Accelerating eigenvector and pseudospectra computation using blocked multi-shift triangular solves

Tim Moon[1] and Jack Poulson[1,2]

[1]Institute for Computational and Mathematical Engineering,
Stanford University, CA, USA
[2]Department of Mathematics, Stanford University, CA, USA

July 21, 2016

Abstract

Multi-shift triangular solves are basic linear algebra calculations with applications in eigenvector and pseudospectra computation. We propose blocked algorithms that efficiently exploit Level 3 BLAS to perform multi-shift triangular solves and safe multi-shift triangular solves. Numerical experiments indicate that computing triangular eigenvectors with a safe multi-shift triangular solve achieves speedups by a factor of 60 relative to LAPACK. This algorithm accelerates the calculation of general eigenvectors threefold. When using multi-shift triangular solves to compute pseudospectra, we report ninefold speedups relative to EigTool.

1 Introduction

A common task in numerical linear algebra is the design of algorithms that can be efficiently implemented using the Basic Linear Algebra Subroutines (BLAS). BLAS is a specification of low-level Fortran routines for vector operations (Level 1) [14], matrix-vector operations (Level 2) [8, 9], and matrix-matrix operations (Level 3) [6, 7]. It has been in development since the late 1970s and there currently exist a multitude of high-quality (and often vendor-tuned) implementations such as OpenBLAS [20, 23], ATLAS [21], and Intel MKL. Leveraging BLAS, especially Level 3 BLAS, can achieve substantial improvements in performance since it uses highly optimized code that efficiently exploits a machine's memory architecture and processor capabilities.

Despite a large body of work devoted to utilizing Level 3 BLAS in high-quality linear algebra packages like LAPACK [1], there still exist important

routines that are limited to Level 2 BLAS. For instance, the calculation of triangular eigenvectors in the LAPACK routine `xTREVC` is dominated by safe triangular solves that employ Level 2 BLAS routines [11]. In addition, matrix pseudospectra are typically computed with embarrassingly parallel Level 2 BLAS triangular solves [18]. This paper proposes blocked implementations for multi-shift triangular solves that address both of these problems.

2 Multi-shift Triangular Solve

Given an $m \times m$ upper triangular matrix U and right-hand side vectors $b_1, \cdots, b_n$, the triangular solve problem seeks solution vectors $x_1, \cdots, x_n$ that solve n triangular systems of the form[1]

$$U x_j = b_j. \tag{1}$$

Consolidating the b_j's and x_j's into $m \times n$ matrices B and X, respectively, this is equivalent to solving the matrix problem $UX = B$. Assuming U is well-conditioned, the naive approach is to apply back substitution to each right-hand side:

Algorithm 1 Single triangular solve with back substitution
---
    procedure $\text{TRSV}(U, b)$ $\triangleright$ b is overwritten with x
        for $i = m : -1 : 1$ **do**
            $\mathcal{I}_0 := 1 : i - 1$
            $b(i) := b(i)/U(i, i)$ $\triangleright$ Diagonal step
            $b(\mathcal{I}_0) := b(\mathcal{I}_0) - b(i) * U(\mathcal{I}_0, i)$ $\triangleright$ Substitution step (`xAXPY`)

Applying this algorithm to one right-hand side takes m divisions in the diagonal step and $\sim m^2$ flops in the substitution step, so the computational work is dominated by the substitution step if the matrix dimension is sufficiently large. This routine is backward stable and implemented in the Level 2 BLAS routine `xTRSV`. However, given a block size n_b, we can apply a blocked algorithm [12]:

Algorithm 2 Triangular solve with blocked back substitution
---
    procedure $\text{TRSM}(U, B)$ $\triangleright$ B is overwritten with X
        for $i = m - n_b + 1 : -n_b : 1$ **do** $\triangleright$ Assume m is a multiple of n_b
            $\mathcal{I}_0 := 1 : i - 1$
            $\mathcal{I}_1 := i : i + n_b - 1$
            for $j = 1 : n$ **do**
                $\text{TRSV}(U(\mathcal{I}_1, \mathcal{I}_1), B(\mathcal{I}_1, j))$ $\triangleright$ Diagonal block step (`xTRSV`)
            $B(\mathcal{I}_0, :) := B(\mathcal{I}_0, :) - U(\mathcal{I}_0, \mathcal{I}_1) * B(\mathcal{I}_1, :)$ $\triangleright$ Substitution step (`xGEMM`)
---

[1]This paper will focus on the left upper triangular case. However, similar techniques can be applied for the lower triangular and right matrix cases. For instance, the left lower triangular case requires forward substitution instead of back substitution.

The blocked algorithm takes the same number of flops as the naive algorithm, but it uses the Level 3 BLAS routine `xGEMM` to perform matrix multiplication in the substitution step. The fraction of flops taking place in the substitution step (the "level-3 fraction") is approximately $1 - n_b/m$, so the calculation is efficient if $m \gg n_b$. This routine is implemented in the Level 3 BLAS routine `xTRSM`.

The multi-shift triangular solve problem is a variant of the standard triangular solve problem. Given scalar shifts $\lambda_1, \cdots, \lambda_n$, we seek to solve n triangular systems of the form

$$(U - \lambda_j I) x_j = b_j. \tag{2}$$

Each triangular system has a different matrix, so a naive approach is to apply unblocked back substitution on each system. However, the matrices only differ in the diagonal entries. This implies we can use blocked back substitution with a modification to the diagonal block step:

Algorithm 3 Multi-shift triangular solve with blocked back substitution

---
    procedure MULTISHIFTTRSM(U, λ, B)   ▷ B is overwritten with X
        for $i = m - n_b + 1 : -n_b : 1$ **do**   ▷ Assume m is a multiple of n_b
           $\mathcal{I}_0 := 1 : i - 1$
           $\mathcal{I}_1 := i : i + n_b - 1$
           for $j = 1 : n$ **do**
               TRSV($U(\mathcal{I}_1, \mathcal{I}_1) - \lambda(j) * I, B(\mathcal{I}_1, j)$) ▷ Diagonal block step (`xTRSV`)
           $B(\mathcal{I}_0, :) := B(\mathcal{I}_0, :) - U(\mathcal{I}_0, \mathcal{I}_1) * B(\mathcal{I}_1, :)$ ▷ Substitution step (`xGEMM`)

As before, the bulk of the computation is performed efficiently with Level 3 BLAS in the substitution step.

