A Package for Matrix Powers in R,
with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

Bay Area R Users Group, August 12, 2014

current slides:
heather.cs.ucdavis.edu/matpow/BARUGmatpow.pdf
Goals of this talk:

• Show how useful matrix powers can be in data science, especially for parallel computation
• Present a small R package that facilitates matrix power computation, including parallel approaches.
• Demonstrate a trick useful for accommodating varied data types.
Goals of this talk:
Goals

Goals of this talk:

• Show how useful matrix powers can be in data science, especially for parallel computation
Goals

Goals of this talk:

• Show how useful matrix powers can be in data science, especially for parallel computation
• Present a small R package that facilitates matrix power computation, including parallel approaches.
Goals of this talk:

- Show how useful matrix powers can be in data science, especially for parallel computation
- Present a small R package that facilitates matrix power computation, including parallel approaches.
- Demonstrate a trick useful for accommodating varied data types.
Why Matrix Powers?

Various apps (see below).

For very large problems, parallel computation is desirable.

If we can recast a large problem in terms of matrix powers, this may yield good (if not optimal) speedup.

Multiplication is easy to parallelize: Matrix multiplication is "embarrassingly parallel."

Mat. mult. works especially well on GPUs.

Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel, so it's good to have embarrassingly parallel alternatives.

R has tons of ways of doing parallel matrix multiplication.

"Pretty Good Parallelism": If can obtain fairly good speedup very conveniently, we may not pursue optimal solutions.
Why Matrix Powers?

Why are matrix powers so important in data science?
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
- For very large problems, parallel computation is desirable.
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
- For very large problems, parallel computation is desirable.
  - If we can recast a large problem in terms of matrix powers, this may yield good (if not optimal) speedup.
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
- For very large problems, parallel computation is desirable.
  - If we can recast a large problem in terms of matrix powers, this may yield good (if not optimal) speedup.
  - Multiplication is easy to parallelize:
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
- For very large problems, parallel computation is desirable.
  - If we can recast a large problem in terms of matrix powers, this may yield good (if not optimal) speedup.
  - Multiplication is easy to parallelize: Matrix multiplication is “embarrassingly parallel.”
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
- For very large problems, parallel computation is desirable.
  - If we can recast a large problem in terms of matrix powers, this may yield good (if not optimal) speedup.
  - Multiplication is easy to parallelize: Matrix multiplication is “embarrassingly parallel.”
  - Mat. mult. works especially well on GPUs.
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
- For very large problems, parallel computation is desirable.
  - If we can recast a large problem in terms of matrix powers, this may yield good (if not optimal) speedup.
  - Multiplication is easy to parallelize: Matrix multiplication is “embarrassingly parallel.”
  - Mat. mult. works especially well on GPUs.
  - Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel,
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
- For very large problems, parallel computation is desirable.
  - If we can recast a large problem in terms of matrix powers, this may yield good (if not optimal) speedup.
  - Multiplication is easy to parallelize: Matrix multiplication is “embarrassingly parallel.”
  - Mat. mult. works especially well on GPUs.
  - Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel, so it’s good to have embarrassingly parallel alternatives.
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
- For very large problems, parallel computation is desirable.
  - If we can recast a large problem in terms of matrix powers, this may yield good (if not optimal) speedup.
  - Multiplication is easy to parallelize: Matrix multiplication is “embarrassingly parallel.”
  - Mat. mult. works especially well on GPUs.
  - Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel, so it’s good to have embarrassingly parallel alternatives.
  - R has tons of ways of doing parallel matrix multiplication.
Why Matrix Powers?

Why are matrix powers so important in data science?

- Various apps (see below).
- For very large problems, parallel computation is desirable.
  - If we can recast a large problem in terms of matrix powers, this may yield good (if not optimal) speedup.
  - Multiplication is easy to parallelize: Matrix multiplication is “embarrassingly parallel.”
  - Mat. mult. works especially well on GPUs.
  - Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel, so it’s good to have embarrassingly parallel alternatives.
  - R has tons of ways of doing parallel matrix multiplication.
  - “Pretty Good Parallelism”: If can obtain fairly good speedup very conveniently, we may not pursue optimal solutions.
Examples of Apps
Examples of Apps

Matrix powers have various applications, e.g.:

- determination of graph connectivity
  For adjacency matrix $A$, the graph is connected if and only if
  for some $k > 0$, $\tilde{A}^k > 0$
  elementwise
  where $\tilde{A}$ is $A$ with all 1s on the diagonal.
  Moreover, the elements of $\tilde{A}^k$ can give you the
  distance from each $i$ to each $j$. 
Examples of Apps

Matrix powers have various applications, e.g.:

- determination of graph connectivity
Examples of Apps

Matrix powers have various applications, e.g.:

- determination of graph connectivity

  For adjacency matrix $A$, the graph is connected if and only if

  $$\text{for some } k > 0, \, \tilde{A}^k > 0 \text{ elementwise}$$

  where $\tilde{A}$ is $A$ with all 1s on the diagonal.
Matrix powers have various applications, e.g.:

- determination of graph connectivity

  For adjacency matrix $A$, the graph is connected if and only if
  
  $$\tilde{A}^k > 0 \text{ elementwise}$$

  where $\tilde{A}$ is $A$ with all 1s on the diagonal. Moreover, the elements of $\tilde{A}^k$ can give you the distance from each $i$ to each $j$. 
• (new app?) finding stationary distribution $\pi$ of a finite, aperiodic Markov chain
• (new app?) finding stationary distribution $\pi$ of a finite, aperiodic Markov chain

\[ \text{Exploit the fact that} \]
\[ \lim_{n \to \infty} P(X_n = j | X_0 = i) = \pi_j. \text{ It implies that} \]
\[ \text{for transition matrix } P, \ \pi \ \text{vector is approximately} \]
\[ \text{pivec} <- \ \text{colMeans}(P^k) \]

\[ \text{Could also adapt the graph-connect method to determine periodicity of a finite chain.} \]
• (principal) eigenvector computation
• (principal) eigenvector computation

For “most” square matrices $A$ and initial guess vectors $x$, 

$$
\frac{A^k x}{\|A^k x\|}
$$

converges to the principal eigenvector of $A$. 
So, set an initial $x$, then iterate $x \leftarrow Ax/\|Ax\|$. 

(principal) eigenvector computation

For “most” square matrices $A$ and initial guess vectors $x$,

$$
\frac{A^k x}{\|A^k x\|}
$$

converges to the principal eigenvector of $A$.

So, set an initial $x$, then iterate $x \leftarrow Ax/\|Ax\|$.

computation of generalized matrix inverse

Iterate $B \leftarrow B(2I - AB)$, starting with $B$ a small multiple of $A'$.
A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

R Package: matpow

• We have developed a small but convenient and general package for computation of matrix powers, matpow, whether done serially or in parallel.
• Key feature: Allows callback functions after each iteration.
• E.g. graph connectivity app: Callback checks to see if all \( \tilde{A}^i \) are already > 0, can stop iterating. Or, an element changes from 0 to > 0, we know that is the shortest distance.
• Form of call (raise matrix \( m \) to power \( k \)): matpow<function( m, k=NULL, squaring=FALSE, genmulcmd=NULL, dup=NULL, callback=NULL, ... )
R Package: matpow

- We have developed a *small but convenient and general package* for computation of matrix powers, *matpow*,

```r
matpow <- function(m, k = NULL, squaring = FALSE, genmulcmd = NULL, dup = NULL, callback = NULL, ...)
```
R Package: matpow

- We have developed a **small but convenient and general package** for computation of matrix powers, **matpow**, whether done serially or in parallel.
A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

R Package: matpow

- We have developed a **small but convenient and general package** for computation of matrix powers, **matpow**, whether done serially or in parallel.
- Key feature: Allows callback functions after each iteration.
R Package: matpow

- We have developed a **small but convenient and general package** for computation of matrix powers, **matpow**, whether done serially or in parallel.
- Key feature: Allows callback functions after each iteration.
- E.g. graph connectivity app:
R Package: matpow

- We have developed a **small but convenient and general package** for computation of matrix powers, **matpow**, whether done serially or in parallel.
- Key feature: Allows callback functions after each iteration.
- E.g. graph connectivity app: Callback checks to see if all of $\tilde{A}^i$ are already $> 0$, can stop iterating.
R Package: matpow

- We have developed a **small but convenient and general package** for computation of matrix powers, **matpow**, whether done serially or in parallel.

- Key feature: Allows callback functions after each iteration.

- E.g. graph connectivity app: Callback checks to see if all of $\tilde{A}^i$ are already $> 0$, can stop iterating. Or, an element changes from 0 to $> 0$, we know that is the shortest distance.

- Form of call (raise matrix m to power k):
A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis
e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

**R Package: matpow**

- We have developed a **small but convenient and general package** for computation of matrix powers, **matpow**, whether done serially or in parallel.
- Key feature: Allows callback functions after each iteration.
- E.g. graph connectivity app: Callback checks to see if all of $A^i$ are already $> 0$, can stop iterating. Or, an element changes from 0 to $> 0$, we know that is the shortest distance.
- Form of call (raise matrix m to power k):

```r
matpow <- function (m, k=NULL, squaring=FALSE, genmulcmd=NULL, dup=NULL, callback=NULL, ...) {
```

Powers by Squaring

A Package for Matrix Powers in R,
with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

Powers by Squaring

Say you want to find $M^8$. You could square $M$, then square the result, then square that result. Thus get $M^k$ in about $\log_2 k$ steps.

Example: Good for determining matrix connectivity, but not for finding the minimum distances.

In call to `matpow()`, set `squaring = TRUE`. 
Powers by Squaring

- Say you want to find $M^8$. 

A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com
Powers by Squaring

• Say you want to find $M^8$. You could square $M$, 

Powers by Squaring

- Say you want to find $M^8$. You could square $M$, then square the result,
Powers by Squaring

• Say you want to find $M^8$. You could square M, then square the result, then square that result.
Powers by Squaring

- Say you want to find $M^8$. You could square $M$, then square the result, then square that result.
- Thus get $M^k$ in about $\log_2 k$ steps.
Powers by Squaring

- Say you want to find $M^8$. You could square $M$, then square the result, then square that result.
- Thus get $M^k$ in about $\log_2 k$ steps.
- Example: Good for determining matrix connectivity, but not for finding the minimum distances.
Powers by Squaring

• Say you want to find $M^8$. You could square $M$, then square the result, then square that result.

• Thus get $M^k$ in about $\log_2 k$ steps.

• Example: Good for determining matrix connectivity, but not for finding the minimum distances.

• In call to matpow(), set squaring = TRUE.
Sharing Data
Sharing Data

**Issue:** How do we arrange TWO-WAY communication between `matpow()` and the callback function (if any)?
Sharing Data

**Issue:** How do we arrange TWO-WAY communication between `matpow()` and the callback function (if any)? Can NOT use an R list. E.g.

```r
> l <- list(x=3, y=8)
> f
function(lst) {
  l$x[1] <- 88
}
> f(l)
> l$x
[1] 3  # didn’t change!
```

R makes copies of arguments, if they are changed by the function. The change to `l` was to the copy, not to the original.
Sharing Data

**Issue:** How do we arrange TWO-WAY communication between `matpow()` and the callback function (if any)?

Can NOT use an R list. E.g.

```r
> l <- list(x=3, y=8)
> f
function(lst) {
  lst$x[1] <- 88
}
> f(l)
> l$x
[1] 3  # didn't change!
```

R makes copies of arguments, if they are changed by the function.
Sharing Data

**Issue:** How do we arrange TWO-WAY communication between `matpow()` and the callback function (if any)? Can NOT use an R list. E.g.

```r
> l ← list(x=3, y=8)
> f
function(lst) {
  l$x[1] ← 88
}
> f(l)
> l$x
[1] 3  # didn’t change!
```

R makes copies of arguments, if they are changed by the function. The change to `l` was to the copy, not to the original.
R Environments

Like lists, but R doesn't copy them when used as arguments. The function `matpow()` maintains an R environment `ev`, accessible to the callback function. Most important: The callback can change components of `ev`. (Could use R reference classes to be fancy.) Contents of `ev`:

• the matrix `m`
• the target exponent `k`
• `i`, the current iteration number
• `stop`; TRUE means stop iterations
• `squaring` etc.
• app-specific data
Like lists, but R doesn’t copy them when used as arguments.
R Environments

Like lists, but R doesn’t copy them when used as arguments. The function `matpow()` maintains an R environment `ev`, accessible to the callback function.
Like lists, but R doesn’t copy them when used as arguments. The function `matpow()` maintains an R environment `ev`, accessible to the callback function. **Most important:** The callback can change components of `ev`. 
R Environments

Like lists, but R doesn’t copy them when used as arguments. The function `matpow()` maintains an R environment `ev`, accessible to the callback function. **Most important:** The callback can change components of `ev`. (Could use R reference classes to be fancy.)
Like lists, but R doesn’t copy them when used as arguments. The function `matpow()` maintains an R environment `ev`, accessible to the callback function. **Most important:** The callback can change components of `ev`. (Could use R reference classes to be fancy.)

