Assignment II

Question 1

```
% Asn2, question 1: modified stopping condition to stop when the
% sum is changed by less than 2*{machine epsilon}.

format long g

MAXITER = 2e8;

disp('Single Precision');
sum0 = single(0);
EPSMACH=eps('single');

%disp('Double Precision');
%sum0 = 0;
%EPSMACH=eps;

sum1 = 42;  % just some dummy non-zero value

tic
n = 1;
while(1)
  sum1 = sum0 + (1/n^2);

  if (sum1 - sum0 <= 2*EPSMACH)
    disp(['Found solution, stopped after ' num2str(n) ' iterations'])
    break;
  end

  sum0 = sum1;
  if (n >= MAXITER)
    disp('maximum iterations exceeded, stopping');
    break;
  end
  n = n + 1;
end
toc

soln = sum0
% convert sum0 to double to do the error computation in double precision
AbsErr = abs(double(sum0) - (pi^2/6))
RelErr = abserr/(pi^2/6)

Results:
Double Precision
Found solution, stopped after 42443373 iterations
Solution =
          1.64493404344305
```

```
AbsErr =
      2.34051791281331e-08
RelErr =
      1.42286427157403e-08

Single Precision
Found solution, stopped after 2048 iterations
Solution =
        1.644446
AbsErr =
      0.000488409118123867
RelErr =
      0.000296917139700123
```

Question 2

a.

$$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 0.3333 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 0.8333 \end{bmatrix} \tag{1}$$

Using Gaussian elimination method we obtain following matrix by performing $E2 = E2 - 0.5E1$:

$$\left[\begin{array}{cc|c} 1 & 0.5 & 1.5 \\ 0 & 0.0833 & 0.0833 \end{array}\right] \tag{2}$$

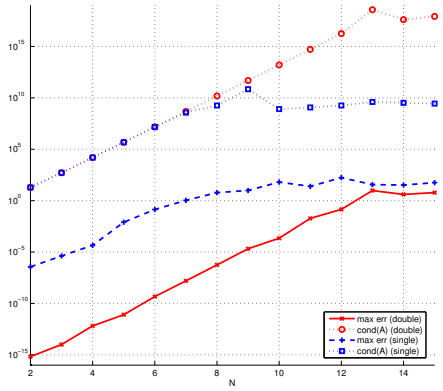
With backward substitution we find the result $[x_1\ x_2]^T = [1\ 1]^T$.
The difficulties are:

- Representation of infinite numbers like $\frac{1}{3}$.
- Division by small numbers during backward substitution.
- Subtraction of two small numbers.

b.

As n increases in both single and double precision, the error increases too. The matrix is becoming closer and closer to singular (note that the determinant is approaching zero). The condition number is also increasing which indicates the results will have larger and larger error as n increases.
Assuming that an error larger and 0.1 is unacceptable then the single precision code is only useful for $n \leq 5$ and the double precision code is useful for $n \leq 11$.

n	Single precision			Double precision		
	max err	det	cond	max err	det	cond
2	3.5763e-07	0.083333	19.281	6.6613e-16	0.083333	19.281
3	4.1723e-06	0.00046296	524.05	9.992e-15	0.00046296	524.06
4	4.4763e-05	1.6534e-07	15513	6.4904e-13	1.6534e-07	15514
5	0.007958	3.7502e-12	4.7604e+05	7.9992e-12	3.7493e-12	4.7661e+05
6	0.13628	5.431e-18	1.4458e+07	4.6507e-10	5.3673e-18	1.4951e+07
7	1.0554	6.7144e-25	3.6697e+08	1.5714e-08	4.8358e-25	4.7537e+08
8	6.0459	1.9957e-32	1.8029e+09	5.6506e-07	2.7371e-33	1.5258e+10
9	9.7596	-1.5216e-39	6.9346e+10	2.0868e-05	9.725e-43	4.9315e+11
10	63.734	0	8.0403e+08	0.00022082	0	1.6025e+13
11	24.317	0	1.173e+09	0.018574	0	5.2239e+14
12	172.86	0	1.8111e+09	0.1416	0	1.7945e+16
13	36.165	0	3.9871e+09	9.681	0	3.7544e+18
14	33.16	0	3.2863e+09	4.0209	0	4.0746e+17
15	57.031	0	2.7611e+09	6.0638	0	8.488e+17




```
% Asn2, Question 2: scaling partial pivoting GE of Hilbert matrix
% following the algorithm from the text

%n=15;

USE_SINGLE = true;


if (USE_SINGLE)
  A = single(hilb(n)); % single prec
else
  A = hilb(n);         % double prec
end

b = sum(A,2);
% augmented matrix A|b
AA = [A b];

% allocate some space
NROW = zeros(1,n);
if (USE_SINGLE)
  s = single(zeros(1,n));
  m = single(zeros(n,n));
  x = single(zeros(n,1));
else
  s = zeros(1,n);
  m = zeros(n,n);
  x = zeros(n,1);
end


% step 1
for i=1:n
  s(i) = max( abs(AA(i,1:n)) );  % note hidden loop here
  if (s(i) == 0)
    error('no unique soln exists');
  end
```

```
  NROW(i) = i;
end

% step 2: repeat steps 3 to 6
for i=1:(n-1)
  % step 3, find p
  max0 = -100;
  for j=i:n
    max1 = abs( AA(NROW(j),i) ) / s(NROW(j));
    if ( max1 > max0 )   % cannot be >=
      p = j;
      max0 = max1;
    end
  end

  % step 4
  if ( AA(NROW(p),i) == 0 )
    error('no unique soln exists');
  end

  % step 5: simulated row swap
  if ( NROW(i) ~= NROW(p) )
    NCOPY = NROW(i);
    NROW(i) = NROW(p);
    NROW(p) = NCOPY;
  end

  % step 6
  for j=(i+1):n
    % step 7
    m(NROW(j),i) = AA(NROW(j),i)/AA(NROW(i),i);
    % step 8 (note hidden for loop)
    AA(NROW(j),:) = AA(NROW(j),:) - m(NROW(j),i)*AA(NROW(i),:);
  end

end  % step 2

% step 9
if ( AA(NROW(n),n) == 0 )
  error('no unique soln exists');
end


% step 10: backsub
x(n) = AA(NROW(n),n+1) / AA(NROW(n),n);
% step 11: backsub
for i=(n-1):-1:1
  sum0 = 0;
  for j=(i+1):n
    sum0 = sum0 + AA(NROW(i),j)*x(j);
  end
```

```
  x(i) = ( AA(NROW(i),n+1) - sum0 ) / AA(NROW(i),i);
end
```