3   Safe Multi-Shift Triangular Solve

If the $m \times m$ upper triangular matrix U is ill-conditioned or singular, performing a triangular solve with back substitution may result in division by zero or floating point overflow. To avoid these pitfalls, we consider the (single) safe triangular solve problem. Given a nonzero right-hand side vector b, we seek a nonzero solution vector x and a scaling factor s in the unit interval $[0, 1]$ such that

$$Ux = sb. \tag{3}$$

The LAPACK routine `xLATRS` solves this problem with safeguarded back substitution [2]. This routine begins by estimating the growth of entries during standard back substitution. If we run j iterations of back substitution, $b(i)$ will be overwritten with $x(i)$, where $i = m - j + 1$. At this stage of the calculation

we define $M(i) = |b(i)|$ and $G(i) = \|b(1:i-1)\|_\infty$. We have the initial values $M(m+1) = 0$ and $G(m+1) = \|b\|_\infty$ and the bounds

$$
M(i) \leq \frac{G(i+1)}{|U(i,i)|}
$$

$$
\leq \max\left\{\frac{G(i+1)}{|U(i,i)|}, M(i+1)\right\} \tag{4}
$$

$$
G(i) \leq G(i+1) + M(i)\,\|U(1:i-1,i)\|_\infty
$$

$$
\leq G(i+1)\left(1 + \frac{\|U(1:i-1,i)\|_\infty}{|U(i,i)|}\right). \tag{5}
$$

These bounds can be computed recursively[2] and they increase monotonically with each iteration. Thus, the worst-case growth can be estimated by computing bounds for $M(1)$ and $G(1)$. If the growth is not too large, i.e. less than a machine-dependent overflow constant Ω, then we can confidently apply standard back substitution and set $s = 1$. Otherwise, we must check at each step of back substitution to avoid overflow and division by zero. In the jth iteration of back substitution, we compute the following quantity prior to the diagonal step,

$$
M(i) = \left|\frac{b(i)}{U(i,i)}\right|. \tag{6}
$$

In the case $U(i,i) = 0$, we approximate $U(i,i)$ with some nonzero $\delta = O(\epsilon\|U\|_\infty)$, where ϵ is machine epsilon. Note that this approximation does not disrupt the backward stability of back substitution. If $M(i) \geq \Omega$, then s is reduced until $M(i) < \Omega$ to protect against numerical issues in the diagonal step. After the diagonal step and before the substitution step, we compute the bound

$$
G(i) \leq G(i+1) + M(i)\,\|U(1:i-1,i)\|_\infty. \tag{7}
$$

If $G(i) \geq \Omega$, then s is reduced until $G(i) < \Omega$ to protect against numerical issues in the substitution step. The complete algorithm is outlined below:

---

[2]In practice, we compute lower bounds to the reciprocals $1/M(i)$ and $1/G(i)$ to avoid overflow.

Algorithm 4 Single safe triangular solve with safeguarded back substitution
---
procedure $\text{SAFETRSV}(U, b, s)$ $\qquad\qquad\qquad\triangleright b$ is overwritten with x
$\quad s := 1$
$\quad M := 0$
$\quad G := \|b\|_\infty$
$\quad$**for** $i = m : -1 : 1$ **do**
$\qquad M := \max\{G/|U(i,i)|, M\}$
$\qquad G := G * (1 + \|U(1:i-1, i)\|_\infty / |U(i,i)|)$
$\quad$**if** $M < \Omega$ and $G < \Omega$ **then**
$\qquad \text{TRSV}(U, b)$ $\qquad\qquad\qquad\qquad\qquad\qquad\triangleright$ xTRSV
$\quad$**else**
$\qquad G := \|b\|_\infty$
$\qquad$**for** $i = m : -1 : 1$ **do**
$\qquad\quad \mathcal{I}_0 := 1 : i - 1$
$\qquad\quad M := |b(i)/U(i,i)|$ $\qquad\qquad\qquad\triangleright$ Assume $U(i,i) \neq 0$
$\qquad\quad$**if** $M \geq \Omega$ **then**
$\qquad\qquad$Choose $t \in (0,1)$ so that $t * M < \Omega$
$\qquad\qquad b := t * b$ $\qquad\qquad\qquad\qquad\qquad\qquad\triangleright$ xSCAL
$\qquad\qquad s := t * s$
$\qquad\qquad M := t * M$
$\qquad\qquad G := t * G$
$\qquad\quad b(i) := b(i)/U(i,i)$ $\qquad\qquad\qquad\triangleright$ Diagonal step
$\qquad\quad G := G + M * \|U(1:i-1, i)\|_\infty$
$\qquad\quad$**if** $G \geq \Omega$ **then**
$\qquad\qquad$Choose $t \in (0,1)$ so that $t * G < \Omega$
$\qquad\qquad b := t * b$ $\qquad\qquad\qquad\qquad\qquad\qquad\triangleright$ xSCAL
$\qquad\qquad s := t * s$
$\qquad\qquad M := t * M$
$\qquad\qquad G := t * G$
$\qquad\quad b(\mathcal{I}_0) := b(\mathcal{I}_0) - b(i) * U(\mathcal{I}_0, i)$ $\qquad\triangleright$ Substitution step (xAXPY)

In the best-case scenario, i.e. if U is well-conditioned, forming the initial growth bounds is the only additional work compared to standard back substitution. Note that these bounds require computing norms of the columns of U, excluding the diagonal. In the worst-case scenario, rescaling b at each step in back substitution will triple the flop count. However, we find in practice that rescaling is a relatively rare event, even in ill-conditioned and singular matrices.

