Apollo参考线平滑之分段spiral平滑

描 述: 算法来自于Baidu Apollo，本文是算法理论详解与代码的个人复现，供大家学习交流使用

本 文 代 码 开 源:

作 者: 董泰宏

邮 箱: 2396203400@qq.com

参 考: https://github.com/ApolloAuto/apollo/tree/r6.0.0

https://zhuanlan.zhihu.com/p/445586655

https://blog.csdn.net/xl_courage/article/details/121569105

0.引言

参考线模块的意义与前面的SplineSmoother提到的相同。

这里说一下不同的点:

在SplineSmoother构造的QP问题中，我们使用OSQP库进行求解，OSQP求解器描述问题的核心就在于目标函数以及约束条件中**矩阵的构造**。

然而标准的QP问题是很难构造的，我们更常遇见的是一些普通的NLP问题，针对普通的NLP问题，需要用到Ipopt库进行求解，Ipopt区别与OSQP库，它的核心不是构造目标变量的系数矩阵，而是用最直接的方式：**构造函数表达式+构造函数的雅可比矩阵**，所以ipopt的核心是函数的构造以及**函数的偏导**求解。

本文就是一个标准的Ipopt库应用案例，接下来请跟我一步一步从0构造一个NLP问题，并且用Ipopt进行求解。

1.分段spiral平滑算法

1.1 介绍

spiral螺旋线的参数方程为:

$$\theta(s) = as^5 + bs^4 + cs^3 + ds^2 + es + f$$

它描述的是道路的切角与弧长的关系，而**道路的曲率——**切角与弧长的变化:

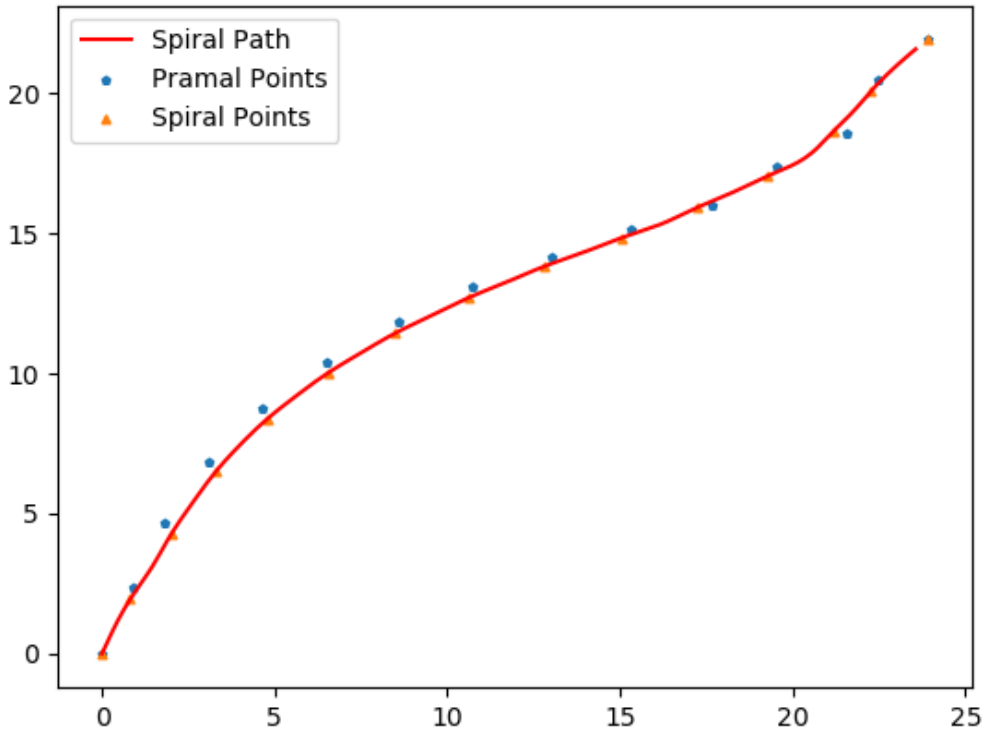
$$k(s) = \frac{d\theta}{ds} = \dot{\theta}(s) = 5as^4 + 4bs^3 + 3cs^2 + 2ds + e$$

螺旋线的优势：能够很直接方便的描述道路的曲率！而我们做参考线平滑，经常提到连续平滑，道路连续平滑的本质其实就是控制道路的曲率变化。所以螺旋线是参考线平滑中重要的一种算法，因为它能够很好的限制道路的总体曲率。

但是**螺旋线也有劣势**：前文spline平滑得到的结果就是直接的（x，y），而螺旋线需要经过菲涅尔积分才能够得到（x，y），不幸的是菲涅尔积分没有闭式解，只能通过数值积分的方法近似求解。

$$x_{i+1} = x_i + \int_0^{\triangle s_i} cos(\theta(s))ds$$
$$y_{i+1} = y_i + \int_0^{\triangle s_i} sin(\theta(s))ds$$

1.2 复现效果展示



2 算法剖析

NLP的标准型(Ipopt):

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ s.t. \quad & g_l \leq g(x) \leq g_u \\ & x_l \leq x \leq x_u \end{aligned}$$

同样这个标准型中构成问题的要素：**目标函数 + 条件约束**。但是它不要求函数有某些特定的形式，而是强调函数本身。

2.1 spiral平滑算法数学模型

每段螺旋线有6个未知系数a, b, c, d, e, f，理论上讲我们可以同spline一样将这些系数作为我们的目标变量，但其实更好的做法是直接以每个点的状态来倒推出曲线是更优秀的思想，因为我们可以把我们想要控制的量直接作为目标变量，相当于直接抓住了问题的本质。

2.1.1 目标变量

每个点我们控制这些内容：

$$[\theta_i \quad \dot{\theta}_i \quad \ddot{\theta}_i \quad x_i \quad y_i \quad \triangle s_i]$$

那么全部N个点构成的**目标变量**为（整理一下顺序）：

$$[\theta_0 \quad \dot{\theta}_0 \quad \ddot{\theta}_0 \quad x_0 \quad y_0 \quad \dots \quad \theta_{n-1} \quad \dot{\theta}_{n-1} \quad \ddot{\theta}_{n-1} \quad x_{n-1} \quad y_{n-1} \quad \triangle s_0 \quad \dots \quad \triangle s_{n-2}]_{6n-1}$$

