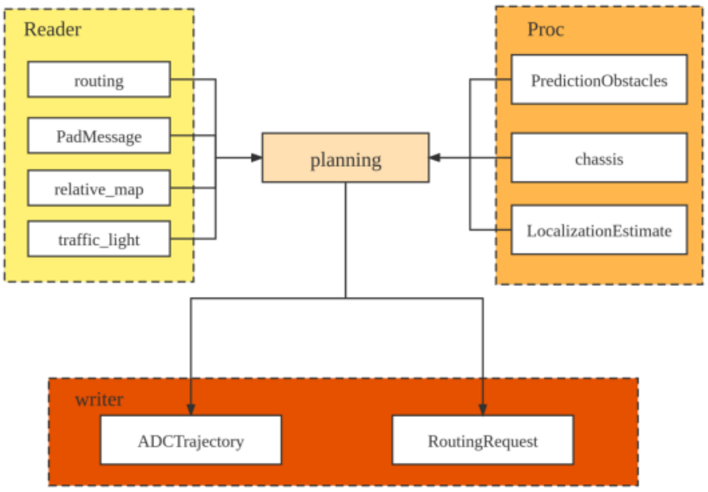
Apollo planning模块框架

planning 输入输出



输入：

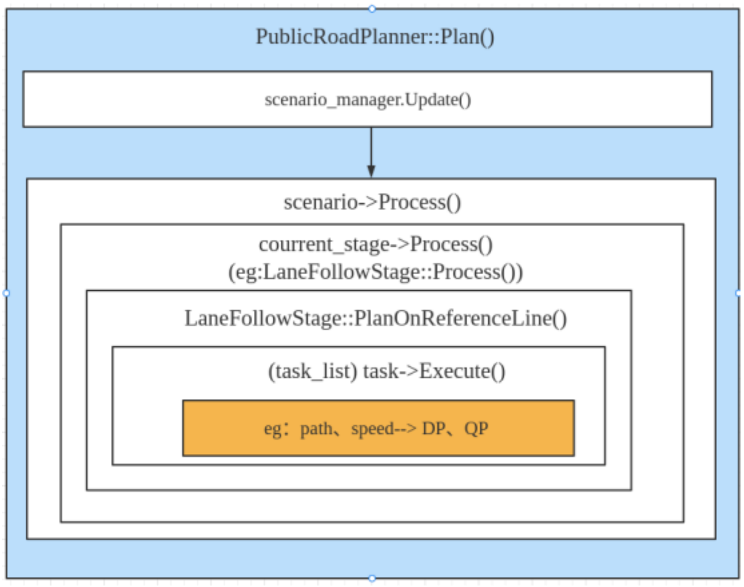
- `Reader` 消息订阅：`Hdmap`、`Routing`、`TrafficLight`、`PadMsg`
- `process` 入口调用：`Prediction`、`localization`、`Chassis`

输出：

- `ADCTrajectory`：发送给控制信号的自车局部轨迹信息

实现功能：

1. 启动ReferenceLineProvider来提供参考线，后面生成的轨迹都是在参考线的基础上做优化，ReferenceLineProvider启动了一个单独的线程，每隔50ms执行一次，和Planning主流程并行执行。
2. 执行Planning主流程。先选择对应的Planner,主线程上还是基于场景划分的思路，在配置文件中定义了Planner支持的场景(Scenario)，把规划分为具体的几个场景来执行，每个场景又分为几个阶段(Stage)，每个阶段会执行多个任务(Task)，任务执行完成后，对应的场景就完成了。不同场景间的切换是由一个状态机(ScenarioDispatch)来控制的。


PublicRoadPlanner::Plan()
scenario_manager.Update()
scenario->Process()
current_stage->Process()
(eg:LaneFollowStage::Process())
LaneFollowStage::PlanOnReferenceLine()
(task_list) task->Execute()
eg：path、speed--> DP、QP

> 多数场景下还是采用基于ReferenceLine的规划算法，对于泊车相关场景，则调用OpenSpacePlanning规划器

场景分类

蓝色框内的5种场景为大类，剩余的场景在这5大类中再细分做出判断。另外需要注意的是，蓝色框内的5种场景是有优先级顺序的，即如果判断为某种场景后，后续的场景也就不再判断。

多数场景下还是采用基于ReferenceLine的规划算法，对于泊车相关场景，则调用OpenSpacePlanning规划器。目前Apollo的场景划分为了16种，在proto文件中可以查看到

```
/modules/planning/proto/planning_config.proto 中定义的基本场景
// scenario configs
message ScenarioConfig {
  enum ScenarioType {
    LANE_FOLLOW = 0;  // default scenario
    // intersection involved 交叉路口
    BARE_INTERSECTION_UNPROTECTED = 2;
    STOP_SIGN_PROTECTED = 3;
    STOP_SIGN_UNPROTECTED = 4;
    TRAFFIC_LIGHT_PROTECTED = 5;
    TRAFFIC_LIGHT_UNPROTECTED_LEFT_TURN = 6;
    TRAFFIC_LIGHT_UNPROTECTED_RIGHT_TURN = 7;
    YIELD_SIGN = 8;

    // parking 停车
    PULL_OVER = 9; //靠边停车
    VALET_PARKING = 10;

    EMERGENCY_PULL_OVER = 11;
    EMERGENCY_STOP = 12;

    // misc 杂项
    NARROW_STREET_U_TURN = 13;
    PARK_AND_GO = 14;

    //apollo6.0中新增的几个场景（基于学习）
    //learning model sample
    LEARNING_MODEL_SAMPLE = 15;
    //true around
    DEADEND_TURNAROUND = 16;
  }
```

ParkAndGo 即停即走

用于路边停车，并开始生成到达下一个目的地的

PullOver 靠边停车

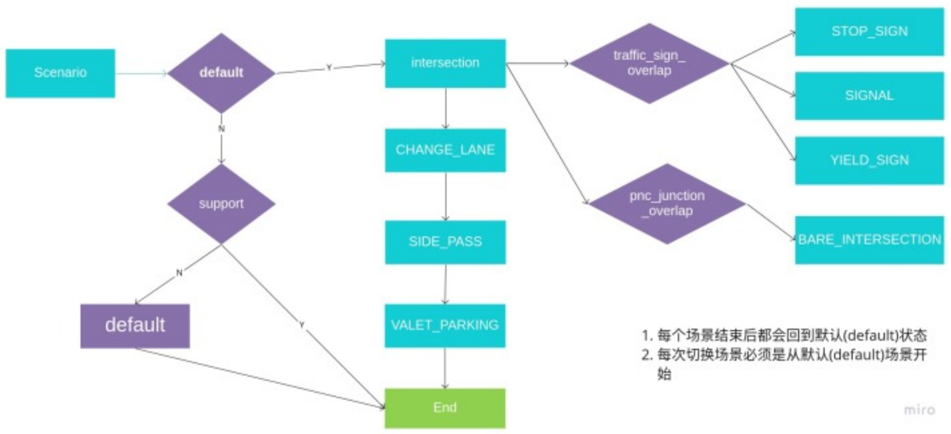
场景转换

场景转换的实现在"scenario_manager.cc"中，其中实现了场景注册，创建场景和更新场景的功能

```
bool ScenarioManager::Init(
    const std::set<ScenarioConfig::ScenarioType>& supported_scenarios) {
  // 注册场景
  RegisterScenarios();
  default_scenario_type_ = ScenarioConfig::LANE_FOLLOW;
  supported_scenarios_ = supported_scenarios;
  // 创建场景，默认为lane_follow
  current_scenario_ = CreateScenario(default_scenario_type_);
  return true;
}

// 更新场景
void ScenarioManager::Update(const common::TrajectoryPoint& ego_point,
                             const Frame& frame) {
  CHECK(!frame.reference_line_info().empty());
  // 保留当前帧
  Observe(frame);
  // 场景分发
  ScenarioDispatch(ego_point, frame);
}

// 通过一个有限状态机，决定当前的场景
void ScenarioManager::ScenarioDispatch(const common::TrajectoryPoint& ego_point,
                                       const Frame& frame) {
  ...
}
```



