Automatic Sparsity Detection in LAPACK

Razvan Carbunescu

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-224
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-224.html

December 18, 2014

Automatic Sparsity Detection in LAPACK

by

Razvan Corneliu Carbunescu

A thesis submitted in partial satisfaction of the
requirements for the degree of

Master of Science
in

Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Fall 2014

Automatic Sparsity Detection in LAPACK

Abstract

Automatic Sparsity Detection in LAPACK

by

Razvan Corneliu Carbunescu

Master of Science in Computer Sciences

University of California, Berkeley

Professor Katherine Yelick, Chair

Many applications call linear algebra libraries as methods of achieving better performance and reliability. LAPACK (Linear Algebra Package) is a standard software library for numerical linear algebra that is widely used in the industrial and scientific community. LAPACK functions require the user to know the sparsity and other mathematical structure of their inputs to be able to take advantage of the fastest codes: General Matrix (GE), General Band (GB), Positive Definite (PO) etc. If a user is unsure of their matrix structure or cannot easily express it in the formats available (profile matrices, arrow matrices etc.) they are forced to use a more general structure, which includes their input, and so run less efficiently than possible.

The goal of this thesis is to allow for automatic sparsity detection (ASD) within LAPACK that is completely hidden from the user and provides no slowdown for users running fully dense matrices. This work adds modular support for the detection of blocked sparsity within LAPACK LU and Cholesky functions. It also creates the infrastructure and the algorithms to potentially expand sparsity detection to other factorizations, more input matrix structures, or provide further timing and memory improvements via integration directly in the solver routines. Two general approaches are implemented named 'Profile' (ASD1) and 'Sparse block' (ASD2) with a third more complicated method named 'Full sparsity' (ASD3) being described more abstractly, only at an algorithm level.

With these algorithms we obtain benefits of up to an order of magnitude (35.10$\times$ faster over the same LAPACK function) for matrices displaying 'blocked sparsity' patterns and large benefits over the best LAPACK algorithms for patterns that don't fit into LAPACK categories (4.85$\times$ faster over the best LAPACK function). For matrices exhibiting no sparsity these implementations incur either a negligible penalty (an overhead of 1%) or incur a small overhead (10-30%) that quickly decreases with the size of matrix n or band b (less than 5% for $n, b > 500$).

Contents

List of Figures

List of Tables

Acknowledgements

I would first like to thank my advisor James Demmel for the time and energy he has provided not only during the writing of this thesis but along my six and a half year stay at UC Berkeley. I also want to thank profusely my committee members Katherine Yelick and Jonathan Shewchuk for taking the time to give input, criticism and ideas for the thesis.

More generally I'd like to thank my department EECS for the patience and understanding to let me arrive at a topic which I considered important myself for a thesis topic.

Last but not least I would also like to thank all the current and former graduate students in the CS department who I've interacted over the years: Andrew Gearhart, Grey Ballard, Erin Carson, Nicholas Knight, Mark Hoemman, Katya Gonina, Aditya Devarakonda, Edgar Solomonik, Andrew Waterman, Sarah Bird, Yunsup Lee to name just a few, as casual discussions on wide range of topics have improved my knowledge of Computer Science by an immense amount.

Chapter 1

Introduction

1.1 Motivation

LAPACK (Linear Algebra Package) is a standard software library for numerical linear algebra that is widely used in the industrial and scientific community. Over time, the full matrix input format (GE) has been supplemented with other formats (Banded (GB), Positive Definite (PO), etc.) to allow codes to take advantage of reduced memory and floating point operations [11, 3, 9]. There are now 28 different matrix formats [1] (shown in Table 1.1) that a user can choose from for a matrix input, although not all combinations of routines and matrix formats are provided or logical. These formats are restricted to non-sparse representations with no sparse options available within LAPACK. In this thesis we consider a band format dense because all entries stored in the format are treated as nonzeros. Taking advantage of the structures listed above only works if the user knows beforehand and transforms the input to the required formats which are then fed into the LAPACK routines. This 'direct knowledge' is what allows for lower flop and memory costs for performing the routines.

This requirement limits non-expert users that do not check their matrices from receiving the best performance that could be used to solve their problems. Even expert users might be forced to call non-optimal codes for structures that do not fit into exactly into the categories provided by LAPACK. For the non-standard structures, users usually decide to call the more general routine which could be much more costly. An aspect to consider is that, due to the popularity of LAPACK as a numerical linear algebra library and the relative ease of its installation on most platforms, there are cases when matrices that could be best solved via sparse solvers like SuperLU [7, 10] are processed via their dense or band equivalent calls. For example, such matrices may be combinations of stencils that do not fit the simple stencil models but could benefit greatly from having their sparsity structure exploited.

Name	Description
BD	bidiagonal
DI	diagonal
GB	general band
GE	general (i.e., unsymmetric, in some cases rectangular)
GG	general matrices, generalized problem (i.e., a pair of general matrices)
GT	general tridiagonal
HB	(complex) Hermitian band
HE	(complex) Hermitian
HG	upper Hessenberg matrix, generalized problem (i.e. a Hessenberg and a triangular matrix)
HP	(complex) Hermitian, packed storage
HS	upper Hessenberg
OP	(real) orthogonal, packed storage
OR	(real) orthogonal
PB	symmetric or Hermitian positive definite band
PO	symmetric or Hermitian positive definite
PP	symmetric or Hermitian positive definite, packed storage
PT	symmetric or Hermitian positive definite tridiagonal
SB	(real) symmetric band
SP	symmetric, packed storage
ST	(real) symmetric tridiagonal
SY	symmetric
TB	triangular band
TG	triangular matrices, generalized problem (i.e., a pair of triangular matrices)
TP	triangular, packed storage
TR	triangular (or in some cases quasi-triangular)
TZ	trapezoidal
UN	(complex) unitary
UP	(complex) unitary, packed storage

Table 1.1. Matrix types for LAPACK routines.

Another option available for exploiting the sparsity structure of a matrix is to try to automatically detect it upon matrix entry or creation during the sub-steps of the algorithms. There exists one LAPACK routine, the QR factorization xGEQRF, that takes partial advantage of detecting sparsity when applying the block Householder rotation matrix to the remaining unfactorized part of the matrix. This optimization significantly reduces the running time of the QR factorization by up to an order of magnitude for matrices with many trailing zero rows/columns [8].

1.2 Goal

The goal of this thesis is to allow for "Automatic Sparsity Detection" (ASD) within LAPACK that a) is completely hidden from the user, b) provides no slowdown for users running fully dense matrices, and c) provides as close as possible to the performance that the user would have gotten using the best code/format available. The results gathered show that we achieve these goals, with the extra advantage that, due to the more detailed exploitation of sparsity within each factorization block rather than the matrix as a whole, we are sometimes able to beat the best LAPACK alternatives. For sparse matrices we can also achieve a high improvement over the current LAPACK alternatives, but we do not hope to beat special-purpose sparse solvers like SuperLU [7] or UMFPack [4].

Keeping in mind our goal of eventually expanding ASD to all relevant LAPACK routines, the detection, auxiliary, and LU/Cholesky routines are broken up into multiple modular parts rather than implemented within each routine. This modularity allows for reuse and also for future easy improvement on sub-parts of the sparsity exploitation or methods of improving and lowering overhead for the more complicated ASD algorithms.

1.3 Outline

The sparsity detection codes are presented in Chapter 2. These codes serve to perform the actual comparisons with zeros and return either bounds or sparsity estimates. They are split into "Sparse Zero Checks" (xSZCHK), "Blocked Zero Checks" (xBZCHK), and "Full Zero Checks" (xFZCHK).

The auxiliary functions are presented in Chapter 3 and are split into two categories. The first are the integer limit manipulations which work solely on matrix bounds expressed in integer limit arrays, which indicate the boundaries of groups of nonzero rows or columns. These are: "Sparsity Selection" (IZSS), "Zero Limit Combination" (IZCOMB), and "Zero Limit Subtraction" (IZSUB). The second are the integer limit LAPACK extensions which take the computed limit arrays from the first category and apply them to the normal LU auxiliary functions within each block. The wrapper functions are xTRSM_ILIM, xGEMM_ILIM, xSYRK_ILIM, xGETF2_ILIM, xBZCHK_ILIM.

The LU factorization algorithms are presented in Chapter 4 and represent three general approaches named "Profile" (ASD1), "Sparse block" (ASD2) and "Full sparsity" (ASD3). The first two approaches mentioned are implemented as F77/F90 functions taking advantage of the auxiliary codes and sparsity checks listed above. These can be directly integrated into LAPACK with no change (for ASD1) and an extra integer workspace (for ASD2). The ASD3 approach is only described at the algorithm and pseudo-code level since it would have required an interface change within LAPACK to allow for column permutations, and it can also potentially influence the pivoting of the answer.

In Chapter 5 we present the results of the ASD1 and ASD2 implementations when applied to full dense, dense banded, arrow, and varying banded (profile and bulge) matrices. A test suite of sparse matrices taken from the University of Florida sparse matrix collection is also presented. These results show great potential benefits for the implementations with very little overhead as can be seen at in Table 1.2 [1]:

Input	ASD1(GE)	ASD2(GE)	ASD1(GB)	ASD1(PO)	ASD2(PO)	ASD1(PB)
Arrow	1.00	24.20	N/A	7.31	11.15	N/A
Band	32.81	31.72	0.99	17.87	12.18	1
Profile	35.10	33.90	2.203	19.00	14.76	2.23
Bulge	21.82	21.32	3.88	14.14	9.26	4.85
Sparse	23.42	27.21	2.20	395.80	10.32	2.44

Table 1.2. Maximum speed-up over current LAPACK routine.

In the table above ASDz(YY) represents: ASD1 or ASD2 depending on $z = 1$ or 2, which is running the ASDz_S(YY)TRF routine that is compared to the LAPACK S(YY)TRF routine in order to determine speedup.

Plans for inclusion of the ASD implementations into the next LAPACK release as well as other potential future work are described in Chapter 6.

Appendix A contains copies of all code implementations for the single-precision variants of ASD1 and ASD2. Appendix B contains copies of the raw timing results in Comma Separated Value (CSV) format.

[1]All generated matrices in the summary results are random diagonally dominant and symmetric positive definite.

Chapter 2

Sparsity Detection

Detecting sparsity in LAPACK is a simple process of checking the entries for zero values. However the problem comes in ensuring that these checks are fast enough that no extra time is added for the most common cases. For example, while a full dense matrix could be checked for zeros in $O(n^2)$ time, which is smaller than the flop count of $O(n^3)$, this could still present a problem for small values of n where the slowdown could be noticeable to the user.

To address this issue, three decisions were made. First, all the sparsity checks were added to the BLAS3 versions only, with any matrix that doesn't satisfy the size requirements, for using the BLAS3 version of LU, defaulting to the BLAS2 routines before any sparsity checks are called. In this way the smallest matrices most susceptible to slowdowns are avoided.

Second, any check on a matrix is only done once per element in the array. While this means that extra data structures and algorithmic complexity is needed for the more complicated algorithms, this ensures that the input matrix entries are referenced at most twice to check sparsity, and even checking twice is a very rare occurrence.

Third, the checks themselves are performed in stages on the input matrices, ensuring that minimal checks are performed on full dense matrices and that the full $O(n^2)$ or $O(nb)$ checks are performed, where b is the matrix bandwidth, only on matrices that have large sparsity that could be exploited.

To facilitate this third point, the checks themselves can be split into three categories: sparse, blocked, and full checks. All checks currently assume that a general matrix is given as input since this can be used for general, banded, positive definite, and positive definite band via different input choices to the functions.

2.1 Sparse Zero Checks - xSZCHK

2.1.1 Description

The sparse zero checks are designed to give a fast $O(n)$ estimate on whether a input matrix contains any sparsity. It also provides the basis for determining the number of sub-blocks to be used in the panel and the updates of the rest of the matrix. Since the goal of the function is to analyse and guess multiple structures, no single deterministic choice for the elements to check exists. Therefore the function uses a random pattern to guess the structure of the input matrix and the implementation provides one of eight possible patterns to be used by the caller. These eight options are presented below and in Figure 2.1. They are as follows:

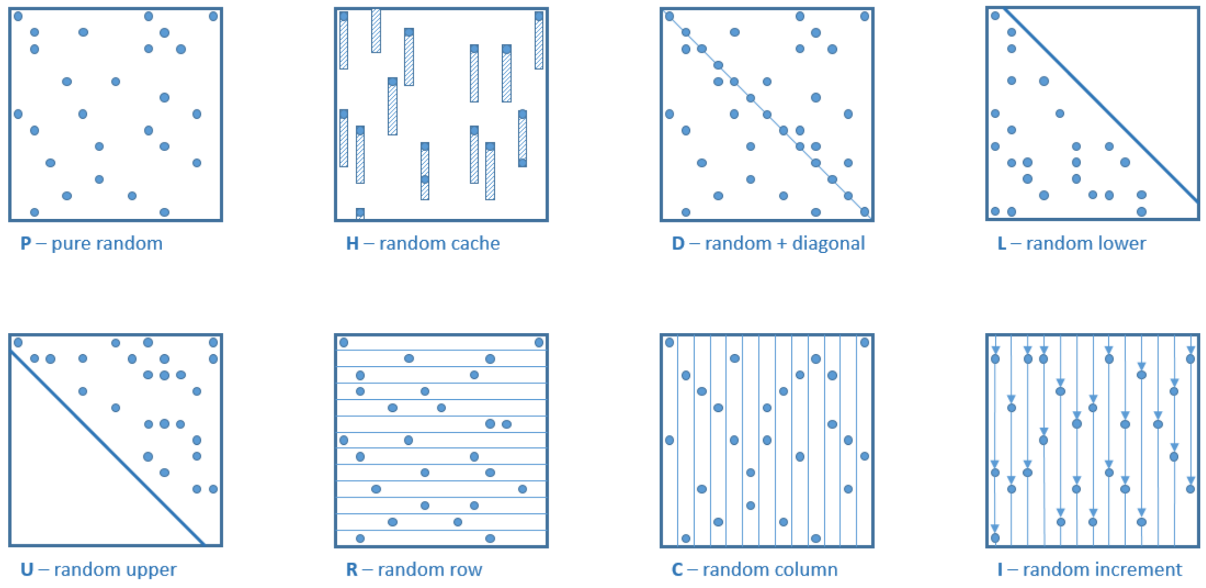


Figure 2.1. Sparse check patterns.

P pure random: select $O(n)$ random elements from the matrix

H cached random: select $O(n)$ random elements but then check all elements in the cache line (assumes matrix is cache-aligned on entry)

L pure random: select $O(n)$ random elements from the lower-triangular part of the matrix

U pure random: select $O(n)$ random elements from the upper-triangular part of the matrix

D random with diagonal: select $O(n)$ elements at random but also select diagonal elements

R random row: select $O(1)$ random elements in each row

C random column: select $O(1)$ random elements in each column

I random increment: select $O(n)$ random elements by choosing a small random increment

Regardless of the random pattern, the sparse checks return a number that represents the fraction of non-zero elements out of the total number of elements checked. This estimate for the matrix is used to determine whether to apply any further sparsity checks to the matrix and to determine the number of sub-blocks to potentially split the matrix into. This sub-block split is what allows the later block checks to catch zero rows inside the interiors of panels or zero columns inside the interior of blocks of A rather than just being able to detect the profile of a matrix.

Since these patterns are separated from the linear algebra routines that use them, if better options for sparsity detection are invented, they can be integrated into a single function. Afterwards, all sparsity algorithms can be changed via a simple parameter modification. This split also allows for different algorithms to use different sparsity checks depending on the needs of their codes. A simple example of this is the POTRF functions for Cholesky factorization that need to use the 'L' or 'U' options to ensure that only the upper-triangular or lower-triangular parts of A are addressed by the checks.

2.1.2 Implementation

The interface for the function calls is xSZCHK($m, n, A, lda, ctype, fracnz, sc, nnz$) with:

- Inputs:
  - m, n: the dimensions of matrix A
  - A: the matrix to be checked
  - lda: the leading dimension of A
  - sc: the sparsity constant to use to check $sc \times (m + n)$ elements
  - $ctype$: the type of pattern to use
- Outputs:
  - $fracnz$: the fraction of non-zeros
  - nnz: the number of non-zeros
- Algorithm:

```
Reset random number generator
Calculate number of elements to check ($NC$)
if Correct pattern selected then
    Reset $nnz$ to 0
    for $k$ from 1 to $NC$ do
```

```
            Generate random number
            Compute index position $i, j$ from random number
            Apply index fixes for pattern to $i, j$ (Ex: if $i > j$ but pattern is ‘U’ switch $i, j$)
            if $A(i, j) \neq 0$ then
                Increase $nnz$ by 1
            end if
        end for
        Compute fraction of non-zeros $fracnz$
    end if
    return $fracnz$ and $nnz$
```

2.1.3 Considerations

The implementation above returns the fraction of non-zeros $fracnz$ and number of non-zeros nnz out of the $sc \times (m+n)$ tested numbers. By having this relative definition for sparsity, we accommodate the possibility of using this check on subblocks of matrices. This implementation is only used on the full input matrices in ASD2_xGETRF and ASD2_xPOTRF.

If an incorrect pattern is entered, the function will return a $fracnz = 1.0$ representing a full matrix, since this guarantees a correct result via the previous LAPACK functions.

The current random number generator used is the intrinsic IRAND function from Fortran, which is reliable enough for the purposes of this function. While the patterns ‘U’ and ‘L’ avoid referencing parts of the array the function is implemented for general matrices only with no other format being currently supported.

In the current implementation the code treats special elements like NaN and Inf as non-zeros for the purposes of these checks.

2.1.4 Accuracy

To determine the accuracy of the various patterns, simple tests were run both over artificial input matrices (Arrow Matrix) seen in Table 2.1 and over real sparse matrices (Nasa2910) seen in Table 2.2. It can be seen from these tests that the sparsity constant sc does not strongly influence the accuracy of the fraction except for the “Diagonal” test, which performs the worst. It can be seen that for this implementation the “Diagonal” tests do not work well, but all other patterns got within a few percentage points of the true sparsity of the input matrix.

Because the tests showed very little variation it was decided to use the “Pure random” matrix sampling for the General LU factorization and the “Lower triangular” and “Upper triangular” sampling methods for the Cholesky factorization. This latter choice was made to preserve the property that Cholesky only addresses the lower or upper parts of the input matrix A.

sc	P	H	L	U	D	R	C	I
1	0.6270	0.6341	0.6365	0.6420	0.7510	0.6260	0.6200	0.5570
2	0.6432	0.6416	0.6432	0.6397	0.7202	0.6335	0.6350	0.5537
3	0.6365	0.6396	0.6431	0.6396	0.6885	0.6346	0.6303	0.5651
4	0.6318	0.6425	0.6397	0.6453	0.6816	0.6310	0.6385	0.5621

Table 2.1. Predicted $fracnz$ (arrow matrix with $n = 1000, b = 200$, true $fracnz = 0.6406$).

sc	P	H	L	U	D	R	C	I
1	0.0221	0.0215	0.0204	0.0218	0.3457	0.0226	0.0237	0.0192
2	0.0229	0.0245	0.0202	0.0213	0.2157	0.0201	0.0197	0.0201
3	0.0206	0.0209	0.0205	0.0217	0.1594	0.0224	0.0180	0.0201
4	0.0211	0.0196	0.0205	0.0195	0.1298	0.0174	0.0198	0.0187

Table 2.2. Predicted $fracnz$ (nasa2910 with $n = 2910, b = 859$, true $fracnz = 0.0205$).

2.2 Blocked Zero Checks - xBZCHK

2.2.1 Description

The blocked zero checks determine how many zero rows/columns are present in a matrix A looking both from the lower and higher ends (Figure 2.2). They return the indices of the first and last non-zero rows/columns - or in the case of an empty matrix, they return zeros for both the first and last indices.

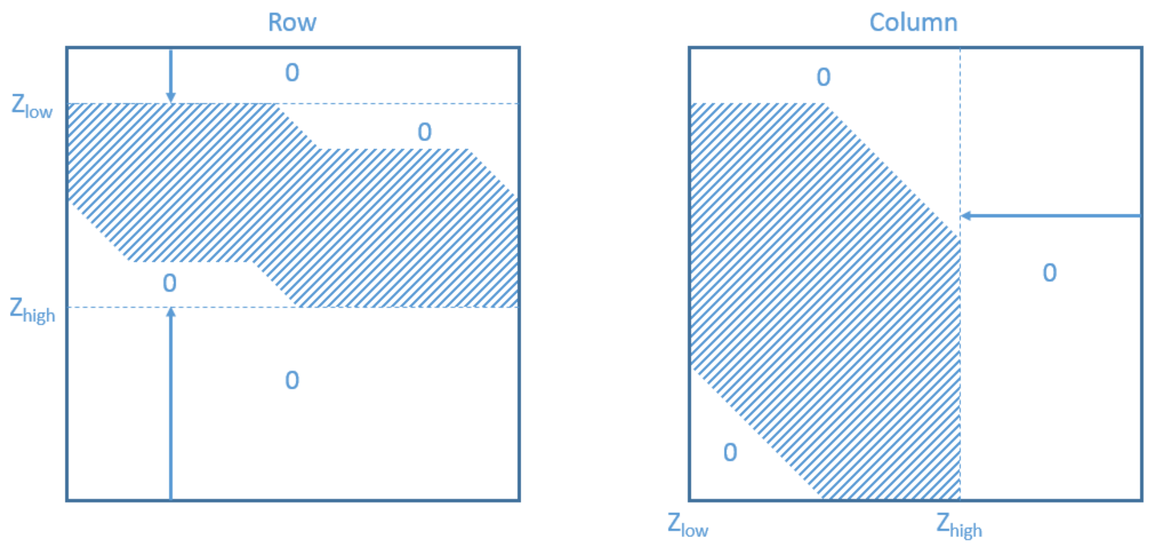


Figure 2.2. Blocked zero checks.

While the blocked zero checks are the only detection functions used by the ASD1 algorithms, they can be considered more as block limit checks to determine the bounds of BLAS3 routines, and are used by all the ASD algorithms.

2.2.2 Implementation

The interface for the function calls is xBZCHK($m, n, A, lda, dir, zlow, zhigh$) with:

- Inputs:
  - m, n: the dimensions of matrix A
  - A: the matrix to be checked
  - lda: the leading dimension of A
  - dir: the direction ('row' or 'column') to check in
- Outputs:
  - $zlow$: the index of the first non-zero row/column
  - $zhigh$: the index of the last non-zero row/column
- Algorithm:

```
if $dir = row$ then
    for $i$ from $m$ to 0 do
        Set $current\_row$ to $i$
        if $i = 0$ then Break
        end if
        for $j$ from 1 to $n$ do
            if $A(i,j) \neq 0$ then Break
            end if
        end for
    end for
    Set $zhigh$ to $current\_row$
    for $i$ from 1 to $zhigh$ do
        Set $current\_row$ to $i$
        for $j$ from 1 to $N$ do
            if $A(i,j) \neq 0$ then Break
            end if
        end for
    end for
    Set $zlow$ to $current\_row$
else if $dir = col$ then
    for $j$ from $n$ to 0 do
        Set $current\_column$ to $j$
        if $j = 0$ then Break
        end if
        for $i$ from 1 to $m$ do
            if $A(i, j) \neq 0$ then Break
            end if
        end for
    end for
    Set $zhigh$ to $current\_column$
    for $j$ from 1 to $zhigh$ do
        Set $current\_column$ to $j$
        for $i$ from 1 to $m$ do
            if $A(i, j) \neq 0$ then Break
            end if
        end for
    end for
    Set first index $zlow$ to $current\_column$
else Incorrect direction ($dir \neq row|column$)
    Set $zlow$ and $zhigh$ to $-1$
end if
return $zlow, zhigh$
```

An interesting aspect of this algorithm is that for a full matrix, there are only two comparisons that are done, and the full $O(m \times n)$ comparisons are not performed unless the matrix is close to empty.

The blocked zero checks assume a general matrix as input and also treat all special elements (Ex: NaN or Inf) as non-zero elements for the purposes of determining sparsity.

2.3 Full Zero Checks - xFZCHK

2.3.1 Description

The full zero checks are useful in the cases where a matrix is exhibiting high sparsity but would suffer from high fill-in if conventional LU is used. Their purpose is to calculate both the sparsity of the entire matrix A and the sparsities on a set of subblocks, the number of which is determined by running the sparse approximation similar to Figure 2.3.

These tools could then be used to suggest how to permute the matrix A to have low fraction sub-blocks to the upper-left and push all high fraction sub-blocks to the lower-right. The calculation of the permutation to be applied would not be considered inside the function. It would be the job of the factorization calling the zero check routine to determine the permutation.

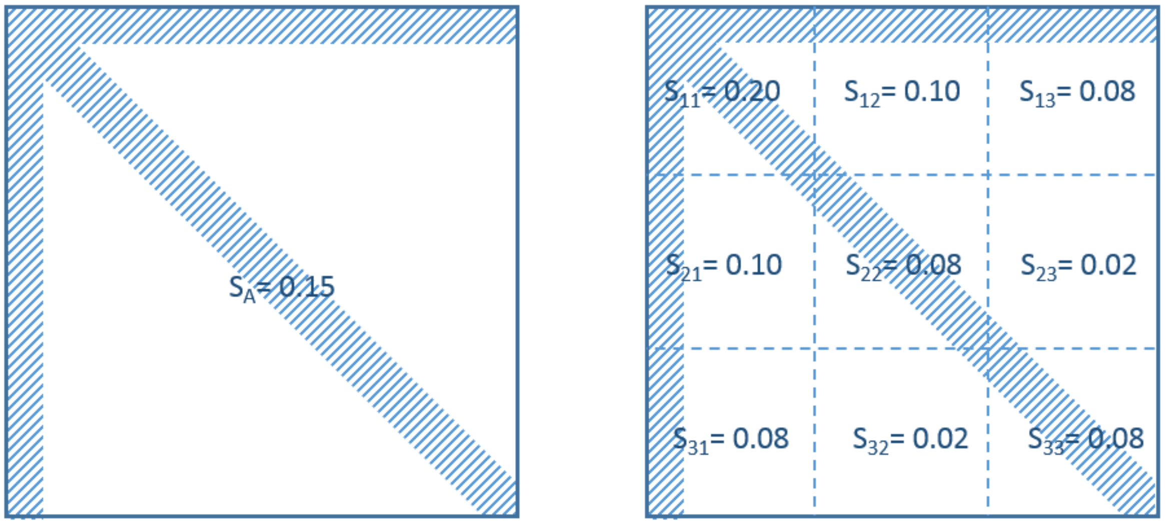


Figure 2.3. Full zero checks.

2.3.2 Implementation

The full zero check algorithm would look similar to the following pseudocode:

```
Set $Nnz$ to zero
for Each sub-block indexed by $sb$ do
    Set $Nnz_{sb}$ to zero
    for $i$ from 1 to $M_{sb}$ do
        for $j$ from 1 to $N_{sb}$ do
            if $A_{sb}(i,j) \neq 0$ then $Nnz_{sb}$++
            end if
        end for
    end for
    Set $S(i_{sb}, j_{sb}) = Nnz_{sb}/(M_{sb} \times N_{sb})$
    $Nnz = Nnz + Nnz_{sb}$
end for
Set $S_A = Nnz/(M \times N)$
return Matrix of sub-block sparsities $S$ and general sparsity $S_A$
```

Since neither ASD1 nor ASD2 implementations use the xFZCHK routines, the current implementation only provides the fraction of non-zero S_A as an output and is used to test the previous sparse and blocked implementations.

For eventual use within the “Full Sparsity” ASD3 routines, the full matrix of sub-block sparsities S shown above in pseudocode will require implementation.

Chapter 3

Sparsity Auxiliary Functions

To implement automatic sparsity detection in a modular way and to allow for ease of understanding of the different components present in the LU and Cholesky factorizations, the creation of a variety of auxiliary methods was required. These methods are all used by the more complicated “Sparse Block” (ASD2) implementations. They are split into two main categories: functions that deal with integer limit operations and block wrapper functions that call other methods over multiple independent non-zero blocks.

All of the operations either create or interact with the nonzero limit array *ilim* and the number of nonzero blocks *is*. The array presents a method of representing more complex zero structures in a way that can be exploited by the routines presented in this chapter. The array will always contain the start and end indices of nonzero rows/columns, and for a given number of blocks *is* will have $2 \times is$ elements. An example of an *ilim* array representation for the column limits of a matrix can be seen in Figure 3.1. These arrays are defined and used per each direction with certain functions like xGEMM_ILIM using both m, n nonzero limit arrays.

3.1 Integer Limit Operations

The integer limit operations either determine or modify the integer nonzero limit array *ilim* by determining the size, combining elements, or subtracting rows/columns from the entries. These functions all have the common property of using just the integer limits, size and other statistics of the matrix instead of any values. The functions are shared across all data types. There are three methods that fall under this category: Sparsity Selection (IZSS), Zero Limit Combination (IZCOMB) and Zero Limit Subtraction (IZSUB).

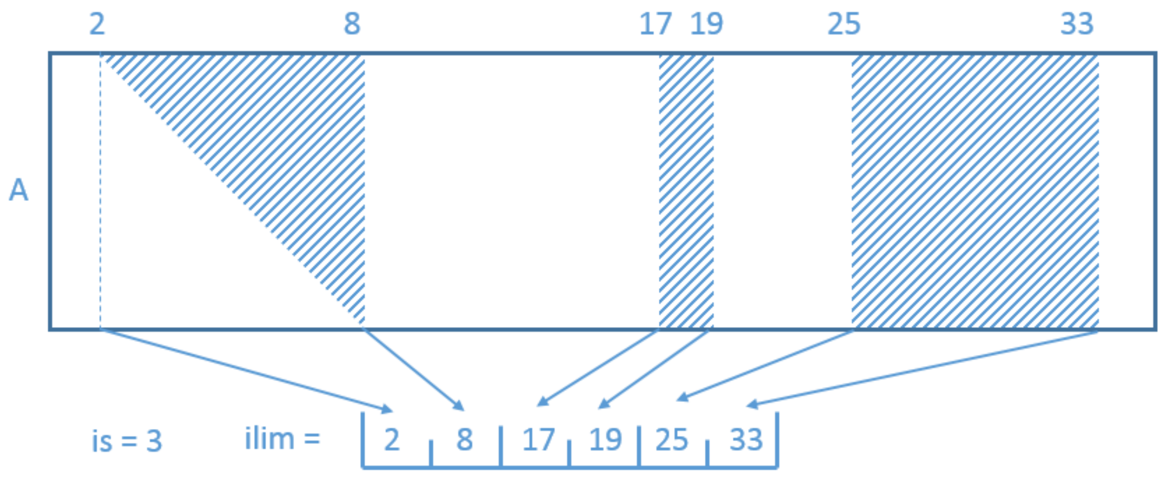


Figure 3.1. Matrix column limits represented in array *ilim*.

3.1.1 Sparsity Selection - IZSS

Description

Sparsity Selection uses the size of the matrix, panel or sub-block along with the general sparsity to determine the number of non-zero regions to consider for a direction. The method also takes into account the minimum size desired for a non-zero region.

Implementation

The interface for the function calls is $nis = \text{IZSS}(n, fracnz, minlvl)$ with:

- Inputs:
  - n: the dimension of the subblock in A
  - $minlvl$: the minimum size for a subblock
  - $fracnz$: the fraction of non-zeros in matrix A
- Outputs:
  - nis: the number of subblocks for n
- Algorithm:

```
if $n = 0$ then
    return $nis = 0$
else if $fracnz = 0$ then
```

```
        return $nis = max\{n/minlvl, 1\}$
    else
        $nis_1 = max\{n/minlvl, 1\}$
        $nis_2 = 1/fracnz + 1$
        return $nis = min\{nis_1, nis_2\}$
    end if
```

3.1.2 Zero Limit Combination - IZCOMB

Description

This auxiliary function takes as input the raw block limits from array *ilim*, deletes empty sub-blocks marked by 0 limits and combines remaining indices to create a new array of non-zero row/column limits that can serve as input for the wrapper functions. The input *ilim* can be thought of as lower and upper bounds ($2 \times is$ in number) on individual matrices that need to be combined to properly look at the whole panel or submatrix. An example can be seen in Figure 3.2.

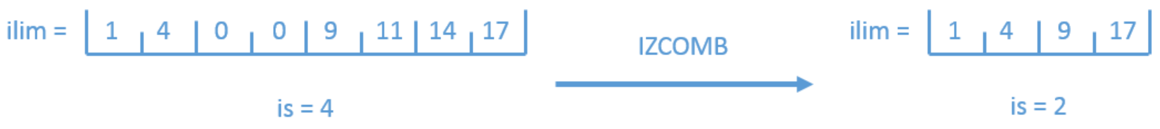


Figure 3.2. Processing an array of block limits with IZCOMB.

Implementation

The interface for the function call is IZCOMB($ilim, is, mincomb$) with:

- Inputs:
  - *ilim*: the array of $2 \times is$ limit values
  - *is*: the number of subblocks
  - *mincomb*: the number of zero rows/columns to ignore when combining
- Outputs:
  - *ilim*: the array of updated $2 \times is$ limit values
  - *is*: the updated number of subblocks

- Algorithm:

```
/* First pass to remove empty submatrices */
Set $is_{new} = 0$
**for** $i$ **from** 1 **to** $is$ **do**
    **if** $ilim(2 \times i - 1) \neq 0$ **then**
        $is_{new}++$
        $ilim(2 \times is_{new} - 1) = ilim(2 \times i - 1)$
        $ilim(2 \times is_{new}) = ilim(2 \times i)$
    **end if**
**end for**
$is = is_{new}$
/* Second pass to combine elements within $mincomb$ */
**if** $is > 0$ **then**
    Set $is_{new} = 1$
    $tmphigh = ilim(2)$
    **for** $i$ **from** 2 **to** $is$ **do**
        /* If current upper limit is within mincomb of next lower */
        **if** $ilim(2 \times i - 1) - tmp\_high \leq mincomb$ **then**
            $tmphigh = ilim(2 \times i)$
        **else**
            /* Create new blocks and set bounds */
            $ilim(2 \times is_{new}) = tmp\_high$
            $is_{new}++$
            $ilim(2 \times is_{new} - 1) = ilim(2 \times i - 1)$
            $tmphigh = ilim(2 \times i)$
        **end if**
    **end for**
    $ilim(2 \times is_{new}) = tmp\_high$
**end if**
$is = is_{new}$
**return** $ilim, is$
```

3.1.3 Zero Limit Subtraction - IZSUB

Description

This is another auxiliary function that manipulates the *ilim* array of non-zero bounds, but in this case we are looking to remove all elements within a set distance from the lower end and update all subsequent values to reflect the new low end. Standard behaviour with IZSUB being called to subtract eight rows can be seen in Figure 3.3. Because subtracting elements from the start of an array was the only manipulation required by the ASD2 Cholesky and LU routines implemented, the structure of the function is limited to that scope, although it is easy to imagine modifications that allow for subtracting/adding from either start/end directions or even combining two *ilim* arrays together.

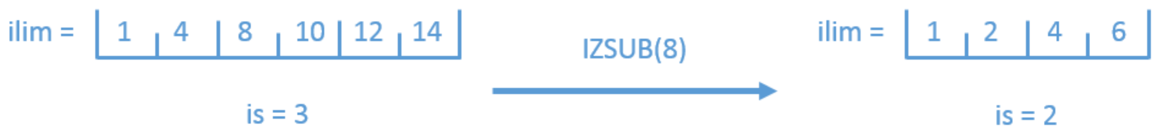


Figure 3.3. Processing an array of block limits with IZSUB.

Implementation

The interface for the function call is IZSUB($ilim, is, bs$) with:

- Inputs:
  - $ilim$: the array of $2 \times is$ limit values
  - is: the number of subblocks
  - bs: the number of rows/columns to subtract from start of array
- Outputs:
  - $ilim$: the array of updated $2 * \times is$ limit values
  - is: the updated number of subblocks
- Algorithm:

```
/* First pass subtracts $bs$ rows from every limit */
**for** $i$ **from** 1 **to** $is$ **do**
    $ilim(2 \times i - 1) = ilim(2 \times i - 1) - bs$
    $ilim(2 \times i) = ilim(2 \times i) - bs$
**end for**
/* Second pass removes negative elements within $ilim$ */
Set $is_{new} = 0$
**for** $i$ **from** 1 **to** $is$ **do**
    /* If current upper limit is positive*/
    **if** $ilim(2 \times i) > 0$ **then**
        $is_{new} + +$
        $ilim(2 \times is_{new} - 1) = MAX(ilim(2 \times i - 1), 1)$
        $ilim(2 \times is_{new}) = ilim(2 \times i)$
    **end if**
**end for**
$is = is_{new}$
**return** $ilim, is$
```

3.2 Block Wrapper Functions

These methods can be described as wrappers around functions like the BLAS that take into account and avoid doing any flops for the zero regions indicated via the *ilim* parameters. These functions usually simply have an outer call for each subblock in which they call the underlying function for the limited bounds. Calling any function with $ilim = \{1, n\}$ and $is = 1$ is the same as calling the original function.

Since these functions operate on the matrix entries separate calls have to be implemented for each data type used, although the algorithms remain the same. Most of the wrapped functions calls like xGEMM or xTRSM offer many variants for the directions of multiplications or the operations ('Normal', 'Transpose', 'Conjugate') but for the purposes of ASD only the combinations used by the LU and Cholesky calls were implemented. These are highlighted for each algorithm.

3.2.1 Subblock Blocked Zero Check - xBZCHK_ILIM

Description

This call allows for the sparsity detection function xBZCHK to be called over multiple sub-blocks, saving the results into the nonzero limit array *ilim* in $2 \times is$ positions. The algorithm assumes the direction (row or column) for which the limits are not calculated is full. This could be potentially expanded in the future to an input *other_dir_ilim* paramater to allow for checking only in the element locations that will be determined to be potentially non-zero.

Most parameters are copied directly and not addressed by the wrapper commands except as parameters for the sub-blocks.

Implementation

The interface for the function calls is xBZCHK_ILIM$(m, n, A, lda, dir, ilim, is)$ with:

- Inputs:
  - m, n: the dimensions of matrix A
  - A: the matrix to be checked
  - lda: the leading dimension of A
  - dir: the direction ('row' or 'column') to check in
  - is: the number of subblocks

- Outputs:
  - *ilim*: the array of $2 \times is$ limit values
- Algorithm:

```
if $dir = row$ then
    $Step = m/is$
    $Curi = 1$
    for $i$ from 1 to $is$ do
        Call BZCHK($Step, n, A(Curi, 1), lda, dir, ilim(2 \times i - 1), ilim(2 \times i)$)
        if $ilim(2 \times i - 1) \neq 0$ then
            $ilim(2 \times i - 1) = ilim(2 \times i - 1) + Curi - 1$
            $ilim(2 \times i) = ilim(2 \times i) + Curi - 1$
        end if
        $Curi = Curi + Step$
    end for
else if $dir = col$ then
    $Step = n/is$
    $Curi = 1$
    for $i$ from 1 to $is$ do
        BZCHK($m, Step, A(1, Curi), lda, dir, ilim(2 \times i - 1), ilim(2 \times i)$)
        if $ilim(2 \times i - 1) \neq 0$ then
            $ilim(2 \times i - 1) = ilim(2 \times i - 1) + Curi - 1$
            $ilim(2 \times i) = ilim(2 \times i) + Curi - 1$
        end if
        $Curi = Curi + Step$
    end for
end if
return $ilim$
```

The implementation has a special branch for $i = is$ to account for m or n not dividing exactly by is where it calls BZCHK with all the remaining rows instead of just *Step*.

3.2.2 Subblock Triangular Solve - xTRSM_ILIM

Description

This function computes the BLAS3 triangular solve ($op(T) \times X = alpha \times B$ or $X \times op(T) = alpha \times B$ where T is triangular and $op(T)$ could be T, T^T or T^H) taking into account the non-zero array limits in *ilim* which describe the nonzero columns in B, and so X, when solving $T \times X = B$, and the nonzero rows in B, and so X, when solving $X \times T = B$. The auxiliary routine serves as a wrapper with all calls being performed by TRSM function calls on submatrices. The algorithm provides implementations for only two cases that cover the three possible call sequences from LU and Cholesky:

- $SIDE =$ '*Left*' for ASD2_GE, and ASD2_PO ('UPPER')
- $SIDE =$ '*Right*' for ASD2_PO ('LOWER')

Using these parameters the function decides which indices the *ilim* array replaces for the sub-block calls as seen in Figure 3.4. The direction influences both the bounds and the matrix locations that are passed on. All other parameters are copied or passed on directly without modification.

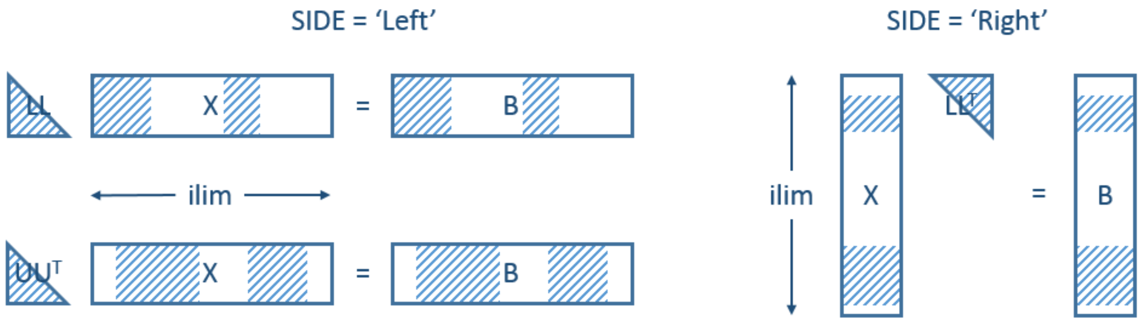


Figure 3.4. Directions *ilim* influences in xTRSM_ILIM.

Implementation

The interface for the function calls is xTRSM_ILIM($SIDE, UPLO, TRANSA, DIAG, m, n, alpha, A, lda, B, ldb, ilim, is$) with:

- Inputs:
  - $SIDE$: either 'Left' for $op(A) \times X = alpha \times B$ or 'Right' for $X \times op(A) = alpha \times B$
  - $UPLO$: 'U' or 'L', determines which triangular part of A will be accessed
  - $TRANSA$: 'N' for $op(A) = A$, 'T' for $op(A) = A^T$, or 'C' for $op(A) = A^H$
  - $DIAG$: 'N' for non-unit and 'U' for unit diagonal of A
  - m, n: the dimensions of matrix B
  - A: the matrix containing the upper or lower triangle
  - $alpha$: the constant multiplier
  - lda: the leading dimension of A

  - B: the right side matrix that gets overwritten with solution X
  - ldb: the leading dimension of B
  - $ilim$: the array of $2 \times is$ limit values
  - is: the number of subblocks

- Outputs:
  - B: the overwritten solution X

- Algorithm:

```
if $SIDE = Left$ then
    for $k$ from 1 to $is$ do
        $k_{low} = ilim(2 \times k - 1), k_{high} = ilim(2 \times k)$
        TRSM($SIDE, UPLO, TRANSA, DIAG, m, k_{high} - k_{low} + 1, alpha, A, lda,$
                $B(1, k_{low}), ldb$)
    end for
else
    for $k$ from 1 to $is$ do
        $k_{low} = ilim(2 \times k - 1), k_{high} = ilim(2 \times k)$
        TRSM($SIDE, UPLO, TRANSA, DIAG, k_{high} - k_{low} + 1, n, alpha, A, lda,$
                $B(k_{low}, 1), ldb$)
    end for
end if
```

3.2.3 Subblock Symmetric Multiply - xSYRK_ILIM

Description

This functions performs the BLAS3 symmetric multiplication ($uplo(C) = alpha \times A \times A^T + beta \times C$ or $uplo(C) = alpha \times A^T \times A + beta \times C$ where $uplo(C)$ describes whether the upper or lower part of C is updated) taking into account the non-zero array limits in *ilim* which describe the nonzero row or columns of A. The routine is mainly a wrapper and consists of calls to SYRK functions on submatrices. There are two parameter case implementations that cover the possible call sequences from Cholesky:

- $TRANS$ = 'T' for ASD2_PO ('UPPER')
- $TRANS$ = 'N' for ASD2_PO ('LOWER')

Regardless of this parameter the *ilim* array replaces the direction and parameter k with the sub-block calls as can be seen in Figure 3.5. However, the direction influences which matrix locations are passed on as the starting position for matrix A. All other parameters are copied or passed on directly without modification.

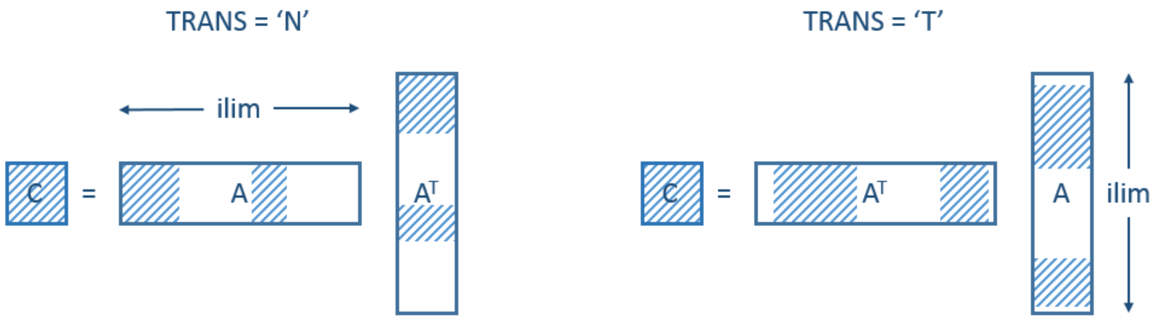


Figure 3.5. Directions *ilim* influences in xSYRK_ILIM.

Implementation

The interface for the function calls is xSYRK_ILIM($UPLO, TRANS, n, k, alpha, A, lda,$ $beta, C, ldc, ilim, is$) with:

- Inputs:
  - $UPLO$: 'U' or 'L', determines which triangular part of C will be updated
  - $TRANS$: 'N' for $op(A) = A$, 'T' for $op(A) = A^T$, or 'C' for $op(A) = A^H$
  - n, k: the dimensions of matrix A
  - $alpha$: the constant multiplier for A
  - A: the matrix to be multiplied
  - lda: the leading dimension of A
  - $beta$: the constant multiplier for C
  - C: the target matrix
  - ldb: the leading dimension of B
  - $ilim$: the array of $2 \times is$ limit values
  - is: the number of subblocks
- Outputs:
  - C: the updated matrix
- Algorithm:

```
if $TRANS = T$ then
    for $i$ from 1 to $is$ do
        $k_{low} = ilim(2 \times i - 1), k_{high} = ilim(2 \times i)$
        SYRK($UPLO, TRANS, n, k_{high} - k_{low} + 1, alpha, A(k_{low}, 1), lda, beta, C, ldc$)
    end for
```

```
else
    for $i$ from $1$ to $is$ do
        $k_{low} = ilim(2 \times i - 1), k_{high} = ilim(2 \times i)$
        SYRK($UPLO, TRANS, n, k_{high} - k_{low} + 1, alpha, A(1, k_{low}), lda, beta, C, ldc$)
    end for
end if
```

3.2.4 Subblock General Matrix Multiply - xGEMM_ILIM

Description

This functions performs the BLAS3 general matrix multiplication ($C = alpha \times op(A) \times op(B) + beta \times C$ where $op(X)$ can be X, X^T or X^H) taking into account the non-zero array limits in *milim* and *nilim*. The routine is mainly a wrapper and consists of calls to GEMM functions on submatrices. There are three parameter case implementations that cover the possible call sequence from LU and Cholesky:

- $TRANSA =$ 'N', $TRANSB =$ 'N' for ASD2_GE
- $TRANSA =$ 'N', $TRANSB =$ 'T'|'C' for ASD2_PO ('UPPER')
- $TRANSA =$ 'T'|'C', $TRANSB =$ 'N' for ASD2_PO ('LOWER')

Using these parameters, the function decides which indices (m, n, k) the $milim, nilim$ arrays replace for each subblock call as seen in Figure 3.6. The direction influences both the bounds and the matrix locations that are passed on. All other parameters are copied or passed on directly without modification.

Implementation

The interface for the function calls is xGEMM_ILIM($TRANSA, TRANSB, m, n, k, alpha, A, lda, B, ldb, beta, C, ldc, milim, mis, nilim, nis$) with:

- Inputs:
  - $TRANSA$: 'N' for $op(A) = A$, 'T' for $op(A) = A^T$, or 'C' for $op(A) = A^H$
  - $TRANSB$: 'N' for $op(B) = B$, 'T' for $op(B) = B^T$, or 'C' for $op(B) = B^H$
  - m, n, k: the dimensions of matrices A, B and C
  - $alpha$: the constant multiplier for $op(A) \times op(B)$
  - A, B: the matrices to be multiplied
  - lda, ldb: the leading dimension of A, B

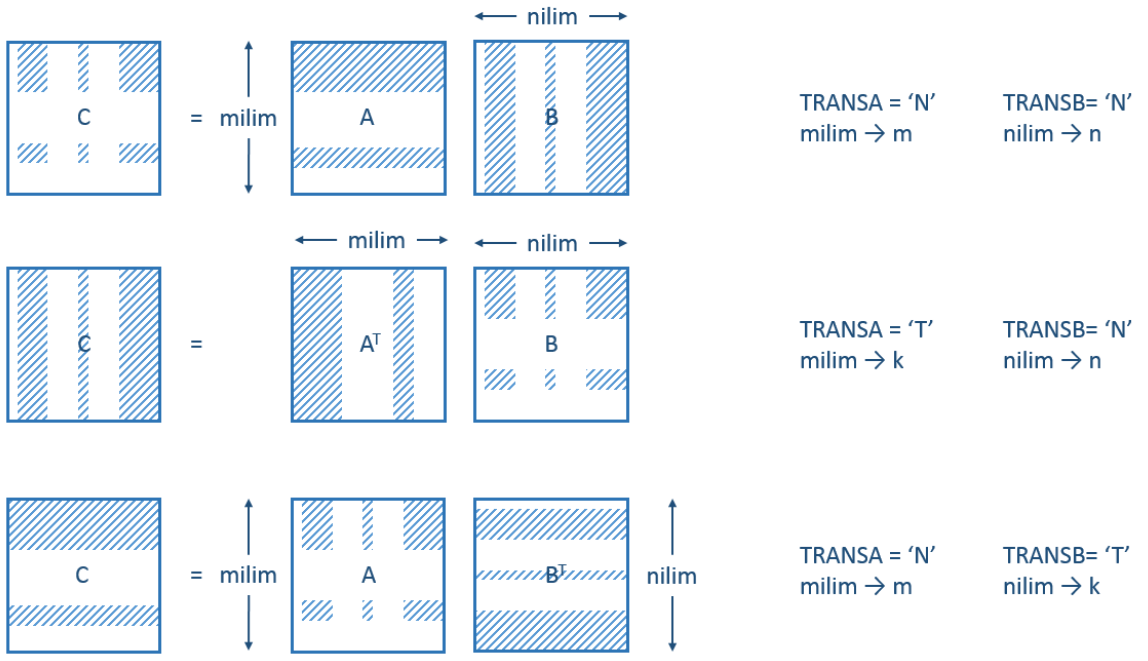


Figure 3.6. Directions $milim, nilim$ influence in xGEMM_ILIM

- $beta$: the constant multiplier for C
- C: the result matrix
- ldc: the leading dimension of C
- $milim, nilim$: the arrays of $2 \times mis, 2 \times nis$ limit values of $op(A)$ and $op(B)$
- mis, nis: the number of subblocks in $op(A)$ and $op(B)$

- Outputs:
  - C: the updated matrix

- Algorithm:

```
if $TRANSA = N$ and $TRANSB = N$ then
    for $i$ from 1 to $mis$ do
        $m_{low} = milim(2 \times i - 1), m_{high} = milim(2 \times i)$
        for $j$ from 1 to $nis$ do
            $n_{low} = nilim(2 \times j - 1), m_{high} = nilim(2 \times j)$
            GEMM($TRANSA, TRANSB, m_{high} - m_{low} + 1, n_{high} - n_{low} + 1, k, alpha,$
                    $A(m_{low}, 1), lda, B(1, n_{low}), ldb, beta, C(m_{low}, n_{low}), ldc$)
        end for
    end for
else if $TRANSA = N$ and $TRANSB = T$ then
```

```
    for $i$ from 1 to $mis$ do
        $m_{low} = milim(2 \times i - 1), m_{high} = milim(2 \times i)$
        for $j$ from 1 to $nis$ do
            $n_{low} = nilim(2 \times j - 1), m_{high} = nilim(2 \times j)$
            GEMM($TRANSA, TRANSB, m_{high} - m_{low} + 1, n, n_{high} - n_{low} + 1, alpha,$
                    $A(m_{low}, n_{low}), lda, B(1, n_{low}), ldb, beta, C(m_{low}, 1), ldc$)
        end for
    end for
else if $TRANSA = T$ and $TRANSB = N$ then
    for $i$ from 1 to $mis$ do
        $m_{low} = milim(2 \times i - 1), m_{high} = milim(2 \times i)$
        for $j$ from 1 to $nis$ do
            $n_{low} = nilim(2 \times j - 1), m_{high} = nilim(2 \times j)$
            GEMM($TRANSA, TRANSB, m, n_{high} - n_{low} + 1, m_{high} - m_{low} + 1, alpha,$
                    $A(m_{low}, 1), lda, B(m_{low}, n_{low}), ldb, beta, C(1, n_{low}), ldc$)
        end for
    end for
end if
```

3.2.5 Subblock BLAS2 LU - xGETF2_ILIM

Description

This function computes the BLAS2 implementation of LU taking into account the nonzero array limits *ilim* which represent the nonzero rows of A. While this function uses a very similar call structure to the regular GETF2 routine it cannot simply call that function with different values and has to implement the BLAS2 operations directly taking into account *ilim*. In practice this means that while the previous auxiliary routines generally had a few parameter checks and then went on to call the wrapped functions for each submatrix, here each operation (pivot selection, matrix update etc.) will be called on each submatrix and combined before continuing on to the next step.

The algorithm also ensures that at least the $min(m, n)$ by n top values are checked regardless of the non-zero array limits in *ilim* to ensure the pivoting array values in *ipiv* give the same answers as the regular $xGETF2$ implementation. Another influence of the pivot selection that is eliminated is the possibility of multiple pivots of the same magnitude. Via using ISAMAX's property of returning the first element of largest magnitude and the choice for the checks we can ensure that the pivoting order is preserved. The overall diagram of the steps for $xGETF2\_ILIM$ can be seen in Figure 3.7.

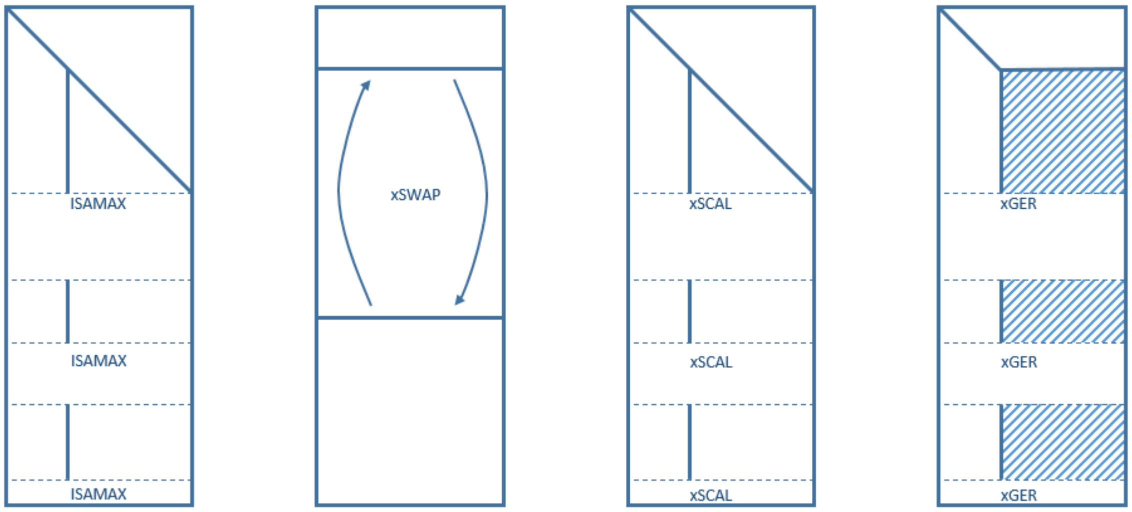


Figure 3.7. Steps for xGETF2_ILIM.

Implementation

The interface for the function calls is xGETF2_ILIM($m, n, A, lda, ipiv, ilim, is, info$) with:

- Inputs:
  - m, n: the dimensions of matrix A
  - A: the matrix to be factorized
  - lda: the leading dimension of A
  - $ipiv$: the pivoting array
  - $ilim$: the array of $2 \times is$ limit values
  - is: the number of subblocks in A
  - $info$: an error parameter for incomplete factorizations
- Outputs:
  - A: the factorized matrix
- Algorithm:

```
/* Compute machine min */
$sfmin$ = SLAMCH('$S$')
**for** $j$ **from** 1 **to** $min\{m, n\}$ **do**
    /* Find pivot */
```

```
$j_p = j - 1$+ISAMAX($min\{m,n\} - j + 1, A(j,j), 1$)
$curmax = |A(j_p, j)|$ **and** $k_p = j_p$
**for** $k$ **from** $1$ **to** $is$ **do**
    $k_{low} = ilim(2 \times k - 1), k_{high} = ilim(2 \times k)$
    **if** $k_{low} > min\{m,n\}$ **then**
        $k_p = k_{low} - 1$+ISAMAX($k_{high} - k_{low} + 1, A(k_{low}, j), 1$)
    **else if** $k_{high} > min\{m,n\}$ **then**
        $k_p = min\{m,n\}$+ISAMAX($k_{high} - min\{m,n\}, A(k_{low}, j), 1$)
    **end if**
    **if** $|A(k_p, j)| > curmax$ **then**
        $j_p = k_p$ **and** $curmax = |A(k_p, j)|$
    **end if**
**end for**
$ipiv(j) = j_p$
**if** $A(j_p, j) \neq 0$ **then**
    /* Switch pivot row */
    **if** $j_p \neq j$ **then**
        xSWAP($n, A(j,1), lda, A(j_p,1), lda$)
    **end if**
    /* Scale column elements */
    **if** $j < m$ **then**
        **if** $|A(j,j)| \geq sfmin$ **then**
            /* numerically stable use SCAL */
            **for** $k$ **from** $1$ **to** $is$ **do**
                $k_{low} = ilim(2 \times k - 1), k_{high} = ilim(2 \times k)$
                **if** $k_{low} > j$ **then**
                    xSCAL($k_{high} - k_{low} + 1, 1/A(j,j), A(k_{low}, j), 1$)
                **else if** $k_{high} > j$ **then**
                    xSCAL($k_{high} - j, 1/A(j,j), A(j+1, j), 1$)
                **end if**
            **end for**
        else
            /* Inverse of pivot too close to machine limit use for loop */
            **for** $k$ **from** $1$ **to** $is$ **do**
                $k_{low} = ilim(2 \times k - 1), k_{high} = ilim(2 \times k)$
                **if** $k_{low} > j$ **then**
                    **for** $i$ from $k_{low}$ to $k_{high}$ **do**
                        $A(i,j) = A(i,j)/A(j,j)$
                    **end for**
                **else if** $k_{high} > j$ **then**
                    **for** $i$ from $j+1$ to $k_{high}$ **do**
                        $A(i,j) = A(i,j)/A(j,j)$
                    **end for**
                **end if**
            **end for**
            end if
        end if
    else if $info = 0$ then
        $info = j$
    end if
    /* Compute xGER updates to rest of A */
    if $j < min\{m, n\}$ then
        for $k$ from 1 to $is$ do
            $k_{low} = ilim(2 \times k - 1), k_{high} = ilim(2 \times k)$
            if $k_{low} > j$ then
                xGER($k_{high} - k_{low} + 1, n - j, -1, A(k_{low}, j), 1, A(j, j + 1), lda,$
                        $A(k_{low}, j + 1), lda$)
            else if $k_{high} > j$ then
                xGER($k_{high} - j, n - j, -1, A(j + 1, j), 1, A(j, j + 1), lda, A(j + 1, j + 1), lda$)
            end if
        end for
    end if
end for
```

Chapter 4

LU and Cholesky ASD Factorizations

There are three "Automatic Structure Detection" algorithms that each can serve as a replacement for the corresponding current LAPACK implementations. Each subsequent algorithm below (ASD1, ASD2 and ASD3) has increasing overhead, higher complexity and more potential speedup when compared to the corresponding LAPACK implementation. The ASD algorithms consist of:

- Profile (ASD1) - A simple algorithm that checks for trailing zero rows/columns and avoids doing the work for those parts similar to one used in *xGEQRF* [8]
- Block Sparse (ASD2) - An algorithm that performs the same checks that ASD1 does but on multiple subblocks within a panel, as opposed to the whole panel
- Full Sparsity (ASD3) - An algorithm that on detecting high fill-in permutes the matrix to reduce fill-in then continues with ASD2

While each higher level encompasses the checks of the previous and therefore detects the same or better sparsity structure, the best routine, from a performance point of view, could be any of the three algorithms because of the competing values of overhead versus sparsity exploitation. It should be noted that as the matrix size increases the overhead differences between the algorithms decreases as the flop costs start to dominate.

A general aspect of all the ASD algorithms is that most of the actual work for implementing the sparsity detection and exploitation is hidden behind easy to understand auxiliary functions and detection routines. Here we differ from previous structure detection algorithm present in *xGEQRF* since most changes are presented as either: a) simple parameter changes to the existing routines, b) calls to detection, c) calls to integer limit routines or d)wrapper calls to _ILIM auxiliary functions. This modular approach can serve as a good potential first step for future improvements to individual components and also to other factorizations.

4.1 Profile (ASD1)

4.1.1 Description

The first Automatic Sparsity Detection algorithm is named “Profile” or ASD1 for short. It relies on a simple check on the furthest row/column and only continues the checks if it encounters zeros. For a full matrix the number of checks that this algorithm adds is $O(n/n_b)$ where, n is the size of the matrix and n_b is the blocking factor used by LAPACK. It uses the single detection routine $xBZCHK$ and no other auxiliary routines. This simple extension means that no extra workspace is required by the function and that the auxiliary function calling order within LU and Cholesky remains unaltered except for the addition of the $xBZCHK$ calls. Because of the simplicity of these checks even small amounts of zeros within the input matrix are able to be exploited and provide a small benefit.

Large speedups however can be observed on a series of common matrices that that are particularly well suited for showing the benefits the ASD profile detection:

- banded - for the general matrix LU and Cholesky the banded matrices can serve to lower the cost from $O(n^3)$ to $O(n \times b^2)$
- profile - this example of a ‘special’ banded matrix that does not fit into the normal LAPACK definitions and can be exploited for faster performance over LAPACK
- Cuthill-McKee - these reorderings of sparse matrices create a structure with varying profile and low bandwidth b which can represent a very high potential speedup.

At a general level all ASD1 functions also do few checks per panel with only 1-3 calls to the ”Block Sparse” $xBZCHK$ function on submatrices of A. The locations that are checked in the ASD1 calls can be seen in Figure 4.1.

4.1.2 Implementations

For the ‘Profile’ Automatic Sparsity Detection (ASD1) implementations were provided for four LAPACK routines: $GETRF$, $POTRF$, $GBTRF$, $PBTRF$ each with small variations on the general algorithm of running checks on panels and then using the updated parameters for the function calls.

Because the checks are done within the factorization only there are no requirement for any interface changes nor any extra workspace requirements, which will allow these routines to be directly integrated into LAPACK.

In the algorithms presented below the blue colored lines represent additional calls or parameter changes with all other parts of the algorithm remaining the same as default LAPACK.

