Some comments before you begin: Problems 3 & 4 are intended to sharpen your skills with manipulating matrices and vectors in MATLAB. You should be able to do every problem without using one for or while loop. In fact, you will not receive full credit if you use a loop in any of these problems! In case you want to brush up on your skills, I recommend reading chapter 2 of “Learning MATLAB” by Toby Driscoll (see the course webpage for a link). Some of the problems below are adapted (or straight copied) from the exercises at the end of chapter 2 of this book.

1. **(Maximizing a function, 5 pts)** Use MATLAB fminbnd to find the maximum of the function

\[ f(x) = (\sin(x))^6 \tan(1 - x)e^{5x} \]

over the interval \([0, 1]\). Report the maximum value of \(f\) and the what value of \(x\) gives this maximum. Plot the function and mark the maximum with an open red circle.

2. **(Distance between planets, 15 pts)** The orbits of Mercury and Earth can be (ideally) parameterized with respect to time as follows:

- **Mercury**

  \[
  x_m(t) = -11.9084 + 57.9117 \cos(2\pi t/87.97) \\
  y_m(t) = 56.6741 \sin(2\pi t/87.97)
  \]

- **Earth**

  \[
  x_e(t) = -2.4987 + 149.6041 \cos(2\pi t/365.25) \\
  y_e(t) = 149.5832 \sin(2\pi t/365.25)
  \]

The units on the position are in \(10^6\)km and the units on time are days. The coordinate system has been arranged so that the sun is at the center.

(a) Use the MATLAB function fminbnd to determine a time between 0 and 1000 for which the distance between the Earth and Mercury is minimal. Report this time and plot the two orbits together with the positions when the distance is at a minimum clearly marked on the curves. Is the time you found the global minimum over the interval \(0 \leq t \leq 1000\), or just a local minimum? Explain how you determined this with, for example, some kind of plot.

(b) Repeat part (a), but now find a time for which the distance is maximal. Report this value and produce a similar plot to part (a) with the positions marked.

3. **(Manipulating matrices, 20pts)** There are many useful functions in MATLAB for manipulating matrices and vectors, like the diag function from the previous problem. Some other examples include the tril, triu, reshape, and repmat functions. The tril and triu can be used to return any portion of the lower and upper part of a matrix \(A\), respectively. The reshape command allows you to change the dimensions of a matrix from \(n\)-by-\(m\) to \(p\)-by-\(q\), provided \(mn = pq\). For example,

\[ A = \text{reshape}(1:25,[5 5]) \]

produces the matrix

\[
A = \begin{bmatrix}
1 & 6 & 11 & 16 & 21 \\
2 & 7 & 12 & 17 & 22 \\
3 & 8 & 13 & 18 & 23 \\
4 & 9 & 14 & 19 & 24 \\
5 & 10 & 15 & 20 & 25
\end{bmatrix}
\]

The repmat command allows you to create a matrix with a repeating pattern. For example,
\[ a = 1:2:9; \]
\[ A = \text{repmat}(a,[5 1]) \]

produces the matrix

\[
A = \begin{bmatrix}
1 & 3 & 5 & 7 & 9 \\
1 & 3 & 5 & 7 & 9 \\
1 & 3 & 5 & 7 & 9 \\
1 & 3 & 5 & 7 & 9 \\
1 & 3 & 5 & 7 & 9
\end{bmatrix}
\]

Note that this is just the row vector \( a \) repeated 5 times.

Use the above functions, or any combination of them, in the problems below. Your code may not use \( \text{any} \) loops and you may not just input these matrices directly in MATLAB. However, you may use any operations such as addition, subtraction, and multiplication. You may also use the additional MATLAB functions \( \text{diag} \), \( \text{eye} \), and \( \text{transpose} \). The code you produce should require no more than two lines and should generalize to any size matrix with the same pattern.

