CS6210 - Homework/Assignment-3

Arnab Das(u1014840)

October 4, 2016

Question-1: Chapter-4: Exercise-12

The condition number of an eigen value, λ , of a matrix A is defined as

$$s(\lambda) = \frac{1}{\mathbf{x}^T\mathbf{w}}$$

Referencing from example 4.7/4.6, let is define the two matrices as:
$A_1 = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$ and $A_2 = \begin{bmatrix} 4 & 1 \\ 0 & 4 \end{bmatrix}$

Both the matrices have eigen value of 4 with algebraic multiplicaity 2, that is both its eigen values are 4,4. Now first let us consider A_1. Its eigen vectors corresponding to eigen valuesm of 4 are $x_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $x_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Thus the geometric multiplicity is also 2. The left eigen vectors, w^T, will be $w_1^T = \begin{bmatrix} 1 & 0 \end{bmatrix}$ and $w_2^T = \begin{bmatrix} 0 & 1 \end{bmatrix}$.
Thus, for each of the above eigen vectors for A_1, the inner product $\frac{1}{\mathbf{x}^T\mathbf{w}}$ is 1, hence the condition number turns out to be,

$$S(\lambda = 4)_{A_1} = 1 \tag{1}$$

Let us consider A_2 for now. The eigen values for A_2 is 4 with algebraic multiplicity 2. However , it has only one right eigen vector, $x_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, and only one left eigen vector, $w_1^T = \begin{bmatrix} 0 & 1 \end{bmatrix}$.
Thus, for the above pair of left and right eigen vector for A_2, the inner product $\frac{1}{\mathbf{x}^T\mathbf{w}}$ is $\frac{1}{0} = \infty$. Hence the condition number turns out to be

$$S(\lambda = 4)_{A_2} = \infty \tag{2}$$

Condition numbers indicate how stable the computation is expected to be, such that lower condition numbers indicate more stability. If we refer to example-4.7, where the experiment was done with small perturbation to the matrix, A_1 came to be well-conditioned while A_2 came to be ill-conditioned. Our evaluation of the condition number also suggests that since condition number for A_1 is small it is numerically more stable and hence well conditioned while A_2 has condition number number of ∞ and hence ill-conditioned.

Question-2: Chapter-5: Exercise-2

For an $n \times n$ matrix A, and a vector vector b, the pseudoCode for the Gauss-Jordan elimination method for Solving $Ax = b$, is as described below(Assuming no pivoting):

(a) PseudoCode:

for k=1 : $n-1$
  for i=1 : n
    if (i $\neq$ k)
      $l_{i,k} = \frac{a_{i,k}}{a_{k,k}}$
      for $j = k+1 : n$
        $a_{i,j} = a_{i,j} - l_{i,k}a_{k,j}$
      $b_i = b_i - l_{i,k}b_k$

Since, it does the update for all rows except the row k, one **if** condition is introduced to check for $i \neq k$, and the row traversal instead of $k+1$ to n, has been increased as 1 to n.

(b) The cost of the Gauss-Jordan algorithm in terms of operation count(or flop count) is as follows:

$$\sum_{k=1}^{n-1} 2(n-1)(n-k) + 2(n-1) + (n-1)$$

$$\begin{aligned}
&= \sum_{k=1}^{n-1} 2(n-1)(n-k+1) + (n-1) \\
&= (n-1)\sum_{k=1}^{n-1} 2(n-k+1) + 1 \\
&= (n-1)\sum_{k=1}^{n-1} (2n-2k+3) \\
&= (n-1)\left(2n(n-1) - 2\frac{n(n-1)}{2} + 3(n-1)\right) \\
&= (n-1)^2\left(2n-n+3\right) \\
&= (n-1)^2(n+3) \\
&= n^3 - 2n^2 + n + 3n^2 - 6n + 3 \\
&= n^3 + O(n^2)
\end{aligned}$$

(Proved).

