MATLAB Project: Using Backslash to Solve Ax = b Name_____

Purpose: To learn about backslash and why it is the preferred method for solving systems $A\mathbf{x} = \mathbf{b}$ when A is invertible

Prerequisite: Section 2.2 and the discussion of condition number in Section 2.3

MATLAB functions used: \, `inv`, `format`, `rand`, `norm`, `diary`, `hilb`; and `ref` from Laydata4 Toolbox

Background: This project is about square invertible matrices only. Suppose A is such a matrix and you want to solve $A\mathbf{x} = \mathbf{b}$. By Theorem 5 in Section 2.2, there is a unique solution to this system. MATLAB has a special operator called backslash for solving this type of system, and it usually gives excellent results. It is the method people use in professional settings. To use it, store A and **b** and type **`A\b`** .

You may know two other methods in MATLAB for solving the matrix equation $A\mathbf{x} = \mathbf{b}$. You could type **`ref([A b])`** to get the reduced echelon form of the augmented matrix, and then read the solution from its last column. Alternatively, since A is assumed to be invertible here, you could type **`inv(A)*b`** , which by Theorem 5 must give the unique solution.

Backslash is the best of these methods. It uses an algorithm that is fast and minimizes roundoff error. It also checks the condition number of the coefficient matrix. If the condition number is large, it will be hard to get an accurate answer using any numerical method. Fortunately such matrices occur rarely in real world problems. But if backslash does detect a very large condition number, it will warn you by printing a message "`Matrix is close to singular or badly scaled. Results may be inaccurate`." Do not ignore such a warning if you ever see it, for it means the solution is probably not correct to very many digits. If you need more accuracy, consult a numerical analyst.

The **`inv`** function also checks the condition number, but calculating A^{-1} requires a lot more arithmetic than backslash.

It is definitely not wise to use `ref` to solve real world problems. That function was written to help students learn linear algebra, so its algorithm is not optimal, and `ref` will not warn you if your linear system is one of those rare ones for which it is hard to get an accurate solution.

1. (MATLAB) Here you will use the square matrices in exercises 29, 31, 39 and 41 in Section 2.2. For each of these, you will create a linear system $A\mathbf{x} = \mathbf{b}$ and solve it using all three methods described above. You will not see any warnings, so these are "good" problems. You will also see that the solutions are almost identical as expected.

(a) To get started, determine the path where you will store your work. For example, if you install a flash drive into the computer's drive `E:` drive, type **`diary E:\solve`** to open a file called "`solve`" on your flash drive.

```
diary E:\solve
format compact     (this causes fewer blank lines to be printed, so more results fit on the screen)
format long e      (tell MATLAB to display numbers in exponent format with 15 digit mantissas)
```

Type the following lines to use the matrix in exercise 29 for the problem here:

```
c2s2                                  (Opens Chapter 2 Section 2 Problems)
29                                    (Loads the matrix for problem #29)
[n,n] = size(A);
b = rand(n,1)                         (create a 2x1 column with random number entries)
x1 = A\b                              (Method 1: solve Ax = b using backslash )
x2 = inv(A)*b                         (Method 2: solve Ax = b using inv )
R = ref([A b]); x3 = R(:,n+1);        (Method 3: solve Ax = b using ref )
```

Type **`[norm(x1-x2) norm(x1-x3) norm(x2-x3)]`** to calculate the lengths of the differences of the three solution vectors, and view them side by side. *Record the norm* of each difference vector, in the table below. You can round each mantissa to an integer.

(b) Repeat the instructions above for the matrices in exercises 31, 39 and 41 in Section 2.2. Note that in exercises 39 and 41 the matrix is called *D*, not *A*, so modify your commands with **`D\b,inv(D)*b,`** **`rref([D b])`**.

Norms of the difference vectors

	Exercise 29	Exercise 31	Exercise 39	Exercise 41
x1-x2				
x1-x3				
x2-x3				

2. Fortunately, most matrices that show up in real world problems behave well in numerical calculations, like those in question 1. However, it is worthwhile to see some with large condition numbers, so you know they do exist!

One of the classic types of matrices for which none of the methods above tends to yield a very accurate solution is the type called *Hilbert* matrices. There is a Hilbert matrix of every size, and MATLAB has a special command `hilb` for creating them, since they are frequently used as examples and test cases for new software.

You will understand the pattern in their entries best if you display each decimal as a fraction. Type the following lines to see the 4x4 and 5x5 Hilbert matrices in rational format:

```
format rat, hilb(4), hilb(5)
```

Study these to see the pattern of entries. Think what the first and last row of the 20x20 Hilbert matrix will look like.

3. (MATLAB) Now solve $A\mathbf{x} = \mathbf{b}$ when A is the 20x20 Hilbert matrix, using the three methods above. You already know what this big matrix looks like, so use semicolons as indicated to avoid printing the matrix and the large vectors:

```
A = hilb(20); b = rand(20,1);      (create the 20x20 Hilbert matrix and a random 20x1 column)
x1 = A\b;
x2 = inv(A)*b;
R = rref([A b]); x3 = R(:,21);
```

If you did not see warnings from the calculations `A\b` and `inv(A)*b` , you may be working on a version which keeps more significant digits in its floating point calculations. If thisis the case , repeat the calculations using larger sizes of Hilbert matrices until you do see warnings by typing . so try
`A = hilb(21); b=rand(21,1); A = hilb(22); b=rand(22,1);` etc.

(a) Record the size which finally produces warnings: _____.

After solving $A\mathbf{x} = \mathbf{b}$ with a matrix that causes warnings to appear, type the following line in order to view the three solutions you have created, in a format that makes it fairly easy to compare them:

```
format long e, [x1 x2 x3]
```

(b) The mantissas of the entries in x1 and `x2` may agree in some digits, but this is quite misleading. To see how different these vectors really are, type **`[norm(x1-x2 norm(x1-x3) norm(x2-x3)]`** to calculate the norms of the difference vectors, and record those. Round to integers as before:

Norms of the difference vectors for the Hilbert matrix problem

x1–x2: _____ **x1–x3**: _____ **x2–x3:** _____

Clearly the warnings are justified! Even the difference **x1-x2** is very large, and **x1** and **x2** were calculated with the two professionally coded methods. These large differences in answers calculated by different algorithms reinforce the warnings. It would be very unwise to assume any of these solutions is very accurate.

4. To finish, type **`diary off`**, which will close the file called "`solve`" on your computer. Exit MATLAB and open this file with your favorite text editor. If it contains more than 4 pages, try to reduce its length before printing. For example, erase unnecessary blank lines or big matrices you may have created, and perhaps reduce the font size. Print the file and attach that printout to this project.

Caution: This project is about square invertible coefficient matrices only. Do not use backslash when you want to solve a system in which the coefficient matrix is not square or may not be invertible. The reason is, the command `A\b` will give you an answer but it won't be what you expect! If you want the "general solution" of such a system, use `ref([A b])` and then write the general solution as you learned to do in Chapter 1. When A is not invertible, the answer produced by `A\b` is a type of approximate numerical solution called a "least squares solution." (See Chapter 6.) This type of approximate solution is important for a large number of applications, which is why MATLAB has an easy way to calculate them.