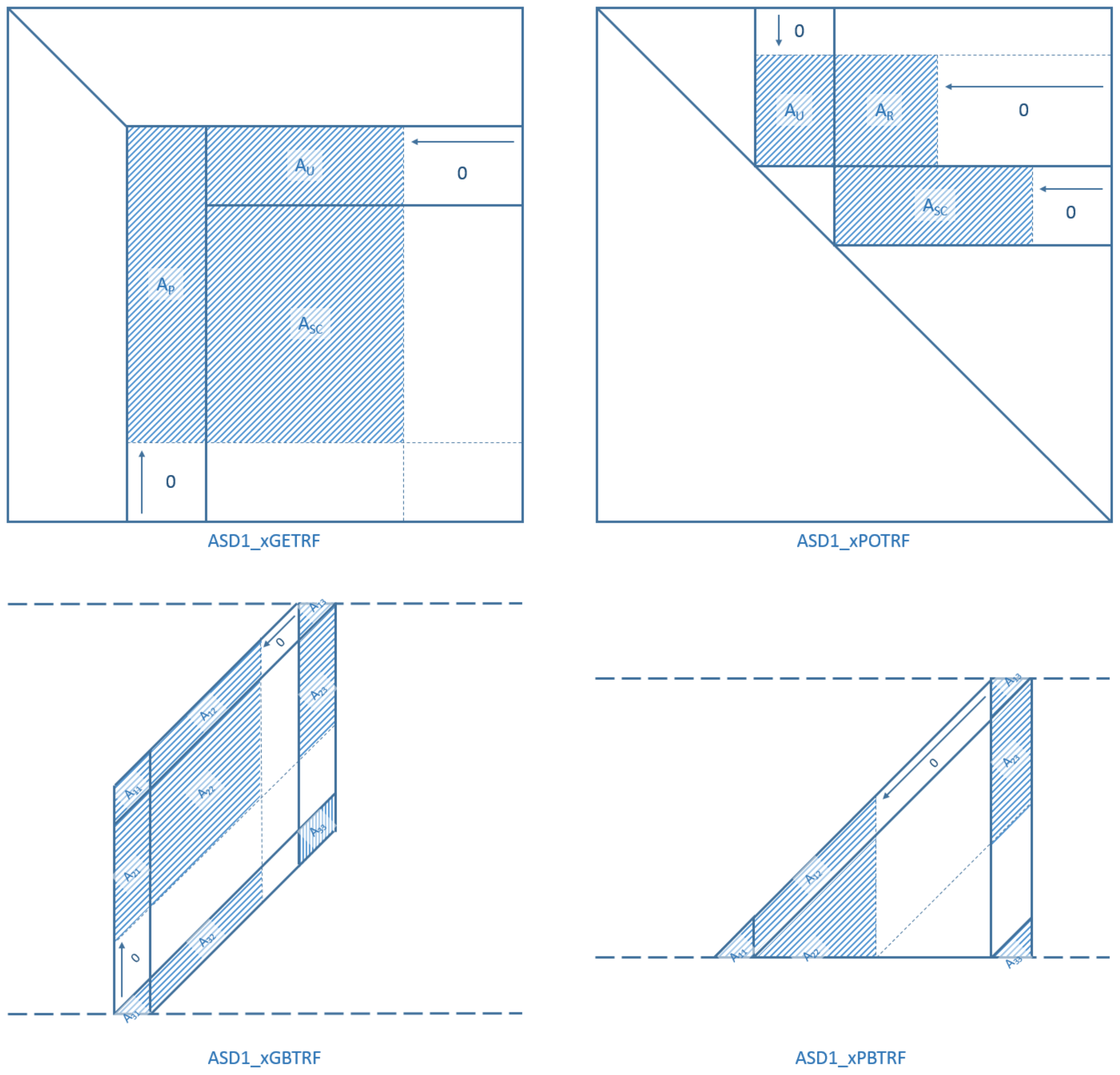


Figure 4.1. ASD1 sparsity exploitation parts of A.

General full LU ($ASD1\_xGETRF$)

The implementation below can provide a high speedup for matrices with low profiles but because it still has to perform the $O(n^2)$ checks for those matrices it usually will not outperform the banded LU call xGBTRF for very low bandwidths b. This implementation does 2 checks on the A_P and A_U parts and limits 3 function calls on A_P, A_U, A_{SC} via their parameters as can be seen in Figure 4.2.

An aspect to point out is that the choice was made to implement this version without limiting the row swaps which still apply over the entire row. This was a compromise since the

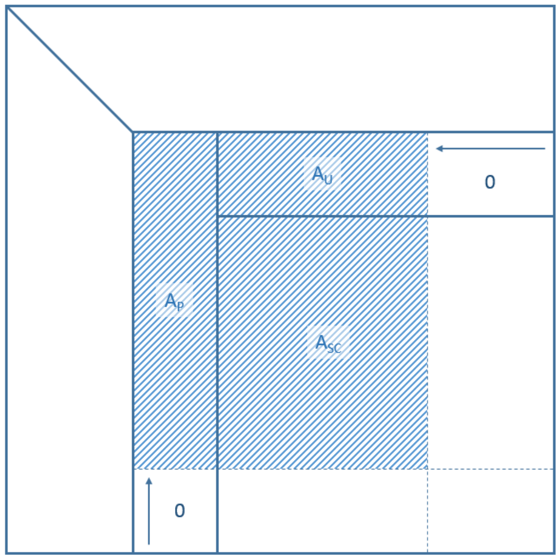


Figure 4.2. ASD1_xGETRF factorization step.

only options to prevent this behaviour was either keeping some extra workspace to maintain the values from previous panel checks or to do new checks on the elements with both options ending up being more costly than simply allowing the full swaps for the rows.

Interface ASD1_xGETRF($m, n, A, lda, ipiv, info$) with:

- Inputs:
  - m, n: the dimensions of matrix A
  - A: the matrix to be factorized
  - lda: the leading dimension of A
  - $info$: an error parameter for incomplete factorizations
- Outputs:
  - $ipiv$: the pivoting array
  - A: the factorized matrix
- Algorithm:

```
Get $n_b$ from $ILAENV$
**if** $n_b \leq 1$ or $n_b \geq min\{m, n\}$ **then**
    Call BLAS2 version xGETF2($m, n, A, lda, ipiv, info$)
**else**
    **for** $j$ **from** 1 **to** $min(m, n)$ **by** $n_b$ **do**
        $j_b = min(n_b, min\{m, n\} - j + 1)$
```

```
            xBZCHK($m - j + 1, j_b, A(j,j), lda, row, m_{low}, m_{high}$)
            xGETF2($m_{high}, n, A(j,j), lda, ipiv(j), iinfo$)
            if $info = 0$ and $iinfo > 0$ then
                $info = iinfo + j - 1$
            end if
            for $i$ from $j$ to $min(m, j + j_b - 1)$ do
                $ipiv(i) = ipiv(i) + j - 1$
            end for
            xLASWP($j - 1, A, lda, j, j + j_b - 1, ipiv, 1$)
            if $j + j_b < n$ then
                xLASWP($n - j - jb + 1, A(1, j + jb), lda, j, J + j_b - 1, ipiv, 1$)
                xBZCHK($j_b, n - j - jb, A(j, j + j_b), lda, col, n_{low}, n_{high}$)
                xTRSM($Left, Lower, No\ transpose, Unit, j_b, n_{high}, 1, A(j,j), lda,$
                          $A(1, 1 + j_b), lda)$)
                if $j_b < m_{high}$ then
                    xGEMM($No\ transpose, No\ transpose, m_{high} - j_b, n_{high}, j_b, -1$
                          $, A(j + j_b, j), lda, A(j, j + j_b), lda, 1, A(j + j_b, j + j_b), lda$)
                end if
            end if
        end for
    end if
```

General full Cholesky ($ASD1\_xPOTRF$)

An important difference between the Cholesky and the LU implementations is the type of algorithm used and how it relates to what is kept in the part of the matrix remaining to be factorized.

For LU which used a ‘right-looking’ algorithm this represents the current updated Schur Complement and the steps taken involve:

- factorizing the panel
- triangular solve on U
- updating the Schur Complement

For Cholesky which uses a ‘left-looking’ algorithm the remaining part of the matrix is kept to represent the unaltered part of A, therefore its steps involve (assuming an ’Upper’ Cholesky factorization):

- updating the j_b square block
- factorizing the block
- updating the U factor

- triangular solve on U

Because of this difference in algorithm 3 checks are required on the submatrices A_U, A_R, A_{SC} (shown in Figure 4.3) which are also the 3 matrices that are limited in the function calls used by the Cholesky algorithm.

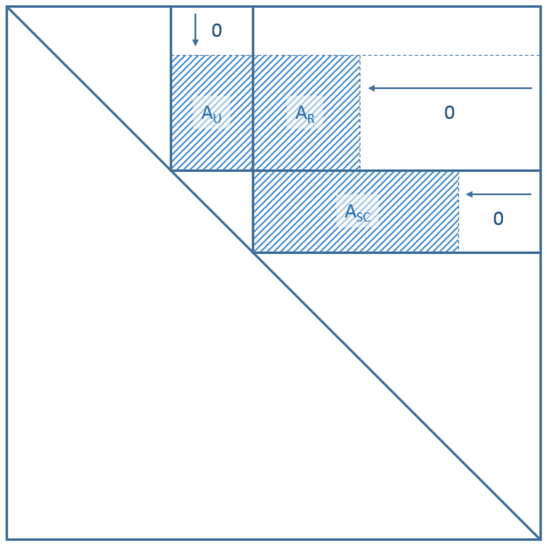


Figure 4.3. ASD1_xPOTRF ‘Upper’ factorization step.

Interface ASD1_xPOTRF$(uplo, n, A, lda, info)$ with:

- Inputs:
  - $uplo$: ‘U’ for $A = U^T \times U$ factorization or ‘L’ for $A = L \times L^T$ factorization
  - n: the dimensions of matrix A
  - A: the matrix to be factorized
  - lda: the leading dimension of A
  - $info$: an error parameter for incomplete factorizations
- Outputs:
  - A: the factorized matrix
- Algorithm:

```
Get $n_b$ from $ILAENV$
if $n_b \leq 1$ or $n_b \geq n$ then
    Call BLAS2 version xPOTF2$(uplo, n, A, lda, info)$
else
```

```
if uplo = Upper then
    for $j$ from 1 to $n$ by $n_b$ do
        $j_b = min\{n_b, n - j + 1\}$
        xBZCHK($j - 1, j_b, A(1, j), lda, row, m_{low}, m_{high}$)
        if $m_{low} > 0$ then
            xSYRK($Upper, Transpose, j_b, j - m_{low}, -1, A(m_{low}, j), lda, 1,$
                        $A(j, j), lda$)
        end if
        xPOTF2($Upper, jb, A(j, j), lda, info$)
        if $info \neq 0$ then
            $info+ = j - 1$ and return
        end if
        xBZCHK($j - m_{low}, n - j - j_b + 1, A(m_{low}, j + j_b), lda, col, n_{low}, n_{high}$)
        if $(m_{low} > 0)$ and $(n_{high} > 0)$ then
            xGEMM($Transpose, No\ transpose, j_b, n_{high}, j - m_{low}, -1, A(m_{low}, j),$
                        $lda, A(m_{low}, j + j_b), lda, 1, A(j, j + j_b), lda$)
        end if
        xBZCHK($j_b, n - j - j_b + 1, A(j, j + j_b), lda, col, n_{low}, n_{high}$)
        if $n_{high} > 0$ then
            xTRSM($Left, Upper, Transpose, Nonunit, j_b, n_{high}, 1, A(j, j), lda,$
                        $A(j, j + j_b), lda$)
        end if
    end for
else
    /* Algorithm is mirrored for lower */
    for $j$ from 1 to $n$ by $n_b$ do
        $j_b = min\{n_b, n - j + 1\}$
        xBZCHK($j_b, j - 1, A(j, 1), lda, col, n_{low}, n_{high}$)
        if $n_{low} > 0$ then
            xSYRK($Lower, No\ transpose, j_b, j - n_{low}, -1, A(j, n_{low}), lda, 1,$
                        $A(j, j), lda$)
        end if
        xPOTF2($Lower, jb, A(j, j), lda, info$)
        if $info \neq 0$ then
            $info+ = j - 1$ and return
        end if
        xBZCHK($n - j - j_b + 1, j - n_{low}, A(j + j_b, n_{low}), lda, row, m_{low}, m_{high}$)
        if $(n_{low} > 0)$ and $(m_{high} > 0)$ then
            xGEMM($No\ transpose, Transpose, m_{high}, j_b, j - n_{low}, -1, A(j, n_{low}),$
                        $lda, A(j + j_b, n_{low}), lda, 1, A(j + j_b, j), lda$)
        end if
        xBZCHK($n - j - j_b + 1, j_b, A(j + j_b, j), lda, row, m_{low}, m_{high}$)
        if $m_{high} > 0$ then
            xTRSM($Right, Lower, Transpose, Nonunit, m_{high}, j_b, 1, A(j, j), lda,$
                        $A(j + j_b, j), lda$)
            end if
        end for
    end if
end if
```

General band LU ($ASD1\_xGBTRF$)

The banded LU algorithm is based on a compressed banded format Ab that maps subparts of columns of A to columns of Ab and diagonals of A to rows of Ab. A representation of this can be seen in Figure 4.4 with the area encompassed by the dotted line representing the values Ab stores and the area encompassed by the thin solid line representing the values worked on in a step in A and in Ab.

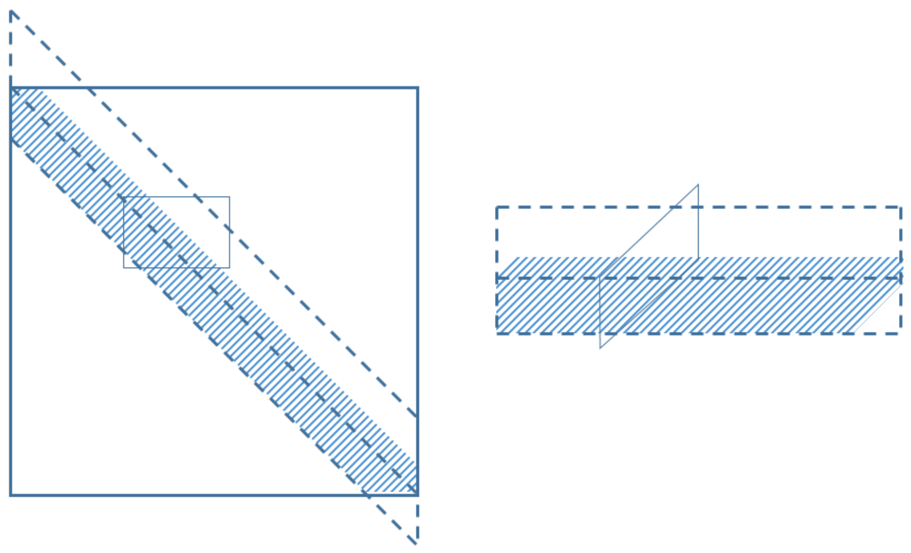


Figure 4.4. Matrix format: *left)* dense A representation *right)* banded Ab representation

Due to the format of the banded algorithm, while there are still 2 checks that are computed, the check on the panel was not used in the factorization on the panel. The blocked zero checks are done on A_{21} and A_{12} and this information used to limit the parameters in the 4 updates to A_{12}, A_{22}, A_{32} and A_{23} as seen in Figure 4.5.

Interface ASD1_xGBTRF($m, n, kl, ku, Ab, ldab, ipiv, info$) with:

- Inputs:
  - m, n: the dimensions of matrix A
  - kl, ku: the lower and upper bandwidths
  - Ab: the matrix to be factorized in banded storage format

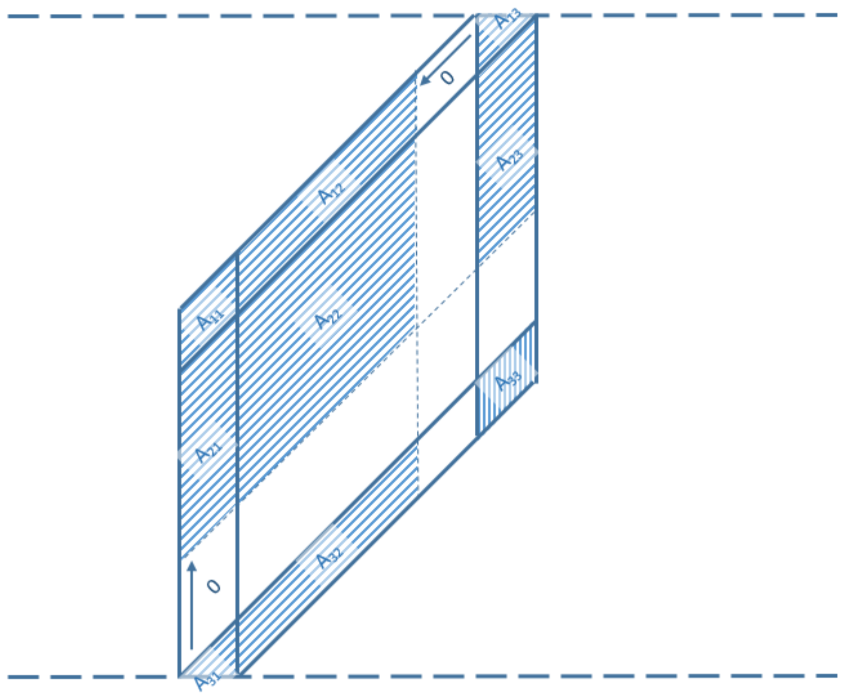


Figure 4.5. ASD1_xGBTRF factorization step.

  - $ldab$: the leading dimension of Ab (the banded storage of A)
  - $info$: an error parameter for incomplete factorizations

- Outputs:
  - $ipiv$: the pivoting array
  - Ab: the factorized matrix

- Algorithm:

```
$kv = kl + ku$
Get $n_b$ from $ILAENV$
**if** $n_b \leq 1$ or $n_b \geq kl$ **then**
    Call BLAS2 version xGBTF2($m, n, kl, ku, Ab, ldab, ipiv, info$)
**else**
    Zero out the superdiagonal elements of $work_{13}$
    Zero out the subdiagonal elements of $work_{31}$
    Set $Ab$ fill-in elements in columns $ku + 2$ to $kv$ to zero
    $j_u = 1$
    **for** $j$ **from** 1 **to** $min\{m, n\}$ **by** $n_b$ **do**
        $j_b = min\{n_b, min\{m, n\} - j + 1\}$
        $i_2 = min\{kl - j_b, m - j - j_b + 1\}$
        $i_3 = min\{j_b, m - j - kl + 1\}$
        xBZCHK($i_2, j_b, Ab(kv+1+j_b, j), dlab-1, row, i_{low2}, i_{high2}$)
        Do BLAS2 Factorization of panel $[A_{11}, A_{21}, A_{31}]^T$
        if $j + j_b \leq n$ then
            $j_2 = min\{j_u - j + 1, kv\} - j_b$
            $j_3 = max\{0, j_u - j - kv + 1\}$
            Apply $ipiv$ permutations to $[A_{12}, A_{22}, A_{32}]^T$ with $xLASWP$
            Adjust pivot indices $ipiv$by subtracting $j - 1$
            Apply $ipiv$ permutations to $[A_{13}, A_{23}, A_{33}]^T$
            xBZCHK($j_b, j_2, Ab(kv+1-j_b, j+j_b), ldab-1, col, j_{low2}, j_{high2}$)
            if $j_{high2} > 0$ then
                xTRSM($Left, Lower, No\ transpose, Unit, j_b, j_{high2}, 1, Ab(kv+1, j),$
                        $ldab-1, Ab(kv+1-j_b, j+j_b), ldab-1$)
                if $i_{high2} > 0$ then
                    xGEMM($No\ transpose, No\ transpose, i_{high2}, j_{high2}, j_b, -1,$
                            $Ab(kv+1+j_b, j), ldab-1, Ab(kv+1-j_b, j+j_b), ldab-1, 1,$
                            $Ab(kv+1, j+jb), ldab-1$)
                end if
                if $i_3 > 0$ then
                    xGEMM($No\ transpose, No\ transpose, i_3, j_{high2}, j_b, -1, work_{31},$
                            $ldwork, Ab(kv+1-j_b, j+j_b), ldab-1, 1$
                            $, Ab(kv+kl+1-j_b, j+jb), ldab-1$)
                end if
            end if
            if $j_3 > 0$ then
                Copy $A_{13}$ into $work_{13}$
                xTRSM($Left, Lower, No\ transpose, Unit, j_b, j_3, 1, Ab(kv+1, j), ldab-1,$
                        $work_{13}, ldwork$)
                if $i_{high2} > 0$ then
                    xGEMM($No\ transpose, No\ transpose, i_{high2}, j_3, j_b, -1,$
                            $Ab(kv+1+j_b, j), ldab-1, work_{13}, ldwork, 1,$
                            $Ab(1+jb, j+kv), ldab-1$)
                end if
                if $i_3 > 0$ then
                    xGEMM($No\ transpose, No\ transpose, i_3, j_3, j_b, -1, work_{31}, ldwork,$
                            $work_{13}, ldwork, 1, Ab(1+kl, j+kv), ldab-1$)
                end if
                Copy lower triangle of $work_{13}$ into $A_{13}$
            end if
        else
            Adjust pivot indices $ipiv$by subtracting $j - 1$
        end if
        Partially undo pivoting to restore upper triangular form of $A_{31}$
    end for
end if
```

General band Cholesky ($ASD1\_xPBTRF$)

The banded Cholesky algorithm returns to the 'right-looking' implementation that updates the Schur complement and therefore we are able to use only a single check on A_{12} (for an 'Upper' Cholesky factorization). This check is then used to limit the computations on A_{12}, A_{22} and A_{23}.

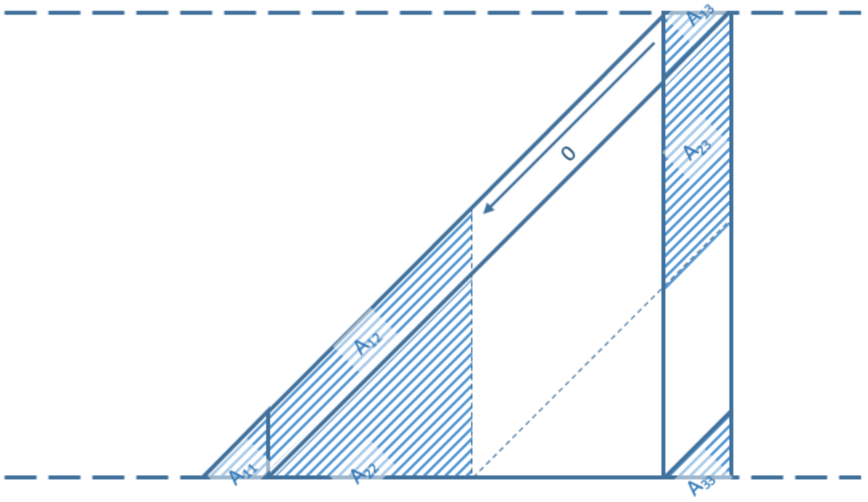


Figure 4.6. ASD1_xPBTRF 'Upper' factorization step.

Interface ASD1_xPBTRF($uplo, n, kd, Ab, ldab, info$) with:

- Inputs:
  - $uplo$: 'U' for $A = U^T \times U$ factorization or 'L' for $A = L \times L^T$ factorization
  - n: the dimensions of matrix A
  - kd: the bandwidth of the matrix
  - Ab: the matrix to be factorized in banded storage format
  - $ldab$: the leading dimension of Ab (the banded storage of A)
  - $info$: an error parameter for incomplete factorizations
- Outputs:
  - Ab: the factorized matrix
- Algorithm:

```
Get $n_b$ from $ILAENV$
**if** $n_b \leq 1$ or $n_b > kd$ **then**
    Call BLAS2 version xPBTRF($uplo, n, kd, Ab, ldab, info$)
**else**
```

```
if uplo = Upper then
    Zero out upper triangle of $work$ matrix
    for $i$ from 1 to $n$ by $n_b$ do
        $i_b = min\{n_b, n - i - 1\}$
        xPOTF2($uplo, i_b, Ab(kd + 1, i), ldab - 1, ii$)
        if $ii \neq 0$ then
            return
        end if
        if $i + i_b \leq n$ then
            $i_2 = min\{kd - i_b, n - i - i_b + 1\}$
            $i_3 = max\{i_b, n - i - kd + 1\}$
            xBZCHK($i_b, i_2, Ab(kd + 1 - i_b, i + i_b), ldab - 1, col, i_{low2}, i_{high2}$)
            if $i_{high2} > 0$ then
                xTRSM($Left, Upper, Transpose, Nonunit, i_b, i_{high2}, 1, Ab(kd + 1, i),$
                          $ldab - 1, Ab(kd + 1 - i_b, i + i_b), ldab - 1$)
                $xSYRK(Upper, Transpose, i_{high2}, i_b, -1, Ab(kd + 1 - i_b, i + i_b),$
                          $ldab - 1, 1, Ab(kd + 1, i + i_b), ldab - 1)$
            end if
            if $i_3 > 0$ then
                Copy lower triangle of $A_{13}$ into $work$
                xTRSM($Left, Upper, Transpose, Nonunit, i_b, i_3, 1, Ab(kd + 1, i)$
                          $, ldab - 1, work, ldwork$)
                if $i_{high2} > 0$ then
                    xGEMM($Transpose, No\ transpose, i_{high2}, i_3, i_b, -1,$
                              $Ab(kd + 1 - i_b, i + i_b), ldab - 1, work, ldwork, 1,$
                              $Ab(1 + i_b, i + kd), ldab - 1$)
                end if
                xSYRK($Upper, Transpose, i_3, i_b, -1, work, ldwork, 1,$
                          $Ab(1 + kd, i + kd), ldab - 1$)
                Copy lower triangle of $work$ into $A_{13}$
            end if
        end if
    end for
else
    /* Algorithm is mirrored for lower */
    Zero out lower triangle of $work$ matrix
    for $i$ from 1 to $n$ by $n_b$ do
        $i_b = min\{n_b, n - i - 1\}$
        xPOTF2($uplo, i_b, Ab(1, i), ldab - 1, ii$)
        if $ii \neq 0$ then
            return
        end if
        if $i + i_b \leq n$ then
            $i_2 = min\{kd - i_b, n - i - i_b + 1\}$
            $i_3 = max\{i_b, n - i - kd + 1\}$
```

```
                xBZCHK($i_2, i_b, Ab(1+i_b, i), ldab-1, row, i_{low2}, i_{high2}$)
                if $i_{high2} > 0$ then
                    xTRSM($Right, Lower, Transpose, Nonunit, i_{high2}, i_b, 1, Ab(1, i),$
                              $ldab-1, Ab(1+i_b, i), ldab-1$)
                    xSYRK($Lower, NoTranspose, i_{high2}, i_b, -1, Ab(1+i_b, i),$
                              $ldab-1, 1, Ab(1, i+i_b), ldab-1$)
                end if
                if $i_3 > 0$ then
                    Copy upper triangle of $A_{31}$ into $work$
                    xTRSM($Right, Lower, Transpose, Nonunit, i_3, i_b, 1, Ab(1, i),$
                              $ldab-1, work, ldwork$)
                    if $i_{high2} > 0$ then
                        xGEMM($No\ transpose, Transpose, i_3, i_{high2}, i_b, -1,$
                                  $work, ldwork, Ab(1+i_b, i), ldab-1, 1,$
                                  $Ab(1+kd-i_b, i+i_b), ldab-1$)
                    end if
                    xSYRK($Lower, No\ transpose, i_3, i_b, -1, work, ldwork, 1,$
                              $Ab(1, i+kd), ldab-1$)
                    Copy upper triangle of $work$ into $A_{31}$
                end if
            end if
        end for
    end if
end if
```

4.1.3 Considerations

Since the ASD1 implementations do not change any of the pivoting choices there is no significant change in the error norms for the result with the only differences coming from the general floating-point non-reproducibility of BLAS and LAPACK, caused by floating point sums computed in possibly different orders.

There is a very subtle change in the output however for cases where NaN's are created or in cases where NaNs are given as input with regards to their propagation in the output. For fully dense matrices the behaviour for NaNs will remain the same as default LAPACK. For matrices with sparsity however, because the functions ignore zero rows/columns, in some cases where we have $0 * NaN$, the values will not update to NaN in the subsequent calls in the matrix. If a zero value is present in a non-zero row/column and we compute $0 * NaN$ we will get the default NaN propagation. This can be seen in an example in Figure 4.7. Inf values have analogous properties.

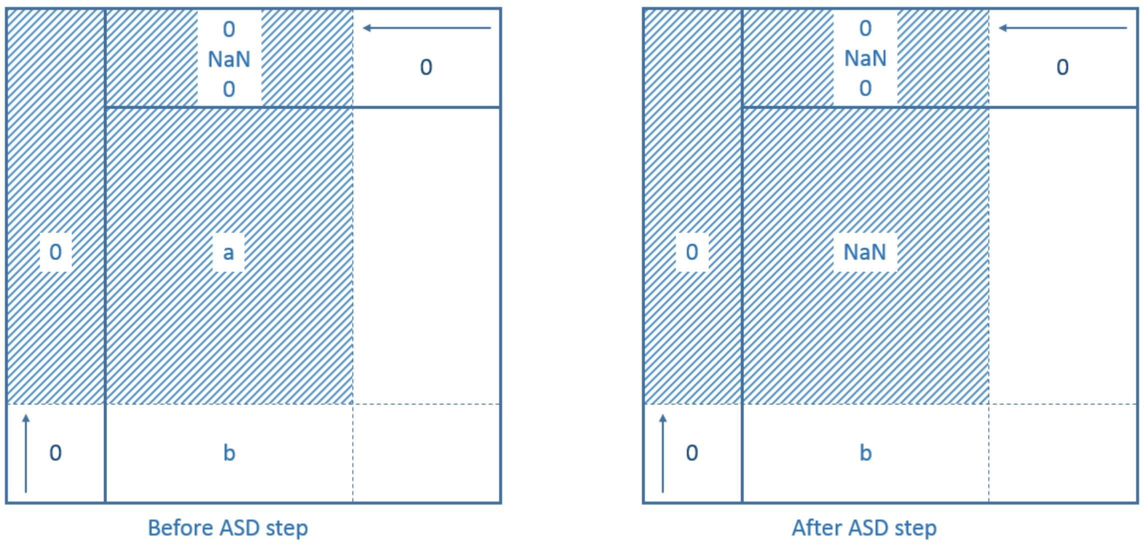


Figure 4.7. ASD1 NaN propagation.

4.2 Sparse Block (ASD2)

4.2.1 Description

The second Automatic Sparsity Detection algorithm is named “Sparse Block” or ASD2 for short. It expands the checks from ASD1 and replaces the single check to determine one limiting value with an array of non-zero block limits which is stored in $ilim$. This array is then used to limit all further calls on the matrices that reference those locations to only the nonzero parts. In particular this type of check can allow the codes to avoid doing work for interior non-zero rows/columns of a submatrix. A representation of this can be seen in Figure 4.8.

In order to determine how many sub-blocks to check, and how large consequently the $ilim$ array will be, the ASD2 algorithm first does a randomized sparse check on the entire matrix A which determines an apparent sparsity fraction $fracnz$. This fraction is then used to determine the number of subblocks for each panel using xBZCHK_ILIM. In order to minimize the overhead and the size of the blocks used, the results on neighbouring blocks are combined in order to create the most compact representation for the non-zero block limits stored in $ilim$.

Because of these extra steps the overhead associated with the ASD2 algorithms is larger. It consists of an initial $O(n)$ cost for the sparsity check and then of another $is \times O(n/n_b)$ checks where, n is the size of the matrix, n_b is the blocking factor used by LAPACK, and is is the number of blocks used.

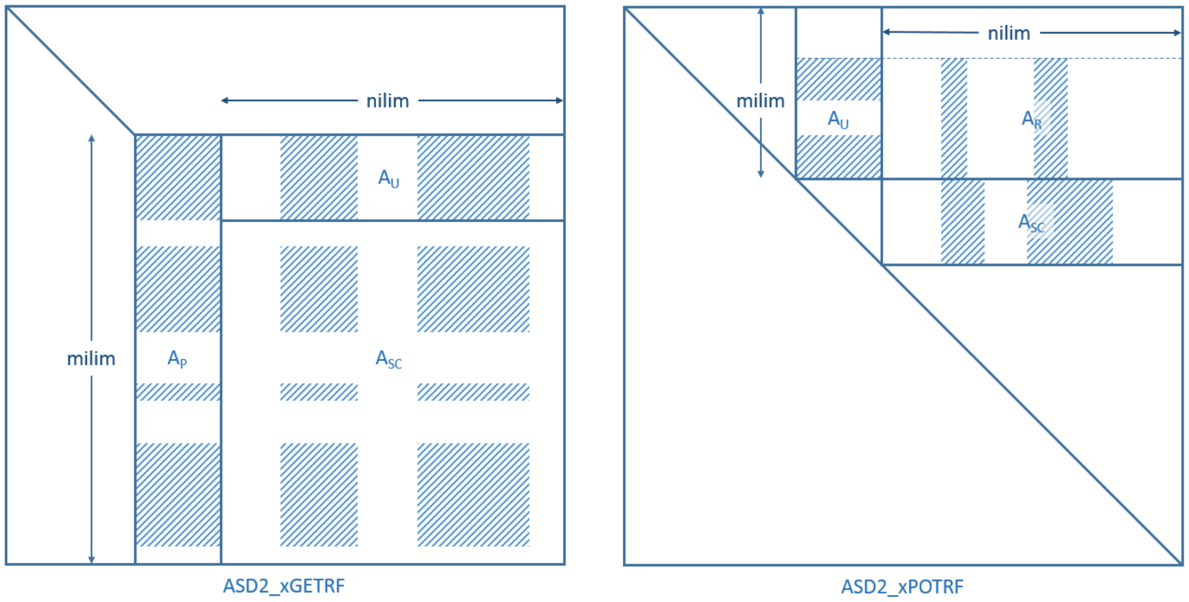


Figure 4.8. ASD2 sparsity exploitation parts of A.

Because of the need to keep an array of the nonzero block limits an extra integer workspace proportional to the dimensions of the matrix is required. Alternatively to avoid changing the subroutine interface and workspace, we could use an integer array of fixed length. While the sequence of function calls performed by the algorithm changes almost fully, the new routines represent either additional sparsity calls ($xSZCHK, xBZCHK$), non-zero array operations or wrappers to the previously called simple LAPACK routines.

Not accounting for the extra overhead initially present for small matrices the ASD2 algorithm maintains the same level of performance for the types of matrices addressed by ASD1. It also expands the types of common matrices that it can provide speedups to the following:

- arrow - banded or low profile matrices that have a few dense row/columns on their last positions
- Nested-Dissection - these reorderings of sparse matrices create a structure with many more dense rows/columns in the ending parts of the recursive format

In general the ASD2 algorithm performs well on any matrix where the structure has most sparsity present in the top-left of the matrix and subsequently does not incur a high fill-in.

4.2.2 Implementations

For the ‘Sparse block’ Automatic Sparsity Detection (ASD2) implementations were provided for two LAPACK routines

General full LU (ASD2_SGETRF)

Interface ASD2_xGETRF($m, n, A, lda, ipiv, info$) with:

- Inputs:
  - m, n: the dimensions of matrix A
  - A: the matrix to be factorized
  - lda: the leading dimension of A
  - $info$: an error parameter for incomplete factorizations
- Outputs:
  - $ipiv$: the pivoting array
  - A: the factorized matrix
- Algorithm:

```
/* Setting minimum subblock size to 16 sets max size for ilim to 2*n/16 */
Allocate integer space for $milim(m/8), nilim(n/8)$
Get $n_b$ from $ILAENV$
**if** $n_b \leq 1$ or $n_b \geq min\{m, n\}$ **then**
    Call BLAS2 version xGETF2($m, n, A, lda, ipiv, info$)
**else**
    xSZCHK($m, n, A, lda$, '$Pure\ Random$', $fracnz, 2, nnz$)
    **for** $j$ **from** 1 **to** $min\{m, n\}$ **by** $n_b$ **do**
        $j_b = min\{n_b, min\{m, n\} - j + 1\}$
        $mis$ = IZSS($m - j - 1, fracnz, 16$)
        $nis$ = IZSS($n - j - 1, fracnz, 16$)
        xBZCHK($m - j + 1, j_b, A(j, j), lda, row, milim, mis$)
        IZCOMB($milim, mis, 4$)
        xGETF2_ILIM($m - j + 1, n, A(j, j), lda, ipiv(j), milim, mis, iinfo$)
        **if** $info = 0$ and $iinfo > 0$ **then**
            $info = iinfo + j - 1$
        **end if**
        **for** $i$ from $j$ to $min(m, j + j_b - 1)$ **do**
            $ipiv(i) = ipiv(i) + j - 1$
        **end for**
        xLASWP($j - 1, A, lda, j, j + j_b - 1, ipiv, 1$)
        **if** $j + j_b < n$ **then**
            xLASWP($n - j - jb + 1, A(1, j + jb), lda, j, J + j_b - 1, ipiv, 1$)
            xBZCHK($j_b, n - j - jb, A(j, j + j_b), lda, col, nilim, nis$)
            IZCOMB($nilim, nis, 4$)
            xTRSM_ILIM($Left, Lower, No\ transpose, Unit, j_b, n - j - j_b + 1,$
                          $1, A(j, j), lda, A(1, 1 + j_b), lda), nilim, nis$)
            IZSUB($milim, mis, jb$)
            if $mis > 0$ then
                xGEMM_ILIM($No\ transpose, No\ transpose, m - j - j_b + 1,$
                                    $n - j - j_b + 1, j_b, -1, A(j + j_b, j), lda, A(j, j + j_b),$
                                    $lda, 1, A(j + j_b, j + j_b), lda, milim, mis, nilim, nis$)
            end if
        end if
    end for
end if
```

General full Cholesky (ASD2_SPOTRF)

Interface ASD2_xPOTRF($uplo, n, A, lda, info$) with:

- Inputs:
  - $uplo$: 'U' for $A = U^T \times U$ factorization or 'L' for $A = L \times L^T$ factorization
  - n: the dimensions of matrix A
  - A: the matrix to be factorized
  - lda: the leading dimension of A
  - $info$: an error parameter for incomplete factorizations
- Outputs:
  - A: the factorized matrix
- Algorithm:

```
/* Setting minimum subblock size to 16 sets max size for ilim to 2*n/16 */
Allocate integer space for $milim(n/8), nilim(n/8)$
Get $n_b$ from $ILAENV$
if $n_b \leq 1$ or $n_b \geq n$ then
    Call BLAS2 version xPOTF2($uplo, n, A, lda, info$)
else
    if uplo = Upper then
        xSZCHK($n, n, A, lda, `UpperRandom', fracnz, 2, nnz$)
        for $j$ from 1 to $n$ by $n_b$ do
            $j_b = min\{n_b, n - j + 1\}$
            $mis$ = IZSS($j - 1, fracnz, 16$)
            $nis$ = IZSS($n - j + 1, fracnz, 16$)
            $nis_2 = nis$
            if $mis > 0$ then
                xBZCHK_ILIM($j - 1, j_b, A(1, j), lda, row, milim, mis$)
                IZCOMB($milim, mis, 4$)
                xSYRK_ILIM($Upper, Transpose, j_b, j-1, -1, A(1,j), lda, 1, A(j,j), lda,$
                            $milim, mis$)
            end if
            xPOTF2($Upper, jb, A(j,j), lda, info$)
            if $info \neq 0$ then
                $info = info + j - 1$ and return
            end if
            if $mis > 0$ then
                xBZCHK_ILIM($j-1, n-j-j_b+1, A(1, j+j_b), lda, col, nilim, nis$)
                IZCOMB($nilim, nis, 4$)
                if $nis > 0$ then
                    xGEMM_ILIM($Transpose, No\ transpose, j_b, n-j-j_b+1,$
                                $j-1, -1, A(1,j), lda, A(1,j+j_b), lda, 1,$
                                $A(j, j+j_b), lda, milim, mis, nilim, nis$)
                end if
            end if
            if $n - j - j_b + 1 > 0$ then
                $nis = nis_2$
                xBZCHK_ILIM($j_b, n-j-j_b+1, A(j, j+j_b), lda, col, nilim, nis$)
                IZCOMB($nilim, nis, 4$)
                if $nis > 0$ then
                    xTRSM_ILIM($Left, Upper, Transpose, Nonunit, j_b, n-j-j_b+1$
                                $, 1, A(j,j), lda, A(j, j+j_b), lda, nilim, nis$)
                end if
            end if
        end for
    else
        /* Algorithm is mirrored for lower */
        xSZCHK($n, n, A, lda,$ '$LowerRandom$'$, fracnz, 2, nnz$)
        for $j$ from 1 to $n$ by $n_b$ do
            $j_b = min\{n_b, n-j+1\}$
            $mis$ = IZSS($n-j+1, fracnz, 16$)
            $mis_2 = mis$
            $nis$ = IZSS($j-1, fracnz, 16$)
            if $nis > 0$ then
                xBZCHK_ILIM($j_b, j-1, A(j,1), lda, col, nilim, nis$)
                IZCOMB($nilim, nis, 4$)
                xSYRK_ILIM($Lower, No\ transpose, j_b, j-1, -1, A(j,1), lda, 1, A(j,j),$
                            $lda, nilim, nis$)
            end if
            xPOTF2($Lower, jb, A(j,j), lda, info$)
            if $info \neq 0$ then
                $info += j - 1$ and return
            end if
            if $nis > 0$ then
```

```
                xBZCHK_ILIM($n-j-j_b+1, j-1, A(j+j_b,1), lda, row, milim, mis$)
                IZCOMB($milim, mis, 4$)
                if $mis > 0$) then
                    xGEMM_ILIM($No\ transpose, Transpose, n-j-j_b+1, j_b, j-1,$
                                    $-1, A(j,1), lda, A(j+j_b,1), lda, 1, A(j+j_b,j),$
                                    $lda, milim, mis, nilim, nis$)
                end if
            end if
            if $n-j-j_b+1 > 0$ then
                $mis = mis_2$
                xBZCHK_ILIM($n-j-j_b+1, j_b, A(j+j_b,j), lda, row, milim, mis$)
                if $mis > 0$ then
                    xTRSM_ILIM($Right, Lower, Transpose, Nonunit, n-j-j_b+1, j_b,$
                                    $1, A(j,j), lda, A(j+j_b,j), lda, milim, mis$)
                end if
            end if
        end for
    end if
end if
```

4.2.3 Considerations

While the ASD2 implementations have a more complicated scheme for determining the nonzero bounds and applying them than ASD1 they still do not change any of the pivoting choices so there is no significant change in the error norms for the result with the only differences coming from the floating point non-reproducibility of BLAS and LAPACK mentioned for ASD1.

The ASD2_xPOTRF factorization has a larger overhead for the checks performed especially for matrices that exhibit sparsity. This is explained by the required check on A_R being more complex and needing to be computed for the each panel. This limits the current potential performance for ASD2_xPOTRF and will be investigated for solutions in the future.

The NaN and Inf propagation properties of ASD2 are the same as ASD1 with most $0 * NaN$ and $0 * Inf$ not propagating NaNs in the resulting matrices.

4.3 Full sparsity (ASD3)

4.3.1 Description

The third and last Automatic Sparsity Detection algorithm is named "Full Sparsity" or ASD3 for short. The main difference for this algorithm lies in the fact that, if it detects that high fill-in would occur due to the first panel factorization, it computes the sparsity matrix, which is a matrix containing the fraction of nonzeros per each subblock, and one would have to permute the entire matrix to move subblocks with a low fraction of nonzero to the upper-left and subblocks with a high fraction of nonzeros to the lower-right. Once this initial permutation finishes it continues and performs the ASD2 algorithm. A representation of this can be seen in Figure 4.9. While an exact permutation to minimize this could be very time-consuming many heuristics exist to give general estimates for good permutations which can then be used as starting guesses by the ASD2 routines.

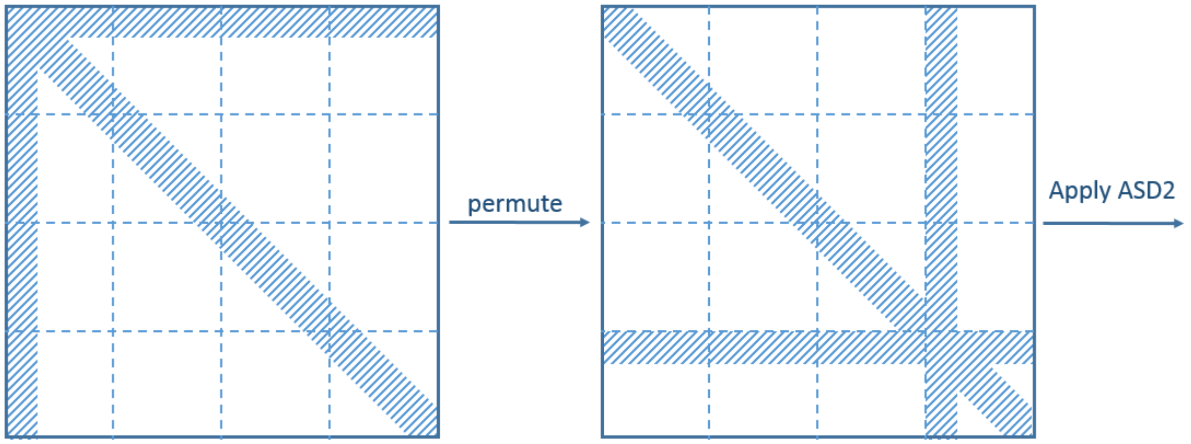


Figure 4.9. ASD3 sparsity exploitation steps for A.

The ASD3 algorithm would initially begin with the steps in ASD2 and compute the fraction of nonzeros estimate $fracnz$ and the initial $ilim$ checks on the panel. From the $ilim$ nonzero bound array it could compute an estimated sparsity for the panel (for $milim$) and the part of U(from $nilim$). If these row and column sparsity estimates are much higher than the general sparsity estimate, then the algorithm performs the full sparse check $xFZCHK$ and computes both the exact sparsity and the sparsity for each subblock.

A permutation would then need to be applied to entire blocks either directly, by moving the elements between positions, or indirectly, by keeping the permutation arrays as parameters that are passed to all ASD2 routines. The advantage of the direct method is that while there is large cost associated with moving the elements initially the rest of the auxiliary functions remain unchanged. The advantage of the indirect method is that the transfer is avoided but at the cost of increased indirection and the inability to merge subblocks together.

The indirect method also requires all wrapper auxiliary functions to be rewritten to take into account the permutation arrays.

For matrices presenting the same structure as those described in ASD2 the ASD3 algorithm would only add a single if check at the first panel so overhead would be light. For the matrices that it would try to optimize, regardless of the direct or indirect implementation method, there would be at least an extra $O(m \times n)$ initial cost over ASD2 algorithm plus the cost of moving the matrix $O(m \times n)$ or of indirect referencing. These extra costs however would only be for matrices that were both a)very sparse (low $fracnz$) and would have high fill-in (determined from $milim, nilim$).

A particular matrix type that could benefit from this is an inverted arrow with the dense rows/columns in the first positions but in general the ASD3 algorithm would provide the most sparsity detection and exploitation that could be achieved without going to a sparse solver.

Because any implementation of ASD3 would require the ability to permute the columns of a matrix rather than just the rows and such a change would require an interface modification for the LU and Cholesky routines it was decided to leave this algorithm for future implementation.

One last aspect to consider is that, if the sparsity detected is very large and the sparsity matrix S, which is obtained by xFZCHK, also does not show a good option for permutation, it might be the case that a sparse solver needs to be called as the only means of extracting speedup. This however would require LAPACK (or a higher level framework) to check for the availability of a sparse solver installed on the same machine and would also require a matrix format transformation for input into such a solver.

Chapter 5

Results

5.1 Testing

5.1.1 Input Matrices

In order to test the new implementations five matrix structures were chosen to test both the overheads and the potential benefits of the algorithms:

- full - full dense matrix
- arrow - band matrix with bandwidth b and last b rows and columns dense
- band - band matrix with bandwidth b
- profile - band matrix with bandwidth varying between 1 and b
- bulge - band matrix with bandwidth b_{large} but most values within $b, b < b_{large}$

A spy plot for the four non-dense matrix structures can be seen in Figure 5.1.

To these five generated structures were added a sample set of ten real sparse matrices from the University of Florida database [5] that presented structures like:

- Reverse Cuthill-McKee ordering
- Nested-Dissection ordering
- Low bandwidth with high nnz

The general characteristics and a description of the sample set matrices can be found in Table 5.1 while the spy plots can be found in Figure 5.2.

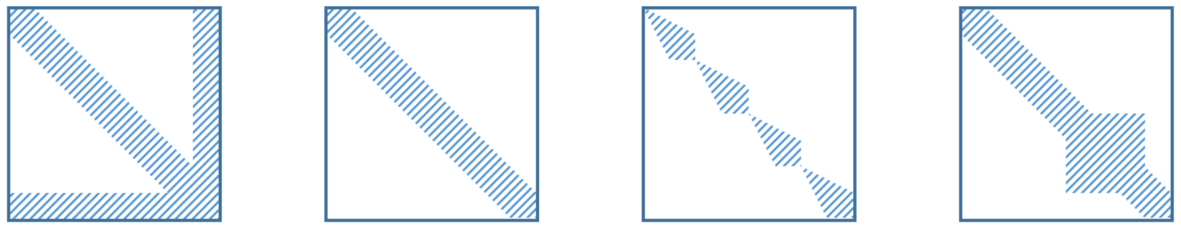


Figure 5.1. Input matrix formats: arrow, band, profile, bulge.

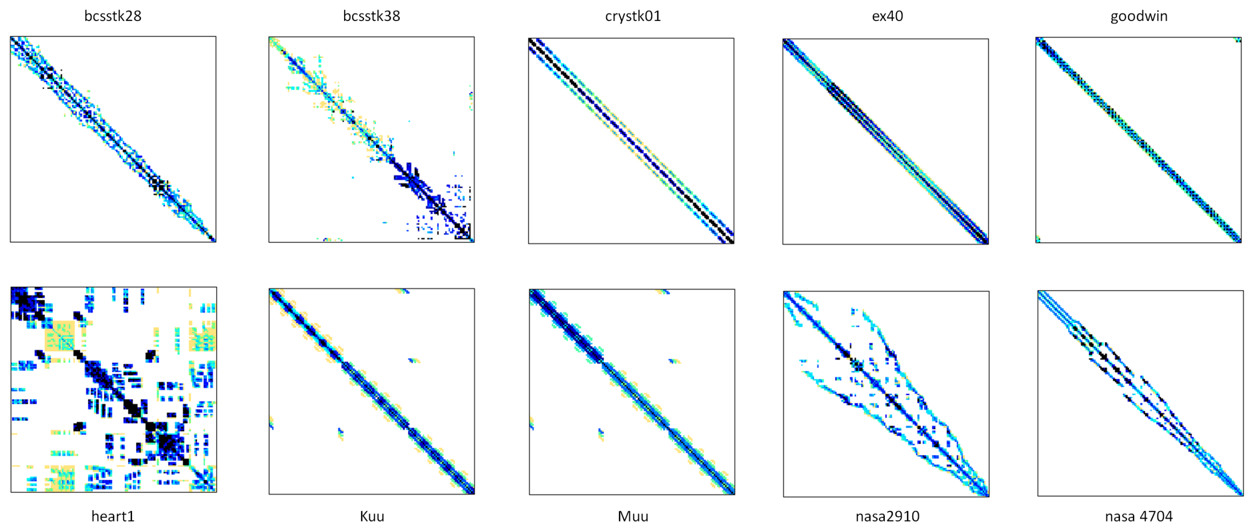


Figure 5.2. Sparse matrix spy plots.

5.1.2 Testing Methodology

The data gathered has four independent components to their information:

- the factorization used: GE, PO, GB, PB
- the algorithm used for the factorization: LAPACK, ASD1, ASD2
- whether the matrix was positive definite diagonally dominant (w or w/o pivoting)
- the dimensions, bandwidths and other parameters describing the sparsity structure of the matrix being factorized

Tests were conducted within each structure for matrices starting from $n = 200$ to 4000 in increments of 200 (for the dense general and positive definite cases the increments chosen were 100). For structures that had bandwidths these went from $b = 100$ to 2000 in increments of 100. For the arrow and dense matrices only the ASD1_SGETRF, ASD1_SPOTRF and ASD2_SGETRF, ASD2_SPOTRF algorithms were tested. For band matrices the tests were

Name	*nnz*	*n*	*b*	SPD	Description
bcsstk28	219024	4410	524	Y	Solid element model (MSC Nastran)
bcsstk38	355460	8032	6029	Y	Stiffness matrix, airplane engine component
crystk01	315891	4875	239	Y	FEM crystal free vibration stiffness matrix
ex40	456188	7740	277	N	CFD test matrix from FIDAP
goodwin	324772	7320	7272	N	CFD FEM, fluid mechanics problem
heart1	692108	3557	3262	N	Quasi-static FEM of a heart
Kuu	340200	7102	4697	Y	Mathworks structural problem
Muu	170134	7102	4696	Y	Mathworks structural problem
nasa2910	174296	2910	859	Y	Structure from Nasa Langley
nasa4704	104756	4704	423	Y	Structure from Nasa Langley

Table 5.1. Sparse matrix sample set.

limited to cases where $b \leq n/2$. The bulge structure was located between $0.5 \times n$ and $0.75 \times n$ and the band tests were limited to $b < 0.25 \times n$.

All ASD and reference LAPACK implementations used a single matrix per characteristic dimension that was factorized for 10 iterations to reduce any timing variations. Timing was done for each factorization individually so as to avoid any overheads due to matrix copying costs. The total time for all 10 iterations is reported back to the user, and so dividing by 10, we report the average time here.

The five matrix structures were created and populated with random numbers between $[-1 : 1]$ for all entries, with the only exception being the diagonals for the Positive Definite matrices which were initialized to $n + rand[0 : 1]$. For the sparse matrices the data was read from an input file containing the matrices in Matrix Market format [2].

The general routines were also used on the Positive Definite input matrices tested to check the performance difference for matrices of the same size and initial structure but with no pivoting overhead as opposed to the other extreme of fully random matrices which means that many pivots will be selected from outside the diagonal.

5.1.3 Test Platform Specifications

All tests were run on NERSC's Hopper Supercomputer which is a Cray XE6 with dual twelve-core AMD 'MagnyCours' 2.1-GHz processors per node which have a 8.4 Gflops/-core peak arithmetic intensity. The default LAPACK routine available on Hopper is The Cray Scientific Libraries package, LibSci version 13.0.0, which was used for the LAPACK comparison.

The default Hopper PGI compilers were used with *ftn* for all the ASD functions and *pgcc* via *cc* for all the benchmarking code attached. The only optimization flags used were "-O3" during the compilation and linking of the C files.

While the correctness runs were sometimes executed on the login nodes all benchmarking data was obtained by runs on the serial nodes on Hopper. The timer routine $gettimeofday$ was used for the performance checks.

5.2 ASD Overhead

The goal of this section is to measure the overhead added by the sparsity checking when the matrix is dense, to ensure there is little to no slowdown in this common case.

For the purposes of this section we will refer to overhead as representing the time of the ASD routine divided by the time of the same LAPACK routine, expressed as a percentage, from which 100 has been subtracted, so $Overhead = 0\%$ would mean both LAPACK and ASD running in the same amount of time.

$Overhead = time_{ASD}/time_{LAPACK}\% - 100$

To summarize our results we can see that ASD1_SGETRF and ASD1_SPOTRF show no slowdown (overhead less than 1%) while ASD1_SGBTRF, ASD1_SPBTRF, ASD2_SGETRF, and ASD2_SPOTRF all show some overhead (from 10% to 35%) for small matrix or bandwidth sizes that quickly goes down to negligible values (from 2% to 5%) as the dimensions increase.

5.2.1 Full Solvers - ASDx_SGETRF and ASDx_SPOTRF

The timing results for running the ASD LU routines ASD1_SGETRF and ASD2_SGETRF on a full dense matrix are presented in Figure 5.3. These results are represented in the top two graphs; with the first one, *Top* having as input a random matrix (with pivoting) and the second graph, *Middle* having as input a diagonally dominant positive definite matrix (without pivoting). The vertical axis represents the overhead the ASD function exhibits over the LAPACK function as a percentage (lower is better) while the horizontal axis represents the size of the matrix n (increasing to the right).

Also presented in the Figure 5.3 in the *Bottom* graph are the results for the ASD Cholesky routines ASD1_SPOTRF and ASD2_SPOTRF ran on a diagonally dominant positive definite matrix.

In order to stress the ASD implementations and clarify the maximum possible overhead, matrices with smaller sizes $n = 65 : 200$ were also tested with the ASD solvers, and are presented in Figure 5.4. The data from $n = 1 : 64$ is not presented because these implementations automatically call the BLAS2 routines for both LAPACK and ASD codes and

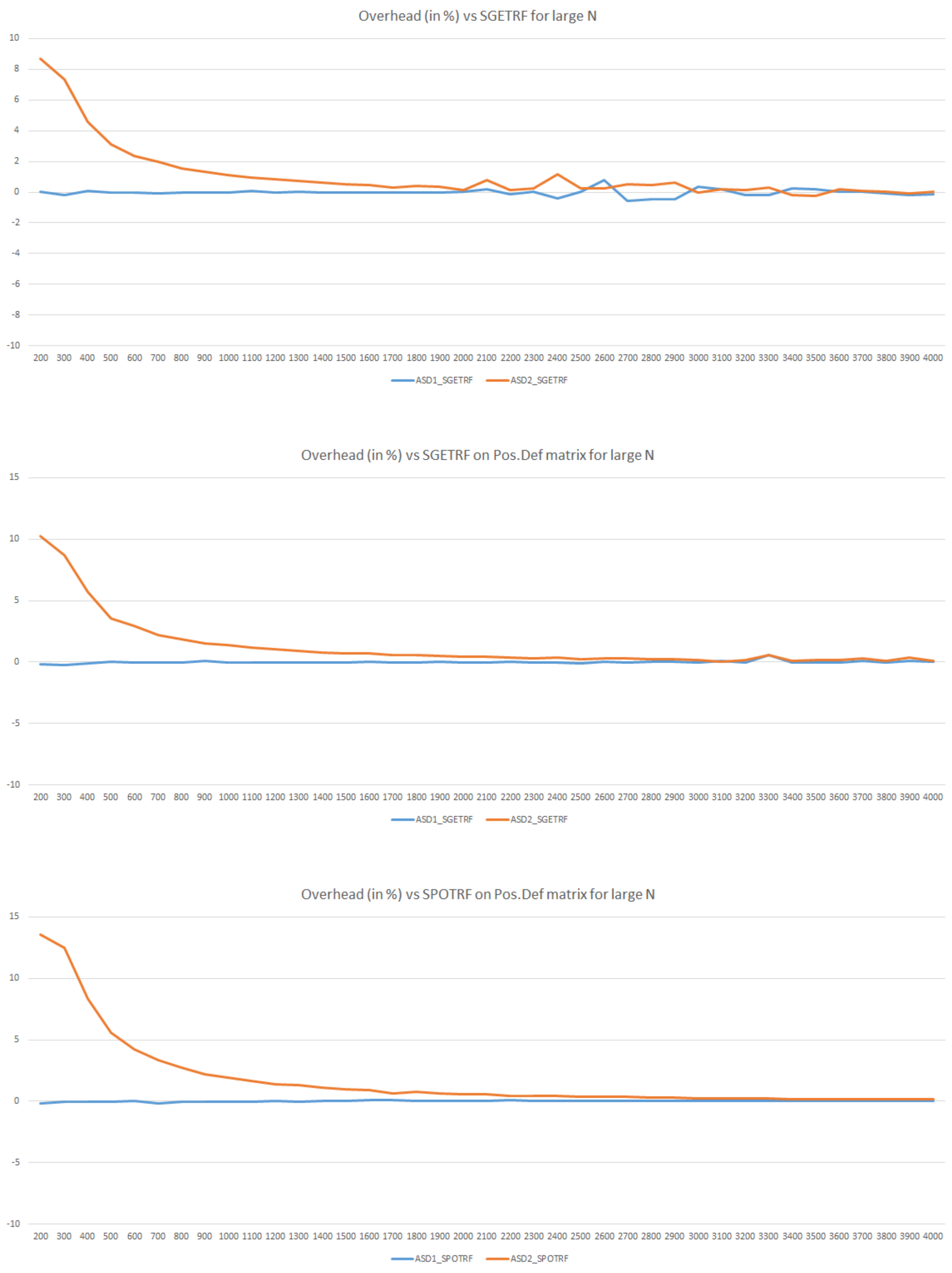
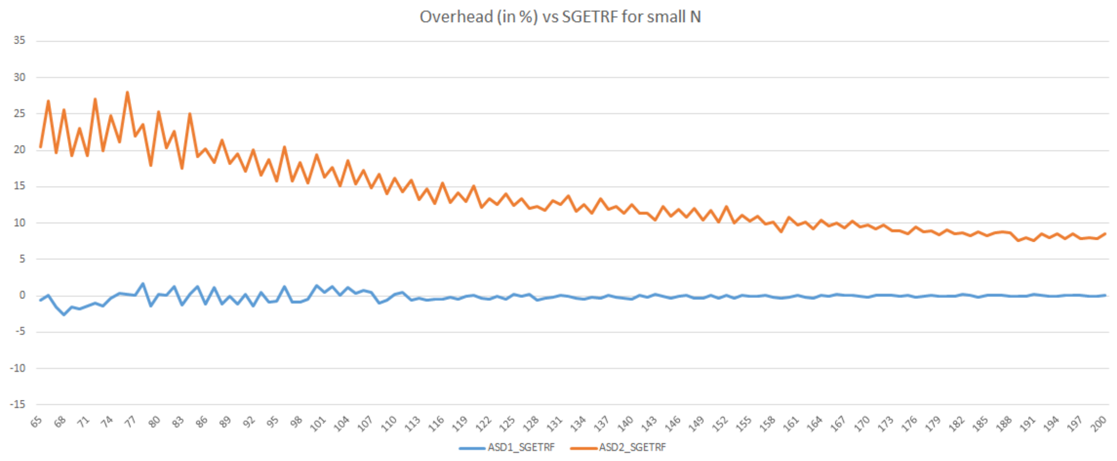
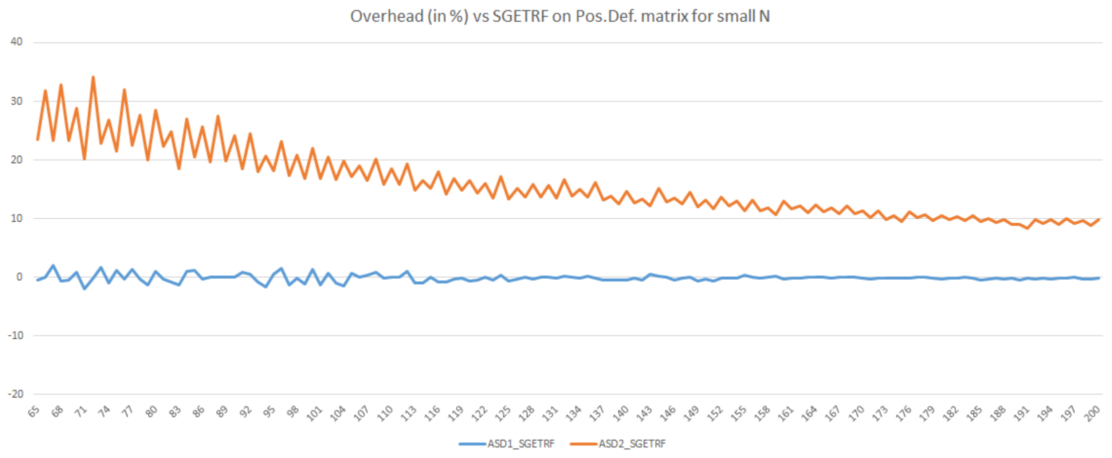


Figure 5.3. Large n overhead for *Top:* ASDx_SGETRF on random matrix (with pivoting) *Middle:* ASDx_SGETRF on random Positive Definite matrix (without pivoting) *Bottom:* ASDx_SPOTRF on random Positive Definite matrix





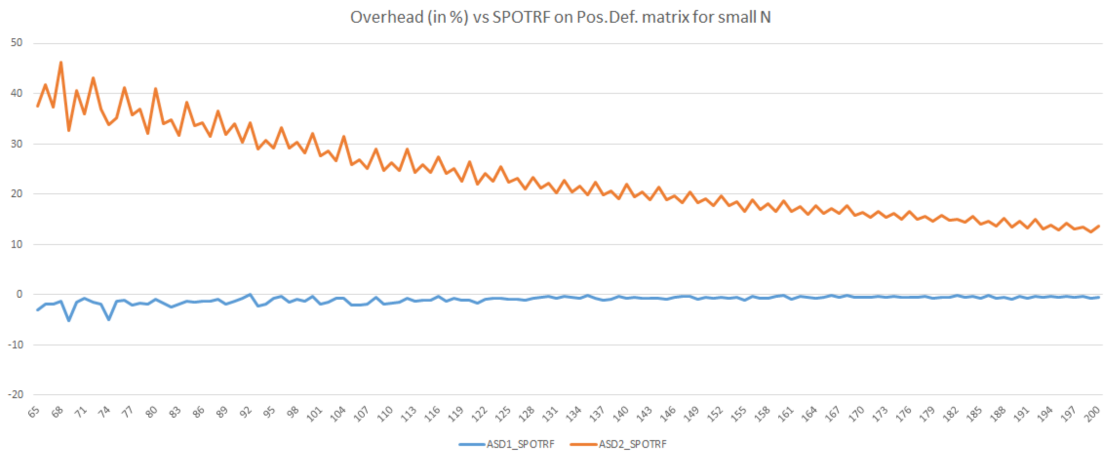


Figure 5.4. Small n overhead for *Top:* ASDx_SGETRF on random matrix (with pivoting) *Middle:* ASDx_SGETRF on random Positive Definite matrix (without pivoting) *Bottom:* ASDx_SPOTRF on random Positive Definite matrix

would not present meaningful data for the ASD routines while at the same time lowering the computed minimum, median and average for the overheads.

