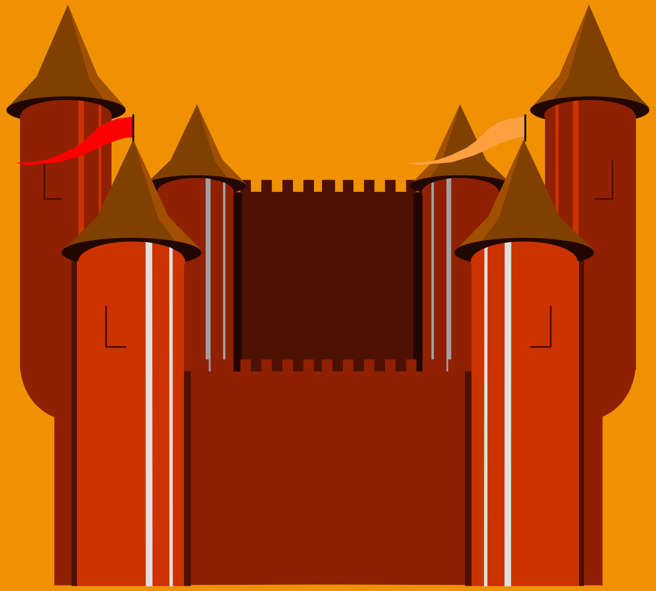
ORF 544

Stochastic Optimization and Learning

Spring, 2019



Warren Powell
Princeton University
http://www.castlelab.princeton.edu

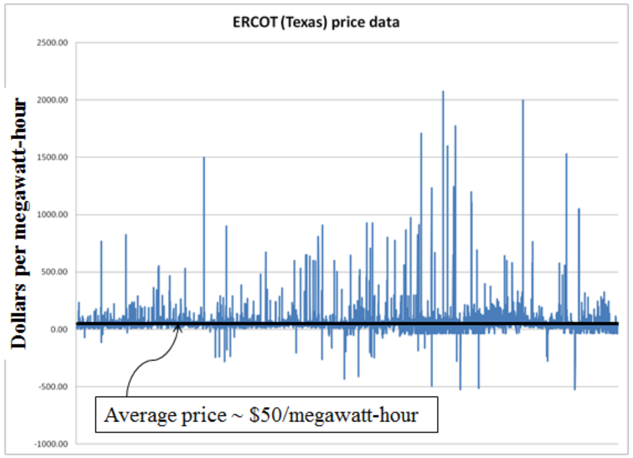
Week 8 – Chapter 11

Policy function approximations
Policy search

Policy function approximations

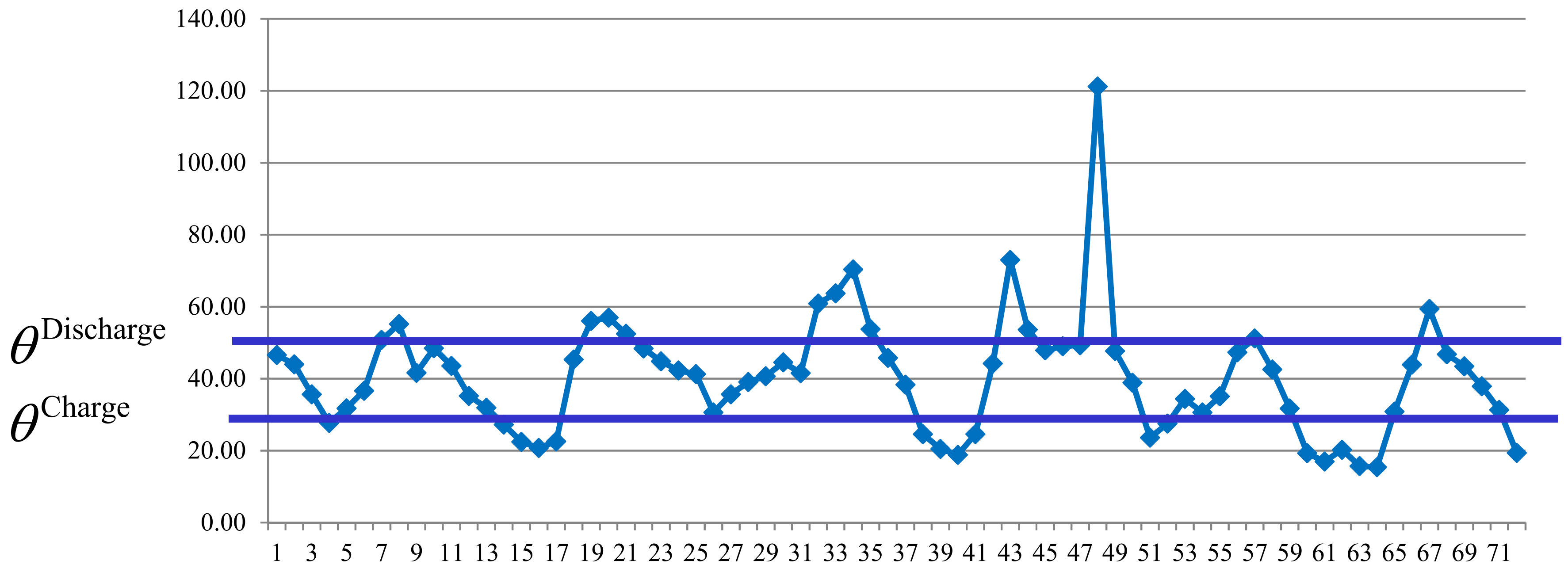
Policy function approximations

- Battery arbitrage – When to charge, when to discharge, given volatile LMPs



Policy function approximations

- Grid operators require that batteries bid charge and discharge prices, an hour in advance.

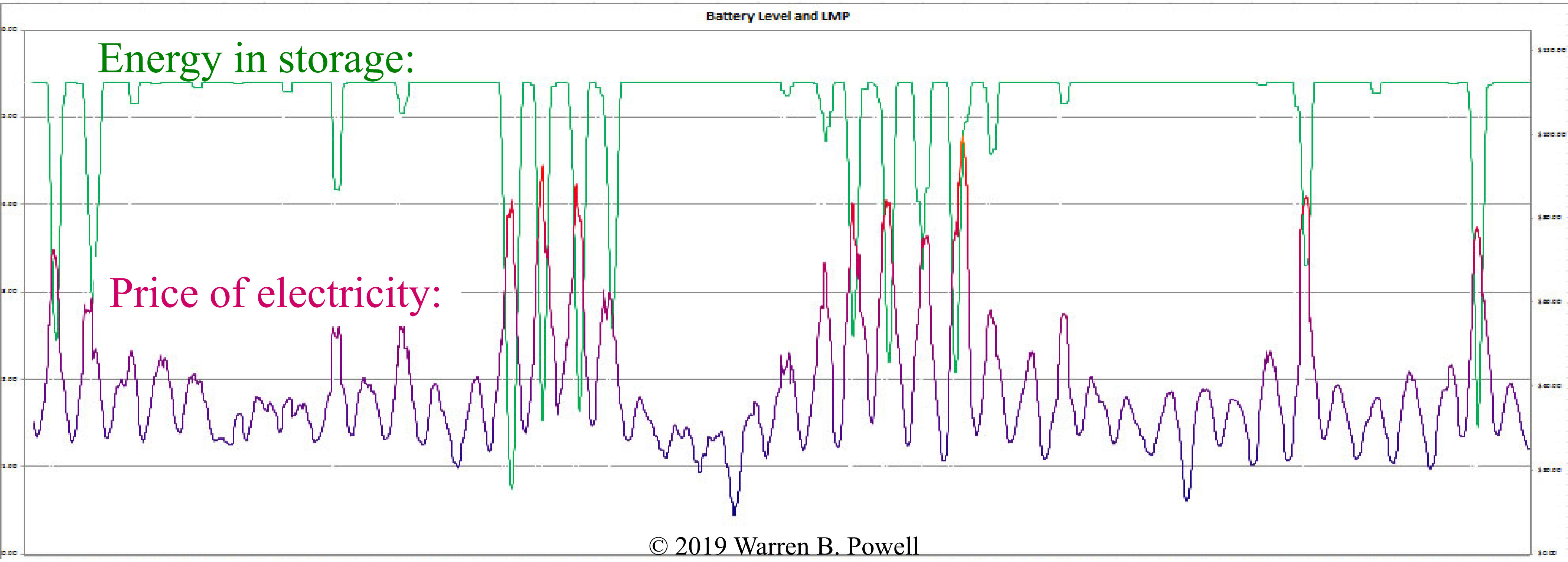


- We have to search for the best values for the policy parameters θ^{Charge} and $\theta^{\text{Discharge}}$.

Policy function approximations

- Our policy function might be the parametric model (this is nonlinear in the parameters):

$$X^{\pi}(S_t \mid \theta) = \begin{cases} +1 & \text{if } p_t < \theta^{\text{charge}} \\ 0 & \text{if } \theta^{\text{charge}} < p_t < \theta^{\text{discharge}} \\ -1 & \text{if } p_t > \theta^{\text{charge}} \end{cases}$$

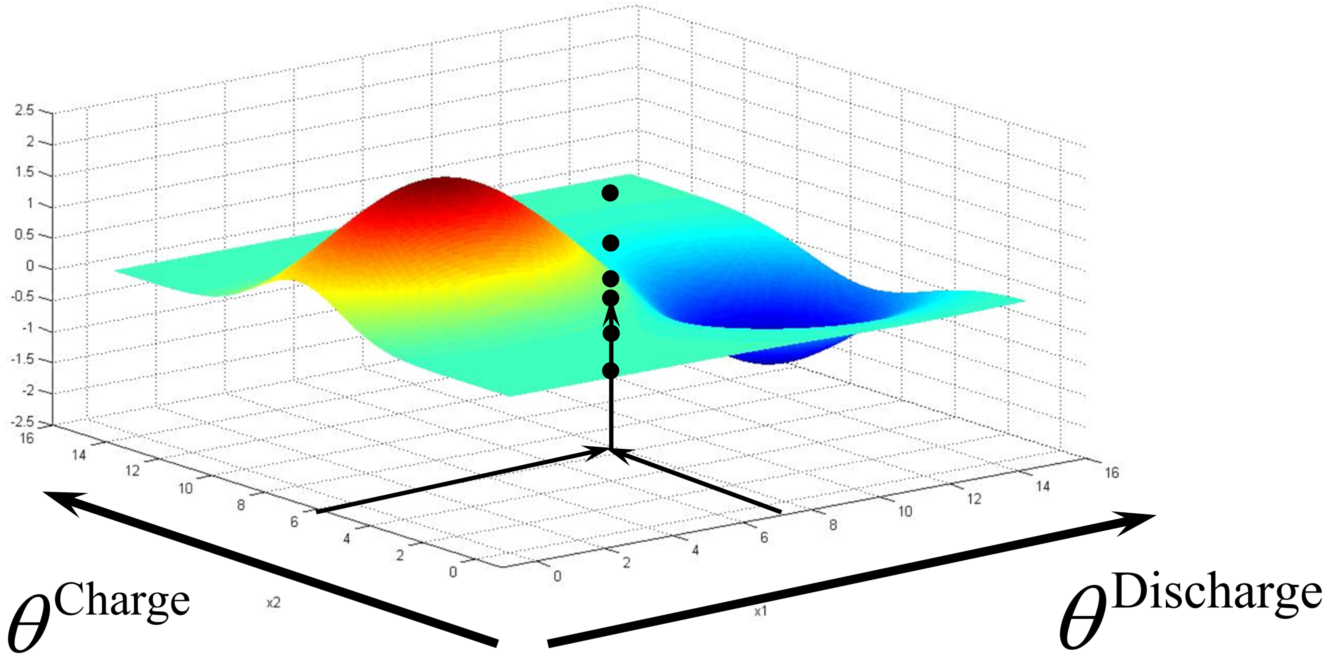


Numerical derivatives

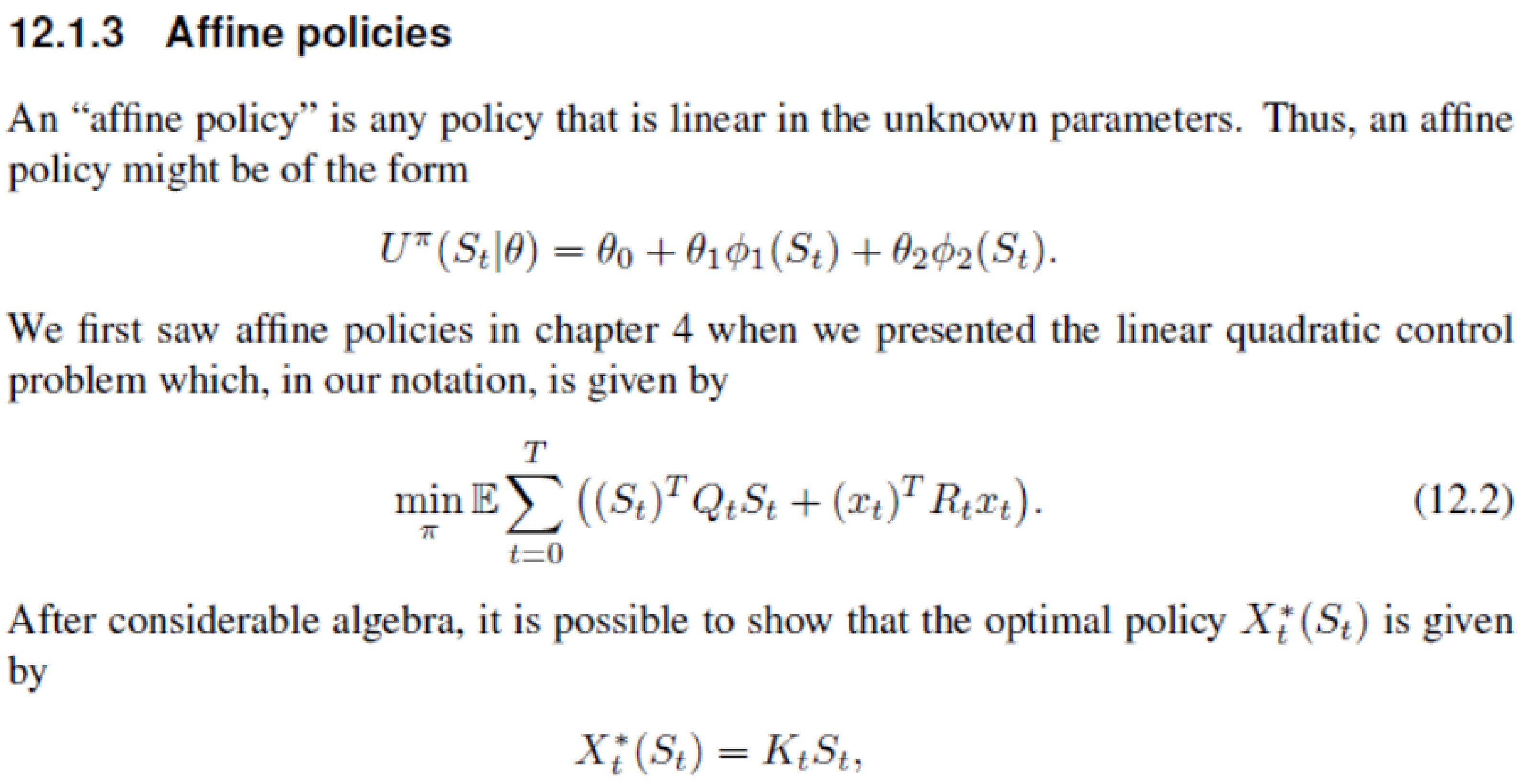
- Finding the best policy
  - » We need to maximize

$$\max_\theta F(\theta) = \mathbb{E}\sum_{t=0}^{T}\gamma^t C\left(S_t, X_t^\pi(S_t \mid \theta)\right)$$

  - » We cannot compute the expectation, so we run simulations:



PFAs



12.1.3 Affine policies

An "affine policy" is any policy that is linear in the unknown parameters. Thus, an affine policy might be of the form

$$U^\pi(S_t|\theta) = \theta_0 + \theta_1\phi_1(S_t) + \theta_2\phi_2(S_t).$$

We first saw affine policies in chapter 4 when we presented the linear quadratic control problem which, in our notation, is given by

$$\min_\pi \mathbb{E} \sum_{t=0}^{T} \left((S_t)^T Q_t S_t + (x_t)^T R_t x_t\right). \tag{12.2}$$

After considerable algebra, it is possible to show that the optimal policy $X_t^*(S_t)$ is given by

$$X_t^*(S_t) = K_t S_t,$$

» Also called "linear decision rules"

■ EXAMPLE 12.1

A basic inventory policy is to order product when the inventory goes below some value θ^L where we order up to some upper value θ^U. If S_t is the inventory level, this policy might be written

$$X^\pi(S_t|\theta) = \begin{cases} \theta^U - S_t & \text{If } S_t < \theta^L, \\ 0 & \text{Otherwise.} \end{cases}$$

■ EXAMPLE 12.3

The outflow u_t of a water reservoir is given by a piecewise linear function of the reservoir level R_t according to:

$$U^\pi(S_t|\theta) = \begin{cases} 0 & R_t < R^{min}, \\ \theta_1 & 0 \times (R^{max} - R^{min}) \le R_t - R^{min} \le .2(R^{max} - R^{min}), \\ \theta_2 & .2 \times (R^{max} - R^{min}) \le R_t - R^{min} \le .4(R^{max} - R^{min}), \\ \theta_3 & .4 \times (R^{max} - R^{min}) \le R_t - R^{min} \le .6(R^{max} - R^{min}), \\ \theta_4 & .6 \times (R^{max} - R^{min}) \le R_t - R^{min} \le .8(R^{max} - R^{min}), \\ \theta_5 & .8 \times (R^{max} - R^{min}) \le R_t - R^{min} \le 1.0(R^{max} - R^{min}), \\ \theta^{max} & R_t > R^{max}. \end{cases}$$

where we would expect $\theta_{i+1} \ge \theta_i$.

PFAs

12.1.4 Locally linear policies

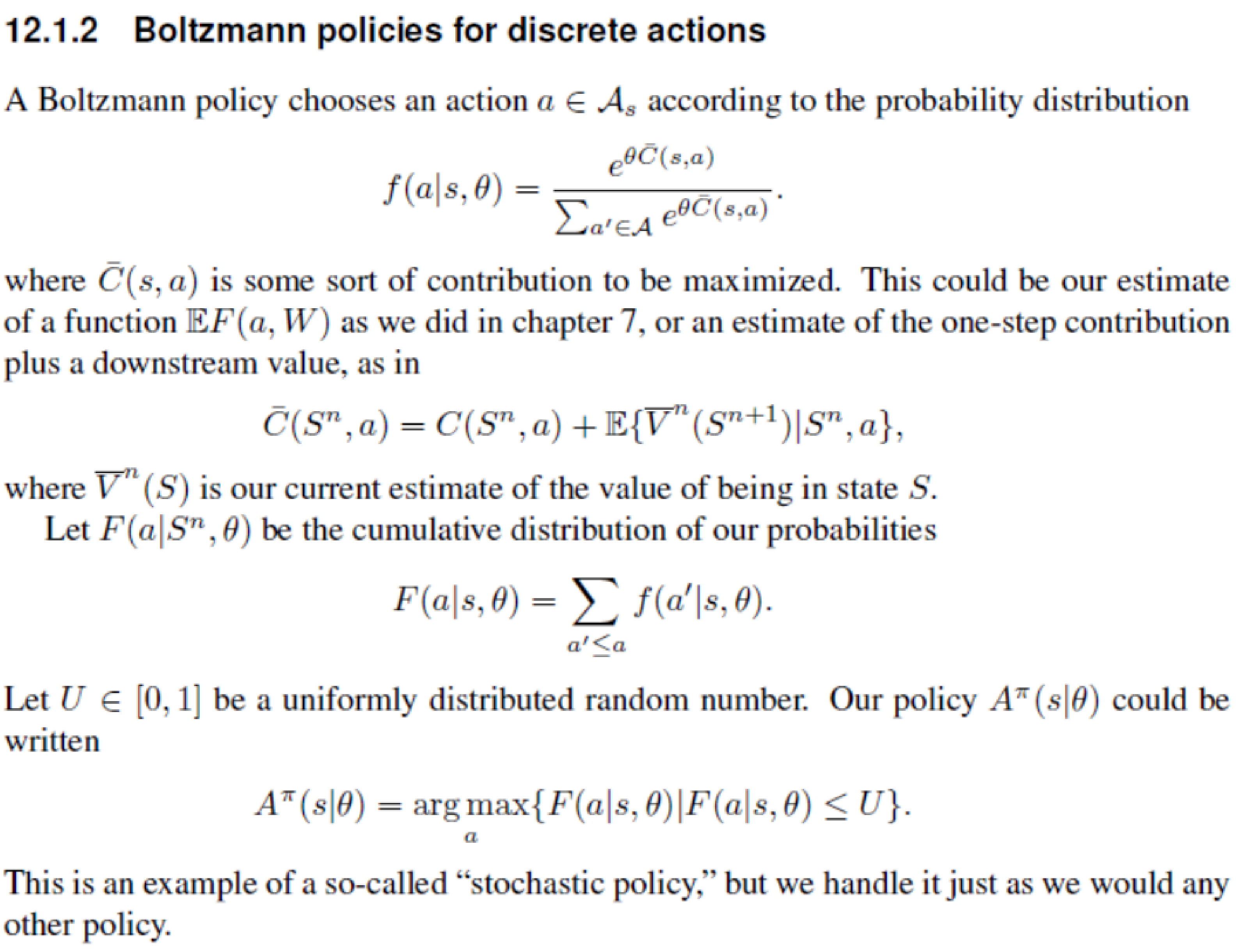
A surprisingly powerful strategy for many problems with continuous states and actions is to assume locally linear responses. For example, S_t may capture the level of a reservoir, or the current speed and altitude of a helicopter. The control x_t could be the rate at which water is released from the reservoir, or the forces applied to the helicopter. Assume that we use our understanding of the problem to create a family of regions $\mathcal{S}_1, \ldots, \mathcal{S}_I$, which are most likely going to be a set of rectangular regions (or intervals if there is only one dimension). We might then create a family of linear (affine) policies of the form

$$X_i^\pi(S_t|\theta) = \theta_{i0} + \theta_{i1}\phi_1(S_t) + \theta_{i2}\phi_2(S_t),$$

for $S_t \in \mathcal{S}_i$.

This approach has been found to be very effective in some classes of control problems. In practice, the regions $\mathcal{S}_i$ are designed by someone with an understanding of the physics of the problem. Further, instead of tuning one vector θ, we have to tune $\theta_1, \ldots, \theta_I$. While this can represent a laboratory challenge, the approach can work quite well, and offers the important feature that they can be computed extremely quickly.

PFAs



12.1.2 Boltzmann policies for discrete actions

A Boltzmann policy chooses an action $a \in \mathcal{A}_s$ according to the probability distribution

$$f(a|s,\theta) = \frac{e^{\theta \bar{C}(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\theta \bar{C}(s,a)}}.$$

where $\bar{C}(s,a)$ is some sort of contribution to be maximized. This could be our estimate of a function $\mathbb{E}F(a,W)$ as we did in chapter 7, or an estimate of the one-step contribution plus a downstream value, as in

$$\bar{C}(S^n,a) = C(S^n,a) + \mathbb{E}\{\overline{V}^n(S^{n+1})|S^n,a\},$$

where $\overline{V}^n(S)$ is our current estimate of the value of being in state S.

Let $F(a|S^n,\theta)$ be the cumulative distribution of our probabilities

$$F(a|s,\theta) = \sum_{a' \leq a} f(a'|s,\theta).$$

Let $U \in [0,1]$ be a uniformly distributed random number. Our policy $A^\pi(s|\theta)$ could be written

$$A^\pi(s|\theta) = \arg\max_a \{F(a|s,\theta)|F(a|s,\theta) \leq U\}.$$

This is an example of a so-called "stochastic policy," but we handle it just as we would any other policy.

12.1.5 Monotone policies

There are a number of problems where the decision increases, or decreases, with the state variable. If the state variable is multidimensional, then the decision (which we assume is scalar) increases, or decreases, with *each* dimension of the state variable. Policies with this structure are known as *monotone policies*. Some examples include:

- There are a number of problems with binary actions that can be modeled as $x \in \{0, 1\}$. For example
  - We may hold a stock ($x_t = 0$) or sell ($x_t = 1$) if the price p_t falls below a smoothed estimate $\bar{p}_t$ which we compute using

    $$\bar{p}_t = (1-\alpha)\bar{p}_{t-1} + \alpha p_t.$$

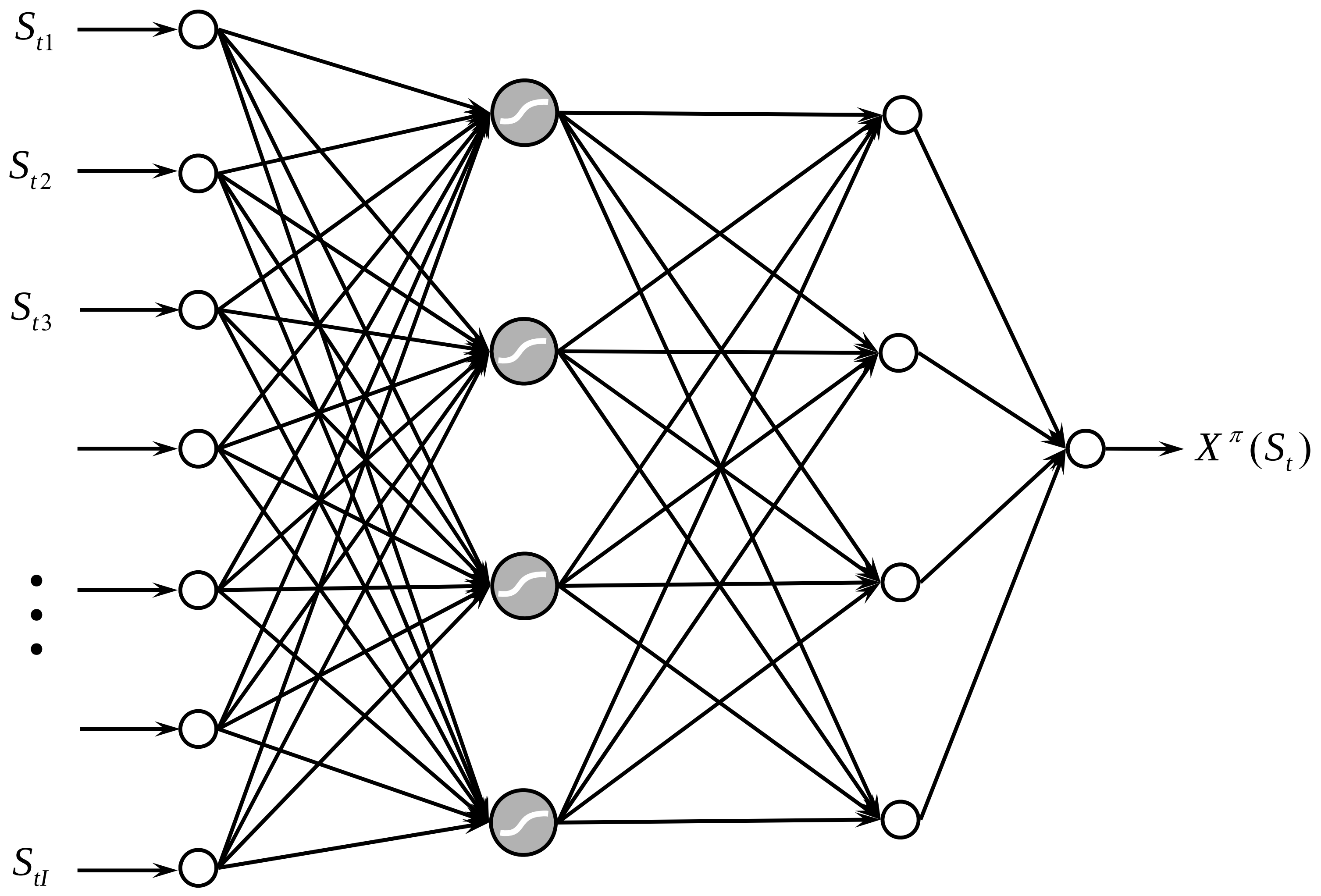
    Our policy is then given by

    $$X^\pi(S_t|\theta) = \begin{cases} 1 & \text{If } p_t \le \bar{p}_t - \theta \\ 0 & \text{Otherwise.} \end{cases}$$

    The function $X^\pi(S_t|\theta)$ decreases monotonically in p_t (as p_t increases, $X^\pi(S_t|\theta)$ goes from 1 to 0).
  - A shuttle bus waits until there are at least R_t customers on the bus, or it has waited τ_t. The decision to dispatch goes from $x_t = 0$ (hold the bus) to $x_t = 1$ (dispatch the bus) as R_t exceeds a threshold θ^R or as τ_t exceeds θ^τ, which means the policy $X^\pi(S_t|\theta)$ increases monotonically in both state variables $S_t = (R_t, \tau_t)$.

