

# 几种多项式插值方法的应用与比较

陶 睿

(中国海洋大学 信息科学与工程学院 青岛 266100)

**摘要：**本文主要讨论插值法中Lagrange插值、Hermite插值、分段三次Hermite插值及三次样条插值，并在python中自己实现了这些算法。在此基础上,用不同的插值方法来逼近函数

$$f(x) = \frac{1}{1+25x^2}, \text{ 比较了各个方法的误差及其优缺点。}$$

**关键词：**Lagrange插值；Hermite插值；分段三次Hermite插值；三次样条插值；误差

**引言：**许多实际问题都用函数来表示某种内在规律的数量关系，其中相当一部分函数是通过实验或观测得到的。在海洋和大气的研究中，许多观测数据都是以区间[a,b]上一系列 $x_i$ 与 $y_i$ 函数值给出的。还有的问题中，虽然 $f(x)$ 在某个区间 $[a,b]$ 上是存在的，有的还是连续的，但却只能给出 $[a,b]$ 上一系列点 $x_i$ 的函数值，这只是一张函数表。因此，我们希望根据给定的函数表做一个既能反映函数 $f(x)$ 的特性，又便于计算简单函数 $p(x)$ ，用 $p(x)$ 近似 $f(x)$ 。通常选一类较简单的函数（如代数多项式或分段代数多项式）作为 $f(x)$ ，并使 $p(x_i)=f(x_i)$ 对 $i=0,1,2,\dots,n$ 成立.这样确定的 $p(x)$ 就是我们希望得到的插值函数。

## 一、几种插值方法的算法

### 1.1 Lagrange插值

已知定义在区间 $[a,b]$ 上的函数 $f(x)$ ，满足 $f(x) \in C^n_{[a,b]}$ ，且 $f^{(n+1)}(x)$ 在 $[a,b]$ 上存在。另有 $n+1$ 个包含于 $[a,b]$ 的插值节点 $x_0, x_1, \dots, x_n$ ，对应函数值为 $y_0, y_1, \dots, y_n$ ，则Lagrange插值的基函数为

$$L_i(x) = \frac{(x-x_0)(x-x_1)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)(x_i-x_1)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)} \quad (1)$$

$n$ 次Lagrange插值多项式为

$$P_n(x) = \sum_{i=0}^n y_i L_i(x) \quad (2)$$

### 1.2 Hermite插值

已知定义在区间  $[a, b]$  上的函数  $f(x)$ , 满足  $f(x) \in C_{[a,b]}^n$ , 且  $f^{(n+1)}(x)$  在  $[a, b]$  上存在。另有  $n+1$  个包含于  $[a, b]$  的插值节点  $x_0, x_1, \dots, x_n$ , 对应函数值为  $y_0, y_1, \dots, y_n$ , 对应的微商值为  $y'_0, y'_1, \dots, y'_n$ , Hermite插值的基函数为

$$h_i(x) = [1 - 2(x - x_i)l'_i(x_i)]l_i^2(x) \quad (3)$$

$$H_i(x) = l_i^2(x)(x - x_i), i = 0, 1, \dots, n \quad (4)$$

$2n+1$  次Hermite多项式为

$$H(x) = \sum_{i=0}^n (y_i h_i(x) + y'_i H_i(x)) \quad (5)$$

其插值余项为

$$R(x) = f(x) - H(x) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \omega_{n+1}^2(x) \quad (6)$$

其中,  $\xi \in (a, b)$ 。

### 1.3 分段三次 Hermite 插值

分段三次 Hermite插值是函数拟合的基本方法,在基础研究和工程技术中有着非常重要的应用。

若  $f(x) \in C^{(1)}[a, b]$ ,  $a = x_0 < x_1 < \dots < x_n = b$ , 则存在唯一的分段插值函数  $I(x)$ , 满足条件:

$$(1) I_n(x) \in C^{(1)}[a, b] \quad (7)$$

$$(2) I_n(x) = f(x_k), I'_n(x_k) = f'(x_k), k = 0, \dots, n; \quad (8)$$

$$(3) I_n(x) \text{ 在每个小区间 } [x_k, x_{k+1}] \text{ 上是关于 } x \text{ 的三次代数多项式。}$$

则称  $I_n(x)$  为  $f(x)$  的分段三次 Hermite插值。

$$I_n(x) = \sum_{i=0}^{i=n} (f(x_i)\alpha_i(x) + f'(x_i)\beta_i(x)) \quad (9)$$

$$\alpha_j(x) = \begin{cases} \left(\frac{x-x_j}{x_j-x_{j-1}}\right)^2 \left(1 + 2\frac{(x-x_j)}{x_{j-1}-x_j}\right) & x_{j-1} \leq x \leq x_j, j=1, \dots, n \\ \left(\frac{x-x_{j+1}}{x_j-x_{j+1}}\right)^2 \left(1 + 2\frac{(x-x_j)}{x_{j+1}-x_j}\right) & x_j \leq x \leq x_{j+1}, j=0, \dots, n-1 \\ 0 & x \notin [x_{j-1}, x_{j+1}] \end{cases} \quad (10)$$

$$\beta_j(x) = \begin{cases} (x-x_j)\left(\frac{x-x_{j-1}}{x_j-x_{j-1}}\right)^2 & x_{j-1} \leq x \leq x_j, j=1, \dots, n \\ (x-x_j)\left(\frac{x-x_{j+1}}{x_j-x_{j+1}}\right)^2 & x_j \leq x \leq x_{j+1}, j=0, \dots, n-1 \\ 0 & x \notin [x_{j-1}, x_{j+1}] \end{cases} \quad (11)$$

### 1.4 三次样条插值

已知区间  $[a, b]$  上  $n+1$  个插值节点  $x_0, x_1, \dots, x_n$ ，则三次样条插值函数为

$$S(x) = \begin{cases} S_1(x) \\ S_2(x) \\ \vdots \\ S_n(x) \end{cases} \quad (12)$$

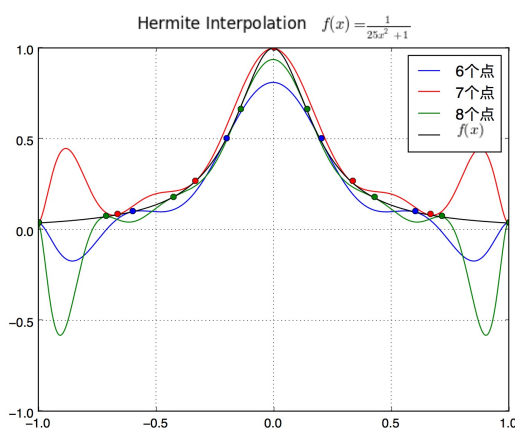
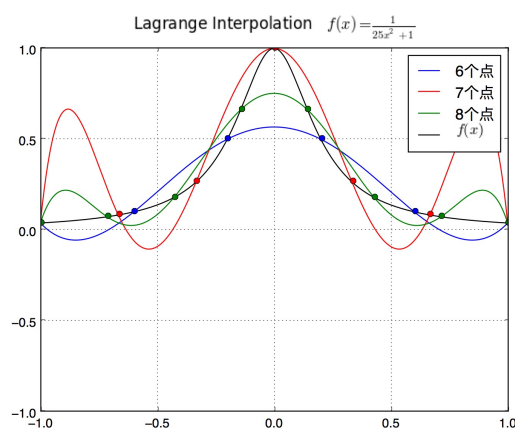
$$S_i(x) = \left( \frac{x-x_i}{h_{i-1}} \right)^2 \left( 1 + 2 \frac{x-x_{i-1}}{h_{i-1}} \right) y_{i-1} + \left( \frac{x-x_{i-1}}{h_{i-1}} \right)^2 \left( 1 + 2 \frac{x-x_i}{h_{i-1}} \right) y_i \\ + \left( \frac{x-x_i}{h_{i-1}} \right)^2 (x-x_{i-1}) m_{i-1} + \left( \frac{x-x_{i-1}}{h_{i-1}} \right)^2 (x-x_i) m_i \quad (13)$$

$$h_{i-1} = x_i - x_{i-1}, x_{i-1} \leq x \leq x_i, i = 1, 2, \dots, n \quad (14)$$

其中， $m_i$  为  $S(x)$  在点  $x_i$  处的微商值。

## 二、Lagrange插值与Hermite插值对比

### 2.1 插值曲线对比



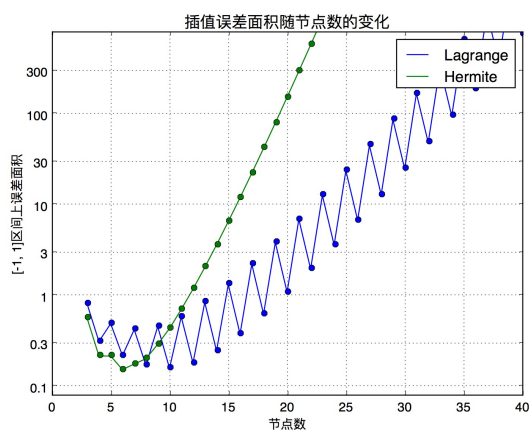
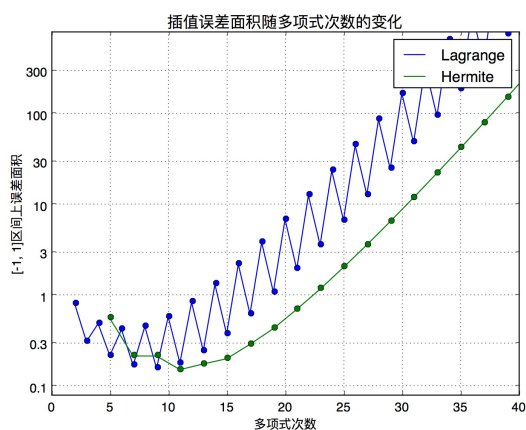
现象及分析：

如上图，左侧为Lagrange插值，右侧为Hermite插值。

可以看出在选取节点数较少时（6~8个点）：

- (1) 曲线两侧端点附近，二者表现相当，都有一定的偏离  $f(x)$  的现象。
- (2) 曲线中间，Hermite插值与  $f(x)$  更加贴近。原因是Hermite插值在节点处还符合一阶导数值相等。

### 2.2 误差对比



现象及分析：

如上图（误差面积\*采用了对数坐标），蓝色为Lagrange插值，绿色为Hermite插值。

可以看出，随着选取节点数以及多项式次数的增加：

（1）二者的误差面积都是先减少后增加，最终呈现指数增长的趋势。二者取得最小值时的多项式次数相近。

Lagrange插值在节点数10，次数9时为最小值。

Hermite 插值在节点数6，次数11时为最小值。

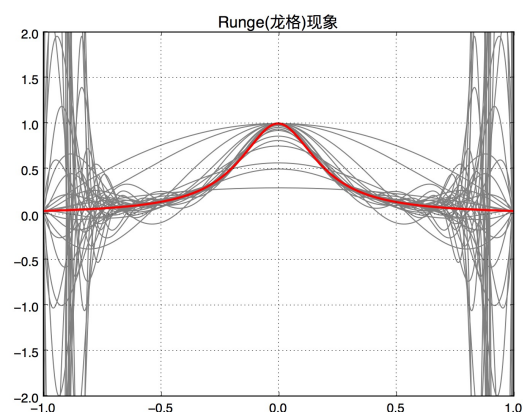
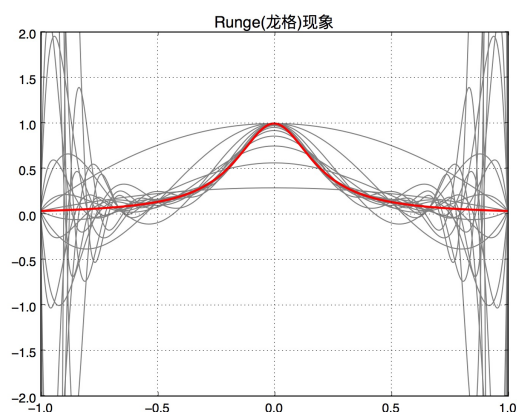
（2）Lagrange插值误差面积呈现奇偶不同的现象。多项式次数为奇数（节点数偶数）时，误差较小；Hermite插值无此现象。其多项式次数均为奇数。原因不明。

（3）在多项式次数相同时，Hermite插值误差面积更小。原因是Hermite插值在节点处还符合一阶导数值相等，数据信息更完善。

（4）在节点数相同时，Lagrange插值误差面积更小。原因是在选取 $n+1$ 个节点时，Lagrange插值多项式为 $n$ 次而Hermite插值多项式为 $2n+1$ 次。更高的多项式次数带来了更严重的数值不稳定现象。

