TACC Technical Report TR-07-02

Applying Formal Derivation Techiques to Krylov Subspace Methods

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

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

Thus, our application domain of iterative methods is a proof-of-concept of the formalization of algorithm design and library generation.

1 Introduction

We present the Conjugate Gradient (CG) algorithm [8] in the FLAME framework [15]. With this, we have two goals in mind. First, we show how FLAME can be used to simplify the process of deriving iterative methods.

Traditional expositions of this method, and ones related to it, posit the basic form of relations between matrices and vectors, and computing the scalar coefficients in them by 'lengthy induction arguments' [11]. Our presentation is very much in the spirit of Householder's derivation [9], where both vector and scalar sequences are summarized as matrices. The big advantage here is that we can dispense with quantified statements over sequences, and instead consider simple predicates over simple, unindexed, objects.

Demonstrating the power of this approach, we derive a CG method for nonsymmetric systems in about half a page; a feat that warranted a whole research paper in the classical approach to polynomial iterative methods [16, 10].

Beyond simply presenting an alternative derivation of these methods, we argue that the essential calculations in a FLAME worksheet, contained in the update step, can be derived mechanically from the loop invariant of the algorithm. Coupling this to ongoing projects for automatic code generation from FLAME worksheetS, this 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.

2 Theory

In this paper we will derive 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.

This first section serves to familiarize the reader with the block formalism, and to establish the basic equations, as well 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, and let x_0 be any vector. Define $r_0 = Ax_0 - b$, then $r_0 = Ax_0 - b$ implies that $x = A^{-1}b = x_0 - A^{-1}r_0$.

Now, the Cayley-Hamilton theorem tells us 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 $x = x_0 + \pi(A)r_0$. Now, *if* we know the coefficients of $\pi(x)$, we would be done in theory. 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 $\{\tilde{\pi}^{(i)}\}_i$ and a sequence of approximations[1]

$$x_i = x_0 + \tilde{\pi}^{(i)}(A)r_0 \tag{1}$$

that, we hope, converge to the solution.

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 jth column of $K\langle A,k_0\rangle$, k_j, is defined by

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

With this, we rewrite Equation (2) as

$$x_{i+1} = x_0 + K\langle A,r_0\rangle \begin{pmatrix} \tilde{\pi}_{0i} \\ \vdots \\ \tilde{\pi}_{ii} \\ 0 \\ \vdots \end{pmatrix} \tag{2}$$

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

$$J = \begin{pmatrix} 0 & 0 & \cdots \\ 1 & 0 & \cdots \\ 0 & \ddots & \ddots \end{pmatrix} \quad \text{and} \quad E = \begin{pmatrix} 1 & 1 & \cdots \\ 0 & 0 & \cdots \\ 0 & \ddots & \ddots \end{pmatrix} \quad \text{so that} \quad J-E = \begin{pmatrix} -1 & -1 & \cdots \\ 1 & 0 & \cdots \\ 0 & \ddots & \ddots \end{pmatrix} \tag{3}$$

we can write Equation (2) in matrix form as

$$X(J-E) = K\tilde{U}, \tag{4}$$

where from now on we will write 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. (Note that we regularly abbreviate vector sequences in block notation: $K = (k_0,k_2,\ldots)$ leaving unspecified whether this is an infinite sequence or a finite part of it.)

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

$$X(J-I) = KU \tag{5}$$

1. In the whole of this document we will use zero-based indexing, including for indexing matrix elements.

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 ith column of upper triangular 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: $X = P\langle\{\pi_i\}_{i\geq 0}, A, x_0, f\rangle$.

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

Definition 2 *We call R a 'residual sequence' if it is the sequence of residuals of a polynomial iterative method X wrt A and f:*

$$R = AX - fe^t \text{ or } r_i = Ax_i - f.$$

Notation: $R = R\langle A, X, f\rangle$.

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

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

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 $P\langle\{\pi_i\}_{i\geq 0}, A, x_0, f\rangle$. Multiplying the equation

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

by A and subtracting f 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 $R = KU$ 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 now immediately see that X can also be defined as

$$X(J - I) = RU \tag{6}$$

for some upper triangular matrix U. We can now prove:

Lemma 2 *Residual sequences satisfy $AR = RH$ with H an upper Hessenberg matrix with zero column sums.*

Proof: Taking equation (6), multiplying by A, and adding $-f+f$ to the lhs, gives

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

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

Now we consider factoring the H matrix. If we leave the length of the R sequence indeterminate, the factors of H will be semi-infinite too; if we take a finite part R_n, then the following statement will hold by letting H be of size $(n+1)\times n$.

Lemma 3 *A hessenberg matrix 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:

$$\begin{cases} AR = RH \\ H \text{ has zero column sums} \end{cases} \Leftrightarrow AR = R(I-J)D^{-1}(I-U) \Leftrightarrow \begin{cases} APD = R(I-J) \\ P(I-U) = R \end{cases} \tag{7}$$

In this, we recognize the traditional formulation

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

Note that this form holds for any polynomial iterative method; various iterative methods (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 Conjugate Gradients method corresponds to U being single upper diagonal, with values deriving from the orthogonality of R.

In the remainder of this paper, we will take the block form (7) for given, and show how FLAME can be used to derive the coefficients in D and U.

3 (ultra) Brief introduction to FLAME

Here we give a very simple (in fact, *simplified* by leaving out many details) example to convey the mode of reasoning in FLAME. Let J be the matrix

$$J = \begin{pmatrix} 0 & & \emptyset \\ 1 & \ddots & \\ & \ddots & \ddots \end{pmatrix}$$

and let us consider the equation $AK = KJ$ with the first column of K given, where all matrices are of size $n \times n$, where $n > 1$. The reader of course recognizes this as a block formulation[2] of the Krylov sequence with coefficient matrix A and starting vector k_0.

We will now derive an iterative algorithm for constructing a matrix K that satisfies this equation. It is enough if we can show that a single iteration leaves a loop invariant predicate satisfied[3] In the FLAME methodology, this start by partitioning the matrices involved[4]:

$$K = \left(\begin{array}{c|c|c} K_L & k_M & K_R \end{array} \right), \quad J = \begin{pmatrix} J_{TL} & 0 & 0 \\ j_{ML}^t & 0 & 0 \\ 0 & j_{MR} & J_{BR} \end{pmatrix}, \quad \begin{cases} j_{ML}^t = (0,\ldots,0,1) \\ j_{MR} = (1,0,\ldots,0)^t \end{cases} \tag{8}$$

This means that, for the equation $AK = KJ$ to hold, the following system needs to be satisfied:

$$\begin{cases} AK_L = K_L J_{TL} + k_M j_{LM}^t \\ Ak_M = K_R j_{MR} \\ AK_R = K_R J_{Br} \end{cases}$$

The crucial step in the derivation is the choice of the invariant; in this case we choose the first equation as the invariant that holds at the start of an iteration.

Now we derive the steps that need to be taken to let the invariant hold at the end of the iteration. We split one column off the K_R block (the thick line indicates the location of the block from which the column is split):

$$\left(\begin{array}{c|c|c} K_L & k_M & K_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c|c} K_0 & k_1 & k_2 & K_3 \end{array} \right)$$

At the end of the iteration, the predicate has to hold for one more column, so the new partition will be

$$\left(\begin{array}{c|c|c|c} K_0 & k_1 & k_2 & K_3 \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} K_L & k_M & K_R \end{array} \right)$$

In other words, in the partitioned equation

$$A \left(\begin{array}{c|c|c|c} K_0 & k_1 & k_2 & K_3 \end{array} \right) = \left(\begin{array}{c|c|c|c} K_0 & k_1 & k_2 & K_3 \end{array} \right) \left(\begin{array}{c|c|c|c} J_{00} & & & \emptyset \\ \hline 1 & 0 & & \\ \hline & 1 & 0 & \\ \hline \emptyset & & j_{23} & J_{33} \end{array} \right) \tag{9}$$

we assume that at the start of an iteration $AK_0 = K_0 J_{00} + k_1$ is satisfied, and at the end of an iteration the additional 'after' equation of $Ak_1 = k_2$ has to be satisfied.

In conclusion, computing $k_2 \leftarrow Ak_1$, where k_1, k_2 are columns in the iteration-dependent partitioning of K, will make the equation $AK = KJ$ inductively be satisfied for the $n \times n$ matrix K.

---

2. In the sense of summarizing a complete vector sequence as a block, not in the sense of iterating on blocks.
3. One of our simplifications is the ignoring of initial and final conditions on the process.
4. In the traditional FLAME approach [15], a 2×2 partition is used. We will use a 3×3 partition instead.

This example, while admittedly of a rather trivial algorithm, illustrates the principle how FLAME, starting from a non-algorithmic description, derives the steps that will keep a loop-invariant predicate inductively satisfied, thus yielding an algorithm implementation that is proved correct.

Step	Annotated Algorithm: Compute K of size $m \times n$ so that $AK_{*,0:n-1} = KJ$ and $K_{*,0} = b$
1a	$\{\ K_{*,0} = b\}$
3	**Partition** $K \rightarrow \left(\begin{array}{c\|c\|c} K_L & k_M & K_R \end{array}\right)$, $J = \begin{pmatrix} J_{TL} & 0 & 0 \\ j_{ML}^t & 0 & 0 \\ 0 & j_{MR} & J_{BR} \end{pmatrix}$, $\begin{cases} j_{ML}^t = (0,\ldots,0,1) \\ j_{MR} = (1,0,\ldots,0)^t \end{cases}$ **where** $n(k_2) = 1$
2	$\{\ AK_L = K_L J_{TL} + k_M j_{ML}^t\}$
4	**While** $n(K_R) > 0$ **do**
2,4	$\{(\ AK_L = K_L J_{TL} + k_M j_{ML}^t) \wedge (\ n(K_R) > 0\ )\}$
5a	**Repartition** $\left(\begin{array}{c\|c\|c} K_L & k_M & K_R \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c\|c} K_0 & k_1 & k_2 & K_3 \end{array}\right)$, $\left(\begin{array}{c\|c\|c} J_{TL} & 0 & 0 \\ \hline j_{ML}^t & 0 & 0 \\ \hline 0 & j_{MR} & J_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c\|c} J_{00} & 0 & 0 & 0 \\ \hline j_{10}^t & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & j_{32} & J_{33} \end{array}\right)$ **where**
6	$\{\ AK_0 = K_0 J_{00} + k_1\ \}$
8	$\{\ k_2 = Ak_1\ \}$
7	$\left\{\ A\left(\begin{array}{cc} K_0 & k_1 \end{array}\right) = \left(\begin{array}{c\|c\|c} K_0 & k_1 & k_2 \end{array}\right) \begin{pmatrix} J_{00} & 0 \\ j_{10}^t & 0 \\ 0 & 1 \end{pmatrix}\ \right\}$
5b	**Continue with** $\left(\begin{array}{c\|c\|c\|c} K_0 & k_1 & k_2 & K_3 \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} K_L & k_M & K_R \end{array}\right)$, likewise for J
2	$\{\ AK_L = K_L J_{TL} + k_M j_{ML}^t\}$
	endwhile
2,4	$\{(\ AK_L = K_L J_{TL} + k_M j_{ML}^t) \wedge \neg(\ n(K_R) > 0\ )\}$
1b	$\{\ AK_{*,0:n-1} = KJ,\ K_{*,0} = b\}$

Figure 1: Worksheet for the Krylov sequence

4 Hestenes-Stiefel CG

We will now show that this notion, can be used to derive iterative methods such as the CG algorithm.

Using a 3×3 partition as in equation (8), the equations for residuals and search directions, under orthogonality of the residuals,

$$APD = R(I-J), \quad P(I-U) = R, \quad R^tR = \Omega \text{ diagonal}$$

turn into the following set of equations:

$$\begin{cases} \left(\begin{array}{c|c|c} AP_LD_L & Ap_Md_M & AP_RD_R \end{array} \right) = \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array} \right) \begin{pmatrix} J_{TL} & 0 & 0 \\ j_{ML}^t & 1 & 0 \\ 0 & j_{MR} & J_{BR} \end{pmatrix} \\ \left(\begin{array}{c|c|c} P_L & p_M & P_R \end{array} \right) \begin{pmatrix} U_{TL} & u_{TM} & u_{TR} \\ 0 & 1 & u_{MR} \\ 0 & 0 & U_{BR} \end{pmatrix} = \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array} \right) \\ \begin{pmatrix} R_L \\ r_M \\ R_R \end{pmatrix} \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array} \right) = \begin{pmatrix} \Omega_{TL} & 0 & 0 \\ 0 & \omega_{MM} & 0 \\ 0 & 0 & \Omega_{BR} \end{pmatrix} \end{cases} \tag{10}$$

The FLAME approach now takes this partition, and lets the middle column, or row and column, traverse the partitioned matrix. That is, the basic iteration:

- Assumes that part of the partitioned matrix, for instance the P_L and p_M blocks, have already been computed correctly;
- Splits off one column, or row and column, from the uncomputed part:

  $$\left(\begin{array}{c|c|c} P_L & p_M & P_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c|c} P_0 & p_1 & p_2 & P_3 \end{array} \right)$$

  This will yield a set of 'before' equations that are presumed to be satisfied in the current iteration. Under the above assumption that P_L and p_M are correctly computed, this will be a set of equations for P_0 and p_1.
- Then considers the implications of advancing the iteration by repartitioning the matrix:

  $$\left(\begin{array}{c|c|c} P_L & p_M & P_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c|c} P_0 & p_1 & p_2 & P_3 \end{array} \right)$$

  This gives rise to a larger set of equations, the 'after' equations. Under the assumption that P_L and p_M are correctly computed, this will be a set of equations for P_0, p_1, and p_2. Some of these may already be satisfied as part of the 'before' equations; the remaining ones define the operations that preserve the loop invariant when the iteration advances.

We now consider the Partitioned Matrix Expression (PME) (10), and for the invariant we let the equation $APD = R(I-J)$ hold in the L column, and $P(I-U) = R$ in both the L and M columns. (This is a conscious

choice by no means the only possible one. We will explore a different choice, which has different computational ramifications, in section 4.3.)

$$\begin{cases} \begin{pmatrix} AP_LD_L \end{pmatrix} = \begin{pmatrix} R_L & r_M \end{pmatrix} \begin{pmatrix} J_{TL} \\ j^t_{ML} \end{pmatrix} \\ \begin{pmatrix} P_L & p_M \end{pmatrix} \begin{pmatrix} U_{TL} & u_{TM} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_L & r_M \end{pmatrix} \\ \begin{pmatrix} R_T \\ r_M \end{pmatrix} \begin{pmatrix} R_L & r_M \end{pmatrix} = \begin{pmatrix} \Omega_{TL} & 0 \\ 0 & \omega_{MM} \end{pmatrix} \end{cases} \tag{11}$$

4.1 Partitions and before/after equations

The partition before the update is

$$\begin{array}{c} \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array}\right) \\ \downarrow \\ \left(\begin{array}{c|c|c|c} R_0 & r_1 & r_2 & R_3 \end{array}\right) \end{array}, \quad \left(\begin{array}{c|c|c} J_{TL} & 0 & 0 \\ \hline j^t_{ML} & 1 & 0 \\ \hline 0 & j_{MR} & J_{BR} \end{array}\right) \\ \downarrow \\ \left(\begin{array}{c|c|c|c} J_{00} & 0 & 0 & 0 \\ \hline j^t_{10} & 1 & 0 & 0 \\ \hline 0 & -1 & 1 & j_{23} \\ \hline 0 & 0 & 0 & J_{33} \end{array}\right), \quad \begin{array}{c} \left(\begin{array}{c|c|c} P_L & p_M & P_R \end{array}\right) \\ \downarrow \\ \left(\begin{array}{c|c|c|c} P_0 & p_1 & p_2 & P_3 \end{array}\right) \end{array}, \quad \left(\begin{array}{c|c|c} U_{TL} & u_{TM} & u_{TR} \\ \hline 0 & 1 & u_{MR} \\ \hline 0 & 0 & U_{BR} \end{array}\right) \\ \downarrow \\ \left(\begin{array}{c|c|c|c} U_{00} & u_{01} & u_{02} & u_{03} \\ \hline 0 & 1 & u_{12} & u_{13} \\ \hline 0 & 0 & 1 & u_{23} \\ \hline 0 & 0 & 0 & U_{33} \end{array}\right)$$

After performing the update, repartition:

$$\begin{array}{c} \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array}\right) \\ \uparrow \\ \left(\begin{array}{c|c|c|c} R_0 & r_1 & r_2 & R_3 \end{array}\right) \end{array}, \quad \left(\begin{array}{c|c|c} J_{TL} & 0 & 0 \\ \hline j^t_{ML} & 1 & 0 \\ \hline 0 & j_{BM} & J_{BR} \end{array}\right) \\ \uparrow \\ \left(\begin{array}{c|c|c|c} J_{00} & 0 & 0 & 0 \\ \hline j^t_{10} & 1 & 0 & 0 \\ \hline 0 & -1 & 1 & 0 \\ \hline 0 & 0 & j_{32} & J_{33} \end{array}\right), \quad \begin{array}{c} \left(\begin{array}{c|c|c} P_L & p_M & P_R \end{array}\right) \\ \uparrow \\ \left(\begin{array}{c|c|c|c} P_0 & p_1 & p_2 & P_3 \end{array}\right) \end{array}, \quad \left(\begin{array}{c|c|c} U_{TL} & u_{TM} & u_{TR} \\ \hline 0 & 1 & u_{MR} \\ \hline 0 & 0 & U_{BR} \end{array}\right) \\ \uparrow \\ \left(\begin{array}{c|c|c|c} U_{00} & u_{01} & u_{02} & u_{03} \\ \hline 0 & 1 & u_{12} & u_{13} \\ \hline 0 & 0 & 1 & u_{23} \\ \hline 0 & 0 & 0 & U_{33} \end{array}\right)$$

Before the update, the invariant reads:

$$\begin{cases} AP_0D_d = R_0J_{00} + r_1 j_{10}^t, \quad \begin{pmatrix} P_0 & p_1 \end{pmatrix} \begin{pmatrix} U_{00} & u_{01} \\ & 1 \end{pmatrix} = \begin{pmatrix} R_0 & r_1 \end{pmatrix}, \\ \begin{pmatrix} R_0^t \\ r_1 \end{pmatrix} \begin{pmatrix} R_0 & r_1 \end{pmatrix} = \begin{pmatrix} \Omega_0 & 0 \\ 0 & \omega_1 \end{pmatrix}, \quad \begin{pmatrix} P_0^t \\ p_1^t \end{pmatrix} A \begin{pmatrix} P_0 & p_1 \end{pmatrix} = \begin{cases} \begin{pmatrix} P_0^tAP_0 & 0 \\ p_1^tAP_0 & p_1^tAp_1 \end{pmatrix} & \text{general case} \\ \begin{pmatrix} P_0^tAP_0 & 0 \\ 0 & p_1^tAp_1 \end{pmatrix} & \text{symmetric case.} \end{cases} \end{cases} \tag{12}$$

After the update, the first line of the invariant reads:

$$A \begin{pmatrix} P_0 & p_1 \end{pmatrix} \begin{pmatrix} D_0 & \\ & d_1 \end{pmatrix} = \begin{pmatrix} R_0 & r_1 \end{pmatrix} \begin{pmatrix} J_{00} & \\ j_{10}^t & 1 \end{pmatrix} + r_2 \begin{pmatrix} \bar{0} & -1 \end{pmatrix}$$

$$\left(\begin{array}{c|c|c} P_0 & p_1 & p_2 \end{array} \right) \begin{pmatrix} U_{00} & u_{01} & u_{02} \\ 0 & 1_{11} & u_{12} \\ 0 & 0 & 1_{22} \end{pmatrix} = \left(\begin{array}{c|c|c} R_0 & r_1 & r_2 \end{array} \right) \tag{13}$$

The extra equations that need to be satisfied to have the invariant satisfied after the update are:

$$Ap_1d_1 = r_1 - r_2, \qquad P_0u_{02} + p_1u_{12} + p_2 = r_2.$$

These equations describe the updates for P and R; they involve coefficients that are still to be determined. For the computation of these, we have the diagonality of $r_i^tAr_j$ and the lower triangularity (general case; diagonality for the symmetric case) of $p_i^tAp_j$.

4.2 Computation of coefficients

Computing the D coefficients In the derivation of the coefficients we need some auxiliary facts that follow from the invariant equations. We start by multiplying equation (13):

$$\left(\begin{array}{c|c|c} R_0 & r_1 & r_2 \end{array} \right) \times \left[\left(\begin{array}{c|c|c} P_0 & p_1 & p_2 \end{array} \right) \begin{pmatrix} U_{00} & u_{01} & u_{02} \\ 0 & 1_{11} & u_{12} \\ 0 & 0 & 1_{22} \end{pmatrix} = \left(\begin{array}{c|c|c} R_0 & r_1 & r_2 \end{array} \right) \right]$$

$$\Rightarrow \begin{pmatrix} R_0^tP_0 & R_0^tp_1 & R_0^tp_2 \\ r_1^tP_0 & r_1^tp_1 & r_1^tp_2 \\ r_2^tP_0 & r_2^tp_1 & r_2^tp_2 \end{pmatrix} \begin{pmatrix} U_{00} & u_{01} & u_{02} \\ 0 & 1_{11} & u_{12} \\ 0 & 0 & 1_{22} \end{pmatrix} = \begin{pmatrix} \Omega_0 & 0 & 0 \\ 0 & \omega_1 & 0 \\ 0 & 0 & \omega_2 \end{pmatrix} \tag{14}$$

From equation (14) we derive necessary conditions such as $r_2^t p_1 = 0$. The sufficient conditions are

$$\begin{cases} R_0^t r_2 = 0 & \text{this is } R_0^t r_1 - R_0^t A p_1 d_1 = 0 - 0 \text{ from the before equations and the lower triangularity of } r_i^t A p_j \\ r_1^t r_2 = 0 & \text{this is } r_1^t r_1 - r_1^t A p_1 d_1 \text{ so this is satisfied if } d_1 = r_1^t r_1 / r_1^t A p_1 \\ r_2^t P_0 U_{00} = 0 & \text{satisfied since } r_2^t P_0 = r_1^t P_0 - d_1 p_1^t A^t P_0 \text{ and } p_i^t A p_j \text{ is lower triangular} \\ r_2^t p_1 = 0 & \text{note } r_2^t p_1 = r_1^t p_1 - d_1 p_1^t A^t p_1 \text{ so this reduces to the requirement that } d_1 = r_1^t p_1 / p_1^t A p_1. \end{cases}$$

Since $r_1^t r_1 = r_1^t p_1$ and $r_1^t A p_1 = p_1^t A p_1$, we find that

$$d_1 = \frac{r_1^t r_1}{p_1^t A p_1}. \tag{15}$$

is the sole necessary and sufficient condition.

Additionally, we now conclude from (14) that the matrix $R_i^t P_j$ is upper triangular.

Equations for $P_i^t A P_j$ Using the upper triangularity of $R_i^t P_j$, or equivalently the lower triangular of $P_i^t R_j$, we get

$$\begin{aligned} \begin{pmatrix} P_0^t R_0 & 0 & 0 \\ p_1^t R_0 & p_1^t r_1 & 0 \\ p_2^t R_0 & p_2^t r_1 & p_2^t r_2 \end{pmatrix} \begin{pmatrix} J_{00} & 0 & 0 \\ j_{10}^t & 1 & 0 \\ 0 & j_{12} & J_{22} \end{pmatrix} &= \begin{pmatrix} P_0^t A P_0 & P_0^t A p_1 & P_0^t A p_2 \\ p_1^t A P_0 & p_1^t A p_1 & p_1^t A p_2 \\ p_2^t A P_0 & p_2^t A p_1 & p_2^t A p_2 \end{pmatrix} \\ &= \begin{pmatrix} P_0^t A P_0 & 0 & 0 \\ p_1^t A P_0 & p_1^t A p_1 & 0 \\ p_2^t A P_0 & p_2^t A p_1 & p_2^t A p_2 \end{pmatrix} \end{aligned} \tag{16}$$

where we conclude the zeros in the rhs from the form of the lhs. In the case of a symmetric coefficient matrix A, the rhs is both lower triangular and symmetric, hence diagonal. This gives the general statement that $p_i^t A p_j = 0$ for $i \neq j$.

Computating the U coefficients Next we consider the computation of the U coefficients in the $P(I-U) = R$ equation. Since the first two columns are satisfied as part of the before equations, it is necessary and sufficient to satisfy the equations in the third column. We multiply them:

$$\begin{pmatrix} R_0^t \\ r_1 \\ r_2 \end{pmatrix} \times \left[\begin{pmatrix} P_0 & p_1 & p_2 \end{pmatrix} \begin{pmatrix} -u_{02} \\ -u_{12} \\ 1 \end{pmatrix} = r_2 \right]$$

giving (where we use parts of equation (14) to zero some coefficients)

$$\begin{pmatrix} R_0^t P_0 & R_0^t p_1 & R_0^t p_2 \\ 0 & r_1^t p_1 & r_1^t p_2 \\ 0 & 0 & r_2^t p_2 \end{pmatrix} \begin{pmatrix} -u_{02} \\ -u_{12} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ r_2^t r_2 \end{pmatrix} \tag{17}$$

