Matlab Project: Ill-Conditioned Systems

Goal: Explore the concepts of the conditioning and nearly singular matrices

Background:

According to the Invertible Matrix Theorem (Theorem 8, page 129), every $n \times n$ (square) matrix A is either *nonsingular* or *singular*. In the first (nonsingular) case, everything "goes right": A is invertible, the columns of A are linearly independent and span $\mathbb{R}^n$, and for every vector $\mathbf{b}$ in $\mathbb{R}^n$, the linear system $A\mathbf{x} = \mathbf{b}$ has exactly one solution (namely, $\mathbf{x} = A^{-1}\mathbf{b}$). In the second, everything "goes wrong" (hence the name "singular", meaning unusual). In theory, this distinction is absolute: a square matrix is either nonsingular or singular, and there is no middle ground. In practice, however, matrices which are nonsingular can be (in a fairly precise sense) close to singular. In this project you will see examples of such matrices and learn how to measure how "bad" they are.

Note: This project will require you to create a MATLAB script file, also known as an **M-file**. This is a file containing a sequence of commands to run, just as you would type them at the MATLAB prompt; it makes it easy to experiment with commands until you get things working the way you want. To create it, use the MATLAB editor (choose **file—new—script**) to create a new M-file. Then put in the commands you want to run, save the file (you can specify where it is saved using the "current folder" window), and run it by typing the filename at the MATLAB command prompt. You can save the script in stages as you write it, testing it step by step until it does exactly what you want. Your final **M-file** must contain the commands you use to produce all results for this project; instructions for precisely what to submit in this file are below.

Step 1: The Hilbert matrix

For each positive integer n, the $n \times n$ Hilbert matrix $H = H_n = [h_{i,j}]$ has entries

$$h_{i,j} = \int_0^1 x^i x^j \, dx = \frac{1}{i+j-1}, \qquad i, j = 1, 2, \ldots, n.$$

For example, the 3×3 Hilbert matrix is

$$H_3 = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}.$$

This matrix arises naturally in the approximation of functions.[1] Unfortunately, the Hilbert matrix has some serious problems.

To see this, use MATLAB to generate the 5×5 Hilbert matrix using the following code:

```
n=5
H = hilb(n)
```

Here we've used a variable **n** for the matrix size so we can change it more easily later on; using the MATLAB function **hilb** simplifies creating the matrix (and may reduce the error in the matrix entries). To generate a linear system $H\mathbf{x} = \mathbf{b}$ for which we know the exact solution, we'll simply *start* with that solution (here, $\mathbf{x}$ is a vector of ones), and construct $\mathbf{b}$ by multiplying H times $\mathbf{x}$:

[1] Specifically, in the least-squares approximation of a function on the interval $[0, 1]$ by a polynomial, using the basis functions $1, x, x^2, \ldots, x^{n-1}$. For details, look under "continuous least-squares" in any numerical analysis book.

```
x=ones(n,1)
b=H*x
```

We can now solve the system $H\mathbf{x} = \mathbf{b}$ using the backslash operator (for help on this, type **help slash**). Since the numerical solution for $\mathbf{x}$ may not be exact (due to roundoff error), we'll use a different symbol $\hat{\mathbf{x}}$ to denote it here:

```
xhat=H\b
```

(note that the slash here is a *backwards* slash). Given the default output of MATLAB, you shouldn't see any difference in **x** and **xhat**. However, if you subtract them to find their difference, you will:

```
diff = xhat - x
```

(note the multiplier that MATLAB outputs before the vector). To measure this difference, we'll use a *norm*, which is a single number related to the size of the entries in a vector. There are many different norms available; here we'll use the so-called "infinity norm", which is just the size of the largest entry (in absolute value).[2] The standard mathematical notation for this norm is $\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$; in MATLAB we can compute it with:

```
err = norm(diff,inf)
```

Check to make sure that the value of **err** matches the largest entry in your vector **diff**. *Write this error value in Table 1 on the answer sheet—use scientific notation with three significant digits (e.g., in the form* 1.23×10^{-8}*).*

The errors in this numerical solution are generated by roundoff error in the matrix entries (for example, the number 1/3 cannot be represented exactly in binary form). These roundoff errors themselves are tiny—the relative error is only about 2×10^{-16}. However, the linear system with the Hilbert matrix is sensitive to any changes (either in the matrix itself or the right-hand side) and thus the error in the numerical solution may be much larger. This sensitivity is measured by the *condition number* of the matrix, which represents the maximum factor by which errors in the input to the linear system can be magnified in its solution (see the box on page 131 and exercise 41 of section 2.3). A linear system (or matrix) is said to be *well-conditioned* if the condition number is "small" (the smallest it can be is 1), and *ill-conditioned* if it is "large". Defining the condition number is beyond the scope of this course (it requires matrix norms), but you can compute it easily in Matlab:

```
condH = cond(H)
```

Write this condition number in Table 1 on the answer sheet (again in scientific notation with three significant digits). You should note that the error **err** you found is less than **condH** times the input error 2×10^{-16}.

To see how things go horribly wrong for the Hilbert matrix, repeat the above calculations for $n = 10$ and $n = 15$ (you may want semicolons after some commands to suppress their output). *Write the resulting errors and condition numbers in Table 1 on the answer sheet (again in scientific notation with three significant digits).* You should find that by $n = 15$ the results of solving this system are worthless (look at the size of the errors and compare them to the solution itself). In

[2] Later in the course (chapter 6) we'll define and use the *standard* or (l_2) norm exclusively.

fact, MATLAB warns you about this, stating that the matrix is "close to singular". In this warning it returns the value **RCOND**, which is the reciprocal of the condition number: if this is close to zero, the condition number is large and the results are probably off (or worthless).

To summarize the situation for the Hilbert matrix, compute the error (**err**) and condition number (**condH**) as defined above for $n = 1, 2, \ldots, 15$ and plot them (on a logarithmic scale) vs. n (for this, use the **semilogy** command instead of **plot** to draw the plot). Label both axes (if you plot both quantities on the same graph, use **legend** to label the curves). *Print and hand in your plot(s) with the answer sheet.*

On the basis of these results, it's safe to conclude that using the Hilbert matrix for n much bigger than 10 is a lousy idea. The matrix is highly ill-conditioned (i.e., close to singular). This is a problem of the matrix itself, not the algorithm used to solve the system (here it's related to the fact that the functions $1, x, x^2, \ldots, x^{n-1}$ all start looking alike on the interval $[0, 1]$). There's no way to fix this problem, short of abandoning it and starting over with a different approach. *This type of difficulty arises in many applied problems, such as remote sensing, inverse problems, and integral equations.*

Step 2: Some well-conditioned matrices

In contrast to the Hilbert case, most matrices are perfectly well-conditioned. To see this, repeat the calculations (but not the plots) of step 1 above for the following two matrices:

- Matrix A: an $n \times n$ matrix with random entries, generated by the commands

  rand('seed',stunum); A = rand(n)

  where **stunum** is *your student number* (e.g., **0123456**) [for unique and reproducible results]
- Matrix B: the $n \times n$ matrix constructed from your A by adding to each diagonal entry the sum of the off-diagonal entries in that row. This can be done easily using the following code:

  ```
  s = sum(A,2) - diag(A)
  B = A + diag(s)
  ```

Since singular matrices are "rare", the matrix A should be reasonably well-conditioned; the matrix B is probably even better-conditioned (since it's *diagonally dominant*). Therefore, you should be able to solve systems using these matrices for large n: try the values $n = 10$, $n = 100$, and $n = 1000$. *Write the resulting errors and condition numbers in Tables 2 and 3 on the answer sheet (again in scientific notation with three significant digits).* Are these matrices well-conditioned?

Step 3: Constructing an ill-conditioned matrix

As a final challenge, construct your own 3×3 matrix C with the following properties:

- no two matrix entries are the same number,
- no matrix entry is zero, and
- the condition number (as reported by **cond(C)** in MATLAB) is between 10^4 and 10^5.

On the answer sheet, write your matrix C, its condition number, and a brief description of how you constructed C.

Completing your M-file:

By the time you're done with the above calculations, you should have written a MATLAB script file (aka **M-file**) to do all of the calculations. Complete this for submission by doing the following:

- At the top of the file put in comment lines (lines starting with **%**) identifying the project and *your* name in the following format:

  % MA339 Project: Condition Numbers
  % Aaron C. Rodgers ⟵ replace with your name

- Remove any commands *other than* those needed to do the above calculations (for example, anything you tried which didn't work).
- Add a few comment lines to identify the main sections of the script.
- Save your script using your *Clarkson login name* as the filename. For example, Aaron would save his script as the file **rodgerac.m**. *Using your Clarkson login name is required*—it avoids having multiple copies of files named `project.m`, and allows me to find your work quickly.
- Verify that it actually works by clearing or restarting Matlab (type *clear* or *exit*) and running it again "cold". This will help catch cases where you've defined something outside the script which the script needs.

Hand in the following:

1. The completed answer sheet. *DO NOT hand in the instructions (I already have a copy).*
2. Your plot(s) from Step 1.
3. A printed copy of your MATLAB script (M-file) containing the commands you used for this project

   Staple your plot(s) and M-file to the answer sheet (in that order).
4. Upload a copy of your MATLAB script to Moodle by the due date (see Moodle for the cutoff time).

Matlab Project: Ill-Conditioned Systems **Name:**_____

ANSWER SHEET

Table 1: Results for the Hilbert matrix H_n:

Size n	error $\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$	condition number cond(H)
5		
10		
15		

Table 2: Results for matrices A and B:

	Results for matrix A		Results for matrix B	
Size n	$\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$	cond(A)	error $\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$	cond(B)
10				
100				
1000				

Step 3:

$C =$ cond(C) =

Explain how you constructed C:

Additional material: Staple your plots(s) and a copy of your **M-file** to this answer sheet.