\*误差面积指插值曲线与原函数曲线在区间 $[-1, 1]$ 上所夹区域的面积。

## 2.3 Runge（龙格）现象

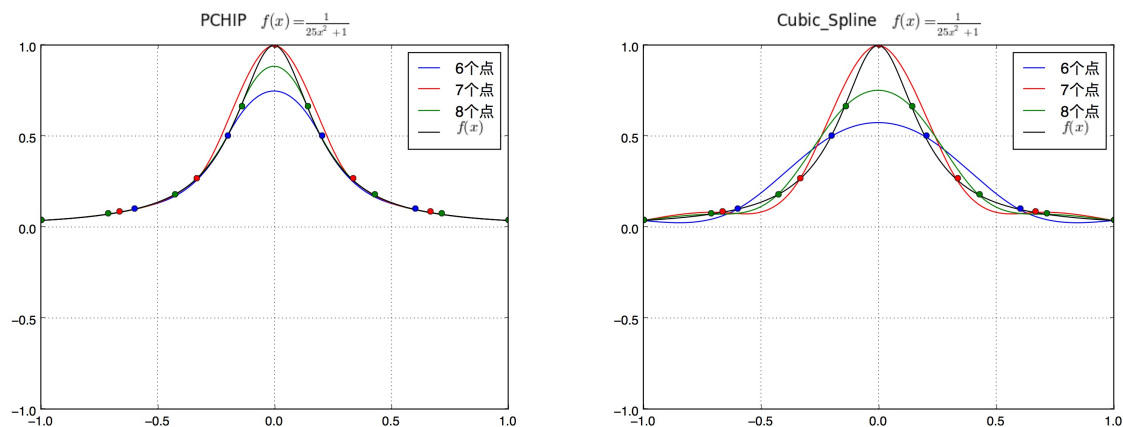


如上图，左侧为Lagrange插值的Runge现象，右侧为Hermite插值的Runge现象。

1901年，Carl David Tolmé Runge意外地发现，用插值多项式逼近函数 $f(x) = \frac{1}{1+25x^2}$ 时出现了一些反常的现象。当次数变高时，插值多项式反而变得更不准确。事实上，当次数 $n$ 趋于无穷时，该区间上的最大误差值也将趋于无穷大！

### 三、分段三次Hermite插值与三次样条插值对比

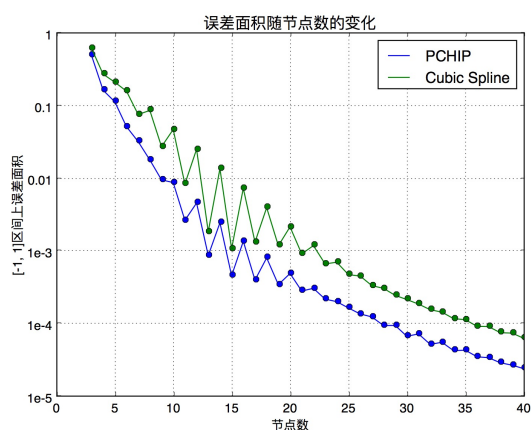
#### 3.1 插值曲线对比



现象及分析：

如上图，左侧为分段三次Hermite插值（下文称PCHIP），右侧为三次样条插值（下文称Spline）。可以看出，PCHIP与 $f(x)$ 更加贴近。原因是PCHIP在节点处的导数值采用了原函数的导数值，数据信息更完善；而Spline的导数值是计算出的，并未参照原函数。

#### 3.2 误差对比



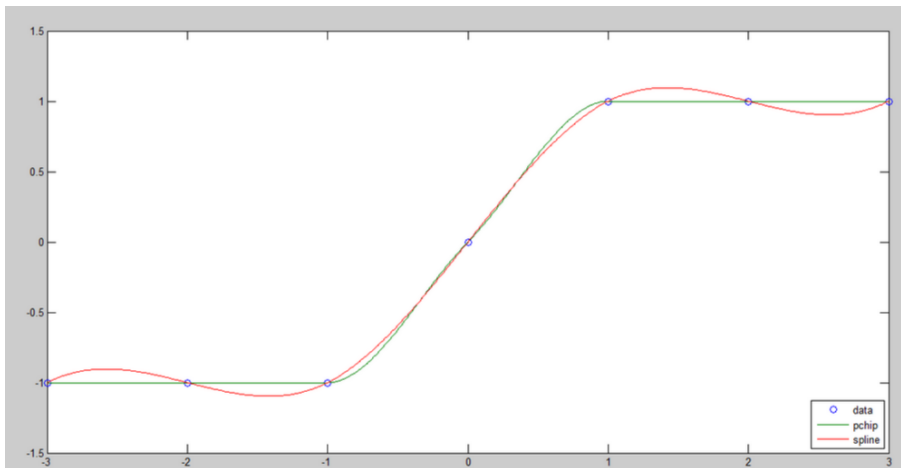
现象及分析：

如上图（误差面积\*采用了对数坐标），蓝色为PCHIP，绿色为Spline。可以看出：

- (1) 二者误差面积均持续减少，最终呈指数减少趋向于零。
- (2) PCHIP误差面积更小。原因同PCHIP曲线与 $f(x)$ 更加贴近的原因。

\*误差面积指插值曲线与原函数曲线在区间 $[-1, 1]$ 上所夹区域的面积。

### 3.2 其他对比



这里以 $X = [-3, -2, -1, 0, 1, 2, 3]$   $Y = [-1, -1, -1, 0, 1, 1, 1]$ 为例。如上图，绿色是 PCHIP，红色是 Spline。

Spline 构造  $S(x)$  的方式几乎与 PCHIP 构造  $P(x)$  的方式相同，Spline 的算法在根据边界条件计算出节点处的导数值向量  $M$  后，再调用 PCHIP 算法所得即为  $S(x)$ 。但是，Spline 在  $x_i$  处选择斜率的方式不同，使得  $S''(x)$  是连续的。这将产生以下效果：