可见目标变量的规模为：5N+N-1；为什么s要少一个呢？因为参考点N个，则中间段只有N-1段。

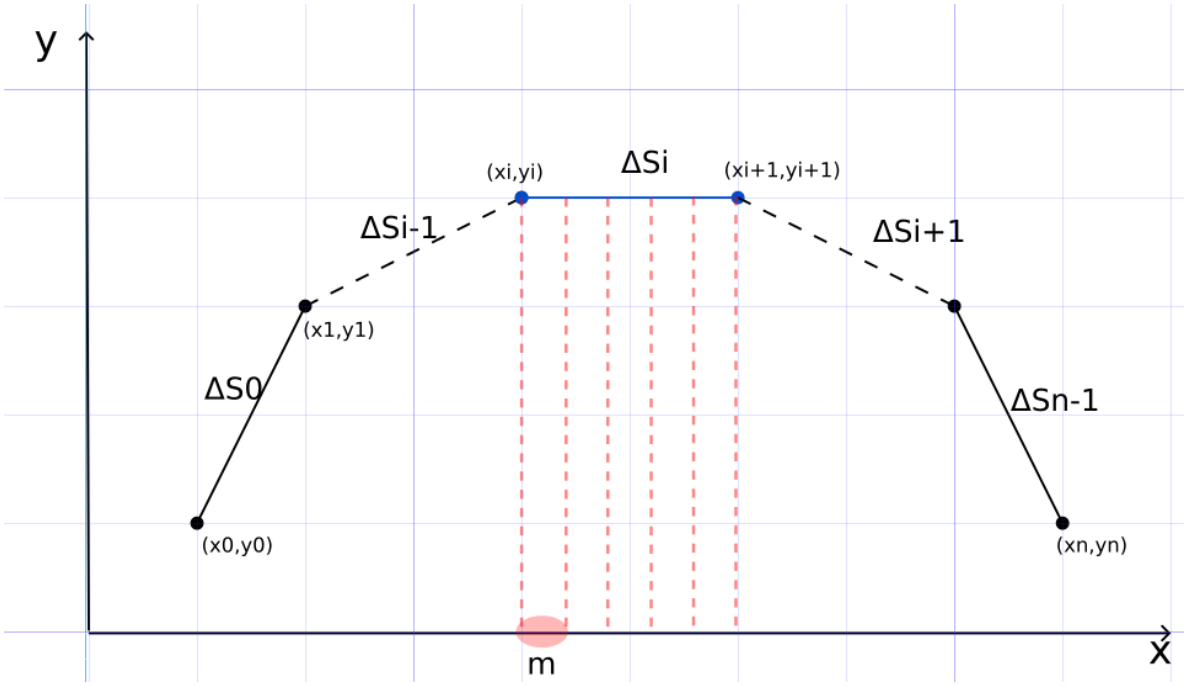
2.2.2 目标函数

目标函数由三项组成：**分段弧长+曲率+曲率变化率**

$$f(x) = w_{length} \cdot \sum_{i=1}^{n-1} \triangle s_i + w_{kappa} \cdot \sum_{i=1}^{n-1}\sum_{j=0}^{m-1} \dot{\theta}_i(s_j)^2 + + w_{dkappa} \cdot \sum_{i=1}^{n-1}\sum_{j=0}^{m-1} \ddot{\theta}_i(s_j)^2$$

为什么这里每个theta有m段？因为前面讲过，螺旋线菲涅尔积分只能通过数值积分的方式求近似解，数值积分中精度较高的是Gauss-Legendre方法，这个方法需要将原本的曲线分成m段进行积分。

为什么要在目标函数中进行分段？直觉上目标函数中不需要进行分段，因为不需要在这里积分出（x，y），但是我们的约束条件中进行了这样的分段积分，目标函数为了与约束保持一致性，所以这里也必须写成分段的形式，不然会造成目标函数和约束函数不是在描述同一个曲线，从而误差大的问题。同时也可以对每段spiral曲线采样更多的kappa,dkappa，从而控制曲线整体的平滑性



2.2.3 约束函数

约束条件有四部分构成：**段连接点约束+起点终点约束+运动学约束+位置约束**

段连接段约束：

$$\theta_{i+1} = \theta_i(\Delta s_i)$$
$$\dot{\theta}_{i+1} = \dot{\theta}_i(\Delta s_i)$$
$$\ddot{\theta}_{i+1} = \ddot{\theta}_i(\Delta s_i)$$
$$x_{i+1} = x_i + \int_0^{\triangle s_i} cos(\theta(s))ds$$
$$y_{i+1} = y_i + \int_0^{\triangle s_i} sin(\theta(s))ds$$

起点终点约束：

$$\begin{cases}\theta_0 = \theta_{start}\\ \dot{\theta}_0 = \dot{\theta}_{start}\\ \ddot{\theta}_0 = \ddot{\theta}_{start}\\ x = x_{start}\\ y_0 = y_{start}\end{cases}$$

$$\begin{cases}\theta_{n-1} = \theta_{end}\\ \dot{\theta}_{n-1} = \dot{\theta}_{end}\\ \ddot{\theta}_{n-1} = \ddot{\theta}_{end}\\ x_{n-1} = x_{end}\\ y_{n-1} = y_{end}\end{cases}$$

中间点运动学与边界约束：

$$\begin{cases}\theta_{iref} - \pi/2 \le \theta_i \le \theta_{iref} + \pi/2\\ -0.25 \le \dot{\theta}_i \le 0.25\\ -0.02 \le \ddot{\theta}_i \le 0.02\\ x_{iref} - r_i \le x_i \le x_{iref} + r_i\\ y_{iref} - r_i \le y_i \le y_{iref} + r_i\\ Distance(i, i+1) \le \triangle s_i \le Distance(i, i+1) \cdot \pi/2\end{cases}$$