Question-3: Chapter-5: Exercise-3

Let A and T be two non-singular, $n \times n$ matrices. Furthermore, we are given two matrices, L and U such that L is unit lower triangular and U is upper triangular and the following relation holds:

$$TA = LU \tag{3}$$

The algorithm to find the solution for $Ax = b$ is detailed below

Algorithm: :
Step-1: Perform a matvec operation to evaluate **Tb**.
Step-2: Solve for y: Ly = Tb ; //By forward substitution
Step-3: Solve for x: Ux = y ; //By backward substituion

Explanation: To evaluate Ax=b, from the given conditions, we first perform a matvec of T and b which is $O(n^2)$, since it invloves a matrix-vector multiplication of $n \times n$ matrix T, and a $n \times 1$ vector b. In second step we solve for y using forward substitution since there is a lower triangular matrix. This step also involves $O(n^2)$ operations. The third step involves the evaluation of the final solution **x**, and since there is an upper triangular matrix, we use backward substitution which is again $O(n^2)$. Thus total flops required is in order of $O(n^2)$.

PseudoCode:

```
SolveForX (L, U, T, b, x)
    for i=1:n
        sum = 0
        for j=1:n
```

sum +=$T_{i,j}$ *b_j

G_i = sum

// G is the matvec result of Tb

// Solve for Ly = G = Tb by forward substitution

for k=1:n

$$y_k = \frac{G_k - \sum_{j=1}^{k-1} L_{k,j} y_j}{L_{k,k}}$$

// Solve for Ux = y by backward substitution

for k=n:1

$$x_k = \frac{y_k - \sum_{j=k+1}^{n} U_{k,j} x_j}{U_{k,k}}$$

Question-4: Chapter-5: Exercise-4

To find the inverse of a matrix, A , one of the simplest mechanism to do so in linear algebra is to augment the matrix A with a identity matrix of the same size. Then perform a sequence of operations such that the region of A is replaced by an identity matrix and in that case the region of the augmented matrix where the identity matrix was initiallly added, contains the inverse of the matrix, A.
Suppose , we are given an $n \times n$ matrix whose matrix we need to find. Suppose by performing k number of matrix operations, each operation represented as T_1, we can derive the identity matrix from A, which means we can write the following:
$T_1 T_2 T_3 \dots T_k A = I$
Multiply both sides by the inverse:
$T_1 T_2 T_3 \dots T_k A A^{-1} = I A^{-1}$
$T_1 T_2 T_3 \dots T_k I = A^{-1}$
which means the same sequence of operations, when applied on the diagonal matrix of the same size, will yeld the inverse of the matrix,A.

Once, we augment A with the diagonal matrix, the size of the overall matrix becomes $n \times 2n$. Let us call this matrix B. The pseudocde for obtaining inverse from B using GaussJordan is described below:

PseudoCode:

for k=1 : $n-1$

for i=1 : n

if (i $\neq$ k)

$$l_{i,k} = \frac{b_{i,k}}{b_{k,k}}$$

for $j = k+1 : 2n$

$b_{i,j} = b_{i,j} - l_{i,k} b_{k,j}$

diag_temp = $b_{k,k}$

for $j = k : 2n$

$$b_{k,j} = \frac{b_{k,j}}{diag\_temp}$$

The computation cost required for this operation in terms of floating point operations can be computed as follows:

$$\sum_{k=1}^{n-1} 2(2n-k)(n-1) + (n-1) + (2n-k+1)$$

$$\begin{aligned}
&= \sum_{k=1}^{n-1} (4n-2k)(n-1) + 3n - k \\
&= \sum_{k=1}^{n-1} 4n^2 - 4n - 2nk + 2k + 3n - k \\
&= \sum_{k=1}^{n-1} 4n^2 - n - 2nk + k \\
&= 4n^2(n-1) - n(n-1) - n^2(n-1) + \frac{n(n-1)}{2} \\
&= 3n^3 + O(n^2)
\end{aligned}$$

The same task performed using LU Decomposition requires $\frac{8}{3}n^3 + O(n^2)$ operation count. Thus, in the order of n^3, this method requires $\frac{1}{3}n^3$ more operation count.

