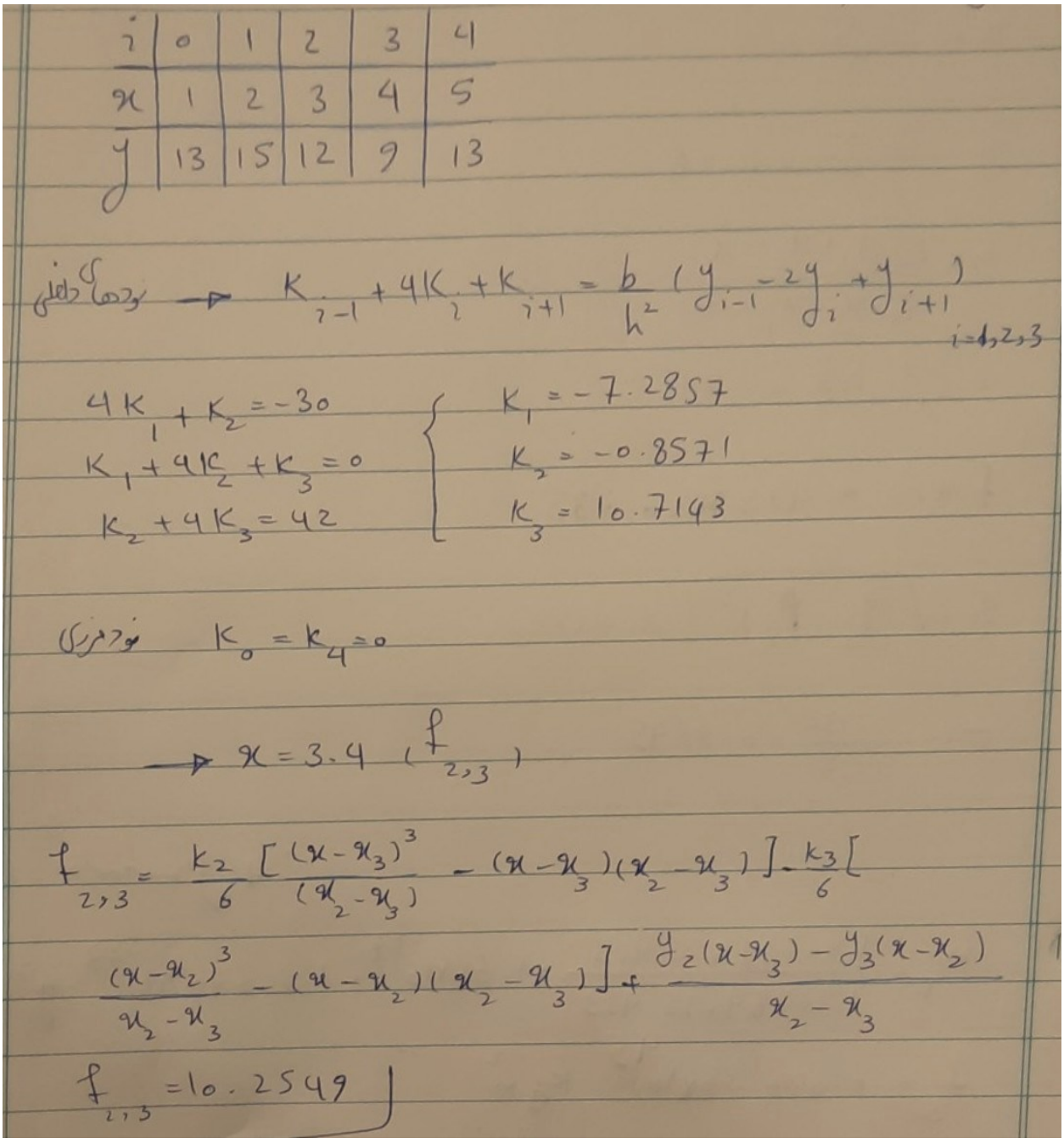
```
    coef[:,0] = y

    for j in range(1,n):
        for i in range(n-j):
            coef[i][j] = \
           (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])

    return coef

def newton_poly(coef, x_data, x):
    '''
    evaluate the newton polynomial
    at x
    '''
    n = len(x_data) - 1
    p = coef[n]
    for k in range(1,n+1):
        p = coef[n-k] + (x -x_data[n-k])*p
    return p
x = np.array([-250, -200, -100,0,100, 300])
y = np.array([0.0163,0.318,0.699,0.870,0.941,1.04])
# get the divided difference coef
a_s = divided_diff(x, y)[0, :]

# evaluate on new data points
x_new = np.arange(-275,310, 25)
y_new = newton_poly(a_s, x, x_new)

plt.figure(figsize = (12, 8))
plt.plot(x, y, 'bo',label='data')
plt.plot(x_new, y_new,'yellow',label='interpolate')

plt.legend(loc='lower right', ncol=2)
```