PFAs

- Neural networks as policies



PFAs

- Neural networks as policies
  - » Each link is characterized by a weight that is normally called w_{ij}, but which we will call θ_{ij} for consistency with our prior notation.
  - » We can represent our neural network policy then as $X^{NN}(S^n|\theta)$.
  - » The weight vector θ may easily have hundreds or even thousands of dimensions. This is not a major problem because we can compute derivatives of the policy.

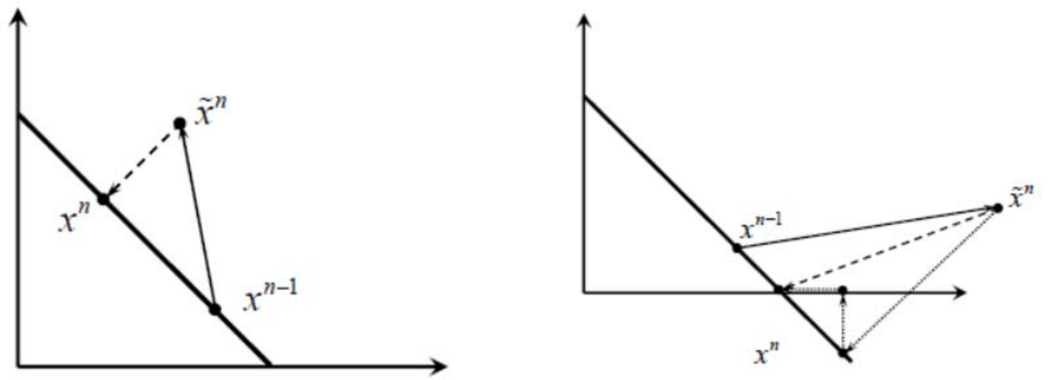
12.1.9 Constraints

An issue that arises with policy function approximations is the handling of constraints, since it can be difficult or impossible to design analytical functions that guarantee that a decision satisfies a set of constraints. Constraints are typically handled using a simple projection. This is represented mathematically by a projfection operator $\Pi_{\mathcal{X}}(x)$ (nothing to do with policies) that maps a point x onto a region $\mathcal{X}$. So, we would write our policy using

$$x_t = \Pi_{\mathcal{X}_t}[X_t^\pi(S_t)].$$

The easiest constraints to handle are box constraints of the form $0 \leq x_t \leq u_t$ where u_t are upper bounds on each dimension of x_t. In this case, if our function $X_t^\pi(S_t)$ returns a (vector-valued) decision x_t, we simply have to check each dimension of x_t and impose these constraints (elements less than 0 are set equal to 0, while elements greater than their corresponding value in u_t are set to the value in u_t).

Slightly harder are constraints of the form $Ax = b_t$ or $Ax \leq b_t$. The project process is illustrated in figure 12.5. Figure 12.5a demonstrates a basic projection of a point $\tilde{x}^n$ from

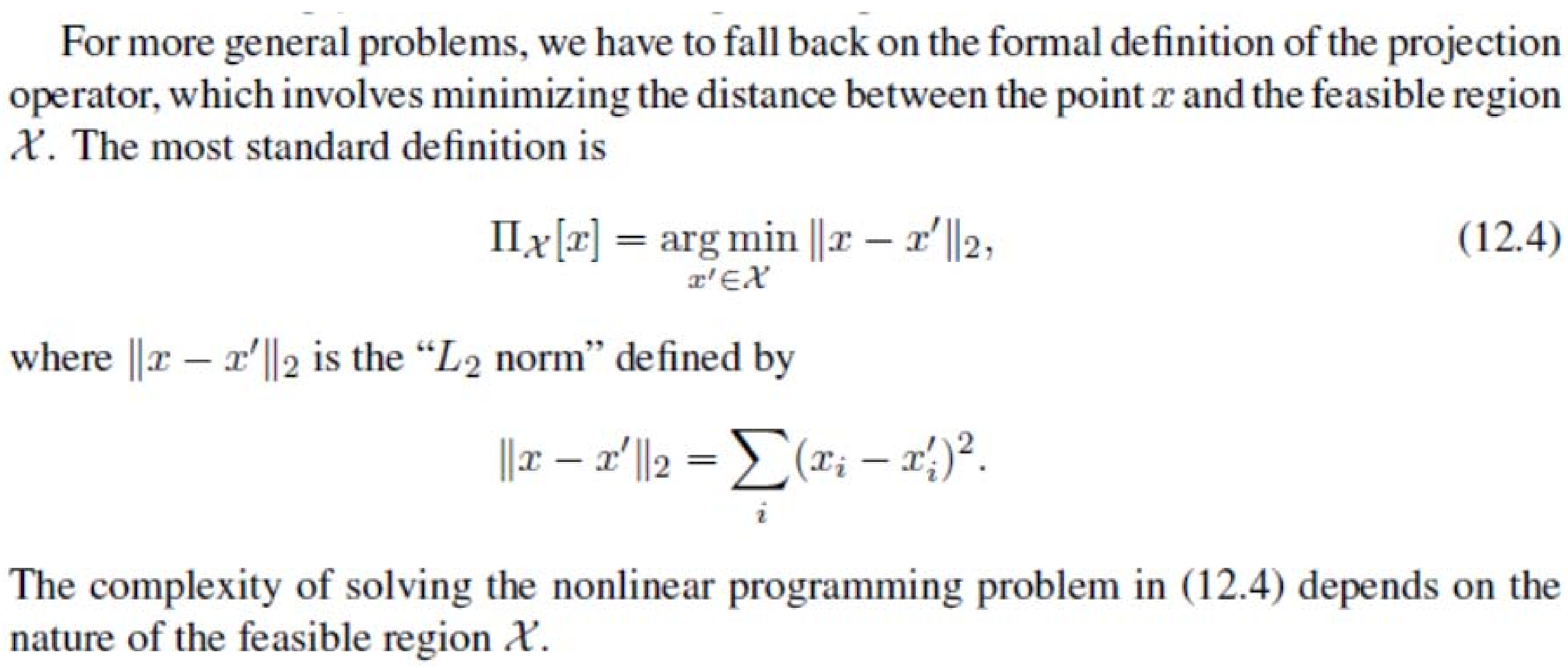


(a): Simple projection (b): Compound projection

Figure 12.5 Illustration of projection onto a linear feasible region.

PFAs

- Constraints:



For more general problems, we have to fall back on the formal definition of the projection operator, which involves minimizing the distance between the point x and the feasible region $\mathcal{X}$. The most standard definition is

$$\Pi_{\mathcal{X}}[x] = \arg\min_{x' \in \mathcal{X}} \|x - x'\|_2, \qquad (12.4)$$

where $\|x - x'\|_2$ is the "L_2 norm" defined by

$$\|x - x'\|_2 = \sum_i (x_i - x'_i)^2.$$

The complexity of solving the nonlinear programming problem in (12.4) depends on the nature of the feasible region $\mathcal{X}$.

Policy search

Derivative-based policy search

Derivative-based policy search

- The core problem of policy search is a classical stochastic search problem:

$$\max_x \mathbb{E}\{F(x,W)|S_0\}$$

where "x" is a policy, W represents any form of learning or testing randomness, and S_0 might capture any prior distributions.

» There will always be two solution approaches:
  - Derivative-based (Chapter 5)
  - Derivative-free (Chapter 7)

» … but there are some details unique to policy search arising from the sequential nature of these problems.

12.2 POLICY SEARCH

Given a parametric (or locally parametric) function parameterized by θ (typically a vector, but not always), we now face the challenge of finding the best value of θ. There are different styles of policy search:

Derivative-based vs. derivative free In some cases we can approximate derivatives with respect to θ, although these are typically quite approximate. Alternatively we can use the derivative-free methods in chapter 7, although it is likely that this will be limited to low-dimensional parameter vectors.

Online vs. offline learning In online learning, we are learning in an environment where updates come to us. As a rule, we have to live with the performance of our policy, which means we are maximizing the cumulative reward. Most policy search uses some form of adaptive algorithm, although this can be done in a laboratory where we use one policy, the *learning policy* to find the best policy to implement, called the *implementation policy*.

Stationary vs. nonstationary environments Most of the analysis of algorithms is performed in the context of stationary (possibly even static) environments, where exogenous information comes from a single distribution. When working in online settings (in the field), it is more often the case that data is coming from a nonstationary setting.

Performance-based vs. supervisory learning Most policy search uses as a goal to maximize the total reward (either the final reward or cumulative reward), but there are settings where we have an "expert" (the supervisor) who will specify what to do, allowing us to fit our policies to the choices of the supervisor.

Derivative-based policy search

address in chapter 5. To do this, it is useful to identify three classes of problems:

Discrete dynamic programs - These are problems where we are at a node (state) s, choose a discrete action a and then transition to a node s' with probability $P(s'|s,a)$ (which we represent but generally cannot compute). An important subclass of graph problems are those where actions are chosen at random (known as a stochastic policy), but transitions are made deterministically. Here, we wish to optimize a parameterized policy $A^\pi(s|\theta)$, where action $a_t = A^\pi(S_t|\theta)$ is discrete.

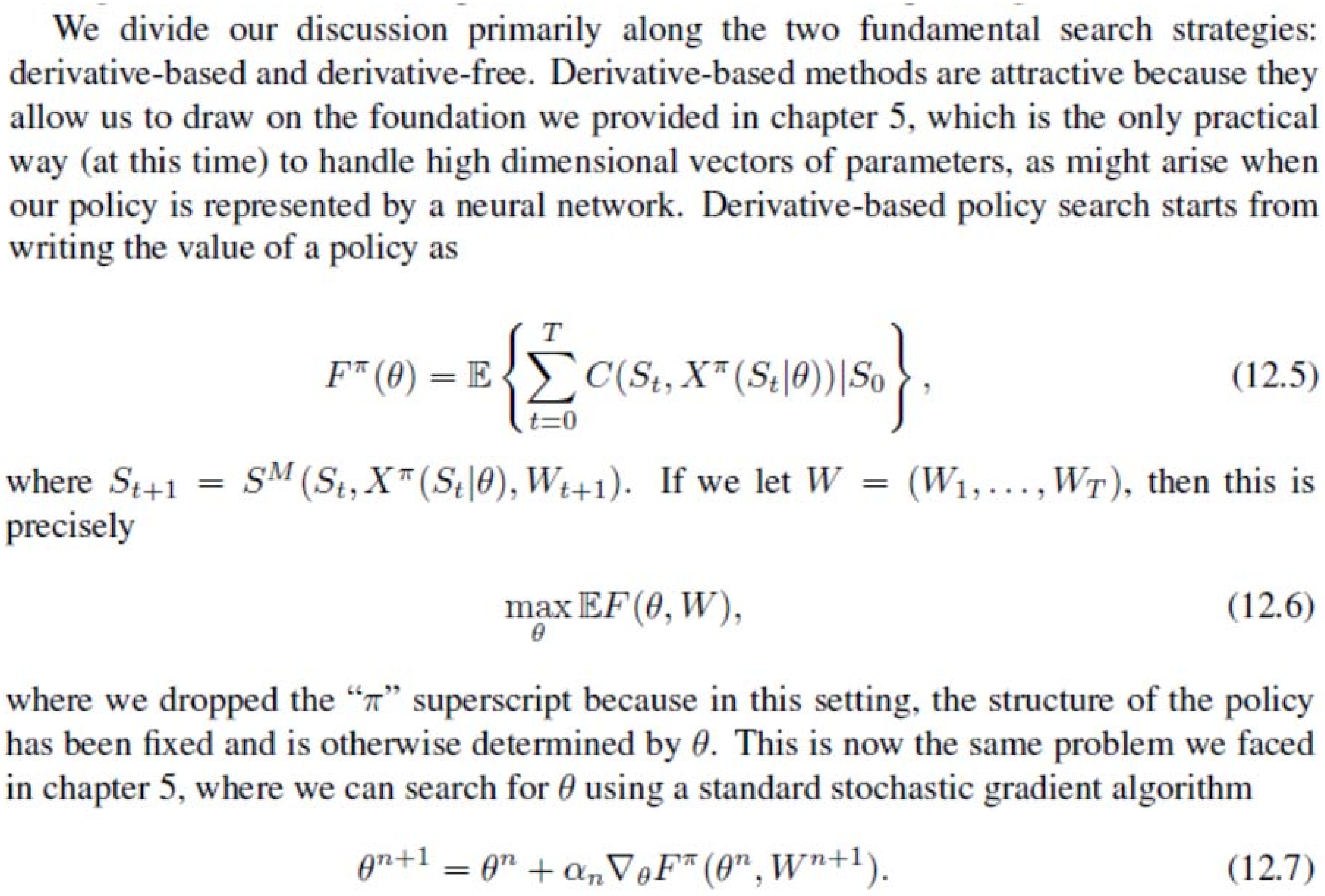
Control problems - In this setting we choose a continuous control u_t that impacts the state S_{t+1} in a continuous way through a known (and differentiable) transition function.

Resource allocation - Here, we have a vector of resources R_t which we move with a vector x_t to produce a new allocation R_{t+1}, possibly with random perturbations, according to the equation

$$R_{t+1} = R_t + A_t x_t + \hat{R}_{t+1},$$

where R_t and x_t are vectors, and A_t is a suitably defined matrix. This problem class includes all of the physical resource allocation problems described in chapter 8. For this problem class, we wish to optimize a parameterized policy $X^\pi(s|\theta)$, where $x_t = X^\pi(S_t|\theta)$ is typically a vector (possibly high dimensional).

Derivative-based policy search



We divide our discussion primarily along the two fundamental search strategies: derivative-based and derivative-free. Derivative-based methods are attractive because they allow us to draw on the foundation we provided in chapter 5, which is the only practical way (at this time) to handle high dimensional vectors of parameters, as might arise when our policy is represented by a neural network. Derivative-based policy search starts from writing the value of a policy as

$$F^\pi(\theta) = \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X^\pi(S_t|\theta))|S_0\right\}, \tag{12.5}$$

where $S_{t+1} = S^M(S_t, X^\pi(S_t|\theta), W_{t+1})$. If we let $W = (W_1, \ldots, W_T)$, then this is precisely

$$\max_\theta \mathbb{E} F(\theta, W), \tag{12.6}$$

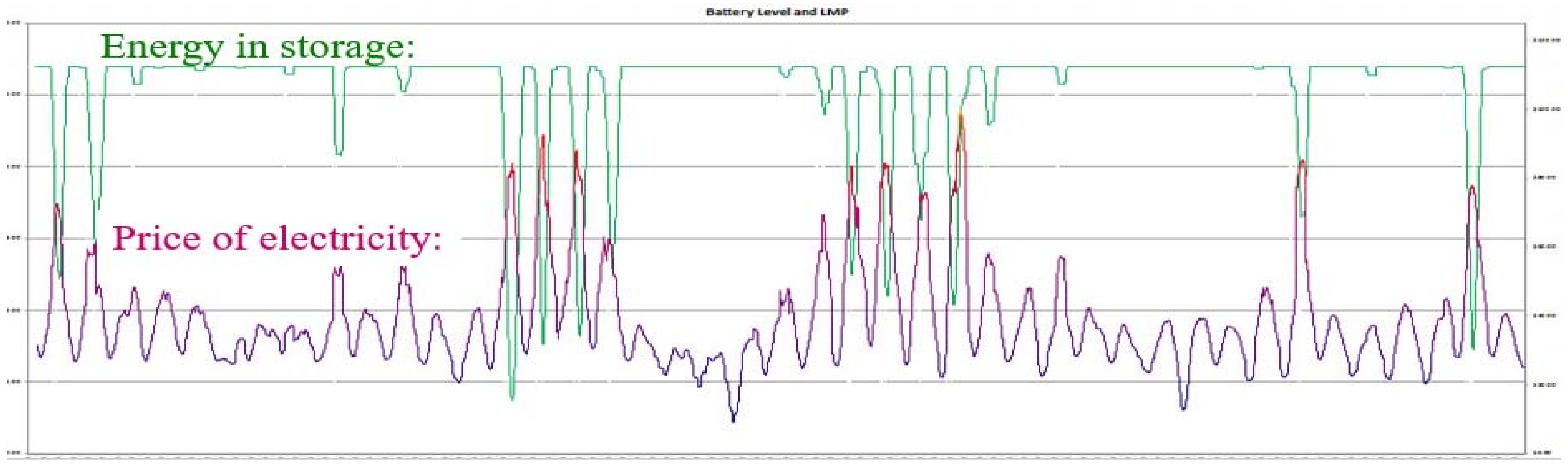
where we dropped the "π" superscript because in this setting, the structure of the policy has been fixed and is otherwise determined by θ. This is now the same problem we faced in chapter 5, where we can search for θ using a standard stochastic gradient algorithm

$$\theta^{n+1} = \theta^n + \alpha_n \nabla_\theta F^\pi(\theta^n, W^{n+1}). \tag{12.7}$$

Derivative-based policy search

- Finite differences
  - » We wish to optimize the decision of when to charge or discharge a battery

$$X^{\pi}(S_t \mid \theta) = \begin{cases} +1 & \text{if } p_t < \theta^{\text{charge}} \\ 0 & \text{if } \theta^{\text{charge}} < p_t < \theta^{\text{discharge}} \\ -1 & \text{if } p_t > \theta^{\text{charge}} \end{cases}$$

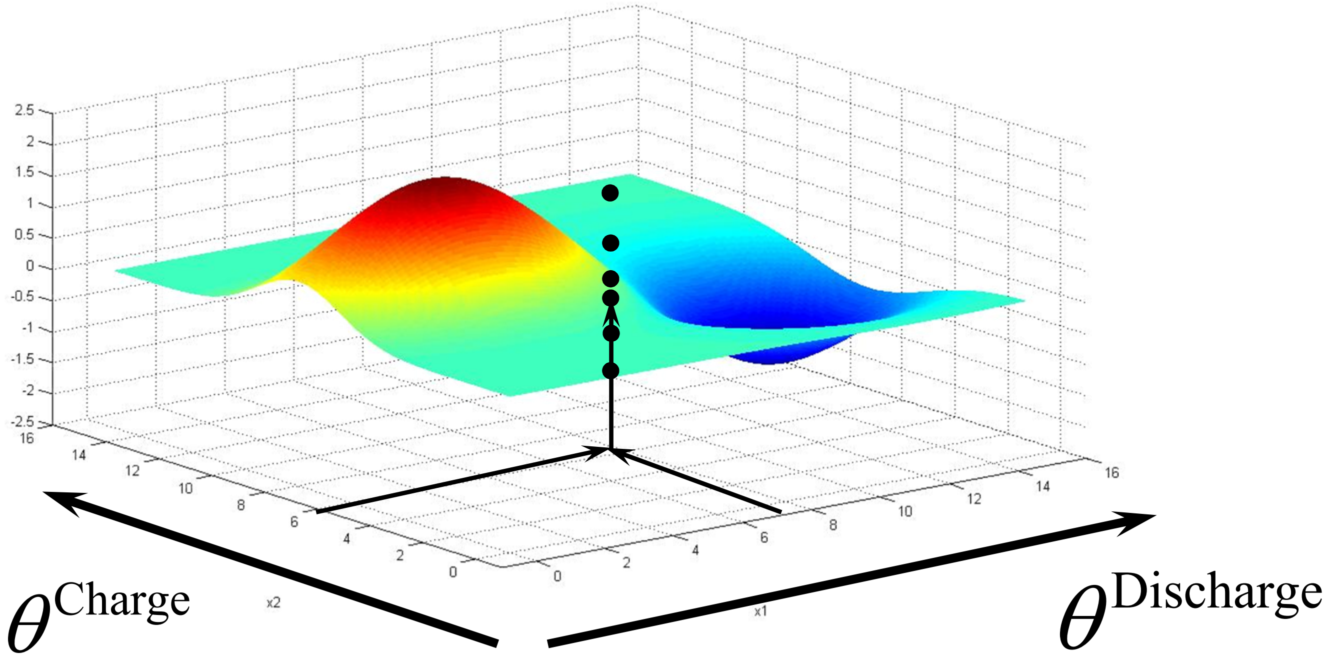


Derivative-based policy search

- Finding the best policy
  - » We need to maximize

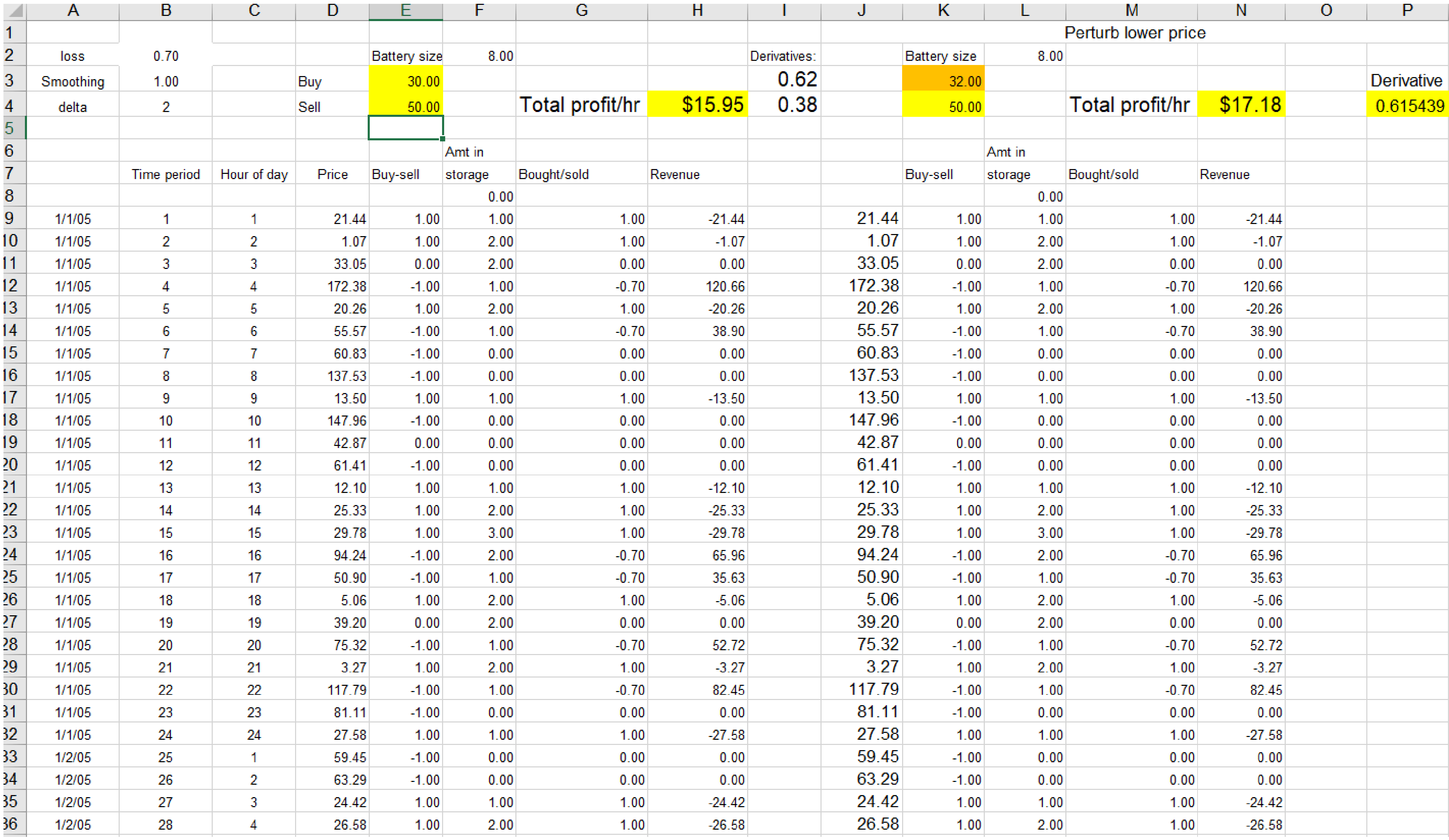
$$\max_\theta F(\theta) = \mathbb{E}\sum_{t=0}^{T}\gamma^t C\left(S_t, X_t^\pi(S_t \mid \theta)\right)$$

  - » We cannot compute the expectation, so we run simulations:



Derivative-based policy search

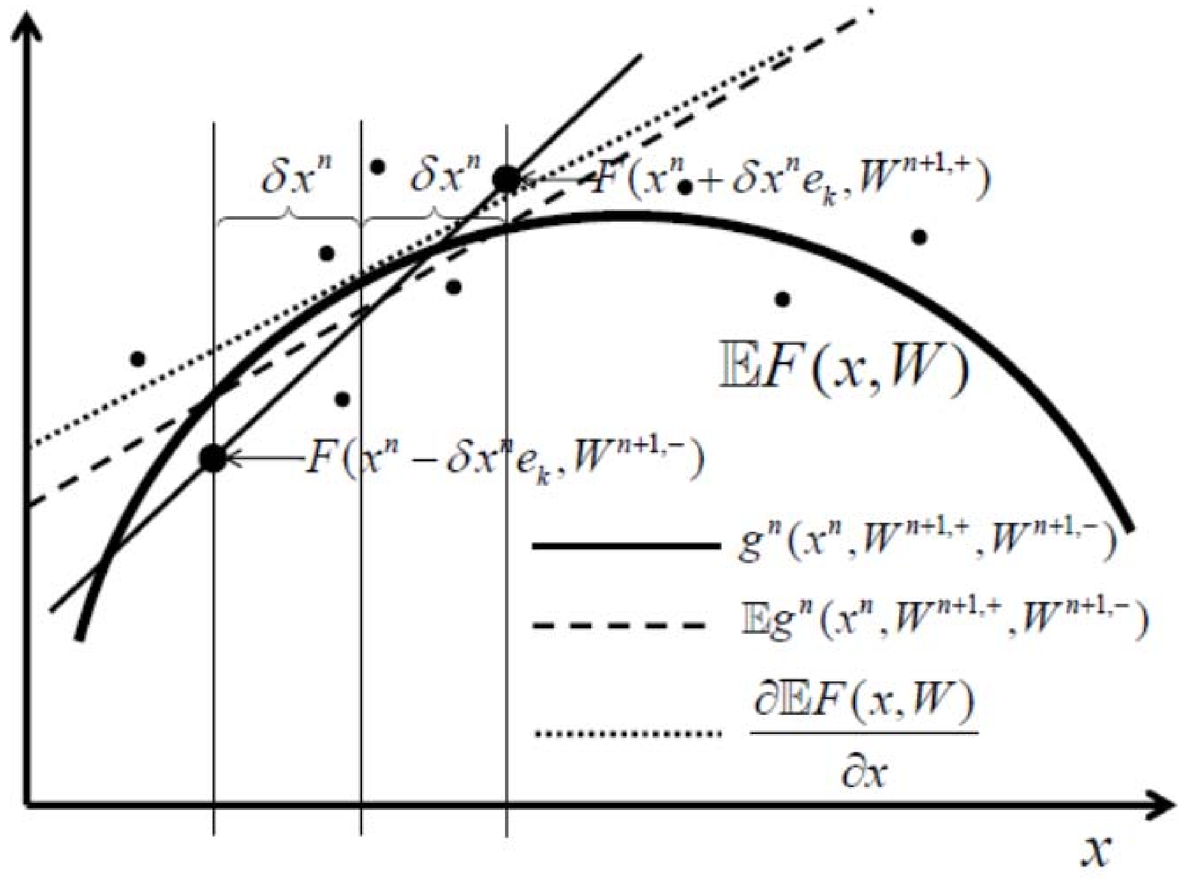
- Simulating finite difference



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1													Perturb lower price			
2	loss	0.70			Battery size	8.00			Derivatives:		Battery size	8.00				
3	Smoothing	1.00		Buy	30.00				0.62		32.00					Derivative
4	delta	2		Sell	50.00		Total profit/hr	$15.95	0.38		50.00		Total profit/hr	$17.18		0.615439
5																
6						Amt in						Amt in				
7		Time period	Hour of day	Price	Buy-sell	storage	Bought/sold	Revenue			Buy-sell	storage	Bought/sold	Revenue		
8						0.00						0.00				
9	1/1/05	1	1	21.44	1.00	1.00	1.00	-21.44		21.44	1.00	1.00	1.00	-21.44		
10	1/1/05	2	2	1.07	1.00	2.00	1.00	-1.07		1.07	1.00	2.00	1.00	-1.07		
11	1/1/05	3	3	33.05	0.00	2.00	0.00	0.00		33.05	0.00	2.00	0.00	0.00		
12	1/1/05	4	4	172.38	-1.00	1.00	-0.70	120.66		172.38	-1.00	1.00	-0.70	120.66		
13	1/1/05	5	5	20.26	1.00	2.00	1.00	-20.26		20.26	1.00	2.00	1.00	-20.26		
14	1/1/05	6	6	55.57	-1.00	1.00	-0.70	38.90		55.57	-1.00	1.00	-0.70	38.90		
15	1/1/05	7	7	60.83	-1.00	0.00	0.00	0.00		60.83	-1.00	0.00	0.00	0.00		
16	1/1/05	8	8	137.53	-1.00	0.00	0.00	0.00		137.53	-1.00	0.00	0.00	0.00		
17	1/1/05	9	9	13.50	1.00	1.00	1.00	-13.50		13.50	1.00	1.00	1.00	-13.50		
18	1/1/05	10	10	147.96	-1.00	0.00	0.00	0.00		147.96	-1.00	0.00	0.00	0.00		
19	1/1/05	11	11	42.87	0.00	0.00	0.00	0.00		42.87	0.00	0.00	0.00	0.00		
20	1/1/05	12	12	61.41	-1.00	0.00	0.00	0.00		61.41	-1.00	0.00	0.00	0.00		
21	1/1/05	13	13	12.10	1.00	1.00	1.00	-12.10		12.10	1.00	1.00	1.00	-12.10		
22	1/1/05	14	14	25.33	1.00	2.00	1.00	-25.33		25.33	1.00	2.00	1.00	-25.33		
23	1/1/05	15	15	29.78	1.00	3.00	1.00	-29.78		29.78	1.00	3.00	1.00	-29.78		
24	1/1/05	16	16	94.24	-1.00	2.00	-0.70	65.96		94.24	-1.00	2.00	-0.70	65.96		
25	1/1/05	17	17	50.90	-1.00	1.00	-0.70	35.63		50.90	-1.00	1.00	-0.70	35.63		
26	1/1/05	18	18	5.06	1.00	2.00	1.00	-5.06		5.06	1.00	2.00	1.00	-5.06		
27	1/1/05	19	19	39.20	0.00	2.00	0.00	0.00		39.20	0.00	2.00	0.00	0.00		
28	1/1/05	20	20	75.32	-1.00	1.00	-0.70	52.72		75.32	-1.00	1.00	-0.70	52.72		
29	1/1/05	21	21	3.27	1.00	2.00	1.00	-3.27		3.27	1.00	2.00	1.00	-3.27		
30	1/1/05	22	22	117.79	-1.00	1.00	-0.70	82.45		117.79	-1.00	1.00	-0.70	82.45		
31	1/1/05	23	23	81.11	-1.00	0.00	0.00	0.00		81.11	-1.00	0.00	0.00	0.00		
32	1/1/05	24	24	27.58	1.00	1.00	1.00	-27.58		27.58	1.00	1.00	1.00	-27.58		
33	1/2/05	25	1	59.45	-1.00	0.00	0.00	0.00		59.45	-1.00	0.00	0.00	0.00		
34	1/2/05	26	2	63.29	-1.00	0.00	0.00	0.00		63.29	-1.00	0.00	0.00	0.00		
35	1/2/05	27	3	24.42	1.00	1.00	1.00	-24.42		24.42	1.00	1.00	1.00	-24.42		
36	1/2/05	28	4	26.58	1.00	2.00	1.00	-26.58		26.58	1.00	2.00	1.00	-26.58		

Derivative-based policy search

- Finite differences

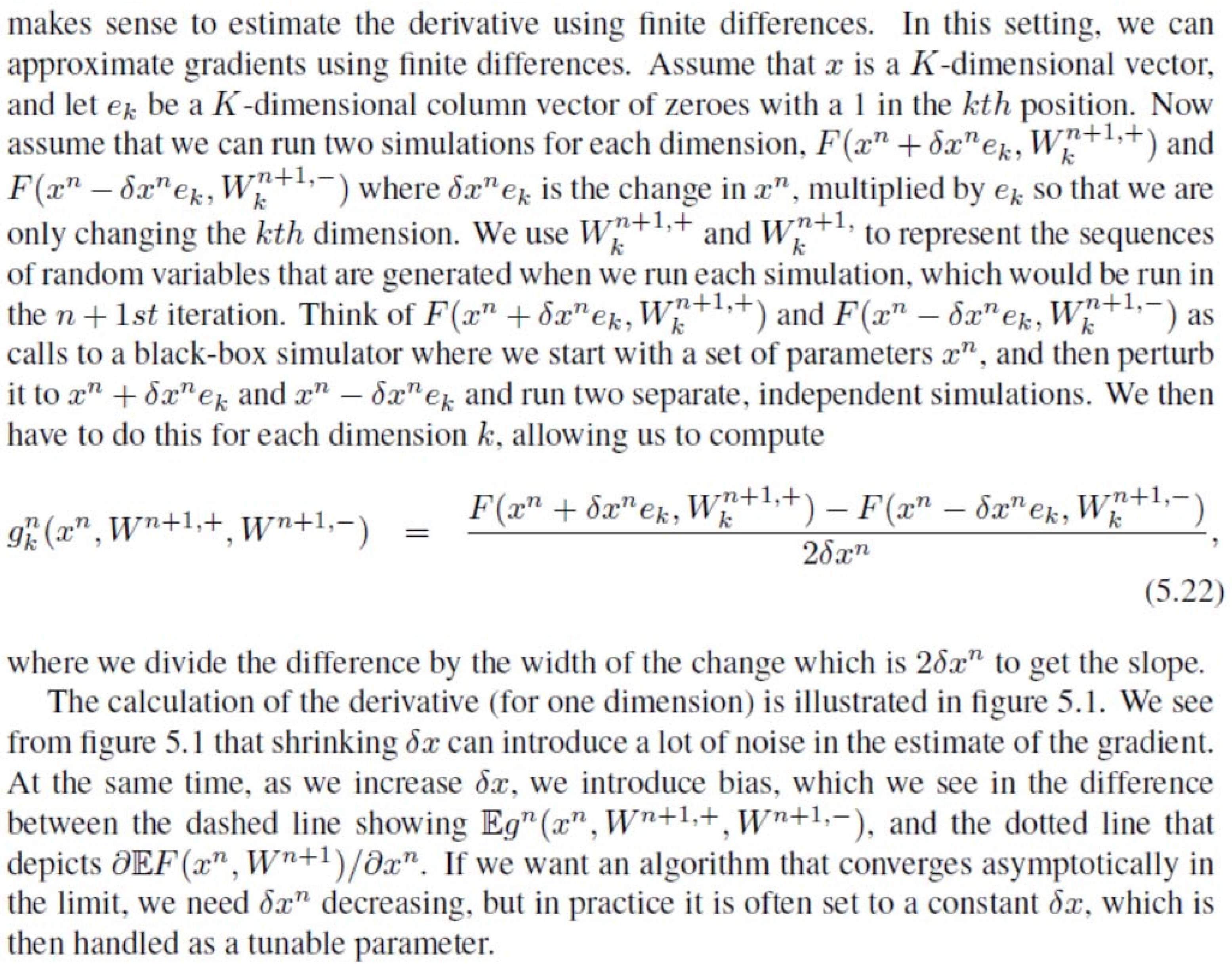


δx^n
δx^n
$F(x^n + \delta x^n e_k, W^{n+1,+})$
$\mathbb{E}F(x,W)$
$F(x^n - \delta x^n e_k, W^{n+1,-})$
$g^n(x^n, W^{n+1,+}, W^{n+1,-})$
$\mathbb{E}g^n(x^n, W^{n+1,+}, W^{n+1,-})$
$\frac{\partial \mathbb{E}F(x,W)}{\partial x}$
x

Figure 5.1 Different estimates of the gradient of $F(x, W)$ with a) the stochastic gradient $g^n(x^n, W^{n+1,+}, W^{n+1,-})$ (solid line), the expected finite difference $\mathbb{E}g^n(x^n, W^{n+1,+}, W^{n+1,-})$ (dashed line), and the exact slope at x^n, $\partial \mathbb{E}F(x^n, W^{n+1})/\partial x^n$.

Derivative-based policy search

- Finite differences



makes sense to estimate the derivative using finite differences. In this setting, we can approximate gradients using finite differences. Assume that x is a K-dimensional vector, and let e_k be a K-dimensional column vector of zeroes with a 1 in the kth position. Now assume that we can run two simulations for each dimension, $F(x^n + \delta x^n e_k, W_k^{n+1,+})$ and $F(x^n - \delta x^n e_k, W_k^{n+1,-})$ where $\delta x^n e_k$ is the change in x^n, multiplied by e_k so that we are only changing the kth dimension. We use $W_k^{n+1,+}$ and $W_k^{n+1,}$ to represent the sequences of random variables that are generated when we run each simulation, which would be run in the $n+1st$ iteration. Think of $F(x^n + \delta x^n e_k, W_k^{n+1,+})$ and $F(x^n - \delta x^n e_k, W_k^{n+1,-})$ as calls to a black-box simulator where we start with a set of parameters x^n, and then perturb it to $x^n + \delta x^n e_k$ and $x^n - \delta x^n e_k$ and run two separate, independent simulations. We then have to do this for each dimension k, allowing us to compute

$$g_k^n(x^n, W^{n+1,+}, W^{n+1,-}) = \frac{F(x^n + \delta x^n e_k, W_k^{n+1,+}) - F(x^n - \delta x^n e_k, W_k^{n+1,-})}{2\delta x^n}, \tag{5.22}$$

where we divide the difference by the width of the change which is $2\delta x^n$ to get the slope.

The calculation of the derivative (for one dimension) is illustrated in figure 5.1. We see from figure 5.1 that shrinking δx can introduce a lot of noise in the estimate of the gradient. At the same time, as we increase δx, we introduce bias, which we see in the difference between the dashed line showing $\mathbb{E}g^n(x^n, W^{n+1,+}, W^{n+1,-})$, and the dotted line that depicts $\partial \mathbb{E}F(x^n, W^{n+1})/\partial x^n$. If we want an algorithm that converges asymptotically in the limit, we need δx^n decreasing, but in practice it is often set to a constant δx, which is then handled as a tunable parameter.

Derivative-based policy search

- Finite differences

There are several strategies we can pursue to reduce this computational burden:

- Instead of perturbing x^n up and down, just do it in one direction (typically up, unless a dimension of x^n is up against a constraint). This means we have to do $K+1$ function evaluations: a base estimate, and then one for each dimension.
- Just estimate the gradient for one dimension, where you would typically choose the dimension at random.
- Randomly perturb all the dimensions all at once. This is known as the *simultaneous perturbation stochastic approximation* procedure (or SPSA).

SPSA computes gradients in the following way. Let $Z_k, k = 1, \ldots, K$ be a vector of zero-mean random variables. We now compute our objective function twice: once to find $F(x^n + Z_k, W_k^{n+1,+})$, and once to find $F(x^n - Z_k, W_k^{n-1,-})$.

$$g^n(x^n, W^{n+1,+}, W^{n+1,-}) = \frac{F(x^n + Z_k, W_k^{n+1,+}) - F(x^n - Z_k, W_k^{n-1,-})}{2c^n Z_k}. \quad (5.23)$$

Derivative-based policy search

- Simultaneous perturbation stochastic approximation
  - » Let:
    - x^n be a p − dimensional vector.
    - δ^n be a scalar perturbation
    - Z^n be a p −dimensional vector, with each element drawn from a normal (0,1) distribution.
  - » We can obtain a sampled estimate of the gradient $\nabla_x F(x^n, W^{n+1})$ using two function evaluations: $F(x^n + \delta^n Z^n)$ and $F(x^n + \delta^n Z^n)$

$$\nabla_x F(x^n, W^{n+1}) = \begin{bmatrix} \dfrac{F(x^n + \delta^n Z^n) - F(x^n + \delta^n Z^n)}{2\delta^n Z_1^n} \\ \dfrac{F(x^n + \delta^n Z^n) - F(x^n + \delta^n Z^n)}{2\delta^n Z_2^n} \\ \vdots \\ \dfrac{F(x^n + \delta^n Z^n) - F(x^n + \delta^n Z^n)}{2\delta^n Z_p^n} \end{bmatrix}$$

Policy search

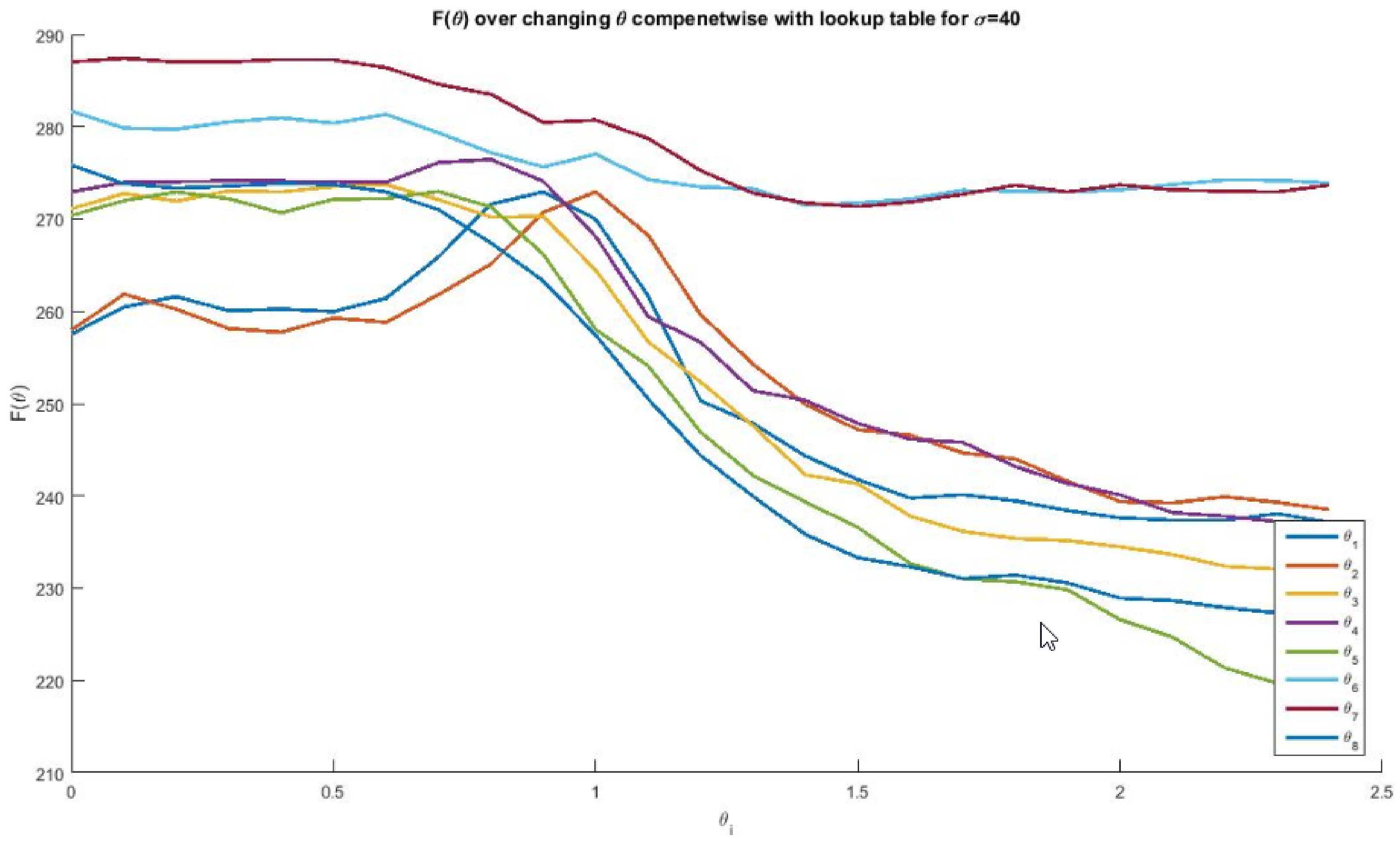
One-dimensional search

Stepsizes for policy search

- Notes:
  - » As we saw before (chapter 5) we trade the elegant simplicity of stochastic gradients for the frustration of designing stepsize rules.
  - » The next few slides suggest that this application is nonconvex, but unimodular.

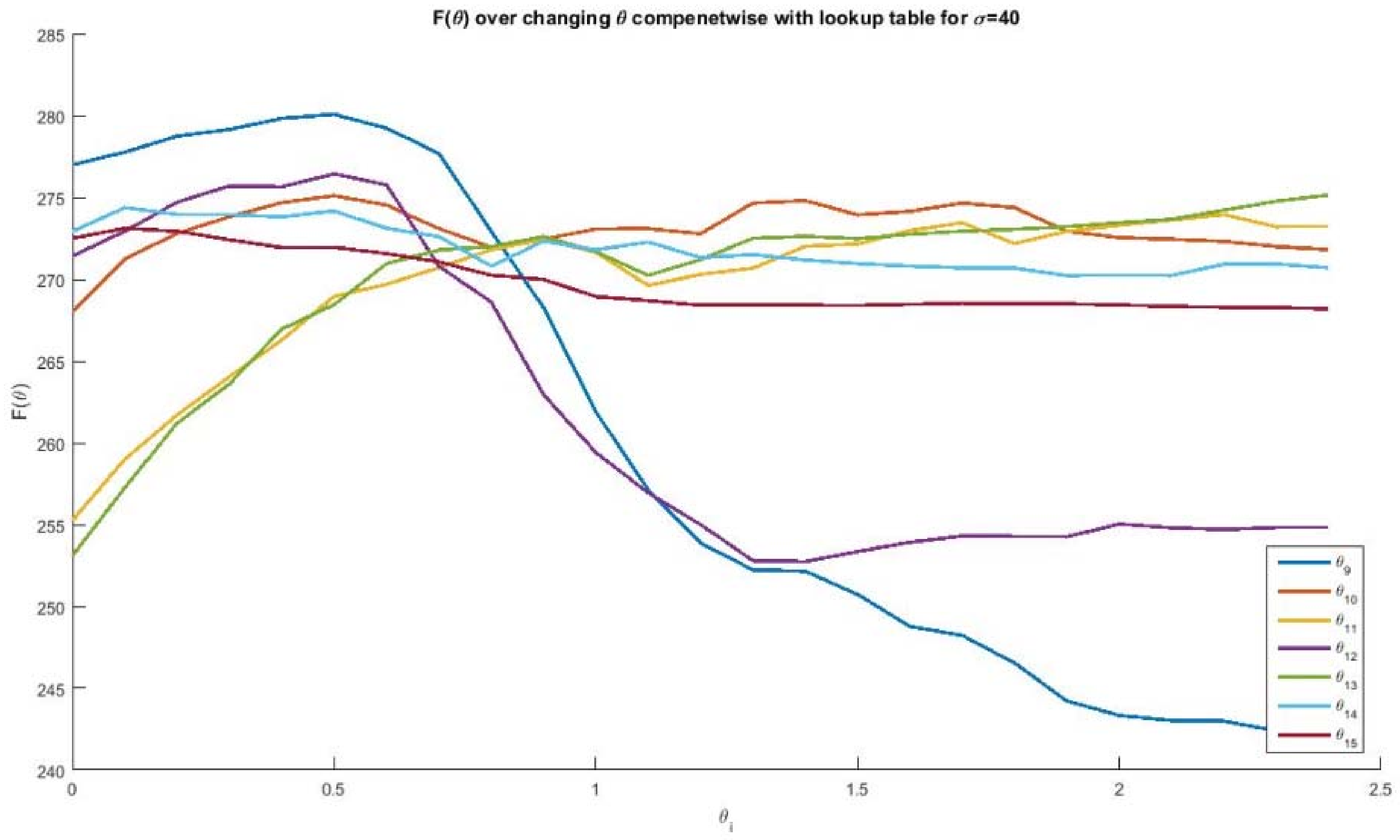
Stepsizes for policy search

- One-dimensional contour plots-uncertain forecast
  - » θ_i for i=1,…, 8 hours into the future.



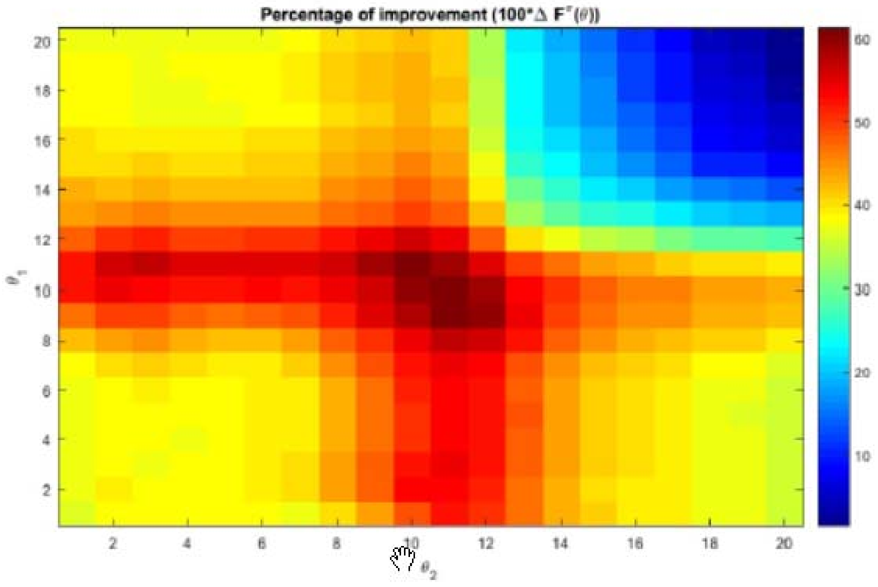
Stepsizes for policy search

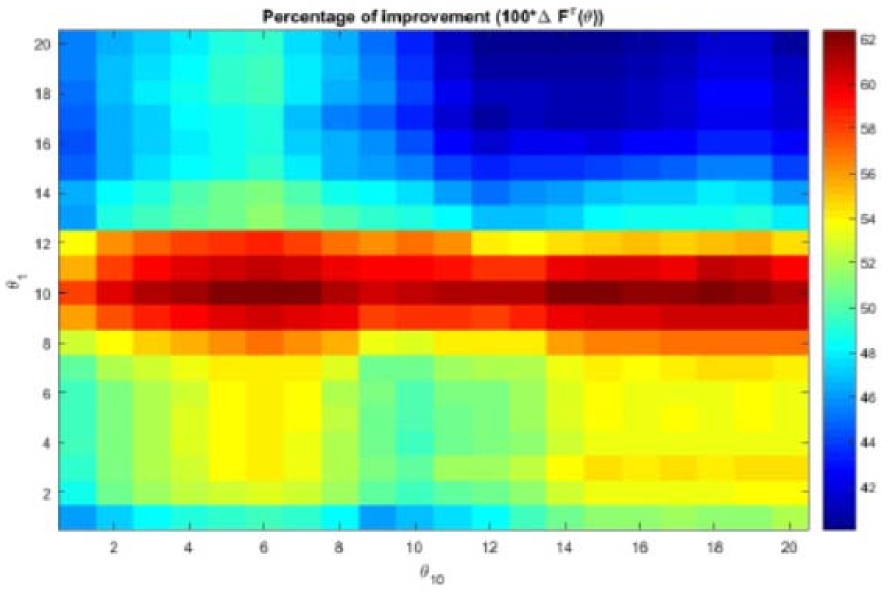
- One-dimensional contour plots-uncertain forecast
  - » θ_i for i=9,…, 15 hours into the future

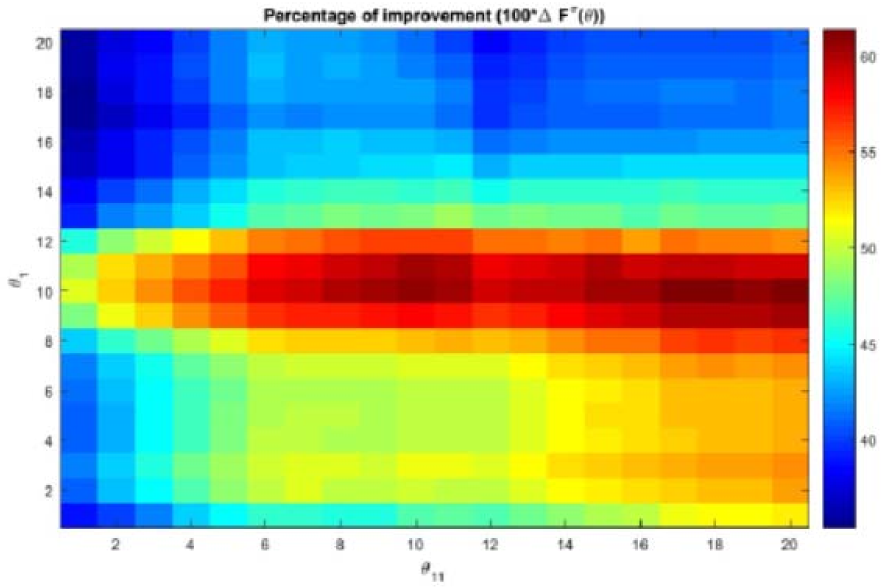


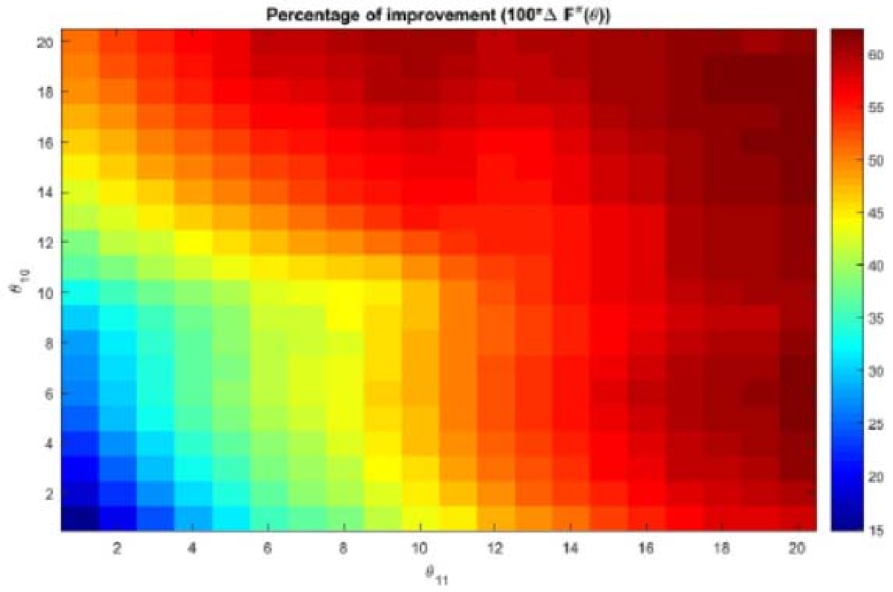
Stepsizes for policy search

- 2-D contours for uncertain forecasts







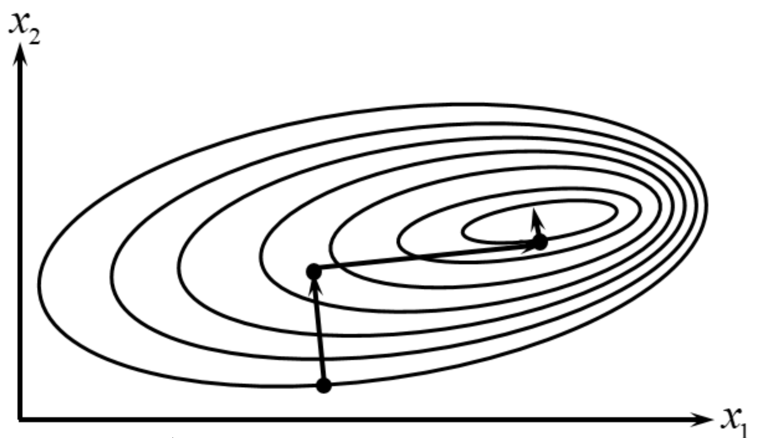


Stepsizes for policy search

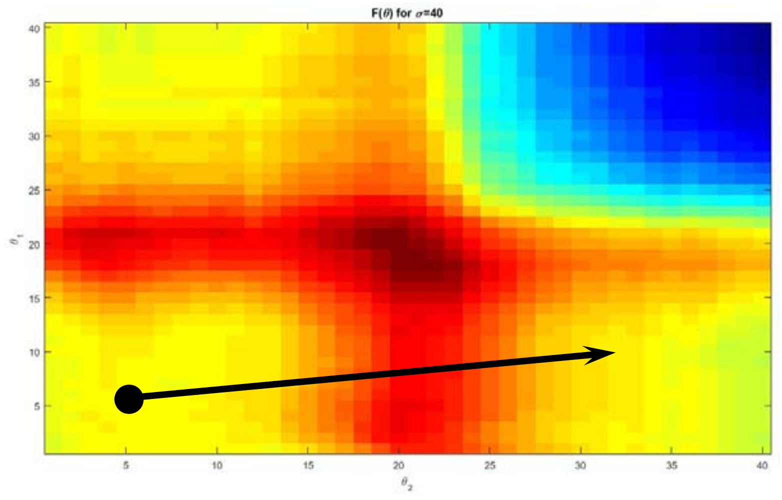
- Stepsizes
  - » With deterministic problems, we perform a one-dimensional search:

$$\min_{\alpha} F(x^n + \alpha \nabla_x F(x^n))$$

  - » With stochastic problems, we use a stepsize rule (or policy) that may miss the optimum completely.
  - » Stepsize rules always have to be tuned, and tuning depends on the starting point.

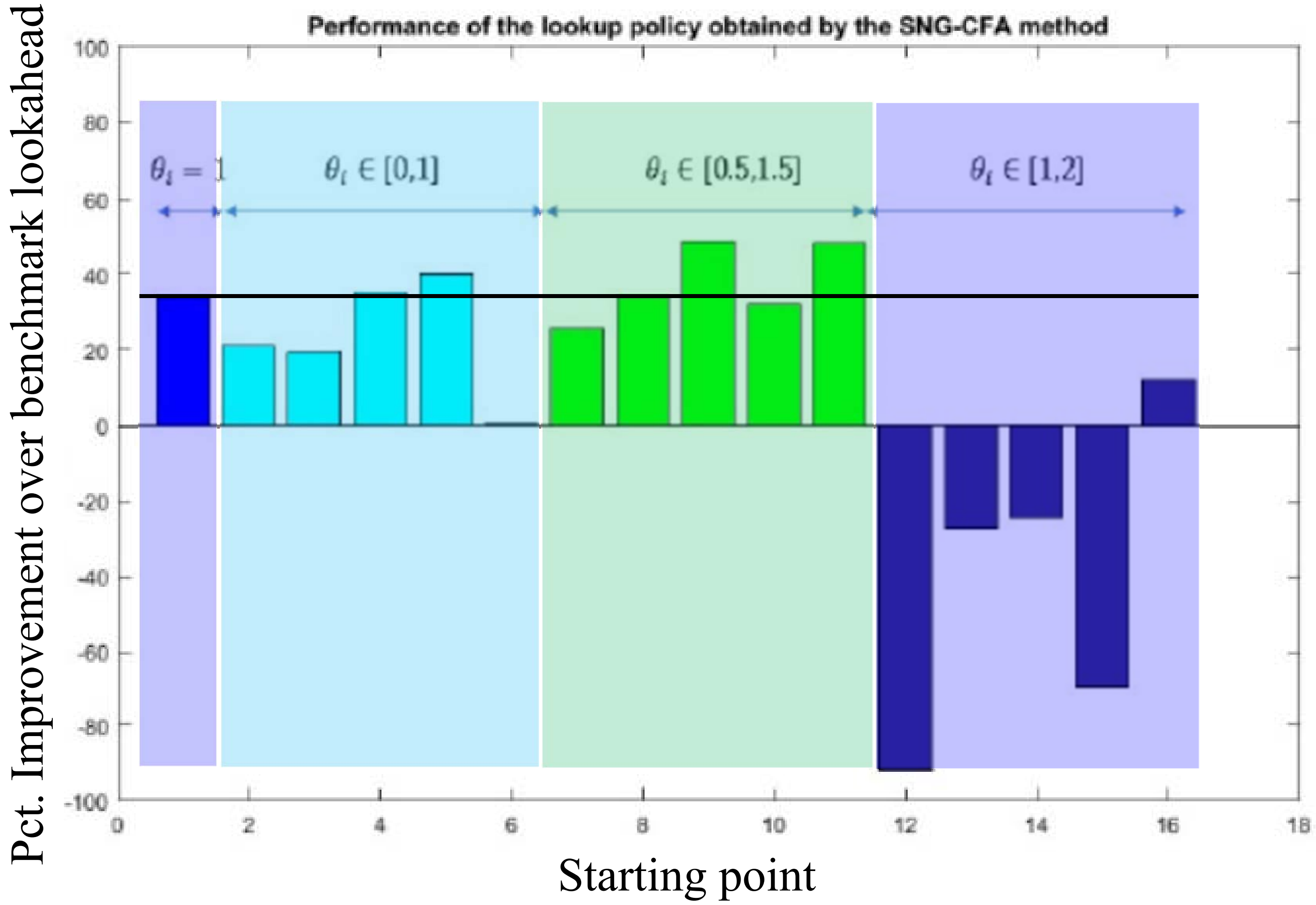


$$x^{n+1} = x^n + \alpha^n \nabla f(x^n)$$



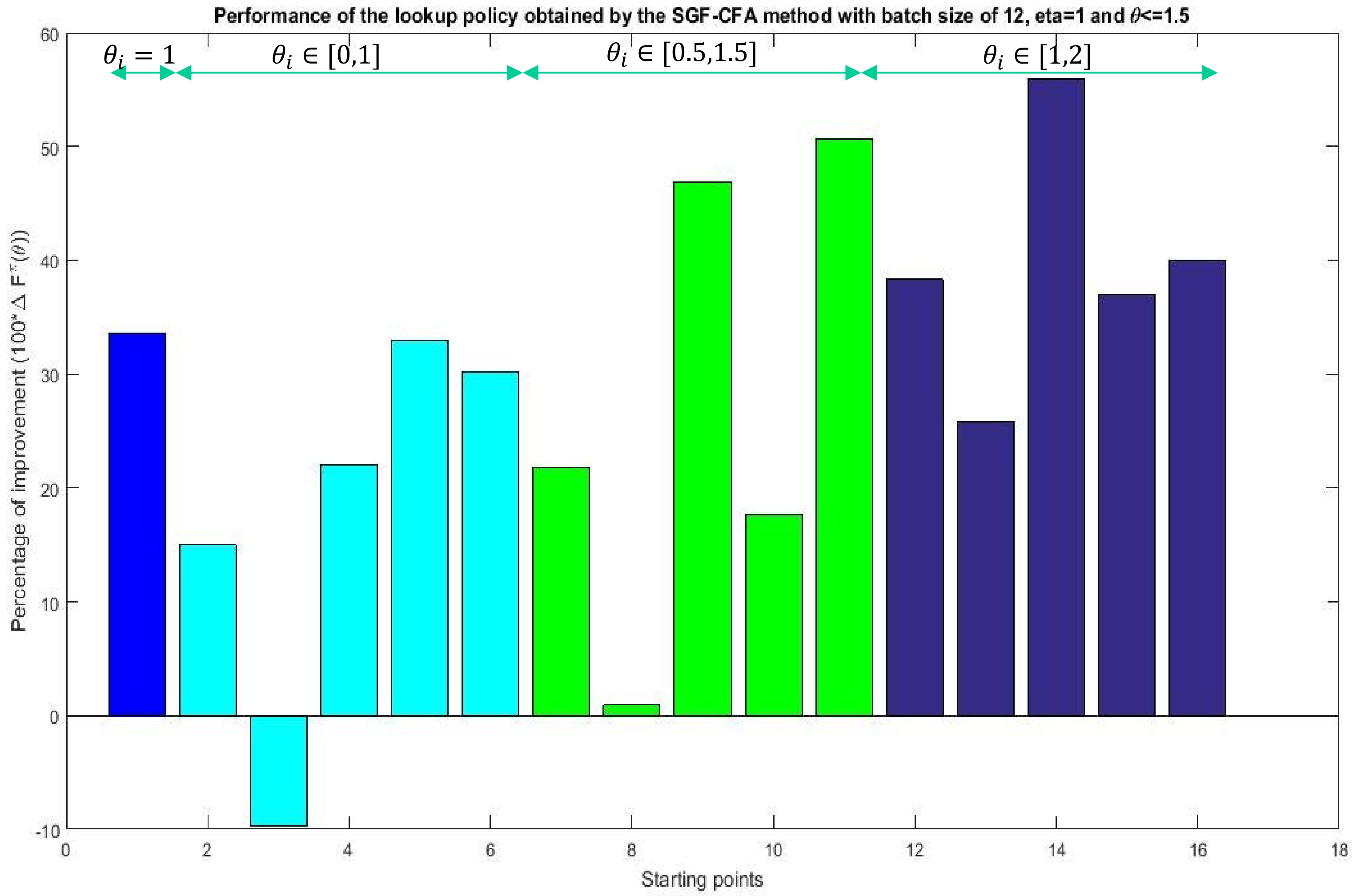
Stepsizes for policy search

- Effect of starting points
  - » Stepsizes tuned for region [0,1].



Stepsizes for policy search

- Tuning the parameters
  - » Stepsizes tuned for region [0,2].



Stepsizes for policy search

- Notes:
  - » The two previous slides illustrate that tuning the stepsize rules does depend on the starting point.
  - » Since stepsize rules for stochastic optimization do not perform a one-dimensional search, tuning is critical.
  - » We have begun experimenting with the idea of using the Fibonacci search, which is an *optimal algorithm* for doing derivative-free search of deterministic, unimodular functions.
  - » We have adapted the Fibonacci search for stochastic problems.

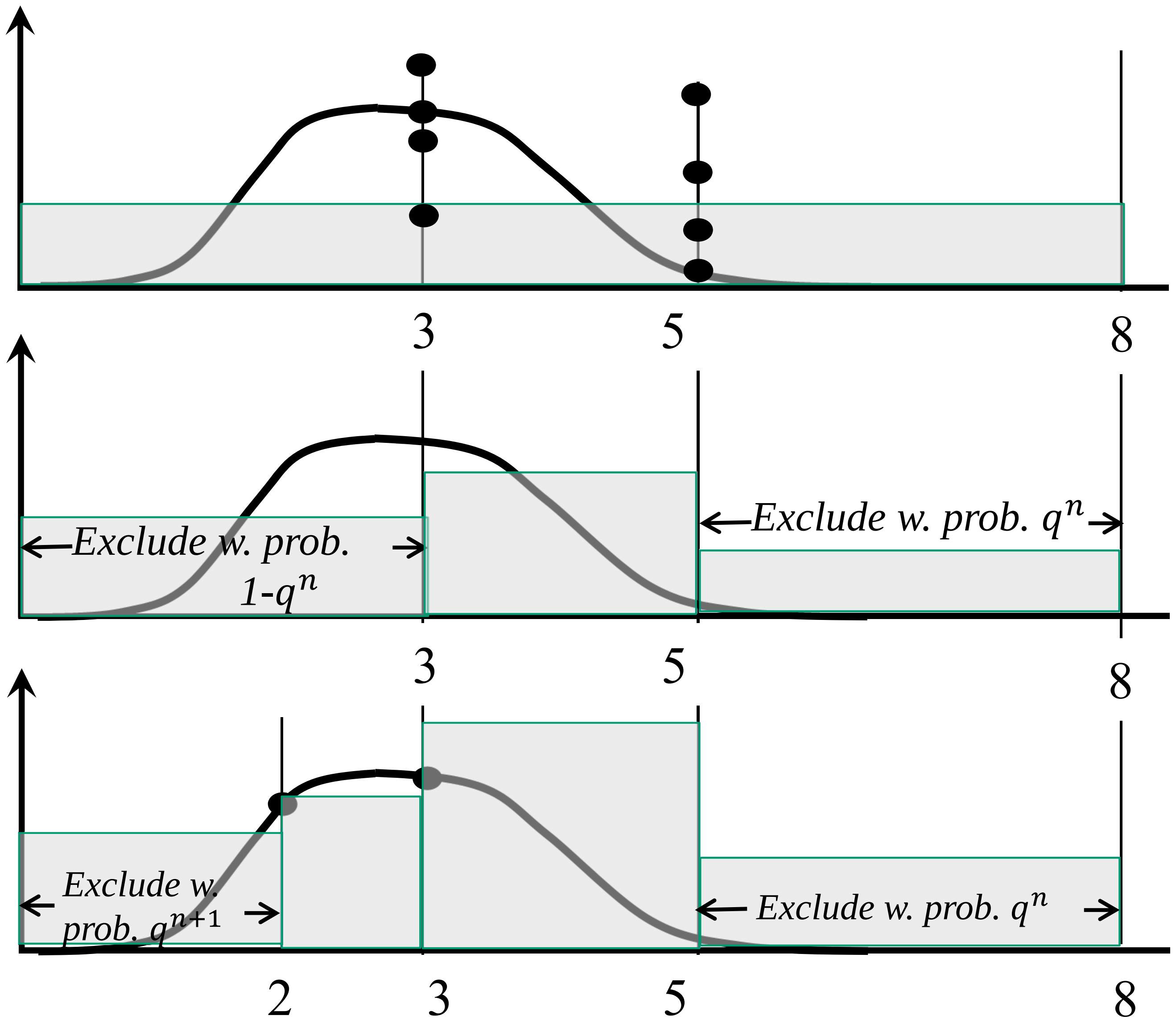
Stepsizes for policy search

- Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13, …

Prior

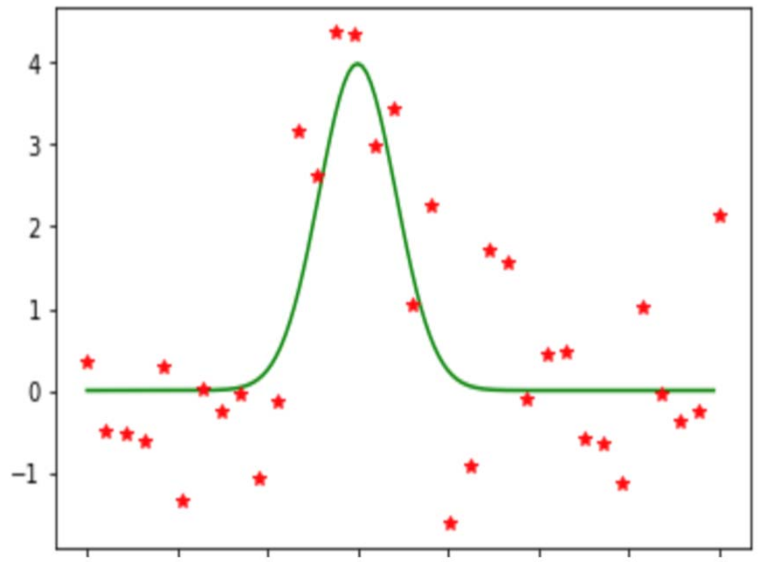
Posterior iter 1

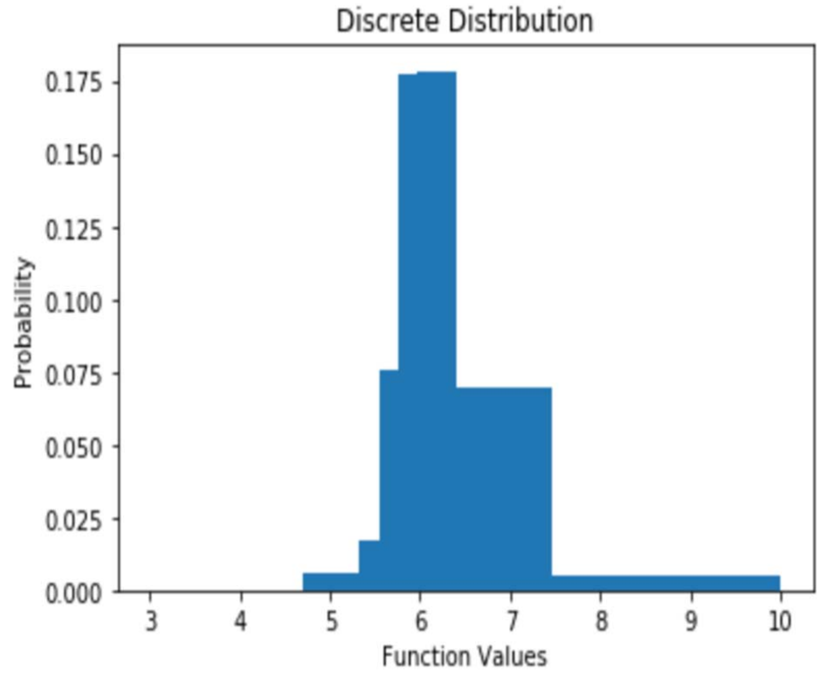
Posterior iter 2



Stepsizes for policy search

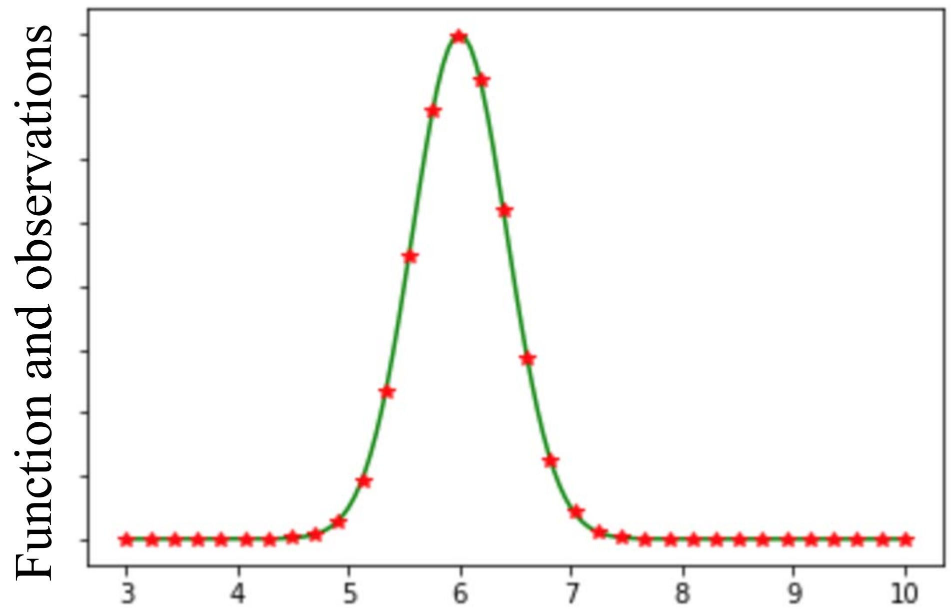
- Fibonacci search
  - » Fibonacci search is an *optimal algorithms* for finding the maximum of a unimodular function.
  - » This process requires that we be able to evaluate the function deterministically.
  - » We assume Lipschitz continuity, as well as knowledge of the error distribution. This allows us to estimate the *probability* that the optimum is toward the left or right.
  - » We then repeat this multiple times, and create a distribution about where the optimum is.

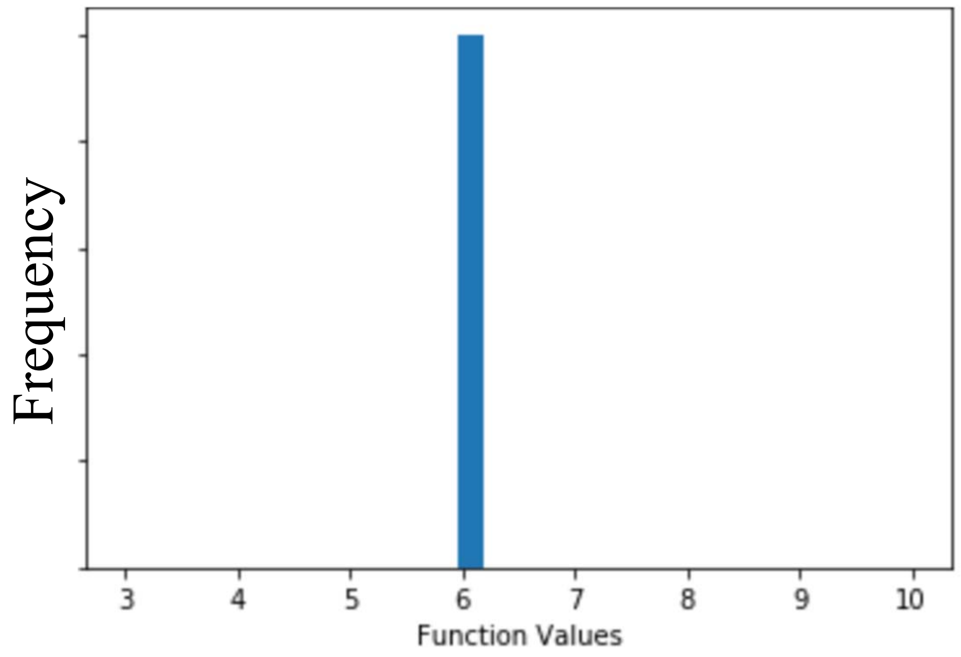




Stepsizes for policy search

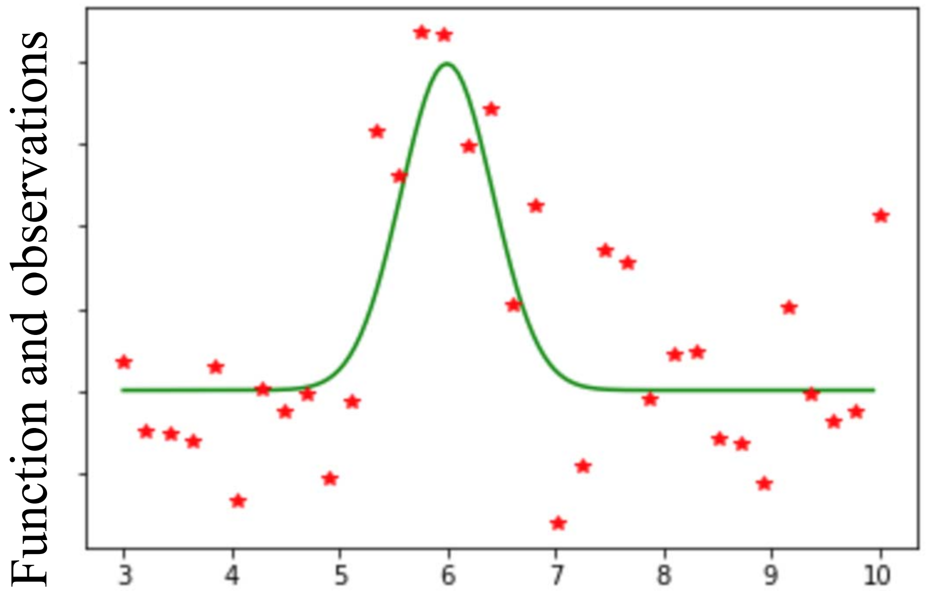
- Fibonacci search for noisy functions
  - 34 Fibonacci numbers
  - Iterations: 1
  - Deterministic response
  - Always finds the correct optimum.

