Dual Control of an Integrator
Using Neuro-Dynamic Programming

Anton Cervin

Department of Automatic Control
Lund Institute of Technology
April 1998
Title and subtitle
Dual Control of an Integrator Using Neuro-Dynamic Programming. (Dual reglering av en integrator med användning av neurodynamisk programmering).

Abstract
The dual control problem for an integrator with time-varying gain is reviewed. The neuro-dynamic programming technique of approximate policy iteration is explained. It is shown how a version of this technique, which employs a Gaussian radial basis function network, can be used to calculate a good sub-optimal dual controller for the integrator. This controller performs much better than the certainty equivalence and the cautious controllers. Finally, extensions to more realistic problems are discussed.
Contents

1 Introduction .................................................. 3
  1.1 Background ............................................... 3
  1.2 Outline of Thesis ......................................... 4

2 The Dual Control Problem ..................................... 4
  2.1 The Process ............................................... 4
  2.2 Non-Dual Controller 1: The Certainty Equivalence Controller ........................................... 5
  2.3 The Controller ............................................ 6
  2.4 Estimation ................................................ 8
  2.5 Dynamic Programming ..................................... 9
  2.6 Non-Dual Controller 2: The Cautious Controller .............................................................. 10
  2.7 Solving the Bellman Equation ............................. 11

3 Neuro-Dynamic Programming .................................... 11
  3.1 Neural Networks as Function Approximators .................. 11
  3.2 RBF Networks ............................................. 12
  3.3 Policy Iteration .......................................... 13
  3.4 Approximate Policy Iteration ............................... 13
    3.4.1 Policy Evaluation .................................... 13
    3.4.2 Policy Update ....................................... 14
  3.5 Further Matters .......................................... 14

4 Example 1: Dual Control of an Integrator with Constant Gain ........................................... 14
  4.1 The Problem ............................................... 15
  4.2 Neural Network Design ...................................... 15
  4.3 Initial Policy and Fall-Back Policy ........................ 16
  4.4 Policy Evaluation .......................................... 16
  4.5 Policy Update ............................................. 16
    4.5.1 Integral Evaluation .................................... 17
    4.5.2 Minimization ......................................... 17
  4.6 Results .................................................. 17
5 Example 2: Dual Control of an Integrator with Time-Varying Gain 18
  5.1 The Problem ................................................. 18
  5.2 Neural Network Design ................................. 18
  5.3 Initial Policy and Fall-Back Policy .................. 20
  5.4 Policy Evaluation .......................................... 20
  5.5 Policy Update .............................................. 21
    5.5.1 Integral Evaluation ................................. 21
    5.5.2 Minimization ......................................... 21
  5.6 Results .................................................. 22

6 Summary and Conclusions .......................... 23
  6.1 Control Strategies ..................................... 23
  6.2 The Approximate Policy Iteration Algorithm .... 24
1 Introduction

1.1 Background

Since the early 1950's, different adaptive control schemes have been developed to cope with processes having unknown or time-varying parameters. Heuristically derived schemes, such as the self-tuning regulator (STR) and the model-reference adaptive system (MRAS), have been successfully employed in many control problems. These schemes are however not optimal, and may even fail in some instances.

![Diagram of self-tuning regulator](image)

Figure 1: The structure of the self-tuning regulator.

The structure of the ordinary STR is shown in figure 1. The parameters of the process are estimated, typically using Kalman filtering. The parameter estimates are then used, in place of true parameter values, when the feedback is calculated in the controller. This approach is said to be based on the certainty equivalence principle. Unfortunately, adaptive schemes based on this principle do not yield optimal controllers. In fact, there is an intricate interplay between identification and control that must be considered.

Feldbaum [9] was among the first to realize that an optimal adaptive controller must indeed be a dual controller. The controller must not only control the process, but also sometimes excite the process, so that better estimates can be obtained and thus improve performance in the long run. The action that introduces the additional excitation is referred to as probing.

The optimal dual controller can be derived using a technique known as dynamic programming. The result is a recursive functional equation, called the Bellman equation. This equation is very difficult to solve. There is no analytical solution, except in trivial cases. A numerical solution requires enormous amounts of computational power and memory, and is impossible to carry out for realistic problems.

Despite the great computational difficulties, the idea of a controller with dual properties remains interesting. Therefore, a number of sub-optimal dual controllers, that all have some element of probing, have been proposed. For a survey, see Wittenmark [12].
1.2 Outline of Thesis

This thesis explores the possibility of using neuro-dynamic programming to obtain good sub-optimal dual controllers.

In section 2, the dual control problem for an integrator with time-varying gain is reviewed. Examples of non-dual controllers, that do not perform very well, are also given. Section 3 introduces the notion of neuro-dynamic programming, and especially the approximate policy iteration algorithm. Then, in section 4 and 5, two examples are given where this algorithm is used to obtain good sub-optimal dual controllers for the integrator. Finally, there are some conclusive remarks on how the algorithm would have to be modified to work with more realistic problems.

2 The Dual Control Problem

In this section, the dual control problem is formulated and solved for an integrator with time-varying gain. Also, the performances of two non-dual controllers are examined.

The solution to the dual control problem for a general ARX-process was given by Åström and Wittenmark [2]. The solution to the dual control problem for the special case of an integrator is given in Helmersson [10]. Additional discussion can be found in Åström and Wittenmark [3].

2.1 The Process

The process that is to be controlled is a discrete-time integrator with time-varying gain, described by

\[ y(t + 1) = y(t) + b(t)u(t) + e(t), \]

where \( u \) is the process input, \( y \) is the measurable process output, and \( \{e(t)\} \) is a sequence of independent Gaussian random variables with mean 0 and variance \( R_2 \).

The time-varying gain \( b \) is given by

\[ b(t + 1) = \Phi b(t) + v(t), \]

where \( \Phi \) is a known constant, and \( \{v(t)\} \) is a sequence of independent Gaussian random variables with mean 0 and variance \( R_1 \). Initially, \( b(0) \) is known to be a Gaussian variable with mean \( m \) and variance \( R_0 \).

Clearly, this process is difficult to control. If only a poor estimate of the gain \( b \) is available, a control action might steer the process in the wrong direction. A few industrial processes, including different kinds of grinders, exhibit similar problems, where the gain in the process might change sign at some critical point in time.

In the examples and simulations given in this thesis, one of the following special cases of the integrating process is used:

- A time-varying integrator with \( R_2 = 1, R_1 = 0.09, \) and \( \Phi = \sqrt{1 - R_1} = 0.95. \)

- An integrator with constant but unknown gain described by \( R_2 = 1, R_1 = 0, \) \( \Phi = 1. \)
2.2 Non-Dual Controller 1: The Certainty Equivalence Controller

Let us digress and see what happens if the integrator with time-varying gain is controlled by an ordinary self-tuning regulator, i.e. a certainty equivalence controller.

The certainty equivalence controller ignores the fact that parameter estimates may differ from the true parameter values. If the parameter \( \hat{b} \) was known at all times, the optimal controller for the process (1) would be given by the minimum variance controller (see e.g. Åström and Wittenmark [4]),

\[
u(t) = -\frac{y(t)}{\hat{b}(t)}.
\]

Substituting the estimated value for the true value of \( b \), we get the certainty equivalence controller

\[
u(t) = -\frac{y(t)}{\hat{b}(t)}.
\]

![Graphs showing system behavior](image)

Figure 2: Simulation of the certainty equivalence control system.

A simulation of the resulting control system is shown in figure 2. Because of the large scales, a close-up of \( y, u, \) and \( \Sigma y^2 \) is shown in figure 3. Clearly, the certainty equivalence controller fails to control the process. The total accumulated loss is \( \Sigma y^2 = 1.8 \cdot 10^7 \).

Most of the loss is due to the incident which occurs just before time \( t = 170 \). There, the parameter estimate \( \hat{b} \) happens to be very close to zero for a few samples, causing
the control signal $u$ to be very large according to (4). This is fatal, since we might not even have the right sign of $b$. At time $t = 166$ for example, $\hat{b} = -0.02$ while $b = 0.38$. Compared to the minimum variance controller, the certainty equivalence controller thus produces an output that is 19 times too large and of the wrong sign!

As seen in figure 3, the control signal is very powerful most of the time, which causes the loss to grow quite rapidly even before the incident at time $t = 170$. Because of the rich excitation provided by the control signal however, the parameter estimate is quite accurate, and the variance of the estimate does not exceed $P = 0.4$.

An ad hoc solution to the problem of the potentially unbounded control signal would be to limit the control to, say, $|u| < 10$. A simulation of this modified control system, using the same noise sequences as before, is shown in figure 4. The variance of the output is much lower now, and the total accumulated loss is $\Sigma y^2 = 1300$. The control signal is still powerful when the estimate is close to zero, but the limitation prevents the control system from deteriorating completely. The output at time $t = 170$ will still add noticeably to the accumulated loss. The parameter estimate is still quite accurate, and the variance does not exceed $P = 0.4$.

The ad hoc solution above does not yield an optimal controller, so let’s return to the formulation of the dual control problem.

2.3 The Controller

Let the purpose of the controller be to keep the process output as close to zero as possible (i.e. we have the reference value $y_r = 0$). This can be translated into the minimization of the loss function

$$J_N(t) = E \left\{ \sum_{k=t+1}^{t+N} y(k)^2 \right\},$$

(5)
Figure 4: Simulation of the limited certainty equivalence control system.
i.e., at each time, we want to minimize the expected sum of the squared outputs over the next $N$ steps. Ideally, we would let $N$ approach infinity (or however long the process is to run).

The loss function could be modified in a number of ways (e.g., by introducing a reference signal or by adding an extra penalty for large control signals) without increasing the complexity of the problem.