Spline 产生更平滑的结果，即二阶导数连续。而 PCHIP 一阶导数连续。不连续的两阶导数隐含着不连续的曲率。人的眼睛可以检测出图形上曲率的不连续。

如果数据由平滑函数的值组成，则 Spline 可获得更精确的结果。

如果数据不平滑，则 PCHIP 不会超过目标值，也不太震荡。

计算二者的时间开销相当，PCHIP 建立的难度更小。

PCHIP 是保形\*的，而 Spline 不一定保形。

## 四、结束语

Lagrange 插值和 Hermite 插值的优点是表达式简单明确。缺点是如果要增加插值节点，公式必须整个改变，增加了计算量。而且在插值节点较多、多项式次数较高时具有数值不稳定的缺点。所以当区间较大、节点较多时，常用分段低次插值。由于分段插值是局部化的，从而带来了计算上的方便，可以步进地进行计算，同时也具有内在的高度稳定性和较好的收敛性。分段插值的缺点在于不能保证连接点处的光滑性。

如果插值总体平滑很重要，应该考虑运用三次样条插值或三次 Hermite 插值。表格数据构成函数的导数不存在时，要使用三次样条插值；要求保形性时，要使用分段三次 Hermite 插值。三次样条插值也是最常用的插值算法。

## 参考文献:

- [1] 彭湘晖. 几种常用插值方法比较分析[J]. 黑龙江水利科技, 2008, 1(36): 62-63.
- [2] 王东, 陶跃珍. 基于 Matlab 三次样条插值的连杆机构轨迹再现优化设计[J]. 机械传动, 2011, 35(1): 38-41.
- [3] 徐萃薇, 孙绳武. 计算方法引论[M]. 北京: 高等教育出版社, 2007.
- [4] 李洪发. 分段三次 Hermite 插值的同时逼近[J]. 天津师范大学学报, 2012, 32(2): 38-40
- [5] Matrix67. Runge 现象: 多项式插值不见得次数越高越准确[Z]. Matrix67 博客
- [6] 不明作者. 几种插值法的应用与比较[Z]. <http://www.docin.com/p-697360378.html>

## 附录（算法实现及绘图Python代码）：

### 1.Lagrange插值

```
# -*- coding:utf-8 -*-
# -----
# Python Lagrange-interpolation 拉格朗日插值
# Author: 陶睿 122345615@qq.com
# Date : 2015-10-18
# version 1.2
# -----

import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate # 求定积分函数

# -----
# 函数名: Lagrange_Interpolation(X, Y, Yd, t)
# 功能: 拉格朗日插值算法
# 说明:
#     X为自变量取值向量。共n+1个值。X[0]...X[n]。
#     Y为对应X的函数值向量。共n+1个值。Y[0]...Y[n]。
#     t是一个值或向量。计算t处的插值结果。如果t是向量，返回一个插值结果的向量。t应满足 X[0]<=t<=X[n]
#     插值多项式Pn(x)的次数为n次。
#     !需要import numpy。 X, Y, Yd为numpy.ndarray类型, t为numpy.float64类型或numpy.ndarray类型。
# 算法:
#     
$$L_i(x) = \prod_{j=0, j \neq i}^n (x - X[j]) / (X[i] - X[j])$$

#     
$$P_n(x) = \sum_{i=0}^n L_i(x) Y[i]$$

# -----
def Lagrange_Interpolation(X, Y, t):
    n = X.size - 1
    Pn = 0 # Pn: 拉格朗日插值多项式 Lagrange polynomial
    for i in range(0, n + 1):
        L = 1 # L: 拉格朗日插值基函数 Lagrange basis polynomials
        for j in range(0, n + 1):
            if (j != i):
                L *= (t - X[j]) / (X[i] - X[j])
        Pn += Y[i] * L
    return Pn

if __name__ == '__main__':

    y = lambda x: 1 / (1 + 25 * x**2)
    a = -1
    b = 1
    testX = np.linspace(a, b, 2001)
    testY = y(testX)
    path = u'/Users/sky/Desktop/计算方法/Interpolation/Figure/' # 图片存储地址
    # 图3: Runge(龙格)现象, 并记录误差err
    fig3 = plt.figure(13)
    plt.title(u'Runge(龙格)现象', fontsize = 15)
    ax3_1 = fig3.add_subplot(111)
    ax3_1.set_ylim(-2, 2)

    # 计算err随n的变化
```



```

nMax = 40
nBest = nMax
errBest = 9999
err = np.zeros(50)
for n in range(nMax, 1, -1): # n = nMax, nMax-1, ..., 2
    X = np.linspace(a, b, n + 1)
    Y = y(X)
    Pn = lambda x: Lagrange_Interpolation(X, Y, x)
    integrand = lambda x: abs(Pn(x) - y(x))
    err[n] = integrate.quad(integrand, a, b, limit = 2001)[0]
    #图3: Runge
    testF = Pn(testX)
    if (n <= 20):
        ax3_1.plot(testX, testF, color = 'gray', linestyle = '-', linewidth = 1)
    print u"#节点数 =", n + 1, u'次数 =', 2*n + 1, "err =", err[n]
    if(err[n] < errBest):
        nBest = n
        errBest = err[n]
print 'nBest = ', nBest
print 'errBest = ', errBest
nList = np.nonzero(err)[0]
errList = np.log10(err[err!=0])

#图3: Lagrange Runge
ax3_1.grid(True)
ax3_1.plot(testX, testY, color = 'red', linestyle = '-', linewidth = 2, label = u'原函数')
fig3.savefig(path + u'Lagrange_Runge.jpg')

#图4: err随节点数的变化(Lagrange VS Hermite)
fig4 = plt.figure(14)
plt.title(u'插值误差面积随节点数的变化', fontsize = 15)
ax4_1 = fig4.add_subplot(111)
ax4_1.set_xlabel(u'节点数')
ax4_1.set_ylabel(u'[-1, 1]区间上误差面积')
ax4_1.yaxis.set_ticks((-1, -0.52288, 0, 0.47712, 1, 1.47712, 2, 2.47712))
ax4_1.yaxis.set_ticklabels(('0.1', '0.3', '1', '3', '10', '30', '100', '300'))
ax4_1.set_ylim(-1.1, 2.9)
ax4_1.set_xlim(0, 40)
ax4_1.plot(nList + 1, errList, linestyle = '-', color = 'blue', label = 'Lagrange')
ax4_1.plot(nList + 1, errList, 'o', color = 'blue')
ax4_1.grid(True)
ax4_1.legend(loc = 'upper right')
fig4.savefig(path + u'误差对比1_2.jpg')

#图2: err随多项式次数的变化(Lagrange VS Hermite)
fig2 = plt.figure(12)
plt.title(u'插值误差面积随多项式次数的变化', fontsize = 15)
ax2_1 = fig2.add_subplot(111)
ax2_1.set_xlabel(u'多项式次数')
ax2_1.set_ylabel(u'[-1, 1]区间上误差面积')
ax2_1.yaxis.set_ticks((-1, -0.52288, 0, 0.47712, 1, 1.47712, 2, 2.47712))
ax2_1.yaxis.set_ticklabels(('0.1', '0.3', '1', '3', '10', '30', '100', '300'))
ax2_1.set_ylim(-1.1, 2.9)
ax2_1.set_xlim(0, 40)
ax2_1.plot(nList, errList, linestyle = '-', color = 'blue', label = 'Lagrange')
ax2_1.plot(nList, errList, 'o', color = 'blue')
ax2_1.grid(True)
ax2_1.legend(loc = 'upper right')
fig2.savefig(path + u'误差对比1_1.jpg')

