math104ahomework4

February 23, 2021

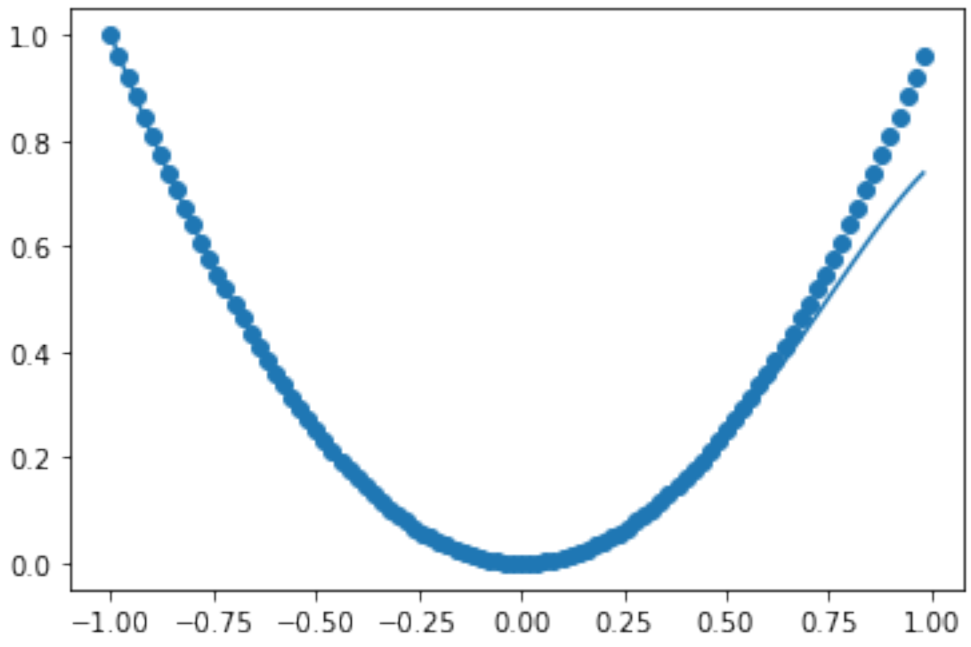
0.0.1 Homework 4

Question 1

[201]:
```
import matplotlib.pyplot as plt
import numpy as np
#yubowei 02/23/2021
# this function is for calculating divided difference table
# input x, y notes and n
# output C_n
def dividedifference(x, y, n):
    for i in range(1, n):
        for j in range(n - i):
            y[j][i] = ((y[j][i - 1] - y[j + 1][i - 1]) / (x[j] - x[i + j]))
    return y
# this function calculates (x-x_0)*(x-x_1)*......
# input: i, value and x notes
# output: the product terms of the function
def xproduct(i, value, x):
    product = 1
    for j in range(i):
        product = product * (value - x[j])
    return product

# this function is for applying Newton's divided difference formula
# input: the value, x, y, and n
# output estimated value for each x_i
def Newton(value, x, y, n):
    sum = y[0][0]
    for i in range(1, n):
        sum = sum + (xproduct(i, value, x) * y[0][i])
    return sum
```

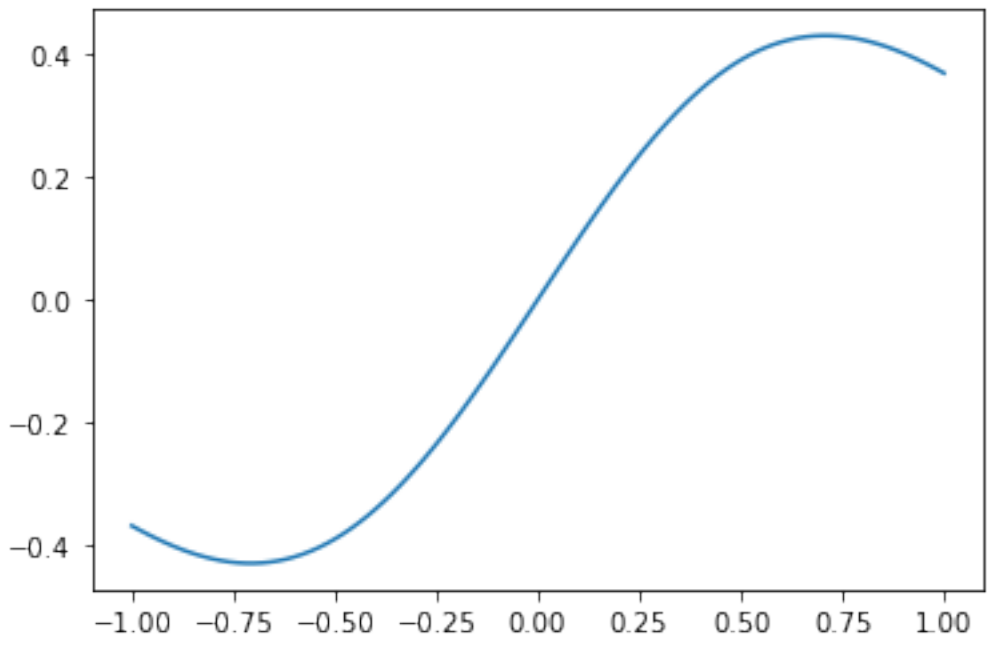
[202]:
```
# test code using x^2 for x in [-1, 1], h = 100
i = np.arange(100)
x = -1 + i*2/100
y = np.zeros((100,100))
y_dot = [0 for i in range(100)]
```

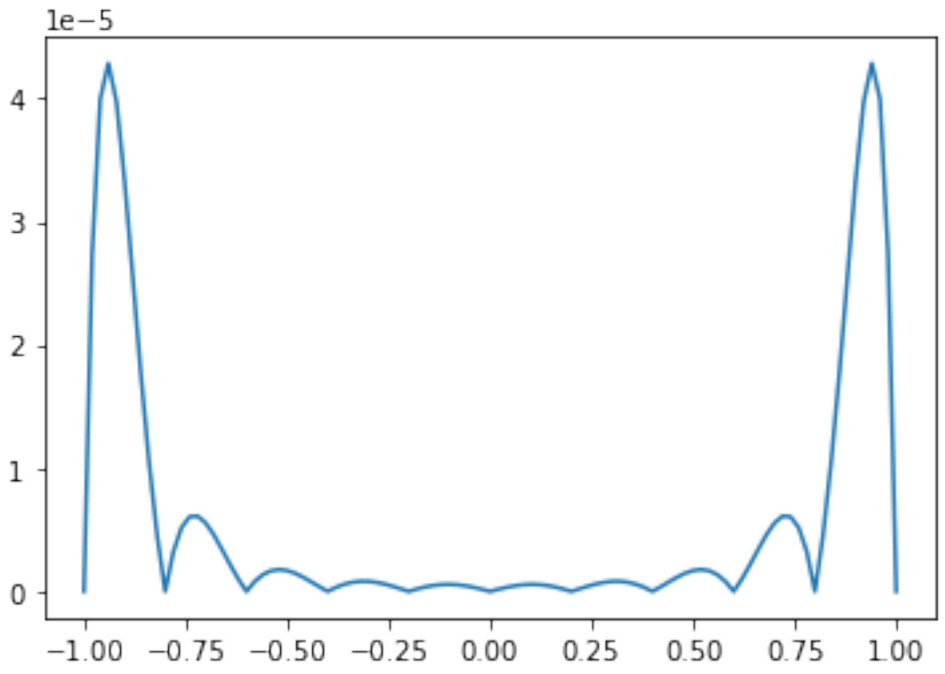
```
for i in range(100):
  y[i][0]= x[i]*x[i]
for i in range(100):
    y_dot[i] = x[i]*x[i]

divided = dividedifference(x, y, n)

y_est=[]
for i in range(100):
  y_est.append(Newton(x_n[i], x, divided, n))
plt.scatter(x,y_dot)
plt.plot(x,y_est);
plt.show();

# looks Okey
```



