CS6210: Homework 3

Christopher Mertin

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

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

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 $i = \{k+1, \ldots, n\}$ but also the elements $a_{i,k}^{(k-1)}$ for $i = \{1, \ldots, k-1\}$, *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

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

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

for $i = 1, \ldots, n$ **do**
  Let p be the smallest integer with $i \le p \le n$ and $a_{p,i} \neq 0$
  if no integer p can be found **then**
    Return 'No unique solution exists'
  end if
  if $p \neq i$ **then**
    $E_p \leftrightarrow E_i$
  end if
  for $j = 1, \ldots, i-1, i+1, \ldots, m$ **do**
    $m_{j,i} = \frac{a_{j,i}}{a_{i,i}}$
    $(E_j - m_{j,i} \cdot E_i) \to E_j$
  end for
end for{Elimination Process}
for $i = 1, \ldots, n$ **do**
  $x_i = \frac{a_{i,n+1}}{a_{i,i}}$
end for
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 alogirhtm 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 equaiton needs $\mathcal{O}(n^2)$ flops overall.

Algorithm 2 LU Decomposition

$\mathbf{c} = T\mathbf{b}$ {Multiply T from the left on $A\mathbf{x} = \mathbf{b}$, with RHS as $\mathbf{c}$}
$\mathbf{y} = \mathbf{c}$
for $i = 2, \ldots, n$ **do**
  for $j = 1, \ldots, i-1$ **do**
    $y_i = c_i - L_{i,j} y_j$
  end for
end for{Solve $L\mathbf{y} = \mathbf{b}$ via forward substitution}
$\mathbf{x} = \mathbf{y}$
$x_n = \frac{y_n}{U_{n,n}}$
for $i = n-1, \ldots, 1$ **do**
  for $j = i+1, \ldots, n$ **do**
    $x_i = \frac{y_i - U_{i,j}}{U_{i,i}}$
  end for
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 (agument) $\mathbb{1}$ 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:

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

Algorithm 3 Matrix Inverse

for $i = 1, \dots, n$ **do**
  for $j = 1, \dots, 2n$ **do**
    $l_{j,i} = \frac{a_{j,i}}{a_{i,i}}$
    for $k = 1, \dots, 2n$ **do**
      if $i \neq j$ **then**
        $a_{j,k} = a_{j,k} - l_{j,i} a_{i,k}$
      end if
    end for
    $b_j = b_j - l_{j,i} b_i$
  end for
end for
for $i = 1, \dots, n$ **do**
  $b = a_{i,i}$
  for $j = 1, \dots, 2n$ **do**
    $a_{i,j} = \frac{a_{i,j}}{b}$
  end for
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:

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:

   (b) What is the sparsity structure of the resulting matrix L (*i.e.*, where are its nonzeros)?
   Solution:

   (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:

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

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: