# cilqr for control

## 1. problem formulation

### 1.1 vehicle model

车辆模型来自论文：Numerically Stable Dynamic Bicycle Model for Discrete-time Control

#### 连续时间非线性方程

$$\dot{X} = f(X,U) = \begin{bmatrix} u\cos(\phi) - v\sin(\phi) \\ u\sin(\phi) + v\cos(\phi) \\ \omega \\ a + v\omega - \frac{1}{m}F_{Y1}sin\delta \\ -u\omega + \frac{1}{m}(F_{Y2}cos\delta + F_{Y2}) \\ \frac{1}{I_z}(l_fF_{Y1}cos\delta - l_rF_{Y2}) \end{bmatrix}$$

$$X = \begin{bmatrix} x \\ y \\ \phi \\ u \\ v \\ \omega \end{bmatrix}, U = \begin{bmatrix} a \\ \delta \end{bmatrix}$$

$$\begin{aligned} F_{Y1} &= k_f\alpha_1 \approx k_f\left(\frac{\nu + l_f\omega}{u} - \delta\right) \\ F_{Y2} &= k_r\alpha_2 \approx k_r\frac{\nu - l_r\omega}{u} \end{aligned}$$

#### 数值稳定的离散时间方程

此离散方程具备数值稳定性

$$X_{k+1} = F(X_k, U_k) = \begin{bmatrix} x_k + T_s(u_k\cos\phi_k - v_k\sin\phi_k) \\ y_k + T_s(v_k\cos\phi_k + u_k\sin\phi_k) \\ \phi_k + T_s\omega_k \\ u_k + T_sa_k \\ \frac{mu_kv_k + T_s(l_fk_f - l_rk_r)\omega_k - T_sk_f\delta_ku_k - T_smu_k^2\omega_k}{mu_k - T_s(k_f + k_r)} \\ \frac{I_zu_k\omega_k + T_s(l_fk_f - l_rk_r)v_k - T_sl_fk_f\delta_ku_k}{I_zu_k - T_s(l_f^2k_f + l_r^2k_r)} \end{bmatrix}$$

$$X = \begin{bmatrix} x \\ y \\ \phi \\ u \\ v \\ \omega \end{bmatrix}, U = \begin{bmatrix} a \\ \delta \end{bmatrix}$$

#### 离散时间方程的Jacobian矩阵

根据符号计算，得到Jacobian矩阵

$$A = \frac{\partial F}{\partial X} = \begin{bmatrix} 1 & 0 & -T_s(v\cos(\phi) + u\sin(\phi)) & T_s\cos(\phi) & -T_s\sin(\phi) & 0 \\ 0 & 1 & T_s(u\cos(\phi) - v\sin(\phi)) & T_s\sin(\phi) & T_s\cos(\phi) & 0 \\ 0 & 0 & 1 & 0 & 0 & T_s \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{T_s\delta k_f - mv + 2T_sm\omega u}{mu - T_s(k_f + k_r)} - \frac{m(muv + T_s\omega(k_fl_f - k_rl_r) - T_sm\omega u^2 - T_s\delta k_fu)}{(mu - T_s(k_f + k_r))^2} & \frac{mu}{mu - T_s(k_f + k_r)} & \frac{-T_smu^2 + T_s(k_fl_f - k_{\cdots}}{mu - T_s(k_f + k_r)} \\ 0 & 0 & 0 & \frac{I_z\omega - T_s\delta k_fl_f}{I_zu - T_s(k_fl_f^2 + k_rl_r^2)} - \frac{I_z(I_z\omega u + T_sv(k_fl_f - k_rl_r) - T_s\delta k_fl_fu)}{(I_zu - T_s(k_fl_f^2 + k_rl_r^2))^2} & \frac{T_s(k_fl_f - k_rl_r)}{I_zu - T_s(k_fl_f^2 + k_rl_r^2)} & \frac{I_zu}{I_zu - T_s(k_fl_f^2 + k_rl_{\cdots}^2} \end{bmatrix}$$

$$B=\frac{\partial F}{\partial U}=\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ T_s & 0 \\ 0 & -\frac{T_s k_f u}{mu-T_s(k_f+k_r)} \\ 0 & -\frac{T_s k_f l_f u}{I_z u-T_s(k_f l_f^2+k_r l_r^2)} \end{bmatrix}$$

## 1.2 constraints and cost function

### constraints

可以对状态量和控制量施加约束

$$\begin{bmatrix} x_{min} \\ y_{min} \\ \phi_{min} \\ u_{min} \\ v_{min} \\ \omega_{min} \end{bmatrix} \le \begin{bmatrix} x \\ y \\ \phi \\ u \\ v \\ \omega \end{bmatrix} \le \begin{bmatrix} x_{max} \\ y_{max} \\ \phi_{max} \\ u_{max} \\ v_{max} \\ \omega_{max} \end{bmatrix}$$

$$\begin{bmatrix} a_{min} \\ \delta_{min} \end{bmatrix} \le \begin{bmatrix} a \\ \delta \end{bmatrix} \le \begin{bmatrix} a_{max} \\ \delta_{max} \end{bmatrix}$$

### cost function

$$\begin{gathered} \underset{x_{0:N},u_{0:N-1}}{\text{minimize}} \quad \ell_N(x_N)+\sum_{k=0}^{N-1}\ell_k(x_k,u_k,\Delta t)= \\ x_N^T Q_N x_N+\sum_{k=0}^{N-1}\left(x_k^T Q_k x_k\right)+\left(u_k^T R_k u_k\right) \end{gathered}$$

## 1.3 optimal control problem

参考 AL_ilqr_tutorial.pdf OCP问题：

$$\begin{aligned} &\underset{x_{0:N},u_{0:N-1}}{\text{minimize}} \quad \ell_N(x_N)+\sum_{k=0}^{N-1}\ell_k(x_k,u_k,\Delta t) \\ &\text{subject to} \\ &x_{k+1}=f(x_k,u_k,\Delta t),k=1,\ldots,N-1, \\ &g_k(x_k,u_k)\{\le 0\},\forall k, \\ &h_k(x_k,u_k)=0,\forall k, \end{aligned}$$

$$\begin{aligned} &\underset{x_{0:N},u_{0:N-1}}{\text{minimize}} \quad \ell_N(x_N)+\sum_{k=0}^{N-1}\ell_k(x_k,u_k,\Delta t) \\ &\text{subject to} \\ &x_{k+1}=f(x_k,u_k,\Delta t),k=1,\ldots,N-1, \\ &c_k(x_k,u_k)\le 0,\forall k, \end{aligned}$$

增广拉格朗日法（Augmented Lagranguan）常用来处理约束优化问题。
（为什么不采用罚函数？只有当违反约束后罚函数项取无穷大，罚函数法的最优解才收敛至真实的最优解，但是这种方法在有限数值精度下处理ocp问题是不现实的）
（为什么采用增广拉格朗日法？AL根据约束来估计拉格朗日乘子。）
拉格朗日函数

$$\mathcal{L}_A=f(x)+\lambda^T c(x)+\frac{1}{2}c(x)^T I_\mu c(x)$$

拉格朗日乘子$\lambda$，罚函数乘子$\mu$，等式约束$\mathcal{E}$，不等式约束$\mathcal{I}$

$$
I_\mu = \begin{cases} 0 & \text{if } c_i(x) < 0 \wedge \lambda_i = 0, i \in \mathcal{I} \\ \mu_i & \text{otherwise} \end{cases}
$$

对于符合条件的不等式约束，$\lambda_i$为0，否则为$\mu_i$

al-ocp求解步骤：

1. holding $\lambda$, $\mu$ constant, solving $min_x \mathcal{L}(x, \lambda, \mu)$
2. update $\lambda$ and $\mu$

$$
\lambda_i^+ = \begin{cases} \lambda_i + \mu_i c_i(x^*) & i \in \mathcal{E} \\ \max(0, \lambda_i + \mu_i c_i(x^*) & i \in \mathcal{I}, \end{cases}
$$

$$
\mu^+ = \phi\mu, \phi > 1
$$

3. check constraint convergence
4. if tolerance not met, go to step 1

# 2. backward pass

拉格朗日函数：

$$
\begin{aligned}
\mathcal{L}_A =& \ell_N(x_N) + \left(\lambda_N + \frac{1}{2} c_N(x_N) I_{\mu,N}\right)^T c_N(x_N) \\
&+ \sum_{k=0}^{N-1} [\ell_k(x_k, u_k, \Delta t)] \\
&+ \left(\lambda + \frac{1}{2} c_k(x_k, u_k)^T I_{\mu,k}\right)^T c_k(x_k, u_k)] \\
=& \mathcal{L}_N(x_N, \lambda_N, \mu_N) + \sum_{k=0}^{N-1} \mathcal{L}_k(x_k, u_k, \lambda_k, \mu_k)
\end{aligned}
$$

