math104ahw3

February 13, 2021

0.1 Question 1

(a) Write the Lagrangian form of the interpolating polynomial P_2 corresponding to the data in the table below

x_j	$f(x_j)$
-2	0
0	1
1	-1

$$\begin{aligned} P_2(x) &= I_0(x) f_0 + I_1(x) f_1 + I_2(x) f_2 &(1)\\ &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f_1 + \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f2 &(2)\\ &= 0 + \frac{(x+2)(x-1)}{(0+2)(0-1)} 1 + \frac{(x+2)(x-0)}{(1+2)(1-0)}(-1) &(3)\\ &= -\frac{(x+2)(x-1)}{2} - \frac{(x+2)x}{3} &(4) \end{aligned}$$

(b) Use P_2 to approximate $f_{(-1)}$

$$\begin{aligned} P_2(-1) &= -\frac{(-1+2)(-1-1)}{2} - \frac{-(-1+2)}{3} &(5)\\ &= 1 + \frac{1}{3} &(6)\\ &= \frac{4}{3} &(7) \end{aligned}$$

0.2 Question 2

We proved in class that

$$\|f - p_n\|_\infty \leq (1 + \Lambda_n) \|f - p_n^*\|_\infty$$

where p_n is the interpolating polynomial of f at nodes $x_0, \cdots, x_n, p_n^*$ is the best approximation of f in the infinity norm, by a polynomial of degree at most n, and Λ_n is the Lebesgue constant, i.e., $\Lambda_n = \|L_n\|_\infty$ where

$$L_n(x) = \sum_{j=0}^{n} |l_j(x)|$$

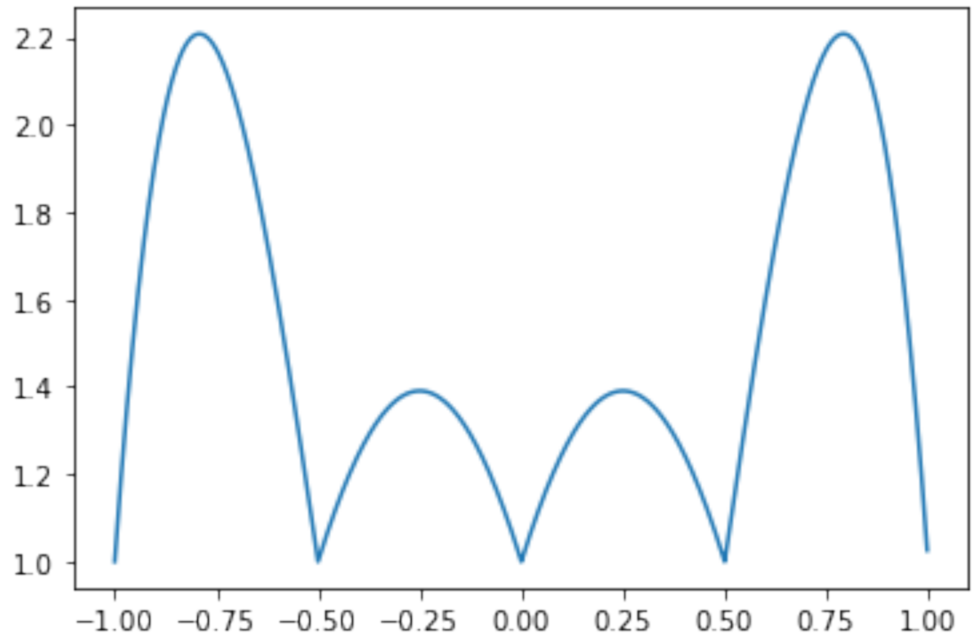
(a) Write a computer code to evaluate the Lebesgue function $\Lambda_n(x)$ associated to a given set of pairwise distinct nodes $x_0, \cdots, x_n$.

[17]:
```
# yubowei 6990006 02/13 9am

import math
import matplotlib.pyplot as plt
import numpy as np
# this code takes interpolating nodes,evaluating points as inputs.
# output Lebesgue Constant
def lag(x_k, x, k):
    temp = []
    for i in range(len(x_k)):
        if i != k:
            temp.append((x-x_k[i])/(x_k[k] - x_k[i]))
    return np.prod(temp)

def l_f(x, j):
    temp = []
    for i in range(len(j)):
        temp.append(abs(lag(j,x,i)))
    return np.sum(temp)
```

(b) Consider the equidistributed points $x_j = -1 + \frac{2j}{n}$ for $j = 0, 1 \cdots, n$. Write a computer code that uses (a) to evaluate and plot $L_n(x)$ (evaluate $L_n(x)$ at a large number of points $\bar{x}_k$ to have a good plotting resolution, e.g. $\bar{x}_k = -1 + k\frac{2}{n_e}$, $k = 0, 1, \cdots, n_e$ with $n_e = 1000$ for $n = 4, 10, 20$. Estimate Λ_n for these three values of n.

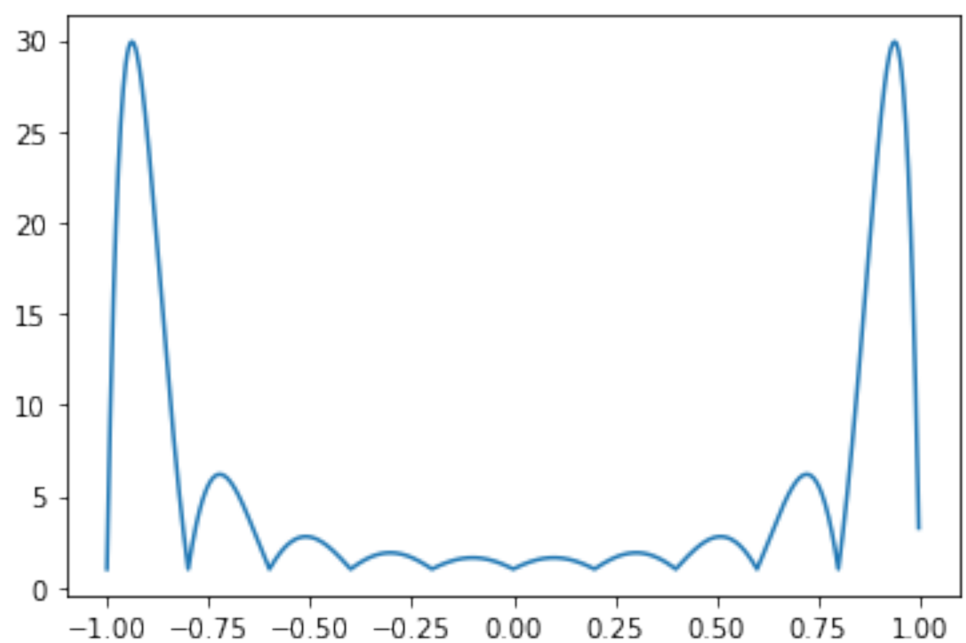
[18]:
```
n1 = 4
n2 = 10
n3 = 20
n = 1000
p1 = [ -1 + j*2/n1 for j in range(n1 + 1) ]
p2 = [ -1 + j*2/n2 for j in range(n2 + 1) ]
p3 = [ -1 + j*2/n3 for j in range(n3 + 1) ]

x_k = np.arange(-1, 1, 2/n)

L1 = [l_f(i, p1) for i in x_k]
L2 = [l_f(i, p2) for i in x_k]
L3 = [l_f(i, p3) for i in x_k]

plt.plot(x_k, L1)
max(L1)
```

