Planning using dynamic optimization

Problem characteristics

- Want optimal plan, not just feasible plan
- We will minimize a cost function C(execution). Some examples:
- $C() = c_T(x_T) + \Sigma c(x_k,u_k)$: deterministic with explicit terminal cost function
- $C() = E(c_T(x_T) + \Sigma c(x_k,u_k))$: stochastic

Examples

- A number of us are currently working on humanoid locomotion. We would like the humanoid to be able to walk, run, vary speed, turn, sit, get up from a chair, handle steps, kick a ball, avoid obstacles, handle rough terrain, …

Dynamic Optimization

- General methodology is dynamic programming (DP).
- We will talk about ways to apply DP.
- Requirement to represent all states, and consider all actions from each state, lead to "curse of dimensionality": $R_x^{dx} \bullet R_u^{du}$
- We will talk about special purpose solution methods.

Dynamic Optimization Issues

- Discrete vs. continuous states and actions?
- Discrete vs. continuous time?
- Globally optimal?
- Stochastic vs. deterministic?
- Clocked vs. autonomous?
- What should be optimized, anyway?

Policies vs. Trajectories

- $u(t)$ open loop trajectory control
- $u = u_{ff}(t) + K(x – x_d(t))$ closed loop trajectory control
- $u(x)$ policy

Types of tasks

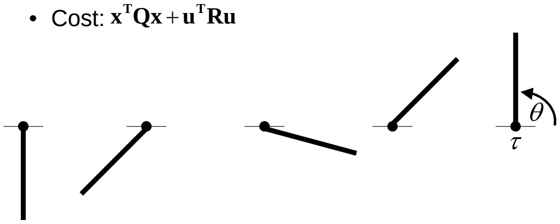
- Regulator tasks: want to stay at x_d
- Trajectory tasks: go from A to B in time T, or attain goal set G
- Periodic tasks: cyclic behavior such as walking

Typical reward functions

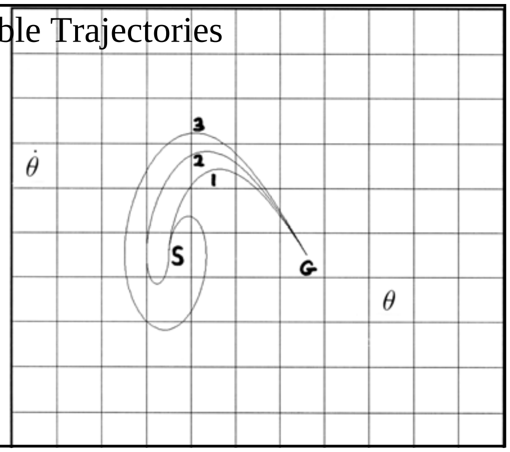
- Minimize error
- Minimum time
- Minimize tradeoff of error and effort

Example: Pendulum Swingup

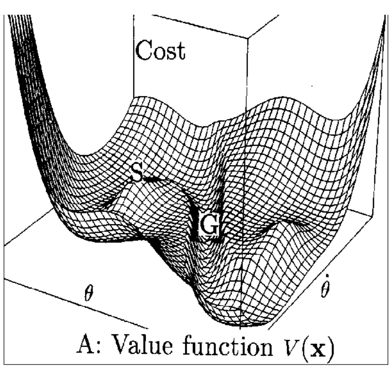
- State: $\mathbf{x} = (\theta, \dot{\theta})$
- Action: $\mathbf{u} = (\tau)$
- Cost: $\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}$

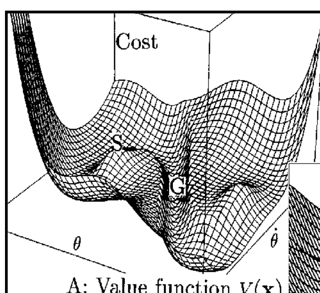


Possible Trajectories

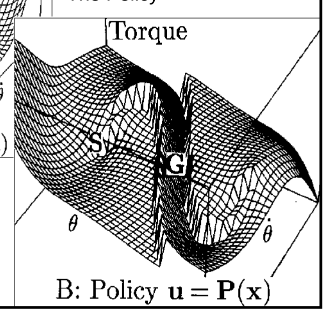


Global Planning Using Dynamic Programming

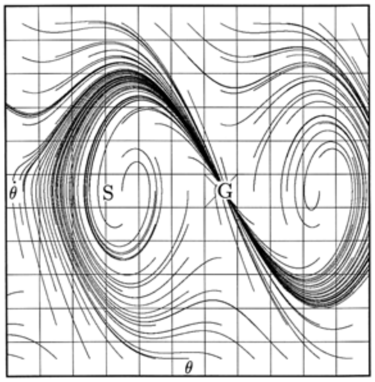




The Policy



Trajectories



Discrete-Time Deterministic Dynamic Programming (DP)