```

```

#图1: Lagrange插值
n = 5
X1 = np.linspace(a, b, n + 1)
X2 = np.linspace(a, b, n + 2)
X3 = np.linspace(a, b, n + 3)
Y1 = y(X1)
Y2 = y(X2)
Y3 = y(X3)
testF1 = Lagrange_Interpolation(X1, Y1, testX)
testF2 = Lagrange_Interpolation(X2, Y2, testX)
testF3 = Lagrange_Interpolation(X3, Y3, testX)

fig1 = plt.figure(11)
plt.title('Lagrange Interpolation' + r'$f(x) = \frac{1}{25x^2 + 1}$', fontsize = 15)
ax1_1 = fig1.add_subplot(111)
ax1_1.set_ylim(-1, 1)
ax1_1.plot(testX, testF1, color = 'b', linestyle = '-', linewidth = 1, label = u'%d个点'%(n +
1))
ax1_1.plot(testX, testF2, color = 'r', linestyle = '-', linewidth = 1, label = u'%d个点'%(n +
2))
ax1_1.plot(testX, testF3, color = 'g', linestyle = '-', linewidth = 1, label = u'%d个点'%(n +
3))
ax1_1.plot(testX, testY, color = 'black', linestyle = '-', linewidth = 1, label = r'$f(x)$')
ax1_1.plot(X1, Y1, 'o', color = 'b')
ax1_1.plot(X2, Y2, 'o', color = 'r')
ax1_1.plot(X3, Y3, 'o', color = 'g')
ax1_1.grid(True)
ax1_1.legend(loc = 'upper right')
fig1.savefig(path + u'Lagrange_1.jpg')

```

## 2.Hermite插值

```
# -*- coding:utf-8 -*-
# -*- coding:utf-8 -*-
# -----
# Python Hermite-interpolation 埃尔米特插值
# Author: 陶睿 122345615@qq.com
# Date : 2015-10-18
# version 1.2
# -----

import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate

# -----
# 函数名: Hermite_Interpolation(X, Y, Yd, t)
# 功能: 埃尔米特插值算法
# 说明:
#     X为自变量取值向量。共n+1个值。X[0]...X[n]。
#     Y为对应X的函数值向量。共n+1个值。Y[0]...Y[n]。
#     Yd为对应X的导数值向量。共n+1个值。Yd[0]...Yd[n]。
#     t是一个值或向量。计算t处的插值结果。如果t是向量，返回一个插值结果的向量。t应满足 X[0]<=t<=X[n]
#     插值多项式Pn(x)的次数为2n+1次。
#     !需要import numpy。 X, Y, Yd为numpy.ndarray类型, t为numpy.float64类型或numpy.ndarray类型。
# 算法:
#      $H_{2n+1}(x) = \sum_{i=0}^n (y[i] * h_i(x) + y'[i] * H_i(x))$ 
#      $H_{2n+1}$  为  $2n+1$  次函数
#      $L_i(x) = \prod_{j=0, j \neq i}^n (x - X[j]) / (X[i] - X[j])$ 
#      $Lid(x[i]) = \sum_{j=0, j \neq i}^n 1 / (X[i] - X[j])$ 
#      $h_i(x) = (1 - 2(x - X[i]) * Lid(i, x)) * Li^2(x)$ 
#      $H_i(x) = (x - X[i]) * Li^2(x)$ 
# -----
def Hermite_Interpolation(X, Y, Yd, t):
    n = X.size - 1
    P = 0 # P: 2n+1次埃尔米特插值多项式 Hermite polynomial of degree 2n+1
    for i in range(0, n + 1):
        L = 1 # L: 拉格朗日插值基函数 Lagrange basis polynomials
        Ld = 0 # Ld: L在t = X[i]处的导数
        for j in range(0, n + 1):
            if (j != i):
                L *= (t - X[j]) / (X[i] - X[j])
                Ld += 1 / (X[i] - X[j])
        h = L**2 * (1 - 2 * (t - X[i]) * Ld)
        H = L**2 * (t - X[i])
        P += Y[i]*h + Yd[i]*H
    return P

if __name__ == '__main__':

    y = lambda x: 1 / (1 + 25 * x**2)
    yd = lambda x: -50*x / (625 * x**4 + 50 * x**2 + 1)
    a = -1
    b = 1
    testX = np.linspace(a, b, 2001)
    testY = y(testX)
    path = u'Users/sky/Desktop/计算方法/Interpolation/Figure/' # 图片存储地址
```

```

# 图3: Runge(龙格)现象, 并记录误差err
fig3 = plt.figure(23)
plt.title(u'Runge(龙格)现象', fontsize = 15)
ax3_1 = fig3.add_subplot(111)
ax3_1.set_ylim(-2, 2)

# 计算err随n的变化
nMax = 40
nBest = nMax
errBest = 9999
err = np.zeros(50)
for n in range(nMax, 1, -1): # n = nMax, nMax-1, ..., 2
    X = np.linspace(a, b, n + 1)
    Y = y(X)
    Yd = yd(X)
    H = lambda x: Hermite_Interpolation(X, Y, Yd, x)
    integrand = lambda x: abs(H(x) - y(x))
    err[n] = integrate.quad(integrand, a, b, limit = 2001)[0]
    #图3: Runge
    testF = H(testX)
    if (n <= 20):
        ax3_1.plot(testX, testF, color = 'gray', linestyle = '-', linewidth = 1)
    print u'#节点数 = ', n + 1, u'次数 = ', 2*n + 1, 'err = ', err[n]
    if(err[n] < errBest):
        nBest = n
        errBest = err[n]
print 'nBest = ', nBest
print 'errBest = ', errBest
nList = np.nonzero(err)[0]
errList = np.log10(err[err!=0])

#图3: Hermite Runge
ax3_1.grid(True)
ax3_1.plot(testX, testY, color = 'red', linestyle = '-', linewidth = 2, label = u'原函数')
fig3.savefig(path + u'Hermite_Runge.jpg')

#图4: err随节点数的变化(Lagrange VS Hermite)
fig4 = plt.figure(14)
plt.title(u'插值误差面积随节点数的变化', fontsize = 15)
ax4_1 = fig4.add_subplot(111)
ax4_1.set_xlabel(u'节点数')
ax4_1.set_ylabel(u'[-1, 1]区间上误差面积')
ax4_1.yaxis.set_ticks((-1, -0.52288, 0, 0.47712, 1, 1.47712, 2, 2.47712))
ax4_1.yaxis.set_ticklabels(('0.1', '0.3', '1', '3', '10', '30', '100', '300'))
ax4_1.set_ylim(-1.1, 2.9)
ax4_1.set_xlim(0, 40)
ax4_1.plot(nList + 1, errList, linestyle = '-', color = 'green', label = 'Hermite')
ax4_1.plot(nList + 1, errList, 'o', color = 'green')
ax4_1.grid(True)
ax4_1.legend(loc = 'upper right')
fig4.savefig(path + u'误差对比1_2.jpg')

#图2: err随多项式次数的变化(Lagrange VS Hermite)
fig2 = plt.figure(12)
plt.title(u'插值误差面积随多项式次数的变化', fontsize = 15)
ax2_1 = fig2.add_subplot(111)
ax2_1.set_xlabel(u'多项式次数')
ax2_1.set_ylabel(u'[-1, 1]区间上误差面积')

