A FAST ALGORITHM FOR MATRIX BALANCING

PHILIP A. KNIGHT[*] AND DANIEL RUIZ[†]

Abstract. As long as a square nonnegative matrix A contains sufficient nonzero elements, then the matrix can be balanced, that is we can find a diagonal scaling of A that is doubly stochastic. A number of algorithms have been proposed to achieve the balancing, the most well known of these being Sinkhorn-Knopp. In this paper we derive new algorithms based on inner-outer iteration schemes. We show that Sinkhorn-Knopp belongs to this family, but other members can converge much more quickly. In particular, we show that while stationary iterative methods offer little or no improvement in many cases, a scheme using a preconditioned conjugate gradient method as the inner iteration can give quadratic convergence at low cost.

Key words. Matrix balancing, Sinkhorn-Knopp algorithm, doubly stochastic matrix, conjugate gradient iteration.

AMS subject classifications. 15A48, 15A51, 65F10, 65H10.

1. Introduction.

For at least 70 years, scientists in a wide variety of disciplines have attempted to transform square nonnegative matrices into doubly stochastic form by applying diagonal scalings. That is, given $A \in \mathbb{R}^{n\times n}$, $A \geq 0$, find diagonal matrices D_1 and D_2 so that $P = D_1AD_2$ is doubly stochastic. Motivations for achieving this balance include interpreting economic data [1], preconditioning sparse matrices [16], understanding traffic circulation [14], assigning seats fairly after elections [3], matching protein samples [4] and ordering nodes in a graph [12]. In all of these applications, one of the main methods considered is SK[1]. This is an iterative process that attempts to find D_1 and D_2 by alternately normalising columns and rows in a sequence of matrices starting with A. Convergence conditions for this algorithm are well known: if A has total support[2] then the algorithm will converge linearly with asymptotic rate equal to the square of the subdominant singular value of P [22, 23, 12].

Clearly, in some cases the convergence will be painfully slow. The principal aim of this paper is to derive some new algorithms for the matrix balancing problem with an eye on speed, especially for large systems. First we look at a simple Newton method for symmetric matrices, closely related to a method proposed by Khachiyan and Kalantari [11] for positive definite (but not necessarily nonnegative) matrices. We will show that as long as Newton's method produces a sequence of positive iterates, the Jacobians we generate will be positive semi-definite and that this is also true when we adapt the method to cope with nonsymmetric matrices.

To apply Newton's method exactly we require a linear system solve at each step, and this is usually prohibitively expensive. We therefore look at iterative techniques for approximating the solution at each step. First we look at splitting methods and we see that SK is a member of this family of methods, as is the algorithm proposed by Livne and Golub in [16]. We give an asymptotic bound on the (linear) rate of convergence of these methods. For symmetric matrices we can get significant improvement

[*]Department of Mathematics, University of Strathclyde, 26 Richmond Street, Glasgow G1 1XH Scotland (`pk@maths.strath.ac.uk`). This work was supported in part by a grant from the Carnegie Trust for the Universities of Scotland.

[†]INPT-ENSEEIHT 2, rue Charles Camichel, Toulouse, France (`Daniel.Ruiz@enseeiht.fr`)

[1]The algorithm has been given many different names, Sinkhorn-Knopp is the name usually adopted by the linear algebra community. We'll refer to it as SK in the rest of this paper

[2]total support is defined in §3.

over SK. Unfortunately, we show this is not true for nonsymmetric matrices.

Next we look at a preconditioned conjugate gradient method for solving the inner iteration. We discuss implementation details and show that asymptotically, super-linear convergence is achievable, even for nonsymmetric matrices. By implementing a box constraint we ensure that our iterates retain positivity and we demonstrate the reliability and speed of our method in tests. The algorithm, the core of which takes up less than forty lines of MATLAB code, is given in an appendix.

To measure the rate of convergence for an iterative process $\mathcal{I}$ we use the root-convergence factor [18, §9.2.1]

$$R(x) = \sup \left\{ \limsup_{k\to\infty} \|x_k - x\|^{1/k} | \{x_k\} \in C \right\},$$

where C is the set of all sequences generated by $\mathcal{I}$ which converge to x.

A number of authors have presented alternative techniques for balancing matrices that can converge faster than SK. For example, Parlett and Landis [19] look at some simple ways of trying to accelerate the convergence by focusing on reducing statistics such as the standard deviation between row sums of the iterates. In certain cases, they show great improvement is possible, suggesting that the rate on convergence is not dependent on the singular values of P; but they also give examples where their alternatives perform significantly worse. We include a comparison of one of these algorithms against our proposed approach in §6. Linial et al. [15] use a similar approach to [19] in the context of estimating matrix permanents, although their upper bound on iteration counts ($O(n^7)$) makes them distinctly unappealing for large problems!

A completely different approach is to view matrix balancing as an optimisation problem. There are many alternative formulations, perhaps the first being that of Marshall and Olkhin [17] who show that balancing is equivalent to minimizing the bilinear form x^TAy subject to the constraints $\Pi x_i = \Pi y_i = 1$. However, the experimental results we have seen for optimisation techniques for balancing [17, 21, 2] suggest that these methods are not particularly cheap to implement.

2. Newton's method.

Let $\mathcal{D} : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ represent the operator that turns a vector into a diagonal matrix, $\mathcal{D}(x) = \text{diag}(x)$, and let e represent a vector of ones. Then to balance a nonnegative matrix, A, we need to find positive vectors r and c such that

$$\mathcal{D}(r)A\mathcal{D}(c)e = \mathcal{D}(r)Ac = e, \ \ \mathcal{D}(c)A^Tr = e. \tag{2.1}$$

Rearranging these identities gives

$$c = \mathcal{D}(A^Tr)^{-1}e, \quad r = \mathcal{D}(Ac)^{-1}e,$$

and one way of writing SK [9, 12] is as

$$c_{k+1} = \mathcal{D}(A^Tr_k)^{-1}e, \quad r_{k+1} = \mathcal{D}(Ac_{k+1})^{-1}e. \tag{2.2}$$

If A is symmetric then (2.1) can be simplified. To achieve balance we need a vector x_* that satisfies

$$f(x_*) = \mathcal{D}(x_*)Ax_* - e = 0. \tag{2.3}$$

This leads to the iterative step

$$x_{k+1} = \mathcal{D}(Ax_k)^{-1}e. \tag{2.4}$$

Exactly the same calculations are performed as for (2.2), one simply extracts r_k and c_k from alternate iterates [12]. If A is nonsymmetric, one can use (2.4) on

$$S = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}.$$

An obvious alternative to SK is to solve (2.3) with Newton's method. Differentiating $f(x)$ gives

$$J(x) = \frac{\partial}{\partial x}(\mathcal{D}(x)Ax - e) = \mathcal{D}(x)A + \mathcal{D}(Ax),$$

a result that is easily confirmed by componentwise calculation or with tensor algebra, hence Newton's method can be written

$$x_{k+1} = x_k - (\mathcal{D}(x_k)A + \mathcal{D}(Ax_k))^{-1}(\mathcal{D}(x_k)Ax_k - e).$$

We can rearrange this equation to get

$$(\mathcal{D}(x_k)A + \mathcal{D}(Ax_k))x_{k+1} = \mathcal{D}(Ax_k)x_k + e,$$

so,

$$\begin{aligned}(A + \mathcal{D}(x_k)^{-1}\mathcal{D}(Ax_k))x_{k+1} &= \mathcal{D}(x_k)^{-1}(\mathcal{D}(Ax_k)x_k + e) \\ &= Ax_k + \mathcal{D}(x_k)^{-1}e,\end{aligned} \tag{2.5}$$

and we can set up each Newton iteration by performing some simple vector operations and updating the diagonal on the left hand side. This matrix plays a key role in our analysis and we introduce the notation

$$\mathcal{A}_k = A + \mathcal{D}(x_k)^{-1}\mathcal{D}(Ax_k).$$

Note that the matrix on the lefthand side of (2.5) inherits the symmetry of A. This algorithm can be implemented by applying one's linear solver of choice. In practical applications, it makes sense to apply an inner-outer iteration scheme. In §4 and §5 we look at some efficient ways of doing this. In particular, we look at how to deal with the nonsymmetric case. Here, we can use the same trick we used to derive (2.4) but, as we'll see, the resulting linear systems are singular.

The idea of using Newton's method to solve the scaling problem is not new and was first proposed by Khachiyan and Kalantari in [11]. Instead of (2.3), the equivalent equation

$$Ax - \mathcal{D}(x)^{-1}e = 0 \tag{2.6}$$

is considered. The authors were interested in the problem of scaling symmetric positive definite (SPD) matrices, rather than nonnegative matrices. In this case, the scaled matrix need not be doubly stochastic. While its row and column sums are 1, it may contain negative entries. The authors did not consider the practical application of their algorithm, although the methods we look at in this paper can be adapted to work for their formulation. In §5 we explain why our approach leads to faster convergence.

Newton's method is also used as a method for solving the balancing problem for symmetric matrices in [16]. Here the authors work with

$$\left(I - \frac{ee^T}{n}\right) \mathcal{D}(Ax)x = 0 \tag{2.7}$$

instead of (2.3). They then use a Gauss-Seidel Newton method to solve the problem and show that this approach can give significant improvements over SK. We will develop this idea in §4.

Yet another formulation of (2.4) can be found in [7], where the author suggests the resulting equation is solved by Newton's method. However no attempt is made to implement the suggested algorithm and the fact that only the righthand side changes in the linear system that is solved at each step suggests that rapid convergence is unlikely in general.

3. Properties of $\mathcal{A}_k$. In order that we can solve the balancing problem efficiently, in particular when A is large and sparse, we will use iterative methods to approximately solve the linear system in (2.5). There are a number of possibilities to choose as $\mathcal{A}_k$, the matrix on the lefthand side of the expression, is symmetric positive semidefinite. This is a consequence of the following result (which doesn't require symmetry).

THEOREM 3.1. *Suppose that $A \in \mathbb{R}^{n\times n}$ is nonnegative and $y \in \mathbb{R}^n$ is positive. Let $D = \mathcal{D}(Ay)\mathcal{D}(y)^{-1}$. Then for all $\lambda \in \sigma(A+D)$,* $\mathrm{Re}(\lambda) \geq 0$.

Proof. Note that $A + D$ is similar to

$$\begin{aligned} \mathcal{D}(y)^{-1}(A+D)\mathcal{D}(y) &= \mathcal{D}(y)^{-1}A\mathcal{D}(y) + \mathcal{D}(y)^{-1}\mathcal{D}(Ay) \\ &= \mathcal{D}(y)^{-1}A\mathcal{D}(y) + \mathcal{D}(\mathcal{D}(y)^{-1}A\mathcal{D}(y)e) \\ &= B + \mathcal{D}(Be), \end{aligned}$$

where $B = \mathcal{D}(y)^{-1}A\mathcal{D}(y)$. Now Be is simply the vector of row sums of B and so adding this to the diagonal of B gives us a diagonally dominant matrix. Since the diagonal is nonnegative, the result follows from Gershgorin's theorem. □

If A has a nonzero entry on its diagonal then at least one of the rows of $B + \mathcal{D}(Be)$ will be strongly diagonally dominant and, by a theorem of Taussky [24], it will be nonsingular. We can also ensure that $\mathcal{A}_k$ is nonsingular by imposing conditions on the connectivity between the nonzeros in A. We can establish the following result.

THEOREM 3.2. *Suppose that $A \in \mathbb{R}^{n\times n}$ is nonnegative. If A is fully indecomposable then $\mathcal{A}_k$ is nonsingular.*

Recall that a matrix A is fully indecomposable if it is impossible to find permutation matrices P and Q such that

$$PAQ = \begin{bmatrix} A_1 & 0 \\ A_2 & A_3 \end{bmatrix}$$

with A_1 and A_2 square (a generalisation of irreducibility).

The proof of Theorem 3.2 is postponed to the end of the section. However in many cases we will not be able to satisfy its conditions. For example, if A is nonsymmetric and we use Newton's method to balance

$$S = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix},$$

then S is not fully indecomposable. In fact it is straightforward to show that in this case the linear systems we have to solve are singular.

We'd also like to use our new algorithms on any nonnegative matrix for which balancing is possible, namely any matrix which has total support ($A \geq 0$ has total support if $A \neq 0$ and all the nonzero elements lie on a positive diagonal). Matrices that have total support but that aren't fully indecomposable also lead to singular systems in (2.5).

