Homework 2

Nima Leclerc
Cornell University
CS 4220
Numerical Analysis: Linear and Nonlinear Problems

February 21, 2018

Question 1:

When discussing Gaussian elimination and then LU decompositions in class, we considered matrices $G^{(k)}$ that when applied to a matrix A introduced zeros in the k^{th} row below the diagonal. Assuming all of the matrices involved in this problem are $n \times n$, we may write these matrices compactly as

$$G^{(k)} = I - \ell^{(k)} e_k^T,$$

where $\ell^{(k)} = [0, \ldots, 0, \ell_{k+1,k}, \ldots \ell_{n,k}]^T$ is zero in the first k entries. We will now prove two properties of these matrices outlined in class.

(a) Prove that $\left(G^{(k)}\right)^{-1} = I + \ell^{(k)} e_k^T$

To show this, it must be proven that $\left(G^{(k)}\right)^{-1}\left(G^{(k)}\right) = I$. Hence,

$$\left(G^{(k)}\right)^{-1}\left(G^{(k)}\right) = (I+\ell^{(k)}e_k^T)(I-\ell^{(k)}e_k^T) = I^2 - I\ell^{(k)}e_k^T + \ell^{(k)}e_k^T I - \ell^{(k)}e_k^T\ell^{(k)}e_k^T = I - \ell^{(k)}e_k^T + \ell^{(k)}e_k^T - \ell^{(k)}e_k^T\ell^{(k)}e_k^T$$

$$= I - \ell^{(k)}e_k^T\ell^{(k)}e_k^T = I - (\ell^{(k)}e_k^T)^2$$

Hence, the second term in the right and side of the equation above must be 0. Now expanding this product would require only multiplying the non-zero entry of e^k with zero entries of $\ell^{(k)}$, and vice versa. Hence, this would result in the $n \times n$ zero matrix. Hence, this results in $\left(G^{(k)}\right)^{-1}\left(G^{(k)}\right) = I$. This proves that $\left(G^{(k)}\right)^{-1} = I + \ell^{(k)}e_k^T$. Q.E.D.

(b) Prove that $\left(G^{(1)}\right)^{-1}\left(G^{(2)}\right)^{-1}\cdots\left(G^{(n-1)}\right)^{-1} = I + \sum_{k=1}^{n-1} \ell^{(k)}e_k^T$

Here, it requires to be shown that,

$$\prod_{k=1}^{n-1}\left(G^{(k)}\right)^{-1} = I + \sum_{k=1}^{n-1}\ell^{(k)}e_k^T$$

We can demonstrate this by expanding out to to $n = 4$,

$$\prod_{k=1}^{4-1}\left(G^{(k)}\right)^{-1} =$$

$$\left(G^{(1)}\right)^{-1}\left(G^{(2)}\right)^{-1}\left(G^{(3)}\right)^{-1} =$$

$$(I + \ell^{(1)}e_1^T)(I + \ell^{(2)}e_2^T)(I + \ell^{(3)}e_3^T) =$$

$$(I^2 + I\ell^{(2)}e_2^T + \ell^{(1)}e_1^T I + \ell^{(1)}e_1^T\ell^{(2)}e_2^T)(I + \ell^{(3)}e_3^T) =$$

$$I^2 + I^2\ell^{(1)}e_1^T + I^2\ell^{(2)}e_2^T + I\ell^{(1)}e_1^T\ell^{(2)}e_2^T + I\ell^{(3)}e_3^T + I\ell^{(2)}e_2^T\ell^{(3)}e_3^T + \ell^{(1)}e_1^T\ell^{(2)}e_2^T\ell^{(3)}e_3^T$$

Grouping the terms together, this follows the trend

$$\prod_{k=1}^{3}\left(G^{(k)}\right)^{-1} = I + \sum_{k=1}^{3}\ell^{(k)}e_k^T + \sum_{j=1}^{3}\sum_{k=2}^{3}\ell^{(j)}e_j^T\ell^{(k)}e_k^T$$

The higher order terms: $I\ell^{(1)}e_1^T\ell^{(2)}e_2^T, I\ell^{(2)}e_2^T\ell^{(3)}e_3^T, \ell^{(1)}e_1^T\ell^{(2)}e_2^T\ell^{(3)}e_3^T$ drop to zero, since these products of matrix elements between matrices $\ell^{(i)}e_i^T$ and $\ell^{(j)}e_j^T$ are ones of products of either zero with zero entries, or zero with non-zero entries. Collectively, this results in a zero matrix. All these cross terms correspond to the double sum in the above expression. Hence, this double sum evaluates to a zero matrix and the product becomes,

$$\prod_{k=1}^{3}\left(G^{(k)}\right)^{-1} = I + \sum_{k=1}^{3}\ell^{(k)}e_k^T$$

This can be generalized to to a product of $n-1$ of these G matrix inverses. Hence, one can say

$$\prod_{k=1}^{n-1}\left(G^{(k)}\right)^{-1} = I + \sum_{k=1}^{n-1} \ell^{(k)} e_k^T$$

Q.E.D.

