Tutorial: Optimization via Simulation with Bayesian Statistics and Dynamic Programming

Peter I. Frazier

Operations Research & Information Engineering, Cornell University

Monday December 10, 2012
Advanced Tutorial
Winter Simulation Conference
Berlin

This talk shows how to create algorithms with good average-case performance

- In this talk, we show how to create algorithms for optimization via simulation with good **average-case** performance.
- We use mathematical ideas from Bayesian statistics, dynamic programming, and decision analysis.
- Advantages of this approach:
  - It provides algorithms that work well, in the sense that they find good solutions using few simulations on "typical" problems.
  - It gives us a framework that makes incorporating new problem-specific structure into algorithms easier.
  - It gives us a rigorous way to think about what the **optimal** algorithm is for a particular problem setting.
- There are drawbacks too, to be discussed along the way.

Outline

The Ranking and Selection Problem

- We have a collection of "alternatives".
- Each alternative is something we can simulate, resulting in a noisy observation of its underlying quality.
- We will go through an experimentation period where we try out each alternative some number of times.
- Our goal is to figure out which alternative is best, in the sense of having the reward distribution with the largest mean.
- Examples:
  - Designs for a supply chain.
  - Different screening strategies in public health.
  - Anything we can apply simulation to, and would want to optimize.

The Ranking and Selection Problem: Notation

- For each alternative x, let $\theta_x \in \mathbb{R}$ be the mean of the alternative's sampling distribution.
- For ease of presentation, we assume a common known sampling variance λ^2, but the ideas presented can be generalized to the heterogeneous unknown variance case.
- We assume independent, normally distributed samples. Similar ideas can be applied to other parametric sampling distributions, e.g., Bernoulli.
- A problem instance is then specified by a vector $\theta = [\theta_1, \ldots, \theta_k]$. The problem instance also implicitly depends on λ^2, but we hide this dependence.
- We have a budget of N samples.

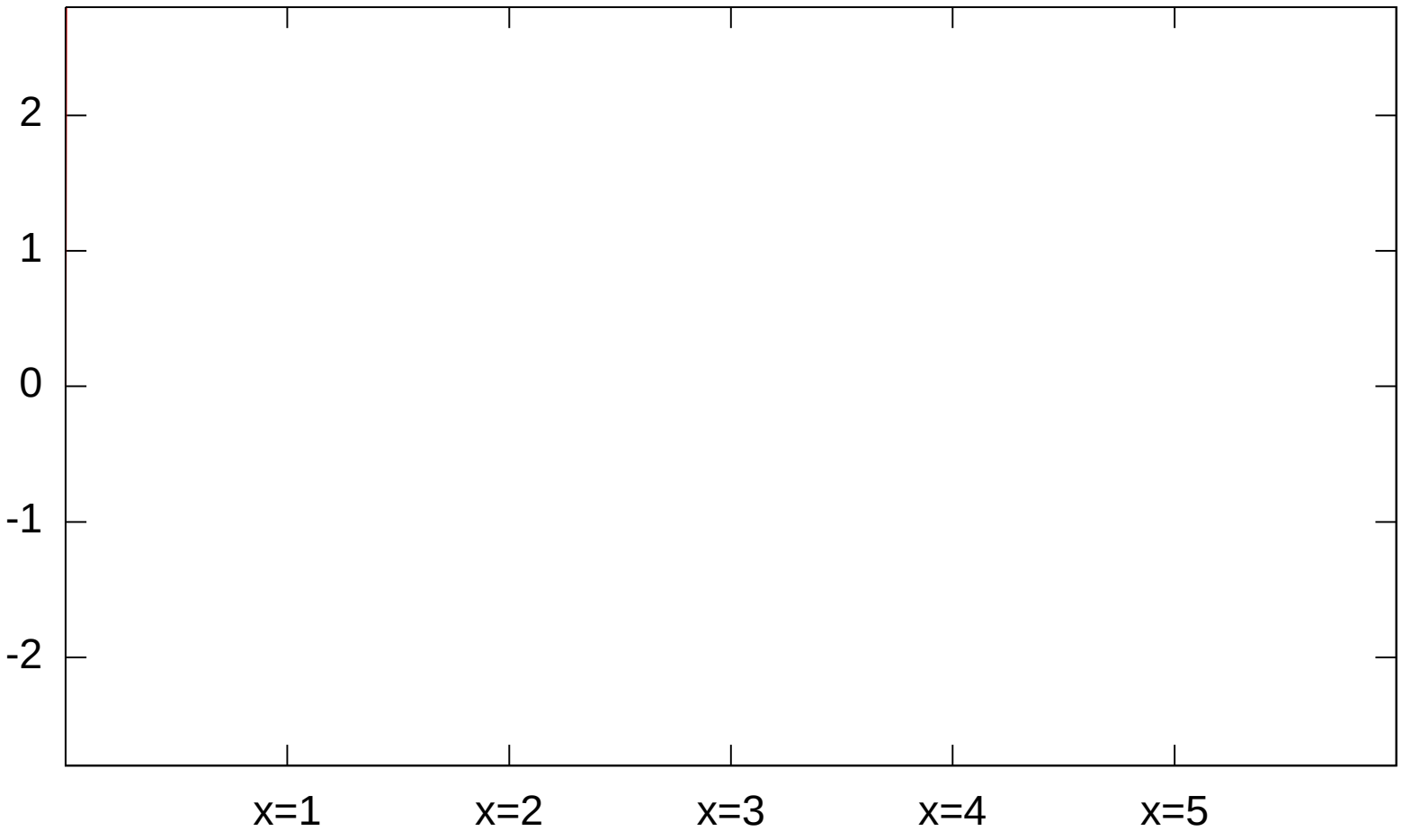
The Ranking and Selection Problem

The expected performance of an algorithm π on a problem instance θ is given by computing the expected reward of:

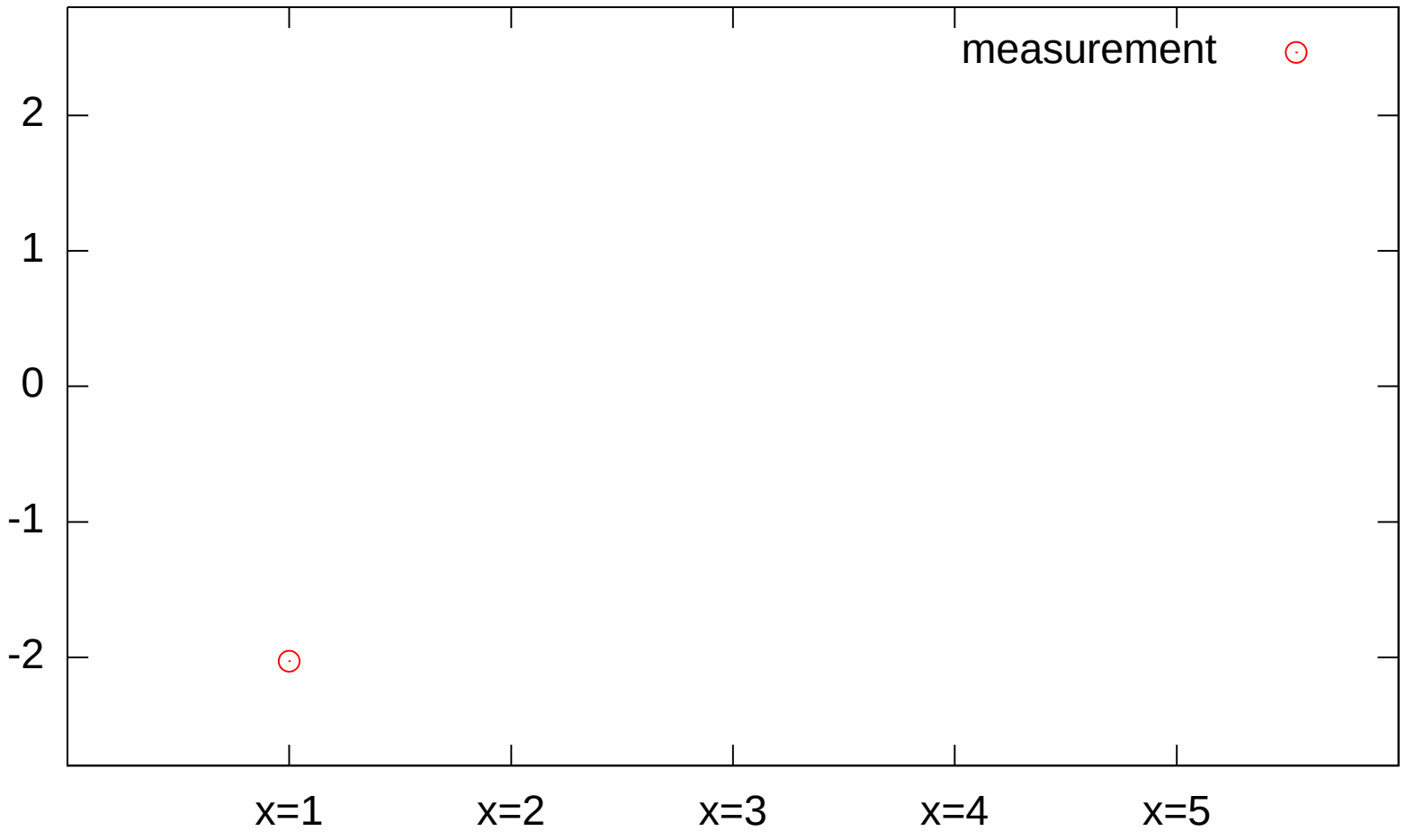
1. For $n = 1, \ldots, N$
   1. Use algorithm π to choose the next alternative to sample, x_n.
   2. Observe $y_n | x_n \sim \text{Normal}(\theta_{x_n})$.
2. Use algoritm π to choose $\widehat{x} \in \{1, \ldots, k\}$ based on $x_1, \ldots, x_N$, $y_1, \ldots, y_N$.
3. Earn a reward equal to the true value $\theta_{\widehat{x}}$ of the selected alternative.

The choice of each x_n may be adaptive, i.e., may depend on previous samples $x_1, \ldots, x_{n-1}, y_1, \ldots, y_{n-1}$.

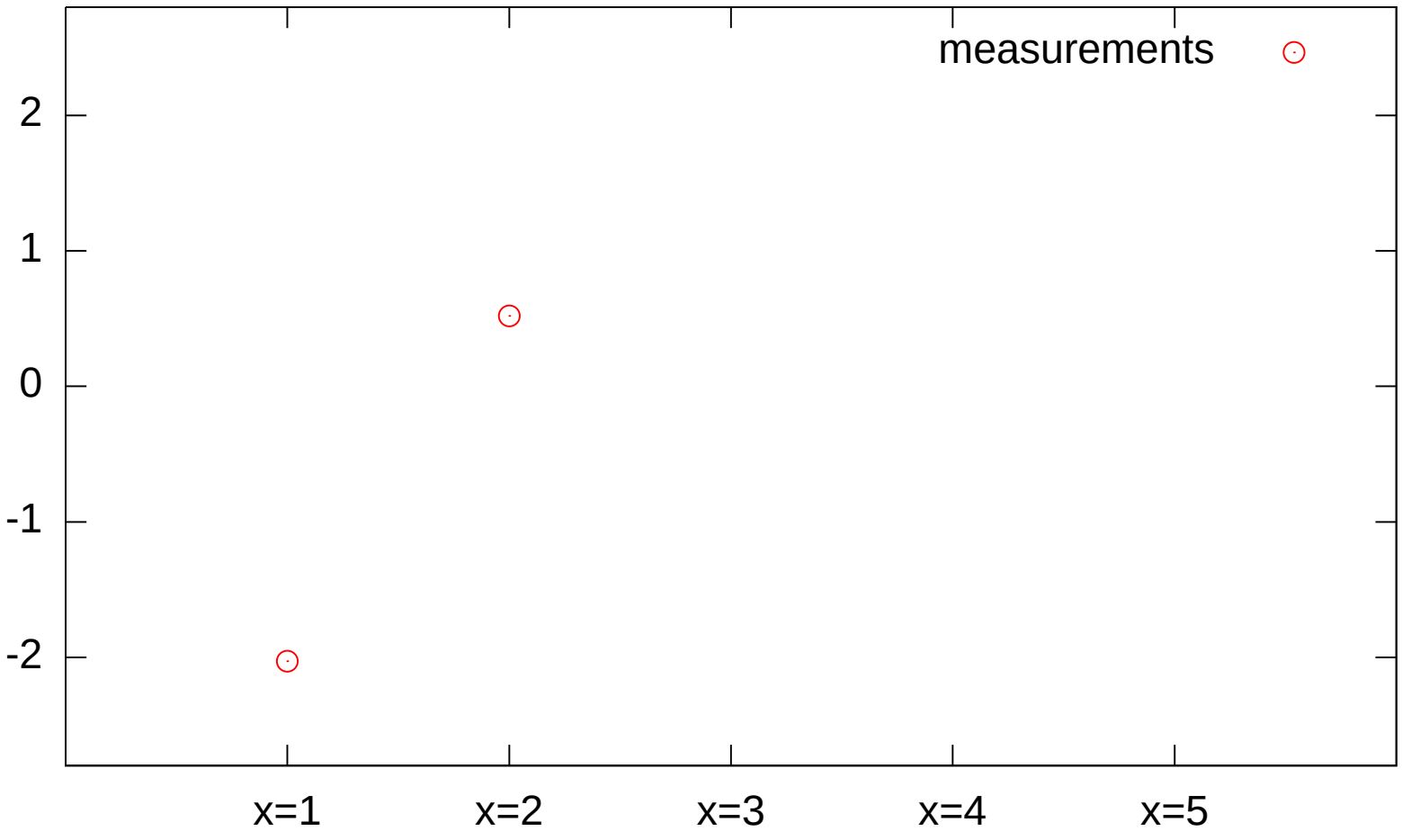
Example, Time 0



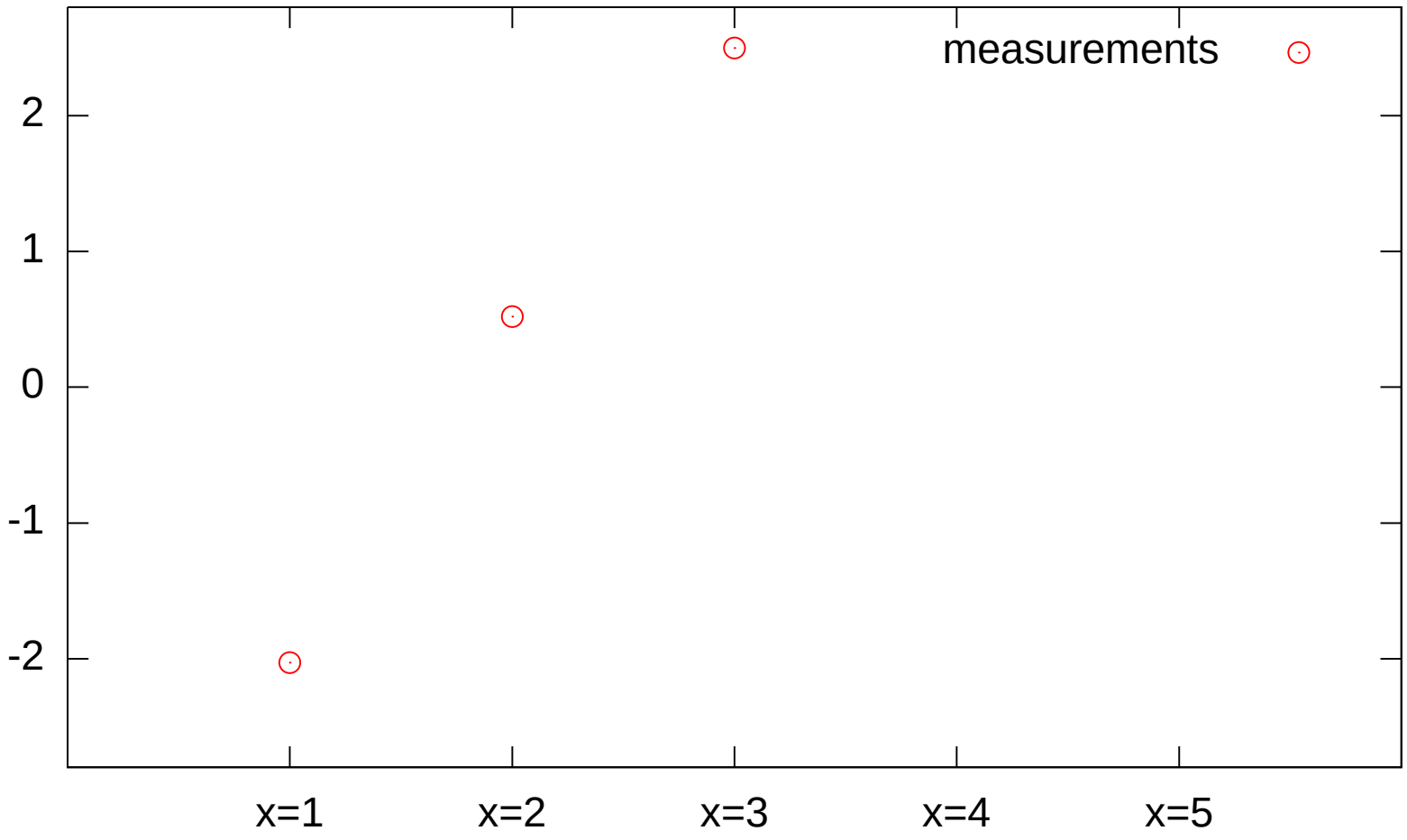
Example, Time 1



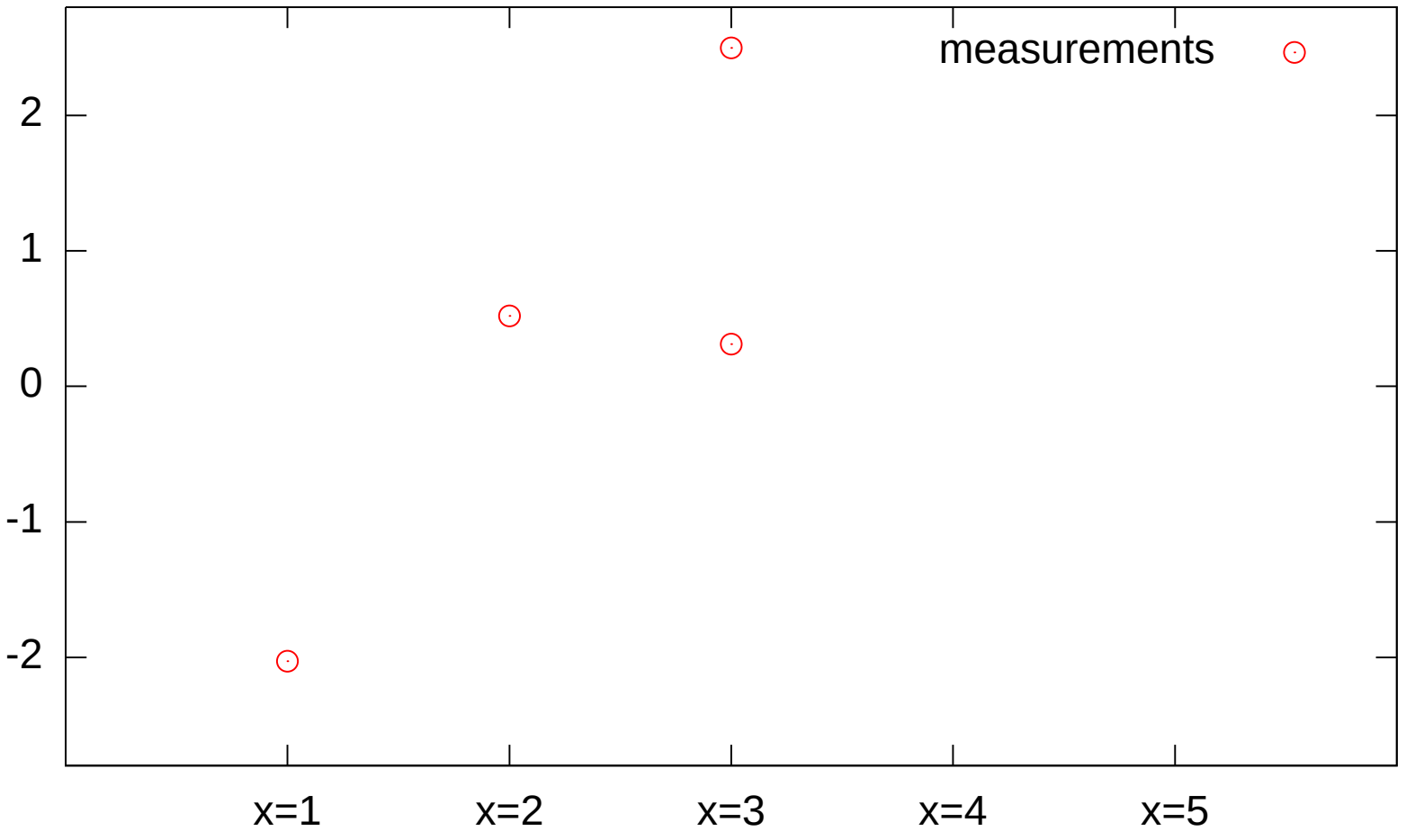
Example, Time 2



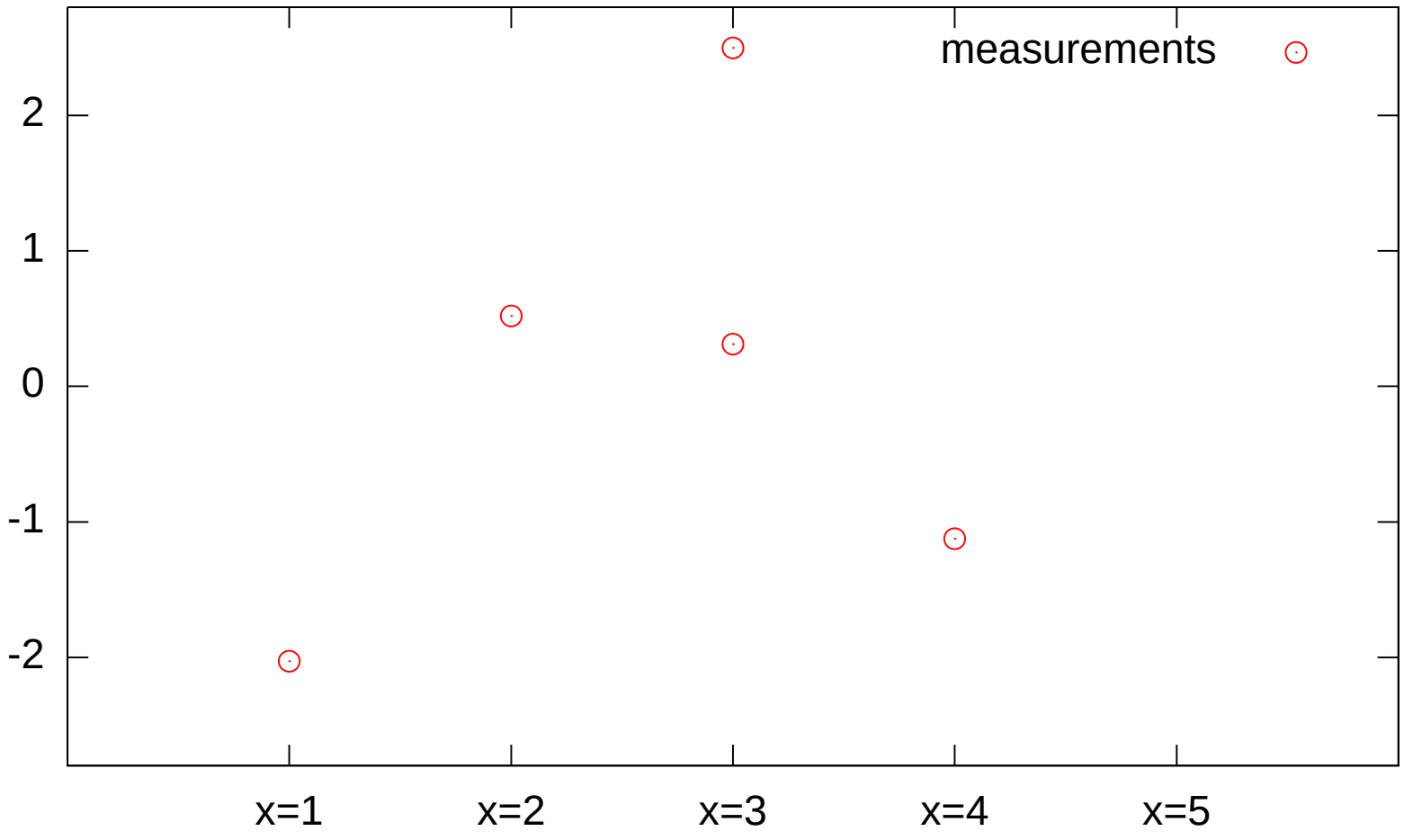
Example, Time 3



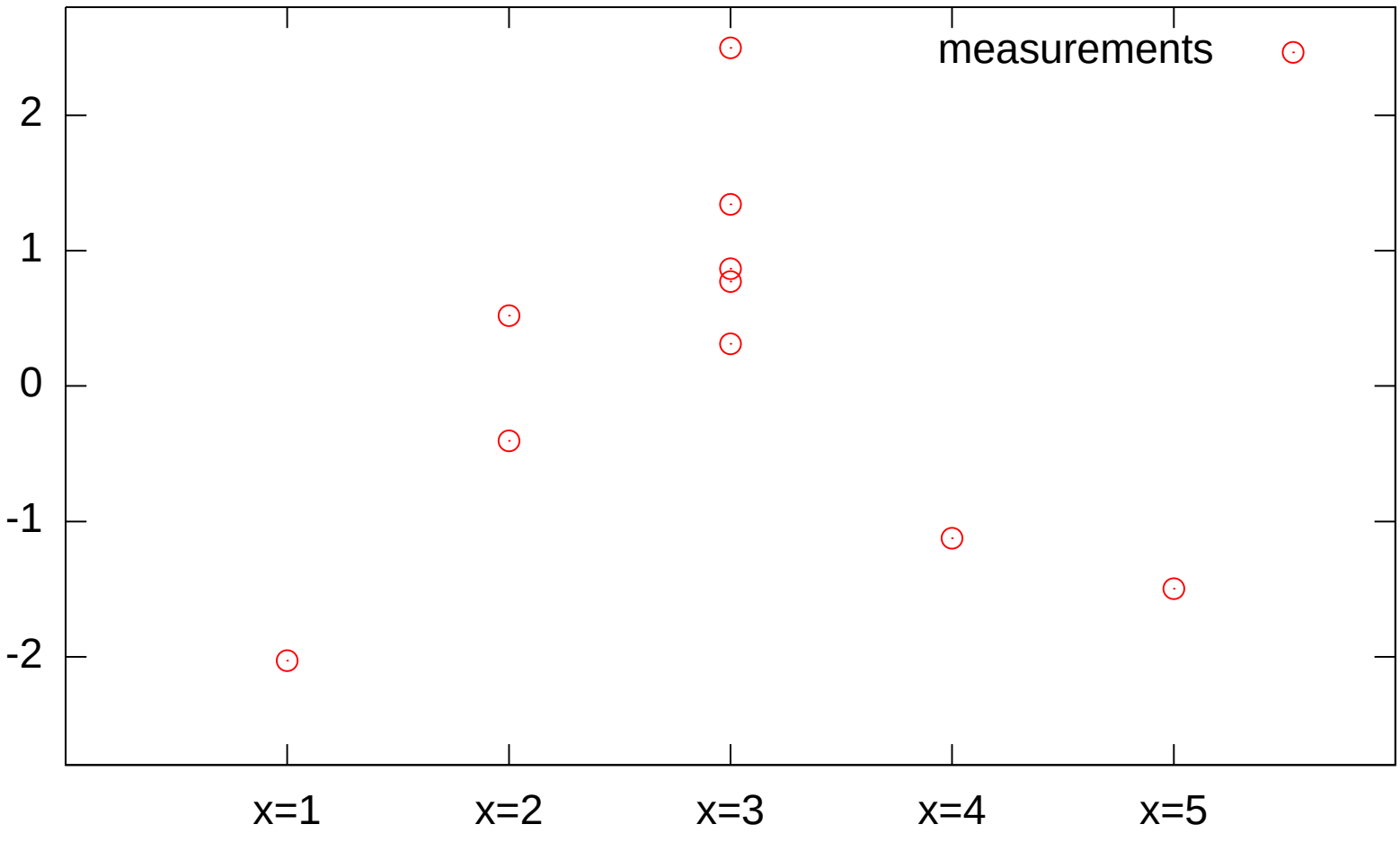
Example, Time 4



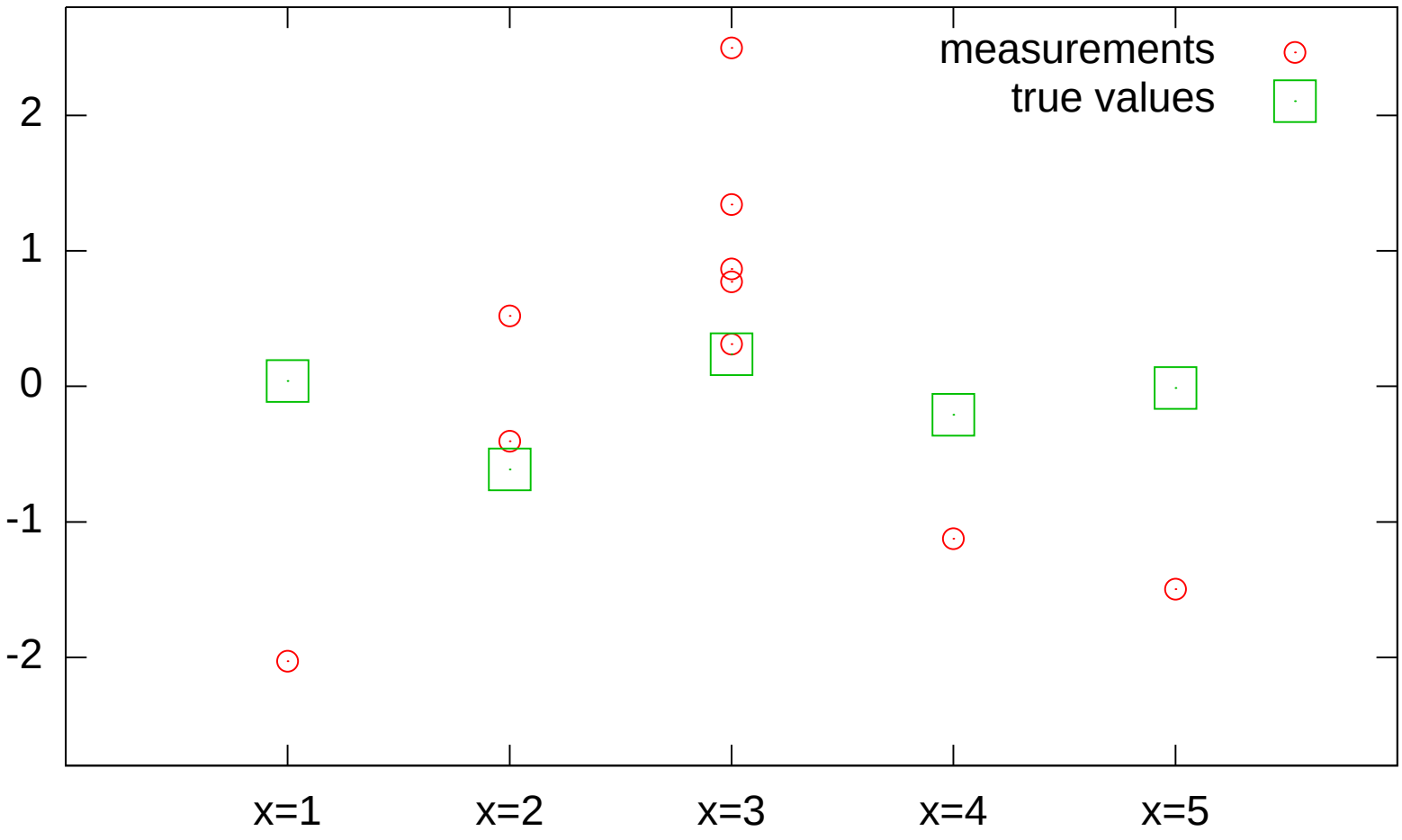
Example, Time 5



Example, Time 10



Example, Time 10



Outline

We define the expected reward of an algorithm in a problem instance

The expected reward from using algorithm π in problem instance θ is

$$R(\pi, \theta) = E^{\pi}[\theta_{\widehat{x}}|\theta]$$

Notes:

- We use the notation E^{π} to indicate that the expectation depends on π (the distribution of $\widehat{x}$ depends on π).
- Other rewards are possible, e.g., $\mathbb{E}^{\pi}[\mathbf{1}_{\{\widehat{x} \in \arg\max_x \theta_x\}}|\theta]$.
- Neither $x_1, \ldots, x_N$ and $y_1, \ldots, y_N$, nor the way that π chooses the x_n, appears in this expression explicitly, but $\widehat{x}$ depends on them implicitly.

Which algorithm should we choose?

- Suppose we have a collection of algorithms, $\pi_1, \ldots, \pi_\ell$.
- How should we choose which one to use?
- If we knew θ, we could choose the one with the best expected reward, $R(\pi, \theta)$,

$$\operatorname*{arg\,max}_{\pi \in \{\pi_1, \ldots, \pi_\ell\}} R(\pi, \theta).$$

- But we don't know θ. If we did, we could just choose $\widehat{x} \in \arg\max_x \theta_x$, and we wouldn't need to sample.

Which algorithm should we choose?

- One idea: Choose a few θ that seem "typical" for our problem class

$$\theta^{(1)}, \ldots, \theta^{(M)}$$

- Then compute the average expected reward across this class of problems,

$$\frac{1}{M} \sum_{m=1}^{M} R(\pi, \theta^{(m)})$$

- Then choose the algorithm for which this average expected reward is largest.

We can generalize by adding weights

- $\frac{1}{M}\sum_{m=1}^{M} R(\pi, \theta^{(m)})$ is one way to measure algorithm quality.
- We can generalize this by allowing weights on the problem instances. Choose $p(\theta^{(m)}) > 0$ for each $\theta^{(m)}$, and measure quality with

$$\sum_{m=1}^{M} p(\theta^{(m)}) R(\pi, \theta^{(m)})$$

- Without loss of generality, $\sum_{m=1}^{M} p(\theta_m) = 1$.
  - If not, we can replace each $p(\theta_m)$ by $p(\theta_m)/\sum_{m'=1} p(\theta_{m'})$ without changing the ordering of the algorithms.

We can generalize further by integrating with respect to a probability measure

- We can generalize this even further, by choosing a probability distribution P on θ and measuring quality as

$$\int_{[0,1]^k} R(\pi, \theta) P(d\theta)$$

- If P is a discrete distribution, with atoms at $\theta^{(m)}$, we recover the expression from the previous slide,

$$\sum_{m=1}^{M} p(\theta^{(m)}) R(\pi, \theta^{(m)})$$

- It also allows considering infinitely many θ: If P is a continuous distribution, with density p, this is

$$\int_{[0,1]^k} R(\pi, \theta) p(\theta) d\theta$$

Question: Which algorithm should we choose?
Answer: The one with the best average-case performance

- How should we choose an algorithm to use among $\pi_1, \ldots, \pi_\ell$?
- First choose a probability measure P over θ, which puts more weight on problem instances that are "typical" for what we want to solve.
  - P is called the **prior probability distribution** in Bayesian statistics.
- Then choose the algorithm with the best average-case performance with respect to P,

$$\operatorname*{arg\,max}_{\pi \in \{\pi_1, \ldots, \pi_\ell\}} \int_{[0,1]^k} R(\pi, \theta) P(d\theta)$$

This approach has advantages and disadvantages

- This approach allows us to fine-tune our choice of algorithm to the problem class we will use it for.
- If we have a lot of experience with our problem class, we often have a good idea about which problem instances are typical.
- . . . , but, if we don't know much about our problem class, it can be unclear which problem instances to choose.
- If the problem we wish to solve looks nothing like $\theta^{(1)}, \ldots, \theta^{(M)}$, either because we chose them poorly or because we were unlucky, the algorithm we choose might not work well.
- Another disadvantage: we do not get a worst-case guarantee on solution quality.

How should we choose our prior P?

One form for P that is both relatively flexible, and that gives analytically convenient expressions for later analysis, is:

- Let $\theta_x \sim \text{Normal}(\mu_{0x}, \sigma_{0x}^2)$ under P, with independence across x.
- Notation: let $\mu_0 = [\mu_{01}, \ldots, \mu_{0k}]$, $\sigma_0^2 = [\sigma_{01}^2, \ldots, \sigma_{0k}^2]$.
- How do we set μ_0 and σ_0^2?
  - One common choice is the **flat prior**: Take the limit as σ_{0x}^2 goes to ∞. In this limit, μ_{0x} doesn't matter, and all problem instances are weighted equally.
  - But if you know something more about your problem, you should include that in P.
  - One concrete way to do this is on the next slide.
- Drawback: if we know, e.g., that θ_x is non-negative, or θ_x is increasing in x, the normal distribution cannot incorporate this, and we may wish to use a different prior.

One way to choose the mean and variance of an independent normal prior

- Suppose that in addition to a simulator, we also have a back-of-the-envelope analytic approximation to θ_x, call it $\widehat{\theta}_x$.
- Before running your R&S algorithm, choose a small number of x at random, and take several replications from each to produce estimates of θ_x, call them $\overline{\theta}_x$.
- Set $\mu_{0x} = \widehat{\theta}_x + b$, where b is a bias correction term equal to the average of $\overline{\theta}_x - \widehat{\theta}_x$.
- Set σ^2_{0x} to the sample variance of the $\widehat{\theta}_x - \overline{\theta}_x$, minus an estimate of $\mathrm{Var}(\theta_x - \overline{\theta}_x)$.

We can find the algorithm with the best average-case performance using dynamic programming

- Rather than choosing among just the algorithms $\pi_1, \ldots, \pi_\ell$, can we find the algorithm with the **best possible** average-case performance with respect to P?
  - Terminology: we call such an algorithm **Bayes-optimal**.
- From a theoretical point of view, the answer is Yes.
- From a practical point of view, the answer is Maybe.
- For both, we use dynamic programming.

Outline

Average-case performance is the expected reward when nature draws the problem instance at random from the prior

The first step is to rewrite the average-case performance with respect to P as follows:

$$\begin{aligned}\int_{[0,1]^k} R(\pi,\theta)P(d\theta) &= E^{\pi}[R(\pi,\theta)|\theta \sim P] \\ &= E^{\pi}[E^{\pi}[\theta_{\widehat{x}}|\theta]|\theta \sim P] \\ &= E^{\pi}[\theta_{\widehat{x}}|\theta \sim P] \\ &= E^{\pi}[\theta_{\widehat{x}}]\end{aligned}$$

- Line 1: definition of expectation
- Line 2: recall $R(\pi,\theta)$ is the expected performance under π and θ
- Line 3: iterated conditional expectations
- Line 4 is notation we will use through most the rest of the talk. If no distribution is noted, and we don't condition on θ, then θ is drawn from P.

R&S is a problem in sequential decision-making under uncertainty

Thus, average-case performance $E^{\pi}[\theta_{\widehat{x}}|\theta \sim P]$ is the expectetd reward of

1. Nature chooses θ at random from P, then holds it fixed.
2. For $n = 1, \ldots, N$
   1. Use π to choose the next alternative to sample, x_n.
   2. Observe $y_n|\theta, x_n \sim \text{Normal}(\theta_{x_n})$.
3. Use π to choose $\widehat{x}$.
4. We earn reward $\theta_{\widehat{x}}$.

This is a problem in sequential decision-making under uncertainty.

Before we discuss how to choose $x_1, \ldots, x_N$, $\widehat{x_*}$, let's nail down the dynamics of the posterior distribution

- Because we have a normal prior with independent normal observations, after any number of samples n, the posterior on θ_x is also normal.

$$\theta_x | x_1, \ldots, x_n, y_1, \ldots, y_n \sim \text{Normal}(\mu_{nx}, \sigma^2_{nx}),$$

- The mean of the posterior, μ_{nx}, is a weighted average of the prior mean μ_{0x} and all observations of x,
- The variance of the posterior, σ^2_{nx}, is a decreasing function of the prior variance σ^2_{0x} and the number of observations of x.
- Notation: $\mu_n = [\mu_{n1}, \ldots, \mu_{nk}]$, $\sigma^2_n = [\sigma^2_{n1}, \ldots, \sigma^2_{nk}]$.
- For details, see, e.g., [DeGroot, 1970], or the references at the end of the talk.

Choosing $\widehat{x}_*$ is straightforward

We start from the end: let's work out how to choose $\widehat{x}_*$.

- If we select $\widehat{x}_* = x$ as the best, our expected reward is

$$E_N[\theta_x] = \mu_{Nx}.$$

  where the subscript N means that we are conditioning on the information available at time N, $x_{1:N}, y_{1:N}$.

- Therefore, the best choice for $\widehat{x}_*$ is

$$\widehat{x}_* \in \arg\max_{x=1,\ldots,k} \mu_{Nx}.$$

  and it has expected value $V_N = \max_x \mu_{Nx}$.

- We call $V_N = V_N(\mu_N, \sigma_N^2)$ the **value function** at time N. It tells us the expected value that we will receive if we act optimally starting with the given posterior distribution.

How should we choose x_{N-1}?

- From the last slide, the value we receive at time N is $V_N = \max_x \mu_{Nx}$.
- At time N, we don't know V_N yet, because it depends on μ_N, and we don't know y_N.
- We can simply take the expectation of V_N to compute the value we will receive.

$$\mathbb{E}_{N-1}[V_N|x_N].$$

- This depends on x_N, because the distribution of Y_N depends on x_N, μ_N depends on Y_N and x_N, and V_N depends on μ_N.
- The optimal choice of x_N is the one that maximizes the expected value we receive,

$$\arg\max_{x_N} \mathbb{E}[V_N|x_N],$$

and this maximal expected value is

$$V_{N-1} = V_{N-1}(\mu_{N-1}, \sigma^2_{N-1}) = \max_{x_N} \mathbb{E}_{N-1}[V_N|x_N].$$

In principle, we can repeat this to find the optimal rule for every x_n

We iterate backward $n = N, N-1, N-2, \ldots, 1$, where in each stage n:

- We computed $V_{n+1}(\mu_{n+1}, \sigma^2_{n+1})$ in the previous stage.
- The optimal choice for x_{n+1} is

$$x_{n+1} \in \arg\max_{x_{n+1}} \mathbb{E}_n[V_{n+1}(\mu_{n+1}, \sigma^2_{n+1})|x_{n+1}]$$

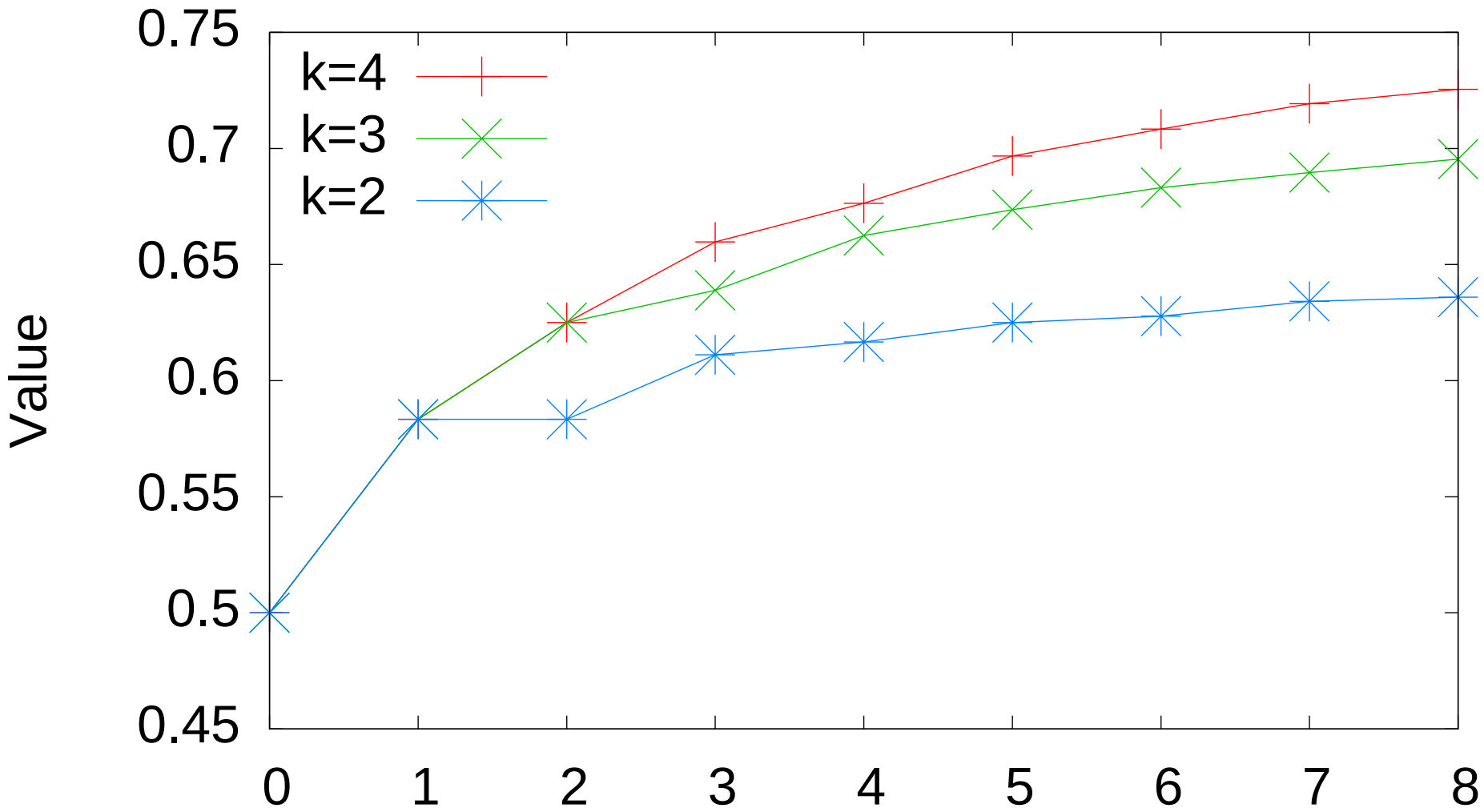
- The value of this decision is

$$V_n(\mu_n, \sigma_n) = \max_{x_{n+1}} \mathbb{E}_n[V_{n+1}(\mu_{n+1}, \sigma^2_{n+1})|x_{n+1}].$$

This is **dynamic programming**.

We can solve the DP exactly for small problems

Here is the value function for a ranking and selection problem with Bernoulli (0/1) observations, and independent beta prior distributions.



For large problems, this does not work because of the curse of dimensionality

- To use dynamic programming, we need to compute and store $V_n(\mu_n, \sigma_n^2)$ for each possible value of μ_n and σ_n^2. (We need to compute V_n for every n, but at any given time we only need V_n and V_{n+1} in memory.)
- There are infinitely many possible values for μ_n. We can discretize, but it is a vector in k dimensions, and so discretizing into m pieces in each dimension allows for m^k possible values.
- σ_n^2 only takes finitely many values, since $(\sigma_{nx}^2)^{-1}/\lambda^{-2}$ is the number of samples of alternative x, but there are still $k^n/n!$ possible values.
- For large values of k (say, $k > 10$), solving the dynamic program is computationally intractable.

Outline

Outline

Approximate the value functions

- The curse of dimensionality causes difficult in all large-scale dynamic programs, not just those appearing in optimal learning problems.
- One very common approach is to approximate the value function through a linear combination of basis functions, or as a piecewise linear convex function.
- Good textbooks on this include
  [Powell, 2007, Judd, 1998, Bertsekas and Tsitsiklis, 1996].
- These technques are common in other dynamic programming applications, but have not been applied to optimization via simulation problems (at least, not extensively). It is an interesting area for further research.

Outline

Value of Information / Knowledge-gradient (KG) methods

- One very powerful technique is to do the following thought experiment: "What would be optimal, if this were my last opportunity to take an observation, and then I had to make a decision immediately afterward?"
- This gives you an observation that would be optimal according to the dynamic program, if you had one step to go.
- Then, take this observation, even if you have more than one step to go.

VOI / KG Methods applied to R&S

In the context of R&S, this idea can be applied as follows:

$$x_{n+1} \in \arg\max_{x_{n+1}} E_n[V_N(\mu_{n+1}, \sigma^2_{n+1})|x_{n+1}]$$
$$= \arg\max_{x} E_n[\max_i \mu_{n+1,i}|x_{n+1}]$$

This decision is unchanged if we subtract the constant $\max_i \mu_{ni}$ from this objective,

$$x_{n+1} \in \arg\max_{x} E_n[\max_i \mu_{n+1,i}|x_{n+1}] - \max_i \mu_{ni}$$

VOI / KG Methods applied to R&S

Put another way, this policy values each potential measurement x according to

$$\begin{aligned}\text{value of measuring } x &= \mathbb{E}_n\left[\max_i \mu_i^{n+1} \mid x_{n+1} = x\right] - \max_i \mu_i^n \\ &= \mathbb{E}\left[\text{best we can do with the measurement}\right] \\ &\quad - \left(\text{best we can do without the measurement}\right).\end{aligned}$$

This is the value of information (VOI) for the measurement, also called the KG factor. The policy then performs the measurement with the largest VOI / KG factor.

VOI / KG Methods applied to R&S with independent normal observations and an independent normal prior

- In the context of the R&S problem with independent normal observations and an independent normal prior, VOI methods were first applied by [Gupta and Miescke, 1996] to create the following algorithm:
- The VOI / KG factor for measuring alternative x at time n is

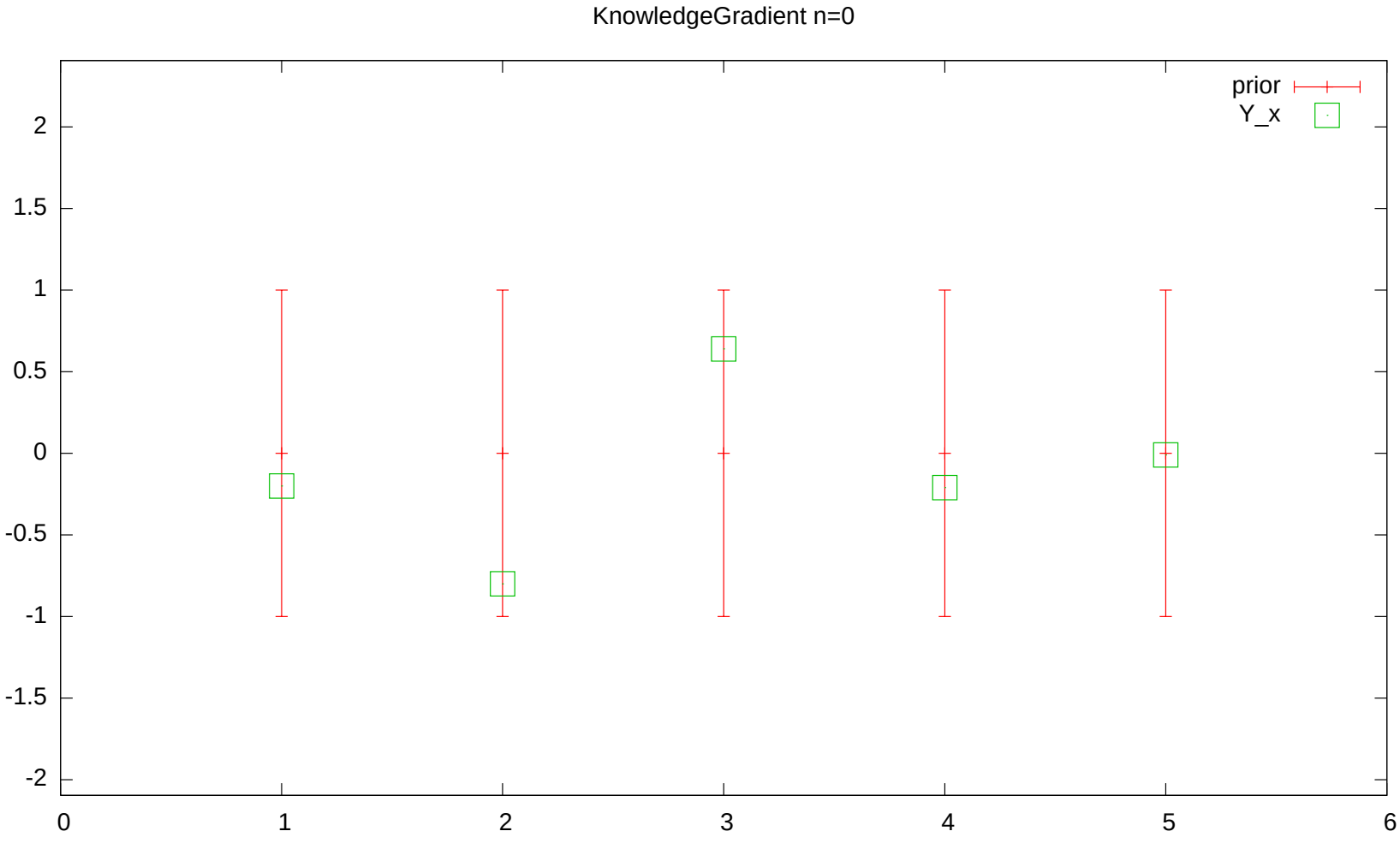
$$\tilde{\sigma}_x^n f\left(-\frac{\Delta_x^n}{\tilde{\sigma}_x^n}\right)$$

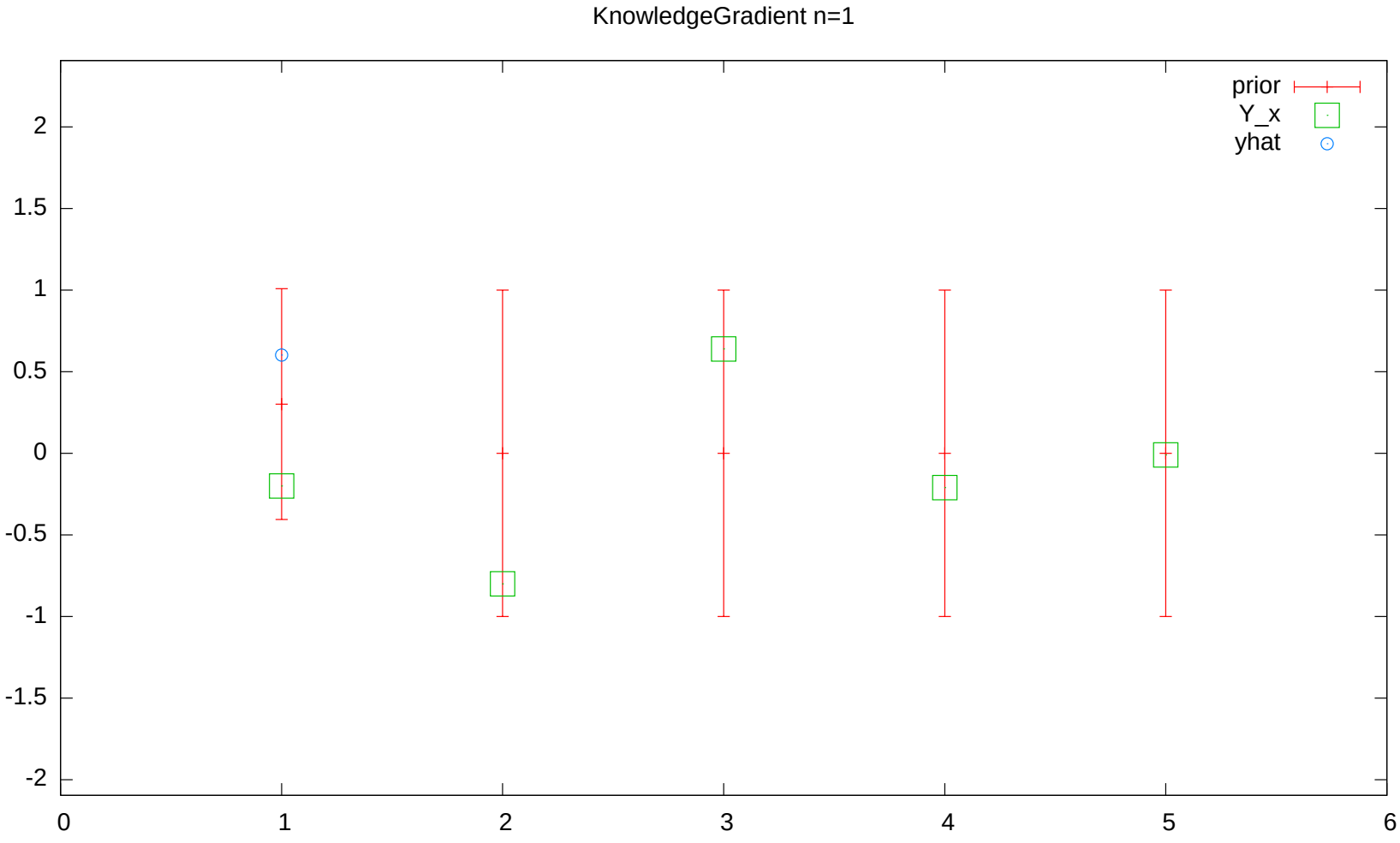
where

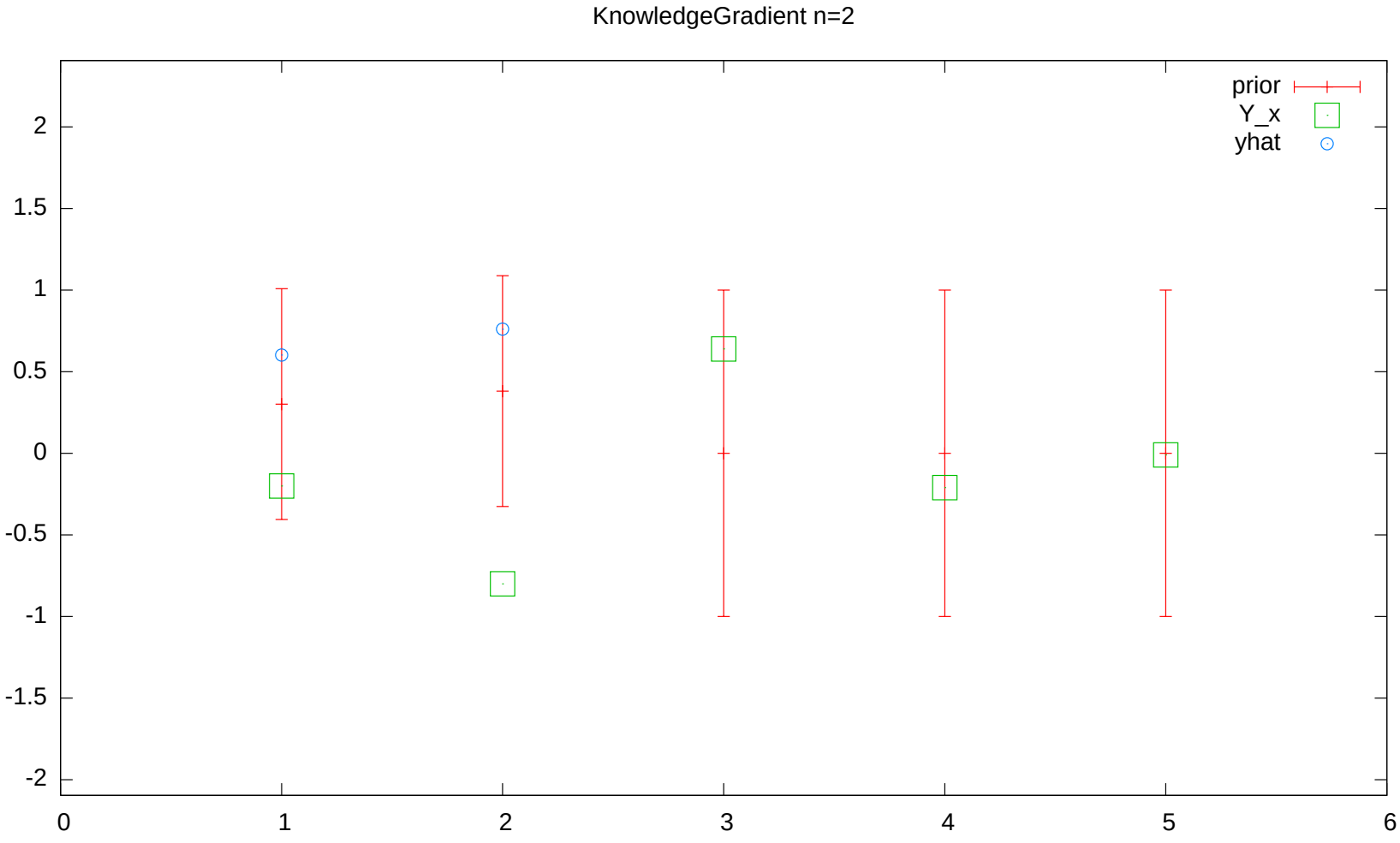
$$\tilde{\sigma}_x^n = (\sigma_x^n)^2/\sqrt{(\sigma_x^n)^2+\lambda^2}, \quad \Delta_x^n = |\mu_x^n - \max_{x' \neq x} \mu_{x'}^n|,$$

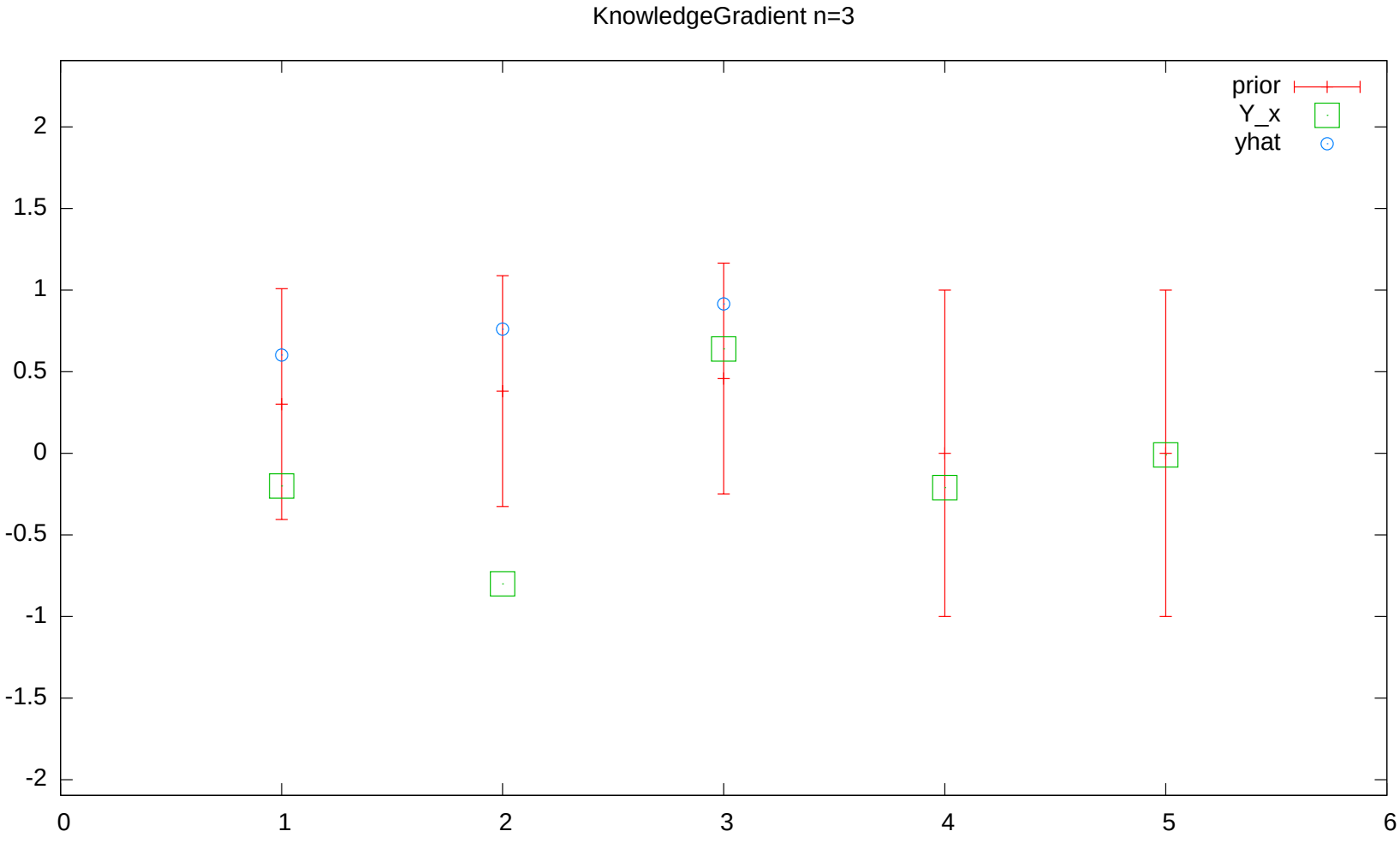
$f(c) = c\Phi(c) + \varphi(c)$, Φ is the normal cdf, φ is the normal pdf.

