

CS6210: Homework 3

Christopher Martin

October 4, 2016

1. The *condition number* of an eigenvalue λ of a given matrix A is defined as

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

where \mathbf{x} is a (right) eigenvector of the matrix, satisfying $A\mathbf{x} = \lambda\mathbf{x}$, and \mathbf{w} is a left eigenvector, satisfying $\mathbf{w}^T A = \lambda\mathbf{w}^T$. Both \mathbf{x} and \mathbf{w} are assumed to have a unit ℓ_2 -norm. Loosely speaking, the condition number determines the difficulty of computing the eigenvalue in question accurately; the smaller $s(\lambda)$ is, the more numerically stable the computation is expected to be.

Determine the condition number of the eigenvalue 4 for the two matrices discussed in Example 4.7. Explain the meaning of your results and how they are related to the observations made in the example.

Solution:

In Example 4.7, the given matrix and its eigenvalue 4 is

$$A = \begin{pmatrix} 4 & 1 \\ 0 & 4 \end{pmatrix}; \quad \mathbf{x} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad \mathbf{w} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

In calculating the condition number, we get

$$\begin{aligned} S(\lambda) &= \frac{1}{\mathbf{x}^T \mathbf{w}} \\ &= \frac{1}{0} = \infty \end{aligned}$$

This means that the matrix is *ill-conditioned* and is the reason for brining out the large discrepancy in eigenvalues with such a small perturbation.

2. The *Gauss-Jordan method* used to solve the prototype linear system can be described as follows. Augment A by the right-hand-side vector \mathbf{b} and proceed as in Gaussian Elimination, except use the pivot element $a_{k,k}^{(k-1)}$ to eliminate not only $a_{i,k}^{(k-1)}$ for $i = \{k+1, \dots, n\}$ but also the elements $a_{i,k}^{(k-1)}$ for $i = \{1, \dots, k-1\}$, *i.e.*, all elements in the k^{th} column other than the pivot. Upon reducing $(A|\mathbf{b})$ into

$$\left[\begin{array}{cccc|c} a_{1,1}^{(n-1)} & 0 & \cdots & 0 & b_1^{(n-1)} \\ 0 & a_{2,2}^{(n-1)} & \ddots & \vdots & b_2^{(n-1)} \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & a_{n,n}^{(n-1)} & b_n^{(n-1)} \end{array} \right]$$

the solution is obtained by setting

$$x_k = \frac{b_k^{(n-1)}}{a_{k,k}^{(n-1)}}, \quad k = \{1, \dots, n\}$$

This procedure circumvents the backward substitution part necessary for the Gaussian Elimination algorithm.

- (a) Write a pseudocode for this Gauss-Jordan procedure using, *e.g.*, the same format as for the one appearing in Section 5.2 for Gaussian Elimination. You may assume that no pivoting (*i.e.*, no row interchanging) is required.

Solution:

Algorithm 1 Gauss-Jordan Elimination

Input: $A \in \mathbb{R}^{n \times m}$

Output: $\vec{x} \in \mathbb{R}^{n \times 1}$

```

1: for  $i = 1, \dots, n$  do
2:   Let  $p$  be the smallest integer with  $i \leq p \leq n$  and  $a_{p,i} \neq 0$ 
3:   if no integer  $p$  can be found then
4:     Return 'No unique solution exists'
5:   end if
6:   if  $p \neq i$  then
7:      $E_p \leftrightarrow E_i$ 
8:   end if
9:   for  $j = 1, \dots, i-1, i+1, \dots, m$  do
10:     $m_{j,i} = \frac{a_{j,i}}{a_{i,i}}$ 
11:     $(E_j - m_{j,i} \cdot E_i) \rightarrow E_j$ 
12:   end for
13: end for{Elimination Process}
14: for  $i = 1, \dots, n$  do
15:    $x_i = \frac{a_{i,n+1}}{a_{i,i}}$ 
16: end for
17: Return  $\vec{x}$ 

```

- (b) Show that the Gauss-Jordan method requires $n^3 + \mathcal{O}(n^2)$ floating point operations for one right-hand-side vector \mathbf{b} – roughly 50% more than what's needed for Gaussian Elimination

Solution:

This can be easily proven (assuming the given matrix is $n \times n$). Line 1 and 9 of the code produce $\mathcal{O}(n^2)$ divisions for line 10. However, Line 11 has to sub in the elements of each row ($\mathcal{O}(n)$), which would make that term be $\mathcal{O}(n^3)$ since it has the two for loops, plus iterating over each element in the columns to change their values.

3. Let A and T be two nonsingular $n \times n$ real matrices. Furthermore, suppose we are given two matrices L and U such that L is the unit lower triangular, U is the upper triangular, and

$$TA = LU$$

Write an algorithm that will solve the problem

$$A\mathbf{x} = \mathbf{b}$$

for any given vector \mathbf{b} in $\mathcal{O}(n^2)$ complexity. First, explain briefly yet clearly why your algorithm requires only $\mathcal{O}(n^2)$ flops (you may assume without proof that solving an upper triangular or a lower triangular system requires only $\mathcal{O}(n^2)$ flops). Then, specify your algorithm in detail (including the details for lower and upper triangular systems) using pseudocode or a MATLAB script.

Solution:

By multiplying T from the left to the equation $A\mathbf{x} = \mathbf{b}$ gives us $TA\mathbf{x} = T\mathbf{b} = \mathbf{c}$ and $LU\mathbf{x} = \mathbf{c}$. This can be solved with normal LU-Decomposition, by first doing a forward and then a backward substitution. Multiplying T to vector b needs $\mathcal{O}(n^2)$ operations, and since solving the upper triangular or lower triangular system requires only $\mathcal{O}(n^2)$ flops, solving this equation needs $\mathcal{O}(n^2)$ flops overall.

Algorithm 2 LU Decomposition

```
1:  $\mathbf{c} = T\mathbf{b}$  {Multiply  $T$  from the left on  $A\mathbf{x} = \mathbf{b}$ , with RHS as  $\mathbf{c}$ }
2:  $\mathbf{y} = \mathbf{c}$ 
3: for  $i = 2, \dots, n$  do
4:   for  $j = 1, \dots, i - 1$  do
5:      $y_i = c_i - L_{i,j}y_j$ 
6:   end for
7: end for{Solve  $L\mathbf{y} = \mathbf{b}$  via forward substitution}
8:  $\mathbf{x} = \mathbf{y}$ 
9:  $x_n = \frac{y_n}{U_{n,n}}$ 
10: for  $i = n - 1, \dots, 1$  do
11:   for  $j = i + 1, \dots, n$  do
12:      $x_i = \frac{y_i - U_{i,j}x_j}{U_{i,i}}$ 
13:   end for
14: end for{Solve  $U\mathbf{x} = \mathbf{y}$  by backward substitution}
```

4. The classical way to invert a matrix A in a basic linear algebra course augments A by the $n \times n$ identity matrix $\mathbb{1}$ and applies the Gauss-Jordan algorithm of Exercise 2 to this augmented matrix (including the solution part, *i.e.*, the division by the pivots $a_{k,k}^{(n-1)}$). Then A^{-1} shows up where $\mathbb{1}$ initially was.

How many floating point operations are required for this method? Compare this to the operation count of $\frac{8}{3}n^3 + \mathcal{O}(n^2)$ required for the same task using LU-decomposition (see Example 5.5).

Solution:

To add (argument) $\mathbb{1}$ to A in order to find the inverse, this takes $\mathcal{O}(n)$ operations. Following this, we can convert A into $\mathbb{1}$ by utilizing row operations, where the *former* matrix $\mathbb{1}$ on the right becomes A^{-1} .

We can count up the operations in Algorithm 3, to get the total number of operations, which results in:

$$\sum_{k=1}^n (2n + 2(2n - 1)^2 + 4n) + \mathcal{O}(n^2) = 8n^3 + \mathcal{O}(n^2)$$

Algorithm 3 Matrix Inverse

```
1: for  $i = 1, \dots, n$  do
2:   for  $j = 1, \dots, 2n$  do
3:      $l_{j,i} = \frac{a_{j,i}}{a_{i,i}}$ 
4:     for  $k = 1, \dots, 2n$  do
5:       if  $i \neq j$  then
6:          $a_{j,k} = a_{j,k} - l_{j,i}a_{i,k}$ 
7:       end if
8:     end for
9:      $b_j = b_j - l_{j,i}b_i$ 
10:  end for
11: end for
12: for  $i = 1, \dots, n$  do
13:    $b = a_{i,i}$ 
14:   for  $j = 1, \dots, 2n$  do
15:      $a_{i,j} = \frac{a_{i,j}}{b}$ 
16:   end for
17: end for {Converting the left matrix to  $\mathbb{1}$ }
```

5. The Cholesky algorithm given on page 116 has all those wretched loops as in the Gaussin Elimination algorithm in its simplest form. In view of Section 5.4 and the program `ainvb` we should be able to achieve also the Cholesky decomposition effect more efficiently.

Write a code implementing the Cholesky decomposition with only one loop (on k), utilizing outer products.

Solution:

Algorithm 4 Cholesky Factorization (Outer Products)

```
1: for  $i = 1, \dots, n$  do
2:    $A_{i,i} = \sqrt{A_{i,i}}$ 
3:   for  $j = i + 1, \dots, n$  do
4:      $A_{j,i} = \frac{A_{j,i}}{A_{i,i}}$ 
5:   end for
6:   for  $j = i + 1, \dots, n$  do
7:     for  $k = j, \dots, n$  do
8:        $A_{k,j} = A_{k,j} - A_{k,i}A_{j,i}$ 
9:     end for
10:  end for
11: end for
```

6. Consider the LU decomposition of an upper Hessenberg (no, it's not a place in Germany) matrix, defined on the facing page, assuming that no pivoting is needed: $A = LU$.

- (a) Provide an efficient algorithm for this LU decomposition (do not worry about questions of memory access and vectorization).

Solution:

Algorithm 5 LU Decomposition of Upper Hessenberg

```

1: for  $i = 1, \dots, n - 1$  do
2:    $j = i + 1$  {Since  $p = 2$ , the second loop on the iterator  $j$  only has a single value}
3:    $l_{j,i} = \frac{a_{j,i}}{a_{i,i}}$ 
4:   for  $k = i + 1, \dots, i + n - 1$  do
5:      $a_{j,k} = a_{j,k} - l_{j,i}a_{i,k}$ 
6:   end for
7: end for

```

- (b) What is the sparsity structure of the resulting matrix L (*i.e.*, where are its non-zeros)?

Solution:

In general, L is a lower-triangular matrix and has a lower-bandwidth of $p = 2$ and an upper-bandwidth of $q = n$. Therefore, we can use the “LU Decomposition for Banded Matrices” on page 120 in the book, which is stated as Algorithm 5 above.

- (c) How many operations (to a leading order) does it take to solve a linear system $A\mathbf{x} = \mathbf{b}$, where A is upper Hessenberg?

Solution:

Line 5 in Algorithm 5 has $(n - 2)$ multiplications and subtractions, resulting in a total of $2(n - 2)$. Therefore, the total number of operations can be found as

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

Where the total time for the decomposition is $\mathcal{O}(n^2)$, for the forward substitution is $\mathcal{O}(n^2)$, and for the backward substitution is $\mathcal{O}(n^2)$. Thus, the resulting overall number of operations is $\mathcal{O}(n^2)$.

- (d) Suppose now that partial pivoting is applied. What are the sparsity patterns of the factors of A ?

Solution:

Page 121 of the text states:

“If the usual row partial pivoting is applied, then the band structure may be altered due to row interchanges. Using such pivoting, the upper-bandwidth q of U may increase to $q + p - 1$.”

In any Hessenberg Matrix, since $p = 2$ and $q = n$, the upper-bandwidth will stay the same and partial pivoting does not have any significant effect on the sparsity of it.

7. For the arrow matrices of Example 5.15, determine the overall storage and flop count requirements for solving the systems with A and with B in the general $n \times n$ case.

Solution:

Converting matrix A to B can be done with an implementation of a row and column swap, requiring us to find the flop count for B . B can be converted into an upper/lower-triangular matrix in $\mathcal{O}(n^2)$ flops, with $(n-1)$ elementary operations. From here, we can solve the linear system with backwards/forwards substitution, which requires $\mathcal{O}(n^2)$ flops. Therefore, the total number of operations is $\mathcal{O}(n^2)$.

The storage requirements can be determined by only storing the non-zeros of the matrix. In the case of an *arrow matrix* from Example 5.15, the *max* is $\mathcal{O}(n)$ as it would be $\mathcal{O}(n)$ along the diagonal, and $\mathcal{O}(2n)$ on the “arrow head” part of the matrix, resulting in an overall storage of $\mathcal{O}(n)$.