EMPIRICAL RESULTS ON CONVERGENCE AND EXPLORATION IN APPROXIMATE POLICY ITERATION

Niket S. Kaisare [*] **Jong Min Lee** [*] **Jay H. Lee** [*,1]

[*] *School of Chemical and Biomolecular Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA*

Abstract: In this paper, we empirically investigate the convergence properties of policy iteration applied to the optimal control of systems with continuous state and action spaces. We demonstrate that policy iteration requires lesser iterations than value iteration to converge, but requires more function evaluations to generate cost-to-go approximations in the policy evaluation step. Two different alternatives to policy evaluation, based on iteration over simulated states and simulation of improved policies are presented. We then demonstrate that the λ-policy iteration method, with $\lambda \in [0,1]$, is a tradeoff between value and policy iteration. Finally, the issue of exploration to expand the coverage of the state space during offline iteration is also considered. *Copyright© 2005 IFAC*

1. INTRODUCTION

Dynamic Programming (DP), introduced by Bellman (1957), generated a lot of interest as it provides a theoretically sound framework for solving multi-stage dynamic optimization problems. Such stagewise optimization is encountered in optimal control problems, Markov Decision Processes, planning and scheduling, etc. DP aims to characterize the optimal solution to the dynamic optimization in the form of "cost-to-go" or "value" function, which expresses the desirability of any state x in the state space with respect to the long-term performance that can be achieved.

While DP is an elegant method, its suffers from the *curse of dimensionality*, a term used to indicate that the method is computationally intractable for even moderate-sized real problems. Researchers working in artificial intelligence and operations research areas developed various methods to alleviate the curse of dimensionality. These are collectively referred to as Reinforcement Learning (Sutton and Bartow, 1998) and Neuro-Dynamic Programming (Bertsekas and Tsitsiklis, 1996). According to Sutton and Bartow (1998), the main idea of these methods is that they aim to obtain good *suboptimal* solutions to the DP problem, starting with suboptimal policies and suboptimal cost-to-go function, iteratively using each to improve the other.

Of particular interest in this paper is simulation-based approximate policy iteration. Policy iteration, often attributed to Howard (1960), generates an improving sequence of policies from the corresponding cost-to-go functions, followed by updating the cost-to-go function by evaluating the generated policy. An alternative to the policy iteration is the value iteration, also referred to as the method of successive approximation, which amounts to iterative substitution of the value function based on Bellman's optimality equation. We previously reported some successes in applying approximate value iteration algorithm to several interesting process control problems (Kaisare *et al.*, 2003; Lee, 2004; Lee *et al.*, 2004). Our aim is to compare these two approaches. Recently,

[1] Corresponding author: jay.lee@chbe.gatech.edu

Santos and Rust (2004) proved that under certain regularity conditions, policy iteration shows superlinear or quadratic rate of convergence. In contrast, value iteration has only a linear rate of convergence. De Farias and Van Roy (2000) argued that approximate value iteration need not converge under some cases. Based on their studies of temporal difference learning (which, like policy iteration, iterates in the policy space), they suggested a variation of approximate value iteration that is guaranteed to converge.

In this paper, we aim to compare the convergence properties of policy iteration. We show that policy iteration converges faster than value iteration; but we demonstrate a case in which policy evaluation fails to converge in a reasonable amount of time. Various alternatives that exist for *policy evaluation* are considered. These are:

- Computing cost-to-go iteratively over the data points determined by the initial suboptimal control simulations (iter-PI)
- Through simulation of the improved policies generated during the policy iteration. Cost-to-go values are directly calculated as the sum of individual costs until steady state (sim-PI)
- Using temporal difference based λ-policy iteration (λ-PI of Bertsekas and Ioffe (1996))

Finally, we allude to the issue of state space coverage. As the cost-to-go approximators are built with data occupying a limited subset of the state space, we need to avoid extreme extrapolations during online control. Additionally, policy iteration can be extended to provide exploration of the unvisited regions of the state space during offline learning phase, through input dithering.

2. BACKGROUND

The main aim of Dynamic Programming (DP) is to obtain an optimal *value* or *cost-to-go* function, which is shown to satisfy the following Bellman equation (Bellman, 1957):

$$J^*(x) = \min_u \{\phi(x,u) + J^*(f_h(x,u))\} \tag{1}$$

where $x \in \mathcal{X}$ is the system state, $u \in \mathcal{U}$ is the control action, $f_h(x,u)$ represents the state transition according to a discrete-time model

$$x_{k+1} = f_h(x_k, u_k), \tag{2}$$

$\phi(x,u)$ is the single stage cost, and $J(x)$ is the cost-to-go. The superscript $\cdot^*$ indicates the optimal conditions. We represent the Bellman equation above in the form of DP operator T as

$$J^* = TJ^* \tag{3}$$

Blackwell (1965) was amongst the first authors to rigorously prove that the DP operator T is a contraction mapping. This is an important result as it guarantees convergence to a unique solution J^* through the use of iterative algorithms.

Once obtained, the optimal cost-to-go function $J^*(x)$ can be used to compute control actions u_k at any state x_k by solving the following optimization problem

$$\begin{aligned} u_k &\triangleq \mu^*(x_k) \\ &= \arg\min_{u_k} \{\phi(x_k,u_k) + J^*(f_h(x_k,u_k))\} \end{aligned} \tag{4}$$

In the above equation, $\phi(x_k, u_k)$ represents the current stage cost incurred in implementing control action u_k at state x_k, while $J(\cdot)$ represents the projected sum of future stage costs, hence the name 'cost-to-go.'

2.1 Dynamic Programming algorithms

Policy iteration, which was first proposed by Howard (1960), is a useful algorithm that computes the optimal cost-to-go function $J^*(x)$ by alternating the sequence of *policy improvement* and *policy evaluation* steps. First, policy improvement is used to find an improving policy $\mu^{i+1}(x)$ using cost-to-go function $J^i(x)$ according to:

$$\mu^{i+1}(x) = \arg\min_u \left\{\phi(x,u) + J^i\left(f_h(x,u)\right)\right\} \tag{5}$$

The second step involves following the policy to compute an updated cost-to-go function, which is given by the following implicit equation

$$J^{i+1}(x) = \phi(x, \mu^{i+1}(x)) + J^{i+1}\left(f_h(x, \mu^{i+1}(x))\right) \tag{6}$$

In contrast, value iteration does not generate an improving policy. Instead, it uses the Bellman equation (1) as an update rule to sequentially improve the cost-to-go function starting from a suboptimal cost-to-go J^0. For every $x \in \mathcal{X}$, value iteration is written as:

$$J^{i+1}(x) = \min_u \left\{\phi(x,u) + J^i\left(f_h(x,u)\right)\right\} \tag{7}$$

Puterman and Brumelle (1979) showed the equivalence between policy iteration and Newton's method. They proved that policy iteration has super linear or quadratic rate of convergence, while value iteration has only a linear rate of convergence. Thus, policy iteration converges faster, and an exact solution (as opposed to an asymptotic solution) can be obtained.

2.2 Approximate Dynamic Programming (ADP)

The DP algorithms are elegant and have proven convergence properties. However, they suffer from the curse of dimensionality, which arises because the cost-to-go values need to be computed and stored for all possible points in the state space. As a result, the memory and computational requirements grow exponentially with an increase in state dimension, making conventional DP infeasible for most practically sized problems.

An alternative is to obtain approximate solution for cost-to-go by solving the DP in a smaller subset of the state space defined through simulations of suboptimal policies. In addition to this, a function approximator is used for interpolating the cost-to-go values within this simulation-defined subset. Thus, the curse of dimensionality is avoided through synergistic use of simulations and function approximation.

The steps in Approximate Policy Iteration are as follows:

(1) Perform closed loop simulations with chosen suboptimal policies to obtain state x_k vs. cost-to-go $J^i(x_k)$ data; where counter $i = 0$
(2) Use a function approximator to express cost-to-go as a function $\tilde{J}^0(x)$ of state x
(3) Perform *policy improvement* for each visited state x_k according to Eq. (5)
(4) Perform the policy evaluation:
    - Initialize the cost-to-go approximator for policy evaluation $\tilde{J}^{i+1,0} = \tilde{J}^i$; let $j=0$
    - Calculate the new cost-to-go value for each state according to

$$J^{i+1,j+1} = \phi(x, \mu^{i+1}(x)) + J^{i+1,j}\left(f_h(x, \mu^{i+1}(x))\right) \tag{8}$$

    - Increment j; iterate until convergence

    We represent this cost-to-go as $\tilde{J}^{i+1}$.
(5) Increment i and iterate steps 3 to 4 until μ^i converges

Value iteration, on the other hand, is more straightforward. A separate policy evaluation is not required. The result of minimization (step 3) is used to update cost-to-go values directly, using Eq. (7), and the iterations are performed until convergence.

Simulation-based evaluation In the policy iteration algorithm mentioned above, we performed the iterations over all the data points in the memory using one-step ahead simulation of the improved policy. The policy evaluation is then iteratively performed using Eq. (8). We call this method "PI based on iterative evaluation" or iter-PI. An alternative method is to start with some selected initial points and perform simulations using the updated policy $\mu^{i+1}(x)$ until steady state. The policy evaluation for the simulated policy is simply the sum of single stage costs until steady state:

$$J^{i+1}(x_k^{sim,i+1}) = \sum_{l=1}^{\infty} \phi(x_{k+l}^{sim,i+1}, u_{k+l}) \tag{9}$$

Here, the superscript ^{sim} is used to identify that the states visited during the simulations. We call this method sim-PI.

λ-Policy Iteration The λ-policy iteration was proposed by Bertsekas and Ioffe (1996) as a method to accelerate the policy evaluation step by introducing a "discount factor" $\lambda \in [0,\ 1]$ that does not alter the cost-to-go. In λ-policy iteration, we use the temporal difference $\delta(x_k)$ for the state transition $x_k \to x_{k+1}$, defined as:

$$\delta(x_k) \triangleq \phi(x_k, u_k) + J(x_{k+1}) - J(x_k), \tag{10}$$

to modify policy evaluation according to

$$J^{i+1}(x_k) = J^i(x_k) + \sum_{m=0}^{\infty} \lambda^m \delta(x_{k+m}) \tag{11}$$

This method reduces to pure value iteration for $\lambda = 0$ and policy iteration for $\lambda = 1$. Thus, one can view this as some form of "interpolation" between value and policy iteration.

3. LINEAR QUADRATIC CONTROL

Before moving on ADP examples, we compare the value and policy iteration algorithms for an unconstrained linear system. Consider the problem of regulating a linear unconstrained system

$$x_{k+1} = Ax_k + Bu_k \tag{12}$$

to the origin. The single stage cost for this system is $\phi(x,u) = x^TQx + u^TRu$. An analytical solution of Bellman equation exists and it leads to the famous Linear Quadratic Regulator (LQR) result.

The optimal cost-to-go function is a quadratic function of the system state (Bertsekas, 2000) and is denoted as $J(x) = x^TSx$. We are interested in solving the ∞-horizon problem:

$$\min_{u_k} \left\{\left[x_k^TQx_k + u_k^TRu_k\right] + x_{k+1}^TS^{\infty}x_{k+1}\right\} \tag{13}$$

3.1 Analytical Solutions

We can show that the input

$$\begin{aligned} u_k &= \mu(x_k) \\ &\triangleq -L \cdot x_k \\ &= -\left(B^TS^{\infty}B + R\right)^{-1} B^TS^{\infty}A \cdot x_k \end{aligned} \tag{14}$$

minimizes the above objective (13).

Value iteration involves updating the cost function $J = x^T S x$ with the minimizer function obtained by solving Eq. (13) above. Thus,

$$x_k^T S^\infty x_k = \min_{u_k} x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T S^\infty x_{k+1}$$

Substituting u_k from Eq. (14), the solution of above equation is given by Ricatti Equation as in the famous LQR problem:

$$\begin{aligned} S^\infty = {} & A^T S^\infty A + Q - \\ & A^T S^\infty B [B^T S^\infty B + R]^{-1} B^T S^\infty A \end{aligned} \tag{15}$$

Iterative solution of this Ricatti Difference Equation is the value iteration.

Policy iteration is a two-step procedure. First, the improved policy is computed according to Eq. (14) as $\mu^{i+1}(x_k) = -L^{i+1} \cdot x_k$ given current estimate of cost-to-go function $J^i(x) = x^T S^i x$. Next, policy evaluation (Eq. 6) is used to compute the updated cost-to-go function as:

$$\begin{aligned} x_k^T S^{i+1} x_k = {} & x_k^T [Q + L^{(i+1)T} R L^{i+1}] x_k + \\ & x_{k+1}^T S^{i+1} x_{k+1} \end{aligned} \tag{16}$$

Thus, policy evaluation is equivalent to solving a discrete Lyapunov equation:

$$Y S^{i+1} Y^T - S^{i+1} + Z = 0 \tag{17}$$

$$\begin{aligned} \text{where} \quad & Y = [A - BL^{i+1}]^T \\ \text{and} \quad & Z = [Q + L^{(i+1)T} R L^{i+1}]. \end{aligned}$$

sim-PI provides an alternative method that generates new cost-to-go function through simulations of the policy $\mu^{i+1}(x)$. The policy improvement step remains the same (Eq. 14). Using Eq. (9) for the linear system, we can show in a straightforward manner that simulation-based policy evaluation also reduces to the same Lyapunov equation (17) as before.

In **λ-Policy Iteration**, policy improvement step is performed as before (Eq. 14). However, policy evaluation is modified using a discount factor λ, which does not alter the cost-to-go structure for a given policy. Policy evaluation reduces to solving a different Lyapunov equation

$$\tilde{Y} S^{i+1} \tilde{Y}^T - S^{i+1} + Z = 0 \tag{18}$$

$$\text{where } \tilde{Y} = \lambda^{0.5} [A - BL^{i+1}]^T$$

$$\tilde{Z} = \lambda [Q + L^{(i+1)T} R L^{i+1}] + (1 - \lambda) S^{i,rd},$$

and $S^{i,rd}$ is obtained as the solution of one step of RDE given by Eq. (15). Proof skipped for brevity.

λ-policy iteration is a balance between value iteration and policy iteration:

- When $\lambda = 0$, $\tilde{Y} = 0$. Thus, Eq. (18) reduces to $-S^{i+1} + S^{i,rd} = 0$; ie. *0-policy iteration* is nothing but value iteration.
- When $\lambda = 1$, $\tilde{Y} = Y$ and $\tilde{Z} = Z$; ie. *1-policy iteration* is the standard policy iteration.

Comparing Eq. (18) with Eq. (17), we can see that λ-policy iteration involves the following modified policy evaluation step:

$$\begin{aligned} J^{i+1}(x_k) = {} & (1 - \lambda) \left[\phi(x_k, u_k) + J^i(x_{k+1}) \right] \\ & + \lambda \left[\phi(x_k, u_k) + J^{i+1}(x_{k+1}) \right] \end{aligned} \tag{19}$$

3.2 Numerical comparison

The weighting matrices for one-stage cost were chosen as $Q = I_2$ and $R = 0.01$. The matrix S^0 used to initialize value and policy iteration and the optimal S^* matrix, obtained using `dlqr` function in MATLAB, are shown in Table 1. The algorithms were said to be converged when $\|S^{i+1} - S^i\| \leq 0.001$. Consider the following numerical example for the system in (12):

$$A = \begin{bmatrix} 1 & 0 \\ 0.1 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Value and policy iteration converged in 43 and 7 iterations respectively. The converged S^{43} and S^7 matrices shown in Table 1 indicate that the convergence of value iteration is asymptotic, while that of policy iteration is exact. However, policy evaluation itself is an iterative step. We did not use `dlyap` function in Matlab but iterated using

$$S^{i+1,j+1} = Y S^{i+1,j} Y^T + Z \tag{20}$$

to perform policy evaluation; it took 346 evaluations for just the first policy iteration to converge. This indicates a potential problem with policy iteration—policy evaluation can be very slow in converging.

Table 2 demonstrates that the λ-policy iteration is a balance between value iteration and policy iteration. As the λ value decreased to 0, more iterations are needed for cost-to-go function to converge, while the number of policy evaluations required for single policy iteration is reduced.

4. EXAMPLES

We apply the policy iteration algorithm to two numerical examples in this section to demonstrate its convergence properties. A Gaussian kernel-based local averager (Lee, 2004) is used as a cost-to-go function approximator. The cost-to-go value at a query point x_q is given by

Table 1. S^i values for unconstrained linear system

S^0	S^*	Value Iteration S^1	Value Iteration S^{43}	Policy Iteration S^1	Policy Iteration S^7
$\begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1.117 & 1.062 \\ 1.062 & 11.522 \end{bmatrix}$	$\begin{bmatrix} 1.02 & 0.1 \\ 0.1 & 1.99 \end{bmatrix}$	$\begin{bmatrix} 1.117 & 1.061 \\ 1.061 & 11.518 \end{bmatrix}$	$\begin{bmatrix} 1.528 & 5.126 \\ 5.126 & 55.76 \end{bmatrix}$	$\begin{bmatrix} 1.117 & 1.062 \\ 1.062 & 11.522 \end{bmatrix}$

Table 2. λ-policy iteration schemes for linear system, for various values of λ. PolEval1: Number of policy evaluations required in the first policy iteration.

λ	$\lambda = 0$	$\lambda = 0.05$	$\lambda = 0.25$	$\lambda = 0.5$	$\lambda = 0.75$	$\lambda = 0.95$	$\lambda = 1$
Iterations	43	41	34	25	16	8	7
PolEval1	–	4	6	11	24	99	346

$$\tilde{J}(x_q) = \begin{cases} \dfrac{\sum_{i=1}^{N} K_\lambda(x_q, \hat{x}_i) J(\hat{x}_i)}{\sum_{i=1}^{N} K_\lambda(x_q, \hat{x}_i)} & \text{if } N > N_{min} \\ J_{max} & \text{otherwise} \end{cases} \quad (21)$$

where N are the number of data points ($\hat{x}_i$) lying in a pre-defined hyper sphere of radius ρ around x_q. If N is less than a threshold number N_{min}, a very high cost J_{max} is assigned to x_q to avoid extrapolation beyond the data covered region. The Gaussian kernel is given by

$$K_\lambda = \exp\left[-\frac{(x_q - \hat{x}_i)^T (x_q - \hat{x}_i)}{\lambda^2}\right] \quad (22)$$

After convergence of the offline iteration method, the resulting cost-to-go function is then used for online control.

4.1 Linear system with "soft" constraints

In this example, we consider the problem of disturbance rejection for a linear system

$$y(s) = \frac{9.62u(s) + [-19s - 57.3]d(s)}{s^2 + 2.4s + 5.05} \quad (23)$$

with sampling time of $0.2\,sec$, and constraints $|y| \leq 5; |u| \leq 5$. We consider the case where disturbance is $d = 0.8$, for which no sequence of inputs u_k can be found to satisfy state constraints. Hence, constraint softening is required.

Two different PI[2] controllers ($K_c = 0.25$, $K_i = 0.15$; $K_c = 0.5$, $K_i = 0.08$) were used as initial suboptimal policy, with the input moves truncated to satisfy the constraint $|u| \leq 5$. A total of 1200 data points were obtained for various values of d and x_0. Initial cost-to-go approximation was calculated from this data. The weighting matrices for one-stage cost were chosen to be $Q = 1$ and $R = 0.04$. For the points where constraints are violated, one-stage cost were weighed 100 times higher (Lee *et al.*, 2004).

[2] To avoid ambiguity, PI is used for Proportional-Integral. Policy iteration is represented as iter-PI, sim-PI or λ-PI.

Table 3. Convergence properties during offline learning: linear system

	VI	iter-PI	sim-PI
Iterations	17	5	6
Pol. Evaluations	—	53	—
Computation time (s)	6641	2012	986

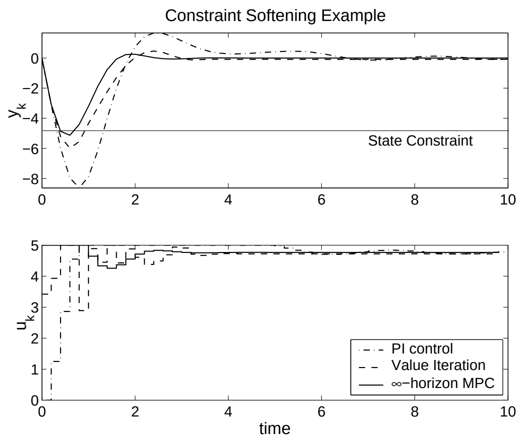

Fig. 1. Online performance of original PI controller, value iteration, and ∞-horizon MPC.

We performed VI and iter-PI for this example. The algorithms were said to be converged when the error between two consecutive iterations was

$$e_{abs} \triangleq \max_{k;k=1,\ldots,1200} |J_k^{i+1} - J_k^i| < 0.001$$

Table 3 shows the comparison between VI and iter-PI. The five policy iterations required a total of 53 policy evaluations. The policy evaluation step requires computation of cost-to-go values for all visited states under the new policy. However, it does not require solving of a minimization problem. Hence, the computational requirement for policy iteration was lower than value iteration.

Figure 1 compares the online performance of ADP with ∞-horizon MPC that is optimal for this example. The online performance of policy iteration without exploration signal is the same as that of value iteration. The ADP method shows significant improvement as compared to the starting suboptimal PI controllers, but is not as good as the optimal ∞-horizon MPC. The MPC controller visits regions of the state space not previously vis-

Table 4. Convergence properties during offline learning for bioreactor example

	VI	iter-PI	λ-PI (0.5)
Iterations	33	3	14
Policy Evaluations	—	*	58
Computation time (hr)	22.9	$> 60^*$	11.0

* Truncated after 40,000 evaluations

ited by the suboptimal schemes; as no information is available in this region, ADP controller avoids this region during online control.

4.2 Nonlinear bioreactor

We now consider a continuous bioreactor containing bacterium Klebsiella oxytoca growing on two substitutable substrates, glucose and arabinose. Based on the preferential utilization of the two substrates, two distinct steady states with vastly different biomass concentrations are obtained. The control objective is to drive the system from an undesirable low biomass to the desirable high biomass yield steady state, under step disturbances in the arabinose feed (s_{2f}). The relevant model and process conditions can be obtained from our previous paper (Kaisare *et al.*, 2003). In Kaisare *et al.* (2003), we used value iteration with neural network as cost approximator for this system. Here, we compare the offline convergence of approximate policy and value policy iteration schemes using Gaussian kernel-based averager.

MPC employing successive linearizations of the nonlinear model (slMPC) was used as the initial suboptimal control law, yielding a total of 1200 data points. The offline convergence of the ADP algorithms using the Gaussian averager is compared in Table 4. Value iteration required 33 iterations and 23 hours of computational time to converge. On the other hand, policy evaluation for the first iteration of iter-PI did not converge even after 2 days of simulation and 40,000 iterations. At this stage, we truncated policy evaluation and used the available cost-to-go approximator to continue policy iterations. We found that only two more iterations (with 18 more policy evaluations) were required for convergence. We therefore applied λ-PI for this system since it provides a tradeoff between iter-PI and VI. As seen in Table 4, λ-PI ($\lambda = 0.5$) showed better offline convergence behavior than VI. We then used the converged solution for online control: all the three ADP controllers gave optimal performance, significantly better than the original slMPC in terms of optimality as well as the computational cost (Kaisare *et al.*, 2003).

5. CONCLUSION AND FUTURE WORK

A comparison between convergence properties of value and policy iteration in ADP was presented in this paper. An LQR example was used to facilitate the understanding of convergence behavior. The ADP schemes using Gaussian kernel-based averager were applied to two different examples. The Gaussian averager guards against extrapolation of the ADP controller in the unvisited subset of the state space. An alternative is to perform guarded exploration to increase the data coverage. In Kaisare *et al.* (2003), we used "policy update" strategy for judiciously increasing the data coverage, using single sweeps of sim-PI within value iteration. Therefore, a comparison of exploration properties of VI and iter-PI is being investigated.

REFERENCES

Bellman, R. E. (1957). *Dynamic Programming.* Princeton Univ. Press. New Jersey.

Bertsekas, D. P. (2000). *Dynamic Programming and Optimal Control.* Vol. 2. Athena Scientific. Belmont, MA.

Bertsekas, D. P. and J. N. Tsitsiklis (1996). *Neuro-Dynamic Programming.* Athena Scientific. Belmont, MA.

Bertsekas, D. P. and S. Ioffe (1996). Temporal difference based policy iteration and applications in neuro-dynamic programming. Report LIDS-P-2349. Lab. for Info. and Decision Systems. MIT, Cambridge, MA.

Blackwell, D. (1965). Discounted dynamic programming. *Annals Math. Stats.* **36**, 226–235.

De Farias, D. P. and B. Van Roy (2000). On the existence of fixed points for approximate value iteration and temporal-difference learning. *J. Optim. Theory Appl.* **105**, 589–608.

Howard, R. (1960). *Dynamic Programming and Markov Proc.* MIT Press. Cambridge MA.

Kaisare, N. S., J. M. Lee and J. H. Lee (2003). Simulation based strategy for nonlinear optimal control: application to a microbial cell reactor. *Int. J. Robust and Nonlinear Control* **13**, 347–363.

Lee, J. M. (2004). A study on architecture, algorithms and applications of approximate Dynamic Programming. PhD thesis. Georgia Tech. Atlanta, GA.

Lee, J. M., N. S. Kaisare and J. H. Lee (2004). Choice of approximator and design of penalty function for an approximate dynamic programming based control approach. *J. Proc. Control.* submitted.

Puterman, M. L. and S. L. Brumelle (1979). On the convergence of policy iteration in stationary DP. *Math. Oper. Res.* **4**, 60–69.

Santos, M. S. and J. Rust (2004). Convergence properties of policy iteration. *SIAM J. Control Optim.* **42**, 2094–2115.

Sutton, R. S. and A. G. Bartow (1998). *Reinforcement Learning: An Introduction.* MIT Press. Cambridge MA.