PIM 2.0
The Parallel Iterative Methods package for Systems of Linear Equations User's Guide (Fortran 77 version)

Rudnei Dias da Cunha
Mathematics Institute and National Supercomputing Centre
Universidade Federal do Rio Grande do Sul
Brasil

Tim Hopkins
Computing Laboratory
University of Kent at Canterbury
United Kingdom

Abstract

We describe PIM (Parallel Iterative Methods), a collection of Fortran 77 routines to solve systems of linear equations on parallel computers using iterative methods.

A number of iterative methods for symmetric and nonsymmetric systems are available, including Conjugate-Gradients (CG), Bi-Conjugate-Gradients (Bi-CG), Conjugate-Gradients squared (CGS), the stabilised version of Bi-Conjugate-Gradients (Bi-CGSTAB), the restarted stabilised version of Bi-Conjugate-Gradients (RBi-CGSTAB), generalised minimal residual (GMRES), generalised conjugate residual (GCR), normal equation solvers (CGNR and CGNE), quasi-minimal residual (QMR) with coupled two-term recurrences, transpose-free quasi-minimal residual (TFQMR) and Chebyshev acceleration.

The PIM routines can be used with user-supplied preconditioners, and left-, right- or symmetric-preconditioning are supported. Several stopping criteria can be chosen by the user.

In this user's guide we present a brief overview of the iterative methods and algorithms available. The use of PIM is introduced via examples. We also present some results obtained with PIM concerning the selection of stopping criteria and parallel scalability. A reference manual can be found at the end of this report with specific details of the routines and parameters.

Contents

1 Introduction

The Parallel Iterative Methods (PIM) is a collection of Fortran 77 routines designed to solve systems of linear equations (SLEs) on parallel computers using a variety of iterative methods.

PIM offers a number of iterative methods, including

- Conjugate-Gradients (CG) [29],
- Conjugate-Gradients for normal equations with minimisation of the residual norm (CGNR) [35],
- Conjugate-Gradients for normal equations with minimisation of the error norm (CGNE) [10],
- Bi-Conjugate-Gradients (Bi-CG) [21],
- Conjugate-Gradients squared (CGS) [44],
- the stabilised version of Bi-Conjugate-Gradients (Bi-CGSTAB) [46],
- the restarted, stabilised version of Bi-Conjugate-Gradients (RBi-CGSTAB) [43],
- the restarted, generalised minimal residual (RGMRES) [42],
- the restarted, generalised conjugate residual (RGCR) [19],
- the quasi-minimal residual with coupled two-term recurrences (QMR) [24],
- the transpose-free quasi-minimal residual (TFQMR) [23] and
- Chebyshev acceleration [32].

The routines allow the use of preconditioners; the user may choose to use left-, right- or symmetric-preconditioning. Several stopping criteria are also available.

PIM was developed with two main goals

1. To allow the user complete freedom with respect to the matrix storage, access and partitioning;
2. To achieve portability across a variety of parallel architectures and programming environments.

These goals are achieved by hiding from the PIM routines the specific details concerning the computation of the following three linear algebra operations

1. Matrix-vector (and transpose-matrix-vector) product
2. Preconditioning step
3. Inner-products and vector norm

Routines to compute these operations need to be provided by the user. Many vendors supply their own, optimised linear algebra routines which the user may want to use.

A number of packages for the iterative solution of linear systems are available including ITPACK [30] and NSPCG [38]. PIM differs from these packages in three main aspects. First, while ITPACK and NSPCG may be used on a *parallel vector supercomputer* like a Cray Y-MP, there are no versions of these packages available for *distributed-memory parallel* computers. Second, there is no debugging support; this is dictated by the fact that in some multiprocessing environments parallel I/O is not available. The third aspect is that we do not provide a collection of preconditioners but leave the responsibility of providing the appropriate routines to the user.

In this sense, PIM has many similarities to a proposed standard for iterative linear solvers, by Ashby and Seager [5]. In that proposal, the user supplies the matrix-vector product and preconditioning routines. We believe that their proposed standard satisfies many of the needs of the scientific community as, drawing on its concepts, we have been able to provide software that has been used in a variety of parallel and sequential environments. PIM does not always follow the proposal especially with respect to the format of the matrix-vector product routines and the lack of debugging support.

Due to the openness of the design of PIM, it is also possible to use it on a sequential machine. In this case, the user can take advantage of the BLAS [15] to compute the above operations. This characteristic is important for testing purposes; once the user is satisfied that the selection of preconditioners and stopping criteria are suitable, the computation can be accelerated by using appropriate parallel versions of the three linear algebra operations mentioned above.

A package similar to PIM is the Simplified Linear Equation Solvers (SLES) by Gropp and Smith [31], part of the PETSc project. In SLES the user has a number of iterative methods (CG, CGS, Bi-CGSTAB, two variants of the transpose-free QMR, restarted GMRES, Chebyshev and Richardson) which can be used together with built-in preconditioners and can be executed either sequentially or in parallel. The package may be used with any data representation of the matrix and vectors with some routines being provided to create matrices dynamically in its internal format (a feature found on ITPACK). The user can also extend SLES in the sense that it can provide new routines for preconditioners and iterative methods without modifying SLES. It is also possible to debug and monitor the performance of a SLES routine.

Portability of code across different multiprocessor platforms is a very important issue. For distributed-memory multiprocessor computers, a number of public-domain software libraries have appeared, including PVM [28], TCGMSG [33], NXLIB [45], p4 [8] (the latter with support for shared-memory programming). These libraries are available on a number of architectures making it possible to port applications between different parallel computers with few (if any) modifications to the code being necessary. In 1993 the "Message-Passing Interface Forum", a consortium of academia and vendors, drawing on the experiences of users of those and other libraries, defined a standard interface for message-passing operations, called MPI [22]. Today we have available implementations of MPI built on top of other, existing libraries, like the CHIMP/MPI library developed at the Edinburgh Parallel Computer Centre [7], and the Unify project [9] which provides an MPI interface on top of PVM. It is expected that native implementations will be available soon. In the previous releases of PIM (1.0 and 1.1) we had distributed examples using PVM, TCGMSG, p4 and NXLIB; however from this release onwards we will support only PVM, the "de-facto" standard for message-passing, and MPI.

We would like to mention two projects which we believe can be used together with PIM. The first is the proposed standard for a user-level sparse BLAS by Duff *et al.* [18] and Heroux [34]. This standard addresses the common problem of accessing and storing sparse matrices in the context of the BLAS routines; such routines could then be called by the user in conjunction with a PIM routine. The second is the BLACS project by Dongarra *et al.* [16] which provides routines to perform distributed operations over matrices using PVM 3.1.

2 An overview of the iterative methods

How to choose an iterative method from the many available is still an open question, since any one of these methods may solve a particular system in very few iterations while diverging on another. In this section we provide a brief overview of the iterative methods present in PIM. More details are available in the works of Ashby *et al.* [4], Saad [40][41], Nachtigal *et al.* [37], Freund *et al.* [25][26] and Barrett *et al.* [6].

We introduce the following notation. CG, Bi-CG, CGS, Bi-CGSTAB, restarted GMRES, restarted GCR and TFQMR solve a non-singular system of n linear equations of the form

$$Q_1 A Q_2 x = Q_1 b \tag{1}$$

where Q_1 and Q_2 are the preconditioning matrices. For CGNR, the system solved is

$$Q_1 A^T A Q_2 x = Q_1 A^T b \tag{2}$$

and for CGNE we solve the system

$$Q_1 A A^T Q_2 x = Q_1 b \tag{3}$$

CG The CG method is used mainly to solve Hermitian positive-definite (HPD) systems. The method minimises the residual in the A-norm and in finite-precision arithmetic it terminates in at most n iterations. The method does not require the coefficient matrix; only the result of a matrix-vector product Au is needed. It also requires a relatively small number of vectors to be stored per iteration since its iterates can be expressed by short, three-term vector recurrences.

With suitable preconditioners, CG can be used to solve nonsymmetric systems. Holter *et al.* [36] have solved a number of problems arising from the modelling of groundwater flow via finite-differences discretisations of the two-dimensional diffusion equation. The properties of the model led to systems where the coefficient matrix was very ill-conditioned; incomplete factorisations and least-squares polynomial preconditioners were used to solve these systems. Hyperbolic equations of the form

$$\frac{\partial u}{\partial t} + \sigma_1 \frac{\partial u}{\partial x} + \sigma_2 \frac{\partial u}{\partial y} = f(x, y, t)$$

have been solved with CG using a Neumann polynomial approximation to A^{-1} as a preconditioner [11].

CG with eigenvalues estimation An important characteristic of CG is its connection to the Lanczos method [29] which allows us to obtain estimates of the eigenvalues of Q_1AQ_2 with only a little extra work per iteration. These estimates, μ_1 and μ_n, are obtained from the Lanczos tridiagonal matrix T_k whose entries are generated during the iterations of the CG method [29, pp. 475-480, 523-524]. If we define the matrices $\Delta = diag(\rho_0, \rho_1, \ldots, \rho_{k-1})$, $G_k = diag(\xi_0, \xi_1, \ldots, \xi_{k-1})$ and

$$B_k = \begin{bmatrix} 1 & -\beta_2 & & & \\ & 1 & -\beta_3 & & \\ & & 1 & \ddots & \\ & & & \ddots & -\beta_k \\ & & & & 1 \end{bmatrix}$$

where $\rho_i = \|\, r_i \,\|_2$, r_i is the residual at the i-th iteration, $\xi_i = p_i^T A p_i$ and $\beta_i = r_i^T r_i / r_{i-1}^T r_{i-1}$ are generated via the CG iterations (at no extra cost), we obtain the Lanczos's matrix via the relation

$$T_k = \Delta^{-1} B_k^T G_k B_k \Delta^{-1} \tag{4}$$

Due to the structure of the matrices B_k, Δ and G_k, the matrix T_k can be easily updated during the CG iterations. The general formula for T_k is

$$a_i = (\beta_i^2 \xi_{i-2} + \xi_{i-1}) / \rho_{i-1}^2, \quad \beta_1 = 0, \quad i = 1, 2, \ldots, k$$

$$b_i = -\xi_{i-1} \beta_{i+1} / (\rho_{i-1} \rho_i), \quad i = 1, 2, \ldots, k-1$$

where a_i and b_i are the elements along the diagonal and subdiagonal of T_k respectively.

The strategy employed to obtain the eigenvalue estimates is based on Sturm sequences [29, pp. 437-439]. For the matrix T_2, obtained during the first iteration of CG, the eigenvalues are obtained directly from the quadratic equation derived from $p(\mu) = \det(T_2 - \mu I)$. We also set an interval $[c, d] = [\mu_1, \mu_n]$.

For the next iterations, we update the interval $[c, d]$ using Gerschgorin's theorem. This is easily accomplished since at each iteration only two new values are added to T_k to give T_{k+1}; the updated interval is then

$$\begin{aligned} c &= \min(c,\, |a_k| - |b_{k-1}| - |b_k|,\, |a_{k+1}| - |b_k|), \\ d &= \max(d,\, |a_k| + |b_{k-1}| + |b_k|,\, |a_{k+1}| + |b_k|) \end{aligned}$$

The new estimates for the extreme eigenvalues are then computed using a bisection routine applied to the polynomial $p(\mu) = \det(T_{k+1} - \mu I)$ which is computed via a recurrence expression [29, pp. 437]. The intervals $[c, \mu_1]$ and $[\mu_n, d]$ are used in the bisection routine to find the new estimates of μ_1 and μ_n respectively.

A possible use of this routine would be to employ *adaptive* polynomial preconditioners (see [2] and [3]) where at each iteration information about the extreme eigenvalues of $Q_1 A Q_2$ is obtained and the polynomial preconditioner is modified to represent a more accurate approximation to A^{-1}. This routine can also be used as a preliminary step before solving the system using the Chebyshev acceleration routine, `PIM_CHEBYSHEV`.

CGNR and CGNE For nonsymmetric systems, one could use the CG formulation applied to systems involving either $A^T A$ or AA^T; these are called CGNR and CGNE respectively. The difference between both methods is that CGNR minimises the residual $\|\, b - Ax_k \,\|_2$ and CGNE the error $\|\, A^{-1}b - x_k \,\|_2$. A potential problem with this approach is that the condition number of $A^T A$ or AA^T is large even for a moderately ill-conditioned A, thus requiring a substantial number of iterations for convergence. However, as noted by Nachtigal *et al.* [37], CGNR is better than GMRES and CGS for some systems, including circulant matrices. More generally, CGNR and CGNE perform well if the eigenvalue spectrum of A has some symmetries; examples of such matrices are the real skew-symmetric and shifted skew-symmetric matrices $A = e^{i\theta}(T + \sigma I)$, $T = T^H$, θ real and σ complex.

Bi-CG Bi-CG is a method derived to solve non-Hermitian systems of equations, and is closely related to the Lanczos method to compute the eigenvalues of A. The method requires few vectors per iteration and the computation of a matrix-vector product as well as a transpose-matrix-vector product $A^T u$. The iterates of Bi-CG are generated in the Krylov subspace $\mathcal{K}(r_0, A) = \{r_0, r_0 A, r_0 A^2, \ldots\}$, where $r_0 = b - Ax_0$.

A Galerkin condition $\bar{w}^H r_k = 0\,,\ \forall\, w \in \mathcal{K}(\tilde{r}_0, A^T)$, is imposed on the residual vector where $\tilde{r}_0$ is an arbitrary vector satisfying $r_k^T \tilde{r}_0 \neq 0$. It is important to note that two sequences of residual vectors are generated, one involving r_k and A and the other $\tilde{r}_k$ and A^T but the solution vector x_k is updated using only the *first* sequence.

Bi-CG has an erratic convergence with large oscillations of the residual 2-norm which usually cause a large number of iterations to be performed until convergence is achieved. Moreover, the method may break down, for example, the iterations cannot proceed when some quantities (dependent on $\tilde{r}_0$) become zero[1].

CGS CGS is a method that tries to overcome the problems of Bi-CG. By rewriting some of the expressions used in Bi-CG, it is possible to eliminate the need for A^T altogether. Sonneveld [44] also noted that it is possible to (theoretically) increase the rate of convergence of Bi-CG at no extra work per iteration. However, if Bi-CG diverges for some system, CGS diverges even faster. It is also possible that CGS diverges while Bi-CG does not for some systems.

Bi-CGSTAB Bi-CGSTAB is a variant of Bi-CG with a similar formulation to CGS. However, steepest-descent steps are performed at each iteration and these contribute to a considerably smoother convergence behaviour than that obtained with Bi-CG and CGS. It is known that for some systems Bi-CGSTAB may present an erratic convergence behaviour as does Bi-CG and CGS.

RBi-CGSTAB The restarted Bi-CGSTAB, proposed by Sleijpen and Fokkema [43], tries to overcome the stagnation of the iterations of Bi-CGSTAB which occurs with a large class of systems of linear equations. The method combines the restarted GMRES method and Bi-CG, being composed of two specific sections: a Bi-CG part where $(l+1)$ u and r vectors are produced (l being usually 2 or 4), and a *minimal residual* step follows, when the residuals are minimized. RBi-CGSTAB is mathematically equivalent to Bi-CGSTAB if $l = 1$, although numerically their iterations will usually differ. The method does not require the computation of transpose matrix-vector products as in Bi-CG and a smaller number of vectors need to be stored per iteration than for other restarted methods like GMRES.

GMRES The GMRES method is a very robust method to solve nonsymmetric systems. The method uses the Arnoldi process to compute an orthonormal basis $\{v_1, v_2, \ldots, v_k\}$ of the Krylov subspace $\mathcal{K}(A, v_1)$. The solution of the system is taken as $x_0 + V_k y_k$ where V_k is a matrix whose columns are the orthonormal vectors v_i, and y_k is the solution of the least-squares problem $H_k y_k = ||\, r_0 \,||_2 e_1$, where the upper Hessenberg matrix H_k is generated during

[1] The PIM implementation of Bi-CG, CGS and Bi-CGSTAB sets $\tilde{r}_0 = r_0$ but the user may modify the code if another choice of $\tilde{r}_0$ is desirable.

the Arnoldi process and $e_1 = (1, 0, 0, \ldots, 0)^T$. This least-squares problem can be solved using a QR factorisation of H_k.

A problem that arises in connection with GMRES is that the number of vectors of order n that need to be stored grows linearly with k and the number of multiplications grows quadratically. This may be avoided by using a *restarted* version of GMRES; this is the method implemented in PIM. Instead of generating an orthonormal basis of dimension k, one chooses a value c, $c \ll n$, and generates an approximation to the solution using an orthonormal basis of dimension c, thereby reducing considerably the amount of storage needed. Although the restarted GMRES does not break down [42, pp. 865], it may, depending on the system and the value of c, produce a stationary sequence of residuals, thus not achieving convergence. Increasing the value of c usually cures this problem and may also increase the rate of convergence.

A detailed explanation of the parallel implementation of the restarted GMRES used can be found in [13].

GMRES with eigenvalues estimation It is very easy to obtain estimates of the eigenvalues of Q_1AQ_2 at each iteration of GMRES, since the upper Hessenberg matrix H_k computed during the Arnoldi process satisfies $Q_1AQ_2V_k = V_kH_k$. The eigenvalues of H_k approximate those of Q_1AQ_2, especially on the boundaries of the region containing $\lambda(Q_1AQ_2)$. The QR algorithm can be used to obtain the eigenvalues of H_k. The LAPACK routine `_HSEQR` [1, pp. 158-159] is used for this purpose.

The routine `PIM_RGMRESEV` returns a box in the complex plane, defining the minimum and maximum values along the real and imaginary axes. These values can then be used by the Chebyshev acceleration routine, `PIM_CHEBYSHEV`.

GCR The GCR method is generally used in its restarted form for reasons similar to those given above for GMRES. It is mathematically equivalent to the restarted version of GMRES but it is not as robust. It is applicable to systems where the coefficient matrix is of the form $A = \mu I + R$, μ complex and R real symmetric and $A = \mu I + S$, μ real and $S^H = -S$, arising in electromagnetics and quantum chromodynamics applications respectively [4].

QMR with coupled two-term recurrences The QMR method by Freund and Nachtigal [27] overcomes the difficulties associated with the Bi-CG method. The original QMR algorithm uses the three-term recurrences as found in the underlying Lanczos process. In finite-precision arithmetic, though, mathematically equivalent coupled two-term recurrences are more robust than the three-term recurrences. PIM implements the coupled two-term recurrence version of the QMR algorithm as described in [24].

TFQMR TFQMR is a variant of CGS proposed by Freund [23]. TFQMR uses all available search direction vectors instead of the two search vectors used in CGS. Moreover, these vectors are combined using a parameter which can be obtained via a quasi-minimisation of the residual. The method is thus extremely robust and has the advantage of not requiring the computation of transpose matrix-vector products. PIM offers TFQMR with 2-norm weights (see [23, Algorithm 5.1]).

Chebyshev acceleration The Chebyshev acceleration is a polynomial acceleration applied to basic stationary methods of the form

$$x_{k+1} = Gx_k + f$$

where $G = I - Q_1A$, $f = Q_1b$. If we consider k iterations of the above method, the iterates x_k may be linearly combined such that $y = \sum_{j=0}^{k} c_j x_j$ is a better approximation to $x^* = A^{-1}b$. The coefficients c_j are chosen so that the norm of error vector is minimized and $\sum_{j=0}^{k} c_j = 1$. If we assume that the eigenvalues of G are contained in an interval $[\alpha, \beta]$, with $-1 < \alpha \leq \beta < 1$, then the Chebyshev polynomials satisfy the above conditions on the c_j's. We refer the user to [32, pp. 45–58, 332–339] for more details.

The Chebyshev acceleration has the property that its iterates can be expressed by short (three-term) recurrence relations and, especially for parallel computers, no inner-products or vector norms are needed (except for the stopping test). The difficulty associated with the Chebyshev acceleration is the need for good estimates either for the smallest or largest eigenvalues of G if the eigenvalues are real, or in the case of a complex eigenspectrum a region in the complex plane containing the eigenvalues of minimum and maximum modulus.

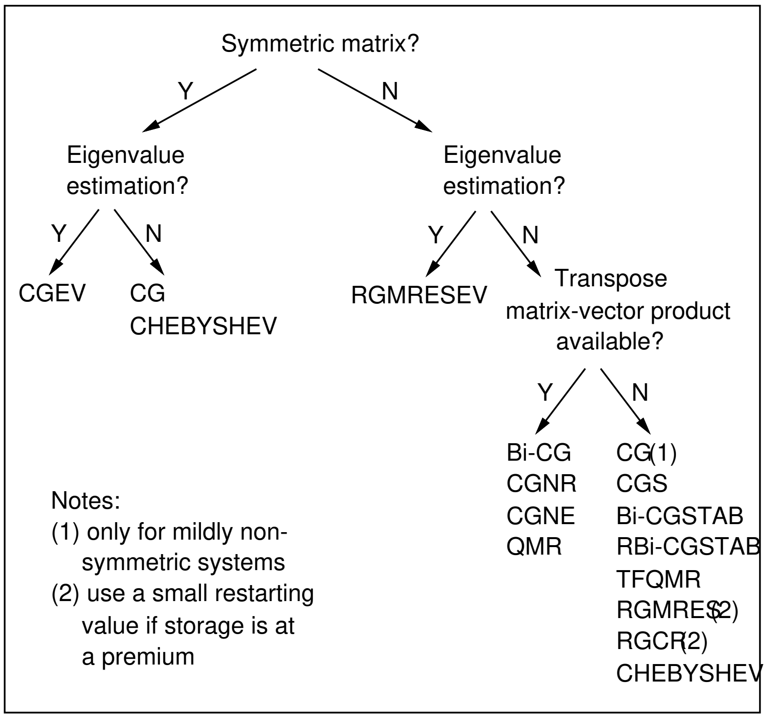
With PIM, the user may make use of two routines, `PIM_CGEV` and `PIM_RGMRESEV`, to obtain such estimates. `PIM_CGEV` covers the case where the eigenvalues of G are real; for the complex case, `PIM_RGMRESEV` should be used. To obtain appropriately accurate estimates, these routines must be used with left-preconditioning, and should be allowed to run for several iterations. The estimates for the eigenvalues of Q_1A should then be modified to those of $I - Q_1A$. This is done by replacing the smallest and largest real values, r and s, by $1 - s$ and $1 - r$ respectively. The imaginary values should not be modified.

We note that even if A has only real eigenvalues, G may have complex (or imaginary only) eigenvalues. In this latter case, the Chebyshev acceleration is defined in terms of a minimum bounding ellipse that contains the eigenvalues of G. If we obtain a box $[r, s, t, u]$ where $r \leq Re(\lambda(G)) \leq s$ and $t \leq Im(\lambda(G)) \leq u$, then the axes of this ellipse are defined as

$$p = \sqrt{2}(r+s)/2, \quad q = \sqrt{2}(t+u)/2$$

These parameters for the Chebyshev iteration are computed by `PIM_CHEBYSHEV`. An example of the use of this routine may be found in Section 4.5.

Figure 1: Selecting an iterative method.



For nonsymmetric systems, one may use a combination of the routine `PIM_RGMRESEV` and `PIM_CHEBYSHEV` as proposed by Elman *et al.* as a *hybrid* method [20, page 847].

To conclude this section, Figure 1 shows a diagram to aid in the selection of an iterative method.

3 Internal details of PIM

3.1 Supported architectures and environments

PIM has been tested on scalar, vector and parallel computers including the Cray Y-MP2E/232, Cray Y-MP C90/16256, SGI Challenge, Intel Paragon, TMC CM-5[2] and networks of workstations under PVM 3.3.6, CHIMP/MPI v1.2, Argonne MPI, p4 v1.2, TCGMSG 4.02 and Intel NX. Table 1 lists the architectures and environments on which PIM has been successfully tested.

[2] The results obtained are based upon a beta version of the software and, consequently, is not necessarily representative of the performance of the full version of this software.

Table 1: Computers where PIM has been tested

Architecture	Compiler and O/S
Sun SPARC	Sun Fortran 1.4 - SunOS 4.1.3
Sun SPARC	Sun Fortran 2.0.1 - SunOS 5.2
Sun SPARC	EPC Fortran 77 - SunOS 4.1.3
Sun SPARC	EPC Fortran 90 - SunOS 4.1.3
Sun SPARC	NAG Fortran 90 - SunOS 4.1.3
DEC AXP 4000/610	DEC Fortran 3.3-1 - DEC OSF/1 1.3
DEC AXP 3000/800	DEC Fortran 3.4-480 - DEC OSF/1 2.0
SGI IRIS Indigo	MIPS Fortran 4.0.5 - SGI IRIX 4.0.5F
SGI IRIS Crimson	MIPS Fortran 4.0.5 - SGI IRIX 4.0.5C
SGI Indy II	MIPS Fortran 5.0 - SGI IRIX 5.1.1
Cray Y-MP2E/232	Cray Fortran 6.0 - UNICOS 7.0.5.2
Cray Y-MP C90/16256	Cray Fortran 7.0 - UNICOS 8.2.3
SGI Challenge	MIPS Fortran 5.2 - SGI IRIX 5.2
Intel Paragon XP/S	Portland if77 4.5 - OSF/1 1.2.6
IBM 9076 SP/1	IBM XL Fortran 6000 2.3 - AIX 3.2
Cray T3D	Cray Fortran 8.0 - UNICOS 8.3.3
TMC CM-5	CM Fortran 77

3.2 Parallel programming model

PIM uses the *Single Program, Multiple Data* (SPMD) programming model. The main implication of using this model is that certain scalar values are needed in each processing element (PE). Two of the user-supplied routines, to compute a global sum and a vector norm, must provide for this, preferably making use of a *reduction* and/or *broadcast* routine like those present on PVM 3.3.6 and MPI.

3.3 Data partitioning

With PIM, the iterative method routines have no knowledge of the way in which the user has chosen to store and access either the coefficient or the preconditioning matrices. We thus restrict ourselves to partitioning the vectors.

The assumption made is that each PE knows the *number of elements* of each vector stored in it and that *all* vector variables in a processor have the *same* number of elements. This is a broad assumption that allows us to accommodate many different data partitioning schemes, including contiguous, cyclic (or wrap-around) and scattered partitionings. We are able to make this assumption because the vector-vector operations used – vector accumulations, assignments and copies – are *disjoint element-wise*. The other operations used involving matrices and vectors which may require knowledge of the individual indices of vectors, are the responsibility of the user.

PIM requires that the elements of vectors must be stored locally starting from position 1; thus the user has a *local* numbering of the variables which can be translated to a *global* numbering if required. For example, if a vector of 8 elements is partitioned in wrap-around fashion among 2 processors, using blocks of length 1, then the first processor stores elements 1, 3, 5 and 7 in the first four positions of an array; the second processor then stores elements 2, 4, 6 and 8 in positions 1 to 4 on its array. We stress that for most of the commonly used partitioning schemes data may be retrieved with very little overhead.

3.4 Increasing the parallel scalability of iterative methods

One of the main causes for the poor scalability of implementations of iterative methods on distributed-memory computers is the need to compute inner-products, $\alpha = u^T v = \sum_{i=1}^{n} u_i v_i$, where u and v are vectors distributed across p processors (without loss of generality assume that each processor holds n/p elements of each vector). This computation can be divided in three parts

1. The *local computation* of partial sums of the form $\beta_j = \sum_{i=1}^{n/p} u_i v_i$, on each processor,
2. The *reduction* of the β_j values, where these values travel across the processors in some efficient way (for instance, as if traversing a binary-tree up to its root) and are summed during the process. At the end, the value of $\alpha = \sum_{j=1}^{p} \beta_j$ is stored in a single processor,
3. The *broadcast* of α to all processors.

During parts 2. and 3. a number of processors are idle for some time. A possible strategy to reduce this idle time and thus increase the scalability of the implementation, is to re-arrange the operations in the algorithm so that parts 2. and 3. accumulate a number of partial sums corresponding to some inner-products. Some of the algorithms available in PIM, including CG, CGEV, Bi-CG, CGNR and CGNE have been rewritten using the approach suggested by D'Azevedo and Romine [14]. Others, like Bi-CGSTAB, RBi-CGSTAB, RGCR, RGMRES and QMR have not been re-arranged but some or all of their inner-products can be computed with a single global sum operation.

The computation of the last two parts depends on the actual message-passing library being used. With MPI, parts 2. and 3. are also offered as a single operation called `MPI_ALLREDUCE`. Applications using the PVM 3.3.6 Fortran interface should however call `PVMFREDUCE` and then `PVMFBROADCAST`.

An important point to make is that we have chosen modifications to the iterative methods that reduce the number of synchronization points while at the same time maintaining their convergence properties and numerical qualities. This is the case of the D'Azevedo and Romine modification; also, in the specific case of GMRES, which uses the Arnoldi process (a suitable reworking of the modified Gram-Schmidt procedure) to compute a vector basis, the computation of several inner-products with a single global sum does not compromise numerical stability.

For instance, in the algorithm for the restarted GMRES (see Algorithm A.9), step 5 involves the computation of j inner-products of the form $V_i^T V_j$, $i = 1, 2, \ldots, j$. It is thus possible to arrange for each processor to compute j partial sums using the BLAS routine `_DOT` and store these in an array. Then in a single call to a reduction routine, these arrays are communicated among the processors and their individual elements are summed. On the completion of the global sum the array containing the respective j inner-products is stored in a single processor and is then broadcast to the remaining processors.

The CGS and TFQMR implementations available on PIM do not benefit from this approach.

3.5 Stopping criteria

PIM offers a number of stopping criteria which may be selected by the user. In Table 2 we list the different criteria used; $r_k = b - Ax_k$ is the *true* residual of the current estimate x_k, z_k is the pseudo-residual (usually generated by linear recurrences and possibly involving the preconditioners) and ε is the user-supplied tolerance. Note that the norms are not indicated; these depend on the user-supplied routine to compute a vector norm.

Table 2: Stopping criteria available on PIM

No.	Stopping criterion
1	$\|\| r_k \|\| < \varepsilon$
2	$\|\| r_k \|\| < \varepsilon \|\| b \|\|$
3	$\sqrt{r_k^T z_k} < \varepsilon \|\| b \|\|$
4	$\|\| z_k \|\| < \varepsilon$
5	$\|\| z_k \|\| < \varepsilon \|\| b \|\|$
6	$\|\| z_k \|\| < \varepsilon \|\| Q_1 b \|\|$
7	$\|\| x_k - x_{k-1} \|\| < \varepsilon$

If speed of execution is of the foremost importance, the user needs to select the stopping criterion that will impose the minimum overhead. The following notes may be of use in the selection of an appropriate stopping criterion

1. If the stopping criterion selected is one of **1**, **2** or **3** then the true residual is computed (except when using TFQMR with either no preconditioning or left preconditioning).
2. The restarted GMRES method uses its own stopping criterion (see [42, page 862]) which is equivalent to the 2-norm of the residual (or pseudo-residual if preconditioning is used).
3. If either no preconditioning or right-preconditioning is used and criterion **6** is selected, the PIM iterative method called will flag the error and exit without solving the system (except for the restarted GMRES routine).

4 Using PIM

4.1 Naming convention of routines

The PIM routines have names of the form

`PIM_`*method*

where `_` indicates single-precision (`S`), double-precision (`D`), complex (`C`) or double-precision complex (`Z`) and *method* is one of: `CG`, `CGEV` (CG with eigenvalue estimation), `CGNR`, `CGNE`, `BICG`, `CGS`, `BICGSTAB`, `RBICGSTAB`, `RGMRES`, `RGMRESEV` (RGMRES with eigenvalue estimation), and `RGCR`, `QMR`, `TFQMR` and `CHEBYSHEV`.

4.2 Obtaining PIM

PIM 2.0 is available via anonymous ftp from

`unix.hensa.ac.uk`, file `/pub/misc/netlib/pim/pim20.tar.Z`

and

`ftp.mat.ufrgs.br`, file `/pub/pim/pim20.tar.gz`

There is also a PIM World-Wide-Web homepage which can be accessed at

`http://www.mat.ufrgs.br/pim-e.html`

which gives a brief description of the package and allows the reader to download the software and related documentation.

The current distribution contains

- The PIM routines in the directories `single`, `double`, `complex` and `dcomplex`
- A set of example programs for sequential and parallel execution (using PVM and MPI) in the directories `examples/sequential`, `examples/pvm` and `examples/mpi`,
- This guide in PostScript format in the `doc` directory.

4.3 Installing PIM

To install PIM, unpack the distributed `compressed` (or `gzipped`), `tar` file:

```
uncompress pim20.tar.Z  (or  gunzip pim20.tar.gz)
tar xfp pim20.tar
cd pim
```

and edit the `Makefile`. The following variables may need to be modified

`HOME` Your top directory, e.g., `/u1/users/fred`

`FC` Your Fortran compiler of choice, usually `f77`

`FFLAGS` Flags for the Fortran compilation of main programs (example programs)

`OFFLAGS` Flags for the Fortran compilation of separate modules (PIM routines and modules of examples)
NOTE: This must include at least the flag required for separate compilation (usually `-c`)

`AR` The archiver program, usually `ar`

`HASRANLIB` Either `t` (true) or `f` (false), indicating if it is necessary to use a random library program (usually `ranlib`) to build the PIM library

`BLASLIB` Either the name of an archive file containing the BLAS library or `-lblas` if the library `libblas.a` has been installed on a system-wide basis

`PARLIB` The compilation switches for any required parallel libraries. This variable must be left blank if PIM is to be used in sequential mode. For example, if PVM 3 is to be used, then `PARLIB` would be defined as
`-L$(PVM_ROOT)/lib/$(PVM_ARCH) -lfpvm3 -lpvm3 -lgpvm3`

Each iterative method routine is stored in a separate file with names *in lower case* following the naming convention of the routines, e.g., the routine `PIMDCG` is stored in the file `pim20/double/pimdcg.f`.

Building the PIM core functions PIM needs the values of some machine-dependent floating-point constants. The single- or double-precision values are stored in the files `pim20/common/smachcons.f` and `pim20/common/dmachcons.f` respectively. Default values are supplied for the IEEE-754 floating-point standard, and are stored separately in the files `pim/common/smachcons.f.ieee754` and `pim/common/dmachcons.f.ieee754` – these are used by default. However if you are using PIM on a computer which does not support the IEEE-754 standard, you may:

1. type make smachcons or make dmachcons; this will compile and execute a program which uses the LAPACK routine _LAMCH, to compute those constants, and the relevant files will be generated.
2. edit either pim/common/smachcons.f.orig or pim/common/dmachcons.f.orig and replace the strings MACHEPSVAL, UNDERFLOWVAL and OVERFLOWVAL by the values of the *machine epsilon*, *underflow* and *overflow thresholds* to those of the particular computer you are using, either in single- or double-precision.

To build PIM, type make makefiles to build the makefiles in the appropriate directories and then make single, make double, make complex or make dcomplex to build the single-precision, double-precision, complex or double complex versions of PIM. This will generate .o files, one for each iterative method routine, along with the library file libpim.a which contains the support routines.

Building the examples Example programs are provided for sequential use, and for parallel use with MPI and PVM[3].

The example programs require a timing routine. The distribution comes with the file examples/common/timer.f which contains examples of the timing functions available on the Cray, the IBM RS/6000 and also the UNIX etime function. By default, the latter is used; this file *must* be modified to use the timing function available on the target machine.

The PVM and MPI example programs use the Fortran INCLUDE statement to include the PVM and MPI header files. Some compilers have a switch (usually -I) which allows the user to provide search directories in which files to be included are located (as with the IBM AIX XL Fortran compiler); while others require the presence of those files in the same directory as the source code resides. In the first case, you will need to include in the FFLAGS variable the relevant switches (see §4.3); in the latter, you will need to install the PVM and MPI header files (fpvm3.h and mpif.h respectively) by typing

```
make install-pvm-include INCFILE=<name-of-fpvm3.h>
make install-mpi-include INCFILE=<name-of-mpif.h>
```

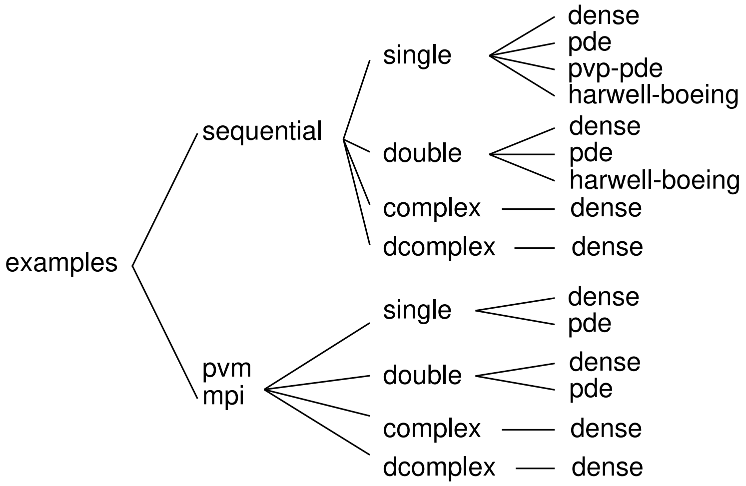
where you should replace <name-of-fpvm3.h> and <name-of-mpif.h> by the full filename of the required include files; for instance, if PVM is installed on /usr/local/pvm3 then you should type

```
make install-pvm-include INCFILE=/usr/local/pvm3/include/fpvm3.h
```

Figure 2 shows the directory tree containing the examples. To build them, type make followed by the name of a subdirectory of examples, e.g., make sequential/single/dense.

[3] The PVM examples use the "groups" library libgpvm.a which provides the *reduction* functions.

Figure 2: Directories containing the examples.



The example programs can also be built locally in those directories by changing to a specific directory and typing `make`.

Cleaning-up You may wish to remove some or all of the compiled codes or other files installed under the PIM directory; in this case you may type one of the following

```
make singleclean
make doubleclean
make complexclean
make dcomplexclean
make sequentialclean
make pvmclean
make mpiclean
make clean-pvm-include
make clean-mpi-include
make examplesclean
make makefilesclean
make realclean
```

which will clean-up the PIM routines, the examples, the Makefiles, the include files and all generated files, returning the package to its distribution form.

Using PIM in your application To use PIM with your application, link your program with the `.o` file corresponding to the PIM iterative method routine being called and with the PIM support library `libpim.a`.

4.4 Calling a PIM iterative method routine

With the exception of the Bi-CG, CGNR, CGNE and QMR methods, all the implemented methods have the same parameter list as CG. The argument list for the double-precision implementation of the CG method is

```
 SUBROUTINE PIMDCG(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,
+                 PDSUM,PDNRM,PROGRESS)
```

and for Bi-CG (as well as for CGNR, CGNE and QMR)

```
 SUBROUTINE PIMDBICG(X,B,WRK,IPAR,DPAR,MATVEC,TMATVEC,PRECONL,PRECONR,
+                   PDSUM,PDNRM,PROGRESS)
```

where the parameters are as follows

Parameter	Description
`X`	A vector of length `IPAR(4)`
	On input, contains the initial estimate
	On output, contains the last estimate computed
`B`	The right-hand-side vector of length `IPAR(4)`
`WRK`	A work vector used internally (see the description
	of each routine for its length)
`IPAR`	An integer array containing input-output parameters
`_PAR`	A floating-point array containing input-output parameters
`MATVEC`	Matrix-vector product external subroutine
`TMATVEC`	Transpose-matrix-vector product external subroutine
`PRECONL`	Left-preconditioning external subroutine
`PRECONR`	Right-preconditioning external subroutine
`P_SUM`	Global sum (reduction) external function
`P_NRM`	Vector norm external function
`PROGRESS`	Monitoring routine

Note in the example above that, contrary to the proposal in [5], PIM uses separate routines to compute the matrix-vector and transpose-matrix-vector products. See the reference manual, sections A.1 and A.2 for the description of the parameters above and the synopsis of the external routine.

4.5 External routines

As stated earlier, the user is responsible for supplying certain routines to be used internally by the iterative method routines. One of the characteristics of PIM is that if external routines are not required by an iterative method routine they are not called (the only exception being the monitoring routines). The user only needs to provide those subroutines that will actually be called by an iterative method routine, depending on the selection of method, preconditioners and stopping criteria; dummy parameters may be passed in place of those that are not used. Some compilers may require the presence of all routines used in the program during the linking phase of the compilation; in this case the user may need to provide stubs for the dummy routines. Section A.2 gives the synopsis of each user-supplied external routine used by PIM.

The external routines have a fixed parameter list to which the user must adhere (see §A.2). Note that (from version 2.0 onwards) the coefficient and the preconditioning matrices do not appear in the parameter list of the PIM routines. Indeed we regard the matrix-vector products and preconditioning routines as *operators* returning only the appropriate resulting vector; thus the PIM routines have no knowledge of the way in which the matrices are stored.

The external routines, however, may access the matrices declared in the main program via `COMMON` blocks. This strategy hides from the PIM routines details of how the matrices are declared in the main program and thus allows the user to choose the most appropriate storage method for her problem; previous versions of PIM were more restrictive in this sense.

Matrix-vector product Consider as an example a dense matrix partitioned by contiguous columns among a number of processors. For illustrative purposes we assume that `N` is an integer multiple of `NPROCS`, and that `LOCLEN=N/NPROCS`. The following code may then be used

```
      PROGRAM MATV

* A IS DECLARED AS IF USING A COLUMN PARTITIONING FOR AT LEAST
* TWO PROCESSORS.
      INTEGER LDA
      PARAMETER (LDA=500)
      INTEGER LOCLEN
      PARAMETER (LOCLEN=250)
      DOUBLE PRECISION A(LDA,LOCLEN)
      COMMON /PIMA/A

* SET UP PROBLEM SOLVING PARAMETERS FOR USE BY USER DEFINED ROUTINES
* THE USER MAY NEED TO SET MORE VALUES OF THE IPAR ARRAY
* LEADING DIMENSION OF A
      IPAR(1)=LDA
* NUMBER OF ROWS/COLUMNS OF A
      IPAR(2)=N
* NUMBER OF PROCESSORS
```

```
      IPAR(6)=NPROCS
* NUMBER OF ELEMENTS STORED LOCALLY
      IPAR(4)=N/IPAR(6)
* CALL PIM ROUTINE
      CALL PIMDCG(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)
      STOP
      END

* MATRIX-VECTOR PRODUCT ROUTINE CALLED BY A PIM ROUTINE. THE
* ARGUMENT LIST TO THIS ROUTINE IS FIXED.
      SUBROUTINE MATVEC(U,V,IPAR)
      DOUBLE PRECISION U(*),V(*)
      INTEGER IPAR(*)
      INTEGER LDA
      PARAMETER (LDA=500)
      INTEGER LOCLEN
      PARAMETER (LOCLEN=250)
      DOUBLE PRECISION A(LDA,LOCLEN)
      COMMON /PIMA/A
      ⋮
      RETURN
      END
```

The scheme above can be used for the transpose-matrix-vector product as well. We note that many different storage schemes are available for storing sparse matrices; the reader may find useful to consult Barrett *et al.* [6, pp. 57ff] where such schemes as well as algorithms to compute matrix-vector products are discussed.

Preconditioning For the preconditioning routines, one may use the scheme outlined above for the matrix-vector product; in some cases this may not be necessary, when there is no need to operate with A or the preconditioner is stored as a vector. An example is the diagonal (or Jacobi) left-preconditioning, where $Q_1 = \mathrm{diag}(A)^{-1}$

```
      PROGRAM DIAGP
      INTEGER LDA
      PARAMETER (LDA=500)
      INTEGER LOCLEN
      PARAMETER (LOCLEN=250)

* Q1 IS DECLARED AS A VECTOR OF LENGTH 250, AS IF USING AT LEAST
* TWO PROCESSORS.
      DOUBLE PRECISION A(LDA,LOCLEN),Q1(LOCLEN)
      COMMON /PIMQ1/Q1
      EXTERNAL MATVEC,DIAGL,PDUMR,PDSUM,PDNRM
```

```
* SET UP PROBLEM SOLVING PARAMETERS FOR USE BY USER DEFINED ROUTINES
* THE USER MAY NEED TO SET MORE VALUES OF THE IPAR ARRAY
* LEADING DIMENSION OF A
      IPAR(1)=LDA
* NUMBER OF ROWS/COLUMNS OF A
      IPAR(2)=N
* NUMBER OF PROCESSORS
      IPAR(6)=NPROCS
* NUMBER OF ELEMENTS STORED LOCALLY
      IPAR(4)=N/IPAR(6)
* SET LEFT-PRECONDITIONING
      IPAR(8)=1
      :
      DO 10 I=1,N
        Q1(I)=1.0D0/A(I,I)
 10 CONTINUE
      :
      CALL DINIT(IPAR(4),0.0D0,X,1)
      CALL PIMDCG(X,B,WRK,IPAR,DPAR,MATVEC,DIAGL,PDUMR,PDSUM,PDNRM,PROGRESS)
      STOP
      END
      :
      SUBROUTINE DIAGL(U,V,IPAR)
      DOUBLE PRECISION U(*),V(*)
      INTEGER IPAR(*)
      INTEGER LOCLEN
      PARAMETER (LOCLEN=250)
      DOUBLE PRECISION Q1(LOCLEN)
      COMMON /PIMQ1/Q1
      CALL DCOPY(IPAR(4),U,1,V,1)
      CALL DVPROD(IPAR(4),Q1,1,V,1)
      RETURN
```

where DVPROD is a routine based on the BLAS DAXPY routine that performs an element-by-element vector multiplication. This example also shows the use of dummy arguments (PDUMR).

Note that it is the responsibility of the user to ensure that, when using preconditioning, the matrix Q_1AQ_2 must satisfy any requirements made by the iterative method being used with respect to the symmetry and/or positive-definiteness of the matrix. For example, if A is a matrix with arbitrary (i.e., non-constant) diagonal entries, then both $\mathrm{diag}(A)^{-1}\,A$ and $A\,\mathrm{diag}(A)^{-1}$ will not be symmetric, and the CG and CGEV methods will generally fail to converge. For these methods symmetric preconditioning, $\mathrm{diag}(A)^{-1/2}\,A\,\mathrm{diag}(A)^{-1/2}$, should be used.

Inner-products, vector norms and global accumulation When running PIM routines on multiprocessor architectures, the inner-product and vector norm routines require reduction and broadcast operations (in some message-passing libraries these can be supplied by a single routine). On vector processors these operations are handled directly by the hardware whereas on distributed-memory architectures these operations involve the exchange of messages among the processors.

When a PIM iterative routine needs to compute an inner-product, it calls _DOT to compute the partial inner-product values. The user-supplied routine P_SUM is then used to generate the global sum of those partial sums. The following code shows the routines to compute the global sum and the vector 2-norm $||\,u\,||_2 = \sqrt{u^T u}$ using the BLAS DDOT routine and the reduction-plus-broadcast operation provided by MPI

```
      SUBROUTINE PDSUM(ISIZE,X)

      INCLUDE 'mpif.h'
      INTEGER ISIZE
      DOUBLE PRECISION X(*)
      DOUBLE PRECISION WRK(10)
      INTEGER IERR
      EXTERNAL DCOPY,MPI_ALLREDUCE
      CALL MPI_ALLREDUCE(X,WRK,ISIZE,MPI_DOUBLE_PRECISION,MPI_SUM,
     +                   MPI_COMM_WORLD,IERR)
      CALL DCOPY(ISIZE,WRK,1,X,1)

      RETURN
      END

      DOUBLE PRECISION FUNCTION PDNRM(LOCLEN,U)

      INCLUDE 'mpif.h'
      INTEGER LOCLEN
      DOUBLE PRECISION U(*)
      DOUBLE PRECISION PSUM
      INTEGER IERR
      DOUBLE PRECISION DDOT
      EXTERNAL DDOT
      INTRINSIC SQRT
      DOUBLE PRECISION WRK(1)
      EXTERNAL MPI_ALLREDUCE
      PSUM = DDOT(LOCLEN,U,1,U,1)
      CALL MPI_ALLREDUCE(PSUM,WRK,1,MPI_DOUBLE_PRECISION,MPI_SUM,
     +                   MPI_COMM_WORLD,IERR)
      PDNRM = SQRT(WRK(1))

      RETURN
      END
```

It should be noted that P_SUM is actually a wrapper to the global sum routines available on a particular machine. Also, when executing PIM on a sequential computer, these routines are *empty* i.e., the contents of the array `X` *must* not be altered in any way since its elements already are the inner-product values.

The parameter list for these routines was decided upon after inspecting the format of the global operations available from existing message-passing libraries.

Monitoring the iterations In some cases, most particularly when selecting the iterative method to be used for solving a specific problem, it is important to be able to obtain feedback from the PIM routines as to how an iterative method is progressing.

To this end, we have included in the parameter list of each iterative method routine an external subroutine (called `PROGRESS`) which receives from that routine the number of vector elements stored locally (`LOCLEN`), the iteration number (`ITNO`), the norm of the residual (`NORMRES`) (according to the norm being used), the current iteration vector (`X`), the residual vector (`RES`) and the *true* residual vector $r_k = b - Ax_k$, (`TRUERES`). This last vector contains meaningful values only if `IPAR(9)` is 1, 2 or 3.

The parameter list of the monitoring routine is fixed, as shown in §A.2. The example below shows a possible use of the monitoring routine, for the `DOUBLE PRECISION` data type.

```
      SUBROUTINE PROGRESS(LOCLEN,ITNO,NORMRES,X,RES,TRUERES)
      INTEGER LOCLEN,ITNO
      DOUBLE PRECISION NORMRES
      DOUBLE PRECISION X(*),RES(*),TRUERES(*)
      EXTERNAL PRINTV
      WRITE (6,FMT=9000) ITNO,NORMRES
      WRITE (6,FMT=9010) 'X:'
      CALL PRINTV(LOCLEN,X)
      WRITE (6,FMT=9010) 'RESIDUAL:'
      CALL PRINTV(LOCLEN,RES)
      WRITE (6,FMT=9010) 'TRUE RESIDUAL:'
      CALL PRINTV(LOCLEN,TRUERES)
      RETURN
 9000 FORMAT (/,I5,1X,D16.10)
 9010 FORMAT (/,A)
      END

      SUBROUTINE PRINTV(N,U)
      INTEGER N
      DOUBLE PRECISION U(*)
      INTEGER I
      DO 10 I = 1,N
          WRITE (6,FMT=9000) U(I)
   10 CONTINUE
      RETURN
 9000 FORMAT (4(D14.8,1X))
```

```
      END
```

As with the other external routines used by PIM, this routine needs to be supplied by the user; we have included the source code for the routine as shown above in the directory `/pim/examples/common` and this may be used as is or can be modified by the user as required. Please note that for large system sizes the routine above will produce very large amounts of output. We stress that this routine is always called by the PIM iterative method routines; if no monitoring is needed a dummy routine *must* be provide.

Note that some of the iterative methods contain an inner loop within the main iteration loop. This means that, for `PIM_RGCR` and `PIM_TFQMR`, the value of `ITNO` passed to `PROGRESS` will be repeated as many times as the inner loop is executed. We did not modify the iteration number passed to `PROGRESS` so as to reflect the true behaviour of the iterative method being used.

4.6 Example programs

In the distributed software the user will find a collection of example programs under the directory `examples`. The example programs show how to use PIM with three different matrix storage formats including dense matrices, those derived from the five-point finite-difference discretisation of a partial differential equation (PDE) and the standard sparse representation found in the Harwell-Boeing sparse matrix collection [17].

Most of the examples are provided for sequential and parallel execution, the latter with separate codes for PVM 3.3.6 and MPI libraries. The examples involving the Harwell-Boeing sparse format are provided for sequential execution only.

The parallel programs for the dense and PDE storage formats have different partitioning strategies and the matrix-vector products have been designed to take advantage of these.

The systems solved have been set-up such that the solution is the vector $x = (1, 1, \ldots, 1)^T$, in order to help in checking the results. For the dense storage format, the real system has the tridiagonal coefficient matrix of order $n = 500$

$$A = \begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{bmatrix}$$

and the complex system of order $n = 100$ has the form $A = \mu I + S$, where $S = S^H$, $\mu = 4 - 4i$ and

$$S = \begin{bmatrix} 0 & 1+i & & & \\ 1-i & 0 & 1+i & & \\ & \ddots & \ddots & \ddots & \\ & & 1-i & 0 & 1+i \\ & & & 1-i & 0 \end{bmatrix} \tag{5}$$

The problem using the Harwell-Boeing format is NOS4 from the LANPRO collection of problems in structural engineering [17, pp. 54-55]. Problem NOS4 has order $n = 100$ and is derived from a finite-element approximation of a beam structure. For the PDE storage format the system being solved is derived from the five-point finite-difference discretisation of the convection-diffusion equation

$$-\epsilon\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + \cos(\alpha)\frac{\partial u}{\partial x} + \sin(\alpha)\frac{\partial u}{\partial y} = 0 \tag{6}$$

on the unit square, with $\epsilon = 0.1$, $\alpha = -\pi/6$ and $u = x^2 + y^2$ on ΩR. The first order terms were discretised using forward differences (this problem was taken from [44]).

A different set of systems is used for the `HYBRID` examples with dense storage format. The real system has a nonsymmetric tridiagonal coefficient matrix of order $n = 500$

$$A = \begin{bmatrix} 2 & -1 & & & \\ 2 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & 2 & 2 & -1 \\ & & & 2 & 2 \end{bmatrix}$$

and the complex system of order $n = 100$ has A defined as

$$A = \begin{bmatrix} 2 & -1+i & & & \\ 2+i & 2 & -1+i & & \\ & \ddots & \ddots & \ddots & \\ & & 2+i & 2 & -1+i \\ & & & 2+i & 2 \end{bmatrix}$$

The examples include the solution of systems using different preconditioners. In the dense and Harwell-Boeing formats the examples include diagonal and polynomial preconditioners; the five-point PDE format includes a variant of the incomplete LU factorisation and polynomial preconditioners. The polynomial preconditioners provided are the Neumann and the weighted and unweighted least-squares polynomials found in [36].

4.6.1 Eigenvalues estimation and Chebyshev acceleration

Consider the use of Chebyshev acceleration to obtain a solution to a linear system whose coefficient matrix has real entries only; the eigenvalues of the iteration matrix $I - Q_1 A$ are known to lie in the complex plane. We can use a few iterations of the routine `PIMDRGMRESEV` to obtain estimates of the eigenvalues of $Q_1 A$ and then switch to `PIMDCHEBYSHEV`. Before the latter is called a transformation on the extreme values on the real axis must be made as described in Section 2.

In the example below, we use the Jacobi preconditioner as shown in §4.5. Note that the vector `X` returned by `PIMDRGMRESEV` may be used as an improved initial vector for the routine `PIMDCHEBYSHEV`. Both routines are combined in a loop to produce a hybrid method; the code below is based on the algorithm given by Elman *et al.* [20, page 847].

```
      PROGRAM HYBRID
      INTEGER MAXIT
      EXTERNAL MATVEC,PRECON,PDUMR,PDSUM,PDNRM2

* SET MAXIMUM NUMBER OF ITERATIONS FOR THE HYBRID LOOP
      MAXIT=INT(N/2)+1

* SET LEFT-PRECONDITIONING
      IPAR(8)=1
      CALL DINIT(N,0.0D0,X,1)
      DO 10 I = 1,MAXIT

* SET SMALL NUMBER OF ITERATIONS FOR RGMRESEV
          IPAR(10)=3
          CALL PIMDRGMRESV(X,B,WRK,IPAR,DPAR,MATVEC,PRECONR,PDUMR,PDSUM,PDNRM,PROGRESS)
          IF (IPAR(12).NE.-1) THEN
              IPAR(11) = I
              GO TO 20
          END IF

* MODIFY REAL INTERVAL TO REFLECT EIGENVALUES OF I-Q1A. BOX CONTAINING
* THE EIGENVALUES IS RETURNED IN DPAR(3), DPAR(4), DPAR(5), DPAR(6),
* THE FIRST TWO ARE THE INTERVAL ALONG THE REAL AXIS, THE LAST TWO ARE
* THE INTERVAL ALONG THE IMAGINARY AXIS.
          MU1 = DPAR(3)
          MUN = DPAR(4)
          DPAR(3) = 1.0D0 - MUN
          DPAR(4) = 1.0D0 - MU1

* SET NUMBER OF ITERATIONS FOR CHEBYSHEV
          IPAR(10)=5
          CALL PIMDCHEBYSHEV(X,B,DWRK,IPAR,DPAR,MATVEC,PRECON,PDUMR,PDSUM,PDNRM2,PROGRESS)
          IF ((IPAR(12).EQ.0) .OR. (IPAR(12).EQ.-6) .OR.
```

```
     +        (IPAR(12).EQ.-7)) THEN
              IPAR(11) = I
              GO TO 20
           END IF
   10 CONTINUE
   20 CONTINUE
      ⋮
```

4.6.2 Dense storage

For the dense case, the coefficient matrix is partitioned by columns among the p processors, which are considered to be *logically* connected on a grid (see Figure 3-A). Each processor stores at most $\lceil n/p \rceil$ columns of A. For the example shown in Figure 3-B, the portion of the matrix-vector product to be stored in processor 0 is computed according to the diagram shown in Figure 3-C. Basically, each processor computes a vector with the same number of elements as that of the *target* processor (0 in the example) which holds the partial sums for each element. This vector is then sent across the network to be summed in a recursive-doubling fashion until the accumulated vectors, carrying the contributions of the remaining processors, arrive at the target processor. These accumulated vectors are then summed together with the partial sum vector computed locally in the target processor, yielding the elements of the vector resulting from the matrix-vector product. This process is repeated for all processors. This algorithm is described in [12].

To compute the dense transpose-matrix-vector product, $A^T u$, each processor broadcasts to the other processors a copy of its own part of u. The resulting part of the v vector is then computed by each processor.

4.6.3 PDE storage

For the PDE storage format, a square region is subdivided into $l+1$ rows and columns giving a grid containing l^2 internal points, each point being numbered as $i+(j-1)l$, $i, j = 1, 2, \ldots, l$ (see Figure 4). At each point we assign 5 different values corresponding to the *center*, *north*, *south*, *east* and *west* points on the stencil ($\alpha_{i,j}$, $\beta_{i,j}$, $\gamma_{i,j}$, $\delta_{i,j}$, $\varepsilon_{i,j}$ respectively) which are derived from the PDE and the boundary conditions of the problem. Each grid point represents a variable; the whole being obtained by solving a linear system of order $n = l^2$.

A matrix-vector product $v = Au$ is obtained by computing

$$v_{i,j} = \alpha_{i,j} u_{i,j} + \beta_{i,j} u_{i+1,j} + \gamma_{i,j} u_{i-1,j} + \delta_{i,j} u_{i,j+1} + \varepsilon_{i,j} u_{i,j-1} \tag{7}$$

where some of the α, β, γ, δ and ε may be zero according to the position of the point relative to the grid. Note that only the neighbouring points in the vertical and horizontal directions are needed to compute $v_{i,j}$.

Figure 3: Matrix-vector product, dense storage format: A) Partitioning in columns , B) Example and C) Computation and communication steps.

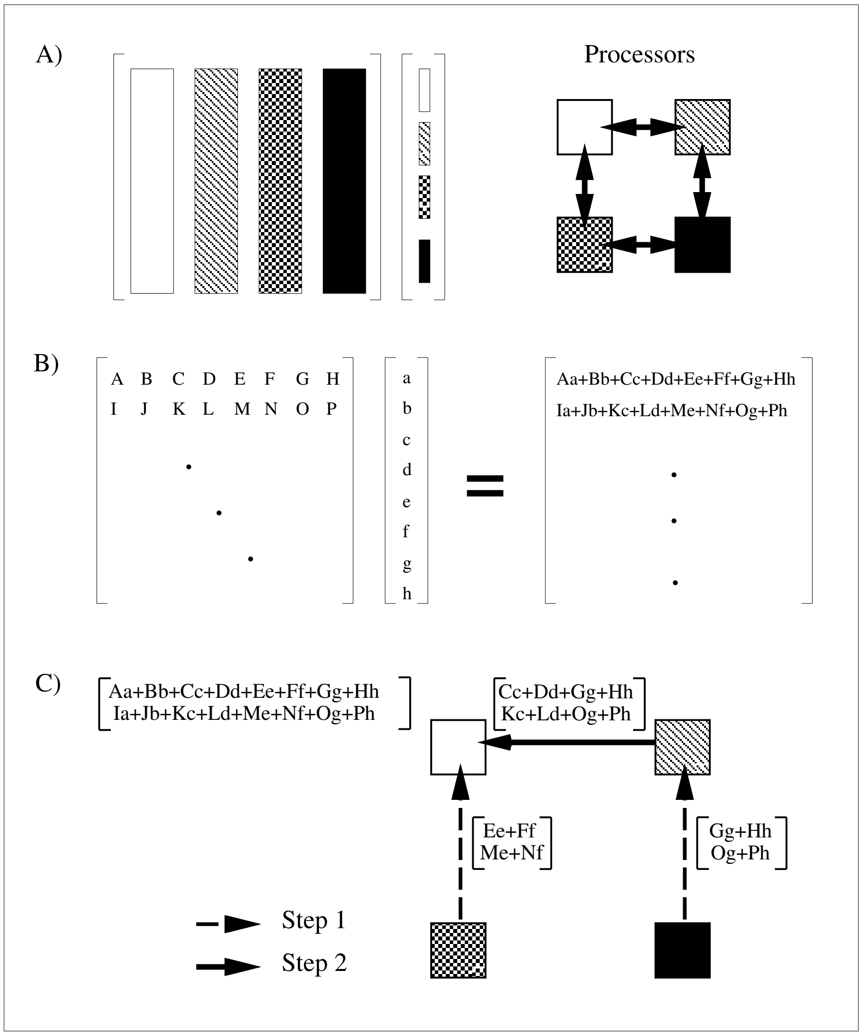
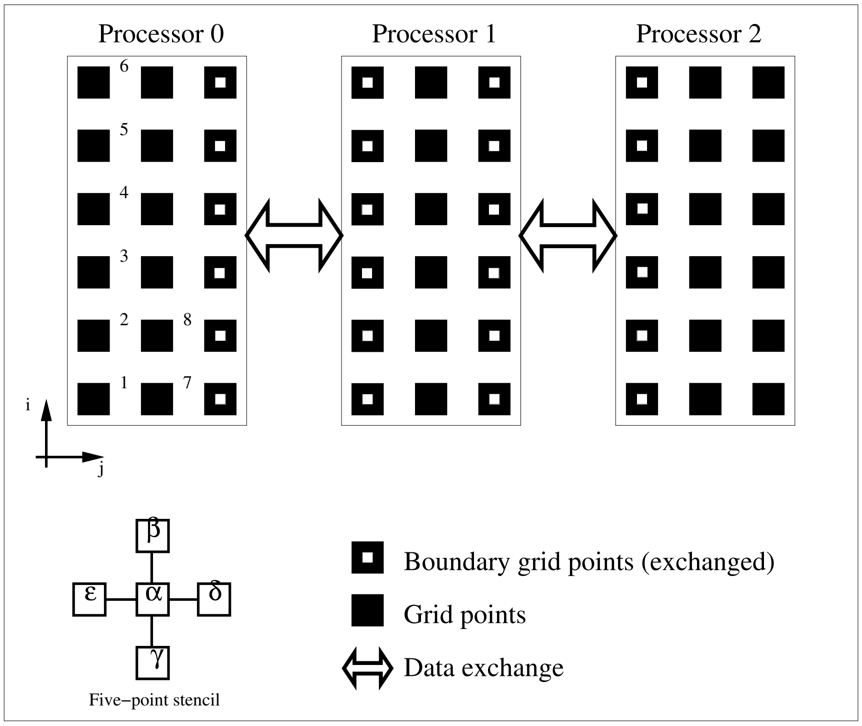


Figure 4: Matrix-vector product, PDE storage format.



