Solution for Assignment 2

1. (12 marks) Write two Matlab or Octave functions:

```
[u,d,l] = decomt(u,d,l)
      b = solvet(u,d,l,b)
```

Suppose a tridiagonal matrix is given in the form of three vectors: the upper diagonal u, the main diagonal d, and the lower diagonal l, the first function `decomt` performs the LU decomposition using the Gaussian elimination without pivoting. In the output, l is the lower diagonal of the lower bidiagonal factor L, d the main diagonal of the upper bidiagonal factor U, and u the upper diagonal of the upper bidiagonal factor U. Note that the input vectors u, d, and l are overwritten by the outputs. The second function `solvet` takes the outputs from `decomt` as inputs and solves the tridiagonal system with the right-side vector b. On return, the solution is stored in b. In your implementations, you may not use matrices. For submission, along with the two functions `decomt.m` and `solvet.m`, explain the tests carried out. The functions should be well documented following the style given in the sample programs.

Solution. See `decomt.m` and `solvet.m`. A testing program

```
% script file: testsolvet.m
%
% testing decomt.m and solvet.m on random tridiagonal matrices

n = input('matrix size: ');

% random tridiagonal
u = rand(n-1, 1);
d = rand(n, 1);
l = rand(n-1, 1);
% construct the rhs with solution ones(n, 1)
b = zeros(n, 1);
b(1) = d(1) + u(1);
for i = 2:n-1
    b(i) = l(i-1) + d(i) + u(i);
end
b(n) = l(n-1) + d(n);

% LU decomposition
[u, d, l] = decomt(u, d, l);
% solve
b = solvet(u, d, l, b);

% check solution
norm(b - ones(n,1)),
```

2. (12 marks) This problem involves verifying two inequalities

$$\frac{\|b - A\hat{x}\|}{\|A\|\,\|\hat{x}\|} \le \rho\,\beta^{-t}$$

and

$$\frac{\|x - \hat{x}\|}{\|\hat{x}\|} \le \rho\,\mathrm{cond}(A)\beta^{-t},$$

where $\hat{x}$ is the solution computed by Gaussian elimination with partial pivoting, the norm of any vector is

$$\|x\| = \sum_{i=1}^{n} |x_i|$$

and the norm of a matrix with columns a_j is

$$\|A\| = \max_{j} \|a_j\|.$$

You are to experimentally check our claims that ρ in the first inequality is almost always less than β and that the quantity `cond` returned by `decomp` is a satisfactory substitute for $\mathrm{cond}(A)$ in the second inequality.

The function `rand(m,n)` generates an m-by-n matrix whose entries are uniformly distributed between 0 and 1. Since the second inequality requires knowing the exact solution, pick x and compute $b = Ax$. Note that due to rounding errors, the equality $Ax = b$ may not be exact, unless you make sure there is no rounding error in A, b, or x.

Use `decomp` to factor the matrix and compute cond. Use `solve` to compute $\hat{x}$. Compute $\|A\|$, $\|\hat{x}\|$, $\|b - A\hat{x}\|$, and $\|x - \hat{x}\|$. Be sure to save copies of A and b, since they are altered by the functions.

Compute ρ so that the first inequality is actually an equality. If you find that ρ is much larger than β, carefully recheck your program. Large values of ρ are theoretically possible, but they are very rare in practice. They are associated with growth in the size of the elements of the matrix during elimination.

Using your value of ρ, check to see if the second inequality is satisfied with cond in place of $\mathrm{cond}(A)$. If it is not, it is because cond is a severe underestimate for the true $\mathrm{cond}(A)$. Again, such examples are very rare.

Do this problem with several different matrices, including ones with condition numbers close to 1 and with very large condition numbers. Matrices with almost linearly dependent columns have large condition numbers. Such a matrix can be constructed by starting with a matrix with linearly dependent columns and then introducing small perturbations to its entries. For example, the columns of the matrix

$$A = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{bmatrix}$$

are linearly dependent. Thus A is exactly singular. However, the entries of A cannot be exactly represented in floating-point. So, the floating-point approximation of A is nearly singular, that is, it has a large condition number.

Solution. See `testerrors.m`. This program first generates a random integer matrix, then makes the matrix singular by setting the last column as a linear combination of the first two columns. Then it constructs a small diagonal matrix whose diagonal entries are about the reciprocal of the desired condition number. The small diagonal matrix is then added to the singular matrix. The condition number of the perturbed matrix is expected to be about the desired condition number. Note that the matrix is constructed so that its entries can be exactly represented by double precision floating-point numbers. Then the right-hand side is constructed so that the solution consists of all ones. Thus the right-hand side is computed exactly. Consequently, the exact solution of the system is indeed the vector of ones.

3. (6 marks) The inverse of a matrix A can be defined as the matrix X whose columns x_j satisfy

$$Ax_j = e_j,$$

where e_j is the jth column of the identity matrix. Write a function

```
[X, rcondA, pvt] = invert(A)
```

which accepts a matrix A of order n as input and returns a matrix X, an approximation to the inverse of A, as well as the condition estimate and the pivot information. Your function should call `decomp` just once and call `solve` a total of n times, once for each column of X. Leave X as a null matrix (of dimension 0) if `decomp` detects singularity.

You may test your function using the measurement:

$$\texttt{norm}(X - \texttt{inv}(A), 1).$$

Note that $\texttt{inv}(A)$ is an approximation of A^{-1} computed by MATLAB/Octave. The function $\texttt{rand}(m, n)$ generates an m-by-n matrix whose entries are uniformly distributed between 0 and 1.

Solution. See `invert.m`. A testing program:

```
% testing invert.m

n = input('matrix size: ');

A = rand(n, n);

% compute the inverse
[X, rcond] = invert(A);    % general matrix A

if (length(X(:,1)) == 0)
    return;
end

% print condition number
1/rcond,

% residual
```

```
norm(A*X - eye(n), 1),  % error measurements
% actual error
norm(X - inv(A), 1),    % actual error, against inv()
```

Note that the above testing program compares the computed inverse with the MATLAB function `inv()`. A better way is to test on matrices with large condition numbers and whose inverses are known.