位置约束(感觉这个条件有点重复了)：

$$(x_i - x_{iref})^2 + (y_i - y_{iref})^2 <= r_i^2$$

从标准式中可以看出来，约束有两种类型：

g(x)——约束函数；在上面的这么多约束中只有三个约束函数（3N-2维）

$$x_{i+1} = x_i + \int_0^{\triangle s_i} cos(\theta(s))ds, N-1\text{维}$$

$$y_{i+1} = y_i + \int_0^{\triangle s_i} sin(\theta(s))ds,\ N-1\text{维}$$

$$(x_i - x_{iref})^2 + (y_i - y_{iref})^2 <= r_i^2,\ N\text{维}$$

x——变量本身。

为什么要区分约束函数和变量本身呢？因为约束函数是需要求雅可比矩阵的，而变量本身不需要，所以我们要单独拎出来以便于分析维度与偏导，防止后续看代码混乱。

2.2 Ipopt建模求解（手动微分版本）

前面已经整理了问题的数学模型，接下来是针对Ipopt求解库，一步一步的构造具体的问题输入。

2.2.0 手动微分与自动微分

见最后的**总结**

2.2.1 Ipopt问题构造概述

求解库主要是继承子TNLP类，然后根据数学模型，实例化各个函数的内容。换而言之，Ipopt只要能够把下面几个函数搞清楚怎么构造的，就算是做完了。所有需要实例化的函数如下：

```
//nlp问题的基本信息：目标变量规模、约束函数规模、约束函数jacobian矩阵非零量、目标hessian矩阵非零量
get_nlp_info();

//约束变量的上下限、约束函数的上下限
get_bounds_info();
```

```
//nlp问题的初始点
get_starting_point();

//构造问题的目标函数表达式
eval_f();

//目标函数的jacobian矩阵
eval_grad_f();

//构造问题的约束函数表达式
eval_g();

//约束函数的jacobian矩阵
eval_jac_g();

//目标函数的hessian矩阵
eval_h();

//无实质作用：可以进行一些求解结果量的打印
finalize_solution();
```

2.2.2 细节剖析与代码

此部分一切来源于上面的数学模型，如果有看不懂 的部分请回头去对照看。

2.2.2.1 get_nlp_info

目标变量的规模：N个点对应6N-1个目标变量。

约束函数的规模：前面提到了约束分为**约束函数**和**变量约束本身**，这里需要的信息是约束函数的规模，与变量约束无关。本问题一共有3个约束函数：规模为（3N-2）=（N-1）*2+N；

$$x_{i+1} = x_i + \int_0^{\triangle s_i} cos(\theta(s))ds, N-1维$$

$$y_{i+1} = y_i + \int_0^{\triangle s_i} sin(\theta(s))ds, \ N-1维$$

$$(x_i - x_{iref})^2 + (y_i - y_{iref})^2 <= r_i^2, \ N维$$

约束函数jacobian矩阵非零量：这里是说约束函数的jacobian矩阵一共有多少个非零量，这里先说结论，怎么来的按下不表（看到eval_grad_f那里自然会明白），(N-1)x2x9+2N;

目标hessian矩阵非零量: 这里为0，因为本问题求解用的是拟牛顿法，不需要hessian矩阵，所以不需要管，设置为0即可。

索引风格：这个风格你是什么语言就怎么设置，它指代索引的顺序，比如C/C++的下标是从0开始的，所以你就设置为C_STYLE就行了。

```
bool SpiralSmoother::get_nlp_info(int& n, int& m, int& nnz_jac_g,
                                  int& nnz_h_lag,
                                  IndexStyleEnum& index_style) {
  //目标变量
  n = number * 5 + number - 1;
  //约束函数规模
  m = (number - 1) * 2 + number;
  //约束函数jacobian矩阵非零量
  nnz_jac_g = (number - 1) * 2 * 9 + number * 2;
  //目标hessian矩阵非零量
```

```
  nnz_h_lag = 0;
  //索引风格
  index_style = IndexStyleEnum::C_STYLE;
  return true;
}
```