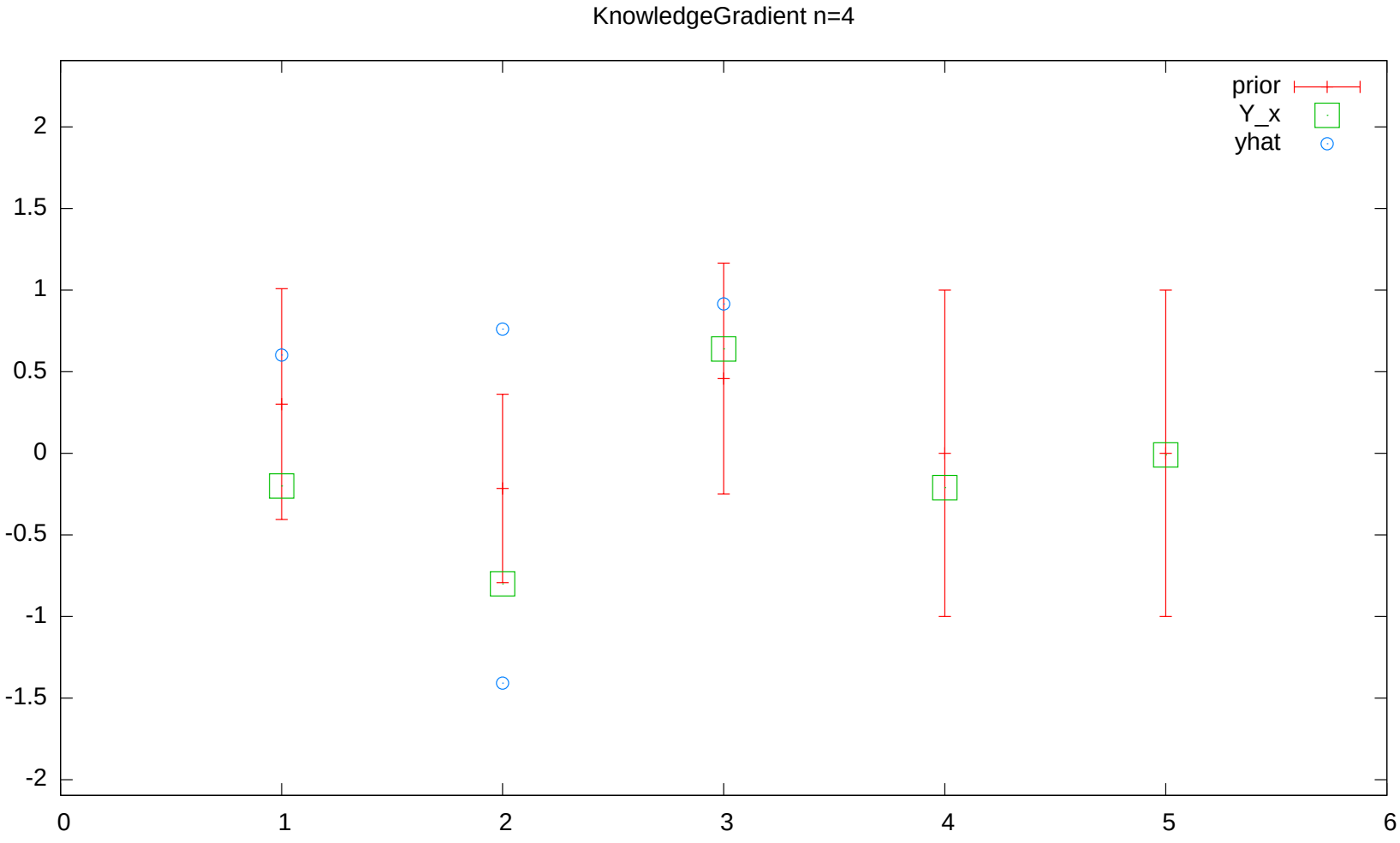
- This algorithm was introduced with the name of the $(R1, \ldots, R1)$ algorithm by [Gupta and Miescke, 1996], and later studied under the name of KG algorithm by [Frazier et al., 2008].

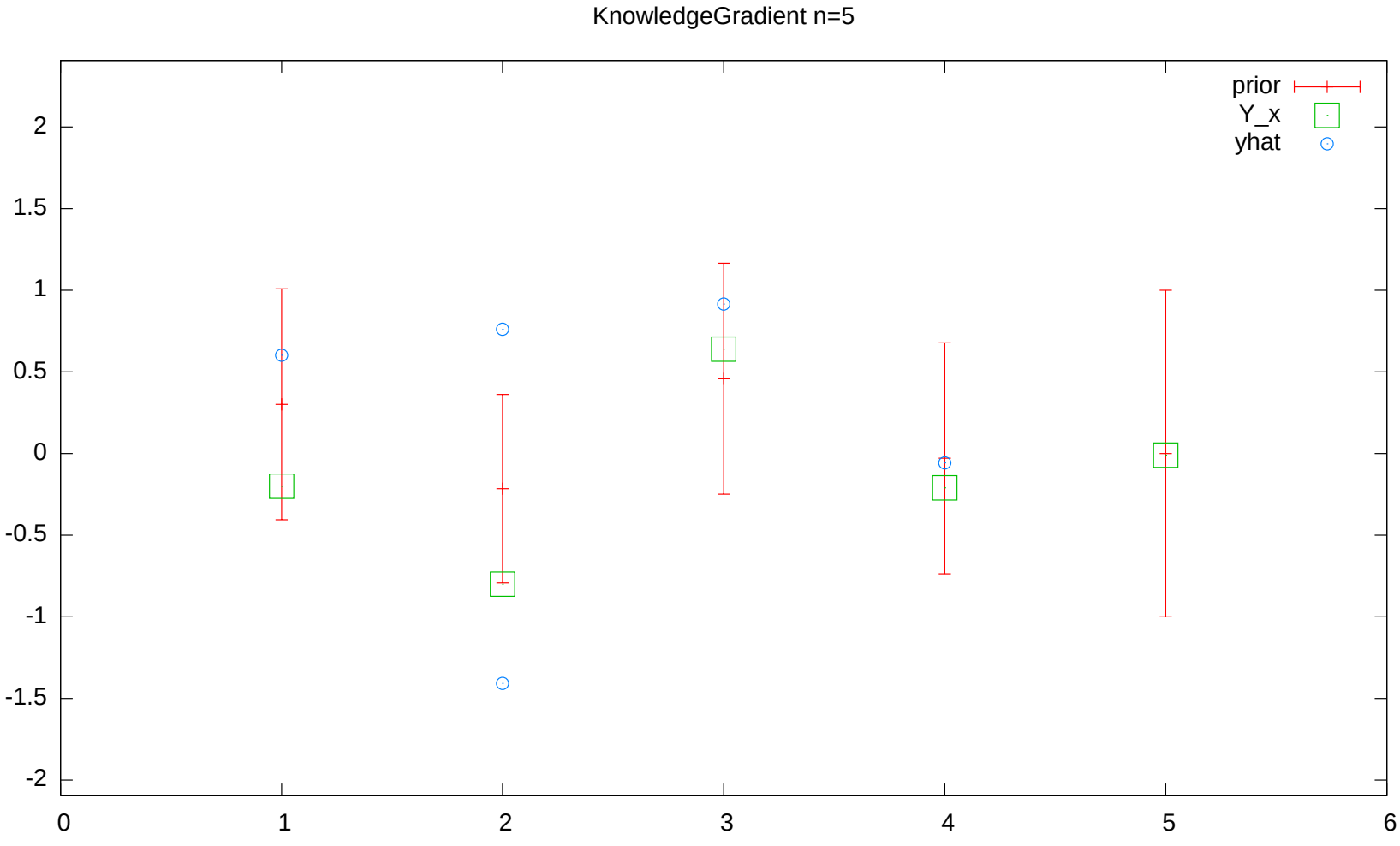

KnowledgeGradient n=0
prior
Y_x
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

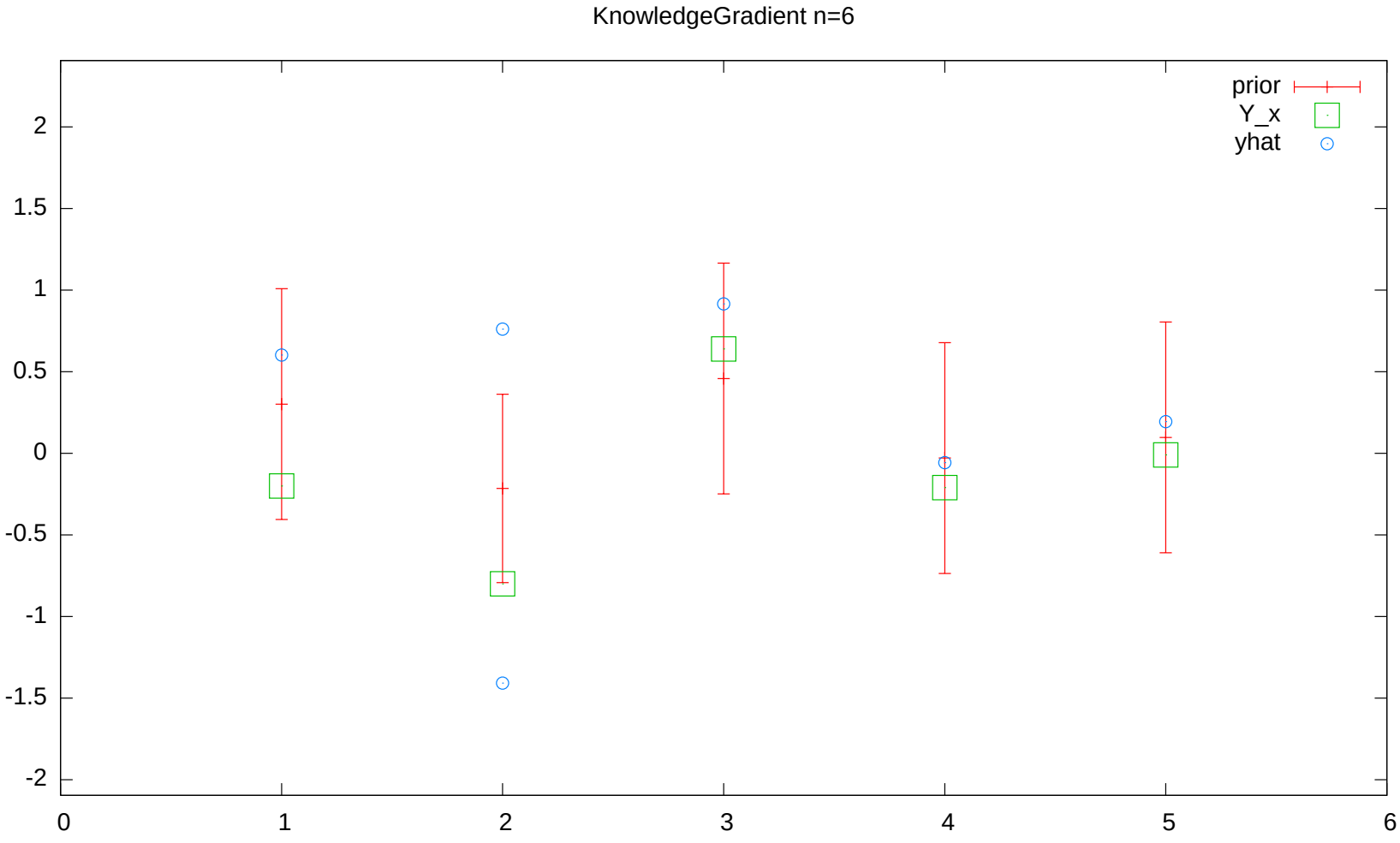

KnowledgeGradient n=1
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

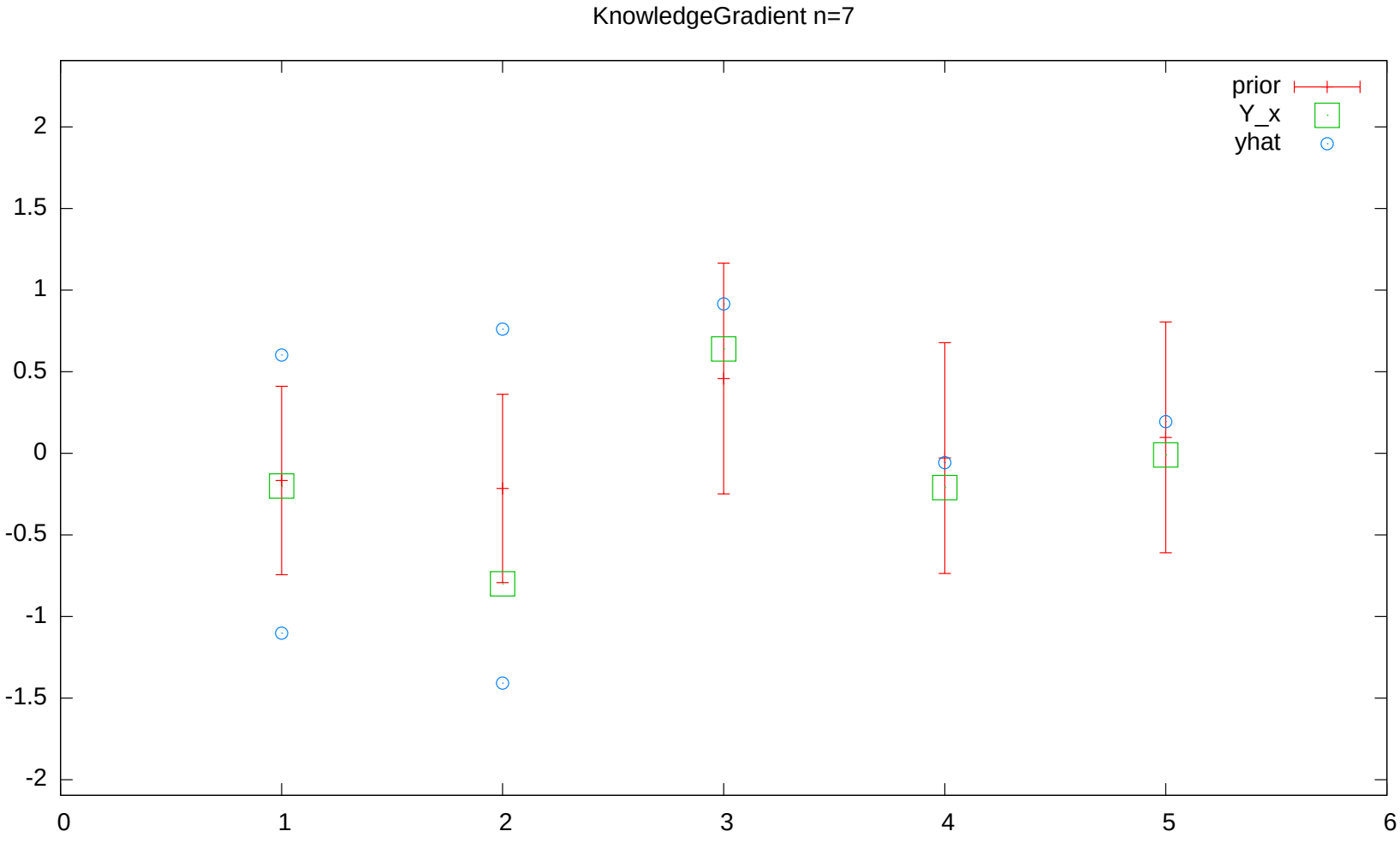

KnowledgeGradient n=2
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

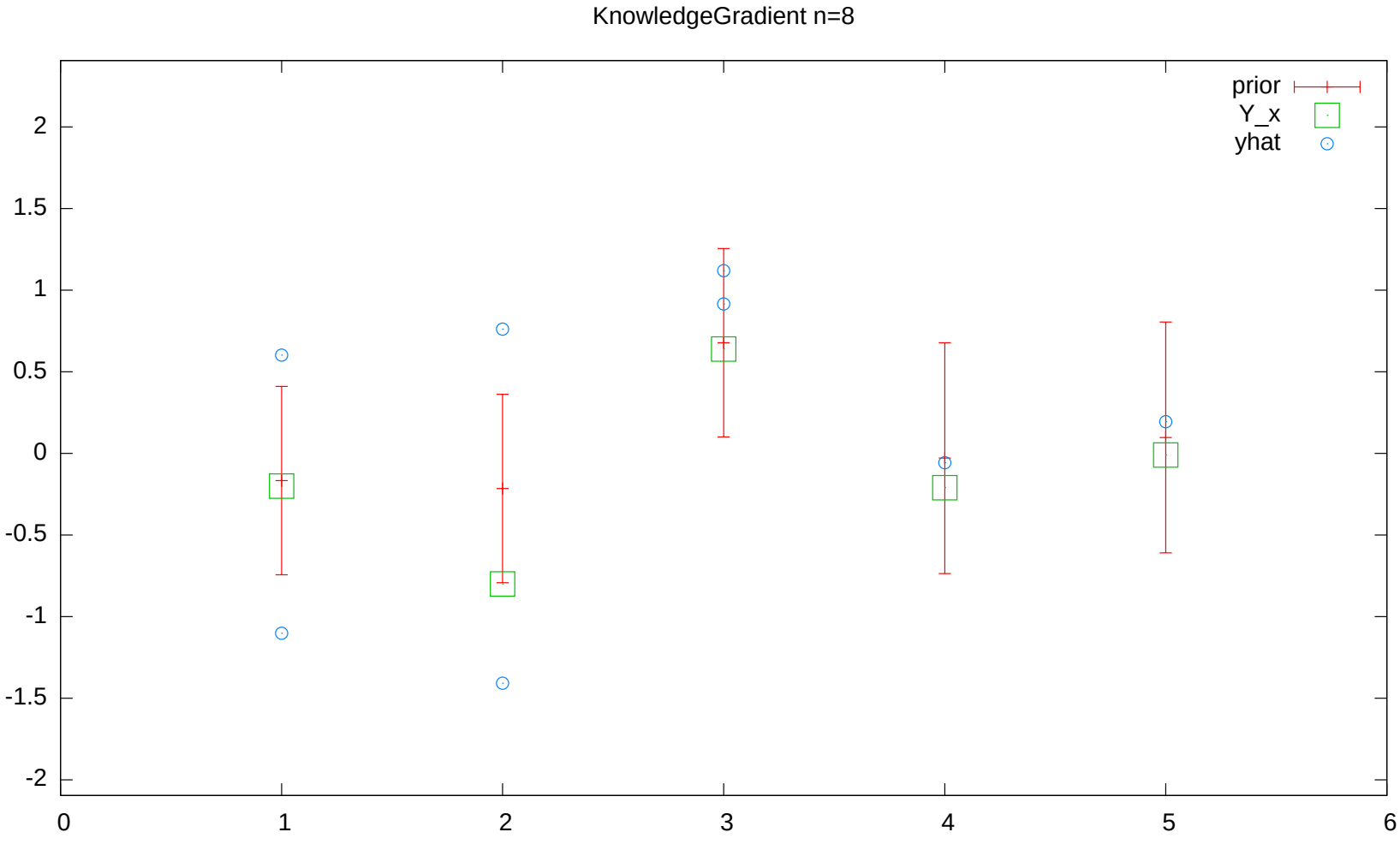

KnowledgeGradient n=3
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

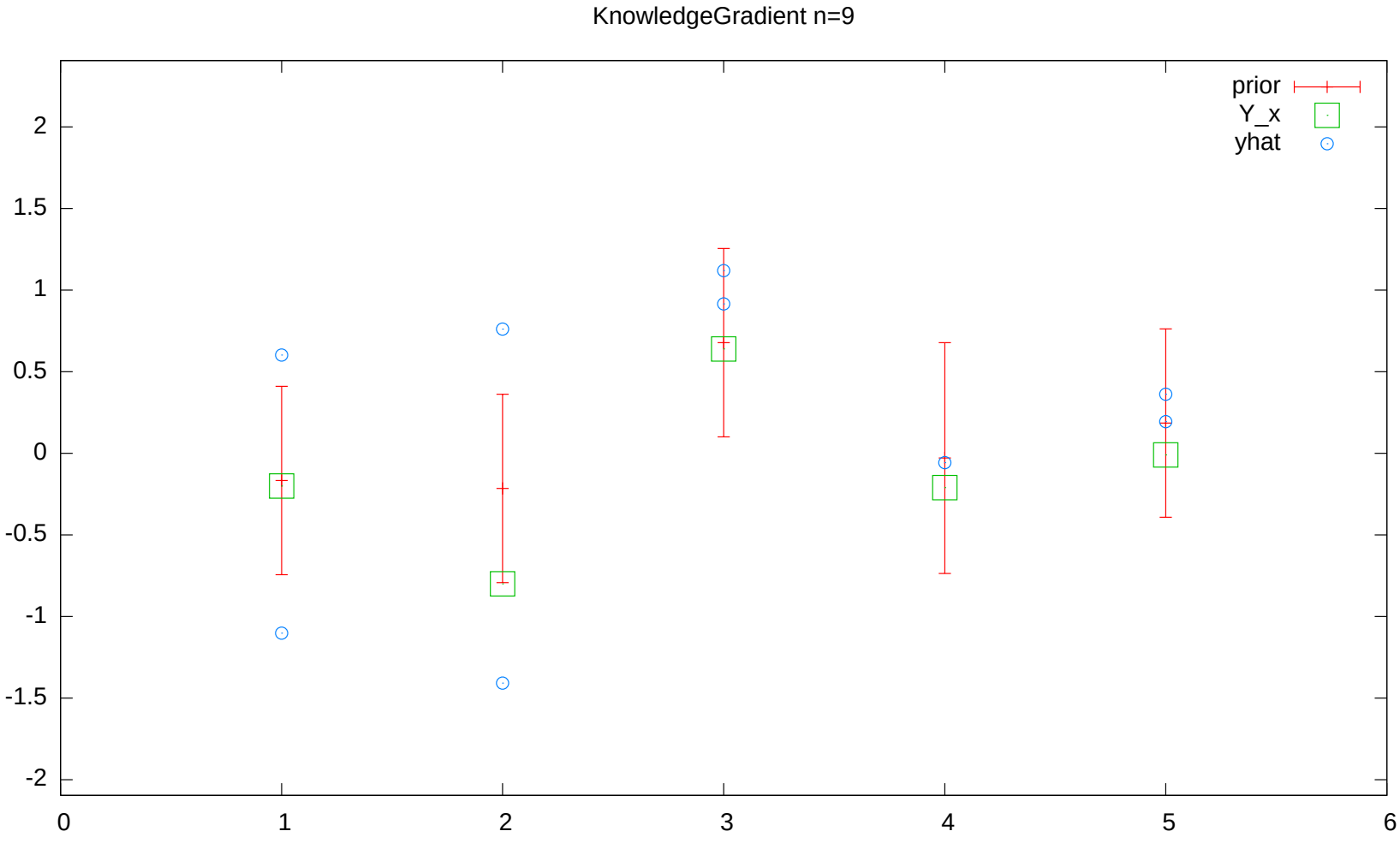

KnowledgeGradient n=4
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

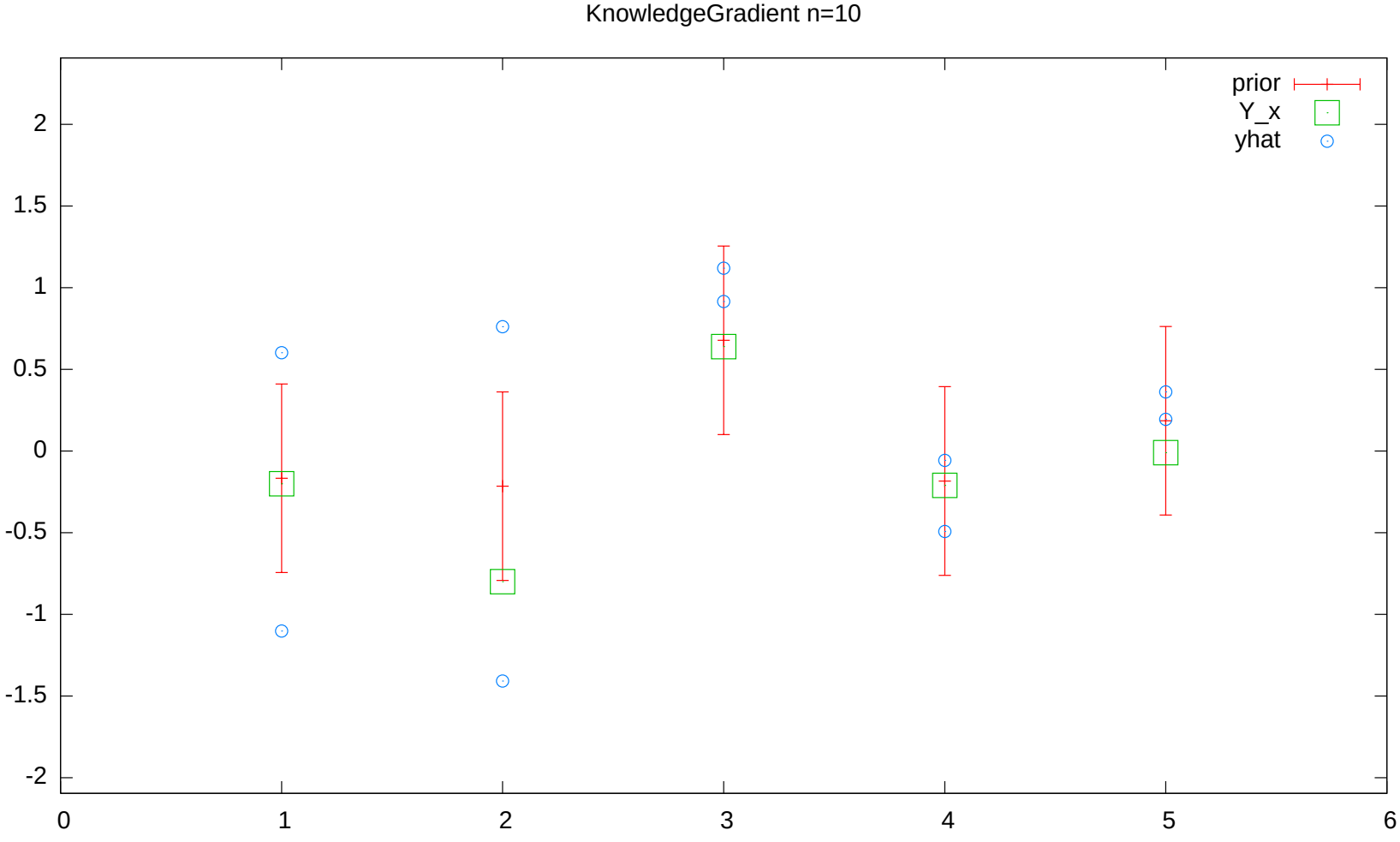

KnowledgeGradient n=5
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

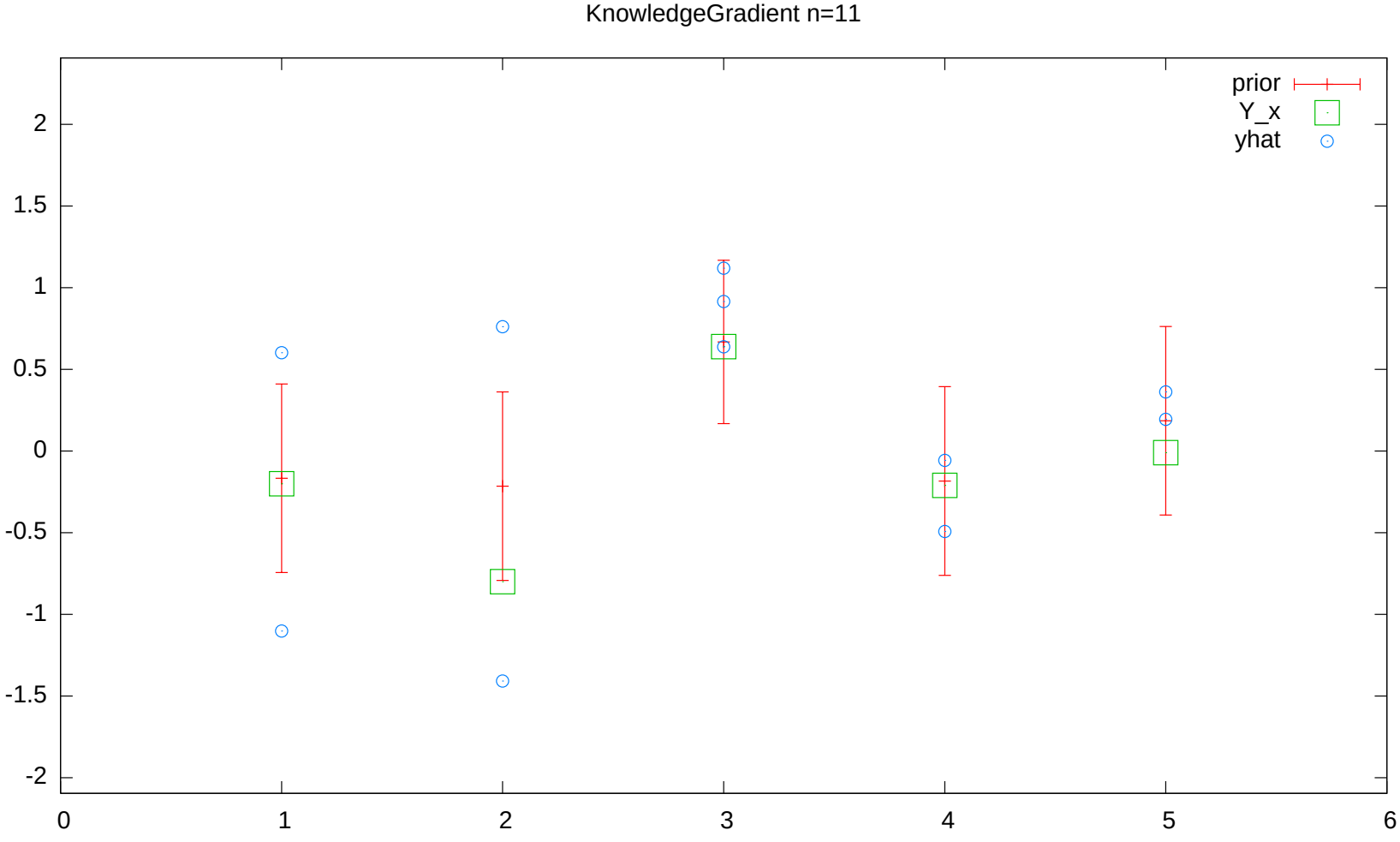

KnowledgeGradient n=6
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

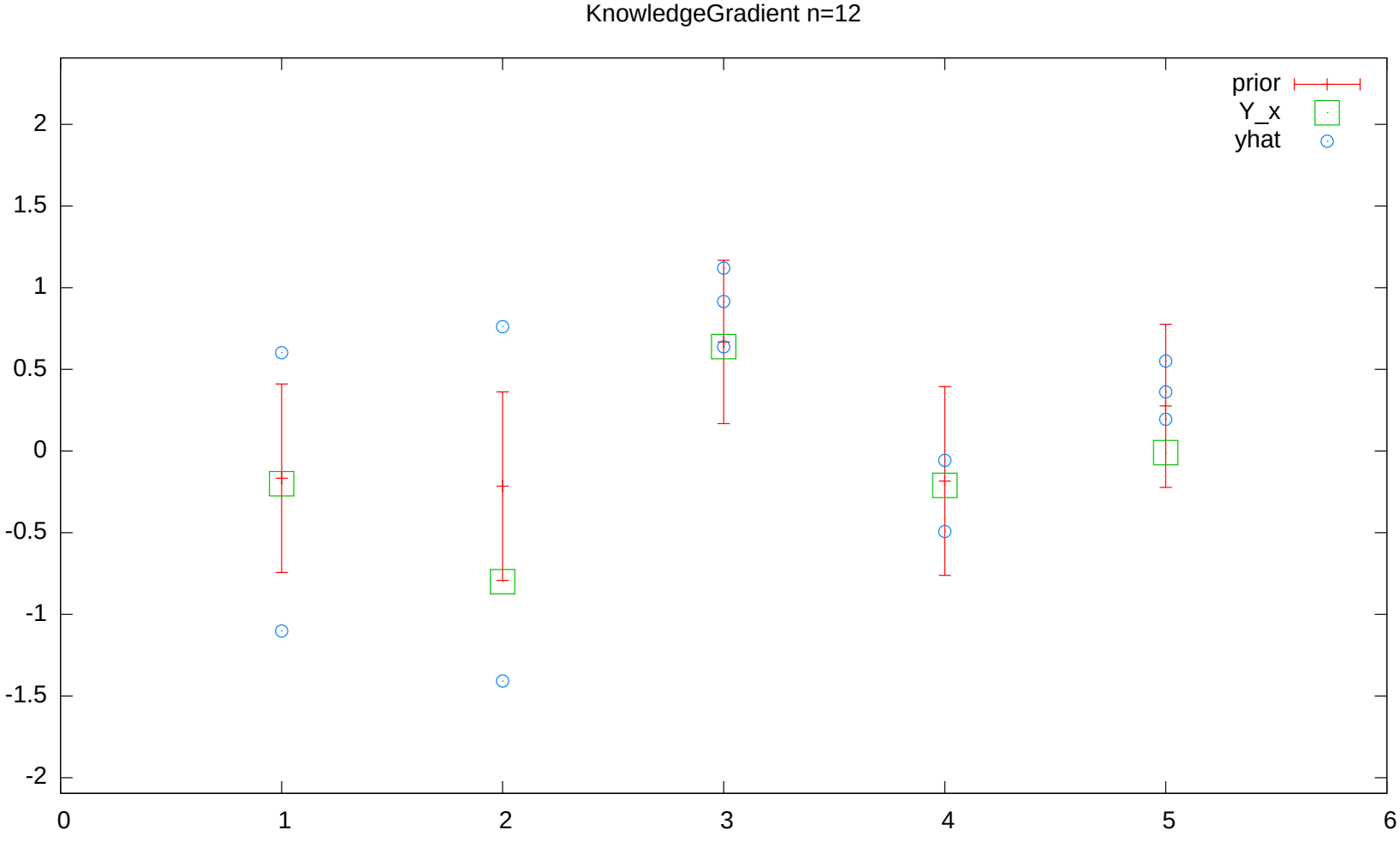

KnowledgeGradient n=7
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

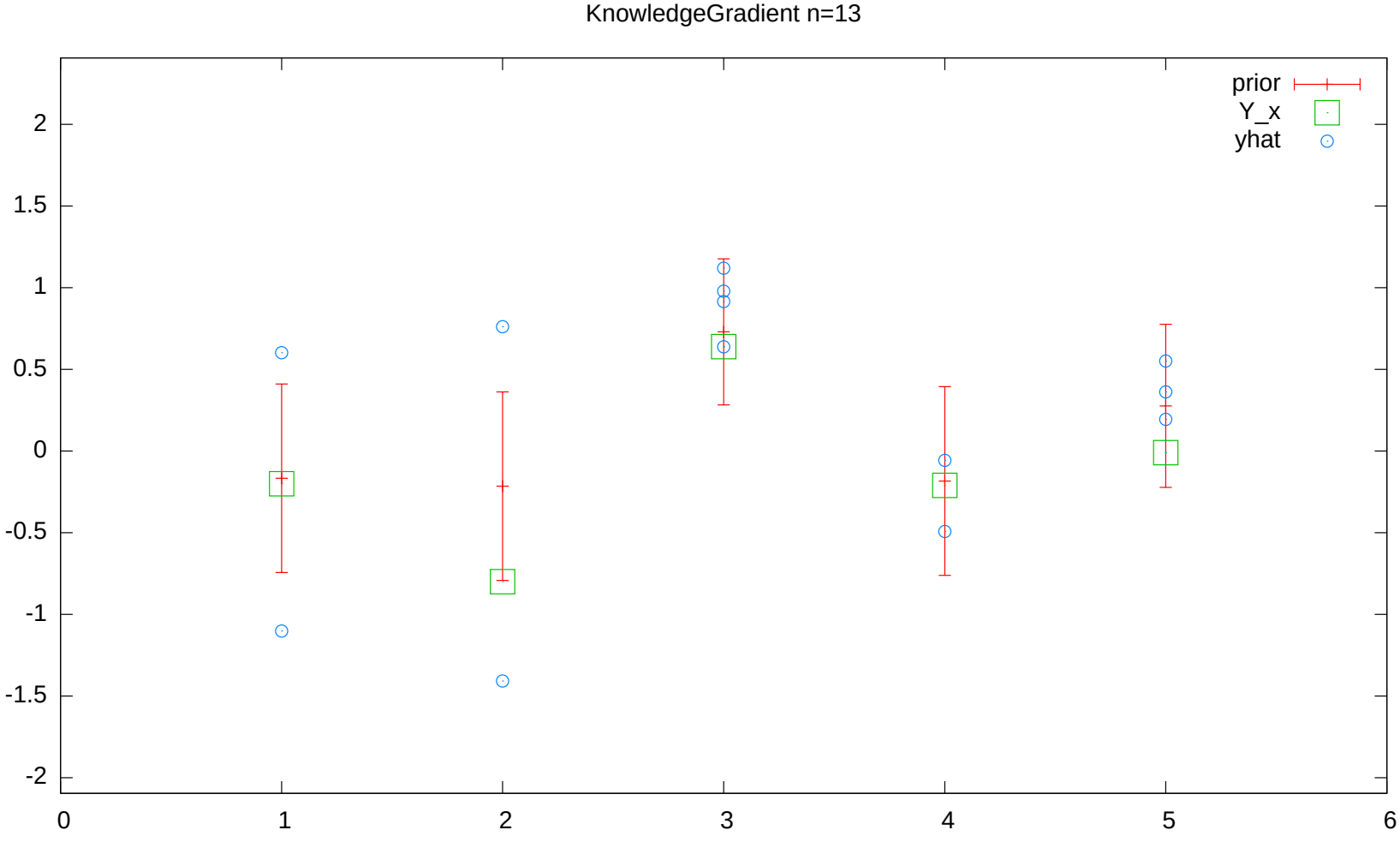

KnowledgeGradient n=8
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

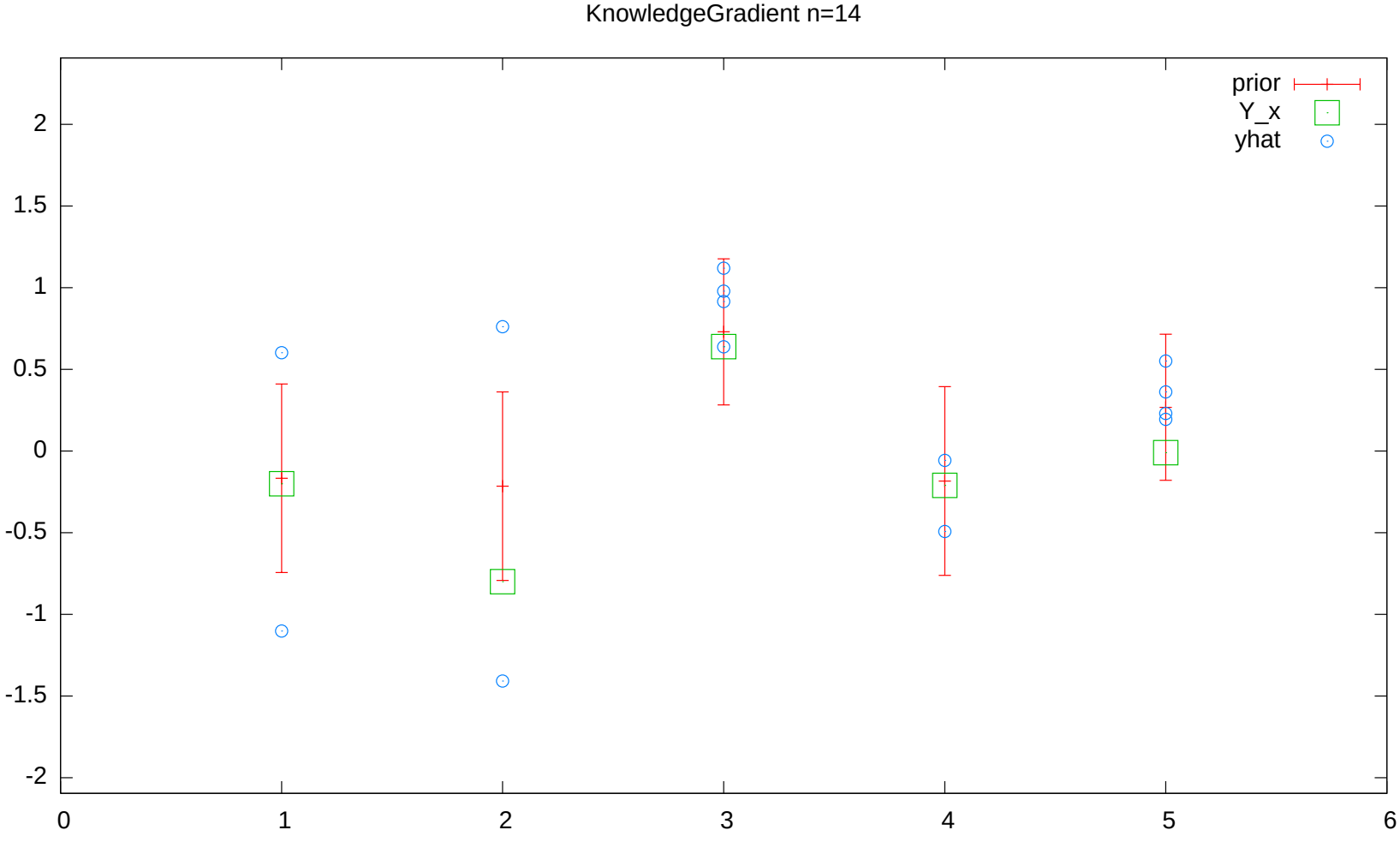

KnowledgeGradient n=9
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

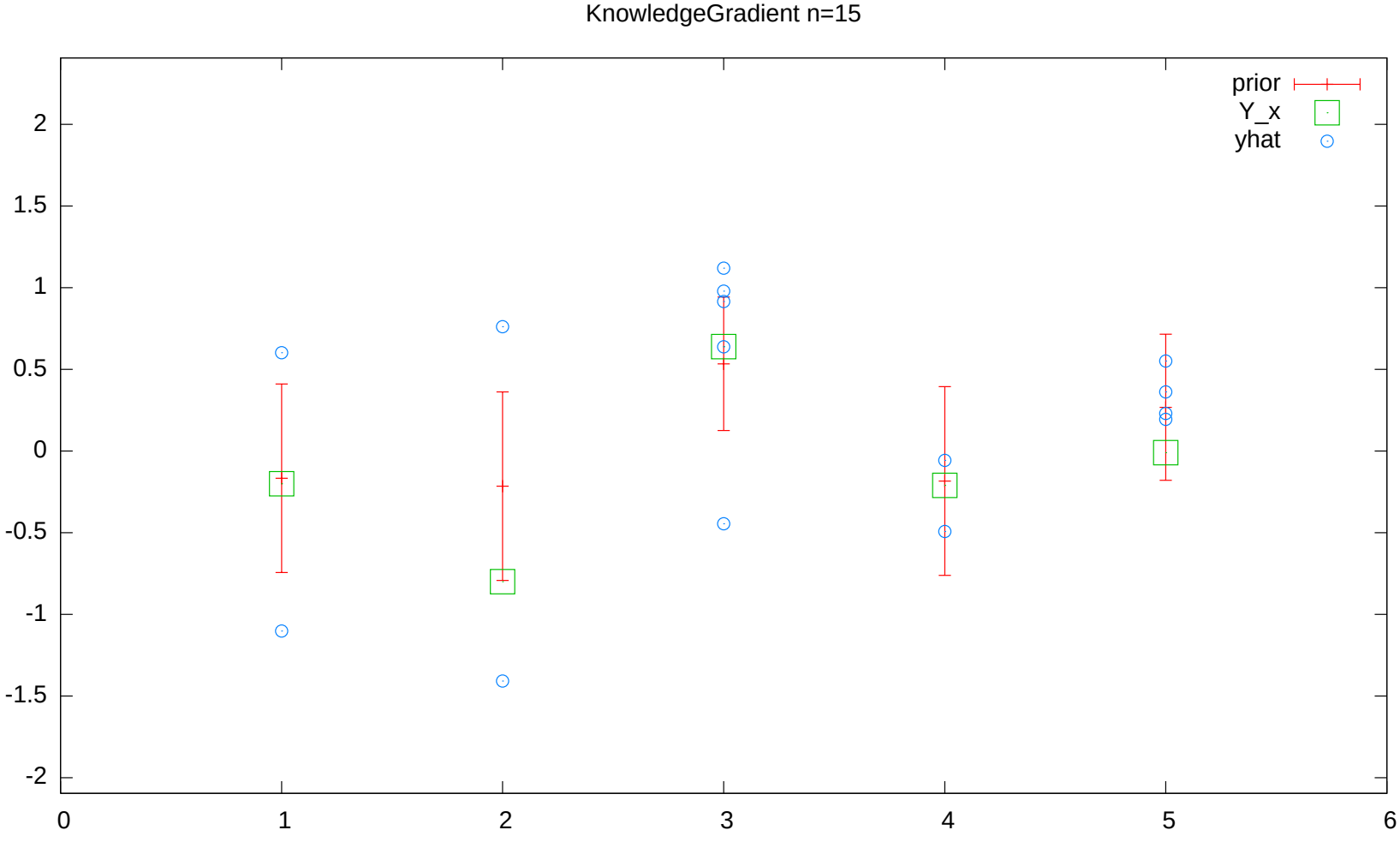

KnowledgeGradient n=10
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

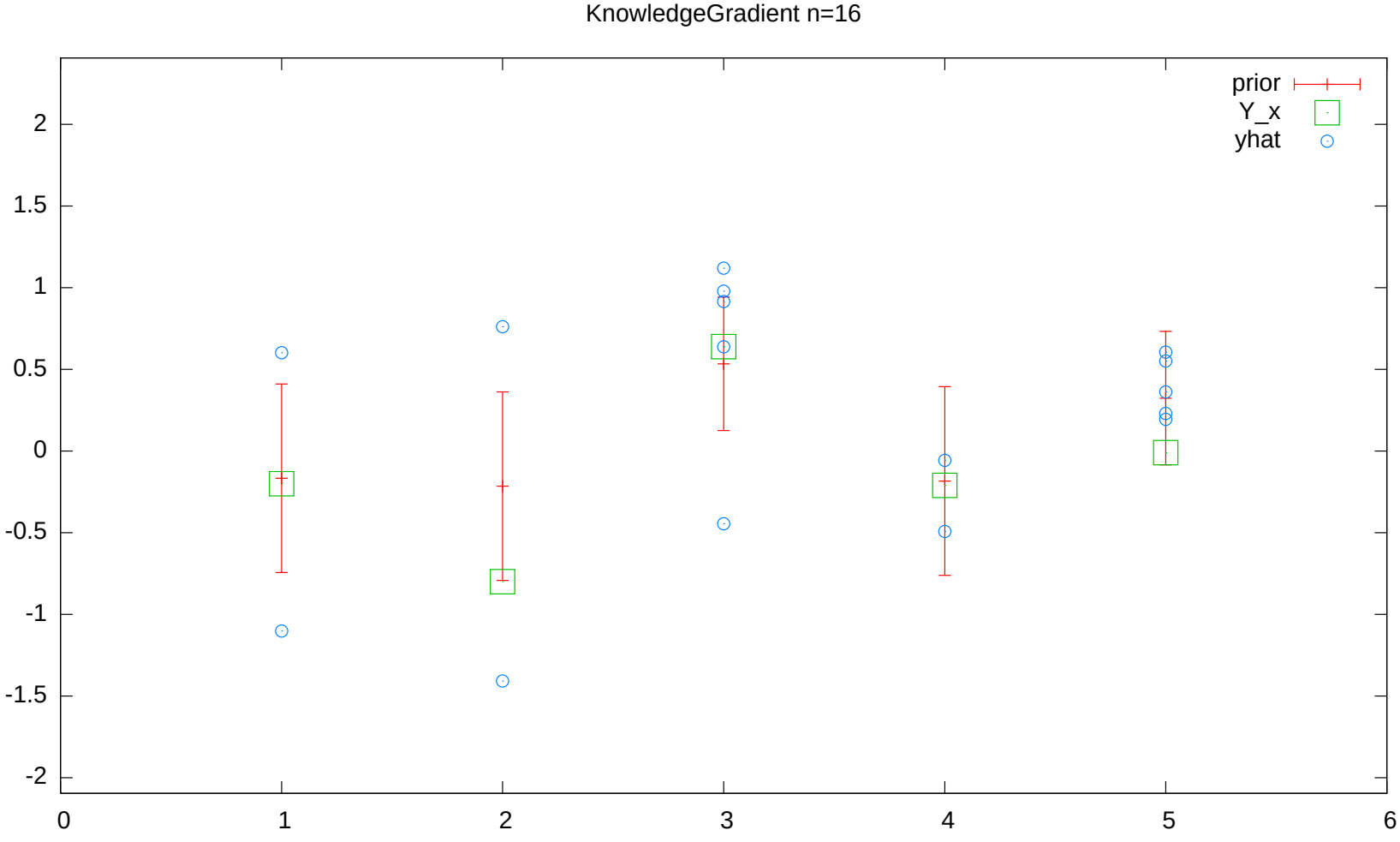

KnowledgeGradient n=11
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

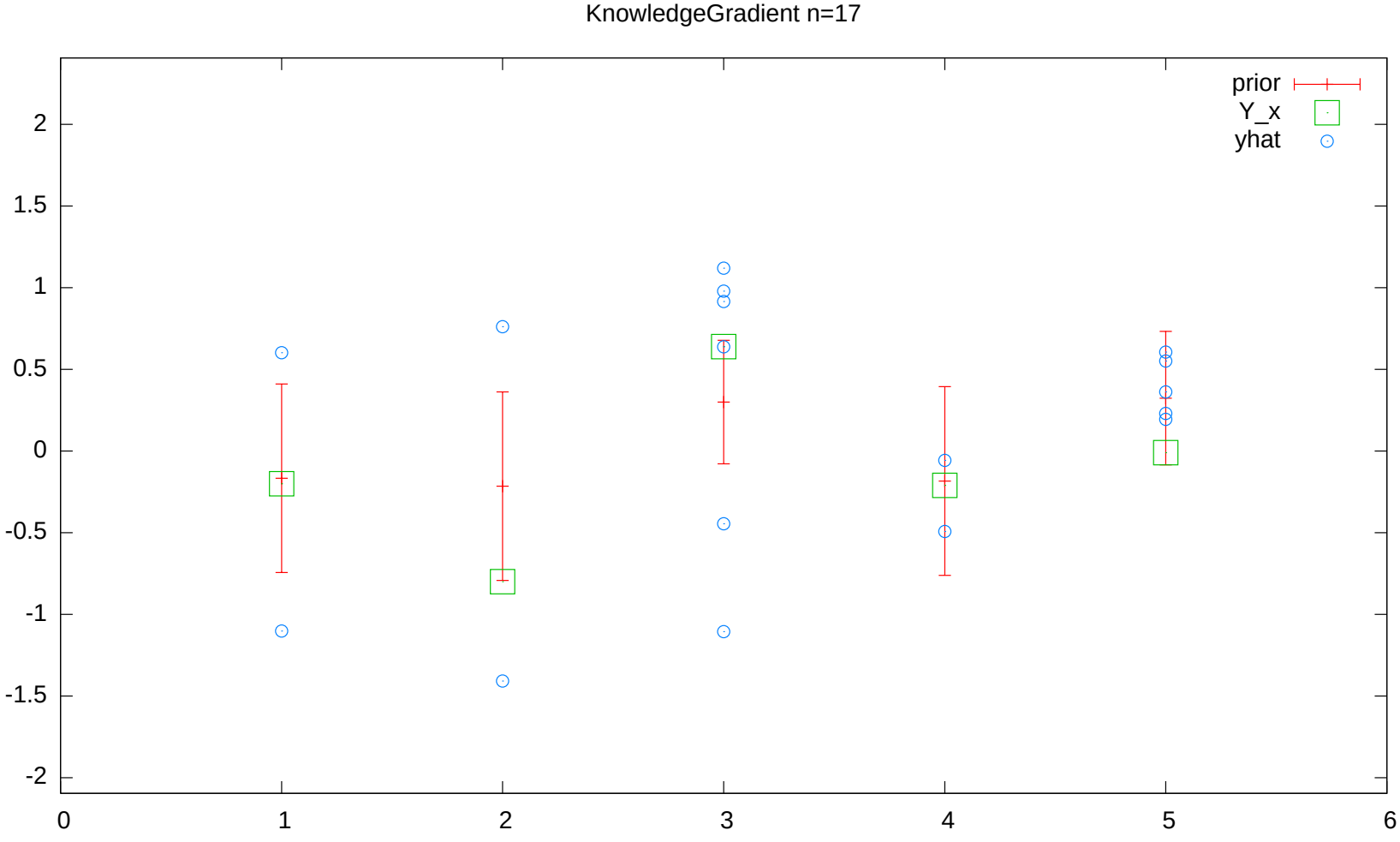

KnowledgeGradient n=12
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

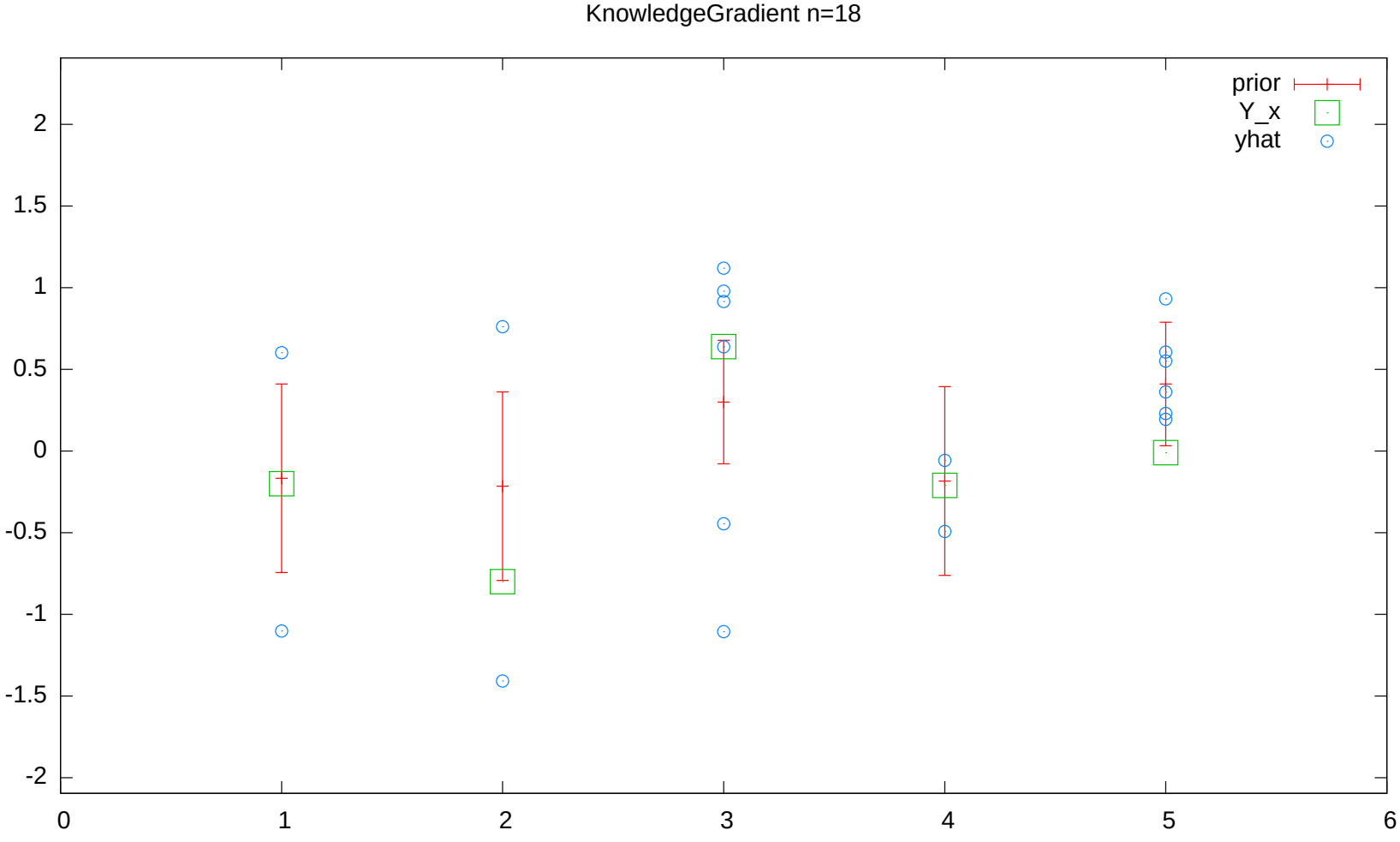

KnowledgeGradient n=13
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

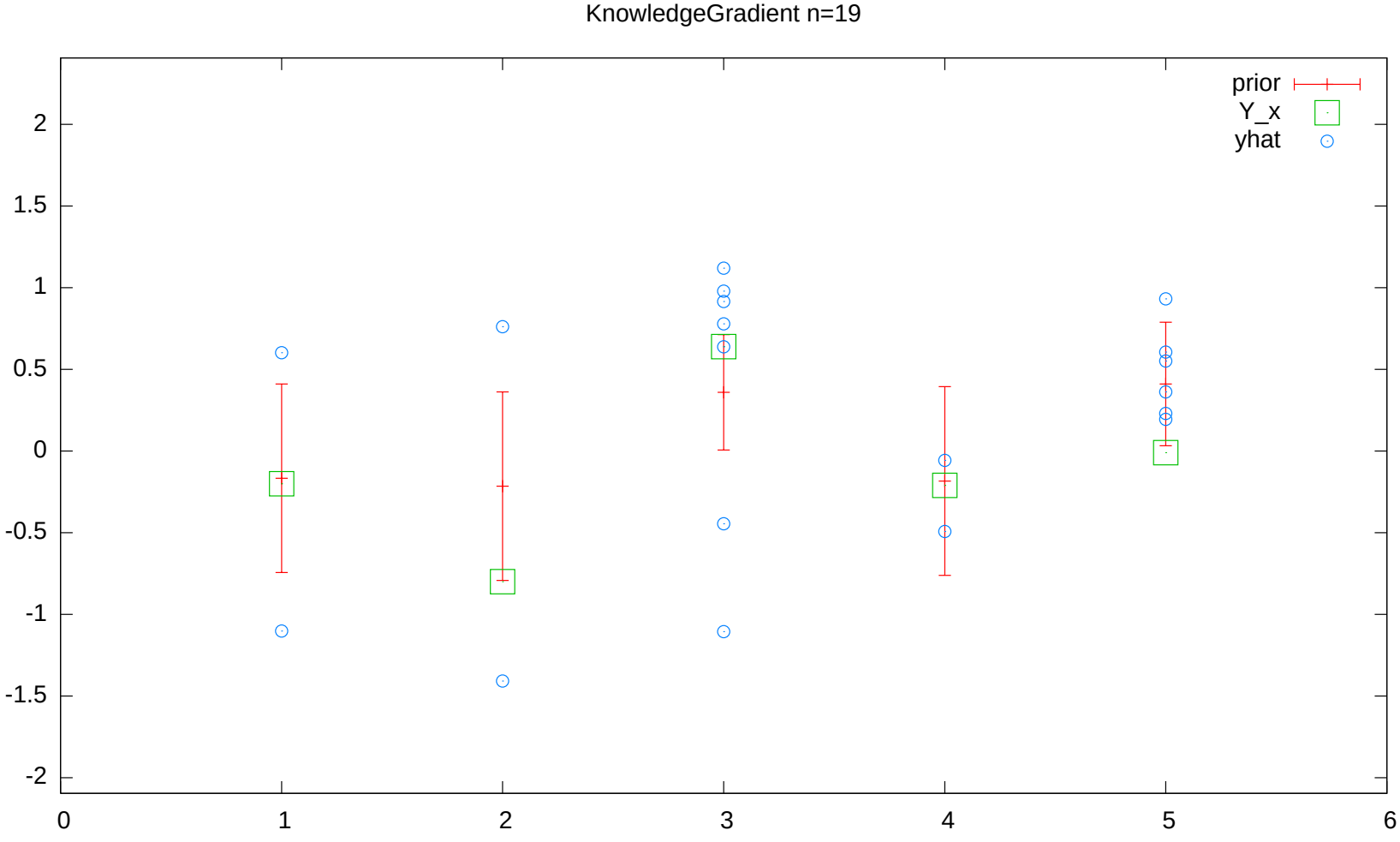

KnowledgeGradient n=14
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

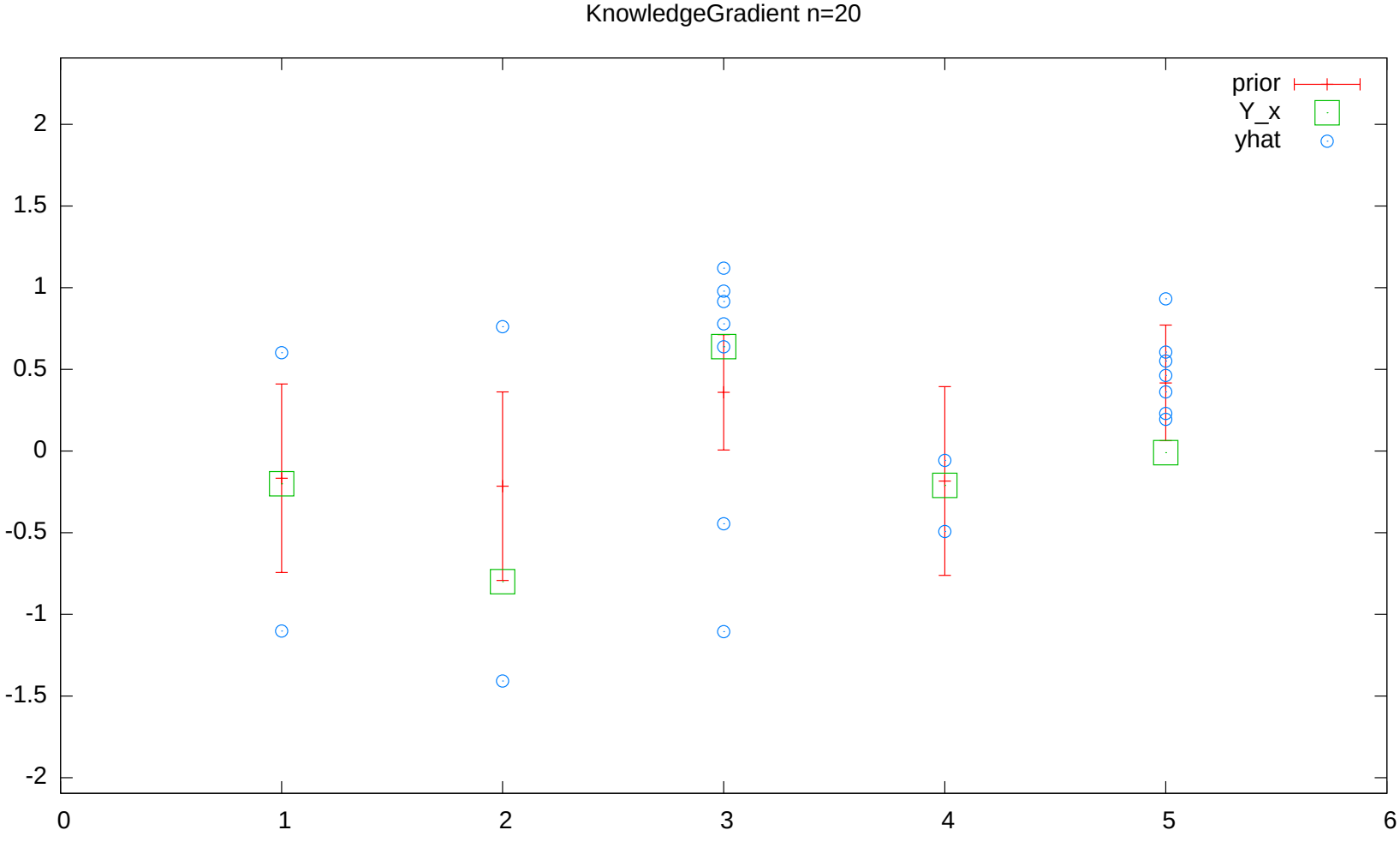

KnowledgeGradient n=15
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6


KnowledgeGradient n=16
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6


KnowledgeGradient n=17
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6


KnowledgeGradient n=18
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

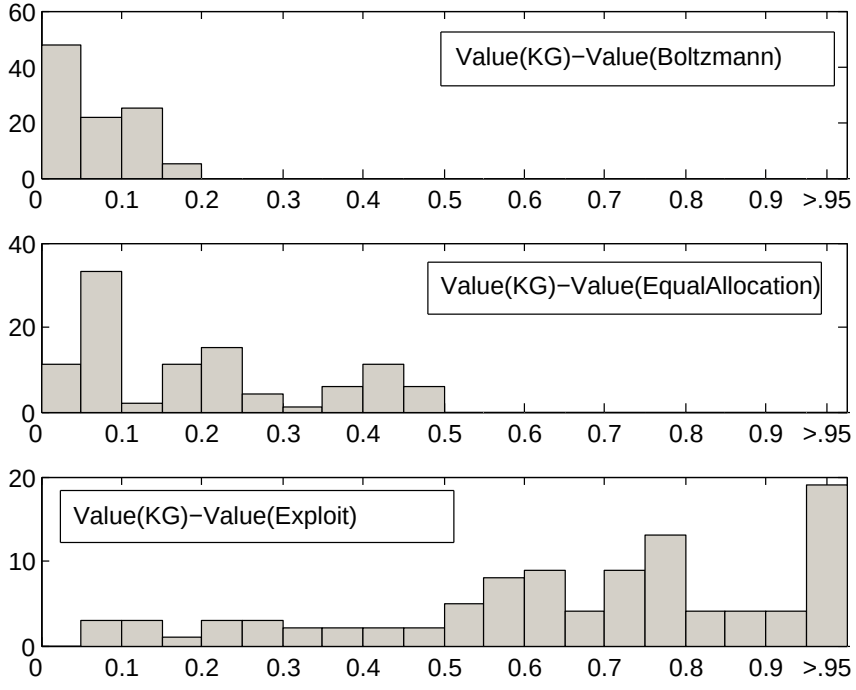

KnowledgeGradient n=19
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

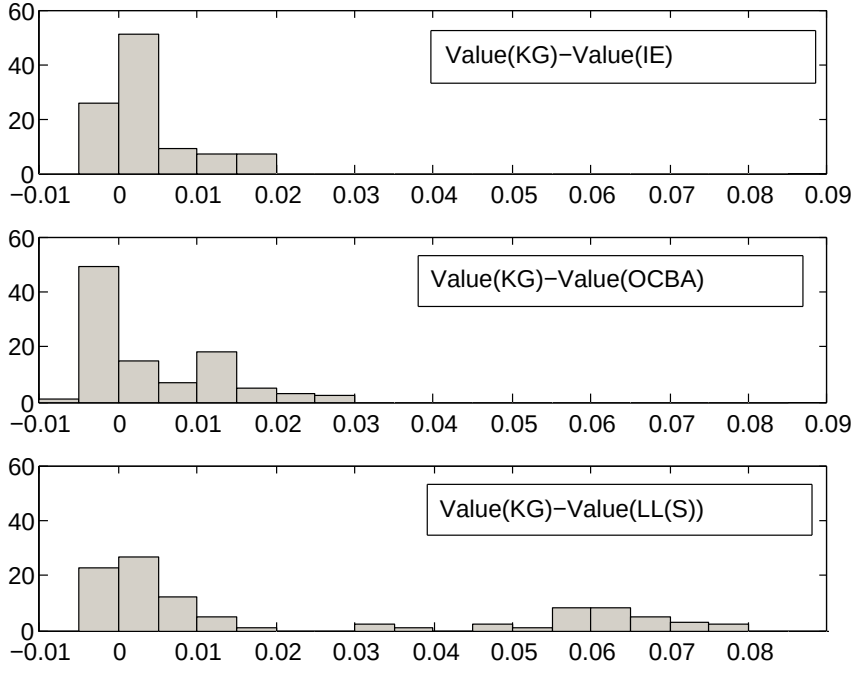

KnowledgeGradient n=20
prior
Y_x
yhat
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1
2
3
4
5
6

One can also do this for the case with unknown sampling variance

- We have assumed common known sampling variance.
- Heterogeneous known sampling variance is a minor tweak.
- To deal with unknown sampling variance, put a prior on the sampling variance. Extra work is required, but the ideas are similar. See [Chick et al., 2007, Chick et al., 2010].

This algorithm works well





Histogram of the sampled difference in value for competing policies aggregated across the 100 randomly generated problems.

Versions of KG that are "Less Myopic"

The KG method discussed thus far is a greedy method. The following are two variants of KG that are less greedy, and that work a bit better in high-noise-variance settings.

- KG(*) policies: Considers not just a single measurement, but multiple measurements of a single alternative to maximize the average value per measurement, in a static setting. [Frazier and Powell, 2010].
- PDE policies: Considers an adaptive stopping rule for measuring a single alternative, so as to maximize expected value.
  [Chick and Frazier, 2012]

Outline

1. The Ranking and Selection Problem
   - Average-case Performance
   - Dynamic Programming
2. Approximate Methods
   - Basis Functions
   - Value of Information / Knowledge-gradient (KG) Methods
3. Integer-ordered Optimization via Simulation
   - Correlated Normal Prior / GP Regression / Kriging
   - Knowledge-Gradient Policy for Correlated Beliefs

Integer-ordered Optimization via Simulation

- Consider optimization via simulation problems where the alternatives correspond to points in a grid.
  - Examples: the (s, S) inventory problem; staffing problems; buffer allocation problems.
- In many such problems, the sampling means at two nearby points are often similar.
  - Example: in an (s, S) inventory problem, if we find out that $(s, S) = (5, 18)$ works extremely well, we will also think that $(5, 17)$ would also work well.
- We choose P to put more weight on problem instances with this property.
- This allows us to do optimization via simulation even when the number of measurements is smaller than the number of alternatives.
- We will then apply a KG policy.

Outline

We use a correlated prior to put more weight on problem instances where nearby points have similar means

- Let $\theta = [\theta_1, \ldots, \theta_k]$ as before.
- Although alternatives are numbered from 1 to k, they might correspond to points on a multidimensinal grid. So alternative 1 might correspond to the point $(0,0)$; alternative 2 to the point $(0,1)$, etc.
- Let P be a multivariate normal prior distribution on θ,

$$\theta \sim \text{Normal}(\mu_0, \Sigma_0).$$

- We choose Σ_0 so that, if two points x and x' are near to each other in the grid, then $\Sigma_0(x, x')$ is high; if they are far from each other, $\Sigma_0(x, x')$ is close to 0.
- One way to accomplish this comes from the literature on Gaussian process priors, a Bayesian version of kriging.

We are doing Gaussian process regression

- What we are doing is called Gaussian process regression, see e.g. [Rasmussen and Williams, 2006].
- It is similar to stochastic kriging [Ankenman et al., 2008, Ankenman et al., 2010].
- The analysis presented here does not deal with unknown and heterogeneous sampling variance — this is possible, see e.g., [Goldberg et al., 1998], but requires more work.

We choose the prior covariance matrix using a standard technique from kriging / Gaussian process regression

- Let $|x_i - x_i'|$ be the distance between alternatives x and x' on the grid in direction i, where the grid has d dimensions.
- Then, choose $\Sigma_0(x, x') = \beta \exp(-\sum_{i=1}^{d} \alpha_i (x_i - x_i')^2)$
- To choose the parameters $\beta, \alpha_1, \ldots, \alpha_d$, we run an initial set of samples from using latin hypercube or uniform sampling, and fit using maximum likelihood. We can update these estimates periodically using data obtained during subsequent sampling.
- Other parameterized choices for Σ_0 are possible — see, e.g., [Rasmussen and Williams, 2006].

Correlated Bayesian Ranking & Selection

- With a multivariate normal prior, the posterior is also multivariate normal,

$$\theta | x_1, \ldots, x_n, y_1, \ldots, y_n \sim \mathcal{N}(\mu_n, \Sigma_n),$$

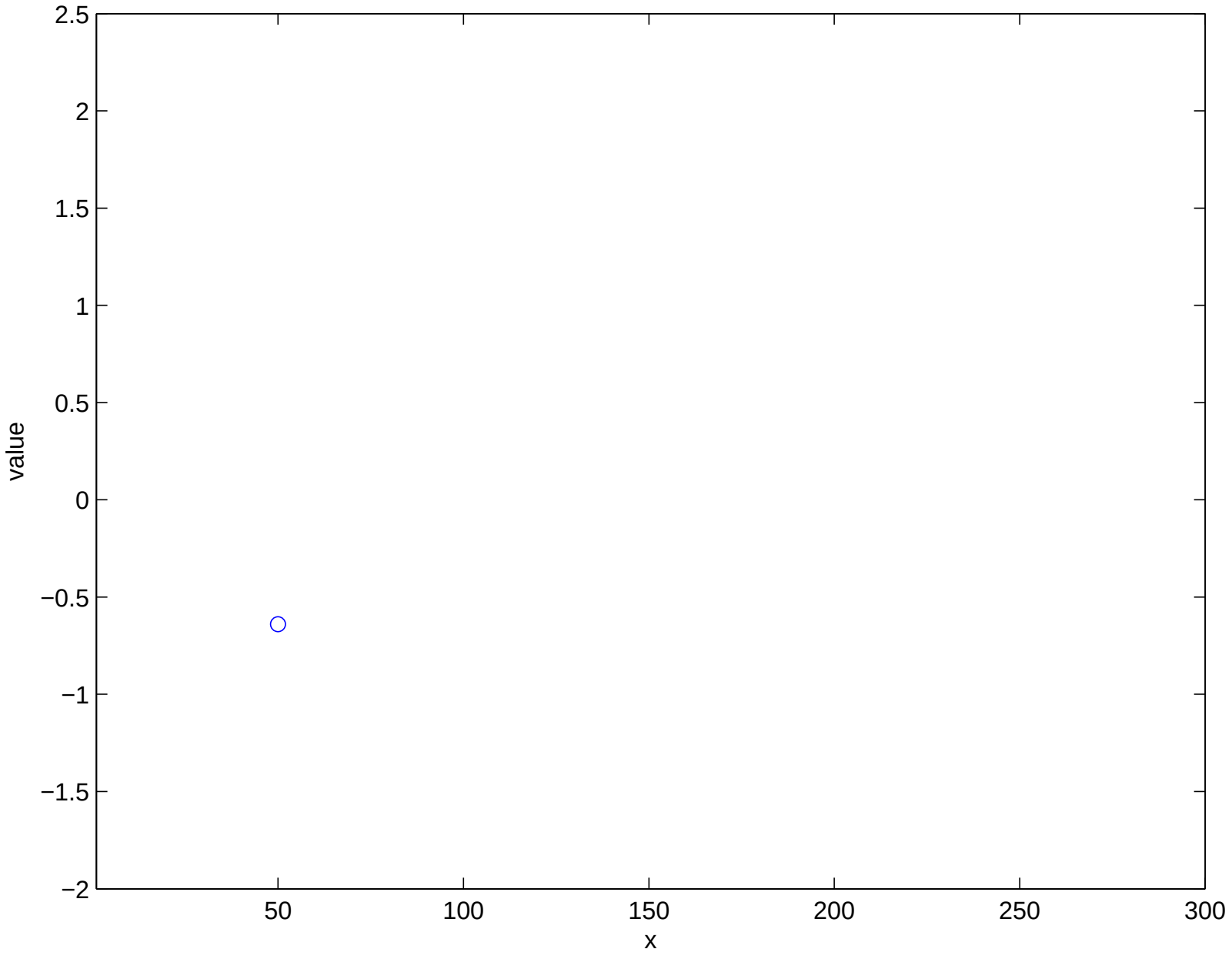
- μ^n and Σ^n may be computed recursively as

$$\mu^{n+1} = \mu^n + \frac{y_{n+1} - \mu_x^n}{\lambda_x + \Sigma_{xx}^n} \Sigma^n e_x,$$

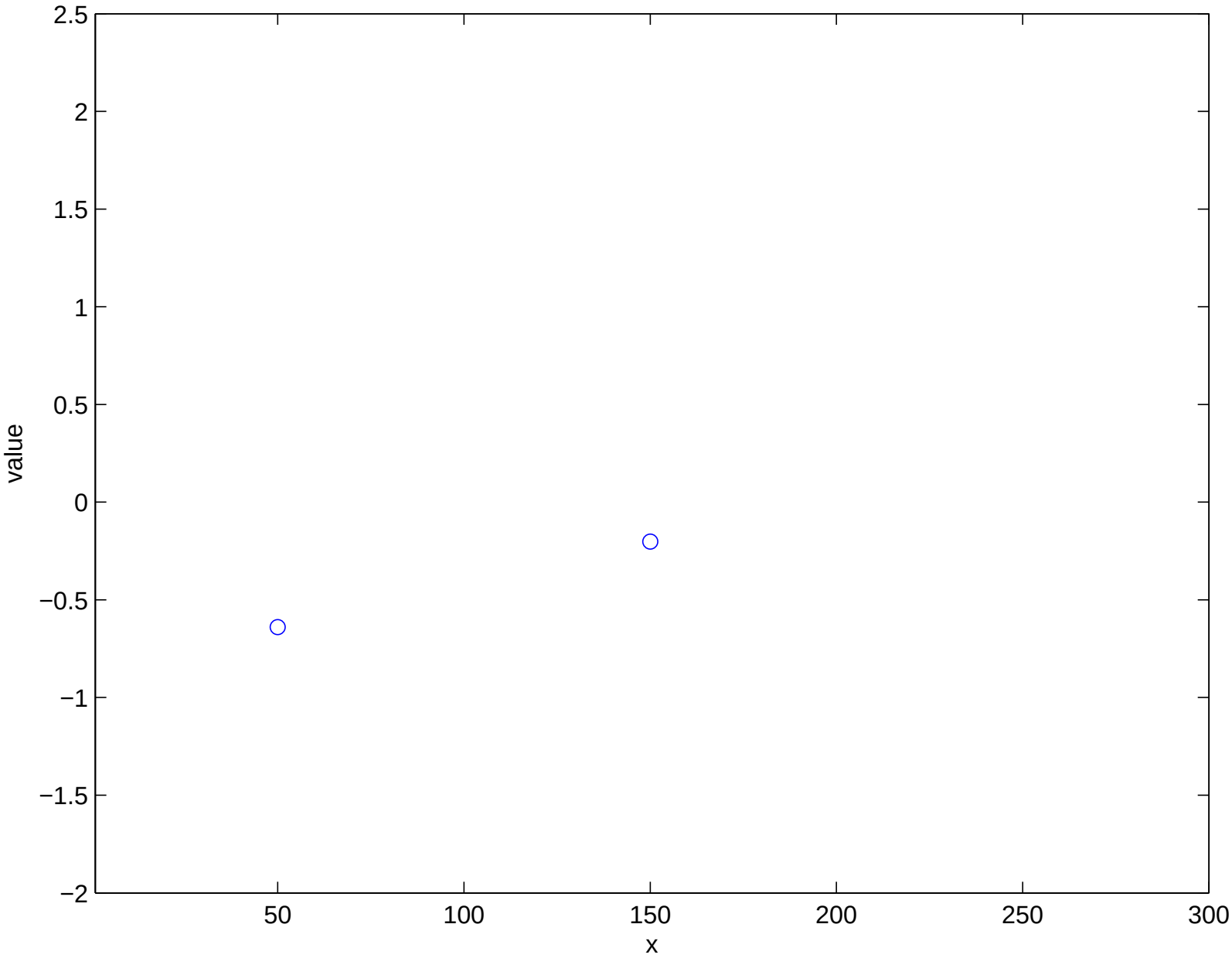
$$\Sigma^{n+1} = \Sigma^n - \frac{\Sigma^n e_x e_x' \Sigma^n}{\lambda_x + \Sigma_{xx}^n}.$$

where $x = x_{n+1}$, and e_x is the vector of all 0s, with a 1 at index x.

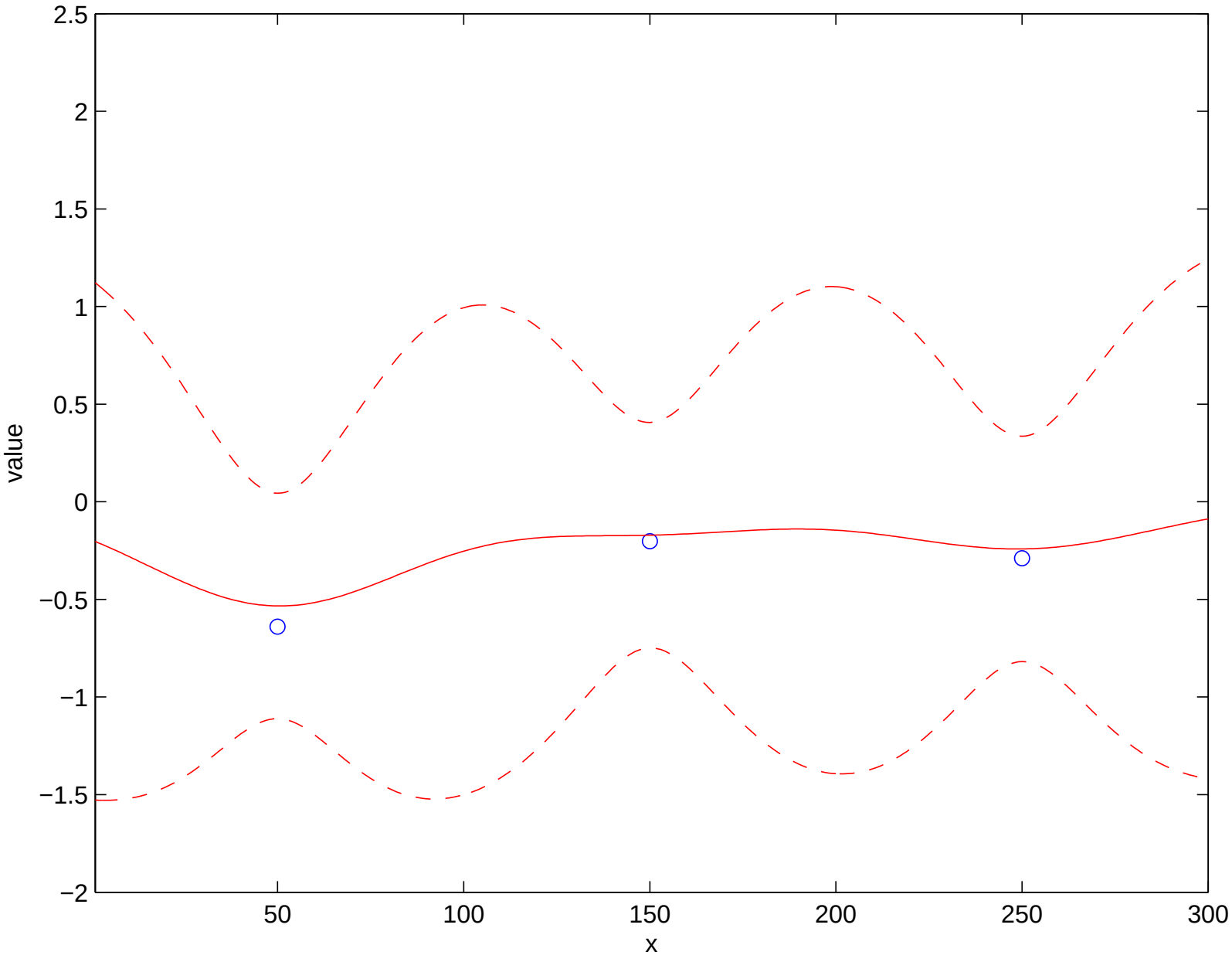
Illustrative 1D Example with Noise



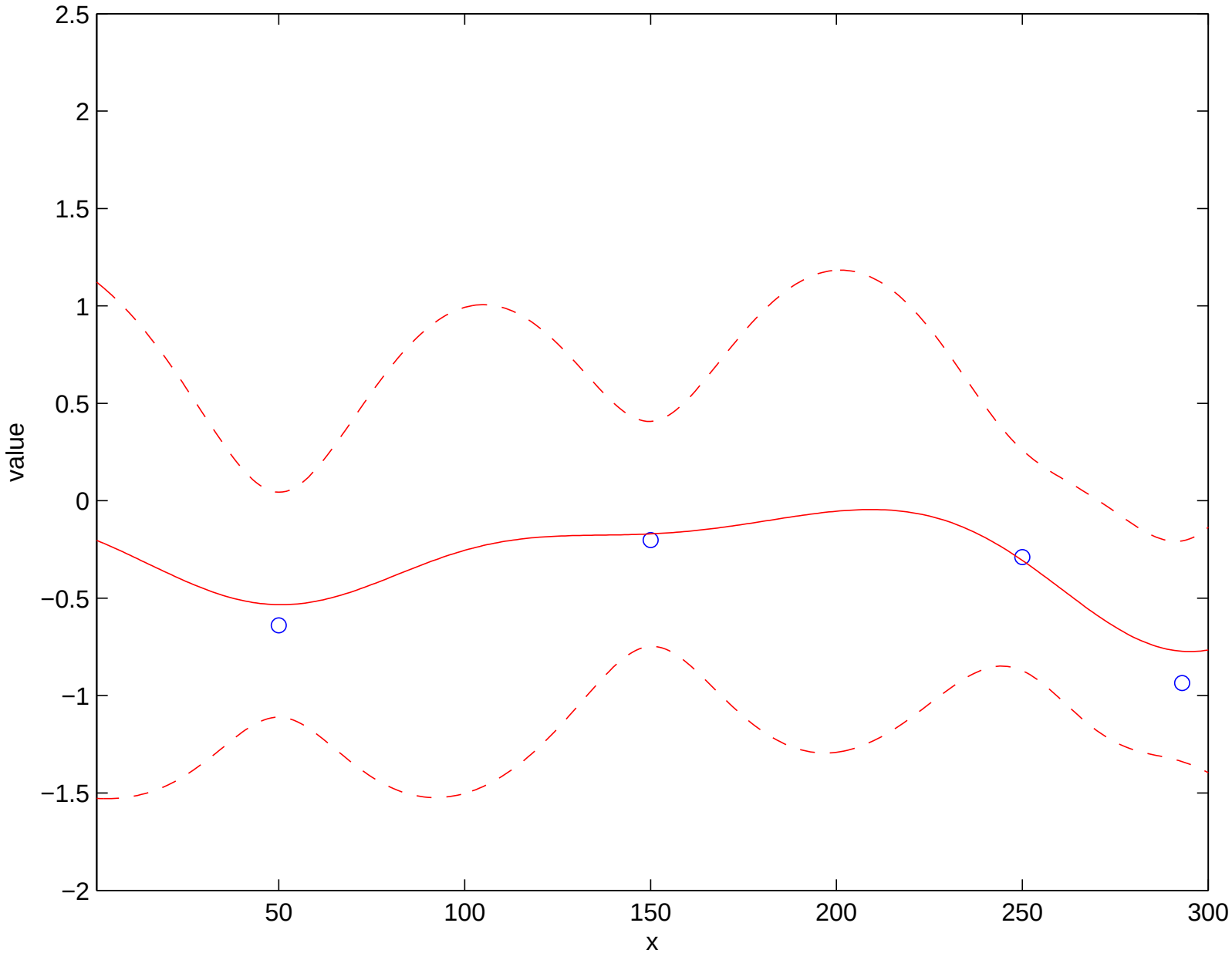
Illustrative 1D Example with Noise



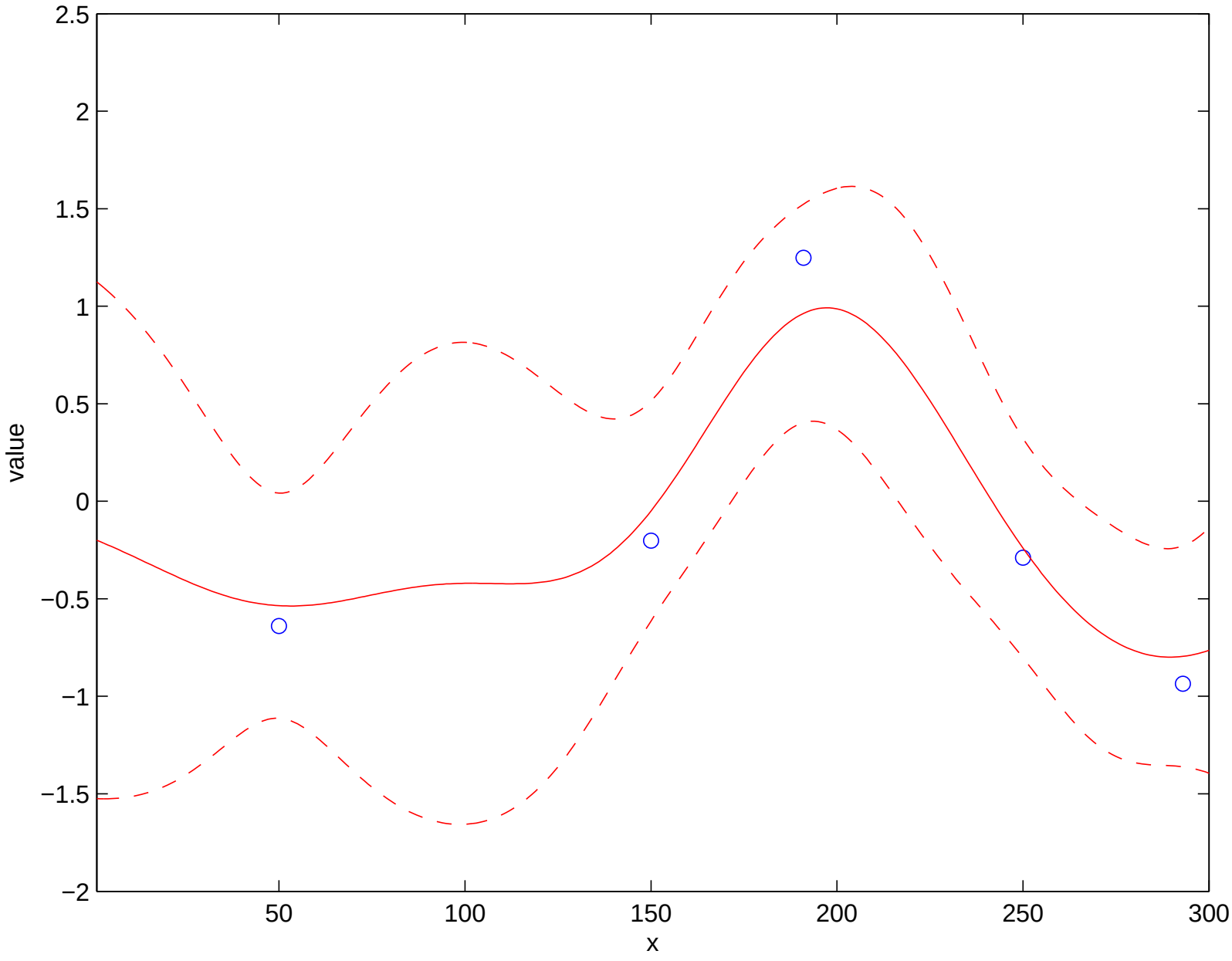
Illustrative 1D Example with Noise



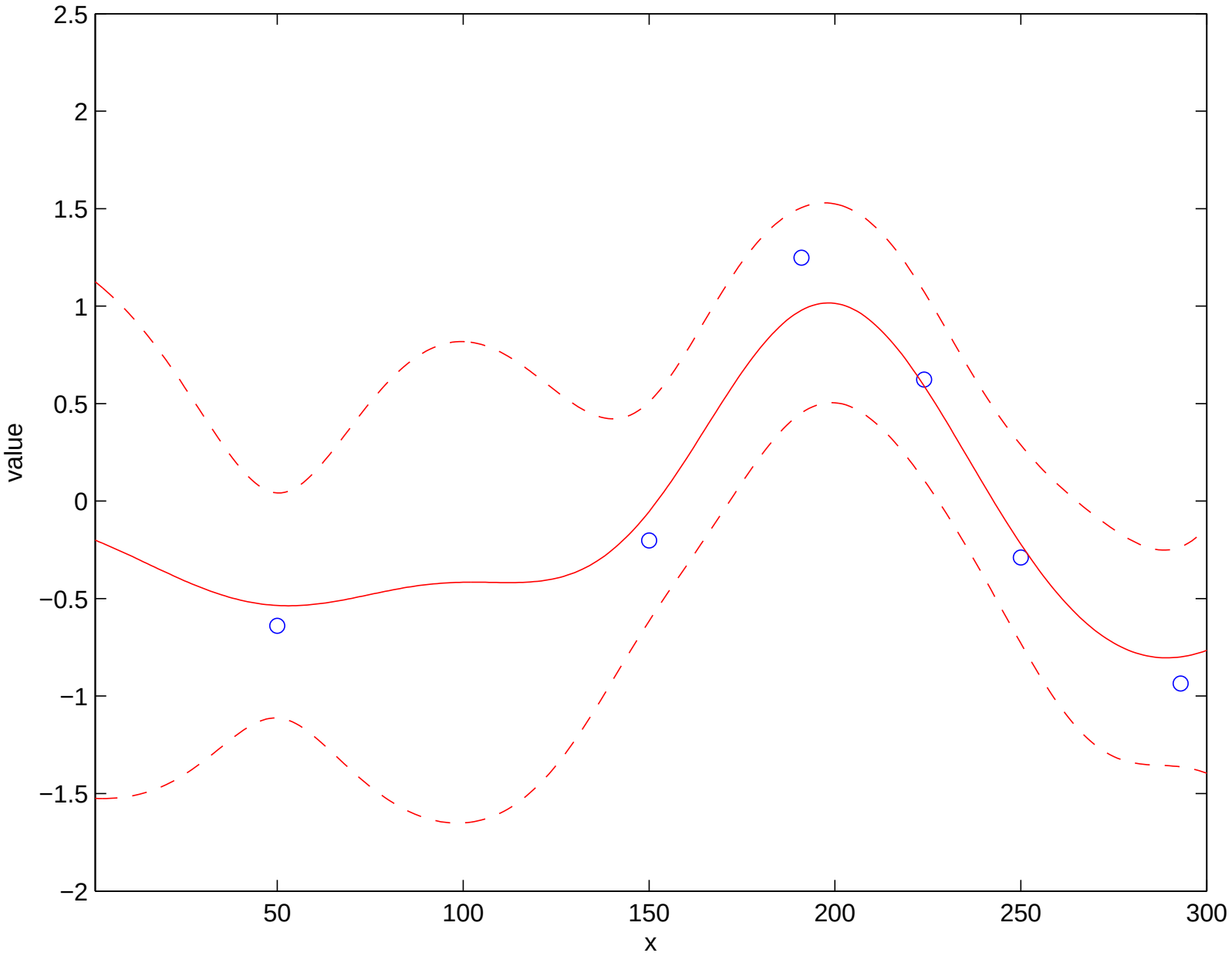
Illustrative 1D Example with Noise



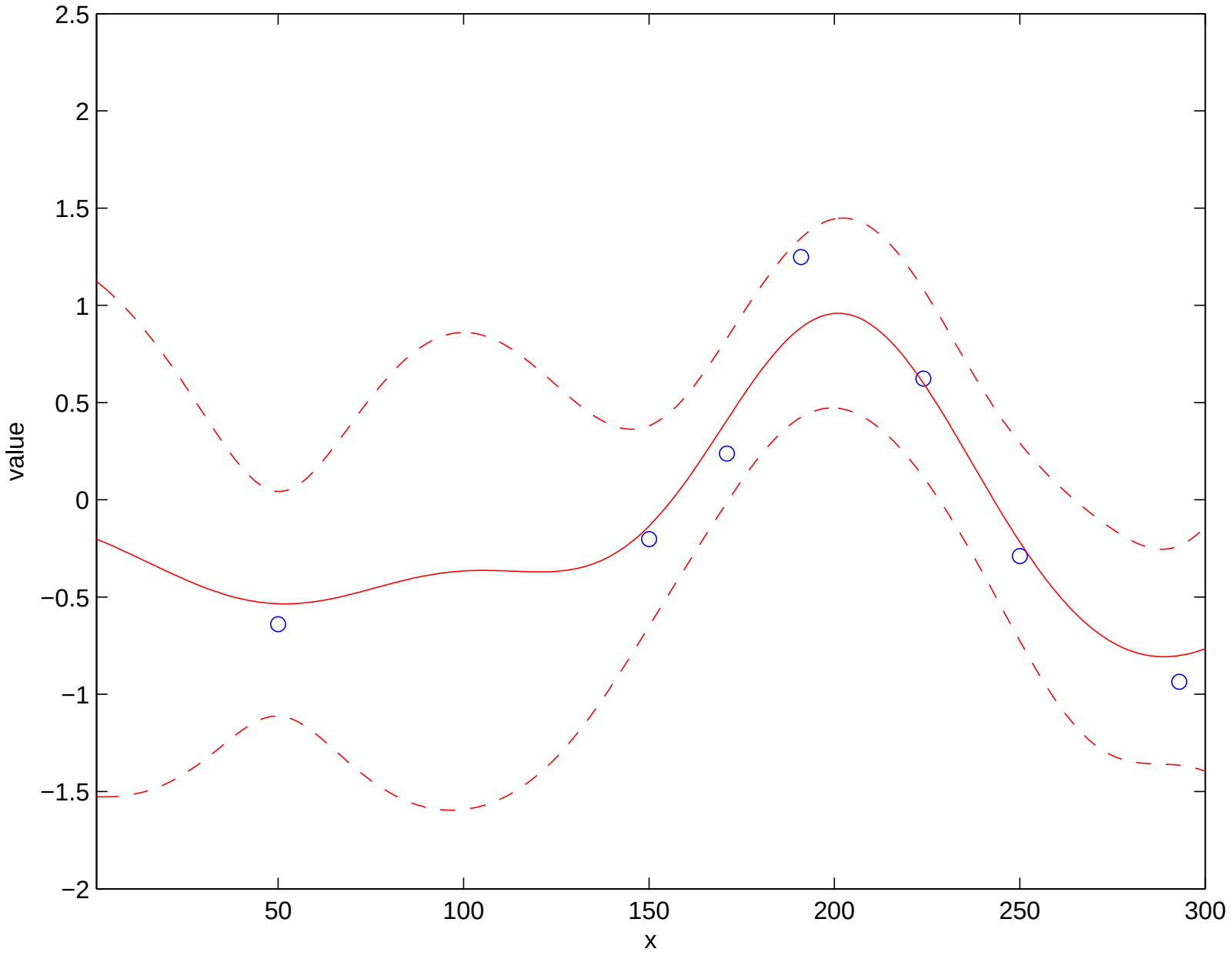
Illustrative 1D Example with Noise



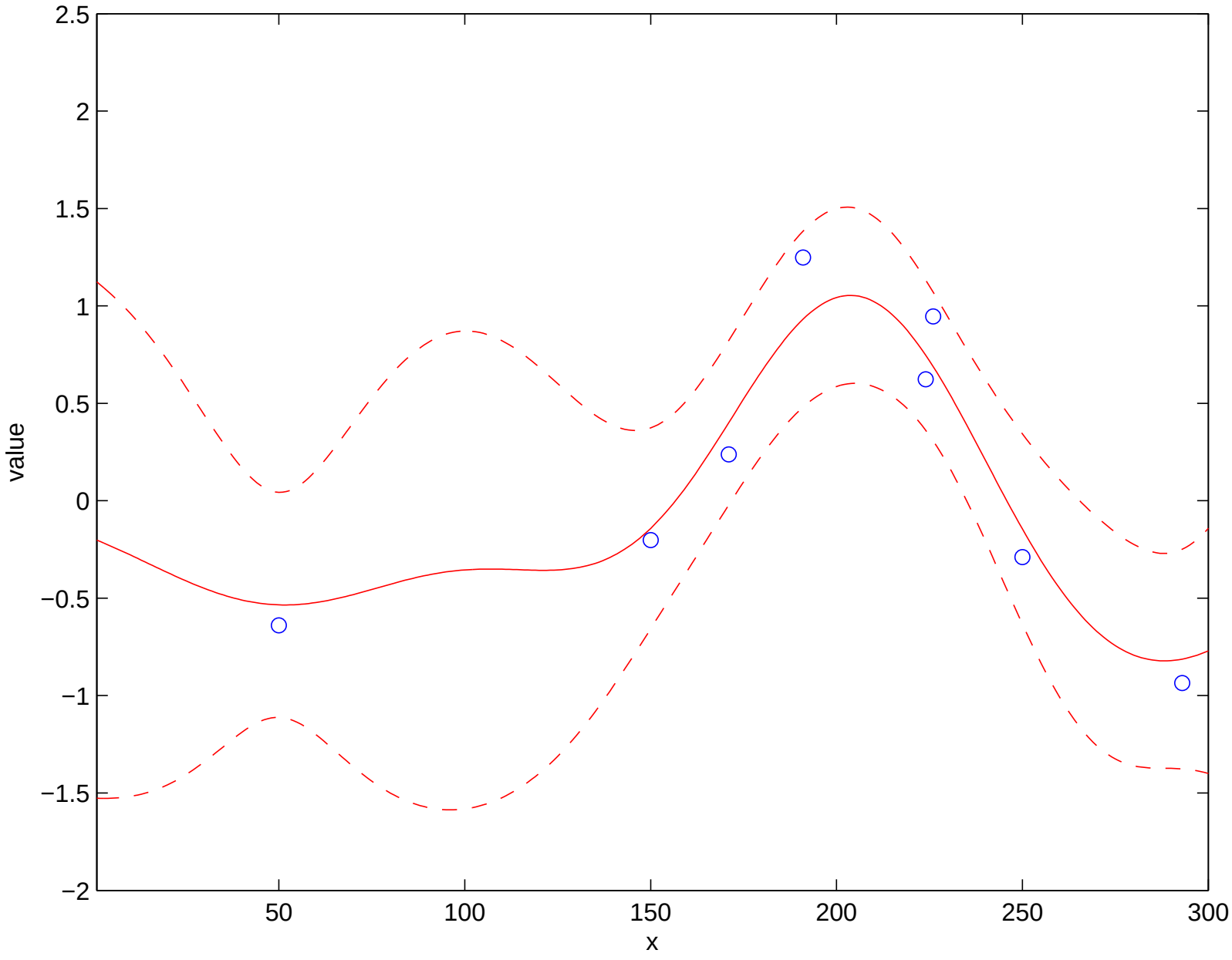
Illustrative 1D Example with Noise



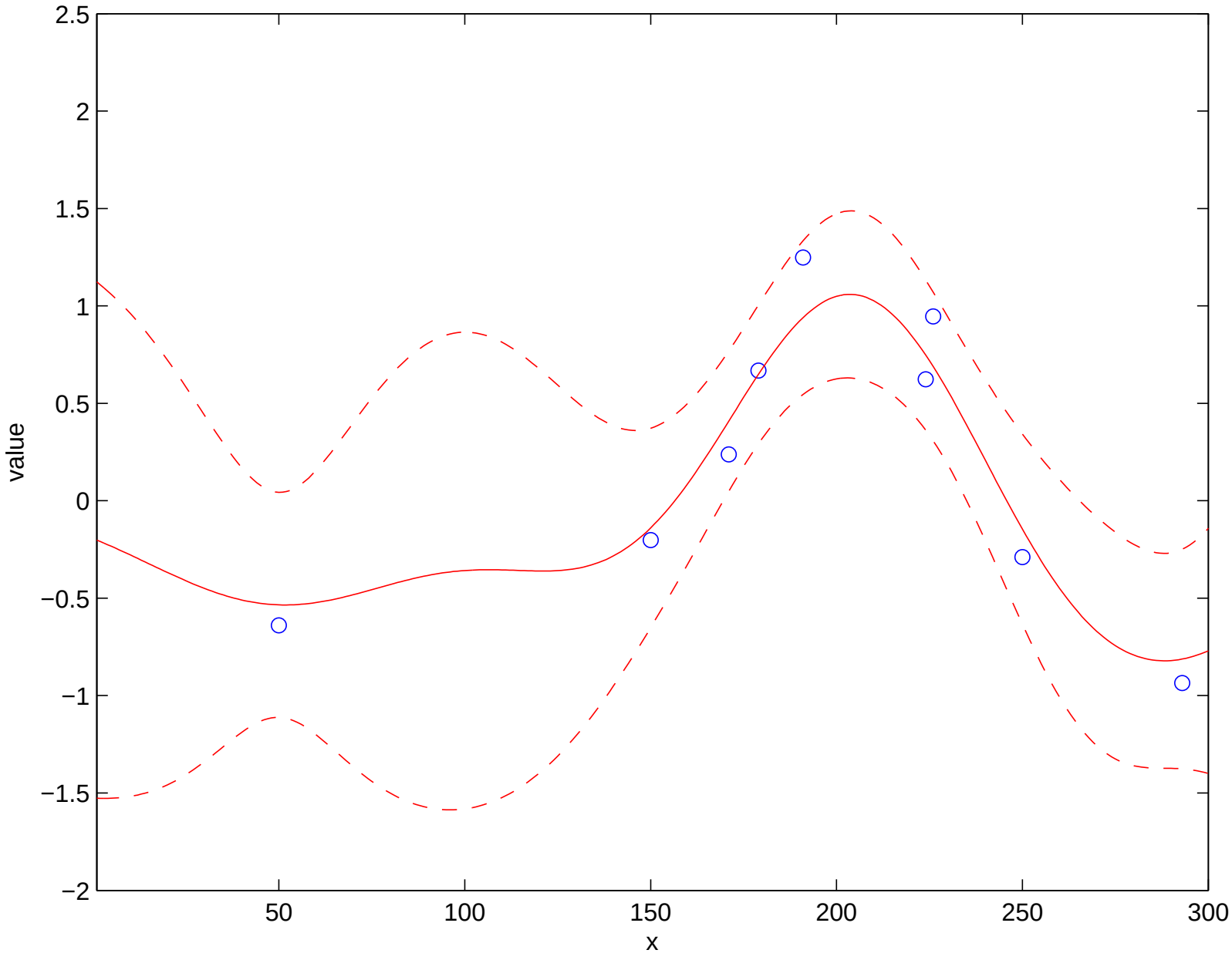
Illustrative 1D Example with Noise



Illustrative 1D Example with Noise



Illustrative 1D Example with Noise



Outline

Knowledge-Gradient Policy

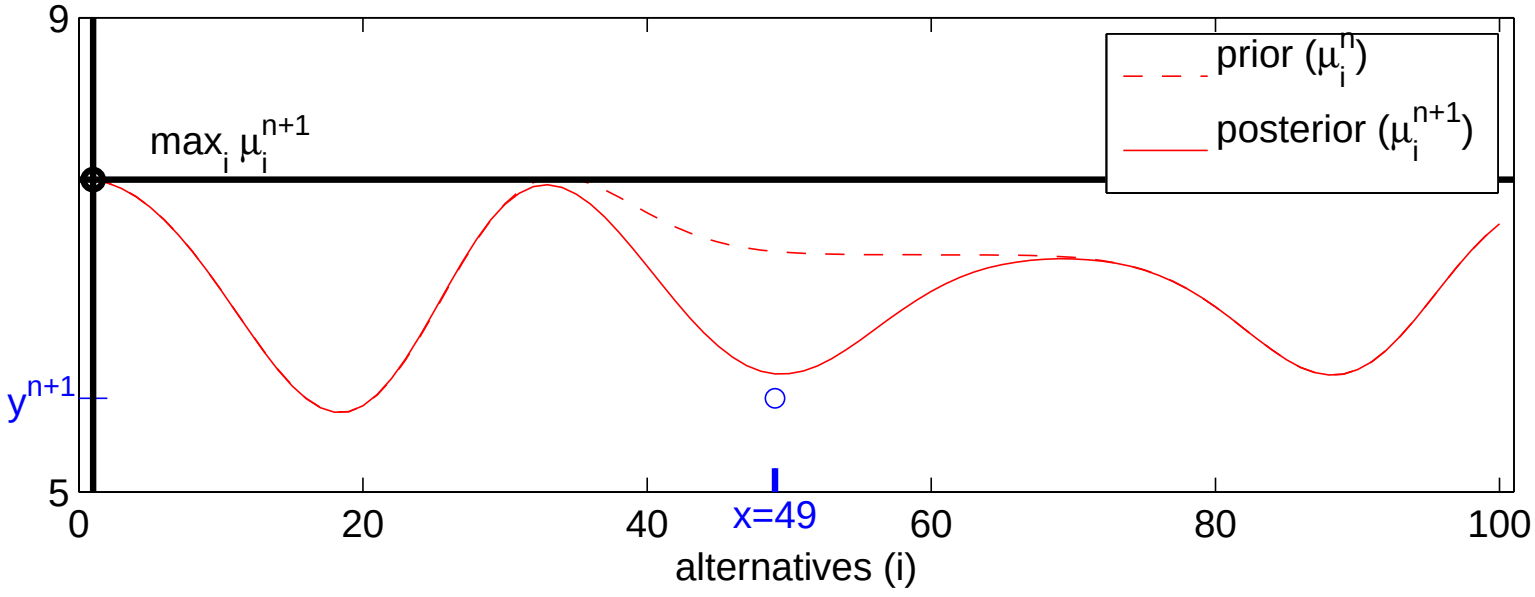
As before, the **knowledge-gradient policy** is the policy that chooses x_{n+1} to maximize this KG factor / value of information:

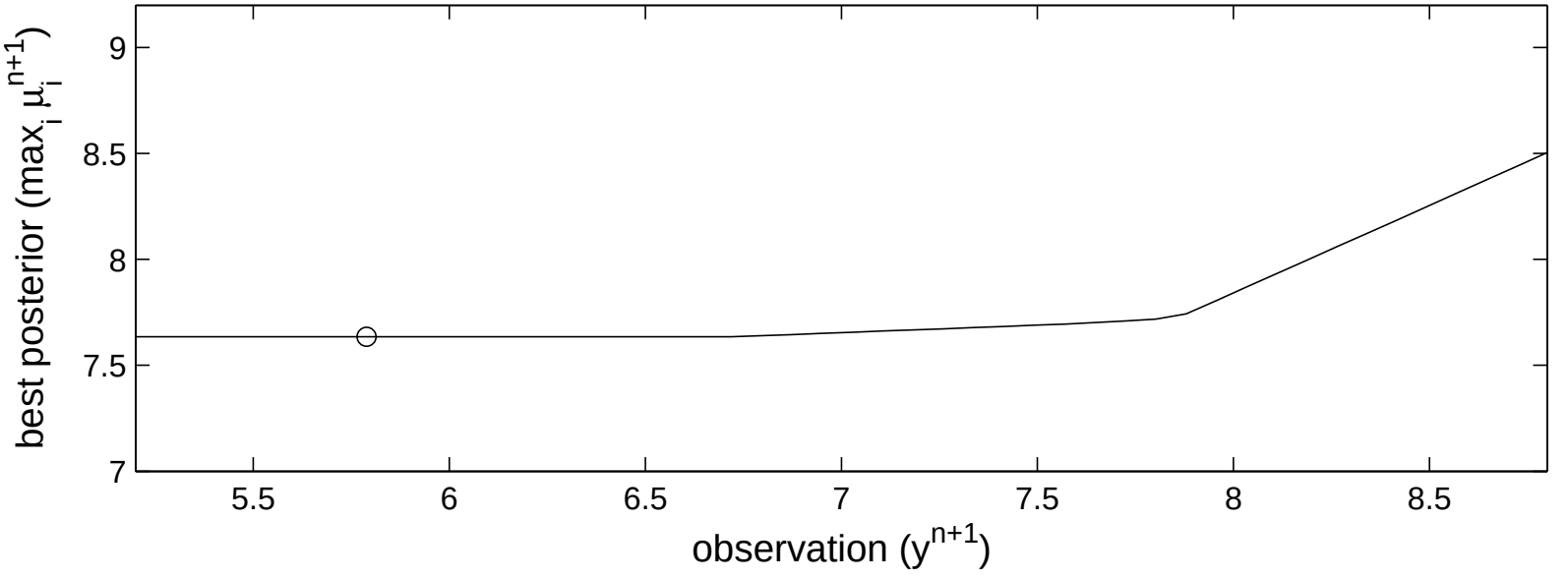
$$\text{value of measuring } x = \mathbb{E}_n \left[\max_i \mu_i^{n+1} \mid x_{n+1} = x \right] - \max_i \mu_i^n.$$

The only thing that changes is P.

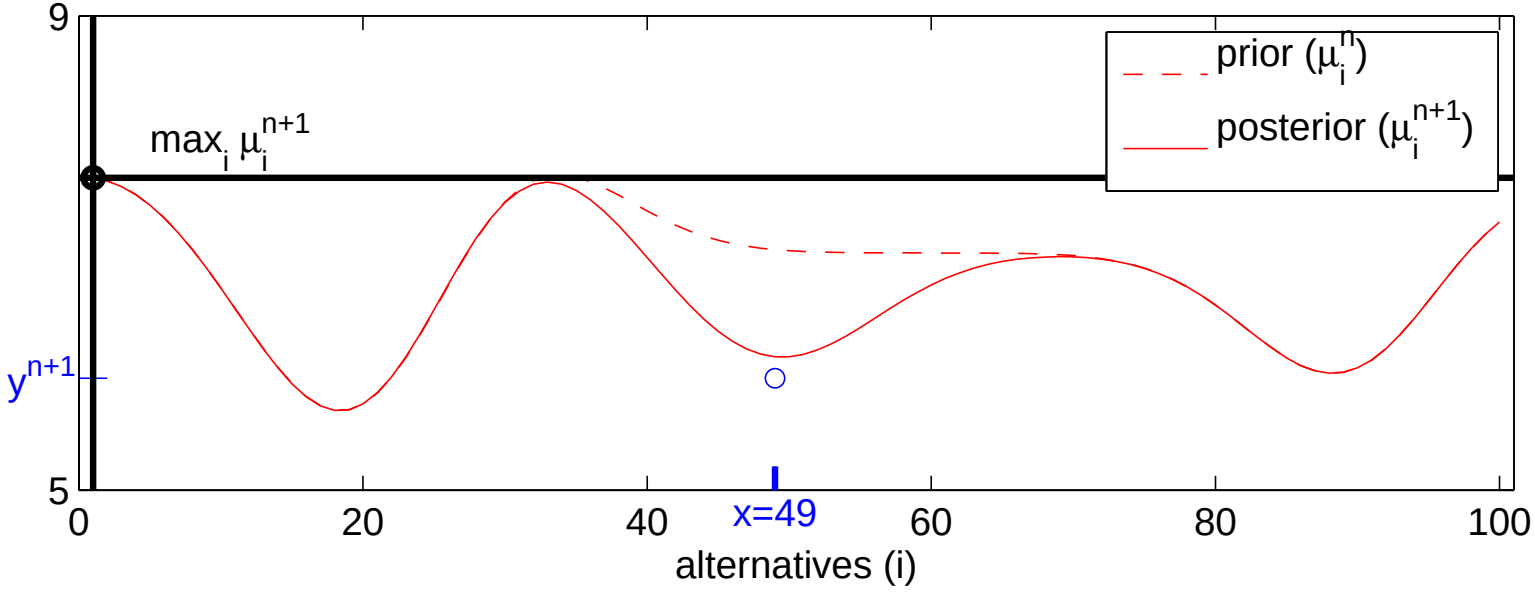
- μ_i^n is the expected value of alt. i given what we know at time n.
- $\max_i \mu_i^n$ is the best we can do given what we know at n.
- $\max_i \mu_i^{n+1}$ is the best we will be able to do given what we know at n and what we learn from our measurement x_{n+1}.
- When Σ^0 is diagonal, we recover the (R1,. . . ,R1) policy of [Gupta & Miescke 1996].

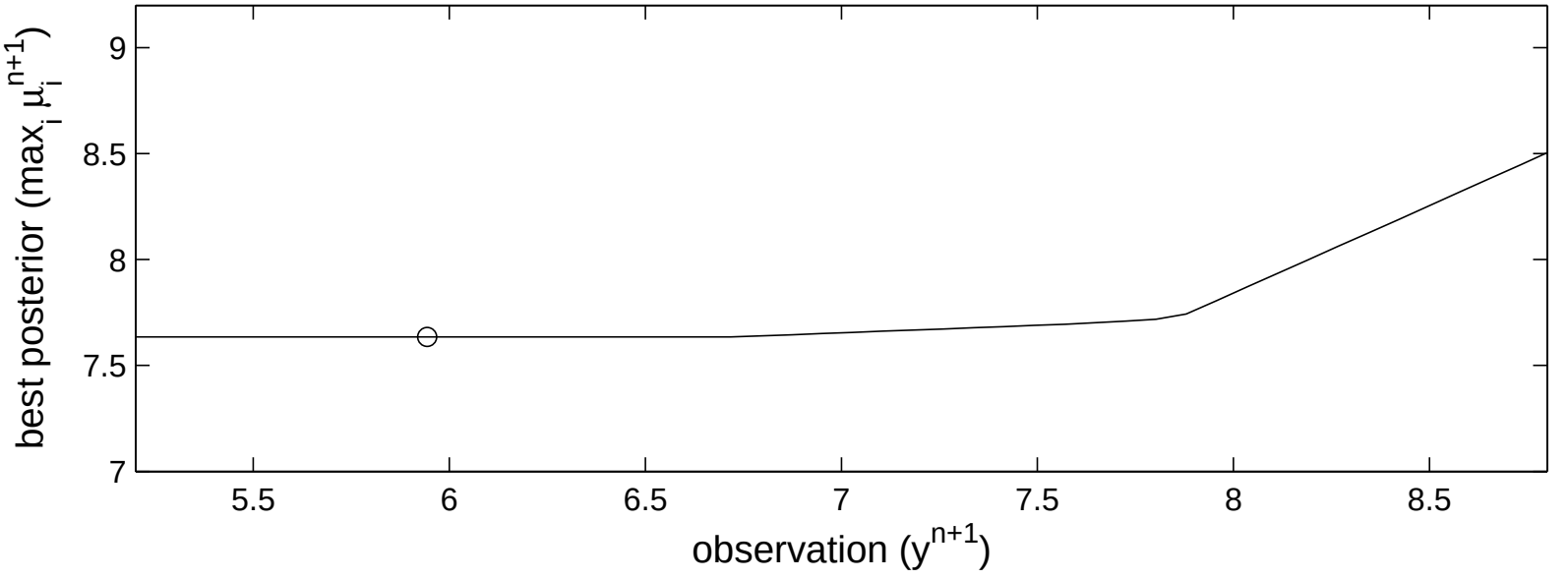
Computing the Knowledge-Gradient



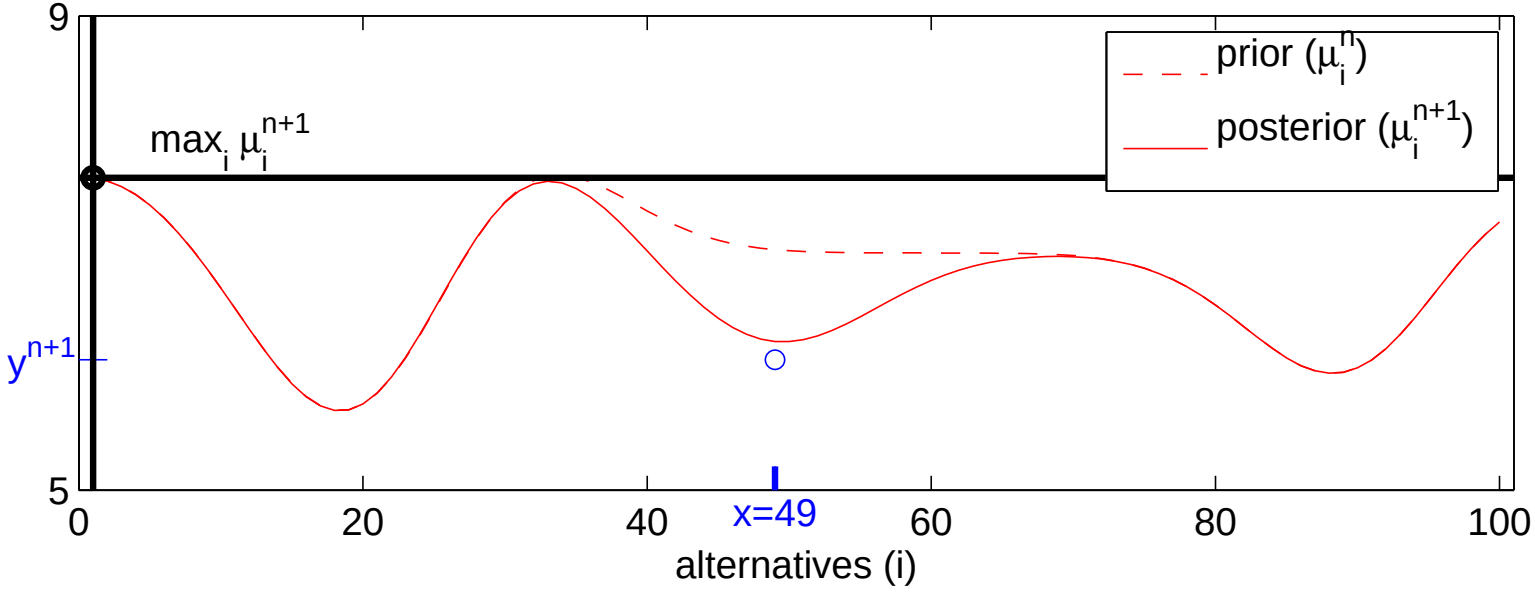


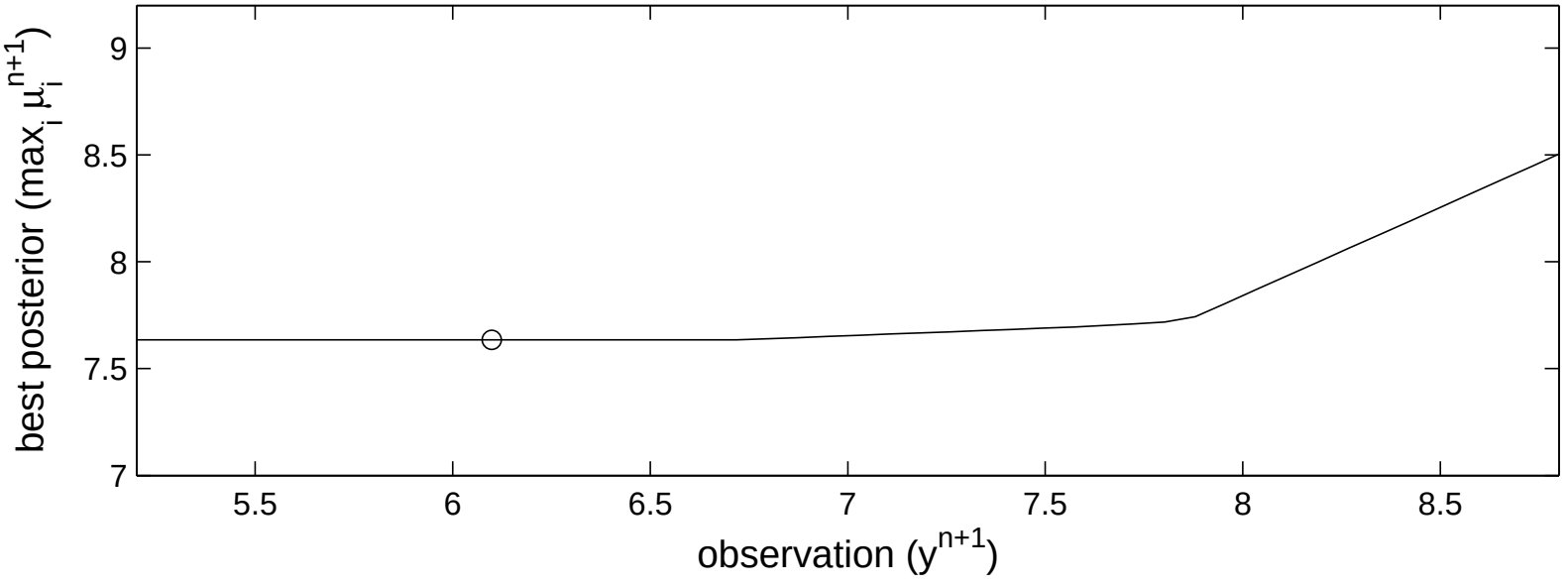
Computing the Knowledge-Gradient



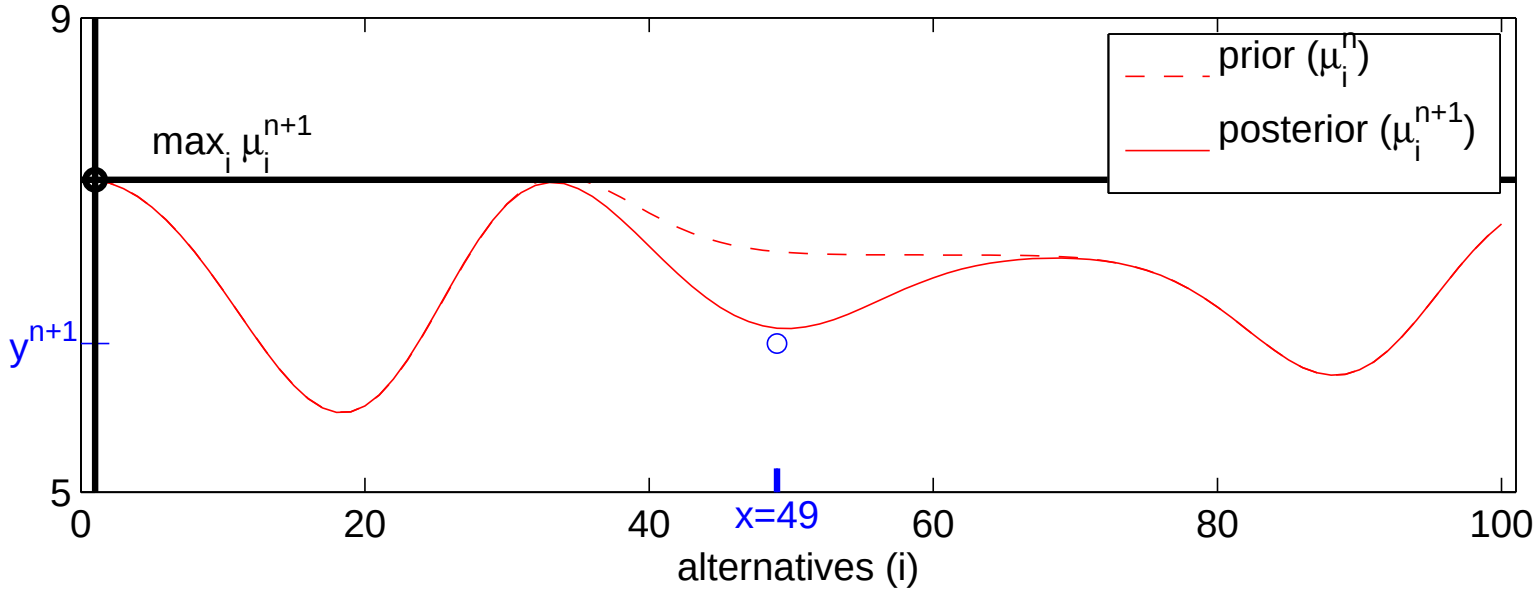


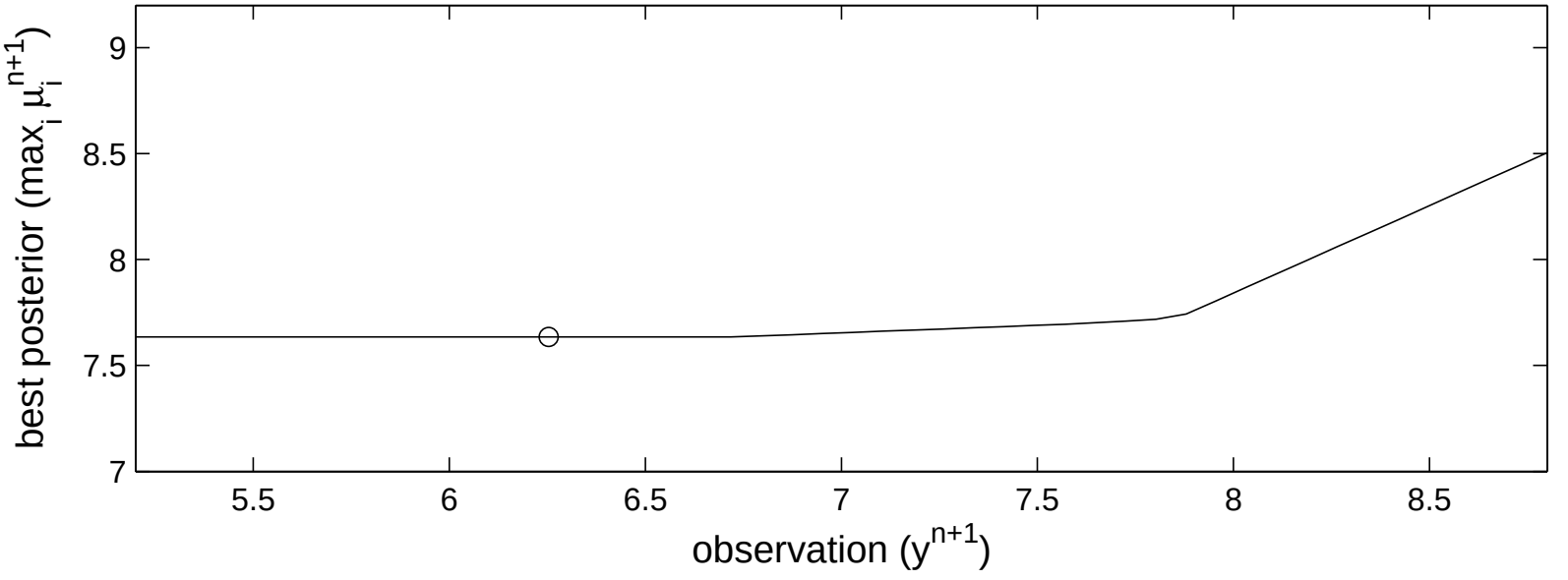
Computing the Knowledge-Gradient



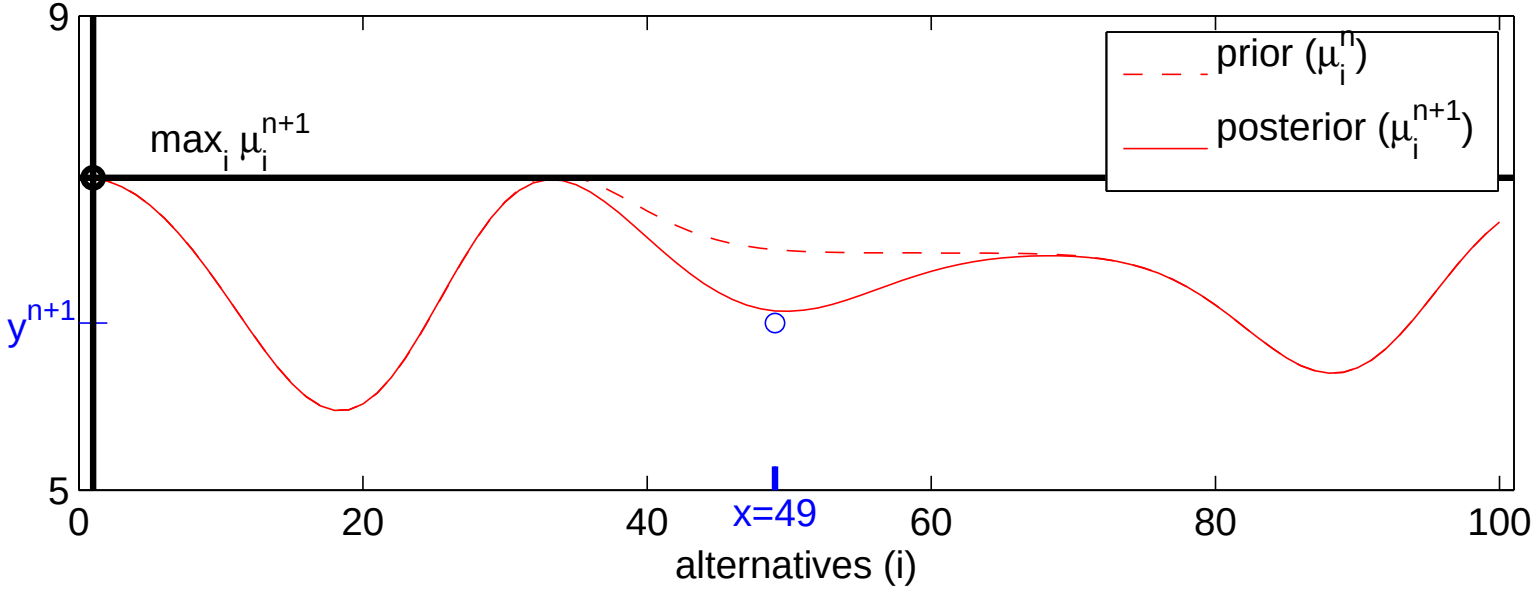


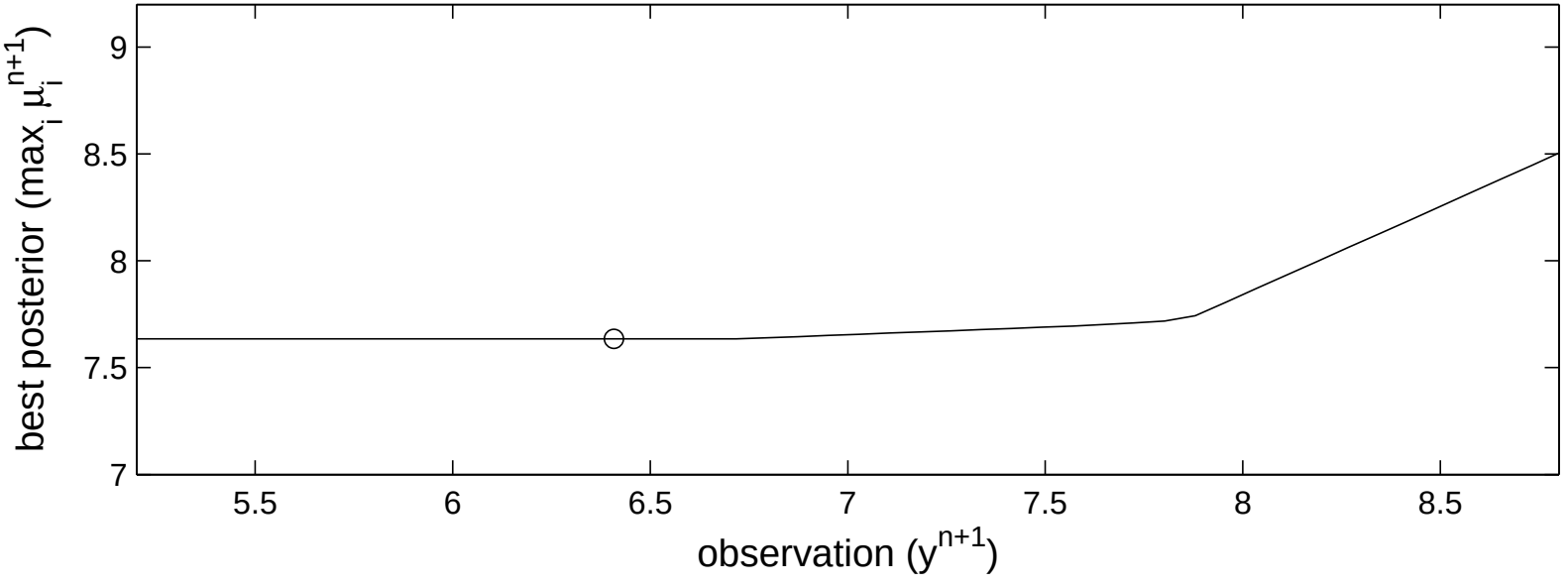
Computing the Knowledge-Gradient



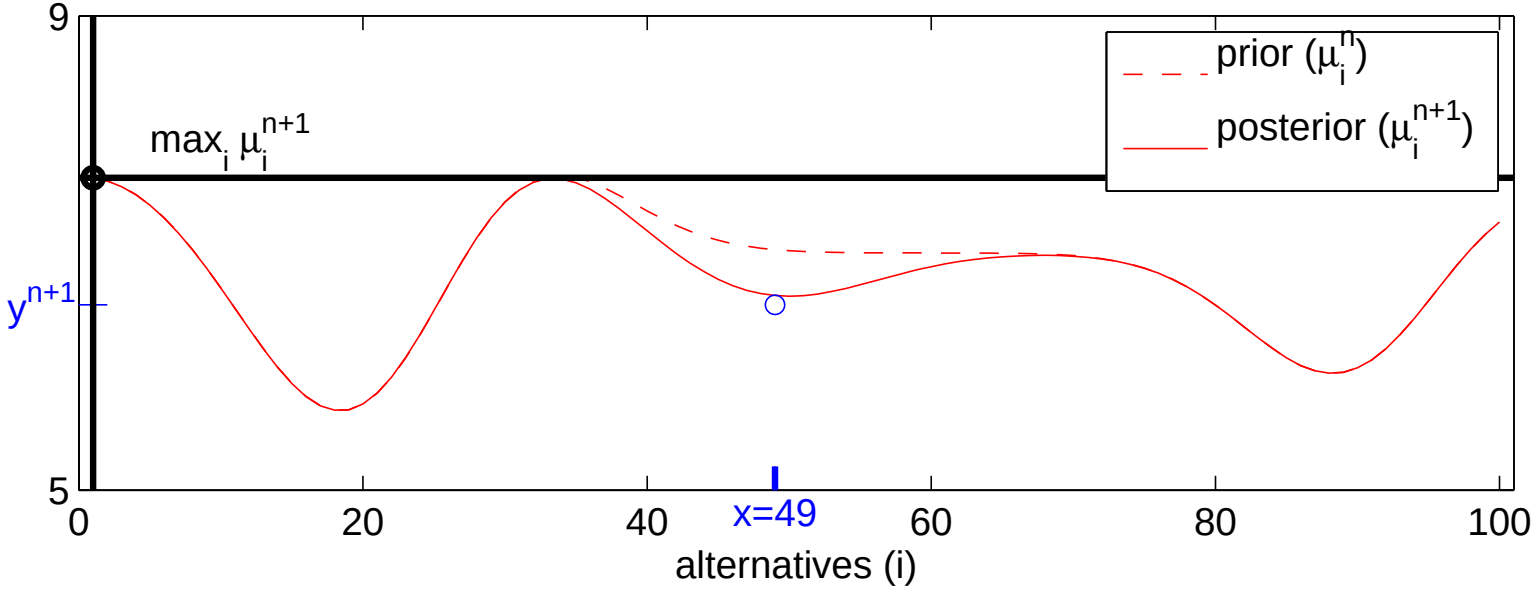


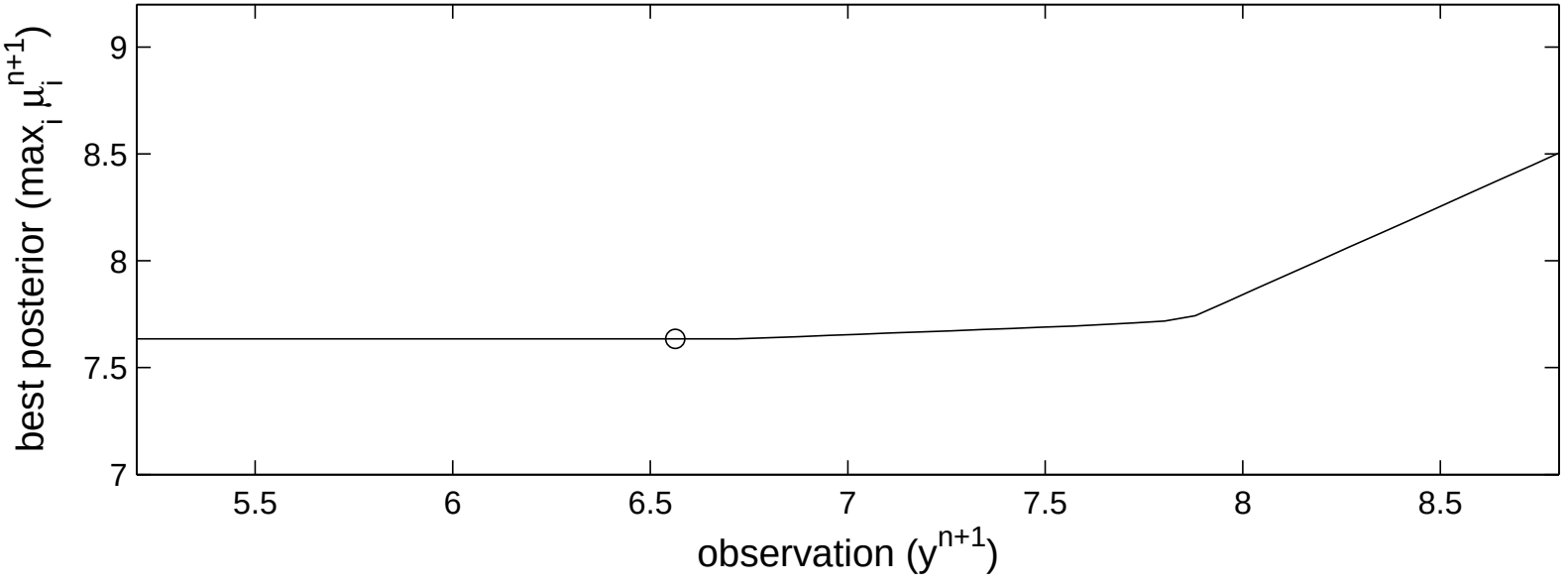
Computing the Knowledge-Gradient



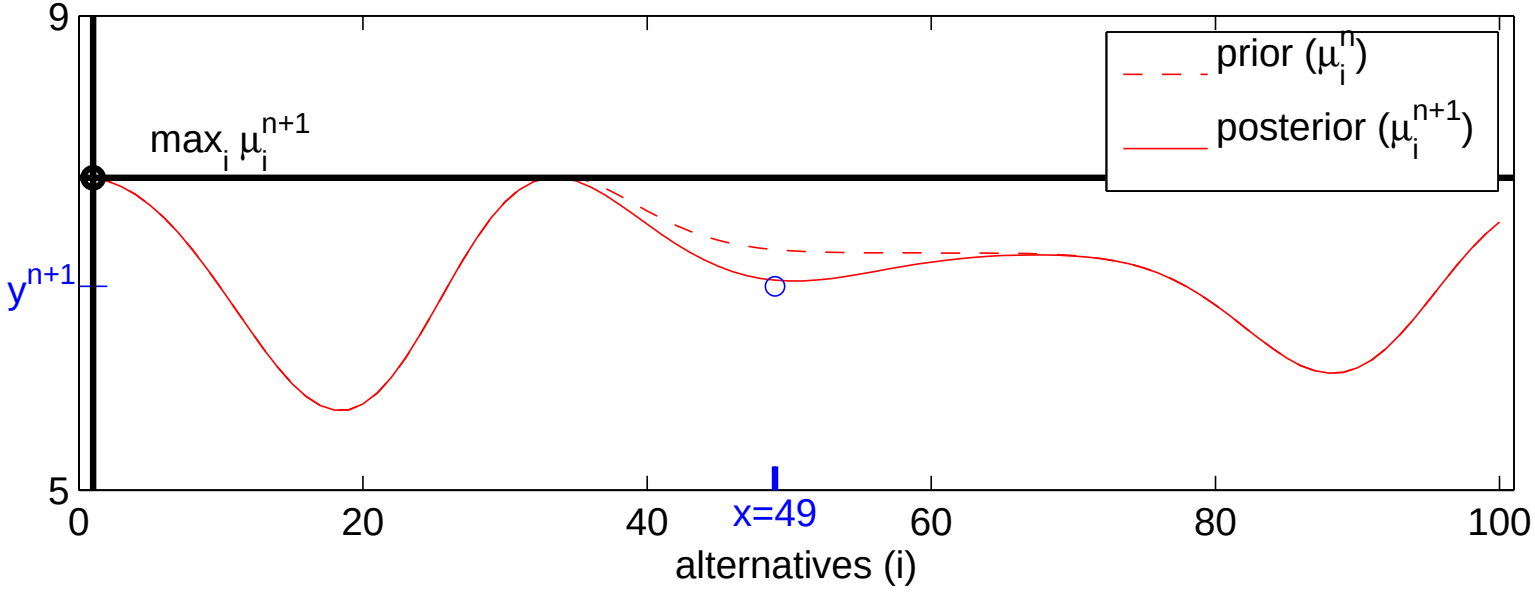


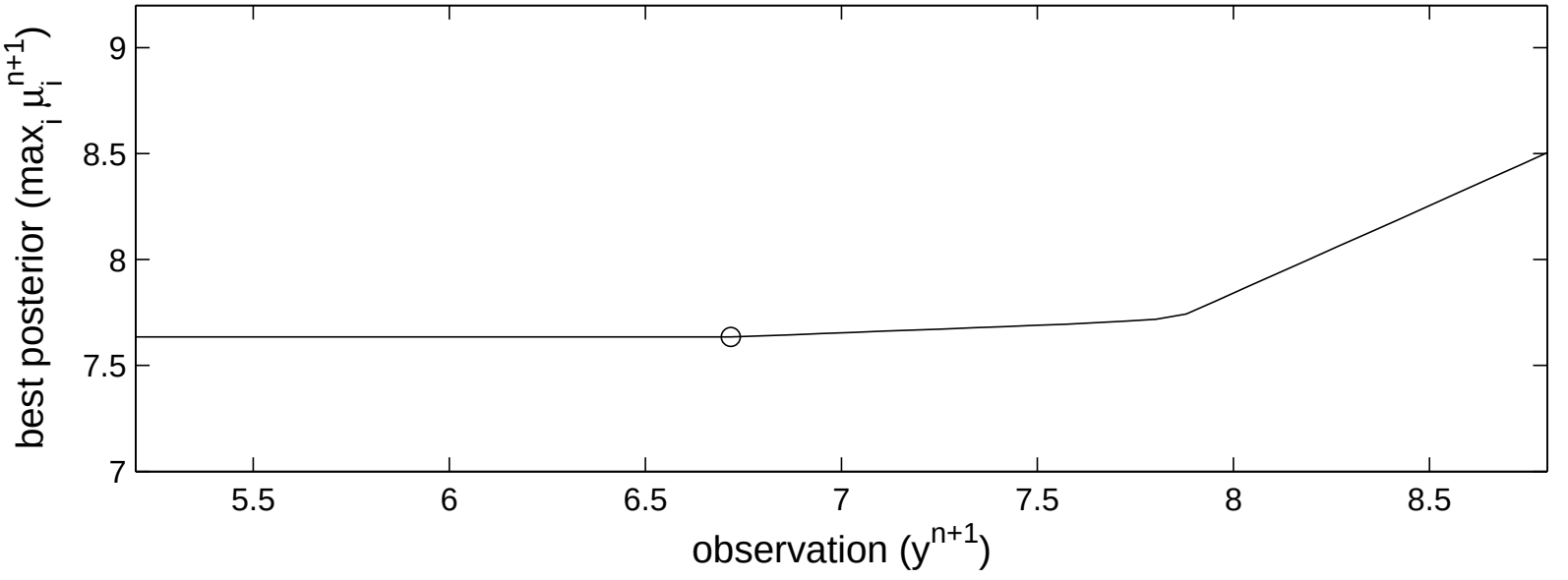
Computing the Knowledge-Gradient



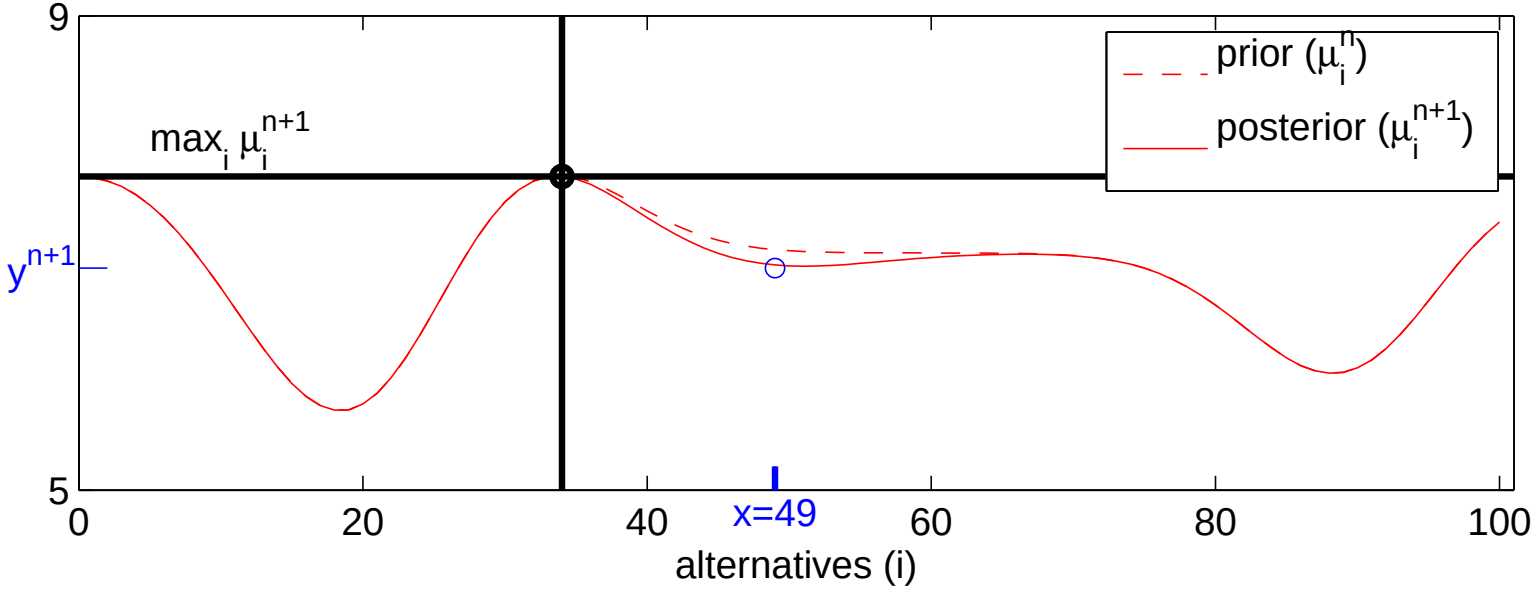


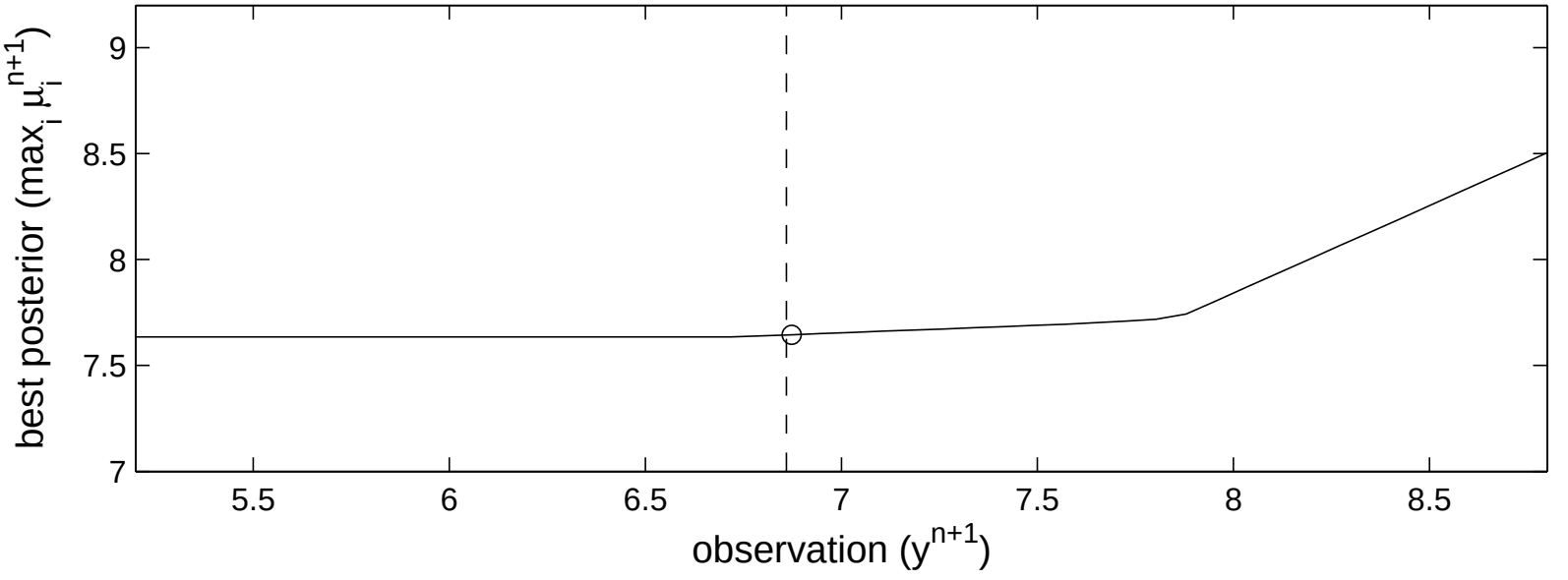
Computing the Knowledge-Gradient



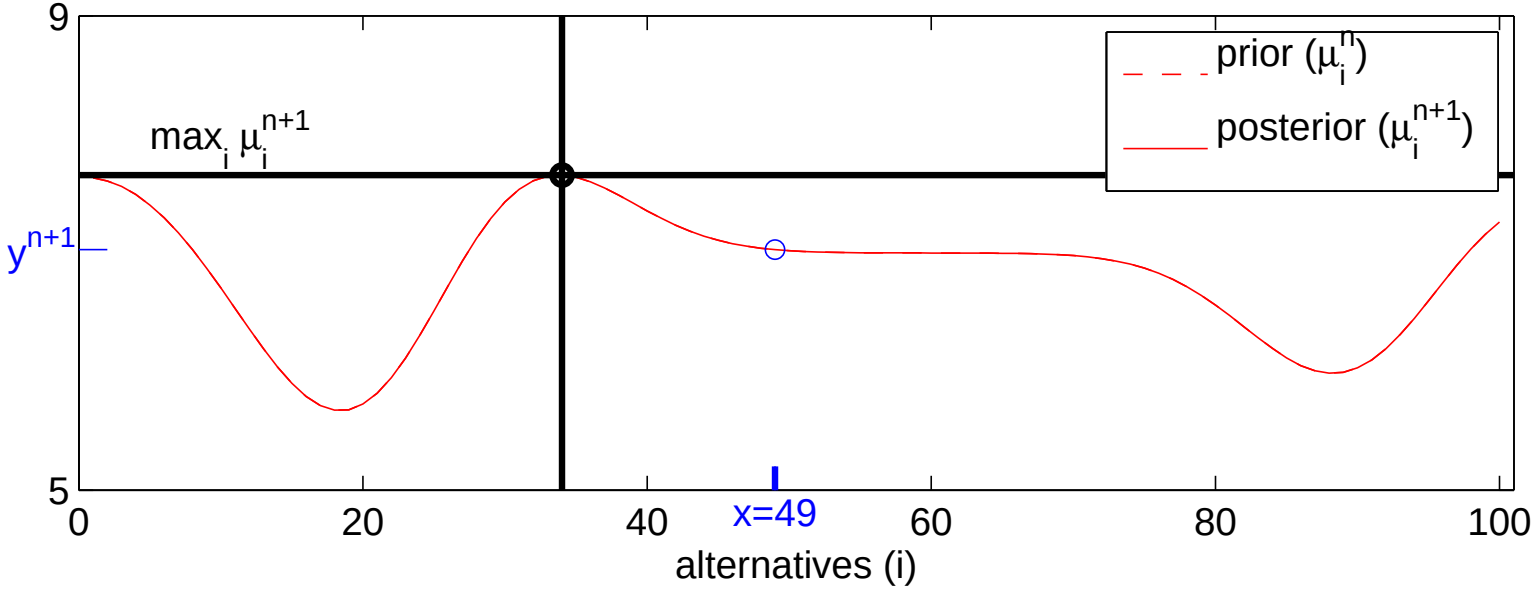


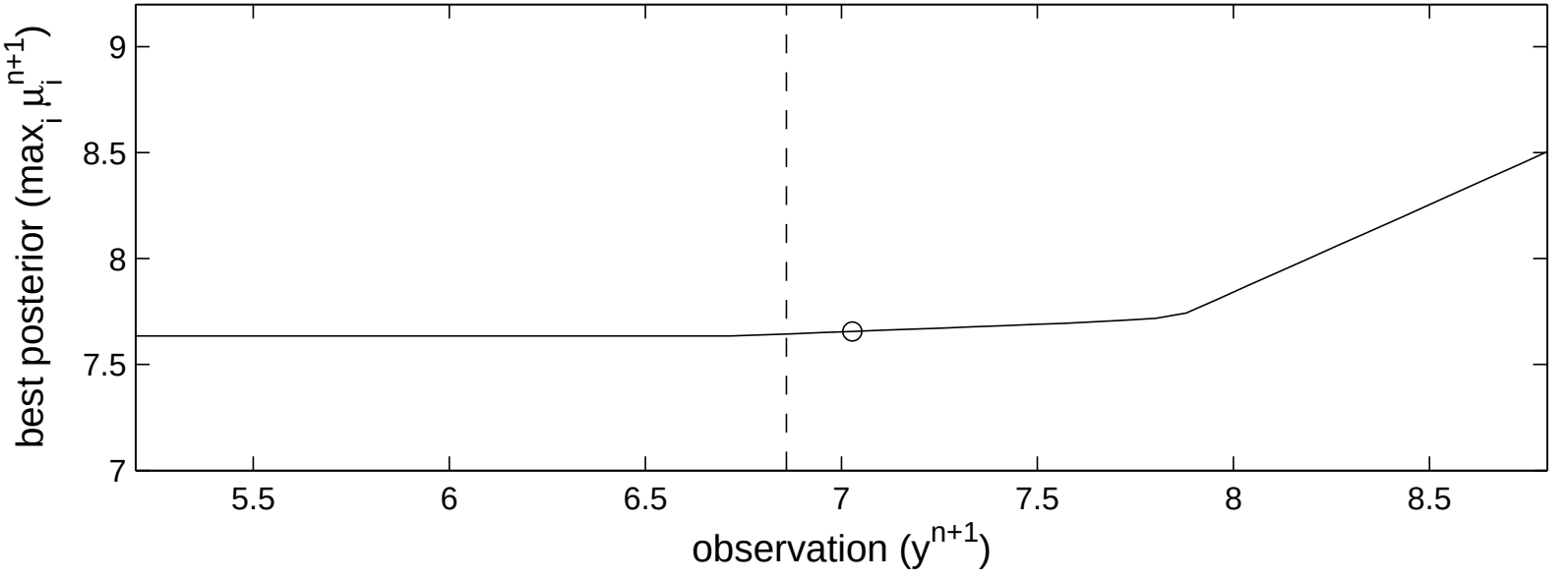
Computing the Knowledge-Gradient



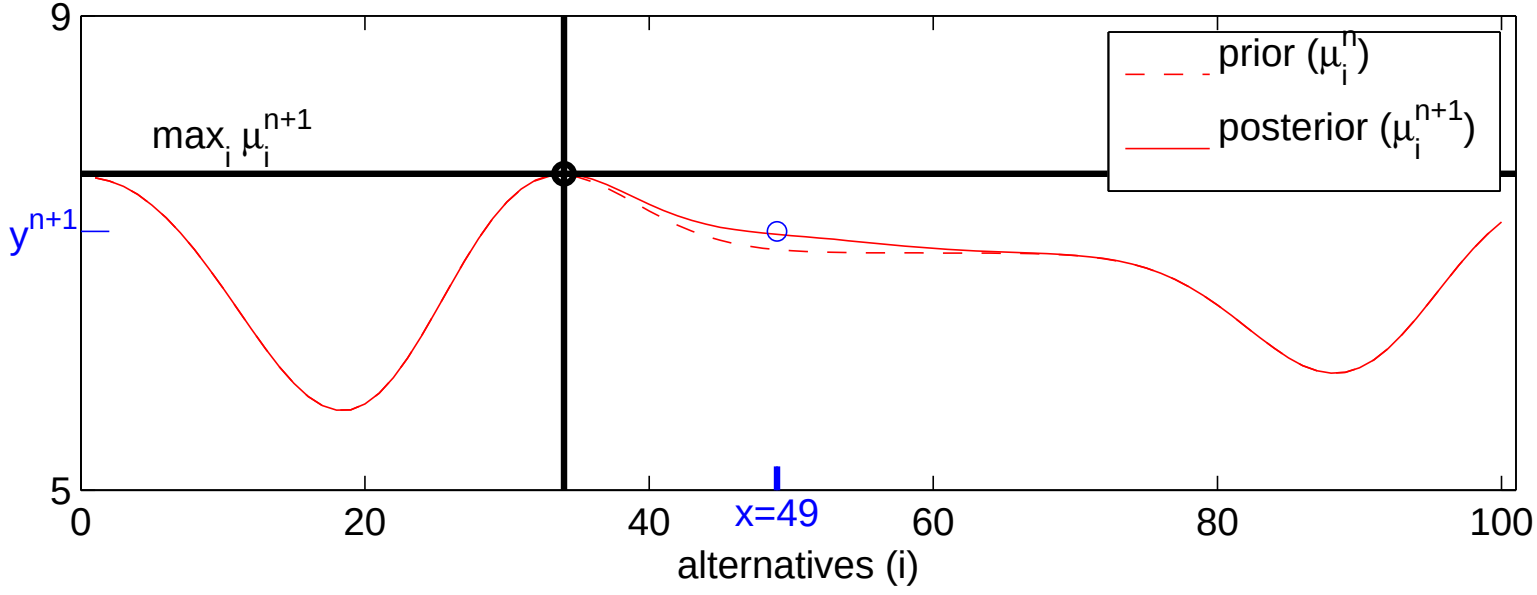


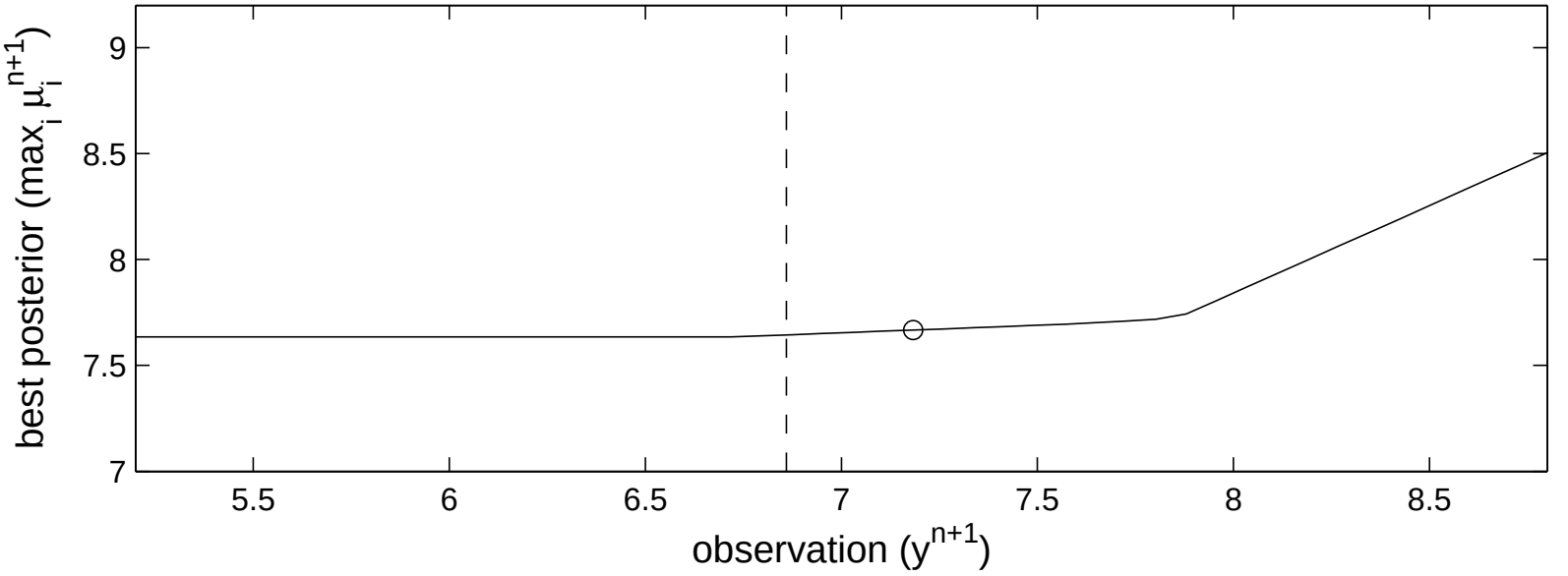
Computing the Knowledge-Gradient



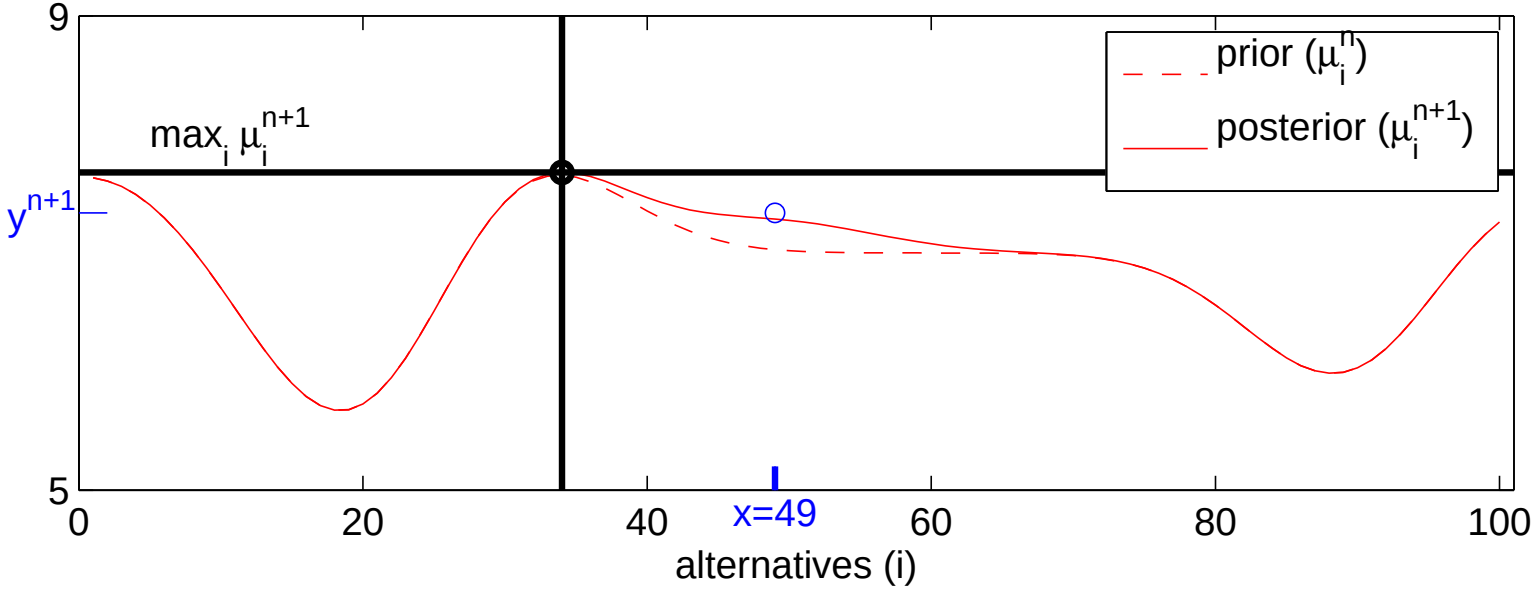


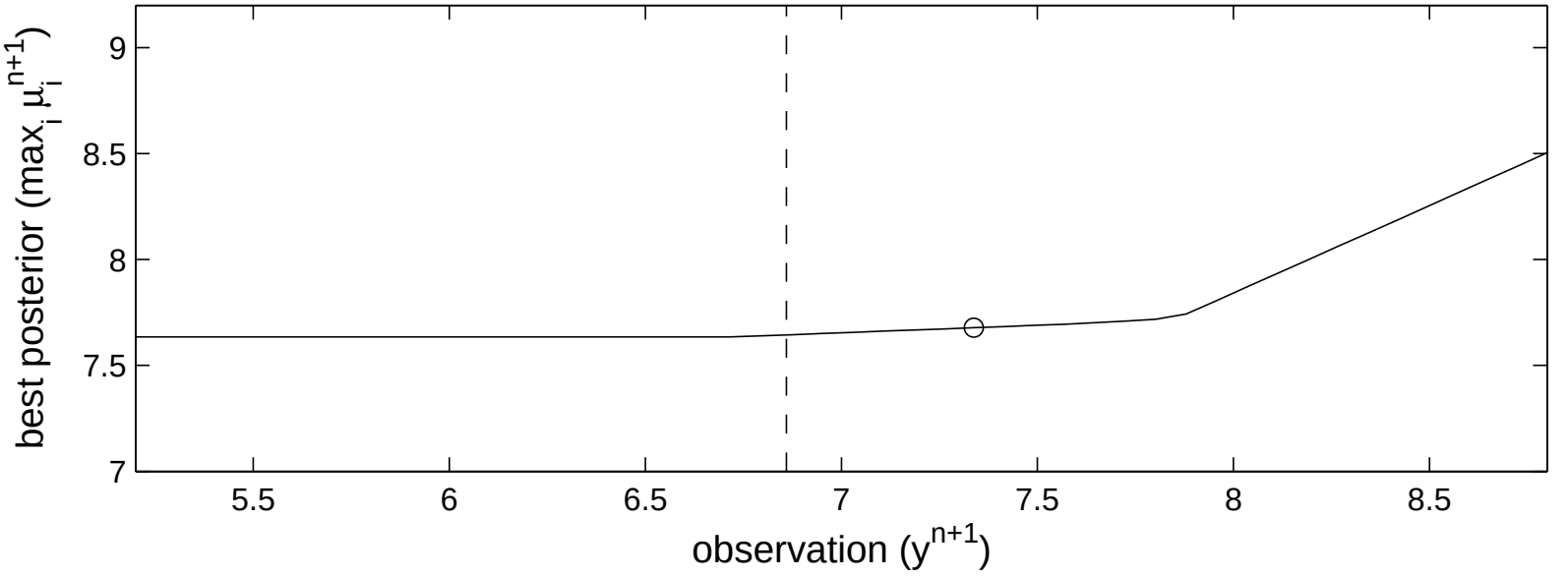
Computing the Knowledge-Gradient



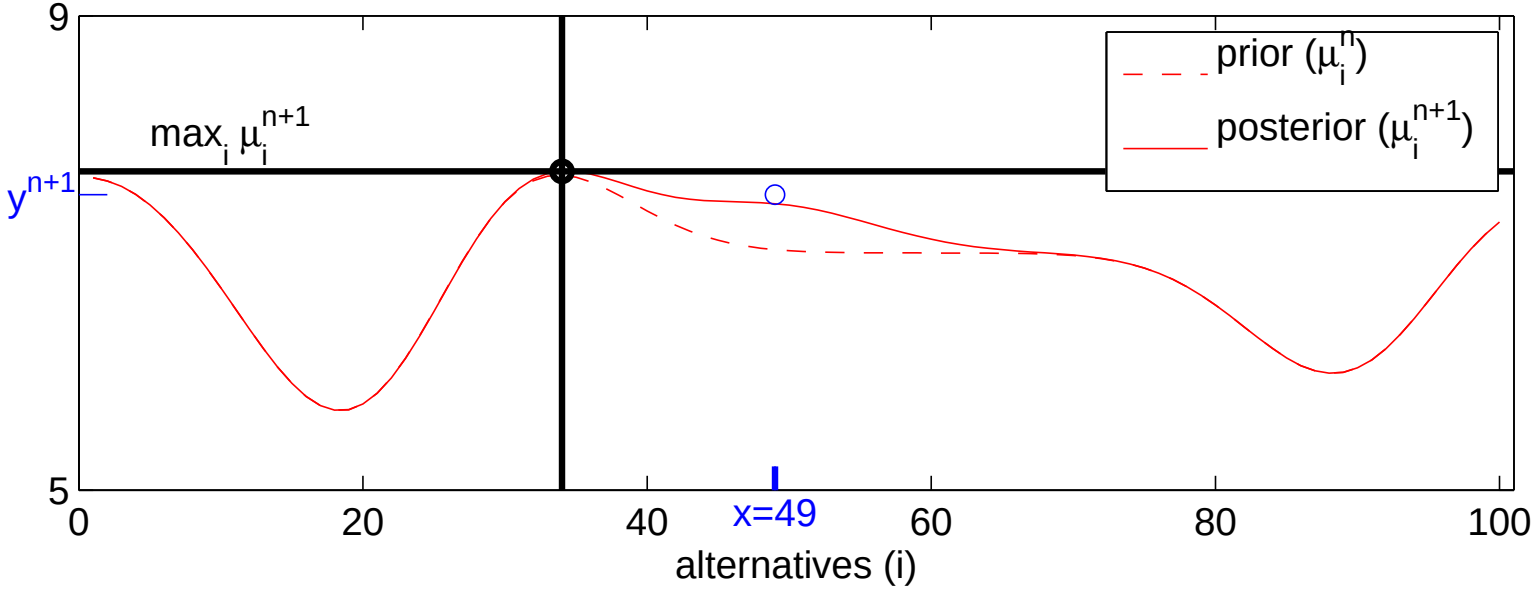


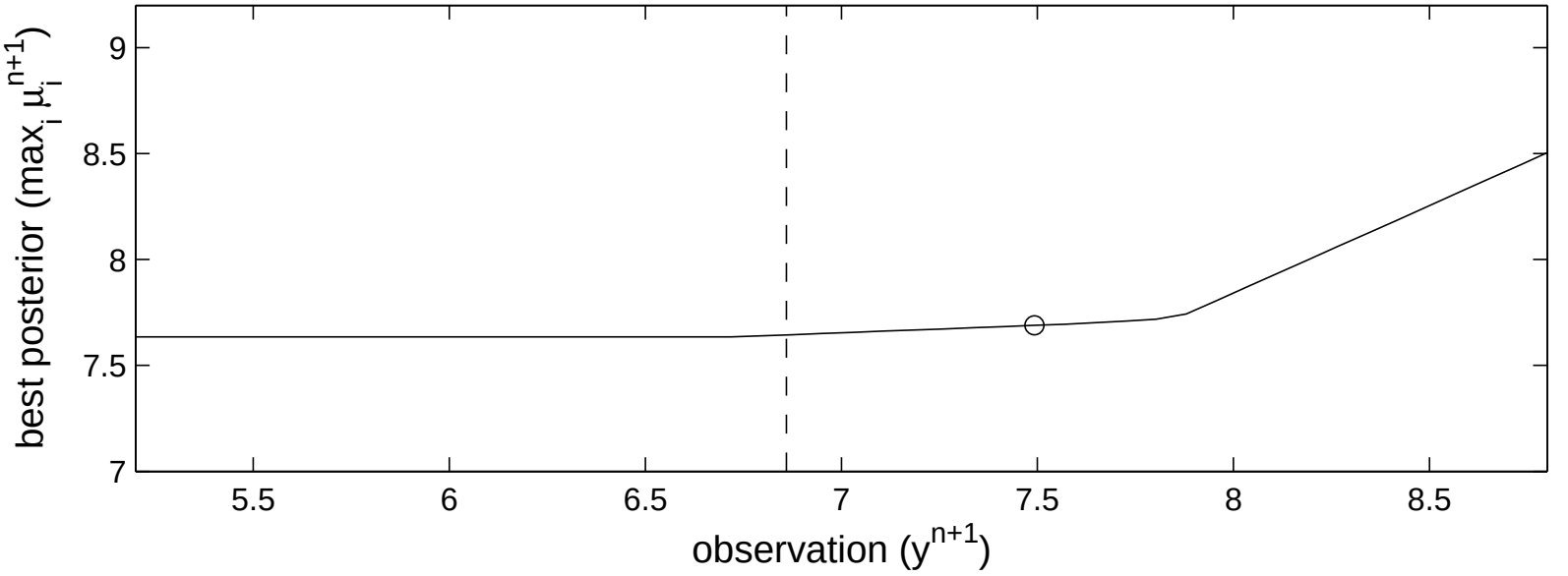
Computing the Knowledge-Gradient



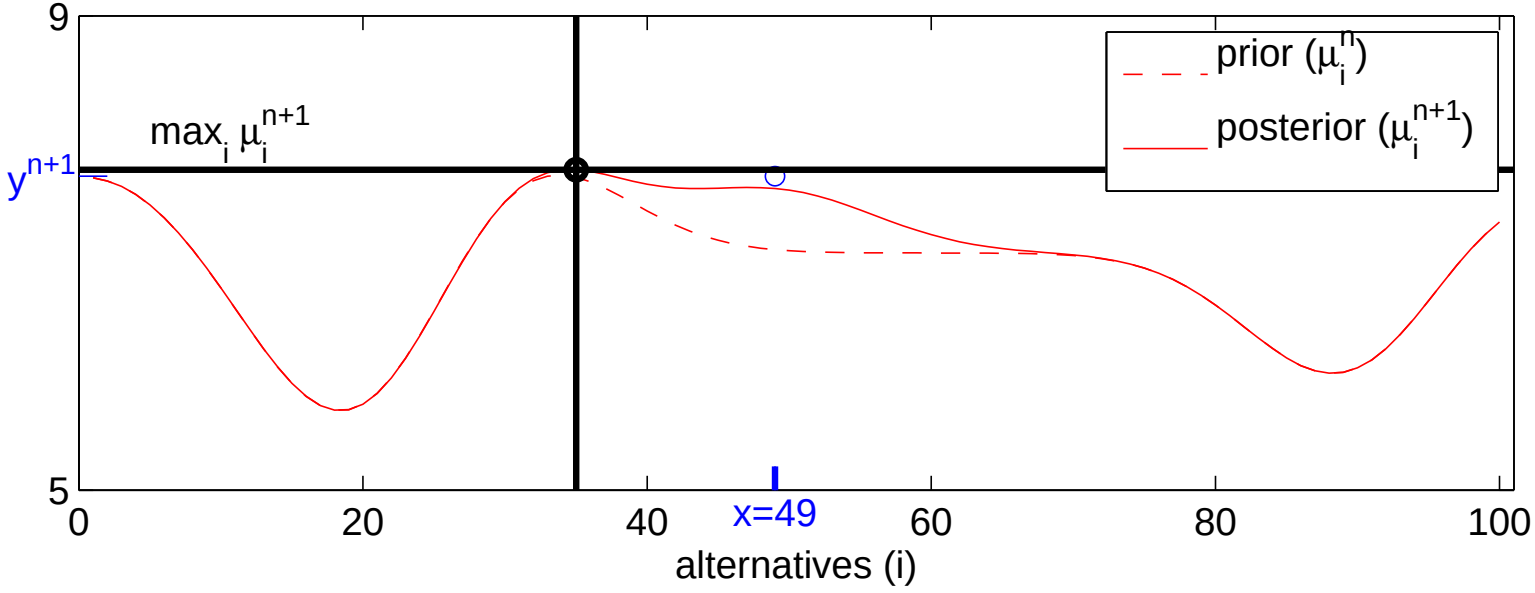


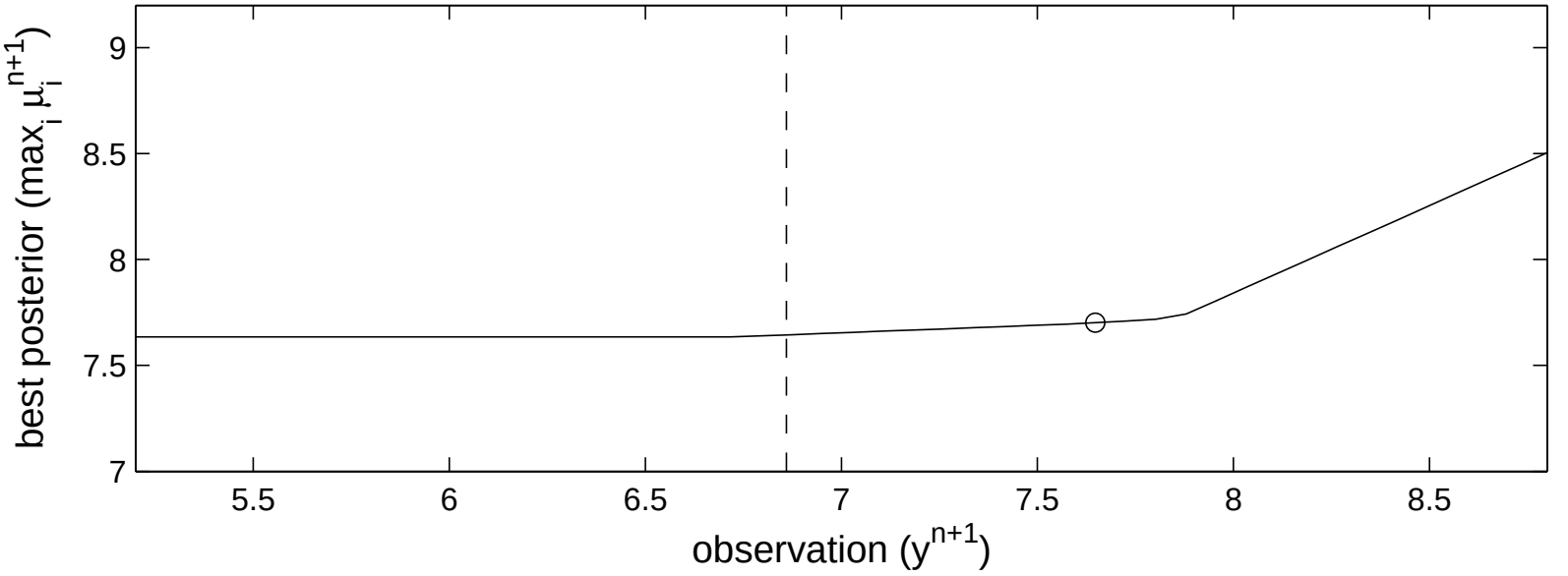
Computing the Knowledge-Gradient



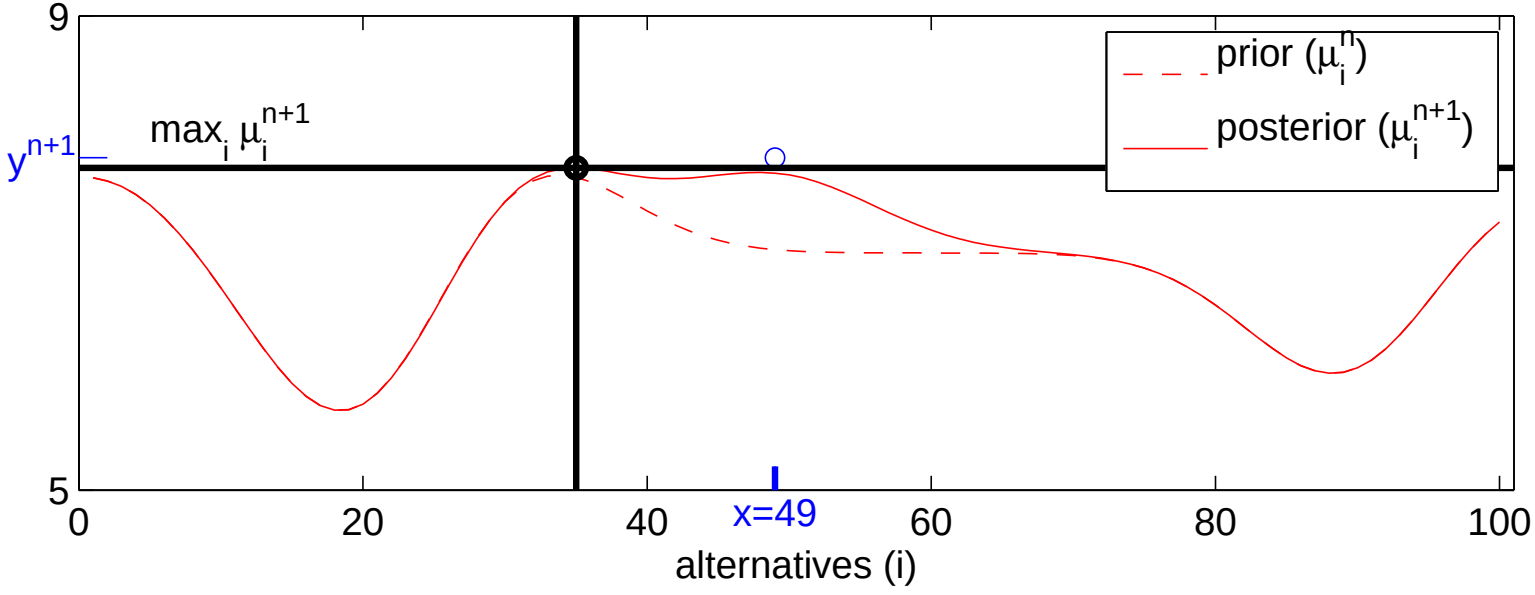


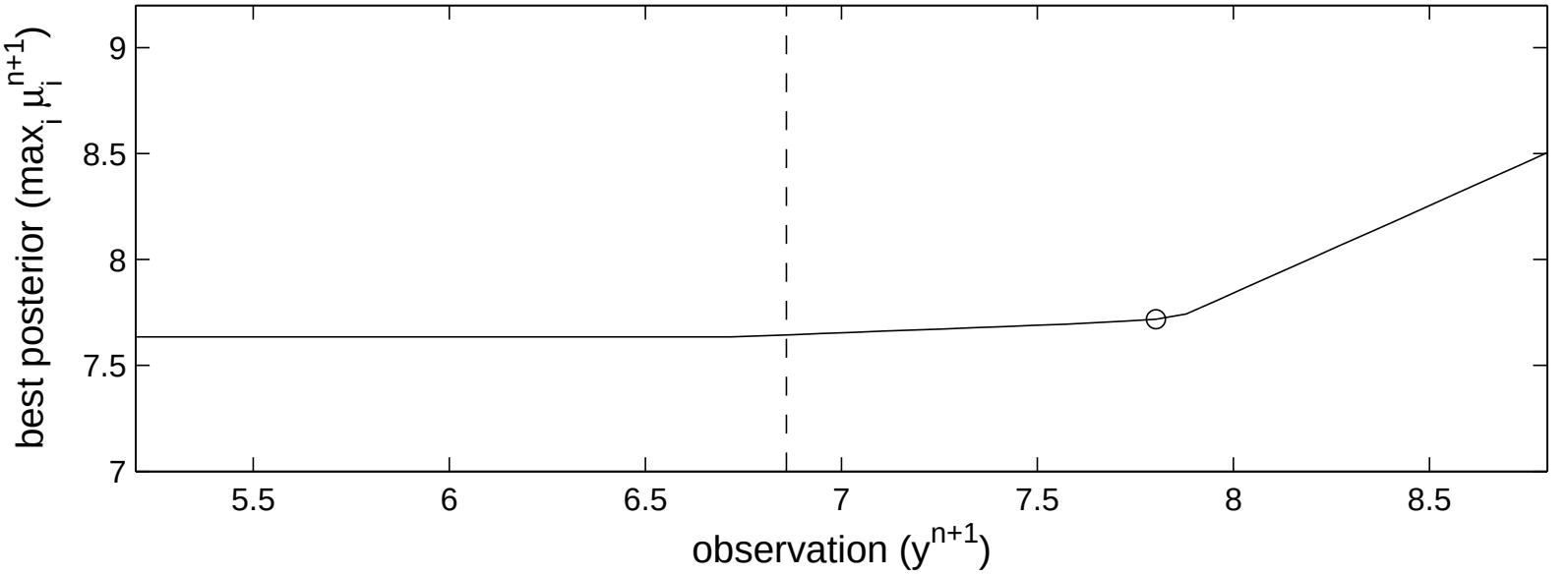
Computing the Knowledge-Gradient



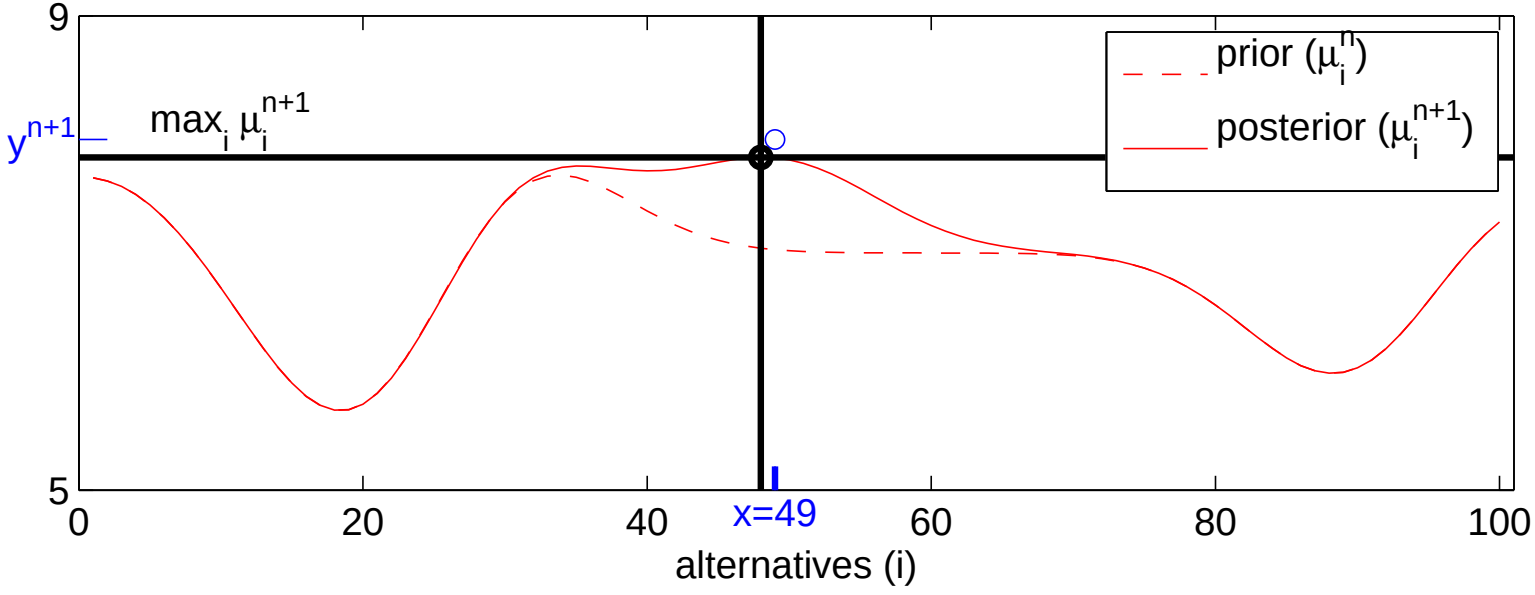


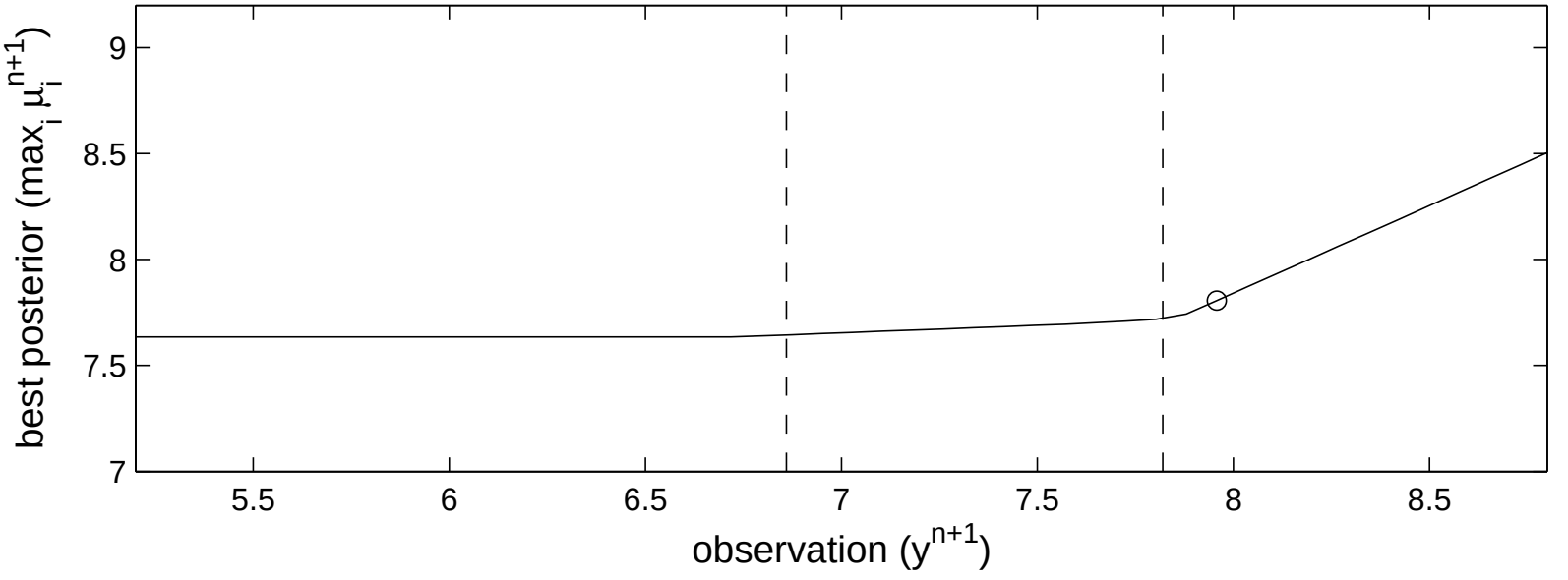
Computing the Knowledge-Gradient



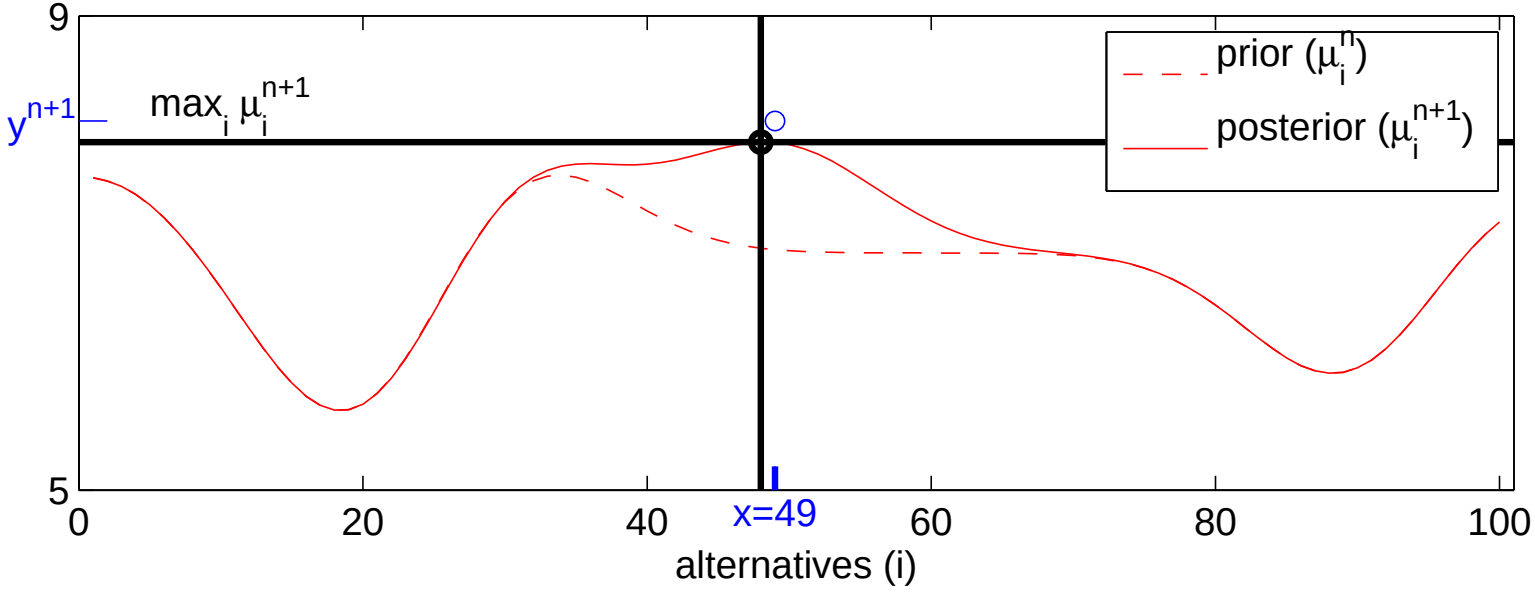


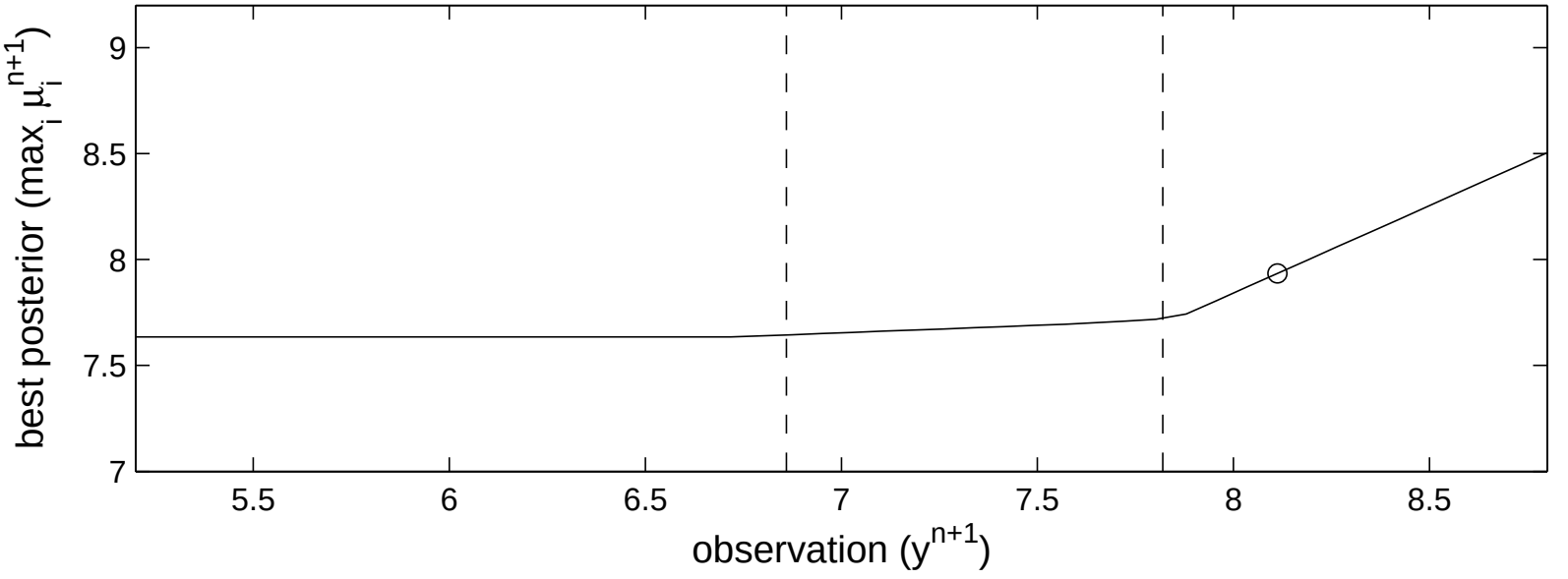
Computing the Knowledge-Gradient



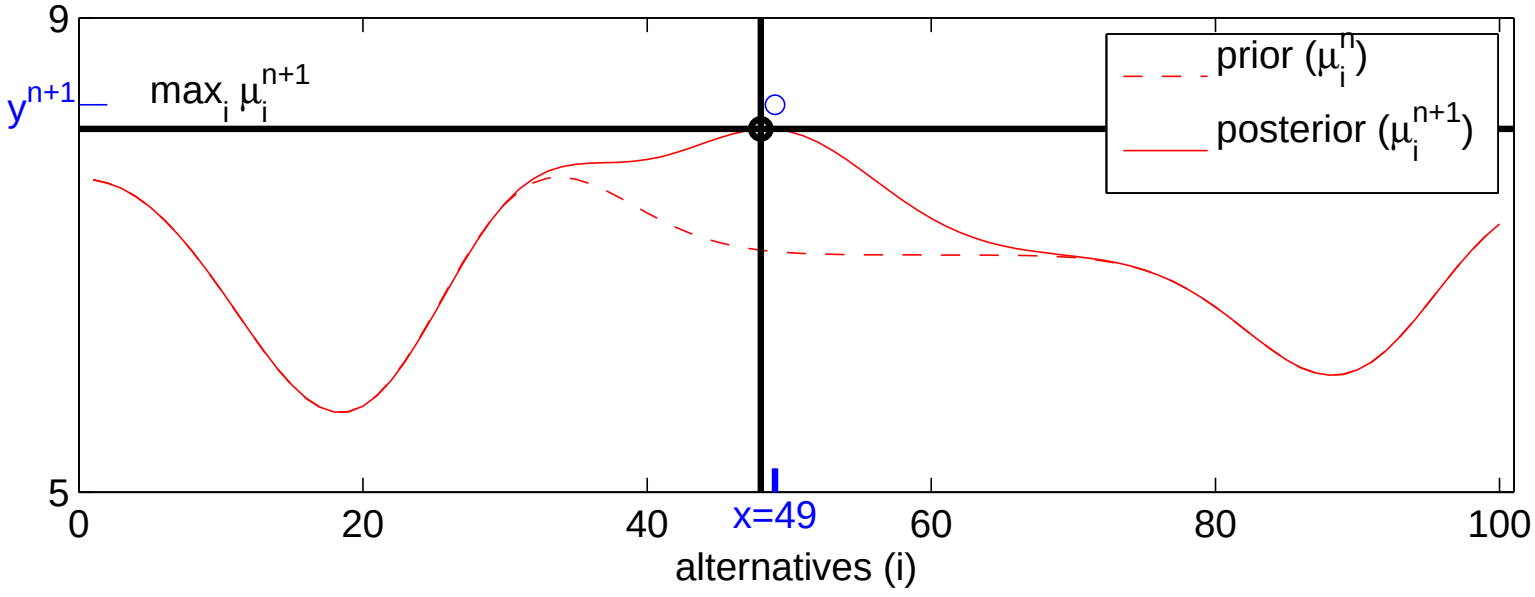


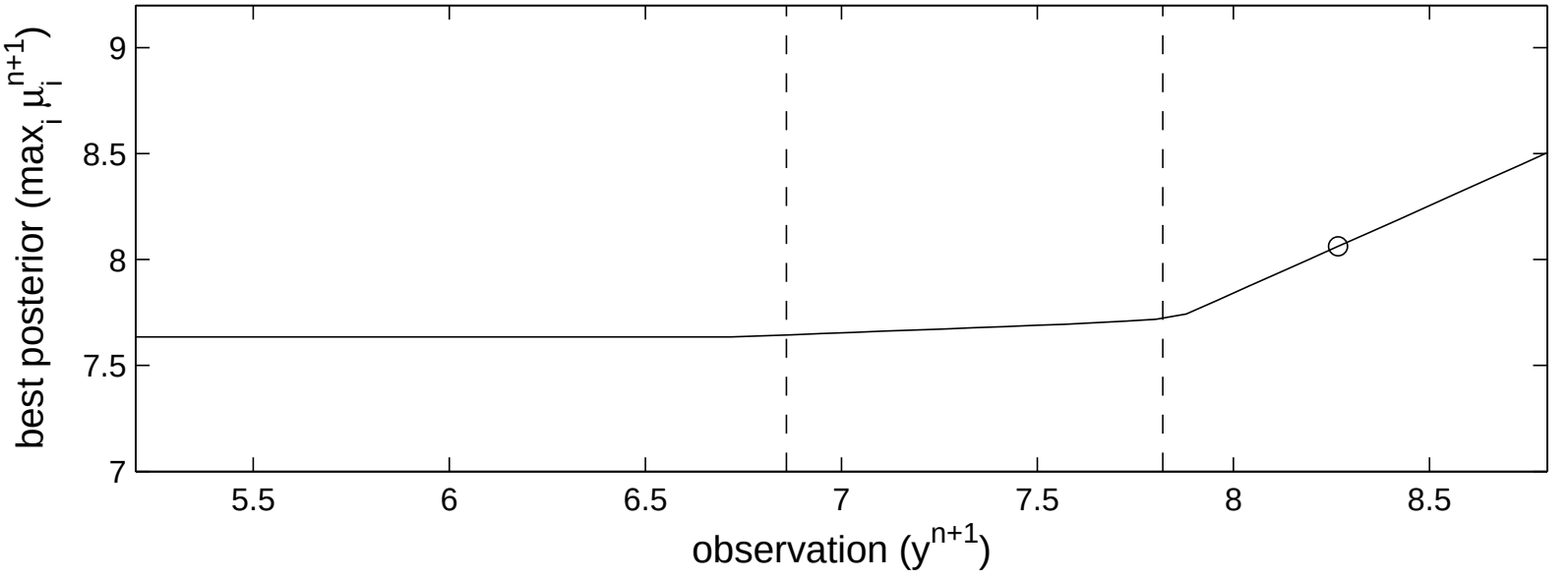
Computing the Knowledge-Gradient



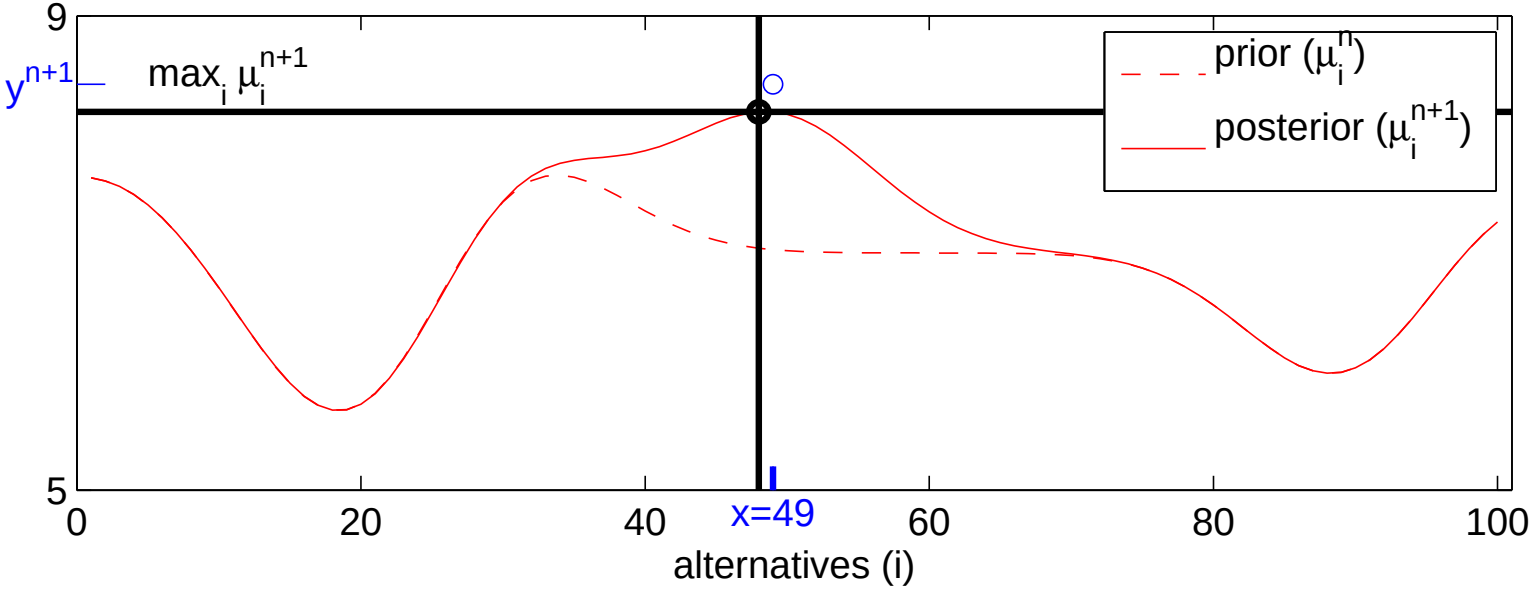


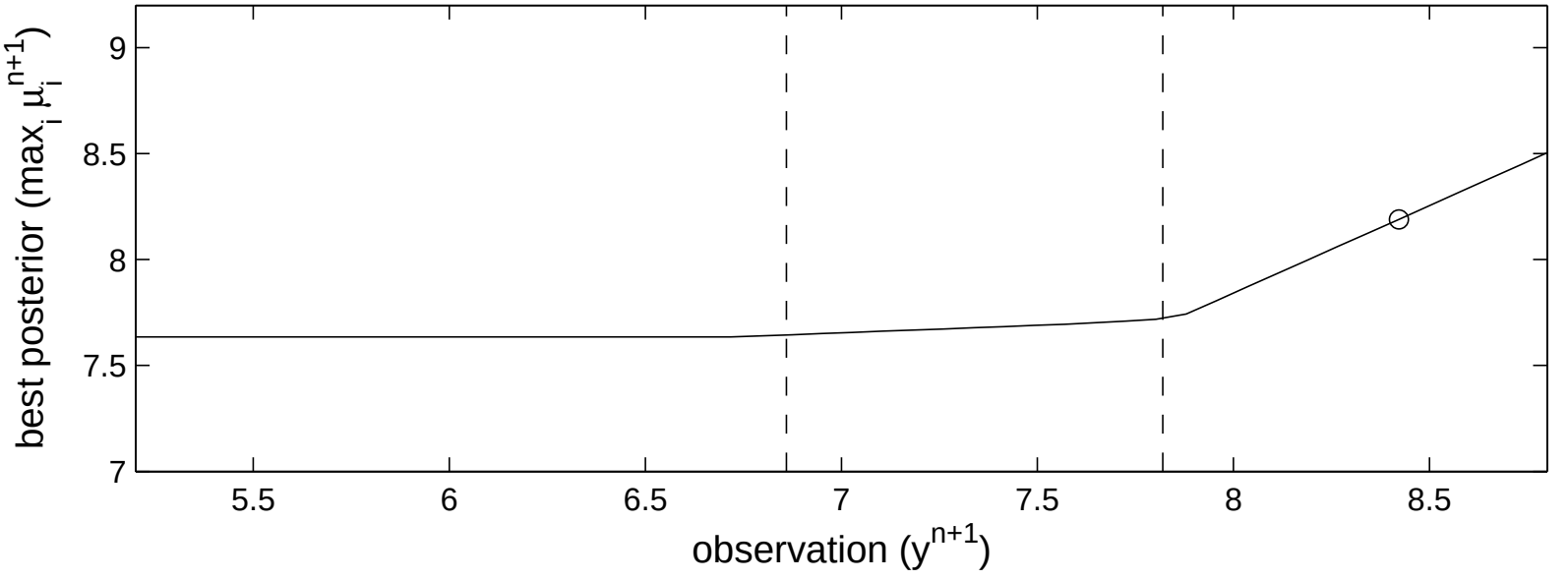
Computing the Knowledge-Gradient



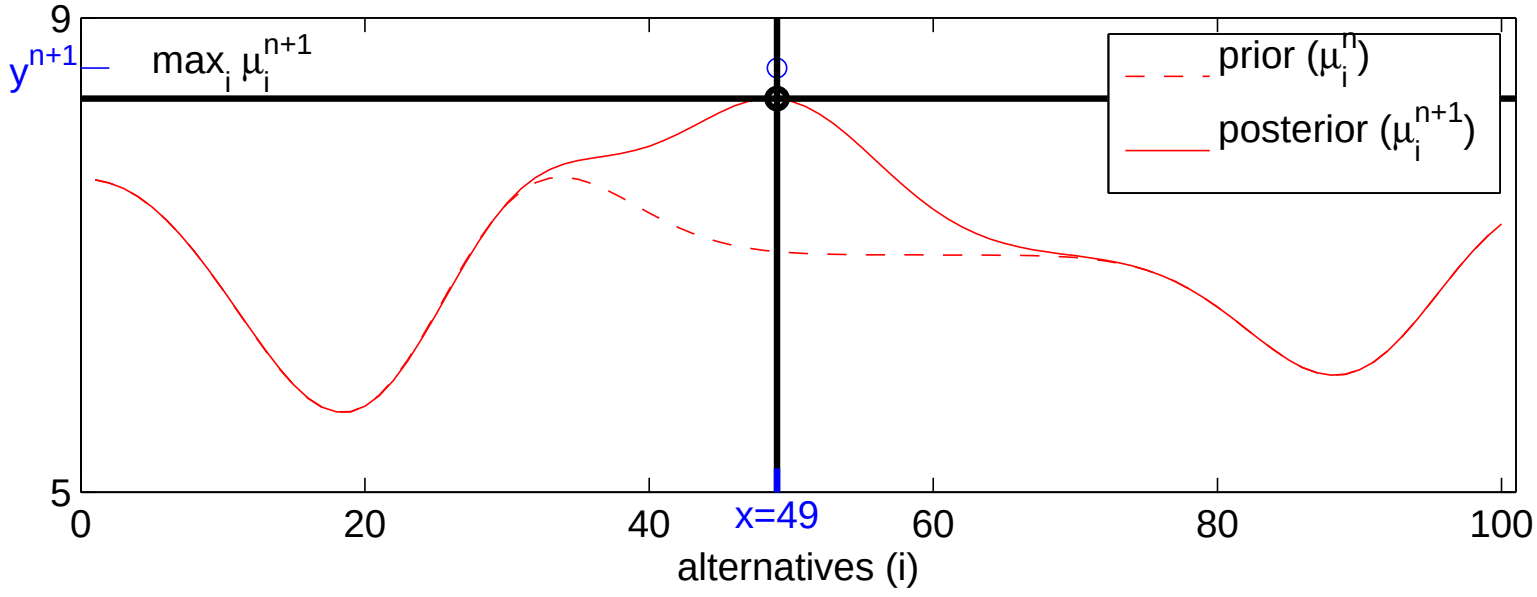


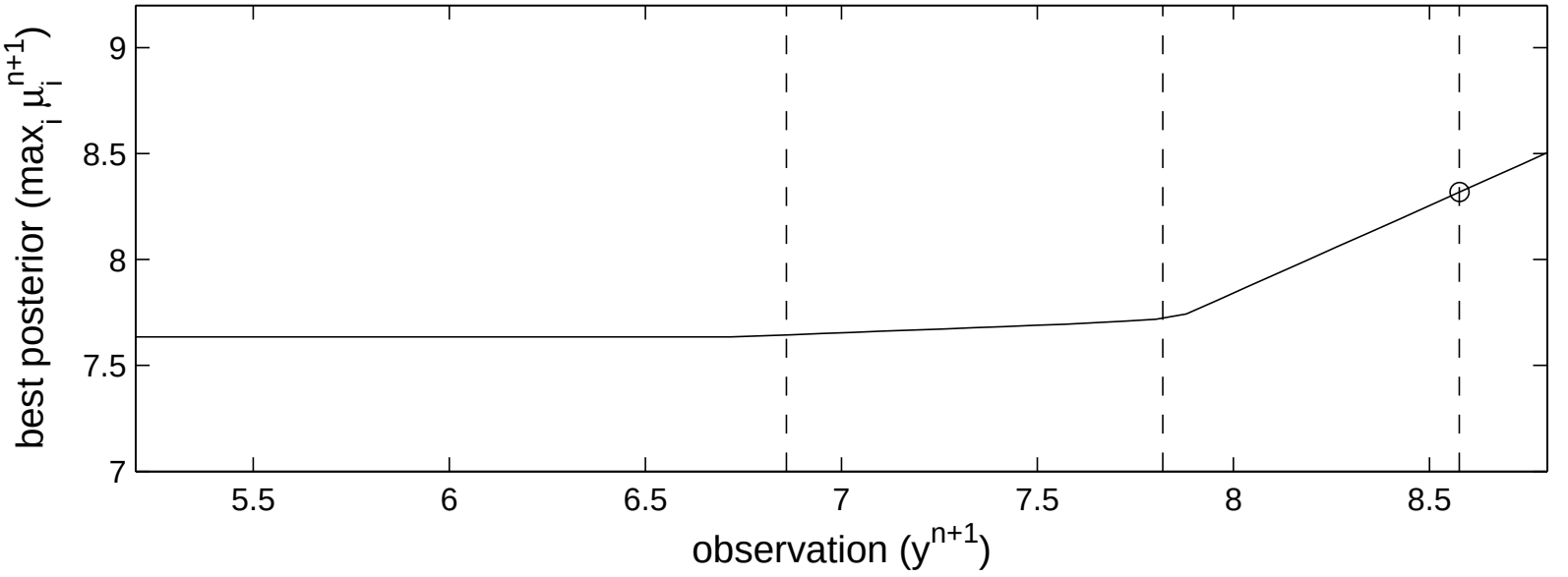
Computing the Knowledge-Gradient



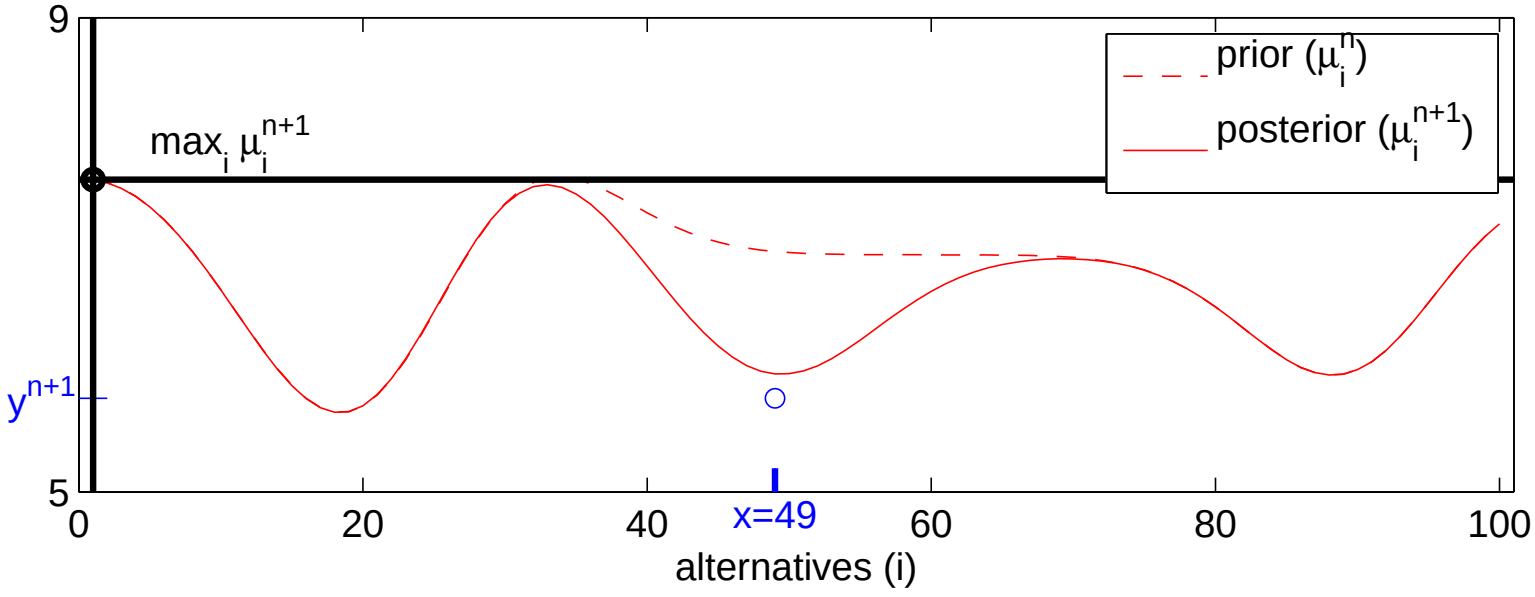


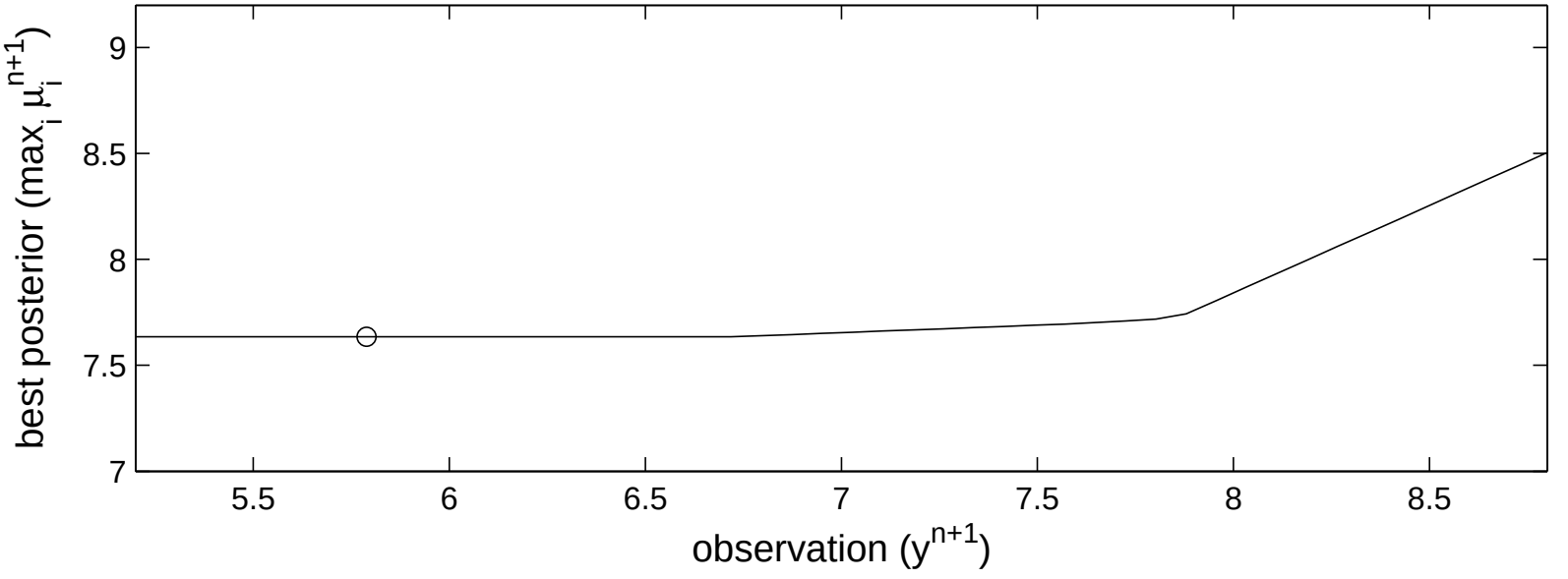
Computing the Knowledge-Gradient



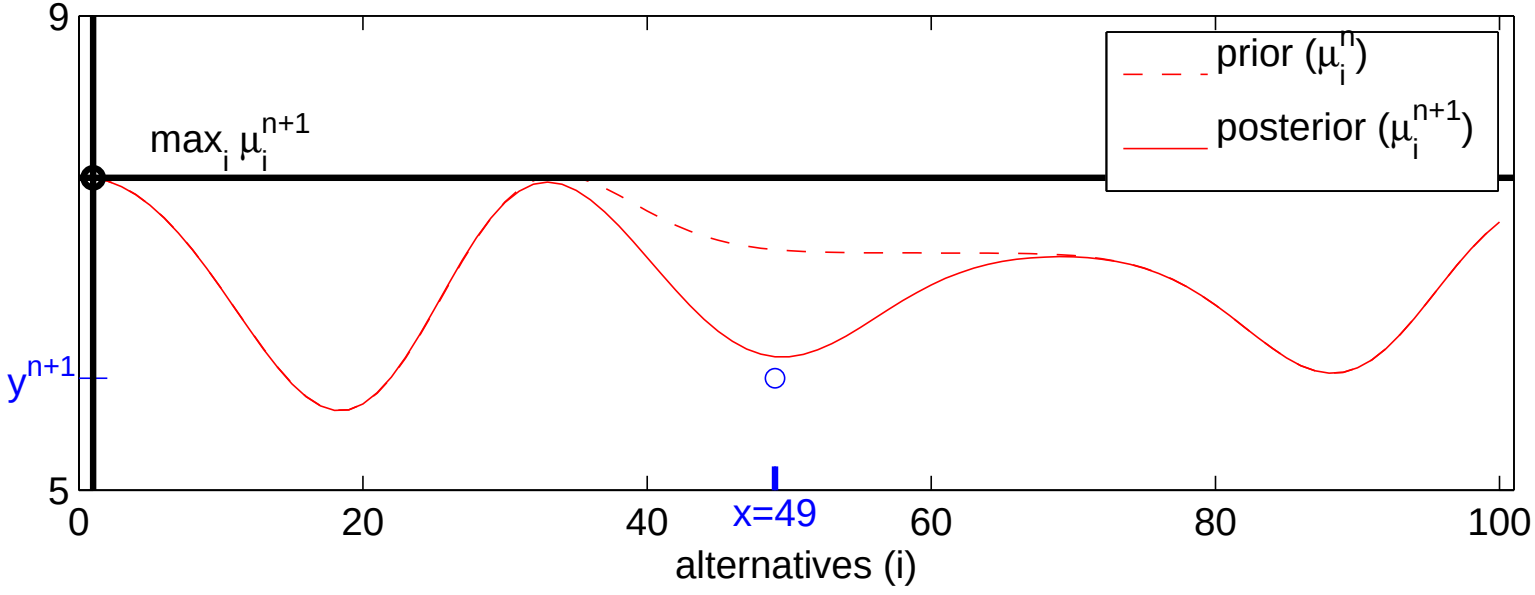


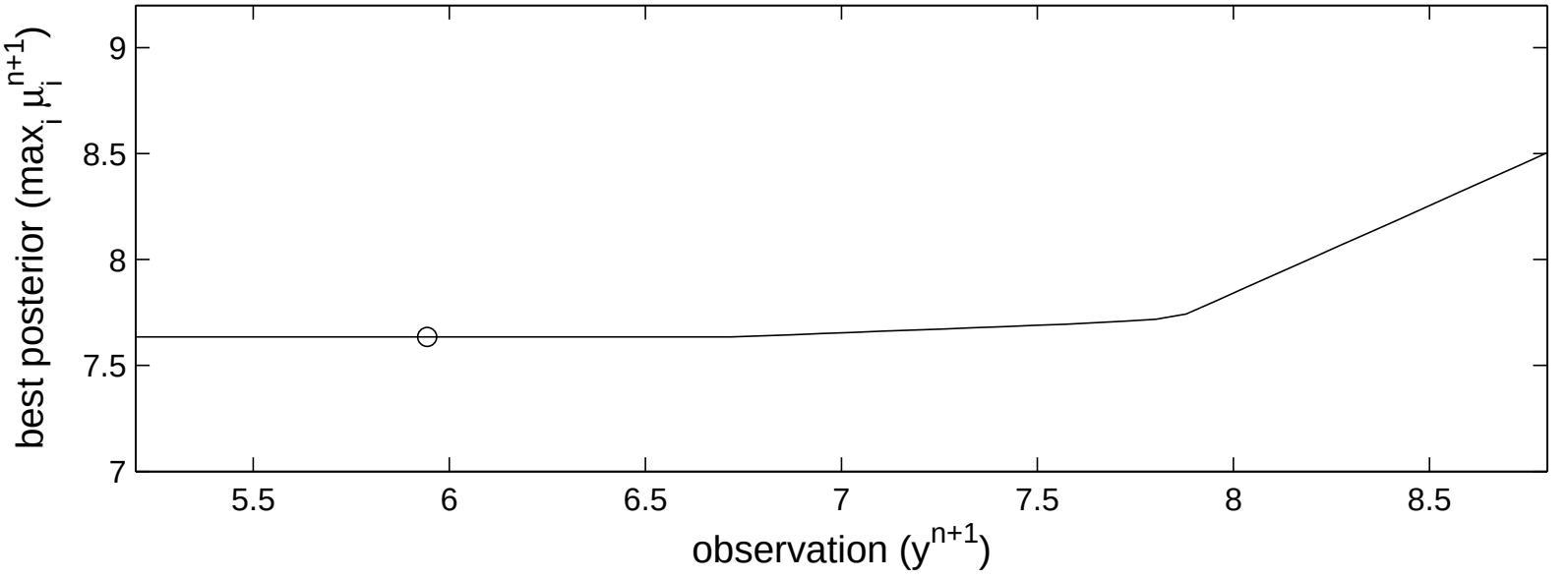
Computing the Knowledge-Gradient



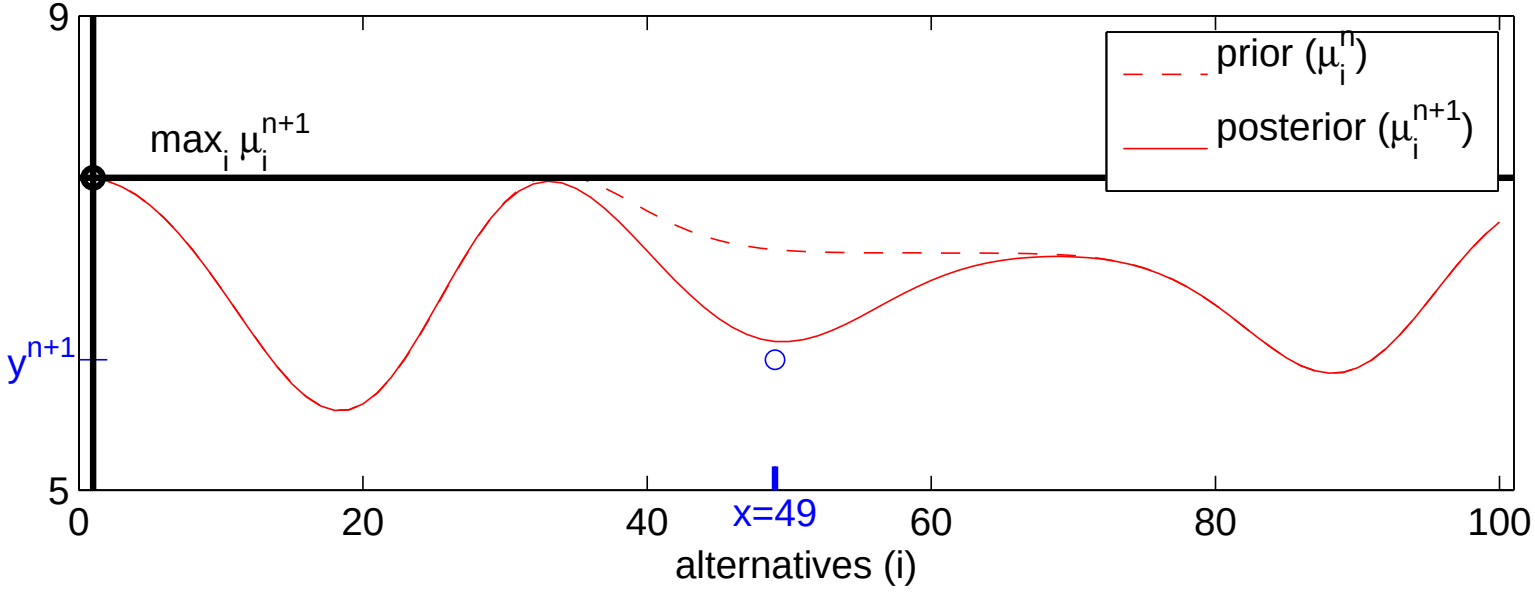


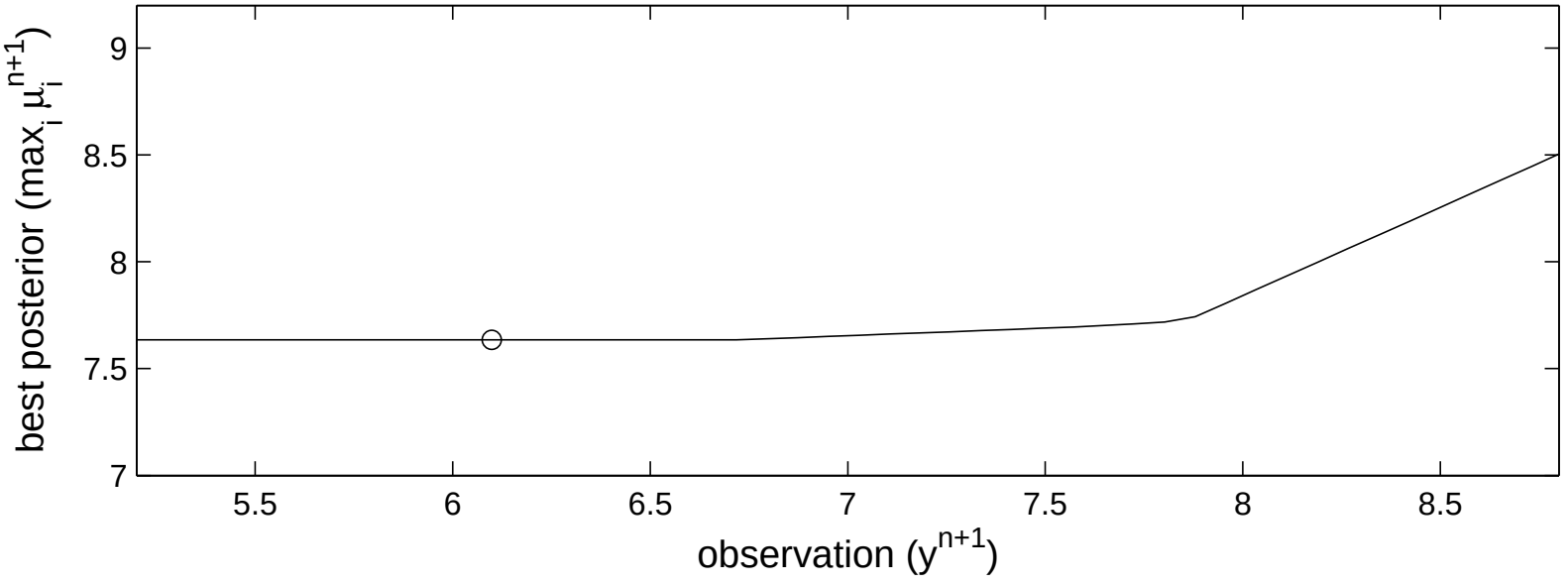
Computing the Knowledge-Gradient



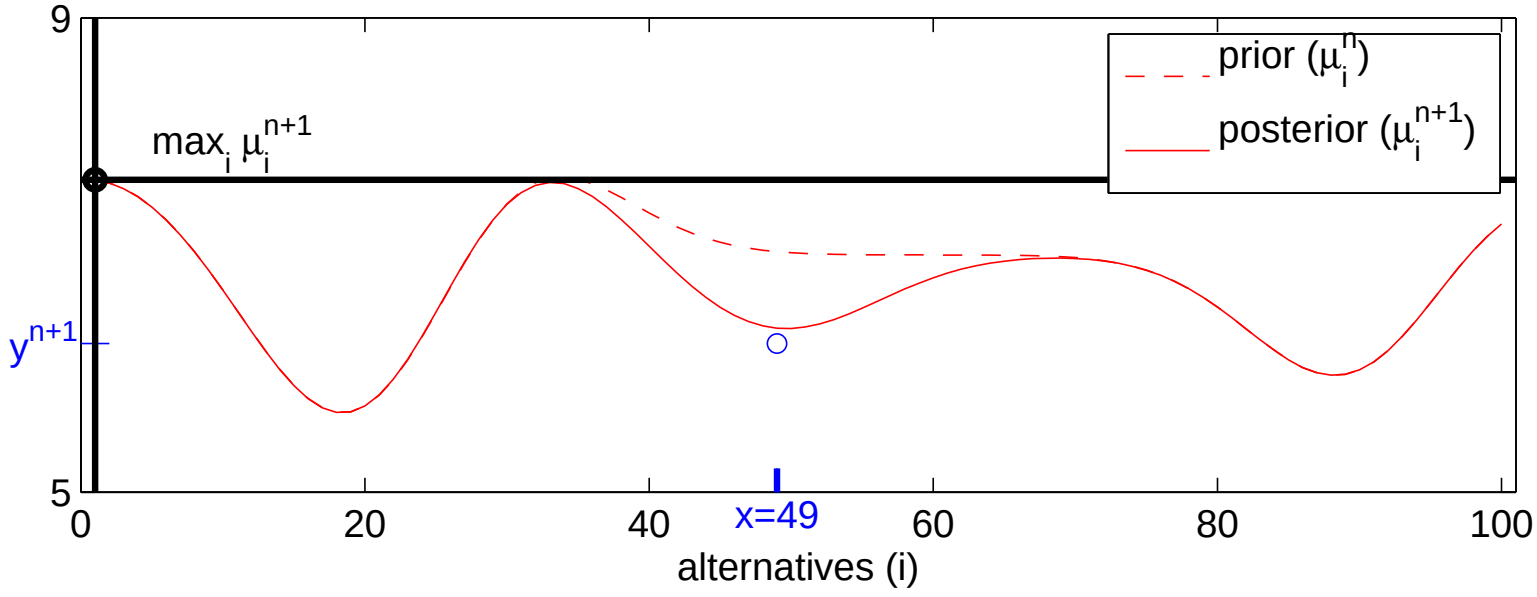


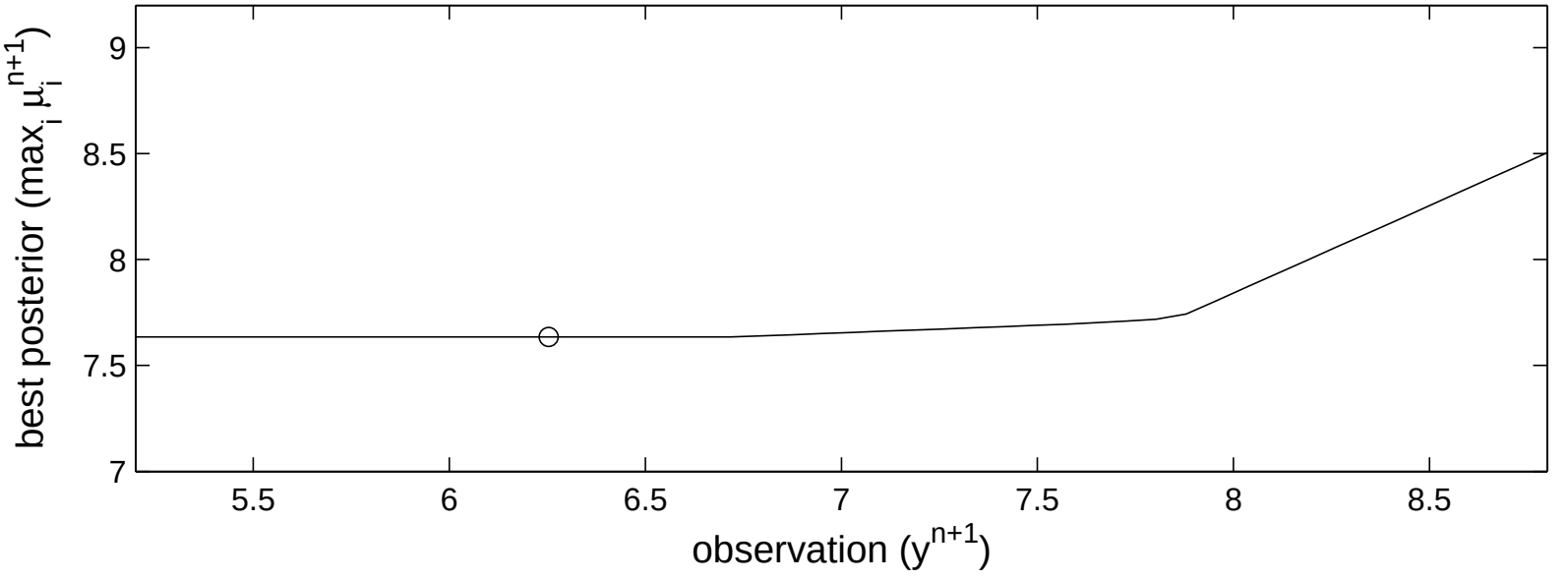
Computing the Knowledge-Gradient



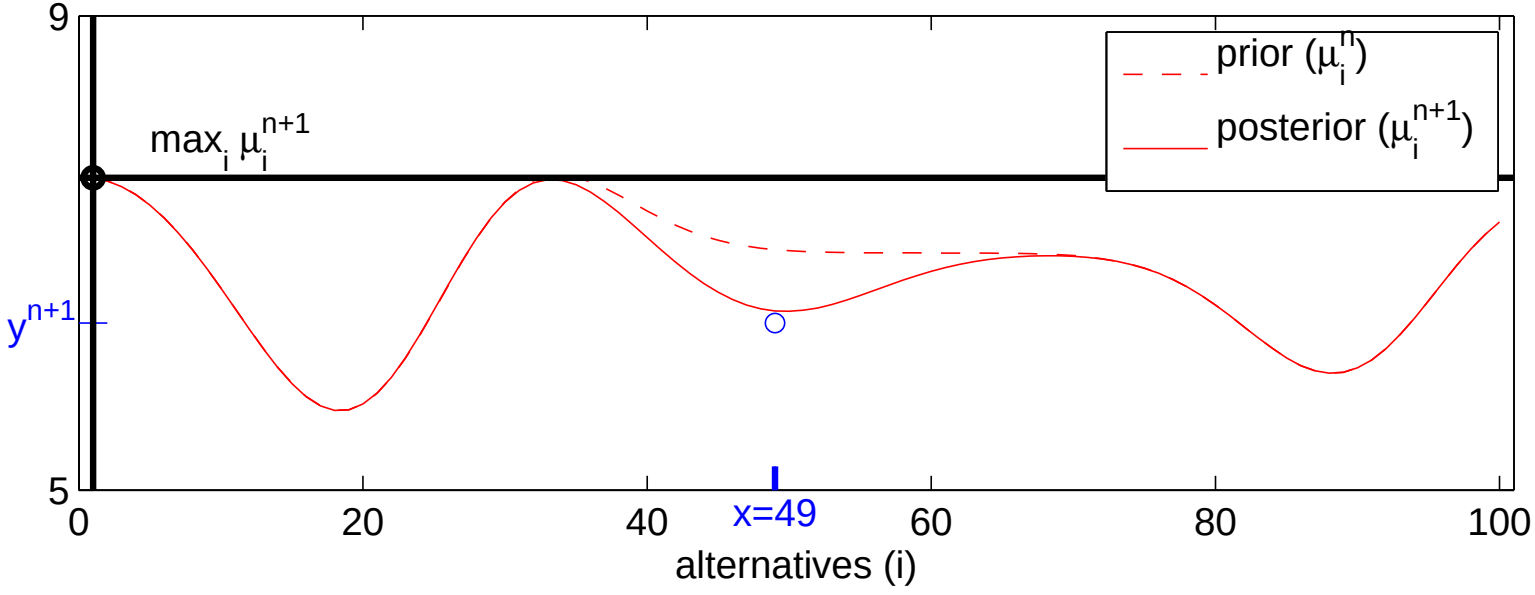


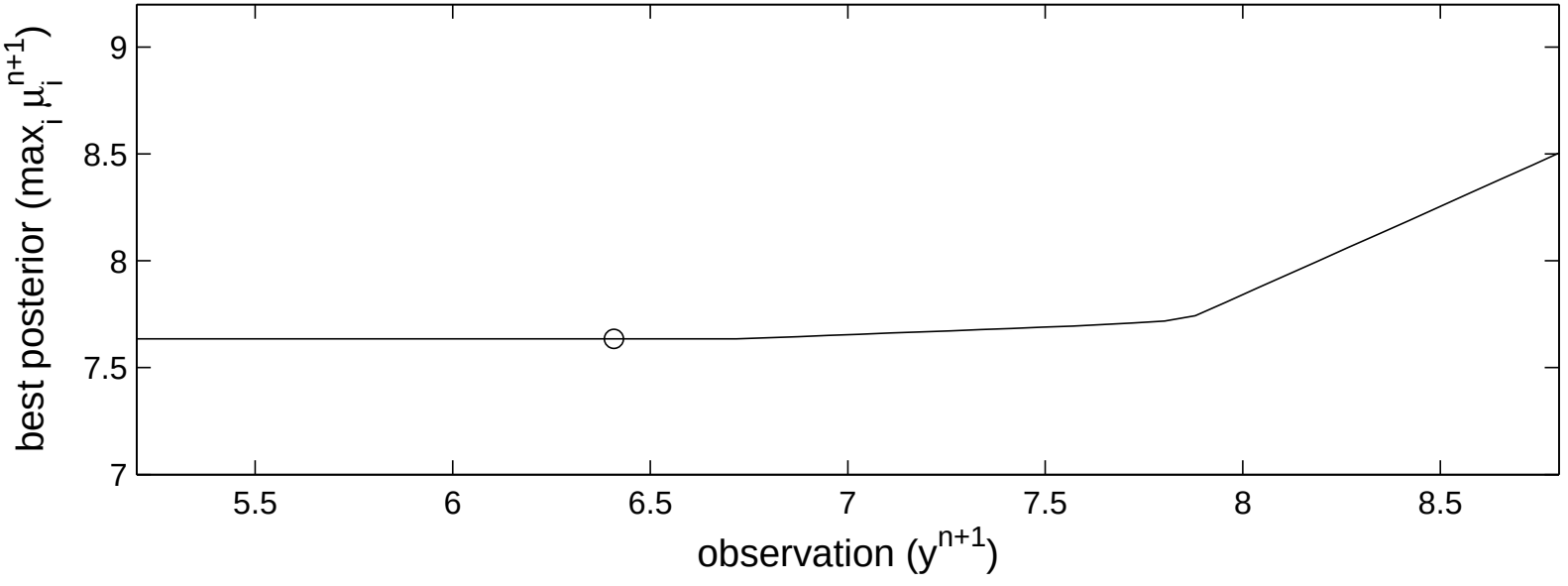
Computing the Knowledge-Gradient



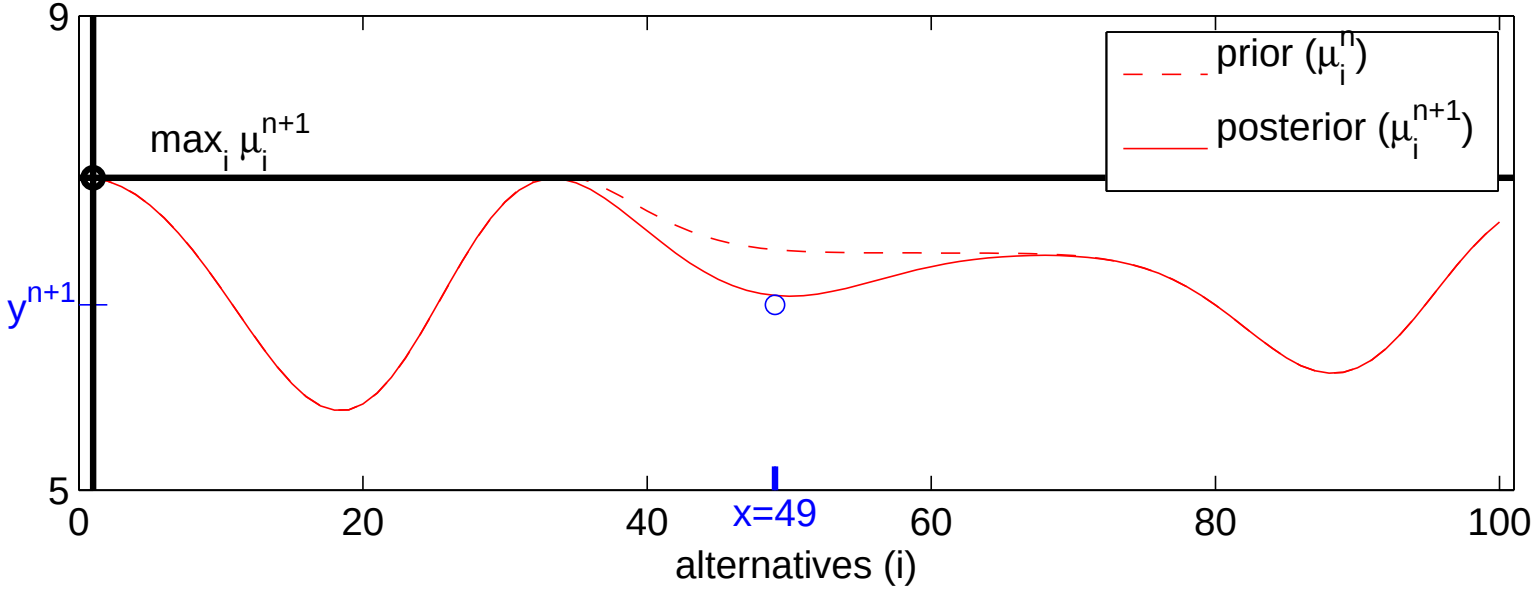


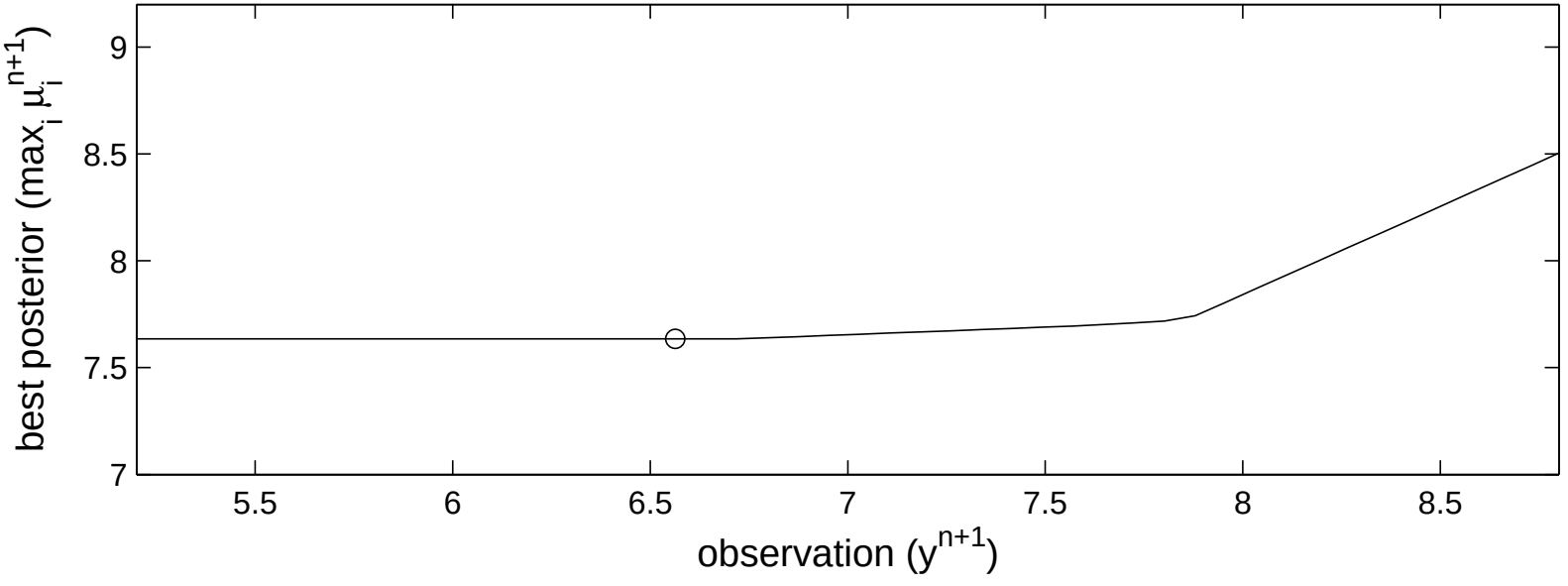
Computing the Knowledge-Gradient



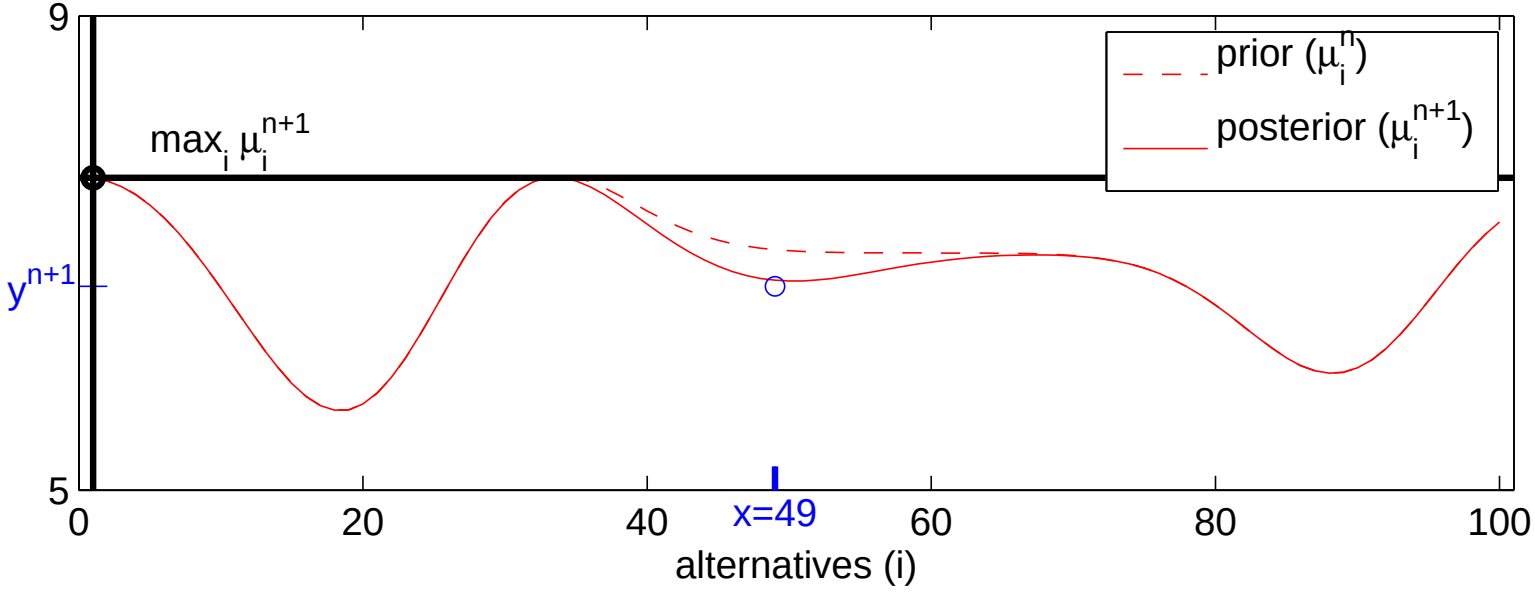


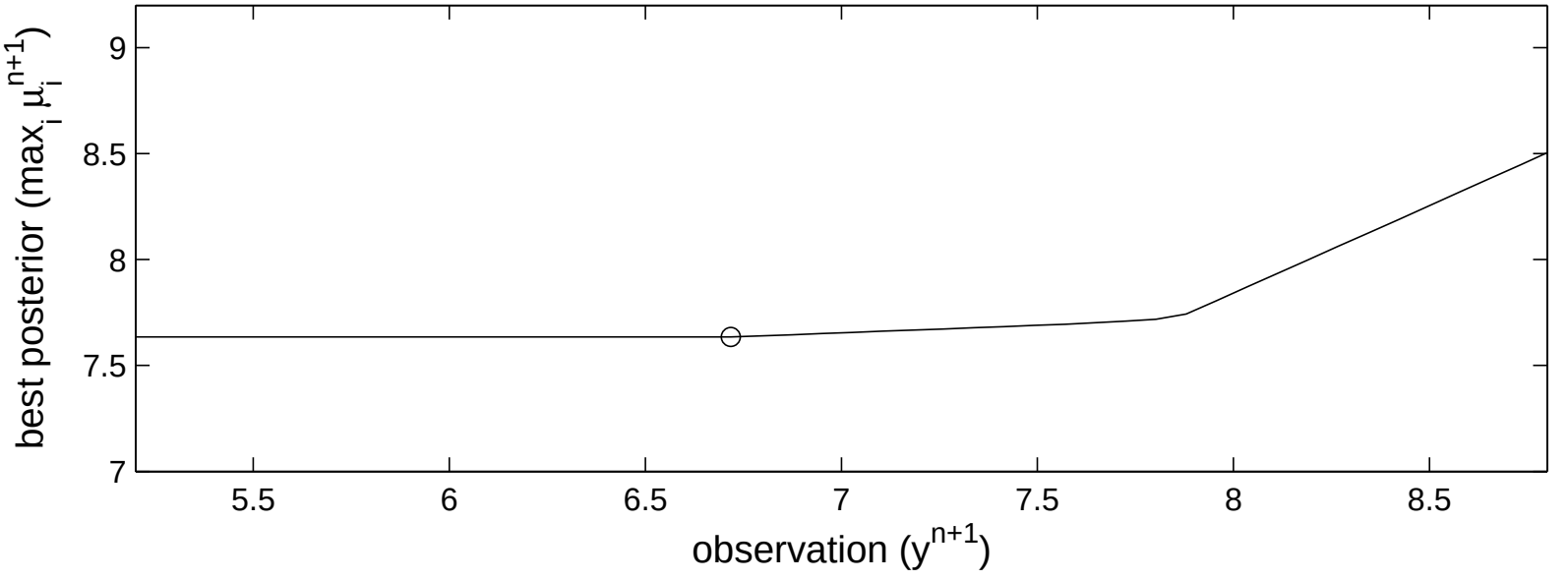
Computing the Knowledge-Gradient



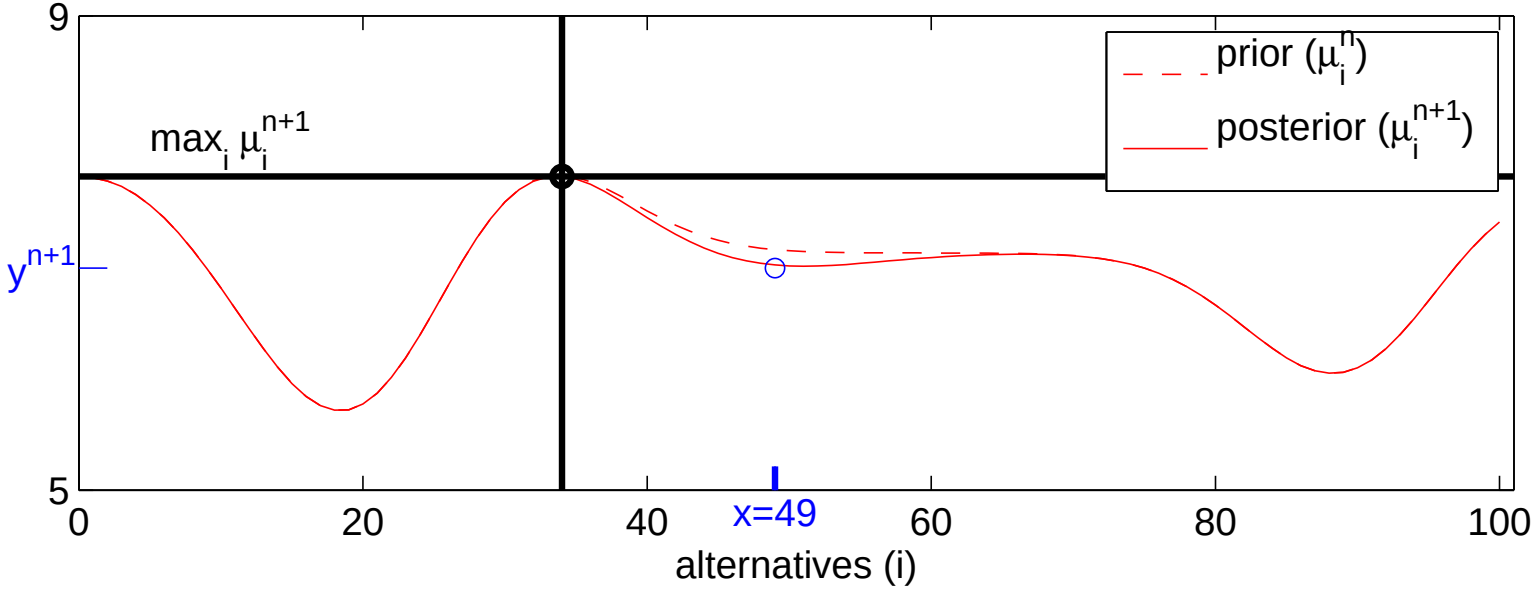


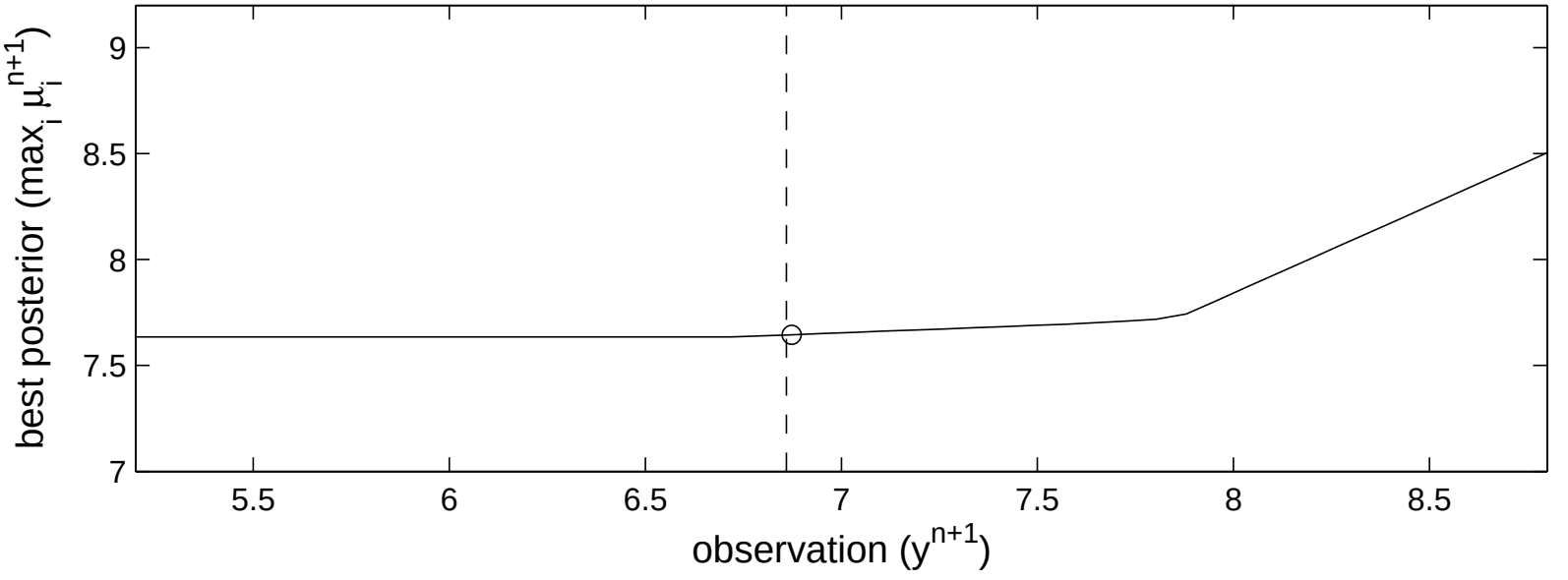
Computing the Knowledge-Gradient



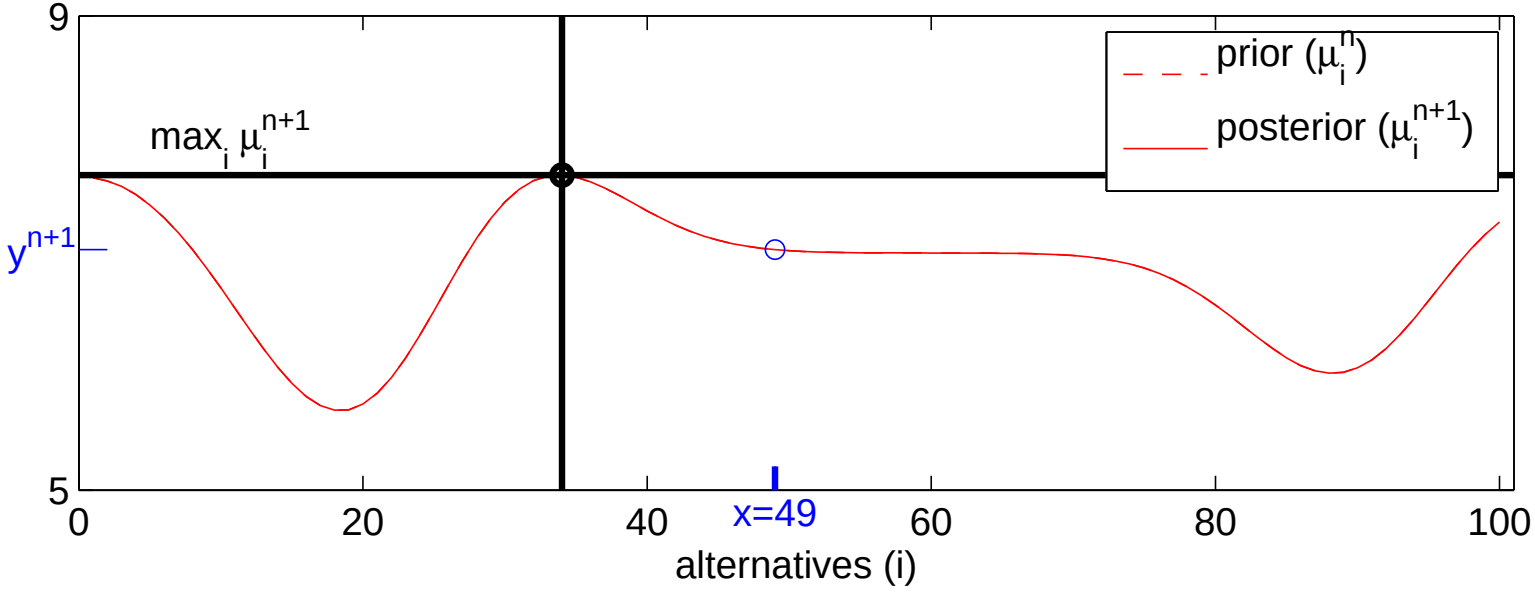


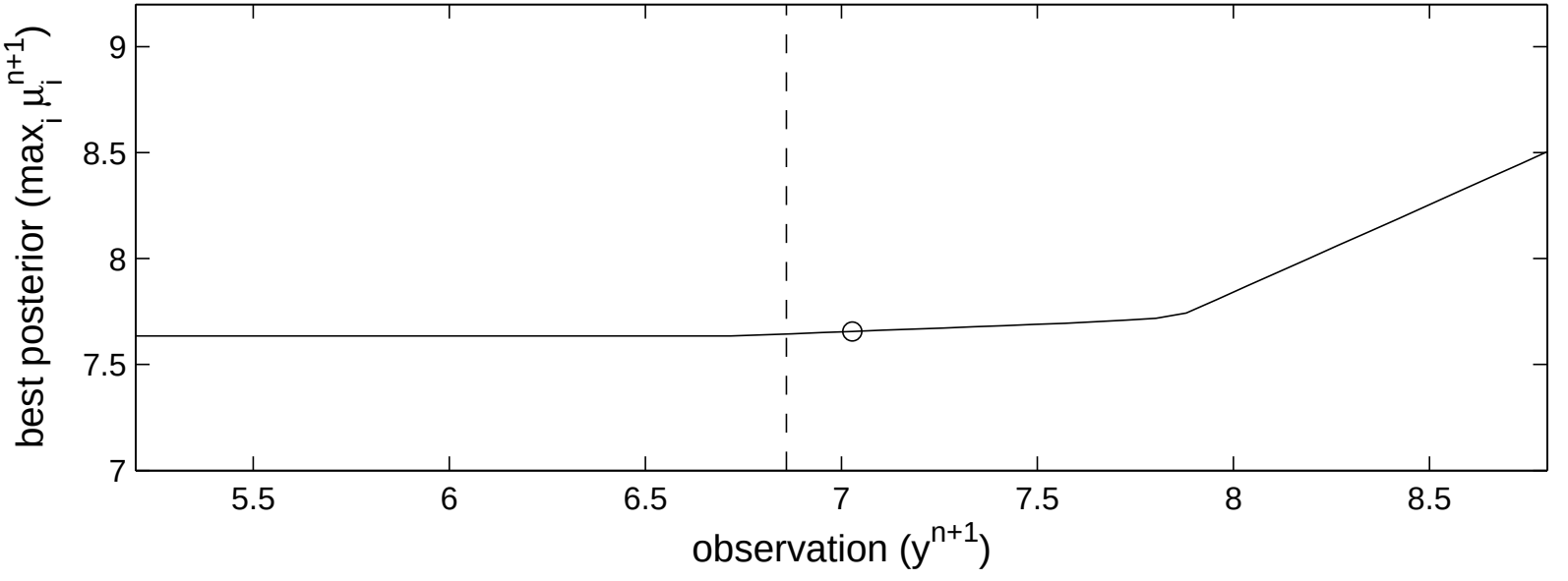
Computing the Knowledge-Gradient



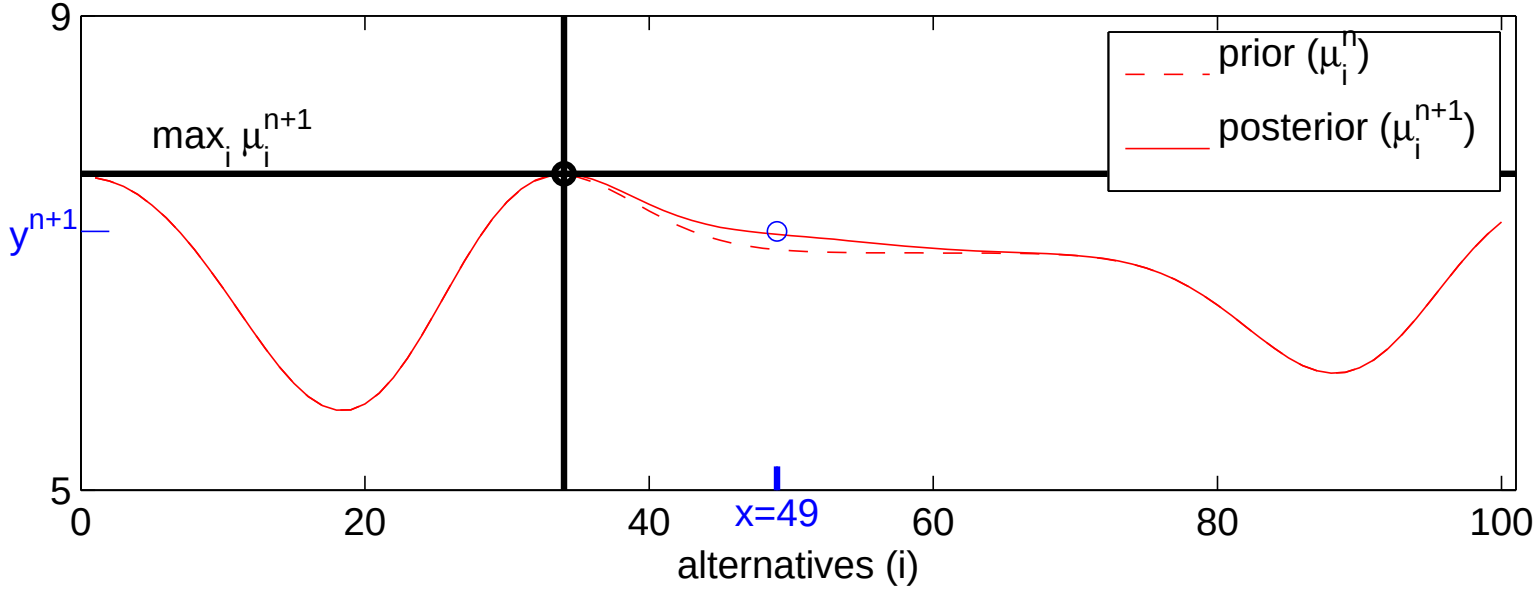


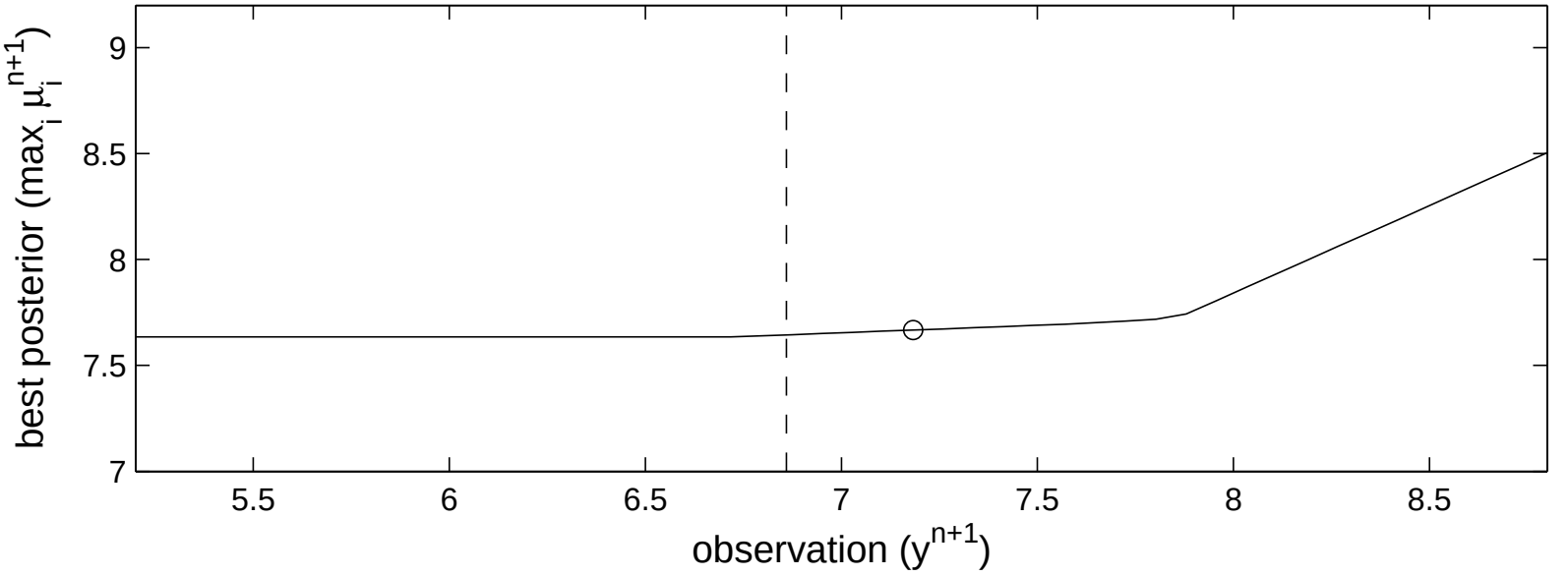
Computing the Knowledge-Gradient



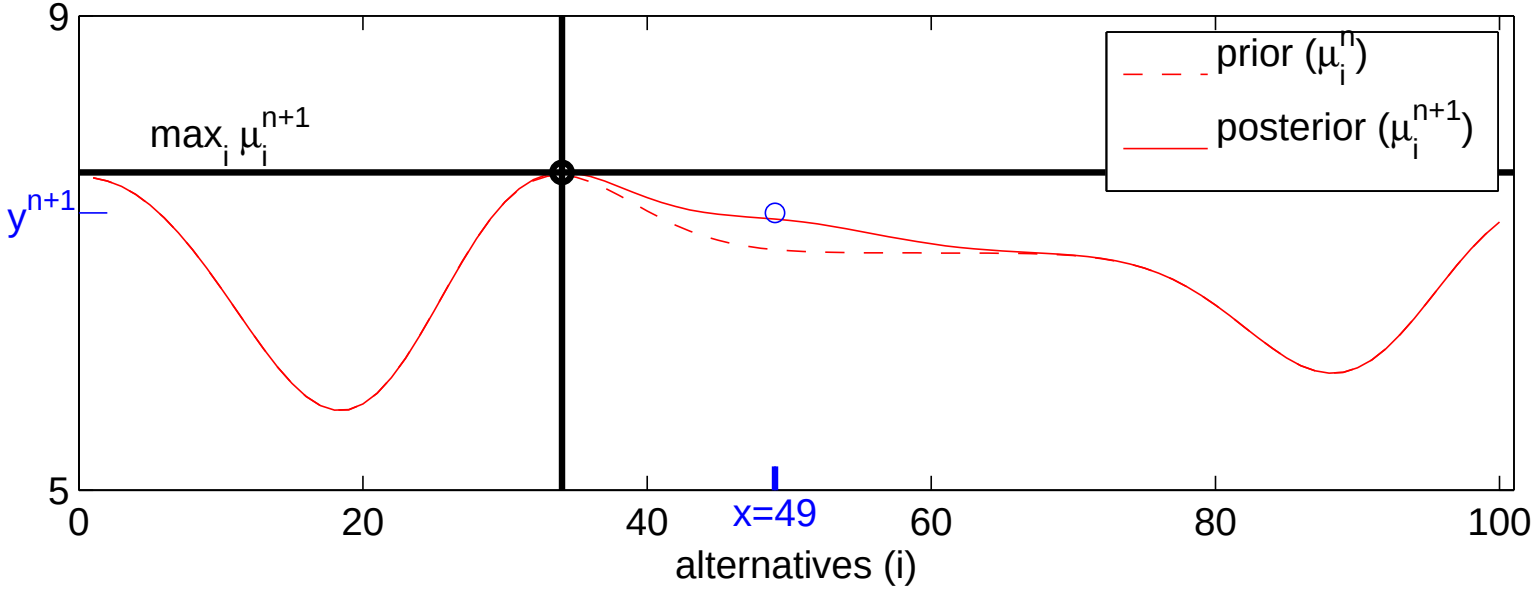


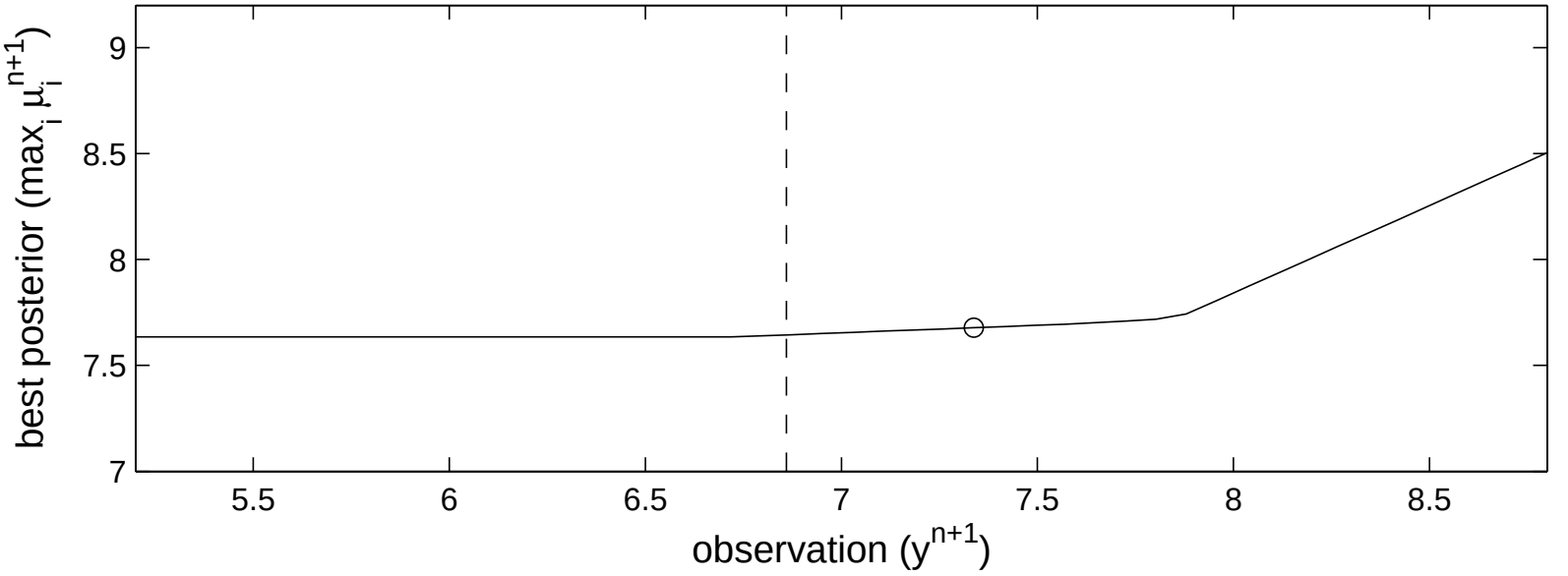
Computing the Knowledge-Gradient



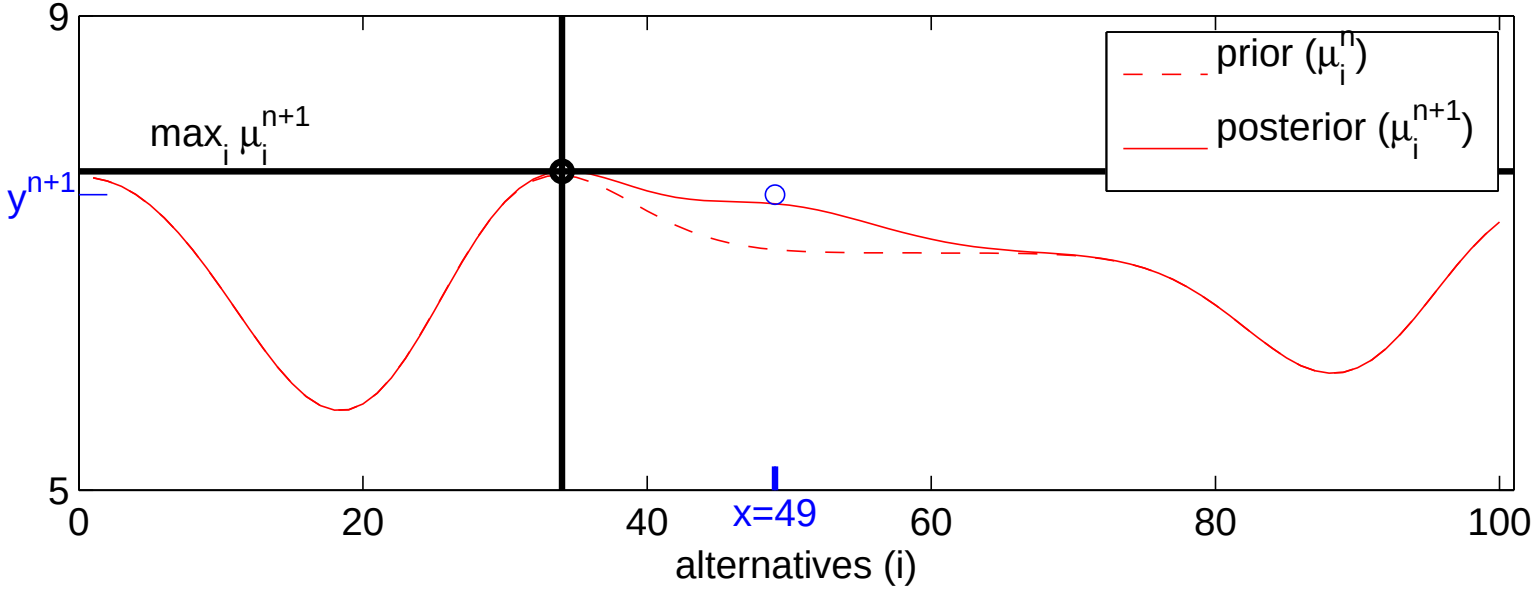


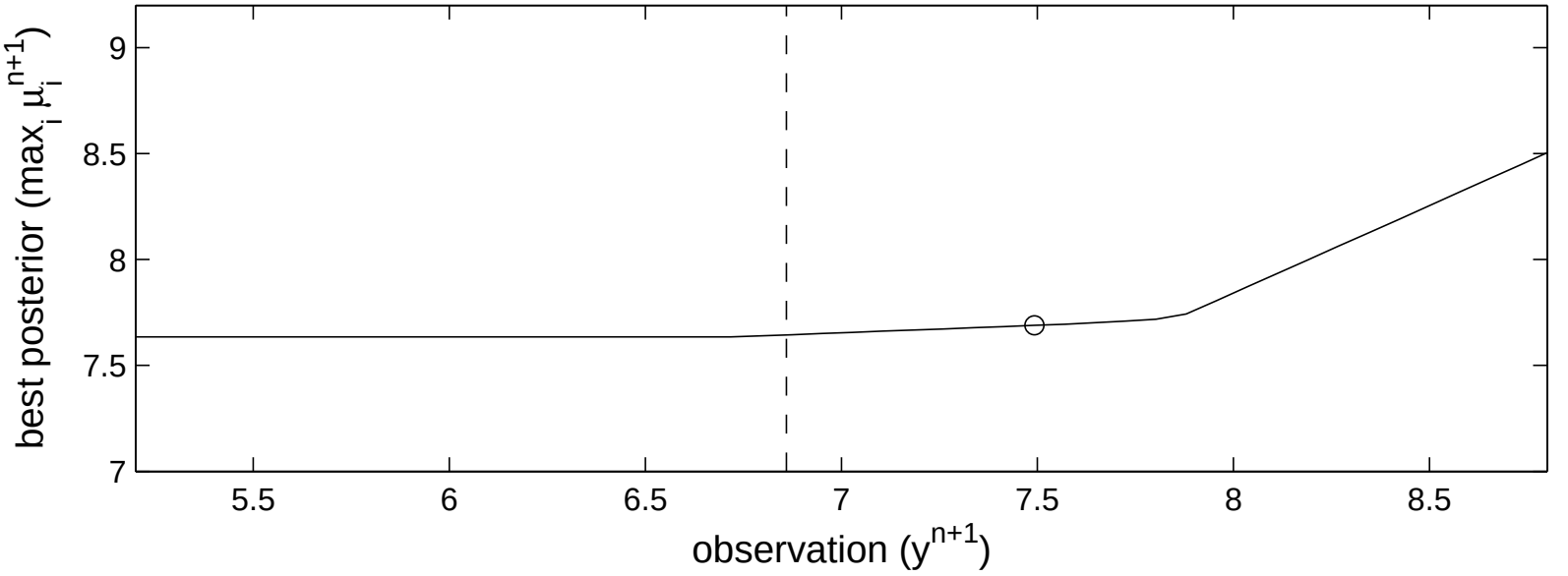
Computing the Knowledge-Gradient



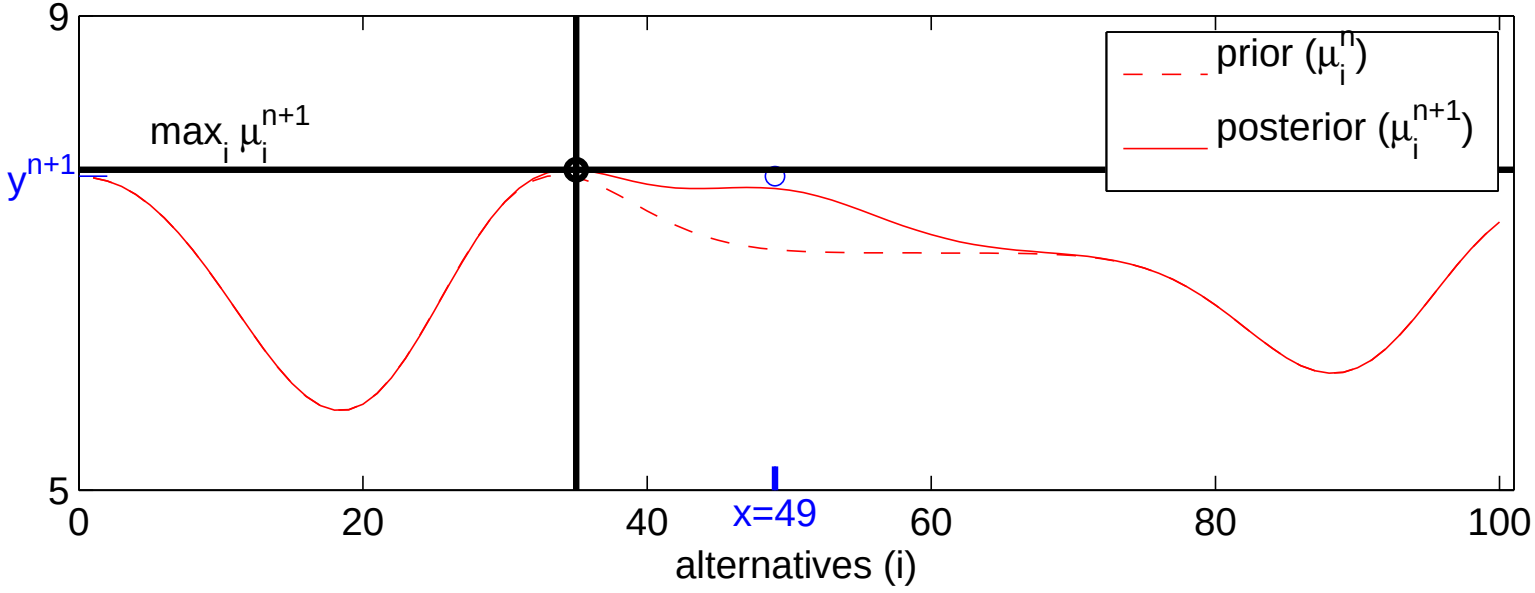


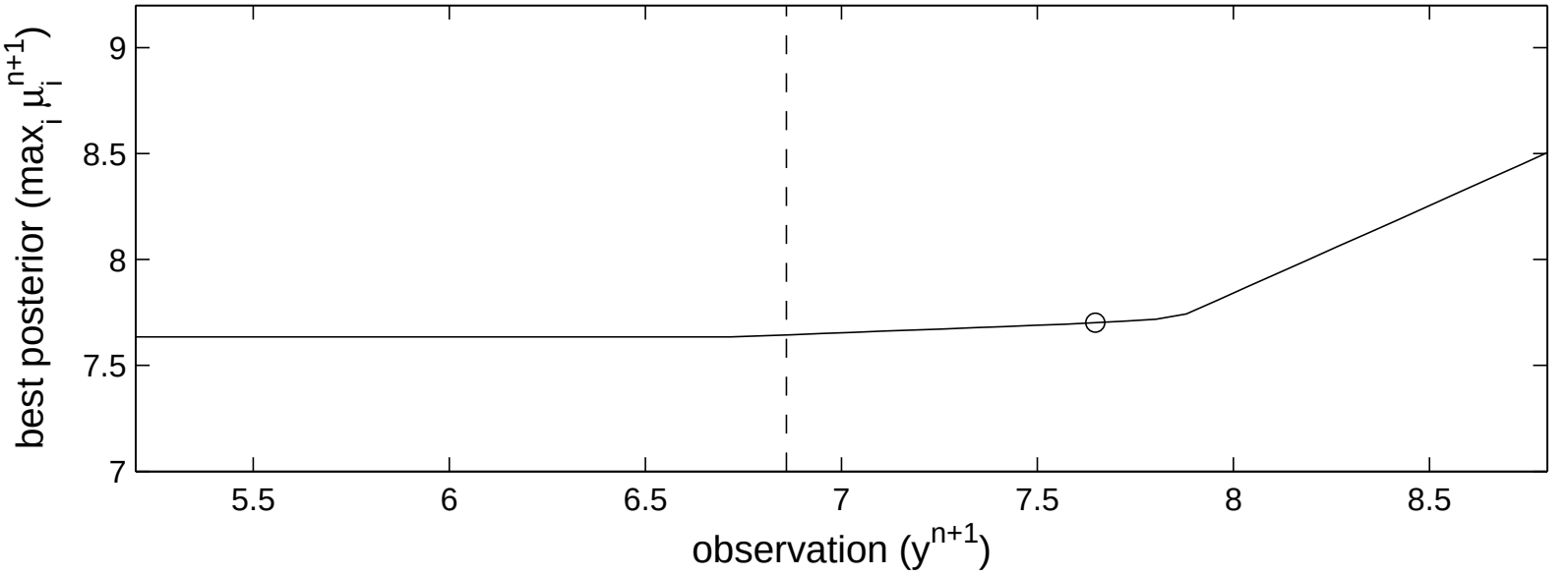
Computing the Knowledge-Gradient



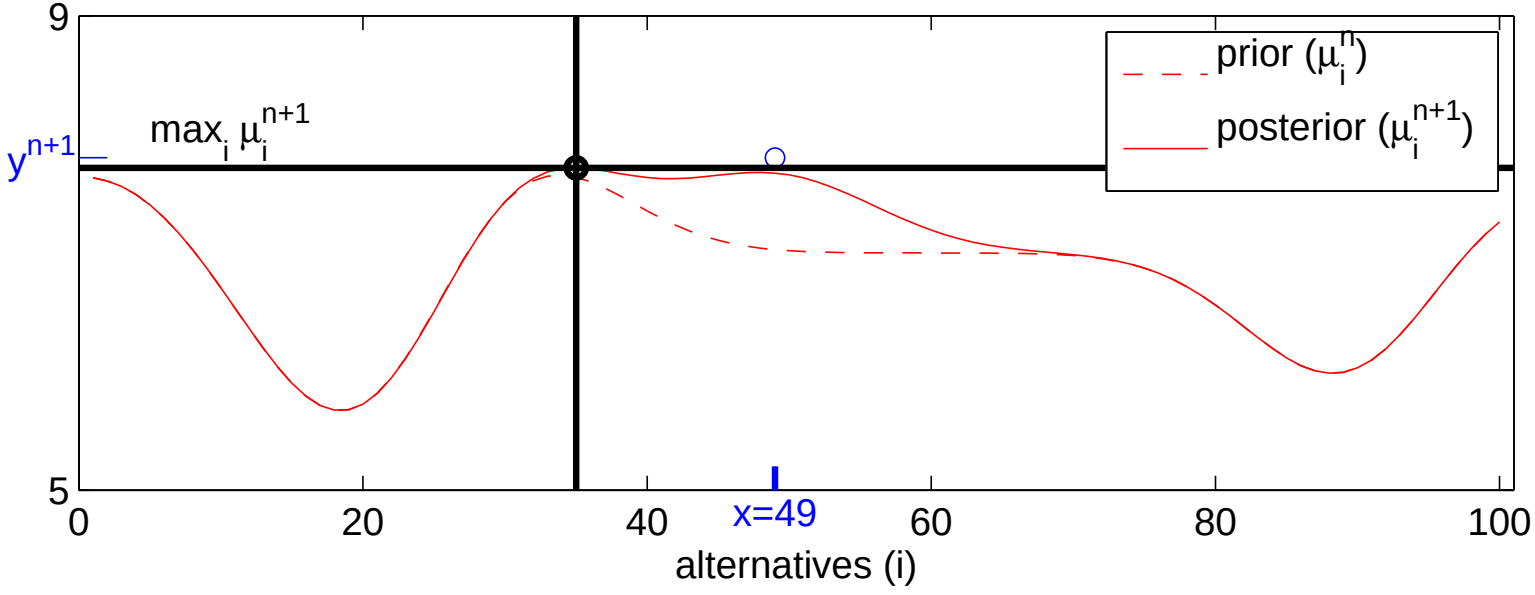


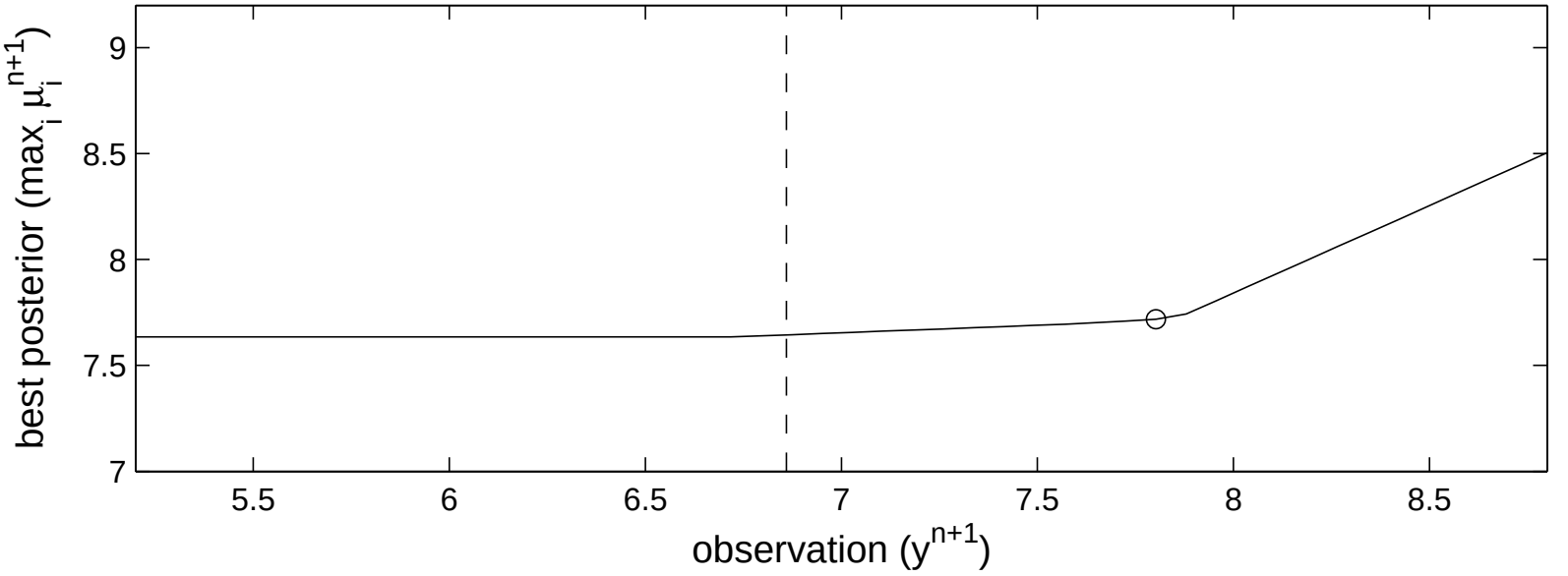
Computing the Knowledge-Gradient



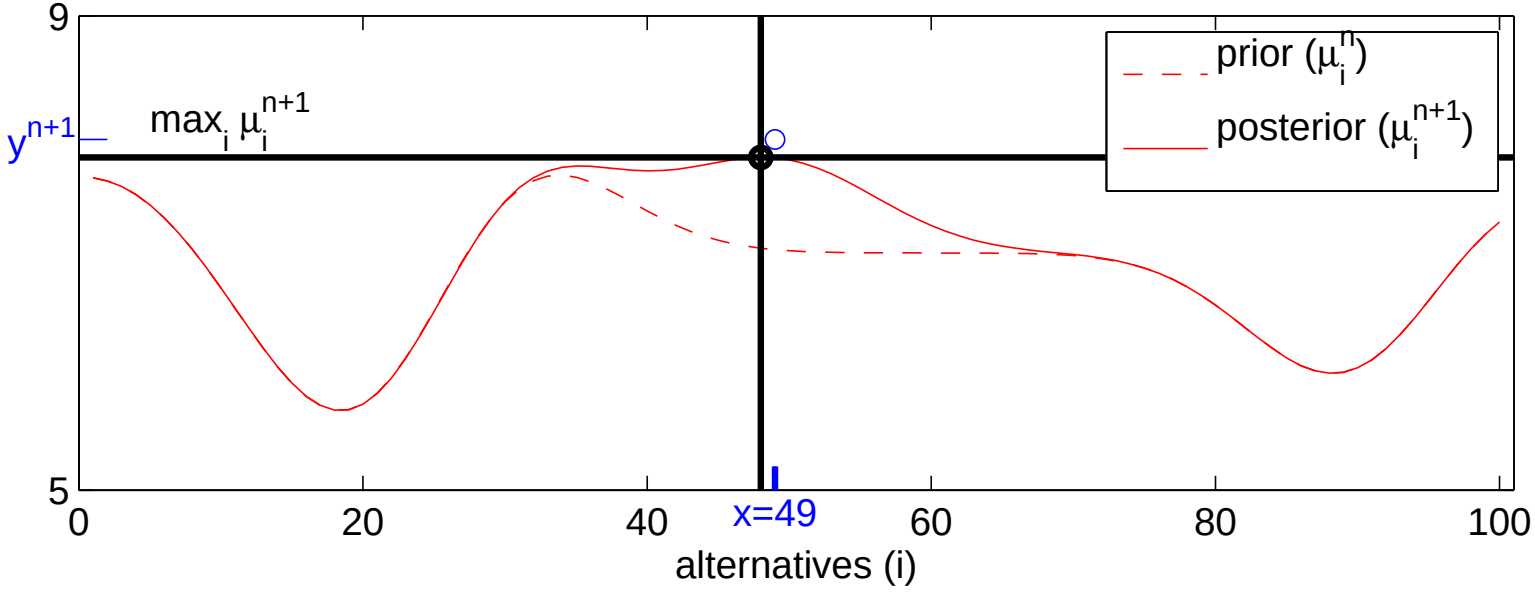


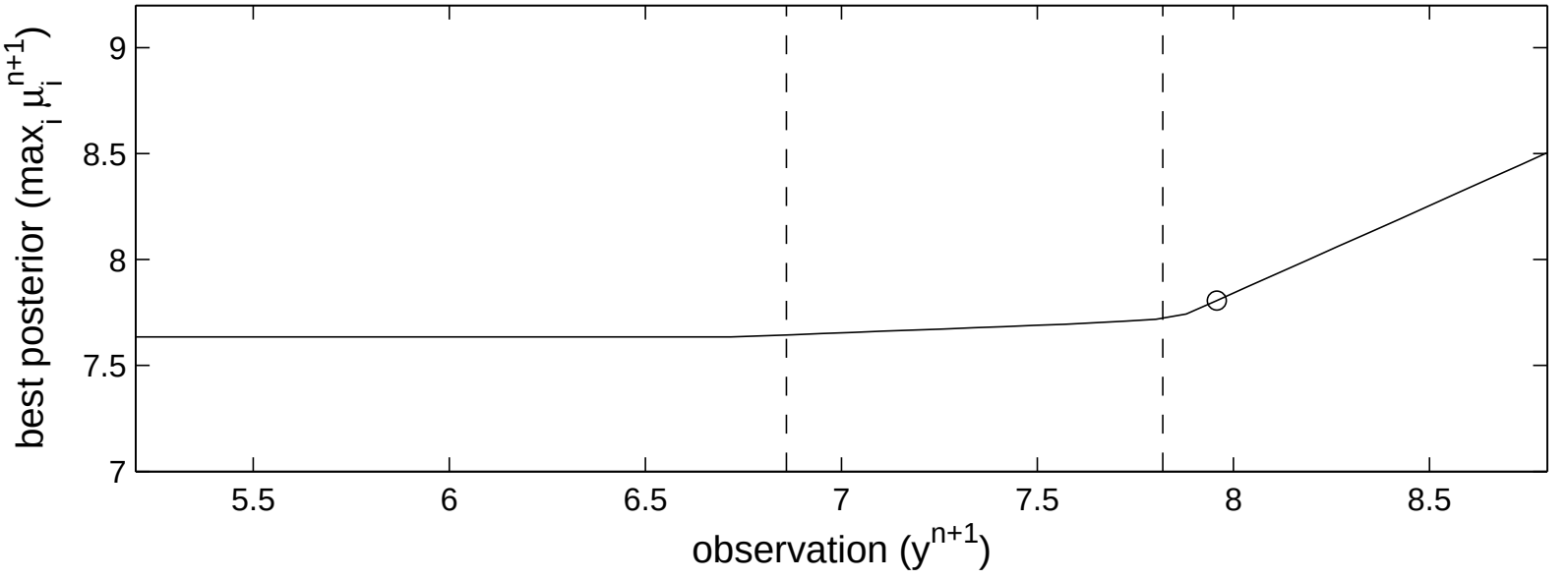
Computing the Knowledge-Gradient



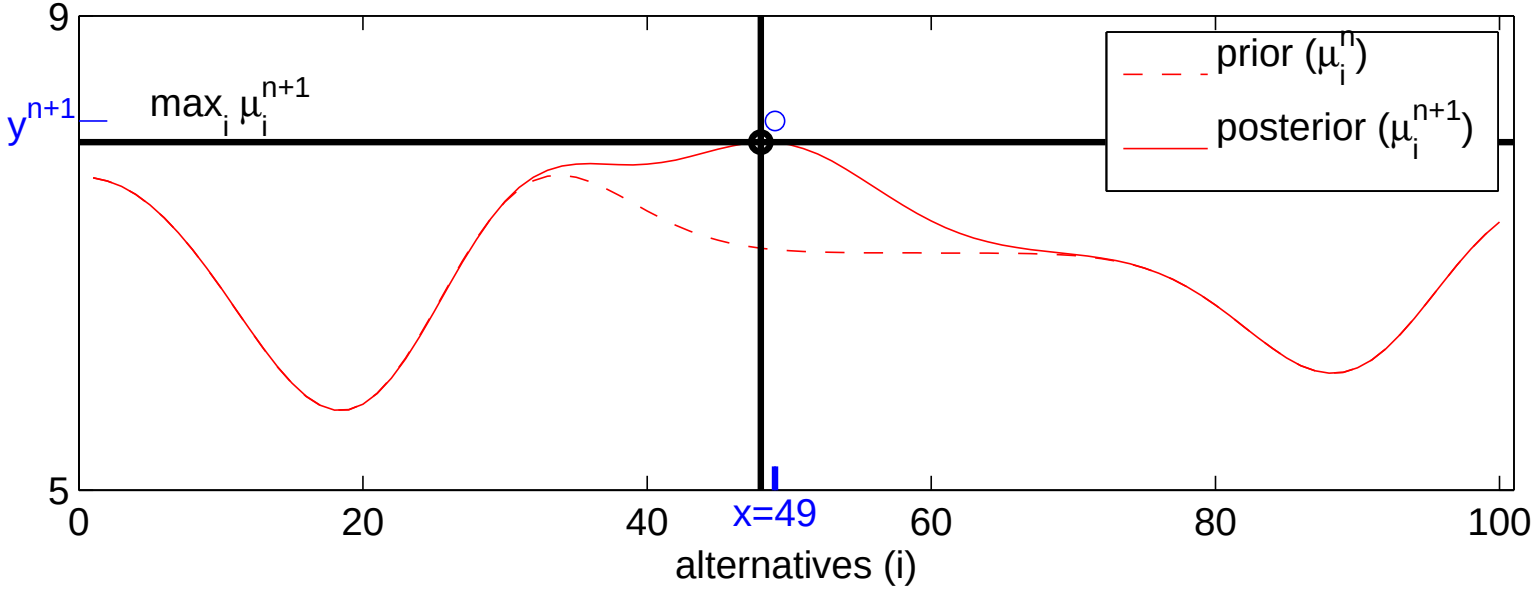


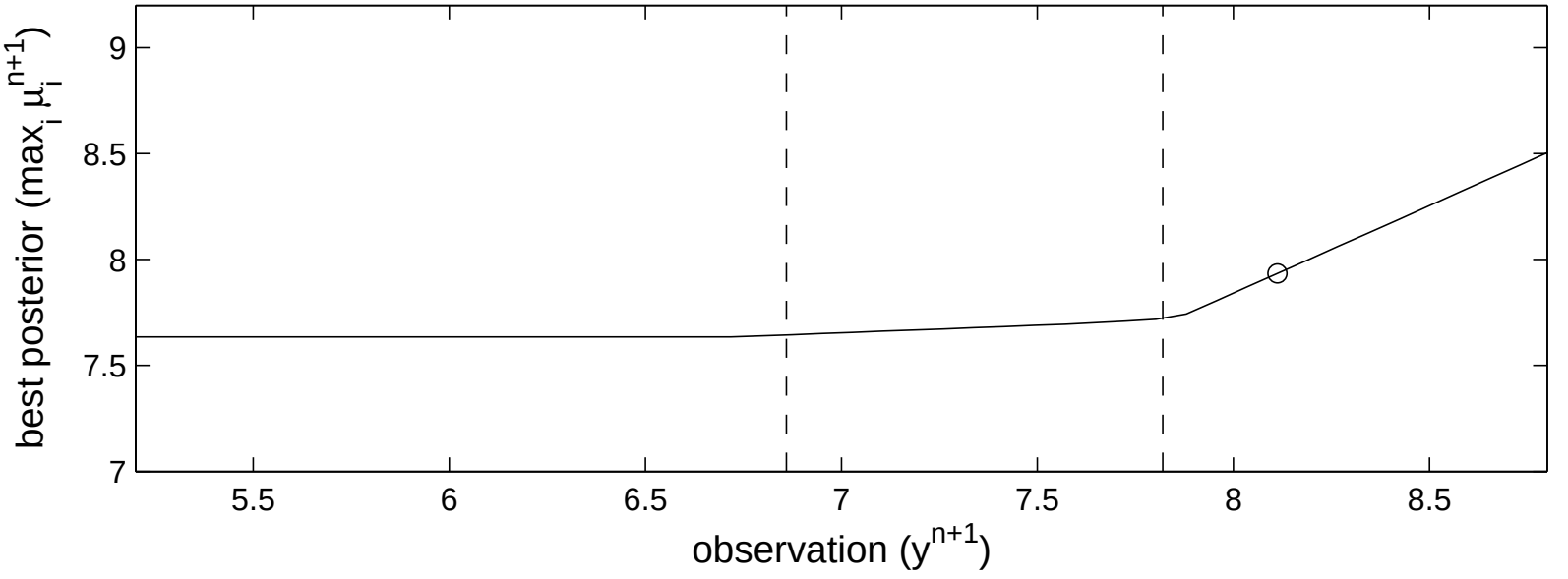
Computing the Knowledge-Gradient



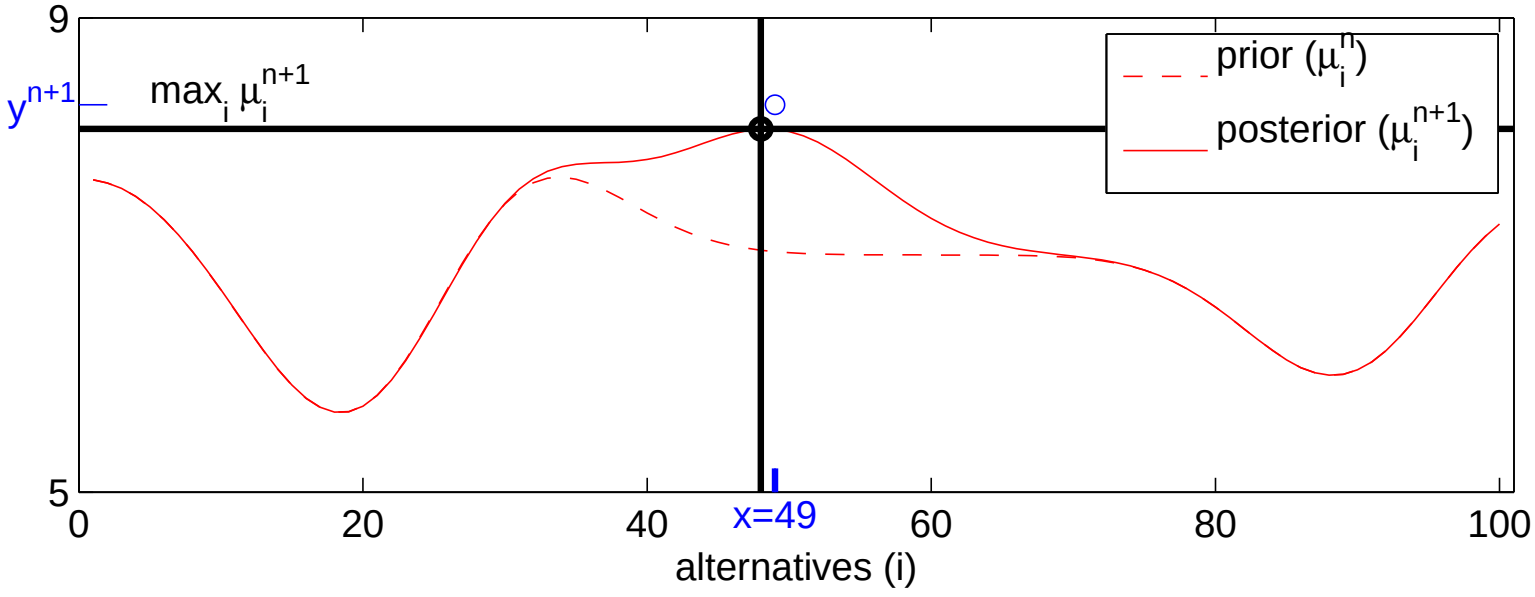


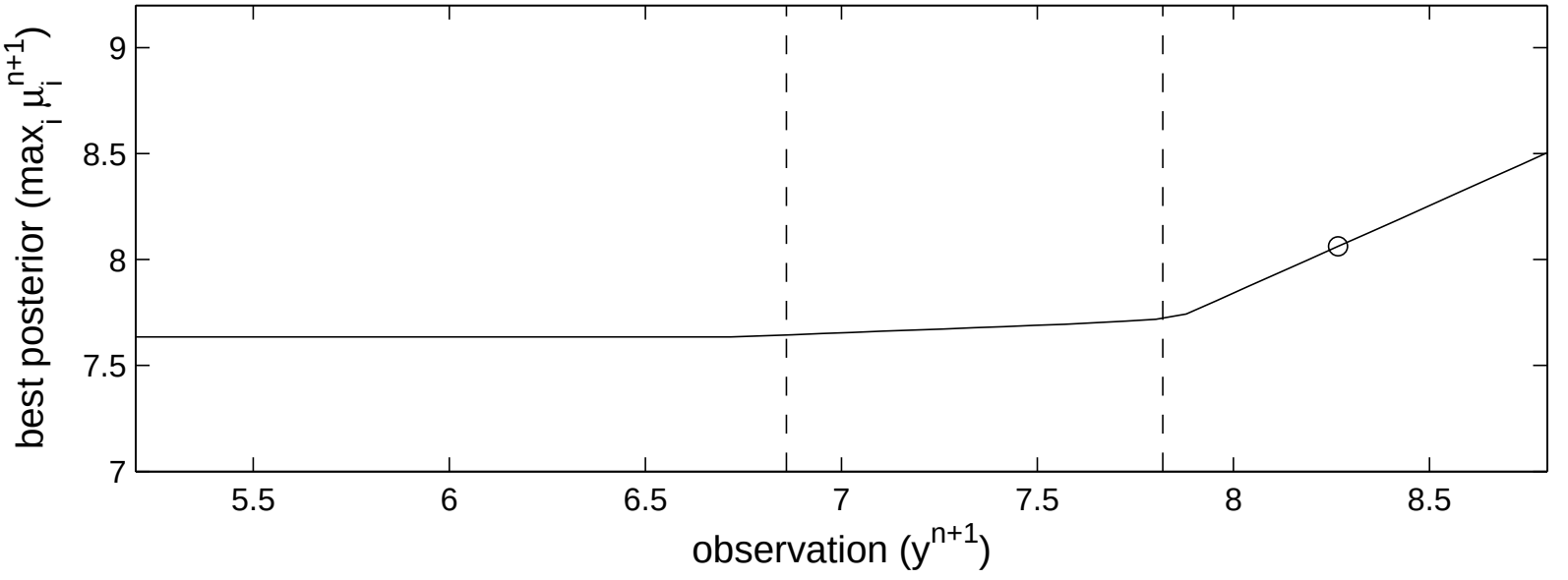
Computing the Knowledge-Gradient



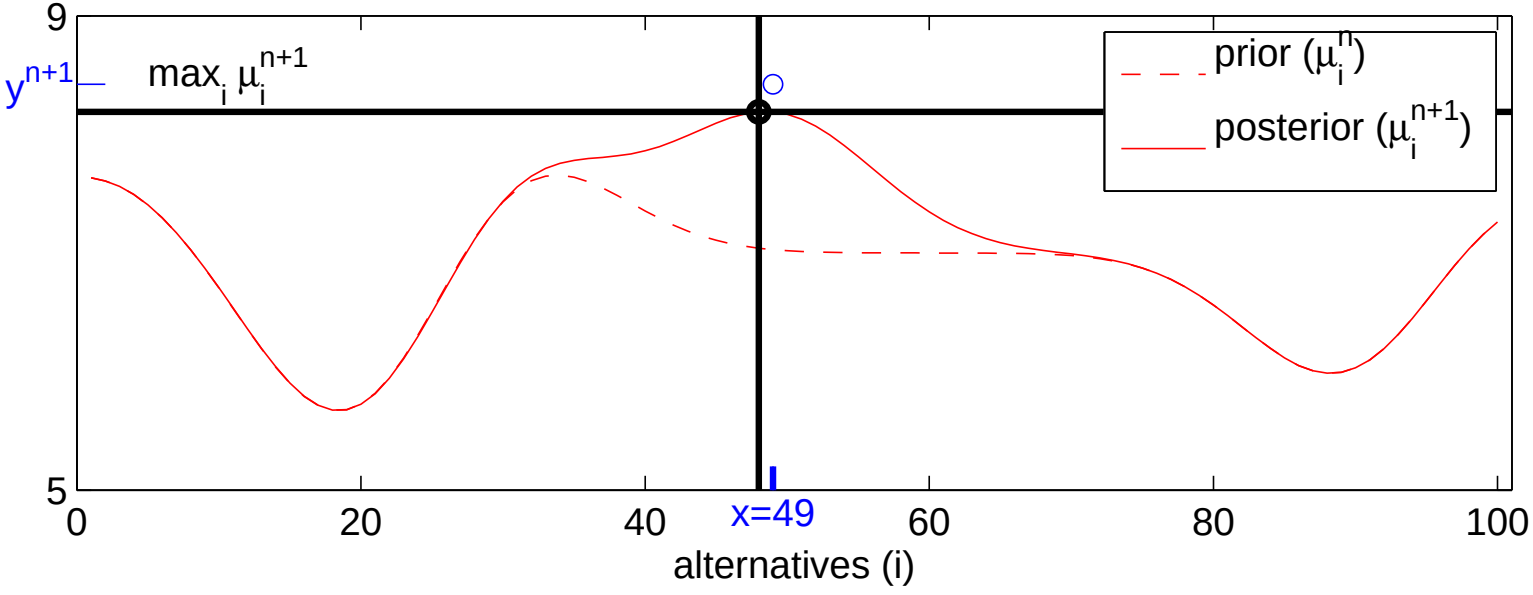


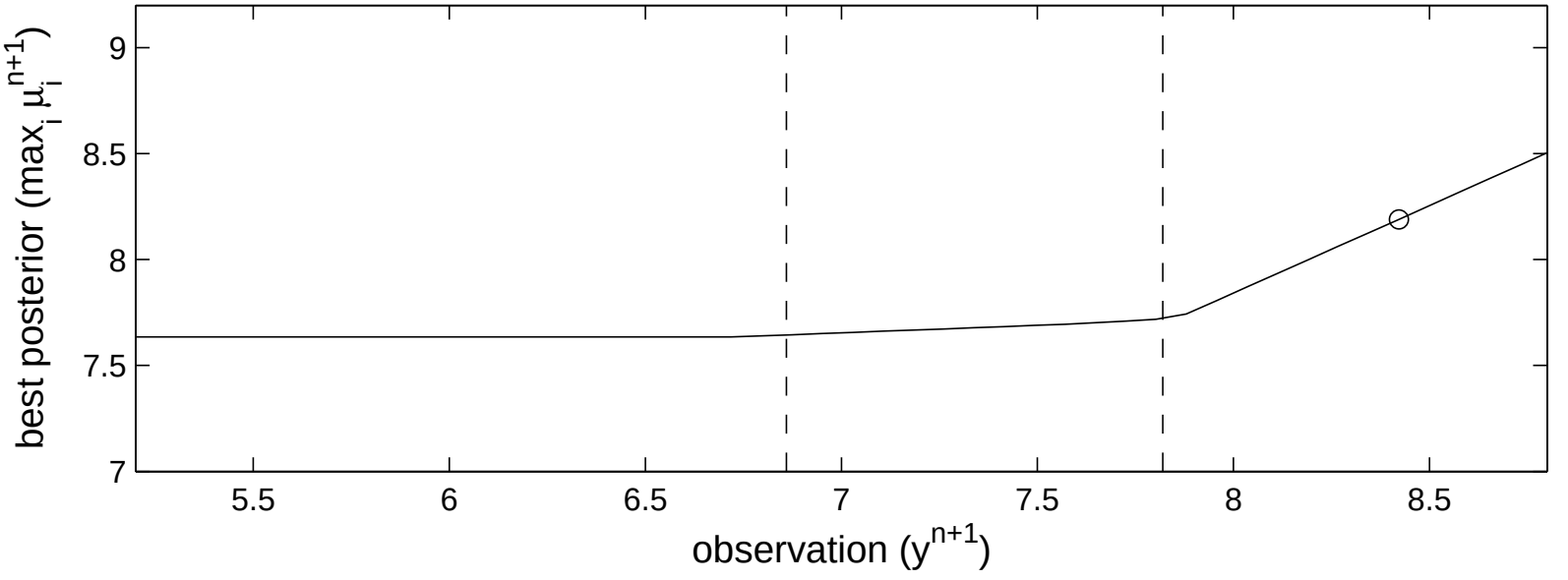
Computing the Knowledge-Gradient



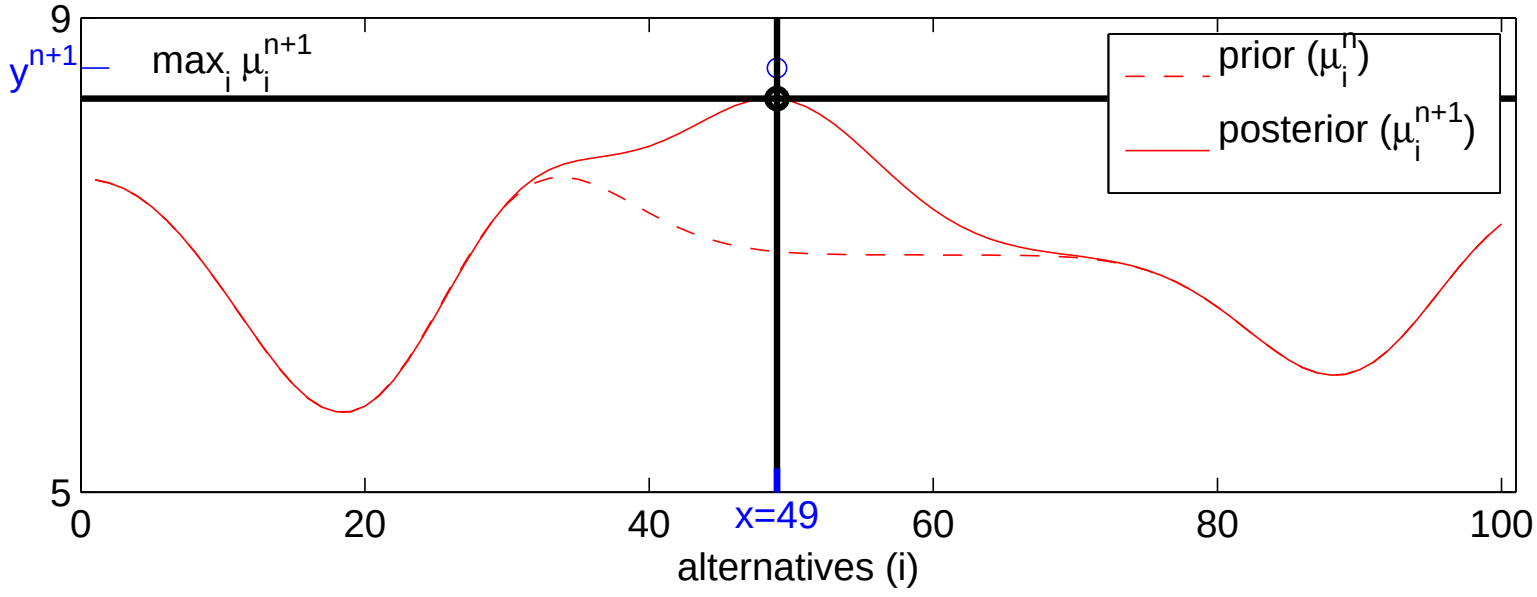


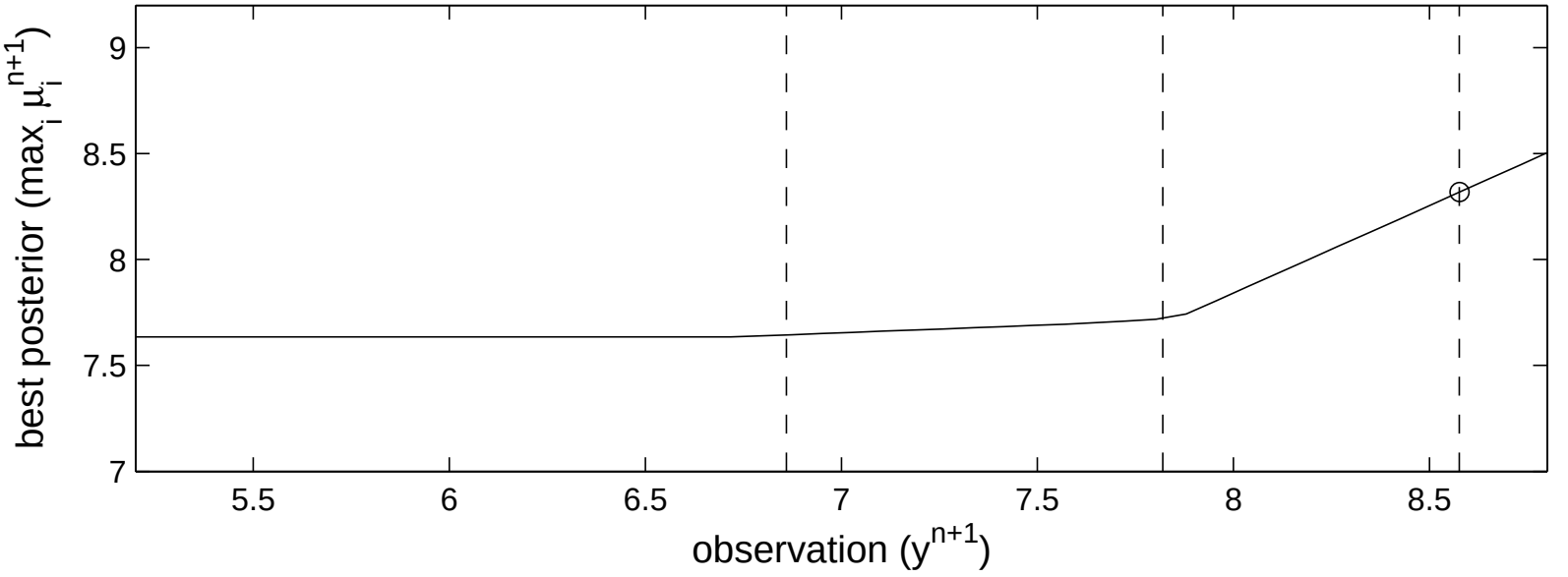
Computing the Knowledge-Gradient





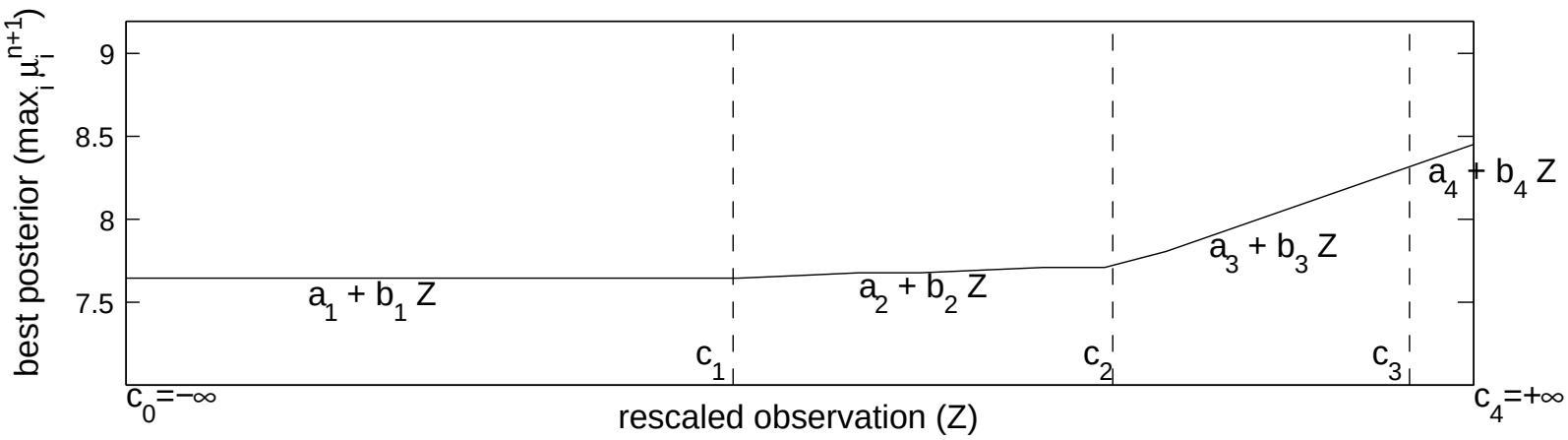
Computing the Knowledge-Gradient





Computing the Knowledge Gradient

Recall that the KG policy chooses the measurement that maximizes the KG factor $\mathbb{E}_n\left[\left(\max_x \mu_x^{n+1}\right)\right] - \max_x \mu_x^n$.
How do we compute this expectation?



We rescale the observation $Z = (y^{n+1} - \mathbb{E}_n[y^{n+1}])/\sqrt{Var_n[y^{n+1}]}$, note that Z is standard normal, and compute the expectation as

$$\mathbb{E}_n\left[\left(\max_x \mu_x^{n+1}\right)\right] = \sum_{j=1}^{4} \mathbb{E}_n\left[(a_j + b_j Z)\mathbf{1}_{\{c_{j-1} \le Z < c_j\}}\right].$$

Computing the Knowledge Gradient

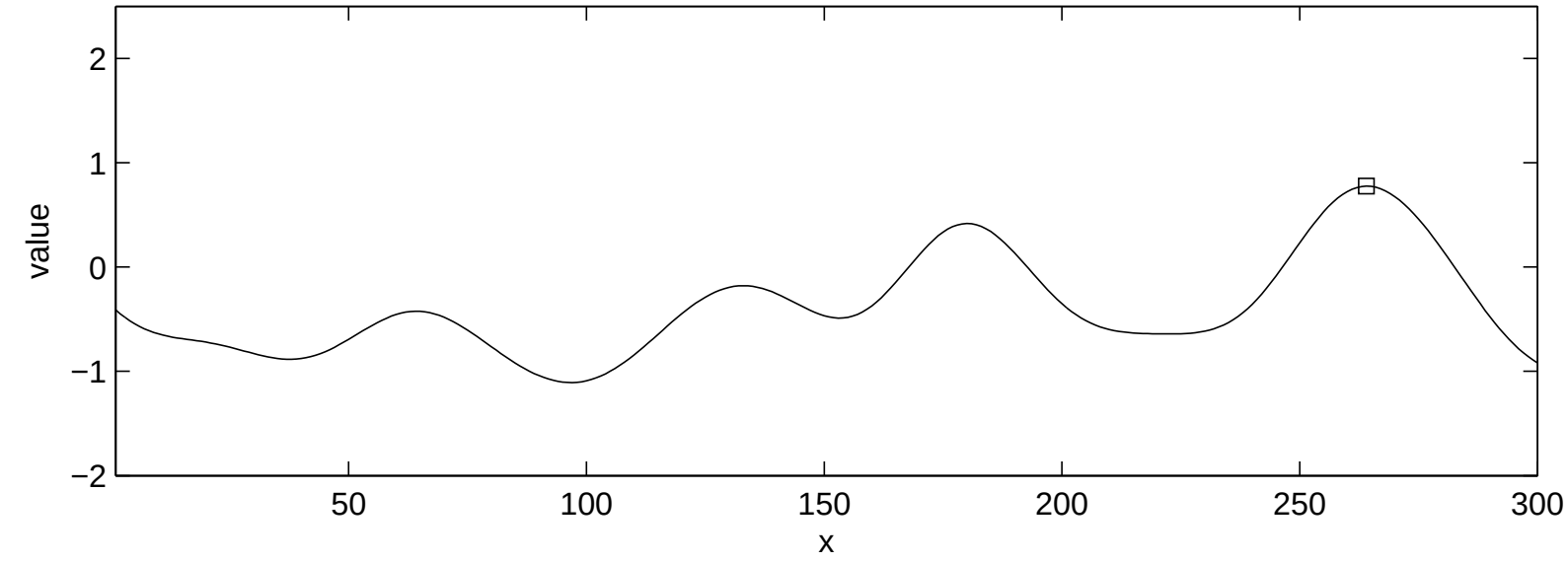
In general, to compute the KG factor for a candidate measurement:

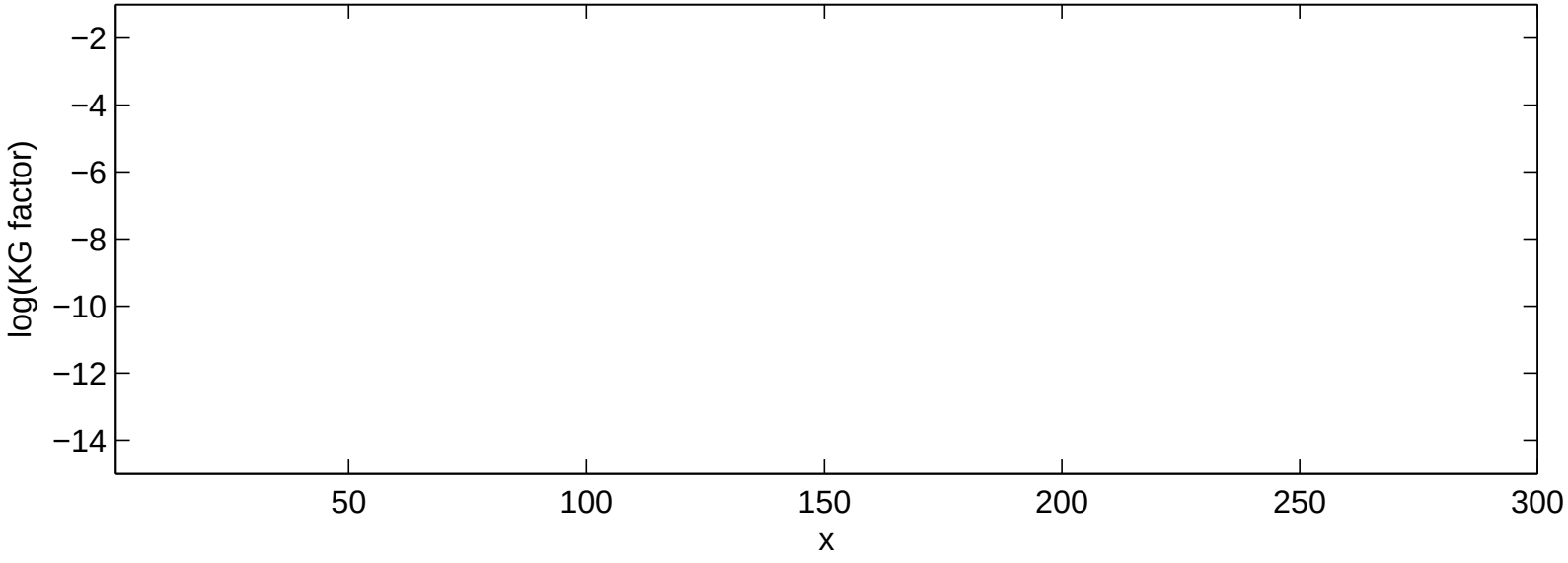
- Let A contain those alternatives that are best under the posterior with nonzero probability.
- Let $[j]$ denote the j^{th} entry in A.
- Let $a_j = \mu^n_{[j]}$ and $b_j = \sqrt{\text{Var}_n\left[\mu^{n+1}_{[j]}\right]}$.
- Sort A in order of increasing b_j.
- The KG factor is

$$\sum_{j=1}^{|A|-1} (b_{i+1} - b_i) f\left(\frac{-|a_{i+1} - a_i|}{b_{i+1} - b_i}\right),$$

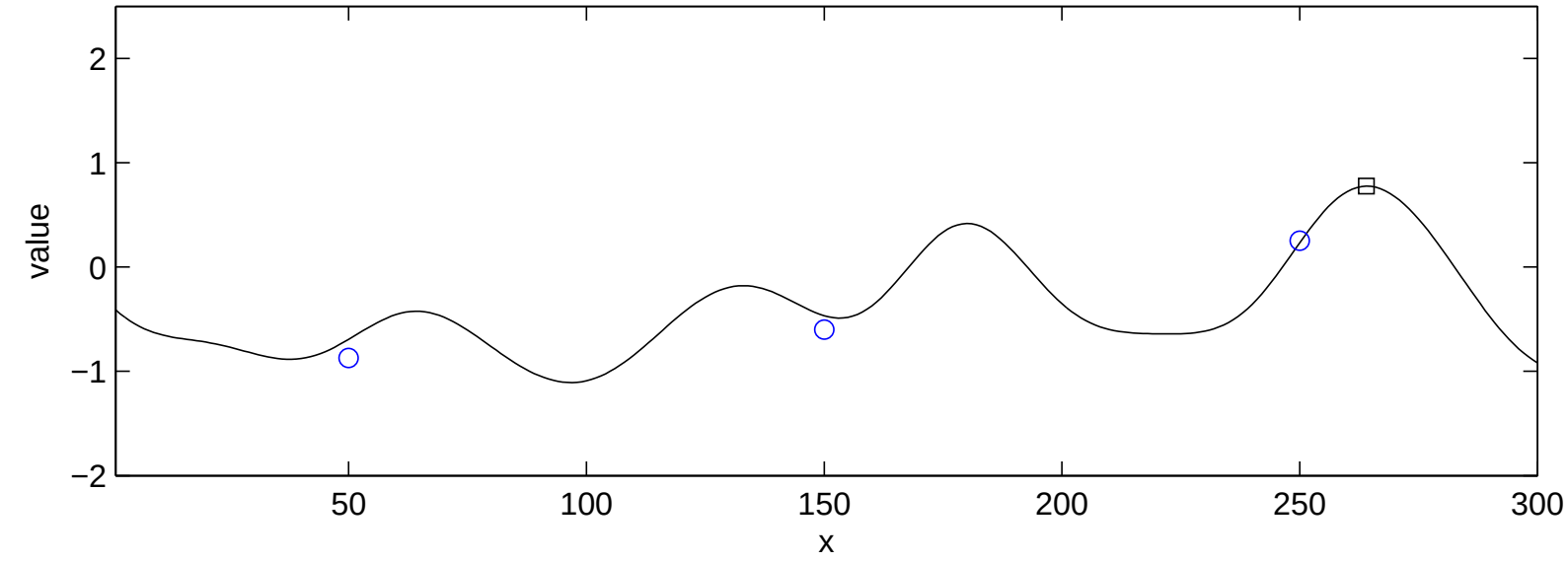
where $f(z) = \varphi(z) + z\Phi(z)$, φ is the normal pdf and Φ is the normal cdf.

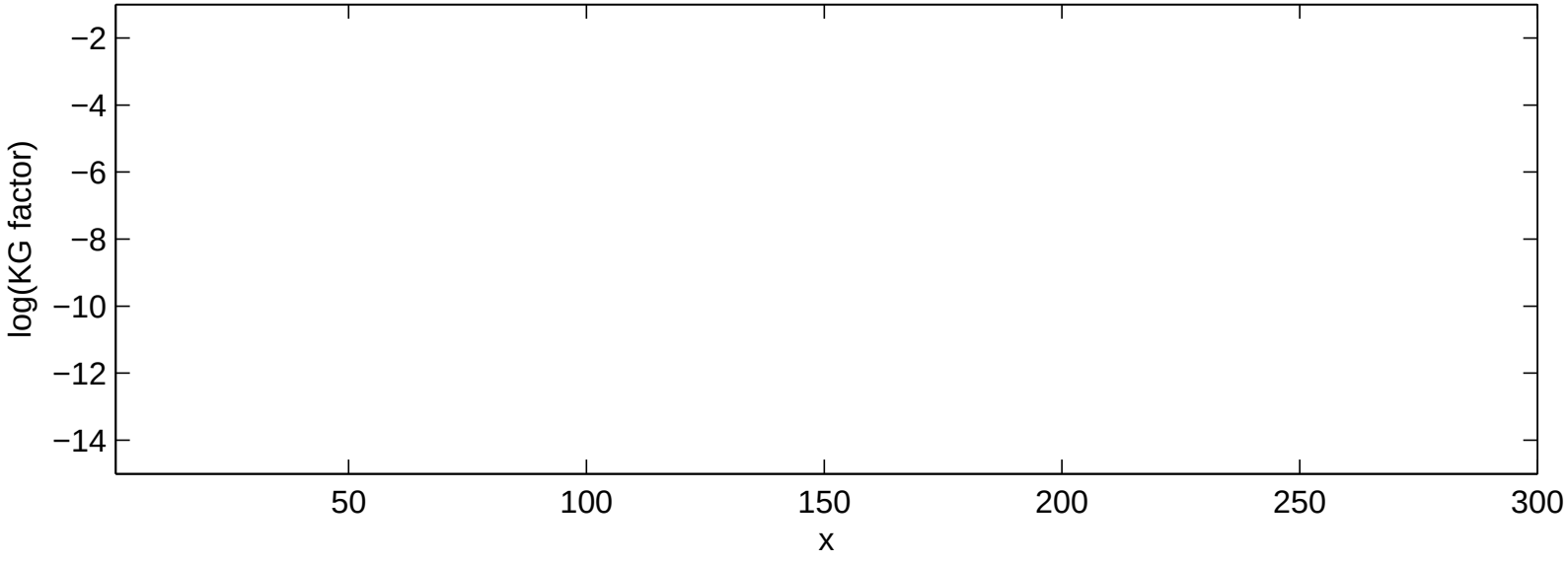
KG Example


value
2
1
0
−1
−2
50
100
150
200
250
300
x

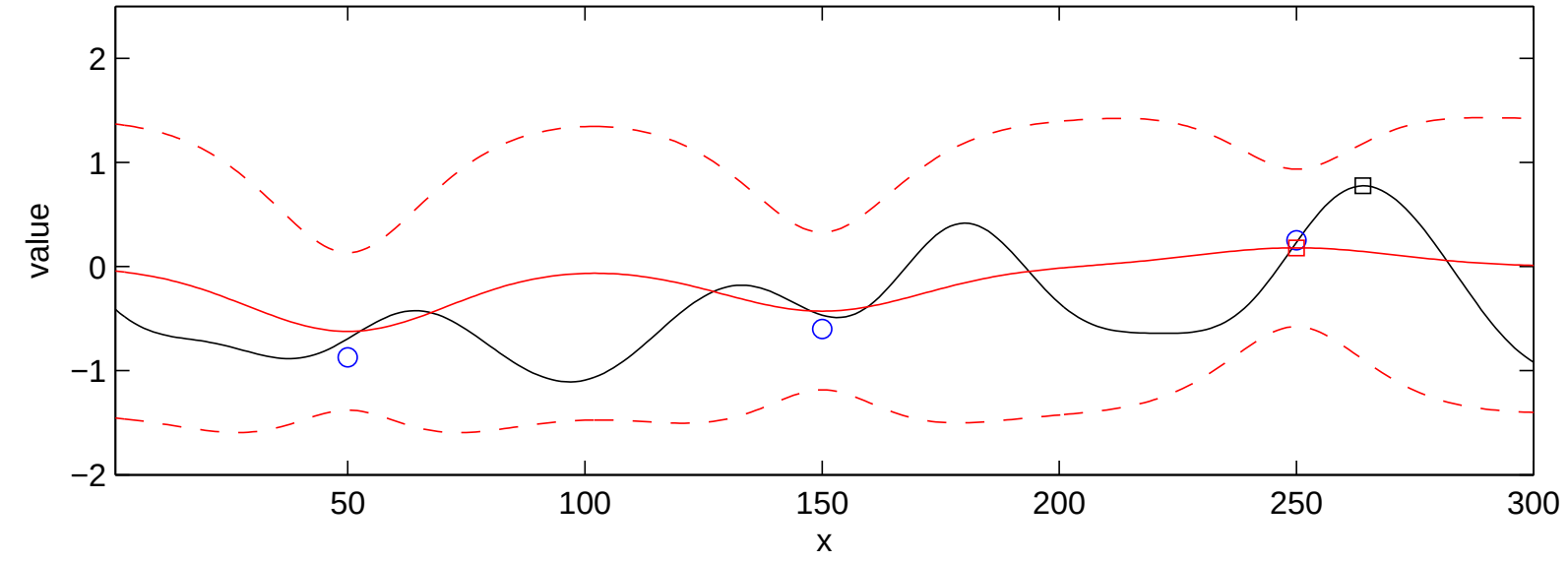

log(KG factor)
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300
x

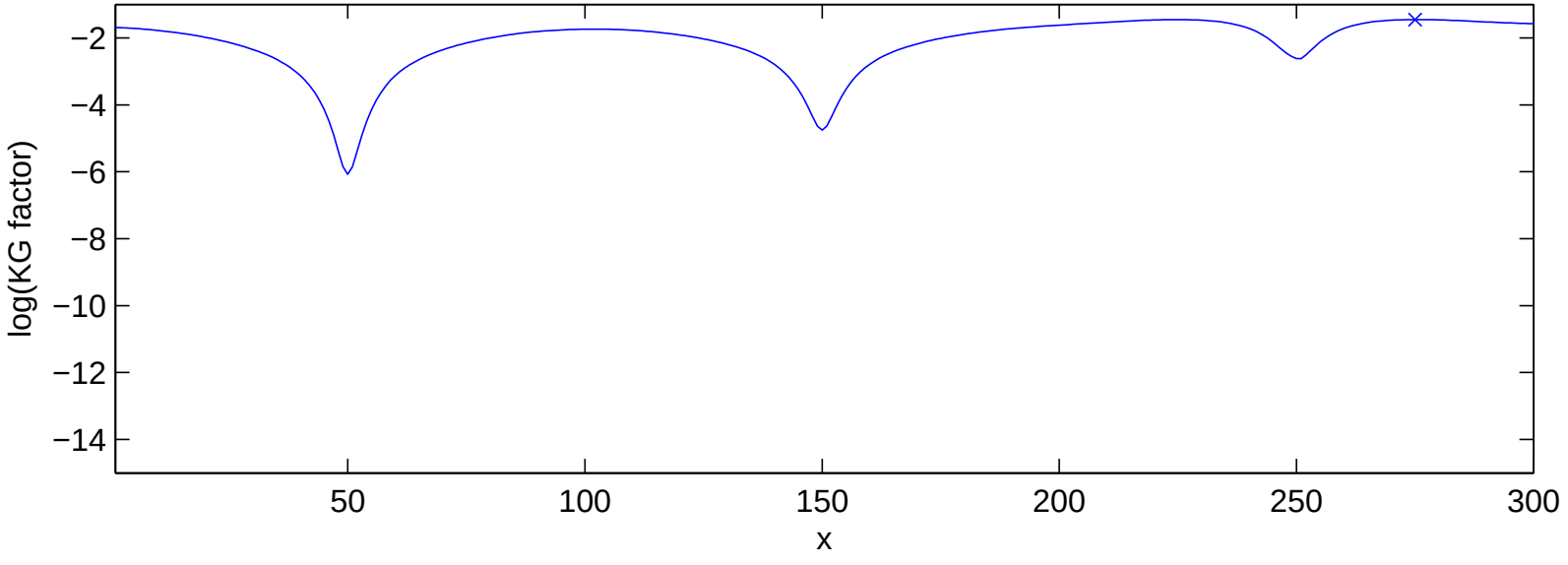
KG Example


value
2
1
0
−1
−2
50
100
150
200
250
300
x

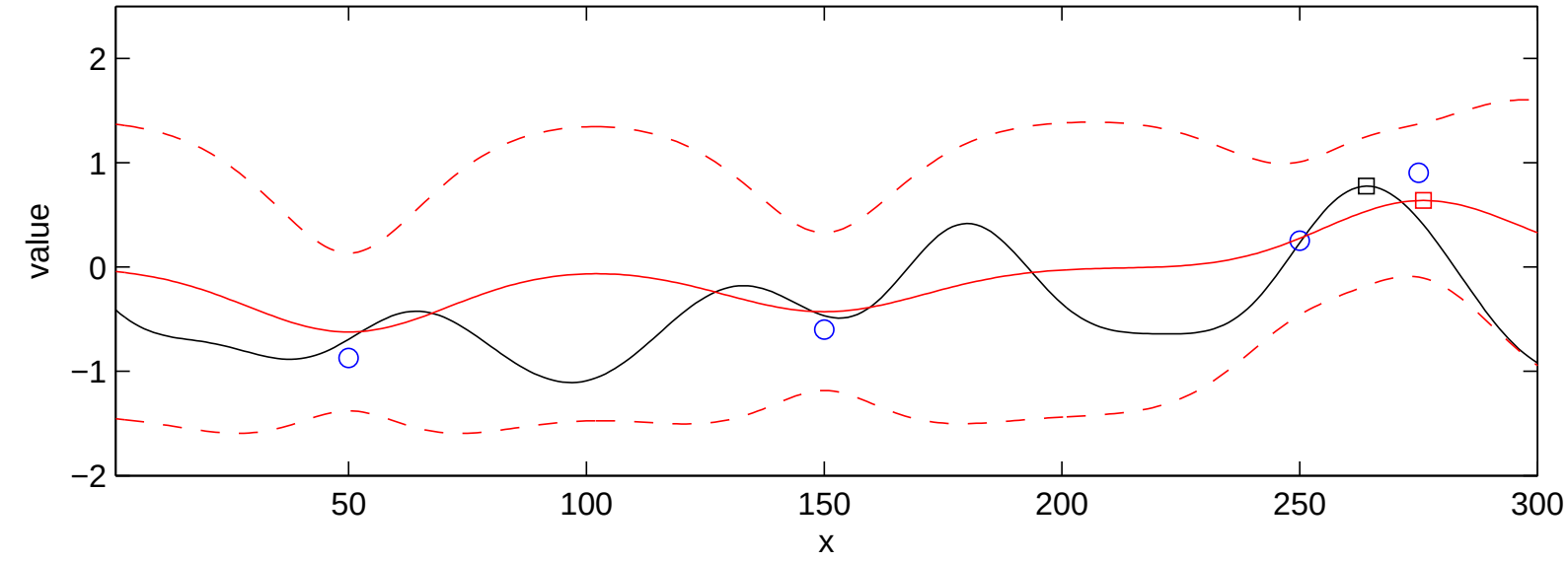

log(KG factor)
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300
x

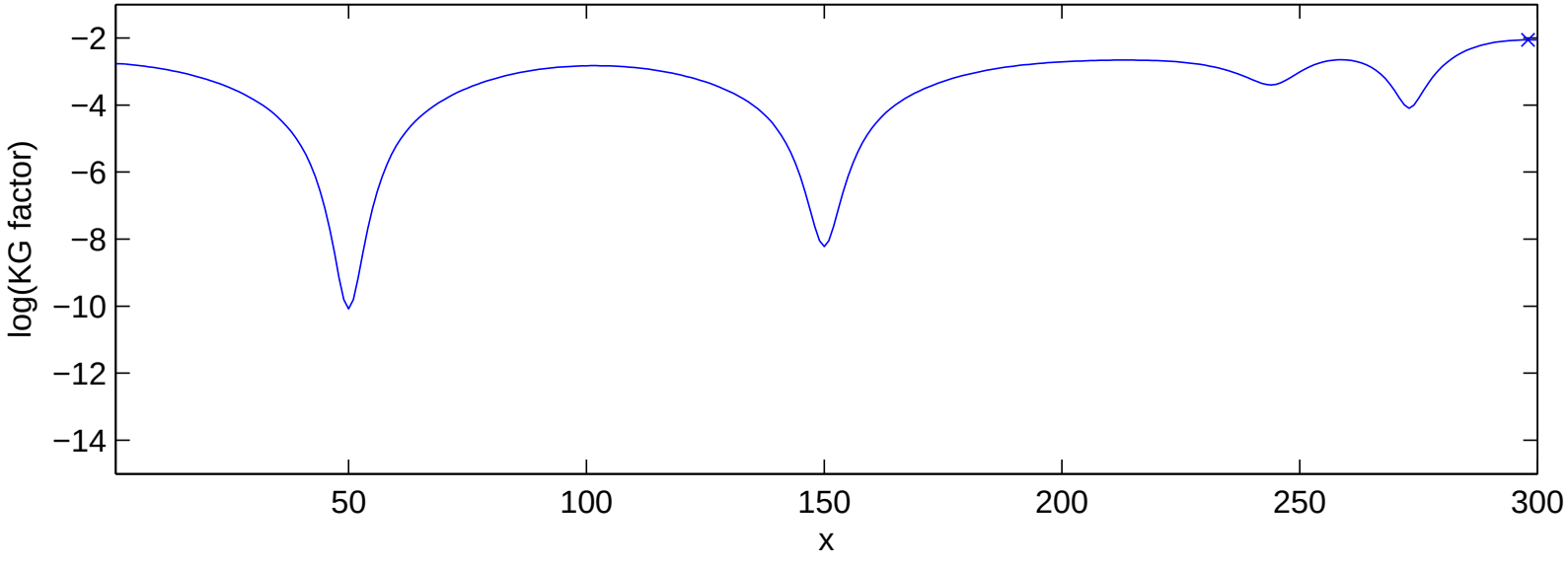
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300

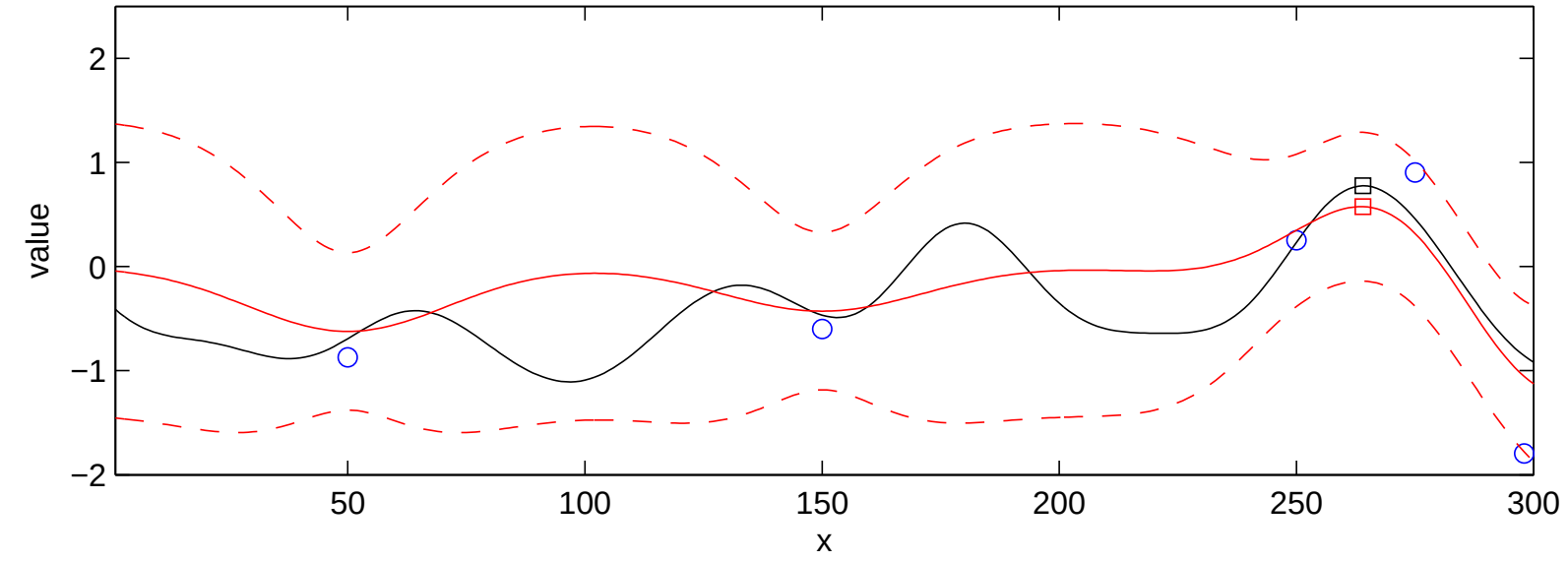

log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

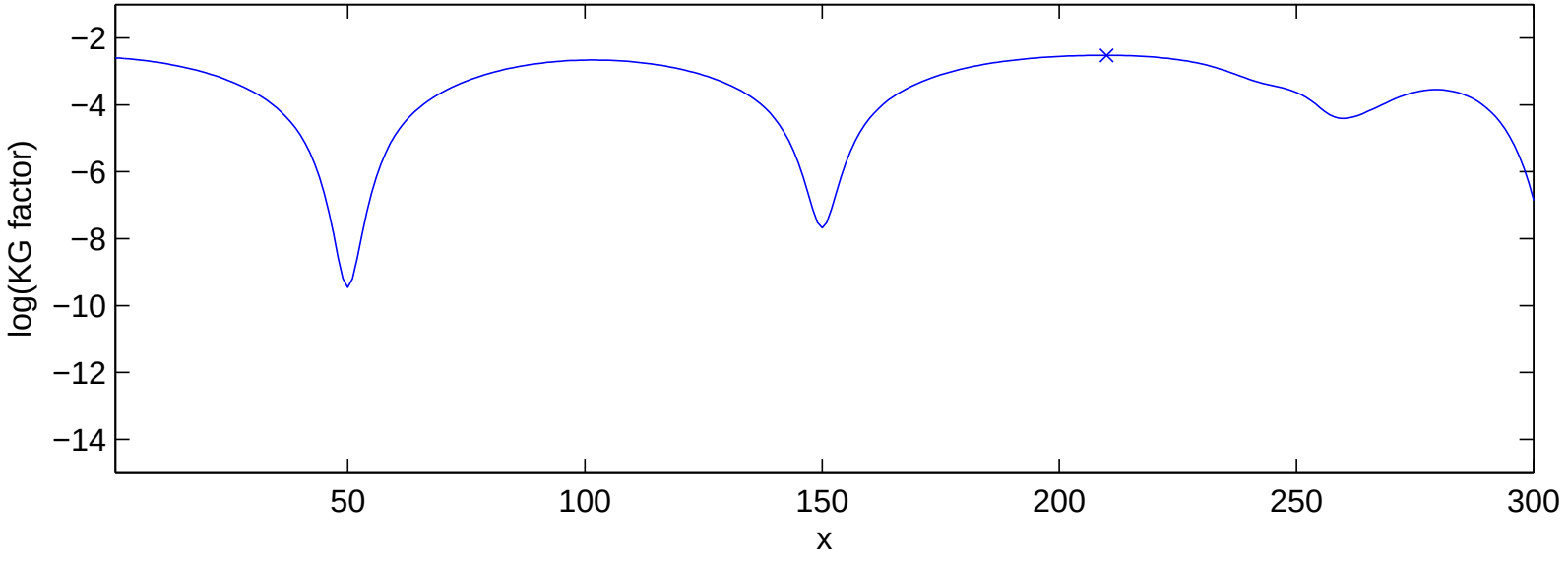
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300

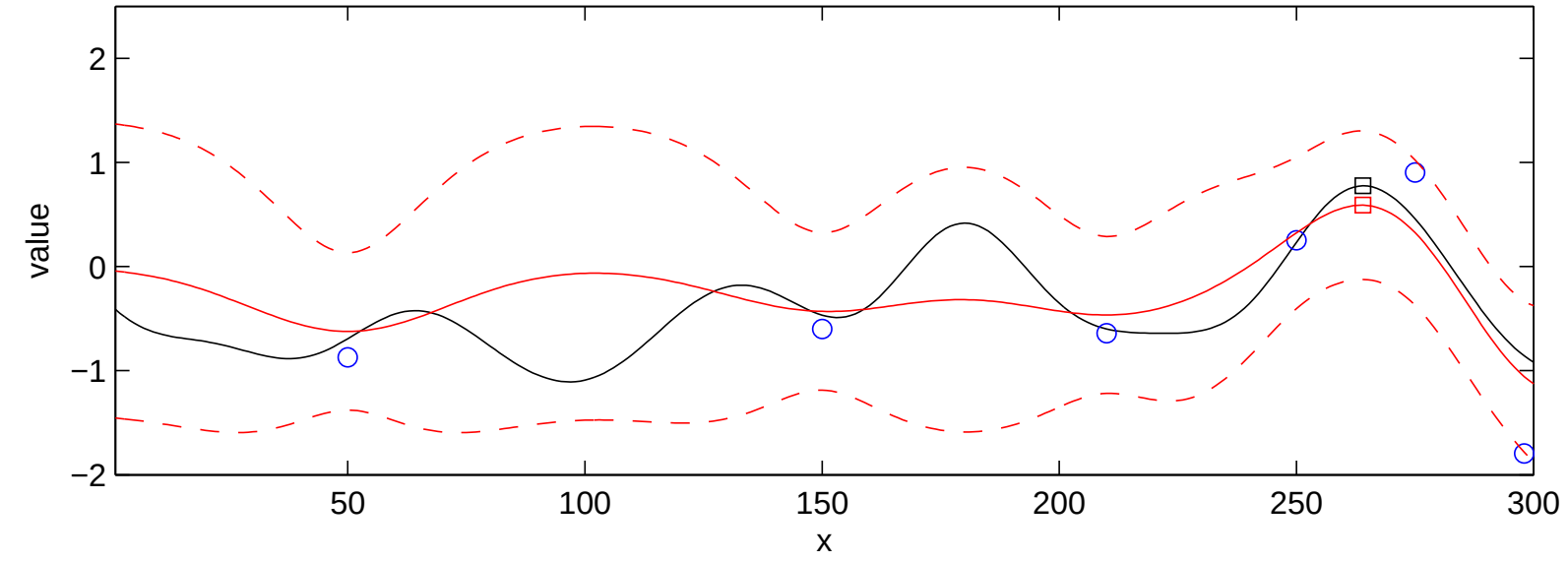

log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

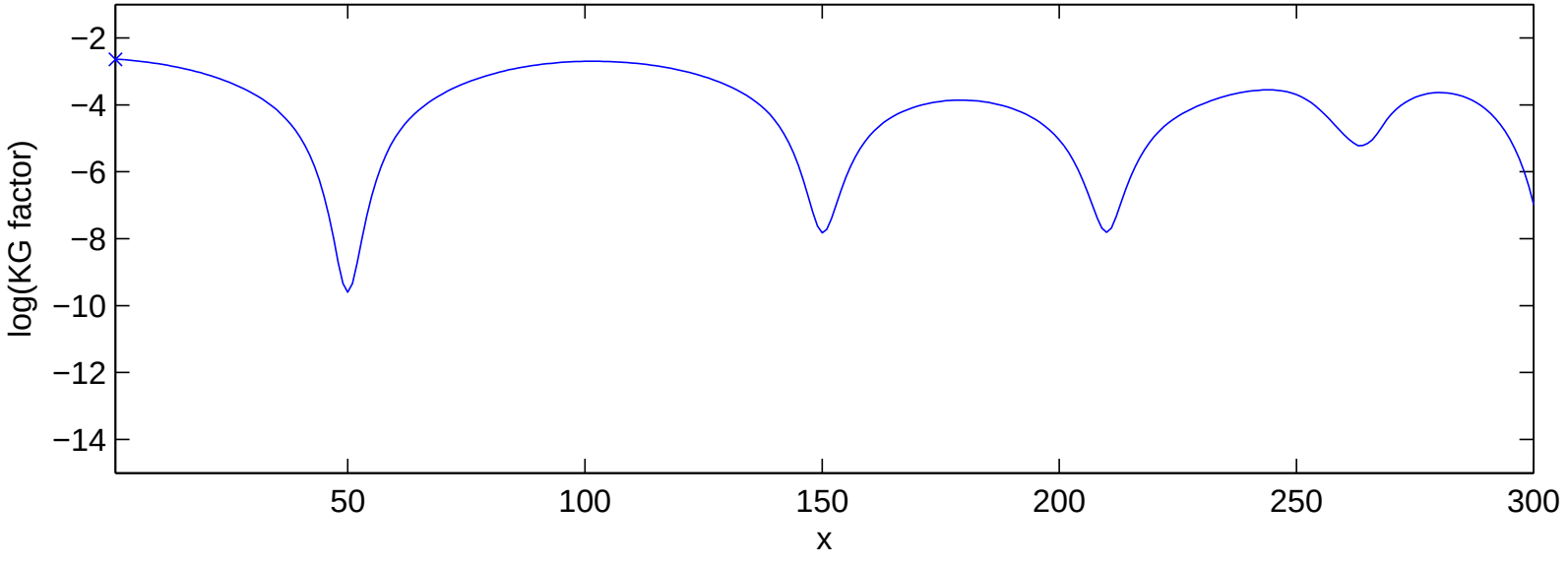
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300

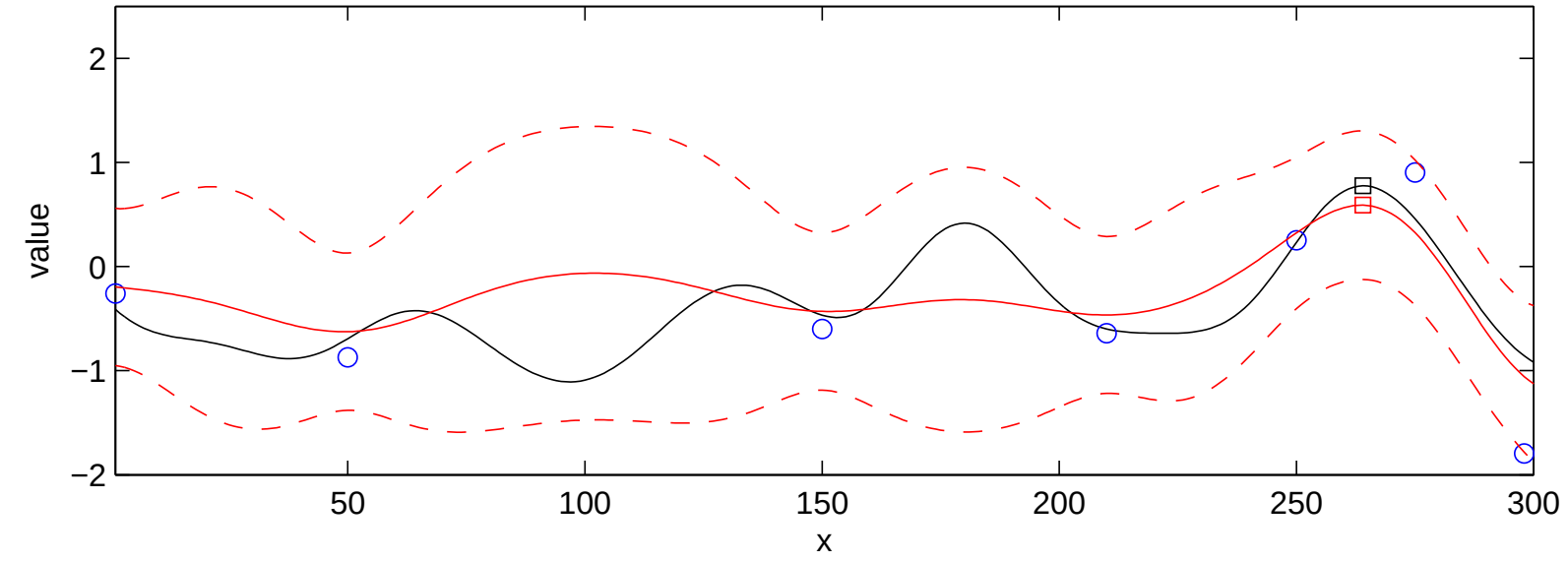

log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

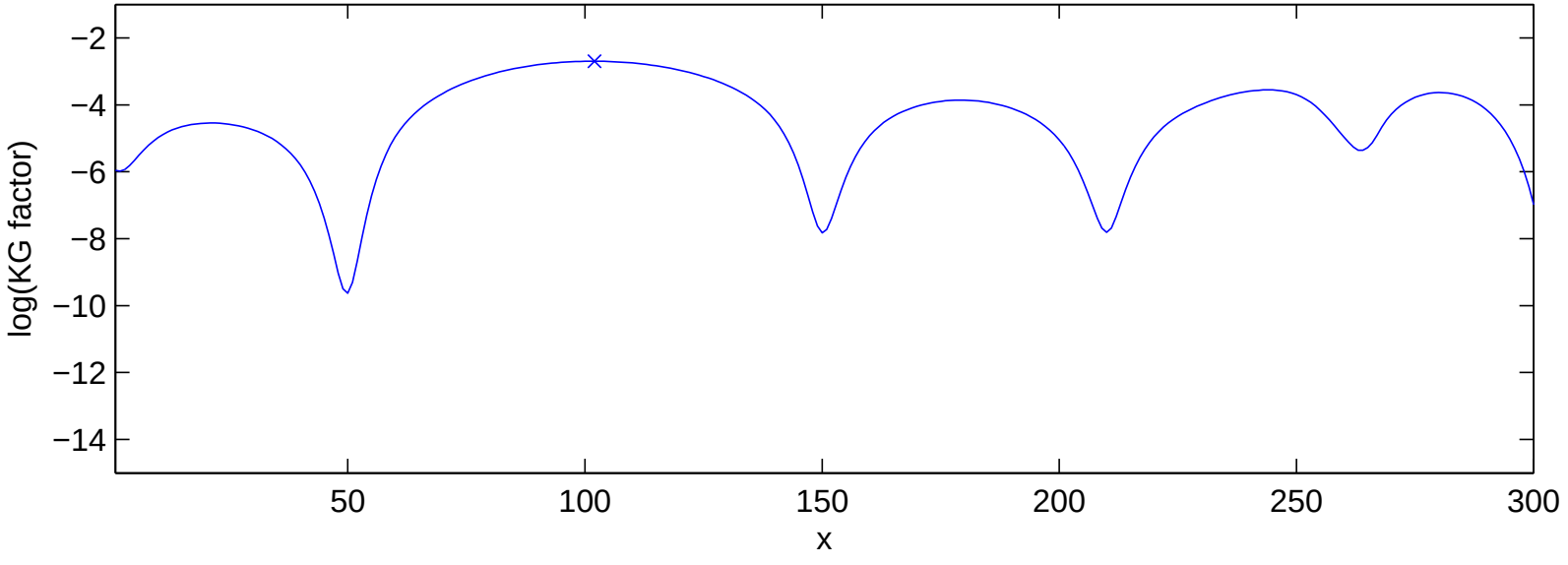
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300

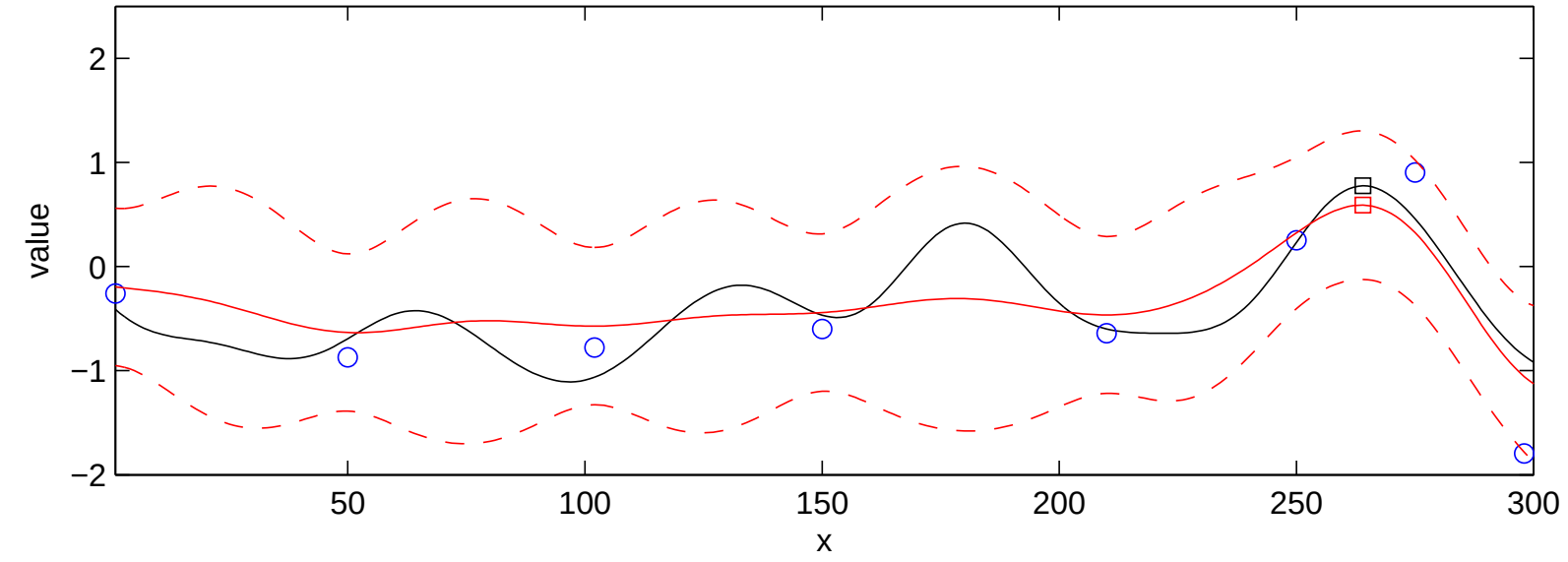

log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

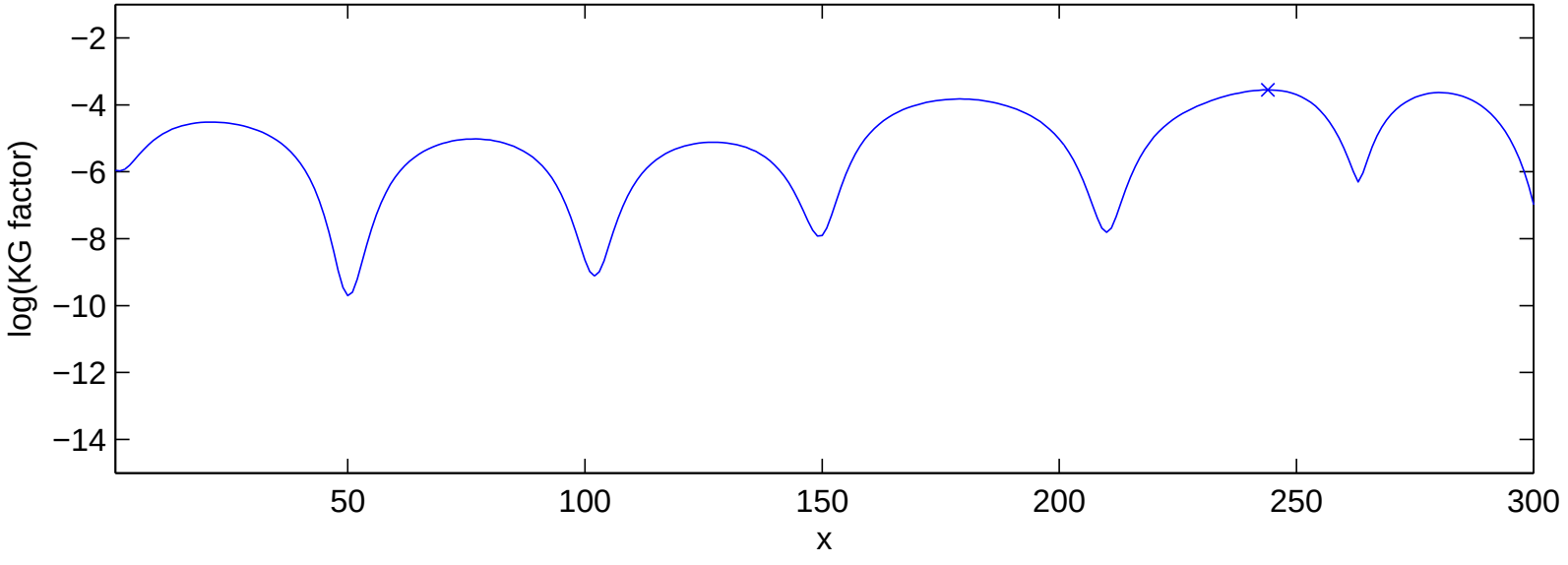
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300

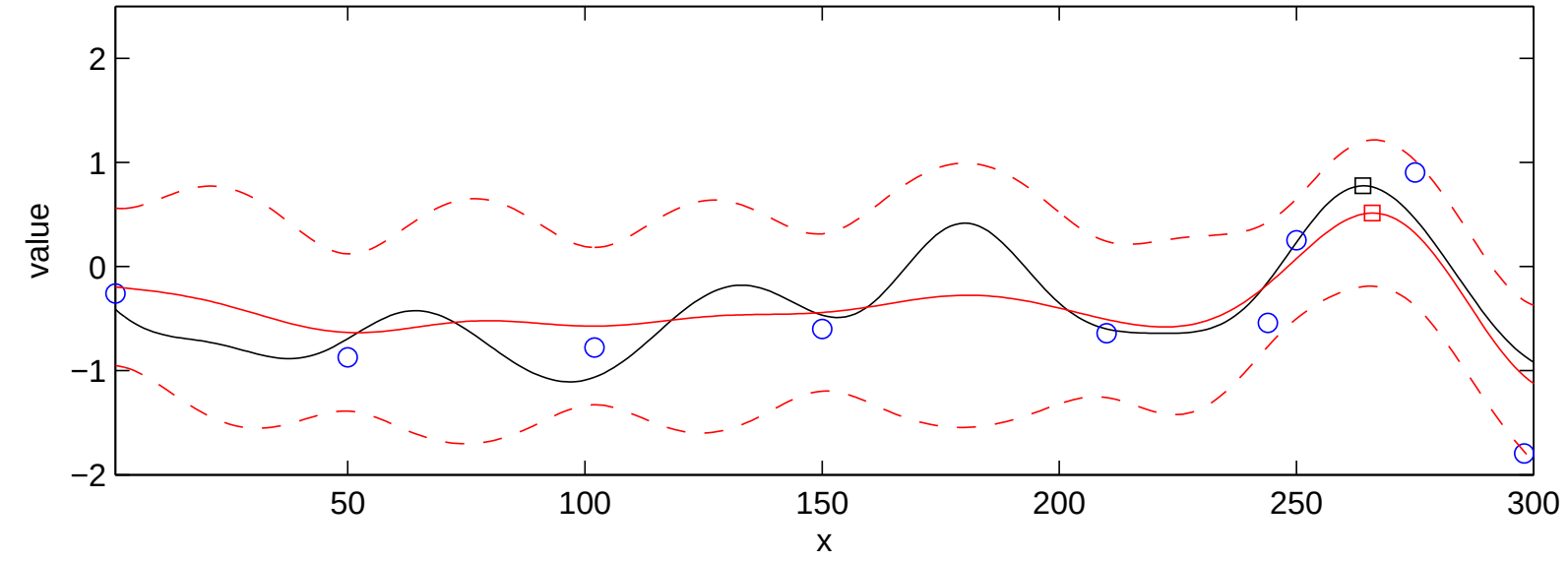

log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

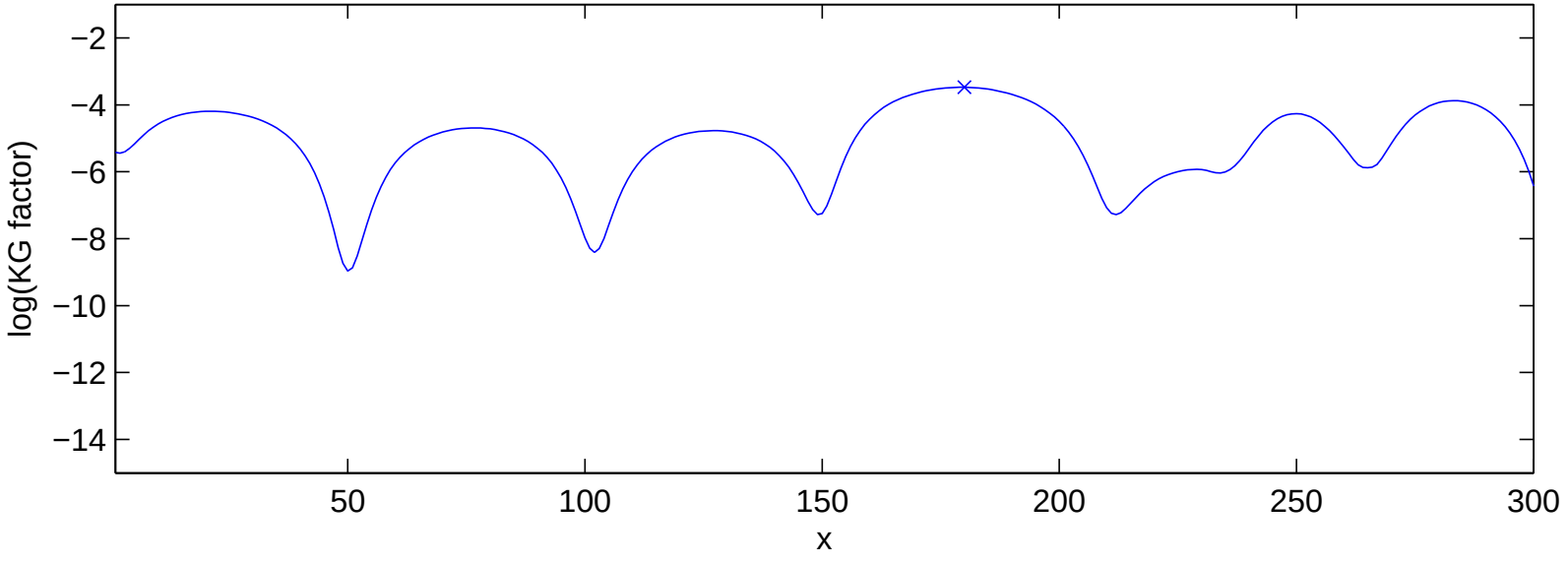
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300

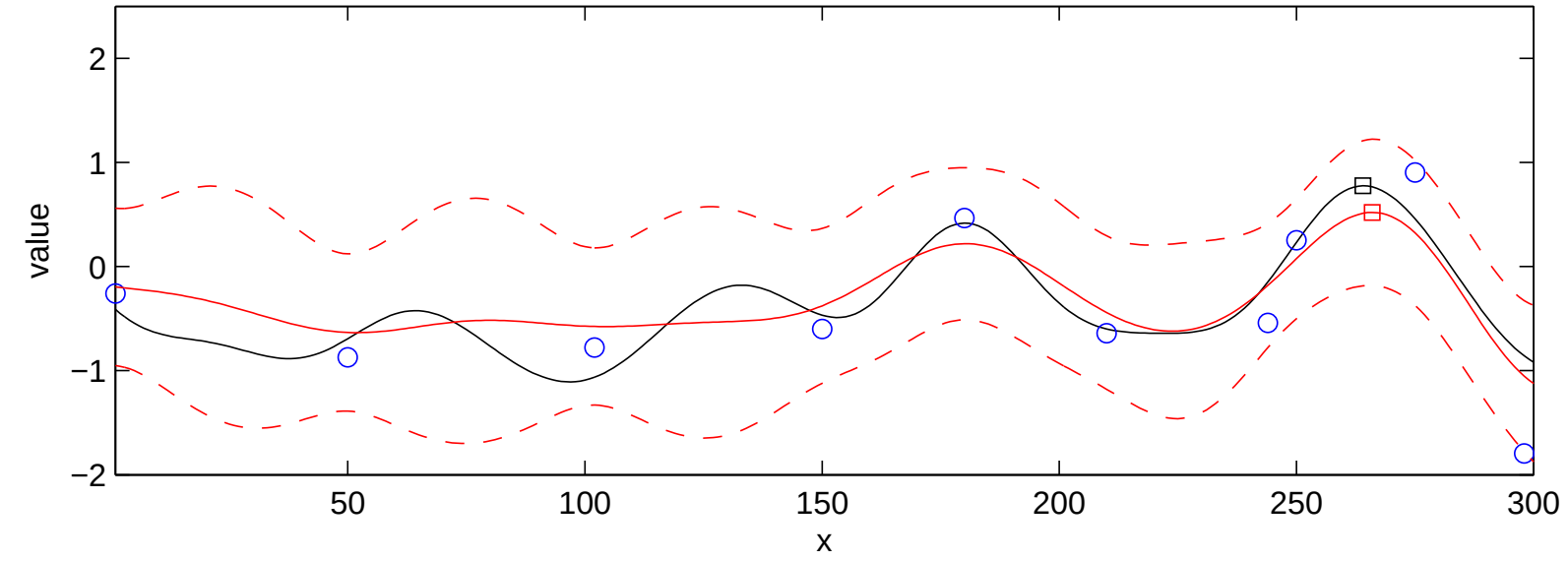

log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

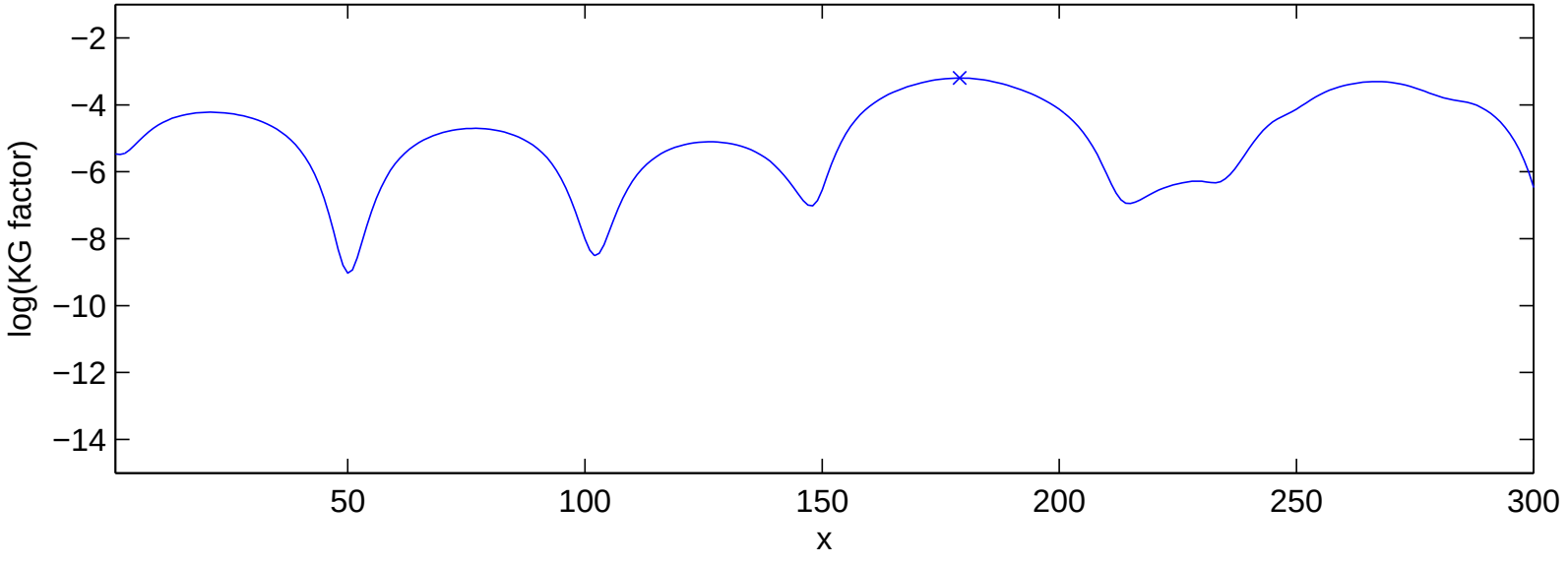
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300

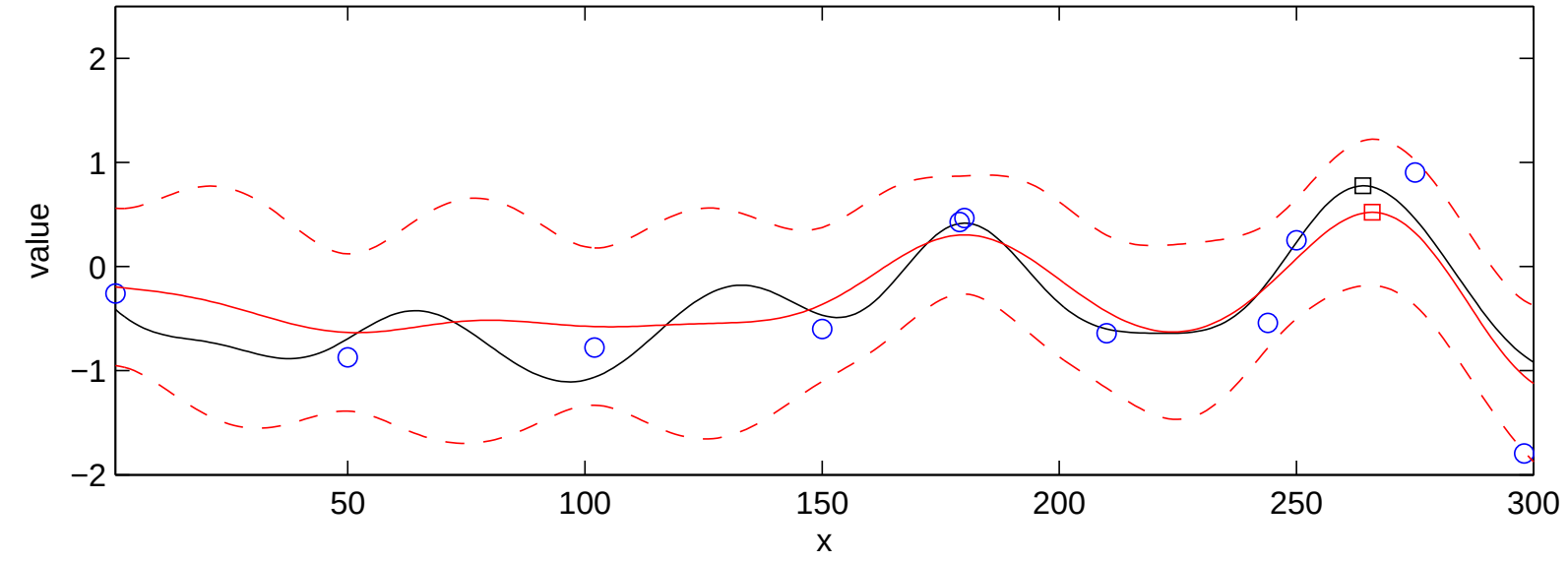

log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

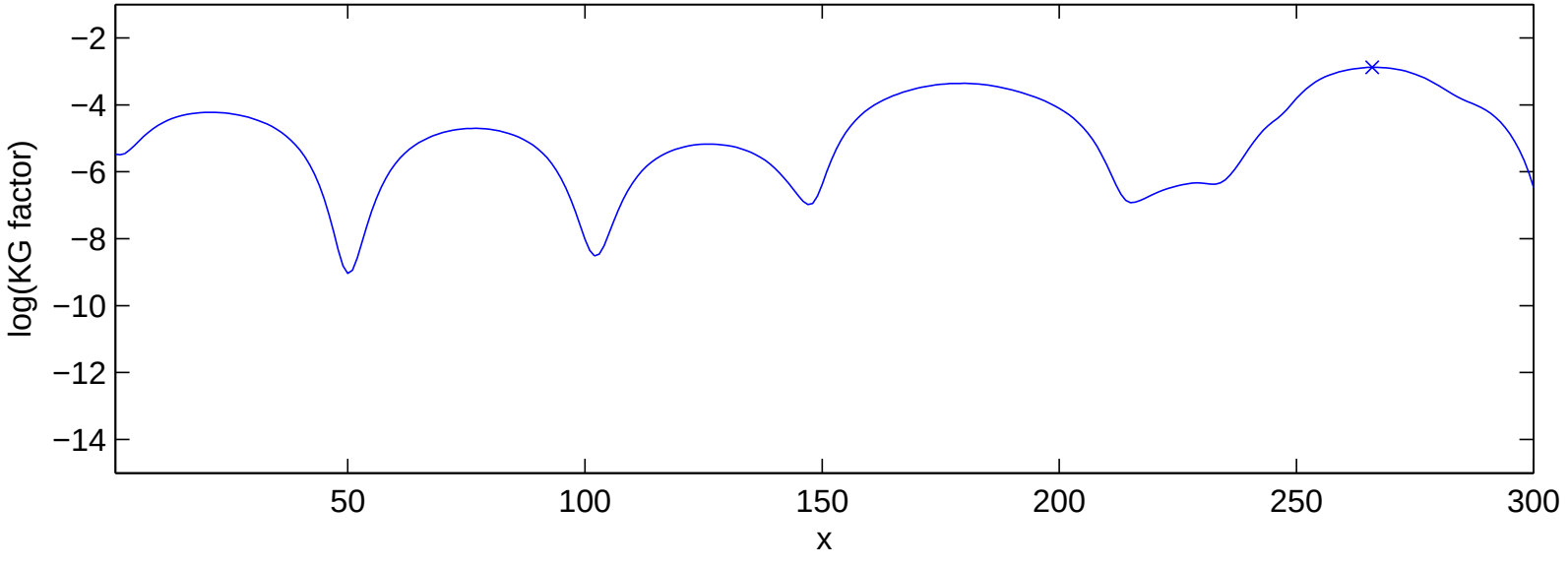
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300

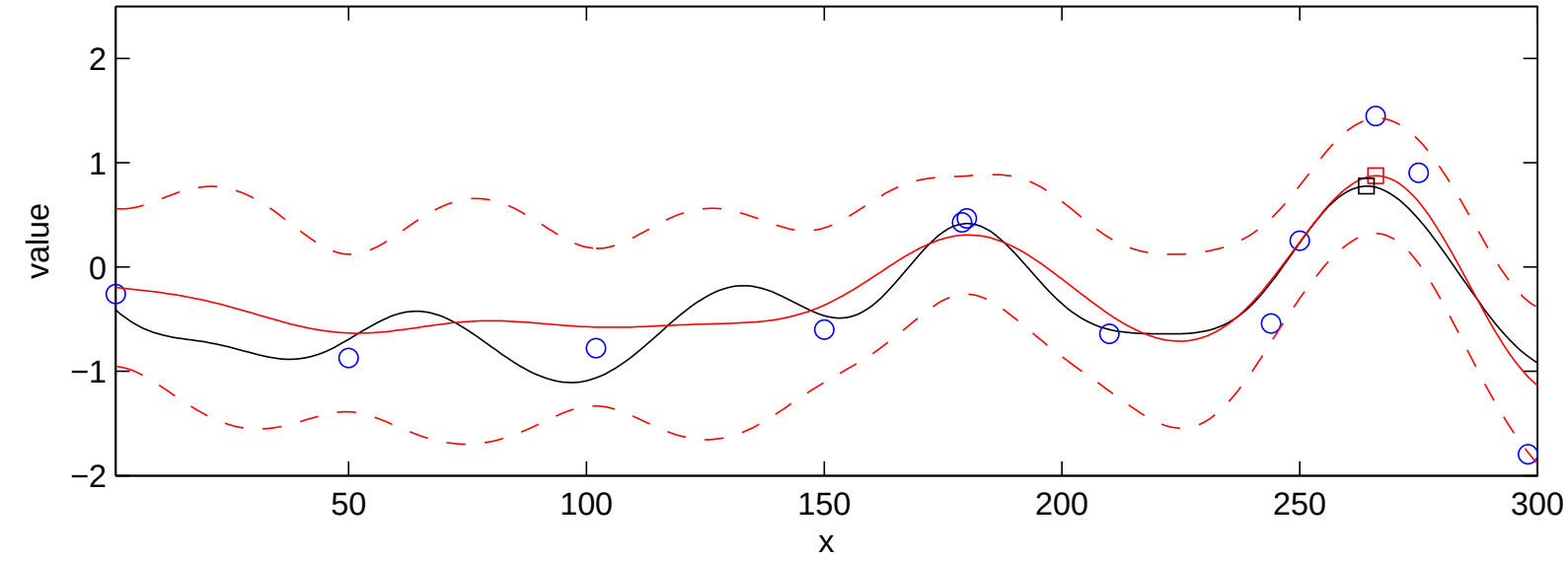

log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

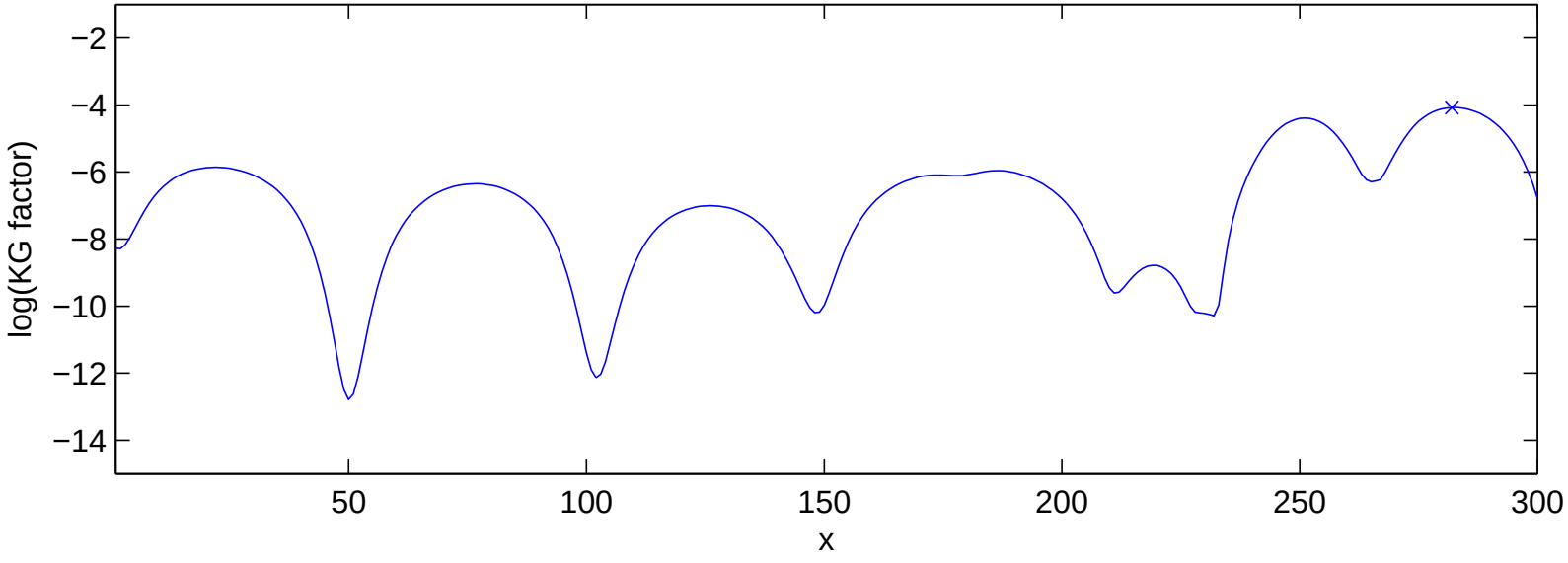
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300

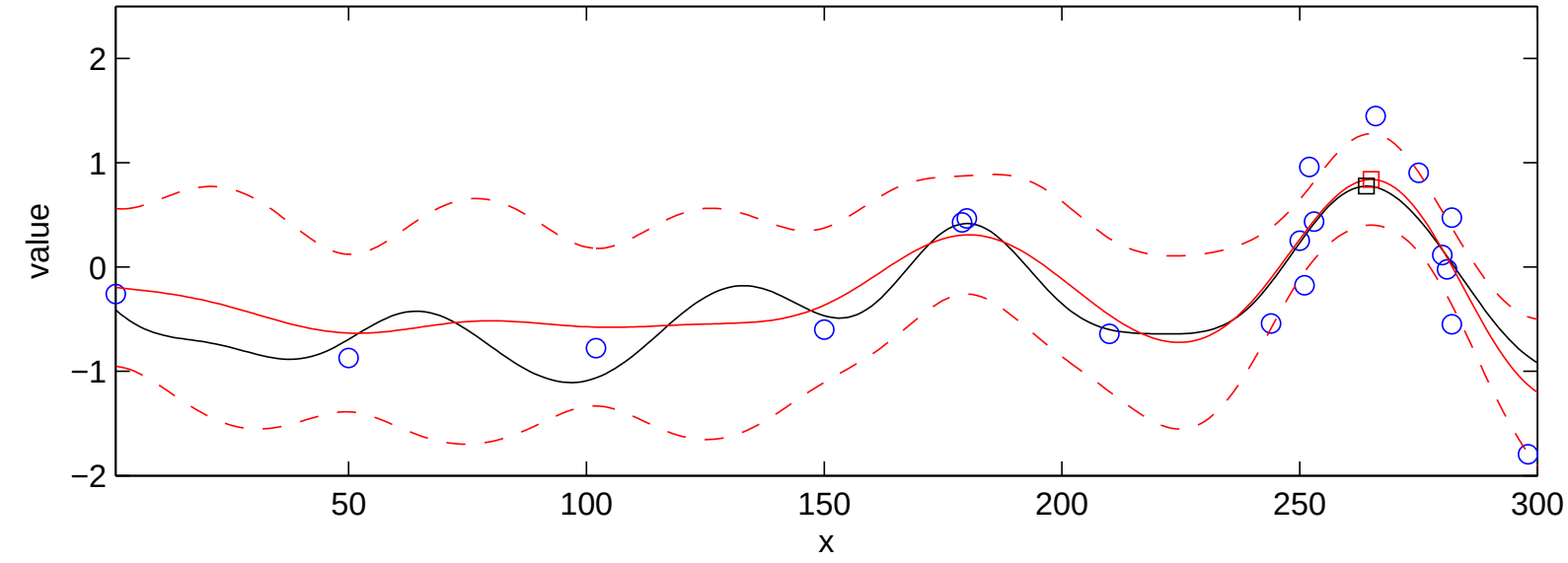

log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

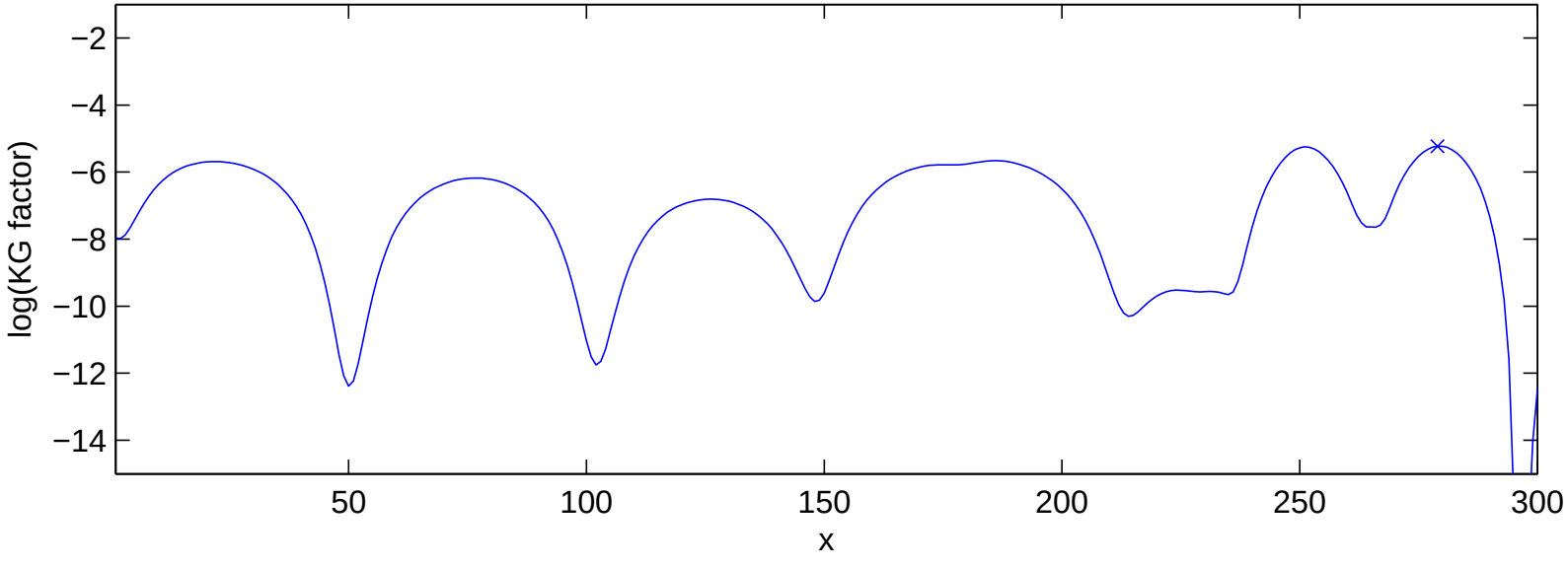
KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300


log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

KG Example


value
x
2
1
0
−1
−2
50
100
150
200
250
300


log(KG factor)
x
−2
−4
−6
−8
−10
−12
−14
50
100
150
200
250
300

Conclusion

- Average-case performance allows us to tune the way we choose between algorithms to our problem class.
- Dynamic programming gives a characterization of the algorithm with optimal average-case performance.
- Value of information is one way to create algorithms with good, but not optimal, average-case performance.
- Using a multivariate normal prior is one way to exploit knowledge of the problem class to create a faster algorithm.

Further Reading

- Warren Powell and Ilya Ryzhov have a recent book called "Optimal Learning" that covers many of these topics.
- Beyond the WSC paper, other survey papers and review articles are available on my website http://people.orie.cornell.edu/pfrazier/
- Material on average-case performance comes from [Waeber et al., 2012].
- Material on KG for independent beliefs comes from [Frazier et al., 2008].
- Material on KG for correlated beliefs comes from [Frazier et al., 2009].

Any Questions?

References I

Ankenman, B., Nelson, B. L., and Staum, J. (2008).

Stochastic kriging for simulation metamodeling.

In *Proceedings of the 40th Conference on Winter Simulation*, pages 362–370. Winter Simulation Conference.

Ankenman, B., Nelson, B. L., and Staum, J. (2010).

Stochastic Kriging for Simulation Metamodeling.

Operations Research, 58(2):371–382.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996).

Neuro-Dynamic Programming.

Athena Scientific, Belmont, MA.

Chick, S. and Frazier, P. (2012).

Sequential sampling for selection with economics of selection procedures.

Management Science, 58:550–569.

Chick, S. E., Branke, J., and Schmidt, C. (2007).

New greedy myopic and existing asymptotic sequential selection procedures: preliminary empirical results.

In *Proceedings of the 2007 Winter Simulation Conference*, Piscataway, NJ. Winter Simulation Conference, IEEE.

Chick, S. E., Branke, J., and Schmidt, C. (2010).

Sequential Sampling to Myopically Maximize the Expected Value of Information.

INFORMS J. on Computing, 22(1):71–80.

DeGroot, M. H. (1970).

Optimal {S}tatistical {D}ecisions.

McGraw Hill, New York.

References II

Frazier, P. and Powell, W. (2010).

Paradoxes in learning and the marginal value of information.
Decision Analysis, 7(4):378–403.

Frazier, P., Powell, W. B., and Dayanik, S. (2008).

A knowledge gradient policy for sequential information collection.
SIAM Journal on Control and Optimization, 47(5):2410–2439.

Frazier, P., Powell, W. B., and Dayanik, S. (2009).

The knowledge gradient policy for correlated normal beliefs.
INFORMS Journal on Computing, 21(4):599–613.

Goldberg, P. W., Williams, C. K. I., and Bishop, C. M. (1998).

Regression with input-dependent noise: A gaussian process treatment.
Advances in neural information processing systems, pages 493–499.

Gupta, S. S. and Miescke, K. J. (1996).

Bayesian look ahead one-stage sampling allocations for selection of the best population.
Journal of statistical planning and inference, 54(2):229–244.

Judd, K. L. (1998).

Numerical methods in economics.
MIT Press, Cambridge, MA.

Powell, W. B. (2007).

Approximate Dynamic Programming: Solving the curses of dimensionality.
John Wiley and Sons, New York.

References III

Rasmussen, C. E. and Williams, C. K. I. (2006).
Gaussian Processes for Machine Learning.
MIT Press, Cambridge, MA.

Waeber, R., Frazier, P., and Henderson, S. (2012).
A framework for selecting a selection procedure.
ACM Transactions on Modeling and Computer Simulation, 22(3).