Aachen Institute for Advanced Study in Computational Engineering Science

Preprint: AICES-2010/06-3

21/June/2010

Proof-Driven Derivation of Krylov Solver Libraries

V. Eijkhout, P. Bientinesi and R. van de Geijn

RWTH AACHEN UNIVERSITY

AICES

Financial support from the Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111 is gratefully acknowledged.

List of AICES technical reports: http://www.aices.rwth-aachen.de/preprints

Proof-Driven Derivation of Krylov Solver Libraries

Victor Eijkhout[1], Paolo Bientinesi[2], and Robert van de Geijn[3]

[1]Texas Advanced Computing Center, The University of Texas at Austin
[2]RWTH Aachen, Germany
[3]Department of Computer Science, The University of Texas at Austin

Abstract

In a series of papers, it has been shown that algorithms for dense linear algebra operations can be systematically and even mechanically derived from the mathematical specification of the operation. A frequent question has been whether the methodology can be broadened to iterative methods. In this paper, we show preliminary evidence that this is indeed the case for so-called Krylov subspace methods.

Our aims with this are two-fold: first of all, we show how the FLAME paradigm can simplify the derivation of subspace iteration methods. In view of this, the fact that we only derive the classical conjugate gradient method should be viewed as a promise for the future, rather than as a limitation of this approach.

Secondly, and more importantly, our use of FLAME shows how mechanical reasoning can derive full algorithm specifications from the constraints (for instance, orthogonality conditions) on the generated results. If we tie this to ongoing research in automatic optimized code generated in FLAME, we see that this research is a necessary building block towards automatic generation of optimized library software.

0.1 Introduction

We present the Conjugate Gradients (CG) algorithm [8] in the FLAME framework [7, 3, 12, 2]. With this, we have two goals in mind. First, we show how FLAME can be used to simplify the process of deriving iterative methods. Second, we make a case that a FLAME-based environment for deriving new iterative methods is a distinct, and attractive, possibility.

Traditional expositions of the CG method, and ones related to it, posit the basic form of relations between matrices and vectors, and compute the scalar coefficients in them by 'lengthy induction arguments' [11]. In the spirit of Householder's derivation [9], we summarize vector and scalar sequences as matrices; the FLAME framework then allows us to derive in a formal manner the actual iteration from properties on the quantities constructed in the algorithm. The big advantage here is that we can dispense with quantified statements over sequences, and instead consider predicates over simple, unindexed, objects. Simultaneously, the inductive arguments that have always been at the heart of traditional expositions are captured in a framework that guides the derivation of the algorithm and the proof of its correctness.

As a demonstration of the potential of this approach, we derive a CG method for nonsymmetric systems. The conciseness of our derivation should be contrasted with the lengthy research papers in the classical approach to polynomial iterative methods [13, 10].

Beyond simply presenting an alternative derivation of these methods, we argue that the essential calculations in a FLAME worksheet can be derived mechanically from the loop invariant of the algorithm. Coupling this to ongoing projects for automatic code generation from FLAME worksheets [2] raises the possibility of automatic generation of numerical libraries. Krylov subspace methods are then merely a proof-of-concept of a much more general idea: the mechanical derivation of algorithms and tuned library software incorporating these algorithms.

0.2 Theory and Notation

In this paper we present the Conjugate Gradients method using a block formalism [1, 9]. Rather than positing the basic form of the coupled recurrences of residuals and search directions, we derive their existence as it were 'from first principles'. This will give a clear separation between the basic form of the update equations, which hold for all polynomial iterative methods, and the specific values of the coefficients which follow from orthogonality requirements.

That said, we start by showing a few specific iterative methods, and how they

fit the common pattern that we will derive.

0.2.1 Some iterative methods

Iterative methods for linear systems of equations come in a great variety, and with greatly differing levels of sophistication.

Richardson iteration The simplest method for iteratively solving $Ax = b$ is probably the *Richardson* iteration

$$x_{i+1} = x_i - \alpha r_i,\ i = 0, \ldots$$

where $r_i = Ax_i - b$. Multiplying this equation by A gives

$$A(x_{i+1} - x_i) = Ax_{i+1} - Ax_i = Ax_{i+1} - b - (Ax_i - b) = r_{i+1} - r_i = -\alpha A r_i$$

so that $r_{i+1} = (I - \alpha A)r_i$. This tells us that the residuals converge to zero and that therefore the iterates converge to the solution, if the spectrum of $(I - \alpha A)$ is bounded in magnitude by one: $|\lambda(I - \alpha A)| < 1$. In the commonly encountered case where the eigenvalues of A satisfy $0 < \lambda_0 \leq \ldots \leq \lambda_{N-1}$, it is easy to show that the optimal α is $2/(\lambda_0 + \lambda_{N-1})$. For future reference we will remark that $r_i = \pi^{(i)}(A)r_0$ where $\pi^{(i)}(x) = (1 - \alpha x)^i$ is an i-th degree polynomial.

Stationary iterative methods The Richardson iteration is a first example of a *stationary* iterative method; more general methods have the form

$$x_{i+1} = x_i - M^{-1} r_i,\ i = 0, \ldots.$$

A similar analysis shows that the condition $|\lambda(I - AM^{-1})| < 1$ now guarantees convergence.

Steepest descent Next, Steepest Descent (SD) is the method

$$x_{i+1} = x_i - \alpha_i r_i,\ i = 0, \ldots$$

where α_i is chosen to minimize $\|r_{i+1}\|_2$. Thinking of this as finding the optimal step size along the line through x_i in the direction r_i explains the term *search direction* for r_i. The convergence analysis for this method is more complicated; we will only remark that now $r_{i+1} = \Pi_{0 \leq j \leq i}(I - \alpha_j A)r_0$.

The Conjugate Gradient method The CG method is most commonly presented as coupled iterations:

$$\begin{cases} x_{i+1} = x_i - \alpha_i p_i \\ p_i = r_i - \beta_i p_{i-1} \end{cases}$$

With a little algebra, this can be seen to be equivalent to

$$x_{i+1} = x_i - \sum_{j \leq i} \gamma_{ij} r_j. \tag{1}$$

The coefficients α_i, β_i are chosen to ensure the orthogonality of the residuals r_i.