2.2.2.2 get_bounds_info

变量的约束：变量的约束就是每个变量的上下界，对应着目标变量的位置来设置即可；

约束函数的上下限：函数没有严格的顺序，根据数学公式设置即可。

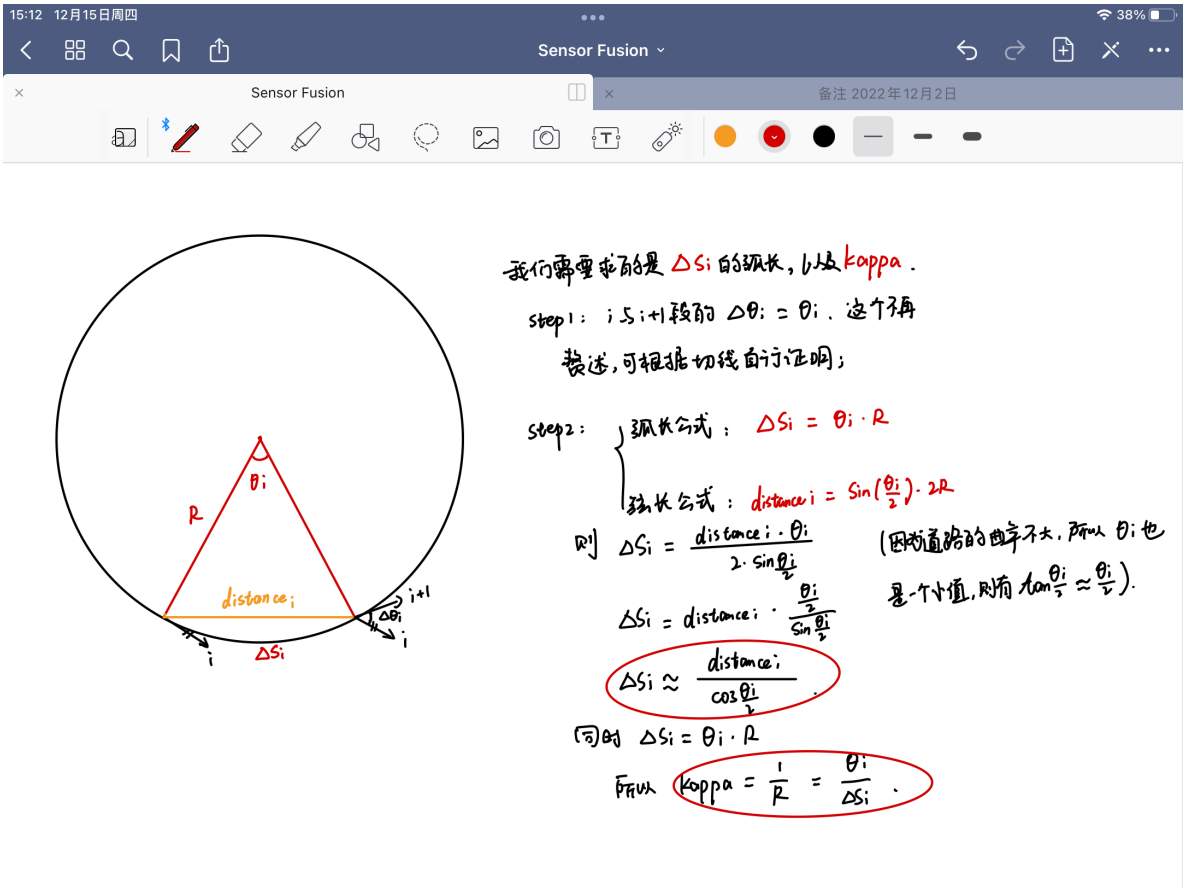
```
bool SpiralSmoother::get_bounds_info(int n, double* x_l, double* x_u,
                                     int m, double* g_l, double* g_u) {
  //每次循环设置的是第i个点的五个量: theta, kappa, dkappa, x, y
  for (int i = 0; i < num_of_points_; ++i) {
    int index = i * 5;
    double theta_lower = 0.0;
    double theta_upper = 0.0;
    double kappa_lower = 0.0;
    double kappa_upper = 0.0;
    double dkappa_lower = 0.0;
    double dkappa_upper = 0.0;
    double x_lower = 0.0;
    double x_upper = 0.0;
    double y_lower = 0.0;
    double y_upper = 0.0;
    if (i == 0) {
      //起点约束
      theta_lower = start_theta_;
      theta_upper = start_theta_;
      kappa_lower = start_kappa_;
      kappa_upper = start_kappa_;
      dkappa_lower = start_dkappa_;
      dkappa_upper = start_dkappa_;
      x_lower = start_x_;
      x_upper = start_x_;
      y_lower = start_y_;
      y_upper = start_y_;
    } else if (i == number - 1) {
      //终点约束
      theta_lower = end_theta_;
      theta_upper = end_theta_;
      kappa_lower = end_kappa_;
      kappa_upper = end_kappa_;
      dkappa_lower = end_dkappa_;
      dkappa_upper = end_dkappa_;
      x_lower = end_x_;
      x_upper = end_x_;
      y_lower = end_y_;
      y_upper = end_y_;
    }else {
      //中间点的运动学与边界约束
      theta_lower = relative_theta_[i] - M_PI * 0.2;
      theta_upper = relative_theta_[i] + M_PI * 0.2;
      kappa_lower = -0.25;
      kappa_upper = 0.25;
      dkappa_lower = -0.02;
```

```
      dkappa_upper = 0.02;
      x_lower = init_points_[i].x() - boundary;
      x_upper = init_points_[i].x() + boundary;
      y_lower = init_points_[i].y() - boundary;
      y_upper = init_points_[i].y() + boundary;
    }
    //将上面的上下界填到x_l, x_u中
    // theta
    x_l[index] = theta_lower;
    x_u[index] = theta_upper;
    // kappa
    x_l[index + 1] = kappa_lower;
    x_u[index + 1] = kappa_upper;
    // dkappa
    x_l[index + 2] = dkappa_lower;
    x_u[index + 2] = dkappa_upper;
    // x
    x_l[index + 3] = x_lower;
    x_u[index + 3] = x_upper;
    // y
    x_l[index + 4] = y_lower;
    x_u[index + 4] = y_upper;
  }

  //delta_s的边界约束
  int variable_offset = number * 5;
  for (int i = 0; i < number - 1; ++i) {
    //TODO:把原始点两两之间的距离给存到vector<double> distance中
    x_l[variable_offset + i] = distance[i] - 2.0 * boundary;
    x_u[variable_offset + i] = distance[i] * 3.1415926 * 0.5;
  }

  //段连接点的位置约束：等式约束，上下界都是0
  for (int i = 0; i < number  - 1; ++i) {
    // x
    g_l[i * 2] = 0.0;
    g_u[i * 2] = 0.0;
    // y
    g_l[i * 2 + 1] = 0.0;
    g_u[i * 2 + 1] = 0.0;
  }
  //每个点的位置约束
  int constraint_offset = 2 * (num_of_points_ - 1);
  for (int i = 0; i < number; ++i) {
    g_l[constraint_offset + i] = 0.0;
    g_u[constraint_offset + i] = pow(boundary, 2);
  }
  return true;
}
```

2.2.2.3 get_starting_point

问题的初始点：根据原始数据初始化即可。

原始的theta, x, y不用多说，重点说一下原始的delta_s, kappa怎么求。




```
bool SpiralSmoother::get_starting_point(int n, bool init_x, double* x,
                                        bool init_z, double* z_L,
                                        double* z_U, int m,
                                        bool init_lambda,
                                        double* lambda) {
  //根据原始数据来赋值第i个点的theta, kappa, dkappa, x, y
  for (int i = 0; i < number; ++i) {
    int index = i * 5;
    x[index] = relative_theta_[i];
    x[index + 1] = 0.0;
    x[index + 2] = 0.0;
    x[index + 3] = input_points[i].first;
    x[index + 4] = input_points[i].second;
  }

  //根据原始数据来赋值第i段的delta_s
  int variable_offset = number * 5;
  for (int i = 0; i < number - 1; ++i) {
    double delta_theta = relative_theta_[i + 1] - relative_theta_[i];
    x[variable_offset + i] = distance[i] / cos(0.5 * delta_theta);
  }

  //曲率的初始值
  for (int i = 0; i + 1 < num_of_points_; ++i) {
    double delta_theta = relative_theta_[i + 1] - relative_theta_[i];
    x[(i + 1) * 5 + 1] = delta_theta / x[variable_offset + i];
  }
  x[1] = x[6];
  return true;
}
```

2.2.2.4 eval_f

$$f(x) = w_{length} \cdot \sum_{i=1}^{n-1} \triangle s_i + w_{kappa} \cdot \sum_{i=1}^{n-1}\sum_{j=0}^{m-1} \dot{\theta}_i(s_j)^2 + + w_{dkappa} \cdot \sum_{i=1}^{n-1}\sum_{j=0}^{m-1} \ddot{\theta}_i(s_j)^2$$

其中:

$$\theta_i(s_j) = a_i s_j^5 + b_i s_j^4 + c_i s_j^3 + d_i s_j^2 + e_i s_j + f_i$$
$$\dot{\theta}_i(s_j) = 5a_i s_j^4 + 4b_i s_j^3 + 3c_i s_j^2 + 2d_i s_j + e_i$$
$$\ddot{\theta}_i(s_j) = 20a_i s_j^3 + 12b_i s_j^2 + 6c_i s_j + 2d_i$$

