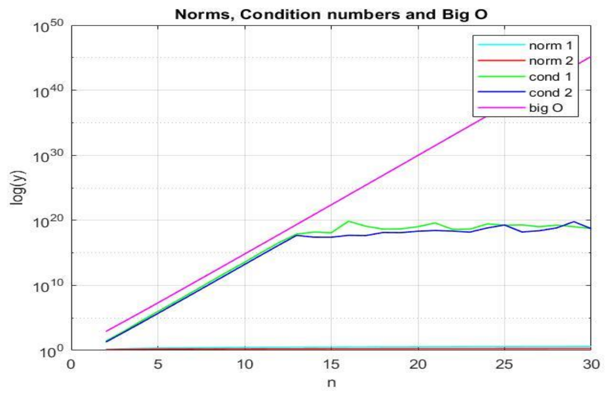
RITIKA GOYAL (301401516)

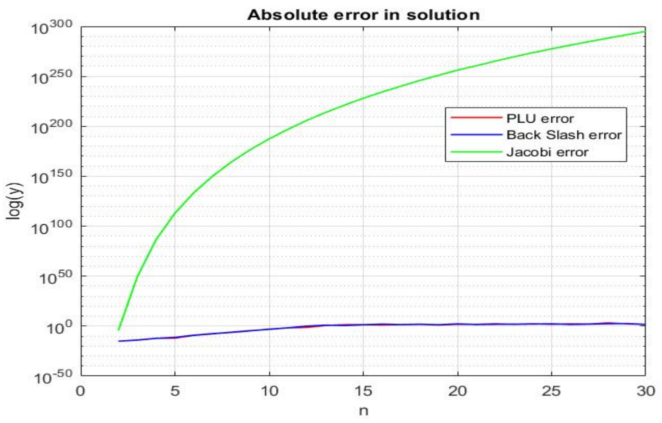
MACM316 CA4

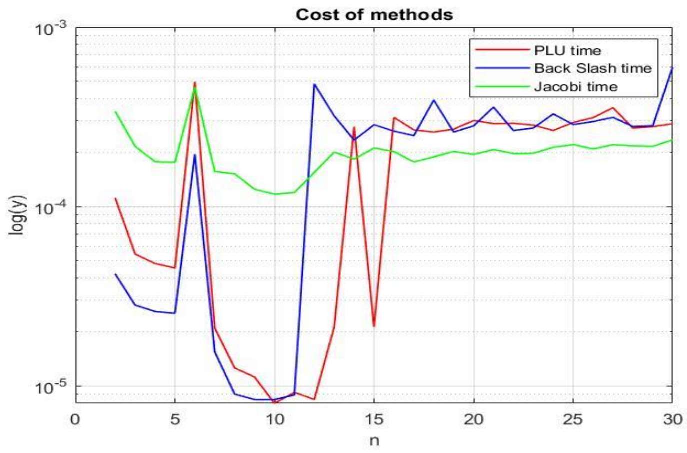
The goal of this assignment is to show that the solution is not easy to compute accurately in floating point arithmetic due to ill conditioned Hilbert matrix even though it is invertible. The norms and condition numbers are displayed on the graph by taking dimension [2:30] and then three different matrix inversion methods are used to discuss the errors in solution.

Part a, b) The given plot shows that how the norm and condition number of the Hilbert Matrix grow with n = [2:30] and the theoretically calculated condition number is also shown in the plot. The 1-norm, 2-norm are used to display norms and cond-1, cond-2 is used to display condition numbers. The semiology is used as axis-scale for the plot. When the theoretical condition number ($\frac{(1+\sqrt{2})^{4n}}{\sqrt{n}}$) is plotted, it is perfectly linear (magenta color in plot) which justifies the choice of axis scaling because condition number grows fast and choosing log-scale for y-axis makes the graph linear. The norm-1 and norm-2 just have slight variation and approximately overlap but cond-1 and cond-2 have slight variation until n = 13 but fluctuates due to closeness to singular nature of Hilbert matrix when n is large and is approximated by MATLAB.

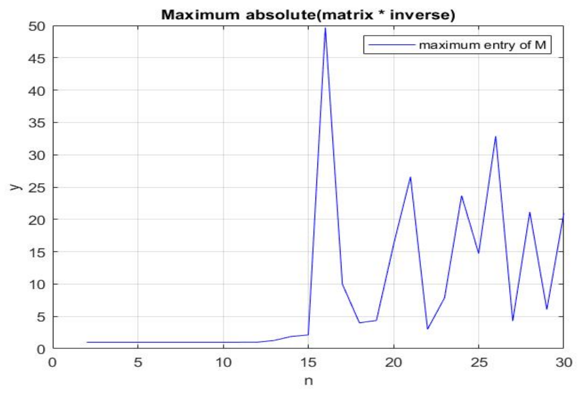


Part c) The absolute solution error in 2-norm is calculated using three different matrix inversion methods – PLU decomposition, Matlab's backslash command and Jacobi iteration whose plot is shown below. Jacobi iterative method works fine with well-conditioned linear system, but when the matrix is ill-conditioned, due to Jacobi's limitation, it gives larger error as we use diagonal entries in Jacobi. The iterations used while computing Jacobi's error is 210 because with larger iteration, the matrices entries are Inf and/or Nan. However, in PLU and backslash command, we get approximately equal absolute error which is high but relatively low that Jacobi's. Further, the cost of each error is calculated by using tic-toc which displays the machine time used to complete the instruction whose graph is displayed below. The cost is approximately similar but fluctuates very fast when n grows larger due to more singular nature (ill-conditioned) of Hilbert matrix.





Part d) The inv function is used to calculate inverse of Hilbert matrix and when the inverse is multiplied by Hilbert matrix, the result should be Identity but it is not because with larger n, the maximum value in resulted matrix is not 1, it keeps increasing. The maximum value fluctuates when n is large because of ill conditioned (close to singular) nature of Hilbert matrix which results in large errors in inverse by using 'inv', however with smaller n (n<=10), it is identity which is displayed in the graph on right.



RITIKA GOYAL (301401516)

MATLAB CODE

```
format long
% Part (a): Set up Hilbert matrices and plot its norm and condition number.
nlist = [2:30]; cond2H = []; norm1H =[]; norm2H =[]; cond1H =[];
for n = nlist
  Hn = hilb(n);
  cond2H = [cond2H, cond(Hn,2)];
  norm1H = [norm1H, norm(Hn,1)];
  norm2H = [norm2H,norm(Hn,2)];
  cond1H = [cond1H, cond(Hn,1)];
end
% ADD CODE TO PLOT CONDITION NUMBERS AND NORMS VERSUS n
semilogy(nlist,norm1H,'c','LineWidth',1); legend('norm 1');hold on;
semilogy(nlist,norm2H,'r','DisplayName','norm 2','LineWidth',1);
semilogy(nlist,cond1H,'g','DisplayName','cond 1','LineWidth',1);
semilogy(nlist,cond2H,'b','DisplayName','cond 2','LineWidth',1);
xlabel("n");ylabel("log(y)");title("Norms, Condition numbers and Big O");grid on; hold off;
% Part (b): ADD THE THEORETICALLY EXPECTED GROWTH TO YOUR PLOT
z = (((1+ sqrt(2)).^(4.*nlist))./(sqrt(nlist)));
hold on ; semilogy(nlist,z,'m','LineWidth',1,'DisplayName','big O'); hold off;
% Part (c): Compare performance for the ill-conditioned linear system using:
PLUerror = []; bs_error = []; J_error = []; PLU_time = []; bs_time = []; J_time = [];
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
  % Compute the solution using 3 different methods:
  tic
  [L, U, P] = lu(Hk);
  xPLU = U \ (L \ (P*b)); % solution from PLU-decomposition
  PLU_time = [PLU_time,toc];
  tic
  xBackslash = Hk \ b;
  bs_time = [bs_time, toc];
  tic
  [xJacobi,niter] = jacobi2(Hk,b,rand(k,1),1e-4,210);
  J_time = [J_time, toc];
  PLUerror = [PLUerror, norm(xPLU-x,2)];
  bs_error =[ bs_error, norm(xBackslash - x,2)];
  J_error = [J_error , norm(xJacobi - x,2),];
end
% Plot solution errors ADD CURVES FOR '\' AND JACOBI TO THIS PLOT
semilogy(klist, PLUerror, 'r','LineWidth',1); legend ('PLU error');hold on;
semilogy(klist, bs_error, 'b','LineWidth',1,'DisplayName','Back Slash error');
semilogy(klist, J_error, 'g','LineWidth',1,'DisplayName','Jacobi error');
xlabel("n");ylabel("log(y)");title("Absolute error in solution");grid on;
hold off;
%plotting time
semilogy(klist, PLU_time, 'r','LineWidth',1);
legend ('PLU time');hold on;
semilogy(klist, bs_time, 'b','LineWidth',1,'DisplayName','Back Slash time');
semilogy(klist, J_time, 'g','LineWidth',1,'DisplayName','Jacobi time');
xlabel("n");ylabel("log(y)");title("Cost of methods");grid on;
hold off;
% Part (d):
max_entry = [];
for n = klist
    Hk = hilb(n);
    hInverse = inv(Hk);
    M = Hk * hInverse;
    M = abs(M);
    max_entry  =[ max_entry,max(max(M))];
end
plot(klist,max_entry,'b');
xlabel("n");ylabel("y");title("Maximum absolute(matrix * inverse)");grid on; legend ('maximum entry
of M');
```