

# Numerical Analysis – Lab 17

## Efficiency of MatLab

### Goals

The goals of this lab are:

1. to determine if MatLab is more efficient at manipulating certain special kinds of matrices than it is at manipulating general ones.
2. to have students design and perform an experiment to answer this question.

### Preliminaries

MatLab keeps track of how hard it works by keeping a statistic called “flops.” That stands for “floating point operations.” The following script shows how we can use the statistic to find how hard it is for MatLab to invert a random 4x4 matrix:

```
EDU>> b=rand(4,4)
b =
    0.2190    0.9347    0.0346    0.0077
    0.0470    0.3835    0.0535    0.3834
    0.6789    0.5194    0.5297    0.0668
    0.6793    0.8310    0.6711    0.4175
EDU>> flops
ans =
    171
EDU>> inv(b)
ans =
   -0.9618    6.3842    8.5617   -7.2163
    1.3011   -1.2531   -1.7389    1.4052
    0.1081   -7.4732   -7.6003    8.0782
   -1.1984    4.1202    1.7486   -1.6465
EDU>> flops
ans =
    385
```

Since MatLab had used 171 flops before inverting the matrix and 385 after the inversion, we conclude that the inversion took 214 flops.

Similarly, multiplying  $2*b$  took 16 flops, and calculating  $b^2$  took 128 flops.

## Questions

### Problem 1: Flops for addition, subtraction and scalar multiplication

For various sized random matrices, find how many flops it takes to add, subtract and scalar multiply those matrices. Try to find a formula in terms of the dimensions of the matrices involved.

### Problem 2: Flops for multiplication and inverses

For various sized random square matrices, find how many flops it takes to multiply and invert those matrices. Again, try to find formulas.

Compare the flops you observe to the number of operations it takes to multiply and invert matrices using the methods you learned (or are learning) in Linear Algebra.

### Problem 3: Efficiency for eigenvalues for upper triangular matrices

The dialog below shows that it takes MatLab 1395 flops to find the eigenvalues of  $b$ . We know, though, that if  $U$  is an upper triangular matrices, then its eigenvalues are just its diagonal elements, and they don't require any flops at all to find. Determine how many flops it takes to find the eigenvalues of a random  $n \times n$  matrix in terms of the dimensions of the matrix. Determine if MatLab is "smart" and is more efficient at finding eigenvalues for special matrices like upper triangular ones.

```
EDU» flops
ans =
    2566
EDU» eig(b)
ans =
    1.4095
    0.1082 + 0.4681i
    0.1082 - 0.4681i
   -0.0763
EDU» flops
ans =
    3961
```

### Problem 4: Matrix powers

We found above that squaring a random  $4 \times 4$  matrix took 128 flops. You may have found that cubing it takes 256 flops, the fourth power takes  $(4-1) \cdot 128 = 384$  flops, and concluded that, in general, an  $n$ th power takes  $(n-1) \cdot 128$  flops. So finding  $b^{101}$  would seem to take 12,800 flops.

There is a faster way that uses the binary expansion of 101. It works as follows:  $101 = 64 + 32 + 4 + 1 = 2^6 + 2^5 + 2^2 + 2^0$ , so in binary, one hundred and one is written 1100101. First,  $b^{101} = b^{64} \cdot b^{32} \cdot b^4 \cdot b^1$ . But  $b^{64}$  is easy to find with just six squarings, as

follows:  $b^{64} = \left( \left( \left( \left( (b^2)^2 \right)^2 \right)^2 \right)^2 \right)^2$ . Likewise when the exponent is 2 or 4 or 32 or any

other power of two. So, to find  $b^{100}$ , first do six squarings to find  $b^n$  for  $n = 1, 2, 4, 8, 16, 32$  and  $64$ . Then do three more multiplications to find  $b^{101} = b^{64} \cdot b^{32} \cdot b^4 \cdot b^1$ . You only have to do seven multiplications instead of 100.

Does MatLab know this trick?

### Problem 5: Special matrices

We have learned about a number of kinds of special matrices that are supposed to be “easy” to invert or to multiply; diagonal, diagonally dominant, upper (or lower) triangular, orthogonal. Determine whether such matrices are “easier” for MatLab.

### Write it all up.

I expect good tables, clear and concisely stated results, and a neat, well-written and well-organized presentation. Due April 27.

### Appendix: Making a random upper triangular matrix in MatLab, and other hints

MatLab has “element by element” options for its operations. To make an operator “element by element,” you put a dot before the operator, as “ $\cdot$ ” for multiplication or “ $\wedge$ ” for exponentiation. Of course, addition and subtraction are already element by element.

If  $U$  is the  $3 \times 3$  upper triangular matrix of all 1's, and if  $B$  is a random  $3 \times 3$  upper triangular matrix, then  $U \cdot B$  is a random  $3 \times 3$  upper triangular matrix, as follows:

```
EDU>> U=[1,1,1;0,1,1;0,0,1]
U =
    1    1    1
    0    1    1
    0    0    1
EDU>> B=rand(3,3)
B =
    0.6868    0.8462    0.6539
    0.5890    0.5269    0.4160
    0.9304    0.0920    0.7012
EDU>> U.*B
ans =
    0.6868    0.8462    0.6539
     0        0.5269    0.4160
     0         0        0.7012
```

Another trick is that if  $B=\text{rand}(4,4)$  is a random matrix and  $I=\text{eye}(4)$  is the  $4 \times 4$  identity matrix, then  $B+4 \cdot I$  will be diagonally dominant. This follows because all the entries of  $B$  are between 0 and 1.

If you discover other clever tricks, put them in an appendix to your report called “clever tricks.”