Differential Dynamic Programming

In paper,
Atkeson and Stephens
Random Sampling of States in Dynamic Programming
NIPS 2007

Presentation by
Sang-Hoon Yeo

Remember LQR,

DT system $x(t+1) = Ax(t) + Bu(t),\ x(0) = x_0$

cost functional $J(U) = \sum_{\tau=0}^{N-1} \left(x(\tau)^T Q x(\tau) + u(\tau)^T R u(\tau) \right) + x(N)^T Q_f x(N)$

DP principle $V_t(z) = \min_w \left(z^T Q z + w^T R w + V_{t+1}(Az + Bw) \right)$

optimal policy $w^* = -(R + B^T P_N B)^{-1} B^T P_N A z$

Riccatti recursion $P_{N-1} = Q + A^T P_N A - A^T P_N B (R + B^T P_N B)^{-1} B^T P_N A$

What is Differential Dynamic Programming?

Applying LQR to the linearized model around a given trajectory
(for DTS: a sequence of points to the goal)

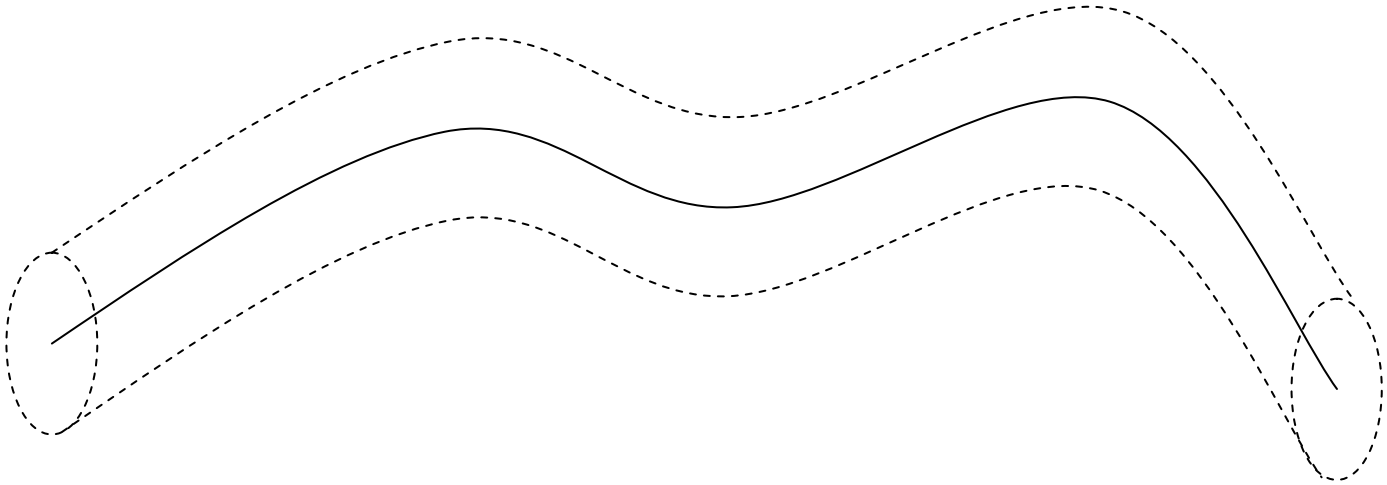
Linearized model includes (for each point)

- a linear model of the system
- a quadratic model of one step cost

By applying LQR, we can get (for each point)

- an improved quadratic model of value function
- an improved linear model of policy

Any nonlinear system can be linearized in the neighborhood of a point (or a trajectory)



- We can do the linear analysis within “tube”

Linearized dynamics of “deviations”: state and policy

Linearization

$$\mathbf{x}_{k+1} = \mathbf{f}^i(\mathbf{x}, \mathbf{u}) \approx \mathbf{f}^i_0 + \mathbf{f}^i_x \hat{\mathbf{x}} + \mathbf{f}^i_u \hat{\mathbf{u}}$$

$$L^i(\mathbf{x}, \mathbf{u}) \approx L^i_0 + L^i_x \hat{\mathbf{x}} + L^i_u \hat{\mathbf{u}} + \tfrac{1}{2} \hat{\mathbf{x}}^{\mathrm{T}} L^i_{xx} \hat{\mathbf{x}} + \tfrac{1}{2} \hat{\mathbf{u}}^{\mathrm{T}} L^i_{uu} \hat{\mathbf{u}}$$

$$V^p(\mathbf{x}) \approx V^p_0 + V^p_x \hat{\mathbf{x}} + \frac{1}{2} \hat{\mathbf{x}}^{\mathrm{T}} V^p_{xx} \hat{\mathbf{x}}$$

$$\mathbf{u}^p(\mathbf{x}) = \mathbf{u}^p_0 - \mathbf{K}^p \hat{\mathbf{x}}$$

Algorithm

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \approx \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$L(\mathbf{x}, \mathbf{u}) \approx \frac{1}{2}(\mathbf{x} - \mathbf{x}_d)^T \mathbf{Q}(\mathbf{x} - \mathbf{x}_d) + \frac{1}{2}(\mathbf{u} - \mathbf{u}_d)^T \mathbf{R}(\mathbf{u} - \mathbf{u}_d) + (\mathbf{x} - \mathbf{x}_d)^T \mathbf{S}(\mathbf{u} - \mathbf{u}_d)$$

Backward sweep

$$Z_x = V_x \mathbf{A} + \mathbf{Q}(\mathbf{x} - \mathbf{x}_d)$$

$$Z_u = V_x \mathbf{B} + \mathbf{R}(\mathbf{u} - \mathbf{u}_d)$$

$$Z_{xx} = \mathbf{A}^T V_{xx} \mathbf{A} + \mathbf{Q}$$

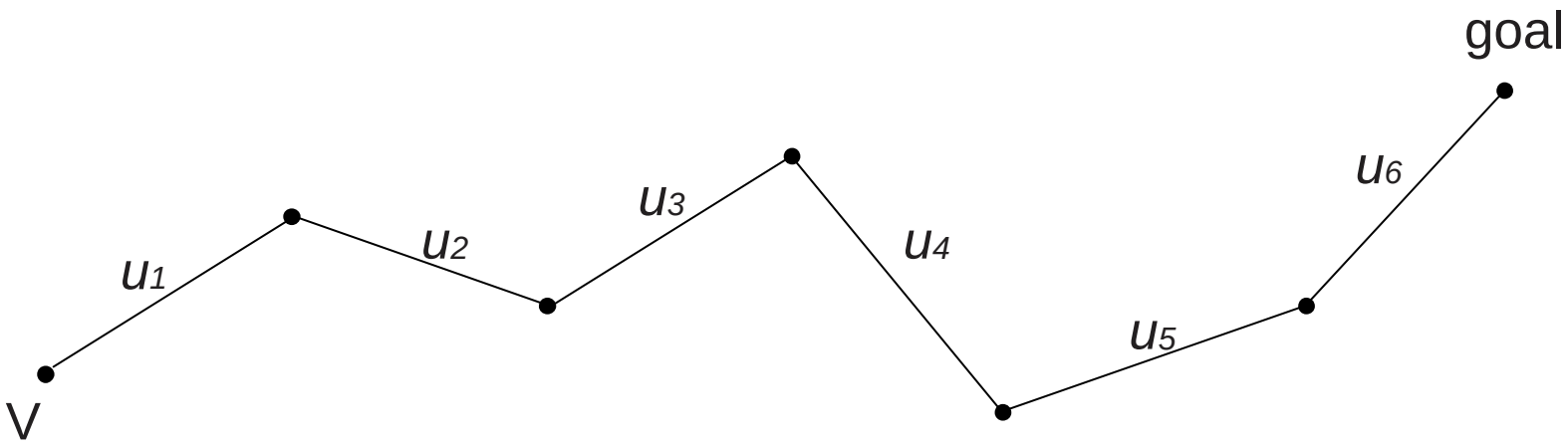
$$Z_{ux} = \mathbf{B}^T V_{xx} \mathbf{A} + \mathbf{S}$$

$$Z_{uu} = \mathbf{B}^T V_{xx} \mathbf{B} + \mathbf{R}$$

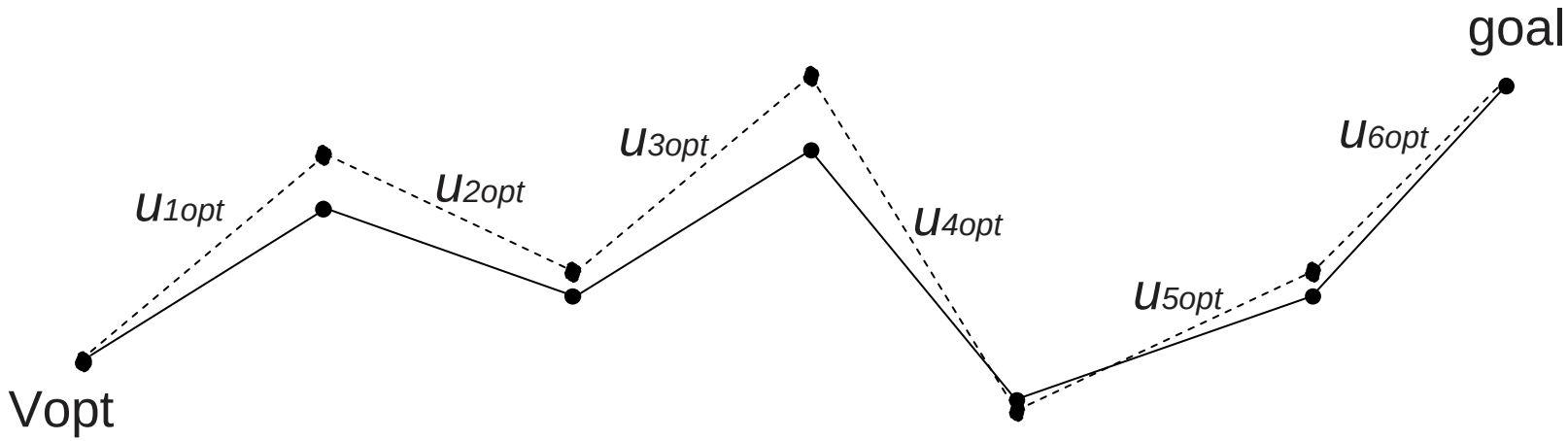
$$\mathbf{K} = Z_{uu}^{-1} Z_{ux}$$

$$V_{x_{k-1}} = Z_x - Z_u \mathbf{K}$$

$$V_{xx_{k-1}} = Z_{xx} - Z_{xu} \mathbf{K}$$



Suppose we have initial guesses of value and policies to the goal



DDP gives a new trajectory with *improved policies and value*

How to use DDP to construct the value function?

Suppose we already have sample points which have quadratic model of the value function and the a linear model of the policy

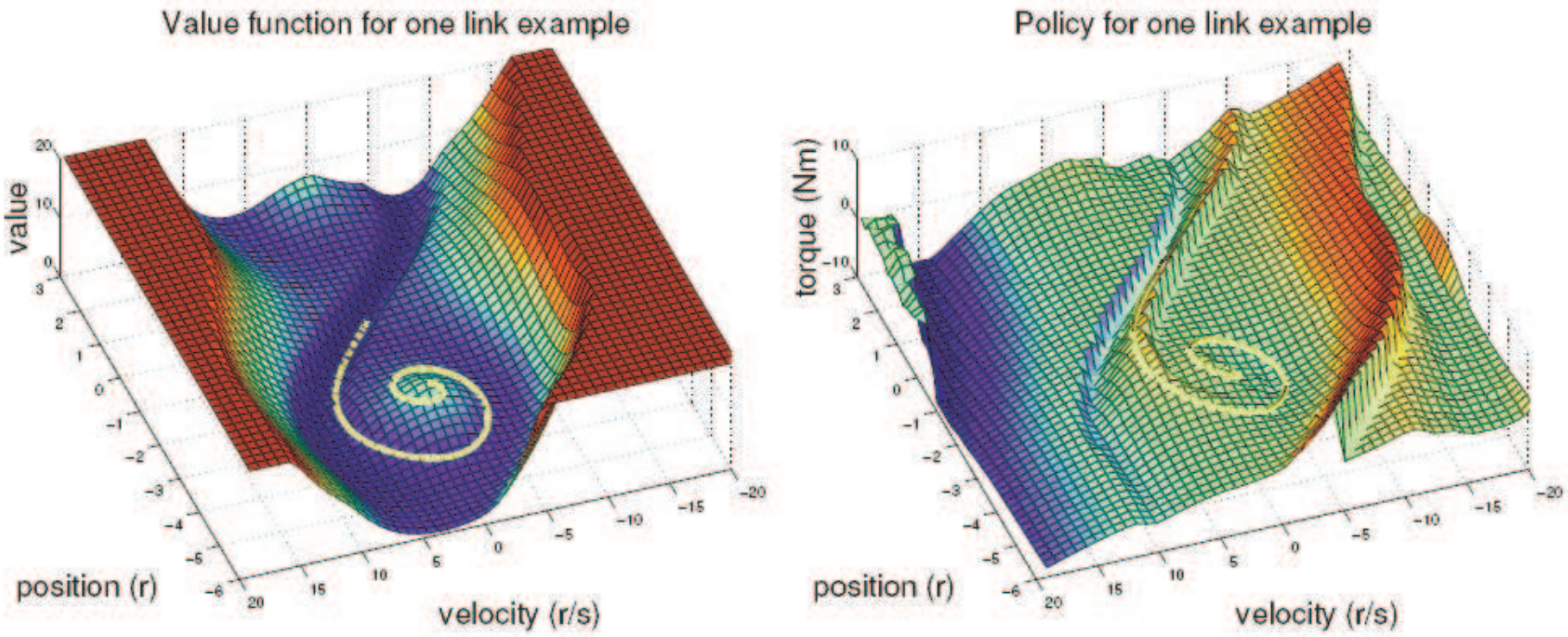
Randomly pick a new sample point

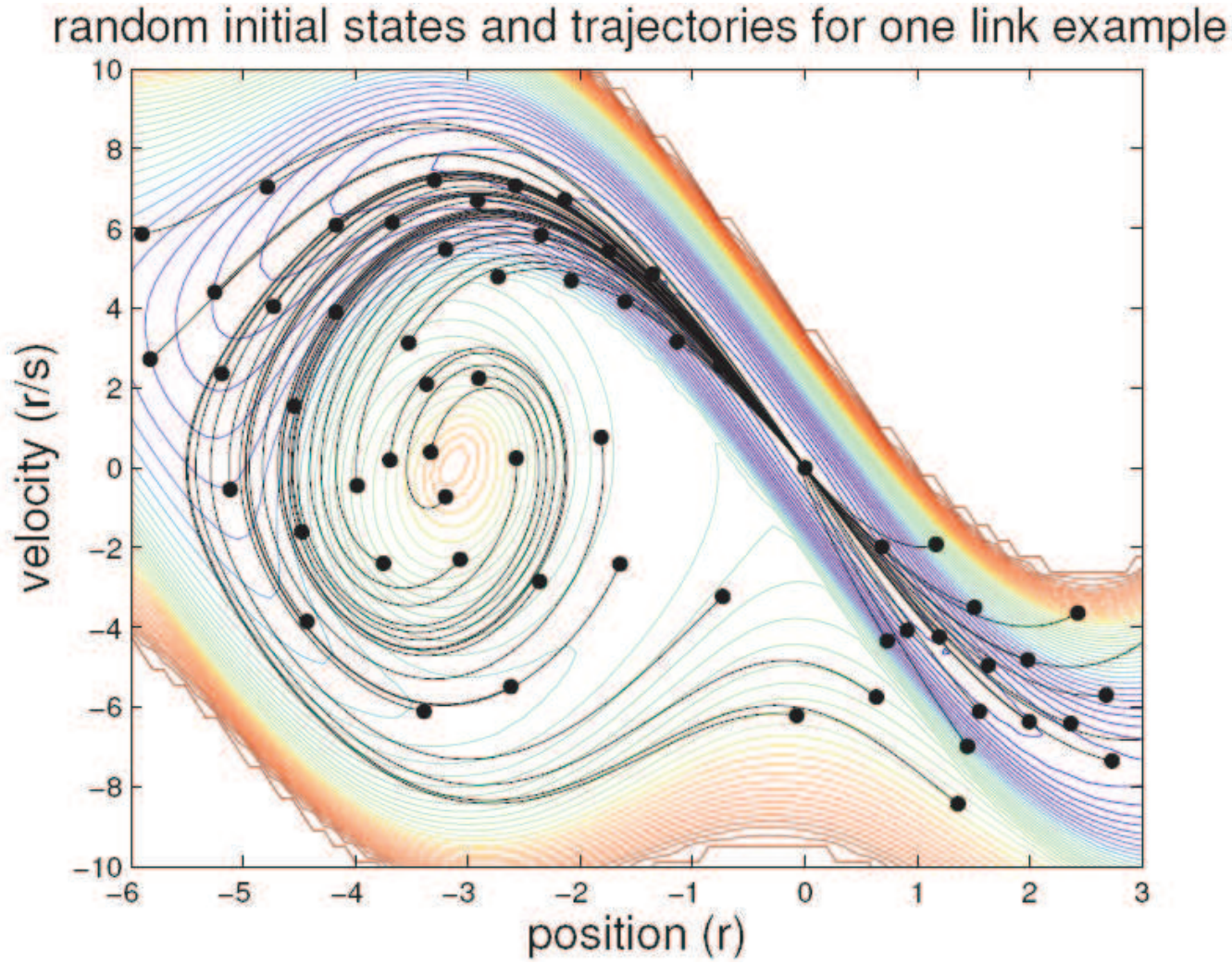
Find the nearest sample point and predict the value and policy of newly picked point

Proceed to the next time step and do it again until we hit the goal

Apply DDP backward sweep to compute improved value and policy of the newly picked point

One link inverted pendulum: value function and policy




random initial states and trajectories for one link example
velocity (r/s)
position (r)
10
8
6
4
2
0
−2
−4
−6
−8
−10
−6
−5
−4
−3
−2
−1
0
1
2
3

-Q: How to overcome the curse of dimensionality?

A: Only store “surprises”, otherwise forget

When the computed value of a newly picked point is very different with it’s previous prediction by neighborhood

Q: What about discontinuities? Should we store more points near discontinuities?

A: Don’t worry, trajectories typically move away from the discontinuities

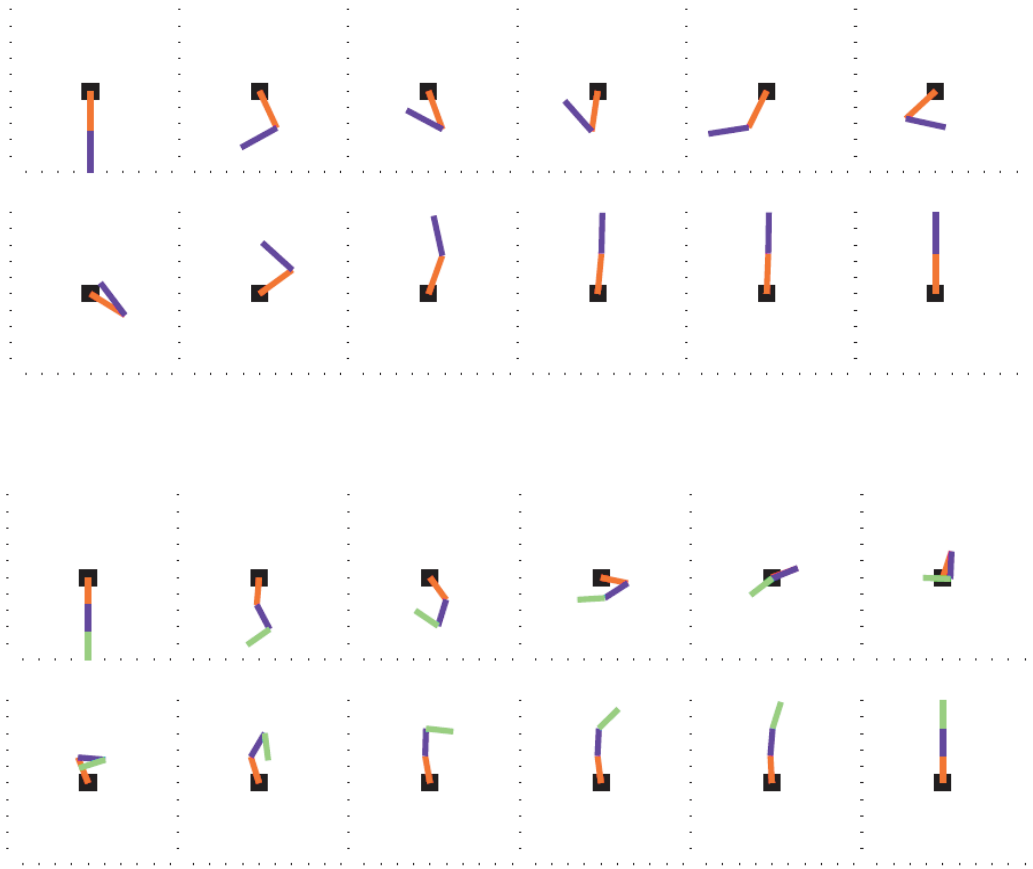
Global optimization using greedy local optimizers

- Global optimization is achieved when all the stored states are "consistent"; any state's local model correctly predicts values of nearby states

- Enforce "consistency" by using the policy of one state of a pair to re-optimize the trajectory of the other state of the pair and vice versa

- Add more stored states in between neighbors that continue to disagree

inverted pendulums



Conclusion

Combined three approaches for DP approximation

1. Random sampling

   J Rust “Using randomization to break the curse of dimensionality” *Econometrica 1997*

2. Differential Dynamic Programming

3. Global optimization using local optimizers

“This paper we show that we can now solve problems we couldn’t solve previously”

Questions?