NUS Mathematics Society
NUS Statistics and Data Science Society

MATLAB Workshop Handout

(Answer key)

1 Introduction

1. MATLAB

   In this workshop, we use a powerful and useful computing software called MATLAB. MATLAB stands for *Matrix Laboratory* which is developed by MathWorks (`https://www.mathworks.com/`).

   As an NUS student, you can install MATLAB in one of your personally owned computers. Click the link `https://sm05.stf.nus.edu.sg/studentmatlab/` to get the instruction for installing MATLAB. (If you are not using NUS network, you need to use nVPN (NUS Virtual Private Network) to access this link.) You are required to sign in using your NUSNET ID in the format `nusstu\{nusnetID}` and password.

2. Statements

   - A MATLAB statement is frequently of the form

     $$variable = expression$$

     which assigns the result of *expression* to *variable*.

   - A statement may have the simpler form

     $$expression$$

     in which case the result of *expression* is assigned to a special variable called `ans` (which stands for *answer*).

   - After the MATLAB command prompt », typing the statement

     `help` *topic*

     will give information and usage about the specified *topic*.

3. Working with Matrices

   MATLAB works with matrices. Vectors are considered as "special" matrices. Vectors are either m by 1 matrices (column vectors), or 1 by n matrices (row vectors).

   How to enter matrices:

- To set up an m by n matrix $\mathbf{A}$, execute the statement

  $$\text{A} = \big[\text{ row 1; row 2; } \cdots \text{ ; row } m \,\big]$$

  where the n entries for each row are separated by one or more blank spaces.

  For example, to key in the matrix

  $$\mathbf{A} = \begin{pmatrix} 2 & 4 & 5 \\ 6 & 7 & 9 \\ 7 & 1 & 2 \end{pmatrix},$$

  write `A=[2 4 5; 6 7 9; 7 1 2]`, which should give the following output:

```
>> A=[2 4 5; 6 7 9; 7 1 2]

A =

    2 4 5
    6 7 9
    7 1 2
```

  Now you have *stored* your first matrix!

- For MATLAB, the square brackets `[` and `]` used in entering a matrix are *not* parts of the matrix and are not stored.

- The variable names used in MATLAB are case-sensitive. For example, the variables `B` and `b` are different.

Some matrix operations:

The following are some basic MATLAB commands to perform operations on matrices $\mathbf{A}$ and $\mathbf{B}$:

`A(i,j)`	the (i,j)-entry of $\mathbf{A}$.
`A(i,:)`	the i^{th} row of $\mathbf{A}$.
`A(:,j)`	the j^{th} column of $\mathbf{A}$.
`A+B`	matrix addition of $\mathbf{A}$ and $\mathbf{B}$.
`A-B`	matrix subtraction of $\mathbf{A}$ and $\mathbf{B}$.
`t*A`	scalar multiplication if t is a real number, i.e. $t\mathbf{A}$.
`A*B`	matrix multiplication of $\mathbf{A}$ and $\mathbf{B}$.
`A^n`	raising a square matrix $\mathbf{A}$ to a positive integral power n, i.e. $\mathbf{A}^n$.
`A′`	transpose of $\mathbf{A}$, i.e. $\mathbf{A}^\top$.
`A.′`	*conjugate* transpose of $\mathbf{A}$, i.e. $\mathbf{A}^*$, That is, the matrix is transposed and all the entries are complex-conjugated
`inv(A)`	inverse of an invertible square matrix A, i.e. $\mathbf{A}^{-1}$.
`zeros(n)`	the $n \times n$ zero matrix, i.e. $\mathbf{0}_{n\times n}$.
`zeros(m,n)`	the $m \times n$ zero matrix, i.e. $\mathbf{0}_{m\times n}$.
`eye(n)`	the $n \times n$ identity matrix, i.e. $\mathbf{I}_n$.
`rref(A)`	the reduced row echelon form of $\mathbf{A}$.
`det(A)`	the determinant of $\mathbf{A}$, i.e. $\det(\mathbf{A})$.

4. Precision

All numeric computations in MATLAB are performed with about 16 decimal digits of precision. The format of the displayed output can be controlled by the following commands:

`format short`	scaled fixed point format with 5 digits (default setting).
`format long`	scaled fixed point format with 15 digits.
`format shorte`	floating point format with 5 digits.
`format longe`	floating point format with 15 digits.
`format rat`	approximate fractions.

Take note: MATLAB will approximate decimals with rational numbers when you use `format rat`. Sometimes, this may cause unexpected results. Occasionally, an asterisk `*` may appear when you expect the quantity to be 0.

5. Useful Tips

- You can use the [up arrow key] (↑) to recall previously typed commands
- »`clc` will clear your command window
- `clear` will clear all previously assigned variables. You can clear variables individually by typing »`clear` *variablename*
- double clicking the *variablename* in the workspace window allows you to edit the variable

2 MATLAB interface & workspace essentials

Key things to take note of

- **Command Window**: quick calculations and testing.
- **Workspace**: variables currently defined; double-click to inspect.
- **Current Folder**: where scripts (`.m`) live.
- **Scripts**: write a sequence of commands, run with the **Run** button.

⚠ Tip

Use semicolons `;` to suppress output. Use `whos` to see variable sizes and types.