- Fudging on whether states are discrete or continuous.

How to do Dynamic Programming (specified end time T)

- Dynamics: $x_{k+1} = f(x_k,u_k)$
- Cost: $C() = c_T(x_T) + \Sigma c(x_k,u_k)$
- Value function $V_k(x)$ is represented by table.
- $V_T(x) = c_T(x)$
- For each x, $V_k(x) = \min_u(c(x,u) + V_{k+1}(f(x,u)))$
- This is Bellman's Equation
- This version of DP is value iteration
- Can also tabulate policy: $u = \pi_k(x)$

How to do Dynamic Programming (no specified end time)

- Cost: $C() = \Sigma c(x_k,u_k)$
- $V_N(x)$ = a guess, or all zeros.
- Apply Bellman's equation.
- $V(x)$ is given by $V_k(x)$ when V stops changing.
- Goal needs to have zero cost, or need to discount so V() does not grow to infinity:
- $V_k(x) = \min_u(c(x,u) + \gamma V_{k+1}(f(x,u)))$, $\gamma < 1$

Policy Iteration

- $u = \pi(x)$: general policy (a table in discrete state case).
- *) Compute $V^\pi(x)$:

$V^\pi_k(x) = c(x,\pi(x)) + V^\pi_{k+1}(f(x,\pi(x)))$

- Update policy $\pi(x) = \text{argmin}_u(c(x,u) + V^\pi(f(x,u)))$
- Goto *)

Stochastic Dynamic Programming

- Cost: $C() = \Sigma E(c(x_k,u_k))$
- Bellman's equation now involves expectations:
- $V_k(x) = \min_u E(c(x,u) + V_{k+1}(f(x,u)))$
  $= \min_u(c(x,u) + \Sigma p(x_{k+1})V_{k+1}(x_{k+1}))$
- Modified Bellman's equation applies to value and policy iteration.
- May need to add discount factor.

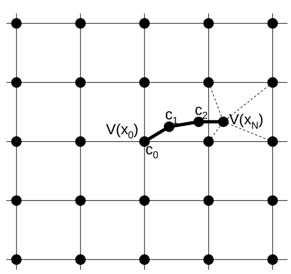
Continuous State DP

- Time is still discrete.
- How do we discretize the states?

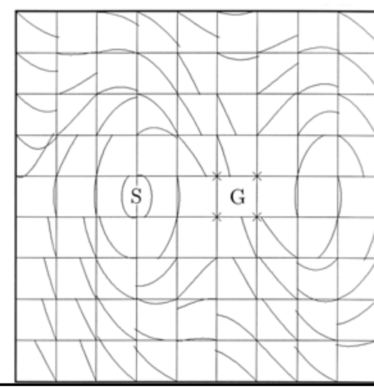
How to handle continuous states.

- Discretize states on a grid.
- At each point (x_0), generate trajectory segment of length N by minimizing $C(u) = \Sigma c(x_k,u_k) + V(x_N)$
- $V(x_N)$: interpolate using surrounding V()
- Typically multilinear interpolation used.
- N typically determined by when $V(x_N)$ independent of $V(x_0)$
- Use favorite continuous function optimizer to search for best u when minimizing C(u)
- Update V() at that cell.

State Increment Dynamic Programming (Larson)



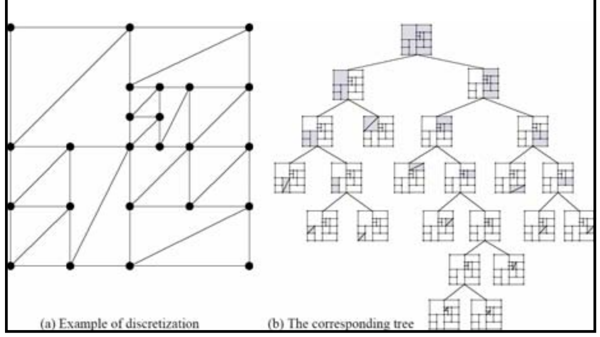
State Increment Dynamic Programming



Munos and Moore, Variable Resolution Discretization in Optimal Control

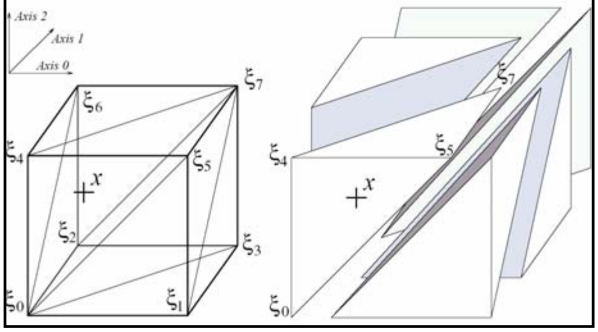
Machine Learning, 49 (2/3), 291-323, 2002

Kuhn Triangulation, kd-trie

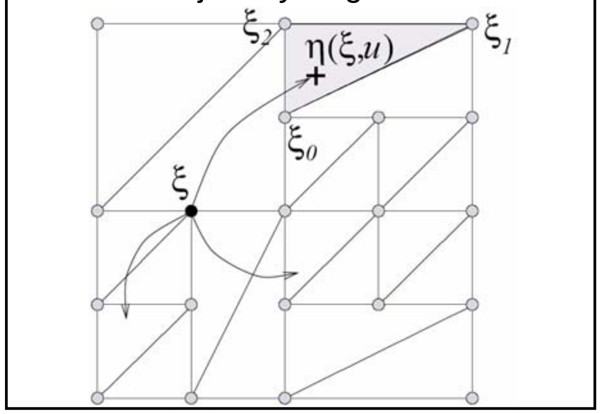


(a) Example of discretization (b) The corresponding tree

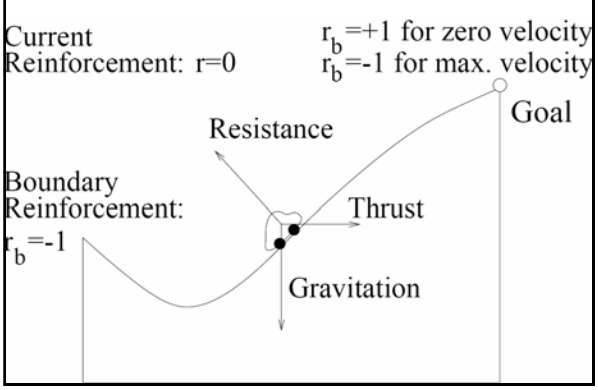
Kuhn Triangulation in 3D



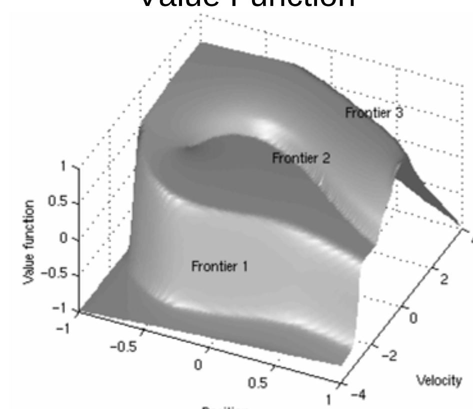
Trajectory Segments



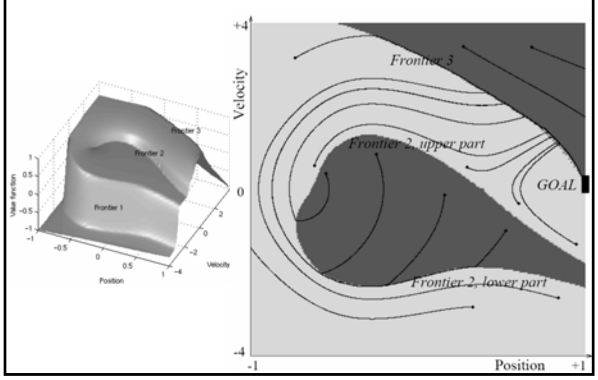
Mountain Car



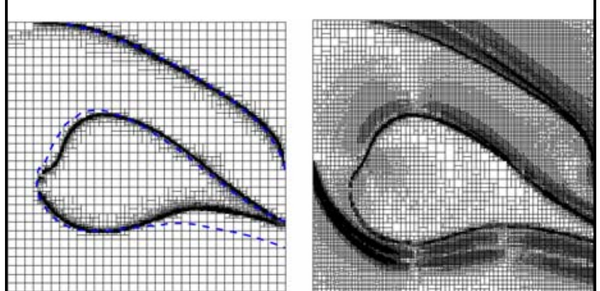
Value Function



Value Function



Discretizations



Policy Iteration: Continuous State

- Discretize states
- Represent policy at discretized states u(x)
- Each cell in table has constant u, or
- u as knot points for linear or higher order spline
- *) Same kind of trajectory segments used to compute $V^{\pi}_{k}(x) = \Sigma c(x,\pi(x)) + V^{\pi}_{k+1}(x_N)$
- Optimize policy $\pi(x) = \mathrm{argmin}_u(c(x,u) + V^{\pi}(f(x,u)))$ using favorite continuous function optimizer.
- Goto *)

Stochastic DP: Continous State

- Cost: $C() = \Sigma E(c(x_k,u_k))$
- Do Monte Carlo sampling of process noise for each trajectory segment (many trajectory segments), or
- Propagate analytic distribution (see Kalman filter)
- Bellman's equation involves expectations:
- $V_k(x) = \min_u E(c(x,u) + V_{k+1}(f(x,u)))$

What about continuous actions?

Regulator tasks

- Examples: balance a pole, move at a constant velocity
- A reasonable starting point is a Linear Quadratic Regulator (LQR controller)
- Might have nonlinear dynamics $x_{k+1} = f(x_k,u_k)$, but since stay around x_d, can locally linearize $x_{k+1} = Ax_k + Bu_k$
- Might have complex scoring function c(x,u), but can locally approximate with a quadratic model $c \approx x^TQx + u^TRu$
- dlqr() in matlab

LQR Derivation

- Assume V() quadratic: $V_{k+1}(x) = x^TV_{xx:k+1}x$
- $C(x,u) = x^TQx + u^TRu + (Ax+Bu)^TV_{xx:k+1}(Ax+Bu)$
- Want $\partial C/\partial u = 0$
- $B^TV_{xx:k+1}Ax = (B^TV_{xx:k+1}B + R)u$
- u = Kx (linear controller)
- $K = -(B^TV_{xx:k+1}B + R)^{-1}B^TV_{xx:k+1}A$
- $V_{xx:k} = A^TV_{xx:k+1}A + Q + A^TV_{xx:k+1}BK$

More general LQR equations

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

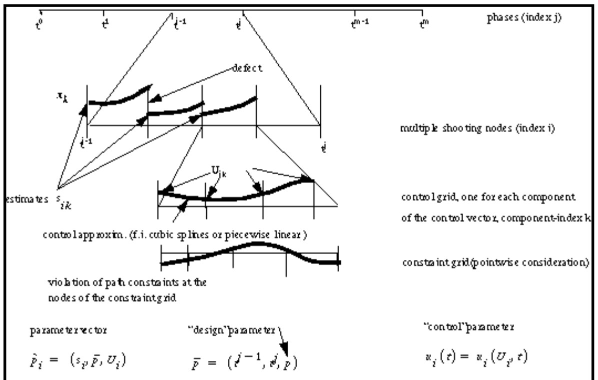
$$L(\mathbf{x},\mathbf{u}) \approx \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \frac{1}{2}\mathbf{u}^T\mathbf{R}\mathbf{u} + \mathbf{x}^T\mathbf{S}\mathbf{u} + \mathbf{t}^T\mathbf{u}$$

$$V(\mathbf{x}) \approx V_0 + V_x\mathbf{x} + \frac{1}{2}\mathbf{x}^TV_{xx}\mathbf{x}$$

$$\mathbf{u}^{opt} = -(\mathbf{R} + \mathbf{B}^TV_{xx}\mathbf{B})^{-1}\times$$
$$(\mathbf{B}^TV_{xx}\mathbf{A}\mathbf{x} + \mathbf{S}^T\mathbf{x} + \mathbf{B}^TV_{xx}\mathbf{c} + V_x\mathbf{B} + \mathbf{t})$$

