A Package for Matrix Powers in R

Norm Matloff and Jack Norman
University of California at Davis

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

useR! 2014
UCLA
July 2, 2014
A Package for Matrix Powers in R

Norm Matloff and Jack Norman
University of California at Davis

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

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 (parallel) matrix power computation, and includes several apps.
Goals of this talk:
Goals of this talk:

- Show how useful matrix powers can be in data science, especially for parallel computation
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 (parallel) matrix power computation, and includes several apps.
Why Matrix Powers?

Why are matrix powers so important in the context of parallel computation?

- Matrix multiplication is "embarrassingly parallel."
- Works especially well on GPUs.
- Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel.
- R has tons of ways of doing parallel matrix multiplication.
- Can exploit R's polymorphic nature: `%*%` means `%*%`, whether for the ordinary R matrix class, the gmatrix class, the Matrix class, etc.
  So, the same power-computing software can work on all of them.

(Some hedging on this later.)
Why Matrix Powers?

Why are matrix powers so important in the context of parallel computation?
Why Matrix Powers?

Why are matrix powers so important in the context of parallel computation?

- Matrix multiplication is “embarrassingly parallel.”
Why Matrix Powers?

Why are matrix powers so important in the context of parallel computation?

- Matrix multiplication is “embarrassingly parallel.”
- Works especially well on GPUs.
Why Matrix Powers?

Why are matrix powers so important in the context of parallel computation?

- Matrix multiplication is “embarrassingly parallel.”
- 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 the context of parallel computation?

- Matrix multiplication is “embarrassingly parallel.”
- Works especially well on GPUs.
- Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel.
- R has tons of ways of doing parallel matrix multiplication.
Why Matrix Powers?

Why are matrix powers so important in the context of parallel computation?

- Matrix multiplication is “embarrassingly parallel.”
- Works especially well on GPUs.
- Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel.
- R has tons of ways of doing parallel matrix multiplication.
- Can exploit R’s polymorphic nature:
Why Matrix Powers?

Why are matrix powers so important in the context of parallel computation?

- Matrix multiplication is “embarrassingly parallel.”
- Works especially well on GPUs.
- Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel.
- R has tons of ways of doing parallel matrix multiplication.
- Can exploit R’s *polymorphic* nature: %*% means %*%, whether for the ordinary R **matrix** class, the **gmatrix** class, the **Matrix** class, etc.
Why Matrix Powers?

Why are matrix powers so important in the context of parallel computation?

- Matrix multiplication is “embarrassingly parallel.”
- Works especially well on GPUs.
- Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel.
- R has tons of ways of doing parallel matrix multiplication.
- Can exploit R’s polymorphic nature: %*% means %*%, whether for the ordinary R matrix class, the gmatrix class, the Matrix class, etc.

So, the same power-computing software can work on all of them.
Why Matrix Powers?

Why are matrix powers so important in the context of parallel computation?

- Matrix multiplication is “embarrassingly parallel.”
- Works especially well on GPUs.
- Ordinary matrix inversion (e.g. Gaussian elimination) and quasi-inversion (e.g. QR) are not embarrassingly parallel.
- R has tons of ways of doing parallel matrix multiplication.
- Can exploit R’s polymorphic nature: `%*%` means `%*%`, whether for the ordinary R `matrix` class, the `gmatrix` class, the `Matrix` class, etc.

So, the same power-computing software can work on all of them. (Some hedging on this later.)
Matrix powers have various applications, e.g.:

- determination of graph connectivity
  For adjacency matrix $A$, the graph is connected if and only if
  $\tilde{A}^k > 0$ 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 $\tilde{A}^k > 0$ elementwise, where $\tilde{A}$ is $A$ with all 1s on the diagonal.
Matrix powers have various applications, e.g.:

- determination of graph connectivity
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.
• (new app?) finding stationary distribution $\pi$ of a finite, aperiodic Markov chain
• (new app?) finding stationary distribution $\pi$ of a finite, aperiodic Markov chain

$\textit{Exploit the fact that}$

$\lim_{n \to \infty} P(X_n = j | X_0 = i) = \pi_j$. \textit{It implies that for transition matrix} $P$, $\pi$ vector is approximately

$pivec \leftarrow \text{colMeans}(P^k)$

$\textit{Could also adapt the graph-connect method to determine periodicity of a finite chain.}$
• (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 $A^\prime$. 
We have developed a small but convenient and general package for parallel (or serial) computation of matrix powers, `parmatpows`.

Works on any matrix class supporting `%*%`.

Key feature: Allows callback functions after each iteration.

Form of call (raise matrix \( m \) to power \( k \)):

```r
parpowm(m, k, squaring=FALSE, callback=NULL, ...)
```

Set `squaring` to `TRUE` if just need a large power, not any exponent in particular.
R Package: parmatpows

- We have developed a **small but convenient and general package** for parallel (or serial) computation of matrix powers, **parmatpows**.
R Package: parmatpows

- We have developed a **small but convenient and general package** for parallel (or serial) computation of matrix powers, **parmatpows**.
- Works on any matrix class supporting \%\*\%. 

---

A Package for Matrix Powers in 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: parmatpows

- We have developed a **small but convenient and general package** for parallel (or serial) computation of matrix powers, **parmatpows**.
- Works on any matrix class supporting %*%.
- Key feature: Allows callback functions after each iteration.
A Package for Matrix Powers in 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: parmatpows

- We have developed a small but convenient and general package for parallel (or serial) computation of matrix powers, parmatpows.
- Works on any matrix class supporting %*%.
- Key feature: Allows callback functions after each iteration.
- Form of call (raise matrix m to power k):
We have developed a **small but convenient and general package** for parallel (or serial) computation of matrix powers, **parmatpows**.

- Works on any matrix class supporting `%*%`.
- Key feature: Allows callback functions after each iteration.
- Form of call (raise matrix \( m \) to power \( k \)):

  ```r
  parpom(m, k, squaring=FALSE, callback=NULL, ...)
  ```
We have developed a small but convenient and general package for parallel (or serial) computation of matrix powers, parmatpows.

- Works on any matrix class supporting \%\*\%.
- Key feature: Allows callback functions after each iteration.
- Form of call (raise matrix m to power k):
  \[
  \text{parpowm}(m, k, \text{squaring}=\text{FALSE}, \text{callback}=\text{NULL}, \ldots)
  \]

Set \text{squaring} to TRUE if just need a large power, not any exponent in particular.
A Package for Matrix Powers in R

Norm Matloff and Jack Norman
University of California at Davis

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

Sharing Data
Sharing Data

The function `parpowm()` maintains an R environment `ev`, accessible to the callback function.
Sharing Data

The function `parpowm()` maintains an R environment `ev`, accessible to the callback function.

Contents:

- the matrix `m`
- the target exponent `k`
- `i`, the current iteration number
- the current power of `m`, `prd`
- `stop`; TRUE means stop iterations
- squaring
- app-specific data

The function returns `ev`. Thus one can obtain the final power from `ev$prd`, check how many iterations were needed via `ev$i`, etc.
Sharing Data

The function `parpowm()` maintains an R environment `ev`, accessible to the callback function.

Contents:

- the matrix `m`
- the target exponent `k`
- `i`, the current iteration number
- the current power of `m`, `prd`
- `stop`; TRUE means stop iterations
- `squaring`
- app-specific data

The function returns `ev`. Thus one can obtain the final power from `ev$prd`, check how many iterations were needed via `ev$i`, etc.
Sharing Data

The function `parpowm()` maintains an R environment `ev`, accessible to the callback function.

Contents:

- the matrix `m`
- the target exponent `k`
- `i`, the current iteration number
- the current power of `m`, `prd`
- `stop`; TRUE means stop iterations
- `squaring`
- app-specific data

The function returns `ev`. 
Sharing Data

The function `parpowm()` maintains an R environment `ev`, accessible to the callback function.

Contents:

- the matrix \( m \)
- the target exponent \( k \)
- \( i \), the current iteration number
- the current power of \( m \), \( prd \)
- `stop`; TRUE means stop iterations
- `squaring`
- app-specific data

The function returns `ev`. Thus one can obtain the final power from `ev$prd`, check how many iterations were needed via `ev$i`, etc.
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.
  - The callback `eig()`:
    - Updates `ev$x`, via $x \leftarrow Ax / \|Ax\|$
    - If convergence reached, 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. 

```r
# Example code
A <- matrix(c(1, 2, 3, 2, 5, 6, 3, 6, 1), nrow = 3, byrow = TRUE)
result <- cgraph(A)
```
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.
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.
  The callback `eig()`:
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. The callback `eig()`:
  - Updates `ev$x`, via $x ← Ax/\|Ax\|$. 