Singularity in these cases is not problematic as the systems are consistent. As Theorem 3.5 shows, we can go further, and we can solve the systems whenever A has support ($A \geq 0$ has support if it has a positive diagonal).

We need some preliminary results.

LEMMA 3.3. *Suppose that $A \in \mathbb{R}^{n\times n}$ is a symmetric nonnegative matrix with support. Then there is a permutation P such that PAP^T is a block diagonal matrix in which all the diagonal blocks are irreducible.*

Proof. We show this by induction on n. Clearly it is true if $n = 1$. Suppose $n > 1$ and our hypothesis is true for all matrices of dimension smaller than n. If A is irreducible there is nothing to prove, otherwise we can find a permutation Q such that

$$QAQ^T = \begin{bmatrix} A_1 & C \\ 0 & A_2 \end{bmatrix} = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix},$$

where $C = 0$ by symmetry. Since A has support so does QAQ^T and hence A_1 and A_2 must each have support, too and we can apply our inductive hypothesis. □

LEMMA 3.4. *Suppose that $A \geq 0$ has support and let $B = A + \mathcal{D}(Ae)$. If either A is symmetric or it is irreducible then the null space of B is orthogonal to e and the null space has a basis whose elements can be permuted into the form*

$$\begin{bmatrix} e \\ -e \\ 0 \end{bmatrix}.$$

(In the case that the null space has dimension one, the zero component can be omitted).

Proof. If B is nonsingular we have nothing to prove, so we assume it is singular (and hence by Taussky's theorem [24], A has an empty main diagonal).

From Theorem 2.3 and Remark 2.9 in [13] we can infer that if B is irreducible then the null space is one dimensional and, since it is weakly diagonally dominant, all components are of equal modulus. All that remains is to show that there are an equal number of positive and negative components in elements from the null space and hence they can be permuted into the required form.

We choose a permutation $i_1, i_2, \dots, i_n$ of $1, \dots, n$ such that for $1 \leq j \leq n$, $b_{i_j i_{j+1}} \neq 0$, where $i_{n+1} = i_1$. Such a permutation exists because A has support and has an empty main diagonal. Now suppose that x is in the null space of B. Since B is diagonally dominant, the sign of x_i must be opposite to that of x_j whenever $b_{ij} \neq 0$ ($i \neq j$). By construction, $x_{i_j} = -x_{i_{j+1}}$ for $1 \leq j \leq n$. This is only possible if n is even, in which case $x^T e = 0$, as required.

If B is not irreducible but is symmetric then (by the previous lemma) $PBP^T = \text{diag}(B_1, B_2, \dots, B_k)$ where the B_i are irreducible. Since the null space of B is formed from a direct sum of the null space of the diagonal blocks, it too must be orthogonal to e and a basis of the requisite form clearly exists. □

THEOREM 3.5. *Suppose that $A \in \mathbb{R}^{n\times n}$ is a symmetric nonnegative matrix with support and that $y > 0$. The system*

$$(A + \mathcal{D}(Ay)\mathcal{D}(y)^{-1})z = Ay + \mathcal{D}(y)^{-1}e$$

is consistent.

Proof. Let $A_y = \mathcal{D}(y)A\mathcal{D}(y)$. Since $(A_y + \mathcal{D}(A_y e))$ is symmetric, elements orthogonal to its null space must lie in its range, so as a consequence of the previous lemma, we can find a vector d such that

$$(A_y + \mathcal{D}(A_y e))d = e.$$

Let $z = y/2 + \mathcal{D}(y)d$. Then

$$\begin{aligned}(A + \mathcal{D}(y)^{-1}\mathcal{D}(Ay))z &= \frac{1}{2}(A + \mathcal{D}(Ay)\mathcal{D}(y)^{-1})y \\ &\qquad +\mathcal{D}(y)^{-1}(\mathcal{D}(y)A\mathcal{D}(y)d + \mathcal{D}(Ay)\mathcal{D}(y)d) \\ &= Ay + \mathcal{D}(y)^{-1}(A_y + \mathcal{D}(A_y e))d \\ &= Ay + \mathcal{D}(y)^{-1}e.\end{aligned}$$

□

While we have proved that Newton's step will converge if $x_k > 0$, we have not shown that $x_{k+1} > 0$. In fact, it needn't be, and this will be an important consideration for us in developing balancing methods in the later sections.

To finish the section, we prove Theorem 3.2.

Proof. Let $B = \mathcal{D}(x_k)^{-1}A\mathcal{D}(x_k)$ and consider $B + \mathcal{D}(Be)$. Suppose that this matrix is singular and v lies in its null space. We can apply Lemma 3.4, and we know that exactly half of its components are positive and half are negative. Suppose that P permutes v so that the first half of the entries of Pv are positive. Then

$$P(B + \mathcal{D}(Be))P^T = \left[\begin{array}{cc} D_1 & B_1 \\ B_2 & D_2 \end{array}\right],$$

where D_1 and D_2 are diagonal: otherwise, diagonal dominance would force some of the entries of $(B + \mathcal{D}(Be))v$ to be nonzero. Hence

$$PBP^T = \left[\begin{array}{cc} 0 & B_1 \\ B_2 & 0 \end{array}\right].$$

But such a matrix is not fully indecomposable, contradicting our hypothesis. □

4. Stationary iterative methods.

If we only want to solve (2.5) approximately, the simplest class of methods to consider is that of stationary iterative methods, in particular splitting methods. As $\mathcal{A}_k$ is symmetric positive semi-definite we know that many of the common splitting methods will converge.

This approach is used in [16] to solve the formulation of the balancing problem given in (2.7). Here, the authors use Gauss-Seidel to solve the Newton step and demonstrate that the rate of convergence is frequently faster than that of SK. Suppose that the symmetric matrix A can be balanced so that $P = DAD$ is doubly stochastic. Following the standard analysis for splitting methods, they give a bound on the rate of convergence in the neighbourhood of the solution, relating it to the modulus of the second largest eigenvalue of $L^{-1}U$, where L is the lower triangular part and U the strictly upper-triangular part of $P + I - 2ee^T/n$. Note that the bound is not in terms of $\rho(L^{-1}U)$ as the matrix in the lefthand side of (2.7) is singular.

In our main result in this section we show that the rate of convergence $R(x_*)$ can be ascertained for a much wider class of splitting methods. We use (2.5) rather than (2.7) as it leads to particularly simple representations of the methods.

