آموزش متلب

مدرس:

دکتر فریده قریشی، زهرا مهدور

راه‌های ارتباطی:

Email : zahra.mahdevar@yahoo.com

Telegram : Z_Mhdr

جلسه دوم: آشنایی با انواع داده، ساختارهای کنترلی و نوشتن اسکریپت و تابع در متلب

مقدمه

در این جلسه با مفاهیم کلیدی در برنامه‌نویسی متلب آشنا خواهیم شد. ابتدا به بررسی انواع داده‌ها می‌پردازیم که پایه‌ای‌ترین اجزای هر برنامه را تشکیل می‌دهند. سپس ساختارهای کنترلی مانند تصمیم‌گیری و تکرار را بررسی می‌کنیم که امکان کنترل دقیق روند اجرای کد را فراهم می‌سازند. در پایان، نحوه نوشتن اسکریپت‌ها و تعریف توابع را خواهیم آموخت تا بتوانیم برنامه‌هایی ساختارمند و قابل استفاده مجدد ایجاد کنیم. این سه بخش، زیربنای برنامه‌نویسی مؤثر و کاربردی در متلب را تشکیل می‌دهند

بخش اول: انواع داده در متلب

در متلب، داده‌ها می‌توانند شکل‌ها و نوع‌های مختلفی داشته باشند که برای کاربردهای متنوع علمی، مهندسی و محاسباتی طراحی شده‌اند. شناخت انواع داده مانند عددی، منطقی، رشته‌ای، سلولی و ساختاری به ما کمک می‌کند تا برنامه‌های دقیق‌تر و مؤثرتری بنویسیم

▷ *Logical:*

نوع دادهٔ منطقی یکی از انواع پایه‌ای در متلب است که فقط دو مقدار ممکن دارد: ۱ برای «درست» و ۰ برای «نادرست». این نوع داده معمولاً در نتیجهٔ عملگرهای مقایسه‌ای مانند بزرگ‌تر (>)، کوچک‌تر (<) یا مساوی (==) به‌دست می‌آید. داده‌های منطقی کاربرد گسترده‌ای در برنامه‌نویسی دارند، به‌ویژه در شرط‌ها و حلقه‌ها، زیرا با استفاده از آن‌ها می‌توان تصمیم‌گیری کرد که کدام بخش از کد اجرا شود. همچنین از ماتریس‌های منطقی برای انتخاب یا فیلتر کردن عناصر خاصی از داده‌ها نیز استفاده می‌شود

```
true
```

```
ans = logical
```

```
1
```

```
false
```

```
ans = logical
```

```
0
```

```
A = [1 5 3; 7 2 6];
A > 4    % مقایسه عددی: بررسی اینکه کدام درایه‌ها بزرگتر از ۴ هستند
```

```
ans = 2×3 logical array
```

```
0   1   0

1   0   1
```

```
A = [1 0 0; -2 1 4];
B = [1 2 3; 0 1 3];
A < B                مقایسهٔ درایه‌های دو ماتریس و نمایش نتیجه به صورت صفر و یک %
```

```
ans = 2×3 logical array

0   1   1

1   0   0
```

```
% logical(A) : (عدد صفر = نادرست، عدد غیرصفر = درست) تبدیل عدد به مقدار منطقی
x = [0 5 -3];
y = logical(x)
```

```
y = 1×3 logical array

0   1   1
```

```
% real(A) : (نادرست به صفر، درست به یک) تبدیل مقدار منطقی به عدد
z = [true false true];
w = real(z)
```

```
w = 1×3

1     0     1
```

```
c = A == B   % بررسی برابری درایه‌های دو ماتریس و ذخیرهٔ نتیجه در متغیر جدید
```

```
C = 2×3 logical array

1   0   0

0   1   0
```

اپراتورهای منطقی ترکیب

and	& - &&
or	| - ||
Exclusive OR	xor
not	~

```
A = [1 5 3; 7 2 6];
A > 4 | A < 2  % بررسی اینکه درایه‌ها یا بزرگتر از ۴ یا کوچکتر از ۲ باشند
```

```
ans = 2×3 logical array

1   1   0

1   0   1
```

توابع بررسی مقدارهای خاص

isinf(a)	بررسی مقدار بی‌نهایت
isnan(a)	بررسی عدد نبودن
isreal(a)	بررسی حقیقی بودن عدد
isfinite(a)	بررسی مقدار عددی محدود
islogical(a)	بررسی داده منطقی بودن
isempty(a)	بررسی خالی بودن متغیر
isrow(a)	بررسی سطری بودن متغیر
iscolumn(a)	بررسی ستونی بودن متغیر

```
a = inf;
isinf(a)
```

```
ans = logical

1
```

▷ *char*

این نوع داده‌ها در متلب برای نمایش رشته‌ها یا تک‌حرف‌ها استفاده می‌شوند. این داده‌ها معمولاً برای ذخیره و پردازش متن به کار می‌روند و می‌توانند شامل حروف، نمادها و اعداد به صورت کاراکتر باشند

نکته: برای نمایش کاراکترها از تک کوتیشن (') و برای رشته‌ها از دبل کوتیشن (") استفاده می‌شود

1. char

```
name1 = 'zahra';
class(name1)
size(name1)
```

```
ans = 'char'

ans = 1×2

1     5
```

```
name1 = 'Zahra';
name2 = 'Maryam';
names = [name1 , name2]
```

```
names = 'ZahraMaryam'
```

```
%names = [name1 ; name2]
```

```
Error using vertcat

Dimensions of arrays being concatenated are not consistent.
```

```
names = ['Zahra*' ; name2]
```

```
names = 2×6 char array

'Zahra*'

'Maryam'
```

```
number = double('zahra maryam') % تبدیل رشته به عدد
```

```
number = 1×12

   122    97   104   114    97    32   109    97   114   121    97   109
```

```
name = char(number) % تبدیل عدد به رشته
```

```
name = 'zahra maryam'
```

2. string

نکته: این رشته‌ها قابلیت‌های بیشتری مثل مقایسه، جستجو و ترکیب دارند

```
st = "hello";
class(st)
```

```
ans = 'string'
```

```
name4 = "Ahmad";
name5 = "Mohammad";
names_boy = [name4 , name5]
```

```
names_boy = 1×2 string

"Ahmad"     "Mohammad"
```

```
names_boy = [name4 ; name5]
```

```
names_boy = 2×1 string

"Ahmad"

"Mohammad"
```

```
names_boy(2,2) = ["Ali"]   % مقداردهی به عنصر سطر دوم و ستون دوم آرایه
```

```
names_boy = 2×2 string

"Ahmad"     <missing>

"Mohammad"  "Ali"
```

```
eq = name4 == name5  % بررسی برابری دو نام و ذخیره نتیجه در متغیر جدید
```

```
eq = logical
```

0

```
strfind(names_boy, 'm')     % یافتن موقعیت‌های کاراکتر
```

ans = 2×2 cell

	1	2
1	3	[]
2	[5,6]	[]

```
strrep(name4, 'a', 'e')    % جایگزینی
```

ans = "Ahmed"

```
upper(names)      % تبدیل حروف به حروف بزرگ
```

ans = 2×6 char array

'ZAHRA*'

'MARYAM'

```
lower(name5)     % تبدیل حروف به حروف کوچک
```

ans = "mohammad"

```
num2str(eps)      % تبدیل عدد به رشته
```

ans = '2.2204e-16'

▷ ***Numeric***

1. double: عدد اعشاری

به‌طور پیش‌فرض متلب از این نوع داده برای اعداد استفاده می‌کند و پرکاربردترین نوع عدد در متلب است

```
x = 3.14;
class(x)
```

ans = 'double'

2. single: عدد اعشاری با دقت کمتر

یک عدد با دقت شناور 32 بیتی است که نسبت به دابل (اعداد 64 بیتی) فضای حافظه کمتری اشغال می‌کند اما دقت کمتری دارد

```
y = single(eps)
```

y = single

2.2204460e-16

3. int8, int16, int32, int64: عدد صحیح علامت‌دار

در این نوع داده‌ها، اعداد می‌توانند مثبت یا منفی باشند. علامت عدد با استفاده از بیت اول مشخص می‌شود

```
intmin("int8") % حداقل مقدار مجاز
intmax("int8") % حداکثر مقدار مجاز
```

```
ans = int8

-128

ans = int8

127
```

```
x = int8(-100)
```

```
x = int8

-100
```

```
x = int16(20000)
```

```
x = int16

20000
```

```
A = int16 ([1,2; 3 4])
```

```
A = 2×2 int16 matrix

1   2

3   4
```

