ESC 407
Computational Methods in Engineering Science
Homework 7/Final Exam
Linear Algebra & Least Squares

Due Date: Wednesday, December 16 at 11:59 p.m.

Problem 1

(20 pts)

Write a MATLAB function named `gaussepp` that solves the system of equations $[A]\{x\} = [B]$ using Gaussian elimination with partial pivoting. The function should take as inputs the $n \times n$ matrix $[A]$ and the $n \times m$ matrix $[B]$, where $[B]$ is matrix consisting of right-hand side columns $\{b\}_i$, $i = 1, \ldots, m$. The outputs should be the $n \times m$ solution array $[X]$, in which column $\{x\}_i$ is the solution to $[A]\{x\} = \{b\}_i$, and the determinant of $[A]$. Write a script called `gauss_test` to test your function on the following $[A]$ and $[B]$

$$\underbrace{\begin{bmatrix} 2 & -6 & -1 \\ -3 & -1 & 7 \\ -8 & 1 & -2 \end{bmatrix}}_{[A]} \underbrace{\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}}_{[X]} = \underbrace{\begin{bmatrix} -38 & -21 & 15 \\ -34 & 29 & 2 \\ -20 & -15 & 15 \end{bmatrix}}_{[B]}.$$

Your function should return $\det[A] = 373$ and

$$[X] = \begin{bmatrix} 4 & 1 & -2 \\ 8 & 3 & -3 \\ -2 & 5 & -1 \end{bmatrix}.$$

Finally, use your function to find $[A]^{-1}$, which should be

$$[A]^{-1} = \begin{bmatrix} -0.013405 & -0.034853 & -0.115282 \\ -0.166220 & -0.032172 & -0.029491 \\ -0.029491 & 0.123324 & -0.053619 \end{bmatrix}.$$

A few notes.

- Don't forget that I gave you pseudocode for most of this.
- To find the largest pivot element in a given column, you can either make use of MATLAB's `max` function or you can use a `for` loop that updates the row p of the max element. If you choose to use `max`, note that if you feed `max` a row or column vector, it will return the algebraically largest element if you only specify one output, if you specify two outputs, it will return the algebraically largest element as the first output and the index of that element in the second. For example,

```
>> [a b] = max([1; 7; 3; -9; 2])
a =
     7
b =
```

```
     2
>> [a b] = max([1 7 3 -9 2])
a =
     7
b =
     2
```

- You can swap rows i and p using the tried and true method taught in every introductory programming course

```
temp = A(i,:);
A(i,:) = A(p,:);
A(p,:) = temp;
```

or you can use a little MATLAB magic

```
A([i p],:) = A([p i],:);
```

- Don't forget to keep track of the number of row swaps so that you get the correct sign on the determinant of $[A]$.

Extra Credit (10%) Recall that if any row or column of $[A]$ is zero, the matrix is singular and therefore the determinant is zero, $[A]^{-1}$ does not exist, and the system $[A]\{x\} = \{b\}$ does not have a unique solution. It is also true that if one row is a linear combination of another, then the matrix is singular. With this in mind, write a function called `singcheck` that is called by `gaussepp` before it starts doing the Gaussian elimination. This will mean adding a third input argument to `gaussepp`, which is a logical variable I called `check`. If `check` is `true`, then `gaussepp` will check if $[A]$ is singular, if `check` is false, it will not. The function should do the following.

(a) Check to see if all the elements in any row or column of $[A]$ are zero or *nearly* zero. In particular, if all elements in any row or column are less than 10^{-13} then $[A]$ should be considered to be singular.

(b) Check to see if any row is a linear combination of, or *nearly* a linear combination of, *any* other row. In particular, if λ is the multiplier that makes E_i equal to E_j, i.e., $\lambda E_i = E_j$, then consider two rows to be linear combinations of one another if every element of $\lambda E_i - E_j < 10^{-11}$. Be clever about your checks—you should not compare two rows more than once.

If `singcheck` does find that $[A]$ is singular, then it should return logical `true` to `gaussepp` and `gaussepp` should report that the matrix $[A]$ is singular or nearly so.

Problem 2

(10 pts)

Write a script called `inverse` that uses MATLAB's `rand` command to generate a 5000×5000 matrix of random numbers evenly distributed on $(-3, 3)$.

(a) Use MATLAB's `inv` command to compute the inverse of the matrix. Use `tic` and `toc` to time the solution.

(b) Use your function to compute the inverse of the same matrix and time it with `tic` and `toc`. Note that it took several minutes to compute the inverse using my function, so be patient.

(c) Compare your $[A]^{-1}$ to MATLAB's by finding the Euclidean norm of the difference between your inverse and MATLAB's.

