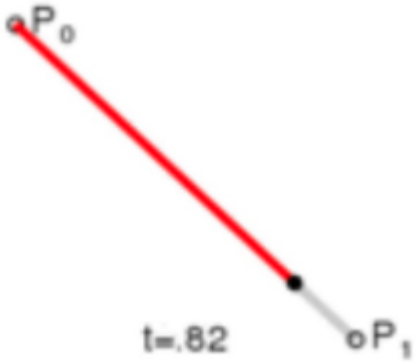
Spline Fusion

Chapter I: 贝塞尔曲线

如果我们想记录某个物体的踪迹，当前最通用的办法是在某个时刻点通过某种方法计算出该物体的位姿，并且用这个值去代表当前整个时间段。当这个时间段足够小的时候，所有人就“假装”它是连续的——你不可能求出它的每个时刻的位姿。这是一种无奈的妥协，但是我们就真的对此束手无策了吗？如果让我来解决这个问题，我会自然而然的想到“参数化”——我并不需要知道这条曲线上的每个点是多少，我只需要知道这条曲线的参数就够了。举个简单的例子，你并不需要知道$y = ax^2$上的每个点，你只需要知道a是多少就能画出整条曲线。事实上，1962年的贝塞尔也是这么想的。为了搞清楚原研哉到底怎么挣得200万(小米新logo由贝塞尔曲线而成)，我决定从更好理解的平面B样条开始。注: 第一章和第二章内容多来自知乎 https://zhuanlan.zhihu.com/p/139759835

1. 贝塞尔曲线的由来

假如我们此刻有两个离散的点，如果要对他进行插值的话我们最先想到的肯定是线性插值



假设$B_0(t)$就是我们要求的点，那这个曲线的方程可表示为:

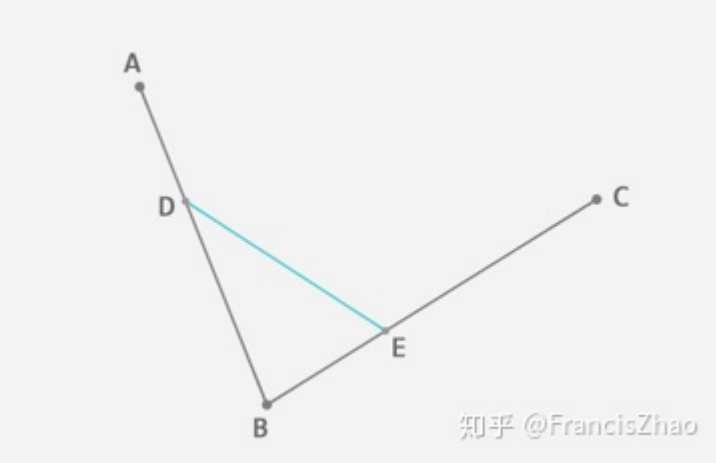
$$B_0(t) = P_0t + (P_1 - P_0)t$$

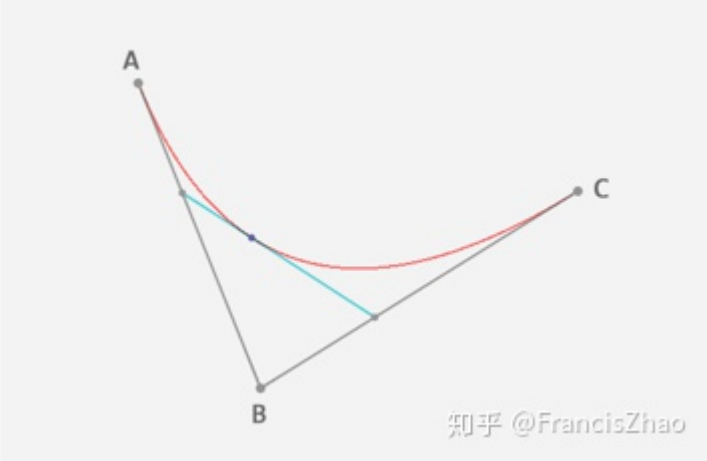
也就是

$$B_0(t) = (1-t)P_0 + tP_1, t \in [0,1]$$

其中，P_0, P_1是点的坐标$[x, y]^T$, $B_0(t)$是以t为参数的方程。当t_0为我们想要的比例的时候，$B_0(t_0)$就是我们要求的点。

当我们需要的是一条二次曲线的时候，这时我们需要三个离散点来生成这条曲线





虽然我们没有办法直接找出插值点，但是我们可以通过对AB和BC分别线性插值的办法，来达到降次的效果。这个时候，在DE上找出对应的点，就跟我们之前一样了。转化为数学表达式:

$$B_{0,0}(t) = (1-t)P_0 + tP_1$$
$$B_{1,0}(t) = (1-t)P_1 + tP_2$$

注意，我们这里B多了一个下标。第一个下标代表的是第几条线段，AB和BC分别代表0和1;第二个下标代表的是线段次数，在这里，AB和BC都是0，DE是1。理解这一点对后面B-Spline非常重要！这个时候我们的问题又转化为之前线性插值，我们对蓝色线段进行之前的操作:

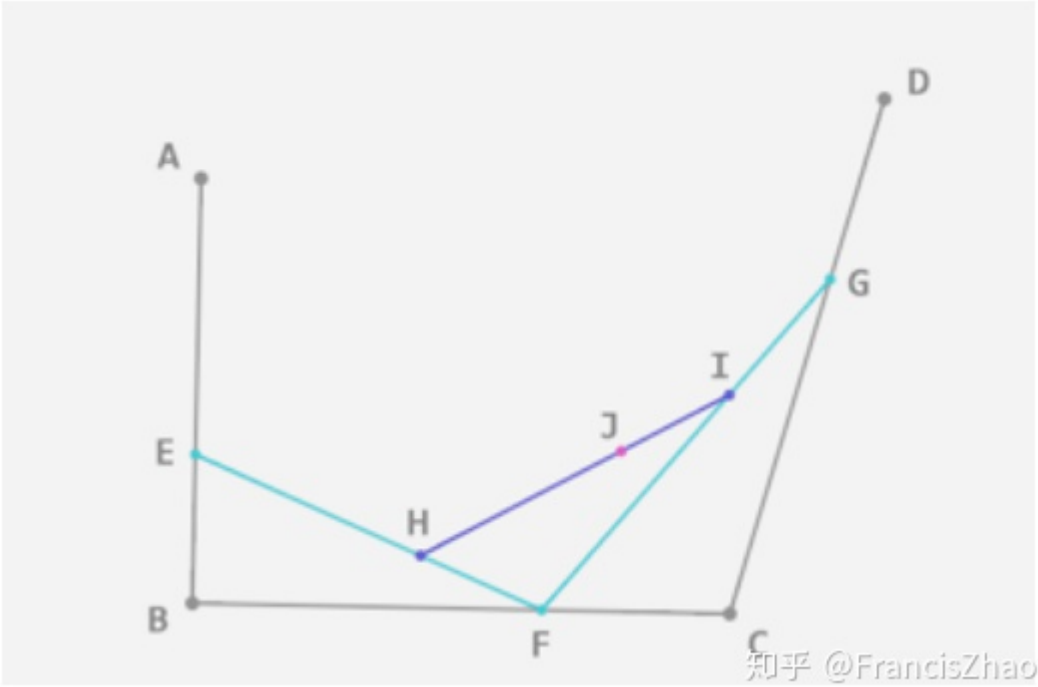
$$B_{0,1} = (1-t)B_{0,0} + tB_{1,0}$$

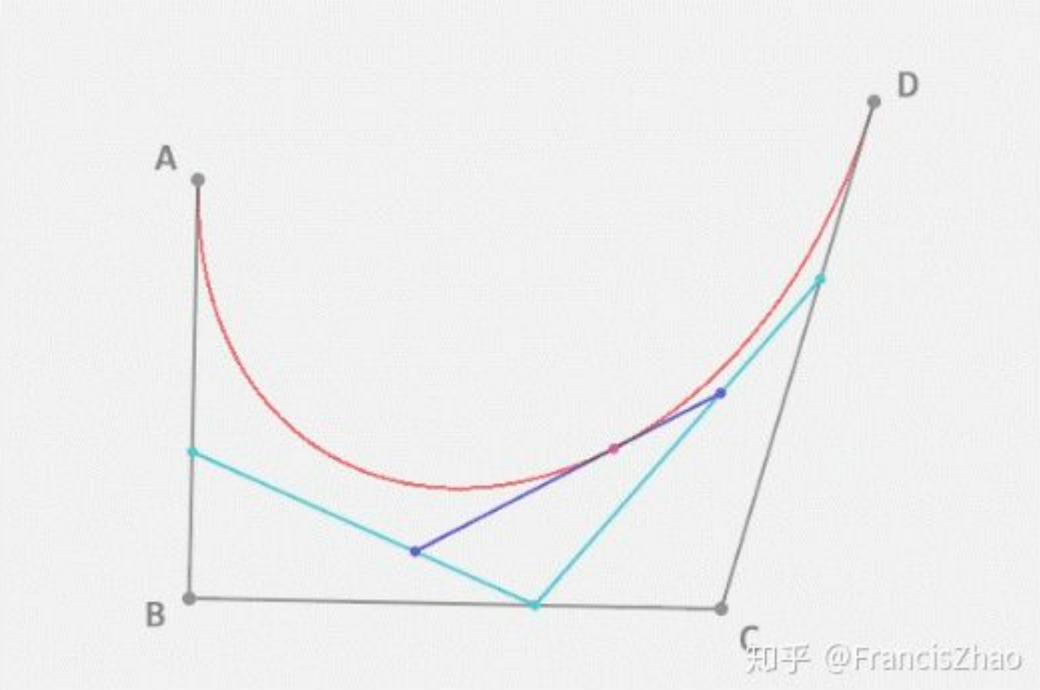
代入$B_{0,0}$和$B_{1,0}$整理得到，

$$B_{0,1} = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

这就是红色曲线的表达式。

更高阶的贝塞尔曲线虽然形式上更复杂，但是操作上异曲同工，只不过多了几次降次的过程，此处便不再赘述。





至此，我们从直观上理解了贝塞尔曲线的由来。总结:

- 贝塞尔曲线能够达到用少量控制点绘制一条曲线的效果。
- 当阶次足够的时候，贝塞尔曲线足够光滑，并且可以多次求导。
- 贝塞尔曲线并不通过所有控制点。

但是，贝塞尔曲线并不完美，它有以下缺点:

- 阶次=控制点个数-1，两者完全绑定。
- 改变一个控制点，整条曲线会被改动。

为了解决以上痛点，B样条曲线应运而生。

Chapter II: B样条曲线

B样条可被理解为多段贝塞尔曲线的拼接。贝塞尔曲线时期，我们要找的比例t就在[0,1]区间上，很好表达。但是B样条时期，在拼接的时候我们要表达各个区间的联系，我们需要引入**节点向量(knots vector)**的概念。它由一个个非递减的**节点(knots)**组成，例如[0,0,25,0.75,1.0]。假如他们之间的间隔是固定的，那就被成为均匀(uniform)B样条曲线。在参考论文和本报告中，大部分讨论的都是均匀B样条曲线。

在贝塞尔曲线中，曲线是通过一次次的线性插值来定义。但是在B样条里，曲线是由各个基函数(basic funcion)和控制点相乘之后的累加来定义的:

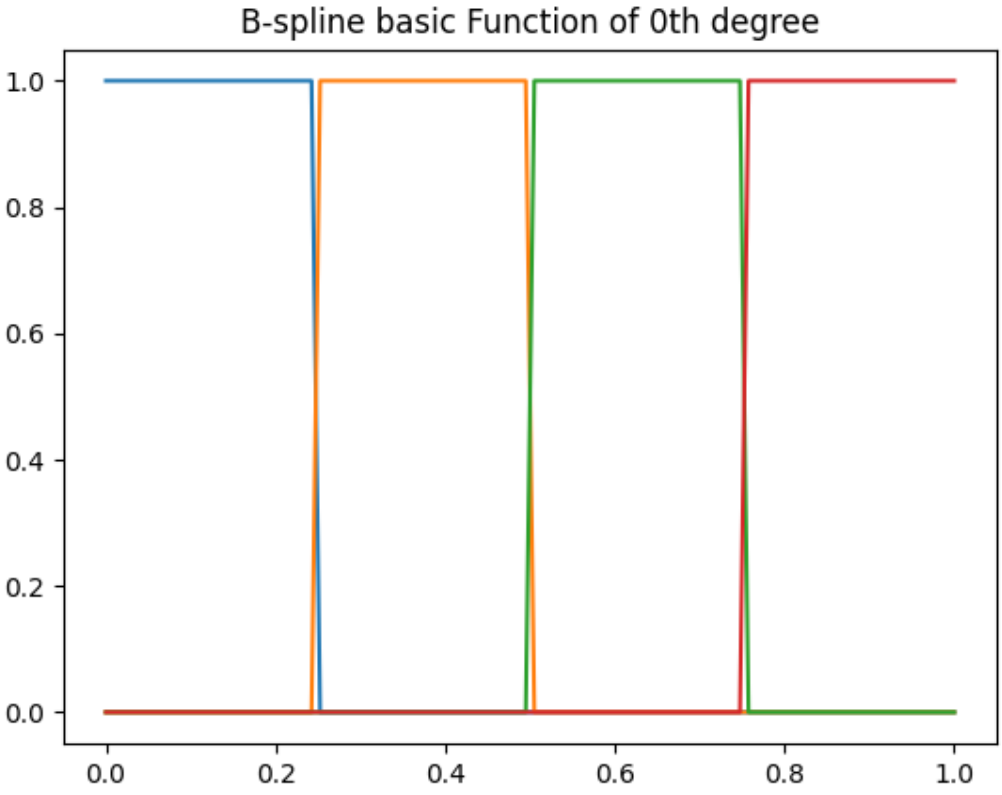
$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i$$

