AM117 Sample Solutions, HW#3

Blurb from the TA

- If you are asked to compare several methods, don't just print their results. You need to discuss which method worked better–and how you judged quality. You should generally have some quantitative measure of quality–the relative error $|\boldsymbol{x} - \boldsymbol{x}_{\text{true}}|/|\boldsymbol{x}_{\text{true}}|$ is typically a good one if it is available.

  As you learn in the second problem, the residual $|A\boldsymbol{x} - \boldsymbol{b}|$ is *not* a good measure of the error.

- If you include computer output, try to keep it concise. Norms (especially norms of errors) are much shorter and more meaningful than full vector output. In general, I'm not interested in intermediate results of your code.

- When you turn in results of computational exercises, include something that makes it credible that you actually did write code for the problem–include the code itself or a (reasonably short) printout of the results.

- Use your own best judgement to turn in *the least amount of material* that you think will convince me that you did the problem and did it correctly.

1. Problem 6, p. 157

Let

$$U = \begin{pmatrix} u_{1,1} & \cdots & u_{1,n} \\ 0 & \ddots & \vdots \\ 0 & 0 & u_{n,n} \end{pmatrix}.$$

We would like to show that

$$\det U_n = u_{1,1} \cdot u_{2,2} \cdots u_{n,n}.$$

For $m \leqslant n$, define the matrix

$$U_m := \begin{pmatrix} u_{1,1} & \cdots & u_{1,m} \\ 0 & \ddots & \vdots \\ 0 & 0 & u_{m,m} \end{pmatrix}.$$

Our proof will closely follow the definition on p. 146. We will proceed by induction on m.

Base case: If $m = 1$, then $U_m = U_1 = (u_{1,1}) = \det(U_1)$, so the claim is trivial.

Induction step: Suppose we know that $\det(U_m) = u_{1,1} \cdot u_{2,2} \cdots u_{m,m}$. We would like to show

$$\det U_{m+1} = u_{1,1} \cdot u_{2,2} \cdots u_{m+1,m+1}.$$

By definition,

$$\det U_{m+1} \overset{\text{Def}}{=} \sum_{j=1}^{m+1} (-1)^{m+1+j} u_{m+1,j} m_{m+1,j},$$

where we have (somewhat arbitrarily) chosen that our sum should go over the $(m+1)$th row. $m_{i,j}$ is the i,j-minor of A, as in the definition. Realize that

$$\begin{aligned}
\det U_{m+1} &= \sum_{j=1}^{m+1} (-1)^{m+1+j} u_{m+1,j} m_{m+1,j}, \\
&= \left[\sum_{j=1}^{m} (-1)^{m+1+j} \underbrace{u_{m+1,j}}_{0} m_{m+1,j} \right] + (-1)^{m+1+m+1} u_{m+1,m+1} m_{m+1,m+1} \\
&= \underbrace{(-1)^{2(m+1)}}_{1} u_{m+1,m+1} \det(U_m) \\
&= u_{1,1} \cdots u_{m,m} \cdot u_{m+1,m+1}
\end{aligned}$$

