# CS 422: Assignment 1

Due: Wednesday, February 6, 2008 (In Lecture)

Scoring for each problem is on a 0-to-5 scale ( 5 = complete success, 4 = overlooked a small detail, 3 = good start, 2 = right idea, 1 = germ of the right idea, 0 = missed the point of the problem.) One point will be deducted for insufficiently commented code. Unless otherwise stated, you are expected to utilize fully MATLAB's vectorizing capability subject to the constraint of being flop-efficient. Test drivers and related material are posted on the course website `http://www.cs.cornell.edu/courses/cs422/2008sp/`. For each problem submit output and a listing of all scripts/functions that *you* had to write in order to produce the output. You are allowed to discuss *background* issues with other students, but the codes you submit must be your own.

**P1. (Floating Point Representation)**

The binomial coefficient $n$-choose-$k$ is defined by

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{k!(n-k)!}$$

The 4-by-4 Pascal matrix is given by

$$P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 3 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 3 & 3 & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{bmatrix}$$

In general, $P_n = B_n B_n^T$ where $B_n$ is a lower triangular arrangement of the "Pascal Triangle". In particular, if $i \geq j$ then

$$B_n(i,j) = \begin{pmatrix} i-1 \\ j-1 \end{pmatrix}$$

The MATLAB function `pascal(n)` can be used to generate $P_n$. It turns out that the condition of these matrices grows rapidly with $n$. (See P2.)

In this problem we want to explore how big the mantissa must be for it to be possible to represent $P_n$ exactly in floating point. The central issue is whether or not

$$P_n(n,n) = \begin{pmatrix} 2n-2 \\ n-1 \end{pmatrix}$$

can be represented exactly. The following table shows how to reason about the mantissa length that is required to store this number:

| $n$ | $P_n(n,n)$ | Base-10 | Base-2 | Floating Point | Mantissa Bits Required |
|---|---|---|---|---|---|
| 1 | 1 | $1 \times 2^0$ | $1 \times 2^0$ | $1. \times 2^{0+0}$ | 0 |
| 2 | 2 | $2 \times 2^0$ | $10 \times 2^0$ | $1. \times 2^{0+1}$ | 0 |
| 3 | 6 | $3 \times 2^1$ | $11 \times 2^1$ | $1.1 \times 2^{1+1}$ | 1 |
| 4 | 20 | $5 \times 2^2$ | $101 \times 2^2$ | $1.01 \times 2^{2+2}$ | 2 |
| 5 | 70 | $35 \times 2^1$ | $100011 \times 2^1$ | $1.00011 \times 2^{5+1}$ | 5 |
| 6 | 252 | $63 \times 2^2$ | $111111 \times 2^2$ | $1.11111 \times 2^{5+2}$ | 5 |
| 7 | 924 | $231 \times 2^2$ | $11100111 \times 2^2$ | $1.1100111 \times 2^{7+2}$ | 7 |

Write a script `P1` that prints a 40-line table. The $n$-th line should display $n$ and the number of mantissa bits required to store $P_n(n,n)$ exactly. Work with base-2 logarithms–do not explicitly work with $P_n(n,n)$. Do not worry about exponent ranges.

Submit a listing of your `P1` implementation and any supporting functions that it uses together with the table. This is a good occasion to review `for`, `while`, `if`, `floor`, `ceil`, `log2`, `rem`, `nchoosek`, `factorial`, `disp`, `fprintf`, etc.

**P2. (Condition Number and Accuracy)**

Write a script `P2` that for $n = 5{:}5{:}20$ displays the solution to the linear system $Ax = b$ where `A = pascal(n)` and $b_i$ is the sum of the entries in $P_n(i,:)$. Use `%25.15f` format. Your script should also display the 2-norm condition of the linear system and the product (machine epsilon)*(relative error). You must obtain the solution via the factorization `[L,U,P] = lu(A)`. Submit a copy of your implementation and the output. Review `lu`, `eps`, `cond`, and `norm`.

**P3. (Using the LU Factorization)**

Write a MATLAB function `[X,Y] = Solve(A,B,C,u,v)` that returns $n$-by-$n$ solutions to the (assumed nonsingular) linear systems $AXC = B$ and $AY(C + uv^T) = B$ where $A, B, C \in \mathbb{R}^{n\times n}$ and $u, v \in \mathbb{R}^n$ are given. Make effective use of the LU factorization and the Sherman-Morrison formula

$$(C + uv^T)^{-1} = C^{-1} + \alpha w z^T$$

where $w = C^{-1}u$, $z^T = v^T C^{-1}$, and $\alpha = -1/(1 + v^T C^{-1} u)$. Submit a listing of your implementation and the output produced when the test script `P3` is applied.

**P4. (A Fast Matrix-Vector Multiply)**

We say that $A \in \mathbb{R}^{n\times n}$ has *property S* if there are $n$-vectors $d$, $u$, and $v$ such that

$$a_{ij} = \begin{cases} u_i v_j & \text{if } i > j \\ d_i & \text{if } i = j \\ v_i u_j & \text{if } i < j \end{cases}$$

An example:

$$d = \begin{bmatrix} 10 \\ 20 \\ 30 \\ 40 \end{bmatrix}, \quad u = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad v = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} \quad \Rightarrow A = \begin{bmatrix} 10 & 10 & 15 & 20 \\ 10 & 20 & 18 & 24 \\ 15 & 18 & 30 & 28 \\ 20 & 24 & 28 & 40 \end{bmatrix}$$

Note that $u_1$ and $v_n$ are not "used" in the definition. This is a notational convenience. It also makes the construction of an $S$-matrix easy:

```
A = diag(d) + tril(u*v',-1) + triu(v*u',1)
```

In this problem you are to implement a MATLAB function

```
  function [y,nFlops] = Sprod(d,u,v,x)
% d, u, v, and x are column n-vectors.
% y = (diag(d) + tril(u*v',-1) + triu(v*u',1))*x
% nFlops is the approximate number of flops required.
```

It should require $O(n \log n)$ flops.

The key to achieving this level of efficiency is recursion. Suppose $n = n_1 + n_2$ and

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad A_{11} \in \mathbb{R}^{n_1\times n_1},\ A_{12} \in \mathbb{R}^{n_1\times n_2},\ A_{21} \in \mathbb{R}^{n_2\times n_1},\ A_{22} \in \mathbb{R}^{n_2\times n_2}$$

is an $S$-matrix. It follows that $A_{11}$ and $A_{22}$ are $S$-matrices and $A_{12}$ and $A_{21}$ are rank-1 matrices. If $n_1 \approx n_2 \approx n/2$ then we can split a given $S$-matrix/vector product into a pair of half-sized $S$-matrix/vector products and a pair of rank-1 matrix/vector products:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 \\ A_{21}x_1 + A_{22}x_2 \end{bmatrix}$$

Recall that the product of a rank-1 matrix $fg^T$ and a vector $h$ is linear since $(fg^T)h = f(g^T h)$. Thus, the products $A_{12}x_2$ and $A_{21}x_1$ require $O(n)$ flops instead of the usual $O(n^2)$ flops.

Test your implementation of `Sprod` with the script `P4` found on the website. Submit a listing of `Sprod` and the output produced upon running `P4`. In counting flops, take into consideration only those flops that arise in the context of vector inner products and saxpys.