LESSONS FROM ALPHAZERO FOR OPTIMAL, MODEL PREDICTIVE, AND ADAPTIVE CONTROL

Dimitri P. Bertsekas
Arizona State University

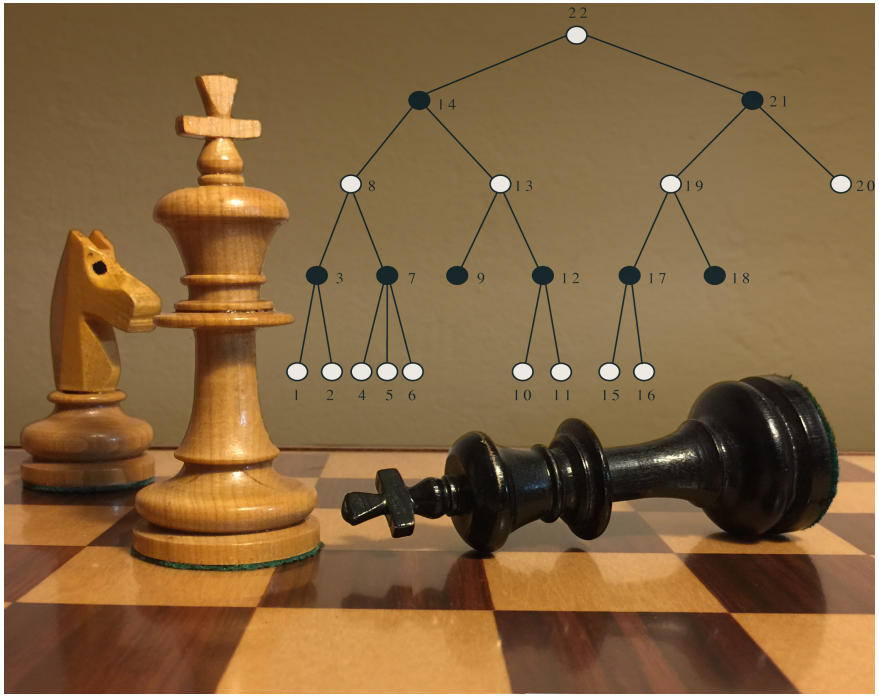
Lecture at KTH on Nov. 17, 2021

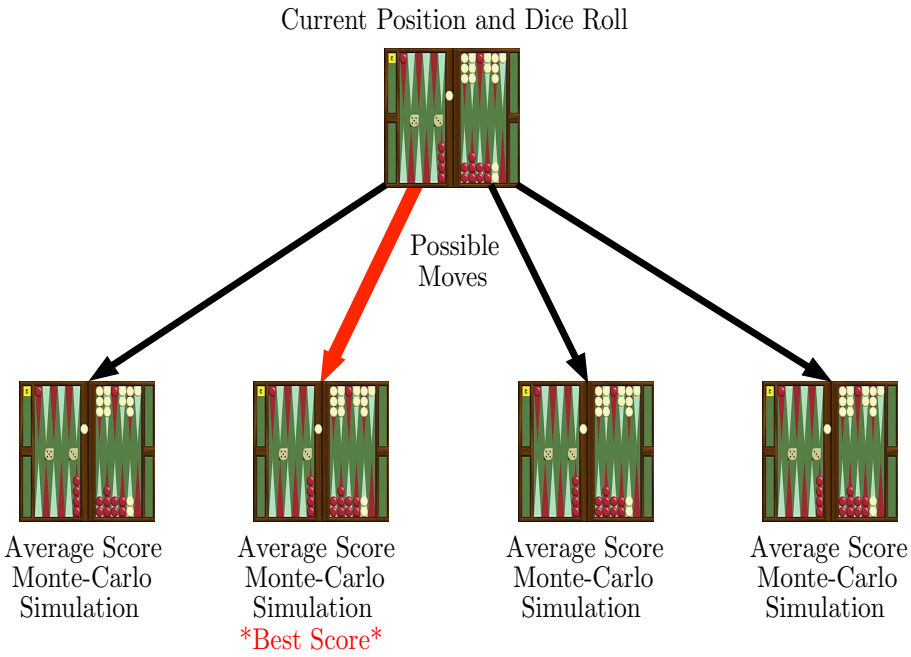
Summary of my forthcoming book, Summer 2022

Draft of the book, related videolectures, and a copy of these slides can be obtained from my website
http://web.mit.edu/dimitrib/www/RLbook.html

Based on analysis from my books
Rollout, Policy Iteration, and Distributed Reinforcement Learning, 2020
Abstract Dynamic Programming, 2nd Edition, 2018
Reinforcement Learning and Optimal Control 2019

Chess and Backgammon - Off-Line Training and On-Line Play





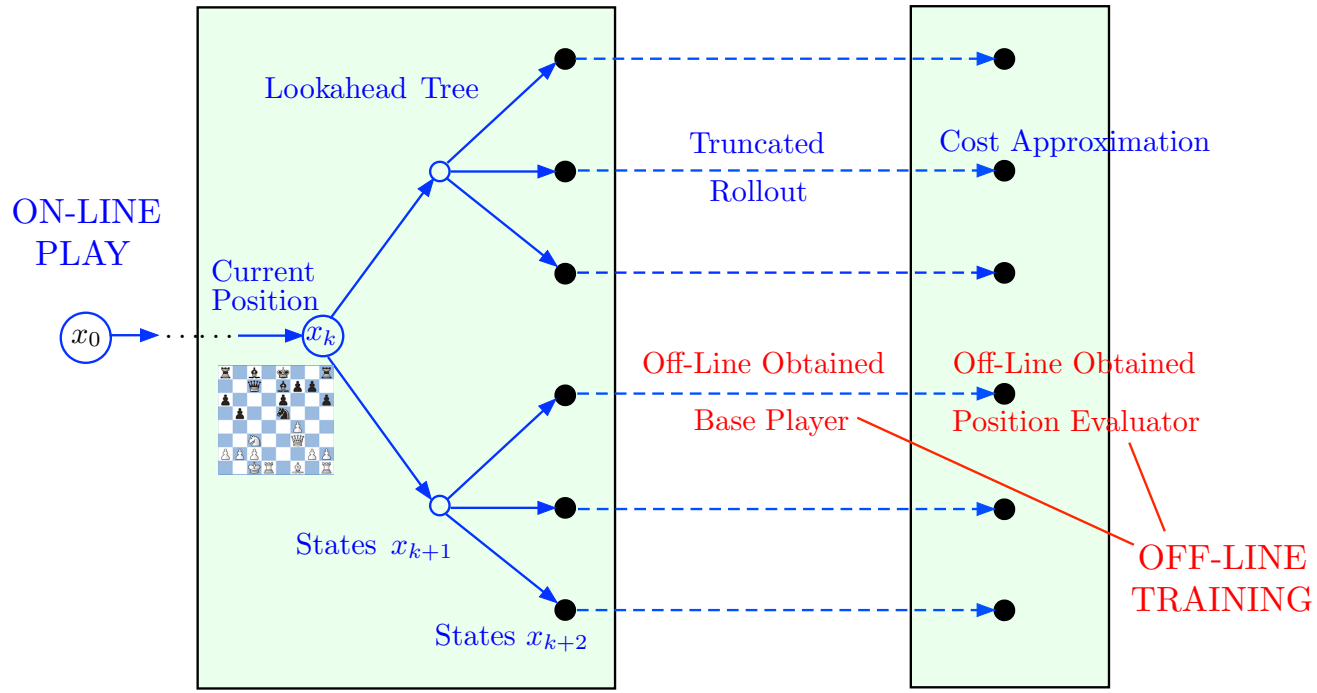
Both AlphaZero (2017) and TD-Gammon (1996) involve two algorithms:

- Off-line training of value and/or policy neural network approximations
- On-line play by multistep lookahead, rollout, and cost function approximation

Strong connections to DP, policy iteration, and RL-type methodology

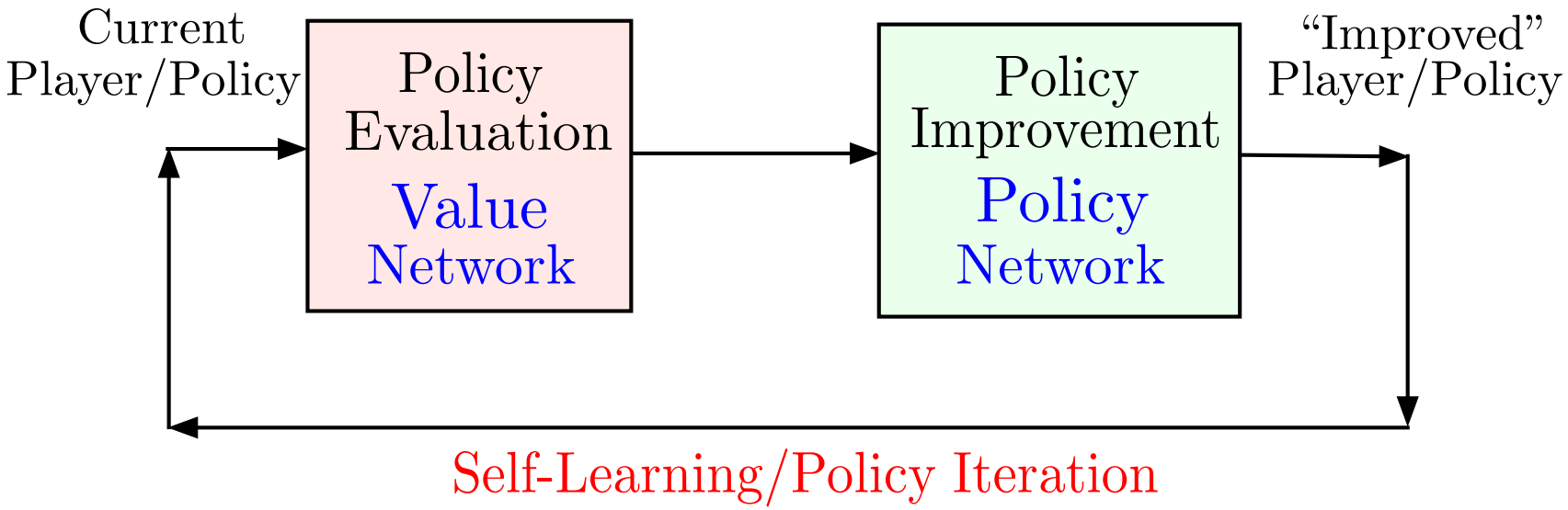
- We are aiming to understand this methodology, so it applies far more generally
- We focus on connections with control system design (MPC and adaptive control), but there are extensions to discrete optimization

