BIT Numerical Mathematics
200x, Vol. xx, No. x, pp. xxx–xxx

0006-3835/03/4301-0001 $16.00

MATRIX-FREE PRECONDITIONING USING PARTIAL MATRIX ESTIMATION *

J.K. CULLUM[1] and M. TŮMA[2] †

[1] *Los Alamos National Laboratory, Los Alamos, 87545 NM, USA*

[2] *Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 18207 Praha 8, Libeň. email: tuma@cs.cas.cz*

Abstract.

We consider *matrix-free* solver environments where information about the underlying matrix is available only through matrix vector computations which do not have access to a fully assembled matrix. We introduce the notion of *partial matrix estimation* for constructing *good* algebraic preconditioners used in Krylov iterative methods in such *matrix-free* environments, and formulate three new graph coloring problems for partial matrix estimation. Numerical experiments utilizing one of these formulations demonstrate the viability of this approach.

AMS subject classification (2000): 65F10, 65F50, 49M37, 90C06.

Key words: Matrix-free algorithms, linear algebraic equations, large sparse matrices, preconditioned iterative methods.

1 Introduction.

We use the terminology *matrix-free* to denote Krylov solver environments where information about the matrix is available only through matrix-vector computations which do not have access to a fully assembled matrix.

This type of solver environment frequently occurs within the application of popular Newton-Krylov methods to the solution of nonlinear partial differential equations [33]. The inner loop computations consist of the application of Krylov methods to a sequence of associated linear problems. The matrix vector computations required for the Krylov method are accomplished by applying finite differencing techniques to the nonlinear functions. The use of finite differencing introduces error into the matrix vector computations, adding additional complexity to the computations.

The focus of this paper is on the solution of systems of linear algebraic equations

*Received May 21, 2004. Accepted in revised form August 15, 2006. Communicated by Petter Bjørstad.

†This work was supported by the U.S. Department of Energy under project No. W-7405-ENG-36, the DOE Office of Science, MICS, AMS Program; KC-07-01-01, by the National Program of Research "Information Society" under project 1ET400300415, by the Grant Agency of the Academy of Sciences of the Czech Republic, under No. IAA1030405, and by the institutional research plan No. AV0Z10300504. Part of this work was done during the visit of the second author at the Los Alamos National Laboratory and its support is warmly appreciated.

$$Ax = b \tag{1.1}$$

by Krylov subspace iterative methods within matrix-free environments where an assembled matrix is not available. We will not address any additional complications which are induced by the use of finite differencing.

Typically, a preconditioner is needed to achieve acceptable convergence of a Krylov method. Right preconditioners P transform system (1.1) into the equivalent system,

$$AP^{-1}y = b \text{ where } y = Px. \tag{1.2}$$

Recent work, see for example, [33], [19], [32], has demonstrated the level of difficulty in constructing *good* preconditioners in a Newton-Krylov solver environment. Not having direct access to the fully assembled underlying matrices prevents the use of many algebraic preconditioners. In matrix-free environments the emphasis has been on the use of preconditioners derived from simplifications of the problem for which explicit matrices can be readily constructed. These types of preconditioners have been demonstrated to work well on a variety of test problems.

It is well-known, however, that the use of a *good* algebraic preconditioner, such as approximate inverse preconditioning [5] or incomplete and truncated factorizations, can often accelerate the convergence of a Krylov method. The construction of algebraic preconditioners requires direct access to entries in the matrix.

In this paper we explore the feasibility of constructing algebraic preconditioners when working in a matrix-free solver environment. Specifically, we address the following question. Is it possible, using a *few* matrix vector computations, to extract sufficient information about the matrix to construct *good* algebraic preconditioners for the original problem?

We note that if the answer to this questions is yes, then there are obvious extensions of these ideas to the use of such preconditioners within sequences of matrices constructed during either the solution of nonlinear equations or during the solution of linear time dependent equations. For example, it would be possible to extract detailed information of the first matrix in any such sequence, construct a good algebraic preconditioner based upon that detailed information and then to reuse some or even all of the information about that matrix or about the corresponding preconditioner to construct preconditioners for other matrices in the sequence.

Using the same preconditioner over two or more similar linear solves is accepted practice [36], [37], [43]. For example, a Jacobian may be reused over several steps in a modified Newton method [31]. The work in this paper extends that idea to the possible reuse of simplified patterns for preconditioners in conjunction with the estimation of partial numerical information about the matrix to construct algebraic preconditioners for each linear solve.

DEFINITION 1.1. *The pattern (or pattern of nonzeros) of a matrix* $A \in R^{n \times n}$ *is*

$$\{(i,j) | a_{ij} \neq 0\}. \tag{1.3}$$

Algebraic preconditioners which are based on sparsified patterns of the original system matrix or powers of the matrix have been constructed and used successfully in a variety of applications, [6], [8], and [27]. The computations do not however need to be restricted to these types of patterns.

DEFINITION 1.2. *A matrix pattern will be said to be a sparsified pattern for a matrix A if it is obtained by neglecting one or more of the nonzero entries in A.*

Typically, a sparsified pattern is obtained by neglecting the matrix entries which are smaller in magnitude than a given threshold ϵ relative to the size of other matrix entries. For example, for each row i, discard any off-diagonal entries in that row which are smaller in magnitude than one-tenth of the element of maximum magnitude in that row. Sparsifications arise naturally in some applications. An example can be a matrix from discretized elliptic second-order PDE where matrix entries decay rapidly as their distance from the matrix diagonal increases [2]. Then a reasonable sparsification which is often used for preconditioner computation consists of a few diagonals of the matrix.

DEFINITION 1.3. *A matrix pattern will be said to be a prescribed pattern for a matrix A if the entries in A which correspond to that pattern are to be used.*

Prescribed matrix patterns can be used to specify a submatrix or can include some entries which are zero in the original matrix. For example, a zero entry may become a nonzero entry in a corresponding factorization used to compute an associated preconditioner.

In many applications the pattern of nonzero entries in the underlying matrix A is available. If Equation (1.1) arises from a partial differential equation, this structure can correspond to the mesh or to the dual mesh connectivity. Therefore, we make the following assumption.

