利用python进行样条曲线拟合

笔记本:	我的第一个笔记本		
创建时间:	2019/4/26 15:02	**更新时间:**	2019/4/28 10:37
作者:	977616672@qq.com		
URL:	file:///C:/Users/SelfDriving/Desktop/论文/每周资料/20190421-邵.docx		

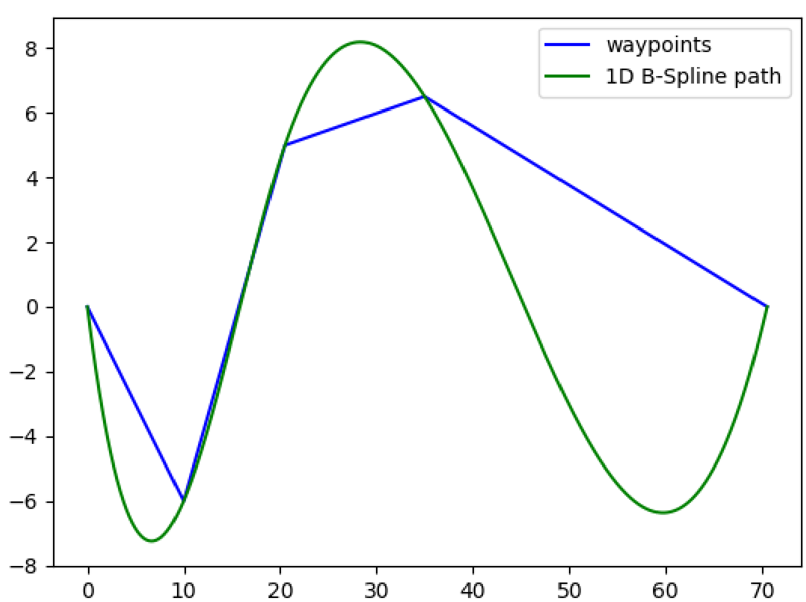
利用python进行样条曲线拟合

一、使用python的scipy的interpolate库函数进行1D样条曲线拟合

输入要求：横坐标单调递增

```
x = [0.0, 10.0, 20.5, 35.0, 70.5]
y = [0.0, -6.0, 5.0, 6.5, 0.0]
t=interpolate.splrep(x,y,k=3)
x0=np.linspace(0,x[-1],1000)
y0=interpolate.splev(x0,t)
flg, ax = p.subplots(1)
p.plot(x,y,color='blue',label="waypoints")
p.plot(x0,y0,color='green',label="1D B-Spline path")
```

splrep函数用于计算系数矩阵，结果作为splev的第二个参数，其中参数k为样条次数。
splev函数用于计算样条曲线纵坐标。



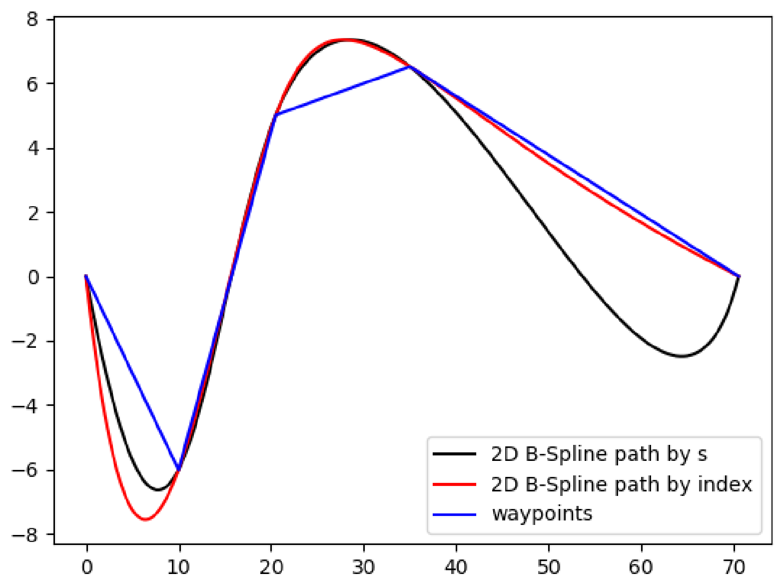
二、使用python的库函数进行2D样条曲线拟合

方法1：将（x,y）用参数方程表示，分为（s,x）和（s,y），其中s为线段距离；

方法2：将（x,y）用参数方程表示，分为（index,x）和（index,y），其中index=0,...len(x)-1为索引值

```
t1=interpolate.splrep(csp.s,x,k=3)
t2=interpolate.splrep(csp.s,y,k=3)
t0=np.linspace(0,csp.s[-1],1000)
x1=interpolate.splev(t0,t1)
y1=interpolate.splev(t0,t2)
p.plot(x1,y1,color='black', label="2D B-Spline path by s")

t11=interpolate.splrep([tt for tt in range(len(x))],x,k=3)
t21=interpolate.splrep([tt for tt in range(len(x))],y,k=3)
t01=np.linspace(0,len(x)-1,1000)
x11=interpolate.splev(t01,t11)
y11=interpolate.splev(t01,t21)
p.plot(x11,y11,color='red', label="2D B-Spline path by index")
p.plot(x,y,color='blue',label="waypoints")
p.legend()
p.show()
```



三、自定义函数进行2D样条插值

3.1 三次样条曲线原理

假设有以下节点

$$
\begin{array}{lcccccccc}
x: & a = x_0 & < & x_1 & < & \cdots & < & x_n = b \\
y: & y_0 & & y_1 & & \cdots & & y_n
\end{array}
$$

样条曲线 $S(x)$ 是一个分段定义的公式。给定n+1个数据点，共有n个区间，三次样条方程满足以下条件：

a. 在每个分段区间 $[x_i, x_{i+1}]$ （i = 0, 1, ..., n-1，x递增）， $S(x) = S_i(x)$ 都是一个三次多项式。

b. 满足$S(x_i) = y_i$ （i = 0, 1, …, n ）

c. $S(x)$ ，导数$S'(x)$ ，二阶导数$S''(x)$ 在[a, b]区间都是连续的，即$S(x)$ 曲线是光滑的。

所以n个三次多项式分段可以写作：

$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$ ，i = 0, 1, …, n-1

其中ai, bi, ci, di代表4n个未知系数。

3.2 求解

已知:

a. n+1个数据点[xi, yi], i = 0, 1, …, n

b. 每一分段都是三次多项式函数曲线

c. 节点达到二阶连续

d. 左右两端点处特性（自然边界，固定边界，非节点边界）

根据定点，求出每段样条曲线方程中的系数，即可得到每段曲线的具体表达式。

插值和连续性：

$S_i(x_i) = y_i$

$S_i(x_{i+1}) = y_{i+1}$, 其中 i = 0, 1, …, n-1

微分连续性：

$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$

$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$, 其中 i = 0, 1, …, n-2

样条曲线的微分式：

$$
\begin{aligned}
S_i(x) &= a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \\
S_i'(x) &= b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2 \\
S_i''(x) &= 2c_i + 6d_i(x - x_i)
\end{aligned}
$$

将步长$h_i = x_{i+1} - x_i$ 带入样条曲线的条件：

a. 由 $S_i(x_i) = y_i$ (i = 0, 1, …, n-1)推出

$$a_i = y_i$$

b. 由$S_i(x_{i+1}) = y_{i+1}$ (i = 0, 1, …, n-1)推出

$$a_i + h_i b_i + h_i^2 c_i + h_i^3 d_i = y_{i+1}$$

c. 由 $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$ (i = 0, 1, …, n-2)推出

$S'_i(x_{i+1}) = b_i + 2c_i(x_{i+1} - x_i) + 3d_i(x_{i+1} - x_i)^2 = b_i + 2c_i h + 3d_i\, h^2$
$S'_{i+1}(x_{i+1}) = b_{i+1} + 2c_i(x_{i+1} - x_{i+1}) + 3d_i(x_{i+1} - x_{i+1})^2 = b_{i+1}$