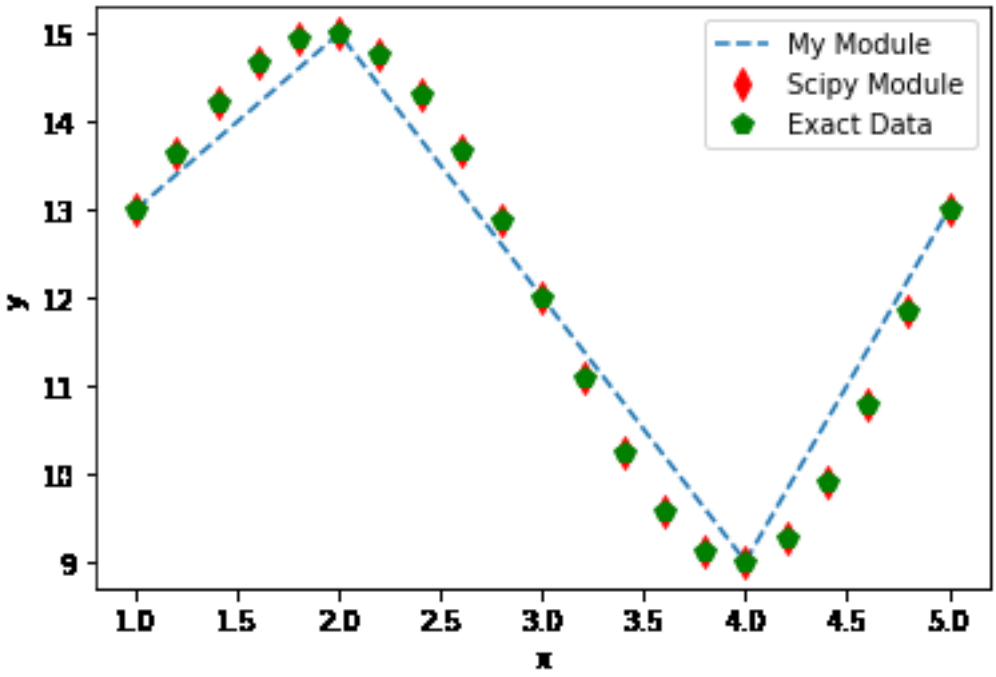
Question2:

Problem 2: For the given data points determine the natural cubic spline interpolant at $x = 3.4$. Compare your result with `scipy.interpolate.CubicSpline`.

