Design, Usage and Features of ⟨T⟩LAPACK
github.com/tlapack/tlapack

Julien Langou[1,2] Weslley Pereira[3] Johnathan Rhyne[1]
Thijs Steel[4]

[1]University of Colorado Denver, Denver - CO, USA
[2]Inria, France
[3]National Renewable Energy Laboratory, Golden - CO, USA
[4]KU Leuven, Belgium

SIAM LA Minitutorial
May 14, 2024

Agenda

- General introduction to ⟨T⟩LAPACK (T. Steel)
- Experience of a new developer (J. Rhyne)
- Level 0 (J. Langou)
- Mixed precision and fp8 (J. Langou)
- Rotation sequences (T. Steel)
- Some hands on exercises

What is ⟨T⟩LAPACK ?

github.com/tlapack/tlapack

- C++ Template-based Linear Algebra Package.
- Routines are free functions using high-level abstraction.
- Interoperability with: SLATE, mdspan, Eigen3, StarPU.

```cpp
// User's code looks like:

auto A = std::mdspanA( A_ptr, 100, 100 );
auto B = Eigen::MatrixXd::Random(100,100).eval();

int infoA = getrf( A, piv );
int infoB = getrf( B, piv );
```

```cpp
// Inside potrf we find commands like:

A(piv[0], 0) = A(0, 0);
auto A00 = slice(A, range(0, k0), range(0, k0));
```

Contributors

Senior Team Members

- Julien Langou, CU Denver
- Weslley Pereira, CU Denver
- Thijs Steel, KU Leuven

Undergrads, Grads, Postdocs (CU Denver)

- Heidi Meier, Lindsay Slager, Naomi Wilson, Natnael Tebeje, Yuxin Cai (Summer 2022, undergraduate)
- Ali Lotfi (Fall 2022, postdoc)
- Michael Pultorak, Racheal Asamoah, Skylar Johns (Spring 2023, undergraduate)
- Jonhathan Rhyne (AY 22-23, graduate)
- Alexander Hatle, Aslak Djupskas, Dominic Lioce, Hugh Kadhem, Seok-Ho Yoon, Sudhanva Kulkarni (Fall 2023, undergraduate)

Why ⟨T⟩LAPACK ?

LAPACK

- provides reference implementation for a wide set of state-of-the-art algorithms.
- is largely used for its efficiency and reliability.
- has been incorporated into several packages.

BLAS / LAPACK

- are not templated, currently support 4 data types.
- are not templated, replicate a single algorithm on 4 files.
- work on a few data layouts.

⟨T⟩LAPACK

- uses templates that encompass data types and layouts.
- interoperates with SLATE, mdspan and Eigen3.
- uses modern software standards.

High-level design

Templates

- Any type works
- Any matrix layout works (but need to write plugin)
  - Access through ()
  - Submatrix through slice()
  - sizes through nrows, ncols
  - ...
- float, double, Eigen::Half, std::complex
- Your own matrix, Eigen, mdspan, ...
- All the same interface!

Power of templates: example

- Magma (GPU) has: gehrd and orghr, but not ormhr (Hessenberg factorization and applying the orthogonal factor)
- ormhr is just a wrapper around ormqr

```
1       CALL DORMQR( SIDE, TRANS, MI, NI, NH, A( ILO+1,
    ILO ), LDA, TAU( ILO ), C( I1, I2 ), LDC, WORK,
    LWORK, IINFO )
2
```

- In ⟨T⟩LAPACK , ormhr would not need to be rewritten for GPU!

High-level design

Optional arguments

- Workspace (allocated if not provided)
- Blocksize
- Algorithm selection
- Many parameters can be runtime or compile time

```
1 PotrfOpts opts;
2 opts.variant = PotrfVariant::Blocked;
3 opts.nb = 64;
4 potrf(Uplo::Lower, A, opts);
```

High-level design

Dimension errors

- Matrix and vector objects know their dimension
- Bounds checks on element access
- Checks for compatible dimensions in routines
- Bounds checks on submatrix/subvector
- Runtime checks can be disabled for performance

```
1 // A is a 5x5 matrix, x and y are size 4 vectors
2 // This throws an error
3 A(5,2) += 1.5;
4 // This throws an error
5 x[4] = 3;
6 // This throws an error
7 gemv( Op::NoTrans, 1.0, A, x, 0.0, y);
8 // This throws an error
9 auto A2 = slice(A, pair{0,5}, pair{3,6});
```

High-level design

Testing and debugging

- Let's be honest, the LAPACK test suite is not great
- ⟨T⟩LAPACK offers:
  - Unit tests with an actual framework! (Catch2)
  - Synergy with bounds checks
  - Visualizer for debugging
- Less developer time spent on simple issues, more time spent on cool algorithms!

High-level design

Optimized BLAS

- Paradigm of LAPACK: use highly optimized implementations of matrix-matrix multiplication for portable high-performance.
- $\langle$T$\rangle$LAPACK uses BLAS++ and LAPACK++ from SLATE to access optimized BLAS and LAPACK.

High-level design

Open question: naming scheme

- In Fortran 77: limited amount of characters for routine names.
- Old names are now commonly used: geqrf, geev, ...
- New names would make much more sense for the future

High-level design

Open question: naming scheme

- In Fortran 77: limited amount of characters for routine names.
- Old names are now commonly used: geqrf, geev, ...
- New names would make much more sense for the future

Audience participation what do these routines do?:

- dpotrf
- dgghrd
- dgeqp3

High-level design

Open question: naming scheme

- In Fortran 77: limited amount of characters for routine names.
- Old names are now commonly used: geqrf, geev, ...
- New names would make much more sense for the future

Audience participation what do these routines do?:

- dpotrf Cholesky factorization
- dgghrd
- dgeqp3

High-level design

Open question: naming scheme

- In Fortran 77: limited amount of characters for routine names.
- Old names are now commonly used: geqrf, geev, ...
- New names would make much more sense for the future

Audience participation what do these routines do?:

- dpotrf Cholesky factorization
- dgghrd Hessenberg-triangular reduction
- dgeqp3

High-level design

Open question: naming scheme

- In Fortran 77: limited amount of characters for routine names.
- Old names are now commonly used: geqrf, geev, ...
- New names would make much more sense for the future

Audience participation what do these routines do?:

- dpotrf Cholesky factorization
- dgghrd Hessenberg-triangular reduction
- dgeqp3 Pivoted QR

High-level design

Generality has a price

- ⟨T⟩LAPACK needs a full reimplementation. Extremely large amount of developer time!
- C++ is not slower, but also not faster.
- Near-term benefit is limited. Most of the benefits are down the line.

Case studies: Cholesky and Nonsymmetric eigenvalues

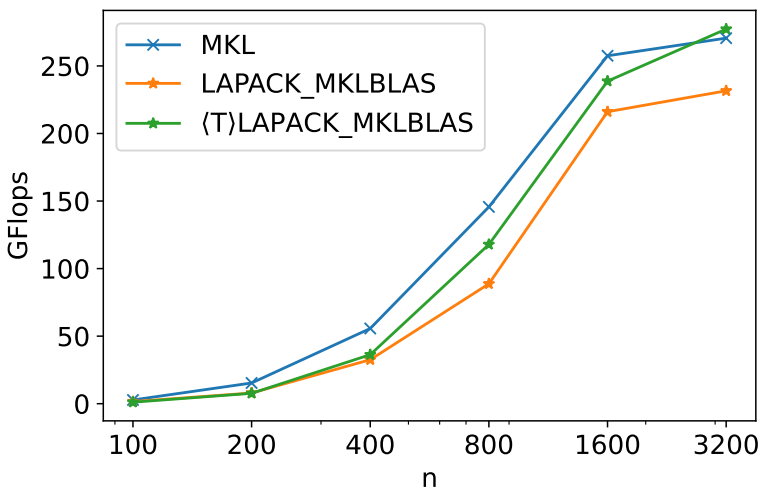
Wrappers to optimized BLAS

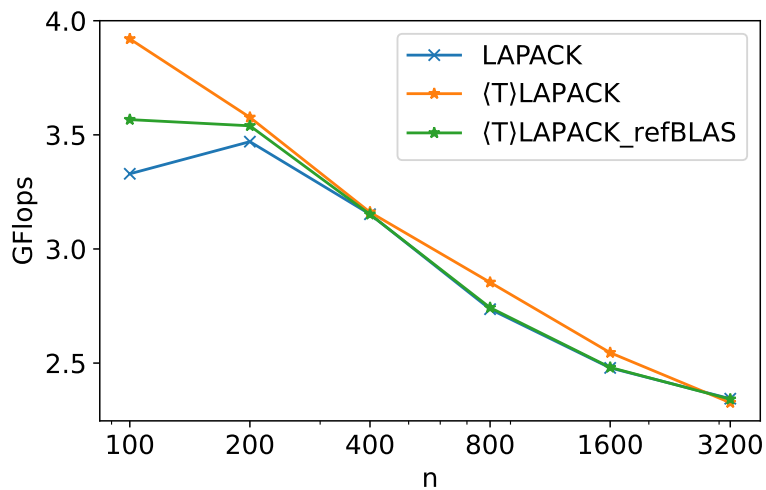
⟨T⟩LAPACK uses BLAS++ and LAPACK++ from SLATE to access optimized BLAS and LAPACK libraries.

```
1  /// <T>LAPACK implementation
2  template<TLAPACK_VECTOR vector_t,
3           disable_if_allow_optblas_t< vector_t > = 0>
4  auto nrm2( const vector_t& x )
5  { ... }
6
7  #ifdef TLAPACK_USE_LAPACKPP
8
9      /// Wrapper to BLAS++
10     template<TLAPACK_LEGACY_VECTOR vector_t,
11              enable_if_allow_optblas_t< vector_t > = 0>
12     auto nrm2( const vector_t& x )
13     {
14         auto x_ = legacy_vector(x);
15         const auto& n = x_.n;
16         return ::blas::nrm2( n, x_.ptr, x_.inc );
17     }
18
19 #endif
```

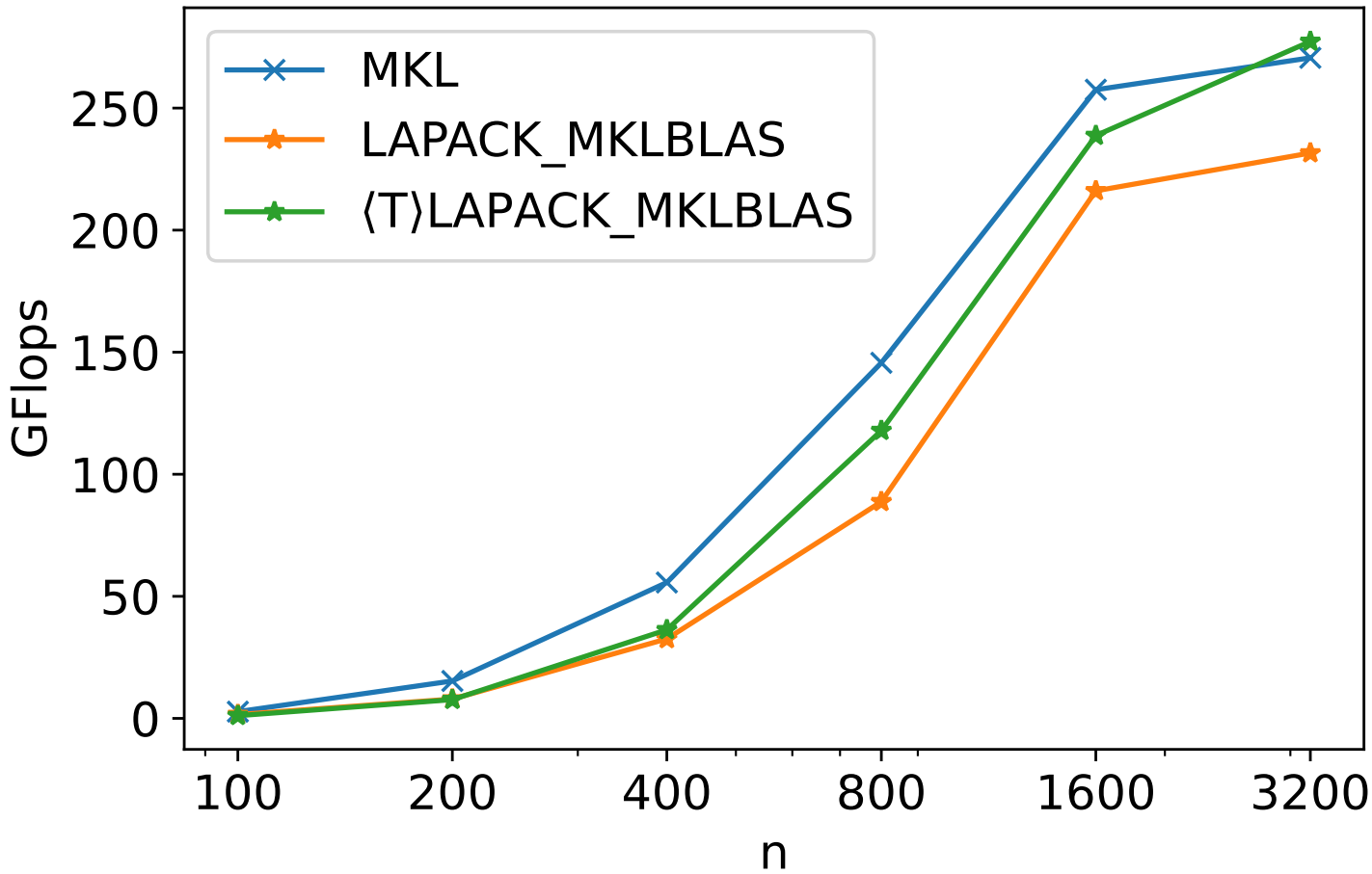
Performance test with recursive Cholesky

- 11th Gen i7-11800H @ 2.30GHz, 16 cores
- gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1 20.04)
- LAPACK version 3.9.0
- oneAPI/MKL version 2022.0.2
- Main code: `github.com/wesslleyspereira/tlapack/blob/try-performance-example/examples/potrf/example_potrf.cpp`
- Double matrix of size n-by-n

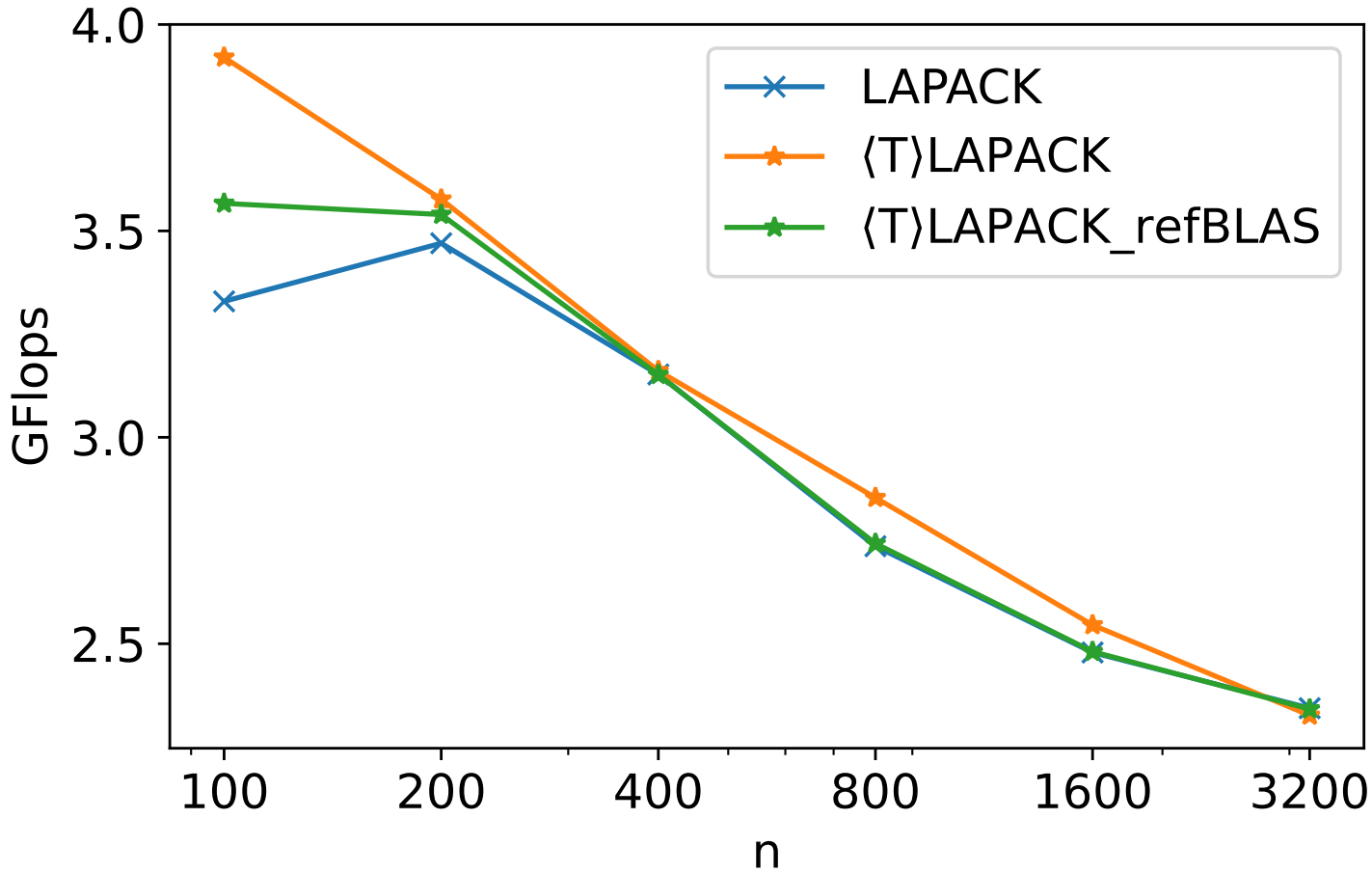




Performance test with recursive Cholesky



Performance test with recursive Cholesky



Integration with StarPU

In a few lines:

- `starpu_data_handle_t`: handle to manage a piece of data. Splits data in a grid of nx-by-ny tiles.
- `tlapack::starpu::Matrix<T>`: matrix abstraction to `starpu_data_handle_t`. Allows for blocking that doesn't match with the nx-by-ny grid.
- ⟨T⟩LAPACK plugin: provides `slice`, `nrows`, `ncols`, etc.

```
1   int main() {
2     /* initialize StarPU here */
3     float *A_;
4     starpu_malloc((void**)&A_, 90 * 90 * sizeof(float));
5     genHermitianMatrix(A_, 90, 90); // 90x90 matrix
6     Matrix<float> A(A_, 90, 90, 20, 20); // 20x20 grid
7
8     potrf_opts_t<size_t> opts; opts.nb = 9; // 9x9 blocks
9     int info = potrf(upperTriangle, A, opts);
10    /* terminate StarPU here */
11  }
12
```

Integration with StarPU

- **Key aspect:** No need to modify top-level routines!
- Blocked Cholesky calls `gemm`, `herk`, `trsm` and `potf2`.
- Those four routines have a StarPU (tiled) implementation.

