bdspec—Special-Band-Matrix Solver

Richard Wood

February 3, 2011

bdspec solves band matrices, but with a twist. Instead of the region above the highest superdiagonal containing zeros, it is filled with values identical to the highest superdiagonal element of the same row. Such matrices shall be referred to as special band matrices.

Throughout, `lb` is the number of subdiagonals, `ub` is the number of superdiagonals, and `n`×`n` is the dimension of the special band matrix.

Some examples of special band matrices

$$\texttt{n} = 6 \left\{ \begin{pmatrix} \alpha_0 & \overbrace{\beta_0}^{\texttt{ub}=1} & \beta_0 & \beta_0 & \beta_0 & \beta_0 \\ & \alpha_1 & \beta_1 & \beta_1 & \beta_1 & \beta_1 \\ & & \alpha_2 & \beta_2 & \beta_2 & \beta_2 \\ & & & \alpha_3 & \beta_3 & \beta_3 \\ & & & & \alpha_4 & \beta_4 \\ & & & & & \alpha_5 \end{pmatrix} \right. \underset{\texttt{lb}=0}{} , \tag{1}$$

$$\texttt{n} = 6 \left\{ \begin{pmatrix} \beta_0 & \overbrace{\gamma_0 \quad \delta_0}^{\texttt{ub}=2} & & \delta_0 & \delta_0 & \delta_0 \\ \alpha_1 & \beta_1 & \gamma_1 & \delta_1 & \delta_1 & \delta_1 \\ & \alpha_2 & \beta_2 & \gamma_2 & \delta_2 & \delta_2 \\ & & \alpha_3 & \beta_3 & \gamma_3 & \delta_3 \\ & & & \alpha_4 & \beta_4 & \gamma_4 \\ & & & & \underbrace{\alpha_5}_{\texttt{lb}=1} & \beta_5 \end{pmatrix} \right. . \tag{2}$$

1 Storage

The matrices are stored in compact form to save space. In compact form elements remain on their original row, but are shifted so that diagonals are stored in columns and repeated elements are removed. `lb` columns of padding are added to the left to be used for LU factorization. The compact versions of the two matrices above are stored in the arrays

$$\mathtt{n}=6\left\{\begin{pmatrix} \alpha_0 & \beta_0 \\ \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \\ \alpha_4 & \beta_4 \\ \alpha_5 & * \end{pmatrix}\right. , \qquad \text{with } \overbrace{}^{\mathtt{ub}=1} \text{ over the second column} \tag{3}$$

$$\mathtt{n}=6\left\{\begin{pmatrix} * & * & \beta_0 & \gamma_0 & \delta_0 \\ * & \alpha_1 & \beta_1 & \gamma_1 & \delta_1 \\ * & \alpha_2 & \beta_2 & \gamma_2 & \delta_2 \\ * & \alpha_3 & \beta_3 & \gamma_3 & \delta_3 \\ * & \alpha_4 & \beta_4 & \gamma_4 & * \\ * & \alpha_5 & \beta_5 & * & * \end{pmatrix}\right. , \tag{4}$$

with $\overbrace{\gamma\ \delta}^{\mathtt{ub}=2}$ over the last two columns and $\underbrace{*\ \alpha}_{\mathtt{lb}=1}$ under the first two columns,

respectively, where $*$ indicates unused elements.

For a special band matrix A, to solve the matrix problem $A\mathbf{x}=\mathbf{b}$ for vector $\mathbf{x}$ Crout LU decomposition with partial pivoting is performed. The factorization is performed in-place. The first `lb` columns of the compact array store the subdiagonals of lower triangular matrix L. The diagonal of matrix L contains ones and is not stored. The remaining $\mathtt{ub}+\mathtt{lb}+1$ columns of the compact array store the diagonal and superdiagonals of the upper triangular matrix U. The rows of the compact array are re-ordered as pivoting takes place.

A small example helps understand the storage of the LU decomposition. A special band matrix A with $\mathtt{n}=6$, $\mathtt{lb}=1$, and $\mathtt{ub}=2$:

$$A=\begin{pmatrix} 0.31 & 0.35 & -0.63 & -0.63 & -0.63 & -0.63 \\ -0.99 & -0.02 & -0.53 & -0.24 & -0.24 & -0.24 \\ & -0.88 & -0.82 & 0.27 & -0.00 & -0.00 \\ & & 0.76 & 0.03 & 0.38 & 0.74 \\ & & & -0.81 & -0.07 & 0.25 \\ & & & & 0.24 & -0.25 \end{pmatrix}. \tag{5}$$

Compact storage of A:

$$\texttt{bA} = \begin{pmatrix} * & * & 0.31 & 0.35 & -0.63 \\ * & -0.99 & -0.02 & -0.53 & -0.24 \\ * & -0.88 & -0.82 & 0.27 & -0.00 \\ * & 0.76 & 0.03 & 0.38 & 0.74 \\ * & -0.81 & -0.07 & 0.25 & * \\ * & 0.24 & -0.25 & * & * \end{pmatrix}. \tag{6}$$

The LU decomposition of A:

$$L = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ -0.31 & -0.39 & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & -0.68 & 0.47 & -0.28 & 1 \end{pmatrix}, \tag{7}$$

$$U = \begin{pmatrix} -0.99 & -0.02 & -0.53 & -0.24 & -0.24 & -0.24 \\ & -0.88 & -0.82 & 0.27 & -0.00 & -0.00 \\ & & -1.11 & -0.60 & -0.70 & -0.70 \\ & & & -0.81 & -0.07 & 0.25 \\ & & & & 0.24 & -0.25 \\ & & & & & 0.07 \end{pmatrix}. \tag{8}$$

Compact storage of L and U:

$$\texttt{bLU} = \begin{pmatrix} -0.31 & -0.99 & -0.02 & -0.53 & -0.24 \\ -0.39 & -0.88 & -0.82 & 0.27 & -0.00 \\ -0.68 & -1.11 & -0.60 & -0.70 & -0.70 \\ 0.47 & -0.81 & -0.07 & 0.25 & * \\ -0.28 & 0.24 & -0.25 & * & * \\ * & 0.07 & * & * & * \end{pmatrix} \tag{9}$$

The reordering of the rows due to partial pivoting is stored in an integer array `indx`. To obtain the permutation matrix P from `indx`:

```
Let $P := I =$ identity matrix
for $j = 0, 1, 2, \dots \texttt{n} - 1$ do
  swap rows $j$ and $\texttt{indx}[j]$ of $P$
end for
```

For the above example:

$$\texttt{indx} = \begin{pmatrix} 1 & 2 & 0 & 4 & 5 & 3 \end{pmatrix} \tag{10}$$

$$P = \begin{pmatrix} & 1 & & & & \\ & & 1 & & & \\ 1 & & & & & \\ & & & & 1 & \\ & & & & & 1 \\ & & & 1 & & \end{pmatrix}. \tag{11}$$

2 Functions

`void bdspecLUmlt(double *bA, int n, int lb, int ub, double x[], double out[]);`

Calculates $A\mathbf{x}$ for special band matrix A stored compactly in array `bA`. Results are stored in `out`.

`void bdspecLUfactor(double *bA, int n, int lb, int ub, int indx[])`

Performs an Crout LU factorization for special band matrix stored compactly in array `bA`. Results are stored in-place as described above.

`void bdspecLUfactorscale(double *bA, int n, int lb, int ub, int indx[])`

The same as `bdspecLUfactor`, but partial pivoting is performed based on the size of elements after being normalized. Normalizing elements involves scaling them by the element of the largest magnitude that is in the same row.

`void bdspecLUfactormeschscale(double *bA, int n, int lb, int ub, int indx[])`

The same as `bdspecLUfactorscale`, but deliberately includes a bug found in the meschach linear algebra library version 1.2A. The bug is seldom important, and deliberately including it allows careful testing against Meschach results.

`void bdspecLUsolve(double *bLU, int n, int lb, int ub, int indx[], double b[])`

Solve $LU\mathbf{x} = P\mathbf{b}$ for vector $\mathbf{x}$. As detailed above, LU is stored compactly in array `bLU`, P is stored compactly in `indx`. Results are stored in-place in `b`.

Other included functions in `bdspec.c` are intended for testing only. Note that full testing requires the Meschach library to test against. Self consistency tests are insufficient because special band matrices are often nearly singular—for some matrices large (or even infinite) errors are a fact of life and sometimes they are completely unacceptable.

3 Complexity

It should be noted that only `bdspecLUsolve` has been optimized, other functions are expected to be used less frequently.

`bdspecLUmlt` is $\mathcal{O}(\texttt{n} \times (\texttt{lb+ub}))$

`bdspecLUsolve` is $\mathcal{O}(\texttt{n} \times (\texttt{lb+ub}))$

`bdspecLUfactor` and its cousins are $\mathcal{O}(\texttt{n} \times \texttt{lb} \times (\texttt{lb+ub}))$

4 Complex versions

Complex versions of each routine exist:

void zbdspecLUmlt(bdspec_complex *bA, int n, int lb, int ub, bdspec_complex x[], bdspec_complex out[]);

void zbdspecLUfactor(bdspec_complex *bA, int n, int lb, int ub, int indx[])

void zbdspecLUfactorscale(bdspec_complex *bA, int n, int lb, int ub, int indx[])

void zbdspecLUfactormeschscale(bdspec_complex *bA, int n, int lb, int ub, int indx[])

void zbdspecLUsolve(bdspec_complex *bLU, int n, int lb, int ub, int indx[], bdspec_complex b[])

The `bdspec_complex` type is defined:

```
typedef struct { double r, i; } bdspec_complex;
```

and is binary compatible with both c++'s `complex<double>` class and c99's `complex double` type.

The complex routines would be slightly neater written in c++ or c99. Standard c89 code has been used as it is the lowest common denominator.

5 Bug Reports

Please send bug reports to `richbwood@gmail.com`