拉格朗日插值及牛顿插值算法的实现

陈岳阳

2022 年 10 月 7 日

目录

1 实验目的

1.1 实现拉格朗日插值

使用 python 实现拉格朗日插值算法，均匀选取固定个数的样本点的情况下，选用不同的函数，观察插值曲线的拟合程度。

1.2 实现牛顿插值，显示差商结果

与实验目的 1 中类似，仅将“拉格朗日插值算法”改为“牛顿插值算法”，不做其他改变。

1.3 验证随着样本点的增多，插值曲线的变化情况

不改变函数的选用，改变插值点的个数，观察插值曲线的变化情况。

1.4 比较拉格朗日插值与牛顿插值的插值结果是否相同

考查相同函数、相同样本点的情况下，拉格朗日插值法和牛顿插值法的插值结果的差别。

1.5 二维插值：反距离权重插值法 (IDW) 的应用场景和简单实现

在这部分中，我们寻找了 IDW 的使用场景，并进行了基本代码的实现和简单的测试。

2 实验内容 (python3 实现)

为使插值算法的结果更加直观，实验中先实现了两个插值算法，然后对他们的插值效果进行测试。

2.1 实现拉格朗日插值

拉格朗日算法的公式如下：

$$y = \sum_{i=0}^{n} l(x_i) y_i$$

其中 $l(x_i)$ 表示第 i 个点的基函数。

由此可见，拉格朗日插值算法的实现，核心是基函数的实现。求基函数，并实现拉格朗日插值法的代码如下：

```
1  def Factorial(X, x, i):
2          # L_function
3      num = 1
4      for val in X:
5          if X[i] == val:
6              continue
7          num = num * (x - val)
8      return num
9
10
11 def Lagrange_interpolation(X, Y, x):
12         # Return the result of interpolation.
13     y = 0
14     for i in range(len(X)):
15         y += Factorial(X, x, i)/ Factorial(X, X[i], i) * Y[i]
16     return y
```

2.2 实现牛顿插值，显示差商结果

牛顿插值算法的公式如下：

$$\begin{aligned} y =& f(x_0) + f(x_0, x_1)(x - x_0) + f(x_0, x_1, x_2)(x - x_0)(x - x_1) \\ &+ ... + f(x_0, x_1, ..., x_n)(x - x_0)...(x - x_{n-1}) \end{aligned} \tag{1}$$

其中 $f(x_0, ...x_n)$ 是 x_0 到 x_n 的差商。

与拉格朗日插值算法相比，牛顿插值算法的优点在：对于多个插值点的情况，牛顿插值法不需要从头计算，从而有较低的复杂度。因此，在使用牛顿插值算法时，先将差商求出，然后对每个插值点进行计算。由此可见，牛顿插值算法的实现，核心是差商的计算。

求差商，并显示差商的牛顿插值算法的代码如下：

```
1         def Dif(X, Y, *, showDif=True):
2             # 差商
3             D = Y
4             for i in range(1, len(D)):
5                 for j in range(len(D)-1, i-1, -1):
6                     D[j] = (D[j] - D[j - 1]) / (X[j] - X[j - i])
7
8             # showDif
9             if showDif:
10                print("差商:")
11                for i in range(0, len(D)):
12                    print("f(", end="")
13                    for j in range(0, i ):
14                        print(f"x{j},", end="")
15                    print(f"x{i}", end="")
16                    print(f")={D[i]}")
17            return D
```

在完整代码中，由于显示差商占用过多的屏幕空间，故当样本点个数大于 10 时，不显示差商。

```
[9]: frac_test(points=5, bound_bias=0.1)
     frac_test(points=11, bound_bias=0.1)
     frac_test(points=15, bound_bias=0.1, showDif=False)
```

```
差商:
f(x0)=0.8
f(x0,x1)=1.2
f(x0,x1,x2)=0.4
f(x0,x1,x2,x3)=-0.7999999999999999
f(x0,x1,x2,x3,x4)=0.39999999999999997
```

