Polynomials in MATLAB

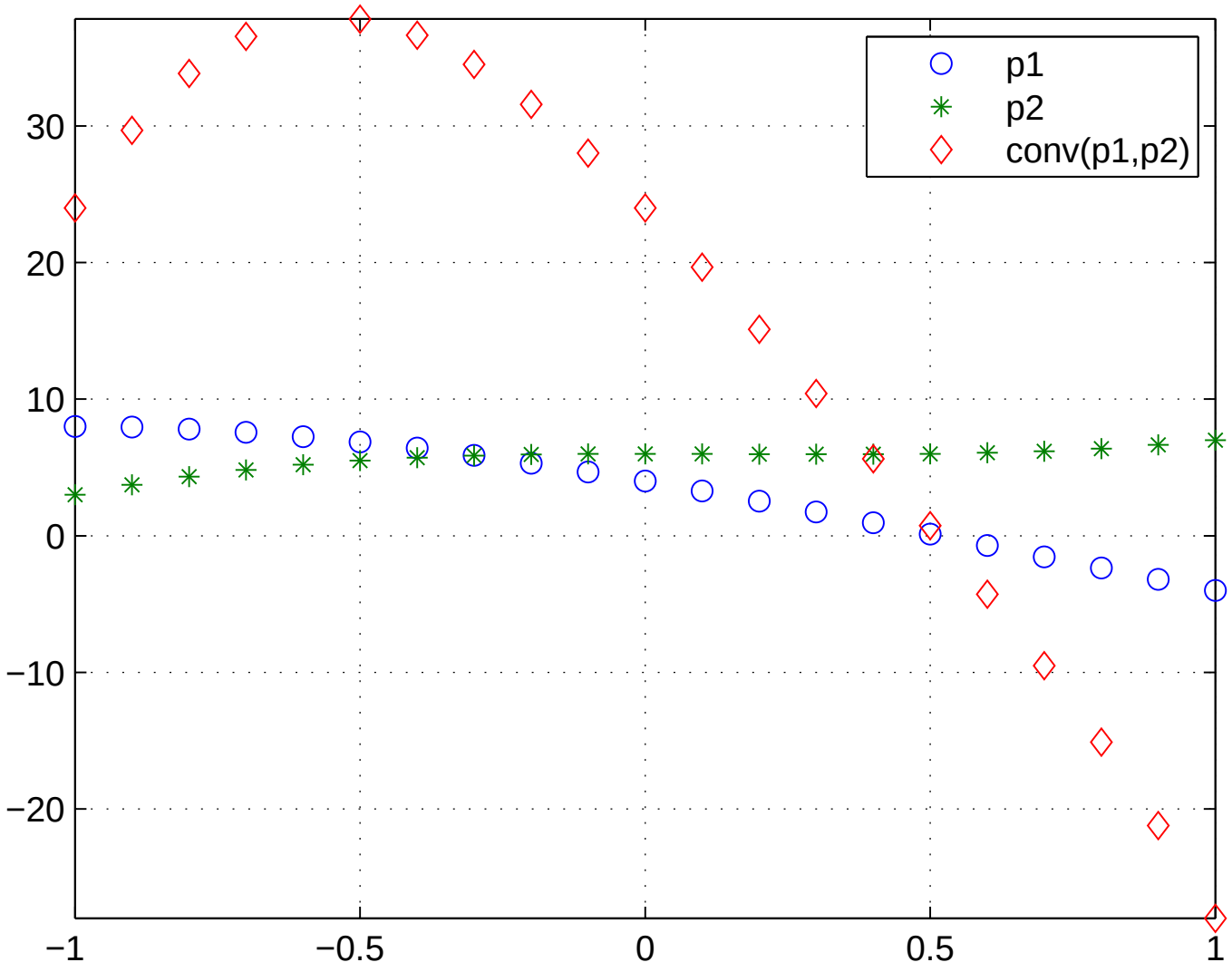
- In fact, all polynomials of n-th order with addition and multiplication to scalars form a vector space, denoted by $\mathbb{P}_n$.
- In general, $f(x)$ is said to be a polynomial of n-order if $f(x)$is given by
$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0,$$
where $a_n \neq 0$.
- It is convenient to express a polynomial by a coefficient vector $(a_n, a_{n-1}, \ldots, a_0)$.
  - Note that the elements are the coefficients of polynomial in descending powers.

