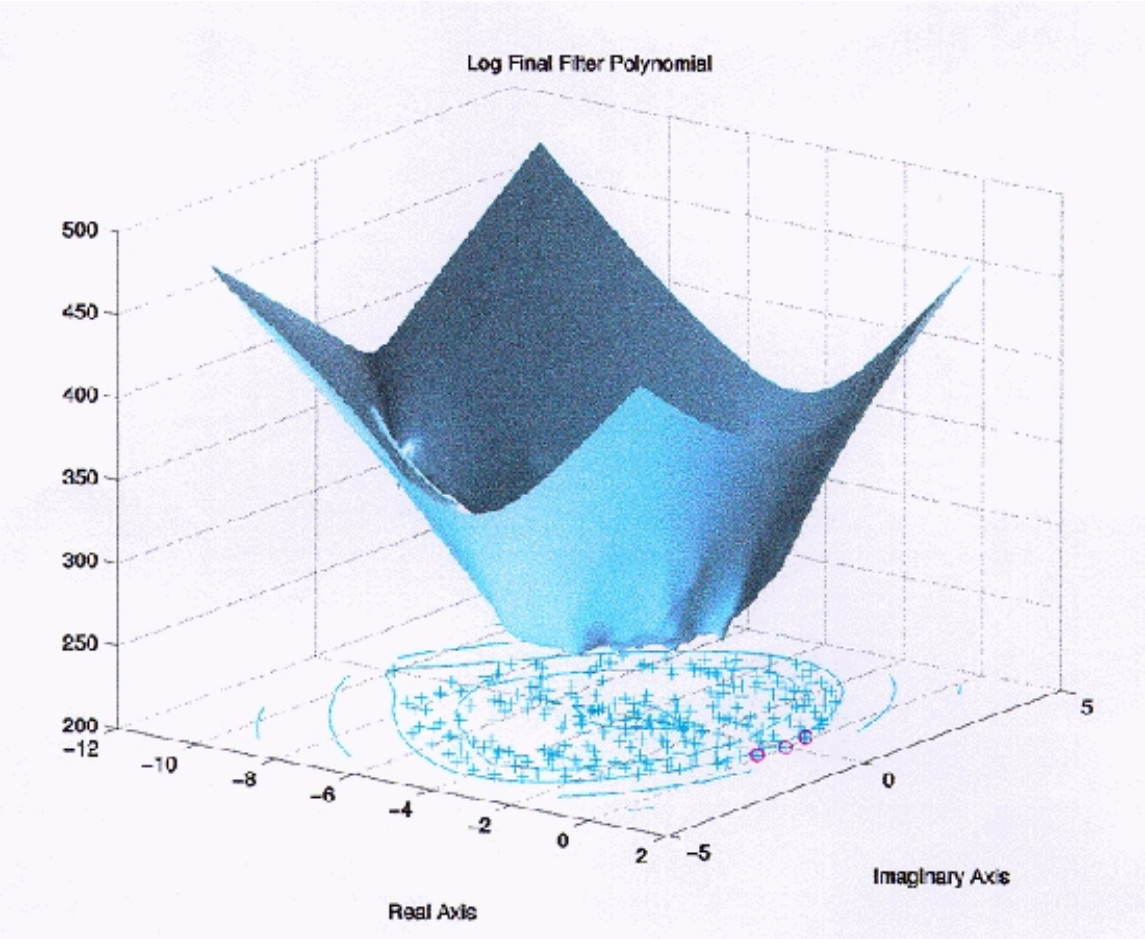
ARPACK

Dick Kachuma & Alex Prideaux

Oxford University Computing Laboratory

November 3, 2006

What is ARPACK?



- ARnoldi PACKage
- Collection of routines to solve large scale eigenvalue problems
- Developed at Rice University, Texas
  www.caam.rice.edu/software/ARPACK/

The History of ARPACK

Main authors:

- Danny Sorensen (Rice University, Texas)
- Kristi Maschhoff (Rice University, Texas)
- Rich Lehoucq (Sandia National Laboratories)
- Chao Yang (Berkeley National Laboratory, California)

Funded by:

- National Science Office
- ARPA (administered by US Army Research Office)

What can it do?

- Designed to compute a few eigenvalues and corresponding eigenvectors of a general n by n matrix A.
- Most appropriate for large sparse or structured matrices A where structured means that a matrix-vector product $w \leftarrow Av$ requires order n rather than the usual order n^2 floating point operations.
- Suitable for solving large scale symmetric, nonsymmetric, and generalized eigenproblems from significant application areas.
- Eigenvalues and vectors found can be chosen to have specified features such as those of largest real part or largest magnitude.
- Storage requirements are on the order of $n \times k$ locations. Where k is the number of eigenvalues requested.

Algorithms used

- In the general case, the Implicitly Restarted Arnoldi Method (IRAM) is used
- If A is symmetric it reduces to the Implicitly Restarted Lanczos Method (IRLM).
- For most problems, a matrix factorization is not required. Only the action of the matrix on a vector is needed, or even more generally, an operation defined.

Libraries used by ARPACK

ARPACK makes use of

- LAPACK
- BLAS
- SuperLU (depending on problem being solved)
- UMFPack (depending on problem being solved)

The required LAPACK and BLAS routines are shipped with ARPACK (although different versions can be used), SuperLU and UMFPack must be installed separately.

LAPACK routines used

Main LAPACK routines used are:

- Xlahqr (computes Schur decomposition of Hessenberg matrix)
- Xgeqr (computes QR factorization of a matrix)
- Xsteqr (diagonalizes a symmetric tri-diagonal matrix)
- Ytrevc (computes the eigenvectors of a matrix in upper (quasi-)triangular form)

BLAS routines used

Main BLAS routines used are:

- Xtrmm (matrix times upper triangular matrix - Level 3)
- Xgemv (matrix vector product - Level 2) - **IMPORTANT**
- Xger (rank one matrix update - Level 2)
- Cgeru (rank one complex matrix update - Level 2)
- Many other Level 1 routines.

Code structure - Fortran implementation

- The original code is written in Fortran77.
- Functions are complicated - using a 'Reverse Communication Interface' - a function that is called repeatedly with a number of parameters, and after each call operations are carried out according to its return value and the updated parameters used.
- Functions are then called for 'Post-Processing' to return the values required.

Code structure - Fortran function notation

Reverse Communication Functions:

- XYaupd RCF using IRAM or IRLM.

Post-processing functions:

- XYeupd Return computed eigenvalues/eigenvectors
- Xneigh Compute Ritz values and error bounds for the non-symmetric case
- Xseigt Compute Ritz values and error bounds for the symmetric case

Other (auxiliary) functions:

- XYaup2
- Xgetv0
- XYconv
- XYapps

Code structure - C++ Implementation

- Fortunately for us, a C++ version is now available, the latest version being released in 2000 (although still in beta test).
- Library is actually only an interface to the Fortran, written by D. Sorensen and F. Gomes, but it has a number of improvements.
- Library has many possible Class Templates to define your operator in order to avoid the complications of the Reverse Communication Interface (although this is still available if required).

Code structure - C++ function notation

- Either use a simple 'Matlab-like' function - AREig
- Or use various functions/templates for definition of matrix or operation (uses SuperLU (CSC format), UMFPack (CSC format), LAPACK (banded or dense))
- Use your choice of function dependent on problem (eigenvalue, SVD etc.).

The AREig function

The simple way to use ARPACK++! Many different overloaded functions designed to take different parameters eg for eigenvalues & eigenvectors required from a matrix in CSC form:

int **AREig**(FLOAT **EigValR[ ]**, FLOAT **EigValI[ ]**, FLOAT **EigVec[ ]**,
int **n**, int **nnz**, FLOAT **A[ ]**, int **irow[ ]**, int **pcol[ ]**, int **nev**,
...)

- EigValR (real part of eigenvalues)
- EigValI (imaginary part of eigenvalues)
- EigVec (eigenvectors)
- n (dimension)
- nnz, A, irow, pcol (CSC form for matrix)
- nev (number eigenvalues required)

Expected Performance - according to the documentation!

Asymptotic performance:

- For a fixed number k of requested eigenvalues and a fixed length ncv Arnoldi basis, the computational cost scales linearly with n.
- The rate of execution (in FLOPS) for the IRA iteration is asymptotic to the rate of execution of Xgemv, ie time $= \mathrm{O}(nk^2)$

Fortran vs C++ performance

- Performance is very much comparable in the real variable case.
- Performance is much worse in C++ for the complex case. Matrix/vector multiplications are of the order of 750% slower (how?!)
- In their complex test case, which was VERY sparse, performance was down 31% on the Fortran implementation.

Implementation - Matlab vs ARPACK vs ARPACK++

We implemented six test problems on Henrici:

- Problem 1 - 1D Laplacian (Real Symmetric)
- Problem 2 - 2D Harmonic Oscillator (Real Symmetric)
- Problem 3 - Convection-Diffusion Operator (Real Non-symmetric)
- Problem 4 - Squire Equation (Complex)
- Problem 5 - Toy Problem (Real Symmetric)
- Problem 6 - Grcar Matrix (Real Non-symmetric, highly non-normal)

Problem 1 - 1D Laplacian

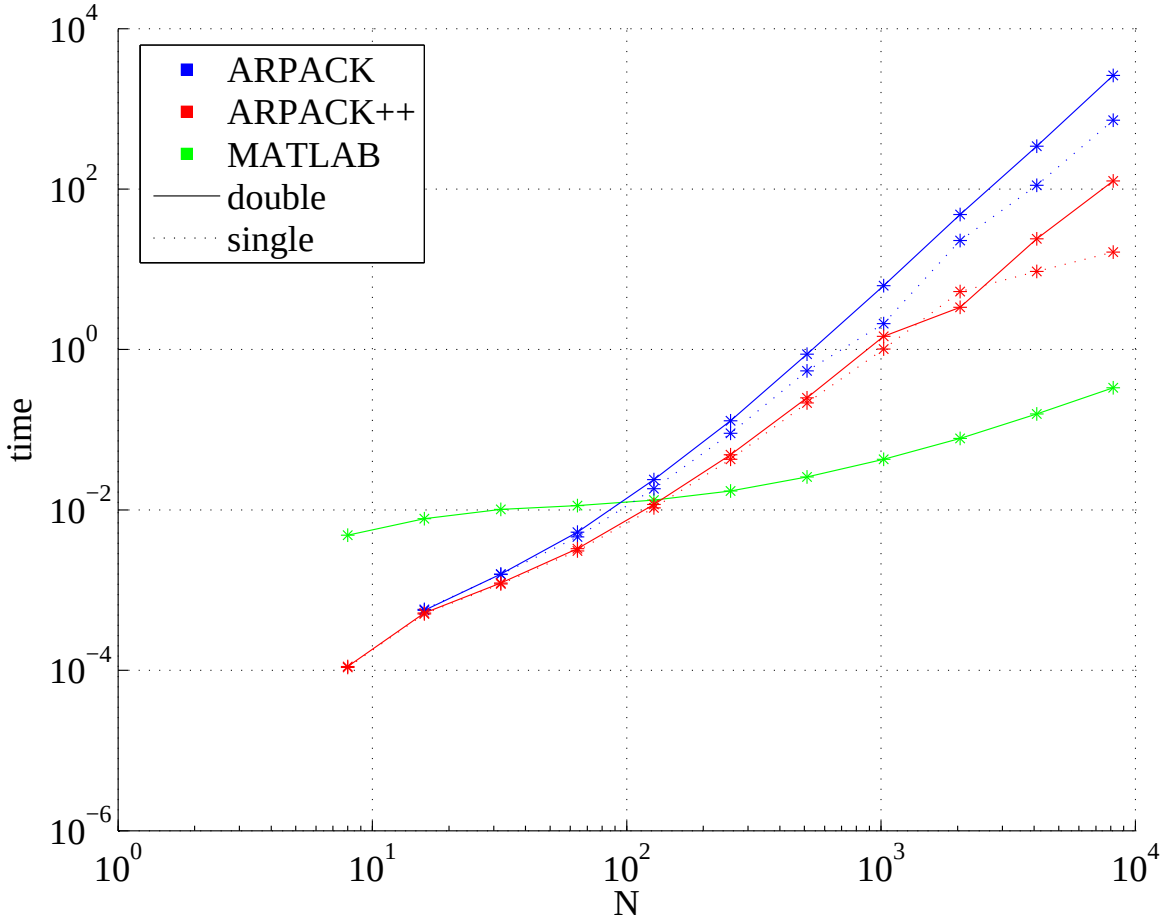
Eigenvalues of the 1D Laplacian operator:

$$-\frac{d^2u}{dx^2} = \lambda u, \qquad x \in [0,1] \tag{1}$$

Discretized using central finite differences.

Benchmarking was carried out by measuring time taken to calculate the smallest 4 eigenvalues using implementations in ARPACK and ARPACK++, in comparison with the Matlab 'eigs'.