A parallel computation of (7) may be organised as follows. The grid points are partitioned by vertical panels among the processors as shown in Figure 4. A processor holds at most $\lceil l/p \rceil$ columns of l grid points. To compute the matrix-vector product, each processor exchanges with its neighbours the grid points in the "interfaces" between the processors (the points marked with white squares in Figure 4). Equation (7) is then applied independently by each processor at its local grid points, except at the local interfacing points. After the interfacing grid points from the neighbouring processors have arrived at a processor, Equation (7) is applied using the local interfacing points and those from the neighbouring processors.

This parallel computation offers the possibility of overlapping communication with the computation. If the number of local grid points is large enough, one may expect that while Equation (7) is being applied to those points, the interfacing grid points of the neighbouring processors will have been transferred and be available for use. This method attempts to minimize the overheads incurred by transferring the data (note that we only make gains if the *asynchronous* transfer of messages is available). The example below is taken from the matrix-vector product routine using MPI

```
      SUBROUTINE PDMVPDE(NPROCS,MYID,LDC,L,MYL,COEFS,U,V,UEAST,UWEST)
      INCLUDE 'mpif.h'
* Declarations...

* Send border U values to (myid+1)-th processor
      MSGTYPE = 1000
      TO = MYID + 1
      CALL MPI_ISEND(U(EIO),L,MPI_DOUBLE_PRECISION,TO,MSGTYPE,
     +               MPI_COMM_WORLD,SID0,IERR)

* Post to receive border U values from (myid+1)-th processor
      MSGTYPE = 1001
      CALL MPI_IRECV(UEAST,L,MPI_DOUBLE_PRECISION,MPI_ANY_SOURCE,
     +               MSGTYPE,MPI_COMM_WORLD,RID0,IERR)

* Send border U values to (myid-1)-th processor
      MSGTYPE = 1001
      TO = MYID - 1
      CALL MPI_ISEND(U(WIO),L,MPI_DOUBLE_PRECISION,TO,MSGTYPE,
     +               MPI_COMM_WORLD,SID1,IERR)

* Post to receive border U values from (myid-1)-th processor
      MSGTYPE = 1000
      CALL MPI_IRECV(UWEST,L,MPI_DOUBLE_PRECISION,MPI_ANY_SOURCE,
     +               MSGTYPE,MPI_COMM_WORLD,RID1,IERR)

* Compute with local grid points...

* Need "eastern" data,wait for completion of receive
      CALL MPI_WAIT(RID0,ISTAT,IERR)

* Compute with local interfacing grid points in the "east"...

* Need "west" data,wait for completion of receive
      CALL MPI_WAIT(RID1,ISTAT,IERR)

* Compute with local interfacing grid points in the "west"...

* Release message IDs
      CALL MPI_WAIT(SID0,ISTAT,IERR)
      CALL MPI_WAIT(SID1,ISTAT,IERR)

      RETURN
      END
```

The computation of the transpose-matrix-vector product for the PDE case is performed in a similar fashion. Before the computation starts, each processor exchanges with its left and right neighbouring processors the *east* and *west* coefficients corresponding to the interfacing grid points[4]. The computation performed is then similar to the matrix-vector product described above except that for each interfacing grid point we apply

$$v_{i,j} = \alpha_{i,j}u_{i,j} + \gamma_{i+1,j}u_{i+1,j} + \beta_{i-1,j}u_{i-1,j} + \varepsilon_{i,j+1}u_{i,j+1} + \delta_{i,j-1}u_{i,j-1} \tag{8}$$

Comparing (8) to (7) we see that the coefficients are swapped in the north-south and east-west directions. Note that due to the partitioning imposed we do not need to exchange the north and south coefficients.

A matrix-vector product for parallel vector architectures For parallel vector architectures like the Cray Y-MP2E, the routines outlined above are not efficient, because of the small vector lengths involved. A routine requiring the use of long vectors may be obtained by writing the matrix-vector product for the 5-point stencil as a sequence of `_AXPY`s. The use of `_AXPY`s also provides a better performance because these operations are usually very efficient on such machines.

Consider the storage scheme described above i.e., five coefficients (α, β, γ, δ and ε) are stored per grid point, and numbered sequentially as $i + (j-1)l$, $i, j = 1, 2, \ldots, l$. The coefficients can then be stored in five separate arrays of size $n = l^2$. The matrix-vector product $v = Au$ can then be obtained by the following sequence of operations

$$\begin{aligned}
v_k &= \alpha_k u_k, \quad k = 1, 2, \ldots, n && (9)\\
v_k &= v_k + \beta_k u_{k+1}, \quad k = 1, 2, \ldots, n-1 && (10)\\
v_k &= v_k + \gamma_k u_{k-1}, \quad k = 2, 3, \ldots, n && (11)\\
v_k &= v_k + \delta_k u_{k+l}, \quad k = 1, 2, \ldots, n-l && (12)\\
v_k &= v_k + \varepsilon_k u_{k-l}, \quad k = l+1, l+2, \ldots, n && (13)
\end{aligned}$$

and the transpose matrix-vector product $v = A^T u$ is obtained similarly,

$$\begin{aligned}
v_k &= \alpha_k u_k, \quad k = 1, 2, \ldots, n && (14)\\
v_{k+1} &= v_{k+1} + \beta_k u_k, \quad k = 1, 2, \ldots, n-1 && (15)\\
v_{k-1} &= v_{k-1} + \gamma_k u_k, \quad k = 2, 3, \ldots, n && (16)\\
v_{k+l} &= v_{k+l} + \delta_k u_k, \quad k = 1, 2, \ldots, n-l && (17)\\
v_{k-l} &= v_{k-l} + \varepsilon_k u_k, \quad k = l+1, l+2, \ldots, n && (18)
\end{aligned}$$

Experiments on the Cray Y-MP2E/232 showed that this approach gave a three-fold improvement in the performance, from 40MFLOPS to 140MFLOPS. A separate set of the PDE examples containing these matrix-vector product routines are provided under the `pim20/examples/sequential/pvp-pde` directory.

[4]This may only need to be done once if the coefficient matrix is unchanged during the solution of the system.

4.6.4 Preconditioners

The examples involving an incomplete LU factorisation as the preconditioner for the PDE case are a modification of the usual $ILU(0)$ method. This modification was made to allow the computation of the preconditioning step without requiring any communication to be performed. To achieve this we note that the matrices arising from the five-point finite-difference discretisation have the following structure

$$A = \begin{bmatrix} B & E & & \\ F & B & \ddots & \\ & \ddots & \ddots & E \\ & & F & B \end{bmatrix}, \quad B = \begin{bmatrix} \alpha & \beta & & \\ \gamma & \alpha & \ddots & \\ & \ddots & \ddots & \beta \\ & & \gamma & \alpha \end{bmatrix} \tag{19}$$

where E and F are diagonal matrices and α, β and γ are the *central*, *north* and *south* coefficients derived from the discretisation (the subscripts are dropped for clarity). Each matrix B is approximating the unknowns in a single vertical line on the grid on Figure 4.

To compute a preconditioner $Q = LU$, we modify the $ILU(0)$ algorithm in the sense that the blocks E and F are discarded (because only the diagonal blocks are considered we refer to this factorisation as $IDLU(0)$). The resulting L and U factors have the following structure

$$\begin{aligned} L &= \begin{bmatrix} X & & & \\ & X & & \\ & & \ddots & \\ & & & X \end{bmatrix}, \quad X = \begin{bmatrix} 1 & & & \\ \check{\gamma} & 1 & & \\ & \ddots & \ddots & \\ & & \check{\gamma} & 1 \end{bmatrix}, \\ U &= \begin{bmatrix} Y & & & \\ & Y & & \\ & & \ddots & \\ & & & Y \end{bmatrix}, \quad Y = \begin{bmatrix} \check{\alpha} & \check{\beta} & & \\ & \check{\alpha} & \ddots & \\ & & \ddots & \check{\beta} \\ & & & \check{\alpha} \end{bmatrix} \end{aligned} \tag{20}$$

where $\check{\alpha}$, $\check{\beta}$ and $\check{\gamma}$ are the modified coefficients arising from the $ILU(0)$ algorithm. ¿From the structure of L and U it may be clearly seen that applying the preconditioning step reduces to the solution of small (order l), independent, triangular systems. Each of these systems correspond to a vertical line in the grid; since it was partitioned in vertical panels, these systems can be solved independently in each processor.

The polynomial preconditioners used can be expressed by

$$\left(\sum_{i=0}^{m} \gamma_{m,i} \left(I - (diag(A))^{-1} A \right)^{i} \right) (diag(A))^{-1} \tag{21}$$

which can be easily computed as a sequence of vector updates and matrix-vector products using Horner's algorithm. Note that the $\gamma_{m,i}$ coefficients define the kind of polynomial preconditioner being used. The Neumann preconditioner is obtained when $\gamma_{m,i} = 1, \forall i$; the weighted and unweighted least-squares polynomial preconditioners are those reported in [36]. The maximum available degree of the polynomial for these latter two is $m = 13$.

4.6.5 Results

In this section we provide some results for the example programs discussed above.

Stopping criteria As pointed out earlier, the selection of the stopping criterion has a substantial effect on the execution time. Evidently, there is a trade-off between the time spent on each iteration and the total number of iterations required for convergence. In Table 3 we show, for each of the stopping criteria provided, the execution time per iteration when `PIMDCG` is applied to the tridiagonal system (described in §4.6) of order $n = 500$ with diagonal left-preconditioning. The increase in execution time of each stopping criterion with respect to criterion **4** (the "cheapest" one) is shown.

Table 3: Effect of different stopping criteria on an iterative method routine.

Stopping criterion	k^*	$\|\| r_{k^*} \|\|_2$	Time(s)/ iteration	Increase
1	19	3.56×10^{-11}	0.3331	2.66
2	15	6.91×10^{-9}	0.3531	2.79
3	14	1.29×10^{-8}	0.3286	2.62
4	18	3.32×10^{-11}	0.1254	$--$
5	14	6.45×10^{-9}	0.1967	1.57
6	15	1.73×10^{-9}	0.2904	2.32
7	19	3.70×10^{-11}	0.4148	3.31

General results We present below the results obtained from solving a system of $n = 64$ equations derived from the 5-point finite-differences discretisation of Equation (6).

We used both the $IDLU(0)$ and the Neumann polynomial preconditioner of degree 1 as *left*-preconditioners to solve this problem. The stopping criterion used was number **5** with $\varepsilon = 10^{-10}$ and the 2-norm; using this criterion a solution will be accepted if $\| z_k \|_2 < 3.802 \times 10^{-14}$, except for `PIMDRGMRES` which stops its iterations when the norm of the residual is less than ε. The maximum number of iterations allowed was 32 and the initial value of the solution vector was $(0, 0, \ldots, 0)^T$. For the restarted GMRES and GCR the restarting value used was 10. The results are reported for the double-precision versions of the routines.

Tables 4 and 5 show the results obtained with the PIM routines for the $IDLU(0)$ and Neumann preconditioners on a single workstation. A status value of 0 on exit from a PIM routine indicates that the convergence conditions have been satisfied; a non-zero status value indicates that a problem has been encountered. In particular a status value of -1 is returned when the maximum number of iterations specified by the user has been exceeded. This example is characteristic of the problems facing the user of iterative method i.e., not all methods converge to the solution and some preconditioner may cause an iterative method to diverge (or converge slowly). We stress that the methods that have failed to converge in this example do converge for other systems.

Scalability In Table 6 we present the execution times obtained by solving the test problem above, but with $n = 16384$ equations, with the `PIMDRGMRES` routine (using 10 basis vectors) and the Neumann polynomial preconditioner (of first degreee) on the IBM SP/1, Intel Paragon XP/S, Kendall Square Research KSR1, SGI Challenge, Cray Y-MP2E and Cray C9016E. The `PIMDRGMRES` routine converged to a tolerance of 10^{-13} in 70 iterations. The results for the Cray machines were obtained with the modified matrix-vector product routines described in §4.6.3. The results for the KSR1 are obtained using the KSRBLAS routines. The programs running on the SGI Challenge are from the set of examples available with the PIM distributed software using the PVM message-passing library. The results for the IBM SP/1 are obtained using the IBM PVMe 2.0 version of PVM, which enables the use of the IBM SP/1 High Performance Switch. Note that for the IBM SP/1, SGI Challenge and Intel Paragon XP/S superlinear effects occur; we believe this is due to the specific memory organization of those machines (hierarchic memories and/or the presence of a cache memory).

5 Summary

We have described in this report PIM, the Parallel Iterative Methods package, a collection of Fortran 77 routines for the parallel solution of linear systems using iterative methods.

The package was designed to be used in a variety of parallel environments without imposing any restriction on the way the coefficient matrices and the preconditioning steps are handled. The user may thus explore characteristics of the problem and of the particular parallel architec-

Table 4: Example with $IDLU(0)$ preconditioner.

Method	k^*	Time(s)	$\|\|r^{(k^*)}\|\|_2$	Status
CG	32	0.0900	59.8747	-1
CGEV	32	0.1500	59.8747	-1
Bi-CG	32	0.1000	10.8444	-1
CGS	11	0.0500	1.8723×10^{-11}	0
Bi-CGSTAB	12	0.0400	1.1099×10^{-11}	0
RBi-CGSTAB	7	0.0300	3.7795×10^{-12}	0
RGMRES	3	0.0600	2.8045×10^{-12}	0
RGMRESEV	3	0.4500	2.8045×10^{-12}	0
RGCR	3	0.0800	2.8048×10^{-12}	0
CGNR	32	0.0800	81.5539	-1
CGNE	32	0.0900	37.0570	-1
QMR	32	0.1200	1.7739	-1
TFQMR	11	0.0500	5.3482×10^{-11}	0

Table 5: Example with Neumann polynomial preconditioner.