Question 2:

Assume that you are given an $n \times n$ non-singular matrix of the form

$$A = D + ue_n^T + e_n v^T$$

where u and v are length n vectors, and D is a diagonal matrix whose diagonal entries we will encode in the vector d, *i.e.* $D_{i,i} = d_i$. (In the course of thinking about this problem, you may stumble across the so-called Sherman-Morrison-Woodbury formula and find that mathematically it provides a potential solution—you are welcome to try it out and see what happens. However, it will yield an even worse solution, both in terms of accuracy and the residual, for the problem instance in part (c) than working off of an LU factorization (even though we have omitted pivoting in this case).

*(a) If we assume that we may compute the LU decomposition of A without any pivoting (**note that in general this is not a good idea, but for the purpose of this question we will make such an assumption**), devise a means to solve a linear system of the form $Ax = b$ using $\mathcal{O}(n)$ time. Here we will also consider the cost of memory allocation and therefore you cannot form A explicitly as that would take $\mathcal{O}(n^2)$ time.*

Here, one first seeks 2 $n \times n$ matrices L and U such that $A = LU$, for the given matrix A. If one expands A, the following is obtained,

$$A = D + ue_n^T + e_n v^T = \begin{bmatrix} d_{11} & 0 & \dots & & \\ 0 & \ddots & 0 & \dots & \\ \vdots & 0 & & \dots & d_{nn} \end{bmatrix} + \begin{bmatrix} 0 & 0 & \dots & u_1 \\ \vdots & 0 & \dots & \vdots \\ 0 & 0 & \dots & u_n \end{bmatrix} + \begin{bmatrix} 0 & 0 & \dots & 0 \\ \vdots & 0 & \dots & \vdots \\ v_1 & v_i & \dots & v_n \end{bmatrix} =$$

$$\begin{bmatrix} d_{11} & 0 & \dots & u_1 \\ 0 & \ddots & \dots & u_i \\ \vdots & 0 & d_{ii} & \vdots \\ v_1 & v_i & \dots & (v_n + u_n + d_{nn}) \end{bmatrix}$$

Hence, one seeks L and U without pivoting, such that LU yields the above matrix. Let's start with finding L,

$$L = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & \ddots & \dots & \vdots \\ \vdots & 0 & 1 & \\ v_1/d_{11} & \dots & v_{n-1}/d_{n-1,n-1} & 1 \end{bmatrix}$$

This matrix can be computed straightforwardly, since it is known that all diagonal entries have value of 1, while every other entry except row n takes a value of 0. Hence, one only needs to determine the elements of the nth row. Similarly, the matrix U would take the form,

$$U = \begin{bmatrix} d_{11} & 0 & \dots & u_1 \\ 0 & \ddots & \dots & u_i \\ \vdots & 0 & d_{ii} & \vdots \\ 0 & \dots & 0 & (d_{nn} + u_n + v_n) - \sum_{i=1}^{n-1} \frac{v_i u_i}{d_{ii}} \end{bmatrix}$$

Now, the challenge remains in solving $Ax = b$ for vectors x, b of dimension $n \times 1$. Using the $A = LU$ decomposition above, one can solve the expressions $Ly = b$ and $Ux = y$ separately. Each of these operations can be done in $\mathcal{O}(n)$ time. Now, solving for y one obtains $y_i = b_i$ for $i \leq n - 1$. y_n would be computed as follows

$$y_n = b_n - \sum_{i=1}^{n-1} \frac{y_i v_i}{d_{ii}}$$

Hence,

$$y = \begin{bmatrix} b_i \\ \vdots \\ b_n - \sum_{i=1}^{n-1} \frac{y_i v_i}{d_{ii}} \end{bmatrix}$$

