

Simple Benchmarks for Matlab and Clones

Derek O'Connor
derekroconnor@eircom.net
www.dereroconnor.net

August 19, 2006

1 INTRODUCTION

The tables below are the results of a set of very simple benchmarks performed on MATLAB, O-MATRIX, OCTAVE, and SCILAB.

These tests are not meant to be definitive, exhaustive, etc. They do nothing more than test the standard matrix functions on a single 1000×1000 matrix A generated by the statement $A = \text{rand}(1000, 1000)$. The MATLAB code is shown in Section 3.

Each of these numerical software systems is based on the matrix data type. If such a system does not perform well on the standard matrix functions then something is wrong.

The first column in each table uses MATLAB 6.5 with the reference BLAS kernel. This kernel is not optimized for any processor. It is included to show the huge effect an optimized kernel has on the speed of these operations.

Table 1: Benchmarks on 1000×1000 d.p. Random Matrix A (secs)

Software Version	Matlab 6.5	Matlab 6.5	Matlab 7.1	O-Matrix 6	Octave 2.1.73	Scilab 4
BLAS Kernel Version	Ref	Atlas ?	MKL ?	MKL 8.0.1.0	Atlas ?	Atlas ?
Cost (approx)	€ 900	€ 900	€ 900	\$145-365	Free	Free
$A * A$	38.8	3.9	3.5	3.3	30.6	4.0
LU(A)	11.0	1.8	1.6	1.4	9.2	3.0
Inv(A)	36.5	5.0	4.3	3.7	29.2	5.2
SVD(A)	155.2	106.2	101.6	58.4	32.3	106.3
QR(A)	30.6	11.9	9.0	13.2	12.5	21.4
Eig(A)	180.1	113.8	104.0	89.8	74.2	111.1
Det(A)	11.0	1.7	1.6	1.3	9.4	1.9
Rank(A)	36.3	16.3	12.9	21.2	32.5	16.4
Cond(A)	36.3	16.3	12.9	21.3	32.9	16.4
Total	536.0	277.0	252.5	213.6	252.5	286.9

Dell Precision 620, Pentium III Xeon, 800MHz, 640MB RAM, Windows 2000 Prof. SP4

Table 2: Normalized Benchmarks on 1000×1000 d.p. Random Matrix A

Software Version	Matlab 6.5	Matlab 6.5	Matlab 7.1	O-Matrix 6	Octave 2.1.73	Scilab 4
BLAS Kernel Version	Ref	Atlas ?	MKL ?	MKL 8.0.1.0	Atlas ?	Atlas ?
Cost (approx)	€ 900	€ 900	€ 900	\$145-365	Free	Free
$A * A$	11.8	1.2	1.1	1	9.3	1.2
LU(A)	7.9	1.3	1.1	1	6.6	2.1
Inv(A)	9.9	1.4	1.2	1	7.9	1.4
SVD(A)	4.8	3.3	3.1	1.8	1	3.3
QR(A)	3.4	1.3	1	1.5	1.4	2.4
Eigen(A)	2.4	1.5	1.4	1.2	1	1.5
Det(A)	8.5	1.3	1.2	1	7.2	1.5
Rank(A)	2.3	1.3	1	1.6	2.5	1.3
Cond(A)	2.3	1.3	1	1.7	2.6	1.3
Total	2.5	1.3	1.2	1	1.2	1.3

Dell Precision 620, Pentium III Xeon, 800MHz, 640MB RAM, Windows 2000 Prof. SP4

2 NOTES

2.1 Anomalies and Questions

1. Can anyone explain why OCTAVE is the slowest on the simplest problem, matrix multiplication (30 secs), yet the fastest on the two hardest problems, SVD (32 secs), and EIG (74 secs)?
2. Can anyone explain why SCILAB gives this result?
 - $S = \text{svd}(A)$ 16 secs
 - $[U, S, V] = \text{svd}(A)$ 106 secs

2.2 Answers

- 1.
2. This is because the singular values are computed first and in a second time (if needed) the matrices U and V (this second step being more time consuming).

Nevertheless scilab don't use the "new" lapack routine `dgesdd` but the old one `dgesvd` and the algorithm behind `dgesdd` is much faster for computing the left and right singular vectors.

Experimenting with `dgesvd` and `dgesdd` on a 1000×1000 matrix give me :

```
dgesvd : 59 s
dgesdd : 14.5 s
```

hth Bruno

I am not sure who Bruno is but his partial email address is `bruno.pin...@iecn.u-nancy.fr` so I suppose he is one of the people working on SCILAB.

2.3 Rounding problems with Intel Math Kernel library

The following is from the FAQ of IntLab 5.3, Siegfried Rump's Interval Lab, a MATLAB toolkit that does interval arithmetic, among other things.

Recent versions of Matlab choose to use IMKL as default for BLAS operations. The IMKL library takes full control over the control word forcing internal computation to be done in extended and setting the rounding mode to nearest. After return to Matlab the control word is reset to its previous value.

This has peculiar consequences. For example, the results `res1` and `res2` in

```
x = randn(3,1), res1 = x'*x, res2 = x(1)*x(1)+x(2)*x(2)+x(3)*x(3)
```

should mathematically be equal. However, `res1` is a dot product and computed by IMKL, so the intermediate sums are accumulated in extended precision, whereas the second result `res2` is computed by conventional multiplication and addition in Matlab, so intermediate sums are accumulated in double precision. Note this peculiarity occurs in ordinary Matlab and has nothing to do with INTLAB.

Even worse, IMKL switches the control word to "round to nearest", independent of its current setting. Therefore, INTLAB cannot work with IMKL. For example, define `x=intval([1 1e-30 1])`, then the results of

```
x*ones(3,1) and x(1)+x(2)+x(3)
```

will be different because the first uses (implicitly) IMKL, the latter does not.

Since Version 5.1 of INTLAB we test for IMKL and give a corresponding message. If you use an older version of INTLAB, you can easily test yourself by

```
x = 0.1*ones(2,1); d = diam(x'*intval(x))
```

The diameter `d` should not be zero due to rounding errors in the computation of `diam(x'*intval(x))`. If `d==0`, then you have a newer version of Matlab and use the IMKL. In order to make INTLAB work properly, you have to change to another BLAS library, for instance the Atlas library which was used up to now by Matlab. Unfortunately, I cannot do that from Matlab but the user has to do that outside Matlab as a system configuration.

For that you have to change the environment variable `BLAS_VERSION`. From Matlab release notes:

```
"If you want to take advantage of the potential performance enhancements provided by the Intel BLAS, you can set the value of the environment variable BLAS_VERSION to the name of the MKL library, mkl.dll. MATLAB uses the BLAS specified by this environment variable, if it exists. To set the BLAS_VERSION environment variable, follow this procedure: Click the Start button, go to the Settings menu, and select Control Panel. On the Control Panel menu, select System. In the System Properties dialog box, click the Advanced tab. On the Advanced panel, click the Environment Variables button. In the Environment Variables dialog box, click the New button in the User variables section. In the New User Variable dialog box, enter the name of the variable as BLAS_VERSION and set the value of the variable to the name of the MKL library: mkl.dll. "
```

Please consult http://www.mathworks.com/access/helpdesk/help/techdoc/rn/r14sp1_math_new.html

3 THE MATLAB BENCHMARK

All other benchmarks are translations of this code.

```

function n = MatBench;
disp( '=====')
disp( ' Written by Derek O'Connor, Dec 2004, Aug 2006')
disp( ' derekroconnor@eircom.net')
clear all;
n = input('Enter Matrix size: ');
A = rand(n,n);
B = A*A; % Warmup : load dlls ??
totalT = 0.0;
%----- Start Tests -----
tic; B = A*A; t1 = toc;
totalT = totalT + t1;
disp(['Multiply      ' num2str(t1) ' secs' ])
%
tic; [L,U,P] = lu(A); t2 = toc;
totalT = totalT + t2;
disp(['LU Decomp    ' num2str(t2) ' secs' ])
%
tic; B = inv(A); t3 = toc;
totalT = totalT + t3;
disp(['Invert A     ' num2str(t3) ' secs' ])
%
tic; [U,S,V] = svd(A); t4 = toc;
totalT = totalT + t4;
disp(['SDV A       ' num2str(t4) ' secs '])
%
tic; [Q,R,P] = qr(A); t5 = toc;
totalT = totalT + t5;
disp(['QR Decomp   ' num2str(t5) ' secs '])
%
tic; [v,d] = eig(A); t6 = toc;
totalT = totalT + t6;
disp(['Eigen A     ' num2str(t6) ' secs '])
%
tic; d = det(A); t7 = toc;
totalT = totalT + t7;
disp(['DET A      ' num2str(t7) ' secs '])
%
tic; r = rank(A); t8 = toc;
totalT = totalT + t8;
disp(['Rank A     ' num2str(t8) ' secs '])
%
tic; c = cond(A);
t9 = toc; totalT = totalT + t9;
disp(['Cond A     ' num2str(t9) ' secs '])
%
disp(['Total Time ' num2str(totalT) ' secs']);
%
disp('=====Finished SimpleBench =====')

```