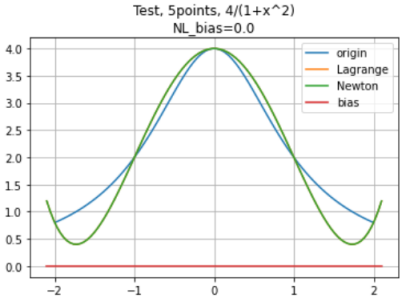


图 1: 差商的显示

有了差商的计算公式，即可求出牛顿插值法的结果。代码如下：

```
def Newton_interpolation(X, Y, x, D):
    """
    牛顿插值
    """
    mult = 1
    y = 0
    for i in range(0, len(X)):
        y += mult * D[i]
        mult = mult * (x - X[i])
    return y


def Newton_para(X, Y, x, *, showDif_para=True):
    """
    牛顿插值的参数，避免D的重复计算和错误输入
    """
    D = Dif(X, Y, showDif=showDif_para)
    return Newton_interpolation(X, Y, x, D)
```

2.3 验证随着样本点的增多，插值曲线的变化情况

我们选取分式函数 $y = \frac{4}{1+x^2}$、多项式函数 $y = x^3 + 3x^2 + x$ 和三角函数 $y = \sin x$ 进行插值，每种函数选取 3、5、11、15 个节点进行测试。结果图标题标注了选取点数 points、所选取函数的表达式、牛顿插值法与拉格朗日插值法的结果差值 (保留 12 位小数)。图中的非红色的三条曲线分别代表原函数 (Origin)，牛顿插值法的结果 (Newton) 和拉格朗日插值法的结果 (Lagrange)，红色曲线代表牛顿插值法与拉格朗日插值法在同一插值点的插值结果之差。

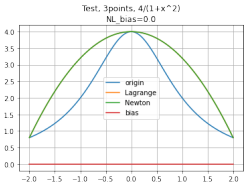
2.3.1 分式函数

我们选取的分式函数是

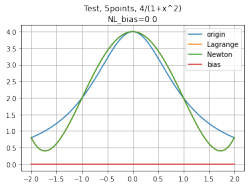
$$y = \frac{4}{1+x^2}$$

区间是 [-2,2]

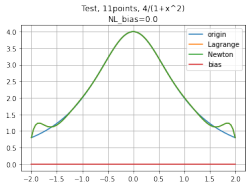
随着选取节点个数的增加，插值曲线的拟合程度逐渐优化，但是在边界处仍有很大误差。将 frac test 中的参数 bound bias 设为 0.1，可以看到在边界外，拟合情况迅速恶化，即所谓的龙格现象 (Runge phenomenon)。



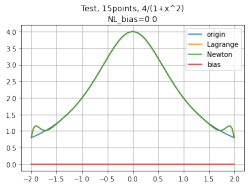
(a) 样本点 =3



(b) 样本点 =5

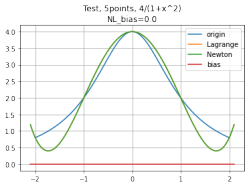


(c) 样本点 =11

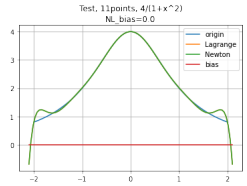


(d) 样本点 =15，边界处误差仍然较大

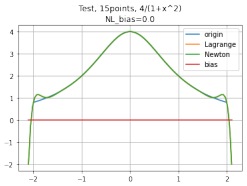
图 2: 分式函数



(a) 样本点 =5



(b) 样本点 =11，样本区间外 0.1 处拟合程度已恶化



(c) 样本点 =15，拟合程度恶化加剧

图 3: 龙格现象

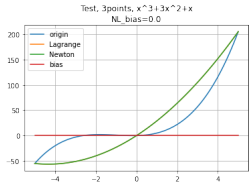
2.3.2 多项式函数

我们选取的多项式函数是

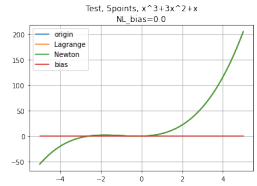
$$y = x^3 + 3x^2 + x$$

区间是 [-5,5]

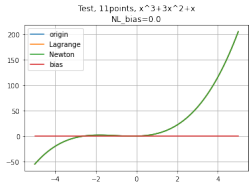
在样本点个数为 3 时，可以看出无法进行很好的拟合；而当样本点个数高于多项式次数时，拉格朗日插值法和牛顿插值法都可以完美拟合多项式函数。



