第五章作业分享

主讲人 PaulX

纲要

- 第一部分：minimum snap QP求解
- 第二部分：minimum snap 闭式求解

minimum snap QP求解

深蓝学院 shenlanxueyuan.com

- 目标

构建目标函数与约束条件,带入Matlab的QP求解器

- matlab中的QP求解器

`x = quadprog(H,f,A,b,Aeq,beq)` solves the preceding problem subject to the additional restrictions `Aeq*x = beq`. `Aeq` is a matrix of doubles, and `beq` is a vector of doubles. If no inequalities exist, set `A = []` and `b = []`.

A与b是不等式约束,因此本次作业设为[] []

计算等式约束的Aeq 与beq

minimum snap QP求解

深蓝学院 shenlanxueyuan.com

●QP求解的目标函数

$$J(T)=\int_{T_{j-1}}^{T_j}\left(f^4(t)\right)^2dt=\sum_{i\geq 4,l\geq 4}\frac{i(i-1)(i-2)(i-3)j(l-1)(l-2)(l-3)}{i+l-7}(T_j^{i+l-7}-T_{j-1}^{i+l-7})p_i\,p_l$$

在作业中$T_{j-1}=0, T_j=t(k)$

```
% 确定Q矩阵的尺寸
Q_k = zeros(n_order+1);
for i = 4:n_order
    for j = 4:n_order
        Q_k(i+1, j+1) = (i*(i-1)*(i-2)*(i-3)*j*(j-1)*(j-2)*(j-3)/(i+j-7))*ts(k)^(i+j-7);
    end
end
% 对角矩阵拼接
Q = blkdiag(Q, Q_k);
```

minimum snap QP求解

- 等式约束

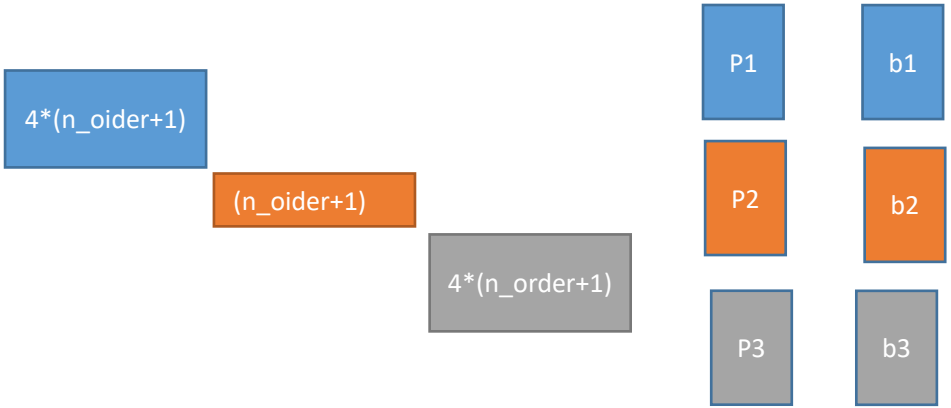
等式约束分为导数约束与连续性约束,最后构成约束的线性方程组为Aeq *x = beq;

难点:确定不同约束在Aeq矩阵中的位置,以三段的minimun snap为例子

导数约束中各段参数的位置

minimum snap QP求解

- 导数约束



- 因此导数约束的矩阵长度为 n_seg*(n_order+1)；

minimum snap QP求解

- 起点约束线性方程在矩阵中的位置

```
%起点的状态约束
for i = 1 : n_state
    Aeq_start(i,1:n_coef) = getDerivative(0, n_order, i-1);
end
beq_start = start_cond';
```

- 终点约束在矩阵中的位置

```
% 终点的状态约束
Aeq_end = zeros(4, n_all_poly);
for i = 1 : n_state
    Aeq_end(i,n_coef*(n_seg-1)+1:n_coef*n_seg) = getDerivative(ts(end), n_order, i-1);
end
beq_end = end_cond';
```

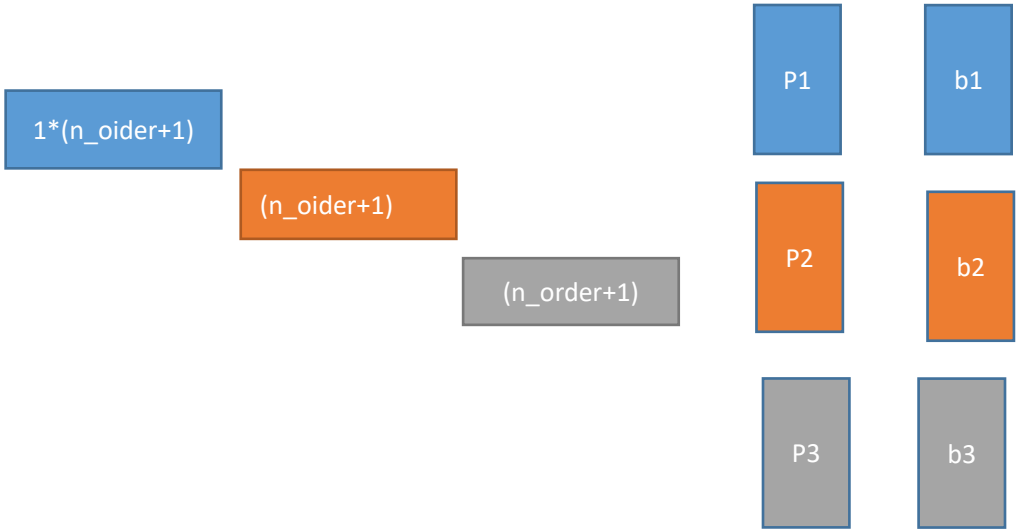
minimum snap QP求解

- 中间位置的约束

```
% 中间过程的位置约束
Aeq_wp = zeros(n_seg-1, n_all_poly);
beq_wp = zeros(n_seg-1, 1);
for i = 1:n_seg-1
    Aeq_wp(i,n_coef*i+1:n_coef*(i+1)) = getDerivative(0, n_order, 0);
    beq_wp(i) = waypoints(i+1);% 位置约束
end
```

minimum snap QP求解

- 连续约束



minimum snap QP求解

深蓝学院 shenlanxueyuan.com

中间状态连续性约束

```
% 求多项式对应行k阶导数的值

function derivative_k = getDerivative(T, n_order, k)
    derivative_k = zeros(1,n_order+1);
    for i=k+1:n_order+1
        derivative_k(i) = factorial(i-1) /factorial(i-k-1) *T^(i-k-1);
    end

end
```

```
% 连续性约束
% 01 位置连续性约束
Aeq_con_p = zeros(n_seg-1, n_all_poly);
beq_con_p = zeros(n_seg-1, 1);
Aeq_con_v = zeros(n_seg-1, n_all_poly);
beq_con_v = zeros(n_seg-1, 1);
Aeq_con_a = zeros(n_seg-1, n_all_poly);
beq_con_a = zeros(n_seg-1, 1);
Aeq_con_j = zeros(n_seg-1, n_all_poly);
beq_con_j = zeros(n_seg-1, 1);
for i=1:n_seg-1
    % 位置的连续性约束
    p1 = getDerivative(ts(i),n_order,0);
    p2 = getDerivative(0,n_order,0);
    Aeq_con_p(i,n_coef*(i-1)+1:n_coef*(i+1)) = [p1,-p2];
    % 速度的连续性约束
    v1 = getDerivative(ts(i),n_order,1);
    v2 = getDerivative(0,n_order,1);
    Aeq_con_v(i,n_coef*(i-1)+1:n_coef*(i+1)) = [v1,-v2];
    % 加速度的连续性约束
    a1 = getDerivative(ts(i),n_order,2);
    a2 = getDerivative(0,n_order,2);
    Aeq_con_a(i,n_coef*(i-1)+1:n_coef*(i+1)) = [a1,-a2];
    % 加加速度的连续性约束
    j1 = getDerivative(ts(i),n_order,3);
    j2 = getDerivative(0,n_order,3);
    Aeq_con_j(i,n_coef*(i-1)+1:n_coef*(i+1)) = [j1,-j2];

end
```

minimum snap闭式求解

- 思想

将有约束的QP最优化问题转换为无约束问题

- 方法

将等式约束条件添加到优化的目标函数上

minimum snap闭式求解

- 构建M矩阵是将导数约束和联系性施加到目标函数中

```
for k = 1:n_seg
    M_k = zeros(n_order +1);

    % 4个状态量
    d_state_num = 4;
    for i =1:d_state_num
        % 每段起点的导数映射
        M_k(i,:)=getDerivative(0, n_order, i-1);
        % 每段终点的导数映射
        M_k(d_state_num+i,:)=getDerivative(ts(k), n_order, i-1);
    end
    % 对角矩阵叠加
    M = blkdiag(M, M_k);
```

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

minimum snap闭式求解

深蓝学院 shenlanxueyuan.com

- 构建C矩阵是将已知的和需要求解的参数分离
  - 已知状态量(df)数目4+(n − 1) + 4 = n+7 (n为段的数目)

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\bullet\begin{bmatrix}p_0\\v_0\\a_0\\j_0\end{bmatrix}=\begin{bmatrix}p_0\\v_0\\a_0\\j_0\end{bmatrix}$$

df中第1块矩阵转换

$$\begin{bmatrix}1\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}\bullet[P_i]=\begin{bmatrix}P_i\\0\\0\\0\\P_i\\0\\0\\0\end{bmatrix}$$

df中第i块矩阵转换

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\bullet\begin{bmatrix}p_n\\v_n\\a_n\\j_n\end{bmatrix}=\begin{bmatrix}p_n\\v_n\\a_n\\j_n\end{bmatrix}$$

df中第n块矩阵转换

minimum snap闭式求解

- 构建C矩阵是将已知的和需要求解的参数分离
  - 已知状态量(df)数目4+(n - 1) + 4 = n+7 (n为段的数目)

```
Cstart = eye(4);
% 将终点段的state全部取出
Cend = eye(4);

% 中间已知的约束为位置约束,将中间约束放到前面
% 接下来的状态,即是前一段终点的位置,又是下一段起点的位置
Cfk = [1;0;0;0;1;0;0;0];
Cfk_all = [];
for i = 1:(n_seg-1)
    Cfk_all = blkdiag(Cfk_all,Cfk);
end
Cf = blkdiag(Cstart,Cfk_all,Cend);
```

minimum snap闭式求解

深蓝学院 shenlanxueyuan.com

- 构建C矩阵是将已知的和需要求解的参数分离
  - 未知状态量(dp)的数目 3*(n-1)

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} v_i \\ a_i \\ j_i \end{bmatrix} = \begin{bmatrix} 0 \\ v_i \\ a_i \\ j_i \\ 0 \\ v_i \\ a_i \\ j_i \end{bmatrix}$$

dp中第i块矩阵转换

```
Cp = [];
% 取出除了位置信息以外的数据
Cpk = [[0,0,0];eye(3);[0,0,0];eye(3)];
% 中间关键点,需要求解的未知量
for i = 1:(n_seg-1)
    Cp = blkdiag(Cp, Cpk);
end
% 首尾的状态都不取, 中间的状态,位置信息不取
% 待求的未知数的个数 3*(n_seg-1)
Cp = [zeros(4,3*(n_seg-1));Cp;zeros(4,3*(n_seg-1))];
% C矩阵的转置
Ct = [Cf,Cp];
```

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening