Introduction to MATLAB Programming

MATLAB Fundamentals

MATLAB Fundamentals

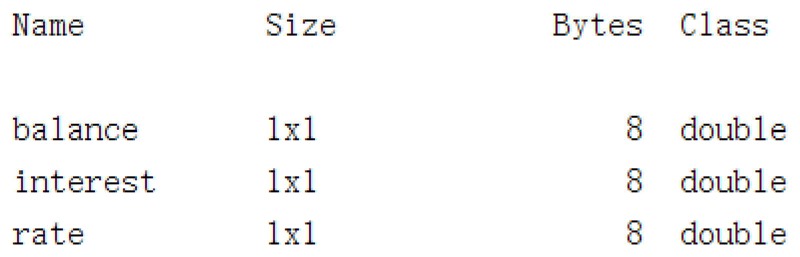
- Variables, operators, and expressions
- Arrays (including vectors and matrices)
- Basic input and output
- Repetition (for)
- Decision (if)

Variables

- A variable name must comply with the following two rules:
- It may consist only of the letters a-z, the digits 0-9, and the underscore(_)
- It must start with a letter
- The name may be as long as you like, but MATLAB only remembers the first 63 characters. (Check it with MATLAB built-in function namelengthmax)
  - Eg. R2d2 pay_day
  - 2a, name$, _2a
- Assigning a value to a
  - a=98
- In MATLAB, the variable name is case-sensitive
  - check balance, Balance, Balance

The workspace

- The function who lists the variables saved in the workspace
- All the variables that are created during the session in the workspace until you clear them
- The function ans returns the value of the last expression evaluated but not assigned to a variable
- The command whos lists the size of each variable as well



```
  Name           Size            Bytes  Class

  balance        1x1                 8  double
  interest       1x1                 8  double
  rate           1x1                 8  double
```

Each variable occupies eight bytes of storage. The Class double array means that the variable holds numeric values as double-precision floating-point.

Adding commonly used constants to the workspace

- If you often use the same physical or mathematical constants in your MATLAB sessions, you can save them in an M-file and run the file at the start of a session.
- For example, the following statements can be saved in myconst.m
  - g=9.8;   % acceleration due to gravity
  - avo=6.023e23;   %Avogadro’s number
  - bar_to_kP=101.325   % atmosphere to kiloPascals

Arrays : Vectors and Matrices

- Initializing vectors: Explicit lists

```
X=[1 3 0 -1 5]
disp(x)
whos
A=[5,6,7] %create array with commas
X=[130-15] or x=[130 -15]
A=[1 2 3]; b=[4 5]; c=[a –b]
a=[1 3 7]; a=[a 0 -1]
X=[ ]
```

Initializing vectors: The colon operator

- x=1:10
- x=1:0.5:4
- x=10:-1:1
- x=1:2:6
- x=0:-2:-5
- x=1:0

The linspace and logspace functions

The function linspace can be used to initialize a vector of equally spaced values

linspace(0, pi/2, 10)

Y=logspace(0, 2, 10)

Subscripts

- r=rand(1,7)
- r(3)
- r(2:4)
- r(1:2:7)
- r([1 7 2 6]
- r([1 7 2])=[ ]

Matrices

- a=[1 2 3; 4 5 6]
- x=0:30:180;
- Table=[x’ sin(x*pi/180)’]

Structure plan

- A structure plan is a top-down design of the steps required to solve a particular problem with a computer.
- It is typically written in what is called pseudo-code that is , statements in English, mathematics, and MATLAB describing in detail how to solve a problem.
- You don’t have to be an expert in any particular computer language to understand psedudo-code. A structure plan may be written at a number of levels, each of increasing complexity, as the logical structure of the program is developed

Structure plan

- Suppose we want to write a script to convert a temperature on the Fahrenheit scale to the Celsius scale. A first-level structure plan might be a simple statement of the problem:
   1. Initialize Fahrenheit temperature
   2. Calculate and display Celsius temperature
   3. Stop
- The second-level plan can be
   1. Initialize Fahrenheit temperature(F)
   2. Calculate and display Celsius temperature(C) as follows: Substract 32 from F and multiply by 5/9
   3. Display the value of C
   4. Stop

Structure plan

- Script to implement this as follows:

```
% Script file to convert temperatures from F to C
%
F=input('Temperature in degreee F:')
C=(F-32)*5/9;
disp(['Temperature in degrees C= ', num2str(C)])
%STOP
```

Structure plan: Vertical motion under gravity

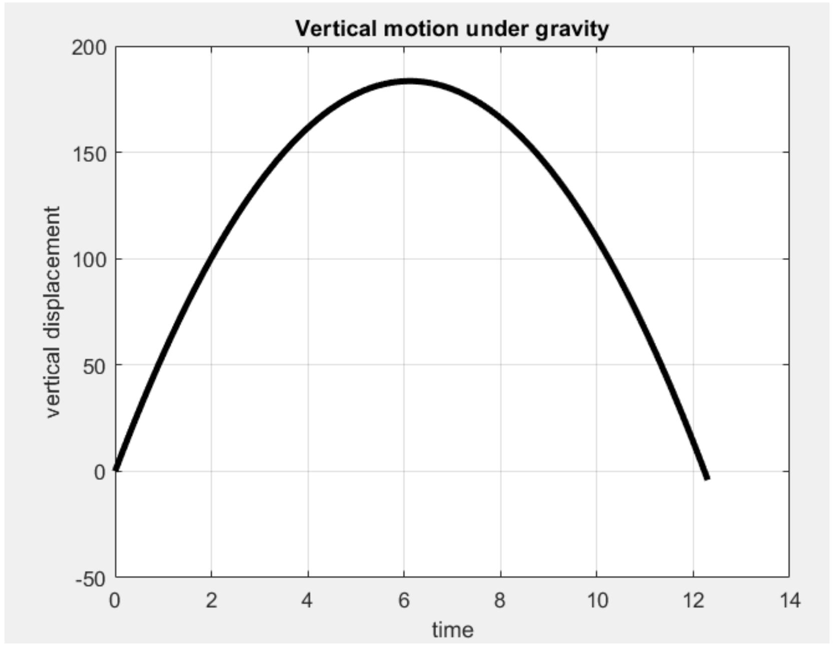
Question : If a stone is thrown vertically upward with an initial speed u, its vertical displacement s after an elapsed time t is given by the formula $s = ut - gt^2/2$, where g is the acceleration due to gravity. Air resistance is ignored. We would like to compute the value of s over a period of about 12.3s at intervals of 0.1s and plot the distance-time graph over this period.

- The structure plan for this problem is as follows:
    1. % Assign the data (g, u, and t) to MATLAB variables
    2. % Calculate the value of s according to the formula
    3. % Plot the graph of s against t
    4. % Stop

Structure plan: Vertical motion under gravity

- 1. Type the structure plan into the Editor
- 2. Paste a second copy of the plan directly below the first.
- 3. Translate each line in the second copy into a MATLAB statement or statements (add % comments)
- 4. Paste all the translated MATLAB statements into the Command Window an run them (or you can just click on the run icon in the toolbar of the Editor to execute your script)

```
% Vertical motion under gravity
g=9.81; % acceleration due to gravity
u=60;  % initial velocity in meter/sec
t=0:0.01:12.3; % time in seconds
s=u*t-g/2*t.^2; % vertical displacement in meters.
plot(t, s, 'k', 'LineWidth', 3)
title('Vertical motion under gravity')
xlabel('time');
ylabel('vertical displacement')
grid
```



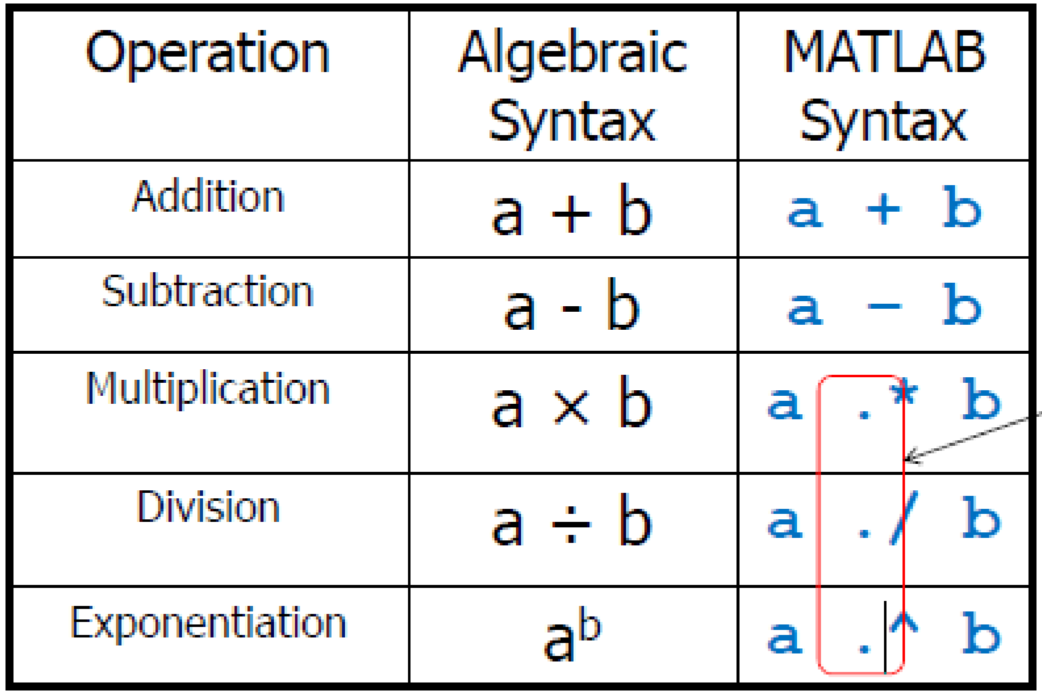
Operators, Expressions, and Statements

- What a program basically do is evaluating expressions

```
u*t-g/2*t.^2
balance=balance+interest
Iter=iter^2-1
```

- MATLAB is described as expression based language because it interprets and evaluates typed expression.
- Expressions are constructed from a variety of things such as numbers, variables, and operators.

Arithmetic operators

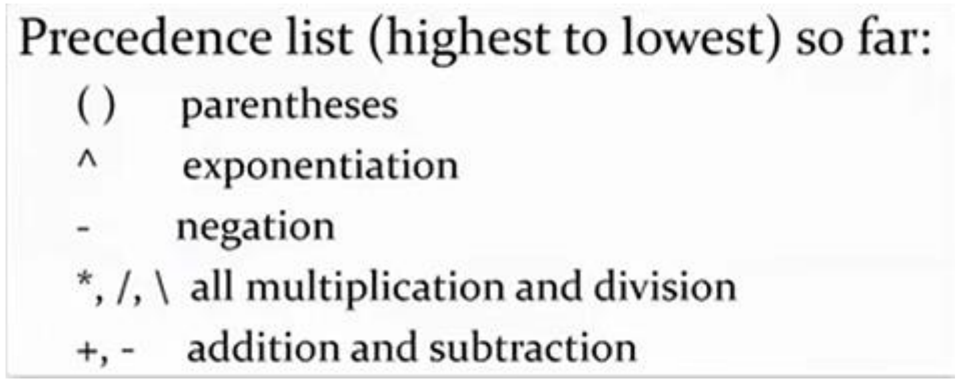


Operation	Algebraic Syntax	MATLAB Syntax
Addition	a + b	`a + b`
Subtraction	a - b	`a – b`
Multiplication	a × b	`a .* b`
Division	a ÷ b	`a ./ b`
Exponentiation	a^b	`a .^ b`

Operation is done to every value in the array by adding the period

Precedence of Arithmetic Operations

Basic operators
+ Addition
- Subtraction
*** Multiplication**
/ Right division (traditional)
\ Left division
^ Exponentiation
( ) Parenthesis



Precedence list (highest to lowest) so far:

( )	parentheses
^	exponentiation
-	negation
*, /, \	all multiplication and division
+, -	addition and subtraction

Practice

- Evaluate the following MATLAB expressions before checking the answers in MATLAB

```
1 + 2 * 3
4 / 2 * 2
1 + 2 / 4
1 + 2 \ 4
2 * 2 ^ 3
2 * 3 \ 3
2 ^ (1 + 2)/3
1/2e-1
```

Use MATLAB to evaluate the following expressions.