Arithmetic Operations

- $P_1 + P_2$ returns the addition result of two polynomial.
- $P_1 - P_2$ returns the subtraction result of two polynomial.
- **conv**(P_1, P_2) returns the resulting coefficient vector of multiplying the two polynomials P_1 and P_2.[1]
- $[Q, R] =$**deconv**(B, A) deconvolves vector A out of vector B.
  - The result is returned in vector Q and the remainder in vector R such that
    $$B = conv(A, Q) + R.$$
  - This is so-called "Euclidean division algorithm."
- **polyval**(P, X) returns the values of a polynomial P evaluated at $x \in X$.

[1] See Convolution.

```
clear; clc;
% main
p1 = [1 -2 -7 4];
p2 = [2 -1 0 6];
x = -1 : 0.1 : 1;
p_a = p1 + p2 % addition
p_b = p1 - p2 % substraction
p_c = conv(p1, p2) % multiplcaition
[q, r] = deconv(p1, p2) % division: q is quotient ...
    and r is remainder.
plot(x, polyval(p1, x), 'o', x, polyval(p2, x), ...
    '*', ...
x, polyval(p_c,x), 'd');
grid on; axis tight;
legend('p1', 'p2', 'conv(p1, p2)');
```


p1
p2
conv(p1,p2)
30
20
10
0
−10
−20
−1
−0.5
0
0.5
1

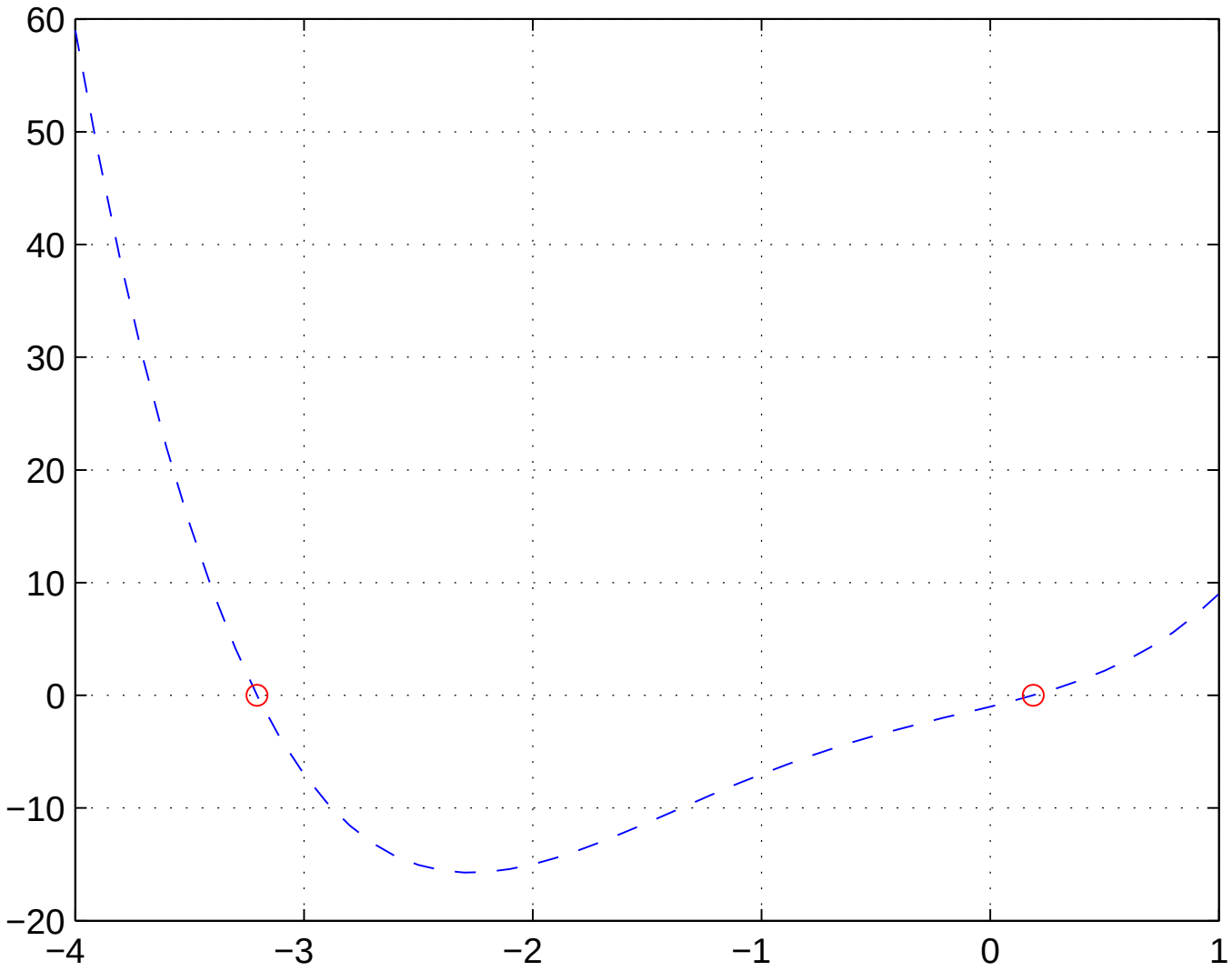
Roots Finding

- **roots**(p) returns a column vector whose elements are the roots of the polynomial p.
- For example,

```
clear; clc;
% main
p = [1, 3, 1, 5, -1];
r = roots(p) % To find all roots
x = -4 : 0.1 : 1;
plot(x, polyval(p, x), '--'); hold on; grid on;
for i = 1 : length(r)
    if isreal(r(i)) == 1
        plot(r, polyval(p, r(i)), 'ro');
    end
end
polyval(p, r) % To verify the roots
```

```
>> r =

   -3.2051
    0.0082 + 1.2862i
    0.0082 - 1.2862i
    0.1886

>> ans =

   1.0e-013 *

    0.4041
   -0.0133 + 0.0529i
   -0.0133 - 0.0529i
         0
```

- Why not exactly zero?


60
50
40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1

Integral and Derivative of Polynomials

- **polyder**(P) returns the derivative of the polynomial whose coefficients are the elements of vector P in descending powers.
- **polyint**(P, K) returns a polynomial representing the integral of polynomial P, using a scalar constant of integration K.

```
clear; clc
% main
p = [4 3 2 1];
p_der = polyder(p)
p_int = polyint(p, 0)
```

Exercise

- Let $f(x) = 4x^3 + 3x^2 + 2x + 1$ for $x \in \mathbb{R}$. Then find its derivative f' and integration $F(x) = \int_0^x f(t)dt$.
  - Do not use the built-in functions provided by MATLAB.
  - Try to manipulate array indexing using **for** loop.

```
clear; clc
% main
p  =[4 3 2 1];
c = 0; % constant of integration
q1 = zeros(1, length(p) - 1);
for i = 1 : length(p) - 1
    q1(i) = p(i) * (length(p) - i);
end
q2 = zeros(1, length(p));
for i = 1 : length(p)
    q2(i) = 1 / (length(p) - i + 1) * p(i);
end
p
q1
[q2 c]
```

- Compare your result to the answer provided by **polyder** and **polyint**.

```
>> Lecture 6
>>
>>              -- User-Controlled Input and Output
>>
```

Contents

- High-Level File I/O
- Low-Level File I/O
- Access to Internet

ASCII

- Everything is encoded in binary codes.
- American Standard Code for Information Interchange[2], aka ASCII, is a character-encoding scheme originally based on the English alphabet that encodes 128 specified characters into the 7-bit binary integers:
  - the numbers $0 - 9$,
  - the letters a-z and A-Z,
  - some basic punctuation symbols,
  - some control codes that originated with teletype machines,
  - and a blank space.

[2] See ASCII.

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

- ASCII can be traced back to 1967, and the current version was issued in 1986.
- Unicode[3] became a standard for the modern systems from 2007.
- ASCII was incorporated into the Unicode character set as the first 128 symbols, so the ASCII characters have the same numeric codes in both sets.

[3] See here.