Summary The iterative methods in this section are seen to conform to a scheme where the iterates are updated with a single vector, which itself is a combination of residual vectors.

0.2.2 Block formalism

The above presentation is typical for how textbooks on numerical linear algebra may progress, going from simple to more complicated methods, and arriving at the general scheme (1). Here we show that this scheme is in fact a natural choice, and we express it in a block formalism. This section serves to familiarize the reader with the block formalism, and to establish the basic equations, as well as the question of their essential degrees of freedom. These dofs will then be derived in subsequent sections.

Let the linear system $Ax = b$ be given, let x_0 be any (initial) vector and define $r_0 = Ax_0 - b$, the initial residual. Then $r_0 = Ax_0 - b$ implies that $x = A^{-1}b = x_0 - A^{-1}r_0$. Now, the Cayley-Hamilton theorem states that for every A there exists a polynomial $\phi(x)$ such that $\phi(A) = 0$. Without loss of generality, we can write $\phi(x) = 1 + x\pi(x)$ with π another polynomial. Then $0 = \phi(A) = I + A\pi(A)$ and hence $A^{-1} = -\pi(A)$ so that $x = x_0 + \pi(A)r_0$. In theory, *if* we were to know the coefficients of $\pi(x)$, we would be done.

The problem is that we don't know the coefficients and even if we knew them, the degree of the polynomial might be too high for practical use. Thus we arrive at the notion of a sequence of polynomials $\{\tilde{\pi}^{(i)}\}_{i>0}$ and a sequence of approximations

$$x_i = x_0 + \tilde{\pi}^{(i)}(A)r_0, \, i > 0 \tag{2}$$

that, we hope, converge to the solution x, to a reasonable accuracy, in a reasonable number of iterations.

We now need the basic concept of a 'Krylov sequence', which, given $n \times n$ matrix A and initial vector k_0, is defined as the matrix with infinite number of columns

$$K\langle A, k_0\rangle \equiv \left(\; k_0 \;\middle|\; Ak_0 \;\middle|\; A^2k_0 \;\middle|\; \cdots \; \right).$$

In other words, the j-th column of $K\langle A, k_0\rangle$, k_j, is defined by the recurrence

$$k_j = \begin{cases} k_0 & if \text{ j} = 0 \\ Ak_{j-1} & otherwise. \end{cases}$$

With this, we rewrite (2) as

$$x_{i+1} = x_0 + K\langle A, r_0\rangle \begin{pmatrix} \tilde{\pi}_{0i} \\ \vdots \\ \tilde{\pi}_{ii} \\ 0 \\ \vdots \end{pmatrix}, \tag{3}$$

where $\tilde{\pi}_{ij}$ is the i-th coefficient of the polynomial $\tilde{\pi}^{(j)}$. Introducing the matrices

$$J = \begin{pmatrix} 0 & 0 & \cdots \\ 1 & 0 & \cdots \\ 0 & \ddots & \ddots \end{pmatrix} \text{ and } E = \begin{pmatrix} 1 & 1 & \cdots \\ 0 & 0 & \cdots \\ 0 & \ddots & \ddots \end{pmatrix} \text{ so that } J - E = \begin{pmatrix} -1 & -1 & \cdots \\ 1 & 0 & \cdots \\ 0 & \ddots & \ddots \end{pmatrix}, \tag{4}$$

we can write Equation (2) in matrix form as

$$X(J - E) = K\tilde{U}, \tag{5}$$

where from now on we use K for $K\langle A, r_0\rangle$, $X = (x_0, x_1, \cdots)$, and $\tilde{U}$ is the upper triangular matrix containing the $\tilde{\pi}_{ij}$ coefficients. Using the matrix J, the definition of the Krylov sequence can be written as

$$AK = KJ. \tag{6}$$

If Equation (6) denotes a relation between finite segments of the infinite series K, we have to worry about correctness in the last column. We ensure this by a notational shorthand. First of all, we let the matrix J be rectangular with one more row than columns. Second, if K is a matrix with n columns, then $\underline{K}$ denotes the matrix consisting of the first $n-1$ columns of K. With this, equation (6) becomes

$$A\underline{K} = KJ.$$

Equivalently, by substituting $U \leftarrow \tilde{U}(I - J^t)$, we find

$$X(J - I) = \underline{K}U \tag{7}$$

which is now shorthand for the common form $x_{i+1} = x_i + \pi^{(i)}(A)r_0$ for some sequence of polynomials $\{\pi^{(i)}\}_i$, where the i-th column of the upper triangular matrix U holds the coefficients of polynomial $\pi^{(i)}$. This leads us to our formal definition:

Definition 1 *We call a sequence $X = (x_0, x_1, \ldots)$ a polynomial iterative method if it satisfies*

$$x_{i+1} = x_i + \pi^{(i)}(A)r_0$$

for some sequence of polynomials $\{\pi^{(i)}\}_i$ with $\deg(\pi^{(i)}) = i$. Notation: $X = P\langle\{\pi_i\}_{i\geq 0}, A, x_0, b\rangle$.

Of equal interest is the corresponding sequence of residuals: $R = (r_0, r_1, \ldots)$, where $r_i = Ax_i - b$:

Definition 2 *We call R a 'residual sequence' if it is the sequence of residuals of a polynomial iterative method X generated from A, x_0, and b:*

$$R = AX - be^t \text{ or } r_i = Ax_i - b$$

where $e = (1, 1, \ldots)^t$. Notation: $R = R\langle A, X, b\rangle$.

Lemma 1 *Let A, f and X be a given matrix, vector, and sequence, respectively. Let $R = R\langle A, X, b\rangle$. Then*

$$\begin{aligned} & \exists_{\{\pi_i \in \mathbf{P}^{(\mathbf{n})}\}} : X = P\langle\{\pi_i\}_{i\geq 0}, A, x_0, b\rangle \\ \Leftrightarrow \quad & \exists_{U \in \mathbf{U}^{(\mathbf{n})}} : R\langle A, X, b\rangle = K\langle A, x_0, b\rangle U \end{aligned}$$

where $\mathbf{U}^{(\mathbf{n})}$ is the set of upper triangular matrices U satisfying $u_{0*} = 1$.

