MA2252 Introduction to Computing

Lecture 1: Introduction and MATLAB Basics

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

- Understanding what computing means
- Introduction to MATLAB as a programming language
- Introduction to the course on Blackboard
- Overview of MATLAB environment

What is computing?

Input
x=1,
y=2

statements

It has all the commands

Output/
result

Using computers to solve a problem.

Some programming languages:

plot
x=5

- C , C++ , Java
- Python
- MATLAB ✓

Why do programming?

- To do tedious calculation tasks
- To build your own calculator!
- To find numerical solutions

long calculations by hand

multiplying two big 100x100 matrices

what if you don't have exact solutions to equations

calculate an integral

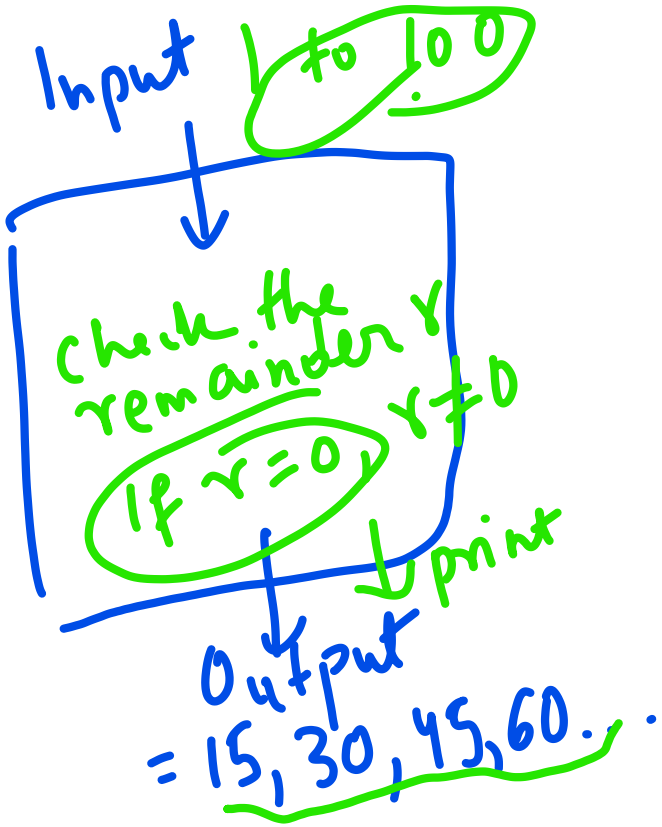
calendar

draw art/plot

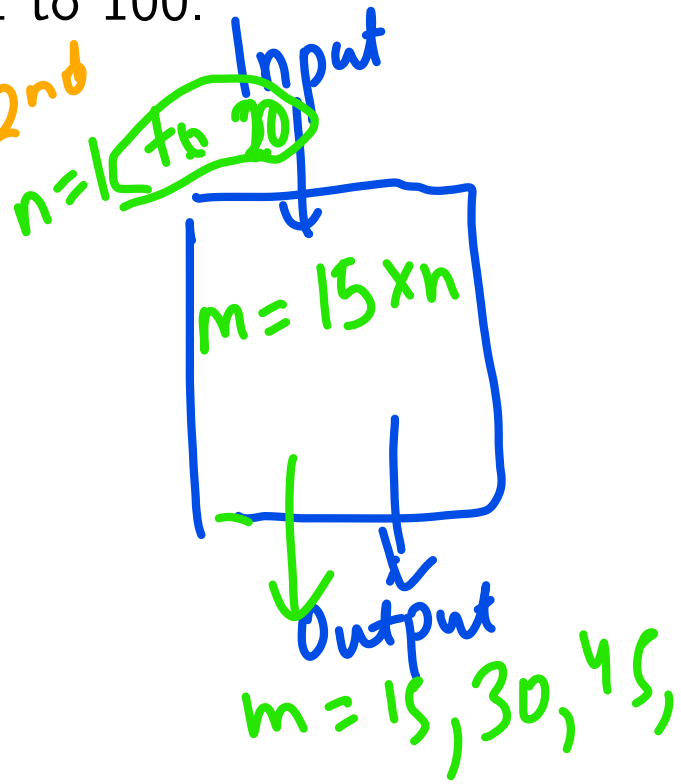
Example of a program

Find all numbers divisible by 15 from 1 to 100.

1st



2nd



Polling

I want to know if you have past programming experience.

Please go to the link https://www.menti.com/al76t3kh6sg7 provided in chat

or

visit https://www.menti.com and enter the code **84161441**

MATLAB platform

MATLAB is a popular programming language used by mathematicians, engineers and scientists to solve problems.

Features:

- User-friendly interface
- Provides tools for visualising data
- Includes large collection of built-in functions
- Easy matrix manipulations

MATLAB installation

Install MATLAB here

University of Leicester provides free MATLAB license for students.

Course introduction on Blackboard

Let's go to Blackboard!

MATLAB environment

The MATLAB environment contains four windows:

- Current Folder → current directory
- Command window → enter the commands
- Command history window → history of commands
- Workspace window → where all variables → matrix, x, y, z, shared, will be stored

MATLAB environment (contd.)

Demo

End of Lecture 1

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 3: Variables and Arrays

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

- Creating variables and assigning values
- Basic understanding of arrays and their types
- Creating some basic arrays

Variables

- Variables in MATLAB are used to store data e.g. a number or a word.
- Variable names examples: x, y, z, t_1, t_2, omega, etc.
- Variable names can contain letters, numbers or underscore.

Note: Variable name must start with a letter.

It's a good idea to name variables as they appear in your problems

Variable assignment

- The assignment operator '=' is used to store a value to a variable.
- **Examples:**

  x=5

  omega=1.2

  name='Sharad'

- The assigned variables are stored in workspace.

Variable assignment (contd.)

Demo

Variable assignment (contd.)

Useful tips:

- semicolon suppresses the variable assignment output in command window.
- 'clear all' deletes all variables in workspace.
- 'clc' clears text in command window.

Variable assignment (contd.)

Demo

Variable assignment (contd.)

The assignment operator '=' in MATLAB is different from equality sign '=' in Mathematics.

Try this!

```
x=2;

y=5;

y=x+1;
```

$x=2$
$y=5$

$y \neq x+1$
$5 \quad 3$

Arrays

$B = [1\ 0\ 2\ 3\ 5]$ → row vector

$A = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}_{2\times 2}$ → 2x2 array

- 'Matrices' in Mathematics are 'Arrays' in MATLAB.

$A(1,1) \leftarrow A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ → $A(2,1)$

$C = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix}$ → column vector

- Arrays are used to store and organise data.
- Arrays in MATLAB can be multi-dimensional.

↓ out of scope for this lecture

Some array types

- **Double array:** stores numbers
- **Char array:** stores alphanumeric characters

letters, & numbers

Double array

Double array is a very useful array for mathematicians and engineers.

Examples:

- x=[2 4 5]
- y= $\begin{bmatrix} 3 \\ 5 \\ 12 \end{bmatrix}$
- z= $\begin{bmatrix} 1 & 2 & 8 \\ 5 & 10 & 11 \\ 3 & 4 & 9 \end{bmatrix}_{3\times3}$

Array creation

Basic method to create an array

Put elements of array inside square brackets with a comma or space between elements. Separate rows by semicolons.

x=[1,2,3; 4,5,6; 7,8,9]

Example:

x=[1 2 3;4 5 6;7 8 9]

creates the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$.

Array creation (contd.)

Demo

Array creation (contd.)

Linearly-spaced 1D array (row vector) can be created by using

- colon operator (:)

$B = [3 \quad 9 \quad 15 \quad 21 \quad 27]$

- linspace function

$A = [1 \quad 2 \quad 3 \quad 4 \quad 5]$

1 1 1 5

Arithmetic progression (A.P.)

Array creation (contd.)

Using colon operator:

1 : 1 : 6

[1 2 3 4 5 6]

The command initial value : increment : final value

1 common difference / spacing 6

1

- creates an array with elements starting from initial value with specified increment until or equal to final value.

very useful when you know the spacing or increment

- **Example:**

  3:2:14 creates the array

  [3 5 7 9 11 13]

  2 2 2 2 2 is

Array creation (contd.)

Demo

Array creation (contd.)

Using linspace function:

$a = 1;\quad b = 10;\quad n = 5$

$\frac{b-a}{n-1} = \frac{10-1}{5-1} = \frac{9}{4}$

$a=1 \quad 1+d \quad 1+2d \quad 1+3d \quad b=10$

$1+4d$

$[1 \quad 3.25 \quad 5.5 \quad \ldots \quad 10]$

linspace(a,b,n)

- creates an array with equally-spaced specified number of elements n starting at initial value a and ending at final value b.

$1+4d = 10$

$4d = 9$

$d = 9/4 = 2.25$

- **Example:**

  linspace(1,2,11) creates the array

  [1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2] containing 11 elements.

  0.1 0.1

$\frac{2-1}{(11-1)} = \frac{1}{10} = 0.1$

Array creation (contd.)

Demo

Array creation (contd.)

To check your understanding of colon operator and linspace, let's do a mentimeter poll.

Please go to the link https://www.menti.com/alnd6z1jfk4g provided in chat

or

0 : 2 : 11

[0 2 4 6 8 10]

+2 +2 +2 +2 +2

increment

visit https://www.menti.com and enter the code **67866952**

End of Lecture 3

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 4: Variables and Arrays (contd.)

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

- Create some special matrices
- Learn array indexing and arithmetic operations
- Understand Char Arrays
- Use load and save functions

Special matrices

ones(3) $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}_{3\times 3}$ $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}_{2\times 2}$ ones(2)

- Matrix of ones: use function 'ones'
- Null matrix: use function 'zeros' zeros(2) $= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}_{2\times 2}$
- Identity matrix: use function 'eye' $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ eye(3)
- Diagonal matrix: use function 'diag'

$v = [2\ 3\ 10]$;
diag(v) $= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 10 \end{bmatrix}$

Special matrices (contd.)

Demo

Special matrices (contd.)

$A = [\quad]_{4\times 8} = 32$ → to create bigger matrices from a smaller matrix

$16\times 2 = 32$

Using 'repmat' and 'reshape' functions

- **repmat** creates copies of a given matrix

$[1\ 2\ 3\ 4]$

- **reshape** transforms size of a given matrix

$\text{reshape}(A,1,4)$ $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{2\times 2}$

$[1\ 2\ 3\ 4]_{1\times 4}$

$\text{repmat}(A,2) = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{bmatrix}$

$\text{reshape}(B,9,1)$ $B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}_{3\times 3}$

$(B,9,1)$

Special matrices (contd.)

Demo

Array indexing

Array indexing means extracting elements of array.

B(:,2)

1 is the index of element 5 in array A

Examples:

- Let A=[5 3 1 0]. Then A(1)=5, A(2)=3 and so on.

A(3)=1
A(4)=0

B

- If Let B= $\begin{bmatrix} 2 & 6 & 5 \\ 7 & 1 & 3 \\ 5 & 2 & 8 \end{bmatrix}$

3rd row

1st row

1st column

B(3,[1,2,3])

Then B(1,1)=2, B(1,2)=6 and so on.

Array indexing (contd.)

- Use colon operator to extract a row or column.

  3rd row

  Example: B(3,:) = [5 2 8] (extracts the third row of B)

  all columns

- You can also extract arrays from arrays!

  Example: B([2,3],[2,3]) = extracts the array $\begin{bmatrix} 1 & 3 \\ 2 & 8 \end{bmatrix}$.

Array indexing (contd.)

Demo

Char Arrays

- Char Arrays store alphanumeric characters such as numbers and letters.

combination of letters & numbers ' ' '

Char Arrays (contd.)

Demo

Char Arrays (contd.)

To modify a pre-defined string

Char arrays can also be made using **sprintf** function.

Examples:

plot of sinx vs x

```
title1=sprintf('plot of %s vs x','sinx')
```

plot of cosx vs x

```
title2=sprintf('plot of %s vs x','cosx')
```

call ~~the~~ it from history command

Char Arrays (contd.)

Demo

Arithmetic operations with arrays

$A^3 = \begin{bmatrix} 0 & 1 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 5 & 6 \end{bmatrix}$ → scalar

$B = \begin{bmatrix} 1 & 2 \\ 8 & 9 \end{bmatrix}$

- Operations between a constant (say c) and a matrix (say A)

  Examples: A+c, A-c, A*c, A/c, A^c

  $A + B = \begin{bmatrix} 1 & 3 \\ 13 & 15 \end{bmatrix}$

- Operations between two matrices (say A and B)

  Examples: A+B, A-B, A*B, A/B, A.*B, A./B, A.^B

  . means element by element

$A.*B = \begin{bmatrix} 0 & 2 \\ 40 & 54 \end{bmatrix}$

$A = \begin{bmatrix} 0 & 1 \\ 5 & 6 \end{bmatrix}$

$c = 3$

$A + 3 = \begin{bmatrix} 3 & 4 \\ 8 & 9 \end{bmatrix}$

$A - 3 = \begin{bmatrix} -3 & -2 \\ 2 & 3 \end{bmatrix}$

Arithmetic operations with arrays (contd.)

eg $Y = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \end{bmatrix}$

- Transpose of a matrix

Example:

1st row

1st column

2nd column

If $A = \begin{bmatrix} 2 & 3 \\ 7 & 0 \end{bmatrix}$ then $A' = \begin{bmatrix} 2 & 7 \\ 3 & 0 \end{bmatrix}$

2nd row

Arithmetic operations with arrays (contd.)

Demo

Arithmetic operations with arrays (contd.)

factorial(4) = 4x3x2x1 = 24

Arithmetic functions can also take arrays as input.

Example:

factorial([1 2 3 4 5])

x=[1 2 3 4 5] ;

factorial(x)=[1 2 6 24 120]

output will be an array

sin(x)
cos(x)
sqrt(x)
log(x)
x as array

Arithmetic operations with arrays (contd.)

Demo

Saving and loading variable data

- **save function:** stores variable data in a .mat file

  Command: save 'filename' 'variables to store'

  .mat

  x=1, y=2

  title='plot sinx'

  myfile x, y, title

- **load function:** recalls the variable data from a .mat file

  Command: load 'filename'

  load myfile.mat

Saving and loading variable data (contd.)

Demo

End of Lecture 4

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 5: Functions

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

- Basic understanding of functions
- Create your own functions
- Using Comments in programming
- Understand Function Workspace

Function definition

A function in MATLAB is a sequence of instructions that takes some input and gives the desired output.

lines of code in a sequence

Some familiar examples:

- sin(x)
- sqrt(x)
- linspace(a,b,n)

x is input variable

sqrt(4) = 2

x=4

y = sqrt(x)

y=2

input: you give values to input variables

Note: Unlike in Mathematics, a function in MATLAB can have multiple outputs!

output: you get some values of output variables

Building a function

A function's code is written in the EDITOR window.