On-Line Play in AlphaZero/AlphaGo/TD-Gammon: Approximation in Value Space (Also Called “On-Line Tree Search")



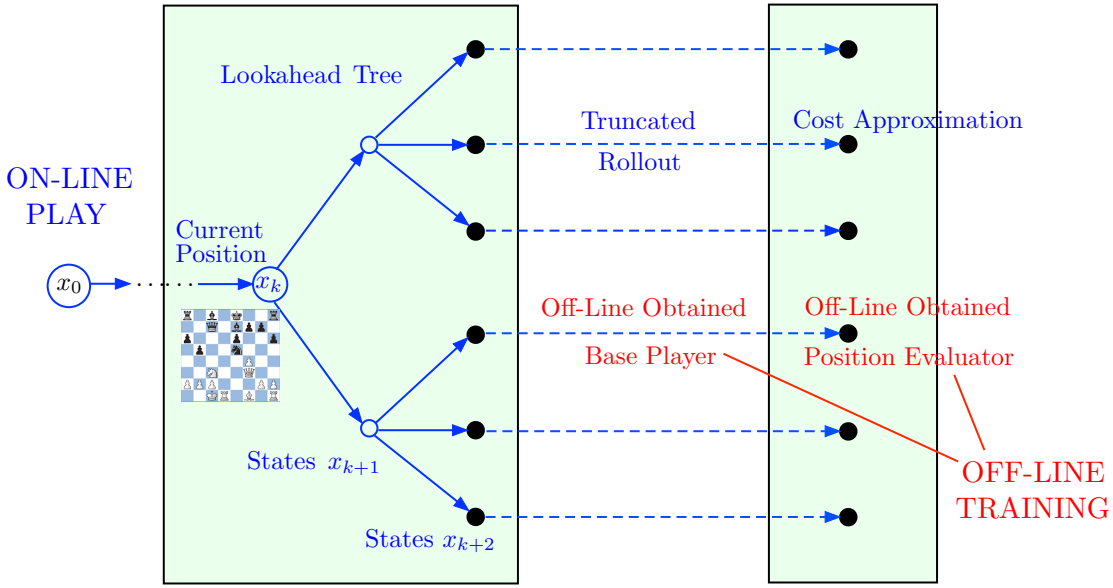
- On-line play uses the results of off-line training, which are: A position evaluator and a base player
- It aims to improve the base player by:
  - Searching forward for several moves through the lookahead tree
  - Simulating the base player for some more moves at the tree leaves
  - Approximating the effect of future moves by using the terminal position evaluation
  - Calculating the “values" of the available moves at the root and playing the best move
- Similarities with Model Predictive Control (MPC) architecture

Off-Line Training in AlphaZero: Approximate Policy Iteration (PI) Using Self-Generated Data



- The current player is used to train an improved player, and the process is repeated
- The current player is "evaluated" by playing many games
- Its evaluation function is represented by a value neural net through training
- The current player is "improved" by using a form of approximate multistep lookahead minimization, called Monte-Carlo Tree Search (MCTS)
- The "improved player" is represented by a policy neural net through training
- TD-Gammon uses similar PI algorithm for off-line training of a value network (does not use MCTS and does not use a policy network)

Some Major Empirical Observations



The AlphaZero on-line player plays much better than the off-line-trained player

TD-Gammon plays much better with truncated rollout than without rollout (Tesauro, 1996)

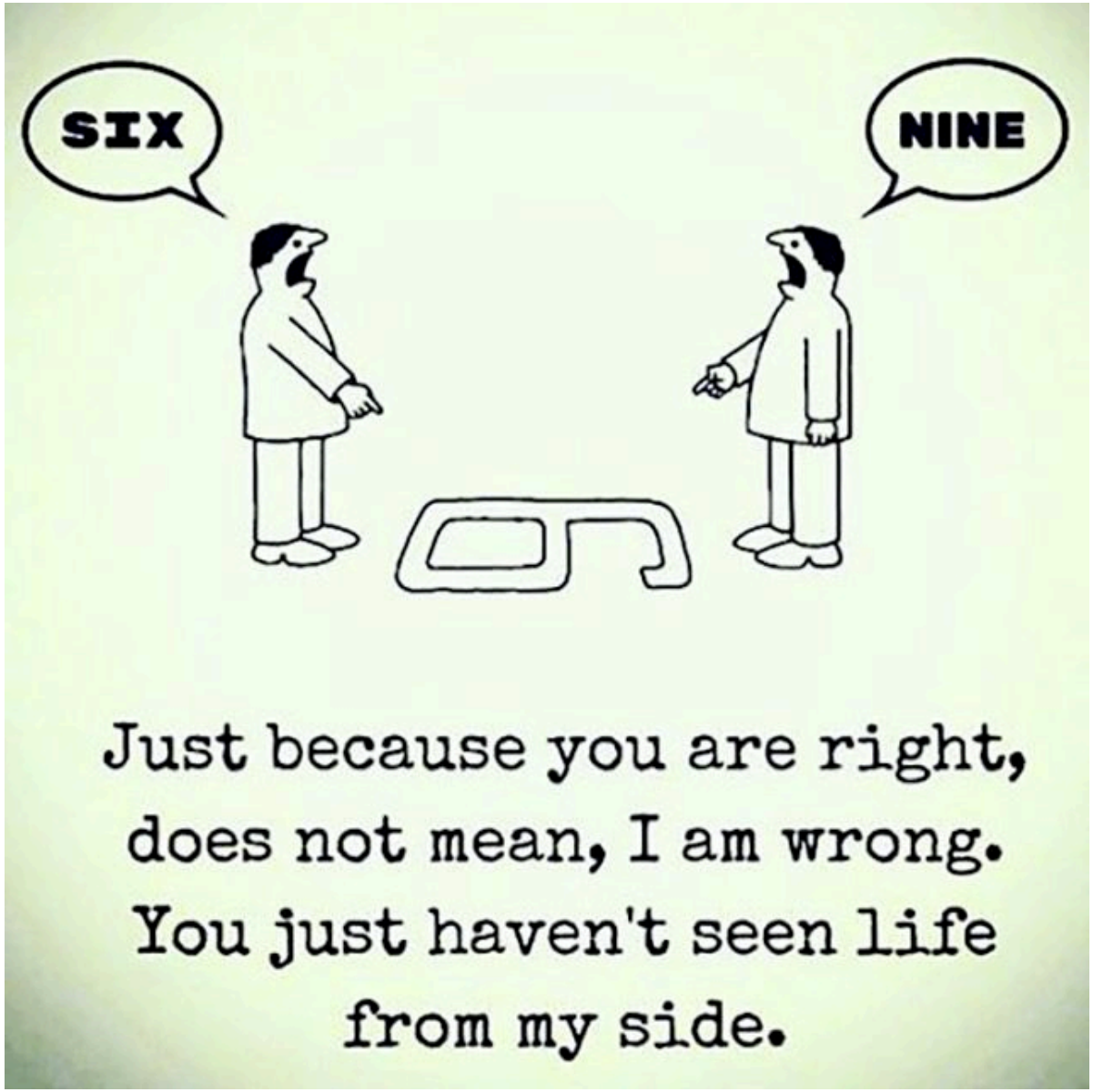
We will aim for explanations, insights, and generalizations through

abstract Bellman operators, visualization, and a focus on the central role of Newton's method

Principal Viewpoints of this Talk

- On-line play is a single step of Newton's method for solving the Bellman equation (or Newton-SOR in case of multistep lookahead and/or truncated rollout)
- Off-line training provides the start point for the Newton step
- On-line play is the real workhorse ... off-line training plays a secondary role. A major reason: On-line play is an exact Newton step. It is not degraded by NN approximations
- Imperfections/differences in off-line training affect the start point, but are washed out by the (superlinear) Newton step
- A cultural difference that we will aim to bridge:
  - Reinforcement Learning/AI research is focused largely on off-line training issues (except in the special case of armed bandit problems)
  - Model Predictive and Adaptive Control research is focused largely on on-line play and stability issues
- Adaptive control with multistep lookahead and rollout is an exact Newton step applied to an on-line estimated Bellman equation ... It's still a Newton step!
- All of this applies in great generality through the power of abstract DP (arbitrary state and control spaces, stochastic, deterministic, hybrid systems, multiagent systems, minimax, finite and infinite horizon, discrete optimization)

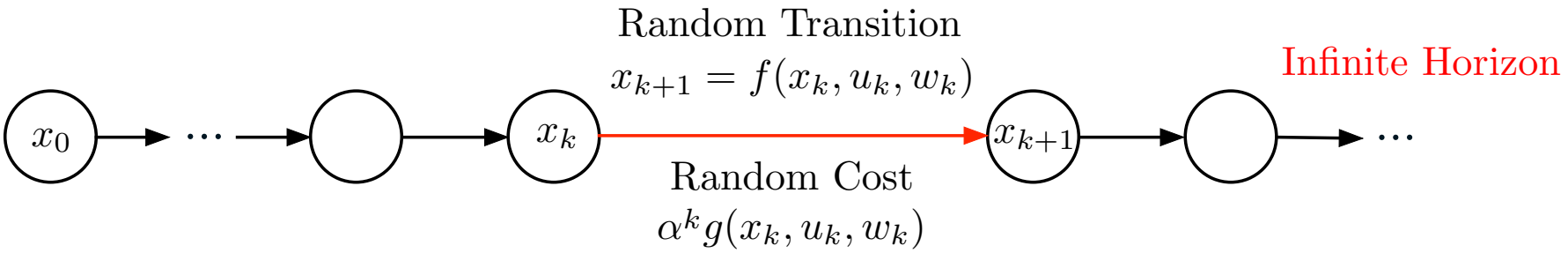
On Viewpoints and Objective Truth



Outline

1. Discounted and undiscounted infinite horizon problems
2. Abstract DP concepts: Bellman operators and Bellman equations
3. Visualization of on-line play as a Newton step
4. Region of stability and its visualization
5. Rollout and policy iteration visualizations
6. Linear quadratic problem visualizations
7. Model predictive control
8. Adaptive control with model estimation (indirect adaptive control)

Infinite Horizon Problems



Infinite number of stages, and stationary system and cost

- System $x_{k+1} = f(x_k, u_k, w_k)$ with state, control, and random disturbance
- Stationary policies $x \mapsto \mu(x)$ satisfying a control constraint $\mu(x) \in U(x)$ for all x
- Cost of stage k: $\alpha^k g\big(x_k, \mu(x_k), w_k\big)$; $0 < \alpha \leq 1$ is the discount factor
- Cost of a policy μ: The limit as $N \to \infty$ of the N-stage costs

$$J_\mu(x_0) = \lim_{N\to\infty} E_{w_k} \left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu(x_k), w_k\big) \right\}$$

- Optimal cost function $J^*(x_0) = \min_\mu J_\mu(x_0)$
- Discounted problems: $\alpha < 1$ and g is bounded (the "nice" case)
- Stochastic shortest path problems: $\alpha = 1$ and special cost-free termination state t
- Control/MPC-type problems: Deterministic, $g \geq 0$, termination state is $t = 0$

Infinite Horizon Problems - Main (Exact DP) Theory

J^* satisfies Bellman's equation:

$$J^*(x) = \min_{u \in U(x)} E_w\Big\{g(x,u,w) + \alpha J^*\big(f(x,u,w)\big)\Big\}, \qquad \text{for all states } x \quad \text{(uniquely ??)}$$

Optimality condition: If $\mu^*(x)$ attain the min in the Bellman equation for every x, the policy μ^* is optimal (??)