其中s_j为$deltas$分段后的s

由于每段曲线的系数是由首尾点的6个方程确定的，则每段系数的表达式为:

$$a_i = \frac{-6\theta_i}{s^5} - \frac{3\dot{\theta}_i}{s^4} - \frac{\ddot{\theta}_i}{2s^3} + \frac{6\theta_{i+1}}{s^5} - \frac{3\dot{\theta}_{i+1}}{s^4} + \frac{\ddot{\theta}_{i+1}}{2s^3}$$
$$b_i = \frac{15\theta_i}{s^4} + \frac{8\dot{\theta}_i}{s^3} + \frac{3\ddot{\theta}_i}{2s^2} - \frac{15\theta_{i+1}}{s^4} + \frac{7\dot{\theta}_{i+1}}{s^3} - \frac{\ddot{\theta}_{i+1}}{s^2}$$
$$c_i = \frac{-10\theta_i}{s^3} - \frac{6\dot{\theta}_i}{s^2} - \frac{3\ddot{\theta}_i}{2s} + \frac{10\theta_{i+1}}{s^3} - \frac{4\dot{\theta}_{i+1}}{s^2} + \frac{\ddot{\theta}_{i+1}}{2s}$$
$$d_i = \frac{\ddot{\theta}_i}{2}$$
$$e_i = \dot{\theta}_i$$
$$f_i = \theta_i$$

其中,$s_i = deltas, deltas$的推导公式上面已经提到过了。

关于各个系数的偏导，下面以a为例展示一下，其他系数可自行去推导:

$$\frac{\partial a_i}{\partial \theta_i} = -\frac{6}{s^5}, \quad \frac{\partial a_i}{\partial \dot{\theta}_i} = -\frac{3}{s^4}, \quad \frac{\partial a_i}{\partial \ddot{\theta}_i} = -\frac{1}{2s^2}, \quad \frac{\partial a_i}{\partial \theta_{i+1}} = \frac{6}{s^5}, \quad \frac{\partial a_i}{\partial \dot{\theta}_{i+1}} = -\frac{3}{s^4}, \quad \frac{\partial a_i}{\partial \ddot{\theta}_{i+1}} = \frac{1}{2s^2}$$
$$\frac{\partial a_i}{\partial \triangle s_i} = \frac{30\theta_i}{s^6} + \frac{12\dot{\theta}_i}{s^5} + \frac{1.5\ddot{\theta}_i}{s^4} - \frac{30\theta_{i+1}}{s^6} + \frac{12\dot{\theta}_{i+1}}{s^5} - \frac{1.5\ddot{\theta}_{i+1}}{s^4}$$

```
bool SpiralSmoother::eval_f(int n, const double* x, bool new_x,
                            double& obj_value) {
  obj_value = 0.0;
  for (int i = 0; i + 1 < num_of_points_; ++i) {
    double delta_si = x[5  * number + i];
    double theta_i = x[5 * i + 0];
    double dtheta_i = x[5 * i + 1];
    double ddtheta_i = x[5 * i + 2];
    double theta_i_1 = x[5 * (i + 1) + 0];
    double dtheta_i_1 = x[5 * (i + 1) + 1];
    double ddtheta_i_1 = x[5 * (i + 1) + 2];
    //第一项
    obj_value += x[5 * number + i] * weight_curve_length_;

    for (int j = 0; j < num_of_internal_points_; ++j) {
      double ratio =
          static_cast<double>(j) / static_cast<double>(num_of_internal_points_);

      //第二项
      double kappa =
          5 * pow(ratio, 4) * (-6 * theta_i / delta_si - 3 * dtheta_i - 0.5 *
ddtheta_i * delta_si + 6 * theta_i_1 / delta_si - 3 * dtheta_i_1 + 0.5 *
ddtheta_i_1 * delta_si)+
          4 * pow(ratio, 3) *  (15 * theta_i / delta_si + 8 * dtheta_i + 1.5 *
ddtheta_i * delta_si - 15 * theta_i_1 / delta_si + 7 * dtheta_i_1 - ddtheta_i_1
* delta_si) +
```

```
          3 * pow(ratio, 2) * (-10 * theta_i / delta_si - 6 * dtheta_i - 1.5 *
ddtheta_i * delta_si + 10 * theta_i_1 /delta_si - 4 * dtheta_i_1 + 0.5 *
ddtheta_i_1 * delta_si) +
          2 * ratio * ddtheta_i * 0.5 * delta_si +
          dtheta_i;
      obj_value += kappa * kappa * weight_kappa_;

      //第三项
      double dkappa =
          20 * pow(ratio, 3) * (-6 * theta_i / pow(delta_si, 2) - 3 * dtheta_i
 / delta_si - 0.5 * ddtheta_i + 6 * theta_i_1 / pow(delta_si, 2) - 3 *
dtheta_i_1 / delta_si + 0.5 * ddtheta_i_1) +
          12 * pow(ratio, 2) * (15 * theta_i / pow(delta_si, 2) + 8 * dtheta_i /
delta_si + 1.5 * ddtheta_i - 15 * theta_i_1 / pow(delta_si, 2) + 7 * dtheta_i_1
/ delta_si - ddtheta_i_1) +
          6 * ratio * (-10 * theta_i / pow(delta_si, 2) - 6 * dtheta_i /
delta_si - 1.5 * ddtheta_i + 10 * theta_i_1 / pow(delta_si, 2) - 4 * dtheta_i_1
/ delta_si + 0.5 * ddtheta_i_1) +
          2 * ddtheta_i * 0.5;
      obj_value += dkappa * dkappa * weight_dkappa_;
    }
  }
  return true;
}
```