The code is built in three parts:

- Header
- Body
- End statement

Header

Construction:

function [output variables] = function_name(input variables)

function [x] = mysum(a, b)

Header includes:

- name of function
- lists of input variables
- lists of output variables

Building a function (contd.)

$1, 2, 3, \ldots n$ $\quad$ $\text{Sum} = \frac{n(n+1)}{2}$

Sum

$\text{Sum} = 1 + 2 + \ldots n$

$\text{Sum} = n + (n-1) + \cdots 2 + 1$

$2 \times \text{Sum} = (n+1) + (n+1) + \ldots (n+1)$ $\times n$ times

$\text{Sum} = \frac{n(n+1)}{2}$

Header examples:

- function [sum] = myfun1(n)
- function [sum]= myfun2(a,d,n)
- function [sum,a_n] = myfun3(a,d,n)

$a + (n-1)d$

Note: Function name must start with a letter, contain alphanumeric characters or underscore.

$\text{Sum} = \frac{n}{2}\left[2a + (n-1)d\right]$ $\quad$ $\underbrace{a,\ a+d,\ a+2d,\ \ldots\ a+(n-1)d}_{n \text{ terms}}$

Body

Group of statements or lines of code written to perform the desired task.

Body examples: myfun1( )

- sum=n*(n+1)/2;
- sum=(n/2)*(2*a + (n-1)*d); → myfun2( )
- sum=(n/2)*(2*a + (n-1)*d); → myfun3( )
  a_n=a+(n-1)*d;

End statement

The end statement is written using end keyword which

- denotes end of function
- is placed after the body

Building a function (contd.)

A sample function:

```
function [sum,a_n] = myfun3(a,d,n)

sum=(n/2)*(2*a + (n-1)*d);
a_n=a+(n-1)*d;

end
```

Building a function (contd.)

Demo

Saving a function file

- Use ‘Ctrl+S’ or ‘Save’ button in EDITOR menu to save the function file.
- Always save the function file with the function name.
- If using MATLAB online, save your function in the current directory.
- Function files are saved as .m file files.

Calling a function

- To call/use the function, type

~~function~~ [output variables] = function_name(input variables' values)

in the command window and press 'Enter'.

- Example:

sum, a-n

2, 4, 6, 8, 10, 12, ... 20

~~57~~ SUM = ~~50~~ 110

myfun3

~~function~~ [SUM,A_N] = ~~function_name~~(2,2,10)

A_N = 20

Calling a function (contd.)

Demo

Comments

Comments are text in the .m file that MATLAB doesn't execute.

MATLAB ignores comments

Comments are used

- to help you and others understand your code.
- to add additional information e.g. author name, date, notes.

command + /

To comment: Use % symbol or press 'Ctrl + r'.
To uncomment: Use 'Ctrl + Shift + r'.

works in both (online & offline)

command + t

Ctrl + t → doesn't work in MATLAB online

Comments (contd.)

Demo

Function Workspace

A function workspace contains variables which were used to create that function.

Example: The function workspace of myfun3() contains the variables sum, a_n,a,d and n.

Note: A function workspace does not share variables with command window workspace and vice-versa.

This is because there is no overlap, confusion & also users have their freedom to name their variable

Activity 1

Define the function myfun3() as mentioned before. Type the following in MATLAB command window:

```
>> a=3;
>> d=4;
>> n=7;
>> [SUM,A_N]=myfun3(1,1,10);
```

function [sum, a_n] = myfun3(a,d,n)

sum = $\frac{n}{2}$[2*a + (n-1)*d];

a_n = a + (n-1)*d;

end

$\frac{n(n+1)}{2} = \frac{10 \times 11}{2} = 55$

What should be the final values of a,d and n?

Student 1 a=1, d=1, n=10, sum=55

Student 2 a=3, d=4, n=7

Activity 2

using clear all in command window

Clear all the stored variables. Now, type the following in MATLAB command window:

```
>> sum=1;
>> a_n=1;
>> [SUM,A_N]=myfun3(1,1,10);
```

in command window workspace

What should be the final values of sum and a_n?

In variable names, lower & upper case matters!

End of Lecture 5

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 6: Functions (contd.)

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to understand and create

- Subfunctions
- Function handles
- Script files

Subfunctions

A subfunction is a function defined under a parent/main function in the same .m file.

main function ← function [ ] = ___( )

call this subfunction here: outputvariables = subfunction name(1,2)

end

subfunction ← function [ ] = name( )

end

Subfunctions (contd.)

Example:

```
function [x1,x2] = myrootsfun(a,b,c)

%This function calculates roots of quadratic equation with
%coefficients a, b and c.
%Sharad
%28-09-2022

D = mydiscriminant(a,b,c);    %call the mydiscriminant subfunction
x1=(-b+sqrt(D))/(2*a);
x2=(-b-sqrt(D))/(2*a);

end
```

$ax^2+bx+c=0$

$x1, x2$

$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$

$D = b^2-4ac$

subfunction

$x = \frac{-b \pm \sqrt{D}}{2a}$

$D \rightarrow$ discriminant

main function variables $\rightarrow$ (D, a, b, c, x1, x2)

Subfunctions (contd.)

```
function [disc] = mydiscriminant(A,B,C)

%This subfunction calculates the discriminant D.

disc=B^2-4*A*C;

end
```

(disc, A, B, C)

Subfunctions (contd.)

Note:

only parent function is allowed to call its subfunction

- Only parent function can call its subfunction.
- Subfunction retains a separate workspace from its parent function.

Subfunctions (contd.)

Demo

Activity

Consider again the subfunction mydiscriminant(A,B,C). What happens when you type

DISC=mydiscriminant(1,-7,10) in command window and hit 'Enter' ?

$DISC = (-7)^2 - 4 \times 1 \times 10$

$= 49 - 40 = 9$

Let's do a mentimeter poll!

Activity (contd.)

Please go to the link https://www.menti.com/al9p1z6skq4f provided in chat

or

visit https://www.menti.com and enter the code **39701413**

Function handles

x, y, z

A function handle is a variable which stores some function.

Construction:

1. Using built-in functions: function_handle=@function_name

   assignment operator

   function's name you want to store in your function handle

   name of your function handle

   Example: F=@sin

   Here, variable F is function handle which stores sine function.

   function handle

Function handles (contd.)

user-defined function

definiti

2. Using anonymous function:

function_handle=@(input variables) function_definition

variables which your function contains

Examples:

- reciprocal=@(x) 1/x

  reciprocal = $\frac{1}{x}$

  reciprocal(10) = $\frac{1}{10}$ = 0.1

- myimplicit=@(x,y) x+y+x*y

  myplicit = x+y+xy

- combinations=@(n,r) factorial(n)/(factorial(r)*factorial(n-r))

no. of ways to choose r objects from n objects

$^nC_r = \frac{n!}{r!(n-r)!}$

myimplicit(1,2)
= 1+2+2
= 5

combinations(5,2)
= 5!/2!3! = $\frac{5\times\cancel{4}^{2}\times\cancel{3}}{2\cdot\cancel{3}}$ = 10

Function handles (contd.)

F = @sin
F(pi/2)

sin(x)
~~sin(x,y)~~

Demo

Function handles (contd.)

f = ~~combinatorics~~ combinations.

Using function handle to pass a function to other functions

Example: f=@sin; g=@cos; → sumfun is a function which calculates f(x)+g(x)

f,g,h

```
function [y] = sumfun(f,g,x)

%This function calculates the sum f(x)+g(x) for any two given
%functions f and g.

y=f(x)+g(x);

end
```

f = sinx
g = cosx

$x = \pi/2$

$\sin \pi/2 + \cos \pi/2$

$= 1 + 0$

$= 1$

function handles

f(x)*g(x) Y = sumfun(@sin, @cos, x)

fog(x) = f(g(x)) Y = sumfun(f, g, Pi/2)

Function handles (contd.)

Demo

sin, sin

F

Using Function Handles

Pros

1. To use function as a variable whenever needed.
2. No need to write a .m file to define your function.
3. To pass function as input to other functions.

~~f~~
header
body
end

Function handles (contd.)

[x1, x2] = my roots fun(a,b,c)

2 outputs

Cons

1. Only works with functions with one output.
2. Only useful when your function has a simple definition.

better make a .m file

Script files

Script file is a .m file where you write your code.

- To open a script file, click 'New Script' in HOME menu.
- To save, click 'Save' in EDITOR menu.
- To run the script file, click 'Run' in EDITOR menu.

Script files (contd.)

Example:

```
%This script file calculates the roots of a quadratic equation with
%coefficients a,b and c.
clc
clear all
a=1;
b=-7;
c=10;
D = b^2-4*a*c;     %calculate the discriminant D
x1=(-b+sqrt(D))/(2*a);     %calculate first root x_1
x2=(-b-sqrt(D))/(2*a);     %calculate second root x_2
```

→ good idea to start with clc & clear all

clc: clean the screen (command window)

clear all: to clean previously stored variables

a=1
b=-7
c=10
D=9
x1=5
x2=2

Script files (contd.)

Demo

Script files (contd.)

Script files vs Functions

- Script files share their workspace with command window workspace. Functions have their own workspace.

goal
which doesn't need changing

- Script files are used for specific task. Functions are useful when the same task has to be done for different inputs.

your inputs

with functions you can change inputs

End of Lecture 6

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 7: Branching Statements

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to understand and create

- If-Statements
- Switch statements

Branching Statements

Branching statements are used in programming to execute a section of code under specific conditions.

Branching statements also called conditional statements.

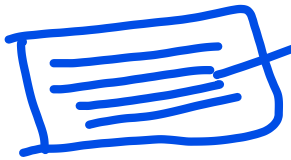
cls
clear all

Examples:

1. If-Statements
2. Switch Statements

code block 1 (x>0)

n is even



code block 2 (x<0)

n is odd

If-Statements

If statements can be constructed in different ways and they always end with end keyword.

1. Using if keyword

   Construction:

   if logical expression

   code block

   end

if $x > 0$

end

If-Statements (contd.)

$x = 50, y = 80 \rightarrow$ PASS

x & y are marks in two subjects

Example: total

```
function [t] = total_marks(x,y)
if x<40||y<40
   disp('FAIL')
end
t=x+y;
end
```

x<40 || y<40

or

displays FAIL in command window.

x = 20
y = 70
x+y = 90

If-Statements (contd.)

2. Using if and else keywords

Construction:

```
if logical expression
      code block 1
else
      code block 2
end
```

if this expression is true execute code block 1

otherwise execute code block 2

If-Statements (contd.)

Example:

```
function [t] = total_marks(x,y)
if x<40||y<40
   disp('FAIL')
else
   disp('PASS')
end
t=x+y;
end
```

$(x<40) \,||\, (y<40)$

to be on safe side

calculates total

order of precedence

$*$, $/$

$+$, $-$

$<$, $=$, $>=$, $<=$

$x||y<40$

$(x||y)<40$

If-Statements (contd.)

Demo

If-Statements (contd.)

3. Using if, elseif and else keywords

Use elseif when you have more than 1 (logical) condition to check

Construction:

```
if logical expression P
    code block 1
elseif logical expression Q
    code block 2
elseif logical expression R
    code block 3
else
    code block 4
end
```

if P is true

if P is false & Q is true

if both P & Q are false & R is true

when all of P, Q & R are false

If-Statements (contd.)

Example

```
function [t] = total_marks(x,y)
if x<0||y<0
   disp('marks cannot be negative')
elseif x>100||y>100
   disp('marks cannot exceed 100')
elseif x<40||y<40
   disp('FAIL')
else
   disp('PASS')
end
t=x+y;
end
```

P

Q

R

calculating the total

x = 200
y = 30
not legal marks

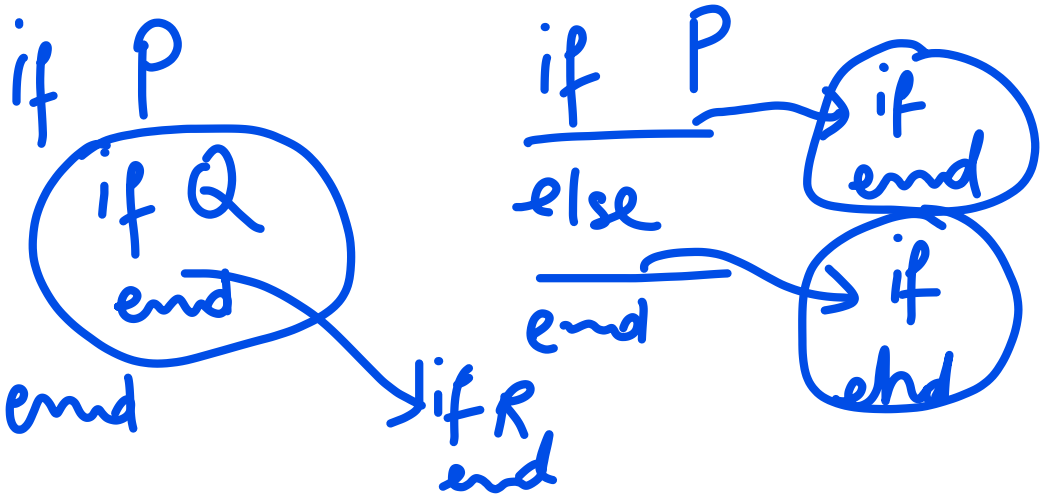
x = −5, y = 10
invalid marks

If-Statements (contd.)

Demo

Nested If-Statement

If-statement nested/contained within another if-statement.



If-Statements (contd.)

Example:

```
function [t] = total_marks(x,y)
if x<0||y<0
   disp('marks cannot be negative')
   t=sprintf('cannot calculate total marks');
elseif x>100||y>100
   disp('marks cannot exceed 100')
   t=sprintf('cannot calculate total marks');
```

If-Statements (contd.)

```
else
t=x+y;
   if x<40||y<40
      disp('FAIL')
   else
      disp('PASS')
   end
end
end
```

total is calculated in the else part

this is nested inside the parent if statement

If-Statements (contd.)

Demo

Activity

```
function [weight] = myweight(x)
if weight>70
   disp('Eat healthy')
elseif weight>80
   disp('Exercise more')
elseif weight>100
   disp('Call the doctor')
else
   disp('Relax!')
end
```

110 > 70 True

are not checked by MATLAB

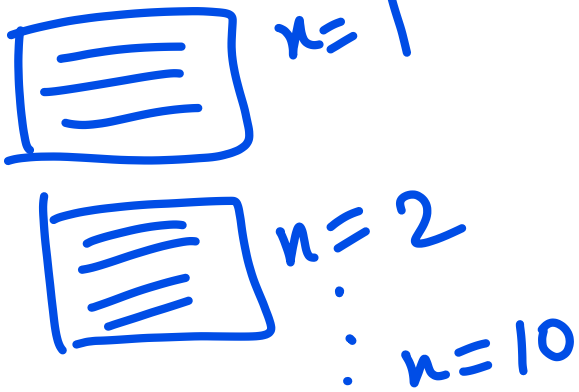
```
if weight>100
   call the doctor
else if weight>80
   weight>70
else if
```

My weight is 110 kg. What suggestion this code will give? To answer, please go to mentimeter link in the chat.

Switch Statements

learn from the books S. Attaway & Otto-Denier

Switch statements are used to check if an expression is equal to one of possible values. Each value is called a case.



if x==1 case 1

else if x==2 case 2

e

Switch Statements (contd.)

Construction:

variable, string

x

$x < 40 \cancel{\|} y < 40$

```
switch expression
   case value 1
      code block 1
   case value 2
      code block 2
   case value 3
      code block 3
   otherwise
      code block 4
end
```

Switch x

case 1 $\quad x == 1$

case 2

just like else in if

Switch Statements (contd.)

Example:

x is marks out of 10

```
function grade = mygrade_switch(x)
%this function calculates grade based on marks from 0 to 10 using switch
switch x
   case 10
      grade='A+';    %assign A+ if marks=10
   case 9
      grade='A';
   case{7,8}
      grade='B';
   otherwise
      grade='C'
end
end
```

7 or 8

6, 5, 4, 3, 2, 1, 0

use if else to state the validity of marks

Switch Statements (contd.)

Demo

If vs Switch

1. If-statements are useful when you deal with logical conditions in your code.

   use if

2. Switch becomes handy when you deal with one expression taking multiple possible values. → just to check many values of a variable, string etc.

3. Switch statements can also be written using if-statements but vice-versa is not true in general.

   if-else can be used at anytime

End of Lecture 7

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 8
Part 1: Iteration

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to understand and create

- For-Loop
- While-Loop

Iteration

running your code block to do a task again & again until a desired result is reached.

Iteration means to perform a task repeatedly.

Example:

To find the sum of first n natural numbers, addition of two numbers taken at a time should be done repeatedly.

$$1+2+3+4+\cdots n = \frac{n(n+1)}{2}$$

Iteration (repetitive work)

$3+3=6$

$6+4=10$

$10+5=15$

Iteration (contd.)

In MATLAB, iteration can be done via

1. For-loop
2. While-loop

For-Loop

Construction:

for looping variable = looping array
  code block
end

```
for x = linspace(2:2:10)
y=sinx ;
end
```

sin2
sin4
[2 4 6 8 10]
sin6
sin8
sin10

Function: For-loop executes the code block for values of looping variable from first to last element of looping array.

For-Loop (contd.)

$\frac{n(n+1)}{2} = \frac{\cancel{10}^{5} \times 11}{\cancel{2}} = 55$

Example 1:

$Sum = 1+2+3+4+\dots N$

$1 \qquad 10$

Find the sum of first n natural numbers.

```
clc
clear all
sum=1;
N=10;
for n=2:N
    sum=sum+n;
end
```

looping variable

$[2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10]$

3

$sum = 1, \quad sum = 1+2 = 3$

$sum = 1+2+3 = 6$

1st iteration $Sum = sum + 2$

$sum = 1+2 = 3$

2nd iteration: $sum = sum + 3 = 3+3 = 6$

For-Loop (contd.)

Example 2:

Find the following sum:

$$\sum_{n=1}^{10} n \times (n+1) = \sum_{n=1}^{10} (n^2 + n) = \sum_{n=1}^{10} n^2 + \sum_{n=1}^{10} n$$

$$1\text{x}2 + 2\text{x}3 + 3\text{x}4 + \cdots + 10\text{x}11$$

$2 + 6 + 12$

$=$

```
clc
clear all
product=0;
for n=1:10
   product = product + n*(n+1);
end
```

product = 0

1st iteration: $2 = 0 + 1 \times 2$

$n=2$: $8 = 2 + 2 \times 3$

$n=3$: $20 = 8 + 3 \times 4$

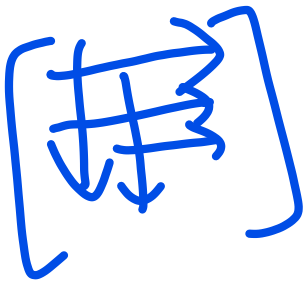
$\vdots$

$( ) = ( ) + 10 \times 11$

For-Loop (contd.)

Demo

very useful when iterating in more than one dimension



Nested For-Loops

A for-loop entirely contained in other for-loop is called a nested for-loop.

2 for loops nested

3 for loops nested

For-Loop (contd.)

Example: Create a 4x4 identity matrix using nested for-loops.

```
N=4;
A=zeros(N);
for i=1:N
   for j=1:N
      if i==j
         A(i,j)=1;
      else
         A(i,j)=0;
      end
   end
end

disp(A)
```

→ dimension of matrix

→ good idea

→ i is row index

A(i,j)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4\times 4}$$

eye(4) =

1st iteration:

i=1 first row

j=1 A(1,1)=1

j=2 A(1,2) = 0

For-Loop (contd.)

Fn + F5 → shortcut to run your ~~code~~ script file

Demo

For-Loop (contd.)

Using break and continue keywords

you want to exit earlier

- break is used to exit the for-loop.
- continue skips the rest of for-loop's code and begins the next iteration.

next value of looping value

Note: In nested for-loops, break only exits the inner-most for-loop in which it is contained.

for
for
break
end
end

For-Loop (contd.)

Example:

$i = 2, 3, 4, \ldots, 38$

Write a function to test if a given number (greater than 2) is prime or not.

```
function s = test_prime(x)
for i = 2:x-1
   if mod(x,i)==0
      s=sprintf('%d is not prime',x);
      break
   elseif i==x-1
      s=sprintf('%d is prime',x);
   else
      continue
   end
end
```

x is natural number input by user

all possible factors

39 is not prime. $39 = 3 \times 13$

37 is prime

$39/2$

$39/3 = 13$

mod(x,i) → gives remainder when x is divided by i

2, 3, 4, ... 36

Conti.

For-Loop (contd.)

Demo

While-Loop

A while loop executes the code as long as a given logical expression is true.

Construction:

doesn't have a looping array

```
while logical expression
code block
end
```

While-Loop (contd.)

Example:

Find all square numbers less than 50.

$1, 4, 9, 16, 25$

$\| \quad \| \quad \| \quad \| \quad \|$

perfect $2^2 \quad 3^2 \quad 4^2 \quad 5^2$

squares

$1^2, 2^2, 3^2, 4^2$

$1, 4, 9, 16 \ldots 49$

$\| \; 7^2$

```
clc
clear all
x=1;
while x^2<50
   disp(x^2)
   x=x+1;
end
```

$x = 1+1 = 2$

While-Loop (contd.)

Demo

Activity

A student wrote an alternative code to find all square numbers less than 50. What is happening with this code?

```
clc
clear all
x=1;
y=x^2;
while y<50
   disp(y)
   x=x+1;
end
```

Open your MATLAB, write this code and interpret the output.

Infinite Loop

An infinite loop runs forever.

Note: In this scenario, use ctrl+c to stop the code execution by MATLAB.

End of Part 1

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 8
Part 2: Recursion

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to

- understand recursion
- create recursive functions
- understand the difference between recursion and iteration
- solve recursion problems e.g. games

Recursion

Recursion occurs when something is defined in terms of itself.

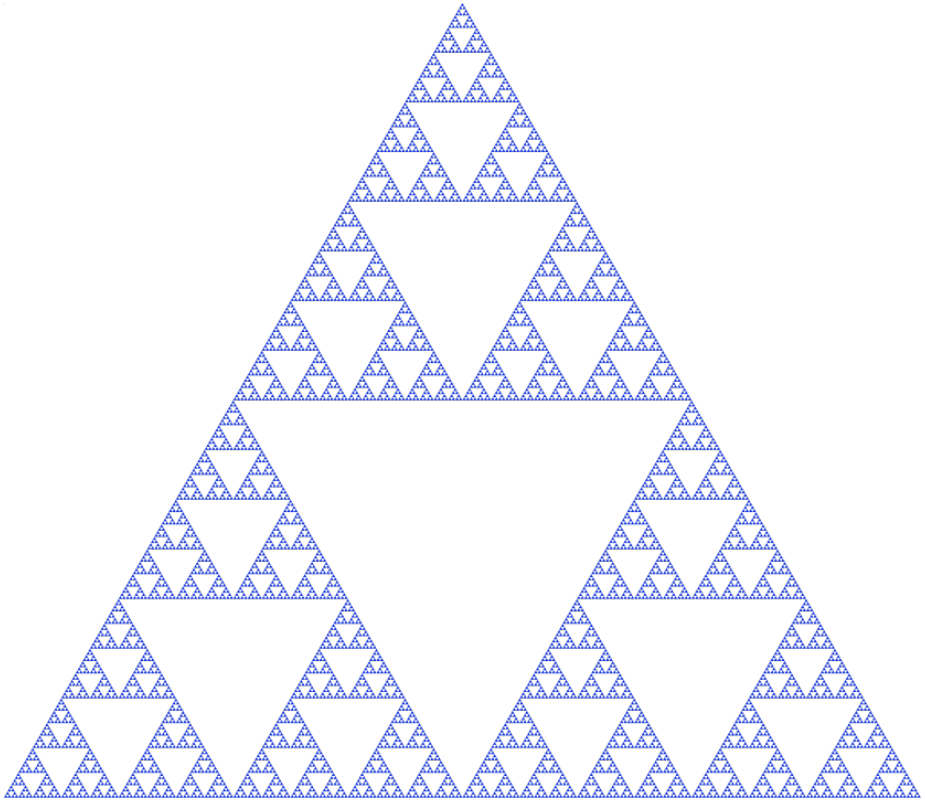
Examples:

- Romanesco broccoli[1]

example in biology

Recursion (contd.)

- Sierpiński triangle [2]



[1] Picture credit: Ivar Leidus
[2] Picture credit: Beojan Stanislaus

Recursive functions

A recursive function is defined in terms of itself.

Examples:

$0! = 1$

- Factorial function

base case

recursive step

$$n! = \begin{cases} 1, & n = 0 \\ n(n-1)!, & n \in \mathbb{N}. \end{cases}$$

$n = 1, 2, 3, \ldots$

$\sqrt{2+\sqrt{2+\sqrt{2+\cdots}}} = ?$

- $f(3) =$

$$f(n) = \begin{cases} \sqrt{2}, & n = 1 \\ \sqrt{2 + f(n-1)}, & n > 1. \end{cases}$$

$f(1) = \sqrt{2}$, $f(2) = \sqrt{2 + f(1)} = \sqrt{2+\sqrt{2}}$

Recursive functions (contd.)

The definition of a recursive function includes

case which doesn't need any recursive formula

- **Base case:** Function's value is given or can be calculated without using recursion.

  1, 1, 2, 3, 5, 8, 13
  n=1

- **Recursive step:** Function's value is calculated by calling to itself.

$$f(n) = \begin{cases} 1, & n = 1, 2 \\ f(n-1) + f(n-2), & n > 2 \end{cases}$$

For example, in factorial function's definition, the cases n=0 and n>0 are base case and recursive step respectively.

n=1, 1
n=2, 1
} → 2 base steps

Recursive functions (contd.)

Example: Write a recursive function in MATLAB to find n!

```
function out = myfactorial(n)
if n==0
out=1;
else
out=n*myfactorial(n-1)
end
end
```

$0! = 1$

$n \times (n-1)!$

$3! = 6$

$5! = 5 \times 4 \times 3 \times 2 \times 1$
$= 120$

~~factorial(n)~~

$n = 3$

out = 3 * myfactorial(2)

= 3 * 2 * myfactorial(1)

= 3 × 2 × 1 × myfactorial(0)

= 6, myfactorial(0) = 1

Recursive functions (contd.)

Demo

Recursive functions (contd.)

The factorial function can also be calculated using while loop. Observe how tricky it is to code now!

```
function out = myfactorial_while(n)
if n==0||n==1
   out=1;
else
   fact=n;
   while n>1
      fact=fact*(n-1);
      n=n-1;
   end
   out=fact;
end
end
```

$0! = 1$, $1! = 1$ $\quad$ $n>1$

$n(n-1) = 2(2-1)$

$5! = 5 \times 4 \times 3 \times 2 \times 1$

not a recursive function.

Recursion vs Iteration

e.g. using while loop was harder

Recursion	Iteration
Easier to code recursive functions	Harder to code recursive functions
creates extra workspace	uses only one workspace
consumes more memory	consumes less memory
code runs rather slow	code runs faster

Tower of Hanoi

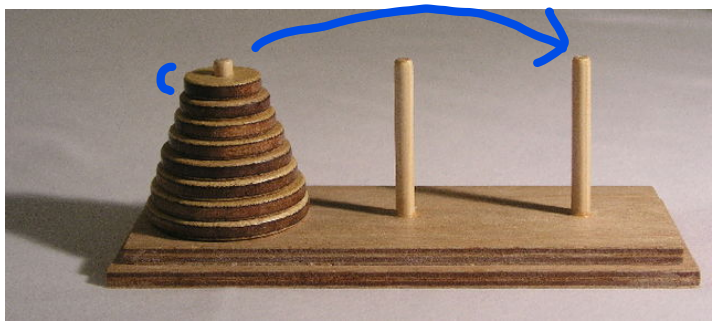


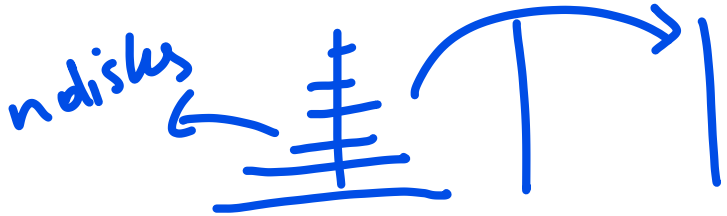
Figure: Tower of Hanoi game with 3 pegs [3]

Goal: To move an entire stack of disks from one peg to another.

Rules:

- Only one ~~peg~~ disk can be used at a time.
- Only top disk of a stack can be moved to an empty peg or top of other stack.
- Larger diameter ~~peg~~ disk cannot go on top of smaller diameter ~~peg~~ disk.

Recursion in Games (contd.)



Question:
What is the minimum number of moves required to move a stack of n disks to other peg?

steps

Possible strategy:
Exploit the recursive property of Tower of Hanoi game.

Recursion in Games (contd.)

The recursive definition of minimum number of moves function $f(n)$ is given as

$$f(n) = \begin{cases} 1, & n = 1 \\ 2 * f(n-1) + 1, & n > 1. \end{cases}$$

Recursion in Games (contd.)

Write a recursive MATLAB function to find the minimum number of moves needed to play Tower of Hanoi game with n disks and 3 pegs.

