Aachen Institute for Advanced Study in Computational Engineering Science

Preprint: AICES-2010/04-1

03/April/2010

Towards Mechanical 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

Towards Mechanical Derivation of Krylov Solver Libraries

Victor Eijkhout*, Paolo Bientinesi†, and Robert van de Geijn‡

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 framework named FLAME (Formal Linear Algebra Methods Environment) has been developed to realize this aim. The goals of this paper are two-fold: first, we show how the approach can be used to derive a nonsymmetric CG algorithm, which provides strong evidence that the approach applies to Krylov subspace methods in general. Secondly, and more importantly, we show that the reasoning can be made sufficiently systematic that mechanical derivation is within reach. Thus this research shows a promise for providing necessary building blocks towards automatic generation of library software for iterative methods for solving linear systems.

1. Introduction

We show how the FLAME methodology [7, 11] can be used to derive the Conjugate Gradients (CG) algorithm [8]. We use this example to make the case that a FLAME-based environment for deriving, and reasoning about, new iterative methods and implementations is a distinct, and attractive, possibility.

CG and related methods for solving a linear system $Ax = b$ are iterative processes, computing a new approximation x_i to the solution in each i-th iteration. This process involves complicated computations of (scalar) quantities that follow from conditions, such as orthogonality of the residuals $r_i = Ax_i - b$. In this paper we will show how a very high level description of the algorithm leads by a mechanical process, first to a predicate to be satisfied by the essential loop body of the algorithm, then, driven by the correctness proof of this loop body, to the actual instructions.

Our paper proceeds as follows.

In Section 2, we present a block formalism for iterative methods, first proposed by Householder [9]. This casts iterative methods as a set of governing conditions on the matrices that express the vector sequences, which makes it amenable to the FLAME approach for deriving algorithms. For this, in Section 2 we extend the FLAME notation slightly.

Traditional expositions of the CG method, and other Krylov methods, posit the basic form of relations between matrices and vectors, and compute the scalar coefficients in them by 'lengthy induction arguments' [10]. In Section 3 we show how the FLAME methodology dispenses with these induction proofs, and how it can be applied to a representative computation yielding a nonsymmetric CG algorithm.

Finally, in Section 4 we present a systematic framework for determining how variables must be defined as part of the computation. Our derivation is driven by the construction of Hoare triples that together constitute a correctness proof of the algorithm derived.

Together these insights support the vision of systematic and, ultimately, mechanical derivation of Krylov subspace methods libraries. In this paper we derive some existing algorithms, leading to the prospect of automatically generated, proved correct, library software[4]. Our methodology holds the fur-

*Texas Advanced Computing Center, The University of Texas at Austin
†RWTH-Aachen, Germany
‡Department of Computer Science, The University of Texas at Austin

[4]This prospect is already largely realized through FLAME for dense direct matrix algorithms [3].

ther prospect of easy experimentation with new methods and derivation of computationally advantageous variants of old ones.

For readers familiar with the FLAME concept, we remark that this research marks several new ideas. First of all, our basic partitioning scheme is 3-way, rather than 2-way. Next, our Partitioned Matrix Expression (PME) is not only more elaborate than in previous research, we introduce the device of augmenting it with new equations that are not part of the original problem definition. Finally, the update step derived gives only implicit definition of the quantities to be computed; the actual computation requires its own correctness proof.

2. Theory and Notation

We start by presenting, in this section, the basic form of iterative methods such as the CG method in a block formalism [1, 9]. Then we phrase this block formalism in terms of the FLAME framework. Our formulation will be quite general, giving a basic form that holds for all polynomial iterative methods. The specific computations in the algorithm will follow in a later section from orthogonality requirements.

2.1. 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 *dof*s will then be derived in subsequent sections.

We need the basic concept of a 'Krylov sequence', which, given a square matrix A and initial vector k_0, is defined as the matrix with n columns $K\langle A,k_0\rangle \equiv \left(\begin{array}{c|c|c|c|c} k_0 & Ak_0 & A^2k_0 & \cdots & A^{n-1}k_0 \end{array} \right)$. In other words, the jth column of $K\langle A,k_0\rangle$, k_j, is defined by the recurrence

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

which we can write as $A\underline{K} = KJ$ where the underline means that the last column is omitted, and J is defined as the $n \times (n-1)$ matrix with entries

$$J_{ij} = \begin{cases} 1 & \text{if } i = j+1, \\ 0 & \text{otherwise.} \end{cases}$$

We now state the coupled recurrences form of polynomial iterative methods. In block form they are

$$\begin{cases} A\underline{P}D = R(I-J) \\ P(I+U) = R \end{cases} \qquad \text{and in scalar form} \qquad \begin{cases} r_{i+1} = r_i - Ap_i\delta_{ii}, \\ p_{i+1} = r_{i+1} - \sum_{j\leq i} p_j\upsilon_{ji}. \end{cases} \tag{1}$$

where R are residuals, P search directions, D a diagonal matrix, and U strictly upper triangular. This form can be derived from 'first principles' (for full details, see [6]), but anyone familiar with iterative methods will recognize that they mostly conform to this scheme, testifying to its generality. The different iterative methods that exist (CG, MinRes, BiCGstab) all follow from imposing certain conditions on R, or equivalently on the coefficients of D and U. For instance, stationary iteration and steepest descent correspond to $U \equiv 0$; the CG 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 of (1) as our starting point, and show how FLAME can be used to derive the vector recurrences as well as the coefficients in D and U.

2.2. FLAME Notation for Representing Krylov Subspace Methods

In this section, we use the CG iteration to motivate "index-free" notation that in Section 3 will allow us to systematically derive the algorithm.

Step	**Annotated Algorithm:** Compute R, P, D, U as given in Eqn. (2)
1a	$\{$*precondition holds*$\}$
4	$R \rightarrow \left(\begin{array}{c\|c\|c} R_L & r_M & R_R \end{array}\right), P \rightarrow \left(\begin{array}{c\|c\|c} P_L & p_M & P_R \end{array}\right),$ $J \rightarrow \left(\begin{array}{c\|c\|c} J_{TL} & 0 & 0 \\ \hline e_r^t & 0 & 0 \\ \hline 0 & e_0 & J_{BR} \end{array}\right), U \rightarrow \left(\begin{array}{c\|c\|c} U_{TL} & u_{TM} & U_{TR} \\ \hline 0 & 0 & u_{MR}^T \\ \hline 0 & 0 & U_{BR} \end{array}\right), d \rightarrow \left(\begin{array}{c} d_T \\ \hline \delta_M \\ \hline d_B \end{array}\right)$
2	$\{$*loop-invariant holds*$\}$
3	**while** $n(R_R) > 0$ **do**
2,3	$\{$ (*loop-invariant holds*) $\wedge$ ($n(R_R) > 0$) $\}$
5a	$\left(\begin{array}{c\|c\|c} R_L & r_M & R_R \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c\|c} R_0 & r_1 & r_2 & R_3 \end{array}\right), \left(\begin{array}{c\|c\|c} P_L & p_M & P_R \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c\|c} P_0 & p_1 & p_2 & P_3 \end{array}\right),$ $\left(\begin{array}{c\|c\|c} J_{TL} & 0 & 0 \\ \hline j_{ML}^t & 0 & 0 \\ \hline 0 & j_{MR} & J_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c\|c} J_{00} & 0 & 0 & 0 \\ \hline e_r^t & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & e_0 & J_{33} \end{array}\right),$ $\left(\begin{array}{c\|c\|c} U_{TL} & u_{TM} & U_{TR} \\ \hline 0 & 0 & u_{MR}^T \\ \hline 0 & 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c\|c} U_{00} & u_{01} & u_{02} & U_{03} \\ \hline 0 & 0 & \upsilon_{12} & u_{13}^T \\ \hline 0 & 0 & 0 & u_{23}^T \\ \hline 0 & 0 & 0 & U_{33} \end{array}\right), \left(\begin{array}{c} d_T \\ \hline \delta_M \\ \hline d_B \end{array}\right) \rightarrow \left(\begin{array}{c} d_0 \\ \hline \delta_1 \\ \hline \delta_2 \\ \hline d_3 \end{array}\right)$
6	$\{$*before predicate holds*$\}$
8	S_0: $d_1 := r_1^t r_1 / r_1^t A p_1$ S_1: $r_2 := r_1 - A p_1 d_1$ S_2: $u_{02} := (P_0^t A P_0)^{-1} P_0^t A r_2$ S_3: $\upsilon_{12} := (p_1^t A p_1)^{-1} (p_1^t A r_2 - p_1^t A P_0 u_{02})$ S_4: $p_2 := r_2 - P_0 u_{02} + p_1 \upsilon_{12}$
7	$\{$*after predicate holds*$\}$
5b	$\left(\begin{array}{c\|c\|c} R_L & r_M & R_R \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c\|c} R_0 & r_1 & r_2 & R_3 \end{array}\right), \left(\begin{array}{c\|c\|c} P_L & p_M & P_R \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c\|c} P_0 & p_1 & p_2 & P_3 \end{array}\right),$ $\left(\begin{array}{c\|c\|c} J_{TL} & 0 & 0 \\ \hline j_{ML}^t & 0 & 0 \\ \hline 0 & j_{MR} & J_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c\|c} J_{00} & 0 & 0 & 0 \\ \hline e_r^t & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & e_0 & J_{33} \end{array}\right),$ $\left(\begin{array}{c\|c\|c} U_{TL} & u_{TM} & U_{TR} \\ \hline 0 & 0 & u_{MR}^T \\ \hline 0 & 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c\|c} U_{00} & u_{01} & u_{02} & U_{03} \\ \hline 0 & 0 & \upsilon_{12} & u_{13}^T \\ \hline 0 & 0 & 0 & u_{23}^T \\ \hline 0 & 0 & 0 & U_{33} \end{array}\right), \left(\begin{array}{c} d_T \\ \hline \delta_M \\ \hline d_B \end{array}\right) \leftarrow \left(\begin{array}{c} d_0 \\ \hline \delta_1 \\ \hline \delta_2 \\ \hline d_3 \end{array}\right)$
2	$\{$*loop-invariant holds*$\}$
	endwhile
2,3	$\{$ (*loop-invariant holds*) $\wedge \neg$ ($n(R_R) > 0$) $\}$
1b	$\{$*postcondition holds*$\}$

Figure 1. Algorithm the nonsymmetric Conjugate Gradients method presented as part of a "worksheet".

Let us consider the nonsymmetric CG iteration[5] given in Figure 1. In that figure, for the moment ignore the column marked by "Step" and the gray-shaded boxes. To understand the notation used to present the algorithm, it suffices to note that $\left(\begin{array}{c\|c\|c} R_L & r_M & R_R \end{array}\right)$ represents matrix R partitioned into three key components: the "old" residual vectors (R_L), the most recently computed residual vector (r_M), and the residual vectors yet to be computed (R_R). At the top of the loop, the next residual vector to be

[5] See [6] for a similar derivation of the (symmetric) Hestenes-Stiefel CG method.

computed is exposed as r_2, and at the bottom of the loop that newly computed vector becomes r_M. Thus, thick and thin lines are used to indicate movement through the matrices. Vector d represents the diagonal of diagonal matrix D. Vectors e_0 and e_r correspond to the zero vector with a 1 in the first and last entry, respectively.

Careful examination reveals that the so presented algorithm exposes the relationship between the computations of the CG algorithm and the matrices in Eqn. (2).

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. This 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.

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

$$X(I-J)=PD, \quad A\underline{P}D=R(I-J), \quad P(I+U)=R, \quad R^tR=\Omega \text{ (diagonal)} \tag{2}$$

We will ignore the first equation, since X can be computed almost trivially from the search direction sequence P, and other quantities do not depend on it. From the above equations we can derive other relations, in particular

$$P^tA\underline{P}=(I+U)^{-t}R^tR(I-J)D^{-1}=(I+U)^{-t}\Omega(I-J)D^{-1}, \tag{3}$$

which implies that P^tAP is upper triangular (and diagonal if A is symmetric). Now, the precondition is $\{Re_\ell = Ax_0 - b\}$ where x_0 is an initial guess for the solution, and the postcondition is formed by combining the precondition and equations (2) and (3):

$$\{A\underline{PD}=R(I-J)\wedge P(I+U)=R\wedge R^tR=\Omega\wedge P^tAP=\text{lower triangular}\wedge Re_\ell=Ax_0-b.\}$$

This information is entered in the worksheet.

Determining the Partitioned Matrix Expression (PME)

We are interested in expressing the postcondition in terms of partitioned matrices. This yields

$$\begin{cases} \left(\begin{array}{c|c|c} AP_LD_L & Ap_Md_M & A\underline{P}_RD_R \end{array}\right) = \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array}\right) \left(\begin{array}{c|c|c} I_{TL}-J_{TL} & 0 & 0 \\ \hline -e_r^t & 1 & 0 \\ \hline 0 & e_0 & I_{BR}-J_{BR} \end{array}\right), \\ \left(\begin{array}{c|c|c} P_L & p_M & P_R \end{array}\right) \left(\begin{array}{c|c|c} I_{TL}+U_{TL} & u_{TM} & u_{TR} \\ \hline 0 & 1 & u_{MR} \\ \hline 0 & 0 & I_{BR}+U_{BR} \end{array}\right) = \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array}\right), \\ \left(\begin{array}{c} R_L^t \\ \hline r_M^t \\ \hline R_R^t \end{array}\right) \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array}\right) = \left(\begin{array}{c|c|c} \star & 0 & 0 \\ \hline 0 & \star & 0 \\ \hline 0 & 0 & \star \end{array}\right), \left(\begin{array}{c|c|c} P_L^tAP_L & P_L^tAp_M & P_L^tAP_R \\ \hline p_M^tAP_L & p_M^tAp_M & p_M^tAP_R \\ \hline P_R^tAP_L & P_R^tAp_M & P_R^tAP_R \end{array}\right) = \left(\begin{array}{c|c|c} \star & 0 & 0 \\ \hline \star & \star & 0 \\ \hline \star & \star & \star \end{array}\right), \end{cases} \tag{4}$$

where $\star$ indicates that the exact value is not of interest. 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 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

$$\begin{array}{l} AP_LD_L = R_L(I_{TL} - J_{TL}) - r_M e_r^t \wedge \left(\begin{array}{c|c|c} (I_{TL}+U_{TL})P_L & P_L u_{TM} + p_M \end{array} \right) = \left(\begin{array}{c|c|c} R_L & r_M \end{array} \right) \wedge \\ \left(\begin{array}{c|c} R_L^t R_L & R_L^t r_M \\ \hline r_M^t R_L & r_M^t r_M \end{array} \right) = \left(\begin{array}{c|c} \star & 0 \\ \hline 0 & \star \end{array} \right) \wedge \left(\begin{array}{c|c} P_L^t A P_L & P_L^t A p_M \\ \hline p_M^t A P_L & p_M^t A p_M \end{array} \right) = \left(\begin{array}{c|c} \star & 0 \\ \hline \star & \star \end{array} \right). \end{array} \tag{5}$$

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 loop invariant (5) to the 4-way partitionin derived in step 5a.

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

$$P_{\text{before}}: \begin{cases} AP_0D_0 = \left(\begin{array}{c|c} R_0 & r_1 \end{array}\right)\left(\begin{array}{c|c} J_{00} & \\ \hline j_{10}^t & \end{array}\right), \qquad \left(\begin{array}{c|c} P_0 & p_1 \end{array}\right)\left(\begin{array}{c|c} I+U_{00} & u_{01} \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} R_0 & r_1 \end{array}\right), \\ \left(\begin{array}{c} R_0^t \\ \hline r_1^t \end{array}\right)\left(\begin{array}{c|c} R_0 & r_1 \end{array}\right) = \left(\begin{array}{c|c} \star & 0 \\ \hline 0 & \star \end{array}\right), \quad \left(\begin{array}{c|c} P_0^tAP_0 & P_0^tAp_1 \\ \hline p_1^tAP_0 & p_1^tAp_1 \end{array}\right) = \left(\begin{array}{c|c} \star & 0 \\ \hline \star & \star \end{array}\right). \end{cases} \tag{6}$$

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:

$$P_{\text{after}}: \begin{cases} A\left(\begin{array}{c|c} P_0 & p_1 \end{array}\right)\left(\begin{array}{c|c} D_0 & 0 \\ \hline 0 & \delta_1 \end{array}\right) = \left(\begin{array}{c|c} R_0 & r_1 \end{array}\right)\left(\begin{array}{c|c} J_{00} & 0 \\ \hline j_{10}^t & 1 \end{array}\right) + r_2\left(\begin{array}{c|c} 0 & -1 \end{array}\right), \\ \left(\begin{array}{c|c|c} P_0 & p_1 & p_2 \end{array}\right)\left(\begin{array}{c|c|c} I+U_{00} & u_{01} & u_{02} \\ \hline 0 & 1 & \upsilon_{12} \\ \hline 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|c|c} R_0 & r_1 & r_2 \end{array}\right), \\ \left(\begin{array}{c} R_0^t \\ \hline r_1^t \\ \hline r_2^t \end{array}\right)\left(\begin{array}{c|c|c} R_0 & r_1 & r_2 \end{array}\right) = \left(\begin{array}{c|c|c} \star & 0 & 0 \\ \hline 0 & \star & 0 \\ \hline 0 & 0 & \star \end{array}\right), \\ \left(\begin{array}{c|c|c} P_0^tAP_0 & P_0^tAp_1 & P_0^TAp_2 \\ \hline p_1^tAP_0 & p_1^tAp_1 & p_1^TAp_2 \\ \hline p_2^tAP_0 & p_2^tAp_1 & p_2^TAp_2 \end{array}\right) = \left(\begin{array}{c|c|c} \star & 0 & 0 \\ \hline \star & \star & 0 \\ \hline \star & \star & \star \end{array}\right). \end{cases} \tag{7}$$

Step 8: The update

Comparing the before and after predicates yields

$$\begin{aligned} P_{\text{after}} = P_{\text{before}} \; & \wedge(\delta_1 Ap_1 = r_1 - r_2) \wedge (P_0u_{02} + \upsilon_{12}p_1 + p_2 = r_2) \\ & \wedge(R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0)) \wedge (P_0^t Ap_2 = 0) \wedge (p_1^T Ap_2 = 0). \end{aligned}$$

Step 8 of Fig. 1 has to update and compute variables in such a way that, given the 'before' predicate, the 'after' predicate will hold. 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.

4. Deriving the update

There are two critical steps in the above derivation 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 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.

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

4.2. Application to the CG algorithm

The above theory can be used to derive the update in Fig. 1. The idea is that we wish to determine expressions $exp_0,\ldots,exp_4$ such that[6]

$$
\begin{array}{l}
\{P_{\text{before}}\} \\
\{Q_0 = \mathrm{wp}("\delta_1 := exp_0", Q_1)\} \\
S_0 : \delta_1 := exp_0 \\
\{Q_1 = \mathrm{wp}("r_2 := exp_1", Q_2)\} \\
S_1 : r_2 := exp_1 \\
\{Q_2 = \mathrm{wp}("u_{02} := exp_2", Q_3)\} \\
S_2 : u_{02} := exp_2 \\
\{Q_3 = \mathrm{wp}("\upsilon_{12} := exp_3", Q_4)\} \\
S_3 : \upsilon_{12} := exp_3 \\
\{Q_4 = \mathrm{wp}("p_2 := exp_4", P_{\text{after}})\} \\
S_4 : p_2 := exp_4 \\
\left\{ \begin{array}{ll} P_{\text{after}} = P_{\text{before}} & \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (P_0 u_{02} + \upsilon_{12} p_1 + p_2 = r_2) \\ & \wedge(R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (r_2^t r_2 = \omega_2) \wedge (P_0^t A p_2 = 0) \wedge (p_1^t A p_2 = 0) \end{array} \right\}
\end{array}
$$

[6] In section 5 we will address the fact that we need not lay out explicitly the sequence in which quantities are to be computed.

Now,

$$\begin{aligned} Q_4 &= \text{wp}(\text{"}p_2 := exp_4\text{"}, P_{\text{after}}) \\ &= \{P_{\text{before}} \quad \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (P_0 u_{02} + \upsilon_{12} p_1 + exp_4 = r_2) \\ &\qquad \wedge(R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0) \wedge (P_0^t A exp_4 = 0) \wedge (p_1^t A exp_4 = 0)\} \end{aligned}$$

from which we deduce that $exp_4 = r_2 - P_0 u_{02} - \upsilon_{12} p_1$ and

$$\begin{aligned} Q_4 &= \text{wp}(\text{"}p_2 := r_2 - P_0 u_{02} - \upsilon_{12} p_1\text{"}, P_{\text{after}}) \\ &= \{P_{\text{before}} \quad \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge T \wedge (R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0)) \\ &\qquad \wedge(P_0^t A(r_2 - P_0 u_{02} - \upsilon_{12} p_1) = 0) \wedge (p_1^t A(r_2 - P_0 u_{02} - \upsilon_{12} p_1) = 0)\} \\ &= \{P_{\text{before}} \quad \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0)) \\ &\qquad \wedge(P_0^t A r_2 - P_0^t A P_0 u_{02} = 0) \wedge (p_1^t A r_2 - p_1^t A P_0 u_{02} - \upsilon_{12} p_1^t A p_1 = 0)\} \end{aligned}$$

Similarly, we can determine $\upsilon_{12} := \exp_3 = (p_1^t A r_2 - p_1^t A P_0 u_{02}) / p_1^t A p_1$ and

$$\begin{aligned} Q_3 &= \text{wp}(\text{"}\upsilon_{12} := (p_1^t A r_2 - p_1^t A P_0 u_{02}) / p_1^t A p_1\text{"}, Q_4) \\ &= \{P_{\text{before}} \quad \wedge(\delta_1 A p_1 = r_1 - r_2) \wedge (R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0)) \wedge (P_0^t A r_2 - P_0^t A P_0 u_{02} = 0) \wedge T\} \end{aligned}$$

Next we can determine $u_{02} := \exp_2 = (P_0^t A P_0)^{-1} P_0^t A r_2$ and

$$\begin{aligned} Q_2 &= \text{wp}(\text{"}u_{02} := (P_0^t A P_0)^{-1} P_0^t A r_2\text{"}, Q_2) \\ &= \{P_{\text{before}} \wedge (\delta_1 A p_1 = r_1 - r_2) \wedge (R_0^t r_2 = 0) \wedge (r_1^t r_2 = 0)) \wedge T\} \end{aligned}$$

followed by $r_2 := \exp_1 = r_1 - \delta_1 A p_1$ and

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

(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

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

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

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

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 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 $R^t R = \Omega$ (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 (3), 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 algorithms that combine inner products [4, 5].

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

- **Choices for updates of individual variables.** There are often different choices for computing variables all of which lead to correct algorithms, although possibly with different computational or numerical properties. For example, manipulation of the governing equation in our example show that $r_i^t p_i = r_i^t r_i$, which leads to a different algorithmic version for the same loop-invariant.

  A related source of variant algorithms derives from the observation that in Section 4 we started by fixing the order in which the variables were to be computed. In practice, there may be multiple orders that lead to further versions for a given loop-invariant. One possibility is to examine the 'before' and 'after' predicates for inherent dependencies that determine the order. Another would be to try all possible orderings.

- **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).

REFERENCES

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, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. Representing linear algebra algorithms in code: The FLAME application programming interfaces. *ACM Transactions on Mathematical Software*, 31(1):27–59, March 2005.
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. 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.
11. Robert A. van de Geijn and Enrique S. Quintana-Ortí. *The Science of Programming Matrix Computations*. `www.lulu.com`, 2008.