定义 cost-to-go function 和 action-value function

$$
V_N(x_N)|_{\lambda,\mu} = \mathcal{L}_N(x_N, \lambda_N, \mu_N)
$$

$$
\begin{aligned}
V_k(x_k)|_{\lambda,\mu} &= \min_{u_k} \{\mathcal{L}_k(x_k, u_k, \lambda_k, \mu_k) \\
&\quad + V_{k+1}(f(x_k, u_k, \Delta t))|_{\lambda,\mu}\} \\
&= \min_{u_k} Q(x_k, u_k)|_{\lambda,\mu},
\end{aligned}
$$

cost-to-go function 2-order approximation:

$$
\delta V_k(x) \approx \frac{1}{2} \delta x_k^T P_k \delta x_k + p_k^T \delta x_k
$$

minimize state-action function 2-order approximation with respect to $\delta u$.

$$
\delta Q_k = \frac{1}{2} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix}^T \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} + \begin{bmatrix} Q_x \\ Q_u \end{bmatrix}^T \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix}
$$

$P_k$和$p_k$ 分别是 k 时刻 cost-to-go function 的 Hessian 和 gradient：

矩阵维度

$$
\begin{aligned}
n_s &= \text{state dimension} \\
n_u &= \text{control dimension} \\
n_c &= \text{constraint dimension}
\end{aligned}
$$

$$
\begin{gathered}
V_x : n_s \times 1, V_{xx} : n_s \times n_s \\
Q_x : n_s \times 1, Q_{xx} : n_s \times n_s, Q_{uu} : n_u \times n_u, Q_{ux} : n_u \times n_s \\
\frac{\partial \ell}{\partial x} : n_s \times 1, \frac{\partial \ell}{\partial u} : n_s \times n_s \\
\frac{\partial^2 \ell}{\partial x \partial x} : n_s \times n_s, \frac{\partial^2 \ell}{\partial u \partial u} : n_u \times n_u, \frac{\partial^2 \ell}{\partial u \partial x} : n_u \times n_s \\
\lambda : n_c \times 1, I_\mu : n_c \times n_c, c(x) : n_c \times 1
\end{gathered}
$$

当 $k = N$ 时:

$$
p_N = \frac{\partial V}{\partial x}|_N = (\ell_N)_x + (c_N)_x^T(\lambda + I_{\mu_N} c_N)
$$

$$
P_N = \frac{\partial^2 V}{\partial x^2}|_N = (\ell_N)_{xx} + (c_N)_x^T I_{\mu_N} (c_N)_x
$$

当 $k < N$ 时:

$$
\begin{aligned}
Q_x &= \frac{\partial \ell}{\partial x}|_k + \frac{\partial f}{\partial x}^T|_k \frac{\partial V}{\partial x}^T|_{k+1} + (c_k)_x^T(\lambda + I_{\mu_N} c_k) \\
Q_u &= \frac{\partial \ell}{\partial u}|_k + \frac{\partial f}{\partial u}^T|_k \frac{\partial V}{\partial x}^T|_{k+1} + (c_k)_u^T(\lambda + I_{\mu_N} c_k) \\
Q_{ux} &= \frac{\partial^2 \ell}{\partial u \partial x}|_k + \frac{\partial J}{\partial u}|_k \frac{\partial^2 V}{\partial x^2}^T|_{k+1} \frac{\partial J}{\partial x}|_k + (c_k)_u^T I_{\mu_N} (c_k)_x \\
Q_{xx} &= \frac{\partial^2 \ell}{\partial x \partial x}|_k + \frac{\partial J}{\partial x}|_k \frac{\partial^2 V}{\partial x^2}^T|_{k+1} \frac{\partial J}{\partial x}|_k + (c_k)_x^T I_{\mu_N} (c_k)_x \\
Q_{uu} &= \frac{\partial^2 \ell}{\partial u \partial u}|_k + \frac{\partial J}{\partial u}|_k \frac{\partial^2 V}{\partial x^2}^T|_{k+1} \frac{\partial J}{\partial u}|_k + (c_k)_u^T I_{\mu_N} (c_k)_u
\end{aligned}
$$