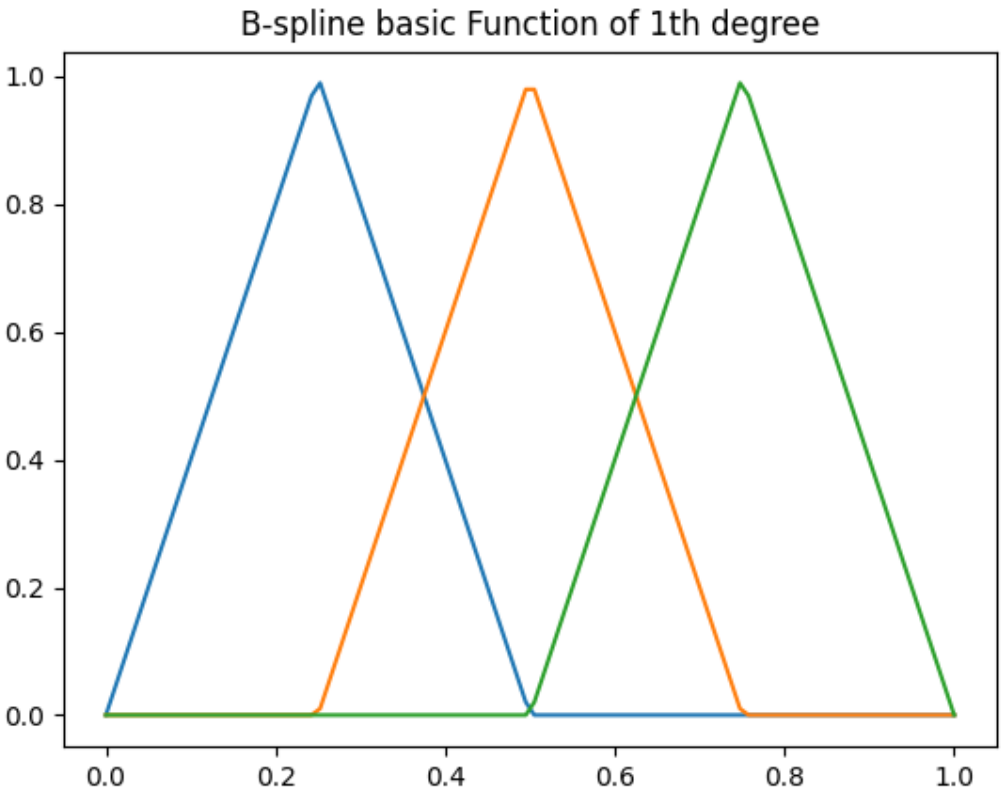
如果你对我之前写的东西还有印象，会发现这个式子和贝塞尔曲线的式子有异曲同工之妙——这也很正常，毕竟B样条是由此脱胎而来。很大的不同是，控制点之前的系数被当成了参数化的函数来处理，即基函数。根据cox De boor公式，基函数被定义为:

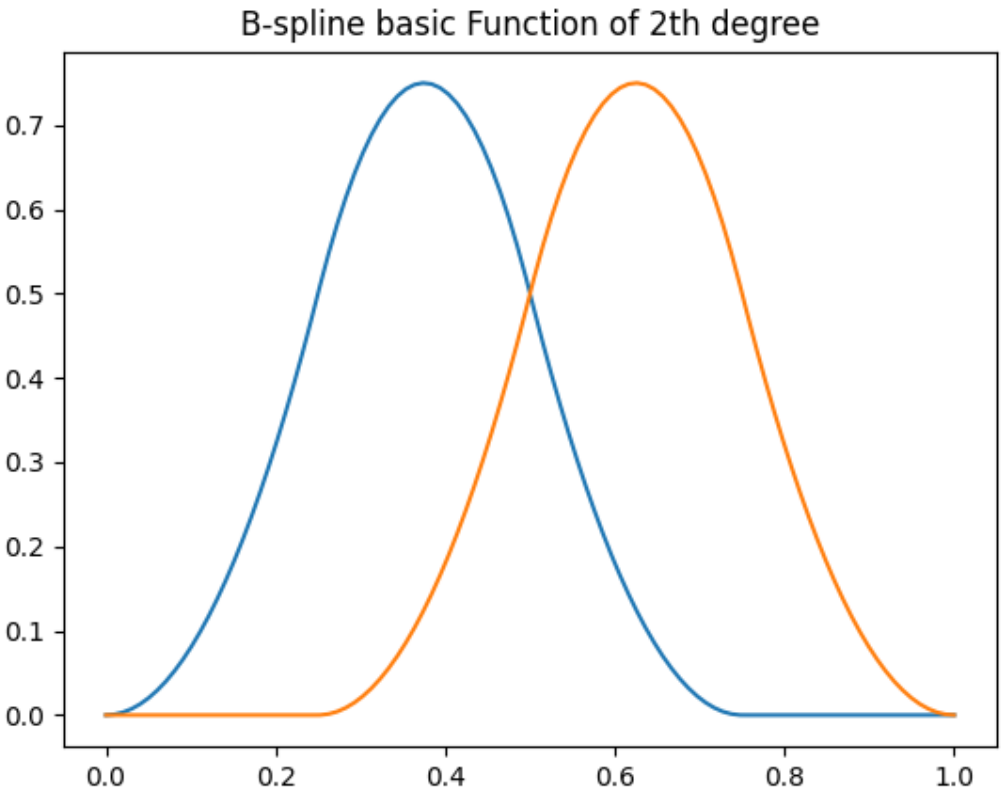
$$N_{i,0}(u) = \begin{cases} 1 & if\ \ u_i < u <= u_{i+1} \\ 0 & otherwise \end{cases}$$

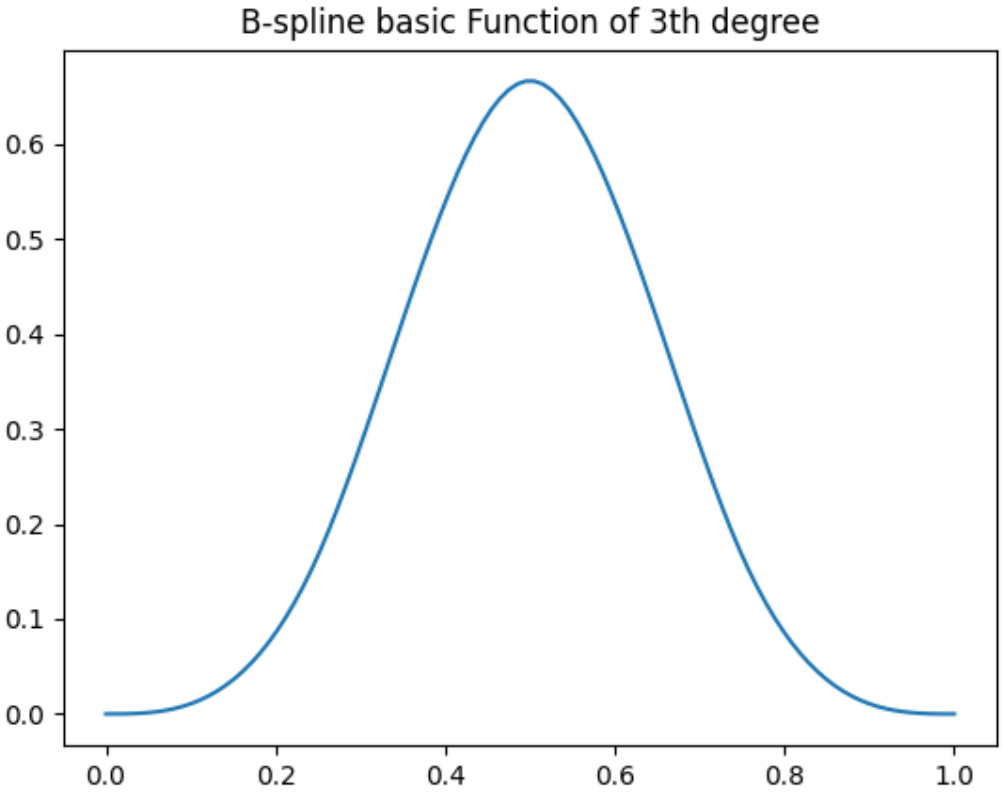
$$N_{i,p}(u) = \frac{u - u_i}{u_{i+1} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

这是一个递归定义的公式。也许有些许难理解，让我上个图:




B-spline basic Function of 1th degree
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


B-spline basic Function of 2th degree
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8
1.0

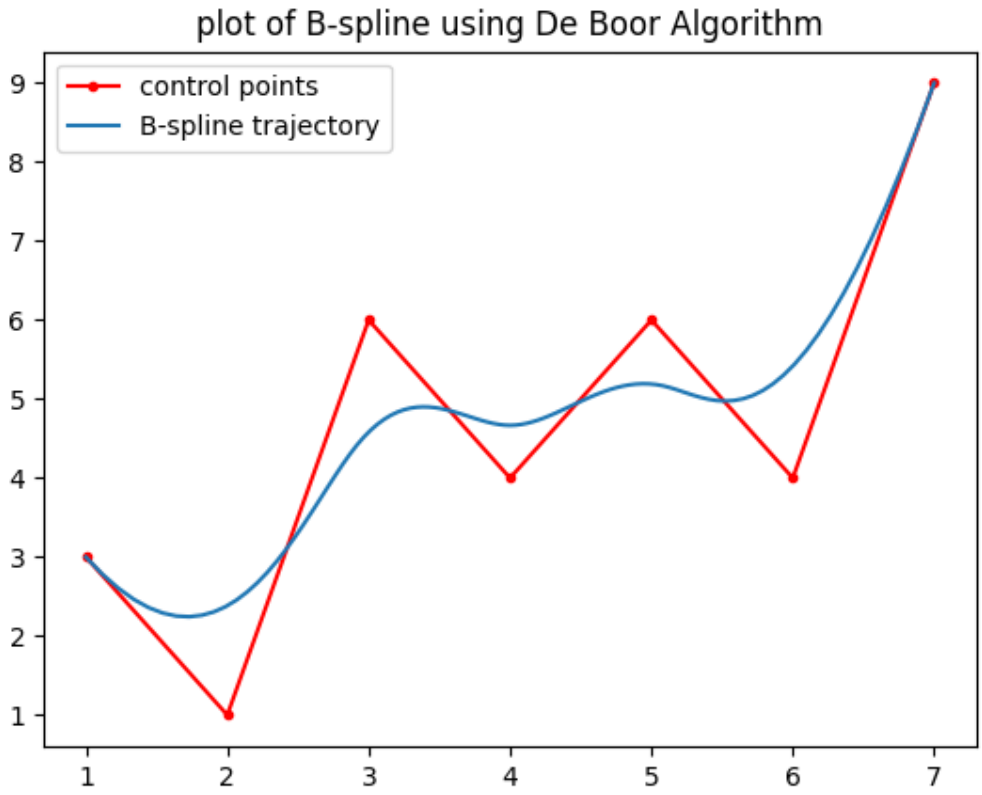

B-spline basic Function of 3th degree
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8
1.0

蓝橙绿红分别代表$N_{i,p}$中的i=0,1,2,3时的函数

从图上我们可以看到基函数的性质

- 随着阶次p的上升，基函数的非0区间在扩大。当p=0，非0区域只有0.25，此后每上升1，非0区域会上升0.25。由于基函数是直接跟控制点相乘的，这也意味着当阶次上升时，每个控制点的就会对更远的曲线产生影响。这个对应了贝塞尔曲线的第一个痛点，我们是可以选择我们的阶次p，多余的控制点对该段曲线的影响会被直接切断。
- 当p确定的时候，在$[u_i, u_{i+1})$区间内最多有p+1个基函数不为0。还是由于基函数是直接跟控制点相乘的，所以最多只有p+1个控制点对曲线产生影响。这解决了贝塞尔曲线的第二个痛点。改变一个控制点，只有部分的曲线会被改动。

初步理解了B样条曲线的由来以及构成后，非常容易产生一个问题。当我的阶次p足够高的时候，我的头，尾两端是没法用p次的B样条表示的——从基函数中看出，阶次p越高，基函数越少了，p=4时我们只能表示$[u_0, u_1)$这个区间。我查了资料，解决的办法也很简单，那就是在节点的首尾重复p-1次。例如，[0,0.25,0.5,0.75,1]变成[0,0,0,0,0.25,0.5,0.75,1,1,1,1]就行了，这种B样条也被成为clamped B-spline。这里给个示例。



可以看到，clamped B-spline必定会经过首，尾控制点。

Chapter III: Cumulative Form

在对平面中的B样条有所了解后，我们便可以把它扩展到李代数中。事实上，这也没那么容易——毕竟在实数域里，实数加实数还是实数。但是在李代数中，我们只能用expm映射到李群上用乘法运算才能保证它的封闭性。