Problem 1 - Results



Problem 2 - 2D Harmonic Oscillator

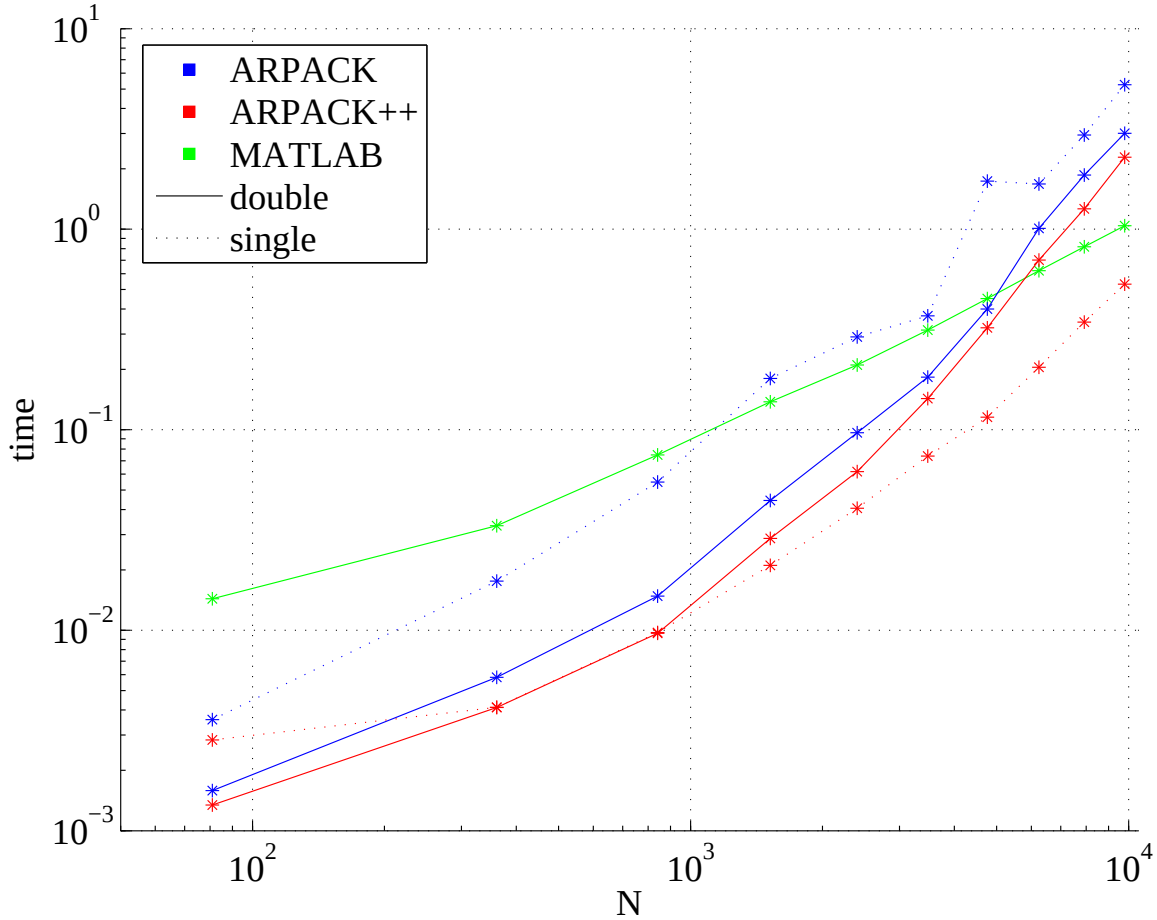
Eigenvalues of the harmonic oscillator:

$$-\frac{d^2u}{dx^2} - \frac{d^2u}{dy^2} + (x^2 + y^2)u = \lambda u, \qquad (x, y) \in [-5, 5] \times [-5, 5] \quad (2)$$

Discretized using central finite differences.

Benchmarking was carried out by measuring time taken to calculate the smallest 4 eigenvalues using implementations in ARPACK and ARPACK++, in comparison with the Matlab 'eigs'.