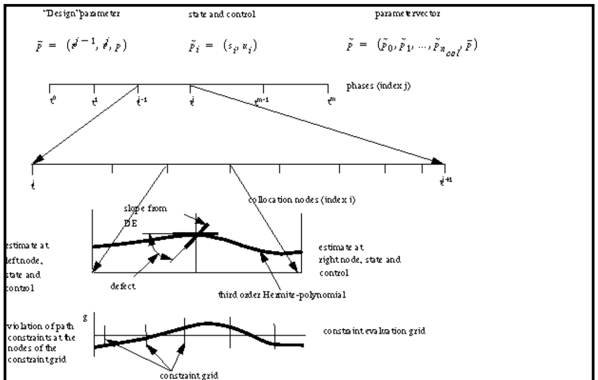
Trajectory Optimization (open loop)

- Calculus of variations
- Multiple shooting
- Function optimization
  - Represent x(t) and u(t) as splines, knot point vector θ
  - Optimize cost(θ) with dynamics $x_{k+1}=f(x_k,u_k)$ a constraint or with dynamic error part of cost.
  - DIRCOL example of current state of the art.



PROMIS is a direct multiple shooting method capable of treating multi phase trajectory optimization problems. The problem is transcribed to a nonlinear programming problem by discretizing state- and control- and path constraint functions on a sequence of discretization grids. The optimization parameters are the states' initial guesses and the control discretization parameters. At phase times, boundary constraints may be enforced. A solution is found when the defects at the multiple shooting nodes and all the (discretized) constraint violations vanish (see picture).

http://www.robotic.dlr.de/control/space/promis.html



TROPIC is a direct collocation method capable of treating multi phase trajectory optimization problems. The problem is transcribed to a nonlinear programming problem by discretizing state- and control- and path constraint functions on a sequence of discretization grids. The optimization parameters are the states' and controls' estimates at the collocation grid points. At phase times, boundary constraints may be enforced. A solution is found when the collocation defects, measured in the center of the collocation intervals, and all the (discretized) constraint violations vanish (see picture).

http://www.robotic.dlr.de/control/space/tropic.html

Trajectory Optimization (closed loop)

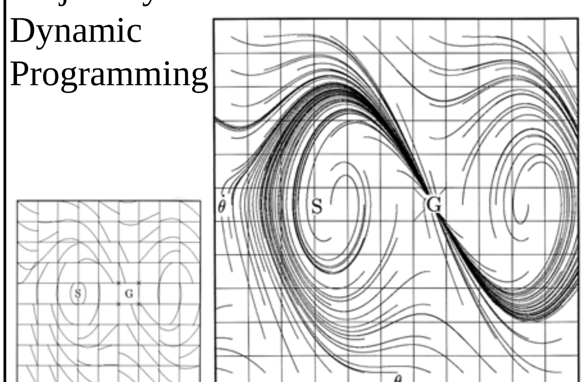
- Differential Dynamic Programming (local approach to DP).

Propagate Value Function V() Along Trajectories

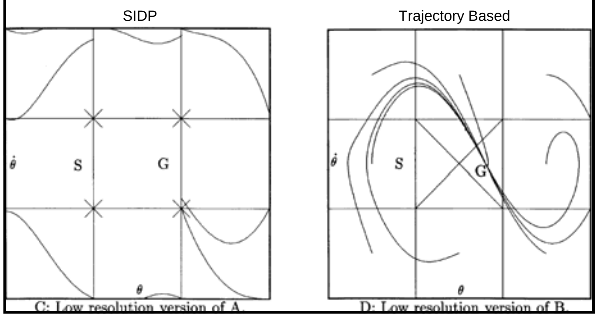
$$V(\mathbf{x}) \approx V_0 + V_x\mathbf{x} + \frac{1}{2}\mathbf{x}^T V_{xx}\mathbf{x}$$

