

18.335 Problem Set 1

Due Friday, 14 September 2012.

Problem 1: Gaussian elimination

Trefethen, problem 20.4.

Problem 2: Asymptotic notation

This problem asks a few simple questions to make sure that you understand the asymptotic notations O , Ω , and Θ as defined in the handout in class, and also to make sure you are comfortable with simple proofs. (A detailed review of asymptotic notation can be found in any computer-science textbook, or on many sites online.)

- If $f(n)$ is $\Theta[F(n)]$ and $g(n)$ is $\Theta[G(n)]$ for nonnegative functions f , g , F , and G , prove that $f(n) + g(n)$ is $\Theta[F(n) + G(n)]$.
- Prove that $f(n)$ is $O[g(n)]$ if and only if $g(n)$ is $\Omega[f(n)]$. [For example, n^2 is $O(n^3)$ and n^3 is $\Omega(n^2)$.]
- If $f(n)$ is $O[F(n)]$, prove that any function that is $O[f(n) + cF(n)]$ must also be $O[F(n)]$ for any constant $c \neq 0$ —that is, if we regard $O[\cdot]$ as a set of functions, prove $O[f(n) + cF(n)] \subseteq O[F(n)]$. [For example, $O(n^2 + 3n^3) = O(n^3)$.] Is it also true that $\Theta[f(n) + cF(n)] \subseteq \Theta[F(n)]$ for any $c \neq 0$ if $f(n)$ is $O[F(n)]$? Explain.
- Explain why the statement, “The running time of this algorithm is $O(n^2)$ or worse,” cannot provide any information about the algorithm.

Problem 3: Caches and matrix multiplications

In class, we considered the performance and cache complexity of matrix multiplication $A = BC$, especially for square $m \times m$ matrices, and showed how to reduce the number of cache misses using various forms of blocking. In this problem, you will be comparing optimized matrix-matrix products to optimized matrix-vector products, using Matlab.

- The code `matmul_bycolumn.m` posted on the 18.335 web page computes $A = BC$ by multiplying B by each column of C individually (using Matlab’s highly-optimized BLAS matrix-vector product). Benchmark this: plot the flop rate for square $m \times m$ matrices as a function of m , and also benchmark Matlab’s built-in matrix-matrix product and plot it too. For example, Matlab code to benchmark Matlab’s $m \times m$ products for $m = 1, \dots, 1000$, storing the flop rate ($2m^3/\text{nanoseconds}$) in an array `gflops(m)`, is:

```
gflops = zeros(1,1000);
for m = 1:1000
    A = rand(m,m);
    B = rand(m,m);
    t = 0;
    iters = 1;
    % run benchmark for at least 0.1 seconds
    while (t < 0.1)
        tic
        for iter = 1:iters
```

```

        C = A * B;
    end
    t = toc; % elapsed time in seconds
    iters = iters * 2;
end
gflops(m) = 2*m^3 * 1e-9 / (t * 2/iters);
disp(sprintf('gflops for m=%d = %g after %d iters',m,gflops(m),iters/2));
drawnow update;
end
plot([1:1000], gflops, 'r-')

```

- (b) Compute the cache complexity (the asymptotic number of cache misses in the ideal-cache model, as in class) of an $m \times m$ matrix-vector product implemented the “obvious” way (a sequence of row-column dot products).
- (c) Propose an algorithm for matrix-vector products that obtains a better asymptotic cache complexity (or at least a better constant coefficient, e.g. going from $\sim 3m^2$ to $\sim 2m^2$, even if it is still the same $\Theta[\cdot \cdot]$ complexity) by dividing the operation into some kind of blocks.
- (d) Assuming Matlab uses something like your “improved” algorithm from part (c) to do matrix-vector products, compute the cache complexity of `matmul_bycolumn`. Compare this to the cache complexity of the blocked matrix-matrix multiply from class. Does this help to explain your results from part (a)?

Problem 4: Caches and backsubstitution

In this problem, you will consider the impact of caches (again in the ideal-cache model from class) on the problem of *backsubstitution*: solving $Rx = b$ for x , where R is an $m \times m$ upper-triangular matrix (such as might be obtained by Gaussian elimination). The simple algorithm you probably learned in previous linear-algebra classes (and reviewed in the book, lecture 17) is (processing the rows from bottom to top):

$$\begin{aligned}
 &x_m = b_m / r_{mm} \\
 &\text{for } j = m - 1 \text{ down to } 1 \\
 &\quad x_j = (b_j - \sum_{k=j+1}^m r_{jk} x_k) / r_{jj}
 \end{aligned}$$

Suppose that X and B are $m \times n$ matrices, and we want to solve $RX = B$ for X —this is equivalent to solving $Rx = b$ for n different right-hand sides b (the n columns of B). One way to solve the $RX = B$ for X is to apply the standard backsubstitution algorithm, above, to each of the n columns in sequence.

- (a) Give the asymptotic cache complexity $Q(m, n; Z)$ (in asymptotic Θ notation, ignoring constant factors) of this algorithm for solving $RX = B$.
- (b) Suppose $m = n$. Propose an algorithm for solving $RX = B$ that achieves a better asymptotic cache complexity (by cache-aware/blocking or cache-oblivious algorithms, your choice). Can you gain the factor of $1/\sqrt{Z}$ savings that we showed is possible for square-matrix multiplication?