Suppose that $\mathcal{A}_k = M_k - N_k$ where M_k is nonsingular. We can then attempt to solve the balancing problem with an inner-outer iteration where the outer iteration is given by the Newton step and the inner iteration by the splitting method

$$M_k y_{j+1} = N_k y_j + Ax_k + \mathcal{D}(x_k)^{-1}e,$$

where $y_0 = x_k$. However, we can update M_k extremely cheaply at any point—we only need to update its diagonal and the vectors we need for this are available to us at each step—so it seems sensible to run only one step of the inner iteration. In this case, our splitting method for (2.5) becomes

$$\begin{aligned} Mx_{k+1} &= N_k x_k + Ax_k + \mathcal{D}(x_k)^{-1}e \\ &= (M_k - \mathcal{D}(x_k)^{-1}\mathcal{D}(Ax_k))x_k + \mathcal{D}(x_k)^{-1}e \\ &= M_k x_k - \mathcal{D}(Ax_k)e + \mathcal{D}(x_k)^{-1}e \\ &= M_k x_k - Ax_k + \mathcal{D}(x_k)^{-1}e. \end{aligned}$$

In other words, our iteration can be written

$$x_{k+1} = x_k + M_k^{-1}(\mathcal{D}(x_k)^{-1}e - Ax_k). \tag{4.1}$$

Note that if we choose $M_k = \mathcal{D}(x_k)^{-1}\mathcal{D}(Ax_k)$ in (4.1), then the resulting iteration is

$$\begin{aligned} x_{k+1} &= x_k + \mathcal{D}(x_k)\mathcal{D}(Ax_k)^{-1}(\mathcal{D}(x_k)^{-1}e - Ax_k) \\ &= x_k + \mathcal{D}(Ax_k)^{-1}e - \mathcal{D}(x_k)e \\ &= \mathcal{D}(Ax_k)^{-1}e, \end{aligned}$$

and we recover (2.4) hence SK can be neatly classified amongst our splitting methods.

To prove a general result we use Theorem 10.3.1 from [18] which in our notation can be stated as follows.

THEOREM 4.1. *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be Gateaux differentiable in an open neighbourhood S_0 of a point x_* at which f' is continuous and $f(x_*) = 0$.*

Suppose that $f'(x) = M(x) - N(x)$ where $M(x_*)$ is continuous and nonsingular at x_* and that $\rho(G(x_*)) < 1$ where $G(x) = M(x)^{-1}N(x)$. Then for $m \geq 1$ the iterative process $\mathcal{I}$ defined by

$$x_{k+1} = x_k - \left(\sum_{j=1}^{m} G(x_k)^{j-1}\right) M(x_k)^{-1} f(x_k), \quad k = 0, 1, \ldots, \tag{4.2}$$

has attraction point x_* and $R(x_*) = \rho(G(x_*)^m)$.

COROLLARY 4.2. *Suppose that $A \in \mathbb{R}^{n\times n}$ is a symmetric nonnegative matrix with support. Let $x_* > 0$ be a vector such that $P = \mathcal{D}(x_*)A\mathcal{D}(x_*)$ is stochastic. Then for the iterative process defined by (4.1) $R(x_*) = \rho(M^{-1}N)$ where $M - N$ is the splitting applied to $A + \mathcal{D}(x_*)^{-2}$.*

Proof. The existence of x_* is guaranteed [22]. Let $m = 1$, $f(x) = \mathcal{D}(x)Ax - e$ and let $M(x_k) = \mathcal{D}(x_k)M_k$. Then (4.1) can be rewritten as (4.2) and

$$G(x_k) = M(x_k)^{-1}N(x_k) = M_k^{-1}N_k.$$

So in the limit for a sequence $\{x_k\}$ converging to x_*, $R(x)$ is governed by the spectral radius of the splitting of

$$\lim_{k\to\infty} \mathcal{A}_k = A + \mathcal{D}(x_*)^{-1}\mathcal{D}(Ax_*) = A + \mathcal{D}(x_*)^{-2}.$$

□

For many familiar splittings, $R(x_*)$ has an alternative representation as we can often show that $\rho(M^{-1}N)$ is invariant under the action of diagonal scaling on A. If this is the case we can replace $A + \mathcal{D}(x_*)^{-2}$ with

$$\mathcal{D}(x_*)(A + \mathcal{D}(x_*)^{-2})\mathcal{D}(x_*) = P + I,$$

that is the asymptotic rate of convergence can be measured by looking at the spectral radius of the splitting matrix for $P + I$. Many familiar splitting methods exhibit this scaling invariance. For example, the Jacobi method, Gauss-Seidel and successive over relaxation are covered by the following result, where $X \circ Y$ represents the Hadamard product of two matrices.[3]

THEOREM 4.3. *Suppose that $A, H \in \mathbb{R}^{n\times n}$ and D is a nonsingular diagonal matrix. Let $M = A \circ H$ and $N = M - A$. Then the spectrum of $M^{-1}N$ is unchanged if we replace A with DAD.*

Proof. Notice that $N = A \circ G$ where $G = ee^T - H$ and so

$$(DAD \circ H)^{-1}(DAD \circ G) = (D(A \circ H)D)^{-1}D(A \circ G)D = D^{-1}(A \circ H)^{-1}(A \circ G)D.$$

□

In particular, in the case of Gauss-Seidel, we get a bound comparable with the formulation adopted in [16] mentioned at the start of this section. $R(x_*)$ is usually a little smaller for Livne and Golub's method but for sparse matrices the difference is frequently negligible.

For SK, though, we get $R(x_*) = \rho(P) = 1$. This is because the formulation described in (2.4) oscillates rather than converges. This phenomenon is discussed in [12], where it is shown that $R(x_*)$ is equal to the modulus of the subdominant eigenvalue of P.

If A is symmetric, then it is possible to significantly improve convergence speed over SK with an appropriate choice of M. However, things are different if A is nonsymmetric. For example, consider the effect of using Gauss-Seidel in (4.1) where A is replaced by

$$S = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$$

and $x_k = \begin{bmatrix} r_k^T & c_k^T \end{bmatrix}^T$. We have,

$$\begin{bmatrix} r_{k+1} \\ c_{k+1} \end{bmatrix} = \begin{bmatrix} r_k \\ c_k \end{bmatrix} + \begin{bmatrix} \mathcal{D}(Ac_k)\mathcal{D}(r_k)^{-1} & 0 \\ A^T & \mathcal{D}(A^Tr_k)\mathcal{D}(c_k)^{-1} \end{bmatrix}^{-1} \begin{bmatrix} \mathcal{D}(r_k)^{-1}e - Ac_k \\ \mathcal{D}(c_k)^{-1}e - A^Tr_k \end{bmatrix}.$$

[3]Similarity is not restricted to the splittings covered by this theorem, for example we can show that the result also holds for SSOR.

