Block-circulant matrix package for Matlab

Introduction

Circulant matrices arise in many matrix problems where the underlying mathematical model has rotational symmetry. A circulant matrix has the structure:

```
Mat=[
     abcde
     eabcd
     deabc
     cdeab
     bcdea
];
```

where a...e are numbers and the symmetry, *k*, is 5. Circulant matrices have simple eigenvectors: all are roots of unity times a constant: the kth value of the nth eigenvector is $e^{ink/N}$, where k=0...N-1 and n=0...N-1. The structure of the eigenvectors suggests that there should be a DFT-based algorithm for performing matrix operations on circulant matrices.

In many problems, the problem representation is not a simple circulant matrix, but a *block-circulant* matrix. A block-circulant matrix is like the one above but with repeating submatrices, not just numbers:

```
Mat=[
     ABCDE
     EABCD
     DEABC
     CDEAB
     BCDEA
];
```

Each of the A...E in the example (here, *k*=5) above is an *M* by *N* submatrix. This type of matrix arises in finite element, boundary element, tomography problems, etc. with cylindrical geometry. A typical boundary element problem for which this matrix package is designed would have an underlying block size on the order of *M*x*N* = 256 x 256 and a symmetry order of 512, i.e., a total size of 131072 x 131072 = 17 billion elements. The operations needed to invert this matrix would be O(2.25×10^{15})!

To perform many matrix operations on block-circulant matrices it is sufficient to think of the submatrices A...E as "playing cards" arranged as ABCDE. This uniquely specifies the block-circulant matrix. Then, "stack the cards" the way you would in a card deck.

This deck has dimensions *M* x *N* x *k*. Finally, perform an FFT in the dimension "through the deck": *fMat*=fft(*Mat*,[],3). *fMat* has the same dimensions as *Mat*.

With this transformed representation, many matrix operations (e.g., Matlab inv, mtimes) can be performed simply by performing the corresponding operation on each "card" of the deck (i.e., the transformed submatrices), then performing the inverse fft on the result. The speedup lies in the fact that the operation counts for these important operations are $O(kM(kN)^2)$ for the original matrix ($= k^3O(MN^2)$) while the operation count for the operations on the transformed matrix is $kO(MN^2)$. This results in a reduction factor in the operation count of $O(2^k)$! For the above example (M=N=256 and k=512) this reduces the operation count to less than $9\text{x}10^9$; at a Gflop/sec, this is the difference between months vs. minutes of compute time for a matrix multiply or matrix inverse. Note that for *k*=power of 2 the ffts require an insignificant $O(MNk*\ln_2(k))$ operations. For highly composite *k*, the speedup over the DFT is still significant; even for prime *k*, the DFT-related operations are still a small part of the overall operations count.

Matlab implementation

The package is built around the BlockCirculant object:

Mat=BlockCirculant(Slice,k).

In the example above, Slice is a single "row slice" of a block-circulant matrix (e.g., m=ABCD). The matrix can be single or double, real or complex. The constructor stores the data and its symmetry order as a BlockCirculant object, Mat.

For example, if the individual block size is MxN=4x3 and the symmetry order *k* is 4, the size of matrix Mat would be 4x12. This would represent a matrix ABCD where A, B, C, and D are 4x3. The matrix object would be the compact representation of a 16x12 block-circulant matrix. The usual NxN circulant matrix (shown at the beginning of the introduction) used in convolutions, etc., would have a block size of 1x1 and would be created by Mat=BlockCirculant(m,length(m)), where m is a matrix with dimension 1x*M*.

Below is an example of a large random block circulant matrix with *M*=*N*=128 and *k*=128. The actual underlying matrix, *A*, would be 16384 by 16384 elements. BlockCirculant object methods are in red.

```
>> k=128;A=BlockCirculant(rand(k,k^2),k);
>> A
A =
  BlockCirculant
  Properties:
       value: [128x16384 double]
    symmetry: 128
>> tic,B=inv(A);toc %invert
Elapsed time is 0.645901 seconds.
>> tic,B1=inv(double(A));toc %convert to full matrix and invert
Elapsed time is 384.165892 seconds.
```

Note that the speedup for this problem is ~600.

Here is an example showing the internal structure of a small block-circulant matrix:

```
>> k=2;
>> A=BlockCirculant(rand(k,k^2),k);

>> unpack(A)
ans =
      0.15541      0.82805      0.13432       0.4258
     0.050059     0.014199      0.52641      0.68018
>> double(A)
ans =
      0.15541      0.82805      0.13432       0.4258
     0.050059     0.014199      0.52641      0.68018
      0.13432       0.4258      0.15541      0.82805
      0.52641      0.68018     0.050059     0.014199
>> A
A =
  BlockCirculant

  Properties:
       value: [2x4 double]
    symmetry: 2
```

Note the three methods: *BlockCirculant* (the constructor), *unpack* (the internal representation of the data), and *double* (used to give the full matrix).

Here is a full list of the methods:

```
>> methods(A)

Methods for class BlockCirculant:

BlockCirculant  horzcat         mtimes          subsasgn
unpack          ctranspose      inv             pinv
subsref         uplus           double          minus
plus            transpose       vertcat         end
mldivide        submatrix       uminus
```

The BlockCirculant source code has comments clarifying the use of the above methods. The purely arithmetic operations are straightforward, but the subscripting and concatenation require some explanation:

1. horzcat - two block-circulant matrices can be concatenated and produce a block-circulant matrix if the submatrices are concatenated instead of simply appending one matrix to the end of the other:

   ```
   M1=BlockCirculant([A B C D],k);
   M2=BlockCirculant([a b c d],k);
   M=horzcat(M1,M2) <=> M=BlockCirculant([A a B b C c D d],k);
   ```

   and similarly for vertcat. When building up matrices this way it is important to match the ordering of the columns with the ordering of a row matrix to be multiplied, etc. by the matrix M (e.g., x=[X Y]\b requires the corresponding interleaved row ordering of the elements of x).

2. subsref, subsasgn - the indices in A(r1:r2,c1:c2) can only range over the size of the underlying matrix slice. The result is a block-circulant matrix.

3. iscirculant(A,k,tol) - function included to check if a matrix is block circulant within a certain tolerance. If a matrix is approximately circulant, a quick solution to Ax=b can be implemented by "polishing" the inverse found by using the circulant approximation A_c to solve $x=A_c\backslash b$, substituting x into b=Ax then iterating on the error. See *Numerical Recipes* for details. Fast convergence depends on the eigenvalues of $(I-A_c^{-1}*A)$ being close to 0.

Test program

The test program *test_BlockCirculant* checks the accuracy of supported matrix operations. It is bundled in the object's directory.

Application Notes

The full power of block circulant matrices comes into play when used to solve problems where the underlying problem geometry has cylindrical symmetry but the actual input data does not. For example, a typical boundary element problem solution reduces to A*x=b, or x=A\b. In this problem, the matrix A describes the interactions of the boundary elements of the geometry, and is thus block-circulant, while b is typically an arbitrary distribution on the surface (e.g., in an acoustics problem b might be surface velocity which can be freely specified; solving for x gives pressure on the surface). In this case, b and x are not circulant, but the speedup is large because the fft formulation still applies to both A and b. The fft of b which is done during the "\" operation can best be thought of expressing b in complex exponential basis functions b_n, where n=0...k-1. This suggests the interpretation of the ffts of A as expansion in complex angular modes of the original physical problem.

In an axial tomography problem, the A matrix would represent the effect of small interior regions (in cylindrical geometry) on a beam in a given direction from source to detector. Since the problem is constructed to have circular symmetry (e.g., rotate the source/target while generating the beams), the matrix is block circulant. In this case, b is the vector of detectors' responses to probe beams at each of k symmetric positions and x is the vector of blocking strengths for (r,θ) elements within the circular region of interest.