可以看到，每次切换场景必须是从默认场景(LANE_FOLLOW)开始，即每次场景切换之后都会回到默认场景。

场景定义

以LANE_FOLLOW 为例说明:

```
/modules/planning/conf/scenario/lane_follow_config.pb.txt
scenario_type: LANE_FOLLOW
stage_type: LANE_FOLLOW_DEFAULT_STAGE
stage_config: {
  stage_type: LANE_FOLLOW_DEFAULT_STAGE
  enabled: true
  task_type: LANE_CHANGE_DECIDER
  task_type: PATH_REUSE_DECIDER
  task_type: PATH_LANE_BORROW_DECIDER
  task_type: PATH_BOUNDS_DECIDER
  task_type: PIECEWISE_JERK_PATH_OPTIMIZER
  task_type: PATH_ASSESSMENT_DECIDER
  task_type: PATH_DECIDER
  task_type: RULE_BASED_STOP_DECIDER
  task_type: ST_BOUNDS_DECIDER
  task_type: SPEED_BOUNDS_PRIORI_DECIDER
  task_type: DP_ST_SPEED_OPTIMIZER
  task_type: SPEED_DECIDER
  task_type: SPEED_BOUNDS_FINAL_DECIDER
  # task_type: PIECEWISE_JERK_SPEED_OPTIMIZER
  task_type: PIECEWISE_JERK_NONLINEAR_SPEED_OPTIMIZER
  task_type: DECIDER_RSS

  task_config: {
    task_type: LANE_CHANGE_DECIDER
    lane_change_decider_config {
      enable_lane_change_urgency_check: true
    }
  }
  task_config: {
    task_type: PATH_REUSE_DECIDER
    path_reuse_decider_config {
      reuse_path: false
    }
  }
  ...
```

总结

基于优化的轨迹生成

横向轨迹优化

> Piecewise Jerk Path Optimizer

该task的代码位于`/modules/planning/tasks/optimizers/piecewise_jerk_path`,主要靠调用`modules/planning/math/piecewise_jerk` 下的类方法来帮助其完成二次规划问题的构造及求解工作。

```
/*
 * @brief:
 * FEM stands for finite element method.
 * This class solve an optimization problem:
 * x
 * |
 * |                       P(s1, x1)  P(s2, x2)
 * |            P(s0, x0)                       ... P(s(k-1), x(k-1))
 * |P(start)
 * |
 * |________________________________________________________ s
 *
 * we suppose s(k+1) - s(k) == s(k) - s(k-1)
 *
 * Given the x, x', x'' at P(start),  The goal is to find x0, x1, ... x(k-1)
 * which makes the line P(start), P0, P(1) ... P(k-1) "smooth".
 */
class PiecewiseJerkPathProblem : public PiecewiseJerkProblem {
  ...
};
```

```
/modules/planning/tasks/optimizers/piecewise_jerk_path
.
├── BUILD
├── piecewise_jerk_path_ipopt_solver.cc
├── piecewise_jerk_path_ipopt_solver.h
├── piecewise_jerk_path_optimizer.cc
└── piecewise_jerk_path_optimizer.h

modules/planning/math/piecewise_jerk
.
├── BUILD
├── piecewise_jerk_path_problem.cc
├── piecewise_jerk_path_problem.h #派生类
├── piecewise_jerk_problem.cc
├── piecewise_jerk_problem.h #基类
├── piecewise_jerk_speed_problem.cc
└── piecewise_jerk_speed_problem.h #派生类

#path优化和speed优化的约束条件是一致的，都是在基类中实现的那个约束条件构造函数
```

1 整体流程

1. **Process**
2. **OptimizePath** (task)
3. **set (Points,weight,DP_ref)**(目标点个数及坐标，各项优化目标权重，DP规划出来的path)
4. **Optimize**(优化函数的入口，设置默认迭代次数4000)
5. **FormulateProblem**(用于构造二次优化问题的具体矩阵，也就是将规划问题的求解条件转化为OSQP可求解形式的接口)
    1. **CalculateKernel()**
    2. **CalculateAffineConstraint()**
    3. **CalculateOffset()**
6. **SolverDefaultSettings**(默认配置的参数接口)
7. **osqp setup**(osqp库接口)
8. **osqp solve**(osqp求解接口)
9. **FreeData**(删除数据，释放内存)

    ```
    //osqp求解步骤
      OSQPData* data = FormulateProblem();
      OSQPSettings* settings = SolverDefaultSettings();
      settings->max_iter = max_iter;
      OSQPWorkspace* osqp_work = osqp_setup(data, settings);
      osqp_solve(osqp_work);
    ```

2 FormulateProblem()

> 二次规划问题中P、A是稀疏矩阵

1. **CalculateKernel**

    构造二次项系数矩阵p的压缩矩阵

2. **CalculateAffineConstraint**

   构造A矩阵以及上下边界lower_bounds和upper_bounds的压缩矩阵
3. **CalculateOffset**

   构造一次项系数矩阵q的压缩矩阵
4. **csc matrix**

将上述转换得到的矩阵压入OSQPData中

```
data->n = kernel_dim;
data->m = num_affine_constraint;
data->P = csc_matrix(kernel_dim, kernel_dim, P_data.size(), CopyData(P_data),
                     CopyData(P_indices), CopyData(P_indptr));
data->q = CopyData(q);
data->A =csc_matrix(num_affine_constraint, kernel_dim, A_data.size(),
                 CopyData(A_data), CopyData(A_indices), CopyData(A_indptr));
```

如何构造一个最优化问题?

以四个点(p1、p2、p3、p4)为例构造最优化问题

二次规划的一般形式

$$\begin{gathered} minimize\frac{1}{2} \cdot x^T \cdot P \cdot x + Q \cdot x \\ s.t. LB \leq A \cdot x \leq UB \end{gathered} \tag{1}$$

CalculateKernel()构造目标函数矩阵

1. `x`矩阵

- x矩阵即为需要优化的变量

$$x^T = |l_1\ l_2\ l_3\ l_4\ l'_1\ l'_2\ l'_3\ l'_4\ l''_1\ l''_2\ l''_3\ l''_4| \tag{2}$$

2. `p、q`矩阵

通过构造函数来构造`p、q`矩阵，其中代价函数分为三部分

1. 曲线平滑`l,l',l'',l'''`
2. 与参考线的偏差`(l - lref)`
3. 终点位置的软约束

总体代价函数公式

$$\begin{gathered} cost\ \ function = w_l \cdot \sum_{i=0}^{n-1} l_i^2 + w_{l'} \cdot \sum_{i=0}^{n-1} {l_i}'^2 + w_{l''} \cdot \sum_{i=0}^{n-1} {l_i}''^2 + w_{l'''} \cdot \sum_{i=0}^{n-2} (\frac{{l_{i+1}}'' - {l_i}''}{\Delta s})^2 + \\ w_{end_l} \cdot (l_{n-1} - l_{endref})^2 + w_{end_{dl}} \cdot (l'_{n-1} - {l_{endref}}')^2 + w_{end_{ddl}} \cdot (l''_{n-1} - {l_{endref}}'')^2 + \\ w_{ref} \cdot \sum_{i=0}^{n-1} (l_i - l_{ref})^2 + \\ w_{end_l} \cdot (l_{n-1} - l_{endref})^2 + w_{end_{dl}} \cdot (l'_{n-1} - {l_{endref}}')^2 + w_{end_{ddl}} \cdot (l''_{n-1} - {l_{endref}}'')^2 \end{gathered} \tag{3}$$

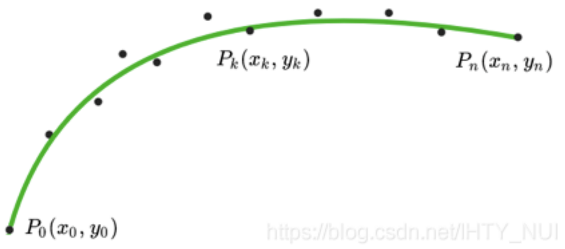
①曲线平滑的`cost`

$$w_l \cdot \sum_{i=0}^{n-1} l_i^2 + w_{l'} \cdot \sum_{i=0}^{n-1} {l_i}'^2 + w_{l''} \cdot \sum_{i=0}^{n-1} {l_i}''^2 + w_{l'''} \cdot \sum_{i=0}^{n-2} (\frac{{l_{i+1}}'' - {l_i}''}{\Delta s})^2 \tag{4}$$

- 转化为p矩阵(12*12)，记为`p1`

$$p1 = \begin{vmatrix} w_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{l'} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{l'} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{l'} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{l'} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{l''} + \frac{w_{l'''}}{\Delta s^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2\frac{w_{l'''}}{\Delta s^2} & w_{l''} + 2\frac{w_{l'''}}{\Delta s^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2\frac{w_{l'''}}{\Delta s^2} & w_{l''} + 2\frac{w_{l'''}}{\Delta s^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2\frac{w_{l'''}}{\Delta s^2} & w_{l''} + \frac{w_{l'''}}{\Delta s^2} \end{vmatrix} \tag{5}$$

②与参考线偏差



$$w_{ref} \cdot \sum_{i=0}^{n-1} (l_i - l_{ref})^2 \tag{6}$$

- 二次项转化为p矩阵(4*12)，记为`p2`

$$p2 = \begin{vmatrix} w_{ref_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{ref_2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{ref_3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{ref_4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix} \tag{7}$$

- 一次项转化为q矩阵(4*1)，记为`q1`

  注：去掉上述约束方程的常量项

$$q1 = \begin{vmatrix} -2w_{ref_1} \cdot l_{ref_1} \\ -2w_{ref_2} \cdot l_{ref_2} \\ -2w_{ref_3} \cdot l_{ref_3} \\ -2w_{ref_4} \cdot l_{ref_4} \end{vmatrix} \tag{8}$$

③终点

$$w_{end_l} \cdot (l_{n-1} - l_{endref})^2 + w_{end_{dl}} \cdot (l'_{n-1} - l'_{endref})^2 + w_{end_{ddl}} \cdot (l''_{n-1} - l''_{endref})^2 \tag{9}$$

- 二次项转化为p矩阵(12*12)，记为`p3`

$$p3 = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{end_l} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{end_{dl}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{end_{ddl}} \end{vmatrix} \tag{10}$$

- 一次项转化为q矩阵(12*1)，记为`q2`

$$q2 = \begin{vmatrix} 0 \\ 0 \\ 0 \\ -2w_{end_l} \cdot endl \\ 0 \\ 0 \\ 0 \\ -2w_{end_{dl}} \cdot enddl \\ 0 \\ 0 \\ 0 \\ -2w_{end_{ddl}} \cdot endddl \end{vmatrix} \tag{11}$$

3.构造优化目标函数

综合`x, p, q`可得到`cost function`:

$$minimize\ f(l(s)) = x^T_{(1*12)}(p_1^T p_1 + p_2^T p_2 + p_3^T p_3)_{(12*12)} x_{(12*1)} + (q_1^T + q_2^T)_{(1*12)} x_{(12*1)} \tag{12}$$

记:

$$\begin{gathered} P_{(12*12)} = p_1^T p_1 + p_2^T p_2 + p_3^T p_3 \\ Q_{(1*12)} = q_1^T + q_2^T \end{gathered} \tag{13}$$

得到最终目标函数的表达式:

$$minimize\ f(l(s)) = x^T P x + Q x \tag{14}$$

其中:

$$x^T_{(1*12)} = |l_1\ l_2\ l_3\ l_4\ l'_1\ l'_2\ l'_3\ l'_4\ l''_1\ l''_2\ l''_3\ l''_4| \tag{15}$$

$$P_{(12*12)}=\left|\begin{matrix} w_l+w_{ref_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_l+w_{ref_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_l+w_{ref_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_l+w_{ref_1}+w_{end_l} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{l'} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{l'} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{l'} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{l'}+w_{end_{dl}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{l''}+\frac{w_{l'''}}{\Delta s^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2\frac{w_{l'''}}{\Delta s^2} & w_{l''}+2\frac{w_{l'''}}{\Delta s^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2\frac{w_{l'''}}{\Delta s^2} & w_{l''}+2\frac{w_{l'}}{\Delta s} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2\frac{w_{l'''}}{\Delta s^2} \end{matrix}\right.$$

$$Q_{(12*1)}=\begin{bmatrix} -2w_{ref_1}\cdot l_{ref_1} \\ -2w_{ref_2}\cdot l_{ref_2} \\ -2w_{ref_3}\cdot l_{ref_3} \\ -2w_{ref_4}\cdot l_{ref_4}-2w_{end_l}\cdot endl \\ 0 \\ 0 \\ 0 \\ -2w_{end_{dl}}\cdot enddl \\ 0 \\ 0 \\ 0 \\ -2w_{end_{ddl}}\cdot endddl \end{bmatrix} \qquad (17)$$

扩展到n个点?

假设约束维度任然是二阶(l、l'、l''），那么上述的P矩阵为(3n* 3n)，Q矩阵为(3n* * 1)

$$x^T=|l_1\ \dots l_n\ l_1'\ \dots l_n'\ l_1''\dots\ l_n''|_{3n\times 1} \qquad (18)$$

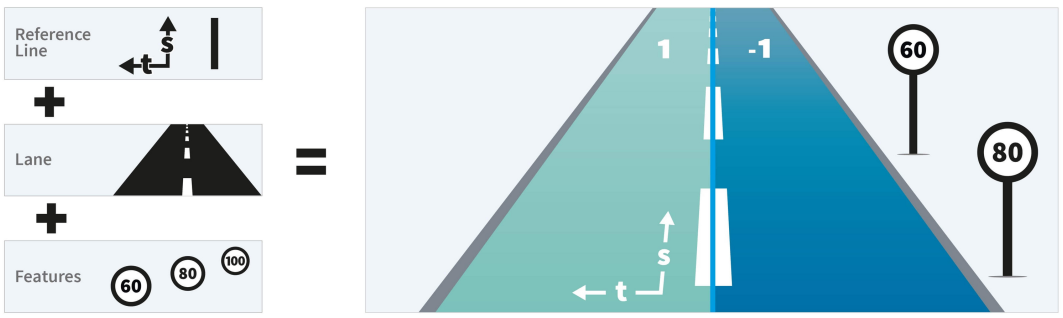
$$P=\left[\begin{matrix} \begin{bmatrix} w_l+w_{ref_1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & w_l+w_{ref_1}+w_{end_l} \end{bmatrix}_{n\times n} & & \\ & \begin{bmatrix} w_{l'} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & w_{l'}+w_{end_{dl}} \end{bmatrix}_{n\times n} & \\ & & \begin{bmatrix} w_{l''}+\frac{w_{l'''}}{\Delta s^2} & 0 & \dots & \dots \\ -2\frac{w_{l'''}}{\Delta s^2} & w_{l''}+\frac{w_{l'''}}{\Delta s^2} & 0 & \dots \\ 0 & -2\frac{w_{l'''}}{\Delta s^2} & \ddots & \dots \\ \vdots & \ddots & \ddots & \\ 0 & \dots & -2\frac{w_{l'''}}{\Delta s^2} & w_{l''}+\frac{w_{l'''}}{\Delta s^2}+w_{ena} \end{bmatrix} \end{matrix}\right.$$

$$Q=\begin{bmatrix} \begin{bmatrix} -2w_{ref_1}\cdot l_{ref_1} \\ \vdots \\ -2w_{ref_4}\cdot l_{ref_4}-2w_{end_l}\cdot endl \end{bmatrix}_{n\times 1} \\ \begin{bmatrix} 0 \\ \vdots \\ -2w_{end_{dl}}\cdot enddl \end{bmatrix}_{n\times 1} \\ \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ -2w_{end_{ddl}}\cdot endddl \end{bmatrix}_{n\times 1} \end{bmatrix}_{3n\times 1} \qquad (20)$$

CalculateAffineConstraint()构造约束矩阵

1.对l的约束

车辆行驶位置，即对道路边界的约束



```
DEFINE_double(longitudinal_jerk_lower_bound, -4.0,
              "The lower bound of longitudinal jerk.");
DEFINE_double(longitudinal_jerk_upper_bound, 2.0,
              "The upper bound of longitudinal jerk.");
```

构造矩阵(4*12):

$$\begin{bmatrix} lb_{s1} \\ lb_{s2} \\ lb_{s3} \\ lb_{s4} \end{bmatrix} \leq \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x \leq \begin{bmatrix} ub_{s1} \\ ub_{s2} \\ ub_{s3} \\ ub_{s4} \end{bmatrix} \tag{21}$$

2.对l'的约束

轨迹的一阶导为heading，可以近似理解为横向运动的“速度”，希望不要横向走的太快

```
DEFINE_double(lateral_derivative_bound_default, 2.0,
              "the default value for lateral derivative bound.");
```

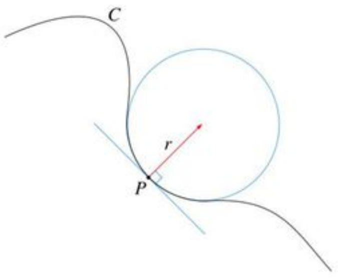
构造矩阵(4*12):

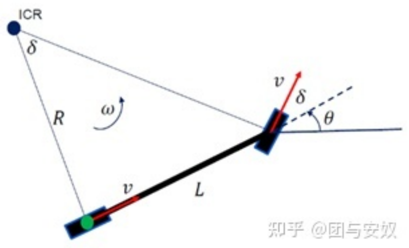
$$\begin{bmatrix} -2.0 \\ -2.0 \\ -2.0 \\ -2.0 \end{bmatrix} \leq \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} x \leq \begin{bmatrix} 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \end{bmatrix} \tag{22}$$

3.对l''的约束

轨迹的二阶导可以近似理解为横向运动的“加速度”，希望方向盘不要打得太猛

因为车辆存在最小的转弯半径,即存在最大行驶曲率，所以要对车辆运动学进行限制。



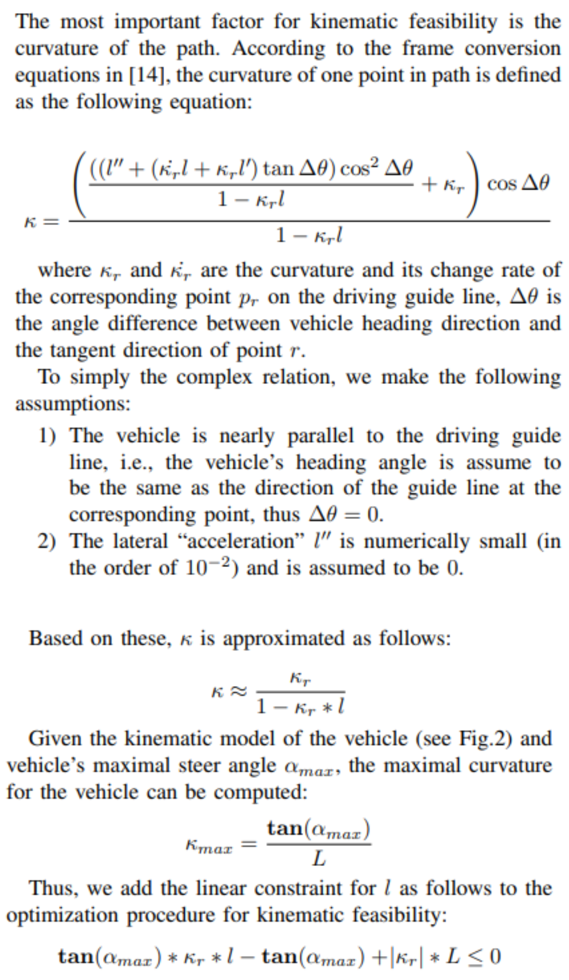


$$K_{max} = \frac{1}{R} = \frac{tan(\delta_{max})}{L} = \frac{tan(\frac{maxSteerAngle}{steerRadio})}{L} \tag{23}$$
$$\text{其中：}R\text{为车辆的最大转弯半径，}L\text{为车辆轴距}$$
$$\delta_{max}\text{为前轮最大转角，}steerRadio\text{为车辆转向传动比}$$

问题(bug)

代码中的意思是直接把`l''`等价为当前轨迹的曲率k，而实际的k值计算如下文所示

- 在百度发表的论文中对车辆曲率的约束如下：[Optimal Vehicle Path Planning Using Quadratic Optimization for Baidu Apollo Open Platform]()



The most important factor for kinematic feasibility is the curvature of the path. According to the frame conversion equations in [14], the curvature of one point in path is defined as the following equation:

$$\kappa = \frac{\left(\frac{((l'' + (\dot{\kappa_r} l + \kappa_r l')\tan\Delta\theta)\cos^2\Delta\theta}{1-\kappa_r l} + \kappa_r\right)\cos\Delta\theta}{1-\kappa_r l}$$

where κ_r and $\dot{\kappa_r}$ are the curvature and its change rate of the corresponding point p_r on the driving guide line, $\Delta\theta$ is the angle difference between vehicle heading direction and the tangent direction of point r.

To simply the complex relation, we make the following assumptions:

1) The vehicle is nearly parallel to the driving guide line, i.e., the vehicle's heading angle is assume to be the same as the direction of the guide line at the corresponding point, thus $\Delta\theta = 0$.
2) The lateral "acceleration" l'' is numerically small (in the order of 10^{-2}) and is assumed to be 0.

Based on these, κ is approximated as follows:

$$\kappa \approx \frac{\kappa_r}{1-\kappa_r * l}$$

Given the kinematic model of the vehicle (see Fig.2) and vehicle's maximal steer angle α_{max}, the maximal curvature for the vehicle can be computed:

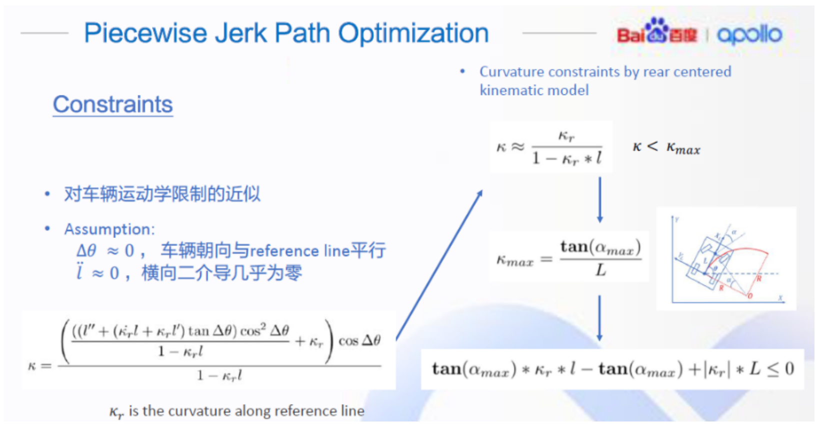
$$\kappa_{max} = \frac{\tan(\alpha_{max})}{L}$$

Thus, we add the linear constraint for l as follows to the optimization procedure for kinematic feasibility:

$$\tan(\alpha_{max}) * \kappa_r * l - \tan(\alpha_{max}) + |\kappa_r| * L \le 0$$

$$\kappa_r\kappa\text{和}\kappa r^{\cdot}\dot{\kappa_r}\text{是参考线在}p_r\text{处的曲率和曲率变化率，}\Delta\theta\text{是车辆和参考线点}p_r\text{处切线方向的角度差。}$$
$$\text{简化：假设车辆几乎在沿着道路方向行驶，因此}\Delta\theta = 0\text{；"横向加速度"}l''\text{是很小的，数量级在}10^{-2}\text{，因此}l'' = 0 \tag{24}$$

- 百度公开课讲解的曲率约束内容如下



Piecewise Jerk Path Optimization

Baidu百度 | apollo

Constraints

- 对车辆运动学限制的近似
- Assumption:
  $\Delta\theta \approx 0$，车辆朝向与reference line平行
  $\ddot{l} \approx 0$，横向二介导几乎为零

$$\kappa = \frac{\left(\frac{((l'' + (\dot{\kappa_r} l + \kappa_r l')\tan\Delta\theta)\cos^2\Delta\theta}{1-\kappa_r l} + \kappa_r\right)\cos\Delta\theta}{1-\kappa_r l}$$

κ_r is the curvature along reference line

- Curvature constraints by rear centered kinematic model

$$\kappa \approx \frac{\kappa_r}{1-\kappa_r * l} \qquad \kappa < \kappa_{max}$$

$$\kappa_{max} = \frac{\tan(\alpha_{max})}{L}$$

$$\tan(\alpha_{max}) * \kappa_r * l - \tan(\alpha_{max}) + |\kappa_r| * L \le 0$$

- 但是在代码中A矩阵的赋值根据代码直接设为1，也就是说`l''`直接等于了当前车辆的行驶曲率k
  - 给A矩阵赋值

```cpp
int constraint_index = 0;
// set x, x', x'' bounds
for (int i = 0; i < num_of_variables; ++i) {
  if (i < n) {
    variables[i].emplace_back(constraint_index, 1.0);
    lower_bounds->at(constraint_index) =
        x_bounds_[i].first * scale_factor_[0];
    upper_bounds->at(constraint_index) =
        x_bounds_[i].second * scale_factor_[0];
  } else if (i < 2 * n) {
    variables[i].emplace_back(constraint_index, 1.0);

    lower_bounds->at(constraint_index) =
        dx_bounds_[i - n].first * scale_factor_[1];
    upper_bounds->at(constraint_index) =
        dx_bounds_[i - n].second * scale_factor_[1];
  } else {
    variables[i].emplace_back(constraint_index, 1.0);
    lower_bounds->at(constraint_index) =
        ddx_bounds_[i - 2 * n].first * scale_factor_[2];
    upper_bounds->at(constraint_index) =
        ddx_bounds_[i - 2 * n].second * scale_factor_[2];
  }
  ++constraint_index;
}

//给A矩阵赋值
int ind_p = 0;
for (int i = 0; i < num_of_variables; ++i) {
  A_indptr->push_back(ind_p);
  for (const auto& variable_nz : variables[i]) {
    // coefficient
    A_data->push_back(variable_nz.second);

    // constraint index
    A_indices->push_back(variable_nz.first);
    ++ind_p;
  }
```

- 上下边界赋值

```
//车辆运动学约束，由车轮最大转角推导行驶过程中的最大曲率
const double lat_acc_bound =
        std::tan(veh_param.max_steer_angle() / veh_param.steer_ratio()) /
        veh_param.wheel_base();

//要考虑道路的曲率,所以要减去道路的kappa值
double kappa = reference_line.GetNearestReferencePoint(s).kappa();
      ddl_bounds.emplace_back(-lat_acc_bound - kappa, lat_acc_bound - kappa);
```

根据代码构造出矩阵如下(4*12):

$$\begin{bmatrix} -lat\_acc\_bound - kappa_{s1} \\ -lat\_acc\_bound - kappa_{s2} \\ -lat\_acc\_bound - kappa_{s3} \\ -lat\_acc\_bound - kappa_{s4} \end{bmatrix} \le \begin{bmatrix} 0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \end{bmatrix} x \le \begin{bmatrix} lat\_acc\_bound - kappa_{s1} \\ lat\_acc\_bound - kappa_{s2} \\ lat\_acc\_bound - kappa_{s3} \\ lat\_acc\_bound - kappa_{s4} \end{bmatrix} \tag{25}$$

上述矩阵相当于直接把了**l''等价为曲率**,而实际k的约束如论文中所示，两个地方约束计算并不相同。

$$tan(\delta_{max}) \times k_r \times l - tan(\delta_{max}) + k_r \times L \le 0 \tag{26}$$

4.对l'''的约束

由差分求导可得到轨迹的三阶导数，可以理解为人打方向盘的加速度，此时是对`jerk`的约束，`delta_s_ = 1.0;`

$$l''' = \frac{l''_{i+1} - l''_i}{\Delta s} \tag{27}$$

横摆角速度:

$$yaw\_rate = \frac{(w_1 - w_2) \cdot R_r}{A} \tag{28}$$

```
double PiecewiseJerkPathOptimizer::EstimateJerkBoundary(
    const double vehicle_speed, const double axis_distance,
    const double max_yaw_rate) const {
  return max_yaw_rate / axis_distance / vehicle_speed;
}
```

$$\begin{bmatrix} -jerk_1 * \Delta s \\ -jerk_2 * \Delta s \\ -jerk_3 * \Delta s \end{bmatrix} \le \begin{bmatrix} 0&0&0&0&0&0&0&0&-1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&-1&1&0 \\ 0&0&0&0&0&0&0&0&0&0&-1&1 \end{bmatrix} x \le \begin{bmatrix} jerk_1 * \Delta s \\ jerk_2 * \Delta s \\ jerk_3 * \Delta s \end{bmatrix} \tag{29}$$

5.对起点p1的约束

起点必须在初始点的位置

$$\begin{bmatrix} ego_l \\ ego_{dl} \\ ego_{ddl} \end{bmatrix} \le \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0 \end{bmatrix} x \le \begin{bmatrix} ego_l \\ ego_{dl} \\ ego_{ddl} \end{bmatrix} \tag{30}$$

6.路径连续性约束

```
  // x(i+1)' - x(i)' - 0.5 * delta_s * x(i)'' - 0.5 * delta_s * x(i+1)'' = 0
  for (int i = 0; i + 1 < n; ++i) {
    ...
    lower_bounds->at(constraint_index) = 0.0;
    upper_bounds->at(constraint_index) = 0.0;
    ++constraint_index;
```

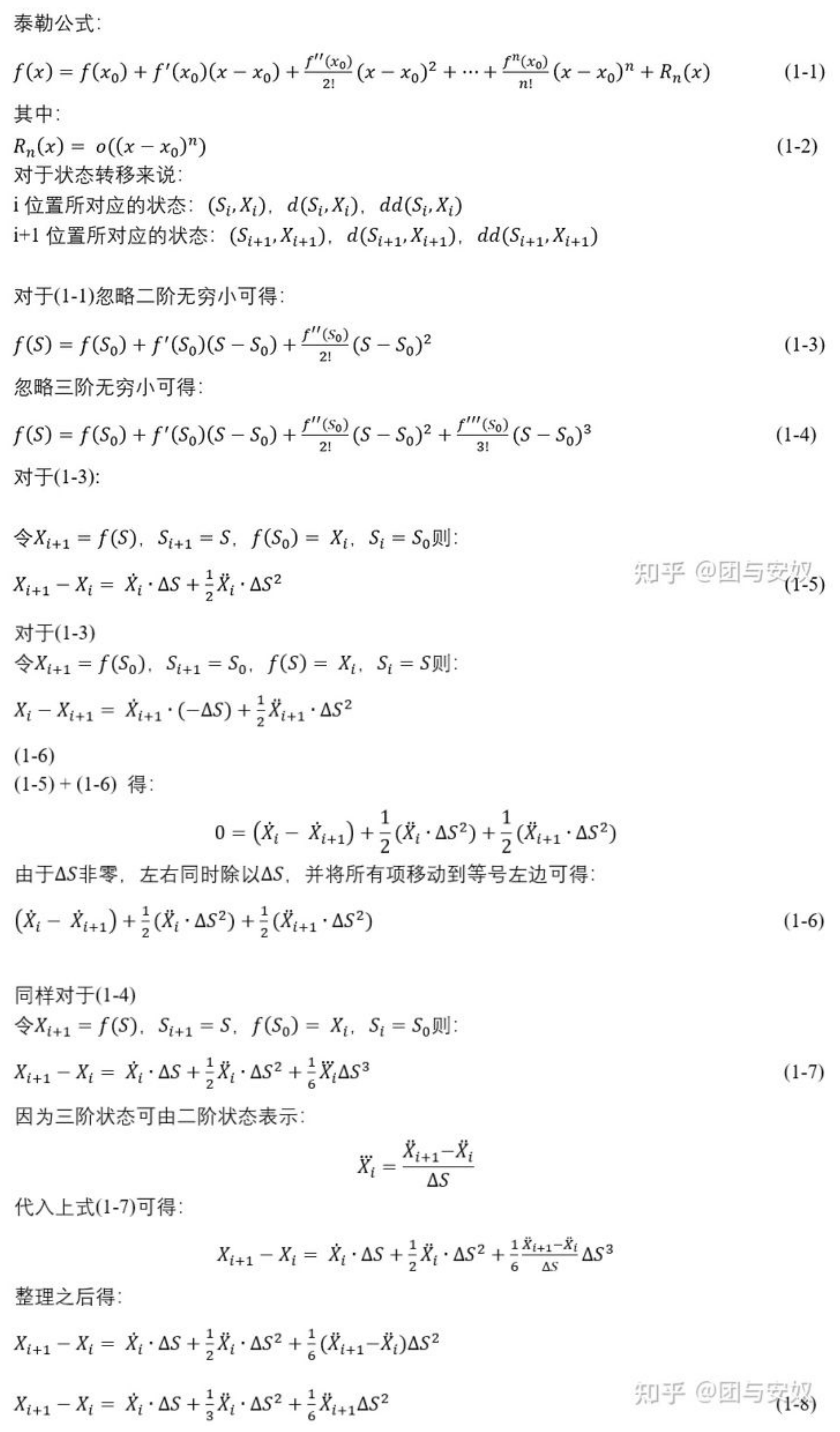
```
}

// x(i+1) - x(i) - delta_s * x(i)'
// - 1/3 * delta_s^2 * x(i)'' - 1/6 * delta_s^2 * x(i+1)''
auto delta_s_sq_ = delta_s_ * delta_s_;
for (int i = 0; i + 1 < n; ++i) {
  ..._{(12*1)}

  lower_bounds->at(constraint_index) = 0.0;
  upper_bounds->at(constraint_index) = 0.0;
  ++constraint_index;
}
```

对上述代码中的公式推导：

> 将零阶状态用一二阶状态进行线性表示，使其更为合理地表示**各界状态的关联关系**，确保**路径路径的连续性**。轨迹的三阶导数会随着二阶导数的变化而变化，但两点的三阶导保持相等



泰勒公式：

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^n(x_0)}{n!}(x - x_0)^n + R_n(x) \qquad (1\text{-}1)$$

其中：

$$R_n(x) = o((x - x_0)^n) \qquad (1\text{-}2)$$

对于状态转移来说：

i 位置所对应的状态：(S_i, X_i)，$d(S_i, X_i)$，$dd(S_i, X_i)$

i+1 位置所对应的状态：(S_{i+1}, X_{i+1})，$d(S_{i+1}, X_{i+1})$，$dd(S_{i+1}, X_{i+1})$

对于(1-1)忽略二阶无穷小可得：

$$f(S) = f(S_0) + f'(S_0)(S - S_0) + \frac{f''(S_0)}{2!}(S - S_0)^2 \qquad (1\text{-}3)$$

忽略三阶无穷小可得：

$$f(S) = f(S_0) + f'(S_0)(S - S_0) + \frac{f''(S_0)}{2!}(S - S_0)^2 + \frac{f'''(S_0)}{3!}(S - S_0)^3 \qquad (1\text{-}4)$$

对于(1-3)：

令$X_{i+1} = f(S)$，$S_{i+1} = S$，$f(S_0) = X_i$，$S_i = S_0$则：

$$X_{i+1} - X_i = \dot{X}_i \cdot \Delta S + \frac{1}{2}\ddot{X}_i \cdot \Delta S^2 \qquad (1\text{-}5)$$

对于(1-3)

令$X_{i+1} = f(S_0)$，$S_{i+1} = S_0$，$f(S) = X_i$，$S_i = S$则：

$$X_i - X_{i+1} = \dot{X}_{i+1} \cdot (-\Delta S) + \frac{1}{2}\ddot{X}_{i+1} \cdot \Delta S^2$$

(1-6)

(1-5) + (1-6) 得：

$$0 = \left(\dot{X}_i - \dot{X}_{i+1}\right) + \frac{1}{2}(\ddot{X}_i \cdot \Delta S^2) + \frac{1}{2}(\ddot{X}_{i+1} \cdot \Delta S^2)$$

由于ΔS非零，左右同时除以ΔS，并将所有项移动到等号左边可得：

$$\left(\dot{X}_i - \dot{X}_{i+1}\right) + \frac{1}{2}(\ddot{X}_i \cdot \Delta S^2) + \frac{1}{2}(\ddot{X}_{i+1} \cdot \Delta S^2) \qquad (1\text{-}6)$$

同样对于(1-4)

令$X_{i+1} = f(S)$，$S_{i+1} = S$，$f(S_0) = X_i$，$S_i = S_0$则：

$$X_{i+1} - X_i = \dot{X}_i \cdot \Delta S + \frac{1}{2}\ddot{X}_i \cdot \Delta S^2 + \frac{1}{6}\dddot{X}_i \Delta S^3 \qquad (1\text{-}7)$$

因为三阶状态可由二阶状态表示：

$$\dddot{X}_i = \frac{\ddot{X}_{i+1} - \ddot{X}_i}{\Delta S}$$

代入上式(1-7)可得：

$$X_{i+1} - X_i = \dot{X}_i \cdot \Delta S + \frac{1}{2}\ddot{X}_i \cdot \Delta S^2 + \frac{1}{6}\frac{\ddot{X}_{i+1} - \ddot{X}_i}{\Delta S}\Delta S^3$$

整理之后得：

$$X_{i+1} - X_i = \dot{X}_i \cdot \Delta S + \frac{1}{2}\ddot{X}_i \cdot \Delta S^2 + \frac{1}{6}(\ddot{X}_{i+1} - \ddot{X}_i)\Delta S^2$$

$$X_{i+1} - X_i = \dot{X}_i \cdot \Delta S + \frac{1}{3}\ddot{X}_i \cdot \Delta S^2 + \frac{1}{6}\ddot{X}_{i+1}\Delta S^2 \qquad (1\text{-}8)$$

通过上述(1-6)(1-8)可得到:

$$\begin{gathered} l'_{i+1} - l'_i - \frac{1}{2}\Delta s * l''_i - \frac{1}{2}\Delta s * l''_{i+1} = 0 \\ l_{i+1} - l_i - \Delta s \cdot l'_i - \frac{1}{3}\Delta s^2 \cdot l''_i - \frac{1}{6}\Delta s^2 \cdot l''_{i+1} = 0 \end{gathered} \tag{31}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \le \begin{bmatrix} 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -\frac{1}{2}\cdot\Delta s & -\frac{1}{2}\cdot\Delta s & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -\frac{1}{2}\cdot\Delta s & -\frac{1}{2}\cdot\Delta s & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -\frac{1}{2}\cdot\Delta s & -\frac{1}{2}\cdot\Delta s \end{bmatrix} x \le \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{32}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \le \begin{bmatrix} -1 & 1 & 0 & 0 & -\Delta s & 0 & 0 & 0 & -\frac{1}{3}\Delta s^2 & -\frac{1}{6}\Delta s^2 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & -\Delta s & 0 & 0 & 0 & -\frac{1}{3}\Delta s^2 & -\frac{1}{6}\Delta s^2 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & -\Delta s & 0 & 0 & 0 & -\frac{1}{3}\Delta s^2 & -\frac{1}{6}\Delta s^2 \end{bmatrix} x \le \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{33}$$

7.构造约束条件

$$x^T = |l_1\ l_2\ l_3\ l_4\ l'_1\ l'_2\ l'_3\ l'_4\ l''_1\ l''_2\ l''_3\ l''_4|_{1\times 12} \tag{34}$$

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -\frac{1}{2}\cdot\Delta s & -\frac{1}{2}\cdot\Delta s & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -\frac{1}{2}\cdot\Delta s & -\frac{1}{2}\cdot\Delta s & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -\frac{1}{2}\cdot\Delta s & -\frac{1}{2}\cdot\Delta s \\ -1 & 1 & 0 & 0 & -\Delta s & 0 & 0 & 0 & -\frac{1}{3}\Delta s^2 & -\frac{1}{6}\Delta s^2 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & -\Delta s & 0 & 0 & 0 & -\frac{1}{3}\Delta s^2 & -\frac{1}{6}\Delta s^2 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & -\Delta s & 0 & 0 & 0 & -\frac{1}{3}\Delta s^2 & -\frac{1}{6}\Delta s^2 \end{bmatrix}_{24\times 12} \tag{35}$$

$$LB = \begin{bmatrix} lb_{s1} \\ lb_{s2} \\ lb_{s3} \\ lb_{s4} \\ -2.0 \\ -2.0 \\ -2.0 \\ -2.0 \\ -lat\_acc\_bound - kappa_{s1} \\ -lat\_acc\_bound - kappa_{s2} \\ -lat\_acc\_bound - kappa_{s3} \\ -lat\_acc\_bound - kappa_{s4} \\ -jerk_1 * \Delta s \\ -jerk_2 * \Delta s \\ -jerk_3 * \Delta s \\ ego_l \\ ego_{dl} \\ ego_{ddl} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{24\times 1} = \begin{bmatrix} Lbsi_{(4*1)} \\ Heading1_{(4*1)} \\ Kappa1_{(4*1)} \\ Jerk1_{(3*1)} \\ Ego1_{(3*1)} \\ Continuous1_{(6*1)} \end{bmatrix}; UB = \begin{bmatrix} ub_{s1} \\ ub_{s2} \\ ub_{s3} \\ ub_{s4} \\ 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ lat\_acc\_bound - kappa_{s1} \\ lat\_acc\_bound - kappa_{s2} \\ lat\_acc\_bound - kappa_{s3} \\ lat\_acc\_bound - kappa_{s4} \\ jerk_1 * \Delta s \\ jerk_2 * \Delta s \\ jerk_3 * \Delta s \\ ego_l \\ ego_{dl} \\ ego_{ddl} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{24\times 1} = \begin{bmatrix} Ubsi_{(4*1)} \\ Heading2_{(4*1)} \\ Kappa2_{(4*1)} \\ Jerk2_{(3*1)} \\ Ego2_{(3*1)} \\ Continuous2_{(6*1)} \end{bmatrix} \qquad (3$$

综上得到约束条件:

$$LB_{(12*1)} \le A_{(24*12)} x_{(12*1)} \le UB_{(12*1)} \tag{37}$$

扩展到n个点?

$$A=\begin{bmatrix}
1 & \dots & \dots & 0 \\
\vdots & \ddots & & \vdots \\
\vdots & & \ddots & \vdots \\
0 & \dots & \dots & 1 \\
 & & & & 1 & \dots & \dots & 0 \\
 & & & & \vdots & \ddots & & \vdots \\
 & & & & \vdots & & \ddots & \vdots \\
 & & & & 0 & \dots & \dots & 1 \\
 & & & & & & & & 1 & \dots & \dots & 0 \\
 & & & & & & & & \vdots & \ddots & & \vdots \\
 & & & & & & & & \vdots & & \ddots & \vdots \\
 & & & & & & & & 0 & \dots & \dots & 1 \\
 & & & & & & & & -1 & 1 & \dots & 0 \\
 & & & & & & & & \vdots & \ddots & \ddots & \vdots \\
 & & & & & & & & \vdots & & \ddots & \ddots \\
 & & & & & & & & 0 & \dots & -1 & 1 \\
 \\
1 & \dots & \dots & 0 \\
0 & \dots & & \dots & 1 & \dots & \dots & 0 \\
0 & \dots & & & & & & \dots & 1 & \dots & \dots & 0 \\
 \\
0 & \dots & \dots & 0 & -1 & 1 & \dots & \dots & -\frac{1}{2}\cdot\Delta s & -\frac{1}{2}\cdot\Delta s & \dots & \dots \\
0 & \dots & & \dots & 0 & -1 & 1 & \dots & \dots & -\frac{1}{2}\cdot\Delta s & -\frac{1}{2}\cdot\Delta s & \dots \\
\vdots & \dots & & & \dots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \dots & & & & \dots & 0 & -1 & 1 & \dots & \cdots-\frac{1}{2}\cdot\Delta s & -\frac{1}{2}\cdot\Delta s \\
 \\
-1 & 1 & \dots & \dots & -\Delta s & 0 & 0 & 0 & -\frac{1}{3}\Delta s^2 & -\frac{1}{6}\Delta s^2 & \dots & \dots \\
0 & -1 & 1 & \dots & \dots & -\Delta s & \dots & \dots & \dots & -\frac{1}{3}\Delta s^2 & -\frac{1}{6}\Delta s^2 & \dots \\
\vdots & \dots & \ddots & \ddots & \dots & \dots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \dots & -1 & 1 & \dots & \dots & -\Delta s & 0 & 0 & 0 & -\frac{1}{3}\Delta s^2 & -\frac{1}{6}\Delta s^2
\end{bmatrix}_{6n\times 3n} \tag{38}$$

$$LB=\begin{bmatrix} Lbsi_{(n*1)} \\ Heading1_{(n*1)} \\ Kappa1_{(n*1)} \\ Jerk1_{((n-1)*1)} \\ Ego1_{(3*1)} \\ Continuous1_{((2n-2)*1)} \end{bmatrix}_{6n\times 1} \qquad UB=\begin{bmatrix} Ubsi_{(n*1)} \\ Heading2_{(n*1)} \\ Kappa2_{(n*1)} \\ Jerk2_{((n-1)*1)} \\ Ego2_{(3*1)} \\ Continuous2_{((2n-2)*1)} \end{bmatrix}_{6n\times 1} \tag{39}$$

总结

假设有n个点，优化维度为三维(l、l'、l''），通过构造`P,Q,A,LB,UB`矩阵方程，将此问题转化为二次规划问题

$$\begin{gathered} minimize\ f(l(s)) = x^TPx + Qx \\ s.t.\ \ LB \le Ax \le UB \end{gathered} \tag{40}$$

其中:

$$\begin{gathered} x^T = |l_1\ \dots l_n\ l'_1\ \dots l'_n\ l''_1 \dots\ l''_n|_{3n\times 1} \\ P = [\dots]_{3n\times 3n} \\ Q = [\dots]_{3n\times 1} \\ A = [\dots]_{6n\times 3n} \\ LB = [\dots]_{6n\times 1} \\ UB = [\dots]_{6n\times 1} \end{gathered} \tag{41}$$

Picewise Jerk Speed Optimizer

纵向速度轨迹优化

SL规划保证车辆的横向偏移足够平滑，ST规划保证车辆的前进方向速度变化足够平滑.

1. x矩阵

x矩阵即为需要优化的变量

$$x^T = |s_1 \ \dots s_n \ s_1' \ \dots s_n' \ s_1'' \dots s_n''| \tag{42}$$

2. p、q矩阵

跟path optimizer区别在于p矩阵，speed多了对参考线偏差的一阶偏差约束

①曲线平滑

$$w_s \cdot \sum_{i=0}^{n-1} s_i^2 + w_{ds} \cdot \sum_{i=0}^{n-1} (s_i')^2 + w_{dds} \cdot \sum_{i=0}^{n-2} (\frac{s_{i+1}'' - s_i''}{\Delta s^2})^2 \tag{43}$$

②与参考线偏差

$$w_{xref} \cdot \sum_{i=0}^{n-1} (s_i - s_{ref})^2 + w_{dxref} \cdot \sum_{i=0}^{n-1} (s_i' - s_{ref}')^2 \tag{44}$$

③终点

$$w_{end_l} \cdot (s_{n-1} - s_{endref})^2 + w_{end_{dl}} \cdot (s_{n-1}' - s_{endref}')^2 + w_{end_{ddl}} \cdot (s_{n-1}'' - s_{endref}'')^2 \tag{45}$$

3.约束条件

约束分为六个部分(6n*1)

$$\begin{gathered}
\text{对变量的约束}(3n)\text{：} LowerBounds < s, s', s'', s''' < upperBounds \\
\text{对}Jerk\text{的约束}(n-1)\text{：} LowerBounds < s''' < upperBounds\text{对起点的约束}(3)\text{：} ego_1 \le s_1, s_1', s_1'' \le ego_1 \\
\text{连续性约束}(2n-2)\text{：} s_{i+1} - s_i - \Delta s \cdot s_i' - \frac{1}{3}\Delta s^2 \cdot s_i'' - \frac{1}{6}\Delta s^2 \cdot s_{i+1}'' = 0 \\
s_{i+1}' - s_i' - \frac{1}{2}\Delta s * s_i'' - \frac{1}{2}\Delta s * s_{i+1}'' = 0
\end{gathered} \tag{46}$$

分享

工具使用

- CMake
- Google test
- matplotlib-cpp
- valgrind

移植 Autoware/Apollo math下的方法

1. 算法原理和实现逻辑
2. 参考Google test的测试逻辑，弄清输入输出量
3. 改造函数接口，去掉不必要的方法
4. 修改对应CMakeLists.txt,生成动态链接库或者直接头文件引用

example

移植 autoware 下的线性插值方法

文件位于`/AutowareArchitectureProposal.iv-use-autoware-auto-msgsl/common/math/interpolation`

1. 原理很简单，初中的知识
2. 修改CMakeList.txt

```
//修改之后的CMakeLists.txt

//autoware基于ros2来进行通讯，用到了重新基于CMake工具构造的ament编译期，跟catkin_make差别不大
cmake_minimum_required(VERSION 3.5)
project(interpolation)

### Compile options
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 17)
endif()
if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic -Werror)
endif()

find_package(ament_cmake_auto REQUIRED)
ament_auto_find_build_dependencies()

ament_auto_add_library(interpolation SHARED
  src/linear_interpolation.cpp
  src/spline_interpolation.cpp
)

# Test
if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()

  find_package(ament_cmake_gtest REQUIRED)

  file(GLOB_RECURSE test_files test/**/*.cpp)

  ament_add_gtest(test_interpolation ${test_files})

  target_link_libraries(test_interpolation
    interpolation
  )
endif()

ament_auto_package()
```

```
//修改之后的CMakeLists.txt
cmake_minimum_required(VERSION 3.8)
project(interpolation VERSION 0.1.0)
//指定源文件 并取别名
aux_source_directory(${PROJECT_SOURCE_DIR}/src DIR_LIB_SRCS)

include_directories(
  ${PROJECT_SOURCE_DIR}/include
)
# 生成链接库,默认生成的是静态库
add_library(${PROJECT_NAME} SHARED ${DIR_LIB_SRCS})
# add_library(MathFunctions_o STATIC ${DIR_LIB_SRCS})

# 其他的库都可以链接，但是自己不链接 指定了需求之后，可以安全地从顶层CMakeLists.txt中移除对变量EXTRA_INCLUDES的使用
target_include_directories(${PROJECT_NAME}
    INTERFACE ${CMAKE_CURRENT_SOURCE_DIR}
)
```

如何调用有两个思路

1.直接引用头文件

2.生成链接库(用到上述的CMakeLists.txt)

　相当于把上述文件生成一个子CMakelists,下文83行

```
 1  cmake_minimum_required(VERSION 3.8)
 2  project(OsqpEigen-Example)
 3  
 4  # 设置c++版本
 5  # set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall")
 6  set(CMAKE_CXX_STANDARD 20)
 7  
 8  # 设置编译版本 bebug / release
 9  SET(CMAKE_BUILD_TYPE Debug)
10  
11  # 设置静态库文件目录
12  set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/lib)
13  # # 动态库文件目录
14  set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/lib)
15  # 可执行文件目录
16  set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/bin)
17  
18  include_directories(
19    ${PROJECT_SOURCE_DIR}/include
20    ${PROJECT_SOURCE_DIR}/common/math
21    ${PROJECT_SOURCE_DIR}/common/util
22    ${PROJECT_SOURCE_DIR}/common/math/interpolation/include
23    ${PROJECT_SOURCE_DIR}/common/math/osqp_interface/include
24    ${PROJECT_SOURCE_DIR}/common/math/piecewise_jerk/include
25  
26  )
27  
28  # find_package(OsqpEigen)
29  find_package(Eigen3)
30  
31  # Find OSQP library and headers
32  find_package(osqp REQUIRED)
33  # get_target_property(OSQP_INCLUDE_DIR osqp::osqp INTERFACE_INCLUDE_DIRECTORIES)
34  
35  include_directories(SYSTEM ${EIGEN3_INCLUDE_DIR})
36  # include_directories(SYSTEM ${OSQP_INCLUDE_DIR})
37  # Link the OSQP shared library
38  # target_link_libraries(yourTarget PRIVATE osqp::osqp)
39  
40  # or...
41  
42  # # Link the OSQP static library
43  # target_link_libraries(yourTarget PRIVATE osqp::osqpstatic)
44  #MPCExample
45  # add_executable(MPCExample src/MPCExample.cpp)
46  # target_link_libraries(MPCExample OsqpEigen::OsqpEigen)
47  
48  #matplotlib-cpp
49  include(GNUInstallDirs)
50  set(PACKAGE_NAME matplotlib_cpp)
51  set(INSTALL_CONFIGDIR ${CMAKE_INSTALL_LIBDIR}/${PACKAGE_NAME}/cmake)
52  
53  
54  # Library target
55  add_library(matplotlib_cpp INTERFACE)
56  target_include_directories(matplotlib_cpp
57    INTERFACE
58      $<BUILD_INTERFACE:${PROJECT_SOURCE_DIR}/src>
59      $<INSTALL_INTERFACE:include>
60  )
61  target_compile_features(matplotlib_cpp INTERFACE
62    cxx_std_20
63  )
64  # TODO: Use `Development.Embed` component when requiring cmake >= 3.18
65  find_package(Python3 COMPONENTS Interpreter Development REQUIRED)
66  target_link_libraries(matplotlib_cpp INTERFACE
67    Python3::Python
68    Python3::Module
69  )
70  find_package(Python3 COMPONENTS NumPy)
71  if(Python3_NumPy_FOUND)
72    target_link_libraries(matplotlib_cpp INTERFACE
73      Python3::NumPy
74    )
75  else()
76    target_compile_definitions(matplotlib_cpp INTERFACE WITHOUT_NUMPY)
77  endif()
78  install(
79    TARGETS matplotlib_cpp
```

```
  EXPORT install_targets
)

#添加子CMakeLists.txt
add_subdirectory(${PROJECT_SOURCE_DIR}/common/math/interpolation)
list(APPEND EXTRA_LIBS interpolation)

# Examples
add_executable(lineInterpoltion src/lineInterpoltion.cpp)
target_link_libraries(lineInterpoltion PRIVATE matplotlib_cpp PUBLIC ${EXTRA_LIBS})
# set_target_properties(midpoints PROPERTIES RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/bin")

add_executable(splineInterpoltion src/splineInterpoltion.cpp)
target_link_libraries(splineInterpoltion PUBLIC matplotlib_cpp PUBLIC ${EXTRA_LIBS})
# set_target_properties(animation PROPERTIES RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/bin")



add_subdirectory(${PROJECT_SOURCE_DIR}/common/math/osqp_interface)
list(APPEND EXTRA_LIBS osqp_csc)

add_executable(csc_conv src/csc_matrix_osqp.cpp)
# 多个库文件链接顺序问题
# 在链接命令中给出所依赖的库时，需要注意库之间的依赖顺序，依赖其他库的库一定要放到被依赖库的前面，这样才能真正避免undefined reference的错误，完成编译链接
target_link_libraries(csc_conv PUBLIC matplotlib_cpp PUBLIC ${EXTRA_LIBS} PUBLIC osqp::osqp )

# add_subdirectory(${PROJECT_SOURCE_DIR}/common/math/piecewise_jerk)
# list(APPEND EXTRA_LIBS path_optimize)

# add_executable(path_jerk src/piecewise_jerk_path_optimizer.cc)
# 多个库文件链接顺序问题
# 在链接命令中给出所依赖的库时，需要注意库之间的依赖顺序，依赖其他库的库一定要放到被依赖库的前面，这样才能真正避免undefined reference的错误，完成编译链接
# target_link_libraries(path_jerk PUBLIC matplotlib_cpp PUBLIC ${EXTRA_LIBS} PUBLIC osqp::osqp )


# add_executable(pathoptim src/PathOptimize.cpp)
add_executable(pathoptim src/pathDemo.cpp src/PathOptimize.cpp)
# add_executable(pathoptim src/osqp_test.cpp)
# 多个库文件链接顺序问题
# 在链接命令中给出所依赖的库时，需要注意库之间的依赖顺序，依赖其他库的库一定要放到被依赖库的前面，这样才能真正避免undefined reference的错误，完成编译链接
target_link_libraries(pathoptim PUBLIC ${EXTRA_LIBS} PUBLIC matplotlib_cpp  PRIVATE osqp::osqp )
```

注意事项

1. 如果想用移植apollo的osqp库求解，安装版本一定要是osqp_0.4.1，新版本不支持p矩阵不为上三角的运算

```
osqp库
    头文件使用格式 #include "osqp/osqp.h"
    1.下载源码
    git clone -b 1.4.1 https://github.com/osqp/osqp
    2.解压之后
        cd osqp
        mkdir build
        cd build
    3.创建Makefile
        cmake -G "Unix Makefiles" ..
    4.编译并安装
        sudo cmake --build . --target install

    注意:
        1.如果不加--recursive 会出现“/home/next/osqp/lin_sys/direct/qdldl/qdldl_sources” 不存在的错误,
    所以下载的时候要加上, 如果已经下载好了可以用git submodule update --init --recursive修复
        2.  0.4.1版本的osqp求p矩阵时不要求上三角
```

2. ipopt安装

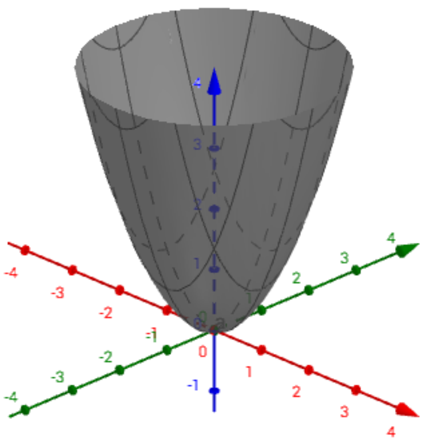
   参考博客, 不要sudo

思考

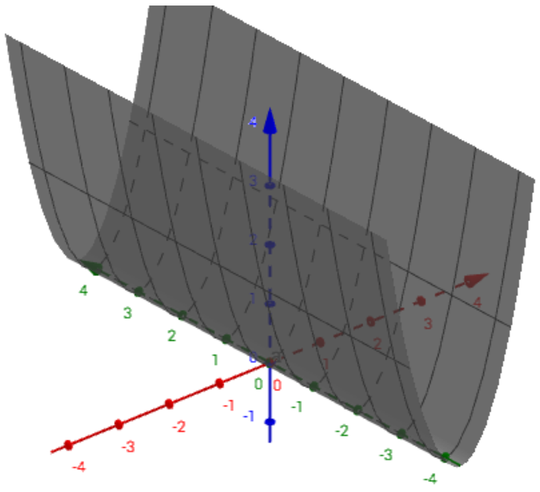
1 为什么需要正定矩阵?

- 如果P是半正定矩阵，那么f(x)是一个凸函数。相应的二次规划为凸二次规划问题；此时若约束条件定义的可行域不为空，且目标函数在此可行域有下界，则该问题有全局最小值。
- 如果P是正定矩阵，则该问题有唯一的全局最小值。
- 若P为非正定矩阵，则目标函数是有多个平稳点]和局部极小点的NP难问题。
- 如果P=0，二次规划问题就变成线性规划问题。

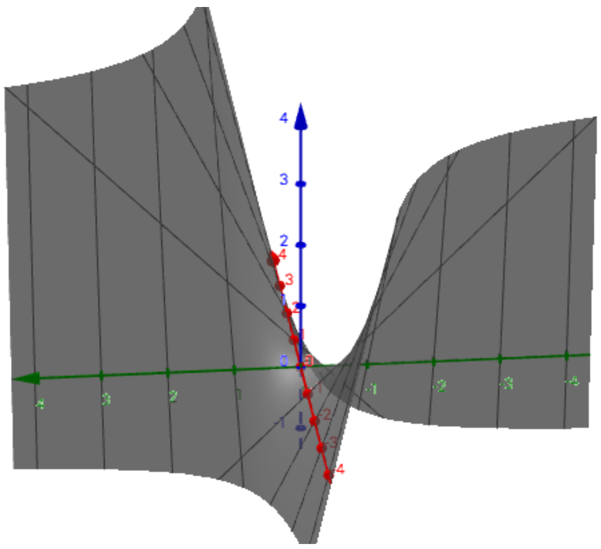
正定是对二次函数有效的一个定义，对方程无效。



$$f(x)>0,x!=0,\text{则}x\text{为正定二次型，}P\text{为正定矩阵} \tag{47}$$



$$f(x)>=0,x!=0,\text{则}f\text{为半正定二次型，}P\text{为半正定矩阵} \tag{48}$$



$$\text{不定} \tag{49}$$

2.压缩矩阵有哪几种方法?

参考英伟达的介绍