The third row contains the known fact that $r_2^t p_2 = r_2^t r_2$, but the second row reads

$$-r_1^t p_1 u_{12} + r_2^t p_2 = 0 \Rightarrow u_{12} = \frac{r_1^t p_2}{r_1^t r_1}$$

In the symmetric case we observe that $p_2^t r_2 = p_2^t r_1 + p_2^t A p_1 d_1 = p_2^t r_1$ (using the symmetry of $P_i^t A P_j$, observed above), so, combined with $p_2^t r_2 = r_2^t r_2$, we find

$$u_{12} = \frac{r_2^t r_2}{r_1^t r_1}. \tag{18}$$

Next we show that $u_{02} = 0$ in the symmetric case. Taking the first column of the $R(J-I) = APD$ equation and left multiplying it by p_2 we get

$$\begin{pmatrix} P_0^t \\ p_1^t \\ p_2^t \end{pmatrix} \left[AP_0D_0 = \begin{pmatrix} R_0 & r_1 \end{pmatrix} \begin{pmatrix} J_{00} \\ j_{10}^t \end{pmatrix} \right]$$

$$\begin{pmatrix} P_0^t AP_0 \\ p_1^t AP_0 \\ p_2^t AP_0 \end{pmatrix} D_0 = \begin{pmatrix} P_0^t AP_0 \\ 0 \\ 0 \end{pmatrix} D_0 = \begin{pmatrix} P_0^t R_0 & 0 \\ p_1^t R_0 & p_1^t r_1 \\ p_2^t R_0 & p_2^t r_1 \end{pmatrix} \begin{pmatrix} J_{00} \\ j_{10}^t \end{pmatrix}$$

In the second row, we see

$$p_1^t r_1 J_{10} = p_1^t R_0 J_{00} + p_1^t AP_0 D_0 = p_1^t R_0 J_{00} \quad \text{so} \quad p_1^t R_0 = p_1^t r_1 J_{10} J_{00}^{-1} = r_1^t r_1 J_{10} J_{00}^{-1}. \tag{19}$$

Similarly, the third row, we have

$$p_2^t r_1 J_{10} = p_2^t R_0 J_{00} + p_2^t AP_0 D_0 = p_2^t R_0 J_{00} \quad \text{so} \quad p_2^t R_0 = p_2^t r_1 J_{10} J_{00}^{-1} = r_2^t r_2 J_{10} J_{00}^{-1}. \tag{20}$$

Together, this gives us for the first row of equation (17)

$$\begin{pmatrix} R_0^t R_0 & J_{00}^{-t} J_{10}^t r_1^t r_1 & J_{10}^{-t} r_2^t r_2 \end{pmatrix} \begin{pmatrix} u_{02} \\ -r_2^t r_2 / r_1^t r_1 \\ 1 \end{pmatrix} = 0$$

from which $u_{02} = 0$ readily follows. (In fact, we are free to ignore this analytical conclusion, and compute u_{02} from this equation, for added numerical stability.)

Computing the u coefficients in the nonsymmetric case .

The above derivation of the coefficients u_{12} (and the derivation that u_{02} was zero) depended on the diagonality of P^tAP. In the nonsymmetric case this no longer holds: as observed before, P^tAP is lower triangular in general. To use equation (17) and the following reasoning (equations (20) and (19)) in the general case, we needs expressions for, among others, $p_2^tAP_0$. This is problematic, since p_2 is unknown.

However, we observe that p_2 is computed after r_2, and so we try to express $p_i^tAp_j$ coefficients in terms of $r_i^tAr_j$.

From the equation $R = P(I-U)$ we get

$$R^tAR = (I-U)^tP^tAP(I-U),$$

which specifically translates as

$$\begin{pmatrix} R_0^tAR_0 & R_0^tAr_1 & R_0^tAr_2 \\ r_1^tAR_0 & r_1^tAr_1 & r_1^tAr_2 \\ r_2^tAR_0 & r_2^tAr_1 & r_2^tAr_2 \end{pmatrix} = \begin{pmatrix} U_{00}^t & & \\ u_{01}^t & 1 & \\ u_{02}^t & u_{12}^t & 1 \end{pmatrix} \begin{pmatrix} P_0^tAP_0 & & \\ p_1^tAP_0 & p_1^tAp_1 & \\ p_2^tAP_0 & p_2^tAp_1 & p_2^tAp_2 \end{pmatrix} \begin{pmatrix} U_{00} & u_{01} & u_{02} \\ & 1 & u_{12} \\ & & 1 \end{pmatrix}$$
$$= \begin{pmatrix} U_{00}^tP_0^tAP_0 & & \\ u_{01}^tP_0^tAP_0 + p_1^tAP_0 & p_1^tAP1 & \\ * & * & * \end{pmatrix} \begin{pmatrix} U_{00} & u_{01} & u_{02} \\ & 1 & u_{12} \\ & & 1 \end{pmatrix}$$

which is an equation of the form $(r_i^tAr_j) = \tilde{L}U$. From this we get, by examining the $(0,2)$ position

$$R_0^tAr_2 = \tilde{L}_{00}u_{02} \text{ where } \tilde{L}_{00} = U_{00}^tP_0^tAP_0$$

which gives us u_{02}, and subsequently the $(1,2)$ position gives

$$r_1^tAr_2 = \tilde{L}_{10}u_{02} + \tilde{L}_{11}u_{12} \text{ where } \tilde{L}_{10} = u_{01}^tP_0^tAP_0 + p_1^tAP_0 \text{ and } \tilde{L}_{11} = p_1^tAp_1$$

from which u_{12} easily follows.