Warm up

Open MATLAB. In the Command Window run:

```
clc; clear; close all;
a = 3; b = 5;
```

Then check the Workspace panel. What do you see? What does `clear` do?

Solution:

- `clc` clears the Command Window text (it does *not* delete variables).
- `clear` removes variables from the Workspace (use `clear a` to clear only `a`).
- `close all` closes all open figure windows.
- After running the script, you should see variables `a` and `b` in the Workspace, and MATLAB prints output in the Command Window.

Exercise 2.1 — Variables, precedence, and ans

Compute

$$y = \frac{(3+5)^2}{7} - 10 + (2 \times 2).$$

in the MATLAB terminal and store it. Then compute y^2 without assigning it.

Remark. You can also use the `ans` variable as MATLAB always stores the unassigned outputs in `ans`

Solution:

```
    >> y = ((3+5)^2)/7 - 10 + 2*2

    >> y^2
```

Exercise 2.2 — Rounding and numeric display

Let `x = pi/7`. Display it with 5 digits, 15 digits, scientific notation, and as a rational approximation. Then compute `round(x,4)`.

Solution:

```
>> x = pi/7;

>> format short % ~5 digits (default short display)
>> x

>> format long % ~15 digits
>> x

>> format shortE % scientific notation
>> x

>> format rat % rational approximation display
>> x

>> format short % switch back (recommended)
>> round(x,4) % round to 4 decimal places
```

3 Built-in functions

Here, **x** is an arbitrary vector. The following code block illustrates the various built-in functions in MATLAB that are useful.

```
>> y = sqrt(2); % square root

>> z = exp(1); % e

>> u = log(10); % natural log

>> v = log10(10); % base-10 log

>> pi_val = pi; % constant pi

>> m = mean(x); % average

>> md = median(x); % middle value (robust to outliers)

>> s = std(x); % standard deviation (sample by default)

>> v = var(x); % variance (sample by default)

>> mx = max(x); % maximum of all elements in x

>> mn = min(x); % minimum of all elements in x

>> s = sum(x); % sum of all elements in x

>> n = numel(x); % number of elements

>> sz = size(x); % dimensions

>> ln = length(x); % length of longest dimension
```

On the other hand, if **X** is an arbitrary matrix, then many built-in functions in MATLAB operate *column-wise* by default. The following code block illustrates common examples.

```
>> M = mean(X); % mean of each column (1-by-n)

>> MD = median(X); % median of each column (1-by-n)

>> S = std(X); % standard deviation of each column (1-by-n)

>> V = var(X); % variance of each column (1-by-n)

>> MX = max(X); % maximal element of each column (1-by-n)

>> MN = min(X); % minimal element of each column (1-by-n)
>> SM = sum(X); % column sums (1-by-n)

>> N = numel(X); % total number of elements in X

>> SZ = size(X); % dimensions [rows, cols]

>> LN = length(X); % largest dimension (max(rows,cols))

>> R = size(X,1); % number of rows

>> C = size(X,2); % number of columns

% If you want a single number across ALL entries, reshape or use X(:):
>> m_all = mean(X(:)); % mean of all entries (scalar)

>> s_all = std(X(:)); % std of all entries (scalar)

>> mx_all = max(X(:)); % max of all entries (scalar)

>> mn_all = min(X(:)); % min of all entries (scalar)

>> sum_all = sum(X(:)); % sum of all entries (scalar)

% Row-wise versions (operate across columns) using a second argument:
>> M_row = mean(X,2); % row means (m-by-1)

>> S_row = std(X,0,2); % row std (m-by-1), 0 = sample std

>> MX_row = max(X,[],2); % row maxima (m-by-1)

>> SM_row = sum(X,2); % row sums (m-by-1)
```

As seen above, many built-in functions such as `sum`, `mean`, `max`, `min`, `std`, `var` accept a *second argument* that tells MATLAB *which dimension to operate over.*

- For a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$:
  - `mean(X,1)` computes *column-wise* means (result is $1 \times n$).
  - `mean(X,2)` computes *row-wise* means (result is $m \times 1$).
- For a 3D array $\mathbf{X} \in \mathbb{R}^{m \times n \times p}$:
  - `mean(X,3)` averages across the *3rd dimension* ("pages/slices"), giving an $m \times n$ result.

If you omit the second argument (e.g. `mean(X)`), MATLAB uses the *first non-singleton dimension* by default (for a matrix, this usually means column-wise). But this is out of the scope of this workshop.

Exercise 3.1

Create a vector $\mathbf{v} = (2, 4, 6, 8, 10)^\top$. Extract the second and fourth entries, and replace the last entry with 99.

Solution:

```
  >> v = [2; 4; 6; 8; 10];
  >> second = v(2);
  >> fourth = v(4);
  >> v(5) = 99; % you can also use v(end) instead

  >> v % verify you have replaced it
```

Exercise 3.2

Create the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

Extract row 2 and column 3

Solution:

```
>> A = [1 2 3; 4 5 6; 7 8 9];

>> row2 = A(2,:);
>> col3 = A(:,3);
```

Exercise 3.3

Make the vector $\mathbf{v} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)^\top$, reshape into a 3×4 matrix by columns, then transpose it.

Solution:

```
>> v = 1:12
>> M = reshape(v, 3, 4)
>> Mt = M’
```

Exercise 3.4

Let $\mathbf{u} = (1, -2, 3)^\top$ and $\mathbf{v} = (4, 0, -1)^\top$. Compute the dot product of $\mathbf{u}$ and $\mathbf{v}$, their norms, and the angle between them.

⚠ Tip

Use `format short`, `format long`, `format rat` to control display.

Solution:

```
>> u = [1; -2; 3];
>> v = [4; 0; -1];

>> dp = dot(u,v)
>> nu = norm(u)
>> nv = norm(v)

>> theta = acos(dp/(nu*nv)); % answer is in radians, acos = arccos
>> theta_deg = theta*180/pi % convert to degree
```

Exercise 3.5

For `t = 2.4`, compute `sin(t)`, `cos(t)`, `tan(t)`, `exp(t)`, `log(t)`, `sqrt(t)`, and verify that $\sin^2(t) + \cos^2(t) \approx 1$.

Solution:

```
>> t = 2.4;

>> sin(t)
>> cos(t)
>> tan(t)
>> exp(t)
>> log(t) % natural log
>> sqrt(t)

>> val = sin(t)^2 + cos(t)^2;
>> err = abs(val - 1) % should be very small (floating-point error)
```

Exercise 3.6 — Summary statistics (matrix data)

Generate a random data matrix $\mathbf{X} \in \mathbb{R}^{100\times 3}$ using `randn(100,3)`. Think of each row as one sample and each column as one feature.

(a) Compute the column-wise mean, median, standard deviation, and variance of $\mathbf{X}$.

(b) Compute the row-wise mean of $\mathbf{X}$ (i.e. one mean per row).

(c) Compute the overall mean of all entries in $\mathbf{X}$ (a single number).

(d) Verify the dimensions (sizes) of your answers using `size(...)`.

Solution:

```
  >> X = randn(100,3);

  % (a) Column-wise statistics (one number per column)
  >> mu_col = mean(X); % 1x3
  >> med_col = median(X); % 1x3
```

```
>> sd_col = std(X); % 1x3 (sample std by default)
>> var_col = var(X); % 1x3 (sample var by default)

% (b) Row-wise mean (one number per row)
>> mu_row = mean(X, 2); % 100x1

% (c) Overall mean of all entries
>> mu_all = mean(X(:)); % scalar

% (d) Verify dimensions
>> size(mu_col)
>> size(med_col)
>> size(sd_col)
>> size(var_col)
>> size(mu_row)
>> size(mu_all)
```

Exercise 3.7 — Matrix construction + basic summaries

Create a 5×5 matrix $\mathbf{B}$ whose entries are the integers $1, 2, \ldots, 25$ filled **by columns**.

Hint: Use a built-in constructor, then reshape.

(a) Extract the 3×3 submatrix in the top-left corner.

(b) Compute the column sums of $\mathbf{B}$ and the row sums of $\mathbf{B}$.

(c) Find the maximum entry of $\mathbf{B}$.

Solution:

```
% Create B with entries 1..25 filled by columns
>> B = reshape(1:25, 5, 5);

% (a) Extract the 3x3 top-left submatrix
>> TL = B(1:3, 1:3)

% (b) Column sums and row sums
>> col_sums = sum(B, 1) % 1x5
>> row_sums = sum(B, 2) % 5x1

% (c) Maximum entry and its location (row, column)
>> mx = max(B(:)); % mx is the max value, idx is linear index
>> mx
```

4 Arrays, vectors, matrices, and indexing

Creation

```
>> v = [1 2 3 4]; % row vector
>> w = [1; 2; 3; 4]; % column vector
>> A = [1 2 3; 4 5 6; 7 8 9]; % 3x3 matrix

>> I = eye(3); % 3x3 identity matrix
>> Z = zeros(2,4); % 2x4 zero matrix
>> R = rand(2,2); % random matrix
```

Indexing and slicing

```
A(2,3) % row 2, col 3
A(2,:) % whole row 2
A(:,3) % whole col 3
A(1:2,2:3) % submatrix rows 1-2, cols 2-3
```

Warm up

Let

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 \\ 1 & 3 & 4 \\ -2 & 5 & 1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ 5 \\ 0 \end{pmatrix}.$$

(a) Enter $\mathbf{A}$ and $\mathbf{b}$ in MATLAB.

(b) Extract `A(2,:)`, `A(:,1)`, and the submatrix `A(1:2,2:3)`.

(c) Compute $\mathbf{A}^2$, $\mathbf{A}'$ (transpose), and $\det(\mathbf{A})$.

Solution:

```
% part (a)
>> A = [ 2 -1 0 ; 1 3 4; -2 5 1]; % colon required as the question is asking you
    to enter, no output needed
>> b = [1; 5; 0];

% part (b)
>> A(2,:)
>> A(:, 1)

% submatrix A(1:2, 2:3)
>> subA = A(1:2, 2:3);

% part (c)
>> A^2 % Take note: no colon so that output will be shown
>> A'
>> det(A)
```

Exercise 4.1

Input the following three matrices

$$A = \begin{pmatrix} 7 & -2 & 0 & -2 \\ -3 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 16 & -3 & 0 & -2 \end{pmatrix}, \qquad B = \begin{pmatrix} -2 & 5 & 17 & -2 \\ 2 & 6 & -15 & -1 \\ -2 & 6 & 19 & -2 \\ 1 & 2 & -6 & 0 \end{pmatrix}, \qquad C = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

(i) Using MATLAB, compute the products $(\mathbf{AB})^{-1}$, $\mathbf{A}^{-1}\mathbf{B}^{-1}$ and $\mathbf{B}^{-1}\mathbf{A}^{-1}$. What is the relation between these matrices?

Solution:

$\mathbf{B}^{-1}\mathbf{A}^{-1} = (\mathbf{AB})^{-1} \neq \mathbf{A}^{-1}\mathbf{B}^{-1}$.

(ii) Compute $\mathbf{C}^{-1}$. What result do you get? Why?

Solution:

Error message, because $\mathbf{C}$ is not invertible

(iii) Compute $(\mathbf{A}+\mathbf{B})^{-1}$ and $\mathbf{A}^{-1}+\mathbf{B}^{-1}$. Are these matrices equal?

Solution:

The matrices are not equal

(iv) Compute the products $(\mathbf{AB})^\top$, $\mathbf{A}^\top\mathbf{B}^\top$ and $\mathbf{B}^\top\mathbf{A}^\top$. What is the relation between these matrices?

Solution:

$\mathbf{B}^\top\mathbf{A}^\top = (\mathbf{AB})^\top \neq \mathbf{A}^\top\mathbf{B}^\top$.

(v) Compute $(\mathbf{A}^\top)^{-1}$ and $(\mathbf{A}^{-1})^\top$. Are these matrices equal? Is this relation true for any invertible matrix $\mathbf{A}$?

Solution:

This relation is true for any invertible matrix.

(vi) Compute $\mathbf{C}^2$, $\mathbf{C}^3$ and $\mathbf{C}^4$. What do you observe? Can you generalize this observation to $n \times n$ upper triangular matrices with all the diagonal entries 0?

Solution:

Observe that $\mathbf{C}^4 = \mathbf{0}$. In general, if $\mathbf{D}$ is a $n \times n$ upper triangular matrix whose diagonal entries are 0, then $\mathbf{D}^m = \mathbf{0}$ for all $m \geq n$.

Practice

Create a 5×5 matrix M whose (i, j) entry is $i + j$. Hint: use `meshgrid` or loops.

Solution:

```
% Method 1: loops
M = zeros(5,5);
for i = 1:5
    for j = 1:5
        M(i,j) = i + j;
    end
end
M
```

```
% Method 2: meshgrid (vectorized)
>> [i,j] = meshgrid(1:5, 1:5);
>> M = i + j;
M
```

(For the meshgrid version: `i` stores the column indices repeated down rows, `j` stores the row indices repeated across columns, and adding gives `i+j` entrywise.)

Bonus activity

Enter the following statements in the MATLAB command window and observe the outputs. Describe what MATLAB has done on the right.

Commands	**Describe the action**
`» x = [ 1  2  3 ]`	
`» b = [ 1;  2;  3 ]`	
`» A = [ 1  2  pi;  0.1  5  6;  7  8  1/2 ]`	
`» a = A(2,3)`	
`» r = A(2,:)`	
`» c = A(:,3)`	
`» format rat`	
`» A`	
`» format short`	
`» A`	
`» format long`	
`» A`	
`» format`	
`» A`	
`» A1 = [ b  A(:,2)  A(:,3) ]`	
`» 2*3`	
`» ans`	
`» 0.3*A`	
`» ans`	
`» B = ans*b`	
`» A*x`	
`» A*x’`	

Solution:

- `x = [1  2  3]`: create a 1×3 row vector.
- `b = [1;  2;  3]`: create a 3×1 column vector.
- `A = [...]`: create a 3×3 matrix (rows separated by `;`).
- `a = A(2,3)`: access entry $(2,3)$.
- `r = A(2,:)`: extract row 2.
- `c = A(:,3)`: extract column 3.
- `format rat`: change the precision to rational format.
- `A`: display the matrix `A`.
- `format short`: change the precision to scaled fixed point format with 5 digits.
- `A`: display the matrix `A`.
- `format long`: change the precision to scaled fixed point format with 15 digits.
- `A`: display the matrix `A`.
- `format`: reset the precision to default format (scaled fixed point with 5 digits).
- `A`: display the matrix `A`.
- `A1 = [b  A(:,2)  A(:,3)]`: `A1` is a 3×3 matrix obtained from `A` by replacing the first column with `b`
- `2*3`: compute the result and save in `ans`.
- `ans`: display the current value in `ans`.
- `0.3*A`: compute the scalar-matrix multiplication and save in `ans`.
- `ans`: display the current value in `ans`.
- `B = ans*b`: pre-multiply current matrix stored in `ans` with `b`, and save the value in `B`
- `A*x` gives an error, because the number of columns of `A` do not match the number of rows of `x`. So matrix multiplication is not well-defined here.
- `A*x'`: pre-multiply `A` to the transpose of `x`

5 Gaussian elimination & solving linear systems

Concept

Solving $\mathbf{Ax} = \mathbf{b}$ can be done by row operations (Gaussian elimination). In MATLAB:

- `x = A\b;` solves $\mathbf{Ax} = \mathbf{b}$.
- `rref([A b])` produces reduced row echelon form.

Example

```
>> A = [2 -1 1; 1 2 -1; 3 -1 2];
>> b = [1; 0; 4];
>> x = A\b
```

Exercise 5.1

Consider the following system of linear equations

$$\begin{aligned} x + y + 2z &= 1 \\ 3x + 6y - 5z &= -1 \\ 2x + 4y + 3z &= 0 \end{aligned}$$

1. Enter the matrix $\mathbf{A} = \begin{pmatrix} 1 & 1 & 2 \\ 3 & 6 & -5 \\ 2 & 4 & 3 \end{pmatrix}$ and $\mathbf{b} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$.
2. Type `inv(A)*b` to get a solution for this system.
3. Type `A\b`. What do you get?
4. Type `rref([A b])` to get the **reduced** row-echelon form of the augmented matrix.
5. Use the three different methods you learnt to solve the following system:

$$\begin{aligned} x + 2y + z &= 1 \\ x + 2y + 2z &= 1 \\ 2x + 4y + z &= 2 \end{aligned}$$

Solution:

```
>> A = [1 1 2; 3 6 -5; 2 4 3];
>> b = [1; -1; 0];

>> x_inv = inv(A)*b
>> x_bs = A\b
>> rref([A b])
```

Unique solution:

$$(x, y, z) = \left(\frac{33}{19}, -\frac{18}{19}, \frac{2}{19} \right).$$

For the second system (singular matrix), `inv` fails, but `\` and `rref` show infinitely many solutions:

$$(x, y, z) = (1 - 2t,\ t,\ 0), \quad t \in \mathbb{R}.$$

Exercise 5.2

(a) Consider the following system of linear equations

$$\begin{aligned} 3x_1 + 6x_2 + 9x_3 + 5x_4 + 25x_5 &= 53 \\ 7x_1 + 14x_2 + 21x_3 + 9x_4 + 53x_5 &= 105 \\ -4x_1 - 8x_2 - 12x_3 + 5x_4 - 10x_5 &= 11. \end{aligned}$$

Let $[\mathbf{M} \ \ \mathbf{w}]$ be the augmented matrix representing the system. Clearly $\mathbf{M}^{-1}$ does not exist. Does `M\w` yield a solution? How do you confirm that this is indeed a solution for the system?

(Remark: You are not required to know how matrix left division "\" works for this course. You may type "`help mldivide`" in MATLAB to find out more about this operator.)

(b) Find a general solution to this system using `rref([M w])`.

Solution:

```
>> M = [ 3 6 9 5 25;
      7 14 21 9 53;
     -4 -8 -12 5 -10];
>> w = [53; 105; 11];

>> x = M\w
>> res = norm(M*x - w)
>> R = rref([M w])
```

General solution (free parameters x_2, x_3, x_5):

$$x_1 = 6 - 2x_2 - 3x_3 - 5x_5, \qquad x_4 = 7 - 2x_5.$$

Exercise 5.3

Write a function that performs elimination to produce an upper triangular matrix **U** and modified RHS **c**, then back-substitute.

Solution:

```
function [U,c,x] = gauss_elim(A,b)
A = double(A); b = double(b);
n = size(A,1);

% Forward elimination
for k = 1:n-1
    for i = k+1:n
        m = A(i,k)/A(k,k);
        A(i,k:n) = A(i,k:n) - m*A(k,k:n);
        b(i) = b(i) - m*b(k);
    end
end
U = A; c = b;

% Back substitution
x = zeros(n,1);
for i = n:-1:1
    x(i) = (c(i) - U(i,i+1:n)*x(i+1:n)) / U(i,i);
end
end
```

(For real work, add pivoting for stability.)

6 Gram–Schmidt orthonormalization

Goal

Given independent vectors $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{R}^n$, Gram–Schmidt constructs orthonormal vectors $\mathbf{q}_1, \dots, \mathbf{q}_k$ spanning the same subspace.

Formulae

For $j \geq 2$, $\mathbf{u}_j, \mathbf{v}_j, \mathbf{q}_i \in \mathbb{R}^n$

$$\mathbf{u}_j = \mathbf{v}_j - \sum_{i=1}^{j-1} (\mathbf{v}_j \cdot \mathbf{q}_i)\, \mathbf{q}_i, \qquad \mathbf{q}_j = \frac{\mathbf{u}_j}{\|\mathbf{u}_j\|}$$

where $\cdot$ denotes the dot product between two vectors.

Warm up

Let

$$\mathbf{v}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{v}_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \quad \mathbf{v}_3 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}.$$

(a) Implement Gram-Schmidt in MATLAB to produce $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$.
(b) Verify orthonormality by checking $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$ where $\mathbf{Q} = [\mathbf{q}_1\, \mathbf{q}_2\, \mathbf{q}_3]$.
(c) Compare with MATLAB’s `orth` and `qr`.

Solution:

```
% Example implementation (Gram--Schmidt, columns of A)
function Q = gs(A)
[m,n] = size(A);
Q = zeros(m,n);
for j = 1:n
    v = A(:,j);
    for i = 1:j-1
        v = v - (Q(:,i)’*A(:,j))*Q(:,i); % subtract projection
    end
    Q(:,j) = v / norm(v);
end
end
```

(Verify orthonormality via `Q’*Q`, which should be close to the identity.)

Exercise 6.1 (Needs some Linear Algebra knowledge)

Orthonormalize the columns of the matrix

$$\mathbf{V} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

using Gram-schmidt. Compare the output with `orth(V)`.

Solution:

$$e_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \qquad e_2 = \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}.$$

```
V = [1 1; 1 0; 0 1];

v1 = V(:,1);
e1 = v1 / norm(v1);

v2 = V(:,2);
u2 = v2 - dot(v2,e1)*e1;
e2 = u2 / norm(u2);

Q = [e1 e2];
Q’*Q
orth(V)
```

Exercise 6.2

Create a function `gs(A)` that returns orthonormal columns.

> ⚠ Tip
>
> Use dot products as `dot(u,v)` and norms as `norm(u)`.

Solution:

```
function Q = gs(A)
[m,n] = size(A);
Q = zeros(m,n);

for j = 1:n
    v = A(:,j);
    for i = 1:j-1
        v = v - dot(v, Q(:,i))*Q(:,i);
    end
    Q(:,j) = v / norm(v);
end
end
```

7 Plotting basics

Let a, b and c be numbers.

Take note:

- `linspace(a, b, c)` gives us
  (1) Domain: $[a, b]$
  (2) c is the number of samples (or points) joining between a and b.
- `'LineWidth'` is an optional argument; but if you want to use it, you need to specify the thickness beside it.

  For example, suppose you have set the domain `x` and the function `y`.

  – Simply executing the command `plot(x,y)` is okay, but it defaults to Matlab's default line thickness.

  – If you want to change the thickness, you need to parse the `'LineWidth'` argument alongside the desired thickness of your curve lines.

  Say you have two curves `y1` and `y2` defined on a domain `x`. If you want `y1` to have a thickness of say, 3 points and `y2` to have a thickness of 2 points, execute the following commands in your terminal:

```
>> plot(x, y1, 'LineWidth', 3)
>> plot(x, y2, 'LineWidth', 2)
```

  Changing the line thicknesses is important, especially when it comes to exporting the resulting graph as a PDF/SVG file to be put into slides or your papers. Sometimes, journals and publications require a certain thickness of the lines so that the diagram is visible on screen and on print, according to their manuscript dimensions.

- The legend command displays a box that labels each plotted line. The labels are matched to plotted lines *in the order the lines were created.*

  Basic usage (labels only):

  `legend('y1','y2','y3',...)`

  – `'y1','y2',...` are the text labels that appear in the legend.

  – The first label corresponds to the first plotted line, the second label to the second line, etc.

  Using options (property–value pairs):

  `legend('y1','y2',...,'PropertyName','PropertyValue',...)`

  – After the labels, you may add *options* in pairs:

  `'PropertyName','PropertyValue'`

- **Important:** the option arguments come in pairs. If you include a property name, you must also include its value.

Common option: legend position (`Location`):

```
legend('y1','y2','Location','best')
```

- `'Location'` is the property name.
- The value controls where the legend appears.
- `'best'` tells MATLAB to automatically choose a position that avoids covering the plotted lines as much as possible.

Other common `Location` values:

- `'north'` (top center), `'south'` (bottom center), `'east'` (right center), `'west'` (left center)
- `'northeast'`, `'northwest'`, `'southeast'`, `'southwest'`
- `'best'` (automatic placement)

How many arguments can `legend` take?

- Any number of label arguments (one per line you want to label), plus
- Any number of option pairs `'PropertyName','PropertyValue'`.
- Example with multiple option pairs:

```
legend('y1','y2','Location','northwest','Box','off')
```

Remarks:

- The legend uses the *plot order*. If you plot a new line later, the legend labels may no longer match what you expect unless you update the legend.
- You can also call `legend` after plotting everything, which is usually best practice.

Regarding the last point, there are also (many) other option arguments. Here are a few:

Common `Orientation` values:

- `'vertical'` (stack entries top-to-bottom)
- `'horizontal'` (arrange entries left-to-right)

Common `Box` values:

- `'on'` (show a border around the legend)
- `'off'` (hide the border)

Common `FontWeight` values:

- `'normal'` (regular text)
- `'bold'` (bold text)

Common `FontAngle` values:

- `'normal'` (regular text)
- `'italic'` (italic text)

Common `NumColumns` values:

- `1` (single column; default for most legends)
- `2`, `3`, ... (split entries into multiple columns)

Common `AutoUpdate` values:

- `'on'` (legend updates automatically when new plots are added)
- `'off'` (legend stays fixed)

Common `Interpreter` values:

- `'none'` (treat text literally; no special formatting)
- `'tex'` (default; allows basic formatting like `\alpha`, `^`, `_`)
- `'latex'` (LaTeX-style math formatting, e.g. '$y = x^2$')

Common `Units` values:

- `'normalized'` (relative to the figure; numbers between 0 and 1)
- `'pixels'` (absolute pixel units)
- `'inches'`, `'centimeters'` (useful for print layouts)

Other Common `Color` values (legend background):

- `'none'` (transparent background)
- `'white'`, `'black'`, `'yellow'`, etc. (named colors)
- RGB triplet, e.g. `[1 1 1]` (white), `[0 0 0]` (black)

Other Common `EdgeColor` values (legend border):

- `'none'` (no border line)
- `'black'`, `'blue'`, etc. (named colors)
- RGB triplet, e.g. `[0 0 0]` (black)

Example: plotting two curves

```
>> x = linspace(-2*pi, 2*pi, 400);
>> y1 = sin(x);
>> y2 = cos(x);

>> plot(x,y1,'LineWidth',1.5); hold on;
>> plot(x,y2,'LineWidth',1.5);
>> xlabel('x'); ylabel('y');
>> title('Sine and Cosine');
>> legend('sin(x)','cos(x)','Location','best');
>> grid on; hold off;
```

Warm up

Plot $y = x^2$ and $y = 2x + 1$ on the same axes for $x \in [-3, 3]$. Include labels, title, legend, and a grid.

Solution:

```
>> x = linspace(-3, 3, 400);
>> y1 = x.^2;
>> y2 = 2*x + 1
>> plot(x, y1, 'LineWidth', 1.5);
>> hold on;
>> plot(x, y2, 'LineWidth', 1.5);

>> grid on;
>> xlabel('x'); ylabel('y');
>> title('Plots of y = x^2 and y = 2x+1');
>> legend('y = x^2', 'y = 2x + 1', 'Location', 'best');

>> xlim([-3 3]);
```

Exercise 7.1

Plot the curves $y_1 = \sin x$ and $y_2 = \cos x$ for $0 \leq x \leq 2\pi$. Add title, labels, legend and grid.

Solution:

```
>> x = linspace(0, 2*pi, 500);
>> y1 = sin(x);
>> y2 = cos(x);

>> plot(x, y1, x, y2);
>> grid on;
>> xlabel('x'); ylabel('y');
>> title('y_1 = sin(x) and y_2 = cos(x)');
>> legend('sin(x)','cos(x)','Location','best');
```

Exercise 7.2

Generate noisy data $y \approx 2x + 1$, scatter it, and overlay the least-squares line.

Solution:

```
>> x = linspace(0, 10, 50).';
>> y = 2*x + 1 + 0.8*randn(size(x));

>> scatter(x,y); grid on; hold on;

>> p = polyfit(x,y,1);
>> yfit = polyval(p,x);
>> plot(x,yfit);

>> xlabel('x'); ylabel('y');
>> title('Noisy data with least-squares line');
>> legend('data','least-squares fit','Location','best');
```

8 Logical operations and basic control flow

Examples

```
a = 3;
if a > 0
    disp('positive');
elseif a == 0
    disp('zero');
else
    disp('negative');
end
```

```
s = 0;
for k = 1:100
    if mod(k,2)==0
        s = s + k; % sum of even numbers
    end
end
```

Warm up

Given a number x, print whether it's negative, zero, or positive.

Solution:

```
% Example: logical operations + if/for
x = randi(20, 1, 10); % 10 random integers from 1..20
evens = (mod(x,2) == 0); % logical vector
countEven = sum(evens);

if countEven > 5
    disp('More than 5 even numbers');
else
    disp('5 or fewer even numbers');
end
```

Exercise 8.1

Compute $n!$ using a loop (don't use `factorial`).

Solution:

```
n = 10;
fact = 1;
for k = 1:n
    fact = fact * k;
end
fact
```

Exercise 8.2

Definition 1. (Newton-Raphson method) Let $f : \mathbb{R} \to \mathbb{R}$ be a differentiable function. Suppose we want to find a root of f, i.e., a solution of $f(x) = 0$. Starting from an initial guess x_0 with $f'(x_0) \neq 0$, define iteratively the next guess by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0, 1, 2, \ldots$$

Geometrically, x_{n+1} is the x-intercept of the tangent line to $y = f(x)$ at $x = x_n$. If $x_n \to r$ and $f'(r) \neq 0$, then r is a root of f.

Approximate $\sqrt{a}$ using the Newton-Raphson method until the change is below `1e-8`.

Solution:

We want to approximate $\sqrt{a}$. Ideally, this is a root of the function $f(x) = x^2 - a$, so we compute the derivative as $f'(x) = 2x$ and we substitute:

$$\begin{aligned} x_{n+1} &= x_n - \frac{x_n^2 - a}{2x_n} \\ &= x_n - \left(\frac{x_n}{2} - \frac{a}{2x_n}\right) \\ &= \frac{x_n}{2} + \frac{a}{2x_n} \\ &= \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \end{aligned}$$

So use the definition to key in the formula into a conditional while-loop:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right).$$

Implementing into MATLAB code, we have

```
>> a = 10;
>> x = a;
>> tol = 1e-8;

>> while true
    xNew = 0.5*(x + a/x);
    if abs(xNew - x) < tol
        break
    end
    x = xnew;
end

% display output
>> xnew
>> sqrt(a)
```

Exercise 8.3

Create 20 random numbers, then extract only those > 0.5 and set those < 0.2 to 0.

Solution:

```
>> x = randn(20,1);
>> pos = x(x>0.5);
>> x(x<0.2) = 0;

>> pos
>> x
```

Exercise 8.4

Write a script that:

1. creates a random vector `r` of length 20;
2. counts how many entries in `r` are > 0.5;
3. replaces entries ≤ 0.5 by 0 and entries > 0.5 by 1.

Solution:

```
% (1) random vector of length 20
r = rand(20,1);

% (2) count entries > 0.5
count = sum(r > 0.5);

% (3) threshold to 0/1
r(r <= 0.5) = 0;
r(r > 0.5) = 1;

count % return count
r % return r
```

Final task: Calibrating a vibration sensor and estimating uncertainty

You are testing a simple vibration sensor on your device (e.g., an IMU-derived vibration metric or a piezo/vibration pickup). Assume that when the device is *not vibrating*, the sensor readings fluctuate around an unknown baseline value μ due to noise.

When the device is stationary, each sensor reading is modeled as

$$y_1, \dots, y_n \sim \mathcal{N}(\mu, \sigma^2),$$

where σ is known (from the sensor datasheet or a previous calibration), and μ is the unknown baseline offset.

Your task:

(a) Simulate $n = 200$ readings with `randn(200,1)`. The model that fits the data is proposed to be

$$y = 0.02 + 0.01\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, 1).$$

Here, $\mu \approx 0.02$ is a small baseline offset, and $\sigma = 0.01$ is the noise level.

(b) Compute the maximum likelihood estimate (MLE) of the baseline:

$$\hat{\mu} = \frac{1}{200} \sum_{i=1}^{200} y_i.$$

Interpret $\hat{\mu}$ as the sensor offset you would subtract from future readings.

(c) In MATLAB, to define an anonymous function $f(\mathbf{x})$, for instance $f(\mathbf{x}) = \mathbf{x}^2 + 1$, one writes:

$$\texttt{f = } \underbrace{\texttt{@(x)}}_{\text{declare } f(x)} \ \underbrace{\texttt{x.\^{}2 + 1}}_{\text{definition}}$$

You can also declare a function of multiple variables, say

```
g = @(x,y) x + 2*y
```

and this precisely defines a function $g(x, y) = x + 2y$.

Using the information above, define the *log-likelihood function*

$$\ell(\mu) := -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mu)^2,$$

using an *anonymous function* (also known as a lambda function).

(d) Choose a small step size, e.g. $h = 10^{-4}$, and approximate the second derivative at $\hat{\mu}$ by

$$\ell''(\hat{\mu}) \approx \frac{\ell(\hat{\mu} + h) - 2\ell(\hat{\mu}) + \ell(\hat{\mu} - h)}{h^2}.$$

Define the *observed information* as:

$$\mathcal{J}(\hat{\mu}) := -\ell''(\hat{\mu}).$$

(e) Use the curvature-based approximation

$$\widehat{\mathrm{Var}}(\hat{\mu}) \approx \frac{1}{\mathcal{J}(\hat{\mu})}, \qquad \widehat{\mathrm{SE}}(\hat{\mu}) = \sqrt{\widehat{\mathrm{Var}}(\hat{\mu})}.$$

Report $\hat{\mu}$ and $\widehat{\mathrm{SE}}(\hat{\mu})$.

(f) A common engineering rule in Data Science is, if a future reading exceeds

$$\hat{\mu} + 3\,\widehat{\mathrm{SE}}(\hat{\mu}),$$

then the measurement is likely not just baseline noise (it suggests real vibration or a changed operating condition). Compute this threshold and state it clearly.

(g) The theoretical standard error of the mean is $\sigma/\sqrt{n}$. Compare the $\widehat{\mathrm{SE}}(\hat{\mu})$ obtained with $\sigma/\sqrt{200}$ in MATLAB.

Remark. No calculus is required. You are *not* asked to derive $\ell''(\mu)$ analytically—just compute the numerical second derivative in MATLAB. The purpose is to practice: vectors, `mean`, anonymous functions, and numerical differentiation.

Solution:

```
%% Vibration sensor baseline estimation (solution)

% Given
n = 200;
sigma = 0.01; % known noise std
mu_true = 0.02; % baseline offset in the proposed model
h = 1e-4; % step size for numerical second derivative

%% part (a)
y = mu_true + sigma * randn(n,1);

%% part (b) MLE of mu is the sample mean
mu_hat = mean(y);

mu_hat % display mu_hat

%% part (c)
loglike = @(m) -(1/(2*sigma^2))*sum((y - m).^2);

%% part (d)
ell_pp = ( loglike(mu_hat + h) - 2*loglike(mu_hat) + loglike(mu_hat - h) ) / h^2;

ell_pp % display 2nd derivative

% Observed information
J = -ell_pp;

J % display observed information

%% part (e)
```

```
var_hat = 1 / J;
se_hat = sqrt(var_hat);

% display estimated variance and standard error
var_hat
se_hat

%% part (f)
threshold = mu_hat + 3*se_hat;

% display threshold
threshold

%% part (g)
se = sigma / sqrt(n);

% display SE
se

% show absolute difference
abs(se_hat - se)
```

END OF HANDOUT