Method	k^*	Time(s)	$\|\|r^{(k^*)}\|\|_2$	Status
CG	32	0.1000	41.9996	-1
CGEV	32	0.1300	41.9996	-1
Bi-CG	32	0.1000	13.8688	-1
CGS	14	0.0500	7.4345×10^{-11}	0
Bi-CGSTAB	14	0.0500	5.3863×10^{-12}	0
RBi-CGSTAB	8	0.0500	8.4331×10^{-14}	0
RGMRES	3	0.0600	4.0454×10^{-11}	0
RGMRESEV	3	0.3900	4.0454×10^{-11}	0
RGCR	3	0.0900	4.0455×10^{-11}	0
CGNR	32	0.0800	7.4844×10^{-1}	-1
CGNE	32	0.1000	3.6205×10^{2}	-1
QMR	32	0.1600	1.1474	-1
TFQMR	14	0.1100	7.6035×10^{-11}	0

Table 6: Execution time (in seconds) for test problem ($n = 16384$) solved by `PIMDRGMRES` with the Neumann polynomial preconditioner (of first degreee).

p	IBM SP/1	Intel Paragon XP/S	Intel iPSC/860*	SGI Challenge†	KSR1‡	Cray T3D$^\diamond$	Cray Y-MP2E	Cray C9016
1	150.42	265.83		99.60	453.20			
2	39.60	131.64	80.90		297.40		11.64	
4	20.43	60.54	46.95	26.73				4.99
8	7.10	29.82	31.62		166.80			
16		16.35	35.38			4.04		
32		11.20						

* W.H. Purvis, Daresbury Laboratory, U.K.
† S. Thomas, CERCA/Montréal
‡ A. Pindor, U. of Toronto [39]
$^\diamond$ P.T.M. Bulhões, Cray Research Inc.

ture being used. Indeed, the performance of a PIM routine is dependent on the user-supplied routines for the matrix-vector products, inner-products and vector norms and the computation of the preconditioning steps.

PIM is an ongoing project and we intend to improve it and include other iterative methods. We encourage readers to send their comments and suggestions; the authors may be contacted via e-mail at either *rudnei@mat.ufrgs.br* or *trh@ukc.ac.uk* .

Acknowledgements

We would like to thank the *National Supercomputing Centre (CESUP), Brazil*, the *National Laboratory for Scientific Computing (LNCC/CNPq), Brazil*, the *Parallel Laboratory, University in Bergen, Norway*, the *Army High Performance Computing Research Center, Minnesota, USA* and *Digital Equipment Corporation* (via the Internet Alpha Program), who kindly made their facilities available for our tests.

We also thank our collaborators, Matthias G. Imhof (MIT), Paulo Tibério M. Bulhões (Cray Research), Steve Thomas (CERCA/Montréal), Andrzej Pindor (University of Toronto), William H. Purvis (Daresbury Laboratory, U.K.) and Ramiro B. Willmersdorf (LNCC, Brazil) for their help on testing PIM.

This work was supported in part by the Army High Performance Computing Research

Center, under the auspices of Army Research Office contract number DAAL03-89-C-0038 with the University of Minnesota.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1992.

[2] S.F. Ashby. Minimax polynomial preconditioning for Hermitian linear systems. Report UCRL-JC-103201, Numerical Mathematics Group, Computing & Mathematics Research Division, Lawrence Livermore National Laboratory, March 1990.

[3] S.F. Ashby, T.A. Manteuffel, and J.S. Otto. A comparison of adaptive Chebyshev and least squares polynomial preconditioning for Hermitian positive definite linear systems. Report UCRL-JC-106726, Numerical Mathematics Group, Computing & Mathematics Research Division, Lawrence Livermore National Laboratory, March 1991.

[4] S.F. Ashby, T.A. Manteuffel, and P.E. Saylor. A taxonomy for Conjugate Gradient methods. *SIAM Journal of Numerical Analysis*, 27:1542–1568, 1990.

[5] S.F. Ashby and M.K. Seager. A proposed standard for iterative linear solvers (version 1.0). Report UCRL-102860, Numerical Mathematics Group, Computing & Mathematics Research Division, Lawrence Livermore National Laboratory, January 1990.

[6] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donald, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, Philadelphia, 1993.

[7] R.A.A. Bruce, J.G. Mills, and G.A. Smith. CHIMP/MPI user guide. EPCC-KTP-CHIMP-V2-USER 1.2, Edinburgh Parallel Computer Centre, University of Edinburgh, June 1994.

[8] R. Butler and E. Lusk. User's guide to the p4 programming system. ANL-92/17, Argonne National Laboratory, October 1992.

[9] F.-C. Cheng, P. Vaughan, D. Reese, and A. Skjellum. The Unify system. User's guide, document for version 0.9.2, NSF Engineering Research Center, Mississippi State University, September 1994.

[10] E.J. Craig. The N-step iteration procedures. *Journal of Mathematical Physics*, 34:64–73, 1955.

[11] R.D. da Cunha. *A Study on Iterative Methods for the Solution of Systems of Linear Equations on Transputer Networks.* PhD thesis, Computing Laboratory, University of Kent at Canterbury, July 1992.

[12] R.D. da Cunha and T.R. Hopkins. Parallel preconditioned Conjugate-Gradients methods on transputer networks. *Transputer Communications*, 1(2):111–125, 1993. Also as TR-5-93, Computing Laboratory, University of Kent at Canterbury, U.K.

[13] R.D. da Cunha and T.R. Hopkins. A parallel implementation of the restarted GMRES iterative method for nonsymmetric systems of linear equations. *Advances in Computational Mathematics*, 2(3):261–277, April 1994. Also as TR-7-93, Computing Laboratory, University of Kent at Canterbury.

[14] E.F. D'Azevedo and C.H. Romine. Reducing communication costs in the Conjugate Gradient algorithm on distributed memory multiprocessors. Research Report ORNL/TM-12192, Oak Ridge National Laboratory, 1992.

[15] J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, 1988.

[16] J.J. Dongarra, R.A. van de Geijn, and R.C. Whaley. A users' guide to the BLACS v0.1. Technical report, Computer Science Department, University of Tennessee, 1993.

[17] I.S. Duff, R.G. Grimes, and J.G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection. Report TR/PA/92/86, CERFACS, October 1992.

[18] I.S. Duff, M. Marrone, and G. Radicati. A proposal for user level sparse BLAS – SPARKER working note #1. Report TR/PA/92/85, CERFACS, October 1992.

[19] S.C. Eisenstat. A note on the generalized Conjugate Gradient method. *SIAM Journal of Numerical Analysis*, 20:358–361, 1983.

[20] H.C. Elman, Y. Saad, and P. Saylor. A hybrid Chebyshev Krylov subspace algorithm for solving nonsymmetric systems of linear equations. *SIAM Journal of Scientific and Statistical Computing*, 7:840–855, 1986.

[21] R. Fletcher. *Conjugate Gradient Methods for Indefinite Systems*, volume 506 of *Lecture Notes in Mathematics*, pages 73–89. Spring-Verlag, Heidelberg, 1976.

[22] Message Passing Interface Forum. MPI: A message-passing interface standard. TR CS-93-214, University of Tennessee, November 1993.

[23] R.W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. Submitted to SIAM Journal of Scientific and Statistical Computing.

[24] R.W. Freund. Implementation details of the coupled QMR algorithm. Numerical Analysis Manuscript 92-12, AT&T Bell Laboratories, October 1992.

[25] R.W. Freund, G.H. Golub, and N.M. Nachtigal. Iterative solution of linear systems. *Acta Numerica*, 1:57–100, 1991.

[26] R.W. Freund, G.H. Golub, and N.M. Nachtigal. Recent advances in Lanczos-based iterative methods for nonsymmetric linear systems. RIACS Technical Report 92.02, Research Institute for Advanced Computer Science, NASA Ames Research Center, 1992. To appear on *Algorithmic trends for Computational Fluid Dynamics in the 90's*.

[27] R.W. Freund and M. Hochbruck. A Biconjugate Gradient-type algorithm for the iterative solution of non-Hermitian linear systems on massively parallel architectures. RIACS Technical Report 92.08, Research Institute for Advanced Computer Science, NASA Ames Research Center, 1992. To appear on *Computational and Applied Mathematics I-Algorithms and Theory*.

[28] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V.S. Sunderam. PVM 3 user's guide and reference manual. Research Report ORNL/TM-12187, Oak Ridge National Laboratory, May 1993.

[29] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[30] R.G. Grimes, D.R. Kincaid, and D.M. Young. ITPACK 2.0 user's guide. Report No. CNA-150, Center for Numerical Analysis, University of Texas at Austin, August 1979.

[31] W. Gropp and B. Smith. Simplified linear equation solvers users manual. ANL-93/8, Argonne National Laboratory, February 1993.

[32] L.A. Hageman and D.M. Young. *Applied Iterative Methods*. Academic Press, New York, 1981.

[33] R.J. Harrison. TCGMSG Send/receive subroutines – version 4.02. User's manual, Battelle Pacific Northwest Laboratory, January 1993.

[34] M.A. Heroux. A proposal for a sparse BLAS toolkit – SPARKER working note #2. Report TR/PA/92/90, CERFACS, October 1992.

[35] M.R. Hestenes and E.L. Stiefel. Method of Conjugate Gradients for solving linear systems. *Journal of Research National Bureau of Standards*, 49:435–498, 1952.

[36] W.H. Holter, I.M. Navon, and T.C. Oppe. Parallelizable preconditioned Conjugate Gradient methods for the Cray Y-MP and the TMC CM-2. Technical report, Supercomputer Computations Research Institute, Florida State University, December 1991.

[37] N.M. Nachtigal, S.C. Reddy, and L.N. Trefethen. How fast are nonsymmetric matrix iterations. *SIAM Journal on Matrix Analysis and Applications*, 13(3):778–795, 1992.

[38] T.C. Oppe, W.D. Joubert, and D.R. Kincaid. NSPCG user's guide – version 1.0. Report No. CNA-216, Center for Numerical Analysis, University of Texas at Austin, April 1988.

[39] A. Pindor. Experiences with implementing PIM (Parallel Iterative Methods) package on KSR1. In *Supercomputing Symposium '94*, Toronto, June 1994.

[40] Y. Saad. Krylov subspace methods for solving large unsymmetric systems. *Mathematics of Computation*, 37:105–126, 1981.

[41] Y. Saad and M.H. Schultz. Conjugate Gradient-like algorithms for solving nonsymmetric linear systems. *Mathematics of Computation*, 44(170):417–424, 1985.

[42] Y. Saad and M.H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7:856–869, 1986.

[43] G.L.G. Sleijpen and D.R. Fokkema. BiCGSTAB(L) for linear matrices involving unsymmetric matrices with complex spectrum. *ETNA*, 1:11–32, September 1993.

[44] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 10:36–52, 1989.

[45] G. Stellner, S. Lamberts, and T. Ludwig. NXLIB User's Guide. Technical report, Institut für Informatik, Lehrstuhl für Rechnertechnik und Rechnerorganisation, Technische Universität München, October 1993.

[46] H.A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 13:631–644, 1992. Also as Report No. 90-50, Mathematical Institute, University of Utrecht.

A Reference manual

In this section we provide details of each individual subroutine in PIM. Each entry describes the purpose, the name and parameter list, storage requirements, function dependencies and restrictions of the respective routine.

For each iterative method routine we also provide a description of the implemented algorithm. Vectors and scalar values are denoted by lower case letters and matrices by capital letters. Subscripts indicate either the iteration or a vector column, the latter in the case of a matrix.

Each computational step in the algorithm is numbered; if a routine suffers a breakdown the step number where the failure occurred is returned in `IPAR(13)`.

Whenever an underscore _ appears it indicates the type of a variable or function (`REAL`, `DOUBLE PRECISION`, `COMPLEX`, `DOUBLE COMPLEX`) and should be replaced by `S`, `D`, `C` or `Z`, whichever is appropriate.

The `COMPLEX` and `DOUBLE COMPLEX` PIM routines compute inner-products using the BLAS `CDOTC` and `ZDOTC` routines respectively.

A.1 Description of parameters

The parameters used in an iterative method routine are

Parameter	Description
`X`	A vector of length `IPAR(4)` On input, contains the initial estimate On output, contains the last estimate computed
`B`	The right-hand-side vector of length `IPAR(4)`
`WRK`	A work vector used internally (see the description of each routine for its length)
`IPAR`	see below
`_PAR`	see below
`MATVEC`	Matrix-vector product external subroutine
`TMATVEC`	Transpose-matrix-vector product external subroutine
`PRECONL`	Left-preconditioning external subroutine
`PRECONR`	Right-preconditioning external subroutine
`P_SUM`	Inner-product external function
`P_NRM`	Vector norm external function
`PROGRESS`	Monitoring routine

`IPAR` (input)

Element	Description
1	Leading dimension of `A`
2	Number of rows or columns of `A` (depending on partitioning)
3	Block size (for cyclic partitioning)
4	Number of vector elements stored locally
5	Restarting parameter used in GMRES, GCR and RBi-CGSTAB
6	Number of processors
7	Processor identification
8	Preconditioning 0: No preconditioning 1: Left preconditioning 2: Right preconditioning 3: Symmetric preconditioning
9	Stopping criterion (see Table 2)
10	Maximum number of iterations allowed

IPAR (output)

Element	Description
11	Number of iterations
12	Exit status: 0: converged to solution –1: no convergence has been achieved –2: "soft"-breakdown, solution may have been found –3: "hard"-breakdown, no solution –4: conflict in preconditioner and stopping criterion selected; if IPAR(8)=0 or IPAR(8)=2 then IPAR(9)≠6 –5: error in stopping criterion 3, $r_k^T z_k < 0$ –6: stopping criterion invalid on PIM_CHEBYSHEV –7: no estimates of eigenvalues supplied for PIM_CHEBYSHEV –8: underflow while computing μ_1 on PIM_CGEV –9: overflow while computing μ_1 on PIM_CGEV –10: underflow while computing μ_n on PIM_CGEV –11: overflow while computing μ_n on PIM_CGEV
13	If IPAR(12) is either –2 or –3, it gives the step number in the algorithm where a breakdown has occurred

_PAR (input)

Element	Description
1	The value of ε for use in the stopping criterion

_PAR (output)

Element	Description
2	The left-hand side of the stopping criterion selected
3	Minimum real part of the eigenvalues of Q_1AQ_2
4	Maximum real part of the eigenvalues of Q_1AQ_2
5	Minimum imaginary part of the eigenvalues of Q_1AQ_2
6	Maximum imaginary part of the eigenvalues of Q_1AQ_2

A.2 External routines

Purpose

To compute the matrix-vector product, transpose-matrix-vector product, left-preconditioning, right-preconditioning, global sum of a vector, vector norm, and to monitor the progress of the iterations.

Note The coefficient matrix and the preconditioning matrices can be made available to `MATVEC`, `TMATVEC`, `PRECONL` and `PRECONR` using `COMMON` blocks.

Synopsis

Matrix-vector product $v = Au$

```
SUBROUTINE MATVEC(U,V,IPAR)
precision U(*),V(*)
INTEGER IPAR(*)
```

Parameters	Type
U	INPUT
V	OUTPUT
IPAR	INPUT

Transpose matrix-vector product $v = A^T u$

```
SUBROUTINE TMATVEC(U,V,IPAR)
precision U(*),V(*)
INTEGER IPAR(*)
```

Parameters	Type
U	INPUT
V	OUTPUT
IPAR	INPUT

Left preconditioning $v = Qu$

```
SUBROUTINE PRECONL(U,V,IPAR)
precision U(*),V(*)
INTEGER IPAR(*)
```

Parameters	Type
U	INPUT
V	OUTPUT
IPAR	INPUT

Right preconditioning $v = Qu$

```
SUBROUTINE PRECONR(U,V,IPAR)
precision U(*),V(*)
INTEGER IPAR(*)
```

Parameters	Type
U	INPUT
V	OUTPUT
IPAR	INPUT

Synopsis

Parallel sum

```
SUBROUTINE P_SUM(ISIZE,X)
INTEGER ISIZE
precision X(*)
```

Parameters	Type
ISIZE	INPUT
X	INPUT/OUTPUT

Parallel vector norm

```
precision FUNCTION P_NRM(LOCLEN,U)
INTEGER LOCLEN
precision U(*)
```

Parameters	Type
LOCLEN	INPUT
U	INPUT

Monitoring routine

```
 SUBROUTINE PROGRESS(LOCLEN,ITNO,
+                    NORM,X,RES,
+                    TRUERES)
 INTEGER LOCLEN,ITNO
 precision NORM,X(*),RES(*),
+        TRUERES(*)
```

Parameters	Type
LOCLEN	INPUT
ITNO	INPUT
NORM	INPUT
X	INPUT
RES	INPUT
TRUERES	INPUT

Notes

1. Replace *precision* by `REAL`, `DOUBLE PRECISION`, `COMPLEX`, `DOUBLE COMPLEX`.
2. In the monitoring routine `PROGRESS` above, the array `TRUERES` contains the value of the true residual $r_k = b - Ax_k$ only if `IPAR(9)` is 1, 2 or 3.

