HOMEWORK
David F. Gleich
October 15, 2018

Homework 6

Please answer the following questions in complete sentences in a clearly prepared manuscript and submit the solution by the due date on Blackboard (around Sunday, October 21th, 2018.)

Remember that this is a graduate class. There may be elements of the problem statements that require you to fill in appropriate assumptions. You are also responsible for determining what evidence to include. An answer alone is rarely sufficient, but neither is an overly verbose description required. Use your judgement to focus your discussion on the most interesting pieces. The answer to "should I include 'something' in my solution?" will almost always be: Yes, if you think it helps support your answer.

Problem 0: Homework checklist

- Please identify anyone, whether or not they are in the class, with whom you discussed your homework. This problem is worth 1 point, but on a multiplicative scale.
- Make sure you have included your source-code and prepared your solution according to the most recent Piazza note on homework submissions.

Problem 1: Estimating real-world performance for elimination

In class, we showed how to count the number of floating point operations involved in computing an LU decomposition. Here, we are going to seek a few ways to make this count more realistic. More generally, this type of study would be called performance analysis. But in this case, we'll just make a few simple observations.

1. Repeat the analysis of the LU for class, but account for the number of floating point operations in light of a single "fused-multiply-add" operation that is now available on various Intel CPUs. https://en.wikipedia.org/wiki/Multiply%E2%80%93accumulate_operation (assume that you also have a fused-multiply subtract if it's easier)
2. Having the count of FLOPs is only relevant if all floating point operations have the same cost. Develop an estimate for the relative cost of the following floating point operations: `+`, `-`, `*`, `/`, `sqrt`, `sin`, `abs` My code to do this involves generating random numbers between [-1,1], and then performing the operation on the random numbers. Then we subtract off the time just to generate the random numbers. The key is not to store anything in intermediate variables otherwise, you'll benchmark the memory system. Now, assume that the cheapest flop costs 1, what is the number of effective flops in terms of real-world cost of these operations?

Problem 2: Using accelerators

A recent innovation are the presence of large "tensor-cores" that support "matrix multiplication" directly on GPUs or other CPU accelerators. These can make matrix-mulplication operations much much faster than expected. We are going to use them to solve a linear system via block-substitution.

1. Write down the steps of an algorithm to solve $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ via recursive block-elimination. That is, divide $\boldsymbol{A}$ into blocks:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \\ \boldsymbol{A}_3 & \boldsymbol{A}_4 \end{bmatrix}$$

   where all blocks have the same size. In this case, assume that $\boldsymbol{A}$ has a power-of-two dimension so that you can do this recursively as well until you get down to a 1×1 system. Then if

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$$

   are the vectors that conform to the partitioning of $\boldsymbol{A}$, develop an algorithm that uses as a subroutine:

```
# return the matrix after solving k linear systems $A^{-1} B$
function solve(A::Matrix, B::Matrix)
end
```

   and matrix-matrix multiplication to solve the linear system.

2. Count the number of floating point operations involved in the algorithm from step 1 in terms of flops that occur in matrix-multiplication and those that occur in other steps. This will require you to solve for the recurrences involved in the recursive algorithm.

Problem 3: Implement a face search

1. Load the Yale face database of 165 pictures from the data on our website:

   - http://www.cs.purdue.edu/homes/dgleich/cs515-2018/homeworks/Yale_64.csv
   - http://www.cs.purdue.edu/homes/dgleich/cs515-2018/homeworks/Yale_64_ids.csv

   This can be read into Julia via

```
A = readdlm("Yale_64.csv",',')
labels = readdlm("Yale_64_ids.csv")
```

   More about the data is here: http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html This was converted from the Matlab file into the CSV files.

   Prepare and plot a heatmap/picture of image 67.

2. We are going to use all of the even numbered images as a database $\boldsymbol{D}$. How accureately can we represent the odd-numbered images in terms of the even numbered images in a least squares sense? (That is, suppose we consider representing an image $\boldsymbol{T}$ What is the image with the worst reconstruction? What is the image with the best reconstruction?

3. How well does least-squares image search tolerate errors? Suppose we preturb an image with random normal noise as follows:

$$\mathbf{v}' = \mathbf{v} + \sigma N(0,1)$$

where $N(0,1)$ is a vector a random normal variables and σ is the scale parameter. (This is equivalent to using a normal with σ^2 as the variance.) Suppose that we use all images as the database $\boldsymbol{D}$, and we fix an image, say, number 67. How large does σ need to be before we won't recognize image 67 as image 67 anymore? How large does σ need to be before we won't recognize this as person 7 anymore? Suppose that we define recognition failing as when at least 25/100 random trials identify another person. (Hint: I strongly recommend using a QR factorization of the image database to make this experiment solvable in a small amount of time, feel free to use Julia's built in QR factorization.)

Problem 4: Givens vs. Householder

(Based on class discussion.) In this question, you'll study some of the questions that came up in class. Given a vector $\mathbf{a}$, we can use a single Householder transform to produce an orthogonal matrix $\boldsymbol{H}$ such that $\boldsymbol{H}\mathbf{a} = \pm\|\mathbf{a}\|\mathbf{e}_1$. We can also use a set of $n-1$ givens rotations:

$$\boldsymbol{G}_{n-1}\boldsymbol{G}_{n-2} \cdot \boldsymbol{G}_1\mathbf{a} = \pm\|\mathbf{a}\|\mathbf{e}_1.$$

1. Compare (in theory) the orthogonal matrices that arise from these two ways of looking at the problem for a length 2 vector.
2. Compare (in theory) the orthogonal matrices that arise from these two ways of looking at the problem for a length 3 vector.
3. State a conjecture or conclusion from the results of part 1 and 2.
4. Note that the matrix $\boldsymbol{H}$ only depends on knowing the vector that determines the Householder reflector. Also, note that the matrices $\boldsymbol{G}_i$ only depend on knowing $\cos\theta_i$ and $\sin\theta_i$. Analyze the number of floating point operations required to compute $\mathbf{v}$ in Householder compared with the cosines and sines in the Givens rotations.