The control at time $t$ is allowed to be a function of all measurements up to time $t$, which are denoted

$$\mathcal{Y}_t = [y(0) \ y(1) \ \ldots \ \ y(t)].$$

### 2.4 Estimation

The unknown parameter $b$ in the process can be estimated using standard Kalman filtering:

$$\begin{align*}
\hat{b}(t) &= \Phi \hat{b}(t-1) + K(t-1)(y(t) - y(t-1) - \hat{b}(t-1)u(t-1)), \\
P(t) &= (\Phi - K(t-1)u(t-1))P(t)\Phi + R_1, \\
K(t-1) &= \frac{\Phi u(t-1)P(t-1)}{R_2 + u(t-1)^2P(t-1)},
\end{align*}$$

with initial conditions

$$\begin{align*}
\hat{b}(0) &= m, \\
P(0) &= R_0.
\end{align*}$$

The Kalman filter has the interesting property that the conditional distribution of $\hat{b}(t)$ given $\mathcal{Y}_t$ is Gaussian with mean $\hat{b}(t)$ and variance $P(t)$. The complete state of the process and the estimator can thus be described by the three scalars

$$\xi(t) = [y(t) \ \hat{b}(t) \ P(t)].$$

The vector $\xi(t)$ is referred to as the hyper-state of the control system. State variables and parameters are treated as equals, which is an important difference from traditional control schemes, where there is a clear distinction between the (typically) rapidly changing state variables and the slowly changing parameters. For more realistic problems, the hyper-state will be of very high dimension.

Figure 5: The structure of the dual control system.
The structure of the dual control system is shown in figure 5. Note the striking similarity to the structure of the STR in figure 1. The dual control system is more complicated though, because the control signal will generally be a non-linear function of the high-dimensional hyper-space.

2.5 Dynamic Programming

Minimizing the loss function (5) over u is very difficult. The control decision \( u(t) \) affects the probability distribution of \( \xi(t+1) \), which in turn affects the control decision \( u(t+1) \), which in turn affects the probability distribution of \( \xi(t+2) \), and so on, up to time \( t+N \).

The way to attack the problem is to use the technique of dynamic programming, introduced by Bellman, see e.g. [5]. Let the function

\[
V_N(\xi(t)) = \min_{u(t)} \left\{ \sum_{k=t+1}^{t+N} y(k)^2 \left| \xi(t) \right\} \right.
\]

denote the minimum expected loss over the next \( N \) steps, given that the system is in state \( \xi(t) \). The dynamic programming algorithm now states that the optimal choice of control should minimize the sum of the expected loss at the next step and the minimum expected loss over the remaining steps:

\[
V_N(\xi(t)) = \min_{u(t)} \left\{ y(t+1)^2 + V_{N-1}(\xi(t+1)) \left| \xi(t) \right\} \right.
\]

When there are no remaining steps, the expected loss is zero:

\[
V_0(\xi(t)) = 0.
\]

The optimal control policy will be a non-linear function from the hyper-state space to the control space, and is given by

\[
\mu(\xi(t)) = \arg \min_{u(t)} V_N(\xi(t)).
\]

The recursive functional equation (6) is called the Bellman equation of the control problem. It cannot be solved analytically, because of the complicated nesting of the expected values and the minimization. The expected values can be computed though, using the fact that the predicted values of \( y(t+1) \) and \( \hat{b}(t+1) \) are Gaussian, and that the predicted value of \( P(t+1) \) is a point distribution, see [1]. This yields

\[
V_N(y(t), \hat{b}(t), P(t)) = \min_{u(t)} \left\{ (y(t) + \hat{b}(t)u(t))^2 + R_2 + u(t)^2 P(t) \right. \right.
\]

\[
+ \int_{-\infty}^{\infty} \frac{e^{-y^2/2}}{\sqrt{2\pi}} V_{N-1}(y_p, \hat{b}_p, P_p) dy_p
\]

where

\[
y_p = y(t) + \hat{b}(t)u(t) + \sqrt{R_2 + u(t)^2 P(t)} \epsilon,
\]

\[
\hat{b}_p = \Phi \hat{b}(t) + \frac{\Phi u(t) P(t)}{\sqrt{R_2 + u(t)^2 P(t)}} \epsilon,
\]

\[
P_p = \frac{\Phi^2 R_2 P(t)}{R_2 + u(t)^2 P(t)} + R_1.
\]
2.6 Non-Dual Controller 2: The Cautious Controller

Facing the difficulties of solving the Bellman equation, let us digress one more time and look at another non-dual controller, namely the cautious controller. This controller is the result of solving the Bellman equation with optimization over only one step, i.e. by setting $N = 1$. The controller will always prefer control to probing, since a probing action will never start to pay off until at least two steps ahead.

The Bellman equation in this case reduces to

$$V_1(\xi(t)) = \min_{u(t)} \left\{ (y(t) + \hat{b}(t)u(t))^3 + R_2 + u(t)^2 P(t) \right\}. \quad (10)$$

Here, an analytical solution is possible, and the minimum is obtained for

$$u(t) = -\frac{\hat{b}(t)y(t)}{\hat{b}(t)^2 + P(t)}. \quad (11)$$

A simulation of the time-varying integrator being controlled by the cautious controller, using the same noise sequences as before, is shown in figure 6. The simulation shows

![Graphs of u, y, P, and Σy²](image)

Figure 6: Simulation of the cautious control system.

that this controller is indeed quite cautious. The control signal is much smaller, compared to the limited certainty equivalence controller in figure 4. The variance of the output is often lower, but the total accumulated loss is $\Sigma y^2 = 1250$, which is just about the same as for the limited certainty equivalence controller.
At the times $t = 25$, $t = 50$, and $t = 170$, the phenomenon of turn-off can be clearly observed. There, the parameter uncertainty is large ($P > 0.6$), and the cautious controller limits its output according to (11). This provides less excitation and causes the uncertainty to grow even larger. The estimate itself approaches zero, and the output starts to drift away. This adds noticeable to the loss at around time $t = 60$.