Contents of `ev`:

- the matrix `m`
- the target exponent `k`
- `i`, the current iteration number
- `stop`; TRUE means stop iterations
- squaring
- etc.
- app-specific data
R Environments

Like lists, but R doesn’t copy them when used as arguments. The function `matpow()` maintains an R environment `ev`, accessible to the callback function. **Most important:** The callback can change components of `ev`. (Could use R reference classes to be fancy.)

Contents of `ev`:

- the matrix `m`
- the target exponent `k`
- `i`, the current iteration number
- `stop`; TRUE means stop iterations
- `squaring`
- etc.
- app-specific data
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.

- Example: Graph connectivity and distance computation. The callback `cgraph()` does the following:
  - Checks to see if all elements > 0. If so, sets `ev$stop` to `TRUE`, indicating graph found to be connected.
  - Optionally checks if product element (i,j) changed from 0 to nonzero in this iteration. If so, then records that the distance from i to j is `ev$i + 1`.

- Example: Eigenvalue computation.
  - Updates `ev$x`, via `x ← Ax / ∥Ax∥`.
  - If convergence reached (< ϵ change), sets `ev$stop` to `TRUE`. 
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers.
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.

- **Example:** Graph connectivity and distance computation.
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.

- **Example:** Graph connectivity and distance computation. The callback `cgraph()` does the following:
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.

- **Example:** Graph connectivity and distance computation. The callback `cgraph()` does the following:
  - Checks to see if all elements $> 0$. If so, sets `ev$stop` to `TRUE`, indicating graph found to be connected.
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.

- **Example:** Graph connectivity and distance computation. The callback `cgraph()` does the following:
  - Checks to see if all elements > 0. If so, sets `ev$stop` to TRUE, indicating graph found to be connected.
  - Optionally checks if product element (i,j) changed from 0 to nonzero in this iteration.
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.

- **Example:** Graph connectivity and distance computation. The callback `cgraph()` does the following:
  - Checks to see if all elements > 0. If so, sets `ev$stop` to TRUE, indicating graph found to be connected.
  - Optionally checks if product element (i,j) changed from 0 to nonzero in this iteration. If so, then records that the distance from i to j is `ev$i + 1`. 
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.

- **Example:** Graph connectivity and distance computation. The callback `cgraph()` does the following:
  - Checks to see if all elements $> 0$. If so, sets `ev$stop` to TRUE, indicating graph found to be connected.
  - Optionally checks if product element $(i,j)$ changed from 0 to nonzero in this iteration. If so, then records that the distance from $i$ to $j$ is $ev$i + 1.

- **Example:** Eigenvalue computation.
  - Updates `ev$x`, via $x ← Ax/∥Ax∥$. If convergence reached ($< \epsilon$ change), sets `ev$stop` to TRUE.
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.

- **Example:** Graph connectivity and distance computation. The callback `cgraph()` does the following:
  - Checks to see if all elements $> 0$. If so, sets `ev$stop` to TRUE, indicating graph found to be connected.
  - Optionally checks if product element $(i,j)$ changed from 0 to nonzero in this iteration. If so, then records that the distance from $i$ to $j$ is `ev$i + 1`.

- **Example:** Eigenvalue computation.
  - Updates `ev$x`, via $x \leftarrow Ax/\|Ax\|$. 
The Key Role of Callbacks

Our goal is to provide a convenient general framework for diverse applications of matrix powers. Key to this is the callback functions.

- **Example:** Graph connectivity and distance computation. The callback `cgraph()` does the following:
  - Checks to see if all elements $> 0$. If so, sets `ev$stop` to TRUE, indicating graph found to be connected.
  - Optionally checks if product element $(i,j)$ changed from 0 to nonzero in this iteration. If so, then records that the distance from $i$ to $j$ is $ev$i + 1$.