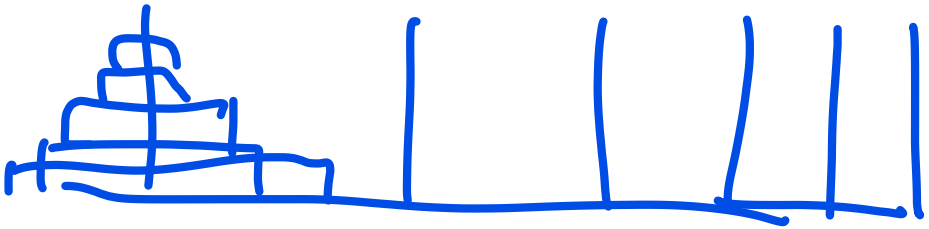
```
function moves = tower_of_hanoi(n)
if n==1
   moves=1;
else
   moves=2*tower_of_hanoi(n-1)+1;
end
end
```

$f(n) = 2^n - 1$

n	1	2	3	4	5	6
f(n)	1	3	7	15	31	63
	1	4-1	8-1	16-1	32-1	64-1

Recursion in Games (contd.)

Demo



Beyond 3 pegs?

beyond scope of this module

Frame-Stewart algorithm can be used to find minimum number of moves. The proof of optimality of this algorithm is still an unsolved problem in Mathematics.

Recreational mathematics
solve puzzles or games

[3] Picture credit: Evanherk

End of Part 2

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 9
Representation of Numbers

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to understand

- Base-N numbers
- Binary numbers
- Floating Point Numbers

Motivation

Sometimes, MATLAB gives unexpected outputs. Consider these examples:

false in MATLAB

- exp(i∗pi)+1==0
- sin(pi), cos(pi/2)
- 5+8e-17>5
- 1 - 3∗(4/3 - 1)
- 2^-1073==0, 2^-1075==0
- sqrt(1e-16 + 1) - 1

$8e-17 = 8\times10^{-17}$

$\sqrt{10^{-16}+1} - 1 > 0$

The goal of this lecture is to understand the reason behind these strange results.

Base-N numbers

Base: Number of unique digits to represent a number.

Example:

Decimal number (Base-10 number)

10 unique digits

1. Uses digits from 0 to 9. So, base is 10.

2. Decimal number can be expanded in powers of 10.

   Examples:

   $582 = 5 \times 10^2 + 8 \times 10 + 2 \times 10^0$,

   300 80 2

   $25.36 = 2 \times 10^1 + 5 \times 10^0 + 3 \times 10^{-1} + 6 \times 10^{-2}$.

   20 5 0·3 0·06

Base-N numbers (contd.)

Other examples:

1011011101

- **Binary number:** Uses only two digits: 0 and 1.

  Base 2

  0 1 2 3 4 5 6 7

- **Octal number:** Uses digits from 0 to 7.

  Base 8 → 8 digits

- **Hexadecimal number:** Uses 16 digits: 0-9 followed by A-F or a-f.

  Base 16

  10 + 6 = 16

Binary numbers

- Binary numbers are represented by only 0s and 1s. So, the base is 2.
- A digit in binary number is called a **bit**.
- **Example:** The number 1001000110.

11010

$60 = 32 + 16 + 8 + 4 + 0\times2^1 + 0\times2^0$

$2^6 = 64$

This number can be converted to a decimal number as follows:

decimal ← 60 ⟶ binary

$$1\times2^9 + 0\times2^8 + 0\times2^7 + 1\times2^6 + 0\times2^5 + 0\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0 = 582$$

512 0 0 64 0 0 0 4 2 0 = 582

111100

Binary numbers (contd.)

Pros

computers treat operations like switches/gates

OR, AND, NOT

- Easier for a computer to store and work with.

add, multiply, divide

- A computer can do all arithmetic operations with them.

$$\begin{array}{r} 101 \\ +101 \\ \hline 1100 \end{array}$$

- A 32-bit computer can represent upto 2^{23} binary numbers.

0 or 1

10

$2 = 1 \times 2^1 + 0 \times 2^0$

$2 = 10$

32 digit binary number

Binary numbers (contd.)

$+(\ )\times 2^1 + (\ )\times 2^0$

Cons

-5.78

1 10 11

1 2 3 4

- Harder for humans to do binary algebra.

gap between two consecutive number

- Limited **range** and **precision** to perform all mathematical calculations.

1.005

32 bit computer

2^{23} values

Floating Point Numbers

MATLAB uses **floating point** numbers or **floats** to achieve the desired range and precision required to perform mathematical calculations.

Note: Floating point numbers are always **rational**. MATLAB approximates irrational numbers (e.g. π) by rational numbers.

Types of float:

$1.\underbrace{3333\ldots\ldots}_{\text{repeating}}\ldots$ | $\underbrace{1.5479}_{\text{rational}}$

- single precision (32 bits)
- double precision (64 bits)

irrational ← non-terminating, non-repeating

Single precision float

$0 \le e \le 255$

$2^8 = 256$

In the **IEEE754** standard for single precision, a float which is represented as

2^{23} values range

$$n = -1^s 2^{e-127}(1+f)$$

range

can take large no. of values

$0 \times 2^{()} + \square \times 2^{()} + \square \times \ldots + \square \times 2^0$

Here, s, e and f are sign indicator, exponent and fraction respectively.

$0 \le x < 1$

0 or 1s

domain → f → range

domain

32 bits are allocated as follows:

- s has 1 bit so it takes values of 0 or 1.
- e has 8 bits so it can take 2^8 values.
- f has 23 bits so it can take 2^{23} values. $0 \le f < 1$.

$-1^0 = 1$

$-1^1 = -1$

fraction

32

0 or 1

Floating Point Numbers (contd.)

Example:

Convert the number

1 10000011 11000000000000000000000 (IEEE754)

into decimal (base 10) number.

$s = 1$

$e = 1\times2^7 + 0 + 0 + 0 + 0 + 0 + 1\times2^1 + 1\times2^0$

$= 128 + 2 + 1$

$= 131$

$n = (-1)^1 2^{131-127} \times (1 + 0.f)$

$f = 1\times2^{-1} + 1\times2^{-2} + \ldots 0 + + 0\times2^{-22} + 0\times2^{-23} = 0.5 + 0.25 = 0.75$

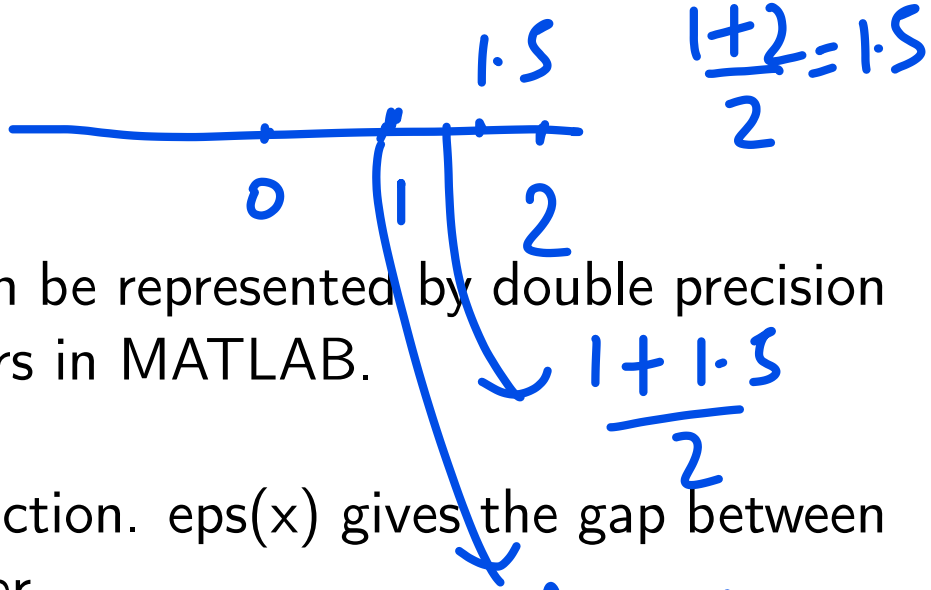
Floating Point Numbers (contd.)

Solution:

$s = 1,$
e=$1\text{x}2^7 + 0 + 0 + 0 + 0 + 0 + 1\text{x}2^1 + 1\text{x}2^0 = 131,$
f=$1\text{x}2^{-1} + 1\text{x}2^{-2} + \cdots = 0.75.$

$$n = -1^1 2^{131-127}(1 + 0.75) = -2^4\text{x}1.75 = -28.$$

Gaps between numbers



Not every decimal (base 10) number can be represented by double precision float. This causes gaps between numbers in MATLAB.

To find the gap, use MATLAB's eps function. eps(x) gives the gap between number x and next representable number.

5 ( ) 6

no number

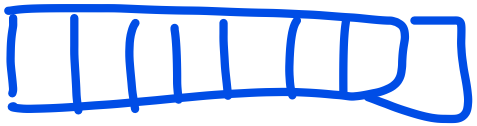
gap

MATLAB takes discrete values

Example:
eps(5)=8.881784197001252e-16.

The gap increases as numbers get large because the factor 2^{e-127} grows in size.

Special cases



1. e=0 (00000000(base 2))
   In this case, the float is calculated using

$$n = -1^s 2^{-126} f$$

$\lim_{n \to 0} \frac{1}{n} \to \infty$

subnormal numbers

2. e=255 (11111111(base 2)) $\to \frac{0}{0}, \frac{Inf}{Inf}$
   - $f \neq 0$: n=NaN (Not a Number)
   - f=0 and s=0: n=Inf
   - f=0 and s=1: n=-Inf

$\frac{1}{0} \to$ undefined

Floating Point Numbers (contd.)

Single precision float interesting facts

$2^7 \times 1 + 2^6 \times 1 + \cdots$

- Largest defined number $1 \times 2^{-1} + \cdots + 1 \times 2^{-22} + 1 \times 2^{-23}$
  0 11111110 11111111111111111111111 $(1+f)$
  (3.40282346638528 9e+38)

- Smallest defined positive 'normal' number
  0 00000001 00000000000000000000000
  (1.175494350822288e-38)
  (2^-126)

Note: Use MATLAB functions realmax('single') and realmin('single') to find the above results.

Floating Point Numbers (contd.)

- Smallest defined subnormal number
  0 00000000 00000000000000000001
  (1.401298464324817e-45)
  (2^-149)

Note: Use MATLAB single() function to represent a number in single precision.

Double precision float

MATLAB uses default double precision of **IEEE754** standard. A double precision is represented as

$$n = -1^s 2^{e-1023}(1+f)$$

Again, s, e and f are sign indicator, exponent and fraction respectively.

64 bits are allocated as follows:

- s has 1 bit so it takes values of 0 or 1.
- e has 11 bits so it can take 2^{11} values.
- f has 52 bits so it can take 2^{52} values. $0 \leq f < 1$.

Floating Point Numbers (contd.)

The special cases for double precision float are similar to single-precision float.

Double precision float interesting facts

- Largest defined number
  1.797693134862316e+308 (type realmax in command window)
- Smallest defined positive 'normal' number
  2.225073858507201e-308 (type realmin in command window)
- Smallest defined subnormal number
  4.940656458412465e-324
  (2^-1074)

Activity

Typing 1e309 in the command window gives the output

1. 1.0000e+309
2. Inf
3. NaN
4. Some error

To answer, please go to mentimeter link in the chat.

Overflow and Underflow

- Any number larger than the largest defined floating point number causes overflow. MATLAB assigns the result to Inf.
- Any number smaller than the smallest defined floating point number causes underflow. MATLAB assigns the result to 0.

End of Lecture 9

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 10
Plotting

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to understand

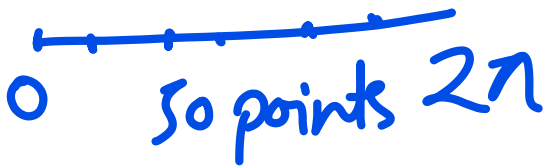
- Plotting 2D figures in MATLAB
- Basic formatting in plots

plot function

MATLAB's plot function can be used to create 2D plots.

Example: This script file creates a plot of y=sinx.

```
x=linspace(0,2*pi,50)
y=sin(x);
plot(x,y)
```



x=0:pi/2:2*pi

To save your figure, type savefig('filename') in the command window.

plot function (contd.)

Demo

Line style, marker and color

shapes o, *, Δ, ...

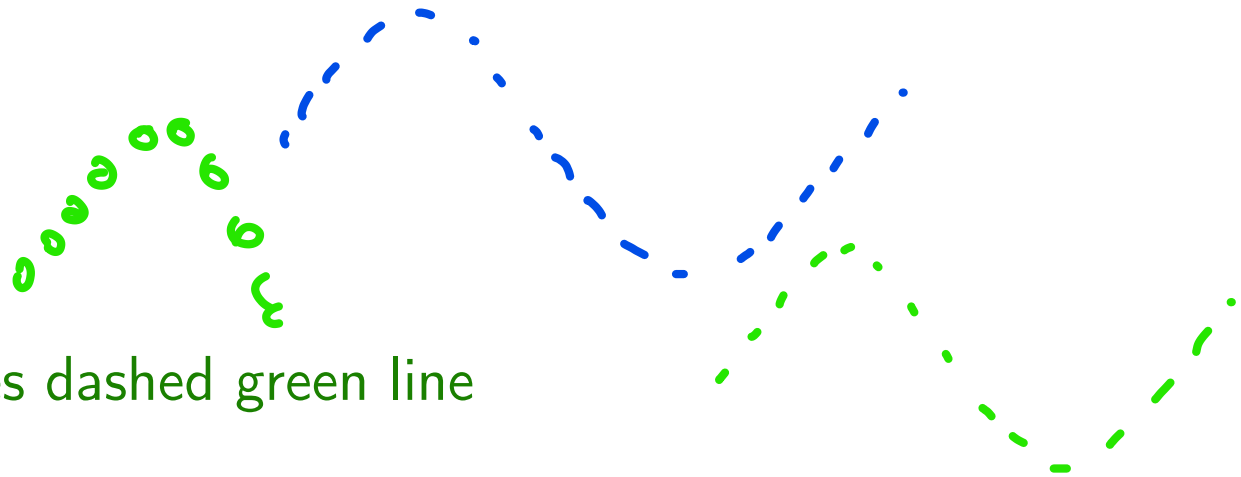
plot(X,Y,LineSpec) creates the plot using the specified line style, marker, and color. Specify the relevant symbol as a string.

For list of colors, list of line styles and markers, refer book.

Example:

go

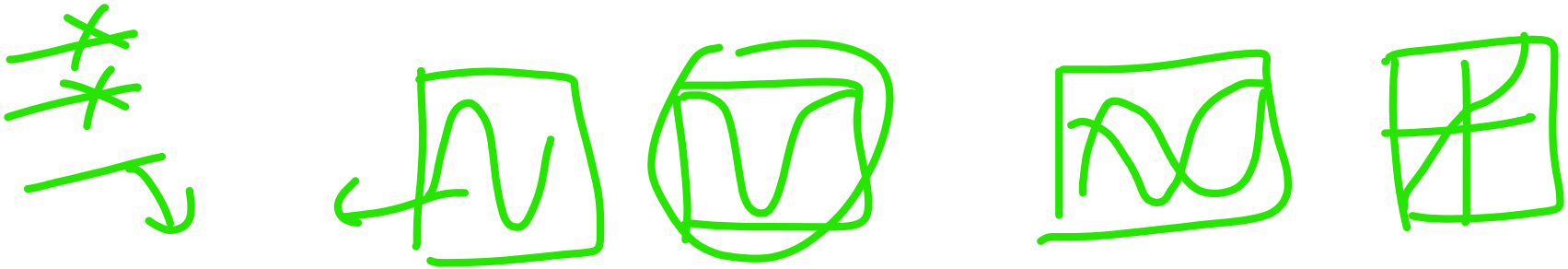
```
x=linspace(0,2*pi,50)
y=sin(x);
plot(x,y,'g- -')    %creates dashed green line
```



Line style, marker and color (contd.)

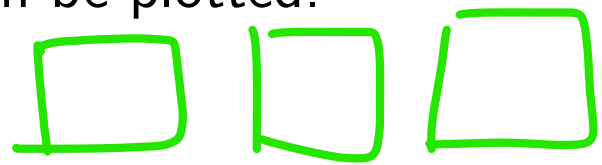
Demo

hold function



hold on

Retains the existing plot so that multiple plots can be plotted.



hold off

Sets the hold state to off and only the final plot will be plotted.

Note: By default, the hold state is off.

hold function (contd.)

Example:

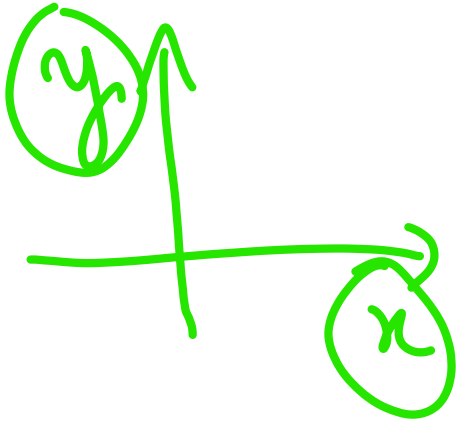
```
x=linspace(0,2*pi,50);
hold on
plot(x,sin(x))
plot(x,cos(x))
hold off
```

Title, Axes labels and Legend

to specify what your plot is all about!

- title() function creates title of plot.
- xlabel() and ylabel() functions create labels for x and y axis respectively.
- legend(label1,label2,...) function creates labels for the plotted data.

legend(sinx , cosx)



Title, Axes labels and Legend (contd.)

Example 1

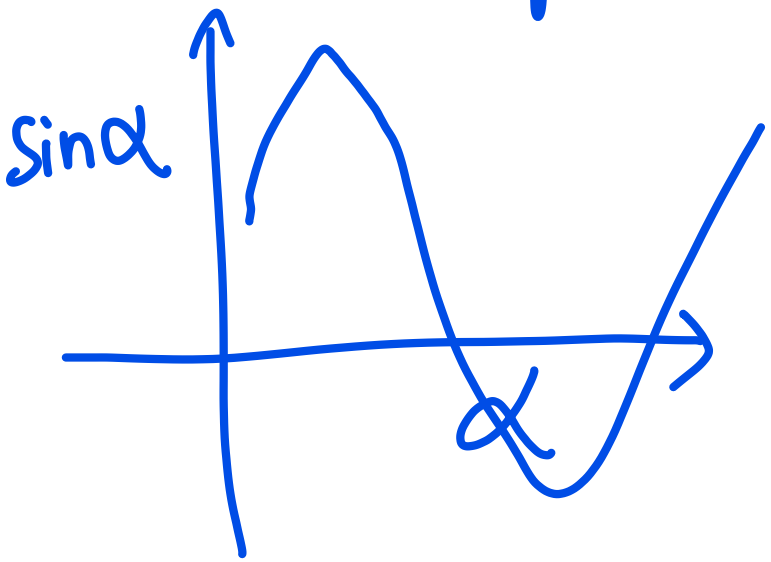
```
x=linspace(0,2*pi,50);
hold on
plot(x,sin(x))
plot(x,cos(x))
title('Plot of sinx and cosx')
xlabel('x') → x
ylabel('y') → y
legend('sin(x)','cos(x)')
hold off
```

Title, Axes labels and Legend (contd.)

Example 2

See lecture recording.

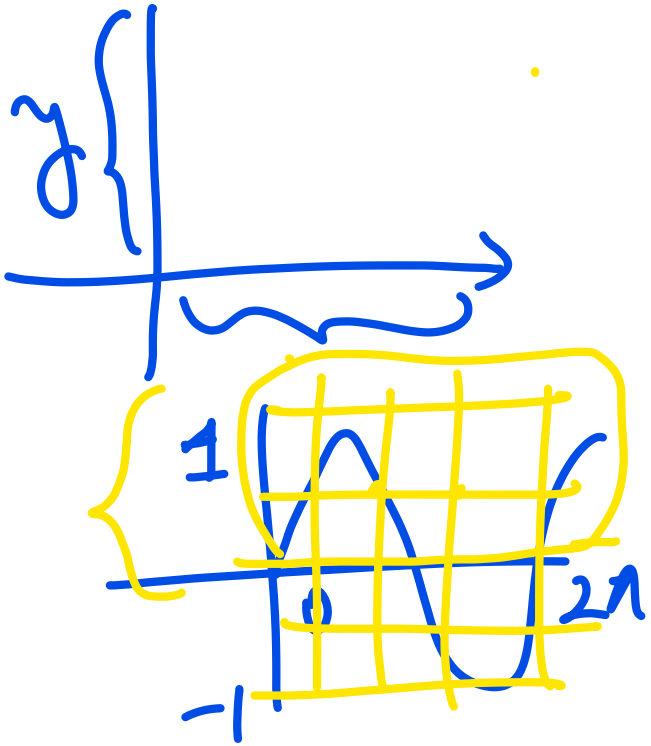
Plot of $\sin\alpha$ vs α



Setting axis and grid

axis([0 2π 0 1])

- axis(limits) function sets axis limits.
- 'grid on' creates a grid to the axis.
- 'grid off' removes the grid.



Here, limits takes the form of a vector [xmin xmax ymin ymax].

Setting axis and grid (contd.)

Example:

```
x=linspace(0,2*pi,50);
hold on
plot(x,sin(x))
plot(x,cos(x))
title('Plot of sinx and cosx')
xlabel('x')
ylabel('y')
legend('sin(x)','cos(x)')
axis([0 2*pi 0 1])
grid on
hold off
```

figure and close all commands

- figure command opens a new figure without overwriting the current figure.

can type in command window

figure → open figure 1

plot(x, sin(x))

figure → open figure 2

plot(x, cos(x))

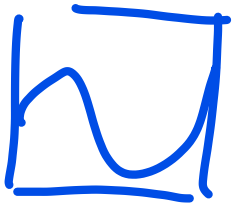
- close all command closes all the figures.

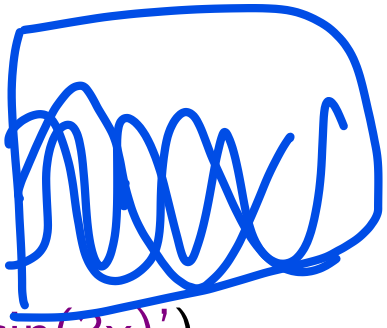
you can use on top of script file

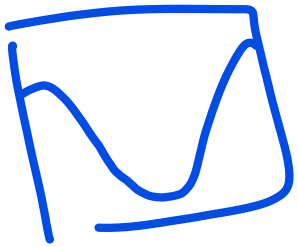
figure and close all commands (contd.)

Example:

```
close all
x=linspace(0,2*pi,50);
figure %opens Figure 1
plot(x,sin(x))
figure %opens Figure 2
%plotting graphs of sinx,sin2x and sin3x on same axes
hold on
plot(x,sin(x))
plot(x,sin(2*x))
plot(x,sin(3*x))
legend('sin(x)','sin(2x)','sin(3x)')
hold off
figure%opens Figure 3
plot(x,cos(x))
```







End of Lecture 10

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 11
Solving System of Linear Equations
Part 1

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to

- understand basic theory of system of linear equations
- use MATLAB to find solutions to the system

Introduction

A system of linear equations is represented as $m \times n$

$m \neq n$

$$\left.\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_2 \end{array}\right\} \tag{1}$$

m equations

$x_1, x_2, \ldots, x_n \rightarrow n$ variables

The aim of this lecture is to find solution to the above system.

Introduction (contd.)

Consider first this simple equation:

$A_{1\times 1}$ special case of $Ax = b$

$$ax = b, \quad a, b \in \mathbb{R}. \tag{2}$$

Find a solution in the following cases.

$\frac{a \times b}{a} = b$

1. $a \neq 0$ $\quad x = b/a$
2. $a = 0$ and $b \neq 0$ $\quad \underbrace{0 \times x}_{=0} = \underbrace{b}_{\neq 0}$
3. $a = 0$ and $b = 0$ $\quad 0 \times x = 0$, $x \in \mathbb{R}$ $\quad 0\times 1 = 0$, $0 \times 2 = 0$, $0 \times 3 = 0$

Introduction (contd.)

The solutions in the three cases:

1. x=b/a (unique solution)
2. $x = \emptyset$ (no solution) empty set
3. $x \in \mathbb{R}$ (infinitely many solutions)

Introduction (contd.)

Now consider this matrix equation: $x = \underline{0}$

$$Ax = b \tag{3}$$

x & x^*

$x = x^*$ exercise

where A is a n x n matrix. Find a solution in the following cases.

square matrix

1. $|A| \neq 0$ → $x = A^{-1}b$ unique solution; A is non-singular
2. $|A| = 0$ and $b \neq \underline{0}$ → zero vector → no solution or infinitely many solutions
3. $|A| = 0$ and $b = \underline{0}$ → infinitely many sol^n

$Ax = b$

$A^{-1}Ax = A^{-1}b$

$Ix = A^{-1}b$

$x = A^{-1}b$

rank(A) < n when A is singular.

Introduction (contd.)

The solutions in the three cases:

$\underbrace{Ax = \underline{0}}$

$x = \underline{0}$ $\quad |A| = 0$

1. $x = A^{-1}b$ (unique solution)
2. $x = \emptyset$ (no solution) $\quad b = \underline{0}$
3. $x = N(A)$ (infinitely many solutions)

Here, $N(A)$ means nullspace of matrix A.

A^{-1} exists

$x = A^{-1}b$

$= 0$

Question: What is the solution to (3) when $|A| \neq 0$ and $b = 0$?

$Ax = b$

Backslash operator

$x = A\backslash b$ solves the system (1) of linear equations $Ax = b$.

back slash

forward slash

Example: Solve the system of equations:

$$\begin{aligned} 2x + y &= 4 \\ x - y &= -1 \end{aligned} \tag{4}$$

$[A\ b] = \begin{bmatrix} 2 & 1 & 4 \\ 1 & -1 & -1 \end{bmatrix}$

Here, $A = \begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix}$ and $b = \begin{bmatrix} 4 \\ -1 \end{bmatrix}$.

$2 \times 1 + 2 = 4$

$1 - 2 = -1$

So, $x = A\backslash b$ gives $x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

$x = A^{-1} b$

inv(A) * b

Question: Can you find this solution using MATLAB's inv() function?

inv(A) = A^{-1}

Backslash operator (contd.)

Demo

Backslash operator (contd.)

$Ax = b$

$ax = b$

- If A is a square matrix then $A\backslash b$ and $inv(A) * b$ are equivalent.
- For scalars a and b, $a\backslash b$ solves the equation $ax = b$. So, $a\backslash b$ and b/a are equivalent.

Backslash operator (contd.)

Let us return back to system of equations (1).

For solving this system, $x = A\backslash b$ gives unexpected results when

1. the system has no solution. → no need to find inv(A)*b using MATLAB
2. the system has infinitely many solutions. In this case, a particular solution may be found using $x = pinv(A) * B$. Here, $pinv(A)$ computes the 'pseudo-inverse' of A.

$Ax = b \Rightarrow x = A^{-1}b$

A'

$Ax = b$

$A^T A x = A^T b$

$(A^T A)^{-1} A^T A x = (A^T A)^{-1} A^T * b$

Backslash operator (contd.)

Exercise

When A^{-1} exists then $A^{-1} = (A^T A)^{-1} A^T$

$$\underset{\substack{\| \\ x}}{I x} = \underbrace{(A^T A)^{-1} A^T}_{\text{pseudo inverse}} b$$

Demo

Rank of a matrix

To find out if the system (1) has a unique or infinitely many solutions, we need to understand 'rank' of a matrix.

Definition

Rank of a matrix A is defined as the maximum number of linearly independent rows/columns of A.

$v_1 \nparallel v_2$

Question: Find out the rank of these matrices.

$$A = \begin{bmatrix} 5 & 4 \\ 3 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 4 & 8 \end{bmatrix}$$

$\lambda_1 v_1 + \lambda_2 v_2 = 0$ if & only if $\lambda_1 = \lambda_2 = 0$

MATLAB's rank() function finds the rank of a matrix.

for big matrices, it is very useful.

Rank of a matrix (contd.)

Demo

Rank method

rank test

Rank can be used to determine if the system (1) has no solution, unique solution or infinitely many solutions.

n = no. of variables of A → RHS $\neq 0$

n = no. of columns

- Non-homogeneous equations ($Ax = b$, $b \neq 0$)

$[A \;\; b]$ → augmented matrix of $Ax = b$

$\left[\begin{array}{c|c} A & b \end{array}\right]$

  1. $rank(A) < rank([A \quad b]) \implies$ No solution
  2. $rank(A) = rank([A \quad b]) = n \implies$ Unique solution
  3. $rank(A) = rank([A \quad b]) = k < n \implies$ Infinitely many solutions

- Homogeneous equations ($Ax = 0$) $n \leq m$

  1. $rank(A) = n \implies$ Unique solution (the trivial solution) → $x = \underline{0}$
  2. $rank(A) = k < n \implies$ Infinitely many solutions

Sol^n always exists because $\underline{0}$ is always a sol^n

Finding solutions

For non-homogeneous equations:

- First, check the existence of solution using Rank method.
- If the solution exists and is unique, find the solution using $x = A \backslash b$.
- If there are infinitely many solutions, first find the particular solution (say x^*) using $x^* = pinv(A) * b$. The general solution is given by $x = x^* + N(A)$ where $N(A)$ is the nullspace of A.

$A(x^* + N(A)) = b$

$Ax^* = b$

$A(x^* + X) = b$

$AX = 0$

Finding solutions (contd.)

Rank-nullity theorem

Follow these steps to find the nullspace $N(A)$:

$\text{rank}(A) + \underbrace{\text{nullity of } A}_{p} = n$

- Use MATLAB's null(A) function to create a matrix containing orthonormal basis of N as column vectors.

no. of columns

- Let $P =$ null(A) and $p =$nullity of A. Then $p = n - rank(A)$, (**Why?**)

size(P, 2)

no. of columns in null(A) ~~vectors~~ or dimension of basis of A.

- The nullspace of A is then given by

size(P, 1)

$$N(A) = c_1 * P(:, 1) + c_2 * P(:, 2) + \cdots + c_p P(:, p)$$

where the constants $c_1, c_2, \cdots, c_p \in \mathbb{R}$.

Finding solutions (contd.)

Example: Solve the system of equations:

$$\begin{aligned} x + y + z + w &= 6 \\ x + 2y - 3z - w &= -4 \\ y - 4z - 2w &= -10 \\ 2x + 3y - 2z &= 2 \end{aligned} \tag{5}$$

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & -3 & -1 \\ 0 & 1 & -4 & -2 \\ 2 & 3 & -2 & 0 \end{bmatrix} \qquad b = \begin{bmatrix} 6 \\ -4 \\ -10 \\ 2 \end{bmatrix}$$

Finding solutions (contd.)

Demo

Finding solutions (contd.)

For homogeneous equations:

- Find $rank(A)$. If $rank(A) = n$, then the trivial solution $x = \underline{0}$ is the only solution.
- Otherwise, the solution is $x = N(A)$.

End of Lecture 11

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 12
Solving System of Linear Equations
Part 2

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to

- understand Gaussian elimination method
- implement this method in MATLAB

Introduction

Gaussian elimination is a method for solving system of linear equations. It consists of two steps:

- Forward elimination
- Back substitution

Gaussian Elimination method

Please refer to notes by Steven Pav.

Gaussian Elimination method (contd.)

Demo

End of Lecture 12

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 13
Least Squares Regression

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

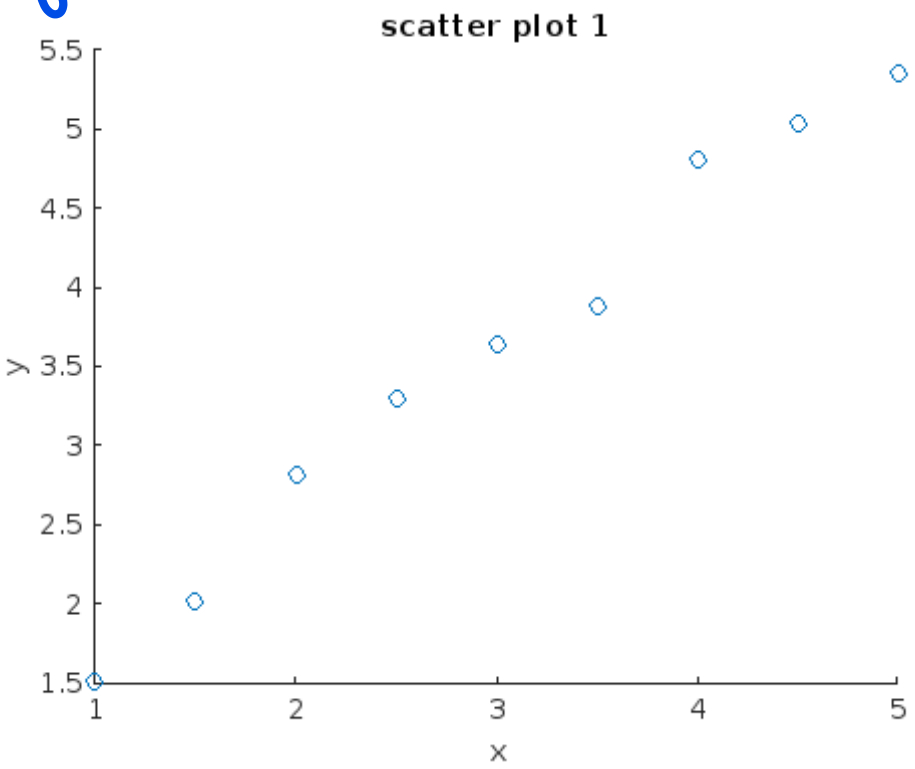
At the end of lecture, students will be able to

- understand scatter plot and regression
- understand theory of Least Squares Regression
- solve basic regression problems in MATLAB

Scatter plot

A **scatter plot** plots two different sets of data using dots. Unlike line plot, the dots are not connected by a curve.

2D scatterplot plots data of two different variables as dots.



plot(x,y)

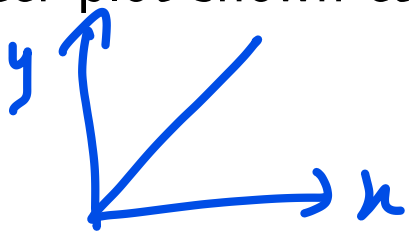
scatter(x,y)

plot(x,y,'o')

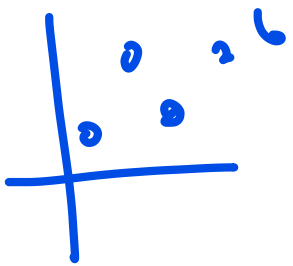
Scatter plot (contd.)

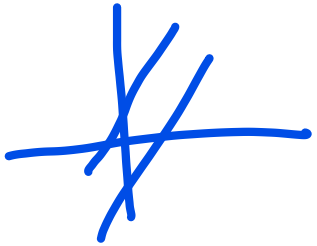
MATLAB's scatter() function can be used to create a scatter plot.

Example: The code below creates the scatter plot shown earlier.




```
x=1:0.5:5; %create data for vector x
s=size(x); %find size of vector x
y=x+rand(s); %create a vector y=x+'some random values'
scatter(x,y) %create scatter plot
title('scatter plot 1')
xlabel('x')
ylabel('y')
```





Scatter plot (contd.)

Demo

Regression

Regression is a statistical technique used to find the 'best-fit curve' that describes a scatter plot.

curve which is closest approximation to the data

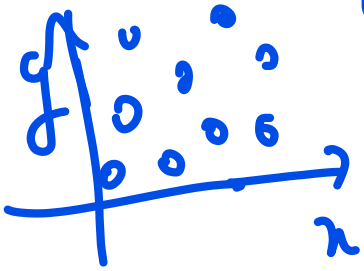
Depending on the data trend in a scatter plot, one may use

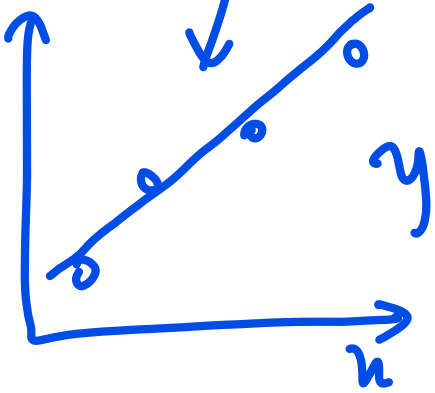
- ~~Linear Regression~~

Linear Regression curve

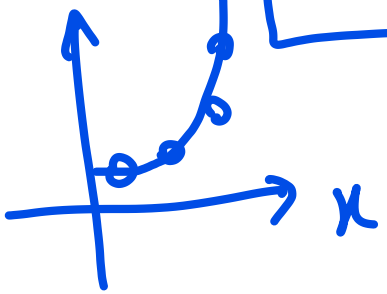
- ~~Non-linear Regression~~

Non-linear Regression curve





$y = mx + c$



Linear Regression curve example

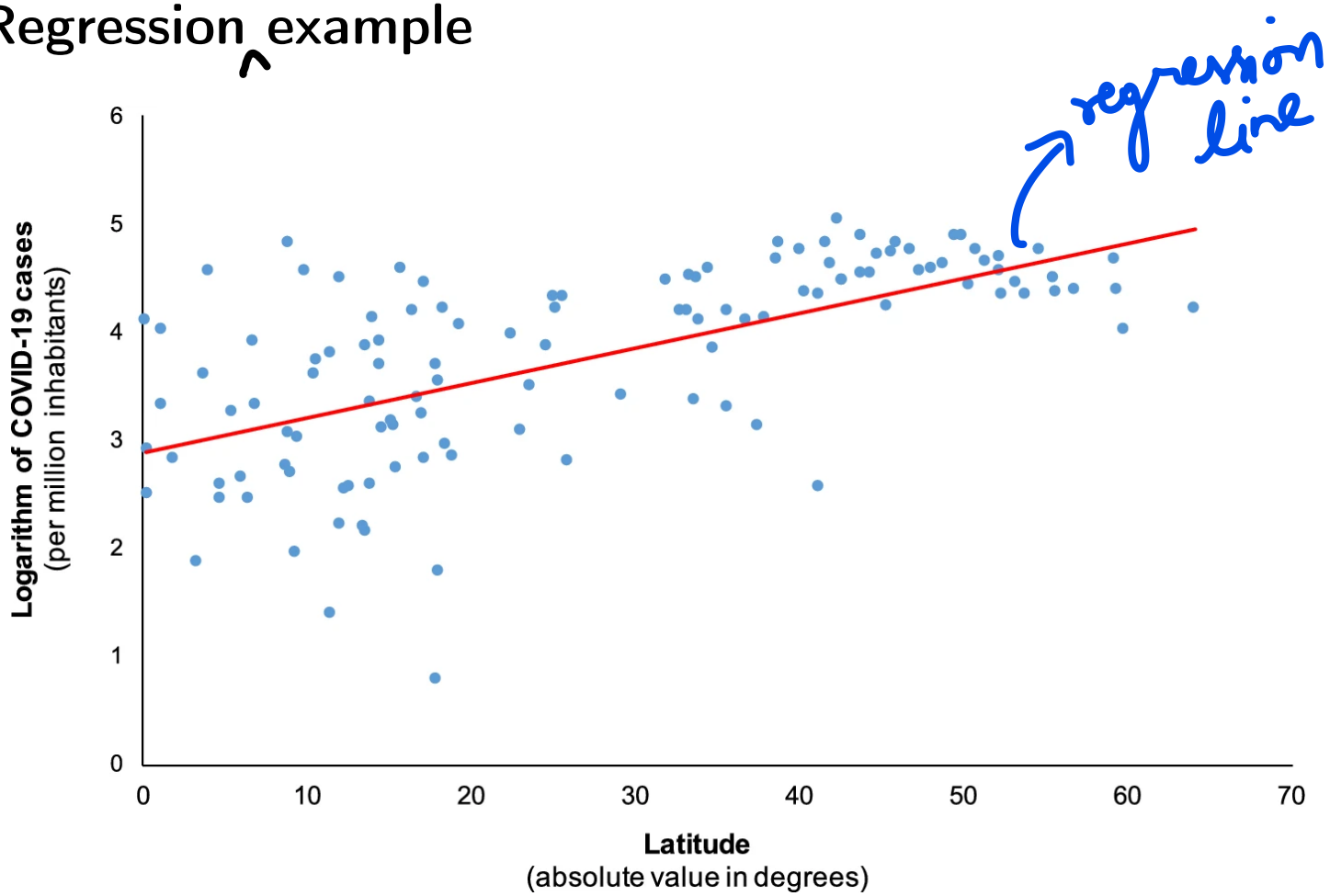


Figure: Scatter plot showing linear trend [1]

Non-linear Regression curve example

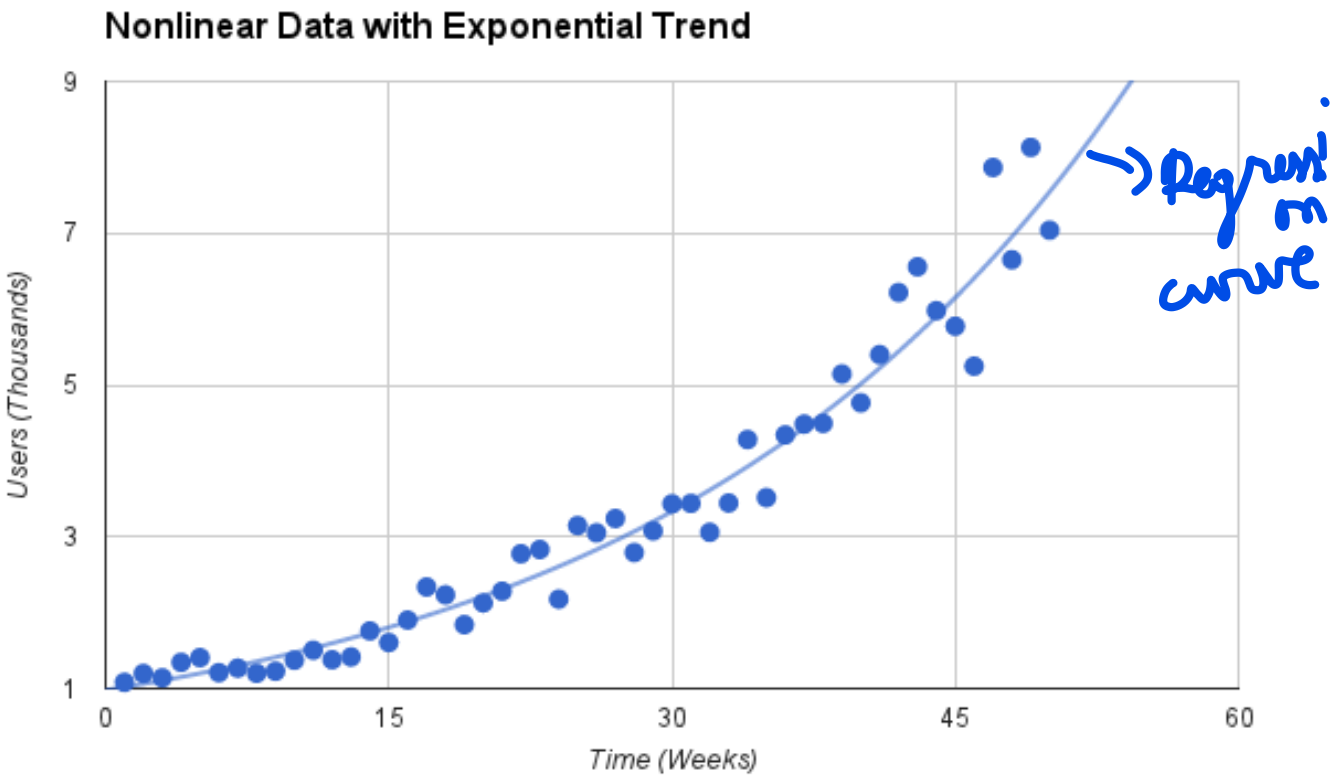


Figure: Data showing number of active users on a website with time [2]

[1] Chen, S., Prettner, K., Kuhn, M. et al. Climate and the spread of COVID-19. Sci Rep 11, 9042 (2021). https://doi.org/10.1038/s41598-021-87692-z

[2] http://sam-koblenski.blogspot.com

Regression model

linear function
or nonlinear function

A regression model provides a function to describe the relationship between one (or more) independent variables and a dependent variable.

A basic regression model is the 'Least Squares Regression model'.