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. The callback `eig()`:
  - Updates `ev$x`, via $x \leftarrow Ax/\|Ax\|$.
  - If convergence reached, sets `ev$stop` to `TRUE`. 
Example Callback: Graph Connectivity
Example Callback: Graph Connectivity

\[
\text{parpowm}(m, k, \text{callback} = \text{cgraph}, \text{mindist} = \text{TRUE})
\]

\[
c\text{graph} \leftarrow \begin{array}{l}
\text{function}(ev, \text{cbinit} = \text{FALSE}, \text{mindist} = \text{FALSE}) \{ \\
\text{if (cbinit)} \{ \\
\text{ev}$\text{dists} \leftarrow \text{ev}$m \\
\text{return}() \\
\} \\
\text{if (all(ev}$\text{prd} > 0)) \{ \\
\text{ev}$\text{stop} \leftarrow \text{TRUE} \\
\} \\
\text{if (mindist)} \{ \\
\text{tmp} \leftarrow \text{ev}$\text{prd} > 0 \\
\text{ev}$\text{dists}[\text{tmp} \& \text{ev}$\text{dists} == 0] \leftarrow \text{ev}$i+1 \\
\} \\
\}\end{array}
\]
Example Callback: Eigenvalue Computation
Example Callback: Eigenvalue Computation

```r
eig <- function (ev, cbinit=FALSE, x=NULL) {
  m <- ev$m
  if (cbinit) {
    if (is.null(x)) ev$x <- rep(1, nrow(m))
    return()
  }
  mx <- m %*% ev$x
  nx <- sqrt(as.numeric(t(mx) %*% mx))
  ev$x <- (1/nx) * mx
  ev$lamb <- nx
}
```
Current Apps
Current Apps

Callbacks currently included in the package:
Current Apps

Callbacks currently included in the package:

- `graph()`
- `eig()`
- `markov()`
A Package for Matrix Powers in R

Norm Matloff and Jack Norman
University of California at Davis

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

Example of Speedup: Markov chain solution
A Package for Matrix Powers in R

Norm Matloff and Jack Norman
University of California at Davis

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

Example of Speedup: Markov chain solution

- Markov chain, 5500 x 5500 matrix
Example of Speedup: Markov chain solution

- Markov chain, 5500 x 5500 matrix
- CPU: Intel Core i7-2600K CPU, 3.40GHz
Example of Speedup: Markov chain solution

- Markov chain, 5500 x 5500 matrix
- CPU: Intel Core i7-2600K CPU, 3.40GHz
- GPU: GeForce GTX 550 T, accessed via `gmatrix`
Example of Speedup: Markov chain solution

- Markov chain, 5500 x 5500 matrix
- CPU: Intel Core i7-2600K CPU, 3.40GHz
- GPU: GeForce GTX 550 T, accessed via \texttt{gmatrix}
- CPU, using R \texttt{solve}():
  16.974s
Example of Speedup: Markov chain solution

- Markov chain, 5500 x 5500 matrix
- CPU: Intel Core i7-2600K CPU, 3.40GHz
- GPU: GeForce GTX 550 T, accessed via gmatrix
- CPU, using R `solve()`:
  16.974s
- GPU, using `parpowm()`:
  11.481s

Why the weird number, 5500? To explained later.
Example of Speedup: Markov chain solution

- Markov chain, 5500 x 5500 matrix
- CPU: Intel Core i7-2600K CPU, 3.40GHz
- GPU: GeForce GTX 550 Ti, accessed via `gmatrix`
- CPU, using R `solve()`:
  16.974s
- GPU, using `parpowm()`:
  11.481s
- Why the weird number, 5500? To explained later.
Example of Speedup: Principal Eigenvalue
Example of Speedup: Principal Eigenvalue

Serial CPU vs. GPU, using `gmatrix`, 2000 × 2000 matrix:

```r
> system.time(z1 <- eigen(m))
  user  system elapsed
 83.499  0.070  84.024
> mg <- gmatrix(m, ncol=n)
> system.time(eig(z2 <- parpowm(mg, 50, 
+ callback=eig)))
  user  system elapsed
 0.539  0.307  0.847
```
Example of Speedup: Principal Eigenvalue

Serial CPU vs. GPU, using \texttt{gmatrix}, 2000 \times 2000 matrix:

\begin{verbatim}
> system.time(z1 <- eigen(m))

  user  system elapsed
 83.499   0.070   84.024

> mg <- gmatrix(m, ncol=n)

> system.time(eig(z2 <- parpowm(mg, 50,
+ callback=eig)))

  user  system elapsed
  0.539   0.307   0.847
\end{verbatim}

Disclaimer on next page!
One Eigenvalue, a Few, or All?