$\frac{1}{2\times 3}$

$2^{2\times 3}$

$1.5\times 10^{-4}+2.5\times 10^{-2}$

Statements

- Variable=expression
- The value of the expression on the right is assigned to the variable (s) on the left. Assignment always works in this direction.

```
x=29;
pi/2
a+b=c
```

- A statement that is too long to fit on one line may be continued to the next line with an ellipsis of at least three dots:

```
x=3*4-8…
/2^2
```

- Statements on the same line may be separated by commas.

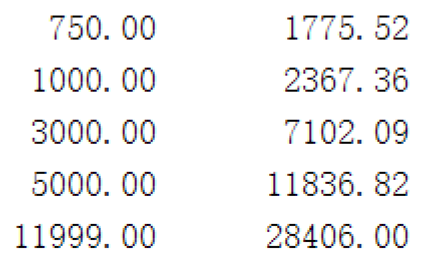
```
a=2; b=3; c==4
```

Formula vectorization

- With array operations, you can easily evaluate a formula repeatedly for a large set of data. This is one of MATLAB’s most useful and powerful features.

  An amount of money A invested over a period of years n with an annual interest rate r grows to an amount $A(1+r)^n$. Suppose we want to calculate final balances for investments of $750, $1000, $5000, and $11,999 over 10 years with an interest rate of 9%

```
format bank
A=[750 1000 3000 5000 11999];
r=0.09;

n=10;
B=A*(1+r)^n;
disp([A' B'])
```



```
  750.00      1775.52
 1000.00      2367.36
 3000.00      7102.09
 5000.00     11836.82
11999.00     28406.00
```

Question: Adjust the program to find the balances for a single amount A($1000) over 1, 5, 10, 15 and 20 years.

Output

- There are two straightforward ways of getting output from MATLAB:
  - Entering a variable name, assignment, or expression on the command line, without a semicolon
  - Using the disp statement

```
disp(variable)
disp('Pilate said, "What is truth?"');
X=2;
disp(['The answer is ', num2str(x)]);
disp([x y z])
```

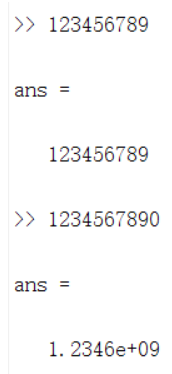
  - If you forget to switch on more before displaying a huge matrix, you can always stop the display with Ctrl+C

The format command

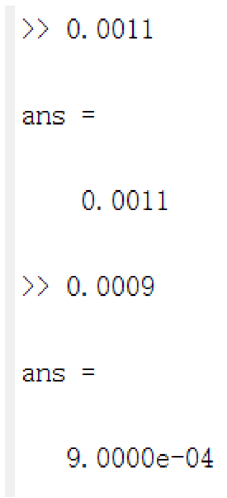
The term *format* refers to how something is laid out: in this case MATLAB output. The default format in MATLAB has the following basic output rules:

It always attempts to display integers (whole numbers) exactly. However, if the integer is too large, it is displayed in scientific notation with five significant digits—1234567890 is displayed as `1.2346e+009` (i.e., 1.2346×10^9).

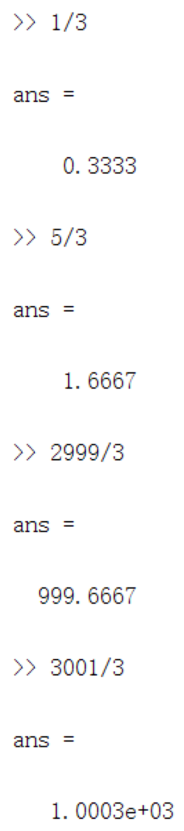
Numbers with decimal parts are displayed with four significant digits. If the value x is in the range $0.001 < x \leqslant 1000$, it is displayed in fixed-point form; otherwise, scientific (floating-point) notation is used, in which case the *mantissa* is between 1 and 9.9999 (e.g., 1000.1 is displayed as `1.0001e+003`).



```
>> 123456789

ans =

   123456789

>> 1234567890

ans =

   1. 2346e+09
```



```
>> 0. 0011

ans =

    0. 0011

>> 0. 0009

ans =

   9. 0000e-04
```

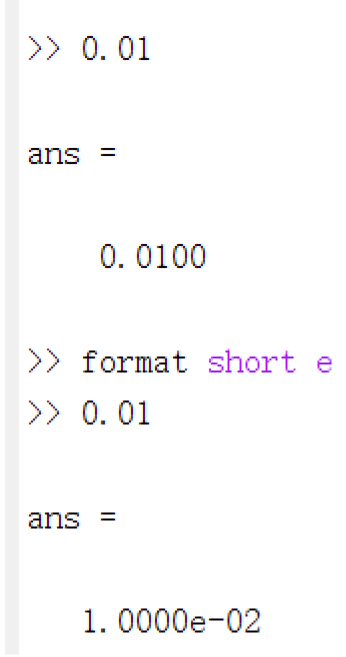


```
>> 1/3

ans =

    0. 3333

>> 5/3

ans =

    1. 6667

>> 2999/3

ans =

  999. 6667

>> 3001/3

ans =

   1. 0003e+03
```

The format command

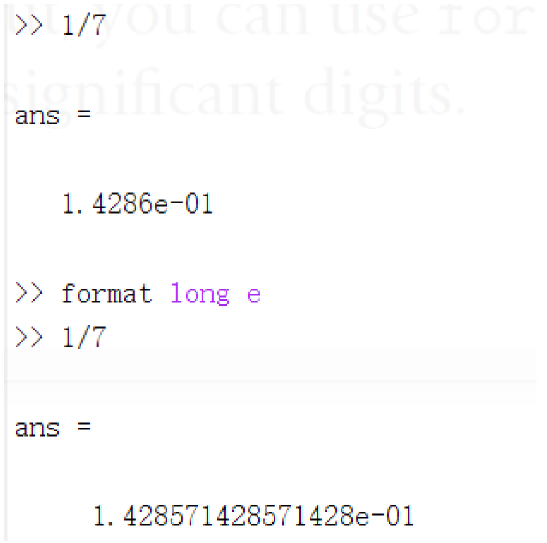
You can change from the default with variations on the `format` command, as follows. If you want values displayed in scientific notation (floating-point form) whatever their size, enter the command:

```
format short e
```



```
>> 0.01

ans =

    0.0100

>> format short e
>> 0.01

ans =

   1.0000e-02
```

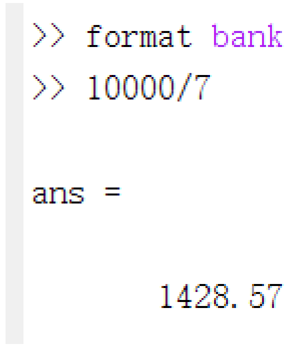
If you want more accurate output, you can use `format long e`. This also gives scientific notation but with 15 significant digits.



```
>> 1/7

ans =

   1.4286e-01

>> format long e
>> 1/7

ans =

    1.428571428571428e-01
```

Use `format bank` for financial calculations; you get fixed point with two decimal digits (for cents).

```
format compact
format loose
format rat
```



```
>> format bank
>> 10000/7

ans =

        1428.57
```

```
>> format compact
>> x = [1e3 1 1e-4]
 x =
   1.0e+003 *
     1.0000     0.0010     0.0000
```

```
>>format bank
>>x
x =
          1000.00            1.00            0.00
>>format short e
>>x
x =
   1.0000e+003    1.0000e+000    1.0000e-004
```

Repeating with for

```
for i = 1:5, disp(i), end
```

Now change it slightly to,

```
for i = 1:3, disp(i), end
```

And what about,

```
for i = 1:0, disp(i), end
```

Factorials!

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

```
n = 10;
fact = 1;
for k = 1:n
    fact = k * fact;
    disp( [k fact] )
end
```

Experiment to find the largest value of n for which MATLAB can find the n factorial. (You had better leave out the `disp` statement! Or you can move it from above the `end` command to below it.).

Limit of a sequence

$$x_n = \frac{a^n}{n!}, \quad n = 1, 2, 3, \ldots,$$

$$x_n = \frac{a x_{n-1}}{n}.$$

```
a = 10;
x = 1;
k = 20;                 % number of terms

for n = 1:k
  x = a * x / n;
  disp( [n x] )
end
```

The basic for construct(1)

In general the most common form of the `for` loop (for use in a program, not on the command line) is:

```
for index = j:k
    statements;
end
```

or

```
for index = j:m:k
    statements;
end
```

Note the following points carefully:

- $j\!:\!k$ is a vector with elements $j, j+1, j+2, \ldots, k$.
- $j\!:\!m\!:\!k$ is a vector with elements $j, j+m, j+2m, \ldots$, such that the last element does not exceed k if $m > 0$ or is not less than k if $m < 0$.
- *index* must be a variable. Each time through the loop it will contain the next element of the vector $j\!:\!k$ or $j\!:\!m\!:\!k$, and *statements* (there may be one or more) are carried out for each of these values.

The basic for construct (2)

If the `for` construct has the form:

```
for k = first:increment:last
```

The number of times the loop is executed may be calculated from the following equation:

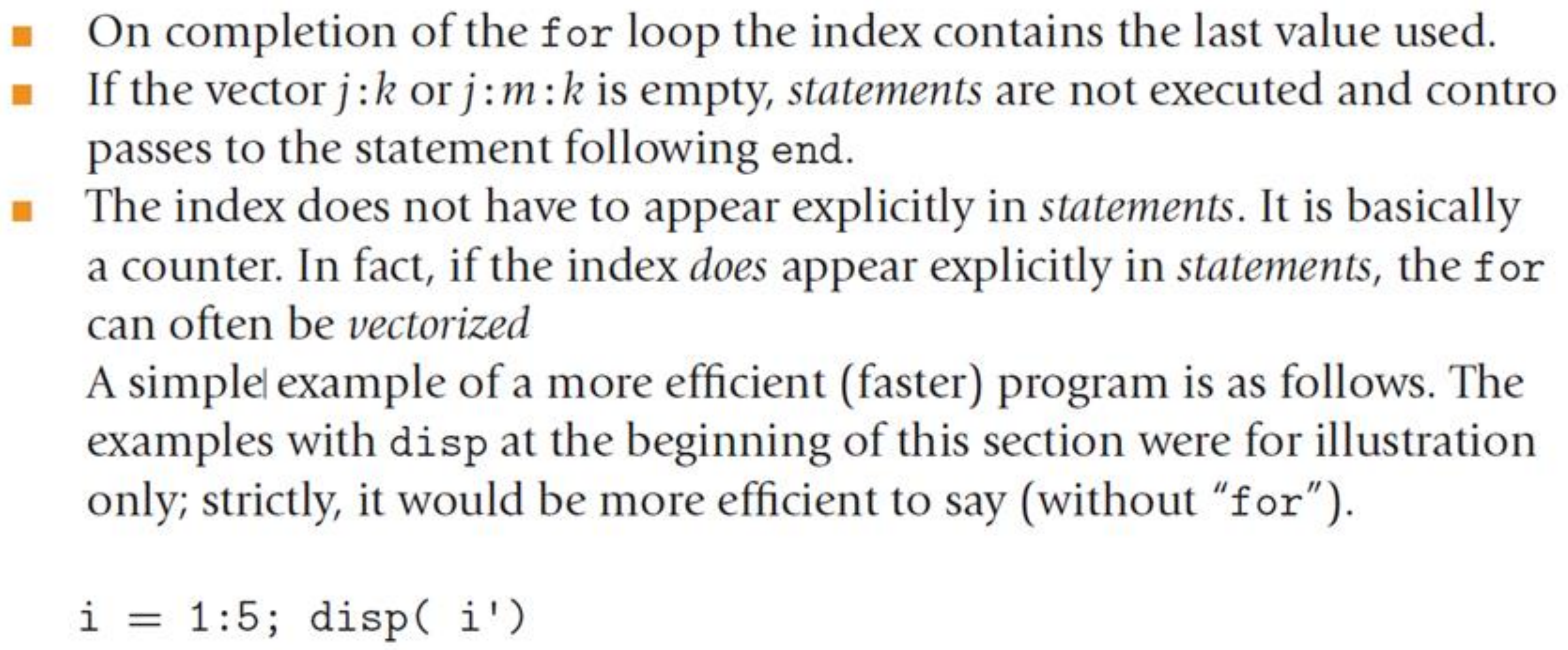
$$\texttt{floor}\left(\frac{\texttt{last}-\texttt{first}}{\texttt{increment}}\right)+1,$$

where the MATLAB function `floor (x)` rounds xdown toward $-\infty$. This value is called the iteration or trip count. As an example, let us consider the statement `for i = 1:2:6`. It has an iteration count of:

$$\texttt{floor}\left(\frac{6-1}{2}\right)+1=\texttt{floor}\left(\frac{5}{2}\right)+1=3.$$

Thus `i` takes the values 1, 3, 5. Note that if the iteration count is negative, the loop is not executed.

The basic for construct (3)



- On completion of the `for` loop the index contains the last value used.
- If the vector *j*:*k* or *j*:*m*:*k* is empty, *statements* are not executed and contro passes to the statement following `end`.
- The index does not have to appear explicitly in *statements*. It is basically a counter. In fact, if the index *does* appear explicitly in *statements*, the `for` can often be *vectorized*
  A simple example of a more efficient (faster) program is as follows. The examples with `disp` at the beginning of this section were for illustration only; strictly, it would be more efficient to say (without "`for`").

```
i = 1:5; disp( i')
```

for in a single line

```
for index = j:k; statements; end
```

or

```
for index = j:m:k; statements; end
```

A more general form of `for` is:

```
for index = v
```

where v is any vector. The index moves through each element of the vector in turn, providing a neat way of processing each item in a list.

Avoid for loops by vectorizing (1)

There are situations where a `for` loop is essential, as in many of the examples in this section so far. However, given the way MATLAB has been designed, `for` loops tend to be inefficient in terms of computing time. If you have written a `for` loop that involves the index of the loop in an expression, it may be possible to vectorize the expression, making use of array operations where necessary, as the following examples show.

Suppose you want to evaluate:

$$\sum_{n=1}^{1\,00\,000} n$$

(and can't remember the formula for the sum). Here's how to do it with a `for` loop (run the program, which also times how long it takes):

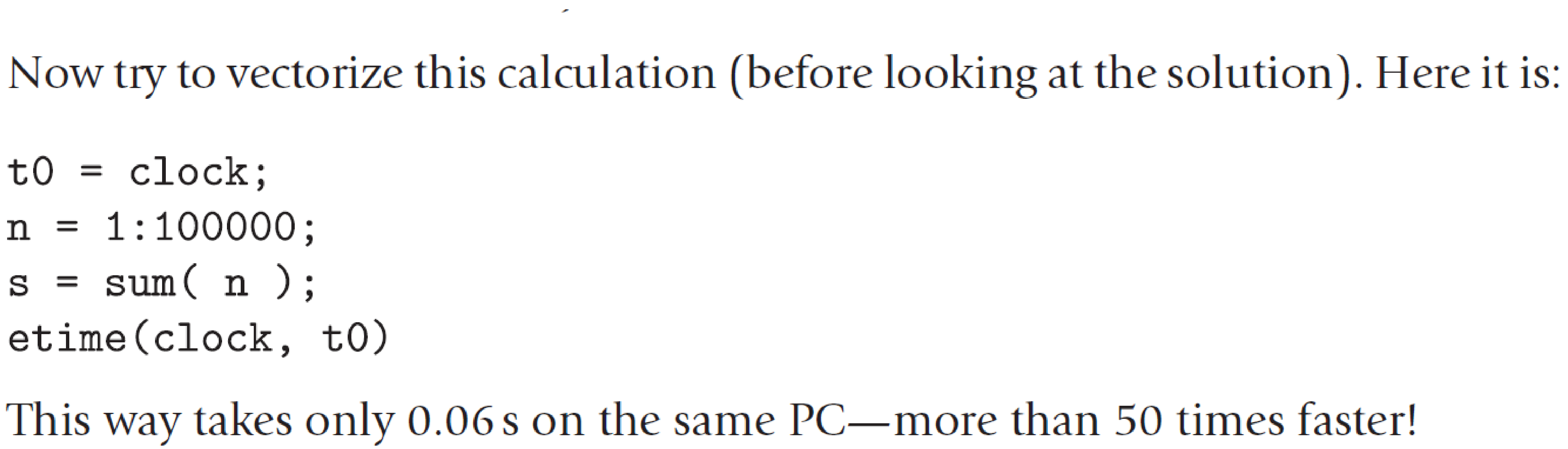
Avoid for loops by vectorizing (2)

```
t0 = clock;
s = 0;
for n = 1:100000
    s = s + n;
end
etime(clock, t0)
```

The MATLAB function `clock` returns a six-element vector with the current date and time in the format year, month, day, hour, minute, seconds. Thus, `t0` records when the calculation starts.

The function `etime` returns the time in seconds elapsed between its two arguments, which must be vectors as returned by `clock`.

Avoid for loops by vectorizing (2)



Now try to vectorize this calculation (before looking at the solution). Here it is:

```
t0 = clock;
n = 1:100000;
s = sum( n );
etime(clock, t0)
```

This way takes only 0.06 s on the same PC—more than 50 times faster!

Avoid for loops by vectorizing (3)

$$\sum_{n=1}^{1\,00\,000} \frac{1}{n^2}.$$

Here's the `for` loop version:

```
tic
s = 0;
for n = 1:100000
    s = s + 1/n^2;
end
toc
```

which takes about 6 s on the same PC. Once again, try to vectorize the sum:

```
tic
n = 1:100000;
s = sum( 1./n.^2 );
toc
```

Avoid for loops by vectorizing (4)

Series with alternating signs are a little more challenging.

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \cdots .$$

```
sign = -1;
s = 0;

for n = 1:9999
sign = -sign;
    s = s + sign / n;
end
```

The vectorized version is as follows:

```
n = 1:2:9999;
s = sum (1./n - 1./(n+1) )
```

Practice

Write MATLAB programs to find the following sums with `for` loops and by vectorization. Time both versions in each case.

- $1^2 + 2^2 + 3^2 + \cdots + 1000^2$ (sum is 333,833,500).
- $1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots - \frac{1}{1003}$ (sum is 0.7849—converges slowly to $\pi/4$).
- Sum the left-hand side of the series:

$$\frac{1}{1^2 \cdot 3^2} + \frac{1}{3^2 \cdot 5^2} + \frac{1}{5^2 \cdot 7^2} + \cdots = \frac{\pi^2 - 8}{16}$$ (sum is 0.1169 – with 500 terms).

Decisions

```
r = rand
if r > 0.5 disp( 'greater indeed' ), end
```

MATLAB should only display the message `greater indeed` if `r` is in fact greater than 0.5 (check by displaying `r`). Repeat a few times—cut and paste from the Command History window (make sure that a new `r` is generated each time).

The one-line if statement (1)

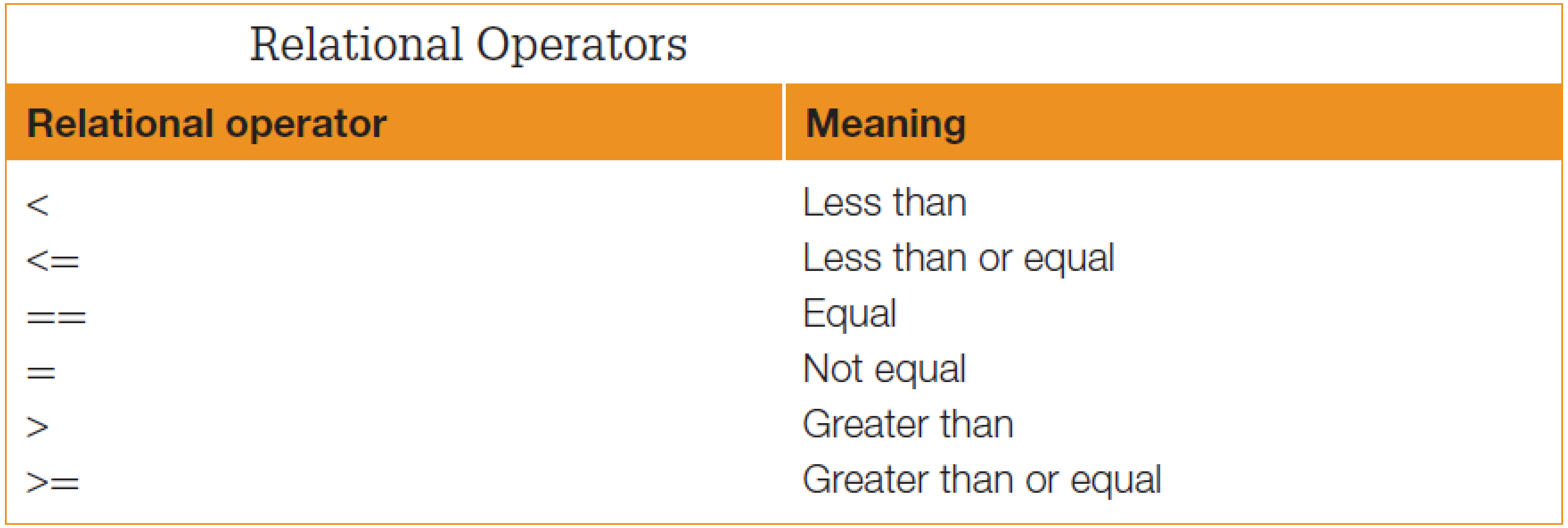
In the last example MATLAB has to make a decision; it must decide whether or not `r` is greater than 0.5. The `if` construct, which is fundamental to all computing languages, is the basis of such decision making. The simplest form of `if` in a single line is:

```
if condition; statements; end
```

- *condition* is usually a *logical expression* (i.e., it contains a *relational operator*), which is either *true* or *false*.
- If *condition* is true, *statement* is executed, but if *condition* is false, nothing happens.
- *condition* may be a vector or a matrix, in which case it is true only if *all* of its elements are nonzero. A single zero element in a vector or matrix renders it false:

```
    if condition statement, end
```

The one-line if statement (2)



Relational Operators

Relational operator	Meaning
<	Less than
<=	Less than or equal
==	Equal
=	Not equal
>	Greater than
>=	Greater than or equal

`b^2 < 4*a*c` $(b^2 < 4ac)$

`x >= 0` $(x \geq 0)$

`a ~= 0` $(a \neq 0)$

`b^2 == 4*a*c` $(b^2 = 4ac)$

```
if x == 0; disp( 'x equals zero' ); end
```

Exercise

The following statements all assign logical expressions to the variable x. See if you can correctly determine the value of x in each case before checking your answer with MATLAB.

(a) `x = 3 > 2`
(b) `x = 2 > 3`
(c) `x = -4 <= -3`
(d) `x = 1 < 1`
(e) `x = 2~= 2`
(f) `x = 3 == 3`
(g) `x = 0 < 0.5 < 1`

The if-else construct (1)

```
x = 2;
if x < 0 disp( 'neg' ), else disp( 'non-neg' ), end
if 79 disp( 'true' ), else disp( 'false' ), end

bal = 10000 * rand;

if bal < 5000
  rate = 0.09;
else
  rate = 0.12;
end

newbal = bal + rate * bal;
disp( ’New balance after interest compounded is:’ )
format bank
disp( newbal )
```

The if-else construct (2)

```
if condition
   statementsA
else
   statementsB
end
```

- *statementsA* and *statementsB* represent one or more statements.
- If *condition* is true, *statementsA* are executed, but if *condition* is false, *statementsB* are executed. This is essentially how you force MATLAB to choose between two alternatives.
- `else` is optional.

The one-line if-else statement

```
if condition; statementA; else; statementB; end
```

- Commas (or semicolons) are essential between the various clauses.
- `else` is optional.
- `end` is mandatory; without it, MATLAB will wait forever.

Elseif (1)

Suppose the Random Bank now offers 9% interest on balances of less than $5000, 12% for balances of $5000 or more but less than $10,000, and 15% for balances of $10,000 or more.

```
bal = 15000 * rand;

if bal < 5000
  rate = 0.09;
elseif bal < 10000
  rate = 0.12;
else
  rate = 0.15;
end

newbal = bal + rate * bal;
format bank
disp( 'New balance is:' )
disp( newbal )
```

Elseif (2)

```
if   condition1
     statementsA
elseif   condition2
     statementsB
elseif   condition3
     statementsC
...
else
     statementsE
end
```

1. *condition1* is tested. If it is true, *statementsA* are executed; MATLAB then moves to the next statement after `end`.
2. If *condition1* is false, MATLAB checks *condition2*. If it is true, *statementsB* are executed, followed by the statement after `end`.
3. In this way, all conditions are tested until a true one is found. As soon as a true condition is found, no further `elseifs` are examined and MATLAB jumps off the ladder.
4. If none of the conditions is true, *statements* after `else` are executed.
5. Arrange the logic so that not more than one of the conditions is true.
6. There can be any number of `elseifs`, but at most one `else`.
7. `elseif`*must* be written as one word.
8. It is good programming style to indent each group of statements as shown.

Logical operators

$ax^2 + bx + c = 0$

```
if (b ^ 2 - 4*a*c == 0) & (a ~= 0)
      x = -b / (2*a);
end
```

Multiple ifs versus elseif

```
bal = 15000 * rand;

if bal < 5000
    rate = 0.09;
end
if bal >= 5000 & bal < 10000
    rate = 0.12;
end
if bal >= 10000
    rate = 0.15;
end

newbal = bal + rate * bal;
format bank
disp( 'New balance is' )
disp( newbal )
```

Nested ifs

An `if` construct can contain further `if`s and so on. This is called *nesting* and should not be confused with the `elseif` ladder. You have to be careful with `else`s. In general, `else` belongs to the most recent `if` that has not been `end`ed.

```
...
d = b ^ 2 - 4*a*c;
if a ~= 0
  if d < 0
    disp( 'Complex roots' )
  else
    x1 = (-b + sqrt( d )) / (2*a);
    x2 = (-b - sqrt( d )) / (2*a);
  end    % first end  <<<<<<<<<<<<
end
```

```
d = b ^ 2 {-} 4*a*c;
if a ~ = 0
  if d < 0
    disp( 'Complex roots' )
  end     % first end moved up now   <<<<<<<<<<<<
  else
    x1 = (-b + sqrt( d )) / (2*a);
    x2 = (-b {-} sqrt( d )) / (2*a);

end
```

The switch statement

switch executes certain statements based on the value of a variable or expression.

Multiple expressions can be handled in a single case statement by enclosing the case expression in a cell array

```
d = floor(3*rand) + 1
switch d
case 1
    disp( 'That''s a 1!' );
case 2
    disp( 'That''s a 2!' );
otherwise
    disp( 'Must be 3!' );
end
```

```
d = floor(10*rand);
switch d
case {2, 4, 6, 8}
    disp( 'Even' );
case {1, 3, 5, 7, 9}
    disp( 'Odd' );
otherwise
    disp( 'Zero' );
end
```

Summary (1)

- The MATLAB desktop consists of a number of tools: the Command Window, the Workspace browser, the Current Directory browser, and the Command History window.
- MATLAB has a comprehensive online Help system. It can be accessed through the Help button (?) on the desktop toolbar or the **Help** menu in any tool.
- A MATLAB program can be written in the Editor and cut and pasted to the Command Window (or it can be executed from the editor by clicking the green right arrow in the toolbar at the top of the Editor window).
- A script file is a text file (created by the MATLAB Editor or any other text editor) containing a collection of MATLAB statements. In other words, it is a program. The statements are carried out when the script file name is entered at the prompt in the Command Window. A script file name must have the `.m` extension. Script files are therefore also called M-files.
- The recommended way to run a script is from the Current Directory browser. The output from the script will then appear in the Command Window.

Summary (2)

- A variable name consists only of letters, digits, and underscores, and must start with a letter. Only the first 63 characters are significant. MATLAB is case-sensitive by default. All variables created during a session remain in the workspace until removed with `clear`. The command `who` lists the variables in the workspace; `whos` gives their sizes.
- MATLAB refers to all variables as *arrays*, whether they are scalars (single-valued arrays), vectors, (1D arrays), or matrices (2D arrays).
- MATLAB names are case-sensitive.
- The Workspace browser on the desktop provides a handy visual representation of the workspace. Clicking a variable in it invokes the Array Editor, which may be used to view and change variable values.
- Vectors and matrices are entered in square brackets. Elements are separated by spaces or commas. Rows are separated by semicolons. The colon operator is used to generate vectors, with elements increasing (decreasing) by regular increments (decrements). Vectors are row vectors by default. Use the apostrophe transpose operator (`'`) to change a row vector into a column vector.
- An element of a vector is referred to by a subscript in parentheses. A subscript may itself be a vector. Subscripts always start at 1.

Summary (3)

- Statements on the same line may be separated by commas or semicolons.
- A statement may be continued to the next line with an ellipsis of at least three dots.
- Numbers may be represented in fixed-point decimal notation or in floating-point scientific notation.
- MATLAB has 14 data types. The default numeric type is double precision. All mathematical operations are carried out in double precision.
- The six arithmetic operators for scalars are `+`, `-`, `*`, `\`, `/`, and `^`. They operate according to rules of precedence.
- An expression is a rule for evaluating a formula using numbers, operators, variables, and functions. A semicolon after an expression suppresses display of its value.
- Array operations are element-by-element between vectors or between scalars and vectors. The array operations of multiplication, right and left division, and exponentiation are indicated by `.*`, `./`, `.\`, and `.^` to distinguish them from vector and matrix operations of the same name. They may be used to evaluate a formula repeatedly for some or all of the elements of a vector. This is called vectorization of the formula.
- `disp` is used to output (display) numbers and strings. `num2str` is useful with `disp` for displaying strings and numbers on the same line.

Summary (4)

- The `for` statement is used to repeat a group of statements a fixed number of times. If the index of a `for` statement is used in the expression being repeated, the expression can often be vectorized, saving a great deal of computing time.
- `tic` and `toc` may be used as a stopwatch.
- Logical expressions have the value true (1) or false (0) and are constructed with the six relational operators `>`, `>=`, `<`, `<=`, `==`, and `~=`. Any expression that evaluates to zero is regarded as false. Any other value is true. More complicated logical expressions can be formed from other logical expressions using the logical operators & (and), | (or), and ~ (not).
- `if--else` executes different groups of statements according to whether a logical expression is true or false. The `elseif` ladder is a good way to choose between a number of options, only one of which should be true at a time.
- `switch` enables choices to be made between discrete cases of a variable or expression.