□

2. Problem 10, p. 158

This problem was to be done in finite-precision arithmetic (3 significant digits, in this case). Here, I emulate this by rounding to hundredths. If you look closely at the resulting matrix, you will find that there are elements on the order of 10^{-2} left throughout the lower-left half. This phenomenon occurs as well if you use double-precision arithmetic, although then the "leftovers" are on the order of machine precision (about 10^{-15}).

```
GNU Octave, version 2.9.8 (i486-pc-linux-gnu).
Copyright (C) 2006 John W. Eaton.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTIBILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).

------------------------------------------------
Part a) -- original matrix
------------------------------------------------
Ab =

   3.02000  -1.05000   2.53000  -1.61000
   4.33000   0.56000  -1.78000   7.23000
  -0.83000  -0.54000   1.47000  -3.38000

------------------------------------------------
Begin step 1
pivot =  3.0200
row(2) <- row(2) - 4.33/3.02 * row(1)
row(3) <- row(3) - -0.83/3.02 * row(1)
Ab =

   3.020000  -1.050000   2.530000  -1.610000
   0.010000   2.060000  -5.400000   9.530000
```

```
  -0.010000  -0.820000   2.150000  -3.810000

------------------------------------------------
Begin step 2
pivot =  2.0600
row(3) <- row(3) - -0.82/2.06 * row(2)
Ab =

   3.02000  -1.05000   2.53000  -1.61000
   0.01000   2.06000  -5.40000   9.53000
  -0.01000   0.00000  -0.01000   0.00000

------------------------------------------------
Backsubstitution
sol =

   1.08000
   4.63000
  -0.00000

residual =

   0.010100
   0.018600
  -0.010800

exact_solution =

   1.00000
   2.00000
  -1.00000

------------------------------------------------
Part b) -- changed A(1,1)
------------------------------------------------
Ab =

   3.01000  -1.05000   2.53000  -1.61000
   4.33000   0.56000  -1.78000   7.23000
  -0.83000  -0.54000   1.47000  -3.38000

------------------------------------------------
Begin step 1
pivot =  3.0100
row(2) <- row(2) - 4.33/3.01 * row(1)
row(3) <- row(3) - -0.83/3.01 * row(1)
Ab =

   3.01000  -1.05000   2.53000  -1.61000
  -0.00000   2.07000  -5.42000   9.55000
   0.01000  -0.83000   2.18000  -3.83000

------------------------------------------------
Begin step 2
```

```
pivot =  2.0700
row(3) <- row(3) - -0.83/2.07 * row(2)
Ab =

   3.01000  -1.05000   2.53000  -1.61000
  -0.00000   2.07000  -5.42000   9.55000
   0.01000  -0.00000   0.01000  -0.01000

------------------------------------------------
Backsubstitution
sol =

   1
   2
  -1

residual =

  -0.0100000
   0.0100000
   0.0100000

percentage_change =

    7.4074
   56.8035
       Inf

exact_solution =

   1.06001
   4.12536
  -0.18537

------------------------------------------------
Part c) -- changed b(3)
------------------------------------------------
b2 =

  -1.6100
   7.2300
  -3.3900

Ab =

   3.02000  -1.05000   2.53000  -1.61000
   4.33000   0.56000  -1.78000   7.23000
  -0.83000  -0.54000   1.47000  -3.39000

------------------------------------------------
Begin step 1
pivot =  3.0200
row(2) <- row(2) - 4.33/3.02 * row(1)
row(3) <- row(3) - -0.83/3.02 * row(1)
```

```
Ab =

   3.020000  -1.050000   2.530000  -1.610000
   0.010000   2.060000  -5.400000   9.530000
  -0.010000  -0.820000   2.150000  -3.820000

------------------------------------------------
Begin step 2
pivot =  2.0600
row(3) <- row(3) - -0.82/2.06 * row(2)
Ab =

   3.02000  -1.05000   2.53000  -1.61000
   0.01000   2.06000  -5.40000   9.53000
  -0.01000   0.00000  -0.01000  -0.01000

------------------------------------------------
Backsubstitution
sol =

   1.1500
   7.2500
   1.0000

residual =

   5.0000e-04
   1.6500e-02
  -1.1500e-02

percentage_change =

    6.4815
   56.5875
       Inf

exact_solution =

   1.1855
   8.6991
   1.5588
```

3. Problem 18, p. 170

The matrix in the problem is called a *Hilbert matrix*. Depending on the size of n, it has an exponentially big condition number, which means we should expect increasingly bad solutions.

(Part a) and b) are treated together.)

```
GNU Octave, version 2.9.8 (i486-pc-linux-gnu).
Copyright (C) 2006 John W. Eaton.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTIBILITY or
```

```
FITNESS FOR A PARTICULAR PURPOSE.  For details, type ‘warranty’.

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).


n=5, cond(hilb(5))=476607

no piv.                   - |A*x-b| =              0 - rel.error [%] =     1.55304e-10
partial piv. (swap)       - |A*x-b| =              0 - rel.error [%] =     3.49344e-11
partial piv. (indirect)   - |A*x-b| =    1.11022e-16 - rel.error [%] =      3.4933e-11
scaled partial piv.       - |A*x-b| =              0 - rel.error [%] =     4.70864e-10

n=6, cond(hilb(6))=1.49511e+07

no piv.                   - |A*x-b| =              0 - rel.error [%] =     6.36985e-09
partial piv. (swap)       - |A*x-b| =              0 - rel.error [%] =     3.17407e-09
partial piv. (indirect)   - |A*x-b| =              0 - rel.error [%] =     3.17407e-09
scaled partial piv.       - |A*x-b| =    1.11022e-16 - rel.error [%] =     1.48292e-08

n=7, cond(hilb(7))=4.75367e+08

no piv.                   - |A*x-b| =    1.11022e-16 - rel.error [%] =     1.82942e-07
partial piv. (swap)       - |A*x-b| =              0 - rel.error [%] =     5.92343e-08
partial piv. (indirect)   - |A*x-b| =              0 - rel.error [%] =     5.92342e-08
scaled partial piv.       - |A*x-b| =    5.76889e-16 - rel.error [%] =     3.18237e-08

n=11, cond(hilb(11))=5.21823e+14

no piv.                   - |A*x-b| =    2.71948e-16 - rel.error [%] =        0.280551
partial piv. (swap)       - |A*x-b| =    2.48253e-16 - rel.error [%] =        0.085787
partial piv. (indirect)   - |A*x-b| =    6.66134e-16 - rel.error [%] =        0.085787
scaled partial piv.       - |A*x-b| =    6.18146e-16 - rel.error [%] =        0.374256

n=12, cond(hilb(12))=1.6523e+16

no piv.                   - |A*x-b| =    2.71948e-16 - rel.error [%] =         4.05758
partial piv. (swap)       - |A*x-b| =    2.48253e-16 - rel.error [%] =         0.71061
partial piv. (indirect)   - |A*x-b| =    1.11022e-16 - rel.error [%] =         0.71061
scaled partial piv.       - |A*x-b| =    4.71028e-16 - rel.error [%] =         3.27588

n=13, cond(hilb(13))=6.37569e+17

no piv.                   - |A*x-b| =    2.22045e-16 - rel.error [%] =         334.283
partial piv. (swap)       - |A*x-b| =    1.57009e-16 - rel.error [%] =         101.336
partial piv. (indirect)   - |A*x-b| =    4.71028e-16 - rel.error [%] =         101.336
scaled partial piv.       - |A*x-b| =    4.71028e-16 - rel.error [%] =         264.157
```

(Note: `|*|` in the output dump refers to the 2-norm $|\boldsymbol{a}| = (\sum_i a_i^2)^{1/2}$. `rel.error` refers to the relative error in per cent, i.e. $100 \cdot |\boldsymbol{x} - \boldsymbol{x}_{\text{true}}|/|\boldsymbol{x}_{\text{true}}|$.)

It can be seen that partial pivoting yields a substantial decrease in the error of the final solution. Scaled partial pivoting seems conterproductive in our case–the values in our initial matrix are all of fairly comparable magnitude, so that improved pivoting does not help much.

Also observe that the residual $A\,\boldsymbol{x} - \boldsymbol{b}$ remains roughly the same (namely, machine zero) for all cases, while the solution varies wildly.

Here is the code corresponding to the above output:

```
function p3
  values = [5,6,7,11,12,13];

  for n=1:length(values)
    run_all_tests(values(n));
  end
end
% ------------------------------------------------------------
function run_all_tests(n)
  disp('')

  % The matrix in the problem is called a 'Hilbert matrix'.
  % Depending on the size of n, it has an exponentially big
  % condition number, which means we should expect increasingly
  % bad solutions.

  fprintf('n=%d, cond(hilb(%d))=%g\n', n, n, cond(hilb(n)))
  disp('')
  gauss_test(n, @gauss_vanilla, 'no piv.');
  gauss_test(n, @gauss_partial_direct, 'partial piv. (swap)');
  gauss_test(n, @gauss_partial_indirect, 'partial piv. (indirect)');
  gauss_test(n, @gauss_scaled_partial, 'scaled partial piv.');
  %gauss_test(n, @backslash, 'matlab-backslash');
end
% ------------------------------------------------------------
function sol = gauss_test(n, linsolve, description)

  mat = hilb(n);

  %mat = [5,7,9;1,2,4;9,3,19];
  %n = 3;

  real_sol = ones(n,1);
  rhs = mat*real_sol;
  sol = linsolve(mat, rhs);

  residual = mat*sol - rhs;
  err = sol-real_sol;
  fprintf('%-25s - |A*x-b| = %15g - rel.error [%%] = %15g\n', ...
    description,...
    sqrt(residual'*residual), ...
    100 * sqrt(err'*err)/sqrt(real_sol'*real_sol))
end
% ------------------------------------------------------------
```

```
function sol = backslash(A, b)
  sol = A \ b;
end
% ------------------------------------------------------------
function sol = gauss_vanilla(A, b)
  [n, dummy] = size(A);

  for step = 1:n-1
    pivot = A(step, step);
    for i = step+1:n
      factor = A(i,step)/pivot;
      A(i,step:n) = A(i,step:n) - factor*A(step,step:n);
      b(i) = b(i) - factor*b(step);
    end
  end

  sol = b;
  for step = 0:n-1
    row = n-step;
    sol(row) = sol(row) - A(row,(row+1):n)*sol(row+1:n);
    sol(row) = sol(row) / A(row,row);
  end
end
% ------------------------------------------------------------
function sol = gauss_partial_direct(A, b)
  % A version of Gauss elimination with partial pivoting
  % using direct row swapping.

  [n, dummy] = size(A);

  for step = 1:n-1
    % find argmax_{step <= piv_row <= n} |a(piv_row, step)|
    max_row = 0;
    max_value = 0;

    for piv_row = step:n
      if (abs(A(piv_row, step))>max_value)
        max_value = abs(A(piv_row, step));
        max_row = piv_row;
      end
    end

    % swap rows
    if (step ~= max_row)
      temp = A(max_row, step:n);
      A(max_row, step:n) = A(step, step:n);
      A(step, step:n) = temp;

      temp = b(max_row);
      b(max_row) = b(step);
      b(step) = temp;
    end

    pivot = A(step,step);
```

```
    for i = step+1:n
      factor = A(i,step)/pivot;
      A(i,step:n) = A(i,step:n) - factor*A(step,step:n);
      b(i) = b(i) - factor*b(step);
    end
  end

  sol = b;
  for step = 0:n-1
    row = n-step;
    sol(row) = sol(row) - A(row,(row+1):n)*sol(row+1:n);
    sol(row) = sol(row) / A(row,row);
  end
end
% ------------------------------------------------------------------
function sol = gauss_partial_indirect(A, b)
  % A version of Gauss elimination with partial pivoting
  % using indirect access.

  [n, dummy] = size(A);

  permut = 1:n;

  for step = 1:n-1
    % find argmax_{step <= piv_row <= n} |a(piv_row, step)|
    max_row = 0;
    max_value = 0;

    for piv_row = step:n
      if (abs(A(permut(piv_row), step))>max_value)
        max_value = abs(A(permut(piv_row), step));
        max_row = piv_row;
      end
    end

    % save values for postcond. check
    prev_idx = permut(step);
    target_idx = permut(max_row);

    % swap out rows in permutation
    if (max_row ~= step)
      temp = permut(max_row);
      permut(max_row) = permut(step);
      permut(step) = temp;
    end

    pms = permut(step);
    pivot = A(pms, step);

    % verify postconditions
    assert(pms == target_idx)
    assert(permut(max_row) == prev_idx)

    % subtract out pivot row
```

```
    for i = step+1:n
      pmi = permut(i);
      factor = A(pmi,step)/pivot;
      A(pmi,step:n) = A(pmi, step:n) - factor*A(pms, step:n);
      b(pmi) = b(pmi) - factor*b(pms);
    end
  end

  % this prints a triangular matrix, if enabled:
  if (0)
    for i=1:n
      A(permut(i),1:n)
    end
  end

  % perform backsubstitution
  sol = zeros(n,1);
  for step = 0:n-1
    row = n-step;
    prow = permut(row);

    sol(row) = b(prow);

    for j = row+1:n
      sol(row) = sol(row) - A(prow,j)*sol(j);
    end
    sol(row) = sol(row) / A(prow,row);
  end
end
% ------------------------------------------------------------------
function sol = gauss_scaled_partial(A, b)
  % A version of Gauss elimination with scaled partial pivoting.

  [n, dummy] = size(A);

  permut = 1:n;

  % build row maxima table
  rowmaxes = zeros(n,1);
  for i=1:n
    rowmaxes(i) = max(abs(A(i,1:n)));
  end

  for step = 1:n-1
    % find argmax_{step <= piv_row <= n} |a(piv_row, step)|/rowmaxes(piv_row)
    max_row = 0;
    max_value = 0;

    for piv_row = step:n
      value = abs(A(permut(piv_row),step))/rowmaxes(permut(piv_row));
      if (value>max_value)
        max_value = value;
        max_row = piv_row;
      end
```

```
    end

    % save values for postcond. check
    prev_idx = permut(step);
    target_idx = permut(max_row);

    % swap out rows in permutation
    if (max_row ~= step)
      temp = permut(max_row);
      permut(max_row) = permut(step);
      permut(step) = temp;
    end

    pms = permut(step);
    pivot = A(pms, step);

    % verify postconditions
    assert(pms == target_idx)
    assert(permut(max_row) == prev_idx)

    % subtract out pivot row
    for i = step+1:n
      pmi = permut(i);
      factor = A(pmi,step)/pivot;
      A(pmi,step:n) = A(pmi, step:n) - factor*A(pms, step:n);
      b(pmi) = b(pmi) - factor*b(pms);
    end
  end

  % this prints a triangular matrix, if enabled:
  if (0)
    for i=1:n
      A(permut(i),1:n)
    end
  end

  % perform backsubstitution
  sol = zeros(n,1);
  for step = 0:n-1
    row = n-step;
    prow = permut(row);

    sol(row) = b(prow);

    for j = row+1:n
      sol(row) = sol(row) - A(prow,j)*sol(j);
    end
    sol(row) = sol(row) / A(prow,row);
  end
end
% --------------------------------------------------------------
function assert(m)
  if (m == 0)
    error('assertion failed')
```

```
  end
end
```

4. Problem 20, p. 170

Here are the results for this problem:

```
no piv.                   - |A*x-b| =              0 - |x-x_true| =         80.0603
partial piv. (swap)       - |A*x-b| =     3.63842e-12 - |x-x_true| =         6.54865
partial piv. (indirect)   - |A*x-b| =      3.0003e-11 - |x-x_true| =         6.54865
scaled partial piv.       - |A*x-b| =      3.0003e-11 - |x-x_true| =         6.54865
```

Before we begin discussing this problem, note that the matrix is actually singular ($\det A = 0$). Gauss elimination in exact arithmetic would thus fail, but we are getting (bad) results. What is to be learned here is that you can't trust Gauss elimination to detect singular matrices. In particular, observe that our residual $A\,\boldsymbol{x} - \boldsymbol{b}$ is fairly small, so if we didn't know the exact solution, we would not suspect anything to be wrong.

We observe that the partial pivoting methods fare significantly better than the non-pivoting ones, however scaled partial pivoting does not really have a huge impact. This stems from the fact that while partial pivoting favors clearly favors "large" rows, scaled partial pivoting does not particularly favor any of the rows in this matrix (their $a_{i,i}/\max_{i\leqslant j\leqslant n}\ |a_{i,j}|$ rations are roughly the same)–thus choosing rows almost "at random".

The code can be found on the solutions web page; it is mostly a repeat of the last problem.

5. Problem 7, p. 202

a) Suppose we find values $u_{i,j}$ and $l_{i,j}$ such that

$$\begin{pmatrix} l_{1,1} & \\ l_{2,1} & l_{2,2} \end{pmatrix}\begin{pmatrix} u_{1,1} & u_{1,2} \\ & u_{2,2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

Then

$$\begin{aligned} l_{1,1}u_{1,1} &= 0 \Rightarrow l_{1,1} = 0 \vee u_{1,1} = 0, \\ l_{1,1}u_{1,2} &= 1 \Rightarrow l_{1,1} \neq 0 \Rightarrow u_{1,1} = 0, \\ l_{2,1}u_{1,1} &= 1 \Rightarrow u_{1,1} \neq 0, \end{aligned}$$

which is a contradiction.

b)

We need to find values $u_{i,j}$ and $l_{i,j}$ such that

$$\begin{pmatrix} l_{1,1} & \\ l_{2,1} & l_{2,2} \end{pmatrix}\begin{pmatrix} u_{1,1} & u_{1,2} \\ & u_{2,2} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}}_{A:=}$$

Oddly enough, A is already upper triangular so that if we put $L = I$ (the identity matrix) and $U = A$, we've found an LU decomposition.

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

is the other solution.

6. Problem 10, p. 203

a) We will try to find values $u_{i,j}$ and $l_{i,j}$ such that

$$\begin{pmatrix} 1 & & \\ l_{2,1} & 1 & \\ l_{3,1} & l_{3,2} & 1 \end{pmatrix}\begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ & u_{2,2} & u_{2,3} \\ & & u_{3,3} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 & 2 \\ -1 & 0 & 2 \\ 3 & 2 & -1 \end{pmatrix}}_{A:=}.$$

By direct factoring (which I don't write out here):

$$\begin{pmatrix} 1 & & \\ -1 & 1 & \\ 3 & -1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 2 \\ & 1 & 4 \\ & & -3 \end{pmatrix} = A.$$

b) Observe that

$$\begin{pmatrix} 1 & & \\ & 1 & \\ & & -3 \end{pmatrix}\begin{pmatrix} 1 & 1 & 2 \\ & 1 & 4 \\ & & 1 \end{pmatrix}=\begin{pmatrix} 1 & 1 & 2 \\ & 1 & 4 \\ & & -3 \end{pmatrix}.$$

Thus

$$\begin{pmatrix} 1 & & \\ -1 & 1 & \\ 3 & -1 & 1 \end{pmatrix}\begin{pmatrix} 1 & & \\ & 1 & \\ & & -3 \end{pmatrix}\begin{pmatrix} 1 & 1 & 2 \\ & 1 & 4 \\ & & 1 \end{pmatrix} = A.$$

c) Finally, observe that

$$\begin{pmatrix} 1 & & \\ -1 & 1 & \\ 3 & -1 & 1 \end{pmatrix}\begin{pmatrix} 1 & & \\ & 1 & \\ & & -3 \end{pmatrix}=\begin{pmatrix} 1 & & \\ -1 & 1 & \\ 3 & -1 & -3 \end{pmatrix}.$$

Thus

$$\begin{pmatrix} 1 & & \\ -1 & 1 & \\ 3 & -1 & -3 \end{pmatrix}\begin{pmatrix} 1 & 1 & 2 \\ & 1 & 4 \\ & & 1 \end{pmatrix}=A.$$