Robotic Systems II

Lecture 8: Differential Dynamic Programming

Konstantinos Chatzilygeroudis - costashatz@upatras.gr

Department of Electrical and Computer Engineering
University of Patras

Template made by Panagiotis Papagiannopoulos

Pseudocode:

$$
\begin{aligned}
& V_K(\boldsymbol{x}) = \ell_F(\boldsymbol{x}) \\
& k = K \\
& \text{while } k > 1 \\
& \qquad V_{k-1}(\boldsymbol{x}) = \min_{\boldsymbol{u}} \Big[\ell(\boldsymbol{x}, \boldsymbol{u}) + V_k(f_{\text{discrete}}(\boldsymbol{x}, \boldsymbol{u}))\Big] \\
& \qquad k = k - 1
\end{aligned}
$$

If we have $V_k(\boldsymbol{x})$**, then:**

$$\boldsymbol{u}_k(\boldsymbol{x}) = \operatorname*{argmin}_{\boldsymbol{u}} \Big[\ell(\boldsymbol{x}, \boldsymbol{u}) + V_{k+1}(f_{\text{discrete}}(\boldsymbol{x}, \boldsymbol{u}))\Big]$$

LQR with full terms

$$\underset{\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}}{\operatorname{argmin}} \mathcal{J}(\boldsymbol{x}_{1:K},\boldsymbol{u}_{1:K-1}) = \sum_{k=1}^{K-1}\left(\frac{1}{2}\begin{bmatrix}\boldsymbol{x}_k\\ \boldsymbol{u}_k\end{bmatrix}^\top \boldsymbol{Q}_k \begin{bmatrix}\boldsymbol{x}_k\\ \boldsymbol{u}_k\end{bmatrix} + \boldsymbol{q}_k^\top \begin{bmatrix}\boldsymbol{x}_k\\ \boldsymbol{u}_k\end{bmatrix}\right)$$

$$+\frac{1}{2}\boldsymbol{x}_K^T\boldsymbol{Q}_K\boldsymbol{x} + \boldsymbol{q}_K^T\boldsymbol{x}_K$$

$$\text{s.t.} \qquad \boldsymbol{x}_{k+1} = \boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k + \gamma_k$$

where

$$\boldsymbol{Q}_k = \begin{bmatrix}\boldsymbol{Q}_{\boldsymbol{xx},k} & \boldsymbol{Q}_{\boldsymbol{xu},k}\\ \boldsymbol{Q}_{\boldsymbol{ux},k} & \boldsymbol{Q}_{\boldsymbol{uu},k}\end{bmatrix}, \qquad \boldsymbol{q}_k = \begin{bmatrix}\boldsymbol{q}_{\boldsymbol{x},k}\\ \boldsymbol{q}_{\boldsymbol{u},k}\end{bmatrix}$$

Dynamic Programming for LQR

We assume a quadratic value function (cost-to-go):

$$V_k(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{P}_k\boldsymbol{x} + \boldsymbol{p}_k^T\boldsymbol{x} + c_k$$

At $k = K$:

$$\boldsymbol{P}_K = \boldsymbol{Q}_K, \qquad \boldsymbol{p}_K = \boldsymbol{q}_K, \qquad c_K = 0$$

Bellman Recursion:

$$V_k(\boldsymbol{x}) = \min_{\boldsymbol{u}}\Big[\ \underbrace{\ell(\boldsymbol{x},\boldsymbol{u}) + V_{k+1}(\boldsymbol{x}^+)}_{Q_k(\boldsymbol{x},\boldsymbol{u})\text{ Action-Value function}}\ \Big]$$

where $\boldsymbol{x}^+ = \boldsymbol{A}_k\boldsymbol{x} + \boldsymbol{B}_k\boldsymbol{u} + \boldsymbol{\gamma}_k$.

Let's expand the Action-Value function:

$$Q_k(\boldsymbol{x},\boldsymbol{u}) = \underbrace{\frac{1}{2}\begin{bmatrix}\boldsymbol{x}_k\\ \boldsymbol{u}_k\end{bmatrix}^\top \boldsymbol{Q}_k\begin{bmatrix}\boldsymbol{x}_k\\ \boldsymbol{u}_k\end{bmatrix} + \boldsymbol{q}_k^\top\begin{bmatrix}\boldsymbol{x}_k\\ \boldsymbol{u}_k\end{bmatrix}}_{\ell(\boldsymbol{x},\boldsymbol{u})}$$

$$+\underbrace{\frac{1}{2}(\boldsymbol{A}_k\boldsymbol{x} + \boldsymbol{B}_k\boldsymbol{u} + \boldsymbol{\gamma}_k)^\top\boldsymbol{P}_{k+1}(\boldsymbol{A}_k\boldsymbol{x} + \boldsymbol{B}_k\boldsymbol{u} + \boldsymbol{\gamma}_k) + \boldsymbol{p}_{k+1}^\top(\boldsymbol{A}_k\boldsymbol{x} + \boldsymbol{B}_k\boldsymbol{u} + \boldsymbol{\gamma}_k) + c_{k+1}}_{V_{k+1}(\boldsymbol{x}^+)}$$