此处省略$\delta u_k^*$的推导过程，由两部分组成：反馈和前馈。为了保证正则性，需要对$Q_{uu}$进行正则化。（在cmu16 745 中，对 cost to go 的 Hessian 进行正则化）

$$
\delta u_k^* = -(Q_{uu} + \rho I)^{-1}(Q_{ux}\delta x_k + Q_u) \equiv K_k \delta x_k + d_k
$$

$$
K_k = -(Q_{uu} + \rho I)^{-1} Q_{ux}
$$

$$
d_k = -(Q_{uu} + \rho I)^{-1} Q_u
$$

将最优控制率带入 cost-to-go function 2-order approximation，得到 k 时刻 cost-to-go function 的 Hessian 和 gradient 的闭式解以及 cost-to-go 的change：

$$
\begin{aligned}
P_k &= Q_{xx} + K_k^T Q_{uu} K_k + K_k^T Q_{ux} + Q_{xu} K_k \\
p_k &= Q_x + K_k^T Q_{uu} d_k + K_k^T Q_u + Q_{xu} d_k \\
\Delta V_k &= d_k^T Q_u + \frac{1}{2} d_k^T Q_{uu} d_k.
\end{aligned}
$$

# 3. forward pass

在backward pass中，我们从终端状态计算最优控制率，在forward pass中，基于上一帧/初始状态的nominal trajectory和当前车辆的状态，通过dynamics前向推演出新的nominal trajectory。

$$
\begin{aligned}
\delta x_k &= \bar{x}_k - x_k \\
\delta u_k &= K_k \delta x_k + \alpha d_k \\
\bar{u}_k &= u_k + \delta u_k \\
\bar{x}_{k+1} &= f(\bar{x}_k, \bar{u}_k)
\end{aligned}
$$

## 3.1 line search

在非线性优化中，line search 用来让 cost 充分下降： armijo conditon 和 AL_ilqr_tutorial.pdf 中提到的 line search 准则（下简称BJack）

armijo condition:

$$
\begin{aligned}
x^{k+1} &= x^k + \tau d \\
d &= -\nabla f(x^k)
\end{aligned}
$$

$$
\tau \in \left\{ \alpha \mid f(x^k) - f(x^k + \alpha d) \geq -c \cdot \alpha d^{\mathrm{T}} \nabla f(x^k) \right\}
$$

bjack condition:

$$
z = \frac{J(X, U) - J(\bar{X}, \bar{U})}{-\Delta V(\alpha)}
$$

$$
\Delta V(\alpha) = \sum_{k=0}^{N-1} \alpha d_k^T Q_u + \alpha^2 \frac{1}{2} d_k^T Q_{uu} d_k
$$

z 的意义是 cost 真实下降量与 cost 期望下降量之比。如果 $z \in [\beta_1, \beta_2]$ 通常 $[1e-4, 10]$ ，rollout 的轨迹可以被接受，如果不被接受，则减小 $\alpha$，$\alpha = \gamma\alpha(\gamma < 1)$.

## 3.2 regularization

1. 如果 cost 多次迭代没有降低，或者违反约束造成 cost 爆炸，则进行正则化。通过提升$\rho$，**重新backward pass。**
2. 正则化就是在 backward 之前，对 $Q_{uu}$ 进行正则化。正则项 $\rho$ 增大，则 $Q_{uu}$ 越来越接近 Identity matrix，则高斯牛顿法越接近牛顿梯度下降。
3. 如果在backward pass 的过程中$Q_{uu}$不满秩，则也需要加强正则化，**并重新进行反向传播。**
4. 如果梯度下降方向不正确，通常是Quu矩阵不满秩的缘故。
5. 如果 cost 多次迭代没有降低，还可能是解进入了local minima，需要重新初始化初始解，并**加入随机的噪声**让解离开local minima。

