18.335 Mid-term Exam (Fall 2009)

Problem 1: Caches and QR (30 pts)

In class, we learned the Gram-Schmidt and modified Gram-Schmidt algorithms to form the (reduced) $A = QR$ factorization of an $m \times n$ matrix A (with independent columns $a_1, a_2, \ldots$ and $n \leq m$). In particular, for simplicity let us consider the computation of the $m \times n$ matrix Q only (whose columns are the orthonormal basis for the column space of A), not worrying about keeping track of R, and for simplicity consider classical (not modified) Gram-Schmidt:

$q_1 = a_1 / \|a_1\|$
`for` $j = 2, 3, \ldots, n$
  $v_j = a_j - \sum_{i=1}^{j-1} q_i (q_i^* a_j)$
  $q_j = v_j / \|v_j\|$

In this question, you will consider the cache complexity of this algorithm with an ideal cache of size Z (no cache lines). If the algorithm is implemented directly as written above, there is little temporal locality and $\Theta(mn^2)$ misses are required, independent of Z. You are also **given** that you can multiply an $m \times n$ matrix with an $n \times p$ matrix using $\Theta(mn + np + mp + mnp/\sqrt{Z})$ misses, and can add two $m \times n$ matrices using $\Theta(mn)$ misses.

1. Suppose that n is even and we have performed QR factorization (by some algorithm) on the *first-half* $n/2$ columns of A to obtain an $m \times (n/2)$ matrix Q_1, and also on the *second-half* $n/2$ columns *separately* to obtain an $m \times (n/2)$ matrix Q_2. Using Q_1 and Q_2 , describe how to (efficiently) find the $m \times n$ matrix Q from the QR factorization of A, and give the number of cache misses (in Θ notation, ignoring constant factors).

2. Describe an algorithm to compute the Q from the QR factorization of A that has fewer than $\Theta(mn^2)$ misses asymptotically, and give the number of cache misses (in Θ notation, ignoring constant factors). (You can describe either a cache-oblivious or blocked algorithm, but I find a recursive cache-oblivious algorithm easier.) You can assume that n is a power-of-2 size, for convenience.

Problem 2: Lanczos (30 pts)

Let A be a Hermitian $m \times m$ matrix with eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_m$ and corresponding orthonormal eigenvectors $\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_m$. Consider the Lanczos algorithm applied to A:

$\beta_0 = 0$, $q_0 = 0$, $b =$ `arbitrary`, $q_1 = b/\|b\|$
`for` $n = 1, 2, 3, \ldots$
  $v = Aq_n$
  $\alpha_n = q_n^T v$
  $v \leftarrow v - \beta_{n-1} q_{n-1} - \alpha_n q_n$
  $\beta_n = \|v\|$
  $q_{n+1} = v/\beta_n$

After m steps, recall that this gives a unitary matrix $Q = (q_1 q_2 \cdots q_m)$ and a tridiagonal matrix $T = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \beta_2 & \alpha_3 & \ddots \\ & & \ddots & \ddots \end{pmatrix}$ such that $AQ = QT$.

Suppose that the initial b is orthogonal to one of the eigenvectors $\hat{q}_i$ corresponding to a simple (not repeated) eigenvalue λ_i. Explain why the Lanczos process must break down ($\beta_n = 0$ for some n) if it is carried out in *exact arithmetic* (no rounding), and the T_n matrix (the $n \times n$ upper-left corner of T) at the n-th step (where breakdown occurs) cannot have an eigenvalue λ_i.

Problem 3: Backwards stability (30 pts)

Let A be any invertible $m \times m$ matrix and b be any vector in $\mathbb{C}^n$, and consider the function $f(A, b) = A^{-1}b$: that is, the function returning the solution to $Ax = b$. Now, consider the analogous function $\tilde{f}(A, b)$ implemented in floating-point arithmetic by a **backwards-stable** algorithm, e.g. Gaussian elimination with partial pivoting, or Householder QR factorization. That is, if we let $f(A, b) = x$ (the solution: x is the *output* in this case) and $\tilde{f}(A, b) - f(A, b) = \delta x$ (the rounding error in the solution), then there is some δA and δb where $(A + \delta A)(x + \delta x) = b + \delta b$ and δA and δb are .... (yadda yadda...you should know the precise definition by now).

Show that if $\tilde{f}(A, b)$ is backwards stable with respect to the inputs A and b, then it must be backwards stable with respect to A **alone**. That is, find a small $\delta A' = \|A\| O(\varepsilon_{\text{machine}})$ such that $(A + \delta A')(x + \delta x) = b$.

(Hint: if you need to construct a matrix to turn one vector into another, you can always use a unitary rotation followed by a rescaling. And, of course, you can pick any convenient norm that you want, by the equivalence of norms.)