Differential Dynamic Programming

(A trajectory optimization technique)

Shubham Kamble & Ahmet Hakan Duman

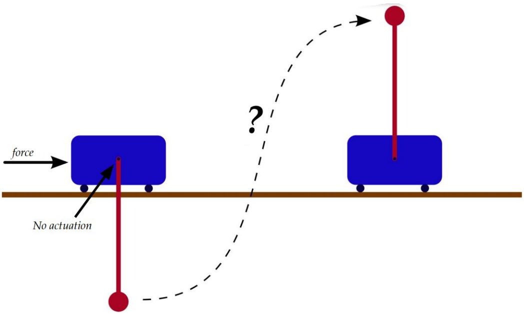
July 12, 2022

Outline

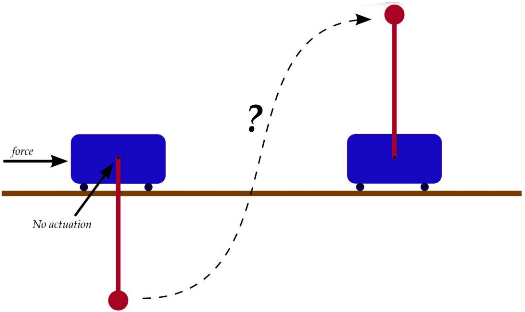
Table of Contents

Trajectory Optimization

Trajectory optimization is the process of designing a trajectory.



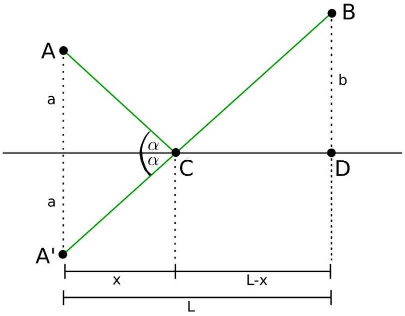
Trajectory Optimization



How do we choose $\mathrm{u} = \pi(\mathrm{x})$ such that the pendulum reaches the target upright position?

General Optimization view of the World

In general, it is a powerful framework, viewed as a way to generate goal-directed behavior (in our case trajectories).

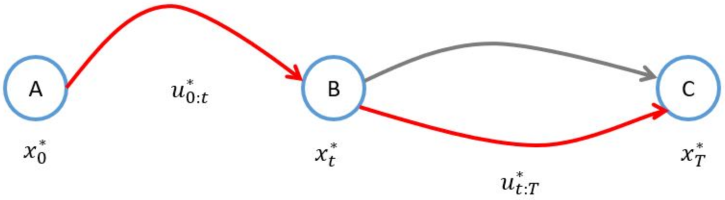


Heron's Problem

In trajectory optimization, we encode this goal-directed behavior using cost and constraint functions.

Principle of Optimality

- If the optimal path from A to C goes through B, then the tail portion of the path that starts in B and ends in C must also be optimal.



Optimal Control Problem for Trajectory Optimization

- Find a policy, $\mathrm{u} = \pi(\mathrm{x})$ that minimizes:

$$U^* = \min_U J_0(\mathrm{x}, \mathrm{U}) = \min_U \sum_{j=0}^{N-1} \ell(\mathrm{x}_j, \mathrm{u}_j) + \ell_f(\mathrm{x}_N)$$

- Subject to the dynamics:

$$\mathrm{x}_{i+1} = f(\mathrm{x}_i, \mathrm{u}_i) \quad \text{(discrete time)}$$

Table of Contents

Dynamic Programming

- Cost: $J(\mathrm{x}, \mathrm{U}) = \sum_{j=0}^{N-1} \ell(\mathrm{x}_j, \mathrm{u}_j) + \ell_f(\mathrm{x}_N)$

- Defining $V^*(\mathrm{x}, i)$ as the lowest possible cost remaining starting with state x at timestep i

- At timestep N, there's nothing to do!
$$V^*(\mathrm{x}, N) = \ell_f(\mathrm{x}_N)$$