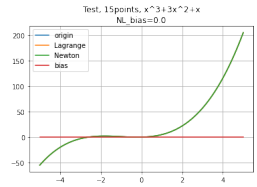
(a) 样本点 =3，误差较大



(b) 样本点 =5，开始完美拟合

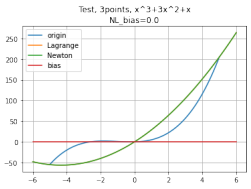


(c) 样本点 =11

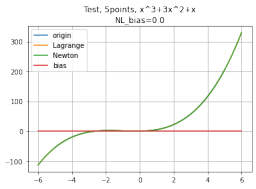


(d) 样本点 =15

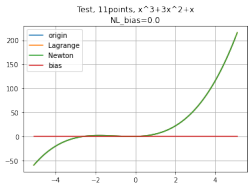
图 4: 多项式函数



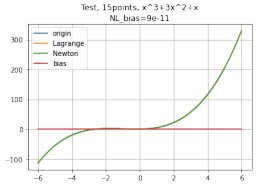
(a) 样本点 =3，误差较大



(b) 样本点 =5，在样本区间外仍完美拟合



(c) 样本点 =11



(d) 样本点 =15

图 5: 多项式函数在样本区间外的拟合情况

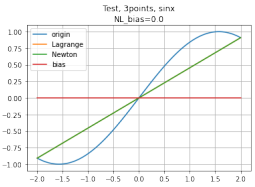
2.3.3 三角函数

我们选取的三角函数是

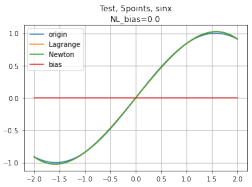
$$y = \sin x$$

区间是 [-2,2]

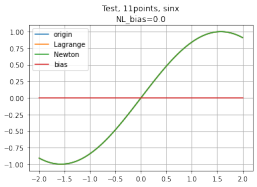
随着选取样本点个数的增加，插值函数迅速逼近原函数。选取 15 个样本点，可以看出在边界附近拟合程度也很高，这是由于 $y = \sin x$ 可以使用多项式级数逼近。但是在边界较远处，拟合程度仍会变差。



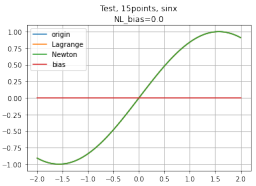
(a) 样本点 =3，误差较大



(b) 样本点 =5，仍有可见的误差

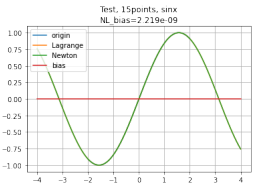


(c) 样本点 =11，拟合较完美

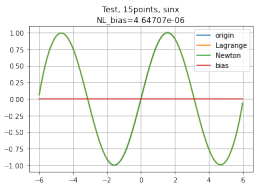


(d) 样本点 =15

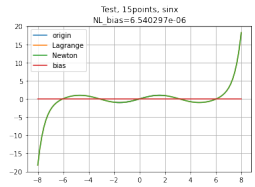
图 6: 三角函数



(a) 样本区间 =(-4,4)，拟合程度很高



(b) 样本区间 =(-6,6)，拟合程度仍很高



(c) 样本区间 =(-8,8)，拟合程度明显下降

图 7: 15 个样本点，不同样本区间中，三角函数的拟合情况

5.4. Interpolation Methods

Interpolation	Consistent?	PSNR↑	SSIM↑	LPIPS↓
Nearest	–	26.40	0.887	0.280
IDW 2D	✓	26.59	0.890	0.278
IDW 3D	–	26.57	0.890	0.278
Pixelwise Visibility	–	27.39	0.906	0.242
Imagewise Visibility	–	27.41	0.907	0.242

Table 3. Comparison of interpolation methods. For our flythrough video results, we opt for 2D inverse distance weighting (IDW) as it produces temporally consistent results.

We explore interpolation techniques in Table 3. The simple method of only rendering the nearest Block-NeRF to the camera requires the least amount of compute but results in harsh jumps when transitioning between blocks. These transitions can be smoothed by using inverse distance weighting (IDW) between the camera and Block-NeRF centers, as described in § 4.3.2. We also explored a variant of IDW where the interpolation was performed over projected 3D points predicted by the expected Block-NeRF depth. This method suffers when the depth prediction is incorrect, leading to artifacts and temporal incoherence.

Finally, we experiment with weighing the Block-NeRFs based on per-pixel and per-image predicted visibility. This produces sharper reconstructions of further-away areas but is prone to temporal inconsistency. Therefore, these methods are best used only when rendering still images. We provide further details in the supplement.