Proof: Suppose X is generated by a polynomial iterative method $P\langle\{\pi_i\}_{i\geq 0}, A, x_0, b\rangle$. Multiplying the equation

$$x_{i+1} = x_0 + \pi_i(A)r_0$$

by A and subtracting b on both sides gives

$$r_{i+1} = r_0 + A\pi_i(A)r_0,$$

in other words, $r_{i+1} = \phi_{i+1}(A)r_0$ with $\phi_i(x) = 1 + x\pi_i(x)$. This can clearly be written as $R = KU$ where $U = U(\phi_i) \in \mathbf{U}^{(\mathbf{n})}$ and $K = K\langle A, r_0\rangle$. It is easy to see that all implications in this proof are equivalences. •

With X a polynomial iterative method and R its residual sequence, we notice that X can also be defined as

$$X(J-I) = RU \tag{8}$$

for some upper triangular matrix U. The above notational convention of underlining is extended to letting $J-I$ be rectangular with one more row than columns and writing $X(J-I) = \underline{R}U$.

We can now prove:

Lemma 2 *Residual sequences satisfy $A\underline{R} = RH$ with H an upper Hessenberg matrix with zero column sums.*

Proof: Taking equation (8), multiplying by A, and adding $-f+f$ to the lhs, gives

$$R(J-I) = A\underline{R}U \Rightarrow A\underline{R} = R(I-J)U^{-1} = RH$$

where we note that H has zero column sums as stated. We omit the proof that this is in fact an equivalence. •

Now we consider factoring the H matrix.

Lemma 3 *Let H be a Hessenberg matrix of size $(n+1)\times n$, then H has zero column sums iff its factorization is of the form*

$$H = (I-J)U.$$

We now derive the coupled recurrences form of polynomial iterative methods:

$$\begin{cases} A\underline{R} = RH \\ H \text{ has zero column sums} \end{cases} \Leftrightarrow A\underline{R} = R(I-J)D^{-1}(I+U) \Leftrightarrow \begin{cases} A\underline{P}D = R(I-J) \\ P(I+U) = R \end{cases} \tag{9}$$

In the right side of this equivalence, we recognize the traditional formulation[1]

$$r_{i+1} = r_i - Ap_i d_{ii}, \quad p_{i+1} = r_{i+1} - \sum_{j\le i} p_j u_{ji}.$$

[1] In some early literature on conjugate gradients, this is called the 'coupled two-term recurrences' form of the method, as opposed to the three-term formulation in which no search directions appear. In yet another exposition of conjugate gradients, it is considered a polynomial acceleration of a stationary iterative method.

Thus we have derived a basic form which holds for any polynomial iterative method, though some methods use an implementation that is mathematically equivalent to it. Various iterative methods, such as CG, MinRes, or BiCGstab, all follow from imposing certain conditions on R, or equivalently on the coefficients of D and U. For instance, stationary iteration and SD correspond to $U \equiv 0$; the Conjugate Gradients method corresponds to U being single upper diagonal (upper bidiagonal), with values deriving from the orthogonality of R.

In the remainder of this paper, we will take the block form (9) for given, and show how FLAME can be used to derive the coefficients in D and U in the specific case of the Conjugate Gradients method.

0.3 Applying the FLAME Methodology to the CG Algorithm

We now show how the FLAME framework can be used to derive iterative methods such as the CG algorithm. This section shows how the basic block form of the CG algorithm leads—through a systematic process—to the predicate that needs to be satisfied by the loop body of the algorithm. Such a predicate is translated into concrete instructions in the next section.

The reader should imagine the "worksheet" in Fig. 1. as being initially empty. We fill it out in the order indicated in the column marked "Step".

Step 1: Precondition and postcondition The precondition and postcondition indicate the states of the variables before and after the algorithm is executed, respectively. This includes the size the variables, as well as properties such as symmetry or invertibility.

The defining equations for iterates, residuals and search directions, under orthogonality of the residuals are given by

$$X(I-J) = PD, \quad A\underline{P}D = R(I-J), \quad P(I+U) = R, \quad R^tR = \Omega \text{ (diagonal)},$$

and we will construct the postcondition from these.

We remark that, given the quantities in D, the sequence X can be computed almost trivially from the search direction sequence P, and other quantities do not depend on it.

In addition to the above equations, which are going to form the postcondition, from them we can derive other relations. These are redundant, but can be added to the postcondition. (We will discuss the ramifications of this in section 0.5.) In particular, we derive

$$P^tA\underline{P} = (I+U)^{-t}R^tR(I-J)D^{-1} = (I+U)^{-t}\Omega(I-J)D^{-1}, \tag{10}$$

which implies that P^tAP is upper triangular (and diagonal if A is symmetric). Now, the precondition is $\{Re_0 = Ax_0 - b, \quad A \in \mathbb{R}^{m\times n}, x_0, b \in \mathbb{R}^m\}$ where $e_0 = (1,0,0,\ldots)^t$. where x_0 is an initial guess for the solution, and the postcondition is formed by combining the precondition and equations (9) and (10):

$$\left\{ \begin{array}{l} A\underline{PD} = R(I-J) \wedge \underline{PD} = X(I-J) \wedge \underline{PD} = R(I-J) \wedge P(I+U) = R \wedge \\ R^tR = \Omega \wedge P^tAP = \text{lower triangular} \wedge \\ Re_0 = Ax_0 - b, \quad A \in \mathbb{R}^{m\times n}, x_0, b \in \mathbb{R}^m \end{array} \right\}$$

This information is entered in the worksheet.

Determining the Partitioned Matrix Expression We are interested in expressing the postcondition in terms of partitioned matrices. This yields

$$\left\{ \begin{array}{l} \left(AP_LD_L \left| Ap_Md_M \right| A\underline{P}_RD_R \right) = \left(R_L \left| r_M \right| R_R \right) \left(\begin{array}{c|c|c} I_{TL} - J_{TL} & 0 & 0 \\ \hline -e_r^t & 1 & 0 \\ \hline 0 & e_0 & I_{BR} - J_{BR} \end{array} \right), \\ \left(P_LD_L \left| p_Md_M \right| \underline{P}_RD_R \right) = \left(X_L \left| x_M \right| X_R \right) \left(\begin{array}{c|c|c} I_{TL} - J_{TL} & 0 & 0 \\ \hline -e_r^t & 1 & 0 \\ \hline 0 & e_0 & I_{BR} - J_{BR} \end{array} \right), \\ \left(P_L \left| p_M \right| P_R \right) \left(\begin{array}{c|c|c} I_{TL} + U_{TL} & u_{TM} & u_{TR} \\ \hline 0 & 1 & u_{MR} \\ \hline 0 & 0 & I_{BR} + U_{BR} \end{array} \right) = \left(R_L \left| r_M \right| R_R \right), \\ \left(\begin{array}{c} R_L^t \\ \hline r_M^t \\ \hline R_R^t \end{array} \right) \left(R_L \left| r_M \right| R_R \right) = \left(\begin{array}{c|c|c} \star & 0 & 0 \\ \hline 0 & \star & 0 \\ \hline 0 & 0 & \star \end{array} \right), \left(\begin{array}{c|c|c} P_L^tAP_L & P_L^tAp_M & P_L^tAP_R \\ \hline p_M^tAP_L & p_M^tAp_M & p_M^tAP_R \\ \hline P_R^tAP_L & P_R^tAp_M & P_R^tAP_R \end{array} \right) = \left(\begin{array}{c|c|c} \star & 0 & 0 \\ \hline \star & \star & 0 \\ \hline \star & \star & \star \end{array} \right), \end{array} \right. \tag{11}$$

where $\star$ indicates that the exact value is not of interest, and $e_r = (0,\ldots,0,1)^t$.

In general, the PME is true regardless of the size of the partitioned objects. In other words, we are not splitting the operands in specific 'cut' points, but exposing part of the operands, without specifying any particular size. On the other hand, the equalities only hold if the partitionings are such that the Top-Left (and Bottom-Right) quadrants of the triangular & symmetric matrices are square.

Note our convention that upper case letters (R,P) denote matrices, lower case vectors (e,r,p), and lowercase Greek letters scalars (δ,ω).

Step 2: Loop-invariant The loop-invariant is a predicate on the state of the variables that is satisfied before and after each iteration of the loop. The Fundamental Theorem of Invariance establishes that if the loop completes, then the loop-invariant and the negation of the loop-guard hold true after the loop. This is all captured in Fig. 1.

One of the key concepts of the FLAME methodology is that of selecting a loop-invariant *a priori*, and then constructing an algorithm around it. In terms of program correctness this means that we set up a proof of correctness first, and then build an algorithm that satisfies such a proof.

To derive a loop-invariant, it is observed that while the loop executes, not all results in the Partitioned Matrix Expression (PME) have yet been achieved. Thus, the loop-invariant consists of subresults that are part of the PME. For space considerations we will not go into further detail here. The point is that there is a systematic way of choosing loop-invariants from the PME, and that choice is often non-unique (which then leads to different algoritms). We choose the loop-invariant

$$\left\{ \begin{array}{l} AP_LD_L = \left(\begin{array}{c|c} R_L & r_M \end{array} \right) \begin{pmatrix} I_{TL} - J_{TL} \\ e_r^t \end{pmatrix} \wedge P_LD_L = \left(\begin{array}{c|c} X_L & x_M \end{array} \right) \begin{pmatrix} I_{TL} - J_{TL} \\ e_r^t \end{pmatrix} \wedge \\ \left(\begin{array}{c|c} P_L & p_M \end{array} \right) \begin{pmatrix} I_{TL} + U_{TL} & u_{TM} \\ 0 & 1 \end{pmatrix} = \left(\begin{array}{c|c} R_L & r_M \end{array} \right) \wedge \\ \left(\begin{array}{c|c} R_L^t R_L & R_L^t r_M \\ \hline r_M^t R_L & r_M^t r_M \end{array} \right) = \left(\begin{array}{c|c} \star & 0 \\ \hline 0 & \star \end{array} \right) \wedge \left(\begin{array}{c|c} P_L^t A P_L & P_L^t A p_M \\ \hline p_M^t A P_L & p_M^t A p_M \end{array} \right) = \left(\begin{array}{c|c} \star & 0 \\ \hline \star & \star \end{array} \right). \end{array} \right\} \tag{12}$$

Steps 3 and 4: The loop-guard and initialization The loop-guard is the condition under which control remains in the loop. If the loop-invariant is maintained, then it will be true after the loop completes and the loop-guard will be *false*. Together these predicates must imply that the post-condition has been computed. Thus, the loop-invariant and the postcondition dictate the choice of the loop-guard. This loop-guard is given in Fig. 1.

Similarly, the loop-invariant must be true before the loop commences. Thus, an initialization, given in Step 4 of Fig. 1, is dictated by the precondition and the loop-invariant. (We note that the initial partitionings of the operands are merely indexing operations.)

Step 5: Traversing the operands The computation must make progress through the operands, so that the loop-guard will eventually be false and the loop terminates. This dictates the updates of partitionings in Steps 5a and 5b. In Step 5a we

split off one column from the 'R' block, going from a 3-way to a 4-way partitioning, for instance

$$\left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array}\right) \rightarrow \left(\begin{array}{c|c|c|c} R_0 & r_1 & r_2 & R_3 \end{array}\right);$$

after the intervening computations, in Step 5b we compact the first two partitions into the new 'L'-block, for instance

$$\left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array}\right) \leftarrow \left(\begin{array}{c|c|c|c} R_0 & r_1 & r_2 & R_3 \end{array}\right).$$

Step 6: The 'before' predicate The repartitioning of the operands in Step 5a is purely an indexing step; no computations are implied. Thus, at Step 6 (*before* the computation in Step 8) the contents of the different submatrices are still prescribed by the loop-invariant. These contents can be derived by applying the 4-way partitioning derived in Step 5a to the loop invariant (12).

This process yields what we will call the 'before' predicate:

$$P_{\text{before}}: \begin{cases} AP_0D_0 = \left(\begin{array}{c|c} R_0 & r_1 \end{array}\right)\left(\begin{array}{c} J_{00} \\ \hline j_{10}^t \end{array}\right), & \left(\begin{array}{c|c} P_0 & p_1 \end{array}\right)\left(\begin{array}{c|c} I+U_{00} & u_{01} \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} R_0 & r_1 \end{array}\right), \\ \left(\begin{array}{c} R_0^t \\ \hline r_1^t \end{array}\right)\left(\begin{array}{c|c} R_0 & r_1 \end{array}\right) = \left(\begin{array}{c|c} \star & 0 \\ \hline 0 & \star \end{array}\right), & \left(\begin{array}{c|c} P_0^tAP_0 & P_0^tAp_1 \\ \hline p_1^tAP_0 & p_1^tAp_1 \end{array}\right) = \left(\begin{array}{c|c} \star & 0 \\ \hline \star & \star \end{array}\right). \end{cases} \tag{13}$$

Step 7: The 'after' predicate At the bottom of the loop the loop-invariant must again be *true*. This means that the update in Step 8 must place the submatrices in a state were the loop-invariant is again true after the redefinition of the partitioned operands (Step 5b). The state that the submatrices must be in can be derived by substituting equivalent submatrices (as defined by Step 5b) into the loop-invariant

after which algebraic manipulation yields the desired 'after' predicate in Step 7:

$$P_{\text{after}}: \begin{cases} A\left(\begin{array}{c|c} P_0 & p_1 \end{array}\right)\left(\begin{array}{c|c} D_0 & 0 \\ \hline 0 & \delta_1 \end{array}\right) = \left(\begin{array}{c|c} R_0 & r_1 \end{array}\right)\left(\begin{array}{c|c} J_{00} & 0 \\ \hline j_{10}^t & 1 \end{array}\right) + r_2\left(\begin{array}{c|c} 0 & -1 \end{array}\right), \\ \left(\begin{array}{c|c|c} P_0 & p_1 & p_2 \end{array}\right)\left(\begin{array}{c|c|c} I+U_{00} & u_{01} & u_{02} \\ \hline 0 & 1 & \upsilon_{12} \\ \hline 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|c|c} R_0 & r_1 & r_2 \end{array}\right), \\ \left(\begin{array}{c} R_0^t \\ \hline r_1^t \\ \hline r_2^t \end{array}\right)\left(\begin{array}{c|c|c} R_0 & r_1 & r_2 \end{array}\right) = \left(\begin{array}{c|c|c} \star & 0 & 0 \\ \hline 0 & \star & 0 \\ \hline 0 & 0 & \star \end{array}\right), \\ \left(\begin{array}{c|c|c} P_0^tAP_0 & P_0^tAp_1 & P_0^TAp_2 \\ \hline p_1^tAP_0 & p_1^tAp_1 & p_1^TAp_2 \\ \hline p_2^tAP_0 & p_2^tAp_1 & p_2^TAp_2 \end{array}\right) = \left(\begin{array}{c|c|c} \star & 0 & 0 \\ \hline \star & \star & 0 \\ \hline \star & \star & \star \end{array}\right). \end{cases} \tag{14}$$

Step 8: The update Comparing the before and after predicates yields

$$\begin{aligned} P_{\text{after}} = P_{\text{before}} \; & \wedge(\delta_1 Ap_1 = r_1 - r_2) \wedge (P_0u_{02} + \upsilon_{12}p_1 + p_2 = r_2) \\ & \wedge(R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0)) \wedge (P_0^tAp_2 = 0) \wedge (p_1^TAp_2 = 0). \end{aligned}$$

The computation at Step 8 of Fig. 1 has to update and compute variables in such a way that, when executed in a state in which the 'before' predicate holds, it terminates and yields a state in which the 'after' predicate holds. In the next section we show how the actual computation again follows by a mechanical process from these predicates.

Final algorithm The described process constructs the algorithm by systematically deriving predicates that indicate the state that the variables must be in, which in turn dictates the actual computational statements. Eliminating the predicates leaves the final algorithm.

0.4 Deriving the update

There are two critical steps in the derivation of a worksheet that are less than straightforward for a more complex algorithm like the CG algorithm: choosing the loop-invariant, and identifying a set of updates of the operands that transform

the 'before' predicate into the 'after' predicate. In the previous section we showed how the loop invariant can be derived from the PME; in this section, we focus on how the update can be systematically derived. This derivation is much more systematic than in previous papers of ours that focus on dense matrix computations, for the reason that in those cases the update step was relatively obvious.

0.4.1 Deriving assignment statements

To understand the approach one must first understand some fundamental results from computer science related to the derivation of algorithms. Consider the triple $\{Q\}S\{R\}$ where Q and R are predicates indicating the state of variables and S is a command in, or segment of, the algorithm. This is known as a Hoare triple and is itself a predicate that evaluates to *true* if the command S, when initiated with variables in a state where Q is *true*, completes in a state where R is *true*. In this triple Q is the precondition and R is the postcondition. In our discussion in Section 0.3 and Fig. 1 we have seen many examples of Hoare triples and how they can be used to reason about the correctness of an algorithm. A Hoare triple can be used to assert a code segment correct. For example, $\{\chi = \eta\}\chi := \chi + 1\{\chi = \eta + 1\}$ takes on the value *true*.

The next question becomes "Under what circumstances is the Hoare triple $\{Q\}x := exp\{R\}$ *true*, where *exp* is an expression. To answer to this question the operator $\mathrm{wp}(S,R)$ is introduced: this returns the *weakest precondition* (least restrictive predicate) that describes the state of variables such that if the statement S is executed, then this command completes in a state where R is *true*. Now, $\{Q\}S\{R\}$ if and only if Q implies $\mathrm{wp}(S,R)$.