After some straightforward manipulation, this becomes

$$\begin{bmatrix} r_{k+1} \\ c_{k+1} \end{bmatrix} = \begin{bmatrix} \mathcal{D}(Ac_k)^{-1}e \\ \mathcal{D}(A^T r_k)^{-1}e + c_k - \mathcal{D}(c_k)\mathcal{D}(A^T r_k)^{-1}A^T r_{k+1} \end{bmatrix}. \tag{4.3}$$

In the spirit of Gauss-Seidel, we can replace r_k in the righthand side of (4.3) with r_{k+1}, giving

$$\begin{bmatrix} r_{k+1} \\ c_{k+1} \end{bmatrix} = \begin{bmatrix} \mathcal{D}(Ac_k)^{-1}e \\ \mathcal{D}(A^T r_{k+1})^{-1}e \end{bmatrix},$$

and this is precisely (2.2), SK. In other words the Gauss-Seidel Newton method offers no improvement over SK for nonsymmetric matrices.

There are any number of choices for M in (4.1), but in tests we have not been able to gain a consistent and significant improvement over SK when A is nonsymmetric. As most of the applications for balancing involve nonsymmetric matrices we need a different approach.

5. Conjugate gradient method.

Recall that if A is symmetric and nonnegative the Newton step (2.5) can be written

$$\mathcal{A}_k x_{k+1} = Ax_k + \mathcal{D}(x_k)^{-1}e,$$

where $\mathcal{A}_k = A + \mathcal{D}(x_k)^{-1}\mathcal{D}(Ax_k)$. By Theorems 3.1 and 3.5, we know that $\mathcal{A}_k$ is positive semi-definite and (2.5) is consistent whenever $x_k > 0$ and we can solve the Newton step with the conjugate gradient method [8]. Essentially, all we need to do is to find an approximate solution to (2.5) and iterate, ensuring that we never let components of our iterates become negative. We now look in more detail at how we implement the method efficiently.

First, motivated by the proof of Theorem 3.1, we note that we can apply a diagonal scaling to (2.5) to give a symmetric diagonally dominant system. Premultiplying each side of the equation by $\mathcal{D}(x_k)$ and writing $y_{k+1} = \mathcal{D}(x_k)^{-1}x_{k+1}$ we get

$$(B_k + \mathcal{D}(B_k e))y_{k+1} = (B_k + I)e, \tag{5.1}$$

where $B_k = \mathcal{D}(x_k)A\mathcal{D}(x_k)$. We needn't form B_k explicitly, as all our calculations can be performed with A. The natural choice as initial iterate for every inner iteration[4] is e, for which the initial residual is

$$r = (B_k + I)e - (B_k + \mathcal{D}(B_k e))e = e - B_k e = e - v_k,$$

where $v_k = x_k \circ Ax_k$. Inside the conjugate gradient iteration we need to perform a matrix-vector product involving $B_k + \mathcal{D}(B_k e)$. For a given vector p, we can perform this efficiently using the identity

$$(B_k + \mathcal{D}(B_k e))p = x_k \circ (A(x_k \circ p)) + v_k \circ p.$$

Since the rest of the conjugate gradient algorithm can be implemented in a standard way, this matrix-vector product is the dominant factor in the cost of the algorithm as a whole.

[4]We also use e as the default choice to start the algorithm.

As a stopping criterion for the scaling algorithm we use the residual measure

$$\|e - \mathcal{D}(x_k)Ax_k\|_2 = \|e - v_k\|_2.$$

In contrast to our experience with stationary iterative methods, it pays to run the inner iteration for a number of steps, and so we need a stopping criterion for the inner iteration, too. As is standard with inexact Newton methods, we don't need a precise solution while we are a long way from the solution and we adopt the approach outlined in [10, Chap. 3], using the parameters η_{max} to limit the size of the forcing term η and γ to scale the forcing term to the residual measure. In our algorithm the default values are $\eta_{max} = 0.1$, $\gamma = 0.9$.

While we know that $B_k + \mathcal{D}(B_k e)$ is SPD if $x_k > 0$, we cannot guarantee that (5.1) will have a positive solution. Furthermore, we do not know that our Newton method will converge if our initial guess is a long way from the solution. We therefore apply a box constraint inside the inner iteration. We introduce a parameter δ which determines how much closer to the edge of the positive cone we are willing to let our current iterate move and a second parameter Δ bounding how far we can move away. We don't want components of our scaling to grow too large as this generally forces another component to shrink towards 0. By rewriting (2.5) in the form (5.1) we ensure that all coordinate directions are treated equally. Before we move along a search direction in the conjugate gradient algorithm, we check whether this would move our iterate outside our box. If it does, we can either reject the step or only move part of the way along it and then restart the inner iteration. In our experience, it pays not to reject the step completely and instead we move along it until the minimum element of y_{k+1} equals δ or the maximum equals Δ. In general the choices $\delta = 0.1$ and $\Delta = 3$ seem to work well.

We report the results of our tests of the method in the next section. As is to be expected, the fact that the method converges independently of the spectrum of our target matrix means that the conjugate-gradient method has the potential to converge much more quickly than the other methods we have discussed. Our experience is that it is no only quick but it is robust, too.

While stationary iterative methods couldn't improve on SK for nonsymmetric systems the picture changes for the conjugate gradient method. If A is nonsymmetric we can apply the symmetric algorithm to

$$S = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}.$$

The resulting method is no longer connected to SK. From the results in §3 we know that the Jacobian will be singular at each step, but that the linear systems we solve will be consistent. Although B_k is singular the fact that it is semidefinite and the system is consistent means CG will work prvided the initial residual is in the othogonal complement of the kernel of S. We ensure this is the case by setting $y_0 = e$. Our tests show our method is robust in these cases, although the residual may not decrease monotonically, suggesting that the algorithm may be moving between local minima of the function $\|\mathcal{D}(x)Sx - e\|^2$.

