Last Time:

- Convex MPC

Today:

- Nonlinear Trajectory Optimization
- Differential Dynamic Programming a.k.a. Iterative LQR

---

* Nonlinear Traj Opt Problem

$$\min_{x_{1:N},\, u_{1:N-1}} \; J = \sum_{n=1}^{N-1} \ell_k(x_n, u_n) + \ell_N(x_N)$$

nonlinear/nonconvex

$$\text{s.t.} \quad x_{k+1} = f(x_n, u_n)$$

← dynamics constraints (nonlinear)

$$x_n \in \mathcal{X}_n$$
$$u_n \in \mathcal{U}_n$$

} other state/input constraints (nonlinear/nonconvex)

- Usually we assume that costs are C^2 (continuous 2nd derivatives)
- Constraints are usually assumed to be C^1 (continuous first derivatives)

* Differential Dynamic Programming (DDP)

- Nonlinear traj opt method based on approximate DP
- Use 2nd-order Taylor approximations of cost-to-go in DP recursion to compute improvements to our input trajectory and local feedback policy
- Can think of this as a mash-up of Newton's method and DP
- Very fast convergence is possible

- Cost-to-go expansion:

$$V_n(x+\Delta x) \approx V_n(x) + \underbrace{p_n^T}_{\text{gradient}} \Delta x + \frac{1}{2} \Delta x^T \underbrace{P_n}_{\text{Hessian}} \Delta x$$

$$p_N = \nabla_x \ell_N(x) \qquad P_N = \nabla_{xx}^2 \ell_N(x)$$

- Action-Value function expansion:

$$S_n(x+\Delta x, u+\Delta u) \approx S_n(x,u) + \begin{bmatrix} g_x \\ g_u \end{bmatrix}^T \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix}^T \begin{bmatrix} G_{xx} & G_{xu} \\ G_{ux} & G_{uu} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix}$$

$$(G_{xu} = G_{ux}^T)$$

- Bellman Equation:

$$V_{N-1}(x+\delta x) = \min_{\delta u}\Big[S(x,u) + g_x^T \delta x + g_u^T \delta u + \tfrac{1}{2}\delta x^T G_{xx}\delta x + \tfrac{1}{2}\delta u^T G_{uu}\delta u + \tfrac{1}{2}\delta x^T G_{xu}\delta u + \tfrac{1}{2}\delta u^T G_{ux}\delta x\Big]$$

$$\frac{\partial[\,]}{\partial \delta u} = g_u + G_{uu}\delta u + G_{ux}\delta x = 0$$

$$\Rightarrow \boxed{\delta u_{N-1} = \underbrace{-G_{uu}^{-1} g_u}_{d_{N-1}} \underbrace{- G_{uu}^{-1} G_{ux}}_{K_{N-1}} \delta x}$$

"feed-forward term" $= -d_{N-1}$; $-K_{N-1}\delta x$ } "feedback term"

- Plug δu back into S_{N-1} to recover $V_{N-1}(x+\delta x)$

$$\Rightarrow V_{N-1}(x+\delta x) \cong V_{N-1}(x) + g_x^T \delta x + g_u^T(-d_{N-1} - K_{N-1}\delta x) + \tfrac{1}{2}\delta x^T G_{xx}\delta x + \tfrac{1}{2}(d_{N-1} + K_{N-1}\delta x)^T G_{uu}(d_{N-1} + K_{N-1}\delta x) - \tfrac{1}{2}\delta x^T G_{xu}(d_{N-1} + K_{N-1}\delta x) - \tfrac{1}{2}(d_{N-1} + K_{N-1}\delta x)^T G_{ux}\delta x$$

$$\Downarrow$$

$$\boxed{\begin{aligned} P_{N-1} &= G_{xx} + K_{N-1}^T G_{uu} K_{N-1} - G_{xu}K_{N-1} - K_{N-1}^T G_{ux} \\ p_{N-1} &= g_x - K_{N-1}^T g_u + K_{N-1}^T G_{uu} d_{N-1} - G_{xu} d_{N-1} \end{aligned}}$$

- Need some more math to calculate g and G...

* Matrix Calculus

- given $f(x): \mathbb{R}^n \to \mathbb{R}^m$, let's look at 2nd-order Taylor expansion:

- If $m = 1$:

$$f(x+\delta x) \approx f(x) + \underbrace{\frac{\partial f}{\partial x}}_{\mathbb{R}^{1\times n}} \delta x + \frac{1}{2} \delta x^T \underbrace{\frac{\partial^2 f}{\partial x^2}}_{\mathbb{R}^{n\times n}} \delta x$$

- If $m > 1$:

$$f(x+\delta x) \approx f(x) + \underbrace{\frac{\partial f}{\partial x}}_{\mathbb{R}^{m\times n}} \delta x + \frac{1}{2} \underbrace{\left(\frac{\partial}{\partial x}\left[\frac{\partial f}{\partial x}\delta x\right]\right)\delta x}_{\text{ugly!}}$$

- for $m > 1$, $\frac{\partial^2 f}{\partial x^2}$ is a 3rd-rank tensor. Can think of this like a "3D matrix". We need more notation to keep track of which dimension(s) we're multiplying along.