Problem 2 - Results



Problem 3 - Convection-Diffusion Operator

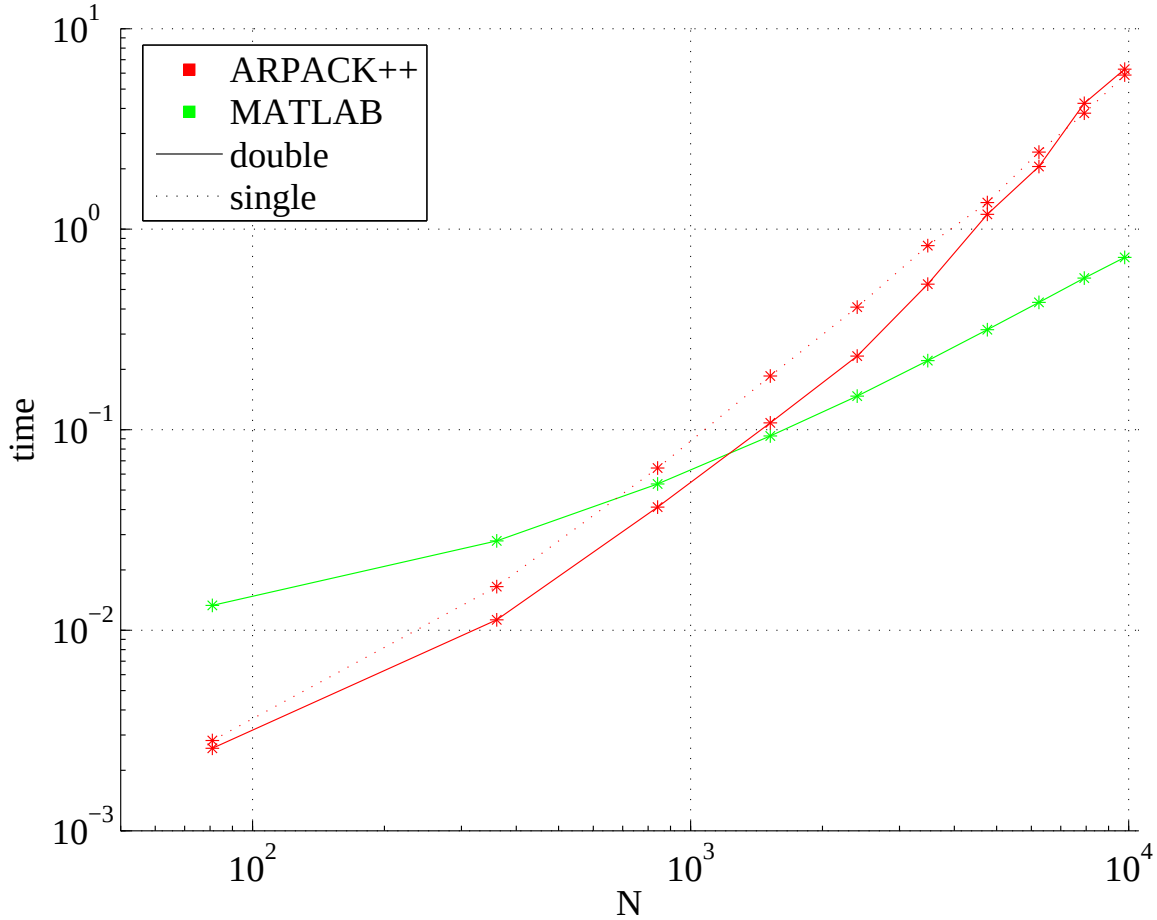
Eigenvalues of the convection-diffusion operator:

$$-\Delta u + 2\mathbf{a}.\nabla u = \lambda u, \qquad (x, y) \in [0, 1] \times [0, 1], \quad \mathbf{a} = (-1, 2)^T \tag{3}$$

Discretized using central finite differences.

Benchmarking was carried out by measuring time taken to calculate the smallest 4 eigenvalues using an implementation in ARPACK++ in comparison with the Matlab 'eigs'.

Problem 3 - Results



Problem 4 - Squire Equation

Eigenvalues of the Squire equation:

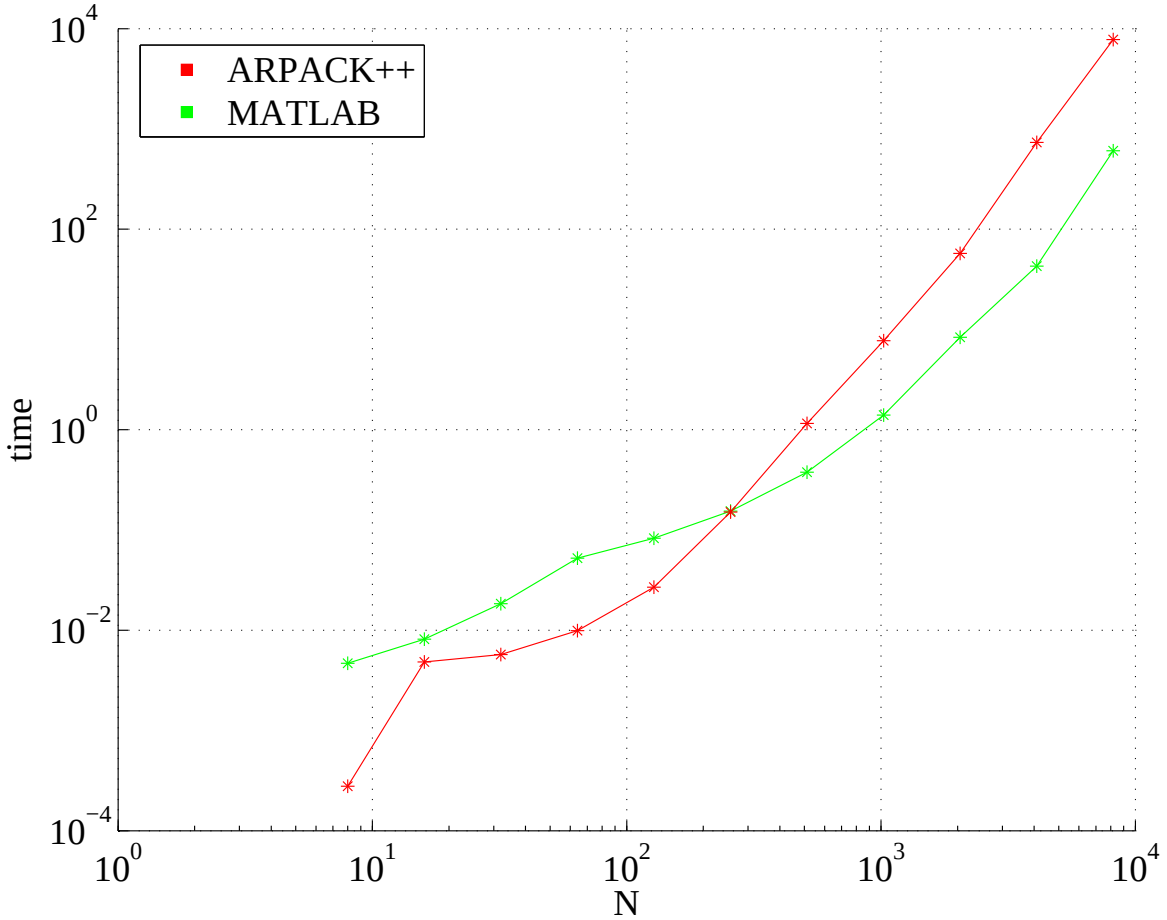
$$\left(U(y) - \frac{1}{i\ k\ Re}\left(\frac{d^2}{dy^2} - k^2\right)\right)\omega = c\omega \tag{4}$$