Dynamic Programming for LQR (2)

$$
\begin{aligned}
V_{k+1}(\boldsymbol{x}^+) &= \frac{1}{2}(\boldsymbol{A}_k\boldsymbol{x} + \boldsymbol{B}_k\boldsymbol{u} + \boldsymbol{\gamma}_k)^\top \boldsymbol{P}_{k+1}(\boldsymbol{A}_k\boldsymbol{x} + \boldsymbol{B}_k\boldsymbol{u} + \boldsymbol{\gamma}_k) + \boldsymbol{p}_{k+1}^\top(\boldsymbol{A}_k\boldsymbol{x} + \boldsymbol{B}_k\boldsymbol{u} + \boldsymbol{\gamma}_k) + c_{k+1}\\
&= \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{A}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{A}_k\boldsymbol{x} + \frac{1}{2}\boldsymbol{u}^\top\boldsymbol{B}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{B}_k\boldsymbol{u} + \boldsymbol{u}^\top\boldsymbol{B}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{A}_k\boldsymbol{x}\\
&\quad + \boldsymbol{x}^\top\boldsymbol{A}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{\gamma}_k + \boldsymbol{u}^\top\boldsymbol{B}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{\gamma}_k + \frac{1}{2}\boldsymbol{\gamma}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{\gamma}_k\\
&\quad + \boldsymbol{x}^\top\boldsymbol{A}_k^\top\boldsymbol{p}_{k+1} + \boldsymbol{u}^\top\boldsymbol{B}_k^\top\boldsymbol{p}_{k+1} + \boldsymbol{p}_{k+1}^\top\boldsymbol{\gamma}_k + c_{k+1}
\end{aligned}
$$

Now we can group terms:

$$
Q_k(\boldsymbol{x},\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^\top\boldsymbol{H}_k\boldsymbol{u} + \boldsymbol{u}^\top\boldsymbol{G}_k\boldsymbol{x} + \boldsymbol{u}^\top\boldsymbol{g}_k + \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{Z}_k\boldsymbol{x} + \boldsymbol{z}_k^\top\boldsymbol{x} + \underbrace{\frac{1}{2}\boldsymbol{\gamma}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{\gamma}_k + \boldsymbol{p}_{k+1}^\top\boldsymbol{\gamma}_k + c_{k+1}}_{\text{constant terms}}.
$$

with

$$
\begin{aligned}
\boldsymbol{H}_k &= \boldsymbol{Q}_{\boldsymbol{uu},k} + \boldsymbol{B}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{B}_k,\\
\boldsymbol{G}_k &= \boldsymbol{Q}_{\boldsymbol{ux},k} + \boldsymbol{B}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{A}_k,\\
\boldsymbol{g}_k &= \boldsymbol{q}_{\boldsymbol{u},k} + \boldsymbol{B}_k^\top(\boldsymbol{P}_{k+1}\boldsymbol{\gamma}_k + \boldsymbol{p}_{k+1}),\\
\boldsymbol{Z}_k &= \boldsymbol{Q}_{\boldsymbol{xx},k} + \boldsymbol{A}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{A}_k,\\
\boldsymbol{z}_k &= \boldsymbol{q}_{\boldsymbol{x},k} + \boldsymbol{A}_k^\top(\boldsymbol{P}_{k+1}\boldsymbol{\gamma}_k + \boldsymbol{p}_{k+1}).
\end{aligned}
$$

Dynamic Programming for LQR (3)

We can now solve the Bellman recursion:

$$V_k(\boldsymbol{x}) = \min_{\boldsymbol{u}} Q_k(\boldsymbol{x}, \boldsymbol{u})$$

Since Q_k is quadratic in $\boldsymbol{u}$, we get $\nabla_{\boldsymbol{u}} Q_k(\boldsymbol{x}, \boldsymbol{u})$ and set it to zero:

$$\begin{aligned} \nabla_{\boldsymbol{u}} Q_k &= \boldsymbol{H}_k \boldsymbol{u} + \boldsymbol{G}_k \boldsymbol{x} + \boldsymbol{g}_k \\ \boldsymbol{u}_k^\star &= -\boldsymbol{H}_k^{-1} \boldsymbol{G}_k \boldsymbol{x} - \boldsymbol{H}_k^{-1} \boldsymbol{g}_k = \underbrace{\boldsymbol{K}_k}_{\text{“Feedback term”}} \boldsymbol{x} + \underbrace{\boldsymbol{k}_k}_{\text{“Feedforward term”}} \end{aligned}$$

with $\boldsymbol{K}_k = -\boldsymbol{H}_k^{-1}\boldsymbol{G}_k, \qquad \boldsymbol{k}_k = -\boldsymbol{H}_k^{-1}\boldsymbol{g}_k$.
Now we subtitute $\boldsymbol{u}^\star$ into V_k:

$$\begin{aligned} V_k(\boldsymbol{x}) &= \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Z}_k \boldsymbol{x} + \boldsymbol{z}_k^\top \boldsymbol{x} - \frac{1}{2}(\boldsymbol{G}_k \boldsymbol{x} + \boldsymbol{g}_k)^\top \boldsymbol{H}_k^{-1} (\boldsymbol{G}_k \boldsymbol{x} + \boldsymbol{g}_k) + \text{const} \\ &= \frac{1}{2}\boldsymbol{x}^\top \left(\boldsymbol{Z}_k - \boldsymbol{G}_k^\top \boldsymbol{H}_k^{-1} \boldsymbol{G}_k\right) \boldsymbol{x} + \left(\boldsymbol{z}_k - \boldsymbol{G}_k^\top \boldsymbol{H}_k^{-1} \boldsymbol{g}_k\right)^\top \boldsymbol{x} + c_k. \end{aligned}$$

Dynamic Programming for LQR (4)

$$V_k(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Z}_k \boldsymbol{x} + \boldsymbol{z}_k^\top \boldsymbol{x} - \frac{1}{2}(\boldsymbol{G}_k \boldsymbol{x} + \boldsymbol{g}_k)^\top \boldsymbol{H}_k^{-1}(\boldsymbol{G}_k \boldsymbol{x} + \boldsymbol{g}_k) + \text{const}$$
$$= \frac{1}{2}\boldsymbol{x}^\top \left(\boldsymbol{Z}_k - \boldsymbol{G}_k^\top \boldsymbol{H}_k^{-1} \boldsymbol{G}_k\right) \boldsymbol{x} + \left(\boldsymbol{z}_k - \boldsymbol{G}_k^\top \boldsymbol{H}_k^{-1} \boldsymbol{g}_k\right)^\top \boldsymbol{x} + c_k.$$

