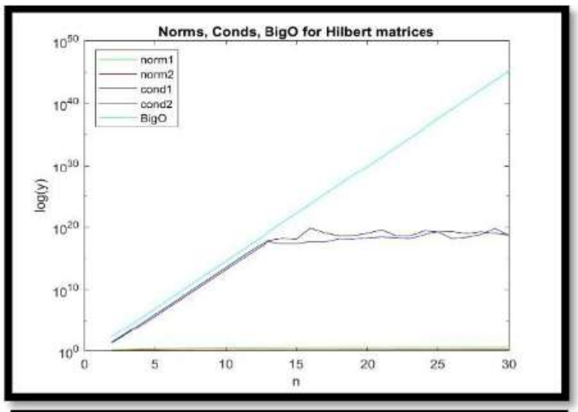
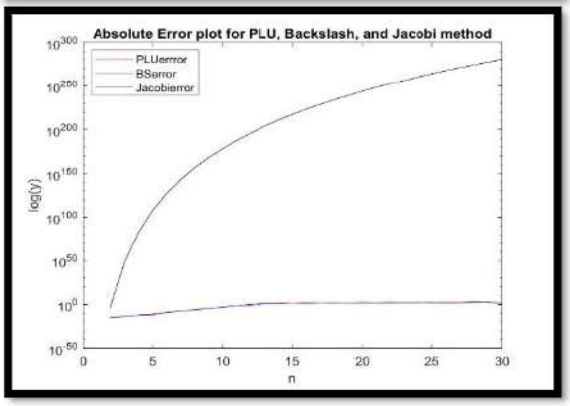
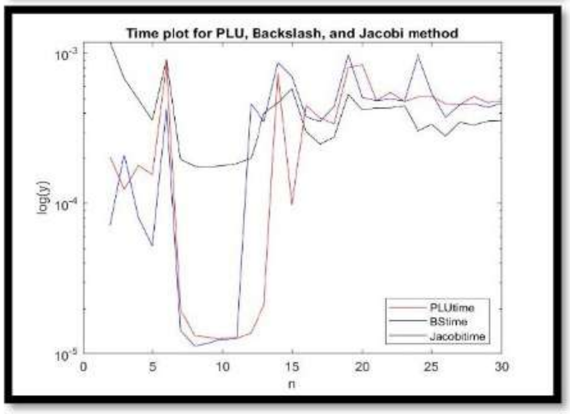
COMPUTING ASSIGNMENT 4

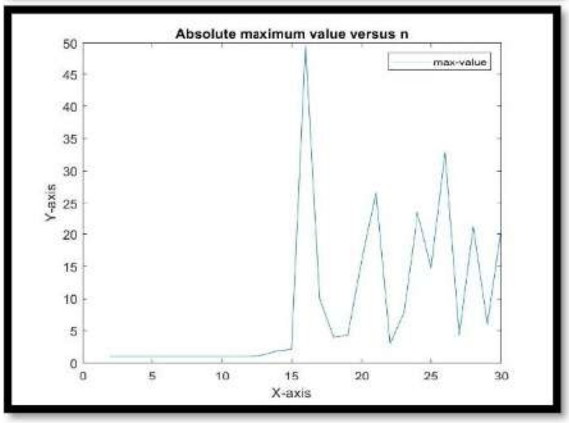
NAVJOT KAUR (301404765)

This assignment concerns with the fact that because Hilbert matrix is an ill-conditioned matrix, it is difficult to compute its solution matrix i.e, $x = (x_1, x_2, x_3, \dots, x_n)^T$ accurately in floating point arithmetic.









In part a and b, the 1-norm, 2-norm, their condition numbers, and the growth of these condition numbers are calculated for Hilbert matrices from n = [2,30]. The graph shown here shows how these norms and condition numbers varies when plotted against n. It can be clearly seen that both the norms, 1-norm and 2-norm, grow steady as expected even for larger n, however, the condition numbers cond1 and cond2 grow very fast when n is larger and Hilbert matrices get closer to singular so, MATLAB approximates the condition numbers, which is the reason for the fluctuations in the graph as n grows after n = 13. In part b, the BigO function $O\frac{(1+\sqrt{2})^{4n}}{\sqrt{n}}$ (to calculate condition numbers theoretically) represents the upper bound of the condition numbers which can be seen from the graph as the growth plot (cyan color line) is above the condition numbers plot. This rate of growth is consistent with the results as it varies linearly and is above the condition numbers plot as they grow fast when n is larger. The graph used here is semilogy (taken log(y) at y axis) because x only varies between 2 and 30 however, y grows very quickly to very large numbers as n grows that it is so hard to compare the condition numbers, and BigO grows exponentially and appear linear.