In order to achieve good control, optimization must be performed over many steps, so let us return to the problem of solving the Bellman equation in the general case.

### 2.7 Solving the Bellman Equation

Note that the Bellman equation can be solved off-line. The solution will yield a dual control policy, which can then be used to control the process on-line.

The most straightforward way to solve the Bellman equation is to carry out the entire recursion, as implied by (9). Starting with the known function $V_0 = 0$, the successive functions $V_1, V_2, \ldots, V_N$ are computed by repeated integration and minimization. This procedure is sometimes referred to as value iteration.

The main difficulty of a numerical solution is the representation and storage of the non-linear functions $V$ between iterations. A loss-function will typically have to be evaluated in a large number of discrete points in the hyper-space and then stored in a lookup-table. To find the minimum at each point, the integral must be numerically evaluated for a large number of possible choices of controls.

For the simple problem at hand, a straightforward numerical solution would be possible. However, as the dimension of the hyper-space grows, the computational demands grow exponentially. A more realistic problem would quickly exhaust the available computing resources on Earth. Therefore, good approximate solutions must be sought.

### 3 Neuro-Dynamic Programming

The term neuro-dynamic programming was coined by Bertsekas and Tsitsiklis [6] to describe a set of methods where neural networks are used to obtain approximate solutions to dynamic programming problems. This section will concentrate on radial basis function networks and the approximate policy iteration algorithm.

#### 3.1 Neural Networks as Function Approximators

Neural networks are generally used as non-linear function approximators. Given a representative set of function inputs and function outputs, a neural network can be trained to mimic a function by adjusting a number of network parameters. The parameters then constitute a compact description of the function.

There are two obvious possible uses of neural networks in the context of the dual control problem:

- As approximators of loss functions $V(\xi)$
- As approximators of control policies $\mu(\xi)$
Both of these functions, for reasonable large problems, would require enormous amounts of memory if stored in a lookup-table.

3.2 RBF Networks

A certain type of neural networks employed in this thesis are the radial basis function (RBF) networks, see e.g. [7]. They have been successfully used for identification in control systems, and a number sophisticated training algorithms have been developed, see e.g. [8].

In the Gaussian RBF network, a number of Gaussian kernels are placed with centers at different points in the input space. The output of the network consists of a weighed sum of the kernels. Given a point $\xi$ in the input space, the network produces an output $y$ according to

$$y = \sum_{i=1}^{n} w_i e^{-s^2 ||c_i-\xi||^2},$$

(12)

where $c_i$ and $w_i$ denote the center and the weight of the $i$:th kernel. The constant $s$ determines the spread of the kernels. An example of a simple Gaussian RBF network is shown in figure 7.

![Figure 7: A simple Gaussian RBF network with two-dimensional input space and four Gaussian kernels.](image)

The centers of the kernels are usually placed on a uniform grid, or at a subset of the input points. A more sophisticated training algorithm would allow for a more flexible placement of the centers, and perhaps even the possibility of using a different spread for each kernel.

A desirable property of the RBF networks is that the output is a linear combination of the kernels. Given a set of training data and a set of kernels, the weights can be found by solving a linear system of equations.
3.3 Policy Iteration

The original policy iteration algorithm provides an alternative method of solving the Bellman equation. The algorithm considers successive approximations of the optimal control policy (as opposed to the value iteration algorithm, where successive loss functions are considered.) Good descriptions are found in [6] and [11].

The algorithm is initialized by choosing an initial control policy \( \mu_0(\xi) \). Then, for each iteration \( i = 1, 2, \ldots \), the following steps are performed:

- The expected loss over \( N - 1 \) steps, given the control policy \( \mu_{i-1}(\xi) \), is calculated:

\[
V_{N-1}^{\mu_{i-1}}(\xi(t)) = E \left\{ \sum_{k=t+1}^{t+N-1} y(k)^2 \right\} \left\{ \xi(t), \mu_{i-1} \right\}.
\]

This is called the policy evaluation step.

- A new policy \( \mu_i(\xi) \) is given by calculating

\[
\mu_i(\xi(t)) = \arg \min_{\mu(t)} E \left\{ y(t + 1)^2 + V_{N-1}^{\mu_{i-1}}(\xi(t + 1)) \right\} \left\{ \xi(t) \right\}
\]

This is called the policy update step.

The basic idea of the algorithm is that the expected loss, given a control policy, should be easy to calculate. The algorithm is guaranteed to converge to the optimal control policy, and it will typically require very few iterations to do so.

The policy evaluation step for the dual control problem at hand is unfortunately very hard to carry out, but an approximate version of the algorithm will still be useful for our purposes.

3.4 Approximate Policy Iteration

Several variants of an approximate policy iteration algorithm are described in [6]. The simple variant described here involves using Monte Carlo simulation to evaluate (13), and using a neural network to store the loss functions between the steps in the algorithm.

