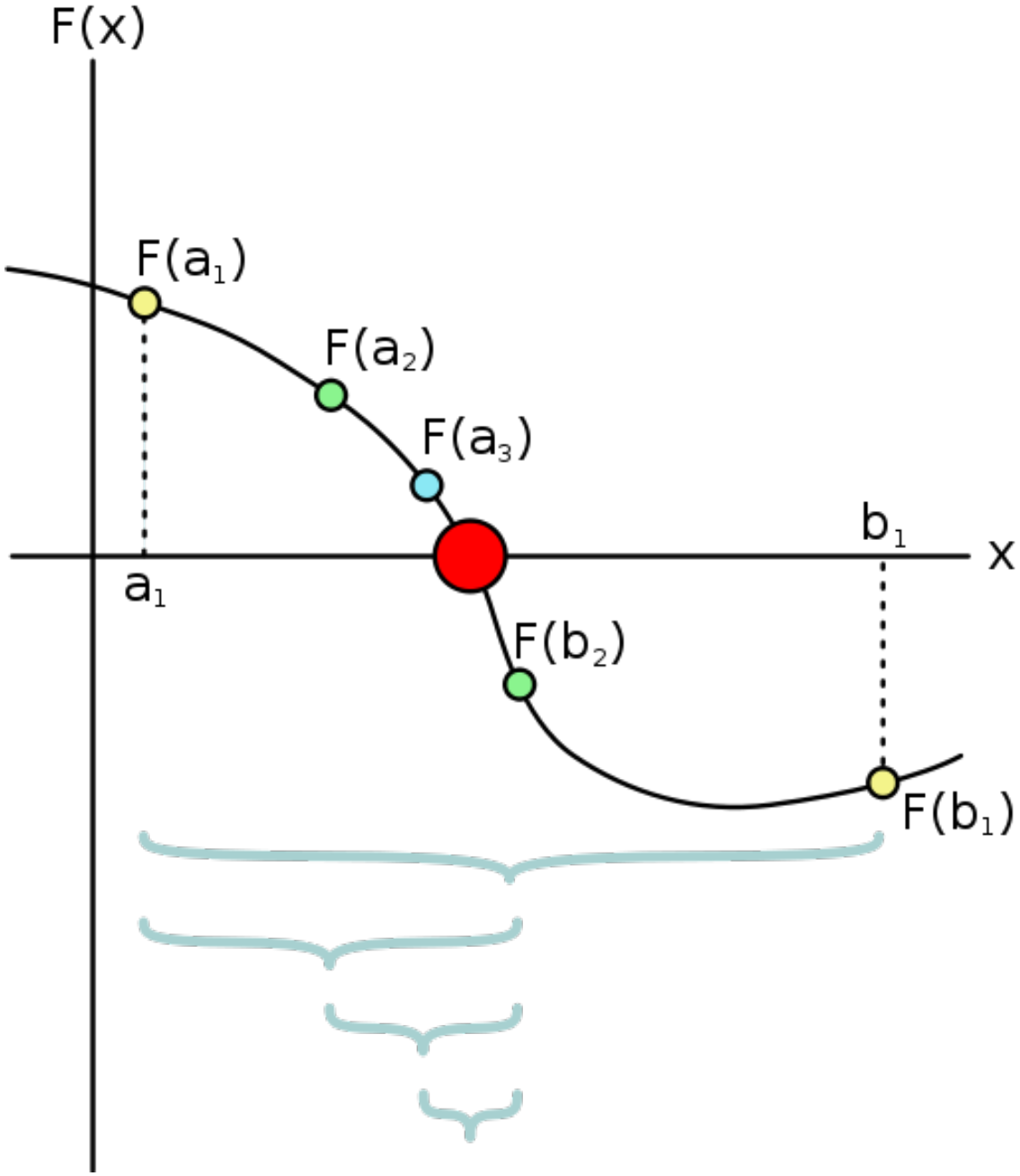
Example: Bisection Method

- The bisection method in mathematics is a root-finding method that repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. (Why?)
- It is often used to obtain an approximate solution.


F(x)
F(a1)
F(a2)
F(a3)
b1
x
a1
F(b2)
F(b1)

Idea of Bisection Method

1. At each step, the algorithm divides the interval in two by computing the midpoint $c = (a + b)/2$ of the interval and the value of the function $f(c)$ at that point.
2. Unless c is itself a root (which is very unlikely, but possible), there are now two possibilities:
   - either $f(a)$ and $f(c)$ have opposite signs and bracket a root,
   - or $f(c)$ and $f(b)$ have opposite signs and bracket a root.

3. The method selects the subinterval that is a bracket as a new interval to be used in the next step.
4. In this way the interval that contains a zero of f is reduced in width by $\frac{1}{2}$ at each step.
5. The process is continued until the interval is sufficiently small.[1]

[1] How small? A certain number you give is to be the stop criteria.

Problem Formulation

Input

- Endpoints of the interval $[a, b]$ for any real numbers $a < b$
- Minimal interval length $eps_{int} = |b - a|$
- Absolute error $eps_{abs} = |f(x_i) - 0|, i = a, b$

Output

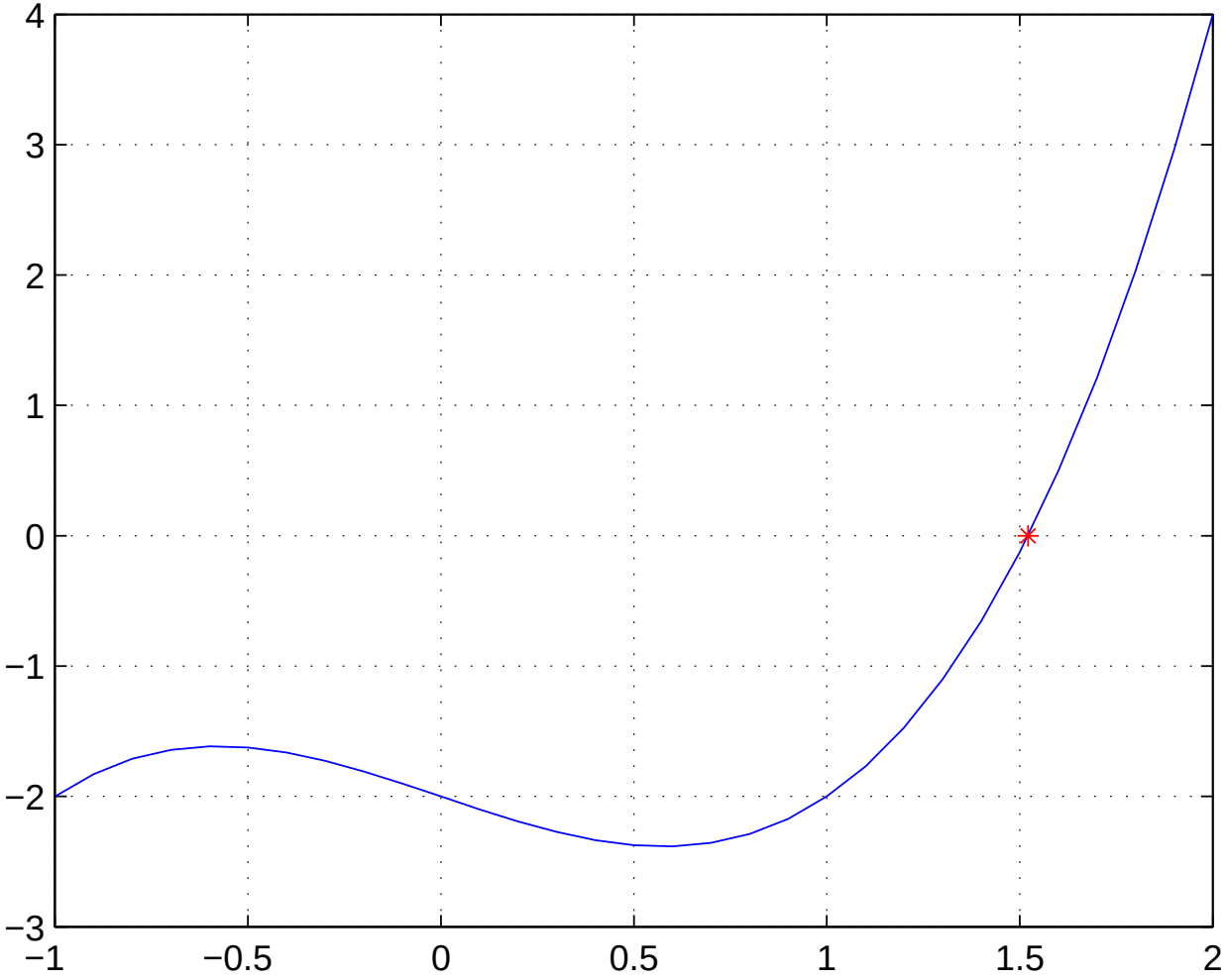
- the root r

- Note that a, b will be updated in each step.

Solution

```
clear; clc; format long;
% main: input parameters
disp('Root Finding using Bisection Method in [a,b].')
disp('f = x.^3-x-2'); % target function
a = input('a = ?\n');
b = input('b = ?\n');
f = @(x) x.^3-x-2; % target function
eps_int = 1e-5;
eps_abs = 1e-9;
%% main: algorithm
if f(a) == 0
    r = a; % lucky a
elseif f(b) == 0
    r = b; % lucky b
else
    while ( (b-a > eps_int) || ...
            (abs(f(a)-0) > eps_abs && abs(f(b)-0) ...
                > eps_abs ) )
```

```
        c = (a+b)/2; % middle point
        if (f(c) == 0)
            r = c; % lucky c
            break;
        elseif (f(a)*f(c) < 0)
            b = c;
        elseif (f(b)*f(c) < 0)
            a = c;
        else
            error('Failure: f(a)*f(c)>0 and ...
                f(c)*f(b)>0.');
            return; % terminate the program
        end
    end
    r = c % approximate solution which satisfies ...
        the criterion
end
```



$$c = 1.52137970691547$$

Remarks

- Time complexity: $O(log_2(n))$
  - What is n? $n = \dfrac{b-a}{eps_{int}}$.
  - This means that you need to make a trade-off between the numerical precision, that is, the number of digits, and the computation time.
- Be aware that this algorithm works well only with the premise that the behavior in $[a, b]$ is mild.
- Approximate solutions may be significantly influenced by the initial interval $[a, b]$.
- $f(c) \approx 0$ but not equal to exactly 0. (Why?)