While the use of the augmented matrix S may seem inefficient, it is seemingly inevitable that we will have to work with both A and A^T in the algorithm in order to balance rows and columns simultaneously. Rather than working with S, we can unravel the method and work directly with A and A^T. This is particularly useful in solving the large scale balancing problems discussed in [12], where a rank one correction is applied implicitly to A.

As we mentioned early, Khachiyan and Kalantari did not discuss the practical application of their algorithm in [11]. However it too can be implemented using the conjugate gradient method. For this formulation, (5.1) is replaced with

$$(B_k + I)y_{k+1} = 2e.$$

While this is also straightforward to implement, we can no longer guarantee that the systems will be semi-definite unless A is, too. In experiments, we have found that the Khachiyan-Kalantari approach is significantly slower than the one we propose.

6. Results. We now compared the performance of the conjugate gradient based approach against a number of other algorithms. The algorithms considered are as follows. BNEWT is our implementation of an inexact Newton iteration with conjugate gradients; SK is the Sinkhorn-Knopp algorithm and EQ is the "equalize" algorithm from [19]; GS is a Gauss-Seidel implementation of (4.1). We use this in preference to the method outlined in [16], as for large matrices it is much easier to implement in MATLAB. In terms of rate of convergence, the algorithms are similar.

We have tested the algorithm against both symmetric and nonsymmetric matrices. All our tests were carried out in MATLAB. We measure the cost in terms of the number of matrix-vector products taken to achieve the desired level of convergence. In our tests on nonsymmetric matrices, we have counted the number of products involving A or A^T (double the number for the symmetrised matrix S). We ran our algorithms until an appropriate residual norm fell below a tolerance of 10^{-6}. For BNEWT, and other algorithms for symmetric matrices, we measured $\|\mathcal{D}(x_k)Ax_k - e\|_2$ (replacing A with S if A was nonsymmetric). For SK we measured $\|\mathcal{D}(c_k)A^T r_k - e\|_2$, where c_k and r_k are defined in (2.2). In all cases, our initial iterate is a vector of ones.

In BNEWT there are a number of parameters (η_{max}, γ, δ and Δ) we can tune to improve performance connected to the convergence criterion of the inner step and the line search. Unless otherwise stated, we have used the default values for these parameters.

Our first batch of test matrices were used by Parlett and Landis [19] to compare their balancing algorithms to SK. These are all 10×10 upper Hessenberg matrices defined as follows. $H = (h_{ij})$ where

$$h_{ij} = \begin{cases} 0, & j < i - 1 \\ 1, & \text{otherwise,} \end{cases}$$

H_2 differs from H in only that h_{12} is replaced by 100 and $H_3 = H + 99I$. Our results are given in Table 6.1. In this experiment, the tolerance changed to 10^{-5} as this was the choice in [19]. The algorithm of Parlett and Landis, EQ, uses mainly vector-vector operations, and we have converted this to an (approximately) equivalent number of matrix-vector products.

We see the consistent performance of BNEWT, outperforming the other choices. The results for GS confirm our analysis in § 4, showing it is virtually identical to SK. The more fine-grained nature of EQ means that a comparison with other algorithms in terms of operation counts is misleading. In our tests it took a similar number of iterations as SK to converge, and significantly longer time.

We next tested BNEWT on the $n \times n$ version of H_3. For large values of n, this becomes very challenging to balance as the ratio between the smallest and largest

	BNEWT	SK	EQ	GS
H	76	110	141	114
H_2	90	144	131	150
H_3	94	2008	237	2012

TABLE 6.1
Rate of convergence for Parlett and Landis test matrices.

elements of the balancing factors grows extremely large (the matrix becomes very close in a relative sense to a matrix without total support). The cost of convergence is given in Table 6.2, along with the ratio $r_n = (\max_i x_i)/(\min_i x_i)$.

	$n = 10$	25	50	100
BNEWT	124	300	660	1792
SK	3070	16258	61458	235478
r_n	217	7×10^6	2×10^{14}	2×10^{29}

TABLE 6.2
Rate of convergence for H_n.

BNEWT still copes in extremely trying circumstances, although the convergence of the residual is far from monotonic (we illustrate the progress after each inexact Newton step for $n = 50$ in Figure 6.1).

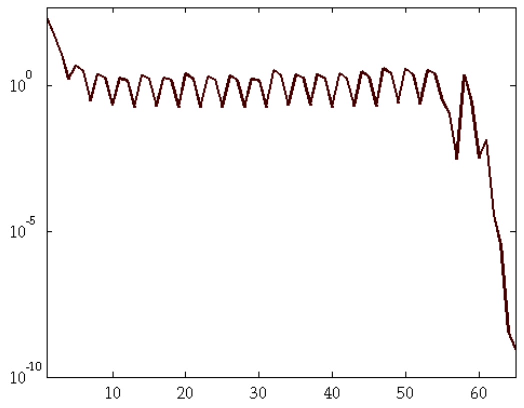

FIG. 6.1. *Convergence graph of BNEWT for H_{50}.*

The oscillatory behaviour is undesirable, and it can be ameliorated somewhat by varying the parameters in BNEWT. This is illustrated in Figure 6.2. In terms of cost, the choice $\eta_{\max} = 10^{-2}$, $\delta = 0.25$ proved best in reducing the matrix-vector product count to 568.

In our next test we look at three matrices from the Harwell-Boeing collection [5], namely GRE185, GRE343 and GRE1107 (the number representing the dimension). The smallest of these was considered in [16] as a candidate for preconditioning, but convergence was slow. We illustrate the progress of BNEWT in Figure 6.3. The cost in matrix vector products (in ascending order of n) was 198, 112 and 320. Again,

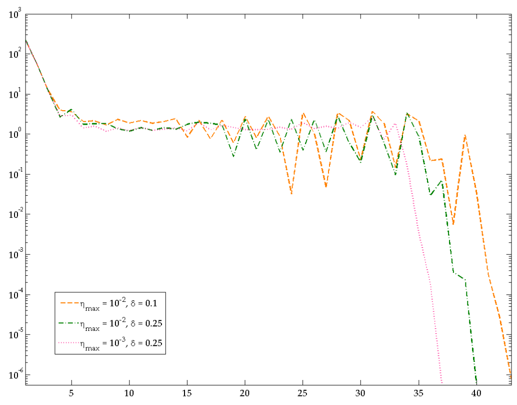

FIG. 6.2. *Smoothing convergence of BNEWT for H_{50}.*

the method proves robust and we avoid the oscillatory behaviour we saw for our previous more extreme example.

