Math 128a - Homework 9 - Due May 9

1) Let A be m-by-n, B be n-by-k and C be m-by-k, and x be k-by-1. Define matrix vector multiplication in the usual way: $y = B \cdot x$ is n-by-1 with $y_i = \sum_{j=1}^{k} B_{i,j} \cdot x_j$. Now suppose that $A \cdot (B \cdot x) = C \cdot x$ for all k-by-1 vectors x. Show that C must be given by

$$C_{i,j} = \sum_{m=1}^{n} A_{i,m} \cdot B_{m,j}$$

This explains why matrix-matrix multiplication is defined the way it is.

Answer: Fix some $j \in \{1, \ldots, k\}$ and let x be the k-by-1 vector with a 1 in the j'th place and 0's everywhere else; in other words $x_i = 0$ if $i \neq j$ and $x_j = 1$. Then let $v = C \cdot x$; v is a m-by-1 column vector with entries $v_i = C_{i,j}$. Also let $w = B \cdot x$; now w is a n-by-1 column vector with entries $w_l = B_{l,j}$. Thus we have $v = A \cdot w$, so

$$v_i = \sum_{l=1}^{n} A_{i,l} \cdot w_l$$

and hence

$$C_{i,j} = \sum_{l=1}^{n} A_{i,l} \cdot B_{l,j} \ .$$

2) Let $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ be m-by-n, where A_{ij} is m_i-by-n_j (we clearly assume that $m = m_1 + m_2$ and $n = n_1 + n_2$. A is sometimes called a "block matrix". Similarly, let $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$ be n-by-k, where B_{ij} is n_i-by-k_j (we clearly assume that $k = k_1 + k_2$). Prove the following "block matrix multiplication formula"

$$A \cdot B = \begin{bmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{bmatrix}$$

This basically says that if the entries of A and B are themselves matrices with appropriate dimensions, the usual matrix multiplication formula works by substituting the blocks into the formula.

Answer: To keep notation straight, we let $A(i,j)$ denote the (i,j) entry of the entire matrix A, and $A_{s,t}(i,j)$ denote the (i,j) entry of the block s,t where s and t are in $\{1,2\}$. As usual we have:

$$(A \cdot B)(i,j) = \sum_{s=1}^{n} A(i,s) \cdot B(s,j)$$

We can consider each block entry of $A \cdot B$ separately but after doing the 1,1 block it will be clear that the others are the same. In the 1,1 block $1 \leq i \leq m_1$ and $1 \leq j \leq k_1$. Then

$$\begin{aligned} (A \cdot B)(i,j) &= \sum_{s=1}^{n_1} A(i,s) \cdot B(s,j) + \sum_{s=n_1+1}^{n} A(i,s) \cdot B(s,j) \\ &= \sum_{s=1}^{n_1} A_{1,1}(i,s) \cdot B_{1,1}(s,j) + \sum_{s=1}^{n_2} A_{1,2}(i,s) \cdot B_{2,1}(s,j) \\ &= (A_{1,1} \cdot B_{1,1})(i,j) + (A_{1,2} \cdot B_{2,1})(i,j) \\ &= (A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1})(i,j) \ . \end{aligned}$$

The other three block entries of $A \cdot B$ are similar.

3) Assuming that A is square and nonsingular, show that

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \cdot \begin{bmatrix} A & B \\ 0 & D - CA^{-1}B \end{bmatrix}$$

where all the blocks have dimensions as in the last problem. $D-CA^{-1}B$ is called the *Schur complement* of A, and shows up as an intermediate result in Gaussian elimination. Hint: use the last result of the last question.

Answer: Here we just calculate using the result from the last problem:

$$\begin{aligned}
\begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \cdot \begin{bmatrix} A & B \\ 0 & D-CA^{-1}B \end{bmatrix} &= \begin{bmatrix} I\cdot A + 0\cdot 0 & I\cdot B + 0\cdot (D-CA^{-1}B) \\ (CA^{-1})\cdot A + I\cdot 0 & (CA^{-1})\cdot B + I\cdot (D-CA^{-1}B) \end{bmatrix} \\
&= \begin{bmatrix} A & B \\ CA^{-1}A & CA^{-1}B + D - CA^{-1}B \end{bmatrix} \\
&= \begin{bmatrix} A & B \\ C & D \end{bmatrix}
\end{aligned}$$

4) Prove the *Sherman-Morrison formula*: If A is nonsingular, u and v are column vectors such that $A+u\cdot v^T$ is nonsingular, then $(A+u\cdot v^T)^{-1} = A^{-1} - A^{-1}\cdot u\cdot v^T\cdot A^{-1}/(1+v^T\cdot A^{-1}\cdot u)$ is an explicit expression for the inverse of $A+u\cdot v^T$. Hint: Multiply by $A+u\cdot v^T$ and simplify.

Answer: To show that this equation holds we have to show that, when we multiply the right-hand side by $A+u\cdot v^T$, we get the identity. Here is the multiplication:

$$\begin{aligned}
\left(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}\right)\cdot\left(A+uv^T\right) &= I + A^{-1}uv^T - \frac{A^{-1}uv^T + A^{-1}u(v^TA^{-1}u)v^T}{1+v^TA^{-1}u} \\
&= I + A^{-1}uv^T - \frac{A^{-1}uv^T + (v^TA^{-1}u)A^{-1}uv^T}{1+v^TA^{-1}u} \\
&\quad \text{(because } v^TA^{-1}u \text{ is a scalar)} \\
&= I + A^{-1}uv^T - \frac{1+v^TA^{-1}u}{1+v^TA^{-1}u}A^{-1}uv^T \\
&= I
\end{aligned}$$

5) Suppose that we have done Gaussian elimination on A, so that solving $Ax=b$ for a new b costs just $O(n^2)$, since we can reuse A's L and U factors. Show that the following algorithm solves $(A+u\cdot v^T)x=b$ in $O(n^2)$ time. This is much cheaper than doing Gaussian elimination on $A+u\cdot v^T$, which would cost $2/3n^3$. This is useful because it is common to have to solve several different systems of linear equations where the matrices differ just by adding $u\cdot v^T$ for some columns vectors u and v. Hint: Use the Sherman-Morrison formula.

1) Solve $Az=b$ for z
2) Solve $Ay=u$ for y
3) Compute $\alpha = v^Ty$ (a dot product)
4) Compute $\beta = v^Tz$ (a dot product)
5) Compute $x = z - \frac{\beta}{1+\alpha}y$ (adding multiple of one vector to another)

Answer: Steps 1 and 2 are $O(n^2)$ while steps 3, 4 and 5 are $O(n)$, therefore the whole algorithm works in $O(n^2)$ time. That it solves $(A+uv^T)x=b$ follows from the Sherman-Morrison formula:

$$\begin{aligned}
x &= z - \frac{\beta}{1+\alpha}y \\
&= A^{-1}b - \frac{v^TA^{-1}b}{1+v^TA^{-1}u}(A^{-1}u)
\end{aligned}$$

$$
\begin{aligned}
&= \left(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}\right) b \\
&= (A+uv^T)^{-1}b
\end{aligned}
$$

6) Suppose that we need to solve $Ax_i = b_i$ for $i = 1, ..., m$, i.e. for m vectors b_i. There are two obvious algorithms for this:

```
Algorithm 1
Use Gaussian elimination to compute the L and U factors of $A$
For $i = 1$ to $m$
    Use the $L$ and $U$ factors to solve $Ax_i = b_i$
end

Algorithm 2
Use Gaussian elimination to compute the L and U factors of $A$
For $i = 1$ to $n$              ... compute the inverse of $A$
    Use the $L$ and $U$ factors to solve $Ay_i = e_i$, where $e_i$ is the $i$-th column of $I$
end
... Note that $A^{-1} = [y_1, y_2, ..., y_n]$
for $i = 1$ to $m$
    $x_i = A^{-1} \cdot b_i$               ... matrix-vector multiplication
end
```

Explain why the n-by-n matrix $[y_1, ..., y_n]$ gotten by putting the vectors y_i together in matrix is A^{-1}. Count the operations for each algorithm (your answers should look like $c_1n^3 + c_2mn^2 + O(n^2)$, where you need to figure out the constants c_1 and c_2). Count all operations (multiplication, addition, subtraction and division) equally. Conclude that Algorithm 1 is faster than Algorithm 2. This means that you should never compute an explicit inverse of A to solve linear systems of equations, no matter how many of them you have; instead you should always use A's L and U factors.

Answer: We count multiplications, additions, subtractions and divisions as having the same cost. This is an approximation to the actual cost of arithmetic (not the topic of this course). Generally, modern computers do multiplication, addition and subtraction in the same time (one cycle, the minimum to do any operation), whereas division costs several times as much (up to 20x on some machines). Since the number of divisions is so much smaller than the number of other operations, we usually don't worry much about it. (Since multiplication and addition almost always come in pairs, we can simply get the number of multiplications by dividing the quantities below by 2.)

The n-by-n matrix $[y_1, \ldots, y_n]$ is A^{-1} because $Ay_i = e_i$ implies $A \cdot [y_1, \ldots, y_n] = [e_1, \ldots, e_n] = I$.

For algorithm 1, Gaussian elimination takes $\frac{2}{3}n^3 + O(n^2)$ operations, and solving each equation then takes $2n^2 + O(n)$ operations (solving 2 triangular systems, one with L and one with U) so all together we have $\frac{2}{3}n^3 + 2mn^2 + O(n^2) + O(mn)$.

For algorithm 2, Gaussian elimination is still $\frac{2}{3}n^3 + O(n^2)$. Computing A^{-1} is solving n equations and so takes $n(2n^2+O(n)) = 2n^3+O(n^2)$ operations if done straightforwardly, but if we take advantage of all the zeros in e_i we can save a little work, dropping the cost to $\frac{4}{3}n^3 + O(n^2)$. Then, each matrix-vector multiplication takes $2n^2$ operations, so altogether we have $\frac{2}{3}n^3 + \frac{4}{3}n^3 + 2mn^2 + O(n^2) = 2n^3 + 2mn^2 + O(n^2)$ operations if we take advantage of the zeros in the e_i, or even more if we do not.

Either way, algorithm 2 is slower than algorithm 1.