Question-5: Chapter-5: Exercise-12

To reformat the cholesky algorithm from the naive one given with multiple for-loops, below is the pseducode for the vectorized one:

PseudoCode:

for k=1:n-1

$a_{k,k} = \sqrt{a_{k,k}}$

$A_{k+1:n,k} = \left(\frac{a_{k+1,k}}{a_{k,k}}, \frac{a_{k+2,k}}{a_{k,k}} \ldots, \frac{a_{n,k}}{a_{k,k}}\right)$

$A_{k+1:n,k+1:n} = A_{k+1:n,k+1:n}$ - $[A_{k+1:n,k+1:n}]$ * $[A_{k+1:n,k+1:n}]^T$

$A_{k,k+1:n} = 0$

$A_{n,n} = \sqrt{(A_{n,n})}$

The implemented code in Matlab is available as $Prob5\_Cholesky.m$ in the associated submission tar. Run in the matlab window as :
$Prob5\_Cholesky(n)$; // where n is the size of the matrix. It generates an $n \times n$ matrix and reports the result of the implemented cholesky and the matlab cholesky function to compare the output.

Question-6: Chapter-5: Exercise-20

Hessenberg Matrix: An $n \times n$ matrix A is said to be in Hessenberg or upper Hessenberg form if all its elements below the first subdiagonal are zero, such that:

$$a_{i,j} = 0, i > j + 1$$

(a) The psedoCode for LU decomposition of an upper Hessenberg Matrix,A, of size $n \times n$ is as follows:

PseudoCode:

for k = 1:n-1
$l_{k+1,k} = \frac{a_{k+1,k}}{a_{k,k}}$
for j=k+1:n
$a_{k+1,j} = a_{k+1,j} - l_{k+1,k} * a_{k+1,j}$

The upper traingular part of A will contain the U matrix formed due to the decomposition. We have not updated the zero values here.

(b) The sparsity structure of the matrix L will look as follows:

$$\mathrm{L} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ l_{2,1} & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & l_{3,1} & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & l_{4,1} & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \\ \dots & \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & 0 & 0 & \dots & l_{n,n-1} & 1 \end{bmatrix}$$

Thus, L contains one subdiagonal below the main diagonal.

(c) The cost of operation for the decomposition can be evaluated as:

$$\sum_{k=1}^{n-1} 2(n-k)+1$$
$$= 2n(n-1) - n(n-1) + (n-1)$$
$$= O(n^2)$$

To solve Ax = b or LU = b, where A=LU, we solve Ly=b first using forward substituion and then Ux=y using backward substituion. Since L is banded, the general formula for forward substituion in this case will become:
for k=1:n-1
$y_k = \frac{b_k - L_{k,k-1} * y_{k-1}}{L_{k,k}}$
The cost for forward substituion will be 3n $\approx$ O(n)

For the backward substituion for Ux=y, will still be $O(n^2)$ since the upper traingle remains dense and does not have a sparsity pattern.
Therefor, **total cost of operation** $= O(n^2) + O(n) + O(n^2) \approx O(n^2)$

(d) If we enable partial pivoting for the LU decomposition of A, then we get $\mathbf{PA} = L'U'$, where L' is the corresponding lower triangular matrix and U' is the corresponding upper matrix for PA. The sparcity structure of L' will be same as L. However, since we have applied partial pivoting, we need to recober the factors of A. The permutation matrix has the following property ,

$$PP^T = I$$
$$=> P^T = P^{-1}$$

Thus the folowing holds if we multiply $\mathbf{PA} = L'U'$ by P^T:

$$P^T PA = P^T L' U'$$
$$A = (P^T L')U'$$

Since L and L' has the same sparsity structure, the product $P^T L'$ will destroy the sparsity structure of L', however the number of zeros for every column will still remain the same.

