

Algorithm 8xx: PIRO_BAND, Pipelined Plane Rotations for Blocked Band Reduction

TIMOTHY A. DAVIS and SIVASANKARAN RAJAMANICKAM
University of Florida

PIRO_BAND is a library for bidiagonal reduction of unsymmetric banded matrices and tridiagonal reduction of symmetric banded matrices. It supports both real and complex matrices stored in packed band format. The software also can handle single and double precision arithmetic with 32-bit and 64-bit integers. We provide both a C library and MATLAB callable interfaces. The library is about 2 to 7 times faster than LAPACK's band reduction routines. It is about twice as fast as the SBR toolbox for larger matrices.

Categories and Subject Descriptors: G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis, efficiency*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Band Reduction, Band Matrices

1. OVERVIEW

Band reduction methods are an important part of algorithms for eigen value computations of symmetric band matrices and the singular value decompositions of unsymmetric band matrices. Band reduction algorithms use plane rotations [Givens 1958] or Householder transformations [Lang 1993] to reduce the band matrix to the desired form.

LAPACK libraries' [Anderson et al. 1999] band reduction methods are explained in [Kaufman 2000] and [Kaufman 1984]. They use a technique for reducing and chasing more than one entry that are each separated by a particular distance. Band reduction methods that use dense matrix kernels are discussed in [Bischof et al. 2000b]. The SBR toolbox [?] [Bischof et al. 2000a] is optimized to take advantage of the BLAS and the memory heirarchy. The framework of SBR can choose to optimize between the floating point operations, available workspace and using the BLAS. Both the LAPACK and SBR methods are based on the idea of reducing diagonals of the matrix and chasing the fill introduced in [Rutishauser 1963] and [Schwarz 1963]. In [Rajamanickam and Davis 2009] we show that we can use the

Dept. of Computer and Information Science and Engineering and Univ. of Florida, Gainesville, FL, USA. email: davis@cise.ufl.edu. <http://www.cise.ufl.edu/~davis>. Dept. of Computer and Information Science and Engineering and Univ. of Florida, Gainesville, FL, USA. email: srajan@cise.ufl.edu. <http://www.cise.ufl.edu/~srajan>. This work was supported by the National Science Foundation, under grants xxxx and xxxx.

technique to reduce one row/column at a time introduced in [Schwarz 1968] can be used efficiently combined with blocking. When the plane rotations to reduce the matrix are pipelined it lead to algorithms that are faster than both LAPACK and SBR toolbox. In [Rajamanickam and Davis 2009] we also showed that we can utilize the non-zero pattern while accumulating the plane rotations in the left and right. This lead to performance improvements over both LAPACK and SBR band reduction routines. See [Rajamanickam and Davis 2009] for detailed performance comparisons. We implement the algorithms described in [Rajamanickam and Davis 2009] in the library PIRO_BAND.

PIRO_BAND is a library for tridiagonal reduction of symmetric matrices and bidiagonal reduction of unsymmetric band matrices. We also provide functions to compute the band singular value decomposition using the band reduction functions and a simplicial left looking band QR factorization.

Section 2 introduces the features in the PIRO_BAND library. We summarize some additional performance results of PIRO_BAND in section 3.

2. FEATURES

PIRO_BAND is a library with both MATLAB and C interfaces for band reduction. The C interfaces are described in 2.1 and 2.2. The MATLAB interfaces are described in 2.3.

PIRO_BAND accepts general sparse and dense matrices in the MATLAB interface. The C library requires the input matrices to be in the packed band format : Given a m -by- n matrix A with bl diagonals in the upper triangular side and bu diagonals in the upper triangular size the packed band data structure is of the size $(bl+bu+1)$ -by- n where entry $A(i, j)$ will be stored in $(i - j + bu, j)$. The MATLAB interface will accept both sparse/dense matrices and convert them to band form using the structure of the matrix(`sparse`) or the numerical values(`dense`).

Both the C interface and the MATLAB interface support real and complex matrices. Complex matrices are expected to be stored in the C99 style complex format where the real and imaginary part of every entry is stored in consecutive memory locations. C99 is not expected for compiling and using the libraries though. But the test coverage for the library will require C99 support. PIRO_BAND C libraries does not support complex matrices where the real and imaginary part are stored in separate arrays.

2.1 PIRO_BAND interface

The primary function for band reduction is `reduce`. There are eight different versions of this function for all the combination of double precision and single precision arithmetic, real and complex matrices and 32-bit and 64-bit integers. The eight versions of the functions can be defined as:

```
piro_band_reduce_<x><y><z>
x := 'd' | 's' (for double or single precision)
y := 'r' | 'c' (for real of complex matrices)
z := 'i' | 'l' (for 32-bit or 64-bit integers)
```

For example, `piro_band_reduce_dri`, is the function name for using double precision arithmetic for reducing a real band matrix with 32-bit integers. The prototype

for this function will be

```
int piro_band_reduce_dri
(
    int blks[], int m, int n, int nrc, int bl, int bu, double *A,
    int ldab, double *B1, double *B2, double *U, int ldu, double *VT,
    int ldv, double *C, int ldc, double *dws, int sym
)
```

The `reduce` functions destroy their input matrix A as they reduce it to bidiagonal/tridiagonal form. The two diagonals are always returned. The plane rotations are accumulated in the right and left only if they are required. We don't provide separate interface for symmetric and unsymmetric matrices. Instead we use an input flag to differentiate between the two (`sym`).

All the eight versions of the `reduce` functions require an optional block size (`blks`) and a workspace (`dws`) as a parameter. `blks[]` is an array of size four where the first two entries specify the number of columns and rows in the block for the reducing the super diagonals. The next two entries specify the number of rows and columns in the block for reducing the sub diagonals. The block size can be different for reduction in the upper triangular and lower triangular part. The work space should be two times the maximum of the two block sizes. To find the recommended block size we recommend using the function `piro_band_get_blocksize` if you use 32-bit integers. The 64-bit version of the function has a suffix `_l`. `PIRO_BAND` will get the recommended block size and allocate the required work space if they are not passed to the library. See the [gen 2009] for the exact prototype and the description of the parameters. We explain how we choose the block size and the impact on the performance below.

`PIRO_BAND` interface will have some differences from the LAPACK style interface we provide (described below). But it will be faster as it will avoid few transposes. The major differences are :

- `PIRO_BAND` requires the upper bandwidth to be one.
- For symmetric matrices `PIRO_BAND` will require the upper triangular part to be stored.
- `PIRO_BAND` finds $C^T U$ instead of $U^T C$, and V instead of V^T , as in LAPACK. These are more efficient to compute.

The LAPACK style interfaces given below will not have any of these restrictions.

2.2 LAPACK style interface

The LAPACK style interface supports all the eight functions for band reduction in the LAPACK library. The corresponding function names in `PIRO_BAND` will have a prefix `piro_band_` added to it. For example, LAPACK's band reduction function `dsbtrd` will be `piro_band_dsbtrd` in our interface. For the version that supports 64-bit integers we add a suffix `_l` to the function name. The interface will have the exact functionality as the LAPACK libraries except a one difference : the return values of the functions are slightly different. Like LAPACK's functions, a return value of zero is success and a negative value will mean failure. But the exact error codes are different from LAPACK's functions as we use the one function for

symmetric and unsymmetric matrices. A return value of -i may not indicate the *i*th argument is invalid in the LAPACK style interfaces.

2.3 MATLAB interface

The MATLAB interface provides the following functions.

<code>piro_band</code>	Bidiagonal reduction routine for band matrices
<code>piro_band_qr</code>	QR factorization of a band matrix
<code>piro_band_svd</code>	SVD of a band matrix
<code>piro_band_lapack</code>	MATLAB interface to the LAPACK style interfaces
<code>storeband</code>	Store a sparse/full matrix in packed band format

The MATLAB functions `piro_band` and `piro_band_lapack` are straightforward interfaces for the band reduction functions. `storeband` finds a band structure, if one exists, and stores the matrix in band format. It uses the numerical values when the input is a dense matrix. It use the structure of the matrix if the input is sparse to find the band matrix.

The function `piro_band_svd` computes the singular value decomposition of a band matrix. We will be able to compute both the full and economy singular value decomposition of the band matrix. The usage also supports finding only the singular values. We use our band reduction algorithm to reduce the matrix to bidiagonal/tridiagonal form. We use LAPACK (bundled within MATLAB) to diagonalize bidiagonal/tridiaonal matrix to compute the singular values. We use a simplicial left-looking QR factorization, and then reduce the upper triangular banded matrix R to find the economy singular value decomposition.

`piro_band_qr` provides a MATLAB interface for this left-looking band QR factorization. It provides an option to find both Q and R and an option to find the householder vectors instead of Q . See [gen 2009] for the exact usage of all these functions.

3. PERFORMANCE

All the performance results here were taken on a machine with sixteen dual core 2.2 GHz opteron processors with 64GB of main memory. We used MATLAB 7.6.0.324 (R2008a). PIRO_BAND was compiled with gcc version 4.1.1 with the option `-O3`.

3.1 Choosing a Block Size

Choosing the right block size for different machines is a difficult problem. We provide a function in C that estimates the recommended block size based on the number of floating point operations. The function will not optimize block size for a specific hardware or compiler. The simple rule we use is to use the block size of 8-by-8 when the required flops is less $1e+10$ and the block size of 32-by-32 otherwise. This will be our default block size.

To verify this simple block size selection we compare five different options : an entire row as the block, block sizes of 8-by-8, 16-by-16, 32-by-32 and 64-by-64. When we use the entire row as the block size we will perform exactly the same amount flops as Schwarz' row reduction method. We will perform few more flops for the other four blocking sizes. Figure 1 shows the performance comparison of various block sizes in relation to the default block size. When the flop count is smaller 8-by-8 is the ideal block size. As the flops reach $1e+10$ there are a few cases

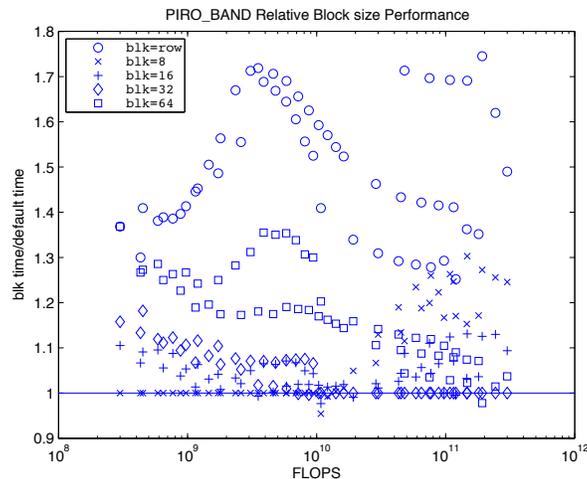


Fig. 1. Performance of `piro_band_reduce` for different block sizes

where 16-by-16 is better. But 32-by-32 are ideal when the flop count is higher. Note that except choosing the entire row as the block, all the other block size perform within 30% of the default size. We can also see that 64-by-64 is almost as good as 32-by-32 for high flops. There can be a cut-off point where 64-by-64 may start performing better than 32-by-32. We chose to leave the default at 32-by-32 and let the user experiment for their common problem sizes.

As the block sizes are so platform dependent we repeated this experiment on a machine with Pentium 3.2 GHz and 4 GB of memory. Though our default case is not the best as often as shown in figure 1 it was never more than 15% worse than the best block size.

3.2 Results

We compare the performance results of PIRO_BAND's singular value decomposition and PIRO_BAND's QR factorization with their MATLAB's equivalents in this section. Detailed performance results of PIRO_BAND package against the LAPACK and SBR band reduction routines can be found in [Rajamanickam and Davis 2009].

MATLAB does not have a band singular value decomposition. The sparse singular value decomposition is too slow when compared with the dense singular value decomposition, especially when all the singular values are required. So we compare our band singular value decomposition against the dense algorithm. As we operate only on the band, our algorithm is asymptotically faster than the dense SVD. The timing results for finding only the singular values are summarized in the table below.

N	Bandwidth	PIRO_BAND SVD (seconds)	MATLAB DENSE SVD (seconds)
1000	100	1.7	47.1
2000	200	11.2	149.3
3000	300	34.9	306.0
4000	300	64.2	540.5

We compare PIRO_BAND's QR factorization against MATLAB's dense QR factorization and MATLAB's sparse QR factorization. The performance results are provided in the following table.

N	Bandwidth	PIRO_BAND QR	DENSE QR	SPARSE QR
1000	100	1.5	5.3	11.9
2000	200	14.7	34.0	159.9
3000	300	48.4	106.0	1907.3
4000	300	87.0	239.0	4145.9

We should note that MATLAB's sparse QR is not a very fast algorithm. As our QR factorization is a support routine used only for economy singular value decomposition we use a simplicial algorithm. A true supernodal/multifrontal QR factorization algorithm (cite spqr) will be able to do better than this algorithm with a small additional cost in the integer workspace.

4. CONCLUSION

We provide a library for band reduction that supports all the data types with both C and MATLAB interfaces. PIRO_BAND's band reduction is faster than LAPACK's band reduction and SBR's band reduction routines in the order of 2 to 8 times for the various types of input matrices. Its Singular value decomposition and QR factorization performance is described here. PIRO_BAND is available as a collected ACM TOMS algorithm. The Software can be downloaded at <http://www.cise.ufl.edu/research/sparse>. PIRO_BAND's MATLAB interface uses LAPACK libraries that come bundled with MATLAB. Other than that PIRO_BAND does not have any other dependencies.

REFERENCES

2009. PIRO_BAND user guide.
- ANDERSON, E., BAI, Z., BISCHOF, C., BLACKFORD, S., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., AND SORENSEN, D. 1999. *LAPACK Users' Guide*, Third ed. Society for Industrial and Applied Mathematics, Philadelphia, PA.
- BISCHOF, C. H., LANG, B., AND SUN, X. 2000a. Algorithm 807: The SBR toolbox—software for successive band reduction. *ACM Trans. Math. Softw.* 26, 4, 602–616.
- BISCHOF, C. H., LANG, B., AND SUN, X. 2000b. A framework for symmetric band reduction. *ACM Trans. Math. Softw.* 26, 4, 581–601.
- GIVENS, W. 1958. Computation of plane unitary rotations transforming general matrix to triangular form. *Journal of the Society of Industrial and Applied Mathematics.* 6, 1, 26–50.
- KAUFMAN, L. 1984. Banded eigenvalue solvers on vector machines. *ACM Trans. Math. Softw.* 10, 1, 73–85.
- KAUFMAN, L. 2000. Band reduction algorithms revisited. *ACM Trans. Math. Softw.* 26, 4, 551–567.
- LANG, B. 1993. A parallel algorithm for reducing symmetric banded matrices to tridiagonal form. *SIAM J. Sci. Comput.* 14, 6, 1320–1338.

- RAJAMANICKAM, S. AND DAVIS, T. A. 2009. Blocked band reduction for symmetric and unsymmetric matrices. *Submitted to ACM Trans. Math. Softw.*
- RUTISHAUSER, H. 1963. On Jacobi rotation patterns. *Proceedings of Symposia in Applied Mathematics 15*, 219–239.
- SCHWARZ, H. R. 1963. Algorithm 183: Reduction of a symmetric bandmatrix to triple diagonal form. *Communications of the ACM 6*, 6, 315–316.
- SCHWARZ, H. R. 1968. Tridiagonalization of a symmetric band matrix. *Numer. Math. 12*, 4, 231–241.