> Note that here we have derived in less than a page an algorithm for the nonsymmetric CG method, a feat that used to merit a whole research paper.

4.3 Invariant #2

Derivations made with FLAME are by no means unique. Different choices of PMEs can lead to algorithms that differ, for instance, by a loop interchange, or their choice of kernel operations. We will illustrate this for the CG algorithm. The variant we derive is correct, but can not express the work savings in the symmetric case. Hence, it is identical to the previous algorithm in the nonsymmetric case, but wasteful in the symmetric case.

We base the algorithm on just taking the L column of both equations:

$$\left(\begin{array}{c|c|c} AP_LD_L & Ap_Md_M & AP_RD_R \end{array}\right) = \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array}\right) \left(\begin{array}{c|c|c} J_{TL} & 0 & 0 \\ \hline j_{ML}^t & 1 & 0 \\ \hline 0 & j_{MR} & J_{BR} \end{array}\right)$$

and

$$\left(\begin{array}{c|c|c} P_L & p_M & P_R \end{array}\right) \left(\begin{array}{c|c|c} U_{TL} & u_{TM} & u_{TR} \\ \hline 0 & 1 & u_{MR} \\ \hline 0 & 0 & U_{BR} \end{array}\right) = \left(\begin{array}{c|c|c} R_L & r_M & R_R \end{array}\right)$$

After the update, the invariant reads:

$$A \begin{pmatrix} P_0 & p_1 \end{pmatrix} \begin{pmatrix} D_0 & \\ & d_1 \end{pmatrix} = \begin{pmatrix} R_0 & r_1 \end{pmatrix} \begin{pmatrix} J_{00} & \\ j_{10}^t & 1 \end{pmatrix} + r_2 \begin{pmatrix} \bar{0} & -1 \end{pmatrix}$$

$$\begin{pmatrix} P_0 & p_1 \end{pmatrix} \begin{pmatrix} U_{00} & u_{01} \\ & 1 \end{pmatrix} = \begin{pmatrix} R_0 & r_1 \end{pmatrix}$$

giving update equations

$$Ap_1d_1 = r_1 - r_2,$$

We now have the problem of computing the U_{01} coefficient. The solution is much like before.

We multiply the P update equation:

$$\begin{pmatrix} R_0^t \\ r_1^t \end{pmatrix} \times \left[\begin{pmatrix} P_0 & p_1 \end{pmatrix} \begin{pmatrix} -u_{01} \\ 1 \end{pmatrix} = r_1 \right]$$

Using the fact that $r_1^tP_0 = 0$, we get

$$\begin{pmatrix} R_0^tP_0 & R_0^tp_1 \\ 0 & r_1^tp_1 \end{pmatrix} \begin{pmatrix} -u_{01} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ r_1^tr_1 \end{pmatrix}$$

which gives $r_1^tp_1 = r_1^tr_1$ and

$$u_{01} = \frac{R_0^tp_1}{R_0^tP_0} = \frac{r_1^tr_1}{R_0^tP_0}$$

where for the numerator we used

$$p_1^t \times [r_1 = R_0 + AP_0] \Rightarrow p_1^tr_1 = p_1^tR_0 + p_1^tAP_0 = p_1^tR_0$$

5 On formal derivation

Our problem of constructing P and R can be formulated as a general problem

$$\Theta(A,x_0,b,R,P,U,D)=0 \qquad \text{where} \qquad \Theta(A,x_0,b,R,P,U,D)=\begin{cases} R_{*0}-(Ax_0-b) \\ APD-R(I-J) \\ P(I-U)-R \\ R^tR-\mathrm{diag}(R^tR) \end{cases}$$

The problem is in realizing the transition from this prescriptive statement to an algorithmic formulation. Partly, this requires structural reasoning. For instance, the equation $APD = R(I-J)$ translates to $Ap_id_i = r_i - r_{i+1}$, and since we know r_0, the r_i vectors can now be derived in sequence.

Other facts are harder to determine. The value of d_i (and u_{ij}) only followed from a complicated reasoning where equations were multiplied by quantities, and known facts substituted in the resulting equation. Ultimately, an equation with a single unknown was produced.

Formally, this is a breadth-first search process, where new levels are derived by multiplying results in previous levels together. In fact, continuing this process can lead to discovery of new algorithms. We will illustrate this by deriving a CG method that was motivated by a reorganization of the inner products [3, 12, 14, 6]. Such reorganizations are mostly motivated from a point of parallel computing, where each inner product is a synchronization point.

5.1 A CG variant

The CG algorithm, as described above, contains two inner products that are interdependent, thus introducing two global synchronization points per iteration in a parallel computing context. For this reason, a number of variants have been derived where the inner products can be computed simultaneously. Here we give the derivation of one possible variant [3, 5]; several others exist [12, 4, 7].