The overhead results are summarized in Table 5.2 for ASD1 and in Table 5.3 for ASD2 and present a clear split in potential overhead between the two ASD algorithms.

	ASD1_SGETRF		ASD1_SPOTRF
	w pivoting	w/o pivoting	
large n			
Min	-0.6	-0.26	-1.22
Median	0.02	0.01	0.02
Average	0.04	0.01	-0.02
Max	0.79	0.58	0.11
small n			
Min	-2.60	-2.01	-5.18
Median	-0.12	-0.09	-0.92
Average	-0.02	-0.11	-0.66
Max	1.73	2.02	0.00

Table 5.2. Percentage overhead for ASD1_SGETRF and ASD1_SPOTRF

The ASD1_SGETRF and ASD1_SPOTRF routines show negligible overhead (maximum 2% for small n) when applied to full matrices. The overhead seems to be independent of matrix dimension and with an average overhead of 0.04% and -0.02% we can safely state that for all practical purposes the ASD1 routines exhibit no overhead.

	ASD2_SGETRF		ASD2_SPOTRF
	w pivoting	w/o pivoting	
large n			
Min	-0.23	0.02	0.14
Median	0.46	0.43	0.54
Average	1.08	1.29	1.79
Max	8.72	10.24	13.59
small n			
Min	7.66	8.41	12.58
Median	12.34	14.02	21.49
Average	13.76	15.82	23.53
Max	28.06	34.24	46.23

Table 5.3. Percentage overhead for ASD2_SGETRF and ASD2_SPOTRF

The full implementations of ASD2 for LU and Cholesky however exhibits a different performance with the highest overhead reaching 34% and 46%. The data also shows that as less pivoting occurs inside an LU factorization the ASD routines exhibit higher overhead (by up to 6.2%) because of the lower number of total operations that can hide the extra work. While these numbers are significant and would discourage its use from replacing any LAPACK implementation, it is important to see that all the overhead is largest for the smallest matrices with the overhead cost decreasing drastically as n increases. For ASD2_SGETRF the overhead drops below 5% by $n = 500$ and for ASD2_xPOTRF the overhead drops below 5% by $n = 600$. This high drop-off above a certain range implies that the ASD2 algorithm could still be effectively utilized but should be limited to use in the higher ranges of n. For lower values of n we could look at the creation of a hybrid ASD2/ASD1 algorithm that switches to ASD1 for these lower values while staying with ASD2 for the large n.

A large part of the overhead for ASD2 can be attributed to the extra sparse scan that gets computed at the start of any factorization that does not satisfy the bounds for BLAS2. Therefore a potential optimization could be to skip that initial step and assume a value for the fraction of nonzeros $fracnz$ which would select only a few blocks per panel.

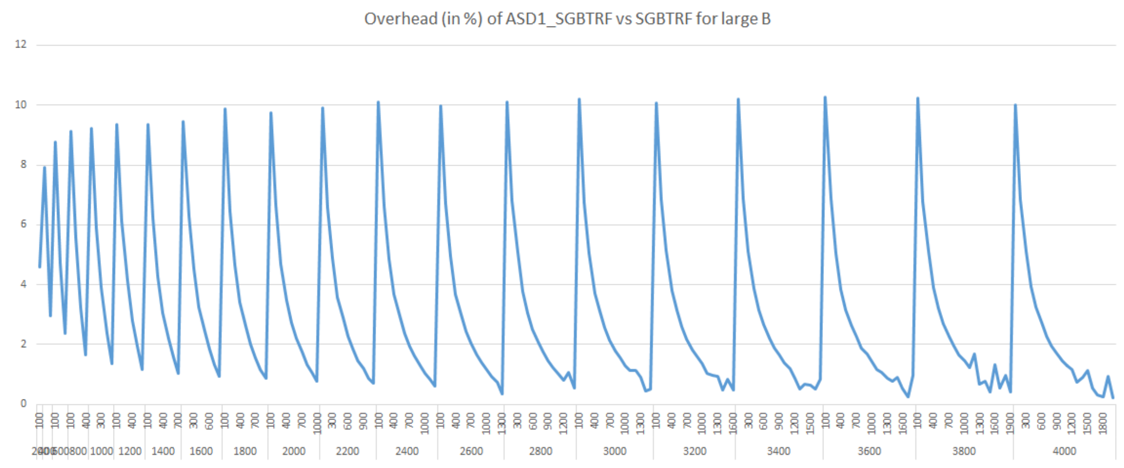
5.2.2 Band Solvers - ASD1_SGBTRF and ASD1_SPBTRF

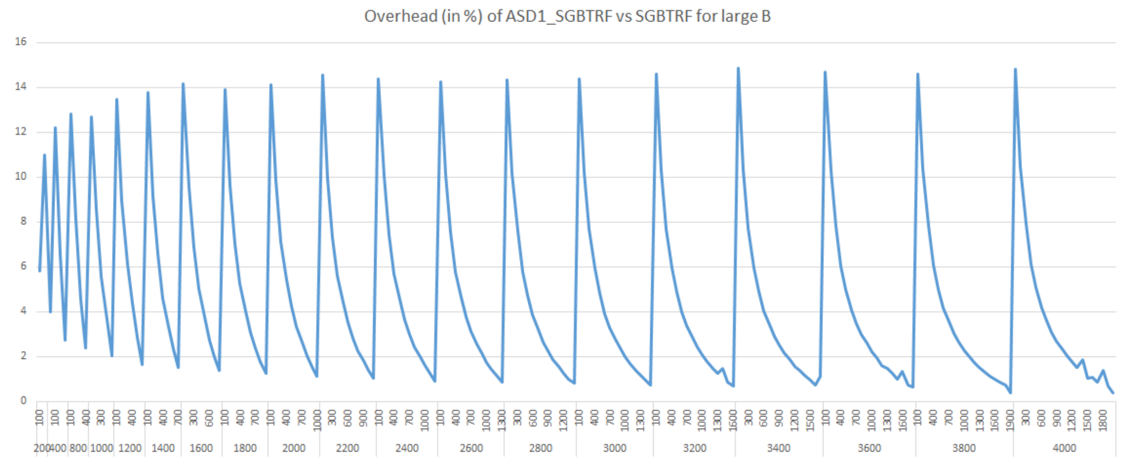
The timing results for running the banded ASD LU routine ASD1_SGBTRF on a full band matrix are presented in Figure 5.5. The results are shown in two graphs; in the *Top* graph the performance for a random input matrix is presented (with pivoting) and in the *Middle* graph the input is a random banded Positive Definite matrix (without pivoting).

The vertical axis represents the overhead of the ASD function over the same LAPACK function as a percentage (lower is better) while the horizontal axis represents at the outer level the size of the matrix n (increasing to the right) and at the inner level the size of the band b (increasing to the right within a constant matrix size).

In order to stress the ASD implementations and clarify the maximum possible overhead, matrices with smaller bandwidths $b = 65 : 200$ were also tested with the ASD solvers, and are presented in Figure 5.6. The data from $b = 1 : 64$ is not presented because these implementations automatically call the BLAS2 routines for both LAPACK and ASD codes and would not present meaningful data for the ASD routines while at the same time lowering the computed minimum, median and average for the overheads. Since the largest overhead was being initially reported for the largest matrix size n, the smaller band figure only contains the overhead results for a single value of $n = 4000$.

These results are summarized in Table 5.4 for both the LU and Cholesky factorizations. The data shows that a relatively noticeable maximum overhead of 15% for the band LU and 10% for the band Cholesky implementations. This overhead lowers as the size of the bandwidth increases with overhead getting below 5% when the bandwidth gets above $b = 300$.





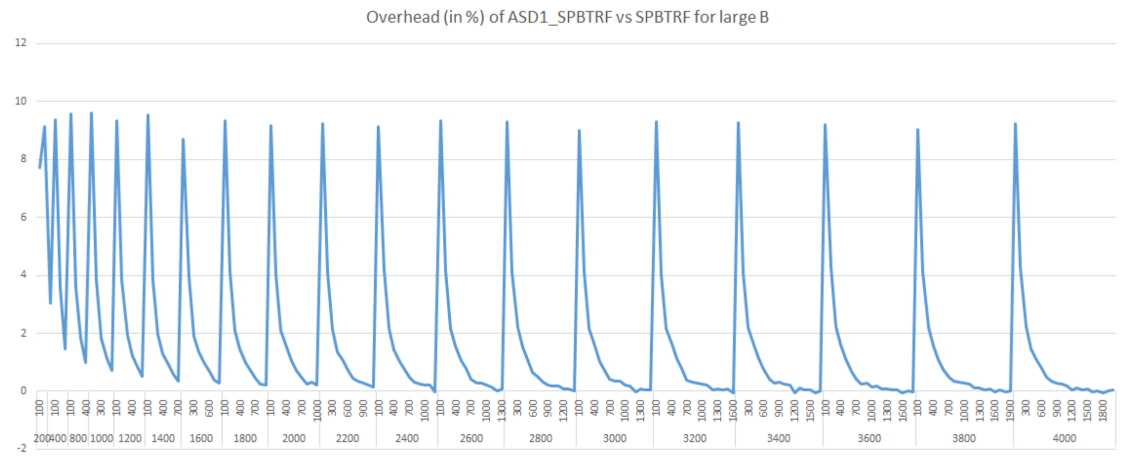


Figure 5.5. Large b overhead for *Top:* ASD1_SGBTRF on random matrix (with pivoting) *Middle:* ASD1_SGBTRF on random Positive Definite matrix (without pivoting) *Bottom:* ASD1_SPBTRF on random Positive Definite matrix

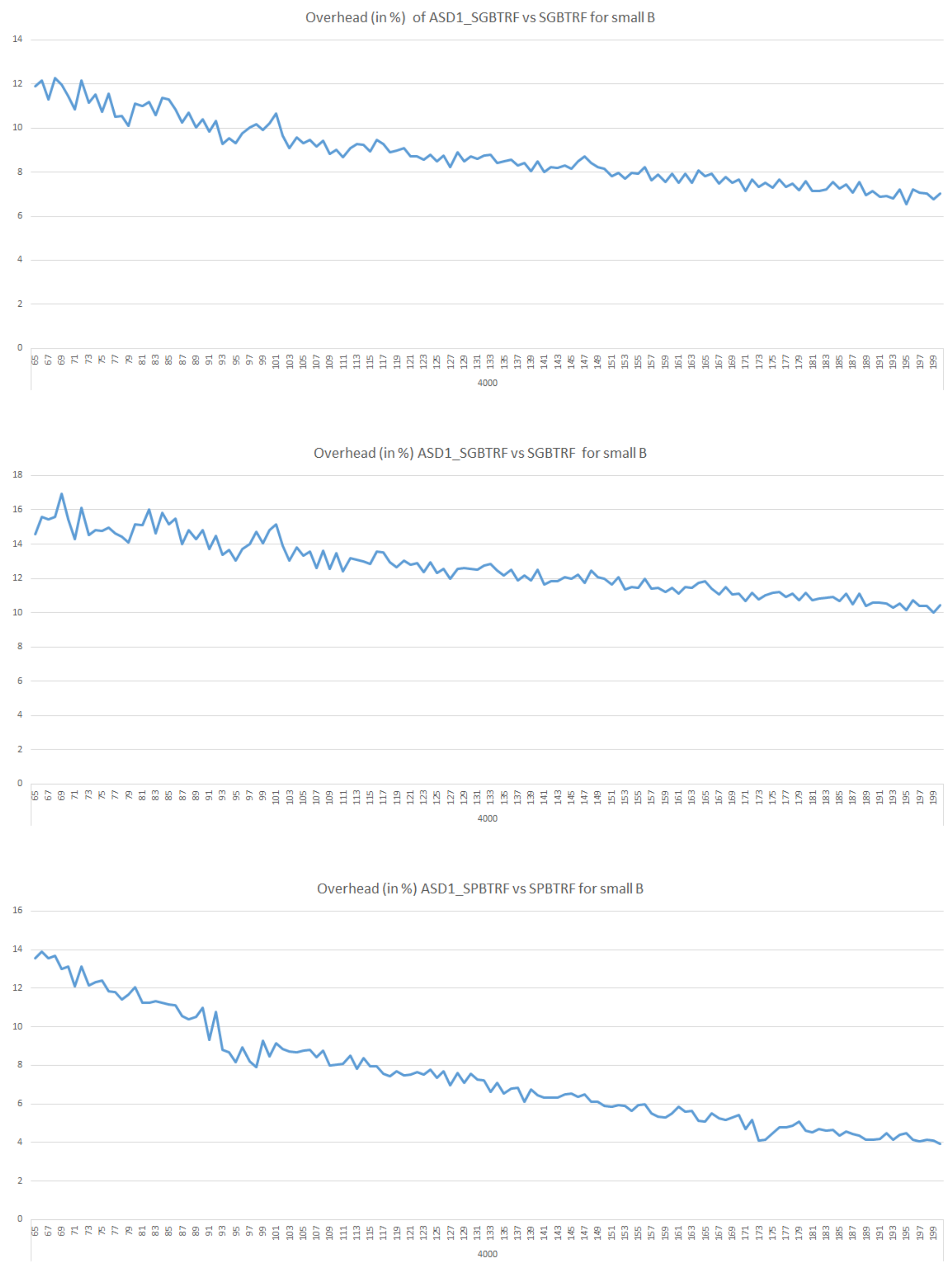


Figure 5.6. Small b overhead for *Top:* ASD1_SGBTRF on random matrix (with pivoting) *Middle:* ASD1_SGBTRF on random Positive Definite matrix (without pivoting) *Bottom:* ASD1_SPBTRF on random Positive Definite matrix

	ASD1_SGBTRF		ASD1_SPBTRF
	w pivoting	w/o pivoting	
large n			
Min	0.21	0.38	-0.06
Median	2.05	3.10	0.51
Average	3.09	4.60	1.75
Max	10.29	14.87	9.62
small n			
Min	6.53	10.02	3.9
Median	8.81	12.44	7.02
Average	8.53	12.66	7.46
Max	12.27	16.95	13.91

Table 5.4. Percentage overhead for ASD1_SGBTRF and ASD1_SPOTRF

5.3 Performance Results

The detailed results for the ASD routines are presented below according to the matrix input formats that were tested. They present the speedup of the ASD codes with respect to the reference LAPACK routines that they replace as well as the 'speedup' of the best LAPACK routine possible for a given input matrix to compare how much the ASD routines really get versus the current optimal solution.

$speedup = time_{refLAPACK}/time_{ASD}$ and $speedup = time_{bestLAPACK}/time_{refLAPACK}$

The results in this section can be summarized in Table 5.6 and Table **??** that show the potential speedup for expert and non-expert users of LAPACK. Both tables show the maximum speedup over all dimensions for the non-dense structures. The difference is that the non-expert table will shows the ASD routine with the best speedup over SGETRF while the expert table will show the ASD routine and speedup versus the best LAPACK routine that can be used for the input matrix.

Input Matrix	LAPACK call	ASD call	Speedup
Arrow	SGETRF	ASD2_SGETRF	7.26
PosDef Arrow	SGETRF	ASD2_SGETRF	24.20
Band	SGETRF	ASD1_SGETRF	22.01
PosDef Band	SGETRF	ASD1_SGETRF	32.82
Profile	SGETRF	ASD1_SGETRF	24.08
PosDef Profile	SGETRF	ASD1_SGETRF	35.10
Bulge	SGETRF	ASD1_SGETRF	14.37
PosDef Bulge	SGETRF	ASD1_SGETRF	21.82

Table 5.5. Speedup for non-expert user

The axes for the plots in this section contain on the vertical axis the speedup, while on the horizontal axis the outer level represents the size of the matrix n (increases to the right)

Input Matrix	LAPACK call	ASD call	Speedup
Arrow	SGETRF	ASD2_SGETRF	7.26
PosDef Arrow	SPOTRF	ASD2_SPOTRF	7.62
Band	SGBTRF	ASD1_SGBTRF	1
PosDef Band	SPBTRF	ASD1_SPBTRF	1
Profile	SGBTRF	ASD1_SGBTRF	2.17
PosDef Profile	SPBTRF	ASD1_SPBTRF	2.23
Bulge	SGBTRF	ASD1_SGBTRF	3.82
PosDef Bulge	SPBTRF	ASD1_SPBTRF	4.85

Table 5.6. Speedup for expert user

and the inner level represents the size of the band b (increasing to the right within a constant matrix size).

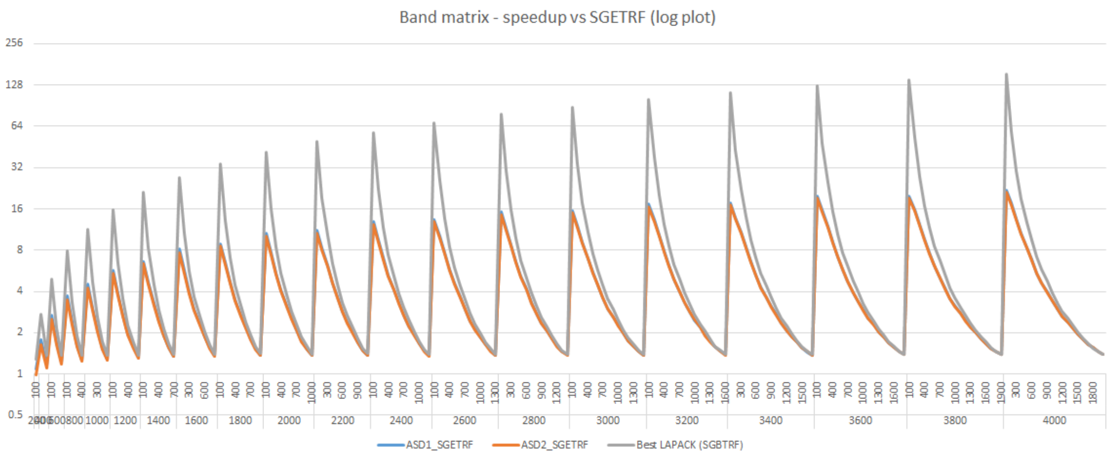
5.3.1 Band Matrix

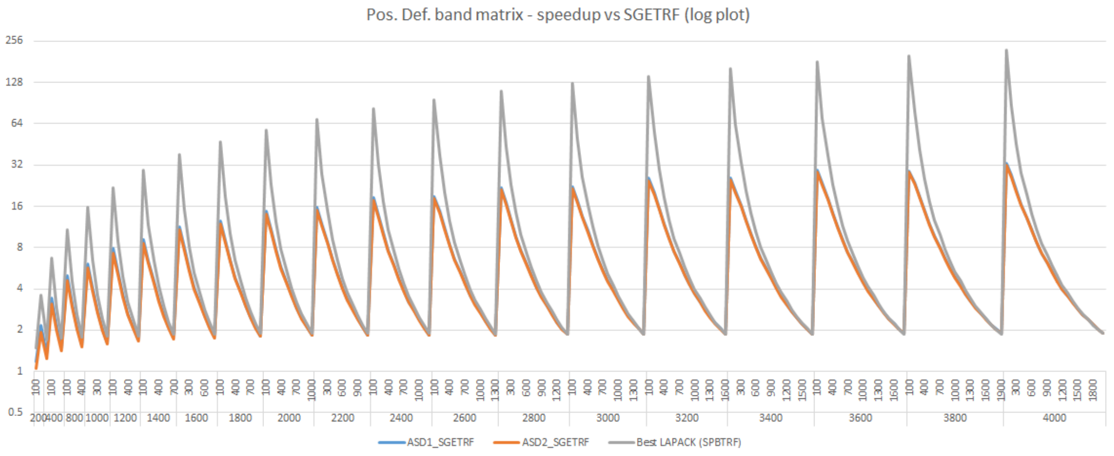
While the band matrix was used to test the overhead for the ASD1_SGBTRF and ASD1_SPBTRF routines it also served a first test for the ASD routines when used on the general LU and Cholesky LAPACK functions.

The performance of ASD LU routines ASD1_SGETRF and ASD2_SGETRF is shown in Figure 5.7. The *Top* graph in this figure represents the speedup obtained when run on a random band matrix with full pivoting while the *Middle* graph presents the speedup obtained when run on a diagonally dominant Positive Definite matrix (DDPD - requires no pivoting).

It can be seen from the figures that since the only structure to exploit is a single band or block the ASD1_SGETRF and ASD2_SGETRF algorithms are almost indistinguishable with the lines overlapping for most of the graphs. For the general random input case we can see that ASD1_SGETRF and ASD2_SGETRF obtain speedups of up to 22$\times$ over the dense SGETRF (for $n = 4000, b = 100$) so they provide a good benefit for the non-expert user. An expert user that calls SGBTRF which is the best LAPACK routine for the case can get a 154$\times$ vs SGETRF meaning a 7$\times$ speedup over the ASD routine. This is explained by the extra $O(n^2)$ checks that the ASD routine needs to perform in order to determine the structure of the input. It can also be seen from the graphs that this difference between the ASD algorithms and the best LAPACK (SGBTRF) is quickly lowered as the bandwidth increases with it becoming only 1.5$\times$ for $n = 4000, b = 500$.

It can also be observed that from the *Middle* graphs that a band matrix requiring no pivoting gets an improvement for the maximum speedup obtained by ASD1/ASD2_SGETRF which is now up to 32$\times$ over the dense SGETRF (for $n = 4000, b = 100$)for a non-expert user. However, in this case an expert user can utilize the SPBTRF routine meaning that they can get a speedup over SGETRF of 221$\times$ which results in a 6.7$\times$ speedup over the ASD algorithms.





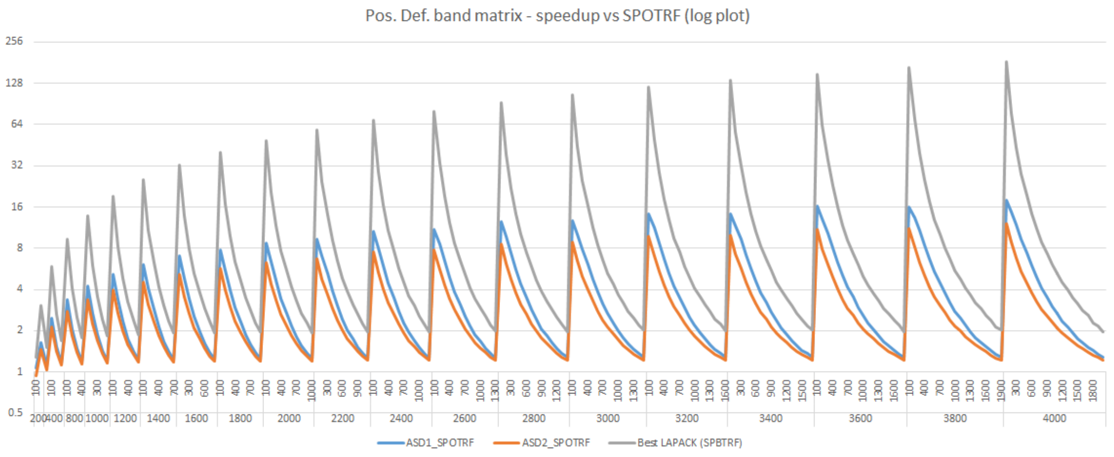


Figure 5.7. Speedup for arrow matrices with *Top:* ASDx_SGETRF vs SGETRF on random matrix (with pivoting) *Middle:* ASDx_SGETRF vs SGETRF on random Positive Definite matrix (without pivoting) *Bottom:* ASDx_SPOTRF vs SPOTRF on random Positive Definite matrix

Also included in Figure 5.7 in the *Bottom* graph are the speedups ASD Cholesky routines ASD1_SPOTRF and ASD2_SPOTRF obtained. For this graph we can notice that ASD2_SPOTRF suffers from higher overhead and is consistently below ASD1_SPOTRF. We can see that we obtain speedups of up to 18× for the ASD1_SPOTRF and 12× for the ASD2_SPOTRF routines compared to SPOTRF. These are again contrast with the expert user that can obtain a speedup of 184× over SPOTRF using SPBTRF which means being 10× faster than ASD1 routine and 15× faster than the ASD2 routine. This is explained as before by the extra $O(n^2)$ checks that the ASDx_SPOTRF routines need to perform to detect the structure.

5.3.2 Arrow Matrix

The arrow matrix can be looked at as a band matrix of bandwidth b which has its last b rows and column fully filled in. This structure means that only the dense LAPACK solvers can be called efficiently on the input. The performance of ASD LU routines ASD1_SGETRF and ASD2_SGETRF and the ASD Cholesky routines ASD1_SPOTRF and ASD2_SPOTRF is shown in Figure 5.8.

We can first observe that the ASD1 algorithm is not able to take advantage of any of the arrow structure and remains at the baseline with SGETRF and a speedup of 1×. The ASD2_SGETRF routine however is able to detect the structure and obtains a speedup of up to 7.26× for a general random input matrix with pivoting.

When run on a matrix with diagonally dominant positive definite structure (without pivoting) the ASD2_SGETRF routine obtains a speedup of 24.20× over SGETRF. This increase in speedup is explained by the fact that once we apply the normal permutations the matrix loses it's arrow structure and therefore is required to produce a higher fill-in.

By comparison, when we look at ASD1_SPOTRF we can notice that ASD1 routines are able to detect the arrow structure, due to the 'left-looking' algorithm, obtaining a speedup of up to 7.32×. This improvement versus SPOTRF is surpassed by the ASD2_SPOTRF routine which is able to obtain speedups of up to 11.16× versus SPOTRF.

This suggests that a left-looking ASD1_SGETRF might have advantages over the current right-looking version with respect to automatic sparsity detection.

5.3.3 Profile Matrix

A profile matrix is a special type of a band matrix with the bandwidth increasing from 1 to b then decreasing from b to 1 for the entire length n of an input matrix. Therefore a profile band matrix with a larger bandwidth b and same n would not necessarily cover all the non-zeros of the previous smaller sized bandwidth. Because of this property and the way that the panel blocks interact with the edges of the profile structure for varying bandwidths we can see in Figure 5.10 that the algorithm timing is not monotonic in the speedups we get

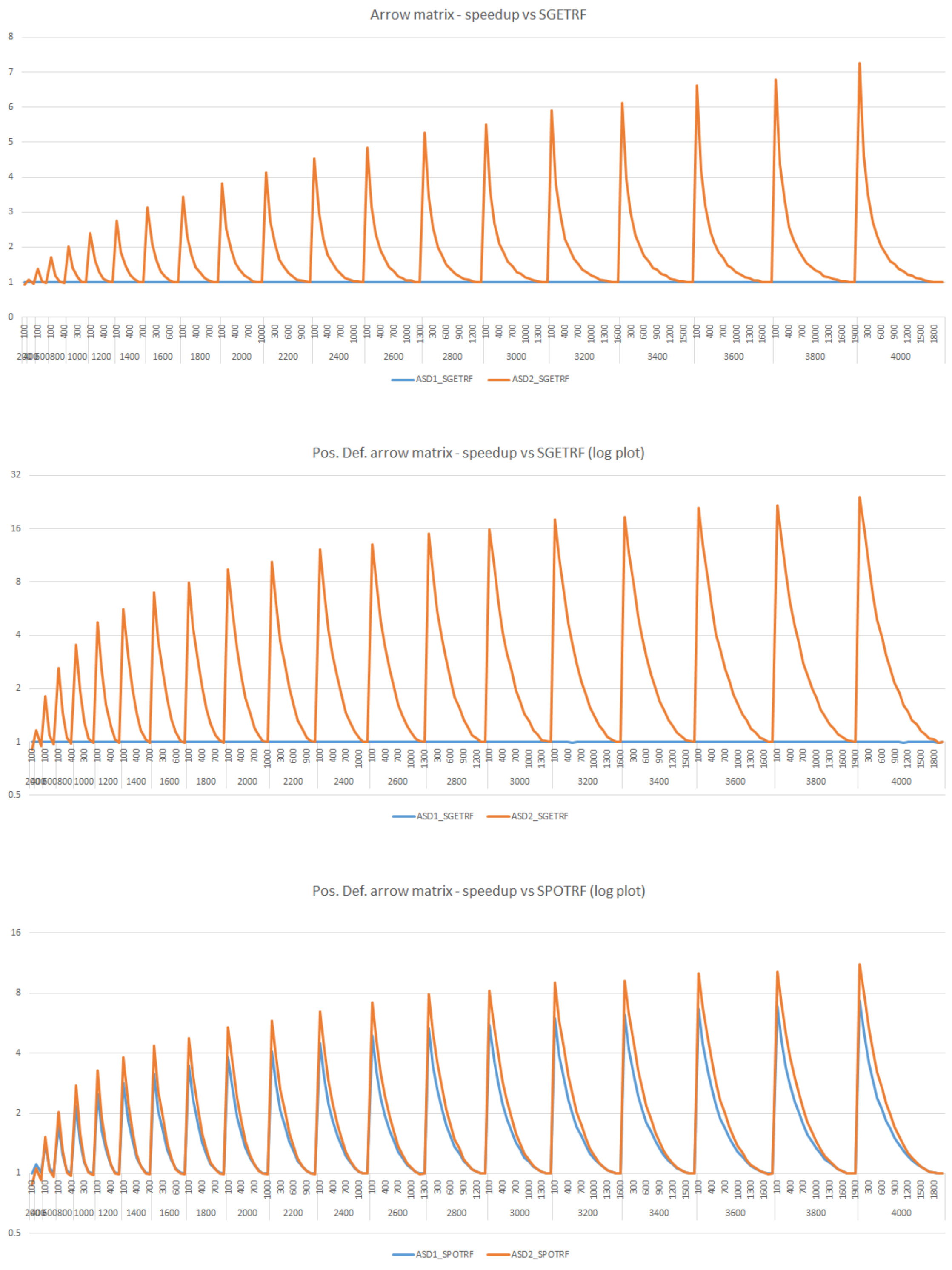


Figure 5.8. Speedup for arrow matrices with *Top:* ASDx_SGETRF vs SGETRF on random matrix (with pivoting) *Middle:* ASDx_SGETRF vs SGETRF on random Positive Definite matrix (without pivoting) *Bottom:* ASDx_SPOTRF vs SPOTRF on random Positive Definite matrix

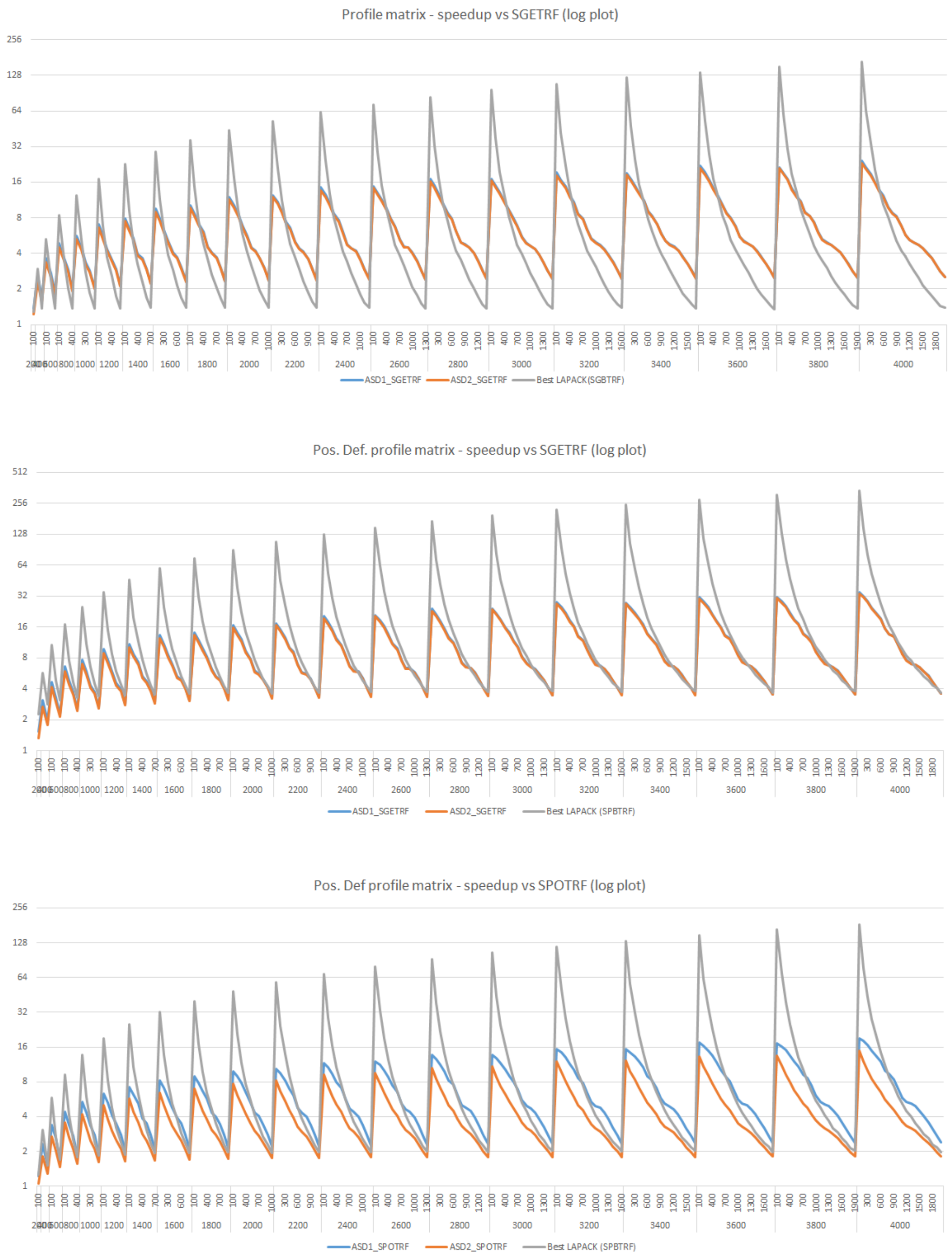
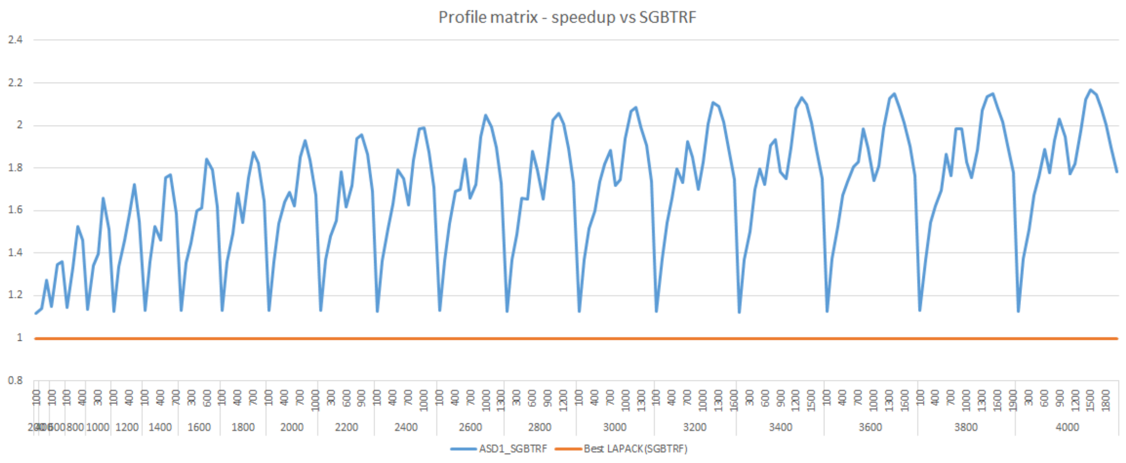
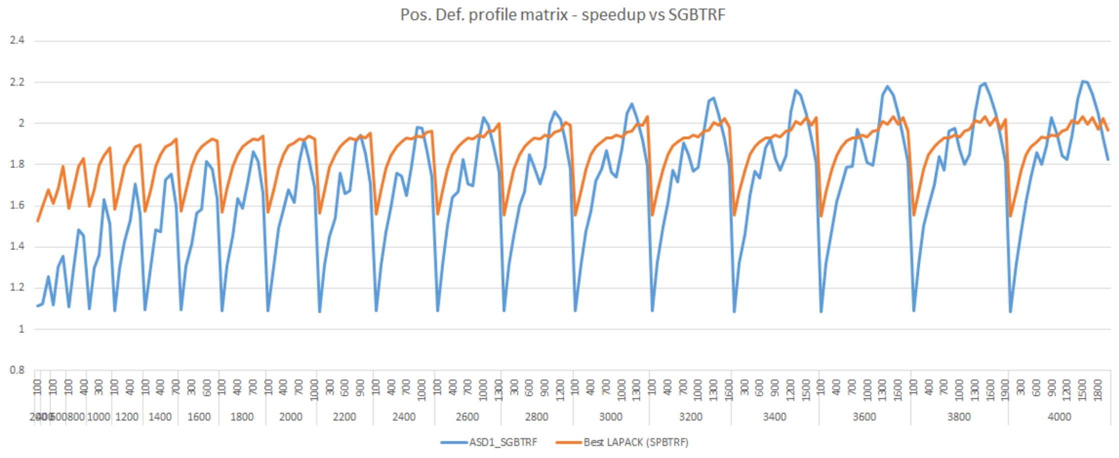


Figure 5.9. Speedup for profile matrices with *Top:* ASDx_SGETRF vs SGETRF on random matrix (with pivoting) *Middle:* ASDx_SGETRF vs SGETRF on random Positive Definite matrix (without pivoting) *Bottom:* ASDx_SPOTRF vs SPOTRF on random Positive Definite matrix





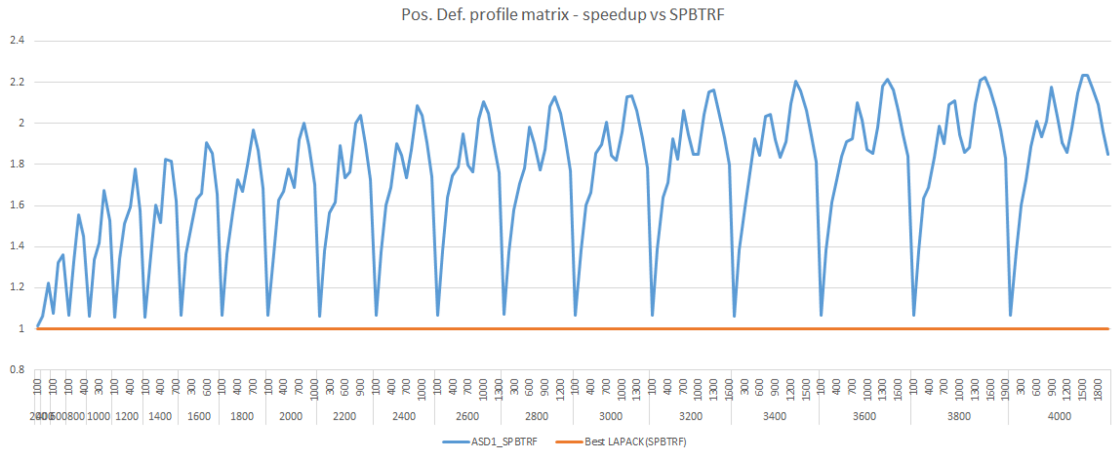


Figure 5.10. Speedup for profile matrices with *Top:* ASD1_SGBTRF vs SGBTRF on random matrix (with pivoting) *Middle:* ASD1_SGBTRF vs SGBTRF on random Positive Definite matrix (without pivoting) *Bottom:* ASD1_SPBTRF vs SPBTRF on random Positive Definite matrix

with respect to increasing/decreasing bandwidths b but is monotonic with respect to keeping bandwidth steady and increasing the size of the matrix n.

The results from the 'full' ASD1_SGETRF and ASD2_SGETRF routines can be seen in Figure 5.9 and show speedups of up to 24× and 23× respectively vs SGETRF running on a full matrix with pivoting. For the same functions ran on a diagonally dominant positive definite matrix the speedups vs SGETRF are 34× and 33× respectively.

While the SGBTRF and SPBTRF routines are more efficient for low bandwidths b, the ASD implementations are able to also provide speedups over the 'best' LAPACK routines for certain parameters as can be seen for bandwidths close to but not at the highest bandwidth $b = 1700$. For these sizes we see that ASD1_SGETRF and ASD2_SGETRF are 2.1× faster than the best LAPACK(SGBTRF).

For the ASD1 banded implementations present in Figure 5.10 we can see that compared to their LAPACK equivalents the ASD1 banded routines ASD1_SGBTRF and ASD1_SPBTRF both offer speedups of up to 2.2×. This is also the case for a positive definite input on ASD1_SGBTRF where the speedup versus the Best routine is lower because of the ability to call SPBTRF as LAPACKs choice.

5.3.4 Bulge Matrix

A bulge matrix can be described as a band matrix of bandwidth b in which a b_{large} by b_{large} dense subblock is located somewhere along the diagonal. For the purposes of this test the 'bulge' is always located starting at position $0.5 \times n$ and is $0.25 \times n$ large.

The axes for the graphs present in Figure 5.11 and Figure 5.12 contain on the vertical axis the speedup, while on the horizontal axis the outer level represents the size of the matrix n (increases to the right), the middle level represents the size of the bulge($0.25 \times n$), and the inner level represents the size of the band b (increasing to the right within a constant matrix size).

The results for ASDx_SGETRF and ASDx_SPOTRF are presented in Figure 5.11. From the *Top* graph which represents a full random input matrix (with pivoting) we can observe that not only ASD1_SGETRF is able to achieve a speedup of up to 14.37× and ASD2_SGETRF is able to achieve a speedup 14.02× but moreover the two ASD general LU algorithms are able to achieve a speedup of 5× between the ASD routines and the Best LAPACK routine (which represents SGBTRF for the top plot)

From the *Middle* graph we can see that the potential speedup for ASD1_SGETRF has increased with to 21.81× vs SGETRF and ASD2_SGETRF can achieve a value of 21.33× vs SGETRF. While we are still able to beat the Best LAPACK routine by a factor of 2 there now exists a crossover point for the bandwidth above which the best LAPACK routine (SPBTRF) is fastest.

The last graph *Bottom* in the the figure completes the look at the 'full' solvers and it shows the maximum speedups of 14.14× for ASD1_SPOTRF and 9.26× for ASD2_SPOTRF.

It is interesting to note that in this case, if compared to the best LAPACK routine for the input bulge matrix SPBTRF, we still achieve a 4.85× speedup for the ASD1 routine and a 2× speedup for the ASD2 routine.

For the expert users that might be calling the banded routines that performance improvement for ASD1_SGBTRF and ASD1_SPBTRF the results are presented in Figure 5.12. In the top two graphs in the figure called *Top* and *Middle* we can see ASD1_SGBTRF obtaining a speedup of up to 3.88× over the 'Best' LAPACK routine that could be used. If we look at ASD1_SPBTRF in the *Bottom* graph we can see it also achieves a maximum speedup of 4.85× versus SPBTRF.

5.3.5 Sparse Matrices

The final tests that were run on the algorithms were the selection of sparse matrices taken from the university of Florida Sparse matrix database. For determining which ASD and LAPACK routines to call two parameters were used:

- the positive definite property of the matrix determined whether to call SPOTRF (LAPACK, ASD1 and ASD2) and SPBTRF (LAPACK, ASD1)
- the property of $b \leq n/2$, where b is the bandwidth of the matrix and n is the size of the matrix, determined whether to call SGBTRF (LAPACK, ASD1) and SPBTRF (LAPACK, ASD1)

All speedup results presented refer to the improvement compared to the current LAPACK reference implementations that the ASD function would replace (Ex: ASD1_SGBTRF is compared vs SGBTRF). The results for the ten matrices selected can be seen represented in the Table 5.7.

Input	ASD1(GE)	ASD2(GE)	ASD1(GB)	ASD1(PO)	ASD2(PO)	ASD1(PB)
bcsstk28	19.16	21.08	2.20	38.73	7.89	2.44
bcsstk38	2.09	20.36	N/A	395.80	10.32	N/A
crystk01	23.42	25.05	0.94	14.13	8.91	0.97
ex40	19.01	20.10	1.03	N/A	N/A	N/A
goodwin	1.00	23.00	N/A	N/A	N/A	N/A
heart1	1.04	1.88	N/A	N/A	N/A	N/A
Kuu	3.26	26.21	N/A	15.51	7.30	N/A
Muu	3.24	27.21	N/A	14.86	7.32	N/A
nasa2910	4.54	6.38	1.75	4.75	3.59	1.89
nasa4704	19.40	20.19	1.55	15.31	8.93	1.80

Table 5.7. Speedup over current LAPACK routine implementation.

While many of the speedups on the sparse matrices are very large for most implementations, they should be compared to the time a sparse solver would take, which would have to

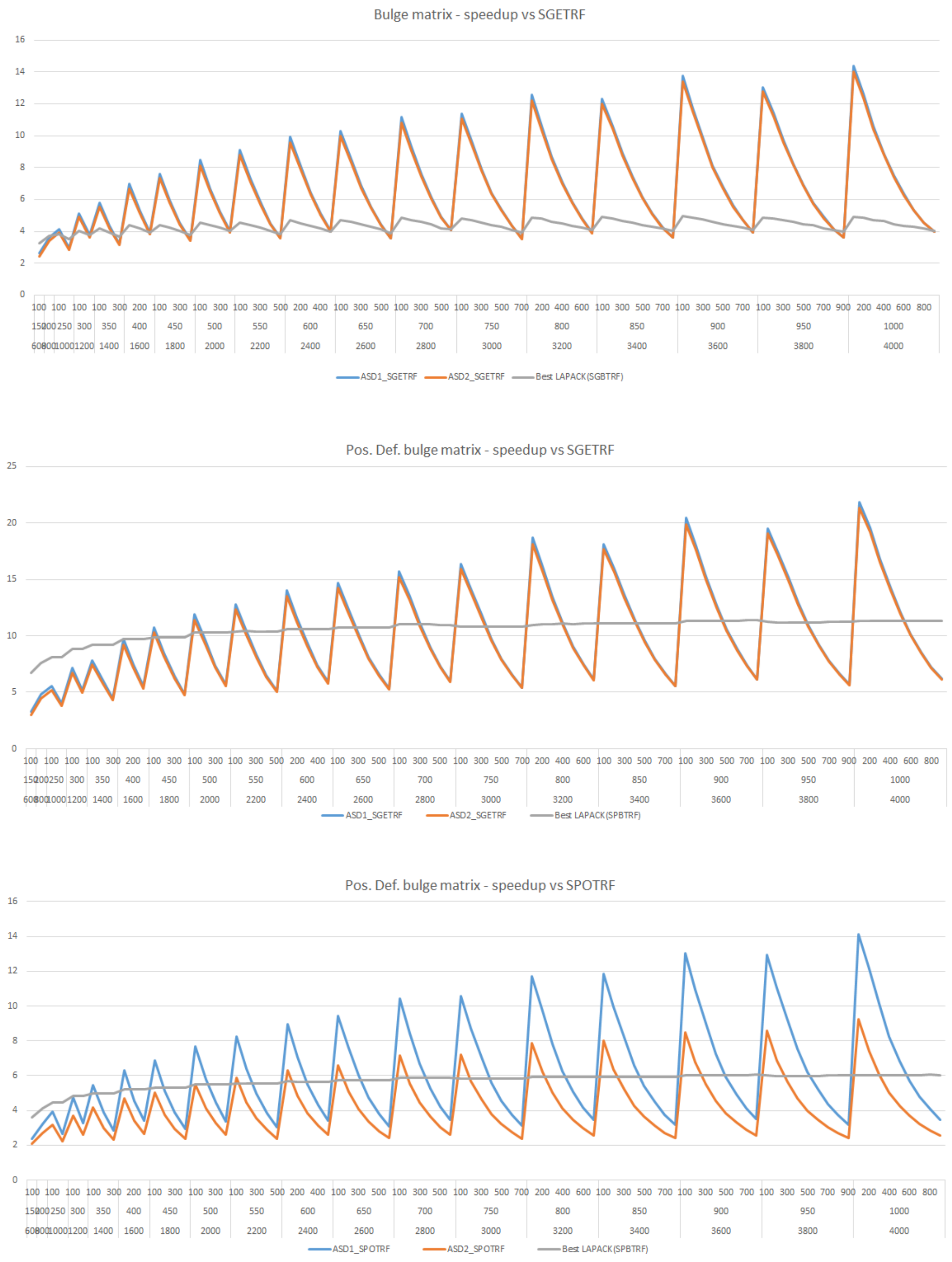


Figure 5.11. Speedup for bulge matrices with *Top:* ASDx_SGETRF vs SGETRF on random matrix (with pivoting) *Middle:* ASDx_SGETRF vs SGETRF on random Positive Definite matrix (without pivoting) *Bottom:* ASDx_SPOTRF vs SPOTRF on random Positive Definite matrix

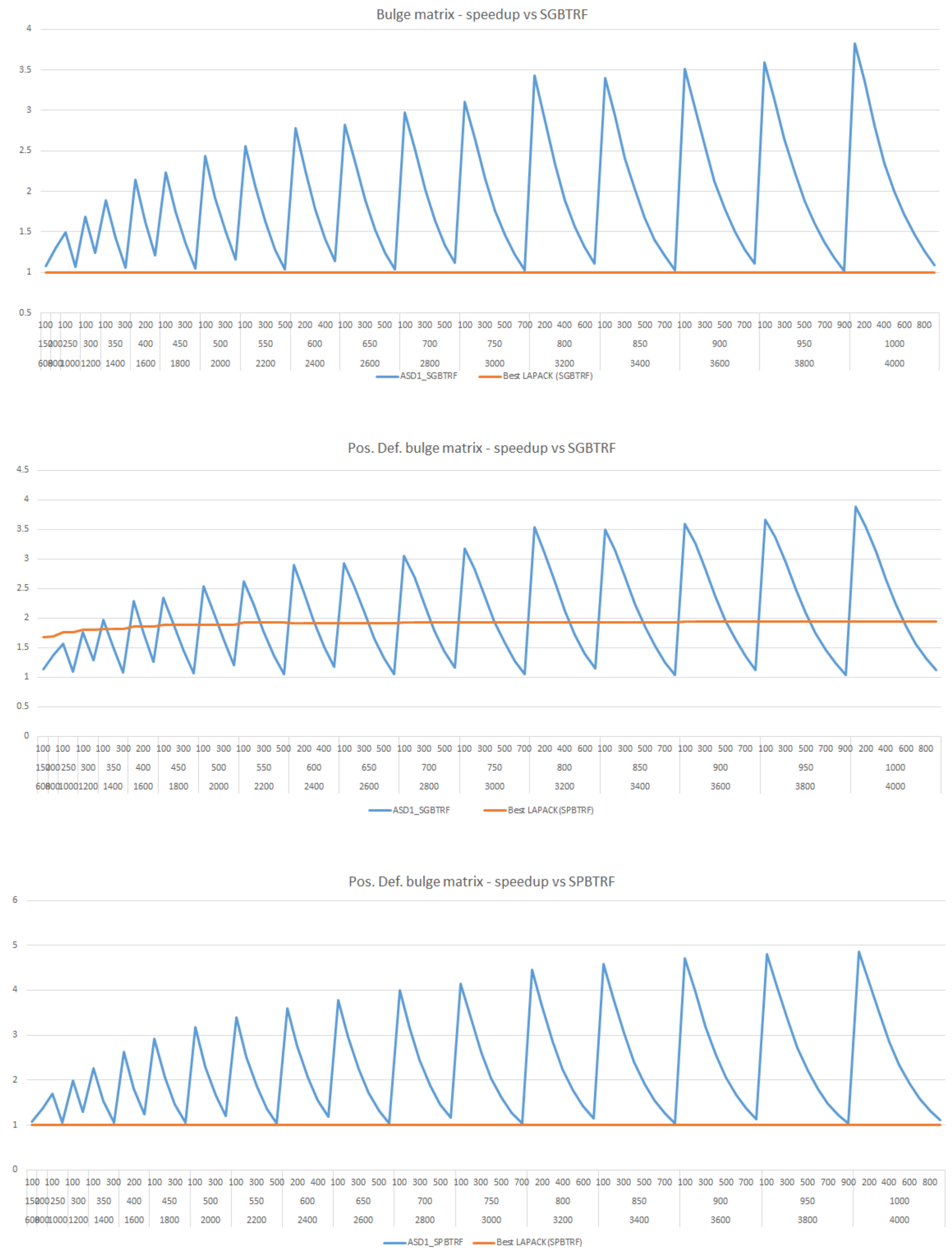


Figure 5.12. Speedup for bulge matrices with *Top:* ASD1_SGBTRF vs SGBTRF on random matrix (with pivoting) *Middle:* ASD1_SGBTRF vs SGBTRF on random Positive Definite matrix (without pivoting) *Bottom:* ASD1_SPBTRF vs SPBTRF on random Positive Definite matrix

include the time required to copy from the dense or band format to the format required by the sparse solver.

5.4 Correctness Check

For each of the matrix structures presented a random matrix A was generated, factorized and compared with the original matrix A and the LAPACK factorization. For the sparse matrices the input matrix A was read from a file and then factorized with both the ASD and LAPACK routines.

While exact reproducibility can not be assured, since LAPACK itself is not reproducible [6], we observed that for all our tests on the single platform the ASD routines performed factorizations whose results did not differ by more than $n \times eps$[1] in the single precision entries of the LU factors and that all pivot choices were maintained between the reference LAPACK and ASD routines. This can be expressed mathematically as:

- for all $i, j : |L(i,j) - L_{ASD}(i,j)| \leq n \times eps \times |L(i,j)|$
- for all $i, j : |U(i,j) - U_{ASD}(i,j)| \leq n \times eps \times |U(i,j)|$
- for all $i : ipiv(i) = ipiv_{ASD}(i)$

[1]$eps = 1e - 8$ and n is the size of the matrix

Chapter 6

Future Work

6.1 Integrating ASD Codes in LAPACK

In Chapter 5 we saw that the ASD1 codes present both very little overhead to no overhead and offer high benefits for many matrix structures. Since these codes have no interface changes and no extra workspace requirement they are a logical step for integration into LAPACK.

In order to complete this integration the stress tests that run on LAPACK installation have to be adapted from the correctness and performance tests implemented for this thesis. It is desirable also that initially the functions should exist separately from the reference LAPACK implementations, perhaps with the *asd_* modifier, so they can be tested thoroughly by the community before they proceed to replace the reference implementation and deprecate the modified name.

Once the testing code is integrated the functions can be easily expanded to all precisions, since there is nothing within any of the implementations that is specific to single-precision.

6.2 Further ASD Code Improvements

A simple improvement that would benefit all ASD algorithms would be the lowering of the blocking factor *nb* in cases where sparsity is detected since would allow for finer grained sparsity within a panel by xBZCHK or xBZCHK_ILIM. This parameter would have to be tuned with the balancing factor of wanting to maintain a high enough *nb* to allow attaining BLAS3 performance from the blocked operations.

Further improvement could be implemented into the xGETF2_ILIM routines to first remember the last block that was checked and using that for an initialization for the ISAMAX part of the algorithm. The top-left square check could also be avoided by initializing *ipiv* elements to the diagonal and then proceeding to only check and modify the locations within *ilim* exclusively.

As could be seen from the results section on ASD1_xPOTRF the 'left-looking' algorithm is able to provide speedup for formats such as the arrow matrix so another avenue of potential improvement is the implementation and comparison of the 'left-looking' algorithms for general LU (GE) , banded LU (GB) and banded Cholesky (PB).

It could also be beneficial to stratify the use of the ASD2 algorithm to lower the overhead to reasonable levels by either using a predetermined factor for the fraction of nonzeros for low values of n or by only applying ASD1 calls for low n. This perhaps could be implemented as an extension of the current LAPACK practice of calling the BLAS2 routines for very small values of n. The resulting implementation would then switch on the value of n so if: a) n is very small call BLAS2 b) n is small call ASD1 c) n is large call ASD2. The exact switch-over points would most likely require tuning for different machines.

For the ASD2 algorithms a change could be made to limit the maximum number of subblocks to a set number (for ex: 100). This would in turn get rid of the proportionality requirement for the array, thus allowing ASD2 to be implemented with no interface changes visible to the user. The downside of this potential implementation is that sparsity structures could be missed for matrices exhibiting a high sparsity which would suggest an initial split into more than 100 subblocks. An important distinction here is that this limits just the subblock number, not the limits in *ilim*, so any single rows or small sparsity elements could still be captured.

Also for ASD2 there are no particular algorithmic difficulties in applying the algorithm to the banded implementations but these require careful consideration especially in the face of potentially combined overheads between ASD2 and banded limitations.

Another possible improvement to the codes could come not to the factorizations themselves but from integration of the solvers and changes to the interface to allow for possible simpler formats for detected high sparsity matrices. Such a change could also add the ability to support column swaps within the factorization and therefore allow the implementation and test of ASD3.

The last suggestion for improvement that should be mentioned is that all of the mentioned ASD algorithms read a zero element at least twice: once during the zero detection and another time during multiplication. If this could be reduced to a single addressing during the implementation potential time savings could be accumulated.

6.3 Other Factorizations and ScaLAPACK

The modular implementation of the checks was always designed with the goal of extending this work to other factorizations and routines. The QR routines already have a simple algorithm similar to ASD1 but they could be simplified in their structure to use all the auxiliary routines provided for a more modular approach.

Besides the ‘one-sided factorizations’ (LU, QR) we can also look to expand the current sparsity detection and exploitation to ‘two-sided factorizations’ like SVD or eigenvalue decomposition. These algorithms require more analysis to check whether only the BLAS 3 parts can be changed to take advantage of the ASD or whether the BLAS 2 parts can also be modified to limit their action on the input matrix.

The methods present in the ASD algorithms can be extended to their ScaLAPACK equivalents pretty directly but a few new problems appear when considering their implementation. Firstly any checks on a panel will add at least $\log p$ communication overhead on top of any sparsity overhead. Once a limit or array of nonzero block limits is decided a method should be devised to avoid the potential load imbalance created by the zero blocks. Attention also needs to be placed to ensuring that for ASD2 the processor boundaries and the subblock boundaries line up and that no combinations between subblocks on different processors is allowed.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, et al. *LAPACK Users' Guide: Third Edition*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics, 1999.

[2] Ronald F Boisvert, Roldan Pozo, and Karin A Remington. The matrix market exchange formats: initial design. *National Institute of Standards and Technology Internal Report, NISTIR*, 5935, 1996.

[3] Jeremy Du Croz, Peter Mayes, and Guiseppe Radicati. Factorizations of band matrices using level 3 BLAS. Technical Report 21, LAPACK Working Note, July 1990.

[4] Timothy A. Davis. Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, June 2004.

[5] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[6] James Demmel and Hong Diep Nguyen. Fast reproducible floating-point summation. In *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, pages 163–172. IEEE, 2013.

[7] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.

[8] James W. Demmel, Yozo Hida, Mark F. Hoemmen, and E. Jason Riedy. Non-negative diagonals and high performance on low-profile matrices from householder QR. Technical Report 203, LAPACK Working Note, May 2008.

[9] James W. Demmel, Nick Higham, and Rob Schreiber. Block LU factorization. Technical Report 40, LAPACK Working Note, February 1992.

[10] X.S. Li, J.W. Demmel, J.R. Gilbert, iL. Grigori, M. Shao, and I. Yamazaki. SuperLU Users' Guide. Technical Report LBNL-44289, Lawrence Berkeley National Laboratory, September 1999. `http://crd.lbl.gov/~xiaoye/SuperLU/`. Last update: August 2011.

[11] Peter Mayes and Giuseppe Radicati. Banded cholosky factorization using level 3 BLAS. Technical Report 12, LAPACK Working Note, August 1989.

Appendix A

F77/90 source files

Below are provided the Fortran 77/90 codes for all of the utilized functions in ASD1 and ASD2 implementations.

A.1 Sparsity Checks

A.1.1 sbzchk.f

```
*> \brief \b SBZCHK
*
* Definition:
* ===========
*
* SUBROUTINE SBZCHK( M, N, A, LDA, DIR, ZLOW, ZHIGH)
*
* .. Scalar Arguments ..
* CHARACTER DIR
* INTEGER M, N, LDA, ZLOW, ZHIGH
* ..
* .. Array Arguments ..
* REAL A( LDA, *)
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> This function checks matrix A for complete zero rows or columns and
*> returns the index of the first non-zero row/column from both
*> directions in ZLOW and ZHIGH
*>
*> \endverbatim
*
* Arguments:
* ==========
*
```

```
*> \param[in] M
*> \verbatim
*> M is INTEGER
*> The number of rows of the matrix A. M >= 0.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The number of columns of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in] A
*> \verbatim
*> A is REAL array, dimension (LDA,N)
*> The M-by-N matrix to be checked for zeros
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> The leading dimension of the array A. LDA >= max(1,M).
*> \endverbatim
*>
*> \param[in] DIR
*> \verbatim
*> DIR is CHARACTER
*> The direction in which to search for zeros
*> 'r' or 'R' for row
*> 'c' or 'C' for column
*> \endverbatim
*>
*> \param[out] ZLOW
*> \verbatim
*> ZLOW is INTEGER
*> The index of the first non-zero row or 0 if empty matrix
*> \endverbatim
*>
*> \param[out] ZHIGH
*> \verbatim
*> ZHIGH is INTEGER
*> The index of the last non-zero row or 0 if empty matrix
*> \endverbatim

*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup auxOTHERauxiliary
*
* =====================================================================
      SUBROUTINE SBZCHK( M, N, A, LDA, DIR, ZLOW, ZHIGH)

* -- LAPACK auxiliary routine (verion TBD) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2014
*
* .. Scalar Arguments ..
      CHARACTER DIR
      INTEGER M, N, LDA, ZLOW, ZHIGH
```

```
* ..
* .. Array Arguments ..
      REAL A(LDA, *)
*
* ============================================================
*
* .. Local Scalars ..
      INTEGER I, J, ML, NL

*
* Check proper direction
*
      IF( LSAME(DIR,'R') ) THEN
*
* Check could be improved to take into account cache lines
*
      DO 10 I=M,0,-1
        ML = I
        IF (ML.EQ.0) THEN
          GO TO 30
        END IF
        DO 20 J=1,N
          IF (A(I,J).NE.0) THEN
            GO TO 30
          END IF
   20 CONTINUE
   10 CONTINUE
   30 ZHIGH = ML

      DO 11 I=1,ZHIGH,1
        ML = I
        DO 21 J=1,N
          IF (A(I,J).NE.0) THEN
            GO TO 31
          END IF
   21 CONTINUE
   11 CONTINUE
   31 ZLOW = ML

      ELSE IF ( LSAME(DIR,'C') ) THEN

      DO 40 J=N,0,-1
        NL = J
        IF (NL.EQ.0) THEN
          GO TO 60
        END IF
        DO 50 I=1,M
          IF (A(I,J).NE.0) THEN
            GO TO 60
          END IF
   50 CONTINUE
   40 CONTINUE
   60 ZHIGH = NL


      DO 41 J=1,ZHIGH,1
        NL = J
        DO 51 I=1,M
          IF (A(I,J).NE.0) THEN
            GO TO 61
          END IF
   51 CONTINUE
   41 CONTINUE
   61 ZLOW = NL

      ELSE
*
* Wrong direction entered, returning matrix
```

```
*
      ZLOW = −1
      ZHIGH = −1
      END IF
*
* End of SBZCHK
*
      END
```

A.1.2 sszchk.f

```
*> \brief \b SSZCHK
*
* Definition:
* ===========
*
* SUBROUTINE SSZCHK( M, N, A, LDA, SC, CTYPE, S, NNZ)
*
* .. Scalar Arguments ..
* CHARACTER*( * ) CTYPE
* INTEGER M, N, LDA, SC, NNZ
* REAL  S
* ..
* .. Array Arguments ..
* REAL A( LDA, *)
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> This function does a sparse O(m+n) check on matrix A to determine the amount of nonzeros and reports a guess on the sparsity of the matrix.
*>
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] M
*> \verbatim
*> M is INTEGER
*> The number of rows of the matrix A. M >= 0.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The number of columns of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in] A
*> \verbatim
*> A is REAL array, dimension (LDA,N)
*> The M−by−N matrix to be checked for zeros
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> The leading dimension of the array A. LDA >= max(1,M).
*> \endverbatim
*>
```

```
*> \param[in] SC
*> \verbatim
*> SC is INTEGER
*> Sparse checks multiplier which depending on CTYPE does SC*O(m+n) values.
*> \endverbatim
*>
*> \param[in] CTYPE
*> \verbatim
*> CTYPE is CHARACTER
*> which sparse pattern to use when checking for zeros
*> 'p' or 'P' for pure random
*> 'h' or 'H' for cached random (assumes A is cache alligned)
*> 'l' or 'L' for pure random in lower triangular part of A
*> 'u' or 'U' for pure random in upper triangular part of A
*> 'd' or 'D' for pure random + main diagonal
*> 'r' or 'R' for random row
*> 'c' or 'C' for random column
*> 'i' or 'I' for random increment
*> \endverbatim
*>
*> \param[out] S
*> \verbatim
*> S is REAL
*> The sparsity calculated as nnz/m*n
*> \endverbatim
*>
*> \param[out] NNZ
*> \verbatim
*> NNZ is INTEGER
*> Number of non-zero elements out of the SC*(m+n) checked
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup auxOTHERauxiliary
*
* =====================================================================
      SUBROUTINE SSZCHK( M, N, A, LDA, SC, CTYPE, S, NNZ)

* -- LAPACK auxiliary routine (version TBD) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2014
*
* .. Scalar Arguments ..
      CHARACTER CTYPE
      INTEGER M, N, LDA, SC, NNZ
      REAL S
* ..
* .. Array Arguments ..
      REAL A(LDA, *)
*
* =====================================================================
*
*
* .. External Functions ..
      LOGICAL LSAME
      EXTERNAL LSAME
* .. Local Scalars ..
      INTEGER I, J, K, L, NC, R, RI, RJ, R2
```

```
* ..
*
* Initializing values
      NNZ = 0
      NC = SC*(M+N)
*
* Resetting random number gen to make sure to select same elements
*
      R = IRAND(1)
*
* Switch on sparse pattern CTYPE to use for checking sparsity
*
      IF( LSAME(CTYPE,'P') ) THEN
* pure random checks on all matrix
*
      DO 10 K=1,NC,1
        R = IRAND(0)
        R = MOD(R,M*N)
        RI = MOD(R,LDA)+1
        RJ = R/LDA+1
        IF (A(RI,RJ).NE.0) THEN
          NNZ = NNZ + 1
        END IF
   10 CONTINUE
      S = REAL(NNZ) / NC

      ELSE IF ( LSAME(CTYPE,'H') ) THEN
* Check the all 4 values of the cache line
*
      DO 20 K=1,NC,1
        R = IRAND(0)
        R = MOD(R,M*N)
                R = (R/4)*4
        DO 30 L=1,4
          R2 = R+L-1
          RI = MOD(R2,LDA)+1
          RJ = R2/LDA+1
          IF (A(RI,RJ).NE.0) THEN
            NNZ = NNZ + 1
          END IF
   30 CONTINUE
   20 CONTINUE
      S = REAL(NNZ) / ( NC*4)

      ELSE IF ( LSAME(CTYPE,'D') ) THEN
* pure random checks + all diagonal values
*
      DO 40 K=1,NC,1
        R = IRAND(0)
        R = MOD(R,M*N)
        RI = MOD(R,LDA)+1
        RJ = R/LDA+1
        IF (A(RI,RJ).NE.0) THEN
          NNZ = NNZ + 1
        END IF
   40 CONTINUE
      DO 50 I=1,MIN(M,N)
        IF (A(I,I).NE.0) THEN
          NNZ = NNZ + 1
        END IF
   50 CONTINUE
      S = REAL(NNZ) / (NC+ MIN(M,N))

      ELSE IF ( LSAME(CTYPE,'L') ) THEN
* pure random checks on lower triangular values
*
      DO 41 K=1,NC,1
        R = IRAND(0)
```

```
         R = MOD(R,M*N)
         RI = MOD(R,LDA)+1
         RJ = R/LDA+1
         IF (RI.GT.RJ) THEN
            IF (A(RI,RJ).NE.0) THEN
               NNZ = NNZ + 1
            END IF
         ELSE
            IF (A(RJ,RI).NE.0) THEN
               NNZ = NNZ + 1
            END IF
         END IF
   41 CONTINUE
      S = REAL(NNZ) / (NC)

      ELSE IF ( LSAME(CTYPE,'U') ) THEN
* pure random checks on upper triangular values
*
      DO 51 K=1,NC,1
         R = IRAND(0)
         R = MOD(R,M*N)
         RI = MOD(R,LDA)+1
         RJ = R/LDA+1
         IF (RI.LT.RJ) THEN
            IF (A(RI,RJ).NE.0) THEN
               NNZ = NNZ + 1
            END IF
         ELSE
            IF (A(RJ,RI).NE.0) THEN
               NNZ = NNZ + 1
            END IF
         END IF
   51 CONTINUE
      S = REAL(NNZ) / (NC)

      ELSE IF ( LSAME(CTYPE,'R')) THEN
* in each row check SC values
*
      DO 60 I=1,M
         DO 70 K=1,SC
         R = IRAND(0)
         R = MOD(R,N)+1
         IF (A(I,R).NE.0) THEN
            NNZ = NNZ + 1
         END IF
   70 CONTINUE
   60 CONTINUE
      S = REAL(NNZ) / (SC*M)

      ELSE IF ( LSAME(CTYPE,'C')) THEN
* in each column check SC values
*
      DO 80 J=1,N
         DO 90 K=1,SC
         R = IRAND(0)
         R = MOD(R,M)+1
         IF (A(R,J).NE.0) THEN
            NNZ = NNZ + 1
         END IF
   90 CONTINUE
   80 CONTINUE
      S = REAL(NNZ) / (SC*N)

      ELSE IF ( LSAME(CTYPE,'I')) THEN
* choosing small random increment
*
      R2 = 0
      DO 11 K=1,NC,1
```

```
         R = IRAND(0)
         R = MOD(R,(M*N)/NC)
         R2 = R2 + R
         RI = MOD(R2,LDA)+1
         RJ = R2/LDA+1
         IF (A(RI,RJ).NE.0) THEN
           NNZ = NNZ + 1
         END IF
   11 CONTINUE
       S = REAL(NNZ) / NC

       ELSE
* Incorrect pattern entered, returning full matrix
*
         S = 1.0;
       END IF
*
* End of SSZCHK
*
       END
```

A.2 Auxiliary Routines

A.2.1 izss.f

```
*> \brief \b IZSS
*
* Definition:
* ===========
*
* INTEGER FUNCTION IZSS( N, LDA, FRACNZ, MINLVL)
*
* .. Scalar Arguments ..
* INTEGER N, LDA, MINLVL
* REAL FRACNZ
* ..
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> This function selects the number of blocks to use based on
*> the sparsity and the size of the matrix
*>
*> IZSS returns an INTEGER
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The size of matrix A in the direction being considered. N >= 0.
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> The leading dimension of the array A. LDA >= max(1,M).
*> \endverbatim
```

```
*>
*> \param[in] MINLVL
*> \verbatim
*> MINLVL is INTEGER
*> The smallest size for a block
*> \endverbatim
*>
*> \param[in] FRACNZ
*> \verbatim
*> FRACNZ is REAL
*> The estimated sparsity of the matrix
*> \endverbatim
*>

*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup auxOTHERauxiliary
*
* =====================================================================
      INTEGER FUNCTION IZSS( N, LDA, FRACNZ, MINLVL)
* -- LAPACK auxiliary routine (version TBD) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* August 2014
*
* .. Scalar Arguments ..
      INTEGER N, LDA, MINLVL
      REAL FRACNZ
* ..
*
* =====================================================================
*
* .. Local Scalars ..
      INTEGER MIS1,MIS2
* ..
* .. Intrinsic Functions ..
      INTRINSIC MIN,MAX
* ..
*
* Check if fracnz is zero
*
      IF (N.EQ.0) THEN
        IZSS = 0
      ELSE IF( FRACNZ.EQ.0) THEN
*
* If all checked values were zero select as many blocks of LVL size as possible
*
        IZSS = MAX(N/MINLVL,1)
      ELSE
*
* Select between minimum levels and number of blocks suggested by sparsity
*
        MIS1 = INT(1.0/FRACNZ) + 1
        MIS2 = MAX(N/MINLVL,1)
        IZSS = MIN(MIS1,MIS2)
      END IF
*
* End of IZSS
*
```

```
      END
```

A.2.2 izcomb.f

```
*> \brief \b IZCOMB
*
* Definition:
* ===========
*
* SUBROUTINE FUNCTION IZCOMB( ILIM, IS, MINCOMB)
*
* .. Scalar Arguments ..
* INTEGER IS, MINCOMB
* ..
* .. Array Arguments ..
* INTEGER ILIM( *)
* ..
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> This function combines the results in ILIM
*>
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in/out] ILIM
*> \verbatim
*> ILIM is INTEGER array, dimension (N)
*> The array of non-zero start/end positions
*> \endverbatim
*>
*> \param[in/out] IS
*> \verbatim
*> IS is INTEGER
*> The number of blocks
*> \endverbatim
*>
*> \param[in] MINCOMB
*> \verbatim
*> MINCOMB is INTEGER
*> The number of zero rows/columns to ignore when combining
*> \endverbatim
*>

*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup auxOTHERauxiliary
*
* =====================================================================
      INTEGER FUNCTION IZCOMB( ILIM, IS, MINCOMB)
```

```
* -- LAPACK auxiliary routine (version TBD) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* August 2014
*
* .. Scalar Arguments ..
      INTEGER IS, MINCOMB
* ..
* .. Array Arguments ..
      INTEGER ILIM( *)
*
* =====================================================================
*
* .. Local Scalars ..
      INTEGER NIS,PREHIGH
* ..

*
* First get rid of all 0 sub-matrices
*
      NIS = 0
      DO 10 I=1,IS
        IF ( ILIM(2*(I-1)+1).NE.0) THEN
          NIS = NIS + 1
          ILIM(2*(NIS-1)+1) = ILIM(2*(I-1)+1)
          ILIM(2*(NIS-1)+2) = ILIM(2*(I-1)+2)
        END IF
   10 CONTINUE
      IS = NIS

*
* Now combine row/column blocks less then MINCOMB apart
*
      IF (IS.GT.0) THEN
        NIS = 1
        PREHIGH = ILIM(2)
        DO 20 I=2,IS
          IF ( (ILIM(2*(I-1)+1) - PREHIGH -1) .LE. MINCOMB) THEN
            PREHIGH = ILIM(2*(I-1)+2)
          ELSE
                    ILIM(2*(NIS-1)+2) = PREHIGH
            NIS = NIS + 1
            ILIM(2*(NIS-1)+1) = ILIM(2*(I-1)+1)
            PREHIGH = ILIM(2*(I-1)+2)
          END IF
   20 CONTINUE
        ILIM(2*(NIS-1)+2) = PREHIGH
      END IF
          IS = NIS
*
* End of IZCOMB
*
      END
```

A.2.3 izsub.f

```
*> \brief \b IZSUB
*
* Definition:
* ===========
*
* SUBROUTINE FUNCTION IZSUB( ILIM, IS, BS)
*
* .. Scalar Arguments ..
* INTEGER IS, BS
* ..
* .. Array Arguments ..
* INTEGER ILIM( *)
* ..
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> This function subtracts JB rows/columns from ILIM
*>
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in/out] ILIM
*> \verbatim
*> ILIM is INTEGER array, dimension (N)
*> The array of non-zero start/end positions
*> \endverbatim
*>
*> \param[in/out] IS
*> \verbatim
*> IS is INTEGER
*> The number of blocks
*> \endverbatim
*>
*> \param[in] BS
*> \verbatim
*> BS is INTEGER
*> The number of zero rows/columns to subtract
*> \endverbatim
*>
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup auxOTHERauxiliary
*
* =====================================================================
      INTEGER FUNCTION IZSUB( ILIM, IS, BS)
* -- LAPACK auxiliary routine (version TBD) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* August 2014
*
* .. Scalar Arguments ..
      INTEGER IS, BS
* ..
* .. Array Arguments ..
      INTEGER ILIM( *)
*
* =====================================================================
*
*
* .. External Functions ..
      LOGICAL LSAME
      EXTERNAL LSAME
```

```
* ..
* .. Intrinsic Functions ..
      INTRINSIC MAX, MIN
* ..
* .. Local Scalars ..
      INTEGER NIS,PREHIGH
* ..

      NIS = 0
      DO 10 I=1,IS
        ILIM(2*(I−1)+1) = ILIM(2*(I−1)+1)−BS
        ILIM(2*(I−1)+2) = ILIM(2*(I−1)+2)−BS
   10 CONTINUE
      DO 20 I=1,IS
        IF (ILIM(2*(I−1)+2).GT.0) THEN
          NIS = NIS + 1
          ILIM(2*(NIS−1)+1) = MAX(ILIM(2*(I−1)+1),1)
          ILIM(2*(NIS−1)+2) = ILIM(2*(I−1)+2)
        END IF
   20 CONTINUE
      IS = NIS
*
* End of IZSUB
*
      END
```

A.2.4 sbzchk_ilim.f

```
*> \brief \b SBZCHK_ILIM
*
* Definition:
* ===========
*
* SUBROUTINE SBZCHK_ILIM( M, N, A, LDA, DIR, ILIM, IS)
*
* .. Scalar Arguments ..
* CHARACTER DIR
* INTEGER M, N, LDA, IS
* ..
* .. Array Arguments ..
* INTEGER ILIM( *)
* REAL A( LDA, *)
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> This function runs multiple SBZCHK on the panel and saves
*> the results into ILIM
*>
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] M
*> \verbatim
*> M is INTEGER
*> The number of rows of the matrix A. M >= 0.
*> \endverbatim
*>
```

```
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The number of columns of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in] A
*> \verbatim
*> A is REAL array, dimension (LDA,N)
*> The M-by-N matrix to be checked for zeros
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> The leading dimension of the array A. LDA >= max(1,M).
*> \endverbatim
*>
*> \param[in] DIR
*> \verbatim
*> DIR is CHARACTER
*> The direction in which to search for zeros
*> 'r' or 'R' for row
*> 'c' or 'C' for column
*> \endverbatim
*>
*> \param[in] IS
*> \verbatim
*> IS is INTEGER
*> The number of blocks
*> \endverbatim
*>
*> \param[out] ILIM
*> \verbatim
*> ILIM is INTEGER array, dimension (N)
*> The array of non-zero start/end positions
*> \endverbatim

*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup auxOTHERauxiliary
*
* =====================================================================
      SUBROUTINE SBZCHK_ILIM( M, N, A, LDA, DIR, ILIM, IS)

* -- LAPACK auxiliary routine (verion TBD) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2014
*
* .. Scalar Arguments ..
      CHARACTER DIR
      INTEGER M, N, LDA, IS
* ..
* .. Array Arguments ..
      INTEGER ILIM( *)
      REAL A(LDA, *)
*
* =====================================================================
*
* .. Local Scalars ..
      INTEGER I, STEP, CURI

*
* Check proper direction
*
      IF( LSAME(DIR,'R') ) THEN

*
* Calculate the size of each block
*
      STEP = M/IS
      CURI = 1

* For each block
      DO 10 I=1,IS−1
*
* Run SBZCHK on submatrix
        CALL SBZCHK( STEP, N, A(CURI,1), LDA, 'row', ILIM(2*(I−1)+1),
     $ ILIM(2*(I−1)+2) )
*
* If submatrix isn't empty adjust indeces
        IF ( ILIM(2*(I−1)+1).NE.0) THEN
           ILIM(2*(I−1)+1) = ILIM(2*(I−1)+1) + CURI −1
           ILIM(2*(I−1)+2) = ILIM(2*(I−1)+2) + CURI −1
        END IF
*
*  Incrementing current I position
        CURI = CURI + STEP
   10 CONTINUE
*
* Running last submatrix
      CALL SBZCHK(M−CURI+1, N, A(CURI,1), LDA, 'row', ILIM(2*(IS−1)+1),
     $ ILIM(2*(IS−1)+2) )
*
* If submatrix isn't empty adjust indeces
      IF ( ILIM(2*(IS−1)+1).NE.0) THEN
         ILIM(2*(IS−1)+1) = ILIM(2*(IS−1)+1) + CURI −1
         ILIM(2*(IS−1)+2) = ILIM(2*(IS−1)+2) + CURI −1
      END IF

      ELSE IF ( LSAME(DIR,'C') ) THEN



*
* Calculate the size of each block
*
      STEP = N/IS
      CURI = 1

* For each block
      DO 20 I=1,IS−1
*
* Run SBZCHK on submatrix
        CALL SBZCHK( M, STEP, A(1,CURI), LDA, 'col', ILIM(2*(I−1)+1),
     $ ILIM(2*(I−1)+2) )
*
* If submatrix isn't empty adjust indeces
        IF ( ILIM(2*(I−1)+1).NE.0) THEN
           ILIM(2*(I−1)+1) = ILIM(2*(I−1)+1) + CURI −1
           ILIM(2*(I−1)+2) = ILIM(2*(I−1)+2) + CURI −1
        END IF
*
*  Incrementing current I position
        CURI = CURI + STEP
   20 CONTINUE
*
```

```
* Running last submatrix
        CALL SBZCHK(M, N−CURI+1, A(1,CURI), LDA, 'col', ILIM(2*(IS−1)+1),
      $ ILIM(2*(IS−1)+2) )
*
* If submatrix isn't empty adjust indeces
        IF ( ILIM(2*(IS−1)+1).NE.0) THEN
            ILIM(2*(IS−1)+1) = ILIM(2*(IS−1)+1) + CURI −1
            ILIM(2*(IS−1)+2) = ILIM(2*(IS−1)+2) + CURI −1
        END IF

        ELSE
*
* Wrong direction entered, returning matrix
*
        END IF
*
* End of SBZCHK_ILIM
*
        END
```

A.2.5 sgetf2_ilim.f

```
*> \brief \b SGETF2_ILIM computes the LU factorization of a general m−by−n matrix using partial pivoting with row interchanges (unblocked algorith
*
* =========== DOCUMENTATION ===========
*
* Online html documentation available at
* http://www.netlib.org/lapack/explore−html/
*
*> \htmlonly
*> Download SGETF2 + dependencies
*> <a href="http://www.netlib.org/cgi−bin/netlibfiles.tgz?format=tgz&filename=/lapack/lapack_routine/sgetf2.f">
*> [TGZ]</a>
*> <a href="http://www.netlib.org/cgi−bin/netlibfiles.zip?format=zip&filename=/lapack/lapack_routine/sgetf2.f">
*> [ZIP]</a>
*> <a href="http://www.netlib.org/cgi−bin/netlibfiles.txt?format=txt&filename=/lapack/lapack_routine/sgetf2.f">
*> [TXT]</a>
*> \endhtmlonly
*
* Definition:
* ===========
*
* SUBROUTINE SGETF2( M, N, A, LDA, IPIV, ILIM, IS, INFO )
*
* .. Scalar Arguments ..
* INTEGER INFO, LDA, M, N, IS
* ..
* .. Array Arguments ..
* INTEGER IPIV( * ), ILIM ( * )
* REAL A( LDA, * )
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> SGETF2 computes an LU factorization of a general m−by−n matrix A
*> using partial pivoting with row interchanges.
*>
*> The factorization has the form
*> A = P * L * U
*> where P is a permutation matrix, L is lower triangular with unit
*> diagonal elements (lower trapezoidal if m > n), and U is upper
*> triangular (upper trapezoidal if m < n).
*>
*> This is the right−looking Level 2 BLAS version of the algorithm.
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] M
*> \verbatim
*> M is INTEGER
*> The number of rows of the matrix A. M >= 0.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The number of columns of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in,out] A
*> \verbatim
*> A is REAL array, dimension (LDA,N)
*> On entry, the m by n matrix to be factored.
*> On exit, the factors L and U from the factorization
*> A = P*L*U; the unit diagonal elements of L are not stored.
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> The leading dimension of the array A. LDA >= max(1,M).
*> \endverbatim
*>
*> \param[in] IS
*> \verbatim
*> IS is INTEGER
*> The number of row blocks
*> \endverbatim
*>
*> \param[in] ILIM
*> \verbatim
*> ILIM is INTEGER array, dimension (N)
*> The array of non−zero start/end row positions
*> \endverbatim
*>
*> \param[out] IPIV
*> \verbatim
*> IPIV is INTEGER array, dimension (min(M,N))
*> The pivot indices; for 1 <= i <= min(M,N), row i of the
*> matrix was interchanged with row IPIV(i).
*> \endverbatim
*>
*> \param[out] INFO
*> \verbatim
*> INFO is INTEGER
*> = 0: successful exit
*> < 0: if INFO = −k, the k−th argument had an illegal value
*> > 0: if INFO = k, U(k,k) is exactly zero. The factorization
*> has been completed, but the factor U is exactly
*> singular, and division by zero will occur if it is used
*> to solve a system of equations.
*> \endverbatim
*
* Authors:
* ========
*
```

```
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup realGEcomputational_ASD
*
* =====================================================================
      SUBROUTINE SGETF2_ILIM( M, N, A, LDA, IPIV, ILIM, IS, INFO )
*
* -- LAPACK computational routine (version 3.4.2) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* September 2012
*
* .. Scalar Arguments ..
      INTEGER INFO, LDA, M, N, IS
* ..
* .. Array Arguments ..
      INTEGER IPIV( * ), ILIM ( * )
      REAL A( LDA, * )
* ..
*
* =====================================================================
*
* .. Parameters ..
      REAL ONE, ZERO
      PARAMETER ( ONE = 1.0E+0, ZERO = 0.0E+0 )
* ..
* .. Local Scalars ..
      REAL SFMIN, CURMAX
      INTEGER I, J, JP, K, KLOW, KHIGH, MINMN
* ..
* .. External Functions ..
      REAL SLAMCH
      INTEGER ISAMAX
      EXTERNAL SLAMCH, ISAMAX
* ..
* .. External Subroutines ..
      EXTERNAL SGER, SSCAL, SSWAP, XERBLA
* ..
* .. Intrinsic Functions ..
      INTRINSIC MAX, MIN, ABS
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'SGETF2_ILIM', -INFO )
         RETURN
      END IF
*
* Quick return if possible
*
      IF( M.EQ.0 .OR. N.EQ.0 )
     $ RETURN
*
```

```
* Compute machine safe minimum
*
      SFMIN = SLAMCH('S')
*
      MINMN = MIN(M,N)
      DO 10 J = 1, MINMN
*
* Find pivot and test for singularity.
* First check 1:MINMN (top square) and add to that all ILIM positions
*
         JP = J − 1 + ISAMAX( MINMN−J+1, A( J, J ), 1 )
         CURMAX = ABS(A(JP,J))
         KP = JP
         DO 30 K=1,IS
           KLOW = ILIM(2∗(K−1)+1)
           KHIGH = ILIM(2∗(K−1)+2)
           IF (KLOW.GT.MINMN) THEN
             KP = KLOW−1 + ISAMAX( KHIGH − KLOW +1, A( KLOW, J ), 1 )
           ELSE IF (KHIGH.GT.MINMN) THEN
                     KP = MINMN + ISAMAX( KHIGH − MINMN, A( MINMN +1 , J ), 1 )
           END IF
           IF (ABS(A(KP,J)).GT.CURMAX) THEN
             JP = KP
             CURMAX = ABS(A(KP,J))
           END IF
   30 CONTINUE
         IPIV( J ) = JP
         IF( A( JP, J ).NE.ZERO ) THEN
*
* Apply the interchange to columns 1:N.
*
            IF( JP.NE.J )
     $ CALL SSWAP( N, A( J, 1 ), LDA, A( JP, 1 ), LDA )
*
* Compute elements J+1:M of J−th column in ILIM positions
*
            IF( J.LT.M ) THEN
               IF( ABS(A( J, J )) .GE. SFMIN ) THEN
                 DO 40 K=1,IS
                   KLOW = ILIM(2∗(K−1)+1)
                   KHIGH = ILIM(2∗(K−1)+2)
                                   IF (KLOW.GT.J) THEN
                      CALL SSCAL( KHIGH−KLOW+1, ONE / A( J, J ),
     $     A( KLOW, J ), 1 )
                   ELSE IF (KHIGH.GT.J) THEN
                      CALL SSCAL( KHIGH−J, ONE / A( J, J ),
     $     A( J+1, J ), 1 )
                   END IF
   40 CONTINUE
               ELSE
                 DO 50 K=1,IS
                   KLOW = ILIM(2∗(K−1)+1)
                   KHIGH = ILIM(2∗(K−1)+2)
                                   IF (KLOW.GT.J) THEN
                     DO 60 I = 1, KHIGH−KLOW+1
                       A( KLOW−1+I, J ) = A( KLOW−1+I, J ) / A( J, J )
   60 CONTINUE
                   ELSE IF (KHIGH.GT.J) THEN
                     DO 70 I = 1, KHIGH−J
                       A( J+I, J ) = A( J+I, J ) / A( J, J )
   70 CONTINUE
                   END IF
   50 CONTINUE
               END IF
            END IF
*
         ELSE IF( INFO.EQ.0 ) THEN
*
```

```
            INFO = J
         END IF
*
         IF( J.LT.MINMN ) THEN
*
* Update trailing submatrix via multiple calls to SGER.
*
            DO 80 K=1,IS
               KLOW = ILIM(2*(K-1)+1)
               KHIGH = ILIM(2*(K-1)+2)
               IF (KLOW.GT.J) THEN
                  CALL SGER( KHIGH-KLOW+1, N-J, -ONE, A( KLOW, J ), 1,
     $ A( J, J+1 ), LDA, A( KLOW, J+1 ), LDA )
               ELSE IF (KHIGH.GT.J) THEN
                  CALL SGER( KHIGH-J, N-J, -ONE, A( J+1, J ), 1,
     $ A( J, J+1 ), LDA, A( J+1, J+1 ), LDA )
               END IF
   80 CONTINUE
         END IF
   10 CONTINUE
      RETURN
*
* End of SGETF2
*
      END
```

A.2.6 strsm_ilim.f

```
*> \brief \b STRSM_ILIM
*
* =========== DOCUMENTATION ===========
*
* Online html documentation available at
* http://www.netlib.org/lapack/explore-html/
*
* Definition:
* ===========
*
* SUBROUTINE STRSM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB,ILIM,IS)
*
* .. Scalar Arguments ..
* REAL ALPHA
* INTEGER LDA,LDB,M,N,IS
* CHARACTER DIAG,SIDE,TRANSA,UPLO
* ..
* .. Array Arguments ..
* INTEGER ILIM(*)
* REAL A(LDA,*),B(LDB,*)
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> STRSM solves one of the matrix equations
*>
*> op( A )*X = alpha*B, or X*op( A ) = alpha*B,
*>
*> where alpha is a scalar, X and B are m by n matrices, A is a unit, or
*> non-unit, upper or lower triangular matrix and op( A ) is one of
*>
*> op( A ) = A or op( A ) = A**T.
```

```
*>
*> The matrix X is overwritten on B.
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] SIDE
*> \verbatim
*> SIDE is CHARACTER*1
*> On entry, SIDE specifies whether op( A ) appears on the left
*> or right of X as follows:
*>
*> SIDE = 'L' or 'l' op( A )*X = alpha*B.
*>
*> SIDE = 'R' or 'r' X*op( A ) = alpha*B.
*> \endverbatim
*>
*> \param[in] UPLO
*> \verbatim
*> UPLO is CHARACTER*1
*> On entry, UPLO specifies whether the matrix A is an upper or
*> lower triangular matrix as follows:
*>
*> UPLO = 'U' or 'u' A is an upper triangular matrix.
*>
*> UPLO = 'L' or 'l' A is a lower triangular matrix.
*> \endverbatim
*>
*> \param[in] TRANSA
*> \verbatim
*> TRANSA is CHARACTER*1
*> On entry, TRANSA specifies the form of op( A ) to be used in
*> the matrix multiplication as follows:
*>
*> TRANSA = 'N' or 'n' op( A ) = A.
*>
*> TRANSA = 'T' or 't' op( A ) = A**T.
*>
*> TRANSA = 'C' or 'c' op( A ) = A**T.
*> \endverbatim
*>
*> \param[in] DIAG
*> \verbatim
*> DIAG is CHARACTER*1
*> On entry, DIAG specifies whether or not A is unit triangular
*> as follows:
*>
*> DIAG = 'U' or 'u' A is assumed to be unit triangular.
*>
*> DIAG = 'N' or 'n' A is not assumed to be unit
*> triangular.
*> \endverbatim
*>
*> \param[in] M
*> \verbatim
*> M is INTEGER
*> On entry, M specifies the number of rows of B. M must be at
*> least zero.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> On entry, N specifies the number of columns of B. N must be
*> at least zero.
*> \endverbatim
*>
```

```
*> \param[in] ALPHA
*> \verbatim
*> ALPHA is REAL
*> On entry, ALPHA specifies the scalar alpha. When alpha is
*> zero then A is not referenced and B need not be set before
*> entry.
*> \endverbatim
*>
*> \param[in] A
*> \verbatim
*> A is REAL array of DIMENSION ( LDA, k ),
*> where k is m when SIDE = 'L' or 'l'
*> and k is n when SIDE = 'R' or 'r'.
*> Before entry with UPLO = 'U' or 'u', the leading k by k
*> upper triangular part of the array A must contain the upper
*> triangular matrix and the strictly lower triangular part of
*> A is not referenced.
*> Before entry with UPLO = 'L' or 'l', the leading k by k
*> lower triangular part of the array A must contain the lower
*> triangular matrix and the strictly upper triangular part of
*> A is not referenced.
*> Note that when DIAG = 'U' or 'u', the diagonal elements of
*> A are not referenced either, but are assumed to be unity.
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> On entry, LDA specifies the first dimension of A as declared
*> in the calling (sub) program. When SIDE = 'L' or 'l' then
*> LDA must be at least max( 1, m ), when SIDE = 'R' or 'r'
*> then LDA must be at least max( 1, n ).
*> \endverbatim
*>
*> \param[in,out] B
*> \verbatim
*> B is REAL array of DIMENSION ( LDB, n ).
*> Before entry, the leading m by n part of the array B must
*> contain the right-hand side matrix B, and on exit is
*> overwritten by the solution matrix X.
*> \endverbatim
*>
*> \param[in] LDB
*> \verbatim
*> LDB is INTEGER
*> On entry, LDB specifies the first dimension of B as declared
*> in the calling (sub) program. LDB must be at least
*> max( 1, m ).
*> \endverbatim
*>
*> \param[in] IS
*> \verbatim
*> IS is INTEGER
*> The number of row blocks if SIDE='r'
*> The number of column blocks if SIDE='l'
*> \endverbatim
*>
*> \param[in] ILIM
*> \verbatim
*> ILIM is INTEGER array, dimension (2*M or 2*N)
*> The array of non-zero start/end row positions if side='r'
*> The array of non-zero start/end column positions if side='l'
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup realGEcomputational_ASD
*
*> \par Further Details:
* =====================
*>
*> \verbatim
*>
*> Level 3 Blas routine.
*>
*>
*> -- Written on 8-February-1989.
*> Jack Dongarra, Argonne National Laboratory.
*> Iain Duff, AERE Harwell.
*> Jeremy Du Croz, Numerical Algorithms Group Ltd.
*> Sven Hammarling, Numerical Algorithms Group Ltd.
*> \endverbatim
*>
* =====================================================================
      SUBROUTINE STRSM_ILIM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB
     $ ,ILIM,IS)
*
* -- Reference BLAS level3 routine (version 3.4.0) --
* -- Reference BLAS is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2011
*
* .. Scalar Arguments ..
      REAL ALPHA
      INTEGER LDA,LDB,M,N,IS
      CHARACTER DIAG,SIDE,TRANSA,UPLO
* ..
* .. Array Arguments ..
      INTEGER ILIM(*)
      REAL A(LDA,*),B(LDB,*)
* ..
*
* =====================================================================
*
* .. External Functions ..
      LOGICAL LSAME
      EXTERNAL LSAME
* ..
* .. External Subroutines ..
      EXTERNAL STRSM
* ..
* .. Local Scalars ..
      INTEGER K,KLOW,KHIGH
      LOGICAL LSIDE
* ..

* Needs fix for error messages.

*
* Test the input parameters.
*
      LSIDE = LSAME(SIDE,'L')
*
* Start the operations.
*

      IF (LSIDE) THEN
        DO 10 K=1,IS
```

```
         KLOW=ILIM(2*(K-1)+1)
         KHIGH=ILIM(2*(K-1)+2)
         CALL STRSM(SIDE,UPLO,TRANSA,DIAG,M,KHIGH-KLOW+1,ALPHA,A,LDA,
     $ B(1,KLOW),LDB)
   10 CONTINUE
      ELSE
        DO 20 K=1,IS
         KLOW=ILIM(2*(K-1)+1)
         KHIGH=ILIM(2*(K-1)+2)
         CALL STRSM(SIDE,UPLO,TRANSA,DIAG,KHIGH-KLOW,N,ALPHA,A,LDA,
     $ B(KLOW,1),LDB)
   20 CONTINUE
      END IF

      RETURN
*
* End of STRSM .
*
      END
```

A.2.7 sgemm_ilim.f

```
*> \brief \b SGEMM_ILIM
*
* =========== DOCUMENTATION ===========
*
* Online html documentation available at
* http://www.netlib.org/lapack/explore-html/
*
* Definition:
* ===========
*
* SUBROUTINE SGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC,MILIM,MIS,NLIM,NIS)
*
* .. Scalar Arguments ..
* REAL ALPHA,BETA
* INTEGER K,LDA,LDB,LDC,M,N,MIS,NIS
* CHARACTER TRANSA,TRANSB
* ..
* .. Array Arguments ..
* REAL A(LDA,*),B(LDB,*),C(LDC,*)
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> SGEMM performs one of the matrix-matrix operations
*>
*> C := alpha*op( A )*op( B ) + beta*C,
*>
*> where op( X ) is one of
*>
*> op( X ) = X or op( X ) = X**T,
*>
*> alpha and beta are scalars, and A, B and C are matrices, with op( A )
*> an m by k matrix, op( B ) a k by n matrix and C an m by n matrix.
*> \endverbatim
*
* Arguments:
* ==========
*
```

```
*> \param[in] TRANSA
*> \verbatim
*> TRANSA is CHARACTER*1
*> On entry, TRANSA specifies the form of op( A ) to be used in
*> the matrix multiplication as follows:
*>
*> TRANSA = 'N' or 'n', op( A ) = A.
*>
*> TRANSA = 'T' or 't', op( A ) = A**T.
*>
*> TRANSA = 'C' or 'c', op( A ) = A**T.
*> \endverbatim
*>
*> \param[in] TRANSB
*> \verbatim
*> TRANSB is CHARACTER*1
*> On entry, TRANSB specifies the form of op( B ) to be used in
*> the matrix multiplication as follows:
*>
*> TRANSB = 'N' or 'n', op( B ) = B.
*>
*> TRANSB = 'T' or 't', op( B ) = B**T.
*>
*> TRANSB = 'C' or 'c', op( B ) = B**T.
*> \endverbatim
*>
*> \param[in] M
*> \verbatim
*> M is INTEGER
*> On entry, M specifies the number of rows of the matrix
*> op( A ) and of the matrix C. M must be at least zero.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> On entry, N specifies the number of columns of the matrix
*> op( B ) and the number of columns of the matrix C. N must be
*> at least zero.
*> \endverbatim
*>
*> \param[in] K
*> \verbatim
*> K is INTEGER
*> On entry, K specifies the number of columns of the matrix
*> op( A ) and the number of rows of the matrix op( B ). K must
*> be at least zero.
*> \endverbatim
*>
*> \param[in] ALPHA
*> \verbatim
*> ALPHA is REAL
*> On entry, ALPHA specifies the scalar alpha.
*> \endverbatim
*>
*> \param[in] A
*> \verbatim
*> A is REAL array of DIMENSION ( LDA, ka ), where ka is
*> k when TRANSA = 'N' or 'n', and is m otherwise.
*> Before entry with TRANSA = 'N' or 'n', the leading m by k
*> part of the array A must contain the matrix A, otherwise
*> the leading k by m part of the array A must contain the
*> matrix A.
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> On entry, LDA specifies the first dimension of A as declared
*> in the calling (sub) program. When TRANSA = 'N' or 'n' then
*> LDA must be at least max( 1, m ), otherwise LDA must be at
*> least max( 1, k ).
*> \endverbatim
*>
*> \param[in] B
*> \verbatim
*> B is REAL array of DIMENSION ( LDB, kb ), where kb is
*> n when TRANSB = 'N' or 'n', and is k otherwise.
*> Before entry with TRANSB = 'N' or 'n', the leading k by n
*> part of the array B must contain the matrix B, otherwise
*> the leading n by k part of the array B must contain the
*> matrix B.
*> \endverbatim
*>
*> \param[in] LDB
*> \verbatim
*> LDB is INTEGER
*> On entry, LDB specifies the first dimension of B as declared
*> in the calling (sub) program. When TRANSB = 'N' or 'n' then
*> LDB must be at least max( 1, k ), otherwise LDB must be at
*> least max( 1, n ).
*> \endverbatim
*>
*> \param[in] BETA
*> \verbatim
*> BETA is REAL
*> On entry, BETA specifies the scalar beta. When BETA is
*> supplied as zero then C need not be set on input.
*> \endverbatim
*>
*> \param[in,out] C
*> \verbatim
*> C is REAL array of DIMENSION ( LDC, n ).
*> Before entry, the leading m by n part of the array C must
*> contain the matrix C, except when beta is zero, in which
*> case C need not be set on entry.
*> On exit, the array C is overwritten by the m by n matrix
*> ( alpha*op( A )*op( B ) + beta*C ).
*> \endverbatim
*>
*> \param[in] LDC
*> \verbatim
*> LDC is INTEGER
*> On entry, LDC specifies the first dimension of C as declared
*> in the calling (sub) program. LDC must be at least
*> max( 1, m ).
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup single_blas_level3
*
*> \par Further Details:
* =====================
*>
*> \verbatim
*>
*> Wrapper for ASD2 ILIM aware level 3 Blas routine.
```

```
*>
*> \endverbatim
*>
* =====================================================================
      SUBROUTINE SGEMM_ILIM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,
     $ C,LDC,MILIM,MIS,NILIM,NIS)
*
* -- LAPACK computational routine (version 3.6.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2014
*
* .. Scalar Arguments ..
      REAL ALPHA,BETA
      INTEGER K,LDA,LDB,LDC,M,N,MIS,NIS
      CHARACTER TRANSA,TRANSB
* ..
* .. Array Arguments ..
      INTEGER MILIM(*),NILIM(*)
      REAL A(LDA,*),B(LDB,*),C(LDC,*)
* ..
*
* =====================================================================
*
* ..
* .. External Subroutines ..
      EXTERNAL SGEMM
* ..
* .. Local Scalars ..
      INTEGER I,J,MLOW,MHIGH,NLOW,NHIGH
* ..
*
* Needs fix for error messages.
      IF ((LSAME(TRANSA,'N')).AND.(LSAME(TRANSB,'N'))) THEN
*
* For each subblock call SGEMM
*
      DO 10 I=1,MIS
        MLOW = MILIM(2*(I-1)+1)
        MHIGH = MILIM(2*(I-1)+2)
        DO 20 J=1,NIS
          NLOW = NILIM(2*(J-1)+1)
          NHIGH = NILIM(2*(J-1)+2)
          CALL SGEMM (TRANSA,TRANSB,MHIGH-MLOW+1,NHIGH-NLOW+1,K,ALPHA,
     $ A(MLOW,1),LDA,B(1,NLOW),LDB,BETA,C(MLOW,NLOW),LDC)
   20 CONTINUE
   10 CONTINUE

      ELSE IF ((LSAME(TRANSA,'T')).AND.(LSAME(TRANSB,'N'))) THEN
      DO 30 I=1,MIS
        MLOW = MILIM(2*(I-1)+1)
        MHIGH = MILIM(2*(I-1)+2)
        DO 40 J=1,NIS
          NLOW = NILIM(2*(J-1)+1)
          NHIGH = NILIM(2*(J-1)+2)
          CALL SGEMM (TRANSA,TRANSB,M,NHIGH-NLOW+1,MHIGH-MLOW+1,ALPHA,
     $ A(MLOW,1),LDA,B(MLOW,NLOW),LDB,BETA,C(1,NLOW),LDC)
   40 CONTINUE
   30 CONTINUE

      ELSE IF ((LSAME(TRANSA,'N')).AND.(LSAME(TRANSB,'T'))) THEN
      DO 50 I=1,MIS
        MLOW = MILIM(2*(I-1)+1)
        MHIGH = MILIM(2*(I-1)+2)
        DO 60 J=1,NIS
          NLOW = NILIM(2*(J-1)+1)
          NHIGH = NILIM(2*(J-1)+2)
          CALL SGEMM (TRANSA,TRANSB,MHIGH-MLOW+1,N,NHIGH-NLOW+1,ALPHA,
     $ A(MLOW,NLOW),LDA,B(1,NLOW),LDB,BETA,C(MLOW,1),LDC)
   60 CONTINUE
   50 CONTINUE
      END IF

      RETURN
*
* End of SGEMM_ILIM .
*
      END
```

A.2.8 ssyrk_ilim.f

```
*> \brief \b SSYRK_ILIM
*
* =========== DOCUMENTATION ===========
*
* Online html documentation available at
* http://www.netlib.org/lapack/explore-html/
*
* Definition:
* ===========
*
* SUBROUTINE SSYRK_ILIM(UPLO,TRANS,N,K,ALPHA,A,LDA,BETA,C,LDC,ILIM,IS)
*
* .. Scalar Arguments ..
* REAL ALPHA,BETA
* INTEGER K,LDA,LDC,N,IS
* CHARACTER TRANS,UPLO
* ..
* .. Array Arguments ..
* INTEGER ILIM(*)
* REAL A(LDA,*),C(LDC,*)
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> SSYRK performs one of the symmetric rank k operations
*>
*> C := alpha*A*A**T + beta*C,
*>
*> or
*>
*> C := alpha*A**T*A + beta*C,
*>
*> where alpha and beta are scalars, C is an n by n symmetric matrix
*> and A is an n by k matrix in the first case and a k by n matrix
*> in the second case.
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] UPLO
*> \verbatim
*> UPLO is CHARACTER*1
*> On entry, UPLO specifies whether the upper or lower
*> triangular part of the array C is to be referenced as
*> follows:
*>
```

∗> UPLO = 'U' **or** 'u' **Only** the upper triangular part of C
∗> is **to** be referenced.
∗>
∗> UPLO = 'L' **or** 'l' **Only** the lower triangular part of C
∗> is **to** be referenced.
∗> \endverbatim
∗>
∗> \param[**in**] TRANS
∗> \verbatim
∗> TRANS is **CHARACTER**∗1
∗> On **entry**, TRANS specifies the operation **to** be performed as
∗> follows:
∗>
∗> TRANS = 'N' **or** 'n' C := alpha∗A∗A∗∗T + beta∗C.
∗>
∗> TRANS = 'T' **or** 't' C := alpha∗A∗∗T∗A + beta∗C.
∗>
∗> TRANS = 'C' **or** 'c' C := alpha∗A∗∗T∗A + beta∗C.
∗> \endverbatim
∗>
∗> \param[**in**] N
∗> \verbatim
∗> N is **INTEGER**
∗> On **entry**, N specifies the order of the matrix C. N must be
∗> at least zero.
∗> \endverbatim
∗>
∗> \param[**in**] K
∗> \verbatim
∗> K is **INTEGER**
∗> On **entry** with TRANS = 'N' **or** 'n', K specifies the **number**
∗> of columns of the matrix A, **and** on **entry** with
∗> TRANS = 'T' **or** 't' **or** 'C' **or** 'c', K specifies the **number**
∗> of rows of the matrix A. K must be at least zero.
∗> \endverbatim
∗>
∗> \param[**in**] ALPHA
∗> \verbatim
∗> ALPHA is **REAL**
∗> On **entry**, ALPHA specifies the scalar alpha.
∗> \endverbatim
∗>
∗> \param[**in**] A
∗> \verbatim
∗> A is **REAL** array of **DIMENSION** (LDA, ka), **where** ka is
∗> k when TRANS = 'N' **or** 'n', **and** is n otherwise.
∗> Before **entry** with TRANS = 'N' **or** 'n', the leading n by k
∗> part of the array A must contain the matrix A, otherwise
∗> the leading k by n part of the array A must contain the
∗> matrix A.
∗> \endverbatim
∗>
∗> \param[**in**] LDA
∗> \verbatim
∗> LDA is **INTEGER**
∗> On **entry**, LDA specifies the first **dimension** of A as declared
∗> **in** the calling (sub) **program**. When TRANS = 'N' **or** 'n'
∗> **then** LDA must be at least **max**(1, n), otherwise LDA must
∗> be at least **max**(1, k).
∗> \endverbatim
∗>
∗> \param[**in**] BETA
∗> \verbatim
∗> BETA is **REAL**
∗> On **entry**, BETA specifies the scalar beta.
∗> \endverbatim
∗>
∗> \param[**in**,**out**] C

```
*> \verbatim
*> C is REAL array of DIMENSION ( LDC, n ).
*> Before entry with UPLO = 'U' or 'u', the leading n by n
*> upper triangular part of the array C must contain the upper
*> triangular part of the symmetric matrix and the strictly
*> lower triangular part of C is not referenced. On exit, the
*> upper triangular part of the array C is overwritten by the
*> upper triangular part of the updated matrix.
*> Before entry with UPLO = 'L' or 'l', the leading n by n
*> lower triangular part of the array C must contain the lower
*> triangular part of the symmetric matrix and the strictly
*> upper triangular part of C is not referenced. On exit, the
*> lower triangular part of the array C is overwritten by the
*> lower triangular part of the updated matrix.
*> \endverbatim
*>
*> \param[in] LDC
*> \verbatim
*> LDC is INTEGER
*> On entry, LDC specifies the first dimension of C as declared
*> in the calling (sub) program. LDC must be at least
*> max( 1, n ).
*> \endverbatim
*>
*> \param[in] ILIM
*> \verbatim
*> ILIM is INTEGER array, dimension (2*M or 2*N)
*> The array of non-zero start/end row positions if uplo='u'
*> The array of non-zero start/end column positions if uplo='l'
*> \endverbatim
*>
*> \param[in] IS
*> \verbatim
*> IS is INTEGER
*> The number of row blocks if uplo='u'
*> The number of column blocks if uplo='l'
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2011
*
*> \ingroup single_blas_level3
*
*> \par Further Details:
* =====================
*>
*> \verbatim
*>
*> Wrapped for ASD2 ILIM Level 3 Blas routine.
*>
*> -- Written on 8-February-1989.
*> Jack Dongarra, Argonne National Laboratory.
*> Iain Duff, AERE Harwell.
*> Jeremy Du Croz, Numerical Algorithms Group Ltd.
*> Sven Hammarling, Numerical Algorithms Group Ltd.
*> \endverbatim
*>
* =====================================================================
      SUBROUTINE SSYRK_ILIM(UPLO,TRANS,N,K,ALPHA,A,LDA,BETA,C,LDC,ILIM,
     $ IS)
*
```

```
* -- Reference BLAS level3 routine (version 3.4.0) --
* -- Reference BLAS is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2011
*
* .. Scalar Arguments ..
      REAL ALPHA,BETA
      INTEGER K,LDA,LDC,N,IS
      CHARACTER TRANS,UPLO
* ..
* .. Array Arguments ..
      INTEGER ILIM(*)
      REAL A(LDA,*),C(LDC,*)
* ..
*
* =====================================================================
*
* .. External Functions ..
      LOGICAL LSAME
      EXTERNAL LSAME
* ..
* .. External Subroutines ..
      EXTERNAL SSYRK
* ..
* .. Local Scalars ..
      INTEGER I,KLOW,KHIGH
* ..

      IF (LSAME(UPLO,'U')) THEN
        DO 10 I=1,IS
          KLOW=ILIM(2*(I-1)+1)
          KHIGH=ILIM(2*(I-1)+2)
          CALL SSYRK(UPLO,TRANS,N,KHIGH-KLOW+1,ALPHA,A(KLOW,1),LDA,BETA
     $ ,C,LDC)
   10 CONTINUE
      ELSE
        DO 20 I=1,IS
          KLOW=ILIM(2*(I-1)+1)
          KHIGH=ILIM(2*(I-1)+2)
          CALL SSYRK(UPLO,TRANS,KHIGH-KLOW+1,K,ALPHA,A(1,KLOW),LDA,BETA
     $ ,C,LDC)
   20 CONTINUE
      ENDIF
*
* End of SSYRK_ILIM.
*
      END
```

A.3 LU & Cholesky calls

A.3.1 asd1_sgetrf.f

```
*> \brief \b ASD1_SGETRF
*
* =========== DOCUMENTATION ===========
*
* TO DO - update dependencies links/archives and documentation to include zero checks
*
* Online html documentation available at
* http://www.netlib.org/lapack/explore-html/
*
*> \htmlonly
*> Download SGETRF + dependencies
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.tgz?format=tgz&filename=/lapack/lapack_routine/sgetrf.f">
*> [TGZ]</a>
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.zip?format=zip&filename=/lapack/lapack_routine/sgetrf.f">
*> [ZIP]</a>
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.txt?format=txt&filename=/lapack/lapack_routine/sgetrf.f">
*> [TXT]</a>
*> \endhtmlonly
*
* Definition:
* ===========
*
* SUBROUTINE ASD1_SGETRF( M, N, A, LDA, IPIV, INFO )
*
* .. Scalar Arguments ..
* INTEGER INFO, LDA, M, N
* ..
* .. Array Arguments ..
* INTEGER IPIV( * )
* REAL A( LDA, * )
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> ASD1_SGETRF computes an LU factorization of a general M-by-N matrix A
*> using partial pivoting with row interchanges and taking advantage of sparsity.
*>
*> The factorization has the form
*> A = P * L * U
*> where P is a permutation matrix, L is lower triangular with unit
*> diagonal elements (lower trapezoidal if m > n), and U is upper
*> triangular (upper trapezoidal if m < n).
*>
*> This is the right-looking Level 3 BLAS version of the algorithm.
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] M
*> \verbatim
*> M is INTEGER
*> The number of rows of the matrix A. M >= 0.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The number of columns of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in,out] A
*> \verbatim
*> A is REAL array, dimension (LDA,N)
*> On entry, the M-by-N matrix to be factored.
*> On exit, the factors L and U from the factorization
*> A = P*L*U; the unit diagonal elements of L are not stored.
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> The leading dimension of the array A. LDA >= max(1,M).
*> \endverbatim
```

```
*>
*> \param[out] IPIV
*> \verbatim
*> IPIV is INTEGER array, dimension (min(M,N))
*> The pivot indices; for 1 <= i <= min(M,N), row i of the
*> matrix was interchanged with row IPIV(i).
*> \endverbatim
*>
*> \param[out] INFO
*> \verbatim
*> INFO is INTEGER
*> = 0: successful exit
*> < 0: if INFO = −i, the i−th argument had an illegal value
*> > 0: if INFO = i, U(i,i) is exactly zero. The factorization
*> has been completed, but the factor U is exactly
*> singular, and division by zero will occur if it is used
*> to solve a system of equations.
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup realGEcomputational
*
* =====================================================================
      SUBROUTINE ASD1_SGETRF( M, N, A, LDA, IPIV, INFO )
*
* −− LAPACK computational routine (version TBD) −−
* −− LAPACK is a software package provided by Univ. of Tennessee, −−
* −− Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..−−
* November 2014
*
* .. Scalar Arguments ..
      INTEGER INFO, LDA, M, N
* ..
* .. Array Arguments ..
      INTEGER IPIV( * )
      REAL A( LDA, * )
* ..
*
* =====================================================================
*
* .. Parameters ..
      REAL ONE
      PARAMETER ( ONE = 1.0E+0 )
* ..
* .. Local Scalars ..
      INTEGER I, IINFO, J, JB, NB, MLOW, NLOW, MHIGH, NHIGH
* ..
* .. External Subroutines ..
      EXTERNAL SGEMM, SGETF2, SLASWP, STRSM, XERBLA, SBZCHK
* ..
* .. External Functions ..
      INTEGER ILAENV
      EXTERNAL ILAENV
* ..
* .. Intrinsic Functions ..
      INTRINSIC MAX, MIN
* ..
* .. Executable Statements ..
*
```

```
* Test the input parameters.
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = −1
      ELSE IF( N.LT.0 ) THEN
         INFO = −2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = −4
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'ASD1_SGETRF', −INFO )
         RETURN
      END IF
*
* Quick return if possible
*
      IF( M.EQ.0 .OR. N.EQ.0 )
     $ RETURN
*
* Determine the block size for this environment.
*
      NB = ILAENV( 1, 'SGETRF', ' ', M, N, −1, −1 )
      IF( NB.LE.1 .OR. NB.GE.MIN( M, N ) ) THEN
*
* Use unblocked code.
*
         CALL SGETF2( M, N, A, LDA, IPIV, INFO )
      ELSE
*
* Use blocked code.
*
         DO 20 J = 1, MIN( M, N ), NB
            JB = MIN( MIN( M, N )−J+1, NB )
*
* Calculate the number of zero rows and avoid doing the
* work for the zero rows (using only MHIGH)
*
            CALL SBZCHK( M−J+1, JB, A(J ,J), LDA, 'row', MLOW, MHIGH)
*
* Factor diagonal and subdiagonal blocks and test for exact
* singularity.
*
            CALL SGETF2( MAX(MHIGH,JB), JB, A( J, J ), LDA, IPIV( J ),
     $ IINFO )
*
* Adjust INFO and the pivot indices.
*
            IF( INFO.EQ.0 .AND. IINFO.GT.0 )
     $ INFO = IINFO + J − 1
            DO 10 I = J, MIN( M, J+JB−1 )
               IPIV( I ) = J − 1 + IPIV( I )
   10 CONTINUE
*
* Apply interchanges to columns 1:J−1.
*
            CALL SLASWP( J−1, A, LDA, J, J+JB−1, IPIV, 1 )
*
            IF( J+JB.LE.N ) THEN
*
* Apply interchanges to columns J+JB:N.
*
               CALL SLASWP( N−J−JB+1, A( 1, J+JB ), LDA, J, J+JB−1,
     $ IPIV, 1 )
*
* Calculate the number of zero columns and avoid doing
* the work for the zero columns (using only NHIGH)
*
```

```
               CALL SBZCHK( JB, N−J−JB+1, A(J ,J+JB) , LDA, 'col', NLOW,
     $ NHIGH)
*
* Compute block row of U.
*
               CALL STRSM( 'Left', 'Lower', 'No transpose', 'Unit', JB,
     $ NHIGH, ONE, A( J, J ), LDA, A( J, J+JB), LDA)
               IF( JB.LT.MHIGH ) THEN
*
* Update trailing submatrix. Use MHIGH,NHIGH to avoid zeros
*
                  CALL SGEMM( 'No transpose', 'No transpose', MHIGH−JB,
     $ NHIGH, JB, −ONE, A( J+JB, J ), LDA,
     $ A( J, J+JB ), LDA, ONE, A( J+JB, J+JB ),
     $ LDA )
               END IF
            END IF
   20 CONTINUE
      END IF
      RETURN
*
* End of ASD1_SGETRF
*
      END
```

A.3.2 asd1_spotrf.f

```
*> \brief \b ASD1_SPOTRF
*
* =========== DOCUMENTATION ===========
*
* Online html documentation available at
* http://www.netlib.org/lapack/explore−html/
*
*> \htmlonly
*> Download SPOTRF + dependencies
*> <a href="http://www.netlib.org/cgi−bin/netlibfiles.tgz?format=tgz&filename=/lapack/lapack_routine/spotrf.f">
*> [TGZ]</a>
*> <a href="http://www.netlib.org/cgi−bin/netlibfiles.zip?format=zip&filename=/lapack/lapack_routine/spotrf.f">
*> [ZIP]</a>
*> <a href="http://www.netlib.org/cgi−bin/netlibfiles.txt?format=txt&filename=/lapack/lapack_routine/spotrf.f">
*> [TXT]</a>
*> \endhtmlonly
*
* Definition:
* ===========
*
* SUBROUTINE ASD1_SPOTRF( UPLO, N, A, LDA, INFO )
*
* .. Scalar Arguments ..
* CHARACTER UPLO
* INTEGER INFO, LDA, N
* ..
* .. Array Arguments ..
* REAL A( LDA, * )
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> ASD1_SPOTRF computes the Cholesky factorization of a real symmetric
*> positive definite matrix A taking into account sparsity.
*>
*> The factorization has the form
*> A = U**T * U, if UPLO = 'U', or
*> A = L * L**T, if UPLO = 'L',
*> where U is an upper triangular matrix and L is lower triangular.
*>
*> This is the block version of the algorithm, calling Level 3 BLAS.
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] UPLO
*> \verbatim
*> UPLO is CHARACTER*1
*> = 'U': Upper triangle of A is stored;
*> = 'L': Lower triangle of A is stored.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The order of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in,out] A
*> \verbatim
*> A is REAL array, dimension (LDA,N)
*> On entry, the symmetric matrix A. If UPLO = 'U', the leading
*> N-by-N upper triangular part of A contains the upper
*> triangular part of the matrix A, and the strictly lower
*> triangular part of A is not referenced. If UPLO = 'L', the
*> leading N-by-N lower triangular part of A contains the lower
*> triangular part of the matrix A, and the strictly upper
*> triangular part of A is not referenced.
*>
*> On exit, if INFO = 0, the factor U or L from the Cholesky
*> factorization A = U**T*U or A = L*L**T.
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> The leading dimension of the array A. LDA >= max(1,N).
*> \endverbatim
*>
*> \param[out] INFO
*> \verbatim
*> INFO is INTEGER
*> = 0: successful exit
*> < 0: if INFO = -i, the i-th argument had an illegal value
*> > 0: if INFO = i, the leading minor of order i is not
*> positive definite, and the factorization could not be
*> completed.
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup realPOcomputational
```

```
*
* ==========================================================================
      SUBROUTINE ASD1_SPOTRF( UPLO, N, A, LDA, INFO )
*
* -- LAPACK computational routine (version TBD) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2014
*
* .. Scalar Arguments ..
      CHARACTER UPLO
      INTEGER INFO, LDA, N
* ..
* .. Array Arguments ..
      REAL A( LDA, * )
* ..
*
* ==========================================================================
*
* .. Parameters ..
      REAL ONE
      PARAMETER ( ONE = 1.0E+0 )
* ..
* .. Local Scalars ..
      LOGICAL UPPER
      INTEGER J, JB, NB, NLOW, NHIGH, MLOW, MHIGH
* ..
* .. External Functions ..
      LOGICAL LSAME
      INTEGER ILAENV
      EXTERNAL LSAME, ILAENV
* ..
* .. External Subroutines ..
      EXTERNAL SGEMM, SPOTF2, SSYRK, STRSM, XERBLA, SBZCHK
* ..
* .. Intrinsic Functions ..
      INTRINSIC MAX, MIN
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
      INFO = 0
      UPPER = LSAME( UPLO, 'U' )
      IF( .NOT.UPPER .AND. .NOT.LSAME( UPLO, 'L' ) ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
         INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'SPOTRF', -INFO )
         RETURN
      END IF
*
* Quick return if possible
*
      IF( N.EQ.0 )
     $ RETURN
*
* Determine the block size for this environment.
*
      NB = ILAENV( 1, 'SPOTRF', UPLO, N, -1, -1, -1 )
      IF( NB.LE.1 .OR. NB.GE.N ) THEN
*
* Use unblocked code.
*
```

```
         CALL SPOTF2( UPLO, N, A, LDA, INFO )
      ELSE
*
*  Use blocked code.
*
         IF( UPPER ) THEN
*
*  Compute the Cholesky factorization A = U**T*U.
*
            DO 10 J = 1, N, NB

               JB = MIN( NB, N-J+1 )
*
*  Calculate the number of zero rows and avoid doing the
*  work for the zero rows (using only MLOW)
*
               CALL SBZCHK( J-1, JB, A(1 ,J) , LDA, 'row', MLOW,
     $   MHIGH)
*
*  Update the current diagonal block if MLOW > 0
*
               IF (MLOW.GT.0) THEN
                  CALL SSYRK( 'Upper', 'Transpose', JB, J-MLOW, -ONE,
     $ A( MLOW, J ), LDA, ONE, A( J, J ), LDA )
               END IF
*
*  Factorize the current diagonal block and test
*  for non-positive-definiteness.
*
               CALL SPOTF2( 'Upper', JB, A( J, J ), LDA, INFO )
               IF( INFO.NE.0 )
     $ GO TO 30
*
*  Calculate the number of zero cols and avoid doing the
*  work for the zero cols (using only NHIGH)
*
               CALL SBZCHK( J-MLOW, N-J+1-JB, A(MLOW ,J+JB) , LDA, 'col',
     $   NLOW, NHIGH)
               IF( NHIGH.GT.0 ) THEN
*
*  Compute the current block row.
*
                                      IF (MLOW.GT.0) THEN
                     CALL SGEMM( 'Transpose', 'No transpose', JB,
     $ NHIGH, J-MLOW, -ONE, A( MLOW, J ), LDA,
     $ A(MLOW, J+JB ), LDA, ONE, A( J, J+JB ),
     $  LDA)
                  END IF
               END IF
*
*  Calculate the number of zero cols and avoid doing the
*  work for the zero cols (using only NHIGH)
*
               CALL SBZCHK( JB, N-J+1-JB, A(J ,J+JB) , LDA, 'col',
     $   NLOW, NHIGH)
               IF( NHIGH.GT.0 ) THEN
*
*  Compute the current block row.
*
                  CALL STRSM( 'Left', 'Upper', 'Transpose', 'Non-unit',
     $ JB, NHIGH, ONE, A( J, J ), LDA,
     $ A( J, J+JB ), LDA )
               END IF
   10       CONTINUE
*
         ELSE
*
*  Compute the Cholesky factorization A = L*L**T.
```

```
*
            DO 20 J = 1, N, NB
               JB = MIN( NB, N-J+1 )
*
* Calculate the number of zero cols and avoid doing the
* work for the zero cols (using only NLOW)
*
               CALL SBZCHK( JB, J-1, A(J ,1) , LDA, 'col', NLOW ,
     $  NHIGH)
*
* Update and factorize the current diagonal block and test
* for non-positive-definiteness.
*
               IF(NLOW.GT.0) THEN
                 CALL SSYRK( 'Lower', 'No transpose', JB, J-NLOW, -ONE,
     $ A( J, NLOW ), LDA, ONE, A( J, J ), LDA )
               END IF
               CALL SPOTF2( 'Lower', JB, A( J, J ), LDA, INFO )
               IF( INFO.NE.0 )
     $ GO TO 30
*
* Calculate the number of zero rows and avoid doing the
* work for the zero rows (using only MHIGH)
*
               CALL SBZCHK( N-J+1-JB, J-MLOW, A(J+JB ,MLOW) , LDA, 'row',
     $   MLOW, MHIGH)
               IF( MHIGH.GT.0 ) THEN
*
* Compute the current block column.
*
                  IF (NLOW.GT.0) THEN
                    CALL SGEMM( 'No transpose', 'Transpose', NHIGH, JB,
     $ J-NLOW, -ONE, A( J+JB, NLOW), LDA,
     $ A( J, NLOW), LDA, ONE, A( J+JB, J ),
     $  LDA )
                  END IF
               END IF
*
* Calculate the number of zero rows and avoid doing the
* work for the zero rows (using only MHIGH)
*
               CALL SBZCHK( N-J+1-JB, JB, A(J+JB ,J) , LDA, 'row', MLOW,
     $   MHIGH)
               IF( MHIGH.GT.0 ) THEN
                  CALL STRSM( 'Right', 'Lower', 'Transpose', 'Non-unit',
     $ NHIGH, JB, ONE, A( J, J ), LDA,
     $ A( J+JB, J ), LDA )
               END IF
   20 CONTINUE
         END IF
      END IF
      GO TO 40
*
   30 CONTINUE
      INFO = INFO + J - 1
*
   40 CONTINUE
      RETURN
*
* End of SPOTRF
*
      END
```

A.3.3 asd1_sgbtrf.f

```
*> \brief \b ASD1_SGBTRF
*
* =========== DOCUMENTATION ===========
*
* Online html documentation available at
* http://www.netlib.org/lapack/explore-html/
*
*> \htmlonly
*> Download SGBTRF + dependencies
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.tgz?format=tgz&filename=/lapack/lapack_routine/sgbtrf.f">
*> [TGZ]</a>
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.zip?format=zip&filename=/lapack/lapack_routine/sgbtrf.f">
*> [ZIP]</a>
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.txt?format=txt&filename=/lapack/lapack_routine/sgbtrf.f">
*> [TXT]</a>
*> \endhtmlonly
*
* Definition:
* ===========
*
* SUBROUTINE ASD1_SGBTRF( M, N, KL, KU, AB, LDAB, IPIV, INFO )
*
* .. Scalar Arguments ..
* INTEGER INFO, KL, KU, LDAB, M, N
* ..
* .. Array Arguments ..
* INTEGER IPIV( * )
* REAL AB( LDAB, * )
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> SGBTRF computes an LU factorization of a real m-by-n band matrix A
*> using partial pivoting with row interchanges taking advantage of sparsity.
*>
*> This is the blocked version of the algorithm, calling Level 3 BLAS.
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] M
*> \verbatim
*> M is INTEGER
*> The number of rows of the matrix A. M >= 0.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The number of columns of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in] KL
*> \verbatim
*> KL is INTEGER
*> The number of subdiagonals within the band of A. KL >= 0.
*> \endverbatim
*>
*> \param[in] KU
*> \verbatim
*> KU is INTEGER
*> The number of superdiagonals within the band of A. KU >= 0.
*> \endverbatim
*>
*> \param[in,out] AB
*> \verbatim
*> AB is REAL array, dimension (LDAB,N)
*> On entry, the matrix A in band storage, in rows KL+1 to
*> 2*KL+KU+1; rows 1 to KL of the array need not be set.
*> The j−th column of A is stored in the j−th column of the
*> array AB as follows:
*> AB(kl+ku+1+i−j,j) = A(i,j) for max(1,j−ku)<=i<=min(m,j+kl)
*>
*> On exit, details of the factorization: U is stored as an
*> upper triangular band matrix with KL+KU superdiagonals in
*> rows 1 to KL+KU+1, and the multipliers used during the
*> factorization are stored in rows KL+KU+2 to 2*KL+KU+1.
*> See below for further details.
*> \endverbatim
*>
*> \param[in] LDAB
*> \verbatim
*> LDAB is INTEGER
*> The leading dimension of the array AB. LDAB >= 2*KL+KU+1.
*> \endverbatim
*>
*> \param[out] IPIV
*> \verbatim
*> IPIV is INTEGER array, dimension (min(M,N))
*> The pivot indices; for 1 <= i <= min(M,N), row i of the
*> matrix was interchanged with row IPIV(i).
*> \endverbatim
*>
*> \param[out] INFO
*> \verbatim
*> INFO is INTEGER
*> = 0: successful exit
*> < 0: if INFO = −i, the i−th argument had an illegal value
*> > 0: if INFO = +i, U(i,i) is exactly zero. The factorization
*> has been completed, but the factor U is exactly
*> singular, and division by zero will occur if it is used
*> to solve a system of equations.
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup realGBcomputational
*
*> \par Further Details:
* =====================
*>
*> \verbatim
*>
*> The band storage scheme is illustrated by the following example, when
*> M = N = 6, KL = 2, KU = 1:
*>
*> On entry: On exit:
*>
*> * * * + + + * * * u14 u25 u36
*> * * + + + + * * u13 u24 u35 u46
*> * a12 a23 a34 a45 a56 * u12 u23 u34 u45 u56
*> a11 a22 a33 a44 a55 a66 u11 u22 u33 u44 u55 u66
*> a21 a32 a43 a54 a65 * m21 m32 m43 m54 m65 *
*> a31 a42 a53 a64 * * m31 m42 m53 m64 * *
*>
*> Array elements marked * are not used by the routine; elements marked
*> + need not be set on entry, but are required by the routine to store
*> elements of U because of fill−in resulting from the row interchanges.
*> \endverbatim
*>
* =====================================================================
      SUBROUTINE ASD1_SGBTRF( M, N, KL, KU, AB, LDAB, IPIV, INFO )
*
* −− LAPACK computational routine (version TBD) −−
* −− LAPACK is a software package provided by Univ. of Tennessee, −−
* −− Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..−−
* November 2014
*
* .. Scalar Arguments ..
      INTEGER INFO, KL, KU, LDAB, M, N
* ..
* .. Array Arguments ..
      INTEGER IPIV( * )
      REAL AB( LDAB, * )
* ..
*
* =====================================================================
*
* .. Parameters ..
      REAL ONE, ZERO
      PARAMETER ( ONE = 1.0E+0, ZERO = 0.0E+0 )
      INTEGER NBMAX, LDWORK
      PARAMETER ( NBMAX = 64, LDWORK = NBMAX+1 )
* ..
* .. Local Scalars ..
      INTEGER I, I2, I3, II, IP, J, J2, J3, JB, JJ, JM, JP,
     $ JU, K2, KM, KV, NB, NW, ILOW2, IHIGH2, JLOW2,
     $  JHIGH2
      REAL TEMP
* ..
* .. Local Arrays ..
      REAL WORK13( LDWORK, NBMAX ),
     $ WORK31( LDWORK, NBMAX )
* ..
* .. External Functions ..
      INTEGER ILAENV, ISAMAX
      EXTERNAL ILAENV, ISAMAX
* ..
* .. External Subroutines ..
      EXTERNAL SCOPY, SGBTF2, SGEMM, SGER, SLASWP, SSCAL,
     $ SSWAP, STRSM, XERBLA, SBZCHK
* ..
* .. Intrinsic Functions ..
      INTRINSIC MAX, MIN
* ..
* .. Executable Statements ..
*
* KV is the number of superdiagonals in the factor U, allowing for
* fill−in
*
      KV = KU + KL
*
* Test the input parameters.
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = −1
      ELSE IF( N.LT.0 ) THEN
         INFO = −2
      ELSE IF( KL.LT.0 ) THEN
```

```
         INFO = −3
      ELSE IF( KU.LT.0 ) THEN
         INFO = −4
      ELSE IF( LDAB.LT.KL+KV+1 ) THEN
         INFO = −6
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'ASD1_SGBTRF', −INFO )
         RETURN
      END IF
*
* Quick return if possible
*
      IF( M.EQ.0 .OR. N.EQ.0 )
     $ RETURN
*
* Determine the block size for this environment
*
      NB = ILAENV( 1, 'SGBTRF', ' ', M, N, KL, KU )
*
* The block size must not exceed the limit set by the size of the
* local arrays WORK13 and WORK31.
*
      NB = MIN( NB, NBMAX )
*
      IF( NB.LE.1 .OR. NB.GT.KL ) THEN
*
* Use unblocked code
*
         CALL SGBTF2( M, N, KL, KU, AB, LDAB, IPIV, INFO )
      ELSE
*
* Use blocked code
*
* Zero the superdiagonal elements of the work array WORK13
*
         DO 20 J = 1, NB
            DO 10 I = 1, J − 1
               WORK13( I, J ) = ZERO
   10 CONTINUE
   20 CONTINUE
*
* Zero the subdiagonal elements of the work array WORK31
*
         DO 40 J = 1, NB
            DO 30 I = J + 1, NB
               WORK31( I, J ) = ZERO
   30 CONTINUE
   40 CONTINUE
*
* Gaussian elimination with partial pivoting
*
* Set fill−in elements in columns KU+2 to KV to zero
*
         DO 60 J = KU + 2, MIN( KV, N )
            DO 50 I = KV − J + 2, KL
               AB( I, J ) = ZERO
   50 CONTINUE
   60 CONTINUE
*
* JU is the index of the last column affected by the current
* stage of the factorization
*
         JU = 1
*
         DO 180 J = 1, MIN( M, N ), NB
            JB = MIN( NB, MIN( M, N )−J+1 )
*
```

```
* The active part of the matrix is partitioned
*
* A11 A12 A13
* A21 A22 A23
* A31 A32 A33
*
* Here A11, A21 and A31 denote the current block of JB columns
* which is about to be factorized. The number of rows in the
* partitioning are JB, I2, I3 respectively, and the numbers
* of columns are JB, J2, J3. The superdiagonal elements of A13
* and the subdiagonal elements of A31 lie outside the band.
*
            I2 = MIN( KL−JB, M−J−JB+1 )
            I3 = MIN( JB, M−J−KL+1 )
*
* Calculate the number of zero rows and avoid doing the
* work for the zero rows (using only MU)
*
            CALL SBZCHK( I2, JB, AB(KV+1+JB,J) , LDAB−1, 'row', ILOW2,
     $   IHIGH2)
*
* J2 and J3 are computed after JU has been updated.
*
* Factorize the current block of JB columns
*
            DO 80 JJ = J, J + JB − 1
*
* Set fill−in elements in column JJ+KV to zero
*
               IF( JJ+KV.LE.N ) THEN
                  DO 70 I = 1, KL
                     AB( I, JJ+KV ) = ZERO
   70             CONTINUE
               END IF
*
* Find pivot and test for singularity. KM is the number of
* subdiagonal elements in the current column.
*
               KM = MIN( KL, M−JJ )
               JP = ISAMAX( KM+1, AB( KV+1, JJ ), 1 )
               IPIV( JJ ) = JP + JJ − J
               IF( AB( KV+JP, JJ ).NE.ZERO ) THEN
                  JU = MAX( JU, MIN( JJ+KU+JP−1, N ) )
                  IF( JP.NE.1 ) THEN
*
* Apply interchange to columns J to J+JB−1
*
                     IF( JP+JJ−1.LT.J+KL ) THEN
*
                        CALL SSWAP( JB, AB( KV+1+JJ−J, J ), LDAB−1,
     $ AB( KV+JP+JJ−J, J ), LDAB−1 )
                     ELSE
*
* The interchange affects columns J to JJ−1 of A31
* which are stored in the work array WORK31
*
                        CALL SSWAP( JJ−J, AB( KV+1+JJ−J, J ), LDAB−1,
     $ WORK31( JP+JJ−J−KL, 1 ), LDWORK )
                        CALL SSWAP( J+JB−JJ, AB( KV+1, JJ ), LDAB−1,
     $ AB( KV+JP, JJ ), LDAB−1 )
                     END IF
                  END IF
*
* Compute multipliers
*
                  CALL SSCAL( KM, ONE / AB( KV+1, JJ ), AB( KV+2, JJ ),
     $ 1 )
*
```

```
* Update trailing submatrix within the band and within
* the current block. JM is the index of the last column
* which needs to be updated.
*
                  JM = MIN( JU, J+JB−1 )
                  IF( JM.GT.JJ )
     $ CALL SGER( KM, JM−JJ, −ONE, AB( KV+2, JJ ), 1,
     $ AB( KV, JJ+1 ), LDAB−1,
     $ AB( KV+1, JJ+1 ), LDAB−1 )
               ELSE
*
* If pivot is zero, set INFO to the index of the pivot
* unless a zero pivot has already been found.
*
                  IF( INFO.EQ.0 )
     $ INFO = JJ
               END IF
*
* Copy current column of A31 into the work array WORK31
*
               NW = MIN( JJ−J+1, I3 )
               IF( NW.GT.0 )
     $ CALL SCOPY( NW, AB( KV+KL+1−JJ+J, JJ ), 1,
     $ WORK31( 1, JJ−J+1 ), 1 )
   80 CONTINUE
            IF( J+JB.LE.N ) THEN
*
* Apply the row interchanges to the other blocks.
*
               J2 = MIN( JU−J+1, KV ) − JB
               J3 = MAX( 0, JU−J−KV+1 )

*
* Use SLASWP to apply the row interchanges to A12, A22, and
* A32.
*
               CALL SLASWP( J2, AB( KV+1−JB, J+JB ), LDAB−1, 1, JB,
     $ IPIV( J ), 1 )
*
* Adjust the pivot indices.
*
               DO 90 I = J, J + JB − 1
                  IPIV( I ) = IPIV( I ) + J − 1
   90 CONTINUE
*
* Apply the row interchanges to A13, A23, and A33
* columnwise.
*
               K2 = J − 1 + JB + J2
               DO 110 I = 1, J3
                  JJ = K2 + I
                  DO 100 II = J + I − 1, J + JB − 1
                     IP = IPIV( II )
                     IF( IP.NE.II ) THEN
                        TEMP = AB( KV+1+II−JJ, JJ )
                        AB( KV+1+II−JJ, JJ ) = AB( KV+1+IP−JJ, JJ )
                        AB( KV+1+IP−JJ, JJ ) = TEMP
                     END IF
  100 CONTINUE
  110 CONTINUE
*
* Calculate the number of zero rows and avoid doing the
* work for the zero rows (using only JHIGH2)
*
               CALL SBZCHK( JB, J2, AB(KV+1−JB,J+JB) , LDAB−1, 'col',
     $ JLOW2, JHIGH2)
*
* Update the relevant part of the trailing submatrix
```

```
*
                    IF( JHIGH2.GT.0 ) THEN
*
* Update A12
*
                        CALL STRSM( 'Left', 'Lower', 'No transpose', 'Unit',
     $ JB, JHIGH2, ONE, AB( KV+1, J ), LDAB−1,
     $ AB( KV+1−JB, J+JB ), LDAB−1 )
*
                        IF( IHIGH2.GT.0 ) THEN
*
* Update A22
*
                            CALL SGEMM( 'No transpose', 'No transpose', IHIGH2,
     $      JHIGH2, JB, −ONE, AB( KV+1+JB, J ), LDAB−1,
     $ AB( KV+1−JB, J+JB ), LDAB−1, ONE,
     $ AB( KV+1, J+JB ), LDAB−1 )
                        END IF
*
                        IF( I3.GT.0 ) THEN
*
* Update A32
*
                            CALL SGEMM( 'No transpose', 'No transpose', I3,
     $ JHIGH2,JB, −ONE, WORK31, LDWORK,
     $ AB( KV+1−JB, J+JB ), LDAB−1, ONE,
     $ AB( KV+KL+1−JB, J+JB ), LDAB−1 )
                        END IF
                    END IF
*
                    IF( J3.GT.0 ) THEN
*
* Copy the lower triangle of A13 into the work array
* WORK13
*
                        DO 130 JJ = 1, J3
                            DO 120 II = JJ, JB
                                WORK13( II, JJ ) = AB( II−JJ+1, JJ+J+KV−1 )
  120 CONTINUE
  130 CONTINUE
*
* Update A13 in the work array
*
                        CALL STRSM( 'Left', 'Lower', 'No transpose', 'Unit',
     $ JB, J3, ONE, AB( KV+1, J ), LDAB−1,
     $ WORK13, LDWORK )
*
                        IF( IHIGH2.GT.0 ) THEN
*
* Update A23
*
                            CALL SGEMM( 'No transpose', 'No transpose', IHIGH2,
     $ J3, JB, −ONE, AB( KV+1+JB, J ), LDAB−1,
     $ WORK13, LDWORK, ONE, AB( 1+JB, J+KV ),
     $ LDAB−1 )
                        END IF
*
                        IF( I3.GT.0 ) THEN
*
* Update A33
*
                            CALL SGEMM( 'No transpose', 'No transpose', I3, J3,
     $ JB, −ONE, WORK31, LDWORK, WORK13,
     $ LDWORK, ONE, AB( 1+KL, J+KV ), LDAB−1 )
                        END IF
*
* Copy the lower triangle of A13 back into place
*
```

```
                  DO 150 JJ = 1, J3
                     DO 140 II = JJ, JB
                        AB( II−JJ+1, JJ+J+KV−1 ) = WORK13( II, JJ )
  140 CONTINUE
  150 CONTINUE
                END IF
             ELSE
*
* Adjust the pivot indices.
*
                DO 160 I = J, J + JB − 1
                  IPIV( I ) = IPIV( I ) + J − 1
  160 CONTINUE
             END IF
*
* Partially undo the interchanges in the current block to
* restore the upper triangular form of A31 and copy the upper
* triangle of A31 back into place
*
             DO 170 JJ = J + JB − 1, J, −1
                JP = IPIV( JJ ) − JJ + 1
                IF( JP.NE.1 ) THEN
*
* Apply interchange to columns J to JJ−1
*
                  IF( JP+JJ−1.LT.J+KL ) THEN
*
* The interchange does not affect A31
*
                     CALL SSWAP( JJ−J, AB( KV+1+JJ−J, J ), LDAB−1,
     $ AB( KV+JP+JJ−J, J ), LDAB−1 )
                  ELSE
*
* The interchange does affect A31
*
                     CALL SSWAP( JJ−J, AB( KV+1+JJ−J, J ), LDAB−1,
     $ WORK31( JP+JJ−J−KL, 1 ), LDWORK )
                  END IF
                END IF
*
* Copy the current column of A31 back into place
*
                NW = MIN( I3, JJ−J+1 )
                IF( NW.GT.0 )
     $ CALL SCOPY( NW, WORK31( 1, JJ−J+1 ), 1,
     $ AB( KV+KL+1−JJ+J, JJ ), 1 )
  170 CONTINUE
  180 CONTINUE
      END IF
*
      RETURN
*
* End of SGBTRF
*
      END
```

A.3.4 asd1_spbtrf.f

```
*> \brief \b ASD1_SPBTRF
*
* =========== DOCUMENTATION ===========
*
* Online html documentation available at
* http://www.netlib.org/lapack/explore−html/
```

```
*
*> \htmlonly
*> Download SPBTRF + dependencies
*> <a href="http://www.netlib.org/cgi−bin/netlibfiles.tgz?format=tgz&filename=/lapack/lapack_routine/spbtrf.f">
*> [TGZ]</a>
*> <a href="http://www.netlib.org/cgi−bin/netlibfiles.zip?format=zip&filename=/lapack/lapack_routine/spbtrf.f">
*> [ZIP]</a>
*> <a href="http://www.netlib.org/cgi−bin/netlibfiles.txt?format=txt&filename=/lapack/lapack_routine/spbtrf.f">
*> [TXT]</a>
*> \endhtmlonly
*
* Definition:
* ===========
*
* SUBROUTINE ASD1_SPBTRF( UPLO, N, KD, AB, LDAB, INFO )
*
* .. Scalar Arguments ..
* CHARACTER UPLO
* INTEGER INFO, KD, LDAB, N
* ..
* .. Array Arguments ..
* REAL AB( LDAB, * )
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> SPBTRF computes the Cholesky factorization of a real symmetric
*> positive definite band matrix A.
*>
*> The factorization has the form
*> A = U**T * U, if UPLO = 'U', or
*> A = L * L**T, if UPLO = 'L',
*> where U is an upper triangular matrix and L is lower triangular.
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] UPLO
*> \verbatim
*> UPLO is CHARACTER*1
*> = 'U': Upper triangle of A is stored;
*> = 'L': Lower triangle of A is stored.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The order of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in] KD
*> \verbatim
*> KD is INTEGER
*> The number of superdiagonals of the matrix A if UPLO = 'U',
*> or the number of subdiagonals if UPLO = 'L'. KD >= 0.
*> \endverbatim
*>
*> \param[in,out] AB
*> \verbatim
*> AB is REAL array, dimension (LDAB,N)
*> On entry, the upper or lower triangle of the symmetric band
*> matrix A, stored in the first KD+1 rows of the array. The
*> j−th column of A is stored in the j−th column of the array AB
*> as follows:
*> if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j;
*> if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).
*>
*> On exit, if INFO = 0, the triangular factor U or L from the
*> Cholesky factorization A = U**T*U or A = L*L**T of the band
*> matrix A, in the same storage format as A.
*> \endverbatim
*>
*> \param[in] LDAB
*> \verbatim
*> LDAB is INTEGER
*> The leading dimension of the array AB. LDAB >= KD+1.
*> \endverbatim
*>
*> \param[out] INFO
*> \verbatim
*> INFO is INTEGER
*> = 0: successful exit
*> < 0: if INFO = -i, the i-th argument had an illegal value
*> > 0: if INFO = i, the leading minor of order i is not
*> positive definite, and the factorization could not be
*> completed.
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup realOTHERcomputational
*
*> \par Further Details:
* =====================
*>
*> \verbatim
*>
*> The band storage scheme is illustrated by the following example, when
*> N = 6, KD = 2, and UPLO = 'U':
*>
*> On entry: On exit:
*>
*> * * a13 a24 a35 a46 * * u13 u24 u35 u46
*> * a12 a23 a34 a45 a56 * u12 u23 u34 u45 u56
*> a11 a22 a33 a44 a55 a66 u11 u22 u33 u44 u55 u66
*>
*> Similarly, if UPLO = 'L' the format of A is as follows:
*>
*> On entry: On exit:
*>
*> a11 a22 a33 a44 a55 a66 l11 l22 l33 l44 l55 l66
*> a21 a32 a43 a54 a65 * l21 l32 l43 l54 l65 *
*> a31 a42 a53 a64 * * l31 l42 l53 l64 * *
*>
*> Array elements marked * are not used by the routine.
*> \endverbatim
*
*> \par Contributors:
* ==================
*>
*> Peter Mayes and Giuseppe Radicati, IBM ECSEC, Rome, March 23, 1989
*
* =====================================================================
      SUBROUTINE ASD1_SPBTRF( UPLO, N, KD, AB, LDAB, INFO )
*
* -- LAPACK computational routine (version TBD) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2014
*
* .. Scalar Arguments ..
      CHARACTER UPLO
      INTEGER INFO, KD, LDAB, N
* ..
* .. Array Arguments ..
      REAL AB( LDAB, * )
* ..
*
* ==========================================================================
*
* .. Parameters ..
      REAL ONE, ZERO
      PARAMETER ( ONE = 1.0E+0, ZERO = 0.0E+0 )
      INTEGER NBMAX, LDWORK
      PARAMETER ( NBMAX = 32, LDWORK = NBMAX+1 )
* ..
* .. Local Scalars ..
      INTEGER I, I2, I3, IB, II, J, JJ, NB, ILOW2, IHIGH2
* ..
* .. Local Arrays ..
      REAL WORK( LDWORK, NBMAX )
* ..
* .. External Functions ..
      LOGICAL LSAME
      INTEGER ILAENV
      EXTERNAL LSAME, ILAENV
* ..
* .. External Subroutines ..
      EXTERNAL SGEMM, SPBTF2, SPOTF2, SSYRK, STRSM, XERBLA,
     $ SBZCHK
* ..
* .. Intrinsic Functions ..
      INTRINSIC MIN
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
      INFO = 0
      IF( ( .NOT.LSAME( UPLO, 'U' ) ) .AND.
     $ ( .NOT.LSAME( UPLO, 'L' ) ) ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( KD.LT.0 ) THEN
         INFO = -3
      ELSE IF( LDAB.LT.KD+1 ) THEN
         INFO = -5
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'ASD1_SPBTRF', -INFO )
         RETURN
      END IF
*
* Quick return if possible
*
      IF( N.EQ.0 )
     $ RETURN
*
* Determine the block size for this environment
*
```

```
      NB = ILAENV( 1, 'SPBTRF', UPLO, N, KD, −1, −1 )
*
* The block size must not exceed the semi−bandwidth KD, and must not
* exceed the limit set by the size of the local array WORK.
*
      NB = MIN( NB, NBMAX )
*
      IF( NB.LE.1 .OR. NB.GT.KD ) THEN
*
* Use unblocked code
*
         CALL SPBTF2( UPLO, N, KD, AB, LDAB, INFO )
      ELSE
*
* Use blocked code
*
         IF( LSAME( UPLO, 'U' ) ) THEN
*
* Compute the Cholesky factorization of a symmetric band
* matrix, given the upper triangle of the matrix in band
* storage.
*
* Zero the upper triangle of the work array.
*
            DO 20 J = 1, NB
               DO 10 I = 1, J − 1
                  WORK( I, J ) = ZERO
   10 CONTINUE
   20 CONTINUE
*
* Process the band matrix one diagonal block at a time.
*
            DO 70 I = 1, N, NB
               IB = MIN( NB, N−I+1 )
*
* Factorize the diagonal block
*
               CALL SPOTF2( UPLO, IB, AB( KD+1, I ), LDAB−1, II )
               IF( II.NE.0 ) THEN
                  INFO = I + II − 1
                  GO TO 150
               END IF
               IF( I+IB.LE.N ) THEN
*
* Update the relevant part of the trailing submatrix.
* If A11 denotes the diagonal block which has just been
* factorized, then we need to update the remaining
* blocks in the diagram:
*
* A11 A12 A13
* A22 A23
* A33
*
* The numbers of rows and columns in the partitioning
* are IB, I2, I3 respectively. The blocks A12, A22 and
* A23 are empty if IB = KD. The upper triangle of A13
* lies outside the band.
*
                  I2 = MIN( KD−IB, N−I−IB+1 )
                  I3 = MIN( IB, N−I−KD+1 )
*


*
* Calculate the number of zero rows/cols and avoid doing the
* work for the zero rows/cols (using only IHIGH2)
*
                  CALL SBZCHK( IB, I2, AB(KD+1−IB,I+IB), LDAB−1, 'col',
     $    ILOW2, IHIGH2 )
```

```
                    IF( IHIGH2.GT.0 ) THEN
*
* Update A12
*
                       CALL STRSM( 'Left', 'Upper', 'Transpose',
     $ 'Non−unit', IB, IHIGH2, ONE,
     $ AB( KD+1, I ), LDAB−1,
     $ AB( KD+1−IB, I+IB ), LDAB−1 )
*
* Update A22
*
                       CALL SSYRK( 'Upper', 'Transpose', IHIGH2, IB, −ONE,
     $ AB( KD+1−IB, I+IB ), LDAB−1, ONE,
     $ AB( KD+1, I+IB ), LDAB−1 )
                    END IF
*
                    IF( I3.GT.0 ) THEN
*
* Copy the lower triangle of A13 into the work array.
*
                       DO 40 JJ = 1, I3
                          DO 30 II = JJ, IB
                             WORK( II, JJ ) = AB( II−JJ+1, JJ+I+KD−1 )
   30 CONTINUE
   40 CONTINUE
*
* Update A13 (in the work array).
*
                       CALL STRSM( 'Left', 'Upper', 'Transpose',
     $ 'Non−unit', IB, I3, ONE, AB( KD+1, I ),
     $ LDAB−1, WORK, LDWORK )
*
* Update A23
*
                       IF( IHIGH2.GT.0 )
     $ CALL SGEMM( 'Transpose', 'No Transpose', IHIGH2,
     $ I3, IB, −ONE, AB( KD+1−IB, I+IB ),
     $ LDAB−1, WORK, LDWORK, ONE,
     $ AB( 1+IB, I+KD ), LDAB−1 )
*
* Update A33
*
                       CALL SSYRK( 'Upper', 'Transpose', I3, IB, −ONE,
     $ WORK, LDWORK, ONE, AB( KD+1, I+KD ),
     $ LDAB−1 )
*
* Copy the lower triangle of A13 back into place.
*
                       DO 60 JJ = 1, I3
                          DO 50 II = JJ, IB
                             AB( II−JJ+1, JJ+I+KD−1 ) = WORK( II, JJ )
   50 CONTINUE
   60 CONTINUE
                    END IF
                 END IF
   70 CONTINUE
              ELSE
*
* Compute the Cholesky factorization of a symmetric band
* matrix, given the lower triangle of the matrix in band
* storage.
*
* Zero the lower triangle of the work array.
*
                 DO 90 J = 1, NB
                    DO 80 I = J + 1, NB
                       WORK( I, J ) = ZERO
   80 CONTINUE
   90 CONTINUE
*
* Process the band matrix one diagonal block at a time.
*
            DO 140 I = 1, N, NB
               IB = MIN( NB, N−I+1 )
*
* Factorize the diagonal block
*
               CALL SPOTF2( UPLO, IB, AB( 1, I ), LDAB−1, II )
               IF( II.NE.0 ) THEN
                  INFO = I + II − 1
                  GO TO 150
               END IF
               IF( I+IB.LE.N ) THEN
*
* Update the relevant part of the trailing submatrix.
* If A11 denotes the diagonal block which has just been
* factorized, then we need to update the remaining
* blocks in the diagram:
*
* A11
* A21 A22
* A31 A32 A33
*
* The numbers of rows and columns in the partitioning
* are IB, I2, I3 respectively. The blocks A21, A22 and
* A32 are empty if IB = KD. The lower triangle of A31
* lies outside the band.
*
                  I2 = MIN( KD−IB, N−I−IB+1 )
                  I3 = MIN( IB, N−I−KD+1 )
*
* Calculate the number of zero rows/cols and avoid doing the
* work for the zero rows/cols (using only IHIGH2)
*
                  CALL SBZCHK( I2, IB, AB(1+IB,I) , LDAB−1, 'row',
     $    ILOW2, IHIGH2 )

                  IF( IHIGH2.GT.0 ) THEN
*
* Update A21
*
                     CALL STRSM( 'Right', 'Lower', 'Transpose',
     $ 'Non−unit', IHIGH2, IB, ONE, AB( 1, I),
     $ LDAB−1, AB( 1+IB, I ), LDAB−1 )
*
* Update A22
*
                     CALL SSYRK( 'Lower', 'No Transpose', IHIGH2, IB,
     $ −ONE, AB( 1+IB, I ), LDAB−1, ONE,
     $ AB( 1, I+IB ), LDAB−1 )
                  END IF
*
                  IF( I3.GT.0 ) THEN
*
* Copy the upper triangle of A31 into the work array.
*
                     DO 110 JJ = 1, IB
                        DO 100 II = 1, MIN( JJ, I3 )
                           WORK( II, JJ ) = AB( KD+1−JJ+II, JJ+I−1 )
  100 CONTINUE
  110 CONTINUE
*
* Update A31 (in the work array).
*
                     CALL STRSM( 'Right', 'Lower', 'Transpose',
     $                      'Non-unit', I3, IB, ONE, AB( 1, I ),
     $                      LDAB-1, WORK, LDWORK )
*
*                 Update A32
*
                  IF( IHIGH2.GT.0 )
     $               CALL SGEMM( 'No transpose', 'Transpose', I3,
     $                           IHIGH2, IB, -ONE, WORK, LDWORK,
     $                           AB( 1+IB, I ), LDAB-1, ONE,
     $                           AB( 1+KD-IB, I+IB ), LDAB-1 )
*
*                 Update A33
*
                  CALL SSYRK( 'Lower', 'No Transpose', I3, IB, -ONE,
     $                        WORK, LDWORK, ONE, AB( 1, I+KD ),
     $                        LDAB-1 )
*
*                 Copy the upper triangle of A31 back into place.
*
                  DO 130 JJ = 1, IB
                     DO 120 II = 1, MIN( JJ, I3 )
                        AB( KD+1-JJ+II, JJ+I-1 ) = WORK( II, JJ )
  120                CONTINUE
  130             CONTINUE
               END IF
            END IF
  140    CONTINUE
         END IF
      END IF
      RETURN
*
  150 CONTINUE
      RETURN
*
*     End of SPBTRF
*
      END
```

A.3.5 asd2_sgetrf.f

```
*> \brief \b ASD2_SGETRF
*
*  =========== DOCUMENTATION ===========
*
* Online html documentation available at
*            http://www.netlib.org/lapack/explore-html/
*
*> \htmlonly
*> Download SGETRF + dependencies
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.tgz?format=tgz&filename=/lapack/lapack_routine/sgetrf.f">
*> [TGZ]</a>
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.zip?format=zip&filename=/lapack/lapack_routine/sgetrf.f">
*> [ZIP]</a>
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.txt?format=txt&filename=/lapack/lapack_routine/sgetrf.f">
*> [TXT]</a>
*> \endhtmlonly
*
*  Definition:
*  ===========
*
*       SUBROUTINE ASD2_SGETRF( M, N, A, LDA, IPIV, INFO )
*
*       .. Scalar Arguments ..
*       INTEGER            INFO, LDA, M, N
* ..
* .. Array Arguments ..
* INTEGER IPIV( * )
* REAL A( LDA, * )
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> ASD2_SGETRF computes an LU factorization of a general M-by-N matrix A
*> using partial pivoting with row interchanges and multi-block sparsity detection
*>
*> The factorization has the form
*> A = P * L * U
*> where P is a permutation matrix, L is lower triangular with unit
*> diagonal elements (lower trapezoidal if m > n), and U is upper
*> triangular (upper trapezoidal if m < n).
*>
*> This is the right-looking Level 3 BLAS version of the algorithm.
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] M
*> \verbatim
*> M is INTEGER
*> The number of rows of the matrix A. M >= 0.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The number of columns of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in,out] A
*> \verbatim
*> A is REAL array, dimension (LDA,N)
*> On entry, the M-by-N matrix to be factored.
*> On exit, the factors L and U from the factorization
*> A = P*L*U; the unit diagonal elements of L are not stored.
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> The leading dimension of the array A. LDA >= max(1,M).
*> \endverbatim
*>
*> \param[out] IPIV
*> \verbatim
*> IPIV is INTEGER array, dimension (min(M,N))
*> The pivot indices; for 1 <= i <= min(M,N), row i of the
*> matrix was interchanged with row IPIV(i).
*> \endverbatim
*>
*> \param[out] INFO
*> \verbatim
*> INFO is INTEGER
*> = 0: successful exit
*> < 0: if INFO = -i, the i-th argument had an illegal value
*> > 0: if INFO = i, U(i,i) is exactly zero. The factorization
*> has been completed, but the factor U is exactly
*> singular, and division by zero will occur if it is used
*> to solve a system of equations.
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2014
*
*> \ingroup realGEcomputational
*
* =====================================================================
      SUBROUTINE ASD2_SGETRF( M, N, A, LDA, IPIV, INFO )
*
* -- LAPACK computational routine (version 3.6.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2014
*
* .. Scalar Arguments ..
      INTEGER INFO, LDA, M, N
* ..
* .. Array Arguments ..
      INTEGER IPIV( * )
      REAL A( LDA, * )
* ..
*
* =====================================================================
*
* .. Parameters ..
      REAL ONE
      PARAMETER ( ONE = 1.0E+0 )
* ..
* .. Local Scalars ..
      INTEGER I, IINFO, J, JB, NB, NNZ, MIS, NIS, MINLVL,
     $ MINCOMB,SC
      REAL FRACNZ
* ..
* .. Local Arrays ..
      INTEGER MILIM(M/8), NILIM(N/8)
* ..
* .. External Subroutines ..
      EXTERNAL SGEMM, SGETF2, SLASWP, STRSM, XERBLA, SSZCHK,
     $ SBZCHK_ILIM, SGEMM_ILIM, SGETF2_ILIM,
     $ STRSM_ILIM, IZCOMB, IZSUB
* ..
* .. External Functions ..
      INTEGER ILAENV, IZSS
      EXTERNAL ILAENV, IZSS
* ..
* .. Intrinsic Functions ..
      INTRINSIC MAX, MIN
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'ASD2_SGETRF', -INFO )
         RETURN
      END IF
*
* Quick return if possible
*
      IF( M.EQ.0 .OR. N.EQ.0 )
     $ RETURN
*
* Determine the block size for this environment.
*
      NB = ILAENV( 1, 'SGETRF', ' ', M, N, -1, -1 )
      IF( NB.LE.1 .OR. NB.GE.MIN( M, N ) ) THEN
*
* Use unblocked code.
*
         CALL SGETF2( M, N, A, LDA, IPIV, INFO )
      ELSE

*
* Initialize all values SC - sparsity constant, MINLVL - size of min block
* and MINCOMB - size at which to ignore zero rows and combine neighbouring blocks
* MILIM and NILIM are 2*M/MINLVL and 2*N/MINLVL respectively
*
      SC = 2
      MINLVL = 16
      MINCOMB = 4
*
* Calculate Sparsity
*
* PRINT*,"Calling SSZCHK"
       CALL SSZCHK( M, N, A, LDA, SC, 'pure_random',FRACNZ, NNZ)
* PRINT*,"Finished SSZCHK, FRACNZ=",FRACNZ
* FRACNZ=0.2
*
* Use blocked code.
*
         DO 20 J = 1, MIN( M, N ), NB

            JB = MIN( MIN( M, N )-J+1, NB )
*
* Calculate number of blocks to use
*

            MIS = IZSS (M-J+1, LDA, FRACNZ,MINLVL)
            NIS = IZSS (N-J+1, LDA, FRACNZ,MINLVL)
*
* Calculate all 2*MIS low-high values for panel in MILIM by MIS
* calls to SBZCHK
*
            CALL SBZCHK_ILIM(M-J+1, JB, A(J ,J),LDA,'row', MILIM, MIS)
*
* Combining MILIM results
*
            CALL IZCOMB (MILIM,MIS,MINCOMB)
*
* Factor diagonal and subdiagonal blocks and test for exact
* singularity taking into account MILIM.
*
            CALL SGETF2_ILIM( M-J+1, JB, A( J, J ), LDA, IPIV( J ),
     $ MILIM, MIS, IINFO )
*
* Adjust INFO and the pivot indices.
*
            IF( INFO.EQ.0 .AND. IINFO.GT.0 )
     $ INFO = IINFO + J - 1
```

```
            DO 10 I = J, MIN( M, J+JB−1 )
               IPIV( I ) = J − 1 + IPIV( I )
   10    CONTINUE
*
* Apply interchanges to columns 1:J−1.
*
            CALL SLASWP( J−1, A, LDA, J, J+JB−1, IPIV, 1 )
*
            IF( J+JB.LE.N ) THEN
*
* Apply interchanges to columns J+JB:N.
*
               CALL SLASWP( N−J−JB+1, A( 1, J+JB ), LDA, J, J+JB−1,
     $ IPIV, 1 )
*
* Calculate the number of zero columns and avoid doing
* the work for the zero columns by using NILIM
*
               CALL SBZCHK_ILIM( JB, N−J−JB+1, A(J ,J+JB) , LDA, 'col',
     $ NILIM, NIS, MINCOMB)

*
* Combining MILIM results
*
               CALL IZCOMB (NILIM,NIS,MINCOMB)
*
* Compute block row of U.
*
               CALL STRSM_ILIM( 'Left', 'Lower', 'No transpose', 'Unit',
     $ JB, N−J−JB+1, ONE, A( J,J ), LDA, A( J, J+JB ),
     $ LDA, NILIM, NIS)
*
*  Subtract JB from start of MILIM
*
               CALL IZSUB(MILIM,MIS,JB)
*
*  If needed update trailing matrix
*
               IF( MIS.GT.0) THEN
*
* Update trailing submatrix using sparse block multiply.
*
                  CALL SGEMM_ILIM( 'No transpose', 'No transpose',
     $ M−J−JB+1, N−J−JB+1, JB, −ONE,
     $ A( J+JB, J ), LDA, A( J, J+JB ),
     $ LDA, ONE, A( J+JB, J+JB ), LDA,
     $ MILIM, MIS, NILIM, NIS)
               END IF
            END IF
   20    CONTINUE
      END IF
      RETURN
*
* End of ASD2_SGETRF
*
      END
```

A.3.6 asd2_spotrf.f

```
*> \brief \b ASD2_SPOTRF
*
* =========== DOCUMENTATION ===========
*
* Online html documentation available at
* http://www.netlib.org/lapack/explore-html/
*
*> \htmlonly
*> Download SPOTRF + dependencies
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.tgz?format=tgz&filename=/lapack/lapack_routine/spotrf.f">
*> [TGZ]</a>
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.zip?format=zip&filename=/lapack/lapack_routine/spotrf.f">
*> [ZIP]</a>
*> <a href="http://www.netlib.org/cgi-bin/netlibfiles.txt?format=txt&filename=/lapack/lapack_routine/spotrf.f">
*> [TXT]</a>
*> \endhtmlonly
*
* Definition:
* ===========
*
* SUBROUTINE ASD2_SPOTRF( UPLO, N, A, LDA, INFO )
*
* .. Scalar Arguments ..
* CHARACTER UPLO
* INTEGER INFO, LDA, N
* ..
* .. Array Arguments ..
* REAL A( LDA, * )
* ..
*
*
*> \par Purpose:
* =============
*>
*> \verbatim
*>
*> ASD2_SPOTRF computes the Cholesky factorization of a real symmetric
*> positive definite matrix A taking into account sparsity.
*>
*> The factorization has the form
*> A = U**T * U, if UPLO = 'U', or
*> A = L * L**T, if UPLO = 'L',
*> where U is an upper triangular matrix and L is lower triangular.
*>
*> This is the block version of the algorithm, calling Level 3 BLAS.
*> \endverbatim
*
* Arguments:
* ==========
*
*> \param[in] UPLO
*> \verbatim
*> UPLO is CHARACTER*1
*> = 'U': Upper triangle of A is stored;
*> = 'L': Lower triangle of A is stored.
*> \endverbatim
*>
*> \param[in] N
*> \verbatim
*> N is INTEGER
*> The order of the matrix A. N >= 0.
*> \endverbatim
*>
*> \param[in,out] A
*> \verbatim
*> A is REAL array, dimension (LDA,N)
*> On entry, the symmetric matrix A. If UPLO = 'U', the leading
*> N-by-N upper triangular part of A contains the upper
*> triangular part of the matrix A, and the strictly lower
*> triangular part of A is not referenced. If UPLO = 'L', the
*> leading N-by-N lower triangular part of A contains the lower
*> triangular part of the matrix A, and the strictly upper
*> triangular part of A is not referenced.
```

```
*>
*> On exit, if INFO = 0, the factor U or L from the Cholesky
*> factorization A = U**T*U or A = L*L**T.
*> \endverbatim
*>
*> \param[in] LDA
*> \verbatim
*> LDA is INTEGER
*> The leading dimension of the array A. LDA >= max(1,N).
*> \endverbatim
*>
*> \param[out] INFO
*> \verbatim
*> INFO is INTEGER
*> = 0: successful exit
*> < 0: if INFO = -i, the i-th argument had an illegal value
*> > 0: if INFO = i, the leading minor of order i is not
*> positive definite, and the factorization could not be
*> completed.
*> \endverbatim
*
* Authors:
* ========
*
*> \author Univ. of Tennessee
*> \author Univ. of California Berkeley
*> \author Univ. of Colorado Denver
*> \author NAG Ltd.
*
*> \date November 2011
*
*> \ingroup realPOcomputational
*
* =====================================================================
      SUBROUTINE ASD2_SPOTRF( UPLO, N, A, LDA, INFO )
*
* -- LAPACK computational routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
* November 2011
*
* .. Scalar Arguments ..
      CHARACTER UPLO
      INTEGER INFO, LDA, N
* ..
* .. Array Arguments ..
      REAL A( LDA, * )
* ..
*
* =====================================================================
*
* .. Parameters ..
      REAL ONE
      PARAMETER ( ONE = 1.0E+0 )
* ..
* .. Local Scalars ..
      LOGICAL UPPER
      INTEGER J, JB, NB, NNZ, MIS, NIS, MINLVL, MINCOMB,SC,
     $ MIS2,NIS2
      REAL FRACNZ
* ..
* .. Local Arrays ..
      INTEGER MILIM(N/8), NILIM(N/8)
* ..
* .. External Functions ..
      LOGICAL LSAME
      INTEGER ILAENV,IZSS
      EXTERNAL LSAME, ILAENV,IZSS
```

```
* ..
* .. External Subroutines ..
      EXTERNAL SGEMM, SPOTF2, SSYRK, STRSM, XERBLA, SSZCHK,
     $ SBZCHK_ILIM, SGEMM_ILIM, SSYRK_ILIM,
     $ STRSM_ILIM, IZCOMB
* ..
* .. Intrinsic Functions ..
      INTRINSIC MAX, MIN
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
      INFO = 0
      UPPER = LSAME( UPLO, 'U' )
      IF( .NOT.UPPER .AND. .NOT.LSAME( UPLO, 'L' ) ) THEN
         INFO = −1
      ELSE IF( N.LT.0 ) THEN
         INFO = −2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
         INFO = −4
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'ASD2_SPOTRF', −INFO )
         RETURN
      END IF
*
* Quick return if possible
*
      IF( N.EQ.0 )
     $ RETURN
*
* Determine the block size for this environment.
*
      NB = ILAENV( 1, 'SPOTRF', UPLO, N, −1, −1, −1 )
      IF( NB.LE.1 .OR. NB.GE.N ) THEN
*
* Use unblocked code.
*
         CALL SPOTF2( UPLO, N, A, LDA, INFO )
      ELSE
* Initialize all values SC − sparsity constant, MINLVL − size of min block
* and MINCOMB − size at which to ignore zero rows and combine neighbouring blocks
*
      SC = 2
      MINLVL = 16
      MINCOMB = 4
*
* Use blocked code.
*
         IF( UPPER ) THEN
*
* Calculate Sparsity
*
            CALL SSZCHK( N, N, A, LDA, SC, 'upper_random',FRACNZ, NNZ)
*
* Compute the Cholesky factorization A = U**T*U.
*
            DO 10 J = 1, N, NB
*
* Calculate number of blocks to use
*
            MIS = IZSS (J−1, LDA, FRACNZ,MINLVL)
            NIS = IZSS (N−J+1, LDA, FRACNZ,MINLVL)
            NIS2 = NIS

            JB = MIN( NB, N−J+1 )
            IF (MIS.GT.0) THEN
```

```
*
* Calculate all 2*MIS low-high values for symetric multiply
* in MILIM by MIS calls to SBZCHK
*
          CALL SBZCHK_ILIM(J-1, JB, A(1,J),LDA,'row', MILIM, MIS)
*
* Combining MILIM results
*
          CALL IZCOMB (MILIM,MIS,MINCOMB)
*
* Update the current diagonal block
*
             CALL SSYRK_ILIM( 'Upper', 'Transpose', JB, J-1, -ONE,
     $ A( 1, J ), LDA, ONE, A( J, J ), LDA ,
     $ MILIM ,MIS)
          END IF
*
* Factorize the current diagonal block and test
* for non-positive-definiteness.
*
             CALL SPOTF2( 'Upper', JB, A( J, J ), LDA, INFO )
             IF( INFO.NE.0 )
     $ GO TO 30

*
* Calculate the number of zero columns and avoid doing
* the work for the zero columns by using NILIM
*
             IF( MIS.GT.0 ) THEN
             CALL SBZCHK_ILIM( J-MILIM(1), N-J-JB+1, A(MILIM(1),J+JB),
     $ LDA, 'col', NILIM, NIS)
*
* Combining NILIM results
*
             CALL IZCOMB (NILIM,NIS,MINCOMB)
               IF( NIS.GT.0 ) THEN
*
* Compute the current block row.
*

                CALL SGEMM_ILIM( 'Transpose', 'No transpose', JB,
     $ N-J-JB+1, J-1, -ONE, A( 1, J ), LDA,
     $ A( 1, J+JB ), LDA, ONE, A( J, J+JB ),LDA,
     $ MILIM,MIS,NILIM,NIS)
               END IF
             END IF
*
* Calculate the number of zero columns and avoid doing
* the work for the zero columns by using NILIM
*
             IF ((N-J-JB+1).GT.0) THEN
                         NIS = NIS2
               CALL SBZCHK_ILIM( JB, N-J-JB+1, A(J ,J+JB) , LDA, 'col',
     $ NILIM, NIS)
*
* Combining NILIM results
*
               CALL IZCOMB (NILIM,NIS,MINCOMB)
*
               IF( NIS.GT.0 ) THEN
                         CALL STRSM_ILIM( 'Left', 'Upper', 'Transpose', 'Non-unit',
     $ JB, N-J-JB+1, ONE, A( J, J ), LDA,
     $ A( J, J+JB ), LDA , NILIM, NIS)
               END IF
                       END IF
   10 CONTINUE
*
        ELSE
```

```
*
* Calculate Sparsity
*
              CALL SSZCHK( N, N, A, LDA, SC, 'lower_random',FRACNZ, NNZ)
*
* Compute the Cholesky factorization A = L*L**T.
*
              DO 20 J = 1, N, NB
*
* Update and factorize the current diagonal block and test
* for non-positive-definiteness.
*

              MIS = IZSS (N-J+1, LDA, FRACNZ,MINLVL)
              MIS2 = MIS
              NIS = IZSS (J-1, LDA, FRACNZ,MINLVL)

              JB = MIN( NB, N-J+1 )
              IF (NIS.GT.0) THEN
*
* Calculate all 2*NIS low-high values for symetric multiply
* in NILIM by NIS calls to SBZCHK
*
              CALL SBZCHK_ILIM(JB,J-1, A(J,1),LDA,'col', NILIM, NIS)
*
* Combining MILIM results
*
              CALL IZCOMB (NILIM,NIS,MINCOMB)
*
* Update the current diagonal block
*
                 CALL SSYRK_ILIM( 'Lower', 'No transpose', JB, J-1, -ONE,
     $ A( J, 1 ), LDA, ONE, A( J, J ), LDA ,
     $ NILIM ,NIS)
              END IF
*
* Factorize the current diagonal block and test
* for non-positive-definiteness.
*
                 CALL SPOTF2( 'Lower', JB, A( J, J ), LDA, INFO )
                 IF( INFO.NE.0 )
     $ GO TO 30

*
* Calculate the number of zero columns and avoid doing
* the work for the zero columns by using NILIM
*
                 IF( NIS.GT.0 ) THEN
                 CALL SBZCHK_ILIM( N-J-JB+1, N-MILIM(1), A(J+JB,NILIM(1)),
     $ LDA, 'row', MILIM, MIS)
*
* Combining NILIM results
*
                 CALL IZCOMB (MILIM,MIS,MINCOMB)
                   IF( MIS.GT.0 ) THEN
*
* Compute the current block row.
*

                    CALL SGEMM_ILIM( 'No transpose', 'Transpose',
     $ N-J-JB+1, JB, J-1, -ONE, A( J, 1 ), LDA,
     $ A( J+JB, 1), LDA, ONE, A( J+JB, J ),LDA,
     $ MILIM,MIS,NILIM,NIS)
                   END IF
                 END IF
*
* Calculate the number of zero columns and avoid doing
* the work for the zero columns by using NILIM
```

```
*
               IF ((N−J−JB+1).GT.0) THEN
                           MIS = MIS2
                 CALL SBZCHK_ILIM( N−J−JB+1, JB, A(J+JB, J) , LDA, 'row',
     $ MILIM, MIS)
*
* Combining NILIM results
*
                 CALL IZCOMB (NILIM,NIS,MINCOMB)
*
                 IF( MIS.GT.0 ) THEN
                           CALL STRSM_ILIM( 'Right', 'Lower', 'No transpose', 'Non−unit',
     $ N−J−JB+1, JB, ONE, A( J, J ), LDA,
     $ A( J+JB, J), LDA , MILIM, MIS)
                 END IF
                          END IF
   20 CONTINUE

         END IF
      END IF
      GO TO 40
*
   30 CONTINUE
      INFO = INFO + J − 1
*
   40 CONTINUE
      RETURN
*
* End of SPOTRF
*
      END
```

Appendix B

Raw CSV result data

For those interested in looking at further relations between the data but that are unable to do the full runs on the same machine I'm attaching all the raw CSV files here. All results present the total time for 10 iterations so if comparing with a single run divide the total time for all entries by 10.

B.1 General Matrices - Non Positive Definite

B.1.1 Full Dense Matrix - $n = 200 : 4000$

```
FULL MATRIX TEST
 N,SGETRF,ASD1_SGETRF,ASD2_SGETRF
100,0.001669,0.001659,0.001988
200,0.008248,0.008250,0.008969
300,0.023264,0.023218,0.024920
400,0.050120,0.050170,0.052461
500,0.093314,0.093287,0.096191
600,0.151236,0.151188,0.154746
700,0.232678,0.232522,0.237153
800,0.334602,0.334526,0.339763
900,0.471038,0.470822,0.477023
1000,0.621856,0.621776,0.628691
1100,0.823957,0.824453,0.832126
1200,1.048651,1.048484,1.057328
1300,1.352895,1.352982,1.362549
1400,1.660318,1.659831,1.669925
1500,2.080349,2.080151,2.091379
1600,2.476748,2.476560,2.487932
1700,3.062068,3.060598,3.070311
1800,3.513356,3.512797,3.526236
1900,4.245123,4.245072,4.259410
2000,4.773572,4.775326,4.780624
2100,5.704078,5.713501,5.757049
2200,6.328212,6.321375,6.330382
2300,7.473211,7.473716,7.492138
2400,8.087285,8.054784,8.146673
2500,9.577401,9.577604,9.598547
2600,10.210960,10.291242,10.316407
2700,12.003537,11.933262,11.997421
2800,12.735965,12.675315,12.733733
2900,14.844707,14.778913,14.868130
```

```
3000,15.622427,15.675116,15.670724
3100,17.954920,17.990331,18.027816
3200,18.885355,18.848731,18.869903
3300,21.711861,21.669213,21.730676
3400,22.478360,22.537128,22.498414
3500,25.647983,25.695471,25.633678
3600,26.492309,26.501381,26.548304
3700,30.304324,30.307457,30.325646
3800,31.187988,31.163621,31.173076
3900,35.206543,35.143628,35.124177
4000,36.183847,36.141899,36.155045
```

B.1.2 Full Dense Matrix - $n = 1 : 200$

```
FULL MATRIX TEST
 N,SGETRF,ASD1_SGETRF,ASD2_SGETRF
1,0.000004,0.000004,0.000005
2,0.000005,0.000005,0.000006
3,0.000007,0.000006,0.000008
4,0.000008,0.000008,0.000009
5,0.000010,0.000010,0.000012
6,0.000013,0.000013,0.000014
7,0.000016,0.000015,0.000017
8,0.000019,0.000018,0.000020
9,0.000022,0.000022,0.000024
10,0.000026,0.000026,0.000027
11,0.000032,0.000031,0.000033
12,0.000036,0.000035,0.000037
13,0.000041,0.000041,0.000043
14,0.000044,0.000044,0.000045
15,0.000051,0.000051,0.000052
16,0.000053,0.000052,0.000054
17,0.000061,0.000060,0.000062
18,0.000067,0.000066,0.000067
19,0.000077,0.000097,0.000079
20,0.000076,0.000076,0.000078
21,0.000093,0.000093,0.000094
22,0.000095,0.000094,0.000096
23,0.000107,0.000106,0.000108
24,0.000103,0.000104,0.000105
25,0.000128,0.000127,0.000129
26,0.000124,0.000125,0.000126
27,0.000145,0.000145,0.000147
28,0.000134,0.000134,0.000135
29,0.000168,0.000168,0.000168
30,0.000161,0.000161,0.000163
31,0.000189,0.000188,0.000190
32,0.000177,0.000177,0.000178
33,0.000223,0.000224,0.000225
34,0.000202,0.000202,0.000218
35,0.000279,0.000279,0.000280
36,0.000214,0.000214,0.000215
37,0.000313,0.000312,0.000313
38,0.000254,0.000254,0.000255
39,0.000337,0.000337,0.000339
40,0.000264,0.000264,0.000266
41,0.000377,0.000376,0.000379
42,0.000311,0.000325,0.000311
43,0.000404,0.000404,0.000406
44,0.000315,0.000316,0.000316
45,0.000452,0.000468,0.000454
46,0.000372,0.000371,0.000372
47,0.000485,0.000485,0.000486
48,0.000381,0.000381,0.000384
```

```
49,0.000531,0.000531,0.000533
50,0.000462,0.000440,0.000442
51,0.000570,0.000569,0.000572
52,0.000442,0.000461,0.000443
53,0.000620,0.000621,0.000623
54,0.000516,0.000531,0.000516
55,0.000667,0.000667,0.000670
56,0.000514,0.000529,0.000515
57,0.000722,0.000723,0.000724
58,0.000616,0.000601,0.000603
59,0.000773,0.000773,0.000795
60,0.000596,0.000593,0.000598
61,0.000830,0.000849,0.000832
62,0.000693,0.000691,0.000709
63,0.000889,0.000890,0.000891
64,0.000725,0.000705,0.000704
65,0.001025,0.001019,0.001235
66,0.000847,0.000848,0.001074
67,0.001094,0.001077,0.001310
68,0.000845,0.000823,0.001061
69,0.001134,0.001117,0.001353
70,0.000970,0.000953,0.001194
71,0.001196,0.001179,0.001427
72,0.000907,0.000898,0.001153
73,0.001217,0.001200,0.001460
74,0.001039,0.001036,0.001297
75,0.001292,0.001296,0.001565
76,0.001005,0.001007,0.001287
77,0.001319,0.001320,0.001609
78,0.001159,0.001179,0.001432
79,0.001440,0.001420,0.001699
80,0.001095,0.001097,0.001372
81,0.001432,0.001434,0.001724
82,0.001248,0.001265,0.001531
83,0.001555,0.001536,0.001827
84,0.001210,0.001212,0.001513
85,0.001576,0.001596,0.001878
86,0.001398,0.001382,0.001680
87,0.001673,0.001692,0.001980
88,0.001324,0.001309,0.001607
89,0.001702,0.001702,0.002013
90,0.001503,0.001487,0.001797
91,0.001815,0.001820,0.002127
92,0.001464,0.001444,0.001758
93,0.001866,0.001874,0.002177
94,0.001645,0.001631,0.001953
95,0.001988,0.001973,0.002301
96,0.001584,0.001605,0.001909
97,0.002020,0.002002,0.002340
98,0.001779,0.001765,0.002106
99,0.002112,0.002103,0.002441
100,0.001738,0.001764,0.002076
101,0.002189,0.002199,0.002547
102,0.001945,0.001971,0.002289
103,0.002312,0.002315,0.002662
104,0.001880,0.001902,0.002231
105,0.002380,0.002387,0.002747
106,0.002107,0.002122,0.002470
107,0.002495,0.002509,0.002866
108,0.002085,0.002065,0.002434
109,0.002601,0.002587,0.002968
110,0.002302,0.002308,0.002676
111,0.002694,0.002706,0.003078
112,0.002253,0.002239,0.002612
113,0.002801,0.002794,0.003173
114,0.002491,0.002476,0.002859
115,0.002920,0.002908,0.003291
116,0.002425,0.002413,0.002800
```

```
117,0.003014,0.003010,0.003400
118,0.002692,0.002681,0.003073
119,0.003132,0.003130,0.003538
120,0.002592,0.002596,0.002983
121,0.003226,0.003218,0.003617
122,0.002891,0.002879,0.003276
123,0.003355,0.003354,0.003776
124,0.002819,0.002806,0.003214
125,0.003457,0.003464,0.003887
126,0.003105,0.003102,0.003519
127,0.003606,0.003615,0.004041
128,0.003109,0.003092,0.003493
129,0.003790,0.003779,0.004234
130,0.003385,0.003378,0.003828
131,0.003902,0.003905,0.004391
132,0.003305,0.003302,0.003761
133,0.004030,0.004019,0.004497
134,0.003644,0.003629,0.004102
135,0.004159,0.004154,0.004633
136,0.003507,0.003498,0.003974
137,0.004230,0.004234,0.004732
138,0.003872,0.003865,0.004349
139,0.004408,0.004395,0.004910
140,0.003830,0.003814,0.004311
141,0.004673,0.004677,0.005207
142,0.004314,0.004307,0.004804
143,0.004975,0.004984,0.005497
144,0.003988,0.003986,0.004477
145,0.004741,0.004725,0.005262
146,0.004359,0.004357,0.004876
147,0.004933,0.004936,0.005466
148,0.004248,0.004235,0.004761
149,0.005059,0.005045,0.005584
150,0.004634,0.004636,0.005178
151,0.005232,0.005215,0.005765
152,0.004478,0.004481,0.005030
153,0.005315,0.005295,0.005851
154,0.004874,0.004880,0.005416
155,0.005508,0.005509,0.006078
156,0.004840,0.004835,0.005369
157,0.005812,0.005814,0.006385
158,0.005405,0.005394,0.005954
159,0.006208,0.006190,0.006752
160,0.005102,0.005095,0.005652
161,0.005904,0.005910,0.006482
162,0.005477,0.005465,0.006036
163,0.006125,0.006108,0.006692
164,0.005370,0.005377,0.005932
165,0.006247,0.006247,0.006849
166,0.005797,0.005808,0.006379
167,0.006489,0.006491,0.007096
168,0.005645,0.005653,0.006229
169,0.006552,0.006552,0.007170
170,0.006107,0.006100,0.006701
171,0.006826,0.006834,0.007459
172,0.006084,0.006087,0.006680
173,0.007179,0.007190,0.007819
174,0.006748,0.006749,0.007356
175,0.007599,0.007601,0.008244
176,0.006329,0.006320,0.006930
177,0.007287,0.007286,0.007925
178,0.006834,0.006839,0.007449
179,0.007537,0.007538,0.008174
180,0.006679,0.006674,0.007287
181,0.007721,0.007719,0.008386
182,0.007229,0.007249,0.007860
183,0.007957,0.007968,0.008612
184,0.007010,0.006994,0.007626
```

```
185,0.008070,0.008076,0.008743
186,0.007527,0.007535,0.008178
187,0.008319,0.008326,0.009047
188,0.007477,0.007474,0.008122
189,0.008745,0.008742,0.009415
190,0.008238,0.008239,0.008899
191,0.009238,0.009255,0.009946
192,0.007812,0.007819,0.008479
193,0.008948,0.008948,0.009663
194,0.008372,0.008372,0.009083
195,0.009277,0.009285,0.010002
196,0.008199,0.008202,0.008905
197,0.009358,0.009369,0.010100
198,0.008806,0.008800,0.009516
199,0.009661,0.009654,0.010426
200,0.008610,0.008615,0.009348
```

B.1.3 Band Matrix - $n = 200 : 4000$ and $b = 100 : 2000$

```
BANDED MATRIX TEST
 N,B,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SGBTRF,ASD1_SGBTRF
200,100,0.008247,0.007580,0.008337,0.006498,0.006797
400,100,0.049306,0.027803,0.030306,0.018088,0.019520
400,200,0.049642,0.042556,0.045065,0.036479,0.037558
600,100,0.146093,0.054745,0.058595,0.029953,0.032583
600,200,0.147246,0.086659,0.090446,0.067584,0.070768
600,300,0.148520,0.121330,0.125041,0.108682,0.111242
800,100,0.321439,0.086574,0.092351,0.040559,0.044262
800,200,0.324180,0.136493,0.142130,0.097610,0.103035
800,300,0.325702,0.201392,0.206798,0.172363,0.177925
800,400,0.327577,0.260587,0.265977,0.239022,0.243001
1000,100,0.594155,0.132409,0.139674,0.052289,0.057120
1000,200,0.597580,0.198458,0.205760,0.127362,0.134856
1000,300,0.600528,0.288230,0.295647,0.231144,0.240156
1000,400,0.604057,0.391075,0.398174,0.348604,0.356839
1000,500,0.606847,0.474120,0.481273,0.447207,0.453335
1200,100,1.000780,0.176149,0.186125,0.063216,0.069134
1200,200,1.006221,0.262784,0.272559,0.158450,0.168133
1200,300,1.009620,0.376103,0.385600,0.287431,0.299265
1200,400,1.014361,0.514989,0.523896,0.443793,0.456256
1200,500,1.018693,0.652362,0.660938,0.603148,0.614804
1200,600,1.023426,0.777945,0.786420,0.751843,0.760572
1400,100,1.573018,0.237340,0.248306,0.074412,0.081380
1400,200,1.580299,0.339926,0.351332,0.189447,0.201262
1400,300,1.587062,0.488313,0.499458,0.357634,0.372887
1400,400,1.593450,0.656937,0.668267,0.546893,0.563547
1400,500,1.601848,0.839046,0.849958,0.755971,0.772537
1400,600,1.608380,1.026297,1.036902,0.974470,0.990063
1400,700,1.616462,1.200467,1.210786,1.157769,1.169745
1600,100,2.323125,0.286410,0.302035,0.086566,0.094740
1600,200,2.333883,0.412941,0.427763,0.220781,0.234682
1600,300,2.343836,0.591600,0.605494,0.415959,0.434714
1600,400,2.351452,0.795316,0.809237,0.642834,0.663786
1600,500,2.362401,1.020457,1.033354,0.900773,0.923032
1600,600,2.373197,1.263955,1.276691,1.178506,1.200447
1600,700,2.386097,1.540611,1.554014,1.465795,1.485229
1600,800,2.395535,1.767411,1.780040,1.717320,1.733215
1800,100,3.280253,0.369568,0.384922,0.096762,0.106334
1800,200,3.291826,0.508404,0.524077,0.250981,0.267189
1800,300,3.306078,0.707439,0.722603,0.467802,0.489626
1800,400,3.321862,0.957209,0.971990,0.745780,0.771246
1800,500,3.336730,1.242480,1.257469,1.062394,1.090249
1800,600,3.351889,1.534484,1.549233,1.394873,1.422902
1800,700,3.364452,1.866771,1.881724,1.735332,1.762524
```

```
1800,800,3.385016,2.220397,2.234956,2.123030,2.147556
1800,900,3.402869,2.492792,2.507448,2.424059,2.444939
2000,100,4.474759,0.424979,0.446361,0.108242,0.118781
2000,200,4.488160,0.587698,0.607840,0.278781,0.297273
2000,300,4.508112,0.828005,0.846912,0.536346,0.561568
2000,400,4.521229,1.110159,1.127873,0.840196,0.869453
2000,500,4.540001,1.437958,1.455193,1.207563,1.240679
2000,600,4.556378,1.792433,1.809165,1.597974,1.633391
2000,700,4.579361,2.181133,2.197667,1.997868,2.032708
2000,800,4.607357,2.645937,2.662268,2.484391,2.516987
2000,900,4.625072,3.043336,3.059452,2.944042,2.975030
2000,1000,4.641004,3.392447,3.408429,3.331625,3.357640
2200,100,5.916603,0.532372,0.554383,0.120106,0.132003
2200,200,5.936566,0.714770,0.736135,0.312015,0.332569
2200,300,5.958167,0.966832,0.987294,0.583005,0.611468
2200,400,5.970965,1.288563,1.308646,0.941028,0.974798
2200,500,5.997122,1.649522,1.668952,1.333859,1.372604
2200,600,6.034128,2.095376,2.114225,1.826809,1.868104
2200,700,6.069651,2.561102,2.578252,2.274678,2.316869
2200,800,6.091359,3.079451,3.097370,2.808946,2.850315
2200,900,6.116058,3.588128,3.603743,3.396150,3.437134
2200,1000,6.117796,4.098872,4.136564,3.998071,4.032624
2200,1100,6.157294,4.518684,4.535868,4.395935,4.427520
2400,100,7.613785,0.591536,0.621234,0.131903,0.145238
2400,200,7.643153,0.805409,0.832146,0.342675,0.365355
2400,300,7.653164,1.090493,1.115581,0.652132,0.683802
2400,400,7.674076,1.453679,1.476877,1.040617,1.078973
2400,500,7.707616,1.865273,1.887986,1.496771,1.540859
2400,600,7.762058,2.344676,2.367205,2.015837,2.063817
2400,700,7.794603,2.913038,2.933897,2.547811,2.597789
2400,800,7.812145,3.470043,3.489682,3.138814,3.189309
2400,900,7.844283,4.086449,4.106545,3.854638,3.904683
2400,1000,7.883421,4.718451,4.740896,4.550263,4.597335
2400,1100,7.899604,5.361270,5.382482,5.220433,5.263621
2400,1200,7.952374,5.849510,5.874584,5.702681,5.737344
2600,100,9.627633,0.722023,0.750043,0.141611,0.155742
2600,200,9.661480,0.953226,0.980709,0.374566,0.399673
2600,300,9.698092,1.272197,1.298997,0.710790,0.746035
2600,400,9.735083,1.679445,1.704476,1.151386,1.193826
2600,500,9.789987,2.143766,2.168128,1.649231,1.698718
2600,600,9.812556,2.701523,2.725211,2.241491,2.295900
2600,700,9.859054,3.333531,3.357134,2.841771,2.899474
2600,800,9.905381,4.067491,4.087094,3.579436,3.638460
2600,900,9.938818,4.701406,4.725731,4.301011,4.360833
2600,1000,9.962876,5.385211,5.406365,5.079074,5.137639
2600,1100,10.022845,6.159256,6.168940,5.896081,5.952059
2600,1200,10.032358,6.817900,6.857137,6.596727,6.645564
2600,1300,10.076182,7.387984,7.423187,7.274808,7.300572
2800,100,11.939805,0.782095,0.818759,0.152604,0.168021
2800,200,11.982287,1.031964,1.066533,0.397875,0.424908
2800,300,12.022788,1.378208,1.410007,0.761051,0.799660
2800,400,12.064067,1.827293,1.856731,1.234782,1.281316
2800,500,12.089800,2.348777,2.375252,1.808711,1.863980
2800,600,12.091384,2.924926,2.961160,2.420447,2.481264
2800,700,12.202220,3.711254,3.736238,3.145031,3.209964
2800,800,12.243085,4.415296,4.432544,3.868396,3.935969
2800,900,12.300005,5.222576,5.245391,4.761975,4.831669
2800,1000,12.263520,5.919334,5.941124,5.568488,5.637467
2800,1100,12.334490,6.835035,6.858916,6.538742,6.605109
2800,1200,12.392413,7.759926,7.802846,7.510243,7.571658
2800,1300,12.437257,8.502689,8.525485,8.351120,8.440181
2800,1400,12.490605,9.056692,9.090090,9.051986,9.100637
3000,100,14.605510,0.948426,0.982670,0.164779,0.181584
3000,200,14.652760,1.217289,1.251470,0.432197,0.461339
3000,300,14.684486,1.594490,1.627383,0.828750,0.870574
3000,400,14.697321,2.106548,2.137467,1.371265,1.422262
3000,500,14.762829,2.669892,2.703050,1.948150,2.008928
3000,600,14.886468,3.340092,3.367620,2.625909,2.693207
```

```
3000,700,14.943617,4.153567,4.180873,3.381625,3.454194
3000,800,14.987725,4.965868,4.982827,4.219542,4.295317
3000,900,15.056351,5.778000,5.802544,5.108640,5.188611
3000,1000,15.076079,6.651621,6.726962,6.155990,6.235760
3000,1100,15.142167,7.632930,7.617149,7.182988,7.263367
3000,1200,15.201682,8.677928,8.662845,8.223066,8.315384
3000,1300,15.235752,9.633069,9.585612,9.341853,9.426788
3000,1400,15.286844,10.425640,10.439426,10.356995,10.402326
3000,1500,15.305869,11.119643,11.191560,11.071139,11.125963
3200,100,17.701100,1.025241,1.074293,0.176770,0.194552
3200,200,17.758266,1.330684,1.373245,0.462889,0.494506
3200,300,17.763353,1.726888,1.767578,0.880150,0.925342
3200,400,17.906276,2.283133,2.318945,1.449729,1.505010
3200,500,17.944871,2.910876,2.946105,2.090804,2.156919
3200,600,17.997208,3.653262,3.673947,2.851829,2.925710
3200,700,18.054007,4.524378,4.556418,3.650158,3.729411
3200,800,18.118240,5.479496,5.490059,4.641910,4.726439
3200,900,18.175758,6.356237,6.359896,5.594013,5.682814
3200,1000,18.231415,7.334648,7.332890,6.684068,6.774181
3200,1100,18.304123,8.386327,8.371990,7.949307,8.032402
3200,1200,18.349037,9.552643,9.611214,8.980706,9.068071
3200,1300,18.355765,10.586693,10.598709,10.167943,10.264080
3200,1400,18.444774,11.609477,11.669404,11.558454,11.612784
3200,1500,18.489163,12.636549,12.660771,12.506949,12.613180
3200,1600,18.512962,13.440488,13.463297,13.308507,13.374000
3400,100,21.074339,1.198676,1.238402,0.188300,0.207512
3400,200,21.094088,1.518377,1.558528,0.490052,0.523722
3400,300,21.172827,1.964794,2.005441,0.947530,0.995982
3400,400,21.372498,2.537536,2.573869,1.529259,1.588596
3400,500,21.434460,3.259858,3.295072,2.248716,2.319982
3400,600,21.498330,4.069989,4.106361,3.053231,3.134102
3400,700,21.483446,5.013127,5.076703,3.932936,4.020301
3400,800,21.638867,6.020204,6.054651,4.943975,5.036868
3400,900,21.556741,6.989437,7.028257,6.010118,6.108839
3400,1000,21.737835,8.012102,8.085116,7.197282,7.298164
3400,1100,21.859269,9.299053,9.366909,8.566232,8.668438
3400,1200,21.878424,10.542744,10.586233,9.858642,9.945813
3400,1300,21.870539,11.760045,11.785488,11.212248,11.269323
3400,1400,21.987896,12.763890,12.725936,12.486980,12.571514
3400,1500,22.027146,14.053663,14.095547,13.833388,13.924036
3400,1600,21.933295,15.167232,15.142692,14.925161,15.003267
3400,1700,22.028558,16.117326,16.122124,15.943571,16.078040
3600,100,24.989225,1.259912,1.313315,0.197634,0.217927
3600,200,25.064806,1.595272,1.643436,0.518456,0.554178
3600,300,25.151222,2.110853,2.153118,1.016220,1.067605
3600,400,25.122326,2.736008,2.777784,1.660792,1.724587
3600,500,25.347232,3.512202,3.550467,2.434189,2.511207
3600,600,25.404690,4.408680,4.425191,3.306685,3.393589
3600,700,25.481588,5.334292,5.371735,4.151927,4.246927
3600,800,25.443675,6.487181,6.505720,5.367867,5.469324
3600,900,25.595999,7.545301,7.597372,6.499260,6.608396
3600,1000,25.642921,8.737858,8.794832,7.892486,8.004516
3600,1100,25.602472,10.043865,10.061028,9.164898,9.270596
3600,1200,25.799550,11.206194,11.270961,10.587033,10.700009
3600,1300,25.753411,12.639449,12.651296,12.114240,12.218637
3600,1400,25.943289,13.842423,13.821096,13.541050,13.647389
3600,1500,25.926124,15.274667,15.313644,15.056003,15.193603
3600,1600,26.010380,16.729804,16.730309,16.496617,16.585824
3600,1700,26.073622,17.860648,17.878391,17.817174,17.860688
3600,1800,26.088964,18.713647,18.847742,18.893571,19.073667
3800,100,29.338485,1.489658,1.535114,0.211063,0.232666
3800,200,29.444572,1.860143,1.906295,0.552613,0.590031
3800,300,29.468373,2.387745,2.430498,1.071041,1.125964
3800,400,29.440770,3.070104,3.114767,1.743258,1.811430
3800,500,29.747639,3.891604,3.932890,2.537867,2.619923
3800,600,29.812622,4.828147,4.843327,3.463274,3.556816
3800,700,29.940512,5.964299,6.003967,4.500683,4.603475
3800,800,29.971445,7.125581,7.163992,5.655546,5.765638
```

```
3800,900,29.926193,8.402604,8.435691,7.081788,7.199705
3800,1000,30.059543,9.597975,9.617059,8.420183,8.543234
3800,1100,30.293141,11.038502,11.037195,9.957617,10.078423
3800,1200,30.336712,12.380943,12.466863,11.318372,11.510806
3800,1300,30.383639,13.829141,13.876965,13.039862,13.127684
3800,1400,30.465606,15.068111,15.122363,14.543614,14.656284
3800,1500,30.443821,16.822097,16.795222,16.370806,16.437781
3800,1600,30.530703,18.211290,18.307880,17.733347,17.969112
3800,1700,30.448602,19.840524,19.767907,19.570675,19.679327
3800,1800,30.658514,21.074709,21.150363,21.066010,21.270204
3800,1900,30.705088,22.184161,22.220676,22.235373,22.329662
4000,100,34.203640,1.553815,1.616037,0.221718,0.243916
4000,200,34.261881,1.948930,2.008023,0.587165,0.627280
4000,300,34.387781,2.532192,2.583347,1.139364,1.197517
4000,400,34.465560,3.271338,3.319524,1.833721,1.906086
4000,500,34.583218,4.136618,4.180868,2.702240,2.789880
4000,600,34.685814,5.128695,5.161634,3.661926,3.761325
4000,700,34.824155,6.390474,6.407045,4.823862,4.934269
4000,800,34.904716,7.610202,7.639938,6.042159,6.160350
4000,900,34.983497,8.828340,8.893985,7.409640,7.538042
4000,1000,35.005667,10.231377,10.277640,9.000537,9.133043
4000,1100,35.035992,11.689098,11.685446,10.546899,10.683932
4000,1200,35.137697,13.371836,13.388196,12.266835,12.409928
4000,1300,35.298557,14.802824,14.897493,13.928860,14.029542
4000,1400,35.288376,16.224495,16.201274,15.641846,15.781244
4000,1500,35.414627,17.939241,18.037910,17.450245,17.649465
4000,1600,35.474519,19.659677,19.710172,19.178865,19.284242
4000,1700,35.490388,21.538478,21.545650,21.280004,21.349659
4000,1800,35.617025,23.001585,23.052803,23.129726,23.185255
4000,1900,35.562526,24.420658,24.494827,24.474543,24.705984
4000,2000,35.593180,25.647092,25.672891,25.721007,25.776156
```

B.1.4 Band Matrix - $n = 4000$ and $b = 1 : 200$

```
BANDED MATRIX TEST
 N,B,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SGBTRF,ASD1_SGBTRF
4000,1,33.851924,1.237533,1.299566,0.006739,0.006450
4000,2,33.802701,1.256382,1.316375,0.007601,0.007500
4000,3,33.795583,1.277454,1.341845,0.008299,0.008292
4000,4,33.976539,1.288329,1.343372,0.011692,0.011680
4000,5,33.972663,1.292484,1.354176,0.012534,0.012510
4000,6,33.978957,1.301710,1.370637,0.014387,0.014372
4000,7,33.985382,1.304707,1.365243,0.016102,0.016111
4000,8,33.981828,1.312630,1.368564,0.016835,0.016798
4000,9,33.944912,1.314100,1.371039,0.018806,0.018749
4000,10,33.935089,1.313074,1.372661,0.020570,0.020308
4000,11,33.855262,1.313301,1.368514,0.023324,0.023442
4000,12,33.885379,1.324685,1.384024,0.023802,0.023639
4000,13,34.020005,1.320445,1.379433,0.026402,0.026414
4000,14,33.994503,1.325685,1.379188,0.029043,0.029118
4000,15,34.000025,1.329466,1.383563,0.031983,0.032063
4000,16,34.008484,1.335166,1.396264,0.031438,0.031589
4000,17,33.966763,1.334307,1.384141,0.036834,0.036684
4000,18,33.963271,1.330152,1.389738,0.037977,0.037742
4000,19,34.021649,1.339947,1.394161,0.041278,0.041504
4000,20,34.057179,1.342208,1.396021,0.041253,0.041710
4000,21,34.054143,1.345287,1.401049,0.045088,0.045579
4000,22,34.013942,1.345909,1.396766,0.046357,0.046719
4000,23,34.015562,1.352828,1.402411,0.050950,0.051347
4000,24,34.031426,1.353170,1.410225,0.049504,0.049837
4000,25,34.005827,1.356804,1.415682,0.054308,0.054786
4000,26,33.953707,1.351607,1.405333,0.053881,0.054241
4000,27,33.901054,1.358604,1.412073,0.058644,0.059072
4000,28,34.008308,1.359332,1.417555,0.057713,0.058342
```

```
4000,29,34.028917,1.358295,1.410607,0.062549,0.063032
4000,30,34.033749,1.367626,1.422123,0.063620,0.064082
4000,31,34.033633,1.369318,1.420390,0.070380,0.070865
4000,32,34.034946,1.371095,1.435813,0.063615,0.063994
4000,33,34.026473,1.367206,1.423811,0.075721,0.076289
4000,34,34.001607,1.380387,1.445541,0.073871,0.075139
4000,35,33.979219,1.377203,1.427814,0.083032,0.083698
4000,36,33.897522,1.366768,1.428900,0.077476,0.078311
4000,37,34.091620,1.389557,1.448212,0.089077,0.090066
4000,38,34.085823,1.384504,1.438477,0.085556,0.086470
4000,39,34.043454,1.392236,1.442610,0.095816,0.096729
4000,40,34.046790,1.388565,1.452563,0.087263,0.088151
4000,41,34.047596,1.393825,1.451403,0.100743,0.101968
4000,42,34.051763,1.401189,1.453422,0.098391,0.099489
4000,43,34.034987,1.403839,1.456119,0.108025,0.109181
4000,44,33.982440,1.396050,1.452535,0.100450,0.101548
4000,45,33.943013,1.400469,1.456938,0.115238,0.116335
4000,46,33.977733,1.408162,1.461450,0.111482,0.112567
4000,47,34.100232,1.414600,1.466129,0.125301,0.126740
4000,48,34.102181,1.407597,1.481179,0.111660,0.113622
4000,49,34.060005,1.415390,1.474888,0.128887,0.130242
4000,50,34.052139,1.415323,1.463464,0.121212,0.122750
4000,51,34.063337,1.432024,1.480343,0.135189,0.136658
4000,52,33.992536,1.411799,1.478032,0.121681,0.123040
4000,53,33.990217,1.419725,1.475103,0.142009,0.143544
4000,54,34.110793,1.436437,1.487370,0.134456,0.136013
4000,55,34.115345,1.436504,1.483802,0.151965,0.153777
4000,56,34.075132,1.425004,1.487615,0.135196,0.136727
4000,57,34.075856,1.437533,1.491832,0.151091,0.152249
4000,58,34.067676,1.436363,1.487750,0.142302,0.143085
4000,59,34.069962,1.445638,1.495747,0.164960,0.166938
4000,60,34.037051,1.436876,1.501922,0.149160,0.150866
4000,61,34.049174,1.445306,1.500965,0.175001,0.176891
4000,62,34.082823,1.451822,1.500866,0.163271,0.164019
4000,63,34.135110,1.460422,1.507452,0.186459,0.188555
4000,64,34.115057,1.445364,1.508877,0.159515,0.160974
4000,65,34.074770,1.455560,1.517790,0.144464,0.161641
4000,66,34.076299,1.460529,1.519162,0.139152,0.156070
4000,67,34.075970,1.466742,1.516733,0.150028,0.166980
4000,68,34.030576,1.462786,1.518789,0.139563,0.156686
4000,69,34.017319,1.466975,1.514407,0.149575,0.167485
4000,70,34.028998,1.471944,1.522885,0.150072,0.167253
4000,71,34.134107,1.481868,1.527312,0.161412,0.178908
4000,72,34.124589,1.474644,1.546008,0.150504,0.168794
4000,73,34.093215,1.488055,1.560793,0.159142,0.176866
4000,74,34.094297,1.487549,1.558977,0.161220,0.179801
4000,75,34.095034,1.492490,1.540446,0.168355,0.186407
4000,76,34.082714,1.486746,1.546194,0.160234,0.178754
4000,77,34.088635,1.492777,1.542892,0.172694,0.190865
4000,78,34.079221,1.499879,1.549518,0.169422,0.187290
4000,79,34.040709,1.503091,1.546413,0.179826,0.198001
4000,80,34.022368,1.499948,1.562479,0.169419,0.188269
4000,81,34.021344,1.512790,1.581535,0.178069,0.197667
4000,82,34.146442,1.510925,1.570612,0.174907,0.194493
4000,83,34.101962,1.515466,1.568176,0.186913,0.206692
4000,84,34.104174,1.513977,1.575835,0.176155,0.196196
4000,85,34.098394,1.520316,1.568776,0.186911,0.208022
4000,86,34.095242,1.515665,1.567451,0.185188,0.205279
4000,87,34.090092,1.522394,1.572472,0.197330,0.217564
4000,88,34.049279,1.527292,1.595106,0.189352,0.209591
4000,89,34.018237,1.528122,1.597889,0.199862,0.219911
4000,90,34.017501,1.535450,1.601578,0.196092,0.216461
4000,91,34.102781,1.544630,1.599520,0.209456,0.230048
4000,92,34.130951,1.539995,1.603175,0.197733,0.218152
4000,93,34.113908,1.543634,1.594003,0.210477,0.229973
4000,94,34.114936,1.543952,1.592304,0.209743,0.229779
4000,95,34.115893,1.553251,1.599174,0.221637,0.242318
4000,96,34.099294,1.555941,1.632457,0.211669,0.232319
```

```
4000,97,34.070300,1.560520,1.617359,0.222740,0.245101
4000,98,33.958265,1.554559,1.632214,0.220441,0.242848
4000,99,34.099769,1.562251,1.615808,0.232787,0.255859
4000,100,34.118024,1.562336,1.631000,0.221147,0.243717
4000,101,34.117320,1.564887,1.616025,0.235237,0.260331
4000,102,34.126038,1.576824,1.627331,0.234366,0.256955
4000,103,34.119742,1.576680,1.622710,0.245717,0.268098
4000,104,34.123753,1.582090,1.653836,0.235477,0.258070
4000,105,34.069047,1.590040,1.646115,0.246498,0.269458
4000,106,34.025703,1.582544,1.655397,0.243095,0.266137
4000,107,34.108041,1.592831,1.641204,0.258567,0.282302
4000,108,34.172338,1.594808,1.658815,0.246255,0.269516
4000,109,34.149837,1.598872,1.646970,0.259042,0.281960
4000,110,34.126094,1.594741,1.648167,0.254789,0.277759
4000,111,34.131379,1.606860,1.651965,0.270600,0.294091
4000,112,34.136147,1.610330,1.668825,0.255453,0.278699
4000,113,34.094420,1.616981,1.671697,0.269771,0.294761
4000,114,34.058818,1.613266,1.670300,0.268798,0.293632
4000,115,34.132820,1.626207,1.673369,0.280602,0.305662
4000,116,34.176410,1.617104,1.670918,0.269310,0.294834
4000,117,34.133293,1.625836,1.673915,0.281858,0.308035
4000,118,34.135866,1.626287,1.674345,0.282927,0.308084
4000,119,34.145980,1.635965,1.680284,0.293980,0.320393
4000,120,34.136423,1.633666,1.684646,0.281754,0.307341
4000,121,34.080597,1.639500,1.685335,0.294147,0.319743
4000,122,33.991208,1.625201,1.683882,0.292710,0.318264
4000,123,34.119189,1.650721,1.697216,0.307234,0.333537
4000,124,34.151131,1.647206,1.694955,0.296015,0.322008
4000,125,34.142253,1.648309,1.693797,0.307157,0.333213
4000,126,34.146384,1.650434,1.698823,0.305432,0.332138
4000,127,34.155895,1.661178,1.704072,0.321374,0.347809
4000,128,34.124287,1.659671,1.705002,0.307616,0.334996
4000,129,34.086464,1.662359,1.711370,0.323309,0.350779
4000,130,34.001326,1.659020,1.713579,0.320085,0.347958
4000,131,34.136185,1.669505,1.719380,0.334081,0.362781
4000,132,34.185311,1.672166,1.726028,0.318068,0.345971
4000,133,34.152183,1.672193,1.723535,0.335684,0.365225
4000,134,34.158738,1.679127,1.729580,0.336302,0.364650
4000,135,34.155077,1.687228,1.731839,0.347981,0.377528
4000,136,34.167289,1.695561,1.744397,0.337157,0.366046
4000,137,34.138615,1.689599,1.747484,0.346778,0.375609
4000,138,34.112981,1.689546,1.752576,0.346064,0.375183
4000,139,34.184800,1.700342,1.753068,0.361812,0.390963
4000,140,34.216601,1.707808,1.760200,0.347442,0.376966
4000,141,34.168572,1.703493,1.751481,0.361005,0.389866
4000,142,34.160259,1.707930,1.755865,0.361198,0.390883
4000,143,34.165665,1.721927,1.765148,0.377194,0.408097
4000,144,34.137804,1.720597,1.777207,0.361190,0.391187
4000,145,34.139409,1.730405,1.785583,0.379345,0.410310
4000,146,34.159083,1.727134,1.786750,0.374520,0.406375
4000,147,34.214395,1.732545,1.778053,0.386571,0.420207
4000,148,34.182583,1.736204,1.789957,0.373990,0.405472
4000,149,34.178069,1.734153,1.782808,0.389416,0.421492
4000,150,34.170949,1.738754,1.786509,0.389191,0.420996
4000,151,34.171272,1.749562,1.790838,0.407735,0.439677
4000,152,34.148247,1.748883,1.806104,0.392089,0.423382
4000,153,34.090043,1.760466,1.810351,0.409029,0.440619
4000,154,34.095464,1.761180,1.821225,0.402664,0.434757
4000,155,34.232328,1.766541,1.815060,0.415879,0.448888
4000,156,34.216076,1.762705,1.815989,0.400793,0.433709
4000,157,34.183086,1.774925,1.819642,0.421980,0.454246
4000,158,34.185183,1.776672,1.823895,0.423169,0.456617
4000,159,34.177788,1.783319,1.828749,0.434606,0.467403
4000,160,34.114522,1.767542,1.836596,0.420205,0.453506
4000,161,34.087152,1.788168,1.852600,0.440388,0.473503
4000,162,34.231138,1.791663,1.853451,0.432989,0.467247
4000,163,34.233634,1.800391,1.841404,0.448358,0.482091
4000,164,34.194284,1.795442,1.847491,0.429910,0.464700
```

```
4000,165,34.192845,1.811739,1.854437,0.454502,0.490013
4000,166,34.194493,1.807552,1.849049,0.445666,0.481011
4000,167,34.194627,1.821761,1.865192,0.466340,0.501272
4000,168,34.142839,1.813795,1.870495,0.450361,0.485383
4000,169,34.056993,1.807773,1.862727,0.465999,0.501054
4000,170,34.017648,1.815666,1.875577,0.467937,0.503834
4000,171,34.190330,1.819722,1.866244,0.484529,0.519167
4000,172,34.197029,1.832298,1.884127,0.465685,0.501416
4000,173,34.195417,1.843248,1.887318,0.488028,0.523825
4000,174,34.195981,1.843617,1.887405,0.481542,0.517742
4000,175,34.132297,1.843885,1.885197,0.492992,0.529002
4000,176,34.018712,1.840338,1.903935,0.480128,0.516886
4000,177,34.198573,1.857700,1.911313,0.500468,0.537228
4000,178,34.198019,1.851710,1.900017,0.496040,0.533187
4000,179,34.198427,1.866327,1.908319,0.514666,0.551615
4000,180,34.194310,1.869109,1.921834,0.497111,0.534785
4000,181,34.147470,1.872864,1.919860,0.522915,0.560348
4000,182,34.071082,1.871995,1.921074,0.533077,0.571159
4000,183,34.254848,1.886872,1.926859,0.528144,0.566206
4000,184,34.217046,1.880130,1.944400,0.512820,0.551622
4000,185,34.200408,1.885780,1.949408,0.533045,0.571722
4000,186,34.204364,1.888955,1.940432,0.528896,0.568251
4000,187,34.166438,1.900518,1.948067,0.546061,0.584703
4000,188,34.095163,1.892952,1.947138,0.528441,0.568287
4000,189,34.198165,1.917961,1.959210,0.551740,0.590065
4000,190,34.226076,1.916330,1.961960,0.545902,0.584858
4000,191,34.218964,1.927329,1.967478,0.561719,0.600391
4000,192,34.236442,1.929200,1.990540,0.552480,0.590617
4000,193,34.218695,1.935661,1.996981,0.578951,0.618407
4000,194,34.190937,1.934616,1.994295,0.569970,0.611094
4000,195,34.164014,1.952443,1.997559,0.627650,0.668630
4000,196,34.072812,1.930111,1.994126,0.563962,0.604593
4000,197,34.181126,1.954052,2.002670,0.588260,0.629765
4000,198,34.238939,1.957229,2.000310,0.580906,0.621801
4000,199,34.228488,1.969366,2.007772,0.602318,0.643113
4000,200,34.237702,1.970595,2.018354,0.587523,0.628725
```

B.1.5 Arrow Matrix - $n = 200 : 4000$ and $b = 100 : 2000$

```
ARROW MATRIX TEST
 N,B,SGETRF,ASD1_SGETRF,ASD2_SGETRF
200,100,0.008266,0.008268,0.009000
400,100,0.049849,0.049864,0.046789
400,200,0.050182,0.050147,0.052466
600,100,0.147661,0.147522,0.107844
600,200,0.149995,0.149997,0.146016
600,300,0.151278,0.151228,0.154723
800,100,0.325573,0.325445,0.190210
800,200,0.329056,0.328981,0.274682
800,300,0.332530,0.332601,0.325943
800,400,0.335155,0.334910,0.340161
1000,100,0.602002,0.601570,0.298989
1000,200,0.607785,0.607820,0.436405
1000,300,0.614164,0.614235,0.542369
1000,400,0.619659,0.619617,0.613249
1000,500,0.621333,0.621272,0.628232
1200,100,1.014963,1.014526,0.424059
1200,200,1.025167,1.025051,0.633795
1200,300,1.033884,1.033918,0.806979
1200,400,1.043321,1.043398,0.951003
1200,500,1.048451,1.048457,1.032690
1200,600,1.052849,1.052954,1.061537
1400,100,1.596052,1.595684,0.578584
1400,200,1.612436,1.612477,0.870529
```

```
1400,300,1.627530,1.627485,1.122842
1400,400,1.642343,1.642012,1.366995
1400,500,1.655712,1.655390,1.525851
1400,600,1.661743,1.661549,1.646294
1400,700,1.668024,1.668055,1.677775
1600,100,2.364303,2.364070,0.756860
1600,200,2.396254,2.396032,1.156175
1600,300,2.415930,2.415872,1.502127
1600,400,2.435504,2.435501,1.859757
1600,500,2.450526,2.449818,2.093240
1600,600,2.464538,2.464503,2.330700
1600,700,2.474871,2.474715,2.448987
1600,800,2.478790,2.478359,2.489424
1800,100,3.333718,3.333384,0.970345
1800,200,3.373744,3.373499,1.468975
1800,300,3.403507,3.402971,1.917951
1800,400,3.432866,3.433076,2.401118
1800,500,3.453288,3.453050,2.735273
1800,600,3.471123,3.470976,3.099023
1800,700,3.490618,3.489491,3.306778
1800,800,3.496796,3.495717,3.482257
1800,900,3.503019,3.501408,3.510770
2000,100,4.538163,4.537371,1.183396
2000,200,4.591784,4.591702,1.820090
2000,300,4.629806,4.627554,2.394581
2000,400,4.657743,4.656495,3.018293
2000,500,4.694510,4.704185,3.463550
2000,600,4.737883,4.738238,3.976234
2000,700,4.760716,4.760815,4.288311
2000,800,4.766527,4.766055,4.613346
2000,900,4.775053,4.774723,4.731953
2000,1000,4.783280,4.777531,4.789646
2200,100,5.995965,5.994169,1.449604
2200,200,6.056407,6.056209,2.218009
2200,300,6.108666,6.105535,2.964738
2200,400,6.156488,6.154561,3.732560
2200,500,6.230410,6.236450,4.296981
2200,600,6.266725,6.265815,4.948849
2200,700,6.286903,6.286382,5.368273
2200,800,6.294685,6.265386,5.870119
2200,900,6.326134,6.325204,6.080209
2200,1000,6.333426,6.322827,6.259525
2200,1100,6.342487,6.333498,6.349874
2400,100,7.736524,7.735898,1.705171
2400,200,7.830168,7.829224,2.648876
2400,300,7.899159,7.898228,3.548137
2400,400,7.957125,7.957964,4.482665
2400,500,8.002192,7.984900,5.142648
2400,600,8.070236,8.060899,5.996336
2400,700,8.092098,8.073192,6.544540
2400,800,8.135616,8.121808,7.229996
2400,900,8.138484,8.127380,7.538543
2400,1000,8.147438,8.154952,7.938711
2400,1100,8.153073,8.114518,8.051720
2400,1200,8.155747,8.155639,8.171554
2600,100,9.752889,9.750480,2.017230
2600,200,9.854885,9.851034,3.110704
2600,300,9.951407,9.988961,4.190221
2600,400,10.077873,10.078756,5.324713
2600,500,10.133976,10.133701,6.131808
2600,600,10.189927,10.186606,7.160696
2600,700,10.170575,10.201211,7.838509
2600,800,10.256228,10.256213,8.712794
2600,900,10.239440,10.278222,9.122050
2600,1000,10.280324,10.288844,9.745449
2600,1100,10.302901,10.288874,9.932439
2600,1200,10.323346,10.320936,10.266253
2600,1300,10.309085,10.309061,10.327447
```

```
2800,100,12.119617,12.117360,2.303188
2800,200,12.250779,12.236298,3.589985
2800,300,12.307235,12.302878,4.817693
2800,400,12.438258,12.461511,6.243120
2800,500,12.539171,12.538035,7.122751
2800,600,12.542312,12.530206,8.340013
2800,700,12.652564,12.650723,9.167493
2800,800,12.640649,12.691485,10.293759
2800,900,12.693627,12.741725,10.809055
2800,1000,12.680285,12.725143,11.628925
2800,1100,12.707781,12.738423,11.974801
2800,1200,12.766840,12.763951,12.539982
2800,1300,12.788374,12.778614,12.720207
2800,1400,12.770373,12.795074,12.802720
3000,100,14.840447,14.831307,2.690211
3000,200,14.976413,14.959708,4.176103
3000,300,15.151452,15.173792,5.648661
3000,400,15.286011,15.286489,7.271673
3000,500,15.389996,15.373328,8.288515
3000,600,15.493214,15.462320,9.748327
3000,700,15.452624,15.525315,10.694515
3000,800,15.526809,15.487800,12.045151
3000,900,15.584945,15.575049,12.682044
3000,1000,15.622097,15.636239,13.738744
3000,1100,15.618825,15.633094,14.226476
3000,1200,15.657057,15.640628,15.026815
3000,1300,15.654789,15.644509,15.248089
3000,1400,15.666095,15.640708,15.625787
3000,1500,15.660501,15.637851,15.691399
3200,100,17.939330,17.930175,3.038807
3200,200,18.116577,18.080757,4.758674
3200,300,18.326212,18.321032,6.393522
3200,400,18.420184,18.374710,8.211370
3200,500,18.534395,18.504937,9.442391
3200,600,18.612912,18.610223,11.147423
3200,700,18.675522,18.668008,12.276197
3200,800,18.726301,18.724112,13.845009
3200,900,18.766330,18.764010,14.554328
3200,1000,18.796838,18.782811,15.909790
3200,1100,18.816668,18.800283,16.586373
3200,1200,18.868983,18.853505,17.660293
3200,1300,18.861740,18.840769,17.976502
3200,1400,18.804459,18.854824,18.547037
3200,1500,18.867645,18.882687,18.764833
3200,1600,18.850917,18.866136,18.883756
3400,100,21.431400,21.428372,3.499020
3400,200,21.677649,21.734402,5.491830
3400,300,21.919962,21.888041,7.299215
3400,400,22.060995,22.033717,9.470835
3400,500,22.167759,22.154973,10.884179
3400,600,22.255719,22.204260,12.720886
3400,700,22.289173,22.340964,13.996604
3400,800,22.399921,22.359309,15.857660
3400,900,22.406700,22.407413,16.630389
3400,1000,22.413745,22.341685,18.234535
3400,1100,22.462812,22.404296,19.000232
3400,1200,22.504238,22.454914,20.447482
3400,1300,22.532139,22.502377,20.887496
3400,1400,22.520403,22.483397,21.791360
3400,1500,22.510464,22.511410,22.068681
3400,1600,22.538436,22.550520,22.532693
3400,1700,22.545857,22.528021,22.569533
3600,100,25.331499,25.335258,3.826570
3600,200,25.564888,25.664284,6.077809
3600,300,25.844706,25.845378,8.127200
3600,400,26.015721,25.921526,10.532250
3600,500,26.122677,26.169874,12.294943
3600,600,26.231950,26.249063,14.184062
```

```
3600,700,26.254043,26.307129,15.653640
3600,800,26.322126,26.372034,17.818567
3600,900,26.434859,26.382756,18.848699
3600,1000,26.439103,26.473681,20.694668
3600,1100,26.436542,26.451802,21.648772
3600,1200,26.541936,26.557010,23.378509
3600,1300,26.554123,26.555286,23.866308
3600,1400,26.520574,26.569729,25.099096
3600,1500,26.547027,26.525381,25.488709
3600,1600,26.549132,26.560258,26.305981
3600,1700,26.560395,26.551900,26.418645
3600,1800,26.553967,26.580248,26.590845
3800,100,29.613703,29.688802,4.368600
3800,200,30.126346,30.094756,6.883081
3800,300,30.333500,30.375853,9.209319
3800,400,30.577710,30.547337,11.923601
3800,500,30.708293,30.676538,13.783653
3800,600,30.779900,30.755517,16.052991
3800,700,30.881849,30.869625,17.716743
3800,800,30.939434,30.921947,20.090265
3800,900,30.983062,31.014264,21.275684
3800,1000,31.026904,31.032104,23.410437
3800,1100,31.067666,30.980144,24.377797
3800,1200,30.975107,30.960889,26.467937
3800,1300,31.092892,31.137302,27.184245
3800,1400,31.130893,31.057277,28.639674
3800,1500,31.062522,31.052379,29.180538
3800,1600,31.128696,31.062754,30.277749
3800,1700,31.059876,31.051599,30.550157
3800,1800,31.084635,31.084237,31.096972
3800,1900,31.168661,31.169122,31.185568
4000,100,34.572700,34.571254,4.760100
4000,200,34.920208,34.866683,7.525317
4000,300,35.202262,35.164125,10.051593
4000,400,35.421777,35.371656,13.124616
4000,500,35.532269,35.527987,15.216321
4000,600,35.642596,35.607523,17.742157
4000,700,35.785476,35.796339,19.648370
4000,800,35.812263,35.838618,22.364377
4000,900,35.972400,35.965705,23.732007
4000,1000,35.957598,35.987159,26.161685
4000,1100,36.040607,36.009330,27.385015
4000,1200,36.015199,36.039758,29.797136
4000,1300,36.092776,36.055460,30.487368
4000,1400,36.060018,36.064797,32.396900
4000,1500,36.099571,36.090347,33.068309
4000,1600,36.121797,36.165950,34.623721
4000,1700,36.158602,36.175378,34.951939
4000,1800,36.090890,36.150517,35.833632
4000,1900,36.140159,36.109208,36.001906
4000,2000,36.145727,36.176741,36.180090
```

B.1.6 Profile Matrix - $n = 200 : 4000$ and $b = 100 : 2000$

```
PROFILE MATRIX TEST (Band= 1−>B−>1−>B−> ... etc)
 N,B,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SGBTRF,ASD1_SGBTRF
200,100,0.008200,0.005983,0.006742,0.006441,0.005768
400,100,0.048934,0.018866,0.021300,0.016667,0.014617
400,200,0.049420,0.029455,0.031856,0.036231,0.028441
600,100,0.144953,0.039546,0.043167,0.027303,0.023802
600,200,0.146298,0.054038,0.057841,0.061194,0.045443
600,300,0.147164,0.079568,0.083224,0.107933,0.079470
800,100,0.320142,0.065463,0.070969,0.037871,0.033087
800,200,0.321720,0.087719,0.093323,0.090851,0.068164
```

```
800,300,0.322655,0.111444,0.116753,0.160094,0.104999
800,400,0.324471,0.164054,0.169474,0.238142,0.163128
1000,100,0.596120,0.105843,0.112339,0.048231,0.042513
1000,200,0.599228,0.133532,0.140569,0.115947,0.086299
1000,300,0.601343,0.179691,0.187080,0.215511,0.154450
1000,400,0.602435,0.209383,0.216493,0.325673,0.196457
1000,500,0.605145,0.295367,0.302038,0.444567,0.293823
1200,100,1.000279,0.141986,0.152460,0.058868,0.052216
1200,200,1.004498,0.178951,0.189518,0.147134,0.109927
1200,300,1.007819,0.231145,0.240712,0.273762,0.187536
1200,400,1.012647,0.287267,0.296682,0.397926,0.251127
1200,500,1.012865,0.343581,0.353027,0.579556,0.336308
1200,600,1.016936,0.472417,0.481613,0.743602,0.480403
1400,100,1.568598,0.197968,0.208546,0.069124,0.061004
1400,200,1.575796,0.242319,0.253558,0.172371,0.126523
1400,300,1.580634,0.293687,0.304916,0.324864,0.212825
1400,400,1.587180,0.401131,0.412680,0.487961,0.334248
1400,500,1.591512,0.438964,0.450273,0.692884,0.394938
1400,600,1.594396,0.535522,0.546638,0.938374,0.530838
1400,700,1.601849,0.711974,0.722706,1.145511,0.721757
1600,100,2.313999,0.241903,0.258148,0.080326,0.070953
1600,200,2.321906,0.294793,0.310522,0.203824,0.150343
1600,300,2.330247,0.373209,0.388147,0.370586,0.255697
1600,400,2.336118,0.458213,0.472562,0.598713,0.374714
1600,500,2.348003,0.577304,0.591190,0.801944,0.497467
1600,600,2.349984,0.630353,0.644187,1.100477,0.598132
1600,700,2.356370,0.777239,0.790766,1.403235,0.783769
1600,800,2.370072,1.024307,1.037182,1.691446,1.045921
1800,100,3.266231,0.320213,0.334807,0.089656,0.079172
1800,200,3.278373,0.379184,0.395257,0.226674,0.166478
1800,300,3.289943,0.462185,0.478127,0.437561,0.293330
1800,400,3.298389,0.543618,0.559519,0.684703,0.406904
1800,500,3.313371,0.727367,0.743130,0.930574,0.603166
1800,600,3.329260,0.813708,0.829344,1.252684,0.714740
1800,700,3.332431,0.903659,0.919031,1.620391,0.863915
1800,800,3.340975,1.112843,1.128165,2.027192,1.111386
1800,900,3.363450,1.436157,1.450099,2.403781,1.461725
2000,100,4.438911,0.369858,0.391823,0.100702,0.089160
2000,200,4.452871,0.439116,0.460391,0.258580,0.190368
2000,300,4.466587,0.518641,0.538862,0.485714,0.315698
2000,400,4.485198,0.651394,0.670609,0.758447,0.462500
2000,500,4.495431,0.800360,0.819372,1.107448,0.657654
2000,600,4.521318,1.012120,1.030402,1.424298,0.877179
2000,700,4.529717,1.077169,1.095497,1.822142,0.985249
2000,800,4.528448,1.234452,1.251739,2.315956,1.201728
2000,900,4.546607,1.509520,1.526550,2.826462,1.537837
2000,1000,4.559825,1.924752,1.941767,3.304507,1.976614
2200,100,5.874296,0.472585,0.493247,0.111826,0.098894
2200,200,5.892467,0.542562,0.564266,0.282215,0.206312
2200,300,5.915322,0.646389,0.668005,0.528637,0.357164
2200,400,5.934737,0.808549,0.829819,0.844383,0.543918
2200,500,5.958358,0.907422,0.928646,1.241319,0.697249
2200,600,5.983412,1.194674,1.214940,1.616239,1.000747
2200,700,6.008889,1.366573,1.386220,2.020187,1.176997
2200,800,5.998534,1.450542,1.469435,2.575801,1.330010
2200,900,6.016977,1.673084,1.691295,3.222711,1.647635
2200,1000,6.032149,2.047347,2.066378,3.848467,2.062363
2200,1100,6.060192,2.544546,2.561912,4.358199,2.580019
2400,100,7.531442,0.519508,0.550756,0.121295,0.107590
2400,200,7.556404,0.603348,0.633833,0.313596,0.229413
2400,300,7.613480,0.715424,0.743628,0.594598,0.392727
2400,400,7.636418,0.880838,0.908192,0.955660,0.585138
2400,500,7.653936,1.007982,1.034164,1.358178,0.758145
2400,600,7.679119,1.269996,1.294746,1.857795,1.061290
2400,700,7.718749,1.609092,1.633480,2.248060,1.383952
2400,800,7.730648,1.742910,1.763857,2.828894,1.539278
2400,900,7.693052,1.826508,1.849068,3.513508,1.770476
2400,1000,7.757699,2.169025,2.190480,4.303045,2.165299
```

```
2400,1100,7.775698,2.640163,2.660893,5.057736,2.698265
2400,1200,7.784942,3.213556,3.230607,5.612056,3.286081
2600,100,9.564814,0.649029,0.674208,0.131946,0.116757
2600,200,9.593344,0.739665,0.767704,0.336976,0.246599
2600,300,9.623467,0.849335,0.877549,0.643547,0.418169
2600,400,9.658986,1.018806,1.046169,1.046912,0.619813
2600,500,9.690811,1.230612,1.257632,1.459869,0.858834
2600,600,9.717241,1.423489,1.449801,2.052921,1.114442
2600,700,9.760833,1.849901,1.874274,2.538439,1.532230
2600,800,9.779450,2.134389,2.164265,3.109683,1.806838
2600,900,9.836538,2.185117,2.208301,3.852348,1.976021
2600,1000,9.827471,2.418746,2.441537,4.720740,2.306740
2600,1100,9.831211,2.834955,2.860331,5.592650,2.805660
2600,1200,9.842612,3.409742,3.433844,6.504061,3.429877
2600,1300,9.914958,4.099207,4.120360,7.209246,4.170517
2800,100,11.894084,0.696593,0.734623,0.142011,0.126177
2800,200,11.912271,0.799855,0.836094,0.367650,0.268420
2800,300,11.946315,0.943162,0.977601,0.691637,0.464328
2800,400,12.017980,1.127053,1.160803,1.116020,0.672687
2800,500,12.042453,1.374458,1.405751,1.596379,0.966381
2800,600,12.063268,1.536250,1.567221,2.212234,1.178859
2800,700,12.120908,1.943339,1.971100,2.883751,1.609761
2800,800,12.175123,2.444661,2.472813,3.444238,2.080126
2800,900,12.221593,2.564243,2.584400,4.171787,2.278323
2800,1000,12.197550,2.704775,2.726748,5.117026,2.527847
2800,1100,12.199916,3.019938,3.038845,6.074355,2.948976
2800,1200,12.237594,3.549127,3.568078,7.096332,3.535324
2800,1300,12.248003,4.215082,4.259746,8.210213,4.336334
2800,1400,12.296582,5.045446,5.052284,9.004443,5.205802
3000,100,14.569837,0.854549,0.886135,0.152318,0.135317
3000,200,14.612738,0.968902,1.003435,0.393215,0.286819
3000,300,14.640903,1.118907,1.154109,0.757178,0.499341
3000,400,14.661421,1.345004,1.378380,1.198488,0.750047
3000,500,14.656163,1.550808,1.583901,1.773577,1.020146
3000,600,14.784705,1.828080,1.862811,2.366622,1.301090
3000,700,14.826352,2.130126,2.161808,3.136268,1.666317
3000,800,14.891194,2.705021,2.739093,3.798809,2.213353
3000,900,14.936734,3.027929,3.071059,4.535191,2.600194
3000,1000,15.040348,3.238913,3.265494,5.507398,2.838948
3000,1100,14.988503,3.440523,3.459626,6.573874,3.182564
3000,1200,14.981862,3.867745,3.885192,7.739360,3.712209
3000,1300,14.999934,4.443557,4.447396,8.841205,4.431276
3000,1400,15.055442,5.249729,5.278302,10.154428,5.325872
3000,1500,15.102927,6.211280,6.221620,10.992561,6.322761
3200,100,17.640095,0.916595,0.964877,0.162357,0.144191
3200,200,17.664475,1.046388,1.097698,0.421770,0.307169
3200,300,17.782550,1.205350,1.250996,0.804503,0.521793
3200,400,17.809545,1.447556,1.491235,1.317094,0.793701
3200,500,17.847584,1.654862,1.692670,1.892742,1.054591
3200,600,17.834681,2.056883,2.099106,2.523720,1.458825
3200,700,17.938282,2.269491,2.309817,3.340263,1.735380
3200,800,17.909700,2.804803,2.841673,4.243782,2.291485
3200,900,18.075342,3.354216,3.420406,4.930450,2.901651
3200,1000,18.141989,3.679497,3.724186,5.903423,3.229603
3200,1100,18.170378,3.884752,3.896663,7.070183,3.522975
3200,1200,18.155877,4.193311,4.221155,8.322258,3.944997
3200,1300,18.130040,4.668639,4.720085,9.650749,4.617368
3200,1400,18.164030,5.414671,5.477989,11.073276,5.494165
3200,1500,18.148894,6.331342,6.368270,12.242224,6.473538
3200,1600,18.242791,7.466842,7.476557,13.189529,7.550323
3400,100,21.085008,1.102659,1.139677,0.172656,0.153908
3400,200,21.184022,1.238131,1.278727,0.446319,0.325632
3400,300,21.244323,1.422090,1.462844,0.850132,0.566046
3400,400,21.245806,1.637150,1.676492,1.400457,0.824469
3400,500,21.331881,1.903264,1.942657,2.012305,1.119741
3400,600,21.418390,2.357447,2.393561,2.742856,1.593172
3400,700,21.477528,2.615646,2.649861,3.530651,1.853349
3400,800,21.474453,3.036053,3.066531,4.573736,2.364067
```

```
3400,900,21.537501,3.656578,3.683583,5.424599,3.044266
3400,1000,21.645707,4.247195,4.246271,6.339559,3.619567
3400,1100,21.719168,4.535799,4.575725,7.493521,3.949965
3400,1200,21.618751,4.759150,4.791168,8.911383,4.279120
3400,1300,21.617003,5.109410,5.166085,10.297092,4.834239
3400,1400,21.677341,5.770717,5.829774,11.883002,5.663800
3400,1500,21.620037,6.620011,6.642421,13.304069,6.605993
3400,1600,21.781687,7.654059,7.681620,14.616693,7.744095
3400,1700,21.802484,8.846053,8.917670,15.900092,9.074938
3600,100,24.868174,1.137487,1.192430,0.183444,0.162778
3600,200,24.891666,1.291928,1.345911,0.476662,0.346769
3600,300,25.079084,1.489105,1.538185,0.916526,0.601135
3600,400,25.111177,1.757356,1.804344,1.479689,0.884684
3600,500,25.185339,2.085528,2.127884,2.124836,1.224335
3600,600,25.161985,2.445146,2.489004,2.990540,1.655899
3600,700,25.317446,2.888894,2.926328,3.730219,2.041579
3600,800,25.326146,3.170905,3.205386,4.870417,2.453141
3600,900,25.419607,3.744817,3.796805,5.975632,3.155784
3600,1000,25.508065,4.590226,4.616338,6.913412,3.972069
3600,1100,25.573376,5.037524,5.088846,8.020115,4.426580
3600,1200,25.578905,5.315846,5.359857,9.440732,4.745880
3600,1300,25.595927,5.583434,5.608317,11.049924,5.191765
3600,1400,25.591772,6.094660,6.138936,12.692287,5.905276
3600,1500,25.616976,6.820897,6.881043,14.259585,6.840819
3600,1600,25.610085,7.816562,7.830189,15.891809,7.906679
3600,1700,25.563638,8.972963,8.971633,17.579212,9.248558
3600,1800,25.605334,10.363877,10.347480,18.881918,10.694982
3800,100,29.278701,1.364776,1.407003,0.192518,0.170554
3800,200,29.336161,1.529068,1.576613,0.496221,0.362559
3800,300,29.339518,1.719363,1.764871,0.966792,0.625871
3800,400,29.296731,2.066933,2.119250,1.560427,0.961888
3800,500,29.608264,2.417800,2.463299,2.254553,1.328684
3800,600,29.651530,2.703824,2.747634,3.170698,1.698699
3800,700,29.546696,3.325941,3.377764,3.954879,2.244268
3800,800,29.814778,3.582709,3.600669,5.113649,2.577338
3800,900,29.881870,4.057672,4.098527,6.441870,3.245684
3800,1000,29.871508,4.929983,4.978670,7.553715,4.131808
3800,1100,30.013635,5.691168,5.753486,8.627036,4.918174
3800,1200,30.101973,6.083175,6.164302,9.948278,5.281307
3800,1300,30.019115,6.382575,6.412983,11.698725,5.649899
3800,1400,30.076057,6.781383,6.808838,13.399225,6.276432
3800,1500,30.121587,7.361132,7.381804,15.271450,7.110261
3800,1600,29.978816,8.231033,8.266198,16.766074,8.053247
3800,1700,30.131291,9.239319,9.337666,18.863956,9.364261
3800,1800,30.224143,10.661970,10.703777,20.734686,10.876128
3800,1900,30.221596,12.218211,12.285268,22.109278,12.450424
4000,100,34.027186,1.413165,1.480265,0.205374,0.182523
4000,200,34.114005,1.592100,1.656601,0.530160,0.385449
4000,300,34.195326,1.820906,1.882395,1.011680,0.668390
4000,400,34.122053,2.122714,2.181813,1.673798,1.001959
4000,500,34.371483,2.504276,2.560443,2.437697,1.384356
4000,600,34.326665,2.795743,2.882815,3.325569,1.762066
4000,700,34.540636,3.513052,3.543941,4.245060,2.388493
4000,800,34.666674,3.942172,3.993251,5.402354,2.801687
4000,900,34.439006,4.161359,4.216556,6.740087,3.318183
4000,1000,34.625172,5.044308,5.082580,8.272203,4.247074
4000,1100,34.817420,6.088024,6.132267,9.319307,5.249901
4000,1200,34.923117,6.722470,6.786260,10.703668,5.880204
4000,1300,35.052927,7.101568,7.145337,12.327087,6.237491
4000,1400,35.004070,7.384862,7.435859,14.211681,6.702590
4000,1500,34.857459,7.892778,7.899795,16.260223,7.494266
4000,1600,34.831351,8.599617,8.624795,17.950698,8.378012
4000,1700,34.902224,9.590187,9.639407,20.169865,9.692812
4000,1800,35.005337,10.916307,10.931019,22.178513,11.084315
4000,1900,34.928581,12.375775,12.358321,24.228505,12.766918
4000,2000,35.160921,13.956328,14.041879,25.492755,14.308674
```

B.1.7 Bulge Matrix - $n = 200 : 4000$ and $b = 100 : 2000$

```
BULGE MATRIX TEST (B−>B−>FULL(BP1:BP2)−>B)
 N,B,BLARGE,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SGBTRF,ASD1_SGBTRF
600,100,150,0.145914,0.055678,0.059646,0.045039,0.041743
800,100,200,0.321596,0.089204,0.094806,0.086783,0.066503
1000,100,250,0.594984,0.144852,0.151806,0.154770,0.103507
1000,200,250,0.598403,0.202646,0.209824,0.171306,0.160234
1200,100,300,1.003293,0.195433,0.205208,0.247974,0.146793
1200,200,300,1.006193,0.267905,0.277759,0.266959,0.215275
1400,100,350,1.572604,0.272530,0.283548,0.373945,0.198114
1400,200,350,1.578370,0.365523,0.377065,0.400290,0.281972
1400,300,350,1.583610,0.493165,0.504049,0.429737,0.405939
1600,100,400,2.322867,0.333356,0.348190,0.531623,0.248758
1600,200,400,2.329468,0.440216,0.454478,0.559406,0.344406
1600,300,400,2.336635,0.597076,0.610602,0.598879,0.495437
1800,100,450,3.280464,0.431592,0.447623,0.749445,0.335440
1800,200,450,3.289289,0.552740,0.568393,0.779466,0.446548
1800,300,450,3.300395,0.723305,0.739232,0.823208,0.606782
1800,400,450,3.314070,0.955538,0.969789,0.881440,0.844049
2000,100,500,4.460498,0.527056,0.547940,0.984620,0.403787
2000,200,500,4.469501,0.668386,0.688056,1.020251,0.530986
2000,300,500,4.481029,0.855070,0.873807,1.062064,0.706397
2000,400,500,4.494949,1.120071,1.137760,1.121521,0.964029
2200,100,550,5.904607,0.650777,0.672954,1.298091,0.508188
2200,200,550,5.916479,0.811321,0.832591,1.347088,0.652999
2200,300,550,5.930437,1.019700,1.040064,1.397632,0.855721
2200,400,550,5.947104,1.315654,1.335058,1.476275,1.155846
2200,500,550,5.972261,1.660255,1.680240,1.557667,1.498807
2400,100,600,7.610023,0.765381,0.794388,1.623615,0.584351
2400,200,600,7.622899,0.944599,0.971561,1.691218,0.749832
2400,300,600,7.639312,1.188566,1.213331,1.760873,0.989262
2400,400,600,7.658168,1.500923,1.525026,1.826878,1.294162
2400,500,600,7.683724,1.902378,1.924830,1.927515,1.695443
2600,100,650,9.618549,0.936590,0.965467,2.037395,0.721515
2600,200,650,9.628424,1.129460,1.156673,2.091751,0.891301
2600,300,650,9.650345,1.406012,1.432182,2.180639,1.153861
2600,400,650,9.656094,1.741837,1.767056,2.264117,1.481550
2600,500,650,9.685741,2.175071,2.199137,2.345750,1.892236
2600,600,650,9.708986,2.714213,2.737330,2.494138,2.414598
2800,100,700,11.954343,1.069505,1.106004,2.470247,0.829590
2800,200,700,12.004837,1.292907,1.325801,2.561266,1.024225
2800,300,700,12.029372,1.587660,1.618154,2.623448,1.297593
2800,400,700,12.059810,1.959625,1.988786,2.722051,1.660411
2800,500,700,12.084465,2.462270,2.488929,2.873660,2.156500
2800,600,700,12.113939,2.958416,2.984508,2.924637,2.610139
3000,100,750,14.664431,1.288003,1.322555,3.052435,0.983156
3000,200,750,14.704697,1.517660,1.550851,3.125568,1.187178
3000,300,750,14.731748,1.861584,1.892687,3.239537,1.508064
3000,400,750,14.746849,2.298103,2.326928,3.371210,1.917180
3000,500,750,14.722197,2.727410,2.754979,3.421721,2.347078
3000,600,750,14.737697,3.378211,3.406294,3.590750,2.943238
3000,700,750,14.857062,4.198529,4.224683,3.794992,3.681252
3200,100,800,17.728924,1.411238,1.456004,3.632925,1.059723
3200,200,800,17.761742,1.674736,1.715792,3.715890,1.298372
3200,300,800,17.800350,2.052343,2.091433,3.850116,1.648059
3200,400,800,17.805644,2.511227,2.545006,3.960523,2.092156
3200,500,800,17.775645,3.041864,3.073874,4.077672,2.602467
3200,600,800,17.972796,3.747991,3.781231,4.247708,3.243021
3200,700,800,18.013690,4.591716,4.622404,4.435047,3.992589
3400,100,850,21.221418,1.727579,1.769161,4.321213,1.272352
3400,200,850,21.263646,2.001867,2.039625,4.447125,1.516347
3400,300,850,21.233208,2.394940,2.432542,4.566508,1.896104
3400,400,850,21.215553,2.851766,2.889760,4.685571,2.312174
3400,500,850,21.455079,3.497522,3.532571,4.911810,2.937040
3400,600,850,21.471476,4.190557,4.224535,4.981672,3.558852
3400,700,850,21.543138,5.031637,5.064249,5.133330,4.281377
```

```
3400,800,850,21.564909,5.973539,5.987080,5.345933,5.202698
3600,100,900,25.135115,1.830753,1.876316,5.061416,1.440419
3600,200,900,25.141035,2.137663,2.183738,5.192006,1.704347
3600,300,900,25.144920,2.530743,2.568367,5.305402,2.061927
3600,400,900,25.209613,3.107716,3.145915,5.484458,2.581458
3600,500,900,25.316476,3.749126,3.790353,5.673247,3.184161
3600,600,900,25.203019,4.469124,4.539694,5.797802,3.861079
3600,700,900,25.431910,5.410500,5.430645,5.989895,4.676106
3600,800,900,25.480130,6.492397,6.528722,6.269319,5.684528
3800,100,950,29.274715,2.242540,2.296336,6.007499,1.671183
3800,200,950,29.559875,2.589295,2.634437,6.155368,1.960979
3800,300,950,29.628324,3.055622,3.101192,6.325205,2.392246
3800,400,950,29.543584,3.588118,3.626073,6.436318,2.883013
3800,500,950,29.811635,4.290885,4.324878,6.739356,3.592061
3800,600,950,29.862247,5.175254,5.217688,6.822827,4.262154
3800,700,950,29.733709,6.083016,6.174035,7.063294,5.168577
3800,800,950,30.008608,7.250749,7.220488,7.347414,6.258658
3800,900,950,30.082358,8.330845,8.326663,7.546597,7.387683
4000,100,1000,34.190808,2.379310,2.437325,6.962471,1.820706
4000,200,1000,34.360544,2.720993,2.779179,7.096316,2.115126
4000,300,1000,34.248460,3.232719,3.299474,7.260169,2.585929
4000,400,1000,34.505589,3.895164,3.936732,7.426701,3.163434
4000,500,1000,34.479362,4.606800,4.650761,7.723989,3.862208
4000,600,1000,34.590151,5.434015,5.509672,7.931150,4.628545
4000,700,1000,34.723463,6.530054,6.520259,8.105308,5.551721
4000,800,1000,34.855871,7.679101,7.700334,8.346134,6.616206
4000,900,1000,34.909390,8.793845,8.782225,8.624846,7.919325
```

B.2 Diagonally Dominant Positive Definite

B.2.1 Full Positive Definite Matrix - $n = 200 : 4000$

```
POSDEF FULL MATRIX TEST
 N,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SPOTRF,ASD1_SPOTRF,ASD2_SPOTRF
100,0.001452,0.001450,0.001792,0.000823,0.000813,0.001089
200,0.007276,0.007265,0.008009,0.004077,0.004070,0.004623
300,0.020739,0.020686,0.022492,0.011355,0.011352,0.012769
400,0.044858,0.044797,0.047355,0.024201,0.024193,0.026214
500,0.084098,0.084127,0.087092,0.044547,0.044527,0.047009
600,0.135288,0.135248,0.139222,0.073918,0.073923,0.077028
700,0.209848,0.209818,0.214439,0.114215,0.114008,0.117795
800,0.301886,0.301810,0.307392,0.165491,0.165430,0.169953
900,0.427174,0.427553,0.434139,0.231964,0.231943,0.237096
1000,0.562869,0.562821,0.570731,0.311468,0.311447,0.317399
1100,0.749113,0.748957,0.757759,0.410642,0.410500,0.417244
1200,0.954858,0.954590,0.964552,0.525817,0.526054,0.533342
1300,1.231391,1.231244,1.242031,0.663586,0.663213,0.671901
1400,1.502760,1.502354,1.514280,0.819789,0.819947,0.829287
1500,1.883423,1.882612,1.895956,1.002237,1.002554,1.012203
1600,2.226815,2.227816,2.242810,1.208076,1.208986,1.219704
1700,2.761604,2.761428,2.776705,1.442688,1.444205,1.453062
1800,3.156467,3.155966,3.173365,1.699797,1.700539,1.713707
1900,3.837070,3.836975,3.855053,1.991786,1.992625,2.004981
2000,4.312086,4.311607,4.330554,2.311626,2.312930,2.326394
2100,5.201429,5.200338,5.221237,2.669052,2.669563,2.683964
2200,5.702532,5.703667,5.724184,3.056964,3.059332,3.072573
2300,6.797281,6.795484,6.817384,3.484410,3.485384,3.501222
2400,7.381396,7.380746,7.405970,3.953544,3.954916,3.972448
2500,8.733166,8.722462,8.740012,4.455584,4.456855,4.474252
2600,9.329315,9.329092,9.356339,4.998400,5.000597,5.018940
2700,10.924049,10.921450,10.951230,5.599070,5.600228,5.619075
2800,11.652220,11.652534,11.681621,6.227329,6.229177,6.247507
```

```
2900,13.587684,13.588109,13.616713,6.908256,6.908875,6.929625
3000,14.258839,14.251593,14.275519,7.638977,7.642764,7.661992
3100,16.458832,16.472790,16.476482,8.412750,8.413438,8.434249
3200,17.207947,17.202943,17.236409,9.250323,9.254389,9.275707
3300,19.785617,19.900286,20.020092,10.132280,10.134309,10.157347
3400,20.704110,20.701165,20.727791,11.073892,11.078307,11.099992
3500,23.599433,23.589108,23.627998,12.060085,12.062587,12.085885
3600,24.506681,24.504277,24.543322,13.110300,13.115584,13.135806
3700,28.011074,28.031085,28.110522,14.232830,14.233680,14.257918
3800,28.684685,28.677136,28.705600,15.399914,15.406507,15.427769
3900,32.467709,32.509491,32.627837,16.623588,16.626023,16.649907
4000,33.524126,33.524853,33.568031,17.971476,17.976417,18.003913
```