Preliminary results with $n = 7680$ in single precision:

1. Using only CPU workers:

MKL potrf: 589.0 GFLOPS,
StarPU Cholesky example: 519.3 GFLOPS,
⟨T⟩LAPACK potrf: 477.9 GFLOPS.

2. Using GPU and CPU workers:

StarPU Cholesky example: 1835.1 GFLOPS,
⟨T⟩LAPACK potrf: 1445.6 GFLOPS.

Ubuntu 20.04.6 LTS x86_64. 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz (8 cores). NVIDIA GeForce RTX 3050 (2048 cores, 4GB). Driver version: 530.30.02. CUDA 11.0 (cuBLAS 11.2.0 cuSOLVER 10.6.0). MKL 2023.1.0. GNU compilers, version 9.4.0. StarPU scheduler: dmdas.

Performance of the real eigenvalue solver

Schur decomposition:

$$A = ZTZ^H$$

Two input scenarios:

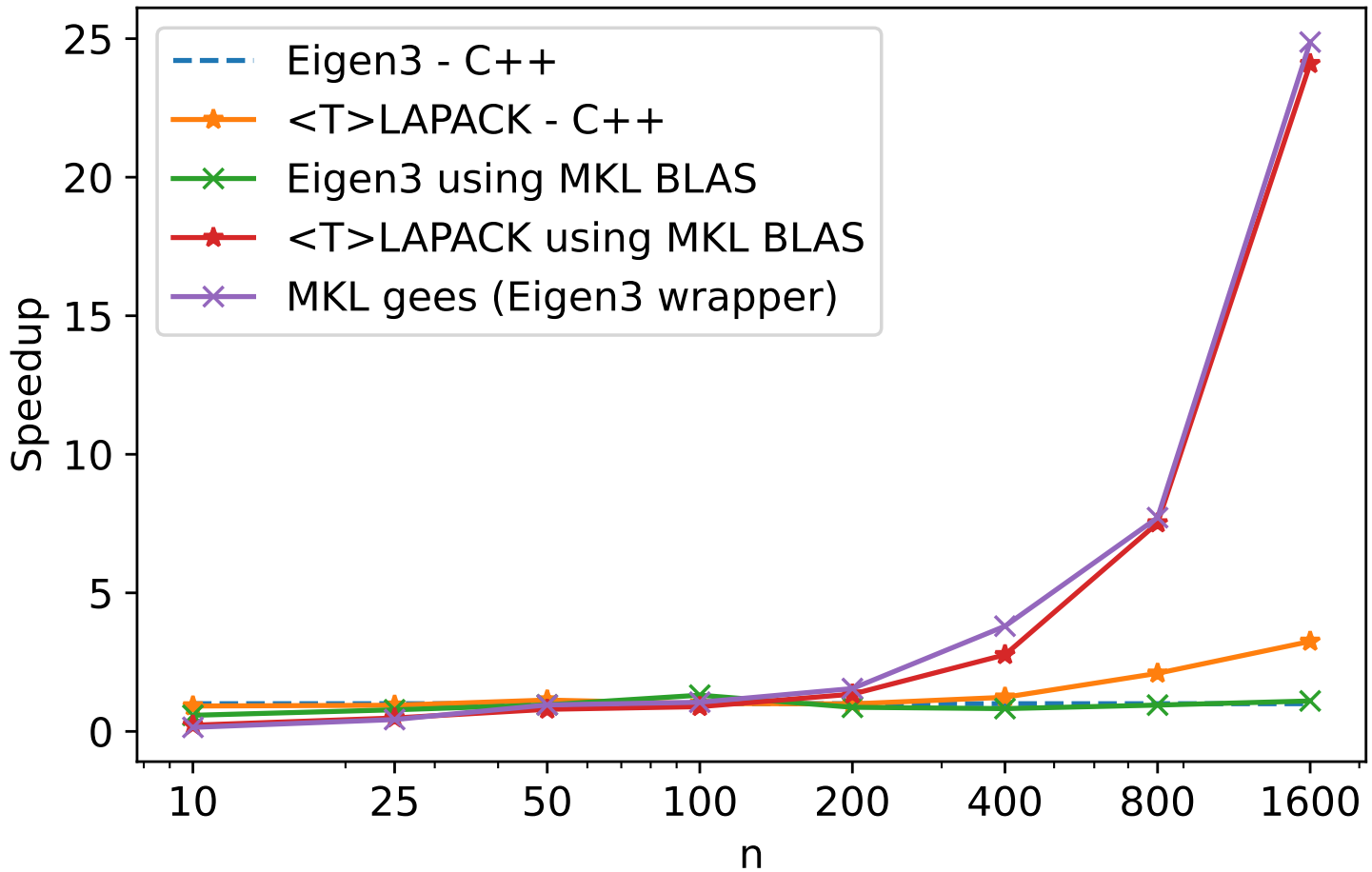
1. Random A, with entries taken uniformly in $[0, 1]$.
2. $A = (Q_1 D Q_2)\Lambda(Q_2^H D^{-1} Q_1^H)$, where
   - Λ is diagonal with entries taken uniformly in $[0, 1]$.
   - D is diagonal with $D_{ii} := 2^{(i \mod (54))}$.
   - Given A_i is random full rank, generate Q_i the Q-factor of the QR factorization of A_i, $i = 1, 2$.

Configurations in the following slides:

- Experiments in double precision.
- Package Eigen3 v3.3.7.
- Package mdspan v0.4.0.
- oneAPI/MKL version 2022.2.1.

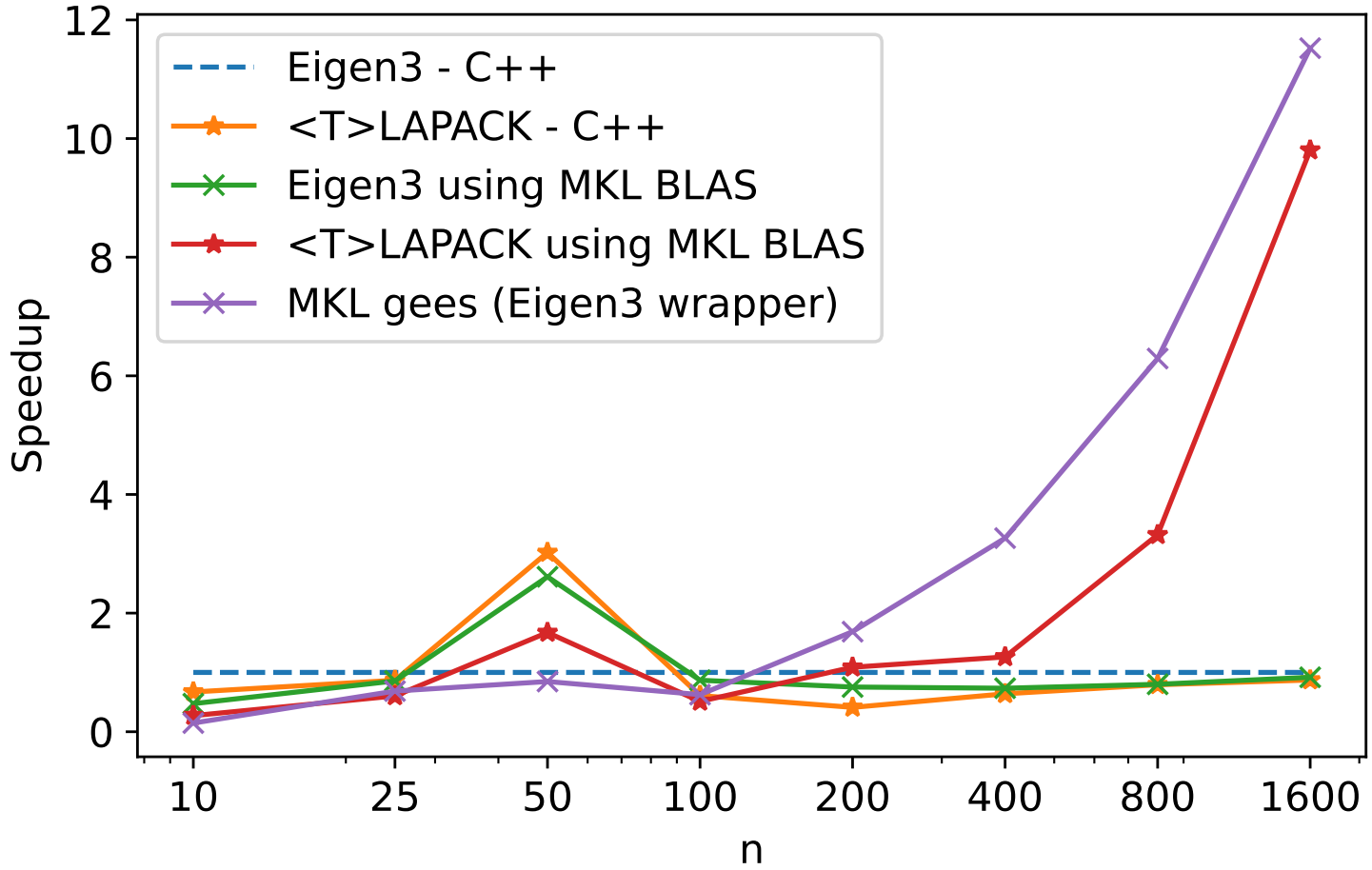
Performance in Scenario 1

Time to compute the eigenvalues compared to Eigen3.



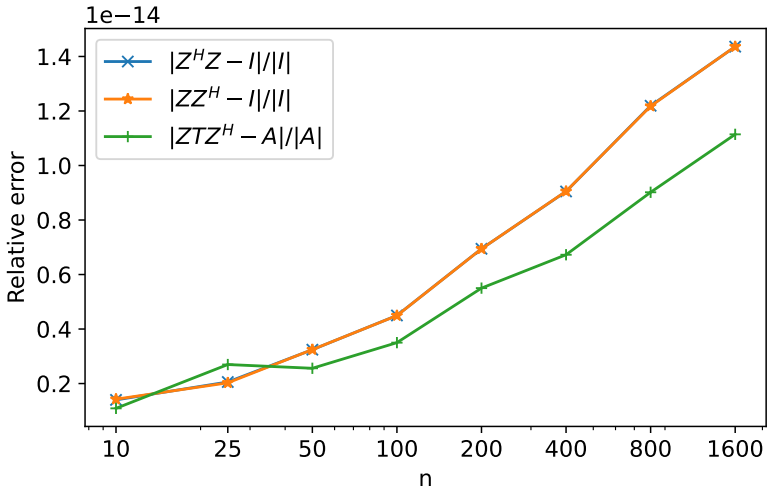
Performance in Scenario 2

Time to compute the eigenvalues compared to Eigen3.

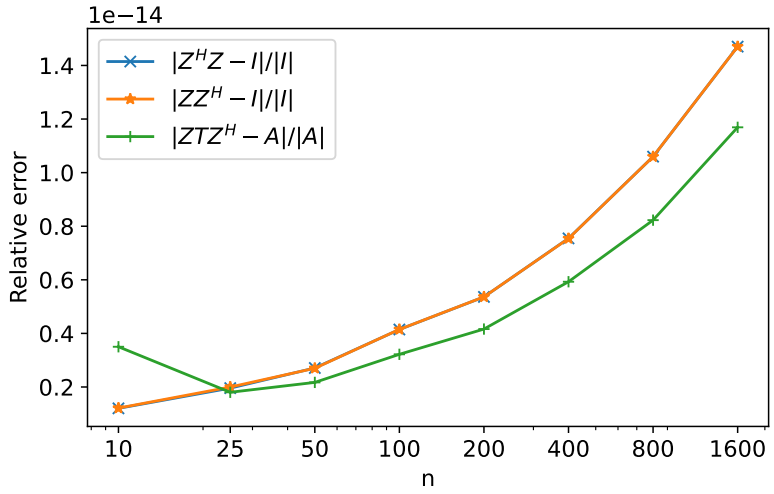


Stability analysis of the C++ codes

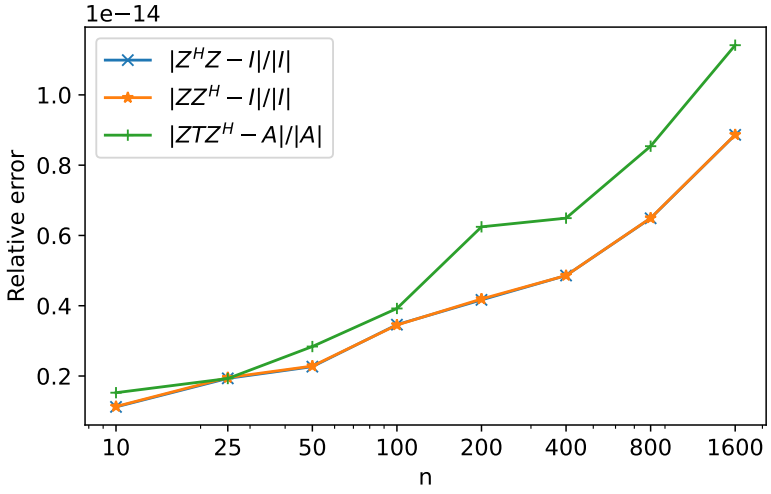
⟨T⟩LAPACK scenario 1



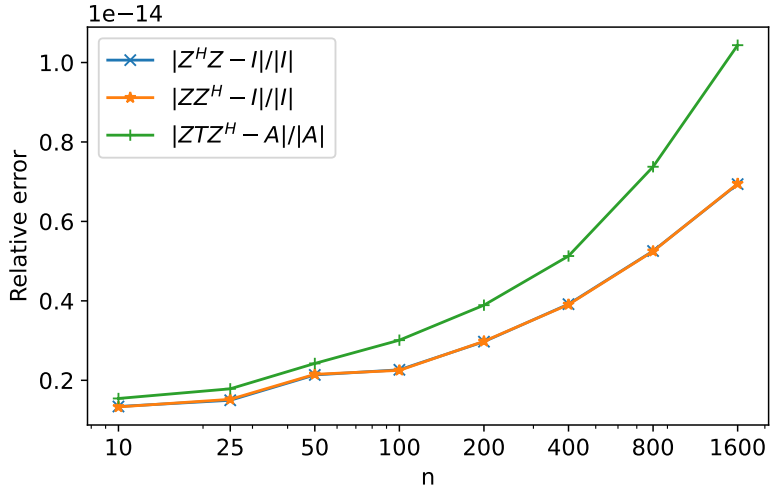
Eigen3 scenario 1



⟨T⟩LAPACK scenario 2

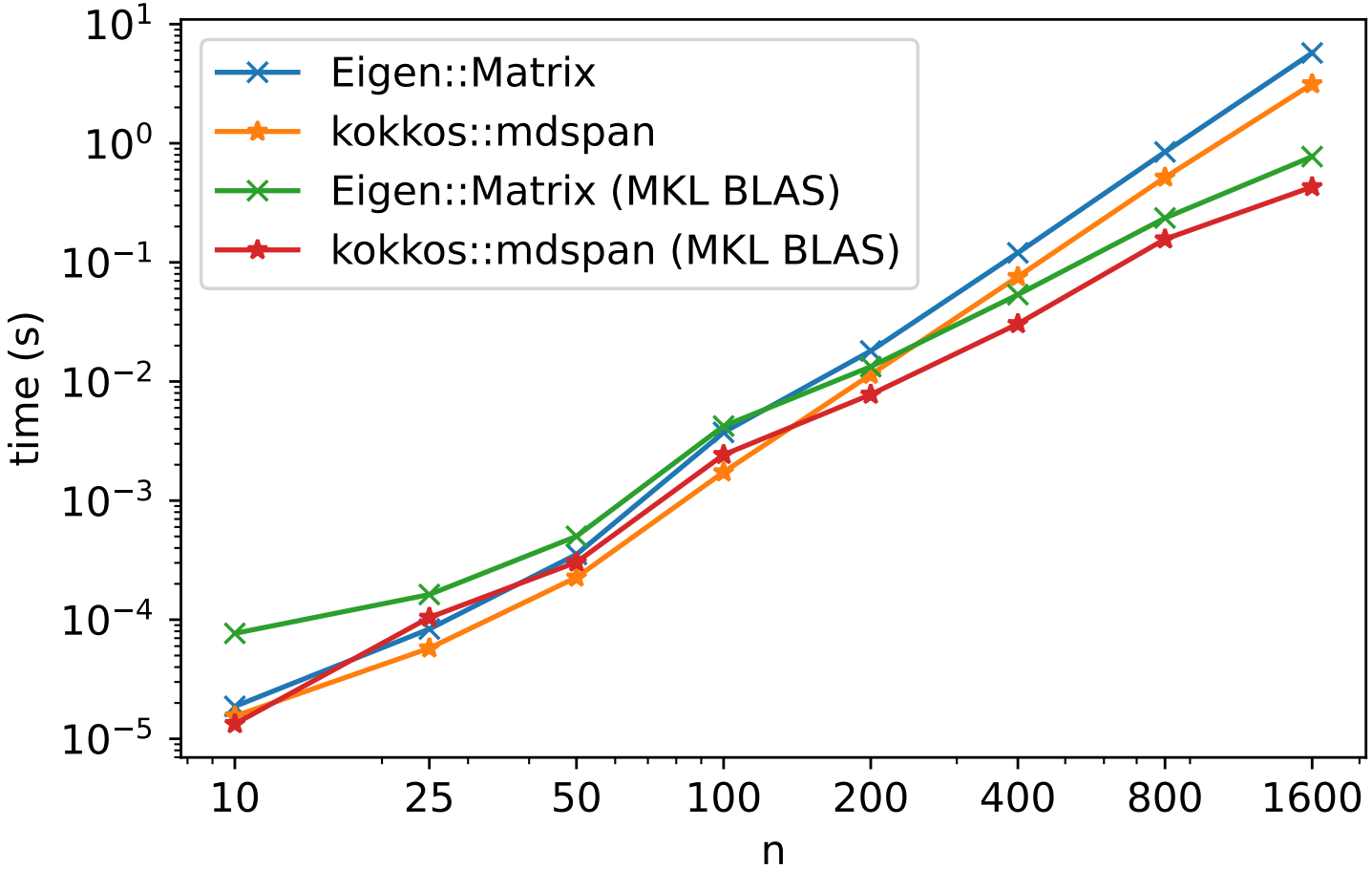


Eigen3 scenario 2



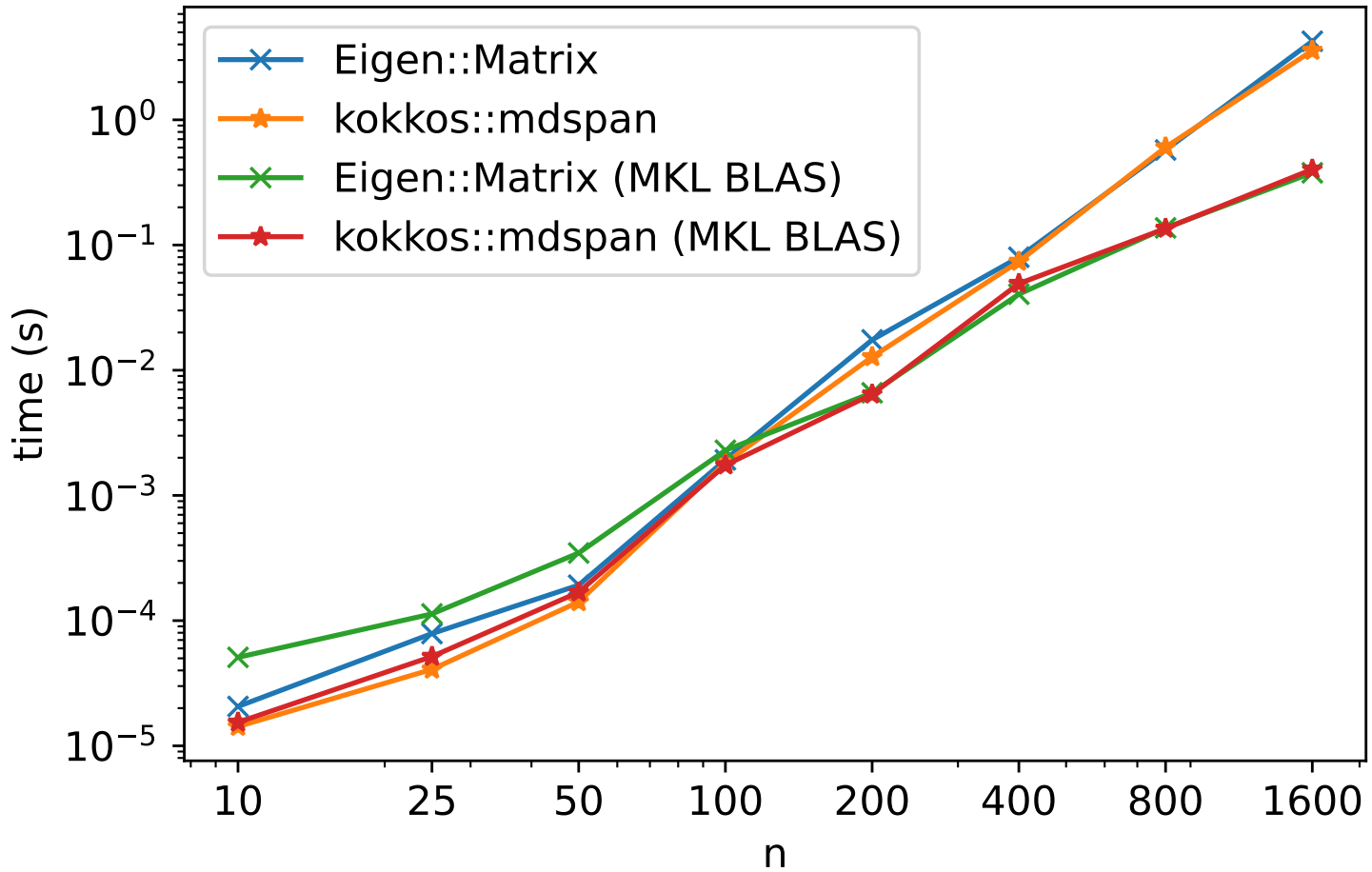
⟨T⟩LAPACK using mdspan from kokkos

Scenario 1 using different data structures with ⟨T⟩LAPACK



⟨T⟩LAPACK using mdspan from kokkos

Scenario 2 using different data structures with ⟨T⟩LAPACK



Code in the last example

Step-by-step (⟨T⟩LAPACK) x Straight-forward (Eigen3)

Example using ⟨T⟩LAPACK implementation:

```
// Hessenberg reduction
Eigen::Matrix<T,-1, 1> tau(n);
tlapack::gehrd( 0, n, A, tau );

// Discard reflectors
for (idx_t j = 0; j < n; ++j)
  for (idx_t i = j + 2; i < n; ++i)
    A(i,j) = T(0);

// Compute eigenvalues and Schur decomposition
Eigen::Matrix<std::complex<T>,-1,1> lambda(n);
Eigen::Matrix<T,-1,-1> Z(n,n); Z.setIdentity();
tlapack::multishift_qr( false, false, 0, n, A, lambda, Z );
```

Example using Eigen3 implementation:

```
Eigen::EigenSolver<decltype(A)> solver( A, false );
auto lambda = solver.eigenvalues();
```

Developing with ⟨T⟩LAPACK from a student perspective

Developing for ⟨T⟩LAPACK

As was said before, the codebase for ⟨T⟩LAPACK is written in header files. This allows us to take advantage of templates!

My project within ⟨T⟩LAPACK

I worked on some software that was for a fork of ⟨T⟩LAPACK . The code can be found here: `https://github.com/jprhyne/tlapack/tree/hqr`, which was investigating porting EISPACK hqr into the ⟨T⟩LAPACK framework.

My project within ⟨T⟩LAPACK

```
template <class matrix_t,
          class vector_t,
          enable_if_t<
              is_complex<type_t<vector_t>>::value,
             int> = 0
             >
int eispack_hqr(
    bool want_T,
    bool want_Q,
    size_type<matrix_t> low,
    size_type<matrix_t> igh,
    matrix_t &A,
    vector_t &eigs,
    matrix_t &Q,
    real_type<type_t<matrix_t>> &norm )
```

My workflow

1. Write in a language I am comfortable with (C)
2. Segment the code (create more functions!)
3. Transcribe a function at a time along with white-box testing
4. Transcribe the main driver

Examples of calling code

```
//Call hqr
real_t norm = real_t(zero);
retCode = tlapack::eispack_hqr(true, true, ilo, igh, H, s, Q, norm);
```

Looking at the testing files, we don't need to do anything special in order to call our templated functions. In fact, it is very similar to how you would use an interface like CBLAS or LAPACKE.
It's possible to omit the namespace dereference in the function calls, but it is overall not too cumbersome to leave it this way and makes it much more clear where functions you are using come from.

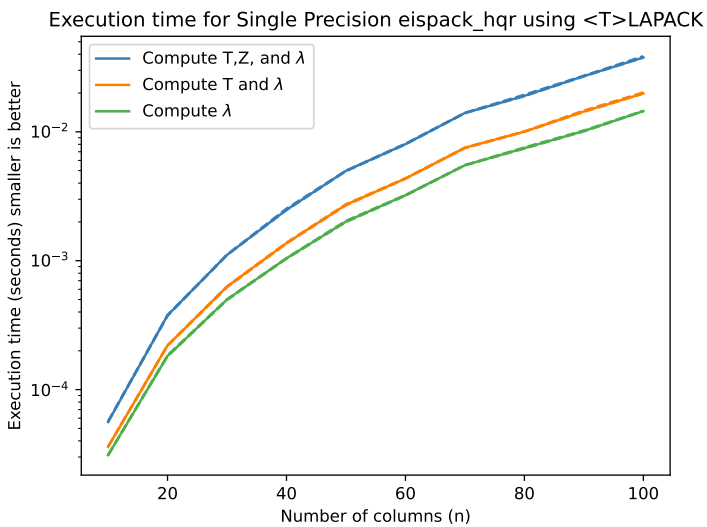
Testing our code

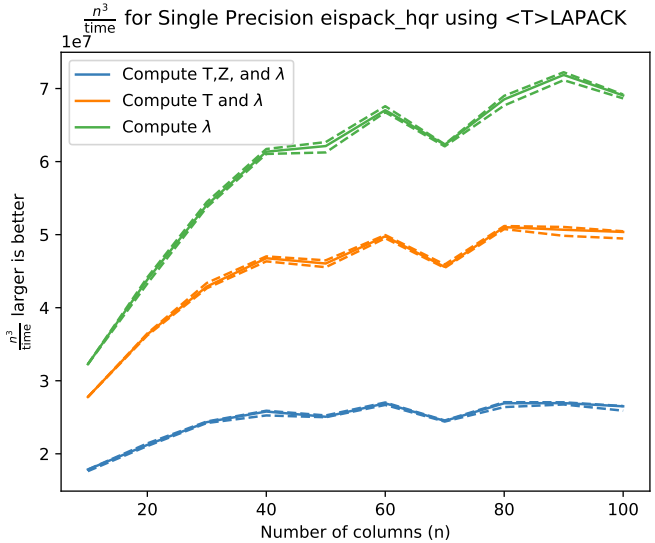
```
/*******************************************************************************************\
|* Single precision section                                                                *|
\*******************************************************************************************/
std::cout << "Single precision" << std::endl;
computeTimes<legacyMatrix<float,std::size_t,Layout::ColMajor>>(n);
/*******************************************************************************************\
|* Double precision section                                                                *|
\*******************************************************************************************/
std::cout << "Double precision" << std::endl;
computeTimes<legacyMatrix<double,std::size_t,Layout::ColMajor>>(n);
```

As we can see, when we template, the code that is called can be much simpler and if we are being given our data from another source, we don't need to implement any kind of checking to determine the correct call!

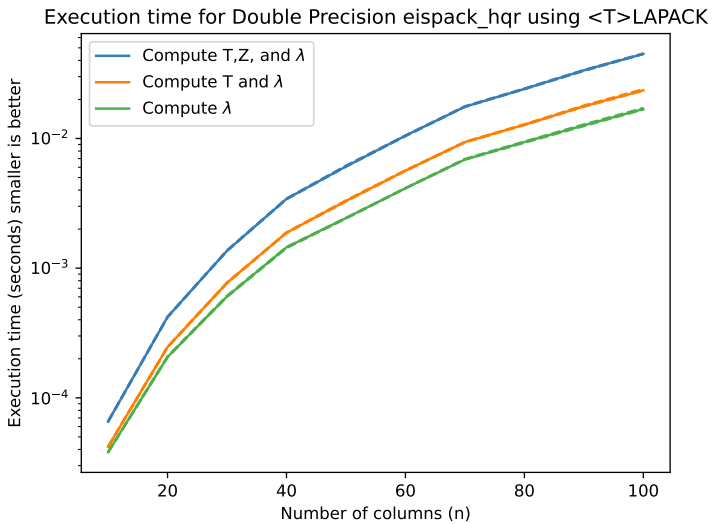
Timing of my project

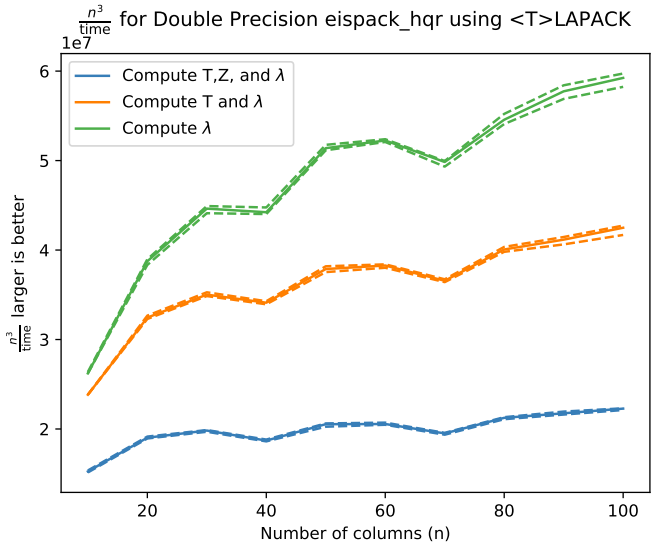
We first present the first two "obvious" options for our types, single and double precision





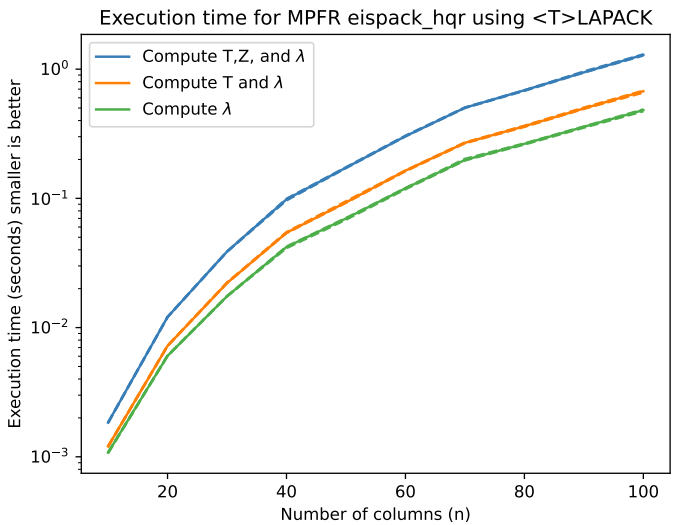
Timing of my project

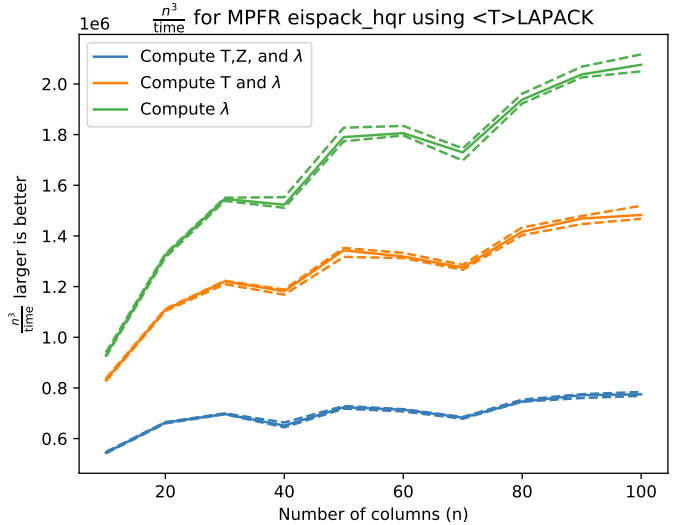




Timing of my project

Using multi precision was also just a drag and drop!





Level 0

Level 0

- easier for “compilers” to work with
- try to have more “versions” of algorithm and not necessarily one algorithm
- might offer interesting speedup for small sizes
- easier for humans to work with, (batched version, etc.)

Related to for example “User Guide for LDL, a concise sparse Cholesky package”, Tim Davis, version 3.3.2, Mar 22, 2024.

User Guide for LDL, a concise sparse Cholesky package - Tim Davis

```
void ldl_symbolic
(
    int n,              /* A and L are n-by-n, where n >= 0 */
    int Ap [ ],         /* input of size n+1, not modified */
    int Ai [ ],         /* input of size nz=Ap[n], not modified */
    int Lp [ ],         /* output of size n+1, not defined on input */
    int Parent [ ],     /* output of size n, not defined on input */
    int Lnz [ ],        /* output of size n, not defined on input */
    int Flag [ ],       /* workspace of size n, not defn. on input or output */
    int P [ ],          /* optional input of size n */
    int Pinv [ ]        /* optional output of size n (used if P is not NULL) */
)
{
    int i, k, p, kk, p2 ;
    if (P)
    {
        /* If P is present then compute Pinv, the inverse of P */
        for (k = 0 ; k < n ; k++)
        {
            Pinv [P [k]] = k ;
        }
    }
    for (k = 0 ; k < n ; k++)
    {
        /* L(k,:) pattern: all nodes reachable in etree from nz in A(0:k-1,k) */
        Parent [k] = -1 ;           /* parent of k is not yet known */
        Flag [k] = k ;              /* mark node k as visited */
        Lnz [k] = 0 ;               /* count of nonzeros in column k of L */
        kk = (P) ? (P [k]) : (k) ;  /* kth original, or permuted, column */
        p2 = Ap [kk+1] ;
        for (p = Ap [kk] ; p < p2 ; p++)
        {
            /* A (i,k) is nonzero (original or permuted A) */
            i = (Pinv) ? (Pinv [Ai [p]]) : (Ai [p]) ;
            if (i < k)
            {
                /* follow path from i to root of etree, stop at flagged node */
                for ( ; Flag [i] != k ; i = Parent [i])
                {
                    /* find parent of i if not yet determined */
                    if (Parent [i] == -1) Parent [i] = k ;
                    Lnz [i]++ ;                     /* L (k,i) is nonzero */
                    Flag [i] = k ;                  /* mark i as visited */
                }
            }
        }
    }
    /* construct Lp index array from Lnz column counts */
    Lp [0] = 0 ;
    for (k = 0 ; k < n ; k++)
    {
        Lp [k+1] = Lp [k] + Lnz [k] ;
    }
}
```

```
int ldl_numeric         /* returns n if successful, k if D (k,k) is zero */
(
    int n,              /* A and L are n-by-n, where n >= 0 */
    int Ap [ ],         /* input of size n+1, not modified */
    int Ai [ ],         /* input of size nz=Ap[n], not modified */
    double Ax [ ],      /* input of size nz=Ap[n], not modified */
    int Lp [ ],         /* input of size n+1, not modified */
    int Parent [ ],     /* input of size n, not modified */
    int Lnz [ ],        /* output of size n, not defn. on input */
    int Li [ ],         /* output of size lnz=Lp[n], not defined on input */
    double Lx [ ],      /* output of size lnz=Lp[n], not defined on input */
    double D [ ],       /* output of size n, not defined on input */
    double Y [ ],       /* workspace of size n, not defn. on input or output */
    int Pattern [ ],    /* workspace of size n, not defn. on input or output */
    int Flag [ ],       /* workspace of size n, not defn. on input or output */
    int P [ ],          /* optional input of size n */
    int Pinv [ ]        /* optional input of size n */
)
{

    double yi, l_ki ;
    int i, k, p, kk, p2, len, top ;
    for (k = 0 ; k < n ; k++)
    {
        /* compute nonzero Pattern of kth row of L, in topological order */
        Y [k] = 0.0 ;               /* Y(0:k) is now all zero */
        top = n ;                   /* stack for pattern is empty */
        Flag [k] = k ;              /* mark node k as visited */
        Lnz [k] = 0 ;               /* count of nonzeros in column k of L */
        kk = (P) ? (P [k]) : (k) ;  /* kth original, or permuted, column */
        p2 = Ap [kk+1] ;
        for (p = Ap [kk] ; p < p2 ; p++)
        {
            i = (Pinv) ? (Pinv [Ai [p]]) : (Ai [p]) ;   /* get A(i,k) */
            if (i <= k)
            {
                Y [i] += Ax [p] ;  /* scatter A(i,k) into Y (sum duplicates) */
                for (len = 0 ; Flag [i] != k ; i = Parent [i])
                {
                    Pattern [len++] = i ;   /* L(k,i) is nonzero */
                    Flag [i] = k ;          /* mark i as visited */
                }
                while (len > 0) Pattern [--top] = Pattern [--len] ;
            }
        }
        /* compute numerical values kth row of L (a sparse triangular solve) */
        D [k] = Y [k] ;             /* get D(k,k) and clear Y(k) */
        Y [k] = 0.0 ;
        for ( ; top < n ; top++)
        {
            i = Pattern [top] ;     /* Pattern [top:n-1] is pattern of L(:,k) */
            yi = Y [i] ;            /* get and clear Y(i) */
            Y [i] = 0.0 ;
            p2 = Lp [i] + Lnz [i] ;
            for (p = Lp [i] ; p < p2 ; p++)
            {
                Y [Li [p]] -= Lx [p] * yi ;
            }
            l_ki = yi / D [i] ;     /* the nonzero entry L(k,i) */
            D [k] -= l_ki * yi ;
            Li [p] = k ;            /* store L(k,i) in column form of L */
            Lx [p] = l_ki ;
            Lnz [i]++ ;             /* increment count of nonzeros in col i */
        }
        if (D [k] == 0.0) return (k) ;      /* failure, D(k,k) is zero */
    }
    return (n) ;        /* success, diagonal of D is all nonzero */
}
```

```
void ldl_lsolve
(
    int n,              /* L is n-by-n, where n >= 0 */
    double X [ ],       /* size n.  right-hand-side on input, soln. on output */
    int Lp [ ],         /* input of size n+1, not modified */
    int Li [ ],         /* input of size lnz=Lp[n], not modified */
    double Lx [ ]       /* input of size lnz=Lp[n], not modified */
)
{
    int j, p, p2 ;
    for (j = 0 ; j < n ; j++)
    {
        p2 = Lp [j+1] ;
        for (p = Lp [j] ; p < p2 ; p++)
        {
            X [Li [p]] -= Lx [p] * X [j] ;
        }
    }
}

void ldl_dsolve
(
    int n,              /* D is n-by-n, where n >= 0 */
    double X [ ],       /* size n.  right-hand-side on input, soln. on output */
    double D [ ]        /* input of size n, not modified */
)
{
    int j ;
    for (j = 0 ; j < n ; j++)
    {
        X [j] /= D [j] ;
    }
}

void ldl_ltsolve
(
    int n,              /* L is n-by-n, where n >= 0 */
    double X [ ],       /* size n.  right-hand-side on input, soln. on output */
    int Lp [ ],         /* input of size n+1, not modified */
    int Li [ ],         /* input of size lnz=Lp[n], not modified */
    double Lx [ ]       /* input of size lnz=Lp[n], not modified */
)
{
    int j, p, p2 ;
    for (j = n-1 ; j >= 0 ; j--)
    {
        p2 = Lp [j+1] ;
        for (p = Lp [j] ; p < p2 ; p++)
        {
            X [j] -= Lx [p] * X [Li [p]] ;
        }
    }
}
```

Level 0

gemm

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < p; k++)
      C[i][j] += A[i][k] * B[k][j];
```

syrk

```
for (i = 0; i < n; i++)
  for (k = 0; k < m; k++)
    for (j = 0; j <= i; j++)
      C[i][j] += A[i][k] * A[j][k];
```

lu

```
for (i = 0; i < n; i++) {
  for (j = 0; j <i; j++) {
     for (k = 0; k < j; k++) {
        A[i][j] -= A[i][k] * A[k][j];
     }
      A[i][j] /= A[j][j];
  }
  for (j = i; j < n; j++) {
     for (k = 0; k < i; k++) {
        A[i][j] -= A[i][k] * A[k][j];
     }
  }
}
```

apply_givens_rot_sequence

```
for (p = 0; p < k; p++) {
    for (j = 0; j < n - 1; j++) {
        for (i = 0; i < m; i++) {
            temp = C[j][p] * A[i][j] + S[j][p] * A[i][j + 1];
            A[i][j + 1] = -S[j][p] * A[i][j] + C[j][p] * A[i][j + 1];
            A[i][j] = temp;
        }
    }
}
```

cholesky

```
for (i = 0; i < n; i++) {
   for (j = 0; j < i; j++) {
      for (k = 0; k < j; k++) {
         A[i][j] -= A[i][k] * A[j][k];
      }
      A[i][j] /= A[j][j];
   }
   for (k = 0; k < i; k++) {
      A[i][i] -= A[i][k] * A[i][k];
   }
   A[i][i] = sqrt(A[i][i]);
}
```

Level 0

cgs – Classical Gram-Schmidt

```
for (j = 0; j < N; j++) {
 for (i = 0; i < j; i++) {
    R[i][j] = 0.0e+00;
    for (k = 0; k < M; k++)
       R[i][j] += A[k][i] * A[k][j];
 }
 for (i = 0; i < j; i++)
    for (k = 0; k < M; k++)
       A[k][j] -= A[k][i] * R[i][j];
 R[j][j] = 0.0e+00;
 for (k = 0; k < M; k++)
    R[j][j] += A[k][j] * A[k][j];
 R[j][j] = sqrt(R[j][j]);
 for (k = 0; k < M; k++)
    A[k][j] /= R[j][j];
}
```

a2v – geqr2 – Householder QR factorization

```
for(k = 0; k < N; k++){
 norma2 = 0.e+00;
 for(i = k+1; i < M; i++){
    norma2 += A[i][k] * A[i][k];
 }
 norma = sqrt( A[k][k] * A[k][k] + norma2 );

 A[k][k] = ( A[k][k] > 0 ) ? ( A[k][k] + norma ) : ( A[k][k] - norma ) ;

 tau[k] = 2.0 / ( 1.0 + norma2 / ( A[k][k] * A[k][k] ) ) ;

 for(i = k+1; i < M; i++){
    A[i][k] /= A[k][k];
 }
 A[k][k]= ( A[k][k] > 0 ) ? ( - norma ) : ( norma ) ;

 for(j = k+1; j < N; j++){
    tau[j] = A[k][j];
    for(i = k+1; i < M; i++){
          tau[j] += A[i][k] * A[i][j];
    }
    tau[j] = tau[k] * tau[j];
    A[k][j] = A[k][j] - tau[j];
    for(i = k+1; i < M; i++){
       A[i][j] = A[i][j] - A[i][k] * tau[j];
    }
 }
}
```

mgs – Modified Gram-Schmidt

```
for (j = 0; j < N; j++) {
  for (i = 0; i < j; i++) {
     R[i][j] = 0.0e+00;
     for (k = 0; k < M; k++)
        R[i][j] += A[k][i] * A[k][j];
     for (k = 0; k < M; k++)
        A[k][j] -= A[k][i] * R[i][j];
  }
  R[j][j] = 0.0e+00;
  for (k = 0; k < M; k++)
     R[j][j] += A[k][j] * A[k][j];
  R[j][j] = sqrt(R[j][j]);
  for (k = 0; k < M; k++)
     A[k][j] /= R[j][j];
 }
```

v2q – orqr2 – Construction of Q from Householder

```
for(k = N-1; k > -1; k--){
 for(j = k+1; j < N; j++){
    tau[j] = 0.e+00;
       for(i = k+1; i < M; i++){
          tau[j] += A[i][k] * A[i][j];
       }
 }
 for(j = k+1; j < N; j++){
    tau[j] *= tau[k];
 }
 A[k][k] = 1.0e+00 - tau[k];
 for(j = k+1; j < N; j++){
    A[k][j] = -tau[j];
 }
 for(j = k+1; j < N; j++){
    for(i = k+1; i < M; i++){
       A[i][j] -= A[i][k] * tau[j];
    }
 }
 for(i = k+1; i < M; i++){
    A[i][k] = - A[i][k] * tau[k];
 }
}
```

Level 0

gebd2 – reduction to bidiagonalization

```
for(k = 0; k < N; k++){
 norma2 = 0.e+00;
 for(i = k+1; i < M; i++){
    norma2 += A[i][k] * A[i][k];
 }
 norma = sqrt( A[k][k] * A[k][k] + norma2 );
 A[k][k] = ( A[k][k] > 0 ) ? ( A[k][k] + norma ) : ( A[k][k] - norma ) ;
 tauq[k] = 2.0 / ( 1.0 + norma2 / ( A[k][k] * A[k][k] ) ) ;
 for(i = k+1; i < M; i++){
    A[i][k] /= A[k][k];
 }
 A[k][k]= ( A[k][k] > 0 ) ? ( - norma ) : ( norma ) ;
 for(j = k+1; j < N; j++){
    ttmp = A[k][j];
    for(i = k+1; i < M; i++){
       ttmp += A[i][k] * A[i][j];
    }
    ttmp = tauq[k] * ttmp;
    A[k][j] = A[k][j] - ttmp;
    for(i = k+1; i < M; i++){
       A[i][j] = A[i][j] - A[i][k] * ttmp;
    }
 }
 norma2 = 0.e+00;
 for(j = k+2; j < N; j++){
    norma2 += A[k][j] * A[k][j];
 }
 norma = sqrt( A[k][k+1] * A[k][k+1] + norma2 );
 A[k][k+1] = ( A[k][k+1] > 0 ) ? ( A[k][k+1] + norma ) : ( A[k][k+1] - norma ) ;
 taup[k] = 2.0 / ( 1.0 + norma2 / ( A[k][k+1] * A[k][k+1] ) ) ;
 for(j = k+2; j < N; j++){
    A[k][j] /= A[k][k+1];
 }
 A[k][k+1]= ( A[k][k+1] > 0 ) ? ( - norma ) : ( norma ) ;
 for(i = k+1; i < M; i++){
    ttmp = A[i][k+1];
    for(j = k+2; j < N; j++){
       ttmp += A[i][j] * A[k][j];
    }
    ttmp = ttmp * taup[k];
    A[i][k+1] = A[i][k+1] - ttmp;
    for(j = k+2; j < N; j++){
       A[i][j] = A[i][j] - ttmp * A[k][j] ;
    }
 }
}
```

gehd2 – reduction to Hessenberg form

```
for ( j = 0; j < n-2; j++ ) {
   norma2 = 0.0;
   for ( i = j+2; i < n; i++ ) {
      norma2 += A[i][j] * A[i][j] ;
   }
   norma = sqrt ( A[j+1][j] * A[j+1][j] + norma2 ) ;
   A[j+1][j] = ( A[j+1][j] > 0 ) ? ( A[j+1][j] + norma ) : ( A[j+1][j] - norma ) ;
   tau = 2.0 / ( 1.0 + norma2 / ( A[j+1][j] * A[j+1][j] ) ) ;
   for ( i = j+2; i < n; i++ ) {
      A[i][j] /= A[j+1][j] ;
   }
   A[j+1][j] = ( A[j+1][j] > 0 ) ? ( - norma ) : ( norma ) ;
   for (i = j+1; i < n; i++) {
      tmp[i] = A[j+1][i] ;
      for (k = j+2; k < n; k++) {
         tmp[i] += A[k][j] * A[k][i];
      }
   }
   for (i = j+1; i < n; i++) {
      tmp[i] *= tau ;
   }
   for (i = j+1 ; i < n ; i++) {
      A[j+1][i] -= tmp[i] ;
   }
   for (i = j+2; i < n; i++) {
      for (k = j+1; k < n; k++) {
         A[i][k] -= A[i][j] * tmp[k];
      }
   }
   for (i = 0; i < n; i++) {
      tmp[i] = A[i][j+1];
      for (k = j+2; k < n; k++) {
         tmp[i] += A[i][k] * A[k][j];
      }
   }
   for (i = 0; i < n; i++) {
      tmp[i] *= tau ;
   }
   for (i = 0; i < n; i++) {
      A[i][j+1] -= tmp[i] ;
   }
   for (i = 0; i < n; i++) {
      for (k = j+2; k < n; k++) {
         A[i][k] -= tmp[i] * A[k][j] ;
      }
   }
}
```

Level 0

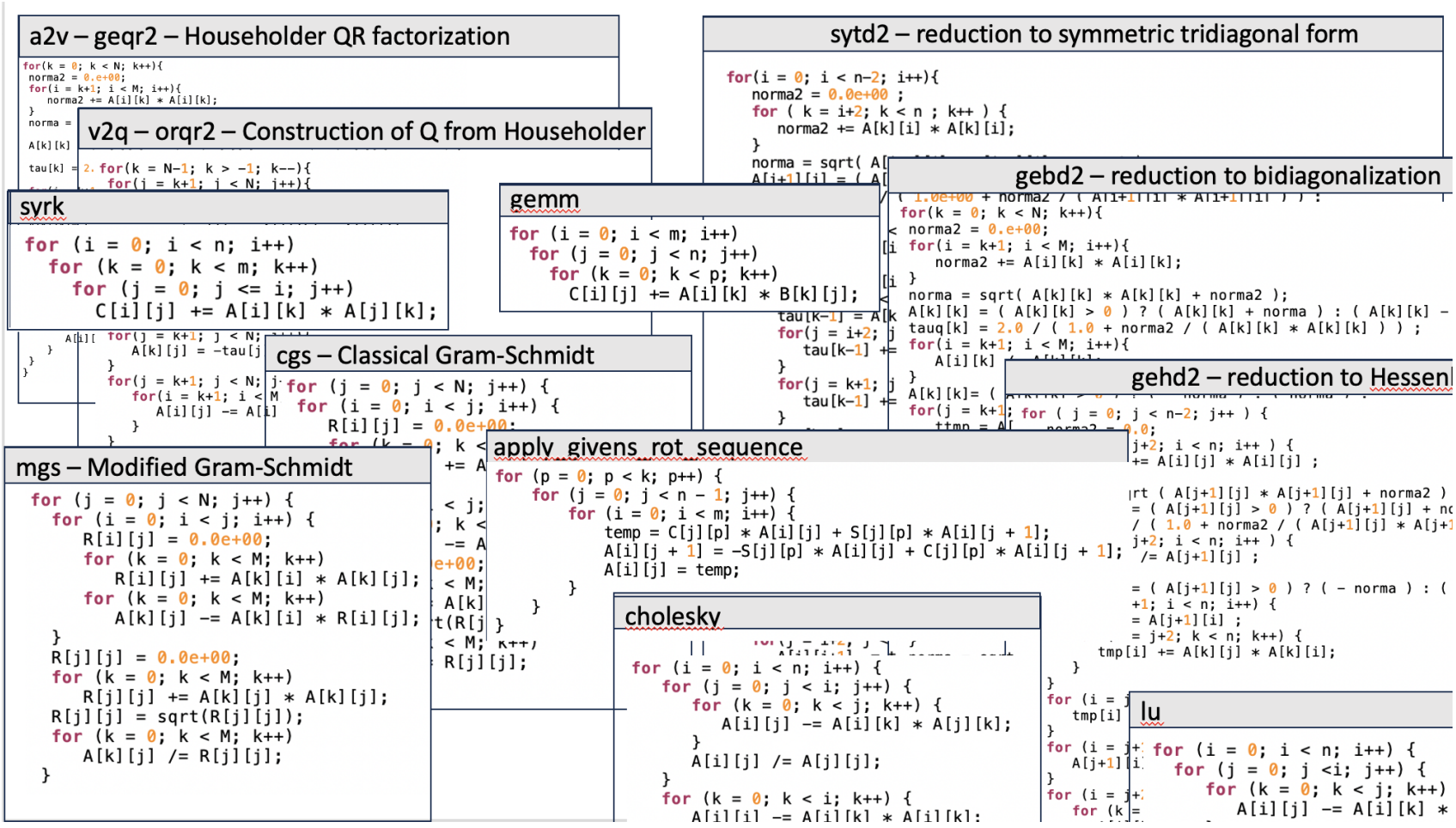
sytd2 – reduction to symmetric tridiagonal form

```
for(i = 0; i < n-2; i++){
   norma2 = 0.0e+00 ;
   for ( k = i+2; k < n ; k++ ) {
      norma2 += A[k][i] * A[k][i];
   }
   norma = sqrt( A[i+1][i] * A[i+1][i] + norma2 ) ;
   A[i+1][i] = ( A[i+1][i] > 0 ) ? (A[i+1][i] + norma) : (A[i+1][i] - norma) ;
   taui = 2.0e+00 / ( 1.0e+00 + norma2 / ( A[i+1][i] * A[i+1][i] ) ) ;

   for (k = i+2; k < n ; k++) {
      A[k][i] /= A[i+1][i] ;
   }
   A[i+1][i] = ( A[i+1][i] > 0.0e+00 ) ? ( - norma ) : ( norma ) ;
   for(k = i+1; k < n; k++){
      tau[k-1] = A[k][i+1];
      for(j = i+2; j <= k; j++){
         tau[k-1] += A[k][j] * A[j][i];
      }
      for(j = k+1; j < n; j++){
         tau[k-1] += A[j][k] * A[j][i];
      }
      tau[k-1] *= taui;
   }
   alpha = tau[i];
   for(k = i+2; k < n; k++){
      alpha += tau[k-1] * A[k][i];
   }
   alpha = -0.5e+00 * taui * alpha;
   tau[i] += alpha ;
   for(k = i+2; k < n; k++){
      tau[k-1] += alpha * A[k][i];
   }
   A[i+1][i+1] -= 2.0e+00 * tau[i] ;
   for(j = i+2; j < n; j++){
      A[j][i+1] -= tau[j-1] ;
      A[j][i+1] -= tau[i] * A[j][i];
      for(k = i+2; k <= j; k++){
         A[j][k] -= tau[j-1] * A[k][i];
         A[j][k] -= tau[k-1] * A[j][i];
      }
   }
   tau[i] = taui;
}
```

gghd2 – reduction to triangular Hessenberg form

```
for (j = 0; j < _PB_N-2; j++) {
  for (i = _PB_N-2; i > j; i--) {
    nrm = SQRT_FUN ( A[i][j] * A[i][j] + A[i+1][j] * A[i+1][j] );
    c = A[i][j] / nrm;
    s = A[i+1][j] / nrm;
    A[i][j] = nrm;
    A[i+1][j] = SCALAR_VAL(0.0);
    for (k = j+1; k < _PB_N; k++) {
      tmp = c * A[i][k] + s * A[i+1][k];
      A[i+1][k] = - s * A[i][k] + c * A[i+1][k];
      A[i][k] = tmp;
    }
    for (k = i; k < _PB_N; k++) {
      tmp = c * B[i][k] + s * B[i+1][k];
      B[i+1][k] = - s * B[i][k] + c * B[i+1][k];
      B[i][k] = tmp;
    }
    for (k = 0; k < _PB_N; k++) {
      tmp = c * Q[i][k] + s * Q[i+1][k];
      Q[i+1][k] = - s * Q[i][k] + c * Q[i+1][k];
      Q[i][k] = tmp;
    }
    nrm = SQRT_FUN ( B[i+1][i+1] * B[i+1][i+1] + B[i+1][i] * B[i+1][i] );
    c = B[i+1][i+1] / nrm;
    s = B[i+1][i] / nrm;
    B[i+1][i+1] = nrm;
    B[i+1][i] = SCALAR_VAL(0.0);
    for (k = 0; k <= i; k++) {
      tmp = c * B[k][i] - s * B[k][i+1];
      B[k][i+1] = s * B[k][i] + c * B[k][i+1];
      B[k][i] = tmp;
    }
    for (k = 0; k < _PB_N; k++) {
      tmp = c * A[k][i] - s * A[k][i+1];
      A[k][i+1] = s * A[k][i] + c * A[k][i+1];
      A[k][i] = tmp;
    }
    for (k = i-j; k < _PB_N; k++) {
      tmp = c * Z[k][i] - s * Z[k][i+1];
      Z[k][i+1] = s * Z[k][i] + c * Z[k][i+1];
      Z[k][i] = tmp;
    }
  }
}
```

Level 0


a2v – geqr2 – Householder QR factorization
v2q – orqr2 – Construction of Q from Householder
sytd2 – reduction to symmetric tridiagonal form
gebd2 – reduction to bidiagonalization
syrk
gemm
cgs – Classical Gram-Schmidt
gehd2 – reduction to Hessen
apply_givens_rot_sequence
mgs – Modified Gram-Schmidt
cholesky
lu

Level 0

A. Olivry, J. Langou, L-N Pouchet, P. Sadayappan, and F. Rastello. **Automated derivation of parametric data movement lower bounds for affine programs.** In the Proceedings of PLDI 2020: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, page 808–822, June 2020. (link)

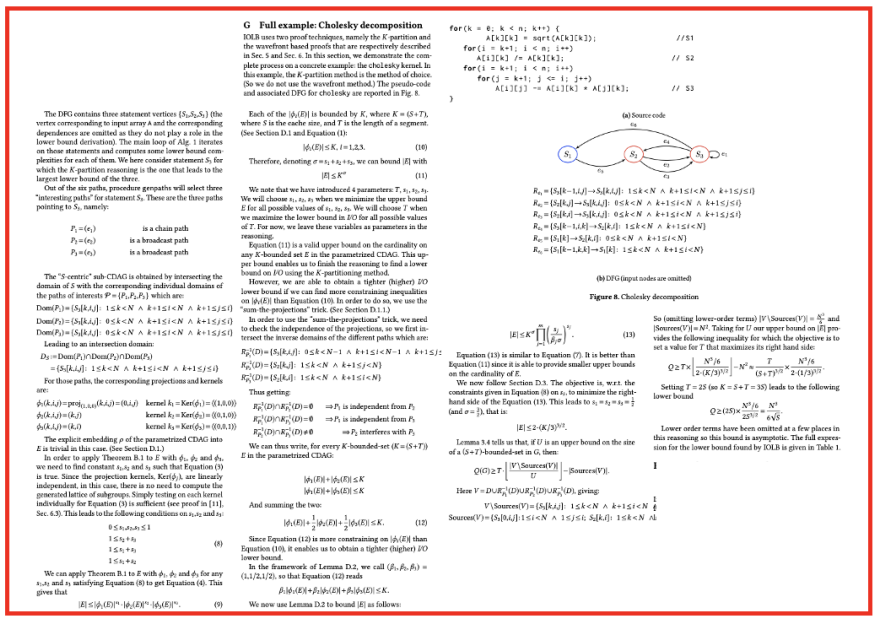
Output #1 is an IO lower bound

cholesky	$\max\left(\frac{1}{2}N(N+1), \frac{1}{6}\frac{1}{\sqrt{S}}(N-1)(N-2)(N-3)+\frac{1}{2\sqrt{2}}\frac{1}{S}(N-1)(N-2) -(N-2)(N-7)-4\sqrt{2}S\right)$	$\frac{1}{6}\frac{1}{\sqrt{S}}N^3$

input is an affine code

```
for (i = 0; i < _PB_N; i++) {
   for (j = 0; j < i; j++) {
      for (k = 0; k < j; k++) {
         A[i][j] -= A[i][k] * A[j][k];
      }
      A[i][j] /= A[j][j];
   }
   for (k = 0; k < i; k++) {
      A[i][i] -= A[i][k] * A[i][k];
   }
   A[i][i] = SQRT_FUN(A[i][i]);
}
```

cholesky code



Output #2 is a proof

Cholesky

Level 0

kernel	# input data	#ops	ratio	OI_{up}	OI_{manual}	ratio
2mm	$N_iN_k + N_kN_j$ $+N_jN_l + N_iN_l$	NiN_jN_k $+N_iN_jN_l$	–	$\sqrt{S}$	$\sqrt{S}$	1 ✓
3mm	$N_iN_k + N_kN_j$ $+N_jN_m + N_mN_l$	$NiN_jN_k + N_jN_lN_m$ $+N_iN_jN_l$	–	$\sqrt{S}$	$\sqrt{S}$	1 ✓
cholesky	$\frac{1}{2}N^2$	$\frac{1}{3}N^3$	$\frac{2}{3}N$	$2\sqrt{S}$	$\sqrt{S}$	2
correlation	MN	M^2N	M	$2\sqrt{S}$	$\sqrt{S}$	2
covariance	MN	M^2N	M	$2\sqrt{S}$	$\sqrt{S}$	2
doitgen	$N_pN_qN_r$	$2N_qn_rN_p^2$	$2N_p$	$\sqrt{S}$	$\sqrt{S}$	1 ✓
fdtd-2d	$3N_xN_y$	$11N_xN_yT$	$\frac{11}{3}T$	$22\sqrt{2}\sqrt{S}$	$\frac{11}{24}\sqrt{3}\sqrt{S}$	$\frac{48\sqrt{2}}{\sqrt{3}}$
floyd-warshall	N^2	$2N^3$	$2N$	$2\sqrt{S}$	$\sqrt{S}$	2
gemm	$N_iN_j + N_jN_k + N_iN_k$	$2N_iN_jN_k$	–	$\sqrt{S}$	$\sqrt{S}$	1 ✓
heat-3d	N^3	$30N^3T$	$30T$	$\frac{160}{3\sqrt[3]{3}}\sqrt[3]{S}$	$\frac{5}{2}\sqrt[3]{S}$	$\frac{64}{3\sqrt[3]{3}}$
jacobi-1d	N	$6NT$	$6T$	$24S$	$\frac{3}{2}S$	16
jacobi-2d	N^2	$10N^2T$	$10T$	$15\sqrt{3}\sqrt{S}$	$\frac{5}{4}\sqrt{S}$	$12\sqrt{3}$
lu	N^2	$\frac{2}{3}N^3$	$\frac{2}{3}N$	$\sqrt{S}$	$\sqrt{S}$	1 ✓
ludcmp	N^2	$\frac{2}{3}N^3$	$\frac{2}{3}N$	$\sqrt{S}$	$\sqrt{S}$	1 ✓
seidel-2d	N^2	$9N^2T$	$9T$	$\frac{27\sqrt{3}}{2}\sqrt{S}$	$\frac{9}{4}\sqrt{S}$	$6\sqrt{3}$
symm	$\frac{1}{2}M^2 + 2MN$	$2M^2N$	–	$\sqrt{S}$	$\sqrt{S}$	1 ✓
syr2k	$\frac{1}{2}N^2 + 2MN$	$2MN^2$	–	$2\sqrt{S}$	$\sqrt{S}$	2
syrk	$\frac{1}{2}N^2 + MN$	MN^2	–	$2\sqrt{S}$	$\sqrt{S}$	2
trmm	$\frac{1}{2}M^2 + MN$	M^2N	–	$\sqrt{S}$	$\sqrt{S}$	1 ✓
atax	MN	$4MN$	4	4	4	1 ✓
bicg	MN	$4MN$	4	4	4	1 ✓
deriche	HW	$32HW$	32	32	$\frac{16}{3}$	6
gemver	N^2	$10N^2$	10	10	5	2
gesummv	$2N^2$	$4N^2$	2	2	2	1 ✓
mvt	N^2	$4N^2$	4	4	4	1 ✓
trisolv	$\frac{1}{2}N^2$	N^2	2	2	2	1 ✓
adi	N^2	$30N^2T$	$30T$	30	5	6
durbin	N	$2N^2$	$2N$	4	$\frac{2}{3}$	6
gramschmidt	MN	$2MN^2$	$2N$	$2\sqrt{S}$	1	$2\sqrt{S}$
nussinov	$\frac{1}{2}N^2$	$\frac{1}{3}N^3$	$\frac{2}{3}N$	$2\sqrt{S}$	1	$2\sqrt{S}$

Integration and Interaction

LAPACK 3.11 (Nov 2022)

criterion for deflation in the QZ algorithm by @thijssteel

– New "strict criterion" and back to the normwise criterion for detecting infinite eigenvalues (as opposed to elementwise criterion) (PR#698). See also `https://arxiv.org/abs/2208.02057`.
Consider a pencil (H, T) in Hessenberg-triangular form:

$$(H,T) = \left(\begin{bmatrix} h_{11} & \dots & \dots & h_{1,n} \\ h_{21} & \ddots & & \vdots \\ & \ddots & \ddots & \vdots \\ 0 & & h_{n,n-1} & h_{nn} \end{bmatrix}, \begin{bmatrix} t_{11} & \dots & \dots & t_{1,n} \\ & \ddots & & \vdots \\ & & \ddots & \vdots \\ 0 & & & t_{nn} \end{bmatrix} \right).$$

- normwise (original): $|h_{i,i-1}| \leq u\|H\|_F$
- elementwise (since 3.9.1): $|h_{i,i-1}| \leq u(|h_{i-1,i-1}| + |h_{i,i}|)$
- strict (since 3.11.0): $|h_{i,i-1}| \leq u(|h_{i-1,i-1}| + |h_{i,i}|)$ and $|h_{i-1,i}t_{i,i} - h_{i,i}t_{i-1,i}||h_{i,i-1}| \leq u|h_{i,i}||h_{i-1,i-1}t_{i,i} - h_{i,i}t_{i-1,i-1}|$

LAPACK 3.11 (Nov 2022)

criterion for deflation in the QZ algorithm by @thijssteel

In single precision (32-bit), consider the pencil:

$$(H, T) = \left(\begin{bmatrix} 1 & c & 0 \\ \eta & (1+d) & 1 \\ 0 & \eta & (1+2d)c^{-1} \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & c^{-1} \end{bmatrix} \right),$$

with $\eta = 1.1 \cdot 10^{-8}$, $c = 1.1 \cdot 10^{5}$ and $d = 10^{-2}$.

- The eigenvalues of this pencil are approximately:
  0.95371503, 1.0261424, and 1.0501426.

LAPACK 3.11 (Nov 2022)

criterion for deflation in the QZ algorithm by @thijssteel

In single precision (32-bit), consider the pencil:

$$(H, T) = \left(\begin{bmatrix} 1 & c & 0 \\ \eta & (1+d) & 1 \\ 0 & \eta & (1+2d)c^{-1} \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & c^{-1} \end{bmatrix} \right),$$

with $\eta = 1.1 \cdot 10^{-8}$, $c = 1.1 \cdot 10^{5}$ and $d = 10^{-2}$.

- The eigenvalues of this pencil are approximately:
  0.95371503, 1.0261424, and 1.0501426.
- Criteria applied only to H (the normwise criterion (3.9 and before), the elementwise criterion (3.10) or even the criterion by Ahues and Tisseur 1997) would consider $h_{3,2}$ to be small enough in IEEE single precision. The returned eigenvalues are then
  1., 1.02 and 1.04,
  resulting in less than 2 digits of accuracy.

LAPACK 3.11 (Nov 2022)

criterion for deflation in the QZ algorithm by @thijssteel

In single precision (32-bit), consider the pencil:

$$(H, T) = \left(\begin{bmatrix} 1 & c & 0 \\ \eta & (1+d) & 1 \\ 0 & \eta & (1+2d)c^{-1} \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & c^{-1} \end{bmatrix} \right),$$

with $\eta = 1.1 \cdot 10^{-8}$, $c = 1.1 \cdot 10^{5}$ and $d = 10^{-2}$.

• The eigenvalues of this pencil are approximately:

0.95371503, 1.0261424, and 1.0501426.

• Criteria applied only to H (the normwise criterion (3.9 and before), the elementwise criterion (3.10) or even the criterion by Ahues and Tisseur 1997) would consider $h_{3,2}$ to be small enough in IEEE single precision. The returned eigenvalues are then

1., 1.02 and 1.04,

resulting in less than 2 digits of accuracy.

• When using the strict criterion (3.11), the QZ algorithm performs 2 more iterations, resulting in smaller subdiagonal entries. The returned eigenvalues are:

0.95371503, 1.0261424, and 1.0501425,

which are almost accurate to the working precision.

LAPACK 3.11 (Nov 2022)

level-3 BLAS solvers for the triangular system and triangular Sylvester equation by @angsch

The triangular Sylvester equation has been recognized to be prone to overflow. For that purpose, *TRSYL utilizes a scaling factor to represent the solution as and solve the scaled equation . Due to the scaling factor, there is some flexibility in the representation of the solution. The proposed level-3 BLAS version, *TRSYL3, computes the scaling factors based on the upper bounds of blocks to enable level-3 BLAS. The scaling is typically slightly more aggressive so that an alternatively scaled final solution is computed. This is no problem as long as the scaling factor does not get flushed to zero (PR#651). The same upper bound calculation was used to write the level-3 BLAS solver for the triangular system, *LATRS3.

LAPACK 3.11 (Nov 2022)

level-3 BLAS solvers for the triangular system and triangular Sylvester equation by @angsch

Carl Christian Kjelgaard Mikkelsen, Angelika Beatrix Schwarz and Lars Karlsson. Parallel robust solution of triangular linear systems. In Concurrency and Computation: Practice and Experience, e5064, Wiley Online Library, 2018.

Dimensions		Good Systems						Bad Systems			
m	n	DTRS{V,M}		Kiya		DLATRS		Kiya		DLATRS	
10000	1	0.00572	20	0.0183	5.6	0.0791	1.3	0.0183	5.3	0.0780	1.3
20000	1	0.0175	23.5	0.0494	8.2	0.328	1.2	0.0496	8.2	0.327	1.2
40000	1	0.0625	26.7	0.143	11.2	1.47	1.1	0.158	10.1	1.47	1.1
10000	4000	0.510	784	0.971	412	23.9	16.7	0.983	407	23.9	16.7
20000	1000	0.483	828	0.932	429	53.6	7.46	0.932	429	57.3	6.98
40000	250	0.665	601	1.04	385	226	1.8	1.05	381	229	1.7

Comparison of different solvers. Kiya is *LATRS3 introduced in 3.11. Timings are given in seconds with three significant digits and are followed by the performance given in GFLOPS/s. Timings of DLATRS for $n > 1$ right hand sides are extrapolated from measurements with 112 right-hand sides

LAPACK 3.11 (Nov 2022)

Givens rotations generated with less accumulation errors by @weslleyspereira

New algorithms for computing Givens rotations in complex arithmetic that reduce the accumulation errors for computing each of the outputs, c, s, r. The new algorithms are, on average, more accurate than both the algorithms from LAPACK 3.9.1 and LAPACK 3.10.0 (PR#631). See also `https://arxiv.org/abs/2211.04010`.

Motivated by a bug report about the algorithm in LAPACK 3.10 (See Issue #629). The author reported that the new Givens rotations may have lower accuracy than the ones that were in LAPACK up to release 3.9. This could be easily verified by noticing that, after applying several rotations to a unitary 2×2 matrix, the departure from unitaricity was larger (in average) for 3.10 than 3.9. We performed additional experiments where we couldn't find the referred problem nor verify that the algorithm in LAPACK 3.10 was less accurate.

LAPACK 3.11 (Nov 2022)

Givens rotations generated with less accumulation errors by @weslleyspereira

Issue #629 – In the test 5000 of plane rotations are applied to the vector of length 2 with the initial 2-norm of 10.

LAPACK 3.10

```
-bash-4.2$ ./a.out
...
The 2-norm of the first vector =
0.10001301E+02
...
Max of all errors printed above
0.13008118E-02
```

LAPACK 3.9

```
-bash-4.2$ ./a.out
...
The 2-norm of the first vector =
0.99999943E+01
..
Error abs(norm of the first column - initial norm )
0.57220459E-05
...
Max of all errors printed above
0.57220459E-05
```

LAPACK 3.11 (Nov 2022)

Givens rotations generated with less accumulation errors by @weslleyspereira

We run each code several (10^6) times with random input data.

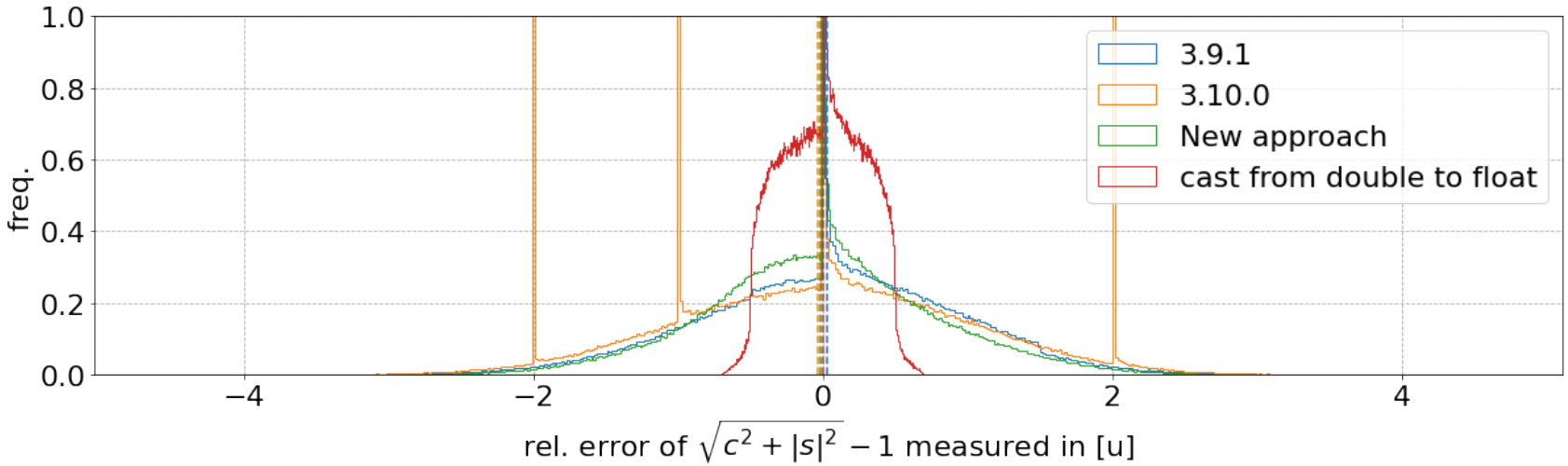


Figure: Histogram of the error in the singular values, $Err := \sqrt{c^2 + |s|^2} - 1$, for each algorithm measured in unit roundoff.

LAPACK 3.11 (Nov 2022)

Givens rotations generated with less accumulation errors by @weslleyspereira

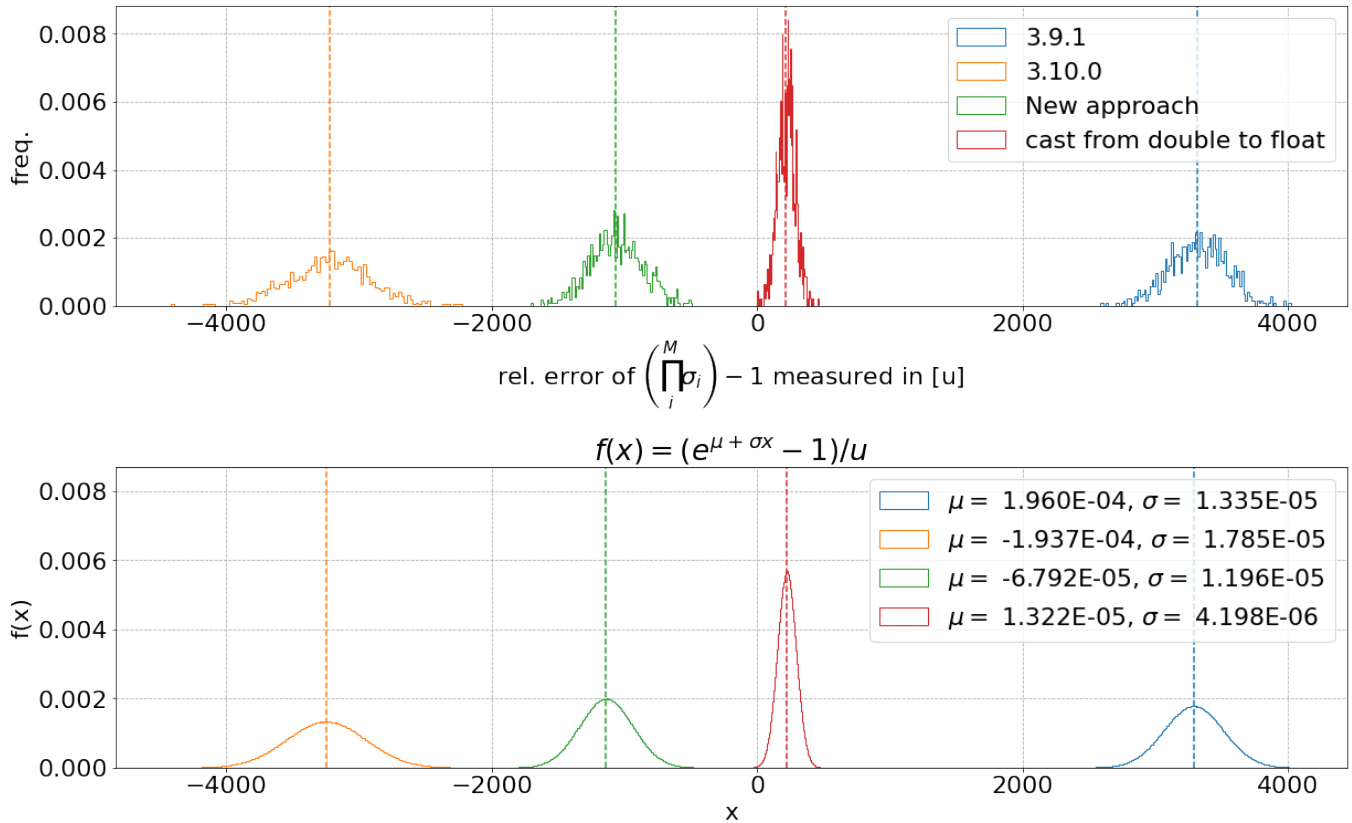


Figure: Top: Histogram of $\left(\prod_{i=1}^{M} \sigma_i\right) - 1$, $M = 10^5$, for each algorithm measured in unit roundoff. Repeat $N = 10^3$ times. Bottom: Estimates from formulae.

– The new algorithms, *GELST, are similar to *GELS. *GELST avoids computing triangular blocks twice as in *GELS, which means *GELST runs faster (PR#739).

LAPACK 3.11 (Nov 2022)

Improvements on the building system and the Continuous Testing framework

- Regression test for illegal modification of Y in *GEMV by @matcross in PR#622
- Test the Fortran intrinsic ABS and complex divisions and report failures during build by @weslleyspereira in PR#623
- Appveyor is working in the Continuous Testing once more by @martin-frbg in PR#627
- More tests and fixes for type mismatches by @angsch in PR#675
- Solve build issues with IBM XLF by @friedc in PR#677
- Explicit type casts to INT by @angsch in PR#684
- Fix an out-of-bounds write in the tests by @angsch in PR#685
- Adding support for NAG Fortran compiler (nagfor) by @ACSimon33 in PR#686
- Fix time check flag propagation and default value by @ACSimon33 in PR#696
- Fixed format warnings in 64 bit integer builds by @ACSimon33 in PR#700
- Explicit type casts and more rigorous CI by @weslleyspereira in PR#703
- Fix type mismatches in function calls of testing code by @mjacobse in PR#706
- Fix lower triangular band matrix generation by @angsch in PR#728

LAPACK 3.11 (Nov 2022)

Other improvements

• Improves the computation of norms in S,DORBDB6 and C,ZUNBDB6 by @christoph-conrads in PR#647

• Uses a Newton step to compute the initial guess for ETA in S,DLAED4 to reduce the number of iterations by @weslleyspereira, suggestion from Ren-Cang Li, in PR#655

• Return immediately when scaling with one by @angsch in PR#674

• Rearrange the application of the Householder reflectors in *LAQR5, which lowers the instruction count when FMA is available by @angsch in PR#681

• Add C,ZROTG, CS,ZDROT, S,DCABS1 to CBLAS by @angsch in PR#721

• Add *LANGB to LAPACKE by @ACSimon33 in PR#725

• NaN check for trapezoidal matrices on LAPACKE by @ACSimon33 in PR#738 and PR#742

LAPACK 3.11 (Nov 2022)

Uses a Newton step to compute the initial guess for ETA in S,DLAED4 to reduce the number of iterations by @weslleyspereira, suggestion from Ren-Cang Li

• Uses a Newton step to compute the initial guess for ETA in S,DLAED4 to reduce the number of iterations by @weslleyspereira in PR#655

line 333 in xLAED4 in 3.10

```
ETA = DLTUB - TAU
```

line 336 in xLAED4 in 3.11

```
ETA = -W / ( DPSI+DPHI )
```

See as well Issue#99

LAPACK 3.11 (Nov 2022)

Bug fixes

- Fix LAPACKE_*tpmqrt_work for row-major matrices by @weslleyspereira in PR#540
- Fix leading dimension check in LAPACKE's *geesv[x] and *gges[x] by @angsch in PR#655
- Fix the left-looking variant of GEQRF by @weslleyspereira in PR#690
- Fix workspace query for *SYEVD and *HEEVD routines by @neil-lindquist in PR#691
- Fix behavior of SCALE in *LATBS and *LATRS, and avoids NaN generation if entries in CNORM exceed the overflow threshold by @angsch in PR#712

LAPACK 3.11 (Nov 2022)

New Contributors

- @PhillHS made their first contribution in PR#601
- @mtowara made their first contribution in PR#663
- @dbakshee made their first contribution in PR#671
- @ACSimon33 made their first contribution in PR#686
- @friedc made their first contribution in PR#677
- @neil-lindquist made their first contribution in PR#691
- @ivan-pi made their first contribution in PR#699
- @mjacobse made their first contribution in PR#702
- @MonicaLiu0311 made their first contribution in PR#720
- @eltociear made their first contribution in PR#735

LAPACK 3.12 (June/July 2023?)

• xGEDMD and xGEDMDQ – Nonsymmetric DMD by Zlatko Drmač and Daniel Bielich and AIMdyn Inc. Santa Barbara, CA – PR#736

• xGEQP3RK – Truncated QR With Column Pivoting by Igor Kozachenko and Jim Demmel.
Igor's branch

• xRSCL – Reciprocal Scaling by Weslley Pereira
PR#749

• Better Organization for the Documentation by Mark Gates
PR#802

• (maybe) IxAMAX – exception handling "compliant" MAX search by Weslley Pereira with input from Jim Demmel, Greg Henry, Jason Riedy, etc.

LAPACK updates

J. Demmel, J. Dongarra, M. Gates, G. Henry, J. Langou, X. Li, P. Luszczek, W. Pereira, Weslley, J. Riedy, and C. Rubio-Gonzalez. Proposed Consistent Exception Handling for the BLAS and LAPACK. In Sixth International Workshop on Software Correctness for HPC Applications (Correctness 2022), a workshop of ACM/IEEE SC 2022 Conference (SC'22), Dallas, TX, USA, November 13-18, 2022 10.1109/Correctness56720.2022.00006

- rcond is 0 for NaN matrix in lapack-3.11.

`https://github.com/Reference-LAPACK/lapack/issues/763`
`https://savannah.gnu.org/bugs/?63384`

- proper scaling function (reciprocal scaling)
- NaN propagation (IxAMAX)

Related to BLAS:

- testBLAS to test and find inconsistencies in the use of the BLAS standard. Testing MKL, BLIS, OpenBLAS, and also BLAS++.

NaN propagation in BLAS

```
alpha = 1
beta = 0
call GEMM for C := alpha * A * B + beta * C
```

Do we want the NaNs in "input C" to propagate to "output C"?

testBLAS – compiler part

```
./test_zcomplexabs 2> test_zcomplexabs.err
 # All tests pass for ABS(a+b*I)
./test_zcomplexdiv 2> test_zcomplexdiv.err
 !! Some (x+x*I)/(x+x*I) differ from 1
 !! Some (x+x*I)/(x-x*I) differ from I
 #        12602  tests out of        12606  pass for complex division,
 # Please check the failed divisions in [stderr]
./test_zcomplexmult 2> test_zcomplexmult.err
 # All tests pass for complex multiplication.
./test_zminMax 2> test_zminMax.err
[i8] MIN(  NaN,   0.) =    0.
[i8] MAX(  NaN,   0.) =    0.
 #          14  tests out of          16  pass for intrinsic MIN and
```

Continuous Testing for ⟨T⟩LAPACK

testBLAS: test corner cases and exception handling in the BLAS.

- Started in Jun 20, 2021.
- Used in Demmel, J. et.al. (2022). Proposed Consistent Exception Handling for the BLAS and LAPACK

BLAS++ and LAPACK++ testers for comparing with legacy code.

- Tests compare output and execution time with LAPACKE.

Tests inside the ⟨T⟩LAPACK project.

- Test different data types, layouts and matrix classes, e.g., legacyMatrix, Eigen::Matrix and mdspan.

Impact on other software

Related to Eigen:

- Commented on bug https://gitlab.com/libeigen/eigen/-/issues/408
- GCC complex was incompatible to Eigen::half. We proposed a PR that, we believe, is being evaluated: https://github.com/gcc-mirror/gcc/pull/8.

Related to StarPU:

- StarPU does not have the means to declare static constant codelets. https://github.com/starpu-runtime/starpu/pull/20.
- Bug on one of their examples: https://github.com/starpu-runtime/starpu/issues/18.
- Bug in cuSOLVER 11: https://github.com/starpu-runtime/starpu/issues/12. (Still need to report it)
- Makes StarPU compatible with CUDA 12: https://github.com/starpu-runtime/starpu/pull/10
- Task on elements: This is something outside StarPU in our opinion. They provide the means to do that. But they do not define a matrix class.

Related to SLATE, BLAS++, LAPACK++, and LAPACK:

- Test suite for Inf and NaN propagation and corner cases that works on both ⟨T⟩LAPACK and BLAS++: https://github.com/tlapack/testBLAS.
- Failures in the compilation of ⟨T⟩LAPACK with SLATE helps improving the latter, e.g., https://bitbucket.org/icl/lapackpp/pull-requests/14, https://github.com/icl-utk-edu/blaspp/pull/11, https://bitbucket.org/icl/blaspp/pull-requests/38.
- Reciprocal scaling: https://github.com/Reference-LAPACK/lapack/pull/749

Related to BLAS: • testBLAS to test and find inconsistencies in the use of the BLAS standard: https://arxiv.org/abs/2207.09281. Testing MKL, BLIS, OpenBLAS, and also BLAS++.

Happening Now

- Thijs Steel working on the symmetric eigenvalue solver. See `github.com/tlapack/tlapack/pulls`.
- Julien Langou, Johnathan Rhyne, Thijs Steel on SIAM LA'24. See `siam.org/conferences/cm/program/minitutorials/la24-minitutorials`
- Strategies for more sustainable software at NREL, running ⟨T⟩LAPACK in low precision. See `linkedin.com/jobs/search/?currentJobId=3826121686`. (Graduate Summer Internship)

Mixed precision in ⟨T⟩LAPACK

New data types come as plugins, e.g.,

- `float16`, `gnuquad`, `mpreal`.

Steps to add a new type:

1. Create a plugin file based on other plugins and on `concepts.hpp`.

It is actually just one step! But you can do better:

2. Add tests for the new type.
3. Add an example of usage.
4. Maybe, add overload a routine for better use with that type.

See ongoing addition of `bfloat16` at `https://github.com/tlapack/tlapack/pull/382`.

Mixed precision in ⟨T⟩LAPACK

Functions in ⟨T⟩LAPACK work with matrices of different types. They also handle mixed precision. How?

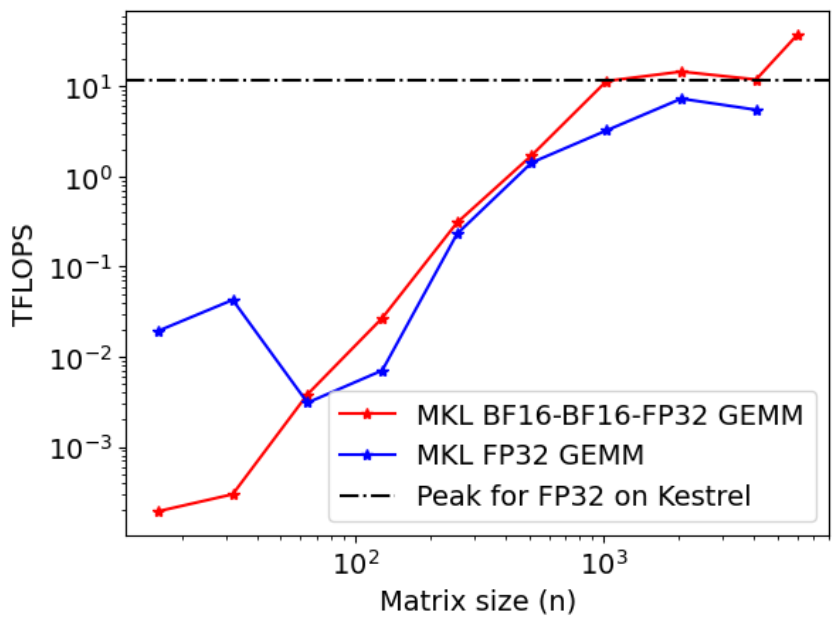
- `matrixA_t` and `matrixB_t` have entry types `TA` and `TB`, resp.
- Operations between `A(i,j)` and `B(k,l)` may have inputs in different precision.
- Any accumulator have the type common type between `TA` and `TB`, e.g.,
  `double` for `pair<float,double>`,
  `complex<double>` for `pair<complex<float>,double>`.

What if I want to operate in a different precision?

Mixed precision in ⟨T⟩LAPACK : An ongoing experiment

Intel AMX [1]:

- TMUL GEMM operates in mixed precision.
- `cblas_gemm_bf16bf16f32()` available in MKL (2023-2024).
- Less memory + Less time (up to 4x) + Lower accuracy.



Mixed precision in ⟨T⟩LAPACK : TRMM blocked (and mixed)