```

```

ax2_1.yaxis.set_ticks((-1, -0.52288, 0, 0.47712, 1, 1.47712, 2, 2.47712))
ax2_1.yaxis.set_ticklabels(('0.1', '0.3', '1', '3', '10', '30', '100', '300'))
ax2_1.set_ylim(-1.1, 2.9)
ax2_1.set_xlim(0, 40)
ax2_1.plot(2 * nList + 1, errList, linestyle = '-', color = 'green', label = 'Hermite')
ax2_1.plot(2 * nList + 1, errList, 'o', color = 'green')
ax2_1.grid(True)
ax2_1.legend(loc = 'upper right')
fig2.savefig(path + u'误差对比1_1.jpg')

```

#图1: Hermite插值

```

n = 5
X1 = np.linspace(a, b, n + 1)
X2 = np.linspace(a, b, n + 2)
X3 = np.linspace(a, b, n + 3)
Y1 = y(X1)
Y2 = y(X2)
Y3 = y(X3)
Yd1 = yd(X1)
Yd2 = yd(X2)
Yd3 = yd(X3)
testF1 = Hermite_Interpolation(X1, Y1, Yd1, testX)
testF2 = Hermite_Interpolation(X2, Y2, Yd2, testX)
testF3 = Hermite_Interpolation(X3, Y3, Yd3, testX)

fig1 = plt.figure(21)
plt.title('Hermite Interpolation' + r'$f(x) = \frac{1}{25x^2 + 1}$', fontsize = 15)
ax1_1 = fig1.add_subplot(111)
ax1_1.set_ylim(-1, 1)
ax1_1.plot(testX, testF1, color = 'b', linestyle = '-', linewidth = 1, label = u'%d个点'%(n +
1))
ax1_1.plot(testX, testF2, color = 'r', linestyle = '-', linewidth = 1, label = u'%d个点'%(n +
2))
ax1_1.plot(testX, testF3, color = 'g', linestyle = '-', linewidth = 1, label = u'%d个点'%(n +
3))
ax1_1.plot(testX, testY, color = 'black', linestyle = '-', linewidth = 1, label = r'$f(x)$')
ax1_1.plot(X1, Y1, 'o', color = 'b')
ax1_1.plot(X2, Y2, 'o', color = 'r')
ax1_1.plot(X3, Y3, 'o', color = 'g')
ax1_1.grid(True)
ax1_1.legend(loc = 'upper right')
fig1.savefig(path + u'Hermite_1.jpg')

```

### 3.分段三次Hermite插值

```
# -*- coding:utf-8 -*-
# -----
# Python PCHIP 分段三次Hermite插值
# Author: 陶睿 122345615@qq.com
# Date : 2015-10-30
# version 1.2
# -----

import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate

# -----
# 函数名: pchip(X, Y, Yd, t)
# Piecewise Cubic Hermite Interpolating Polynomial
# 功能: 分段三次Hermite插值算法
# 说明:
#     X为自变量取值向量。共n+1个值。X[0]...X[n]。
#     Y为对应X的函数值向量。共n+1个值。Y[0]...Y[n]。
#     Yd为对应X的导数值向量。共n+1个值。Yd[0]...Yd[n]。
#     t是一个值或向量。计算t处的插值结果。如果t是向量，返回一个插值结果的向量。t应满足  $X[0] \leq t \leq X[n]$ 。
#     插值多项式 $P_n(x)$ 的次数为3次。
#     !需要import numpy。 X, Y, Yd为numpy.ndarray类型, t为numpy.float64类型或numpy.ndarray类型。
# 算法:
#     对每一段 $[X_i, X_{i+1}]$ 调用Hermite_Interpolation算法。
# -----
def Hermite_Interpolation(X, Y, Yd, t):
    n = X.size - 1
    P = 0 # P:  $2n+1$ 次埃尔米特插值多项式 Hermite polynomial of degree  $2n+1$ 
    for i in range(0, n + 1):
        L = 1 # L: 拉格朗日插值基函数 Lagrange basis polynomials
        Ld = 0 # Ld: L在 $t = X[i]$ 处的导数
        for j in range(0, n + 1):
            if (j != i):
                L *= (t - X[j]) / (X[i] - X[j])
                Ld += 1 / (X[i] - X[j])
        h = L**2 * (1 - 2 * (t - X[i]) * Ld)
        H = L**2 * (t - X[i])
        P += Y[i]*h + Yd[i]*H
    return P

def pchip(X, Y, Yd, t):
    import numpy
    n = X.size - 1

    def _pchip(t):
        yi = 0
        pos = 0 # t在区间 $[X[pos], x[pos+1]]$ 中
        for i in range(0, n):
            if ((X[i] <= t) and (t <= X[i+1])):
                pos = i
        yi = Hermite_Interpolation(X[pos: pos+2], Y[pos: pos+2], Yd[pos: pos+2], t)
        return yi

    if (type(t) == numpy.float64 or type(t) == float):
        return _pchip(t)
```

```

elif (type(t) == numpy.ndarray):
    P = np.zeros((t.size))
    for i in range(0, t.size):
        P[i] = _pchip(t[i])
    return P
# -----
# End of pchip(X, Y, Yd, t)
# -----

if __name__ == '__main__':

    y = lambda x: 1 / (1 + 25 * x**2)
    yd = lambda x: -50*x / (625 * x**4 + 50 * x**2 + 1)
    a = -1
    b = 1
    testX = np.linspace(a, b, 2001)
    testY = y(testX)
    path = u'Users/sky/Desktop/计算方法/Interpolation/Figure/' # 图片存储地址

    # 计算err随n的变化
    nMax = 40
    nBest = nMax
    errBest = 9999
    err = np.zeros(50)
    for n in range(nMax, 1, -1): # n = nMax, nMax-1, ..., 2
        X = np.linspace(a, b, n + 1)
        Y = y(X)
        Yd = yd(X)
        H = lambda x: pchip(X, Y, Yd, x)
        integrand = lambda x: abs(H(x) - y(x))
        err[n] = integrate.quad(integrand, a, b, limit = 2001)[0]
        print '#n = ', n, 'err = ', err[n]
        if(err[n] < errBest):
            nBest = n
            errBest = err[n]
    print 'nBest = ', nBest
    print 'errBest = ', errBest
    nList = np.nonzero(err)[0]
    errList = np.log10(err[err!=0])

    # 图2: err随节点数的变化(PCHIP VS Cubic_Spline)
    fig2 = plt.figure(32)
    plt.title(u'误差面积随节点数的变化', fontsize = 15)
    ax2_1 = fig2.add_subplot(111)
    ax2_1.set_xlabel(u'节点数')
    ax2_1.set_ylabel(u'[-1, 1]区间上误差面积')
    ax2_1.yaxis.set_ticks((-5, -4, -3, -2, -1, 0))
    ax2_1.yaxis.set_ticklabels(('1e-5', '1e-4', '1e-3', '0.01', '0.1', '1'))
    ax2_1.set_ylim(-5, 0)
    ax2_1.set_xlim(0, 40)
    ax2_1.plot(nList + 1, errList, linestyle = '-', color = 'blue', label = 'PCHIP')
    ax2_1.plot(nList + 1, errList, 'o', color = 'blue')
    ax2_1.grid(True)
    ax2_1.legend(loc = 'upper right')
    fig2.savefig(path + u'误差对比2.jpg')

    #图1: PCHIP
    n = 5
    X1 = np.linspace(a, b, n + 1)

```

```

X2 = np.linspace(a, b, n + 2)
X3 = np.linspace(a, b, n + 3)
Y1 = y(X1)
Y2 = y(X2)
Y3 = y(X3)
Yd1 = yd(X1)
Yd2 = yd(X2)
Yd3 = yd(X3)
testF1 = pchip(X1, Y1, Yd1, testX)
testF2 = pchip(X2, Y2, Yd2, testX)
testF3 = pchip(X3, Y3, Yd3, testX)

fig1 = plt.figure(31)
plt.title('PCHIP ' + r'$f(x) = \frac{1}{25x^2 + 1}$', fontsize = 15)
ax1_1 = fig1.add_subplot(111)
ax1_1.set_ylim(-1, 1)
ax1_1.plot(testX, testF1, color = 'b', linestyle = '-', linewidth = 1, label = u'%d个点'
'%(n+1))
ax1_1.plot(testX, testF2, color = 'r', linestyle = '-', linewidth = 1, label = u'%d个点'
'%(n+2))
ax1_1.plot(testX, testF3, color = 'g', linestyle = '-', linewidth = 1, label = u'%d个点'
'%(n+3))
ax1_1.plot(testX, testY, color = 'black', linestyle = '-', linewidth = 1, label = r'$f(x)$')
ax1_1.plot(X1, Y1, 'o', color = 'blue')
ax1_1.plot(X2, Y2, 'o', color = 'red')
ax1_1.plot(X3, Y3, 'o', color = 'green')
ax1_1.grid(True)
ax1_1.legend(loc = 'upper right')
fig1.savefig(path + u'PCHIP_1.jpg')

```



## 4.三次样条插值

```
# -*- coding:utf-8 -*-
# -----
# Python Cubic Spline 三次样条插值
# Author: 陶睿 122345615@qq.com
# Date : 2015-10-31
# version 1.2
# -----

import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate

# -----
# 函数名: Cubic_Spline(X, Y, t)
# Cubic Spline Interpolation
# 功能: 三次样条插值算法(边界条件为端点处二阶微商已知且为0, 即 $s''(x_0) = 0$ ,  $s''(x_n) = 0$ )
# 说明:
#     X为自变量取值向量。共n+1个值。X[0]...X[n]。
#     Y为对应X的函数值向量。共n+1个值。Y[0]...Y[n]。
#     t是一个值或向量。计算t处的插值结果。如果t是向量, 返回一个插值结果的向量。t应满足  $X[0] \leq t \leq X[n]$ 。
#     插值多项式 $P_n(x)$ 的次数为3次。
#     !需要import numpy。 X, Y, Yd为numpy.ndarray类型, t为numpy.float64类型或numpy.ndarray类型。
# 算法:
#     计算出M向量后。再调用分段三次插值的算法(pchip(X, Y, Yd, t))。
# -----
def Hermite_Interpolation(X, Y, Yd, t):
    n = X.size - 1
    P = 0 # P: 2n+1次埃尔米特插值多项式 Hermite polynomial of degree 2n+1
    for i in range(0, n + 1):
        L = 1 # L: 拉格朗日插值基函数 Lagrange basis polynomials
        Ld = 0 # Ld: L在 $t = X[i]$ 处的导数
        for j in range(0, n + 1):
            if (j != i):
                L *= (t - X[j]) / (X[i] - X[j])
                Ld += 1 / (X[i] - X[j])
        h = L**2 * (1 - 2 * (t - X[i]) * Ld)
        H = L**2 * (t - X[i])
        P += Y[i]*h + Yd[i]*H
    return P

def pchip(X, Y, Yd, t):
    import numpy as np
    n = X.size - 1

    def _pchip(t):
        yi = 0
        pos = 0 # t在区间[X[pos], x[pos+1]]中
        for i in range(0, n):
            if ((X[i] <= t) and (t <= X[i+1])):
                pos = i
        yi = Hermite_Interpolation(X[pos: pos+2], Y[pos: pos+2], Yd[pos: pos+2], t)
        return yi

    if (type(t) == np.float64 or type(t) == float):
        return _pchip(t)
    elif (type(t) == np.ndarray):
```

```

        P = np.zeros((t.size))
        for i in range(0, t.size):
            P[i] = _pchip(t[i])
        return P

def Cubic_Spline(X, Y, t):
    import numpy as np
    n = X.size - 1
    h = np.zeros(X.size)
    h[0: n] = X[1: n+1] - X[0: n]
    dds0 = 0. # S''(x0)
    ddsn = 0. # S''(xn)
    A = np.zeros((n + 1))
    B = np.zeros((n + 1))
    beta = np.zeros((n + 1))
    alpha = np.zeros((n + 1))
    alpha[0] = 1.
    alpha[n] = 0.
    beta[0] = 3./h[0] * (Y[1] - Y[0]) - h[0]/2. * dds0
    beta[n] = 3./h[n-1] * (Y[n] - Y[n-1]) - h[n-1]/2. * ddsn
    for i in range(1, n):
        alpha[i] = h[i-1]/(h[i-1] + h[i])
        beta[i] = 3.*( (1-alpha[i])/h[i-1]*(Y[i] - Y[i-1]) \
            +( alpha[i])/h[i] * (Y[i+1] - Y[i]) )

    #追赶法解三对角矩阵
    A[0] = -alpha[0]/2.
    B[0] = beta[0]/2.
    for i in range(1, n + 1):
        if (i < n):
            A[i] = -alpha[i]/(2.+(1-alpha[i])*A[i-1])
            B[i] = (beta[i] - (1.-alpha[i])*B[i-1]) / (2. + (1.-alpha[i])*A[i-1])
    M = np.zeros((n + 1))
    M[n] = B[n]
    for i in range(n - 1, -1, -1):
        M[i] = A[i]*M[i+1]+B[i]
    #M = np.linalg.solve(A, beta) #numpy.linalg.solve(A, B)是numpy中求解线性方程组的函数
    return pchip(X, Y, M, t)

# -----
# End of Cubic_Spline(X, Y, t)
# -----

if __name__ == '__main__':

    y = lambda x: 1 / (1 + 25 * x**2)
    a = -1
    b = 1
    testX = np.linspace(a, b, 2001)
    testY = y(testX)
    path = u'/Users/sky/Desktop/计算方法/Interpolation/Figure/' # 图片存储地址

    # 计算err随n的变化
    nMax = 40
    nBest = nMax
    errBest = 9999
    err = np.zeros(50)
    for n in range(nMax, 1, -1): # n = nMax, nMax-1, ..., 2
        X = np.linspace(a, b, n + 1)
        Y = y(X)
        H = lambda x: Cubic_Spline(X, Y, x)

```

```

        integrand = lambda x: abs(H(x) - y(x))
        err[n] = integrate.quad(integrand, a, b, limit = 2001)[0]
        print '#n = ', n, 'err = ', err[n]
        if(err[n] < errBest):
            nBest = n
            errBest = err[n]
    print 'nBest = ', nBest
    print 'errBest = ', errBest
    nList = np.nonzero(err)[0]
    errList = np.log10(err[err!=0])

# 图2: err随节点数的变化(PCHIP VS Cubic_Spline)
fig2 = plt.figure(32)
plt.title(u'误差面积随节点数的变化', fontsize = 15)
ax2_1 = fig2.add_subplot(111)
ax2_1.set_xlabel(u'节点数')
ax2_1.set_ylabel(u'[-1, 1]区间上误差面积')
ax2_1.yaxis.set_ticks((-5, -4, -3, -2, -1, 0))
ax2_1.yaxis.set_ticklabels(('1e-5', '1e-4', '1e-3', '0.01', '0.1', '1'))
ax2_1.set_ylim(-5, 0)
ax2_1.set_xlim(0, 40)
ax2_1.plot(nList + 1, errList, ls = '-', color = 'green', label = 'Cubic Spline')
ax2_1.plot(nList + 1, errList, 'o', color = 'green')
ax2_1.grid(True)
ax2_1.legend(loc = 'upper right')
fig2.savefig(path + u'误差对比2.jpg')

#图1: Cubic_Spline
n = 5
X1 = np.linspace(a, b, n + 1)
X2 = np.linspace(a, b, n + 2)
X3 = np.linspace(a, b, n + 3)
Y1 = y(X1)
Y2 = y(X2)
Y3 = y(X3)
testF1 = Cubic_Spline(X1, Y1, testX)
testF2 = Cubic_Spline(X2, Y2, testX)
testF3 = Cubic_Spline(X3, Y3, testX)

fig1 = plt.figure(41)
plt.title('Cubic_Spline' + r'$f(x) = \frac{1}{25x^2 + 1}$', fontsize = 15)
ax1_1 = fig1.add_subplot(111)
ax1_1.set_ylim(-1, 1)
ax1_1.plot(testX, testF1, color = 'b', linestyle = '-', linewidth = 1, label = u'%d个点'
'%(n+1))
ax1_1.plot(testX, testF2, color = 'r', linestyle = '-', linewidth = 1, label = u'%d个点'
'%(n+2))
ax1_1.plot(testX, testF3, color = 'g', linestyle = '-', linewidth = 1, label = u'%d个点'
'%(n+3))
ax1_1.plot(testX, testY, color = 'black', linestyle = '-', linewidth = 1, label = r'$f(x)$')
ax1_1.plot(X1, Y1, 'o', color = 'blue')
ax1_1.plot(X2, Y2, 'o', color = 'red')
ax1_1.plot(X3, Y3, 'o', color = 'green')
ax1_1.grid(True)
ax1_1.legend(loc = 'upper right')
fig1.savefig(path + u'Cubic_Spline_1.jpg')

```