A.3 PIM_CG

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the CG method.

Synopsis

```
PIMSCG(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDCG(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCCG(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZCG(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	6*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

None

Algorithm A.1 *CG*

1. $r_0 = Q_1(b - AQ_2x_0)$
2. $p_0 = r_0$
3. $\varrho_0 = r_0^T r_0$
4. $w_0 = Q_1AQ_2p_0$
5. $\xi_0 = p_0^T w_0$

   for $k = 1, 2, \ldots$

6. $\alpha_{k-1} = \varrho_{k-1}/\xi_{k-1}$
7. $x_k = x_{k-1} + \alpha_{k-1}p_{k-1}$
8. $r_k = r_{k-1} - \alpha_{k-1}w_{k-1}$
9. *check stopping criterion*
10. $s_k = Q_1AQ_2r_k$
11. $\varrho_k = r_k^T r_k$
12. $\delta_k = r_k^T s_k$
13. $\beta_k = \varrho_k/\varrho_{k-1}$
14. $p_k = r_k + \beta_k p_{k-1}$
15. $w_k = s_k + \beta_k w_{k-1}$
16. $\xi_k = \delta_k - \beta_k^2 \xi_{k-1}$

   endfor

A.4 PIM_CGEV

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the CG method; returns, at each iteration, the estimates of the smallest and largest eigenvalues of $Q_1 A Q_2$ derived from the associated Lanczos tridiagonal matrix.

Synopsis

```
PIMSCGEV(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDCGEV(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	6*IPAR(4)+2*IPAR(10)+1
IPAR	13
_PAR	4

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$ $\boxed{-8}$ $\boxed{-9}$ $\boxed{-10}$ $\boxed{-11}$

Function dependencies

BLAS	_COPY, _AXPY, _DOT
LIBPIM	

Notes

- If more accuracy is required in the computation of the estimates of the eigenvalues, the user may modify the value of the maximum number of iterations allowed in the routine BISECTION (files pim20/single/src/pimscgev.f or pim20/double/src/pimdcgev.f).
- Not available in COMPLEX or DOUBLE COMPLEX versions.

Algorithm A.2 *CGEV*

1. $r_0 = Q_1(b - AQ_2x_0)$
2. $p_0 = r_0$
3. $\varrho_0 = r_0^T r_0$
4. $w_0 = Q_1AQ_2p_0$
5. $\xi_0 = p_0^T w_0$

   for $k = 1, 2, \ldots$

6. $\alpha_{k-1} = \varrho_{k-1}/\xi_{k-1}$
7. $x_k = x_{k-1} + \alpha_{k-1}p_{k-1}$
8. $r_k = r_{k-1} - \alpha_{k-1}w_{k-1}$
9. *check stopping criterion*
10. $s_k = Q_1AQ_2r_k$
11. $\varrho_k = r_k^T r_k$
12. $\delta_k = r_k^T s_k$
13. $\beta_k = \varrho_k/\varrho_{k-1}$
14. $p_k = r_k + \beta_k p_{k-1}$
15. $w_k = s_k + \beta_k w_{k-1}$
16. $\xi_k = \delta_k - \beta_k^2 \xi_{k-1}$
17. *compute estimates of eigenvalues*

   endfor

A.5 PIM_CGNR

Purpose

Solves the system $Q_1 A^T A Q_2 x = Q_1 A^T b$ using the CGNR method.

Synopsis

```
PIMSCGNR(X,B,WRK,IPAR,SPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDCGNR(X,B,WRK,IPAR,DPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCCGNR(X,B,WRK,IPAR,SPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZCGNR(X,B,WRK,IPAR,DPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	6*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

None

Algorithm A.3 *CGNR*

1. $r_0 = Q_1(A^T b - A^T A Q_2 x_0)$
2. $p_0 = r_0$
3. $\varrho_0 = r_0^T r_0$
4. $w_0 = Q_1 A^T A Q_2 p_0$
5. $\xi_0 = p_0^T w_0$

for $k = 1, 2, \ldots$

6. $\alpha_{k-1} = \varrho_{k-1}/\xi_{k-1}$
7. $x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$
8. $r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$
9. *check stopping criterion*
10. $s_k = Q_1 A^T A Q_2 r_k$
11. $\varrho_k = r_k^T r_k$
12. $\delta_k = r_k^T s_k$
13. $\beta_k = \varrho_k / \varrho_{k-1}$
14. $p_k = r_k + \beta_k p_{k-1}$
15. $w_k = s_k + \beta_k w_{k-1}$
16. $\xi_k = \delta_k - \beta_k^2 \xi_{k-1}$

endfor

A.6 PIM_CGNE

Purpose

Solves the system $Q_1 A A^T Q_2 x = Q_1 b$ using the CGNE method.

Synopsis

```
PIMSCGNE(X,B,WRK,IPAR,SPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDCGNE(X,B,WRK,IPAR,DPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCCGNE(X,B,WRK,IPAR,SPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZCGNE(X,B,WRK,IPAR,DPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	6*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

None

Algorithm A.4 *CGNE*

1. $r_0 = Q_1(b - AA^T Q_2 x_0)$
2. $p_0 = r_0$
3. $\varrho_0 = r_0^T r_0$
4. $w_0 = Q_1 AA^T Q_2 p_0$
5. $\xi_0 = p_0^T w_0$

   for $k = 1, 2, \ldots$

6. $\alpha_{k-1} = \varrho_{k-1}/\xi_{k-1}$
7. $x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$
8. $r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$
9. *check stopping criterion*
10. $s_k = Q_1 AA^T Q_2 r_k$
11. $\varrho_k = r_k^T r_k$
12. $\delta_k = r_k^T s_k$
13. $\beta_k = \varrho_k / \varrho_{k-1}$
14. $p_k = r_k + \beta_k p_{k-1}$
15. $w_k = s_k + \beta_k w_{k-1}$
16. $\xi_k = \delta_k - \beta_k^2 \xi_{k-1}$

   endfor

A.7 PIM_BICG

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the Bi-CG method.

Synopsis

```
PIMSBICG(X,B,WRK,IPAR,SPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDBICG(X,B,WRK,IPAR,DPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCBICG(X,B,WRK,IPAR,SPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZBICG(X,B,WRK,IPAR,DPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	8*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

None

Algorithm A.5 *Bi-CG*

1. $r_0 = Q_1(b - AQ_2x_0)$
2. $\tilde{r}_0 = \tilde{p}_0 = p_0 = r_0$
3. $\rho_0 = \tilde{r}_0^T r_0$
4. $w_0 = Q_1AQ_2p_0$
5. $\xi_0 = \tilde{p}_0^T w_0$

   for $k = 1, 2, \ldots$

6. $\alpha_{k-1} = \rho_{k-1}/\xi_{k-1}$
7. $x_k = x_{k-1} + \alpha_{k-1}p_{k-1}$
8. $r_k = r_{k-1} - \alpha_{k-1}w_{k-1}$
9. *check stopping criterion*
10. $\tilde{r}_k = \tilde{r}_{k-1} - \alpha_{k-1}Q_1A^TQ_2\tilde{p}_{k-1}$
11. $s_k = Q_1AQ_2r_k$
12. $\rho_k = \tilde{r}_k^T r_k$
13. $\delta_k = \tilde{r}_k^T s_k$
14. $\beta_k = \rho_k/\rho_{k-1}$
15. $p_k = r_k + \beta_k p_k$
16. $\tilde{p}_k = \tilde{r}_k + \beta_k \tilde{p}_k$
17. $w_k = s_k + \beta_k w_k$
18. $\xi_k = \delta_k - \beta_k^2 \xi_{k-1}$

   endfor

A.8 PIM_CGS

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the CGS method.

Synopsis

```
PIMSCGS(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDCGS(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCCGS(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZCGS(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	10*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

None

Algorithm A.6 *CGS*

1. $r_0 = Q_1(b - AQ_2x_0)$
2. $p_0 = s_0 = \tilde{r}_0 = r_0$
3. $\rho_0 = \tilde{r}_0^T r_0$

   for $k = 1, 2, \ldots$

4. $w_{k-1} = Q_1AQ_2p_{k-1}$
5. $\xi_{k-1} = \tilde{r}_0^T w_{k-1}$
6. $\alpha_{k-1} = \rho_{k-1}/\xi_{k-1}$
7. $t_{k-1} = s_{k-1} - \alpha_{k-1}w_{k-1}$
8. $w_{k-1} = s_{k-1} + t_{k-1}$
9. $x_k = x_{k-1} + \alpha_{k-1}w_{k-1}$
10. $r_k = r_{k-1} - \alpha_{k-1}Q_1AQ_2w_{k-1}$
11. *check stopping criterion*
12. $\rho_k = \tilde{r}_0^T r_k$
13. $\beta_k = \rho_k/\rho_{k-1}$
14. $s_k = r_k + \beta_k t_{k-1}$
15. $w_k = t_{k-1} + \beta_k p_{k-1}$
16. $p_k = s_k + \beta_k w_k$

   endfor

A.9 PIM_BICGSTAB

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the Bi-CGSTAB method.

Synopsis

```
PIMSBICGSTAB(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDBICGSTAB(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCBICGSTAB(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZBICGSTAB(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	10*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-2}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

None

Algorithm A.7 *Bi-CGSTAB*

1. $r_0 = Q_1(b - AQ_2x_0)$
2. $\tilde{r}_0 = r_0$
3. $p_0 = v_0 = 0$
4. $\rho_0 = \alpha_0 = \omega_0 = 1$

for $k = 1, 2, \ldots$

5. $\rho_k = \tilde{r}_{k-1}^T r_{k-1}$
6. $\beta_k = \rho_k \alpha_{k-1} / (\rho_{k-1} \omega_{k-1})$
7. $p_k = r_{k-1} + \beta_k (p_{k-1} - \omega_{k-1} v_{k-1})$
8. $v_k = Q_1 A Q_2 p_k$
9. $\xi_k = \tilde{r}_0^T v_k$
10. $\alpha_k = \rho_k / \xi_k$
11. $s_k = r_{k-1} - \alpha_k v_k$
12. *if* $||\, s \,|| <$ `macheps` *soft-breakdown has occurred*
13. $t_k = Q_1 A Q_2 s_k$
14. $\omega_k = t_k^T s_k / t_k^T t_k$
15. $x_k = x_{k-1} + \alpha_k p_k + \omega_k s_k$
16. $r_k = s_k - \omega_k t_k$
17. *check stopping criterion*

endfor

A.10 PIM_RBICGSTAB

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the restarted Bi-CGSTAB method.

Synopsis

```
PIMSRBICGSTAB(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDRBICGSTAB(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCRBICGSTAB(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZRBICGSTAB(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	(6+2*IPAR(5))*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

1. The degree of the MR polynomial (the maximum degree is 10) must be stored in IPAR(5). If the user needs to use a larger degree then the parameter IBDIM, defined on PIM_RBICGSTAB must be changed accordingly.

Algorithm A.8 *RBi-CGSTAB*

1. $r = Q_1(b - AQ_2x)$
2. $\tilde{r} = r$
3. $u_0 = 0$
4. $\rho_0 = 1, \alpha = 0, \omega = 1$

for $k = 1, 2, \ldots$

5. $\rho_0 = -\omega\rho_0$

for $j = 0, 1, \ldots \text{restart} - 1$

6. $\rho_1 = r_j^T \tilde{r}$
7. $\beta = \alpha\rho_1/\rho_0$
8. $\rho_0 = \rho_1$
9. $u_i = r_i - \beta u_i, \quad i = 0, \ldots, j$
10. $u_{j+1} = Q_1AQ_2u_j$
11. $\xi = u_{j+1}^T \tilde{r}$
12. $\alpha = \rho_0/\xi$
13. $r_i = r_i - \alpha u_{i+1}, \quad i = 0, \ldots, j$
14. $r_{j+1} = Q_1AQ_2r_j$
15. $x_0 = x_0 + \alpha u_0$

endfor

16. *check stopping criterion*
17. $\sigma_1 = r_1^T r_1, \gamma_1' = r_0^T r_1/\sigma_1$

for $j = 2, 3, \ldots \text{restart}$

18. $\tau_{i,j} = r_j^T r_i/\sigma_i, r_j = r_j - \tau_{i,j}r_i$
19. $\sigma_j = r_j^T r_j, \gamma_j' = r_0^T r_j/\sigma_j$

endfor

20. $\gamma_{\text{restart}} = \omega = \gamma'_{\text{restart}},$
$\gamma_j = \gamma_j' - \sum_{i=j+1}^{\text{restart}} \tau_{j,i}\gamma_i, \quad j = \text{restart} - 1, \ldots, 1$
21. $\gamma'' = \gamma_{j+1} + \sum_{i=j+1}^{\text{restart}-1} \tau_{j,i}\gamma_{i+1}, \quad j = 1, \ldots, \text{restart} - 1$
22. $x_0 = x_0 + \gamma_1 r_0$
23. $r_0 = r_0 - \gamma'_{\text{restart}} r_{\text{restart}}$
24. $u_0 = u_0 - \gamma_{\text{restart}} u_{\text{restart}}$
25. $u_0 = u_0 - \gamma_j u_j, \quad j = 1, \ldots \text{restart} - 1$
26. $x_0 = x_0 + \gamma_j'' r_j, \quad j = 1, \ldots \text{restart} - 1$
27. $r_0 = r_0 - \gamma_j' r_j, \quad j = 1, \ldots \text{restart} - 1$

endfor

A.11 PIM_RGMRES

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the restarted GMRES method.

Synopsis

```
PIMSRGMRES(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDRGMRES(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCRGMRES(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZRGMRES(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	(4+IPAR(5))*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-2}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC), _SCAL, _TRSV
LIBPIM	

Notes

1. The size of the orthonormal basis (maximum of 50 vectors) must be stored in IPAR(5). If the user needs to use a larger basis then the parameter IBDIM, defined on PIM_RGMRES must be changed accordingly.

2. The user must supply a routine to compute the 2-norm of a vector.

Algorithm A.9 *RGMRES*

1. $r_0 = Q_1(b - AQ_2x_0)$
2. $\beta_0 = ||r_0||_2$
   for $k = 1, 2, \ldots$
3. $g = (\beta_{k-1}, \beta_{k-1}, \ldots)^T$
4. $V_1 = r_{k-1}/\beta_{k-1}$
   for $j = 1, 2, \ldots, \text{restart}$
5. $R_{i,j} = V_i^T Q_1 A Q_2 V_j, \quad i = 1, \ldots, j$
6. $\hat{v} = Q_1 A Q_2 V_j - \sum_{i=1}^{j} R_{i,j} V_i$
7. $R_{j+1,j} = ||\hat{v}||_2$
8. $V_{j+1} = \hat{v}/R_{j+1,j}$
9. *apply previous Givens's rotations to* $R_{:,j}$
10. *compute Givens's rotation to zero* $R_{j+1,j}$
11. *apply Givens's rotation to* g
12. *if* $|g_{j+1}| <$ `RHSSTOP` *then*
    perform steps 13 *and* 14 *with* restart$\equiv j$
    stop
    endif
    endfor
13. *solve* $Ry = g$ *(solution to least-squares problem)*
14. $x_k = x_{k-1} + Vy$ *(form approximate solution)*
15. $r_k = Q_1(b - AQ_2x_k)$
16. $\beta_k = ||r_k||_2$
    endfor

A.12 PIM_RGMRESEV

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the restarted GMRES method; returns, at each iteration, the estimates of the smallest and largest eigenvalues of $Q_1 A Q_2$ obtained from the upper Hessenberg matrix produced during the Arnoldi process.

Synopsis

```
PIMSRGMRESEV(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDRGMRESEV(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCRGMRESEV(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZRGMRESEV(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	(4+IPAR(5))*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-2}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC), _SCAL, _TRSV
LAPACK	_HSEQR
LIBPIM	

Notes

1. The size of the orthonormal basis (maximum of 50 vectors) must be stored in IPAR(5). If the user needs to use a larger basis then the parameter IBDIM, defined on PIM_RGMRESEV must be changed accordingly.

