Math 5334: Homework 2 Solutions

February 14, 2009

Problem 2.3

For this problem, we rewrite the 13 equations (and 13 unknowns) as a linear system $Ff = b$ and solve for f with $f = F\backslash b$. The script `prob2_3.m` builds F and b and solves for f.

Problem 2.5

a. If the given matrix is symmetric positive definite (SPD), MATLAB's `chol` function will return the upper triangular factor R. If it is not SPD, `chol` will print an error message. I will use $n = 5$ to test each matrix. For example:

```
>> M = magic(5)
M =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> R = chol(M)
??? Error using ==> chol
Matrix must be positive definite.
```

Following the same procedure with the rest of the test matrices we see: `M = magic(5)` (not SPD; not even symmetric)
`H = hilb(5)` (SPD)
`P = pascal(5)` (SPD)
`II= eye(5)` (SPD)
`R = randn(5)` (not SPD)
`R = randn(5); A = R' * R` (SPD)
`R = randn(5); A = R' + R` (not SPD)
`R = randn(5); I = eye(n,n; A = R' + R + n*I` (SPD, usually)
This last matrix can sometimes not be SPD, depending on the random matrix generated.

b. For the Cholesky algorithm, the formulas are easiest to see with a 3×3 example:

$$\begin{aligned}
\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} &= \begin{bmatrix} r_{11} & & \\ r_{21} & r_{22} & \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ & r_{22} & r_{23} \\ & & r_{33} \end{bmatrix} \\
&= \begin{bmatrix} r_{11}^2 & r_{11}r_{12} & r_{11}r_{13} \\ r_{11}r_{12} & r_{12}^2 + r_{22}^2 & r_{12}r_{13} + r_{22}r_{23} \\ r_{11}r_{13} & r_{12}r_{13} + r_{22}r_{23} & r_{13}^2 + r_{23}^2 + r_{33}^2 \end{bmatrix}
\end{aligned}$$

This gives us nine equations (one for each element of the matrix). Or, since we know A and $R^T R$ are symmetric, we have six equations (the other three are redundant):

$$\begin{aligned}
a_{11} &= r_{11}^2 \\
a_{12} &= r_{11}r_{12} \\
a_{13} &= r_{11}r_{13} \\
a_{22} &= r_{12}^2 + r_{22}^2 \\
a_{23} &= r_{12}r_{13} + r_{22}r_{23} \\
a_{33} &= r_{13}^2 + r_{23}^2 + r_{33}^2
\end{aligned}$$

We can instead solve the equations for the R elements given the A elements:

$$\begin{aligned}
r_{11} &= \sqrt{a_{11}} \\
r_{12} &= a_{12}/r_{11} \\
r_{13} &= a_{13}/r_{11} \\
r_{22} &= \sqrt{a_{22} - r_{12}^2} \\
r_{23} &= (a_{23} - r_{12}r_{13})/r_{22} \\
r_{33} &= \sqrt{a_{33} - r_{13}^2 - r_{23}^2}
\end{aligned}$$

These can be generalized for $n \times n$ matrices. The diagonal elements of R are given by

$$r_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} r_{ki}^2}.$$

The off-diagonal elements of R are given by

$$r_{ij} = \left(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj} \right) / r_{ii},$$

where $j > i$.

A simple (and inefficient) version of these equations is implemented in `prob2_5.m`.

Problem 2.8

There are three places in `lutx.m` that use vector (or matrix) operations that you need to turn into loops.

The loops are as follows. I have commented out the original vectorized code:

```
% Swap pivot row
%  if (m ~= k)
%     A([k m],:) = A([m k],:);
%     p([k m]) = p([m k]);
%  end

  if (m ~= k)
     for j = 1:n;
         A([k m],j) = A([m k],j);
     end
         p([k m]) = p([m k]);
  end

% Compute multipliers
%  i = k+1:n;
%  A(i,k) = A(i,k)/A(k,k);
  for i = k+1:n;
      A(i,k) = A(i,k)/A(k,k);
  end



% Update the remainder of the matrix
%  j = k+1:n;
%  A(i,j) = A(i,j) - A(i,k)*A(k,j);
   for j = k+1:n;
      for i = k+1:n;
          A(i,j) = A(i,j) - A(i,k)*A(k,j);
      end
   end
```

On my laptop, in order to take approximately 10 seconds to run, the three codes needed problem sizes:

- `lutx2`: $n \approx 800$
- `lutx`: $n \approx 900$

- `lu`: $n \approx 4000$

Problem 2.11

I've solved this problem in two ways. In the first, I made a copy of `bslashtx.m` and edited it to find the inverse of a matrix A. This method is implemented in `myinv.m`. In the second, I made a simpler version, `myinv2.m`, that does not have checks for special cases like `bslashtx.m` has.

To compare, I make a couple of random matrices and compare the inverse from my functions to that of MATLAB's built-in `inv` function:

```
A = rand(3,3)

A =
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575

X = myinv(A)

X =
   -1.9958    3.0630   -1.1690
    2.8839   -2.6919    0.6987
   -0.0291   -0.1320    1.1282

X = myinv2(A)

X =
   -1.9958    3.0630   -1.1690
    2.8839   -2.6919    0.6987
   -0.0291   -0.1320    1.1282


inv(A)

ans =
   -1.9958    3.0630   -1.1690
    2.8839   -2.6919    0.6987
   -0.0291   -0.1320    1.1282


% now a slightly bigger test
A = rand(10,10);
X = myinv(A);
% I won't print out these larger matrices, but will look at the norm of the
% difference between my X and inv(A)

norm(X - inv(A))
```

```
ans =

   1.3156e-14


X = myinv2(A);

norm(X - inv(A))

ans =

   1.3156e-14
```

Problem 2.19

The given system of equations in matrix form is:

$$\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ n-1 \\ n \end{bmatrix}$$

```
n = 100;

% (a)
a = -1*ones(n-1,1);
d = 2*ones(n,1);
A = diag(a,-1) + diag(d) + diag(a,1);
rhs = (1:n)';

% solve with lutx:
[L,U,p] = lutx(A);
% i.e., LUx = b(p)
% so to solve we want x = U^{-1} L^{-1} b(p):
x1 = U \ (L \ rhs(p));

% solve with bslashtx
x2 = bslashtx(A,rhs);

% (b)
% one simple form of the call to spdiags needs the vector a and b to be the
% same length. So we will append an nth value -1 to the vector a:
c = a;
c(n) = -1;
```



```
A = spdiags([c d c], [-1 0 1], n, n);
x3 = A\rhs;

% (c)
x4 = tridisolve(a,d,a,rhs);

% (d)
cnum = condest(A);
disp(sprintf('condest(A) = %e', cnum));

   condest(A) = 5.100000e+03
```