[18]: 2.207824277504



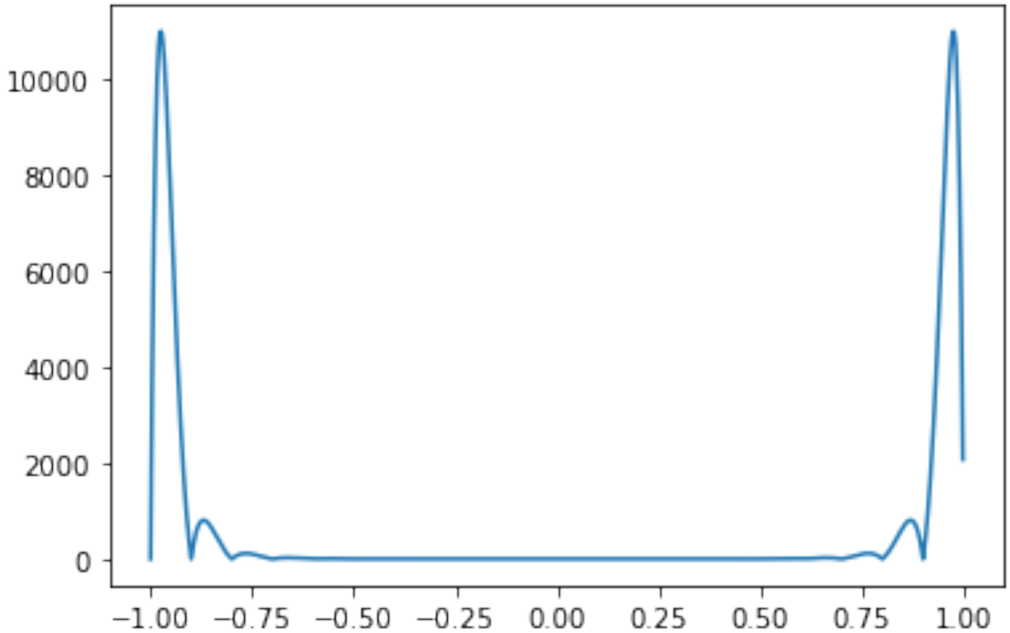
[19]:
```
plt.plot(x_k, L2)
max(L2)
```

[19]: 29.898141093562188



[16]:
```
plt.plot(x_k, L3)
max(L3)
```

[16]: 10979.243923985889

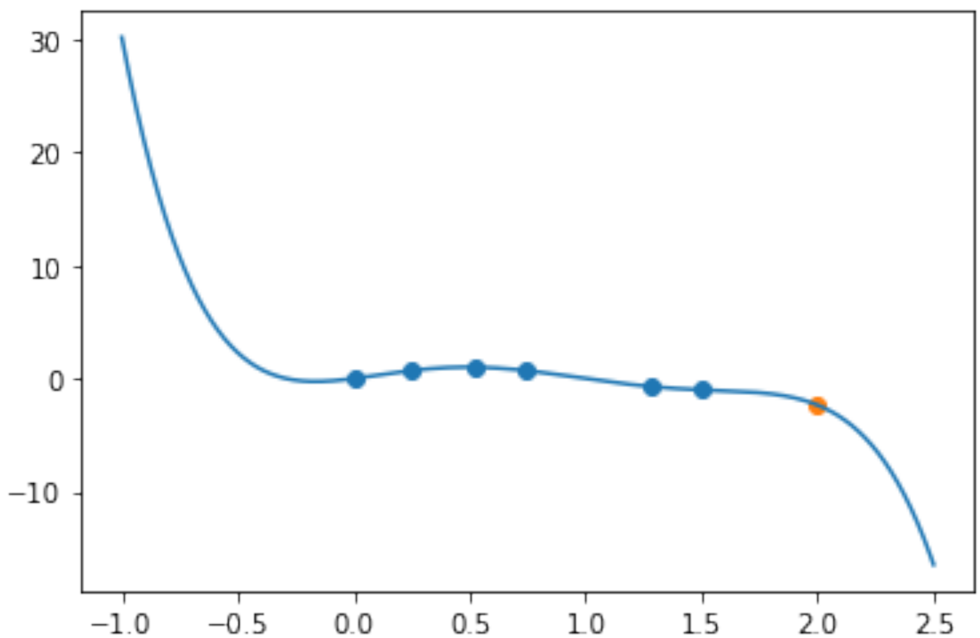


0.3 Question 3

(a) Implement the Barycentric Formula for evaluating the interpolating polynomial for arbitrary distributed nodes $x_0, \cdots, x_n$, you need to write a function or script that computes the barycentric weights λ_j, for $j = 0, 1 \cdots, n$, first and another code to use these values in the Barycentric Formula. Make sure to test your implementation.

[44]:
```
# this function takes evaluating points, existing x,y notes as input
# outputs P_n
def bw(node):
    mu = []
    for i in range(len(node)):
        temp = []
        for j in range(len(node)):
            if j != i:
                temp.append(node[j]-node[i])
        mu.append(1/np.prod(temp))
    return mu

def bari(z,x,y):
    L_1 = [bw(x)[i]*y[i]/(z-x[i])
```

```
            for i in range(len(x))]
    L_2 = [bw(x)[i]/(z-x[i])
            for i in range(len(x))]
    return sum(L_1)/sum(L_2)

x_t = np.linspace(-1, 2.5, 1000)
x = np.array([0.00, 0.25, 0.52, 0.74, 1.28, 1.50])
y = np.array([0.0000, 0.7070, 1.0000, 0.7071, -0.7074, -1.0000])


plt.scatter(x,y)
plt.scatter(2,bari(2,x,y))
plt.plot(x_t,bari(x_t,x,y))
plt.show()
```



(b) Consider the following table of data, use your code in (a) to find $P_5(2)$ as an approximation of $f(2)$

x_j	$f(x_j)$
0.00	0.0000
0.25	0.7070
0.52	1.0000
0.74	0.7071
1.28	−0.7074
1.5	−1.0000

[40]:
```
bari(2,x,y)
```

[40]: -2.3438296081728365

4. (20 points) • Equating the leading coefficient of the Lagrange form of the interpolating polynomial $p_n(x)$ with that of the Newton's form to deduce that

$$f[x_0, x_1, \cdots, x_n] = \sum_{j=0}^{n} \frac{f(x_j)}{\prod_{k=0,k\neq j}^{k=n}(x_j - x_k)} \tag{1}$$

• use (1) to conclude that the divided difference are symmetric functions of their arguments, i.e, any permutation of $x_0, \cdots, x_n$ leaves the corresponding divided difference unchanged.

Given polynomial:

$$W_n(x) = (x-x_0)(x-x_1)\cdots(x-x_{n-1})$$

$$P_n(x) = P_{n-1}(x) + C_n W_n(x)$$

let $x = x_i$ for some $i \in \{0, \cdots, n-1\}$ $W_n(x_i) = 0$.

$$I_n(x) = P_{n_1}(x_i) = f(x_i)$$

then, $f(x_n) = P_{n-1}(x_n) + C_n W_n(x_n)$

$$C_n = \frac{f(x_n) - P_{n-1}(x_n)}{W_n(x_n)}$$

$$P_0(x) = f(x_0), \quad P_n(x) = \sum_{k=0}^{n} C_k W_k(x)$$

then the coefficient of lagrange $[x_0 x_1 \cdots x_k] f = \dfrac{[x_1 \cdots x_k] f - [x_1 \cdots x_{k-1}] f}{x_k - x_0}$

thus, $P_{n+1}(x) = P_n(x) + f(x_0, x_1 \cdots x_{n+1}) W_{n+1}(x)$

$$W_{n+1}(x) = (x - x_n) W_n(x)$$

$$P_n(x) = \sum_{j=0}^{n} \frac{x - x_k}{\prod_{k=0, k\neq j}^{n} (x_j - x_k)} f(x_j)$$

then we can extract the coefficient:

$$f[x_0, x_1 \cdots\cdots x_n] = \sum_{j=0}^{n} \frac{f(x_j)}{\prod_{k=0, k\neq j}^{k} (x_j - x_k)}$$

P_k is independent of the points, $f[x_0 \cdots\cdots x_n]$ is the divided difference that is symmetric function of its argument