Safeguarded back substitution is easily generalized to the safe multi-shift triangular solve problem. Given nonzero right-hand side vectors $b_1, \cdots, b_n$ and shifts $\lambda_1, \cdots, \lambda_n$, we seek nonzero solution vectors $x_1, \cdots, x_n$ and scaling factors $s_1, \cdots, s_n$ in $[0,1]$ such that

$$(U - \lambda_j I) x_j = s_j b_j. \qquad\qquad (8)$$

For safeguarded, blocked back substitution, we just need to ensure that the matrix multiplications in the substitution step do not cause numerical issues:

Algorithm 5 Safe multi-shift triangular solve with safeguarded, blocked back substitution

---

   procedure SAFEMULTISHIFTTRSM(U, λ, B, s) ▷ B is overwritten with X
      $s := [1, \cdots, 1]^T$ ▷ n entries
      for $j = 1 : n$ **do**
         $G(j) := \|B(:, j)\|_\infty$
      for $i = m - n_b + 1 : -n_b : 1$ **do**   ▷ Assume m is a multiple of n_b
         $\mathcal{I}_0 := 1 : i - 1$
         $\mathcal{I}_1 := i : i + n_b - 1$
         $\mathcal{I}_2 := i + n_b : m$
         for $j = 1 : n$ **do**
            SAFETRSV$(U(\mathcal{I}_1, \mathcal{I}_1) - \lambda(j) * I, B(\mathcal{I}_1, j), t)$ ▷ Diagonal block step
            if $t < 1$ **then**
               $B(\mathcal{I}_0, j) := t * B(\mathcal{I}_0, j)$ ▷ xSCAL
               $B(\mathcal{I}_2, j) := t * B(\mathcal{I}_2, j)$ ▷ xSCAL
               $s(j) := t * s(j)$
               $G(j) := t * G(j)$
         for $j = 1 : n$ **do**
            $G(j) := G(j) + \sum_{k \in \mathcal{I}_1} \|U(\mathcal{I}_0, k)\|_\infty \|B(\mathcal{I}_1, j)\|_\infty$
            if $G(j) \geq \Omega$ **then**
               Choose $t \in (0, 1)$ so that $t * G(j) < \Omega$
               $B(:, j) := t * B(:, j)$ ▷ xSCAL
               $s(j) := t * s(j)$
               $G(j) := t * G(j)$
         $B(\mathcal{I}_0, :) := B(\mathcal{I}_0, :) - U(\mathcal{I}_0, \mathcal{I}_1) * B(\mathcal{I}_1, :)$ ▷ Substitution step (xGEMM)

Note that each application of SAFETRSV requires norms of the columns of $U(\mathcal{I}_1, \mathcal{I}_1)$, excluding the diagonal, in order to form growth bounds. Thus, we can improve performance in the diagonal block step by reusing this data for each right-hand side.

4   Eigenvector Computation

We shall now apply a safe multi-shift triangular solve to compute the eigenvectors of a general $n \times n$ matrix A. Assuming that A is nondefective, we seek an eigenvalue decomposition $A = X \Lambda X^{-1}$ where Λ is a diagonal eigenvalue matrix and X an eigenvector matrix.[3] We begin by computing the Schur decomposition $A = Q T Q^H$ where T is upper triangular and Q unitary. In LAPACK's general eigensolver routine xGEEV, this is performed efficiently with

---

[3] If A is defective, a nondefective matrix can be obtained with a small perturbation of A.

Level 3 BLAS by converting A to upper Hessenberg form (xGEHRD) [17], converting Householder transforms to a unitary matrix (xUNGHR), and applying the QR algorithm (xHSEQR) [4,5]. Since similar matrices have identical eigenvalues, Λ can be obtained by simply reading off the diagonal of T. Now, all that remains is to find a triangular eigenvector matrix Z such that $T = Z\Lambda Z^{-1}$ and to compute the back substitution $X = QZ$. Assuming that Z is upper triangular and letting λ_k and z_k respectively denote the kth eigenvalue and triangular eigenvector, we require

$$\begin{bmatrix} T_{11} & u & T_{13} \\ 0 & \lambda_k & v^T \\ 0 & 0 & T_{33} \end{bmatrix} \begin{bmatrix} \hat{z} \\ s \\ 0 \end{bmatrix} = \lambda_k \begin{bmatrix} \hat{z} \\ s \\ 0 \end{bmatrix}. \tag{9}$$

This system is satisfied if and only if $\hat{z}$ is a solution to a $k - 1 \times k - 1$ shifted triangular system,

$$(T_{11} - \lambda_k I)\,\hat{z} = -su. \tag{10}$$

This is similar to the form of the safe multi-shift triangular solve problem, but each triangular system has a different size. LAPACK's triangular eigenvector routine xTREVC approaches this problem by calling xLATRS for each triangular eigenvector and back transforming with calls to xGEMV. These routines are limited to Level 2 BLAS and hence achieve poor performance. On multicore architectures, this procedure can be accelerated by performing xLATRS in parallel and by blocking the back substitution into Level 3 BLAS xGEMM calls [11]. However, a hardware-independent solution is preferable for the sake of portability. Observe that (10) holds if and only if

$$\left(\begin{bmatrix} T_{11} & u & T_{13} \\ 0 & \lambda_k & v^T \\ 0 & 0 & T_{33} \end{bmatrix} - \lambda_k I \right) \begin{bmatrix} \hat{z} \\ 0 \\ 0 \end{bmatrix} = s \begin{bmatrix} -u \\ 0 \\ 0 \end{bmatrix}. \tag{11}$$

This is the form for a safe multi-shift triangular solve.[4] Furthermore, the right-hand side matrix will be strictly upper triangular, so we can shortcut the back substitution to avoid unnecessary flops involving zeros. After computing a safe multi-shift triangular solve, the triangular eigenvectors are obtained by putting the scaling factors on the diagonal of the solution matrix. The algorithm is outlined below:

---

[4]If we apply back substitution to a right-hand vector where the last $n - k + 1$ entries are zero, the corresponding entries in the solution will also be zero.