Dynamic Programming

- $V^*(\mathrm{x}, N-1) = \boxed{\min_u [\ell(\mathrm{x}, \mathrm{u}) + \ell_f(\mathrm{x}')]} \qquad \mathrm{x}' = f(\mathrm{x}, \mathrm{u})$

- $V^*(\mathrm{x}, N-2) = \min_u \left[\ell(\mathrm{x}, \mathrm{u}) + \boxed{\min_{u'} [\ell(\mathrm{x}', \mathrm{u}') + \ell_f(\mathrm{x}'')]}\right] \quad \mathrm{x}'' = f(\mathrm{x}', \mathrm{u}')$

$$= \min_u [\ell(\mathrm{x}, \mathrm{u}) + V^*(\mathrm{x}, N-1)]$$

- Key idea .... Bellman's equation
  $V^*(\mathrm{x}, i) = \min_u [\ell(\mathrm{x}, \mathrm{u}) + V^*(\mathrm{x}, i+1)]$

Dynamic Programming

- Clearly $V^*(\mathrm{x}, i)$ is useful, so how can we compute it?

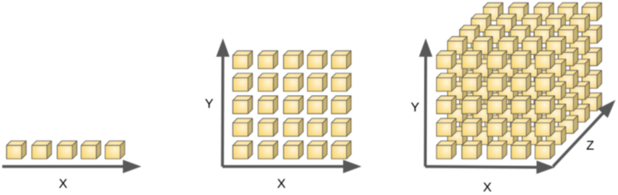
- Turn Bellman equation into an update rule:

$$V^*(\mathrm{x}, i) \leftarrow \min_{u}[\ell(\mathrm{x}, \mathrm{u}) + V^*(\mathrm{x}, i+1)]$$

- Simple Algorithm:
  - Discretize state space.
  - Loop over all states.
    - Apply Bellman update
  - Until V^* converged.

Dynamic Programming

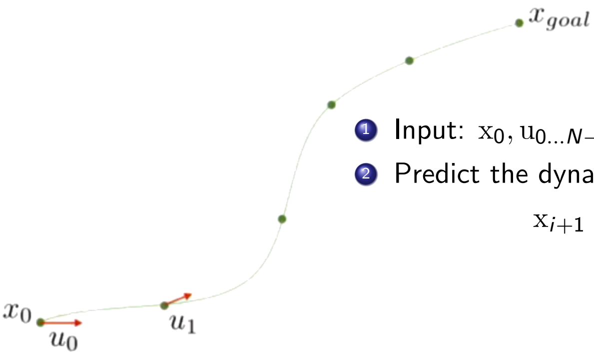
- But as state dimension increases, discretization results in an exponential blowup, ... Curse of dimensionality



- What if we settle for local policies around a trajectory?
  - Which trajectory?
  - How "local"?

Table of Contents

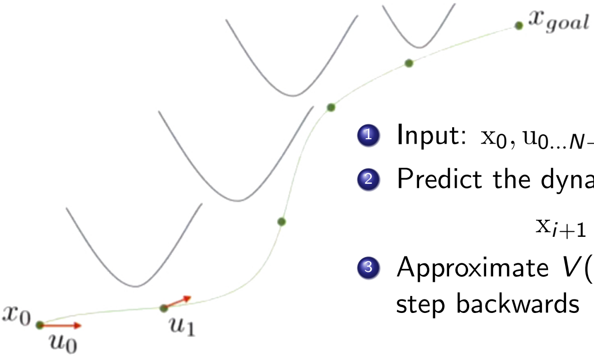
Differential Dynamic Programming



1. Input: $\mathrm{x}_0, \mathrm{u}_{0 \ldots N-1}$
2. Predict the dynamics using:

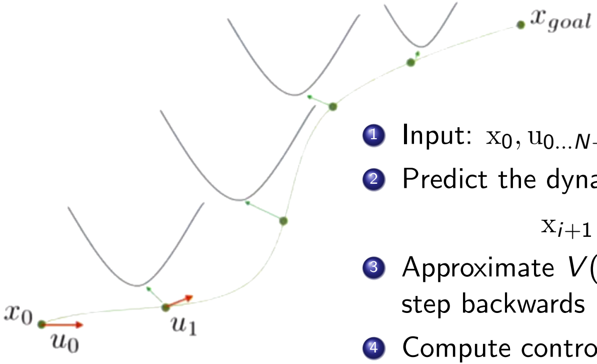
$$\mathrm{x}_{i+1} = f(\mathrm{x}_i, \mathrm{u}_i)$$