importdata

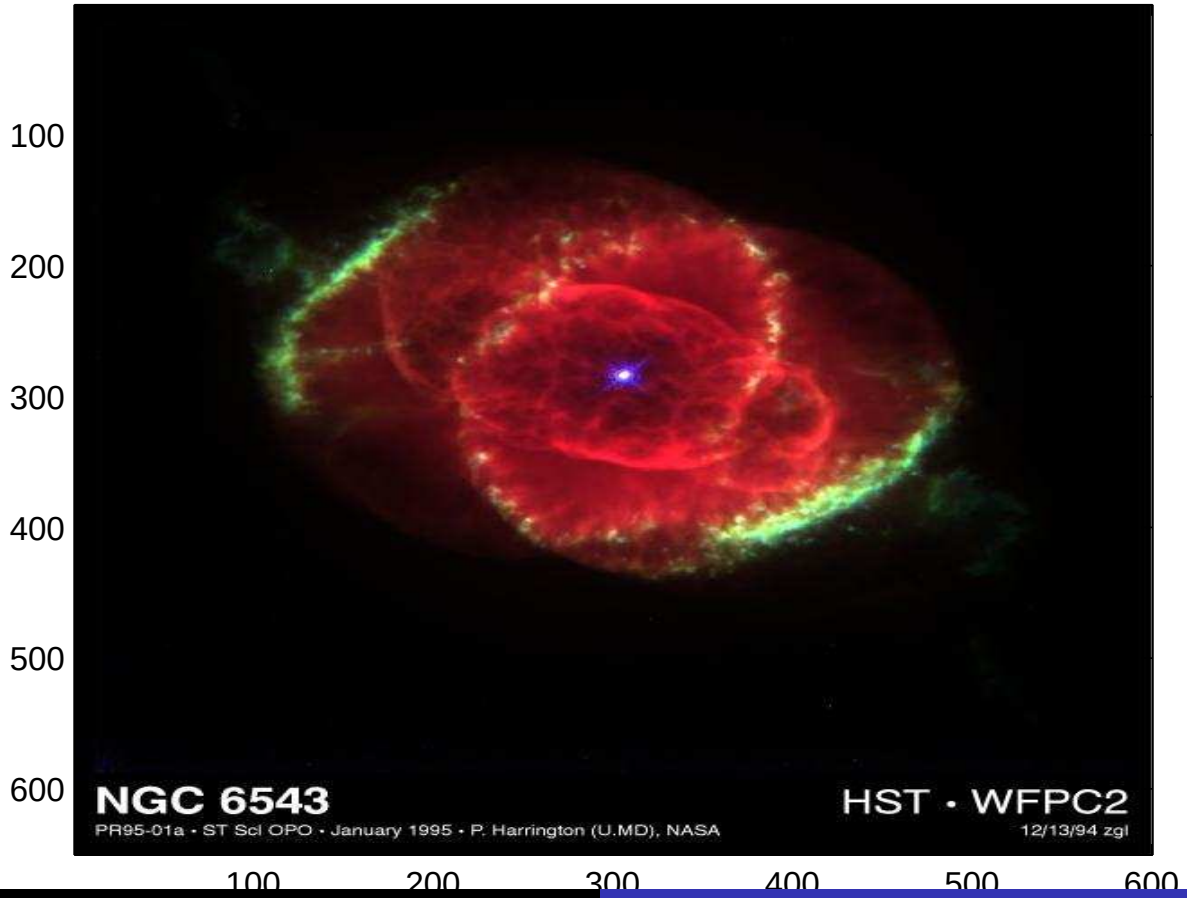
- $A =$**importdata**(*filename*) loads data into array A.
  - **importdata** can recognize the common file extension.
  - If not, **importdata** interprets the file as a delimited ASCII file.
- $A =$**importdata**($'-pastespecial'$) loads data from the system clipboard rather than from a file.
- Support file types:
  - MAT-files
  - ASCII files and spreadsheets
  - Images[4]
  - Audio files[5]

[4] Also see **imread**.

[5] Also see **audioread**.

Example

```
>> A=importdata('ngc6543a.jpg');
>> image(A);
```



Example

- Using a text editor, create a space-delimited ASCII file with column headers called myfile01.txt.

```
Day1   Day2   Day3   Day4   Day5   Day6   Day7
35.627 48.483 35.94  41.978 42.941 48.429 37.958
37.976 45.544 54.247 53.332 54.411 45.959 53.038
45.23  47.361 54.34  51.759 44.33 40.981 51.937
46.924 36.816 42.832 41.372 38.775 45.613 40.885
45.632 36.362 40.214 51.419 49.265 44.252 44.048
```

```
clear; clc;
% main
filename = 'myfile01.txt';
delimiter = ' ';
headerlines = 1; % starting line number
A = importdata(filename, delimiter, headerlines);
for k = [3,5]
    disp(A.colheaders{1, k}) % headers of columns
    disp(A.data(:, k)) % numeric data
end
```

- Note that A is a structure array. (Why?)

```
Day3
         35.94
        54.247
         54.34
        42.832
        40.214


Day5
        42.941
        54.411
         44.33
        38.775
        49.265
```

Access to Delimited Text Files

- **dlmread**(*filename*, *delimiter*) reads ASCII-delimited file of numeric data into matrix.
- **dlmwrite**(*filename*, M, *delimiter*) writes array M to the file, *filename*, using the specified delimiter, *delimiter*, to separate array elements.
  - **dlmwrite**(*filename*, M,$'-append'$) appends the data to the end of the existing file.
- **dlmread**(*filename*, *delimiter*, R, C) reads data whose upper left corner is at row R and column C in the file.
  - R and C start from 0. $(R, C) = (0, 0)$ specifies the first value in the file.