Matching coefficients with $V_k(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{P}_k \boldsymbol{x} + \boldsymbol{p}_k^\top \boldsymbol{x} + c_k$ yields the Riccati recursions:

$$\boxed{\boldsymbol{P}_k = \boldsymbol{Z}_k - \boldsymbol{G}_k^\top \boldsymbol{H}_k^{-1} \boldsymbol{G}_k}$$

$$\boxed{\boldsymbol{p}_k = \boldsymbol{z}_k - \boldsymbol{G}_k^\top \boldsymbol{H}_k^{-1} \boldsymbol{g}_k}$$

and (for optimal cost value – we do not need this for updates)

$$\boxed{c_k = c_{k+1} + \frac{1}{2}\gamma_k^\top \boldsymbol{P}_{k+1}\gamma_k + \boldsymbol{p}_{k+1}^\top \gamma_k - \frac{1}{2}\boldsymbol{g}_k^\top \boldsymbol{H}_k^{-1} \boldsymbol{g}_k.}$$

Computing the improvement of the value function:

$$Q_k(\boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^\top \boldsymbol{H}_k \boldsymbol{u} + \boldsymbol{u}^\top(\boldsymbol{G}_k \boldsymbol{x} + \boldsymbol{g}_k) + \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Z}_k \boldsymbol{x} + \boldsymbol{z}_k^\top \boldsymbol{x} + \mathrm{const.}$$

$$\boldsymbol{b}_k(\boldsymbol{x}) \triangleq \boldsymbol{G}_k \boldsymbol{x} + \boldsymbol{g}_k.$$

$$\boldsymbol{u}_k^\star = -\boldsymbol{H}_k^{-1}\boldsymbol{b}_k(\boldsymbol{x}).$$

$$\min_{\boldsymbol{u}} \left(\frac{1}{2}\boldsymbol{u}^\top \boldsymbol{H}_k \boldsymbol{u} + \boldsymbol{u}^\top \boldsymbol{b}_k(\boldsymbol{x})\right) = -\frac{1}{2}\boldsymbol{b}_k(\boldsymbol{x})^\top \boldsymbol{H}_k^{-1}\boldsymbol{b}_k(\boldsymbol{x}).$$

$$\boxed{\Delta V_k(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{G}_k \boldsymbol{x} + \boldsymbol{g}_k)^\top \boldsymbol{H}_k^{-1}(\boldsymbol{G}_k \boldsymbol{x} + \boldsymbol{g}_k)}$$

Note: if $\boldsymbol{H}_k \succ 0$, then $\Delta V_k(\boldsymbol{x}) \leqslant 0$.

Reminder: Deterministic Optimal Control

Discrete Time:

$$\underset{\boldsymbol{x}_{1:K}, \boldsymbol{u}_{1:K-1}}{\operatorname{argmin}} \mathcal{J}(\boldsymbol{x}_{1:K}, \boldsymbol{u}_{1:K-1}) = \sum_{k=1}^{K-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \ell_F(\boldsymbol{x}_K)$$

$$\text{s.t.} \qquad \boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

where $\boldsymbol{x}_k \in \mathbb{R}^N$ and $\boldsymbol{u}_k \in \mathbb{R}^M$ are vectors.

Differential Dynamic Programming

Key ideas:

1. Linearize the dynamics and quadratically approximate the cost(s),
2. Use LQR on $\Delta \boldsymbol{x}, \Delta \boldsymbol{u}$ to find $\Delta \boldsymbol{u}^\star$,
3. Line-search on $\Delta \boldsymbol{u}^\star$ and actual non-linear dynamics.

This is (almost) equivalent to Sequential Quadratic Programming.

DDP Algorithm

Differential Dynamic Programming (DDP):

1. Start with initial guess $\boldsymbol{U} = \{\boldsymbol{u}_0, \cdots, \boldsymbol{u}_{K-1}\}$,
2. Rollout with non-linear dynamics to get $\boldsymbol{X} = \{\boldsymbol{x}_0, \cdots, \boldsymbol{x}_K\}$,
3. Quadratically approximate the costs around $(\boldsymbol{X}, \boldsymbol{U})$,
4. Linearize the dynamics around $(\boldsymbol{X}, \boldsymbol{U})$,
5. Backward Pass: LQR on $\Delta\boldsymbol{x}, \Delta\boldsymbol{u}$
   for $k = K - 1 \cdots 0$:
   - $\boldsymbol{P}_k = \boldsymbol{Z}_k - \boldsymbol{G}_k^\top \boldsymbol{H}_k^{-1} \boldsymbol{G}_k$
   - $\boldsymbol{p}_k = \boldsymbol{z}_k - \boldsymbol{G}_k^\top \boldsymbol{H}_k^{-1} \boldsymbol{g}_k$
   - $\boldsymbol{K}_k = -\boldsymbol{H}_k^{-1} \boldsymbol{G}_k,$
   - $\boldsymbol{k}_k = -\boldsymbol{H}_k^{-1} \boldsymbol{g}_k,$
   - $\Delta\boldsymbol{u}_k^\star = \boldsymbol{K}_k \Delta\boldsymbol{x}_k + \boldsymbol{k}_k$
6. Forward Rollout with Line Search on $\Delta\boldsymbol{u}_k^\star$
7. Go to 3 if not converged

DDP - Gradients and Hessians

We approximate $\mathcal{J}$ as a quadratic:

$$\begin{aligned} \boldsymbol{q}_{\boldsymbol{x},k} &= \nabla_{\boldsymbol{x}}\ell, & \boldsymbol{q}_{\boldsymbol{u},k} &= \nabla_{\boldsymbol{u}}\ell \\ \boldsymbol{Q}_{\boldsymbol{xx},k} &= \nabla^2_{\boldsymbol{xx}}\ell, & \boldsymbol{Q}_{\boldsymbol{uu},k} &= \nabla^2_{\boldsymbol{uu}}\ell \\ \boldsymbol{Q}_{\boldsymbol{xu},k} &= \nabla^2_{\boldsymbol{xu}}\ell, & \boldsymbol{Q}_{\boldsymbol{ux},k} &= \nabla^2_{\boldsymbol{ux}}\ell \\ \boldsymbol{q}_K &= \nabla_{\boldsymbol{x}}\ell_F, & \boldsymbol{Q}_K &= \nabla^2_{\boldsymbol{xx}}\ell_F \end{aligned}$$

We linearize the dynamics:

$$\boldsymbol{A}_k = \left.\frac{\partial f_{\text{discrete}}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_k,\boldsymbol{u}_k}, \qquad \boldsymbol{B}_k = \left.\frac{\partial f_{\text{discrete}}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_k,\boldsymbol{u}_k}, \qquad \gamma_k = f_{\text{discrete}}(\boldsymbol{x}_k, \boldsymbol{u}_k) - \boldsymbol{x}_{k+1} = \boldsymbol{0}$$

Adapting the recursion coefficients:

$$\begin{aligned} \boldsymbol{Z}_k &= \boldsymbol{Q}_{xx,k} + \boldsymbol{A}_k^\top \boldsymbol{P}_{k+1}\boldsymbol{A}_k + \boldsymbol{\Omega}_{\boldsymbol{xx},k} \\ \boldsymbol{G}_k &= \boldsymbol{Q}_{ux,k} + \boldsymbol{B}_k^\top \boldsymbol{P}_{k+1}\boldsymbol{A}_k + \boldsymbol{\Omega}_{\boldsymbol{xu},k} \\ \boldsymbol{H}_k &= \boldsymbol{Q}_{uu,k} + \boldsymbol{B}_k^\top \boldsymbol{P}_{k+1}\boldsymbol{B}_k + \boldsymbol{\Omega}_{\boldsymbol{uu},k} \end{aligned}$$

$$\boldsymbol{\Omega}_{\boldsymbol{xx},k} \triangleq \underbrace{\boldsymbol{p}_{k+1} \cdot \left.\frac{\partial^2 f_{\text{discrete}}}{\partial \boldsymbol{x}^2}\right|_{\boldsymbol{x}_k,\boldsymbol{u}_k}}_{\text{“contraction of vector with tensor”}}, \qquad \boldsymbol{\Omega}_{\boldsymbol{xu},k} \triangleq \boldsymbol{p}_{k+1} \cdot \left.\frac{\partial^2 f_{\text{discrete}}}{\partial \boldsymbol{x}\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_k,\boldsymbol{u}_k}, \qquad \boldsymbol{\Omega}_{\boldsymbol{uu},k} \triangleq \boldsymbol{p}_{k+1} \cdot \left.\frac{\partial^2 f_{\text{discrete}}}{\partial \boldsymbol{u}^2}\right|_{\boldsymbol{x}_k,\boldsymbol{u}_k}$$

Predicted Improvement on Cost-to-Go in DDP

Local quadratic Q-function (around nominal):

$$Q_k(\Delta \boldsymbol{x}_k, \Delta \boldsymbol{u}_k) = \frac{1}{2}\Delta \boldsymbol{u}_k^\top \boldsymbol{H}_k \Delta \boldsymbol{u}_k + \Delta \boldsymbol{u}_k^\top \boldsymbol{G}_k \Delta \boldsymbol{x}_k + \Delta \boldsymbol{u}_k^\top \boldsymbol{g}_k + \frac{1}{2}\Delta \boldsymbol{x}_k^\top \boldsymbol{Z}_k \Delta \boldsymbol{x}_k + \boldsymbol{z}_k^\top \Delta \boldsymbol{x}_k + \dots.$$

Define the affine term in $\Delta \boldsymbol{u}_k$:

$$\boldsymbol{b}_k(\Delta \boldsymbol{x}_k) \triangleq \boldsymbol{G}_k \Delta \boldsymbol{x}_k + \boldsymbol{g}_k.$$

Optimal perturbation (Riccati/LQR step):

$$\Delta \boldsymbol{u}_k^\star = -\boldsymbol{H}_k^{-1} \boldsymbol{b}_k(\Delta \boldsymbol{x}_k) = \boldsymbol{K}_k \Delta \boldsymbol{x}_k + \boldsymbol{k}_k, \quad \boldsymbol{K}_k = -\boldsymbol{H}_k^{-1}\boldsymbol{G}_k,\ \boldsymbol{k}_k = -\boldsymbol{H}_k^{-1}\boldsymbol{g}_k.$$

Predicted improvement in cost-to-go (minimized $\Delta \boldsymbol{u}$-quadratic):

$$\Delta V_k(\Delta \boldsymbol{x}_k) = -\frac{1}{2}\, \boldsymbol{b}_k(\Delta \boldsymbol{x}_k)^\top \boldsymbol{H}_k^{-1} \boldsymbol{b}_k(\Delta \boldsymbol{x}_k) = -\frac{1}{2}(\boldsymbol{G}_k \Delta \boldsymbol{x}_k + \boldsymbol{g}_k)^\top \boldsymbol{H}_k^{-1} (\boldsymbol{G}_k \Delta \boldsymbol{x}_k + \boldsymbol{g}_k) \leqslant 0.$$

Line-search prediction (at nominal $\Delta \boldsymbol{x}_k = \boldsymbol{0}$):

$$\Delta V_k(\alpha) = \alpha\, \boldsymbol{g}_k^\top \boldsymbol{k}_k + \frac{1}{2}\alpha^2\, \boldsymbol{k}_k^\top \boldsymbol{H}_k \boldsymbol{k}_k \equiv \alpha\, \Delta V_{1,k} + \alpha^2\, \Delta V_{2,k}.$$

DDP Algorithm - Pseudocode

```python
# ---------- Backward pass ----------
# Preallocate
K_list = [np.zeros((m, n)) for _ in range(K_horizon)]
k_list = [np.zeros((m,)) for _ in range(K_horizon)]
dV1_list = np.zeros(K_horizon)
dV2_list = np.zeros(K_horizon)

# Terminal value derivatives
p_k = dterminal_cost_dx(X[-1])          # shape (n,1)
P_k = dterminal_cost_dxx(X[-1])         # shape (n,n)

backward_fail = False

for k in reversed(range(K_horizon)):
    x_k = X[k]
    u_k = U[k]

    A_k = dstep_dx(x_k, u_k, k)             # (n,n)
    B_k = dstep_du(x_k, u_k, k)             # (n,m)

    Ax = dstep_dxx(x_k, u_k, k)             # tensor!
    Bx = dstep_dux(x_k, u_k, k)             # tensor!
    Bu = dstep_duu(x_k, u_k, k)             # tensor!

    q_x  = dstage_cost_dx(x_k, u_k, k)      # (n,1)
    q_u  = dstage_cost_du(x_k, u_k, k)      # (m,1)
    Q_xx = dstage_cost_dxx(x_k, u_k, k)     # (n,n)
    Q_uu = dstage_cost_duu(x_k, u_k, k)     # (m,m)
    Q_ux = dstage_cost_dux(x_k, u_k, k)     # (m,n)

    # Q function derivatives
    z_k  = q_x  + A_k.T @ p_k
    g_k  = q_u + B_k.T @ p_k
    Z_k = Q_xx + A_k.T @ P_k @ A_k + tm(p_k, Ax)
    H_k = Q_uu + B_k.T @ P_k @ B_k + tm(p_k, Bu)
    G_k = Q_ux + B_k.T @ P_k @ A_k + tm(p_k, Bx)

    # Gains
    K_k = -np.linalg.solve(H_k, G_k)        # (m,n)
    k_k = -np.linalg.solve(H_k, g_k)        # (m,1)

    K_list[k] = K_k
    k_list[k] = k_k

    # Expected improvement terms (for line search diagnostics)
    dV1_list[k] = g_k.T @ k_k
    dV2_list[k] = 0.5 * k_k.T @ H_k @ k_k

    # Value function recursion (same as affine LQR)
    p_k = z_k + G_k.T @ k_k
    P_k = Z_k + G_k.T @ K_k
```

DDP Algorithm - Pseudocode (2)

```
# ---------- Forward pass with line search ----------
accepted = False
best_J = np.inf
best_X = None
best_U = None

for alpha in alphas:
    X_new = np.zeros_like(X)
    U_new = np.zeros_like(U)
    X_new[0] = x0
    cost_new = 0.0

    for k in range(K_horizon):
        dx = X_new[k] - X[k]
        du = alpha * k_list[k] + K_list[k] @ dx
        U_new[k] = U[k] + du

        cost_new += stage_cost(X_new[k], U_new[k], k)
        X_new[k+1] = step(X_new[k], U_new[k], k)

    cost_new += terminal_cost(X_new[-1])

    if cost_new < best_J:
        best_J = cost_new
        best_X = X_new
        best_U = U_new

    if cost_new < J:
        accepted = True
        break
```

iterative-LQR (iLQR):

Same as DDP but we throw away $\boldsymbol{\Omega}$ terms in the backward pass and we have:

$$
\begin{aligned}
\boldsymbol{z}_k &= \boldsymbol{q}_{x,k} + \boldsymbol{A}_k^\top\left(\boldsymbol{P}_{k+1}\gamma_k + \boldsymbol{p}_{k+1}\right)\\
\boldsymbol{g}_k &= \boldsymbol{q}_{u,k} + \boldsymbol{B}_k^\top\left(\boldsymbol{P}_{k+1}\gamma_k + \boldsymbol{p}_{k+1}\right)\\
\boldsymbol{Z}_k &= \boldsymbol{Q}_{xx,k} + \boldsymbol{A}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{A}_k\\
\boldsymbol{G}_k &= \boldsymbol{Q}_{ux,k} + \boldsymbol{B}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{A}_k\\
\boldsymbol{H}_k &= \boldsymbol{Q}_{uu,k} + \boldsymbol{B}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{B}_k
\end{aligned}
$$

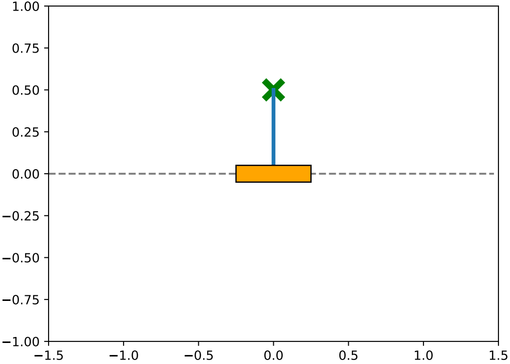
iterative-LQR (iLQR):

Same as DDP but we throw away $\boldsymbol{\Omega}$ terms in the backward pass and we have:

$$
\begin{aligned}
\boldsymbol{z}_k &= \boldsymbol{q}_{x,k} + \boldsymbol{A}_k^\top\left(\boldsymbol{P}_{k+1}\gamma_k + \boldsymbol{p}_{k+1}\right)\\
\boldsymbol{g}_k &= \boldsymbol{q}_{u,k} + \boldsymbol{B}_k^\top\left(\boldsymbol{P}_{k+1}\gamma_k + \boldsymbol{p}_{k+1}\right)\\
\boldsymbol{Z}_k &= \boldsymbol{Q}_{xx,k} + \boldsymbol{A}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{A}_k\\
\boldsymbol{G}_k &= \boldsymbol{Q}_{ux,k} + \boldsymbol{B}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{A}_k\\
\boldsymbol{H}_k &= \boldsymbol{Q}_{uu,k} + \boldsymbol{B}_k^\top\boldsymbol{P}_{k+1}\boldsymbol{B}_k
\end{aligned}
$$

- This is basically *Gauss-Newton* instead of full Newton's method
- iLQR usually needs more iterations, but iterations are much cheaper
- In practice, we usually use just iLQR

DDP - Code Examples



DDP with control limits?

- DDP (like LQR) does not account for control limits
- How can we insert them?
- "Simplest" idea is to just **clamp the controls during the forward pass** (aka simulation), but this is **catastrophic** for convergence!
- Another idea is to define a **squashing function** $\tilde{\boldsymbol{u}}_k = \text{sq}(\boldsymbol{u}_k)$ inside the dynamics
  - All gradients and Hessians include this transformation in the computations
  - Example function: $\text{sq}(\boldsymbol{u}) = \frac{\boldsymbol{u}_{\text{max}} - \boldsymbol{u}_{\text{min}}}{2} \tanh(\boldsymbol{u}) + \frac{\boldsymbol{u}_{\text{max}} + \boldsymbol{u}_{\text{min}}}{2}$

DDP with control limits?

- DDP (like LQR) does not account for control limits
- How can we insert them?
- "Simplest" idea is to just **clamp the controls during the forward pass** (aka simulation), but this is **catastrophic** for convergence!
- Another idea is to define a **squashing function** $\tilde{\boldsymbol{u}}_k = \text{sq}(\boldsymbol{u}_k)$ inside the dynamics
  - All gradients and Hessians include this transformation in the computations
  - Example function: $\text{sq}(\boldsymbol{u}) = \frac{\boldsymbol{u}_{\max} - \boldsymbol{u}_{\min}}{2}\tanh(\boldsymbol{u}) + \frac{\boldsymbol{u}_{\max} + \boldsymbol{u}_{\min}}{2}$
  - This can leads us to bad local minima, create vanishing gradient problems, etc...!

DDP with control limits? (2)

- **A few observations:**
  - The actual system (aka f_{discrete}) **HAS** to enforce control limits
  - DDP chooses new control inputs $\boldsymbol{u}$ through $\Delta\boldsymbol{u}_k$
  - $\Delta\boldsymbol{u}_k$ depends on $\boldsymbol{k}_k$ and $\boldsymbol{K}_k$: $\Delta\boldsymbol{u}_k = \boldsymbol{k}_k + \boldsymbol{K}_k\Delta\boldsymbol{x}$
  - $\boldsymbol{K}_k$ depends on the current state, thus difficult to handle
  - $\boldsymbol{k}_k = -\boldsymbol{H}_k^{-1}\boldsymbol{g}_k$

DDP with control limits? (2)

- **A few observations:**
  - The actual system (aka f_{discrete}) **HAS** to enforce control limits
  - DDP chooses new control inputs $\boldsymbol{u}$ through $\Delta\boldsymbol{u}_k$
  - $\Delta\boldsymbol{u}_k$ depends on $\boldsymbol{k}_k$ and $\boldsymbol{K}_k$: $\Delta\boldsymbol{u}_k = \boldsymbol{k}_k + \boldsymbol{K}_k\Delta\boldsymbol{x}$
  - $\boldsymbol{K}_k$ depends on the current state, thus difficult to handle
  - $\boldsymbol{k}_k = -\boldsymbol{H}_k^{-1}\boldsymbol{g}_k$
- **Key idea:** Enforce $\bar{\boldsymbol{u}}_k + \Delta\boldsymbol{u}$ is inside the bounds!

DDP with control limits? (2)

- **A few observations:**
  - The actual system (aka f_{discrete}) **HAS** to enforce control limits
  - DDP chooses new control inputs $\boldsymbol{u}$ through $\Delta\boldsymbol{u}_k$
  - $\Delta\boldsymbol{u}_k$ depends on $\boldsymbol{k}_k$ and $\boldsymbol{K}_k$: $\Delta\boldsymbol{u}_k = \boldsymbol{k}_k + \boldsymbol{K}_k\Delta\boldsymbol{x}$
  - $\boldsymbol{K}_k$ depends on the current state, thus difficult to handle
  - $\boldsymbol{k}_k = -\boldsymbol{H}_k^{-1}\boldsymbol{g}_k$
- **Key idea:** Enforce $\bar{\boldsymbol{u}}_k + \Delta\boldsymbol{u}$ is inside the bounds!
- Instead of solving directly for $\boldsymbol{k}_k$ using the inverse, we solve the following problem:

$$\boldsymbol{k}_k = \underset{\Delta\boldsymbol{u}}{\operatorname{argmin}}\, Q_k(\bar{\boldsymbol{x}}_k + \Delta\boldsymbol{x}, \bar{\boldsymbol{u}}_k + \Delta\boldsymbol{u})$$
$$s.t.\ \boldsymbol{u}_{\min} \leqslant \bar{\boldsymbol{u}}_k + \Delta\boldsymbol{u} \leqslant \boldsymbol{u}_{\max}$$

DDP with control limits? (3)

- BUT this is actually a QP:

$$\boldsymbol{k}_k = \underset{\Delta\boldsymbol{u}}{\operatorname{argmin}} \frac{1}{2}\Delta\boldsymbol{u}^\top \boldsymbol{H}_k \Delta\boldsymbol{u} + \Delta\boldsymbol{u}^\top \boldsymbol{g}_k$$
$$s.t.\ \boldsymbol{u}_{\min} \leqslant \bar{\boldsymbol{u}}_k + \Delta\boldsymbol{u} \leqslant \boldsymbol{u}_{\max}$$

- **Why?**

DDP with control limits? (3)

- BUT this is actually a QP:

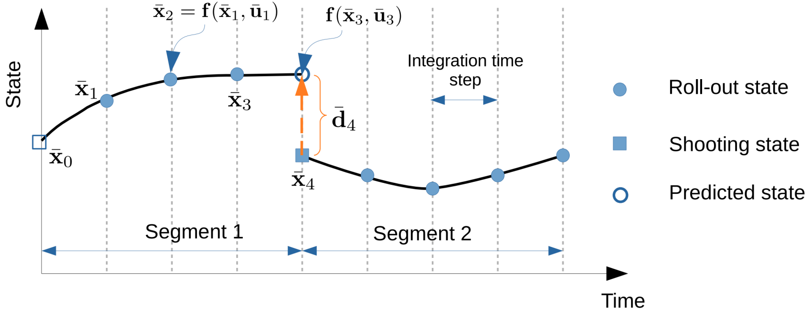
$$\boldsymbol{k}_k = \underset{\Delta\boldsymbol{u}}{\text{argmin}} \frac{1}{2}\Delta\boldsymbol{u}^\top \boldsymbol{H}_k \Delta\boldsymbol{u} + \Delta\boldsymbol{u}^\top \boldsymbol{g}_k$$
$$s.t.\ \boldsymbol{u}_{\min} \leqslant \bar{\boldsymbol{u}}_k + \Delta\boldsymbol{u} \leqslant \boldsymbol{u}_{\max}$$

- **Why?**

$$\begin{aligned} Q_k(\bar{\boldsymbol{x}}_k + \Delta\boldsymbol{x}, \bar{\boldsymbol{u}}_k + \Delta\boldsymbol{u}) &\approx \\ &= Q_k(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \frac{1}{2}\Delta\boldsymbol{u}^\top \boldsymbol{H}_k \Delta\boldsymbol{u} \\ &+ \Delta\boldsymbol{u}^\top(\boldsymbol{G}_k\Delta\boldsymbol{x} + \boldsymbol{g}_k) + \frac{1}{2}\Delta\boldsymbol{x}^\top \boldsymbol{Z}_k \Delta\boldsymbol{x} + \boldsymbol{z}_k^\top \Delta\boldsymbol{x} + \text{const.} \end{aligned}$$

- **Important:** In order to be able to compute $\boldsymbol{K}_k$, we need the QP solver to return us $\boldsymbol{H}_k^{-1}$ or at least a factorization!

Multiple-Shooting DDP



Credits: *Li, H., Yu, W., Zhang, T. and Wensing, P.M., 2023, October. A unified perspective on multiple shooting in differential dynamic programming. In 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 9978-9985).*

DDP vs Trajectory Optimization

- DDP can be very fast (because of the exploitation of the DP structure)
- DDP solutions are always dynamically feasible (even if not successful!)
- DDP comes with a tracking controller (optimally around the linearization of the trajectory)!
- DDP needs more work to handle constraints!
- DDP needs more work to be able to initialize it with an unfeasible trajectory!
- DDP can suffer from numerical issues!

Thank you

- **Any Questions?**
- **Office Hours**:
  - **Tue & Thu** *(09:00-11:00)*
  - 24/7 by email (`costashatz@upatras.gr`, subject: *ECE_RSII_AM*)
- **Material and Announcements**

Laboratory of Automation & Robotics

Math for Hessians of the dynamics!

We have a function $f(\boldsymbol{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Let's look at the 2nd order Taylor expansion:

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \frac{\partial f}{\partial \boldsymbol{x}}\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T\frac{\partial^2 f}{\partial \boldsymbol{x}^2}\Delta\boldsymbol{x}$$

$\frac{\partial f}{\partial \boldsymbol{x}} \in \mathbb{R}^{m\times n}$. What are the dimensions of $\frac{\partial^2 f}{\partial \boldsymbol{x}^2}$?

Math for Hessians of the dynamics!

We have a function $f(\boldsymbol{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Let's look at the 2nd order Taylor expansion:

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \frac{\partial f}{\partial \boldsymbol{x}}\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T\frac{\partial^2 f}{\partial \boldsymbol{x}^2}\Delta\boldsymbol{x}$$

$\frac{\partial f}{\partial \boldsymbol{x}} \in \mathbb{R}^{m\times n}$. What are the dimensions of $\frac{\partial^2 f}{\partial \boldsymbol{x}^2}$? It's a 3D matrix (or a 3rd rank tensor!). But, we can write it as follows:

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \frac{\partial f}{\partial \boldsymbol{x}}\Delta\boldsymbol{x} + \frac{1}{2}\Big(\frac{\partial}{\partial \boldsymbol{x}}\big(\frac{\partial f}{\partial \boldsymbol{x}}\Delta\boldsymbol{x}\big)\Big)\Delta\boldsymbol{x}$$

What do we gain from writing it like this?

Math for Hessians of the dynamics!

We have a function $f(\boldsymbol{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Let's look at the 2nd order Taylor expansion:

$$f(\boldsymbol{x}+\Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \frac{\partial f}{\partial \boldsymbol{x}}\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T\frac{\partial^2 f}{\partial \boldsymbol{x}^2}\Delta\boldsymbol{x}$$

$\frac{\partial f}{\partial \boldsymbol{x}} \in \mathbb{R}^{m\times n}$. What are the dimensions of $\frac{\partial^2 f}{\partial \boldsymbol{x}^2}$? It's a 3D matrix (or a 3rd rank tensor!). But, we can write it as follows:

$$f(\boldsymbol{x}+\Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \frac{\partial f}{\partial \boldsymbol{x}}\Delta\boldsymbol{x} + \frac{1}{2}\Big(\frac{\partial}{\partial \boldsymbol{x}}\big(\frac{\partial f}{\partial \boldsymbol{x}}\Delta\boldsymbol{x}\big)\Big)\Delta\boldsymbol{x}$$

What do we gain from writing it like this?

$\frac{\partial f}{\partial \boldsymbol{x}}\Delta\boldsymbol{x} \in \mathbb{R}^{m\times 1}$!!

Kronnecker product:

$$\underbrace{\boldsymbol{A}}_{\ell\times m} \otimes \underbrace{\boldsymbol{B}}_{n\times p} = \underbrace{\begin{bmatrix} A_{11}\boldsymbol{B} & A_{12}\boldsymbol{B} & \cdots \\ A_{21}\boldsymbol{B} & A_{22}\boldsymbol{B} & \cdots \\ \vdots & \ldots & \ddots \end{bmatrix}}_{\ell n\times mp}$$

Kronnecker product:

$$\underbrace{\boldsymbol{A}}_{\ell\times m} \otimes \underbrace{\boldsymbol{B}}_{n\times p} = \underbrace{\begin{bmatrix} A_{11}\boldsymbol{B} & A_{12}\boldsymbol{B} & \cdots \\ A_{21}\boldsymbol{B} & A_{22}\boldsymbol{B} & \cdots \\ \vdots & \dots & \ddots \end{bmatrix}}_{\ell n\times mp}$$

Vectorization Operator (flattening):

$$\underbrace{\boldsymbol{A}}_{\ell\times m} = \begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2 & \cdots & \boldsymbol{A}_m\end{bmatrix}$$

$$\text{vec}(\boldsymbol{A}) = \underbrace{\begin{bmatrix}\boldsymbol{A}_1 \\ \boldsymbol{A}_2 \\ \vdots \\ \boldsymbol{A}_m\end{bmatrix}}_{\ell m\times 1}$$

The "vec trick"

$$\text{vec}(\boldsymbol{ABC}) = (\boldsymbol{C}^T \otimes \boldsymbol{A})\text{vec}(\boldsymbol{B})$$

We can make one of $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$, the identity matrix and (**"vec trick"**):

$$\begin{aligned}\text{vec}(\boldsymbol{AB}) &= (\boldsymbol{B}^T \otimes \boldsymbol{I})\text{vec}(\boldsymbol{A}) \\ &= (\boldsymbol{I} \otimes \boldsymbol{A})\text{vec}(\boldsymbol{B})\end{aligned}$$

Now we can take the derivative of matrix with respect to a vector:

$$\frac{\partial \boldsymbol{A}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial \text{vec}(\boldsymbol{A})}{\partial \boldsymbol{x}}$$

Also:

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{A}^T(x)\boldsymbol{B}) = (\boldsymbol{B}^T \otimes \boldsymbol{I})\boldsymbol{T}\frac{\partial \boldsymbol{A}}{\partial \boldsymbol{x}}$$

$$\boldsymbol{T}\text{vec}(\boldsymbol{A}) = \text{vec}(\boldsymbol{A}^T)$$

Computing $\frac{\partial}{\partial \boldsymbol{x}}\left(\frac{\partial f}{\partial \boldsymbol{x}}\Delta \boldsymbol{x}\right)$ (2)

$$f(\boldsymbol{x}+\Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}}_{\boldsymbol{A}}\Delta\boldsymbol{x} + \frac{1}{2}(\Delta\boldsymbol{x}^T \otimes \boldsymbol{I})\frac{\partial \mathrm{vec}(\boldsymbol{A})}{\partial \boldsymbol{x}}\Delta\boldsymbol{x}$$

BUT how?!

Computing $\frac{\partial}{\partial \boldsymbol{x}}\left(\frac{\partial f}{\partial \boldsymbol{x}}\Delta x\right)$ (2)

$$f(\boldsymbol{x}+\Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}}_{\boldsymbol{A}}\Delta\boldsymbol{x} + \frac{1}{2}(\Delta\boldsymbol{x}^T \otimes \boldsymbol{I})\frac{\partial \text{vec}(\boldsymbol{A})}{\partial \boldsymbol{x}}\Delta\boldsymbol{x}$$

BUT how?!

$$\begin{aligned}
\text{vec}(\boldsymbol{A}\Delta\boldsymbol{x}) &= \text{vec}(\boldsymbol{I}\boldsymbol{A}\Delta\boldsymbol{x}) \Rightarrow \\
\frac{\partial}{\partial \boldsymbol{x}}\left(\frac{\partial f}{\partial \boldsymbol{x}}\Delta\boldsymbol{x}\right) = \frac{\partial \boldsymbol{A}\Delta\boldsymbol{x}}{\partial \boldsymbol{x}} &= \frac{\partial\left[\text{vec}(\boldsymbol{I}\boldsymbol{A}\Delta\boldsymbol{x})\right]}{\partial \boldsymbol{x}} \\
&= \frac{\partial\left[(\Delta\boldsymbol{x}^T \otimes \boldsymbol{I})\text{vec}(\boldsymbol{A})\right]}{\partial \boldsymbol{x}} \\
&= (\Delta\boldsymbol{x}^T \otimes \boldsymbol{I})\frac{\partial \text{vec}(\boldsymbol{A})}{\partial \boldsymbol{x}}
\end{aligned}$$

DDP with new math!

$$
\begin{aligned}
\boldsymbol{z}_k &= \boldsymbol{q}_{\boldsymbol{x},k} + \boldsymbol{A}_k^T \boldsymbol{p}_{k+1} \\
\boldsymbol{g}_k &= \boldsymbol{q}_{\boldsymbol{u},k} + \boldsymbol{B}_k^T \boldsymbol{p}_{k+1} \\
\boldsymbol{Z}_k &= \boldsymbol{Q}_{\boldsymbol{xx},k} + \boldsymbol{A}_k^T \boldsymbol{P}_{k+1} \boldsymbol{A}_k + (\boldsymbol{p}_{k+1}^T \otimes \boldsymbol{I}) \boldsymbol{T} \frac{\partial \text{vec}(\boldsymbol{A}_k)}{\partial \boldsymbol{x}} \\
\boldsymbol{H}_k &= \boldsymbol{Q}_{\boldsymbol{uu},k} + \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{B}_k + (\boldsymbol{p}_{k+1}^T \otimes \boldsymbol{I}) \boldsymbol{T} \frac{\partial \text{vec}(\boldsymbol{B}_k)}{\partial \boldsymbol{u}} \\
\boldsymbol{G}_k &= \boldsymbol{Q}_{\boldsymbol{ux},k} + \boldsymbol{B}_k^T \boldsymbol{P}_{k+1} \boldsymbol{A}_k + (\boldsymbol{p}_{k+1}^T \otimes \boldsymbol{I}) \boldsymbol{T} \frac{\partial \text{vec}(\boldsymbol{B}_k)}{\partial \boldsymbol{x}}
\end{aligned}
$$