ASSUMPTION 1.1. *The directed graph structure of the underlying matrix A is available for use in the construction of the preconditioner.*

In Section 2 we review briefly the history of sparse matrix estimation algorithms. In Section 3 we pose three new partial matrix estimation problems and indicate modifications of existing graph coloring algorithms to address each problem. For example, we show how the estimation of diagonally compensated matrices [3] can be accomplished without having to explicitly extract the original matrix in advance.

In Section 4 we develop the ideas presented in Problem 3.1 in Section 3, demonstrating that partial matrix estimation can be used to compute *good* algebraic preconditioners for a range of problems. In Section 5 we indicate directions for future research.

2 Matrix estimation using graph coloring

Popular algorithms for the optimization of scalar nonlinear differentiable functions employ approximations to the second derivative matrix, the Hessian matrix. This matrix is the Jacobian matrix of the corresponding gradient of the criteria function. Matrix estimation algorithms which are based upon the selec-

tion and execution of a set of well-chosen matrix vector computations were first proposed and used in the optimization literature to obtain estimates of these typically nonsymmetric Jacobian matrices.

In that application, given a gradient vector function $F(x)$, a specified vector x_0, a direction vector y and a perturbation parameter ϵ, finite differencing is employed to estimate the Jacobian matrix-vector multiplications, $F'(x_0)y$. Specifically,

$$[F(x_0 + \epsilon y) - F(x_0)]/\epsilon \quad \approx \quad F'(x_0)y.$$

Curtis, Powell and Reid, [15] were the first to demonstrate that in many cases the complete matrix can be estimated using very few functions evaluations. The Curtis, Powell, Reid (CPR) algorithm uses structural orthogonality of matrix columns to split the set of all columns into groups in such a way that all of the columns within any particular group are structurally orthogonal. Two columns are structurally orthogonal if and only if no row has a nonzero entry in both columns.

The columns within any given group j define a vector d_j which is the sum of all of the coordinate vectors corresponding to all of the rows in this group with nonzero entries. The successive computation of approximations to $F'(x_0)d_j$ for $j = 1, \ldots, p$ yields all of the nonzero entries in the matrix $F'(x_0)$.

PROBLEM 2.1. *Given the sparsity pattern of a matrix $A \in \mathbb{R}^{n\times n}$ find vectors, $D_p \equiv \{d_1, \ldots, d_p\}$ $d_j \in \mathbb{R}^n$, such that for each nonzero entry $a_{ij} \in A$ there exists a vector $d_k \in D_p$ such that $(Ad_k)_i = a_{ij}(d_k)_j$ and the cardinality p of D_p is minimized.*

For example, if A is tridiagonal, we have $p = min(n, 3)$. E.g., if $n = 10$, d_1 has only nonzero entries with indices $1, 4, 7, 10$, d_2 has nonzero entries with indices $2, 5, 8$, and d_3 has nonzeros only for the remaining indices $3, 6, 9$. In general, minimizing the cardinality p is an NP-hard problem. Therefore, in practice we typically look for D_p such that p is only approximately minimum.

Subsequently, Coleman and Moré demonstrated that Problem 2.1 is equivalent to a graph coloring problem for the associated undirected graph of the matrix A^TA. We denote this graph by $\mathcal{G}(A^TA)$. Structural pattern of the matrix A will be denoted by *struct*(A). This is the *column adjacency* or *column intersection* graph of A. The vertices of this graph correspond to the columns of A and an edge e_{ij} exists whenever the two columns i and j are not structurally orthogonal. Therefore, Problem 2.1 can be reformulated as Problem 2.2, the graph coloring problem (GCP). Two vertices i, j are adjacent if there is an edge $e_{i,j}$ connecting them.

PROBLEM 2.2. *Graph Coloring Problem (GCP). For $A \in \mathbb{R}^{n\times n}$ determine the minimum number of colors sufficient to color all of the vertices of the graph $\mathcal{G}(A^TA)$ such that any two adjacent vertices have been assigned different colors.*

The CPR algorithm in [15] provides direct estimation of the matrix entries. Later algorithms relax this requirement. Matrix entries may be obtained by solving a triangular system (substitution methods) or even by the solution of a general linear algebraic system (elimination methods). For additional details see, for example, [38], [14], [25], and [26]. Matrix estimation algorithms can be

designed to exploit any symmetry of the matrix efficiently, [12], [11].

The number of matrix vector computations which is required differs from one approximate coloring algorithm to another approximate coloring algorithm. If matrix vector computations with both the matrix A and its transpose A^T can be performed, their overall number required for the estimation of the matrix A can be decreased even more [13], [14]. Operations with A^T can be achieved, for example, using the reverse mode of automatic differentiation [21], [20]. Several authors have proposed partitioning based upon rows of the matrix [24], [23].

For practical computations, it is important to observe that it is not necessary to form the graph $\mathcal{G}(A^T A)$ in order to work with it. It is sufficient to work with the two directed graphs $\mathcal{G}(A)$ and $\mathcal{G}(A^T)$. Working with these two directed graphs is equivalent to using the bipartite graph model for A, [9], [18].

The algorithms referenced in this section focus on estimating the entire matrix A. In the remainder of this paper we concentrate on the partial matrix estimation problem. The primary objective is to obtain enough information about the matrix to enable the construction of *good* algebraic preconditioners for solving Equation(1.1).

In Section 3 we propose three new graph problems associated with partial matrix estimation.

3 Partial estimation and graph coloring problems.

Full estimation of a given matrix may require a significant number of matrix vector operations. To construct *good* algebraic preconditioners we may not need all of the underlying matrix. Given a matrix A, it may, for example, be sufficient to compute a preconditioner based upon a sparsification of A or upon some other prescribed pattern S which was obtained using auxiliary knowledge about the properties of the given problem. We denote the resulting matrix by $\widetilde{A}$. We can then estimate $\widetilde{A}$ instead of estimating all of A and compute a preconditioner P based upon $\widetilde{A}$. The computed P would then be used in Equation(1.1).

For some matrices there may be a significant reduction in the numbers of matrix vector computations required for estimating all of $\widetilde{A}$ when compared with the number required for estimating all of A. If for example, the underlying continuous problem is time dependent, it is possible that portions of the matrix A do not change markedly with time and thus do not need to be estimated at every time step. In some applications the diagonal entries of the matrix can be computed readily by other means and do not have to be included in the matrix estimation problem.

Thus, the focus in this section is on coloring algorithms for partial estimation of a given matrix A. Related questions can be found in [18] where the terminology *partial coloring problem* is used.

We specify a particular *starting pattern*, $struct(P)$, which for example, could be obtained by sparsification of the pattern for A. The mask represented by $struct(P)$ determines which entries of A are involved in the preconditioner construction. The actual preconditioner P is computed from the matrix $\widetilde{A}$ resulting from the application of this structural mask to A. The final pattern of the com-

puted preconditioner will differ typically from that of $\widetilde{A}$. For example an incomplete LU may introduce fill-in. Other preconditioners, for example, incomplete factorization with dual thresholds (ILUT) [41] and approximate inverse preconditioners (AINV) [5] may significantly alter the sparsity pattern from that of $\widetilde{A}$.

We want to estimate $\widetilde{A}$, those entries of A which correspond to the pattern $struct(P)$. The use of $\widetilde{A}$ is analogous to the *gangster operator* which was introduced by Toint [42] for sparse quasi-Newton methods in numerical optimization (cf. also techniques used in [4] and [34]). Toint's operator punched holes in the matrix in locations specified by a pattern. In contrast we punch holes in the matrix on the complement of the specified pattern.

We apply a coloring algorithm to $\widetilde{A}$ to identify a set of vectors $\widetilde{D}_p \equiv \{d_1, \ldots, d_p\}$ which would enable the full estimation of $\widetilde{A}$, if matrix vector computations for $\widetilde{A}$ were available. We cannot, however, work directly with $\widetilde{A}$. We only have matrix vector computations for A. Thus, the estimation of $\widetilde{A}$ is accomplished using matrix vector computations for A on the set of vectors $\widetilde{D}_p$. The influence of the matrix entries in the *difference of sets* $struct(A) \setminus struct(P)$ cannot be eliminated, so we incur an *approximation error*. For any estimated entry $\widetilde{a}_{i,j}$ in $\widetilde{A}$,

$$\sum_{k \in \{(i,k) \in (struct(A) \setminus struct(P))\}} |a_{ik}| \tag{3.1}$$

is an upper bound on the approximation error. Clearly, this error depends strongly on the sizes of the entries in A which are not in the starting pattern.

This suggests that a sparsification of A may yield a *good* starting pattern $struct(P)$. This choice leads to the following estimate of the resulting approximation error. Equation (3.2) incorporates the effect of the sparsification relative drop tolerance ϵ and its application by rows, and the effect of the choice of coloring used in the estimation of $\widetilde{A}$

$$\epsilon \max_k |a_{ik}||S_i|, \tag{3.2}$$

where $|S_i|$ denotes the cardinality of the following set

$$S_i \quad \equiv \quad \{k | (i,k) \in [(struct(A) \setminus struct(P)) \cap C(j)]\}.$$

More in detail, the approximation error incurred in the estimation of any particular entry, $\widetilde{a}_{ij}$, in $\widetilde{A}$ is a function of the number of entries in $struct(A) \setminus struct(P)$ and of the $\widetilde{A}$ coloring used. Only those entries a_{ik} in $A \setminus P$ with $k \in C(j)$ induce approximation error where $C(j)$ denotes the coloring group containing the j^{th} column of $\widetilde{A}$. Therefore, the actual approximation error in any estimated $\widetilde{a}_{ij}$ satisfies the following coloring dependent upper bound.

3.1 Balanced partial coloring problem(BPCP).

In the experiments in Section 4 we will attempt to decrease and spread out these approximation errors by utilizing a new coloring algorithm which tries to

maintain a small number of groups while attempting to minimize the variation in cardinality between the individual groups (colors). This leads to Problem 3.1 which we will call the *Balanced partial graph coloring problem (BPCP)*. Its solution might lead to a better distribution of the errors among the entries in a partial reconstruction of A. Problem 3.1 is, of course, NP-hard.

PROBLEM 3.1. *Balanced partial coloring problem (PBCP). Find a coloring of the graph* $\mathcal{G}(\widetilde{A}^T\widetilde{A})$ *which maximizes the cardinality of the smallest group of columns and uses the minimum number of colors* χ.

Standard sequential graph coloring routines can be modified in order to obtain approximations to a solution of the PBCP problem. For example, if a given column is to be colored and there exists more than one possible choice of color, from the existing set of candidate colors, we can choose a color which maximizes the cardinality of the smallest of the existing sets of columns corresponding to the current acceptable choices of colors for that column. Ties can be broken arbitrarily. More complicated heuristics could be introduced. For example, in the color selection process, we could take into account the number of nonzeros of A in $A \setminus P$ which would be picked up by the use of each of the acceptable existing groups of columns. The experiments in Section 4 used the first simple heuristic.

3.2 Diagonal partial coloring problem (DPCP).

A class of preconditioners, suitable for certain symmetric and positive definite matrices, which originated in [17] (see also [35]) is based on modifying matrix diagonal entries during incomplete factorizations to compensate for the dropped entries. In special cases, there is strong theoretical justification for such modifications [22]. The more general case is developed in [1] where several practical strategies are presented.

In practical cases, attempting to average the approximation error across the extracted matrix, as in the solution to the BPCP problem, may yield approximation errors which are fairly evenly distributed. However, better solutions may be obtained if the errors can be moved to well-chosen places in the matrix. Therefore, we ask the following question. Is it possible, in our partial matrix estimation process, to move all of the approximation error to the matrix diagonal or to positions $struct(A) \setminus struct(P)$? Consider the following equivalent problem.

PROBLEM 3.2. *Diagonal partial coloring problem (DPCP). Find a coloring of the graph of* $\mathcal{G}(\widetilde{A}^T\widetilde{A})$ *which allows the estimation, without any approximation error, of all of the off-diagonal entries in* $\widetilde{A}$*, using the minimum number of colors* χ.

The off-diagonal entries in $\widetilde{A}$ are the off-diagonal entries in A with locations which match the pattern $struct(P)$. A solution to Problem 3.2 can be obtained by generating a graph coloring solution to a particular graph $\widetilde{\mathcal{G}}$ which is a subgraph of the graph $\mathcal{G}(A^TA)$.

DEFINITION 3.1. *Given a matrix* A *and a pattern* $struct(P)$*. For each row* i *in* A*, let* G_i *denote the complete graph corresponding to the nonzero edges in row*

i. For each i define the graph $\widetilde{G}$ *which is obtained from* G_i *using the following rules. Let* (k,l) *denote any edge in* $\widetilde{G}_i$*. If the edge* $(i,k) \notin struct(P)$ *and the edge* $(i,l) \notin struct(P)$ *then remove edge* (k,l) *from* $\widetilde{G}_i$*. Similarly, remove edge* (i,k) *if* $(i,k) \notin struct(P)$*. Let* $\hat{G}_i$ *denote the modified* $\widetilde{G}_i$*. Define* $\hat{\mathcal{G}}$ *as the union of the* $\hat{G}_i$*.*

We note that edges (k,l) with $(i,k) \notin struct(P)$ and edge $(i,l) \notin struct(P)$ correspond to edges linking entries in row i of A which are in locations which are not part of the pattern $struct(P)$. The edges (i,k) which are removed correspond to edges linking a diagonal entry with some entry whose location is not in the pattern $struct(P)$.

THEOREM 3.1. *The solution of Problem 3.2 is the solution of the general coloring problem (GCP) for the subgraph* $\hat{\mathcal{G}}$ *of* $\mathcal{G}(A^TA)$*.*

PROOF. The proof is an immediate consequence of Definition 3.1. □

Removal of the indicated edges enables the possibility of the transfer of all of the approximation error to the diagonal entries. Moving these entries may also result in reducing the number of groups required by the coloring. Whether two columns can actually be merged depends on all the graphs $\widetilde{G}_i$.

Obtaining a heuristic algorithm for solving DPCP from the standard graph coloring implementation is straightforward. Since coloring algorithms for $\mathcal{G}(A^TA)$ are naturally based on the implicit representation by the graphs of A and of A^T, it is enough to mark each entry in A according to whether it is in the pattern $struct(P)$ or outside of the pattern $struct(P)$. For example, if the graphs are expressed in compressed sparse row (CSR) storage format [41], then locations can be distinguished by the sign of the column indices. The coloring algorithm can use these marks during the coloring process to identify the entries in the graph $\hat{\mathcal{G}}$.

Approximate solutions to the DPCP problem can be used to construct special and useful types of preconditioners. For example the preconditioner can be based on diagonally compensated reduction (DCR) [3]. This technique (or, more precisely, a special case of this technique) developed for symmetric positive definite (SPD) matrices consists of moving positive off-diagonal matrix entries to the diagonal to generate an associated M-matrix. The pattern of nonzeros $struct(P)$ from which the preconditioner is determined consists of the diagonal and the negative entries in A.

If the structure of A and the positions of the positive entries are known, then the solution to the DPCP problem gives the minimum number of matrix vector computations required to construct the starting pattern for a preconditioner which is based upon diagonally compensated sparsification. Given for example a sequence of linear solves inside of a nonlinear solve where the positive negative pattern in the matrix does not change during the sequence of solves, a full matrix estimation for the first matrix in the sequence could provide a suitable pattern for the computation of the preconditioners for the other solves in the sequence.

In some applications the diagonal entries of A can be computed directly and do not need to be included in the partial matrix estimation. In this situation a solution to Problem 3.2, when combined with the known diagonal entries, pro-

vides the partial matrix without any approximation error. If some part of the matrix is known a priori and does not need to be included in the subsequent partial matrix estimation problem, the authors in [18] labeled this type of problem a *partial coloring problem.*

3.3 Equivalent preconditioner by partial matrix estimation.

To formulate the question raised in this section, assume for the moment that all of the entries in the matrix A are explicitly available. In the computation of a preconditioner with a particular sparsity pattern $struct(P)$, it may be the case that not every nonzero in A is used in corresponding preconditioner computation. In the other words, some entries of A are not needed in this computation, and thus they do not need to be estimated.

$$A = \begin{pmatrix} * & * & & * & * \\ * & * & & & * \\ & * & * & * & \\ & & * & * & * \\ & & * & * & * \end{pmatrix}$$

Fig. 1. An example for Problem 3.3.

Consider the matrix in Fig. 1. If we have a tridiagonal starting pattern $struct(P)$ for this A, and therefore, also a final tridiagonal pattern, the matrix entries a_{14}, a_{15} and a_{25} are not used in the computation of the ILU(0) preconditioner for A.

To be specific consider the following problem statement for computing an incomplete LU(0) preconditioner.

PROBLEM 3.3. *Given a matrix A and a pattern $struct(P)$ find a submatrix $\widehat{A}$ of A such that those entries in the ILU(0) factorization of $\widehat{A}$ which are inside of the pattern $struct(P)$ are identical to the corresponding entries in the factors generated by applying ILU(0) directly to A.*

For Problem 3.3 there are two patterns. The starting pattern $struct(P)$ specifies the pattern for the preconditioner. The second pattern, $\widehat{struct(A)}$, specifies the pattern to be used in the partial matrix estimation of A. The second pattern identifies the set of elements in A which are needed to compute the ILU(0) preconditioner with the sparsity pattern $struct(P)$. A coloring algorithm would be applied to the second pattern. Entries in A which have no influence on the computation of the ILU(0) preconditioner are not estimated.

The sparsity pattern of $\widehat{A}$ can be characterized as follows.

OBSERVATION 3.1. *The pattern $\widehat{struct(A)}$ which is specified in Problem 3.3 is the minimum pattern satisfying the following conditions. For each (i,j) in this pattern.*

$$\begin{array}{llll} \textit{For } i<j, & \exists k>i, & (k,j)\in struct(P) & \textit{and} \quad (k,i)\in struct(A) \\ \textit{For } i>j, & \exists k>i, & (i,k)\in struct(P) & \textit{and} \quad (i,k)\in struct(A) \end{array} \tag{3.3}$$

Therefore, given the patterns $struct(P)$ and $struct(A)$ we can easily generate the required pattern $\widehat{struct(A)}$. As before, if A is only available through matrix vector computations, approximation error will be incurred during the partial estimation of $\widehat{A}$.

Since during the factorization, any two structurally symmetric off-diagonals entries will result in modifications to the corresponding diagonal entry, the proposed pattern corresponding to $\widehat{A}$ may be useful primarily when A is strongly nonsymmetric or made nonsymmetric by a sparsification, or when some of the entries in the symmetric part of A, for example, the diagonal entries, are available by other means.

The formation of $\widehat{struct(A)}$ may be considered a pattern extension of $struct(P)$. It would be possible to continue recursively applying this idea to the pattern for $\widehat{A}$.

In Section 4 we present the results of numerical experiments which confirm that the concept of partial estimation for the purposes of obtaining *good* algebraic preconditioners is viable.

4 Numerical experiments.

In this Section we summarize the results of numerical experiments which are focused on Problem 3.1. The test matrices reside in two sources. Most of the matrices are from the Tim Davis Collection [16] or its subcollections. e.g. the Harwell-Boeing package.

Several matrices were generated using the Los Alamos Truchas code [43] which is a parallel 3D finite volume code with unstructured grids used to simulate solidification manufacturing processes, most notably metal casting and welding operations. Some of the Truchas test matrices correspond to approximate Jacobian matrices obtained by systematic finite differencing of various nonlinear functions. Inside of Truchas, matrices are accessible only by matrix vector computations and the matrix vector computations do not work with assembled matrices.

For the purposes of these tests to demonstrate the viability of partial matrix estimation to generate *good* algebraic preconditioners, we work with fully assembled test matrices. In our experiments, we concentrate on the use of balanced partial coloring algorithm from Section 3.1 which is the most general of presented algorithms. It allows us to show that matrix-free environment can be successfully used for computations with iterative methods preconditioned by more different preconditioners. The main goal of the experiments is then to show that the partial estimation does work in the general setting, and that it is robust. The other described strategies were given mainly in order to show additional theoretical potential of partial graph colorings, and/or their possible use for preconditioners more tightly connected with particular problems. Diagonal partial graph coloring algorithm is the method which helps to decrease number of matrix vector computations (since the graph coloring is based on a submatrix

of the original matrix) and still to get exact (up to roundoff error) DCR preconditioner. Possible power of computing equivalent preconditioners by partial matrix estimations was indicated above. Detection of matrix patterns which do not need to be estimated for specific preconditioners in matrix-free environment seems to be a challenge.

4.1 Test procedure.

We first worked directly with each original matrix A. We used the coloring algorithms to determine the cost of full matrix estimation of A, as measured by the number of matrix vector computations which are required to fully estimate the matrix. We then computed either a ILUT, ILU(1), or AINV preconditioner for A, combined that preconditioner with the Krylov iterative method, restarted Flexible Generalized Minimal Residual, FGMRES(10) [39], and solved Equation(1.1).

Then for each test matrix, we obtained a starting pattern $struct(P)$ for a preconditioner by a sparsification of the original matrix. We labeled the sparsified matrix as $\widetilde{A}$. This corresponds to having applied the starting pattern $struct(P)$ of the sparsified matrix to A.

We then applied a coloring algorithm to the pattern $\widetilde{struct(A)}$ to determine the cost in terms of the number of matrix vectors computations required and the corresponding set of vectors, $\tilde{D}_p$ required to fully estimate $\widetilde{struct(A)}$. We then used the vectors $\widetilde{D}_p$ generated for $\widetilde{struct(A)}$ in matrix vector computations for A as a partial matrix estimation of A corresponding to the pattern $struct(P)$.

The resulting matrix $C = \widetilde{A} + \Delta$ where Δ represents the approximation error incurred by the use of A and not $\widetilde{A}$ in the matrix vector computations. We then computed the same type of preconditioner using C as we did with A, ILUT, ILU(1), or AINV. The computed preconditioner was then combined with FGMRES(10) to solve Equation(1.1). The results of these computations are summarized in Table 3.

4.2 Test matrices.

The test matrices are listed in Table 1. In Table 1, n is the size of the matrix, nnz is the number of nonzero entries in the matrix, and $Type$ indicates the type of application from which the matrix was drawn.

*cube*1 is a very simple example of a heat conduction problem. $gravflow$ represent a 2D flow problem driven by gravity. *palloy* are simple matrices from a discretized nonlinear simulation of a phase change in a metallic alloy. *circuit_*1, *add*20, *add*32 and *memplus* are from digital circuit simulations. *wang*1 is from a semiconductor simulation. *saylr*3 and *sherman*5 are from petroleum reservoir simulations. *venkat*1 and *venkat*25 correspond to different time steps in a 2D Euler solver on an unstructured grid. The $raefsky$ matrices represent various flow problems. *lung*1 corresponds to water transport in the lungs and *stomach* is from a 3D electro-physical model of the duodenum. For interest we added the nonphysical *appu* matrix from a NASA application benchmark.

4.3 Results of tests.

Table 3 contains representative results from a series of experiments as described in Section 4.1. For each test matrix listed, we present the results of two experiments.

The first set of experiments determined the cost of the full estimation of the original matrix without any approximation error. The coloring algorithms which were used had the same functionality as the algorithms and codes in [10] [9].

In the second set of experiments, a starting pattern $struct(P)$ was obtained by sparsification of A. For each row in A, the position of any nonzero entry in that row of A which was larger than a fixed fraction of the entry of maximum magnitude in that row was included in the the pattern $struct(P)$. This fixed fraction was specified by a drop tolerance. The larger the drop tolerance the sparser the pattern $struct(P)$. The number of nonzero entries in this pattern is recorded in Table 2 and Table 3 under the heading $Size\_S$. There are, of course, other possibilities for choosing the starting pattern [6] [27], including specific ones which are application dependent.

The partial estimation of the entries of A, those in $\widetilde{A}$, was achieved by applying the coloring algorithm described in Section 3.1 which approximates a solution to Problem 3.1. In order to be able to compare total amount of work and numbers of matrix vector computations required across the two sets of experiments, only results of those experiments which resulted in approximately the same size preconditioners are included in Table 3.

For most of the matrices the ILUT preconditioner [40] was computed. For the four circuit matrices we used the AINV preconditioner [5] which provides superior performance for these matrices. For some matrices, represented here by $venkat25$ and $raefsky3$, standard sparse preconditioning such as ILUT, even allowing many additional entries in each of L and U, did not work. In these cases we used an ILU(1) preconditioner.

In each case FGMRES(10) was used. The linear systems were preconditioned from the right, as indicated in Equation(1.2). In each test convergence was declared on the first iteration where the relative residual norm had decreased to 10^{-8}. Each right-hand side was computed by multiplying the vector of all ones by the matrix A. All experiments were performed on an Intel uniprocessor running Linux and compiled by a Lahey Fortran 95 compiler.

Table 2 provides a detailed and instructive examination of the results obtained across a range of different sparsification drop tolerances and preconditioner parameters and sizes. This matrix was chosen because of its interesting behavior and also because the study can be carried out using the one parameter preconditioning AINV algorithm. The presentation of results, e.g., for the popular ILUT preconditioner, when one parameter can replace another one, but only partially, seems to be more demanding.

Prior to working with the matrix *memplus* we removed all of the zero entries. In Tables 2 and 3 we use MV to denote the number of matrix-vector multiplications required for the estimation, and ITS to denote the number of preconditioned Krylov iterations to achieve convergence. $Size\_P$ denotes the

Table 1: Subset of Matrices Used in Tests

$Matrix$	Type	n	nnz
$cube1$	Heat Conduction	216	4096
$gravflow1$	2D Gravitational Flow	750	3580
$gravflow2$	2D Gravitational Flow	3750	23900
$palloy2$	Phase Change	29952	203312
$palloy3$	Phase Change	103200	707272
$circuit\_1$	Digital Circuit	2624	35823
$add20$	Digital Circuit	2395	17319
$add32$	Digital Circuit	4960	23884
$memplus$	Digital Circuit	17758	126150
$saylr3$	Reservoir Simulation	1000	3750
$sherman5$	Reservoir Simulation	3312	20793
$venkat01$	2D Euler	62424	1717792
$venkat25$	2D Euler	62424	1717792
$raefsky1$	Flow	3242	294276
$raefsky3$	Flow	21200	1488768
$raefsky5$	Flow	6316	168658
$lung1$	Biomedical	1650	7419
$stomach$	Biomedical	213360	3021648
$wang1$	Semiconductors	2903	19093
$appu$	NASA Benchmark	14000	1853104

number of nonzeros in the computed preconditioner. Combining $Size\_P$ with the number of iterations provides a reasonable idea about the efficiency of the preconditioner. For the tests which used partial matrix estimation, we also provide $Size\_S$, the size of the starting pattern for the preconditioner, which may play a significant role in the overall observed convergence.

First, it is clear from the Table 2 that for this matrix and using a starting pattern $struct(P)$ obtained by sparsification of A, can yield a large decrease in the number of matrix vector computations required for the partial matrix estimations.

For matrices generated by discretizing partial differential equations using regular stencils, it is expected that the decrease in the number of matrix vector computations will be limited.

Second, we observe that sparsification can provide matrices which are very sparse but which still produce *good* preconditioners. In particular, using partial matrix estimation with approximation errors, we obtain preconditioners which require the same number of Krylov iterations as the preconditioner computed from the original matrix A, even though these preconditioners are sparser.

In these tests, the dependence of the number of iterations on the size of the sparsified pattern, $Size\_S$, and on the size of the preconditioner, $Size\_P$, is only roughly monotonic. The number of iterations can decrease even when these sizes increase. Which of the choices is the best depends on a number

Table 2: Partial Matrix Estimation for Computing AINV Preconditioners Used in FGMRES(10)

No Sparsification		$Size\_S = 59984$		$Size\_S = 62281$	
$MV = 353$		$MV = 19$		$MV = 35$	
$Size\_P$	ITS	$Size\_P$	ITS	$Size\_P$	ITS
221349	16	131266	43	249209	17
159570	20	92270	58	154208	29
151681	36	81379	32	146159	48
147823	43	69656	84	85443	63
112129	202	67042	83	69762	466
76502	181	66185	73	68945	536
76044	220	63910	170	64261	449
75359	236	63722	157	62460	368
68991	264	63679	179	62254	285
63093	272	63669	178	62247	385
59547	240	62147	1121	62246	398
54228	880	62108	1257	61811	967
53780	1611	60986	2019	61797	1155

of considerations. In some cases the number of matrix vector computations required is very important. In other cases, both $Size\_S$ and and $Size\_P$ can play a crucial role. This is strongly application dependent.

Table 3 presents results of experiments with the matrices listed in Table 1. As indicated earlier, we used three different preconditioners. For each test which used ILUT or AINV preconditioners we present two results obtained using different preconditioner parameters. On tests which used the ILU(1) preconditioner we present only one result that represents the behavior for the original and for the partially estimated matrix. We did not do exhaustive searches for optimal parameter choices but the results presented in Table 3 seem to be representative. Additional tests were carried out on other matrices with similar results and are not included here. For example, the experiments on the matrix $venkat50$ were very similar to those on $venkat25$.

Because the partially estimated matrices are sparser than the original matrices, typically there is less fill-in in the corresponding preconditioners. Therefore, in order to compare the efficiency of the partial matrix estimation with the full matrix estimation, the preconditioner parameters for the partial matrix estimation tests were selected to generate preconditioners of similar size as those generated in the corresponding cases with full matrix estimation.

The results summarized in Table 3 confirm what we observed in Table 2. In most cases the preconditioners constructed from a partial matrix estimation require similar, and sometimes even smaller numbers of Krylov iterations and fewer matrix vector computations.

We note that although ILUT is a popular choice for preconditioners, it does not keep track of the original matrix pattern when deciding which entries to drop.

Table 3: Partial Matrix Estimation for Preconditioners Used in FGMRES(10)

	Full Matrix Estimation			Partial Matrix Estimation			
Matrix	*MV*	*Size_P*	*ITS*	*MV*	*Size_S*	*Size_P*	*ITS*
*cube*1	27	3166	20	24	2168	3166	24
*cube*1	27	3636	17	25	2395	4084	18
*gravflow*1	7	2849	55	5	2302	2950	44
*gravflow*1	7	5144	33	5	2302	3010	43
*gravflow*2	11	14544	72	7	11830	14962	71
*gravflow*2	11	14244	75	7	11830	15334	53
*palloy*2	12	156321	5	9	150000	150788	6
*palloy*2	12	145144	6	9	150000	167887	6
*palloy*3	12	576185	6	8	509808	539460	7
*palloy*3	12	731472	6	8	509808	730851	7
*sherman*5	27	22159	54	19	11298	21187	55
*sherman*5	27	24848	43	19	12395	23949	43
*add*20	84	8611	15	5	8239	8031	16
*add*20	84	10972	14	5	8239	10112	15
*add*32	15	20092	10	5	16646	20764	11
*add*32	15	25992	9	5	16646	23071	10
*saylr*3	9	3337	41	7	3015	3339	41
*saylr*3	9	4002	24	7	3015	3993	26
*circuit_*1	2571	27749	9	4	7539	15118	11
*circuit_*1	2571	32943	8	5	9723	17347	10
*wang*1	12	5525	213	11	15195	12355	207
*wang*1	12	12637	127	11	16868	12656	121
*lung*1	8	5884	28	4	4996	5884	5
*lung*1	8	5061	126	4	4945	5061	21
stomach	31	320678	38	13	1004437	354390	38
stomach	31	366566	29	25	1479804	368496	29
*venkat*01	44	72393	160	36	1076995	72106	157
*venkat*01	44	80249	149	37	1362721	80035	147
*raefsky*1	140	99344	252	121	157585	92445	233
*raefsky*1	140	132646	286	89	113726	132646	286
*raefsky*5	65	22903	4	55	71573	26316	5
*raefsky*5	65	28785	4	58	88134	28325	5
*raefsky*3	93	2148096	53	82	960589	1663634	48
*venkat*25	44	2267958	116	43	1621131	2238075	118
appu	1679	327688	57	706	1718426	327721	57
appu	1679	1075605	57	1476	1718426	1075747	57

In some cases, on difficult problems, a prespecified pattern based preconditioner like ILU(1) can be better. In other cases, preconditioners like ILUT and AINV provide better convergence when using partial matrix estimation.

If a matrix is too *regular* in the size of the matrix entries, there may not be good parameters for sparsification of the matrix. Selecting a parameter either resulted in dropping a lot of entries (resulting in a weak preconditioner no matter what parameters were then used) or in not dropping enough entries to decrease the required number of matrix vector computations.

The actual gain achieved by using preconditioning which is based on partially estimated matrices depends strongly on the ratio of the time required for the matrix vector computations as compared to the time required to precondition the system. There are, however, other considerations. In parallel computations, we may require or use partial estimation in order to decrease the amount of communication required. In other cases it may happen that we do not gain much over full estimation when we compare the sum of the number of matrix vector computations required for the estimation with the number required by the Krylov method.

Partial matrix estimation provides a mechanism for constructing algebraic preconditioners for systems where an assembled matrix is not available and information about the matrix can only be gained through matrix vector computations.

5 Conclusions.

We have presented three practically motivated graph coloring problems and tested the viability of one of them. The resulting numerical experiments confirm that matrix sparsification combined with partial matrix estimation provides a viable mechanism for constructing algebraic preconditioners for many different types of systems where an assembled matrix is not available and information about the matrix can only be gained through matrix vector computations.

This approach will be of importance in the application of Newton-Krylov methods for solving large systems of nonlinear equations, e.g. from nonlinear partial differential equations. It will also be of importance in generating practical preconditioners within time dependent problems where the matrices are slowly changing over intervals of time.

The idea of partial matrix estimation has potential applications in many other areas, for example, in optimization problems. The solution of related symmetric graph coloring problems would contribute to methods for solving sequences of problems in numerical optimization which estimate Hessian matrices.

It can be shown that the DPCP problem, Problem 3.2, often requires a number of colors somewhere between those required for the BPCP, Problem 3.1 and the standard coloring problems. The potential for its use may appear in preconditioning based on diagonally compensated matrices, and we will explore this possibility.

6 Acknowledgements.

The authors would like to thank Doug Kothe for introducing us to the functionality of the Los Alamos Truchas code, Bryan Lally for teaching the first author how to work with the Truchas code, and both Lally and John Turner for help with extracting sample test matrices from the code. We would like to thank to both anonymous referees for careful reading of the paper.