```
// trmm's interface:
void trmm_blocked_mixed(..., const matrixA_t& A, matrixB_t&
    B, work_t& work)
```

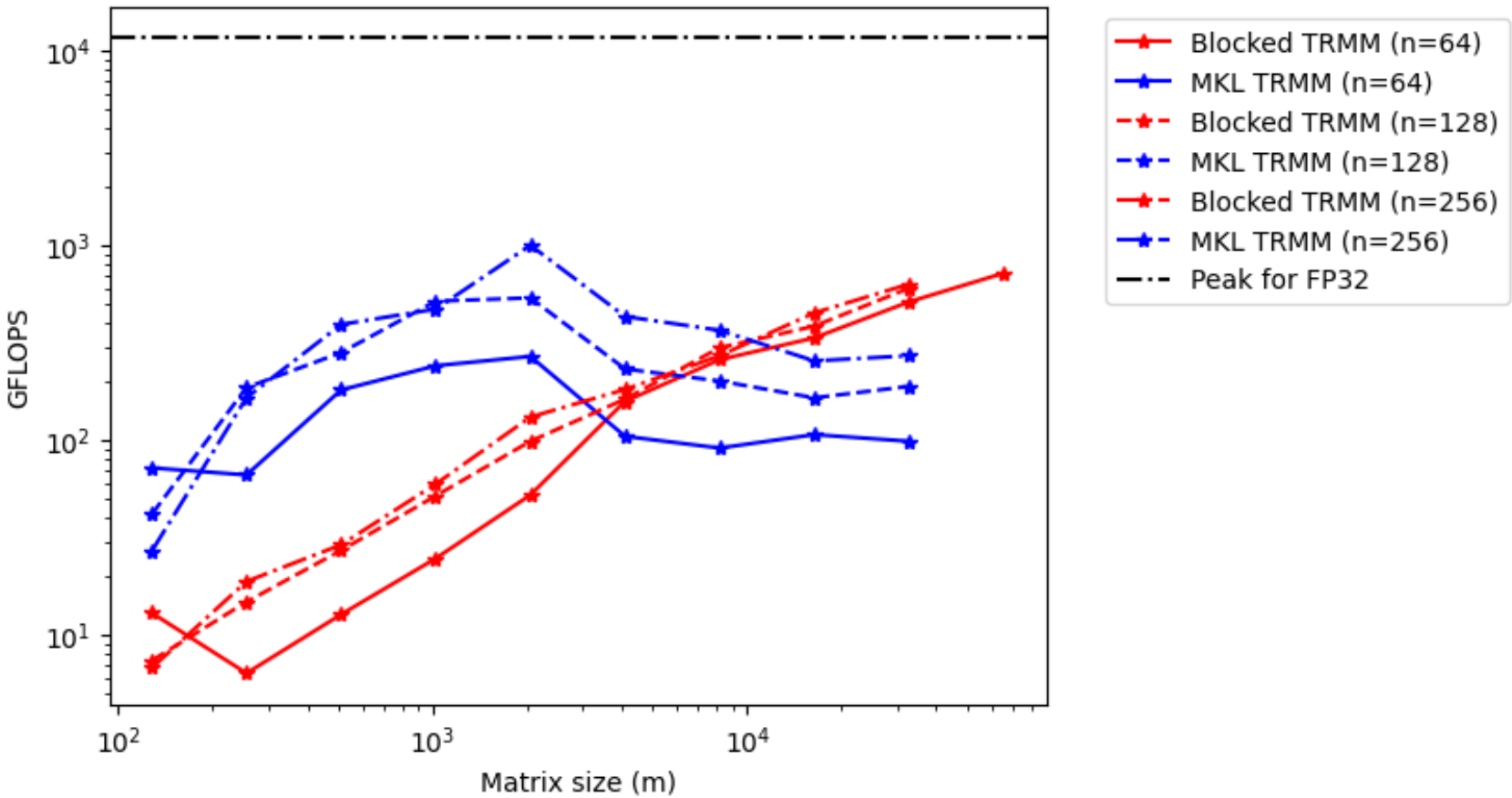
- matrixA_t and work_t in low precision (bfloat16).
- matrixB_t in high precision (float).

```
// Inside trmm's iteration:
lacpy(GENERAL, Bi, BiLowPrecision);
gemm(NO_TRANS, NO_TRANS, alpha, A0i, BiLowPrecision,
     real_t(1), B0);
trmm(side, uplo, trans, diag, alpha, Aii, Bi);
```

- lacpy() does the type cast.
- gemm() has an overload to cblas_gemm_bf16bf16f32().
- trmm() in mixed precision in ⟨T⟩LAPACK , native C++.

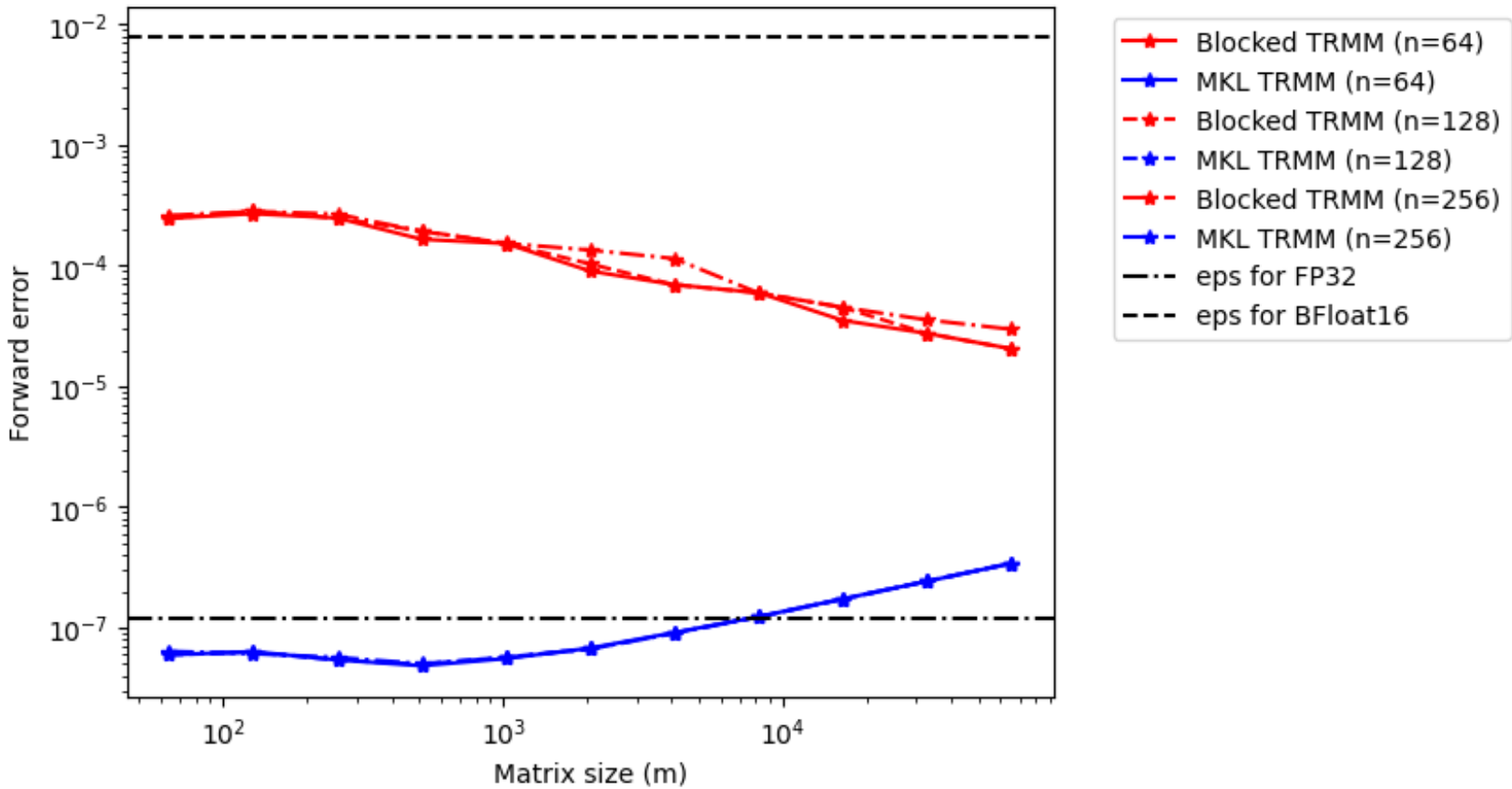
Mixed precision in ⟨T⟩LAPACK : TRMM blocked (and mixed)

- Random double-precision matrices with values in [0,1).
- $n_B = 110$ for blocked TRMM (Not necessarily optimal).



Mixed precision in ⟨T⟩LAPACK : TRMM blocked (and mixed)

- Error: $\|B_{double} - \tilde{B}\|_1 / \|B_{double}\|_1$.



Thank you!

Questions?

Weslley.daSilvaPereira@nrel.gov

Github repo:

`https://github.com/tlapack/tlapack`

This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

testing fp8 linear algebra in ⟨T⟩LAPACK

Sudhanva Kulkarni

adding fp8 in $\langle T \rangle LAPACK$

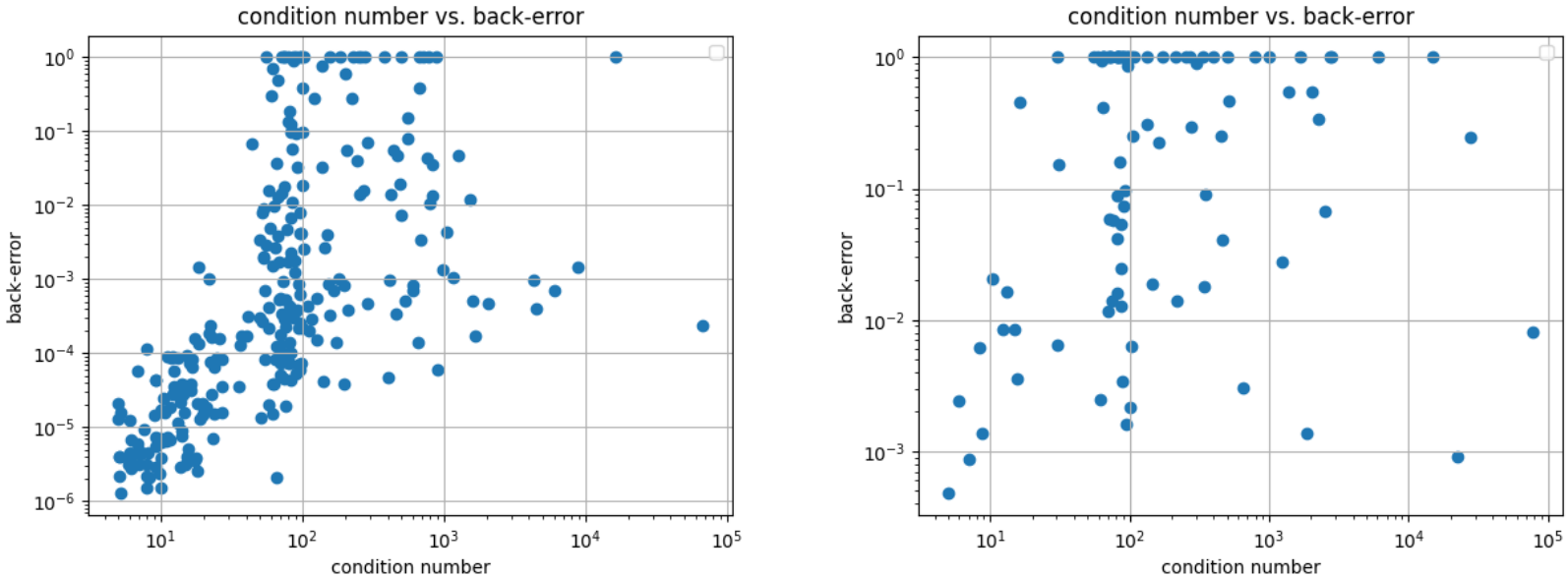
- IEEE P3109 format (8-bit floats) currently in development. Software implementation easy to use in $\langle T \rangle LAPACK$.
- Constraints on new fp types enforced by the use of "Scalar" concept in concepts.hpp .
- arithmetic for fp8 simulated by using higher precision datatypes like fp32 (operator+, operator-, etc). Can include stochastic rounding by changing the fp8 constructor. Generates random bitstring with Mersenne Twister PRNG (std::mt19937)
- Simulate mixed precision BLAS by accumulating in fp32/fp16/bfloat vector and casting down

LAPACK in fp8 - getrf

- Example routine - getrf for PLU decomposition
- Modifications made to reduce error include mixed precision BLAS/accumulation in fp32, equilibration in fp32, LU based iterative refinement.
- Experiments seem to convey that fp8 works well only for extremely well conditioned small matrices. Could use GMRES based IR for stronger refinement.
- Simple error analysis using Doolittle form gives element-wise worst case error bound $\sim |\Delta A| \leq ((1+\epsilon_8)^2 - 1)|L||U|$ for LU on small matrices.
- need to account for underflow due to large underflow thresholds for fp8 types!

```
0x00 = 0_00000_00 = +0b0.00*2^-15 = 0
0x01 = 0_00000_01 = +0b0.01*2^-15 = 7.62939e-06
0x02 = 0_00000_10 = +0b0.10*2^-15 = 1.52588e-05
0x03 = 0_00000_11 = +0b0.11*2^-15 = 2.28882e-05
0x04 = 0_00001_00 = +0b1.00*2^-15 = 3.05176e-05
0x05 = 0_00001_01 = +0b1.01*2^-15 = 3.8147e-05
0x06 = 0_00001_10 = +0b1.10*2^-15 = 4.57764e-05
0x07 = 0_00001_11 = +0b1.11*2^-15 = 5.34058e-05
0x08 = 0_00010_00 = +0b1.00*2^-14 = 6.10352e-05
0x09 = 0_00010_01 = +0b1.01*2^-14 = 7.62939e-05
0x0a = 0_00010_10 = +0b1.10*2^-14 = 9.15527e-05
0x0b = 0_00010_11 = +0b1.11*2^-14 = 0.000106812
0x0c = 0_00011_00 = +0b1.00*2^-13 = 0.00012207
0x0d = 0_00011_01 = +0b1.01*2^-13 = 0.000152588
0x0e = 0_00011_10 = +0b1.10*2^-13 = 0.000183105
0x0f = 0_00011_11 = +0b1.11*2^-13 = 0.000213623

0x10 = 0_00100_00 = +0b1.00*2^-12 = 0.000244141
0x11 = 0_00100_01 = +0b1.01*2^-12 = 0.000305176
0x12 = 0_00100_10 = +0b1.10*2^-12 = 0.000366211
0x13 = 0_00100_11 = +0b1.11*2^-12 = 0.000427246
0x14 = 0_00101_00 = +0b1.00*2^-11 = 0.000488281
0x15 = 0_00101_01 = +0b1.01*2^-11 = 0.000610352
0x16 = 0_00101_10 = +0b1.10*2^-11 = 0.000732422
0x17 = 0_00101_11 = +0b1.11*2^-11 = 0.000854492
0x18 = 0_00110_00 = +0b1.00*2^-10 = 0.000976562
0x19 = 0_00110_01 = +0b1.01*2^-10 = 0.0012207
0x1a = 0_00110_10 = +0b1.10*2^-10 = 0.00146484
0x1b = 0_00110_11 = +0b1.11*2^-10 = 0.00170898
0x1c = 0_00111_00 = +0b1.00*2^-9 = 0.00195312
0x1d = 0_00111_01 = +0b1.01*2^-9 = 0.00244141
0x1e = 0_00111_10 = +0b1.10*2^-9 = 0.00292969
0x1f = 0_00111_11 = +0b1.11*2^-9 = 0.00341797

0x20 = 0_01000_00 = +0b1.00*2^-8 = 0.00390625
0x21 = 0_01000_01 = +0b1.01*2^-8 = 0.00488281
0x22 = 0_01000_10 = +0b1.10*2^-8 = 0.00585938
0x23 = 0_01000_11 = +0b1.11*2^-8 = 0.00683594
0x24 = 0_01001_00 = +0b1.00*2^-7 = 0.0078125
0x25 = 0_01001_01 = +0b1.01*2^-7 = 0.00976562
0x26 = 0_01001_10 = +0b1.10*2^-7 = 0.0117188
0x27 = 0_01001_11 = +0b1.11*2^-7 = 0.0136719
0x28 = 0_01010_00 = +0b1.00*2^-6 = 0.015625
0x29 = 0_01010_01 = +0b1.01*2^-6 = 0.0195312
0x2a = 0_01010_10 = +0b1.10*2^-6 = 0.0234375
0x2b = 0_01010_11 = +0b1.11*2^-6 = 0.0273438
0x2c = 0_01011_00 = +0b1.00*2^-5 = 0.03125
0x2d = 0_01011_01 = +0b1.01*2^-5 = 0.0390625
0x2e = 0_01011_10 = +0b1.10*2^-5 = 0.046875
0x2f = 0_01011_11 = +0b1.11*2^-5 = 0.0546875

0x30 = 0_01100_00 = +0b1.00*2^-4 = 0.0625
0x31 = 0_01100_01 = +0b1.01*2^-4 = 0.078125
0x32 = 0_01100_10 = +0b1.10*2^-4 = 0.09375
0x33 = 0_01100_11 = +0b1.11*2^-4 = 0.109375
0x34 = 0_01101_00 = +0b1.00*2^-3 = 0.125
0x35 = 0_01101_01 = +0b1.01*2^-3 = 0.15625
0x36 = 0_01101_10 = +0b1.10*2^-3 = 0.1875
0x37 = 0_01101_11 = +0b1.11*2^-3 = 0.21875
0x38 = 0_01110_00 = +0b1.00*2^-2 = 0.25
0x39 = 0_01110_01 = +0b1.01*2^-2 = 0.3125
0x3a = 0_01110_10 = +0b1.10*2^-2 = 0.375
0x3b = 0_01110_11 = +0b1.11*2^-2 = 0.4375
0x3c = 0_01111_00 = +0b1.00*2^-1 = 0.5
0x3d = 0_01111_01 = +0b1.01*2^-1 = 0.625
0x3e = 0_01111_10 = +0b1.10*2^-1 = 0.75
0x3f = 0_01111_11 = +0b1.11*2^-1 = 0.875

0x40 = 0_10000_00 = +0b1.00*2^0 = 1
0x41 = 0_10000_01 = +0b1.01*2^0 = 1.25
0x42 = 0_10000_10 = +0b1.10*2^0 = 1.5
0x43 = 0_10000_11 = +0b1.11*2^0 = 1.75
0x44 = 0_10001_00 = +0b1.00*2^1 = 2
0x45 = 0_10001_01 = +0b1.01*2^1 = 2.5
0x46 = 0_10001_10 = +0b1.10*2^1 = 3
0x47 = 0_10001_11 = +0b1.11*2^1 = 3.5
0x48 = 0_10010_00 = +0b1.00*2^2 = 4
0x49 = 0_10010_01 = +0b1.01*2^2 = 5
0x4a = 0_10010_10 = +0b1.10*2^2 = 6
0x4b = 0_10010_11 = +0b1.11*2^2 = 7
0x4c = 0_10011_00 = +0b1.00*2^3 = 8
0x4d = 0_10011_01 = +0b1.01*2^3 = 10
0x4e = 0_10011_10 = +0b1.10*2^3 = 12
0x4f = 0_10011_11 = +0b1.11*2^3 = 14

0x50 = 0_10100_00 = +0b1.00*2^4 = 16
0x51 = 0_10100_01 = +0b1.01*2^4 = 20
0x52 = 0_10100_10 = +0b1.10*2^4 = 24
0x53 = 0_10100_11 = +0b1.11*2^4 = 28
0x54 = 0_10101_00 = +0b1.00*2^5 = 32
0x55 = 0_10101_01 = +0b1.01*2^5 = 40
0x56 = 0_10101_10 = +0b1.10*2^5 = 48
0x57 = 0_10101_11 = +0b1.11*2^5 = 56
0x58 = 0_10110_00 = +0b1.00*2^6 = 64
0x59 = 0_10110_01 = +0b1.01*2^6 = 80
0x5a = 0_10110_10 = +0b1.10*2^6 = 96
0x5b = 0_10110_11 = +0b1.11*2^6 = 112
0x5c = 0_10111_00 = +0b1.00*2^7 = 128
0x5d = 0_10111_01 = +0b1.01*2^7 = 160
0x5e = 0_10111_10 = +0b1.10*2^7 = 192
0x5f = 0_10111_11 = +0b1.11*2^7 = 224

0x60 = 0_11000_00 = +0b1.00*2^8 = 256
0x61 = 0_11000_01 = +0b1.01*2^8 = 320
0x62 = 0_11000_10 = +0b1.10*2^8 = 384
0x63 = 0_11000_11 = +0b1.11*2^8 = 448
0x64 = 0_11001_00 = +0b1.00*2^9 = 512
0x65 = 0_11001_01 = +0b1.01*2^9 = 640
0x66 = 0_11001_10 = +0b1.10*2^9 = 768
0x67 = 0_11001_11 = +0b1.11*2^9 = 896
0x68 = 0_11010_00 = +0b1.00*2^10 = 1024
0x69 = 0_11010_01 = +0b1.01*2^10 = 1280
0x6a = 0_11010_10 = +0b1.10*2^10 = 1536
0x6b = 0_11010_11 = +0b1.11*2^10 = 1792
0x6c = 0_11011_00 = +0b1.00*2^11 = 2048
0x6d = 0_11011_01 = +0b1.01*2^11 = 2560
0x6e = 0_11011_10 = +0b1.10*2^11 = 3072
0x6f = 0_11011_11 = +0b1.11*2^11 = 3584

0x70 = 0_11100_00 = +0b1.00*2^12 = 4096
0x71 = 0_11100_01 = +0b1.01*2^12 = 5120
0x72 = 0_11100_10 = +0b1.10*2^12 = 6144
0x73 = 0_11100_11 = +0b1.11*2^12 = 7168
0x74 = 0_11101_00 = +0b1.00*2^13 = 8192
0x75 = 0_11101_01 = +0b1.01*2^13 = 10240
0x76 = 0_11101_10 = +0b1.10*2^13 = 12288
0x77 = 0_11101_11 = +0b1.11*2^13 = 14336
0x78 = 0_11110_00 = +0b1.00*2^14 = 16384
0x79 = 0_11110_01 = +0b1.01*2^14 = 20480
0x7a = 0_11110_10 = +0b1.10*2^14 = 24576
0x7b = 0_11110_11 = +0b1.11*2^14 = 28672
0x7c = 0_11111_00 = +0b1.00*2^15 = 32768
0x7d = 0_11111_01 = +0b1.01*2^15 = 40960
0x7e = 0_11111_10 = +0b1.10*2^15 = 49152
0x7f = 0_11111_11 = +Inf

0x80 = 1_00000_00 = NaN
0x81 = 1_00000_01 = -0b0.01*2^-15 = -7.62939e-06
0x82 = 1_00000_10 = -0b0.10*2^-15 = -1.52588e-05
0x83 = 1_00000_11 = -0b0.11*2^-15 = -2.28882e-05
0x84 = 1_00001_00 = -0b1.00*2^-15 = -3.05176e-05
0x85 = 1_00001_01 = -0b1.01*2^-15 = -3.8147e-05
0x86 = 1_00001_10 = -0b1.10*2^-15 = -4.57764e-05
0x87 = 1_00001_11 = -0b1.11*2^-15 = -5.34058e-05
0x88 = 1_00010_00 = -0b1.00*2^-14 = -6.10352e-05
0x89 = 1_00010_01 = -0b1.01*2^-14 = -7.62939e-05
0x8a = 1_00010_10 = -0b1.10*2^-14 = -9.15527e-05
0x8b = 1_00010_11 = -0b1.11*2^-14 = -0.000106812
0x8c = 1_00011_00 = -0b1.00*2^-13 = -0.00012207
0x8d = 1_00011_01 = -0b1.01*2^-13 = -0.000152588
0x8e = 1_00011_10 = -0b1.10*2^-13 = -0.000183105
0x8f = 1_00011_11 = -0b1.11*2^-13 = -0.000213623

0x90 = 1_00100_00 = -0b1.00*2^-12 = -0.000244141
0x91 = 1_00100_01 = -0b1.01*2^-12 = -0.000305176
0x92 = 1_00100_10 = -0b1.10*2^-12 = -0.000366211
0x93 = 1_00100_11 = -0b1.11*2^-12 = -0.000427246
0x94 = 1_00101_00 = -0b1.00*2^-11 = -0.000488281
0x95 = 1_00101_01 = -0b1.01*2^-11 = -0.000610352
0x96 = 1_00101_10 = -0b1.10*2^-11 = -0.000732422
0x97 = 1_00101_11 = -0b1.11*2^-11 = -0.000854492
0x98 = 1_00110_00 = -0b1.00*2^-10 = -0.000976562
0x99 = 1_00110_01 = -0b1.01*2^-10 = -0.0012207
0x9a = 1_00110_10 = -0b1.10*2^-10 = -0.00146484
0x9b = 1_00110_11 = -0b1.11*2^-10 = -0.00170898
0x9c = 1_00111_00 = -0b1.00*2^-9 = -0.00195312
0x9d = 1_00111_01 = -0b1.01*2^-9 = -0.00244141
0x9e = 1_00111_10 = -0b1.10*2^-9 = -0.00292969
0x9f = 1_00111_11 = -0b1.11*2^-9 = -0.00341797

0xa0 = 1_01000_00 = -0b1.00*2^-8 = -0.00390625
0xa1 = 1_01000_01 = -0b1.01*2^-8 = -0.00488281
0xa2 = 1_01000_10 = -0b1.10*2^-8 = -0.00585938
0xa3 = 1_01000_11 = -0b1.11*2^-8 = -0.00683594
0xa4 = 1_01001_00 = -0b1.00*2^-7 = -0.0078125
0xa5 = 1_01001_01 = -0b1.01*2^-7 = -0.00976562
0xa6 = 1_01001_10 = -0b1.10*2^-7 = -0.0117188
0xa7 = 1_01001_11 = -0b1.11*2^-7 = -0.0136719
0xa8 = 1_01010_00 = -0b1.00*2^-6 = -0.015625
0xa9 = 1_01010_01 = -0b1.01*2^-6 = -0.0195312
0xaa = 1_01010_10 = -0b1.10*2^-6 = -0.0234375
0xab = 1_01010_11 = -0b1.11*2^-6 = -0.0273438
0xac = 1_01011_00 = -0b1.00*2^-5 = -0.03125
0xad = 1_01011_01 = -0b1.01*2^-5 = -0.0390625
0xae = 1_01011_10 = -0b1.10*2^-5 = -0.046875
0xaf = 1_01011_11 = -0b1.11*2^-5 = -0.0546875

0xb0 = 1_01100_00 = -0b1.00*2^-4 = -0.0625
0xb1 = 1_01100_01 = -0b1.01*2^-4 = -0.078125
0xb2 = 1_01100_10 = -0b1.10*2^-4 = -0.09375
0xb3 = 1_01100_11 = -0b1.11*2^-4 = -0.109375
0xb4 = 1_01101_00 = -0b1.00*2^-3 = -0.125
0xb5 = 1_01101_01 = -0b1.01*2^-3 = -0.15625
0xb6 = 1_01101_10 = -0b1.10*2^-3 = -0.1875
0xb7 = 1_01101_11 = -0b1.11*2^-3 = -0.21875
0xb8 = 1_01110_00 = -0b1.00*2^-2 = -0.25
0xb9 = 1_01110_01 = -0b1.01*2^-2 = -0.3125
0xba = 1_01110_10 = -0b1.10*2^-2 = -0.375
0xbb = 1_01110_11 = -0b1.11*2^-2 = -0.4375
0xbc = 1_01111_00 = -0b1.00*2^-1 = -0.5
0xbd = 1_01111_01 = -0b1.01*2^-1 = -0.625
0xbe = 1_01111_10 = -0b1.10*2^-1 = -0.75
0xbf = 1_01111_11 = -0b1.11*2^-1 = -0.875

0xc0 = 1_10000_00 = -0b1.00*2^0 = -1
0xc1 = 1_10000_01 = -0b1.01*2^0 = -1.25
0xc2 = 1_10000_10 = -0b1.10*2^0 = -1.5
0xc3 = 1_10000_11 = -0b1.11*2^0 = -1.75
0xc4 = 1_10001_00 = -0b1.00*2^1 = -2
0xc5 = 1_10001_01 = -0b1.01*2^1 = -2.5
0xc6 = 1_10001_10 = -0b1.10*2^1 = -3
0xc7 = 1_10001_11 = -0b1.11*2^1 = -3.5
0xc8 = 1_10010_00 = -0b1.00*2^2 = -4
0xc9 = 1_10010_01 = -0b1.01*2^2 = -5
0xca = 1_10010_10 = -0b1.10*2^2 = -6
0xcb = 1_10010_11 = -0b1.11*2^2 = -7
0xcc = 1_10011_00 = -0b1.00*2^3 = -8
0xcd = 1_10011_01 = -0b1.01*2^3 = -10
0xce = 1_10011_10 = -0b1.10*2^3 = -12
0xcf = 1_10011_11 = -0b1.11*2^3 = -14

0xd0 = 1_10100_00 = -0b1.00*2^4 = -16
0xd1 = 1_10100_01 = -0b1.01*2^4 = -20
0xd2 = 1_10100_10 = -0b1.10*2^4 = -24
0xd3 = 1_10100_11 = -0b1.11*2^4 = -28
0xd4 = 1_10101_00 = -0b1.00*2^5 = -32
0xd5 = 1_10101_01 = -0b1.01*2^5 = -40
0xd6 = 1_10101_10 = -0b1.10*2^5 = -48
0xd7 = 1_10101_11 = -0b1.11*2^5 = -56
0xd8 = 1_10110_00 = -0b1.00*2^6 = -64
0xd9 = 1_10110_01 = -0b1.01*2^6 = -80
0xda = 1_10110_10 = -0b1.10*2^6 = -96
0xdb = 1_10110_11 = -0b1.11*2^6 = -112
0xdc = 1_10111_00 = -0b1.00*2^7 = -128
0xdd = 1_10111_01 = -0b1.01*2^7 = -160
0xde = 1_10111_10 = -0b1.10*2^7 = -192
0xdf = 1_10111_11 = -0b1.11*2^7 = -224

0xe0 = 1_11000_00 = -0b1.00*2^8 = -256
0xe1 = 1_11000_01 = -0b1.01*2^8 = -320
0xe2 = 1_11000_10 = -0b1.10*2^8 = -384
0xe3 = 1_11000_11 = -0b1.11*2^8 = -448
0xe4 = 1_11001_00 = -0b1.00*2^9 = -512
0xe5 = 1_11001_01 = -0b1.01*2^9 = -640
0xe6 = 1_11001_10 = -0b1.10*2^9 = -768
0xe7 = 1_11001_11 = -0b1.11*2^9 = -896
0xe8 = 1_11010_00 = -0b1.00*2^10 = -1024
0xe9 = 1_11010_01 = -0b1.01*2^10 = -1280
0xea = 1_11010_10 = -0b1.10*2^10 = -1536
0xeb = 1_11010_11 = -0b1.11*2^10 = -1792
0xec = 1_11011_00 = -0b1.00*2^11 = -2048
0xed = 1_11011_01 = -0b1.01*2^11 = -2560
0xee = 1_11011_10 = -0b1.10*2^11 = -3072
0xef = 1_11011_11 = -0b1.11*2^11 = -3584

0xf0 = 1_11100_00 = -0b1.00*2^12 = -4096
0xf1 = 1_11100_01 = -0b1.01*2^12 = -5120
0xf2 = 1_11100_10 = -0b1.10*2^12 = -6144
0xf3 = 1_11100_11 = -0b1.11*2^12 = -7168
0xf4 = 1_11101_00 = -0b1.00*2^13 = -8192
0xf5 = 1_11101_01 = -0b1.01*2^13 = -10240
0xf6 = 1_11101_10 = -0b1.10*2^13 = -12288
0xf7 = 1_11101_11 = -0b1.11*2^13 = -14336
0xf8 = 1_11110_00 = -0b1.00*2^14 = -16384
0xf9 = 1_11110_01 = -0b1.01*2^14 = -20480
0xfa = 1_11110_10 = -0b1.10*2^14 = -24576
0xfb = 1_11110_11 = -0b1.11*2^14 = -28672
0xfc = 1_11111_00 = -0b1.00*2^15 = -32768
0xfd = 1_11111_01 = -0b1.01*2^15 = -40960
0xfe = 1_11111_10 = -0b1.10*2^15 = -49152
0xff = 1_11111_11 = -Inf
```

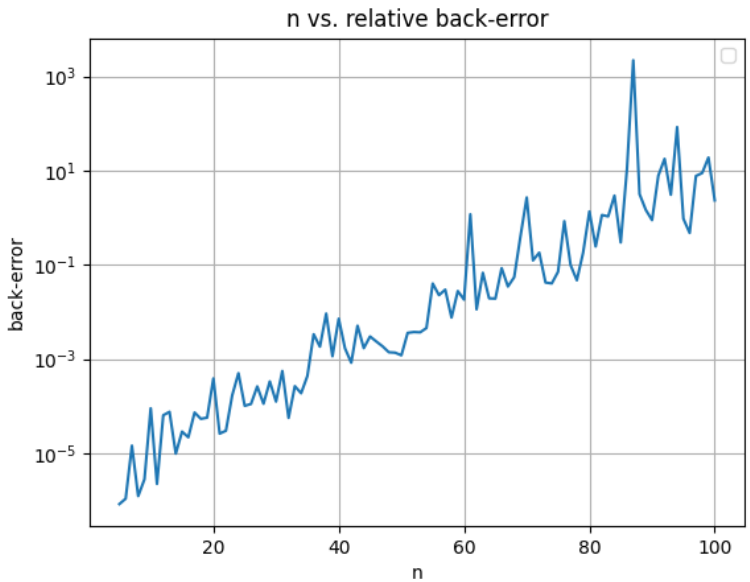
Plots



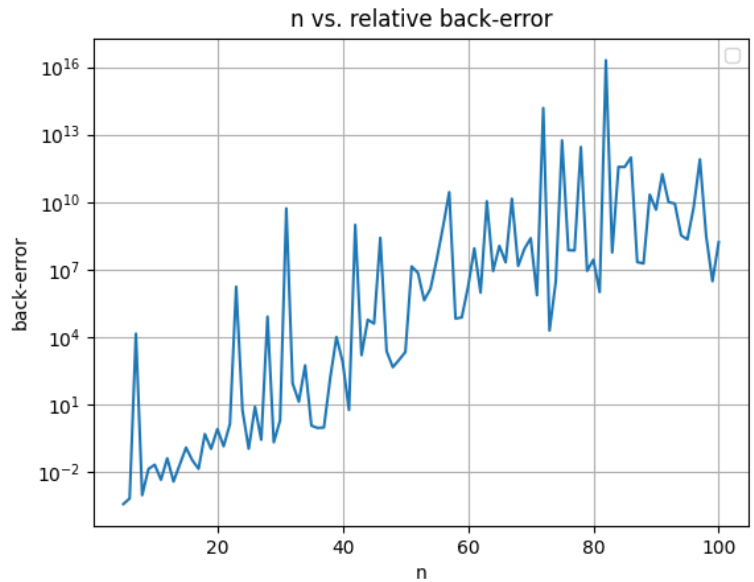
(a) $n = 30$ in binary8p4; $\epsilon_{mach} = 0.125$ (b) $n = 30$ in binary8p3; $\epsilon_{mach} = 0.25$

Figure: Relative Backwards error vs condition number for different precisions after 5 iterations of iterative refinement in fp32

More plots



(a) *cond* ≈ 20 in binary8p4; $\epsilon_{mach} = 0.125$



(b) *cond* ≈ 20 in binary8p3; $\epsilon_{mach} = 0.25$

Figure: Relative Backwards error vs n for different precisions after 5 iterations of iterative refinement in fp32

Some hands-on experiences

Open your laptops!

`https://github.com/tlapack/tutorials`

We will make:

- LU in multi-precision
- Recursive matrix-matrix multiplication