Differential Dynamic Programming



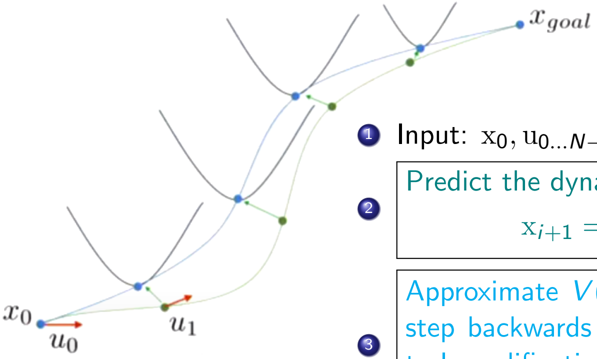
1. Input: $\mathrm{x}_0, \mathrm{u}_{0\ldots N-1}$
2. Predict the dynamics using:
$$\mathrm{x}_{i+1} = f(\mathrm{x}_i, \mathrm{u}_i)$$
3. Approximate $V(\mathrm{x}, i)$ at every time step backwards

Differential Dynamic Programming



1. Input: $\mathrm{x}_0, \mathrm{u}_{0 \ldots N-1}$
2. Predict the dynamics using:
$$\mathrm{x}_{i+1} = f(\mathrm{x}_i, \mathrm{u}_i)$$
3. Approximate $V(\mathrm{x}, i)$ at every time step backwards
4. Compute control modifications using Bellman's equation

Differential Dynamic Programming



1. Input: $\mathrm{x}_0, \mathrm{u}_{0 \ldots N-1}$
2. Predict the dynamics using:
$$\mathrm{x}_{i+1} = f(\mathrm{x}_i, \mathrm{u}_i)$$
3. Approximate $V(\mathrm{x}, i)$ at every time step backwards and compute control modifications using Bellman's equation
4. goto #2 until convergence

Differential Dynamic Programming - Backward Pass

- How do we approximate $V(\mathrm{x}, i)$?
  - Taylor approximation upto second order terms!

- We take the argument of the *min* from the Bellman's equation, $\ell(\mathrm{x}, \mathrm{u}) + V(\mathrm{x}, i+1)$, and perturb it around the i^{th} pair (x, u)

$$
\begin{aligned}
Q(\delta\mathrm{x}, \delta\mathrm{u}) = \ell(\mathrm{x}+\delta\mathrm{x}, \mathrm{u}+\delta\mathrm{u}) - \ell(\mathrm{x}, \mathrm{u}) \\
+V(f(\mathrm{x}+\delta\mathrm{x}, \mathrm{u}+\delta\mathrm{u}), i+1) - V(f(\mathrm{x}, \mathrm{u}), i+1)
\end{aligned}
$$

$$
Q(\delta\mathrm{x}, \delta\mathrm{u}) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta\mathrm{x} \\ \delta\mathrm{u} \end{bmatrix}^T \begin{bmatrix} 0 & Q_\mathrm{x}^T & Q_\mathrm{u}^T \\ Q_\mathrm{x} & Q_\mathrm{xx} & Q_\mathrm{xu} \\ Q_\mathrm{u} & Q_\mathrm{ux} & Q_\mathrm{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathrm{x} \\ \delta\mathrm{u} \end{bmatrix}
$$

Differential Dynamic Programming - Backward Pass

$$Q(\delta\mathrm{x}, \delta\mathrm{u}) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta\mathrm{x} \\ \delta\mathrm{u} \end{bmatrix}^T \begin{bmatrix} 0 & Q_{\mathrm{x}}^T & Q_{\mathrm{u}}^T \\ Q_{\mathrm{x}} & Q_{\mathrm{xx}} & Q_{\mathrm{xu}} \\ Q_{\mathrm{u}} & Q_{\mathrm{ux}} & Q_{\mathrm{uu}} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathrm{x} \\ \delta\mathrm{u} \end{bmatrix}$$

$$Q_{\mathrm{x}} = \ell_{\mathrm{x}} + (f_{\mathrm{x}})^T V'_{\mathrm{x}} \tag{1a}$$

$$Q_{\mathrm{u}} = \ell_{\mathrm{u}} + (f_{\mathrm{u}})^T V'_{\mathrm{x}} \tag{1b}$$

$$Q_{\mathrm{xx}} = \ell_{\mathrm{xx}} + (f_{\mathrm{x}})^T V'_{\mathrm{xx}} f_{\mathrm{x}} + V'_{\mathrm{x}} \cdot f_{\mathrm{xx}} \tag{1c}$$

$$Q_{\mathrm{uu}} = \ell_{\mathrm{uu}} + (f_{\mathrm{u}})^T V'_{\mathrm{xx}} f_{\mathrm{u}} + V'_{\mathrm{x}} \cdot f_{\mathrm{uu}} \tag{1d}$$

$$Q_{\mathrm{ux}} = \ell_{\mathrm{ux}} + (f_{\mathrm{u}})^T V'_{\mathrm{xx}} f_{\mathrm{x}} + V'_{\mathrm{x}} \cdot f_{\mathrm{ux}} \tag{1e}$$

Differential Dynamic Programming - Backward Pass

- How do we get the control modifications for each time step i?
  - Differentiate our approximation $Q(\delta\mathrm{x}, \delta\mathrm{u})$ with respect to $\delta\mathrm{u}$ and equate it to zero.

$$\delta\mathrm{u}^* = \min_{\delta\mathrm{u}} Q(\delta\mathrm{x}, \delta\mathrm{u}) = -Q_{\mathrm{uu}}^{-1}(Q_{\mathrm{u}} + Q_{\mathrm{ux}}\delta\mathrm{x})$$

$$\mathbf{k} = -Q_{\mathrm{uu}}^{-1} Q_{\mathrm{u}} \tag{2a}$$

$$\mathbf{K} = -Q_{\mathrm{uu}}^{-1} Q_{\mathrm{ux}} \tag{2b}$$

Differential Dynamic Programming - Backward Pass

- Plugging this $\delta\mathrm{u}^*$ into $Q(\delta\mathrm{x}, \delta\mathrm{u})$:

$$Q(\delta\mathrm{x}) \approx -\tfrac{1}{2} Q_\mathrm{u} Q_\mathrm{uu}^{-1} Q_\mathrm{u} + \left(Q_\mathrm{x} - Q_\mathrm{u} Q_\mathrm{uu}^{-1} Q_\mathrm{ux}\right)\delta\mathrm{x} + \tfrac{1}{2}\delta\mathrm{x}^T \left(Q_\mathrm{xx} - Q_\mathrm{xu} Q_\mathrm{uu}^{-1} Q_\mathrm{ux}\right)\delta\mathrm{x}$$

- This yields us the Taylor approximation coefficients:

$$\Delta V(i) = -\frac{1}{2} Q_\mathrm{u} Q_\mathrm{uu}^{-1} Q_\mathrm{u} \tag{3a}$$

$$V_\mathrm{x}(i) = Q_\mathrm{x} - Q_\mathrm{u} Q_\mathrm{uu}^{-1} Q_\mathrm{ux} \tag{3b}$$

$$V_\mathrm{xx}(i) = Q_\mathrm{xx} - Q_\mathrm{xu} Q_\mathrm{uu}^{-1} Q_\mathrm{ux} \tag{3c}$$

Differential Dynamic Programming - Forward Pass

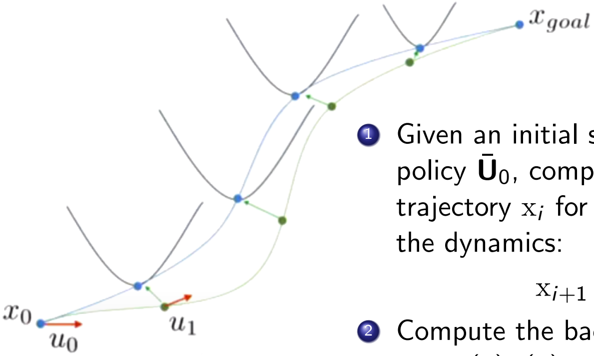
- Once the backward pass is complete by computing (1), (2), and (3), we compute the forward pass using (4)

$$\hat{\mathrm{x}}_0 = \mathrm{x}_0 \tag{4a}$$

$$\hat{\mathrm{u}}_i = \mathrm{u}_i + \mathbf{k}_i + \mathbf{K}_i(\hat{\mathrm{x}}_i - \mathrm{x}_i) \tag{4b}$$

$$\hat{\mathrm{x}}_{i+1} = f(\hat{\mathrm{x}}_i, \hat{\mathrm{u}}_i) \tag{4c}$$

DDP Algorithm



1. Given an initial state x_0 and a nominal policy $\bar{\mathbf{U}}_0$, compute the nominal trajectory x_i for $i = 1, .., N-1$ using the dynamics:

$$\mathrm{x}_{i+1} = f(\mathrm{x}_i, \mathrm{u}_i)$$

2. Compute the backward pass using (1), (2), and (3).
3. Compute the forward pass using (4)
4. goto #2 until convergence

Table of Contents

Rate of Convergence

- Idea: Construct a convergent sequence of $\{\mathrm{x}^k\}_{k=1}^{\infty}$, which fulfills the following condition:
  $\exists \bar{k} \geq 0 : f(x^{k+1}) < f(x^k),\ \forall k > \bar{k}$ and $\lim_{k\to\infty} x^k = x^* \in \mathrm{R}^n$

- Order p of convergence rate: if there exists a constant $C > 0$, such that,

$$\|x^{k+1} - x^*\| \leq C \cdot \|x^k - x^*\|^p$$

- When $p = 2$, it is called quadratic and hence the sequence is said to have quadratic rate of convergence

DDP Quadratic Convergence Theorem

DDP Quadratic Convergence Theorem:

There exists a constant $C > 0$ such that $\|\mathbf{U}^{j+1} - \mathbf{U}^*\| \leq C \cdot \|\mathbf{U}^j - \mathbf{U}^*\|^2$ for all $j \geq 0$ provided $\|\mathbf{U}^0 - \mathbf{U}^*\|$ is sufficiently small and the following set of assumptions are satisfied.

DDP Quadratic Convergence Assumptions I

Assumption 1:

$\ell(\mathrm{x}_i, \mathrm{u}_i)$, $f(\mathrm{x}_i, \mathrm{u}_i)$, and $\ell_f(\mathrm{x}_N)$ have continuous third partial derivatives with respect to x_i and u_i over a closed bounded convex set $D \subset R^{n+m}$, where n and m are the dimensions of x_i and u_i respectively.

Assumption 2:

$\mathbf{U}^j$ is the control trajectory obtained by the j^{th} iteration of the DDP algorithm.

Assumption 3:

$(\mathrm{x}_i^j, \mathrm{u}_i^j)$ and $\mathrm{x}_N^j \in D$ for $i = 0, ..., N-1$ and for all $j > 0$, where x_i^j, $i = 0, ..., N$ and u_i^j, $i = 0, ..., N-1$ are the components of the $\mathbf{x}^j$ and $\mathbf{U}^j$ computed in the j^{th} iteration of the DDP algorithm.

DDP Quadratic Convergence Assumptions II

Assumption 4:

$\ell(\mathrm{x}_i, \mathrm{u}_i)$ and $f(\mathrm{x}_i, \mathrm{u}_i)$, i.e., the running cost functions and transition functions are identical for $i = 0, ..., N-1$

Assumption 5:

The matrices $Q^j_{\mathrm{uu}_i}$ for $i = 0, ..., N-1$ computed at the j^{th} iteration of the DDP algorithm are all positive definite in D.

Assumption 6:

$\mathbf{U}^j$ converges to $\mathbf{U}^*$, where x^* is the trajectory associated with $\mathbf{U}^*$, and $(\mathrm{x}^*_i, \mathrm{u}^*_i)$ and $\mathrm{x}^*_N \in int(D)$ for $i = 0, ..., N-1$.

DDP Quadratic Convergence Proof

- L.-Z. Liao and C. Shoemaker, "The proof of quadratic convergence of differential dynamic programming," Cornell University, Technical Report, 1990. [Online]. Available: https://ecommons.cornell.edu/handle/1813/8800
- ——, "Convergence in unconstrained discrete- time differential dynamic programming," IEEE Transactions on Automatic Control, vol. 36, no. 6, pp. 692–706, 1991.

- Our report!

Table of Contents

Regularization Techniques for Q_{uu_i}

- In order to preserve quadratic convergence it is necessary to have all $Q^j_{\mathrm{uu}_i}$ positive definite.

- There are three ways you can achieve this:
  - Constant Shift:
    $$\tilde{Q}_{\mathrm{uu}} = Q_{\mathrm{uu}} + \mu^c I$$
  - Active Shift:
    $$\tilde{Q}_{\mathrm{uu}} = Q_{\mathrm{uu}} + \mu^a(\delta) I$$
    $$\text{where } \mu^a(\delta) = \begin{cases} \delta - \lambda(Q_{\mathrm{uu}}) & \text{if } \lambda(Q_{\mathrm{uu}}) < \delta, \\ 0 & \text{if } \lambda(Q_{\mathrm{uu}}) \geq \delta \end{cases}$$
    $\lambda(Q_{\mathrm{uu}})$ denotes the minimum eigenvalue of Q_{uu}
  - Adaptive Shift: Combination of above both.

Drawbacks of Above Regularization Techniques

1. Selection of μ is not obvious

2. If μ is too large then the convergence is slow and if it is too small then it may run into numerical difficulties

3. Any of these modifications amounts to adding a quadratic cost around the current control-sequence, making the steps more conservative

Modified Regularizer - Solution to Third Drawback

Therefore introducing a scheme that penalizes deviations from the future states rather than controls is more suitable.

$$\tilde{V}'_{\mathrm{xx}} = V'_{\mathrm{xx}} + \mu I \tag{5}$$

This regularizer places a quadratic state-cost around the previous state sequence. Unlike the standard control-based regularization, the feedback gains $\mathbf{K}_i$ do not vanish as $\mu \to \infty$ but rather force the new trajectory closer to the old one, significantly improving robustness, and thus given a solution to the third drawback.

Modification to the Backward Pass

Finally last two equations from (1), the open loop and feedback gain terms (2), and the value updates (3) are improved as follow:

$$\tilde{Q}_{\mathrm{uu}} = \ell_{\mathrm{uu}} + (f_{\mathrm{u}})^T \tilde{V}'_{\mathrm{xx}} f_{\mathrm{u}} + V'_{\mathrm{x}} \cdot f_{\mathrm{uu}} \tag{6a}$$

$$\tilde{Q}_{\mathrm{ux}} = \ell_{\mathrm{ux}} + (f_{\mathrm{u}})^T \tilde{V}'_{\mathrm{xx}} f_{\mathrm{x}} + V'_{\mathrm{x}} \cdot f_{\mathrm{ux}} \tag{6b}$$

$$\mathbf{k} = -\tilde{Q}_{\mathrm{uu}}^{-1} Q_{\mathrm{u}} \tag{6c}$$

$$\mathbf{K} = -\tilde{Q}_{\mathrm{uu}}^{-1} \tilde{Q}_{\mathrm{ux}} \tag{6d}$$

$$\Delta V(i) = +\frac{1}{2}\mathbf{k}^T Q_{\mathrm{uu}} \mathbf{k} + \mathbf{k}^T Q_{\mathrm{u}} \tag{6e}$$

$$V_{\mathrm{x}}(i) = Q_{\mathrm{x}} + \mathbf{K}^T Q_{\mathrm{uu}} \mathbf{k} + \mathbf{K}^T Q_{\mathrm{u}} + Q_{\mathrm{ux}}^T \mathbf{k} \tag{6f}$$

$$V_{\mathrm{xx}}(i) = Q_{\mathrm{xx}} + \mathbf{K}^T Q_{\mathrm{uu}} \mathbf{K} + \mathbf{K}^T Q_{\mathrm{ux}} + Q_{\mathrm{ux}}^T \mathbf{K} \tag{6g}$$

Modified Regularizer - Solution to the first two Drawbacks

- Resolving the first two drawbacks gives rise to three conflicting requirements.
  - If we are near the minimum we would like μ to quickly go to zero for fast convergence.
  - If the backward pass fails due to a non-PD Q_{uu}, we would like it to increase very rapidly, since the minimum value of μ which prevents divergence is often very large.
  - If we are in a regime where some $\mu > 0$ is required, we would like to accurately tweak it to be as close as possible to the minimum value, but not smaller.

Modified Regularizer - Solution to the first two Drawbacks

How to change μ?

Choose μ_{min} (typically 10^{-6})

Choose Δ_0 (typically 2)

Increase μ :

$$\Delta \leftarrow \max(\Delta_0, \Delta \cdot \Delta_0)$$

$$\mu \leftarrow \max(\mu_{min}, \mu \cdot \Delta)$$

Decrease μ :

$$\Delta \leftarrow \min(\frac{1}{\Delta_0}, \frac{\Delta}{\Delta_0})$$

$$\mu \leftarrow \begin{cases} \mu \cdot \Delta & \text{if } \mu \cdot \Delta > \mu_{min} \\ 0 & \text{if } \mu \cdot \Delta < \mu_{min} \end{cases}$$

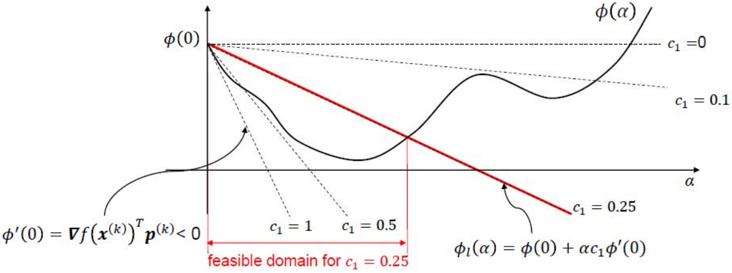
Table of Contents

Backtracking Line Search - Armijo Condition

Armijo Condition Theorem:

Let f be continuously differentiable, $\mathbf{p}^k$ a descent direction, and let $c_1 \in (0, 1)$ be given. Then there exists an $\alpha > 0$, such that for $\phi(\alpha) = f(\mathrm{x}^k + \alpha \mathbf{p}^k)$, the condition $\phi(\alpha) \leq \phi(0) + \alpha c_1 \phi'(0)$ holds.

Geometric Interpretation:



Simple Backtracking Line Search

Simple Backtracking Line Search Algorithm

Choose $\alpha_0 > 0, \tau \in (0,1), c_1 \in (0,1)$

Set $\alpha \leftarrow \alpha_0$

While $\phi(\alpha) \geq \phi(0) + \alpha c_1 \phi'(0)$

$\quad\alpha_k \leftarrow \tau\alpha$

End while

Return $\alpha_k = \alpha$

DDP Backtracking Line Search I

Backtracking line search helps to scale the open loop term of equation (4b) using a parameter α such that $0 < \alpha \leq 1$.

$$\hat{\mathrm{u}}_i = \mathrm{u}_i + \alpha \mathbf{k}_i + \mathbf{K}_i(\hat{\mathrm{x}}_i - \mathrm{x}_i) \tag{7}$$

Therefore as per the new improved equations (7) and (6e), we can better estimate the reduction in the value as,

$$\Delta V(\alpha) = \alpha \sum_{i=0}^{N-1} \mathbf{k}_i^T Q_{\mathrm{u}_i} + \frac{\alpha^2}{2} \sum_{i=0}^{N-1} \mathbf{k}_i^T Q_{\mathrm{uu}_i} \mathbf{k}_i \tag{8}$$

The step size α must be accepted by comparing the predicted estimate with the actual estimate using (9) such that $0 < c < z$ (this condition goes into the while loop of the above backtracking line search).

$$z = (V(\mathrm{x}, 0) - V(\hat{\mathrm{x}}, 0))/\Delta V(\alpha) \tag{9}$$

DDP Backtracking Line Search II

DDP Backtracking Line Search Algorithm

Choose $\alpha_0 > 0, \tau \in (0, 1), c \in (0, 1)$

Set $\alpha \leftarrow \alpha_0$

Set $\Delta V(\alpha) \leftarrow Eq.\ (8)$

Set $z \leftarrow Eq.\ (9)$

While $c \geq z$

$\quad \alpha_k \leftarrow \tau\alpha$

End while

Return $\alpha_k = \alpha$

Algorithm Modified DDP

Inputs:
$\bar{\mathbf{U}}_0, \mathbf{x}_0$
Initialize:
$\mathbf{x}_0 \leftarrow \bar{\mathbf{x}}_0, \quad \mathbf{u}_{0...N-1} \leftarrow \bar{\mathbf{U}}_0, \quad \mathbf{x}_{i+1} \leftarrow f(\mathbf{x}_i, \mathbf{u}_i),\ i = 0, \ldots, N-1, \quad V(\mathrm{N}) \leftarrow \ell_f(\mathbf{x}_N),$
$V_{\mathbf{x}}(N) \leftarrow \frac{\partial}{\partial x}\ell_f(\mathbf{x}_N), \quad V_{\mathbf{xx}}(N) \leftarrow \frac{\partial^2}{\partial x^2}\ell_f(\mathbf{x}_N)$
repeat
backward pass:
for $i \leftarrow N-1$ **to** 0 **do**
$Q_{\mathrm{x}_i}, Q_{\mathrm{u}_i}, Q_{\mathrm{xx}_i}, Q_{\mathrm{uu}_i}, Q_{\mathrm{ux}_i} \leftarrow (1)$
$\tilde{V}_{\mathrm{xx}_i}, \tilde{Q}_{\mathrm{uu}_i}, \tilde{Q}_{\mathrm{ux}_i}, \mathbf{k}_i, \mathbf{K}_i, \Delta V(i), V_{\mathbf{x}}(i), V_{\mathbf{xx}}(i) \leftarrow$ eqs. (5) and (6)
if $\tilde{Q}_{\mathrm{uu}_i}$ **is not** positive-definite **then**
Increase μ
restart *backward pass*
end if
end for
Decrease μ
$\alpha \leftarrow 1, \quad \hat{\mathbf{x}}_0 \leftarrow \mathbf{x}_0$
forward pass:
for $i \leftarrow 0$ **to** $N-1$ **do**
$\hat{\mathrm{u}}_i \leftarrow \mathrm{u}_i + \alpha \mathbf{k}_i + \mathbf{K}_i(\hat{\mathrm{x}}_i - \mathrm{x}_i)$
$\hat{\mathbf{x}}_{i+1} \leftarrow f(\hat{\mathbf{x}}_i, \hat{\mathbf{u}}_i)$
end for
if $z > c > 0$ **then** accept changes
$\mathbf{u}_{0...N-1} \leftarrow \hat{\mathbf{u}}_{0...N-1},$
$\mathbf{x}_{0...N-1} \leftarrow \hat{\mathbf{x}}_{0...N-1}$
else
Decrease α: $\alpha \leftarrow \tau\alpha$
restart *forward pass*.
end if
until convergence

Table of Contents

Cartpole

Let's see the implementation in the Jupyter Notebook!

Table of Contents

Q & A

Any Questions .... feel free ... no silly questions are stupid :)