Comput Optim Appl (2008) 40: 1–12
DOI 10.1007/s10589-007-9073-5

How good are projection methods for convex feasibility problems?

Nicholas I.M. Gould

Received: 21 September 2006 / Revised: 4 January 2007 / Published online: 23 October 2007

Abstract We consider simple projection methods for solving convex feasibility problems. Both successive and sequential methods are considered, and heuristics to improve these are suggested. Unfortunately, particularly given the large literature which might make one think otherwise, numerical tests indicate that in general none of the variants considered are especially effective or competitive with more sophisticated alternatives.

Keywords Projection methods · Convex feasibility problems · Numerical evaluation

1 Introduction

One of the basic tasks in computational science is to find a point satisfying a given set of equations and/or inequalities. In general this may be a very hard problem, but fortunately for many realistic examples, the solution set of feasible points is convex. Over the years, a number of simple methods have been proposed for such problems, in which a sequence of projections into simpler regions defined by subsets of

This work was supported by the EPSRC grant GR/S42170.

Electronic supplementary material The online version of this article (http://dx.doi.org/10.1007/s10589-007-9073-5) contains supplementary material, which is available to authorized users.

N.I.M. Gould (✉)
Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0QX, England, UK
e-mail: n.i.m.gould@rl.ac.uk

N.I.M. Gould
Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, England, UK
e-mail: nick.gould@comlab.ox.ac.uk

the required equations/inequalities are combined in appropriate ways. Many of these methods have strong convergence guarantees [2, 6, 9], and their simplicity makes them appear most appealing for large-scale computation. However, despite the large number of theoretical papers devoted to generalizations and convergence issues, there appears to have been little effort to investigate how they really perform in practice. In this paper, we attempt to do so in perhaps the most favourable circumstances, for which the feasible region is defined as the intersection of two convex sets and for which individual projections may be easily computed but for which the joint projection may not.

The particular case we investigate is the linear-feasibility problem. Here we wish to find a point $x \in \mathbb{R}^n$ satisfying

$$Ax = b \quad \text{and} \tag{1.1a}$$

$$x^l \le x \le x^u, \tag{1.1b}$$

where A is m $(\le n)$ by n and, for simplicity, of full rank, and any/all of the bounds x^l and x^u may be infinite. Given an initial point x_0, many of the methods described in the literature aim to find the x which is "closest" to x_0 in a well-defined sense; in the methods we will consider here, closest will be measured in terms of minimizing $\|x - x_0\|_2$.

The problem (1.1) is important both in its own right, and as a way of means to solving other optimization problems—for example, the solution set for the linear-programming problem (in standard form),

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ c^T x \quad \text{subject to } Ax = b \text{ and } x \ge 0$$

may be expressed as

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ c^T & -b^T & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} b \\ c \\ 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} x \\ z \end{pmatrix} \ge 0,$$

where the unknowns y and z are Lagrange multipliers and dual variables respectively.

The paper is organised as follows. Sequential and successive projection methods are reviewed in Sect. 2.1, and suggestions on how they might be accelerated for the special problem (1.1) are proposed in Sect. 3. The results of numerical experiments are reported in Sect. 4, and conclusions drawn in Sect. 5.

2 Projection methods

We now examine the basic classes of projection methods that have been proposed. We consider the general problem for which $\mathcal{C}_i$, $i \in \mathcal{S} \overset{\text{def}}{=} \{1, \ldots, s\}$, are closed, convex sets, and we wish to find a point $x \in \mathcal{C} \overset{\text{def}}{=} \bigcap_{i\in\mathcal{S}} \mathcal{C}_i$. For simplicity, we restrict ourselves to finite-dimensional spaces, but note that much of what we say can be generalized to Hilbert space [1–3].

2.1 Simple projections

We denote the (orthogonal) projection $P_i(x)$ of x onto $\mathcal{C}_i$ as

$$P_i(x) = \arg\min_{y \in \mathcal{C}_i} F(y, x), \tag{2.2}$$

where $F(y,x) = \frac{1}{2}\|y - x\|_2^2$. This is an important special case of a very general framework in which $F(y,x) = f(y) - f(x) - \langle \nabla_x f(x), y - x \rangle$ for a given inner product $\langle \cdot, \cdot \rangle$, where the differentiable f is a so-called Bregman function [9, Sect. 2.1]. Our fundamental assumption will be that we are able to solve (2.2) at some acceptable cost for each $i \in \mathcal{S}$.