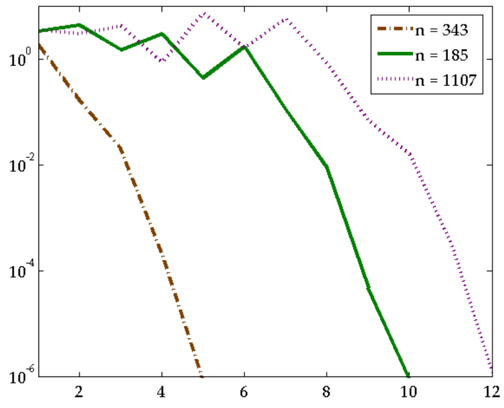

FIG. 6.3. *Convergence of BNEWT for sparse nonsymmetric matrices.*

We have also carried out more comprehensive tests on a selection of matrices from the University of Florida Sparse Matrix Collection (UFL)[5]. First we attempted to scale 45 sparse symmetric matrices with dimensions between 5,000 and 50,000 from the matrix sets `Schenk_IBMNA` and `GHS_indef` which have been observed to be resistant to scaling algorithms [20]. Each of the algorithms BNEWT, GS and SK was run until our default tolerance was reached or 2,000 matrix-vector products had been computed. In all but one case, BNEWT converged within the limit while GS met the convergence criterion in 25 cases, and SK in only 3. In Figure 6.4 we plot the number of matrix-vector products required by BNEWT (along the x-axis) against the number required by GS. Cases where the algorithms failed to converge in time

[5]Web address `http://www.cise.ufl.edu/research/sparse/matrices`

lie at the limits of the axes. The three cases where SK converged are circled (in each case requiring significantly more work than the other algorithms). A pattern emerges where GS converges very quickly on "easy" examples but BNEWT is far more robust and almost always converges at reasonable speed.

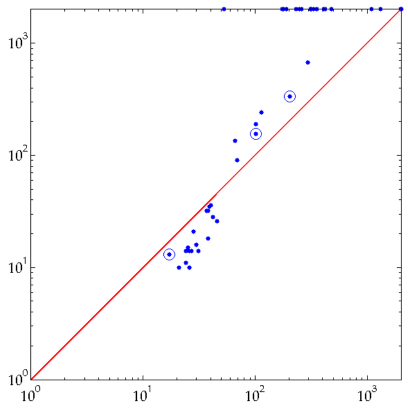

FIG. 6.4. *Comparison of algorithms on sparse symmetric matrices.*

Similar tests were carried out on 60 representative nonsymmetric matrices from the UFL collection with dimensions between 50 and 50,000 to compare BNEWT and SK. The results are shown in Figure 6.5. In this case, the algorithms were given up to 50,000 matrix-vector products to converge. We see the clear superiority of BNEWT in these examples, regularly at least 10 times as fast as SK, although on occasions convergence wasn't particularly rapid, particularly when $(\max_i x_i)/(\min_i x_i)$ is large. SK failed to converge in over half the cases.

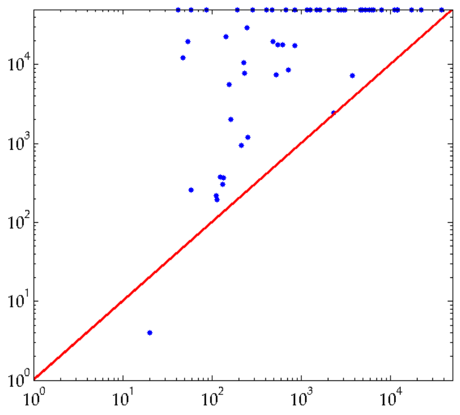

FIG. 6.5. *Comparison of algorithms on sparse nonsymmetric matrices.*

We note that many of the sparse matrices in the UFL collection do not have total support, and we omitted these from the experiment. We tested BNEWT on a number of sparse nonsymmetric matrices which have support but lack total support. In these cases, balancing is impossible but an approximate solution can be found. In many cases BNEWT is robust enough to find an approximate solution in reasonable time (nonzero elements which don't lie on a positive diagonal are almost obliterated by the scaling). Such approximate scalings can be used as preconditioners, but we have found that equally good preconditioners can be found by applying a handful of iterations of SK, even though the resulting diagonal factors do not come close to balancing the matrix.

7. Conclusions.

In many applications, the Sinkhorn-Knopp algorithm offers a simple and reliable approach to balancing; but its limitations become clear when the matrices involved exhibit a degree of sparsity (in particular, the a priori measure of convergence given in [6] fails if the matrix to be balanced has even a single zero). We have clearly shown that an inexact Newton method offers a simple way of overcoming these limitations. By taking care to remain in the positive cone, we see improvements for symmetric and nonsymmetric matrices. Further speed ups many be possible by using more sophisticated preconditioning techniques for the conjugate gradient algorithm. In examples where our new algorithm converge slowly the diagonal factors typically contain widely varying components $((\max_i x_i)/(\min_i x_i) \gg 10^{10}$, say) and we are not aware of applications for accurate balancing in these cases.

REFERENCES

[1] MICHAEL BACHARACH, *Biproportional Matrices & Input-Output Change*, CUP, 1970.

[2] HAMSA BALAKRISHNAN, INSEOK HWANG AND CLAIRE J. TOMLIN, *Polynomial approximation algorithms for belief matrix maintenance in identity management*, Appears in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Vol. 5 (2004), pp. 4874–4879.

[3] MICHEL BALINSKI, *Fair majority voting (or how to eliminate gerrymandering)*, American Mathematical Monthly, 115 (2008), pp. 97–113.

[4] M. DASZYKOWSKI, E. MOSLETH FÆGESTAD, H. GROVE, H. MARTENS AND B. WALCZAK, *Matching 2D gel electrophoresis images with Matlab Image Processing Toolbox*, Chemometrics and Intelligent Laboratory Systems, 96 (2009), pp. 188–195.

[5] IAIN S. DUFF, R. G. GRIMES AND J. G. LEWIS, *Users' Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)*, Technical Report RAL 92-086, Rutherford Appleton Laboratory (1992).