2.2.2.5 eval_grad_f

以第i段为例

第一项的偏导:

$$\triangle s_i\text{仅与}\triangle s_i\text{本身相关,与任意的}\theta,\dot{\theta},\ddot{\theta},x,y\text{无关,因此:}$$

$$\frac{\partial\triangle s_i}{\partial\triangle s_i}=1,\text{其他时候都为}0.$$

第二项的偏导:（其中各系数的偏导在2.2.2.4节中提到，不懂的回头去看）

$$\text{每一项}\sum_{j=0}^{m-1}\dot{\theta}_i(s_j)^2,\text{仅与相邻两项}\theta_i,\dot{\theta}_i,\ddot{\theta}_i,\theta_{i+1},\dot{\theta_{i+1}},\ddot{\theta_{i+1}},\triangle s_i\text{相关，与其它无关，因此：}$$

$$\begin{cases}
\frac{\partial\dot{\theta}_i(s_j)^2}{\partial\theta_i}=2\dot{\theta}_i(s_j)\frac{\partial\dot{\theta}_i(s_j)}{\partial\theta_i}=2\dot{\theta}_i(s_j)\frac{\partial(5a_is_j^4+4b_is_j^3+3c_is_j^2+2d_is_j+e_i)}{\partial\theta_i}\\
\qquad=2\dot{\theta}_i(s_j)[5\frac{\partial a_i}{\partial\theta_i}s_j^4+4\frac{\partial b_i}{\partial\theta_i}s_j^3+3\frac{\partial c_i}{\partial\theta_i}s_j^2+2\frac{\partial d_i}{\partial\theta_i}s_j+\frac{\partial e_i}{\partial\theta_i}]\\
\frac{\partial\dot{\theta}_i(s_j)^2}{\partial\dot{\theta}_i}=2\dot{\theta}_i(s_j)[5\frac{\partial a_i}{\partial\dot{\theta}_i}s_j^4+4\frac{\partial b_i}{\partial\dot{\theta}_i}s_j^3+3\frac{\partial c_i}{\partial\dot{\theta}_i}s_j^2+2\frac{\partial d_i}{\partial\dot{\theta}_i}s_j+\frac{\partial e_i}{\partial\dot{\theta}_i}]\\
\frac{\partial\dot{\theta}_i(s_j)^2}{\partial\ddot{\theta}_i}=2\dot{\theta}_i(s_j)[5\frac{\partial a_i}{\partial\ddot{\theta}_i}s_j^4+4\frac{\partial b_i}{\partial\ddot{\theta}_i}s_j^3+3\frac{\partial c_i}{\partial\ddot{\theta}_i}s_j^2+2\frac{\partial d_i}{\partial\ddot{\theta}_i}s_j+\frac{\partial e_i}{\partial\ddot{\theta}_i}]\\
\frac{\partial\dot{\theta}_i(s_j)^2}{\partial\theta_{i+1}}=2\dot{\theta}_i(s_j)[5\frac{\partial a_i}{\partial\theta_{i+1}}s_j^4+4\frac{\partial b_i}{\partial\theta_{i+1}}s_j^3+3\frac{\partial c_i}{\partial\theta_{i+1}}s_j^2+2\frac{\partial d_i}{\partial\theta_{i+1}}s_j+\frac{\partial e_i}{\partial\theta_{i+1}}]\\
\frac{\partial\dot{\theta}_i(s_j)^2}{\partial\dot{\theta}_{i+1}}=2\dot{\theta}_i(s_j)[5\frac{\partial a_i}{\partial\dot{\theta}_{i+1}}s_j^4+4\frac{\partial b_i}{\partial\dot{\theta}_{i+1}}s_j^3+3\frac{\partial c_i}{\partial\dot{\theta}_{i+1}}s_j^2+2\frac{\partial d_i}{\partial\dot{\theta}_{i+1}}s_j+\frac{\partial e_i}{\partial\dot{\theta}_{i+1}}]\\
\frac{\partial\dot{\theta}_i(s_j)^2}{\partial\ddot{\theta}_{i+1}}=2\dot{\theta}_i(s_j)[5\frac{\partial a_i}{\partial\ddot{\theta}_{i+1}}s_j^4+4\frac{\partial b_i}{\partial\ddot{\theta}_{i+1}}s_j^3+3\frac{\partial c_i}{\partial\ddot{\theta}_{i+1}}s_j^2+2\frac{\partial d_i}{\partial\ddot{\theta}_{i+1}}s_j+\frac{\partial e_i}{\partial\ddot{\theta}_{i+1}}]\\
\frac{\partial\dot{\theta}_i(s_j)^2}{\partial\triangle s_i}=2\dot{\theta}_i(s_j)\frac{\partial\dot{\theta}_i(s_j)}{\partial\triangle s_i}=2\dot{\theta}_i(s_j)\frac{\partial(5a_is_j^4+4b_is_j^3+3c_is_j^2+2d_is_j+e_i)}{\partial\triangle s_i}\\
\qquad=2\dot{\theta}_i(s_j)[5\frac{\partial a_i}{\partial\triangle s_i}s_j^4+4\frac{\partial b_i}{\partial\triangle s_i}s_j^3+3\frac{\partial c_i}{\partial\triangle s_i}s_j^2+2\frac{\partial d_i}{\partial\triangle s_i}s_j+\frac{\partial e_i}{\partial\triangle s_i}\\
\qquad\qquad+20a_is_j^3\frac{\partial s_j}{\partial\triangle s_i}+12b_is_j^2\frac{\partial s_j}{\partial\triangle s_i}+6c_is_j\frac{\partial s_j}{\partial\triangle s_i}+2d_i\frac{\partial s_j}{\partial\triangle s_i}]
\end{cases}$$

$$\text{其他偏导均为}0.$$

第三项的偏导：

每一项 $\sum_{j=0}^{m-1}\ddot{\theta}_i(s_j)^2$，仅与相邻两项 $\theta_i,\dot{\theta}_i,\ddot{\theta}_i,\theta_{i+1},\dot{\theta_{i+1}},\ddot{\theta_{i+1}},\triangle s_i$ 相关，与其它无关，因此：

$$
\begin{cases}
\frac{\partial\ddot{\theta}_i(s_j)^2}{\partial\theta_i}=2\ddot{\theta}_i(s_j)\frac{\partial\ddot{\theta}_i(s_j)}{\partial\theta_i}=2\ddot{\theta}_i(s_j)\frac{\partial(20a_is_j^3+12b_is_j^2+6c_is_j+2d_i)}{\partial\theta_i} \\
\qquad=2\ddot{\theta}_i(s_j)[20\frac{\partial a_i}{\partial\theta_i}s_j^3+12\frac{\partial b_i}{\partial\theta_i}s_j^2+6\frac{\partial c_i}{\partial\theta_i}s_j+2\frac{\partial d_i}{\partial\theta_i}] \\
\frac{\partial\ddot{\theta}_i(s_j)^2}{\partial\dot{\theta}_i}=2\ddot{\theta}_i(s_j)[20\frac{\partial a_i}{\partial\dot{\theta}_i}s_j^3+12\frac{\partial b_i}{\partial\dot{\theta}_i}s_j^2+6\frac{\partial c_i}{\partial\dot{\theta}_i}s_j+2\frac{\partial d_i}{\partial\dot{\theta}_i}] \\
\frac{\partial\ddot{\theta}_i(s_j)^2}{\partial\ddot{\theta}_i}=2\ddot{\theta}_i(s_j)[20\frac{\partial a_i}{\partial\ddot{\theta}_i}s_j^3+12\frac{\partial b_i}{\partial\ddot{\theta}_i}s_j^2+6\frac{\partial c_i}{\partial\ddot{\theta}_i}s_j+2\frac{\partial d_i}{\partial\ddot{\theta}_i}] \\
\frac{\partial\ddot{\theta}_i(s_j)^2}{\partial\theta_{i+1}}=2\ddot{\theta}_i(s_j)[20\frac{\partial a_i}{\partial\theta_{i+1}}s_j^3+12\frac{\partial b_i}{\partial\theta_{i+1}}s_j^2+6\frac{\partial c_i}{\partial\theta_{i+1}}s_j+2\frac{\partial d_i}{\partial\theta_{i+1}}] \\
\frac{\partial\ddot{\theta}_i(s_j)^2}{\partial\dot{\theta}_{i+1}}=2\ddot{\theta}_i(s_j)[20\frac{\partial a_i}{\partial\dot{\theta}_{i+1}}s_j^3+12\frac{\partial b_i}{\partial\dot{\theta}_{i+1}}s_j^2+6\frac{\partial c_i}{\partial\dot{\theta}_{i+1}}s_j+2\frac{\partial d_i}{\partial\dot{\theta}_{i+1}}] \\
\frac{\partial\ddot{\theta}_i(s_j)^2}{\partial\ddot{\theta}_{i+1}}=2\ddot{\theta}_i(s_j)[20\frac{\partial a_i}{\partial\ddot{\theta}_{i+1}}s_j^3+12\frac{\partial b_i}{\partial\ddot{\theta}_{i+1}}s_j^2+6\frac{\partial c_i}{\partial\ddot{\theta}_{i+1}}s_j+2\frac{\partial d_i}{\partial\ddot{\theta}_{i+1}}] \\
\frac{\partial\ddot{\theta}_i(s_j)^2}{\partial\triangle s_i}=2\ddot{\theta}_i(s_j)\frac{\partial\dot{\theta}_i(s_j)}{\partial\triangle s_i}=2\ddot{\theta}_i(s_j)\frac{\partial(20a_is_j^3+12b_is_j^2+6c_is_j+2d_i)}{\partial\triangle s_i} \\
\qquad=2\ddot{\theta}_i(s_j)[20\frac{\partial a_i}{\partial\triangle s_i}s_j^3+12\frac{\partial b_i}{\partial\triangle s_i}s_j^2+6\frac{\partial c_i}{\partial\triangle s_i}s_j+2\frac{\partial d_i}{\partial\triangle s_i} \\
\qquad\qquad+60a_is_j^2\frac{\partial s_j}{\partial\triangle s_i}+24b_is_j\frac{\partial s_j}{\partial\triangle s_i}+6c_i\frac{\partial s_j}{\partial\triangle s_i}]
\end{cases}
$$

其他偏导均为 0.

```
//这里代码太长了，就不放出来了，可以参照上面的公式，去我的源码对照着看
bool SpiralSmoother::eval_grad_f(int n, const double* x, bool new_x,
                                 double* grad_f) {
  int variable_offset = number * 5;
  //注意这里有循环了n-1次，因为一共只有n个点一共只有n-1段。
  for (int i = 0; i + 1 < number; ++i) {
    int index0 = i * 5;
    int index1 = (i + 1) * 5;

    auto& spiral_curve = piecewise_paths_[i];
    double delta_s = spiral_curve.ParamLength();

    grad_f[variable_offset + i] += weight_curve_length_ * 1.0;

    for (int j = 0; j < num_of_internal_points_; ++j) {
        ...
    }
  }
```

2.2.2.6 eval_g

约束函数：

$$
\begin{aligned}
x_{i+1}&=x_i+\int_0^{\triangle s_i}cos(\theta(s))ds, N-1\text{维} \\
y_{i+1}&=y_i+\int_0^{\triangle s_i}sin(\theta(s))ds,\ N-1\text{维} \\
(x_i-x_{iref})^2&+(y_i-y_{iref})^2<=r_i^2,\ N\text{维}
\end{aligned}
$$

前面提到菲涅尔积分无闭式解，需要用**高斯-勒让德公式**进行数值近似：

$$x_{i+1} = x_i + \frac{\triangle s_i}{2} \cdot \sum_{i=0}^{m-1} w_i \cdot cos(\theta(\frac{\triangle s_i}{2} \cdot \xi_i + \frac{\triangle s_i}{2}))$$

$$y_{i+1} = y_i + \frac{\triangle s_i}{2} \cdot \sum_{i=0}^{m-1} w_i \cdot sin(\theta(\frac{\triangle s_i}{2} \cdot \xi_i + \frac{\triangle s_i}{2}))$$

