AM117 Sample Solutions, HW#5

1./2. Problems 7 and 8, p.188

Results:

```
-------------------------------------------------
a)
-------------------------------------------------
A =

   1.0000   2.0000
   1.0010   2.0000

cond_A =  6002.0
rel_error =  2
relative_residual =  6.6644e-04
error_bound =  4
-------------------------------------------------
b)
-------------------------------------------------
A =

   2.0100   1.9900
   1.9900   2.0100

cond_A =  200.00
rel_error =  1
relative_residual =  0.0050000
error_bound =  1.0000
-------------------------------------------------
c)
-------------------------------------------------
A =

   1  -1  -1
   0   1  -1
   0   0   1

cond_A =  12
rel_error =  0.050000
relative_residual =  0.10000
error_bound =  1.2000
-------------------------------------------------
d)
-------------------------------------------------
A =

   1.00000   0.50000   0.33333
   0.50000   0.33333   0.25000
   0.33333   0.25000   0.20000

cond_A =  748.00
rel_error =  0.020000
```

```
relative_residual =  0.020000
error_bound =  14.960
```

Discussion: The error bounds are kept in all cases, as expected, though with varying degrees of accuracy. (Once the error bound is even achieved.)

Code:

```
function p1
  test_condition_predictions('a)', [1,2;1.001,2],[3;3.001],[1;1],[3;0]);
  test_condition_predictions('b)', [2.01,1.99;1.99,2.01],[4;4],[1;1],[2;0]);
  test_condition_predictions('c)', [1,-1,-1;0,1,-1;0,0,1],[0;2;0],[2;2;0],[1.9;2.1;-0.1])
  test_condition_predictions('d)', hilb(3), [1;7/12;13/30],[1;-2;3],[1.02;-1.96;2.94])
end
function test_condition_predictions(title, A, b, x, xtilde)
  norm_A = norm_inf(A);
  assert(abs(norm_A - norm(A, Inf))<1e-15);

  norm_A_inv = norm_inf(inv(A));


  fprintf('-------------------------------------------------\n');
  disp(title)
  fprintf('-------------------------------------------------\n');
  A

  cond_A = norm_A * norm_A_inv

  residual = A*xtilde-b;
  rel_error = norm(x - xtilde, Inf)/norm(x, Inf)
  relative_residual = norm(residual, Inf) / norm(b, Inf)
  error_bound = cond_A * relative_residual

end
function result = norm_inf(A)
  [m,n] = size(A);
  result = 0;
  for i = 1:m
    result = max(result, sum(abs(A(i,1:n))));
  end
end
function assert(v)
  if (v == 0)
    error('Assertion failed.')
  end
end
```

3. Problem 10, p.189

Results:

```
A =

   25   19
   21   16

Part a)

cond_inf_A =  2024

Part b)

x_exact =

   1.00000
  -1.00000

x_perturbed =

   1.3500
  -1.4600

relative_change_in_x =  0.46000
relative_change_in_b =  0.0016667
predicted_bound_on_relative_change_in_x =  3.3733

Part c)

x_exact =

  -3.0000
   4.0000

x_perturbed =

  -2.6500
   3.5400

relative_change_in_x =  0.11500
relative_change_in_b =  0.010000
predicted_bound_on_relative_change_in_x =  20.240
```

Discussion: As expected, the bounds are kept. However, use this exercise as a warning: In b), a small change in $\boldsymbol{b}$ causes a "relatively small" bound, but it happens to cause a large change in $\boldsymbol{x}$, bringing that change close to the predicted bound. In c), however, the change in $\boldsymbol{b}$ is large, but the change in $\boldsymbol{x}$ is small–and a good ways away from the bound.

Thus observe that these estimates are very dependent on how "sensitive" the solution process is in the direction of $\boldsymbol{b}$. The condition number *only* tells you about the *most sensitive* direction, which your $\boldsymbol{b}$ may not lie in, as for example in c), causing unnecessarily large bounds. Note however that a change of the same ratio as in c) would have almost made us achieve the bound in b).