Least Squares Regression

$m >> n$

Here, the relationship between dependent data points $y_i (i = 1, 2, ...m)$ and independent data points x_i is modelled as

For m data points,

$$\hat{y}(x) = \sum_{i=1}^{n} \alpha_i f_i(x) \tag{1}$$

$\alpha_1 f_1(x) + \alpha_2 f_2(x) + \cdots f_n(x)$

will give m eqns.

Plot your regression curve

where

m parameters

- $\hat{y}(x)$ is an estimation function
- α_i are parameters of estimation function
- $f_i(x)$ are linearly independent basis functions

$\hat{y}(x_1) = \sum_{i=1}^{n} \alpha_i f_i(x_1)$

$\hat{y}(x_2) = \cdots f_i(x_2)$

$\vdots$

$\hat{y}(x_m) \cdots$

Least Squares Regression (contd.)

The parameters are then found by minimising the total squared error E.

$$E = \sum_{i=1}^{m} (\hat{y} - y_i)^2 \tag{2}$$

Substituting (1) in (2) gives

$$E = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \alpha_j f_j(x_i) - y_i \right)^2 \tag{3}$$

E is a function of n variables namely $\alpha_j (j = 1, 2, \cdots, n)$.

$$\frac{\partial E}{\partial \alpha_j} = 0$$

Least Squares Regression (contd.)

The solution for n parameters α_j which minimise the total squared error E is given as

$$\beta = pinv(A) * Y \tag{4}$$

Here,

- β is a column vector with n entries α_j
- A is a mxn matrix with entries $A(i,j) = f_j(x_i)$
- $pinv(A)$ is the pseudo-inverse of A
- Y is a column vector with m entries y_i

Least Squares Regression (contd.)

Derivation of (4)**:** Please refer book and lecture recording

Least Squares Regression (contd.)

Example: Perform a least squares regression for the scatter plot created before using estimation function $\hat{y}(x) = \alpha_1 x + \alpha_2$.

$f_1(x) = x$, $f_2(x) = 1$

handle

```
openfig('scatter_plot_1.fig') %opens figure scatter_plot_1.fig
a = get(gca,'Children');
x = get(a, 'XData'); %extract x-data points
y = get(a, 'YData'); %extract y-data points
A=[x',ones(size(x'))]; %create the matrix A of basis functions
beta=pinv(A)*y'; %evaluate vector beta containing values of parameters

%plot the regression line
hold on
plot(x,beta(1)*x+beta(2))
```

$F_1(x)$

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} \quad m \times 2$$

$\beta = pinv(A) * Y$

$$\beta = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}$$

$\alpha_1 = Beta(1)$, $\alpha_2 = Beta(2)$

Least Squares Regression (contd.)

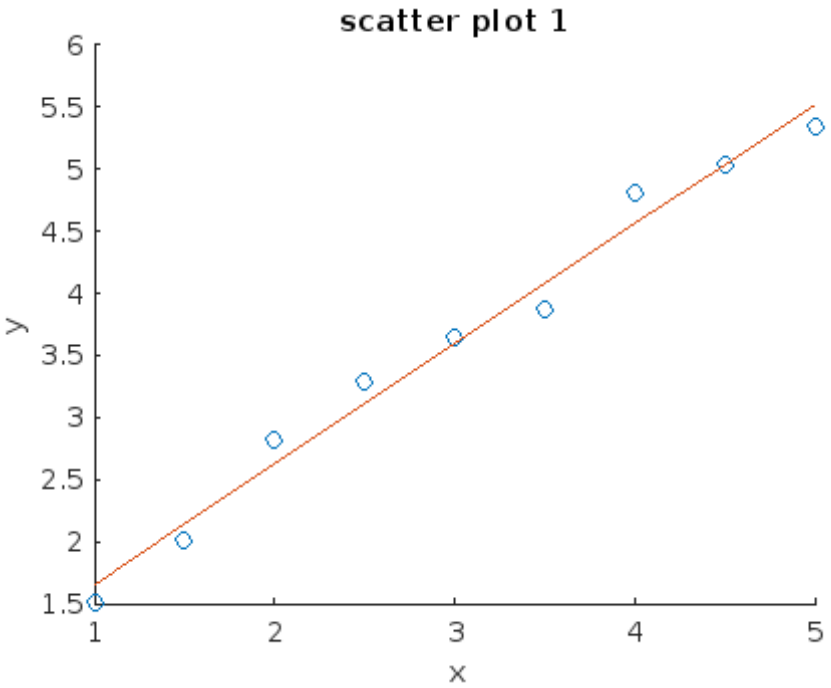


Figure: Regression line

polyfit and polyval functions

When the estimation function $\hat{y}(x)$ is a polynomial, MATLAB's polyfit and polyval functions can be used.

For vectors x and y containing x-data and y-data points respectively,

- p=polyfit(x,y,n) creates a vector p of the coefficients of regression polynomial p(x) of degree n
- polyval(p,x) calculates the values of p(x)

$y = \alpha_1 x + \alpha_2$

$n = 1$

$p = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}$ p = polyfit(x,y,1)

polyfit and polyval functions (contd.)

Example: This code creates the same regression line as was shown before.

```
openfig('scatter_plot_1.fig') %opens figure scatter_plot_1.fig
a = get(gca,'Children');
x = get(a, 'XData'); %extract x-data points
y = get(a, 'YData'); %extract y-data points

p= polyfit(x,y,1); %creates coefficients of regression polynomial of degree 1
Y=polyval(p,x); %evaluates the value of polynomial at x-data points
hold on
plot(x,polyval(p,x)) %plot the regression polynomial
```

polyfit and polyval functions (contd.)

Demo

Nonlinear Estimation Functions

$\frac{1}{\alpha x+\beta}$

Sometimes, a nonlinear estimation function provides the best fit for a scatter plot. This means we require

non-linear in α_j

$y = \frac{1}{x}, \sin x, \log x$

2^x

$$\hat{y}(x) = g(\alpha_1, \alpha_2, \cdots, \alpha_n, x) \tag{5}$$

$\hat{y} = \sum_{i=1}^{m} \alpha_j \cancel{f_j(x_i)}$

where g is some nonlinear function.

In some special cases, a transformation such as

$$\tilde{y}(x) = h(\hat{y}(x)) \tag{6}$$

$\hat{y} = \alpha_1 x^2 + \alpha_2 x + \alpha_3$

$f_1(x) = x^2, \; f_2(x) = x$

$f_3(x) = 1$

can linearise the equation (5) into (1).

h is a clever transformation

Nonlinear Estimation Functions (contd.)

Example: Consider the estimation function

$\alpha_1(\ ) + \alpha_2(\ )$

$$\hat{y}(x) = \alpha_1 e^{\alpha_2 x} \tag{7}$$

$\log \hat{y} = \log(\alpha_1 e^{\alpha_2 x})$

$= \log \alpha_1 + \log(e^{\alpha_2 x})$

$\beta = \text{pinv}A \cdot Y$

Applying the transformation

$\beta = \begin{bmatrix} \tilde{\alpha}_1 \\ \alpha_2 \end{bmatrix}$

$$\tilde{y}(x) = log(\hat{y}(x)) \tag{8}$$

converts (7) into

$$\tilde{y}(x) = \tilde{\alpha}_1 + \alpha_2 x \tag{9}$$

$= \log \alpha_1 + \alpha_2 x$

where we define $\tilde{\alpha}_1 = log(\alpha_1)$. Now, least squares regression can be applied to equation (9).The parameter α_1 can be found using $\alpha_1 = e^{\tilde{\alpha}_1}$.

$\alpha_1 = e^{\tilde{\alpha}_1}$

$f_1(x) = 1, \quad f_2(x) = x$

End of Lecture 13

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 14
Interpolation

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

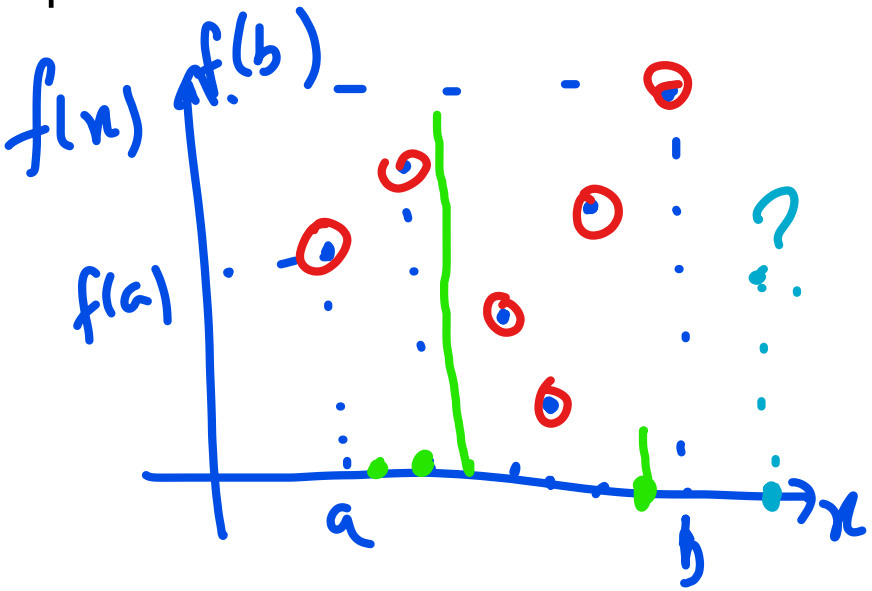
At the end of lecture, students will be able to

- understand interpolation problem
- understand theory of interpolation methods
- implement interpolation methods in MATLAB

Introduction

In mathematics, **interpolation** means to estimate the value of a function $f(x)$ in a given interval $x \in [a, b]$ based on some known values of the function inside and at end points of this interval.

extrapolation



Introduction (contd.)

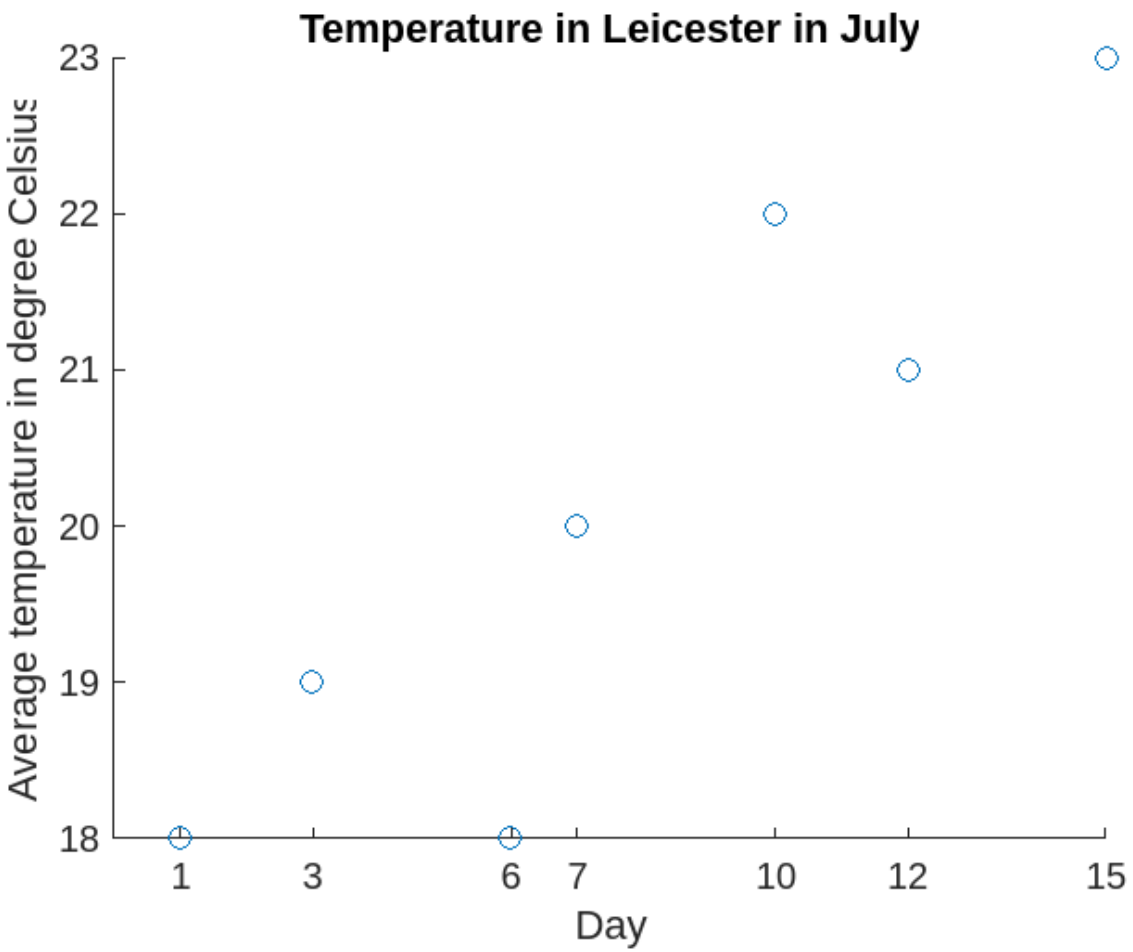
Example: The table below shows the average temperatures in Leicester in July. Can we predict temperature on other days between 1st and 15th July?

Day	Average Temperature °C
1	18
3	19
6	18
7	20
10	22
12	21
15	23

2nd July → ?

Days = 2, 4, 5, 8, 9, 11, 13, 14

Introduction (contd.)



Interpolation vs Regression

- Both techniques are used to describe the given data set as good as possible.
- Unlike regression, interpolation requires the estimation function to pass through all data points.

Interpolation Problem Statement

Suppose we have a data set containing n data points $(x_i, y_i), i = 1, 2, \cdots, n$.

Goal: To find an estimation function $\hat{y}(x)$ with domain $x \in [x_1, x_n]$ such that $\hat{y}(x_i) = y_i$. $\hat{y}$ should pass through all data points.

The function $\hat{y}(x)$ is called **interpolation function**.

Note: The choice of interpolation function depends on other factors such as accuracy, underlying physics etc. Depending on this choice, there are different interpolation methods.

Some Interpolation methods

- Linear Interpolation
- Cubic Spline Interpolation
- Lagrange Polynomial Interpolation

Linear interpolation

Here, the interpolation function $\hat{y}(x)$ is defined piecewise by linear polynomials (or straight lines). So,

$mx + c$

$$\hat{y}_i(x) = y_i + \frac{(y_{i+1} - y_i)(x - x_i)}{x_{i+1} - x_i}, \quad x_i \le x \le x_{i+1} \quad (i = 1, 2, \cdots, n-1) \tag{1}$$

$\hat{y}(x)$ $\quad \hat{y}_1 \quad \hat{y}_2 \quad \hat{y}_{n-1}$ $\quad$ $n-1$ segments

(x_1, y_1) $\quad (x_2, y_2)$ $\quad (x_3, y_3)$ $\quad (x_{n-1}, y_{n-1})$ $\quad (x_n, y_n)$

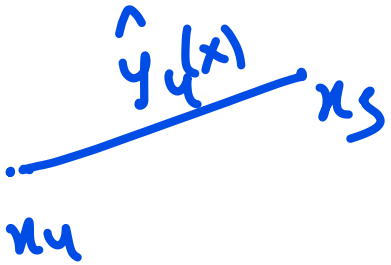
MATLAB's interp1() function can be used to make life easier.

$$\hat{y}_4(x) = y_4 + \frac{(y_5 - y_4)}{x_5 - x_4}(\underset{8}{x} - x_4) = 20.66\cdots$$

Linear interpolation (contd.)

