Biomimicry for Optimization, Control, and Automation

Kevin M. Passino

Dept. Electrical Engineering
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210-1272
(614) 292-5716, passino@ee.eng.ohio-state.edu
http://eewww.eng.ohio-state.edu/~passino

THE OHIO STATE UNIVERSITY

Copyright

LEARNING

Focus:

- Learning and function approximation (basics, heuristic adaptive control)
- Least squares methods (training/learning, linear)
- Gradient methods (training/learning, nonlinear)
- Adaptive control (direct/indirect, stability)

Learning and Function Approximation

Psychology and Neuroscience of Learning: Classical Conditioning

➙ Learning: "any process through which experience at one time can alter an individual's behavior at a future time" [5].

- Need "memory" (not necessarily in a nervous system).

➙ Learning: "learning is an enduring change in the mechanisms of behavior involving specific stimuli and/or responses that results from prior experience with similar stimuli and experiences." [3]

➙ Control engineering: Learning is the process of the organism interacting with its environment and using that experience to modify its behavior so that it in the future it is more successful in its environment.

- Does learning always imply performance improvement?

Perhaps not.

- Two types of learning:

  1. Learning aspects of the environment and storing facts, relations, characteristics, etc., in "explicit memory" (i.e., memory available to our consciousness so we can deliberately recall it)
  2. Learning how to do things (e.g., when you acquire motor or perceptual skills) that we store in "implicit memory" (i.e., a type of memory that is unavailable to consciousness so it generally cannot be recalled).

Habituation and Sensitization: Nonassociative Learning

- Habituation involves learning to ignore benign stimuli (e.g., some smells, sounds).
- Sensitization involves learning to react to important stimuli.

- “Stimulus specificity” characteristic to habituation (e.g., some animals can only be habituated to certain stimuli) that is generally not present for sensitization (many animals can become sensitized to almost any stimuli that they can sense).
- Habituation and sensitization are learning processes that modify existing stimulus-response patterns.

The Classical Conditioning Process: Pavlov's Dog

- ➙ Classical conditioning—a behaviorist approach to learning.
- ➙ Organism has a natural (instinctual) reflexive type response (called the "unconditioned response," UR) to some stimulus (called the "unconditioned stimulus," US).
  - Suppose there exisits a stimulus (called the "conditioned stimulus," CS) that will not instinctually elicit this same response.
  - Learning (training) process is conducted where the unconditioned and conditioned stimuli are "paired" by presenting the conditioned stimulus somewhat before the unconditioned stimulus to the organism.
  - Repeat this experiment several times
  - If the stimuli and the length of time between their presentation

are chosen properly, the organism will actually evoke the unconditioned response (the "conditioned respsonse," CR) when only the conditioned stimulus is applied.

- It learned to pair (associate) the conditioned and unconditioned stimuli so that even when the unconditioned stimulus is not present, the conditioned stimulus can evoke the response.
- The learning of reflexes from instinctual reflexes.

➙ Example: Pavlov's dog.

Example: Classical Conditioning at the Neural Level in Aplysia

Figure 122: *Aplysia* in its natural habitat (figure taken from[5]).

- Aplysia only have about 20,000 neurons—and some are quite large.

- Several of the Aplysia's natural behaviors can be modified by learning—and involve only about 100 neurons.
- A behavior of this type is the so called "gill-withdrawal" reflex where if the Aplysia is touched anywhere on its skin it pulls its gill into its body as if it were protecting against an attack.
- Unconditioned response is the normal gill-withdrawal reflex in response to a touch to the skin.
- Consider neural-level

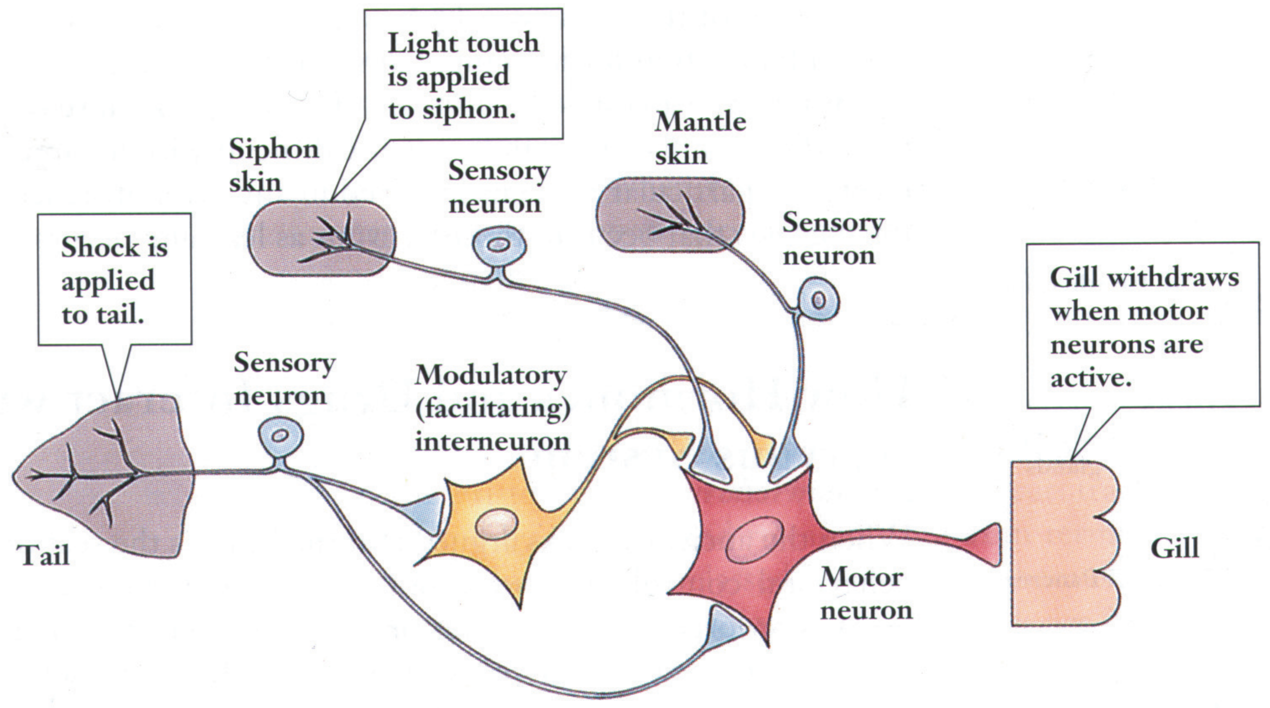
Light touch is applied to siphon.
Siphon skin
Sensory neuron
Mantle skin
Sensory neuron
Shock is applied to tail.
Sensory neuron
Modulatory (facilitating) interneuron
Gill withdraws when motor neurons are active.
Tail
Motor neuron
Gill

Figure 123: Neural level learning for gill-withdrawal learning in Aplysia (figure taken from [5]).

- Stimulation of sufficient strength anywhere on the skin excites the sensory neurons, which then excite the motor neurons that signal muscles to withdraw the gill.

- The sensory neurons signal "modulatory (facilitating) interneurons" but these are only activated when there is a particularly strong stimulus (e.g., an electic shock, or in nature when bitten by a predator).
- When the modulatory interneurons are active they release a chemical substance called a "neuromodulator" at some "slow" synapses onto axon terminals of sensory neurons.
- When these modulatory interneurons repeat this many times the neuromodulator chemicals have the capability to start a chain reaction in the sensory neurons where they grow new synaptic connections onto motor neurons and "strengthen" existing ones.
- This makes the motor neurons more sensitive to inputs from the sensory neurons so that a weak stimulus that normally

would not cause a gill reflex becomes capable of evoking it (sensitization).

- In fact, the sensory neurons are impacted more significantly by the neuromodulator if they have have just been activated; this provides a possible mechanism for classical conditioning.

➙ Example training (can model as gradient, Hebbian):

1. An electrical shock to the tail—US
2. A very weak stimulus to the skin (in particular the "siphon") can serve as the CS.
3. Learning–several times pair the conditioned and unconditioned stimuli are paired
4. Result: If only the conditioned stimulus is applied it will evoke the gill-withdrawal reflex (motor neurons become more sensitive to the sensory neurons so that a light touch can evoke the gill-reflex).

Characteristics of Classical Conditioning

- ➙ Classical Conditioning as Learning to Predict Events:
  - Learning how to predict the US by observing the CS. May be a genetic predisposition to for associating the CS and US.
  - Call the time elapsed between application of the CS and US the "inter-stimulus interval" (ISI).

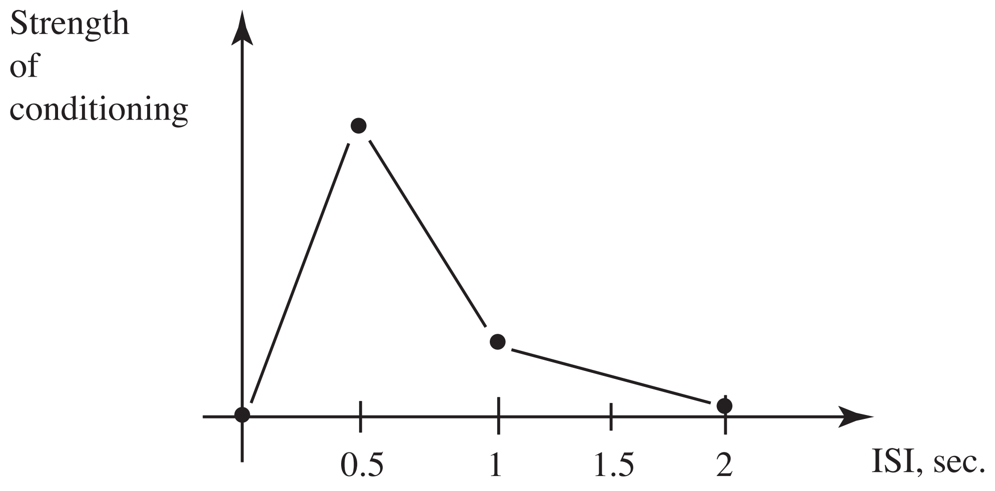

Figure 124: Effect of inter-stimulus interval on strength of conditioning for Aplysia (data taken from [7], however, only the general shape is plotted).

- Strength of conditioning (i.e., how much is learned)
- Aplysia can best learn to predict events that are spaced at about 0.5 sec. (too close or too far of spacing does not result in learning)

- Why?
- Evolutionary forces (driven by environment)?

➙ Blocking Phenomena:

- Organisms try to use the minimal amount of information to predict an event.

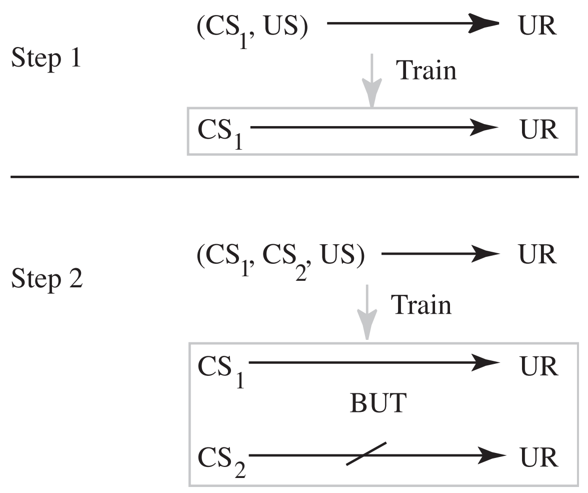

Figure 125: Blocking phenomenon in classical conditioning.

- Learning to predict the US via CS_2 was “blocked” by CS_1.

  - The amount of conditioning depends on how "surprising" the UR is.

- ➙ Extinction:
  - How permanent is the "conditioned reflex" (e.g., will the dog, for the rest of its life, always salivate when it hears a bell ring?)?
  - "Extinction" of the conditioned reflex occurs if the bell rings a number of times without the unconditioned stimulus (food).
  - Learning new associations can inhibit past associations that were learned.
  - But, a type of "spontaneous recovery" occurs
  - Extinction should not be thought of as forgetting, but as a process of learning something new (e.g., that something will not occur).

➙ Generalization and Discrimination:

- "Generalization"—after training with CS_1, other stimuli (CS_i, $i \neq 1$) that are similar to the conditioned stimulus will actually evoke the conditioned reflex in the same way as the conditioned stimulus.
- With sufficient sensory capabilities organisms can learn to "discriminate" between similar stimuli.
- "Discrimination training" can be used to reduce the effects of generalization (suppose CS_1, CS_2 similar so after training CR produced by either; train again with CS_1 presented with the unconditioned stimulus, but CS_2 is presented repeatedly without the unconditioned stimulus $\rightarrow$ conditioned reflex between CS_2 and the unconditioned response will become extinct so it learns different responses to the two stimuli).
- Related to sensitization.

Psychology of Learning While Acting: Operant Conditioning

➙ Key concept: Operant conditioning involves learning which actions are most likely to lead to goal achievement.

- Some view habituation, sensitization, and classical conditioning as “building blocks” for learning (evolution)
- Shades of these in operant conditioning and more complex learning processes (e.g., learning how to predict, plan, and schedule in a *dynamic and stochastic* environment).

Training Pigeons for Missile Guidance

- Skinner trained pigeons to peck at images of boats projected onto a screen by giving them a food reward each time that they pecked at the correct position

➙ Operant conditioning is a training process

➙ Strengthening of likelihood of actions in direction of success.

➙ An increase in frequency of occurrence for successful activities.

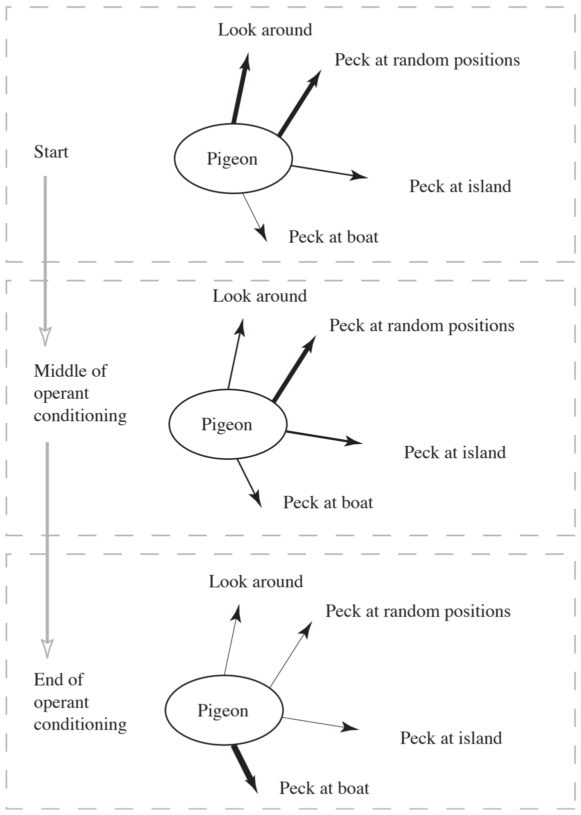

Figure 126: Depiction of reinforcement in operant conditioning (Thorndike's "law of effect").

Characteristics of Operant Conditioning

- Operant Conditioning as Learning to Predict Consequences of Actions:
  - Operant conditioning can be viewed as learning how to predict consequences of actions.
  - Organism modifies its frequency of taking various actions to optimize the likelihood of getting rewards.
  - Operant conditioning is constrained by the interval of time between action and reward.
  - Environmental context can affect operant learning
  - Operant learning generally gives the organism an ability to shape its own environment so it is best suited for survival, rather than just trying to cope with a given environment.
- Shaping, Partial Reinforcement, Extinction, Reinforcer Control:
  - Concepts related to "generalization" in classical

conditioning hold.

– “Shaping”—for pigeons, the random pecking response is “shaped” in a way that it is similar to the proper response in the sense that it will give a “partial reward”
– “Extinction” and “spontaneous recovery” conceptually similar to classical conditioning
– “Reinforcer”—term meaning goal or reward.
– There exist “partial reinforcers” (periodic or lower magnitude reward), “positive reinforcement” (a process that increases the likelihood that a response will occur) and “negative reinforcement” (when the removal of a stimulus after a response makes the response more likely to occur).
– There are schedules for providing rewards (fixed, variable). The “partial reinforcement learning effect,” is that the variable schedule training methods are typically more resistant to extinction—the animal also learns that it has to

be patient.

➙ Discrimination Training and Chaining:

- Can do "discrimination training" in operant conditioning.
- You can train an animal to recognize the *situation* that it is currently in and to only take actions when in that situation (it uses "contextual information").
- Can use this to train animals to do sequences of actions.
- After discrimination training the situation is associated with receiving a reinforcer so the situation itself acquires some reinforcing value.
- The situation is sometimes said to be a "secondary reinforcer" (for humans, e.g., money).
- Training to execute a sequence (chaining):

$$\text{action } 1 \rightarrow \text{situation } 1 \text{ (secondary reinforcer } 1),$$
$$\ldots$$

$$\text{action } n-1 \rightarrow \text{situation } n \text{ (secondary reinforcer } n) \rightarrow$$
$$\text{action } n \rightarrow \text{goal (reinforcer)}$$

– Training can occur in a “backward manner” in the learning process.

– Training via chaining may result in an ability to predict sequences of events.

Control System Model of Operant Conditioning

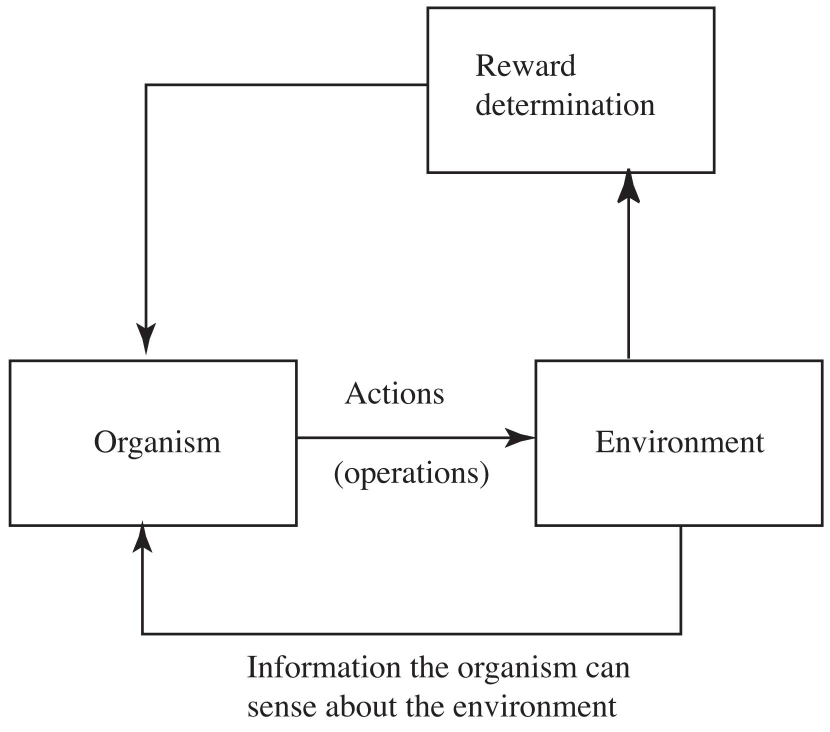

Figure 127: The operant conditioning process.

Heuristic Adaptive Control

- Use either:
  - Biomimicry of adaptation functionalities (e.g., of neural networks, operant conditioning)
  - Human-mimicry of adaptation capabilities (e.g., how humans act to control a process)
- "Heuristic approach"—good or bad? Can use an optimization perspective, or a stability-based approach.
- Relevance of planning, attention, development?

Function Approximation as Learning

➙ Let

$$F(x, \theta)$$

denote a tunable nonlinear function (neural, fuzzy, etc.) that we will use as a "function approximator."

- The input, which is known, is $x = [x_1, x_2, \ldots, x_n]^\top$ and the parameter vector $\theta = [\theta_1, \theta_2, \ldots, \theta_p]^\top$.
- There is an "approximator structure" and tunable θ
- p is the "size" of the approximator.

Using Functions to Represent Mappings in Data

➙ Let

$$y = G(x, z)$$

where its input is $x = [x_1, x_2, \ldots, x_n]^\top$, $z = [z_1, z_2, \ldots, z_{n_z}]^\top$ is an unknown "auxiliary variable," and its output is the scalar y.

- If $n_z = 0$, $G(x, z)$ is not a function of z, and denote it by $G(x)$.
- We do not have an explicit mathematical description of $G(x, z)$
- Can learn about it by performing experiments and gathering input and output data from it.
- Suppose that for the i^{th} experiment we let the input data be

$$x(i) = [x_1(i), x_2(i), ..., x_n(i)]^\top$$

the auxiliary variable

$$z(i) = [z_1(i), z_2(i), ..., z_{n_z}(i)]^\top$$

and the output data be

$$y(i) = G(x(i), z(i))$$

(hence, $x_j(i)$ is the j^{th} element of the i^{th}).

- Typically, we know that

$$x(i) \in X \subset \Re^n$$

for some (bounded) set X that we know a priori; similarly for

$$z(i) \in Z \subset \Re^{n_z}$$

➙ We will call the pair $(x(i), y(i))$ an input-output data pair (training data pair).

- A set of input-output data pairs is the training data set

$$G = \{(x(1), y(1)), \ldots, (x(M), y(M))\} \tag{25}$$

where M denotes the number of input-output data pairs contained in G.

- The function approximation problem is the problem of how to pick the value for the parameter vector θ in $F(x, \theta)$ so

$$G(x, z) = F(x, \theta) + e(x, z) \tag{26}$$

where the "approximation error" $e(x, z)$ is as small as possible for all $x \in \Re^n$ and $z \in \Re^{n_z}$, even at x such that $(x, y) \notin G$

➙ This is quite challenging if we know nothing of the function $G(x, z)$ besides what is in the training data G.

- Example: Challenge—is classical conditioning function approximation?

Choosing the Training Data Set

- We would like G to contain as much information as possible about $G(x, z)$.

➙ Unfortunately, most often the number of training data pairs is relatively small, or it is difficult to use too much data since this affects the computational complexity of the algorithms that are used to adjust θ.

➙ The key question is then, How would we like the limited amount of data in G structured so that we can adjust θ so that $F(x, \theta)$ matches $G(x, z)$ very closely?

Uniform Coverage May Help:

- Then expect to have information about how the mapping $G(x, z)$ is shaped in all regions so we should be able to approximate it well in all regions (assuming small influences from z).

- Accuracy will generally depend on the slope of $G(x,z)$ in various regions.
- Assuming the influence of z is small, in regions where the slope is high, we may need more data points to get more information so that we can do good approximation.

We Often Cannot Control What is in the Data Set:

- For instance, most often in "system identification" you cannot directly pick the data pairs since the input portion can contain both the inputs and outputs of the system.
- Basically this raises an issue of controllability of the system, which for the nonlinear case can quickly become complicated.
- In adaptive control, a conflict: input chosen for good tracking and to be "persistently exciting" to get good identification.

Relationships to Persistent Excitation:

- Intuitively, for system identification we must choose an input signal to "excite" the dynamics of the system so that we can "see," via the plant input-output data, what the dynamics are (i.e., we can see inside the "black-box").
- Excitation with a noise signal (or a random binary signal) will have a tendency to place points in X over a whole range of locations; however, there is no guarantee that uniform coverage will be achieved for nonlinear identification problems.

  It is a difficult problem to know how to pick the input signal so that G is a good data set for solving a function approximation problem.

Data Scaling:

- There are times when scaling the data can be helpful in the sense that the algorithms that are used to process the data to find θ can sometimes perform better if it is scaled.
- One simple way to scale the data is to simply multiply by a number that will force all the data values to be between -1 and $+1$.
- Such scaling can help with numerical issues, and may speed convergence of some training methods.

Example: Collecting Data for Function Approximation

- Consider an unknown function $G(x, z)$ where x is a scalar $(n = 1)$ that we can pick and at first we assume that $n_z = 0$.
- Assuming we can get uniformly spaced data in $X = [-6, 6]$, for $M = 7$ pieces of training data we get Figure 128.

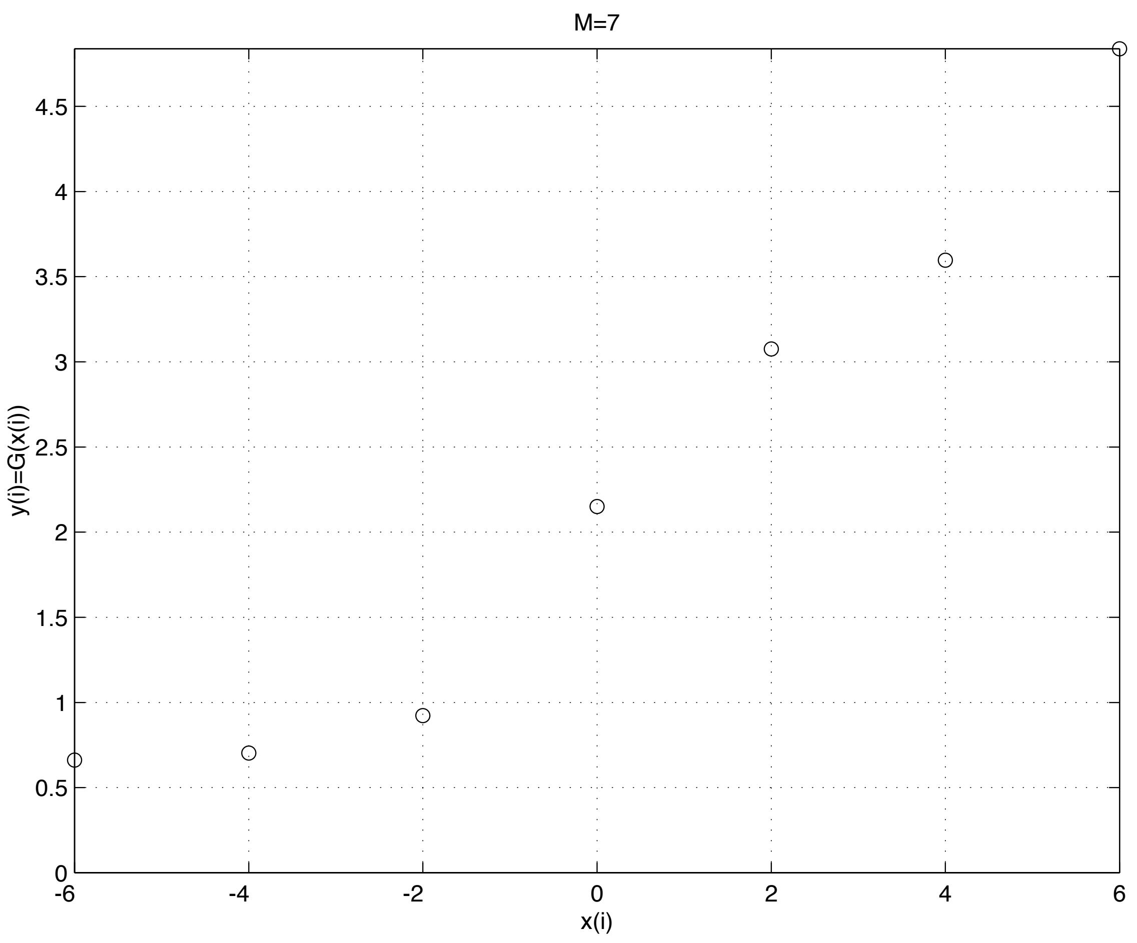

Figure 128: The training data G generated from the function $G(x, z)$, $M = 7$, $n_z = 0$.

- ➙ Notice that there is some interesting nonlinear behavior that is exhibited by the training data.
- For $M = 121$ we get Figure 129.

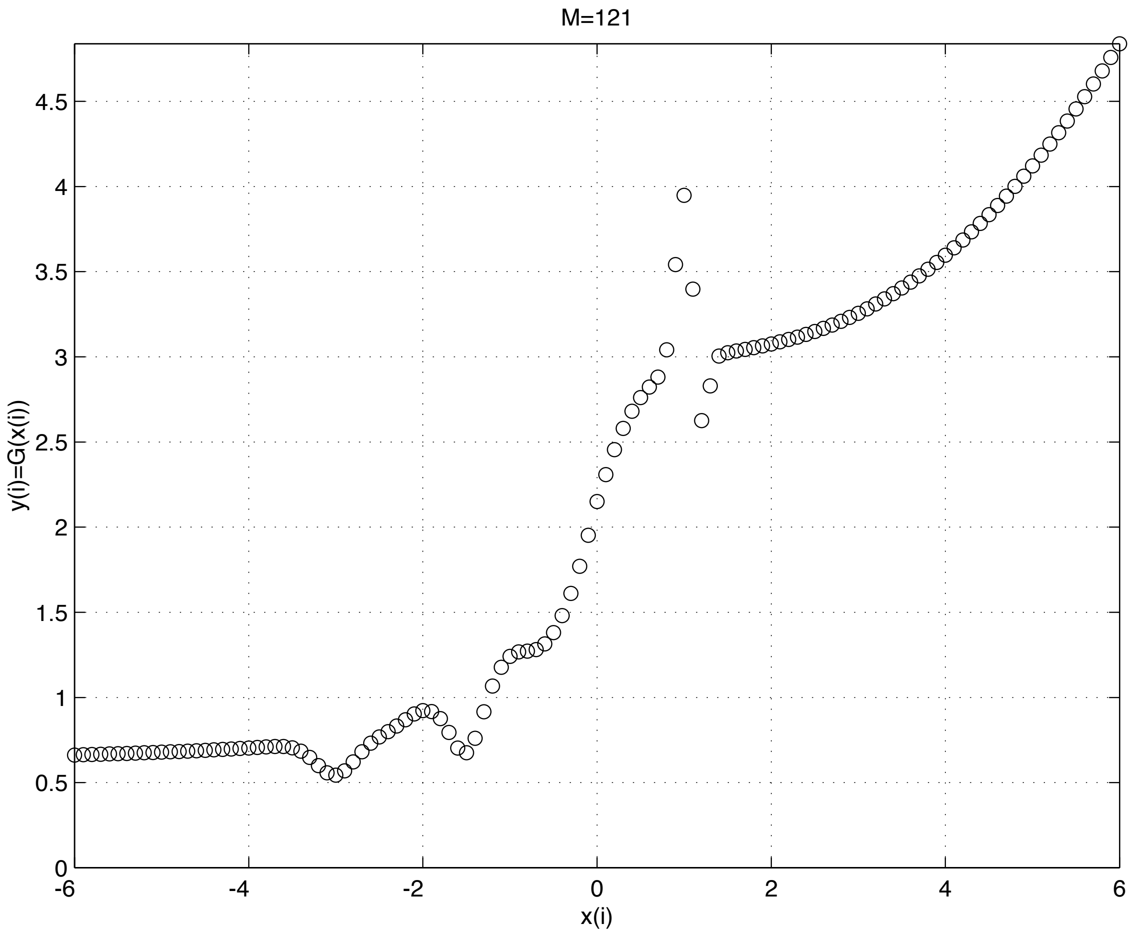

Figure 129: The training data G generated from the function $G(x, z)$, $M = 121$, $n_z = 0$.

★ The higher frequency oscillations were not seen before since our grid size was too large; is there more hidden nonlinear behavior?

- We see that without additional information about the unknown function (e.g., the maximum slope of the function) it is quite difficult to know when you have enough data to have a good representation of the function.
- As an example of how the z variable can complicate the function approximation problem, consider the case when $n_z = 1$.
- There are many types of influences that z can have on $G(x, z)$.
- When we collect training data for the $M = 121$ case we get the data shown in Figure 130.

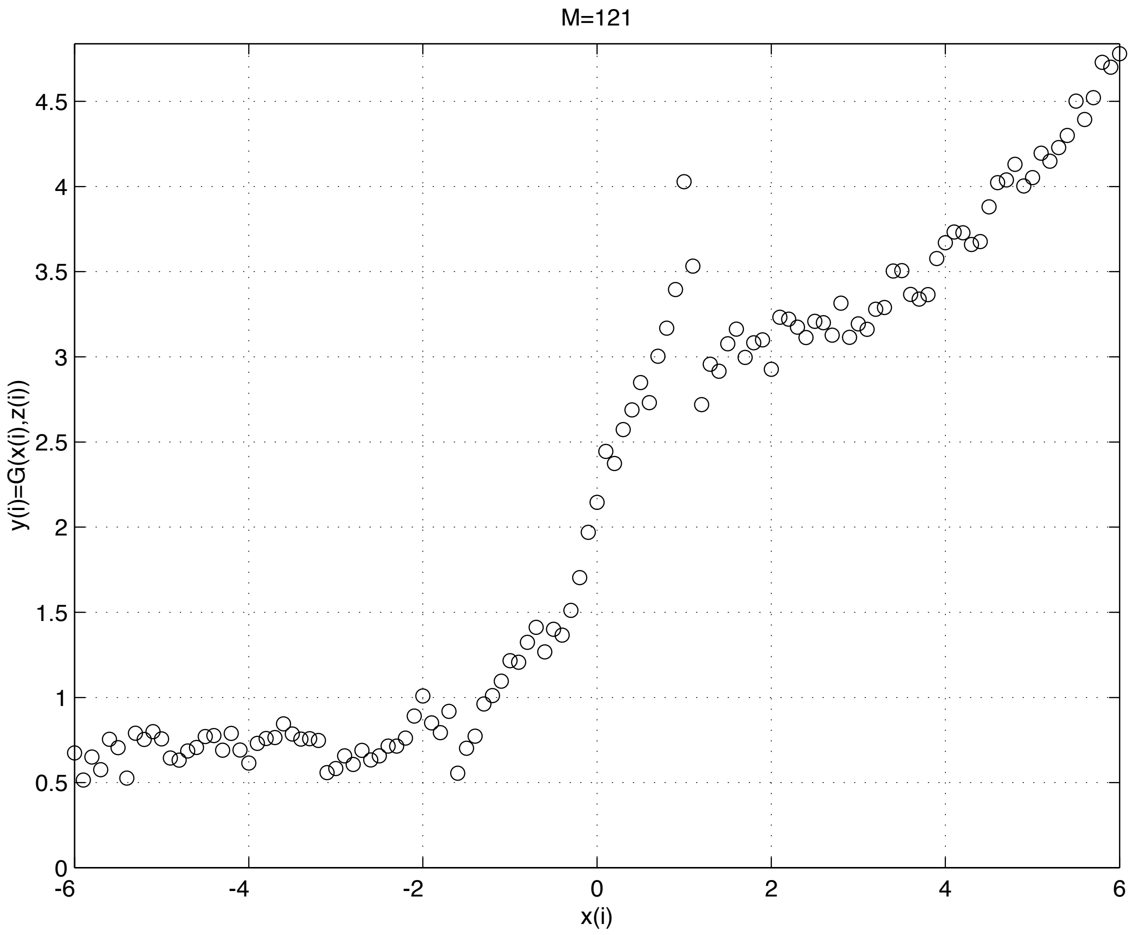

Figure 130: The training data G generated from the function $G(x, z)$, $M = 121$, and influences of the auxiliary variable z are shown.

- Now we see that $G(x, z)$ appears to be a very complex function to approximate accurately.
- There could be very high frequency information in the function when in actuality this is a type of noise (that has in fact masked some of the behavior of the function that we saw in Figure 129).
- Even though we have gathered a lot of data it is not clear how much data should be gathered since this data might not be providing useful information.

Measuring Approximation Accuracy: Using a Test Set

➙ How do we evaluate how closely the function $F(x, \theta)$ approximates the function $G(x, z)$ for a given θ?

- Notice that

$$W = \sup_{x \in X, z \in Z} \{|G(x, z) - F(x, \theta)|\} \tag{27}$$

  is a bound on the approximation error $e(x, z)$ (if it exists) and

$$W^* = \inf_{\theta} \sup_{x \in X, z \in Z} \{|G(x, z) - F(x, \theta)|\} \tag{28}$$

  is the "ideal approximation error" for a given approximator structure.

- The value (or values) of θ that achieves W^* is called the "ideal parameter value" and it is denoted by θ^*.

➙ Unfortunately, in practice, specification of such a bound like W

requires that the function $G(x, z)$ be completely known; however, we only know G.

- Therefore, in practice we are only able to evaluate the accuracy of approximation by evaluating the error between $G(x, z)$ and $F(x, \theta)$ at certain points $x \in X$ given by available input-output data.

➙ We call this set of input-output data the *test set* and denote it as Γ, where

$$\Gamma = \{(x(1), y(1)), \ldots, (x(M_\Gamma), y(M_\Gamma))\} \tag{29}$$

- You can use error measures like

$$e_\Gamma = \frac{1}{M_\Gamma} \sum_{(x(i),y(i))\in\Gamma} (y(i) - F(x(i), \theta))^2 \tag{30}$$

(the mean squared error) or

$$e_\Gamma = \sup_{(x(i),y(i))\in\Gamma} \{|y(i) - F(x(i),\theta)|\} \tag{31}$$

to measure the approximation error.

- Γ should contain a lot data that were not used to construct $F(x,\theta)$.
- In fact, one way to see if you have chosen M large enough is to use a test set with $M_\Gamma >> M$ (i.e., signficantly bigger than M) and find the error that results for the training data set and the test set and compare them.
- If the two resulting error measures are close, then it is likely that you have chosen the training data set size (i.e., M) to be large enough.

Approximator Structures

- Introduce approximator structures and tune them by hand.

➙ We will later study automatic methods for finding θ from the data G, but:

  – Some of these will only tune the parameters that enter linearly (so we must pick the others manually)

  – Even in the case where the method will tune all the parameters we need to be able to initialize the parameters (and better initializations typically lead to better results).

➙ This is why the ideas on how to manually tune approximators are valuable.

Linear and Polynomial Approximator Structures

Linear Approximators:

- In this case,

$$y = F_l(x,\theta) = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n + \theta_{n+1}$$

(notice that the last term is simply a constant so it is an "affine" mapping) or in vector notation if we let $\phi = [x^\top, 1]^\top$

$$y = F_l(x,\theta) = \theta^\top \phi(x) \tag{32}$$

where $x = [x_1, x_2, \ldots, x_n]^\top$, $\theta = [\theta_1, \theta_2, \ldots, \theta_n, \theta_{n+1}]^\top$, and we have $p = n + 1$.

➙ Linear approximators only provide for perfect representation of a class of functions that are linear (likewise for affine).

Polynomial Approximators:

- Next, consider a polynomial approximator

$$
\begin{aligned}
F_{poly}(x,\theta) \quad = \quad & \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n + \theta_{n+1} + \cdots \\
& + \theta_{n+2} x_1 x_2 + \theta_{n+3} x_1 x_3 + \cdots \\
& + \theta_{n+k} x_1^2 x_2 + \cdots
\end{aligned}
\tag{33}
$$

  where the parameters θ_i that we adjust are used to scale terms that are products of the x_i (to any finite order), and notice that F_l is a special case of F_{poly}.

- We can, of course, pick θ so that $F_{poly}(x,\theta)$ is a linear function of θ or a nonlinear function of θ.

Neural Network and Fuzzy System Approximator Structures

Multilayer Perceptron, One Hidden Layer:

- In this case, the network is shown in Figure 131.

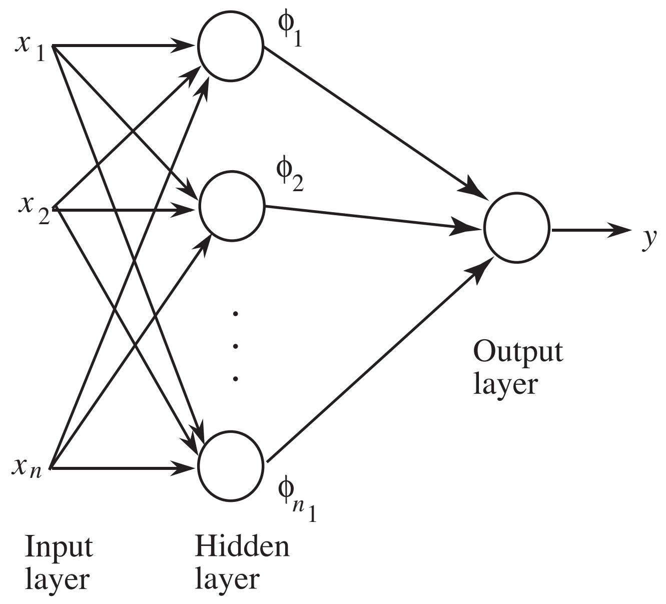

Figure 131: Multilayer perceptron with one hidden layer.

- Only consider the MISO case (easy to extend to MIMO)
- For our single hidden layer network let ϕ_j, $j = 1, 2, \ldots, n_1$ denote the output of the j^{th} neuron in the hidden layer, let b_j be the bias, and let

$$w^j = [w_{1,j}, w_{2,j}, \ldots, w_{n,j}]^\top$$

- Hence, we have

$$\phi_j = f(b_j + (w^j)^\top x)$$

  where f is the activation function (we could have different activation functions for each neuron, but for simplicity we let them all be the same).
- We will assume that the neurons in the hidden layer use nonlinear activation functions.
- Let w_j, $j = 1, 2, \ldots, n_1$ denote a weight in the output layer and let b be the bias.

- Let

$$w = [w_1, w_2, \ldots, w_{n_1}]^\top$$

- With this, and the choice of using a linear activation function in the output layer, the mathematical formula describing Figure 131 is

$$y = b + \sum_{j=1}^{n_1} w_j \left(f(b_j + (w^j)^\top x) \right) = b + \sum_{j=1}^{n_1} w_j \phi_j$$

- Now, if we choose

$$\phi = [\phi_1, \phi_2, \ldots, \phi_{n_1}, 1]^\top$$

so that

$$y = F_{mlp}(x, \theta) = [w^\top, b]^\top \phi \tag{34}$$

(note that we could use $F_{mlp}^{(1)}$ to denote this single hidden layer perceptron, but we omit the "(1)" for simplicity).

- We will consider two different choices for the parameter θ:
  - Nonlinear in the parameters: Choose

$$\theta = [(w^1)^\top, b_1, (w^2)^\top, b_2, \ldots, (w^{n_1})^\top, b_{n_1}, w^\top, b]^\top$$

  - Linear in the parameters: Suppose that you know the values of the w^j and b_j, $j = 1, 2, \ldots, n_1$. Choose

$$\theta = [w^\top, b]^\top$$

Notice that if the w^j and b_j, $j = 1, 2, \ldots, n_1$, are known, then the ϕ_j, $j = 1, 2, \ldots, n_1$ are known once the input x is specified, so ϕ is known. For this choice of θ,

$$y = F_{mlp}(x, \theta) = \theta^\top \phi$$

so y is a linear function of the parameter vector θ.

Clearly, the nonlinear in the parameter case is more general; however, that we have better methods to adjust linear in the parameter approximators.

- Example: $n = 1$, $n_1 = 2$ (i.e., two neurons in the hidden layer), and let each neuron be the logistic (sigmoidal) nonlinearity.
- We have $w^1 = [w_{1,1}]^\top$, $w^2 = [w_{1,2}]^\top$ as the weights and b_1 and b_2 as the biases for the hidden layer.
- For the output layer $w = [w_1, w_2]^\top$ where w_j, $j = 1, 2$, are weights and b is the bias, for the neuron in the output layer.

➙ All these weights and biases specify the shape of the nonlinearity that is implemented by the neural network.

- If you try, for example, to approximate the function in Figure 130 with a network that has a single sigmoid, you may think of it as a "smooth step function" where:

- You can shift the step horizontally (i.e., to the left and right) by changing the value of the bias b_1,
- Change the steepness of the step by adjusting $w^1 = w_{1,1}$,
- Scale the size of the step by changing $w = w_1$,
- Offset the step vertically by adjusting the bias b.

➙ Using these ideas, manual tuning of the parameters (both the ones that enter linearly and in a nonlinear fashion) results in the approximator in Figure 132.