Example

```
>> M = gallery('integerdata', 100, [5 8], 0); % ...
    Help gallery
>> dlmwrite('myfile.txt', M, 'delimiter', '\t')
```

```
>> dlmread('myfile.txt')
>> dlmread('myfile.txt', '\t', 2, 3) % different?
```

textread

- **textread** is useful for reading text files with a known format.
- $[A, B, C, \ldots] =$**textread**$(\mathit{filename}, \mathit{format}, N)$ reads data from the file *filename* into the variables A, B, C, and so on, using the specified format, *format* for N lines.
  - *format* determines the number and types of return arguments. (See next page.)
  - If you drop N, then **textread** reads until the end of file.

format

- %n: read a number, integer or float
- %d: read as signed integer value.
- %f: read as a floating-point value.
- %s: read as a white-space or delimiter-separated string.
- %q: read as a double quoted string, ignoring the quotes.

Example

- Create a *mydata.dat* with the following content:

```
Sally   Level1 12.34 45 Yes
Arthur  Level2 19.85 29 No
```

```
>> [names, types, x, y, ...
    answer]=textread('mydata.dat','%s %s %f %d ...
    %s') % normal usage
```

```
>> [names, types, x, ...
    answer]=textread('mydata.dat','%s Level%d %f ...
    %*d %s', 1) % check the difference!
```

- In %*f, * ignores the matching characters specified by *.

Access to Excel Files

- **xlsread**(*filename*, *sheet*, *xlRange*) reads from the specified sheet and range.
  - *sheet* refers to the sheet name[6] or sheet number in the excel file.
  - *xlRange* refers to the rectangular portion of the worksheet to read.
  - For example, *xlRange* ='B:B' is used to import column B.
  - Another example[7] is that *xlRange* ='B1:B1' is used to import B1.
- **xlswrite**(*filename*, *A*, *sheet*, *xlRange*) writes array *A* to the specified sheet and range.

[6] The default sheet name is "工作表1".

[7] Contribution by Mr. Tsung-Yu Hsieh (MAT24409) on August 27, 2014.

Example

```
>> values={1, 2, 3; 4, 5, 'x'; 7, 8, 9};
>> headers={'First', 'Second', 'Third'};
>> xlswrite('myExample.xlsx', [headers; values]); ...
    % write
```

```
>> filename='myExample.xlsx';
>> sheet=1;
>> xlRange='B2:C3';
>> subsetA=xlsread(filename, sheet, xlRange) % read
>>

subsetA =

     2     3
     5   NaN
```

Low-Level File I/O

- Low-level file I/O functions allow the most control over reading or writing data to a file.
- However, these functions require that you specify more detailed information about your file than the easier-to-use high-level functions, such as **importdata**.
- If the high-level functions cannot import your data, you may consider to use low-level file I/O.
- The normal procedure looks like:
  1. Open a file.
  2. Read or write data into the file.
  3. Close the file.

Open Files

- *fid* = **fopen**(*filename*, *permission*) opens the file, *filename*, for binary read access, and returns an integer as *fid* equal to or greater than 3.
  - *fid* refers to the file identifier.
  - *permission*: file access type, specified as a string.
  - MATLAB reserves *fid*s 1 and 2 for standard output on the screen and standard error, respectively. (You will see later.)
  - If **fopen** fails to open the file with 'r', then *fid* is -1.
- You can **fopen** a new file with permission 'w'.
  - Be aware that you can repeatedly open a new file with the same file name.

fopen – Permission Codes

- 'r' is to open file for reading.
  - 'r+' additionally for writing.
- 'w' is to open or create new file for writing.[8]
  - 'w+' additionally for reading.
- 'a' is to open or create new file for writing. Also, it appends data to the end of the file.
  - 'a+' additionally for reading.

[8] Be aware that it will discard existing contents, if any.

Common Commands in Low-Level File I/O

- **feof**(*fid*), which refers to "end-of-file", returns 1 if a previous operation set the end-of-file indicator for the specified file.
- *tline* = **fgetl**(*fid*) returns the next line of the specified file, removing the newline characters.
  - *tline* is a text string unless the line contains only the end-of-file marker. In this case, *tline* is the numeric value -1.

Example

```
clear all;
clc
% main
fid=fopen('fgetl.m'); % open a file
tline=fgetl(fid);
while ischar(tline) % ischar tells if tline is a char
    disp(tline)
    tline=fgetl(fid);
end
fclose(fid);
```

Close Files

- **fclose**(*fid*) closes an open file.
- **fclose**('all') closes all open files.
  - You can also use **fclose** all.
- **fclose** returns a status of 0 when the close operation is successful.
  - Otherwise, it returns -1.