matlab language skills and debugging

Aarthi Sridhar, Ph.D., Jenni Rinker, Ph.D., Kim Bourne, Ph.D., and Henri P. Gavin, Ph.D.
June 3, 2024

1 Orientation

MATLAB and its open-source derivative Octave provide a programming language and computing environments that are efficient at calculations involving vectors and matrices. As a computing environment, MATLAB / Octave allow users to interactively carry out calculations by manually entering code or calculations in the Command window. The Command window looks like a terminal window with a >> prompt. As a programming language, matlab allows users to write algorithms into scripts or functions saved as .m-files. The Workspace window conveniently displays the dimensions and values of variables available within the Command window.

Code in the matlab language looks like matrix equations. For example, if A and B are matrices of compatible dimensions, they can be multiplied just like this ... \( C = A*B \). This and other features of the matlab language present opportunities for writing terse code, and often eliminate the need for writing for-loops, especially when computing with matrices.

2 Download and Install

Octave for Windows , Octave for OSX , Octave for Linux , or MATLAB

2.1 Set-up the Octave Environment

- Edit > Set Path > Add Folder > Folder with Subfolders ...
  navigate to your m-files folder and add it to the path ... Save > Close
- Window > for now, uncheck: Show Command History, Show File Browser, Show Workspace
  CTRL+0: switch to the Command window. CTRL+4: switch to the Editor window.
- Edit > Preferences > General , Command , Editor ... adjust comment, font and colors

2.2 Set-up the MATLAB Environment

- Home > Environment > Set Path > Add with Subfolders ...
  navigate to your m-files folder and add it to the path ... Save
- Home > Environment > Preferences > Numeric Display > Compact
- Home > Environment > Preferences > MATLAB > Appearance > Theme > Dark
- How to change the MATLAB layout (recommended)

2.3 Other editors

Code in the matlab language is written in “.m-files,” which are plain text files and may be edited outside of MATLAB or Octave. You may prefer to edit your .m-files in notepad++, Sublime, VS Code, or some other text editor. This VS code plugin allows you to edit and run your .m-files from within VS Code, without first opening MATLAB.
3  matlab language tutorial videos

- https://www.youtube.com/watch?v=TqwSLc3bObg (94 minutes)
- https://www.youtube.com/watch?v=T_ekAD7U-wU (266 minutes)

Refer to the indexed Time Stamps in the descriptions.

4  Useful tips

1. **Comment your code.** Start one-line comments with `%` or `...`. Text and code following the `%` or `...` and going to the end of the line is not executed. It’s fine and good to comment every line.

   Blocks of text or code starting with `%{ as the first characters of the line at the top of the block and ending with `%} as the first characters of the line at the bottom of the block are not executed.

2. **Suppress output.** End every line with a `;` to keep intermediate results from being displayed to the MATLAB Command window. To display the value of any variable of interest to the Command window, remove the `;` from the end of the line.

3. **Re-run a previous command.** Scroll through your previous command-line inputs by selecting the Command window and hitting the up arrow and the down arrow on your keyboard.

4. **Clear the Command Window.** Type `clc` into the Command window.

5. **Clear variables from the Workspace.** Type `clear <varname1> <varname2> ...` to clear certain variables from the workspace, and the memory allocated to them. Doing so, you will see these variables disappear from the Workspace window. Clear all variables by typing `clear all`.

6. **Break up long lines of code in your .m-files.** If a line of code is longer than 80 characters (one page width of text), you may continue an expression onto subsequent lines by splitting the line with an ellipsis (`. . .`). This is not a strict rule, and there are some instances in which a line of code should be very long. Note that when splitting a line with an ellipsis, no blank space may follow the ellipsis.

   An ellipses is unnecessary with matrices; you may enter matrix row by row on separate lines like this ...

   $$ M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} $$

7. **Abort.** Hit Ctrl-C in the command window to stop any calculation for any reason.
5 Assigning values to scalar, vector, matrix, and cell variables

Use square brackets \([\ ]\) to begin and end vectors and matrices. Commas or blank space separate elements in a single row. Semicolons or new-lines separate rows.

1. Scalar \(a = 4\)

2. Row vector \(r = [1\ 2\ 3]\) or \(r = [1\ ,\ 2\ ,\ 3]\) or \(r = [1\ ;\ 2\ ;\ 3]'\)

3. Column vector \(c = [1\ 2\ 3]'\) or \(c = [1\ ;\ 2\ ;\ 3]\) or \(c = [1\ ,\ 2\ ,\ 3]'\)

4. Matrix \(M = [1\ 3\ 5\ ;\ 2\ 4\ 6]\)

5. Blank space can have meaning.
   - \(a = [1\ 2\ 3-1]\) is the \((1 \times 3)\) vector \([1\ 2\ 2]\)
   - \(a = [1\ 2\ 3\ -1]\) is the \((1 \times 4)\) vector \([1\ 2\ 3\ -1]\)
   - \(a = [1\ 2\ 3-1]\) is the \((1 \times 3)\) vector \([1\ 2\ 2]\)
   - \(a = [1\ ,\ 2\ ,\ 3\ ,\ -1]\) is the \((1 \times 4)\) vector \([1\ 2\ 3\ -1]\)

6. Cell Array (a set of dissimilar quantities, such as a set including character strings and numerical quantities)
   - \(S = \{ 'hello', [3\ 5\ ;\ 7\ 11], [2\ 4\ 6\ 8], 'appreciate' \}\)
   - \(S\{5\} = 'today'\)

7. Dimensions of variables are shown in the Workspace window or can be found by Command window commands:
   - \([nr,nc] = \text{size}(X)\) ... \(nr = \text{size}(X,1)\) ... \(nc = \text{size}(X,2)\) ... \(n = \text{length}(x)\)

6 Shortcuts to assign values to vectors and matrices

1. Define vectors of uniformly spaced values using the colon
   - \(v = [\text{start} : \text{increment} : \text{end}]\) represents a row vector from \text{start} up to and including \text{end} with \text{increment} between sequential values.
   - For example \(v = [3 : 3 : 24]\) represents \(v = [3\ 6\ 9\ 12\ 15\ 18\ 21\ 24]\) and \(v = [2 : 10]\) represents \(v = [2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10]\) and \(v = [10 : -1 : 2]\) represents \(v = [10\ 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2]\)

2. Define vectors using linspace and logspace
   - \(v = \text{linspace}(\text{start}, \text{end}, N)\) represents a row vector of \(N\) values from \text{start} up to and including \text{end} with a uniform increment between values.
   - \(v = \text{logspace}(\text{start}, \text{end}, N)\) represents a row vector of \(N\) values from \(10^{\text{start}}\) up to and including \(10^{\text{end}}\) with log-increment between values.
   - For example \(v = \text{linspace}(1,4,5)\) represents the row vector \(v = [1.00, 1.75, 2.50, 3.25, 4.00]\) and \(v = \text{logspace}(-1,1,3)\) represents the row vector \(v = [0.1, 1.0, 10.0]\)
3. Common Values
The matlab language provides a number of commonly used built-in numerical values, such as \( \pi \) (3.14159...), \( e \) (2.71828...), \( i \) or \( j \) (\( \sqrt{-1} \)), \( \text{Inf} \) (\( \infty \), 1/0), \( \text{eps} \) (2.22 \( \times \) 10^{-16}), and \( \text{NaN} \) (not a number, 0/0). There is no need to write \( \pi = 3.14159 \). Avoid re-assigning other values to built-in variables.

4. Vector or Matrix of zeros
\( \text{zeros}(m,n) \) creates an \( m \)-by-\( n \) matrix of zeros

5. Vector or matrix of ones
\( \text{ones}(m,n) \) creates an \( m \)-by-\( n \) matrix of ones

6. Identity matrix
\( I = \text{eye}(n) \) creates an \( n \)-by-\( n \) identity matrix.
The identity matrix is all zeros except for the main diagonal which has values of 1.

7. Diagonal matrix
\( D = \text{diag}(v) \) creates a diagonal matrix whose diagonal elements are the elements of the vector \( v \)
\( D = \text{diag}(v,1) \) creates a diagonal matrix whose first diagonal above the main diagonal are the elements of the vector \( v \)
\( D = \text{diag}(v,-2) \) creates a diagonal matrix whose second diagonal below the main diagonal are the elements of the vector \( v \)

8. Matrices of Random Values
\( M = \text{rand}(m,n) \) creates a \( m \)-by-\( n \) matrix of random values uniformly distributed between 0 and 1.
\( M = \text{randn}(m,n) \) creates a \( m \)-by-\( n \) matrix of random values normally distributed with mean of 0 and standard deviation of 1.
\( M = \text{randi}(\text{min, max, m, n}) \) creates a \( m \)-by-\( n \) matrix of random integers uniformly distributed between \( \text{min} \) and \( \text{max} \).

9. Assigning values to other kinds of matrices
Other built-in shortcuts used to assign values to certain kinds of matrices include: \( \text{tril} \), \( \text{triu} \), \( \text{pascal} \), \( \text{magic} \), \( \text{hadamard} \), \( \text{compan} \), \( \text{vander} \), \( \text{toeplitz} \), and \( \text{hankel} \).
You can read about these, and any other function, by typing (for example) \( \text{help tril} \) at the Command window prompt >>

7 Vector and Matrix operations
1. Select elements from vector
\( v(i) \) selects the \( i \)th entry of \( v \), where \( 1 \leq i \leq \text{length of } v \)
\( v(\text{idx}) \), where \( \text{idx} \) is a vector of integers from 1 to the length of \( v \), selects the elements of \( v \) whose indices correspond to \( \text{idx} \). For example,
if \( v = [5 1 3 7] \) and \( \text{idx} = [1 4 3] \) then \( v(\text{idx}) \) represents \([5 7 3]\)

2. Select to end of vector
\( v(i{:}\text{end}) \) selects the \( i \)th entry to the last entry of \( v \).
\( v(i{:}\text{end-1}) \) selects up to the second-to-last entry of \( v \).
This works for any index number from 1 up to the length of the vector.
3. **Select elements from matrix**
   This follows the same principle as vectors, but indices for row and column are separated by a comma.
   \( M(i,j) \) is the scalar in the \( i \)th row and \( j \)th column of matrix \( M \).
   \( M(:,i) \) represents the vector in the \( i \)th row of matrix \( M \), a *row* vector.
   \( M(i,:) \) represents the vector in the \( j \)th column of matrix \( M \), a *column* vector.
   If \( \text{idc} = [3:2:9] \) then \( M(:,\text{idc}) \) represents the odd column of matrix \( M \) from column 3 to column 9.
   If \( \text{idr} = [2:2:4] \) and \( \text{idc} = [3:2:9] \) then \( M(\text{idr},\text{idc}) \) represents the values of \( M \) in the even rows from row 2 to row 4 and the odd columns from column 3 to column 9.

4. **Transpose**
   If \( v \) is a row vector, \( v' \) is a column vector. If \( v \) is a column vector, \( v' \) is a row vector.
   If \( M \) is a \( m \)-by-\( n \) matrix then \( M' \) represents a \( n \)-by-\( m \) matrix.
   Row \( i \) of \( M' \) is the transpose of column \( i \) of \( M \).
   If some values in \( v \) or \( M \) are complex, the transpose can also involve taking the complex-conjugate of the values in \( v \) or \( M \). If \( v \) is a complex row (column) vector, then \( v' \) represents the column (row) vector containing the complex conjugates of the elements of \( v \). If \( v \) is a complex row (column) vector, then \( v.' \) represents the the column (row) vector with the same elements of \( v \).
   If \( v \) and \( M \) are real, the transposes \( v' \) and \( v.' \) are equivalent, and the transposes \( M \) and \( M.' \) are equivalent.

5. **Tiling Matrices**
   If \( A = \text{zeros}(3); B = \text{ones}(3,4); \) and \( D = \text{randn}(4); \) then \( M = [A, B ; -B', D] \) is a 7-by-7 matrix in which the first three rows of the first three columns are all zero, the first three rows of columns 4 through 7 are all ones, the last four rows of the first three columns are all negative ones, and the last four rows of the last four columns are random.

\[
M = \begin{bmatrix}
0.00 & 0.00 & 0.00 & 1.00 & 1.00 & 1.00 & 1.00 \\
0.00 & 0.00 & 0.00 & 1.00 & 1.00 & 1.00 & 1.00 \\
0.00 & 0.00 & 0.00 & 1.00 & 1.00 & 1.00 & 1.00 \\
-1.00 & -1.00 & -1.00 & -0.21 & 2.10 & 0.89 & -0.18 \\
-1.00 & -1.00 & -1.00 & 0.78 & -0.13 & -0.90 & -0.17 \\
-1.00 & -1.00 & -1.00 & 1.51 & -1.10 & -0.68 & 0.65 \\
-1.00 & -1.00 & -1.00 & -0.29 & -0.11 & -1.57 & -1.08 
\end{bmatrix}
\]

6. **Addition, Subtraction, Multiplication, and Division of a matrix (or a vector) with a scalar**
   If \( M \) is a matrix and \( s \) is a scalar, then:
   \( A = M+s \) is a matrix of the sum of each element of \( M \) and \( s \) ... \( A(i,j) = M(i,j)+s \)
   \( A = M-s \) is a matrix of the difference of each element of \( M \) and \( s \) ... \( A(i,j) = M(i,j)-s \)
   \( A = M*s \) is a matrix of the product of each element of \( M \) and \( s \) ... \( A(i,j) = M(i,j)*s \)
   \( A = M/s \) is a matrix of the ratio of each element of \( M \) over \( s \) ... \( A(i,j) = M(i,j)/s \)
7. Vector Addition and Subtraction
If \( u \) and \( v \) are vectors the same dimension (both row vectors or both column vectors, and of the same length) then \( \mathbf{a} = \mathbf{u} + \mathbf{v} \) represents a vector of the sum of each element of \( u \) with the corresponding element of \( v \), and has the same dimensions as \( u \) and \( v \)

\[ a_i = u_i + v_i \]

If \( u \) and \( v \) are both column vectors (or both row vectors) of different dimensions, they cannot be added or subtracted, and MATLAB / Octave will return the error: \textit{Matrix dimensions must agree.}

This is a very common error in MATLAB programming, and it’s useful to know what it means. To fix matrix dimension errors, check the dimensions of the involved variables, either by looking at the \textit{Workspace} window or entering the line \([\text{n} \times \text{n} \times \text{col}] = \text{size}(X)\) (without a semi-colon ‘;’) in the \textit{Command} window or in your \textit{.m}-file.

If \( u \) is a column vector of length \( m \) and \( v \) is a row vector of length \( n \), then \( \mathbf{M} = \mathbf{u} + \mathbf{v} \) represents a matrix of dimension \( m \)-by-\( n \), where

\[ M_{ij} = u_i + v_j \]

8. Matrix Addition and Subtraction
If \( P \) and \( Q \) are both \( m \)-by-\( n \) matrices then \( \mathbf{M} = \mathbf{P} + \mathbf{Q} \) is also a \( m \)-by-\( n \) matrix where

\[ M_{ij} = P_{ij} + Q_{ij} \]

If \( P \) and \( Q \) are matrices of different dimension, they cannot be added, and MATLAB / Octave will return the error: \textit{Matrix dimensions must agree.}

9. Vector Multiplication
In MATLAB there are four kinds of vector multiplication: \(*\), \( \ast \), \( .* \), and \texttt{cross}.

If \( u \) and \( v \) are both column vectors of length \( n \), then \( \mathbf{w} = \mathbf{u'} \ast \mathbf{v} \) is a scalar-valued \textit{inner product} (or \textit{dot product})

\[ w = \sum_{i=1}^{n} u_i v_i \]

If \( u \) is a column vector of dimension \( m \)-by-1 and \( v \) is a column vector of dimension \( n \)-by-1, then \( \mathbf{M} = \mathbf{u} \ast \mathbf{v'} \) is a matrix-valued \textit{outer product}

\[ M_{ij} = u_i v_j \]

The matrix \( \mathbf{M} \) has dimension \( m \)-by-\( n \).

If \( u \) and \( v \) are vectors of the same dimension (\( 1 \)-by-\( n \) or \( n \)-by-1), then \( \mathbf{w} = \mathbf{u} \ast \ast \mathbf{v} \) is a vector of the same dimension with elements

\[ w_i = u_i v_i \]

This is called “element by element” multiplication.

If \( u \) and \( v \) are both vectors of dimension 3, then \texttt{cross} \( \langle \mathbf{u}, \mathbf{v} \rangle \) is the \textit{cross product} of \( \mathbf{u} \) and \( \mathbf{v} \) defined by the determinant

\[ u \times v = \det \left( \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{bmatrix} \right) \]
10. **Matrix Multiplication**

In the *matlab* language there are three kinds of matrix multiplication: *, *, and .*

If $P$ has dimension $p$-by-$q$ and $Q$ has dimension $q$-by-$r$ then $M = P * Q$ is the *inner product* (or *dot product*) ans is a $p$-by-$r$ matrix.

$$M_{ij} = \sum_{k=1}^{q} P_{ik}Q_{kj}$$

Note that the *inner dimension* of the two matrices being multiplied must be equal.

If $P$ has dimension $p$-by-$q$ and $Q$ has dimension $q$-by-$r$ and $p$ is not the same as $r$, then $M = Q * P$ cannot be computed since the inner dimensions do not agree, and MATLAB / Octave will display the error message: **Matrix dimensions must agree.**

If $u$ is a column vector of dimension $m$-by-1 and $v$ is a column vector of dimension $n$-by-1, then $M = u \ast v'$ is a matrix-valued *outer product*

$$M_{ij} = u_i v_j$$

The matrix $M$ has dimension $m$-by-$n$.

If $P$ and $Q$ are both $m$-by-$n$ matrices then $M = P \ast Q$ is also a $m$-by-$n$ matrix where

$$M_{ij} = P_{ij}Q_{ij}$$

This is called “element by element” multiplication.

11. **Matrix (and Vector) “Exponentiation”**

If $A$ is a square matrix, then $M = A^2$ is the same as $M = A*A$

If $A$ is a matrix (or a vector) of any dimension, then $M = A^{-2}$ is the same as $M = A.*A$

12. **Matrix (and Vector) “Division”**

In the *matlab* language element-wise division by a matrix (or vector) is done via: ./

If $P$ and $Q$ are matrices (or vectors) with the same dimension, then $M = P ./ Q$ is a matrix (or vector) of the same dimension where

$$M_{ij} = P_{ij}/Q_{ij}$$

This also works if the numerator is a scalar: $M = a ./ Q$ is a matrix (or vector) of the same dimension as $Q$ where

$$M_{ij} = a/Q_{ij}$$

8 **Solving Linear Equations**

Given a full rank matrix $A$ and a vector $b$ in which $A$ and $b$ have the same number of rows, the system of linear equations $Ax = b$ may be solved for $x$ using $x = A \backslash b$. This is called a *matrix left divide*. 
If $A$ is square (and full rank), $x = \text{A} \backslash \text{b}$ is conceptually the same as $x = \text{A}^{-1}\text{b}$, or in the \texttt{matlab} language, $x = \text{inv}(\text{A}) * \text{b}$.

If $A$ is tall (and full rank), there are more (unique) equations (rows of $A$) than unknowns (rows of $x$), meaning there could be no unique solution, in which case $x = \text{A} \backslash \text{b}$ returns the value of $x$ that minimizes the sum of the squares of the difference between $Ax$ and $b$, which is $(Ax - b)'(Ax - b)$. This is used when fitting an equation to data points.

If $A$ is wide (and full rank), there are more unknowns than equations, and there can be an infinite number of solutions, in which case and $x = \text{A} \backslash \text{b}$ returns the value of $x$ that minimizes $x'x$ such that $Ax - b = 0$.

\section{Comparison Operators}

The comparison operators $<$, $>$, $\leq$, $\geq$, $\approx$, $\neg$, \texttt{any}, \texttt{all}, \texttt{find} (with vectors and matrices)

1. The comparison operators $<$, $>$, $\leq$, $\geq$, $\approx$, $\neg$ return a 1 if the comparison is \texttt{true} and return a 0 otherwise.

Examples with vectors ... for $a = \begin{bmatrix} 1 & 3 & 11 \end{bmatrix}$ and $b = \begin{bmatrix} 3 & 2 & 10 \end{bmatrix}$;
$x = a < 5; \text{evaluates to } x = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}$
$x = a <= 7; \text{evaluates to } x = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$
$x = a \approx 5; \text{evaluates to } x = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$
$x = a <= 0; \text{evaluates to } x = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$
$x = a <= b; \text{evaluates to } x = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}$
$x = a == b; \text{evaluates to } x = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$

Examples with matrices ... for $A = a'*b$ and $B = \text{pascal}(4)$
$X = A < 5; \text{evaluates to}$

\begin{align*}
X &= \begin{bmatrix} 1 & 0 & 0 & 0 \\
& 0 & 0 & 0 \\
& 0 & 0 & 0 \\
& 0 & 0 & 0 
\end{bmatrix}
\end{align*}

$X = A == 7; \text{evaluates to}$

\begin{align*}
X &= \begin{bmatrix} 0 & 0 & 1 & 0 \\
& 0 & 0 & 0 \\
& 0 & 0 & 0 \\
& 0 & 0 & 0 
\end{bmatrix}
\end{align*}

$X = A \approx 5; \text{evaluates to}$

\begin{align*}
X &= \begin{bmatrix} 1 & 0 & 1 & 1 \\
& 1 & 1 & 1 \\
& 1 & 1 & 1 \\
& 1 & 1 & 1 
\end{bmatrix}
\end{align*}

$X = A <= 0; \text{evaluates to}$

\begin{align*}
X &= \begin{bmatrix} 1 & 0 & 1 & 1 \\
& 1 & 1 & 1 \\
& 1 & 1 & 1 \\
& 1 & 1 & 1 
\end{bmatrix}
\end{align*}
X = [ 0 0 0 0 \\
      0 0 0 0 \\
      0 0 0 0 \\
      0 0 0 0 ]

X = A <= 9*B; evaluates to

X = [ 1 1 1 1 \\
      0 0 0 1 \\
      0 0 1 1 \\
      0 0 1 1 ]

2. q = any(x); returns 1 if any element of x is non-zero, 0 otherwise

3. q = all(x); returns 1 if all elements of x are non-zero, 0 otherwise

4. idx = find(x); finds the indices of vector x that are non-zero.
   [IDr, IDc] = find(X); finds the row, column indices of matrix X that are non-zero.
   Examples with vectors ... for a = [ 1 : 3 : 11 ]; and b = [ 3 : 2 : 10 ];
   idx = find(a < 5); evaluates to idx = [ 1 2 ]
   idx = find(a == 7); evaluates to idx = 3
   idx = find(a ~= 5); evaluates to idx = [ 1 2 3 4 ]
   idx = find(a <= 0); evaluates to idx = [ ](0x1)
   idx = find(a <= b); evaluates to idx = [ 1 2 3 ]
   idx = find(a == b); evaluates to idx = 3
   Examples with matrices ... for A = a’*b and B = pascal(4)
   [IDr, IDc] = find(A < 5); evaluates to IDr = 1 and IDc = 1
   [IDr, IDc] = find(A == 7); evaluates to IDr = 1 and IDc = 3
   [IDr, IDc] = find(A ~= 5); evaluates to
   IDr = [ 1 2 3 4 2 3 4 1 2 3 4 1 2 3 4 ]
   IDc = [ 1 1 1 2 2 2 3 3 3 4 4 4 4 4 4 ]
   [IDr, IDc] = find(A <= 0); evaluates to IDr = [ ](0x1) and IDc = [ ](0x1)
   [IDr, IDc] = find(A <= 9*B); evaluates to
   IDr = [ 1 1 1 3 4 1 2 3 4 ];
   IDc = [ 1 2 3 3 4 4 4 4 ];

5. Find the indices of the elements of vector u that are greater than u_limit
   idx_u = find( u > u_limit );
   Find the fraction of elements in the vector u that are greater than u_limit
   P = sum( u > u_limit ) / length( u );

6. Use find to confirm mathematical equalities, for example for any square matrix A,
   A = rand(5);
   find(A^2 - A*A) evaluates to [ ](0x1)
&&, ||, &, | (with scalars and vectors)

The logical operators && (AND) and || (OR) return a 1 if the condition is true and return a 0 otherwise.

1. && is used as AND to combine comparison operators or logical operators.
   For example, for scalars a, b, c, d, if both a is less than b AND c is less than d,
   \[ x = (a < b) && (c < d) \]
   evaluates to
   \[ x = 1 \]

2. || is used as OR to combine comparison operators or logical operators.
   For example, for scalars a, b, c, d, if either a is less than b OR c is less than d,
   \[ x = (a < b) || (c < d) \]
   evaluates to
   \[ x = 1 \]

3. & is used as a bitwise AND to combine vector-valued logical statements.
   For example for vectors of equal length a, b, c, d
   \[ x = (a < b) & (c < d) \]
   evaluates to a vector with each element corresponding to the elements of
   \[ (a < b) \text{ AND } (c < d) \].
   If \( a = [1 2 3 4] \); \( b = [4 3 2 1] \); \( c = [4 1 2 3] \); and \( d = [1 2 3 4] \);
   \( (a < b) \) evaluates to \( [1 1 0 0] \) and \( (c < d) \) evaluates to \( [0 1 1 0] \) so
   \[ x = (a < b) \text{ AND } (c < d) \] evaluates to \[ x = [0 1 0 0] \]

4. | is used as a bitwise OR to combine vector-valued logical statements.
   For example for vectors of equal length a, b, c, d
   \[ x = (a < b) | (c < d) \]
   evaluates to a vector with each element corresponding to the elements of
   \[ (a < b) \text{ OR } (c < d) \].
   With the same vectors in the previous example ...
   \( (a < b) \) evaluates to \( [1 1 0 0] \) and \( (c < d) \) evaluates to \( [0 1 1 0] \) so
   \[ x = (a < b) \text{ OR } (c < d) \] evaluates to \[ x = [1 1 1 0] \]

11 the matlab language has many built-in math functions, such as ...
12 max, min, sort, sum, diff (with vectors and matrices)

1. \([a_{\text{max}}, \text{idx}] = \text{max}(x)\)
   ... the most positive value of vector \(x\) and its index
\([A_{\text{max}}, \text{idx}] = \text{max}(X)\)
   ... the max value of each column of matrix \(X\) and the index of each column

2. \([a_{\text{min}}, \text{idx}] = \text{min}(x)\)
   ... the most negative value of vector \(x\) and its index
\([A_{\text{min}}, \text{idx}] = \text{min}(X)\)
   ... the min value of each column of matrix \(X\) and the index of each column

3. \([a_{\text{sort}}, \text{idx}] = \text{sort}(x)\)
   ... sorted values of vector \(x\) in numerically increasing order and the sorted index
\([A_{\text{sort}}, \text{IDX}] = \text{sort}(X)\)
   ... sorted columns of matrix \(X\) and the sorted index of each column

Example with a vector ... \(a = [ 5 7 5 3 9 1 4 ]\);
\([\text{sort}_a, \text{idx}] = \text{sort}(a)\) evaluates to
\(\text{sort}_a = [ 1 3 4 5 5 7 9 ]\)
\(\text{idx} = [ 6 4 7 1 3 2 5 ]\)
\(\text{find}(a(\text{idx}) - \text{sort}_a)\) evaluates to [] (1x0)

4. \(\text{sum}_x = \text{sum}(x)\) ... the sum of the elements of the vector \(x\)
\(\text{sum}_A = \text{sum}(X)\) ... the sum down the columns of matrix \(X\)
\(\text{sum}_A = \text{sum}(X,2)\) ... the sum across the rows of matrix \(X\)

Example with a matrix ... \(M = \text{magic}(5)\);
\(M = \begin{bmatrix}
17 & 24 & 1 & 8 & 15 \\
23 & 5 & 7 & 14 & 16 \\
4 & 6 & 13 & 20 & 22 \\
10 & 12 & 19 & 21 & 3 \\
11 & 18 & 25 & 2 & 9
\end{bmatrix}\)
\(\text{sc} = \text{sum}(M)\) evaluates to \(\text{sc} = [ 65 \ 65 \ 65 \ 65 \ 65 ]\)
\(\text{sc} = \text{sum}(M,2)\) evaluates to
\(\text{sc} = [ 65 \ 65 \ 65 \ 65 ]\)

5. \(\text{diff}_x = \text{diff}(x)\) ... the differences between adjacent elements of the vector \(x\)
\(\text{diff}_A = \text{diff}(X)\) ... the adjacent differences down the columns of matrix \(X\)
\(\text{diff}_A = \text{diff}(X,2)\) ... the adjacent differences across the rows of matrix \(X\)

Example with a vector ... \(a = [ 5 7 5 3 9 1 4 ]\)
\(\text{da} = \text{diff}(a)\); evaluates to \(\text{da} = [ 2 -2 -2 -8 -3 ]\)
# Conditional program flow and loops

## 13.1 if else

```plaintext
if (condition is true)
    (run this code)
else
    (run alternative code)
end  \% close if–else with end
```

## 13.2 switch case otherwise

```plaintext
switch (option)
    case (optionA)
        (this code runs when option = optionA)
    case (optionB)
        (this code runs when option = optionB)
    otherwise
        (this code runs when option is neither option A nor optionB)
end  \% close switch–case–otherwise with end
```

## 13.3 while loops

```plaintext
while (condition is true)
    (run this code)
end  \% close while loops with end
```

For example, using nested while-loops to evaluate

\[
S(n) = \sum_{k=1}^{n} (-1)^{k-1}/(2k-1)
\]

for three values of \( n = [5, 10, 15] \):

```plaintext
n = [5, 10, 15];
S = zeros(1, length(n));
i = 1;
while (i <= length(n))
    k = 1;
    while (k <= n(i))
        % accumulate the sum for the i-th value of vector n and summing over values of k
        S(i) = S(i) + (-1)^(k-1) / (2*k-1)
        k = k+1;
    end
    i = i+1;
end
```

Using a while-loop, the index variables \( i \) and \( k \) need to be initialized and incremented using additional lines of code. The command `break` exits a loop before the end condition (e.g., \( i \) equals \( n \)) is met.

## 13.4 for loops

```plaintext
for i = 1 : n
    (run this code)
end  \% close for loops with end
```

For example, using nested for-loops to evaluate the same sum as above

```plaintext
n = [5, 10, 15];
S = zeros(1, length(n));
for i = 1 : length(n)  \% loop over the values in the vector n
    for k = 1 : n(i)  \% what does \( n(i) \) represent?
        % accumulate the sum for the i-th value of vector n and summing over values of k
        S(i) = S(i) + (-1)^(k-1) / (2*k-1)
    end
end
```
14 Loops should be avoided whenever possible

1. Evaluate $M = P \cdot Q$ where $P$ is a $p \times n$ matrix and $Q$ is a $n \times q$ matrix.

   Don’t do this
   
   ```matlab
   for i = 1 : p
       for j = 1 : q
           M(i,j) = 0;
       end
   end
   ```

   Just do this
   
   ```matlab
   A = P * Q;
   ```

2. Evaluate $y_i = \sin(\pi x_i)$ for $x = [ 0 \ 0.01 \ 0.02 \ ... \ 10 ]$

   Don’t do this
   
   ```matlab
   x = [ 0 : 0.01 : 10 ];
   N = length(x);
   for i = 1 : N
       y(i) = sin(pi*x(i));
   end
   ```

   Just do this
   
   ```matlab
   dx=0.01; N=10/dx; x=[0:N]*dx; % all values of x in a vector
   y = sin(pi*x); % evaluate in one line without a loop
   ```

3. Evaluate $y_i = \sum_{k=1}^{5} a_j \cos(\pi (2k - 1) x_i))$

   for $a = [ 5 \ -4 \ 3 \ -2 \ 1 ]$ and $x = [ 0 \ 0.01 \ 0.02 \ ... \ 10 ]$

   Don’t do this
   
   ```matlab
   a = [ 5 , -4 , 3 , -2 , 1 ];
   x = [ 0 : 0.01 : 10 ];
   M = length(a);
   N = length(x);
   y = zeros(1,N);
   for i = 1 : N
       for k = 1 : M
           y(i) = y(i) + a(k) * cos(pi*(2*k-1)*x(i));
       end
   end
   ```

   Just do this
   
   ```matlab
   a = [ 5 , -4 , 3 , -2 , 1 ]; % coefficient values, row vector
   dx=0.01; N=10/dx; x=[0:N]*dx; % all values of x in a row vector
   k = [ 1 : 5 ]; % integers 1 , 2 , 3 , 4 , 5 * col vector
   y = a * cos(pi*(2*k-1)*x); % evaluate in one line without for-loops
   ```

   • $\pi(2k-1)x$ is an outer product resulting in matrix with 5 rows and 1001 columns,
   • $\cos(\pi(2k-1)x)$ is also a matrix with 5 rows and 1001 columns, and
   • $a * \cos(\pi(2k-1)x)$ is a vector-matrix inner product resulting in a row vector with 1001 columns.
15 Scripts and functions

To perform lots of operations in a single go, write your operations into a .m-file as scripts or functions. You may run a script by typing the name of the script (without the .m file name extension) into the Command window or by entering F-5 in the Editor window. With scripts, numerical values for all variables are already in the Workspace or are provided within the script. With functions, none of the internal variables are stored in the Workspace. Basically, scripts are an assembly of lines you would type into the Command window, whereas functions compute a set of zero or more output results using a set of zero or more input values. To define a function in a .m-file, begin the function with a line like:

```matlab
function [outputA, outputB] = functionName(inputA, inputB)
```

Save the function into a .m-file named functionName.m ... the same as the function named at the beginning of the file. The name of the .m-file must match the name of the main function in the file. To call the function (either in the Command window or within other functions) enter the line of code following the function keyword in the function’s .m-file. For example to call the example function above, type the following lines (either in a different function, a script, or in the Command window)

```matlab
inputA = 7;
inputB = 8;
[outputA, outputB] = functionName(inputA, inputB);
```

More than one function can be defined in a single .m-file. Other functions defined in the .m-file can only be helper functions to the first function included in functionName.m and are not accessible outside of the .m-file. For example, consider an .m-file called mainFunction.m containing code for functions mainFunction, helper1, helper2, etc. If I type mainFunction into the Command window, then function mainFunction will run and can make use of functions helper1 and helper2 written within the .m-file mainFunction.m. However, if I type helper1 in the Command window I will get an error stating that the function helper1 is undefined, since the Command window cannot see the function helper1 within mainFunction.m.

As you write, download and use more and more .m-functions, you will develop a library of your own .m-files that may be of use in a variety of contexts. It is convenient to store all your general purpose .m-files in one place where MATLAB / Octave knows to look for them. Section 2 explains how to set the Path to your own .m-file library in MATLAB / Octave.

You may wish to save .m-files for a particular project elsewhere, like in a separate myProject directory (aka “folder”). That’s fine and good. When you start MATLAB just click the Browse for folder icon on the upper left of your screen and navigate to your myProject directory (aka “folder”). Then you can work with the .m-files in myProject while also having access to your library of .m-files listed in the Path.
15.1 Example of a script and a function

In this example a function will evaluate the sum

\[ y_i = \sum_{k=1}^{m} a_k \cos(\pi (2k - 1) x_i) \]

for any set of \( m \) coefficients \( a_k \) and any set of values of \( x_i \), and a script will call the function for a specific set of values for \( a_k \) and \( x_i \).

Here is the function, written in a .m-file called sumCos.m.

```matlab
function y = sumCos(a,x)
% y = sumCos(a,x)
% evaluate a sum of cosines with coefficients a and at values of x
% the variables a and x must both be row-vectors
k = [ 1 : length(a) ]'; % integers 1, 2, ... m in a column vector
y = a * cos(pi*(2*k-1)*x); % evaluate in one line without for-loops
```

And here is a script in a .m-file called sumCosCalc.m that uses the function sumCos within the .m-file sumCos.m

```matlab
% sumCosCalc.m
a = [ 5 , -4 , 3 , -2 , 1 ];
x = [ 1 : 0.01 : 10 ];
y = sumCos(a,x);
```

This script can be executed by entering the command sumCosCalc within the Command window or by F-5 in the Editor window displaying the script file sumCosCalc.m. The script file sumCosCalc.m and the function file sumCos.m must be either in the present working directory or in a directory listed in the Path.

15.2 Editing and running .m-files using keyboard shortcuts

Familiarity with keyboard shortcuts facilitates programming. Here are some useful keyboard shortcuts with MATLAB / Octave. Or you may choose to develop expertise with another editor with matlab plugins. (e.g., VS Code, Sublime or even some other text editor that makes the mouse obsolete).

<table>
<thead>
<tr>
<th>Action</th>
<th>MATLAB</th>
<th>Octave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment selected code</td>
<td>Ctrl-R</td>
<td>Ctrl-R</td>
</tr>
<tr>
<td>Un-comment selected code</td>
<td>Ctrl-T</td>
<td>Ctrl-Shift-R</td>
</tr>
<tr>
<td>Auto-indent code</td>
<td>Ctrl-I</td>
<td>Edit&gt;Format&gt;Indent Code</td>
</tr>
<tr>
<td>Go to line (n)</td>
<td>Ctrl-L n</td>
<td>Ctrl-L n</td>
</tr>
<tr>
<td>Find (and replace)</td>
<td>Ctrl-F</td>
<td>Ctrl-F</td>
</tr>
<tr>
<td>Undo</td>
<td>Ctrl-Z</td>
<td>Ctrl-Z</td>
</tr>
<tr>
<td>Save</td>
<td>Ctrl-S</td>
<td>Ctrl-S</td>
</tr>
<tr>
<td>Run the .m-script</td>
<td>F-5</td>
<td>F-5</td>
</tr>
<tr>
<td>Run the selected code</td>
<td>F-9</td>
<td>F-9</td>
</tr>
<tr>
<td>Stop code from running</td>
<td>Ctrl-C</td>
<td>Ctrl-C</td>
</tr>
</tbody>
</table>
To start with, here are two commented examples of 2-D plotting of parametric equations.

```matlab
Plots = 1; % 1: draw plots, 0: don't
pdfPlots = 0; % 1: export plots to .pdf files, 0: don't

t = [0:0.01:10]; % points in the form: [start: stepSize: end]
x1 = cos(t) .* exp(t/10); % compute some data for the 1st curve ... why use .* here?
y1 = sin(t) .* exp(t/10);
x2 = sin(t) ./ (t/5).^2; % compute some data for the 2nd curve ... why use ./ here?
y2 = cos(t) .* (t/5).^2;

if Plots
    figure(1); % plot x vs t and y vs t
lw = 5; % line width value
clf % clear the plot
subplot(2,1,1) % 1st of 2 subplots in this figure
plot(t,x1, 'LineWidth', lw) % plot the first curve
hold on % hold the first plot on the axes before plotting the next curve
plot(t,x2, 'LineWidth', lw) % plot the second curve
ylabel('x(t)') % y-axis label
legend('x_1 (t)','x_2 (t)')
legend(' location', ' SouthWest ')

subplot(2,1,2) % 2nd of 2 subplots in this figure
plot(t,y1, 'LineWidth', lw) % plot the first curve
hold on % hold the first plot on the axes before plotting the next curve
plot(t,y2, 'LineWidth', lw) % plot the second curve
ylabel('y(t)') % y-axis label
xlabel('t') % x-axis label
xaxis('square') % equal spacing in x and y axes
grid on % add a grid to the plot
legend('y_1 (t)','y_2 (t)')
legend(' location', ' SouthWest ')

set(gca, 'FontSize', 14) % increase the font size
end

if pdfPlots, print('MSDB-1.pdf', '-dpdfcrop'); end % in matlab just use '-dpdf'

figure(2); % plot y vs x
clf % clear the plot
plot(x1,y1, 'LineWidth', 5) % plot the first curve
hold on % hold the first plot on the axes before plotting the next curve
plot(x2,y2, 'LineWidth', 5) % plot the second curve
xlabel('x(t)') % axis labels
ylabel('y(t)')
axis('square') % equal spacing in x and y axes
grid on % add a grid to the plot
legend('spiral #1', 'spiral #2')
text(-2.3,-1.7, 'Spiral #1', 'FontSize', 17, 'color', 'b'); % put some text at -2.3,-1.7

text(-2.2,-3.1, 'Spiral #2', 'FontSize', 17, 'color', 'r'); % put some text at -2.2,-3.1

set(gca, 'FontSize', 17) % increase the font size
end
```

end % Plots
And here are commented examples of plotting random data.

```matlab
% Plots = 1;  % 1: draw plots, 0: don’t
pdfPlots = 0;  % 1: export plots to .pdf files, 0: don’t

N = 1000;  % a vector of N random numbers uniformly distributed from 0 to 1
rU = rand(N,1);  % a vector of N random numbers normally distributed, mean=0, std.dev=1
rN = randn(N,1);

if Plots
    figure(3)  % scatter plot of rN vs rU
    clf
    plot(rU,rN, ’o’, ’MarkerSize’, 9)
    xlabel(’uniformly distributed X’)  % increase the font size
    ylabel(’normally distributed Y’)
    set(gca,’FontSize’, 17)
    if pdfPlots, print(’MSDB-3.pdf’, ’−dpdfcrop’); end  % in matlab just use ’−dpdf’

    figure(4)  % PDF and CDF of rU and rN
    nBin = 20;  % number of bins in the histogram
    clf
    subplot(2,2,1)  % plot the histogram (pdf) of rU
    hist( rU, nBin, nBin/(max(rU)-min(rU)) );
    ylabel(’PDF’)
    subplot(2,2,2)
    hist( rN, nBin, nBin/(max(rN)-min(rN)) );
    subplot(2,2,3)
    stairs( sort(rU), (1:N)-0.5)/N, ’LineWidth’, 3 );  % plot the ogive (cdf) of rU
    xlabel(’uniformly distributed X’)
    ylabel(’CDF’)
    subplot(2,2,4)
    stairs( sort(rN), (1:N)-0.5)/N, ’LineWidth’, 3 );  % plot the ogive (cdf) of rN
    xlabel(’normally distributed Y’)
    if pdfPlots, print(’MSDB-4.pdf’, ’−dpdfcrop’); end  % in matlab just use ’−dpdf’
end  % Plots
```
And here are two commented examples of 3-D plotting surfaces and contours of surfaces.

```matlab
Plots = 1; % 1: draw plots, 0: don’t
pdfPlots = 0; % 1: export plots to .pdf files, 0: don’t

%% generate some data for plotting
x1 = [-10 : 10]; % a vector of x1 values
x2 = [-10 : 10]; % a vector of x2 values
[X1,X2] = meshgrid(x1,x2); % all combinations of x1, x2 values
f = X1 + 2*X2 - 3*X1.*X2 + 4*X1.^2 + 5*X2.^2; % objective function ...
% why .* why.^?
g1 = 30*X1 - 20*X2 + 200; % constraint inequality 1
% g1 = 30*ones(size(x1)), 20*ones(size(x2)), 200;
g2 = 35*X1 + 25*X2 - 50; % constraint inequality 2
% g2 = 35*ones(size(x1)), 25*ones(size(x2)), 50;

if Plots
    figure(5)
    clf
    cc = zeros(size(X1),1), size(X1,2),3;
    cr = cc; cg = cc; cb = cc;
    cr(:,1) = 1; cg(:,2) = 1; cb(:,3) = 1; % red, green, blue color codes
    mesh(x1,x2,f, cr); % plot the 3D surface of f(x1,x2) in red
    hold on
    mesh(x1,x2,g1, cg); % plot the 3D surface of g1(x1,x2) in green
    mesh(x1,x2,g2, cb); % plot the 3D surface of g2(x1,x2) in blue
    xlabel('x_1')
    ylabel('x_2')
    zlabel('f and g')
    text(2,10,600,'f(x_1,x_2)', 'color', 'r', 'FontSize', 19) % text label for f
    text(-10, 5, 20, 'g_1(x_1,x_2)', 'color', 'g', 'FontSize', 19) % text label for g1
    text(0, -10, 20, 'g_2(x_1,x_2)', 'color', 'b', 'FontSize', 19) % text label for g2
    axis([-10 10 -10 10 0 800]) % limit the span of the plot axes
    set(gca, 'FontSize', 17) % increase the font size
    if pdfPlots, print('MSDB-5.pdf', '-dpdfcrop'); end % in matlab just use '-dpdf'
end

figure(6) % plot contours of figure 5
clf
contour(x1,x2,f, [-100:50:600], '-r') % plot the contours of f in red
hold on
contour(x1,x2,g1, [0.1 -0.1], '-g', 'LineWidth', 5) % plot the zero contour of g1 in green
contour(x1,x2,g2, [0.1 -0.1], '-b', 'LineWidth', 5) % plot the zero contour of g2 in blue
set(gca, 'FontSize', 17) % increase the font size
xlabel('x_1')
ylabel('x_2')
if pdfPlots, print('MSDB-6.pdf', '-dpdfcrop'); end % in matlab just use '-dpdf'
end % Plots
```
17 Importing and Exporting Data