where $U(y) = 1 - y^2$, $k = 1.0$, $Re = 2000$.

The equation was again discretized using central finite differences. Benchmarking was carried out by measuring time taken to calculate the 4

eigenvalues of largest imaginary part using an implementation in ARPACK++ in comparison with the Matlab 'eigs'.

Problem 4 - Results



Problem 5 - Toy Problem

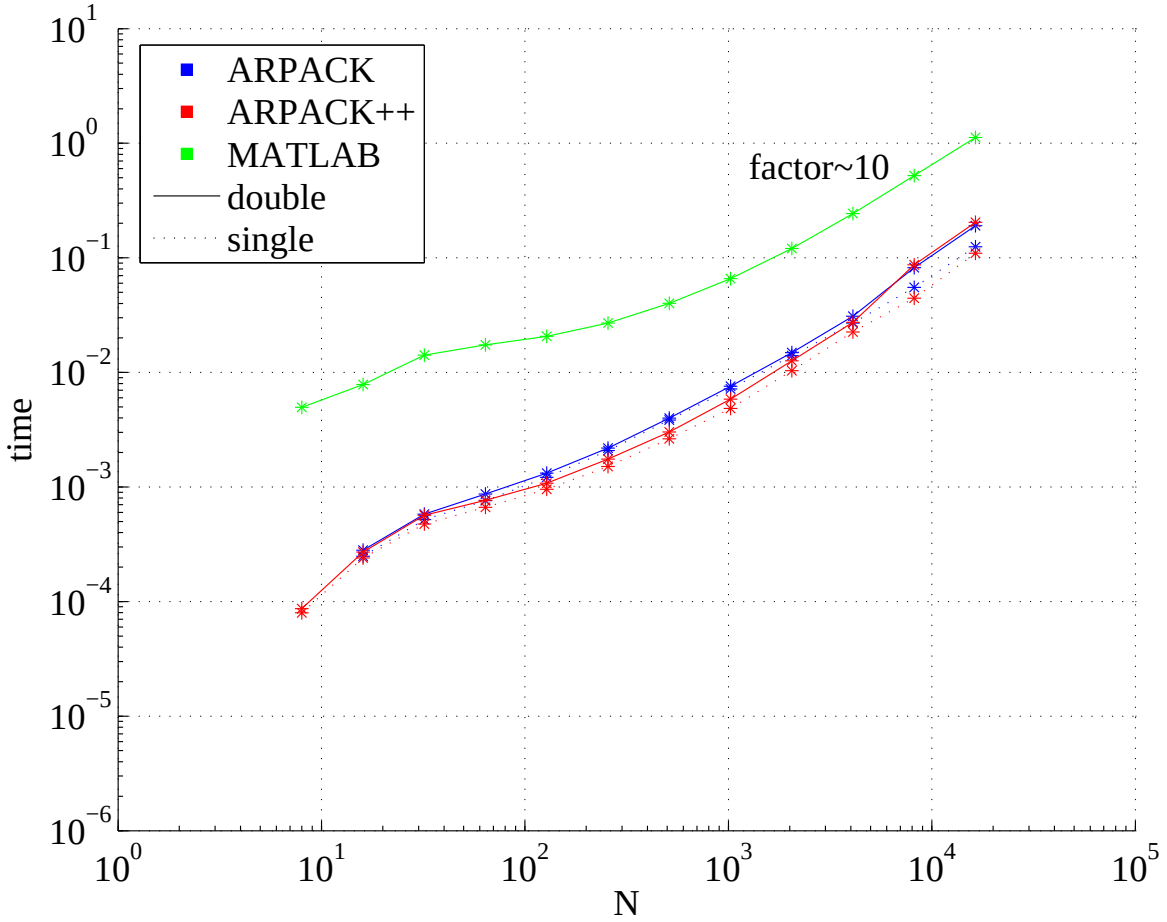
We now look at the eigenvalues of a 'toy' problem, to try and obtain $\mathrm{O}(N)$ time dependence for both Matlab and ARPACK.