x	1	2	3	4	5
y	13	15	12	9	13



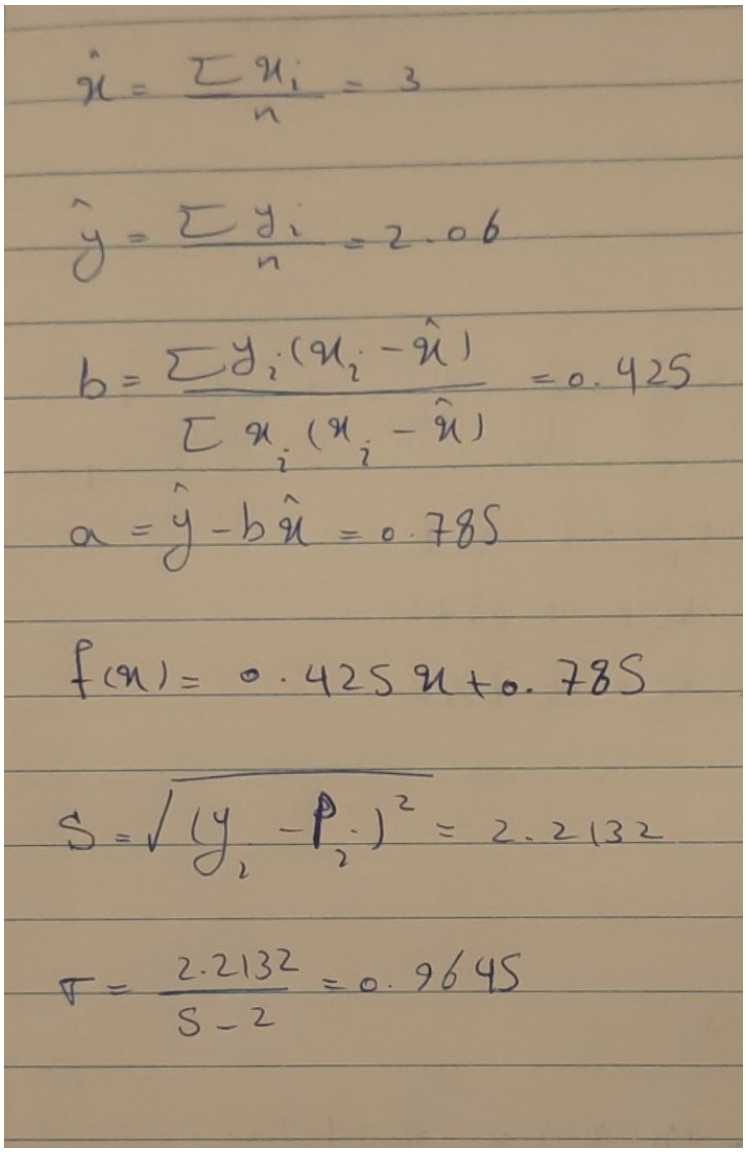
```
import numpy as np
from scipy.linalg import solve_banded
from scipy.interpolate import CubicSpline
import matplotlib.pyplot as plt
def curvatures(xData,yData):
  n = len(xData) - 1
  a = np.zeros((3,n+1)); a[1] = 14
  k = np.zeros(n+1)
  a[0,2:n+1] = xData[1:n] - xData[2:n+1]
  a[1,1:n] = 2.0*(xData[0:n-1] - xData[2:n+1])
  a[2,0:n-1] = xData[0:n-1] - xData[1:n]
  k[1:n] = 6.0*(yData[0:n-1] - yData[1:n])/(xData[0:n-1] - xData[1:n]) \
         -6.0*(yData[1:n] - yData[2:n+1]) \
         /(xData[1:n] - xData[2:n+1])
  k = solve_banded((1,1),a,k)
  return k
def evalspline(xData,yData,k,x):
  def findSegment(xData,x):
    iLeft = 0
    iRight = len(xData)- 1
    while 1:
      if (iRight-iLeft) <= 1: return iLeft
      i = int((iLeft + iRight)/2)
      if x < xData[i]: iRight = i
      else: iLeft = i
  i = findSegment(xData,x)
  h = xData[i] - xData[i+1]
  y = ((x - xData[i+1])**3/h - (x - xData[i+1])*h)*k[i]/6.0 \
    - ((x - xData[i])**3/h - (x - xData[i])*h)*k[i+1]/6.0 \
    + (yData[i]*(x - xData[i+1]) - yData[i+1]*(x - xData[i]))/h
  return y
xData = np.array([1,2,3,4,5])
yData = np.array([13,15,12,9,13])
curve = curvatures(xData,yData)
x = 3.4
y_interpolate1 = evalspline(xData, yData, curve, x)
cs = CubicSpline(xData, yData, bc_type='natural')
y_interpolate2 = cs(x)
x_loop = np.arange(1,5+0.2,0.2)
m = np.size(x_loop)
y_loop1 = np.zeros(m)
y_loop2 = np.zeros(m)
for i in range (m):
  y_loop1[i] = evalspline(xData, yData, curve, x_loop[i])
  y_loop2[i] = cs(x_loop[i])
plt.plot(xData,yData,'--',x_loop,y_loop1,'dr',x_loop,y_loop2,'pg',markersize=8)
plt.legend(('My Module','Scipy Module','Exact Data'),loc=0)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```


My Module
Scipy Module
Exact Data
15
14
13
12
11
10
9
y
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
x

Question 3:

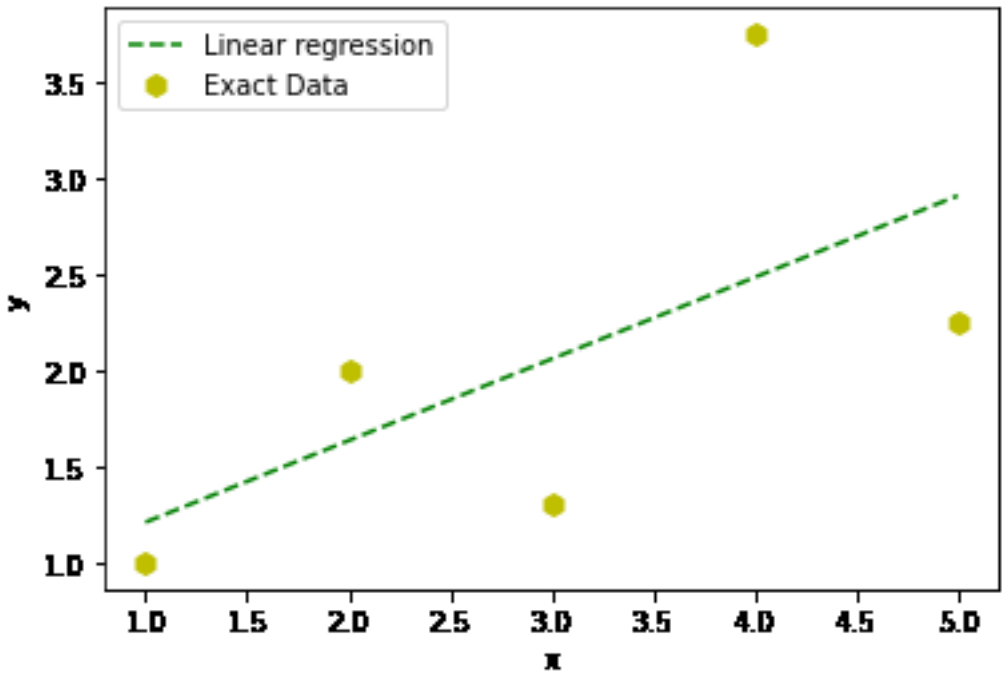
Problem 3: Use linear regression to fit the data and calculate corresponding standard deviation.

x	1.00	2.00	3.00	4.00	5.00
y	1.00	2.00	1.30	3.75	2.25




```
import numpy as np
from scipy.linalg import solve
import matplotlib.pyplot as plt
def polyFit(xData,yData,m):
  a = np.zeros((m+1,m+1))
  b = np.zeros(m+1)
  s = np.zeros(2*m+1)
```

```
  for i in range(len(xData)):
    temp = yData[i]
    for j in range(m+1):
      b[j] = b[j] + temp
      temp = temp*xData[i]
    temp = 1.0
    for j in range(2*m+1):
      s[j] = s[j] + temp
      temp = temp*xData[i]
    for i in range(m+1):
      for j in range(m+1):
        a[i,j] = s[i+j]
  return solve(a,b)
def stdDev(c,xData,yData):
# c is the polyFit coefficients
  def evalPoly(c,x):
    m = len(c) - 1
    p = c[m]
    for j in range(m):
      p = p*x + c[m-j-1]
    return p
  n = len(xData) - 1
  m = len(c) - 1
  sigma = 0.0
  for i in range(n+1):
    p = evalPoly(c,xData[i])
    sigma=sigma+(yData[i]-p)**2
  sigma=np.sqrt(sigma/(n-m))
  return sigma
xData = np.array([1,2,3,4,5])
yData = np.array([1,2,1.3,3.75,2.25])
coeffs = polyFit(xData,yData,1)
dev = stdDev(coeffs, xData, yData)
yReg = np.zeros(np.size(yData));
yReg = coeffs[0]+coeffs[1]*xData
plt.plot(xData,yReg,'--g',xData,yData,'hy',markersize=8)
plt.legend(('Linear regression','Exact Data'),loc=0)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```


Linear regression
Exact Data
3.5
3.0
2.5
2.0
1.5
1.0
y
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
x

Question4:

Problem 4: The following sample data are driven from $f(x) = x^3 - 0.3x^2 - 8.56x + 8.448$ as follows.

x	-2.0	-1.0	0.0	1.0	2.0
$f(x)$	16.368	15.708	8.448	0.588	-1.872

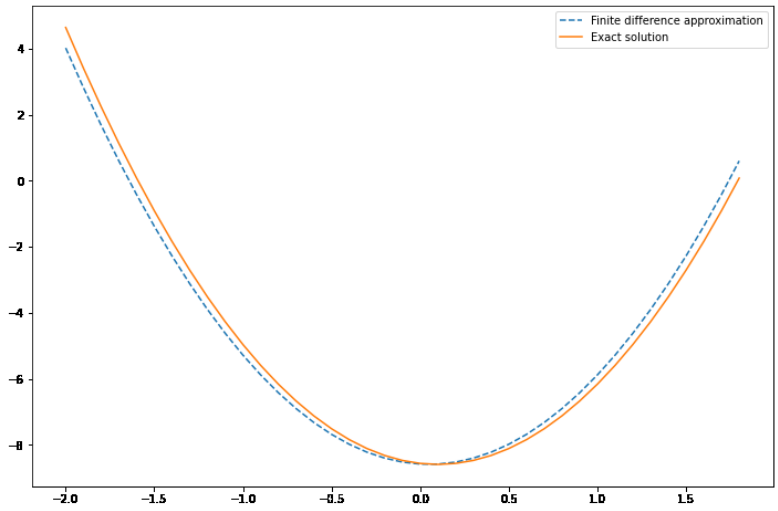
(a) Use second-order central difference to calculate $f'(0)$ and $f''(0)$. Afterward, implement the Richardson's extrapolation to improve the accuracy order.

(b) Use the cubic-spline interpolation method to compute f' and f'' at $x = 0$.

(c) Assess the accuracy of results in (a) and (b) with the exact one from differentiating the analytical profile.

Part a:

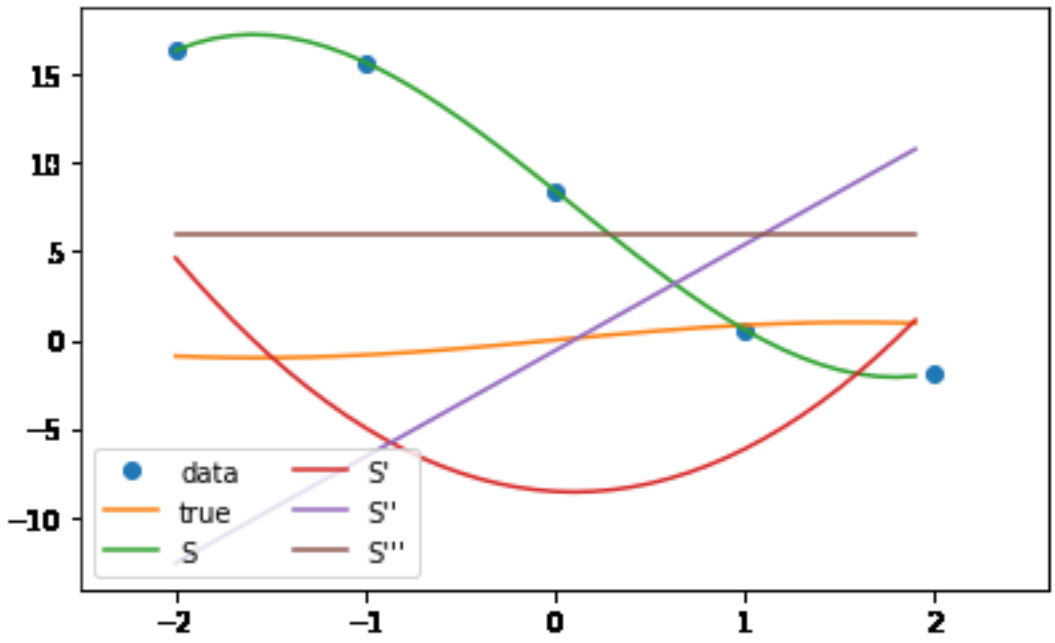
```
# step size
h = 0.1
# define grid
x = np.arange(-2,2, h)
# compute function
y = x**3-(0.3)*(x**2)-8.56*x+8.48

# compute vector differences
forward_diff = np.diff(y)/h
# compute corresponding grid
x_diff = x[:-1:]
# compute exact solution
exact_solution = 3*x_diff**2 - 0.6*x_diff - 8.56

# Plot solution
plt.figure(figsize = (12, 8))
plt.plot(x_diff, forward_diff, '--', \
         label = 'Finite difference approximation')
plt.plot(x_diff, exact_solution, \
         label = 'Exact solution')
plt.legend()
plt.show()

# Compute max error between
# numerical derivative and exact solution
max_error = max(abs(exact_solution - forward_diff))
print(max_error)
```



Part b:

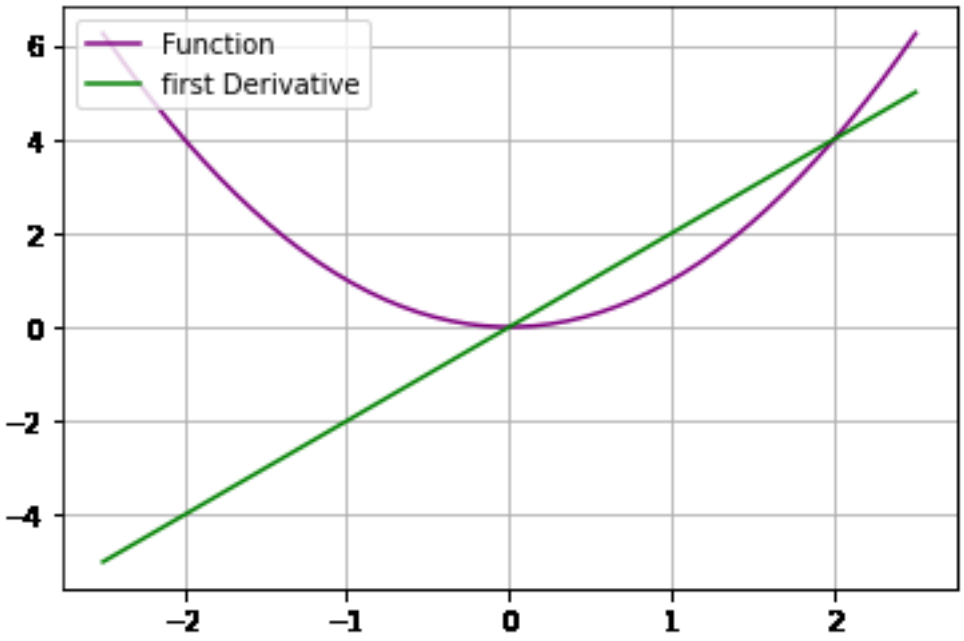
```
from scipy.interpolate import CubicSpline
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-2,3,1)
y = x**3-0.3*x**2-8.56*x+8.448
cs = CubicSpline(x, y)
xs = np.arange(-2, 2, 0.1)
fig, ax = plt.subplots(figsize=(6.5, 4))
ax.plot(x, y, 'o', label='data')
ax.plot(xs, np.sin(xs), label='true')
ax.plot(xs, cs(xs), label="S")
ax.plot(xs, cs(xs, 1), label="S'")
ax.plot(xs, cs(xs, 2), label="S''")
ax.plot(xs, cs(xs, 3), label="S'''")
ax.set_xlim(-2.5,2.6)
ax.legend(loc='lower left', ncol=2)
plt.show()
```

first derivative in x=0: -8.56
second derivative in x=0: -0.6



Part c:

```
# importing the library
import matplotlib.pyplot as plt
from scipy.misc import derivative
import numpy as np

# defining the function
def function(x):
  return x**2#x**3-0.3*x**2-8.56*x+8.48

# calculating its derivative
def deriv(x):
  return derivative(function, x)


# defininf x-axis intervals
y = np.linspace(-2.5,2.5)

# plotting the function
plt.plot(y, function(y), color='purple', label='Function')

# plotting its derivative
plt.plot(y, deriv(y), color='green', label='first Derivative')


# formatting
plt.legend(loc='upper left')
plt.grid(True)
```

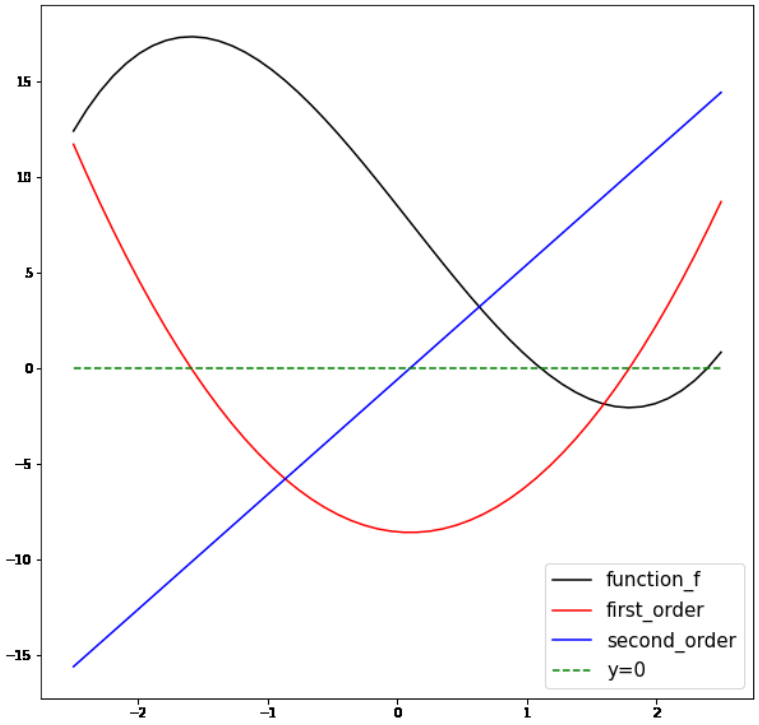



```
from sympy import *
x = Symbol('x')
y = x**3-(0.3)*(x**2)-8.56*x+8.48
## the first derivative
yfirst = y.diff(x)
print('First derivative is:',yfirst)

## the second derivative
ysecond = yfirst.diff(x)
print('Second derivative is:',ysecond)
## the second derivative
ysecond = yfirst.diff(x)
```

```
First derivative is: 3*x**2 - 0.6*x - 8.56
                     Second derivative is: 6*x - 0.6
```

```
test_x = np.linspace(-2.5,2.5)
## corresponding y, first_derivative, second_derivative
test_y = [y.subs({x:v}) for v in test_x]
test_y_f = [yfirst.subs({x:v}) for v in test_x]
test_y_s = [ysecond.subs({x:v}) for v in test_x]
fig, ax = plt.subplots(figsize=(10, 10))
ax.plot(test_x, test_y, color='black', label='function_f')
ax.plot(test_x, test_y_f, color='red', label='first_order')
ax.plot(test_x, test_y_s, color='blue', label='second_order')
ax.plot(test_x, np.zeros(len(test_x)), 'g--', label='y=0')
```

```
ax.legend(fontsize=15)
plt.show()
```



True answer for first derivative in x=0: -8.56
True answer for second derivative in x=0: -0.6

Question5:

Problem 5: Use Gauss-Chebyshev quadrature with six nodes to evaluate

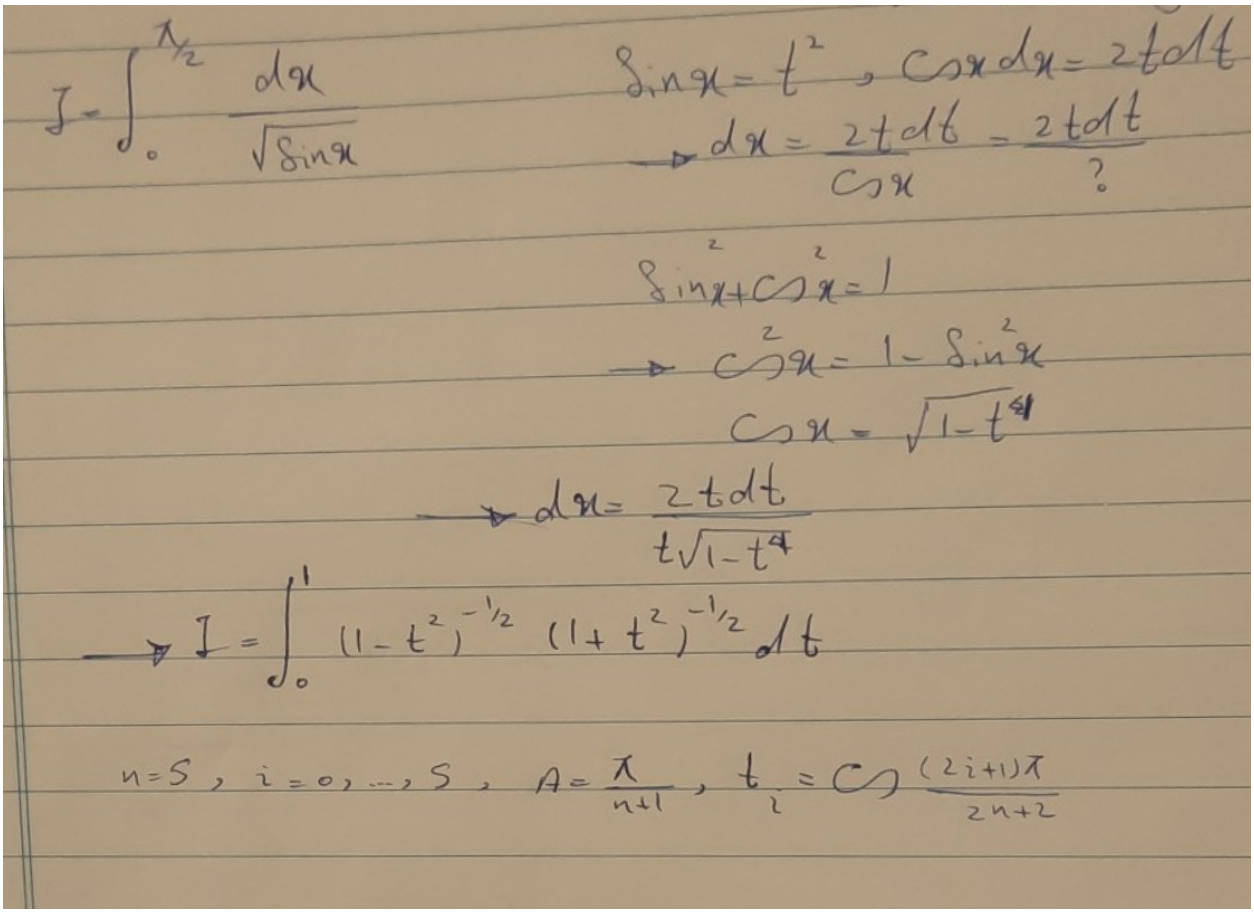
$$I = \int_0^{\pi/2} \frac{dx}{\sqrt{\sin(x)}}$$

