18.335 Problem Set 3

Due Friday, 26 March 2021.

Problem 1: QR and orthogonal bases

(a) Trefethen, problem 10.4.

(b) Trefethen, problem 28.2,

Problem 2: Schur fine

In class, we will show that any square $m \times m$ matrix A can be factorized as $A = QTQ^*$ (the *Schur factorization*), where Q is unitary and T is an upper-triangular matrix (with the same eigenvalues as A, since the two matrices are similar).

(a) A is called "normal" if $AA^* = A^*A$. Show that this implies $TT^* = T^*T$. From this, show that T must be diagonal. Hence, any normal matrix (e.g. unitary or Hermitian matrices) must be unitarily diagonalizable.

(b) Given the Schur factorization of an arbitary A (not necessarily normal), describe an algorithm to find the eigenvalues and eigenvectors of A, assuming for simplicity that all the eigenvalues are distinct. The flop count (count of real $\pm, \times, \div$; assume that your matrices are all real) should be asymptotically $Km^3 + O(m^2)$; give the constant K.

Problem 3: Distribution and association

Suppose you want to compute $x^T(AB+CD)y$, where $x, y \in \mathbb{R}^m$ and $A, B, C, D \in \mathbb{R}^{m \times m}$ for $m = 1000$. You code it up in Julia in two ways:

- `x' * (A*B + C*D) * y`
- `x'*A*B*y + x'*C*D*y`

(a) Which of these two would you expect to be faster, and why? (Note that $*$ in Julia is "left-associative:" performed from left to right, unless you change the order with parentheses.)

(b) Try it and see if it maches your prediction.

A good package for benchmarking in Julia is BenchmarkTools.jl — install it with `] add BenchmarkTools`, load it with `using BenchmarkTools`, allocate random inputs and time them with e.g. `@btime $x' * ($A*$B + $C*$D) * $y`; the $ signs tell the benchmark to evaluate the global variables like `x` before benchmarking to avoid an artificial slowdown (global variables are otherwise slow in Julia).

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):

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

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?