Exercise

- Please make your bisection method algorithm into a user-defined function, say, **bisec**.
- So, you can call **bisec** to find a root for a specific function in the command window or other programs.
- Besides, you should extend the function with more input arguments for parameters used in **bisec**.

Problem Formulation

Input

- Target function f
- Endpoints of the interval $[a, b]$ for any real numbers $a < b$
- Minimal interval length $eps_{int} = |b - a|$
- Absolute error $eps_{abs} = |f(x_i) - 0|, x_i = a, b$

Output

- The root r

```
function r=bisec(f,a,b,eps_int,eps_abs)

...
```

```
>> bisec(@(x) x.^3-x-2,0,3,1e-9,1e-9)

ans =

    1.5214
```

Recursive Functions

- Recursion is when something is defined in terms of itself.
- A recursive function is a function that calls itself.
- Recursion is an alternative form of program control.
- It is essentially repetition without a loop.

Example

The factorial of a non-negative integer n, denoted by $n!$, is the product of all positive integers less than or equal to n.[2]

- For example,

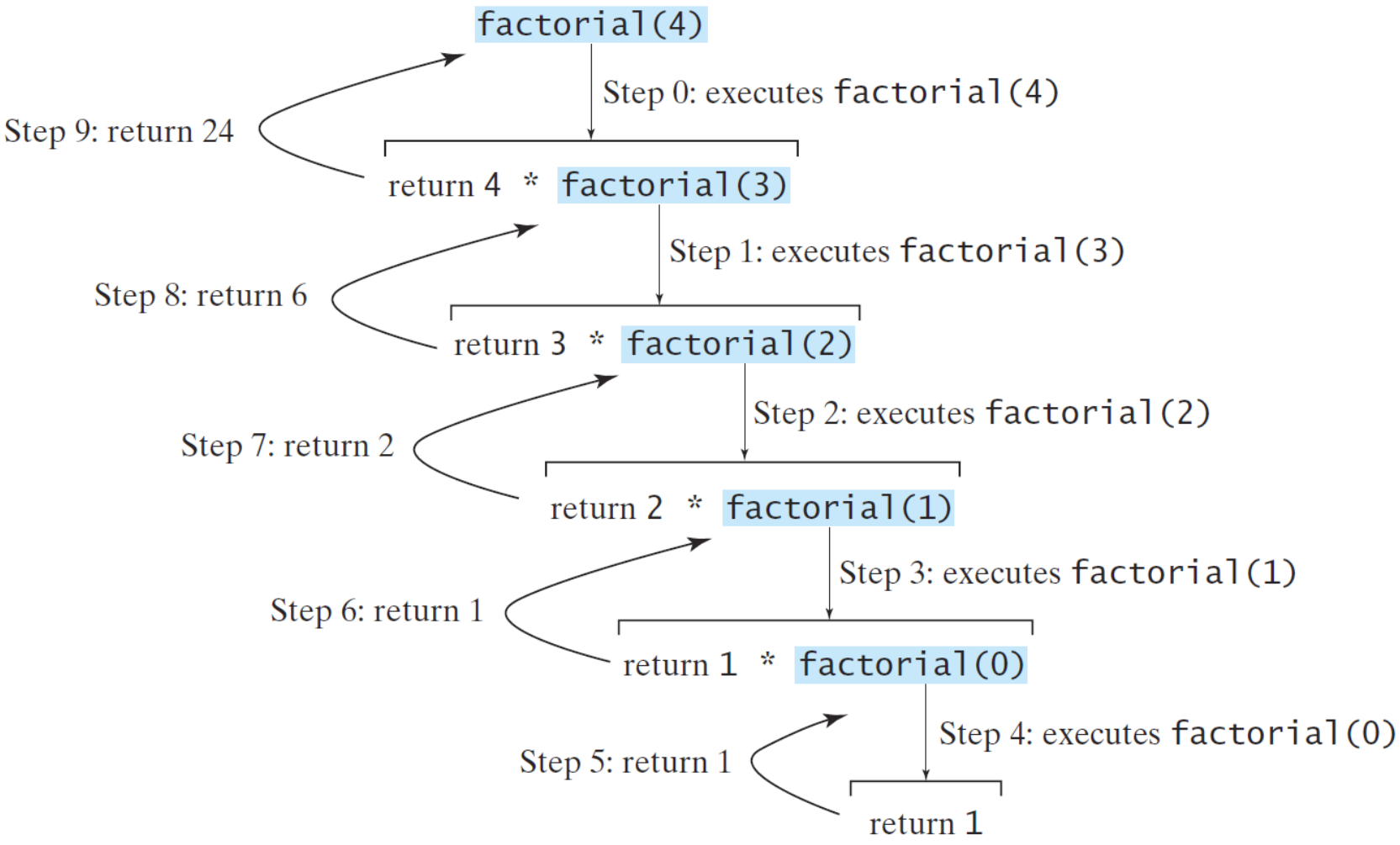
$$
\begin{aligned}
4! &= 4 \times 3! \\
&= 4 \times (3 \times 2!) \\
&= 4 \times (3 \times (2 \times 1!)) \\
&= 4 \times (3 \times (2 \times (1))) \\
&= 4 \times (3 \times (2)) \\
&= 4 \times (6) \\
&= 24.
\end{aligned}
$$

- Can you find the pattern?

[2] Note that $0! = 1$.

Divide and Conquer

- In general, to solve a problem using recursion, you break it into subproblems.
- Each subproblem is the same as the original problem but smaller in size.
- You can apply the same approach to each subproblem to solve it recursively.


factorial(4)
Step 0: executes factorial(4)
Step 9: return 24
return 4 * factorial(3)
Step 1: executes factorial(3)
Step 8: return 6
return 3 * factorial(2)
Step 2: executes factorial(2)
Step 7: return 2
return 2 * factorial(1)
Step 3: executes factorial(1)
Step 6: return 1
return 1 * factorial(0)
Step 4: executes factorial(0)
Step 5: return 1
return 1

Exercise

Write a program which determines return a factorial of n.

```
function y = recursiveFactorial(n)
    if n == 0 || n == 1 % base condition
        y = 1;
    else
        y = n * recursiveFactorial(n-1);
    end
end
```

- Note that there must be a base condition in recursion.
- So, can you replace a recursive method by a loop?

Equivalence: Recursion Replaced by Loops

```
function y = loopFactorial(n)
    y = 1;
    for i = 1 : n
        y = y * i;
    end
end
```

- One more intriguing question is, Can we always turn a recursive method into an iterative one?
- Yes, theoretically.[3]

[3] The Church-Turing thesis proves it if the memory serves.

Example: Fibonacci Numbers[4]

- The Fibonacci sequence is a sequence with the following pattern:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34 \ldots.$$

- Alternatively, the Fibonacci sequence is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2},$$

  where $n \geq 3$ and $F_1 = 0, F_2 = 1$.
- Given a positive integer n, determine F_n.

Input: $n \geq 1$
Output: F_n

---

[4] See Fibonacci numbers.

Solution

```
function y = recursiveFib(n)
    if n == 1
        y = 0;
    elseif n == 2
        y = 1;
    else
        y = recursiveFib(n-1) + recursiveFib(n-2);
    end
end
```

- It is a binary recursion, because there are two calls in the function.
- Time complexity: $O(2^n)$ (Why?)
- Can we implement a linear recursion for this problem?
- Also, you can implement it without recursive algorithms. (Try.)

Remarks

- Recursion bears substantial overhead.
- So, the recursive algorithm may execute a bit more slowly than the iterative equivalent.[5]
- Besides, a deeply recursive method depletes the call stack, which is limited, and causes a exception fast.[6]
- In practice, the decision whether to use recursion or iteration should be based on the nature of, and your understanding of, the problem you are trying to solve.

[5] In modern compiler design, recursion elimination is automatic if the recurrence occurs in the last statement in the recursive function, so called tail recursion.

[6] Stack overflow.

```
>> Lecture 4
>>
>>              -- Graphics
>>
```

Introduction

- Engineers use graphing techniques to make the information easier to understand.
- With a graph, it is easy to identify trends, pick out highs and lows, and isolate data points that may be measurement or calculation errors.
- Graphs can also be used as a quick check to determine whether a computer solution is yielding expected results.
- A set of ordered pairs is used to identify points on a two-dimensional graph.
- The data points are then connected by lines in common cases.

Simple *xy* Plots

- MATLAB provides the simplest function **plot**(x,y,'CLM') which is an overloaded function, where
  - (x,y): data set,
  - C: colors,
  - L: line types,
  - M: data markers.
- A graphic window automatically opens when **plot** is called.