Compare the result with the exact value 2.62206. Hint: Substitute $\sin(x) = t^2$.

```
import numpy as np
n = 5
weight = np.pi/(n+1)
sumation = 0
for i in range (n+1):
  t = np.cos((2*i+1)*np.pi/(2*n+2))
  sumation = sumation + weight/np.sqrt(1+(t)**2)
```

```
sumation
```

```
2.6220271839591267
```

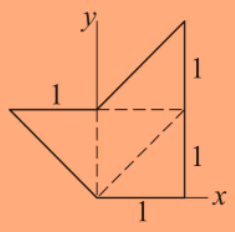


Question6:

Problem 6: Write a program to evaluate $\iint f(x,y)dxdy$ over an irregular region with triangular elements.

(a) Present the discretized domain in a data file as:

```
number_of_cells
n11 n12 n13
n21 n22 n23
...
xp1 yp1
xp2 yp2
xp3 yp3
...
```



The discrete domain.

number_of_cells" is the number of total cells. The list "ni1, ni2, ni3" contains the node numbers constructing the cell i. This list his called the *mesh connectivity*. The list "xpi ypi" are the i-th node coordinates, e.g, if for the first cell, (n11,n12,n13)=(1,2,3), then the cell coordinates are [(xp1,yp1),(xp2,yp2),(xp3,yp3)].

(b) Write a function to read the data file and save the coordinates for each cell.

(c) Write a function to perform a piece of integration $I_k = \iint_{\Omega_k} f(x,y)dxdy$ for each cell Ω_k, sum up the overall integration like $I_{\text{total}} = I_{\text{total}} + I_k$, and return the result. The quadrature points are needed to be tabulated in arrays corresponding to the given numerical accuracy. Use Table 6.7 by Kiusalaas (2013).

(d) Test your code for the domain shown with $f(x,y) = xy(y-x)$.

```
import numpy as np
def triangleQuad(f,xc,yc):
  alpha = np.array([[1.0/3.0, 1.0/3.0, 1.0/3.0], \
                    [0.2, 0.2, 0.6], \
                    [0.6, 0.2, 0.2], \
                    [0.2, 0.6, 0.2]])
  W = np.array([-27.0/48.0,25.0/48.0,25.0/48.0,25.0/48.0])
  x = np.dot(alpha,xc)
  y = np.dot(alpha,yc)
  A = (xc[1]*yc[2] - xc[2]*yc[1] \
       - xc[0]*yc[2] + xc[2]*yc[0] \
       + xc[0]*yc[1] - xc[1]*yc[0])/2.0
  sum = 0.0
  for i in range(4): sum = sum + W[i] * f(x[i],y[i])
  return A*sum
def f(x,y): return x*y*(y - x)

mesh_connectivity = open('mesh_connectivity.txt','r')
i = 0 # looping parameter for reading the .txt file
NodeID = []
x = []
y = []
for line in mesh_connectivity:
  if i == 0: n = eval(line.split()[0]) # reading the number of elements
  elif i>0 and i<=n: # reading the Node IDs
    NodeID.append(int(float(line.split()[0])))
    NodeID.append(int(float(line.split()[1])))
    NodeID.append(int(float(line.split()[2])))
```

```
  else: # reading the node coordinates
    x.append(float(line.split()[0]))
    y.append(float(line.split()[1]))
  i = i +1 # moving forward to the next line
integral = 0.0
for j in range(n): # loop over nodes of each element
  xc = np.array(x[3*j:2+3*j+1])
  yc = np.array(y[3*j:2+3*j+1])
  integral = integral + triangleQuad(f,xc,yc)
```

output:

0.13333333333325

File:

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