(a) Produce the matrices

\[
\begin{bmatrix}
0 & 6 & 11 & 16 & 21 \\
0 & 0 & 12 & 17 & 22 \\
0 & 0 & 0 & 18 & 23 \\
0 & 0 & 0 & 0 & 24 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}, \quad
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
1 & 2 & 0 & 0 & 0 \\
1 & 2 & 3 & 0 & 0 \\
1 & 2 & 3 & 0 & 0
\end{bmatrix}, \quad
\begin{bmatrix}
0 & 6 & 11 & 16 & 21 \\
0 & 0 & 12 & 17 & 22 \\
1 & 0 & 0 & 18 & 23 \\
1 & 2 & 0 & 0 & 24 \\
1 & 2 & 3 & 0 & 0
\end{bmatrix}
\]

(b) Produce the matrices

\[
\begin{bmatrix}
3 & 3 & 3 & 3 & 3 \\
6 & 6 & 6 & 6 & 6 \\
9 & 9 & 9 & 9 & 9 \\
12 & 12 & 12 & 12 & 12 \\
15 & 15 & 15 & 15 & 15
\end{bmatrix}, \quad
\begin{bmatrix}
25 & 20 & 15 & 10 & 5 \\
24 & 19 & 14 & 9 & 4 \\
23 & 18 & 13 & 8 & 3 \\
22 & 17 & 12 & 7 & 2 \\
21 & 16 & 11 & 6 & 1
\end{bmatrix}, \quad
\begin{bmatrix}
0 & 20 & 15 & 10 & 5 \\
24 & 0 & 14 & 9 & 4 \\
23 & 18 & 0 & 8 & 3 \\
22 & 17 & 12 & 0 & 2 \\
21 & 16 & 11 & 6 & 0
\end{bmatrix}, \quad
\begin{bmatrix}
-1 & 20 & 15 & 10 & 5 \\
24 & -1 & 14 & 9 & 4 \\
23 & 18 & -1 & 8 & 3 \\
22 & 17 & 12 & -1 & 2 \\
21 & 16 & 11 & 6 & -1
\end{bmatrix}
\]

(c) Produce the matrices

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 4 & 4 & 4 & 4 \\
1 & 4 & 9 & 9 & 9 \\
1 & 4 & 9 & 16 & 16 \\
1 & 4 & 9 & 16 & 25
\end{bmatrix}, \quad
\begin{bmatrix}
0 & 4 & 9 & 16 & 25 \\
-4 & 0 & 9 & 16 & 25 \\
-9 & -9 & 0 & 16 & 25 \\
-16 & -16 & -16 & 0 & 25 \\
-25 & -25 & -25 & -25 & 0
\end{bmatrix}, \quad
\begin{bmatrix}
-1 & 4 & 9 & 16 & 25 \\
4 & -4 & 9 & 16 & 25 \\
9 & 9 & -9 & 16 & 25 \\
16 & 16 & 16 & -16 & 25 \\
\end{bmatrix}
\]

(d) Produce the matrix (note the block pattern)

\[
\begin{bmatrix}
4 & 25 & 64 & 4 & 25 & 64 \\
9 & 36 & 81 & 9 & 36 & 81 \\
16 & 49 & 100 & 16 & 49 & 100 \\
4 & 25 & 64 & 4 & 25 & 64 \\
9 & 36 & 81 & 9 & 36 & 81 \\
16 & 49 & 100 & 16 & 49 & 100
\end{bmatrix}
\]

4. (Toeplitz matrices, 10pts)
(a) The two matrices $A$ and $B$ below are examples of what are called Toeplitz matrices, which are matrices which are constant along their diagonals. These matrices occur quite often in applications. Read the online help for the MATLAB function `toeplitz` and use this function to produce $A$ and $B$ below. Note: use vector concatenation and the function `zeros` instead of typing all those zeros. Your code should take one line to produce $A$ and one to produce $B$.

$$
A = \begin{bmatrix}
2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
\end{bmatrix},
$$

$$
B = \begin{bmatrix}
-2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \\
\end{bmatrix}
$$

This should again require no loops and exactly one line.

(b) Now use `toeplitz` to create the matrices

$$
A = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
-1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
0 & -1 & 0 & 1 & 2 & 3 & 4 & 5 \\
0 & 0 & -1 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & -1 & 0 & 1 & 2 & 3 \\
0 & 0 & 0 & 0 & -1 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
\end{bmatrix},
$$

$$
B = \begin{bmatrix}
1 & -4 & 7 & -10 & 13 & -16 & 19 \\
-3 & 1 & -4 & 7 & -10 & 13 & -16 \\
5 & -3 & 1 & -4 & 7 & -10 & 13 \\
-7 & 5 & -3 & 1 & -4 & 7 & -10 \\
9 & -7 & 5 & -3 & 1 & -4 & 7 \\
-11 & 9 & -7 & 5 & -3 & 1 & -4 \\
13 & -11 & 9 & -7 & 5 & -3 & 1 \\
\end{bmatrix}
$$

You may not use `triu` or `tril` to construct $A$ and your code needs to use one line. Your code for $B$ should not require explicitly typing a vector and should not use any loops. It should be written so that only one or two small changes are required to produce a similar $B$ of any size.

5. (Rotation matrices, 10 pts) The matrix

$$R = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta \\
\end{bmatrix}$$
is called a rotation matrix since it can be used for rotating vectors (or points) in the \( xy \)-plane counter-clockwise through an angle \( \theta \) about the origin. If \( (x, y) \) is a point in the plane, and \( x = [x \ y]^T \) is a vector containing the coordinates of this point, then the entries of the vector \( \tilde{x} = R \tilde{x} \) will contain the coordinates of the rotated point.

Consider the polygon with vertices \((0, 0), (1, 0), (7/11, 1), (1/2, 1/3), \) and \((3/10, 1)\). Make a plot of this polygon in MATLAB using the \texttt{fill} function. Rotate this polygon by an angle of 103 degrees using matrix-vector multiplication in MATLAB with an appropriately chosen rotation matrix \( R \). Make another plot of the rotated polygon using \texttt{fill}.

6. (Timing a linear solve, 25pts) In this problem, you will investigate the cost of solving linear systems and compare your results to the theoretical computational cost we have discussed in class.

Before starting this problem, you will need the following information about the laptop or PC you are working on.

- The amount of RAM in your computer. Typical values are between 4 and 12 GB (gigabytes).
- The clock speed of your PC or laptop. A typical value is 2.5 GHz (gigahertz).

(a) **Experimental setup.** Determine a matrix size that you can comfortably fit into your available RAM. For example, if you have a 4 GB machine, you should be able to comfortably store a matrix that occupies about 800MB. Store this value in a variable “\( \text{M} \)”. Use the following information to compute a maximum matrix dimension \( N \) that you can store in \( \text{M} \) megabytes of memory.

- A megabyte has 1024 kilobytes
- A kilobyte is 1024 bytes
- A floating point number is 8 bytes.
- An \( N \times N \) matrix contains \( N^2 \) floating point numbers.

Call the \( N \) you compute ’\( \text{nmax} \)’ (\( N_{\text{max}} \)).

(b) **Calibrate the timing experiment.** We need to “calibrate” our experimental set up by timing an operation whose flop count is easy to compute. Create two random matrices \( A \) and \( B \) each of size \( N_{\text{max}} \times N_{\text{max}} \). Using the MATLAB functions \texttt{tic} and \texttt{toc}, determine how much time (seconds) it takes to compute the product \( AB \). Determine the number of floating point operations (additions and multiplications) it takes to compute the \( N_{\text{max}} \times N_{\text{max}} \) matrix-matrix product (see the Linear Algebra lecture slides). Use this number to estimate the number of floating point operations per second (’flops’) your computer can carry out. Call this flop rate ’\( \text{flops} \)’. Compare this number to the theoretical number obtained using your computer’s clock speed. The following information might be useful.

- A gigahertz is \( 10^9 \) hertz.
- A hertz is equal to one clock cycle.
- A typical microprocessor (CPU) can compute 4 flops per clock cycle.

If you have a dual or quad core machine, you may also want to investigate whether Matlab is making use of multiple cores.

(c) **Time a dense linear solve.** Create an integer sequence of values \( \text{Nvec} \) between, say, \( N = 100 \) and the \( N_{\text{max}} \) you found above. To generate this sequence, it is a good idea to use the \texttt{logspace} command.

Using a \texttt{for} loop, loop over the entries \( N_i \) of the sequence. In each pass through the loop, create a random matrix \( A \) of size \( N_i \times N_i \), and a random right hand side vector \( b \) of size \( N_i \times 1 \). Then, using \texttt{tic} and \texttt{toc}, time how long it takes to solve the system \( Ax = b \) using the backslash operator. Store this time as the \( i \)th entry in a vector ’\( \text{lutimes} \)’.

(d) On a \texttt{log-log} plot, plot the time values you found above (stored in \texttt{lutimes}) verses the \( N_i \) in your sequence of \( N \) values. On the same set of axis, plot the curve of the theoretical
Figure 1: Timing plot for problem 3. The x-axis is the matrix size $N$ ($N \times N$ matrix), and the y-axis is the time (in seconds) as measured by `tic` and `toc` for solving a linear system of size $N$.

time estimated by the operation count we discussed in class. Be sure to use the flop rate 'flops' you computed above to get a time (in seconds) from an operation count. The two plots should be very close. Be sure to label your plots. Use the Matlab `legend` command to identify each curve that you get. Your plot should look something like the figure shown in Figure 1.

(e) Visit the website [http://www.top500.org](http://www.top500.org) and answer the following questions.

- Using the value from the 'Linpack Benchmark', what is the performance of the world’s fastest supercomputer, measured in GFlops (= 10$^9$ flops)?
- How much faster is the world’s fastest computer than your computer or laptop?
- How recently would your computer have made it on the Top 500 list of the world’s fastest computers?
- When can we expect to see an exaflop machine?

The 'Linpack Benchmark' solves a dense linear system of equations, very much like what you have done for this problem.

7. (Heat transfer, 30pts) Imagine that we have a metal rod of length $L$ (m) whose temperature at each end is held fixed at 20°C (293.15 K) and which is being heated by a flame held at the midpoint. We can approximate the steady state temperature at equally spaced points along the rod using the following model. We divide the rod into $N$ intervals of equal length $h$ as shown in the figure below. This partitioning of the rod gives us $N+1$ equally spaced points

$$x_j = jh, \quad h = \frac{L}{N}, \quad j = 0, 1, 2, \ldots N$$

The steady state temperature $T_j$ (in Kelvin) at interior points $x_j$ in the rod can be modeled as

$$T_j = \frac{T_{j+1} + T_{j-1}}{2} + \frac{h^2}{2\kappa} f(x_j), \quad j = 1, 2, \ldots, N-1,$$

where $T_0 = T_N = 293.15K$. The flame $f(x)$ is given by

$$f(x) = S \exp \left(-\left(\frac{x-L/2}{\varepsilon}\right)^2\right)$$
where $S$ is the flame heating rate ($W/m^3$), and $\varepsilon$ is a scaling factor specifying the flame width ($m$). The thermal conductivity, $\kappa$, is given by

$$\kappa = \rho c_p \beta$$

where $\beta$ is the thermal diffusivity of the metal ($m^2/s$), $c_p$ is the heat capacity of the metal ($J/(kg \cdot K)$), and $\rho$ is the density of the metal ($kg/m^3$).

The figure below gives a schematic of this problem setup.

---

(a) Using either the tridisolve (from the NCM course materials) or spdiags function and backslash ($\backslash$) in MATLAB, set up a linear system to solve for the steady state temperature distribution in the rod of length $L = 1$ and $N = 201$. This linear system should be of size 200-by-200 to compute the solution at all the interior nodes.

(b) Solve the system for a heat rate $S = 2 \times 10^4$, a scaling factor $\varepsilon = 6 \times 10^{-2}$, a thermal diffusivity of $\beta = 2 \times 10^{-3}$, a heat capacity of $c_p = 210$, and a density of $\rho = 12$.

Check your solution: The temperature $T_2$ is 295.249566 K and $T_3$ is 296.299349 K.

(c) Plot the resulting temperature distribution as a function of $x$. Be sure to add labels and a legend to your graph.

(d) How close to the center of the rod could you touch with your finger without getting burned? Give the value of $x$ and indicate where this point is on the graph from the previous part. Note that you need to find the approximate temperature at which skin burns (be reasonable here, and assume your skin will burn in 1 second or less of being placed on the rod).