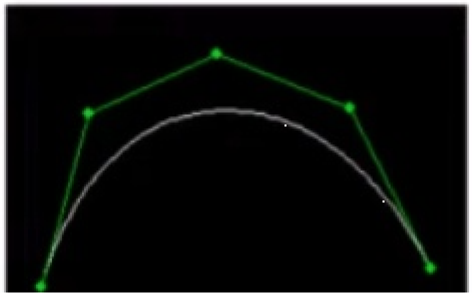
贝塞尔曲线

本文参考这篇关于贝塞尔曲线与曲面的文章，讲的十分全面，值得推荐！

曲线有许多表示方法，如显示表示、隐式表示、参数形式表示等，贝塞尔提出了一种通过连接向量来表示曲线的方法，如下：



即在画曲线前先通过向量绘制一个多边形，代表该曲线的趋势和走向。并且贝塞尔曲线具有交互性，也就是说我们可以通过修改向量来修改曲线。

贝塞尔提出了如下公式用于计算曲线，将曲线表达成向量和基函数的乘积。

> $V(t) = \sum_{i=0}^{n} f_{i,n}(t) A_i$

其中 A_i 代表的就是向量，$f_{i,n}(t)$ 代表的是一个基函数，其内容如下：

> $$f_{i,n}(t) = \begin{cases} 1 & i = 0 \\ \frac{(-t)^i}{(i-1)!} \frac{d^{i-1}}{dt^{i-1}} \frac{(1-t)^{n-1}-1}{t} & \end{cases}$$
>
> 基函数本质上就是一个 $n-1$ 的多项式

有了这个公式后，就可以通过一个多边形计算出一个曲线。从公式中可以发现，**贝塞尔曲线属于一种参数形式（有参数t）表示曲线的方式。**以上两个公式了解就好，不需要去理解。

1. 伯恩斯坦多项式

有关伯恩斯坦多项式的介绍可以参考这篇文章，其中包括很多性质的推导。

原本贝塞尔曲线说的是向量相连，现在我们把这些向量都变成一个个控制点，即给定控制点 $P_0, P_1, P_2, \ldots P_n$，贝塞尔曲线上的任意一点 $P(t)$ 可以定义为：

$$P(t) = \sum_{i=0}^{n} P_i B_i^n(t) \tag{1}$$

其中的 $B_i^n(t)$ 就是第 i 个 n 阶的伯恩斯坦多项式：

$$B_i^n(t) = C_n^i t^i (1-t)^{n-i} \tag{2}$$

其中 C_n^i 为组合数（排列组合），值为：

$$C_n^i = \frac{n!}{i!(n-i)!} \tag{3}$$

因为组合数有很多性质，所以贝塞尔曲线的很多算法和定理都在这个组合数上做手脚。

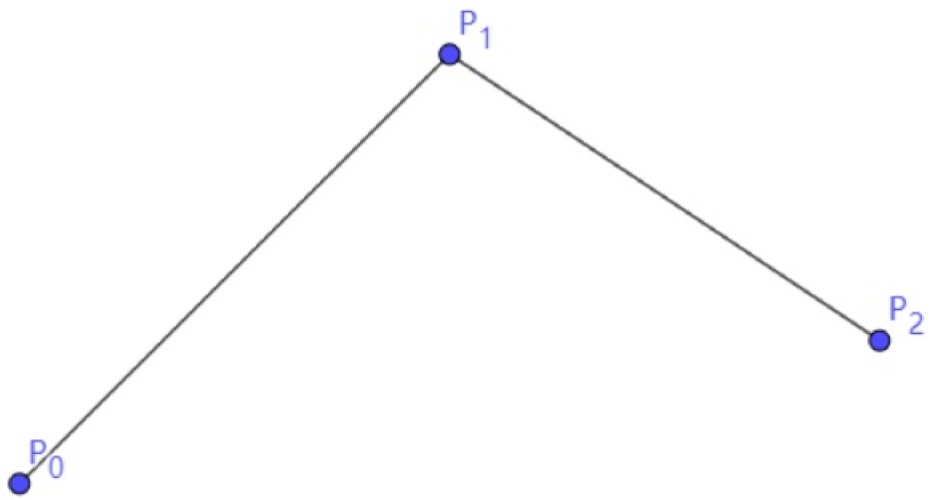
伯恩斯坦多项式代码如下：

```
# 阶乘
def factorial(n):
    if n == 0:
        return 1
    res = 1
    for i in range(1, n+1):
        res *= i
    return res

# 组合数
def combinatorial_num(n, i):
    return factorial(n) / (factorial(i) * factorial(n-i))

# 伯恩斯坦多项式
def bernstein_polynomial(n, i, t):  # n代表阶数
    return combinatorial_num(n, i) * pow(t, i) * pow((1.0-t), (n-i))
```

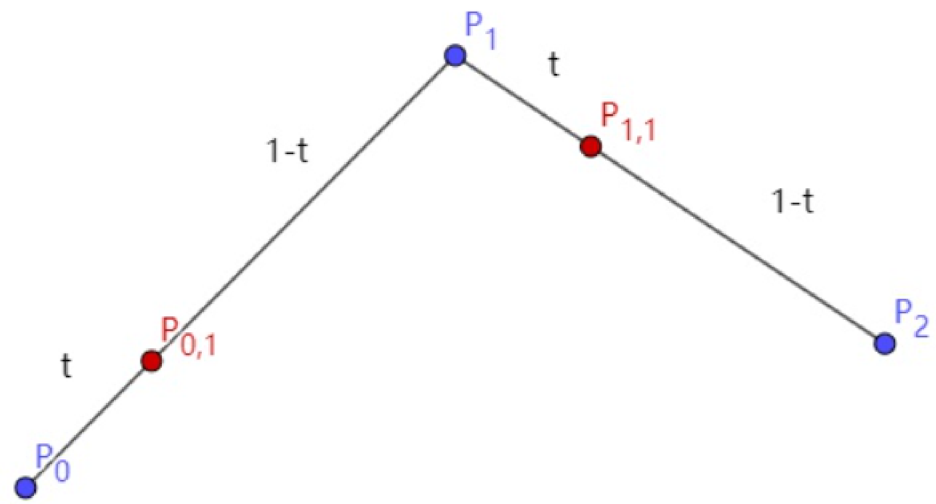
2.德卡斯特里奥算法（de Casteljau Algorithm）

但是实际上，我们并不会使用上面那个算法来求曲线上的任意点 $P(t)$，而是使用一种名为de Casteljau的递归算法来求。该算法比之前的方法慢，但在数值上更为稳定。

接下来我们来看看用de Casteljau算法怎么求出曲线上的一点的。如下图，我们先随便选取三个控制点，并将它们前后连线。

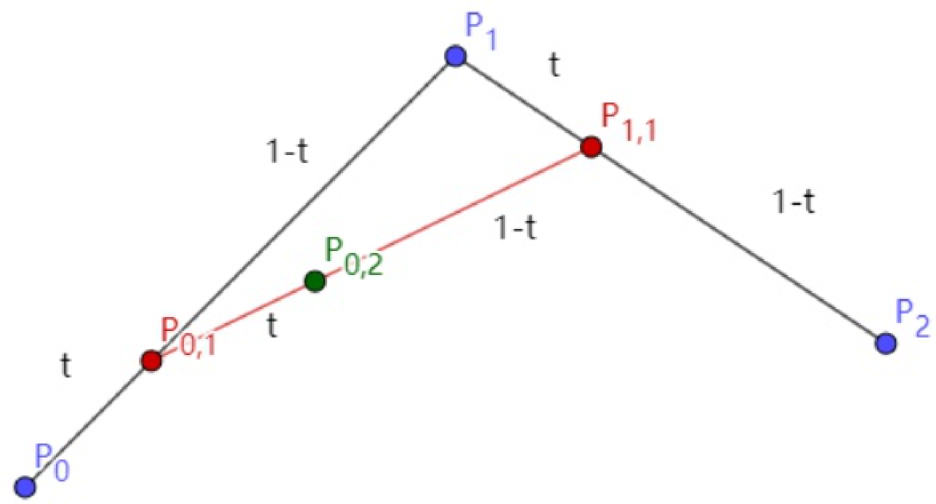


然后我们在 P_0P_1 线段上通过**线性插值**找到点 $P_{0,1}$，使得 $P_0P_{0,1} : P_{0,1}P_1 = t : 1-t$，其他线段上也如此，我们就可得到下图：



第一次做线性插值

接着我们连接 $P_{0,1}P_{1,1}$，得到新的线段，然后在该线段上再取一点使得该线段被分为 t 和 1-t，得到下图：

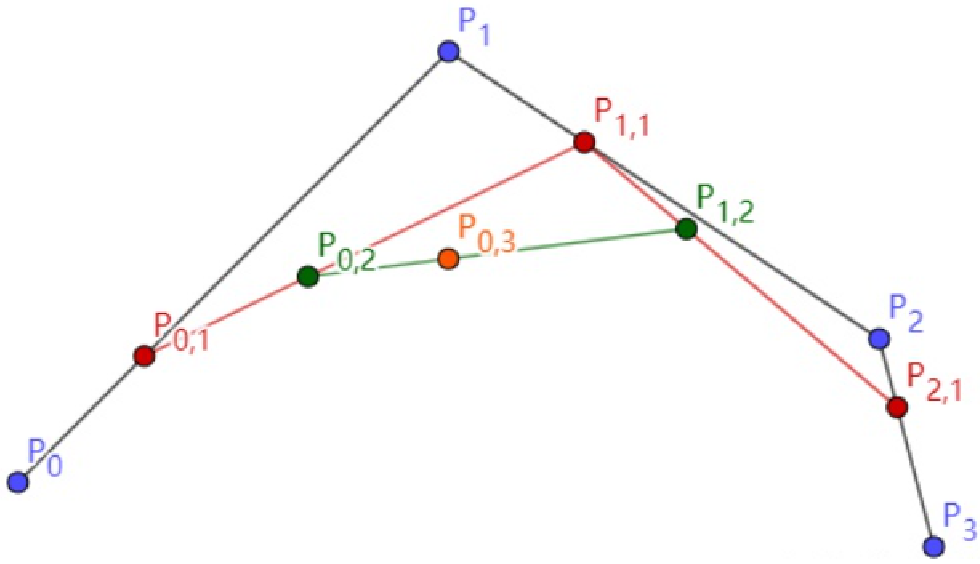


第二次做线性插值

此时已经不能再连线了，而我们得到的点 $P_{0,2}$ 就是这三个控制点对应的贝塞尔曲线在 t 位置上的点。

这样我们就可以通过使 t 从 0 变到 1，得到所有曲线上的点，从而得到曲线。这样求曲线上任意一点方式也就是前面所说的de Casteljau算法。

如果有更多的控制点，我们也可以使用相同的方法来求出曲线上的一点，如下图是四个控制点求曲线上一点的过程：

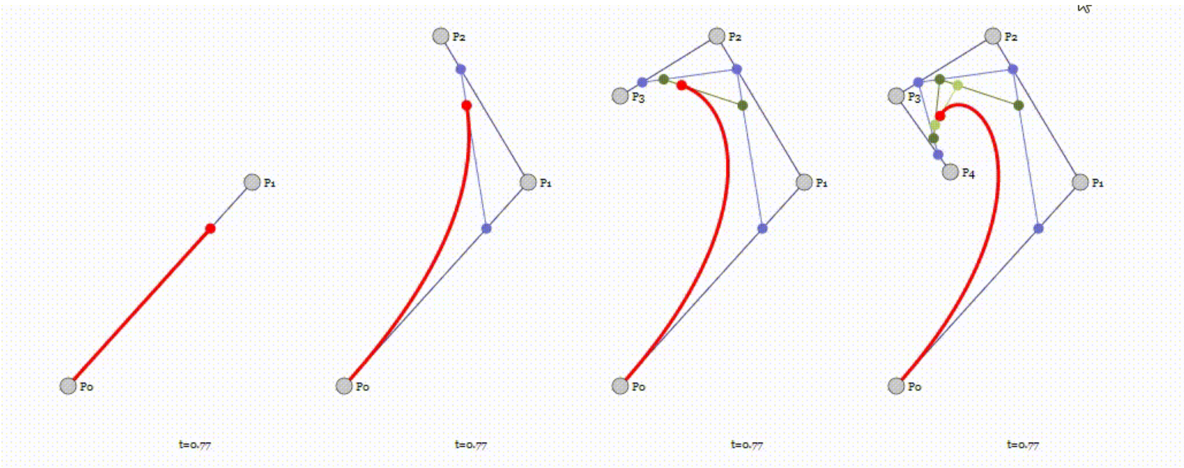


画三阶贝塞尔曲线上的一点

对于三个顶点控制的贝塞尔曲线我们称之为**二阶贝塞尔曲线（Quadratic Bezier）**，那么四个顶点自然是**三阶贝塞尔曲线（Cubic Bezier）**，因此 **n 阶贝塞尔曲线有 n+1 个顶点。**

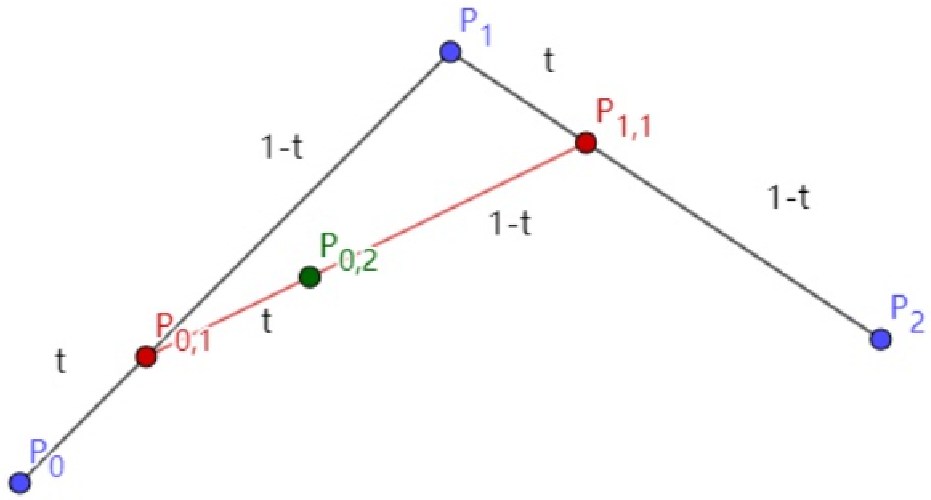
而de Casteljau算法则是把这 n+1 个点，通过参数 t 用线性插值的方法，先变成 n 个新的点，然后在变成 n-1 个新的点，递归下去，最终得到一个点，这个点就是贝塞尔曲线上的点。

如下动图，很直接明了的表示了由二阶到四阶贝塞尔曲线的绘制过程：



3. 贝塞尔曲线

那伯恩斯坦多项式和de Casteljau算法有什么关系呢？我们拿最简单的二阶贝塞尔曲线举例，如下图：



图中蓝色的点为控制点，它们的坐标我们是知道的。那么通过线性插值，我么可以得到红色点的坐标，公式如下（$P_{x,y}$ 用 P_x^y 替代）：

$$P_0^1 = (1-t)P_0 + tP_1 \tag{4}$$

$$P_1^1 = (1-t)P_1 + tP_2 \tag{5}$$

红色点坐标求出后，我们自然可以再求出绿色点的坐标：

$$P_0^2 = (1-t)P_0^1 + tP_1^1 \tag{6}$$

把式 (4) (5) 代入到式 (6) 中，可以得到：

$$P_0^2 = (1-t)((1-t)P_0 + tP_1) + t((1-t)P_1 + tP_2) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \tag{7}$$

我们还可以通过这个方法计算三阶、四阶、乃至 n 阶的贝塞尔曲线，得到的结果为曲线上的任意一点 $P(t)$,是各个顶点的线性组合，即：

$$P(t) = k_0 P_0 + k_1 P_1 + k_2 P_2 + \ldots + k_n P_n \tag{8}$$

此式中**每个顶点前面的系数 k，就是伯恩斯坦多项式。**例如二阶贝塞尔曲线对应的伯恩斯坦多项式为 $B_i^2(t)$，其中 $B_0^2(t) = (1-t)^2$，$B_1^2(t) = 2t(1-t)$，$B_2^2(t) = t^2$，正好对应式 (7) 中前面的三个系数。