For sequences of statements $\{Q_0\}S_0;S_1\{R\}$ we introduce an intermediate predicate

$$\{Q_0\}S_0;S_1\{R\} = \{Q_0\}S_0\{Q_1\} \wedge \{Q_1\}S_1\{R\}.$$

From this we see that

$$Q_0 = \mathrm{wp}(S_0;S_1,R) = \mathrm{wp}(S_0,\{Q_1\}) = \mathrm{wp}(S_0,\mathrm{wp}(S_1,R)).$$

Inductively, if we wish to find a sequence of statements $S_0;S_1;\ldots;S_{k-1}$ such that $\{Q\}S_0;S_1;\ldots;S_{k-1}\{R\}$ then Q must imply

$$\mathrm{wp}(S_0;S_1;\ldots;S_{k-1},R) = \mathrm{wp}(S_0,\mathrm{wp}(S_1,\ldots,\mathrm{wp}(S_{k-1},R)\ldots)).$$

We can summarize this by noting that the following must be *true*:

$$\{Q \Rightarrow \mathrm{wp}(S_0,Q_1)\}S_0\{Q_1 = \mathrm{wp}(S_1,Q_2)\}S_1\{Q_2 = \mathrm{wp}(S_2,Q_3)\}\ldots\{Q_{k-1} = \mathrm{wp}(S_{k-1},R)\}S_{k-1}\{R\}$$

Finally, we recall that wp("$x := exp$", R) equals the predicate R with all instances of x replaced by the expression exp. For example, wp("$x := x+1$", $x = y+4$) $= \{(x+1) = y+4\} = \{x = y+3\}$.

0.4.2 Application to the CG algorithm

The above theory can be used to derive the update (step 8) in Fig. 1. The idea is that we wish to determine expressions $exp_0, \ldots, exp_4$ such that[2]

$\{P_{\text{before}}\}$
$\{Q_0 = \text{wp}(\text{"}\delta_1 := exp_0\text{"}, Q_1)\}$
$S_0 : \delta_1 = exp_0$
$\{Q_1 = \text{wp}(\text{"}r_2 := exp_1\text{"}, Q_2)\}$
$S_1 : r_2 := exp_1$
$\{Q_2 = \text{wp}(\text{"}x_2 := exp_2\text{"}, Q_3)\}$
$S_2 : x_2 := exp_2$
$\{Q_3 = \text{wp}(\text{"}u_{02} := exp_3\text{"}, Q_4)\}$
$S_3 : u_{02} := exp_3$
$\{Q_4 = \text{wp}(\text{"}\upsilon_{12} := exp_4\text{"}, Q_5)\}$
$S_4 : \upsilon_{12} := exp_4$
$\{Q_5 = \text{wp}(\text{"}p_2 := exp_5\text{"}, P_{\text{after}})\}$
$S_5 : p_2 := exp_5$

$$\left\{ \begin{array}{ll} P_{\text{after}} = P_{\text{before}} & \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (\delta_1 p_1 = x_1 - x_2) \\ & \wedge(P_0 u_{02} + \upsilon_{12} p_1 + p_2 = r_2) \wedge (P_0^t A p_2 = 0) \wedge (p_1^t A p_2 = 0) \\ & \wedge(R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (r_2^t r_2 = \omega_2) \end{array} \right\}$$

Now,

$$\begin{array}{rcl} Q_5 & = & \text{wp}(\text{"}p_2 := exp_5\text{"}, P_{\text{after}}) \\ & = & \left\{ \begin{array}{ll} P_{\text{before}} & \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (\delta_1 p_1 = x_1 - x_2) \\ & \wedge(P_0 u_{02} + \upsilon_{12} p_1 + exp_5 = r_2) \wedge (P_0^t A exp_5 = 0) \wedge (p_1^t A exp_5 = 0) \\ & \wedge(R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (r_2^t r_2 = \omega_2) \end{array} \right\} \end{array}$$

[2] In section 0.5 we will address the fact that we need not lay out explicitly the sequence in which quantities are to be computed.

from which we deduce that $exp_5 = r_2 - P_0 u_{02} - \upsilon_{12} p_1$ and

$$\begin{aligned} Q_5 &= \mathrm{wp}(\text{"}p_2 := r_2 - P_0 u_{02} - \upsilon_{12} p_1\text{"}, P_{\text{after}}) \\ &= \left\{ \begin{array}{ll} P_{\text{before}} & \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (\delta_1 p_1 = x_1 - x_2) \\ & \wedge T \wedge (P_0^t A(r_2 - P_0 u_{02} - \upsilon_{12} p_1) = 0) \wedge (p_1^t A(r_2 - P_0 u_{02} - \upsilon_{12} p_1) = 0) \\ & \wedge (R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (r_2^t r_2 = \omega_2) \end{array} \right\} \\ &= \left\{ \begin{array}{ll} P_{\text{before}} & \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (\delta_1 p_1 = x_1 - x_2) \\ & \wedge (P_0^t A r_2 - P_0^t A P_0 u_{02} = 0) \wedge (p_1^t A r_2 - p_1^t A P_0 u_{02} - p_1^t A \upsilon_{12} p_1 = 0) \\ & \wedge (R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (r_2^t r_2 = \omega_2) \end{array} \right\} \end{aligned}$$

Similarly, we determine $\upsilon_{12} := \exp_4 = (p_1^t A r_2 - p_1^t A P_0 u_{02}) / p_1^t A p_1$ and

$$\begin{aligned} Q_4 &= \mathrm{wp}(\text{"}\upsilon_{12} := (p_1^t A r_2 - p_1^t A P_0 u_{02}) / p_1^t A p_1\text{"}, Q_5) \\ &= \left\{ \begin{array}{ll} P_{\text{before}} & \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (\delta_1 p_1 = x_1 - x_2) \\ & \wedge (P_0^t A r_2 - P_0^t A P_0 u_{02} = 0) \wedge T \\ & \wedge (R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (r_2^t r_2 = \omega_2) \end{array} \right\} \end{aligned}$$

Next we can determine $u_{02} := \exp_3 = (P_0^t A P_0)^{-1} P_0^t A r_2$ and

$$\begin{aligned} Q_3 &= \mathrm{wp}(\text{"}u_{02} := (P_0^t A P_0)^{-1} P_0^t A r_2\text{"}, Q_3) \\ &= \left\{ \begin{array}{ll} P_{\text{before}} & \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (\delta_1 p_1 = x_1 - x_2) \\ & \wedge T \\ & \wedge (R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (r_2^t r_2 = \omega_2) \end{array} \right\} \end{aligned}$$

followed by $x_2 := \exp_2 = x_1 - \delta_1 p_1$, giving

$$\begin{aligned} Q_2 &= \mathrm{wp}(\text{"}x_2 := x_1 - \delta_1 p_1\text{"}, Q_3) \\ &= \left\{ P_{\text{before}} \quad \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (r_2^t r_2 = \omega_2) \right\} \end{aligned}$$

and $r_2 := \exp_0 = r_1 - \delta_1 A p_1$ giving

$$\begin{aligned} Q_1 &= \mathrm{wp}(\text{"}r_2 := r_1 - \delta_1 A p_1\text{"}, Q_2) \\ &= \left\{ P_{\text{before}} \quad \wedge(R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (r_2^t r_2 = \omega_2) \right\} \end{aligned}$$

(where $R_0^t r_1 = 0$ is part of the 'before' equations and $R_0^t A p_1 = 0$ can be derived from them) and finally $\delta_1 := \exp_0 = r_1^t r_1 / r_1^t A p_1$ and

$$Q_0 = \mathrm{wp}(\text{"}\delta_1 := r_1^t r_1 / r_1^t A p_1\text{"}, Q_1) = \{ P_{\text{before}} \quad \wedge T \}$$

so that P_{before} implies Q_0, as required.

The updates of the variables can then be entered as Step 8 in Figure 1.

Step	**Annotated Algorithm:** Compute X,R,P,D,U of size $m \times n$ so that $A\underline{PD} = R(I-J)$, $\underline{PD} = X(I-J)$, $P(I+U) =$
4	$R \rightarrow \left(\begin{array}{c\|c\|c} R_L & r_M & R_R \end{array}\right)$, P similarly; $J \rightarrow \left(\begin{array}{c\|c\|c} J_{TL} & 0 & 0 \\ \hline e_r^t & 0 & 0 \\ \hline 0 & e_0 & J_{BR} \end{array}\right)$, U,D similarly
3	**while** $n(R_R) > 0$ **do**
5a	$\left(\begin{array}{c\|c\|c} R_L & r_M & R_R \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c\|c} R_0 & r_1 & r_2 & R_3 \end{array}\right)$, P similar; $\left(\begin{array}{c\|c\|c} J_{TL} & 0 & 0 \\ \hline j_{ML}^t & 0 & 0 \\ \hline 0 & j_{MR} & J_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c\|c} J_{00} & 0 & 0 & 0 \\ \hline e_r^t & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & e_0 & J_{33} \end{array}\right)$, U,D similar
8	$d_1 \leftarrow r_1^t r_1 / r_1^t A p_1$, $r_2 \leftarrow r_1 - A p_1 d_1$, $u_{12} = r_2^t r_2 / r_1^t r_1$, $p_2 \leftarrow r_2 - p_1 u_{12}$
5b	$\left(\begin{array}{c\|c\|c} R_L & r_M & R_R \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c\|c} R_0 & r_1 & r_2 & R_3 \end{array}\right)$, P similar; $\left(\begin{array}{c\|c\|c} J_{TL} & 0 & 0 \\ \hline j_{ML}^t & 0 & 0 \\ \hline 0 & j_{MR} & J_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c\|c} J_{00} & 0 & 0 & 0 \\ \hline e_r^t & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & e_0 & J_{33} \end{array}\right)$, U,D similar
	endwhile

Figure 1: Worksheet for the Conjugate Gradient method

0.4.3 More about the U coefficients

Computing the U coefficients; symmetric case Without stressing the fact, in the previous section we derived a CG method for nonsymmetric systems; ordinarily the symmetric case is the first one derived. In our method of derivation, the symmetric case needs another step of reasoning on the PME level.

We concluded in equation (10) that P^tAP is upper triangular in general; combined with symmetry of A, this means that P^tAP is diagonal in the symmetric case. Also,

$$P^tA\underline{P} = (I+U)^{-t}\Omega(I-J)D^{-1}$$

tells us that $(I+U)^{-t}$ times a lower bidiagonal matrix is diagonal, hence that $I+U$ is upper bidiagonal. Therefore, the u_{02} coefficients no longer appear and we only need to compute υ_{12}. We construct a sequence of update statements and weakest preconditions to fulfill

$$\left\{\begin{array}{ll} P_{\text{after}} = P_{\text{before}} & \wedge(\delta_1 Ap_1 = r_1 - r_2) \wedge (\delta_1 p_1 = x_1 - x_2) \\ & \wedge(\upsilon_{12}p_1 + p_2 = r_2) \wedge (P_0^tAp_2 = 0) \wedge (p_1^tAp_2 = 0) \\ & \wedge(R_0^tr_2 = 0) \wedge (r_1^tr_2 = 0) \wedge (r_2^tr_2 = \omega_2) \end{array}\right\}$$

As before, we deduce that $exp_4 = r_2 - p_1\upsilon_{12}$ and

$$\begin{array}{rcll} Q_4 & = & \text{wp}(\text{"}p_2 := r_2 - p_1\upsilon_{12}\text{"}, P_{\text{after}}) & \\ & = & \{P_{\text{before}} & \wedge(\delta_1 Ap_1 = r_1 - r_2) \wedge T \wedge (R_0^tr_2 = 0) \wedge (r_1^tr_2 = 0)) \\ & & & \wedge(P_0^tA(r_2 - p_1\upsilon_{12}) = 0) \wedge (p_1^tA(r_2 - p_1\upsilon_{12}) = 0)\} \\ & = & \{P_{\text{before}} & \wedge(\delta_1 Ap_1 = r_1 - r_2) \wedge (R_0^tr_2 = 0) \wedge (r_1^tr_2 = 0)) \\ & & & \wedge(P_0^tAr_2 = 0) \wedge (p_1^tAr_2 - p_1\upsilon_{12}^tAp_1 = 0)\} \end{array}$$

Similarly, we can determine $\upsilon_{12} := \text{exp}_3 = p_1^tAr_2/p_1^tAp_1$ and

$$\begin{array}{rcll} Q_3 & = & \text{wp}(\text{"}\upsilon_{12} := p_1^tAr_2/p_1^tAp_1\text{"}, Q_4) & \\ & = & \{P_{\text{before}} & \wedge(\delta_1 Ap_1 = r_1 - r_2) \wedge (R_0^tr_2 = 0) \wedge (r_1^tr_2 = 0)) \wedge P_0^tAr_2 = 0 \wedge T\} \end{array}$$

Elementary manipulation of identities gives

$$\upsilon_{12} = r_2^tr_2/r_1^tr_1.$$

The remaining steps are identical as in the nonsymmetric case.

0.5 Discussion and Conclusion

At first glance, the reader may conclude that the presented extension of the FLAME framework merely provides a 'mental discipline' for deriving known Krylov subspace based algorithms. While this may become a major contribution of the project, we believe it shows a lot more promise than just that.

The reader may have already noticed that there are a number of decisions that were made that led to the derived algorithm. Let us itemize some of these decisions and discuss how different choices will lead to a rich family of algorithms, both differing in mathematical respects and in performance aspects.

- **The governing equation.** In Section 0.2, we started with the governing equation

$$\begin{cases} A\underline{P}D = R(I-J) \\ P(I+U) = R \end{cases}$$

  The additional equation $R^tR = \Omega$ (diagonal) represents one choice of constraints that can be enforced on the residual vectors. As mentioned, different choices lead to different known methods, such as Steepest Descent or GMRES.

  We believe the presented methodology will be able to clarify how all these methods are related, but drawing up the constraints is work still to be undertaken. Our framework will make it far easier for a human expert to derive new algorithms, since only the basic notion (orthogonality of the residuals in the CG case) needs to be specified on top of the basic equations: the derivation of the actual constants is done through a systematic, indeed automatable, process.

- **PME manipulation.** Even within the context of a single method such as CG, manipulation of the PME can be interesting. We already saw this mechanism in action when equation (10), which is not strictly part of the definition of the CG method, was added. In [6], this mechanism was used to argue that our approach can discover variant algorithmic variants that combine inner products [4, 5].

- **Choices of invariants.** The governing equation leads to a PME, which is a recursive definition of the operation. But for each PME there are multiple possible loop-invariants. Some of these may lead to uncomputable formulations; other may lead to distinct algorithms that may or may not have desirable properties for a given situation (see [6] for an example).

- **How to choose.** Given that we expect a large family of algorithms to result from the ultimate approach, we need to develop a way of determining which algorithm is most appropriate for a given situation. Measures of "goodness" could include computational cost, numerical stability, rate of convergence, or ability to reduce communication cost on, for example, a distributed memory parallel architecture. There is a distinct possibility of reasoning about such factors in our framework, which we are undertaking in separate research.

We conclude that our framework supports a vision for exploration of Krylov subspace methods as a coherent family of algorithms, as well as the derivation of proved correct library software. The discussion above shows that the space to be explored is large, which is where mechanization becomes an important part of the solution. How to achieve mechanization of derivation for dense matrix computations was the subject of the dissertation of one of the authors [2]. His system will need to be expanded to achieve what we propose. In other words, there is a lot of interesting research ahead of us.

Acknowledgments

This work was sponsored by NSF through awards CCF 0917096 and OCI-0850750, and by grant GSC 111 of the Deutsche Forschungsgemeinschaft (German Research Association). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

Bibliography

[1] Steven F. Ashby, Thomas A. Manteuffel, and Paul E. Saylor. A taxonomy for conjugate gradient methods. *SIAM J. Numer. Anal.*, 27:1542–1568, 1990.

[2] Paolo Bientinesi. *Mechanical Derivation and Systematic Analysis of Correct Linear Algebra Algorithms*. PhD thesis, The University of Texas at Austin, Department of Computer Sciences, 2006.

[3] Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Transactions on Mathematical Software*, 2005. to appear.

[4] A. Chronopoulos and C.W. Gear. *s*-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25:153–168, 1989.

[5] E.F. D'Azevedo, V.L. Eijkhout, and C.H. Romine. A matrix framework for conjugate gradient methods and some variants of cg with less synchronization overhead. In *Proceedings of the Sixth SIAM Conference on Parallel Procesing for Scientific Computing*, pages 644–646, Philadelphia, 1993. SIAM.

[6] Victor Eijkhout, Paolo Bientinesi, and Robert van de Geijn. Formal derivation of Krylov methods. Technical Report TR-08-03, Texas Advanced Computing Center, The University of Texas at Austin, 2008.

[7] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. Flame: Formal linear algebra methods environment. *ACM Transactions on Mathematical Software*, 27(4):422–455, December 2001.

[8] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Nat. Bur. Stand. J. Res.*, 49:409–436, 1952.

[9] Alston S. Householder. *The theory of matrices in numerical analysis*. Blaisdell Publishing Company, New York, 1964. republished by Dover Publications, New York, 1975.

[10] Kang C. Jea and David M. Young. Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods. *Lin. Alg. Appl.*, 34:159–194, 1980.

[11] John Gregg Lewis and Ronald G. Rehm. The numerical solution of a nonseperable elliptic partial differential equation by preconditioned conjugate gradients. *J. Res. Nat. Bureau of Standards*, 85:367–390, 1980.

[12] Robert A. van de Geijn and Enrique S. Quintana-Ortí. *The Science of Programming Matrix Computations*. `www.lulu.com`, 2008.

[13] P.K.W. Vinsome. ORTHOMIN, an iterative method for solving sparse sets of simultaneous linear equations, paper SPE 5729. In *4th Symposium of Numerical Simulation of Reservoir Performance of the Society of Petroleum Engineers of the AIME, Los Angeles, 19–20 February 1976*.