图 8: IDW 在 View Synthesis[1] 中的应用与性能

2.4 比较拉格朗日插值与牛顿插值的插值结果是否相同

由于拉格朗日插值法与牛顿插值法均使用 n 次多项式插值，理论插值结果与余项均相同。图中的红色曲线代表牛顿插值法与拉格朗日插值法在同一插值点的插值结果之差，结果保留 12 位小数；黄色曲线和绿色曲线分别表示拉格朗日插值法和牛顿插值法的插值曲线。在所有结果图中，黄色曲线和绿色曲线几乎重合，即使在边界外。红色曲线恒为 0。大部分情况中，两种方法的差值不超过 1e-12。较高误差的情况可能是浮点数运算失精引起的。可以验证拉格朗日插值法与牛顿插值法的插值结果和余项均相同。

2.5 二维插值：反距离权重插值法 (IDW) 的应用场景和简单实现

常用的二维插值方法较多，有双线性插值、样条插值、Kriging 等。我们在此介绍反距离权重插值法 (Inverse Distance Weighted)。反距离加权法

的公式如下：

$$W_i = \frac{h_i^{-p}}{\sum_{j=1}^{n} h_j^{-p}}, p \in R^+$$
$$h_i = \sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{2}$$

反距离加权法的结果受幂值 p 影响，通常取 0.5 到 3。我们的程序中，p=2。

反距离加权法是一种常用而简单的空间插值方法，在地理学科中较常用。它以插值点与样本点间的距离生成权重进行加权平均，离插值点越近的样本赋予的权重越大。IDW 简单易行，在样本点分布均匀时有很好的效果，但是容易受到极值的影响。

在 [1] 中，2 维反距离加权法的插值结果表现出了良好的一致性，且各项指标都超过了最邻近插值 (Nearest Interpolation)。

从实现结果中可以看出，IDW 的结果受插值点较近的样本点影响较大。

```
[51]: pointlist = [(1, 2, 20), (2, 3, 26), (3, 4, 23)]
      x = np.random.rand(3,) + 0.5
      y = np.random.rand(3,) + 1.5
      print(f"x={x.round(3)}, y={y.round(3)}, Interpolation={IDW(x, y, pointlist).round(3)}")
      for i in range(2):
          x = x + 1
          y = y + 1
          print(f"x={x.round(3)}, y={y.round(3)}, Interpolation={IDW(x, y, pointlist).round(3)}")

      x=[0.826 1.485 1.216], y=[1.756 1.781 2.253], Interpolation=[20.204 20.905 20.555]
      x=[1.826 2.485 2.216], y=[2.756 2.781 3.253], Interpolation=[25.526 25.138 25.558]
      x=[2.826 3.485 3.216], y=[3.756 3.781 4.253], Interpolation=[23.159 23.186 23.072]
```

图 9: IDW 实现结果

3 实验总结

1. 这次实验体会了插值方法的作用，加深了对牛顿插值和拉格朗日插值的理解。

2. 牛顿插值法和拉格朗日插值法的余项与插值结果均相同。

3. 一般情况下，样本点个数越多，插值函数越精确。

4. 一般情况下，在样本区间内部取点插值，精确度较高，外部则较低。样本点个数较多时，边界可能出现龙格现象。

5. 对于多项式函数，当样本点个数大于多项式次数时，插值曲线可以完美拟合多项式函数。

6. 在查阅资料的过程中加深了对于二维插值的理解，如双线性插值、IDW、Kriging 等。

7. 对于 IDW，插值点的值受距离较近的样本点影响较大。

4 附录

4.1 牛顿插值法与拉格朗日插值算法及测试代码

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


def Factorial(X, x, i):
    """
    用于求基函数
    """
    num = 1
    for val in X:
        if X[i] == val:
            continue
        num = num * (x - val)
    return num


def Lagrange_interpolation(X, Y, x):
    """
    拉格朗日插值法
    """
    y = 0
    for i in range(len(X)):
        y += Factorial(X, x, i)/ Factorial(X, X[i], i) * Y[i]
    return y


def Dif(X, Y, *, showDif=True):
    """
    差商
```

```
    """
    D = Y
    for i in range(1, len(D)):
        for j in range(len(D)-1, i-1, -1):
            D[j] = (D[j] - D[j - 1]) / (X[j] - X[j - i])