• Of course, the above was an unfair comparison. The `eigen()` function found all the eigenvalues, whereas we found just one.

• But often we want just the first eigenvalue, or the first few. After finding the largest eigenvalue $\lambda_1$, with associated eigenvector $v_1$, form $B = A - \lambda_1 v_1 v_1'$, and repeat, to get second eigenvalue, etc.
One Eigenvalue, a Few, or All?

- Of course, the above was an unfair comparison.
One Eigenvalue, a Few, or All?

- Of course, the above was an unfair comparison. The `eigen()` function found all the eigenvalues, whereas we found just one.
One Eigenvalue, a Few, or All?

- Of course, the above was an unfair comparison. The `eigen()` function found all the eigenvalues, whereas we found just one.
- But often we want just the first eigenvalue, or the first few.
One Eigenvalue, a Few, or All?

- Of course, the above was an unfair comparison. The \texttt{eigen()} function found all the eigenvalues, whereas we found just one.
- But often we want just the first eigenvalue, or the first few.
- After finding the largest eigenvalue $\lambda_1$, with associated eigenvector $v_1$, form $B = A - \lambda_1 v_1 v_1'$, and repeat, to get second eigenvalue, etc.
A Package for Matrix Powers in R

Norm Matloff and Jack Norman
University of California at Davis

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

Issues

- Due to functional programming nature of R, multiple copies of the matrix may be made.
  - This can be a problem with limited memory, e.g. on GPUs. (Thus our n = 5500 in the Markov example.)
  - Moreover, if we have no callback function, we may wish to avoid copying from `parpowm()` back to caller after each iteration.
  - Future versions may allow for user-specified temp storage space at the site of computation, e.g. a GPU or nodes in a cluster (adapting `parMM()` in old Snow).
Issues

• Due to functional programming nature of R, multiple copies of the matrix may be made.
A Package for Matrix Powers in R

Norm Matloff and Jack Norman
University of California at Davis

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

Issues

- Due to functional programming nature of R, multiple copies of the matrix may be made.
- This can be a problem with limited memory, e.g. on GPUs.
A Package for Matrix Powers in R

Norm Matloff and Jack Norman
University of California at Davis

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

Issues

- Due to functional programming nature of R, multiple copies of the matrix may be made.
- This can be a problem with limited memory, e.g. on GPUs. (Thus our $n = 5500$ in the Markov example.)
Issues

- Due to functional programming nature of R, multiple copies of the matrix may be made.
- This can be a problem with limited memory, e.g. on GPUs. (Thus our $n = 5500$ in the Markov example.)
- Moreover, if we have no callback function, we may wish to avoid copying from `parpowm()` back to caller after each iteration.
A Package for Matrix Powers in R

Norm Matloff and Jack Norman
University of California at Davis

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

Issues

- Due to functional programming nature of R, multiple copies of the matrix may be made.
- This can be a problem with limited memory, e.g. on GPUs. (Thus our \( n = 5500 \) in the Markov example.)
- Moreover, if we have no callback function, we may wish to avoid copying from `parpown()` back to caller after each iteration.
- Future versions may allow for user-specified temp storage space at the site of computation, e.g. a GPU or nodes in a cluster (adapting `parMM()` in old Snow).
Conclusions

• Matrix powers have lots of uses.
• Especially useful in parallel contexts, due to fast matrix multiplication.
• Our parmatpows package provides a convenient tool for matrix powers apps (including serial computation).
• Further work needs to be done to make this work across classes.

Location of these slides:
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 `parmatpows` 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 `parmatpows` package provides a convenient tool for matrix powers apps (including serial computation).
- Further work needs to be done to make this work across classes.
Conclusions

- Matrix powers have lots of uses.
- Especially useful in parallel contexts, due to fast matrix multiplication.
- Our parmatpows package provides a convenient tool for matrix powers apps (including serial computation).
- Further work needs to be done to make this work across classes.

Location of these slides: