Numerical Algorithm

Assignment 2

Mohammad Mahdi Fallah

Paper & Pencil:

1. Definitions :

$M_k = I - v_k e_k^T$

I = identity matrix

$v_k = [0,\dots, 0, m_{k-1}, m_n]^T$ the kth column of matrix M with 0 as entry for all m_k where $i <= k$.

$e_k =$ kth column of Identity matrix

$M_k^{-1} = I + v_k e_k^T$

a) $M_k M_k^{-1} = (I - v_k e_k^T)(I + v_k e_k^T) = I^2 + I v_k e_k^T - v_k e_k^T I - (v_k e_k^T)^2 =$
$I^2 - v_k e_k^T v_k e_k^T$

e_k^T is a 1*n vector with only 1 non zero entry at k

v_k is a n*1 vector with only zero entries before and at k

==> $e_k^T v_k = 0$ ==>$I^2 - v_k e_k^T v_k e_k^T = I^2 - v_k 0 e_k^T = I^2$ - zeros(n, n) = $I^2 = I$

$\mathbf{M_k M_k^{-1} = M_k^{-1} M_k = I}$

b) $M_k M_j = (I - v_k e_k^T)(I - v_j e_j^T) = I^2 + I v_k e_k^T - v_j e_j^T I - (v_k e_k^T)(v_j e_j^T)$

e_k^T is a 1*n vector with only 1 non zero entry at k

V_j is a n*1 vector with only zero entries before and at k

==> $e_k^T v_j = 0$ ==> $M_k M_j = I^2 + I v_k e_k^T - v_j e_j^T I =$ $\mathbf{I + v_k e_k^T - v_j e_j^T}$ which is their union.

2. Definitions:

cond(A) = inf ==> A is Singular

for any A and non-zero scalar λ: cond(λA) = cond(A)

cond(A) is a good measure of being close to singularity

$|kA| = k^n * |A|$, where n is the size of A

Now, let I be an identity matrix of size n and $k = 10^{-10}$, so:

$| I | = 1$ ==> cond(kI) = 1

But,

$|kI| = (10^{-10})^n * 1$

Which shows there exist a matrix which has a determinant close to 0, but with a condition number of 1 that means far from being singular

Programming: (octave 6 were used for this assignment)

Part I.

1. We know that cond(A) = $\| A \| \| A^{-1} \|$

But we want to avoid computing the inverse of A to get condition number. with solving a linear equation with matrix A and vectors z, y we get:

$$\mathbf{Az = y} \implies \mathbf{z = A^{-1}y} \implies \mathbf{\|z\| = \|A^{-1}y\| <= \|A^{-1}\| \ \|y\|} \implies \mathbf{\|z\| / \|y\| <= \|A^{-1}\|}$$

This means maximizing $\|z\| / \|y\|$ is the best way for estimation of $\|A^{-1}\|$

With the **LINPACK** algorithm we can solve the linear equation system, by getting the LU decomposition of matrix A, then :

solve $U^T w = e$

solve $L^T y = w$

solve $L v = y$

solve $U z = v$

estimated cond = $\| y \| / (\| A \| \| z \|)$

Notice that vector **e** must be defined as follow:

e only contains 1 and -1, and the sign should be chosen so that || w || will be maximized, for this there exist 2 possibilities for each e_k :

$e_k^+ = \text{sign}(t_k)$ $e_k^- = -e_k^+$ $w_k = (e_k - t_k)/U_{kk}$

which gives two possibilities for w_k:

$t_k^{-/+} = e_k^{-/+} - t_k$ $t_j = \sum_{i=1}^{k-1} U_{ij}\, w_i$ $t_j^{-/+} = t_j + U_{kj}\, w_k^{-/+}$

This way we can get sum of absolute values of $t_j^{-/+}$ and by comparing the two we can get the one which results in growth of w, which is our ultimate goal.

2. For implementation first I created est_cond() to solve the following part

$A = L * U$

solve $U^T w = e$

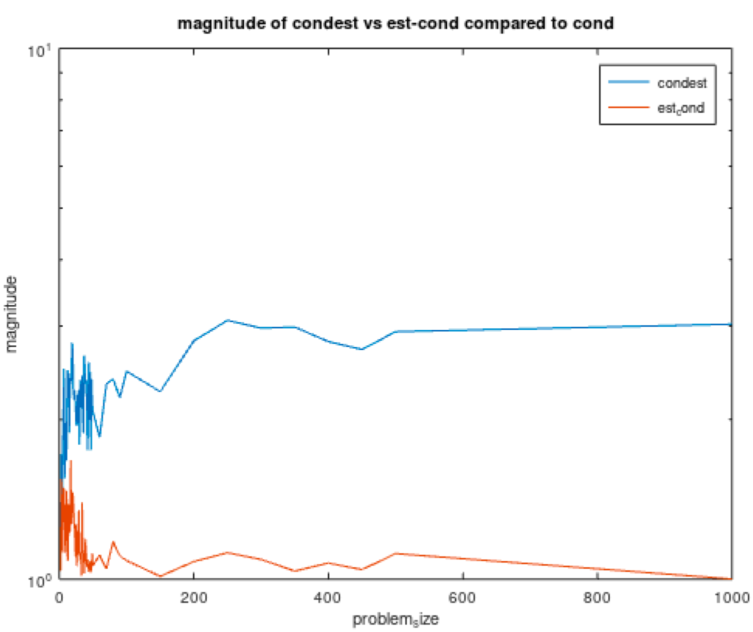
solve $L^T y = w$

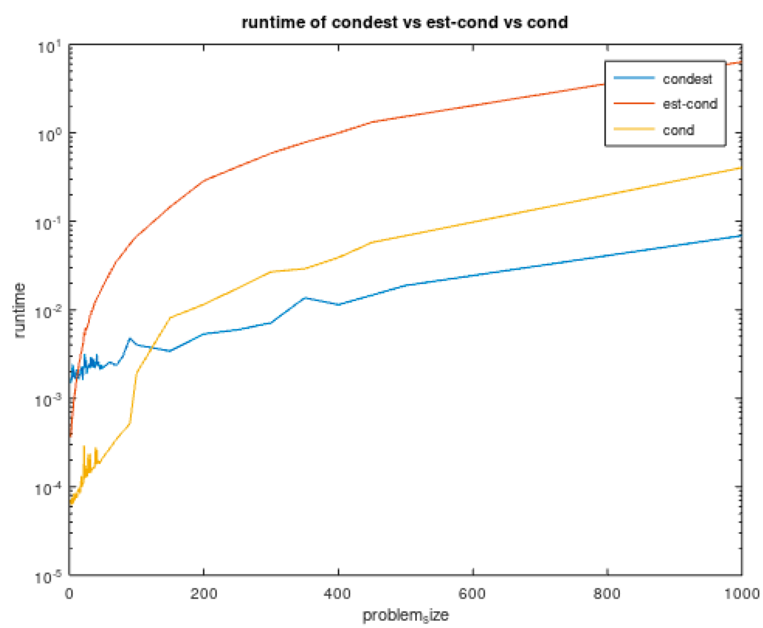
solve $L v = y$

solve $U z = v$

estimated cond = || y || / (|| A || || z ||)

Octave operation lu() and linsolve() was used for implementing lu decomposition and solving equations.
Then in solveU() I calculated w by maximizing
At the end in the assignment I generated random matrixes to compare the accuracy and runtime of implemented functions with the octave function estcond() in range of n = [2:50 60:10:100 150:50:500 1000]





Part II.

For this part I have computed left and right hand side of the bounds equation in range of n = [2:50 60:10:100 150:50:500 1000] as follows:

```
function [x, delta_x] = lhs_perturbation(A, E, b, delta_b)

  %  n × 1 vector x which is the solution to the linear system Ax = b
  x = linsolve(A, b);

  % n × 1 vector delta x which is the difference xˆ - x between x
  % and the solution xˆ to the perturbed linear system
  delta_x = linsolve(A, b + delta_b) - x;
endfunction
```

```
function [E, delta_b] = rhs_perturbation(n)

  % random n*n E with 1norm = 10^-8
  E = rand(n)* 10^-8;

  % random n*1 delta_b with 1norm = 10^-8
  delta_b = rand(n, 1)* 10^-8;

endfunction
```

and generated 50 random A and b for each problem size in order to get the average.

As you can see in the figure the bound clearly always hold.