Last Time:

- Nonlinear trajectory optimization
- DDP / iLQR

Today:

- DDP details + extensions
- Constraints
- Free / minimum time problems

---

* DDP Recap:

- Solve the unconstrained traj opt problem:

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

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

- Backward Pass:

$$V_n(x+\delta x) \simeq V(x) + \frac{1}{2}\delta x P_n \delta x + p_n^T \delta x$$

$$P_N = \nabla^2 \ell_N(x) \qquad p_N = \nabla \ell_N(x)$$

$$V_{n-1}(x+\delta x) = \min_{\delta u} S(x+\delta x, u+\delta u)$$

$$\Rightarrow \delta u_{n-1} = -d_{n-1} - K_{n-1}\delta x_{n-1}$$

$$P_{n-1} = G_{xx} + K^T G_{uu} K - G_{xu} K - K^T G_{ux}$$
$$p_{n-1} = g_x - K^T g_u + K^T G_{uu} d - G_{xu} d$$

- Forward Rollout:

$\Delta J = 0$

$x_1' = x_1$

for $k = 1:N-1$

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

$x_{k+1}' = f(x_k', u_k')$

$\Delta J \leftarrow \Delta J + \alpha g_k^T d_k$

end

- Line Search:

$\alpha = 1$

do:

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

$\alpha \leftarrow c\alpha$

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

$x, u \leftarrow x', u'$

- Repeate until $\|d\|_\infty < \text{tol}$

- Regularization
  - Just like standard Newton, $V(x)$ and/or $S(x,u)$ Hessians can become indefinite in the backward pass
  - Definitely necessary for DDP, often a good idea with iLQR as well.
  - Many options for regularizing
  - Add a multiple of identity to $\nabla^2 \ell(x,u)$ just like standard Newton.
  - Regularize P_k as needed in the backward
  - Regularize $G_{uu} = \nabla^2_{uu} S(x,u)$ as needed in the backward pass (remember this is the only matrix we have to invert)

$$K = G_{uu}^{-1} G_{ux}, \quad d = G_{uu}^{-1} g_u$$

  - This last one is a good choice for iLQR but not DDP.
  - Regularization shouldn't be required with iLQR but can be necessary due to roundoff/truncation error that accumulates in the backward pass.

* DDP Notes:

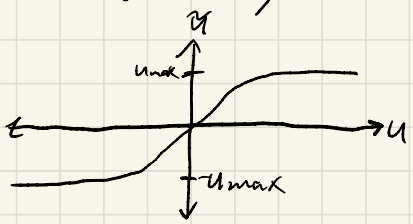
- Can be very fast (iterations + wall clock time)
- One of the most efficient traj opt methods since it exploits DP structure
- Always dynamically feasible due to forward rollout ⇒ can always execute on robot.
- Comes with TVLQR tracking controller for free.

  ⇒ Can be very effective for online use.

- Does not natively handle constraints
- Does not support infeasible initial guess for state trajectory due to forward rollout. Bad for "maze" / "bug trap" scenarios.
- Can suffer from numerical ill-conditioning, especially for long trajectories, due to error accumulation in the backward pass. Careful problem scaling can be needed to make it work.

Handling Constraints with DDP

- Many options depending on type of constraint
- Torque limits are often handled heuristically with a "squashing function" e.g. $\tanh(u)$:



$$\tilde{u} = u_{max} \tanh(u/u_{max})$$

- Effective, but adds nonlinearity and may need more iterations to converge.
- Better option: solve box-constrained QP in the backward pass!

$$\Delta u = \underset{\delta u}{\text{argmin}} \; S(x+\delta x, u+\delta u)$$

$$\text{s.t.} \quad u_{min} \le u + \delta u \le u_{max}$$

- Also need to zero out rows of K corresponding to active torque limit constraints
- State constraints are harder. Often penalties are added to cost function. Can cause ill-conditioning.
- Better option: wrap entire DDP algorithm in an augmented Lagrangian method.
- AL method adds linear (multiplier) and quadratic (penalty) terms to cost function so easily fits into DDP.

- Very effective for achieving fast convergence to coarse constraint tolerances, but can fail to achieve high accuracy due to ill-conditioning

* Handling free initial state:

  - Allow the solver to choose x_1, possibly within some bounds.

  - Add fictitious $k=0$ step to the problem with dynamics:

$$x_1 = f(x_0, u_0) = u_0$$

$$= Ax_0 + Bu_0$$

($A \to 0$, $B \to I$)

  - Now u_0 becomes x_1

* Handling free/minimum time problems:

$$\min_{x(t),\, u(t),\, T} \; J = \int_0^T 1 \, dt$$

← min time problem

$$\text{s.t. } \dot{x} = f(x, u)$$

$$x(T) = x_{goal}$$

$$u_{min} \le u(t) \le u_{max}$$

  - We don't want to change the number of knot points.

  - Make h (time step) from RK a control input

$$x_{n+1} = f(x_n, \tilde{u}_n), \quad \tilde{u} = \begin{bmatrix} u \\ h \end{bmatrix}$$

- Also scale cost by h, e.g.:

$$J(x, \tilde{u}) = \sum_{n=1}^{N-1} h_n \ell(x_n, u_n) + \ell_N(x_N)$$

- This formulation is always nonlinear/non-convex even if dynamics are linear.

- Requires constraints on h, otherwise the solver can "cheat physics" by making h negative or very large to exploit discretization error in the RK method.