CS 421: Assignment 1

Due: Monday, September 11, 2006 (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/cs421/2006fa/`. 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)

Assume that p is an integer that satisfies $1 \leq p \leq 62$. *Positive machine numbers* have the form

$$x = \left(1 + \sum_{i=1}^{p-1} \frac{b_i}{2^i}\right) \times 2^E$$

where each b_i is either 0 or 1 and E is an integer that satisfies

$$-(2^{63-p} - 2) \ \leq \ E \leq \ (2^{63-p} - 1)$$

If $p = 53$ then this is essentially IEEE double precision format. For a given p, let $\phi(p)$ be the largest integer k such that 10^k is a machine number. Write a MATLAB script that prints p and $\phi(p)$ for p = 1:62. Submit output and the script that generated it. Include enough comments in the script so that I can reconstruct your reasoning. Good time to review basic MATLAB constructs: `for, while, if, log10, log2, find, ...` .

P2. (Floating Point Arithmetic)

Write a paragraph or so that explains the output produced by following script:

```
x = -20;
sum = 1; k = 1; term = x; next = sum + term;
while next~=sum
    sum = next; k = k+1; term = x*term/k; next = sum + term;
end
format long
sum = next
exact = exp(x)
```

A copy of the script is on the website.

P3. (Setting Up and Solving a Linear System)

The Chebychev polynomials are defined by

$$T_j(x) = \begin{cases} 1 & \text{if } j = 0 \\ x & \text{if } j = 1 \\ 2xT_{j-1}(x) - T_{j-2}(x) & \text{if } j > 1 \end{cases}$$

Let $f(x) = e^{-x}\cos(2x)$ and for i = 1:11 define $x_i = (i-1)/10$. Write a MATLAB script that determines $a \in \mathbb{R}^{11}$ such that if

$$p(x) = \sum_{j=1}^{11} a_j T_{j-1}(x)$$

then $p(x_i) = f(x_i)$ for i = 1:11. Your script should display the a_i to 16 decimals. This is *Chebychev interpolation*. Note that $p(x_i) = f(x_i)$ is a linear equation involving $a_1, \ldots, a_{11}$. Your script should set up the 11-by-11 linear system and solve it using \. In setting up the matrix of coefficients, make use of MATLAB's vectorizing capabilities. Submit script and output.

P4. (Error and Condition)

Consider the following function that is posted on the website:

```
  function v = Recover(A,v0,k)
% A is an n-by-n nonsingular matrix, v0 is a column n-vector, and
% k is a positive integer.
% Multiplies v0 by A a total of k times and then attempts to recover
% v0 by solving a sequence of k linear systems involving A.
  for j=1:k
     v0 = A*v0;
  end
  for j=1:k
     v0 = A\v0;
  end
  v = v0
```

As it stands, this implementation involves $k(2n^3/3)$ flops. Modify the script so that it involves $2n^3/3$ flops. (Assume $k << n$.) Run the script `P4` available on the website and explain the results. Submit output and your improved implementation of `Recover`.

P5. (Structured Solutions)

The matrix exponential of $At \in \mathbb{R}^{n\times n}$ is given by

$$\exp(At) = \sum_{k=0}^{\infty} \frac{A^k t^k}{k!}$$

This matrix function has an important role to play in applied mathematics because the solution to the initial value problem $\dot{x} = Ax,\ x(0) = x_0$ is given by $x(t) = \exp(At)x_0$.

Let $e \in \mathbb{R}^n$ be the vector of all ones. If A has the property that it's off-diagonal entries are nonnegative and $e^T A = 0$, then it can be shown that $F = \exp(At)$ has non-negative entries and $e^T F = e^T$. This means that F is a *Markov matrix*, that is, its entries are probabilities and the values in each of its columns sum to one. The entry f_{ij} is the probability of transitioning from "state j" to "state i" in the underlying Markov process.

The (p, q) Pade approximation to the matrix exponential is given by

$$R_{pq}(At) \ = = \left(\sum_{j=0}^{q} \frac{(p+q-j)!\, q!}{(p+q)!\, j!\, (q-j)!} (-At)^j \right)^{-1} \left(\sum_{j=0}^{p} \frac{(p+q-j)!\, p!}{(p+q)!\, j!\, (p-j)!} (At)^j \right) \ \approx \ \exp(At)$$

If $F = \exp(At)$ is a Markov matrix, then we would also like $R_{pq}(At)$ to be a Markov matrix. The diagonal $(p = q)$ Pade approximants offer some hope. Does it follow that $e^T R_{qq}(At) = e^T$? Does it follow that $R_{qq}(At)$ is a Markov Matrix? Explore these questions by implementing the following function and running the test script `P5`.

```
  function F = Pade(A,t,q)
% A is an n-by-n matrix, t>0, and q is a positive integer.
% F is the (q,q) Pade approximation of At
```

Submit your implentation of `Pade` and the output when `P5` is executed. Submit answers to the two questions posed above.