[6] JOEL FRANKLIN AND JENS LORENZ, *On the scaling of multidimensional matrices*, Linear Algebra Appl., 114/115 (1989), pp. 717–735.

[7] MARTIN FÜRER, *Quadratic convergence for scaling of matrices*, in Proc. ALENEX/ANALC 2004, Lars Arge, Giuseppe F. Italiano, and Robert Sedgewick, editors, pp. 216-223, 2004.

[8] MAGNUS R. HESTENES, EDUARD STIEFEL, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 49 (1952), pp. 409–436.

[9] BAHMAN KALANTARI AND LEONID KHACHIYAN, *On the complexity of nonnegative-matrix scaling*, Linear Algebra Appl., 240 (1996), pp. 87–103.

[10] C. T. KELLEY, *Solving Nonlinear Equations with Newton's Method*, SIAM, Philadelphia, 2003.

[11] LEONID KHACHIYAN AND BAHMAN KALANTARI, *Diagonal matrix scaling and linear programming*, SIAM J. Opt., 2 (1992), pp. 668–672.

[12] PHILIP A. KNIGHT, *The Sinkhorn-Knopp algorithm: convergence and applications*, SIMAX, 30 (2008), pp.261–275.

[13] L. YU. KOLOTILINA, *Nonsingularity/singularity criteria for nonstrictly block diagonally dominant matrices*, Linear Algebra Appl., 359 (2003), pp. 133–159.

[14] B. LAMOND AND N. F. STEWART, *Bregman's balancing method*, Transportation Research 15B (1981), pp. 239–248.

[15] NATHAN LINIAL, ALEX SAMORODNITSKY AND AVI WIGDERSON, *Deterministic Strongly Polynomial Algorithm for Matrix Scaling and Approximate Permanents*, Combinatorica, 20 (2000) pp. 545–568.

[16] OREN E. LIVNE AND GENE H. GOLUB, *Scaling by binormalization*, Numerical Algorithms, 35 (2004), pp. 97–120.

[17] ALBERT W. MARSHALL AND INGRAM OLKIN, *Scaling of matrices to achieve specified row and column sums*, Numer. Math., 12 (1968), pp. 83–90.

[18] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.
[19] B. N. Parlett and T. L. Landis *Methods for scaling to doubly stochastic form*, Linear Algebra Appl., 48 (1982), pp. 53–79.
[20] Daniel Ruiz and Bora Ucar *A symmetry preserving algorithm for matrix scaling*, Technical report, INRIA RR-7552 (2011).
[21] Micheal H. Schneider *Matrix scaling, entropy minimization, and conjugate duality (II): The dual problem*, Math. Prog. 48 (1990), pp. 103–124.
[22] Richard Sinkhorn and Paul Knopp *Concerning nonnegative matrices and doubly stochastic matrices*, Pacific J. Math. 21 (1967) , pp. 343–348.
[23] George W. Soules, *The rate of convergence of Sinkhorn balancing*, Linear Algebra Appl., 150 (1991), pp. 3–40.
[24] Olga Taussky, *Bounds for characteristic roots of matrices*, Duke Math. J., 15(1948), pp. 1043–1044.

Appendix. The Symmetric Algorithm.

```
function [x,res] = bnewt(A,tol,x0,delta,Delta,fl)
% BNEWT A balancing algorithm for symmetric matrices
%
% X = BNEWT(A) attempts to find a vector X such that
% diag(X)*A*diag(X) is close to doubly stochastic. A must
% be symmetric and nonnegative.
%
% X0: initial guess.  TOL: error tolerance.
% delta/Delta: how close/far balancing vectors can get
% to/from the edge of the positive cone.
% We use a relative measure on the size of elements.
% FL: intermediate convergence statistics on/off.
% RES: residual error, measured by norm(diag(x)*A*x - e).
% Initialise
n = size(A,1); e = ones(n,1); res=[];
if nargin < 6,  fl = 0; end
if nargin < 5,  Delta = 3; end
if nargin < 4,  delta = 0.1; end
if nargin < 3,  x0 = e; end
if nargin < 2,  tol = 1e-6; end
g=0.9; etamax = 0.1; % Parameters used in inner stopping criterion.
eta = etamax; stop_tol = tol*.5;

x = x0; rt = tol^2; v = x.*(A*x); rk = 1 - v;
rho_km1 = rk'*rk; rout = rho_km1; rold = rout;

MVP = 0;  % We'll count matrix vector products.
i = 0; % Outer iteration count.

if fl == 1, fprintf('it    in. it    res\n'), end
while rout > rt  % Outer iteration
    i = i + 1; k = 0; y = e;
    innertol = max([eta^2*rout,rt]);
    while  rho_km1 > innertol %Inner iteration by CG
        k = k + 1;
        if k == 1
    Z = rk./v;  p=Z; rho_km1 = rk'*Z;
else
    beta=rho_km1/rho_km2;
    p=Z + beta*p;
end
% Update search direction efficiently.
w = x.*(A*(x.*p)) + v.*p;
alpha = rho_km1/(p'*w);
ap = alpha*p;
```

```
        % Test distance to boundary of cone.
        ynew = y + ap;
        if min(ynew) <= delta
            if delta == 0, break, end
            ind = find(ap < 0);
            gamma = min((delta  - y(ind))./ap(ind));
            y = y + gamma*ap;
            break
        end
        if max(ynew) >= Delta
            ind = find(ynew > Delta);
            gamma = min((Delta-y(ind))./ap(ind));
            y = y + gamma*ap;
            break
        end
        y = ynew;
        rk = rk - alpha*w;  rho_km2 = rho_km1;
        Z = rk./v;  rho_km1 = rk'*Z;
    end
    x = x.*y; v = x.*(A*x);
    rk = 1 - v; rho_km1 = rk'*rk; rout = rho_km1;
    MVP = MVP + k + 1;
    % Update inner iteration stopping criterion.
    rat = rout/rold;  rold = rout; res_norm = sqrt(rout);
    eta_o = eta;  eta = g*rat;
    if g*eta_o^2 > 0.1
        eta = max([eta,g*eta_o^2]);
    end
    eta = max([min([eta,etamax]),stop_tol/res_norm]);
    if fl == 1
    fprintf('%3d %6d   %.3e %.3e %.3e \n', i,k, r_norm,min(y),min(x));
    res=[res; r_norm];
    end
end
 fprintf('Matrix-vector products = %6d\n', MVP)
```