3.4.1 Policy Evaluation

A very simple way to evaluate the expected loss by Monte Carlo simulations is:

1. Choose a number of representative starting points in the hyper-state space.
2. From each starting point, simulate a number of trajectories of length \( N - 1 \), using the control policy \( \mu_{i-1}(\xi) \), and then calculate the average loss over a trajectory.

It is worth noticing the constant term \( R_2 \) in the Bellman equation (15), which is due to the unpredictable innovation \( \epsilon(t + 1) \). This term can be removed without affecting the equality. Similarly, the actual squared innovations can be subtracted from the
total loss of a simulated trajectory. The remaining part of the loss, which is referred to as the learning loss, has lower variance. Considering the learning loss instead of the actual loss results in fewer trajectories having to be simulated for each starting point.

After the simulation, a number of noisy samples of the function \( V^{π_{t-1}}(ξ) \) are available, and a Gaussian RBF network can be trained to approximate the function. If the number of Gaussian kernels as well as the positions of their centers are fixed, a simple least-squares fit can be used to find the weights.

### 3.4.2 Policy Update

To complete one iteration of the approximate policy iteration algorithm, (14) must be evaluated, but now using the neural network approximation \( \tilde{V}^{π_{t-1}}(ξ) \) instead. This step may be just as time-consuming as before. But as we will see in the examples later, the choice of a Gaussian RBF network can reduce the computation time.

In the method described, it is assumed that the control policy function is evaluated in a number of representative points, and then stored in a lookup-table. In more realistic problems, a neural network would have to be used to store this function as well.

### 3.5 Further Matters

There are a number of matters that must be considered before the algorithm can be implemented to actually solve a dual control problem:

- The hyper-state space must be sampled in a representative way. If this space is infinite, it could be transformed into a finite one. Alternatively, the dual control policy could be calculated in only the most important region of the hyper-state space, and a non-dual fall-back policy be used outside this region.

- The neural network must be carefully designed, so that a good enough approximation of the loss function is obtained. In the case of a Gaussian RBF, this involves selecting an appropriate number of basis functions, and selecting their centers and their spread. As much a priori knowledge of the loss function as possible must be used in designing the network.

- A good initial policy \( μ_0(ξ) \) should be chosen. The better this policy, the faster the algorithm will converge.

### 4 Example 1: Dual Control of an Integrator with Constant Gain

In this first example it is shown how the approximate policy iteration algorithm can be used to obtain a good dual controller for the integrator with constant but unknown gain. This example was chosen because that the optimal dual controller for this process has been numerically computed (using value iteration) in Helmersson [10] and Åström and Helmersson [1].
4.1 The Problem

The integrator with constant but unknown gain is a special case of the process (1) and (2) with \( \Phi = 1 \) and \( R_1 = 0 \), and can thus be written

\[
y(t + 1) = y(t) + bu(t) + e(t).
\]

The time horizon in the optimization was chosen as \( N = 30 \), to correspond with the controller computed in [10].

The process is difficult to control during startup. After a transient, though, the gain \( b \) will be known, and the process might as well be controlled by a certainty equivalence controller.

It is shown in [1] that the complexity of the dual control problem for this process can be reduced by introducing the normalized hyper-state variables

\[
\eta(t) = y(t) / \sqrt{R_2}, \\
\beta(t) = \tilde{b}(t) / \sqrt{P(t)},
\]

and the normalized control signal

\[
\nu(t) = u(t) \sqrt{P(t)} / \sqrt{R_2}.
\]

The Bellman equation (9) then reduces to

\[
V_N(\eta(t), \beta(t)) = \min_{\nu(t)} \left\{ (\eta(t) + \beta(t) \nu(t))^2 + R_2 + \nu(t)^2 \right. \\
+ \left. \int_{-\infty}^{\infty} \frac{e^{-x^2/2}}{\sqrt{2\pi}} V_{N-1}(\eta_p, \beta_p) \, dx \right\},
\]

where

\[
\eta_p = \eta(t) + \beta(t) \nu(t) + \sqrt{1 + \nu(t)^2} \varepsilon, \\
\beta_p = \beta(t) \sqrt{1 + \nu(t)^2} + \nu(t) \varepsilon.
\]

The variable \( \beta \) can be interpreted as the relative certainty of the estimate \( \tilde{b} \). Typically, \( \beta \) will be close to zero during startup, but after a transient, \( \beta \) will approach infinity. The normalized output variable \( \eta \) will stay close to zero if a reasonable good control policy is used.

4.2 Neural Network Design

A Gaussian RBF Network was used to approximate the expected-loss function \( V_{N-1}(\eta, \beta) \). Since this function is symmetric with respect to both \( \eta \) and \( \beta \), only the first quadrant of the hyper-space was considered. The output of the network can be written

\[
\tilde{V}_{N-1}(\eta, \beta) = \sum_{i=1}^{n} w_i e^{-x^2((c_{ni} - \eta)^2 + (c_{bi} - \beta)^2)},
\]

where \( w_i \) and \( (c_{ni}, c_{bi}) \) are the weight and the center of the \( i \)-th Gaussian kernel.