由此可得：

$$b_i + 2h_i c_i + 3h_i^2 d_i - b_{i+1} = 0$$

d. 由 $S''_i(x_{i+1}) = S''_{i+1}(x_{i+1})$ (i = 0, 1, ..., n-2)推出

$$2c_i + 6h_i d_i - 2c_{i+1} = 0$$

设 $m_i = S''_i(x_i) = 2c_i$ ，则

a. $2c_i + 6h_i d_i - 2c_{i+1} = 0$ 可写为：

$m_i + 6h_i d_i - m_{i+1} = 0$ ，推出

$$d_i = \frac{m_{i+1} - m_i}{6h_i}$$

b. 将ci, di带入 $y_i + h_i b_i + h_i^2 c_i + h_i^3 d_i = y_{i+1}$ 可得：

$$b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{2} m_i - \frac{h_i}{6}(m_{i+1} - m_i)$$

c. 将bi, ci, di带入 $b_i + 2h_i c_i + 3h_i^2 d_i = b_{i+1}$ (i = 0, 1, ..., n-2)可得：

$$h_i m_i + 2(h_i + h_{i+1}) m_{i+1} + h_{i+1} m_{i+2} = 6\left[\frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_i}\right]$$

端点条件

由i的取值范围可知，共有n-1个公式， 但却有n+1个未知量m 。要想求解该方程组，还需另外两个式子。所以需要对两端点x0和xn的微分加些限制。 选择不是唯一的，3种比较常用的限制如下。

a. 自由边界(Natural)

首尾两端没有受到任何让它们弯曲的力，即$S'' = 0$ 。具体表示为$m_0 = 0$ 和 $m_n = 0$

则要求解的方程组可写为：

$$
\begin{bmatrix}
1 & 0 & 0 & & \cdots & 0 \\
h_0 & 2(h_0+h_1) & h_1 & 0 & & \\
0 & h_1 & 2(h_1+h_2) & h_2 & 0 & \\
0 & 0 & h_2 & 2(h_2+h_3) & h_3 & \vdots \\
\vdots & & & \ddots & \ddots & \ddots \\
 & & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\
0 & \cdots & & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_n \end{bmatrix}
= 6 \begin{bmatrix}
0 \\
\frac{y_2-y_1}{h_1} - \frac{y_1-y_0}{h_0} \\
\frac{y_3-y_2}{h_2} - \frac{y_2-y_1}{h_1} \\
\frac{y_4-y_3}{h_3} - \frac{y_3-y_2}{h_2} \\
\vdots \\
\frac{y_n-y_{n-1}}{h_{n-1}} - \frac{y_{n-1}-y_{n-2}}{h_{n-2}} \\
0
\end{bmatrix}
$$

b. 固定边界(Clamped)

首尾两端点的微分值是被指定的，这里分别定为A和B。则可以推出

$$
\begin{aligned}
S_0'(x_0) = A \quad &\Longrightarrow \quad b_0 = A \\
&\Longrightarrow \quad A = \frac{y_1-y_0}{h_0} - \frac{h_0}{2}m_0 - \frac{h_0}{6}(m_1-m_0) \\
&\Longrightarrow \quad 2h_0m_0 + h_0m_1 = 6\left[\frac{y_1-y_0}{h_0} - A\right]
\end{aligned}
$$

$$
S_{n-1}'(x_n) = B \quad \Longrightarrow \quad b_{n-1} = B
$$

$$
\Longrightarrow \quad h_{n-1}m_{n-1} + 2h_{n-1}m_n = 6\left[B - \frac{y_n-y_{n-1}}{h_{n-1}}\right]
$$