This algorithm would take $\mathcal{O}(n)$ time, for b_i values are simply being assigned to y_i (linear time) for values $i < n$ and computing b_n would take $\mathcal{O}(n)$ time since a summation is performed over $n - 1$ elements. Then, the overall complexity would be $\mathcal{O}(n)$.
Now, using this computed y vector, the solution to $Ux = y$ can be determined. The last element of x, x_n can be determined as follows,

$$x_n = \frac{b_n - \sum_{i=1}^{n-1} \frac{y_i v_i}{d_{ii}}}{(d_{nn} + u_n + v_n) - \sum_{i=1}^{n-1} \frac{v_i u_i}{d_{ii}}}$$

Similarly, for $i \leq n - 1$ one would seek a solution to (using previously determined x_n) ,

$$\sum_{i=1}^{n-1} d_i x_i + x_n u_i = y$$

This can be solved as follows,

$$x = \sum_{i=1}^{n-1} \frac{y_i - x_n u_i}{d_{ii}}$$

This is simply a sum over all i values, so it would would take $\mathcal{O}(n)$ time. This concludes the solution to x in the system $Ax = b$ for this particular A.

(b) Implement your method and demonstrate that it achieves the desired scaling.
Below is the implemented algorithm in Matlab, for the defined matrix A and vector, b.

```
A= [6 7 8; 1 3 5; 13 17 12]; %test A
b1=[52, 21, 33]; %test b
n = length(b1); %len of b
alpha= A(n,n); %%d(n, n) + u(n) + v(n)
x=zeros(n);
y=zeros(n);
u1 = A(1:n-1, n);
v1 = A(n, 1:n-1);
d1=diag(A);
d2= d1(1:end-1);
s1 = zeros(1,5);
```

```
temp=0;
for k= 1:n-1
  temp= temp + (b1(k)*v1(k))/d1(k);
  y(k) = b1(k);
end
y(n) = b1(n)-temp;
temp2=0;
for k=1:n-1
    temp2 = temp2 + (v1(k)*u1(k))/d(k);
end
x(n) = y(n)/(alpha-temp2);
for k=1:n-1
x(k) = x(k)+((y(k)-x(n)*u1(k))/d2(k));
end
```

The attached plot presents the desired linear scaling.
(c) Attached is the error plot.

(d) By avoiding pivoting, we were able to get a fast, $\mathcal{O}(n)$ algorithm. However, what do you observe about the accuracy of the solution and how might you explain it? The accuracy of the solution decreases with pivoting.

Question 3:

Consider the following $n \times n$ matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ -1 & \cdots & -1 & 1 & 0 & 1 \\ -1 & \cdots & -1 & -1 & 1 & 1 \\ -1 & \cdots & -1 & -1 & -1 & 1 \end{bmatrix}$$

that has all ones one the diagonal and in the last column, and all the entries below the diagonal are -1.
(1) Consider computing an LU decomposition of A (with or without partial pivoting, the answer will be the same). Work out an expression for the largest entry of U in terms of n.

One can compute the LU decomposition for $n = 1 - 4$ and observe the trend, particularly the factor U. Below are the decompositions for $n = 2 - 4$, (computed, via the lu routine in Matlab)

$$U_2 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}, U_3 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 1 & 4 \end{bmatrix}, U_4 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 8 \end{bmatrix},$$

The trend is clear that the largest value in U for each of these matrices scales with n by the expression, $b = 2^{n-1}$. This is the relationship between matrix size and the largest entry in the upper triangular matrix, U for $A = LU$.

(b) If we computed an LU factorization in practice and encountered an entry of this size do you think it would be problematic? why or why not?
This would be problematic if we encountered a matrix of a very large size. We define the machine precision of the LU decomposition as,

$$\mathcal{O}(\mu) = \frac{||\delta A||_2}{||L||_2||U||_2}$$

Hence, if we encountered larger matrices A, we would encounter an exponentially larger values norms of U, as shown in part (a). Effectively, this would yield an exponential decrease in precision (exponential increase in error) as described by the above equation. Hence, this becomes very problematic.

(c) Can you think of a different style of pivoting that would avoid the issues highlighted here? Briefly outline/describe your proposed procedure.

A different style of pivoting in this case would be designed in a manner, such that we avoid the generation of large entries in the last column of U. In the partial pivoting used in the LU decomposition, one encounters the swapping of rows (due to the permutation matrix), such that the last row of U is larger than the first its preceding row. Instead, one should design an algorithm such that the rows are swapped for rows with large entries of A are in lower order rows of A. Hence when pivoting, the largest values of A should be at the in the first rows of A. Effectivley, this would avoid the generation of large non-zero entries accumulating in the last column of U in the decomposition. This would then avoid the error issue, having a smaller norm of U and hence increase the machine precision.