4. uint8, uint16, uint32, uint64: ذخیره اعداد صحیح نامنفی

در این نوع داده‌ها فقط اعداد غیرمنفی ذخیره می‌شوند. از همه‌ی بیت‌ها برای مقداردهی استفاده می‌شود

```
x = uint64(10000000000000000000)
```

```
x = uint64

10000000000000000000
```

تفاوت در مصرف حافظه بین دو نوع داده

```
U = ones(2000, 'uint8');  % 4000000 Bytes
D = ones(2000);           % 32000000 Bytes
```

نکات:

--> اگر عددی خارج از محدوده‌ی نوع داده باشد، متلب مقدار را به مقدار نزدیک‌تر محدود می‌کند

--> استفاده از نوع صحیح برای صرفه‌جویی در حافظه مفید است، مخصوصاً در تصاویر

▷ *Time*

این نوع داده‌ها برای ذخیره‌سازی و پردازش زمان، تاریخ و تفاوت‌های زمانی استفاده می‌شوند. با استفاده از این ابزارها می‌توان به راحتی تاریخ و زمان فعلی را دریافت کرد، عملیات ریاضی بر روی زمان انجام داد و فرمت‌های مختلف تاریخ و زمان را تغییر داد

datetime()	نمایش تاریخ و زمان با دقت بالا
duration()	نمایش فاصله بین دو زمان
calendarDuration()	فاصله‌های زمانی با در نظر گرفتن تقویم

```
t1 = datetime('now')    % زمان فعلی سیستم
```

```
t1 = datetime

12-Apr-2025 07:23:48
```

```
t2 = datetime(2025, 4, 12, 14, 30, 0)   % زمان دستی
```

```
t2 = datetime

12-Apr-2025 14:30:00
```

```
% ایجاد فاصله زمانی 3 ساعت و 45 دقیقه
d = duration(3, 45, 0)  % ساعت، دقیقه، ثانیه
```

```
d = duration

03:45:00
```

```
% ایجاد فاصله زمانی 2 ماه و 5 روز
calmonths(2) + caldays(5)    % نمایش فاصله زمانی تقویمی
```

```
2mo 5d
```

▷ ***Sparse***

این نوع داده تنها مقادیر غیرصفر ماتریس را ذخیره می‌کند، که باعث کاهش مصرف حافظه در ماتریس‌های بزرگ با مقادیر صفر فراوان می‌شود.
استفاده از ماتریس‌های پراکنده به ویژه در مسائل محاسباتی بزرگ و پیچیده که نیاز به صرفه‌جویی در حافظه دارند، بسیار مفید است

```
% 8*m*n --> حافظه مورد نیاز برای ذخیره یک ماتریس
% 3*8*p+8 -->  حافظه مورد نیاز برای ذخیره یک ماتریس اسپارس
%  8*m*n > 3*8*p+8 --> زمان مناسب برای ذخیره ماتریس به صورت اسپارس
```

```
D = 2 * eye(5);
D_sparce = sparse(D)
```

```
D_sparce =

(1,1)        2

(2,2)        2

(3,3)        2
```

```
(4,4)        2
(5,5)        2
```

```
F = full(D_sparce)
```

```
F = 5×5
2     0     0     0     0
0     2     0     0     0
0     0     2     0     0
0     0     0     2     0
0     0     0     0     2
```

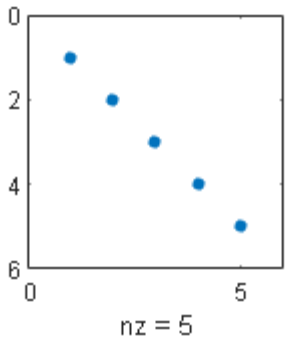
```
a1 = issparse(D_sparce)
```

```
a1 = logical
1
```

```
a2 = issparse(F)
```

```
a2 = logical
0
```

```
spy(D_sparce)   % نمایش گرافیکی ماتریس‌ها
```



▷ *Cell*

در متلب، سلول نوعی ساختار داده‌ای انعطاف‌پذیر است که می‌تواند انواع مختلف داده‌ها را در خود نگهداری کند، مانند عدد، متن، ماتریس، یا حتی سلول‌های دیگر. برخلاف ماتریس‌های معمولی که فقط یک نوع داده را در همهٔ عناصر خود نگه می‌دارند، آرایهٔ سلولی این امکان را می‌دهد که هر عنصر حاوی داده‌ای متفاوت باشد. برای تعریف آرایهٔ سلولی از آکولاد ({}) استفاده می‌شود

```
C = {1, 'matlab', [1 2 3], {2,[1:3]}};
C{2}     % دسترسی به دومین عضو:
```

```
ans = 'matlab'
```

```
c1 = cell(2,2)
```

```
c1 = 2×2 cell
```

	1	2
1	[]	[]
2	[]	[]

```
c2 = {"Zahra", "16", "1400" ; "Maryam" , "17", "1400"};
c2{1,1}
```

```
ans = "Zahra"
```

```
[c2{1,:}]
```

```
ans = 1×3 string

"Zahra"     "16"        "1400"
```

▷ ***Struct***

در متلب، ساختار این نوعی داده پیشرفته است که به شما اجازه می‌دهد مجموعه‌ای از داده‌ها را با نام‌های مشخص (به‌نام فیلد) در یک واحد سازمان دهی کنید. هر فیلد می‌تواند شامل نوع متفاوتی از داده باشد؛ مانند عدد، متن، ماتریس، آرایهٔ سلولی، یا حتی ساختار دیگر. ساختارها برای ذخیرهٔ داده های مرتبط به‌صورت منظم بسیار کاربردی هستند

```
student.name = 'Ali';
student.grade = 18.5;
student.pass = true
```

```
student = struct with fields:

name: 'Ali'

grade: 1.850000000000000e+01

pass: 1
```

▷ ***Table***

در متلب، جدول یک نوع دادهٔ بسیار کاربردی برای سازمان‌دهی و تحلیل داده‌هاست که شباهت زیادی به صفحات گسترده مانند اکسل دارد. جدول‌ها امکان ذخیرهٔ داده‌های مختلف در ستون‌هایی با نام‌های مشخص را فراهم می‌کنند، در حالی که هر ستون می‌تواند نوع متفاوتی از داده داشته باشد (مثلاً عدد، متن، منطقی و ...)

```
t1 = table([1; 2; 3], {'A'; 'B'; 'C'}, [10; 20; 30])
```

```
names = ["Zahra", "Maryam", "Ali"];
last_names = ["Karimi", "Lotfi", "Ahmadi"];
age = [22, 20 , 19];
```

```
var_name = {'Name', 'Last_Name' , 'Age'};
t2 = table(names', last_names', age', 'VariableNames',var_name)
```

t2 = 3×3 table

	Name	Last_Name	Age
1	"Zahra"	"Karimi"	22
2	"Maryam"	"Lotfi"	20
3	"Ali"	"Ahmadi"	19

```
t_sec = t2(:, 1)
```

t_sec = 3×1 table

	Name
1	"Zahra"
2	"Maryam"
3	"Ali"

▷ *Function Handles*

در متلب، این دستور روشی برای ارجاع به یک تابع است، به‌طوری‌که بتوان آن را به متغیر نسبت داد، به عنوان ورودی به تابع دیگر داد، یا در ساختارهای مختلف مانند حلقه‌ها و شرط‌ها استفاده کرد، بدون این‌که خود تابع را مستقیماً فراخوانی کنیم

```
f = @(x) 2*(x^2) + 5;
f(2)
class(f)
```

ans =

13

ans = 'function_handle

جدول اولویت‌بندی عملگرها در متلب

برای گروه‌بندی عملیات ()	پرانتز
! , ~ , + , -	عملگرهای یکتایی (تک عملوندی)
^ , .^	توان
* , .* , / , ./ , \ , .\	ضرب و تقسیم
+ , -	جمع و تفریق
< , > , <= , >= , == , ~=	عملگرهای مقایسه‌ای
& , &&	منطقی AND
| , ||	منطقی OR

Practice 1: Create a logical array that identifies which elements of the vector

`A = [-3, 0, 4.5, NaN, 7, -1]` are greater than zero and valid (i.e., not `NaN`).

بخش دوم: ساختارهای کنترلی در متلب

در متلب، ساختارهای کنترلی برای کنترل جریان اجرای برنامه استفاده می‌شوند. این ساختارها شامل شرط‌ها، حلقه‌ها و انتخاب‌های چندگانه هستند. در ادامه به بررسی مهم‌ترین آن‌ها می‌پردازیم

if

از این ساختار برای بررسی یک یا چند شرط و اجرای بخشی از کد در صورت برقرار بودن آن‌ها استفاده می‌شود

ساختار کلی

```
%if شرط
%    دستورات
%end
```

ساختار کلی همراه با شرط چندگانه

```
% if شرط۱
%     دستورات۱
% elseif شرط۲
%     دستورات۲
% else
%     دستورات پیش‌فرض
% end
```