For the system (1.1) of most interest to us, if we let

$$\mathcal{C}_1 = \mathcal{C}_B \overset{\text{def}}{=} \{x \mid x^l \le x \le x^u\} \quad \text{and} \tag{2.3a}$$

$$\mathcal{C}_2 = \mathcal{C}_A \overset{\text{def}}{=} \{x \mid Ax = b\}, \tag{2.3b}$$

then

$$P_1(x) = P_B(x) \overset{\text{def}}{=} \text{mid}(x^l, x, x^u), \tag{2.4}$$

the component-wise median of the vectors x^l, x and x^u, while $P_2(x) = P_A(x) \overset{\text{def}}{=} p(x)$, where $(p(x), q(x))$ satisfy the linear system

$$\begin{pmatrix} I & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} p(x) \\ q(x) \end{pmatrix} = \begin{pmatrix} x \\ b \end{pmatrix}. \tag{2.5}$$

Certainly (2.4) is trivial, while in many cases (2.5) may be solved by a suitable factorization. More generally, given finite collections of sets $\{\mathcal{B}_j\}_{j=1}^{s_b}$ and $\{\mathcal{L}_i\}_{i=1}^{s_l}$ for which $\bigcup_j \mathcal{B}_j = \mathcal{N} \overset{\text{def}}{=} \{1, \ldots, n\}$ and $\bigcup_i \mathcal{L}_i = \mathcal{M} \overset{\text{def}}{=} \{1, \ldots, m\}$, we might choose

$$\mathcal{C}_j = \{x \mid x_i^l \le x_i \le x_i^u \quad \text{for all } i \in \mathcal{B}_j\}, \quad j = 1, \ldots s_b, \quad \text{and} \tag{2.6a}$$

$$\mathcal{C}_{s_b+i} = \{x \mid a_j^T x = b_j \quad \text{for all} j \in \mathcal{L}_i\}, \quad i = 1, \ldots s_l. \tag{2.6b}$$

In this case

$$(P_j(x))_i = \left\{ \begin{array}{ll} \text{mid}(x_i^l, x_i, x_i^u) & \text{for } i \in \mathcal{B}_j \\ x_i & \text{for } i \notin \mathcal{B}_j \end{array} \right\} \quad \text{for } j = 1, \ldots s_b \tag{2.7}$$

and $P_{s_b+i}(x) = p_i(x)$, where $(p_i(x), q_i(x))$ satisfy the linear system

$$\begin{pmatrix} I & A_{\mathcal{L}_i}^T \\ A_{\mathcal{L}_i} & 0 \end{pmatrix} \begin{pmatrix} p_i(x) \\ q_i(x) \end{pmatrix} = \begin{pmatrix} x \\ b_{\mathcal{L}_i} \end{pmatrix} \tag{2.8}$$

and $A_{\mathcal{L}_i}$ and $b_{\mathcal{L}_i}$ denote the rows of A and entries of b indexed by $\mathcal{L}_i$, for $i = 1, \ldots s_l$. In the extreme case $\mathcal{B}_j = \{j\}$, $j = 1, \ldots n$, and $\mathcal{L}_i = \{i\}$, $i = 1, \ldots m$, all the projections are onto spaces, or segments of spaces, of dimension one and thus trivial to compute.

2.2 Successive projection methods

Armed with suitable ways to apply projections, perhaps the simplest way to try to find a point in $\mathcal{C}$ is to project successively onto the sets $\mathcal{C}_i$. In particular, we have the following scheme.

Algorithm 2.1: Successive projection.

Given x_0 and $\epsilon \in (0, 1)$, for $k = 0, 1, \ldots$ until convergence, set

$$x_{k+1} = (1 - \alpha_k)x_k + \alpha_k P_{j(k)}(x_k),$$

where $j(k) = k \bmod s + 1$, for some $\alpha_k > \epsilon$.

This method is sometimes known as cyclic or alternating projection. The simplest case, when $\alpha_k \equiv 1$, was first analysed for the case where $\{\mathcal{C}_i\}_{i\in\mathcal{S}}$ are affine subspaces by von Neumann [31] (when $s = 2$) and Halperin [20] (for $s > 2$); in this case the algorithm converges to the (orthogonal) projection $P(x_0)$ of x_0 onto $\mathcal{C}$ when $\mathcal{C} \neq \emptyset$. Subsequently, the convergence for more general sets has been addressed, and in particular for polyhedral $\mathcal{C}_i$ (which includes the descriptions of (1.1) given in Sect. 2.1), convergence to a point within $\mathcal{C} \neq \emptyset$ at a globally linear rate has been established [3, Theorem 5.6.3]. For affine subspaces $\mathcal{C}_i$ with non-empty mutual intersection, the linear convergence factor is the "cosine of the angles between the subspaces", [3, Theorem 5.7.8], and one might anticipate that a similar factor involving projections onto "active" constraints would hold in the polyhedral case. [11]. If the projection $P(x_0)$ is required in the non-affine case, it may be necessary to add recursively-computed outward normals before applying each projection onto non-affine $\mathcal{C}_i$—this is Dykstra's algorithm [15] (see also [21]), and when $\mathcal{C}_i = \{x \mid a_i^T x \geq b_i\}$, this is equivalent to Hildreth's method [22].