49 Gaussian kernels were placed on a uniform grid covering the region \([0, 3] \times [0, 3]\) (see figure 8), since this is the most important part of the hyper-space. Outside this region, a fall-back policy must be used. The spread of the kernels was chosen to \( s = 3 \) to obtain a smooth approximation.
4.3 Initial Policy and Fall-Back Policy

As an initial control policy $\mu_0$, the cautious control policy (compare eq. (11)) was used:

$$
\mu_0(\eta(t), \beta(t)) = -\frac{\beta(t)\eta(t)}{\beta(t)^2 + 1}
$$

(17)

This policy was also used as a fall-back policy, whenever the fall-back variables fell outside the region $[-3,3] \times [-3,3]$. The limited certainty equivalence control policy might as well have been used. It is interesting to note that the cautious control policy is very close to the certainty equivalence control policy when $\beta$ is large.

4.4 Policy Evaluation

Samples of the expected-loss function $V_{N-1}(\eta, \beta)$ were obtained through Monte Carlo simulations:

1. 1000 starting points in the region $[0,3] \times [0,3]$ were chosen randomly.

2. From each starting point, 100 trajectories of length 29 were simulated, using the current control policy (or, if necessary, the fall-back policy), and the average learning loss over a trajectory was calculated.

Since the number of basis functions and the placement of their centers were fixed, the weights of the neural network could be obtained from a simple least-squares fit.

4.5 Policy Update

The new policy was calculated in $16 \times 16$ points and stored in a lookup-table. To find the minimum in (15), the integral must be evaluated for several choices of $\nu$. 
4.5.1 Integral Evaluation

The integral in equation (15) is evaluated using the approximate loss function $\tilde{V}_{N-1}$:

$$\tilde{I} = \int_{-\infty}^{\infty} \frac{e^{-x^2/2}}{\sqrt{2\pi}} \tilde{V}_{N-1}(\eta_p, \beta_p) \, d\epsilon.$$  \hfill (18)

Inserting the expression for $\tilde{V}_{N-1}(\eta, \beta)$ from (16) and interchanging the order of summation and integration, we get

$$\tilde{I} = \sum_{i=1}^{n} w_i \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2-s^2((c_{ni}-\eta_i)+((c_{\beta i}-\beta_i)^2))} \, d\epsilon.$$  \hfill (19)

The integrals can now be solved analytically, by means of completing the squares, according to

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-ax^2-2bx-c} \, dx = \frac{1}{\sqrt{2a}} e^{-c+b^2/a}, \quad a > 0.$$  \hfill (20)

Identify:

$$a = \frac{1}{2} + s^2(1 + 2\nu^2),$$
$$b = s^2((c_{ni} - \eta - \beta\nu)\sqrt{1 + \nu^2} + (c_{\beta i} - \beta\sqrt{1 + \nu^2}))\nu),$$
$$c = s^2((c_{ni} - \eta - \beta\nu)^2 + (c_{\beta i} - \beta\sqrt{1 + \nu^2})^2).$$

The integral in the original policy iteration algorithm has thus been replaced by a sum in the approximate policy iteration algorithm.

The formulas above had to be modified before use, since the approximation $\tilde{V}_{N-1}$ is only valid in the region $[0,3] \times [0,3]$. A simple solution was to modify $\eta_p$ and $\beta_p$ using

$$\eta_p \leftarrow \min\{\|\eta_p\|, 3\},$$
$$\beta_p \leftarrow \min\{\|\beta_p\|, 3\}.$$  \hfill (22)

Comparisons showed that this approach produced results very similar to those obtained when using a bigger neural network with 121 basis functions covering the region $[-5,5] \times [-5,5]$.

4.5.2 Minimization

The minimization was implemented as two simple searches. First values of $\nu$ from $-4$ to 0 in steps of 0.2 were tested. A finer search with step-size 0.01 was then performed around the minimum of the first search.

4.6 Results

The approximate policy iteration algorithm was implemented as two Matlab programs: one for the policy evaluation step and one for the policy update step. Execution times, running on a Sun Ultra 1 workstation, were about 25 minutes for a policy evaluation, and about 2 minutes for a policy update.
Figure 9 shows the results from the first three iterations. Clearly, the loss becomes smaller with each iteration, especially for small values of $\beta$. The algorithm converged very quickly, and there was no noticeable difference in successive iterations.

The loss function and the control policy after three iterations (the third row in figure 9) were compared to the numerically computed optimal dual controller from Helmersson [10], which is shown in figure 10. The loss function is only slightly lower for the optimal dual controller, which shows that the approximate policy iteration algorithm has produced a good sub-optimal dual controller.

The control policies are also very similar. Both policies will introduce probing when the output is small and the relative uncertainty is large, i.e. when both $\eta$ and $\beta$ are close to zero. For $\eta = 0$, any control action could be described as a pure probing action, whose sole purpose is to improve the quality of future estimates. The figures show that the suboptimal dual controller has a smaller probing zone than the optimal dual controller. The control signal is smaller also when $\beta = 0$. (This is difficult to see in figure 9.) The diminished probing action may be due to the Monte Carlo simulations and the fall-back policy. Typically, $\beta$ will fall outside $[-3, 3]$ in about ten samples time. Therefore, our version of the approximate policy iteration algorithm will not really perform a 30-step optimization, but perhaps rather a 10-step optimization. Optimization over fewer steps results in a smaller probing zone, as seen in [1].

Simulations which compare the transient behavior of the dual controller to several other controllers can be found in [10].

5 Example 2: Dual Control of an Integrator with Time-Varying Gain

In this section, it is shown how the approximate policy iteration algorithm can be applied to the dual control problem of the integrator with time-varying gain. This example is very similar to the previous example, except that the loss functions and the control policies have to be computed in three dimensions instead of just two. The curse of dimensionality is becoming evident as longer simulations, a larger RBF network, and a larger lookup-table are needed.

5.1 The Problem

The dual control problem for this process is described in section 2. The parameters of the process were chosen as $R_2 = 1$, $R_1 = 0.09$, and $\Phi = \sqrt{1 - R_1} = 0.95$, i.e. the same values that were used in the simulations of the certainty equivalence control system (section 2.2, figures 2, 3, and 4) and the cautious control system (section 2.6, figure 6).

A short time horizon of $N = 6$ was chosen, because of the extremely large variance in the Monte Carlo simulations.

5.2 Neural Network Design

A Gaussian RBF Network was used to approximate the loss function $V_{N-1}(y, \hat{b}, P)$. This function is symmetric with respect to both $y$ and $\hat{b}$, and $P$ can assume only
Figure 9: The first three iterations of the approximate policy iteration algorithm. a), c), and e) show the simulated learning loss after iteration 1, 2, and 3. b), d), and f) show the control policies after iteration 1, 2, and 3.
positive values, so only the first octant of the hyper-space was considered. The output of the network can be written

\[ \hat{V}_{N-1}(y, \hat{b}, P) = \sum_{i=1}^{n} w_i e^{-s_y^2(c_{y_i} - y)^2 - s_b^2(c_{b_i} - \hat{b})^2 - s_P^2(c_{P_i} - P)^2}, \]  

(23)

where \( w_i \) and \((c_{y_i}, c_{b_i}, c_{P_i})\) are the weight and the center of the \( i \)-th basis function, and \( s_y, s_b, \) and \( s_P \) are spread constants along the different dimensions.

From simulations such as those in section 2.2 and 2.6 it was decided that the most important part of the hyper-space was \([-5, 5] \times [-2, 2] \times [0.09, 1]\). The spread constants were chosen as \( s_y = 1.67, s_b = 8.33, \) and \( s_P = 16.65 \).

Because of badly conditioned matrix calculations, the network could no longer be designed using fixed centers and a simple least-squares fit. Instead, Matlab's \texttt{solveverb} function was used to calculate centers and weights of the Gaussian kernels. Unfortunately, a very large number of kernels had to be used to obtain a good approximation. In the first iteration, a total of 1000 kernels was used.

### 5.3 Initial Policy and Fall-Back Policy

As an initial control policy \( \mu_0 \), the cautious control policy from section 2.6 was used. As a fall-back policy, the limited certainty equivalence controller from section 2.2 was used.

### 5.4 Policy Evaluation

Samples of the expected-loss function \( V_{N-1}(y, \hat{b}, P) \) were obtained through Monte Carlo simulations:

1. 1210 starting points were distributed evenly in the region \([0,10] \times [0,2] \times [0.1,1]\).
2. From each starting point, 10000 trajectories of length 5 were simulated, using the current control policy (or, if necessary, the fall-back policy), and the average learning loss over a trajectory was calculated.

A much larger number of trajectories had to be simulated for each point in this example. This is due to the time-varying parameter $b$, which increases the variance of a trajectory dramatically.

Matlab’s `solve` function was then used to calculate centers and weights of a total of 1000 kernels.

5.5 Policy Update

The new policy was calculated in $11 \times 11 \times 10$ points and stored in a lookup-table. As before, integral evaluation and minimization had to be performed.

5.5.1 Integral Evaluation

The integral in the Bellman equation (9) was evaluated using the approximate loss function $\tilde{V}_{N-1}$:

$$\tilde{I} = \int_{-\infty}^{\infty} \frac{e^{-x^2/2}}{\sqrt{2\pi}} \tilde{V}_{N-1}(y, \hat{b}, P) \, dx$$

(24)

Inserting the expression for $\tilde{V}_{N-1}(y, \hat{b}, P)$ from (23) and interchanging the order of summation and integration, we get:

$$\tilde{V}_{N-1}(y, \hat{b}, P) = \sum_{i=1}^{n} w_i e^{-s^2_y(c_{xi} - y)^2 - s^2_b(c_{xi} - \hat{b})^2 - s^2_P(c_{pi} - P)^2}$$

(25)

Just as before, the integrals can be solved analytically according to (20). Identify:

$$a = 1/2 + s^2_y(R_2 + u^2 P) + \frac{s^2_b(\Phi u P)^2}{R_2 + u^2 P}$$

(26)

$$b = s^2_y((c_{xi} - y - \hat{b} u) \sqrt{R_2 + u^2 P} + \frac{s^2_b(c_{xi} - \Phi \hat{b} u P}{\sqrt{R_2 + u^2 P}}\right)$$

$$c = s^2_y(c_{xi} - y - \hat{b} u)^2 + s^2_b(c_{xi} - \Phi \hat{b} u P) + s^2_P \left( c_{pi} - \frac{\Phi^2 R_2 P}{R_2 + u^2 P} - R_1 \right)^2$$

The formulas above again had to be modified before use, since the approximation $\tilde{V}_{N-1}$ is only valid in the region $[0, 5] \times [0, 2] \times [0.09, 1]$, thus

$$y_p \leftarrow \min \{ |y_p|, 5 \},$$

(27)

$$\hat{b}_p \leftarrow \min \{ |\hat{b}_p|, 2 \}.$$

No examination of the effects of these modifications were performed in this example.

5.5.2 Minimization

The minimization was again implemented as two simple searches. This time however, limits and step-size of the first search varied. A finer search with step-size 0.1 was then performed around the minimum of the first search.
5.6 Results

In this example, the approximate policy algorithm was implemented as three Matlab programs: one each for the Monte Carlo simulations, the neural network approximation, and the policy update step. Execution times, running on a Sun Ultra 1 workstation, were about 50 hours for a Monte Carlo simulation, about 1.5 hours for a neural network approximation and about 1.5 hours for a policy update step.

Figure 11 shows the control policy that resulted from a single iteration of the algorithm. Since the control policy is now a function of three variables, projections must be used to visualize it. The control policy is very different for different values of $P$. In a), the estimate of $b$ is accurate, and the controller can react forcefully to deviations in the output. In d), the estimate is inaccurate, and the controller must be more cautious. The probing property of the controller is less visible in the graphs.

Unfortunately, the `solve` function was unable to calculate the weights and centers of the Gaussian kernels in the second iteration. Because of the fast convergence of the algorithm however, we can expect to have a quite good suboptimal dual controller after a single iteration.

Figure 11: Projections of the control policy obtained after one iteration. a) $P = 0.1$, b) $P = 0.2$, c) $P = 0.4$, d) $P = 0.8$. 
A simulation of the dual control system, using the same noise sequences as in previous simulations, is shown in figure 12. The simulation indicates that the controller performs much better than both the limited certainty equivalence controller (see figure 4) and the cautious controller (see figure 6). The total accumulated loss is only $\Sigma y^2 = 470$, which is to be compared to the previous values of $\Sigma y^2 = 1300$ (the limited certainty equivalence controller) and $\Sigma y^2 = 1250$ (the cautious controller).

6 Summary and Conclusions

The different control strategies for the integrator with time-varying gain are reviewed. Then the approximate policy iteration algorithm is evaluated, and extensions to more realistic problems are discussed.

6.1 Control Strategies

Three different control strategies have been employed to control the integrator with time-varying gain. The certainty equivalence controller (see figures 2 and 3) produces very large control signals whenever the parameter estimate is close to zero. Because of this, the variance
of the output and the accumulated loss will be very high. However, the parameter estimate will be quite accurate. Limiting the control signal reduces the loss, but the control signal will still be too powerful when the estimate is close to zero.

The cautious controller (see figure 6) experiences problems that are almost the opposite to the ones of the certainty equivalence controller. The control signal is very small when the parameter estimate is close to zero, but also when the variance of the estimate is large. This can lead to turn-off, where controller shuts down completely and the output starts to drift away. Because of the limited excitation provided by the controller, the estimate will often be poor. Longer simulations show that the cautious controller performs about as well as the limited certainty equivalence controller.

Finally, the dual controller (see figure 12) is more cautious than the certainty equivalence controller, but avoids the turn-off problem of the cautious controller by introducing probing when the estimate is poor. The estimate is fairly accurate at all times. Longer simulations show that the accumulated loss of the dual controller is almost half of the loss of the certainty equivalence and the cautious controllers. For certain processes then, dual control can really pay off.

6.2 The Approximate Policy Iteration Algorithm

The examples have shown that the approximate policy iteration algorithm can be used to obtain good sub-optimal dual controllers for simple processes. However, a very large number of design decisions must be made, especially concerning the neural network. There are no guarantees that the algorithm will produce a good dual controller, but at least it can be evaluated by simulations before use.

There is of course no use in employing the algorithm for simple processes where a numerical calculation of the optimal dual controller is possible. Instead, it should be used to compute good dual controller for more complex problems where other controllers, including simpler sub-optimal dual controllers, have failed. Using Monte Carlo simulations to find the expected loss function is a very general idea, that could be used with any system that can be simulated. However, the simulations will require enormous amounts of computing power.

The problems experienced with the time-varying integrator in section 5 indicate that the simple version of the approximate policy iteration algorithm described in this thesis will not work with more realistic problems. Several modifications would have to be made, including the following:

- The curse of dimensionality prevents the use of lookup-tables, so a second neural network would have to be used to approximate the control policy \( \mu_i(\xi) \).

- More sophisticated simulation methods would have to be investigated, see [6]. Each transition of each simulated trajectory contains information about the loss function, and this should ideally be exploited in the calculations.

- More sophisticated neural network training algorithms would have to be employed. RBF networks are claimed to work well as approximators in higher dimensions, but that requires at least flexible placement of the RBF centers, see e.g. [8].

24
References