Value iteration (VI): Generates cost function sequence $\{J_k\}$

$$J_k(x) = \min_{u \in U(x)} E_w\Big\{g(x,u,w) + \alpha J_{k-1}\big(f(x,u,w)\big)\Big\}, \qquad J_0 \text{ is "arbitrary" (??)}$$

Policy Iteration (PI): Generates sequences of policies $\{\mu^k\}$ and their cost functions $\{J_{\mu^k}\}$; μ^0 is "arbitrary" (??)

The typical iteration starts with a policy μ and generates a new policy $\tilde{\mu}$ in two steps:

- Policy evaluation step, which computes the cost function J_μ
- Policy improvement step, which computes the improved policy $\tilde{\mu}$ using

$$\tilde{\mu}(x) \in \arg\min_{u \in U(x)} E_w\Big\{g(x,u,w) + \alpha J_\mu\big(f(x,u,w)\big)\Big\}$$

On-Line Approximation in Value Space

- Replace J^* with an approximation $\tilde{J}$ in Bellman's equation

First Step "Future"

At x → $\min_{u \in U(x)} E\Big\{ g(x,u,w) + \alpha \tilde{J}\big(f(x,u,w)\big) \Big\}$

One-Step Lookahead

First ℓ Steps "Future"

At x_k → $\min_{u_k, \mu_{k+1}, \ldots, \mu_{k+\ell-1}} E\left\{ g(x_k,u_k,w_k) + \sum_{i=k+1}^{k+\ell-1} \alpha^{i-k} g\big(x_i, \mu_i(x_i), w_i\big) + \alpha^{\ell} \tilde{J}(x_{k+\ell}) \right\}$

Multistep Lookahead

- Defines a lookahead policy $\tilde{\mu}$ with $\tilde{\mu}(x_k)$ the minimizing u_k above

KEY NEW FACT: $J_{\tilde{\mu}}$ is the result of a Newton step to solve Bellman Eq. starting from $\tilde{J}$ (Newton-SOR step for multistep lookahead $\ell > 1$). The error decreases SUPERLINEARLY

$$\frac{J_{\tilde{\mu}} - J^*}{\tilde{J} - J^*} \to 0 \qquad \text{as } \tilde{J} \to J^*$$

An Abstract DP Viewpoint: Bellman Operators and Bellman Equations (Abstract DP Book, 2018, DPB)

$$(T_\mu J)(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big)\Big\}$$

$$(TJ)(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J\big(f(x,u,w)\big)\Big\} = \min_{\mu}(T_\mu J)(x)$$

They define the Bellman equations $J_\mu = T_\mu J_\mu, \quad J^* = TJ^*$

T_μ and T transform real-valued functions J into functions $T_\mu J$ and TJ (assumed real-valued for this talk)

- How many dimensions? Answer: The number of states x
- For each fixed x, $(T_\mu J)(x)$ and $(TJ)(x)$ are functions of J
- Example: For a 2-state system, $(TJ)(1)$ and $(TJ)(2)$ are real-valued functions of the vector $J = \big(J(1), J(2)\big) \in \Re^2$
- In this case T_μ and T map $\Re^2$ to $\Re^2$
- Both T_μ and T are monotone
- T_μ is linear
- T is "concave", i.e., $(TJ)(x)$ is a concave function of J for each fixed x
- For infinite-dimensional state space, T_μ and T are infinite-dimensional operators (map infinite dimensional function space to itself)

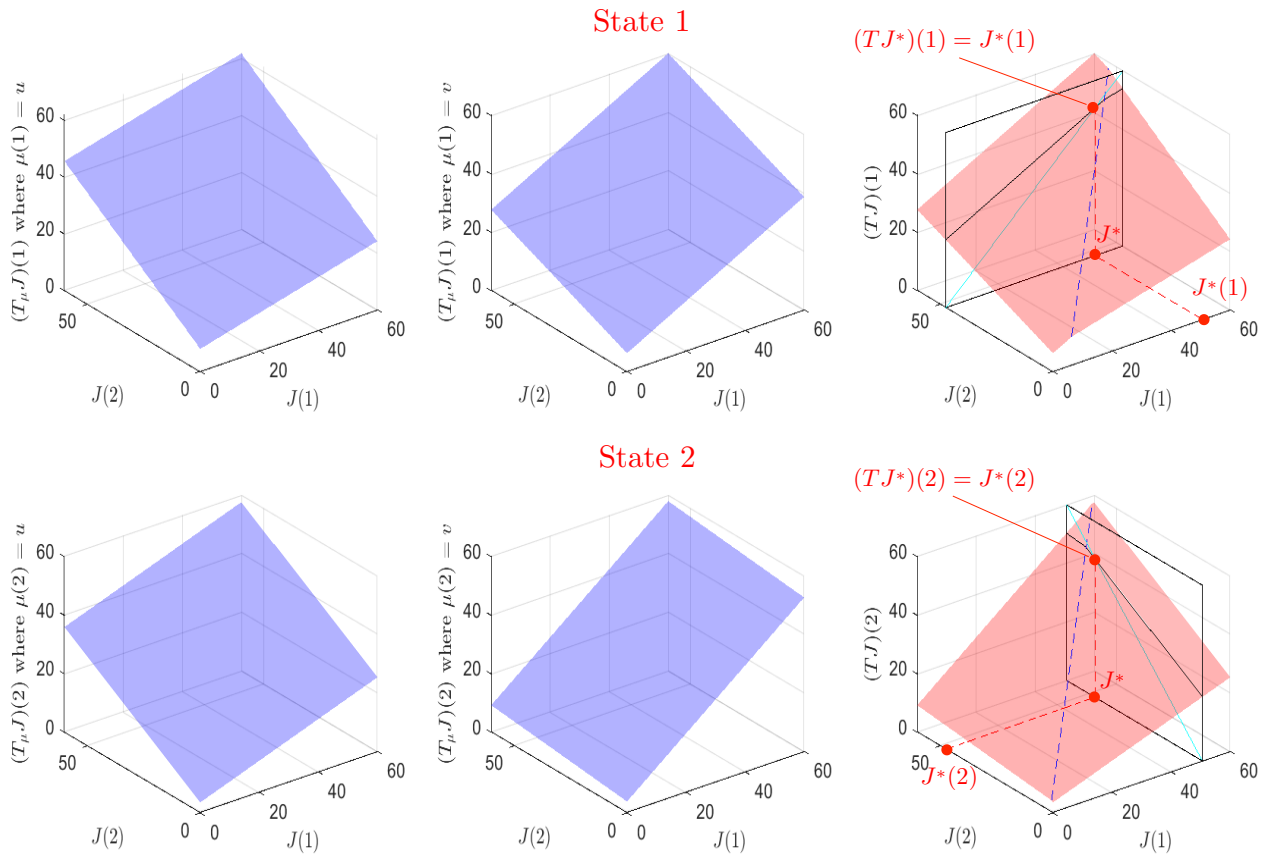
Visualization Using 1-D Slices Through J^*
Two-State and Two-Control Example: A 4-D Graph = Two 3-D Graphs

$$(T_\mu J)(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big)\Big\},\ x = 1,2 \quad \text{(linear monotone)}$$

$$(TJ)(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J\big(f(x,u,w)\big)\Big\},\ x = 1,2 \quad \text{(concave monotone)}$$

They define the Bellman equations $J_\mu = T_\mu J_\mu, \quad J^* = TJ^*$

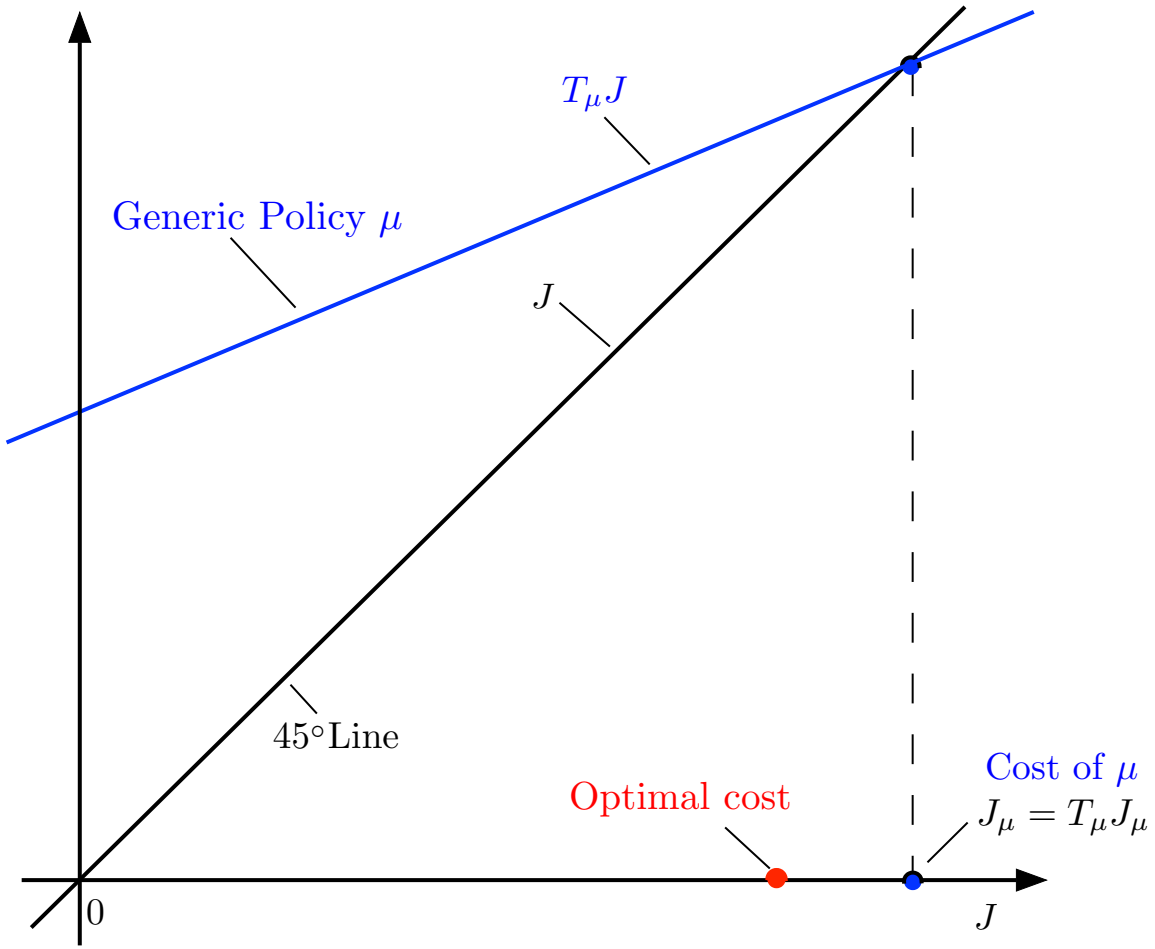
2-State/2-Control Example



μ-Bellman Operator in One Dimension Through J_μ

$$(T_\mu J)(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big)\Big\} \quad \text{(linear monotone)}$$

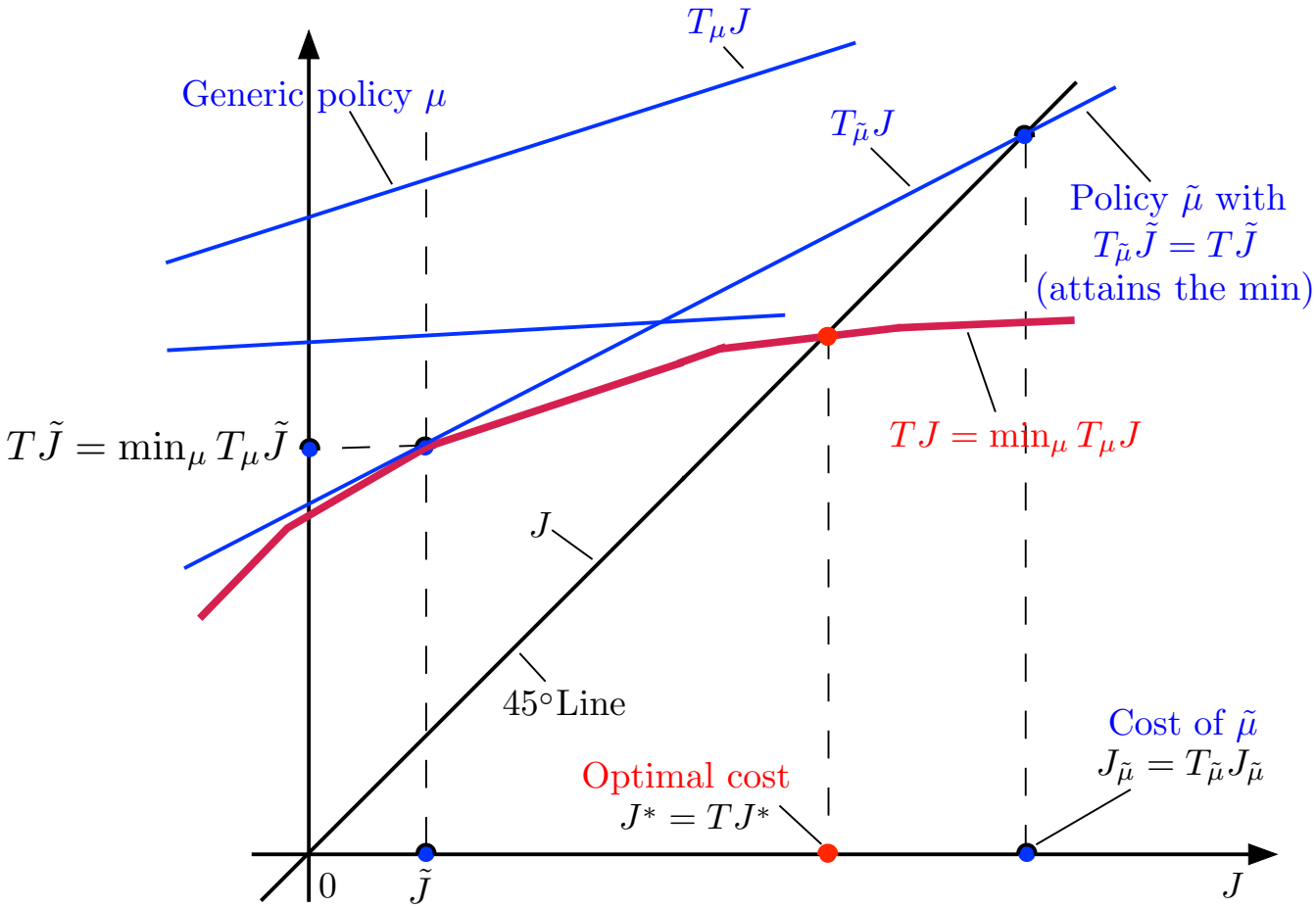
μ-Bellman equation: $J_\mu = T_\mu J_\mu$



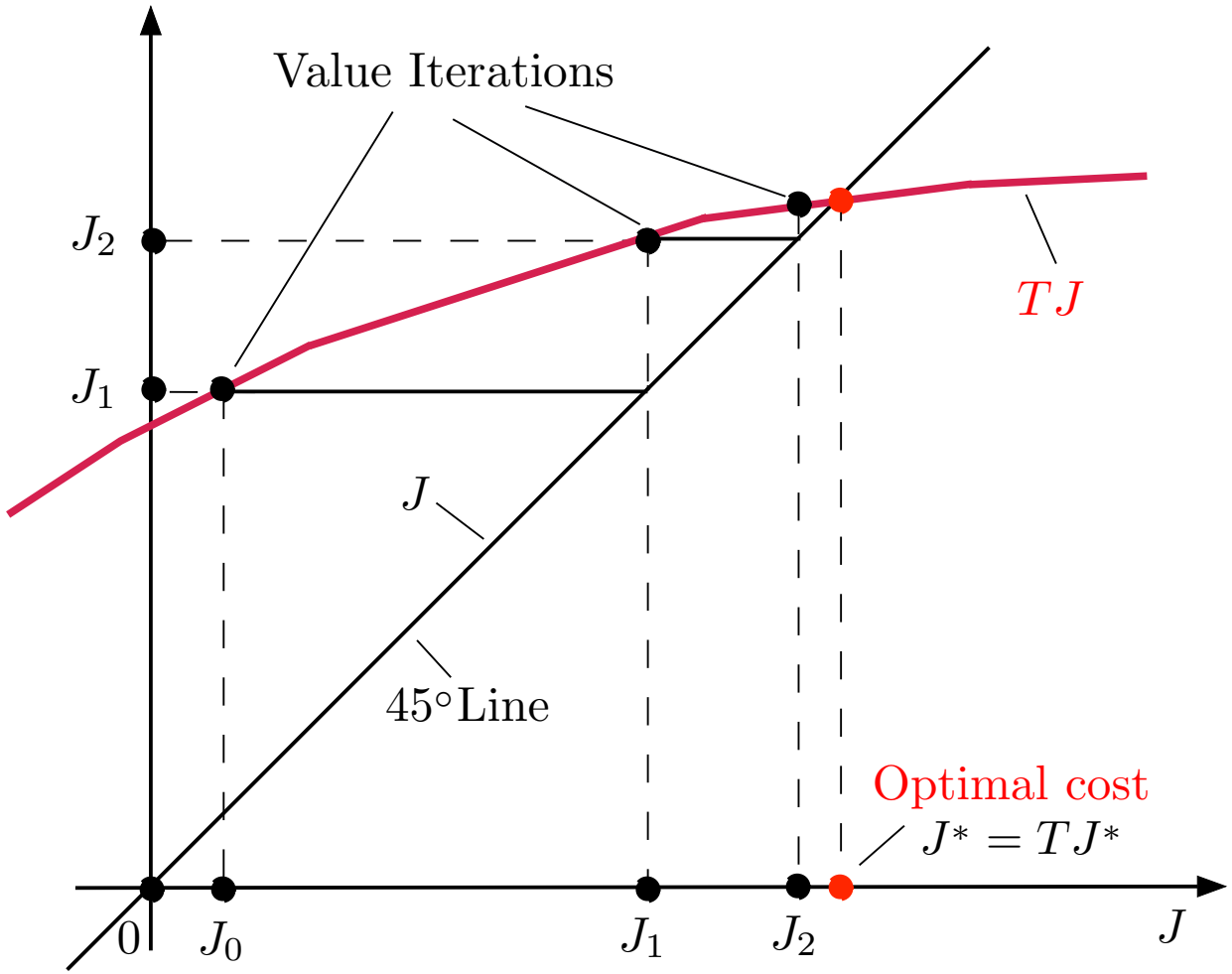
MIn-Bellman Operator in One Dimension Through J^*

$$(TJ)(x) = \min_{u \in U(x)} E_w\Big\{g(x,u,w)+\alpha J\big(f(x,u,w)\big)\Big\} = \min_{\mu}(T_\mu J)(x) \quad \text{(concave monotone)}$$

Min-Bellman equation: $J^* = TJ^*$

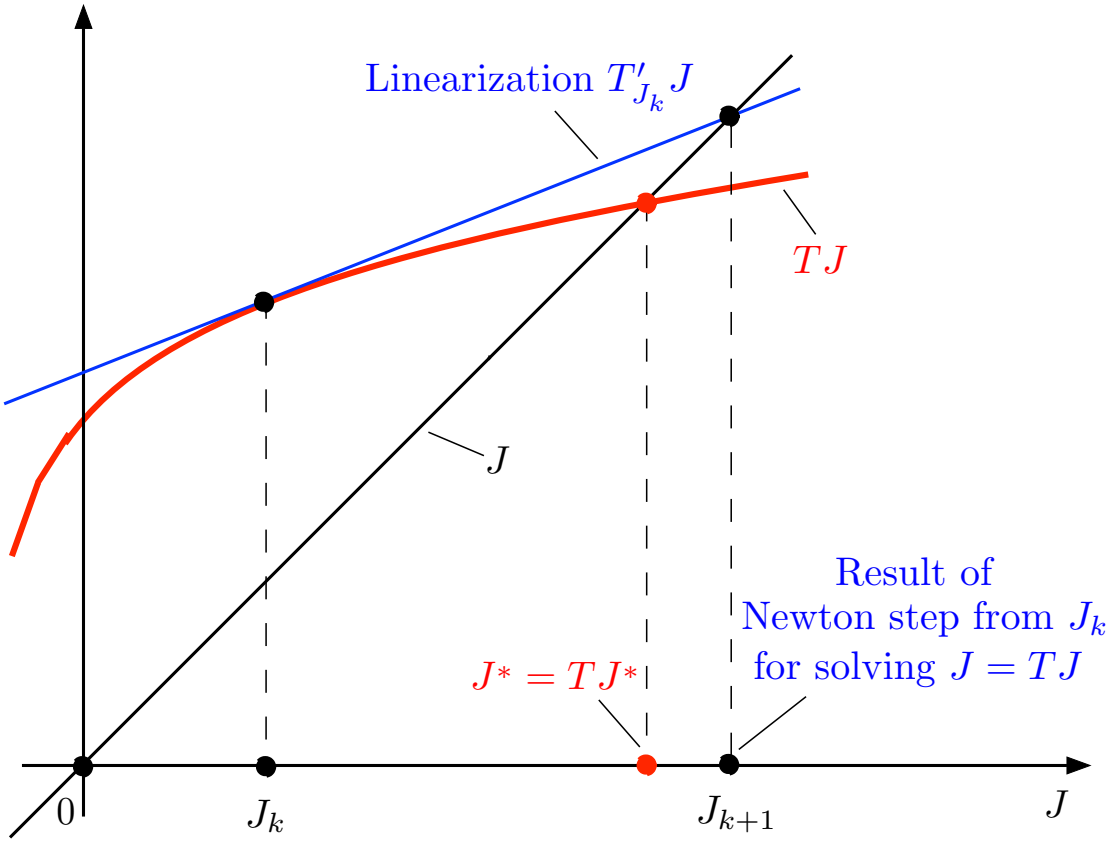


Visualization of Value Iteration: $J_{k+1} = TJ_k$



Convergence $J_k \to J^*$ depends on J_0 and the "slope" of T (e.g., whether T is a contraction)

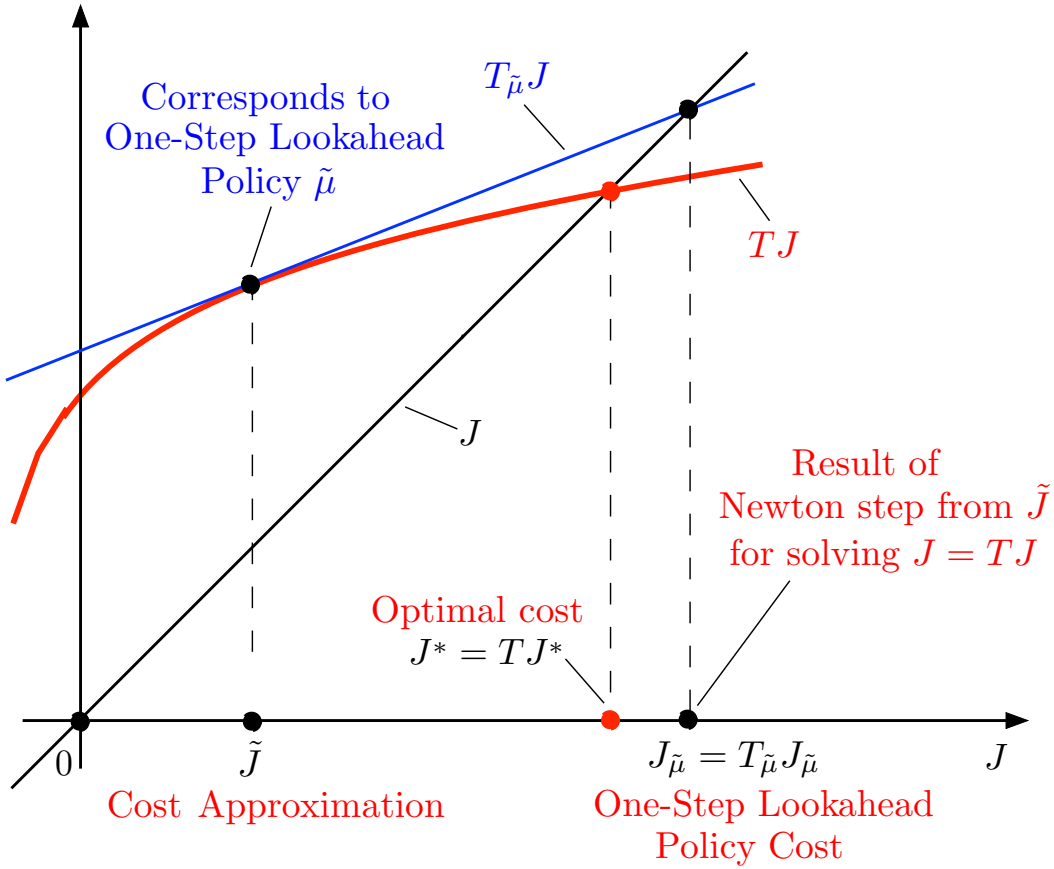
Newton's Method for Solving Fixed Point Equation $J = TJ$



It is an iterative method that generates a sequence $\{J_k\}$. The typical iteration:

- Given J_k
- "Linearize" T at J_k: Replace TJ by the linearization $T'_{J_k} J$
- Solve the linearized fixed point problem $J = T'_{J_k} J$
- The solution of the linearized fixed point problem is the next iterate J_{k+1}

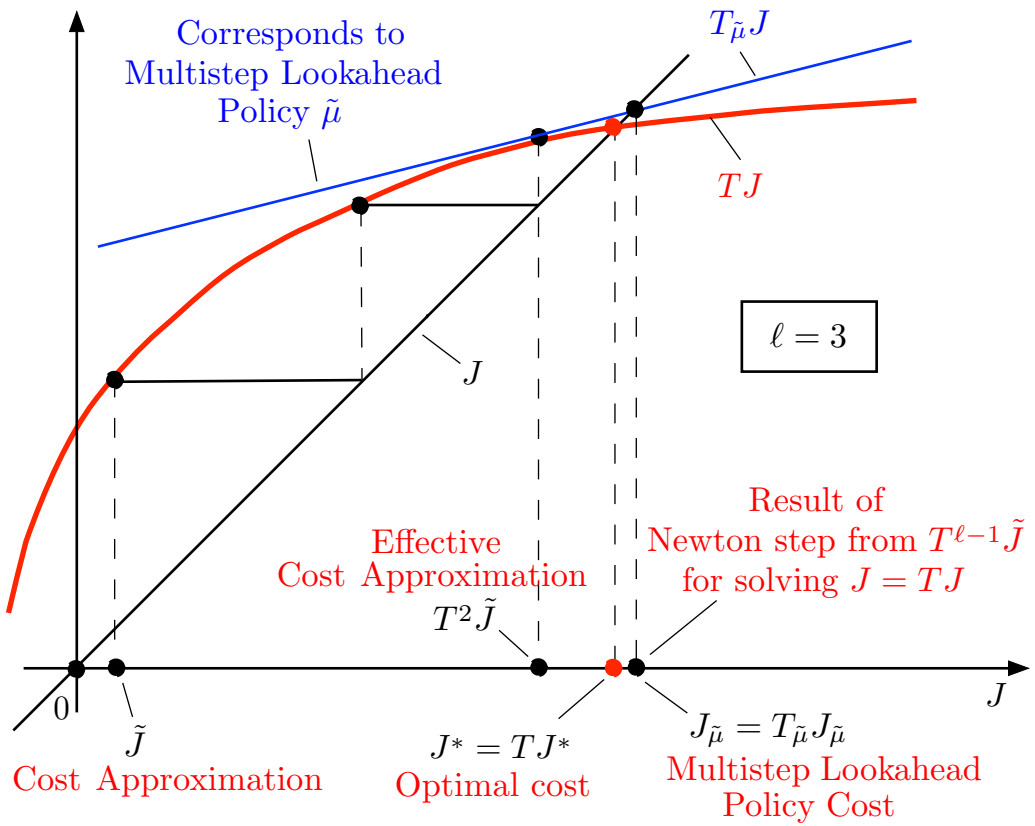
Linearization/Newton Step by One-Step Lookahead



$$(T\tilde{J})(x) = \min_{\mu}(T_{\mu}\tilde{J})(x) = (T_{\tilde{\mu}}\tilde{J})(x) \quad \text{(linearization of } T \text{ at } \tilde{J} \text{ yields } \tilde{\mu})$$

- This is a key new insight. Do we need differentiability of T?
- No! The Newton step can work without differentiability because T is concave and monotone; (assumptions needed, everything is OK for “contractive" problems)
- The Newton step smooths out starting point variations (lots of empirical evidence)

Linearization/Newton-SOR Step by Multistep Lookahead

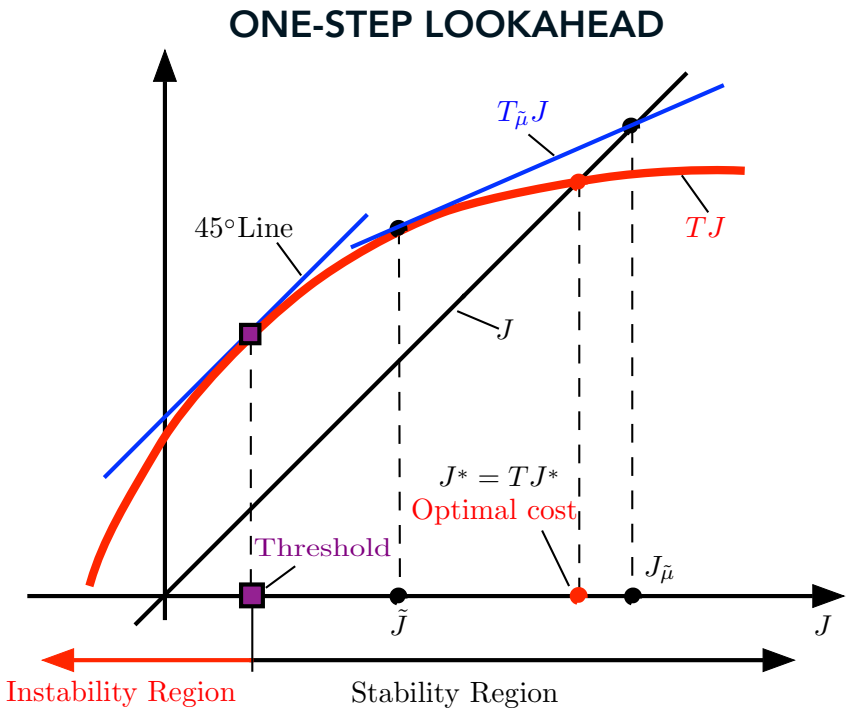


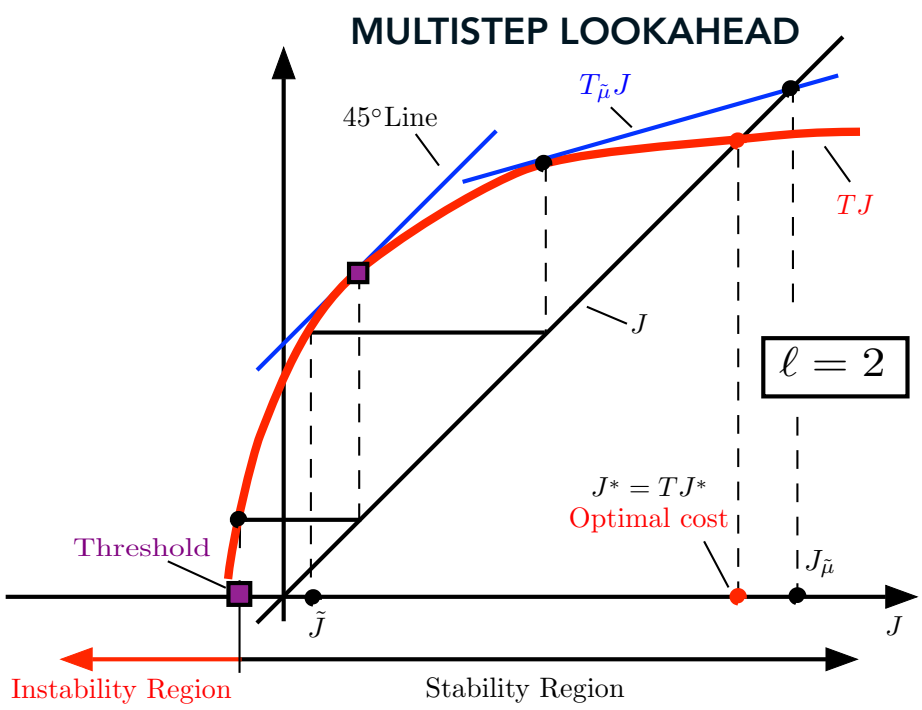
$$(T^\ell \tilde{J})(x) = \min_\mu (T_\mu T^{\ell-1}\tilde{J})(x) = (T_{\tilde{\mu}} T^{\ell-1}\tilde{J})(x) \quad \text{(linearization of } T \text{ at } T^{\ell-1}\tilde{J} \text{ yields } \tilde{\mu})$$

Solution of the linearized equation $J = T_{\tilde{\mu}}J$ yields the cost function $J_{\tilde{\mu}}$ of $\tilde{\mu}$

Newton-SOR converges faster than pure Newton, but is more time-consuming

Stability in the MPC Context (Deterministic Problem, Positive Costs, Cost-Free Terminal State)





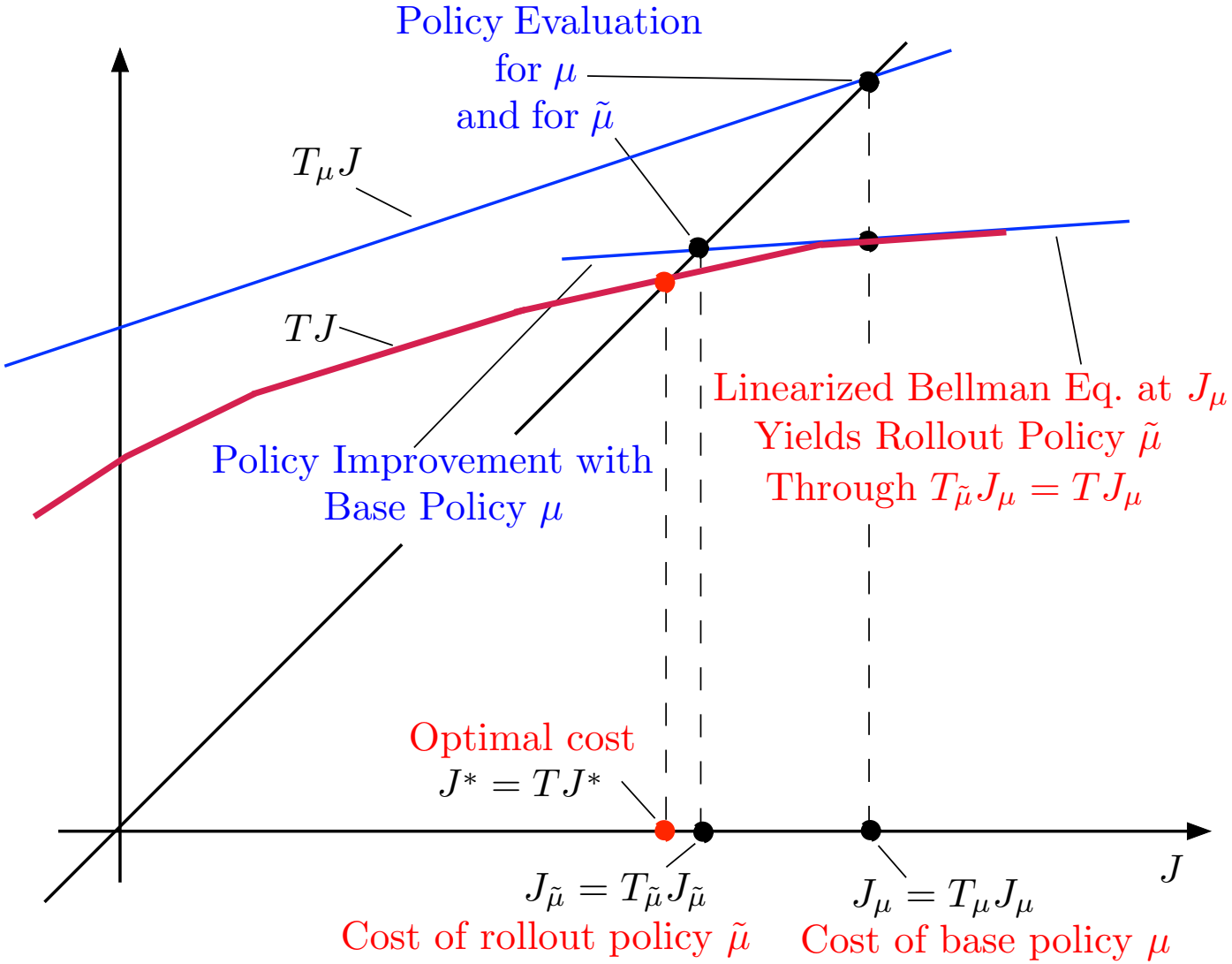
A policy μ is called stable if $J_\mu(x) < \infty$ for all x (a very general definition)

True if T_μ has "slope" < 1 (i.e., T_μ is a contraction)

Region of stability: The set of $\tilde{J}$ for which the lookahead policy $\tilde{\mu}$ is stable

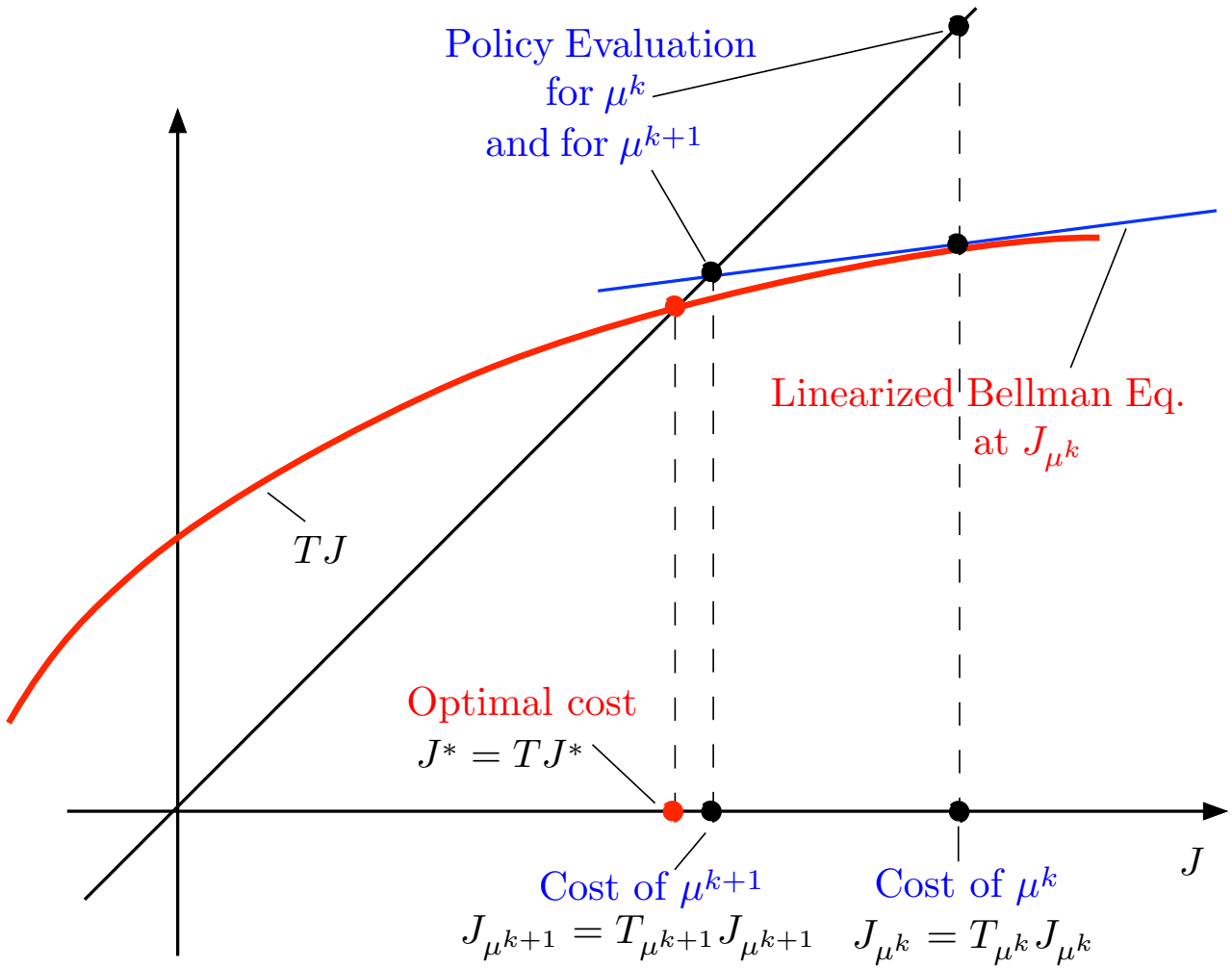
- Depends on the length of lookahead - threshold shifts to the left as ℓ increases
- It makes sense to try to push $\tilde{J}$ towards some J_μ with μ stable (rollout idea)

