Introduction to Scientific and Engineering Computing, BIL108E

Karaman

INTRODUCTION TO SCIENTIFIC & ENGINEERING COMPUTING BIL 108E, CRN24023

Dr. S. Gökhan Karaman

Technical University of Istanbul

March 01, 2010

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

TENTATIVE SCHEDULE

Week	Date	Topics
1	Feb. 10	Introduction to Scientific and Engineering Computing
2	Feb. 17	Introduction to Program Computing Environment
3	Feb. 24	Variables, Operations and Simple Plot
4	Mar. 03	Algorithms and Logic Operators
5	Mar. 10	Flow Control, Errors and Source of Errors
6	Mar. 17	Functions
6	Mar. 20	Exam 1
7	Mar. 24	Arrays
8	Mar. 31	Solving of Simple Equations
9	Apr. 07	Polynomials Examples
10	Apr. 14	Applications of Curve Fitting
11	Apr. 21	Applications of Interpolation
11	Apr. 24	Exam 2
12	Apr. 28	Applications of Numerical Integration
13	May 05	Symbolic Mathematics
14	May 12	Ordinary Differential Equation (ODE) Solutions with Built-in Functions

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

LECTURE # 4

1. NUMERICAL DATA TYPES IN MATLAB
2. ERRORS
3. LOOPS
4. ALGORITHMS
5. DATA ANALYSIS

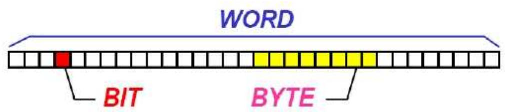
Introduction to Scientific and Engineering Computing, BIL108E

Karaman

BITS, BYTES AND WORDS

VARIABLES ARE REPRESENTED BY WORDS, COMPOSED OF BYTES, COMPOSED OF BITS

- **BIT** = elemental circuit, **ON (1) / OFF (0)**
- **BYTE** = string of **8** BITS
- **WORD** = string of **N** BYTES (partially controllable by the programmer)



BITS, BYTES AND WORDS

base 10	conversion	base 2
1	$1 = 2^0$	0000 0001
2	$2 = 2^1$	0000 0010
4	$4 = 2^2$	0000 0100
8	$8 = 2^3$	0000 1000
9	$8 + 1 = 2^3 + 2^0$	0000 1001
10	$8 + 2 = 2^3 + 2^1$	0000 1010
27	$16 + 8 + 2 + 1 = 2^4 + 2^3 + 2^1 + 2^0$	0001 1011

one byte = 8 bits

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

NUMERICAL DATA TYPES IN MATLAB

NUMERICAL DATA TYPES IN MATLAB

- int8(-128,127), int16(-32768, 32767), int32(- 2 147 483 648, 2 147 483 647), int64(-9 223 372 036 854 775 808, 9 223 372 036 854 775 807), uint8(0, 255), uint16(0, 65535), uint32(0, 4 294 967 295), uint64(0, 18 446 744 073 709 551 615)
- single(2^{-126}, 3.4×10^{38})
- double(2.2251×10^{-308}, 1.7977×10^{308})

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

NUMERICAL DATA TYPES IN MATLAB

REAL NUMBERS, FLOATING–POINT NUMBERS

- Real numbers, $\mathbb{R}$.
- Floating–point numbers, $\mathbb{F}$.

Only a subset $\mathbb{F}$ of finite dimension $\mathbb{R}$ can be represented.

Any real number x is truncated by the machine as $fl(x)$.

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

NUMERICAL DATA TYPES IN MATLAB

FLOATING–POINT NUMBERS

- Numeric values with non-zero fractional parts are stored as floating point numbers.
- All floating point values are represented with a normalized scientific notation.

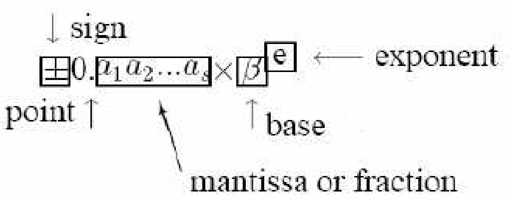
NUMERICAL DATA TYPES IN MATLAB

FLOATING–POINT NUMBERS

Floating–point number representation of a real number

$x = (-1)^s \times (0.a_1 a_2 a_3 \ldots a_t) \times \beta^e$

$a_1 \neq 0$



Introduction to Scientific and Engineering Computing, BIL108E
Karaman

NUMERICAL DATA TYPES IN MATLAB

EXAMPLES;

$12.7887 = 0.127887 \times 10^2$ (base 10)

$-0.099 = -0.99 \times 10^{-1}$ (base 10)

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

DIGITAL STORAGE OF INTEGERS

DIGITAL STORAGE OF INTEGERS

- Integers can be exactly represented by base 2
- Typical size is 16 bits
- 32 bit and larger integers are available

Note: All standard mathematical calculations in Matlab use floating point numbers.

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

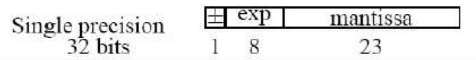
DIGITAL STORAGE OF NON–INTEGER NUMBERS

- Floating point values have fixed number of bits allocated for storage of the mantissa and fixed number of bits allocated for storage of the exponent.
- Two common precisions are provided in numerical computing: single precision and double precision.
- Fixed number of bits are allocated to each number: single precision uses 32 bits per floating point number and double precision uses 64 bits per floating point number

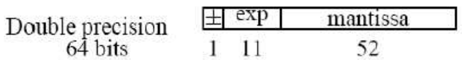
IEEE STANDARD

Total number of bits are split into separate storage for both the mantissa and the exponent.

- single precision: 1 sign bit, 8 bit exponent, 23 bit mantissa



- double precision: 1 sign bit, 11 bit exponent, 52 bit mantissa



DIGITAL STORAGE OF NUMBERS

- Limiting the number of bits allocated for storage of the exponent means that there are upper and lower limits on the magnitude of floating point numbers
- Limiting the number of bits allocated for storage of the mantissa means that there is a limit to the precision (number of significant digits) for any floating point number.

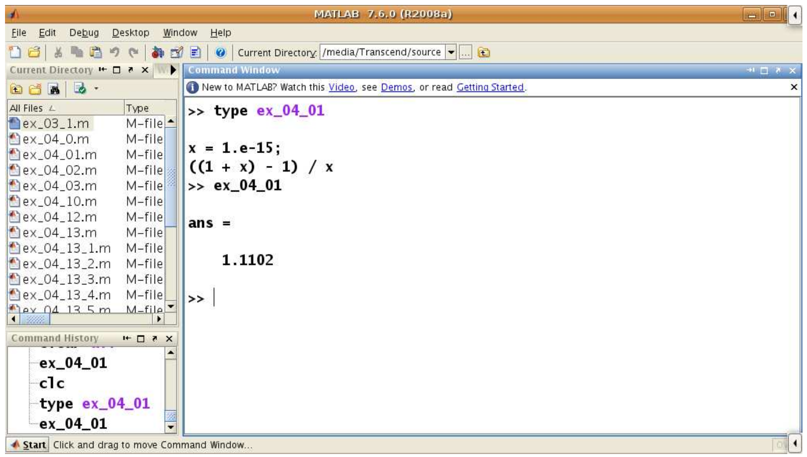
ERRORS

1. PP Physical Problem
2. MP Mathematical Problem
3. NP Numerical Problem

Each of these steps involve errors.

ERRORS

EXAMPLE:



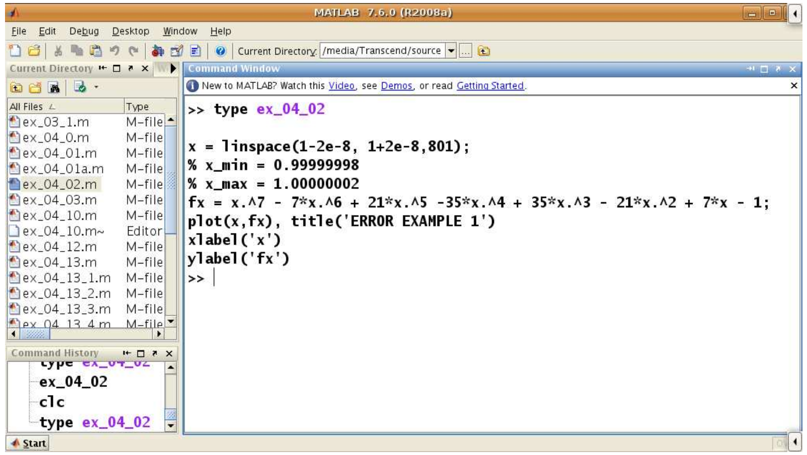
EXAMPLE:

Computational Errors

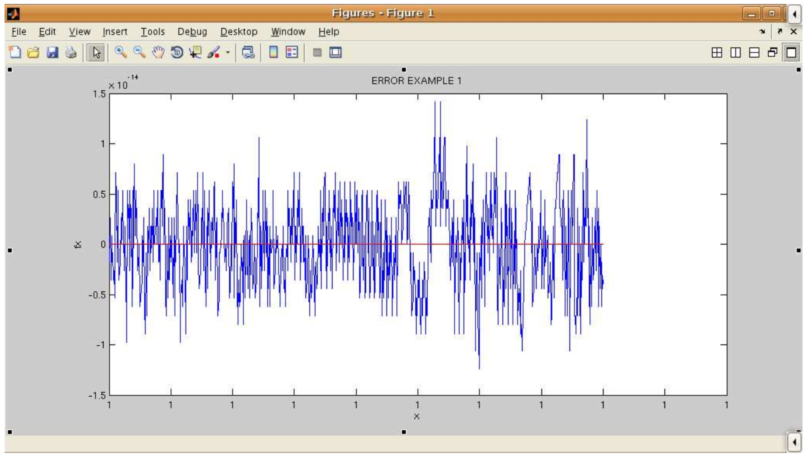
$f(x) = (x-1)^7$

$f(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$

EXAMPLE:



EXAMPLE:



EXAMPLE:

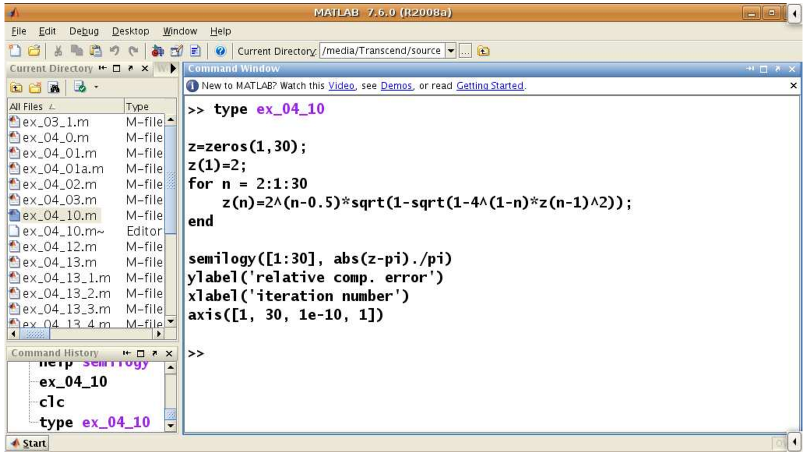
Calculation of *pi*

$z_2 = 2,\ z_{n+1} = 2^{n-1/2}\sqrt{1-\sqrt{1-4^{1-n}z_n^2}}$

$n = 2, 3, \ldots$

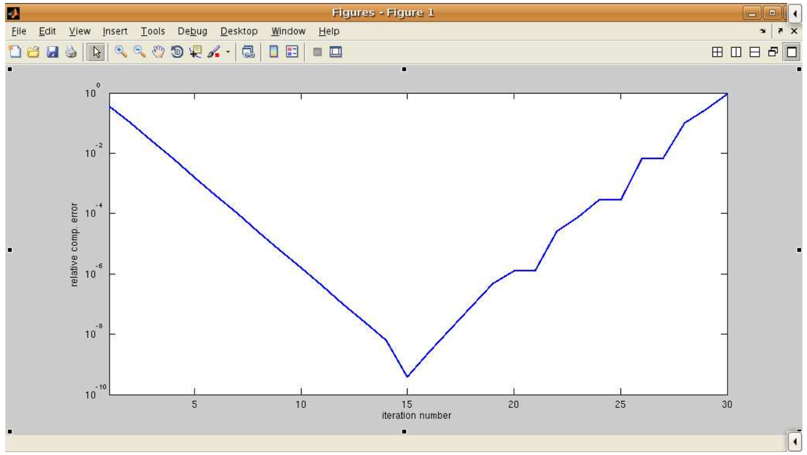
ERRORS

EXAMPLE:



ERRORS

EXAMPLE:



ERRORS

COMPUTATIONAL ERRORS

$\mathbf{e_c}$: computational Error

$\mathbf{x}$: exact solution of mathematical model

$\hat{\mathbf{x}}$: numerical solution of mathematical model

Absolute Computational Error

$$\mathbf{e_c^{abs}} = |\mathbf{x} - \hat{\mathbf{x}}|$$

Relative Computational Error

$$\mathbf{e_c^{rel}} = |\mathbf{x} - \hat{\mathbf{x}}|/|\mathbf{x}|$$

ERRORS

ERRORS RESULTING FROM PROBLEMS

- SYNTAX ERRORS
- LOGIC ERRORS
- ROUNDOFF ERRORS

ERRORS

SYNTAX ERRORS

- Typo errors.
- Incompatible vector sizes.
- Name hiding (try `"help command"`).

ERRORS

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

LOGIC ERRORS

- Try to run the program for some special cases where you know the answer.
- If you don't know any exact answer, use your insight to check whether the answer seems to be of the right order of magnitude.
- Try working through the program by hand to see if you can spot where things start going wrong.

ERRORS

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

ROUNDING ERRORS

- Finite-precision leads round-off in individual calculations
- Effects of round-off accumulate slowly
- The round-off errors are inevitable, solution is to create better algorithms
- Subtracting nearly equal may lead to severe loss of precision

ERRORS

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

MACHINE PRECISION

The magnitude of roundoff errors is quantified by machine precision ϵ_M

There is a number, ϵ_M such that

$$1 + \delta = 1$$

whenever $\delta < \epsilon_M$

In exact arithmetics, ϵ_M is identically 0.

eps $= 2.2204 \times 10^{-16}$ in Matlab

FLOATING–POINT NUMBERS

Roundoff–Error

$$\frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\epsilon_M$$

Ref: Standard for Floating Point Arithmetic P754, IEEE.
$\epsilon_M = \beta^{1-t}$, here t is the distance between 1 and its closest floating–point number greater than 1.

In Matlab ϵ_M is obtained through the command eps.
Number 0 does not belong to $\mathbb{F}$

FLOATING–POINT NUMBERS

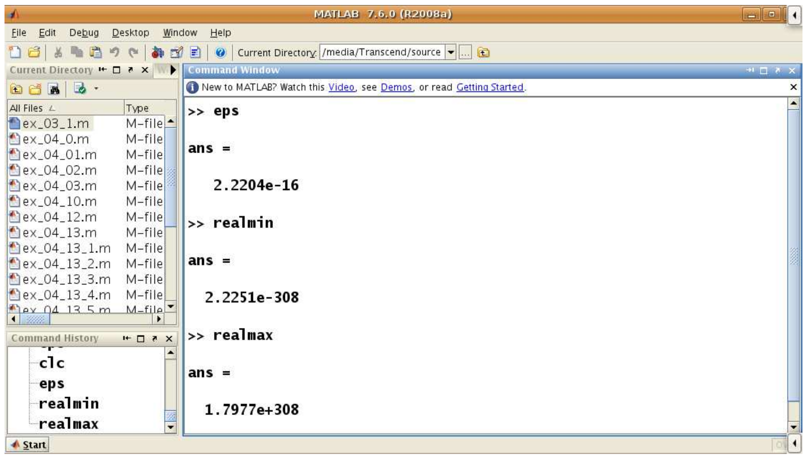
`realmin`, and `realmax`

If x is less than x_{min} is treated as 0, UNDERFLOW

If x is greater than x_{max} `Inf` OVERFLOW

The elements in $\mathbb{F}$ are more dense near x_{min}, and less dense while approaching x_{max}.

EXAMPLE:



TRUNCATION ERROR

Example;
Consider the series for $sinx$

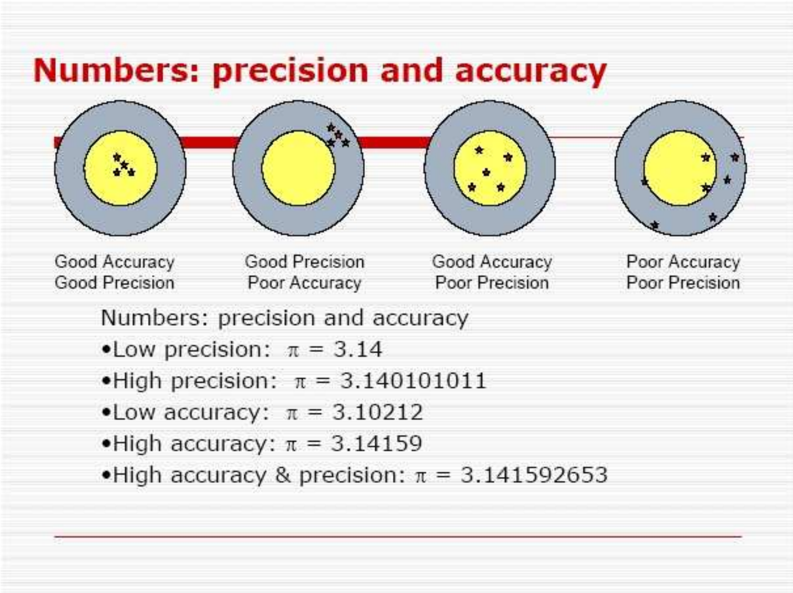
$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$

For small x, only a few terms are needed to get an accurate aproximation to $sinx$. The higher order terms are truncated.

$f_{true} = f_{sum} + truncation\ error$

The size of truncation error depends on x and the number of terms included in f_{sum}

PRECISION AND ACCURACY



Introduction to Scientific and Engineering Computing, BIL108E
Karaman

FLOW CONTROL

1. FOR statements
2. IF statements
3. SWITCH statements
4. WHILE statements

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

LOOPS

REPETITIVE TASKS

Loops are used for repetitive tasks.

Basic `for` Construct

The most common form of the loop is;

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

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

LOOPS

- $j : k$ is a vector with elements $j, j+1, j+2, ..., k$.
- $j : m : k$ is a vector with elements $j, j+m, j+2m, ...$ such that the last element can not exceed k.
- *index* must be a variable. Each time through the loop it will contain the next element of the vector $j : k$ or $j : m : k$.

LOOPS

index = *first* : *increment* : *last*

The number of times that the loop is executed is defined as iteration:

$iteration = \texttt{floor}(\frac{last-first}{increment}) + 1$ Here `floor(x)` is a function, that rounds x down toward $-\infty$

This value is called iteration or trip count

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

LOOPS

- 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 control passes to the statement following *end*.
- If the index does appear explicitly in *statements*, the `for` can often be vectorized. It runs faster.
- It is good programming style to indent (tabulate) the statements inside a `for` loop.

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

LOOPS

`for` in a single line

`for` $index = j : k$, *statements*, `end`

or

`for` $index = j : m : k$, *statements*, `end`

- Don't forget the commas.

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

LOOPS

More general form of the `for` is

`for` $index = v$

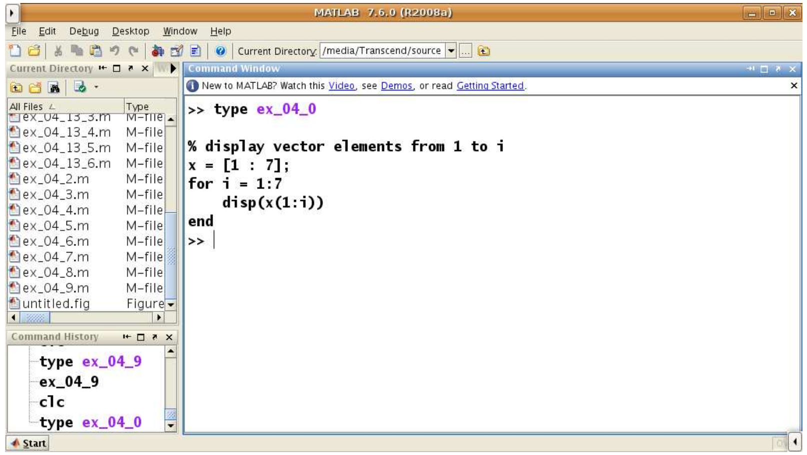
Here v is any vector.

The index moves through each element of the vector.

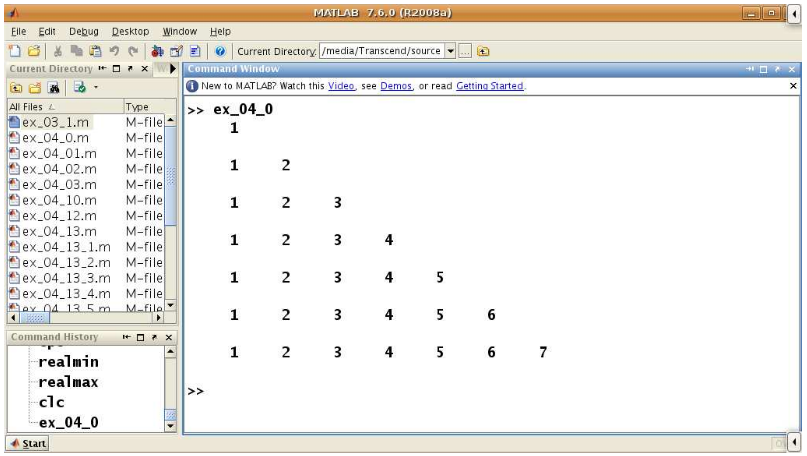
EXAMPLES:

```
% display vector elements from 1 to i
x = [1 : 7];
for i = 1:7
    disp(x(1:i))
end
```

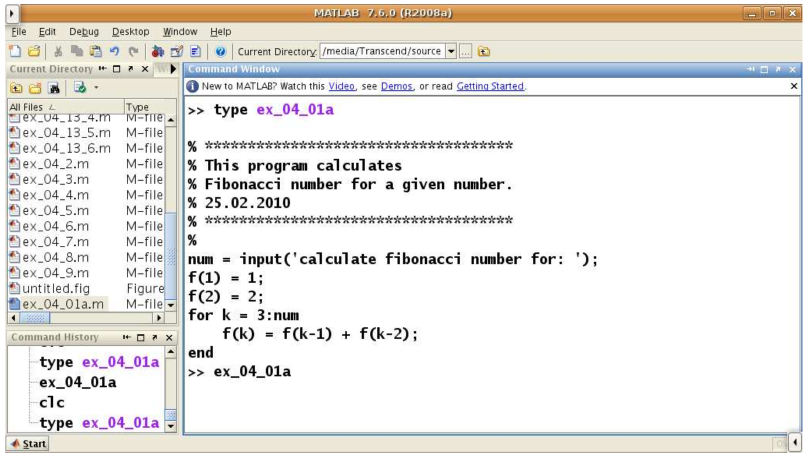

EXAMPLE:



EXAMPLE:

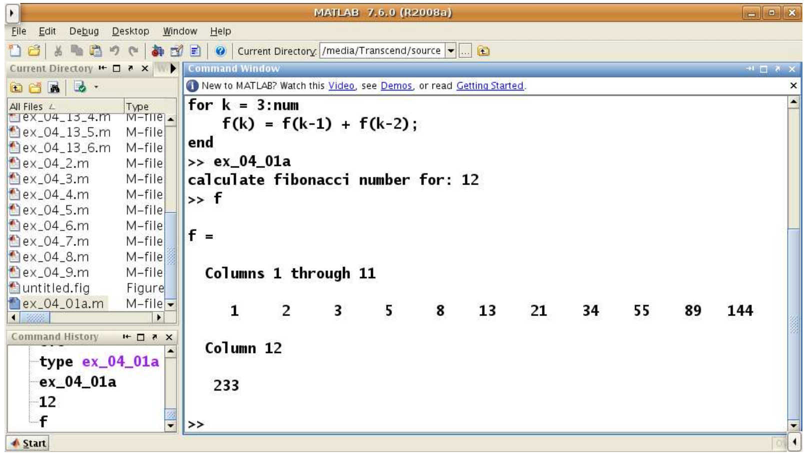


EXAMPLE:



LOOPS

EXAMPLE:



LOOPS

EXAMPLES:

Vectorize if possible

Evaluate the expression given below without the formula for the sum.

$$\sum_{n=1}^{100000} n$$

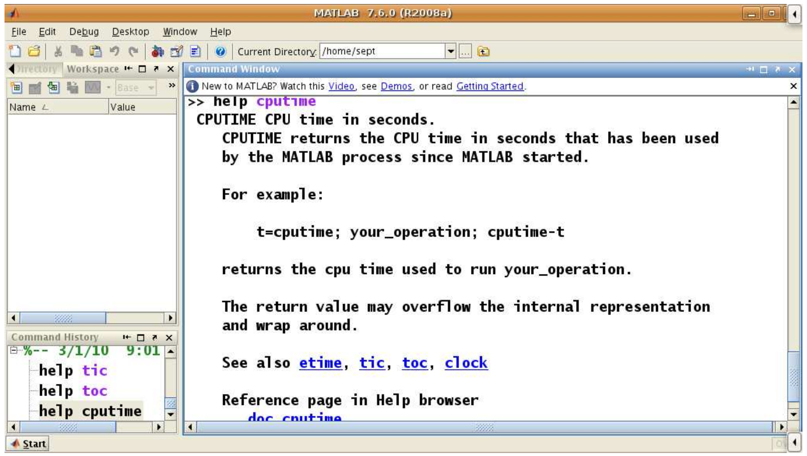
- clock function

  returns a six element vector.
- etime function

  returns the time in seconds between its two arguments.

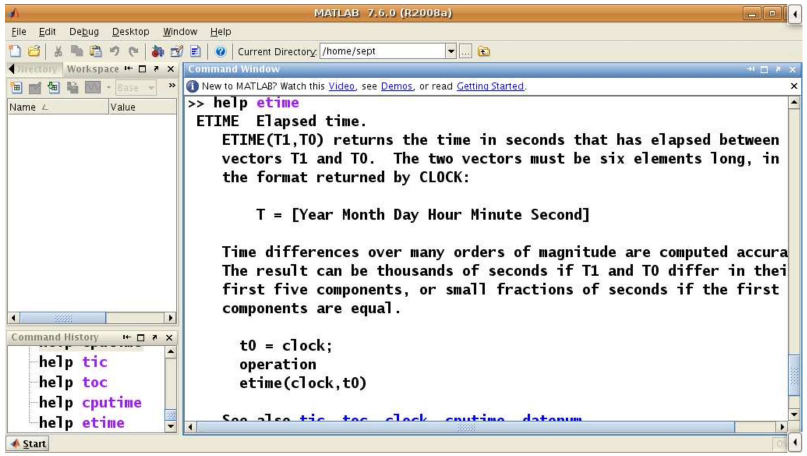
LOOPS

EXAMPLE:



LOOPS

EXAMPLE:

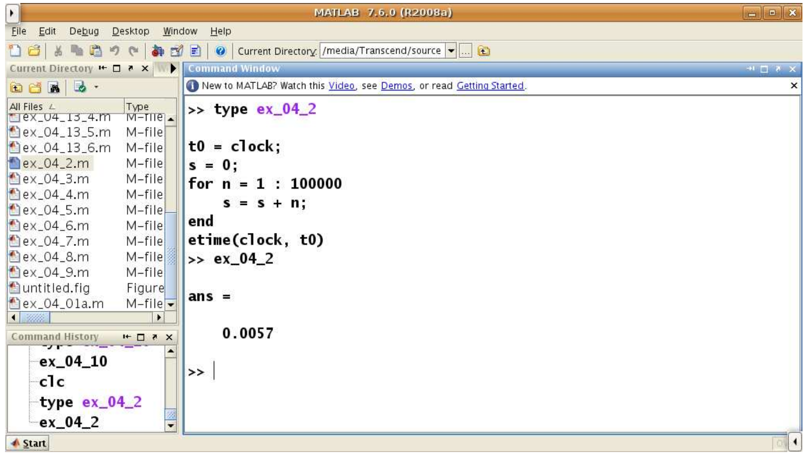


EXAMPLES

with for loop

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


EXAMPLE:

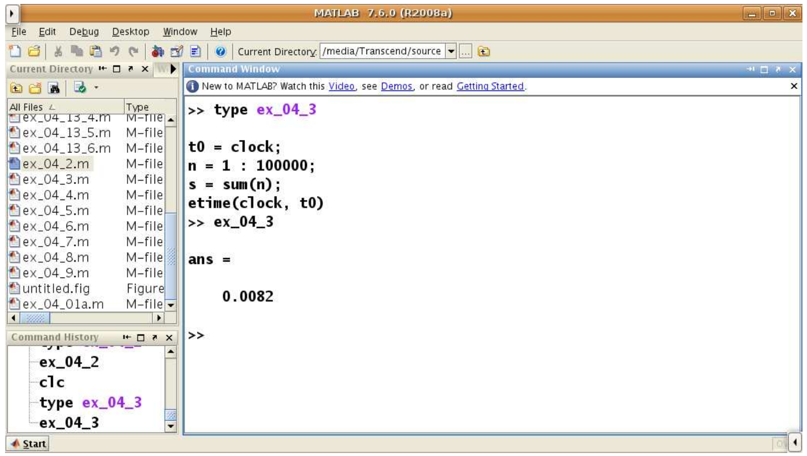


EXAMPLES

with vectorization

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


EXAMPLE:



EXAMPLES

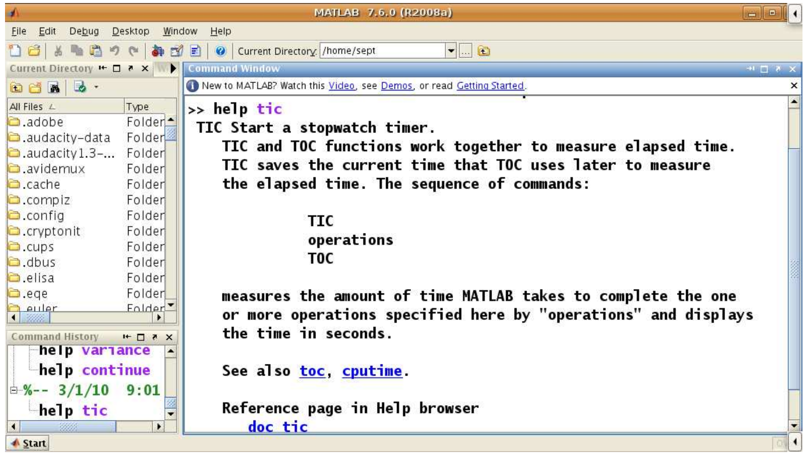
tic and toc functions

monitor the time to interpret MATLAB statements

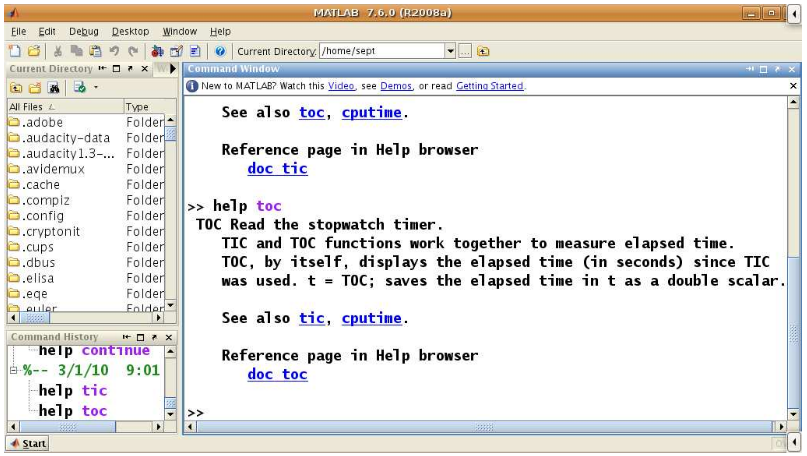
Evaluate;

$$\sum_{n=1}^{100000} \frac{1}{n^2}$$

EXAMPLE:



EXAMPLE:



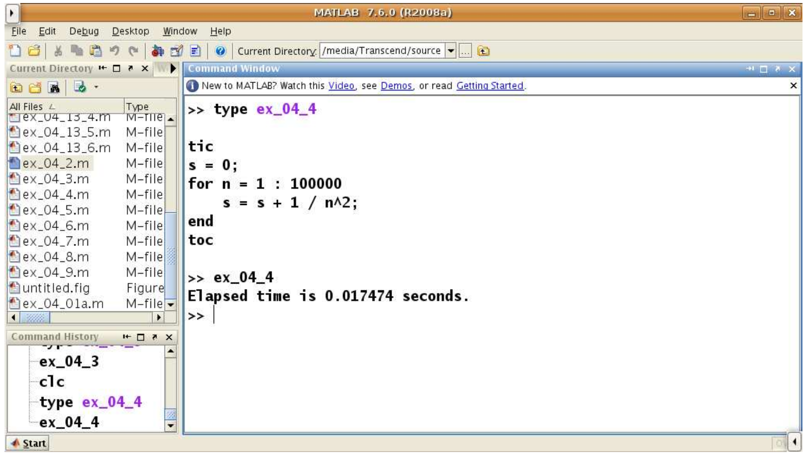
EXAMPLES

with for loop

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

LOOPS

EXAMPLE:



LOOPS

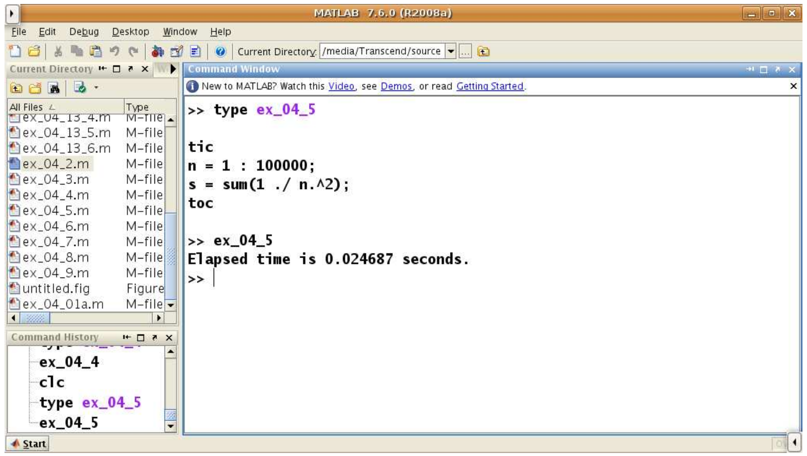
EXAMPLES

with vectorization

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

LOOPS

EXAMPLE:



CONDITIONAL STATEMENTS

`if` statements

- Relational Operators

Usage;

`if` *condition statement*, `end`

- condition is usually a logical expression
- 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.

CONDITIONAL STATEMENTS

LOGICAL EXPRESSIONS

Logical operators are used to combine logical expressions (with "and" or "or"), or to change a logical value with "not"

Operators:
& **AND**, | **OR**, ˜ **NOT**.

INPUT		OUTPUT			
A	B	A&B	A|B	˜A	˜B
false	false	false	false	true	true
false	true	false	true	true	false
true	false	false	true	false	true
true	true	true	true	false	false

CONDITIONAL STATEMENTS

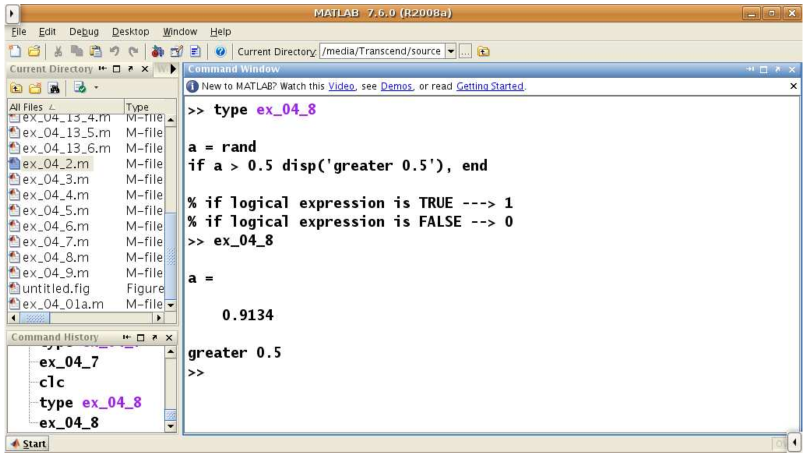
EXAMPLE

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

% if logical expression is TRUE ---> 1
% if logical expression is FALSE --> 0
```


CONDITIONAL STATEMENTS

EXAMPLE:



CONDITIONAL STATEMENTS

IF-ELSE

if *condition*
    blockofstatementsA
else
    blockofstatementsB
end

- *blockofstatementsA* or *blockofstatementsB* represents one or more statements.
- If *condition* is true *blockofstatementsA* is executed and if false *blockofstatementsB* is executed.
- else is optional.

CONDITIONAL STATEMENTS

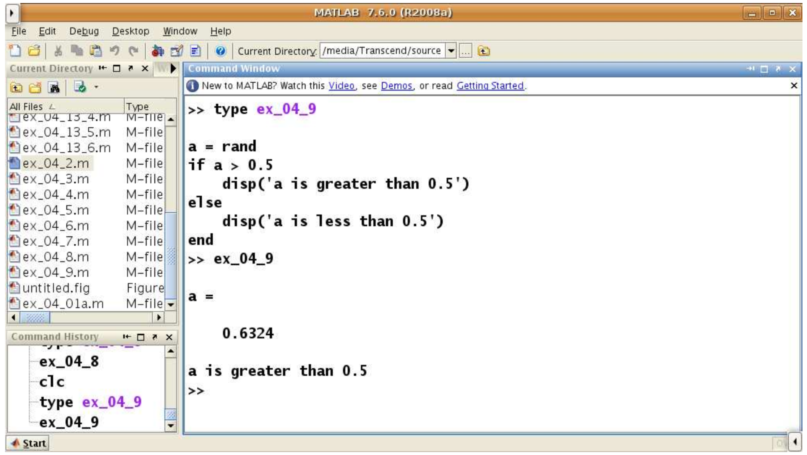
EXAMPLE

```
a = rand
if a > 0.5
    disp('a is greater than 0.5')
else
    disp('a is less than 0.5')
end
```

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

ERRORS

EXAMPLE:



Introduction to Scientific and Engineering Computing, BIL108E
Karaman

CONDITIONAL STATEMENTS

`elseif`

`if` *condition*1
    statementA
`elseif` *condition*2
    statementB
`elseif` *condition*3
    statementC
...
`else`
    statementE
`end`

This is called `elseif` ladder.

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

CONDITIONAL STATEMENTS

`elseif`

1. *condition*1 is tested. If it is true, *statementA* are executed; MATLAB then moves to the next statement after end.
2. If *condition*1 is false, MATLAB checks *condition*2. If it is true, *statementB* 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 andMATLAB jumps off the ladder.
4. If none of the conditions is true, *statementE* after else are executed.

`elseif` cont'd.

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.

CONDITIONAL STATEMENTS

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

NESTED `ifs`

An `if` construct can contain further `ifs`.

This is called NESTING.

`else` belongs to the most recent `ifs`.

CONDITIONAL STATEMENTS

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

`switch` STATEMENT

`switch` *value*

    case *val*1

        *statement*1

    case *val*2

        *statement*2

    case [*val*3 *val*4 *val*5]

        *statement*3

...

    otherweise

        statementN

`end`

CONDITIONAL STATEMENTS

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

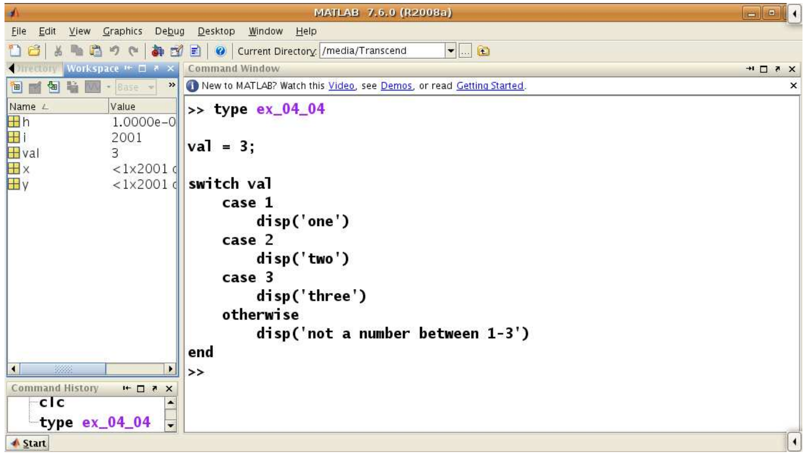
EXAMPLE:

```
val = 3;

switch val
    case 1
        disp('one')
    case 2
        disp('two')
    case 3
        disp('three')
    otherwise
        disp('not a number between 1-3')
end
```

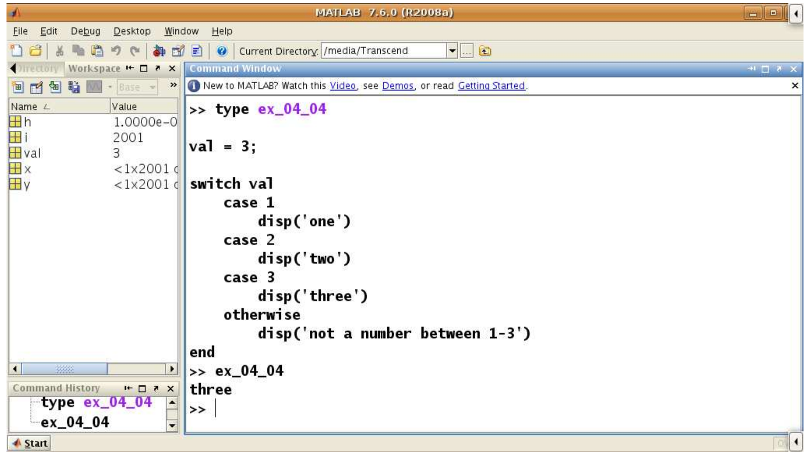
CONDITIONAL STATEMENTS

EXAMPLE:



CONDITIONAL STATEMENTS

EXAMPLE:



LOOPS

WHILE LOOP

While loops are most often used when an iteration is repeated until some termination criterion is met.

Usage;

`while` *expression*

   block of statements

`end`

The block of statements is executed as long as expression is true.

LOOPS

WHILE LOOP

To execute a `while`–`end` loop properly;

- The conditional expression in the `while` command must include at least one variable;
- The variables in the conditional expression must have been assigned when MATLAB executes the while command for the first time;
- At least one of the variables in the conditional execution must be assigned a new value in the commands that are between the `while` and the `end`. Otherwise once the looping starts it will never stop since the conditional expression will remain true.

LOOPS

EXAMPLE:

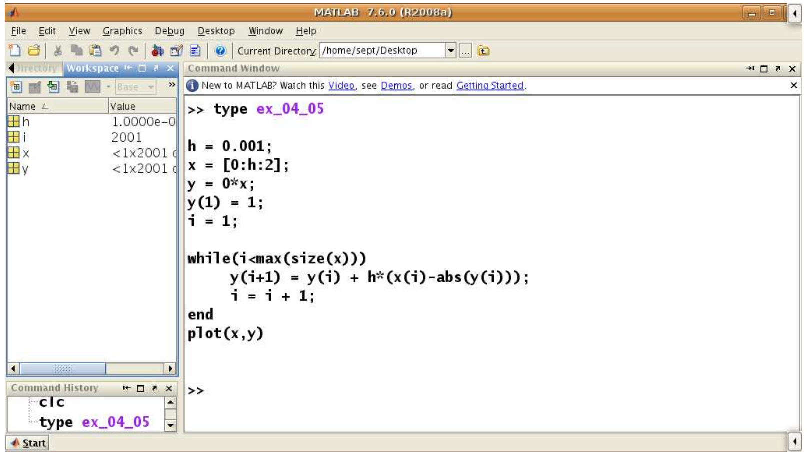
```
h = 0.001;
x = [0:h:2];
y = 0*x;
y(1) = 1;
i = 1;

while(i<max(size(x)))
     y(i+1) = y(i) + h*(x(i)-abs(y(i)));
     i = i + 1;
end
plot(x,y)
```

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

LOOPS

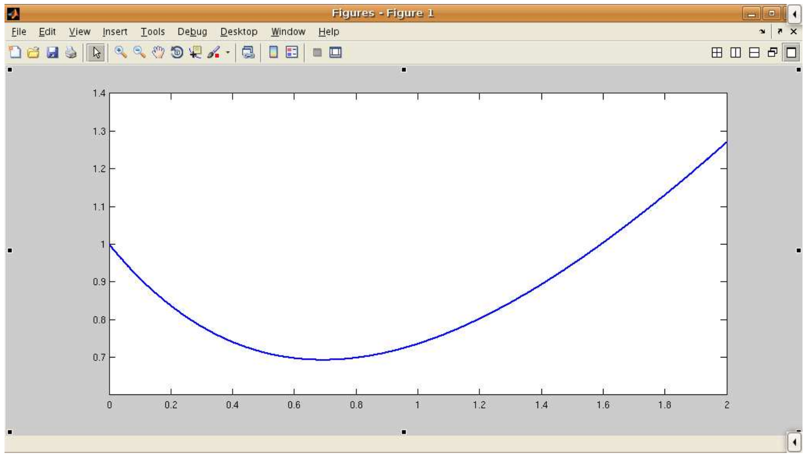
EXAMPLE:



Introduction to Scientific and Engineering Computing, BIL108E
Karaman

LOOPS

EXAMPLE:



Introduction to Scientific and Engineering Computing, BIL108E
Karaman

LOOPS

INTERRUPTING THE INFINITE LOOP

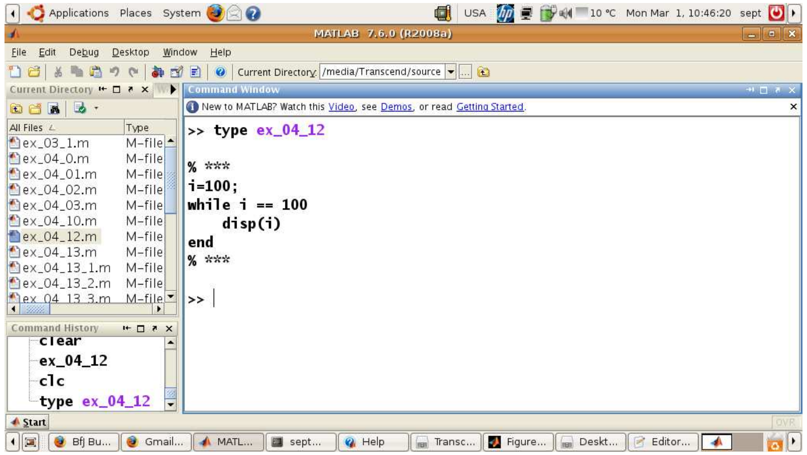
In case of an Infinite Loop

```
% ***
i=100;
while i == 100
    disp(i)
end
% ***
```

Use CTRL+C or CTRL+BREAK to stop the program.

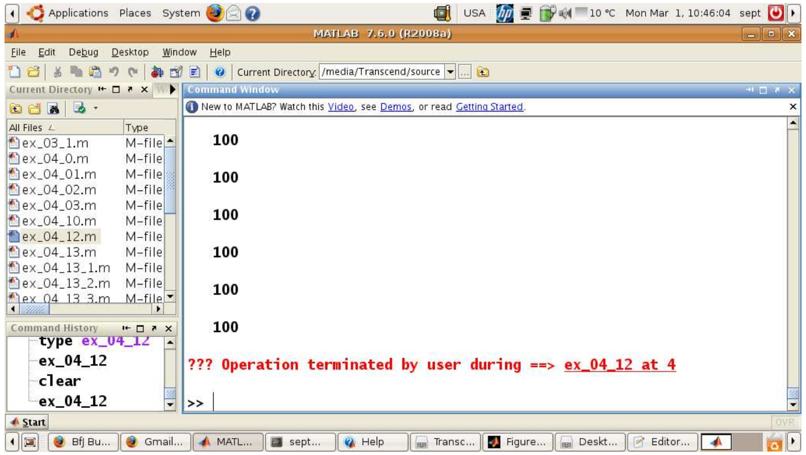
LOOPS

EXAMPLE:



LOOPS

EXAMPLE:



LOOPS

BREAK, CONTINUE AND RETURN

- The `break` and `return` statements provide an alternative way to exit from the flow of the program.
- `continue` passes control to the next iteration of `for` or `while` loop and skips any remaining statements in the body of the loop.
- `break` is used to escape from an enclosing while or for loop. Execution continues at the end of the enclosing loop construct.
- `return` is used to force an exit from a function. This can have the effect of escaping from a loop. Any statement following the loop that are in the function body are skipped (Next week "Functions").

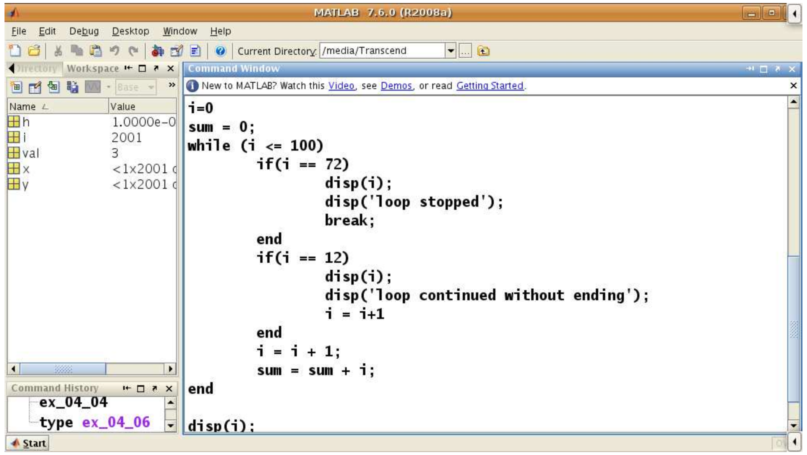
LOOPS

EXAMPLE:

```
i=0
sum = 0;
while (i <= 100)
if(i == 72)
disp(i);
disp('loop stopped');
break;
end
if(i == 12)
disp(i);
disp('loop continued without ending');
i = i+1
end
i = i + 1;
sum = sum + i;
end
```

LOOPS

EXAMPLE:

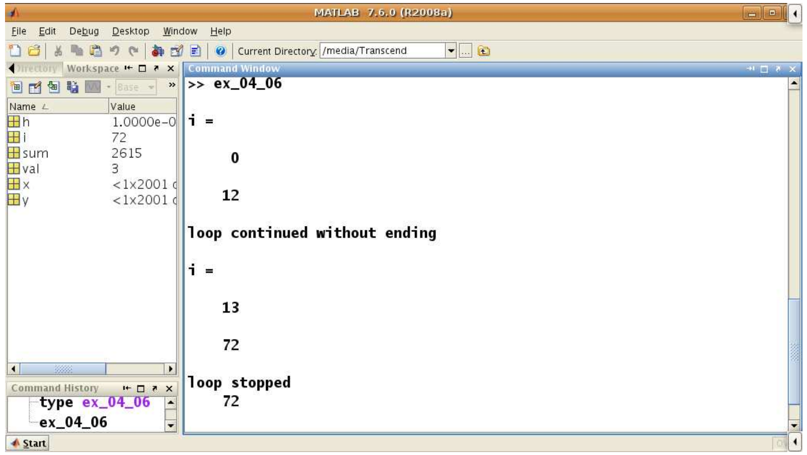


LOOPS

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

EXAMPLE:



PROGRAM DESIGN AND ALGORITHM

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

- Design Process
- Structure Plan

To design a successful program you need to understand a problem thoroughly and break it down into its most fundamental logical stages.

In other words, you have to develop a systematic procedure or an algorithm for solving it.

PROGRAM DESIGN

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

- The program must be readable and hence clearly understandable.
- It is useful to decompose the main program into subprograms that do specific parts of it.
- Add comments and references so that you know exactly what was done and for what purpose.

DESIGN PROCESS

1. Problem analysis.
2. Problem statement. Develop a detailed statement of the mathematical problem to be solved with a computer program.
3. Processing scheme. Define the inputs required and the outputs to be produced by the program.
4. Algorithm. Design the step-by-step procedure in a top-down process that decomposes the overall problem into subordinate problems.

DESIGN PROCESS cont'd.

5. Program algorithm. Translate or convert the algorithm into a computer language.
6. Evaluation. Test all of the options and conduct a validation study of the program. For example, compare results with other programs.
7. Application. Solve the problems, the program was designed to solve. If the program is well designed and useful, it can be saved in your working directory

DESIGN PROCESS

EXAMPLE

A function M-file is a script file designed to handle a particular task that may be activated (invoked) whenever needed.

DESIGN PROCESS

EXAMPLE

TRAJECTORY

A ball is thrown with an initial angle of θ and initial velocity of v_0.

Given

- velocity and theta angle

Find

- Projectile Flight Path
- Projectile speed vs. angle

EXAMPLE

1. $v_{0,x} = v_0 \times cos(\theta_0)$
2. $v_{0,y} = v_0 \times sin(\theta_0)$
3. $x(t) = v_{0,x} \times t$ (horizontal distance from origin as a function of t)
4. $y(t) = v_{0,y} \times t - 0.5 \times gt^2$ (vertical distance from origin as a function of t)
5. $y(x) = \frac{v_{0,y}}{v_{0,x}} \times x - 0.5 \times g\frac{x^2}{v_{0,x}^2}$ (with using # 3 and # 4 vertical location of the point is a function of x horizontal distance)

EXAMPLE

```
%
% The projectile problem with zero air resistance
% in a gravitational field with constant g.
%
% Written by ####### 01.03.2010

% Written by D. T. Valentine ........ September 2006
% Revised by D. T. Valentine ........ November 2008
% An eight-step structure plan applied in MATLAB:
%
```


EXAMPLE cont'd.

```
% 1. Definition of the input variables.
%
% Gravity in m/s**2
g = 9.81;
disp('*** INPUT DATA FOR PROJECTILE PROBLEM ***');
vo = input('Launch speed in m/s: ');
theta = input('Launch angle in degrees: ');
% Convert degrees to radians
theta = pi*theta/180;
```


EXAMPLE cont'd.

```
% 2. Calculate the range and duration of the flight.
%
txmax = (2*v0/g) * sin(theta);
xmax = txmax * v0 * cos(theta);
```

EXAMPLE cont'd.

```
% 3. Calculate the sequence of time
%    steps to compute trajectory.
%
dt = txmax/100;
t = 0:dt:txmax;
%
% 4. Compute the trajectory.
%
x = (v0 * cos(theta)) .* t;
y = (v0 * sin(theta)) .* t -(g/2) .* t.^2;
```

DESIGN PROCESS

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

EXAMPLE cont'd.

```
% 5. Compute the speed and angular
%    direction of the projectile.
% Note that vx = dx/dt, vy = dy/dt.
%
vx = v0 * cos(theta);
vy = v0 * sin(theta) - g .* t;
v = sqrt(vx.*vx + vy.*vy);
th = (180/pi) .* atan2(vy,vx);
%
% 6. Compute the time, horizontal
%    distance at maximum altitude.
%
tymax = (v0/g) * sin(theta);
xymax = xmax/2;
ymax = (v0/2) * tymax * sin(theta);
```

DESIGN PROCESS

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

EXAMPLE cont'd.

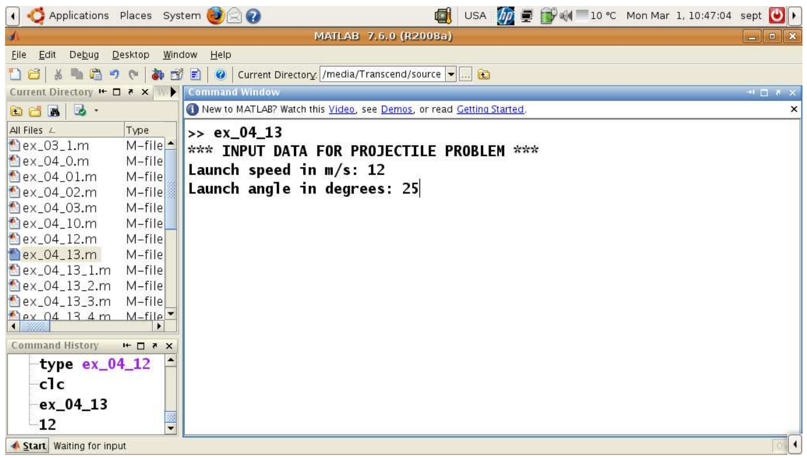
```
% 7. Display ouput.
%
disp([' Range in m = ',num2str(xmax), ...
' Duration in s = ', num2str(txmax)])
disp(' ')
disp([' Maximum altitude in m = ',num2str(ymax), ...
' Arrival in s = ', num2str(tymax)])
plot(x,y,'k',xmax,y(size(t)),'o',xmax/2,ymax,'o')
title([' Projectile flight path, vo =',num2str(vo), ...
' th =', num2str(180*th/pi)])
xlabel(' x '), ylabel(' y ') % Plot of Figure 1.
figure % Creates a new figure.
plot(v,th,'r')
title(' Projectile speed vs. angle ')
xlabel(' V '), ylabel(' \theta ') % Plot of Figure 2.
%
% 8. Stop.
%
```

DESIGN PROCESS

Introduction to Scientific and Engineering Computing, BIL108E

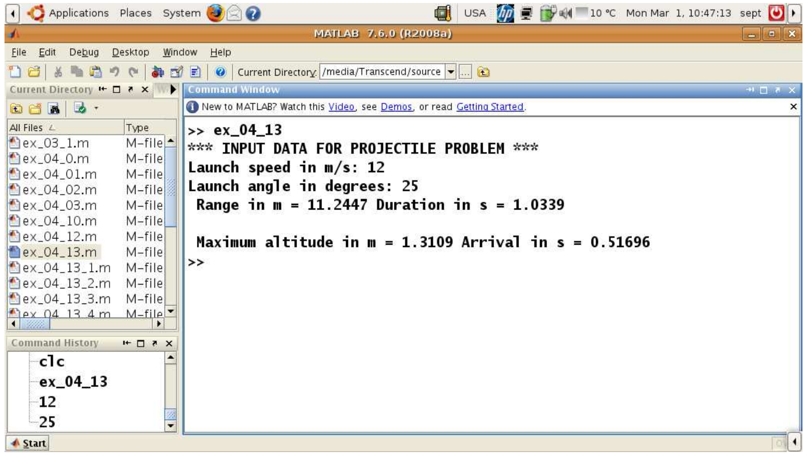
Karaman

EXAMPLE:



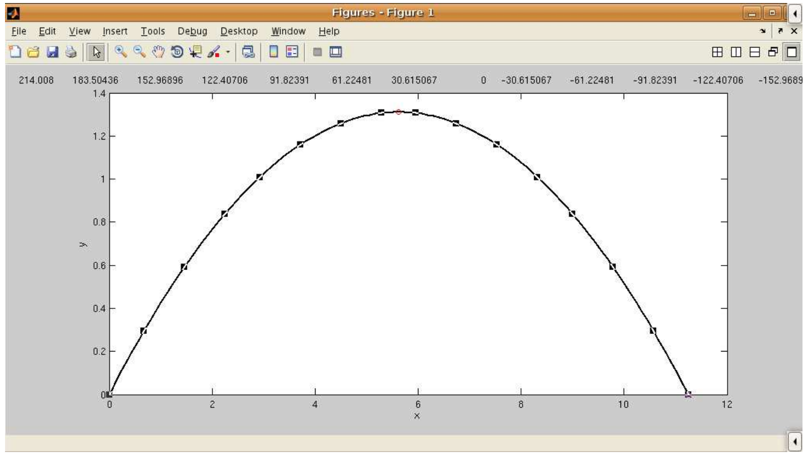
DESIGN PROCESS

EXAMPLE:



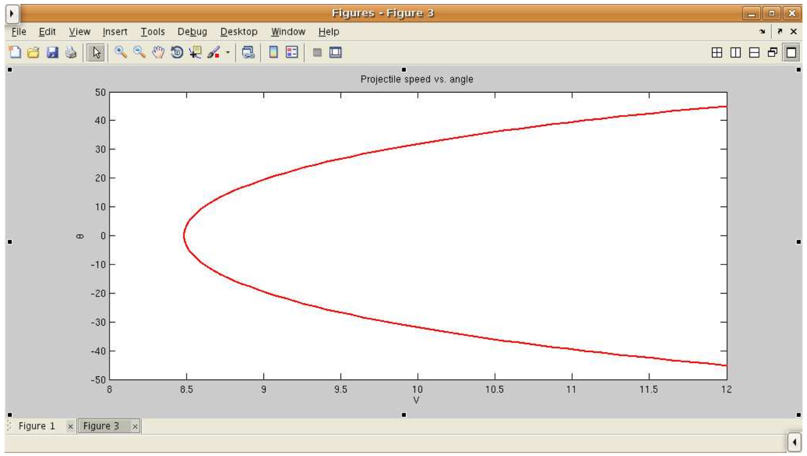
DESIGN PROCESS

EXAMPLE:



DESIGN PROCESS

EXAMPLE:



DATA ANALYSIS FUNCTIONS

DATA ANALYSIS FUNCTIONS

- `max(x)` Determines the largest value in x.
- `min(x)` Determines the smallest value in x.
- `sum(x)` Determines the sum of the elements in x.
- `prod(x)` Determines the product of the elements in x.
- `sort(x)` Returns a vector with the values of x in ascending order.

MEAN AND MEDIAN

- `mean(x)` Computes the mean(average value) of the elements of the vector x.

$$\bar{x} = \frac{\sum_{k=1}^{N} x_k}{N}$$

where $\sum_{k=1}^{N} x_k = x_1 + x_2 + \ldots + x_N$

- `median(x)` Determines the median value of the elements in the vector x.

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

VARIANCE AND STANDARD DEVIATION

- `var(x)` Computes the variation of the values in x.
- `std(x)` Computes the standard deviation of the values in x.
- The standard deviation is defined as the square root of the variance.

$$\sigma^2 = \frac{\sum_{k=1}^{N} (x_k - \bar{x})^2}{(N-1)}$$

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

References for Week 4

1. Alfio Quarteroni, Fausto Saleri, Scientific Computing with Matlab and Octave, Wissenschaftliches Rechnen mit Matlab, , Springer Verlag, 2006.
2. Cleve Moler, Numerical Computing with Matlab, Mathworks, 2008.
3. Brian Hahn, Daniel T.Valentine, Essential Matlab for Engineers and Scientists, Elsevier, 2010.