It is useful to know how to import data into MATLAB code, in order to analyze the data, maybe make some plots, and then to export the processed data in a particular format. The .mat data format is MATLAB’s own file format for exporting and importing MATLAB arrays and structures. For example, to save the three MATLAB variables called A, testData, and results into a single file called myWork.mat:

```
save myWork.mat A testData results
```

And to reload the contents of a .mat file called myWork.mat back into MATLAB,

```
load myWork.mat
```

The save and load commands can import and export data in ASCII (plain text) or binary format, and is compatible with multiple versions of MATLAB. ASCII files with numerical data stored in multiple tab or space delimited columns of equal length, preceded by some lines of header text, in which the first character of each header line is the comment character # may also be imported into a matrix using load. For example, to import the numerical data in an ASCII file called test-20201017.173529.dat

```
% filename test-20201017.173529.dat
% Sat Oct 17 2020 17:35:29
% Experimental Data
% Time Sensor 1 Sensor 2 Sensor 3
0.01 5.02 6.2384 -7.2304
0.02 8.22 7.2484 1.3324
0.03 8.42 -6.2284 -4.2311
0.04 7.32 9.2584 3.4378
0.05 5.18 1.1371 9.3470
```

into a five-row and four-column matrix, use ...

```
data = load('test-20201017.173529.dat');
```

Note here that every row and every column have the same number of values and that all the information in the header is ignored.
To import/export numerical data from/to an ASCII file of delimited columns, use `dlmread` or `dlmwrite`. And to import/export numerical data from/to an ASCII file of comma-delimited columns, (i.e., `.csv` files) use `csvread` or `csvwrite`. In `dlmread` and `csvread`, missing values are replaced by 0. For example, to import the numerical data in a file called `test-20201017.173529.dat` with comma-separated columns

```
filename test-20201017.173529.dat
Sat Oct 17 2020 17:35:29
Experimental Data
Time  Sensor 1  Sensor 2  Sensor 3
0.01  5.02    6.2384   -7.2304
0.02  8.22    7.2484   1.3324
0.03  8.42    -4.2311  
0.04  7.32    9.2584   3.4378
0.05  5.18    1.1371   
```

into a five-row and four-column matrix, skipping the first four lines (rows) of plain text data, skipping over zero columns, and replacing the missing values with zeros, use ...

```matlab
data = dlmread('test-20201017.173529.dat', ',', 4, 0);
```

The most general purpose function for importing ASCII data (text and numerical values) is `textscan`. The `textscan` function scans a data file line by line and saves the results in to a MATLAB structure, having one structure element for each of the format-specified variables. For example, to import the numerical data in contained in the ASCII file called `test-20201017.173529.dat`

```
% filename test-20201017.173529.dat
% Sat Oct 17 2020 17:35:29
% Experimental Data
% Time  Sensor 1  Sensor 2  Sensor 3
0.01  5.02    6.2384   -7.2304
0.02  8.22    7.2484   1.3324
0.03  8.42    -4.2311  
0.04  7.32    9.2584   3.4378
0.05  5.18    1.1371   
```

into a structure of four vectors (one vector for each column), and skipping the first four lines of header information, use ...

```matlab
fid = fopen('test-20201017.173529.dat', 'r'); % file pointer
data = textscan(fid, '%f %f %f %f', 'HeaderLines', 4);
close(fid);
```

in which `data` is a structure with four elements: `data{1}` is a column vector of the first column of the file, `data{2}` is a column vector of the second column of the file, and so on. An alternative usage of the `textscan` command simply skips over all text from `%` to the end of the line, and doesn’t require knowing the number of header lines.

```matlab
data = textscan(fid, '%f %f %f %f', 'CommentStyle', '%');
```

The matlab language supports the C-standard file input/output functions like `fopen()`, `fclose()`, `fprintf()`, and `fscanf()`.

To automate the processing of a number of data files stored in a particular directory, use the `dir` command to create a list of the data files in the directory. For example, to

```matlab
```
import all the data from a set of five-column data files that have header lines and other comments starting with the # character, into a structure of structures of five column vectors,

```matlab
dataDir = '/home/userID/CoolProject/SpecialData/';
files = dir(dataDir); % file names in dataDir
nFiles = size(files,1); % number files in the SpecialData folder

for ff = 3:nFiles % loop over all files in dataDir skipping . and ..
    fn = files(ff).name;
    fid = fopen(fullfile(dataDir,fn) , 'r' ); % file pointer
    dataIn{ff} = textscan( fid, '%f %f %f %f %f', 'CommentStyle', '#');
    fclose(fid);
end
```
18 Skills

1. Expertise with a powerful text editor, with many keyboard shortcuts, (e.g., VS Code, Sublime or even some other text editor that makes the mouse obsolete) is a useful skill. The same editor can be used for coding in any programming language.

2. Understand the differences between a scalar (1-by-1), a row-vector (1-by-n), a column-vector (n-by-1), a matrix (m-by-n), and a multi-dimensionally array (m-by-n-by-r-by-...). By default vectors are row-vectors, unless pre-initialized by a command like:
   
   ```
   my_vector = zeros(100,1);
   ```

3. Understand the need-for, and proper application of element-wise multiplications and vector multiplication:
   
   ```
   X = A * B  ... vector multiplication
   X = A .* B ... element wise multiplication
   ```

   | dimensions of A | 1 × 1 | m × n | m × k | m × n |
   | dimensions of B | m × n | 1 × 1 | k × n | m × n |
   | use:            | *     | *     | *     | . *   |
   | dimensions of X | m × n | m × n | m × n | m × n |

4. Understand the need-for, and proper application of left divide vs. right divide in matrix equations:

   ```
   X = A \ B  ... left divide solves ... AX = B ... for ... X
   ```

   | dimensions of A | 1 × 1 | m × n | m × n | m × n | m × n |
   | dimensions of B | m × n | 1 × 1 | m × k | m × n |
   | use:            | \     | . \   | \     | . \   |
   | dimensions of X | m × n | m × n | n × k | m × n |

   ```
   X = B / A  ... right divide solves ...XA = B ... for ... X
   ```

   | dimensions of A | 1 × 1 | m × n | m × n | m × n | m × n | m × n |
   | dimensions of B | m × n | 1 × 1 | m × n | m × n |
   | use:            | /     | . /   | /     | . /   |
   | dimensions of X | m × n | m × n | k × m | m × n |

5. Understand the need-for, and proper application of element-wise exponentiation and vector exponentiation:

   ```
   X = A ^ B  ... vector exponentiation
   X = A .^ B ... element wise exponentiation
   ```

   | dimensions of A | 1 × 1 | m × n | n × n | m × n |
   | dimensions of B | m × n | 1 × 1 | 1 × 1 | m × n |
   | use:            | ^     | ^     | or    | . ^   |
   | dimensions of X | m × n | m × n | n × n | m × n |

6. Understand the need-for, and proper application of scripts vs. functions.
7. Understanding how to concatenate row-vectors into a matrix and column-vectors into a matrix, or, more generally, how to tile sub-matrices into a larger matrix. Try to answer the following questions with pencil and paper.

Given ...

\[
\begin{array}{c}
\text{a} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \\
\text{b} = \begin{bmatrix} 5 & 6 & 7 & 8 \end{bmatrix} \\
\text{x} = \begin{bmatrix} 5 & 6 & 7 & 8 & 9 \end{bmatrix}
\end{array}
\]

... what are ...

\[
\begin{array}{c}
\text{c} = \begin{bmatrix} \text{a} & \text{b} \end{bmatrix} \\
\text{d} = \begin{bmatrix} \text{a}' & \text{b}' \end{bmatrix} \\
\text{C} = \begin{bmatrix} \text{a} & \text{b} \end{bmatrix} \\
\text{D} = \begin{bmatrix} \text{a}' & \text{b}' \end{bmatrix} \\
\text{X} = \begin{bmatrix} \text{a} & \text{b}' \end{bmatrix} \text{ ... or ... } \text{X} = \begin{bmatrix} \text{a} & \text{b}' \end{bmatrix} \text{ ... or ... } \begin{bmatrix} \text{a} & \text{x} \end{bmatrix} \text{ ... etc.}
\end{array}
\]

Given ...

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

... what is ...

\[
\begin{array}{c}
\text{H} = \begin{bmatrix} \text{A} & \text{B} \\
& \text{C} & \text{D} \end{bmatrix}
\end{array}
\]

8. Use descriptive names for variables, functions, and scripts

9. Start variable names with a letter, not a number.

10. Comment every line

11. Organize your code into sections in this order:

   (a) within scripts, initialize variables with numerical values, or \texttt{NaN} (not a number)

   (b) within functions, pull out vector- or structure- input variables into scalar or matrix variables meaningful names

   (c) calculations

   (d) plots

12. Start programming problems by writing pseudo-code on paper.
19 Debugging

New computer programs almost never work as intended. The ability to locate problematic lines of code within a program is a helpful skill. Fortunately, MATLAB / Octave error messages indicate which lines of code are involved in each error.

Errors in *matlab* often result from trying to operate on matrices of incompatible dimension, divide by zero, index a matrix beyond its dimensions, omit the *end* on a loop, solve an unsolvable matrix equation, etc. Here are eleven common errors and their associated error messages. Clicking on error messages displayed in the *Command* window opens the *Editor* window at the line of the error.

1. Attempting to use C language indexing, as in `A = rand(5); b = A[3][4];`

   _error: parse error near line 11 of file /path/to/some_m_file_

   _syntax error_

   `>>> b = A[3][4]`

2. Attempting to operate on matrices of incompatible dimensions, as in `A = rand(5,3); B = rand(4,7); C = A*B`

   _error: operator *: nonconformant arguments (op1 is 5x3, op2 is 4x7)_
   _error: called from_
   _some_m_file at line 22 column 1_

3. Attempting to evaluate an element of a element vector outside its dimensions, as in `x = rand(1,4); b = x(5);`

   _error: x(5): out of bound 4 (dimensions are 1x4)_
   _error: called from_
   _some_m_file at line 33 column 1_

4. Attempting to evaluate element _i_ of vector _a_ where _i_ is \(\sqrt{-1}\), as in `x = rand(1,4); b = x(i);`

   _error: a(0+1i): subscripts must be real (forgot to initialize i or j?)_
   _error: called from_
   _some_m_file at line 44 column 1_

5. Attempting to compare two values with _=_ instead of _==_, as in `if a = b, which evaluates to the value of b. If b is zero, (a=b) evaluates to _false_, and the if code will not run._

   _warning: suggest parenthesis around assignment used as truth value_
   _near line 55, column 7 in file /path/to/some_m_file.m_
6. Attempting to assign one matrix to another matrix of differing dimension, as in
   \[ A = \text{rand}(5); A(1:2,3:4) = \text{rand}(3,2) \]

   
   
   error: =: nonconformant arguments (op1 is 2x2, op2 is 3x2)
   error: called from
   some_m_file at line 66 column 8

7. Accidentally missing a close parenthesis, as in \[ a = \text{sum}(\text{rand}(4)) \]
   in a file with 76 lines

   
   error: parse error near line 77 of file /path/to/some_m_file.m
   syntax error

   This error message does not identify the line of code with the error.

8. Accidentally missing a close bracket, as in \[ a = [ 2, 4, 6, 8 \]
   in a file with 87 lines

   
   error: parse error near line 88 of file /path/to/some_m_file.m
   syntax error

   This error message does not identify the line of code with the error.

9. Accidentally omitting the \texttt{end} to close a \texttt{for}, \texttt{if} or \texttt{while} statement in a file with 98
   lines

   
   error: parse error near line 99 of file /path/to/some_m_file.m
   syntax error

   This error message does not identify the line of code with the error.

10. Attempting to solve an under-determined system of equations \[ A*x = b \], as in
   \[ a = \text{randn}(6,1); b = \text{randn}(6,1); A = a*a'; x = A \backslash b; \]

   warning: matrix singular to machine precision
   warning: called from
   some_m_file at line 1010 column 1

   Why is the matrix \[ A = a*a' \] not invertible in this example?

11. Attempting to divide by zero, as in \[ x = 1/0 \]

   \[ x = \text{Inf} \]

   No error or warning is provided in this case.
Even if the program runs, it might give nonsensical or otherwise unintended results. These errors can be more difficult to track down.

Here are tips on how to find the lines of code that could be the source of these errors.

1. Read the error messages carefully. MATLAB / Octave will often identify the lines of code involved with attempting operations on matrices with incompatible dimensions, and will indicate the dimensions of the matrices in question.

2. Clicking on error messages displayed in the Command window opens the Editor window at the line of the error.

3. Remove semi-colons (;) from lines in question to view variable values in the Command window. Do the values shown make sense?

4. Check the dimensions and numerical values of a variable using the size function or displaying the value of the variable by omitting the semicolon (;) at the end of the line computing the variable’s value.

5. Use the built-in help function to understand how a function is meant to be used.

6. Save variables into a .mat file in order to read them back in again, after you’ve taken a break from MATLAB programming.