CS 6210 Assignment 2 Due: 9/23/15 (Wed) at 11pm

Scoring for each problem is on a 0-to-5 scale (5 = complete success, 4 = overlooked a small detail, 3 = good job on half the problem, 2 = OK job on half the problem, 1 = germ of a relevant solution idea, 0 = missed the point of the problem.) Independent of this, one point will be deducted for insufficiently commented code. This will be automatic if any function specification is inadequate, i.e., if the input and output parameters are not fully described. If test scripts are supplied then they are posted on the course website `http://www.cs.cornell.edu/courses/cs6210/2015fa/`. All solution M-Files must be submitted through the CMS system. You are allowed to discuss *background* issues with other students, but the codes you submit must be your own. Never show your code to another student either directly or indirectly.

Topics: Triangular system solving, condition estimation, using LU with pivoting, benchmarking.

1 A Triangular Matrix Equation

This problem is about solving the matrix equation

$$LXL^T + X \;=\; B \tag{1}$$

for $X \in \mathbb{R}^{n\times n}$ given that $L \in \mathbb{R}^{n\times n}$ is unit lower triangular and $B \in \mathbb{R}^{n\times n}$ is symmetric. It can be shown that the solution $X = (x_{ij})$ is unique and symmetric. Equation (1) is a linear system with unknowns x_{ij}. It can be reshaped into the traditional "$Ax = b$" form by setting $A = L \otimes L + I_{n^2}$, $x = X(:)$, and $b = B(:)$. But this is not helpful. The key to a successful solution procedure is to solve for X one column at a time using a forward elimination process. If

$$X = [x_1 \,|\, x_2 \,|\, \cdots |\, x_n] \quad \text{and} \quad B = [b_1 \,|\, b_2 \,|\, \cdots |\, b_n]$$

are column partitionings, then by comparing the first columns in (1) we see that

$$(L + I_n)x_1 \;=\; b_1$$

Thus, x_1 can be computed by solving a lower triangular system. Now let's compare second columns:

$$L \cdot (\text{second column of } XL^T) + x_2 \;=\; L(\ell_{12}x_1 + x_2) + x_2 \;=\; b_2$$

and so

$$(L + I_n)x_2 = b_2 - \ell_{12}x_1.$$

Thus, because x_1 is known, we can compute x_2 via another lower triangular system solve. Using this strategy, develop a general recipe for x_k, $k = 1{:}n$ and proceed to implement the following function:

```
  function X = SymSolve(L,B)
% L is nxn and unit lower triangular and B is nxn and symmetric.
% X solves the systen LXL' + X = B
```

You are free to use the backslash operator \. The grading of your submission will involve benchmarking using `tic` and `toc` so make it efficient as best you can by vectorizing and exploiting structure. FYI, the system (1) is well conditioned so correctness can be inferred by comparing what your implementation produces for a small example with `reshape((kron(L,L)+eye(n*n,n*n))\B(:),n,n)`. Submit your implementation of `SymSolve` to CMS.

2 Exponential of a Skew Symmetric Matrix

The exponential of a matrix S is given by

$$e^S \;=\; \sum_{k=0}^{\infty} \frac{S^k}{k!}.$$

Matrix exponentials occur throughout applied mathematics and figure in lots of important computations. For example, the solution to the initial value problem

$$\dot{x}(t) \;=\; Sx(t) \qquad x(0) = x_0 \tag{2}$$

is given by $x(t) = e^{St}x_0$.

One way to approximate the matrix exponential is use a carefully chosen rational function. For example, since

$$e^z \approx \frac{1+z/2}{1-z/2} \;\equiv\; R_{11}(z)$$

it follows that

$$e^S \;\approx\; \left(I - \frac{S}{2}\right)^{-1}\left(I + \frac{S}{2}\right) \;\equiv\; R_{11}(S).$$

Notice that $= R_{11}(S)$ is the solution to a multiple right hand side linear system:

$$\left(I - \frac{S}{2}\right) X \;=\; \left(I + \frac{S}{2}\right).$$

In this problem you will be computing $R_{11}(S)$ for the case when S is skew-symmetric. Note that if $S = -S^T$ then

$$\left(e^S\right)^T e^S \;=\; e^{S^T}e^S \;=\; e^{-S}e^S \;=\; e^{-S+S} \;=\; I$$

showing that the exponential of a skew-symmetric matrix is orthogonal. The approximation $R_{11}(S)$ is appealing in this situation because it too is orthogonal.

We are interested in how e^S changes if the $(1,n)$ and $(n,1)$ entries in S are set to zero. (If you want a story, we are interested in (2) when $\dot{x}_1(t)$ doesn't directly depend on $x_n(t)$ and when $\dot{x}_n(t)$ doesn't directly depend on $x_1(t)$.) One way to explore the effect of this perturbation is to use the MATLAB matrix exponential function `expm`, e.g.,

```
S      = [0 1 2; -1 0 3; -2 -3 0]); F = expm(S);
Stilde = [0 1 0; -1 0 3;  0 -3 0]); Ftilde = expm(Stilde)
```

We will learn more about `expm` and other matrix exponential methods later on in the course. Suffice it to say here that the `expm` route is expensive, especially because it does not exploit the low-rank connection between S and $\tilde{S}$. Our plan is to approximate F with $R_{11}(S)$ using the LU factorization of $(I - S/2)$ and to compute $R_{11}(\tilde{S})$ by making effective use of that factorization and the Sherman-Morrison-Woodbury formula