مثال: بررسی نمره یک دانش‌آموز و نمایش وضعیت او

```
score = 15;

if score >= 17
    disp('عالی')
elseif score >= 14
    disp('خوب')
elseif score >= 10
    disp('قبول')
else
    disp('مردود')
end
```

خوب

for

از این حلقه برای تکرار مجموعه‌ای از دستورات برای یک تعداد مشخص از دفعات استفاده می‌شود

ساختار کلی

```
% for i  = start:step:end
%     دستورات
```

```
% end
```

مثال: محاسبه مجموع اعداد زوج بین 1 تا 10

```
sum = 0;
for i = 2:2:10
    sum = sum + i;
end
disp(['Sum of even numbers = ', num2str(sum)])
```

```
Sum of even numbers = 30
```

while

تا زمانی که یک شرط برقرار باشد، دستورات درون حلقه اجرا می‌شوند

ساختار کلی

```
% while شرط
%     دستورات
% end
```

مثال: پیدا کردن کوچک‌ترین عدد صحیح که مربع آن از 100 بزرگ‌تر باشد

```
x = 1;
while x^2 <= 100
    x = x + 1;
end
disp(['First number with square > 100: ', num2str(x)])
```

```
First number with square > 100: 11
```

break

دستور توقف زمانی استفاده می‌شود که بخواهیم اجرای یک حلقه را به‌طور فوری و پیش از کامل شدن آن متوقف کنیم. یعنی اگر در میانه‌ی اجرای حلقه، به نتیجه‌ی دلخواه برسیم یا شرایط خاصی پیش بیاید، با این دستور می‌توانیم فورا از حلقه خارج شویم و ادامه‌ی دستورات حلقه اجرا نمی‌شود

مثال: اولین عددی را که مربعش بزرگ‌تر از ۱۰۰ باشد پیدا کنید

```
for x = 1:100
    if x^2 > 100
        disp(['First number with square > 100: ', num2str(x)])
        break
    end
end
```

```
First number with square > 100: 11
```

continue

دستور ادامه در داخل یک حلقه استفاده می‌شود و باعث می‌شود که تکرار جاری حلقه به پایان برسد و حلقه به تکرار بعدی خود منتقل شود. این دستور معمولاً زمانی استفاده می‌شود که بخواهیم از اجرای بخشی از کد در یک تکرار خاص صرف‌نظر کنیم و به سراغ تکرار بعدی برویم

مثال: تمام اعداد از فرد از 1 تا 10 را چاپ کنید

```
for x = 1:10
    if mod(x, 2) == 0  % اگر عدد زوج بود
        continue  % به تکرار بعدی می‌رود
    end
    disp(x)  % فقط اعداد فرد چاپ می‌شوند
end
```

```
1

3

5

7

9
```

switch-case

ساختار انتخاب چندگانه در متلب برای انتخاب و اجرای یک بخش خاص از کد بر اساس مقدار یک متغیر استفاده می‌شود. این ساختار زمانی کاربرد دارد که بخواهیم چندین گزینه مختلف را بررسی کرده و بسته به هر کدام یک بخش از کد را اجرا کنیم، به‌جای اینکه از چندین دستور (اگر) و (در غیر این صورت) استفاده کنیم

```
% switch متغیر
%     case مقدار۱
%         دستورات۱
%     case مقدار۲
%         دستورات۲
%     otherwise
%         دستورات پیش‌فرض
% end
```

مثال: روزهای هفته را بر اساس عدد وارد شده نمایش دهید (عدد از ۱ تا ۷)

```
day = 3;

switch day
    case 1
        disp('شنبه')
    case 2
        disp('یکشنبه')
    case 3
        disp('دوشنبه')
    case 4
        disp('سه‌شنبه')
    case 5
        disp('چهارشنبه')
    case 6
        disp('پنج‌شنبه')
    case 7
        disp('جمعه')
    otherwise
        disp('عدد نامعتبر')
```

```
end
```

دوشنبه

مزایای استفاده از انتخاب چندگانه

خوانایی بیشتر: به‌جای استفاده از چندین دستور ا(گر) و (در غیر این صورت)، با استفاده از انتخاب چندگانه کد ساده‌تر و خواناتر می‌شود

کارایی بهتر: در شرایطی که تعداد مقادیر مختلف زیاد باشد، انتخاب چندگانه معمولاً کارایی بالاتری دارد

قابلیت گسترش: می‌توان به راحتی مقادیر جدید به موردها اضافه کرد

Practice 2: Write a program that takes an integer between 1 and 100 from the user and checks whether the number is prime or not, using conditional structures and a loop.

بخش سوم : اسکریپت‌ها و فانکشن

در متلب، برنامه‌ها معمولاً به دو صورت نوشته می‌شوند: اسکریپت‌ها و تابع‌ها (فانکشن‌ها). اسکریپت‌ها مجموعه‌ای از دستورات هستند که به‌صورت متوالی اجرا می‌شوند، در حالی‌که فانکشن‌ها قطعه کدهایی قابل استفاده مجدد هستند که ورودی می‌گیرند و خروجی تولید می‌کنند

1. Scripts

اسکریپت‌ها مجموعه‌ای از دستورهای متلب هستند که به‌صورت خط به خط اجرا می‌شوند

--> بدون ورودی/خروجی

--> کدهای ساده و مستقیم

--> اجرا در (فضای کار) فعلی

2. Functions

تابع در متلب یک بخش از کد است که ورودی می‌گیرد، پردازش انجام می‌دهد و خروجی برمی‌گرداند. توابع معمولا برای کاهش تکرار کد، افزایش خوانایی و ماژولار کردن برنامه‌ها استفاده می‌شوند

```
% function [output1, output2, ...] = functionName(input1, input2, ...)
%     Optional comment or description
%     Function body: operations and calculations
% end
```

توجه

--> تابع باید در فایلی جداگانه با همان نام ذخیره شود

--> نام فایل و نام تابع باید یکی باشد

```
% Example : محاسبه فاکتوریل
function fact = myFactorial(n)
    fact = 1;
    for i = 2:n
        fact = fact * i;
    end
end
```

```
myFactorial(5)
```

```
ans = 120
```

در بسیاری از موارد، نیازی به ایجاد یک فایل تابع جداگانه نداریم؛ بلکه با استفاده از توابع یک خطی می‌توانیم توابع ساده و کاربردی را تعریف کرده و مستقیماً در محل استفاده قرار دهیم. این امکان باعث سهولت کدنویسی و افزایش خوانایی برنامه می‌شود

تابع یک خطی: تابعی است که بدون نام تعریف می‌شود. معمولاً برای عملیات‌های ساده و کوتاه به کار می‌رود

اشاره‌گر تابع: به وسیله علامت @، می‌توانیم به یک تابع (اعم از تابع اصلی موجود در فایل یا تابع ناشناس) اشاره داشته باشیم و آن را به عنوان ورودی به سایر توابع ارسال کنیم. این ویژگی در برنامه‌نویسی تابعی بسیار کارآمد است

ساختار کلی

```
% f = @(arg1, arg2, ..., argN) expression;
```

```
f = @(x1 , x2) x1^2 + x2^2;
y = f(3, 3)
```

```
y = 18
```

بسیاری از توابع داخلی متلب، از توابع یک خطی بهره می‌برند. برای مثال، محاسبه انتگرال

```
result = integral(@(x) sin(x).^2, 0, pi)
```

```
result = 1.5708
```

3.Live Script

لایو اسکریپت یک نسخه تعاملی‌تر از اسکریپت معمولی در متلب است

در اینجا می‌توانید متن فرمت‌شده (مثل درس‌نامه یا توضیحات با فونت زیبا)، معادلات ریاضی، نمودارهای تعاملی، خروجی‌های گرافیکی و جدول را همراه با کد متلب قرار دهید

`Practice 3:` Write a function that returns the mean and standard deviation of a numeric vector. Then, test it in a script.

Exercise 1: Define an anonymous function in MATLAB that computes the numerical derivative of the function $f(x) = x^3 - 5x + 2$ using the central difference method with step size $h = 10^{-5}$. Then, evaluate and print the approximate derivative at the point $x = 2$.

Exercise 2: Solve the linear system $\mathrm{Ax} = b$ where:

A is a square matrix of size $n \times n$,

b is a column vector of size $n \times 1$,

x is the column vector to be solved for.

Use a `switch` statement to determine the solution type based on the rank of matrix A:

If the rank of A equals n, compute the unique solution $x = A^{-1}b$.

If the rank is less than n, indicate that the system has infinite solutions.

If the rank is zero, indicate that the system has no solution.

Write a function to solve the system and return the result.