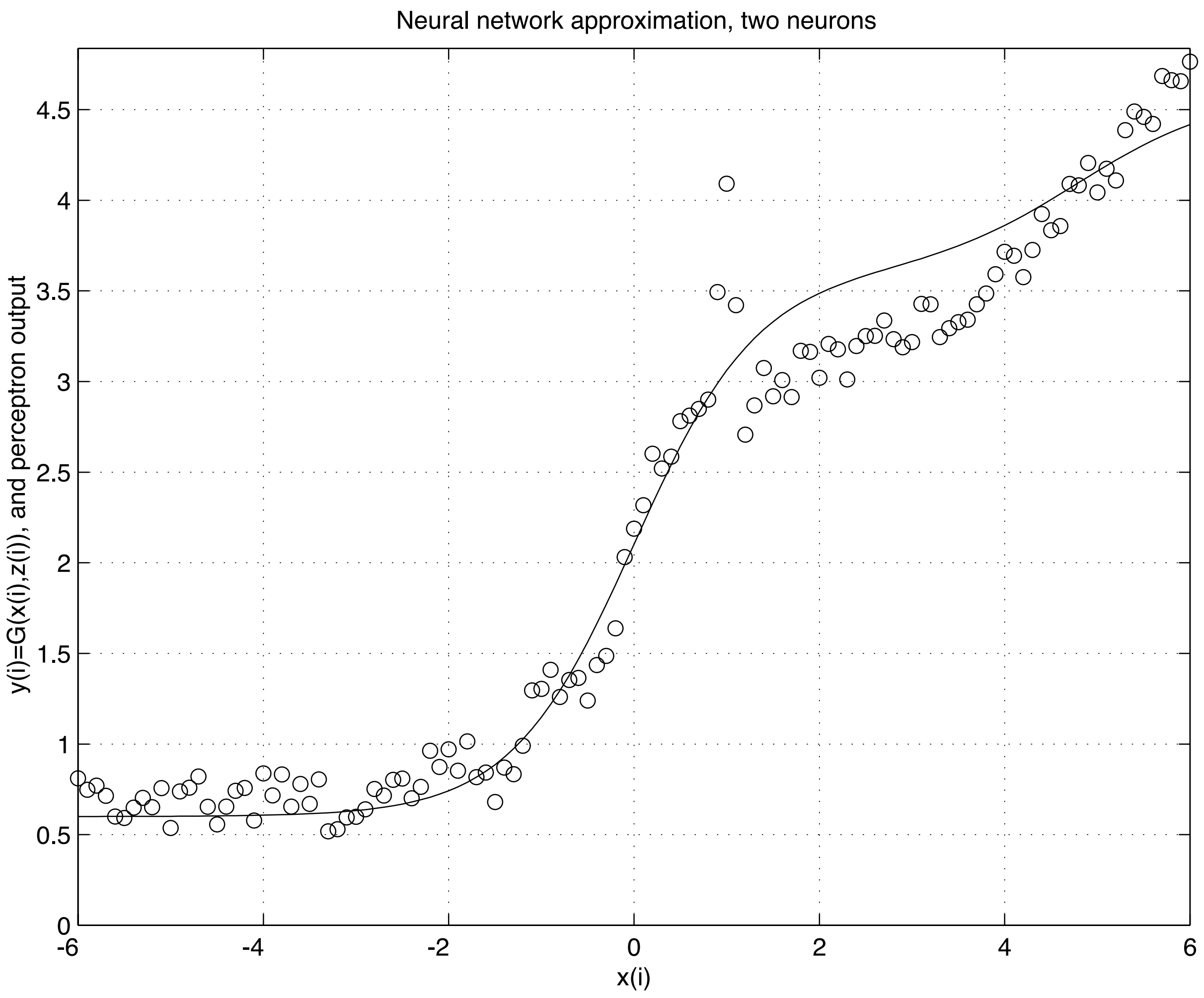

Figure 132: Neural network approximation, two neurons.

Takagi-Sugeno Fuzzy Systems:

- We consider a Takagi-Sugeno fuzzy system that is given by

$$y = F_{ts}(x, \theta) = \frac{\sum_{i=1}^{R} g_i(x)\mu_i(x)}{\sum_{i=1}^{R} \mu_i(x)}$$

  where if $a_{i,j}$ are constants,

$$g_i(x) = a_{i,0} + a_{i,1}x_1 + \cdots + a_{i,n}x_n$$

- Also, for $i = 1, 2, \ldots, R$,

$$\mu_i(x) = \prod_{j=1}^{n} \exp\left(-\frac{1}{2} \left(\frac{x_j - c_j^i}{\sigma_j^i} \right)^2 \right)$$

  where c_j^i is the point in the j^{th} input universe of discourse where the membership function for the i^{th} rule achieves a maximum, and $\sigma_j^i > 0$ is the relative width of the membership function for the j^{th} input and the i^{th} rule.

- We are using center-average defuzzification and product for the premise and implication.
- Consider two different choices for θ:
  - Nonlinear in the parameters: One choice for θ above is to let

$$\begin{aligned}\theta &= [c_1^1, \ldots, c_n^R, \sigma_1^1, \ldots, \sigma_n^R, \\ &\quad a_{1,0}, a_{2,0}, \ldots, a_{R,0}, a_{1,1}, a_{2,1}, \ldots, \\ &\quad a_{R,1}, \ldots, a_{1,n}, a_{2,n}, \ldots, a_{R,n}]^\top\end{aligned}$$

  - Linear in the parameters: Note that

$$y = \frac{\sum_{i=1}^R a_{i,0}\mu_i(x)}{\sum_{i=1}^R \mu_i(x)} + \frac{\sum_{i=1}^R a_{i,1}x_1\mu_i(x)}{\sum_{i=1}^R \mu_i(x)} + \cdots + \frac{\sum_{i=1}^R a_{i,n}x_n\mu_i(x)}{\sum_{i=1}^R \mu_i(x)}$$

Let

$$\begin{aligned}\phi \;&=\; [\xi_1(x), \xi_2(x), \ldots, \xi_R(x), x_1\xi_1(x), x_1\xi_2(x), \ldots, x_1\xi_R(x), \ldots, \\ &\quad x_n\xi_1(x), x_n\xi_2(x), \ldots, x_n\xi_R(x)]^\top\end{aligned}$$

and

$$\theta = [a_{1,0}, a_{2,0}, \ldots, a_{R,0}, a_{1,1}, a_{2,1}, \ldots, a_{R,1}, \ldots, a_{1,n}, a_{2,n}, \ldots, a_{R,n}]^\top$$

and

$$\xi_j = \frac{\mu_j(x)}{\sum_{i=1}^{R} \mu_i(x)}$$

$j = 1, 2, \ldots, R$, so that

$$y = F_{ts}(x, \theta) = \theta^\top \phi(x)$$

represents the Takagi-Sugeno fuzzy system.

- As an example, consider a Takagi-Sugeno fuzzy system with one input and $R = 4$ rules that we will use to approximate our same unknown function in Figure 130.

- In this case, we have to pick the parameters for

$$\mu_i(x) = \exp\left(-\frac{1}{2}\left(\frac{x_1 - c_1^i}{\sigma_1^i}\right)^2\right)$$

$i = 1, 2, 3, 4$ and the parameters of the corresponding g_i functions.

- In the $n = 1$ case we have $g_i(x) = a_{i,0} + a_{i,1}x_1$ (lines).

➙ Hence, it uses the functions μ_i (actually ξ_i) to interpolate between lines.

➙ Split X into regions and use lines as in Figure 133 and we get the approximator mapping in Figure 134.

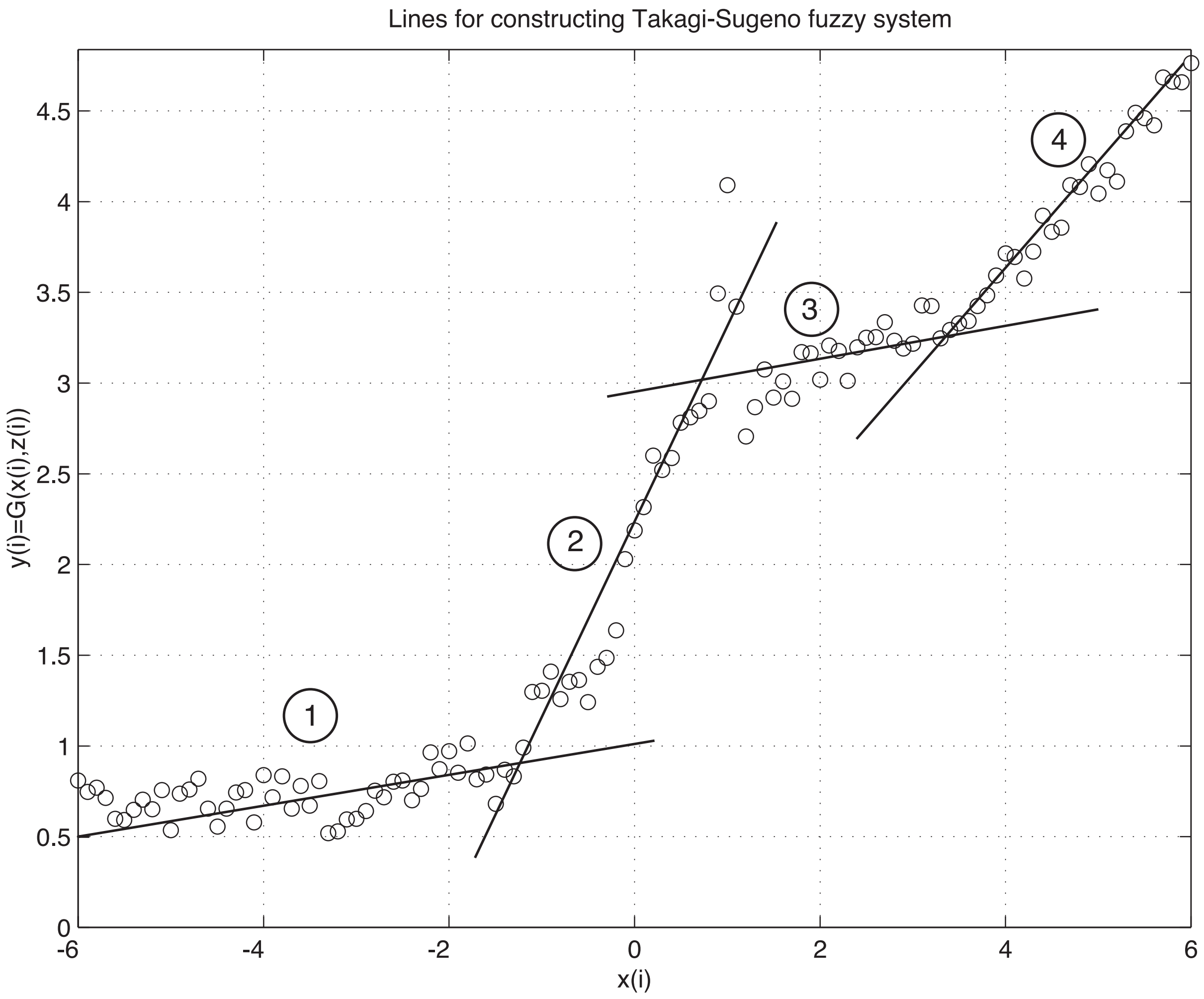

Figure 133: Training data and lines used in the Takagi-Sugeno fuzzy system approximator.

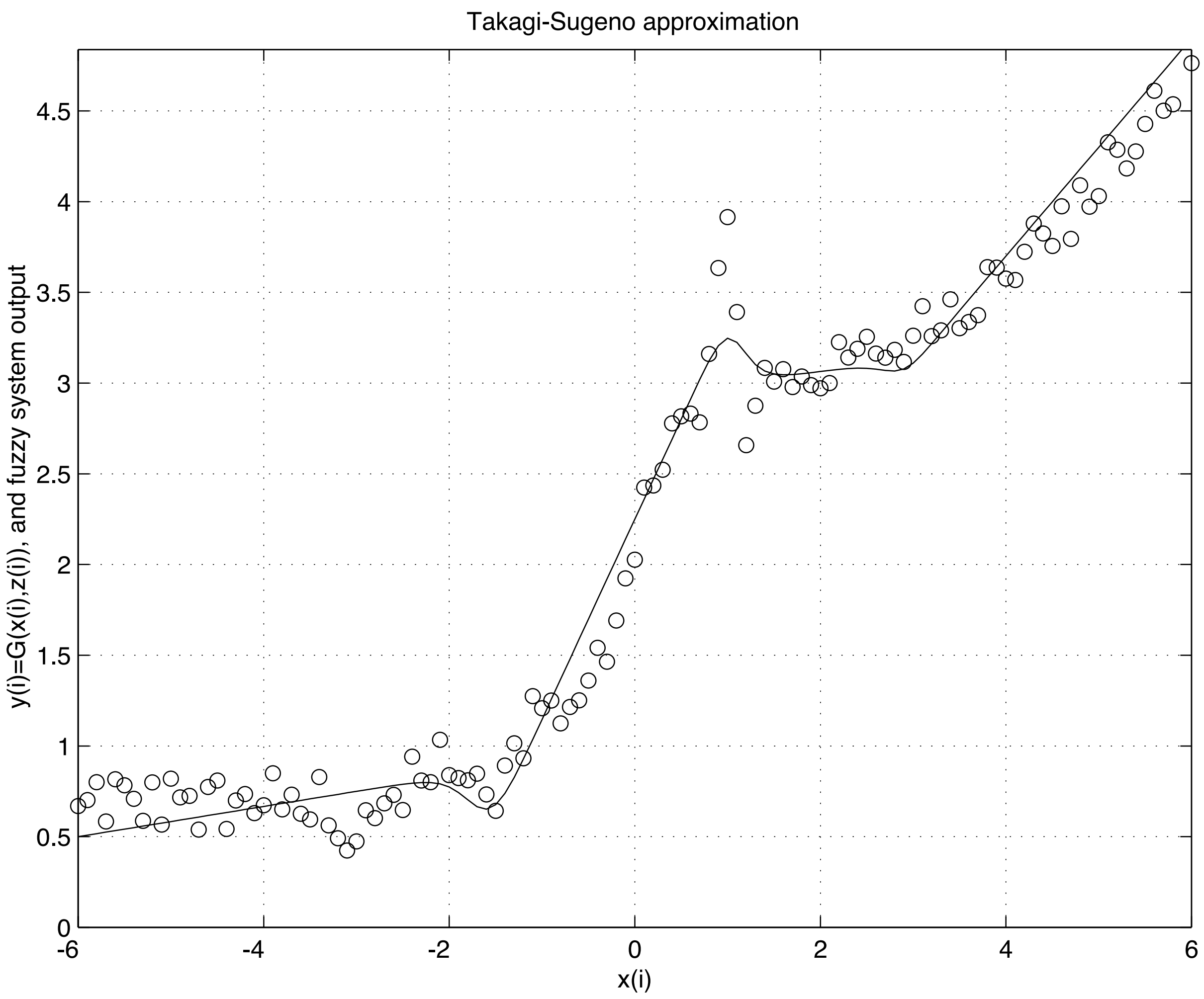

Figure 134: Training data and Takagi-Sugeno fuzzy system approximator.

Properties of Approximators

Universal Approximation Property

- $\mathcal{F}$ – set of all possible approximator structures of type $F(x, \theta)$.
- Example: If $F(x, \theta)$ is a multilayer perceptron with a single hidden layer, then $\mathcal{F}$ would contain an infinite number of approximator structures, each one with different interconnections, nonlinear activation functions, and numbers of neurons in the various layers and values for the tunable parameters θ.
- Assume for now that $n_z = 0$ so no z.

➙ If $G(x)$ is any real valued continuous function defined on a closed and bounded set $X \subset \Re^n$ and for an arbitrary $\epsilon > 0$, there exists an approximator structure $F(x, \theta) \in \mathcal{F}$ such that

$$\sup_{x \in X} |G(x) - F(x, \theta)| < \epsilon$$

then the approximator structure $F(x, \theta)$ is said to satisfy the "universal approximation property."

- Using the Stone-Weierstrass theorem it is easy to show that multilayer perceptrons, radial basis function neural networks, and standard and Takagi-Sugeno fuzzy systems all satisfy the universal approximation property.
- Clearly, however, the linear approximator structure $F_l(x, \theta)$ does not satisfy the universal approximation property.
- Satisfaction of the universal approximation property guarantees that there exists a way to define the particular approximator structure $F(x, \theta)$ and its parameters θ to represent the unknown nonlinearity as accurately as you would like.
- It does *not* say how to find the particular $F(x, \theta)$, or even what p is.

Approximator Complexity

➙ Approximator complexity refers to the complexity of implementing the approximator structure.

> Clearly, the best approximator is a "flexible" one that can be tuned to accurately approximate many types of nonlinear functions (e.g., one that satsifies the universal approximation property) yet only requires minimal memory and processing time to implement on a digital computer.

- Often, complexity analysis can lead you to a quantification of the trade-off between performance and complexity, leading you to conclude that performance costs money (not a surprising conclusion).
- Complexity and neural and fuzzy approximator structures :
  - For multilayer perceptrons be careful in adding layers to the network since more than two layers may not add too much

tuning flexibility, but it certainly adds more complexity.

– Be careful with creating grids of, for example, membership functions of a fuzzy system (or receptive field units of radial basis function neural networks), on the input space since a grid with N membership functions on each of the n input dimensions results in N^n membership functions (i.e., we get an exponential growth in approximator complexity).

Linear or Nonlinear in the Parameter Approximators?

- Results from approximation theory (Barron's) indicate that it is desirable to use approximators that are nonlinear in their parameters since a nonlinear in the parameters approximator can be simpler than a linear in the parameters one (in terms of the size of its structure and hence number of parameters) yet achieve the same approximation accuracy (i.e., the same W).

- The general problem: We know how to tune linear in the parameter approximators, but in certain cases they may not be able to reduce approximation error very well, and we do not know as much about how to tune nonlinear in the parameter approximators, but we know that if can tune them properly we can definitely reduce the approximation error.

  Finding the best approximator structure is a difficult problem that normally requires trial-and-error in practical applications.

On-Line Function Approximation: Dynamic Learning

- Assume that each experiment is performed at the click of a clock, $k = 0, 1, 2, \ldots$ and we get an infinite sequence of training data pairs
$$(x(k), y(k))$$
$k = 0, 1, 2, \ldots$ (notice that we switch the index of the training data to the time index k).

- Suppose that each time we get a new training data pair we want to update the parameter vector $\theta(k)$ of the approximator $F(x, \theta(k))$. This will result in $F(x, \theta(k))$ being a time-varying nonlinear function that is searching for an appropriate shape.

- This "on-line function approximation" scheme is shown in Figure 135 where we illustrate the tuning of $\theta(k)$ (illustrated via the diagonal arrow through the box containing the approximator) using $(x(k), y(k))$ information.

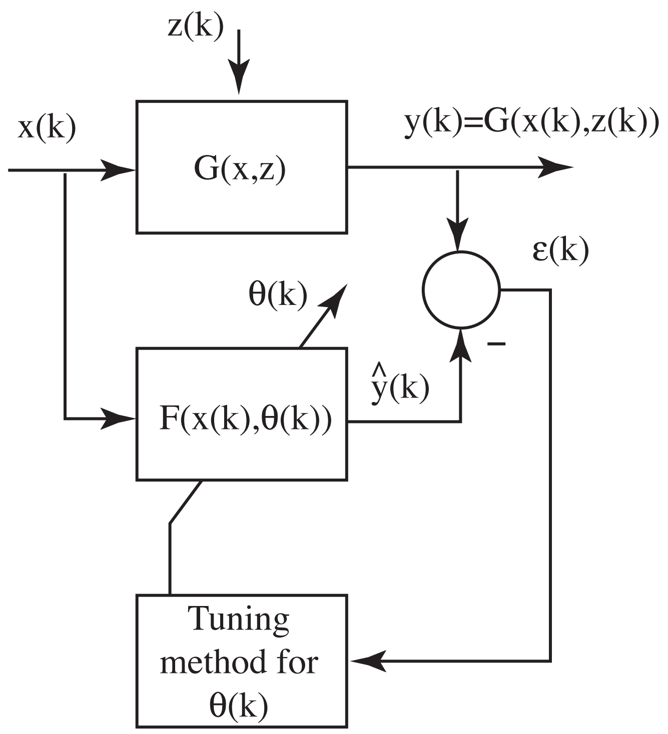

Figure 135: On-line function approximation scheme.

➙ The goal is to get $\epsilon(k) = y(k) - \hat{y}(k) \rightarrow 0$ as $k \rightarrow \infty$.

Linear Least Squares Methods

➙ BLS, RLS for tuning $p \times 1$ vector θ for the linear in the parameters approximator

$$F_{lip}(x, \theta) = \theta^\top \phi(x)$$

where $\phi(x)$ is a known specified $p \times 1$ vector function.

- Use $G = \{(x(i), y(i)) : i = 1, 2, \ldots, M\}$.

➙ Many types of "linear in the parameter" approximators.

Batch Least Squares

Batch Least Squares Derivation

- Define

$$Y(M) = [y(1), y(2), \ldots, y(M)]^\top$$

to be an $M \times 1$ vector of the $y(i)$, $i = 1, 2, \ldots, M$ from G (i.e., $y(i)$ such that $(x(i), y(i)) \in G$).

- We let

$$\Phi(M) = \left[\begin{array}{c} \phi^\top(x(1)) \\ \phi^\top(x(2)) \\ \vdots \\ \phi^\top(x(M)) \end{array} \right]$$

be an $M \times p$ matrix constructed by stacking the $1 \times p$ $\phi^\top(x(i))$ vectors into a matrix (i.e., the $x(i)$ are such that $(x(i), y(i)) \in G$).

- Let $\epsilon(i) = y(i) - F_{lip}(x(i), \theta) = y(i) - \theta^\top \phi(x(i))$ which is the same as

$$\epsilon(i) = y(i) - \phi^\top(x(i))\theta$$

be the error in approximating the data pair $(x(i), y(i)) \in G$ where θ is used in the approximator.

- Define

$$E(M) = [\epsilon(1), \epsilon(2), \ldots, \epsilon(M)]^\top$$

so that

$$E = Y - \Phi\theta$$

➙ Choose

$$J(\theta, G) = \frac{1}{2} E^\top E$$

(a measure of approximation quality for all the data in G for a given θ).

- $J(\theta, G)$ is the sum of the squares of the errors in approximation for each of the training data pairs.

➙ LS: Pick θ to minimize $J(\theta, G)$ ("least squares")

- "Linear" least squares since approximator is linear in the parameters.

➙ $J(\theta, G)$ is convex in $\theta \rightarrow$ local minimum is a global minimum.

- ➞ Calculus: Take partial derivative of J with respect to θ and set it equal to zero, we get an equation for θ, the best estimate of the unknown θ^*.
- ➞ Another approach:

$$2J = E^\top E = Y^\top Y - Y^\top \Phi\theta - \theta^\top \Phi^\top Y + \theta^\top \Phi^\top \Phi\theta$$

- "Complete the square" by assuming that $\Phi^\top\Phi$ is invertible and

$$\begin{aligned} 2J &= Y^\top Y - Y^\top \Phi\theta - \theta^\top \Phi^\top Y + \theta^\top \Phi^\top \Phi\theta \\ &+ Y^\top \Phi(\Phi^\top\Phi)^{-1}\Phi^\top Y - Y^\top \Phi(\Phi^\top\Phi)^{-1}\Phi^\top Y \end{aligned}$$

(add and subtract the same terms at the end).

- Hence,

$$\begin{aligned} 2J &= Y^\top (I - \Phi(\Phi^\top\Phi)^{-1}\Phi^\top)Y \\ &+(\theta - (\Phi^\top\Phi)^{-1}\Phi^\top Y)^\top \Phi^\top\Phi(\theta - (\Phi^\top\Phi)^{-1}\Phi^\top Y) \end{aligned} \quad (35)$$

- The first term in this equation is independent of $\theta \longrightarrow$ ignore.
- Choose θ so that the second term is zero.
- Denote the value of the parameters that achieves the minimization of J by θ

$$\theta = (\Phi^\top \Phi)^{-1}\Phi^\top Y \tag{36}$$

➙ "Batch" least squares.

- If pick inputs to the system so that it is "sufficiently excited", then we will be guaranteed that $\Phi^\top \Phi$ is invertible.

➙ In "weighted" batch least squares we use

$$J(\theta, G) = \frac{1}{2}E^\top W E \tag{37}$$

where, for example, W is an $M \times M$ diagonal matrix with its diagonal elements $w_i > 0$ for $i = 1, 2, \ldots, M$.

- The w_i can be used to weight the importance of certain elements of G more than others.
- We may choose to have it put less emphasis on older data by choosing $w_1 < w_2 < \cdots < w_M$ when $x(2)$ is collected after $x(1)$, $x(3)$ is collected after $x(2)$, and so on.

➞ One approach: $0 < \lambda \leq 1$, then let $w_i = \lambda^{M-i}$, $i = 1, 2, \ldots, M$.

- Can show that:

$$\theta_{wbls} = (\Phi^\top W \Phi)^{-1} \Phi^\top W Y \tag{38}$$

Example: Off-Line Tuning of Neural Networks

- Train MLP to match the training data shown in Figure 130 (this defines G and in our case we have $M = 121$).

Improved Accuracy Over Manual Tuning:

- Recall that we were using the perceptron with a single hidden layer shown in Figure 131 with $n_1 = 2$ neurons in the hidden layer.

➙ Have:

$$y = F_{mlp}(x, \theta) = \theta^\top \phi(x) = [w_1, w_2, b][\phi_1(x), \phi_2(x), 1]^\top$$

where via our heuristic approach we used $f(\bar{x}) = \frac{1}{1+\exp(-\bar{x})}$ and had chosen

$$\phi_1(x) = f(b_1 + w_{1,1} x)$$

with $b_1 = 0$ and $w_{1,1} = 1.5$, and

$$\phi_2(x) = f(b_2 + w_{1,2}x)$$

with $b_2 = -6$ and $w_{1,2} = 1.25$.

- We had chosen $\theta = [3, 1, 0.6]^\top$.

➙ We will use the batch least squares approach to see how if it can pick a better θ.

★ Form Y and Φ, use BLS formula to get

$$\theta = [2.5747, 1.6101, 0.7071]^\top$$

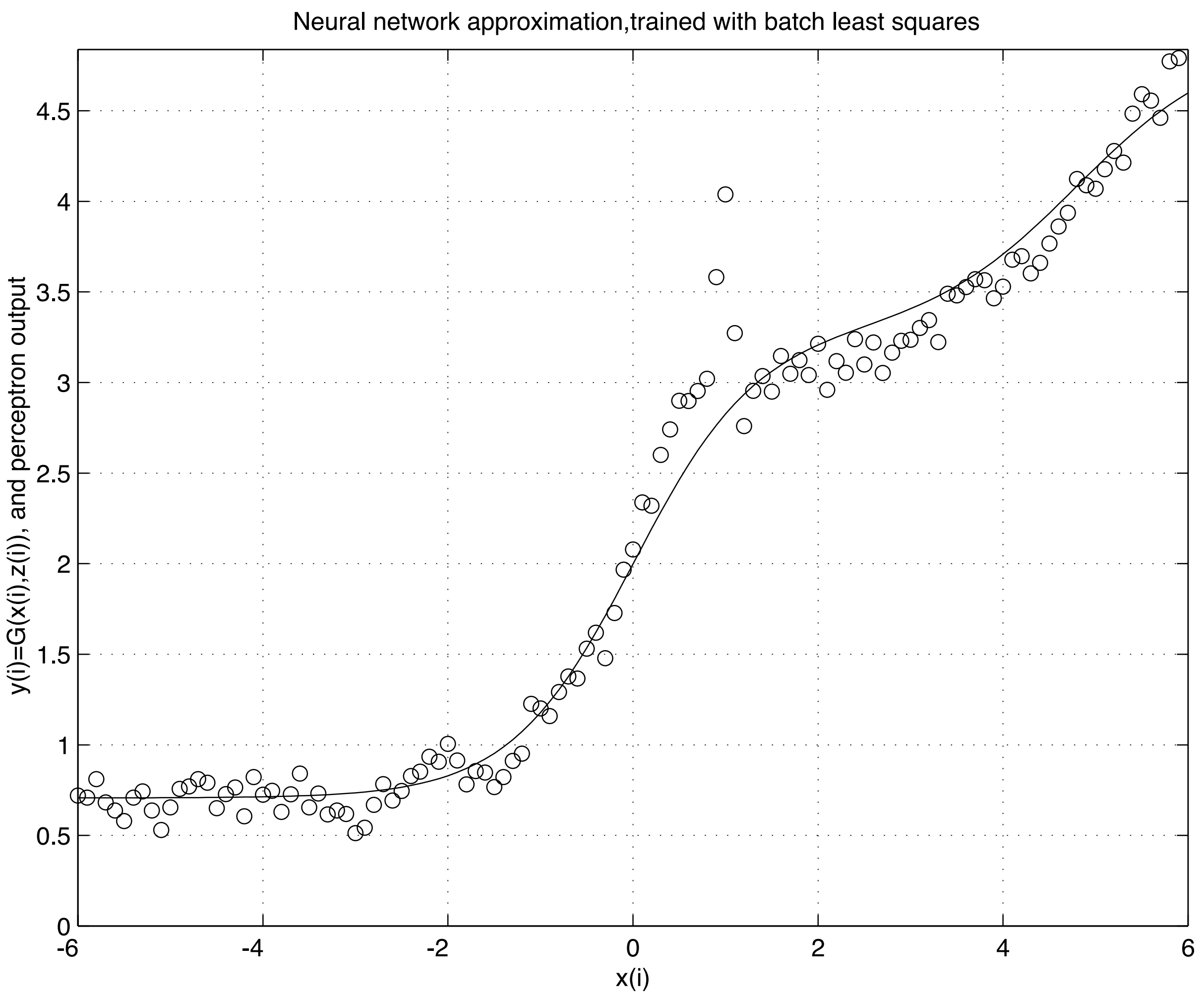

Figure 136: Multilayer perceptron approximator trained with batch least squares, 2 neurons.

Increasing the Number of Hidden Neurons:

➙ Generally, you want to use much more training data than parameters (to avoid what is called "overfitting" below) so since we use $M = 121$ we will now consider $n_1 = 11$ neurons in the hidden layer (for a total of $11(2) + 11 + 1 = 34$ parameters).

- We need a scheme to pick the weights and biases of the hidden layer. Use a simple heuristic approach.

➙ Biases: Pick evenly spaced over -5 to 5, so that $b_1 = -5$, $b_2 = -4$, to $b_{11} = 5$.

- This should help spread the points where the activation functions turn on across the input space.
- The choice for the w^j, $j = 1, 2, \ldots, 11$, is more difficult if you take the view that we did in the manual tuning of the perceptron.

- Notice that there we assumed that we could examine the training data and pick off slopes to set these values.

➙ This is often unrealistic for complex real world problems.

- Here, we will exploit the fact that the scaling factors in w are used to modify the slopes to what we will need, so we simply pick $w^j = 1$, $j = 1, 2, \ldots, 11$ (for applications where $n > 1$ this scheme may not be as effective and in those cases you will want the weights to take on values that will allow for a range of slopes).

- This completes the specification of the hidden layer.

★ BLS to tune the 12 parameters in $\theta = [w^\top, b]^\top$:

$$\begin{aligned} \theta \quad = \quad & [2.7480, 2.0120, -11.9865, 34.7556, \\ & -69.6968, 93.4042, -80.8496, 57.0819, \\ & -34.6710, 15.8048, -3.9398, 0.8087]^\top \end{aligned} \tag{39}$$

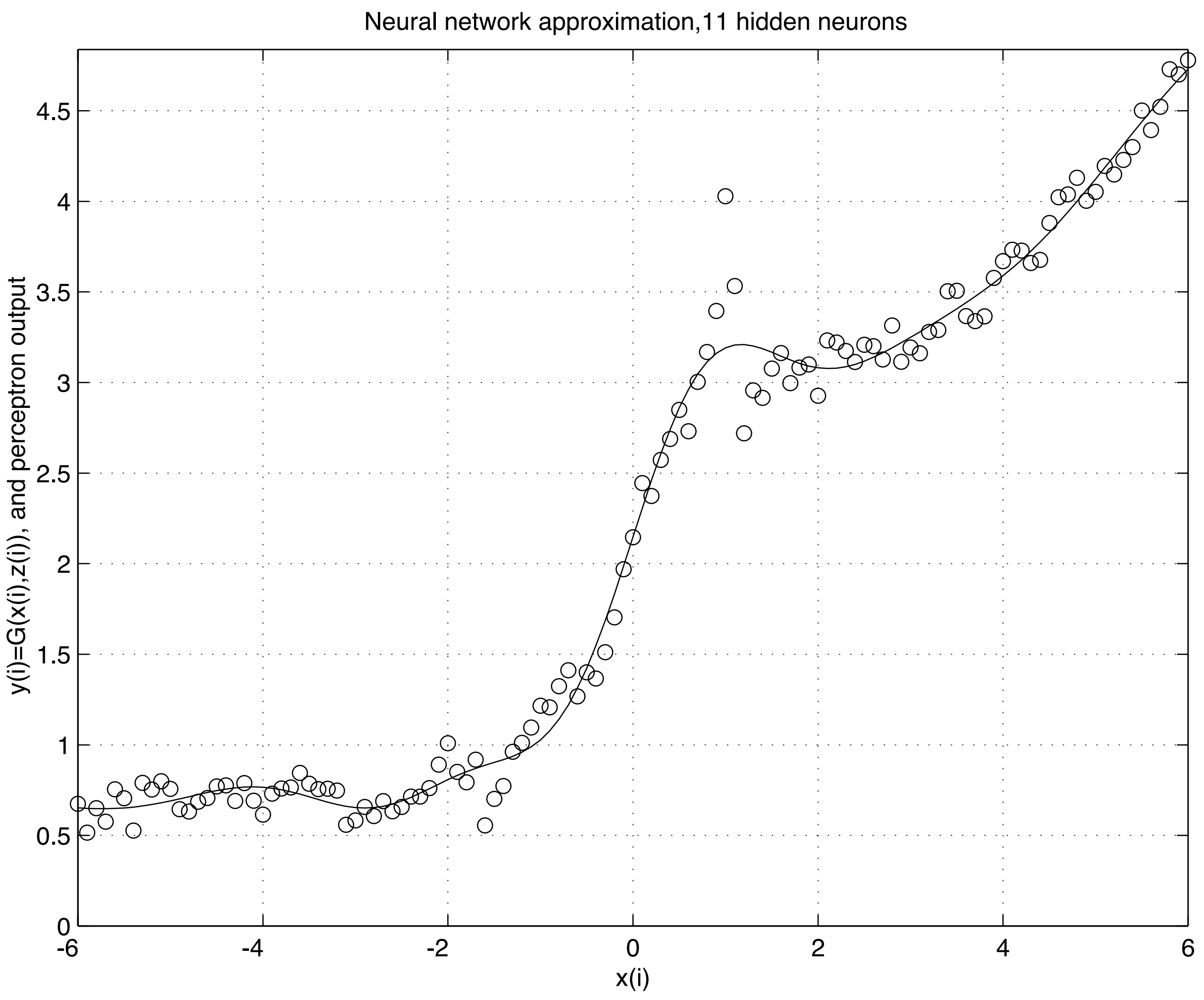

Figure 137: Multilayer perceptron approximator trained with batch least squares, 11 neurons.

★ Signficant improvement over Figure 136 where we used $n_1 = 2$ neurons in the hidden layer and Figure 132 where we tuned the approximator manually.

➙ Vector θ has both positive and negative values (positive and negative slopes).

➙ Clearly it would be quite difficult to tune the approximator manually to get this kind of accuracy.

Fine-Tuning to Capture High Frequency Behavior:

➙ More parameters? Use $n_1 = 25$ neurons (to get a total of $25(2) + 25 + 1 = 76$ parameters).

- Choose the biases in a similar fashion—over the whole range -6 to 6 to get $b_1 = -6$, $b_2 = -5.5$, to $b_{25} = 6$.
- As above, we pick all the weights in the hidden layer to be unity.

- We use batch least squares to tune the 26 parameters in $\theta = [w^\top, b]^\top$.

★ For this case we get the approximation shown in Figure 138 which is an improvement over Figure 137 where we used $n_1 = 11$ neurons (notice that the approximator is starting to find some of the structure of the underlying function that is illustrated in Figure 129; least squares is particularly good at finding this structure, in this case, due to how the noise on z enters).

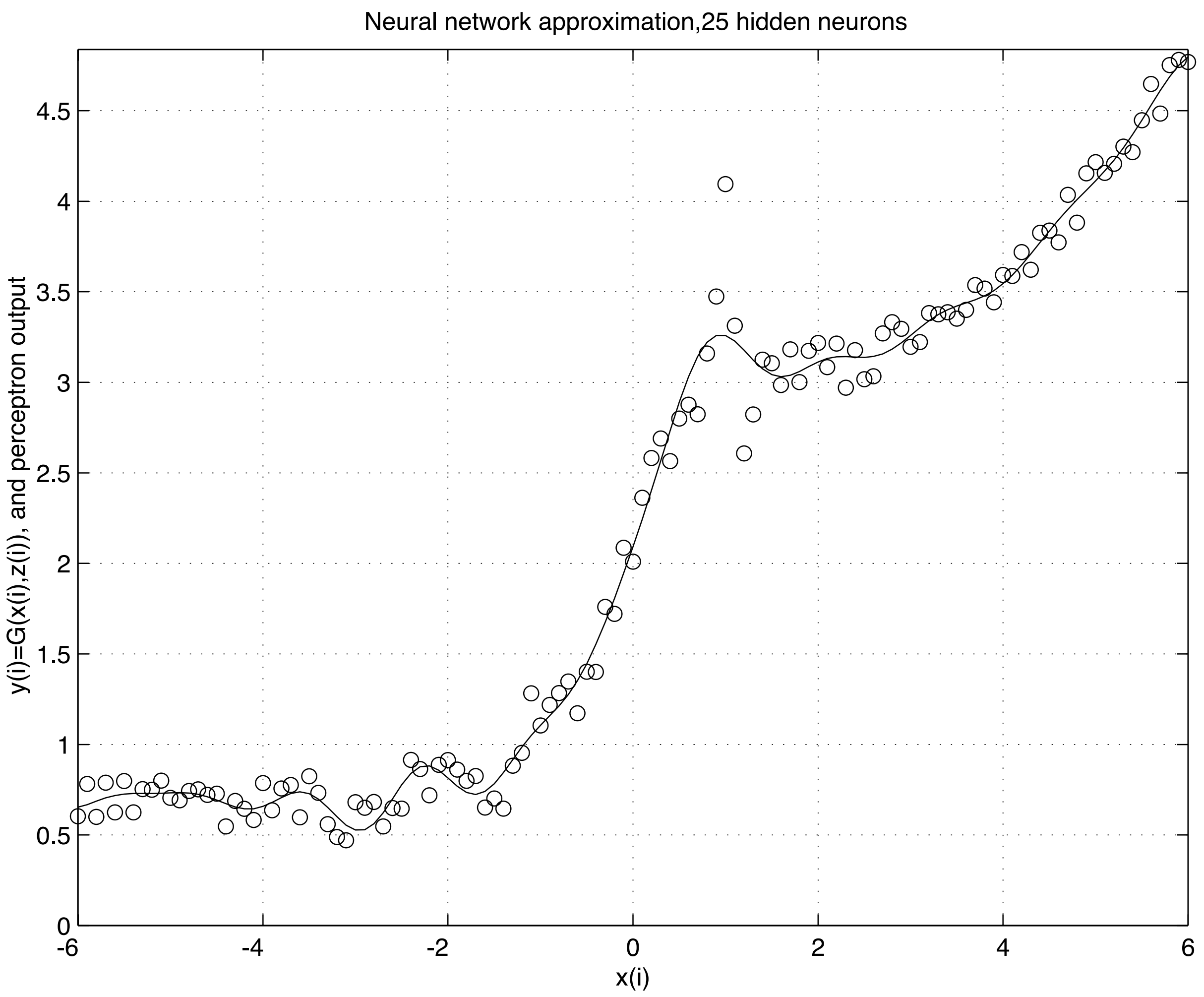

Figure 138: Multilayer perceptron approximator trained with batch least squares, 25 neurons.

★ With more neurons able to approximate more of the "high frequency" behavior in the function.

Overfitting Where the Approximator Seeks to Model Noise:

★ Increasing n_1 can be taken too far.

- Suppose that we choose $n_1 = 121$ (for a total of $121(2) + 121 + 1 = 364$ parameters), $b_1 = -6$, $b_2 = -5.9$, to $b_{121} = 6$, and the weights in the hidden layer as all unity.
- We have θ as an 122×1 vector so $p > M$.

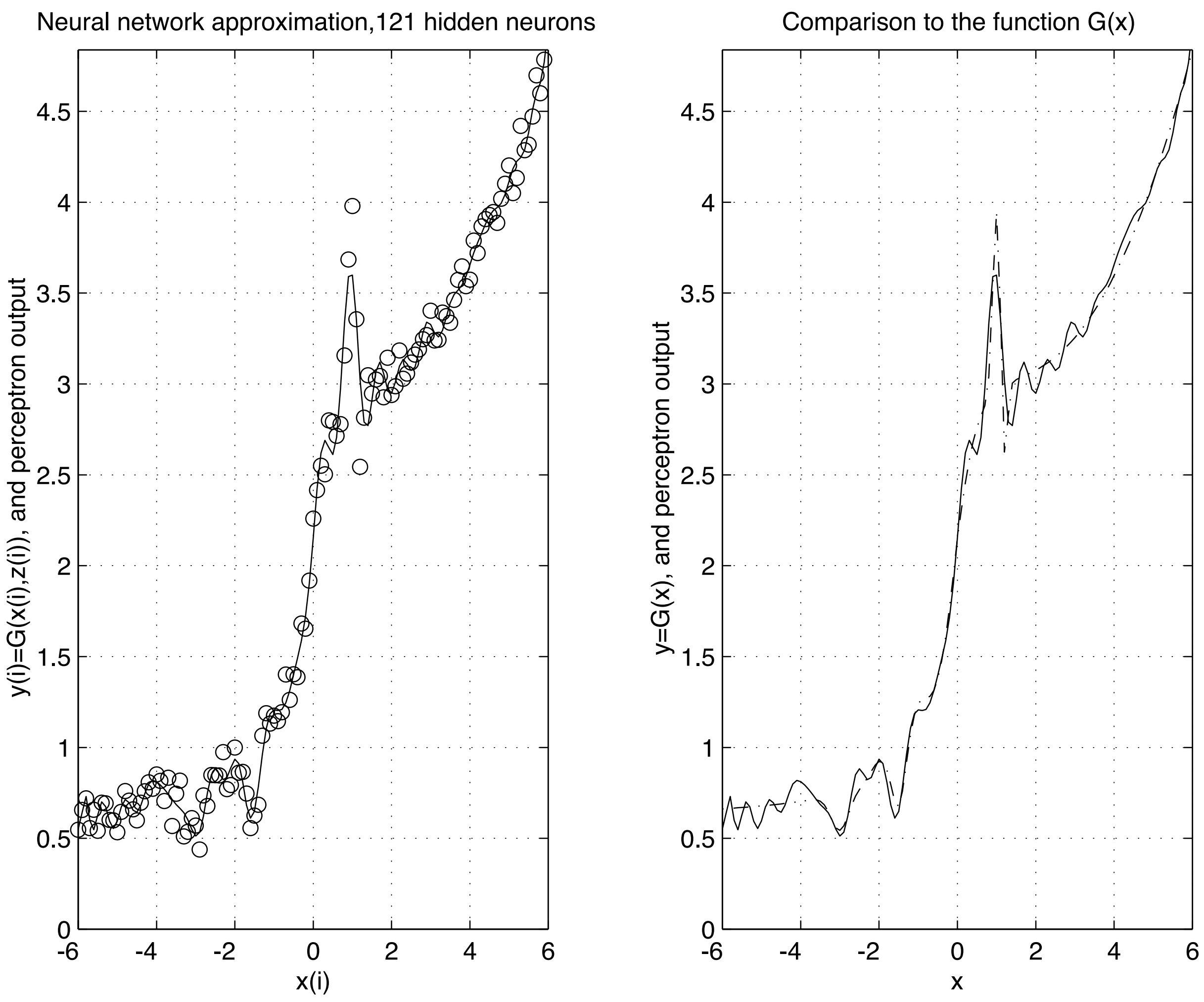

Figure 139: Multilayer perceptron approximator trained with batch least squares, 121 neurons, plus comparison to $G(x)$.

★ Notice that in this plot we have also plotted approximator nonlinearity on top of the function $G(x)$ (i.e., where we have removed the effects of the noise z).

$$\boxed{\text{Important fact: if you use too many parameters you may start trying to approximate characteristics of the noise, and not the underlying function.}}$$

Recursive Least Squares

- ➙ Recursive version of BLS will allow us to update our θ estimate each time we get a new data pair, without using all the old data and without computing $(\Phi^\top \Phi)^{-1}$.

  - This "(weighted) recursive least squares" approach allows us to implement an on-line function approximator.

- ➙ Suppose that the parameters of the physical system vary slowly.

- ➙ Consider minimizing

$$J(\theta, G)|_{M=k} = \frac{1}{2} \sum_{i=1}^{k} \lambda^{k-i} (y(i) - \phi^\top (x(i))\theta)^2$$

where $0 < \lambda \leq 1$ is called a "forgetting factor"—it gives the more recent data higher weight in the optimization (consider effect of λ^{k-i} in above summation).

➙ The equations for WRLS are given by

$$\begin{aligned} \theta(k) &= \theta(k-1) + K(k)\left(y(k) - \phi^\top(x(k))\theta(k-1)\right) \quad (40) \\ K(k) &= \frac{P(k-1)\phi(x(k))}{\lambda + \phi^\top(x(k))P(k-1)\phi(x(k))} \\ P(k) &= \frac{1}{\lambda}\left(I - K(k)\phi^\top(x(k))\right)P(k-1) \end{aligned}$$

(where when $\lambda = 1$ we get the equation for standard RLS).

- We need to initialize the WRLS algorithm (i.e., choose $\theta(0)$ and $P(0)$).

- One approach to do this is to use $\theta(0) = 0$ and $P(0) = P_0$ where $P_0 = \alpha I$ for some large $\alpha > 0$.

➙ This is the choice that is often used in practice.

- Other times, you may pick $P(0) = P_0$ but choose $\theta(0)$ to be a best guess.

➙ In practice "covariance modifications" (e.g., covariance resetting) are used.

Example: Fitting a Line to Data Generated by a Time-Varying Function

- For $n = 1$

$$y = F_{lip}(x,\theta) = \theta^\top \phi(x) = \theta^\top [\phi_1(x), 1]^\top = \theta^\top [x_1, 1]^\top = \theta_1 x_1 + \theta_2 \tag{41}$$

➙ Unknown function:

$$y(k) = G(x(k), z(k)) = \sin(0.01k)x_1(k) + 1$$

where $x(k) = x_1(k)$ and $z(k)$ captures the time-varying nature of the function (e.g., we could say that $z(k) = k$).

- Think of $\sin(0.01k)$ as a time-varying slope and the 1 as the intercept.
- Choose $x(k)$ uniformly distributed rv on $[-1, 1]$.
- Pick $\theta(0) = [0, 0]^\top$ and $P(0) = \alpha I$ where $\alpha = 100$.

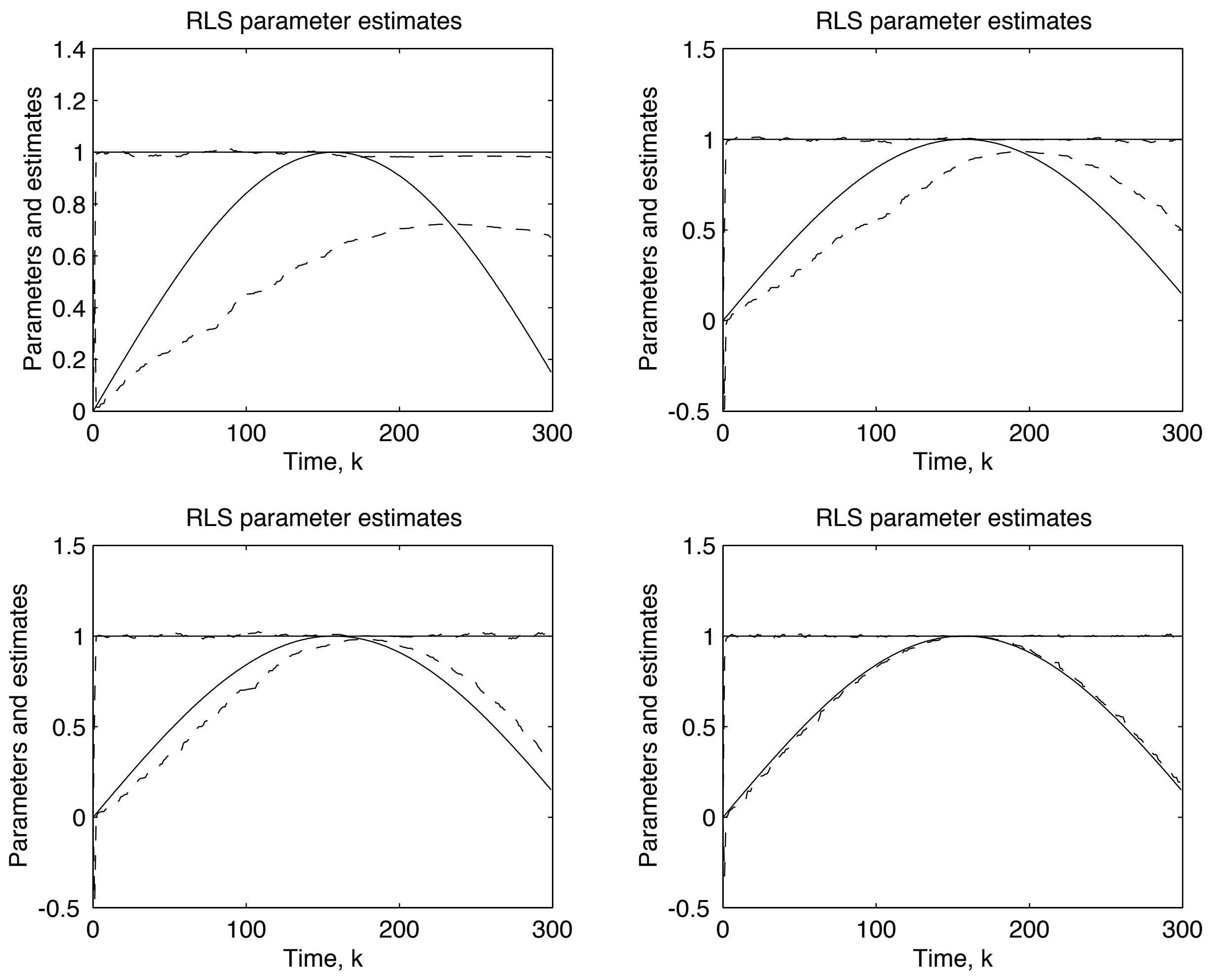

Figure 140: RLS parameter estimates, $\lambda = 1$ (upper left), $\lambda = 0.98$ (upper right), $\lambda = 0.95$ (lower left), $\lambda = 0.7$ (lower right).

- $\lambda = 1$: θ_1 quickly converges to the true value of 1 but that the estimate θ_2 is quite poor (since it does not forget old information).
- $\lambda = 0.98$: Does much better.
- $\lambda = 0.7$: Get a very good estimate.
- But, if too small forgets too much $\rightarrow$ bad performance.

Example: On-Line Tuning of Neural and Fuzzy Systems

Multilayer Perceptrons

- Use $n_1 = 25$ neurons and use all the same values of the biases and weights in the hidden layer that we developed earlier
- Tune weights and bias in output layer (26×1 vector θ).

Relatively Uniform Coverage of the Input Space

- Let the input x be uniformly distributed on $[-6, 6]$ for our theme problem.
- Let $\lambda = 1$ and intialize the algorithm with $\theta(0) = 0$ and $P(0) = \alpha I$ with $\alpha = 100$.

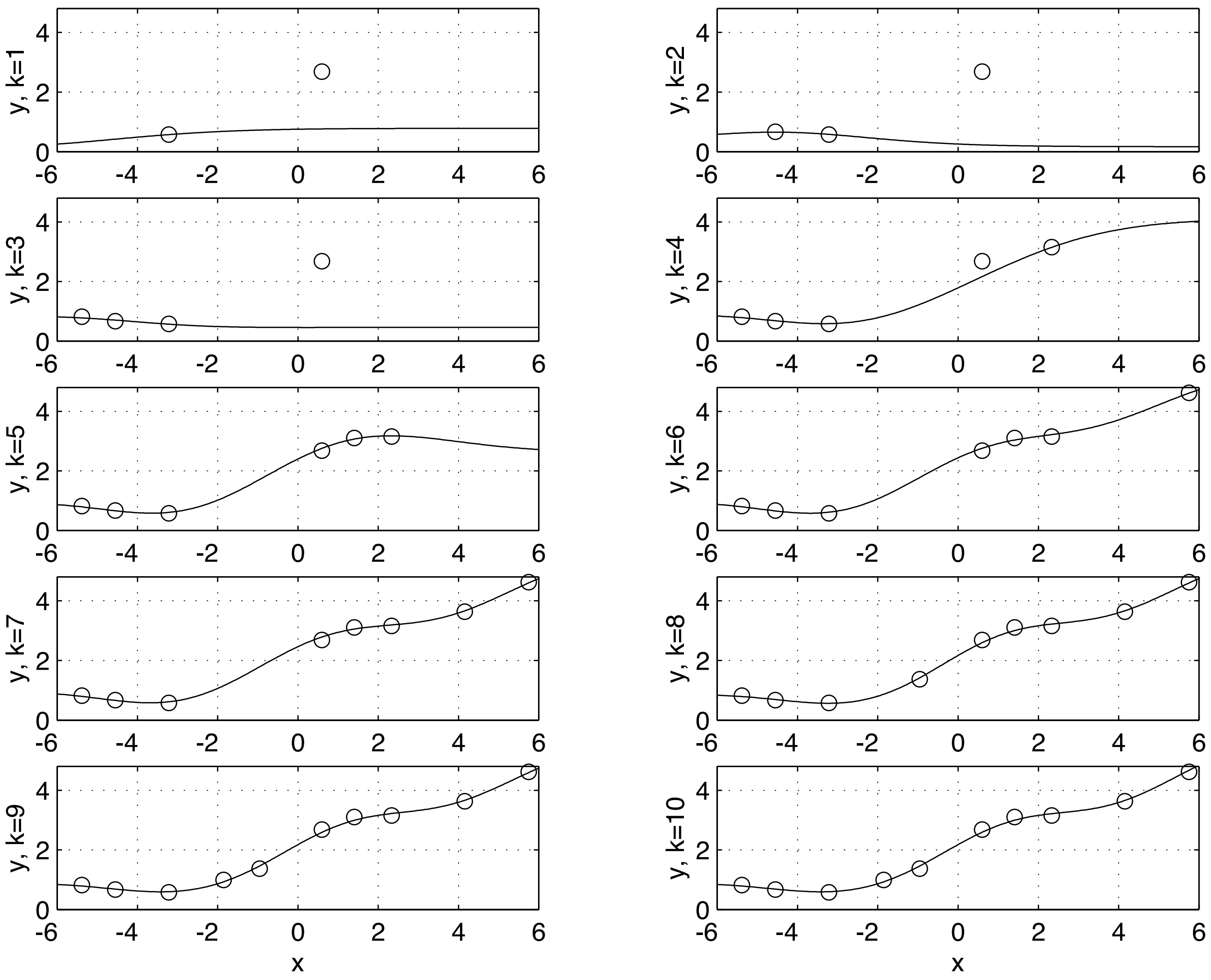

Figure 141: Neural network mappings generated using increasing amounts of training data ($k = 1$ to $k = 10$).

- For $k = 1$ we have two data points.

★ As k increases we get more and more training data and our representation becomes more and more accurate.

Nonuniform Coverage of the Input Space

➙ By random chance it does not place any x data in one region of the input space.

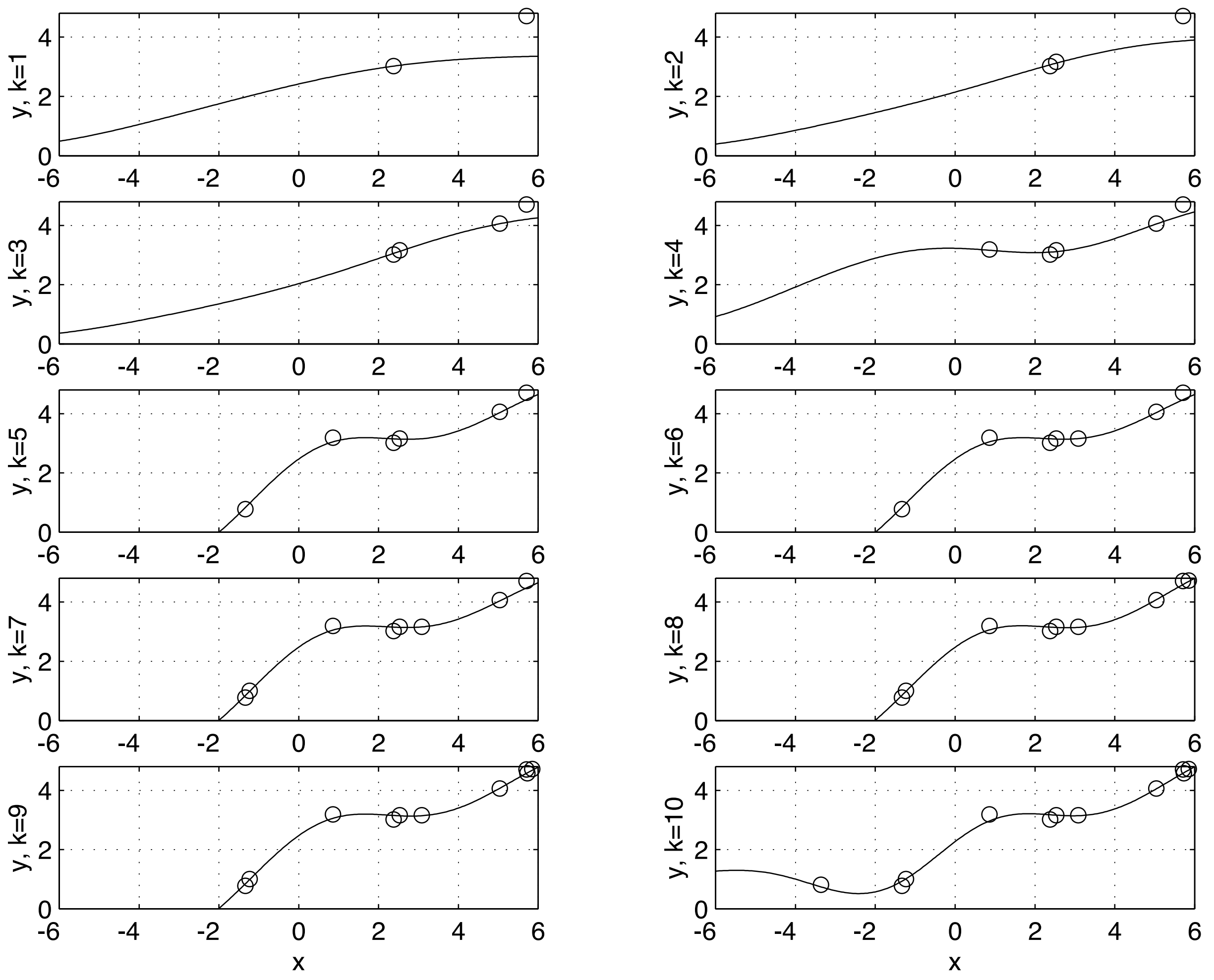

Figure 142: Neural network mappings generated using increasing amounts of training data (no data near $x = -6$ for $k \leq 9$).

➙ The approximator is *extrapolating*

★ If there are “holes” in the input space where there is no data we will generally get poor approximation accuracy in that region.

★ Initialization: Estimation quality generally proportional to initialization quality.

Takagi-Sugeno Fuzzy Systems

- ➙ Due to locality properties of the Takagi-Sugeno fuzzy system it tends to adjust the mapping only where it gets data and tends not to destroy what it has learned in one area when making adjustments in another area.
- ★ Other principles the same as for the neural network case.

Gradient Methods

➙ Gradient techniques offer practical and effective methods to perform optimization.

- Consider minimizing

$$J(\theta, G) = \frac{1}{2}\sum_{i=1}^{M} |y(i) - F(x(i), \theta)|^2 \tag{42}$$

by the choice of θ for a given training data set G (note that in several cases below we will develop the theory for the case where $F(x(i), \theta)$ and $y(i)$ are $\bar{N} \times 1$ vectors).

➙ $J(\theta, G)$ can be a very complex "landscape" as in Figure 143.

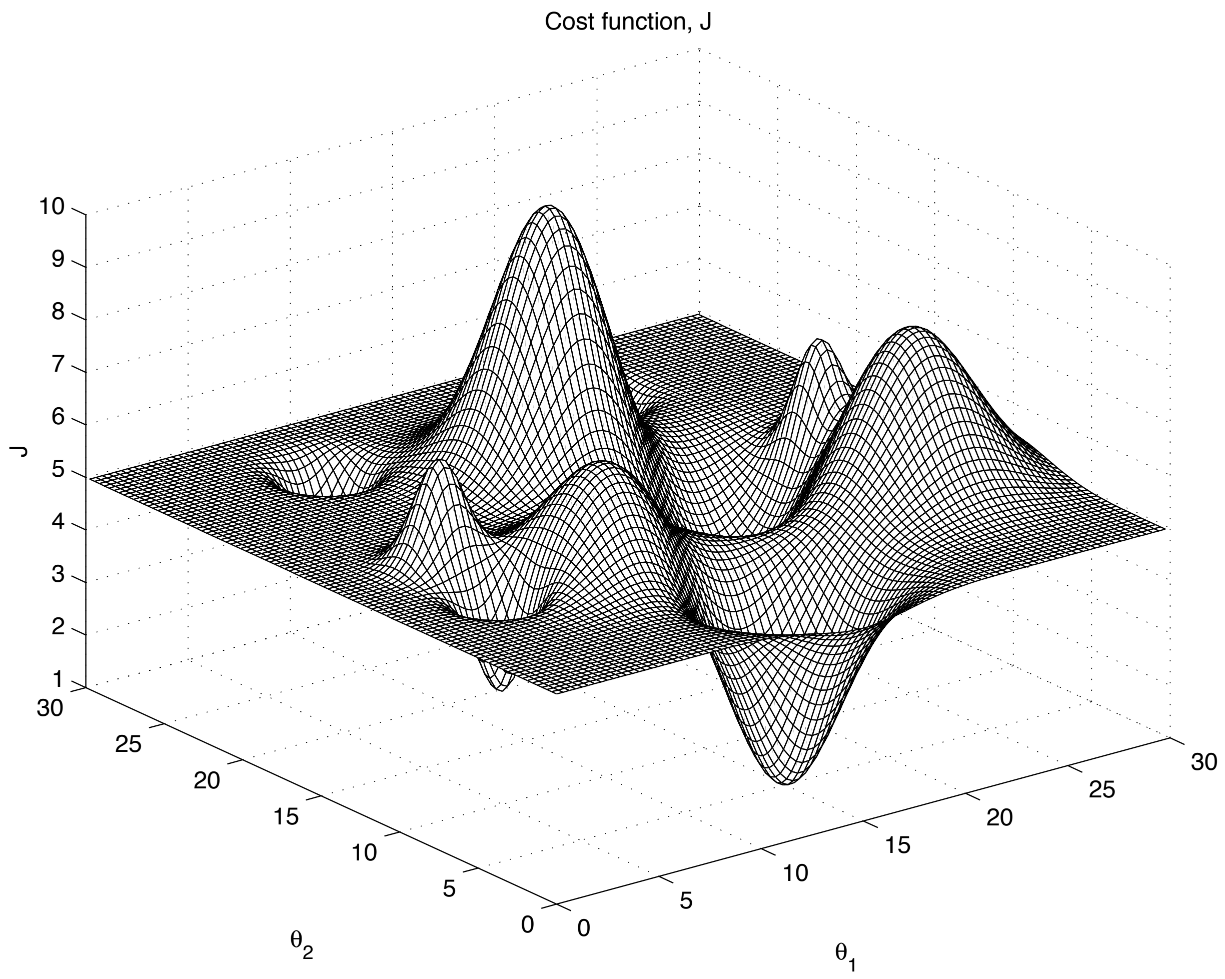

Figure 143: Candidate function for which we may seek to find the minimum.

- Flat regions and local minima complicate the problem.

➙ Will we end at a local or global minimum?

- The most we will be able to hope for is to converge to a "stationary point," that is, a zero slope region.

The Steepest Descent Method

- Let $\theta(j)$ be the current estimate of the parameter vector at iteration j (note that when we indexed θ with time we used k, but here j is used to emphasize that the index may simply be for the training data, not time).

  Steepest Descent Parameter Updates

- The basic form of the update using a gradient method to minimize the function $J(\theta, G)$ via the choice of $\theta(j)$ is

$$\theta(j+1) = \theta(j) + \lambda_j d(j) \tag{43}$$

  where $d(j)$ is the $p \times 1$ "descent direction," and $\lambda_j > 0$ is a (scalar) positive "step size" that can depend on the iteration number j.

- To see the rationale, let θ be a scalar and use the simple quadratic function
$$J(\theta, G) = \theta^2$$
in Figure 144 where we are searching for the point where the function reaches the minimum by picking the scalar θ.

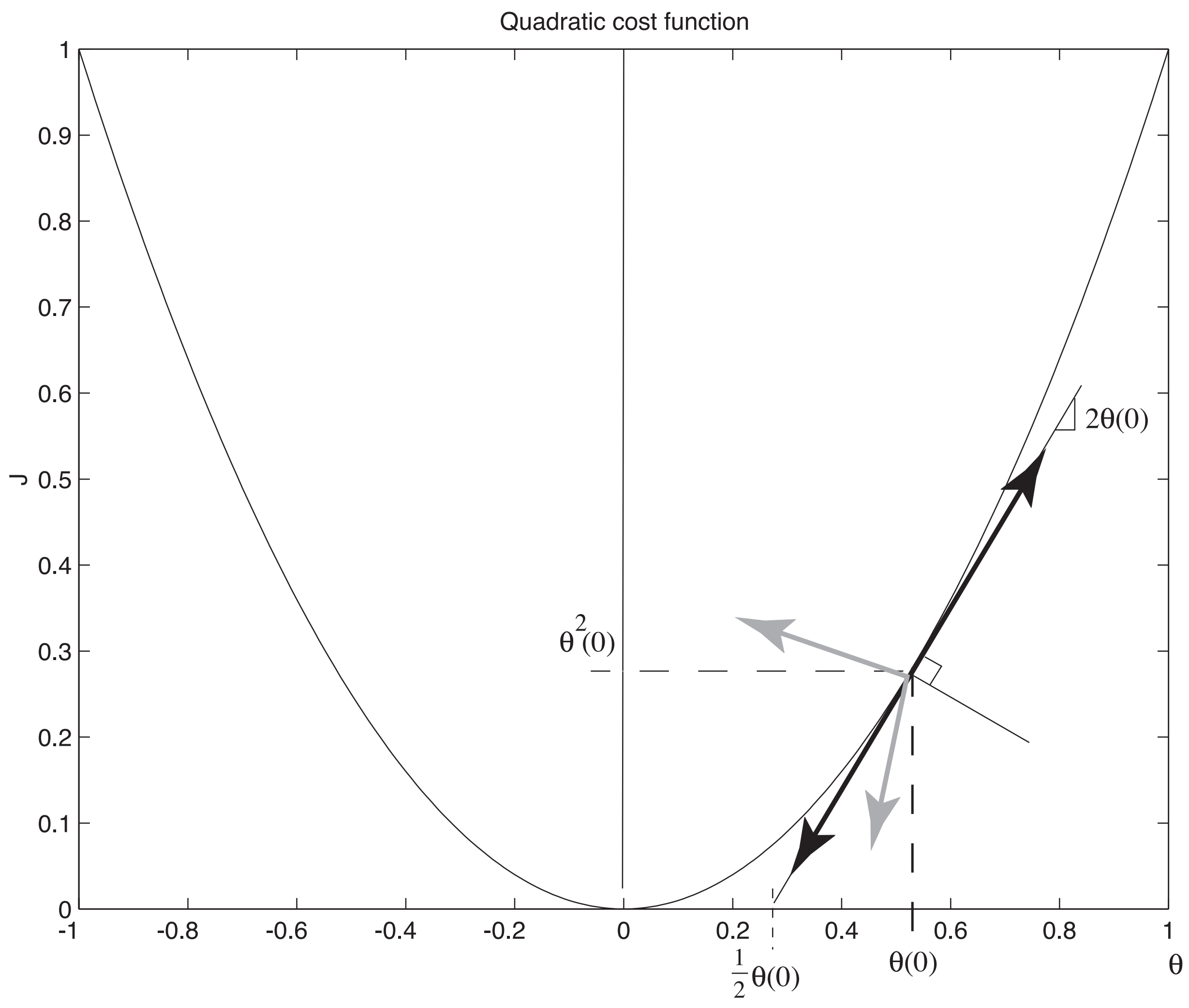

Figure 144: Scalar quadratic $J(\theta, G)$.

- Name the point where the minimum is achieved θ^* and assume that it is unknown and that we want to find its value.
- The update formula for this scalar case, where $\lambda_j = \lambda$ is a positive constant, is

$$\theta(j+1) = \theta(j) + \lambda d(j)$$

- Notice that

$$d(j) = \frac{\theta(j+1) - \theta(j)}{\lambda} \tag{44}$$

- With λ as a step size we see that $d(j)$ is a descent direction in the sense that it is the direction in which the parameter is moving in order to try to minimize $J(\theta, G)$.

➙ What direction would we like this to be?

➙ We would like the parameter updates to always move in a direction that decreases $J(\theta, G)$ because if it does this over a whole sequence of iterations (perhaps an infinite number of iterations) we may get $\theta(j) \rightarrow \theta^*$ as $j \rightarrow \infty$ (so $J(\theta^*, G) \leq J(\theta, G)$ for all other possible θ).

- The above formula (Equation (44)) suggests that the direction should be the slope of the function $J(\theta, G)$ at $\theta = \theta(j)$.
- To see this, consider the example in Figure 144.
- Suppose that the initial (best) guess of θ^* is the $\theta(0)$ shown.
- Based on this guess, how would you next guess at θ?
- That is, how would you generate $\theta(1)$?
- Suppose that we can compute the slope (gradient) of $J(\theta, G)$ at $\theta(0)$.

- For our example this gradient is

$$\left.\frac{\partial J(\theta, G)}{\partial \theta}\right|_{\theta=\theta(0)} = 2\theta|_{\theta=\theta(0)} = 2\theta(0) \tag{45}$$

  and it is shown as the black arrow pointing up and to the right in Figure 144.

- The negative of this gradient is $-2\theta(0)$ and it is shown as the black arrow pointing down and to the left in Figure 144.

- These arrows indicate possible directions $d(j)$ to update the guess at $\theta(0)$.

- Clearly, to move *down* the function $J(\theta, G)$ to minimize it one choice would be to use the direction

$$d(j) = -\left.\frac{\partial J(\theta, G)}{\partial \theta}\right|_{\theta=\theta(j)} \tag{46}$$

  (i.e., to move along the negative gradient).

- Intuitively, this choice is the "direction of steepest descent" (it corresponds to how a skier often moves down a snow-covered mountain) and hence the parameter update formula for the steepest descent method, even for the p-dimensional case, is given by

$$\theta(j+1) = \theta(j) - \lambda_j \left. \frac{\partial J(\theta, G)}{\partial \theta} \right|_{\theta=\theta(j)} \tag{47}$$

- Generally, $\theta(j)$ may not have a limit point, it may diverge as in this example, or it may oscillate; however, gradient methods are generally able to find isolated stationary points (i.e., ones where you can draw a sphere around it and no other stationary points are in the sphere), if they start close to them (this is why initialization of the algorithms is so important).

➙ Clearly, the direction of descent and step size are important parameters for gradient methods.

Descent Direction Possibilities and Momentum

- Considering Figure 144, notice that any direction $d(j)$ is a descent direction provided that the angle it makes with

$$\left.\frac{\partial J(\theta, G)}{\partial \theta}\right|_{\theta=\theta(j)}$$

(indicated by the black arrow in Figure 144 that is pointing up and to the right) is more than 90°.

- For our simple example above, this means that the shaded arrows shown in Figure 144 are also descent directions for that $\theta(0)$.

- Mathematically, the angle is greater than 90° if

$$\left(\frac{\partial J(\theta(j), G)}{\partial \theta(j)}\right)^{\top} d(j) < 0$$

- As indicated, this formula also holds for the vector case.

- In the vector case $d(j)$ is a $p \times 1$ vector and the gradient is a $p \times 1$ vector that is denoted by

$$\nabla J(\theta(j), G) = \frac{\partial J(\theta(j), G)}{\partial \theta(j)} = \begin{bmatrix} \frac{\partial J(\theta(j),G)}{\partial \theta_1(j)} \\ \frac{\partial J(\theta(j),G)}{\partial \theta_2(j)} \\ \vdots \\ \frac{\partial J(\theta(j),G)}{\partial \theta_p(j)} \end{bmatrix} \tag{48}$$

- Clearly, the choice of

$$d(j) = -\left.\frac{\partial J(\theta, G)}{\partial \theta}\right|_{\theta=\theta(j)} = -\nabla J(\theta(j), G)$$

  results in the satisfaction of this formula, but clearly many other choices do also.

- One modification to the descent direction that has been found to be useful in some applications is to use a "momentum term."

- In this case the update formula is

$$\theta(j+1) = \theta(j) - \lambda_j \nabla J(\theta(j), G) + \beta\,(\theta(j) - \theta(j-1)) \qquad (49)$$

where $0 \leq \beta < 1$ is a fixed gain and $\beta\,(\theta(j) - \theta(j-1))$ is the momentum term.

➙ Basically, momentum accelerates progress of the update where the gradients $\nabla J(\theta(j), G)$ are pointing in the same direction, but restricts update sizes when successive gradients are roughly opposite in direction.

- This can tend to dampen out oscillations in the parameter vector and keep the parameter vector moving in the proper direction.

Step Size Choice

Constant Step Size

- For some applications (e.g., in adaptive control) a fixed step size $\lambda_j = \lambda$ for all j can be sufficient.

➙ Generally, if λ is too small we get slow convergence, but if it is too large then we can get divergence.

➙ Indeed, in many problems a constant step size can result in "limit cycling" (oscillations in parameter values) near a local minimum since when you are in a region near a local minimum you often must take successively smaller steps to ensure that you do not "overshoot" the solution.

Diminishing Step Size

- In this case, the step size converges to zero as j goes to infinity, according to some formula.

- That is, we pick an algorithm for forcing

$$\lambda_j \to 0$$

as $j \to \infty$.

- For instance, we may choose

$$\lambda_j = \frac{\lambda}{j+1}$$

where $\lambda > 0$ is a constant.

- While this can be simple to implement, in some cases λ_j may be chosen so that it goes to zero too fast so that the algorithm slows prematurely, before it gets near a solution.

- It is for this reason that often it is required that

$$\sum_{j=0}^{\infty} \lambda_j = \infty$$

which, in effect, forces the step size to persistently update the parameters (provide the gradient is sufficiently large).

Parameter Initialization, Constraints, and Update Termination

Parameter Initialization

➙ While in general you do not know where the optimal solution θ^* is, it is of course best to pick $\theta(0)$ as close to this desired value as possible.

- For a multilayer perceptron, if you know nothing, pick small random values (why?).
- For a fuzzy system use a uniform grid for premise membership functions.

Constraints on Parameters

- Generally, there are several reasons why there are constraints on the parameters in an optimization problem like we study:
  1. If the parameters $\theta(j)$ take on certain values, there are numerical problems.

2. Physical constraints.
3. Sometimes we know a "feasible region" for the optimal parameter values and hence searching outside this set is fruitless.
4. Sometimes, we have extra information about the form of the underlying function that we seek to approximate (e.g., by physical insights, or by simple inspection of the training data), and this can be used to constrain the choices of the parameters.

- In any of these three cases the constraints can be captured by requiring that

$$\theta(j) \in \Theta(j)$$

for all $j \geq 0$ where $\Theta(j)$ is the (known) parameter "constraint set" at iteration j.

- Often, we know that $\Theta(j) = \Theta$, that is, that the constraint set is the same at each iteration.

➙ How do we ensure that $\theta(j) \in \Theta$ for all $j \geq 0$? "Projection"

- First, we initialize so that $\theta(0) \in \Theta$ so that all we need to concern ourselves with is the case for $j > 0$.
- The most common case in practice is when we know scalars θ_i^{min} and θ_i^{max}, $i = 1, 2, \ldots, p$, such that we want

$$\theta_i^{min} \leq \theta_i(j) \leq \theta_i^{max} \tag{50}$$

for all $j \geq 0$ (this specifies a convex set $\Theta = \{\theta : \theta_i^{min} \leq \theta_i \leq \theta_i^{max}, i = 1, 2, \ldots, p\}$).

- Then, each time you use a gradient update formula to generate $\theta(j+1)$ you test Equation (50) for each $i = 1, 2, \ldots, p$ and if it has chosen

$$\theta_i(j+1) > \theta_i^{max}$$

you let

$$\theta_i(j+1) = \theta_i^{max}$$

and if it has chosen

$$\theta_i(j+1) < \theta_i^{min}$$

you let

$$\theta_i(j+1) = \theta_i^{min}$$

- If any generated $\theta_i(j+1)$ is within the range specified by Equation (50), then you accept the update $\theta_i(j+1)$ with no modification.
- In this way, for all $j \geq 1$ we will never update the parameter vector $\theta(j)$ to lie outside Θ.

Parameter Update Termination

- Let $\theta^\star$ be the solution used at termination, i.e., it may be that $\theta^\star \neq \theta^*$.

- It is best to use "scale-free" termination criteria such as the following:

  1. Terminate if Parameter Change Rate is Low: Terminate if

     $$(\theta(j+1)-\theta(j))^\top(\theta(j+1)-\theta(j)) \leq \epsilon\theta(j)^\top\theta(j)$$

     for some $\epsilon > 0$ and let $\theta^\star = \theta(j+1)$.

  2. Terminate if the Gradient is Small: Terminate if

     $$\nabla J(\theta(j), G)^\top \nabla J(\theta(j), G) \leq \epsilon \nabla J(\theta(0), G)^\top \nabla J(\theta(0), G)$$

     for some $\epsilon > 0$ and let $\theta^\star = \theta(j)$.

➙ While such methods are often used, in practice they are often augmented (i.e., used in conjunction with) other criteria such as the use of a "validation set" or simply a set maximum number of iterations.

Off- and On-Line, Serial and Parallel Data Processing

Off-Line Processing

- Know G a priori and hence M is fixed.
- Can process the data set G in "parallel" by repeated application of the gradient update formula to the entire data set.
- But, could use smaller subsets and process these serially, either in a sequential or random order (and in some cases this can improve performance)

On-Line Processing

- The data processing issues are different for the on-line case since we do not know G a priori since $M \to \infty$.
- The most common case in on-line (real-time) processing is to use one training data pair per time step and take one iteration of the gradient formula; this aligns time steps, data acquistions, and iterations of the gradient update formula.
- But, of course, you could sequentially process (small) subsets of data

Gauss-Newton and Levenberg-Marquardt Methods

➙ Newton's method is theoretically interesting (good convergence rates) but not used in practice due to need for computing the inverse of the Hessian.

- Conjugate gradient and quasi-Newton methods found to be effective in some cases.

  $\boxed{\text{The Levenberg-Marquardt method has been found to be effective in solving practical function approximation problems.}}$

- First, consider the Gauss-Newton method that is used to solve a (nonlinear) least squares problem such as finding θ to minimize $J(\theta, G)$ in Equation (42) when we do not use a linear in the parameter approximator.

- To develop the Gauss-Newton method we have

$$J(\theta, G) = \frac{1}{2} \sum_{i=1}^{M} |y(i) - F(x(i), \theta)|^2$$

and let the $\bar{N} \times 1$ vectors

$$\epsilon(i) = y(i) - F(x(i), \theta)$$

(these are the function approximation errors arising at each piece of training data) and define the $\bar{N}M \times 1$ vector

$$\begin{aligned} \epsilon(\theta, G) &= [\epsilon(1)^\top, \epsilon(2)^\top, \ldots, \epsilon(M)^\top]^\top \\ &= [\epsilon_1, \epsilon_2, \ldots, \epsilon_{\bar{N}M}]^\top \end{aligned}$$

(where ϵ_j, $j = 1, 2, \ldots, \bar{N}M$, are scalars) to be a vector containing all of the approximation errors.

- Note that
$$J(\theta, G) = \frac{1}{2}\sum_{i=1}^{M} \epsilon(i)^\top \epsilon(i) = \frac{1}{2}\epsilon(\theta, G)^\top \epsilon(\theta, G)$$
- For functions $J(\theta, G)$ that are quadratic in θ (scalar or vector case) Newton's method gave very fast convergence (in one step).
- For the function approximation problem, to get $J(\theta, G)$ quadratic in θ, we use a linear in the parameter approximator $F(x, \theta) = \theta^\top \phi(x)$ so that the approximation errors $\epsilon(i)$ and hence $\epsilon(\theta, G)$ are linear (affine) with respect to the parameters θ, and then $J(\theta, G)$ is quadratic in θ.
- If $F(x, \theta)$ is nonlinear in the parameters then so are $\epsilon(i)$ and $\epsilon(\theta, G)$.

Gauss-Newton Parameter Update Formula

★ To tune nonlinear in the parameter approximators, in the Gauss-Newton approach at each iteration j we proceed according to the following steps:

1. Linearize the error $\epsilon(\theta, G)$ about the current value of $\theta(j)$.
2. Solve a least squares problem to minimize the linearized error value and provide the next guess at the parameter, $\theta(j+1)$.

➙ Compared to Newton's method, in the Gauss-Newton method you create a quadratic approximation to the function you want to minimize at each iteration, but now it is done via linearization, rather than using second derivative information.

- We discuss these two steps in more detail next.
- First: Linearize $\epsilon(\theta, G)$ around $\theta(j)$ using a truncated Taylor series expansion to get

$$\hat{\epsilon}(\theta, \theta(j), G) = \epsilon(\theta(j), G) + \nabla\epsilon(\theta, G)^\top \Big|_{\theta=\theta(j)} (\theta - \theta(j))$$

  where $\hat{\epsilon}(\theta, \theta(j), G)$ is an approximation of $\epsilon(\theta, G)$ since we omitted the higher order terms (second order and higher) in the Taylor series expansion.
- We use the notation $\hat{\epsilon}(\theta, \theta(j), G)$ to emphasize the dependence on both θ and $\theta(j)$.

- Here,

$$\nabla\epsilon(\theta,G) = \begin{bmatrix} \frac{\partial\epsilon_1}{\partial\theta_1} & & \cdots & & \frac{\partial\epsilon_{\bar{N}M}}{\partial\theta_1} \\ & \ddots & & & \\ \vdots & & & & \vdots \\ & & & \ddots & \\ \frac{\partial\epsilon_1}{\partial\theta_p} & & \cdots & & \frac{\partial\epsilon_{\bar{N}M}}{\partial\theta_p} \end{bmatrix} = [\nabla\epsilon_1, \nabla\epsilon_2, \ldots, \nabla\epsilon_{\bar{N}M}] \tag{51}$$

  is a $p \times \bar{N}M$ matrix and $\nabla\epsilon(\theta,G)^\top$ is the "Jacobian."

- Second: Minimize the (scaled) squared norm of the linearized function $\hat{\epsilon}(\theta,\theta(j),G)$,

$$J_q(\theta,G) = \frac{1}{2}\hat{\epsilon}(\theta,\theta(j),G)^\top\hat{\epsilon}(\theta,\theta(j),G)$$

  which is a quadratic approximation to $J(\theta,G)$ (at $\theta(j)$) which is nonlinear in θ, and different from the one used in Newton's method.

- Let

$$\begin{aligned}\theta(j+1) &= \arg\min_{\theta} J_q(\theta, G) \\ &= \arg\min_{\theta} \frac{1}{2}\hat{\epsilon}(\theta, \theta(j), G)^\top \hat{\epsilon}(\theta, \theta(j), G)\end{aligned}$$

(here, "$\arg\min_\theta$" is simply mathematical notation for the value of θ that minimizes the norm, it is the "argument" that provides the value that achieves the minimization).

➙ But, we know how to solve this problem.

➙ It is the same as the batch least squares problem for the linear in the parameters case.

- To see this, note that

$$J_q(\theta, G) = \frac{1}{2}E^\top E$$

with $E = \hat{\epsilon}(\theta, \theta(j), G)$.

- Recall that we had

$$E = Y - \Phi\theta$$

- Here, we have

$$\begin{aligned}\hat{\epsilon}(\theta, \theta(j), G) \quad &= \quad \left(\epsilon(\theta(j), G) - \nabla\epsilon(\theta, G)^\top\Big|_{\theta=\theta(j)} \theta(j)\right) \\ &\quad + \nabla\epsilon(\theta, G)^\top\Big|_{\theta=\theta(j)} \theta \qquad (52)\end{aligned}$$

so if we let

$$Y = \left(\epsilon(\theta(j), G) - \nabla\epsilon(\theta, G)^\top\Big|_{\theta=\theta(j)} \theta(j)\right)$$

and

$$\Phi = -\nabla\epsilon(\theta, G)^\top\Big|_{\theta=\theta(j)}$$

our least squares solution (the value of the parameter at the next iteration) is given by Equation (36) as

$$\theta(j+1) \quad = \quad (\Phi^\top \Phi)^{-1} \Phi^\top Y$$

and

$$(\Phi^\top \Phi)^{-1} \quad = \quad \left(\nabla \epsilon(\theta(j), G) \nabla \epsilon(\theta(j), G)^\top \right)^{-1}$$

and

$$\Phi^\top Y \quad = \quad -\nabla \epsilon(\theta(j), G) \left(\epsilon(\theta(j), G) - \nabla \epsilon(\theta(j), G)^\top \theta(j) \right)$$

where

$$\nabla \epsilon(\theta(j), G)^\top = \nabla \epsilon(\theta, G)^\top \Big|_{\theta = \theta(j)}$$

so that the resulting Gauss-Newton update formula is

$$\theta(j+1) = \theta(j) - \left(\nabla \epsilon(\theta(j), G) \nabla \epsilon(\theta(j), G)^\top \right)^{-1} \nabla \epsilon(\theta(j), G) \epsilon(\theta(j), G) \tag{53}$$

(if we had included a step size parameter then the method is sometimes referred to as a "damped" Gauss-Newton approach).

- We do not need the Hessian, only the Jacobian.
- Essentially, a Gauss-Newton iteration is an approximation to a Newton iteration (in the sense that the quadratic approximation at each iteration tries to approximate the one in Newton's method that uses second derivative information in its quadratic approximation) so it can typically provide for faster convergence than, for instance, steepest descent, but generally not as fast as a pure Newton method.
- The Gauss-Newton method is the same as the "extended Kalman filter" (EKF) except where the linearizations are performed (to make the methods the same simply involves changing how you process the data).

Levenberg-Marquardt Parameter Update Formula

- To avoid problems with computing the inverse in Equation (53), the method is often implemented as

$$\begin{aligned}\theta(j+1) &= \theta(j) - \big(\nabla\epsilon(\theta(j),G)\nabla\epsilon(\theta(j),G)^\top \\ &+ \Lambda(j)\big)^{-1}\nabla\epsilon(\theta(j),G)\epsilon(\theta(j),G)\end{aligned} \tag{54}$$

  where $\Lambda(j)$ is a $p \times p$ diagonal matrix such that

$$\nabla\epsilon(\theta(j),G)\nabla\epsilon(\theta(j),G)^\top + \Lambda(j)$$

  is positive definite so that it is invertible.

- Sometimes, a "Cholesky factorization" is used to specify $\Lambda(j)$ at each iteration.

- ➙ In the Levenberg-Marquardt method you choose $\Lambda(j) = \lambda I$ where $\lambda > 0$ and I is the $p \times p$ identity matrix.
  - When $\lambda = 0$ we get the standard Gauss-Newton method and as you increase λ the descent direction moves towards the gradient.
- ➙ Hence, generally, thinking of λ as a step size, we expect that for a small value of λ we will get fast convergence, while for a larger value, we should get slower convergence.

Levenberg-Marquardt Training of a Fuzzy System

➙ We study the use of the Levenberg-Marquardt method for training a Takagi-Sugeno fuzzy system with $R = 11$ rules.

- We will tune all 44 parameters of the approximator.
- Here, we consider off-line batch processing of a data set $G = \{(x(i), y(i)) : i = 1, 2, \ldots, M\}$ from Figure 130 (where in this case $n = 1$).
- In this case, our Takagi-Sugeno fuzzy system is given by

$$y = F_{ts}(x, \theta) = \frac{\sum_{i=1}^{R} g_i(x)\mu_i(x)}{\sum_{i=1}^{R} \mu_i(x)}$$

where $g_i(x) = a_{i,0} + a_{i,1}x_1$ and the $a_{i,j}$, $i = 1, 2, \ldots, R$, $j = 1, 2$ are constants.

- Also,

$$\mu_i(x) = \prod_{j=1}^{n} \exp\left(-\frac{1}{2}\left(\frac{x_j - c_j^i}{\sigma_j^i}\right)^2\right) = \exp\left(-\frac{1}{2}\left(\frac{x_1 - c_1^i}{\sigma_1^i}\right)^2\right)$$

  where c_j^i is the point in the j^{th} input universe of discourse where the membership function for the i^{th} rule achieves a maximum, and $\sigma_j^i > 0$ is the relative width of the membership function for the j^{th} input and the i^{th} rule (since $n = 1$ the premise membership functions are the same as the input membership functions).

- Recall that we had defined

$$\xi_j = \frac{\mu_j(x)}{\sum_{i=1}^{R} \mu_i(x)}$$

  $j = 1, 2, \ldots, R$.

- For our case we have

$$\begin{aligned}\theta &= [c_1^1, \ldots, c_1^R, \sigma_1^1, \ldots, \sigma_1^R, \\ &\quad a_{1,0}, a_{2,0}, \ldots, a_{R,0}, a_{1,1}, a_{2,1}, \ldots, a_{R,1}]^\top\end{aligned}$$

for a total of $p = 4R = 44$ parameters to tune.

Update Formula

➙ The update formula is given in Equation (54).

- To make the computations for the update formula we need, for $\bar{N} = 1$, the $p \times M$ matrix $\nabla\epsilon(\theta(j), G)$ and the $M \times 1$ vector $\epsilon(\theta(j), G)$.
- With $\bar{N} = 1$, the scalars

$$\epsilon_i = \epsilon(i) = y(i) - F_{ts}(x(i), \theta)$$

for $i = 1, 2, \ldots, M$, and so

$$\epsilon(\theta, G) = [\epsilon_1, \epsilon_2, \ldots, \epsilon_M]^\top$$

- Here,

$$\nabla\epsilon(\theta,G) = \begin{bmatrix} \frac{\partial\epsilon_1}{\partial\theta_1} & \cdots & \frac{\partial\epsilon_M}{\partial\theta_1} \\ & \ddots & \\ \vdots & & \vdots \\ & \ddots & \\ \frac{\partial\epsilon_1}{\partial\theta_p} & \cdots & \frac{\partial\epsilon_M}{\partial\theta_p} \end{bmatrix}$$

- Now, notice that for $i = 1, 2, \ldots, M$, $j = 1, 2, \ldots, p$,

$$\begin{aligned} \frac{\partial\epsilon_i}{\partial\theta_j} &= \frac{\partial}{\partial\theta_j}\left(y(i) - F_{ts}(x(i),\theta)\right) \\ &= -\frac{\partial}{\partial\theta_j}F_{ts}(x(i),\theta) \end{aligned}$$

- It is convenient to compute this partial by considering various components of the vector in sequence (not forgetting about the minus sign in front of the partials).

- First, consider the update formulas for the centers of the premise membership functions.
- We will use indices i^* and j^* to help avoid confusion with the indices i and j.
- We find, for $j^* = 1, 2, \ldots, R$,

$$\frac{\partial}{\partial c_1^{j^*}} F_{ts}(x(i^*), \theta) = \frac{\partial}{\partial c_1^{j^*}} \left(\frac{\sum_{i=1}^{R} g_i(x(i^*))\mu_i(x(i^*))}{\sum_{i=1}^{R} \mu_i(x(i^*))} \right)$$

where

$$\mu_i(x(i^*)) = \exp\left(-\frac{1}{2} \left(\frac{x(i^*) - c_1^i}{\sigma_1^i} \right)^2 \right)$$

(we replaced x_1 with x since they are the same) and

$$\xi_{j^*}(x(i^*)) = \frac{\mu_{j^*}(x(i^*))}{\sum_{i=1}^{R} \mu_i(x(i^*))}$$

- Hence, we have

$$\begin{aligned}\frac{\partial}{\partial c_1^{j^*}} F_{ts}(x(i^*),\theta) &= \frac{\left(\sum_{i=1}^R \mu_i(x(i^*))\right)\left(g_{j^*}(x(i^*))\frac{\partial}{\partial c_1^{j^*}}\mu_{j^*}(x(i^*))\right)}{\left(\sum_{i=1}^R \mu_i(x(i^*))\right)^2} \\ &\quad - \frac{\left(\sum_{i=1}^R g_i(x(i^*))\mu_i(x(i^*))\right)\left(\frac{\partial}{\partial c_1^{j^*}}\mu_{j^*}(x(i^*))\right)}{\left(\sum_{i=1}^R \mu_i(x(i^*))\right)^2} \\ &= \left(\frac{g_{j^*}(x(i^*)) - F_{ts}(x(i^*),\theta)}{\sum_{i=1}^R \mu_i(x(i^*))}\right)\frac{\partial}{\partial c_1^{j^*}}\mu_{j^*}(x(i^*))\end{aligned}$$

- For this, let

$$\bar{x}^{j^*} = -\frac{1}{2}\left(\frac{x(i^*) - c_1^{j^*}}{\sigma_1^{j^*}}\right)^2$$

so that using the chain rule from calculus

$$\frac{\partial}{\partial c_1^{j^*}} \mu_{j^*}(x(i^*)) = \frac{\partial \mu_{j^*}(x(i^*))}{\partial \bar{x}^{j^*}} \frac{\partial \bar{x}^{j^*}}{\partial c_1^{j^*}}$$

- We have

$$\frac{\partial \mu_{j^*}(x(i^*))}{\partial \bar{x}^{j^*}} = \mu_{j^*}(x(i^*))$$

and

$$\frac{\partial \bar{x}^{j^*}}{\partial c_1^{j^*}} = \frac{x(i^*) - c_1^{j^*}}{\left(\sigma_1^{j^*}\right)^2}$$

so

$$\frac{\partial}{\partial c_1^{j^*}} F_{ts}(x(i^*), \theta) = \left(\frac{g_{j^*}(x(i^*)) - F_{ts}(x(i^*), \theta)}{\sum_{i=1}^{R} \mu_i(x(i^*))} \right) \mu_{j^*}(x(i^*)) \frac{\left(x(i^*) - c_1^{j^*}\right)}{\left(\sigma_1^{j^*}\right)^2}$$

- Next, for the spreads on the premise membership functions we use the same development above to find

$$\frac{\partial}{\partial \sigma_1^{j^*}} F_{ts}(x(i^*),\theta) = \left(\frac{g_{j^*}(x(i^*)) - F_{ts}(x(i^*),\theta)}{\sum_{i=1}^{R} \mu_i(x(i^*))} \right) \mu_{j^*}(x(i^*)) \frac{\left(x(i^*) - c_1^{j^*}\right)^2}{\left(\sigma_1^{j^*}\right)^3}$$

since

$$\frac{\partial \bar{x}^{j^*}}{\partial \sigma_1^{j^*}} = \frac{\left(x(i^*) - c_1^{j^*}\right)^2}{\left(\sigma_1^{j^*}\right)^3}$$

- Next, for the parameters of the consequent functions notice that

$$\frac{\partial}{\partial a_{j^*,0}} F_{ts}(x(i^*),\theta) = \frac{\partial}{\partial a_{j^*,0}} \left(g_{j^*}(x(i^*))\xi_{j^*}(x(i^*))\right) = \xi_{j^*}(x(i^*))$$

and

$$\frac{\partial}{\partial a_{j^*,1}} F_{ts}(x(i^*),\theta) = x_1(i^*)\xi_{j^*}(x(i^*))$$

➙ This gives us all the elements for the $\nabla\epsilon(\theta, G)$ matrix, and hence we can implement the Levenberg-Marquardt update formula.

Parameter Constraint Set and Initialization

- The chosen parameter constraint set simply forces the centers to lie between -6 and $+6$ (hence, we assume that we know the maximum variation on the input domain a priori) and spreads to all to be between 0.1 and 1 and uses projection to maintain this for each iteration.

➙ We place the contraints on the spreads for two reasons, to avoid a divide-by-zero error and since it simply makes sense to have an upper bound.

- We put no constraints on the parameters of the consquent functions.

➙ The centers are initialized to be on a uniform grid across the input space, a reasonable choice if you do not know where high frequency behavior occurs.

- In particular, we choose $c_1^1 = -5$, $c_1^2 = -4$, up to $c_1^{11} = 5$.
- The spreads are all initalized to be 0.5 so that there is a reasonable amount of separation between when one consequent function of one rule turns on and the other turns off.
- The parameters of the consequent functions are simply initialized to be all zero.

Approximator Tuning Results: Effects on the Nonlinear Part

- Here, we first consider the $M = 121$ case for the function shown in Figure 130.
- We will simply show the mapping shape at various iterations and hence will not implement a termination criterion.
- We choose $\lambda = 0.5$ (you can easily tune this parameter where if you make it smaller it tends to make bigger updates).
- Figure 145 shows the mapping after just one iteration.

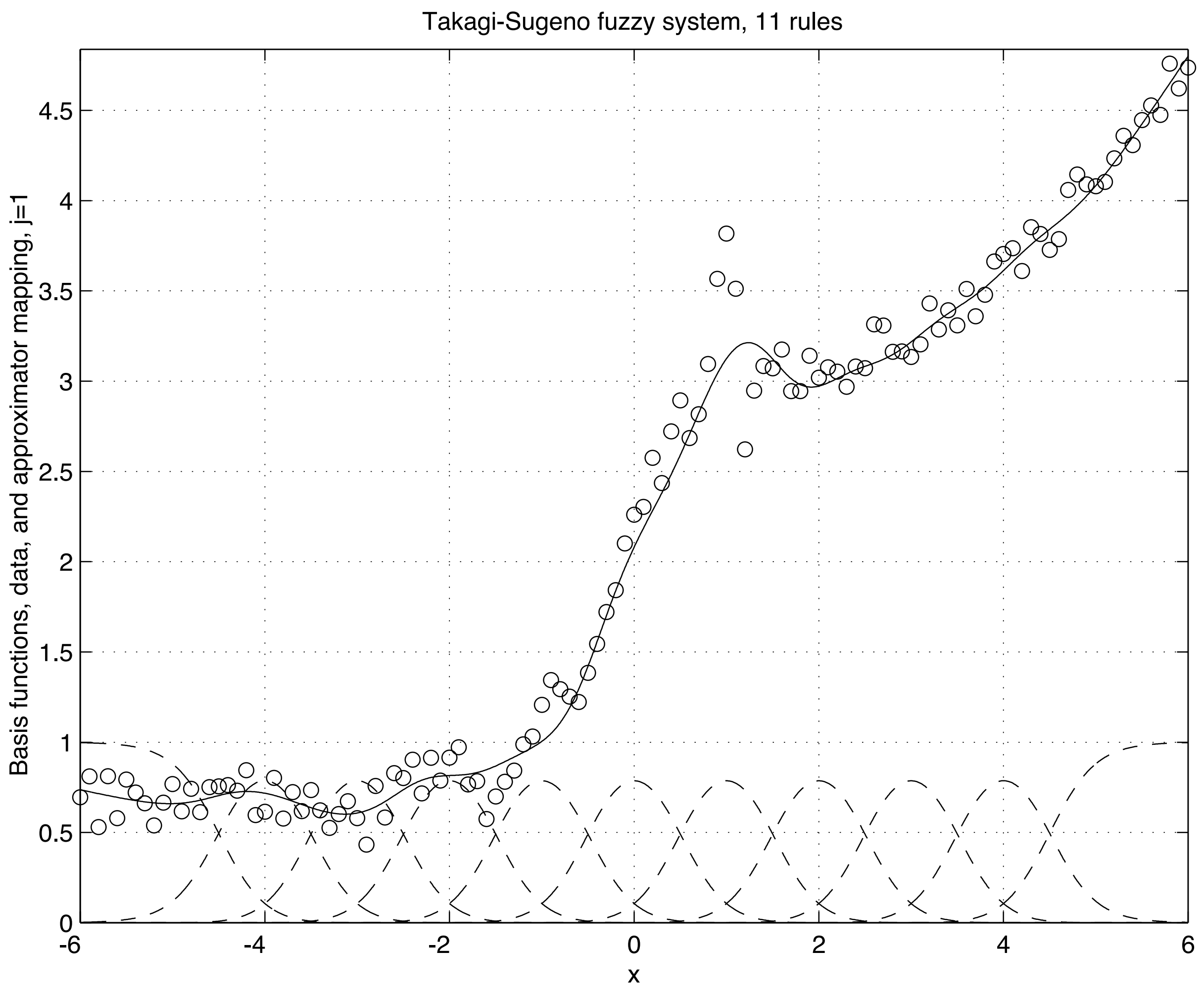

Figure 145: Levenberg-Marquardt training of a Takagi-Sugeno fuzzy system, mapping shape at iteration $j = 1$.

- ★ Clearly, even after one iteration, even though it has not tuned the centers and spreads much, the method has chosen reasonable values for the consequent functions and this is not surprising considering the performance of the batch least squares method for this approach and the similarities to that method.

- ➙ Next, we will focus on how the method tunes the nonlinear part of the approximator (i.e., the μ_i, and hence ξ_i functions) but we must keep in mind that the linear part is also being tuned at the same time.

- ★ Figure 146 shows that by the second iteration there is already significant and successful tuning of the nonlinear part so that approximation errors are reduced, particularly in the region around $x = -2$.

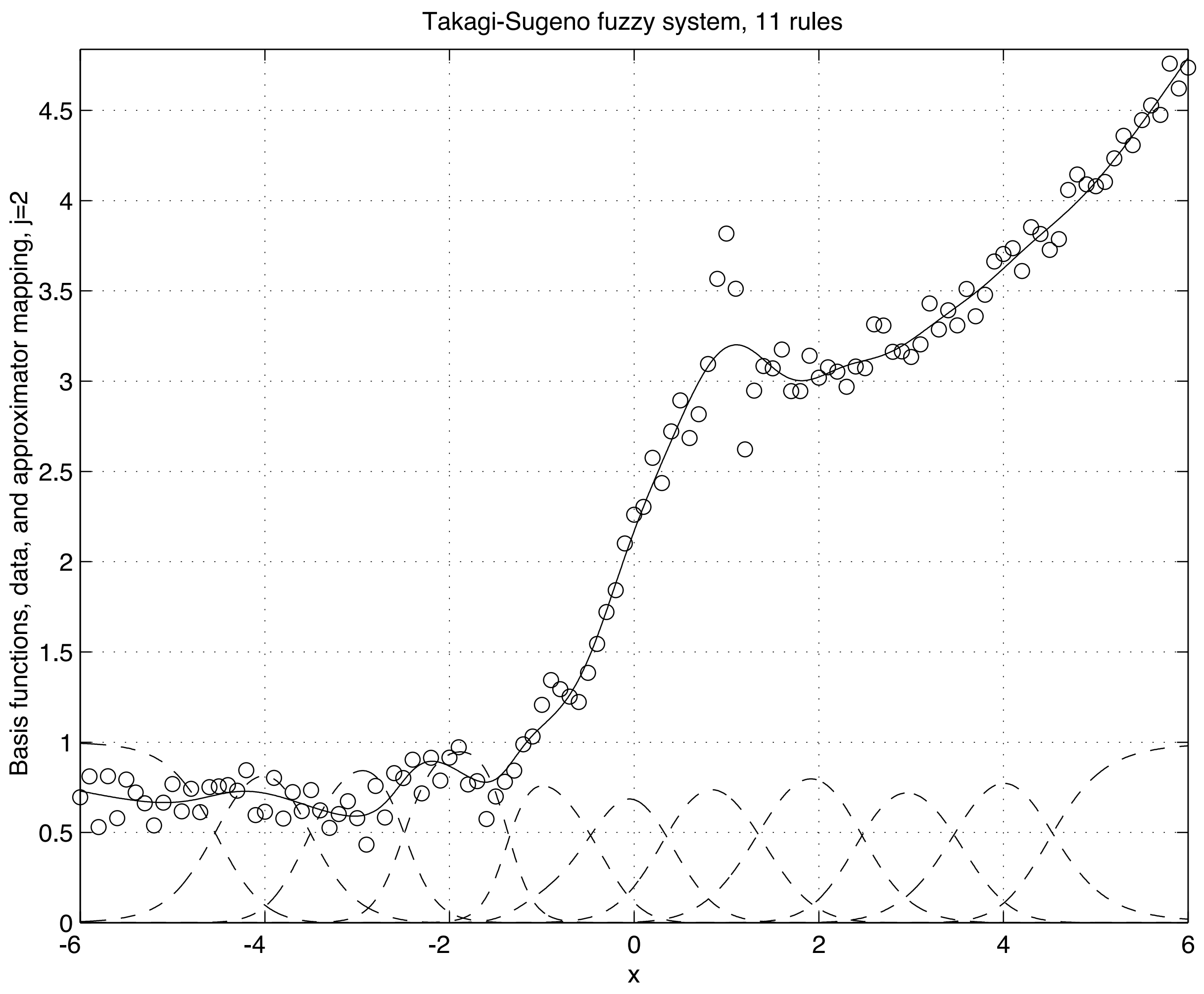

Figure 146: Levenberg-Marquardt training of a Takagi-Sugeno fuzzy system, mapping shape at iteration $j = 2$.

- As the algorithm continues, it continues to tune the nonlinear part of the approximator.

★ In particular, consider Figure 147, at $j = 5$, and we see that at this point the training method has done quite a good job at shaping the nonlinear part to get good accuracy around $x = -2$.

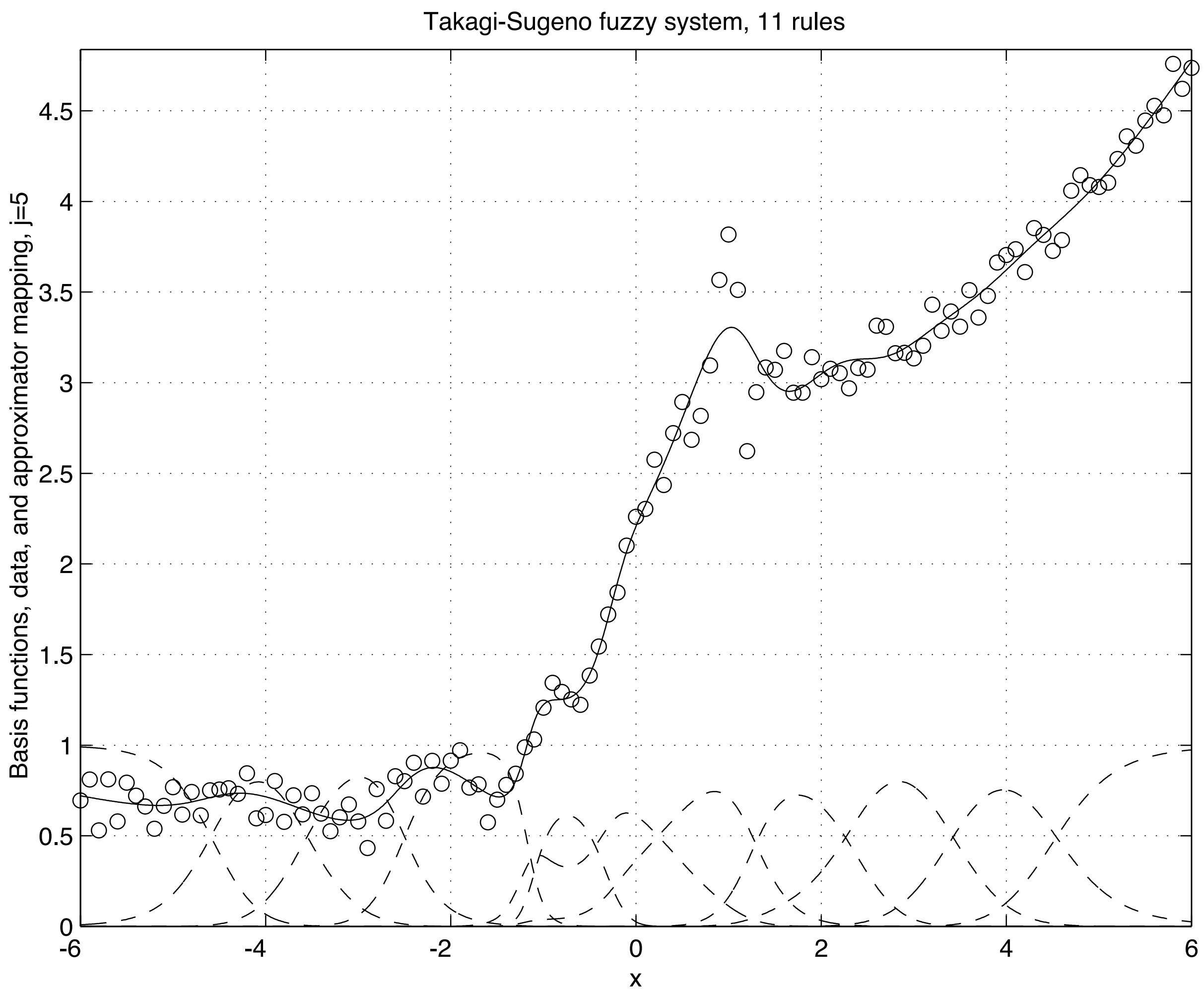

Figure 147: Levenberg-Marquardt training of a Takagi-Sugeno fuzzy system, mapping shape at iteration $j = 5$.

- As the algorithm continues it still continues to tune the nonlinear part of the approximator, both in the region around $x = -2$ and in the high frequency region around $x = 1$.

★ In particular, consider Figure 148, at $j = 12$, and we see that while it has tuned the parameters some, it is not much different in the $x = -2$ region.

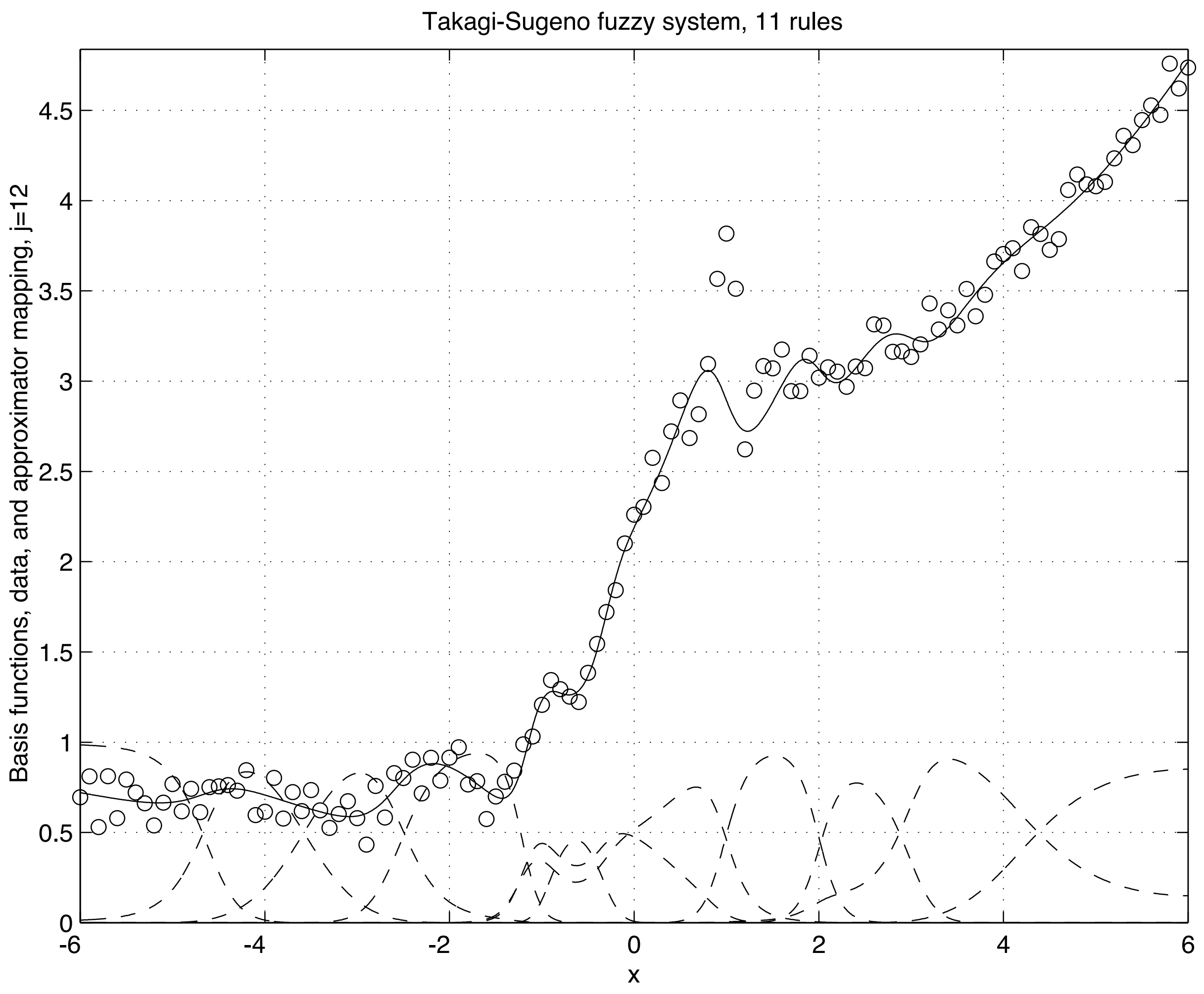

Figure 148: Levenberg-Marquardt training of a Takagi-Sugeno fuzzy system, mapping shape at iteration $j = 12$.

★ It is, however, having difficulties in the $x = 1$ region due to the high frequency behavior.

- It seems that for this example, for higher numbers of iterations, it tends to leave the approximator structure near $x = -2$ pretty much as it is and it tries to "fix" the part near $x = 1$.
- Consider the mapping at iteration $j = 15$ which is shown in Figure 149 and notice that in the $x = 1$ region there is a significant change in the nonlinear shape.

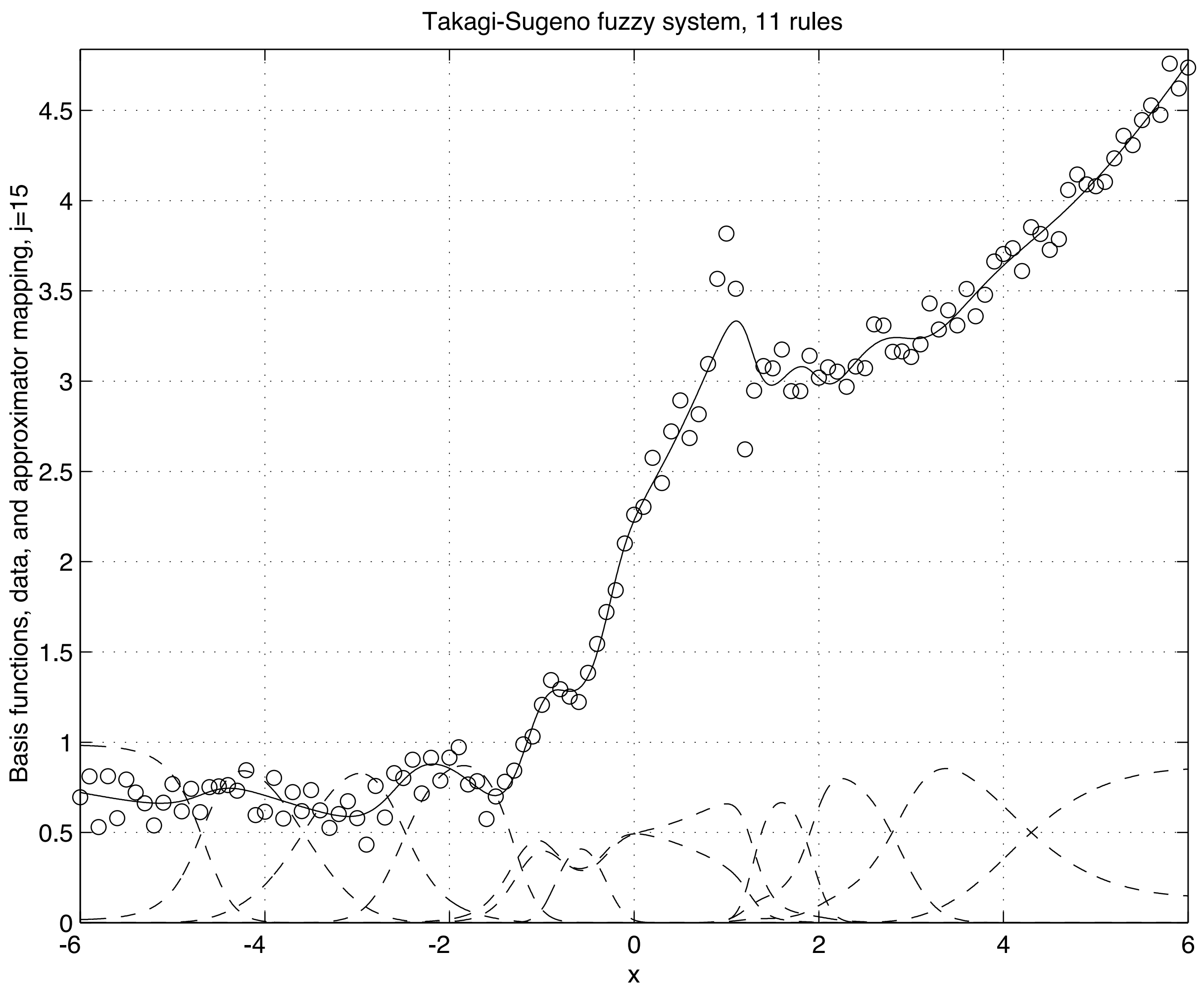

Figure 149: Levenberg-Marquardt training of a Takagi-Sugeno fuzzy system, mapping shape at iteration $j = 15$.

★ It tends to keep moving this shape around near $x = 1$ to try to improve accuracy.

- Now, this is where the issue of termination arises.

➙ Do you terminate at $j = 12$ and declare success?

- Do you try to run the algorithm for many more iterations to see if it can "allocate" more approximator structure to the $x = 1$ high frequency region to improve the accuracy further?

- If you use more iterations will the overall approximation accuracy improve?

➙ Or, will it get even worse that it is here?

➙ These are all important issues, but they tend to be very application dependent.

- It is best if you are aware of all these issues and experiments with the particular application at hand to try to get the best possible results (where the definition of “best” certainly depends on the constraints of the particular application).

Overtraining, Overfitting, and Generalization

- Next, we consider the case where $M = 13$, that is a much smaller data set than used above.

➙ We still use $R = 11$ rules and tune 44 parameters, so our number of parameters is greater than the number of data points.

- We use our earlier choice of initial parameters as $c_1^1 = -5$, $c_1^2 = -4$, up to $c_1^{11} = 5$ with all the spreads as 0.5.
- Also, we use $\lambda = 0.5$ as earlier.

★ In Figure 150 we show the mapping shape at $j = 1$, and we see that it picks a reaonable shape considering how little information it has been given.

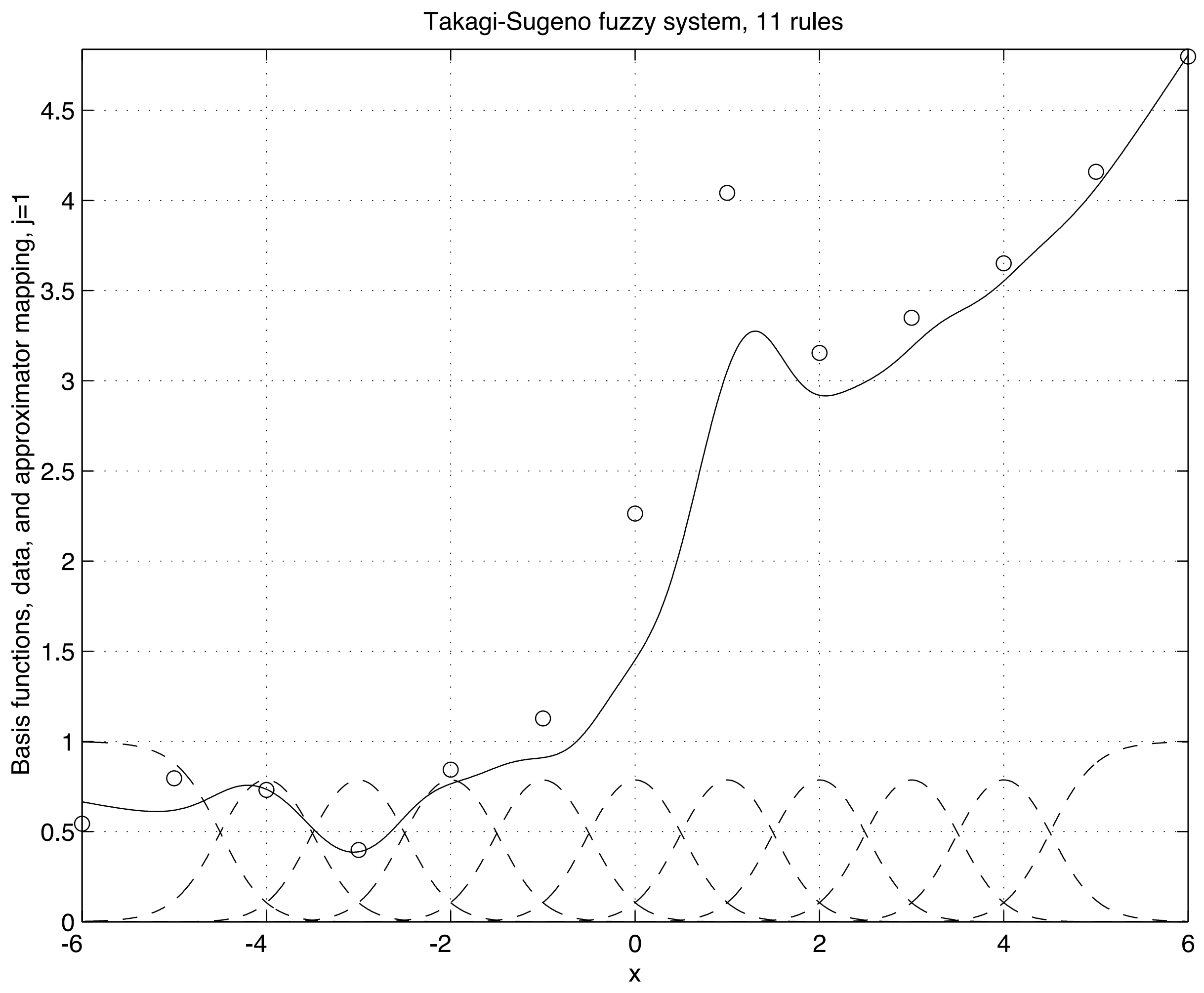

Figure 150: Levenberg-Marquardt training of a Takagi-Sugeno fuzzy system, mapping shape at iteration $j = 1$, $M = 13$.

- ➙ There is, however, a problem when we train with so little data and so many parameters that becomes even clearer if we allow a few more iterations to occur.

- In particular, consider Figure 151, where the mapping is shown at $j = 12$.

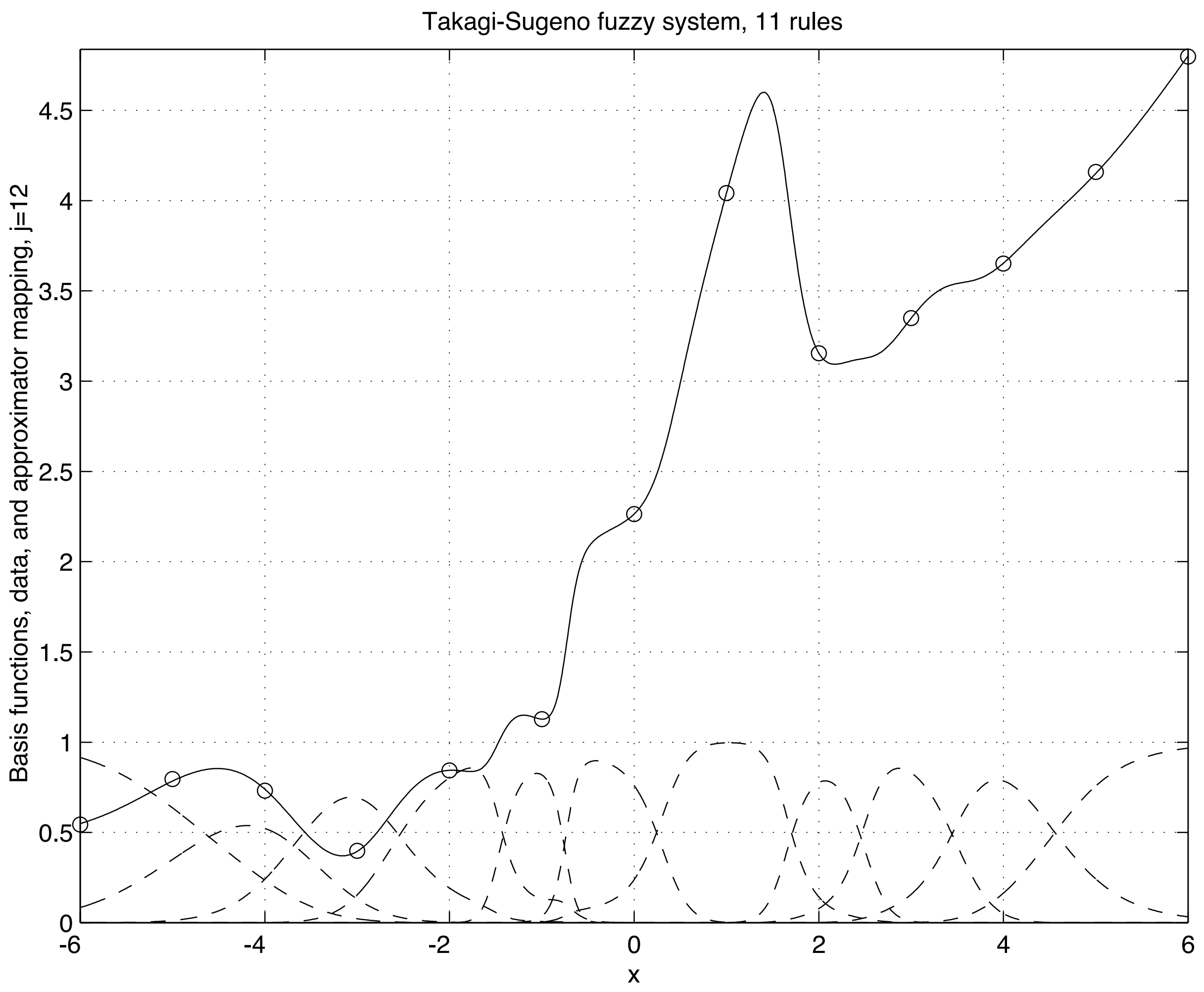

Figure 151: Levenberg-Marquardt training of a Takagi-Sugeno fuzzy system, mapping shape at iteration $j = 12$, $M = 13$.

- ★ We see that the algorithm, in one sense, does a very good job: It matches the training data almost exactly at every point.
- ★ However, this causes a problem since at points outside the training data, the matching to the unknown function is poor (consider, e.g., the large peak near $x = 1$, where even though the mapping goes through one point in that region, we know its shape is not appropriate for the problem at hand).
- ➙ This is called poor "generalization."
- ★ If the approximator generalizes well, then it will produce a good interpolation between the training data, not one that provides large oscillations between the data.
- Moreover, if you study Figure 151 carefully (and compare it to Figure 129 when noise is not added to the function), as we had seen in the least squares case, the approximator is failing also in the sense that it is trying to match the noise in the function

(i.e., it is exhibiting overfitting).

➙ How do we avoid these problems?

- First, you would normally never pick $p > M$, that is, you will normally have fewer parameters than training data pairs.
- Next, in some applications you need to make sure that you do not "overtrain," that is, use too many iterations of the gradient update method.
- Sometimes, this can result in forcing the approximator to match exactly at the data pairs at the expense of performing poor generalization (i.e., poor interpolation between the training data).
- Sometimes, the use of a "validation set" can help to detect when poor generalization is occuring and the updating can be terminated.

Approximation Error Measures: Using a Test Set

➙ Notice that we have been glossing over the issues of the use of a "test set" Γ for evaluating the approximation quality of our approximators, and instead simply have been relying on visual inspection of the plots to comment on approximation accuracy.

- Generally, for more complex multidimensional applications this is not a good approach and you will want to use some type of numerical measure of approximation accuracy where you measure the accuracy *both* at the training data points and at other points (to test interpolation and extrapolation).

Matlab for Training Multilayer Neural Networks and Fuzzy Systems

➙ Matlab Neural Networks Toolbox.

- Matlab Fuzzy Systems Toolbox.
- Matlab Optimization Toolbox.

➙ There are many other software packages available.

Clustering for Classifiers and Approximators

- Can use gradient (and other) methods to construct “classifiers”
  - Classifiers seek to indicate which “class” an input vector lies in
- Training data: pairings between input vectors and classes (e.g., numbered by $1, 2, \ldots, N$)
- A function approximation problem!
  - But often think of training to find an underlying probability density function (that indicates the likelihood that the input vector lies in a certain class).

Adaptive Control

- Some control tasks are instinctual
- Some must be learned

➙ How do learning and evolution, both adaptive methods, apply to the development of control systems?

➙ Here, study how to use adaptation in on-line control.

➙ On-line optimization for adaptation (also heuristic adaptive methods) and stable adaptive methods.

Strategies for Adaptive Control

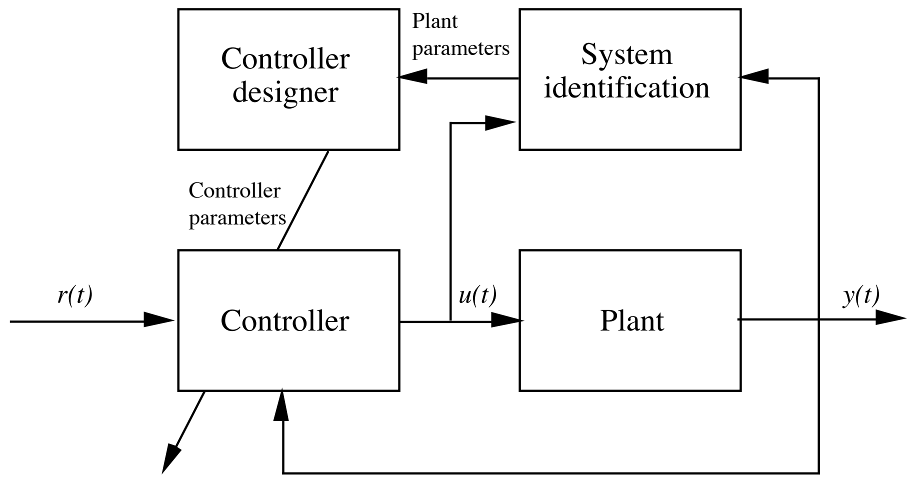

Figure 152: Indirect adaptive control.

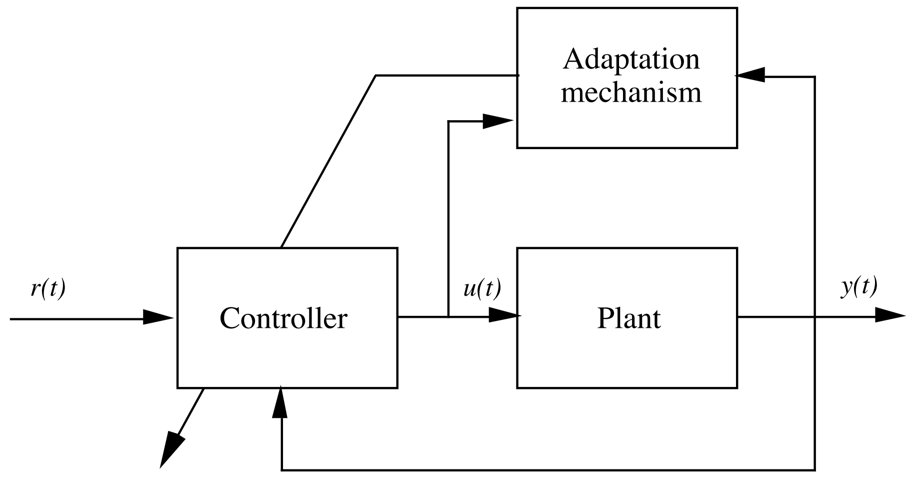

Figure 153: Direct adaptive control.

- Can use approximator structures for model (indirect case) or controller (direct case)
- Can use recursive least squares, gradient methods, genetic/foraging algorithms for adapting (tuning) the approximator.

Stable Adaptive Fuzzy/Neural Control

Class of Nonlinear Systems

Feedback Linearizable Continuous Time Nonlinear Systems

- Consider the plant

$$\dot{x} = f(x) + g(x)u \tag{55}$$

$$y = h(x) \tag{56}$$

  where $x = [x_1, \ldots, x_n]^\top$ is the state vector, u is the (scalar) input, y is the (scalar) output of the plant and functions $f(x)$, $g(x)$, and $h(x)$ are smooth.

- Let $L_g^d h(x)$ be the d^{th} Lie derivative of $h(x)$ with respect to g

$$L_g h(x) = \left(\frac{\partial h}{\partial x}\right)^\top g(x)$$

and, for example,

$$L_g^2 h(x) = L_g(L_g h(x))$$

➙ A system is said to have "strong relative degree" d if

$$L_g h(x) = L_g L_f h(x) =, \cdots, = L_g L_f^{d-2} h(x) = 0$$

and $L_g L_f^{d-1} h(x)$ is bounded away from zero for all x.

- If the system has strong relative degree d, then

$$\begin{aligned}
\dot{z}_1 &= z_2 = L_f h(x) \\
&\vdots \\
\dot{z}_{d-1} &= z_r = L_f^{d-1} h(x) \\
\dot{z}_d &= L_f^d h(x) + L_g L_f^{d-1} h(x) u
\end{aligned} \tag{57}$$

with $z_1 = y$

➙ If we let $y^{(d)}$ denote the d^{th} derivative of y, may be rewritten as

$$y^{(d)} = (\alpha_k(t) + \alpha(x)) + (\beta_k(t) + \beta(x))u \tag{58}$$

- Here, we assume that $y = h(x) = x_1$.
- We will assume that $d = n$ here since it simplifies the stability analysis.

➙ We will assume that if the z_i, $i = 1, 2, \ldots, d = n$ (i.e., $y, \dot{y}, \ldots, y^{(d)}$), are bounded, then so are the x_i, $i = 1, 2, \ldots, d = n$.

- It is assumed that for some $\beta_0 > 0$, we have

$$|\beta_k(t) + \beta(x)| \geq \beta_0$$

so that it is bounded away from zero (could assume $\beta_k(t) + \beta(x) < 0$).

➙ We will assume that $\alpha_k(t)$ and $\beta_k(t)$ are known components of the dynamics of the plant (that may depend on the state) or known exogenous time dependent signals and that $\alpha(x)$ and $\beta(x)$ represent nonlinear dynamics of the plant that are unknown.

- It is assumed that if x is a bounded state vector, then $\alpha(x)$, $\beta(x)$, $\alpha_k(t)$, and $\beta_k(t)$ are bounded signals.
- Throughout the analysis to follow, both $\alpha_k(t)$ and $\beta_k(t)$ may be set to zero for all $t \geq 0$.

Example: Ball on a Beam Experiment

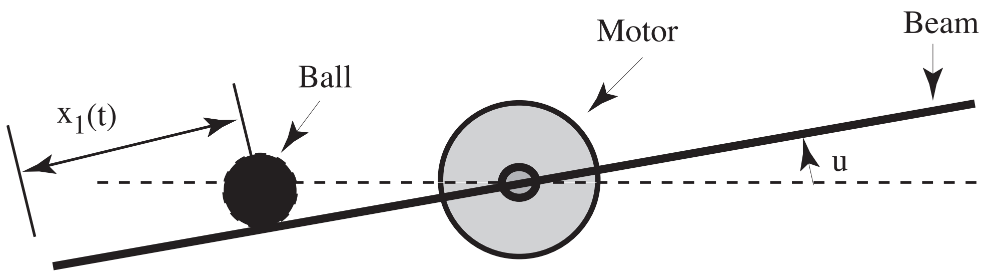

Figure 154: Ball on a beam experiment.

$$\begin{aligned}\dot{x}_1(t) &= x_2(t)\\ \dot{x}_2(t) &= \bar{a}\tan^{-1}(\bar{b}x_2(t))\big(\exp(-\bar{c}x_2^2(t))-1\big)-\bar{d}u(t) \quad (59)\end{aligned}$$

where $x_1(t)$ is the distance from the center of the ball to one end of the beam, $u(t)$ is the angle the beam makes with the horizontal that is controlled with a motor, and $\bar{a} = 9.84$, $\bar{b} = 100$, $\bar{c} = 10^4$, and $\bar{d} = 514.96$.

- While $\bar{d}$ is unknown we assume due to modeling considerations and physical constraints $\bar{d} \in [500, 525]$.

➙ We have $y = x_1$ and

$$\dot{x} = f(x) + g(x)u$$

where

$$f(x) = \begin{bmatrix} x_2 \\ \bar{a}\tan^{-1}(\bar{b}x_2(t))\left(\exp(-\bar{c}x_2^2(t)) - 1\right) \end{bmatrix}$$

and

$$g(x) = \begin{bmatrix} 0 \\ -\bar{d} \end{bmatrix}$$

➜ Relative degree? Take derivatives of the output until the input appears

$$\begin{aligned} \dot{y} &= \dot{h}(x) = \left(\frac{\partial h}{\partial x}\right)^{\top} \dot{x} \\ &= \left(\frac{\partial h}{\partial x}\right)^{\top} f(x) + \left(\frac{\partial h}{\partial x}\right)^{\top} g(x)u \\ &= L_f h(x) + L_g h(x)u \end{aligned}$$

and for our case

$$\frac{\partial h}{\partial x} = [1, 0]^\top$$

so

$$\dot{y} = L_f h(x) = x_2$$

(which is easy to see from the definition of the plant).

- Since $L_g h(x) = 0$ we need to find the second derivative of the plant output.
- Doing this we find (defining $\alpha_k = \beta_k = 0$

$$\begin{aligned}\ddot{y} &= L_f^2 h(x) + L_g L_f h(x) u \\ &= \dot{x}_2 \\ &= \bar{a}\tan^{-1}(\bar{b}x_2(t))\left(\exp(-\bar{c}x_2^2(t)) - 1\right) - \bar{d}u(t) \\ &= \alpha(x) + \beta(x)u\end{aligned}$$

and

$$L_g L_f h(x) = \left(\frac{\partial x_2}{\partial x}\right)^\top \begin{bmatrix} 0 \\ -\bar{d} \end{bmatrix} = [0, 1] \begin{bmatrix} 0 \\ -\bar{d} \end{bmatrix} = -\bar{d} \neq 0$$

so that the relative degree is $d = n = 2$.

- Also, notice that if $y = z_1$ and $\dot{y} = z_2$ are bounded, then x_1 and x_2 are bounded.

➙ Notice, however that there does not exist a $\beta_0 > 0$ such that $\beta(x) \geq \beta_0$ since $\beta(x)$ is known to lie in a fixed interval of *negative* numbers.

Indirect Adaptive Control

Reference Model, Trajectory to be Tracked

➙ We want the output $y(t)$ and its derivatives $\dot{y}(t), \ldots, y^{(d)}(t)$ to track a "reference trajectory" $y_m(t)$ and its derivatives $\dot{y}_m(t), \ldots, y_m^{(d)}(t)$, respectively.

- We will assume that $y_m(t)$ and its derivatives $\dot{y}_m(t), \ldots, y_m^{(d)}(t)$, are bounded.
- A convenient way to specify the reference trajectory signals is to use a "reference model."

➙ One approach: If we have a reference input $r(t)$, with Laplace transform $R(s)$, and $Y_m(s)$ is the Laplace transform of $y_m(t)$,

$$\frac{Y_m(s)}{R(s)} = \frac{q(s)}{p(s)} = \frac{q_0}{s^d + p_{d-1}s^{d-1} + \cdots + p_0}$$

is a reference model where $p(s)$ is the pole polynomial with stable roots and q_0 is a constant.

- As an example, suppose that $r(t) = 0$, $t \geq 0$, so we want $y(t) \to 0$ as $t \to \infty$.
- For this, we could simply choose

$$y_m(t) = \dot{y}_m(t) = \cdots = y_m^{(d)}(t) = 0$$

and this would represent a (perhaps challenging) request to immediately have the output and its derivatives track zero.

➙ To provide a request that the output go to zero "more gently," or according to some dynamics, we could use $r(t) = 0$, $t \geq 0$, so $R(s) = 0$ and

$$p(s)Y_m(s) = 0$$

or

$$(s^d + p_{d-1}s^{d-1} + \cdots + p_0)Y_m(s) = 0$$

or

$$y_m^{(d)}(t) + p_{d-1}y_m^{(d-1)}(t) + \cdots + p_0 y_m(t) = 0$$

- The parameters $p_{d-1}, \ldots, p_0$ specify the dynamics of how $y_m(t)$ evolves over time and hence specifies how we would like $y(t)$ and its derivatives to evolve over time.

On-Line Approximators for Plant Nonlinearities

➙ Approximate the functions $\alpha(x)$ and $\beta(x)$ with

$$\theta_\alpha^\top \phi_\alpha(x)$$

and

$$\theta_\beta^\top \phi_\beta(x)$$

by adjusting the θ_α and θ_β.

➙ The parameter vectors, θ_α and θ_β are assumed to be defined within the compact parameter sets Ω_α and Ω_β, respectively.

➙ In addition, we define the subspace $S_x \subseteq \Re^n$ as the space through which the state trajectory may travel under closed-loop control (a known compact set).

- Notice that

$$\begin{aligned} \alpha(x) &= \theta_\alpha^{*\top} \phi_\alpha(x) + w_\alpha(x) && (60) \\ \beta(x) &= \theta_\beta^{*\top} \phi_\beta(x) + w_\beta(x) && (61) \end{aligned}$$

where

$$\begin{aligned} \theta_\alpha^* &= \arg \min_{\theta_\alpha \in \Omega_\alpha} \left(\sup_{x \in S_x} |\theta_\alpha^\top \phi_\alpha(x) - \alpha(x)| \right) && (62) \\ \theta_\beta^* &= \arg \min_{\theta_\beta \in \Omega_\beta} \left(\sup_{x \in S_x} |\theta_\beta^\top \phi_\beta(x) - \beta(x)| \right) && (63) \end{aligned}$$

so that $w_\alpha(x)$ and $w_\beta(x)$ are approximation errors which arise when $\alpha(x)$ and $\beta(x)$ are represented by finite size approximators.

- We assume that
$$W_\alpha(x) \geq |w_\alpha(x)|$$
and
$$W_\beta(x) \geq |w_\beta(x)|$$
where $W_\alpha(x)$ and $W_\beta(x)$ are known state dependent bounds on the error in representing the actual system with approximators.

- Since we will use universal approximators both $|w_\alpha(x)|$ and $|w_\beta(x)|$ may be made arbitrarily small by a proper choice of the approximator since $\alpha(x)$ and $\beta(x)$ are smooth (of course this may require an arbitrarily large number of parameters p).

- It is important to keep in mind that $W_\alpha(x)$ and $W_\beta(x)$ represent the magnitude of error between the actual nonlinear functions describing the system dynamics and the approximators when the "best" parameters are used, and we do not need to know these best parameters.

- The approximations of $\alpha(x)$ and $\beta(x)$ of the actual system are

$$\begin{aligned} \hat{\alpha}(x) &= \theta_{\alpha}^{\top}(t)\phi_{\alpha}(x) \qquad (64) \\ \hat{\beta}(x) &= \theta_{\beta}^{\top}(t)\phi_{\beta}(x) \qquad (65) \end{aligned}$$

where the vectors $\theta_{\alpha}(t)$ and $\theta_{\beta}(t)$ are updated on line.

- The parameter errors are

$$\begin{aligned} \tilde{\theta}_{\alpha}(t) &= \theta_{\alpha}(t) - \theta_{\alpha}^{*} \qquad (66) \\ \tilde{\theta}_{\beta}(t) &= \theta_{\beta}(t) - \theta_{\beta}^{*} \qquad (67) \end{aligned}$$

- Consider the indirect adaptive control law

$$u = u_{ce} + u_{si} \qquad (68)$$

- The control law is comprised of a "certainty equivalence" control term u_{ce} and a "sliding mode" term u_{si}. We will introduce each of these next.

Certainty Equivalence Control Term

- Let the tracking error be

$$e(t) = y_m(t) - y(t)$$

- Let

$$K = [k_0, k_1, \ldots, k_{d-2}, 1]^\top$$

be a vector of design parameters (whose choice we will discuss below) and

$$e_s(t) = e^{(d-1)}(t) + k_{d-2}e^{(d-2)}(t) + \cdots + k_1\dot{e}(t) + k_0 e(t)$$

- Also, for convenience below we let

$$\bar{e}_s(t) = k_{d-2}e^{(d-1)}(t) + \cdots + k_0\dot{e}(t)$$

so that

$$\bar{e}_s(t) = \dot{e}_s(t) - e^{(d)}(t)$$

- Let

$$L(s) = s^{d-1} + k_{d-2}s^{d-2} + \cdots + k_1 s + k_0$$

and assume that the design parameters in K are chosen so that $L(s)$ has its roots in the (open) left half plane.

➙ Our goal is to drive $e_s(t) \to 0$ as $t \to \infty$.

➙ Notice that $e_s(t)$ is a measure of the tracking error.

- As an example, consider the case where $d = 2$ so $K = [k_0, 1]^\top$ and

$$e_s(t) = \dot{e}(t) + k_0 e(t)$$

- For $L(s)$ to have its roots in the left half plane we have $k_0 > 0$.
- Suppose that we have $e_s(t) = 0$.
- Then,
$$\dot{e}(t) + k_0 e(t) = 0$$
so that
$$\dot{e}(t) = -k_0 e(t)$$
so that $e(t) \to 0$ as $t \to \infty$ and hence $y(t) \to y_m(t)$ as $t \to \infty$.
- The shape of the error dynamics is dictated by the choice of k_0.
- A large k_0 represents that we would like $e(t)$ to go to zero fast, while a small value of k_0 represents that we can accept that $y(t)$ may not achieve good tracking of $y_m(t)$ as fast.
- Note that you do not always want to choose k_0 large because if you make an unreasonable request in the speed of the response the controller may try to use too much control energy to

achieve it.

➙ The certainty equivalence control term is defined as

$$u_{ce} = \frac{1}{\beta_k(t) + \hat{\beta}(x)} \left(- \left(\alpha_k(t) + \hat{\alpha}(x) \right) + \nu(t) \right) \tag{69}$$

where

$$\nu(t) = y_m^{(d)} + \gamma e_s + \bar{e}_s$$

and $\gamma > 0$ is a design parameter whose choice we will discuss below.

➙ As in the discrete-time case we will use projection to ensure that $\beta_k(t) + \hat{\beta}(x)$ is bounded away from zero so that u_{ce} is well-defined.

- The d^{th} derivative of the output error is $e^{(d)} = y_m^{(d)} - y^{(d)}$ so

$$e^{(d)} = y_m^{(d)} - \left(\alpha_k(t) + \alpha(x) \right) - \left(\beta_k(t) + \beta(x) \right) u(t)$$

and since $u = u_{ce} + u_{si}$

$$\begin{aligned} e^{(d)} &= y_m^{(d)} - (\alpha_k(t) + \alpha(x)) - \\ &\quad \frac{\beta_k(t) + \beta(x)}{\beta_k(t) + \hat{\beta}(x)} \left(-(\alpha_k(t) + \hat{\alpha}(x)) + \nu(t)\right) - (\beta_k(t) + \beta(x))u_{si} \end{aligned} \tag{70}$$

- Note that the first two terms $y_m^{(d)} - (\alpha_k(t) + \alpha(x)) =$

$$\begin{aligned} &= y_m^{(d)} - (\alpha_k(t) + \hat{\alpha}(x)) - \alpha(x) + \hat{\alpha}(x) \\ &= (-(\alpha_k(t) + \hat{\alpha}(x)) + \nu(t)) - \alpha(x) + \hat{\alpha}(x) - \nu(t) + y_m^{(d)} \\ &= (-(\alpha_k(t) + \hat{\alpha}(x)) + \nu(t)) - \alpha(x) + \hat{\alpha}(x) - \gamma e_s - \bar{e}_s \end{aligned}$$

- Substituting this into Equation (70) we get

$$\begin{aligned} e^{(d)} &= \left(1 - \frac{\beta_k(t) + \beta(x)}{\beta_k(t) + \hat{\beta}(x)}\right)(-(\alpha_k(t) + \hat{\alpha}(x)) + \nu(t)) - \alpha(x) + \hat{\alpha}(x) \\ &\quad -\gamma e_s - \bar{e}_s - (\beta_k(t) + \beta(x))u_{si} \\ &= (\hat{\alpha}(x) - \alpha(x)) + \left(\hat{\beta}(x) - \beta(x)\right) u_{ce} \\ &\quad -\gamma e_s - \bar{e}_s - (\beta_k(t) + \beta(x))u_{si} \end{aligned} \tag{71}$$

- Since $\bar{e}_s = \dot{e}_s - e^{(d)}$

➙ We get a type of linear relationship between a tracking error measure and model error.

$$\dot{e}_s + \gamma e_s = (\hat{\alpha}(x) - \alpha(x)) + \left(\hat{\beta}(x) - \beta(x)\right) u_{ce} - (\beta_k(t) + \beta(x))u_{si} \tag{72}$$

Parameter Update Laws

- Consider the following Lyapunov function candidate

$$V_i = \frac{1}{2}e_s^2 + \frac{1}{2\eta_\alpha}\tilde{\theta}_\alpha^\top\tilde{\theta}_\alpha + \frac{1}{2\eta_\beta}\tilde{\theta}_\beta^\top\tilde{\theta}_\beta \tag{73}$$

  where $\eta_\alpha > 0$ and $\eta_\beta > 0$ are design parameters whose choice we will discuss below.

- This Lyapunov function quantifies both the error in tracking and in the parameter estimates.

- Using vector derivatives, the time derivative of Equation (73) is

$$\dot{V}_i = e_s\dot{e}_s + \frac{1}{\eta_\alpha}\tilde{\theta}_\alpha^\top\dot{\tilde{\theta}}_\alpha + \frac{1}{\eta_\beta}\tilde{\theta}_\beta^\top\dot{\tilde{\theta}}_\beta \tag{74}$$

- Substituting in the derivative of the tracking error, $\dot{e}_s$ from Equation (72), yields $\dot{V}_i =$

$$e_s\left(-\gamma e_s + (\hat{\alpha}(x) - \alpha(x)) + (\hat{\beta}(x) - \beta(x))u_{ce} - (\beta_k(t) + \beta(x))u_{si}\right)$$
$$+\frac{1}{\eta_\alpha}\tilde{\theta}_\alpha^\top \dot{\tilde{\theta}}_\alpha + \frac{1}{\eta_\beta}\tilde{\theta}_\beta^\top \dot{\tilde{\theta}}_\beta \tag{75}$$

- Notice that

$$\hat{\alpha}(x) - \alpha(x) = \theta_\alpha^\top \phi_\alpha(x) - \theta_\alpha^{*\top}\phi_\alpha(x) - w_\alpha(x) = \tilde{\theta}_\alpha^\top \phi_\alpha(x) - w_\alpha(x)$$

and similarly for $\hat{\beta}(x) - \beta(x)$.
Hence, $\dot{V}_i =$

$$-\gamma e_s^2 +$$
$$\left(\tilde{\theta}_\alpha^\top \phi_\alpha(x) - w_\alpha(x) + \tilde{\theta}_\beta^\top \phi_\beta(x)u_{ce} - w_\beta(x)u_{ce} - (\beta_k(t) + \beta(x))u_{si}\right)e_s$$
$$+\frac{1}{\eta_\alpha}\tilde{\theta}_\alpha^\top \dot{\tilde{\theta}}_\alpha + \frac{1}{\eta_\beta}\tilde{\theta}_\beta^\top \dot{\tilde{\theta}}_\beta$$

➙ Consider the following (gradient) update laws

$$\dot{\theta}_\alpha(t) = -\eta_\alpha \phi_\alpha(x) e_s \tag{76}$$

$$\dot{\theta}_\beta(t) = -\eta_\beta \phi_\beta(x) e_s u_{ce} \tag{77}$$

- We see that $\eta_\alpha > 0$ and $\eta_\beta > 0$ are adaptation gains.
- Picking these gains larger will indicate that you want a faster adaptation.
- Note that since we assume that the ideal parameters are constant $\dot{\tilde{\theta}}_\alpha = \dot{\theta}_\alpha$ and $\dot{\tilde{\theta}}_\beta = \dot{\theta}_\beta$.
- Now, with this notice that

$$\frac{1}{\eta_\alpha}\tilde{\theta}_\alpha^\top \dot{\tilde{\theta}}_\alpha = -\tilde{\theta}_\alpha^\top \phi_\alpha(x) e_s$$

and

$$\frac{1}{\eta_\beta}\tilde{\theta}_\beta^\top \dot{\tilde{\theta}}_\beta = -\tilde{\theta}_\beta^\top \phi_\beta(x) e_s u_{ce}$$

so

$$\begin{aligned}\dot{V}_i &= -\gamma e_s^2 + \left(\tilde{\theta}_\alpha^\top \phi_\alpha(x) - w_\alpha(x) + \tilde{\theta}_\beta^\top \phi_\beta(x) u_{ce} - w_\beta(x) u_{ce}\right) e_s \\ &\quad -(\beta_k(t) + \beta(x)) u_{si} e_s - \tilde{\theta}_\alpha^\top \phi_\alpha(x) e_s - \tilde{\theta}_\beta^\top \phi_\beta(x) e_s u_{ce}\end{aligned}$$

and

$$\dot{V}_i = -\gamma e_s^2 - (w_\alpha(x) + w_\beta(x) u_{ce}) e_s - (\beta_k(t) + \beta(x)) u_{si} e_s \quad (78)$$

Projection Modification to Parameter Update Laws

➙ The above adaptation laws in Equations (76) and (77) will not guarantee that $\theta_\alpha \in \Omega_\alpha$ and $\theta_\beta \in \Omega_\beta$ so we will use projection to ensure this (e.g., to make sure that $(\beta_k(t) + \hat{\beta}(x)) \geq \beta_0$).

- Suppose in particular that we know that the i^{th} component of θ^*_α (θ^*_β) is in the (known) interval

$$\theta^*_{\alpha_i} \in [\theta^{min}_{\alpha_i}, \theta^{max}_{\alpha_i}]$$

  and

$$\theta^*_{\beta_i} \in [\theta^{min}_{\beta_i}, \theta^{max}_{\beta_i}]$$

- Suppose we place the intial values of the parameters in these ranges.

➙ Also, if $\theta_{\alpha_i}(t)$ and $\theta_{\beta_i}(t)$ are strictly within these ranges then you use the update given by the update formulas in Equations (76) and (77).

- ➙ If, however, $\theta_{\alpha_i}(t)$ or $\theta_{\beta_i}(t)$ is on the boundary of its interval and the update formula indicates that it should be moved outside the interval, then you leave it on the boundary of the interval.
- ➙ However, if it is on the boundary and the update law indicates that it should be moved on the boundary or to within the interval, then the update from Equations (76) and (77) is allowed.

- Returning to the stability analysis, clearly since $\theta^*_{\alpha_i}$ and $\theta^*_{\beta_i}$ are within the allowable ranges, this projection modification to the update laws will always result in a parameter estimation error that will decrease V_i at least as much as if the projection were not used; hence, the right hand side of Equation (78) will over bound the $\dot{V}_i$ that would result if projection is used.

➙ For this reason, we conclude that

$$\dot{V}_i \leq -\gamma e_s^2 - (w_\alpha(x) + w_\beta(x) u_{ce}) e_s - (\beta_k(t) + \beta(x)) u_{si} e_s \quad (79)$$

Sliding Mode Control Term

- To ensure that Equation (79) is less than or equal to zero, we choose

$$u_{si} = \frac{(W_\alpha(x) + W_\beta(x)|u_{ce}|)}{\beta_0} \mathrm{sgn}(e_s) \quad (80)$$

where

$$\mathrm{sgn}(e_s) = \begin{cases} 1 & e_s > 0 \\ -1 & e_s < 0 \end{cases} \quad (81)$$

- Note that

$$-(w_\alpha(x) + w_\beta(x) u_{ce}) e_s \leq (|w_\alpha(x)| + |w_\beta(x) u_{ce}|)\, |e_s|$$

- Hence,

$$\begin{aligned}\dot{V}_i &\leq -\gamma e_s^2 + (|w_\alpha(x)| + |w_\beta(x)u_{ce}|)|e_s| \\ &\quad -e_s(\beta_k(t) + \beta(x))\left(\frac{(W_\alpha(x) + W_\beta(x)|u_{ce}|)}{\beta_0}\text{sgn}(e_s)\right)\end{aligned}$$

- Now, considering the last term in this equation and noting that

$$\frac{(\beta_k(t) + \beta(x))}{\beta_0} \geq 1$$

  we have

$$\begin{aligned}\dot{V}_i &\leq -\gamma e_s^2 + |w_\alpha(x)||e_s| + |w_\beta(x)u_{ce}||e_s| \\ &- e_s\text{sgn}(e_s)W_\alpha(x) - e_s\text{sgn}(e_s)W_\beta(x)|u_{ce}|\end{aligned}$$

- Notice that $|e_s| = e_s\text{sgn}(e_s)$ (except at $e_s = 0$) and recall that $|w_\alpha(x)| \leq W_\alpha(x)$ and $|w_\beta(x)| \leq W_\beta(x)$ so

$$|w_\alpha(x)||e_s| - e_s\text{sgn}(e_s)W_\alpha(x) =$$

$$|e_s|(|w_\alpha(x)| - W_\alpha(x)) \leq 0$$

and

$$|w_\beta(x)u_{ce}||e_s| - e_s \text{sgn}(e_s) W_\beta(x)|u_{ce}| = |e_s|(|w_\beta(x)u_{ce}| - W_\beta(x)|u_{ce}|) \leq 0$$

so

$$\dot{V}_i \leq -\gamma e_s^2 \qquad (82)$$

- Since $\gamma e_s^2 \geq 0$ this shows that V_i, which is a measure of the tracking error and parameter estimation error, is a nonincreasing function of time.
- Notice that $\gamma > 0$ has an influence on how fast $V_i \to 0$.
- By picking γ larger you will often get faster convergence of the tracking error.

Asymptotic Convergence of the Tracking Error and Boundedness of Signals

- Given the above assumptions the following hold:
  - The plant output is such that $y, \dot{y}, \ldots, y^{(d-1)}$ are bounded.
  - The input signals u, u_{ce}, and u_{si} are bounded.
  - The parameters $\theta_\alpha(t)$ and $\theta_\beta(t)$ are bounded.
  - We get asymptotic tracking, that is,

$$\lim_{t \to \infty} e(t) = 0$$

- To see this, first note that since V_i is a positive function and

$$\dot{V}_i \leq -\gamma e_s^2 \tag{83}$$

  we know that e_s, θ_α, and θ_β are bounded.

- Since e_s is bounded and y_m and its derivatives (i.e., $y_m, \dot{y}_m, \ldots, y_m^{(d-1)}$) are bounded, we know that $y, \dot{y}, \ldots, y^{(d-1)}$

are bounded.

- Hence, by assumption we have that z and hence x are bounded.
- Hence, $\alpha(x)$, $\hat{\alpha}(x)$, $\alpha_k(t)$, $\beta(x)$, $\hat{\beta}(x)$, and $\beta_k(t)$ are bounded.
- Since x is bounded and $(\beta_k(t) + \hat{\beta}(x)) \geq \beta_0$, u_{ce} and u_{si} and hence u are bounded.
- Next, note that

$$\int_0^\infty \gamma e_s^2 dt \quad \leq \quad -\int_0^\infty \dot{V}_i dt = V_i(0) - V_i(\infty) \tag{84}$$

- This establishes that $e_s \in \mathcal{L}_2$ ($\mathcal{L}_2 = \{z(t) : \int_0^\infty z^2(t)dt < \infty\}$) since $V_i(0)$ and $V_i(\infty)$ are bounded.
- Note that via Equation (72) $\dot{e}_s$ is bounded.
- Hence, since e_s and $\dot{e}_s$ are bounded and $e_s \in \mathcal{L}_2$, we have that $\lim_{t\to\infty} e_s(t) = 0$ (this is what is called Barbalat's lemma).

- It should be clear then, via the definition of $e_s(t)$, that $\lim_{t\to\infty} e(t) = 0$.

Smoothed Control Law

➙ It is possible to augment the above control law with a "bounding control term" that will ensure that the states stay bounded within some region and this can be useful in defining the approximator structures to provide good approximation properties in the region where the states will be.

➙ It is possible to reduce the high frequency signals that can result from the sliding mode control term by using a "smoothed version" of this signal (i.e., one that has a smooth transition from negative to positive values, not the $\text{sgn}(e_s)$ term).

➙ In this case, however, you only get convergence to an ϵ-neighborhood of $e_s = 0$

Direct Adaptive Control

- Can get the same theoretical results via a “direct” adaptive control approach.
- Approximate an unknown controller
- Which approach, direct or indirect, is better?

Design Example: Aircraft Wing Rock Regulation

➙ Aircraft wing rock is a limit cycling oscillation in the aircraft roll angle ϕ and roll rate $\dot{\phi}$.

➙ If δ_A is the actuator output, a model of this phenomenon is given by

$$\ddot{\phi} = a_1\phi + a_2\dot{\phi} + a_3\dot{\phi}^3 + a_4\phi^2\dot{\phi} + a_5\phi\dot{\phi}^2 + b\delta_A$$

where a_i, $i = 1, 2, 3, 4, 5$, and b, are constant but unknown.

- We assume that we know the sign of b.
- Choose the state vector $x = [x_1, x_2, x_3]^\top$ with $x_1 = \phi$, $x_2 = p = \dot{\phi}$, and $x_3 = \delta_A$.
- Suppose that we use a first order model to represent the actuator dynamics of the aileron (the control surface at the outer part of the wing).

- Then we have

$$
\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= a_1 x_1 + a_2 x_2 + a_3 x_2^3 + a_4 x_1^2 x_2 + a_5 x_1 x_2^2 + b x_3 \\
\dot{x}_3 &= -\frac{1}{\tau} x_3 + \frac{1}{\tau} u \\
y &= x_1
\end{aligned}
$$

  where u is the control input to the actuator and τ is the aileron time constant.

- For an angle of attack of 21.5 degrees, $a_1 = -0.0148927$, $a_2 = 0.0415424$, $a_3 = 0.01668756$, $a_4 = -0.06578382$, $a_5 = 0.08578836$.

- Also, $b = 1.5$ and $\tau = \frac{1}{15}$.

- Take these as constant nominal values that you do not know.

- Suppose, however, that you know that $b \in [1, 2]$ and

$\tau \in [\frac{1}{20}, \frac{1}{10}]$.

➙ Also, assume that you know that there is a constant but unknown gain that multiplies the input u (i.e., assume that you know it is not a nonlinear function of x); however, suppose that you do not know that the particular plant nonlinearities are of the form indicated above, or that the parameters appear as they do (i.e., do not use the fact that they enter linearly).

- Suppose that you want the output $y(t)$ to track the reference signal $y_m(t)$ that is zero, and has all its derivatives identical to zero, for all time.
- Assume that you have sensors to measure y, $\dot{y}$, and $\ddot{y}$.
- We will use $x(0) = [0.4, 0, 0]^\top$ in all our simulations.

Indirect Adaptive Controller Development and Results

- The relative degree is $d = n = 3$.
- Assume that $\alpha_k = \beta_k = 0$.
- We use $\beta_0 = 10$.
- After a bit of tuning we chose $k_0 = 100$, $k_1 = 20$, and $\gamma = 2$.
- Also, we tuned the adaptation gains to get fast enough adaptation to meet the objectives.
- In particular, we used $\eta_\alpha = \eta_\beta = 2$.
- Since we assume that we know that b is an unknown constant we can simply use a constant to estimate it
  - We use projection for b to keep it in range.
- To estimate the α term we will use a Takagi-Sugeno fuzzy system but as an input to the premise terms we will only use

x_1 and x_2 (key system variables where nonlinearities enter), while we will use all three state variables as inputs to the consequent (we are trying to avoid problems with computational complexity).

- We placed the centers of the membership functions on the two universes of discourse at -2, 0, and 2, with the spread values all equal to 2, and used all possible combinations of rules so we get $R = 9$ rules.
- This means that we will tune 36 parameters for our approximator.
- We chose $W_\alpha = 0.01$ (simply a guess) and $W_\beta = 0$ (since we know that ideally our approximator can succeed).

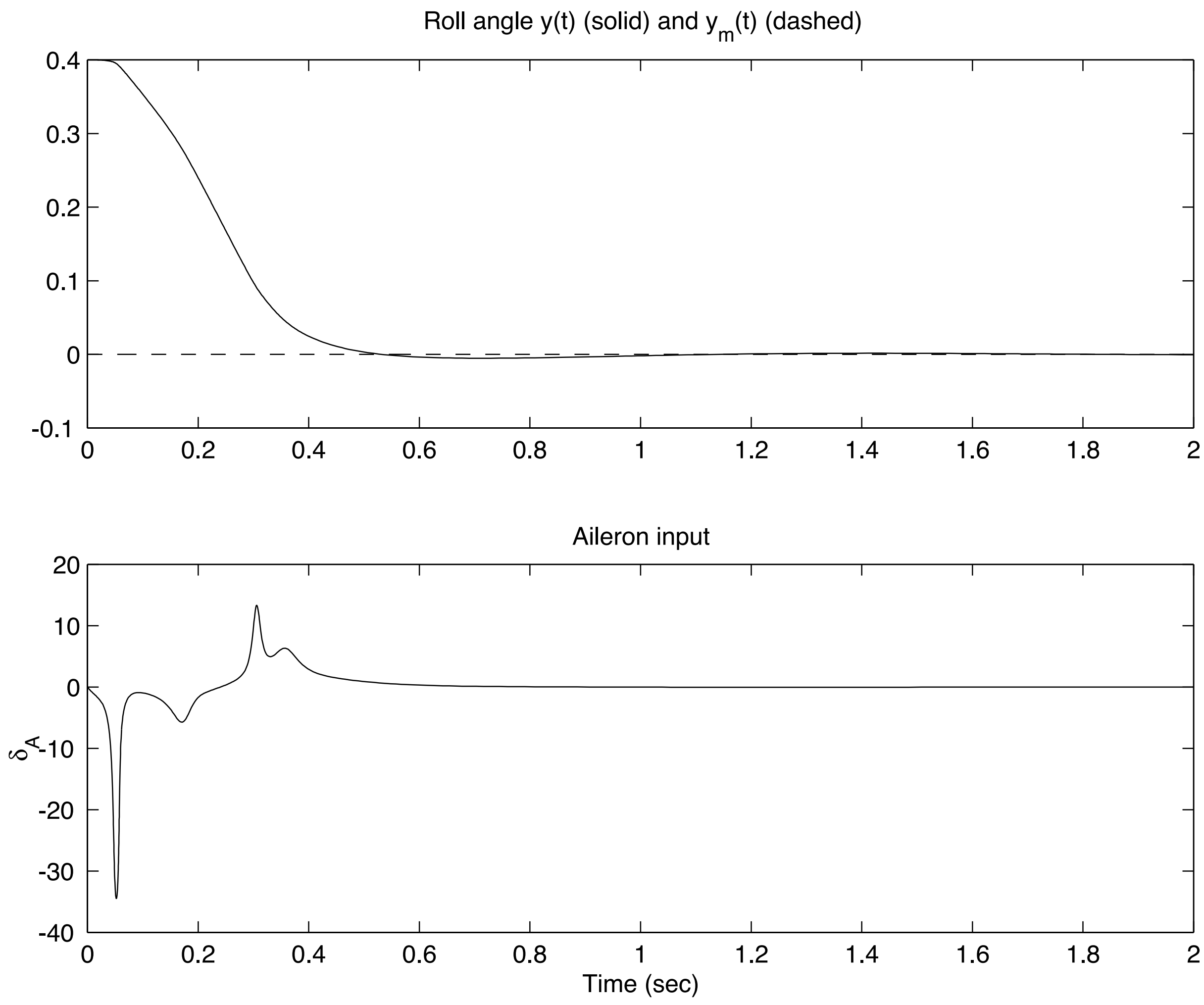

Figure 155: Wing rock controller results, roll angle.

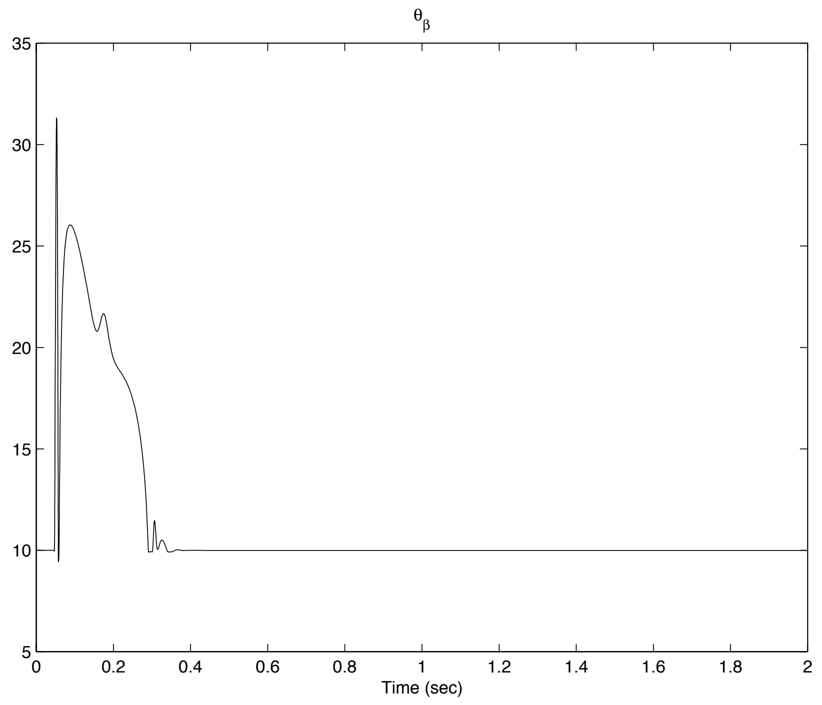

Figure 156: Wing rock controller results, θ_β.

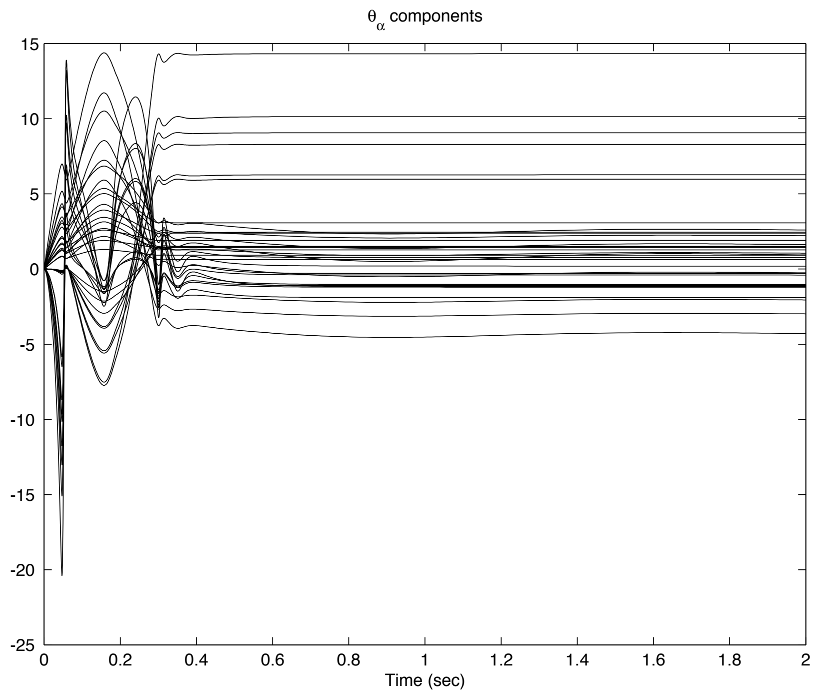

Figure 157: Wing rock controller results, components of θ_α.

EVOLUTION

Focus:

- Genetic algorithms (simulated evolution)
- Stochastic optimization for design (response surface methods, SPSA, set-based, robustness, control design, learning system design)
- Evolutionary adaptive control (GAs for on-line adaptation)

Genetic Algorithms

- GA uses Darwin's theory on natural selection, and Mendel's work in genetics on inheritance, to simulate biological evolution.

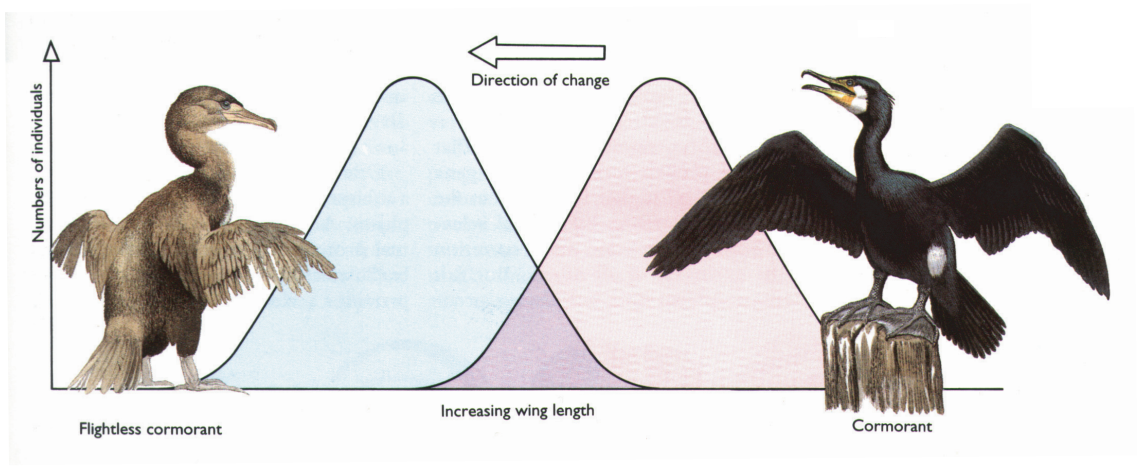

Figure 158: Flightless cormorant.

- Engineering perspective—GA is a stochastic optimization technique that evaluates more than one area of the search space.

- If it "gets stuck" at a local optimum, it tries via multiple search points to simultaneously find other parts of the search space and "jump out" of the local optimum to a global one.

The Population of Individuals

- The "fitness function" measures the fitness of an individual to survive, mate, and produce offspring in a population of individuals for a given environment.

- The genetic algorithm will seek to maximize the fitness function $\bar{J}(\theta)$ by selecting the individuals that we represent with θ.

Strings, Chomosomes, Genes, Alleles, and Encodings

- A string θ represents a chromosome

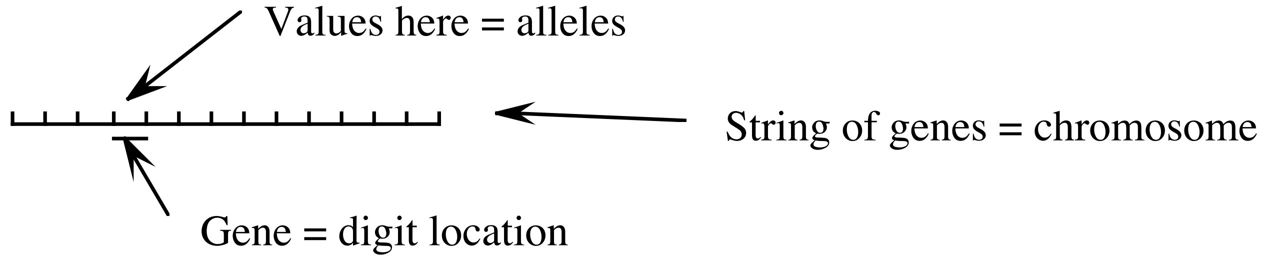

Figure 159: String for representing an individual.

- A chromosome is a string of "genes" that can can take on different "alleles" that are encoded with number systems in a computer.
- Use convention that a gene is a "digit location" that can take on different values from a number system (i.e., different types of alleles).
- For instance, in a base-2 number system, alleles come from the

set $\{0, 1\}$, while in a base-10 number system, alleles come from the set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

➙ Depending on the chosen number system, may need to encode and decode.

➙ Here, study base-2 or base-10 number systems but in examples use base-10 representation to simplify encoding and decoding.

- A 13-position base-10 chromosome:

$$8219345127066$$

➙ Add a gene for the sign of the number (either "+" or "−") and fix a position for the decimal point.

- For the above chromosome we could have

$$+821934.5127066$$

(remember where the decimal point is).

➙ You could also use a floating point representation.

Proportional-Integral-Derivative Controllers: Encoding

- For instance, suppose that you want to evolve a proportional-integral-derivative (PID) controller (e.g., using a fitness function that quantifies closed-loop performance and is evaluated by repeated simulations).
- Three gains K_p, K_i, and K_d and that at some time we have

$$K_p = +5.12,\ K_i = 0.1,\ K_d = -2.137$$

then we would represent this in a chromosome as

$$+051200 + 001000 - 021370$$

which is a concatenation of the digits, where we assume that there are six digits for the representation of each parameter (two before the decimal point and four after it) plus the sign digit (this is why you see the extra padding of zeros).

- We see that each chromosome will have a certain structure (its "genotype" in biological terms, and the entire genetic structure is referred to as the "genome").
- Here, we will use the term from biology "phenotype," to refer to the whole structure of the controller that is to be envolved; hence, in this case the phenotype is

$$K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

where $e = r - y$ is the error input to the PID controller, r is the reference input, and y is the output of the plant.

➙ Similarly, to encode lead-lag compensators, state feedback controllers, nonlinear controllers, fuzzy systems, neural networks, planning systems, attentional systems, learning systems, etc. Structure also.

The Population of Individuals

- Let $\theta_i^j(k)$ be a single parameter at time k (a fixed-length string with sign digit), and suppose that chromosome j is composed of p of these parameters, which are sometimes called "traits" .
- Let, j^{th} chromosome,

$$\theta^j(k) = \left[\theta_1^j(k), \theta_2^j(k), \ldots, \theta_p^j(k)\right]^\top$$

- Concatention–vector representation, use both.
- The population of individuals at time k is

$$P(k) = \left\{\theta^j(k) : j = 1, 2, \ldots, S\right\} \tag{85}$$

➙ S big enough to cover search space, but not too big to hurt computational complexity. S time-varying?

Genetic Operations

- The population $P(k)$ at time k is often referred to as the "generation" of individuals at time k.

➙ Darwin: Most qualified individuals survive to mate and produce off-spring.

- We quantify "most qualified" via an individual's fitness $\bar{J}(\theta^j(k))$ at time k.

- For selection, we create a "mating pool" at time k, something every individual would like to get into, which we denote by

$$M(k) = \left\{ m^j(k) : j = 1, 2, \ldots, S \right\} \tag{86}$$

- The mating pool is the set of chromosomes that are selected for mating.

- Here, we perform selection to decide who gets in the mating pool, mate the individuals via crossover, then induce mutations.
- After mutation we get a modified mating pool at time k, $M(k)$.

➙ To form the next generation for the population, we let

$$P(k+1) = M(k)$$

➙ Evolution occurs as we go from a generation at time k to the next generation at time $k+1$.

Selection

Fitness-Proportionate Selection:

- ➙ Select an individual for mating by letting each $m^j(k)$ be equal to $\theta^i(k) \in P(k)$ with probability

$$p_i = \frac{\bar{J}(\theta^i(k))}{\sum_{j=1}^{S} \bar{J}(\theta^j(k))} \tag{87}$$

- ➙ Use the analogy of spinning a unit circumference roulette wheel where the wheel is cut like a pie into S regions where the i^{th} region is associated with the i^{th} element of $P(k)$.
- ➙ Clearly, individuals who are more fit will end up with more copies in the mating pool; hence, chromosomes with larger-than-average fitness will embody a greater portion of the next generation.
  - At the same time, due to the probabilistic nature of the

selection process, it is possible that some relatively unfit individuals may end up in the mating pool.

Other Selection Strategies:

- Rank and "kill" fixed number of unfit individuals before selection.
- ➙ "Elitist" strategies where very fit individuals are "cloned" into the next generation.

Reproduction Phase, Crossover

- We think of crossover as mating in biological terms, which at a fundamental biological level involves the process of combining (mixing) chromosomes.

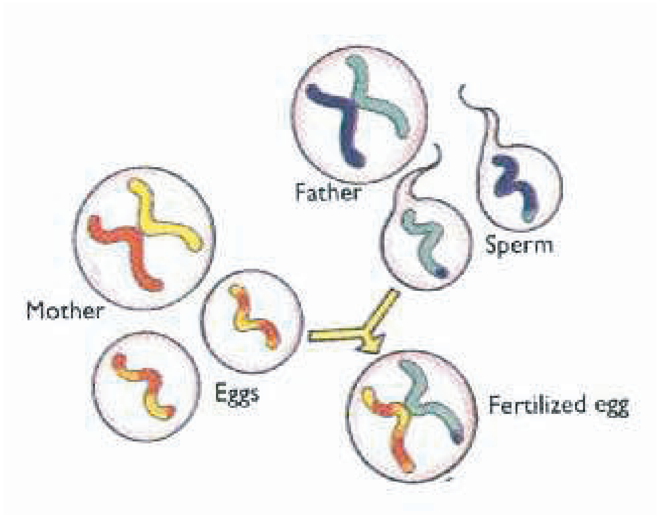

Figure 160: Chomosome swapping in mating.

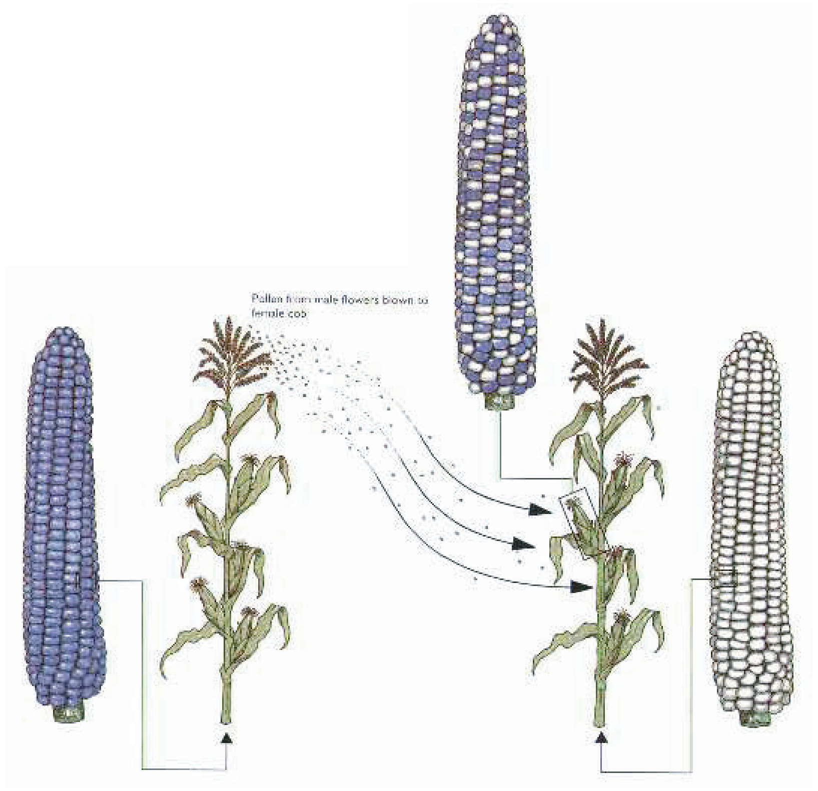

Figure 161: Pollenation in corn, and resulting “offspring.”

- The crossover operation operates on the mating pool $M(k)$.
- “Crossover probability” p_c (usually chosen to be near one since

when mating occurs in biological systems, genetic material is swapped between the parents).

Single-Point Crossover:

➙ Single-point crossover (popular):

1. Randomly pair off the individuals in the mating pool $M(k)$. There are many ways to do this. For instance, you could simply pick each individual from the mating pool and then randomly select a different individual for them to mate with.
2. Consider chromosome pair θ^j, θ^i that was formed in step 1. Generate a random number $r \in [0, 1]$.
   (a) If $r < p_c$ then cross over θ^j and θ^i. To cross over these chromosomes select at random a "cross site" and exchange all the digits to the right of the cross site of one string with those of the other. This process is pictured in Figure 162.

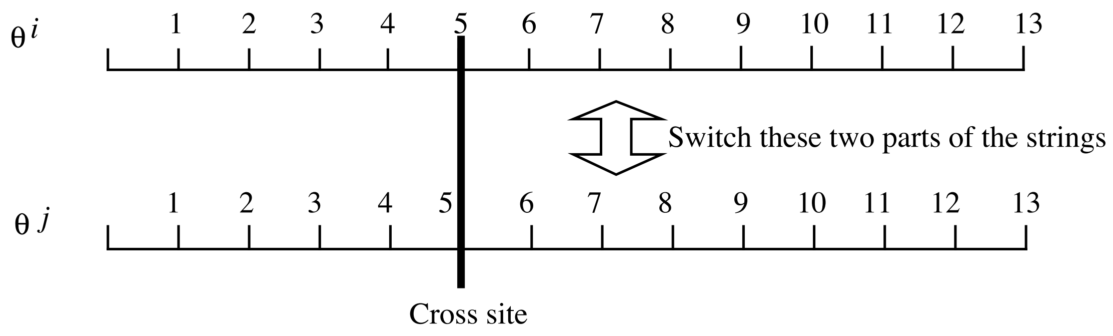

Figure 162: Crossover operation example.

(b) If $r > p_c$ then we will not cross over; hence, we do not modify the strings, and we go to the mutation operation below.

3. Repeat step 2 for each pair of strings that is in $M(k)$.

- As an example, suppose that $S = 10$ and that in step 1 above we randomly pair off the chromosomes.
- Suppose that θ^5 and θ^9 $(j = 5, i = 9)$ are paired off where

$$\theta^5 = +2.9845$$

and

$$\theta^9 = +1.9322$$

- Suppose that $p_c = 0.9$ and that when we randomly generate r we get $r = 0.34$.
- Hence, by step 2 we will cross over θ^5 and θ^9.
- According to step 2 we randomly pick the cross site.
- Suppose that it is chosen to be position three on the string.
- In this case the strings that are produced by crossover are

$$\theta^5 = +2.9322$$

and

$$\theta^9 = +1.9845$$

➙ Basically, crossover perturbs the parameters near good positions to try to find better solutions to the optimization problem.

- It tends to help perform a localized search around the more fit individuals.

- Note: With fitness-proportionate selection we can have multiple copies of an individual in the mating pool; hence, it is possible that an individual will mate with itself in this case.

Other Crossover Methods:

- Multi-point crossover

- Only allow “similar” individuals to mate or “spatially” restrict mating.

Reproduction Phase, Mutation

- Like crossover, mutation modifies the mating pool (i.e., after selection has taken place).
- The operation of mutation is normally performed on the elements in the mating pool after crossover has been performed.

➙ The biological analog of our mutation operation is the random mutation of genetic material.

Gene Mutations:

- To perform mutation in the computer, first choose a mutation probability p_m.

➙ With probability p_m, the most common approach is to change (mutate) each gene location on each chromosome randomly to a member of the number system being used.

- For instance, in a base-2 genetic algorithm, we could mutate

  $$1010111$$

  to

  $$1011111$$

  where the fourth bit was mutated to one.

- What do you do for base-10?

➙ Basically, mutation provides random excursions into new parts of the search space.

➙ It is the main mechanism (crossover can also help) that tries (via luck) to make sure that we do not get stuck at a local maxima and that we seek to explore other areas of the search space to help find a global maximum for $\bar{J}(\theta)$.

- Usually, the mutation probability is chosen to be quite small to avoid degradation to exhaustive search via a random walk in the search space.

  Other Mutation Methods:

➙ Mutate a set of genes (a trait)

➙ Change p_m over time (e.g. decrease it).

Example: Solving an Optimization Problem

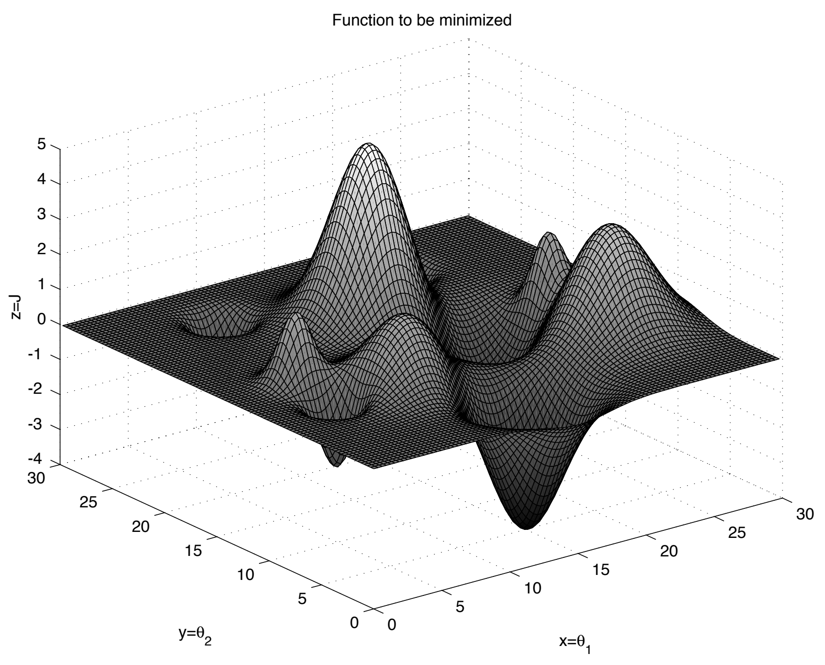

Figure 163: Nonlinear function with multiple extremum points—a fitness function, do not know analytically.

Genetic Algorithm Design

- Matlab + base-10 encoding.

- We use two digits before the decimal point and four after it for a total number of six digits (clearly this constraints the accuracy that we can achieve in the solution to the optimization problem).
- Random or all at zero.
- Know the size of the domain that we want to optimize over and use "projection"
- Since, in Figure 163, the function goes below zero we will shift the whole plot up by a constant (a value of 5 in this case).
- Just shifts fitness values.

➙ Why perform this "shift"? We need postive fitness due to selection method.

- Use fitness-proportionate selection, single-point crossover, and to pair off individuals for mating we pick each one in the mating pool and randomly select a mate for it.

- For a termination criterion we allow no more than a fixed maximum number of iterations (here, 1000).
- Also, we terminate the program when the best fitness in the population has not changed more than $\epsilon = 0.01$ over the last 100 generations.

Algorithm Performance and Tuning

Random Initial Population:

➙ Use $p_c = 0.8$, $p_m = 0.05$, $S = 20$, random initial population.

- Put on contour plot points that represent individuals at some iteration.

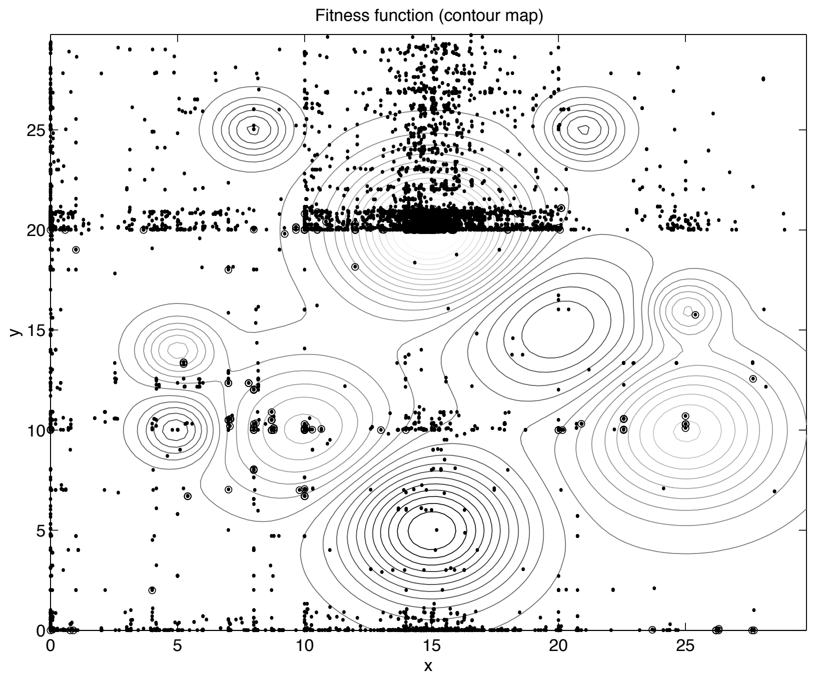

Figure 164: Contour plot of surface in Figure 163 with random initial population.

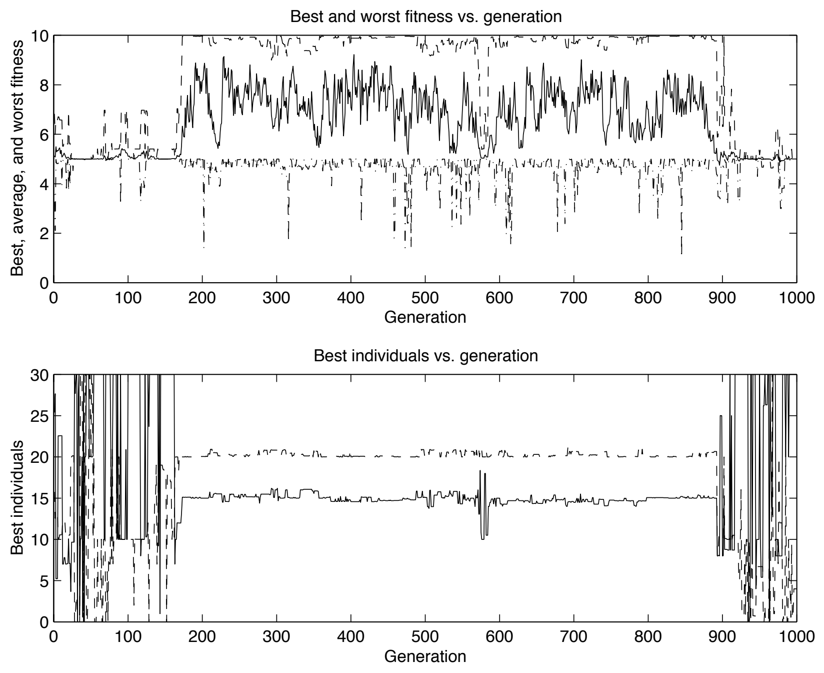

Figure 165: Fitness and optimization parameter evolution, with random initial population.

★ The algorithm performs well, but it does find the best individual, then later it loses it.

★ Note that if you run the algorithm again it may not do as well since it may be unlucky in its random initial choices (this shows why you may want a big population size; if it is big then it is more likely that it will make at least one good initial choice).

Initial Population of all Zeros:

- All same but an initial population with all zeros.

★ Fails to find the optimum point by termination.

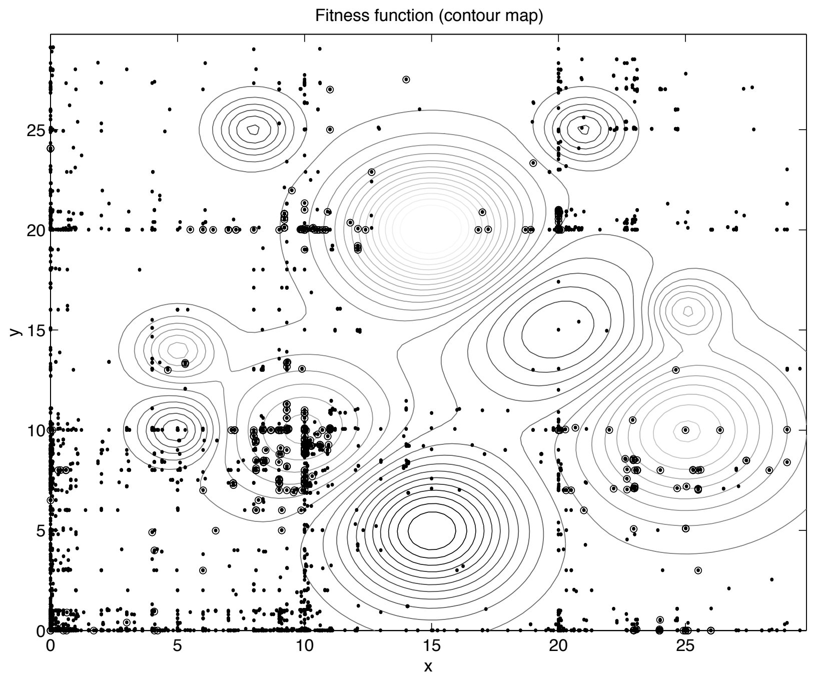

Figure 166: Contour plot of surface in Figure 163 with initial population of all zeros.

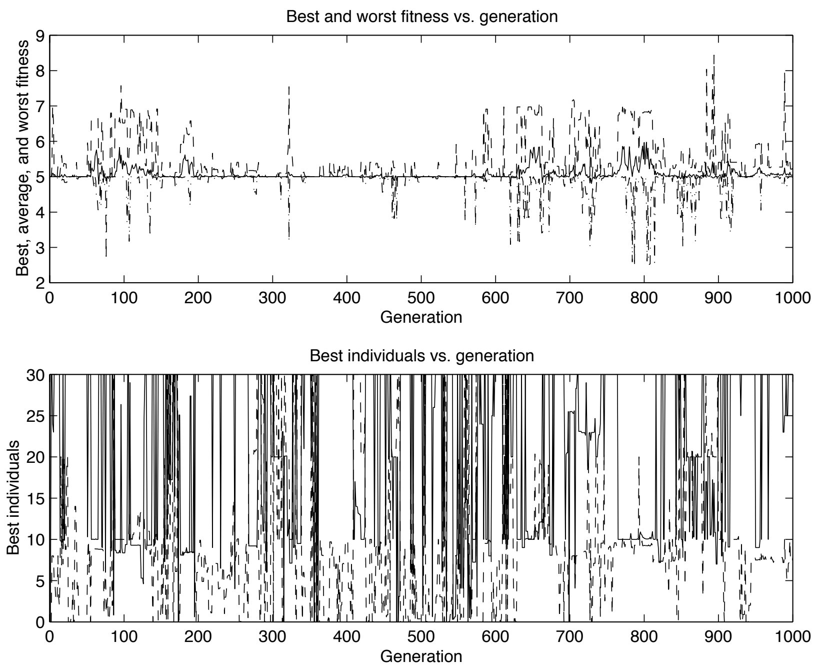

Figure 167: Fitness and optimization parameter evolution, with initial population of all zeros.

Increased Mutation Probability:

- Next, $p_c = 0.8$ and $p_m = 0.1$ (a larger value than above) and $S = 20$.

★ Fails, mutation is destroying good solutions (i.e., it destroys the progress of the method).

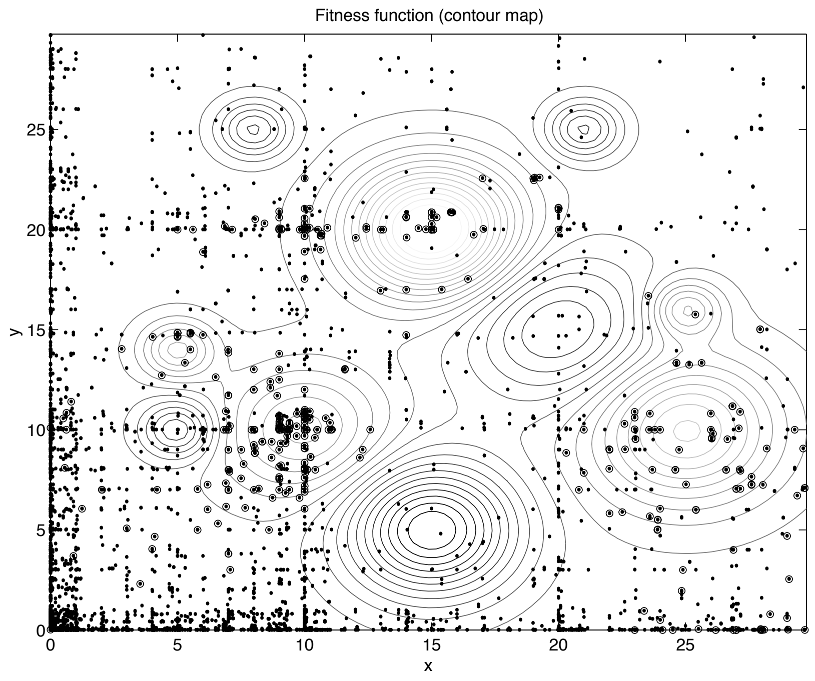

Figure 168: Contour plot of surface in Figure 163 with random initial population and higher mutation probability.

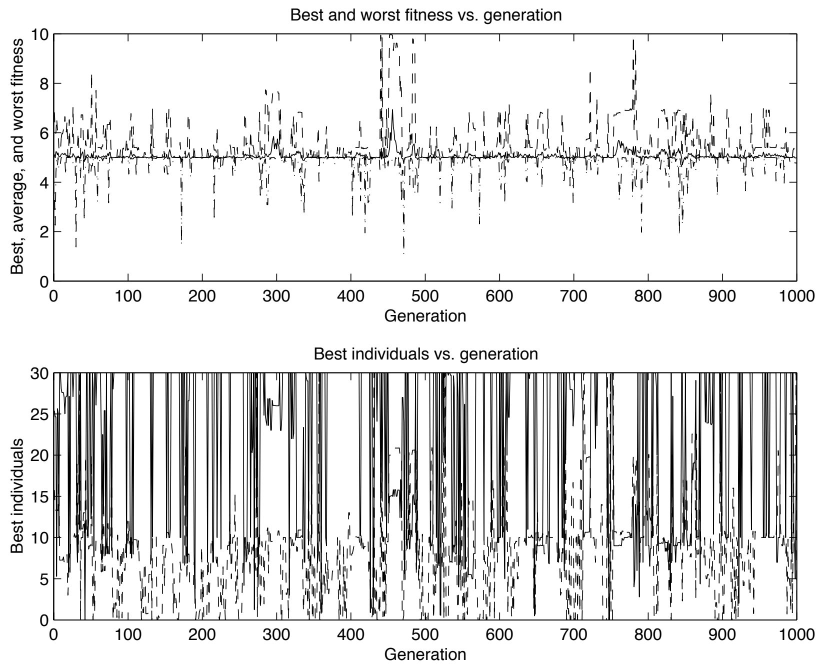

Figure 169: Fitness and optimization parameter evolution, with random initial population and higher mutation probability.

★ Mutation probability too high → “random walk” in the parameter space → attribute its success to “dumb luck.”

Decreased Crossover Probability:

- Next, $p_c = 0.5$ (a smaller value than above) and $p_m = 0.05$ (i.e., return it to its earlier value) and $S = 20$.

★ Lower crossover probability $\longrightarrow$ does less local search between good solutions.

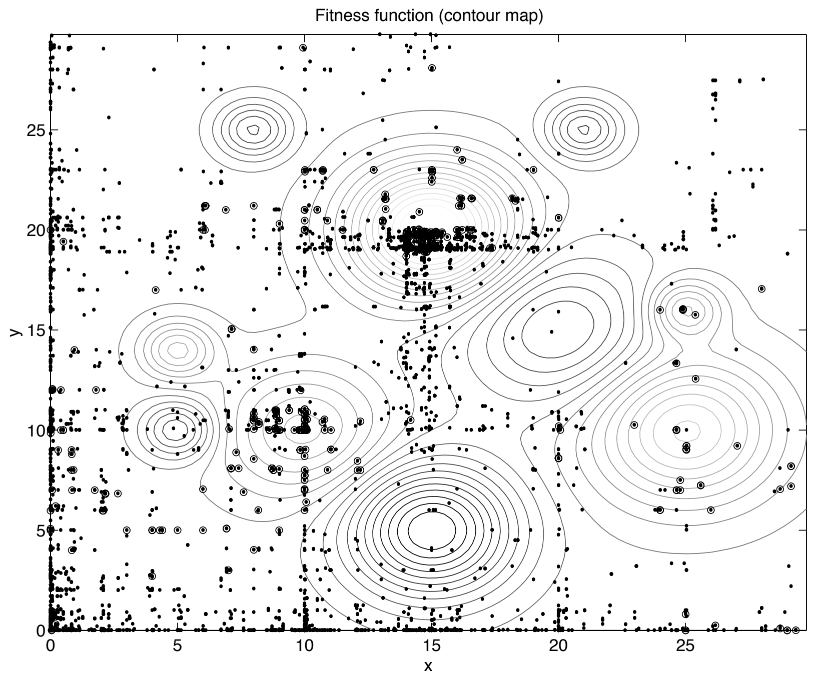

Figure 170: Contour plot of surface in Figure 163 with random initial population and lower crossover probability.

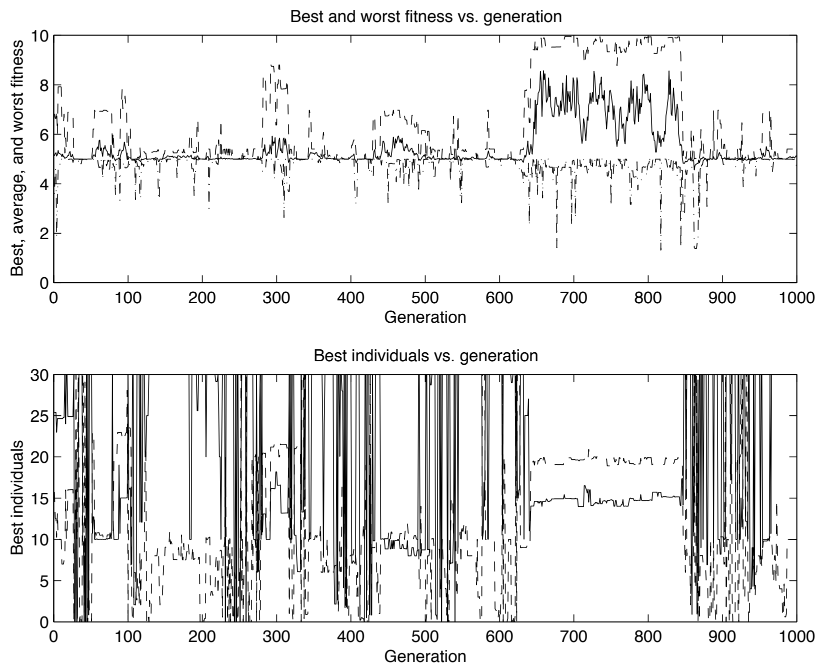

Figure 171: Fitness and optimization parameter evolution, with random initial population and lower crossover probability.

★ If you make $p_c = 0.1$ it fails to find a local optimum (at least for one time the algorithm was run).

★ In this case it is passing too many individuals through the mating process without mixing genetic material; hence it stagnates.

Increased Population Size:

- Next, $p_c = 0.8$ and $p_m = 0.05$ and consider $S = 40$ (i.e., twice as big as earlier).

★ Still get convergence but increasing its size is not necessarily good

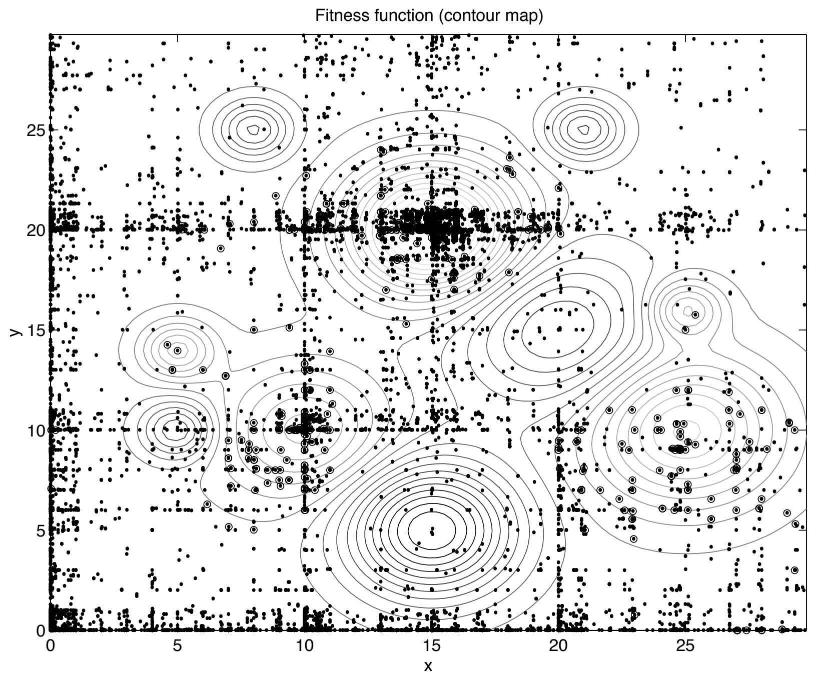

Figure 172: Contour plot of surface in Figure 163 with random initial population and increased population size.

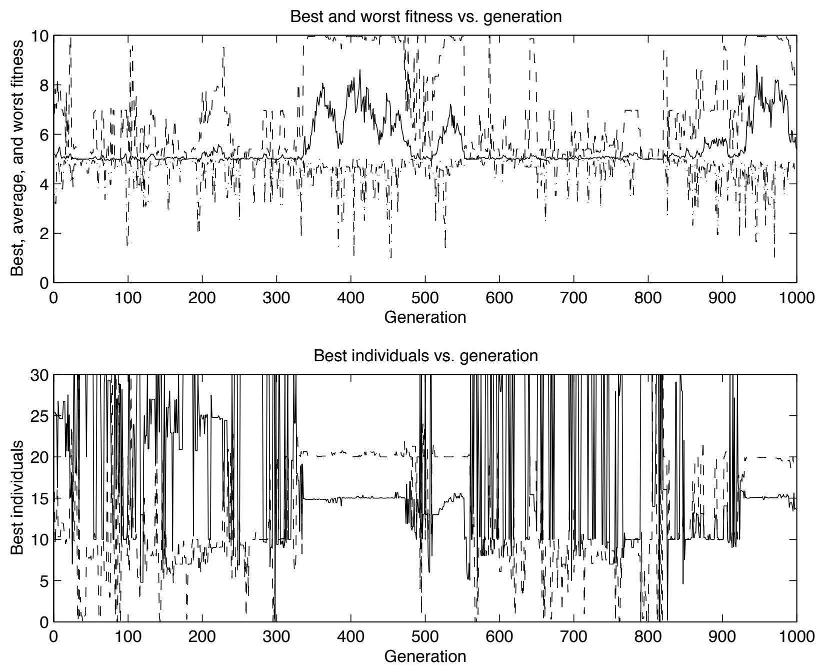

Figure 173: Fitness and optimization parameter evolution, with random initial population and increased population size.

★ Of course, we have to qualify this statement by saying "for this run of the program, with these termination criteria, etc."

★ This simulation was produced simply to make the point that bigger is not always better (even though for the population, for some applications, this may generally be true).

Effects of Elitism:

- Next, $p_c = 0.8$ and $p_m = 0.05$ and $S = 20$.
- Now, however, we use elitism with a single elite member.

★ Quicker convergence (notice that the early termination criterion was invoked) since crossover and mutation do not alter the best individual.

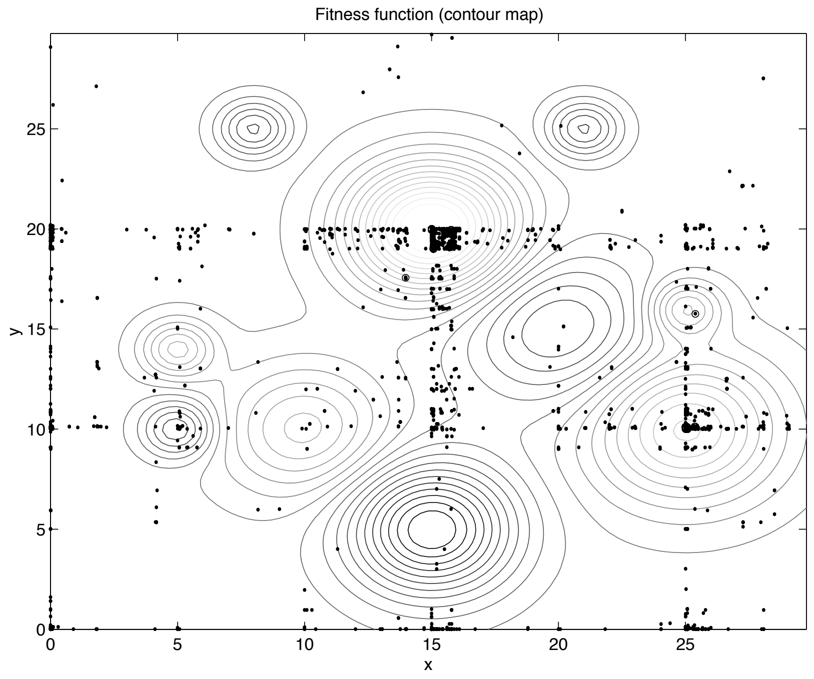

Figure 174: Contour plot of surface in Figure 163 with random initial population and elitism.

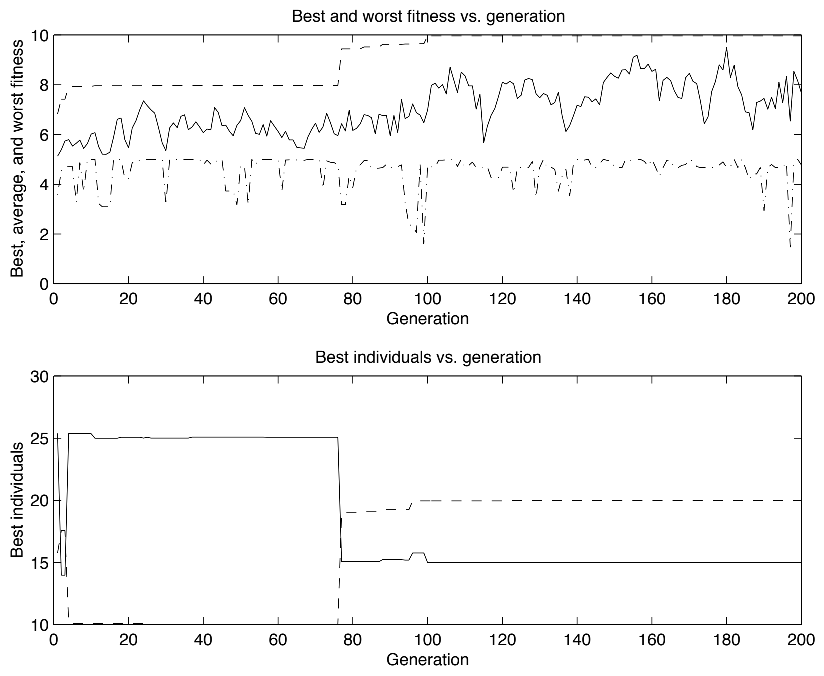

Figure 175: Fitness and optimization parameter evolution, with random initial population and elitism.

★ Elitism successful for a variety of applications.

- Part of the population explores, another part remembers best progress.

Stochastic and Nongradient Optimization for Design

Relevant Theories of Biological Evolution

➙ Evolution of Robust Organisms and Systems:

- Evolution (engineering design) is the design of optimized robust organisms (systems), for typical events encountered in an environment, and complexity may result depending on the ecological niche (problem domain).
- “Highly optimized tolerance” (HOT) by J. Doyle et al.—systems that have been designed, via engineering methodology or evolution, to have optimized performance in an uncertain environment.
- “Robust yet fragile”—Optimal robust designs are for a certain set of conditions, and these designs are sensitive to conditions they were not optimized for—robustness

trade-offs. Get tolerance to some events at expense of sensitivity to others, good design optimizes trade-offs ("conservation principle" at work).

- Familiar concept in robust control (via "sensitivity function").

➙ Learning and Evolution: Synergistic Effects:

- Learning provides one approach to achieve robust behavior.
- Genes determine learning (e.g., example in mice), evolution works on gene pool.
- Learning evolved. Why? Environment has static parts and unknown but predictable parts. Organism simpler than environment, can sense environment but cannot store perfect representation, costs to store information (physiologically), so it operates under uncertainty, with potential to increase success via storage of information.

- Organism will encode static part of environment? Instincts?
- Uncertain part? Store and try to predict potentially successful so selective pressure for learning.
- Above robustness principles (trade-offs) apply—cannot learn everything, so will be best at learning what is most important for survival.
- The Baldwin Effect—learning can accelerate the evolution of instincts! How? Higher probability of “genetic encoding” of information about some aspect of the environment if learning is present.
- Characteristics of the environment drive the construction of an optimal balance between instincts and learning.
- Cultural influence on learning and evolution?

Response Surface Methodology for Design

➙ An intuitive approach, based on surface construction (visually appealing).

- Show via the tanker ship controller design problem

  Proportional Derivative Controller Design for a Tanker Ship

- Design the K_p and K_d gains

$$\delta = K_p e + K_d c$$

  where $e = \psi_r - \psi$ and c is the "change-in-error"

- The i^{th} design is K_p^i and K_d^i.
- Performance objective uses a (discretized) linear first order reference model

$$G(s) = \frac{\frac{1}{150}}{s + \frac{1}{150}}$$

- ψ_r is input, output ψ_m.

➙ Conduct simulation for a specified reference input sequence and the i^{th} controller design, N_s values

$$J_{cl}(K_p^i, K_d^i, \psi_r) = w_1 \sum_{j=0}^{N_s} (\psi(j) - \psi_m(j))^2 + w_2 \sum_{j=0}^{N_s} (\delta(j))^2 \quad (88)$$

- Choose $w_1 = 1$ and $w_2 = 0.01$.

Constructing a Response Surface, Choosing "Optimal Gains"

- Grid the (K_p, K_d) "design space"

$$K_p \in [-1.5, -0.5]$$

and

$$K_d \in [-500, 100]$$

- Compute the performance index $J_{cl}(K_p^i, K_d^i, \psi_r)$ in Equation 88 for nominal conditions, the same reference input sequence ψ_r, w_1, and w_2.

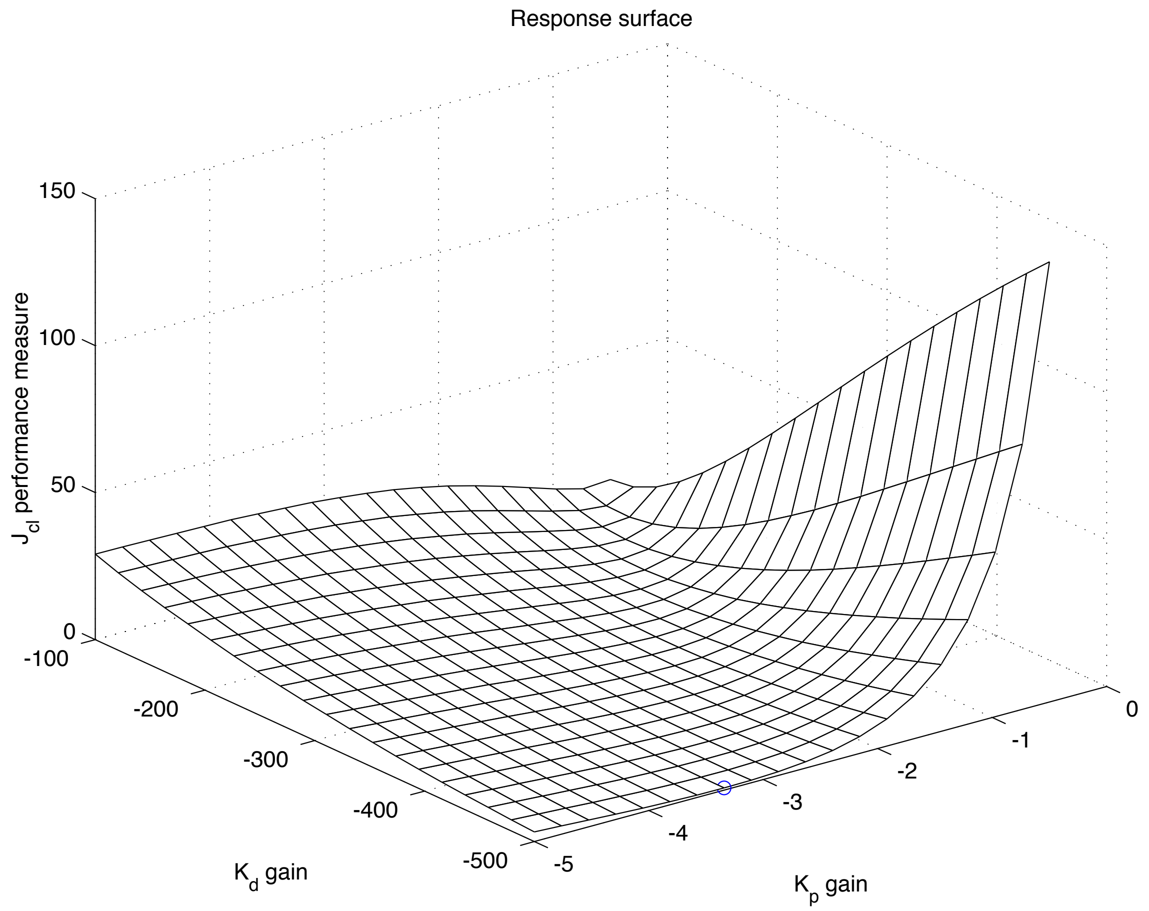

Figure 176: Response surface of J_{cl} for PD controller designs (best gains are $K_p = -3.3421$ and $K_d = -500$).

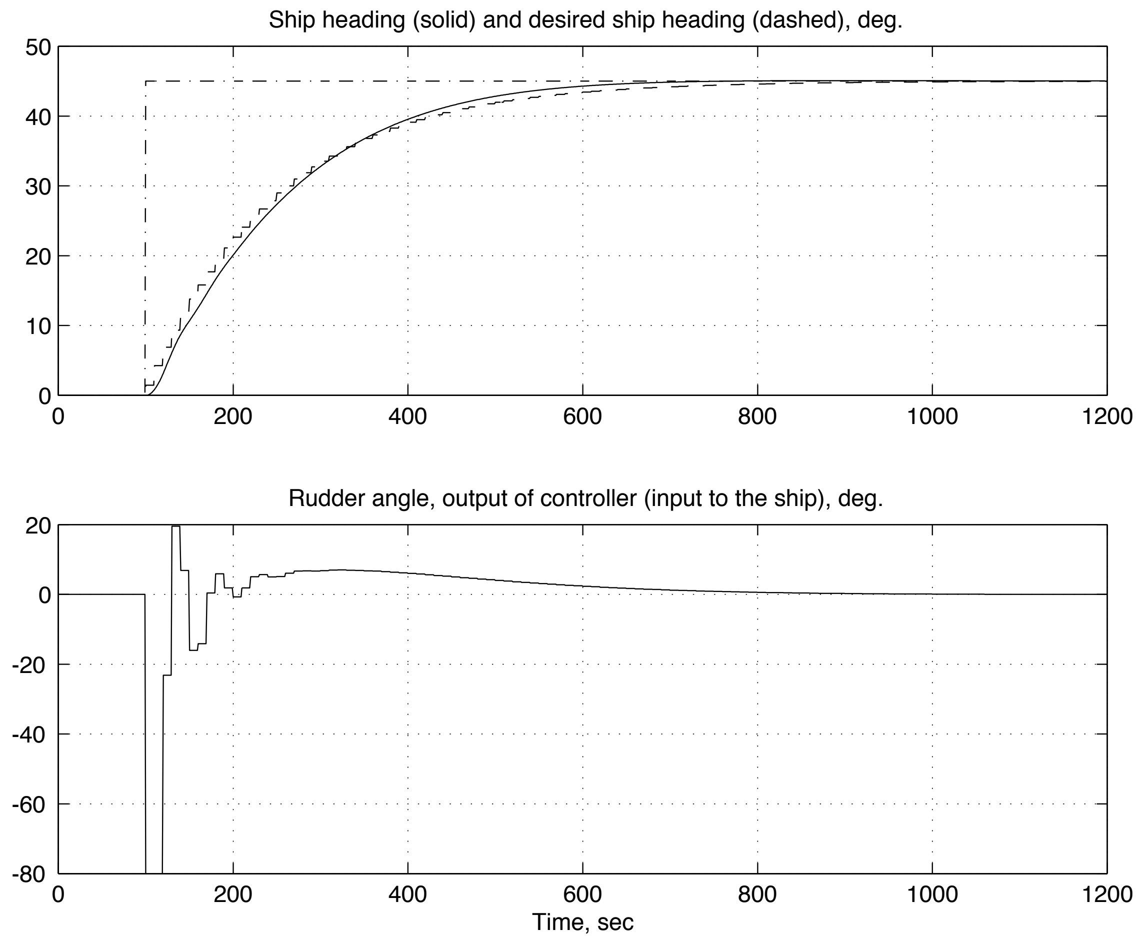

Figure 177: Closed-loop tanker response for best (K_p, K_d) gains as indicated by the response surface in Figure 176.

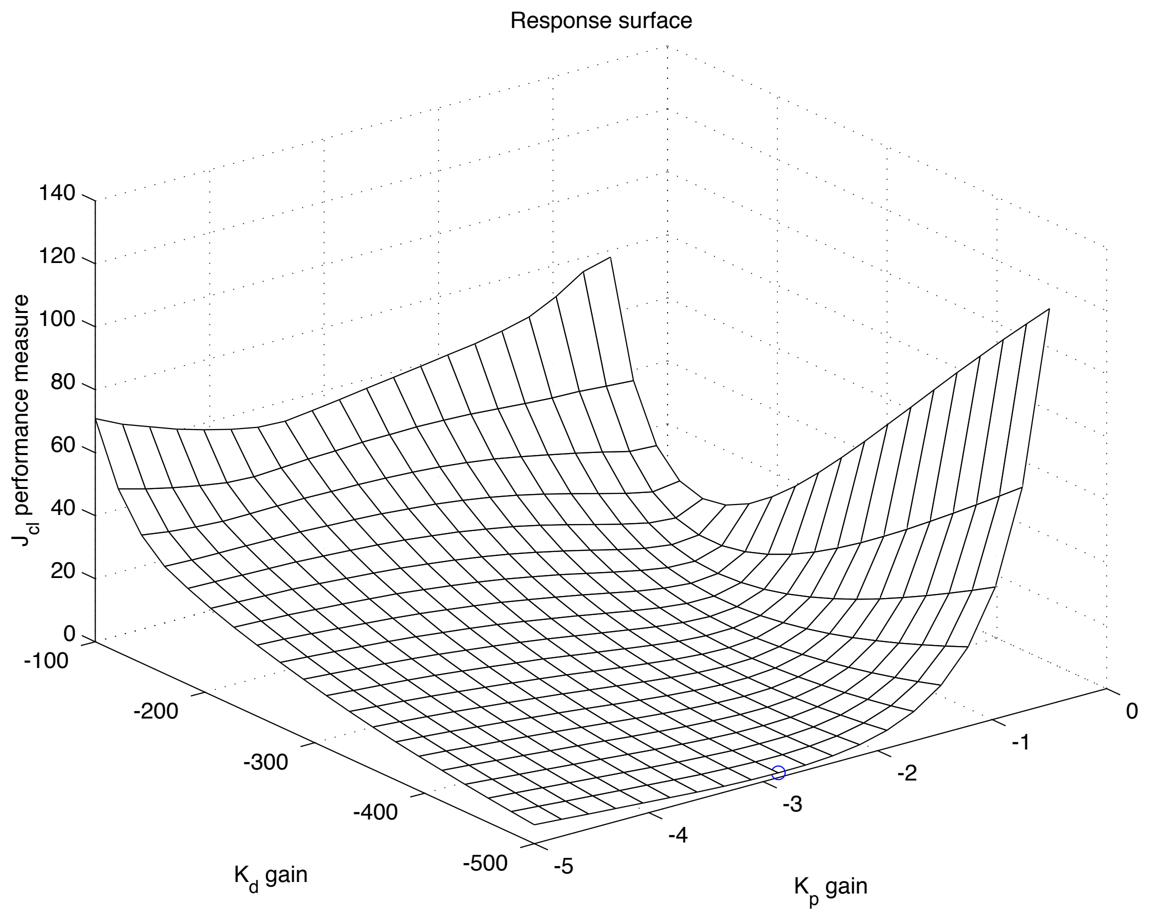

Figure 178: Response surface of J_{cl} for PD controller designs, "full" tanker ship (best gains are $K_p = -2.8684$ (increased–make sense?) and $K_d = -500$).

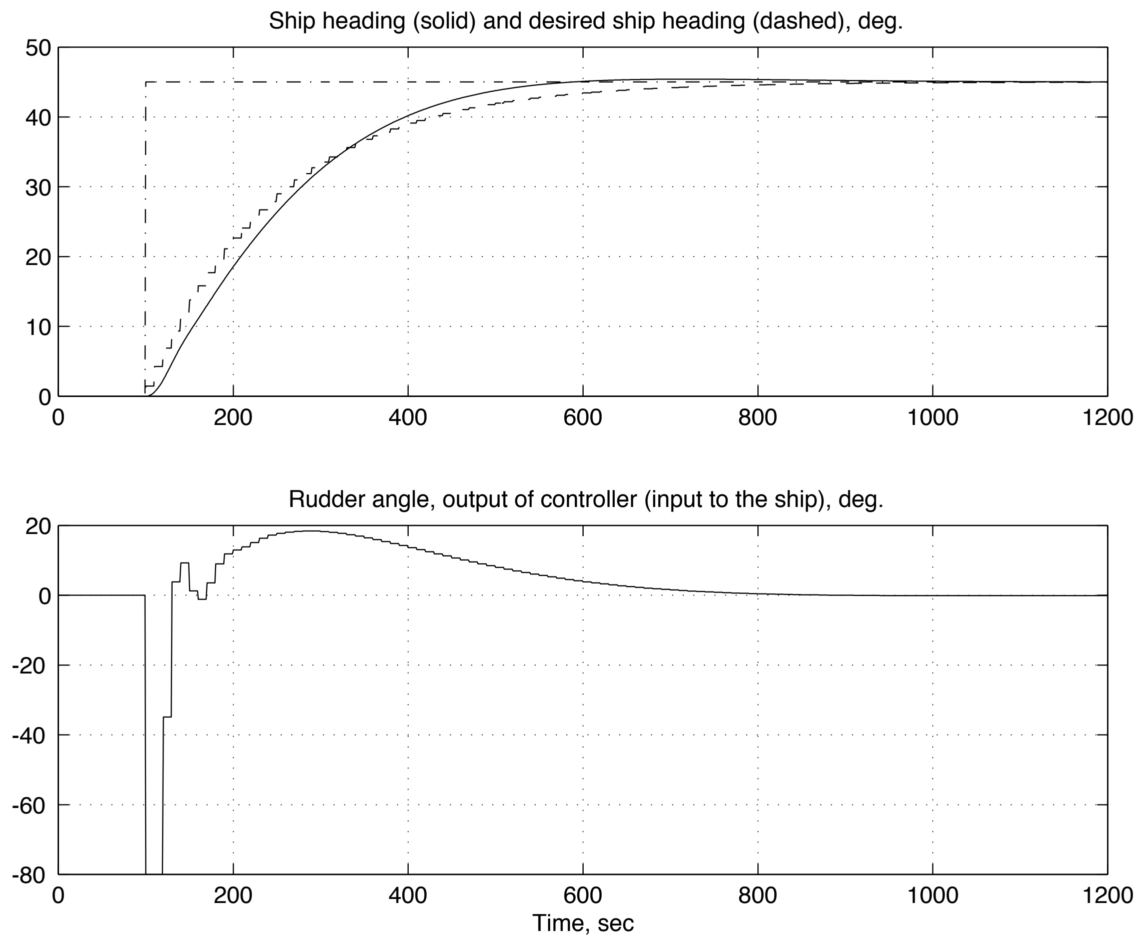

Figure 179: Closed-loop tanker response for best (K_p, K_d) gains as indicated by the response surface in Figure 178.

Design Optimization Over Multiple Response Surfaces

- ➙ Can construct response surfaces for different ψ_r, noise, wind, speeds, ship weights, etc.
  - Theoretically infinite number of response surfaces, each one corresponding to the infinite number of possible conditions that the plant (environment) can present.
- ➙ Theoretically, you could then combine the response surfaces to obtain a single performance measure and then find the minimum point on the resulting surface and call it the "best design" for all the conditions tested.
- ➙ Example: Sum the response surfaces for the full and ballast conditions to get a "combined" response surface (best PD controller if we will encounter both situations with equal probability?)

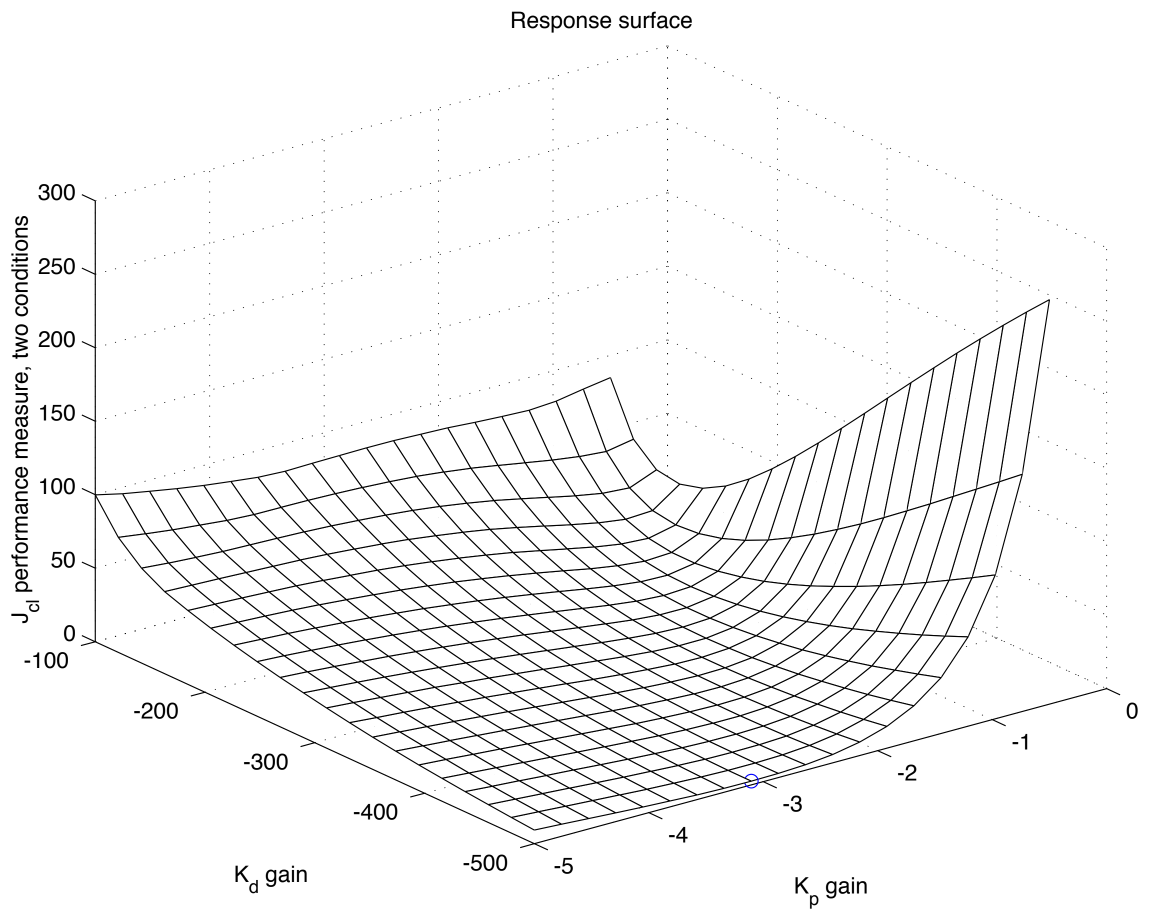

Figure 180: Response surface resulting from the sum of the response surfaces for PD controller designs for the “ballast” and “full” conditions (best gains $K_p = -3.1053$ (in between) and $K_d = -500$).

- ➙ Fundamental trade-offs in design.
- ➙ Performance and hence optimal design choices are different for different plant conditions; hence, finding the “best” single design entails allocating good performance across different situations.
- ➙ If consider stochastic effects, the set of “optimal gains” indicated by one surface is highly unlikely to perfectly correspond to the “optimal gains” suggested by another surface.
- ➙ Can optimize design for narrow range of conditions, but performance will certainly suffer under conditions other than what the design is for.
- Fundamental trade-offs in robustness.

RSM Issues

➙ Choosing Design Points: "Design of Experiments": Cannot use fine grids. Must choose points carefullly. if many dimensions, but could pick two points per dimension—"2^p factorial design"

- Example: Would this have worked for the tanker ship controller design?

➙ Response Surface Construction is Function Approximation: RSM uses "first" or "second order" models, and least squares is used to fit the models to the data. Response surface is the tuned approximator shape.

- Basic principles of approximation apply.
- DoE is training data set choice.
- Use response surface to consider intermediate design points.
- Pick optimal design off response surface.

RSM for Learning System Design

Design Example: Robust Approximator Size Design

- Different G lead to different estimators (e.g., using our theme problem and BLS).

➙ For a fixed size G and fixed unknown nonlinear mappping, what is the best choice for p—approximator size?

- Use RSM for $p \leq M = 100$. Choose $M_\Gamma = 200$.
- From earlier studies, need $p \in [4, 40]$.
- For each p use BLS to construct $N_t = 100$ approximators (TS fuzzy systems, place centers uniformly for each p) and compute the mean squared error relative to the test set.

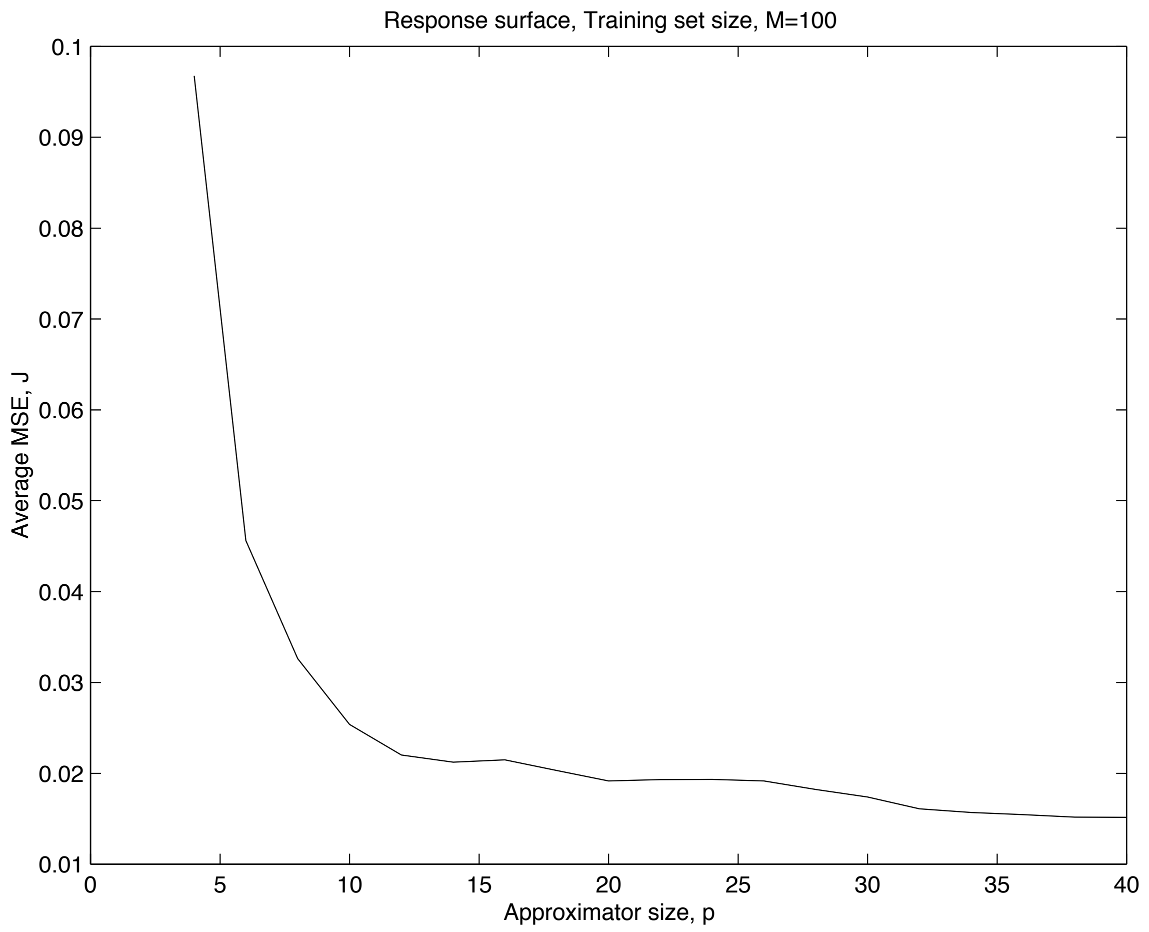

Figure 181: Average MSE response surface for approximator size, $N_t = 100$.

- MSE is a random variable so must pick N_t large enough

★ Typical "knee" in the curve-for increasingly large p little is gained in approximation accuracy.

➙ What happens for very large p?

Design Example: Instinct-Learning Balance in an Uncertain Environment

➙ Example: Estimator design problem, with focus on analogies with learning instinct balance.

- Will discuss problem as if we use evolution, but here really just study the shape of the underlying "fitness function" (inverse of it) which is really a response surface.

Estimator Design Problem, Analogies with Instincts and Learning

➙ Learning: "Any process through which experience at one time can alter an individual's behavior at a future time" [5] (i.e., any system with memory has the potential to be a learning system).

➙ Define *learning ability* to be the number n of values that can be remembered by an organism in performing some activity during a life time.

- ➙ The value of n affects how the organism performs the activity (e.g., in helping it to perform the activity better).
- ➙ The value of n is how much memory the organism has, a key component of learning.
- ➙ Organism can sense some aspect of its environment and store this in a (scalar) variable $y(k, \ell)$ at time k, $k = 1, 2, \ldots, N_L$, for "generation" ℓ, $\ell = 1, 2, \ldots, N_g$.
  - N_L is life time length
  - N_g is the number of generations of evolution (here generations simply correspond to repeated runs of lifetimes of a single individual organism).
- ➙ We use a very simple model of the "environment" that the organism performs the task in where

$$y(k, \ell) = x(k, \ell) + z(k, \ell)$$

- Here, we will assume that $x(k,\ell) = \bar{x} = 2$ is a constant that the organism wants to estimate in order to be successful in performing some activity (which enhances it reproductive success)
- $z(k,\ell)$ is noise representing uncertainty in the environment.
- $z(k,\ell)$ is drawn from a normal distribution with zero mean and a variance

$$\sigma_z^2(k,\ell)$$

that could change over the life time, or over generations.
- First, consider the case where $\sigma_z^2 = 0.5$ for all k and ℓ.

- ➙ Our organism can sense and store (remember) n values of $y(k,\ell)$ at each time k and that it estimates $x(k,\ell)$ via

$$\hat{x}(k,\ell) = \frac{1}{n} \sum_{j=k-n+1}^{k} y(j,\ell)$$

- Uses a "sliding window" of n values that it computes the mean of as an estimate of the x value.

- ➙ Instincts are the intial conditions $\hat{x}_0$ of the estimator.

- At $k = 1$ we need initial values at $k-1, k-2, \ldots, k-n+1$.

- Assume that *all* past intial values needed before $k = 1$ are equal to $\hat{x}_0$.

- To define fitness (seek minimization):

$$\begin{aligned} J(\hat{x}_0, n, \ell) \quad &= \quad w_1 n + w_2 \frac{1}{N_L} \sum_{j=1}^{N_L} (x(j,\ell) - \hat{x}(j,\ell))^2 \\ &\quad + w_3 \exp\left(- \frac{(x(1,\ell) - \hat{x}_0)^2}{\sigma^2} \right) \end{aligned} \tag{89}$$

➜ Three terms:

1. $w_1 n$, quantifies the cost of remembering more sensed values.

2. Mean-squared estimation error over its entire life time (one with particular insincts $\hat{x}_0$ and learning abilities n). Better instincts should lead to better estimation performance, and using more values to compute the mean (higher n) should also improve performance.

3. Cost of accurate instincts (e.g., physiological costs) via a Gaussian function centered at the actual value of the variable to be estimated.

Response Surfaces for Optimal Instinct-Learning Balance

- Use $w_1 = 0.01$, $w_2 = 1$, and $w_3 = 0.05$

★ Get optimal point

$$\hat{x}_0^* = 1.6410, \; n^* = 5$$

➙ Since very good instincts are costly and learning abilities are relatively inexpensive (low w_1), it is best to have a somewhat accurate instinct, coupled with an ability to sense and remember several values from the environment.

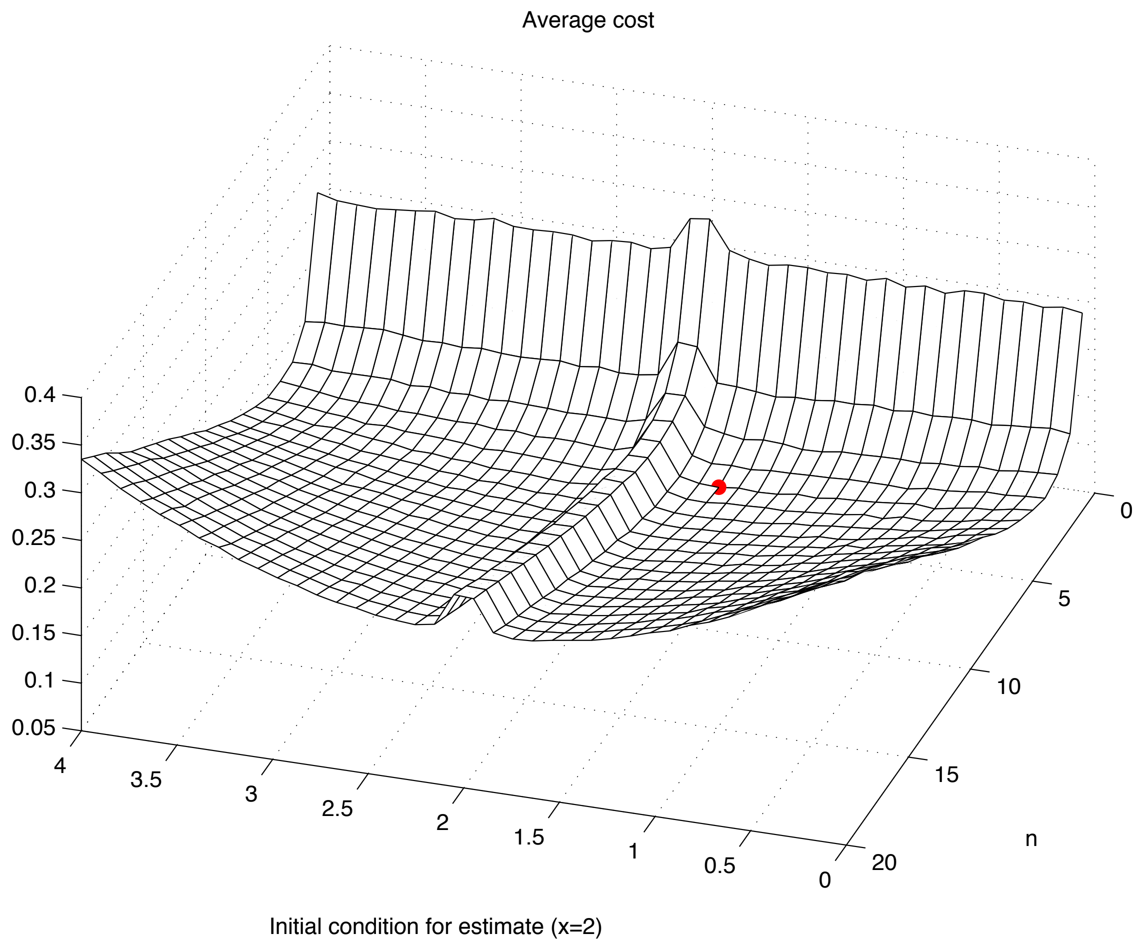

Figure 182: Response surface (fitness landscape) for instinct-learning balance problem, $\sigma_z^2 = 0.5$, optimum point shown with a dot ($N_g = 100$).

- Next, let $w_1 = 0.1$, expensive n.

★ Now, get

$$\hat{x}_0^* = 1.5385, \ n^* = 2$$

➙ Now better to use fewer stored values since storage costs more.

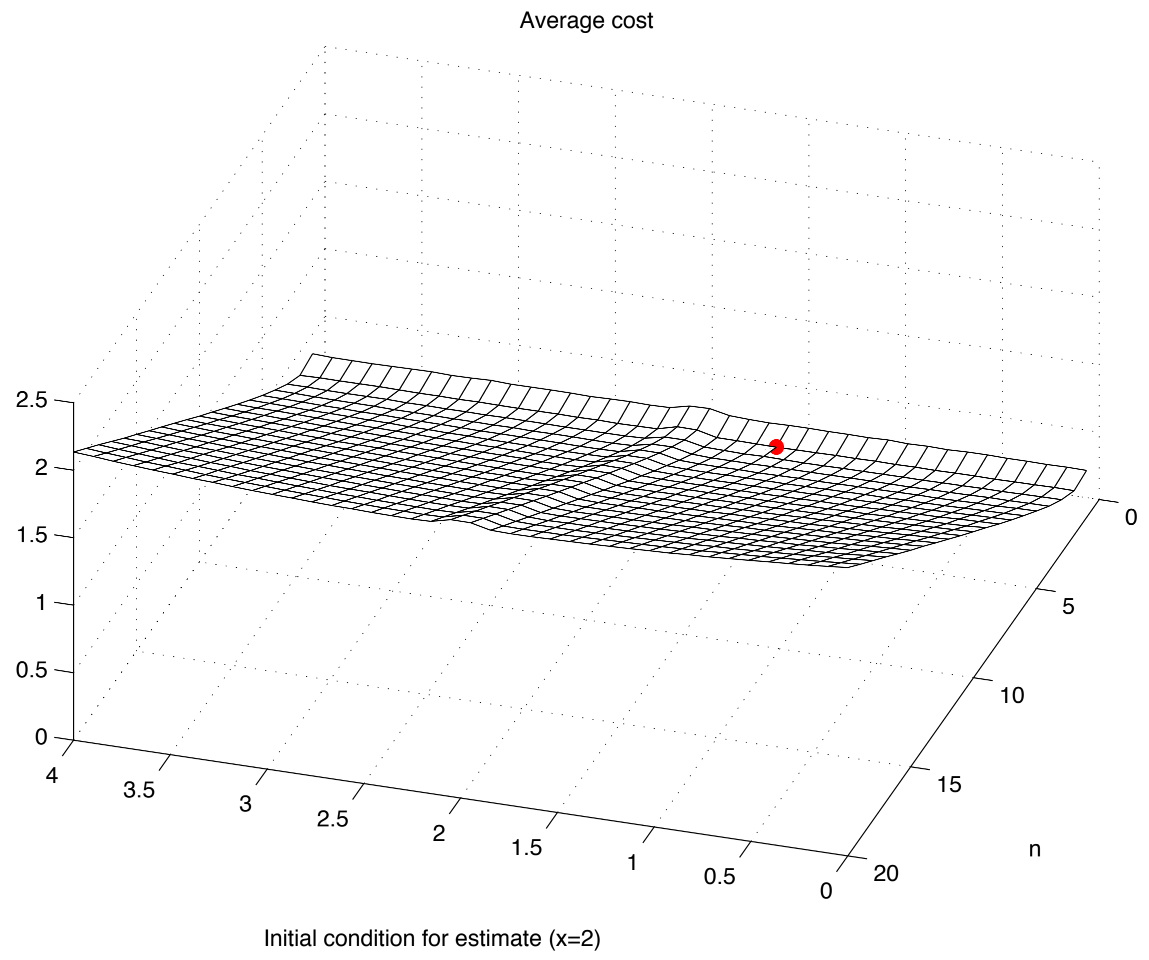

Figure 183: Response surface (fitness landscape) for instinct-learning balance problem, $\sigma_z^2 = 0.5$, $w_1 = 0.1$, optimum point shown with a dot.

- Next, consider an environmental change, $\sigma_z^2 = 0.75$,
- Also, use $w_1 = 0.01$ and $w_2 = 1$ (same as above), but $w_3 = 0.005$—cheap to have good instincts.

★ Gives

$$\hat{x}_0^* = 2.2564, \; n^* = 8$$

➙ Get better instincts, use of higher n.

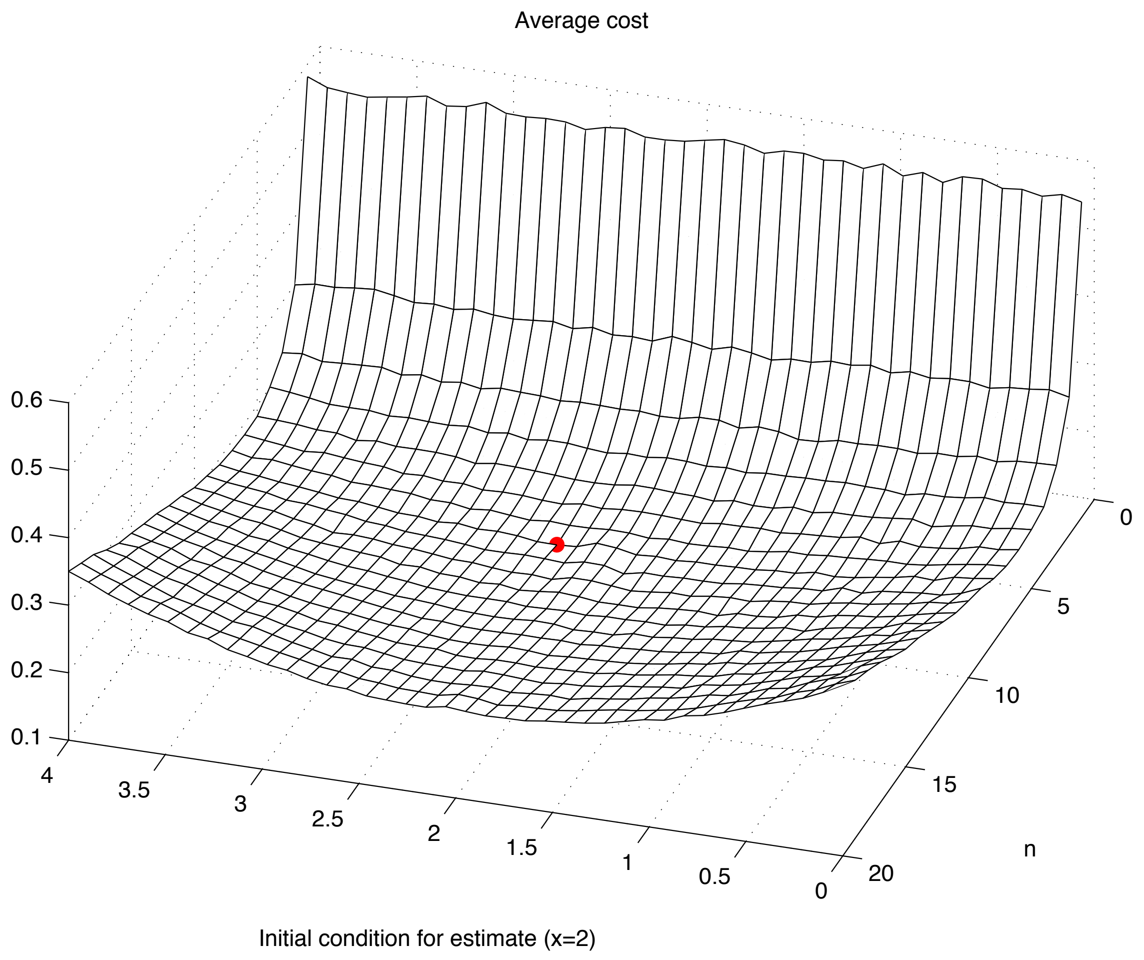

Figure 184: Response surface (fitness landscape) for instinct-learning balance problem, $\sigma_z^2 = 0.75$, $w_3 = 0.005$, optimum point shown with a dot.

Nongradient Optimization

Pattern and Coordinate Search

Approximations to the Gradient

➙ We can use a "central difference formula" to approximate a gradient with respect to each parameter (that we denote with $g_i(j)$, the approximation of the gradient with respect to the i^{th} parameter at the j^{th} iteration).

$$\frac{\partial J(\theta(j))}{\partial \theta_i} \approx \frac{1}{2c}\left(J(\theta(j)+ce_i) - J(\theta(j)-ce_i)\right) = g_i(j) \qquad (90)$$

where c is a positive scalar and e_i is the i^{th} column of the $p \times p$ identity matrix.

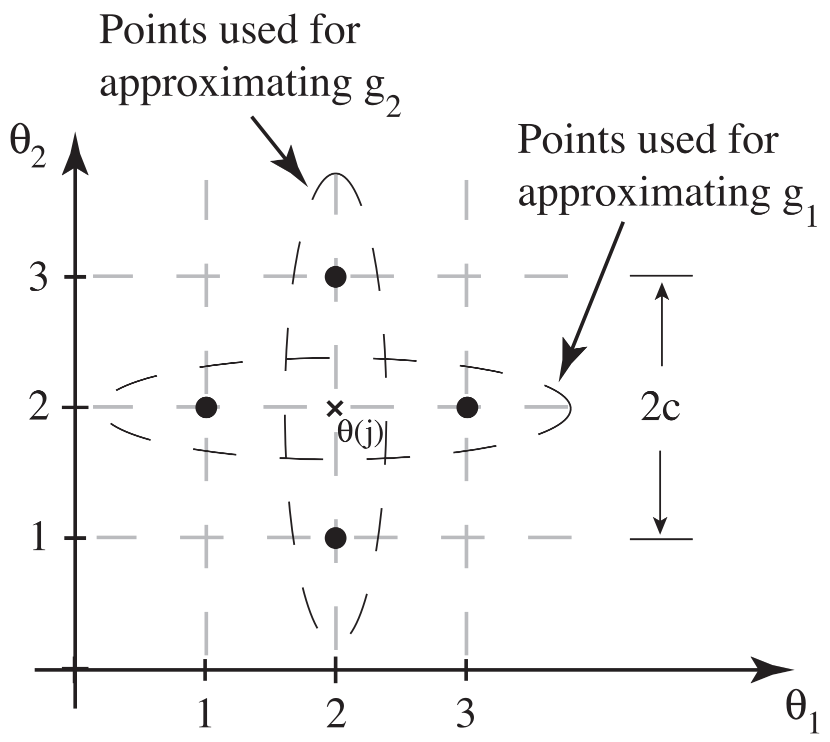
Points used for approximating g_2
Points used for approximating g_1
θ_2
θ_1
3
2
1
1
2
3
$\theta(j)$
2c

Figure 185: Pattern of points for approximation to the gradient, $p = 2$ case.

Simple Pattern Search Methods

- ➙ Basic idea: Compute cost at each point in a pattern of "exploratory" points around the current estimate $\theta(j)$ and then decide how to move the estimate and pattern of points at the next iteration so as to reduce the cost.

- ➙ Sometimes pattern allows for *approximations to the gradient in a region*

- ➙ At iteration j we start with a pattern

$$P = \{\theta^0, \theta^1, \theta^2, \ldots\}$$

  and the method generates another set of points for iteration $j+1$, $j = 0, 1, 2, \ldots$.

- Suppose that θ^0 is always $\theta(j)$ (algorithm maintains this).
- $\theta(j)$ is the current estimate of an optimum point.

➙ Points in $P(j)$ are candidate solutions considered in "parallel"

- Let C denote a matrix whose columns specify perturbations to the current estimate $\theta(j)$ to specify the pattern P.
- Let $\theta_s^i(j)$ denote the i^{th} exploratory perturbation from $\theta(j)$, $i = 1, 2, \ldots, |C|$.
- How to specify a pattern?

<u>Evolutionary Operation Using Factorial Designs Method</u>

- Columns of the matrix C are chosen to have elements that are all possible combinations of $\{-1, 1\}$ and one column of zeros.

➙ This is the 2^p corners of a hypercube centered at $\theta(j)$, plus a column of zeros, which represent the center point $\theta(j) = \theta^0(j)$ (what is the relationship to the design of experiments choice for response surface methodology?).

- $|C| = 2^p + 1 = |P|$.

- Let c^i denote the i^{th} column of C.
- As an example, if $p = 2$ then

$$C = \begin{bmatrix} 1 & 1 & -1 & -1 & 0 \\ 1 & -1 & -1 & 1 & 0 \end{bmatrix}$$

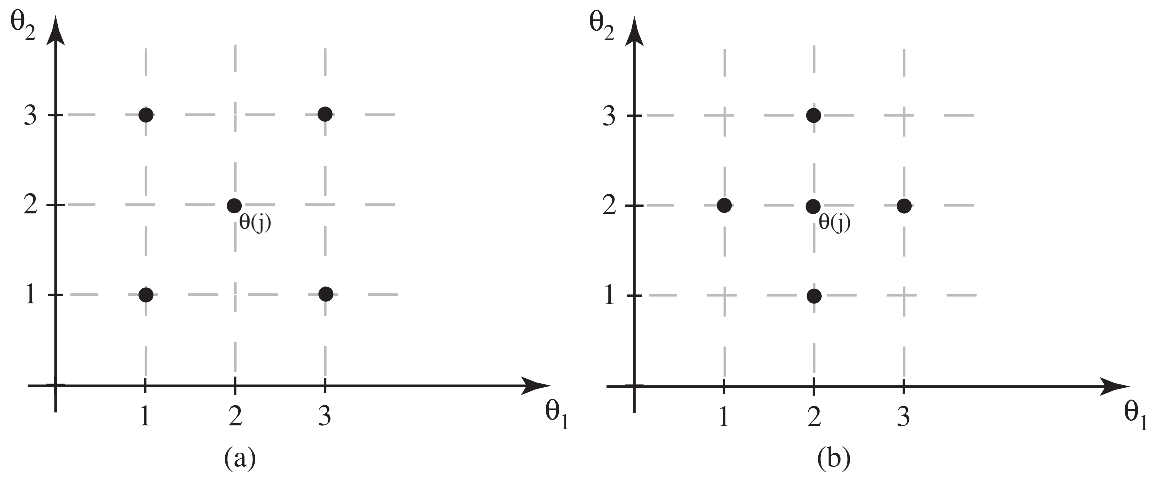

Figure 186: The pattern of points for the $p = 2$ case in evolutionary operation using factorial designs is shown in (a), and in (b) for simple coordinate search.