其中w_i、ξ_i根据分段数m，去查表（高斯—勒让德积分点与权重系数表）得到。

n	x_k	A_k
0	0.0000000	2.0000000
1	± 0.5773503	1.0000000
2	± 0.7745967 0.0000000	0.5555556 0.8888889
3	± 0.8611363 ± 0.3399810	0.3478548 0.6521452
4	± 0.9061798 ± 0.5384693 0.0000000	0.2369269 0.4786287 0.5688889


```
bool SpiralSmootherNLP::eval_g(Ipopt::Index n, const Ipopt::Number* x,
                               bool new_x, Ipopt::Index m, Ipopt::Number* g) {
  //公式1 2, 分别N-1维
  for (int i = 0; i + 1 < SourceData.number; ++i) {
    int index0 = i * 5;
    int index1 = (i + 1) * 5;
    double delta_si = x[5 * SourceData.number + i];
    double theta_i = x[5 * i + 0];
    double dtheta_i = x[5 * i + 1];
    double ddtheta_i = x[5 * i + 2];
    double theta_i_1 = x[5 * (i + 1) + 0];
    double dtheta_i_1 = x[5 * (i + 1) + 1];
    double ddtheta_i_1 = x[5 * (i + 1) + 2];
    CoefInitialize(theta_i, dtheta_i, ddtheta_i, theta_i_1, dtheta_i_1,
                   ddtheta_i_1, delta_si);

    //这一部分是求gauss-legendre数值积分
    vector<double> epsiloni = {-0.90617984594, -0.53846931011, 0, 0.53846931011,
                               0.90617984594};
    vector<double> wi = {0.236926885, 0.4786286705, 0.56888888889, 0.4786286705,
                         0.236926885};
    double gauss_legendre_x = 0;
    double gauss_legendre_y = 0;
```

```
    for (int j = 0; j < SourceData.m; ++j) {
      double segment_s = 0.5 * delta_si * epsiloni[j] + 0.5 * delta_si;
      double theta_segment_s = SpiralCurve(0, 1, segment_s);
      gauss_legendre_x += wi[j] * cos(theta_segment_s);
      gauss_legendre_y += wi[j] * sin(theta_segment_s);
    }
    gauss_legendre_x *= delta_si / 2;
    gauss_legendre_y *= delta_si / 2;

    double x_diff = x[index1 + 3] - x[index0 + 3] - gauss_legendre_x;
    g[i * 2] = x_diff;
    double y_diff = x[index1 + 4] - x[index0 + 4] - gauss_legendre_y;
    g[i * 2 + 1] = y_diff;
  }
  //公式3，N维
  int constraint_offset = 2 * (SourceData.number - 1);
  for (int i = 0; i < SourceData.number; ++i) {
    int variable_index = i * 5;
    double x_cor = x[variable_index + 3];
    double y_cor = x[variable_index + 4];

    double x_diff = x_cor - SourceData.input_points[i].first;
    double y_diff = y_cor - SourceData.input_points[i].second;

    g[constraint_offset + i] = x_diff * x_diff + y_diff * y_diff;
  }
  return true;
}
```

2.2.2.7 eval_jac_g

约束函数的jacobian矩阵(以第i个约束为例):

$$func1: x_{i+1} - x_i - \frac{\triangle s_i}{2} \cdot \sum_{i=0}^{m-1} w_i \cdot cos(\theta(\frac{\triangle s_i}{2} \cdot \xi_i + \frac{\triangle s_i}{2}))$$

$$func2: y_{i+1} - y_i - \frac{\triangle s_i}{2} \cdot \sum_{i=0}^{m-1} w_i \cdot sin(\theta(\frac{\triangle s_i}{2} \cdot \xi_i + \frac{\triangle s_i}{2}))$$

$$func3: (x_i - x_{iref})^2 + (y_i - y_{iref})^2$$

$func1$以及$func2$与$x_i, x_{i+1}, \theta_i, \dot{\theta}_i, \ddot{\theta}_i, \theta_{i+1}, \dot{\theta}_{i+1}, \ddot{\theta}_{i+1}, \triangle s_i$有关。

因此按部就班的求偏导即可，具体偏导的求解前面2.2.2.5节提到过了，这里不再赘述了。

请对照代码查看。PS：在构建jacobian矩阵时，需要注意jacobian矩阵我们**先要去定义矩阵的非零项的位置**，然后**再把偏导填到对应位置！**

```
//具体内容对照代码看，思路很简单，核心就是上面公式中偏导的求法
bool SpiralSmoother::eval_jac_g(int n, const double* x, bool new_x,
                                int m, int nele_jac, int* iRow,
                                int* jCol, double* values) {
  //这里定义矩阵的非零项位置
  if (values == nullptr) {
    int nz_index = 0;

    int variable_offset = num_of_points_ * 5;
    for (int i = 0; i + 1 < num_of_points_; ++i) {
      int variable_index = i * 5;
```

```
      // theta0
      iRow[nz_index] = i * 2;
      jCol[nz_index] = variable_index + 0;
      ++nz_index;
      ...
    }
  }else{//这里填入具体的偏导值
     values[nz_index] += 2.0 * x_diff * (-pos_theta0.first);
     ++nz_index;
  }
}
```

3.总结

3.1 自动微分与手动微分的思考

可以看到算法的理论并不复杂，但是在写程序的时候就很复杂了，因为需要自己去手动微分求这么多的偏导数，又费时又费力还容易出错。

因此在使用ipopt库时，手动微分这个步骤新手很容易搞错，手动微分这么麻烦，而且现在市面上有这么多的自动微分库，为什么不用自动微分库呢？是因为自动微分库的适用函数很有限吗？我们这个优化问题用不了自动微分库？

解释：其实我们这个优化问题能用自动微分库。但是**自动微分库有两个缺点**：1.有些特殊的算子不齐全，这个问题我们倒是没有遇到（我们的模型并没有很复杂的函数😀）；2.速度比手动微分慢，这是核心的。

其实在初学的时候最好还是用手动微分，完成的走一遍对于算法的理解会更深刻一些，在熟悉了流程之后，如果后续再想快速验证优化模型，可以使用自动微分库，更加方便。

3.2 优化求解库自研

以下内容是我个人的思考，受限于个人水平，希望大家能够批判着看并发表各位自己的见解。

市面上这么多好用的优化求解库，确实速度挺快的，而且用起来很方便，但是我个人认为，这些库因为它的设计要适用于很多类型的非线性问题，它的目标是能解，还能广泛稳定的解各种问题，所以它的解法有可能并不是针对当前问题最优最快的。

如果我们仔细的去剖析各个非线性问题的求解，我们去观察这么多种解法到底有什么不同，举例一个最直观的一点大家能意识到的影响效率的就是：为什么牛顿法不如拟牛顿法？因为hessian矩阵很难求等问题，是的，那么假如刚好我这个问题的hessian矩阵不难求呢？是不是牛顿法反而更好呢？

带着上面的这个思考，所以我们应该产生一个猜想：我们自己所构建的优化问题，变量的系数，各种约束的矩阵，我们这些矩阵到底有什么性质，针对这些性质是否可以采取一种最适合它的解法，而不是大一统的采用同一种解法？ 所以我认为这可能就是自研优化求解库的意义：为我们自己独特的问题找到最适合它的求解方法，才能把速度与稳定性发挥到极致。

同时我们在构造优化问题的同时，为了提升模型的求解速度，可以尽可能的通过一些小trick去把有约束问题转化为无约束问题，比如：

对于不等式约束通过BIG-M方法放到目标函数中，对于等式约束通过转移矩阵先化为不等式约束然后再BIG-M放到目标函数中，这样把原问题转化为无约束优化问题，求解起来更快。