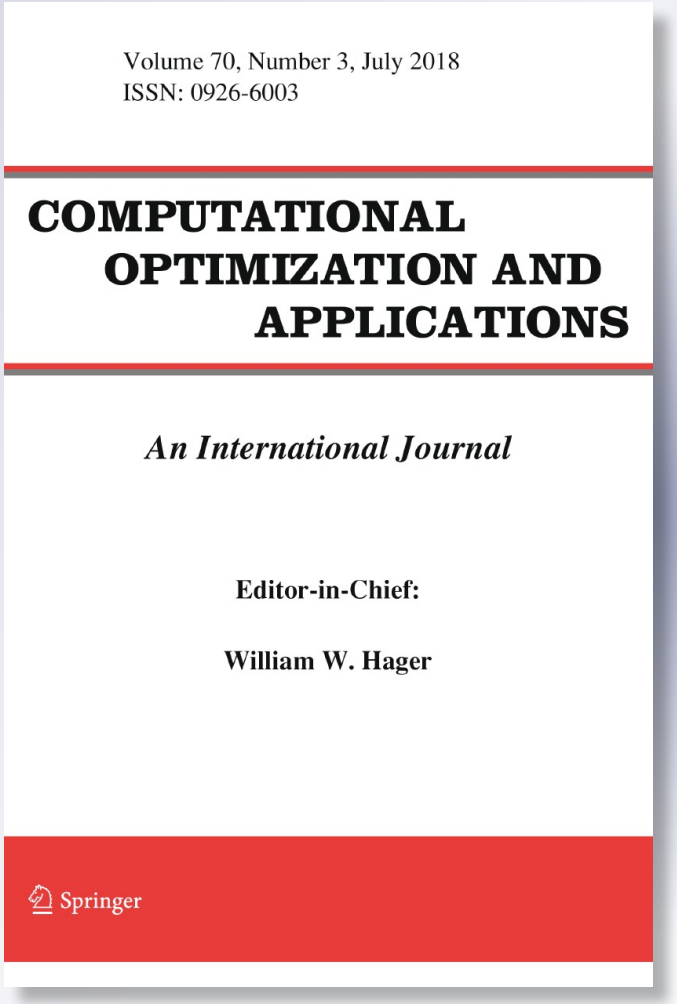
Proximal algorithms and temporal difference methods for solving fixed point problems

Dimitri P. Bertsekas

Computational Optimization and Applications
An International Journal

ISSN 0926-6003
Volume 70
Number 3

Comput Optim Appl (2018) 70:709-736
DOI 10.1007/s10589-018-9990-5



Springer

Comput Optim Appl (2018) 70:709–736
https://doi.org/10.1007/s10589-018-9990-5

Proximal algorithms and temporal difference methods for solving fixed point problems

Dimitri P. Bertsekas[1]

Received: 26 January 2017 / Published online: 2 March 2018

Abstract In this paper we consider large fixed point problems and solution with proximal algorithms. We show that for linear problems there is a close connection between proximal iterations, which are prominent in numerical analysis and optimization, and multistep methods of the temporal difference type such as TD(λ), LSTD(λ), and LSPE(λ), which are central in simulation-based exact and approximate dynamic programming. One benefit of this connection is a new and simple way to accelerate the standard proximal algorithm by extrapolation towards a multistep iteration, which generically has a faster convergence rate. Another benefit is the potential for integration into the proximal algorithmic context of several new ideas that have emerged in the approximate dynamic programming context, including simulation-based implementations. Conversely, the analytical and algorithmic insights from proximal algorithms can be brought to bear on the analysis and the enhancement of temporal difference methods. We also generalize our linear case result to nonlinear problems that involve a contractive mapping, thus providing guaranteed and potentially substantial acceleration of the proximal and forward backward splitting algorithms at no extra cost. Moreover, under certain monotonicity assumptions, we extend the connection with temporal difference methods to nonlinear problems through a linearization approach.

Keywords Proximal algorithm · Temporal differences · Dynamic programming · Convex optimization · Fixed point problems

✉ Dimitri P. Bertsekas
dimitrib@mit.edu

[1] Laboratory for Information and Decision Systems, Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, MA 02139, USA

1 Introduction

In this paper we focus primarily on systems of linear equations of the form

$$x = Ax + b, \tag{1.1}$$

where A is an $n \times n$ matrix and b is a column vector in the n-dimensional space $\Re^n$. We denote by $\sigma(M)$ the spectral radius of a square matrix M (maximum over the moduli of the eigenvalues of M), and we assume the following.

Assumption 1.1 The matrix $I - A$ is invertible and $\sigma(A) \leq 1$.

We consider the proximal algorithm, originally proposed for the solution of monotone variational inequalities by Martinet [45] (see also the textbook treatments by Facchinei and Pang [33], Bauschke and Combettes [4], and the author's [19]). This algorithm has the form

$$x_{k+1} = P^{(c)} x_k,$$

where c is a positive scalar, and for a given $x \in \Re^n$, $P^{(c)}x$ denotes the solution of the following equation in the vector y:

$$y = Ay + b + \frac{1}{c}(x - y).$$

Under Assumption 1.1, this equation has the unique solution

$$P^{(c)}x = \left(\frac{c+1}{c} I - A\right)^{-1} \left(b + \frac{1}{c}x\right), \tag{1.2}$$

because the matrix $\frac{c+1}{c} I - A$ is invertible, since its eigenvalues lie within the unit circle that is centered at $\frac{c+1}{c}$, so they do not include 0.

When A is symmetric, the system (1.1) is the optimality condition for the minimization

$$\min_{x \in \Re^n} \left\{ \frac{1}{2} x' Q x - b' x \right\}, \tag{1.3}$$

where $Q = I - A$ and a prime denotes transposition. The proximal algorithm $x_{k+1} = P^{(c)} x_k$ can then be implemented through the minimization

$$x_{k+1} \in \arg\min_{x \in \Re^n} \left\{ \frac{1}{2} x' Q x - b' x + \frac{1}{2c} \|x - x_k\|^2 \right\},$$

or

$$x_{k+1} = \left(\frac{1}{c} I + Q\right)^{-1} \left(b + \frac{1}{c} x_k\right).$$

In this case, Assumption 1.1 is equivalent to Q being positive definite, with all eigenvalues in the interval $(0, 2]$. Note, however, that for the minimization problem (1.3), the proximal algorithm is convergent for any positive semidefinite symmetric Q, as is well known. Thus Assumption 1.1 is not the most general assumption under which the proximal algorithm can be applied.[1] Still, however, the assumption covers important types of problems, including the case where A is a contraction with respect to some norm, as well as applications in dynamic programming (DP for short), to be discussed shortly.

Let us denote by T the mapping whose fixed point we wish to find,

$$Tx = Ax + b.$$

We will denote by T^ℓ the ℓ-fold composition of T, where ℓ is a positive integer, and we define addition of a finite number and an infinite number of linear operators in the standard way. We introduce the multistep mapping $T^{(\lambda)}$ given by

$$T^{(\lambda)} = (1-\lambda)\sum_{\ell=0}^{\infty}\lambda^\ell T^{\ell+1}, \tag{1.4}$$

where λ is a scalar with $0 < \lambda < 1$. The series defining $T^{(\lambda)}$ is convergent under Assumption 1.1, as we will discuss later. A principal aim of this paper is to establish the relation between the mappings $T^{(\lambda)}$ and $P^{(c)}$, and the ways in which this relation can be exploited algorithmically to compute x^*.

The mapping $T^{(\lambda)}$ has been central in the field that we will refer to as "approximate DP" (the name "reinforcement learning" is also often used in artificial intelligence, and the names "neuro-dynamic programming" and "adaptive dynamic programming" are often used in automatic control, with essentially the same meaning). In particular, $T^{(\lambda)}$ is involved in methods for finding a fixed point of the mapping $\Pi T^{(\lambda)}$, where Π is either the identity or some form of projection onto a low-dimensional subspace S.[2] In the DP context, A is a substochastic matrix related to the Markov chain of a policy and the equation $x = Ax + b$ is the Bellman equation for the cost function x of the policy. Equations of the form $x = Tx$ are solved repeatedly within the exact policy iteration method, which generates a sequence of improved cost functions and associated policies. Equations of the form $x = \Pi T^{(\lambda)}x$ are solved within a corresponding approximate policy iteration method. Detailed accounts of the approximate DP context are given in several books, including the ones by Bertsekas and Tsitsiklis

[1] It is possible to scale the eigenvalues of Q to lie in the interval $(0, 2]$ without changing the problem, by multiplying Q and b with a suitable positive scalar. This, however, requires some prior knowledge about the location of the eigenvalues of Q.

[2] In approximate DP it is common to replace a fixed point equation of the form $x = F(x)$ with the equation $x = \Pi\big(F(x)\big)$. This approach comes under the general framework of Galerkin approximation, which is widely used in a variety of numerical computation contexts (see e.g., the books by Krasnoselskii [39] and Fletcher [34], and the DP-oriented discussion in the paper [68]). A distinguishing feature of approximate DP applications is that F is a linear mapping and the equation $x = \Pi\big(F(x)\big)$ is typically solved by simulation-based methods.

[7], Sutton and Barto [57], Si et al. [50], Powell [48], Busoniu et al. [1], Szepesvari [59], Bertsekas [17], Lewis and Liu [42], and Vrabie, Vamvoudakis, and Lewis [63]. Substantial computational experience has been accumulated with this methodology, and considerable success has been obtained (including prominent achievements with programs that play games, such as Backgammon, Go, and others, at impressive and sometimes above human level; see Tesauro [60], Scherrer et al. [36], [52], and Silver et al. [44], [51]). In challenging approximate DP applications, the dimension of A is very high, the dimension of the approximation subspace S is low by comparison, and the large-scale computations involved in calculating the fixed point of $\Pi T^{(\lambda)}$ are handled by Monte-Carlo simulation schemes.

A variety of simulation-based methods that involve the mapping $T^{(\lambda)}$, such as TD(λ), LSTD(λ), and LSPE(λ), have been proposed in approximate DP. In particular, the fixed point iteration $x_{k+1} = \Pi T^{(\lambda)} x_k$ (where Π is orthogonal projection with respect to a weighted Euclidean norm) has been called PVI(λ) in the author's DP textbook [17] (PVI stands for Projected Value Iteration). Its simulation-based implementation is the LSPE(λ) method (LSPE stands for Least Squares Policy Evaluation) given in joint works of the author with his collaborators Ioffe, Nedić, Borkar, and Yu [2,5,16,46,67]. The simulation-based matrix inversion method that solves the fixed point equation $x = \Pi T^{(\lambda)} x$ is the LSTD(λ) method, given by Bradtke and Barto [23], and further discussed, extended, and analyzed by Boyan [22], Lagoudakis and Parr [40], Nedić and Bertsekas [46], Bertsekas and Yu [10,69], and Yu [71,72] (LSTD stands for Least Squares Temporal Differences). TD(λ), proposed by Sutton [58] in the approximate DP setting, is a stochastic approximation method for solving the equation $x = \Pi T^{(\lambda)} x$. It has the form

$$x_{k+1} = x_k + \gamma_k \Big(sample\big(\Pi T^{(\lambda)} x_k\big) - x_k \Big), \tag{1.5}$$

where $sample\big(\Pi T^{(\lambda)} x_k\big)$ is a stochastic simulation-generated sample of $\Pi T^{(\lambda)} x_k$, and γ_k is a diminishing stepsize satisfying standard conditions for stochastic iterative algorithms, such as $\gamma_k = 1/(k+1)$.[3] The computation of the samples in Eq. 1.5 involves simulation using Markov chains and the notion of temporal differences, which originated in reinforcement learning with the works of Samuel [53,54] on a checkers-playing program. Mathematically, temporal differences are residual-type terms of the form $A^{\ell}(Ax + b - x)$, $\ell \geq 0$, which can be used to streamline various computations within the aforementioned methods. We refer to the sources given above for methods to generate samples of temporal differences in the DP/policy evaluation context, and to [10] for corresponding methods and analysis within the more general linear fixed point context of the present paper.

A central observation of this paper, shown in Sect. 2, is that the proximal mapping $P^{(c)}$ is closely related to the multistep mapping $T^{(\lambda)}$, where

[3] The precise nature of the problem that TD(λ) is aiming to solve was unclear for a long time. The paper by Tsitsiklis and VanRoy [61] showed that it aims to find a fixed point of $T^{(\lambda)}$ or $\Pi T^{(\lambda)}$, and gave a convergence analysis (also replicated in the book [7]). The paper by Bertsekas and Yu [10] (Section 5.3) generalized TD(λ), LSTD(λ), and LSPE(λ) to the linear system context of this paper.

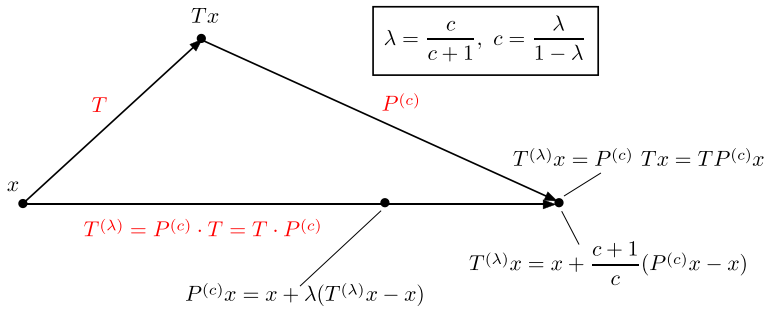


Fig. 1 Relation of the mappings $P^{(c)}$ and $T^{(\lambda)}$. The mapping $P^{(c)}$ is obtained by interpolation between $T^{(\lambda)}$ and the identity. Reversely, $T^{(\lambda)}$ is an extrapolated form of $P^{(c)}$

$$\lambda = \frac{c}{c+1}.$$

In particular $P^{(c)}$ is an interpolated mapping between $T^{(\lambda)}$ and the identity, or reversely, $T^{(\lambda)}$ is an extrapolated form of $P^{(c)}$; see Fig. 1. Moreover, we show in Sect. 2 that under Assumption 1.1, $T^{(\lambda)}$ has a smaller spectral radius than $P^{(c)}$, and as a result extrapolation of the proximal iterates by a factor $\frac{c+1}{c}$ results in convergence acceleration at negligible computational cost. We also characterize the region of extrapolation factors that lead to acceleration of convergence, and show that it is an interval that contains $(1, 1 + 1/c]$, but may potentially be substantially larger. These facts are new to the author's knowledge, and they are somewhat unexpected as they do not seem to readily admit an intuitive explanation.

Aside from its conceptual value and its acceleration potential, the relation between $P^{(c)}$ and $T^{(\lambda)}$ suggests the possibility of new algorithmic approaches for large scale applications where the proximal algorithm can be used conveniently. In particular, one may consider the projected proximal algorithm,

$$x_{k+1} = \Pi P^{(c)} x_k,$$

which aims to converge to a fixed point of $\Pi P^{(c)}$. The algorithm may be based on simulation-based computations of $\Pi T^{(\lambda)}x$, and such computations have been discussed in the approximate DP context as part of the LSPE(λ) method (noted earlier), and the λ-policy iteration method (proposed in [5], and further developed in the book [7], and the papers [18,55]). The simulation-based methods for computing $\Pi T^{(\lambda)}x$ have been adapted to the more general linear equation context in [9,10]; see also [17], Section 7.3. Another possibility is to use simulation-based matrix inversion to solve the fixed point equation $x = \Pi T^{(\lambda)}x$. In the approximate DP context this is the LSTD(λ) method, which has also been extended to the general linear equations context in [10].

For an overview of how to adapt and transfer algorithms between the TD/approximate DP and the proximal contexts, we refer to an extended version of this paper [20]. Our aim there is to highlight the algorithmic possibilities that may allow us to benefit from the accumulated implementation experience within these contexts. To this end, we draw on the individual and joint works of the author, Wang and Yu; see

[9,10,15,16,64,65,68,69,71,72], and the textbook account of [17], Section 7.3, where extensions and analysis of TD(λ), LSTD(λ), and LSPE(λ) for solution of the general linear system $x = \Pi T^{(\lambda)} x$ were given. This includes criteria for $T^{(\lambda)}$ and $\Pi T^{(\lambda)}$ to be a contraction, error bounds, simulation-based implementations, algorithmic variations, dealing with singularity or near singularity of $\Pi P^{(c)}$, etc.

Let us also note that sampling and simulation for solution of linear systems have a long history, starting with a suggestion by von Neumann and Ulam (recounted by Forsythe and Leibler [35]); see also the papers by Curtiss [25,26], and the survey by Halton [38]. More recently, work on simulation methods has focused on using low-order calculations for solving large least squares and other problems. In this connection we note the papers by Strohmer and Vershynin [56], Censor et al. [24], and Leventhal and Lewis [41] on randomized versions of coordinate descent and iterated projection methods for overdetermined least squares problems, and the series of papers by Drineas, Kannan, Mahoney, Muthukrishnan, Boutsidis, and Magdon-Ismail, who consider the use of simulation methods for linear least squares problems and low-rank matrix approximation problems; see [3,27–31].

Our acceleration result of Sect. 2 admits an extension to nonlinear fixed point problems. In particular, in Sect. 3 we show that an extrapolated form of the proximal algorithm provides increased reduction of the distance to the fixed point over the standard proximal algorithm, provided the fixed point problem has a unique solution and involves a nonexpansive mapping (cf. Assumption 1.1). In Sect. 3, we also consider forward–backward splitting algorithms and provide a natural generalization of the extrapolation ideas. To our knowledge, these are the first simple extensions of the proximal and forward–backward algorithms for major classes of nonlinear problems, which guarantee acceleration. Other extrapolation methods, such as the ones of [13] and [14], Section 2.3.1 (for convex optimization), or [32] (for monotone operator problems), guarantee convergence but not acceleration, in the absence of additional prior knowledge.

The convergence theory of temporal difference methods is restricted to linear systems that satisfy Assumption 1.1. Thus, for nonlinear fixed point problems, the connection of temporal difference and proximal algorithms seems considerably weaker. To address this situation, we introduce in Sect. 4 algorithmic ideas based on linearization whereby T is linearized at each iterate x_k, and the next iterate x_{k+1} is obtained with a temporal differences-based (exact, approximate, or extrapolated) proximal iteration using the linearized mapping. This approach is similar to Newton's method for solving nonlinear fixed point problems, where the linearized system is solved exactly, rather than approximately (using a single proximal iteration) as in our case.

2 Interpolation and extrapolation formulas

We first review a known result from [10] regarding the multistep mapping $T^{(\lambda)}$. By repeatedly applying the formula $x = Ax + b$, we can verify that

$$T^{(\lambda)} x = A^{(\lambda)} x + b^{(\lambda)}, \tag{2.1}$$

where

$$A^{(\lambda)} = (1-\lambda)\sum_{\ell=0}^{\infty}\lambda^{\ell}A^{\ell+1}, \qquad b^{(\lambda)} = \sum_{\ell=0}^{\infty}\lambda^{\ell}A^{\ell}b, \tag{2.2}$$

assuming that the series above are convergent. The following proposition shows that under Assumption 1.1, $T^{(\lambda)}$ is well defined by the power series $(1-\lambda)\sum_{\ell=0}^{\infty}\lambda^{\ell}T^{\ell+1}$ [cf. Eq. (1.4)], and that it is a contraction with respect to some norm.

Proposition 2.1 *Let Assumption 1.1 hold and let* $\lambda \in (0,1)$.

(a) *The matrix* $A^{(\lambda)}$ *and the vector* $b^{(\lambda)}$ *are well-defined in the sense that the series in Eq. (2.2) are convergent.*
(b) *The eigenvalues of* $A^{(\lambda)}$ *have the form*

$$\theta_i = (1-\lambda)\sum_{\ell=0}^{\infty}\lambda^{\ell}\zeta_i^{\ell+1} = \frac{\zeta_i(1-\lambda)}{1-\zeta_i\lambda}, \qquad i = 1,\ldots,n, \tag{2.3}$$

where ζ_i, $i = 1,\ldots,n$, *are the eigenvalues of* A. *Furthermore, we have*

$$\sigma(A^{(\lambda)}) < 1, \qquad \lim_{\lambda\to 1}\sigma\big(A^{(\lambda)}\big) = 0.$$

The property $\sigma(A^{(\lambda)}) < 1$ asserted in the preceding proposition, is critical for the subsequent development and depends on the eigenvalues of A being different than 1 (cf. Assumption 1.1). For an intuitive explanation, note that the eigenvalues of $A^{(\lambda)}$ can be viewed as convex combinations of complex numbers from the unit circle at least two of which are different from each other since $\zeta_i \neq 1$ [the nonzero corresponding eigenvalues of A and A^2 are different from each other, cf. Eqs. (2.2), (2.3)]. As a result the eigenvalues of $A^{(\lambda)}$ lie strictly within the interior of the unit circle under Assumption 1.1.

The relation between the proximal mapping $P^{(c)}$ and the multistep mapping $T^{(\lambda)}$ is established in the following proposition, which is illustrated in Fig. 1.