[203]:
```
# we want to evaluate P_10
n = 11

i = np.arange(11)
x = -1 + i*2/10

i = np.arange(101)
x_n = -1 + i*2/100

y = [[0 for i in range(101)]
```

```
        for j in range(101)]

# here we get exact value of x, then simulate the P_10
for i in range(11):
  y[i][0]= np.exp(-x[i]*x[i]) * x[i]

# calculating divided difference
divided = dividedifference(x, y, n)

y_est=[]
for i in range(101):
  y_est.append(Newton(x_n[i], x, divided, n))

y_real = np.exp(-x_n*x_n)* x_n
err = abs(y_est-y_real)
plt.plot(x_n, y_est)
plt.show()
plt.plot(x_n,err)
plt.show()
```



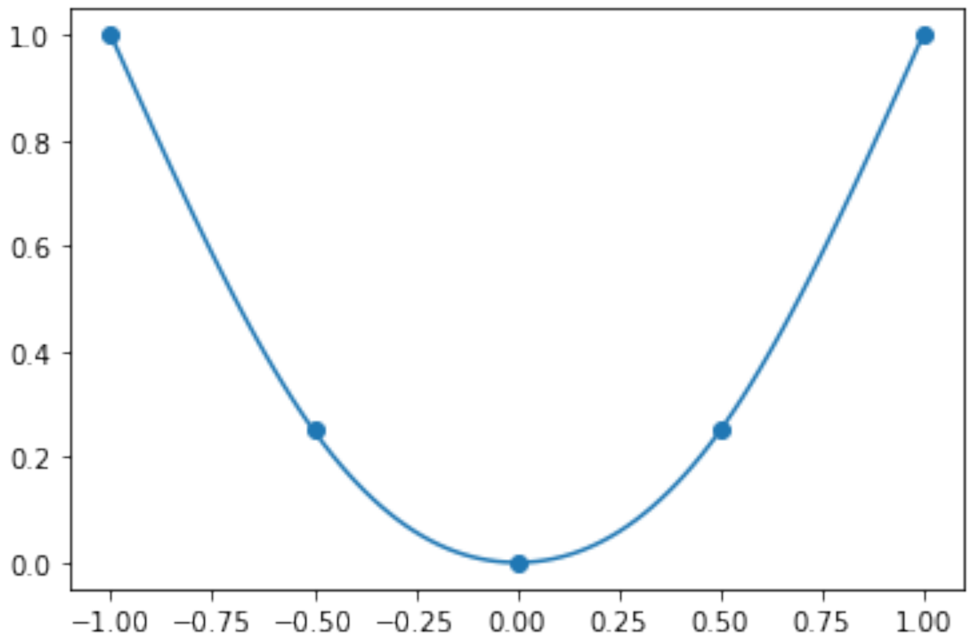


Question 4 4. Write a code to compute a natural spline $S(x)$ which interpolates a collection of given points $(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)$ where $x_0 < x_1 < \cdots < x_n$ (do not assume they are equidistributed). Your code should have a triadiagonal solver for the resulting linear system of equations (you're not allowed to use Matlab's operator to solve the linear system).

[204]:
```
# main reference: https://zhuanlan.zhihu.com/p/62860859
# and: https://www.cnblogs.com/flysun027/p/10371726.html
# yubowei 02/23/2021
#solving the matrix
#input upper diagonal and y vector
def upper(upper, y):
    x = np.zeros(len(y))
    # start from the bottom
    for i in range(len(y)-1, -1, -1):
        temp = [upper[i][j] * x[j] for j in range(len(y)-1, i, -1)]
        x[i] = (y[i] - sum(temp))/upper[i][i]
    return x

# similarly, we solve the lower diagonal
def lower(lower, y):
    z = np.zeros(len(y))
    for i in range(0, len(y)):
        temp = [lower[i][j] * z[j] for j in range(i)]
        z[i] = (y[i] - sum(temp))/lower[i][i]
    return z
```

```
# void function
# input tridiagonal matrxi and zero matrixes.
# output Lower diagonal and Upper diagonal
def void(m, l, u):
    #for eaiser computation, here I used Natural Spline graph from https://
 ↪zhuanlan.zhihu.com/p/62860859
    # basically it adds an edge to the matrix in the lecture.
    # so, let
    l[-1][-1] = 1
    u[0][0] = m[0][0]
    for i in range(len(m) - 1):
        l[i][i] = 1
        u[i][i+1] = m[i][i+1]
        l[i+1][i] = m[i+1][i] / u[i][i]
        u[i+1][i+1] = m[i+1][i+1] - l[i+1][i] * m[i][i+1]

#now solve the tridiagonal_martix. Ax = y
# input tridiagonal_martix and y vector
def solve_tridiagonal_martix(m, y):
    l = zeros((len(m),len(m)))
    u = zeros((len(m),len(m)))
    void(m,l,u)
    z = lower(l, y)
    x = upper(u, z)
    return x

# now compute the natural_spline.
# input x,y and times to do this
# output the estimated value
# Zn Z0 are set to be zero.
def natural_spline(x,y,linespace):
    h = [x[i] - x[i-1] for i in range(1,len(x))]
    tridiagonal_martix = np.zeros((len(x)-2,len(x)-2))
    temp = []
    for i in range(len(x) - 2):
        temp.append(-6/h[i]*(y[i+1]-y[i])+6/h[i+1]*(y[i+2]-y[i+1]))
    for i in range(len(tridiagonal_martix)):
        tridiagonal_martix[i][i] = 2*(h[i]+h[i+1])
        if (i > 0):
            tridiagonal_martix[i][i-1] = h[i]
            tridiagonal_martix[i-1][i] = h[i]
    z = solve_tridiagonal_martix(tridiagonal_martix, temp)
    z = np.concatenate(([0],z[0:],[0]))
    i = 0
    while(linespace >= x[i] and i < len(x) - 1):
        i = i + 1
    hj = (x[i]-x[i-1])
    a = 1/6/hj*(z[i]-z[i-1])
    b = 1/2*z[i-1]
    c = 1/hj*(y[i]-y[i-1])-1/6*hj*(z[i]+2*z[i-1])
    d = y[i-1]
    product = linespace - x[i-1]
    return a*product**3 + b*product**2 + c*product + d
```

[205]:
```
x = [-1,-0.5,0,0.5,1]
y = [1, 0.25, 0, 0.25, 1]
X_n = np.linspace(-1,1,100)
temp = []
for i in X_n:
    temp.append(natural_spline(x,y,i))

plt.plot(X_n,temp)
plt.scatter(x,y)
plt.show()

# Test for X^2. more accurate and looks nice
```



Question5 5. The given table is:

j	t_j	x_j	y_j
0	0	1.5	0.75
1	0.618	0.90	0.90
2	0.935	0.60	1.00
3	1.255	0.35	0.80
4	1.636	0.20	0.45
5	1.905	0.10	0.20
6	2.317	0.50	0.10
7	2.827	1.00	0.20
8	3.330	1.50	0.25

$$t_0 = 0, t_j = t_{j-1} + \sqrt{(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2}, j = 1, 2, \cdots, n$$

[206]:
```
#set up
x_j = [1.5,0.9,0.6,0.35,0.2,0.1,0.5,1,1.5]
y_j = [0.75,0.90,1,0.8,0.45,0.2,0.1,0.2,0.25]
t_j = np.zeros(9)
for i in range(1,len(x_j)):
    t_j[i] = t_j[i-1] + ((x_j[i]-x_j[i-1])**2+(y_j[i]-y_j[i-1])**2)**(1/2)
t_j
```

[206]:
```
array([0.        , 0.61846584, 0.93469361, 1.25484982, 1.63563848,
       1.90489672, 2.31720728, 2.82710923, 3.32960301])
```