We start from the basic iterations

$$APD = R(I-J), \qquad P(I-U) = R$$

and introduce some auxiliary quantities and relations

$$QD = R(I-J), \qquad Q = AP, \qquad P(I-U) = R, \qquad Q(I-U) = S, \qquad S = AR.$$

The crucial step is the replacement of the P^tAP inner product by R^tAR, which follows from the relation

$$(I-U)^tP^tAP(I-U) = R^tAR.$$

Writing this out in block form gives

$$\begin{pmatrix} U_{00}^t & 0 & 0 \\ u_{01}^t & 1 & 0 \\ u_{02}^t & u_{12}^t & U_{22}^t \end{pmatrix} \begin{pmatrix} P_0^t A P_0 & 0 & 0 \\ 0 & p_1^t A p_1 & 0 \\ 0 & 0 & p_2^t A p_2 \end{pmatrix} \begin{pmatrix} U_{00} & u_{01} & u_{02} \\ 0 & 1 & u_{12} \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R_0^t A R_0 & R_0^t A r_1 & R_0^t A r_2 \\ r_1^t A R_0 & r_1^t A r_1 & r_1^t A r_2 \\ r_2^t A R_0 & r_2^t A r_1 & r_2^t A r_2 \end{pmatrix}$$

which contains the following equations:

$$\begin{cases} U_{00}^t P_0^t A P_0 U_{00} = R_0^t A R_0 \\ u_{01}^t P_0^t A P_0 u_{01} + p_1^t A p_1 = r_1^t A r_1 \\ u_{02}^t P_0^t A P_0 u_{02} + u_{12}^t p_1^t A p_1 u_{12} + p_2^t A p_2 = r_2^t A r_2 \end{cases}$$

These can be interpreted as recurrence relations for the $p_i^t A p_i$ quantities, for instance

$$p_1^t A p_1 = r_1^t A r_1 - u_{01}^t P_0^t A P_0 u_{01}.$$

We note that this gives us indeed the $p_1^t A p_1$ quantity that is needed for the computation of d_1; equation (15). However, since u_{01} is a dense vector, this requires the dense block of inner products $P_0^t A P_0$, of which we only know the diagonal. A better solution is to take the third relation, which in the symmetric case reduces to the simple scalar equation

$$p_2^t A p_2 = r_2^t A r_2 - u_{12}^t p_1^t A p_1 u_{12}.$$

Now we have the problem that we are computing $p_2^t A p_2$ instead of $p_1^t A p_1$. In effect, we are computing a quantity one iteration to early. This problem can be solved in the following ways.

- We apply a compiler transformation that will carry this quantity to the next iteration. This involves the introduction of a new technology into the FLAME framework, which we would like to avoid.
- We can use a $4 \times 4 \rightarrow 5 \times 5 \rightarrow 4 \times 4$ scheme. This option lets us stay firmly within the FLAME framework without invoking new capabilities, however, the addition of yet another row and column to the partition feels forced and unnecessary.
- Rather than letting the various inner products be temporary quantities, to be computed on demand, we can incorporate them into the demands of the algorithm. Thus we can compute $p_1^t A p_1$ in one iteration, and use it in the next. While this is an elegant solution, it requires coming up with a non-trivial extension to the invariant.

The jury is still out on this.

6 Conclusion

In this paper we showed how reasoning about Krylov methods can be done systematically, even mechanically, in the FLAME framework. Basic update equations follow from the invariant, analogous to many earlier algorithms derived in FLAME.

A novel aspect is formed by scalars in the algorithm, which we derive from the constraints (such as orthogonality conditions) on the vectors computed. It is our hope that this reasoning can be implemented in a symbolic system, such as an automatic theorem prover. A prototype system exists [2, 13], though of more limited scope.

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 and Robert van de Geijn. Automation in dense linear algebra. Technical Report AICES-2008-2, Aachen Institute for Computational Engineering Science, RWTH Aachen, October 2008.

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

[4] E.F. D'Azevedo, V.L. Eijkhout, and C.H. Romine. Lapack working note 56: Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessor. Technical Report CS-93-185, Computer Science Department, University of Tennessee, Knoxville, 1993.

[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] E.F. D'Azevedo and C.H. Romine. Reducing communcation costs in the conjugate gradient algorithm on distributed memory multiprocessors. Technical Report ORNL/TM-12192, Oak Ridge National Lab, 1992.

[7] J. Demmel, M. Heath, and H. Van der Vorst. Parallel numerical linear algebra. In *Acta Numerica 1993*. Cambridge University Press, Cambridge, 1993.

[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] Gerard Meurant. Multitasking the conjugate gradient method on the CRAY X-MP/48. *Parallel Computing*, 5:267–280, 1987.

[13] Sergey Kolos Paolo Bientinesi and Robert van de Geijn. Automatic derivation of linear algebra algorithms with application to control theory. In *Proceedings of PARA'04 State-of-the-Art in Scientific Computing, June 20-23, 2004*.

[14] Yousef Saad. Practical use of some krylov subspace methods for solving indefinite and nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 5:203–228, 1984.

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

[16] 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*.