User Guide for PIRO_BAND: General band reduction using blocked Givens rotations

Sivasankaran Rajamanickam, Timothy A. Davis
Dept. of Computer and Information Science and Engineering
Univ. of Florida, Gainesville, FL

Version 1.0, May 9, 2014

Abstract

PIRO_BAND is package for reducing both symmetric and unsymmetric band matrices to tridiagonal and bidiagonal matrices respectively using blocked and pipelined Givens rotations. The package also includes MATLAB interfaces for finding the Singular Value Decomposition of a band matrix using the bidiagonal reduction algorithm. It also supports the exact interface for LAPACK's band reduction routines. PIRO_BAND is written in ANSI/ISO C. It is tested on various Unix variants and Microsoft Windows. The packages can handle both real and complex matrices. The package is competitive with other band reduction packages.

Contents

1 Overview

PIRO_BAND is a package for reducing banded matrices **A** such that

$$\mathbf{A} = \mathbf{U} * \mathbf{B} * \mathbf{V}'$$

where **B** is a bidiagonal matrix when **A** is an unsymmetric matrix and **B** is a tridiagonal matrix when A is a symmetric matrix. **U** and **V** are orthonormal matrices. PIRO_BAND uses blocked and pipelined Givens rotations to reduce the band matrices. By pipelining PIRO_BAND will not generate more than a scalar fill. Blocking the Givens rotations enables PIRO_BAND to be cache efficient.

PIRO_BAND provides one common interface for reducing matrices that are either symmetric or unsymmetric, real or complex with 32-bit or 64-bit integer dimensions. The package is tested on various UNIX variants, in both 32 bit and 64 bit architectures, and Microsoft Windows. The C library also supports LAPACK's eight different interfaces for band reduction too.

The MATLAB interface supports band reduction to bidiagonal or tridiagonal form, and a function for computing the singular value decomposition of a sparse matrix that exploits the band. The MATLAB interface supports both full and sparse matrices.

2 Compiling PIRO_BAND

PIRO_BAND requires `make` to compile the C callable libraries and the test coverage. The MATLAB mex files can be compiled with `make` or from within MATLAB itself. Further test coverage assumes C99 style complex data type, but this is not assumed in the rest of the package.

Obtain and install SuiteSparse_config and edit the configuration file in the `SuiteSparse_config/SuiteSparse_c` file. Sample configurations are provided for Linux, Macintosh, and many other systems.
Here are the various parameters that you can control in your `SuiteSparse_config/SuiteSparse_config.mk` file to compile and install PIRO_BAND :

- `CC =` your C compiler, such as `cc`.
- `CFLAGS =` optimization flags, such as `-O`.
- `RANLIB =` your system's `ranlib` program, if needed.
- `AR =` the command to create a library (such as `ar`).
- `RM =` the command to delete a file.
- `MV =` the command to rename a file.
- `LIB =` basic libraries, such as `-lm`.
- `MEX =` the command to compile a MATLAB mexFunction.

2.1 Compiling the C-Callable library

Run `make` in the `PIRO_BAND` directory. This will compile the PIRO_BAND library and a demo code in the `PIRO_BAND/Demo` directory (`test_piro_band`). To run the simple test code run `test_piro_band` in the directory `PIRO_BAND/Test`. To run the entire suite of tests that cover all the lines in the code type `make cov` in the directory `PIRO_BAND/Tcov`. The norms that are printed should be small. PIRO_BAND library is now ready to use in your own applications.

You can include the header files `piro_band.h` and `piro_band_lapack.h` in your code to use the PIRO_BAND's interface or LAPACK style interface in your code. The header files are in `PIRO_BAND/Include` directory. To compile your application successfully include `PIRO_BAND/Include` in your application's include path and link to `PIRO_BAND/Source/libpiro_band.a` with your own code. Alternatively, `make install` will copy this files to `/usr/local/lib` and `/usr/local/include`.