About CLM

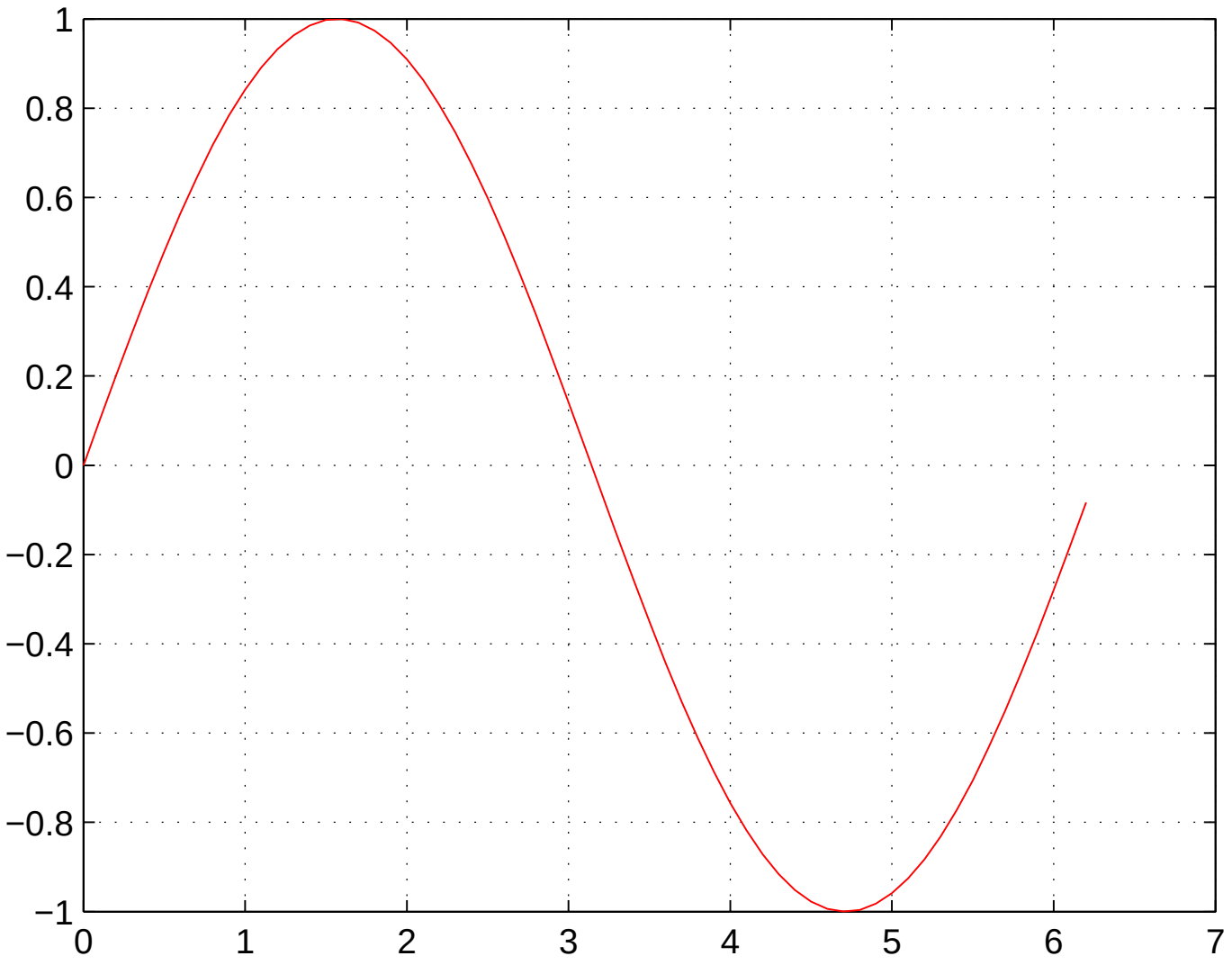
Data markers†		Line types		Colors	
Dot (·)	·	Solid line	-	Black	k
Asterisk (*)	*	Dashed line	- -	Blue	b
Cross (×)	×	Dash-dotted line	-.	Cyan	c
Circle (o)	o	Dotted line	:	Green	g
Plus sign (+)	+			Magenta	m
Square (□)	s			Red	r
Diamond (◇)	d			White	w
Five-pointed star (★)	p			Yellow	y

†Other data markers are available. Search for "markers" in MATLAB Help.

Example

```
clear all;
clc;
% main
x=0:0.1:2*pi;
y=sin(x);
figure(1); % create Figure 1
plot(x,y,'r-'); % red line
grid on; % show grid line
```

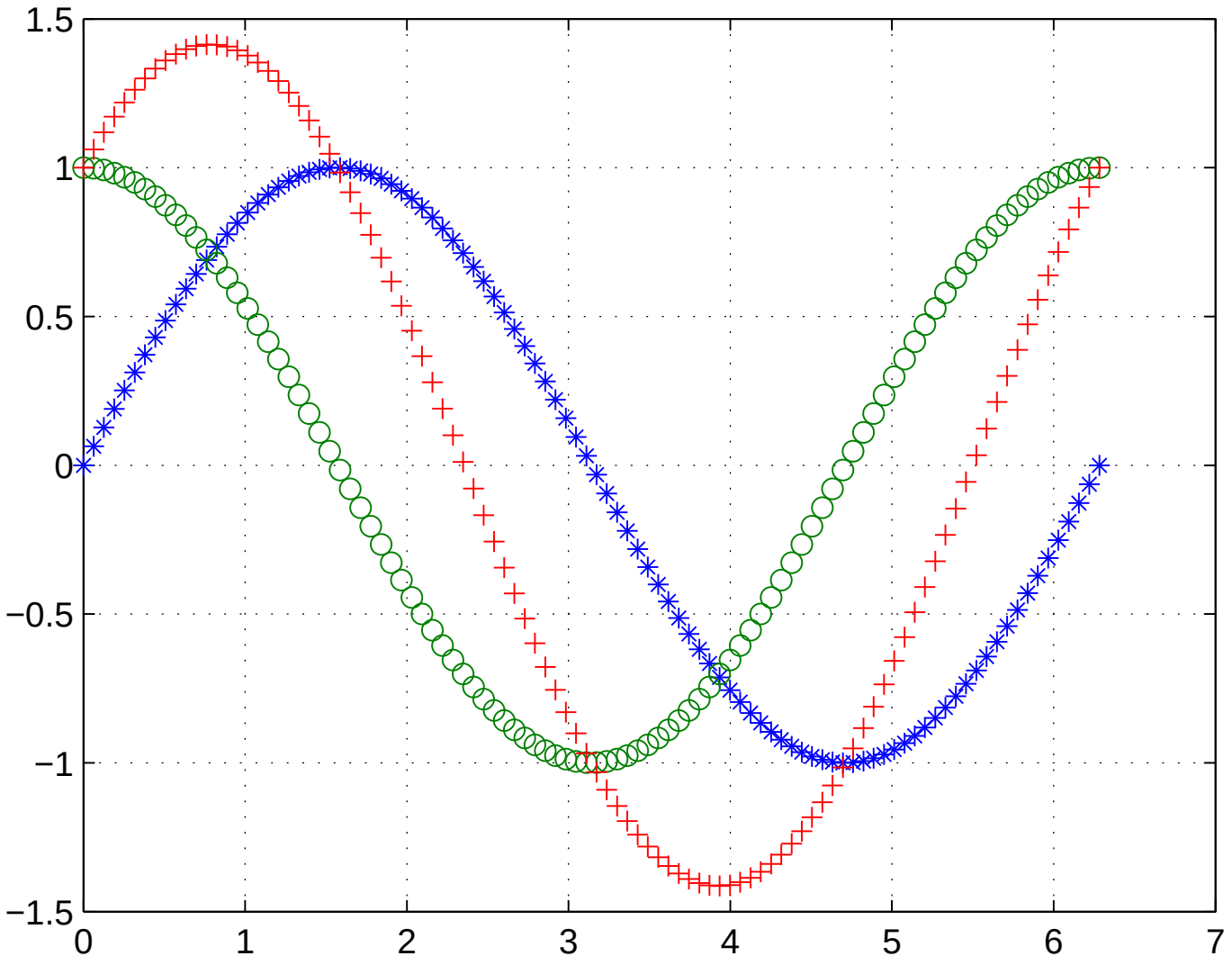
- Note that you can simply call **figure** without specifying a number.