Code:

```
function p3
  A = [25,19;21,16]

  fprintf('Part a)\n\n')
  cond_inf_A = cond_inf(A)

  fprintf('\nPart b)\n\n')

  b1 = [6;5];
  x_exact = A \ b1
  b2 = b1 + [0.01;-0.01];
  x_perturbed = A \ b2
  relative_change_in_x = norm(x_exact-x_perturbed, Inf)/norm(x_exact, Inf)
  relative_change_in_b = norm(b1-b2, Inf)/norm(b1, Inf)
  predicted_bound_on_relative_change_in_x = cond_inf_A * relative_change_in_b

  fprintf('\nPart c)\n\n')

  b1 = [1;1];
  x_exact = A \ b1
  b2 = b1 + [0.01;-0.01];
  x_perturbed = A \ b2
  relative_change_in_x = norm(x_exact-x_perturbed, Inf)/norm(x_exact, Inf)
  relative_change_in_b = norm(b1-b2, Inf)/norm(b1, Inf)
  predicted_bound_on_relative_change_in_x = cond_inf_A * relative_change_in_b
end
function result = cond_inf(A)
  result = norm(A, Inf) * norm(inv(A), Inf);
end
function assert(v)
  if (v == 0)
    error('Assertion failed.')
  end
end
```

4. Problem 11, p.189

Results:

```
A =

   0.25000   0.35000   0.15000
   0.20000   0.20000   0.25000
   0.15000   0.20000   0.25000

b =

   0.60000
   0.90000
   0.70000

Part a)

cond_inf_A =  30.000

Part b)
```

```
x =

   4.0000
  -2.0000
   2.0000


Part c)

deltaA =

   0.010000   0.000000   0.000000
   0.000000   0.000000  -0.010000
   0.000000  -0.010000   0.000000

deltab =

   0.010000
   0.020000
  -0.030000

xtilde =

   6.0313
  -3.4632
   1.6933

relative_change_in_A =  0.013333
relative_change_in_b =  0.033333
relative_change_in_x =  0.50782
bound_on_change_in_x =  2.3333

Part d)

deltaA =

   0.000000  -0.010000   0.010000
  -0.010000   0.010000   0.000000
   0.010000   0.000000   0.010000

deltab =

   0.020000
   0.010000
  -0.010000

xtilde =

   9.6915
  -5.8696
   1.2049

relative_change_in_A =  0.026667
relative_change_in_b =  0.022222
```

```
relative_change_in_x =  1.4229
bound_on_change_in_x =  7.3333
```

Discussion: All bounds are kept.

Code:

```
function p4
  A = [0.25,0.35,0.15;0.2,0.2,0.25;0.15,0.2,0.25]
  b = [0.6;0.9;0.7]

  fprintf('Part a)\n\n')

  cond_inf_A = cond_inf(A)

  fprintf('\nPart b)\n\n')

  x = A \ b

  fprintf('\nPart c)\n\n')

  deltaA = [0.01,0,0;0,0,-.01;0,-0.01,0]
  deltab = [0.01;0.02;-0.03]

  xtilde = (A+deltaA) \ (b+deltab)
  relative_change_in_A = norm(deltaA, Inf)/norm(A, Inf)
  relative_change_in_b = norm(deltab, Inf)/norm(b, Inf)
  relative_change_in_x = norm(x-xtilde, Inf)/norm(x, Inf)
  bound_on_change_in_x = ...
    cond_inf_A/(1-cond_inf_A*relative_change_in_A)...
    *(relative_change_in_b + relative_change_in_A)

  fprintf('\nPart d)\n\n')

  deltaA = [0,-0.01,0.01;-0.01,0.01,0;0.01,0,0.01]
  deltab = [0.02;0.01;-0.01]

  xtilde = (A+deltaA) \ (b+deltab)
  relative_change_in_A = norm(deltaA, Inf)/norm(A, Inf)
  relative_change_in_b = norm(deltab, Inf)/norm(b, Inf)
  relative_change_in_x = norm(x-xtilde, Inf)/norm(x, Inf)
  bound_on_change_in_x = ...
    cond_inf_A/(1-cond_inf_A*relative_change_in_A)...
    *(relative_change_in_b + relative_change_in_A)
end
function result = cond_inf(A)
  result = norm(A, Inf) * norm(inv(A), Inf);
end
function assert(v)
  if (v == 0)
    error('Assertion failed.')
  end
end
```