虽然这么麻烦，但是它还是有它的优点，在论文中提到:

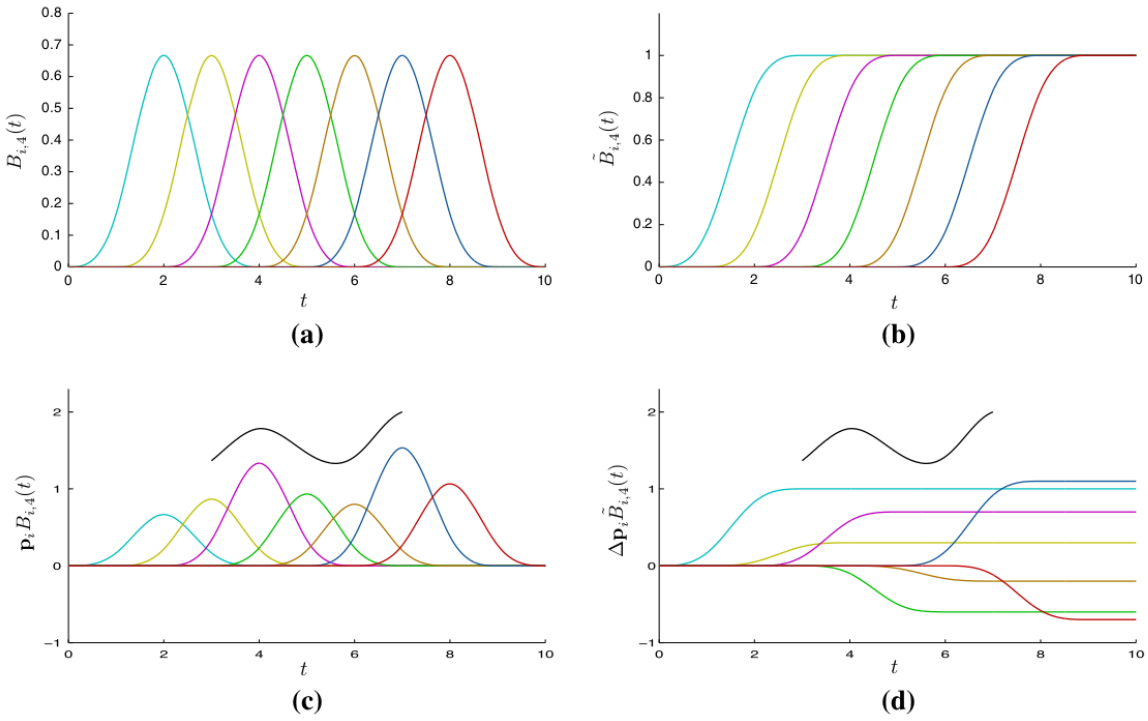
1. Local control, allowing the system to function online as well as in batch. B样条曲线是贝塞尔曲线的拼接，改变某个控制点只会对局部的曲线产生影响。对应的是上一章中提到的基函数的第二个性质。
2. C_2 continuity, to enable inertial predictions. 这是我在上一章没有提到的点。如果我们用B样条曲线来代表踪迹，我们可以对它进行求导，毫不费力地得到速度和加速度——这太棒了，几乎和IMU是天生一对。
3. Good approximation of minimal torque trajectories. 在造船或者其他工业中，B样条曲线意味着扭力最小。

4. A parameterization of rigid-body motion devoid of singularities. 不受万向锁影响——在李群上运算。

在我阅读的几乎所有论文里，BSpline-based trajectory几乎都是用cumulative form表示的。至于原因，个人觉得是**求导**。它由Kim于1995年首次提出:

$$\mathbf{p}(t) = \mathbf{p}_0 \tilde{B}_{0,k}(t) + \sum_{i=1}^{n} (\mathbf{p}_i - \mathbf{p}_{i-1}) \tilde{B}_{i,k}(t)$$

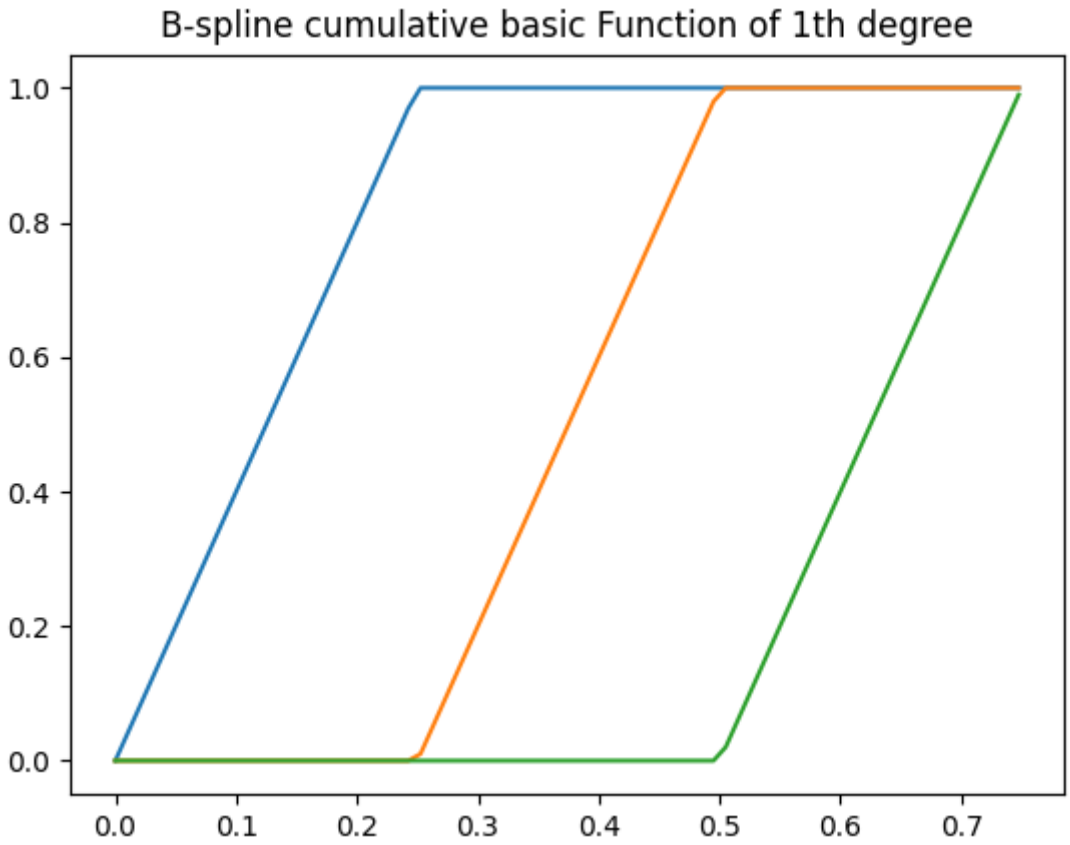
式子中，$\tilde{B}_{i,k}(t) = \sum_{j=i}^{n} B_{j,k}(t)$。 仔细观察式子，发现最大的差别是原来直接用控制点乘以基函数，现在用**控制点之间的距离**乘以新的基函数，这也是cumulative一词的由来。新的基函数和原基函数对比如下图所示:



