第五章作业思路讲解

主讲人 汪世玉

纲要

- Minimum snap问题
  - 时间分配
  - 构建优化模型
- QP求解
  - 求相关矩阵
  - 求解器使用
- Closed-form
  - 求M矩阵
  - 求Ct矩阵

Minimum snap

深蓝学院
shenlanxueyuan.com

- Constrained quadratic programming (QP) formulation:

$$\min \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}$$

$$\text{s.t.} \quad \mathbf{A}_{\mathrm{eq}} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} = \mathbf{d}_{eq}$$

It's a typical **convex optimization** program.

仅含等式约束的二次规划问题
1.QP求解：求Q阵、A阵、d阵送入求解器求解；
2.closed-form：化去等式约束；

Minimum snap：时间分配

$$f(t) = \begin{cases} f_1(t) \doteq \sum_{i=0}^{N} p_{1,i} t^i & T_0 \le t \le T_1 \\ f_2(t) \doteq \sum_{i=0}^{N} p_{2,i} t^i & T_1 \le t \le T_2 \\ \vdots & \vdots \\ f_M(t) \doteq \sum_{i=0}^{N} p_{M,i} t^i & T_{M-1} \le t \le T_M \end{cases}$$

$$f_k(t) = \sum_{i=0}^{N} p_{k,i} t^i, \quad 0 \le t \le \Delta T_k$$

匀速分配，梯形分配，手动给定然后归一化到0~1……etc

Minimum snap：时间分配

```
double accInv = 1.0 / _Acc;
double velInv = 1.0 / _Vel;

// 加速时间与距离，减速需要耗费同样资源
double accTime = _Vel * accInv;          // v = v0 + at, v0 = 0
double accDist = 0.5 * _Vel * accTime;  // d = 0.5 * (v0 + v) * t, v0 = 0
double accTime2 = 2.0 * accTime;
double accDist2 = 2.0 * accDist;     // 包含加速和减速的总距离

for(int i = 0; i < Path.rows() - 1; i++)
{
  double t = 0.0;
  double dist = (Path.row(i + 1) - Path.row(i)).norm();

  if(dist <= accDist2)
    t = 2.0 * std::sqrt(dist * accInv);      // 0.5 * d = v0 * t + 0.5 * a * t * t, v0 = 0
  else
    t = accTime2 + (dist - accDist2) * velInv;

  time(i) = t;
}
```

梯形分配

Minimum snap：优化模型

深蓝学院 shenlanxueyuan.com

1.构建目标函数

- Cost function for one polynomial segment:

$$f(t)=\sum_i p_i t^i$$
$$\Rightarrow f^{(4)}(t)=\sum_{i\geq 4} i(i-1)(i-2)(i-3)t^{i-4}p_i$$
$$\Rightarrow \left(f^{(4)}(t)\right)^2=\sum_{i\geq 4,l\geq 4} i(i-1)(i-2)(i-3)l(l-1)(l-2)(l-3)t^{i+l-8}p_i p_l$$
$$\Rightarrow J(T)=\int_{T_{j-1}}^{T_j}\left(f^4(t)\right)^2 dt=\sum_{i\geq 4,l\geq 4}\frac{i(i-1)(i-2)(i-3)j(l-1)(l-2)(l-3)}{i+l-7}\left(T_j^{i+l-7}-T_{j-1}^{i+l-7}\right)p_i p_l$$
$$\Rightarrow J(T)=\int_{T_{j-1}}^{T_j}\left(f^4(t)\right)^2 dt$$
$$=\begin{bmatrix}\vdots\\ p_i\\ \vdots\end{bmatrix}^T\begin{bmatrix} & \vdots & \\ \cdots & \frac{i(i-1)(i-2)(i-3)l(l-1)(l-2)(l-3)}{i+l-7}T^{i+l-7} & \cdots \\ & \vdots & \end{bmatrix}\begin{bmatrix}\vdots\\ p_l\\ \vdots\end{bmatrix}$$
$$\Rightarrow \boxed{J_j(T)=\mathbf{p}_j^T\mathbf{Q}_j\mathbf{p}_j}$$ Minimize this!