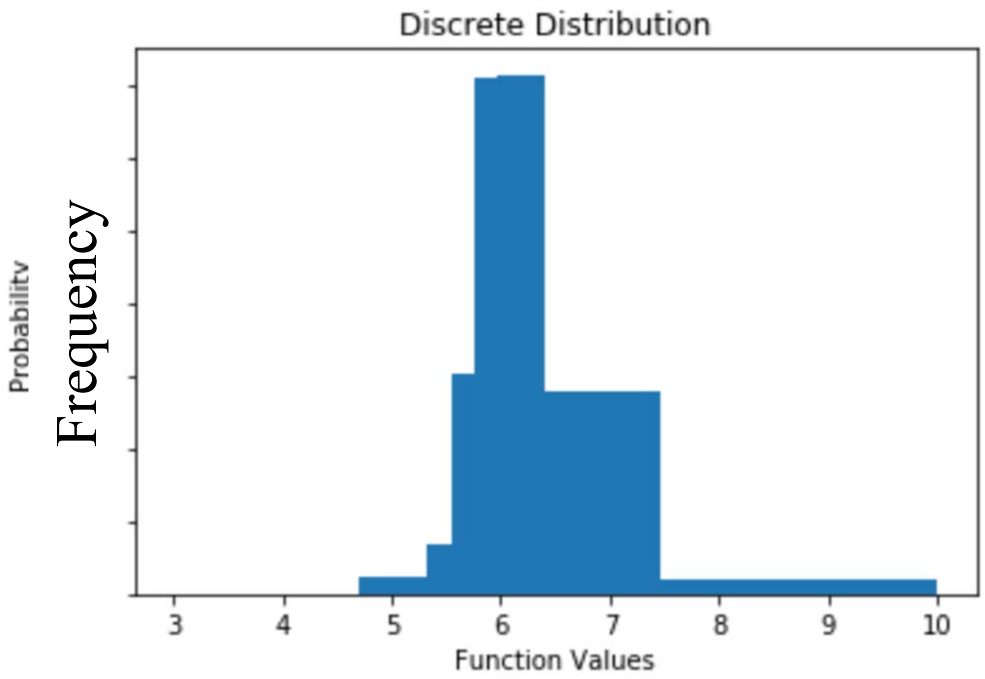




Stepsizes for policy search

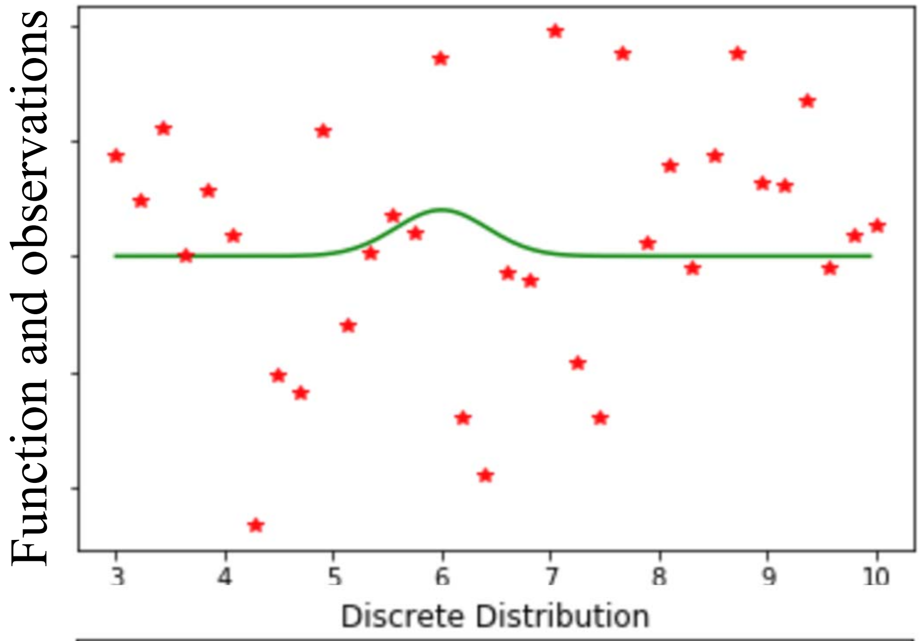
- Fibonacci search for noisy functions
  - » 34 Fibonacci numbers
  - » Iterations: 1
  - » Medium noise

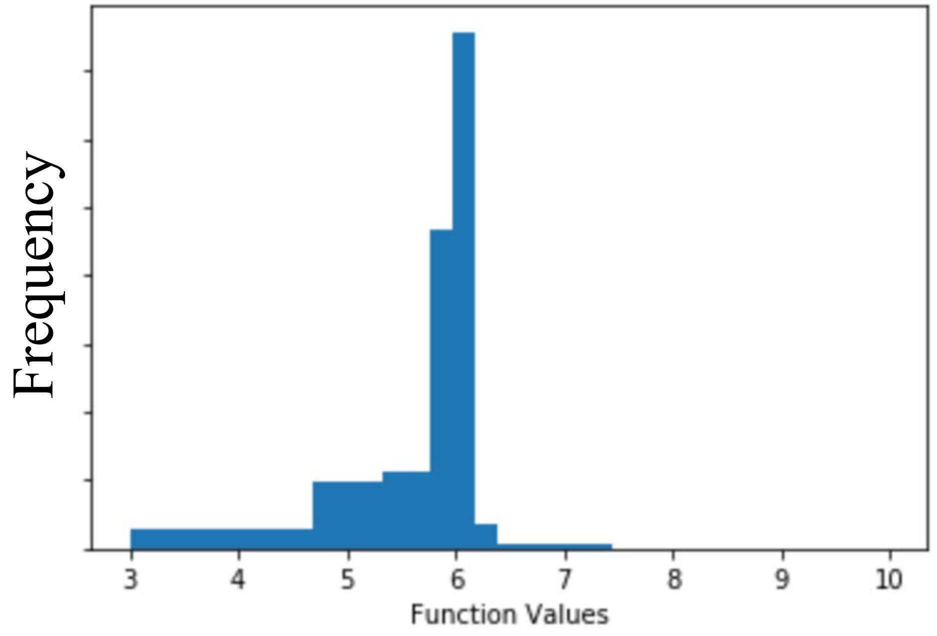




Stepsizes for policy search

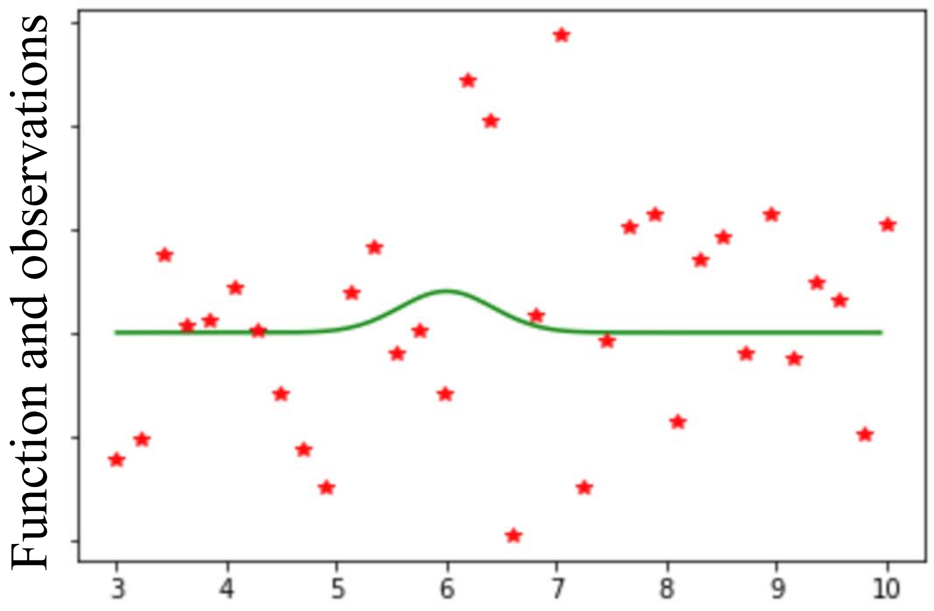
- Fibonacci search for noisy functions
  - » 34 Fibonacci numbers
  - » Iterations: 1
  - » High noise

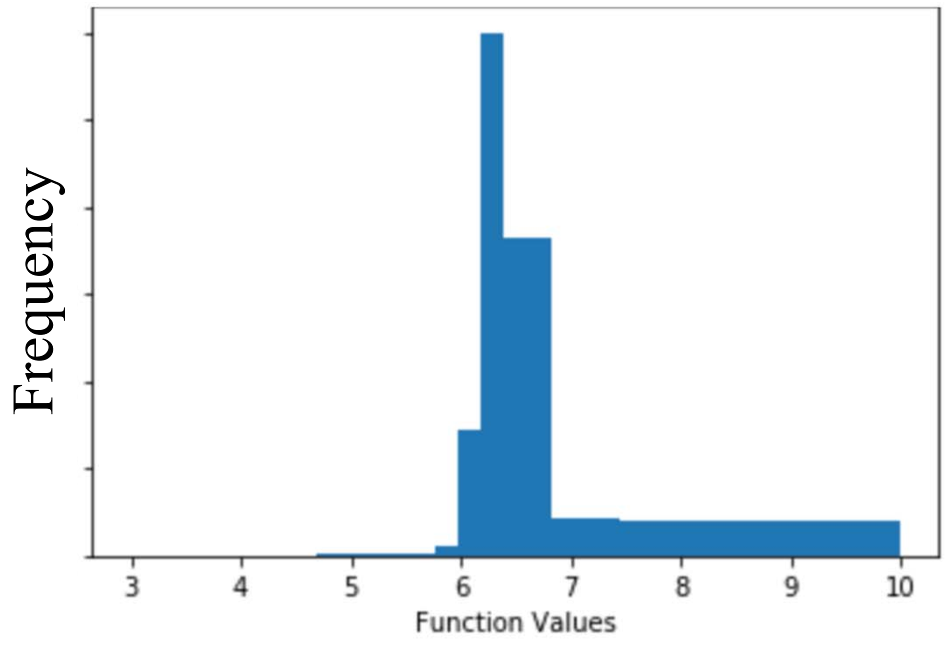




Stepsizes for policy search

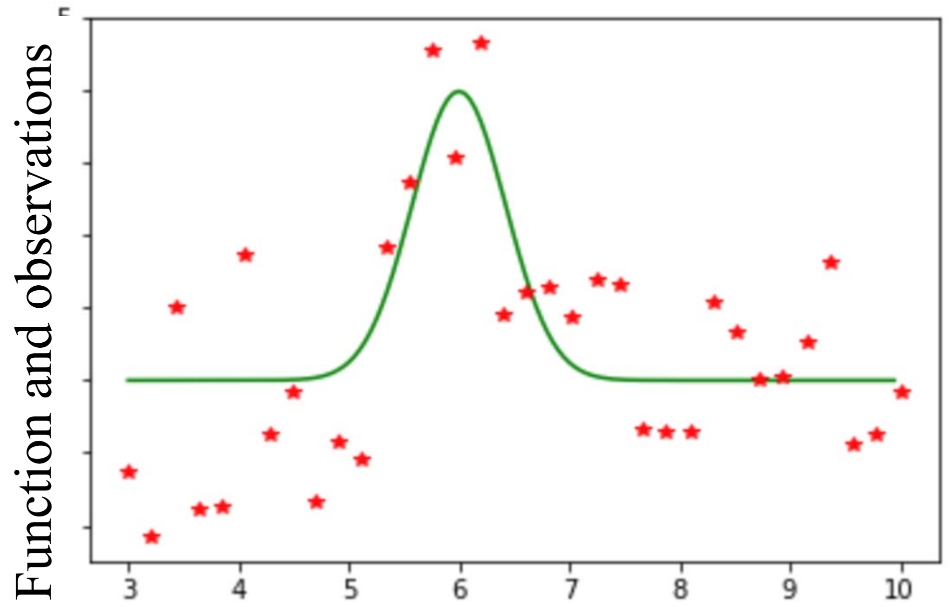
- Fibonacci search for noisy functions
  - » 34 Fibonacci numbers
  - » Iterations: 1
  - » High noise

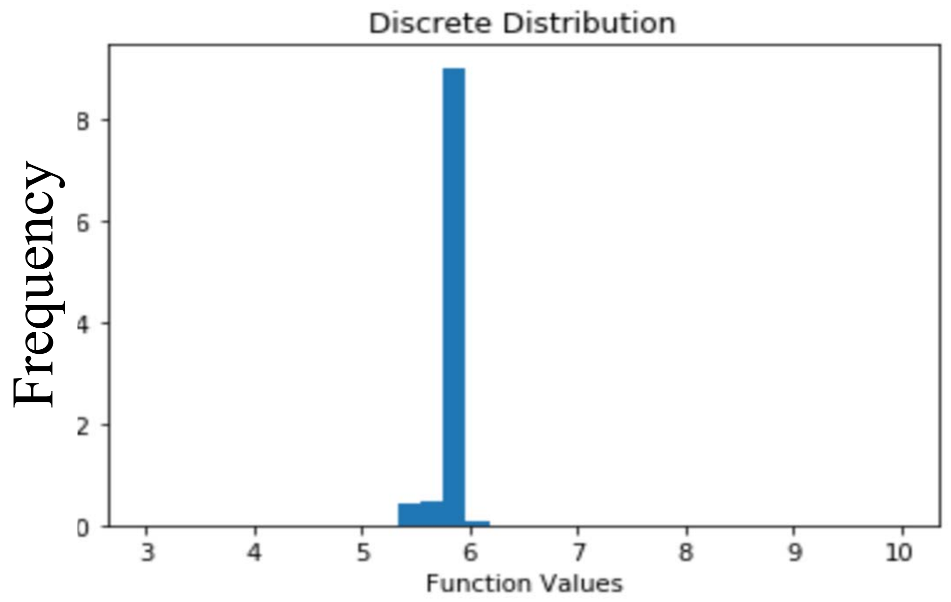




Fibonacci search for noisy functions

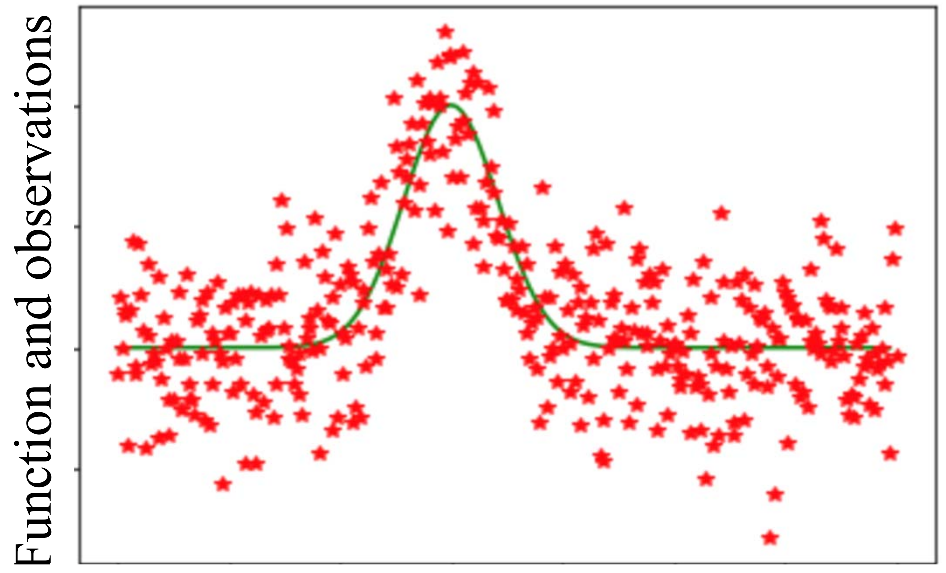
- 34 Fibonacci numbers
- Iterations: 10
- High noise
- Clearly 10 iterations (340 function evaluations) are not needed.

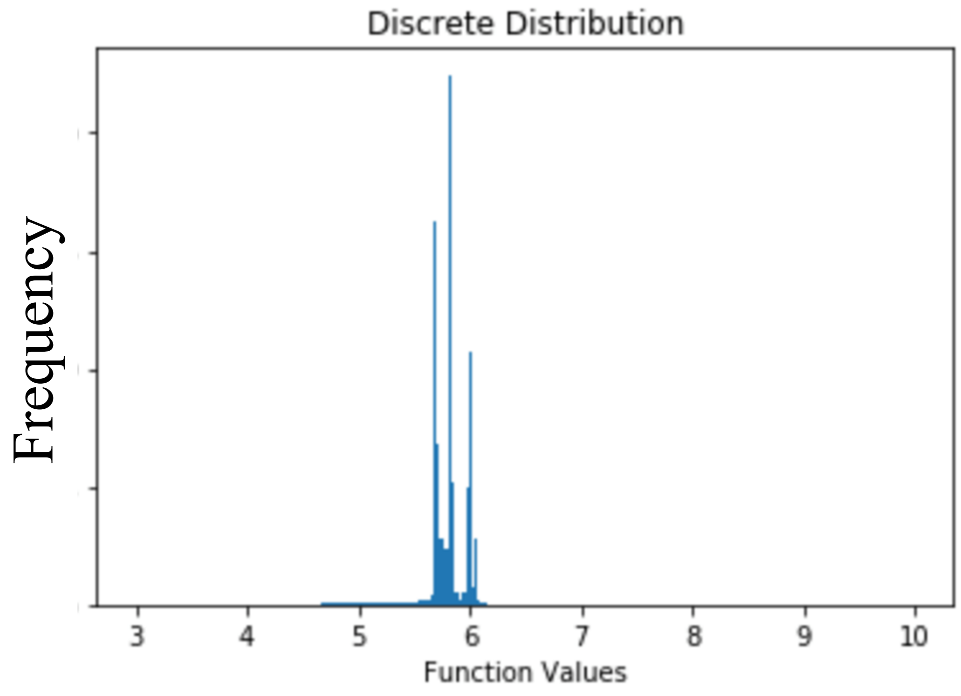




Stepsizes for policy search

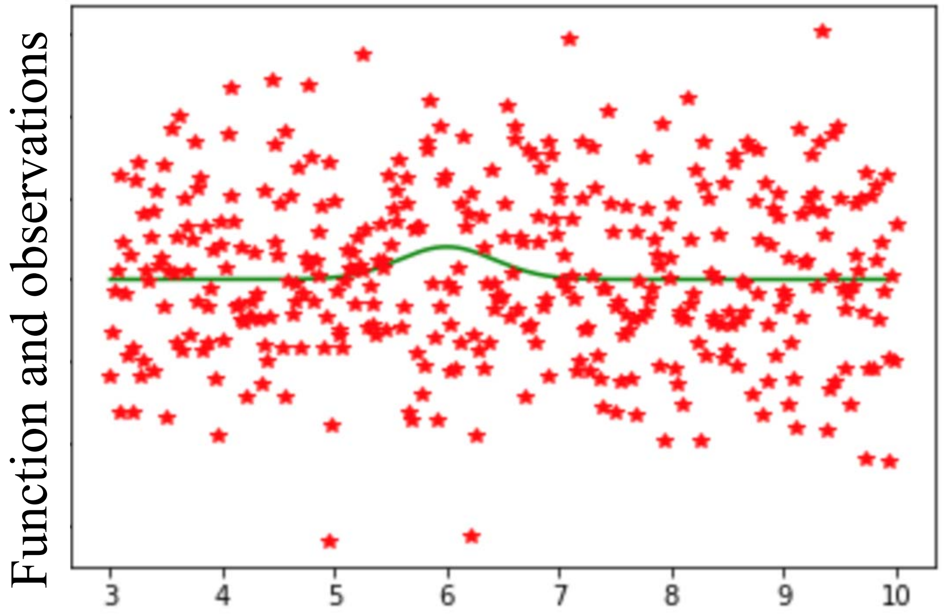
- Fibonacci search for noisy functions
  - » 377 Fibonacci numbers
  - » Iterations: 10
  - » High noise

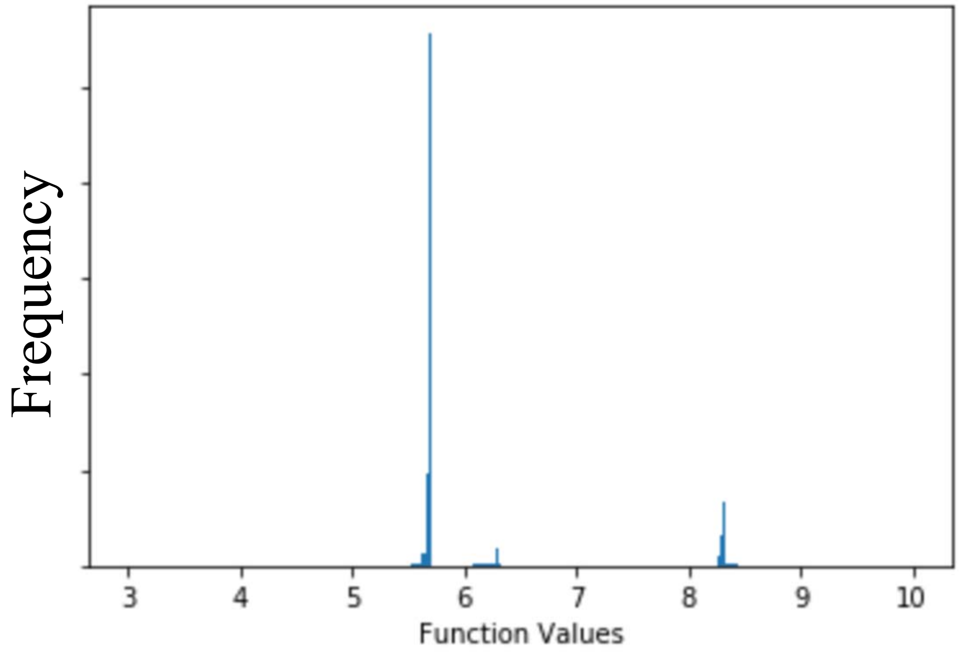




Stepsizes for policy search

- Fibonacci search for noisy functions
  - » 377 Fibonacci numbers
  - » Iterations: 10
  - » High noise





Stepsizes for policy search

- Notes
  - » Even with very high noise, it appears that Fibonacci does a very reliable job of finding a near-optimal point.
  - » The next step is to see how well this works in a stochastic gradient algorithm.

Policy gradient I

Derivative-based:

Categorical actions – Boltzmann policies

Policy search

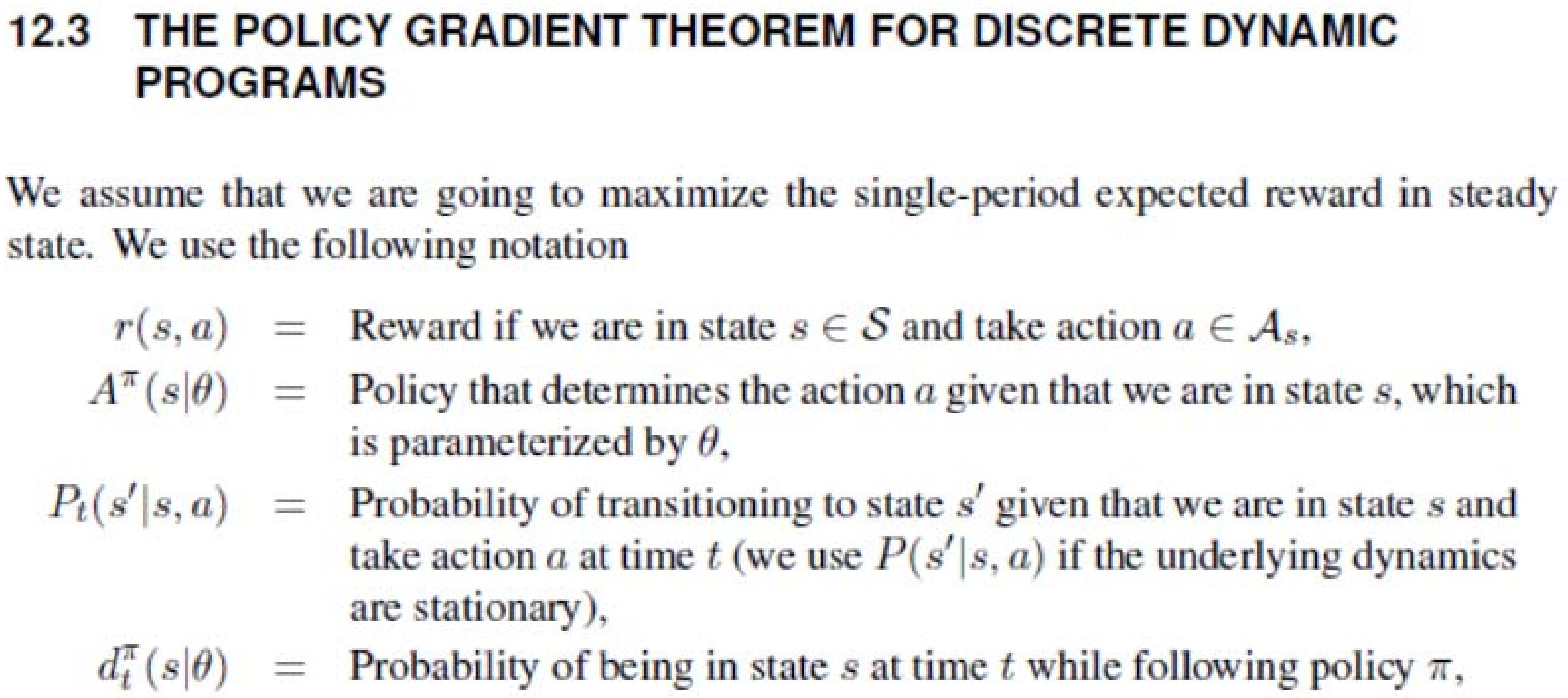
- There are two classes of sequential decision problems that have yielded two different approaches to doing gradient-based policy search:

  » Numerical actions – Here the action is a quantity, and the downstream state is a function of this quantity.

  » Categorical actions – These problems are easily viewed as graphs with possibly stochastic transitions. We are not able to take a derivative with respect to the action.

Policy gradient search

- Discrete dynamic programs
  - » Maximizing expected single period reward



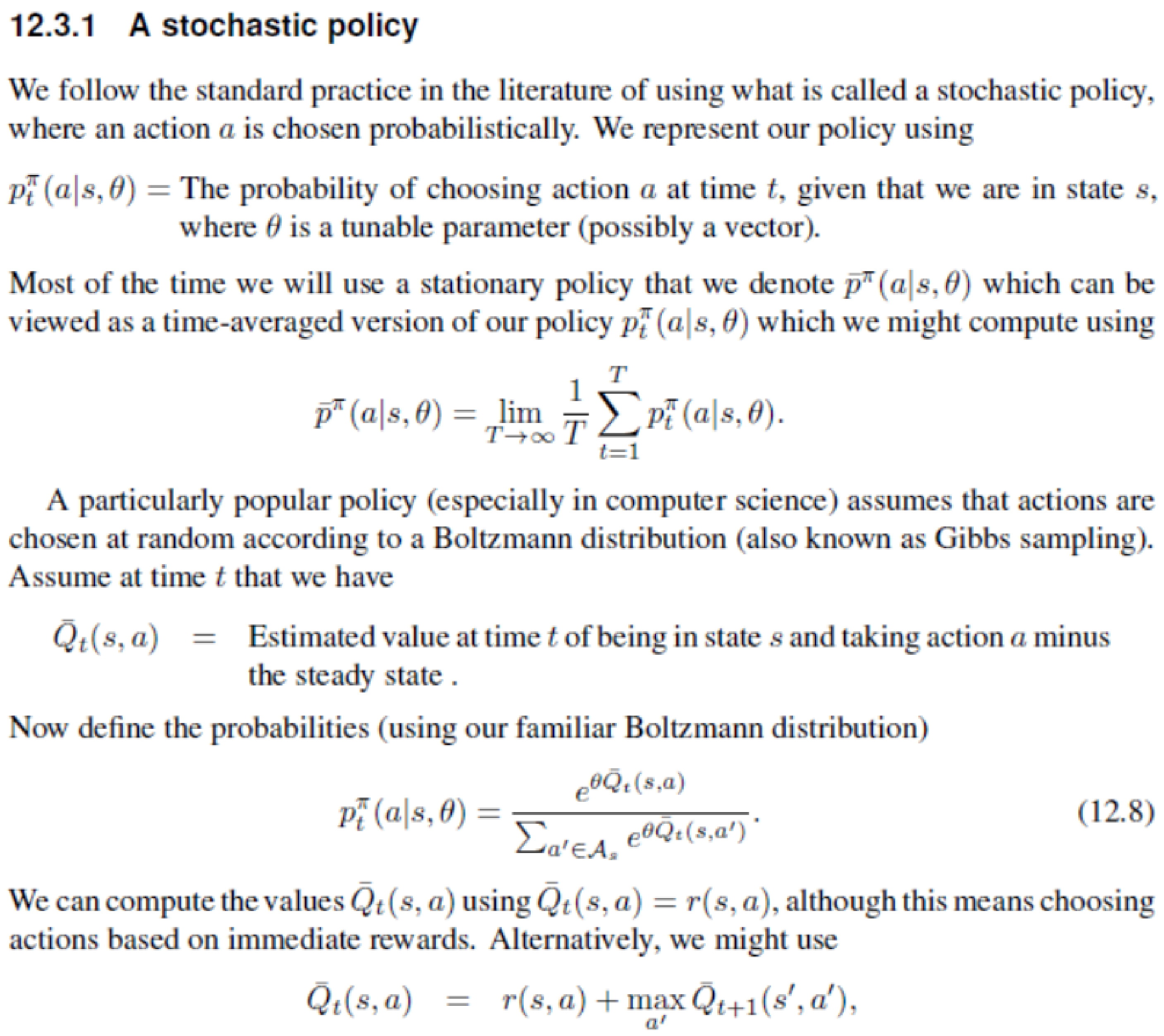
12.3 THE POLICY GRADIENT THEOREM FOR DISCRETE DYNAMIC PROGRAMS

We assume that we are going to maximize the single-period expected reward in steady state. We use the following notation

$r(s,a)$	=	Reward if we are in state $s \in \mathcal{S}$ and take action $a \in \mathcal{A}_s$,
$A^\pi(s\|\theta)$	=	Policy that determines the action a given that we are in state s, which is parameterized by θ,
$P_t(s'\|s,a)$	=	Probability of transitioning to state s' given that we are in state s and take action a at time t (we use $P(s'\|s,a)$ if the underlying dynamics are stationary),
$d_t^\pi(s\|\theta)$	=	Probability of being in state s at time t while following policy π,

  - » Need to skim this – just highlight "policy gradient theorem."

Policy gradient search



12.3.1 A stochastic policy

We follow the standard practice in the literature of using what is called a stochastic policy, where an action a is chosen probabilistically. We represent our policy using

$p_t^\pi(a|s,\theta) =$ The probability of choosing action a at time t, given that we are in state s, where θ is a tunable parameter (possibly a vector).

Most of the time we will use a stationary policy that we denote $\bar{p}^\pi(a|s,\theta)$ which can be viewed as a time-averaged version of our policy $p_t^\pi(a|s,\theta)$ which we might compute using

$$\bar{p}^\pi(a|s,\theta) = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} p_t^\pi(a|s,\theta).$$

A particularly popular policy (especially in computer science) assumes that actions are chosen at random according to a Boltzmann distribution (also known as Gibbs sampling). Assume at time t that we have

$\bar{Q}_t(s,a) =$ Estimated value at time t of being in state s and taking action a minus the steady state .