由此可以得出结论，**对于 n 阶的贝塞尔曲线，曲线 t 位置上点 P(t) 的坐标是由 n+1 个顶点与伯恩斯坦多项式的乘积求和**：

$$P(t) = B_0^n(t)P_0 + B_1^n(t)P_1 + B_2^n(t)P_2 + \ldots + B_n^n(t)P_n = \sum_{i=0}^{n} P_i B_i^n(t) \tag{9}$$

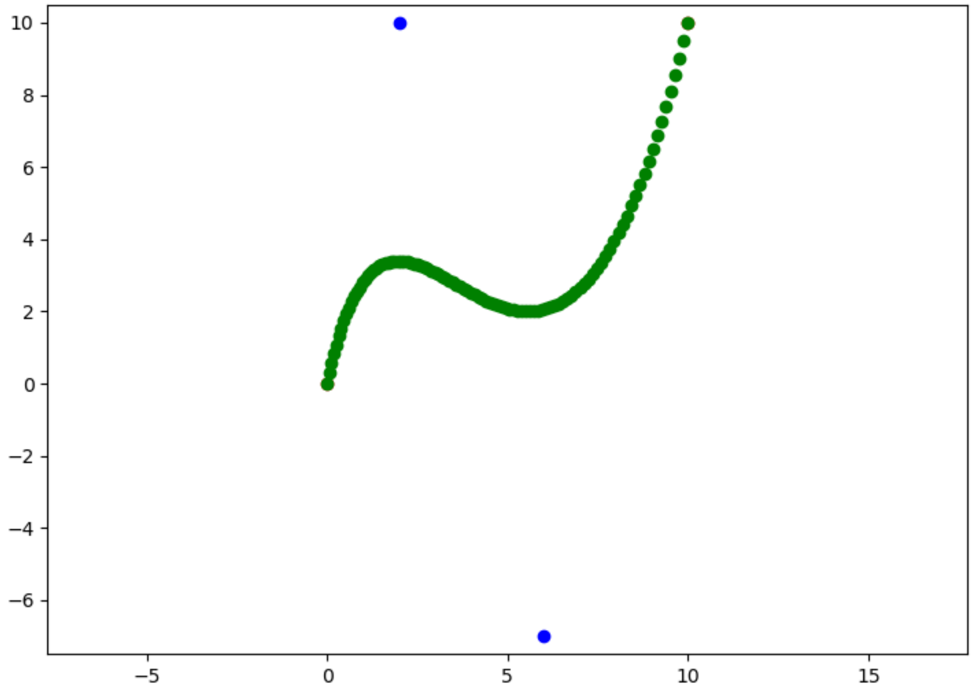
知道了公式后，代码就很简单了：

```python
import numpy as np

def bezier_curve(point_list):
    n = len(point_list) - 1     # order
    t = np.linspace(0, 1, NUM)
    x = 0.0
    y = 0.0
    for i in range(n + 1):
        x += bernstein_polynomial(n, i, t) * point_list[i][0]
        y += bernstein_polynomial(n, i, t) * point_list[i][1]
    return [x, y]
```

简单绘制一条三阶贝塞尔曲线:

```python
import matplotlib.pyplot as plt

p_start1 = [0, 0]
p_0 = [2, 10]
p_1 = [6, -7]
p_end1 = [10, 10]
p_list = [p_start1, p_0, p_1, p_end1]

x, y = bezier_curve(p_list)
res = []
for i in range(NUM):
    res.append([round(x[i], 2), round(y[i], 2)])

plt.subplot(121)
plt.plot(p_start1[0], p_start1[1], 'or')
plt.plot(p_0[0], p_0[1], 'ob')
plt.plot(p_1[0], p_1[1], 'ob')
plt.plot(p_end1[0], p_end1[1], 'or')

plt.plot(x, y, 'og')
plt.axis('equal')
```

曲线形状如下:



三阶贝塞尔曲线