ckronx: Efficient Computation with Kronecker Products

Paul L. Fackler[*]

The Matlab procedure `ckronx` is designed to efficiently compute the product of a series of Kronecker products with a matrix:

$$C = (A_1 \otimes A_2 \otimes \ldots \otimes A_d)\, B$$

where A_i is an $m_i \times n_i$ matrix, B is a $\prod_{i=1}^d n_i \times p$ matrix and C is $\prod_{i=1}^d m_i \times p$ matrix. The A_i matrices and the B matrix can be either full (dense) or sparse. The algorithm can also perform the operation with any of the matrices in transposed form (without actually performing the transposition). The operation is performed by conducting a sequence of d matrix-matrix multiplications, with each step in the sequence incorporating one of the d A_i matrices. It differs from previous implementations of this procedure by (largely) avoiding the rearrangement or shuffling of the elements of the arrays in memory.

The sequence in which the A_i are incorporated can be in a forward direction $(1, \ldots, d)$, a backward direction $(d, \ldots, 1)$ or any arbitrary direction (the latter requires a rearrangement of matrix elements before the first and after the last step in the sequence). The order of operations is (essentially) irrelevant if all of the matrices are the same size and density or if all are square. Otherwise the order can have a significant impact on computational efficiency both in terms of computational time and memory usage.

By default the algorithm determines whether the forward or backward approaches is more efficient and uses that ordering (this can be overridden). Optionally the optimal ordering (in terms of fewest arithmetic operations) can be determined and used. If the optimal ordering is not the forward or backward approaches then the rows of B must be permuted before the first step and the rows of the output must be permuted before it is returned. These operations can be time consuming and hence the optimal ordering may use more memory and may be slower than with the forward or backward approaches.

The simplest syntax to call the function is

```
C=ckronx(A,B)
```

where `A` is a d element cell array containing the d A_i matrices and `B` is an ordinary matrix. A cell array is a that structure in Matlab that allows packaging of variables into an array for which individuation elements can be access by index. In the current context a cell array with 3 elements can be formed using `A={A1,A2,A3};` An optional structure variable can be passed that alters the default operations of the procedure. The calling syntax is

```
C=ckronx(A,B,options)
```

Valid fields for the options structure include setting `options.transpose` to a d-vector of 0s and 1s (or logicals) with a 1 indicating that the associated matrix should be transposed; a single 1 indicates that all of the matrices should be transposed and is equivalent to a

[*]Professor, Department of Agricultural and Resource Economics, North Carolina State University (NCSU), Raleigh, NC 27695. paul_fackler@ncsu.edu

d-vector of 1s. Setting `options.optorder` to 1 instructs the procedure to determine and use the optimal order of processing.

Other options are mainly available for exploration. If `options.forward` is set it overrides `options.optorder`. Setting `options.forward` to 1 forces the forward ordering to be used whereas setting it to 0 forces the backward ordering to be used. Setting `options.print` to 1 displays information about the operation to the screen. The `options.ind` field is designed for backward computability with software in MATLAB based toolboxes developed by the author. It allows the A cell array to be permuted; setting `options.ind` is equivalent to calling the procedure as `C=ckronx(A(ind),B)`.

The `ckronx` procedure can also produce an optional second output argument if the `options.optorder` is set to 1:

```
[C,order]=ckronx(A,B,options)
```

where `order` is a d-element vector specifying the optimal processing order. This can be useful in repeated calls if the A_i matrices are reordered and the rows of B are appropriately permuted.

The `ckronx` procedure will fail if B is not compatible with the A_i, meaning that it has an invalid number of rows. This can be checked in MATLAB using the following line of code:

```
size(B,1) == prod( cellfun( @(x) size(x,2), A ) )
```

which returns 1 (true) if A and B are compatible.

The following code sample gives a 3-matrix example.

```
m = [40 50 60];
n = [30 20 10];
A = arrayfun(@(m,n) randn(m,n), m, n, 'UniformOutput', false);
B = randn(prod(n),1);
C = ckronx(A,B);
```

This example is contrasted with the use of the full Kronecker product matrix in the script file `ckronxExample`.