Proposition 2.2 *Let Assumption 1.1 hold, and let* $c > 0$ *and* $\lambda = \frac{c}{c+1}$. *Then:*

(a) $P^{(c)}$ *is given by*

$$P^{(c)} = (1-\lambda)\sum_{\ell=0}^{\infty}\lambda^{\ell}T^{\ell}, \tag{2.4}$$

and can be written as

$$P^{(c)}x = \overline{A}^{(\lambda)}x + \overline{b}^{(\lambda)}, \qquad x \in \Re^n, \tag{2.5}$$

where

$$\overline{A}^{(\lambda)} = (1-\lambda)\sum_{\ell=0}^{\infty}\lambda^{\ell}A^{\ell}, \qquad \overline{b}^{(\lambda)} = \sum_{\ell=0}^{\infty}\lambda^{\ell+1}A^{\ell}b. \tag{2.6}$$

(b) *We have*

$$T^{(\lambda)} = TP^{(c)} = P^{(c)}T, \tag{2.7}$$

and for all $x \in \Re^n$,

$$P^{(c)}x = (1-\lambda)x + \lambda T^{(\lambda)}x, \qquad T^{(\lambda)}x = -\frac{1}{c}x + \frac{c+1}{c}P^{(c)}x, \tag{2.8}$$

or equivalently

$$P^{(c)}x = x + \lambda\big(T^{(\lambda)}x - x\big), \qquad T^{(\lambda)}x = x + \frac{c+1}{c}\big(P^{(c)}x - x\big). \tag{2.9}$$

Proof (a) The inverse in the definition of $P^{(c)}$ [cf. Eq. (1.2)] is written as

$$\left(\frac{c+1}{c}I - A\right)^{-1} = \left(\frac{1}{\lambda}I - A\right)^{-1} = \lambda(I-\lambda A)^{-1} = \lambda\sum_{\ell=0}^{\infty}(\lambda A)^{\ell},$$

where the power series above is convergent by Proposition 2.1(a). Thus, from Eq. (1.2) and the equation $\frac{1}{c} = \frac{1-\lambda}{\lambda}$,

$$\begin{aligned} P^{(c)}x &= \left(\frac{c+1}{c}I - A\right)^{-1}\left(b + \frac{1}{c}x\right) = \lambda\sum_{\ell=0}^{\infty}(\lambda A)^{\ell}\left(b + \frac{1-\lambda}{\lambda}x\right) \\ &= (1-\lambda)\sum_{\ell=0}^{\infty}(\lambda A)^{\ell}x + \lambda\sum_{\ell=0}^{\infty}(\lambda A)^{\ell}b, \end{aligned}$$

which from Eq. (2.6), is equal to $\overline{A}^{(\lambda)}x + \overline{b}^{(\lambda)}$, thus proving Eq. (2.5).

(b) We have for all $x \in \Re^n$, using Eqs. (2.1), (2.2), (2.5) and (2.6),

$$\begin{aligned} TP^{(c)}x &= A\big(\overline{A}^{(\lambda)}x + \overline{b}^{(\lambda)}\big) + b = (1-\lambda)\sum_{\ell=0}^{\infty}\lambda^{\ell}A^{\ell+1}x \\ &\quad + \sum_{\ell=0}^{\infty}\lambda^{\ell+1}A^{\ell+1}b + b = A^{(\lambda)}x + b^{(\lambda)} = T^{(\lambda)}x, \end{aligned}$$

thus proving the left side of Eq. (2.7). The right side is proved similarly. The interpolation/extrapolation formulas (2.8) and (2.9) follow by a straightforward

calculation from Eq. (2.4) and the definition $T^{(\lambda)} = (1-\lambda)\sum_{\ell=0}^{\infty}\lambda^\ell T^{\ell+1}$ [cf. Eq. (1.4)]. As an example, the following calculation shows the left side of Eq. (2.9) [and hence also the equivalent left side of Eq. (2.8)]:

$$\begin{aligned} x+\lambda\big(T^{(\lambda)}x-x\big) &= (1-\lambda)x+\lambda T^{(\lambda)}x \\ &= (1-\lambda)x+\lambda\left((1-\lambda)\sum_{\ell=0}^{\infty}\lambda^\ell A^{\ell+1}x+\sum_{\ell=0}^{\infty}\lambda^\ell A^\ell b\right) \\ &= (1-\lambda)\left(x+\sum_{\ell=1}^{\infty}\lambda^\ell A^\ell x\right)+\sum_{\ell=0}^{\infty}\lambda^{\ell+1}A^\ell b \\ &= \overline{A}^{(\lambda)}x+\overline{b}^{(\lambda)} \\ &= P^{(c)}x. \end{aligned}$$

□

We will now use the extrapolation formulas of Proposition 2.2(b) to construct interesting variants of the proximal algorithm. The next proposition establishes the convergence and convergence rate properties of the proximal and multistep iterations, and shows how the proximal iteration can be accelerated by extrapolation or interpolation.

Proposition 2.3 *Let Assumption 1.1 hold, and let* $c > 0$ *and* $\lambda = \frac{c}{c+1}$. *Then the eigenvalues of* $\overline{A}^{(\lambda)}$ *are*

$$\overline{\theta}_i = \frac{1-\lambda}{1-\zeta_i\lambda}, \qquad i = 1,\ldots,n, \tag{2.10}$$

where ζ_i, $i = 1,\ldots,n$, *are the eigenvalues of* A. *Moreover,* $A^{(\lambda)}$ *and* $\overline{A}^{(\lambda)}$ *have the same eigenvectors. Furthermore, we have*

$$\frac{\sigma(A^{(\lambda)})}{\sigma(A)} \le \sigma(\overline{A}^{(\lambda)}) < 1, \tag{2.11}$$

so $\sigma(A^{(\lambda)}) < \sigma(\overline{A}^{(\lambda)})$ *if* $\sigma(A) < 1$.

Proof Let e_i be an eigenvector of $A^{(\lambda)}$ corresponding to the eigenvalue θ_i. By using the interpolation formula (2.8) and the eigenvalue formula (2.3) for θ_i, we have

$$\begin{aligned} \overline{A}^{(\lambda)}e_i &= (1-\lambda)e_i+\lambda A^{(\lambda)}e_i = \big((1-\lambda)+\lambda\theta_i\big)e_i \\ &= \left((1-\lambda)+\lambda\frac{\zeta_i(1-\lambda)}{1-\zeta_i\lambda}\right)e_i = \frac{1-\lambda}{1-\zeta_i\lambda}e_i. \end{aligned}$$

Hence, $\overline{\theta}_i = \frac{1-\lambda}{1-\zeta_i\lambda}$ and e_i are the corresponding eigenvalue and eigenvector of $\overline{A}^{(\lambda)}$, respectively.

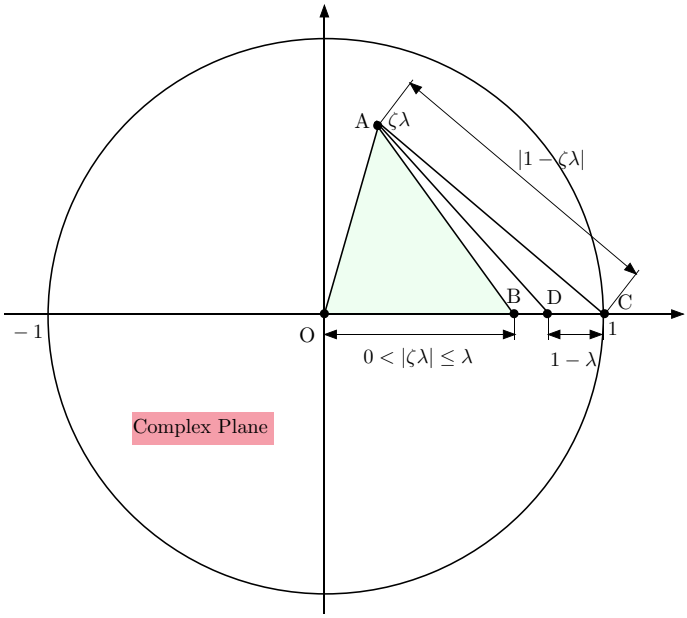


Fig. 2 Proof of the inequality $|\overline{\theta}_i| < 1$, or equivalently that $1 - \lambda < |1 - \zeta\lambda|$ for all complex numbers $\zeta \neq 1$ with $|\zeta| \leq 1$, and $\lambda \in (0, 1)$. We consider the unit circle of the complex plane and the complex number $\zeta\lambda$, and we note that $0 < |\zeta\lambda| \leq \lambda < 1$. If ζ is in the left-hand side of the plane or on the vertical axis, we clearly have $1 - \lambda < 1 \leq |1 - \zeta\lambda|$, so it is sufficient to consider the case where $\zeta \neq 0$ and the real part of ζ is positive, which is depicted in the figure. If ζ is real, we have $\zeta > 0$ as well as $\lambda > 0$, so $|1 - \zeta\lambda| = 1 - \zeta\lambda > 1 - \lambda$, and we are done. If ζ is not real, we consider the isosceles triangle OAB (shaded in the figure), and note that the angles of the triangle bordering the side AB are less than 90°. It follows that the angle ABC and hence also the angle ADC shown in the figure is greater than 90°. Thus the side AC of the triangle ADC is strictly larger than the side DC. This is equivalent to the desired result $1 - \lambda < |1 - \zeta\lambda|$

The proof that $\sigma(\overline{A}^{(\lambda)}) < 1$, or equivalently that $|\overline{\theta}_i| < 1$ for all i, follows from a graphical argument on the complex plane, which is given in the caption of Fig. 2 [An alternative argument is to observe that $\overline{A}^{(\lambda)}$ involves a convex combination of corresponding eigenvalues of the identity and A; cf. Eq. (2.6) and the remark following Proposition 2.1.]

Finally, from Eqs. (2.3) and (2.10), we have

$$|\overline{\theta}_i| = \frac{|\theta_i|}{|\zeta_i|}, \qquad i = 1, \ldots, n,$$

which implies that

$$|\overline{\theta}_i| \geq \frac{|\theta_i|}{\sigma(A)}, \qquad i = 1, \ldots, n.$$

By taking the maximum of both sides over i, we obtain the left side of Eq. (2.11). □

An interesting conclusion can be drawn from Proposition 2.3 about the convergence and the rate of convergence of the proximal iteration $x_{k+1} = P^{(c)}x_k$ and the multistep iteration $x_{k+1} = T^{(\lambda)}x_k$. Under Assumption 1.1, *both iterations are convergent, but the multistep iteration is faster when A is itself a contraction* (with respect to some norm) and is not slower otherwise; cf. Proposition 2.3(a). In the case where A is not a contraction [$\sigma(A) = 1$] it is possible that $\sigma(A^{(\lambda)}) = \sigma(\overline{A}^{(\lambda)})$ (as an example consider a case where all the eigenvalues ζ_i have modulus 1).

Even in the case where $\sigma(A^{(\lambda)}) = \sigma(\overline{A}^{(\lambda)})$, however, it is possible to accelerate the proximal iteration by interpolating strictly between $P^{(c)}x_k$ and $T^{(\lambda)}x_k$. This is shown in the next proposition, which establishes the convergence rate properties of the extrapolated proximal iteration, and quantifies the range of extrapolation factors that lead to acceleration.

Proposition 2.4 *Let Assumption 1.1 hold, and let $c > 0$ and $\lambda = \frac{c}{c+1}$. Consider any iteration that extrapolates from $P^{(c)}$ in the direction of $T^{(\lambda)}$, i.e.,*

$$x_{k+1} = (1-\gamma)P^{(c)}x_k + \gamma T^{(\lambda)}x_k, \qquad \gamma > 0, \tag{2.12}$$

and write it in matrix-vector form as

$$x_{k+1} = A(\lambda,\gamma)x_k + b(\lambda,\gamma),$$

where $A(\lambda,\gamma)$ is an $n \times n$ matrix and $b(\lambda,\gamma) \in \Re^n$. The eigenvalues of $A(\lambda,\gamma)$ are given by

$$\theta_i(\gamma) = (1-\gamma)\overline{\theta}_i + \gamma\theta_i, \qquad i = 1,\ldots,n, \tag{2.13}$$

and we have

$$\sigma\big(A(\lambda,\gamma)\big) < \sigma(\overline{A}^{(\lambda)}), \tag{2.14}$$

for all γ in the interval $(0,\gamma_{\max})$, where

$$\gamma_{\max} = \max\Big\{\gamma > 0 \;\Big|\; |\theta_i(\gamma)| \le \overline{\theta}_i,\ \forall\, i = 1,\ldots,n\Big\}.$$

Moreover, we have $\gamma_{\max} \ge 1$, with equality holding if and only if $\sigma(A) = 1$.

Proof The eigenvalue formula (2.13) follows from the interpolation formula

$$A(\lambda,\gamma) = (1-\gamma)\overline{A}^{(\lambda)} + \gamma A^{(\lambda)},$$

and the fact that $A^{(\lambda)}$ and $\overline{A}^{(\lambda)}$ have the same eigenvectors (cf. Proposition 2.3). For each i, the scalar

$$\max\Big\{\gamma > 0 \;\Big|\; |\theta_i(\gamma)| \le |\overline{\theta}_i|\Big\}$$

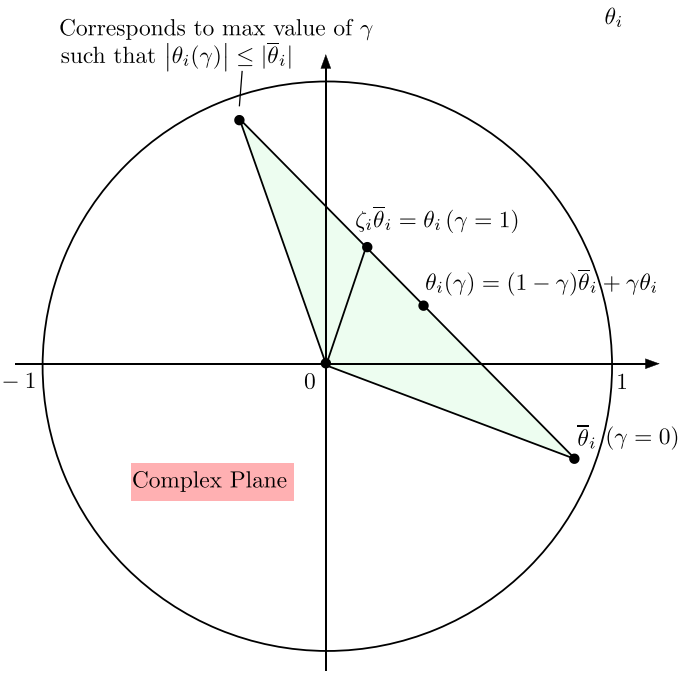


Fig. 3 Illustration of the proof of Proposition 2.4. The eigenvalues $\theta_i(\gamma)$ of $A(\lambda, \gamma)$ are linear combinations (with $\gamma > 0$) of the eigenvalues $\overline{\theta}_i$ and $\theta_i = \zeta_i\overline{\theta}_i$ of $\overline{A}^{(\lambda)}$ and $A^{(\lambda)}$, respectively, and we have $\theta_i \le \overline{\theta}_i$

is the maximum extrapolation factor γ for which $\theta_i(\gamma)$ has at most as large modulus as $\overline{\theta}_i$ (cf. Fig. 3), and the inequality (2.14) follows. The inequality $\gamma_{\max} \ge 1$ follows from the construction of Fig. 3, since $|\theta_i| \le |\overline{\theta}_i|$, $\theta_i \ne \overline{\theta}_i$, and $\gamma = 1$ corresponds to the iteration $x_{k+1} = T^{(\lambda)}x_k$. Finally, we have $\gamma_{\max} = 1$ if and only if $|\theta_i| = |\overline{\theta}_i|$ for some i, which happens if and only if $|\zeta_i| = 1$ for some i, i.e., $\sigma(A) = 1$. □

We may implement the extrapolation/interpolation iteration (2.12) by first implementing the proximal iteration $x_{k+1} = P^{(c)}x_k$ and then the multistep iteration according to

$$x_{k+1} = T^{(\lambda)}x_k = x_k + \frac{c+1}{c}\big(P^{(c)}x_k - x_k\big).$$

In this way, unless $\sigma(A) = 1$ [cf. Eq. (2.11)], we achieve acceleration over the proximal iteration. We may then aim for greater acceleration by extrapolating or interpolating between $P^{(c)}x_k$ and $T^{(\lambda)}x_k$ with some factor, possibly determined by experimentation [strict acceleration can always be achieved with $\gamma \in (0, 1)$]. This provides a simple and reliable method to accelerate the convergence of the proximal algorithm without

knowledge of the eigenvalue structure of A beyond Assumption 1.1.[4] Conversely, we may implement the proximal iteration by interpolating the multistep iteration.

Finally, let us show that the multistep and proximal iterates $P^{(c)}x_k$ and $T^{(\lambda)}x_k$ can be computed by solving fixed point problems involving a contraction of modulus $\lambda\sigma(A)$.

Proposition 2.5 *Let Assumption 1.1 hold, and let $c > 0$ and $\lambda = \frac{c}{c+1}$. The multistep and proximal iterates $T^{(\lambda)}x_k$ and $P^{(c)}x_k$ are the unique fixed points of the contraction mappings W_{x_k} and $\overline{W}_{x_k}$ given by*

$$W_{x_k}x = (1-\lambda)Tx_k + \lambda Tx, \qquad x \in \Re^n,$$

and

$$\overline{W}_{x_k}x = (1-\lambda)x_k + \lambda Tx, \qquad x \in \Re^n,$$

respectively.

Proof Clearly W_{x_k} and $\overline{W}_{x_k}$ are contraction mappings, since they are linear with spectral radius $\lambda\sigma(A) \leq \lambda < 1$. To show that $T^{(\lambda)}x_k$ is the fixed point of W_{x_k}, we must verify that $T^{(\lambda)}x_k = W_{x_k}\big(T^{(\lambda)}x_k\big)$, or equivalently that

$$T^{(\lambda)}x_k = (1-\lambda)Tx_k + \lambda T\big(T^{(\lambda)}x_k\big) = (1-\lambda)Tx_k + \lambda T^{(\lambda)}(Tx_k) \tag{2.15}$$

[here we are applying the formula $T\big(T^{(\lambda)}x\big) = T^{(\lambda)}(Tx)$, which is easily verified using Eqs. (2.1) and (2.2)]. In view of the interpolation formula

$$(1-\lambda)x + \lambda T^{(\lambda)}x = P^{(c)}x, \qquad \forall\, x \in \Re^n, \tag{2.16}$$

[cf. Eq. (2.8)], the right-hand side of Eq. (2.15) is equal to $P^{(c)}(Tx_k)$, which from the formula $T^{(\lambda)} = P^{(c)}T$ [cf. Eq. (2.7)], is equal to $T^{(\lambda)}x_k$, the left-hand side of Eq. (2.15).

Similarly, to show that $P^{(c)}x_k$ is the fixed point of $\overline{W}_{x_k}$, we must verify that $P^{(c)}x_k = \overline{W}_{x_k}(P^{(c)}x_k)$, or equivalently that

$$P^{(c)}x_k = (1-\lambda)x_k + \lambda T\big(P^{(c)}x_k\big).$$

This is proved by combining the formula $T^{(\lambda)} = TP^{(c)}$ [cf. Eq. (2.7)], and the interpolation formula (2.16). □

[4] It is well known that the proximal iteration can be extrapolated by a factor of as much as two while maintaining convergence. This was first shown for the special case of a convex optimization problem in [13], and then for the general case of finding a zero of a monotone operator in [EcB92]; see also a more refined analysis, which quantifies the effects of extrapolation, for the case of a quadratic programming problem, given in [14], Section 2.3.1. However, we are not aware of any earlier proposal of a simple and general scheme to choose an extrapolation factor that maintains convergence *and* simultaneously guarantees acceleration. Moreover, this extrapolation factor, $(c+1)/c$, may be much larger than two.

The fixed point property of the preceding proposition states that $T^{(\lambda)}x$ is the unique solution of the following equation in y:

$$y = (1-\lambda)Tx + \lambda Ty = (1-\lambda)(Ax+b) + \lambda(Ay+b),$$

and thus provides an explicit matrix inversion formula for the multistep mapping $T^{(\lambda)}$:

$$T^{(\lambda)}x = (1-\lambda A)^{-1}\big(b + (1-\lambda)Ax\big). \tag{2.17}$$

This formula should be compared with the formula (1.2) for the proximal mapping, which can be written in terms of λ as

$$P^{(c)}x = (1-\lambda A)^{-1}\big(\lambda b + (1-\lambda)x\big).$$

The fact that the multistep iterate $x_{k+1} = T^{(\lambda)}x_k$ is the fixed point of W_{x_k} is known in exact and approximate DP, and forms the basis for the λ-policy iteration method, first proposed in [5,7], Section 2.3.1. This is a variant of policy iteration where policy evaluation is done by performing a single multistep iteration using the mapping $T^{(\lambda)}$, where A corresponds to the policy being evaluated. The formula (2.17) is given and further discussed in [21], Section 4.3.3. In view of our analysis in this paper, it follows that λ-policy iteration is the approximate version of policy iteration, where the exact policy evaluation phase of the latter (which is to find the fixed point of T), is approximated with a single [extrapolated by a factor $(c+1)/c$] iteration of the proximal algorithm. The λ-policy iteration method admits some interesting simulation-based implementations, which have been discussed in the approximate DP literature ([18,55]), but will not be discussed further here. Based on Proposition 2.5, the proximal iteration $x_{k+1} = P^{(c)}x_k$ admits similar implementations.

Proposition 2.5 also suggests the iteration

$$x_{k+1} = V_m x_k, \tag{2.18}$$

where $V_m x_k$ is obtained by $m > 1$ iterations of the mapping W_{x_k} starting with x_k, i.e.,

$$V_m x_k = (W_{x_k})^m x_k,$$

so $V_m x_k$ is an approximate evaluation of $T^{(\lambda)}x_k$, the fixed point of W_{x_k}. It can be verified by induction that

$$V_m x_k = (1-\lambda)(Tx_k + \lambda T^2 x_k + \cdots + \lambda^{m-1}T^m x_k) + \lambda^m T^m x_k,$$

and that V_m is a contraction mapping [the preceding formula is given in [7], Prop. 2.7(b), while the contraction property of V_m is proved similar to Prop. 3(a) of [10]]. There is also the similar iteration $x_{k+1} = \overline{V}_m x_k$, where $\overline{V}_m x_k$ is obtained by $m > 1$ iterations of the mapping $\overline{W}_{x_k}$ starting with x_k. This iteration may be viewed as an iterative approximate implementation of the proximal algorithm that does not require matrix inversion.

3 Extensions to nonlinear fixed point problems

In this section we consider the solution of the fixed point problem

$$x = T(x), \tag{3.1}$$

where $T : \Re^n \mapsto \Re^n$ is a possibly nonlinear mapping. The proximal algorithm for this problem has the form

$$x_{k+1} = P^{(c)}(x_k), \tag{3.2}$$

where c is a positive scalar, and for a given $x \in \Re^n$, $P^{(c)}(x)$ solves the following equation in the vector y:

$$y = T(y) + \frac{1}{c}(x - y). \tag{3.3}$$

We will operate under assumptions guaranteeing that this equation has a unique solution, so that $P^{(c)}$ will be well defined as a point-to-point mapping.

We focus on the following extrapolated version of the proximal algorithm (3.2):

$$x_{k+1} = E^{(c)}(x_k), \tag{3.4}$$

where

$$E^{(c)}(x) = x + \frac{c+1}{c}\big(P^{(c)}(x) - x\big). \tag{3.5}$$

When T is linear as in Sect. 1, this algorithm coincides with the multistep method $x_{k+1} = T^{(\lambda)}x_k$ (cf. Fig. 1).

The key fact for our purposes is that

$$E^{(c)}(x) = T\big(P^{(c)}(x)\big), \qquad \forall\, x \in \Re^n; \tag{3.6}$$

see Fig. 4. To prove this, we note that from Eq. (3.3) we have

$$P^{(c)}(x) + \frac{1}{c}\big(P^{(c)}(x) - x\big) = T\big(P^{(c)}(x)\big).$$

Using the form (3.5) of $E^{(c)}(x)$ and the preceding equation, we obtain

$$E^{(c)}(x) = x + \frac{c+1}{c}\big(P^{(c)}(x) - x\big) = P^{(c)}(x) + \frac{1}{c}\big(P^{(c)}(x) - x\big) = T\big(P^{(c)}(x)\big),$$

showing Eq. (3.6).

The form of Eq. (3.6) suggests that the extrapolated iteration (3.4) has faster convergence that the proximal iteration (3.2), within contexts where T is contractive with

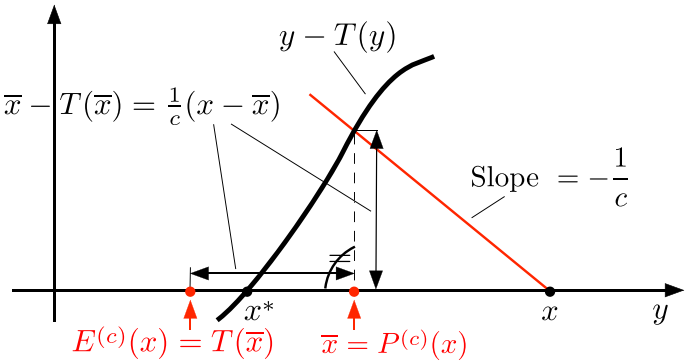


Fig. 4 Illustration of the extrapolated algorithm (3.4). The proximal iterate $P^{(c)}(x)$, denoted $\overline{x}$ in the figure, is extrapolated by $\frac{1}{c}(\overline{x} - x)$. From the definition of $P^{(c)}(x)$, the extrapolated iterate is equal to $T(\overline{x})$ [cf. Eq. (3.6)], and its distance to x^* is strictly smaller than the distance of $\overline{x}$ when T is a contraction

respect to a suitable norm. In particular, if the solution $P^{(c)}(x)$ of Eq. (3.3) exists and is unique for all $x \in \Re^n$, and $P^{(c)}$ and T are contractions with respect to the same norm, then both iterations (3.2) and (3.4) converge to the unique fixed point of T, and the extrapolated iteration converges faster. The following proposition provides specific conditions guaranteeing that this is so.

Proposition 3.1 *Assume that T is a contraction mapping with modulus $\gamma \in (0, 1)$ with respect to a Euclidean norm $\|\cdot\|$, i.e.,*

$$\big\|T(x_1) - T(x_2)\big\| \le \gamma \|x_1 - x_2\|, \qquad \forall\, x_1, x_2 \in \Re^n. \tag{3.7}$$

Then the solution $P^{(c)}(x)$ of Eq. (3.3) exists and is unique for all $x \in \Re^n$, and the mappings $P^{(c)}$ and $E^{(c)}$ are contraction mappings with respect to $\|\cdot\|$. In particular, we have

$$\big\|P^{(c)}(x_1) - P^{(c)}(x_2)\big\| \le \frac{1}{1 + c(1-\gamma)} \|x_1 - x_2\|, \qquad \forall\, x_1, x_2 \in \Re^n,$$
$$\big\|E^{(c)}(x_1) - E^{(c)}(x_2)\big\| \le \frac{\gamma}{1 + c(1-\gamma)} \|x_1 - x_2\|, \qquad \forall\, x_1, x_2 \in \Re^n.$$

Moreover, every sequence $\{x_k\}$ generated by either the proximal algorithm (3.2) or its extrapolated version (3.4) converges geometrically to the unique fixed point x^* of T, and the convergence of the extrapolated version is faster in the sense that

$$\big\|E^{(c)}(x) - x^*\big\| \le \gamma \big\|P^{(c)}(x) - x^*\big\|, \qquad \forall\, x \in \Re^n. \tag{3.8}$$

Proof We first verify that the mapping $x \mapsto x - T(x)$ satisfies the standard strong monotonicity assumption under which the proximal mapping is a contraction. In particular, denoting by $\langle \cdot, \cdot \rangle$ the inner product that defines the Euclidean norm $\|\cdot\|$, and using the Cauchy-Schwarz inequality and Eq. (3.7), we have

$$\begin{aligned}
&\langle x_1 - x_2, x_1 - T(x_1) - x_2 + T(x_2)\rangle \\
&\quad = \|x_1 - x_2\|^2 - \langle x_1 - x_2, T(x_1) - T(x_2)\rangle \\
&\quad \geq \|x_1 - x_2\|^2 - \|x_1 - x_2\| \cdot \big\|T(x_1) - T(x_2)\big\| \qquad \forall\, x_1, x_2 \in \Re^n. \\
&\quad \geq \|x_1 - x_2\|^2 - \gamma \|x_1 - x_2\|^2 \\
&\quad = (1-\gamma)\|x_1 - x_2\|^2,
\end{aligned}$$

This relation shows that the mapping $x \mapsto x - T(x)$ is strongly monotone and from standard results, $P^{(c)}$ is well-defined as a point-to-point mapping and we have

$$\big\|P^{(c)}(x_1) - P^{(c)}(x_2)\big\| \leq \frac{1}{1 + c(1-\gamma)}\|x_1 - x_2\|, \qquad \forall\, x_1, x_2 \in \Re^n,$$

(see [49] or [19], Exercise 5.2). In view of Eq. (3.6) and the contraction property of T, the corresponding contraction property of $E^{(c)}$ and Eq. (3.8) follow. □

3.1 Extrapolation of the forward–backward and proximal gradient algorithms

The forward–backward splitting algorithm applies to the fixed point problem $x = T(x) - H(x)$, where T is a maximally monotone point-to-set mapping in a Euclidean space with inner product $\langle \cdot, \cdot \rangle$ defining a Euclidean norm $\|\cdot\|$, and H is single-valued and strongly monotone, in the sense that for some scalar $\beta > 0$, we have

$$\langle x_1 - x_2, H(x_1) - H(x_2)\rangle \geq \beta \|x_1 - x_2\|^2, \qquad \forall\, x_1, x_2 \in \Re^n.$$

The algorithm has the form

$$x_{k+1} = P^{(\alpha)}\big(x_k - \alpha H(x_k)\big), \qquad \alpha > 0,$$

where $P^{(\alpha)}$ is the proximal mapping corresponding to T; see Fig. 5. This algorithm was analyzed at various levels of generality, by Lions and Mercier [43], Gabay [37], and Tseng [62]. It has been shown to converge to x^* if α is sufficiently small. For a minimization problem where H is the gradient of a strongly convex function, it becomes the popular proximal gradient algorithm; for recent surveys, see Beck and Teboulle [8], Parikh and Boyd [47], and the author's textbook [19] (Ch. 6), among others.

The extrapolated forward–backward algorithm has the form

$$\begin{aligned}
z_k &= x_k - \alpha H(x_k), \qquad \overline{x}_k = P^{(\alpha)}(z_k), \\
x_{k+1} &= \overline{x}_k + \frac{1}{\alpha}(\overline{x}_k - z_k) - H(\overline{x}_k),
\end{aligned}$$

and is illustrated in Fig. 6. It can be seen that

$$x_{k+1} = T(\overline{x}_k) - H(\overline{x}_k)$$

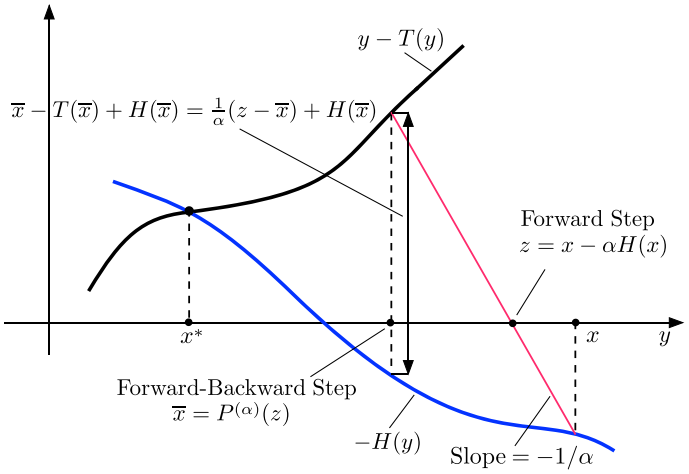


Fig. 5 Illustration of an iteration of the forward backward algorithm

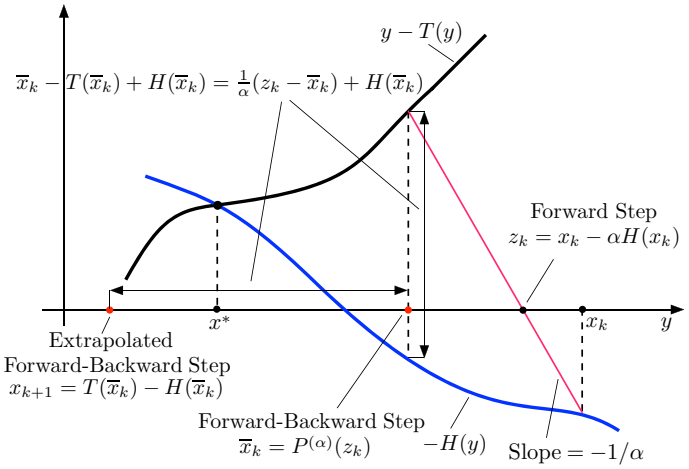


Fig. 6 Illustration of the forward backward algorithm with extrapolation

so there is acceleration if the mapping $T - H$ is contractive. In the linear case where $T(x) = Ax + b$, $H(x) = Bx$, where A and B are $n \times n$ matrices, the algorithm can be related to temporal difference methods, and may be implemented using simulation-based techniques (see [20] for a discussion).

4 Linearized proximal and temporal difference methods for nonlinear problems

The proximal algorithm (3.2) and its extrapolated version (3.5) cannot be related to multistep temporal difference algorithms when T is nonlinear, because then the mapping $P^{(c)}$ does not admit a power series expansion; cf. Eq. (2.4). In this section,

we consider algorithmic ideas based on linearization whereby T is linearized at each iterate x_k, and the next iterate x_{k+1} is obtained with a temporal differences-based (exact, approximate, or extrapolated) proximal iteration using the linearized mapping. This type of algorithm bears similarity to Newton's method for solving nonlinear fixed point problems, the difference being that the linearized system is solved approximately, using a single proximal iteration, rather than exactly (as in Newton's method). The algorithm does not seem to have been considered earlier, to the author's knowledge, although related ideas underlie the λ-policy iteration and optimistic policy iteration methods in DP (see the subsequent discussion).

We focus on the fixed point problem $x = T(x)$ with the ith component of $T(x)$ having the form

$$T(i,x) = \min_{\mu(i)\in M(i)} \big\{a\big(i,\mu(i)\big)'x + b\big(i,\mu(i)\big)\big\}, \qquad x \in \Re^n,\ i = 1,\ldots,n, \tag{4.1}$$

where for each i, $M(i)$ is some set, and $a\big(i,\mu(i)\big)$ and $b\big(i,\mu(i)\big)$ are a (column) vector in $\Re^n$ and scalar, respectively, for each $\mu(i) \in M(i)$. For a given i, this form of $T(i,x)$ includes a very broad class of concave functions of x. Moreover, the case where each $T(i,x)$ is a convex function can be transformed to the concave case through some sign reversals. Intuition suggests that an algorithmic analysis for more general forms of T may be possible, but this is beyond the scope of the present paper.

Let M be the Cartesian product $M(1) \times \cdots \times M(n)$. Given a vector $\mu = \big((\mu(1),\ldots,\mu(n)\big) \in M$, the matrix whose ith row is the vector $a\big(i,\mu(i)\big)'$ is denoted by A_μ and the vector whose ith component is $b\big(i,\mu(i)\big)$ is denoted by b_μ. We denote by T_μ the linear mapping given by

$$T_\mu x = A_\mu x + b_\mu.$$

Our notation here and later is inspired by notation widely used in DP and policy iteration contexts, where i corresponds to state, the components $x(i)$ of x correspond to cost at state i, $\mu(i)$ corresponds to control at state i, $M(i)$ corresponds to the control constraint set at state i, μ corresponds to policy, A_μ and b_μ correspond to the transition probability matrix and cost per stage vector for policy μ, T_μ is the mapping that defines Bellman's equation for the policy μ, and T is the mapping that defines Bellman's equation for the corresponding Markovian decision problem.

We consider the following algorithm. At the typical iteration, given the current iterate x_k, we find $\mu_k(i)$ that attains the minimum over $\mu(i) \in M(i)$ of $a\big(i,\mu(i)\big)'x_k + b\big(i,\mu(i)\big)$, $i = 1,\ldots,n$, and let $\mu_k = \big(\mu_k(1),\ldots,\mu_k(n)\big)$ (the attainment of the minimum will be assumed in what follows). We obtain x_{k+1} via the multistep (extrapolated proximal) iteration

$$x_{k+1} = T_{\mu_k}^{(\lambda)} x_k, \tag{4.2}$$

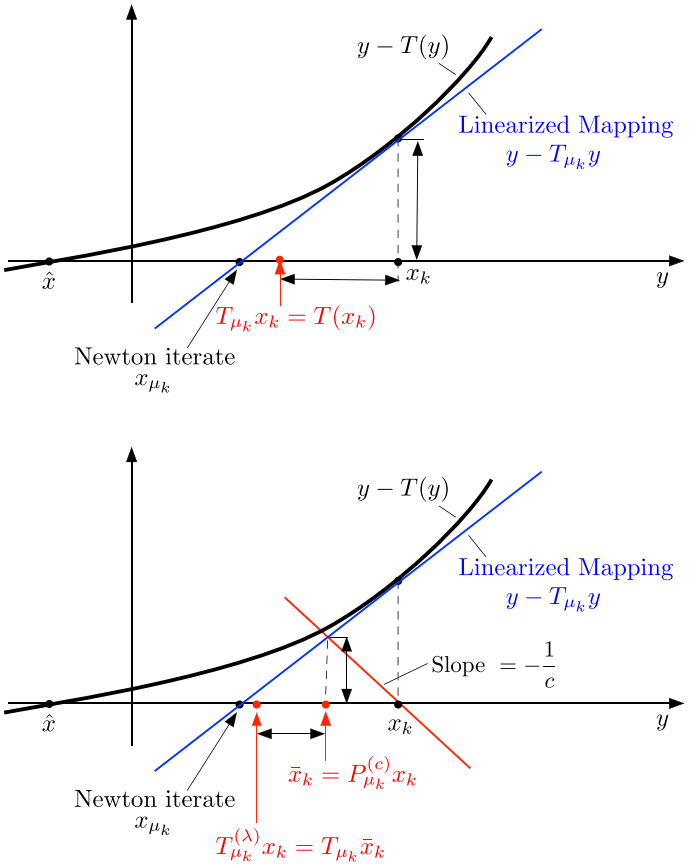


Fig. 7 Illustration of the linearized multistep algorithm (4.2)–(4.3), and its proximal version. At the current iterate x_k, we linearize T and find the proximal iterate $\bar{x}_k = P^{(c)}_{\mu_k} x_k$ that aims to find the fixed point x_{μ_k} of the linearized mapping T_{μ_k}. We can then find the multistep iterate by extrapolation $T^{(\lambda)}_{\mu_k} x_k = T_{\mu_k} \bar{x}_k = \bar{x}_k + \frac{1}{c}(\bar{x}_k - x_k)$; [cf. Eq. (4.2)]. Alternatively, $T^{(\lambda)}_{\mu_k} x_k$ can be found by a temporal differences-based calculation. Note the similarity with the form of Newton's method that finds x_{μ_k}, the unique fixed point of T_{μ_k}, i.e., the iteration $x_{k+1} = x_{\mu_k}$. Newton's method is generally faster but may require much more overhead than the linearized proximal or multistep iteration

where for a given $\lambda \in (0, 1)$, $T^{(\lambda)}_{\mu_k}$ is the multistep mapping corresponding to the linear mapping T_{μ_k},

$$T_{\mu_k} x = A_{\mu_k} x + b_{\mu_k}, \qquad x \in \Re^n; \tag{4.3}$$

cf. Eq. (1.4). The algorithm is illustrated in Fig. 7, together with its proximal version. Note that $a\big(i, \mu_k(i)\big)$ is the gradient of $T(i, \cdot)$ at x_k if $T(i, \cdot)$ is differentiable, and otherwise it is a subgradient of $T(i, \cdot)$ at x_k. This justifies the terms "linearization" and "linearized mapping."

The algorithm (4.2)–(4.3) is related to the λ-policy iteration method for DP problems where $\{\mu_k\}$ is the sequence of generated policies and the fixed point equation $x = T_{\mu_k} x$ corresponds to Bellman's equation for the policy μ_k (see the discussion and references given at the end of Section 2). The algorithm admits several variations where $T_{\mu_k}^{(\lambda)}$ is replaced in Eq. (4.2) by an approximation; for example the iteration (2.18) or the iteration

$$x_{k+1} = P_{\mu_k}^{(c)} x_k, \tag{4.4}$$

where $P_{\mu_k}^{(c)}$ is the proximal mapping corresponding to T_{μ_k}. Another related possibility is the iteration

$$x_{k+1} = T_{\mu_k}^m x_k,$$

where $T_{\mu_k}^m$ is the composition of T_{μ_k} with itself m times ($m \geq 1$). This is related to the optimistic policy iteration method of DP; see [7,17], or [21], Section 2.5.

Let us say that a vector $\mu \in M$ is *proper* if the mapping T_μ has a unique fixed point within $\Re^n$, denoted by x_μ, and we have $T_\mu^k x \to x_\mu$ for all $x \in \Re^n$. Equivalently, μ is proper if and only if A_μ has eigenvalues strictly within the unit circle. If μ is not proper it is called *improper*. The names "proper" and "improper" relate to notions of proper and improper policies in DP and stochastic shortest path problems in particular; see [6,7,17]. Note that for proper μ the algorithmic results of Section 2 come into play. In particular, the multistep and proximal iterations $x_{k+1} = T_\mu^{(\lambda)} x_k$ and $x_{k+1} = P_\mu^{(c)} x_k$ all converge to x_μ starting from any $x_0 \in \Re^n$.

We will assume the following.

Assumption 4.1 (a) For all $x \in \Re^n$ and $i = 1, \ldots, n$, the minimum over $M(i)$ in Eq. (4.1) is attained.

(b) For all $\mu \in M$, the matrix A_μ has nonnegative components.

(c) There exists at least one proper vector, and for each improper vector μ and $x \in \Re^n$, at least one component of the sequence $\{T_\mu^k x\}$ diverges to $+\infty$.

Assumption 4.1(a) is needed to ensure that the algorithm is well defined. Assumption 4.1(b) implies a monotonicity property typically encountered in DP problems, whereby we have for all $\mu \in M$,

$$T_\mu x \leq T_\mu y \qquad \forall\, x, y \in \Re^n \text{ such that } x \leq y, \tag{4.5}$$

as well as

$$Tx \leq Ty \qquad \forall\, x, y \in \Re^n \text{ such that } x \leq y. \tag{4.6}$$

[The relation (4.6) follows from the relation (4.5) by first taking the infimum of the left side to obtain $Tx \leq T_\mu y$ and then by taking again the infimum over $\mu \in M$.] This monotonicity assumption can be replaced by a sup-norm contraction assumption on A_μ, but this requires substantial algorithmic modifications that will be the subject of a separate report. Assumption 4.1(c) is satisfied in particular if all $\mu \in M$ are proper.

We have the following proposition, which ensures that under Assumption 4.1 the algorithm is well defined.

Proposition 4.1 *Let Assumption 4.1 hold. Then:*

(a) *If for some vectors $\mu \in M$ and $x \in \Re^n$ we have $T_\mu x \leq x$, then μ is proper.*
(b) *A sequence $\{x_k, \mu_k\}$ generated by the algorithm (4.2)–(4.3) starting from an initial $x_0 \in \Re^n$ such that $x_0 \geq T(x_0)$ is well defined, and for all k, the vectors μ_k are proper. Moreover, the sequence $\{x_k\}$ is monotonically nonincreasing.*

Proof (a) By the monotonicity of T_μ [cf. Eq. (4.5)], we have $T_\mu^k x \leq x$ for all k, so if μ were improper, Assumption 4.1(c) would be violated.

(b) We first note that for all x and $\mu \in M$, we have

$$x \geq T_\mu x \qquad \Rightarrow \qquad T_\mu x \geq T_\mu^{(\lambda)} x \geq T_\mu \cdot T_\mu^{(\lambda)} x, \tag{4.7}$$

and by part (a), μ is proper. This follows from the power series expansion

$$T_\mu^{(\lambda)} x = (1-\lambda)(T_\mu x + \lambda T_\mu^2 x + \lambda^2 T_\mu^3 x + \cdots),$$

and the fact that $x \geq T_\mu x$ implies that $T_\mu x \geq T_\mu^m x \geq T_\mu^{m+1} x$ for all $m \geq 1$. Moreover we have

$$T_\mu^{(\lambda)} x \geq x_\mu, \tag{4.8}$$

which follows by taking the limit as $m \to \infty$ in the relation

$$\sum_{\tau=0}^{m} \lambda^t T_\mu^{t+1} x \geq \frac{1-\lambda^{m+1}}{1-\lambda} T_\mu^{m+1} x.$$

By the definition of the algorithm, we have $x_0 \geq T(x_0) = T_{\mu_0} x_0$, so by part (a), μ_0 is proper. Since from Eqs. (4.7) and (4.8),

$$x_0 \geq T_{\mu_0} x_0 \geq T_{\mu_0}^{(\lambda)} x_0 = x_1 \geq x_{\mu_0},$$

it follows that $x_1 \in \Re^n$. Continuing this argument for all k, the result follows. □

As noted earlier, for each proper μ, the mapping T_μ has a unique fixed point $x_\mu \in \Re^n$. We introduce the componentwise minimum vector $\hat{x}$, which has components $\hat{x}(i)$ given by

$$\hat{x}(i) = \inf_{\mu:\text{proper}} x_\mu(i), \qquad i = 1, \ldots, n, \tag{4.9}$$

where $x_\mu(i)$ is the ith component of the vector x_μ.

Proposition 4.2 *Let Assumption 4.1 hold and assume that T has at least one fixed point within $\Re^n$. Then:*

(a) *The vector $\hat{x}$ of Eq. (4.9) belongs to $\Re^n$, and is the unique fixed point of T within $\Re^n$.*

(b) *A sequence $\{x_k\}$ generated by the algorithm (4.2) starting from an initial condition x_0 such that $x_0 \geq T(x_0)$ is monotonically nonincreasing and converges to $\hat{x}$.*

Proof (a) Let $x^* \in \Re^n$ be a fixed point of T within $\Re^n$. We will show that $x^* = \hat{x}$. Indeed, using also the monotonicity of T_μ and T [cf. Eqs. (4.5) and (4.6)], we have for every $m \geq 1$ and $\mu \in M$

$$x^* = T^m x^* \leq T_\mu^m x^* \leq \lim_{m\to\infty} T_\mu^m x^* = x_\mu.$$

By taking the infimum of the right side over $\mu \in M$, we obtain $x^* \leq \hat{x}$. For the reverse inequality, let μ^* be such that $x^* = Tx^* = T_{\mu^*}x^*$. Using Proposition 4.1(a), it follows that μ^* is proper, so that x^* is the unique fixed point of T_{μ^*}, i.e., $x^* = x_{\mu^*} \geq \hat{x}$. Thus $x^* = \hat{x}$ and the proof of part (a) is complete.

(b) From the relations (4.7) and (4.8), and the definition of the algorithm we have

$$x_k \geq T(x_k) = T_{\mu_k}x_k \geq T_{\mu_k}^{(\lambda)}x_k = x_{k+1} \geq x_{\mu_k} \geq \hat{x}, \qquad k = 0, 1, \ldots. \tag{4.10}$$

It follows that $x_k \downarrow x_\infty$, where x_∞ is a real-valued n-dimensional vector [since $\hat{x} \in \Re^n$ by part (a)].

Next we show that x_∞ is a fixed point of T. Indeed, from the relation $x_k \geq T(x_k)$ [cf. Eq. (4.10)], we have $x_k \geq T(x_\infty)$ for all k, so that $x_\infty \geq T(x_\infty)$. Also we note that from the relation $T(x_k) \geq x_{k+1}$ we have

$$\lim_{k\to\infty} T(x_k) \geq x_\infty.$$

Moreover for all $\mu \in M$, in view of the linearity of T_μ, we have

$$T_\mu x_\infty = \lim_{k\to\infty} T_\mu x_k \geq \lim_{k\to\infty} T(x_k).$$

By combining the preceding two relations, we obtain $T_\mu x_\infty \geq x_\infty$, so by taking the infimum over $\mu \in M$, we have $T(x_\infty) \geq x_\infty$. This relation, combined with the relation $x_\infty \geq T(x_\infty)$ shown earlier, proves that x_∞ is a fixed point of T within $\Re^n$. From part (a) it follows that $x_\infty = \hat{x}$. □

We can also prove results that are similar to the preceding proposition, but where the nonnegativity Assumption 4.1(b) and the condition $x_0 \geq T(x_0)$ are replaced by alternative conditions. For example, linearization algorithms for finding a fixed point of T are given in the author's monograph [21], Section 2.6.3, under just the assumption that all the matrices A_μ, $\mu \in M$, are contractions with respect to a common weighted sup-norm (see also the papers by Bertsekas and Yu [11,12,70], which relate to the discounted DP and stochastic shortest path contexts). These algorithms, however, do not

use proximal iterations. Another possibility, which involves randomization between proximal iterates involving $T_{\mu_k}^{(\lambda)}$ and nonproximal iterates involving T_{μ_k} is given in Section 5.2 of the extended version of this paper [20]. In any case, an initial vector x_0 with $x_0 \geq T(x_0)$ [cf. the assumption of part (b)] may be obtained in some important cases by adding sufficiently large scalars to the components of some given vector x. For example, suppose that for some μ, the mapping T_μ is a sup-norm contraction. Then given any $x \in \Re^n$, it can be shown that the vector x_0 with components $x(i) + r$ satisfies $x_0 \geq T_\mu x_0 \geq T(x_0)$, provided that the scalar r is sufficiently large.

The proof of Proposition 4.2(b) shows that even without a guarantee of existence of a fixed point of T within $\Re^n$, we have $x_k \downarrow x_\infty$, where x_∞ is a vector that may have some infinite components. For an example of this type, consider the one-dimensional problem of finding a fixed point of the mapping

$$T(x) = \min_{\mu \in (0,1]} \left\{(1 - \mu^2)x - \mu\right\}.$$

Then Assumption 4.1 is satisfied, we have $x_\mu = -1/\mu$, $\hat{x} = -\infty$, $x_k \downarrow \hat{x}$ starting from any $x_0 \in \Re$, while T has no real-valued fixed point.

To see what may happen when there are improper μ and Assumption 4.1(c) is not satisfied, consider the mapping T given by

$$T(x) = \min\{1, x\},$$

which is of the form (4.1) but has multiple fixed points. Here there are two vectors μ. One is $\hat{\mu}$ with $T_{\hat{\mu}} x = 1$, which is proper, and the other is $\bar{\mu}$ with $T_{\bar{\mu}} x = x$, which is improper but does not satisfy Assumption 4.1(c).

5 Concluding remarks

A principal aim of this paper has been to show that proximal and multistep temporal difference methods for linear fixed point problems are closely related, and their implementations can benefit from each other, in both the exact and the approximate simulation-based setting. In particular, within the context of DP, the TD(λ) algorithm for exact policy evaluation, can be written as the stochastic proximal algorithm

$$x_{k+1} = x_k + \gamma_k \Big(sample\big(P^{(c)} x_k\big) - x_k\Big),$$

for solving the linear Bellman equation $x = Tx$ corresponding to a policy [in view of Eq. (1.5), and taking into account the fact that $(T^{(\lambda)} x_k - x_k)$ is the product of $(P^{(c)} x_k - x_k)$ with the scalar $1/\lambda$, where $\lambda = \frac{c}{c+1}$; cf. Fig. 1]. Our Assumption 1.1 is satisfied in broad classes of linear fixed point problems, including problems involving a contraction, and policy evaluation in exact and approximate DP.

Aside from the conceptual and analytical value of the connection between proximal and temporal difference methods, we have shown that under our assumptions, a tangible improvement of the proximal algorithm is possible at no cost. This improvement

is obtained by a simple extrapolation of the proximal iterate, and provides a guaranteed acceleration of convergence (not just guaranteed convergence, like alternative extrapolation schemes). Moreover, this improvement carries over to nonlinear fixed point problems. In addition, our methodology extends naturally to forward–backward splitting and proximal gradient algorithms.

To extend the connection between proximal and temporal difference algorithms, we have also introduced some new proximal-like algorithms for nonlinear fixed point problems. These algorithms are based on linearization, bear a resemblance with Newton's method, and admit temporal differences-based implementations.

Some computational experience with the use of simulation to solve large linear systems, beyond those arising in DP, will be helpful in quantifying the potential benefits of the ideas of this paper and its extended version [20]. Also an interesting question is how to generalize the methods of this paper from the linear equation context to the solution of linear variational inequalities, possibly with a large number of constraints, where both the proximal algorithm and multistep DP-type methods have been applied; see the papers [15,66].

References

1. Busoniu, L., Babuska, R., De Schutter, B., Ernst, D.: Reinforcement Learning and Dynamic Programming Using Function Approximators. CRC Press, New York (2010)
2. Bertsekas, D.P., Borkar, V.S., Nedić, A.: Improved temporal difference methods with linear function approximation. In: Si, J., Barto, A., Powell, W., Wunsch, D. (eds.) Learning and Approximate Dynamic Programming. IEEE Press, New York (2004)
3. Boutsidis, C., Drineas, P., Magdon-Ismail, M.: Near-optimal column-based matrix reconstruction. SIAM J. Comput. **43**, 687–717 (2014)
4. Bauschke, H.H., Combettes, P.L.: Convex Analysis and Monotone Operator Theory in Hilbert Spaces. Springer, New York (2011)
5. Bertsekas, D.P., Ioffe, S.: Temporal differences-based policy iteration and applications in neuro-dynamic programming. Laboratory for Information and Decision Systems Report LIDS-P-2349, MIT (1996)
6. Bertsekas, D.P., Tsitsiklis, J.N.: An analysis of stochastic shortest path problems. Math. OR **16**, 580–595 (1991)
7. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific, Belmont, MA (1996)
8. Beck, A., Teboulle, M.: Gradient-based algorithms with applications to signal-recovery problems. In: Eldar, Y., Palomar, D. (eds.) Convex Optimization in Signal Processing and Communications, pp. 42–88. Cambridge University Press, Cambridge (2010)
9. Bertsekas, D.P., Yu, H.: Solution of large systems of equations using approximate dynamic programming methods. Laboratory for Information and Decision Systems Report LIDS-P-2754, MIT (2007)
10. Bertsekas, D.P., Yu, H.: Projected equation methods for approximate solution of large linear systems. J. Comput. Appl. Math. **227**, 27–50 (2009)
11. Bertsekas, D.P., Yu, H.: Asynchronous distributed policy iteration in dynamic programming. In: Proceedings of Allerton Conference on Communication, Control and Computing, Allerton Park, Ill, pp. 1368–1374 (2010)
12. Bertsekas, D.P., Yu, H.: Q-learning and enhanced policy iteration in discounted dynamic programming. Math. OR **37**, 66–94 (2012)
13. Bertsekas, D.P.: On the method of multipliers for convex programming. IEEE Trans. Auton. Control **20**, 385–388 (1975)
14. Bertsekas, D.P.: Constrained Optimization and Lagrange Multiplier Methods, p. 1997. Academic Press, New York (1982). (Republished by Athena Scientific, Belmont, MA)
15. Bertsekas, D.P.: Temporal difference methods for general projected equations. IEEE Trans. Autom. Control **56**, 2128–2139 (2011)

16. Bertsekas, D.P.: Approximate policy iteration: a survey and some new methods. J. Control Theory Appl. **9**(2011), 310–335 (2011)
17. Bertsekas, D.P.: Dynamic Programming and Optimal Control: Approximate Dynamic Programming, vol. II, 4th edn. Athena Scientific, Belmont, MA (2012)
18. Bertsekas, D.P.: λ-policy iteration: a review and a new implementation. In: Lewis, F., Liu, D. (eds.) Reinforcement Learning and Approximate Dynamic Programming for Feedback Control. IEEE Press, New York (2012)
19. Bertsekas, D.P.: Convex Optimization Algorithms. Athena Scientific, Belmont, MA (2015)
20. Bertsekas, D.P.: Proximal algorithms and temporal differences for large linear systems: extrapolation, approximation, and simulation. Laboratory for Information and Decision Systems Report LIDS-P-3205, MIT (2016)
21. Bertsekas, D.P.: Abstract Dynamic Programming, 2nd edn. Athena Scientific, Belmont, MA (2018). http://web.mit.edu/dimitrib/www/home.html
22. Boyan, J.A.: Technical update: least-squares temporal difference learning. Mach. Learn. **49**, 1–15 (2002)
23. Bradtke, S.J., Barto, A.G.: Linear least-squares algorithms for temporal difference learning. Mach. Learn. **22**, 33–57 (1996)
24. Censor, J., Herman, G.T., Jiang, M.: A note on the behavior of the randomized Kaczmarz algorithm of Strohmer and Vershynin. J. Fourier Anal. Appl. **15**, 431–436 (2009)
25. Curtiss, J.H.: A theoretical comparison of the efficiencies of two classical methods and a Monte Carlo method for computing one component of the solution of a set of linear algebraic equations. In: Proceedings of Symposium on Monte Carlo Methods, pp. 191–233 (1954)
26. Curtiss, J.H.: A Monte Carlo methods for the iteration of linear operators. Uspekhi Mat. Nauk **12**, 149–174 (1957)
27. Drineas, P., Kannan, R., Mahoney, M.W.: Fast Monte Carlo algorithms for matrices I: approximating matrix multiplication. SIAM. J. Comput. **35**, 132–157 (2006)
28. Drineas, P., Kannan, R., Mahoney, M.W.: Fast Monte Calo algorithms for matrices II: computing a low-rank approximation to a matrix. SIAM. J. Comput. **36**, 158–183 (2006)
29. Drineas, P., Mahoney, M.W., Muthukrishnan, S.: Sampling algorithms for L2 regression and applications. In: Proceedings 17th Annual SODA, pp. 1127–1136 (2006)
30. Drineas, P., Mahoney, M.W., Muthukrishnan, S.: Relative-error CUR matrix decompositions. SIAM J. Matrix Anal. Appl. **30**, 844–881 (2008)
31. Drineas, P., Mahoney, M.W., Muthukrishnan, S., Sarlos, T.: Faster least squares approximation. Numer. Math. **117**, 219–249 (2011)
32. Eckstein, J., Bertsekas, D.P.: On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators. Math. Program. **55**, 293–318 (1992)
33. Facchinei, F., Pang, J.-S.: Finite-Dimensional Variational Inequalities and Complementarity Problems. Springer, New York (2003)
34. Fletcher, C.A.J.: Computational Galerkin Methods. Springer, New York (1984)
35. Forsythe, G.E., Leibler, R.A.: Matrix inversion by a Monte Carlo method. Mathematical Tables and Other Aids to Computation **4**, 127–129 (1950)
36. Gabillon, V., Ghavamzadeh, M., Scherrer, B.: Approximate dynamic programming finally performs well in the game of tetris. In: Advances in Neural Information Processing Systems, pp. 1754–1762 (2013)
37. Gabay, D.: Applications of the method of multipliers to variational inequalities. In: Fortin, M., Glowinski, R. (eds.) Augmented Lagrangian Methods: Applications to the Solution of Boundary-Value Problems. North-Holland, Amsterdam (1983)
38. Halton, J.H.: A retrospective and prospective survey of the Monte Carlo method. SIAM Rev. **12**, 1–63 (1970)
39. Krasnoselskii, M.A., et al.: Approximate Solution of Operator Equations. Wolters-Noordhoff Publication, Groningen (1972). Translated by D. Louvish
40. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. J. Mach. Learn. Res. **4**, 1107–1149 (2003)
41. Leventhal, D., Lewis, A.S.: Randomized methods for linear constraints: convergence rates and conditioning. Math. Oper. Res. **35**, 641–654 (2010)
42. Lewis, F.L., Liu, D. (eds.): Reinforcement Learning and Approximate Dynamic Programming for Feedback Control. Wiley, Hoboken, NJ (2013)

43. Lions, P.L., Mercier, B.: Splitting algorithms for the sum of two nonlinear operators. SIAM J. Numer. Anal. **16**, 964–979 (1979)
44. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
45. Martinet, B.: Regularisation d' Inequations Variationnelles par Approximations Successives. Rev. Francaise Inf. Rech. Oper. **4**, 154–158 (1970)
46. Nedić, A., Bertsekas, D.P.: Least squares policy evaluation algorithms with linear function approximation. Discrete Event Dyn. Syst. Theory Appl. **13**, 79–110 (2003)
47. Parikh, N., Boyd, S.: Proximal algorithms. Found. Trends Optim. **1**, 123–231 (2013)
48. Powell, W.B.: Approximate Dynamic Programming: Solving the Curses of Dimensionality. Wiley, New York (2007)
49. Rockafellar, R.T.: Monotone operators and the proximal point algorithm. SIAM J. Control Optim. **14**, 877–898 (1976)
50. Si, J., Barto, A., Powell, W., Wunsch, D. (eds.): Learning and Approximate Dynamic Programming. IEEE Press, New York (2004)
51. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, S., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
52. Scherrer, B., Ghavamzadeh, M., Gabillon, V., Lesner, B., Geist, M.: Approximate modified policy iteration and its application to the game of tetris. J. Mach. Learn. Res. **16**, 1629–1676 (2015)
53. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM J. Res. Dev. **3**, 210–229 (1959)
54. Samuel, A.L.: Some studies in machine learning using the game of checkers. II–recent progress. IBM J. Res. Dev. **11**, 601–617 (1967)
55. Scherrer, B.: Performance bounds for λ-policy iteration and application to the game of tetris. J. Mach. Learn. Res. **14**, 1181–1227 (2013)
56. Strohmer, T., Vershynin, R.: A randomized Kaczmarz algorithm with exponential convergence. J. Fourier Anal. Appl. **15**, 262–278 (2009)
57. Sutton, R.S., Barto, A.G.: Reinforcement Learning. MIT Press, Cambridge, MA (1998)
58. Sutton, R.S.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**, 9–44 (1988)
59. Szepesvari, C.: Algorithms for Reinforcement Learning. Morgan and Claypool Publishers, San Rafael (2010)
60. Tesauro, G.J.: TD-gammon, a self-teaching backgammon program, achieves master-level play. Neural Comput. **6**, 215–219 (1994)
61. Tsitsiklis, J.N., Van Roy, B.: An analysis of temporal-difference learning with function approximation. IEEE Trans. Autom. Control **42**, 674–690 (1997)
62. Tseng, P.: Applications of a splitting algorithm to decomposition in convex programming and variational inequalities. SIAM J. Control Optim. **29**, 119–138 (1991)
63. Vrabie, D., Vamvoudakis, K.G., Lewis, F.L.: Optimal adaptive control and differential games by reinforcement learning principles. The Institution of Engineering and Technology, London (2013)
64. Wang, M., Bertsekas, D.P.: Stabilization of stochastic iterative methods for singular and nearly singular linear systems. Math. Oper. Res. **39**, 1–30 (2013)
65. Wang, M., Bertsekas, D.P.: Convergence of iterative simulation-based methods for singular linear systems. Stoch. Systems **3**, 39–96 (2014)
66. Wang, M., Bertsekas, D.P.: Incremental constraint projection methods for variational inequalities. Math. Program. **150**, 321–363 (2015)
67. Yu, H., Bertsekas, D.P.: Convergence results for some temporal difference methods based on least squares. IEEE Trans. Auton. Control **54**, 1515–1531 (2006)
68. Yu, H., Bertsekas, D.P.: Error bounds for approximations from projected linear equations. Math. Oper. Res. **35**, 306–329 (2010)
69. Yu, H., Bertsekas, D.P.: Weighted Bellman equations and their applications in dynamic programming. Laboratory for Information and Decision Systems Report LIDS-P-2876, MIT (2012)
70. Yu, H., Bertsekas, D.P.: Q-learning and policy iteration algorithms for stochastic shortest path problems. Ann. Oper. Res. **208**, 95–132 (2013)

71. Yu, H.: Convergence of least squares temporal difference methods under general conditions. In: Proceedings of the 27th ICML, Haifa, Israel (2010)
72. Yu, H.: Least squares temporal difference methods: an analysis under general conditions. SIAM J. Control Optim. **50**, 3310–3343 (2012)