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
1
2
3
4
5
6
7

Example 2: Multiple Curves

```
clear all;
clc;
% main
x=linspace(0,2*pi,100);
figure(2); % Create Figure 2
plot(x,sin(x),'*',x,cos(x),'o',x,sin(x)+cos(x),'+');
grid on;
```

- **plot**(x, A), where A is an m-by-n matrix and x is an array of size n, draws a figure for m lines. (Try.)

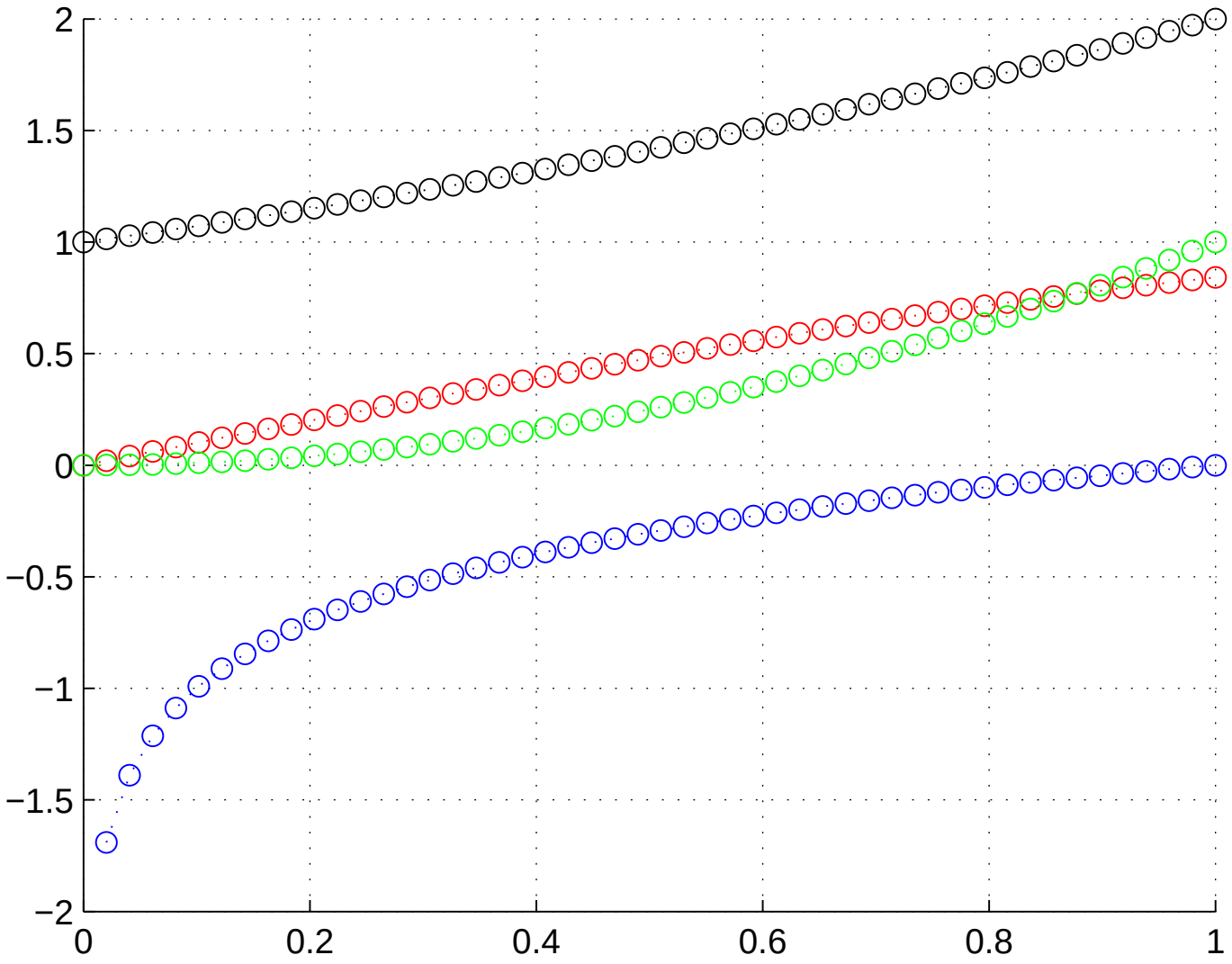

1.5
1
0.5
0
−0.5
−1
−1.5
0
1
2
3
4
5
6
7

Exercise

- Plot $sin(x)$, x^2, $log_{10}(x)$, and 2^x where $x \in [0, 1]$ with 100 points in a single figure.

```
clear; clc;

x = linspace(0,1,100);
y = [sin(x);
     x.^2;
     log10(x);
     2.^x];
C = 'rgby';
M = '.od+';
L = {'-','-.',':','--'};
for i = 1:4
    plot(x,y(i,:),[C(i) M(i) L{i}]);
end
```

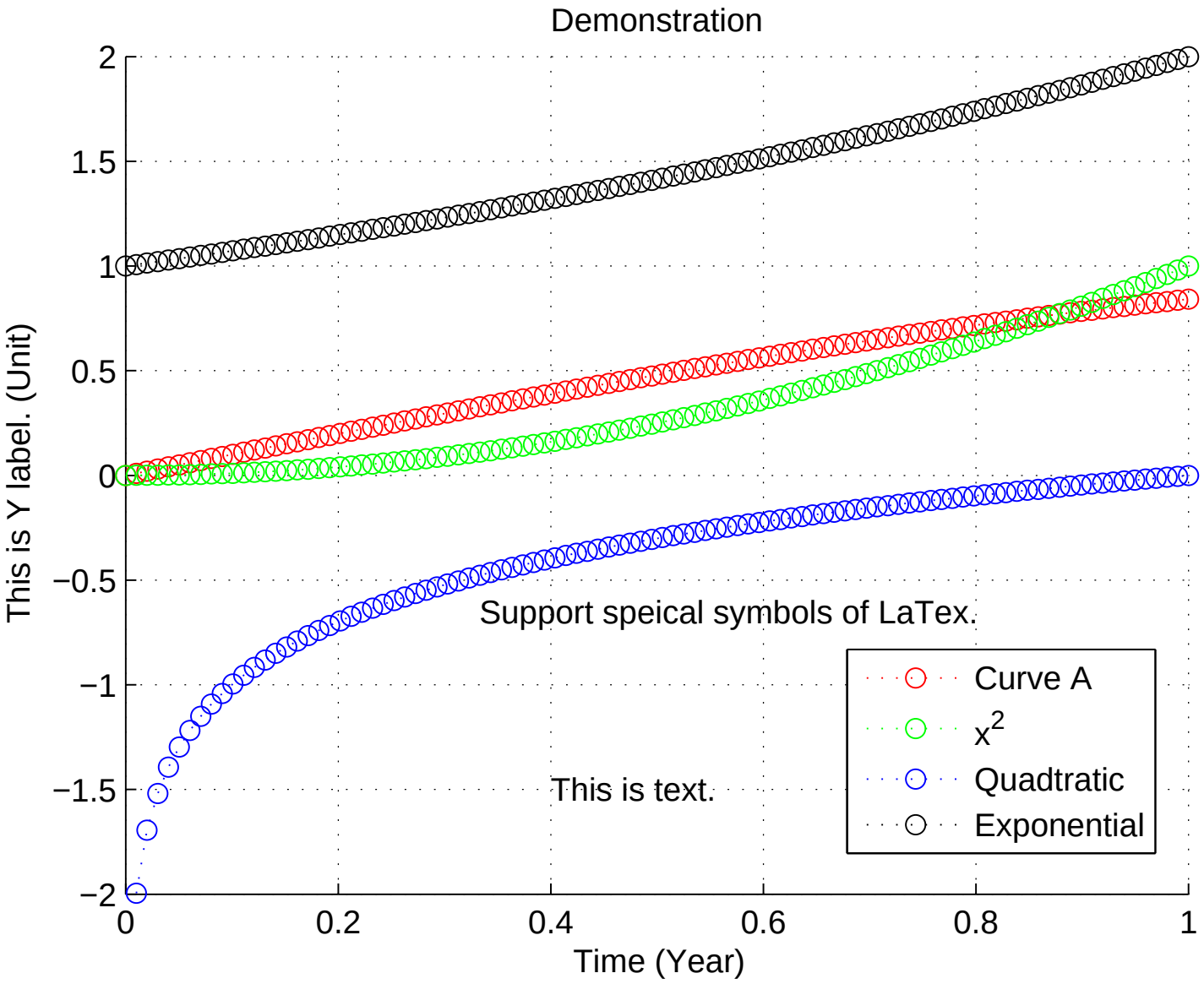

2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
0
0.2
0.4
0.6
0.8
1

Annotating Plots

- **title** adds a title to a plot.
- **xlabel** adds a label to the x-axis.
- **ylabel** adds a label to the y-axis.
- **legend** allows you to add a legend to your graph.
  - The legend shows a sample of the line and lists the string you have specified.
- **text** allows you to add a text box to the graph.
  - The box is placed at the specified x- and y- coordinates and contains the string value specified.
- **gtext** is similar to **text**.
  - The box is placed at a location determined interactively by the user by clicking in the figure window.

