The `bigalgebra` Package

Michael J. Kane, John W. Emerson, and Bryan W. Lewis

December 26, 2012

1 Introduction

The `bigalgebra` package provides arithmetic functions for double-precision valued "Big Matrices" defined by the `bigmemory` package. Infix arithmetic operators are overloaded to use `bigalgebra` functions, resulting in familiar syntax. Operations with mixed `big.matrix` and standard R numeric arguments are supported.

The package may be installed to use the native R basic linear algebra subroutine (BLAS) library (the default case), or to use a reference BLAS library configured with large index support.

When configured with large index support, the package supports operations on matrices with indices of up to about 2^{52} in each dimension. Note that the usual R limit constrains objects to less than 2^{31} total entries in an entire object. The package also supports computations on arrays that are larger than available RAM using `bigmemory`'s file-backed big matrix objects.

2 Installation

The package uses the native R linear algebra libraries (BLAS) by default to simplify installation. Operations are limited to vectors of at most $2^{31} - 1$ elements and matrices of at most $2^{31} - 1$ rows and $2^{31} - 1$ columns. (Note the difference with standard R matrices, which are limited to at most $2^{31} - 1$ total elements.) The following code block illustrates standard command-line installation of the source package:

```
R CMD INSTALL bigalgebra_0.8.1.tar.gz
```

2.1 Installation with Reference BLAS (with large index support)

The default installation ships with a subset of the refererence BLAS and LAPACK libraries available from Netlib[1] compiled at installation time with support for large indices. But, the libraries are not tuned for performance. Enabling large indices supports operations on vectors and matrices with up to $2^{52} - 1$ elements in any dimension.

Enabling the Netlib BLAS with large integer support requires installation using the `bigalgebra` source package from the command line, and is enabled by setting the environment variable `REFBLAS=1`, for example:

```
REFBLAS=1 R CMD INSTALL bigalgebra_0.8.1.tar.gz
```

The package decorates BLAS function names in a way that will not interfere with native R BLAS symbols. Reference BLAS performance (speed) will generally be worse than the native R BLAS performance. Only use the reference BLAS if you really need to work with vectors longer than $2^{31} - 1$.

3 Using the package

The `bigalgebra` package is simple to use. Load the package with:

```
> library("bigalgebra")
```

Once loaded, arithmetic functions are defined for `big.matrix`-class objects. Note that element-wise operations (like Hadamard products), and operations that recycle values are not supported.

The following listing shows a few simple examples.

```
> library("bigalgebra")
> set.seed(1)
> A <- matrix(rnorm(4), nrow=2)
> a <- as.big.matrix(A)       # A tiny, square, big.matrix
> b <- a %*% a                # returns an "anonymous" big.matrix
> c <- a + a                  # another big.matrix result
> d <- 2 * a                  # Mixing scalars and big.matrix arguments
> d <- a * 2                  # scalar multiplication commutes
> f <- a %*% A                # Mixing big.marix and matrix arguments
> g <- a %*% a - b            # Chain of operations
```

[1] http://netlib.org

3.1 Returned values

The package returns a new big.matrix object for operations involving only big.matrix objects.

Operations that involve a mixture of standard R matrices, vectors, or scalars, and a big.matrix may return either an R matrix or a big.matrix depending on the setting of the package option bigalgebra.mixed_arithmetic_returns_R_matrix. The default value of this option is TRUE, which means that operations on mixed-class arguments will allocate and return a standard R matrix or vector. Set the option to FALSE to return results of operations on mixed matrices as new big.matrix objects.

```
# To obtain results as R matrices, set (the default):
options(bigalgebra.mixed_arithmetic_returns_R_matrix=TRUE)
# Here is a tiny example:
> set.seed(1)
> A <- matrix(rnorm(6), nrow=3)
> a <- as.big.matrix(A)
> a %*% t(A)    # Returns an R matrix
          [,1]      [,2]      [,3]
[1,] 2.9373652 0.4106134 -0.7853947
[2,] 0.4106134 0.1423002 -0.4238083
[3,] -0.7853947 -0.4238083 1.3714435

# To obtain resuls as big.matrices, set:
> options(bigalgebra.mixed_airthmetic_returns_R_matrix=FALSE)
> a %*% t(A)    # Now returns a new big.matrix (your pointer address will vary)
An object of class "big.matrix"
Slot "address":
<pointer: 0x1d101b0>
```

3.2 Garbage collection of anonymously-defined big matrices

Some operations performed by the bigalgebra package return "anonymously" allocated big.matrix objects. Such objects are backed by a temporary file defined by the package options options("bigalgebra.tempdir") and options("bigalgebra.temp_pattern"). These big matrices are ephemeral–their backing and descriptor files are unlinked when the corresponding R object is de-allocated by the garbage collector.

You may explicitly observe creation and removal of temporary backing files by setting the option:

```
options(bigalgebra.DEBUG=TRUE)
```

3.3 A practical example

We illustrate using the bigalgebra package together with the irlba package to compute a few singular values and associated singular vectors of a large, dense matrix.

The irlba package[2] (implicitly restarted Lanczos bidiagonalization algoritm) is an R implementation of a state of the art method for efficiently computing a few singular values and singular vectors of a matrix.

The irlba function allows users to supply a function for computing the product of a matrix or its transpose with a vector. We take advantage of this function to avoid explicitly forming the transponse of the (potentially large) big.matrix.

```
> library("bigalgebra")
> library("irlba")

> # Define an efficent matrix/transpose product:
> matmul <- function(A, x, transpose=FALSE)
+ {
+   if(transpose)
+     return(t( t(x) %*% A)) # i.e., t(A) %*% x
+   return (A %*% x)
+ }


# Compute a small example and compare with other methods:
> set.seed(1)
> A  <- matrix(rnorm(100),10)
> a  <- as.big.matrix(A)

# Compute with irlba using a big.matrix:
> La <- irlba(a, nu=2, nv=2, matmul=matmul)

# Compute with irlba using a standard matrix:
> LA <-irlba(A, nu=2, nv=2, matmul=matmul)

# Compute with svd using a standard matrix:
> S  <- svd(A, nu=2, nv=2)

> rbind(La$d,LA$d,S$d[1:2])
         [,1]     [,2]
[1,] 5.154081 4.816501
```

[2] http://cran.r-project.org/web/packages/irlba/index.html

```
[2,] 5.154081 4.816501
[3,] 5.154081 4.816501
```

Note that the singular vectors are unique up to sign (that is, they may have different signs between the answers). Also note that we could use the same matmul function with both `big.matrix` and standard matrix arguments.

This simple example can be used to efficiently compute a few singular values and vectors of very large `big.matrix` objects.

3.4 Large-index support

The following simple example applies if the package was installed with optional large index support as outlined in Section 2. The example sums two vectors with 2^{31} entries, larger than can be indexed by standard R operations. Note that the output vector y will consume approximately 16 GB of disk space.

```
> library("bigalgebra")
> x <- big.matrix(nrow=2^31, ncol=1, type="double", backingfile="x")
> y <- big.matrix(nrow=2^31, ncol=1, type="double", backingfile="y")
> x[5,1] <- 5
> y[5,1] <- 5
> z <- x + y
> print(z[1:10,])
 [1]  0  0  0  0 10  0  0  0  0  0
```

We remark that file-backed big matrices are block-sparse–that is they allocate space for their values in blocks (usually equal to the operating system kernel page size). Only blocks containing values are allocated. Thus, the `x` and `y` matrices in the above example each only consume one page of storage space on disk.