[207]:
```
# this function is for calculating coefficients.
# input x,y,
def coefficients(x,y,linespace):
    h = [x[i] - x[i-1] for i in range(1,len(x))]
    tridiagonal_martix = np.zeros((len(x)-2,len(x)-2))
    temp = []
    for i in range(len(x) - 2):
        temp.append(-6/h[i]*(y[i+1]-y[i])+6/h[i+1]*(y[i+2]-y[i+1]))
    for i in range(len(tridiagonal_martix)):
        tridiagonal_martix[i][i] = 2*(h[i]+h[i+1])
        if (i > 0):
            tridiagonal_martix[i][i-1] = h[i]
            tridiagonal_martix[i-1][i] = h[i]
    z = solve_tridiagonal_martix(tridiagonal_martix, temp)
    z = np.concatenate(([0],z[0:],[0]))
    i = 0
    while(linespace >= x[i] and i < len(x) - 1):
        i = i + 1
    hj = (x[i]-x[i-1])
    a = 1/6/hj*(z[i]-z[i-1])
    b = 1/2*z[i-1]
    c = 1/hj*(y[i]-y[i-1])-1/6*hj*(z[i]+2*z[i-1])
    d = y[i-1]
    return [a,b,c,d]
```

Use the values in Table 1 to construct a smooth parametric representation of a curve passing through the points $(x_j, y_j), j = 0, 1, \cdots, 8$ by finding the two natural cubic splines interpolating and $(t_j, y_j), j = 0, 1, \cdots, 8$, respectively. Tabulate the coefficients of the splines and plot the resulting (parametric) curve.

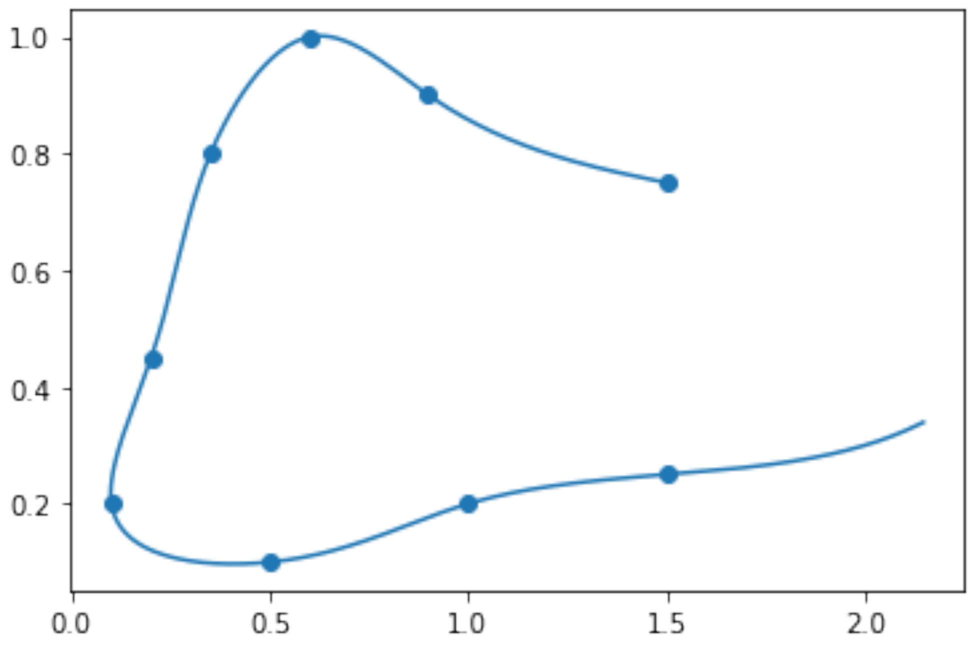
[188]:
```
xA = []
xB = []
xC = []
xD = []
for i in range(8):
    X = coefficients(t_j,x_j,t_j[i])
    xA.append(X[0])
    xB.append(X[1])
    xC.append(X[2])
    xD.append(X[3])
print(xA)
print(xB)
print(xC)
print(xD)
```

```
[0.007105106960089452, 0.11855028535840095, 0.9727629785901298,
-1.767684594909913, 5.377372990633091, -3.3838525297439754, 0.6697214331402968,
-0.14806380841414496]
[0.0, 0.013182797914992007, 0.1256494736136294, 1.0599578044368039,
-0.9593849151979237, 3.3843210524010177, -0.8012733680933655,
0.22320342878274446]
[-0.972860203557566, -0.9647070933208632, -0.9208044742443243,
-0.5412249393362486, -0.5029279240840239, 0.15000611319671553,
1.2150239570374712, 0.9202649669608458]
[1.5, 0.9, 0.6, 0.35, 0.2, 0.1, 0.5, 1]
```

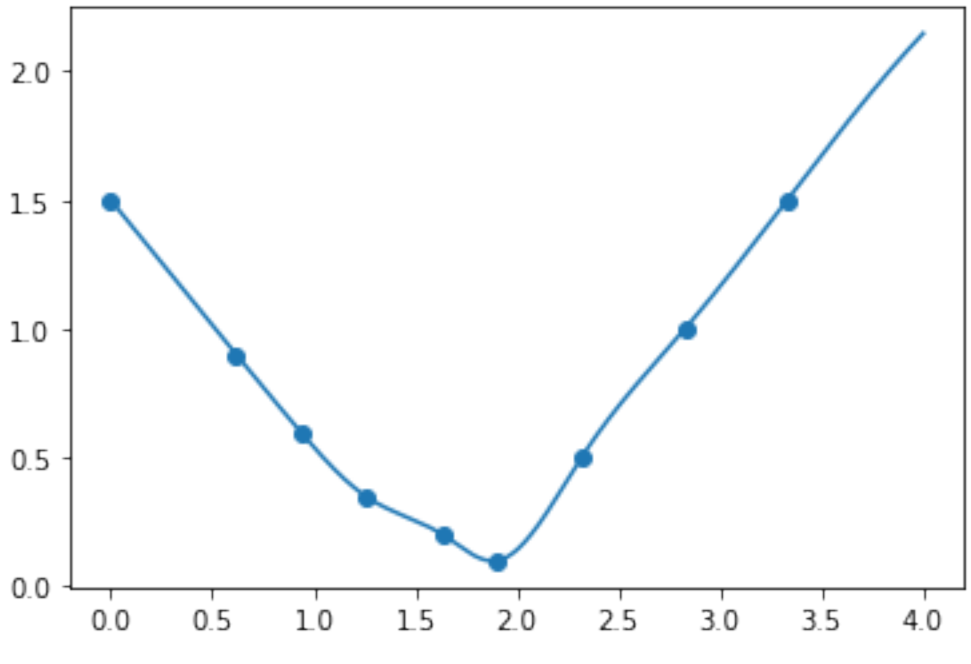
[208]:
```
tA = []
tB = []
tC = []
tD = []
for i in range(8):
    T = coefficients(t_j, y_j, t_j[i])
    tA.append(T[0])
    tB.append(T[1])
    tC.append(T[2])
    tD.append(T[3])
print(tA)
print(tB)
print(tC)
print(tD)
```

```
[0.27738724614358673, -3.0126002652020176, 2.4300600460055626,
-0.2677210946173959, 2.1617914582250504, -0.780564430815353,
-0.4745629644981175, 0.17272856224355387]
[0.0, 0.5146636117721459, -2.3433399435275577, -0.00934348667824844,
-0.31517895351718994, 1.4310615386624386, 0.4655566599068893,
-0.2603850850144132]
[0.13643500338641107, 0.45473686833617677, -0.12354136282270323,
-0.8767675775705756, -1.000342041185063, -0.6998814598641395,
0.0821142565528637, 0.18673164295399364]
[0.75, 0.9, 1, 0.8, 0.45, 0.2, 0.1, 0.2]
```

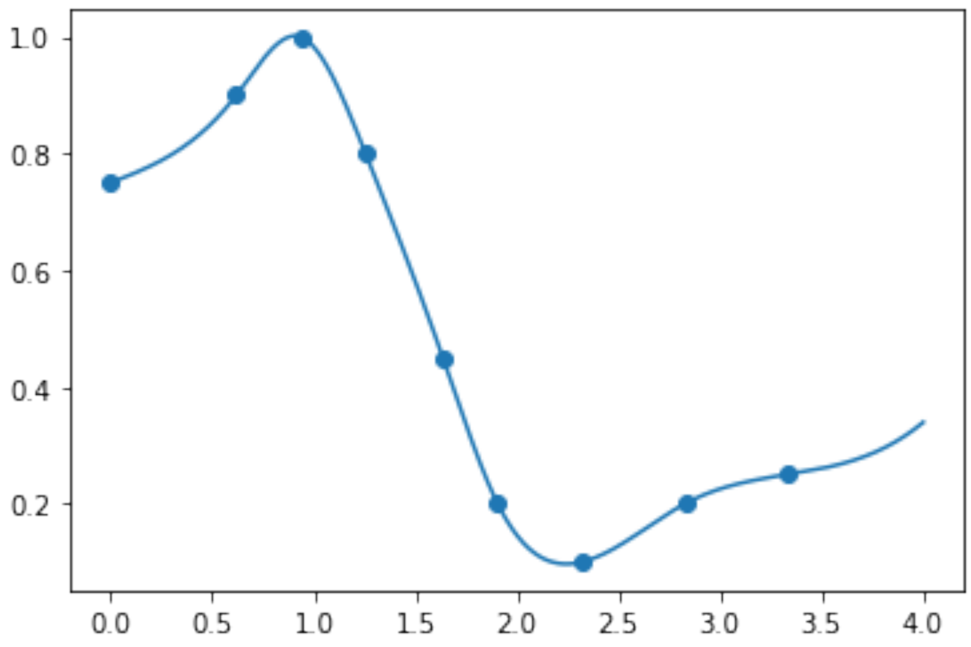
[209]:
```
X = []
Y = []
X_n = np.linspace(0, 4, 1000)

for i in X_n:
    X.append(natural_spline(t_j,x_j,i))
    Y.append(natural_spline(t_j,y_j,i))

plt.scatter(x_j,y_j)
plt.plot(X,Y)
plt.show()

plt.plot(X_n,X)
plt.scatter(t_j,x_j)
plt.show()

plt.plot(X_n,Y)
plt.scatter(t_j,y_j)
plt.show()
```



1.0
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
2.0



2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


1.0
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Math 104 A Numerical Analysis I **Name:** ________________________
Winter 2021
Homework 4
Due date: Feb 22th 9:50 am

You have to integrate all the problems that require coding and/or numerical computation in a single jupyter notebook. Make sure all your codes have a preamble which describes purpose of the code, all the input variables, the expected output, your name, and the date of the last time you modified it. Write your own code, individually. Do not copy codes! The solutions to the problems that do not require coding must be uploaded as a single pdf or as part of the jupyter notebook. It contains 3 pages (including this cover page) and 5 questions. Total of points is 100. Simplify all answers as far as possible. Solutions without proper justification will receive no credit.

It contains 3 pages (including this cover page) and 5 questions.
Total of points is 100. Simplify all answers as far as possible. Solutions without proper justification will receive no credit.

Grade Table (for TA use only)

Question	Points	Score
1	20	
2	10	
3	10	
4	30	
5	30	
Total:	100	

1. (20 points) In newton's form of the interpolation polynomial we need to compute the coefficients, $c_0 = f[x_0], c_1 = f[x_0, x_1], \cdots, c_n = f[x_0, x_1, \cdots, x_n]$. In the table of divided differences we proceed column by column and the needed coefficients are in the uppermost diagonal. A simple 1D array, c of size $n+1$, can be used to store and compute these values. We just have to compute them from bottom to top to avoid losing values we have already computed. The following pseudocode does precisely this:

```
for $j = 0, 1, \cdots, n$
   $c_j = f_j$;
end
for $k = 1, \cdots, n$
      for $j = n, n-1, \cdots, k$
          $c_j = (c_j - c_{j-1})/(x_j - x_{j-k})$;
      end
end
```

The evaluation of the interpolation polynomial in Newton's form can be done with the Horner-like scheme seen in class:

```
$p = c_n$
for $j = n-1, n-2, \cdots, 0$
      $p = c_j + (x - x_j) * p$;
end
```

- Write computer codes to compute the coefficients $c_0, c_1, \cdots, c_n$ and to evaluate the corresponding interpolation polynomial at an arbitrary point x. Test your codes and turn in a run of your test.
- Consider the function $f(x) = xe^{-x^2}$ for $x \in [-1, 1]$ and the nodes $x_j = -1 + j(2/10), j = 0, 1, \cdots, 10$. Use your codes in (a) to evaluate $p_{10}(x)$ at the points $\bar{x}_j = -1 + j(2/100)$, $j = 0, 1, \cdots, 100$ and plot the error $f(x) - p_{10}(x)$

2. (10 points) Obtain the Hermite interpolation polynomial corresponding to the data $f(0) = 0, f'(0) = 0, f(1) = 2, f'(1) = 3$.

3. (10 points) In class, we learned to use piecewise cubic splines that interpolate a function. Find a piecewise *linear* function that interpolates $(0, 2), (1, 2), (2, 1), (3, 9)$.

4. (30 points) Write a code to compute a natural spline $S(x)$ which interpolates a collection of given points $(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)$ where $x_0 < x_1 < \cdots < x_n$ (do not assume they are equidistributed). Your code should have a triadiagonal solver for the resulting linear system of equations (you're not allowed to use Matlab's operator to solve the linear system).

5. (30 points) One important application of spline interpolation is the construction of smooth curves that are not necessarily the graph of a function but that have a parametric representation $x = x(t)$ and $y = y(t)$ for $t \in [a, b]$. Hence one needs to determine two splines interpolating (t_j, x_j) and $(t_j, y_j), (j = 0, 1, \cdots, n)$. The arc length of the curve is a natural choice for the parameter t. However, this is not known a priori and instead the $t_j's$ are usually chosen as the distances of consecutive points:

$$t_0 = 0, t_j = t_{j-1} + \sqrt{(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2}, j = 1, 2, \cdots, n.$$

Use the values in Table 1 to construct a smooth parametric representation of a curve passing through the points (x_j, y_j), $j = 0, 1, \cdots, 8$ by finding the two natural cubic splines interpolating and (t_j, y_j),$j = 0, 1, \cdots, 8$, respectively. Tabulate the coefficients of the splines and plot the resulting (parametric) curve.

j	t_j	x_j	y_j
0	0	1.5	0.75
1	0.618	0.90	0.90
2	0.935	0.60	1.00
3	1.255	0.35	0.80
4	1.636	0.20	0.45
5	1.905	0.10	0.20
6	2.317	0.50	0.10
7	2.827	1.00	0.20
8	3.330	1.50	0.25

2. (10 points) Obtain the Hermite interpolation polynomial corresponding to the data $f(0) = 0, f'(0) = 0, f(1) = 2, f'(1) = 3$.

$f(0) = 0 \qquad f(1) = 2$

$f'(0) = 0 \qquad f'(1) = 3$

x_j	0th	1st	2nd	3rd
0	$f(0)=0$	$f[0,0]=f'(0)=0$	$f[0,0,1]=2$	$f[0,0,1,1]=-1$
0	$f(0)=0$	$f[0,1]=2$	$f[0,1,1]=1$	
1	$f(1)=2$	$f[1,1]=f'(1)=3$		
1	$f(1)=2$			

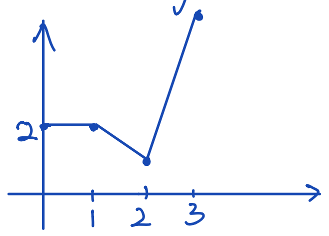
$$P_3(x) = f(0) + f[0,0]\cdot(x-0) + f[0,0,1]\cdot(x-0)(x-0) + f[0,0,1,1]\cdot(x-0)^2\cdot(x-1)$$

$$= 0 + 0 + 2x^2 + x^2\cdot(x-1)\cdot(-1)$$

$$= 3x^2 - x^3$$

3. (10 points) In class, we learned to use piecewise cubic splines that interpolate a function. Find a piecewise *linear* function that interpolates $(0, 2), (1, 2), (2, 1), (3, 9)$.

linear function



$$f(x) = \begin{cases} \frac{2-2}{1-0} \to 2 & x \in [0,1] \\ \frac{1-2}{2-1} \to -x+3 & x \in [1,2] \\ \frac{9-1}{3-2} \to 8x-15 & x \in [2,3] \end{cases}$$