- **Example:** Eigenvalue computation.
  - Updates `ev$x`, via $x \leftarrow Ax/\|Ax\|$.
  - If convergence reached ($< \epsilon$ change), sets `ev$stop` to TRUE.
Example Callback: Graph Connectivity
Example Callback: Graph Connectivity

```
matpow(m, k, callback=cgraph, mindist=TRUE)
...

cgraph <- 
    function(ev, cbinit=FALSE, mindist=FALSE) {
      if (cbinit) {
        ev$dists <- ev$m
        return()
      }
      if (all(ev$prd > 0)) {
        ev$stop <- TRUE
      }
      if (mindist) {
        tmp <- ev$prd > 0
        ev$dists[tmp & ev$dists == 0] <- ev$i+1
      }
    }
```
Use of eval()
Use of eval()

**Issue:** Different matrix types use different syntax for multiplication.

• plain R “matrix” class: \( c \left< - a \%\% b \right. 

• bigmemory “big.matrix” class: \( a \left[ , \right] \%\% b \left[ , \right] 

• gputools multiplication: \( c \left< - \text{gpuMatMult}(a, b) \right. 

We want to be able to handle other matrix multiplication types too, including user-defined ones. How?
Use of `eval()`

**Issue:** Different matrix types use different syntax for multiplication.

- **plain R ”matrix” class:**
  ```r
c <- a %*% b
  ```

We want to be able to handle other matrix multiplication types too, including user-defined ones. How?
Use of `eval()`

**Issue:** Different matrix types use different syntax for multiplication.

- **plain R "matrix" class:**
  ```r
c <- a %*% b
  ```
- **bigmemory "big.matrix" class:**
  ```r
c [,] <- a [,] %*% b [,]
  ```
Use of eval()

**Issue:** Different matrix types use different syntax for multiplication.

- **plain R ”matrix” class:**
  ```r
c <- a %*% b
  ```

- **bigmemory ”big.matrix” class:**
  ```r
c[ , ] <- a[ , ] %*% b[ , ]
  ```

- **gputools multiplication:**
  ```r
c <- gpuMatMult(a, b)
  ```
Use of eval()

**Issue:** Different matrix types use different syntax for multiplication.

- **plain R ”matrix” class:**
  \[ c <- a \%*\% b \]

- **bigmemory ”big.matrix” class:**
  \[ c [ , ] <- a [ , ] \%*\% b [ , ] \]

- **gputools multiplication:**
  \[ c <- gpuMatMult(a, b) \]

We want to be able to handle other matrix multiplication types too, including user-defined ones.
Use of eval()

**Issue**: Different matrix types use different syntax for multiplication.

- **plain R ”matrix” class**:
  
  ```
  c <- a %*% b
  ```

- **bigmemory ”big.matrix” class**:
  
  ```
  c [,] <- a [,] %*% b [,]
  ```

- **gputools multiplication**:
  
  ```
  c <- gpuMatMult(a, b)
  ```

We want to be able to handle other matrix multiplication types too, including user-defined ones. How?
A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

R’s eval() function

```r
x <- 28
s <- "x <- 16"

eval(parse(text=s))
```

So, we can embed the different types of matrix multiplication in strings!
R’s eval() function

```r
> x <- 28
> s <- "x <- 16"
> eval(parse(text=s))
> x
[1] 16
```

So, we can embed the different types of matrix multiplication in strings!
R’s eval() function

```r
> x <- 28
> s <- "x <- 16"
> eval(parse(text=s))
> x
[1] 16
```

So, we can embed the different types of matrix multiplication in strings!
A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

Recall form of call: matpow <- function(m, k=NULL, squaring=FALSE, genmulcmd=NULL, dup=NULL, callback=NULL, ...)

E.g. genmulcmd.gputools <- function(a, b, c) paste(c, '<', 'gpuMatMult(', a, ',', b, ')')

So matpow() code can be general, e.g. eval(parse(text=eval$genmulcmd(m, p1, p2)))
The function genmulcmd() is either sensed by matrix class or specified by user.
Recall form of call:
Recall form of call:

```r
matpow <- function(m, k=NULL, squaring=FALSE,
genmulcmd=NULL, dup=NULL, callback=NULL, ...) {
```

E.g.

```r
genmulcmd.gputools <- function(a, b, c)
paste(c, "<--gpuMatMult(" , a , ", " , b , ")")
```

So `matpow()` code can be general, e.g.

```r
eval(parse(text=eval$genmulcmd(m, p1, p2))
```
Recall form of call:

\[
\text{matpow} \leftarrow \text{function}(m, k=NULL, squaring=FALSE, \text{genmulcmd}=NULL, \text{dup}=NULL, \text{callback}=NULL, \ldots) \text{ } \{ \\
\text{E.g.} \\
\text{genmulcmd.gputools} \leftarrow \text{function}(a, b, c) \\
\text{paste}(c, \text{" <- gputMatMult(\" ,a,\" ,\" ,b,\")")}
\]

A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com
A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

Recall form of call:

```r
matpow <- function(m, k=NULL, squaring=FALSE,
   genmulcmd=NULL, dup=NULL, callback=NULL, ...)
```

E.g.

```r
genmulcmd.gputools <- function(a, b, c)
   paste(c," <- gpuMatMult(" , a , "," , b , ")")
```

So `matpow()` code can be general, e.g.

```r
eval(parse(text=ev$genmulcmd(m, p1, p2)))
```
Recall form of call:

```
matpow <- function(m, k=NULL, squaring=FALSE, 
                  genmulcmd=NULL, dup=NULL, callback=NULL, . . . ) {
```

E.g.

```
genmulcmd.gputools <- function(a, b, c)
  paste(c, " <- gpuMatMult(" , a , " , " , b , " )")
```

So `matpow()` code can be general, e.g.

```
eval(parse(text=eval$genmulcmd(m, p1, p2))
```

The function `genmulcmd()` is either sensed by matrix class or specified by user.
Parallel Operation

We wish to emphasize: The package is useful for BOTH serial AND parallel computation. But let's talk about the parallel case.

- The `matpow()` function handles whatever type of multiplication you give. So, if you give it a parallel multiplication, you compute matrix powers in parallel!
- Example: If you have configured R to use OpenBLAS, your multiplications will use all the cores.
- Example: GPU, say with `gputools`.
Parallel Operation

We wish to emphasize: The package is useful for BOTH serial AND parallel computation.
Parallel Operation

We wish to emphasize: The package is useful for BOTH serial AND parallel computation.
But let’s talk about the parallel case.
Parallel Operation

We wish to emphasize: The package is useful for BOTH serial AND parallel computation. But let’s talk about the parallel case.

- The \texttt{matpow()} function handles whatever type of multiplication you give.
We wish to emphasize: The package is useful for BOTH serial AND parallel computation. But let’s talk about the parallel case.

- The **matpow()** function handles whatever type of multiplication you give. So, if you give it a parallel multiplication, you compute matrix powers in parallel!
- Example: If you have configured R to use OpenBLAS, your multiplications will use all the cores.
Parallel Operation

We wish to emphasize: The package is useful for BOTH serial AND parallel computation. But let’s talk about the parallel case.

- The `matpow()` function handles whatever type of multiplication you give. So, if you give it a parallel multiplication, you compute matrix powers in parallel!
- Example: If you have configured R to use OpenBLAS, your multiplications will use all the cores.
- Example: GPU, say with `gputools`.
Brief Timing Experiment with gputools

• About 20X speedup due to GPU.
• Lots of overhead in the case $k = 2$. 
Brief Timing Experiment with gputools

Modest hardware: Intel Core i7-2600K CPU, 3.40GHz, GeForce GTX 550 T
Brief Timing Experiment with gputools

Modest hardware: Intel Core i7-2600K CPU, 3.40GHz, GeForce GTX 550 T
2000 × 2000 matrix

• About 20X speedup due to GPU.
• Lots of overhead in the case $k = 2$. 
Brief Timing Experiment with gputools

Modest hardware: Intel Core i7-2600K CPU, 3.40GHz, GeForce GTX 550 Ti
2000 \times 2000 matrix

<table>
<thead>
<tr>
<th>k</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6.134</td>
<td>1.836</td>
</tr>
<tr>
<td>3</td>
<td>12.626</td>
<td>0.620</td>
</tr>
<tr>
<td>4</td>
<td>18.981</td>
<td>0.930</td>
</tr>
<tr>
<td>5</td>
<td>25.222</td>
<td>1.235</td>
</tr>
</tbody>
</table>

• About 20X speedup due to GPU.
• Lots of overhead in the case $k = 2$. 

A Package for Matrix Powers in R, with Some Edifying Material on R
Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com
A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis
e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

Brief Timing Experiment with gputools

Modest hardware: Intel Core i7-2600K CPU, 3.40GHz, GeForce GTX 550 Ti
2000 × 2000 matrix

<table>
<thead>
<tr>
<th>k</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6.134</td>
<td>1.836</td>
</tr>
<tr>
<td>3</td>
<td>12.626</td>
<td>0.620</td>
</tr>
<tr>
<td>4</td>
<td>18.981</td>
<td>0.930</td>
</tr>
<tr>
<td>5</td>
<td>25.222</td>
<td>1.235</td>
</tr>
</tbody>
</table>

- About 20X speedup due to GPU.
Brief Timing Experiment with gputools

Modest hardware: Intel Core i7-2600K CPU, 3.40GHz, GeForce GTX 550 T
2000 × 2000 matrix

<table>
<thead>
<tr>
<th>k</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6.134</td>
<td>1.836</td>
</tr>
<tr>
<td>3</td>
<td>12.626</td>
<td>0.620</td>
</tr>
<tr>
<td>4</td>
<td>18.981</td>
<td>0.930</td>
</tr>
<tr>
<td>5</td>
<td>25.222</td>
<td>1.235</td>
</tr>
</tbody>
</table>

- About 20X speedup due to GPU.
- Lots of overhead in the case \( k = 2 \).
A Package for Matrix Powers in R, with Some Edifying Material on R

Norm Matloff and Jack Norman
University of California at Davis

e-mail: matloff@cs.ucdavis.edu
R/stat blog: matloff.wordpress.com

Issues

• In gputools, the current power must be copied from CPU to GPU each time!
• Would be faster to write a different interface to CUBLAS that leaves the power on the CPU at each iteration.
• Same for cluster use: The genmulcmd() function should be written to leave the powers at the cluster nodes. Actually, should have each node maintain a chunk of rows of the current power.
Issues

- In *gputools*, the current power must be copied from CPU to GPU each time!
Issues

- In *gputools*, the current power must be copied from CPU to GPU each time!
- Would be faster to write a different interface to CUBLAS that leaves the power on the CPU at each iteration.
Issues

- In `gputools`, the current power must be copied from CPU to GPU each time!
- Would be faster to write a different interface to CUBLAS that leaves the power on the CPU at each iteration.
- Same for cluster use: The `genmulcmd()` function should be written to leave the powers at the cluster nodes.
Issues

- In `gputools`, the current power must be copied from CPU to GPU each time!
- Would be faster to write a different interface to CUBLAS that leaves the power on the CPU at each iteration.
- Same for cluster use: The `genmulcmd()` function should be written to leave the powers at the cluster nodes. Actually, should have each node maintain a chunk of rows of the current power.
Conclusions

Matrix powers have lots of uses.
• Especially useful in parallel contexts, due to fast matrix multiplication.
• Our matpow package provides a convenient tool for matrix powers apps (including serial computation).
• Further work will be done to supply genmulcmd() functions for other types of matrix multiplication, e.g. for clusters.

Location of the code and these slides:
http://heather.cs.ucdavis.edu/matpow/
Conclusions

- Matrix powers have lots of uses.
Conclusions

- Matrix powers have lots of uses.
- Especially useful in parallel contexts, due to fast matrix multiplication.
Conclusions

- Matrix powers have lots of uses.
- Especially useful in parallel contexts, due to fast matrix multiplication.
- Our matpow package provides a convenient tool for matrix powers apps (including serial computation).
Conclusions

- Matrix powers have lots of uses.
- Especially useful in parallel contexts, due to fast matrix multiplication.
- Our `matpow` package provides a convenient tool for matrix powers apps (including serial computation).
- Further work will be done to supply `genmulcmd()` functions for other types of matrix multiplication, e.g. for clusters.
Conclusions

- Matrix powers have lots of uses.
- Especially useful in parallel contexts, due to fast matrix multiplication.
- Our `matpow` package provides a convenient tool for matrix powers apps (including serial computation).
- Further work will be done to supply `genmulcmd()` functions for other types of matrix multiplication, e.g. for clusters.

Location of the code and these slides:
http://heather.cs.ucdavis.edu/matpow/