B.2.2 Full Positive Definite Matrix - $n = 1 : 200$

```
POSDEF FULL MATRIX TEST
 N,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SPOTRF,ASD1_SPOTRF,ASD2_SPOTRF
1,0.000004,0.000004,0.000005,0.000004,0.000003,0.000005
2,0.000005,0.000005,0.000006,0.000004,0.000004,0.000005
3,0.000006,0.000006,0.000008,0.000006,0.000006,0.000007
4,0.000008,0.000008,0.000009,0.000007,0.000007,0.000009
5,0.000010,0.000010,0.000011,0.000009,0.000009,0.000010
6,0.000012,0.000012,0.000013,0.000011,0.000011,0.000012
7,0.000015,0.000014,0.000016,0.000013,0.000013,0.000014
8,0.000017,0.000017,0.000019,0.000014,0.000015,0.000016
9,0.000021,0.000020,0.000022,0.000017,0.000017,0.000018
10,0.000025,0.000024,0.000026,0.000019,0.000019,0.000020
11,0.000030,0.000029,0.000031,0.000021,0.000021,0.000022
12,0.000030,0.000030,0.000032,0.000023,0.000023,0.000024
13,0.000037,0.000037,0.000039,0.000026,0.000026,0.000027
14,0.000040,0.000040,0.000041,0.000028,0.000028,0.000029
15,0.000047,0.000047,0.000049,0.000031,0.000031,0.000032
16,0.000047,0.000047,0.000048,0.000033,0.000033,0.000034
17,0.000057,0.000056,0.000058,0.000038,0.000038,0.000039
18,0.000061,0.000060,0.000061,0.000040,0.000040,0.000041
19,0.000098,0.000071,0.000073,0.000045,0.000044,0.000045
20,0.000066,0.000065,0.000067,0.000044,0.000044,0.000045
21,0.000086,0.000086,0.000087,0.000052,0.000051,0.000052
22,0.000084,0.000083,0.000085,0.000054,0.000053,0.000055
23,0.000101,0.000100,0.000102,0.000060,0.000059,0.000060
24,0.000090,0.000089,0.000090,0.000060,0.000060,0.000061
25,0.000119,0.000117,0.000119,0.000070,0.000068,0.000070
26,0.000112,0.000112,0.000130,0.000071,0.000069,0.000071
27,0.000136,0.000135,0.000136,0.000078,0.000077,0.000079
28,0.000116,0.000116,0.000117,0.000077,0.000077,0.000078
29,0.000156,0.000155,0.000156,0.000089,0.000088,0.000089
30,0.000143,0.000142,0.000144,0.000088,0.000087,0.000088
31,0.000177,0.000175,0.000202,0.000099,0.000097,0.000098
32,0.000149,0.000148,0.000150,0.000095,0.000094,0.000095
33,0.000210,0.000209,0.000210,0.000112,0.000109,0.000111
34,0.000183,0.000181,0.000183,0.000108,0.000107,0.000108
35,0.000276,0.000256,0.000258,0.000123,0.000121,0.000123
36,0.000187,0.000185,0.000187,0.000115,0.000114,0.000116
37,0.000284,0.000282,0.000285,0.000136,0.000134,0.000135
38,0.000225,0.000226,0.000225,0.000135,0.000133,0.000134
39,0.000311,0.000309,0.000310,0.000147,0.000145,0.000147
40,0.000229,0.000227,0.000230,0.000138,0.000137,0.000138
41,0.000362,0.000342,0.000344,0.000166,0.000163,0.000165
42,0.000275,0.000271,0.000272,0.000160,0.000158,0.000159
43,0.000376,0.000405,0.000376,0.000185,0.000182,0.000183
44,0.000276,0.000274,0.000275,0.000165,0.000164,0.000166
45,0.000413,0.000410,0.000431,0.000199,0.000196,0.000197
46,0.000333,0.000330,0.000331,0.000190,0.000186,0.000188
47,0.000451,0.000448,0.000473,0.000218,0.000215,0.000217
```

```
48,0.000330,0.000328,0.000330,0.000193,0.000191,0.000193
49,0.000491,0.000506,0.000490,0.000234,0.000230,0.000231
50,0.000394,0.000392,0.000394,0.000222,0.000219,0.000220
51,0.000532,0.000528,0.000529,0.000254,0.000251,0.000252
52,0.000388,0.000385,0.000388,0.000246,0.000223,0.000224
53,0.000571,0.000566,0.000568,0.000274,0.000270,0.000270
54,0.000479,0.000460,0.000460,0.000262,0.000259,0.000260
55,0.000619,0.000616,0.000642,0.000294,0.000290,0.000292
56,0.000453,0.000451,0.000454,0.000258,0.000256,0.000257
57,0.000663,0.000659,0.000659,0.000317,0.000313,0.000314
58,0.000558,0.000535,0.000537,0.000295,0.000290,0.000291
59,0.000721,0.000735,0.000720,0.000341,0.000336,0.000337
60,0.000527,0.000521,0.000525,0.000309,0.000294,0.000291
61,0.000764,0.000760,0.000762,0.000381,0.000359,0.000359
62,0.000626,0.000621,0.000623,0.000336,0.000332,0.000351
63,0.000835,0.000827,0.000827,0.000394,0.000406,0.000391
64,0.000606,0.000603,0.000603,0.000338,0.000335,0.000336
65,0.000937,0.000933,0.001158,0.000465,0.000451,0.000640
66,0.000769,0.000769,0.001014,0.000430,0.000422,0.000610
67,0.000990,0.001010,0.001221,0.000493,0.000484,0.000677
68,0.000729,0.000725,0.000968,0.000411,0.000406,0.000601
69,0.001005,0.001001,0.001240,0.000521,0.000494,0.000691
70,0.000845,0.000853,0.001089,0.000475,0.000468,0.000668
71,0.001094,0.001072,0.001314,0.000541,0.000537,0.000736
72,0.000806,0.000805,0.001082,0.000460,0.000453,0.000659
73,0.001096,0.001115,0.001347,0.000553,0.000543,0.000757
74,0.000939,0.000930,0.001191,0.000546,0.000519,0.000731
75,0.001181,0.001196,0.001436,0.000596,0.000589,0.000806
76,0.000892,0.000889,0.001178,0.000515,0.000509,0.000727
77,0.001196,0.001213,0.001465,0.000603,0.000591,0.000819
78,0.001020,0.001017,0.001303,0.000580,0.000570,0.000794
79,0.001299,0.001283,0.001560,0.000657,0.000645,0.000868
80,0.000975,0.000986,0.001253,0.000557,0.000552,0.000786
81,0.001306,0.001302,0.001598,0.000655,0.000644,0.000878
82,0.001121,0.001112,0.001400,0.000638,0.000623,0.000860
83,0.001420,0.001401,0.001682,0.000711,0.000698,0.000937
84,0.001067,0.001079,0.001355,0.000618,0.000610,0.000855
85,0.001414,0.001430,0.001705,0.000715,0.000704,0.000956
86,0.001220,0.001217,0.001534,0.000693,0.000684,0.000931
87,0.001519,0.001520,0.001819,0.000772,0.000762,0.001015
88,0.001168,0.001169,0.001489,0.000672,0.000666,0.000918
89,0.001541,0.001541,0.001848,0.000779,0.000765,0.001028
90,0.001331,0.001331,0.001653,0.000753,0.000743,0.001009
91,0.001649,0.001663,0.001954,0.000834,0.000828,0.001088
92,0.001278,0.001285,0.001591,0.000748,0.000748,0.001004
93,0.001684,0.001670,0.001988,0.000855,0.000836,0.001103
94,0.001467,0.001443,0.001770,0.000828,0.000812,0.001082
95,0.001785,0.001796,0.002111,0.000905,0.000898,0.001169
96,0.001393,0.001415,0.001716,0.000803,0.000800,0.001070
97,0.001820,0.001796,0.002135,0.000919,0.000905,0.001187
98,0.001583,0.001580,0.001913,0.000891,0.000883,0.001162
99,0.001930,0.001910,0.002254,0.000974,0.000962,0.001250
100,0.001530,0.001552,0.001867,0.000869,0.000866,0.001149
101,0.001985,0.001961,0.002319,0.001001,0.000983,0.001277
102,0.001716,0.001727,0.002068,0.000969,0.000954,0.001247
103,0.002094,0.002075,0.002445,0.001051,0.001043,0.001332
104,0.001693,0.001669,0.002030,0.000937,0.000930,0.001232
105,0.002139,0.002156,0.002507,0.001080,0.001057,0.001360
106,0.001876,0.001876,0.002233,0.001051,0.001030,0.001334
107,0.002251,0.002260,0.002622,0.001135,0.001115,0.001421
108,0.001815,0.001831,0.002182,0.001030,0.001024,0.001328
109,0.002325,0.002323,0.002692,0.001163,0.001141,0.001451
110,0.002032,0.002033,0.002408,0.001126,0.001108,0.001423
111,0.002430,0.002432,0.002816,0.001225,0.001207,0.001527
112,0.001966,0.001985,0.002346,0.001090,0.001082,0.001406
113,0.002521,0.002497,0.002894,0.001244,0.001227,0.001546
114,0.002225,0.002203,0.002592,0.001201,0.001187,0.001513
115,0.002620,0.002620,0.003018,0.001306,0.001291,0.001623
```

```
116,0.002143,0.002126,0.002529,0.001177,0.001174,0.001500
117,0.002715,0.002692,0.003099,0.001337,0.001320,0.001659
118,0.002378,0.002372,0.002778,0.001292,0.001284,0.001616
119,0.002811,0.002810,0.003229,0.001414,0.001400,0.001735
120,0.002312,0.002298,0.002694,0.001262,0.001249,0.001596
121,0.002900,0.002885,0.003317,0.001443,0.001418,0.001760
122,0.002564,0.002564,0.002975,0.001380,0.001369,0.001714
123,0.003032,0.003016,0.003444,0.001501,0.001491,0.001841
124,0.002474,0.002482,0.002898,0.001369,0.001360,0.001718
125,0.003127,0.003109,0.003544,0.001534,0.001520,0.001877
126,0.002764,0.002758,0.003183,0.001487,0.001475,0.001833
127,0.003252,0.003254,0.003696,0.001617,0.001599,0.001959
128,0.002667,0.002660,0.003092,0.001491,0.001482,0.001841
129,0.003396,0.003398,0.003861,0.001714,0.001706,0.002077
130,0.003001,0.003003,0.003473,0.001660,0.001654,0.002028
131,0.003523,0.003516,0.004001,0.001802,0.001791,0.002167
132,0.002888,0.002894,0.003368,0.001626,0.001621,0.001998
133,0.003575,0.003575,0.004072,0.001819,0.001809,0.002192
134,0.003198,0.003193,0.003676,0.001767,0.001754,0.002149
135,0.003707,0.003712,0.004217,0.001916,0.001914,0.002297
136,0.003083,0.003081,0.003583,0.001733,0.001720,0.002121
137,0.003800,0.003783,0.004302,0.001932,0.001913,0.002317
138,0.003430,0.003413,0.003906,0.001880,0.001863,0.002270
139,0.003949,0.003934,0.004445,0.002031,0.002023,0.002419
140,0.003357,0.003343,0.003852,0.001847,0.001835,0.002253
141,0.004172,0.004168,0.004705,0.002044,0.002033,0.002442
142,0.003812,0.003793,0.004319,0.001996,0.001982,0.002403
143,0.004460,0.004482,0.005006,0.002154,0.002141,0.002560
144,0.003486,0.003495,0.004017,0.001962,0.001949,0.002381
145,0.004228,0.004228,0.004772,0.002163,0.002145,0.002572
146,0.003853,0.003836,0.004373,0.002108,0.002099,0.002522
147,0.004411,0.004406,0.004965,0.002267,0.002259,0.002683
148,0.003721,0.003723,0.004260,0.002082,0.002074,0.002507
149,0.004493,0.004469,0.005032,0.002292,0.002272,0.002712
150,0.004085,0.004076,0.004627,0.002235,0.002225,0.002664
151,0.004672,0.004643,0.005215,0.002403,0.002388,0.002831
152,0.003953,0.003948,0.004497,0.002208,0.002196,0.002645
153,0.004731,0.004726,0.005307,0.002421,0.002404,0.002849
154,0.004320,0.004314,0.004880,0.002366,0.002353,0.002804
155,0.004947,0.004965,0.005506,0.002545,0.002520,0.002969
156,0.004255,0.004255,0.004820,0.002339,0.002331,0.002783
157,0.005185,0.005179,0.005774,0.002566,0.002549,0.003004
158,0.004788,0.004788,0.005355,0.002512,0.002496,0.002967
159,0.005545,0.005554,0.006136,0.002686,0.002676,0.003130
160,0.004443,0.004434,0.005025,0.002478,0.002474,0.002942
161,0.005259,0.005256,0.005870,0.002717,0.002692,0.003168
162,0.004833,0.004827,0.005423,0.002645,0.002637,0.003112
163,0.005468,0.005469,0.006073,0.002823,0.002810,0.003277
164,0.004705,0.004707,0.005284,0.002627,0.002610,0.003093
165,0.005549,0.005552,0.006166,0.002861,0.002845,0.003326
166,0.005111,0.005101,0.005721,0.002795,0.002791,0.003275
167,0.005761,0.005764,0.006386,0.002976,0.002964,0.003456
168,0.004974,0.004973,0.005584,0.002760,0.002755,0.003252
169,0.005845,0.005847,0.006478,0.003005,0.002989,0.003482
170,0.005393,0.005390,0.006004,0.002945,0.002933,0.003430
171,0.006108,0.006093,0.006736,0.003135,0.003120,0.003620
172,0.005366,0.005356,0.005979,0.002922,0.002912,0.003408
173,0.006432,0.006427,0.007068,0.003164,0.003147,0.003653
174,0.005955,0.005952,0.006579,0.003101,0.003090,0.003605
175,0.006829,0.006825,0.007478,0.003298,0.003279,0.003795
176,0.005569,0.005559,0.006192,0.003063,0.003049,0.003573
177,0.006513,0.006513,0.007177,0.003336,0.003319,0.003840
178,0.006023,0.006024,0.006668,0.003277,0.003266,0.003787
179,0.006759,0.006750,0.007419,0.003464,0.003441,0.003969
180,0.005892,0.005874,0.006517,0.003233,0.003218,0.003747
181,0.006863,0.006857,0.007545,0.003506,0.003486,0.004024
182,0.006313,0.006307,0.006970,0.003437,0.003431,0.003957
183,0.007097,0.007099,0.007788,0.003640,0.003621,0.004163
```

```
184,0.006192,0.006183,0.006849,0.003385,0.003376,0.003917
185,0.007219,0.007188,0.007907,0.003682,0.003657,0.004198
186,0.006639,0.006620,0.007303,0.003609,0.003605,0.004135
187,0.007455,0.007450,0.008157,0.003825,0.003800,0.004348
188,0.006626,0.006606,0.007278,0.003559,0.003544,0.004101
189,0.007849,0.007836,0.008553,0.003883,0.003846,0.004408
190,0.007296,0.007267,0.007959,0.003786,0.003771,0.004338
191,0.008321,0.008308,0.009021,0.004070,0.004043,0.004613
192,0.006857,0.006833,0.007531,0.003753,0.003743,0.004314
193,0.008007,0.007994,0.008743,0.004180,0.004160,0.004730
194,0.007399,0.007383,0.008124,0.004097,0.004081,0.004668
195,0.008239,0.008225,0.008989,0.004346,0.004325,0.004904
196,0.007236,0.007231,0.007957,0.004070,0.004059,0.004648
197,0.008356,0.008357,0.009124,0.004373,0.004355,0.004947
198,0.007749,0.007726,0.008504,0.004292,0.004276,0.004873
199,0.008622,0.008604,0.009391,0.004556,0.004529,0.005129
200,0.007594,0.007591,0.008349,0.004271,0.004254,0.004856
```

B.2.3 Positive Definite Band Matrix - $n = 200 : 4000$ and $b = 100 : 2000$

```
POSDEF BANDED MATRIX TEST
 N,B,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SGBTRF,ASD1_SGBTRF,SPOTRF,ASD1_SPOTRF,ASD2_SPOTRF,SPBTRF,ASD1_SPBTRF
200,100,0.007276,0.006183,0.006976,0.004912,0.005198,0.004080,0.003840,0.004345,0.003205,0.003453
400,100,0.044938,0.020903,0.023448,0.012593,0.013976,0.023982,0.014659,0.016469,0.007829,0.008544
400,200,0.044910,0.033514,0.036159,0.026718,0.027778,0.023987,0.021602,0.023155,0.015893,0.016379
600,100,0.135513,0.039667,0.043485,0.020247,0.022718,0.072947,0.029484,0.034296,0.012490,0.013659
600,200,0.135412,0.063788,0.067771,0.046926,0.050001,0.072974,0.047957,0.051470,0.027686,0.028682
600,300,0.135428,0.091865,0.095873,0.077713,0.079824,0.072929,0.062715,0.065325,0.043276,0.043915
800,100,0.302121,0.060815,0.066130,0.028000,0.031595,0.163837,0.048817,0.058756,0.017577,0.019262
800,200,0.302214,0.096898,0.102571,0.067091,0.072539,0.163959,0.079015,0.088943,0.039692,0.041118
800,300,0.302223,0.143681,0.149392,0.116825,0.122157,0.163981,0.111125,0.117821,0.064901,0.066096
800,400,0.302030,0.194006,0.199333,0.168842,0.172831,0.163994,0.138264,0.142901,0.091968,0.092880
1000,100,0.563626,0.092286,0.098805,0.036017,0.040587,0.309048,0.073183,0.092217,0.022431,0.024588
1000,200,0.563482,0.138747,0.146156,0.087301,0.094778,0.308735,0.114955,0.137280,0.051676,0.053668
1000,300,0.563472,0.203301,0.210898,0.156007,0.164689,0.308637,0.164467,0.183195,0.086693,0.088267
1000,400,0.563414,0.276122,0.283457,0.233297,0.241750,0.308895,0.215046,0.227688,0.126314,0.127756
1000,500,0.563271,0.348510,0.355826,0.314216,0.320592,0.308866,0.256914,0.265296,0.167102,0.168317
1200,100,0.955752,0.121279,0.130330,0.043589,0.049463,0.522332,0.100983,0.131497,0.027285,0.029830
1200,200,0.955601,0.180487,0.189864,0.107554,0.117164,0.521998,0.154870,0.194897,0.063825,0.066277
1200,300,0.955715,0.263554,0.273180,0.195201,0.207171,0.522241,0.221406,0.260327,0.108752,0.110860
1200,400,0.955377,0.361378,0.370613,0.297440,0.310511,0.521906,0.296750,0.327959,0.160966,0.162956
1200,500,0.955633,0.462871,0.472676,0.409639,0.421318,0.522041,0.370060,0.390979,0.217134,0.218960
1200,600,0.955491,0.567062,0.576573,0.528009,0.536693,0.522090,0.428372,0.443505,0.277893,0.279352
1400,100,1.503816,0.164201,0.174196,0.051277,0.058353,0.814845,0.134335,0.179258,0.032148,0.035213
1400,200,1.503952,0.233709,0.244451,0.127678,0.139397,0.813613,0.199476,0.262742,0.075923,0.078851
1400,300,1.504154,0.333923,0.344984,0.233830,0.249219,0.813620,0.283736,0.349863,0.130263,0.132843
1400,400,1.504053,0.454929,0.465602,0.361613,0.378250,0.813292,0.382486,0.442404,0.195202,0.197716
1400,500,1.503902,0.585292,0.596811,0.504803,0.521646,0.813031,0.486961,0.533900,0.267172,0.269630
1400,600,1.503919,0.721796,0.733379,0.661017,0.676351,0.813923,0.581202,0.615023,0.347111,0.349181
1400,700,1.503988,0.867797,0.879538,0.811362,0.823507,0.814098,0.663596,0.687505,0.421753,0.423179
1600,100,2.230532,0.195186,0.207525,0.058935,0.067289,1.200527,0.171351,0.231584,0.037360,0.040616
1600,200,2.230609,0.278202,0.290488,0.147651,0.161752,1.200581,0.248996,0.337710,0.087651,0.091117
1600,300,2.230882,0.398566,0.411357,0.272720,0.291373,1.200546,0.349145,0.448382,0.151940,0.154834
1600,400,2.230713,0.545653,0.558174,0.425333,0.446732,1.200356,0.474061,0.569960,0.229258,0.232406
1600,500,2.230915,0.702974,0.715888,0.599120,0.621599,1.200192,0.608266,0.690431,0.316995,0.320071
1600,600,2.231121,0.875772,0.888781,0.794308,0.816111,1.200188,0.740887,0.808199,0.416465,0.419286
1600,700,2.230914,1.064542,1.077949,0.986447,1.006224,1.200166,0.866492,0.915968,0.512437,0.514355
1600,800,2.230699,1.253854,1.267505,1.193672,1.209996,1.199995,0.965818,1.002314,0.622818,0.624559
1800,100,3.155573,0.252758,0.265414,0.066807,0.076104,1.689094,0.217390,0.298902,0.042129,0.046064
1800,200,3.155360,0.348047,0.361708,0.167949,0.184132,1.688738,0.305030,0.427919,0.099715,0.103837
1800,300,3.155915,0.483273,0.497859,0.311515,0.333476,1.688538,0.422819,0.564669,0.173695,0.177368
1800,400,3.155869,0.650858,0.665554,0.489222,0.514874,1.688751,0.569969,0.714112,0.263625,0.267489
```

```
1800,500,3.155356,0.837586,0.852526,0.693733,0.721755,1.688501,0.735559,0.869298,0.367055,0.370749
1800,600,3.155568,1.042778,1.058575,0.927026,0.955595,1.688603,0.903681,1.019443,0.485781,0.489322
1800,700,3.155605,1.271085,1.287065,1.160748,1.188321,1.688170,1.072377,1.165379,0.602407,0.605219
1800,800,3.155584,1.506851,1.523571,1.420286,1.445253,1.687887,1.219826,1.290550,0.739284,0.741138
1800,900,3.155700,1.739158,1.755211,1.674594,1.695536,1.687828,1.362203,1.411998,0.864530,0.866450
2000,100,4.305044,0.290700,0.306886,0.074368,0.084873,2.297499,0.261769,0.365102,0.047165,0.051484
2000,200,4.304963,0.396349,0.412563,0.188100,0.206639,2.297335,0.360084,0.520786,0.111593,0.116120
2000,300,4.305310,0.554300,0.570628,0.350269,0.375254,2.297016,0.494624,0.686271,0.195673,0.199805
2000,400,4.304668,0.749007,0.765614,0.552969,0.583075,2.296863,0.666156,0.868133,0.298080,0.302610
2000,500,4.305152,0.962495,0.980084,0.788201,0.821787,2.297100,0.861325,1.057421,0.416829,0.421202
2000,600,4.304879,1.203302,1.221036,1.059904,1.095159,2.296740,1.067700,1.246863,0.555040,0.559018
2000,700,4.305119,1.473150,1.491187,1.335220,1.370642,2.296843,1.280245,1.433522,0.692645,0.695860
2000,800,4.305099,1.758471,1.776442,1.646257,1.679772,2.296811,1.476228,1.601486,0.856291,0.858474
2000,900,4.304745,2.041361,2.059644,1.954588,1.985115,2.296729,1.678110,1.772313,1.007834,1.010882
2000,1000,4.305125,2.342773,2.360569,2.280480,2.305752,2.296701,1.834592,1.902926,1.183898,1.186364
2200,100,5.692168,0.364100,0.379593,0.082005,0.093959,3.038086,0.323697,0.452483,0.052213,0.057042
2200,200,5.691701,0.480410,0.497411,0.208098,0.228782,3.037487,0.432075,0.635853,0.123356,0.128431
2200,300,5.692084,0.650928,0.668650,0.388650,0.417041,3.037068,0.583387,0.831904,0.217203,0.221875
2200,400,5.693409,0.864295,0.882670,0.616917,0.651369,3.036844,0.777329,1.047062,0.332891,0.337440
2200,500,5.692039,1.108404,1.127417,0.882991,0.921859,3.036451,1.001898,1.273060,0.467077,0.472085
2200,600,5.692283,1.383830,1.403087,1.192565,1.234462,3.036772,1.243303,1.500868,0.623799,0.628259
2200,700,5.692202,1.691463,1.711401,1.509714,1.552473,3.036525,1.499641,1.731620,0.782811,0.786240
2200,800,5.692616,2.021574,2.041861,1.871124,1.913353,3.036496,1.744061,1.944122,0.972625,0.976034
2200,900,5.692211,2.356283,2.376474,2.234371,2.274695,3.036653,2.001011,2.164958,1.151673,1.154968
2200,1000,5.692411,2.716085,2.736706,2.624784,2.661023,3.036684,2.217994,2.343698,1.361567,1.364406
2200,1100,5.692460,3.096686,3.117070,2.998980,3.029975,3.036903,2.410625,2.511471,1.537003,1.539166
2400,100,7.362470,0.396477,0.415527,0.089780,0.102687,3.930707,0.372667,0.525023,0.057132,0.062351
2400,200,7.361747,0.532500,0.551935,0.228175,0.251058,3.930876,0.492711,0.741262,0.135569,0.141258
2400,300,7.362345,0.721576,0.741686,0.427043,0.458736,3.930421,0.661923,0.972151,0.238895,0.244081
2400,400,7.362523,0.960785,0.981000,0.680467,0.719031,3.930199,0.880850,1.225220,0.366988,0.372231
2400,500,7.361897,1.233975,1.253585,0.977096,1.021667,3.931019,1.136854,1.492273,0.516998,0.522270
2400,600,7.362642,1.544471,1.565762,1.325384,1.373617,3.930113,1.416955,1.766763,0.693554,0.698794
2400,700,7.361017,1.897127,1.919057,1.684550,1.734599,3.930041,1.716300,2.045387,0.873374,0.877535
2400,800,7.360402,2.279372,2.301040,2.096804,2.147779,3.929793,2.009978,2.305613,1.089414,1.092859
2400,900,7.358942,2.666745,2.688370,2.513863,2.563879,3.930400,2.328129,2.583000,1.295880,1.299314
2400,1000,7.358195,3.089739,3.111460,2.969027,3.015862,3.930009,2.609431,2.817478,1.539824,1.543270
2400,1100,7.349622,3.536122,3.557912,3.410553,3.453179,3.928793,2.870287,3.042140,1.745298,1.748798
2400,1200,7.347807,4.008973,4.030438,3.896790,3.932234,3.928941,3.105270,3.240586,1.993205,1.992869
2600,100,9.294678,0.490652,0.509003,0.097686,0.111629,4.968153,0.451974,0.638375,0.062091,0.067899
2600,200,9.293038,0.632686,0.653362,0.248081,0.273195,4.968590,0.581103,0.885725,0.147794,0.153957
2600,300,9.291942,0.839791,0.861445,0.465408,0.500594,4.967815,0.765877,1.151463,0.260885,0.266555
2600,400,9.285351,1.099156,1.121331,0.743930,0.786798,4.967827,1.006137,1.442238,0.401103,0.407365
2600,500,9.273680,1.399210,1.422268,1.070892,1.120752,4.967454,1.291086,1.751481,0.566653,0.572683
2600,600,9.266787,1.739711,1.763249,1.457594,1.512386,4.966602,1.604298,2.069073,0.762649,0.768553
2600,700,9.264970,2.132817,2.157031,1.858068,1.915734,4.965979,1.947871,2.396350,0.962867,0.966749
2600,800,9.264299,2.558049,2.582371,2.322430,2.382101,4.966424,2.287115,2.705045,1.206541,1.209802
2600,900,9.262798,2.996299,3.020929,2.793932,2.853928,4.966197,2.664388,3.033641,1.439925,1.443804
2600,1000,9.263314,3.475038,3.500455,3.312909,3.370593,4.966359,3.003095,3.324044,1.715621,1.719251
2600,1100,9.264758,3.983584,4.008484,3.821513,3.875699,4.966390,3.334321,3.605349,1.955271,1.958251
2600,1200,9.266073,4.523471,4.548228,4.385155,4.433343,4.967829,3.645960,3.867068,2.241644,2.242088
2600,1300,9.284995,5.058902,5.089568,4.978273,5.021024,4.969297,3.918972,4.096558,2.485228,2.487078
2800,100,11.636596,0.530522,0.554382,0.105383,0.120510,6.193280,0.497476,0.718800,0.067025,0.073273
2800,200,11.636975,0.682294,0.706241,0.268282,0.295522,6.194489,0.638043,1.002052,0.159652,0.166256
2800,300,11.640697,0.910381,0.934301,0.504123,0.542499,6.194482,0.840258,1.307281,0.282606,0.288852
2800,400,11.640260,1.201576,1.226235,0.807976,0.854658,6.193376,1.107830,1.642320,0.435468,0.442064
2800,500,11.639749,1.531601,1.556768,1.165310,1.220408,6.193391,1.424223,1.999658,0.616804,0.623507
2800,600,11.637318,1.910360,1.935379,1.590555,1.651641,6.192956,1.777371,2.370316,0.831505,0.836929
2800,700,11.638690,2.347873,2.373514,2.033528,2.098742,6.193404,2.165559,2.749878,1.052911,1.058094
2800,800,11.638487,2.822730,2.848575,2.548819,2.616609,6.193641,2.554058,3.113922,1.321764,1.325880
2800,900,11.637808,3.316423,3.342289,3.074923,3.144434,6.192688,2.991627,3.505216,1.584179,1.587347
2800,1000,11.636797,3.858147,3.883864,3.660149,3.728492,6.193474,3.394496,3.849653,1.894444,1.898063
2800,1100,11.628487,4.443632,4.469858,4.236981,4.302813,6.191718,3.799117,4.196518,2.165785,2.169618
2800,1200,11.618380,5.076475,5.101958,4.883990,4.945293,6.189411,4.185756,4.519873,2.487908,2.489990
2800,1300,11.623989,5.675552,5.700921,5.534893,5.590443,6.188139,4.540773,4.819934,2.771066,2.773356
2800,1400,11.603050,6.225646,6.249997,6.203754,6.253149,6.187067,4.862122,5.084394,3.133061,3.133872
3000,100,14.188718,0.640507,0.662022,0.113138,0.129425,7.589185,0.599239,0.854002,0.072133,0.078628
3000,200,14.187405,0.805968,0.830116,0.288038,0.317417,7.589099,0.749433,1.175898,0.171696,0.178770
3000,300,14.192443,1.042858,1.068205,0.542312,0.583949,7.588894,0.967075,1.522184,0.304108,0.310694
```

```
3000,400,14.264688,1.351404,1.377304,0.870736,0.922592,7.598267,1.254953,1.902053,0.469910,0.477330
3000,500,14.279517,1.710649,1.738649,1.259713,1.320418,7.597656,1.600619,2.307851,0.666857,0.673701
3000,600,14.279481,2.123317,2.151172,1.723482,1.790963,7.599608,1.987436,2.727950,0.900332,0.906772
3000,700,14.278185,2.600455,2.629111,2.207522,2.280105,7.597904,2.420488,3.163084,1.143523,1.148177
3000,800,14.260454,3.118483,3.147543,2.774228,2.850623,7.596609,2.856106,3.581848,1.438867,1.443929
3000,900,14.258724,3.664854,3.693272,3.354440,3.434273,7.596440,3.353793,4.029534,1.727379,1.733261
3000,1000,14.258101,4.265049,4.293080,4.005606,4.084935,7.596574,3.812909,4.437258,2.071050,2.075473
3000,1100,14.257550,4.921747,4.950138,4.649238,4.727559,7.596687,4.287790,4.847802,2.374106,2.378714
3000,1200,14.259004,5.628037,5.657410,5.382636,5.456701,7.597142,4.754696,5.238642,2.739975,2.739865
3000,1300,14.254697,6.302615,6.328547,6.136352,6.207742,7.595895,5.191405,5.606647,3.059281,3.061731
3000,1400,14.228220,6.936900,6.965954,6.889952,6.954320,7.591917,5.596230,5.943409,3.470116,3.472279
3000,1500,14.217198,7.618864,7.646341,7.581148,7.636613,7.588578,5.956846,6.246032,3.762449,3.764488
3200,100,17.184191,0.670983,0.697394,0.120552,0.138133,9.197127,0.648485,0.936635,0.077055,0.084225
3200,200,17.237649,0.853089,0.878297,0.307823,0.339503,9.198277,0.812172,1.299256,0.183914,0.191292
3200,300,17.295266,1.115833,1.141129,0.581015,0.625621,9.213838,1.051027,1.693705,0.325946,0.333013
3200,400,17.294311,1.451748,1.477670,0.934894,0.990293,9.212289,1.365651,2.130079,0.503878,0.512112
3200,500,17.294401,1.836803,1.863961,1.353706,1.419555,9.211927,1.743987,2.588240,0.716640,0.724617
3200,600,17.295245,2.285784,2.313525,1.856292,1.930090,9.212899,2.168920,3.063663,0.968861,0.976509
3200,700,17.283140,2.810990,2.839543,2.381875,2.461882,9.210187,2.648478,3.558484,1.232738,1.237682
3200,800,17.270790,3.386146,3.414123,2.999623,3.084659,9.209818,3.132283,4.039518,1.555965,1.560879
3200,900,17.270186,3.985617,4.014341,3.634559,3.723185,9.210405,3.682777,4.553388,1.870223,1.875490
3200,1000,17.268582,4.645881,4.674722,4.350506,4.440853,9.210034,4.215805,5.028056,2.247519,2.253358
3200,1100,17.268667,5.379836,5.409782,5.061749,5.151819,9.209667,4.763928,5.503955,2.584061,2.589795
3200,1200,17.252518,6.157814,6.187340,5.871270,5.958133,9.204330,5.300950,5.962490,2.988599,2.989804
3200,1300,17.240042,6.909813,6.936086,6.689315,6.771981,9.198153,5.822742,6.401783,3.344096,3.347076
3200,1400,17.189223,7.612442,7.643639,7.543061,7.654918,9.197890,6.322392,6.815761,3.804424,3.806072
3200,1500,17.293332,8.446269,8.472494,8.407154,8.480247,9.209791,6.786305,7.207085,4.139958,4.142801
3200,1600,17.270502,9.245170,9.276712,9.210423,9.274859,9.210134,7.176109,7.526752,4.625046,4.622828
3400,100,20.677989,0.803109,0.827867,0.128425,0.147520,11.017856,0.767007,1.101836,0.082078,0.089680
3400,200,20.670430,1.000785,1.027677,0.327835,0.361769,11.016801,0.938642,1.508292,0.195700,0.203707
3400,300,20.654430,1.272078,1.301040,0.618985,0.666882,11.013491,1.189744,1.945948,0.347618,0.355236
3400,400,20.631101,1.627224,1.657088,0.997999,1.057475,11.007065,1.526054,2.430689,0.537798,0.546384
3400,500,20.628221,2.042752,2.073937,1.447684,1.518683,11.004976,1.931929,2.942526,0.766504,0.775232
3400,600,20.597125,2.524728,2.557035,1.988005,2.068485,11.002828,2.393419,3.473448,1.037310,1.045304
3400,700,20.575251,3.085546,3.118487,2.555021,2.642629,11.001185,2.912833,4.026782,1.322224,1.327957
3400,800,20.568731,3.702872,3.735046,3.224633,3.318796,11.001753,3.445599,4.563255,1.672560,1.677360
3400,900,20.700832,4.356927,4.390292,3.916056,4.014627,11.017869,4.061191,5.140822,2.016167,2.022251
3400,1000,20.703964,5.077605,5.110416,4.696463,4.797492,11.019236,4.645113,5.674873,2.425669,2.431542
3400,1100,20.703786,5.876320,5.909779,5.476839,5.578652,11.018697,5.262876,6.218063,2.791601,2.797613
3400,1200,20.704855,6.736556,6.769324,6.374483,6.473949,11.018384,5.876215,6.759606,3.235195,3.233518
3400,1300,20.677845,7.568697,7.603648,7.301184,7.401012,11.016210,6.484056,7.273019,3.630465,3.634246
3400,1400,20.677826,8.369187,8.402566,8.269615,8.364278,11.016164,7.070490,7.761963,4.139999,4.142187
3400,1500,20.679247,9.250923,9.283124,9.176720,9.264585,11.014296,7.619220,8.220652,4.516541,4.518917
3400,1600,20.623603,10.131552,10.164868,10.026757,10.098497,11.001247,8.101476,8.608822,5.060404,5.057402
3400,1700,20.560309,10.986579,11.020474,10.969589,11.091800,11.013404,8.630779,9.046115,5.459004,5.459959
3600,100,24.542951,0.836761,0.869420,0.136035,0.156029,13.042194,0.805904,1.190976,0.087056,0.095073
3600,200,24.518321,1.042007,1.074279,0.347958,0.383829,13.045334,0.990129,1.643595,0.207153,0.216016
3600,300,24.517019,1.340175,1.373087,0.657422,0.708800,13.043775,1.260877,2.133241,0.369429,0.377675
3600,400,24.516527,1.724959,1.757291,1.061313,1.125404,13.044006,1.624203,2.670571,0.572414,0.581536
3600,500,24.530529,2.171091,2.204758,1.542161,1.619188,13.043035,2.063651,3.244685,0.815394,0.824691
3600,600,24.517058,2.690235,2.724446,2.121021,2.207619,13.040979,2.565307,3.844476,1.106987,1.114722
3600,700,24.474915,3.298030,3.332795,2.729870,2.824861,13.034164,3.131560,4.458566,1.412586,1.418437
3600,800,24.478826,3.969718,4.004201,3.450381,3.552853,13.031380,3.712983,5.059173,1.789113,1.793798
3600,900,24.462694,4.673517,4.707818,4.193754,4.302420,13.023872,4.386023,5.700397,2.157540,2.163824
3600,1000,24.413056,5.457139,5.491216,5.038959,5.151044,13.023484,5.038670,6.305727,2.605662,2.609461
3600,1100,24.494224,6.325923,6.360769,5.886137,5.999457,13.023554,5.726383,6.923102,3.001503,3.007239
3600,1200,24.555467,7.287635,7.321682,6.872144,6.982763,13.046405,6.419100,7.528176,3.483452,3.486474
3600,1300,24.518201,8.198431,8.233617,7.881723,7.995489,13.044629,7.110063,8.132825,3.917378,3.920455
3600,1400,24.516905,9.073131,9.109158,8.946367,9.056933,13.043735,7.789686,8.693182,4.476206,4.478160
3600,1500,24.505125,10.047119,10.077416,9.928981,10.026973,13.030636,8.432583,9.242726,4.891469,4.894466
3600,1600,24.416699,11.012661,11.045182,10.893846,11.037721,13.023429,9.007025,9.709548,5.495610,5.493251
3600,1700,24.549839,12.067002,12.097788,12.042628,12.132362,13.043855,9.660644,10.260343,5.938086,5.939078
3600,1800,24.519066,12.973938,13.010183,13.130719,13.211962,13.044661,10.195330,10.711061,6.678254,6.676999
3800,100,28.762226,0.991797,1.018999,0.143934,0.164955,15.331436,0.951821,1.380581,0.092014,0.100324
3800,200,28.707550,1.208157,1.239593,0.367715,0.405868,15.321651,1.144389,1.881683,0.219948,0.229000
3800,300,28.699336,1.516541,1.549507,0.695439,0.749834,15.312181,1.430275,2.423402,0.391200,0.399842
3800,400,28.624347,1.918175,1.951876,1.124581,1.192809,15.306805,1.813160,3.021207,0.606952,0.616402
3800,500,28.608705,2.393947,2.428818,1.636067,1.717979,15.306472,2.281975,3.657522,0.865944,0.875437
```

```
3800,600,28.614247,2.947783,2.984333,2.252841,2.346411,15.307694,2.816308,4.312968,1.176552,1.185526
3800,700,28.750145,3.592154,3.628292,2.903440,3.006332,15.309937,3.424795,4.993781,1.502621,1.509682
3800,800,28.802577,4.305907,4.342246,3.676088,3.786982,15.330427,4.055653,5.659192,1.905136,1.911996
3800,900,28.774022,5.062045,5.098088,4.474872,4.592759,15.331789,4.793740,6.380801,2.302556,2.309795
3800,1000,28.775591,5.907850,5.944574,5.385874,5.508012,15.332232,5.499046,7.042173,2.780549,2.788211
3800,1100,28.773170,6.844635,6.880684,6.300388,6.425158,15.332884,6.253783,7.730460,3.211676,3.219266
3800,1200,28.771836,7.855995,7.892555,7.358516,7.484264,15.331960,7.027629,8.425973,3.735296,3.739237
3800,1300,28.766243,8.839944,8.877797,8.461864,8.588420,15.324786,7.797888,9.091815,4.204871,4.209745
3800,1400,28.705101,9.772411,9.808035,9.565731,9.687459,15.308683,8.558925,9.723666,4.812160,4.814611
3800,1500,28.708277,10.887936,10.930005,10.749665,10.870629,15.333249,9.314107,10.378789,5.269992,5.274681
3800,1600,28.787488,11.965437,12.003374,11.825781,11.942936,15.331740,9.982921,10.928691,5.932848,5.931423
3800,1700,28.772621,13.062580,13.101307,13.027635,13.136196,15.329756,10.741214,11.554411,6.419143,6.421570
3800,1800,28.721452,14.056833,14.079066,14.126523,14.227355,15.308200,11.369400,12.088234,7.205174,7.202616
3800,1900,28.774713,15.202924,15.241150,15.430288,15.490937,15.329119,11.954916,12.526073,7.600613,7.600745
4000,100,33.467723,1.019812,1.054842,0.151398,0.173851,17.883138,1.000344,1.467133,0.097149,0.106133
4000,200,33.460045,1.240236,1.274617,0.387587,0.428104,17.881552,1.206124,2.026058,0.231282,0.241186
4000,300,33.470310,1.587689,1.623107,0.734007,0.791885,17.880778,1.510973,2.627741,0.412604,0.421771
4000,400,33.440280,2.026553,2.061226,1.187720,1.260420,17.874016,1.923219,3.291237,0.641313,0.650756
4000,500,33.428505,2.525861,2.562206,1.730406,1.817920,17.866698,2.423229,3.988897,0.915713,0.925865
4000,600,33.431909,3.115877,3.152654,2.385894,2.485537,17.858490,3.000794,4.719175,1.245521,1.255200
4000,700,33.353498,3.804176,3.842101,3.077746,3.188300,17.854004,3.657798,5.467785,1.592785,1.600350
4000,800,33.308250,4.571694,4.608338,3.899468,4.019899,17.852603,4.337917,6.196741,2.020075,2.027043
4000,900,33.479479,5.392842,5.430186,4.755363,4.882657,17.884459,5.138128,6.994995,2.446253,2.453250
4000,1000,33.510654,6.307357,6.344892,5.731020,5.864239,17.884873,5.913315,7.735046,2.958723,2.965871
4000,1100,33.464073,7.309842,7.348606,6.712867,6.849688,17.882311,6.735403,8.505743,3.418586,3.425144
4000,1200,33.465712,8.404066,8.443521,7.853010,7.991403,17.882648,7.598681,9.274452,3.984422,3.986281
4000,1300,33.437497,9.459566,9.498774,9.027838,9.163270,17.866014,8.441354,10.027839,4.490197,4.495063
4000,1400,33.370117,10.469074,10.505478,10.236499,10.426221,17.856732,9.305331,10.763416,5.146825,5.149371
4000,1500,33.515202,11.686732,11.726183,11.518264,11.637710,17.884224,10.157267,11.505772,5.646332,5.650435
4000,1600,33.466754,12.862643,12.902447,12.698547,12.833751,17.881820,10.920451,12.139163,6.363938,6.362650
4000,1700,33.464607,14.069872,14.098980,13.971728,14.089784,17.862916,11.782970,12.854225,6.891113,6.892941
4000,1800,33.298995,15.129028,15.168822,15.295972,15.504624,17.885377,12.557570,13.526716,7.810429,7.807230
4000,1900,33.466329,16.414133,16.454418,16.615073,16.726387,17.882252,13.251594,14.053737,8.201984,8.203943
4000,2000,33.465629,17.642313,17.679002,17.764196,17.831760,17.853940,13.868248,14.582876,9.073613,9.076887
```

B.2.4 Positive Definite Band Matrix - $n = 4000$ and $b = 1 : 200$

```
POSDEF BANDED MATRIX TEST
 N,B,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SGBTRF,ASD1_SGBTRF,SPOTRF,ASD1_SPOTRF,ASD2_SPOTRF,SPBTRF,ASD1_SPBTRF
4000,1,33.716096,1.040130,1.101915,0.005837,0.005752,0.000354,0.000319,0.028724,0.000073,0.000069
4000,2,33.519953,0.900116,0.949030,0.005685,0.005677,1.288636,0.154130,0.195480,0.000860,0.000853
4000,3,33.516642,0.899557,0.945384,0.006081,0.006071,17.884406,0.915474,0.942062,0.005566,0.005552
4000,4,33.513904,0.898519,0.944785,0.008616,0.008612,17.883717,0.914026,0.963230,0.006947,0.006887
4000,5,33.517452,0.900164,0.946336,0.008772,0.008755,17.883903,0.914613,0.975398,0.007473,0.007455
4000,6,33.504234,0.901209,0.947137,0.010437,0.010423,17.880872,0.915249,0.982716,0.008840,0.008848
4000,7,33.470488,0.903407,0.949587,0.011284,0.011256,17.873683,0.917087,0.984838,0.009084,0.009068
4000,8,33.499330,0.901570,0.947407,0.011920,0.011948,17.864715,0.915840,1.014321,0.010918,0.010891
4000,9,33.493880,0.903668,0.949486,0.012803,0.012817,17.861984,0.917024,1.020336,0.010971,0.011059
4000,10,33.452523,0.904800,0.950878,0.014639,0.014635,17.856740,0.918303,1.024343,0.012265,0.012229
4000,11,33.535527,0.906506,0.952793,0.016367,0.016412,17.885372,0.919343,1.034579,0.013055,0.012996
4000,12,33.558863,0.904782,0.950726,0.017649,0.017613,17.885372,0.917651,1.041718,0.014025,0.014150
4000,13,33.561797,0.906519,0.952927,0.018481,0.018546,17.885223,0.918549,1.037530,0.014658,0.014563
4000,14,33.538842,0.908014,0.954292,0.021113,0.021482,17.877303,0.919713,1.040133,0.016391,0.016491
4000,15,33.505022,0.910111,0.956097,0.022318,0.022356,17.877523,0.920923,1.044421,0.017063,0.017086
4000,16,33.511951,0.906804,0.952594,0.022232,0.022390,17.884126,0.919517,1.055158,0.017787,0.017759
4000,17,33.515585,0.909459,0.954755,0.025478,0.025514,17.884416,0.921066,1.053410,0.018666,0.018564
4000,18,33.540998,0.910315,0.956074,0.026596,0.026874,17.885730,0.922651,1.063822,0.021223,0.021092
4000,19,33.555121,0.912882,0.958362,0.027796,0.027973,17.885524,0.923712,1.069600,0.020401,0.020381
4000,20,33.512004,0.910114,0.954982,0.028312,0.028623,17.873550,0.922043,1.071318,0.023570,0.023502
4000,21,33.508220,0.912309,0.957423,0.033261,0.033583,17.869046,0.922978,1.076961,0.023164,0.023164
4000,22,33.516150,0.913545,0.958638,0.031492,0.031855,17.865559,0.924481,1.084796,0.027446,0.027261
4000,23,33.502670,0.915886,0.960709,0.035098,0.035163,17.883545,0.925480,1.087070,0.024334,0.024350
4000,24,33.518525,0.912592,0.957428,0.033524,0.033466,17.883936,0.924452,1.106299,0.028113,0.028111
4000,25,33.533440,0.915521,0.959854,0.036225,0.035996,17.884716,0.926059,1.115477,0.029002,0.029833
```

```
4000,26,33.550716,0.916376,0.961174,0.035694,0.035441,17.885299,0.927217,1.120782,0.032410,0.032237
4000,27,33.542067,0.919134,0.964134,0.038710,0.039052,17.880844,0.928707,1.122077,0.028637,0.028617
4000,28,33.508277,0.915811,0.960662,0.036329,0.036802,17.871535,0.927408,1.125610,0.032905,0.032930
4000,29,33.508347,0.918975,0.963880,0.041323,0.041357,17.867502,0.928936,1.135353,0.032163,0.032118
4000,30,33.511947,0.920063,0.964965,0.040538,0.040785,17.900523,0.930114,1.137706,0.038706,0.040059
4000,31,33.519292,0.922632,0.967530,0.044947,0.045282,17.884087,0.931420,1.141345,0.033677,0.033297
4000,32,33.534052,0.919581,0.963936,0.041955,0.042239,17.884734,0.930339,1.142188,0.036716,0.036798
4000,33,33.499437,0.922482,0.966710,0.047941,0.047995,17.870777,0.931830,1.146147,0.036801,0.036691
4000,34,33.515831,0.923555,0.968204,0.046428,0.046720,17.863646,0.933244,1.154261,0.043839,0.043826
4000,35,33.504252,0.926559,0.971049,0.051493,0.052048,17.861659,0.934705,1.156523,0.039713,0.039657
4000,36,33.451061,0.926484,0.970945,0.048355,0.048798,17.854577,0.933197,1.157383,0.044621,0.044570
4000,37,33.499153,0.929209,0.973706,0.055336,0.055890,17.886702,0.935043,1.168242,0.043993,0.043918
4000,38,33.566766,0.930563,0.975104,0.054218,0.054643,17.887439,0.936133,1.175325,0.049909,0.049764
4000,39,33.559606,0.933234,0.978162,0.061368,0.062105,17.886276,0.937867,1.178351,0.044546,0.044578
4000,40,33.555934,0.932225,0.976759,0.054676,0.054950,17.886463,0.936377,1.171079,0.051758,0.051826
4000,41,33.541069,0.935183,0.979316,0.064958,0.065864,17.885129,0.937909,1.176087,0.050282,0.050324
4000,42,33.512377,0.936237,0.980955,0.061522,0.062115,17.883977,0.939211,1.180990,0.057911,0.058012
4000,43,33.517150,0.939407,0.983688,0.067922,0.068634,17.884845,0.940868,1.193019,0.051534,0.054821
4000,44,33.516318,0.937753,0.982020,0.061551,0.062204,17.884241,0.939578,1.202505,0.059597,0.059512
4000,45,33.513955,0.940794,0.985467,0.071740,0.072745,17.884728,0.941602,1.195803,0.057714,0.057666
4000,46,33.511525,0.942108,0.987129,0.067671,0.068385,17.885037,0.942789,1.197093,0.065573,0.065720
4000,47,33.511271,0.945396,0.989927,0.076659,0.077471,17.884015,0.944442,1.202147,0.055834,0.055888
4000,48,33.500228,0.943675,0.987905,0.068657,0.068990,17.879023,0.943450,1.214905,0.066352,0.066325
4000,49,33.491789,0.947064,0.991110,0.080363,0.081210,17.866288,0.945337,1.210833,0.063440,0.063428
4000,50,33.493736,0.947704,0.992504,0.074890,0.075705,17.860674,0.946531,1.222056,0.073141,0.073177
4000,51,33.424561,0.951937,0.995911,0.082572,0.083313,17.855129,0.948386,1.232932,0.062434,0.062525
4000,52,33.410152,0.948663,0.993001,0.074979,0.075741,17.854733,0.947407,1.231983,0.074107,0.074086
4000,53,33.450041,0.952351,0.996166,0.089633,0.090322,17.854302,0.949198,1.235833,0.070697,0.070559
4000,54,33.543198,0.953424,0.997813,0.082493,0.083189,17.877929,0.950804,1.243103,0.080451,0.081384
4000,55,33.550789,0.956937,1.001350,0.092940,0.093724,17.886006,0.952383,1.247295,0.068565,0.068585
4000,56,33.519064,0.953227,0.997468,0.082605,0.083330,18.003406,0.950897,1.255177,0.082076,0.082255
4000,57,33.511172,0.956732,1.001190,0.095729,0.096236,18.002875,0.952807,1.261755,0.078229,0.078271
4000,58,33.515037,0.957789,1.002399,0.090445,0.091239,18.002456,0.954402,1.268004,0.090498,0.089790
4000,59,33.512692,0.961400,1.005693,0.099997,0.100845,18.002961,0.956154,1.274167,0.074243,0.074298
4000,60,33.510169,0.958302,1.002425,0.090248,0.091044,17.884340,0.954886,1.278572,0.090322,0.090511
4000,61,33.513151,0.962255,1.006203,0.102962,0.103729,17.999051,0.957037,1.281480,0.085870,0.085856
4000,62,33.507532,0.963876,1.008072,0.100462,0.101410,17.995731,0.958512,1.284582,0.096912,0.096655
4000,63,33.475333,0.968041,1.011538,0.112407,0.112752,17.993859,0.960555,1.294357,0.080787,0.080947
4000,64,33.472487,0.964063,1.007783,0.099929,0.100745,17.869831,0.959317,1.296041,0.099777,0.100590
4000,65,33.474166,0.967525,1.011163,0.098300,0.112624,17.857151,0.961704,1.304977,0.066787,0.075837
4000,66,33.417228,0.968656,1.012805,0.094763,0.109520,17.854125,0.963232,1.315328,0.065660,0.074793
4000,67,33.407071,0.972668,1.016644,0.104911,0.121111,17.854514,0.965097,1.314842,0.070040,0.079546
4000,68,33.493051,0.969694,1.013716,0.095371,0.110253,17.882476,0.963819,1.316748,0.065790,0.074791
4000,69,33.514770,0.973401,1.017285,0.102890,0.120336,17.900969,0.965771,1.331942,0.070936,0.080157
4000,70,33.513581,0.974737,1.018572,0.102502,0.118320,17.885035,0.967237,1.338632,0.069832,0.078990
4000,71,33.511467,0.978799,1.022792,0.112596,0.128679,17.884995,0.969243,1.342263,0.074432,0.083451
4000,72,33.510003,0.975614,1.019822,0.101826,0.118222,17.885219,0.967910,1.335502,0.069533,0.078659
4000,73,33.513163,0.980009,1.023920,0.110671,0.126744,17.884095,0.970928,1.342134,0.074444,0.083497
4000,74,33.499525,0.981449,1.025032,0.109317,0.125499,17.882777,0.972388,1.346492,0.073331,0.082366
4000,75,33.476723,0.986085,1.029357,0.118839,0.136385,17.874576,0.974828,1.359433,0.078079,0.087754
4000,76,33.484941,0.983213,1.026559,0.108904,0.125216,17.864182,0.973512,1.367132,0.073460,0.082175
4000,77,33.462526,0.987305,1.030904,0.118753,0.136110,17.870769,0.975739,1.362258,0.078379,0.087636
4000,78,33.408376,0.988345,1.032377,0.116024,0.132751,17.854183,0.977427,1.364754,0.077510,0.086361
4000,79,33.409367,0.992655,1.036315,0.126624,0.144460,17.853639,0.979467,1.370317,0.081366,0.090866
4000,80,33.473831,0.988915,1.032224,0.116393,0.134049,17.857926,0.978222,1.383817,0.077248,0.086552
4000,81,33.549201,0.993481,1.037077,0.123648,0.142341,17.886498,0.981136,1.379715,0.081773,0.090977
4000,82,33.541614,0.994520,1.038363,0.119758,0.138927,17.882999,0.982674,1.389116,0.080636,0.089707
4000,83,33.516171,0.998559,1.042547,0.130925,0.150097,17.885157,0.984784,1.400191,0.085406,0.095075
4000,84,33.514681,0.994423,1.038457,0.120326,0.139378,17.884018,0.983373,1.397447,0.080407,0.089443
4000,85,33.513713,0.999074,1.042810,0.130478,0.150258,17.892746,0.986092,1.403794,0.086298,0.095916
4000,86,33.511091,1.000653,1.044543,0.128457,0.148362,17.884645,0.987847,1.411336,0.085050,0.094500
4000,87,33.514127,1.005374,1.048942,0.139840,0.159445,17.885220,0.989858,1.416806,0.089661,0.099143
4000,88,33.502108,1.001244,1.045241,0.129576,0.148804,17.877498,0.989207,1.424580,0.085219,0.094066
4000,89,33.481633,1.006343,1.049932,0.138287,0.158061,17.870503,0.992122,1.435549,0.090044,0.099527
4000,90,33.495089,1.007368,1.050933,0.134616,0.154593,17.861231,0.993382,1.440827,0.088973,0.098741
4000,91,33.440360,1.011877,1.055401,0.145373,0.165297,17.855666,0.996083,1.441768,0.094124,0.102907
4000,92,33.432526,1.007985,1.051327,0.135994,0.155690,17.855913,0.994729,1.446797,0.089052,0.098633
4000,93,33.422972,1.012819,1.056688,0.145819,0.165299,17.874136,0.997663,1.456658,0.094931,0.103276
```

```
4000,94,33.472084,1.013876,1.058074,0.143744,0.163413,17.858161,0.998855,1.459277,0.093994,0.102153
4000,95,33.550096,1.019170,1.062438,0.155837,0.176119,17.891237,1.001365,1.463654,0.098807,0.106888
4000,96,33.560281,1.014104,1.058048,0.145301,0.165199,17.886949,1.000352,1.465153,0.093199,0.101525
4000,97,33.552997,1.019310,1.062898,0.153667,0.175191,17.884258,1.003032,1.473381,0.098879,0.106999
4000,98,33.522808,1.020435,1.064520,0.151215,0.173496,17.882210,1.004622,1.478311,0.097383,0.105072
4000,99,33.513402,1.024752,1.068280,0.162331,0.185140,17.884937,1.006975,1.483016,0.101592,0.111032
4000,100,33.512076,1.021203,1.064810,0.151656,0.174104,17.883016,1.005821,1.484621,0.097859,0.106158
4000,101,33.514036,1.025986,1.069688,0.160847,0.185193,17.883899,1.008772,1.495000,0.102967,0.112393
4000,102,33.531191,1.027445,1.071057,0.160100,0.182350,17.884402,1.010538,1.503839,0.101618,0.110617
4000,103,33.501902,1.032677,1.076432,0.170994,0.193315,17.870454,1.013163,1.508764,0.106394,0.115669
4000,104,33.515329,1.028751,1.072560,0.159460,0.181510,17.865028,1.011984,1.506632,0.101535,0.110361
4000,105,33.516309,1.033739,1.077545,0.168626,0.191114,17.862611,1.014830,1.514455,0.107299,0.116680
4000,106,33.459654,1.035240,1.078315,0.168102,0.190917,17.857101,1.016578,1.518362,0.105835,0.115170
4000,107,33.477219,1.039972,1.083604,0.178573,0.201109,17.876593,1.019321,1.529917,0.110945,0.120295
4000,108,33.544353,1.036262,1.080010,0.167113,0.189828,17.885255,1.018166,1.537454,0.105601,0.114855
4000,109,33.557282,1.041519,1.084644,0.177663,0.199973,17.886013,1.021205,1.534947,0.112288,0.121264
4000,110,33.559122,1.042836,1.086001,0.175299,0.198923,17.886863,1.023138,1.537158,0.110307,0.119169
4000,111,33.548912,1.047749,1.091108,0.187411,0.210690,17.874770,1.025525,1.543449,0.115490,0.124820
4000,112,33.503399,1.042828,1.086552,0.175950,0.199135,17.879258,1.024730,1.551719,0.110851,0.120294
4000,113,33.508423,1.047765,1.091277,0.185552,0.209853,17.884107,1.027482,1.551700,0.116336,0.125448
4000,114,33.511112,1.049340,1.092876,0.182874,0.206662,17.884172,1.029094,1.562419,0.115110,0.124737
4000,115,33.508488,1.054482,1.098141,0.193449,0.218274,17.884456,1.031837,1.572803,0.120151,0.129713
4000,116,33.508771,1.050040,1.093768,0.182221,0.206976,17.884063,1.030895,1.572333,0.114700,0.123843
4000,117,33.512169,1.055686,1.098948,0.192560,0.218596,17.887111,1.034190,1.579910,0.121879,0.131091
4000,118,33.481044,1.057514,1.100891,0.192196,0.217112,17.877979,1.036472,1.584893,0.119889,0.128784
4000,119,33.472200,1.062776,1.106148,0.202383,0.228001,17.871297,1.039007,1.592302,0.125349,0.134990
4000,120,33.499425,1.057061,1.101061,0.191241,0.216177,17.983635,1.037601,1.599021,0.120184,0.129174
4000,121,33.430589,1.062881,1.106410,0.201954,0.227835,17.855525,1.040764,1.608057,0.125971,0.135436
4000,122,33.407888,1.064083,1.107887,0.199708,0.225495,17.853748,1.042471,1.614050,0.125144,0.134731
4000,123,33.412567,1.069470,1.112882,0.210703,0.236713,17.853867,1.045557,1.615751,0.129833,0.139610
4000,124,33.492463,1.064906,1.108706,0.199227,0.225018,17.877196,1.044245,1.619387,0.125715,0.135501
4000,125,33.523526,1.069629,1.113412,0.209967,0.235777,17.899425,1.047606,1.630294,0.131175,0.140832
4000,126,33.513935,1.070701,1.114115,0.207994,0.234078,17.884854,1.049189,1.633322,0.130547,0.140614
4000,127,33.510064,1.076250,1.119657,0.220102,0.246452,17.884231,1.052108,1.635965,0.134751,0.144131
4000,128,33.514071,1.070581,1.114517,0.207738,0.233792,17.884232,1.050905,1.637822,0.131266,0.141226
4000,129,33.514766,1.076410,1.120293,0.219383,0.247019,17.884761,1.054661,1.646074,0.135746,0.145388
4000,130,33.502204,1.077948,1.121829,0.217794,0.245179,17.883455,1.056372,1.652701,0.135557,0.145820
4000,131,33.470217,1.084797,1.127894,0.229569,0.258295,17.871908,1.059533,1.656011,0.139478,0.149589
4000,132,33.486292,1.079674,1.123885,0.217142,0.244828,17.863162,1.058664,1.660324,0.136052,0.145865
4000,133,33.474493,1.085683,1.129196,0.229623,0.259103,17.868412,1.062215,1.670176,0.141198,0.150547
4000,134,33.424798,1.086676,1.130472,0.227722,0.256043,17.856366,1.064086,1.679216,0.140234,0.150169
4000,135,33.407916,1.092803,1.136490,0.240920,0.270189,17.855305,1.066945,1.682915,0.145321,0.154807
4000,136,33.406547,1.087369,1.131354,0.226955,0.255345,17.854055,1.066300,1.678755,0.140055,0.149576
4000,137,33.431410,1.093872,1.137354,0.237919,0.266225,17.858462,1.069656,1.686238,0.145983,0.155989
4000,138,33.513844,1.094498,1.138181,0.236955,0.265758,17.885237,1.071273,1.691059,0.144092,0.152921
4000,139,33.514461,1.100505,1.143995,0.249304,0.278970,17.884876,1.074466,1.702951,0.150974,0.161195
4000,140,33.514022,1.095773,1.139734,0.234339,0.263644,17.883388,1.073121,1.710325,0.144758,0.154114
4000,141,33.514941,1.101916,1.146074,0.247122,0.275851,17.893835,1.076575,1.707031,0.151439,0.161003
4000,142,33.512954,1.103029,1.147605,0.245769,0.274856,17.884145,1.078858,1.709766,0.149523,0.158992
4000,143,33.510295,1.109355,1.153111,0.260433,0.291193,17.884770,1.081868,1.716564,0.154959,0.164741
4000,144,33.514712,1.103620,1.147709,0.243375,0.272763,17.883693,1.081086,1.725204,0.150716,0.160500
4000,145,33.506420,1.110552,1.154412,0.255169,0.285794,17.882139,1.085076,1.724166,0.155925,0.166148
4000,146,33.468717,1.112166,1.156025,0.253021,0.283901,17.869691,1.087375,1.735972,0.155241,0.165119
4000,147,33.490294,1.119019,1.162211,0.266778,0.298129,17.860339,1.090859,1.746359,0.160258,0.170673
4000,148,33.426448,1.112965,1.156754,0.251626,0.282948,17.854875,1.089710,1.745275,0.154699,0.164160
4000,149,33.411669,1.119849,1.163196,0.265112,0.297076,17.859762,1.093531,1.752712,0.163273,0.173234
4000,150,33.462491,1.120653,1.164458,0.264150,0.295733,17.856922,1.095530,1.757663,0.160327,0.169808
4000,151,33.527957,1.127648,1.171183,0.278142,0.310562,17.879512,1.098995,1.765811,0.166131,0.175834
4000,152,33.513484,1.120735,1.164988,0.262412,0.294155,17.884597,1.097890,1.773253,0.161880,0.171527
4000,153,33.512973,1.127817,1.171591,0.274145,0.305261,17.884436,1.101577,1.780958,0.167155,0.176978
4000,154,33.513363,1.128731,1.172769,0.272209,0.303450,17.884112,1.103431,1.788432,0.165971,0.175348
4000,155,33.510711,1.135621,1.179272,0.286451,0.319185,17.884845,1.106839,1.791230,0.172904,0.183202
4000,156,33.510600,1.129921,1.174014,0.270409,0.302807,17.884079,1.105951,1.794551,0.165609,0.175498
4000,157,33.509919,1.137369,1.180652,0.283950,0.316292,17.898269,1.109791,1.804093,0.172956,0.182469
4000,158,33.495712,1.139057,1.182601,0.282942,0.315327,17.878866,1.112244,1.808843,0.172534,0.181752
4000,159,33.481024,1.146593,1.189713,0.297951,0.331328,17.869148,1.115880,1.814400,0.178203,0.187660
4000,160,33.509934,1.138878,1.183082,0.282581,0.314916,17.860931,1.114723,1.816186,0.171971,0.181416
4000,161,33.452593,1.146371,1.189885,0.296884,0.329862,17.856580,1.119054,1.820946,0.179406,0.189906
```

```
4000,162,33.427652,1.147366,1.191104,0.293178,0.326944,17.854955,1.121061,1.826626,0.177008,0.186938
4000,163,33.429317,1.154600,1.197914,0.306114,0.341219,17.854835,1.124872,1.833074,0.182759,0.193049
4000,164,33.548809,1.149887,1.194065,0.290992,0.325099,17.885456,1.123422,1.835829,0.178773,0.187966
4000,165,33.564974,1.157445,1.200638,0.304300,0.340327,17.888398,1.128147,1.847280,0.185109,0.194535
4000,166,33.553130,1.158554,1.202466,0.303100,0.337704,17.886614,1.130643,1.856373,0.182739,0.192824
4000,167,33.511569,1.165091,1.208390,0.318416,0.353580,17.884253,1.133840,1.860990,0.190175,0.200146
4000,168,33.508909,1.158091,1.202183,0.301626,0.336252,17.884290,1.132474,1.855212,0.182996,0.192459
4000,169,33.514307,1.166195,1.209372,0.315621,0.350578,17.884370,1.137146,1.863273,0.190515,0.200630
4000,170,33.515537,1.167309,1.211122,0.315026,0.349960,17.884179,1.139423,1.868636,0.188367,0.198612
4000,171,33.512682,1.174699,1.218108,0.326487,0.361423,17.884204,1.142766,1.880563,0.196207,0.205395
4000,172,33.513490,1.169714,1.213393,0.312407,0.347257,17.883631,1.141353,1.888455,0.188641,0.198377
4000,173,33.508079,1.177691,1.221214,0.326840,0.362069,17.883960,1.146510,1.885640,0.196829,0.204888
4000,174,33.517273,1.178989,1.222902,0.323324,0.358931,17.876152,1.149152,1.888514,0.195638,0.203742
4000,175,33.503299,1.186483,1.229605,0.337290,0.374931,17.865966,1.152671,1.896135,0.201066,0.210110
4000,176,33.521762,1.179071,1.222479,0.321693,0.357713,17.864149,1.151430,1.904472,0.194962,0.204253
4000,177,33.485667,1.186788,1.230052,0.335218,0.371852,17.860068,1.156001,1.904921,0.201379,0.211047
4000,178,33.483898,1.188050,1.231594,0.332186,0.369060,17.875985,1.158684,1.915874,0.200984,0.210792
4000,179,33.529585,1.195353,1.238574,0.348432,0.385875,17.886105,1.161973,1.926915,0.205853,0.216281
4000,180,33.558057,1.189117,1.232705,0.331206,0.368225,17.886574,1.160468,1.925121,0.200720,0.209988
4000,181,33.564085,1.196118,1.239787,0.349159,0.386594,17.891240,1.166282,1.934369,0.208034,0.217462
4000,182,33.561307,1.197593,1.241446,0.345139,0.382509,17.886929,1.168615,1.940532,0.207404,0.217115
4000,183,33.528265,1.204529,1.247916,0.359075,0.398122,17.884929,1.171685,1.948104,0.212866,0.222651
4000,184,33.512703,1.197527,1.241596,0.342543,0.379890,17.883962,1.169953,1.953306,0.207012,0.216616
4000,185,33.513994,1.205130,1.248986,0.357667,0.395773,17.884633,1.174538,1.961327,0.214233,0.223573
4000,186,33.512054,1.206469,1.250229,0.353172,0.392407,17.884276,1.177279,1.969086,0.213285,0.223049
4000,187,33.524384,1.214977,1.258411,0.369790,0.408506,17.881893,1.180745,1.974227,0.219206,0.228911
4000,188,33.501437,1.208480,1.252123,0.350968,0.389902,17.868332,1.180142,1.977662,0.212692,0.221966
4000,189,33.517146,1.217630,1.261011,0.367942,0.406182,17.872281,1.185425,1.986724,0.220526,0.229685
4000,190,33.518614,1.218567,1.262388,0.365444,0.404110,17.861723,1.187577,1.991073,0.219430,0.228553
4000,191,33.479579,1.227109,1.269734,0.379911,0.420077,17.860936,1.190486,1.996888,0.225087,0.234537
4000,192,33.531528,1.219806,1.263202,0.364440,0.402850,17.884990,1.190615,1.999585,0.219089,0.228878
4000,193,33.552285,1.227506,1.270727,0.382292,0.421713,17.885484,1.195857,2.006079,0.226851,0.236213
4000,194,33.559586,1.228930,1.272517,0.380526,0.420647,17.887072,1.197549,2.011355,0.224823,0.234704
4000,195,33.561603,1.236947,1.279550,0.394226,0.434238,17.887478,1.201449,2.017121,0.231384,0.241717
4000,196,33.560098,1.230934,1.274662,0.377878,0.418406,17.886311,1.200794,2.016939,0.225009,0.234289
4000,197,33.539909,1.238886,1.282138,0.393293,0.434176,17.883653,1.206061,2.031917,0.232705,0.242161
4000,198,33.514235,1.239308,1.282813,0.389392,0.429755,17.885986,1.207121,2.038514,0.230920,0.240515
4000,199,33.517407,1.247284,1.290882,0.405206,0.445790,17.886112,1.211718,2.044533,0.237742,0.247519
4000,200,33.521220,1.240582,1.284312,0.387330,0.427786,17.886179,1.210096,2.039807,0.232174,0.241271
```

B.2.5 Positive Definite Arrow Matrix - $n = 200 : 4000$ and $b = 100 : 2000$

```
POSDEF ARROW MATRIX TEST
 N,B,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SPOTRF,ASD1_SPOTRF,ASD2_SPOTRF
200,100,0.007265,0.007236,0.008027,0.004056,0.004040,0.004612
400,100,0.044827,0.044851,0.038514,0.024006,0.021573,0.022625
400,200,0.044886,0.044785,0.047358,0.024027,0.023909,0.025959
600,100,0.136896,0.136753,0.075488,0.072985,0.052010,0.048035
600,200,0.136799,0.136774,0.125996,0.073169,0.068351,0.070552
600,300,0.136832,0.136778,0.140860,0.073090,0.072983,0.076253
800,100,0.309151,0.309569,0.117684,0.164402,0.095922,0.080900
800,200,0.309474,0.309138,0.210987,0.164371,0.133417,0.127483
800,300,0.309476,0.309059,0.292555,0.164462,0.160200,0.163586
800,400,0.309279,0.309489,0.314900,0.164390,0.164261,0.168924
1000,100,0.585806,0.585914,0.165850,0.309697,0.142703,0.112899
1000,200,0.585848,0.585966,0.302193,0.309227,0.216893,0.195367
1000,300,0.585995,0.585753,0.451453,0.309769,0.274814,0.269698
1000,400,0.585919,0.585763,0.562298,0.309374,0.302727,0.308012
1000,500,0.585970,0.585954,0.593542,0.309721,0.309192,0.315510
1200,100,1.008105,1.007924,0.213827,0.525138,0.209992,0.160582
1200,200,1.008165,1.008034,0.399212,0.525329,0.321961,0.276044
1200,300,1.008008,1.007893,0.618797,0.525348,0.399557,0.374340
```

```
1200,400,1.007984,1.008005,0.824050,0.525398,0.476514,0.473790
1200,500,1.008169,1.007784,0.977145,0.525136,0.520475,0.527729
1200,600,1.007948,1.007957,1.017661,0.525022,0.525045,0.532718
1400,100,1.547289,1.547314,0.274503,0.816180,0.288774,0.214318
1400,200,1.547817,1.547346,0.512179,0.816306,0.445069,0.364781
1400,300,1.547555,1.547254,0.775989,0.815889,0.559281,0.500541
1400,400,1.547435,1.547249,1.058562,0.816025,0.679978,0.654881
1400,500,1.547345,1.547214,1.328350,0.816110,0.752566,0.751479
1400,600,1.547408,1.546995,1.503495,0.815842,0.807715,0.816991
1400,700,1.547557,1.547447,1.559140,0.815012,0.815939,0.824913
1600,100,2.302655,2.302168,0.328599,1.205283,0.380846,0.276490
1600,200,2.302363,2.302072,0.614945,1.205359,0.589992,0.466112
1600,300,2.302399,2.302110,0.948610,1.205022,0.746950,0.643260
1600,400,2.302050,2.302071,1.322733,1.204921,0.921030,0.856231
1600,500,2.302526,2.301837,1.710465,1.204935,1.035176,1.012075
1600,600,2.302237,2.301814,2.027487,1.205202,1.142551,1.145448
1600,700,2.302443,2.301957,2.274911,1.205142,1.193823,1.204629
1600,800,2.302246,2.301965,2.315653,1.205215,1.205154,1.216819
1800,100,3.221581,3.220687,0.404439,1.695750,0.490734,0.355034
1800,200,3.221573,3.220391,0.731070,1.695873,0.723770,0.551692
1800,300,3.221188,3.220609,1.126659,1.695830,0.963609,0.802400
1800,400,3.221664,3.220536,1.585885,1.695650,1.194134,1.074877
1800,500,3.221308,3.220600,2.086586,1.696109,1.359435,1.291289
1800,600,3.220510,3.221533,2.526677,1.695716,1.523694,1.501114
1800,700,3.221293,3.220894,2.958521,1.695804,1.619076,1.623072
1800,800,3.220994,3.221054,3.163623,1.696040,1.684399,1.697054
1800,900,3.221951,3.221067,3.237368,1.695742,1.695615,1.709149
2000,100,4.359022,4.358886,0.462189,2.304125,0.603516,0.426606
2000,200,4.359016,4.358477,0.836523,2.303638,0.897537,0.662589
2000,300,4.358917,4.357551,1.295072,2.304119,1.198733,0.965404
2000,400,4.358855,4.358620,1.836815,2.303870,1.444751,1.251629
2000,500,4.358716,4.358283,2.446008,2.303956,1.716881,1.582865
2000,600,4.358589,4.357550,3.024334,2.303677,1.946972,1.876988
2000,700,4.358457,4.357940,3.639360,2.303883,2.093662,2.075096
2000,800,4.358683,4.357909,4.011383,2.303670,2.224875,2.230847
2000,900,4.358858,4.358415,4.321376,2.303462,2.300225,2.313145
2000,1000,4.358960,4.358144,4.376309,2.303723,2.302771,2.318122
2200,100,5.737554,5.735345,0.549196,3.044550,0.743892,0.524063
2200,200,5.734499,5.733694,0.944369,3.044187,1.099152,0.795711
2200,300,5.732773,5.731578,1.544057,3.043895,1.468404,1.154291
2200,400,5.733016,5.732518,2.114131,3.043147,1.778718,1.496123
2200,500,5.734245,5.731851,2.840908,3.042568,2.122132,1.900984
2200,600,5.732784,5.731451,3.536558,3.042348,2.361484,2.223417
2200,700,5.726349,5.723475,4.332948,3.043017,2.631876,2.566286
2200,800,5.722155,5.720404,4.851214,3.042787,2.829006,2.810943
2200,900,5.720787,5.719825,5.388204,3.042276,2.964492,2.968554
2200,1000,5.720834,5.719270,5.643534,3.042859,3.042440,3.054989
2200,1100,5.720362,5.718253,5.739416,3.042529,3.042074,3.060262
2400,100,7.412580,7.410843,0.608480,3.937051,0.878374,0.607233
2400,200,7.412465,7.411551,1.049760,3.936659,1.312047,0.925909
2400,300,7.417269,7.430923,1.722306,3.937111,1.761037,1.346482
2400,400,7.443456,7.441835,2.385179,3.939680,2.141065,1.748568
2400,500,7.443702,7.441409,3.232626,3.939720,2.565775,2.237474
2400,600,7.441926,7.439465,4.080021,3.940047,2.875757,2.641391
2400,700,7.431782,7.430276,5.065915,3.939685,3.227035,3.086916
2400,800,7.431393,7.430654,5.757148,3.939482,3.437875,3.373980
2400,900,7.431674,7.430488,6.537093,3.939750,3.709999,3.691764
2400,1000,7.431602,7.430004,6.998247,3.939732,3.867834,3.871785
2400,1100,7.432544,7.430052,7.403389,3.939217,3.935075,3.952985
2400,1200,7.431941,7.430356,7.453583,3.939756,3.939154,3.958651
2600,100,9.368272,9.365131,0.714633,4.977722,1.017933,0.696608
2600,200,9.366979,9.366405,1.193240,4.977097,1.553740,1.088406
2600,300,9.365570,9.362492,1.937552,4.976850,2.083409,1.566891
2600,400,9.356987,9.351041,2.673090,4.974949,2.539089,2.028631
2600,500,9.350784,9.348963,3.635220,4.973697,3.050475,2.595225
2600,600,9.349960,9.349536,4.599573,4.972546,3.426553,3.077037
2600,700,9.348131,9.346382,5.763369,4.972830,3.867959,3.621019
2600,800,9.340678,9.322644,6.597074,4.973210,4.140624,4.002033
```

```
2600,900,9.322823,9.338720,7.617039,4.971397,4.502942,4.439343
2600,1000,9.376104,9.375237,8.287751,4.977082,4.694599,4.674734
2600,1100,9.378187,9.376827,9.014945,4.977172,4.883063,4.895064
2600,1200,9.381239,9.380071,9.294888,4.975898,4.989008,5.008008
2600,1300,9.361393,9.358144,9.382298,4.974583,4.974334,4.997553
2800,100,11.673057,11.667543,0.777398,6.193313,1.165389,0.785195
2800,200,11.678377,11.678590,1.308209,6.199704,1.796953,1.234131
2800,300,11.679353,11.676987,2.119284,6.199325,2.343034,1.703732
2800,400,11.679023,11.680374,3.064340,6.199666,2.960148,2.320640
2800,500,11.682624,11.683651,4.020379,6.199112,3.569861,2.969097
2800,600,11.689323,11.679734,5.127374,6.195258,4.028995,3.535701
2800,700,11.673475,11.672167,6.488629,6.194047,4.566294,4.194226
2800,800,11.673765,11.670062,7.500736,6.195729,4.913290,4.669241
2800,900,11.678308,11.678295,8.758259,6.197166,5.381857,5.246905
2800,1000,11.677690,11.678613,9.613204,6.196586,5.635759,5.567816
2800,1100,11.678576,11.677677,10.643600,6.198806,5.926815,5.916875
2800,1200,11.680962,11.679733,11.165757,6.198872,6.083829,6.100785
2800,1300,11.685466,11.683775,11.647024,6.195237,6.175201,6.198723
2800,1400,11.659208,11.656932,11.685168,6.193954,6.192471,6.217079
3000,100,14.229414,14.224859,0.898392,7.594420,1.368291,0.927613
3000,200,14.222950,14.217548,1.464413,7.593294,2.089265,1.427422
3000,300,14.194159,14.172353,2.346994,7.591232,2.729665,1.953808
3000,400,14.157344,14.151175,3.386617,7.589337,3.437997,2.644343
3000,500,14.147962,14.148981,4.444790,7.590794,4.038823,3.274496
3000,600,14.148902,14.147973,5.668919,7.589518,4.688537,4.028061
3000,700,14.158120,14.206448,7.231920,7.590904,5.322602,4.787294
3000,800,14.302749,14.304689,8.414052,7.602960,5.763753,5.377335
3000,900,14.307813,14.305875,9.901178,7.603599,6.324935,6.067061
3000,1000,14.289491,14.281535,10.922077,7.602931,6.646139,6.505819
3000,1100,14.283520,14.281279,12.271667,7.602776,7.035986,6.986390
3000,1200,14.284571,14.281694,13.000819,7.602638,7.284851,7.269227
3000,1300,14.283965,14.281216,13.888835,7.602460,7.481270,7.498959
3000,1400,14.282755,14.282441,14.173045,7.603154,7.572971,7.600503
3000,1500,14.282671,14.280159,14.311609,7.602544,7.602088,7.629568
3200,100,17.340790,17.318763,0.957555,9.223207,1.541068,1.023687
3200,200,17.307725,17.305132,1.572731,9.212947,2.376596,1.586167
3200,300,17.266212,17.241586,2.490670,9.207591,3.112589,2.184208
3200,400,17.234503,17.234667,3.672656,9.207242,3.935845,2.969083
3200,500,17.290252,17.365832,4.844398,9.223220,4.638077,3.685178
3200,600,17.379289,17.377567,6.240384,9.228097,5.400082,4.541014
3200,700,17.370564,17.362389,7.978257,9.225678,6.018263,5.283584
3200,800,17.344759,17.345562,9.300826,9.226312,6.671797,6.110716
3200,900,17.343244,17.343292,11.032764,9.226489,7.351961,6.933772
3200,1000,17.345153,17.343844,12.294865,9.226489,7.764645,7.508821
3200,1100,17.343172,17.343881,13.945811,9.226088,8.265245,8.138520
3200,1200,17.345944,17.340675,14.907444,9.220003,8.576821,8.527689
3200,1300,17.315982,17.321810,16.182734,9.214091,8.889083,8.882034
3200,1400,17.314013,17.311812,16.684795,9.212400,9.059783,9.079565
3200,1500,17.286475,17.316226,17.322658,9.226118,9.224715,9.255794
3200,1600,17.372081,17.378139,17.411026,9.227519,9.226681,9.256933
3400,100,20.688563,20.663429,1.110800,11.019270,1.775328,1.197372
3400,200,20.663735,20.671211,1.759691,11.018406,2.643724,1.747103
3400,300,20.708243,20.715949,2.701192,11.031190,3.554953,2.464397
3400,400,20.726349,20.729280,4.012994,11.030926,4.483339,3.334874
3400,500,20.736210,20.692985,5.337498,11.021954,5.284033,4.132812
3400,600,20.662853,20.649952,6.799442,11.017544,6.144628,5.084826
3400,700,20.639358,20.593836,8.700940,11.009424,6.858152,5.928282
3400,800,20.545459,20.538593,10.187373,11.009146,7.621294,6.862889
3400,900,20.708063,20.724086,12.210758,11.031875,8.296530,7.701976
3400,1000,20.758157,20.754878,13.638927,11.031132,8.937796,8.526271
3400,1100,20.726781,20.715902,15.605649,11.031657,9.562861,9.305879
3400,1200,20.725445,20.720159,16.784603,11.030453,9.970011,9.845324
3400,1300,20.717814,20.714784,18.466583,11.030499,10.387554,10.335394
3400,1400,20.718119,20.715194,19.188216,11.030790,10.652613,10.649144
3400,1500,20.706052,20.698644,20.233744,11.030975,10.924270,10.946024
3400,1600,20.686975,20.649422,20.517317,11.017276,10.984621,11.017408
3400,1700,20.623335,20.606103,20.590104,11.009409,11.007784,11.040993
3600,100,24.384130,24.378427,1.164714,13.027244,1.956502,1.299108
```

```
3600,200,24.382623,24.379198,1.868640,13.026965,2.942072,1.913272
3600,300,24.439143,24.503627,2.877825,13.027346,3.974209,2.713927
3600,400,24.588576,24.579011,4.295203,13.058693,4.907934,3.554110
3600,500,24.582012,24.579534,6.096388,13.058481,5.936342,4.596044
3600,600,24.582204,24.576974,7.355930,13.059894,6.929758,5.641745
3600,700,24.580020,24.577146,9.462708,13.058812,7.763167,6.586727
3600,800,24.568432,24.538015,11.086076,13.039587,8.636398,7.648883
3600,900,24.516485,24.498030,13.262727,13.033106,9.402955,8.594797
3600,1000,24.579438,24.598098,14.971773,13.058998,10.177150,9.562959
3600,1100,24.629014,24.624977,17.277690,13.059375,10.758513,10.337101
3600,1200,24.601363,24.584578,18.661302,13.047852,11.424542,11.176983
3600,1300,24.575149,24.576045,20.739435,13.058876,11.972668,11.842178
3600,1400,24.592039,24.594776,21.689896,13.040264,12.306050,12.258267
3600,1500,24.506119,24.440632,23.112994,13.031508,12.704356,12.722758
3600,1600,24.600251,24.626466,23.898523,13.058106,12.886914,12.911843
3600,1700,24.573814,24.543206,24.516743,13.041870,13.076558,13.119428
3600,1800,24.541962,24.555110,24.619448,13.058811,13.056661,13.091807
3800,100,28.837893,28.849219,1.327595,15.350269,2.238296,1.496997
3800,200,28.863998,28.757833,2.066564,15.328856,3.333144,2.170431
3800,300,28.732865,28.718404,3.141508,15.324485,4.487733,3.041418
3800,400,28.807192,28.838853,4.646975,15.349213,5.539367,3.971185
3800,500,28.879205,28.848432,6.465389,15.344491,6.710348,5.117080
3800,600,28.775744,28.752764,7.931718,15.328848,7.647673,6.093134
3800,700,28.710421,28.761212,10.255420,15.349640,8.771587,7.309660
3800,800,28.870717,28.887123,12.050899,15.350820,9.773182,8.503683
3800,900,28.890161,28.828209,14.466604,15.336359,10.629668,9.563311
3800,1000,28.829806,28.834493,16.286063,15.350349,11.525494,10.675358
3800,1100,28.826853,28.829460,18.919376,15.346996,12.205397,11.583449
3800,1200,28.753125,28.739580,20.495207,15.329281,13.004522,12.589515
3800,1300,28.734896,28.710469,22.869911,15.318704,13.506063,13.257475
3800,1400,28.828451,28.848877,24.219744,15.349862,14.144258,14.007145
3800,1500,28.892969,28.889917,26.243562,15.350566,14.667519,14.627581
3800,1600,28.825714,28.786329,27.066792,15.349460,14.926144,14.928588
3800,1700,28.835440,28.839294,28.326430,15.347871,15.346109,15.360935
3800,1800,28.740872,28.679210,28.433748,15.428605,15.394874,15.435628
3800,1900,28.553787,28.548786,28.605312,15.429142,15.428179,15.468110
4000,100,33.557310,33.623119,1.386550,17.905969,2.446992,1.604526
4000,200,33.559237,33.553979,2.173355,17.902730,3.675830,2.346973
4000,300,33.561141,33.551046,3.327395,17.902921,4.964842,3.311911
4000,400,33.538790,33.440497,4.941317,17.877997,6.140912,4.332666
4000,500,33.364009,33.258916,6.856707,17.858104,7.444906,5.575496
4000,600,33.258802,33.294127,8.500015,17.894617,8.546954,6.684213
4000,700,33.574469,33.552314,10.970009,17.903011,9.809164,8.045327
4000,800,33.558223,33.553911,12.935632,18.043966,10.777932,9.183755
4000,900,33.542763,33.453111,15.558911,18.005131,11.940259,10.547519
4000,1000,33.333442,33.255773,17.534232,17.994409,12.989447,11.854246
4000,1100,33.260812,33.526010,20.613557,17.903038,13.791654,12.913537
4000,1200,33.563938,33.554652,22.427647,17.903010,14.720799,14.089792
4000,1300,33.579838,33.499270,25.200843,17.869922,15.307913,14.898931
4000,1400,33.422551,33.370387,26.635905,17.895732,16.084882,15.844090
4000,1500,33.637864,33.630388,29.200772,17.904308,16.660997,16.542346
4000,1600,33.559866,33.554332,30.294096,18.039820,17.216943,17.179470
4000,1700,33.555935,33.553763,32.056536,18.024050,17.645883,17.642907
4000,1800,33.424190,33.356462,32.455669,17.994602,17.812678,17.848410
4000,1900,33.263608,33.428096,33.511122,18.031603,18.043413,18.070143
4000,2000,33.567331,33.555080,33.600883,18.040726,18.037608,18.084755
```

B.2.6 Positive Definite Profile Matrix - $n = 200 : 4000$ and $b = 100 : 2000$

```
POSDEF PROFILE MATRIX TEST
 N,B,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SGBTRF,ASD1_SGBTRF,SPOTRF,ASD1_SPOTRF,ASD2_SPOTRF,SPBTRF,ASD1_SPBTRF
200,100,0.007292,0.004776,0.005559,0.004931,0.004418,0.004082,0.003327,0.003881,0.003229,0.003190
```

```
400,100,0.044829,0.014513,0.016920,0.012571,0.011204,0.024095,0.010465,0.013206,0.007863,0.007394
400,200,0.044836,0.022857,0.025377,0.026795,0.021307,0.024098,0.016130,0.018667,0.015971,0.013083
600,100,0.135565,0.029299,0.032670,0.020242,0.018126,0.073405,0.021516,0.027134,0.012557,0.011679
600,200,0.135550,0.041203,0.045033,0.046908,0.036011,0.073630,0.028261,0.036292,0.027835,0.021038
600,300,0.135537,0.059548,0.063368,0.077679,0.057248,0.073527,0.043008,0.050052,0.043382,0.031854
800,100,0.302123,0.045764,0.051236,0.028061,0.025320,0.164108,0.037196,0.046164,0.017663,0.016565
800,200,0.301962,0.062870,0.068291,0.067053,0.051512,0.164301,0.048573,0.062564,0.039742,0.029950
800,300,0.302217,0.080758,0.086391,0.116905,0.078701,0.164067,0.060062,0.078030,0.065146,0.041842
800,400,0.302181,0.119200,0.124812,0.168822,0.116046,0.164049,0.089810,0.105611,0.092246,0.063487
1000,100,0.563533,0.073611,0.079881,0.035913,0.032711,0.309096,0.058007,0.073185,0.022487,0.021206
1000,200,0.563398,0.095051,0.101973,0.087293,0.067263,0.308931,0.070811,0.095868,0.051947,0.038796
1000,300,0.563667,0.131285,0.138466,0.156276,0.114858,0.309133,0.093355,0.123889,0.086986,0.061268
1000,400,0.563414,0.150482,0.157796,0.233436,0.142985,0.308713,0.111923,0.147977,0.126670,0.075667
1000,500,0.563131,0.211816,0.219262,0.314389,0.207609,0.308943,0.161276,0.192923,0.166887,0.109474
1200,100,0.955707,0.098662,0.107690,0.043543,0.039883,0.522241,0.082604,0.104692,0.027513,0.026011
1200,200,0.955811,0.125102,0.134234,0.107680,0.082990,0.522324,0.099579,0.136971,0.063986,0.047699
1200,300,0.955749,0.161525,0.170491,0.195350,0.136765,0.522101,0.126050,0.173562,0.108939,0.072031
1200,400,0.955714,0.210993,0.220956,0.297943,0.194311,0.522176,0.150449,0.207585,0.161407,0.101465
1200,500,0.955706,0.244138,0.253465,0.409650,0.240292,0.523026,0.187763,0.250730,0.217135,0.122139
1200,600,0.955459,0.336573,0.345998,0.528122,0.338173,0.522287,0.261430,0.316891,0.278087,0.176422
1400,100,1.500733,0.138676,0.147536,0.051094,0.046709,0.814694,0.112699,0.143011,0.032413,0.030615
1400,200,1.500262,0.170569,0.180583,0.127766,0.097581,0.814900,0.131402,0.184305,0.076024,0.055907
1400,300,1.500524,0.206275,0.216870,0.234158,0.157708,0.813541,0.157517,0.228163,0.130779,0.081527
1400,400,1.500654,0.283384,0.294149,0.362261,0.245561,0.813789,0.209102,0.288094,0.195649,0.128986
1400,500,1.500680,0.317524,0.328512,0.504905,0.292906,0.813882,0.231501,0.326205,0.267595,0.146613
1400,600,1.500644,0.380452,0.391533,0.661520,0.376980,0.813779,0.292722,0.393570,0.347614,0.191545
1400,700,1.500571,0.506783,0.518076,0.811896,0.508237,0.814485,0.396355,0.486049,0.422203,0.260275
1600,100,2.222819,0.167293,0.179527,0.058813,0.053717,1.200800,0.147007,0.185973,0.037330,0.034996
1600,200,2.222969,0.202578,0.214959,0.147678,0.112828,1.200770,0.169646,0.238136,0.088209,0.064593
1600,300,2.222935,0.260445,0.272819,0.273060,0.192903,1.200551,0.204559,0.296989,0.152472,0.100667
1600,400,2.223090,0.318084,0.330516,0.426147,0.272563,1.200591,0.252117,0.362178,0.229867,0.141028
1600,500,2.222960,0.420037,0.432801,0.599173,0.378916,1.200706,0.303672,0.428557,0.317225,0.191261
1600,600,2.223128,0.449019,0.461436,0.794733,0.437177,1.200433,0.341257,0.484795,0.417323,0.218726
1600,700,2.223159,0.545523,0.558662,0.987025,0.554754,1.200317,0.432720,0.581358,0.513301,0.276898
1600,800,2.223321,0.717363,0.730651,1.194353,0.730016,1.200272,0.569876,0.703533,0.622805,0.376415
1800,100,3.142658,0.222928,0.234055,0.066547,0.060920,1.689227,0.189954,0.241283,0.042460,0.039796
1800,200,3.142118,0.264704,0.277870,0.168019,0.128479,1.689398,0.214189,0.304730,0.099924,0.073226
1800,300,3.142471,0.318810,0.332023,0.311708,0.214462,1.689195,0.254413,0.376619,0.174084,0.111027
1800,400,3.142718,0.378656,0.392669,0.489651,0.299150,1.689289,0.295996,0.447181,0.264517,0.153444
1800,500,3.142942,0.506502,0.520959,0.694080,0.436816,1.689092,0.387559,0.549383,0.367762,0.220326
1800,600,3.142710,0.590838,0.605420,0.927684,0.542982,1.688650,0.422021,0.610656,0.486562,0.271013
1800,700,3.143454,0.635263,0.650385,1.161454,0.623641,1.689047,0.487447,0.696217,0.602933,0.306040
1800,800,3.142635,0.774472,0.789826,1.420886,0.782898,1.689011,0.611897,0.823167,0.740058,0.396118
1800,900,3.143603,0.989761,1.005428,1.675542,1.010839,1.688707,0.786750,0.978936,0.864753,0.514097
2000,100,4.286235,0.257255,0.273410,0.074284,0.068139,2.297353,0.231340,0.295984,0.047303,0.044407
2000,200,4.286836,0.302280,0.318473,0.188181,0.144001,2.297063,0.259252,0.372540,0.112075,0.082210
2000,300,4.286952,0.354827,0.371305,0.350459,0.234480,2.297026,0.295266,0.452303,0.196427,0.120612
2000,400,4.287178,0.455115,0.471642,0.553587,0.349539,2.297015,0.350603,0.541870,0.299330,0.179097
2000,500,4.287483,0.547032,0.563133,0.788235,0.470100,2.296706,0.436210,0.652983,0.416974,0.234236
2000,600,4.287067,0.715500,0.732484,1.060725,0.655561,2.297132,0.528357,0.763528,0.556120,0.329093
2000,700,4.286809,0.766724,0.783668,1.335904,0.736549,2.296444,0.566393,0.836398,0.693273,0.361247
2000,800,4.287223,0.853474,0.871082,1.646444,0.858004,2.296489,0.664763,0.955769,0.857074,0.428189
2000,900,4.286898,1.032028,1.049881,1.955371,1.070291,2.296899,0.827867,1.121775,1.007893,0.533240
2000,1000,4.287337,1.310706,1.328613,2.282242,1.353218,2.296552,1.047254,1.314133,1.184658,0.695124
2200,100,5.661046,0.325913,0.339036,0.081918,0.075340,3.037299,0.291006,0.368651,0.052411,0.049352
2200,200,5.662318,0.373869,0.389648,0.208256,0.158726,3.036912,0.319260,0.458958,0.124288,0.090219
2200,300,5.662527,0.444410,0.461118,0.388683,0.268255,3.036767,0.366694,0.558584,0.217585,0.138857
2200,400,5.662483,0.554859,0.572167,0.617201,0.400643,3.037026,0.437972,0.670684,0.333583,0.206538
2200,500,5.662690,0.614737,0.632431,0.882969,0.502276,3.036667,0.494549,0.773872,0.467993,0.247218
2200,600,5.663888,0.813076,0.831034,1.193353,0.719151,3.036376,0.634855,0.928342,0.624743,0.360425
2200,700,5.663659,0.974697,0.993233,1.510480,0.901609,3.036052,0.699031,1.030121,0.783549,0.444649
2200,800,5.663169,1.011448,1.030541,1.871665,0.979166,3.035753,0.760301,1.130274,0.974060,0.486838
2200,900,5.664324,1.123533,1.142962,2.235457,1.148596,3.036702,0.894666,1.289416,1.152181,0.565301
2200,1000,5.663084,1.376981,1.396226,2.627055,1.418210,3.036724,1.101165,1.494092,1.362296,0.718119
2200,1100,5.663924,1.711929,1.732501,3.002052,1.756748,3.036091,1.369813,1.729175,1.535479,0.889902
2400,100,7.323555,0.357284,0.376467,0.089584,0.082269,3.929268,0.336242,0.427834,0.057451,0.053788
2400,200,7.324741,0.410769,0.429801,0.228380,0.173695,3.928858,0.368929,0.533543,0.136139,0.098938
2400,300,7.332465,0.482124,0.501106,0.427194,0.289446,3.928710,0.419415,0.647632,0.239704,0.149634
```

```
2400,400,7.351567,0.595051,0.614386,0.680703,0.427188,3.928768,0.492438,0.774368,0.368717,0.218737
2400,500,7.362667,0.681778,0.701308,0.976885,0.555425,3.928354,0.553985,0.893024,0.517807,0.272170
2400,600,7.371887,0.855859,0.875900,1.326408,0.759909,3.931834,0.690784,1.063743,0.694605,0.376477
2400,700,7.373339,1.100815,1.120652,1.685419,1.020591,3.931228,0.838846,1.235176,0.873786,0.504121
2400,800,7.372721,1.228986,1.249901,2.098123,1.172823,3.931266,0.885758,1.334913,1.090439,0.582648
2400,900,7.372724,1.244197,1.265790,2.516392,1.270739,3.930974,0.980798,1.479221,1.296172,0.621693
2400,1000,7.370993,1.449914,1.470956,2.973652,1.504749,3.931436,1.163851,1.684075,1.538711,0.753775
2400,1100,7.368965,1.763871,1.785385,3.417311,1.831203,3.931069,1.421380,1.938168,1.746728,0.915781
2400,1200,7.367822,2.180096,2.202053,3.907287,2.243648,3.931424,1.743804,2.220618,1.992373,1.145258
2600,100,9.326075,0.445290,0.459965,0.097353,0.089249,4.968997,0.411327,0.521297,0.062415,0.058518
2600,200,9.325065,0.503406,0.522249,0.248238,0.189135,4.968811,0.445434,0.645004,0.148267,0.107596
2600,300,9.324793,0.573364,0.593366,0.465980,0.310460,4.969918,0.495271,0.772797,0.261885,0.159824
2600,400,9.324219,0.684540,0.705681,0.744808,0.453564,4.968910,0.567505,0.913450,0.403034,0.230962
2600,500,9.325354,0.834288,0.855702,1.071640,0.642334,4.969018,0.664500,1.070740,0.567608,0.317341
2600,600,9.325223,0.943311,0.965085,1.458898,0.798798,4.969687,0.762984,1.230555,0.764073,0.392143
2600,700,9.326828,1.216900,1.239419,1.859321,1.090186,4.968540,0.961200,1.452107,0.963300,0.535582
2600,800,9.324403,1.470319,1.493530,2.324760,1.369483,4.969781,1.072422,1.610739,1.207209,0.683494
2600,900,9.325536,1.475540,1.498898,2.797399,1.459541,4.968816,1.125606,1.734091,1.440379,0.713538
2600,1000,9.324818,1.612217,1.635996,3.318708,1.634972,4.969829,1.269825,1.923866,1.716551,0.814471
2600,1100,9.325137,1.875227,1.898461,3.829391,1.924876,4.969771,1.499968,2.177734,1.952849,0.953039
2600,1200,9.325554,2.259979,2.284188,4.400093,2.321285,4.968266,1.807497,2.478588,2.240725,1.173693
2600,1300,9.325238,2.745112,2.769106,4.975964,2.822881,4.969395,2.187305,2.808040,2.484869,1.413291
2800,100,11.625071,0.480943,0.505080,0.105037,0.096362,6.192705,0.452785,0.587420,0.067504,0.062962
2800,200,11.625725,0.543026,0.567374,0.268335,0.204197,6.192285,0.491241,0.727006,0.160216,0.116478
2800,300,11.625910,0.632823,0.656870,0.504500,0.346502,6.192156,0.549798,0.876296,0.283352,0.179679
2800,400,11.625654,0.756601,0.781536,0.808321,0.503658,6.191917,0.623233,1.032641,0.437575,0.256675
2800,500,11.626333,0.908192,0.932466,1.165762,0.698744,6.191836,0.748231,1.226219,0.617751,0.346295
2800,600,11.626440,1.018259,1.043037,1.591946,0.860997,6.191490,0.819517,1.381561,0.833268,0.420614
2800,700,11.625941,1.263598,1.288479,2.033771,1.136924,6.191534,1.014684,1.624638,1.054066,0.553870
2800,800,11.626186,1.604784,1.630154,2.550323,1.494784,6.192001,1.229845,1.867221,1.323322,0.745477
2800,900,11.626524,1.749377,1.775254,3.077438,1.723293,6.191442,1.297698,2.015535,1.583598,0.845418
2800,1000,11.625892,1.814076,1.839966,3.665420,1.834385,6.191347,1.392510,2.180605,1.895485,0.911497
2800,1100,11.624927,2.008105,2.033718,4.244268,2.064358,6.191221,1.586801,2.424683,2.166103,1.017124
2800,1200,11.624471,2.347845,2.373418,4.894800,2.422355,6.191679,1.872163,2.730198,2.490143,1.215151
2800,1300,11.625458,2.804637,2.831042,5.556317,2.909368,6.191439,2.241795,3.089046,2.770930,1.441800
2800,1400,11.624160,3.374034,3.400095,6.237548,3.507708,6.190496,2.688232,3.476361,3.134156,1.769834
3000,100,14.219450,0.585606,0.602093,0.112560,0.103476,7.594109,0.552723,0.701620,0.072343,0.067755
3000,200,14.213093,0.653697,0.675566,0.288078,0.218556,7.593110,0.592685,0.862938,0.172263,0.124513
3000,300,14.207181,0.742343,0.765647,0.542698,0.366970,7.591786,0.656652,1.033357,0.305037,0.190323
3000,400,14.203957,0.891813,0.916260,0.871250,0.553011,7.590889,0.749760,1.220957,0.471650,0.283252
3000,500,14.203758,1.012279,1.037101,1.259974,0.731880,7.590615,0.854399,1.419614,0.667785,0.360046
3000,600,14.203096,1.202114,1.228058,1.723849,0.970507,7.589730,0.954611,1.610105,0.901514,0.475119
3000,700,14.209046,1.367901,1.394215,2.207927,1.182442,7.589196,1.110597,1.846378,1.143349,0.570028
3000,800,14.203055,1.732819,1.759605,2.774968,1.571898,7.588910,1.375370,2.144497,1.438910,0.780679
3000,900,14.179746,2.005724,2.033258,3.356959,1.926454,7.587985,1.545511,2.378185,1.726671,0.947845
3000,1000,14.165531,2.156890,2.185152,4.008470,2.127695,7.586904,1.611550,2.534616,2.072658,1.058292
3000,1100,14.159625,2.258827,2.286790,4.654440,2.272085,7.586873,1.754101,2.752445,2.374422,1.115527
3000,1200,14.157930,2.522972,2.551151,5.378352,2.568269,7.587065,2.000663,3.050082,2.738621,1.282734
3000,1300,14.154123,2.928788,2.956724,6.112607,3.011329,7.586300,2.343631,3.410969,3.058315,1.483677
3000,1400,14.187543,3.470900,3.500163,6.918019,3.603665,7.593848,2.776349,3.826109,3.470214,1.801930
3000,1500,14.258704,4.121099,4.149006,7.650173,4.271177,7.597290,3.285327,4.268490,3.763222,2.110141
3200,100,17.275935,0.615312,0.641644,0.120343,0.110548,9.214681,0.600165,0.771173,0.077489,0.072579
3200,200,17.274855,0.688672,0.714644,0.308071,0.232957,9.214847,0.644633,0.950389,0.184351,0.132841
3200,300,17.274321,0.782061,0.807857,0.581268,0.388316,9.213567,0.706827,1.137035,0.327201,0.199616
3200,400,17.275610,0.931563,0.957061,0.935762,0.579247,9.214008,0.807907,1.349817,0.505699,0.295274
3200,500,17.276287,1.056094,1.081833,1.354497,0.764709,9.213647,0.907392,1.566358,0.716497,0.372222
3200,600,17.268733,1.327304,1.353182,1.856841,1.083030,9.212428,1.073748,1.823114,0.970499,0.532255
3200,700,17.252141,1.443663,1.470048,2.382732,1.248393,9.211332,1.178723,2.040214,1.233550,0.597470
3200,800,17.253283,1.780574,1.807640,3.001079,1.624300,9.211865,1.439935,2.363510,1.554834,0.799708
3200,900,17.250798,2.136395,2.164145,3.637341,2.057853,9.212026,1.727704,2.693021,1.871211,1.011511
3200,1000,17.251881,2.463353,2.490765,4.355854,2.435974,9.212619,1.850363,2.904597,2.248774,1.215078
3200,1100,17.258821,2.561304,2.588769,5.069929,2.574667,9.212841,1.941100,3.097600,2.584447,1.265355
3200,1200,17.252666,2.738565,2.766810,5.881419,2.786987,9.212669,2.138073,3.372174,2.988128,1.387525
3200,1300,17.251179,3.073723,3.102114,6.718277,3.162950,9.211511,2.445482,3.728318,3.345932,1.548688
3200,1400,17.247286,3.561974,3.590907,7.586637,3.709909,9.208391,2.856105,4.154678,3.808000,1.848806
3200,1500,17.219557,4.176456,4.205859,8.391529,4.362188,9.204172,3.355398,4.628256,4.141409,2.144146
3200,1600,17.213795,4.923706,4.951681,9.163866,5.109271,9.198748,3.946462,5.143641,4.625297,2.573062
3400,100,20.528293,0.747487,0.764585,0.128404,0.118183,10.997979,0.713513,0.905435,0.082530,0.077623
```

```
3400,200,20.524578,0.824206,0.848049,0.327722,0.247934,10.998525,0.759001,1.110112,0.196359,0.141734
3400,300,20.520515,0.934959,0.961467,0.619365,0.422487,10.998127,0.831640,1.325044,0.349054,0.219025
3400,400,20.562117,1.064201,1.091399,0.998483,0.604937,10.997842,0.919418,1.548298,0.540153,0.307286
3400,500,20.590728,1.219477,1.247913,1.448256,0.818413,10.997854,1.029089,1.790326,0.767668,0.399073
3400,600,20.593021,1.484375,1.513799,1.989061,1.147123,10.998277,1.227706,2.091511,1.040012,0.563386
3400,700,20.670350,1.658201,1.688399,2.556991,1.359859,11.016245,1.322092,2.317672,1.323725,0.651136
3400,800,20.673405,1.909158,1.939725,3.227951,1.677393,11.016163,1.543934,2.639050,1.674017,0.819400
3400,900,20.675520,2.293247,2.324119,3.919464,2.143610,11.015116,1.880454,3.028640,2.016579,1.049856
3400,1000,20.677653,2.746584,2.778232,4.701866,2.648728,11.015787,2.137693,3.347193,2.428034,1.322520
3400,1100,20.674928,3.027124,3.058442,5.484642,2.970976,11.015281,2.228532,3.554722,2.793635,1.462964
3400,1200,20.655686,3.112306,3.143704,6.375491,3.100376,11.012393,2.364328,3.797067,3.237986,1.543370
3400,1300,20.647008,3.356340,3.389704,7.299837,3.379009,11.014384,2.620761,4.135303,3.632554,1.648638
3400,1400,20.646348,3.774955,3.807068,8.273537,3.870088,11.013864,2.992590,4.555852,4.143268,1.918600
3400,1500,20.646029,4.336330,4.368157,9.176816,4.485042,11.014291,3.466062,5.040162,4.519914,2.191521
3400,1600,20.643236,5.052202,5.084645,10.089158,5.232415,11.013126,4.045085,5.584763,5.061364,2.610599
3400,1700,20.665348,5.892272,5.922346,11.057183,6.116194,11.003958,4.706844,6.145918,5.453573,3.012603
3600,100,24.455623,0.775312,0.807387,0.135981,0.125220,13.026349,0.749355,0.979367,0.087838,0.082269
3600,200,24.444487,0.858720,0.891445,0.347878,0.263236,13.024037,0.799248,1.201639,0.208180,0.150392
3600,300,24.417826,0.970527,1.003126,0.658162,0.443015,13.023857,0.874745,1.437429,0.370083,0.229034
3600,400,24.405847,1.139888,1.172681,1.061433,0.654910,13.023866,0.971404,1.688139,0.575410,0.333460
3600,500,24.436108,1.328926,1.361539,1.542444,0.905733,13.024232,1.105599,1.970040,0.817303,0.443823
3600,600,24.503688,1.527058,1.559848,2.122325,1.187166,13.040447,1.276370,2.276738,1.108081,0.579137
3600,700,24.507865,1.841641,1.874915,2.731602,1.523227,13.040080,1.439705,2.568135,1.413940,0.733598
3600,800,24.517899,1.993258,2.026240,3.453717,1.752505,13.056123,1.601789,2.865324,1.790103,0.852310
3600,900,24.523105,2.341519,2.374778,4.199581,2.204097,13.041896,1.934682,3.284894,2.159989,1.071201
3600,1000,24.520922,2.882760,2.916433,5.047753,2.788958,13.041325,2.317787,3.707941,2.605820,1.390729
3600,1100,24.520836,3.339236,3.372779,5.898557,3.285595,13.040930,2.510066,4.001781,3.003617,1.621163
3600,1200,24.494437,3.553333,3.586978,6.871312,3.536718,13.042208,2.603873,4.226985,3.488128,1.759278
3600,1300,24.470062,3.674292,3.707758,7.874670,3.685741,13.028671,2.798341,4.531035,3.918917,1.796954
3600,1400,24.464135,3.996960,4.031335,8.939536,4.096291,13.040419,3.119995,4.937222,4.479006,2.024257
3600,1500,24.485872,4.491948,4.525972,9.951058,4.652517,13.039902,3.560478,5.422909,4.895219,2.262659
3600,1600,24.489133,5.160958,5.196196,10.967279,5.359283,13.039223,4.114572,5.983212,5.498764,2.663058
3600,1700,24.498357,5.964310,5.998670,12.044799,6.232898,13.030064,4.768022,6.590748,5.934625,3.049505
3600,1800,24.466115,6.918320,6.952984,13.109470,7.244247,13.040447,5.522067,7.235494,6.680429,3.633399
3800,100,28.735695,0.924355,0.942762,0.143802,0.132192,15.331485,0.892925,1.136730,0.092394,0.086670
3800,200,28.733487,1.014362,1.041570,0.367892,0.279246,15.330767,0.944671,1.389169,0.220501,0.159669
3800,300,28.746097,1.119605,1.148794,0.696548,0.463769,15.336856,1.017848,1.648602,0.392192,0.239435
3800,400,28.755192,1.316164,1.347739,1.125964,0.705624,15.345472,1.141133,1.942209,0.608413,0.360027
3800,500,28.768567,1.503145,1.535306,1.637469,0.962381,15.331550,1.286169,2.254698,0.867415,0.472193
3800,600,28.747634,1.672259,1.704475,2.254257,1.224595,15.319592,1.415111,2.558558,1.179140,0.594026
3800,700,28.699085,2.067258,2.101332,2.905152,1.640201,15.320491,1.667861,2.934304,1.504666,0.791892
3800,800,28.691672,2.227741,2.261646,3.678109,1.873105,15.318331,1.777733,3.211499,1.905469,0.910224
3800,900,28.691916,2.484037,2.518089,4.478969,2.263001,15.318910,2.068338,3.631786,2.305760,1.091810
3800,1000,28.737753,3.051907,3.087613,5.391253,2.878993,15.331216,2.500075,4.117145,2.783338,1.430183
3800,1100,28.733805,3.636362,3.671837,6.308519,3.501577,15.328824,2.852033,4.536074,3.212974,1.727824
3800,1200,28.744271,4.070885,4.106431,7.366453,3.981147,15.348628,2.989935,4.809382,3.738463,1.983911
3800,1300,28.764646,4.198781,4.234724,8.477107,4.114881,15.332405,3.119033,5.080573,4.206845,2.004880
3800,1400,28.782595,4.403392,4.439678,9.649650,4.420119,15.331887,3.378759,5.455295,4.815416,2.180013
3800,1500,28.748618,4.793976,4.829544,10.723760,4.887665,15.334187,3.768124,5.925467,5.272630,2.371173
3800,1600,28.737527,5.382963,5.419044,11.824825,5.528349,15.329438,4.284945,6.484430,5.932999,2.738890
3800,1700,28.735194,6.116279,6.151979,13.025930,6.364305,15.329787,4.909776,7.110023,6.424425,3.101084
3800,1800,28.733408,7.052793,7.089454,14.225305,7.344894,15.316037,5.649777,7.795650,7.219454,3.672234
3800,1900,28.659925,8.076615,8.111337,15.300926,8.437442,15.307606,6.482471,8.508814,7.567021,4.138785
4000,100,33.413918,0.951984,0.985580,0.151231,0.139060,17.850916,0.939446,1.209642,0.097527,0.091352
4000,200,33.537865,1.043292,1.076843,0.387841,0.293787,17.880273,0.994784,1.481660,0.232550,0.168394
4000,300,33.545689,1.166772,1.200125,0.734619,0.497883,17.880792,1.077008,1.771790,0.413960,0.258116
4000,400,33.549476,1.352943,1.386959,1.188748,0.732205,17.880412,1.197164,2.088597,0.642736,0.372009
4000,500,33.509079,1.542154,1.576017,1.731493,0.994510,17.876568,1.342914,2.429545,0.916774,0.485520
4000,600,33.496306,1.738631,1.772411,2.387275,1.283932,17.881255,1.478272,2.758584,1.248319,0.620518
4000,700,33.490078,2.141947,2.176140,3.083169,1.709637,17.877955,1.771911,3.191951,1.594300,0.824350
4000,800,33.513446,2.447088,2.482248,3.904415,2.065705,17.877578,1.908372,3.504145,2.021367,1.006097
4000,900,33.507267,2.557899,2.592433,4.758418,2.342087,17.863771,2.139544,3.907060,2.446943,1.123419
4000,1000,33.486091,3.106237,3.141505,5.733960,2.945870,17.861591,2.568733,4.426919,2.959891,1.454259
4000,1100,33.477891,3.776368,3.812696,6.721024,3.646334,17.872105,3.051321,4.954574,3.421901,1.796277
4000,1200,33.511750,4.394187,4.431118,7.861484,4.304668,17.902246,3.330787,5.345539,3.984604,2.146170
4000,1300,33.543367,4.763182,4.800306,9.055053,4.666841,17.879224,3.449316,5.622483,4.492527,2.273990
4000,1400,33.493839,4.850568,4.887330,10.300518,4.854692,17.872053,3.634530,5.956643,5.150910,2.395759
4000,1500,33.493538,5.125070,5.162933,11.501802,5.219136,17.877245,3.961520,6.402687,5.647674,2.528381
```

```
4000,1600,33.521718,5.627558,5.666035,12.721387,5.775279,17.874761,4.429968,6.952291,6.367470,2.853481
4000,1700,33.500910,6.262738,6.300209,14.008528,6.535903,17.861500,5.020898,7.580055,6.898211,3.173182
4000,1800,33.504866,7.149333,7.187195,15.385433,7.508353,17.877526,5.743134,8.298643,7.797043,3.731426
4000,1900,33.495870,8.171894,8.210423,16.601588,8.598860,17.877551,6.571438,9.065033,8.197273,4.187330
4000,2000,33.495821,9.359334,9.399404,17.865725,9.792271,17.875500,7.505957,9.862647,9.075044,4.904441
```

B.2.7 Positive Definite Bulge Matrix - $n = 200 : 4000$ and $b = 100 : 2000$

```
POSDEF BULGE MATRIX TEST (B−>B−>FULL(BP1:BP2)−>B)
 N,B,BLARGE,SGETRF,ASD1_SGETRF,ASD2_SGETRF,SGBTRF,ASD1_SGBTRF,SPOTRF,ASD1_SPOTRF,ASD2_SPOTRF,SPBTRF,ASD1_SPBTRF
600,100,150,0.135504,0.041064,0.044902,0.034104,0.029996,0.073185,0.030634,0.035378,0.020270,0.018925
800,100,200,0.302217,0.062795,0.068208,0.067202,0.049231,0.164053,0.050806,0.060469,0.039889,0.029209
1000,100,250,0.563242,0.101378,0.108081,0.122261,0.078040,0.308680,0.078173,0.096816,0.069363,0.040860
1000,200,250,0.563226,0.141223,0.148472,0.122350,0.112250,0.308903,0.116162,0.138127,0.069303,0.065346
1200,100,300,0.955387,0.133113,0.142292,0.195414,0.111384,0.524156,0.110396,0.141526,0.108517,0.054588
1200,200,300,0.955476,0.183799,0.193107,0.195310,0.151277,0.522376,0.159789,0.199332,0.108478,0.083506
1400,100,350,1.502965,0.192288,0.202447,0.297399,0.151095,0.814162,0.149543,0.196146,0.163634,0.072555
1400,200,350,1.502878,0.248693,0.259715,0.297175,0.196440,0.814787,0.210818,0.271843,0.163592,0.106573
1400,300,350,1.502947,0.337470,0.348433,0.297313,0.275350,0.814445,0.286054,0.351631,0.163674,0.155668
1600,100,400,2.227181,0.229690,0.241694,0.426066,0.186626,1.200423,0.189723,0.254520,0.229253,0.087173
1600,200,400,2.227262,0.298063,0.310640,0.425944,0.243160,1.200379,0.262473,0.350421,0.229266,0.126871
1600,300,400,2.227575,0.404003,0.416783,0.425876,0.337014,1.200595,0.353412,0.451762,0.229695,0.184662
1800,100,450,3.148594,0.293397,0.306459,0.602683,0.257261,1.688749,0.245439,0.337485,0.319036,0.109159
1800,200,450,3.148781,0.372070,0.386278,0.602517,0.314476,1.689157,0.329404,0.452480,0.319219,0.154635
1800,300,450,3.148557,0.492093,0.506678,0.602784,0.419093,1.690172,0.435733,0.576529,0.318836,0.218499
1800,400,450,3.148453,0.650708,0.664910,0.602563,0.565687,1.688354,0.572467,0.715983,0.318984,0.304035
2000,100,500,4.300244,0.362447,0.378324,0.788439,0.310048,2.297288,0.299158,0.417484,0.417051,0.131326
2000,200,500,4.300284,0.447449,0.463745,0.788278,0.373904,2.297241,0.395951,0.557256,0.417216,0.181292
2000,300,500,4.300585,0.580955,0.597517,0.788277,0.488111,2.297190,0.517007,0.706615,0.417518,0.252255
2000,400,500,4.299706,0.756779,0.773571,0.788427,0.651669,2.296986,0.677989,0.877860,0.417876,0.345603
2200,100,550,5.683328,0.444226,0.460759,1.052031,0.401897,3.037801,0.368441,0.518391,0.546602,0.160995
2200,200,550,5.683834,0.538650,0.555836,1.052090,0.472061,3.037521,0.473282,0.681245,0.546306,0.216339
2200,300,550,5.680614,0.683738,0.701805,1.051911,0.595318,3.037511,0.611165,0.857941,0.546630,0.293610
2200,400,550,5.680268,0.875863,0.894659,1.052125,0.775305,3.036986,0.790109,1.057568,0.546557,0.398579
2200,500,550,5.679566,1.107978,1.127084,1.051831,0.997038,3.037079,1.002383,1.273192,0.546528,0.524492
2400,100,600,7.342700,0.525005,0.544340,1.325055,0.456758,3.929210,0.438057,0.623613,0.693240,0.192814
2400,200,600,7.342564,0.632259,0.651400,1.325036,0.541274,3.929495,0.557836,0.814507,0.694094,0.251370
2400,300,600,7.340726,0.788338,0.808268,1.324889,0.679490,3.929059,0.715764,1.023702,0.694007,0.336693
2400,400,600,7.339711,0.995998,1.015956,1.325084,0.877809,3.929025,0.914049,1.252710,0.694680,0.450112
2400,500,600,7.336565,1.246162,1.266365,1.325133,1.122502,3.929202,1.149082,1.501405,0.693831,0.590318
2600,100,650,9.292005,0.632084,0.651649,1.660450,0.567755,4.970291,0.527200,0.756198,0.865413,0.228725
2600,200,650,9.290040,0.745589,0.766590,1.660723,0.651223,4.969554,0.655791,0.975611,0.865839,0.292182
2600,300,650,9.290129,0.917606,0.938892,1.660419,0.793041,4.968972,0.827050,1.213373,0.866282,0.385578
2600,400,650,9.290419,1.142533,1.164695,1.660383,1.004574,4.969154,1.051004,1.479905,0.865658,0.507503
2600,500,650,9.290958,1.415979,1.439279,1.660408,1.263470,4.967038,1.307608,1.763581,0.865679,0.656923
2600,600,650,9.289198,1.741884,1.765359,1.660181,1.585840,4.967477,1.604732,2.070355,0.865672,0.835505
2800,100,700,11.587233,0.737226,0.761301,2.032026,0.665033,6.184502,0.592106,0.865130,1.053755,0.263763
2800,200,700,11.587391,0.855883,0.879982,2.031998,0.756131,6.183929,0.734195,1.117301,1.052415,0.332990
2800,300,700,11.580939,1.040029,1.064217,2.031708,0.908306,6.183498,0.924701,1.391266,1.052420,0.431593
2800,400,700,11.567863,1.286519,1.310901,2.031797,1.133065,6.183120,1.175814,1.702433,1.052753,0.563029
2800,500,700,11.554392,1.576311,1.601968,2.031954,1.412875,6.182634,1.458469,2.027253,1.052857,0.721377
2800,600,700,11.551773,1.924621,1.950238,2.031597,1.756215,6.182438,1.788282,2.376962,1.054101,0.911121
3000,100,750,14.108877,0.864441,0.887843,2.509803,0.789746,7.581139,0.716916,1.048800,1.302903,0.314013
3000,200,750,14.109832,0.994427,1.019080,2.509757,0.887947,7.581440,0.869627,1.331947,1.303037,0.387809
3000,300,750,14.109997,1.188222,1.213900,2.509494,1.051398,7.580715,1.076827,1.639831,1.302841,0.495026
3000,400,750,14.106266,1.448429,1.474566,2.509827,1.291640,7.580918,1.350229,1.991596,1.302804,0.636747
3000,500,750,14.104898,1.765664,1.793725,2.509649,1.594109,7.580296,1.657421,2.359643,1.302866,0.808903
3000,600,750,14.103505,2.142564,2.171012,2.509664,1.964757,7.580621,2.016501,2.749656,1.302546,1.015955
3000,700,750,14.102726,2.597899,2.626382,2.509598,2.404109,7.580477,2.422136,3.164759,1.302585,1.256151
3200,100,800,17.084620,0.915390,0.941820,2.996266,0.847033,9.192523,0.784228,1.166081,1.556986,0.348896
3200,200,800,17.084898,1.060823,1.086745,2.996221,0.963592,9.192178,0.944992,1.480187,1.554529,0.426496
3200,300,800,17.107096,1.275296,1.301140,2.996273,1.148914,9.192845,1.173569,1.833842,1.555867,0.542715
```

```
3200,400,800,17.205742,1.560178,1.586778,2.996503,1.416903,9.192956,1.465810,2.232321,1.555416,0.693963
3200,500,800,17.209307,1.902288,1.930166,2.997415,1.739994,9.204252,1.811815,2.654026,1.557655,0.880694
3200,600,800,17.236003,2.311974,2.339947,2.997439,2.140760,9.204459,2.203231,3.097168,1.558215,1.101029
3200,700,800,17.239154,2.811028,2.840011,2.997487,2.616330,9.207901,2.653702,3.567345,1.556430,1.359904
3400,100,850,20.666251,1.141240,1.168627,3.583305,1.025394,11.011635,0.930712,1.376048,1.860377,0.405263
3400,200,850,20.664661,1.295382,1.323683,3.584772,1.136470,11.011784,1.103927,1.729823,1.861581,0.488465
3400,300,850,20.657469,1.513223,1.542576,3.585464,1.323544,11.011780,1.343494,2.120747,1.860696,0.610272
3400,400,850,20.657044,1.808046,1.838281,3.584354,1.594069,11.010413,1.666489,2.569317,1.862016,0.774002
3400,500,850,20.656530,2.165673,2.198281,3.584447,1.939791,11.011269,2.029936,3.034842,1.861759,0.970262
3400,600,850,20.644604,2.596112,2.628050,3.584087,2.362888,11.008425,2.455833,3.528427,1.860854,1.204909
3400,700,850,20.623735,3.110815,3.143849,3.584195,2.864578,11.006981,2.940947,4.048644,1.860509,1.480679
3400,800,850,20.630839,3.702267,3.735304,3.583601,3.451799,11.009701,3.446642,4.570023,1.860214,1.803759
3600,100,900,24.492201,1.199118,1.232299,4.193429,1.168369,13.038623,1.001786,1.534972,2.159614,0.458065
3600,200,900,24.488990,1.361115,1.394086,4.193952,1.284149,13.038738,1.189802,1.929097,2.159378,0.545444
3600,300,900,24.487879,1.603286,1.636869,4.193456,1.482748,13.038729,1.450816,2.364757,2.158105,0.676361
3600,400,900,24.488657,1.923987,1.957795,4.193621,1.769506,13.038293,1.802934,2.859435,2.159106,0.845069
3600,500,900,24.492276,2.310080,2.344601,4.193732,2.128952,13.038711,2.196613,3.380758,2.158906,1.052107
3600,600,900,24.495053,2.773393,2.808451,4.193776,2.573475,13.037624,2.655358,3.932431,2.159629,1.297186
3600,700,900,24.502073,3.332522,3.367517,4.194008,3.101248,13.038145,3.173348,4.500801,2.157244,1.585217
3600,800,900,24.488244,3.974212,4.007648,4.193592,3.721789,13.031023,3.738644,5.084394,2.155799,1.917777
3800,100,950,28.621121,1.469495,1.500825,4.950516,1.349736,15.314014,1.185000,1.783403,2.552880,0.532261
3800,200,950,28.588401,1.636919,1.669177,4.950436,1.470629,15.305687,1.383228,2.221423,2.553326,0.623553
3800,300,950,28.548275,1.882365,1.915509,4.949340,1.679061,15.301571,1.660332,2.704067,2.554240,0.760508
3800,400,950,28.540492,2.211471,2.245411,4.949349,1.981140,15.301020,2.035442,3.254045,2.553631,0.940479
3800,500,950,28.542237,2.613623,2.648846,4.949737,2.363207,15.302495,2.459436,3.831324,2.553578,1.157972
3800,600,950,28.548821,3.095969,3.132526,4.949738,2.832397,15.302267,2.947339,4.433315,2.553223,1.419836
3800,700,950,28.686399,3.676987,3.715041,4.951330,3.390322,15.321750,3.506526,5.066388,2.554867,1.728415
3800,800,950,28.731249,4.338928,4.374975,4.951734,4.043777,15.324389,4.102295,5.703298,2.553979,2.088556
3800,900,950,28.748169,5.070578,5.107236,4.951468,4.788940,15.324415,4.796921,6.382911,2.554867,2.483557
4000,100,1000,33.484768,1.534774,1.569931,5.729631,1.473725,17.876917,1.264367,1.930173,2.958334,0.609873
4000,200,1000,33.423422,1.707021,1.742405,5.728430,1.611193,17.867336,1.471472,2.414394,2.956788,0.704952
4000,300,1000,33.385560,1.985131,2.020928,5.728001,1.837543,17.859412,1.764016,2.948727,2.958290,0.850610
4000,400,1000,33.364657,2.346990,2.381689,5.727720,2.165348,17.853112,2.168335,3.556518,2.957068,1.039211
4000,500,1000,33.450726,2.769666,2.806400,5.729900,2.572544,17.875951,2.621738,4.192632,2.958623,1.267923
4000,600,1000,33.450464,3.285650,3.324104,5.729819,3.071440,17.875910,3.150320,4.861100,2.960846,1.544367
4000,700,1000,33.448671,3.905237,3.943454,5.729862,3.662214,17.874726,3.750404,5.554747,2.959394,1.866519
4000,800,1000,33.458118,4.622619,4.659850,5.729744,4.355563,17.874796,4.399764,6.251664,2.955884,2.243272
4000,900,1000,33.458324,5.403385,5.441043,5.729859,5.144876,17.875818,5.145192,7.007392,2.957731,2.660921
```