ICE

Translate the following expressions into MATLAB:

(a) $p + \frac{w}{u}$

(b) $p + \frac{w}{u+v}$

(c) $\frac{p+\frac{w}{u+v}}{p+\frac{w}{u-v}}$

(d) $x^{1/2}$

(e) y^{y+z}

(f) x^{y^z}

(g) $(x^y)^z$

(h) $x - x^3/3! + x^5/5!$

Translate the following into MATLAB statements:

(a) Add 1 to the value of i and store the result in i.
(b) Cube i, add j to this, and store the result in i.
(c) Set g equal to the larger of the two variables e and f.
(d) If d is greater than 0, set x equal to –b.
(e) Divide the sum of a and b by the product of c and d, and store the result in x.

What's wrong with the following MATLAB statements?

(a) n + 1 = n;
(b) Fahrenheit temp = 9*C/5 + 32;
(c) 2 = x;

If C and F are Celsius and Fahrenheit temperatures, respectively, the formula for conversion from Celsius to Fahrenheit is $F=9C/5+32$.

(a) Write a script that will ask you for the Celsius temperature and display the Fahrenheit equivalent with some sort of comment, such as:

```
The Fahrenheit temperature is:...
```

Try it out on the following Celsius temperatures (answers in parentheses): 0 (32), 100 (212), −40 (−40!), 37 (normal human temperature: 98.6).

(b) Change the script to use vectors and array operations to compute the Fahrenheit equivalents of Celsius temperatures ranging from $20°$ to $30°$ in steps of $1°$, and display them in two columns with a heading, like this:

```
Celsius              Fahrenheit
20.00                68.00
21.00                69.80
...
30.00                86.00
```

Write a single statement to find and display the sum of the successive *even* integers 2, 4, …, 200. (Answer: 10,100)

Ten students in a class take a test. The marks are out of 10. All the marks are entered in a MATLAB vector, `marks`:

```
5    8    0    10   3    8    5    7    9    4    (Answer: 5.9)
```

Write a statement to find and display the average mark. Try it on the following:

What are the values of x and a after the following statements have been executed?

(a) a = 0;
(b) i = 1;
(c) x = 0;
(d) a = a + i;
(e) x = x + i / a;
(f) a = a + i;
(g) x = x + i / a;
(h) a = a + i;
(i) x = x + i / a;
(j) a = a + i;
(k) x = x + i / a;

- Rewrite the statements in more economically by using a for loop. Can you do even better by vectoring the code?

Work out by hand the output of the following script for n=4:

```
n = input( 'Number of terms? ' );
    s = 0;
for k = 1:n
   s = s + 1 / (k ^ 2);
end;
disp(sqrt(6 * s))
```

If you run this script for larger and larger values of n, you will find that the output approaches a well-known limit. Can you figure out what it is? Now rewrite the script using vectors and array operations.

Work through the following script by hand. Draw up a table of the values of `i`, `j`, and `m` to show how they change while the script executes. Check your answers by running the script:

```
v = [3 1 5];
  i = 1;

      for j = v
           i = i + 1;
           if i == 3
              i = i + 2;
              m = i + j;
           end
      end
```

The steady-state current I flowing in a circuit that contains a resistance $R=5$, capacitance $C=10$, and inductance $L=4$ in series is given by:

$$I = \frac{E}{\sqrt{R^2 + (2\pi\omega L - \frac{1}{2\pi\omega C})^2}},$$

where $E=2$ and $\omega=2$ are the input voltage and angular frequency, respectively. Compute the value of I. (Answer: 0.0396)

The electricity accounts of residents in a very small town are calculated as follows:

- If 500 units or fewer are\ used, the cost is 2 cents per unit.
- If more than 500 but not more than 1000 units are used, the cost is $10 for the first 500 units and 5 cents for every unit in excess of 500.
- If more than 1000 units are used, the cost is $35 for the first 1000 units plus 10 cents for every unit in excess of 1000.
- A basic service fee of $5 is charged, no matter how much electricity is used.Write a program that enters the following five consumptions into a vector and uses a `for` loop to calculate and display the total charge for each one: 200, 500, 700, 1000, 1500. (Answers: $9, $15, $25, $40, $90)

If you invest \$1000 for one year at an interest rate of 12%, the return is \$1120 at the end of the year. But if interest is compounded at the rate of 1% *monthly* (i.e., 1/12 of the annual rate), you get slightly more interest because it is compounded. Write a program that uses a `for` loop to compute the balance after a year of compounding interest in this way. The answer should be \$1126.83. Evaluate the formula for this result separately as a check: 1000×1.01^{12}.

A plumber opens a savings account with \$100,000 at the beginning of January. He then makes a deposit of \$1000 at the end of each month for the next 12 months (starting at the end of January). Interest is calculated and added to his account at the end of each month (before the \$1000 deposit is made). The monthly interest rate depends on the amount A in his account at the time interest is calculated, in the following way:

$$\begin{aligned} A \le 1\ 10\ 000 &: \quad 1\% \\ 1\ 10\ 000 < A \le 1\ 25\ 000 &: \quad 1.5\% \\ A > 1\ 25\ 000 &: \quad 2\% \end{aligned}$$

Write a program that displays, under suitable headings, for each of the 12 months, the situation at the end of the month as follows: the number of the month, the interest rate, the amount of interest, and the new balance. (Answer: Values in the last row of output should be 12, 0.02, 2534.58, 130263.78.)

It is useful to work out how the period of a bond repayment changes if you increase or decrease P. The formula for N is given by:

$$N = \frac{\ln\left(\frac{P}{P-rL/12^{-}}\right)}{12\ln(1+r/12)}.$$

(a) Write a new program to compute this formula. Use the built-in function `log` for the natural logarithm ln. How long will it take to pay off a loan of \$50,000 at \$800 a month if the interest remains at 15%? (Answer: 10.2 years—nearly twice as fast as when paying \$658 a month.)

(b) Use your program to find out by trial and error the smallest monthly payment that will pay off the loan this side of eternity. Hint: recall that it is not possible to find the logarithm of a