18.335 Problem Set 1
Due Wednesday, 17 September 2008.

Problem 1: LU revisited
Trefethen, problem 20.4.

Problem 2: LU-ish updates
Suppose that we are given the LU factorization $A = LU$ for the $m \times m$ nonsingular matrix $A$ (again, not worrying about row swaps/pivoting or roundoff errors for now). Now, we change $A$ to $A = A + xy^T$ for some $x, y \in \mathbb{R}^m$ (this is a rank-1 update of $A$). We would like to find the new LU factorization $A = LU$ as quickly as possible.

It turns out that this is a little too hard, so we will relax the problem by supposing that, instead of $L$ being lower triangular, we let $L = M_1 M_2 \cdots M_N$ be a product of matrices $M_k$ which are each the identity matrix plus exactly one nonzero element either above or below the diagonal—let’s call these “near-identity matrices” (a term I just made up). (In traditional LU, these are all lower triangular.) Assume $N$ is $O(m^2)$.

So, you are given the $M_k$ matrices and $U$ (which is still upper triangular), and now you want to find $U$ (upper triangular) and the new $M_k$ near-identity matrices to define $L = M_1 M_2 \cdots M_N$ (for some new $N$). (Note that $L$ need not be lower-triangular; we only require that the $M_k$ matrices be near-identity as defined above.) And we want to do it in $O(m^2)$ time [rather than the $\Theta(m^3)$ time to recompute an LU factorization from scratch].

(a) Assume $N$ is $O(m^2)$. Explain why the storage for $L$ (or its equivalent in terms of the $M_k$’s) and the time to solve $La = b$ can be both $O(m^2)$, just like for traditional LU.

(b) Show that $\hat{A} = L(U + uv^T)$ for some $u, v \in \mathbb{R}^m$ that can be computed in $O(m^2)$ operations.

(c) Show that your answer above is equivalent to writing $\hat{A} = LBD$ where $B$ is an $m \times (m+1)$ matrix and $D$ is an $(m+1) \times m$ matrix. (Hint: $B$ and $D$ are made directly out of $u, U, v^T$, and 1’s/0’s with no arithmetic required. Make $u$ the first column of $B$.)

(d) Your matrix $B$ should be “almost” upper triangular already. Show that, in $O(m)$ operations, you can convert it so the last $m$ columns form an upper-triangular matrix $\hat{U}$ and the first column has only a single nonzero entry in the $\ell$-th row for some $1 \leq \ell \leq m$. That is, show how you can factorize $B$, in $O(m)$ operations, as $B = \hat{L}(\alpha e_\ell, \hat{U})$ for matrix $\hat{L}$ matrix that is the product of $O(m)$ near-identity matrices $M_k$, and some real number $\alpha$ [where $e_\ell$ denotes the column vector with a 1 in the $\ell$-th row and zeros in other rows, and $(\alpha e_\ell, \hat{U})$ denotes the matrix whose first column is $\alpha e_\ell$ and whose remaining columns are the columns of $\hat{U}$].

(e) You now have $\hat{A} = \hat{L}\hat{L}(\alpha e_\ell, \hat{U})D$. Show that $(\alpha e_\ell, \hat{U})D$ is almost upper triangular, except for (at most) one row. Explain how you can convert this back into upper-triangular form with at most $O(m^2)$ operations.

(f) Combining all of the above, show that you now have $L$ (in terms of the $M_k$’s) and $U$ in $Km^2 + O(m)$ flops (adds/subtracts/multiplies), and give the leading coefficient $K$. For this part and for the next part, assume that your starting $L$ was found from ordinary LU decomposition via $m - 1$ elimination steps, so your initial $N$ is $N = m(m - 1)/2$.

(g) Using the above procedure repeatedly (not worrying about roundoff error), we can perform $M$ rank-1 updates in $O(Mm^2)$ flops. How big does $M$ have to be before it would be fewer operations just to re-do the LU factorization from scratch ($2m^3/3$ flops)? If you looked at actual computing time with optimized code, do you think the actual break-even point would be reached for $M$ smaller or larger than this, and why?
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.

(a) The code matmul_by_column.m posted on the 18.335 webpage 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 $m = 1, \ldots, 1000$, storing the flop rate ($2m^3$/nanoseconds) in an array gfpops(m), is:

```matlab
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);
end
```

(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

(d) Assuming Matlab uses something like your “improved” algorithm from part (c) to do matrix-vector products, compute the cache complexity of matmul_by_column. Compare this to the cache complexity of the blocked matrix-matrix multiply from class. Does this help to explain your results from part (a)?