Algorithm 6 Triangular and general eigensolvers

---

    function TRIANGEIG(T)
        $\lambda := \text{diag}(T)$
        $Z := -T$
        $\text{diag}(Z) := 0$
        $s := [1, \cdots, 1]^T$
        SAFEMULTISHIFTTRSM(T, λ, Z, s)   ▷ Shortcutted to exploit structure
        $\text{diag}(Z) := s$
        return λ, Z
    function EIG(A)
        Compute Schur decomposition $A = QTQ^H$ ▷ `xGEHRD`, `xUNGHR`, `xHSEQR`
        $\lambda, Z := \text{TRIANGEIG}(T)$
        $X := QZ$ ▷ `xGEMM`
        return λ, X

Since the QR algorithm and safe multi-shift triangular solve are both backward stable, this method is also backward stable. Thus, we can utilize bounds on the condition number of the eigenvector problem (LAPACK routines `xTRSNA` and `xTRSEN`) [3] to compute error bounds for the eigenvectors.

5   Pseudospectra Computation

The classical approach to study the behavior of an $n \times n$ matrix A is to analyze the distribution of its eigenvalues. The eigenvalues can can be defined in terms of the matrix resolvent $f_A(z) = (zI - A)^{-1}$,

$$\Lambda(A) = \left\{ z \in \mathbb{C} : (zI - A)^{-1} \text{ is unbounded} \right\}. \tag{12}$$

However, eigenvalues may be insufficient to explain the behavior of A if it is highly nonnormal. In particular, small perturbations to A can dramatically change the eigenvalue distribution. To capture this nonnormal behavior, it is preferable to analyze the pseudospectra of A. Given $\epsilon > 0$, the p-norm ϵ-pseudospectrum of A is defined as

$$\Lambda_\epsilon^p(A) = \left\{ z \in \mathbb{C} : \left\| (zI - A)^{-1} \right\|_p \geq \frac{1}{\epsilon} \right\}. \tag{13}$$

A full theory of pseudospectra is developed in [18]. We focus on the case $p = 2$, although similar results can be obtained with $p = 1$ using a more sophisticated algorithm [13].[5]

---

[5]The Hager/Higham algorithm for computing 1-norm pseudospectra is similar to the Van Loan/Lui algorithm discussed below in that it involves establishing a grid in the complex plane, computing a Schur decomposition, and performing shifted triangular solves with grid points as shifts. Thus, it can be similarly accelerated with blocked multi-shift triangular solves.

Pseudospectra are typically computed by establishing a grid with N points on a region of the complex plane, computing the resolvent norm $\|(zI - A)^{-1}\|_2$ at each grid point z, and visualizing with a contour plotter. Letting $\sigma_{\max}(\cdot)$ and $\sigma_{\min}(\cdot)$ denote the largest and smallest singular values of an input matrix, respectively, we remark that the resolvent norm satisfies

$$\left\|(zI - A)^{-1}\right\|_2 = \sigma_{\max}\left((zI - A)^{-1}\right)$$
$$= \frac{1}{\sigma_{\min}(zI - A)}. \tag{14}$$

Thus, one could naively compute pseudospectra by computing the SVD of $zI - A$ for each grid point z and reporting the reciprocal of the smallest singular value. However, this involves a total of $O(Nn^3)$ flops, which is prohibitively expensive for large matrices unless the grid is very coarse. The Van Loan/Lui algorithm improves on the computational cost by proceeding in two stages [15, 19].[6] It begins by computing a Schur decomposition $A = QTQ^H$ where T is upper triangular and Q unitary (see discussion in Section 4). Since matrix norms are invariant under unitary transformations,

$$\left\|(zI - A)^{-1}\right\|_2 = \left\|\left(zQQ^H - QTQ^H\right)^{-1}\right\|_2$$
$$= \left\|-Q\left(T - zI\right)^{-1} Q^H\right\|_2$$
$$= \left\|(T - zI)^{-1}\right\|_2$$
$$= \sigma_{\max}\left((T - zI)^{-1}\right).$$

Letting $\lambda_{\max}(\cdot)$ denote the largest eigenvalue of a Hermitian matrix,

$$\left\|(zI - A)^{-1}\right\|_2 = \sqrt{\lambda_{\max}\left((T - zI)^{-H}(T - zI)^{-1}\right)}. \tag{15}$$

A Krylov eigensolver like the Lanczos algorithm can estimate the largest eigenvalue of $(T - zI)^{-H}(T - zI)^{-1}$ by repeatedly applying it to a vector. Each matrix product can be computed with two triangular solves, for a total cost of $O(n^2)$ flops. The algorithm is outlined below:

Algorithm 7 Van Loan/Lui algorithm

---

    function VANLOANLUI(A, z)
        Compute Schur decomposition $A = QTQ^H$ $\triangleright$ xGEHRD, xUNGHR, xHSEQR
        for $i = 1 : N$ **do**
            $B := (T - z(i) * I)^{-H}(T - z(i) * I)^{-1}$ $\triangleright$ Not formed explicitly
            $\lambda(i) := \lambda_{\max}(B)$ $\triangleright$ Krylov eigensolver
            $r(i) := \sqrt{\lambda(i)}$
        return r $\triangleright$ Visualize with contour plotter

---

[6]Most authors attribute this algorithm to Lui, but the algorithm was presented (in a different context) by Van Loan more than a decade earlier.

This algorithm takes $O(n^3 + Nn^2)$ flops and has been implemented in the popular Matlab package EigTool [22]. However, we make the additional observation that treating each grid point as a shift puts the triangular solves in the form of a multi-shift triangular solve. We can thus achieve Level 3 BLAS performance in both the first and second stage of the Van Loan/Lui algorithm.

6   Implementation

Although the algorithms discussed so far are sequential, they are readily parallelizable on distributed-memory architectures. For instance, if matrix data for a multi-shift triangular solve is stored element-wise across multiple processes, the diagonal block step can be performed (redundantly) on each process and the substitution step can be performed with a distributed matrix product. Sequential and parallel versions of the above algorithms are implemented as part of Elemental, an open-source C++ library for distributed-memory linear algebra and optimization [16]. Several relevant functions in Elemental are listed below:

- `MultiShiftTrsm` - Multi-shift triangular solve.

- `SafeMultiShiftTrsm` - Safe multi-shift triangular solve.

- `TriangEig` - Triangular eigensolver.

- `Eig` - General eigensolver.

- `SpectralCloud` - Resolvent norm with user-specified complex shifts.

- `SpectralWindow` - Resolvent norm over user-specified grid in complex plane.

- `SpectralPortrait` - Resolvent norm over automatically-determined grid in complex plane.

7   Experimental Results

7.1   Multi-shift Triangular Solve

Numerical experiments with the multi-shift triangular solve and safe multi-shift triangular solve were performed with two 4-core Intel Haswell Xeon E5-2623 v3 CPUs at 3.00 GHz. All calculations were performed with double-precision complex numbers and BLAS calls were performed with Intel MKL. A scaling study of the multi-shift triangular solve and safe multi-shift triangular is presented in Figure 1. The safe multi-shift triangular solve was consistently slower than the standard multi-shift triangular solve, which was in turn slower than the Level 3 BLAS triangular solve routine ZTRSM. The performance of the multi-shift triangular solve and safe multi-shift triangular solve were especially poor when the number of right-hand sides was much larger than the matrix dimension, i.e.

$m \ll n$. However, when $m \geq n$, we see that the performance of the multi-shift triangular solve is typically within a factor of 1.5 of Level 3 BLAS performance and that the safe multi-shift triangular solve is typically within a factor of 2.
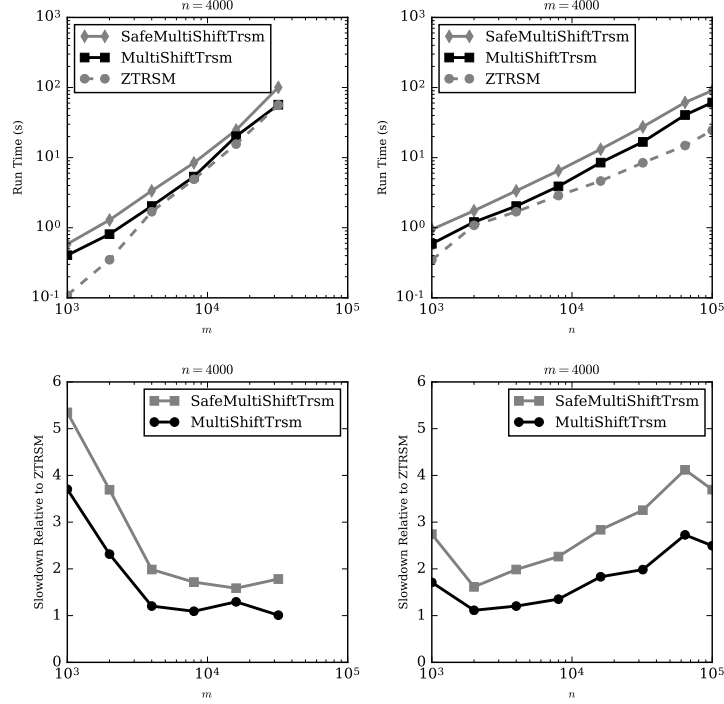

Figure 1: Scaling study of triangular solve, multi-shift triangular solve, and safe multi-shift triangular solve. Matrices were generated by finding Hermitian matrices with uniform random eigenvalues in the interval $[1, 2]$ and deleting entries in the strict lower triangle. Shifts were drawn uniform randomly from the ball $B\,(1.5, 0.5)$.

7.2 Eigenvector Computation

Timing experiments with the triangular and general eigensolvers were performed with the same setup as above (two 4-core Intel Haswell Xeon E5-2623 v3 CPUs at 3.00 GHz, double-precision complex data type, Intel MKL). Scaling studies of the triangular eigensolver and general eigensolver are shown in Figure 2. The triangular eigensolver appears to be asymptotically faster than the LAPACK routine ZTREVC and we report a speedup by a factor of 60 on a 32000×32000 matrix. The general eigensolver appears to converge to a threefold speedup relative to the LAPACK routine ZGEEV. If we inspect the time spent in each stage of the calculation, as shown in Figure 3, we see that Elemental and LAPACK take nearly the same amount of time to compute a Schur decomposition. How-

ever, roughly two thirds of LAPACK's run time takes place in the triangular eigensolver while Elemental's triangular eigensolver takes a negligible amount of time.
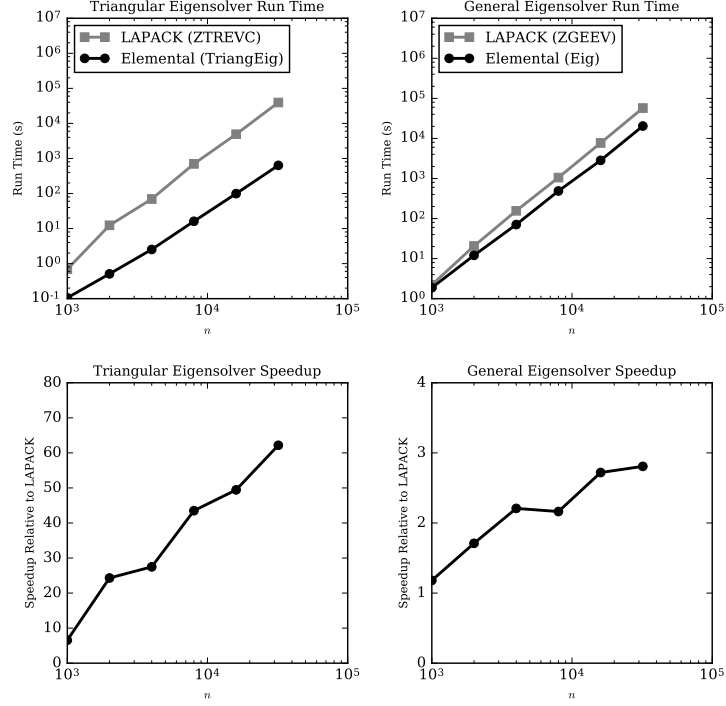