REFERENCES

1. M.A. Ajiz and A. Jennings. A robust incomplete Choleski-conjugate gradient algorithm. *Internat. J. Num. Methods Engng. 20 (1984), pp. 949–966.*
2. O. Axelsson. Iterative Solution Methods. Cambridge University Press, Cambridge, 1994.
3. O. Axelsson and L. Kolotilina. Diagonally compensated reduction and related preconditioning methods. *Numerical Lin. Algebra Appl. 1 (1994), pp. 155–177.*
4. M.W. Benson. Iterative solution of large-scale linear systems. *Master's Thesis, Lakehead University, Thunder Bay, 1973.*
5. M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput. 19 (1998), pp. 968–994.*
6. E. Chow. A priori sparsity patterns for parallel sparse approximate inverse preconditioners. *SIAM J. Sci. Comput. 21 (2000), 1804–1822.*
7. E. Chow, ParaSails User's Guide, Tech. Report UCRL-MA-137863, Lawrence Livermore National Laboratory, Livermore, CA, 2000.
8. E. Chow, Parallel implementation and practical use of sparse approximate inverses with a priori sparsity patterns, Int. J. High Perf. Comput. Appl., 15, 56-74, 2001.
9. T.F. Coleman, B.S. Garbow and J.J. Moré. Software for estimation sparse Jacobian matrices. *ACM Trans. Math. Software 10(1984), 329–345.*
10. T.F. Coleman and J.J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems, *SIAM J. Numer. Anal. 20(1983), 187–209.*
11. T.F. Coleman and J. Cai. The cyclic coloring problem and estimation of sparse Hessian matrices. *SIAM J. Discrete Alg. Methods, 7(1986), pp. 221–235.*
12. T.F. Coleman and J.J. Moré. Estimation of sparse Hessian matrices and graph coloring problems, *Math. Programming 28(1984), 243–270.*
13. T.F. Coleman and A. Verma. Structure and efficient Jacobian calculation. *in: Computational Differentiation: Techniques, Applications and Tools, Berz et al. eds., SIAM, Philadelphia, 1996, pp. 149–159.*
14. T.F. Coleman and A. Verma. The efficient computation of sparse Jacobian matrices using automatic differentiation, *SIAM J. Sci. Comput. 19(1998), 1210–1233.*
15. A.R. Curtis, M.J.D. Powell and J.K. Reid. On the estimation of sparse Jacobian matrices, *J. Inst. Math. Appl. 13(1974), 117–119.*

16. T. Davis. Sparse matrix collection. *NA Digest 42 (1994),* `http://www.cise.ufl.edu/ ~davis/sparse/`.

17. T.F. Dupont, R.P. Kendall and H.H. Rachford. An approximate factorization procedure for solving self-adjoint elliptic difference equations. *SIAM J. Numer. Anal. 5 (1968), pp. 559–573.*

18. A.H. Gebremedhin, F. Manne and A. Pothen. Graph coloring in optimization revisited. *preprint, Department of Informatics, University of Bergen, 2003.*

19. K. Georg. Matrix-free numerical continuation and bifurcation. *Numerical Functional Analysis and Optimization, 22 (2001), pp. 303–320.*

20. A. Griewank. Direct calculation of Newton steps without accumulating Jacobians. *in: Large-Scale Numerical Optimization, T.F. Coleman and Y. Li, eds., SIAM, Philadelphia, 1990, pp. 115–137.*

21. A. Griewank. Some bounds on complexity of gradients, Jacobians and Hessians. *in: Complexity in Numerical Optimization, P.M. Pardalos, ed., World Scientific Publishing Company, River Edge, NJ, 1993.*

22. I. Gustafsson. A class of first order factorization methods. *BIT, 18 (1978), pp. 142–156.*

23. S. Hossain. On the computation of sparse Jacobian matrices and Newton steps. *Technical report No. 146, Department of Informatics, University of Bergen, 1998.*

24. S. Hossain and T. Steihaug. Graph coloring and the estimation of sparse Jacobian matrices with segmented columns, *Technical report No. 72, Department of Informatics, University of Bergen, 1997.*

25. S. Hossain and T. Steihaug. Sparsity issues in the computation of Jacobian matrices, *Technical report No. 233, Department of Informatics, University of Bergen, 2002.*

26. S. Hossain and T. Steihaug. Reducing the number of AD passes for computing of a sparse Jacobian matrix. *in: Automatic Differentiation: From Simulation to Optimization. G. Corliss, C. Faure, A. Griewank, L. Hascoët and U. Naumann, eds., Computer and Information Science, Springer, New York, 2002.*

27. T. Huckle. Approximate sparsity patterns for the inverse of a matrix and preconditioning. *in: Proceedings IMACS World Congress 1977, R. Weiss and W. Schönauer, eds., Berlin, 1997.*

28. D. Hysom, A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. SIAM J. Sci. Comput. 22 (2001), 2194–2215.

29. D. Hysom, A. Pothen. Efficient parallel computation of ILU(k) preconditioners. Proceedings Supercomputing '99, IEEE Computer Society Press, 1999.

30. G. Karypis, V. Kumar. Parallel threshold-based ILU factorization. Technical Report TR-96-061, University of Minnesota, 1996.

31. C.T. Kelley, Solving Nonlinear Equations with Newton's Method, SIAM, Philadelphia, 2003.

32. D.E. Keyes. Terascale implicit methods for partial differential equations. *Contemporary Mathematics 306 (2001), AMS, Providence, pp. 29–84.*

33. D.A.Knoll and D.E. Keyes. Jacobian-free Newton-Krylov methods: A survey of approaches and applications. *J. Computational Physics, to appear, 2003.*

34. L.Yu. Kolotilina, A.A. Nikishin and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings. IV: Simple approaches to rising efficiency. *Numerical Lin. Algebra Appl. 6 (1999), pp. 515–531.*

35. G. Meurant. Computer Solution of Large Linear Systems. *Studies in Mathematics and Its Applications, North-Holland, 1999.*

36. J.L. Morales and J. Nocedal. Automatic preconditioning by limited memory quasi-Newton updates. *SIAM J. on Optimization, 10 (2000), pp. 1079–1096*

37. J.L. Morales and J. Nocedal. Algorithm 809: PREQN: Fortran 77 preconditioning of the conjugate gradient method. *ACM Trans. Math. Software 27 (2001), pp. 83–91.*

38. P.E. Plassman. Sparse Jacobian estimation and factorization on a multiprocessor, *in: T.F. Coleman and Y. Li, eds., Large-Scale Optimization, pp. 152–179, SIAM, Philadelphia, 1990.*

39. Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Stat. Computing, 14 (1993), pp. 461–469.*

40. ILUT: a dual threshold incomplete LU factorization. *Numerical Lin. Algebra Appl. 1 (1994), pp. 387–402.*

41. Y. Saad. Iterative Methods for Sparse Linear Systems *PWS Publishing Company, 1996.*

42. P.L. Toint. On sparse and symmetric matrix updating subject to a linear equation. *Mathematics of Computation, 31 (1977), pp. 954–961.*

43. Truchas User's Guide, *Los Alamos National Laboratory, 2004, to appear.*