从(c)和(d)中可以看出，虽然基函数不同，但是最后生成的曲线是一样的。总结一下:

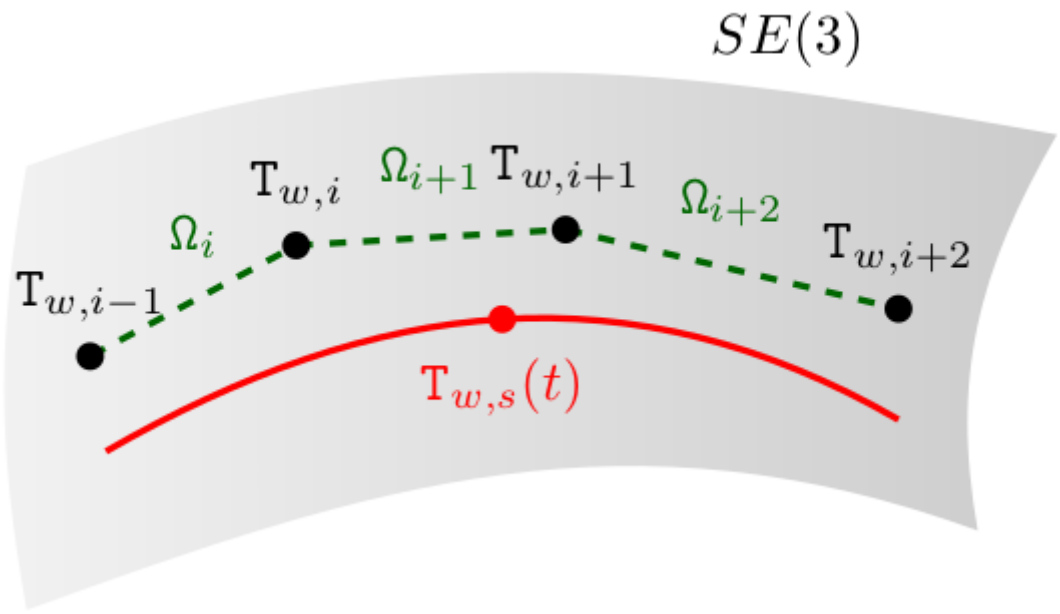
- 原基函数最后会停留在0，现会停留在1。
- 与原基函数比较，新基函数的影响作用范围减少一半(原来非0阶段接近4，现大约2)。

这是我用python自己画的图(p=1):



此时，我们改写一下，把cumulative form从实数域扩展到李代数上，把加法换乘法，减法映射到李群上用矩阵的逆实现，最终得到其在$SE(3)$上的表达:

$$\mathbf{T}_{w,s}(t) = \exp(\tilde{B}_{0,k}\log(\mathbf{T}_{w,0}(t)))\prod_{i=1}^{n}\exp(\tilde{B}_{i,k}\log(\mathbf{T}_{w,i-1}^{-1}\mathbf{T}_{w,i}))$$



SE(3)上利用B样条插值图示

特别的，当我们的B样条是均匀(uniform)且k=4的时候呢，我们可以重写我们的基函数表达式。

$$\tilde{\mathbf{B}}(u) = \mathbf{C}\begin{bmatrix}1\\u\\u^2\\u^3\end{bmatrix}, \quad \dot{\tilde{\mathbf{B}}}(u) = \frac{1}{\Delta t}\mathbf{C}\begin{bmatrix}0\\1\\2u\\3u^2\end{bmatrix},$$

$$\ddot{\tilde{\mathbf{B}}}(u) = \frac{1}{\Delta t^2}\mathbf{C}\begin{bmatrix}0\\0\\2\\6u\end{bmatrix}, \quad \mathbf{C} = \frac{1}{6}\begin{bmatrix}6 & 0 & 0 & 0\\5 & 3 & -3 & 1\\1 & 3 & 3 & -2\\0 & 0 & 0 & 1\end{bmatrix}$$

插值的式子也可以写成:

$$\mathbf{T}_{w,s}(u) = \mathbf{T}_{w,i-1}\prod_{j=1}^{3}\exp\left(\tilde{\mathbf{B}}(u)_j\,\boldsymbol{\Omega}_{i+j}\right),$$

当你想要知道该条曲线的导数，以利用imu的数据进行优化的时候，

$$
\dot{\mathbf{T}}_{w,s}(u) = \mathbf{T}_{w,i-1}\begin{pmatrix}\dot{\mathbf{A}}_0\mathbf{A}_1\mathbf{A}_2 + \\ \mathbf{A}_0\dot{\mathbf{A}}_1\mathbf{A}_2 + \\ \mathbf{A}_0\mathbf{A}_1\dot{\mathbf{A}}_2\end{pmatrix},
$$

$$
\ddot{\mathbf{T}}_{w,s}(u) = \mathbf{T}_{w,i-1}\begin{pmatrix}\ddot{\mathbf{A}}_0\mathbf{A}_1\mathbf{A}_2 + \mathbf{A}_0\ddot{\mathbf{A}}_1\mathbf{A}_2 + \\ \mathbf{A}_0\mathbf{A}_1\ddot{\mathbf{A}}_2 + 2\dot{\mathbf{A}}_0\dot{\mathbf{A}}_1\mathbf{A}_2 + \\ 2\dot{\mathbf{A}}_0\mathbf{A}_1\dot{\mathbf{A}}_2 + 2\mathbf{A}_0\dot{\mathbf{A}}_1\dot{\mathbf{A}}_2\end{pmatrix},
$$

$$
\mathbf{A}_j = \exp\left(\boldsymbol{\Omega}_{i+j}\tilde{\mathbf{B}}(u)_j\right),
$$

$$
\dot{\mathbf{A}}_j = \mathbf{A}_j\boldsymbol{\Omega}_{i+j}\dot{\tilde{\mathbf{B}}}(u)_j,
$$

$$
\ddot{\mathbf{A}}_j = \dot{\mathbf{A}}_j\boldsymbol{\Omega}_{i+j}\dot{\tilde{\mathbf{B}}}(u)_j + \mathbf{A}_j\boldsymbol{\Omega}_{i+j}\ddot{\tilde{\mathbf{B}}}(u)_j
$$

关于求导，TUM的论文Efficient Derivative Computation for Cumulative B-Splines on Lie Groups详细推导了一种快速求导的方法，并且提供了实验的样例。这里就不详细说了。

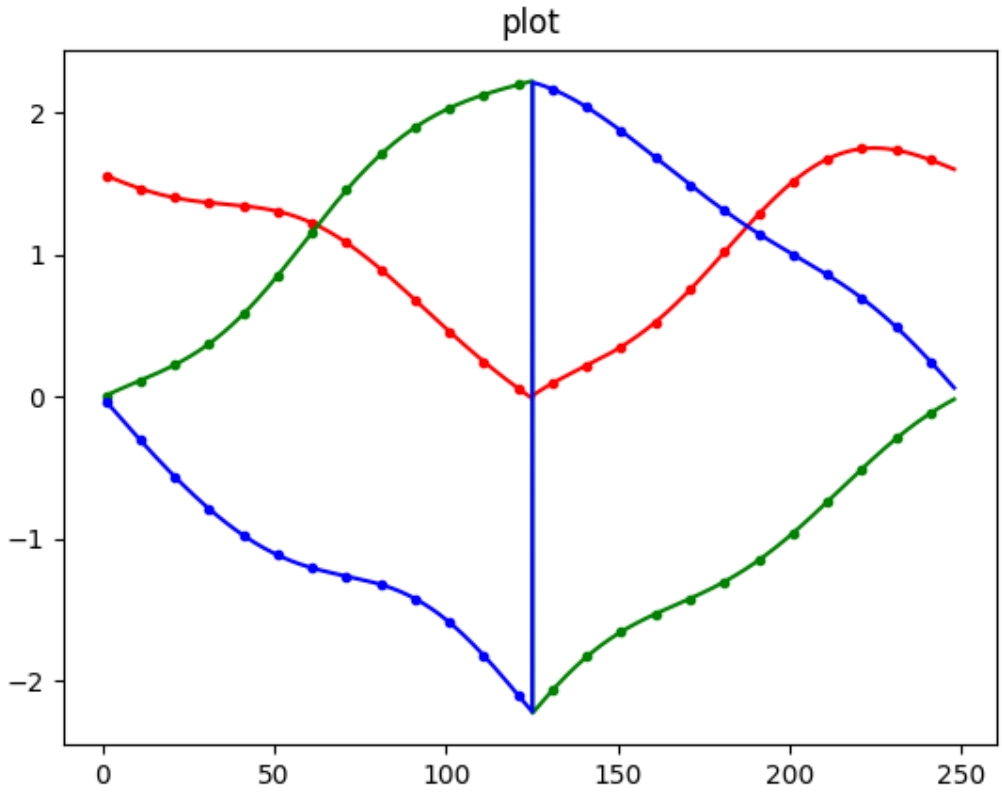
Chapter IV: 小demo

为了更好的理解我们的bspline-based trajectory，我写了一个小demo。

大概的工作流程是这样:

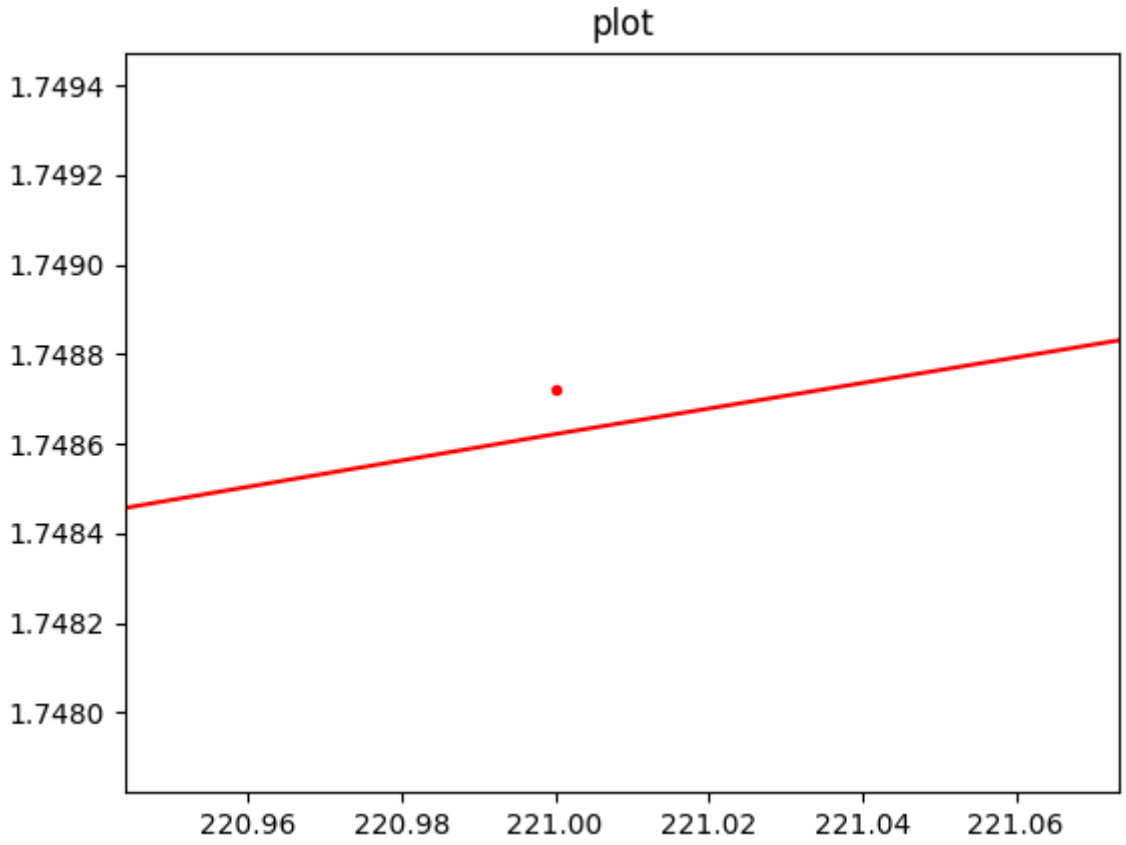
1. 定义bspline类，提供必要的函数存取控制点以及计算插值。
2. 拿一个dataset做实验。每隔10个点读入一个pose，加入到bspline类内。
3. 利用bspline类内的成员函数计算插值。
4. 输出结果，用python画出。

最终结果如图所示:



说明: 本次的实验用的是纯旋转的数据。红，绿，蓝三条曲线对应的是李代数三个值随时间的变化关系。上面的点就是控制点。

如果我们把它放大看，就会发现其实点并不在线上。这就对了——第二章的内容告诉我们，曲线并不一定非要经过控制点。但是这个相机它动的很匀速，这就导致了控制点和曲线十分贴近。



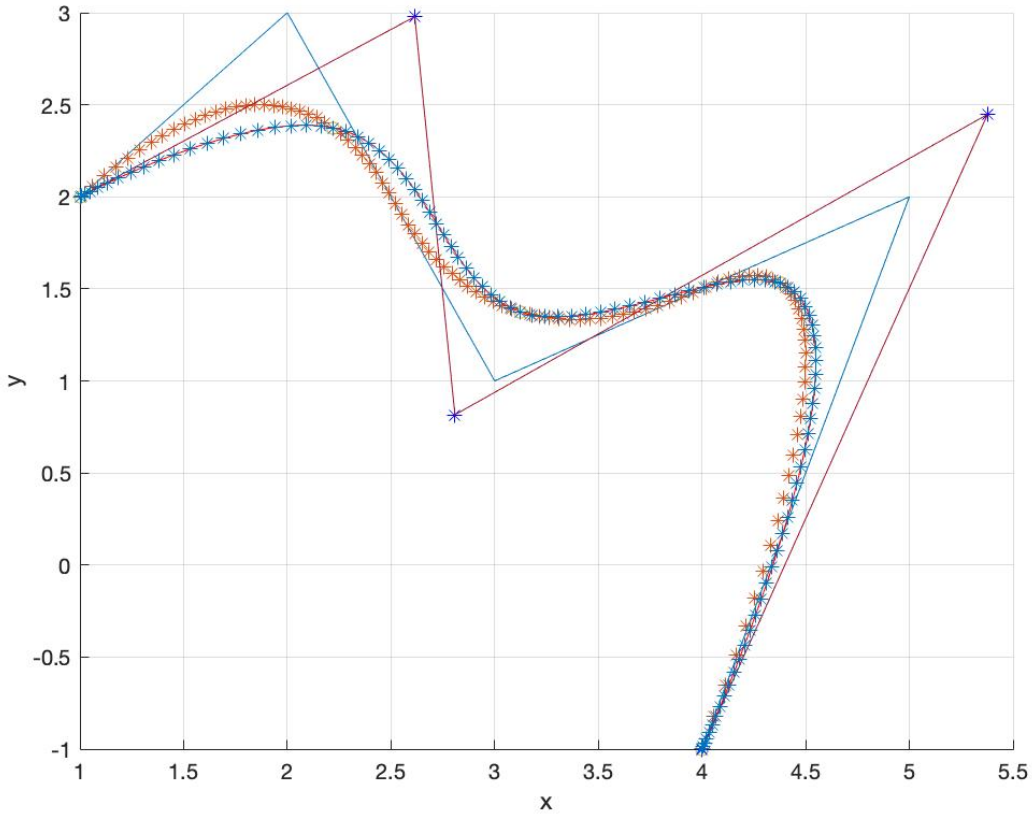
Chapter V: What else?

其实我的实验做的并不对。我用ground truth的值直接作为控制点去生成中间的插值，但是控制点并不应该是曲线上的点。因此，我去查阅了一些资料，找到了一个通过数据点反算控制点的方法。大概的流程就是先改写一下我们生成数据点的方式:

$$\begin{bmatrix} N_{1,3}(u_3) & N_{2,3}(u_3) & & & N_{0,3}(u_3) \\ N_{1,3}(u_4) & N_{2,3}(u_4) & N_{3,3}(u_4) & & \\ \ddots & \ddots & \ddots & & \\ & N_{n-4,3}(u_n) & N_{n-3,3}(u_n) & N_{n-2,3}(u_n) & \\ N_{n-1,3}(u_{n+1}) & & N_{n-3,3}(u_{n+1}) & N_{n-2,3}(u_{n+1}) & \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \\ \vdots \\ \boldsymbol{d}_{n-3} \\ \boldsymbol{d}_{n-2} \end{bmatrix} = \begin{bmatrix} \boldsymbol{q}_0 \\ \boldsymbol{q}_1 \\ \vdots \\ \boldsymbol{q}_{n-4} \\ \boldsymbol{q}_{n-3} \end{bmatrix}$$

这里$N_{i,k}(u)$和之前一样是基函数，$[\mathbf{d_1}, \mathbf{d_2}, \mathbf{...d_{n-3}}, \mathbf{d_{n-2}}]^{\mathbf{T}}$是控制点组成的向量，$[\mathbf{q_0}, \mathbf{q_1}, \mathbf{...q_{n-4}}, \mathbf{q_{n-3}}]^{\mathbf{T}}$是数据点组成的向量。 我们换个思路——把d当成未知量，q当成已知量，等式两边都左乘大矩阵的逆不就可以了！

做了实验，发现确实可以，此处附上我们的结果



蓝色线段的顶点是真实控制点，红色线段顶点星星是估计出来的控制点顶点。可以看到其实最后的结果也能接受。但是这种办法有两个缺点:

1. 有几个数据点就要估计几个控制点——也就是说，如果我们用4个控制点生成了一条100个数据点的曲线，要么我们放弃利用大部分的数据点，也就是我们目前采用的办法，每隔25个点跳着选取数据点，要么计算出100个控制点。
2. 这是实数域上的例子，要转换到李代数上很难——因为李代数对加法并不封闭，因此几乎无法用矩阵来描述生成数据点的过程，又何谈反算呢！

这个时候，我们不禁产生了一个疑问，那论文中是如何解决这个问题的呢？带着这个疑问我在论文中找到了这样一句话To initialize the system, all visual and inertial measurements are added to the spline in batch, and all control knots are set to the identity transform (control points are added every 0.1 s). 相当于我先随便放一个控制点，等我优化的时候再去动它，动的时候让他去贴合数据点。

因此，我个人觉得，可以放弃反算这个思路，找另一个方法去初始化控制点——反正连Identity Transform都行。

在这次作业里我简短地解释了B样条曲线的由来，以及以均匀3次B样条为例如何扩展到李代数中去表达一整条trajectory。由于本人时间才疏学浅，精力有限，很多地方

讲的不明白、很跳跃，甚至没有在代码里实现求导，希望各位看官能够谅解。写这个的主要目的还是为了能记录一下我第一个认真做的科研项目(真的很巧跟VIO的第二次作业能撞题目，很开心)。

感谢大家。

参考资料:

Spline Fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras.

深刻理解B 样条曲线（上）https://zhuanlan.zhihu.com/p/139759835

深刻理解B 样条曲线（下）https://zhuanlan.zhihu.com/p/140921657

Continuous-Time Visual-Inertial Odometry for Event Cameras
http://rpg.ifi.uzh.ch/docs/TRO18_Mueggler.pdf

Introduction to Computing with Geometry Notes
https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/

TUM CVPR 2020, Efficient Derivative Computation for Cumulative B-Splines on Lie Groups: https://arxiv.org/abs/1911.08860

SE3-pose-interpolation-using-bspline https://github.com/306327680/SE3-pose-interpolation-using-bspline

计算机辅助几何与非均匀有理B样条(2001年版)， 施法中