Figure 2: Timing experiments for triangular and general eigensolvers. Matrices were generated by choosing entries uniform randomly from the unit ball.
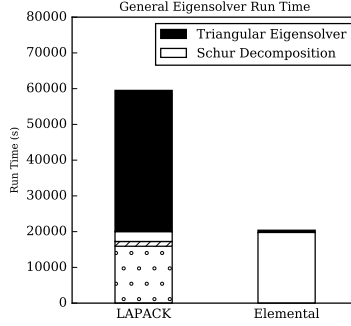
General Eigensolver Run Time

Figure 3: Timings for the general eigensolvers in LAPACK (`ZGEEV`) and Elemental (`Eig`). A 32000×32000 matrix was generated by choosing entries uniform randomly from the unit ball. The timing for LAPACK's Schur decomposition is subdivided into three stages: `ZGEHRD` (dots), `ZUNGHR` (stripes), `ZHSEQR` (no pattern). The timing for Elemental's triangular eigensolver includes back transformation.

The eigenvalue decomposition $A = X \Lambda X^{-1}$ can be validated by computing the relative residual $\|AX - X\Lambda\|_F / \|A\|_F$ and the 2-norm condition number $\|X\|_2 \|X^{-1}\|_2$. As shown in Figure 4,[7] Elemental yields nearly identical results to LAPACK. In particular, the relative residuals are smaller than 10^{-13} and the condition numbers are not singular, suggesting that both eigensolvers achieve reasonable quality.
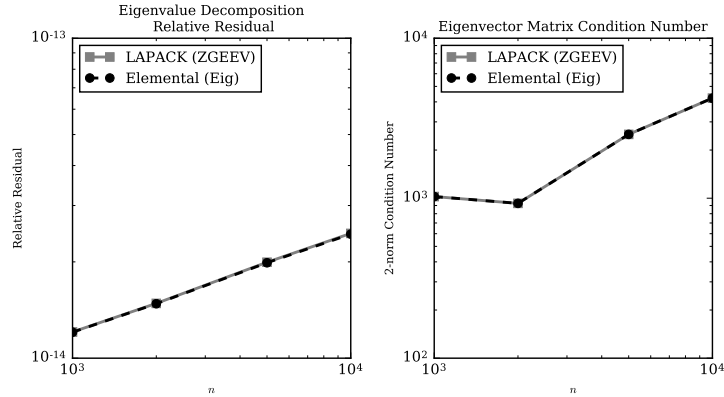

Figure 4: Scaling study for triangular and general eigensolvers. Matrices were generated by choosing entries uniform randomly from the unit ball.

---

[7]These experiments were performed with a 4-core Intel Nehalem Core i7-870 CPU at 2.93 GHz, also using double-precision complex data type and Intel MKL.

7.3 Pseudospectra Computation

Experiments with pseudospectra solvers were performed with a 4-core Intel Nehalem Core i7-870 CPU at 2.93 GHz. Calculations were performed with double-precision complex numbers and BLAS calls were performed with Intel MKL (for both Elemental and MATLAB). Pseudospectra computed by Elemental and EigTool are qualitatively identical, as shown in Figure 5.
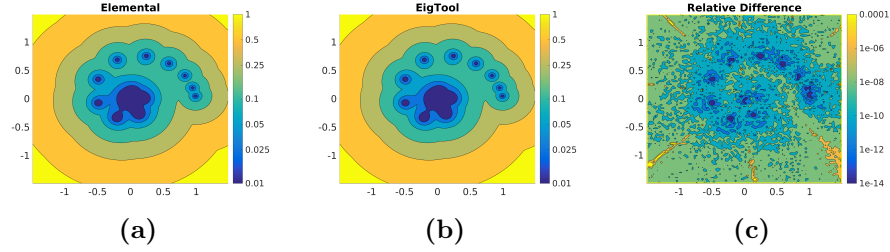

(a) (b) (c)

Figure 5: ϵ-pseudospectra with several values of ϵ and the relative difference in computed resolvent norm. The matrix is a 100×100 discretization of the Fox/Li operator with parameter $F = 10$ and the image is constructed from the resolvent norm computed at 10000 grid points [10]. See Chapter 60 of [18] for a discussion on the pseudospectra of the Fox/Li operator.

Scaling studies of pseudospectra computation are shown in Figure 6. Given sufficiently many grid points, we see that Elemental achieves a speedup by a factor of 1.6 when computing pseudospectra of a 100×100 matrix. This modest result can be explained by noting that Krylov eigensolvers like the Lanczos method or Arnoldi method require tridiagonal or Hessenberg eigensolvers.[8] When the matrix dimension is small, these eigensolvers take a non-trivial portion of the computation. However, when the matrix dimension is large, the computation is dominated by triangular solves. In this regime, Elemental's pseudospectra solver fully exploits Level 3 BLAS performance and achieves a ninefold speedup when computing the resolvent norm of a 3200×3200 matrix at 10000 grid points.

---

[8] EigTool uses the Lanczos method for sufficiently large matrices and Elemental implements a variety of eigensolvers. Our experiments with Elemental use the implicitly restarted Arnoldi method.
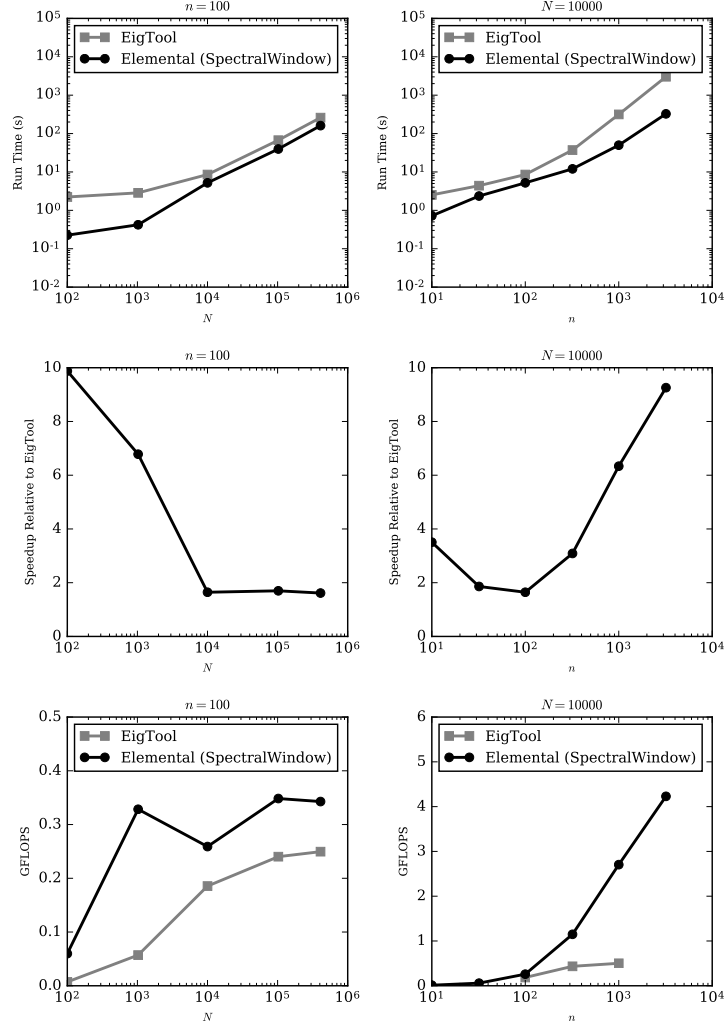
Figure 6: Scaling study for pseudospectra computation on discretizations of the Fox/Li operator with parameter $F = 10$ [10].

8 Conclusion

We have shown that multi-shift triangular solves can be performed efficiently by exploiting Level 3 BLAS routines. Numerical experiments suggest that using a safe multi-shift triangular solve to compute triangular eigenvectors is substantially faster than the algorithm presently used in LAPACK, yielding a 60x speedup when computing the eigenvectors of a triangular matrix and a 3x speedup for a general matrix. Pseudospectra computation with multi-shift

triangular solves also appears to exhibit a ninefold speedup relative to EigTool when the matrix dimension is sufficiently large. Future work will include real arithmetic multi-shift triangular solves and Fortran implementations for inclusion into LAPACK.

9   Acknowledgments

A   Generalized Multi-Shift Triangular Solve

The multi-shift triangular solve problem can be generalized and applied to compute generalized eigenvectors. Given $m \times m$ upper triangular matrices U and V, right-hand side vectors $b_1, \cdots, b_n$, and shifts $\lambda_1, \cdots, \lambda_n$, we seek solution vectors $x_1, \cdots, x_n$ such that

$$(U - \lambda_j V) \, x_j = b_j. \tag{16}$$

This problem can be solved with small modifications to Algorithm 3:

Algorithm 8 Generalized multi-shift triangular solve with blocked back substitution

---
    procedure GENERALIZEDMULTISHIFTTRSM(U, V, λ, B) ▷ B is overwritten
                                                             with X
        for $i = m - n_b + 1 : -n_b : 1$ **do**   ▷ Assume m is a multiple of n_b
          $\mathcal{I}_0 := 1 : i - 1$
          $\mathcal{I}_1 := i : i + n_b - 1$
          for $j = 1 : n$ **do**
              $U' := U(\mathcal{I}_1, \mathcal{I}_1) - \lambda(j) * V(\mathcal{I}_1, \mathcal{I}_1)$ ▷ xAXPY
              TRSV$(U', B(\mathcal{I}_1, j))$ ▷ xTRSV
              $C(:, j) := \lambda(j) * B(\mathcal{I}_1, j)$ ▷ xSCAL
          $B(\mathcal{I}_0, :) := B(\mathcal{I}_0, :) - U(\mathcal{I}_0, \mathcal{I}_1) * B(\mathcal{I}_1, :)$ ▷ xGEMM
          $B(\mathcal{I}_0, :) := B(\mathcal{I}_0, :) + V(\mathcal{I}_0, \mathcal{I}_1) * C$ ▷ xGEMM

Making similar changes to Algorithm 5 yields a robust algorithm:

Algorithm 9 Safe, generalized multi-shift triangular solve with safeguarded, blocked back substitution

---

procedure SAFEGENERALIZEDMULTISHIFTTRSM(U, V, λ, B, s) ▷ B is over-
written
with X

$s := [1, \cdots, 1]^T$ ▷ n entries
for $j = 1 : n$ **do**
$\quad G(j) := \|B(:, j)\|_\infty$
for $i = m - n_b + 1 : -n_b : 1$ **do** ▷ Assume m is a multiple of n_b
$\quad \mathcal{I}_0 := 1 : i - 1$
$\quad \mathcal{I}_1 := i : i + n_b - 1$
$\quad \mathcal{I}_2 := i + n_b : m$
$\quad$**for** $j = 1 : n$ **do**
$\quad\quad U' := U(\mathcal{I}_1, \mathcal{I}_1) - \lambda(j) * V(\mathcal{I}_1, \mathcal{I}_1)$ ▷ xAXPY
$\quad\quad$SAFETRSV$(U', B(\mathcal{I}_1, j), t)$ ▷ Diagonal block step
$\quad\quad$**if** $t < 1$ **then**
$\quad\quad\quad B(\mathcal{I}_0, j) := t * B(\mathcal{I}_0, j)$ ▷ xSCAL
$\quad\quad\quad B(\mathcal{I}_2, j) := t * B(\mathcal{I}_2, j)$ ▷ xSCAL
$\quad\quad\quad s(j) := t * s(j)$
$\quad\quad\quad G(j) := t * G(j)$
$\quad\quad C(:, j) := \lambda(j) * B(\mathcal{I}_1, j)$ ▷ xSCAL
$\quad$**for** $j = 1 : n$ **do**
$\quad\quad G(j) := G(j) + \sum_{k \in \mathcal{I}_1} \left(\|U(\mathcal{I}_0, k)\|_\infty + |\lambda(j)| \|V(\mathcal{I}_0, k)\|_\infty \right) \|B(\mathcal{I}_1, j)\|_\infty$
$\quad\quad$**if** $G(j) \geq \Omega$ **then**
$\quad\quad\quad$Choose $t \in (0, 1)$ so that $t * G(j) < \Omega$
$\quad\quad\quad B(:, j) := t * B(:, j)$ ▷ xSCAL
$\quad\quad\quad s(j) := t * s(j)$
$\quad\quad\quad G(j) := t * G(j)$
$\quad B(\mathcal{I}_0, :) := B(\mathcal{I}_0, :) - U(\mathcal{I}_0, \mathcal{I}_1) * B(\mathcal{I}_1, :)$ ▷ xGEMM
$\quad B(\mathcal{I}_0, :) := B(\mathcal{I}_0, :) + V(\mathcal{I}_0, \mathcal{I}_1) * C$ ▷ xGEMM

This routine can be used to solve the generalized eigenvalue problem:

Algorithm 10 Generalized, triangular and generalized, general eigensolvers
---
   function GENERALIZEDTRIANGEIG(T, S)
      $\lambda := \text{diag}(T)./\text{diag}(S)$
      $Z := -T + S * \text{diag}(\lambda)$
      $\text{diag}(Z) := 0$
      $s := [1, \cdots, 1]^T$
      SAFEGENERALIZEDMULTISHIFTTRSM(T, S, λ, Z, s)
      $\text{diag}(Z) := s$
      return λ, Z
   function GENERALIZEDEIG(A)
      Compute generalized Schur decomposition $A = QTP^H, B = QSP^H$
      $\lambda, Z := $ GENERALIZEDTRIANGEIG(T, S)
      $X := PZ$ ▷ `xGEMM`
      return λ, X

References

[1] E Anderson, Z Bai, C Bischof, S Blackford, J Demmel, J Dongarra, J Du Croz, A Greenbaum, S Hammarling, A McKenney, and D Sorensen. *LAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] Edward Anderson. *Robust triangular solves for use in condition estimation.* University of Tennessee. Computer Science Department, 1991.

[3] Z Bai, James Demmel, and A McKenney. On the conditioning of the nonsymmetric eigenproblem: Theory and software. 1989.

[4] Karen Braman, Ralph Byers, and Roy Mathias. The multishift QR algorithm. part I: Maintaining well-focused shifts and level 3 performance. *SIAM Journal on Matrix Analysis and Applications*, 23(4):929–947, 2002.

[5] Karen Braman, Ralph Byers, and Roy Mathias. The multishift QR algorithm. part II: Aggressive early deflation. *SIAM Journal on Matrix Analysis and Applications*, 23(4):948–973, 2002.

[6] Jack J Dongarra, Jermey Du Cruz, Sven Hammerling, and Iain S Duff. Algorithm 679: A set of level 3 basic linear algebra subprograms: model implementation and test programs. *ACM Transactions on Mathematical Software*, 16(1):18–28, 1990.

[7] Jack J Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain S Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[8] Jack J Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J Hanson. Algorithm 656: an extended set of basic linear algebra subprograms:

model implementation and test programs. *ACM Transactions on Mathematical Software*, 14(1):18–32, 1988.

[9] Jack J Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J Hanson. An extended set of Fortran basic linear algebra subroutines. *ACM Transactions on Mathematical Software*, 14(1):1–17, 1988.

[10] Arthur G Fox and Tingye Li. Resonant modes in a maser interferometer. *Bell System Technical Journal*, 40(2):453–488, 1961.

[11] Mark Gates, Azzam Haidar, and Jack Dongarra. Accelerating computation of eigenvectors in the dense nonsymmetric eigenvalue problem. In *High Performance Computing for Computational Science–VECPAR 2014*, pages 182–191. Springer, 2014.

[12] Gene H Golub and Charles F Van Loan. *Matrix Computations*. The John Hopkins University Press, 4 edition, 2013.

[13] Nicholas J Higham and Françoise Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1185–1201, 2000.

[14] Chuck L Lawson, Richard J. Hanson, David R Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, 1979.

[15] SH Lui. Computation of pseudospectra by continuation. *SIAM Journal on Scientific Computing*, 18(2):565–573, 1997.

[16] Jack Poulson, Bryan Marker, Robert A Van de Geijn, Jeff R Hammond, and Nichols A Romero. Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software*, 39(2):13, 2013.

[17] Gregorio Quintana-Ortí and Robert van de Geijn. Improving the performance of reduction to Hessenberg form. *ACM Transactions on Mathematical Software*, 32(2):180–194, 2006.

[18] Lloyd Nicholas Trefethen and Mark Embree. *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*. Princeton University Press, 2005.

[19] Charles Van Loan. How near is a stable matrix to an unstable matrix? Technical report, Cornell University, 1984.

[20] Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. AUGEM: automatically generate high performance dense linear algebra kernels on x86 CPUs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 25. ACM, 2013.

[21] R Clint Whaley and Antoine Petitet. Minimizing development and mainte-
     nance costs in supporting persistently optimized BLAS. *Software: Practice
     and Experience*, 35(2):101–121, 2005.

[22] Thomas G Wright. EigTool. `http://www.comlab.ox.ac.uk/
     pseudospectra/eigtool/`, 2002.

[23] Xianyi Zhang, Qian Wang, and Yunquan Zhang. Model-driven level 3
     BLAS performance optimization on Loongson 3A processor. In *2012 IEEE
     18th International Conference on Parallel and Distributed Systems*, pages
     684–691. IEEE, 2012.