To compile only the C-callable libraries type `make` in `PIRO_BAND/Source` directory, or `make library` in the `PIRO_BAND` directory. This will generate `PIRO_BAND/Source/libpiro_band.a` but will not compile the test code, test coverage or the MATLAB mex files.

2.2 Compiling and Installing PIRO_BAND for use in MATLAB

PIRO_BAND provides a MATLAB interface for the band reduction and a singular value decomposition for sparse or full matrices that exploits the band.

To compile the MATLAB mexFunctions, type `piro_band_make` while in the `PIRO_BAND/MATLAB` directory. This also updates your path for the current MATLAB session. Running `testall` in the `PIRO_BAND/MATLAB` directory runs an exhaustive test. All the norms printed should be small. Your matlab libraries and m-files are ready to use.

3 Using PIRO_BAND in MATLAB

PIRO_BAND provides various mex functions and m-files to use in MATLAB. The following is a list of functions with a short decription.

`piro_band`	Bidiagonal reduction routine for band matrices
`piro_band_svd`	SVD of a band matrix

The usage for each of the function follows.

3.1 `piro_band` : Bidiagonal reduction of band matrices

```
PIRO_BAND reduce a band matrix to upper bidiagonal form.

 piro_band reduces a band matrix (stored in sparse or full format) to an upper
 bidiagonal matrix using blocked and interleaved Givens rotations.

 Usage:

 [B, U, V] = piro_band (A)
 [B, U, V] = piro_band (A, opts)

 A is m-by-n and can be real or complex.  n must be 2 or more.  B is real, and
 is the same size as A.  It is upper bidiagonal for the unsymmetric case, and
 symmetric tridiagonal for the symmetric case.  A is equal to U*B*V', where U
 and V are full orthogonal matrices.  If A is m-by-n, B is the same size, U is
 m-by-m, and V is n-by-n.  A and B have the same singular values.

 opts can be a single parameter containing the following fields:

   opts.sym:  0 if A is unsymmetric, 1 if symmetric.  Default is 0.
   opts.blks: a vector of size 4

 alternatively, you can specify a list of individual arguments in any order.
 For example:

   ... = piro_band (A, 'sym', [32 32 64 64]) ;

 where 'sym' is the same as opts.sym=1, and a vector of size 4 is used for
 the opts.blks parameter.

 If opts.sym=1, then A must be square and is assumed to be symmetric.  Only
 the entries in the upper triangular part are considered, and the lower
 triangular part is assumed to be the transpose of the upper part.  The
 symmetric matrix C = A + tril(A',-1) is operated on via symmetric reductions,
 and B is a symmetric tridiagonal matrix.  B and C have the same singular
 values, and the same eigenvalues.

 blks is an array of size 4 that determines the block sizes used in the block
 reduction for the upper and lower block sizes respectively.  The block size
 for the upper band is blks(1)-by-blks(2), and the lower band is reduced with
 blocks of size blks(3)-by-blks(4).  The opts.blks option is primarily meant
 for performance experiments, since the default block sizes usually give the
 best performance.

 In contrast to piro_band_svd, no fill reducing ordering is used.  Using
 symrcm and permuting the matrix prior to calling piro_band is a good option
 for reducing the bandwidth and thus the total work.
```

```
Example:

  A = rand (4) ;
  [B,U,V] = piro_band (A) ;
  A - U*B*V'
  svd (A) - svds (B)

See also piro_band_svd, svd, symrcm.
Copyright 2012, Sivasankaran Rajamanickam, Timothy A. Davis
http://www.cise.ufl.edu/research/sparse
```

3.2 piro_band_svd : Singular value Decompostion of band matrices

---

```
PIRO_BAND_SVD singular value decomposition of a sparse or full matrix.
 A may be sparse or full, and real or complex.  This function can be many
 times faster than SVD if A is banded or can be permuted into a form with a
 small band.

 Usage:

 s = piro_band_svd (A) ;

   The singular values of A are returned in the vector s.  This does the same
   the same thing as s = svd (full (A)), but exploiting the band of A.

 [U, S, V] = piro_band_svd (A) ;

   Identical to [U,S,V] = svd (full (A)), except that S is returned as sparse.

 [U, S, V] = piro_band_svd (A, 'econ') ;

   Identical to [U,S,V] = svd (full (A), 'econ'), except that S is sparse.

 [ ... ] = piro_band_svd (A, opts),  where opts is a struct:

       opts.econ: if true (nonzero), this is the same as 'econ', above.
           The default value is false.
       opts.qr: if true, perform a QR factorization first.  if false: skip QR.
           The default is to do the QR.
       opts.ordering: a string describing the fill-reducing ordering to use.
           'rcm': band-reducing ordering (SYMRCM).  This is the default.
           'amd': AMD ordering
           'colamd': COLAMD ordering
           'none': no ordering

 For most accurate results for rank-deficient matrices, download and install
 SPQR from http://www.cise.ufl.edu/research/sparse .

 Example

   load west0479
   A = west0479 ;
   s = piro_band_svd (A) ;
   [U,S,V] = piro_band_svd (A) ;
```

```
 See also SVD, PIRO_BAND, SYMRCM, AMD, COLAMD, SPQR.
 Copyright 2012, Sivasankaran Rajamanickam, Timothy A. Davis
 http://www.cise.ufl.edu/research/sparse
 Details: For most accurate results for rank-deficient matrices (and when
 opts.qr is true), the SPQR function from SuiteSparse should be used with a
 non-default drop tolerance of zero.  Otherwise, the built-in QR will be used
 instead.  The built-in QR uses SPQR, but it does not allow the drop tolerance
 to be modified, and thus small singular values will not be computed
 accurately.  A warning is issued if this case occurs.  To obtain SPQR and all
 of SuiteSparse, see http://www.cise.ufl.edu/research/sparse .

 To set the blocksize used internally in the mexFunction, use
 piro_band_svd(A,opts) where opts.blks is a vector of length 4.  This option
 is meant for development and performance evaluation only.

 For testing purposes, opts.qr can be 0: false, 1: use SPQR if available or
 MATLAB QR if not, 2: use MATLAB QR.
--------------------------------------------------------------------------------
 get input options
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
 compute the SVD
--------------------------------------------------------------------------------
```

4 Using PIRO_BAND C library

Functions in PIRO_BAND C library can be used by including one of two header files in your application `PIRO_BAND/Include/piro_band.h` if you use PIRO_BAND interface for the band reduction or `PIRO_BAND/Include/piro_band_lapack.h` if you prefer to use the LAPACK style of interface. In general we recommend to use our interface as it avoids two transposes. You have to include `SuiteSparse_config/SuiteSparse_config.mk` too. The code will look like

```
#include "SuiteSparse_config.h"
#include "piro\_band.h"
```

or

```
#include "SuiteSparse_config.h"
#include "piro\_band_lapack.h"
```

The primary function for band reduction in PIRO_BAND is `piro_band_reduce` which has eight different versions as described in section 4.3. You can call any of these versions based on the required precision, architecture, and whether the inputs are real or complex.

If you wish to know a good blocksize you can call the function `piro_band_get_blocksize` to get a recommended block size which you may then pass to one of the `reduce` functions.

When your application starts, and prior to calling any PIRO_BAND function (or any other function in SuiteSparse), you should call SuiteSparse_start. This function sets various global function pointers, for malloc, calloc, realloc, free, printf, timing routines, and basic mathematics utility functions. The function modifies a globally-accessible struct, so it is not thread-safe. All threads use the same set of functions for these operations, so no single thread should have its own copy anyway. As of SuiteSparse 4.3.0, calling this function is optional, but this may change in future versions. There is also a corresponding SuiteSparse_finish, which you should call when your entire application exists. Currently, the function is an empty placeholder, but future versions may change this.

4.1 PIRO_BAND Naming conventions and parameters

All the function names provided by PIRO_BAND has the prefix `piro_band_`. There are two different styles for the suffixes. A suffix of the style `_xyz` where `x`, `y` and `z` should be replaced by the appropiate letters given below to get the actual function name.

- `x = d | s` for double or single precision respectively.
- `y = r | c` for real or complex matrices respectively.
- `z = i | l` for 32-bit or 64-bit integers.

For example, the primary function to reduce a band matrix to bidiagoal or tridiagonal form is `piro_band_reduce_xyz` allows all the eight combinations resulting in the following functions. All the prototypes are listed in section 4.3.

piro_band_reduce_dri	for double precision, real matrices and 32 bit integers
piro_band_reduce_drl	for double precision, real matrices and 64 bit integers
piro_band_reduce_sri	for single precision, real matrices and 32 bit integers
piro_band_reduce_srl	for single precision, real matrices and 64 bit integers
piro_band_reduce_dci	for double precision, complex matrices and 32 bit integers
piro_band_reduce_dcl	for double precision, complex matrices and 64 bit integers
piro_band_reduce_sci	for single precision, complex matrices and 32 bit integers
piro_band_reduce_scl	for single precision, complex matrices and 64 bit integers

For functions that differ only in the usage of 32-bit and 64-bit integers the suffix _l is used for 64-bit versions while 32-bit version do not have any suffix. For example, the function to get the recommended blocksize has two versions piro_band_get_blocksize and piro_band_get_blocksize_l for 32-bit and 64-bit versions respectively.

PIRO_BAND functions do not differentiate between symmetric and unsymmetric matrices at the interface level. Instead we use a parameter to the function to differentiate between them.

All the functions in the LAPACK style interface have the prefix piro_band_ added to the original LAPACK names. They may have the suffix _l added to them depending on whether it is the 32-bit or 64-bit version. For example, LAPACK's bidiagonal reduction routine dgbbrd is called piro_band_dgbbrd or piro_band_dgbbrd_l in our LAPACK style interface.

The return value of zero means success. Error return codes are are described in Section 4.7.

4.2 Workspace requirements

PIRO_BAND requires a floating point work space to store the blocked Givens rotations. The size of the work space depends on the block size which is a user controlled parameter : the first parameter in all the eight versions of the piro_band_reduce functions. The block size parameter is an array of size four where the first two entries specify the number of columns and rows in the block to reduce the band in the upper triangular part. The next two entries specify the number of rows and columns in the block to reduce the lower triangular part. The workspace required is twice the maximum blocksize. For example for a 10-by-10 matrix with both upper and lower bandwidth 5 the code snippet to allocate the workspace looks like

```
int blks[4] ;
int msize ;
double *wspace ;

piro\_band_get_blocksize(10, 10, 5, 5, blks) ;
msize = 2 * MAX(blks[0]*blks[1], blks[2]*blks[3]) ;
wspace = (double *) malloc(msize * sizeof(double)) ;
```

If the input matrices are complex then the work space required will be twice the work space required for reducing real matrices. If you pass a NULL block size and NULL work space then PIRO_BAND will determine the best block size and allocate the required work space internally.

4.3 PIRO_BAND interface for bidiagonal reduction of band matrices

This section lists the prototypes for the eight different versions of the piro_band_reduce function.

4.3.1 piro_band_reduce_dri

Purpose: To reduce a real symmetric or unsymmetric band matrix to a tridiagonal or bidiagonal matrix using double precision arithmetic and 32-bit integers.

The other functions that are similar in the funtionality except for the data types are :

4.3.2 `piro_band_reduce_drl`

Purpose: To reduce a real symmetric or unsymmetric band matrix to a tridiagonal or bidiagonal matrix using double precision arithmetic and 64-bit integers. See section 4.3.1 for detailed description of the parameters.

4.3.3 `piro_band_reduce_sri`

Purpose: To reduce a real symmetric or unsymmetric band matrix to a tridiagonal or bidiagonal matrix using single precision arithmetic and 32-bit integers. See section 4.3.1 for detailed description of the parameters.

4.3.4 `piro_band_reduce_srl`

Purpose: To reduce a real symmetric or unsymmetric band matrix to a tridiagonal or bidiagonal matrix using single precision arithmetic and 64-bit integers. See section 4.3.1 for detailed description of the parameters.

4.3.5 `piro_band_reduce_dci`

Purpose: To reduce a complex symmetric or unsymmetric band matrix to a tridiagonal or bidiagonal matrix using double precision arithmetic and 32-bit integers. See section 4.3.1 for detailed description of the parameters.

4.3.6 `piro_band_reduce_dcl`

Purpose: To reduce a complex symmetric or unsymmetric band matrix to a tridiagonal or bidiagonal matrix using double precision arithmetic and 64-bit integers. See section 4.3.1 for detailed description of the parameters.

4.3.7 piro_band_reduce_sci

Purpose: To reduce a complex symmetric or unsymmetric band matrix to a tridiagonal or bidiagonal matrix using single precision arithmetic and 32-bit integers. See section 4.3.1 for detailed description of the parameters.

4.3.8 piro_band_reduce_scl

Purpose: To reduce a complex symmetric or unsymmetric band matrix to a tridiagonal or bidiagonal matrix using single precision arithmetic and 64-bit integers. See section 4.3.1 for detailed description of the parameters.

4.4 Recommened block size for bidiagonal reduction

4.4.1 piro_band_get_blocksize

Purpose: To get the recommended block size for a given matrix in 32-bit architectures.

4.4.2 piro_band_get_blocksize_l

Purpose: To get the recommended block size for a given matrix in 64-bit architectures.

4.5 LAPACK style interface for bidiagonal reduction of unsymmetric band matrices

4.5.1 piro_band_dgbbrd

Purpose: To reduce a real unsymmetric band matrix to a bidiagonal matrix using double precision arithmetic and 32-bit integers.

4.5.2 piro_band_dgbbrd_l

Purpose: To reduce a real unsymmetric band matrix to a bidiagonal matrix using double precision arithmetic and 64-bit integers.

4.5.3 piro_band_zgbbrd

Purpose: To reduce a complex unsymmetric band matrix to a bidiagonal matrix using double precision arithmetic and 32-bit integers.

4.5.4 piro_band_zgbbrd_l

Purpose: To reduce a complex unsymmetric band matrix to a bidiagonal matrix using double precision arithmetic and 64-bit integers.

4.5.5 piro_band_sgbbrd

Purpose: To reduce a real unsymmetric band matrix to a bidiagonal matrix using single precision arithmetic and 32-bit integers.

4.5.6 piro_band_sgbbrd_l

Purpose: To reduce a real unsymmetric band matrix to a bidiagonal matrix using single precision arithmetic and 64-bit integers.

4.5.7 piro_band_cgbbrd

Purpose: To reduce a complex unsymmetric band matrix to a bidiagonal matrix using single precision arithmetic and 32-bit integers.

4.5.8 piro_band_cgbbrd_l

Purpose: To reduce a complex unsymmetric band matrix to a bidiagonal matrix using single precision arithmetic and 64-bit integers.

4.6 LAPACK style interface for bidiagonal reduction of symmetric band matrices

4.6.1 `piro_band_dsbtrd`

Purpose: To reduce a real symmetric band matrix to a bidiagonal matrix using double precision arithmetic and 32-bit integers.

4.6.2 `piro_band_dsbtrd_l`

Purpose: To reduce a real symmetric band matrix to a bidiagonal matrix using double precision arithmetic and 64-bit integers.

4.6.3 `piro_band_zhbtrd`

Purpose: To reduce a hermitian band matrix to a bidiagonal matrix using double precision arithmetic and 32-bit integers.

4.6.4 `piro_band_zhbtrd_l`

Purpose: To reduce a hermitian band matrix to a bidiagonal matrix using double precision arithmetic and 64-bit integers.

4.6.5 `piro_band_ssbtrd`

Purpose: To reduce a real symmetric band matrix to a bidiagonal matrix using single precision arithmetic and 32-bit integers.

4.6.6 `piro_band_ssbtrd_l`

Purpose: To reduce a real symmetric band matrix to a bidiagonal matrix using single precision arithmetic and 64-bit integers.

4.6.7 piro_band_chbtrd

Purpose: To reduce a hermitian band matrix to a bidiagonal matrix using single precision arithmetic and 32-bit integers.

4.6.8 piro_band_chbtrd_l

Purpose: To reduce a hermitian band matrix to a bidiagonal matrix using single precision arithmetic and 64-bit integers.

4.7 Error codes from PIRO_BAND

Return values are described below. The return value of zero means success. The first set of error codes match those codes returned by LAPACK.

PIRO_BAND_OK	0	success
PIRO_BAND_VECT_INVALID	-1	VECT input is not valid
PIRO_BAND_M_INVALID	-2	M input is not valid
PIRO_BAND_N_INVALID	-3	N input is not valid
PIRO_BAND_NRC_INVALID	-4	NRC input is not valid
PIRO_BAND_BL_INVALID	-5	BL input is not valid
PIRO_BAND_BU_INVALID	-6	BU input is not valid
PIRO_BAND_LDAB_INVALID	-8	LDAB input is not valid
PIRO_BAND_LDU_INVALID	-12	LDU input is not valid
PIRO_BAND_LDV_INVALID	-14	LDV input is not valid
PIRO_BAND_LDC_INVALID	-16	LDC input is not valid
PIRO_BAND_A_INVALID	-7	A is a NULL pointer
PIRO_BAND_B1_INVALID	-9	B1 is a NULL pointer
PIRO_BAND_B2_INVALID	-10	B2 is a NULL pointer
PIRO_BAND_U_INVALID	-11	U is a NULL pointer
PIRO_BAND_V_INVALID	-13	V is a NULL pointer
PIRO_BAND_C_INVALID	-15	C is a NULL pointer
PIRO_BAND_SYM_INVALID	-17	SYM input is not valid
PIRO_BAND_BLKSIZE_INVALID	-18	BLKS input is not valid
PIRO_BAND_OUT_OF_MEMORY	-19	out of memory for work space
PIRO_BAND_UPLO_INVALID	-20	UPLO input is not valid
PIRO_BAND_LAPACK_INVALID	-21	internal error
PIRO_BAND_LAPACK_FAILURE	-22	SVD did not converge

5 PIRO_BAND interface vs LAPACK style interface

PIRO_BAND supports two interfaces: a native one, and one that is compatible with LAPACK. They differ in four ways, listed below. The first two simplify the algorithm for band reduction. The last two are for efficient computation.

1. PIRO_BAND interface requires the upper bandwidth to be at least one. The LAPACK style interface adds a zero diagonal to upper triangular part if the upper bandwidth is zero.

2. For symmetric matrices PIRO_BAND requires the upper triangular part to be stored. The lower triangular part may or may not be stored. The LAPACK style interface transposes the input matrix and the results if only the lower triangular part is stored for a symmtric matrix.

3. PIRO_BAND finds $\mathbf{C}'\mathbf{Q}$ instead of $\mathbf{Q}'\mathbf{C}$. The former is more efficient, but the latter is the LAPACK standard. The LAPACK style interface uses two transposes for compatibility.

4. PIRO_BAND finds $\mathbf{V}$ rather than the $\mathbf{V}'$ matrix that LAPACK computes. As in the previous cases the LAPACK style interface transposes to return $\mathbf{V}'$ if required, for compatibility with LAPACK.