Example

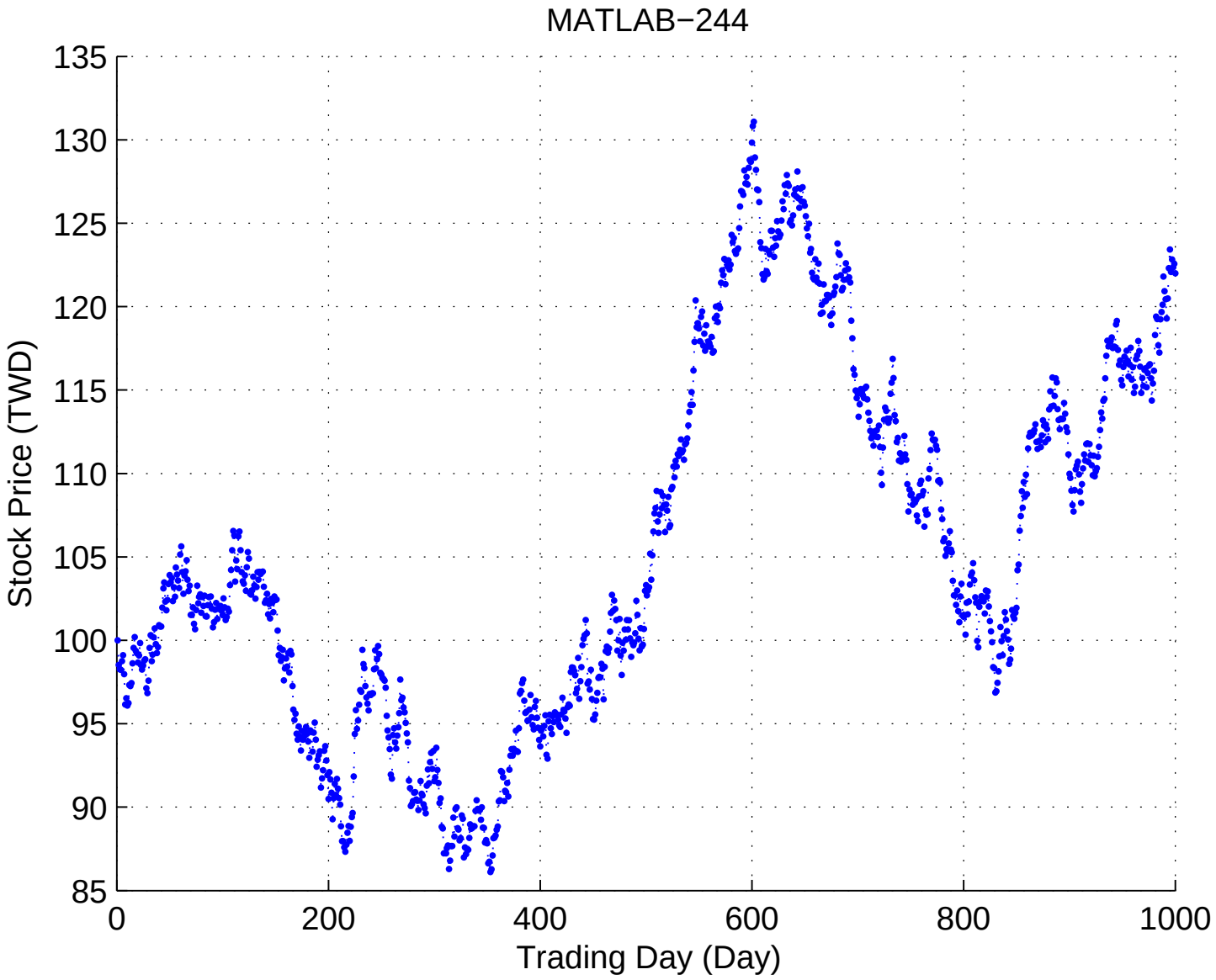
```
clear all;
clc;
% main
x = linspace(0,1,100);
y = [sin(x);x.^2;log10(x);2.^x];
color = 'rgbk';
figure(1);hold on; grid on;
for i = 1 : 4
    plot(x,y(i,:),[color(i) 'o:']);
end
title('Demonstration');
xlabel('Time (Year)');
ylabel('This is Y label. (Unit)');
legend({'Curve A', 'x^{2}', 'Quadtratic', ...
    'Exponential'});
text(0.4,-1.5,'This is text.');
gtext('Support special symbols of LaTex.');
```


Demonstration
This is Y label. (Unit)
Time (Year)
Support speical symbols of LaTex.
This is text.
Curve A
x2
Quadtratic
Exponential
2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
0
0.2
0.4
0.6
0.8
1

Exercise: Simulation on Stock Price[7]

```
clear; clc;
% main
y = zeros(1,1000);
y(1) = 100;
for i = 2 : length(y)
    y(i) = y(i - 1) + 10 * randn;
end
hold on;
plot(1 : length(y), y, '.:');
title('MATLAB-244');
ylabel('Stock Price (TWD)');
xlabel('Trading Day (Day)');
grid on;
```

[7] In financial theory, the stock price S is modelled by Black-Scholes formula, given by $dS = \mu S dt + \sigma S dW$, where μ is the annual yield, σ is the annual volatility, and dW follows a Brownian motion.


MATLAB−244
135
130
125
120
115
110
105
100
95
90
85
Stock Price (TWD)
0
200
400
600
800
1000
Trading Day (Day)