$$Ax = \lambda x \tag{5}$$

where A is tri-diagonal with 1.002 on the diagonal and -0.001 on the off diagonals.

Benchmarking was carried out by measuring time taken to calculate the smallest 4 eigenvalues using an implementation in both ARPACK and ARPACK++ in comparison with the Matlab 'eigs'.

Problem 5 - Results



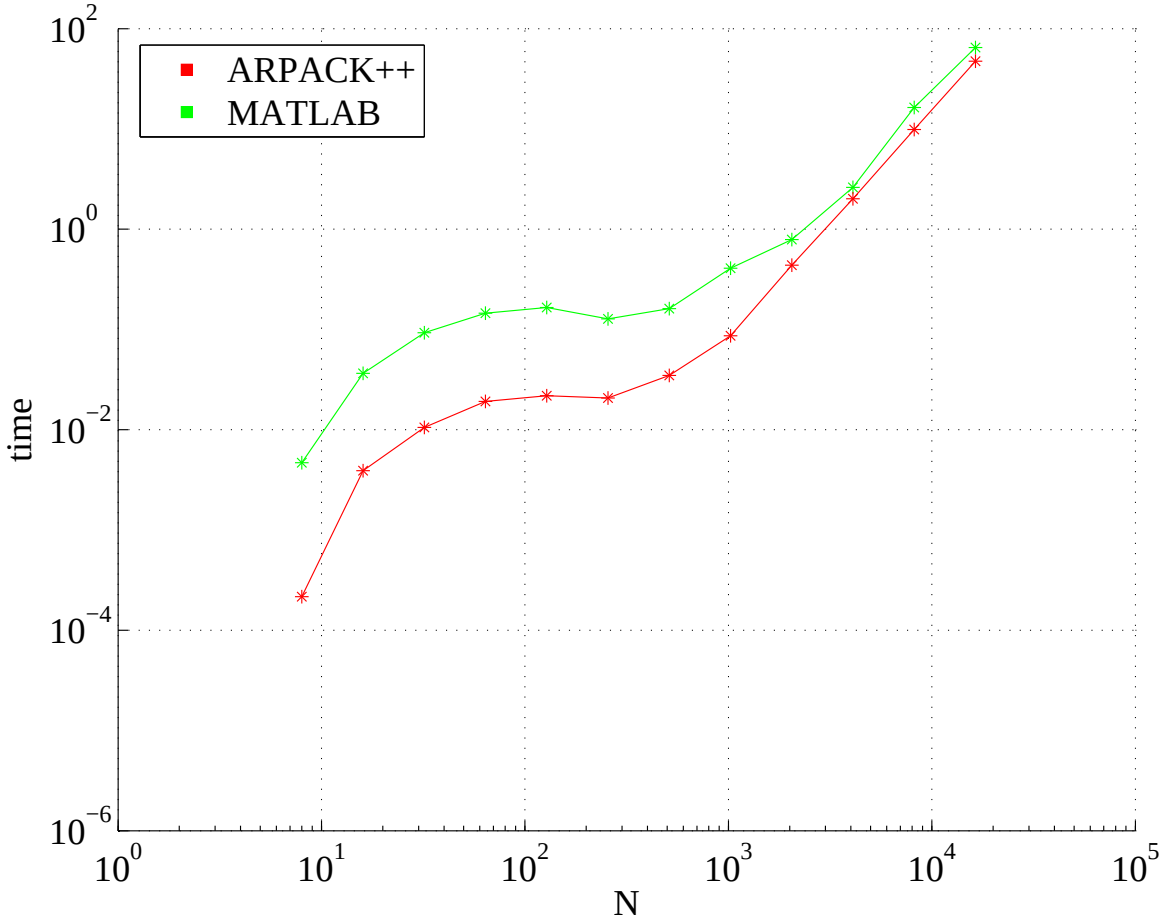
Problem 6 - Grcar Matrix

We finally look at the eigenvalues of the Grcar matrix. The use of this matrix and its non-normality in particular was designed to force Matlab to use stricter error bounds and therefore make it comparable to ARPACK.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & & & & \\ -1 & 1 & 1 & 1 & 1 & & & \\ & -1 & 1 & 1 & 1 & 1 & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & -1 & 1 & 1 & 1 & 1 \\ & & & & -1 & 1 & 1 & 1 \\ & & & & & -1 & 1 & 1 \\ & & & & & & -1 & 1 \end{pmatrix} \tag{6}$$

Benchmarking was carried out by measuring time taken to calculate the 4 eigenvalues of smallest magnitude using an implementation in both ARPACK and ARPACK++ in comparison with the Matlab 'eigs'.

Problem 6 - Results



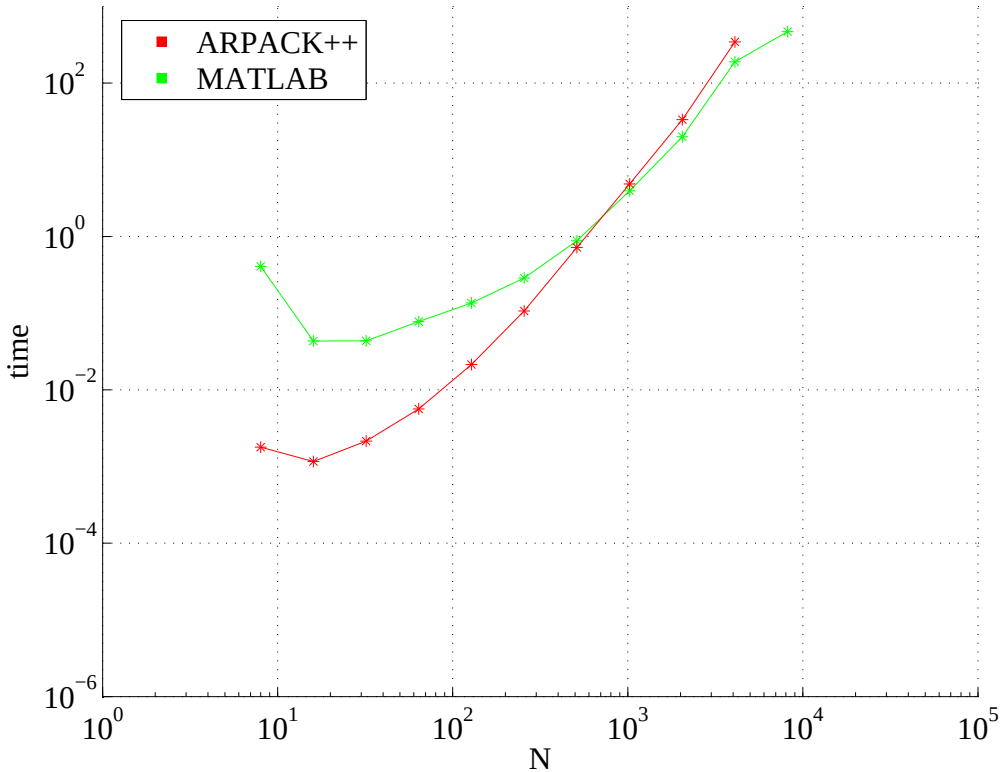
Find the 2-norm condition number of the matrix Toeplitz matrix:

$$\begin{pmatrix} 3 & 4 & & & & 1 & 0 \\ 0 & 3 & 4 & & & & 1 \\ 1 & 0 & 3 & 4 & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & 0 & 3 & 4 & \\ & & & 1 & 0 & 3 & 4 \\ 4 & & & & 1 & 0 & 3 \end{pmatrix} \tag{7}$$

with dimension $n = 2^{20}$.

Problem Solving Squad - BRUTE FORCE

As a demonstration of Matlab's 'svds' and the functions in ARPACK, we show the time taken to calculate the norm simply by generating the sparse matrix and finding the required singular values:



(note the sensible method is to extrapolate the results for small n to $n = 2^{20}$).

Conclusions

- Our results show that Matlab is clearly the package of choice, offering ease of use and good speed.
- We would expect ARPACK to be comparable as this underlies Matlab's 'eigs', but we can only demonstrate this in the highly non-normal case.
- From an implementation point of view, ARPACK++ is vastly preferable to ARPACK, but it should be used only cautiously in the complex case!