$$\begin{aligned}
Z_x &= V_x\mathbf{A} + \mathbf{Q}(\mathbf{x}-\mathbf{x}_d) \\
Z_u &= V_x\mathbf{B} + \mathbf{R}(\mathbf{u}-\mathbf{u}_d) \\
Z_{xx} &= \mathbf{A}^T V_{xx}\mathbf{A} + \mathbf{Q} \\
Z_{ux} &= \mathbf{B}^T V_{xx}\mathbf{A} + \mathbf{S} \\
Z_{uu} &= \mathbf{B}^T V_{xx}\mathbf{B} + \mathbf{R} \\
\mathbf{K} &= Z_{uu}^{-1} Z_{ux} \\
V_{x_{k-1}} &= Z_x - Z_u\mathbf{K} \\
V_{xx_{k-1}} &= Z_{xx} - Z_{xu}\mathbf{K}
\end{aligned}$$

Trajectory-Based Dynamic Programming

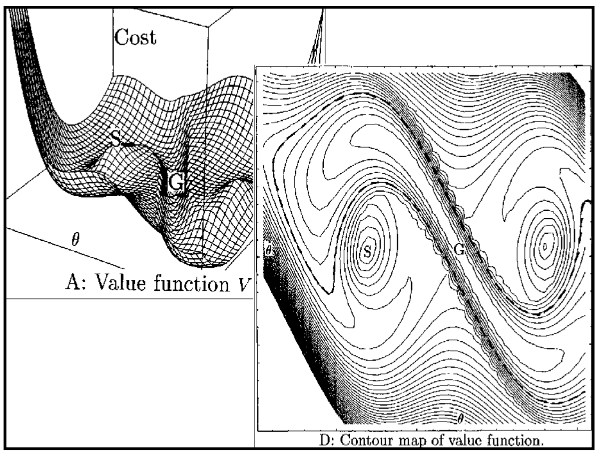


Full Trajectories Helps Reduce Resolution Needed

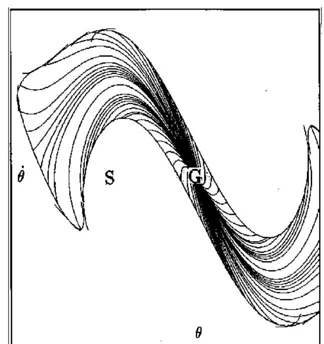


C: Low resolution version of A
D: Low resolution version of B

An Adaptive Grid Approach



A: Value function V
D: Contour map of value function.

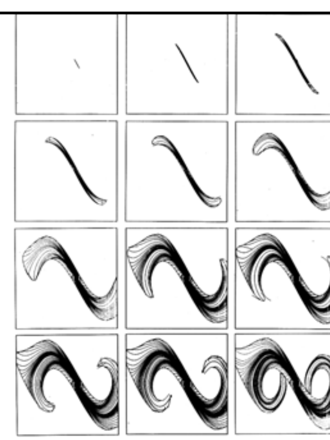


E: Adaptive grid representation

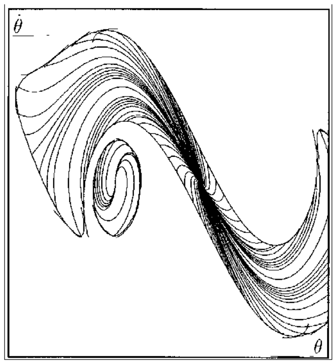
Global Planning
Propagate Value Function Across Trajectories in Adaptive Grid

$$V_{0_1} \approx V_{0_2} + V_{x_2}(\mathbf{x}_1 - \mathbf{x}_2) + \frac{1}{2}(\mathbf{x}_1 - \mathbf{x}_2)^T V_{xx_2}(\mathbf{x}_1 - \mathbf{x}_2)$$

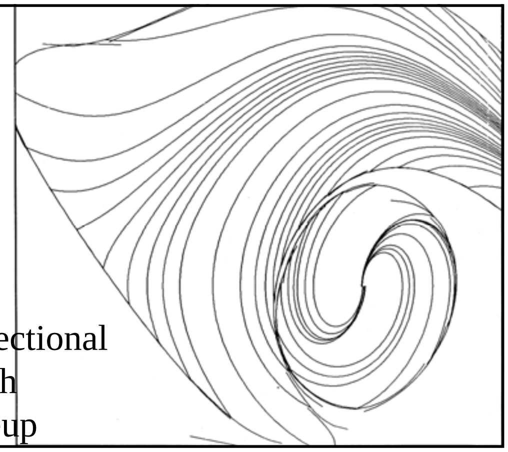
Growing the Explored Region: Adaptive Grids

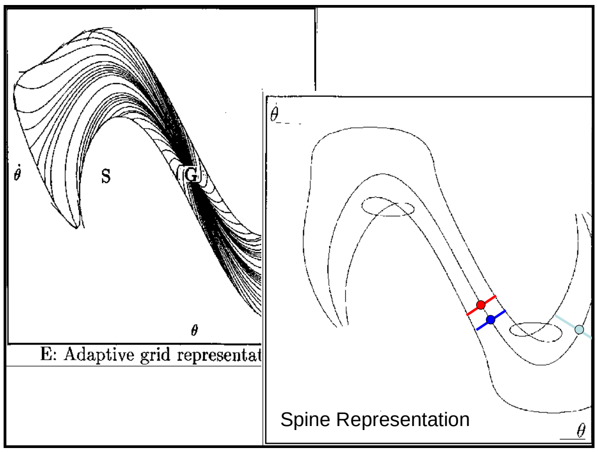


Bidirectional Search

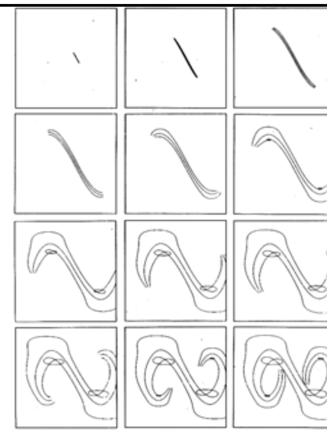


Bidirectional Search Closeup

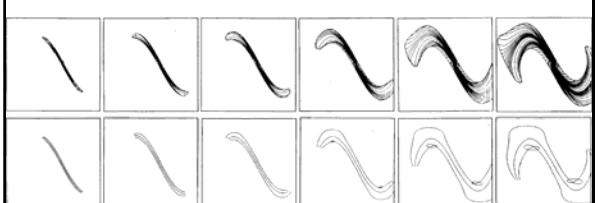




Growing the Explored Region: Spine Representation



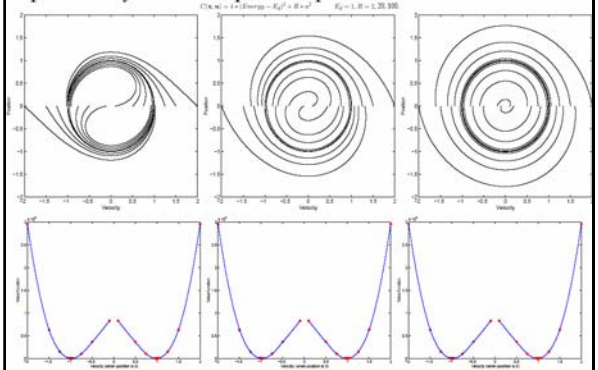
Comparison



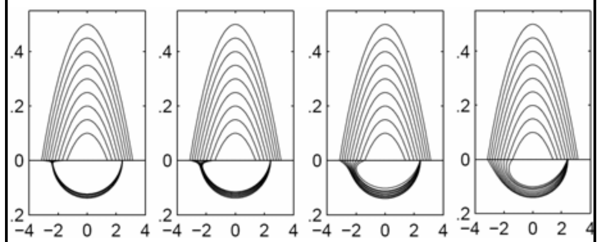
What Changes When Task Periodic?

- Discount factor means V() might increase along trajectory. V() cannot always decrease in periodic tasks.

A periodic system example: Simple harmonic oscillator



Robot Hopper Example



Policy Search

- Parameterized policy $u = \pi(x,\theta)$, θ is vector of adjustable parameters.
- Simplest approach: Run it for a while, and measure total cost.
- Use favorite function optimization approach to search for best θ.
- There are tricks to improve policy comparison, such as using the same perturbations in different trials, and terminating trial early if really bad (racing algorithms).

Policy Search For Structured Policies: Gradient Descent

$$J(\boldsymbol{\theta}) = \int_{\mathbf{x}_0} p(\mathbf{x}_0) V^{\pi}(\mathbf{x}_0, \boldsymbol{\theta}) \mathrm{d}\mathbf{x}_0 \approx \sum_{\mathbf{x}_0} \mathrm{p}(\mathbf{x}_0) V^{\pi}(\mathbf{x}_0, \boldsymbol{\theta})$$

$$\nabla J(\boldsymbol{\theta}) \approx \sum_{\mathbf{x}_0} \mathrm{p}(\mathbf{x}_0) \frac{\partial V^{\pi}(\mathbf{x}_0, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

Computing the derivatives of V()

$$V^{\pi}(\mathbf{x}_k, \boldsymbol{\theta}) = r(\mathbf{x}_k, \boldsymbol{\pi}(\mathbf{x}_k, \boldsymbol{\theta})) + \lambda V^{\pi}(\mathbf{x}_{k+1}, \boldsymbol{\theta})$$

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \boldsymbol{\pi}(\mathbf{x}_k, \boldsymbol{\theta}))$$

$$V = r + \lambda V^{k+1}$$

$$V_{\theta} = r_{\mathbf{u}} \boldsymbol{\pi}_{\theta} + \lambda (V_{\mathbf{x}}^{k+1} \mathbf{f}_{\mathbf{u}} \boldsymbol{\pi}_{\theta} + V_{\theta}^{k+1})$$

$$V_{\mathbf{x}} = r_{\mathbf{x}} + r_{\mathbf{u}} \boldsymbol{\pi}_{\mathbf{x}} + \lambda (V_{\mathbf{x}}^{k+1} \mathbf{f}_{\mathbf{x}} + V_{\mathbf{x}}^{k+1} \mathbf{f}_{\mathbf{u}} \boldsymbol{\pi}_{\mathbf{x}})$$

Policy Search: Stochastic Case

$$J(\boldsymbol{\theta}) \approx \mathrm{E}\left(\sum_{\mathbf{x}_0} \mathrm{p}(\mathbf{x}_0) V^{\pi}(\mathbf{x}_0, \boldsymbol{\theta}) \right) = \sum_{\mathbf{x}_0} \mathrm{p}(\mathbf{x}_0) \mathrm{E}\left(V^{\pi}(\mathbf{x}_0, \boldsymbol{\theta}) \right)$$

$$\begin{aligned} \mathrm{E}(V^{\pi}(\mathbf{x}_0, \boldsymbol{\theta})) \approx & \sum_{k=0}^{\mathrm{T}} \lambda^k (r(\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k) \\ & + \mathrm{Trace}(\Sigma(k)(r_{\mathbf{xx}} + r_{\mathbf{xu}} \boldsymbol{\pi}_{\mathbf{x}} + \boldsymbol{\pi}_{\mathbf{x}}^{\mathrm{T}} r_{\mathbf{ux}} + \boldsymbol{\pi}_{\mathbf{x}}^{\mathrm{T}} r_{\mathbf{uu}} \boldsymbol{\pi}_{\mathbf{x}})|_{\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k, \boldsymbol{\theta}})) \\ \approx & J_d + \sum_{k=0}^{\mathrm{T}} \lambda^k \mathrm{Trace}(\Sigma(k) \mathbf{R}_{\mathbf{xx}}(k)) \end{aligned}$$

Partially Observable Markov Decision Processes (POMDPs)

- Plan using belief state (too expensive?)
- Certainty equivalent approaches: use maximum likelihood estimate of state.
- Policy search
- Dual control problem: want to control, but also want to perturb to reduce uncertainty.

Planning For Dynamic Tasks

- The computational cost of planning is the big challenge for model-based RL.
- Local planning is fast, but only locally optimal.
- Global planning is expensive, but globally optimal.
- Can we combine local and global planning to get fast planning with good plans?

How to do marble maze task: Solving one maze

- Path plan, then LQR servo: A*, RRT, PRM
- Potential field in configuration space.
- Potential field in state space.
- A*/DP in discretized state space.
- Continuous state/action DP
- Policy search

But what can you learn that generalizes across mazes?

Planning and Learning

- Learn better model, and replan.
- Plan faster
  - Initialize value function or policy
  - Find best meta-parameters
  - Find best planning method
- Make better plans
  - Find better optima
  - More robust plans (plan for modeling error)