2.构建等式约束

- Derivative constraint for one polynomial segment
  - Also models waypoint constraint (0^{th} order derivative)

$$f_j^{(k)}(T_j)=x_j^{(k)}$$
$$\Rightarrow \sum_{i\geq k}\frac{i!}{(i-k)!}T_j^{i-k}p_{j,i}=x_{T,j}^{(k)}$$
$$\Rightarrow \begin{bmatrix}\cdots & \frac{i!}{(i-k)!}T_j^{i-k} & \cdots\end{bmatrix}\begin{bmatrix}\vdots\\ p_{j,i}\\ \vdots\end{bmatrix}=x_{T,j}^{(k)}$$
$$\Rightarrow \begin{bmatrix}\cdots & \frac{i!}{(i-k)!}T_{j-1}^{i-k} & \cdots\\ \cdots & \frac{i!}{(i-k)!}T_j^{i-k} & \cdots\end{bmatrix}\begin{bmatrix}\vdots\\ p_{j,i}\\ \vdots\end{bmatrix}=\begin{bmatrix}x_{0,j}^{(k)}\\ x_{T,j}^{(k)}\end{bmatrix}$$
$$\Rightarrow \mathbf{A}_j\mathbf{p}_j=\mathbf{d}_j$$

$$x(t)=p_5t^5+p_4t^4+p_3t^3+p_2t^2+p_1t+p_0$$

$$x(0)=\cdots, x(T)=\cdots$$
$$\dot{x}(0)=\cdots, \dot{x}(T)=\cdots$$
$$\ddot{x}(0)=\cdots, \ddot{x}(T)=\cdots$$
$$\vdots$$

$$p_0=\cdots,$$
$$p_5T^5+p_4T^4+p_3T^3+p_2T^2+p_1T+p_0=\cdots$$

$$[T^5,T^4,T^3,T^2,T,1]\begin{bmatrix}p_5\\ p_4\\ p_3\\ p_2\\ p_1\\ p_0\end{bmatrix}=\cdots$$

34

- Continuity constraint between two segments:
  - Ensures continuity between trajectory segments when no specific derivatives are given

$$f_j^{(k)}(T_j)=f_{j+1}^{(k)}(T_j)$$
$$\Rightarrow \sum_{i\geq k}\frac{i!}{(i-k)!}T_j^{i-k}p_{j,i}-\sum_{l\geq k}\frac{l!}{(l-k)!}T_j^{l-k}p_{j+1,l}=0$$
$$\Rightarrow \begin{bmatrix}\cdots & \frac{i!}{(i-k)!}T_j^{i-k} & \cdots & -\frac{l!}{(l-k)!}T_j^{l-k} & \cdots\end{bmatrix}\begin{bmatrix}\vdots\\ p_{j,i}\\ \vdots\\ p_{j+1,l}\\ \vdots\end{bmatrix}=0$$
$$\Rightarrow \begin{bmatrix}\mathbf{A}_j & -\mathbf{A}_{j+1}\end{bmatrix}\begin{bmatrix}\mathbf{p}_j\\ \mathbf{p}_{j+1}\end{bmatrix}=0$$

QP求解： 求Q矩阵

```
for i = 4:n_order
  for l = 4:n_order
    den = i + l - 7;
    Q_k(i+1,l+1) = i*(i-1)*(i-2)*(i-3)*l*(l-1)*(l-2)*(l-3)/den*(t_k^den);
  end
end
```

$$f(t) = [p_0, p_1, ..., p_7] \cdot [1, t, t^2, ...t^7] = \mathbf{p} \cdot [1, t, t^2, ...t^7]$$
$$f'(t) = \mathbf{p} \cdot [0, 1, 2t, 3t^2, 4t^3, ...7t^6]$$
$$f''(t) = \mathbf{p} \cdot [0, 0, 2, 6t, 12t^2, ...42t^5]$$
$$f'''(t) = \mathbf{p} \cdot [0, 0, 0, 6, 24t, ...210t^4]$$

QP求解：求Aeq矩阵

深蓝学院 shenlanxueyuan.com

- Derivative constraint for one polynomial segment
  - Also models waypoint constraint (0^{th} order derivative)

$$
\begin{aligned}
& f_j^{(k)}(T_j) = x_j^{(k)} \\
\Rightarrow & \sum_{i \ge k} \frac{i!}{(i-k)!} T_j^{i-k} p_{j,i} = x_{T,j}^{(k)} \\
\Rightarrow & \begin{bmatrix} \cdots & \frac{i!}{(i-k)!} T_j^{i-k} & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ p_{j,i} \\ \vdots \end{bmatrix} = x_{T,j}^{(k)} \\
\Rightarrow & \begin{bmatrix} \cdots & \frac{i!}{(i-k)!} T_{j-1}^{i-k} & \cdots \\ \cdots & \frac{i!}{(i-k)!} T_j^{i-k} & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ p_{j,i} \\ \vdots \end{bmatrix} = \begin{bmatrix} x_{0,j}^{(k)} \\ x_{T,j}^{(k)} \end{bmatrix} \\
\Rightarrow & \mathbf{A}_j \mathbf{p}_j = \mathbf{d}_j
\end{aligned}
$$

$$x(t) = p_5 t^5 + p_4 t^4 + p_3 t^3 + p_2 t^2 + p_1 t + p_0$$

$$
\begin{aligned}
x(0) &= \cdots, x(T) = \cdots \\
\dot{x}(0) &= \cdots, \dot{x}(T) = \cdots \\
\ddot{x}(0) &= \cdots, \ddot{x}(T) = \cdots \\
&\vdots
\end{aligned}
$$

$$
\begin{aligned}
& p_0 = \cdots, \\
& p_5 T^5 + p_4 T^4 + p_3 T^3 + p_2 T^2 + p_1 T + p_0 = \cdots
\end{aligned}
$$

$$[T^5, T^4, T^3, T^2, T, 1] \begin{bmatrix} p_5 \\ p_4 \\ p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix} = \cdots$$

```matlab
%#####################################################
% p,v,a,j constraint in start,
Aeq_start = zeros(4, n_all_poly);
T = 0;
for k = 0 : 3 % p,v,a,j
    for i = k : n_order % i >= k
        Aeq_start(k+1,i+1) = factorial(i)/factorial(i-k)*(T^(i-k));
    end
end
beq_start = start_cond';

%#####################################################
% p,v,a,j constraint in end
Aeq_end = zeros(4, n_all_poly);
T = ts(end);
idx = (n_seg-1)*(n_order+1);
for k = 0 : 3 % p,v,a,j
    for i = k : n_order % i >= k
        Aeq_end(k+1,idx+i+1) = factorial(i)/factorial(i-k)*(T^(i-k));
    end
end
beq_end = end_cond';

%#####################################################
% position constrain in all middle waypoints
Aeq_wp = zeros(n_seg-1, n_all_poly);
for n = 0 : n_seg-1-1
    T = ts(n+1);
    idx = (n)*(n_order+1);
    for i = 0 : n_order
        Aeq_wp(n+1, idx+i+1) = T^i;
    end
end
beq_wp = waypoints(2:end-1);
```

QP求解：求Aeq矩阵

深蓝学院
shenlanxueyuan.com

- Continuity constraint between two segments:
  - Ensures continuity between trajectory segments when no specific derivatives are given

$$f_j^{(k)}(T_j) = f_{j+1}^{(k)}(T_j)$$

$$\Rightarrow \sum_{i \geq k} \frac{i!}{(i-k)!} T_j^{i-k} p_{j,i} - \sum_{l \geq k} \frac{l!}{(l-k)!} T_j^{l-k} p_{j+1,l} = 0$$

$$\Rightarrow \begin{bmatrix} \cdots & \frac{i!}{(i-k)!} T_j^{i-k} & \cdots & -\frac{l!}{(l-k)!} T_j^{l-k} & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ p_{j,i} \\ \vdots \\ p_{j+1,l} \\ \vdots \end{bmatrix} = 0$$

$$\Rightarrow \begin{bmatrix} \mathbf{A}_j & -\mathbf{A}_{j+1} \end{bmatrix} \begin{bmatrix} \mathbf{p}_j \\ \mathbf{p}_{j+1} \end{bmatrix} = 0$$

```
%#################################################
% position continuity constrain between each 2 segments
Aeq_con_p = zeros(n_seg-1, n_all_poly);
beq_con_p = zeros(n_seg-1, 1);
% STEP 2.4: write expression of Aeq_con_p and beq_con_p
k = 0; % k = 0,1,2,3
for n = 0 : n_seg-1-1
    T = ts(n+1);
    idx = (n)*(n_order+1);
    for i = k : n_order
        Aeq_con_p(n+1, idx+i+1) = factorial(i)/factorial(i-k)*(T^(i-k));
    end
    T = 0;
    idx = (n+1)*(n_order+1);
    for i = k : n_order
        Aeq_con_p(n+1, idx+i+1) = -factorial(i)/factorial(i-k)*(T^(i-k));
    end
end

% combine all components to form Aeq and beq
Aeq_con = [Aeq_con_p; Aeq_con_v; Aeq_con_a; Aeq_con_j];
beq_con = [beq_con_p; beq_con_v; beq_con_a; beq_con_j];
Aeq = [Aeq_start; Aeq_end; Aeq_wp; Aeq_con];
beq = [beq_start; beq_end; beq_wp; beq_con];
```

QP求解：matlab quadprog

深蓝学院
shenlanxueyuan.com

quadprog

Solve quadratic programming problems

Equation

Finds a minimum for a problem specified by

$$\min_x \frac{1}{2}x^T H x + f^T x \text{ such that } \begin{cases} A \cdot x \le b, \\ Aeq \cdot x = beq, \\ lb \le x \le ub. \end{cases}$$

H, *A*, and *Aeq* are matrices, and *f*, *b*, *beq*, *lb*, *ub*, and *x* are vectors.

Syntax

```
x = quadprog(H,f,A,b)
x = quadprog(H,f,A,b,Aeq,beq)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
x = quadprog(problem)
[x,fval] = quadprog(...)
[x,fval,exitflag] = quadprog(...)
[x,fval,exitflag,output] = quadprog(...)
[x,fval,exitflag,output,lambda] = quadprog(...)
```

QP求解：OSQP求解器使用

osqp-Eigen

```
>>> git clone https://github.com/robotology/osqp-eigen.git
>>> cd osqp-eigen
>>> mkdir build && cd build && cmake ..
>>> sudo make install
>>> echo "export OsqpEigen_DIR=/path/"
```

osqp

```
>>> git clone --recursive https://github.com/oxfordcontrol/osqp.git
>>> cd osqp && mkdir build && cd build && cmake ..
>>> make
>>> sudo make install
```

CmakeLists.txt

```
cmake_minimum_required(VERSION 3.0)
find_package(OsqpEigen REQUIRED)
add_executable(xxx_node src/main.cpp)
target_link_libraries(xxx_node
                      OsqpEigen::OsqpEigen
                      osqp::osqp
                      Eigen3::Eigen)
```

Demo :

```
OsqpEigen::Solver solver;

solver.settings()->setWarmStart(true);
solver.data()->setNumberOfVariables( );                  //变量个数
solver.data()->setNumberOfConstraints( );                //约束条件个数
if( ! solver.data()->setHessianMatrix(H)) return 1;      //信息矩阵 H
if( ! solver.data()->setGradient(g)) return 1;           //梯度 f
if( ! solver.data()->setLinearConstraintsMatrix(linearMartix));
//约束矩阵 lower < A* x < up
if( ! solver.data()->setLowerBound(lowerBound)) return 1;   //下边界
if( ! solver.data()->setUpperBound(upperBound)) return 1;   //上边界

if( !solver.initSolver() ) return 1;
if( !solver.solve() ) return 1;
Eigen::VectorXd QPSolution = solver.getSolution();
```

https://github.com/oxfordcontrol/osqp_benchmarks

https://osqp.org/docs/interfaces/index.html

QP求解：OOQP求解器使用

2.3 Calling from a C++ Program

When calling OOQP from a C++ code, the user must create several objects and call several methods in sequence. The process is more complicated than simply calling a C function, but also more flexible. By varying the classes of the objects created, one can generate customized solvers for QPs of various types. In this section, we focus on the default solver for the formulation (2). The full sequence of calls for this case is shown in Figure 3. In the remainder of this section, we explain each call in this sequence in turn.

```
QpGenSparseMa27 * qp
    = new QpGenSparseMa27( nx, my, mz, nnzQ, nnzA, nnzC );

QpGenData      * prob
    = (QpGenData * ) qp->makeData( /* parameters here */);
QpGenVars      * vars
    = (QpGenVars *) qp->makeVariables( prob );
QpGenResiduals * resid
    = (QpGenResiduals *) qp->makeResiduals( prob );

GondzioSolver  * s     = new GondzioSolver( qp, prob );

s->monitorSelf();
int status = s->solve(prob,vars, resid);
```

Figure 3: The basic sequence for calling OOQP

http://pages.cs.wisc.edu/~swright/ooqp/ooqp-userguide.pdf

https://qiaoxu123.github.io/post/ubuntu-ooqp-useguide/

Closed-form

深蓝学院
shenlanxueyuan.com

- We have $\boldsymbol{M}_j \mathbf{p}_j = \mathbf{d}_j$, where $\boldsymbol{M}_j$ is a mapping matrix that maps polynomial coefficients to derivatives

$$J = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} \qquad J = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix}^T \begin{bmatrix} \boldsymbol{M}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{M}_M \end{bmatrix}^{-T} \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M \end{bmatrix} \begin{bmatrix} \boldsymbol{M}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{M}_M \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix}$$

- Use a selection matrix $\mathbf{C}$ to separate free ($\mathbf{d}_P$) and constrained ($\mathbf{d}_F$) variables
  - Free variables : derivatives unspecified, only enforced by continuity constraints

$$\mathbf{C}^T \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix} \quad \Longrightarrow \quad J = \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}^T \underbrace{\mathbf{C}\boldsymbol{M}^{-T}\mathbf{Q}\boldsymbol{M}^{-1}\mathbf{C}^T}_{\mathbf{R}} \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix} = \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}^T \begin{bmatrix} \mathbf{R}_{FF} & \mathbf{R}_{FP} \\ \mathbf{R}_{PF} & \mathbf{R}_{PP} \end{bmatrix} \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}$$

- Turned into an unconstrained quadratic programming that can be solved in closed form:

$$J = \mathbf{d}_F^T \mathbf{R}_{FF} \mathbf{d}_F + \mathbf{d}_F^T \mathbf{R}_{FP} \mathbf{d}_P + \mathbf{d}_P^T \mathbf{R}_{PF} \mathbf{d}_F + \mathbf{d}_P^T \mathbf{R}_{PP} \mathbf{d}_P$$

$$\mathbf{d}_P^* = -\mathbf{R}_{PP}^{-1} \mathbf{R}_{FP}^T \mathbf{d}_F$$

Closed-form：求M矩阵

$$J=\begin{bmatrix}\mathbf{p}_1\\ \vdots\\ \mathbf{p}_M\end{bmatrix}^T\begin{bmatrix}\mathbf{Q}_1 & 0 & 0\\ 0 & \ddots & 0\\ 0 & 0 & \mathbf{Q}_M\end{bmatrix}\begin{bmatrix}\mathbf{p}_1\\ \vdots\\ \mathbf{p}_M\end{bmatrix} \quad + \quad \boldsymbol{M}_j\mathbf{p}_j=\mathbf{d}_j$$

$$J=\begin{bmatrix}\mathbf{d}_1\\ \vdots\\ \mathbf{d}_M\end{bmatrix}^T\begin{bmatrix}\boldsymbol{M}_1 & 0 & 0\\ 0 & \ddots & 0\\ 0 & 0 & \boldsymbol{M}_M\end{bmatrix}^{-T}\begin{bmatrix}\mathbf{Q}_1 & 0 & 0\\ 0 & \ddots & 0\\ 0 & 0 & \mathbf{Q}_M\end{bmatrix}\begin{bmatrix}\boldsymbol{M}_1 & 0 & 0\\ 0 & \ddots & 0\\ 0 & 0 & \boldsymbol{M}_M\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{d}_1\\ \vdots\\ \mathbf{d}_M\end{bmatrix}$$

$$x(t)=p_5t^5+p_4t^4+p_3t^3+p_2t^2+p_1t+p_0$$
$$x'(t)=5p_5t^4+4p_4t^3+3p_3t^2+2p_2t+p_1$$
$$x''(t)=20p_5t^3+12p_4t^2+6p_3t+2p_2$$

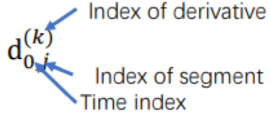
$$M=\begin{bmatrix}0&0&0&0&0&1\\0&0&0&0&1&0\\0&0&0&2&0&0\\T^5&T^4&T^3&T^2&T&1\\5T^4&4T^3&3T^2&2T&1&0\\20T^3&12T^2&6T&2&0&0\end{bmatrix}$$

$$\boldsymbol{M}_j\mathbf{p}_j=\mathbf{d}_j$$

$$f_j^{(k)}(T_j)=x_j^{(k)}$$
$$\Rightarrow\sum_{i\geq k}\frac{i!}{(i-k)!}T_j^{i-k}p_{j,i}=x_{T,j}^{(k)}$$

```
M = [];
for n = 1:n_seg
    M_k = [];
    %#####################################################
    % STEP 1.1: calculate M_k of the k-th segment
    T = 0;
    for k = 0 : 3 % p,v,a,j at t0
        for i = k : n_order
            M_k(k+1,i+1) = factorial(i)/factorial(i-k)*(T^(i-k));
        end
    end
    T = ts(n);
    for k = 0 : 3 % p,v,a,j at T
        for i = k : n_order
            M_k(4+k+1,i+1) = factorial(i)/factorial(i-k)*(T^(i-k));
        end
    end
    M = blkdiag(M, M_k);
end
```

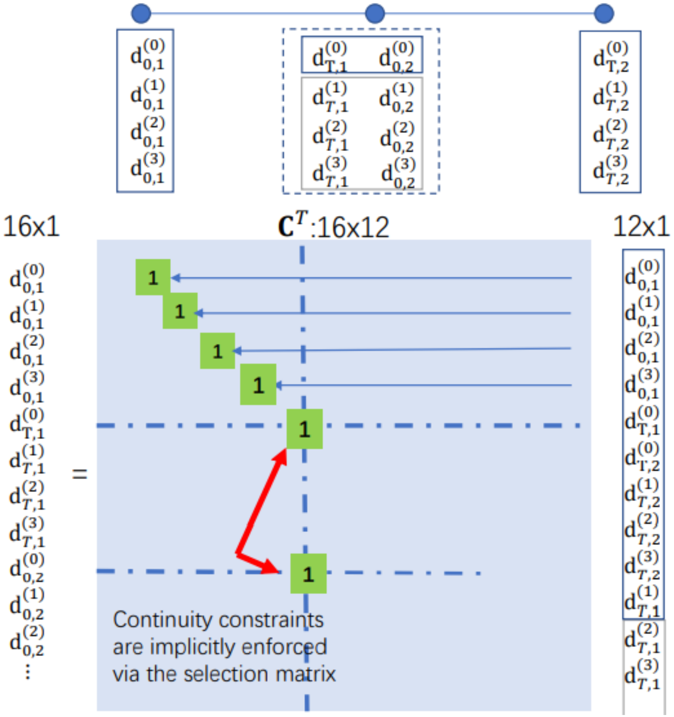
Closed-form：求Ct矩阵



Fixed derivatives: fixed start, goal state, and intermediate positions

Free derivatives: all derivatives at intermediate connections.

$$\begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix} = \mathbf{C}^T \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}$$




```
function Ct = getCt(n_seg, n_order)
    %d1....dM是从各段路径首端(T=0)和末尾(T=ts(i))时刻的pvaj值
    %dF 是确定的导数约束(起点状态pvaj和中间点位置p以及终点状态pvaj)
    %dP 是未定导数约束(中间状态，末端路径的末尾(T=ts(i)时的的vaj)
  %#####################################################
  % STEP 2.1: finish the expression of Ct
  %
  d_size = 8*n_seg; % size(d) = 8*n_seg, 1
  dF_size = 7+n_seg;
  dP_size = 3*(n_seg-1);

  n_rows = d_size % Ct行数
  n_cols= dF_size + dP_size % Ct列数
  Ct = zeros(n_rows, n_cols); %初始化Ct

  i = 1; %指向dF中的元素
  j = dF_size+1; %指向dP中的元素

  for r = 1:n_rows
    if (r<=4 ||  r>=n_rows-3) %满足这些条件意味着d是起始状态和末尾状态 需要与dF区域相对应一次
      Ct(r,i) = 1;
      i = i+1;
    elseif(mod(r,4)==1) %中间点位置，施加连续型约束需要匹配两次
      Ct(r,i) = 1;
      if(mod(r,8)==1) %意味着之前已经用过一次了
        i = i+1;
      end
    else % 意味着d需要与dP区域相对应, 施加未定的连续型约束(vaj)，需要用到两次
      Ct(r,j) = 1;
      if(mod(r,8)==0) % 需要重复一次
        j = j -2;
      else
        j = j+1;
      end

    end

  end
```

感谢各位聆听！

Thanks for Listening