5. Problem 19, p.223

a) The best way to try and show that we've designed an algorithm that correctly computes the Crout LU decomposition of a matrix (short of actually *proving* it...) is to code it up and see if it works. For clarity, we're looking for a decomposition of the type

$$\begin{pmatrix} * & & & & \\ * & \ddots & & & \\ * & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & * & * & * \end{pmatrix}\begin{pmatrix} 1 & * & * & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & * \\ & & & 1 & * \\ & & & & 1 \end{pmatrix} = \begin{pmatrix} * & * & * & & \\ * & \ddots & \ddots & \ddots & \\ * & \ddots & \ddots & \ddots & * \\ & \ddots & \ddots & \ddots & * \\ & & * & * & * \end{pmatrix}$$

So, here goes some code:

```
function p5new
  a = random_pentadiag(6)
  [l,u] = crout_pentadiag(a)
  a_minus_lu = a - l*u
end
function [l, u] = crout_pentadiag(a)
  [n, dummy] = size(a);

  l = zeros(n,n);
  u = zeros(n,n);

  l(1:3,1) = a(1:3,1);
  u(1,1) = 1;

  % 1 + 3*2 = 7 op
  u(1,2) = a(1,2)/l(1,1);
  u(2,2) = 1;
  l(2,2) = a(2,2) - l(2,1)*u(1,2);
  l(3,2) = a(3,2) - l(3,1)*u(1,2);
  l(4,2) = a(4,2) - l(4,1)*u(1,2);

  for j = 3:n-2 % 10 op
    u(j-2,j) = a(j-2,j)  / l(j-2,j-2); % 1 op
    u(j-1,j) = (a(j-1,j) - l(j-1,j-2)*u(j-2,j))/l(j-1,j-1); % 3 op
    u(j,j) = 1;
    l(j,j)   =  a(j,j)   - l(j,j-2)*u(j-2,j) - l(j,j-1)*u(j-1,j); % 4 op
    l(j+1,j) =  a(j+1,j) - l(j+1,j-1)*u(j-1,j); % 2 op
    l(j+2,j) =  a(j+2,j);
  end

  j = n-1; % 10 op
  u(j-2,j) = a(j-2,j)  / l(j-2,j-2);
  u(j-1,j) = (a(j-1,j) - l(j-1,j-2)*u(j-2,j))/l(j-1,j-1);
  u(j,j) = 1;
  l(j,j)   =  a(j,j)   - l(j,j-2)*u(j-2,j) - l(j,j-1)*u(j-1,j);
  l(j+1,j) =  a(j+1,j) - l(j+1,j-1)*u(j-1,j);

  j = n; % 8 op
  u(j-2,j) = a(j-2,j)  / l(j-2,j-2);
  u(j-1,j) = (a(j-1,j) - l(j-1,j-2)*u(j-2,j))/l(j-1,j-1);
  u(j,j) = 1;
  l(j,j)   =  a(j,j)   - l(j,j-2)*u(j-2,j) - l(j,j-1)*u(j-1,j);

  lu = l*u

  assert(abs(a-l*u) < 1e-15)
end
function a = random_pentadiag(n)
  a = zeros(n,n);

  for i = 1:n
    for j = max(i-2,1):min(n,i+2)
      a(i,j) = rand;
    end
  end
end
function assert(v)
  if (v == 0)
    error('Assertion failed.')
  end
end
```

And indeed, it works! See here:

```
a =

   0.91792   0.40620   0.03430   0.00000   0.00000   0.00000
   0.01092   0.78839   0.64271   0.32784   0.00000   0.00000
   0.01381   0.02151   0.61738   0.01836   0.21008   0.00000
   0.00000   0.58220   0.69002   0.57234   0.58252   0.05594
   0.00000   0.00000   0.99205   0.59573   0.06704   0.52013
   0.00000   0.00000   0.00000   0.33635   0.83117   0.41192

lu =

   0.91792   0.40620   0.03430   0.00000   0.00000   0.00000
   0.01092   0.78839   0.64271   0.32784   0.00000   0.00000
   0.01381   0.02151   0.61738   0.01836   0.21008   0.00000
   0.00000   0.58220   0.69002   0.57234   0.58252   0.05594
   0.00000   0.00000   0.99205   0.59573   0.06704   0.52013
   0.00000   0.00000   0.00000   0.33635   0.83117   0.41192

l =

   0.91792   0.00000   0.00000   0.00000   0.00000   0.00000
   0.01092   0.78356   0.00000   0.00000   0.00000   0.00000
   0.01381   0.01539   0.60425   0.00000   0.00000   0.00000
   0.00000   0.58220   0.21278   0.32456   0.00000   0.00000
   0.00000   0.00000   0.99205   0.57616  -1.18065   0.00000
   0.00000   0.00000   0.00000   0.33635   0.30414   0.46236

u =
   1.00000   0.44252   0.03737   0.00000   0.00000   0.00000
   0.00000   1.00000   0.81972   0.41840   0.00000   0.00000
   0.00000   0.00000   1.00000   0.01973   0.34768   0.00000
   0.00000   0.00000   0.00000   1.00000   1.56689   0.17235
   0.00000   0.00000   0.00000   0.00000   1.00000  -0.35644
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000

a_minus_lu =

   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00
   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00
   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00
   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00  -1.1102e-16   0.0000e+00
   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00   6.9389e-17   0.0000e+00
   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00
```

It is a trivial excercise for the reader :) to extract the algorithm in mathematical notation from the above code.

b) To count the operations, realize the Crout decomposition effectively goes through the matrix column by column, filling up each of them from top to bottom. Thus we may associate an operations count with each non-zero entry of the matrix:

$$\begin{pmatrix} 0 & 1 & 1 & & & & \\ 0 & 2 & 3 & \ddots & & & \\ 0 & 2 & 4 & \ddots & 1 & & \\ & 2 & 2 & \ddots & 3 & 1 & \\ & & 0 & \ddots & 4 & 3 & 1 \\ & & & \ddots & 2 & 4 & 3 \\ & & & & 0 & 2 & 4 \end{pmatrix}$$

Note that the computation of L is associated with the entries on the diagonal and below, while the entries above the diagonal are associated with U.

From this table, it is easy to see that the first two and last two columns together add $7+10+8=25$ operations. Further, we have $n-4$ columns in the middle with 10 operations each, yielding

$$10(n-4)+25=10n-15.$$

operations. Observe that while this result is still linear in n, the constants have grown more than linearly when compared to the correspdonding result for tridiagonal matrices.

c) Like above, we will simply jot down the number of operations for each element of the factored matrices required for back- and forward substitution:

$$\begin{pmatrix} 0 & 2 & 2 & & \\ & 0 & 2 & 2 & \\ & & 0 & 2 & 2 \\ & & & 0 & 2 \\ & & & & 0 \end{pmatrix}$$

Thus we obtain

$$4(n-2)+2=4n-6$$

for the backsubstitution step. For forward substitution, we write

$$\begin{pmatrix} 1 & & & & \\ 2 & 1 & & & \\ 2 & 2 & 1 & & \\ & 2 & 2 & 1 & \\ & & 2 & 2 & 1 \end{pmatrix}$$

and immediately see that we need

$$5(n-2)+4=5n-6$$

steps. All in all, the substitution process needs

$$9n-12$$

steps.