    """
    打印差商值
    """
    if showDif:
        print("差商:")
        for i in range(0, len(D)):
            print("f(", end="")
            for j in range(0, i ):
                print(f"x{j},", end="")
            print(f"x{i}", end="")
            print(f")={D[i]}")
    return D


def Newton_interpolation(X, Y, x, D):
    """
    牛顿插值
    """
    mult = 1
    y = 0
    for i in range(0, len(X)):
        y += mult * D[i]
        mult = mult * (x - X[i])
    return y


def Newton_para(X, Y, x, *, showDif_para=True):
```

```
64      """
65      牛顿插值的参数，避免D的重复计算和错误输入
66      """
67      D = Dif(X, Y, showDif=showDif_para)
68      return Newton_interpolation(X, Y, x, D)
69
70
71  def frac_test(*, lower_bound=-2, upper_bound=2, points=11, bound_bias=0, showDif=True):
72
73      """
74
75      [lower_bound, upper_bound]: 给定点的范围
76      points: 取点个数
77      bound_bias: 插值范围的偏移量
78      """
79          # 点数过多时，显示差商过于占用屏幕空间，故不显示
80      if points > 10:
81          showDif=False
82
83      X = np.linspace(lower_bound, upper_bound, 100)
84      Y = 4/(1+X ** 2)
85
86      # 给定点
87      someX = np.linspace(lower_bound, upper_bound, points)
88      someY = 4/(1+someX ** 2)
89
90      # 拉格朗日插值法
91      L_X = np.linspace(lower_bound-bound_bias, upper_bound+bound_bias, 100)
92      L_Y = Lagrange_interpolation(someX, someY, L_X)
93
94      # 牛顿插值法
95      N_X = L_X
96      N_Y = Newton_para(someX, someY, N_X, showDif_para=showDif)
```

```
97
98      # 两种方法的误差
99      bias_X = L_X
100     bias_Y = L_Y-N_Y
101     NL_bias=max(bias_Y)
102
103     fig , ax = plt.subplots()
104     plt.grid( visible =True)
105     ax.plot(X, Y, label="origin")
106     ax.plot(L_X, L_Y, label="Lagrange")
107     ax.plot(N_X, N_Y, label = "Newton")
108     ax.plot(bias_X, bias_Y, label="bias")
109     ax.set_title (f"Test, {points}points, 4/(1+x^2)\nNL_bias={round(NL_bias, 12)}")
110     ax.legend()
111     plt.show()
112
113 frac_test(points=3)
114 frac_test(points=5)
115 frac_test(points=11)
116 frac_test(points=15)
117
118 frac_test(points=5, bound_bias=0.1)
119 frac_test(points=11, bound_bias=0.1)
120 frac_test(points=15, bound_bias=0.1, showDif=False)
121
122
123 def poly_test(*, lower_bound=-5, upper_bound=5, points=11, bound_bias=0, showDif=True):
124
125     """
126
127     [lower_bound, upper_bound]: 给定点的范围
128     points: 取点个数
129     bound_bias: 插值范围的偏移量
```

```
    """
    if points > 10:
        showDif=False

    X = np.linspace(lower_bound, upper_bound, 100)
    Y = X**3 + 3 * X**2 + X

    # 给定点
    someX = np.linspace(lower_bound, upper_bound, points)
    someY = someX**3 + 3 * someX**2 + someX

    # 拉格朗日插值法
    L_X = np.linspace(lower_bound-bound_bias, upper_bound+bound_bias, 100)
    L_Y = Lagrange_interpolation(someX, someY, L_X)

    # 牛顿插值法
    N_X = L_X
    N_Y = Newton_para(someX, someY, N_X, showDif_para=showDif)

    # 两种方法的误差
    bias_X = L_X
    bias_Y = L_Y-N_Y
    NL_bias=max(bias_Y)

    fig, ax = plt.subplots()
    plt.grid(visible=True)
    ax.plot(X, Y, label="origin")
    ax.plot(L_X, L_Y, label="Lagrange")
    ax.plot(N_X, N_Y, label = "Newton")
    ax.plot(bias_X, bias_Y, label="bias")
    ax.set_title(f"Test, {points}points, x^3+3x^2+x\nNL_bias={round(NL_bias, 12)}")
    ax.legend()
    plt.show()
```

```
163
164
165 def sin_test (*, lower_bound=−2, upper_bound=2, points=11, bound_bias=0, showDif=True):
166
167     """
168
169     [lower_bound, upper_bound]: 给定点的范围
170     points: 取点个数
171     bound_bias: 插值范围的偏移量
172     """
173     if points > 10:
174         showDif=False
175
176     X = np.linspace(lower_bound, upper_bound, 100)
177     Y = []
178     for i in range(0, len(X)):
179         Y.append(math.sin(X[i]))
180
181     # 给定点
182     someX = np.linspace(lower_bound, upper_bound, points)
183     someY = []
184     for i in range(0, len(someX)):
185         someY.append(math.sin(someX[i]))
186
187     # 拉格朗日插值法
188     L_X = np.linspace(lower_bound−bound_bias, upper_bound+bound_bias, 100)
189     L_Y = Lagrange_interpolation(someX, someY, L_X)
190
191     # 牛顿插值法
192     N_X = L_X
193     N_Y = Newton_para(someX, someY, N_X, showDif_para=showDif)
194
195     # 两种方法的误差
```

```
196         bias_X = L_X
197         bias_Y = L_Y-N_Y
198         NL_bias=max(bias_Y)
199 
200         fig , ax = plt.subplots()
201         plt.grid( visible =True)
202         ax.plot(X, Y, label="origin")
203         ax.plot(L_X, L_Y, label="Lagrange")
204         ax.plot(N_X, N_Y, label = "Newton")
205         ax.plot(bias_X, bias_Y, label="bias")
206         ax.set_title (f"Test, {points}points, sinx\nNL_bias={round(NL_bias, 12)}")
207         ax.legend()
208         plt.show()
209 
210 poly_test(points=3)
211 poly_test(points=5)
212 poly_test()
213 poly_test(points=15)
214 
215 poly_test(points=3, bound_bias = 1)
216 poly_test(points=5, bound_bias = 1)
217 poly_test(bound_bias = 0.1)
218 poly_test(points=15, bound_bias = 1)
219 
220 sin_test(points=3)
221 sin_test(points=5)
222 sin_test()
223 sin_test(points=15)
224 
225 sin_test(points=15, bound_bias=2)
226 sin_test(points=15, bound_bias=4)
227 sin_test(points=15, bound_bias=6)
```

4.2 反距离加权法的代码实现

```
1  import numpy as np
2  import math as m
3
4
5  # 用于求距离反比平方之和
6  def Dsum(x, y, pointlist =[]):
7      dsum = 0
8      #point[x, y, sample]: 横坐标，纵坐标，样本值
9      for point in pointlist :
10         p_x = point[0]
11         p_y = point[1]
12         d_2 = (p_x - x) ** 2 + (p_y - y) ** 2
13         dsum += 1 / d_2
14     return dsum
15
16
17 # 用于求出pointlist中所有点的权值
18 def Weight(x, y, pointlist =[]):
19     dsum = Dsum(x, y, pointlist)
20     weight = []
21     for point in pointlist :
22         p_x = point[0]
23         p_y = point[1]
24         d_2 = (p_x - x) ** 2 + (p_y - y) ** 2
25         weight.append(1/d_2/dsum)
26     return weight
27 def IDW(x, y, pointlist =[]):
28     weight = Weight(x, y, pointlist )
29     i = 0
30     inter = 0
31     for i in range(len( pointlist )):
32         inter = pointlist[i][2] * weight[i] + inter
```

```
    return inter

pointlist = [(1, 2, 20), (2, 3, 26), (3, 4, 23)]
x = np.random.rand(3,) + 0.5
y = np.random.rand(3,) + 1.5
print(f"x={x.round(3)}, y={y.round(3)}, Interpolation={IDW(x, y, pointlist).round(3)}")
for i in range(2):
    x = x + 1
    y = y + 1
    print(f"x={x.round(3)}, y={y.round(3)}, Interpolation={IDW(x, y, pointlist).round(3)}
```

5 参考文献

参考文献

[1] Wiles, Olivia, Gkioxari, Georgia, Szeliski, Richard, Johnson and Justin. Block-NeRF: Scalable Large Scene Neural View Synthesis. In Proc. IEEE, pages 7465-7475, 2020.

6 致谢

感谢陈岳阳 (本人) 完成本实验所有资料搜寻、代码编写和调试工作。

感谢老师给我充足的时间完成这份实验报告。

感谢中国海洋大学的封校决定，让我有充足时间安全地完成这篇实验报告。