M439 Numerical Analysis Lab: Solution of Linear Systems AX=B by Gaussian Elimination with Pivoting

---

Matrix Form For Linear Systems of Equations

The matrix form for a linear system of *m* equations in *n* variables is A**x** = **b**, where A is the *m*x*n* matrix of coefficients, **x** is the *n*x1 column vector whose elements are the variables, and **b** is the *m*x1 column vector whose elements are the right hand sides of the system of equations. For example, the system below on the left has the matrix form given on the right:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 6 \\ 2x_1 - 3x_2 + 2x_3 &= 14 \\ 3x_1 \quad x_2 - 1x_3 &= -2 \end{aligned} \qquad \begin{pmatrix} 1 & 2 & 3 \\ 2 & -3 & 2 \\ 3 & 1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 14 \\ -2 \end{pmatrix}$$

1. In MATLAB the command to solve this system of equations numerically uses the matrix division operator "\". You create the matrix of coefficients A, the column vector **b** whose entries are the right-hand side of the equation, and then compute **A\b.** (That is, to find **x** such that A**x**=**b**, "divide" A by b.) Note*: A\b gives the solution only if there is a unique solution.*

```
» A = [1 2 3; 2 -3 2; 3 1 -1]
```
Enter the matrix of coefficients A.

```
A =
1 2 3
2 -3 2
3 1 -1
```

```
» b = [6; 14; -2]
```
Enter the column vector representing the right-hand side of system.

```
b =
6
14
-2
```

```
» x = A\b
```
Obtain the solution.

```
x =
1.0000
-2.0000
3.0000
```

```
» A*x
```
Verify the solution -- does Ax = b?

```
ans =
6.0000
14.0000
-2.0000
```

2. Compute the solution to the following systems of equations using the **A\b** operation. Write down the answer returned by MATLAB. Also check the answer returned by MATLAB by "plugging" it in to the system -- i.e., compute **A*ans** and verify that **b** is

returned.

$$
\begin{aligned}
4x_1 + 2x_2 + 3x_3 + x_4 &= 26 \\
x_1 + 4x_2 + x_3 + 2x_4 + 5x_5 &= 58 \\
-2x_1 + 3x_2 + 5x_4 + 6x_5 &= 73 \\
x_1 + 4x_2 + 5x_3 + 2x_4 + 7x_5 &= 72 \\
x_2 + 3x_3 - 2x_4 + 4x_5 &= 13
\end{aligned}
$$

Solution:

In the implementation of the '\'' operator in MATLAB, a number of numerical methods are used to optimize efficiency and accuracy. We look at some of the techniques below (Gaussian elimination with partial pivoting and back substitution).

Gaussian Elimination and Upper Triangular Form for Matrices in MATLAB

Elementary Row Operations

To bring a matrix in upper triangular form, we must use elementary row operations.
An ***elementary row operation*** on a matrix is any of the following:

i) (Scaling) Multiply one row by a nonzero number.
ii) (Replacement) Replace a row with by adding a multiple of one row to that row.
iii) (Interchanges) Interchange two rows.

The matrix form for a system of linear equations can be represented by the ***augmented matrix*** **[**A b] where A is the matrix of coefficients on the left hand side of the equations, and column b is the constants on the right hand side of the system. In MATLAB, we would enter **[A b]**). To solve a system of equations in matrix form, we can use the following method:

Method of Gaussian Elimination: Row-reduce the augmented matrix to upper echelon form -- called upper triangular form for square matrices -- using elementary row operations. Solve the corresponding system using back substitution.
At each step, starting with top row, we use that row to achieve zeros below the leftmost nonzero value in that row. We then proceed to the next row and repeat, until in upper triangular form.

Elementary Row Operations

A. To refer to an individual element in a matrix C, use C(i,j), where i is the row number and j is the column number.
B. To refer to the *i*th row of a matrix C in MATLAB, use C**(*i*,:)**. For example, C**(3,:)** is the third row of C. To refer to the jth column of a matrix in MATLAB, use C(:,j). **,:)**. For example, C**(:,3)** is the third column of C.

C. Perform elementary row operations on A as follows:

i) (Scaling) To multiply one row by a nonzero scalar *c*, use

>> C***(i*,:) = *c*C(i*,:)**

Example: To multiply the third row of C by 4, enter

```
>> C(3,:) = 4 *C(3,:)
```

To divide the second row of C by the element C**(2,1)**, enter

```
>> C(2,:) = C(2,:)/C(2,1)
```

ii) (Replacement) To add *c* times the *j*th row to the *i*th row of C, use

C(i,:) = C(i,:) + *cC(j,:)**

Example: To add 2 times row 1 to row 2 of C use

```
>> C(2,:) = C(2,:) + 2*C(1,:)
```

iii) (Interchanges) To interchange the *i*th and *j*th row use

>> **C([i j],:) = C([j i],:)**

Example: To swap rows 3 and 4 of C use

```
>> C([3 4],:) = C[4 3],:)
```

***Important Note*: When multiplying or dividing by the scalar value of an element in the matrix, use the matrix element, not its numeric value, to avoid rounding off.**

4. Practice Solve using Gaussian Elimination

$$
\begin{aligned}
x_1 + 2x_2 + x_3 + 4x_4 &= 13 \\
2x_1 \qquad + 4x_3 + 3x_4 &= 28 \\
4x_1 + 2x_2 + 2x_3 + x_4 &= 20 \\
-3x_1 + x_2 + 3x_3 + 2x_4 &= 6
\end{aligned}
$$

Enter the following commands (use recalling of commands and editing astutely!)

```
» C = [1 2 1 4 13; 2 0 4 3 28; 4 2 2 1 20; -3 1 3 2 6]
» C(2,:) = C(2,:) - (C(2,1)/C(1,1))*C(1,:)
» C(3,:) = C(3,:) - (C(3,1)/C(1,1))*C(1,:)
» C(4,:) = C(4,:) - (C(4,1)/C(1,1))*C(1,:)
» C(3,:) = C(3,:) - (C(3,2)/C(2,2))*C(2,:)
» C(4,:) = C(4,:) - (C(4,2)/C(2,2))*C(2,:)
» C(4,:) = C(4,:) - (C(4,3)/C(3,3))*C(3,:)
```

Pivots and Pivoting to Reduce Error

The element on the diagonal of the matrix of coefficients that is used to achieve zeros below it is called the ***pivot***. In order to reduce errors in the reduction of a matrix to upper triangular form, ***partial pivoting*** is frequently used in Gaussian elimination.. The revised method is as follows:

Repeat for each row in the matrix (except the last):
Step 1. Find the first column with nonzero components (called the ***pivot column***). Select the component in that column with the largest absolute value. This component is called the ***pivot***.
Step 2. Swap the row containing this pivot with the current row. (*This is the pivoting*).
Step 3. Perform the necessary row operations to obtain zeros in the pivot column below the pivot position.

Pivoting can help reduce round-off error by assuring that in the reduction the largest possible component in a column is used for dividing. Dividing by a small number, which has been rounded already, may produce a significant round-off error. Pivoting may also need to be performed when the pivot in the next row to be processed -- A[p,p] --is zero -- this is called ***trivial pivoting***.

6.. Enter the commands shown in the following MATLAB session demonstrating partial pivoting for a sample matrix:

```
» C = [2 -3.5 1; -5 3 3.3; 12 7.8 4.6];        Enter matrix C.
» C([1,3],:)=C([3,1],:)  Pivot 12 in location (3,1) – swap to get in position (1,1).
C =
12.0000 7.8000 4.6000
-5.0000 3.0000 3.3000
2.0000 -3.5000 1.0000
» C(2,:) = C(2,:)-(A(2,1)/C(1,1))*C(1,:)   Obtain 0's in positions below pivot
C =
12.0000 7.8000 4.6000
0 6.2500 5.2167
2.0000 -3.5000 1.0000
» C(3,:) = C(3,:)-(C(3,1)/C(1,1))*C(1,:)
C =
12.0000 7.8000 4.6000
0 6.2500 5.2167
0 -4.8000 0.2333
Note:  Pivot is 6.25 in position (2,2) -- don't need to swap
»  C(3,:) = C(3,:) - (C(3,2)/C(2,2))*C(2,:)      Get 0 in position below pivot

C =
12.0000 7.8000 4.6000
0 6.2500 5.2167
0 0 4.2397
```

Why We Use Partial Pivoting

Consider the system of equations $\begin{array}{r} y = 1 \\ x + y = 2 \end{array}$ with augmented matrix form: $\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}$

The solution to this system is $x = 1$, $y = 1$

If we assume that the coefficients of x and y in the system represent data measurements subject to error, then the 0 in the matrix could be a value close to zero, say 10^{-12}. We would like the solution to this system to be as close as possible to the true solution.

Enter the following set of commands to set up such a augmented matrix called M and solve for x and y using naïve Gaussian elimination and back substitution.

```
epsilon = 10^-12
M = [epsilon 1 1 ; 1 1 2]
M(2,:) = M(2,:)-(M(2,1)/M(1,1))*M(1,:)
Y1 = M(2,3)/M(2,2)
x1= (M(1,3) - M(1,2)*y)/M(1,1)
```

Now repeat this process with the modification of using partial pivoting first

```
M = [epsilon 1 1 ; 1 1 2]
M([1 2],:) = M([2 1],:)
M(2,:) = M(2,:)-(M(2,1)/M(1,1))*M(1,:)
Y2 = M(2,3)/M(2,2)
x2 = (M(1,3) - M(1,2)*y)/M(1,1)
```

Compare the values (x1,y1) and (x2,y2) representing the two solutions, What happened in naïve Gaussian Elimination cause the loss of accuracy?