科学计算项目作业一

罗俊勋

学号:3210101613

2023 年 4 月 23 日

问题

设你学号的最后一位是 z, 取 $n = \max(2z, 10)$.

设函数 $f(x) = \frac{1}{1+25x^2}, x \in [-1, 1]$. 利用下列条件做插值逼近, 并与函数 $f(x)$ 的图像进行比较.

(a): 用等距节点 $x_i = -1 + \frac{2i}{n}, i = 1, 2, \cdots, n$. 试建立 n 次 $Lagrange$ 插值多项式和 $Newton$ 插值多项式, 绘出插值多项式的图像

(b): 用节点 $x_i = \cos\frac{2i+1}{42}\pi, i = 0, 1, 2, \cdots, 20$, 绘出 20 次 $Lagrange$ 插值多项式的图像;

(c): 用等距节点 $x_i = -1 + \frac{2}{10}i, i = 0, 1, 2, \cdots, 10$, 绘出它的分段线性插值函数

(d): 用等距节点 $x_i = -1 + \frac{2}{10}i, i = 0, 1, 2, \cdots, 10$, 绘出它的分段三次 $Hermite$ 插值函数的图像

公式和算法

Lagrange 插值原理:

$f(x)$ 是需要插值的函数;$(x_0,y_0),(x_1,y_1),\cdots,(x_n,y_n)$ 是 $n+1$ 个插值节点;

(计算方法): 令 $\phi(x)=y_0l_0(x)+y_1l_1(x)+\cdots+y_nl_n(x)$

若 n 次多项式 $l_j(x);(j=0,1,2\cdots,n)$ 在 n+1 个节点 $x_0<x_1<\ldots<x_n$ 上满足:

$$l_k\left(x_j\right)=\begin{cases}1, & j=k\\ 0, & j\neq k\end{cases},(j,k=0,1,\ldots,n)$$ 则称这 n+1 个 n 次多项式为节点 $x_0<x_1<\ldots<x_n$ 上的 n 次插值基函数.

$$\text{lagrange:}l_k(x)=\prod_{i=0,i\neq k}^{n}\frac{x-x_i}{x_k-x_i}=\frac{(x-x_0)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_n)}{(x_k-x_0)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_n)}$$

由此可得:$L_n(x)=\sum\limits_{k=0}^{n}y_kl_k(x)$ 称为 lagrange 插值函数

Newton 插值原理:

$f(x)$ 是需要插值的函数;$(x_0,y_0),(x_1,y_1),\cdots,(x_n,y_n)$ 是 $n+1$ 个插值节点;

(计算方法): 令 $N(x)=f_0+f[x_0,x_1](x-x_0)+\cdots+f[x_0,x_1,\cdots,x_n](x-x_0)\cdots(x-x_{n-1})$

其中 $f[x_i,x_{i+1},\cdots,x_j]=\frac{f[x_i,x_{i+1},\cdots,x_{j-1}]-f[x_{i+1},x_{i+1},\cdots,x_j]}{x_i-x_j}; f[x_i]=f(x_i); i=0,1,\cdots,n$

Piecewise − Linear 分段线性插值原理:

$f(x)$ 是需要插值的函数;$(x_0,y_0),(x_1,y_1),\cdots,(x_n,y_n)$ 是 $n+1$ 个插值节点;

(计算方法): 在每一区间 $[x_i,x_{i+1}]$ 上使用线性插值得到基函数 $\phi_i(x)=\frac{x-x_{i+1}}{x_i-x_{i+1}}y_i+\frac{x-x_i}{x_{i+1}-x_i}y_{i+1}$.

再将所有的 ϕ_i 并起来就是我们所需要的

Piecewise − Hermite 分段 *Hermite* 插值原理:

设 $f\in C^1[a,b]$; $(x_0,y_0,y_0'),(x_1,y_1,y_1'),\cdots,(x_n,y_n,y_n')$ 是 n+1 个插值节点及其导数值.

如果函数 $\varphi(x)$ 满足: (1) $\varphi\in C^1[a,b]$

(2) 满足插值条件 $\varphi\left(x_k\right)=f\left(x_k\right),\varphi'\left(x_k\right)=f'\left(x_k\right),k=0,1,\cdots,n$

(3) 在每个小区间 $[x_k,x_{k+1}]\,(k=0,1,\cdots,n-1)$ 上 $\varphi(x)$ 是一个三次多项式。则称 $\varphi(x)$ 为 $f(x)$ 以 $a=x_0<x_1<\cdots<x_n=b$ 为节点的分段三次 Hermite 插值多项式。

程序

必要的说明都在注释中

```
import time
import copy
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
```

```
#LAGRANGE插值函数
def Lagrange(x_list,y_list):
    n = len(x_list)
    result = Lagrange_basefunction_list(x_list,n)
    return sum([result[i]*y_list[i] for i in range(n)])

def Lagrange_basefunction_list(x_list,n): #基函数列
    result = []
    for i in range(n):
        result.append(Lagrange_basefunction(x_list,i))
    return result
def Lagrange_basefunction(x_list,i): #基函数
    X_list = copy.deepcopy(x_list)
    x_i = X_list[i]
    del X_list[i]
    denominator = 1
    for item in X_list:
        denominator = denominator*(x_i-item)
    coefficients_1 = np.poly(X_list) #计算分子函数的系数
    coefficients_2 = [value/denominator for value in coefficients_1] #计算函数系数
    basefunc = np.poly1d(coefficients_2,r = False , variable='x') #基函数
    return basefunc
```

```
#NEWTON插值函数
def Newton(x_list,y_list):
    dim = len(x_list)
    mat = list([j for j in range(dim)] for i in range(dim))
    def f(i,j):     #定义差商函数
        if j==0:
            return y_list[i]
        else:
            return (f(i-1,j-1)-f(i,j-1))/(x_list[i-j]-x_list[i])
```

```python
    for i in range(len(x_list)):   #计算矩阵
        for j in range(i+1):
            mat[i][j] = f(i,j)
    def N(k):       #定义返回x的多项式的函数
        func = 1
        if k != 0:
            func = np.poly1d(x_list[0:k],r = True ,variable = 'x')
        return func
    newton = 0
    for i in range(dim):
        newton += N(i)*mat[i][i]
    return newton
```

```python
#Hermite插值
def Hermite(x_list,y_list,z_list):
    n = len(x_list)-1
    def h(x_list,i):
        f = 0
        x_i = x_list[i]
        for j in range(n+1):
            if j!=i:
                f += 2*(np.poly1d([x_i],r=True,variable='x')/(x_list[j]-x_i))
        return (1+f)*Lagrange_basefunction(x_list,i)**2
    def H(x_list,i):
        return np.poly1d([x_list[i]],r=True,variable = 'x')*Lagrange_basefunction(x_list,i)**2
    hermite = 0
    for i in range(n+1):
        hermite += y_list[i]*h(x_list,i)+z_list[i]*H(x_list,i)
    return hermite
```

```python
#分段线性插值
def Piecewise_Linear(x_list,y_list,x_0):
    n = len(x_list)-1
    def interval(x_list,x):    #定义区间函数
        for index in range(n):
            if x_list[index] <= x <= x_list[index+1]:
                return index
    def phi(x_list,y_list,i):    #定义区间函数和函数字典
        f = y_list[i]*np.poly1d([x_list[i+1]],r=True,variable='x')/(x_list[i]-x_list[i+1])+\
        y_list[i+1]*np.poly1d([x_list[i]],r=True,variable='x')/(x_list[i+1]-x_list[i])
        return f
    phi_dict = {i:phi(x_list,y_list,i)
```

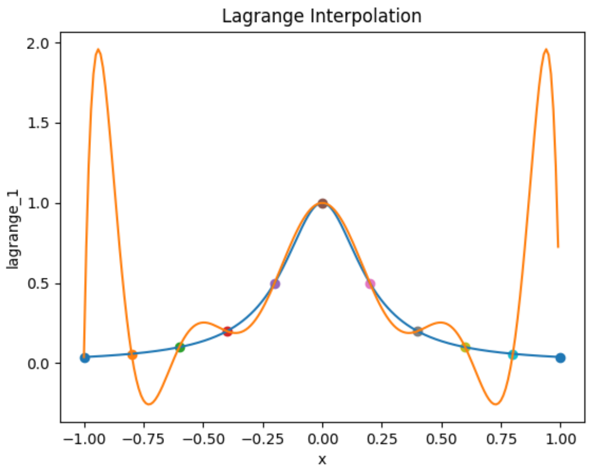
```
for i in range(n)}
    index = interval(x_list,x_0)
    return phi_dict[index]
```

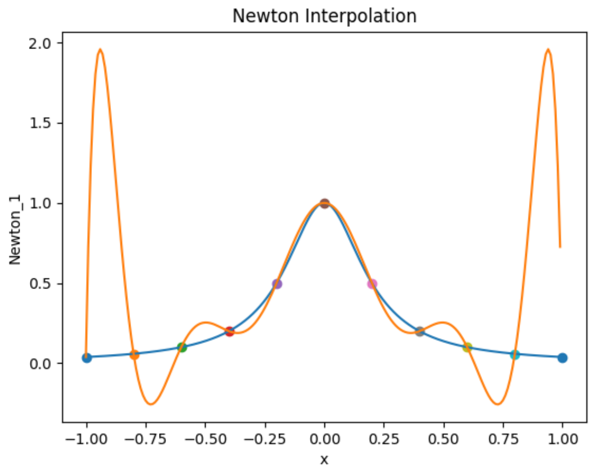
```
#分段HERMITE插值
def Piece_Hermite(x_list,y_list,z_list,x_0):
    n = len(x_list)-1
    def interval(x_list,x): #定义区间函数
        for index in range(n):
            if x_list[index] <= x <= x_list[index+1]:
                return index
    def piece(x_list,y_list,z_list,i):#定义每一段上的插值函数
        x_lst = [x_list[i],x_list[i+1]]
        y_lst = [y_list[i],y_list[i+1]]
        z_lst = [z_list[i],z_list[i+1]]
        return Hermite(x_list=x_lst,y_list=y_lst,z_list=z_lst)
        #函数字典
    piece_hermit_dict = {i:piece(x_list,y_list,z_list,i) for i in range(n)}
    index = interval(x_list,x_0)
    return piece_hermit_dict[index]#返回区间函数
```

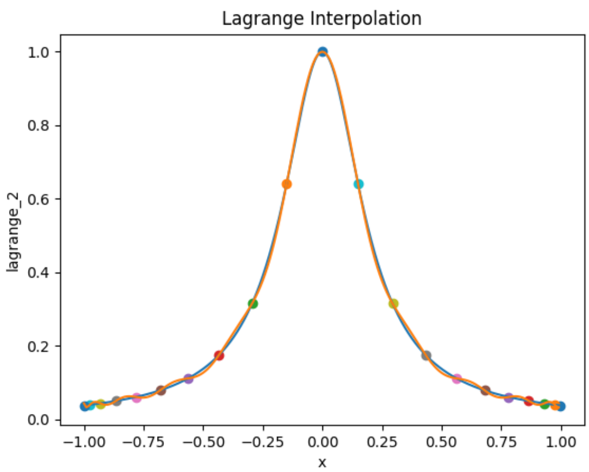
```
#最终实现:这里只展示问题(B)的绘图,其余的情况是类似的
n = max(10,int((input("请输入你的学号:\n"))[-1]))
def f(x):
    return 1/(1+25*x**2)
def diff_f(x):
    return -50*x/(1+25*x**2)**2
x_lst = np.arange(-1,1,0.001)
y_lst = f(x_lst)
def g(x): #定义插值节点函数
    import math
    return math.cos((2*x+1)*math.pi/42)
x_list_2 = []
y_list_2 = []
for i in range(21):
    x_i = g(i)
    x_list_2.append(x_i)
    y_list_2.append(f(x_i))
Lagrange_2 = Lagrange(x_list=x_list_2,y_list=y_list_2)
x_2_L = np.arange(-1,1,0.01)
y_2_L = Lagrange_2(x_2_L)
plt.plot(x_lst,y_lst)
plt.title('Lagrange Interpolation')
plt.plot(x_2_L,y_2_L)
plt.xlabel('x')
plt.ylabel('lagrange_2')
for i in range(21):
    plt.scatter(x_list_2[i],y_list_2[i])
plt.show()
```

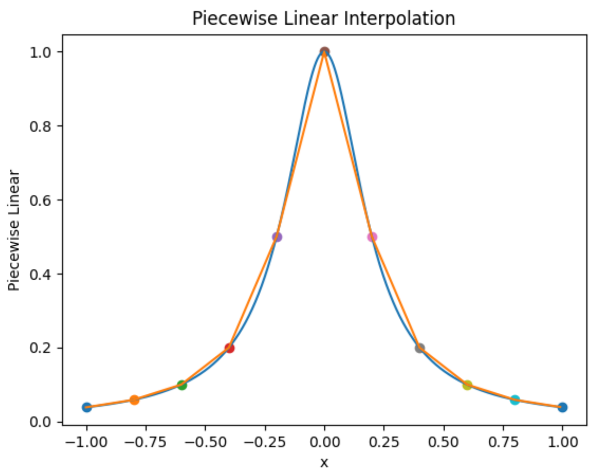
数据结果

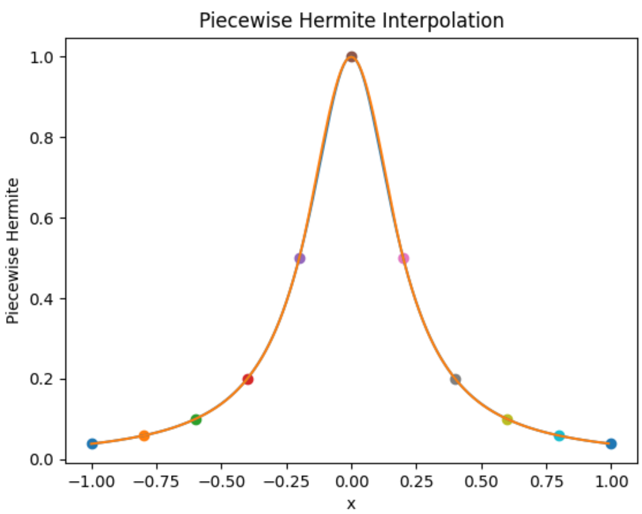
运行以上程序生成五张图像, 按顺序依次为 (a) 中 *lagrange* 插值图像,*Newton* 插值图像;(b)*Lagrange* 插值图像;(c) 分段线性插值图像;(d) 分段三次 *Hermite* 插值图像











结论

1. 从前两张图分钟可以看出 $Lagrange$ 插值和 $Newton$ 插值得到的函数是相同的, 并且采用等距插值节点时 $Lagrange$ 插值和 $Newton$ 插值在 0 附近的拟合效果很好, 但是当 $|x| > 0.3$ 左右的时候, 插值函数出现了剧烈的抖动, 精确度断崖式下降, 当 $x \to 0.9$ 的时候, 误差更是达到了原函数值的四倍之多.

2. 当插值节点更加密集分布在 $0.3 < |x| < 1$ 中时,$Lagrange$ 插值函数的拟合效果比等距节点好的多, 在 0 附近同样有较好的拟合效果, 当 $|x| > 0.3$ 时函数仅有微小的波动.

3. 采用等距节点时, 分段线性插值函数在 0 附近的拟合效果不及 $x \to 1$ 时的效果, 但是依旧没有很大的误差. 整体来说拟合程度较好. 相对的缺点就是在节点处曲线不光滑.

4. 分段 $Hermite$ 插值的拟合效果是最好的, 在 $[-1, 1]$ 上几乎与原函数重合.

总的来说, 分段 $Hermite$ 插值的拟合效果最好, 但是在运行中可以发现其求解速度与其他方法有明显差距; 分段线性插值在函数变化不剧烈, 节点较多时效果也很好;$Lagrange$ 插值和 $Newton$ 插值的拟合效果受节点选取的影响很大, 适当的选择插值节点可以获得更精确的插值函数.