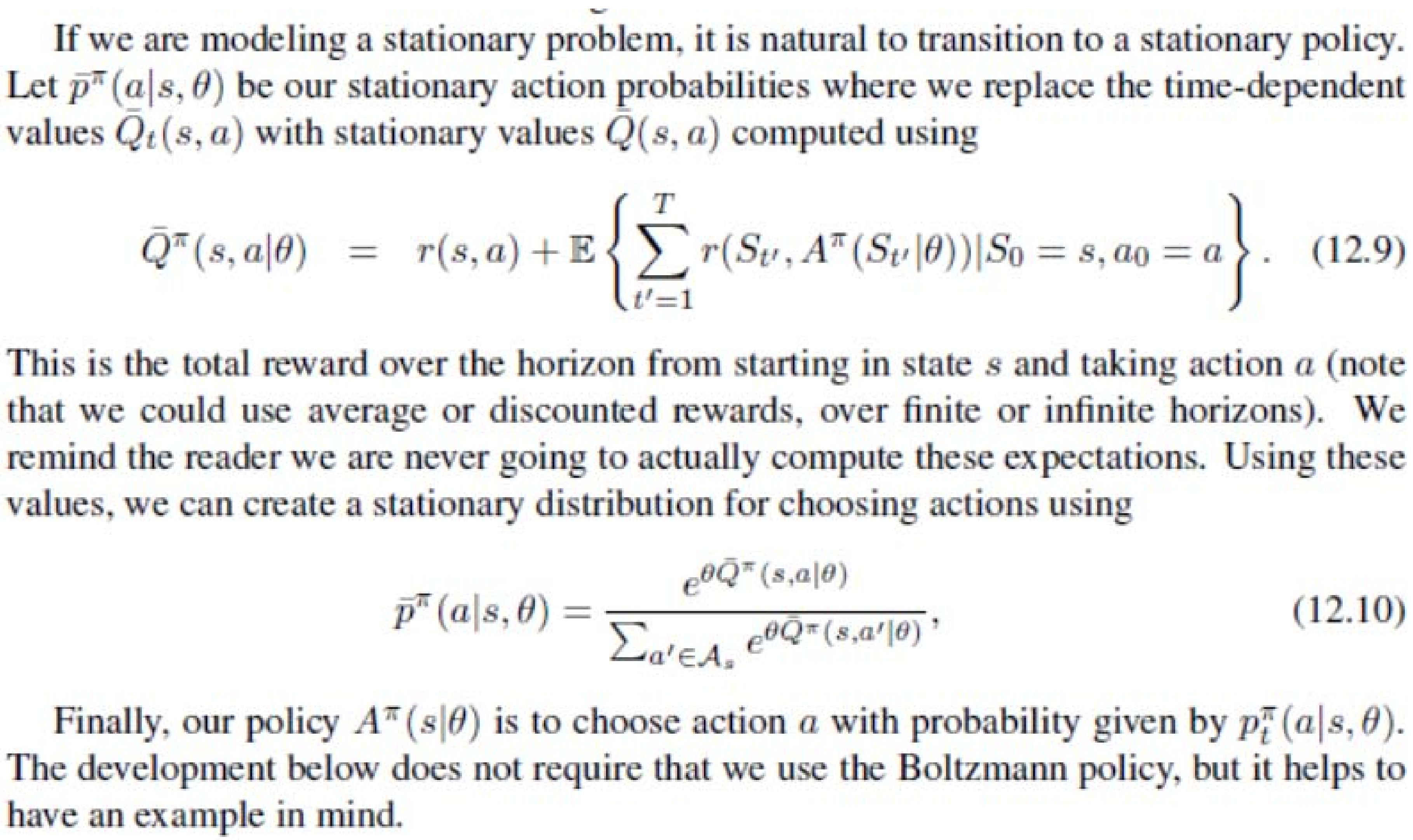
Now define the probabilities (using our familiar Boltzmann distribution)

$$p_t^\pi(a|s,\theta) = \frac{e^{\theta\bar{Q}_t(s,a)}}{\sum_{a'\in\mathcal{A}_s} e^{\theta\bar{Q}_t(s,a')}}. \tag{12.8}$$

We can compute the values $\bar{Q}_t(s,a)$ using $\bar{Q}_t(s,a) = r(s,a)$, although this means choosing actions based on immediate rewards. Alternatively, we might use

$$\bar{Q}_t(s,a) \quad = \quad r(s,a) + \max_{a'} \bar{Q}_{t+1}(s',a'),$$

Policy gradient search



If we are modeling a stationary problem, it is natural to transition to a stationary policy. Let $\bar{p}^{\pi}(a|s,\theta)$ be our stationary action probabilities where we replace the time-dependent values $\bar{Q}_t(s,a)$ with stationary values $\bar{Q}(s,a)$ computed using

$$\bar{Q}^{\pi}(s,a|\theta) \quad = \quad r(s,a) + \mathbb{E}\left\{\sum_{t'=1}^{T} r(S_{t'}, A^{\pi}(S_{t'}|\theta))|S_0 = s, a_0 = a\right\}. \qquad (12.9)$$

This is the total reward over the horizon from starting in state s and taking action a (note that we could use average or discounted rewards, over finite or infinite horizons). We remind the reader we are never going to actually compute these expectations. Using these values, we can create a stationary distribution for choosing actions using

$$\bar{p}^{\pi}(a|s,\theta) = \frac{e^{\theta\bar{Q}^{\pi}(s,a|\theta)}}{\sum_{a'\in\mathcal{A}_s} e^{\theta\bar{Q}^{\pi}(s,a'|\theta)}}, \qquad (12.10)$$

Finally, our policy $A^{\pi}(s|\theta)$ is to choose action a with probability given by $p_t^{\pi}(a|s,\theta)$. The development below does not require that we use the Boltzmann policy, but it helps to have an example in mind.

Policy gradient search

12.3.2 The objective function

To develop the gradient, we have to start by writing out our objective function which is to maximize the average reward over time, given by

$$F^\pi(\theta) = \lim_{T\to\infty} \frac{1}{T} \left\{ \sum_{t=0}^{T} \sum_{s\in\mathcal{S}} \left(d_t^\pi(s|\theta) \sum_{a\in\mathcal{A}_s} r(s,a) p_t^\pi(a|s,\theta) \right) \right\}. \qquad (12.11)$$

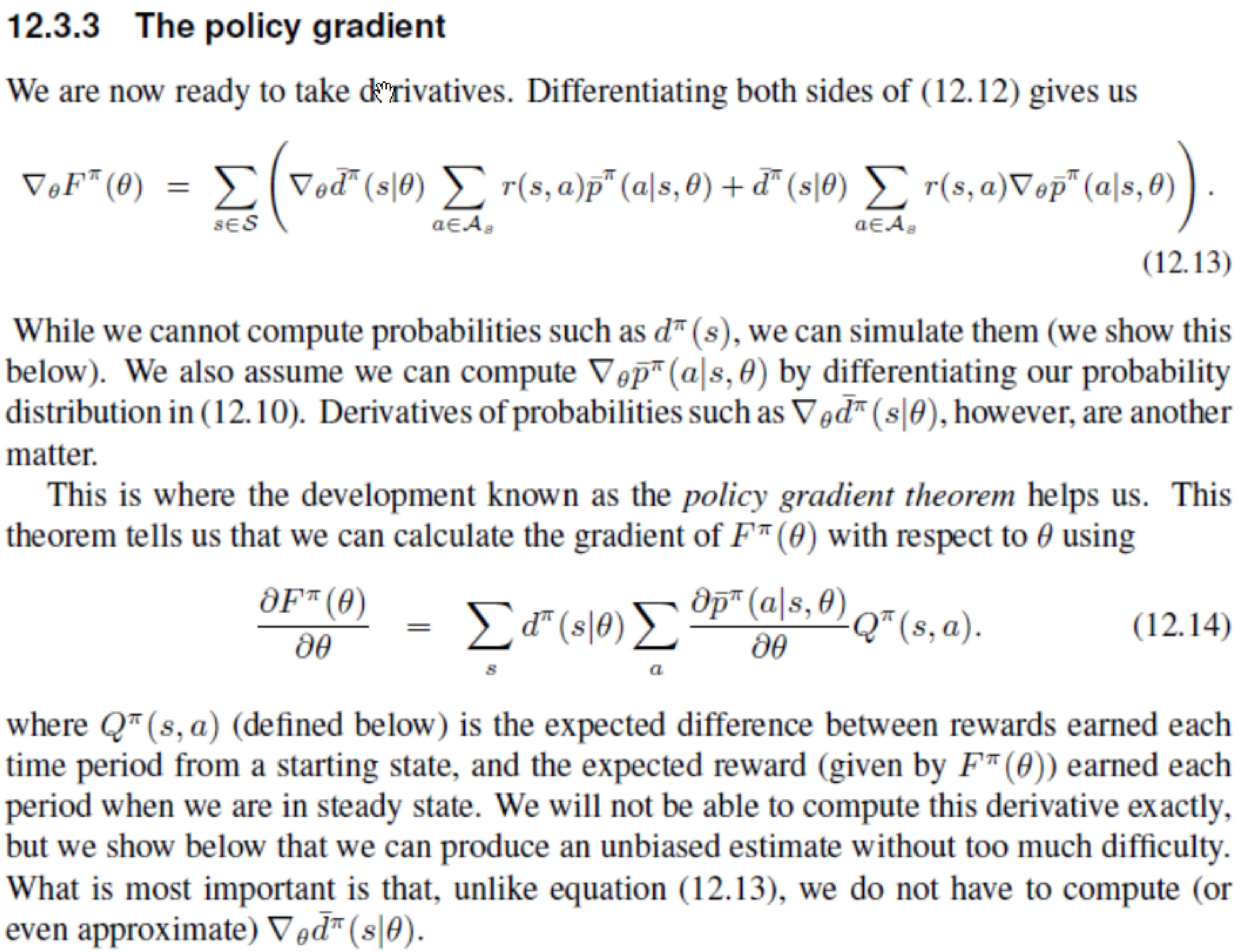
A more compact form involves replacing the time-dependent state probabilities with their time averages (since we are taking the limit). Let

$$\bar{d}^\pi(s|\theta) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} d_t^\pi(s|\theta).$$

We can then write our average reward per time period as

$$F^\pi(\theta) = \sum_{s\in\mathcal{S}} \bar{d}^\pi(s|\theta) \sum_{a\in\mathcal{A}_s} r(s,a)\bar{p}^\pi(a|s,\theta). \qquad (12.12)$$

Policy gradient search



12.3.3 The policy gradient

We are now ready to take derivatives. Differentiating both sides of (12.12) gives us

$$\nabla_\theta F^\pi(\theta) = \sum_{s\in\mathcal{S}} \left(\nabla_\theta \bar{d}^\pi(s|\theta) \sum_{a\in\mathcal{A}_s} r(s,a)\bar{p}^\pi(a|s,\theta) + \bar{d}^\pi(s|\theta) \sum_{a\in\mathcal{A}_s} r(s,a)\nabla_\theta \bar{p}^\pi(a|s,\theta) \right). \tag{12.13}$$

While we cannot compute probabilities such as $d^\pi(s)$, we can simulate them (we show this below). We also assume we can compute $\nabla_\theta \bar{p}^\pi(a|s,\theta)$ by differentiating our probability distribution in (12.10). Derivatives of probabilities such as $\nabla_\theta \bar{d}^\pi(s|\theta)$, however, are another matter.

This is where the development known as the *policy gradient theorem* helps us. This theorem tells us that we can calculate the gradient of $F^\pi(\theta)$ with respect to θ using

$$\frac{\partial F^\pi(\theta)}{\partial\theta} = \sum_s d^\pi(s|\theta) \sum_a \frac{\partial \bar{p}^\pi(a|s,\theta)}{\partial\theta} Q^\pi(s,a). \tag{12.14}$$

where $Q^\pi(s,a)$ (defined below) is the expected difference between rewards earned each time period from a starting state, and the expected reward (given by $F^\pi(\theta)$) earned each period when we are in steady state. We will not be able to compute this derivative exactly, but we show below that we can produce an unbiased estimate without too much difficulty. What is most important is that, unlike equation (12.13), we do not have to compute (or even approximate) $\nabla_\theta \bar{d}^\pi(s|\theta)$.

Policy gradient search

We are going to begin by defining two important quantities:

$$
\begin{aligned}
Q^\pi(s,a|\theta) &= \sum_{t=1}^{\infty} \mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s, a_0 = a\}, \\
V^\pi(s|\theta) &= \sum_{t=1}^{\infty} \mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s\}, \\
&= \sum_{a\in\mathcal{A}} \bar{p}^\pi(a_0 = a|s,\theta) \sum_{t=1}^{\infty} \mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s, a_0 = a\}, \\
&= \sum_{a} \bar{p}^\pi(a|s,\theta) Q^\pi(s,a). \qquad (12.15)
\end{aligned}
$$

Note that $Q^\pi(s,a|\theta)$ is quite different than the quantities $\bar{Q}^\pi(s,a|\theta)$ used above for the Boltzmann policy (which is consistent with Q-learning, which we first saw in section 2.1.10). $Q^\pi(s,a|\theta)$ sums the difference between the reward each period and the steady state reward per period (a difference that goes to zero on average), given that we start in state s and initially take action a. $V^\pi(s|\theta)$ is simply the expectation over all initial actions actions a as specified by our probabilistic policy

We next rewrite $Q^{\pi}(s,a)$ as the first term in the summation, plus the expected value of the remainder of the infinite sum using

$$\begin{aligned} Q^{\pi}(s,a) &= \sum_{t=1}^{\infty} \mathbb{E}\{r_t - F^{\pi}(\theta) | s_0 = s, a_0 = a\}, \\ &= r(s,a) - F^{\pi}(\theta) + \sum_{s'} P(s'|s,a) V^{\pi}(s'), \quad \forall s,\ a, \qquad (12.16) \end{aligned}$$

where $P(s'|s,a)$ is the one-step transition matrix (recall that this does not depend on θ). Solving for $F^{\pi}(\theta)$ gives

$$F^{\pi}(\theta) = r(s,a) + \sum_{s'} P(s'|s,a) V^{\pi}(s') - Q^{\pi}(s,a). \qquad (12.17)$$

Now, note that $F^{\pi}(\theta)$ is not a function of either s or a, even though they both appear in the right hand side of (12.17). Noting that since our policy must pick some action, $\sum_{a \in \mathcal{A}} \bar{p}^{\pi}(a|s,\theta) = 1$, which means

$$\sum_{a \in \mathcal{A}} \bar{p}^{\pi}(a|s,\theta) F^{\pi}(\theta) = F^{\pi}(\theta), \quad \forall a.$$

This means we can take the expectation of (12.17) over all actions, giving us

$$F^{\pi}(\theta) = \sum_{a} \bar{p}^{\pi}(a|s,\theta) \left(r(s,a) + \sum_{s'} P(s'|s,a) V^{\pi}(s') - Q^{\pi}(s,a) \right), \quad \forall s. \ (12.18)$$

Again we note that (12.18) is true for all states s. We can now take derivatives using the following steps (explanations follow the equations):

Policy gradient search

$$\frac{\partial F^\pi(\theta)}{\partial\theta} = \frac{\partial}{\partial\theta}\left(\sum_a \bar{p}^\pi(a|s,\theta)\left(r(s,a)+\sum_{s'}P(s'|s,a)V^\pi(s')-Q^\pi(s,a)\right)\right) \tag{12.19}$$

$$= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}r(s,a)+\sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}\sum_{s'}P(s'|s,a)V^\pi(s')$$
$$+\sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'}P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta}-\frac{\partial}{\partial\theta}\left(\sum_a \bar{p}^\pi(a|s,\theta)Q^\pi(s,a)\right) \tag{12.20}$$

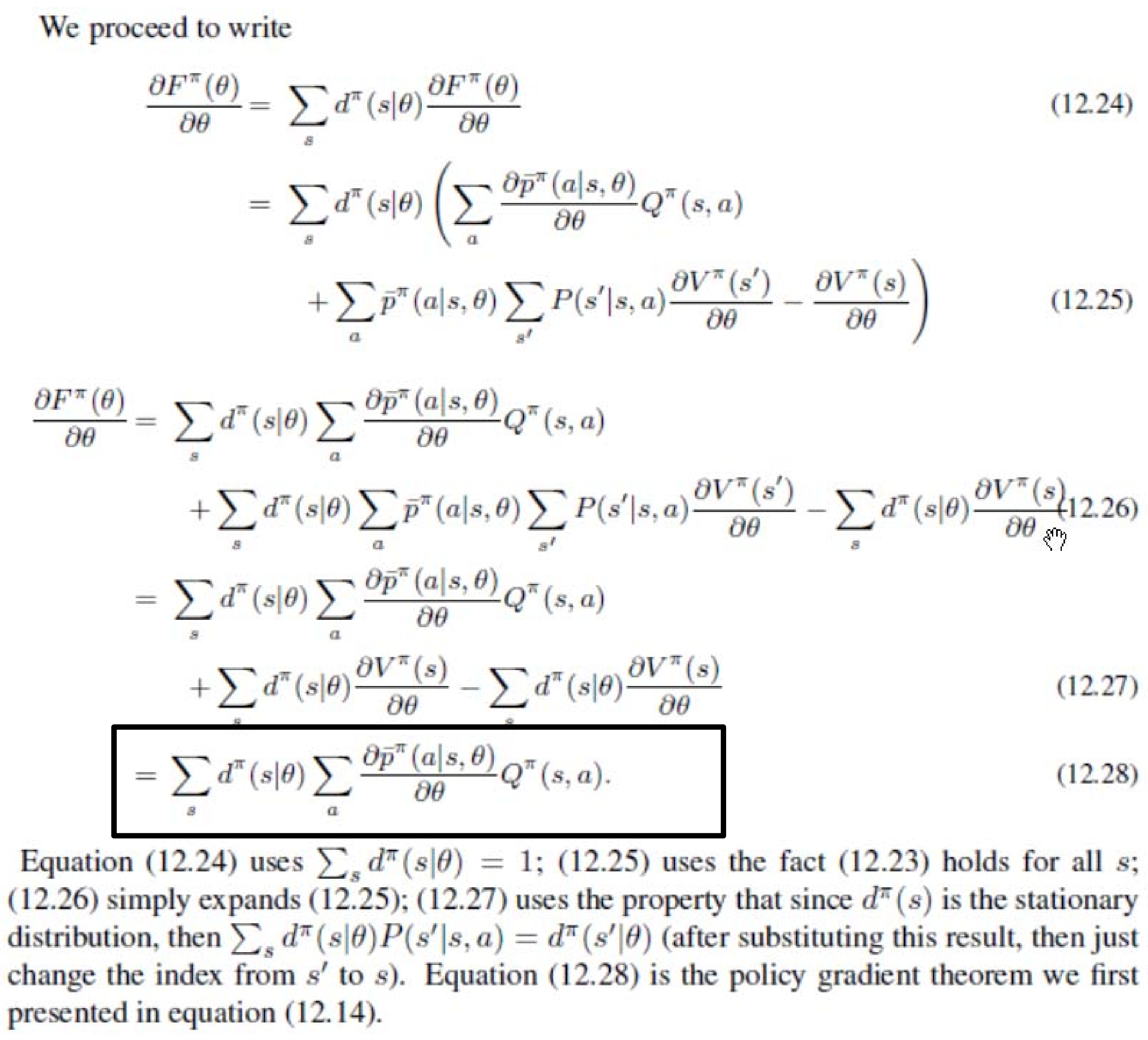
$$= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}\left(r(s,a)+\sum_{s'}P(s'|s,a)V^\pi(s')\right)$$
$$+\sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'}P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta}-\frac{\partial V^\pi(s)}{\partial\theta} \tag{12.21}$$

$$= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}\left(Q^\pi(s,a)+F^\pi(\theta)\right)$$
$$+\sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'}P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta}-\frac{\partial V^\pi(s)}{\partial\theta} \tag{12.22}$$

$$= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a)+\sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'}P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta}-\frac{\partial V^\pi(s)}{\partial\theta}. \tag{12.23}$$

Equation (12.19) is from (12.18); (12.20) is the direct expansion of (12.19), where two terms vanish because $r(s,a)$ and $P(s'|s,a)$ do not depend on the policy $\bar{p}^\pi(a|s,\theta)$; (12.19) uses (12.15) for the last term; (12.22) uses (12.16); (12.15) uses the fact $F^\pi(\theta)$ is constant over states and actions, and $\sum_a \bar{p}^\pi(a|s,\theta)=1$. Finally, note that equation (12.23) is true for all states.

Policy gradient search



We proceed to write

$$\frac{\partial F^{\pi}(\theta)}{\partial \theta} = \sum_s d^{\pi}(s|\theta) \frac{\partial F^{\pi}(\theta)}{\partial \theta} \tag{12.24}$$

$$= \sum_s d^{\pi}(s|\theta) \left(\sum_a \frac{\partial \bar{p}^{\pi}(a|s,\theta)}{\partial \theta} Q^{\pi}(s,a) + \sum_a \bar{p}^{\pi}(a|s,\theta) \sum_{s'} P(s'|s,a) \frac{\partial V^{\pi}(s')}{\partial \theta} - \frac{\partial V^{\pi}(s)}{\partial \theta} \right) \tag{12.25}$$

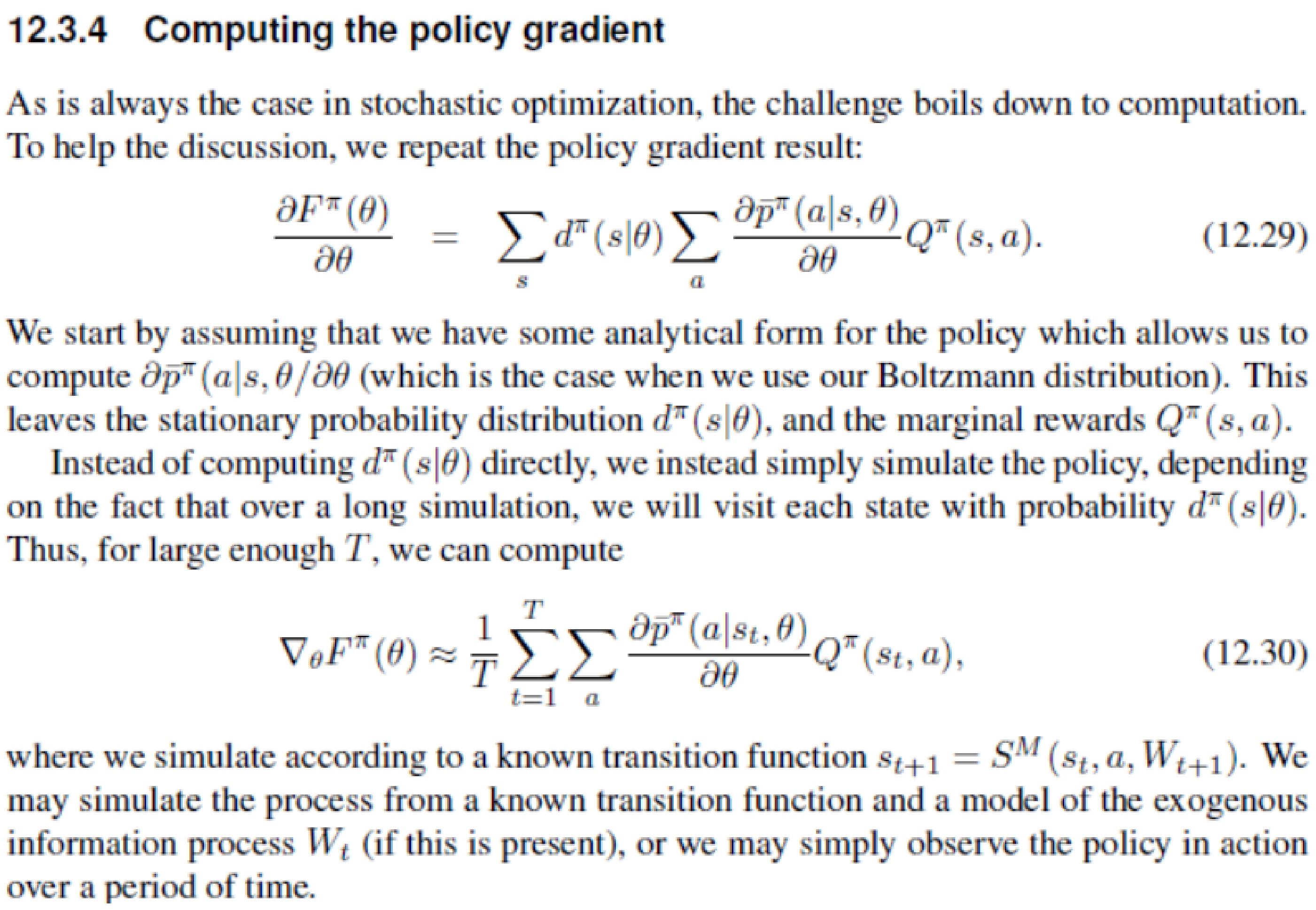
$$\frac{\partial F^{\pi}(\theta)}{\partial \theta} = \sum_s d^{\pi}(s|\theta) \sum_a \frac{\partial \bar{p}^{\pi}(a|s,\theta)}{\partial \theta} Q^{\pi}(s,a) + \sum_s d^{\pi}(s|\theta) \sum_a \bar{p}^{\pi}(a|s,\theta) \sum_{s'} P(s'|s,a) \frac{\partial V^{\pi}(s')}{\partial \theta} - \sum_s d^{\pi}(s|\theta) \frac{\partial V^{\pi}(s)}{\partial \theta} \tag{12.26}$$

$$= \sum_s d^{\pi}(s|\theta) \sum_a \frac{\partial \bar{p}^{\pi}(a|s,\theta)}{\partial \theta} Q^{\pi}(s,a) + \sum_s d^{\pi}(s|\theta) \frac{\partial V^{\pi}(s)}{\partial \theta} - \sum_s d^{\pi}(s|\theta) \frac{\partial V^{\pi}(s)}{\partial \theta} \tag{12.27}$$

$$\boxed{= \sum_s d^{\pi}(s|\theta) \sum_a \frac{\partial \bar{p}^{\pi}(a|s,\theta)}{\partial \theta} Q^{\pi}(s,a).} \tag{12.28}$$

Equation (12.24) uses $\sum_s d^{\pi}(s|\theta) = 1$; (12.25) uses the fact (12.23) holds for all s; (12.26) simply expands (12.25); (12.27) uses the property that since $d^{\pi}(s)$ is the stationary distribution, then $\sum_s d^{\pi}(s|\theta) P(s'|s,a) = d^{\pi}(s'|\theta)$ (after substituting this result, then just change the index from s' to s). Equation (12.28) is the policy gradient theorem we first presented in equation (12.14).