将上述两个公式带入方程组，新的方程组左侧为

$$
\begin{bmatrix}
2h_0 & h_0 & 0 & \cdots & \cdots & 0 \\
h_0 & 2(h_0+h_1) & h_1 & 0 & & \vdots \\
0 & h_1 & 2(h_1+h_2) & h_2 & 0 & \vdots \\
\vdots & 0 & \ddots & \ddots & \ddots & 0 \\
0 & \cdots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\
0 & \cdots & \cdots & 0 & h_{n-1} & 2h_{n-1}
\end{bmatrix}
$$

c. 非节点边界(Not-A-Knot)

指定样条曲线的三次微分匹配，即

$$S_0'''(x_1) = S_1'''(x_1)$$

$$S_{n-2}'''(x_{n-1}) = S_{n-1}'''(x_{n-1})$$

根据$S_i'''(x) = 6d_i$ 和$d_i = \frac{m_{i+1}-m_i}{6h_i}$ ，则上述条件变为

$$h_1(m_1-m_0) = h_0(m_2-m_1)$$

$$h_{n-1}(m_{n-1} - m_{n-2}) = h_{n-2}(m_n - m_{n-1})$$

新的方程组系数矩阵可写为：

$$\begin{bmatrix} -h_1 & h_0 + h_1 & -h_0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & 0 & & \vdots \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & \cdots & \cdots & -h_{n-1} & h_{n-2} + h_{n-1} & -h_{n-2} \end{bmatrix}$$

3.3 算法总结

假定有n+1个数据节点

$$(x_0, y_0), \ (x_1, y_1), \ (x_2, y_2), \ldots, (x_n, y_n)$$

a. 计算步长 $h_i = x_{i+1} - x_i$ (i = 0, 1, ..., n-1)

b. 将数据节点和指定的首位端点条件带入矩阵方程

c. 解矩阵方程，求得二次微分值m_i。

d. 计算样条曲线的系数：

$$a_i = y_i$$

$$b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{2} m_i - \frac{h_i}{6}(m_{i+1} - m_i)$$

$$c_i = \frac{m_i}{2}$$

$$d_i = \frac{m_{i+1} - m_i}{6h_i}$$

其中i = 0, 1, ..., n-1

e. 在每个子区间$x_i \le x \le x_{i+1}$ 中，创建方程

$$g_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

```
class Spline:
    u"""
    Cubic Spline class
    """

    def __init__(self, x, y):
        self.b, self.c, self.d, self.w = [], [], [], []

        self.x = x
        self.y = y

        self.nx = len(x)  # dimension of x
        h = np.diff(x)    #dx

        # calc coefficient c
        self.a = [iy for iy in y]

        # calc coefficient c
```

```
        A = self.__calc_A(h)
        B = self.__calc_B(h)
        self.c = np.linalg.solve(A, B)
        #  print(self.c1)

        # calc spline coefficient b and d
        for i in range(self.nx - 1):
            self.d.append((self.c[i + 1] - self.c[i]) / (3.0 * h[i]))
            tb = (self.a[i + 1] - self.a[i]) / h[i] - h[i] * \
                (self.c[i + 1] + 2.0 * self.c[i]) / 3.0
            self.b.append(tb)

    def calc(self, t):
        u"""
        Calc position
        if t is outside of the input x, return None
        """

        if t < self.x[0]:
            return None
        elif t > self.x[-1]:
            return None

        i = self.__search_index(t)         #找的t属于s坐标系的哪一段
        dx = t - self.x[i]
        result = self.a[i] + self.b[i] * dx + \
            self.c[i] * dx ** 2.0 + self.d[i] * dx ** 3.0
        return result


    def __search_index(self, x):
        u"""
        search data segment index
        """
        idx = bisect.bisect(self.x, x)  #二分查找数值将会插入的位置并返回，而不会插入
        result = idx - 1
        return result

    def __calc_A(self, h):
        u"""
        calc matrix A for spline coefficient c
        """
        A = np.zeros((self.nx, self.nx))
        A[0, 0] = 1.0
        for i in range(self.nx - 1):
            if i != (self.nx - 2):
                A[i + 1, i + 1] = 2.0 * (h[i] + h[i + 1])
            A[i + 1, i] = h[i]
            A[i, i + 1] = h[i]

        A[0, 1] = 0.0
        A[self.nx - 1, self.nx - 2] = 0.0
        A[self.nx - 1, self.nx - 1] = 1.0
        #  print(A)
        return A

    def __calc_B(self, h):
        u"""
        calc matrix B for spline coefficient c
        """
        B = np.zeros(self.nx)
        for i in range(self.nx - 2):
            B[i + 1] = 3.0 * (self.a[i + 2] - self.a[i + 1]) / \
                h[i + 1] - 3.0 * (self.a[i + 1] - self.a[i]) / h[i]
        #  print(B)
        return B

class Spline2D:
    u"""
    2D Cubic Spline class
```

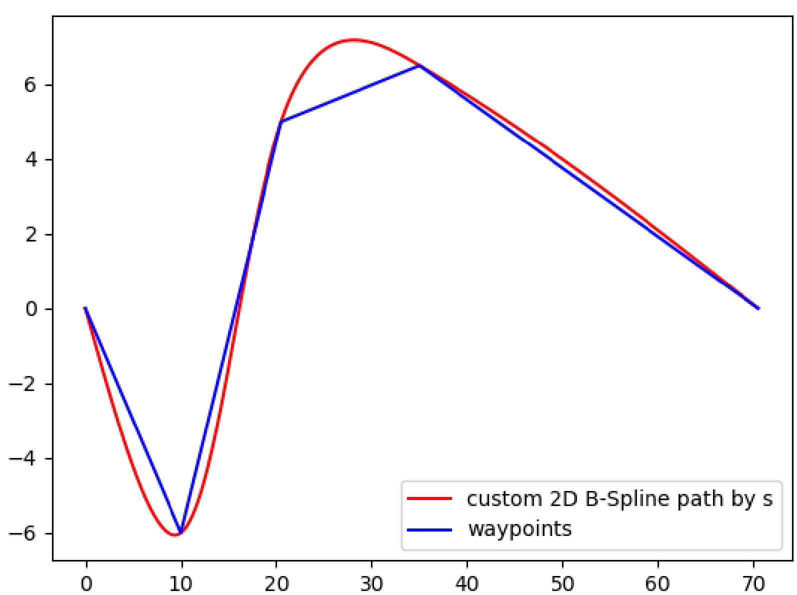
```
    """

    def __init__(self, x, y):         #构造函数
        self.s = self.__calc_s(x, y)
        #self.s=[i for i in range(len(x))]
        #print(self.s)
        self.sx = Spline(self.s, x)    #对（s,x）进行三次样条插值
        #print(self.sx)
        self.sy = Spline(self.s, y)   #对（s,y）进行三次样条插值
        #print(self.sy)

    def __calc_s(self, x, y):      #计算距离s值
        dx = np.diff(x)                            #[2.5, 2.5, 2.5, 2.5, -4.5,
-4.]
        dy = np.diff(y)                            #[-6.7, 11., 1.5, -6.5, 5.,
-7.]
        self.ds = [math.sqrt(idx ** 2 + idy ** 2)  #ds
                   for (idx, idy) in zip(dx, dy)]
        s = [0]
        s.extend(np.cumsum(self.ds))
        return s

    def calc_position(self, s):   #通过s值计算xy
        u"""
        calc position
        """
        x = self.sx.calc(s)
        y = self.sy.calc(s)
        return x, y

def main():
    x = [0.0, 10.0, 20.5, 35.0, 70.5]
    y = [0.0, -6.0, 5.0, 6.5, 0.0]
    tx, ty, csp = generate_target_course(x, y)
    flg, ax = p.subplots(1)
    p.plot(tx,ty,color='red',label="custom 2D B-Spline path by s")
    p.plot(x,y,color='blue',label="waypoints")
    p.legend()
    p.show()
```



参考：https://www.cnblogs.com/xpvincent/archive/2013/01/26/2878092.html