The extension to non-unit α_k is due to Gubin, Polyak and Raik [19]. Convergence for general $\alpha_k \in [\epsilon, 2-\epsilon]$ has been established for special classes of $\mathcal{C}_i$ [9, Sect. 5.2]. Generalisations in which $j(k)$ need only be ℓ infinitely often for each $\ell \in \mathcal{S}$, but not necessarily cyclically so, have also been suggested. For the very special case for which we simply wish to find a solution to $Ax = b$, and $\mathcal{C}_i = \{x | a_i^T x = b_i\}$, $i = 1, \ldots, m$, the above algorithm is that of Kaczmarz [26] when $\alpha_k = 1$.

2.3 Simultaneous projection methods

Rather than successively projecting the current iterate onto individual constraint sets is to project simultaneously onto all of the sets, and then form a strictly convex sum of these projections. We summarize this as follows.

The advantage here is that convergence is ensured even when $\mathcal{C} = \emptyset$, and in the latter case x_k converges to the point for which

$$\sum_{i\in\mathcal{S}} \lambda_i \| P_i(x) - x \|_2^2$$

is minimized. The method may actually be interpreted in the simultaneous projection framework within an appropriate product space when $\alpha_k = 1$ [29], and thus convergence ensured using known results. In particular, for the case of affine sets, this is Cimmino's method [10]. Convergence for any $\alpha_k \in [\epsilon, 2-\epsilon]$ has been established for various classes of $\mathcal{C}_i$ [27, 30].

Algorithm 2.2: Simultaneous projection.

Given x_0, and $\epsilon \in (0,1)$, choose

$$\lambda_i > 0,\ i \in \mathcal{S}, \quad \text{for which } \sum_{i\in\mathcal{S}} \lambda_i = 1.$$

For $k = 0, 1, \ldots$ until convergence, set

$$x_{k+1} = (1-\alpha_k)x_k + \alpha_k \sum_{i\in\mathcal{S}} \lambda_i P_i(x_k)$$

for some $\alpha_k \geq \epsilon$.

If the projection $P(x_0)$ is required, as for the successive projection case it may be necessary to add recursively-computed outward normals before applying each projection onto non-affine $\mathcal{C}_i$ [24, 25].

3 Stepsize choice for the linear-feasibility problem

We now return to the linear-feasibility problem (1.1), and consider the methods proposed in Sect. 2 for the bi-projection case for which $\mathcal{C}_1 = \mathcal{C}_B$ and $\mathcal{C}_2 = \mathcal{C}_A$ as in (2.3). In this case, it is convenient to rewrite Algorithm 2.1 as follows.

Algorithm 3.1: Successive bi-projection.

Given x_0 and $\epsilon \in (0,1)$, for $k = 0, 1, \ldots$ until convergence, set

$$x_{k+1} = (1-\alpha_k^A)x_k^B + \alpha_k^A P_A(x_k^B), \tag{3.1}$$

where

$$x_k^B = (1-\alpha_k^B)x_k + \alpha_k^B P_B(x_k),$$

for some $\alpha_k^A, \alpha_k^B > \epsilon$.

While, as we have indicated, it is usual to pick the stepsizes $\alpha_k^A, \alpha_k^B \in [\epsilon, 2-\epsilon]$, there is some evidence to suggest that larger stepsizes may be beneficial [29]. In particular, Bauschke and Kruk [5] present numerical results that suggest picking $\alpha_k^A = 1$ and $\alpha_k^B = 2$ is better than other $\alpha_k^A, \alpha_k^B \in [0, 2]$, along with an accompanying proof of convergence for this case. More recently, Bauschke, Combettes and Kruk [4] have investigated the $\alpha_k^A = 1$ case in more detail—since projection onto C_A is linear, (3.1) may be written as

$$x_{k+1} = (1 - \alpha_k)x_k + \alpha_k P_A(P_B(x_k))$$

so long as $x_k \in \mathcal{C}_A$, where we have replaced α_k^B by α_k—and have established convergence for α_k (significantly) larger than 2.

How should we pick α_k? Since $\{x_k\}_{k\geq 1} \in \mathcal{C}_A$, it is convenient to measure convergence in terms of $\|P_B(x_k) - x_k\|_2$. Thus, if we let

$$x(\alpha) = (1 - \alpha)x_k + \alpha P_A(P_B(x_k)),$$

it seems reasonable to pick α_k to minimize $\phi(\alpha) = \|P_B(x(\alpha)) - x(\alpha)\|_2^2$.

Since ϕ is a differentiable convex piecewise-quadratic function of α, a global minimizer is easy to find. Indeed, each component of $P_B(x(\alpha)) - x(\alpha)$ has derivative discontinuities at most two values ("breakpoints") of α. Thus arranging the at-most $2n$ breakpoints of $P_B(x(\alpha)) - x(\alpha)$ using a heapsort [32], the resulting quadratic may be examined between increasing pairs of breakpoints until the global minimizer is found. The slope and curvature of the quadratic may be updated trivially as breakpoints are traversed, and the overall cost of the stepsize determination is $O(n \log n)$ integer and floating-point operations.

Turning to simultaneous projection, Algorithm 2.2 simplifies as follows.

Algorithm 3.2: Simultaneous bi-projection.

Given x_0, $\lambda \in (0, 1)$ and $\epsilon \in (0, 1)$, for $k = 0, 1, \ldots$ until convergence, set

$$x_{k+1} = (1 - \alpha_k)x_k + \alpha_k [\lambda P_A(x_k) + (1 - \lambda) P_B(x_k)]$$

for some $\alpha_k \geq \epsilon$.

Here progress is measured in terms of $\lambda\|P_A(x) - x\|_2^2 + (1 - \lambda)\|P_B(x) - x\|_2^2$. Thus it would seem reasonable to consider the line

$$x(\alpha) = (1 - \alpha)x_k + \alpha z_k, \quad \text{where } z_k = \lambda P_A(x_k) + (1 - \lambda) P_B(x_k),$$

and to pick α_k to minimize $\psi(\alpha) = \lambda\|P_A(x(\alpha)) - x(\alpha)\|_2^2 + (1 - \lambda)\|P_B(x(\alpha)) - x(\alpha)\|_2^2$. As for the successive bi-projection case, ψ is a differentiable convex piecewise-quadratic function of α and thus its global minimizer is easy to find.

One small implementational issue is that in order to evaluate $\psi(\alpha)$, it would appear at first sight that we need to evaluate

$$P_A((1 - \alpha)x_k + \alpha z_k) = (1 - \alpha)P_A(x_k) + \alpha P_A(z_k)$$

and thus that we need to compute (potentially expensive) projections at both x_k and z_k. But of course we may subsequently recur

$$P_A(x_{k+1}) = (1-\alpha_k)P_A(x_k) + \alpha_k P_A(z_k)$$

to start the next iteration, and thus each iteration actually only requires a single projection $P_A(z_k)$ into $\mathcal{C}_A$.

4 Numerical experience

We now turn to our numerical experience with the algorithms we discussed in Sect. 3 for the linear feasibility problem (1.1). We have implemented both Algorithms 3.1 and 3.2 as a Fortran 95 module `LCF` (`L`inearly-`C`onstrained `F`easibility) as part of the upcoming release 2.0 of the nonlinear optimization library GALAHAD [18].

`LCF` is actually designed to find a feasible point for the general linear constraint set

$$c^l \le Ax \le c^u \quad \text{and} \quad x^l \le x \le x^u. \tag{4.1}$$

Here any or all of the components of the bounds c^l, c^u, x^l and x^u may be identical or infinite, thus allowing equality constraints and fixed or free variables. Although the details and bookkeeping are more complicated than for (1.1), the underlying method is essentially to introduce slack variables c and to find a feasible point for the equivalent set

$$Ax - c = 0 \quad \text{and} \quad \begin{pmatrix} x^l \\ c^l \end{pmatrix} \le \begin{pmatrix} x \\ c \end{pmatrix} \le \begin{pmatrix} x^u \\ c^u \end{pmatrix}. \tag{4.2}$$

The dominant cost per iteration of each algorithm is, of course, the cost of computing the projection $P_A(x)$. As we suggested in Sect. 2.1, this projection may be calculated as the solution to the symmetric, indefinite (augmented) linear system (2.5), but sometimes it is more convenient to obtain it using the range-space/Schur complement approach, for which

$$p(x) = x - A^T q(x), \quad \text{where } AA^T q(x) = Ax - b$$

and the crucial matrix AA^T is positive definite. `LCF` uses another GALAHAD module `SBLS` (itself built on top of the HSL [23] package `MA27` [14]) to find the projection. `SBLS` uses the Schur complement method by default, but switches to the augmented system approach if A has any relatively dense columns or if AA^T proves to be too ill-conditioned. `MA27` may be trivially replaced by the more powerful (but commercial) package `MA57` [13] if desired.

We consider four strategies. The first two correspond to Algorithm 3.1, the first (later denoted **A3.1**($\alpha = \mathbf{1}$)) with $\alpha_k^A = \alpha_k^B = 1$ being the original von Neumann [31] approach, while the second (**A3.1**($\alpha = \mathbf{opt}$)) is with $\alpha_k^A = 1$ and α_k^B being chosen to minimize $\phi(\alpha)$ as described in Sect. 3. The second pair of strategies correspond to Algorithm 3.2 with $\lambda = \frac{1}{2}$, the first (**A3.2**($\alpha = \mathbf{1}$)) using the traditional $\alpha_k = 1$, and the second (**A3.2**($\alpha = \mathbf{opt}$)) with α_k chosen to minimize $\psi(\alpha)$.

Table 1 Types of failures for each algorithm tested

Variant	$>10^6$ iterations	>1800 CPU seconds
A3.1 ($\alpha = 1$)	36	15
A3.1 ($\alpha = \text{opt}$)	29	15
A3.2 ($\alpha = 1$)	40	18
A3.2 ($\alpha = \text{opt}$)	32	15

In our tests, we start from the (usually infeasible) point x_0 for which $(x_0)_i = x_i^l - 1$, if x_i^l is finite, otherwise $(x_0)_i = x_i^u + 1$, if x_i^u is finite, and otherwise $(x_0)_i = 0$. The corresponding $c_0 = Ax_0$. Each run is terminated as soon as $\max(\|P_A(x,c) - (x,c)\|_2, \|P_B(x,c) - (x,c)\|_2)$ is smaller than a prescribed stopping tolerance, here 0.000001 In addition, a CPU time limit of half an hour, and an iteration bound of one million iterations is imposed to preclude unacceptably poor performance.

No preprocessing of the problems is performed by default, aside from identifying and removing dependent linear equality constraints. We do this by calling yet another GALAHAD module `FDC`. This factorizes

$$\begin{pmatrix} \alpha I & A^T \\ A & 0 \end{pmatrix} = LBL^T$$

using `MA27`, and assesses rank deficiency from tiny eigenvalues of the one-by-one and two-by-two diagonal blocks that make up B—a value $\alpha = 0.01$ is used by default. We recognize that this is not as robust as, for example, a singular-value decomposition or a rank-revealing QR factorization, but fortunately has proved remarkably reliable in our tests.

All of our experiments were performed on a single processor of a 3.06 GHz Dell Precision 650 Workstation. The codes were compiled using full optimization with the Intel ifort compiler, and the computations were performed in double precision.

We consider the complete set of linear programming Netlib and other linear programming test problems as distributed with CUTEr [17]. Our intention here is simply to evaluate projection methods for a diverse and important set of real-life examples. In particular, since almost all of these problems are known to have feasible points, we aim to find such a point efficiently.

For comparison purposes, we also apply the GALAHAD interior-point package `LSQP` to find the projection of x_0 onto $\mathcal{C}$.

The complete set of results for our four variants are given in Appendix 1 of the on-line attachment to this paper. It is immediately clear that none of these methods is generally reliable or fast. We break down the failures for each variant in Table 1.

By way of comparison, in Appendix 2 of the on-line attachment we give results for `LSQP`, which solves all 120 problems with the exception of the (allegedly) infeasible `BCDOUT` and `MPSBCD03`. To illustrate the relative performance of the five methods, we plot their CPU time performance profiles [12], for the 118 problems solved by at least one of the methods, in Fig. 1. It is immediately clear that, at least for this test set, the interior-point method is vastly superior to all four projection methods. Closer scrutiny of the detailed results in the appendices indicates that even for the subset

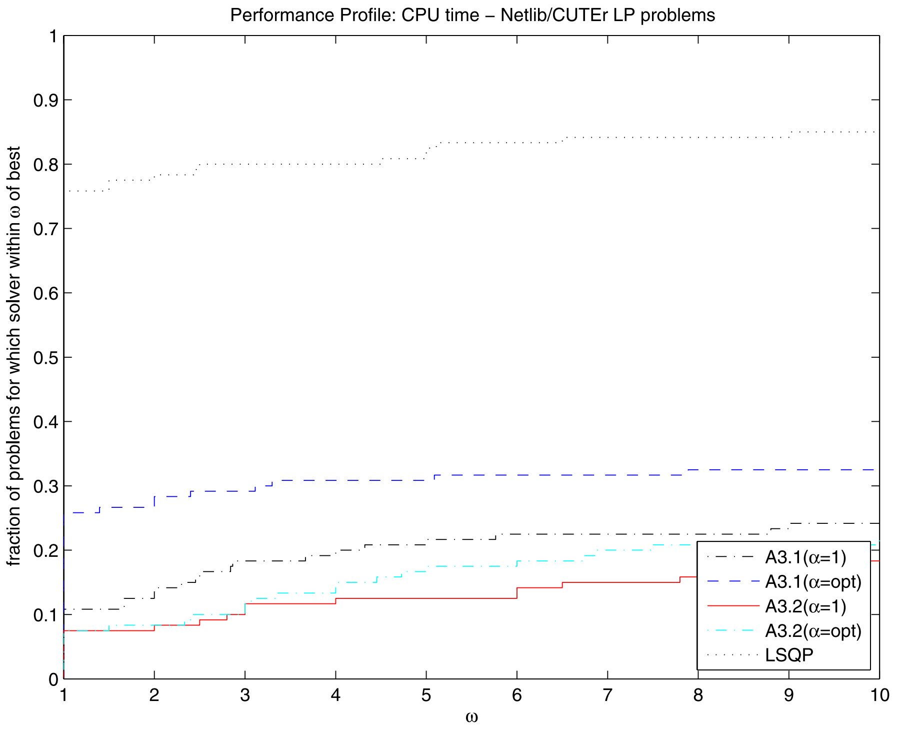

Fig. 1 Performance profile, $p(\omega)$ including `LSQP`: CPU time for the 118 problems under consideration

of problems for which all succeed, the interior-point approach is still a clear overall winner.

We now turn to assessing the projection methods. Since there is little point in comparing the variants on problems for which all fail, we restrict ourselves to the 77 out of the 120 problems that were solved by at least one variant.

In Figs. 2 and 3, we plot the performance profiles for CPU times and iteration counts for the projection methods. These indicate that the successive projection methods are generally superior to the simultaneous ones, and that picking the stepsize to be locally optimal is better than the traditional unit step. Again, examining the tables in Appendix 1 in the on-line attachment in detail, the (often pitifully slow) linear convergence of the methods is apparent.

5 Comments and conclusions

When we started this study, we were under the impression that projection methods would be generally applicable techniques for solving real-life problems. In particular, numerical experience with (small) random problems [4, 5, 7] suggests that modest numbers of iterations are required to achieve reasonable accuracy for methods of the sort we have described here. This is contrary to our numerical experience, and simply suggests to us that there is a significant difference between random and real-life

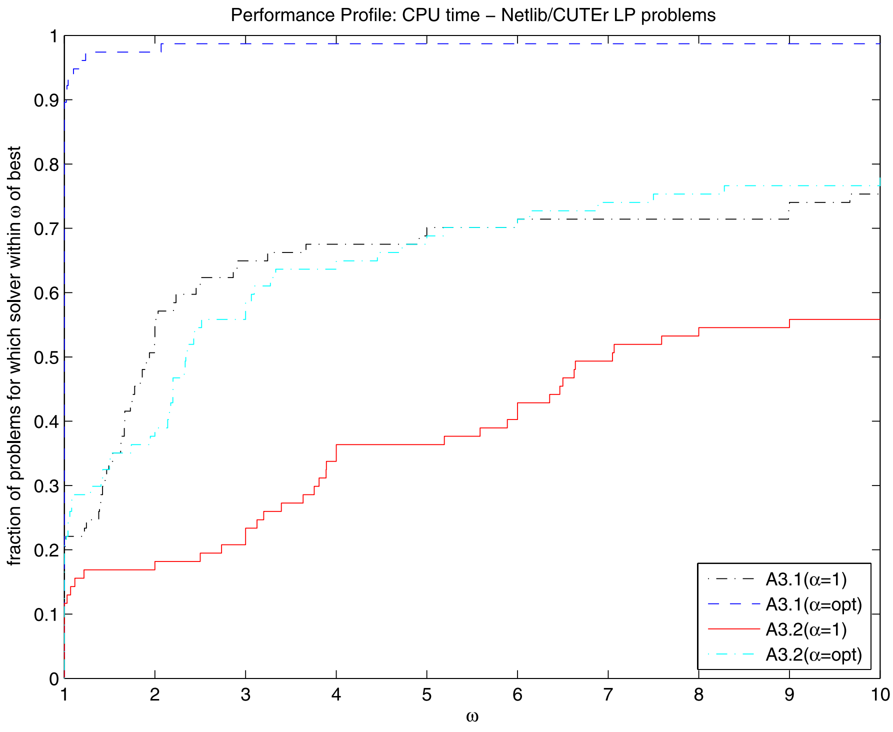

Fig. 2 Performance profile, $p(\omega)$: CPU time for the 77 problems under consideration

problems (similar observations have been made for linear equations, where random problems tend to be well-conditioned [16], and thus often easier to solve than those from applications). The only real hint we found in the literature that these methods might be expensive is for an example from (N-convex) regression [28]. With hindsight, since for linear systems Kaczmarz's method [26] is essentially the Gauss-Seidel iteration, while Cimmino's [10] may be viewed as Jacobi's method for the normal equations, it is perhaps not surprising that these simple projection methods do not perform particularly well; some form of (low-cost) acceleration is undoubtedly and urgently needed.

We do not want to suggest here that successive and simultaneous projection methods are not useful, since in particular they appear to have been applied successfully for many years in medical imaging, radiation therapy planning and signal processing [2, 8, 9]. But our experience suggests that despite the large literature devoted to theoretical analysis, they should not be considered as the method of choice for a given application without further strong empirical evidence to support such a claim.

We also need to be cautious in extrapolating our findings on problems involving two constraint sets to the more general case. Our argument is simply that this is the setting which we had anticipated would put the methods in the best light, and that projecting into a larger number of sets would likely have increase iteration counts (if not costs).

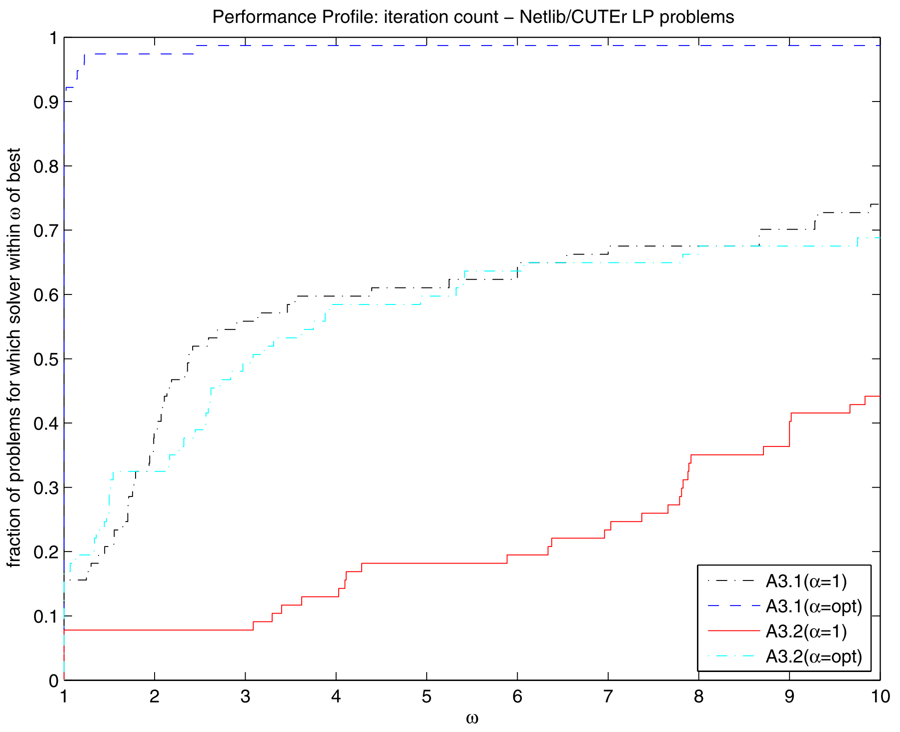

Fig. 3 Performance profile, $p(\omega)$: Iteration counts for the 77 problems under consideration

Of course the observation that successive projection methods tend to require fewer iterations/projections than simultaneous methods is not new [5]; in defense of the latter, it is easier to exploit their natural parallelism, especially when a point in the intersection of a large number of sets is required.

Acknowledgements The author would like to thank Coralia Cartis and Iain Duff for their helpful comments and suggestions on this work.

References

1. Bauschke, H.H., Borwein, J.M.: On the convergence of von Neumann's alternating projection algorithm for two sets. Set-Valued Anal. **1**, 185–212 (1993)
2. Bauschke, H.H., Borwein, J.M.: On projection algorithms for solving convex feasibility problems. SIAM Rev. **38**(3), 367–426 (1996)
3. Bauschke, H.H., Borwein, J.M., Lewis, A.S.: The method of cyclic projections for closed convex sets in Hilbert space. Contemp. Math. **204**, 1–38 (1997)
4. Bauschke, H.H., Combettes, P.L., Kruk, S.G.: Extrapolation algorithm for affine-convex feasibility problems. Numer. Algorithms **41**(3), 239–274 (2006)
5. Bauschke, H.H., Kruk, S.G.: The method of reflection-projection for convex feasibility problems with an obtuse cone. J. Optim. Theory Appl. **120**(3), 503–531 (2004)
6. Censor, Y.: Row-action methods for huge and sparse systems and their applications. SIAM Rev. **23**(4), 444–466 (1981)

7. Censor, Y.: Computational acceleration of projection algorithms for the linear best approximation problem. Linear Algebr. Appl. **416**, 111–123 (2006)
8. Censor, Y., Herman, G.T.: On some optimization techniques in image reconstruction from projections. Appl. Numer. Math. **3**, 365–391 (1987)
9. Censor, Y., Zenios, S.A.: Parallel Optimization: Theory, Algorithms and Applications. Oxford University Press, Oxford (1997)
10. Cimmino, G.: Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. Ric. Sci. XVI Ser. II, Anno IX **1**, 326–333 (1938)
11. Deutsch, F., Hundal, H.: The rate of convergence of Dykstra's cyclic projections algorithm: the polyhedral case. Numer. Funct. Anal. Optim. **15**(5–6), 537–565 (1994)
12. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
13. Duff, I.S.: MA57—a code for the solution of sparse symmetric definite and indefinite systems. ACM Trans. Math. Softw. **30**(2), 118–144 (2004)
14. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear equations. ACM Trans. Math. Softw. **9**(3), 302–325 (1983)
15. Dykstra, R.L.: An algorithm for restricted least squares regression. J. Am. Stat. Assoc. **78**, 837–842 (1983)
16. Edelman, A.: Eigenvalues and condition numbers of random matrices. SIAM J. Matrix Anal. Appl. **9**(4), 543–560 (1988)
17. Gould, N.I.M., Orban, D., Toint, P.L.: CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited. ACM Trans. Math. Softw. **29**(4), 373–394 (2003)
18. Gould, N.I.M., Orban, D., Toint, P.L.: GALAHAD—a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. ACM Trans. Math. Softw. **29**(4), 353–372 (2003)
19. Gubin, L.G., Polyak, B.T., Raik, E.V.: The method of projections for finding the common point of convex sets. U.S.S.R. Comput. Math. Math. Phys. **7**(6), 1–24 (1967)
20. Halperin, I.: The product of projection operators. Acta Sci. Math. **23**, 96–99 (1962)
21. Han, S.P.: A successive projection method. Math. Program. **40**(1), 1–14 (1988)
22. Hildreth, C.: A quadratic programming procedure. Nav. Res. Logist. Q. **4**, 79–85 (1957), Erratum, ibid. p. 361
23. HSL, A collection of Fortran codes for large-scale scientific computation (2004). See http://www.cse.clrc.ac.uk/nag/hsl/
24. Iusem, A.N., De Pierro, A.R.: A simultaneous iterative method for computing projections on polyhedra. SIAM J. Control Optim. **25**(1), 231–243 (1987)
25. Iusem, A.N., De Pierro, A.R.: On the convergence of Han's method for convex-programming with quadratic objective. Math. Program. Ser. B **52**(2), 265–284 (1991)
26. Kaczmarz, S.: Angenährte Auflösung von Systemen linearer Gleichungen. Bull. Int. Acad. Pol. Sci. Let. A **35**, 355–357 (1937)
27. Merzlyakov, Y.I.: On a relaxation method of solving systems of linear inequalities. USSR Comput. Math. Math. Phys. **2**, 504–510 (1962)
28. Perkins, C.: A convergence analysis of Dykstra's algorithm for polyhedral sets. SIAM J. Numer. Anal. **40**(2), 792–804 (2002)
29. Pierra, G.: Decomposition through formalization in a product space. Math. Program. **28**(1), 96–115 (1984)
30. De Pierro, A.R., Iusem, A.N.: A simultaneous projections method for linear inequalities. Linear Algebra Appl. **64**, 243–253 (1985)
31. von Neumann, J.: Functional Operators, vol. II. Princeton University Press, Princeton (1950)
32. Williams, J.W.J.: Algorithm 232, Heapsort. Commun. ACM **7**, 347–348 (1964)