Question-7: Chapter-5: Exercise-21

Given arrow matrices:

$$\text{A} = \begin{bmatrix} x & x & x & x & \dots & x \\ x & x & 0 & 0 & \dots & 0 \\ x & 0 & x & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x & 0 & 0 & \dots & x & 0 \\ x & 0 & 0 & \dots & 0 & x \end{bmatrix} \text{ and B} = \begin{bmatrix} x & 0 & 0 & \dots & 0 & x \\ 0 & x & 0 & \dots & 0 & x \\ 0 & 0 & x & \dots & 0 & x \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & x & x \\ x & x & x & \dots & x & x \end{bmatrix}$$ where x denotes non-zero values.

For the matrix, A, even though it is sparse arrow matrix, even in the first step, as we try solve for Gauss-elimination, a lot of fill-in occurs while trying to zero out the first column. Thus it becomes dense after the first step. This means, rest of the computation for matrix A will have the computation cost same as the dense matrix computations solved earlier , that is $O(n^3)$ while the storage will require full matrix storage, that is , $O(n^2)$.

Interesting things happen with the B matrix since its sparsity structure can be effectively utilized while zeroing out the column elements. The following modified pseudocode will suit the operations on the matrix B for generating the L and U matrices.

PseudoCode:

for k=1:n-1
$l_{n,k} = \frac{a_{n,k}}{a_{k,k}}$
$a_{n,k} = 0$
$a_{n,n} = a_{n,n} - l_{n,k} * a_{k,n}$
$b_n = b_n - l_{n,k} * b_k$

At a specific k, we do not iterate over the rest of the rows after k, except the last row, since others will have their values in k'th column as 0. Also, when we are considering the last row, effectively we are zeroing out its element in the k'th column, hence assigning $a_{n,k} = 0$. Since in the k'th row, we have non-zeros only in the k'th column and the last column, hence the update rule applies only to the n;th column in the n'th row. Similarly, we can also update the b vector's b_n element.

Thus the cost of operation becomes = **5(n-1)** $\approx O(n)$.

In general the L and U matrices can be stored together in a compact form as shown below, with the original matrix discarded

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ l_{21} & u_{22} & u_{23} & \dots & u_{2n} \\ l_{31} & l_{32} & u_{33} & \dots & u_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & \dots & l_{n,n-1} & u_{nn} \end{bmatrix}$$, where we are not storing the unit diagonal elements of the L matrix.

Similarly, we can store the $l_{i,j}$ and $u_{i,j}$ values in such a compact way , such that the matrix looks as below:

$$\text{C} = \begin{bmatrix} u_{11} & 0 & 0 & \dots & u_{1n} \\ 0 & u_{22} & 0 & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & u_{n-1,n-1} & u_{n-1,n} \\ l_{n1} & l_{n2} & \dots & l_{n,n-1} & u_{nn} \end{bmatrix}$$

Thus storage required in this case is $\approx 3n$ or $\approx O(n)$.

For solving the LUx=b, we first solve Ly=b, and the computation for forward substituion will follow the below reduced code:

PseudoCode:

for k=1:n-1

$$y_k = b_k$$

$$y_n = \frac{b_n - \sum_{j=1}^{n-1} C_{n,j} y_j}{1}$$

Cost for forward substitution = 2n $\approx O(n)$

Next, for the backward substituion for Ux=y, the computation will follow the below reduced code:

PseudoCode:

$$x_n = \frac{y_n}{C_{n,n}}$$

for k = n-1:1

$$x_k = \frac{y_k - C_{k,n} * x_n}{C_{k,k}}$$

Cost for backward Substituion = 3n $\approx O(n)$

Thus, total cost for solving LUx=b for the matrix form of B = $O(n) + O(n) + O(n) \approx O(n)$