- Kronnecker Product:

$$\underbrace{A}_{l\times m} \otimes \underbrace{B}_{n\times p} = \underbrace{\begin{bmatrix} a_{11}B & a_{12}B & \cdots \\ a_{21}B & a_{22}B & \\ \vdots & & \ddots \end{bmatrix}}_{ln \times mp}$$

(multiply each element of A by matrix B)

- Vectorization Operator

$$\underbrace{A}_{l \times m} = [A_1 \; A_2 \; \cdots \; A_m]$$

$A_1, A_2, \ldots, A_m$: q column vectors

$$\text{vec}(A) = \underbrace{\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix}}_{lm \times 1}$$

- The "vec trick"

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

$$\Rightarrow \text{vec}(AB) = (B^T \otimes I)\,\text{vec}(A) = (I \otimes A)\,\text{vec}(B)$$

- If we want to diff a matrix w.r.t. a vector, vectorize the matrix:

$$\frac{\partial A(x)}{\partial x} = \underbrace{\frac{\partial \text{vec}(A)}{\partial x}}_{lm \times n}$$

(implied whenever we diff a matrix)

- Back to the Taylor expansion of $f(x)$:

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

$\text{vec}(IA\Delta x) = (\Delta x^T \otimes I)\,\text{vec}(A)$

$\frac{1}{2}(\Delta x^T \otimes I)\frac{\partial A}{\partial x}\Delta x$

$mn \times n$

- Sometimes we need to diff through a transpose:

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

T: "commutator matrix"

$$T\,\mathrm{vec}(A) = \mathrm{vec}(A^T)$$

- Action-Value Function Derivatives:

$$S_n(x,u) = \ell_n(x,u) + V_{n+1}(f(x,u))$$

$$\Rightarrow \frac{\partial S}{\partial x} = \frac{\partial \ell}{\partial x} + \frac{\partial V}{\partial f}\underbrace{\frac{\partial f}{\partial x}}_{A} \Rightarrow \boxed{\begin{aligned} q_x &= \nabla_x \ell(x,u) + A_n^T \nabla V(f(x,u)) \\ &= \nabla_x \ell + A_k^T p_{n+1} \end{aligned}}$$

$$\frac{\partial S}{\partial u} = \frac{\partial \ell}{\partial u} + \frac{\partial V}{\partial f}\underbrace{\frac{\partial f}{\partial u}}_{B} \Rightarrow \boxed{\begin{aligned} q_u &= \nabla_u \ell(x,u) + B_n^T \nabla V(f(x,u)) \\ &= \nabla_x \ell + B_n^T p_{n+1} \end{aligned}}$$

$$\boxed{\begin{aligned}
G_{xx} &= \frac{\partial q_x}{\partial x} = \nabla_{xx}^2 \ell(x,u) + A_k^T \nabla^2 V_{n+1} A_k + (p_{n+1}^T \otimes I) T \frac{\partial A_n}{\partial x} \\
G_{uu} &= \frac{\partial q_u}{\partial u} = \nabla_{uu}^2 \ell(x,u) + B_n^T \nabla^2 V_{n+1} B_n + (p_{n+1}^T \otimes I) T \frac{\partial B_n}{\partial u} \\
G_{xu} &= \frac{\partial q_x}{\partial u} = \nabla_{xu}^2 \ell(x,u) + A_n^T \nabla^2 V_{n+1} B_n + (p_{n+1}^T \otimes I) T \frac{\partial A_n}{\partial u} \\
G_{ux} &= \frac{\partial q_u}{\partial x} = \nabla_{ux}^2 \ell(x,u) + B_n^T \nabla^2 V_{n+1} A_n + (p_{n+1}^T \otimes I) T \frac{\partial B_n}{\partial x}
\end{aligned}}$$

"Tensor terms"

- Often tensor terms are left out because they are expensive to compute. Equivalent to doing Gauss-Newton instead of full Newton.
- Version without tensor terms called iLQR
- Forward Rollout:

$\Delta J = 0$

for $k = 1 : N-1$

$u'_k = u_k - \alpha d_k - K_k (x'_k - x_k)$

(line search step length; new value; old value)

$x'_{k+1} = f(x'_k, u'_k)$

$\Delta J \leftarrow \Delta J + \alpha g_{u_k}^T d_k$

(expected cost change based on 1st-order Taylor expansion)

end

- Line Search:

$\alpha = 1$

do:

$x', u', \Delta J = \text{rollout}(x, u, d, K, \alpha)$

$\alpha \leftarrow c\alpha$ (back tracking parameter $\sim 1/2$)

while $J(x', u') > J(x, u) - b\Delta J$

(Armijo tolerance $\sim 10^{-2}$)

- Repeat until convergence

* Example

- Acrobot swing-up
- DDP can converge in fewer iterations but iterations are more expensive. iLQR often wins on wall clock time.
- Problem is nonconvex $\Rightarrow$ can land in different local optima depending on initial guess