Policy gradient search



12.3.4 Computing the policy gradient

As is always the case in stochastic optimization, the challenge boils down to computation. To help the discussion, we repeat the policy gradient result:

$$\frac{\partial F^\pi(\theta)}{\partial \theta} = \sum_s d^\pi(s|\theta) \sum_a \frac{\partial \bar{p}^\pi(a|s,\theta)}{\partial \theta} Q^\pi(s,a). \tag{12.29}$$

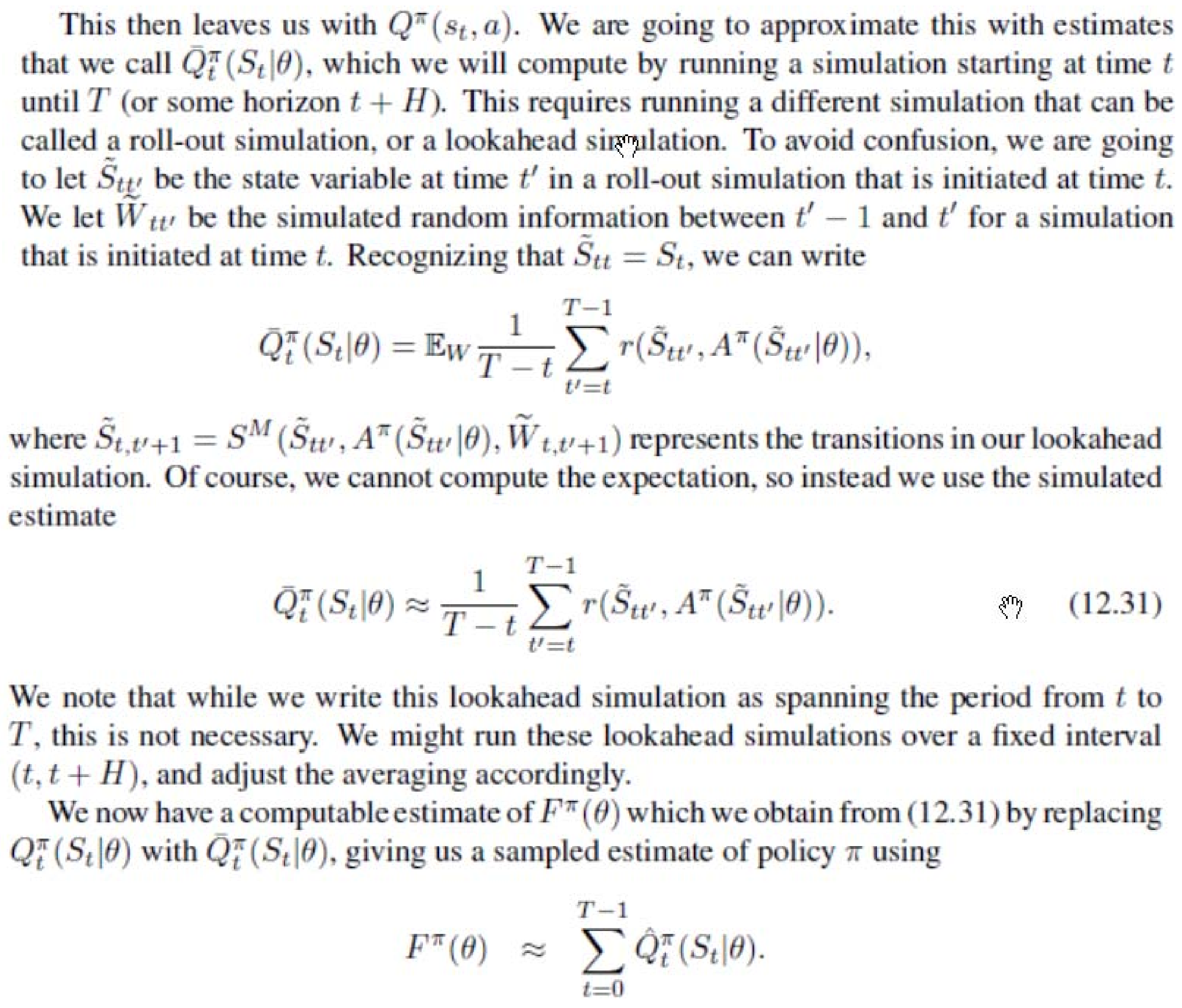
We start by assuming that we have some analytical form for the policy which allows us to compute $\partial \bar{p}^\pi(a|s,\theta/\partial\theta$ (which is the case when we use our Boltzmann distribution). This leaves the stationary probability distribution $d^\pi(s|\theta)$, and the marginal rewards $Q^\pi(s,a)$.

Instead of computing $d^\pi(s|\theta)$ directly, we instead simply simulate the policy, depending on the fact that over a long simulation, we will visit each state with probability $d^\pi(s|\theta)$. Thus, for large enough T, we can compute

$$\nabla_\theta F^\pi(\theta) \approx \frac{1}{T} \sum_{t=1}^{T} \sum_a \frac{\partial \bar{p}^\pi(a|s_t,\theta)}{\partial \theta} Q^\pi(s_t,a), \tag{12.30}$$

where we simulate according to a known transition function $s_{t+1} = S^M(s_t, a, W_{t+1})$. We may simulate the process from a known transition function and a model of the exogenous information process W_t (if this is present), or we may simply observe the policy in action over a period of time.

Policy gradient search



This then leaves us with $Q^\pi(s_t, a)$. We are going to approximate this with estimates that we call $\bar{Q}_t^\pi(S_t|\theta)$, which we will compute by running a simulation starting at time t until T (or some horizon $t + H$). This requires running a different simulation that can be called a roll-out simulation, or a lookahead simulation. To avoid confusion, we are going to let $\tilde{S}_{tt'}$ be the state variable at time t' in a roll-out simulation that is initiated at time t. We let $\tilde{W}_{tt'}$ be the simulated random information between $t' - 1$ and t' for a simulation that is initiated at time t. Recognizing that $\tilde{S}_{tt} = S_t$, we can write

$$\bar{Q}_t^\pi(S_t|\theta) = \mathbb{E}_W \frac{1}{T-t} \sum_{t'=t}^{T-1} r(\tilde{S}_{tt'}, A^\pi(\tilde{S}_{tt'}|\theta)),$$

where $\tilde{S}_{t,t'+1} = S^M(\tilde{S}_{tt'}, A^\pi(\tilde{S}_{tt'}|\theta), \tilde{W}_{t,t'+1})$ represents the transitions in our lookahead simulation. Of course, we cannot compute the expectation, so instead we use the simulated estimate

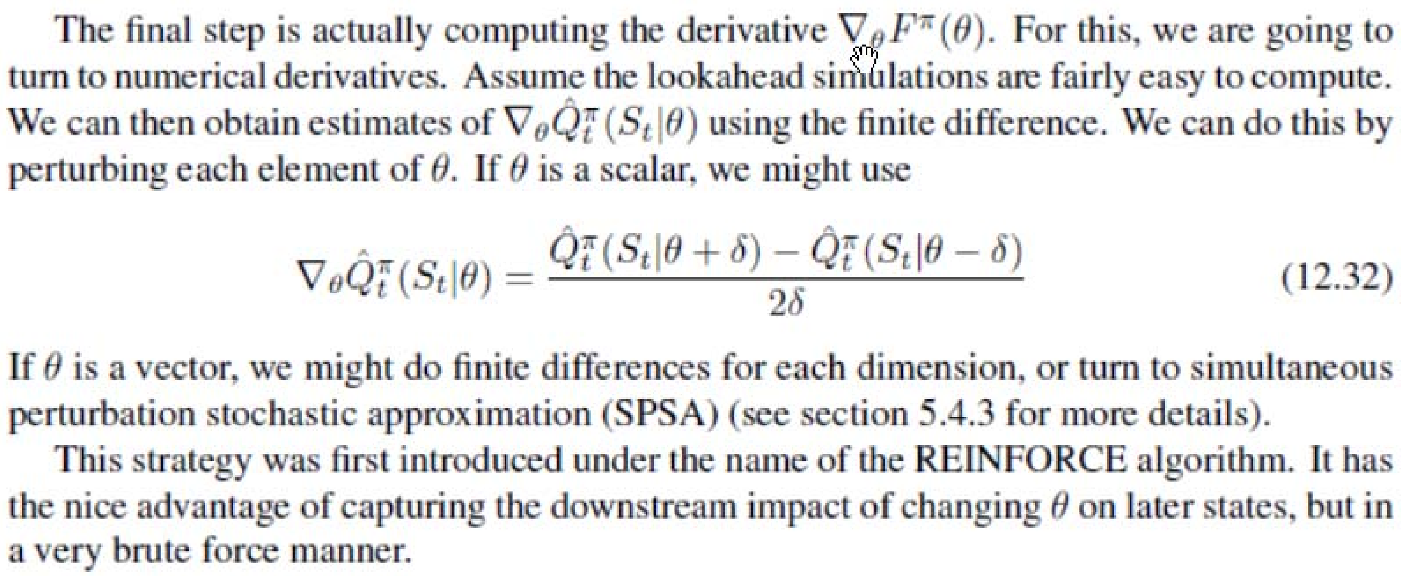
$$\bar{Q}_t^\pi(S_t|\theta) \approx \frac{1}{T-t} \sum_{t'=t}^{T-1} r(\tilde{S}_{tt'}, A^\pi(\tilde{S}_{tt'}|\theta)). \qquad (12.31)$$

We note that while we write this lookahead simulation as spanning the period from t to T, this is not necessary. We might run these lookahead simulations over a fixed interval $(t, t + H)$, and adjust the averaging accordingly.

We now have a computable estimate of $F^\pi(\theta)$ which we obtain from (12.31) by replacing $Q_t^\pi(S_t|\theta)$ with $\bar{Q}_t^\pi(S_t|\theta)$, giving us a sampled estimate of policy π using

$$F^\pi(\theta) \approx \sum_{t=0}^{T-1} \hat{Q}_t^\pi(S_t|\theta).$$

Policy gradient search



The final step is actually computing the derivative $\nabla_\theta F^\pi(\theta)$. For this, we are going to turn to numerical derivatives. Assume the lookahead simulations are fairly easy to compute. We can then obtain estimates of $\nabla_\theta \hat{Q}_t^\pi(S_t|\theta)$ using the finite difference. We can do this by perturbing each element of θ. If θ is a scalar, we might use

$$\nabla_\theta \hat{Q}_t^\pi(S_t|\theta) = \frac{\hat{Q}_t^\pi(S_t|\theta+\delta) - \hat{Q}_t^\pi(S_t|\theta-\delta)}{2\delta} \qquad (12.32)$$

If θ is a vector, we might do finite differences for each dimension, or turn to simultaneous perturbation stochastic approximation (SPSA) (see section 5.4.3 for more details).

This strategy was first introduced under the name of the REINFORCE algorithm. It has the nice advantage of capturing the downstream impact of changing θ on later states, but in a very brute force manner.

Policy gradient search

- Notes:
  - » The policy gradient search is typically illustrated in the context of Boltzmann, which is characterized by a scalar parameter.
  - » This can be optimized using specialized search routines for scalar problems.
  - » It can also be tackled using the SPSA, where the model is viewed as a black-box simulator.
  - » Even policy gradient search involves the stepsize issue.

Policy gradient II

Derivative-based: Control problems

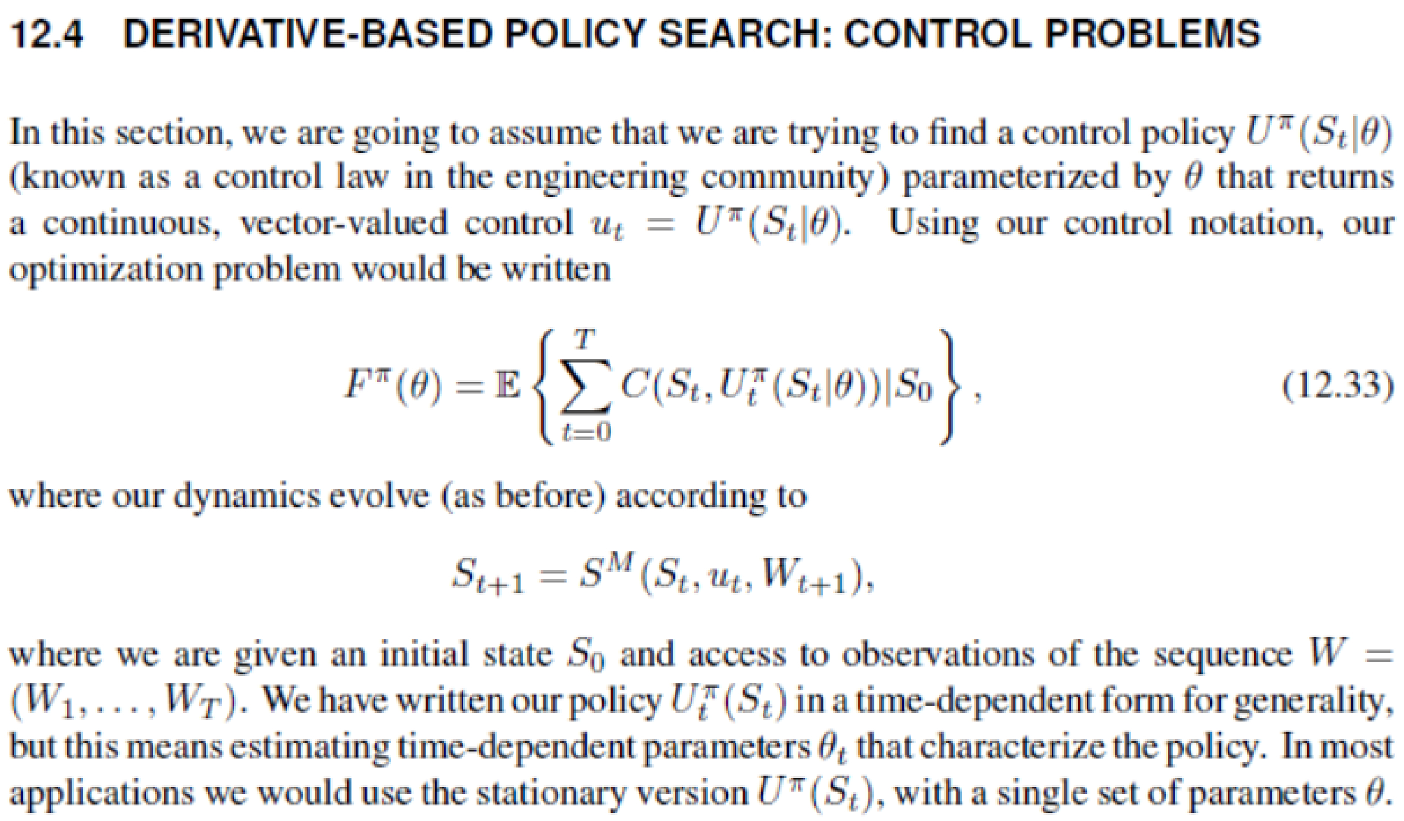
Policy gradient for control problems

- Control problems
  - » We use "control problem" to describe problems where the downstream state is a continuous function of the decision, captured by the transition function:

$$S_{t+1} = S^M(S_t, X^\pi(S_t|\theta), W_{t+1})$$

  - » We need to capture the effect of changing θ on $x_t = X^\pi(S_t|\theta)$, and then the effect of x_t on $S_{t+1} = S^M(S_t, x_t, W_{t+1})$.

Policy gradient for control problems



12.4 DERIVATIVE-BASED POLICY SEARCH: CONTROL PROBLEMS

In this section, we are going to assume that we are trying to find a control policy $U^\pi(S_t|\theta)$ (known as a control law in the engineering community) parameterized by θ that returns a continuous, vector-valued control $u_t = U^\pi(S_t|\theta)$. Using our control notation, our optimization problem would be written

$$F^\pi(\theta) = \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, U_t^\pi(S_t|\theta))|S_0\right\}, \tag{12.33}$$

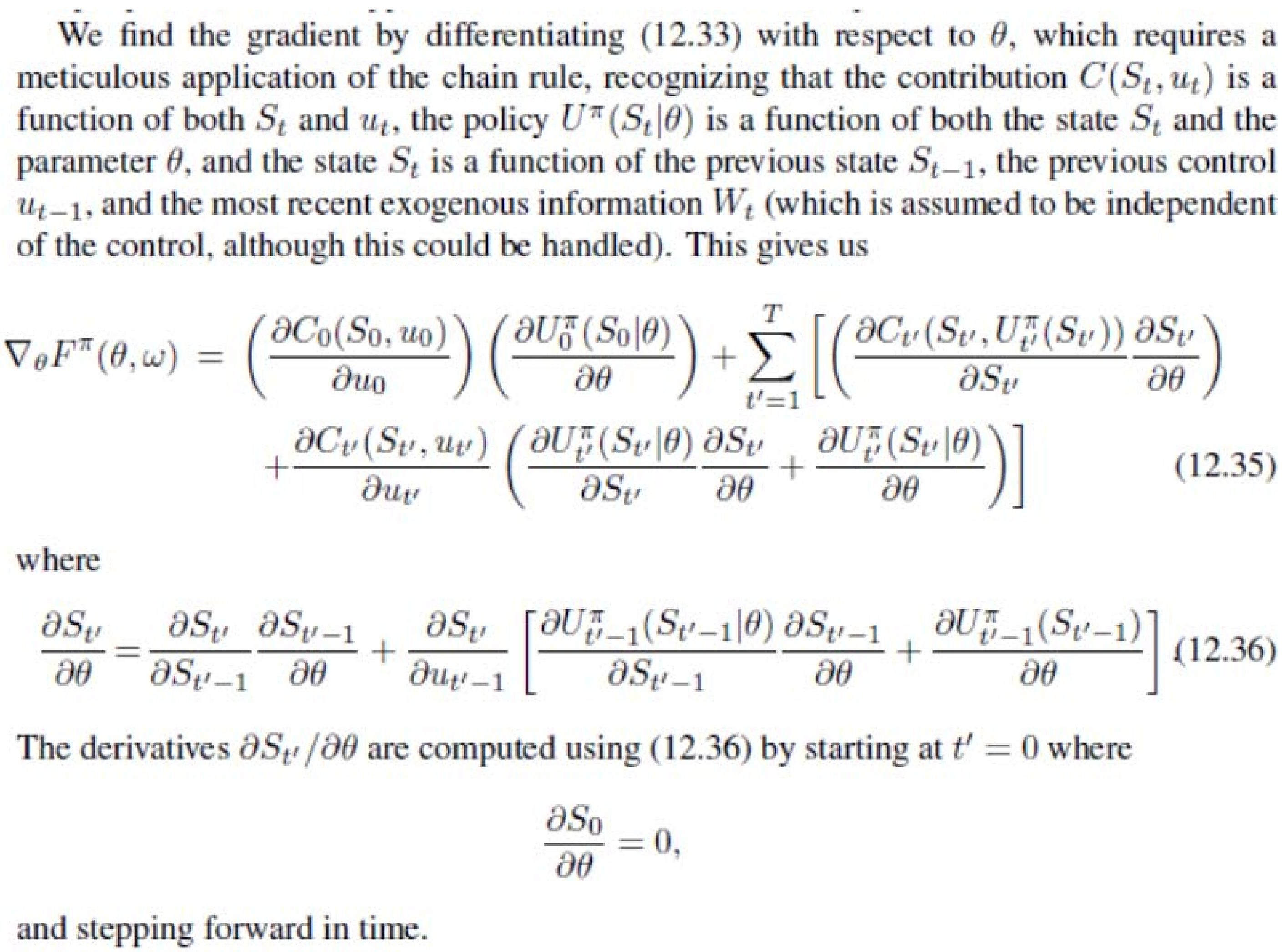
where our dynamics evolve (as before) according to

$$S_{t+1} = S^M(S_t, u_t, W_{t+1}),$$

where we are given an initial state S_0 and access to observations of the sequence $W = (W_1, \ldots, W_T)$. We have written our policy $U_t^\pi(S_t)$ in a time-dependent form for generality, but this means estimating time-dependent parameters θ_t that characterize the policy. In most applications we would use the stationary version $U^\pi(S_t)$, with a single set of parameters θ.

Policy gradient for control problems

- Applying the chain rule:



We find the gradient by differentiating (12.33) with respect to θ, which requires a meticulous application of the chain rule, recognizing that the contribution $C(S_t, u_t)$ is a function of both S_t and u_t, the policy $U^\pi(S_t|\theta)$ is a function of both the state S_t and the parameter θ, and the state S_t is a function of the previous state S_{t-1}, the previous control u_{t-1}, and the most recent exogenous information W_t (which is assumed to be independent of the control, although this could be handled). This gives us

$$\nabla_\theta F^\pi(\theta,\omega) = \left(\frac{\partial C_0(S_0,u_0)}{\partial u_0}\right)\left(\frac{\partial U_0^\pi(S_0|\theta)}{\partial \theta}\right) + \sum_{t'=1}^{T}\left[\left(\frac{\partial C_{t'}(S_{t'},U_{t'}^\pi(S_{t'}))}{\partial S_{t'}}\frac{\partial S_{t'}}{\partial \theta}\right) + \frac{\partial C_{t'}(S_{t'},u_{t'})}{\partial u_{t'}}\left(\frac{\partial U_{t'}^\pi(S_{t'}|\theta)}{\partial S_{t'}}\frac{\partial S_{t'}}{\partial \theta} + \frac{\partial U_{t'}^\pi(S_{t'}|\theta)}{\partial \theta}\right)\right] \quad (12.35)$$

where

$$\frac{\partial S_{t'}}{\partial \theta} = \frac{\partial S_{t'}}{\partial S_{t'-1}}\frac{\partial S_{t'-1}}{\partial \theta} + \frac{\partial S_{t'}}{\partial u_{t'-1}}\left[\frac{\partial U_{t'-1}^\pi(S_{t'-1}|\theta)}{\partial S_{t'-1}}\frac{\partial S_{t'-1}}{\partial \theta} + \frac{\partial U_{t'-1}^\pi(S_{t'-1})}{\partial \theta}\right] \quad (12.36)$$

The derivatives $\partial S_{t'}/\partial\theta$ are computed using (12.36) by starting at $t' = 0$ where

$$\frac{\partial S_0}{\partial \theta} = 0,$$

and stepping forward in time.

Policy gradient for control problems

- Notes:
  - » Now we just have to take the derivative of the:
    - Cost/contribution function
    - Transition function
    - Policy
  - » This approach is very popular using neural networks. So popular that the derivatives can be computed using libraries such as "Tensorflow."
  - » … but, we can also use numerical derivatives.

Policy search

Derivative-free

Derivative-free policy search

- Derivative-free policy search
  - » Apply everything we learned in chapter 7.
  - » Policy search is typically performed in a simulator, in which case we would use a "final reward" objective:

$$\max_{\pi} \mathbb{E}\{F\left(\mathrm{x}^{\pi,N},\widehat{W}\right)|S^0\} = \mathbb{E}_{S^0}\mathbb{E}_{W^1,\dots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}F(x^{\pi,N},\widehat{W})$$

  - » This is for state-independent problems, but what about state-dependent problems? Instead of learning the implementation decision $x^{\pi,N}$, we need to learn the implementation decision as a function of the state S, which is a policy that we call the *implementation policy* $X^{\pi^{imp}}(S)$. We are going to write this as $X^{\pi^{imp}}(S|\theta^{imp})$ to express the likely dependence on tunable parameters to be used in the implementation policy.
  - » Just as we needed a "learning policy" π to learn the implementation decision $x^{\pi,N}$, we need a learning policy π^{lrn} in the state-dependent case to learn the implementation policy π^{imp}.

Derivative-free policy search

- Derivative-free policy search
  - » We are going to assume that we are simulating over time:

$$(S_0, x_0, W_1, S_1, \ldots, S_t, x_t, W_{t+1}, \ldots)$$

  - » Now we need to solve

$$\max_{\pi^{lrn}} \mathbb{E}\{C\left(S, X^{\pi^{imp}}(S|\theta^{imp}), \widehat{W}\right)|S\}$$

  - » This involves three types of uncertainty:
    - Bayesian priors on uncertain parameters (if available), which we capture in the initial state S_0.
    - Uncertainty in the process of learning when applying the learning policy π^{lrn} using the realizations of $W_1, W_2, \ldots$
    - Uncertainty in the process of evaluating the implementation policy $X^{\pi^{imp}}(S_t|\theta^{imp})$.

Derivative-free policy search

- Derivative-free policy search
  - » We have to evaluate

$$\max_{\pi^{lrn}} \mathbb{E}\{C\left(S, X^{\pi^{imp}}(S|\theta^{imp}), \widehat{W}\right)|S\}$$

  - » We first expand the expectations:

$$\max_{\pi^{lrn}} \mathbb{E}_{S_0} \mathbb{E}^{\pi^{lrn}}_{W_1,\ldots,W_T|S_0 \mathbb{E}_{S_0}} \mathbb{E}^{\pi^{imp}}_{S|S_0} \mathbb{E}_{\widehat{W}|S_0}\{C\left(S, X^{\pi^{imp}}(S|\theta^{imp}), \widehat{W}\right)|S\}$$

  - » Next we have to write this out in a form we can simulate. The hardest part is the expectation over the state *S* whose distribution reflects the implementation policy. We do this by simulating the implementation policy:

$$\max_{\pi^{lrn}} \mathbb{E}_{S^0} \mathbb{E}^{\pi^{imp}}_{((W_t^n)_{t=0}^{T})_{n=0}^{N}|S^0} \left(\mathbb{E}^{\pi^{imp}}_{(\widehat{W}_t)_{t=0}^{T}|S^0} \frac{1}{T} \sum_{t=0}^{T-1} C(S_t, X^{\pi^{imp}}(S_t|\theta^{imp}), \widehat{W}_{t+1}) \right)$$

Derivative-free policy search

- Derivative-free policy search
  - » To simulate

$$\max_{\pi^{lrn}} \mathbb{E}_{S^0} \mathbb{E}^{\pi^{imp}}_{((W_t^n)_{t=0}^T)_{n=0}^N | S^0} \left(\mathbb{E}^{\pi^{imp}}_{(\widehat{W}_t)_{t=0}^T | S^0} \frac{1}{T} \sum_{t=0}^{T-1} C(S_t, X^{\pi^{imp}}(S_t|\theta^{imp}), \widehat{W}_{t+1}) \right)$$

  - » Let:

    ψ be a sample of any variables in a Bayesian prior in S_0.

    ω be a sampled sequence $W_1, W_2, \ldots, W_T$ (where we follow the learning policy)

    $\widehat{\omega}$ be a sampled sequence $\widehat{W}_1, \widehat{W}_2, \ldots, \widehat{W}_T$ (where we follow the implementation policy)

  - » Now replace each expectation with a sampled estimate.

Derivative-free policy search

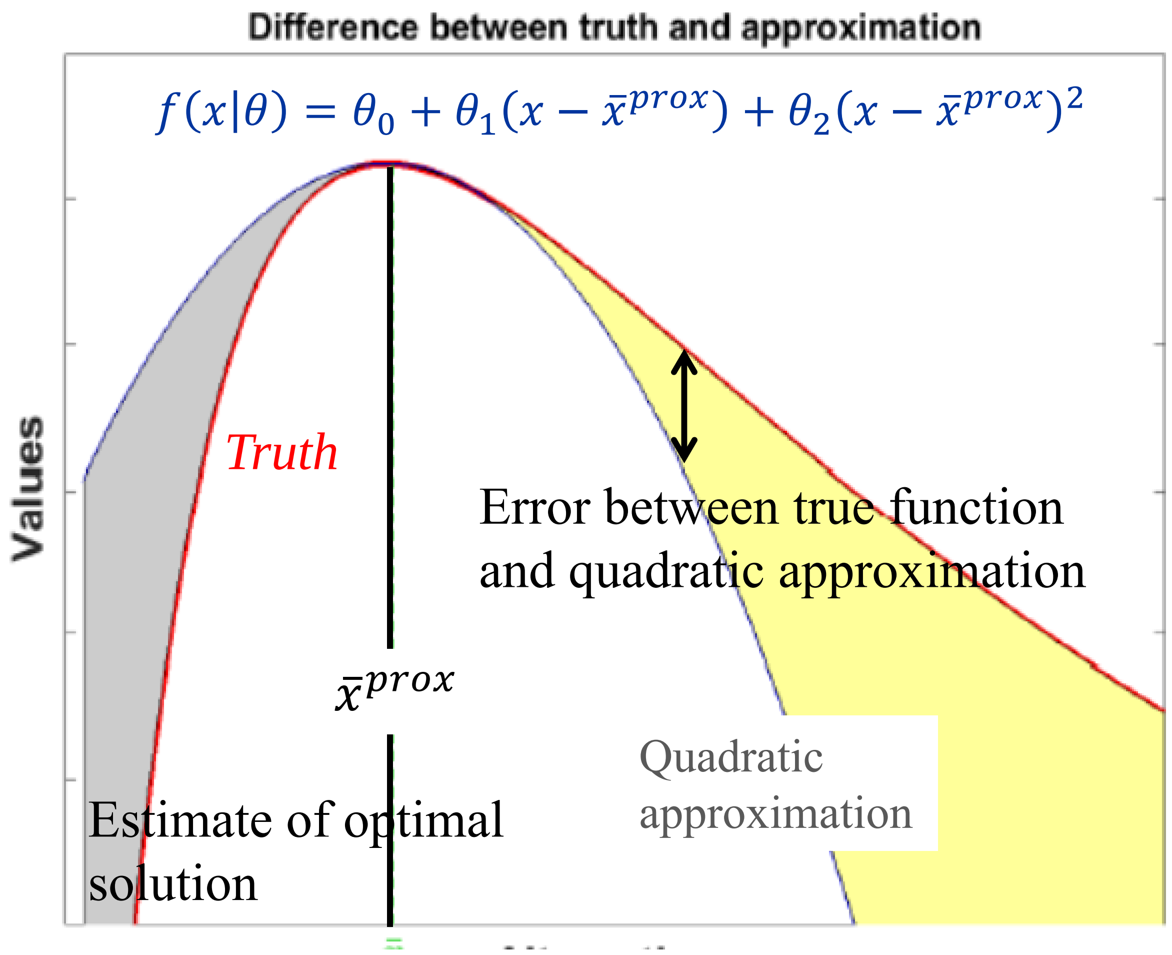
- Issues:
  - » Dimensionality of control vector
    - Scalar (specialized search algorithms)
    - Low dimensional (enumeration?)
    - High dimensional (use sampling)
  - » Creating belief models
    - Linear models
    - Locally linear
    - Correlated beliefs (Gaussian process regression)
  - » Online (cumulative reward) vs. offline (terminal reward)

Policy search

Locally quadratic knowledge gradient

Knowledge gradient with local quadratic belief

- Locally quadratic knowledge gradient
  - » Uses locally quadratic approximation around proximal point $\bar{x}$.
  - » Difference between true function and quadratic approx. is Lipschitz
  - » Encourages learning away from proximal point, but not too far.



Belief mode:

$$\hat{y} = f(x|\theta) + \beta(x|\bar{x}) + \varepsilon$$

Structural uncertainty:

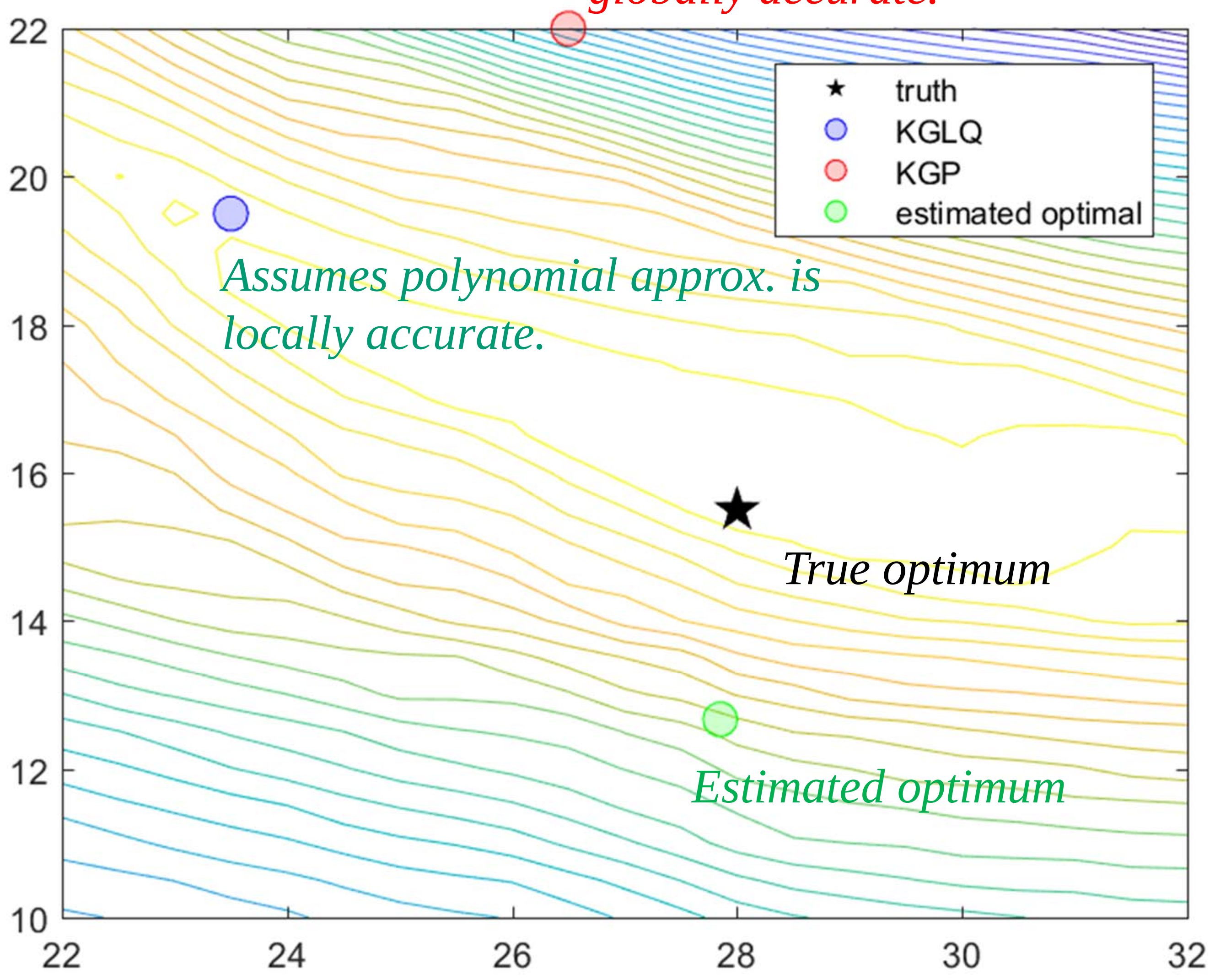
$$\beta(x|\bar{x}) \sim N(0, \kappa|x - \bar{x}|^p)$$

Gaussian noise:

$$\varepsilon \sim N(0, \sigma_\varepsilon^2)$$

Knowledge gradient with local quadratic belief

Assumes polynomial approx. is globally accurate.



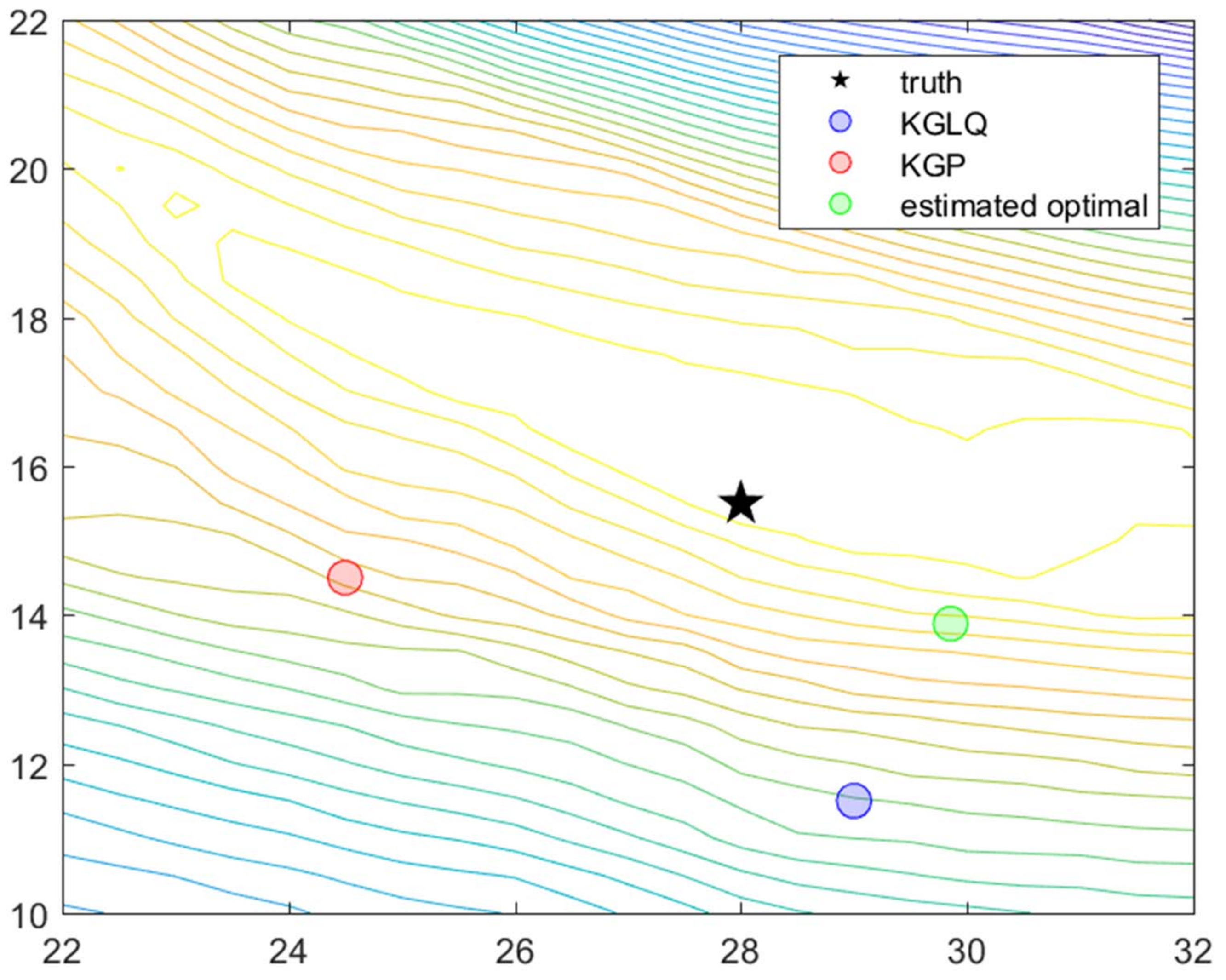
Assumes polynomial approx. is locally accurate.

True optimum

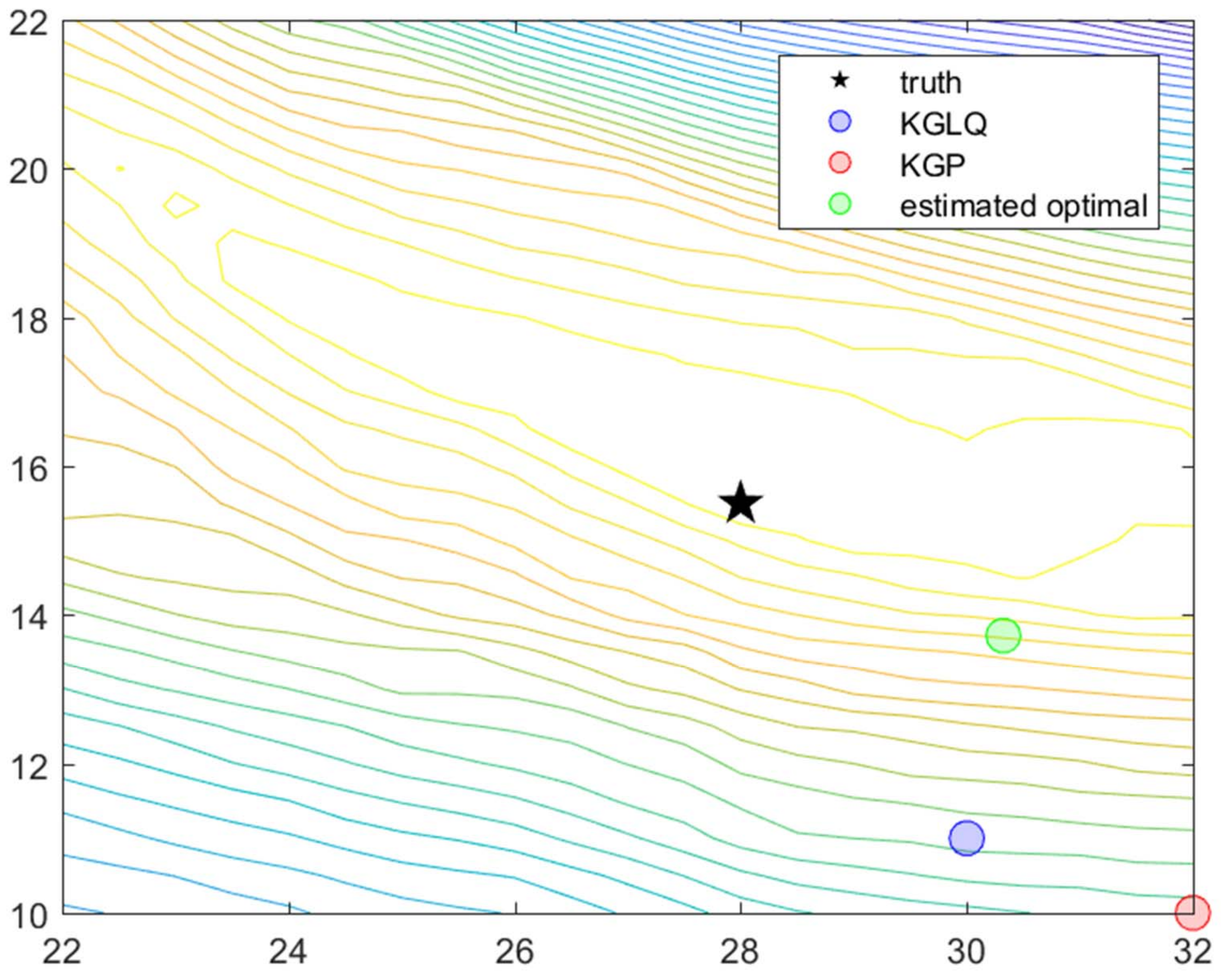
Estimated optimum

Statement A: Approved for public release: distribution unlimited

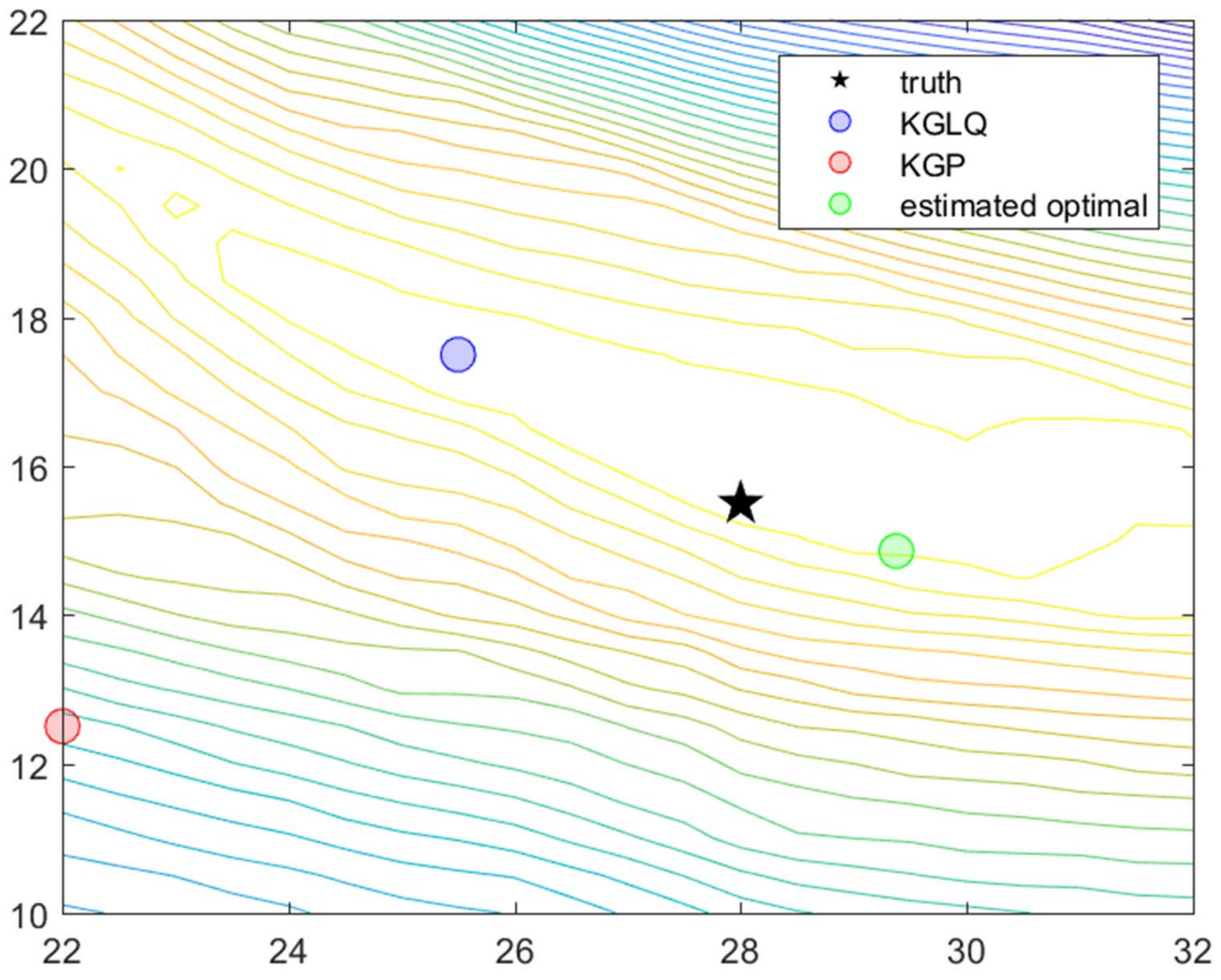
Knowledge gradient with local quadratic belief



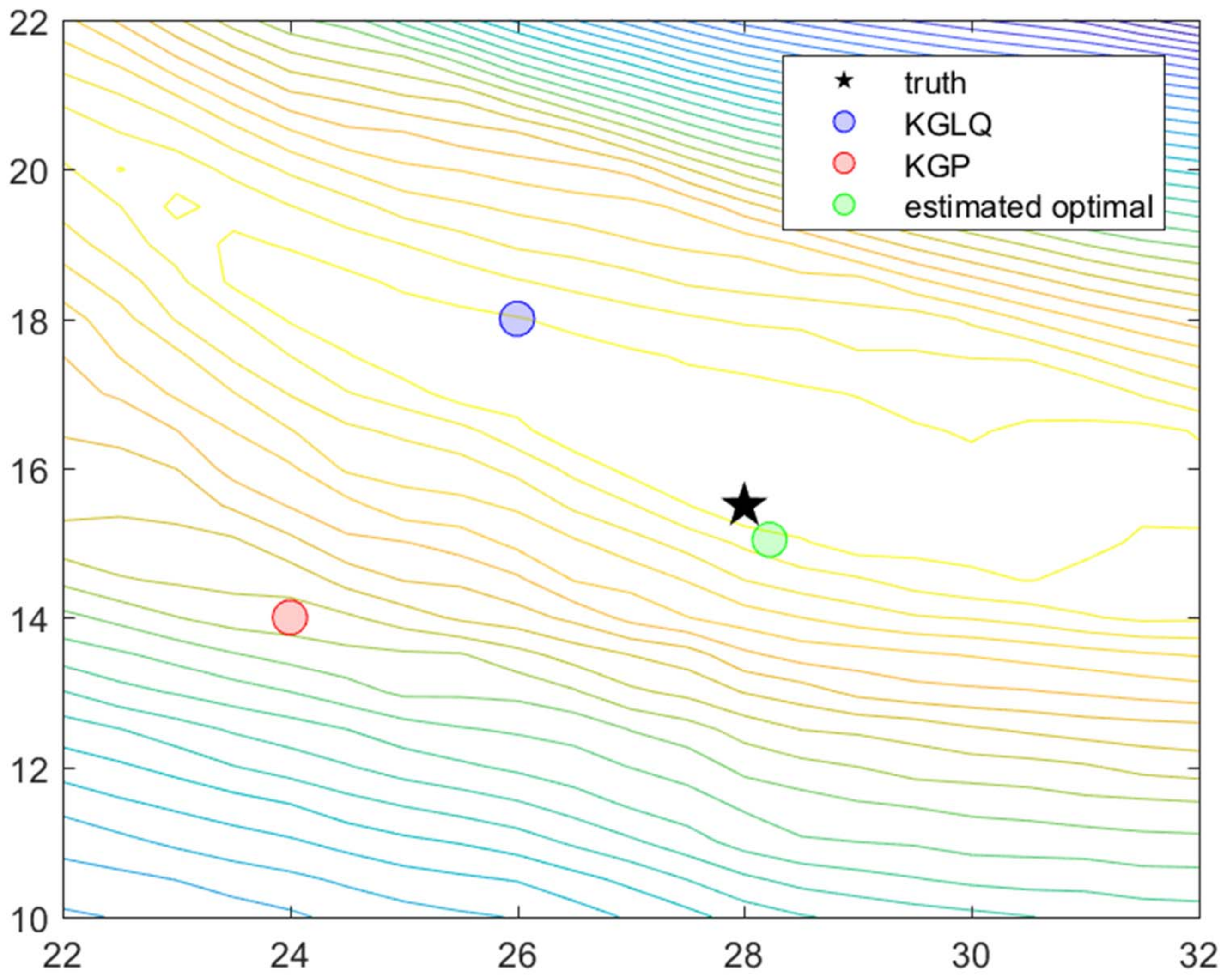
Knowledge gradient with local quadratic belief



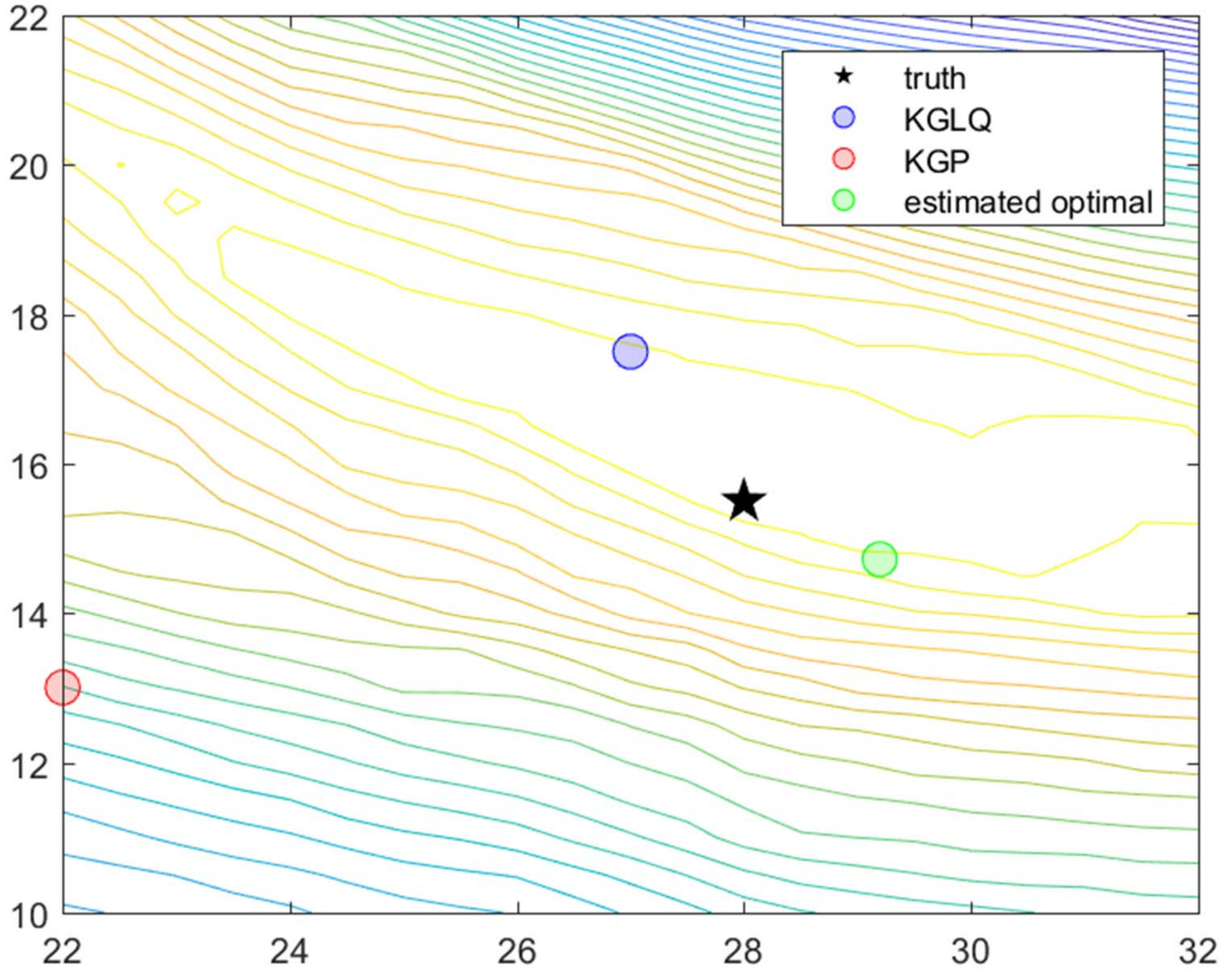
Knowledge gradient with local quadratic belief



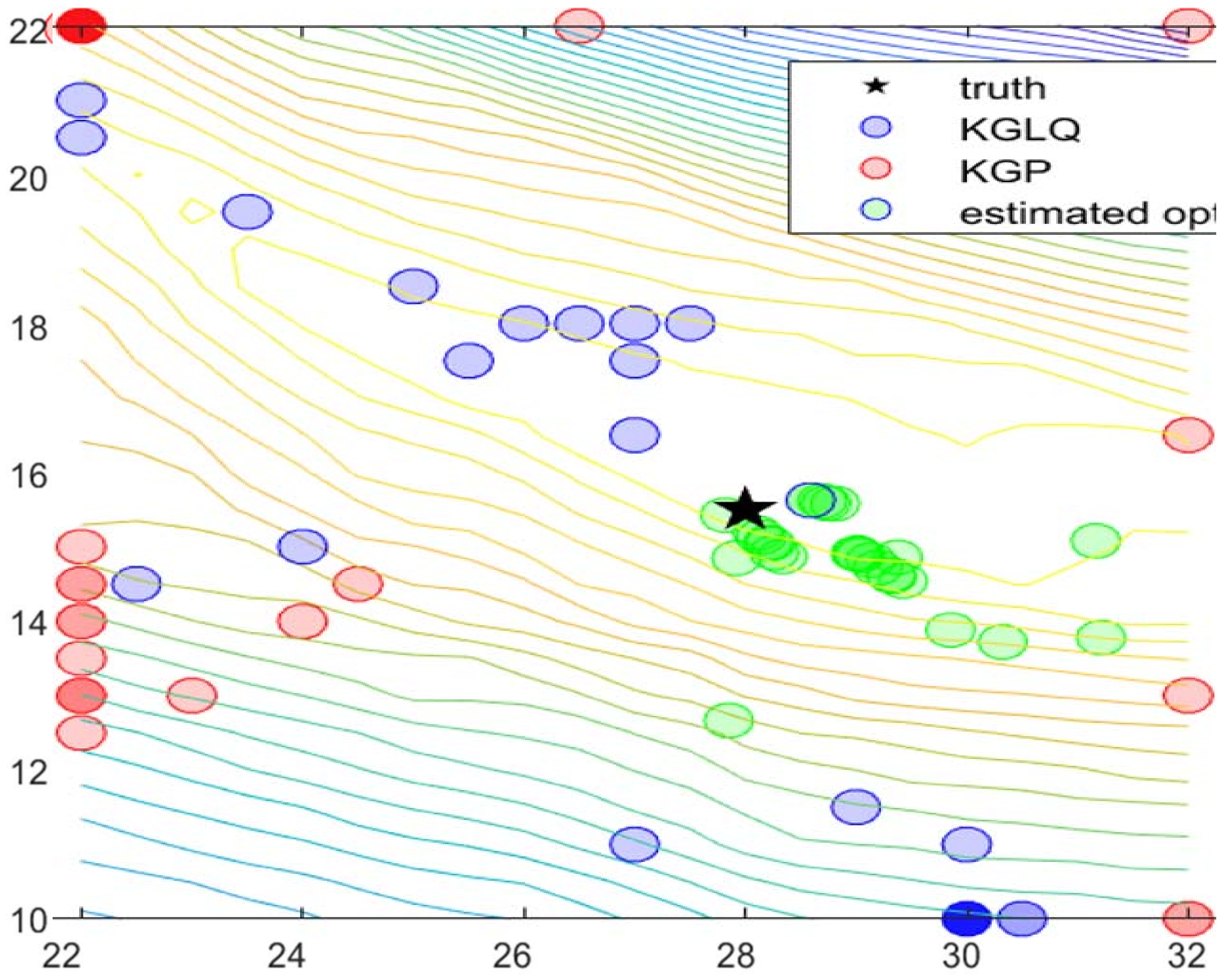
Knowledge gradient with local quadratic belief



Knowledge gradient with local quadratic belief



Knowledge gradient with local quadratic belief


truth
KGLQ
KGP
estimated opt