Problem 3

(20 pts)

The Howe truss shown in Fig. 1 is being used as a roof truss. It has a heavy snow load, wind

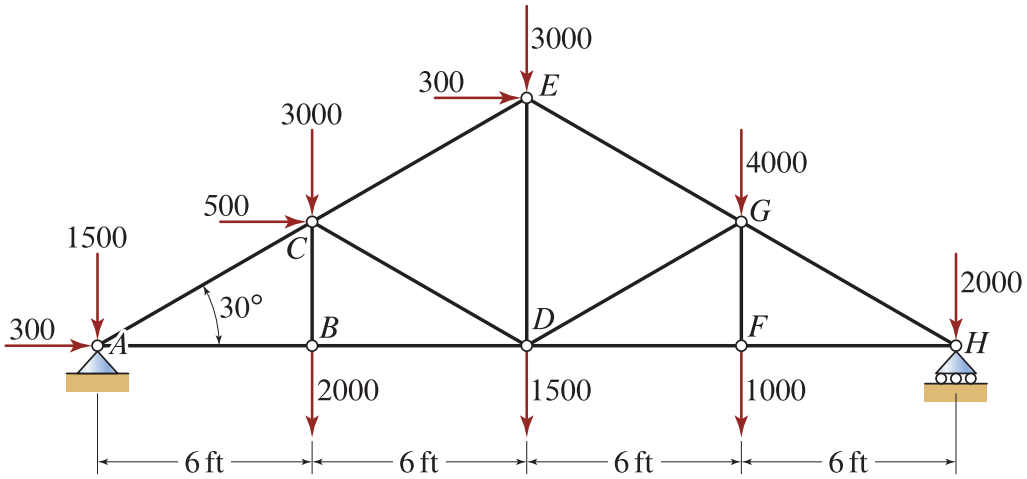

Figure 1: A Howe roof truss. The units on all the loads are in pounds.

loading, and loads due to drywall, insulation, and junk stored in the attic. Your job is to find the load in each member of the truss, as well as the support reactions.

First, write the equilibrium equations containing the forces in all members and the support reactions using the method of joints from statics. Since there are three support reactions and 13 members, there will be 16 unknowns. The 16 equations will come from the two equilibrium equations you can write at each of the eight joints.

Next, write a script called `truss` in which you enter the coefficients of the 16 unknowns in the 16 equations in a matrix $[A]$.* After you have entered $[A]$, enter $\{b\}$ and the do the following.

(a) Use MATLAB's `spy` command to plot the sparsity pattern of $[A]$. There are 256 elements, but only 41 are non-zero! That's 15.6%. The bigger the truss, the sparser the matrix gets. I also created a problem for a Baltimore bridge truss, with 48 equations and 48 unknowns—in that case only 148 out of 2304 elements are non-zero, which is 6.4%.† That matrix is shown in Fig. 2.

*You should definitely check out MATLAB's `equationsToMatrix` command!

†I almost assigned that, but I am quite sure I would have had a revolt if I had. I had fun doing it though!

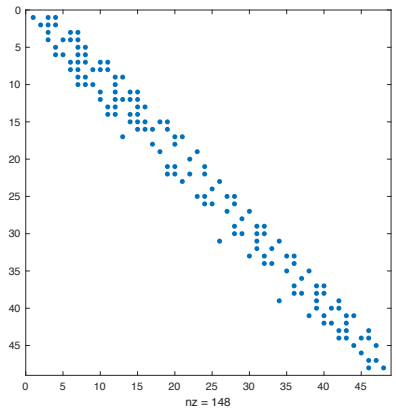

Figure 2: MATLAB's spy command applied to a very sparse matrix.

(b) Use your Gaussian elimination function to solve for all the loads and report the value for each. Don't make me have to figure out which load is which. Here are my results.

```
Load AX =  -1100.0
Load AY =   8591.2
Load AB =  13082.4
Load AC = -14182.5
Load BC =   2000.0
Load BD =  13082.4
Load CD =  -5288.7
Load CE =  -9471.1
Load DE =   6644.3
Load DF =  12832.4
Load DG =  -5000.0
Load EG =  -9817.5
Load FG =   1000.0
Load FH =  12832.4
Load GH = -14817.5
Load HY =   9408.8
```

(c) Use MATLAB's built-in linear system solver to solve for the loads and compare your solution to MATLAB's by finding the Euclidean norm of the difference between your solution and MATLAB's. Since both solutions are vectors, you are literally finding the distance between them in 16-dimensional space.

(d) Now use your function to find $[A]^{-1}$ and then show that the solution for the loads can also be found using $\{x\} = [A]^{-1}\{b\}$. Again use the Euclidean norm to compare the solution for the loads that you found using Gaussian elimination with the solution you just found using $[A]^{-1}\{b\}$.

Problem 4

(15 pts)

Write a MATLAB function named `polyfit` that finds the least squares polynomial of degree n through m data points (x_i, y_i), $i = 1, 2, \ldots, m$. The function should take as inputs the data points

as x and y vectors and the degree of the polynomial you desire to fit to the data. The output should be a vector containing the $n+1$ coefficients of the polynomial in the order from a_n to a_0*, the coefficient of determination r^2 for the fit, and the correlation coefficient r for the fit. Your function should check that n is a positive integer, that x and y are the same length, and that $n < m - 1$.

(a) Write a script called `polyfit_test` to test your function on the data shown in Table 1. Your function should return $a_2 = 0.8437$, $a_1 = 0.8641$, $a_0 = 1.0052$, where $y = a_2x^2 + a_1x + a_0$; $r^2 = 0.999853$, and $r = 0.999926$.

Table 1: Data for polynomial least squares example done in class.

i	1	2	3	4	5
x_i	0.00	0.25	0.50	0.75	1.00
y_i	1.0000	1.2840	1.6487	2.1170	2.7183

(b) Plot the data and the least squares polynomial on the same plot. You can make us of MATLAB's `polyval` command to do this.

Problem 5 (15 pts)

Write a script called `solve_time` that calls your `gaussepp` function to solve a series of linear systems with $[A]_{n\times n}$, $\{b\}_{n\times 1}$, and $n = 1, 2, 3, \ldots, 500$. Both $[A]$ and $\{b\}$ should contain uniformly distributed random numbers in $(0, 1)$. You should generate $[A]$ and $\{b\}$ once and then choose progressively larger parts of $[A]$ and $\{b\}$. So for $n = 1$, you will use a_{11} and b_1, for $n = 2$, you will use the square matrix consisting of a_{11}, a_{12}, a_{21}, and a_{22}, along with b_1 and b_2, etc.

(a) Compute the amount of time it takes for each of the 1000 solutions and store the times in an array.† Plot the solution time versus n.

(b) Use your `polyfit` function to fit a few polynomials to the data you generate in part (a). Keep increasing the order of the polynomials and, after each fit, compute the least squares error or the sum of the square of the residuals, which is

$$s_r = \sum_{i=1}^{m} [y_i - P_n(x_i)]^2,$$

until the approximate relative relative error in s_r drops below 1×10^{-5}. Does the solution time behave as expected?

*This will allow you to use MATLAB's `polyval` command to evaluate the polynomial without altering the output of your function.

†Takes about 6 minutes on my machine, so be patient. It is helpful to have it spit out something for every 100th solution so that you know something is happening.

Problem 6 (20 pts)

An experiment you are doing produces the data found in the file `raw_data.dat` that you can find on Canvas.* When you plot the data, you see what is found in Fig. 3. You need to fit a

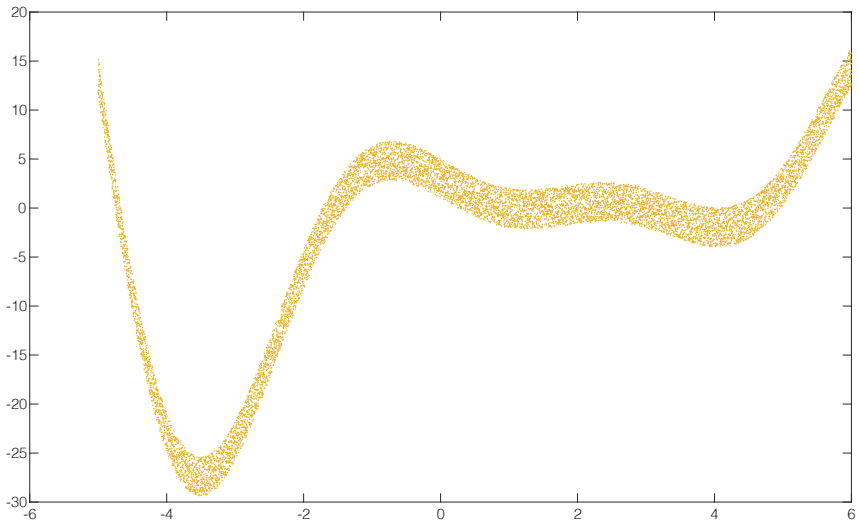

Figure 3: Raw date from an experiment.

curve to this data so that you can have an analytic function with which to predict the behavior of your system, allow you to take derivatives, etc. You quickly realize that you cannot fit a polynomial through all $m = 11{,}001$ points since it would oscillate wildly and would not be the least bit useful. Therefore, you decide to fit a least squares polynomial through the data. With this in mind, here is your task.

(a) Write a script called `polyfit_run` that reads in the raw data, which contains the abscissas (x values) in the first column and the ordinates (y values) in the second. Apply `polyfit` to the data with increasingly higher order polynomials, starting with a straight line. That is, fit polynomials of degree n, with $n = 1, 2, 3, \ldots$.

(b) You should continue generating higher order polynomial fits until the relative relative error in the sum of the squares of the residuals s_r drops below 10^{-5}.†

(c) On a single plot, your script should plot the original data and *all* of the polynomial approximations to the data. It should report the final degree of the polynomial that satisfies the approximate relative least squares error and the final value of that error.

(d) In a separate window, plot the coefficient of determination for each polynomial fit versus the order of the fit. That is, plot r^2 versus n.

(e) Finally, your script should plot, in a separate figure window, the original data and the final polynomial on top of that data.

*Use `dlmread` to read in the data.

†I tried going to 10^{-6} and the degree of polynomial required needed to be greater than 150, and it took forever to run, so I decided to stop at 10^{-5}.

Epilogue

It has been a great semester! I had so much fun working on the assignments with you this semester. I wish you all the best on your final exams.