In part c, the absolute solution errors are calculated for three methods namely, PLU, Backslash, and Jacobi method. It can be observed from the graph that PLU and Backslash have fairly similar errors (overlap) because they use the similar method for calculation. The errors in PLU and Backslash are very large but so small as compared to Jacobi method. The iterations used for Jacobi method in the MATLAB code are 200 because if we take iterations larger than 219, then some matrix entries are Inf and/or Nan, which cannot be used for calculation and plotting. This is because Hilbert matrix is ill-conditioned and in Jacobi method, we use diagonal entries to calculate next iterations. The time plot, calculated by tic-toc, for these three method is shown is graph where it can be seen that the performance for Jacobi method is fairly constant because of the limited number of iterations used, whereas in PLU and backslash as discussed they use similar methods the performance and cost is also approximately similar which fluctuates very fast when n grows larger because of the ill-conditioned nature of Hilbert matrix.

In part d, the matrix product M is calculated which is the product of Hilbert matrix and its inverse, of course it should be equal to Identity matrix I, but here it is not because as n grows, the maximum value in each subsequent matrix grows, which in turn increases the difference between identity and M. It can be noted that for n<=10, M is indeed equal to I because diagonal entries are 1 (ignoring -0 values). As can be seen from the graph that as n > 10, the max value fluctuates, the reason behind this is the use of inv function of MATLAB for a matrix that is ill-conditioned i.e, very close to singular, increasing the errors between them so $M \neq I$.

MATLAB CODE

```
clear all
format long
%Part (a): Set up Hilbert matrices and plot its norm and cond
nlist = [2:30]; % matrix sizes
cond2H = []; cond1H = []; norm1 = []; norm2 = [];
for n = nlist,
  Hn = hilb(n);
  norm1 = [norm1, norm(Hn,1)];
  norm2 = [norm2, norm(Hn,2)];
  cond1H = [cond1H, cond(Hn,1)];
  cond2H = [cond2H, cond(Hn,2)];
end
semilogy(nlist, norm1, 'g'); legend('norm1'); hold on;
semilogy(nlist, norm2, 'r', 'DisplayName', 'norm2');
semilogy(nlist, cond1H, 'k', 'DisplayName', 'cond1');
semilogy(nlist, cond2H, 'b', 'DisplayName', 'cond2');
title("Norms, Conds, BigO for Hilbert matrices");
xlabel("n"); ylabel("log(y)");
% Part (b):
growth = ((1+sqrt(2)).^(4*nlist))./(sqrt(n));
semilogy(nlist, growth, 'c','DisplayName', 'BigO');
hold off;
%Part (c):
PLUerror = []; % list of 2-norm errors for PLU
BackSlashError = []; %list of 2-norm errors for backslash
JacobiError = []; %list of 2-norm errors for Jacobi
PLUtime = []; BStime = []; Jactime = [];
klist = [2 : 30]; % min/max matrix size
for k = klist,
  Hk = hilb(k);  % generate Hilbert matrix
  x = ones(k,1); % exact solution (all 1's)
  b = Hk * x;    % compute RHS for this x
  tic                 % Compute the solution using 3 different methods:
  [L, U, P] = lu(Hk);
  xPLU = U \ (L \ (P*b)); % solution from PLU-decomposition
  PLUtime = [PLUtime, toc];
  tic
  xBackSlash = Hk \ b; % solution from backslash
  BStime = [BStime, toc];
  tic
  x0 = rand(k,1);
  xJacobi = jacobi2( Hk, b, x0, 1e-4, 200); %solution using Jacobi
  Jactime = [Jactime, toc];
  PLUerror = [PLUerror, norm(xPLU-x,2)];
  BackSlashError = [BackSlashError, norm(xBackSlash-x,2)];
  JacobiError = [JacobiError, norm(xJacobi-x, 2)];
end
%Plot solution errors
semilogy(klist, PLUerror, 'm', 'DisplayName', 'PLUerror');
title("Absolute Error plot for PLU, Backslash, and Jacobi method");
xlabel("n"); ylabel("log(y)");
legend('PLUerrror'); hold on;
semilogy(klist, BackSlashError, 'b', 'DisplayName', 'BSerror');
semilogy(klist, JacobiError, 'k', 'DisplayName', 'Jacobierror');
hold off;
semilogy(klist, PLUtime, 'r', 'DisplayName', 'PLUtime');
hold on; legend('PLUtime');
semilogy(klist, BStime, 'b', 'DisplayName', 'BStime');
semilogy(klist, Jactime, 'k', 'DisplayName', 'Jacobitime');
title("Time plot for PLU, Backslash, and Jacobi method");
xlabel("n"); ylabel("log(y)"); hold off;
%Part (d):
mlist = [2:30]; max_value = [];
for m=mlist,
    Hm = hilb(m);
    inverseH = inv(Hm);
    M = Hm * inverseH;
    max_value_col = max(abs(M));
    max_value = [max_value, max(max_value_col)];
end
plot(mlist, max_value); legend("max-value");
title("Absolute maximum value versus n");
xlabel("X-axis"); ylabel("Y-axis");
```