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^TA^{-1}U)^{-1}V^TA^{-1} \tag{3}$$

which is given of page 65 of GVL4. To be specific, you are to implement the following function:

```
  function [F,Ftilde] = TwoExpos(S)
% S is an nxn skew-symmetric matrix
% Returns F = R11(S) and Ftilde = R11(Stilde) where Stilde is
% obtained from S by setting the (1,n) and (n,1) entries to zero.
```

Note that F solves

$$\left(I - \frac{S}{2}\right) F \;=\; \left(I + \frac{S}{2}\right)$$

and $\tilde{F}$ solves

$$\left(I - \frac{\tilde{S}}{2}\right) \tilde{F} \;=\; \left(I + \frac{\tilde{S}}{2}\right)$$

where

$$\tilde{S} \;=\; S - s_{1n}e_1e_n^T + s_{1,n}e_ne_1^T$$

with the definitions $e_1 = I_n(:,1)$ and $e_n = I_n(:,n)$. In your implementation, solve for F by using the MATLAB `lu` function. In particular, compute $P(I - S/2) = LU$. Then use the Sherman-Morrison-Woodbury formula given above to compute $\tilde{F}$. You will have to determine $U \in \mathbb{R}^{n\times 2}$ and $V \in \mathbb{R}^{n\times 2}$ so that

$$I - \frac{\tilde{S}}{2} = I - \frac{S}{2} + UV^T$$

In developing your code remember to think "linear system" any time you see something like $A^{-1}Z$ in an expression that you have to evaluate. Again, the key point in this problem is to use *only one* n-by-n LU factorization. Submit your implementation of `TwoExpos` to CMS.

3 1-norm Condition Estimation

Suppose

$$T = \begin{bmatrix} \tau & w^T \\ 0 & S \end{bmatrix}$$

is nonsingular where $S \in \mathbb{R}^{k\times k}$ is upper triangular, $w \in \mathbb{R}^k$, and $\tau \in \mathbb{R}$. Assume that $d \in \mathbb{R}^k$ has unit 1-norm and that $Sy = d$. Figure out how to choose α and β so that the solution to

$$Tz = \begin{bmatrix} \beta \\ \alpha d \end{bmatrix}$$

has the largest possible 1-norm subject to the constraint that the right hand side has unit 1-norm. We'll call this the "$\alpha\beta$-idea". Without loss of generality you can assume that $0 \le \alpha \le 1$. By repeatedly using the $\alpha\beta$-idea we can obtain a "large norm" solution for an arbitrary upper triangular matrix $T \in \mathbb{R}^{n\times n}$. Start with

$$(t_{nn})y = d$$

where $d = (1)$. Then update y and d using the $\alpha\beta$-idea to get a large norm solution to

$$\begin{bmatrix} t_{n-1,n-1} & t_{n-1,n} \\ 0 & t_{nn} \end{bmatrix} y = d$$

Then update y and d using the $\alpha\beta$-idea to get a large norm solution to

$$\begin{bmatrix} t_{n-2,n-2} & t_{n-2,n-1} & t_{n-2,n} \\ 0 & t_{n-1,n-1} & t_{n-1,n} \\ 0 & 0 & t_{nn} \end{bmatrix} y = d$$

And so on until a final length-n y-vector y_{final} is produced. In the spirit of condition estimation (see GVL4 §3.5.4) we would expect $\kappa_1(T) = \| T \|_1 \| T^{-1} \|_1 \approx \| T \|_1 \| y_{final} \|_1$. Implement the following function so that the value it returns is this approximation.

```
function mu = Kappa1Est(T)
% T is an n-by-n nonsingular upper triangular matrix.
% mu is an estimate of its 1-norm condition number.
```

Submit `Kappa1Est` to CMS.

4 Testing your Condition Estimator

Write a script `ShowKappa1Est` that illustrates the "quality" of the 1-norm condition estimator that you developed in the preceding problem. It should do this by running `Kappa1Est` on a host of examples and neatly reporting how it compares to MATLAB's built-in 1-norm condition estimator `condest`. You can generate random examples with prescribed 2-norm condition using the `gallery` function `randsvd`. Find out how via

```
help private/randsvd
```

Here is an example...

```
>> A = gallery('randsvd',5,10000,3)

A =

    0.4628    0.4016    0.3847    0.2855    0.3360
    0.0807    0.0649    0.0758    0.0577    0.0579
   -0.1801   -0.1130   -0.1422   -0.1464   -0.0723
   -0.0337    0.0141   -0.0223   -0.0566    0.0327
   -0.2264   -0.1877   -0.1926   -0.1507   -0.1568

>> svd(A)

ans =

    1.0000
    0.1000
    0.0100
    0.0010
    0.0001
```

Notes to help you do a good job: When I look at the output that your script produces I should get a sense of how `Kappa1Est` performs as a function of n and for various levels of ill-conditioning. In your examples, keep $n \leq 1000$ and condition $\leq 10^{16}$. Remember, that in the condition estimation business, the goal is order-of-magnitude accuracy, not 16-digit accuracy.

To facilitate grading, your script shouldn't take much longer than a minute to run and the nicely formatted, self-explanatory output that it produces should fit on at most two pages. Make sure your script is well commented so that I can go in and make adjustments if necessary. Submit `ShowKappa1Est` to CMS.