2. The user must supply a routine to compute the 2-norm of a vector.

3. A box containing estimates of the eigenvalues of Q_1AQ_2 is returned in `DPAR(3)`, `DPAR(4)`, `DPAR(5)`, `DPAR(6)`, these values representing the minimum and maximum values in the real and imaginary axes, respectively.

Algorithm A.10 *RGMRESEV*

1. $r_0 = Q_1(b - AQ_2x_0)$
2. $\beta_0 = ||r_0||_2$
   for $k = 1, 2, \ldots$
3. $g = (\beta_{k-1}, \beta_{k-1}, \ldots)^T$
4. $V_1 = r_{k-1}/\beta_{k-1}$
   for $j = 1, 2, \ldots, \mathrm{restart}$
5. $R_{i,j} = V_i^T Q_1AQ_2V_j, \quad i = 1, \ldots, j$
6. $\hat{v} = Q_1AQ_2V_j - \sum_{i=1}^{j} R_{i,j}V_i$
7. $R_{j+1,j} = ||\hat{v}||_2$
8. $V_{j+1} = \hat{v}/R_{j+1,j}$
9. *apply previous Givens's rotations to* $R_{:,j}$
10. *compute Givens's rotation to zero* $R_{j+1,j}$
11. *apply Givens's rotation to* g
12. *if* $|g_{j+1}| <$ `RHSSTOP` *then*
    perform steps 13 *and* 14 *with* restart$\equiv i$
    stop
    endif
    endfor
13. *solve* $Ry = g$ *(solution to least-squares problem)*
14. $x_k = x_{k-1} + Vy$ *(form approximate solution)*
15. *compute eigenvalues of* H_{restart}
16. $r_k = Q_1(b - AQ_2x_k)$
17. $\beta_k = ||r_k||_2$
    endfor

A.13 PIM_RGCR

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the restarted GCR method.

Synopsis

```
PIMSRGCR(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDRGCR(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCRGCR(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZRGCR(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	(5+2*IPAR(5))*IPAR(4)+2*IPAR(5)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

1. The restarting value must be stored in IPAR(5)

Algorithm A.11 *RGCR*

1. $r_0 = Q_1(b - AQ_2x_0)$
   for $k = 1, 2, \ldots$
2. $P_1 = r_{k-1}$
   $x_k = x_{k-1}$, $r_k = r_{k-1}$
   for $j = 1, 2, \ldots,$ restart
3. $W_j = Q_1 A Q_2 P_j$
4. $\zeta_j = W_j^T W_j$
5. $\alpha_j = r_k^T W_j / \zeta_j$
6. $x_k = x_k + \alpha_j P_j$
7. $r_k = r_k - \alpha_j W_j$
8. *check stopping criterion*
9. $q = Q_1 A Q_2 r_k$
10. $P_{j+1} = r_k - \sum_{i=1}^{j} q^T W_i / \zeta_i P_i$
   endfor
   endfor

A.14 PIM_QMR

Purpose

Solves the system $Q_1AQ_2x = Q_1b$ using the QMR method with coupled two-term recurrences.

Synopsis

```
PIMSQMR(X,B,WRK,IPAR,SPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDQMR(X,B,WRK,IPAR,DPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCQMR(X,B,WRK,IPAR,SPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZQMR(X,B,WRK,IPAR,DPAR,MATVEC,TMATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	11*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

1. The weights ω are kept constant ($\omega = 1$) throughout the iterations. Please refer to [24] for a discussion on other choices for ω.

Algorithm A.12 *QMR*

1. $r_0 = Q_1(b - AQ_2x_0)$
2. $\rho_1 = ||r_0||_2$
3. $v_1 = r_0/\rho_1$
4. $w_1 = -r_0/\rho_1$
5. $p_0 = q_0 = d_0 = 0$
6. $c_0 = 1, \epsilon_0 = 1, \xi_1 = 1, \vartheta_0 = 0, \eta_0 = -1, \omega = 1$

for $k = 1, 2, \ldots$

7. $\delta_k = w_k^T v_k$
8. *if* $\epsilon_{k-1} = 0$ *hard-breakdown has occurred*
9. *if* $\delta_k = 0$ *hard-breakdown has occurred*
10. $p_k = v_k - (\xi_k \delta_k / \epsilon_{k-1}) p_{k-1}$
11. $q_k = w_k - (\rho_k \delta_k / \epsilon_{k-1}) q_{k-1}$
12. $\tilde{v}_{k+1} = Q_1 A Q_2 p_k$
13. $\epsilon_k = q_k^T \tilde{v}_{k+1}$
14. $\beta_k = \epsilon_k / \delta_k$
15. $\tilde{v}_{k+1} = \tilde{v}_{k+1} - \beta_k v_k$
16. $\tilde{w}_{k+1} = Q_1 A^T Q_2 q_k - \beta_k w_k$
17. $\rho_{k+1} = ||\tilde{v}_{k+1}||_2$
18. $\xi_{k+1} = ||\tilde{w}_{k+1}||_2$
19. $\vartheta_k = (\omega \rho_{k+1}) / (\omega c_{k-1} |\beta_k|)$
20. $c_k = 1/\sqrt{1 + \vartheta_k^2}$
21. $\eta_k = -\eta_{k-1} \rho_k c_k^2 / (\beta_k c_{k-1}^2)$
22. $d_k = p_k \eta_k + (\vartheta_{k-1} c_k)^2 d_{k-1}$
23. $x_k = x_{k-1} + d_k$
24. $r_k = Q_1(b - Q_2 A x_k)$
25. *check stopping criterion*
26. *if* $\rho_{k+1} = 0$ *hard-breakdown has occurred*
27. *if* $\xi_{k+1} = 0$ *hard-breakdown has occurred*
28. $v_{k+1} = \tilde{v}_{k+1} / \rho_{k+1}$
29. $w_{k+1} = \tilde{w}_{k+1} / \rho_{k+1}$

endfor

A.15 PIM_TFQMR

Purpose

Solves the system $Q_1 A Q_2 x = Q_1 b$ using the TFQMR method with 2-norm weights (see [23, Algorithm 5.1]).

Synopsis

```
PIMSTFQMR(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDTFQMR(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCTFQMR(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZTFQMR(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	10*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-3}$ $\boxed{-4}$

Function dependencies

BLAS	_COPY, _AXPY, (_DOT/_DOTC)
LIBPIM	

Notes

1. The user must supply a routine to compute the 2-norm of a vector.

Algorithm A.13 *TFQMR*

1. $r_0 = Q_1(b - AQ_2x_0)$
2. $w_1 = y_1 = r_0$
3. $v_0 = Q_1AQ_2y_1$
4. $d_0 = 0$
5. $\tau_0 = ||r_0||_2$
6. $\theta_0 = \eta_0 = 0$
7. $\tilde{r}_0 = r_0$
8. $\rho_0 = \tilde{r}_0^T r_0$

   for $k = 1, 2, \ldots$

9. $\quad \sigma_{k-1} = \tilde{r}_0^T v_{k-1}$
10. $\quad \alpha_{k-1} = \rho_{k-1}/\sigma_{k-1}$
11. $\quad y_{2k} = y_{2k-1} - \alpha_{k-1}v_{k-1}$

    for $m = 2k-1, 2k$

12. $\quad\quad w_{m+1} = w_m - \alpha_{k-1}Q_1AQ_2y_m$
13. $\quad\quad \theta_m = ||w_{m+1}||_2/\tau_{m-1}$
14. $\quad\quad c_m = 1/\sqrt{1+\theta_m^2}$
15. $\quad\quad \tau_m = \tau_{m-1}\theta_m c_m$
16. $\quad\quad \eta_m = c_m^2\alpha_{k-1}$
17. $\quad\quad d_m = y_m + (\theta_{m-1}^2\eta_{m-1}/\alpha_{k-1})d_{m-1}$
18. $\quad\quad x_m = x_{m-1} + \eta_m d_m$
19. $\quad\quad \kappa_m = \tau_m\sqrt{m+1}$
20. $\quad\quad$ *if* $\kappa_m < \varepsilon$ *check stopping criterion*

    endfor

21. $\quad \rho_k = \tilde{r}_0^T w_{2k+1}$
22. $\quad \beta_k = \rho_k/\rho_{k-1}$
23. $\quad y_{2k+1} = w_{2k+1} + \beta_k y_{2k}$
24. $\quad v_k = Q_1AQ_2y_{2k+1} + \beta_k(Q_1AQ_2y_{2k} + \beta_k v_{k-1})$

   endfor

A.16 PIM_CHEBYSHEV

Purpose

Solves the system $AQ_2x = b$ using the Chebyshev acceleration.

Synopsis

```
PIMSCHEBYSHEV(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PSSUM,PSNRM,PROGRESS)

PIMDCHEBYSHEV(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PDSUM,PDNRM,PROGRESS)

PIMCCHEBYSHEV(X,B,WRK,IPAR,SPAR,MATVEC,PRECONL,PRECONR,PCSUM,PSCNRM,PROGRESS)

PIMZCHEBYSHEV(X,B,WRK,IPAR,DPAR,MATVEC,PRECONL,PRECONR,PZSUM,PDZNRM,PROGRESS)
```

Storage requirements

Parameter	No. of words
X, B	IPAR(4)
WRK	5*IPAR(4)
IPAR	13
_PAR	6

Possible exit status values returned by IPAR(12): $\boxed{0}$ $\boxed{-1}$ $\boxed{-6}$ $\boxed{-7}$

Function dependencies

BLAS	_COPY, _AXPY, _SWAP, (_DOT/_DOTC)
LIBPIM	

Notes

1. Only stopping tests 1, 2 and 7 are allowed.
2. The box containing the eigenvalues of $I - Q_1AQ_2$ must be stored in DPAR(3), DPAR(4), DPAR(5), DPAR(6), these values representing the minimum and maximum values in the real and imaginary axes, respectively.

Algorithm A.14 *CHEBYSHEV*

1. *Set parameters for iteration:*
   - *If* $\texttt{DPAR(3)} \leq \lambda(I - Q_1 A) \leq \texttt{DPAR(4)}$ *(in the real axis):*
     $\sigma = (\texttt{DPAR(4)} - \texttt{DPAR(3)})/(2 - \texttt{DPAR(4)} - \texttt{DPAR(3)})$
     $\gamma = 2/(2 - \texttt{DPAR(4)} - \texttt{DPAR(3)})$

   - *If* $\texttt{DPAR(5)} \leq \lambda(I - Q_1 A) \leq \texttt{DPAR(6)}$ *(in the imaginary axis):*
     $\sigma^2 = -\max(\texttt{DPAR(5)}, \texttt{DPAR(6)})$
     $\gamma = 1$

   - *If* $\texttt{DPAR(3)} \leq \mathrm{Re}(\lambda(I - Q_1 A)) \leq \texttt{DPAR(4)}$ *and*
     $\texttt{DPAR(5)} \leq \mathrm{Im}(\lambda(I - Q_1 A)) \leq \texttt{DPAR(6)}$ *(in the complex plane):*
     $p = \sqrt{2}(\texttt{DPAR(4)} - \texttt{DPAR(3)})/2$
     $q = \sqrt{2}(\texttt{DPAR(6)} - \texttt{DPAR(5)})/2$
     $d = (\texttt{DPAR(3)} + \texttt{DPAR(4)})/2$
     $\sigma^2 = (p^2 + q^2)/(1 - d)^2$
     $\gamma = 1/(1 - d)$

2. $f = \gamma Q_1 b$
   for $k = 1, 2, \ldots$
3. $$\rho_k = \begin{cases} 1, & k = 1 \\ (1 - \sigma^2/2)^{-1}, & k = 2 \\ (1 - \rho_{k-1}\sigma^2/4)^{-1}, & k > 2 \end{cases}$$
4. $w = (I - Q_1 A Q_2) x_k$
5. $x_{k+1} = \rho_k(\gamma((I - Q_1 A) x_k + f) + (1 - \gamma) x_k) + (1 - \rho) x_{k-1}$
6. *check stopping criterion*
   endfor

A.17 PIM_SETPAR

Purpose

Sets the parameter values in the arrays IPAR and _PAR.

Synopsis

```
PIMSSETPAR(IPAR,SPAR,LDA,N,BLKSZ,LOCLEN,BASISDIM,NPROCS,PROCID,
           PRECONTYPE,STOPTYPE,MAXIT,EPSILON)
INTEGER IPAR(*)
REAL SPAR(*)
INTEGER LDA,N,BLKSZ,LOCLEN,BASISDIM,NPROCS,PROCID,
        PRECONTYPE,STOPTYPE,MAXIT
REAL EPSILON

PIMDSETPAR(IPAR,DPAR,LDA,N,BLKSZ,LOCLEN,BASISDIM,NPROCS,PROCID,
           PRECONTYPE,STOPTYPE,MAXIT,EPSILON)
INTEGER IPAR(*)
DOUBLE PRECISION DPAR(*)
INTEGER LDA,N,BLKSZ,LOCLEN,BASISDIM,NPROCS,PROCID,
        PRECONTYPE,STOPTYPE,MAXIT
DOUBLE PRECISION EPSILON
```

Storage requirements

Parameter	No. of words
IPAR	13
_PAR	6

Notes

1. When using the COMPLEX and DOUBLE COMPLEX PIM routines, call PIMSSETPAR and PIMDSETPAR respectively.

A.18 PIM_PRTPAR

Purpose

Prints the parameter values on the arrays IPAR and _PAR.

Synopsis

```
PIMSPRTPAR(IPAR,SPAR)
INTEGER IPAR(*)
REAL SPAR(*)

PIMDPRTPAR(IPAR,DPAR)
INTEGER IPAR(*)
DOUBLE PRECISION DPAR(*)
```

Storage requirements

Parameter	No. of words
IPAR	13
_PAR	6

Notes

1. May be called only on a processing element with I/O capability.
2. When using the COMPLEX and DOUBLE COMPLEX PIM routines, call PIMSPRTPAR and PIMDPRTPAR respectively.

A.19 _INIT

Purpose

Initialises a vector of length `n` with the scalar value `alpha`. Based on the level 1 BLAS routine _COPY.

Synopsis

```
SINIT(N,ALPHA,SX,INCX)
REAL ALPHA,SX(*)
INTEGER N,INCX

DINIT(N,ALPHA,DX,INCX)
DOUBLE PRECISION ALPHA,DX(*)
INTEGER N,INCX

CINIT(N,ALPHA,CX,INCX)
COMPLEX ALPHA,CX(*)
INTEGER N,INCX

ZINIT(N,ALPHA,ZX,INCX)
DOUBLE COMPLEX ALPHA,ZX(*)
INTEGER N,INCX
```

Storage requirements

Parameter	No. of words
_X	IPAR(4)

Notes

None

Index