### cmu16-745中提到的正则化方法

在cmu16-745中，使用正则化方法是对 cost to go 的 Hessian 正则化，而不是仅仅对 $Q_{uu}$ 正则化。

```
β = 0.1
# regularization操作，这里实际上是对V(x)进行正则化，而不是S(x,u)
while !isposdef(Symmetric([Gxx Gxu; Gux Guu]))  #保证Hessian矩阵的正定从而保证J的下降
    Gxx += A'*β*I*A
    Guu += B'*β*I*B
    Gxu += A'*β*I*B
    Gux += B'*β*I*A
    β = 2*β
    #display("regularizing G")
    #display(β)
end
```

# 4. multipliers update

前文提到ALM的求解过程：

1. holding $\lambda$, $\mu$ constant, solving $min_x \mathcal{L}(x, \lambda, \mu)$
2. update $\lambda$ and $\mu$

$$
\lambda_i^+ = \begin{cases} \lambda_i + \mu_i c_i(x^*) & i \in \mathcal{E} \\ \max(0, \lambda_i + \mu_i c_i(x^*)) & i \in \mathcal{I}, \end{cases}
$$

$$
\mu^+ = \phi\mu, \phi > 1
$$

3. check constraint convergence
4. if tolerance not met, go to step 1

在内循环计算收敛后，乘子更新规则如下：

$$\lambda_{k_i}^{+} = \begin{cases} \lambda_{k_i} + \mu_{k_i} c_{k_i}(x_k^*, u_k^*) & i \in \mathcal{E}_k \\ \max(0, \lambda_{k_i} + \mu_{k_i} c_{k_i}(x_k^*, u_k^*)) & i \in \mathcal{I}_k, \end{cases}$$

$$\mu^{+} = \phi\mu, \phi > 1$$

相比很多启发式更新方法，最实际有效的更新方法就是每次外循环迭代都更新一次乘子。

# 5. Algorithm

**Algorithm 1 Backward Pass**

1. compute $p_N, P_N$
2. for k = N-1, ..., 0,
   i. compute $Q_x, Q_u, Q_{xx}, Q_{uu}, Q_{ux}$
   ii. if $Q_{uu} > 0$
      a. compute $K, d, \Delta V$
   iii. else
      a. Increase $\rho$, go to 2.
3. return $K, d, \Delta V$

**Algorithm 2 Forward Pass**

1. Initialize $\bar{x}_0 = x_0, alpha = 1, J$
2. for k = 0, ..., N-1
   i. $u_k = \bar{u}_k + K_k(x - \bar{x}_k + \alpha d_k)$
   ii. $x_{k+1} = f(x_k, u_k)$
3. $J = cost(x, u)$
4. if J satisfies line search condition
   i. $X \leftarrow \bar{X}, U \leftarrow \bar{U}$
5. else
   i. reduce $\alpha$, goto 2.
6. return $X, U, J$

**Algorith 3 ILQR**

1. Initialize $x_0, U, tolerance$
2. compute $X = f(x_0, U)$
3. do
   i. $J = cost(X, U)$
   ii. do
      a. $J^- = J$
      b. K, d, $\Delta V$ = backward pass(X, U)
      c. X, U, J = forward pass(X, U, K, d, $\Delta V$, $J^-$)
   iii. while $|J - J^-| > tolerance$
4. return X, U, J

**Algorithm 4 AL ILQR**

1. Initialize $x_0, U, \lambda, \mu, tolerance$
2. set $\phi > 1$
3. compute initial trajectory: $X = f(x_0, U)$
4. repeat outer loop (Argumented Largrangian Method)
   i. repeat inner loop (ilqr loop)
      a. backward pass

b. forward pass

ii. until convergence

iii. update $\lambda, \mu$

5. until no constraints violation
6. return $X, U, J$

**Algorithm 5 Penalty ILQR**

1. Initialize $x_0, U, \lambda, \mu, tolerance$
2. set $\phi > 1$
3. compute initial trajectory: $X = f(x_0, U)$
4. repeat inner loop (ilqr loop)
    i. backward pass
    ii. forward pass
    iii. if constraint violation
        a. increase penality $\lambda$
    iv. else
        a. update(X, U, J)
        b. decrease penality $\lambda$
5. until convergence
6. return $X, U, J$