Rollout: A Newton Step Starting from $\tilde{J} = J_\mu$, where μ is a Stable Policy



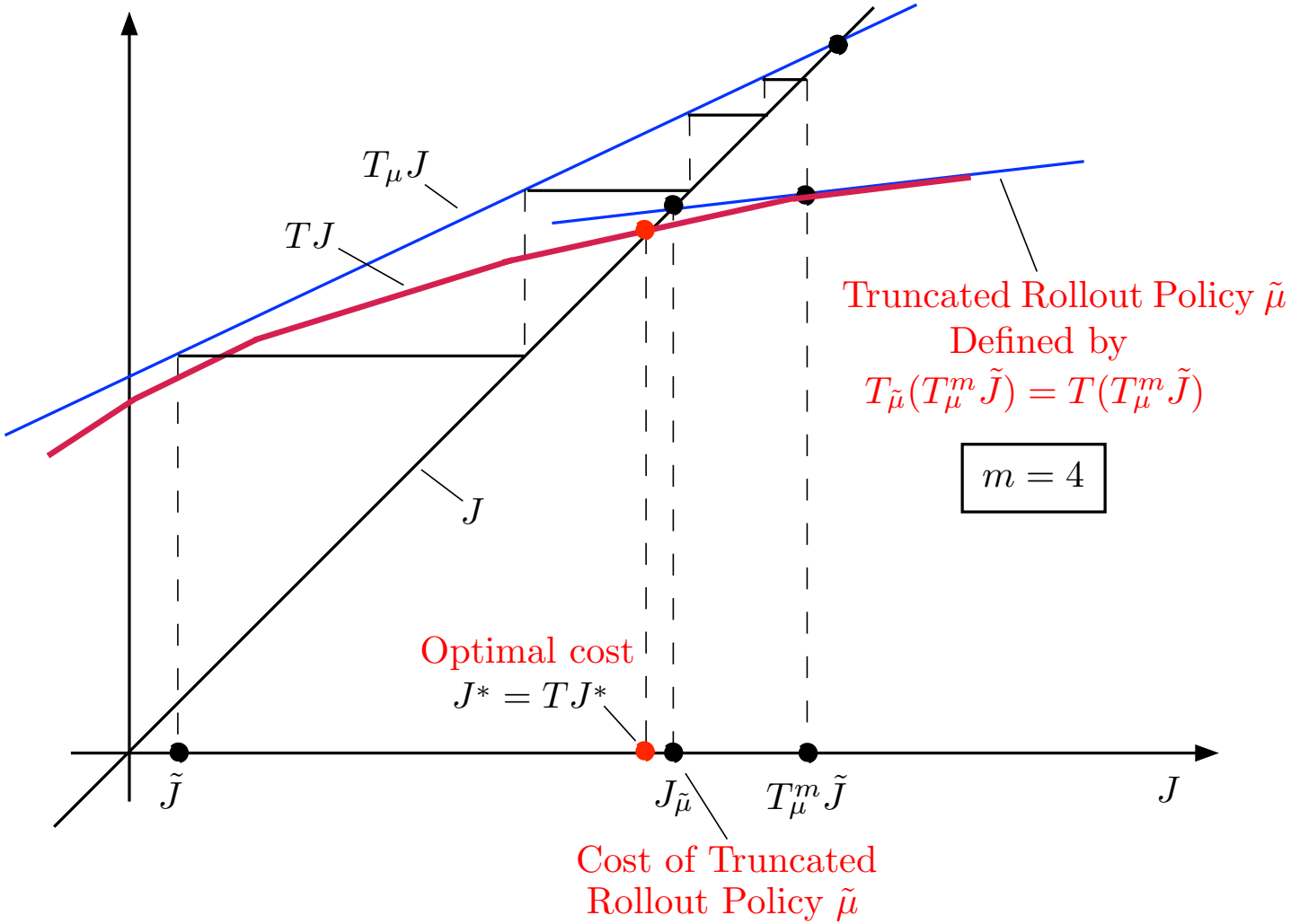
Rollout with a stable policy μ yields a stable policy $\tilde{\mu}$

Policy Iteration (PI) is Repeated Rollout - Starting from a Stable Policy it Produces a Sequence of Stable Policies $\{\mu^k\}$



- Pure form of PI is Newton's method (known for special cases, Kleinman 1968 ++)

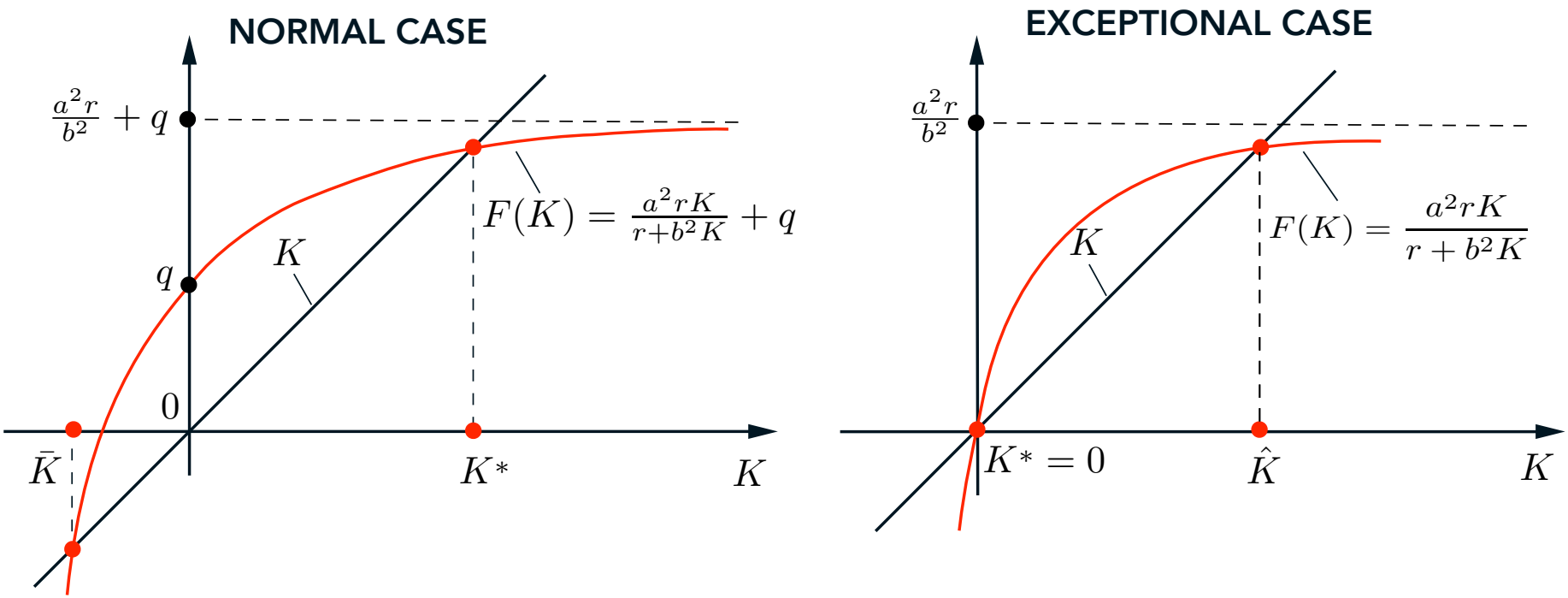
Truncated Rollout with Base Policy μ (Related to Optimistic PI)



Truncated rollout with ℓ-step lookahead is similarly defined: Total lookahead is $\ell + m$

Truncated rollout is an economical substitute for multistep lookahead (e.g., TD-Gammon)

Linear Quadratic Problems: Riccati Instead of Bellman Operators



Riccati operator is the restriction of the Bellman operator to the subspace of quadratics

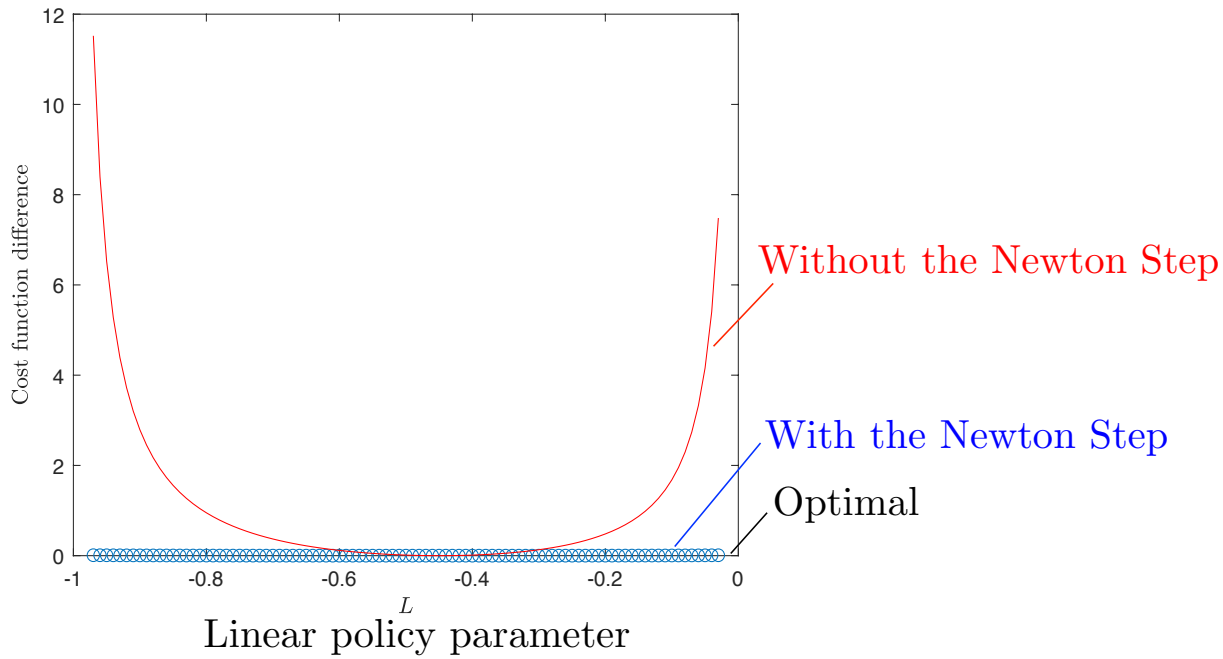
Linear system $x_{k+1} = ax_k + bu_k$. Cost $g(x,u) = qx^2 + ru^2$, $q, r \geq 0$, $\alpha = 1$

- $J^*(x) = K^*x^2$; K^* solves the Riccati Eq. $K = F(K)$
- Normal case: $q > 0$, $r > 0$. Riccati Eq. has K^* as its unique positive solution
- Exceptional case example: $q = 0$, $r > 0$, and unstable system $a > 1$. Riccati Eq. has two nonnegative solutions $K^* = 0$ and $\hat{K} = \frac{r(a^2-1)}{b^2}$

A Common Question: Why Not Just Train a Policy Network and Use it Without On-Line Play?

Pure approx. in policy space (policy gradient, random search, etc) is flawed

It lacks the exact Newton step, which corrects (superlinearly) the errors of off-line training



A one-dimensional linear quadratic example (with known and fixed model)

Consider a parametrized suboptimal linear policy $\mu(x) = Lx$ without one-step lookahead, and its version with one-step lookahead/Newton step

Model Predictive Control (MPC)

Classical form of MPC (1980s+, extensive literature)

- Applies to continuous state and control deterministic problem with positive cost

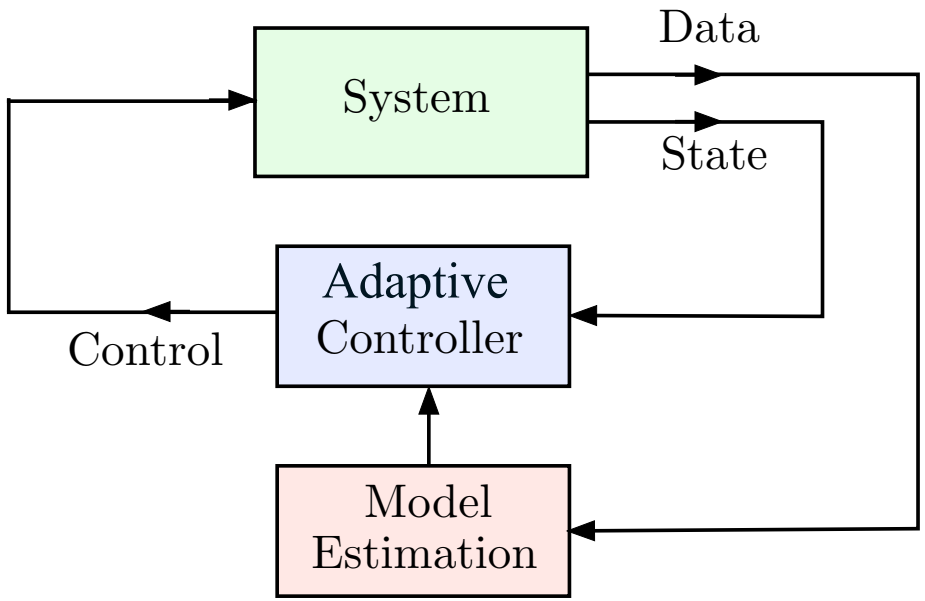
$$x_{k+1} = f(x_k, u_k), \qquad g(x, u) > 0 \ \text{ for all } x \neq 0, \ g(0, u) = 0$$

- MPC is central in control theory, but is culturally different from RL/AI
- MPC's architecture is very similar to AlphaZero ... includes lookahead minimization ("control interval"), rollout ("prediction interval"), and terminal cost
- MPC focuses on continuous spaces, control/stability issues, and places most emphasis on on-line play
- There is some off-line training, like computing off-line terminal cost approximations, base policies, and safe regions/reachable target tubes (to deal with state constraints $x_k \in X$)

Extended forms of MPC (Rawlings+Mayne+Diehl, Borrelli+Bemporad+Morari)

- More complex versions that deal with stochastic uncertainties, hybrid continuous/discrete control space versions, minimax versions, target tubes (to deal with state constraints), etc
- Involve simulation-based rollout with an off-line trained policy. One of these has been called Learning MPC (Rosolia+Borrelli, Li+Johansson+Martensson+DPB)

Adaptive Control - Changing System, On-line System Identification



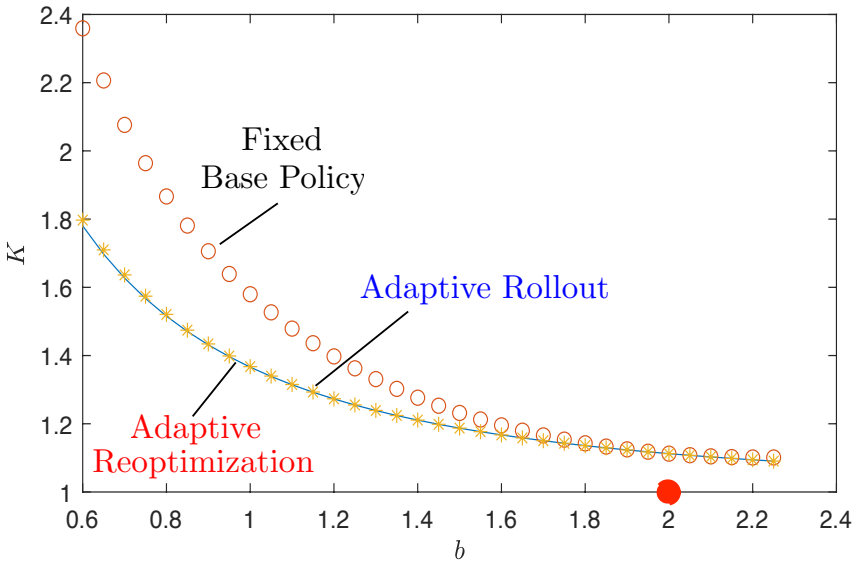
Classical indirect adaptive control (1960s+, extensive book literature)

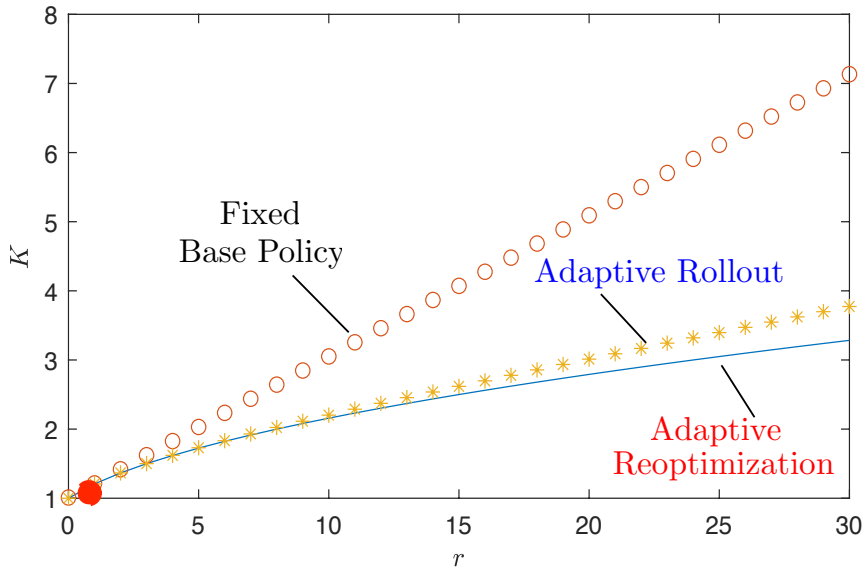
Simply reoptimizes the controller, when the estimated model changes ... but this may be a difficult/time-consuming reoptimization

Faster alternative: Indirect adaptive control by rollout with a (robust) policy

- Use rollout in place of reoptimization - this is simpler (use the current model estimate for lookahead minimization and a nominal/robust base policy for rollout)
- Capitalizes on the fast convergence of the Newton step

Adaptive Control by Rollout: A Linear Quadratic Example





- System: $x_{k+1} = x_k + bu_k$
- Cost $g(x, u) = x^2 + ru^2$
- We use one-step lookahead and rollout with base policy that is optimal for the nominal values $b = 2$, $r = 0.5$
- In the left figure we change the system parameter b
- In the right figure we change the cost parameter r
- Using a “robust" controller as base policy without the Newton step is often flawed
- Using a “robust" controller as base policy with the Newton step corrects the flaw

Concluding Remarks

- There is much to be gained by using on-line play on top of off-line training
- Using just off-line training without on-line play may not work well
  - On-line play uses an exact Newton step (not subject to training errors), and can deal with changing system parameters
- Using just on-line play without off-line training misses out on performance
  - Off-line training can produce good starting points for the Newton step
- The role of Newton's method is central - this is a new insight that can guide both analysis and algorithmic design
- The Newton step is exact ... all the approximation goes into the starting point for the Newton step (which washes out training method differences and errors)
- The cultural divide between RL/AI and control can be bridged by combining off-line training and on-line play
- MPC uses a very similar architecture to AlphaZero; can benefit from RL/AI ideas
- We can approach indirect adaptive control through rollout: Use a Newton step in place of reoptimization
- Generality: Arbitrary state and control spaces, discrete optimization applications, multiagent versions (see the 2020 rollout/distributed RL book)
- There are exceptional behaviors waiting for clarification by analysis

Some Words of Optimism

- The successes of RL and of MPC are solid reasons for optimism
- More success can be expected by combining ideas from both RL/AI and MPC/adaptive control cultures
- On-line long lookahead/rollout can be a computational bottleneck ...
- But massive computational power and distributed computation can mitigate the bottleneck, and allow more sophisticated on-line play strategies
- There is an exciting journey ahead!

Thank you!