➙ For what we will call "simple coordinate search"

$$C = [I \;\; -I \; 0]$$

where I is the $p \times p$ identity matrix, so that C is a $p \times 2p + 1$ matrix.

Pattern Search Algorithm

- Initial pattern $P(0)$ given via the choice of C and $\theta(0) = \theta^0(0)$.
- Use γ_c to specify how the pattern should contract at the next iteration if a lower cost exploratory point was not found in the current pattern (usual choice is $\gamma_c = \frac{1}{2}$).
- If a lower cost point was found on the current pattern, then the pattern is not contracted and the parameter λ_j, for example with $\lambda_0 = 1$, will be used to specify the actual contraction that the algorithm takes at the next step.
- ➙ Let $\theta_s(j)$ denote the perturbation from $\theta(j)$ that is chosen at step j as the *best* point in the pattern of exploratory points (if $\theta_s(j) = 0$ at some step this represents that no better point was found so $\theta(j+1) = \theta(j)$ and the pattern is contracted).
- At each iteration let J_{min} denote a scalar that is the lowest cost point found so far in computing the cost at each point in

the pattern.

➙ Algorithm for evolutionary operation using factorial designs is:

1. For $j = 0, 1, 2, \ldots$:
2. Compute $J(\theta(j))$.
3. Exploratory moves:
   (a) Let $\theta_s(j) = 0$, $\rho_j = 0$, and $J_{min} = J(\theta(j))$.
   (b) For $i = 1, 2, \ldots, |C| - 1$:
      – Compute $J(\theta^i(j))$ where

$$\theta_s^i(j) = \lambda_j c^i$$

      and

$$\theta^i(j) = \theta(j) + \theta_s^i(j)$$

      – If $J(\theta^i(j)) < J_{min}$ let $\rho_j = J(\theta(j)) - J(\theta^i(j))$, $J_{min} = J(\theta^i(j))$, and choose $\theta_s(j) = \theta_s^i(j)$.
      – Next i.

4. Update/contract:
   - If $\rho_j > 0$ then a better point was found on the pattern so let $\theta^0(j+1) = \theta(j+1)$ where
   $$\theta(j+1) = \theta(j) + \theta_s(j)$$
   and let
   $$\lambda_{j+1} = \lambda_j$$
   so that we do not contract since this pattern size seems to be making good progress.
   - If $\rho_j \leq 0$ a better point was not found on the pattern so let
   $$\theta(j+1) = \theta(j)$$
   and contract the pattern by letting
   $$\lambda_{j+1} = \gamma_c \lambda_j$$
5. Next j.

Algorithm Complexity and Convergence

- At each iteration need to compute $|C|$ cost values.
- Under mild restrictions these pattern search algorithms can be shown to possess certain convergence properties.
  - Can be shown that

$$\lim_{j\to\infty} ||\nabla J(\theta(j))|| = 0$$

Coordinate Descent

- Coordinate descent via line search involves iteratively cycling across all the dimensions and performing a one-dimensional line search each time.
- Can also use gradient approximations

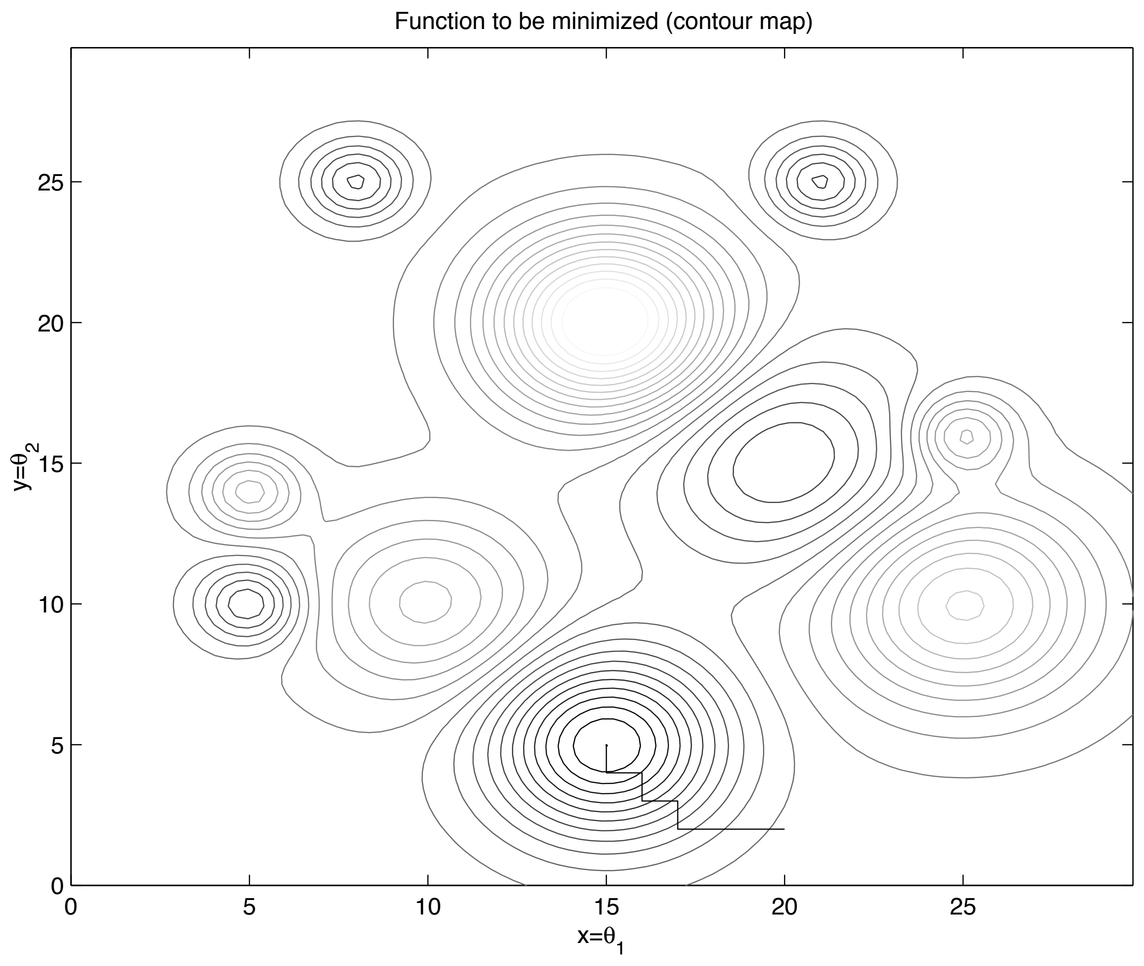

Figure 187: Parameter trajectories for simple coordinate search optimization problem, plotted on the contour plot of the cost function.

The Multidirectional Search Method

- Pattern search method, borrows some ideas from "Nelder-Mead Simplex Method" (which is more popular, but does not have guaranteed convergence)
  - Consider minimizing $J(\theta)$, $\theta \in \Re^p$, and assume that J is continuous in θ and that $\nabla J(\theta)$ *exists*.
- Multidirectional search iterates on a simplex (a "convex hull") of $p+1$ candidate solutions

$$P = \left\{\theta^0, \theta^1, \ldots, \theta^p\right\} \subset \Re^p$$

- Suppose that *at each iteration* $\theta^0(j)$ is the best vertex of $P(j)$ so

$$J(\theta^0(j)) \leq J(\theta^i(j)), \; i = 0, 1, \ldots, p$$

- The goal at each iteration is to find another candidate solution with a cost that is strictly less than $J(\theta^0(j))$.

- To do this, it searches along lines passing through $\theta^0(j)$ and its p adjacent vertices.

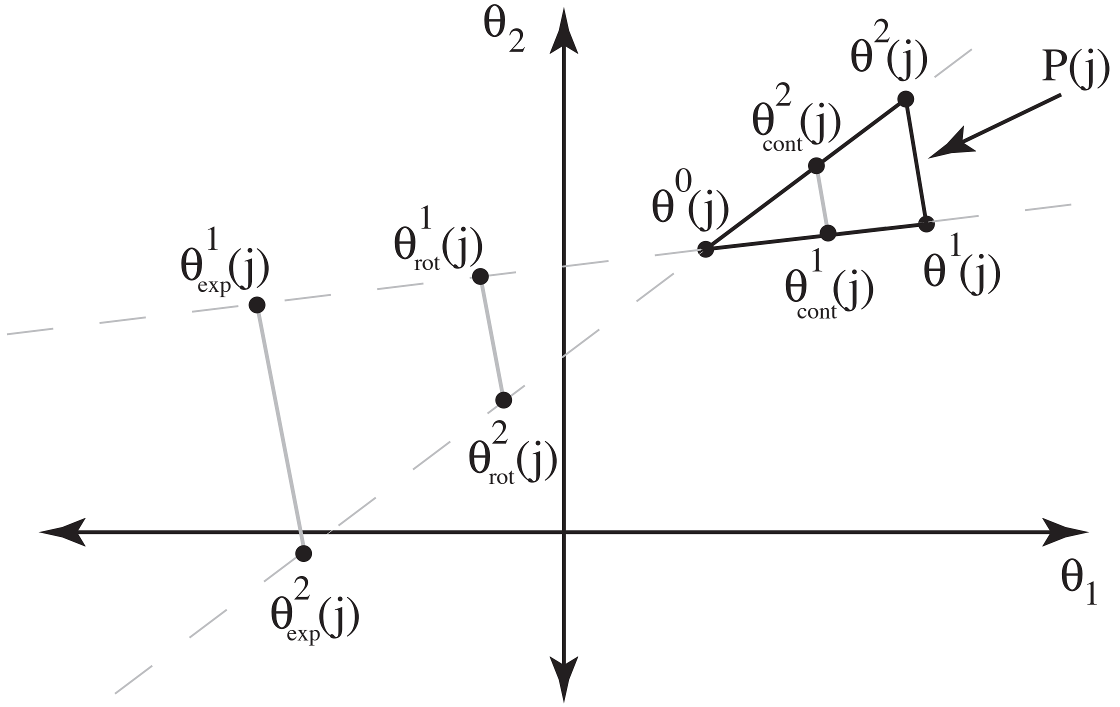

Figure 188: Illustration of search directions and possible next simplices for multidirectional search for $p = 2$.

➙ Simplex $P(j)$ has its vertices connected by black lines.

- Steps:
  1. "Rotation" step: $\theta^1(j)$ and $\theta^2(j)$ are reflected about $\theta^0(j)$ on the gray dashed lines to obtain $\theta^1_{rot}(j)$ and $\theta^2_{rot}(j)$, respectively (hence the simplex $P(j)$ is *rotated* about $\theta^0(j)$).
  2. If the cost at $\theta^i_{rot}(j)$, $i = 1, 2$, is better than the one at $\theta^0(j)$, then an "expansion" is computed by reflecting about $\theta^0(j)$, but more than twice as far as in the rotation step, to produce $\theta^i_{exp}(j)$, $i = 1, 2$.
  3. If the cost at any of these two new vertices, $\theta^i_{exp}(j)$, $i = 1, 2$, is better than the cost at $\theta^i_{rot}(j)$, $i = 1, 2$, then accept the minimum cost vertex from the expansion as the new minimum cost vertex ($\theta^0(j+1)$) and it, with $\theta^i_{exp}(j)$ (i so that $\theta^i_{exp}(j) \neq \theta^0(j+1)$) and $\theta^0(j)$ define the new simplex $P(j+1)$.

4. If expansion does not result in points with lower cost than the costs for the new vertices from the rotation step, then you accept the minimum cost vertex from the rotation step as $\theta^0(j+1)$ and it together with $\theta^0(j)$ and the other vertex from the rotation step form the new simplex.
5. Now, if $\theta^i_{rot}(j)$, $i=1,2$, did not result in a lower cost than the one at $\theta^0(j)$ (so expansion was not used), then you take a "contraction" step where you move the adjacent vertices $\theta^1(j)$ and $\theta^2(j)$ towards $\theta^0(j)$ along the gray dashed lines to the points $\theta^i_{cont}(j)$, $i=1,2$.

- The process of rotation, expansion, and contraction repeats until a stopping criterion is satisfied.
- Explicit analytical gradient information is not available, but there is a type of approximation to the gradient that is being used

- Simplex methods try to ignore “noise” perturbations on the cost by using a region-based approximation to the gradient.

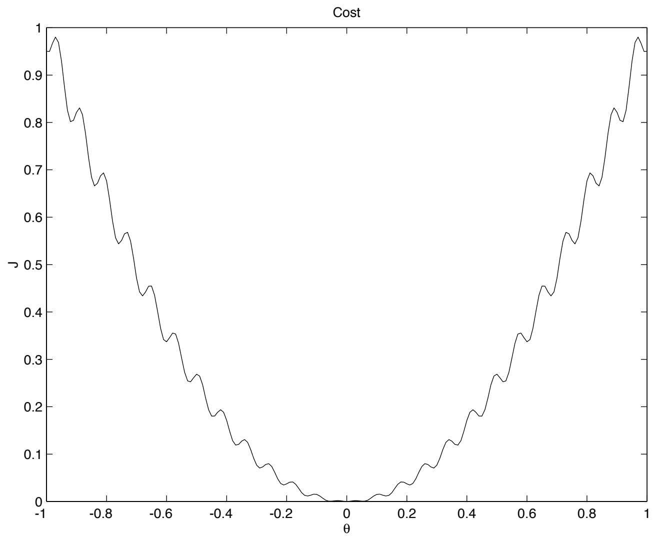

Figure 189: An example cost function with multiple local minima.

Algorithm Complexity and Convergence

- If get cost reduction need $2p$ cost evaluations per iteration, but may need many steps if do not get reduction.
- Under mild restrictions the multidirectional search method possesses convergence properties like the earlier pattern search method.

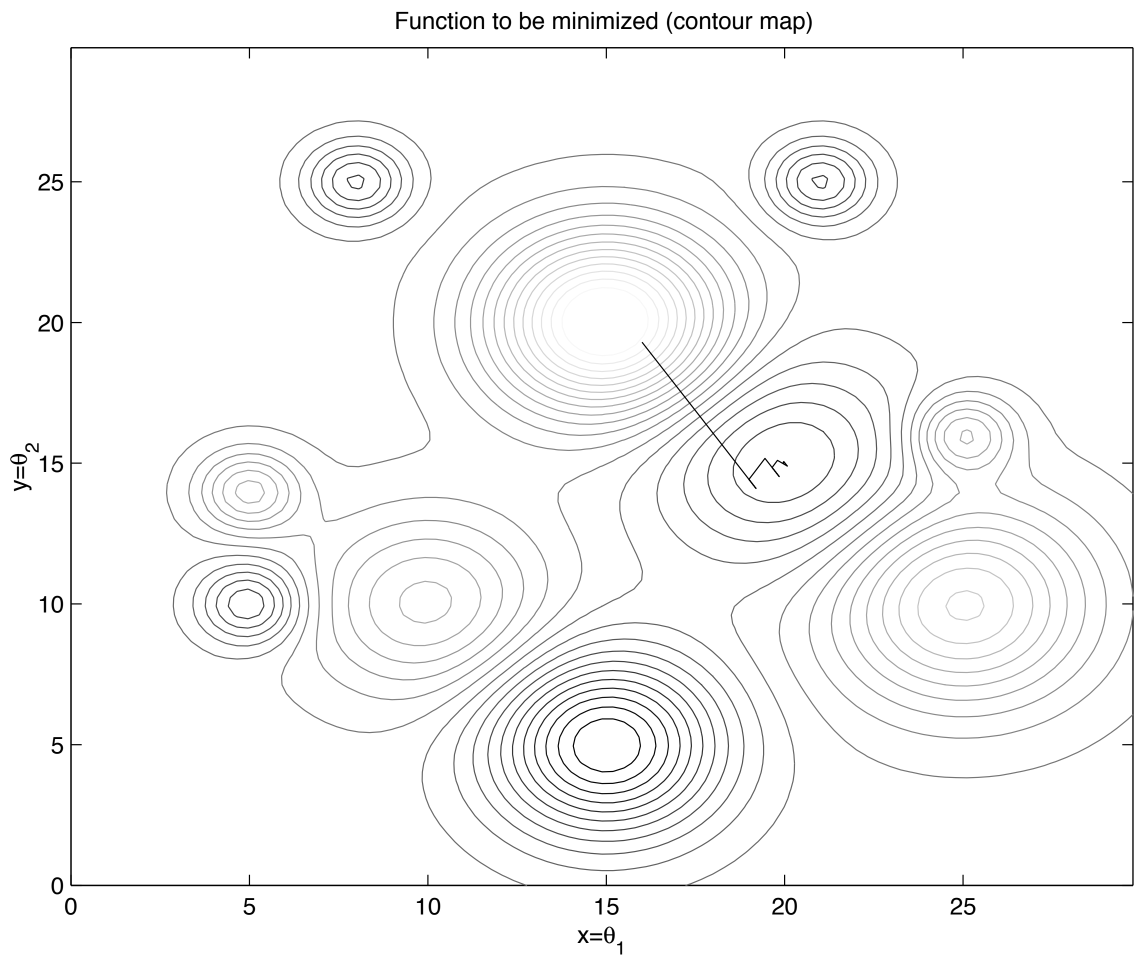

Figure 190: Multidirectional search example, parameter trajectory on the contour plot of the cost function.

Simultaneous Perturbation Stochastic Approximation Algorithm

➙ Minimize $J(\theta)$ by adjusting $\theta \in \Re^p$.

- Gradient $\nabla J(\theta)$ is not known analytically and cannot be measured.
- Can compute or measure $J(\theta)$, but get

$$J_n(\theta) = J(\theta) + w$$

where w is noise, a noisy cost.

➙ Parameter update formula

$$\theta(j+1) = \theta(j) - \lambda_j g(\theta(j), j) \tag{91}$$

where $g(\theta(j), j) \in \Re^p$ is an estimate of $\nabla J(\theta(j))$ at $\theta(j)$, and $\lambda_j > 0$.

➙ For $i = 1, 2, \ldots, p$ the gradient approximation is chosen as

$$g_i(\theta(j), j) = \frac{J_n(\theta(j) + c_j\Delta(j)) - J_n(\theta(j) - c_j\Delta(j))}{2c_j\Delta_i(j)} \tag{92}$$

where $c_j > 0$ for all j and

$$\Delta(j) = \begin{bmatrix} \Delta_1(j) \\ \vdots \\ \Delta_p(j) \end{bmatrix}$$

is a random perturbation vector.

- The components of the vector $\Delta(j)$ should be independently generated from a zero mean probability distribution
- A theoretically valid choice is a Bernoulli ± 1 distribution for each ± 1 outcome.
- In this way, the $\theta(j) \pm c_j\Delta(j)$ lie on corners of a hypercube

centered at $\theta(j)$.

- For $p = 2$,

$$\Delta(j) \in \left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\}$$

- In general, there are 2^p possible $\Delta(j)$ values. For example, see Figure 191.

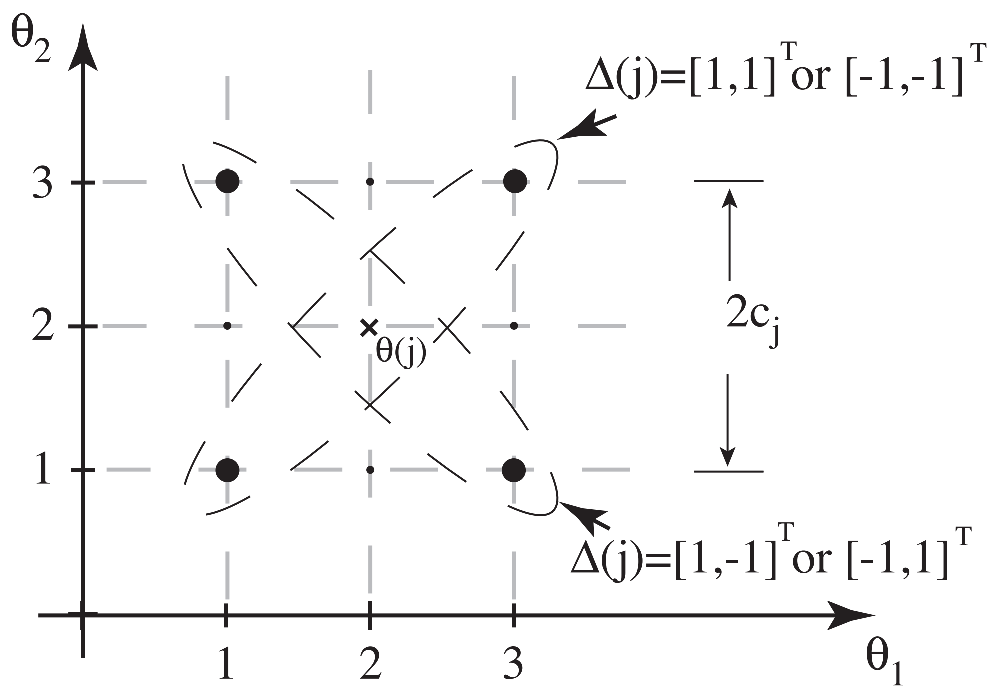

Figure 191: Illustration of what values are used in computing an approximation to the gradient in the SPSA method.

- For this example, if $\Delta(j) = [1, 1]^\top$, then

$$\begin{aligned} \theta^+(j) &= \theta(j) + c_j \Delta(j) = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \\ \theta^-(j) &= \theta(j) - c_j \Delta(j) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{aligned}$$

which are the upper right and lower left corners (denoted with large black dots) of the square centered at $\theta(j)$ in Figure 191 (and these values are swapped if $\Delta(j) = [-1, -1]^\top$).

- ➜ Example: $J_n(\theta) = \theta^\top \theta$ with no noise. Can get cost increase!
  - The c_j and λ_j sequences of values decrease at each iteration, the size of the hypercube and the size of changes to the parameter values at each iteration do also.

Algorithm Complexity and Convergence

- For the Bernoulli $\pm$ distribution for Δ there are 2^p possible $\Delta(j)$ vectors and $2^{(p-1)}$ possible diagonals on the hypercube centered at $\theta(j)$.
- Consider the no noise case.
- There are in general $2^{(p-1)}$ possible approximations to the gradient at each iteration, and hence one of $2^{(p-1)}$ possible update directions is chosen at each iteration. However, only 2 cost calculations are made at each step.
- Stochastic gradient method: analytical gradient information used, and one of an infinite number of possible update directions is chosen. Constraints on the perturbations to the gradient; here, the use of points on the hypercube centered at $\theta(j)$ constrains the size and directions of the update.

- The “Keifer-Wolfowitz finite difference stochastic approximation” (FDSA) algorithm computes the cost at $2p$ points (two per dimension).
- See Figure 191.

➙ With no noise, only one possible update direction per iteration.

- It has been shown that under reasonably general conditions SPSA and FDSA achieve the same level of statistical accuracy for a given number of iterations and

$$\frac{\text{Number of measurements of} J_n(\theta) \text{in SPSA}}{\text{Number of measurements of} J_n(\theta) \text{in FDSA}} \to \frac{1}{p}$$

  as the number of measurements gets large.

- Important for large p.

Guidelines for Choosing SPSA Parameters

➙ First, choose

$$\lambda_j = \frac{\lambda}{(\lambda_0 + j)^{\alpha_1}}$$

where $\lambda > 0$, $\lambda_0 > 0$, and $\alpha_1 > 0$, and

$$c_j = \frac{c}{j^{\alpha_2}}$$

where $c > 0$ and $\alpha_2 > 0$.

- However, if the θ_i have very different magnitudes you may want to use a different λ_j for each of the p dimensions.
- Some actual values that have been found useful in applications are $\alpha_1 = 0.602$ and $\alpha_2 = 0.101$ which are effectively the lowest allowable ones that satsify theoretical conditions.
- However, values $\alpha_1 \in [0.602, 1)$ and $\alpha_2 \in [0.101, \frac{1}{6}]$ may work also.

- In fact $\alpha_1 = 1$ and $\alpha_2 = \frac{1}{6}$ are the “asymptotically optimal” values so if the algorithm runs for a long time it may be beneficial to switch to these values.
- With this choice, if the noise is significant you may need to choose λ smaller and c larger than in a low-noise case.
- With the Bernoulli ± 1 choice, set c to a level that is approximately the standard deviation of the noise $w(j)$ to keep the components of $g(\theta(j), j)$ from being too large in magnitude.
- If there is no noise term $w(j)$, then you should choose some small value $c > 0$.
- You can choose λ_0 to be approximately 10% of the maximum number of iterations and λ to try to achieve a certain amount of change in the cost function values at each iteration.

SPSA for Decision-Making System Design: Examples

- PD controller design, but with sensor noise. Climbs down surface we showed earlier.
- Can we exploit one advantage of the SPSA—that is should be efficient for high p problems, with noise cost, and no gradient information?
- Many possibilities!

Parallel, Interleaved, and Hierarchical Methods

➙ "Parallel" methods ("set-based" techniques) are implemented by executing N copies of an optimization method.

- Is information shared between algorithms?

➙ "Interleaving" involves alternating between the use of different algorithms as the optimization process proceeds.

➙ "Hierarchical" methods employ one algorithm to supervise the operation of several algorithms.

Set-Based Stochastic Optimization for Design

- SPSA considers two designs (cost evaluations) per iteration
- GAs consider S designs at each point
- Are there other "set-based" stochastic optimization methods?

➙ Could they model aspects of evolutionary algorithms, without the overhead of the encoding/decoding used there?

A Set-Based Stochastic Optimization Method

- ➙ Use a set (population) of design points,

$$P(k) = \{\theta^i(k) | i = 1, 2, \ldots, S\}$$

  with S members, with initial value $P(0)$.

  - Cost measure J for each member, want to minimize.
  - At each k, compute cost for each member (could parallelize).

- ➙ Select "best" design and name it i^*.

➙ Create next generation via

$$\theta^{i^*}(k+1) = \theta^{i^*}(k)$$

("elitism") and for $i \neq i^*$,

$$\theta^{i}(k+1) = \theta^{i^*}(k+1) + \begin{bmatrix} \beta_1 r_1 \\ \vdots \\ \beta_p r_p \end{bmatrix}$$

where r_i, $i = 1, 2, \ldots, p$ are random numbers drawn from normal distributions with zero mean and unit variances.

- The parameters β_i, $i = 1, 2, \ldots, p$ scale the variances.

➙ Generate a "cloud" of design points centered at the best design point to form the next generation.

- What are effects of β_i? Range of exploration.
- Can use projecction to keep values in range.
- To avoid getting "stuck," a mutation-type mechanism was added. Select one $i \neq i^*$ and for this i generate the design point at a random point anywhere in the domain (uniform distribution).

Design Example: Evolving Instinct-Learning Balance

- Return to RSM example
- Here, study changes over time in environment, changing from $\sigma_z^2 = 0.5$ to $\sigma_z^2 = 0.75$ after 100 generations.
- What do you expect? Response surface (fitness function) shifts, tracks new optimum point $\rightarrow$ evolution

➙ Algorithm parameters: Evolve $\hat{x}_0$ and n so $p = 2$, but n must be an integer (just randomly increment/decrement by one or leave unchanged), and use $\beta = 0.01$ for cloud generation along instinct dimension, $p_m = 0.1$, and $S = 100$.

➙ Initial conditions: All unity. Poor instincts, no (low) memory/learning capability.

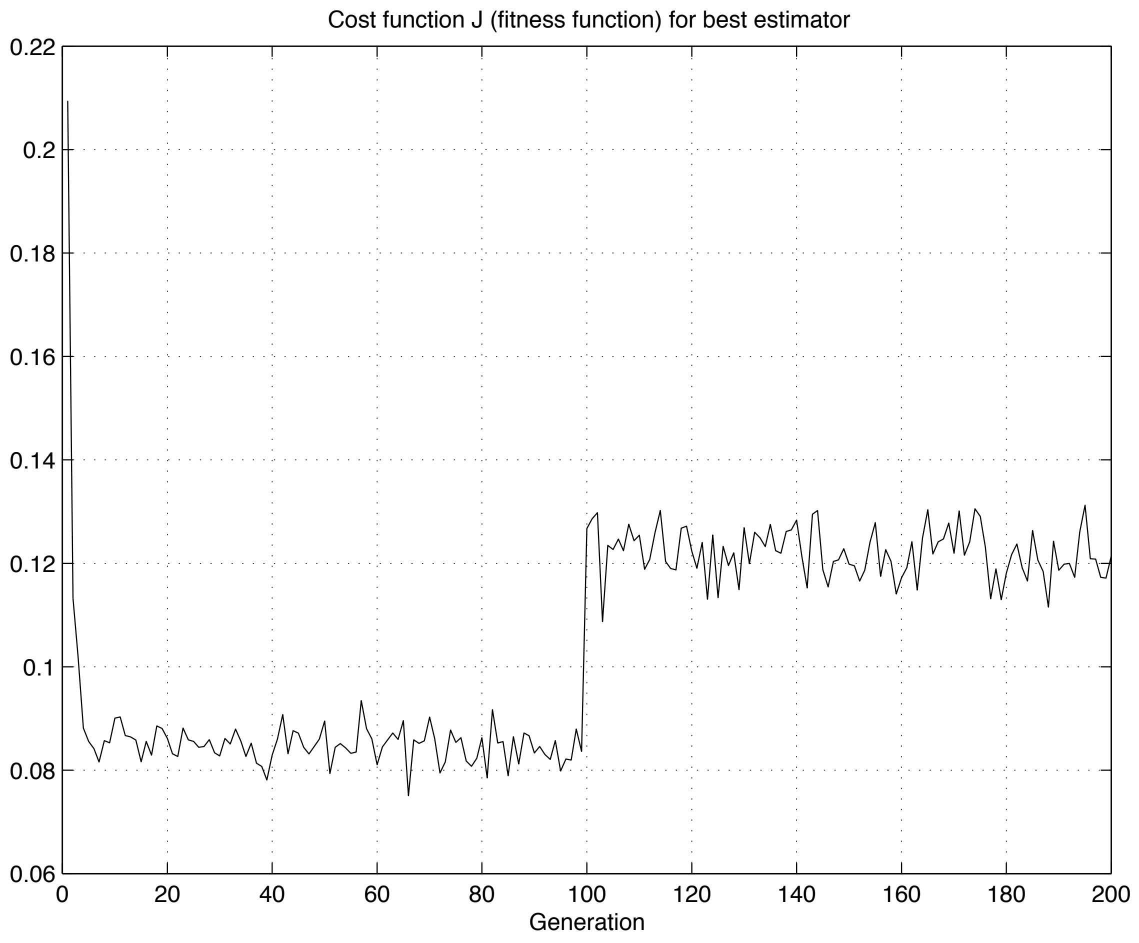

Figure 192: Cost J (fitness function) for set-based stochastic optimization for instinct-learning balance.

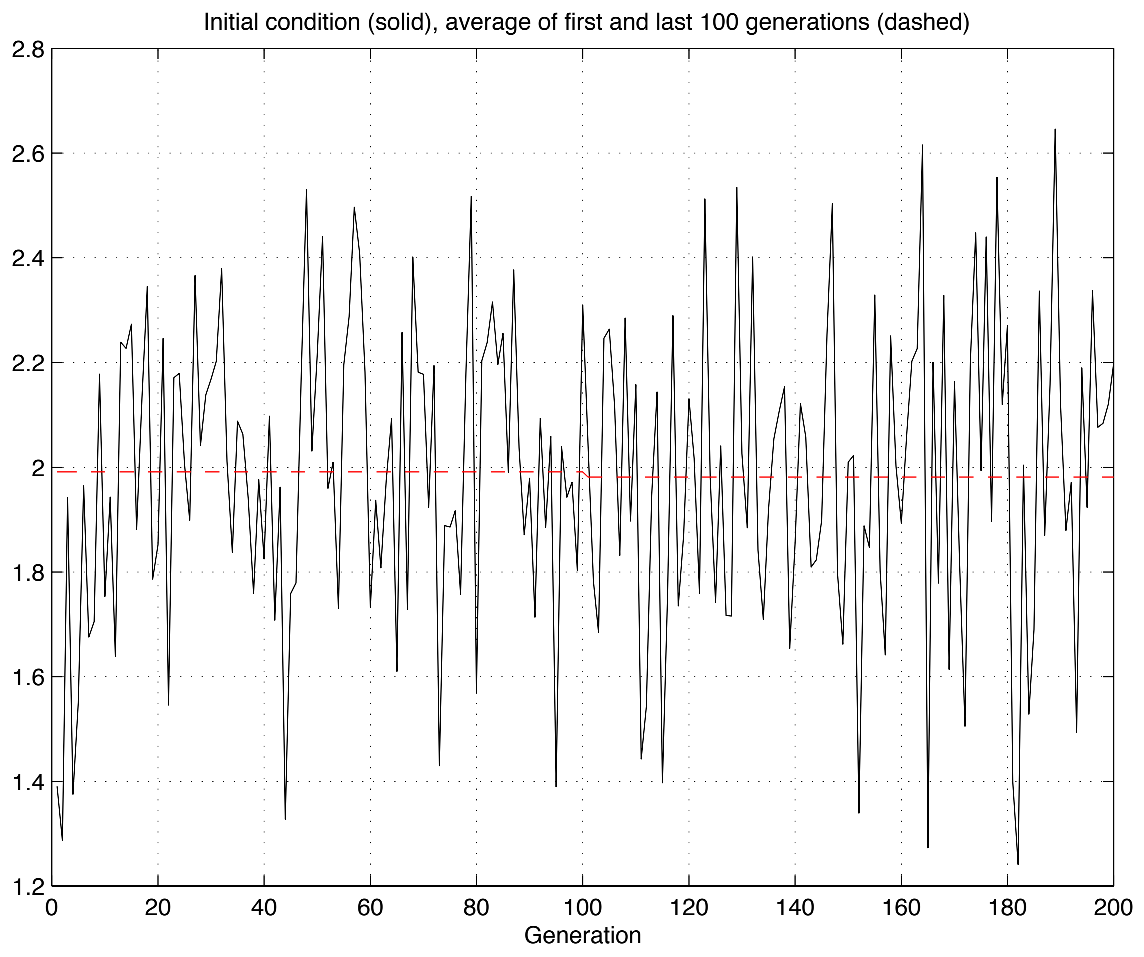

Figure 193: Initial condition $\hat{x}_0$ for set-based stochastic optimization for instinct-learning balance (averages=1.9911 and 1.9811 for first and last 100 generations).

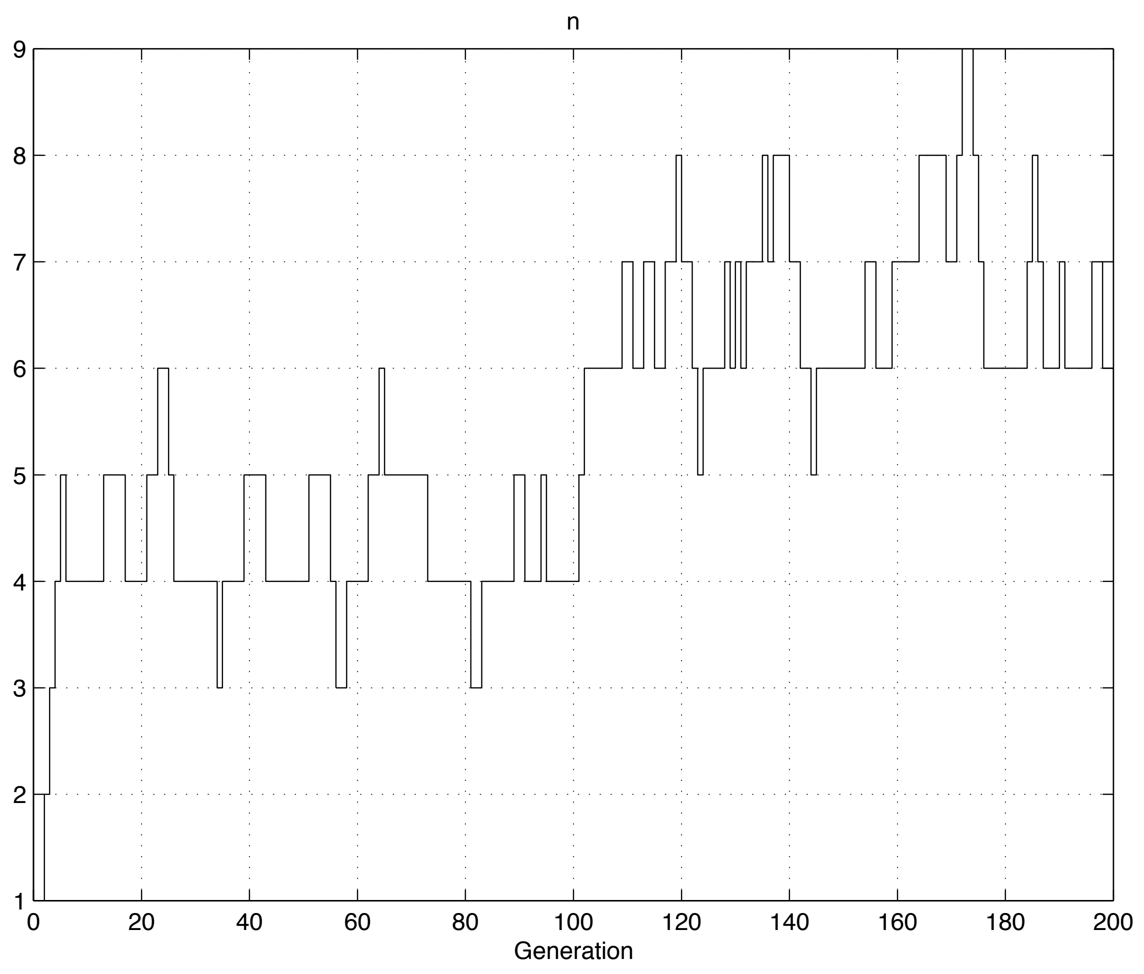

Figure 194: Learning capability n for set-based stochastic optimization for instinct-learning balance (increases).

★ Time-varying features of the environment shift the fitness landscape, and hence bring about changes in the design of an organism.

Evolutionary Control System Design

- CACSD following above ideas.
- Can we evolve hardware?
- Can we evolve software.
- Evolutionary strategies in the marketplace
- Darwinian design in the laboratory $\rightarrow$ robust controller design?

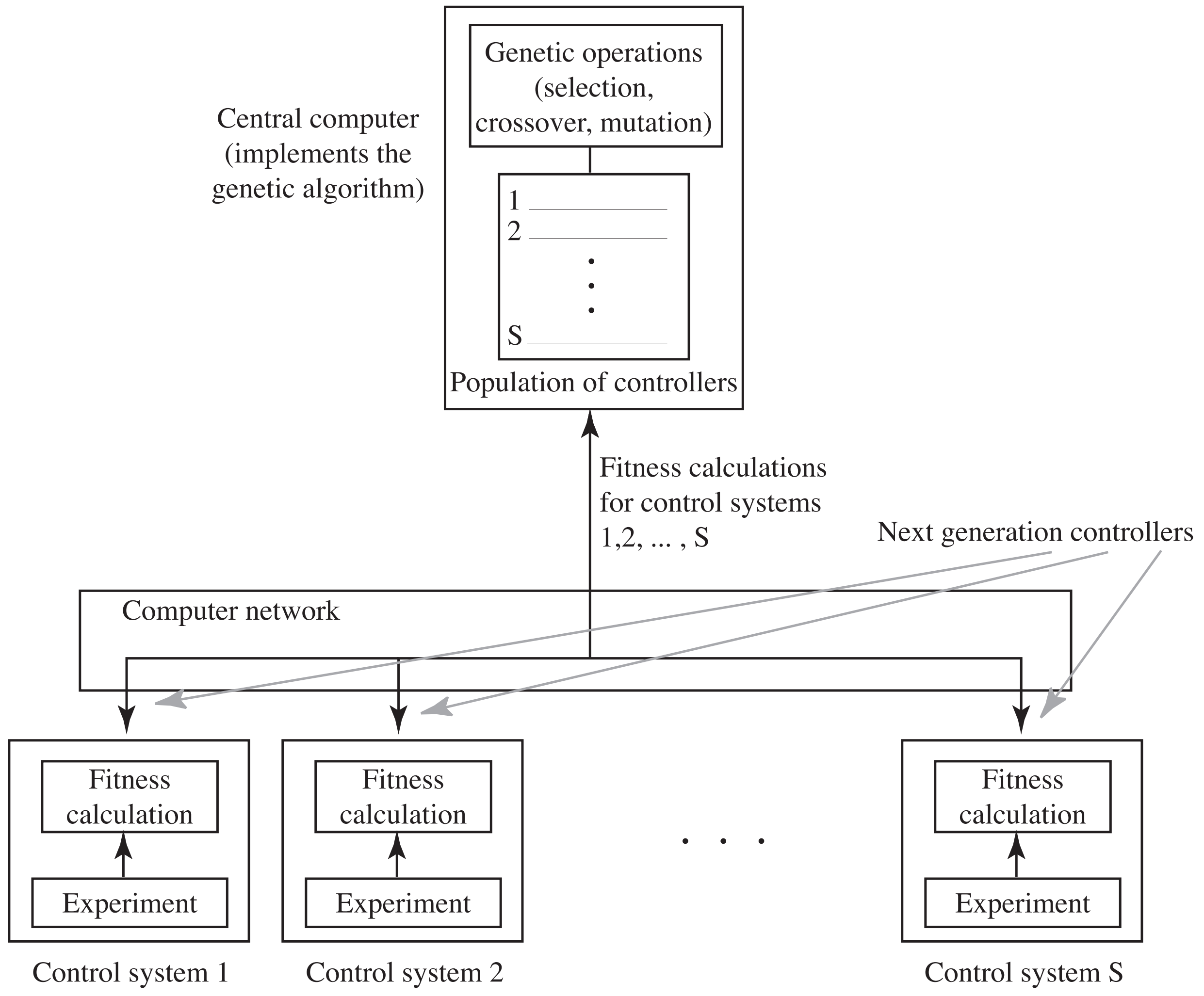
Genetic operations (selection, crossover, mutation)
Central computer (implements the genetic algorithm)
1
2
S
Population of controllers
Fitness calculations for control systems 1,2, ... , S
Next generation controllers
Computer network
Fitness calculation
Experiment
Control system 1
Control system 2
Control system S

Figure 195: Experiment to perform Darwinian design for physical control systems.

References

[1] Neil A. Campbell. *Biology.* Benjamin Cummings Pub. Co., Menlo Park, CA, 1996.

[2] D. Chrisiansen. Pigeon-guided missles that bombed out. *IEEE Spectrum*, page 46, August 1987.

[3] M. Domjan. *The Principles of Learning and Behavior.* Brooks/Cole Pub., NY, 4th edition, 1998.

[4] Michael S. Gazzaniga, Richard B. Ivry, and George R. Mangun. *Cognitive Neuroscience: The Biology of the Mind.* W. W. Norton and Co., NY, 1998.

[5] P. Gray. *Psychology.* Worth Pub., NY, 3 edition, 1999.

[6] M. Groening. *Simpsons Comics: Simpsorama.* HarperPerennial, Kansas City, 1995.

[7] E.R. Kandel, J.H. Schwartz, and T.M. Jessell, editors. *Essentials of neural science and behavior*. Appleton and Lange, Norwalk, CT, 1995.

[8] G. Larson. *The Far Side Gallery*. Andrews and McMeel, Kansas City, 1984.

[9] Irwin B. Levitan and Leonard K. Kaczmarek. *The Neuron: Cell and Molecular Biology*. Oxford University Press, Oxford, 2nd edition, 1997.

[10] M.T. Madigan, J.M. Martinko, and J. Parker. *Biology of Microorganisms*. Prentice Hall, NJ, 8 edition, 1997.

[11] K.M. Passino. *Biomimicry for Optimization, Control, and Automation*. ???, ???, ???

[12] P. Whitfield. *From So Simple a Beginning*. Macmillan Pub. Co. NY, 1993.