Example: Use linear interpolation to find the average temperature in Leicester on 8th July.

	Day	Average Temperature °C	
x_1	1	18	y_1
x_2	3	19	y_2
x_3	6	18	y_3
x_4	7	20	y_4
x_5	10	22	y_5
	12	21	
	15	23	



Linear interpolation (contd.)

Demo

Cubic spline interpolation

A cubic spline is a function defined piecewise by cubic polynomials.

In cubic spline interpolation, the interpolating function is a cubic spline defined as

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \quad x_i \le x \le x_{i+1} \quad (i = 1, 2, \cdots, n) \quad (2)$$

n-1 cubics

Again, MATLAB's interp1 function can be used by giving 'cubic' as argument.

Cubic spline interpolation (contd.)

Example: (from book)

Cubic spline interpolation (contd.)

Demo

Cubic spline interpolation (contd.)

The unknown parameters a_i, b_i, c_i and d_i are found using these conditions:

$$S_i(x_i) = y_i \quad (i = 1, 2, \cdots, n-1) \tag{3a}$$

cubic should pass through nearest point on the left x_i

$$S_i(x_{i+1}) = y_{i+1} \quad (i = 1, 2, \cdots, n-1) \tag{3b}$$

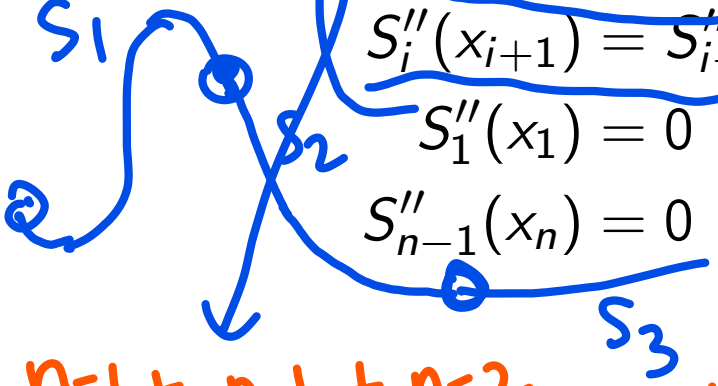
$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \quad (i = 1, 2, \cdots, n-2) \tag{3c}$$

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \quad (i = 1, 2, \cdots, n-2) \tag{3d}$$

$$S_1''(x_1) = 0 \tag{3e}$$

$$S_{n-1}''(x_n) = 0 \tag{3f}$$

curvature should be equal



n data points
↓
n-1 cubics
↓
4(n-1) parameters
↓
4(n-1) equations
↓
4n-4 eq's

$n-1 + n-1 + n-2 + n-2 = 4n-6$

2 eq's

Cubic spline interpolation (contd.)

Derivation of parameters: Please refer book and lecture recording.

Lagrange Polynomial Interpolation

Here, the interpolation function is a **Lagrange polynomial** defined as

$$L(x) = \sum_{i=1}^{n} y_i P_i(x) \tag{4}$$

where

$$P_i(x) = \prod_{j=i, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}. \tag{5}$$

The Lagrange polynomial $L(x)$ of degree $n-1$ passes through n data points i.e. it satisfies $L(x_i) = y_i \quad (i = 1, 2, \cdots, n)$.

Lagrange Polynomial Interpolation (contd.)

Example: (from book)

End of Lecture 14

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 15
Taylor series

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to

- understand Taylor polynomials
- understand Taylor series
- find Taylor series approximations and estimate error

Introduction

Many functions in Mathematics can be approximated by polynomials upto desired accuracy.

To perform such approximations, we need to understand **Taylor polynomials** of a function.

Taylor polynomials

A Taylor polynomial of a function $f(x)$ centered at $x = a$ is a polynomial approximation of $f(x)$.

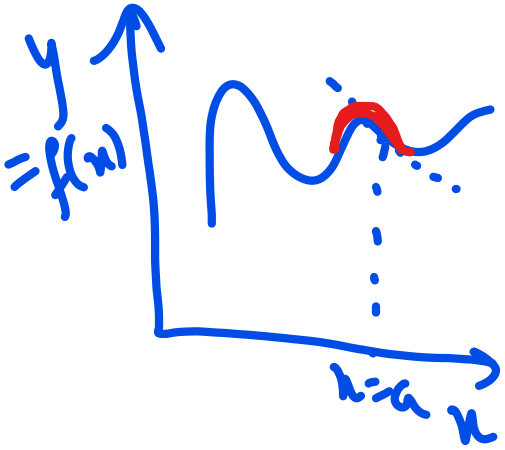


Figure: Brook Taylor

Taylor polynomials (contd.)

Question: Find a quadratic polynomial to approximate $f(x)$ near $x = a$.

$$p_2(x) = \frac{f''(a)}{2}(x-a)^2 + f'(a)(x-a) + f(a)$$

$$= f(a) + f'(a)(x-a) + f''(a)(x-a)^2$$

Taylor polynomials (contd.)

A *nth* order Taylor Polynomial is given as

$$p_n(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n. \quad (1)$$

- $p_0(x) = f(a)$ (**constant function**)
- $p_1(x) = f(a) + \frac{f'(a)}{1!}(x-a)$ (**linear approximation**)
- $p_2(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2$

(**quadratic approximation**)

Do it as an exercise

Taylor polynomials (contd.)

at $x = 0$ $(a=0)$

Example: The Taylor polynomials of $f(x) = \cos x$ upto degree 6:

$\cancel{p_0(0)}$

$p_0(x) = f(0) = \cos 0 = 1$

$p_1(x) = f(0) + \frac{f'(0)}{1!}(x-0)$

$= 1 + f'(0)x$

$f'(x) = -\sin x$

$f'(0) = 0$

- $n = 0: \quad p_0(x) = 1$
- $n = 2: \quad p_2(x) = 1 - \frac{x^2}{2}$
- $n = 4: \quad p_4(x) = 1 - \frac{x^2}{2} + \frac{x^4}{24}$
- $n = 6: \quad p_6(x) = 1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720}.$

$p_2(x) = f(0) + \frac{f'(0)}{1!}(x-0) + \frac{f''(0)}{2!}(x-0)^2 = 1$

Note: It is possible that a *nth* order Taylor polynomial is not of degree n.

$= 1 - \frac{x^2}{2}$

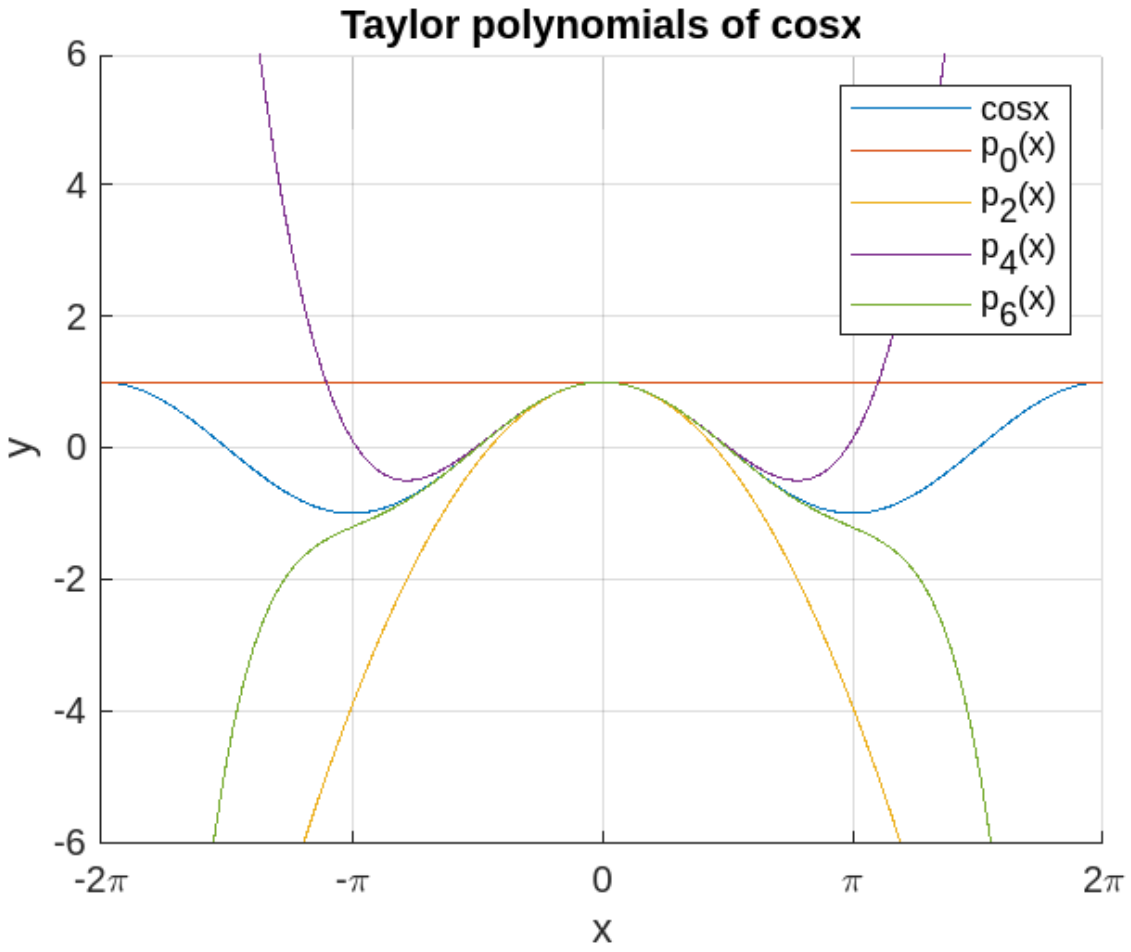
Taylor polynomials (contd.)

Example: Write a script file to plot the Taylor polynomials of $\cos x$ upto degree 6.

Taylor polynomials (contd.)

Demo

Taylor polynomials (contd.)



Taylor series

A Taylor series is an infinite series expansion of a function at a given point in its domain.

Taylor series of a real-valued function $f(x)$ at $x = a$ is defined as

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots \tag{2}$$

or

$$f(x) = \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n \tag{3}$$

Taylor series (contd.)

special case of Taylor series

When $a = 0$, we obtain a **Maclaurin series**.

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \cdots \tag{4}$$

Taylor series (contd.)

Example: Find the Taylor series of $\cos x$ at $x = 0$.

$$f(x) = f(0) + f'(0)x + \frac{f''(0)x^2}{2!} + \frac{f'''(0)x^3}{3!} + \frac{f''''(0)x^4}{4!}$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots$$

$$= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

Taylor's theorem

Theorem

Suppose a function $f(x)$ has $n+1$ continuous derivatives in an open interval I containing $x = a$ then $\forall n$ and $\forall x \in I$,

$$f(x) = \underbrace{f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f''(n)}{2!}(x-a)^n}_{n\text{th order Taylor polynomial}} + R_n(x) \quad (5)$$

where

$$R_n(x) = \frac{f^{n+1}(c)}{(n+1)!}(x-a)^{n+1} \quad (6)$$

$R_n(x) \to 0$ as $n \to \infty$

for some c between a and x.

Here, $R_n(x)$ is called the Taylor remainder or error term.

Error estimation

We don't know the exact value of $R_n(x)$ since exact value of c is unknown. However, an upper bound on the error term can still be found.

If $|f^{n+1}(x)| \leq M$ for all t between a and x, then

$$|R_n(x)| \leq \frac{M}{(n+1)!}|x-a|^{n+1} \tag{7}$$

Lagrange upper bound on remainder

Error estimation (contd.)

$$\cos x = \underbrace{1 - \frac{x^2}{2!}}_{P_2(x)} + \underbrace{\frac{x^4}{4!} + \cdots}_{R_2(x)}$$

2nd order Taylor polynomial near $x = 0$

$$1 - \frac{(x-0)^2}{2}$$

Example: Find the maximum error if $p_2(x) = 1 - \frac{x^2}{2}$ is used to estimate the value of $cos(x)$ at $x = 0.3$. Verify that the error estimate in MATLAB is less than the maximum error.

$$R_2(x) = \frac{f^{n+1}(c)}{(n+1)!}(x-a)^{n+1}$$

$f(x) = \cos x$

$f'(x) = -\sin x$

$f''(x) = -\cos x$

$f'''(x) = \sin x$

$$= \frac{f^{3}(c)\, x^3}{3!} = \frac{\sin c \cdot x^3}{6}$$

Error estimation (contd.)

$-1 \le \sin c \le 1$

$|\sin c| \le 1$

$M = 1$

$$|R_2(x)| = \frac{|\sin c|\,|x|^3}{6} \le \frac{|x|^3}{6} = \frac{(0.3)^3}{6} = \frac{0.027}{6} = 0.0045$$

$\downarrow$

max error

$0 < c < 0.3$

$\sin c < \sin(0.3)$

Demo

$\sin(0.3)$

Applications of Taylor series

Some applications:

- Small-angle approximations e.g. $\sin\theta \approx \theta$ for small values of θ.
- Finding non-elementary definite integrals e.g. $\int_1^2 \frac{sinx}{x} dx$
- Deriving formulas for numerical differentiation and integration (we'll study later in this course)

End of Lecture 15

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 16
Root finding

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to

- understand root-finding methods
- implement these methods in MATLAB

Introduction

Finding roots of functions is something we've been doing since school.

A root of a function $f(x)$ is the value of x for which $f(x) = 0$.

zero

Example: The roots of $f(x) = x^2 - 3x + 2$ are 1 and 2.

$f(1) = 1^2 - 3 + 2 = 3 - 3 = 0$

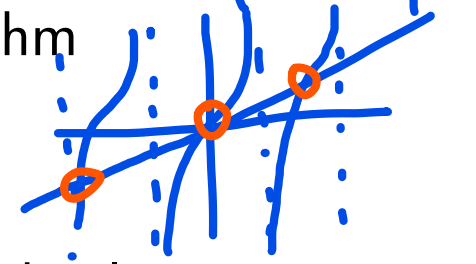
$f(2) = 0$

Introduction (contd.)

There are many ways to find the roots of a function:

- Try a guess value! → not a good idea because so many guess values
- Use a mathematical formula → e.g. quadratic formula
- Sketch the graph of the function
- Apply a root-finding method/algorithm

$\tan x = x$



The last approach is what we'll study in this lecture.

Introduction (contd.)

Sometimes, finding roots is not that easy! For example, how can we find roots of transcendental equations?

$e^0 - 0 = 1$

$e^x - x = 0$

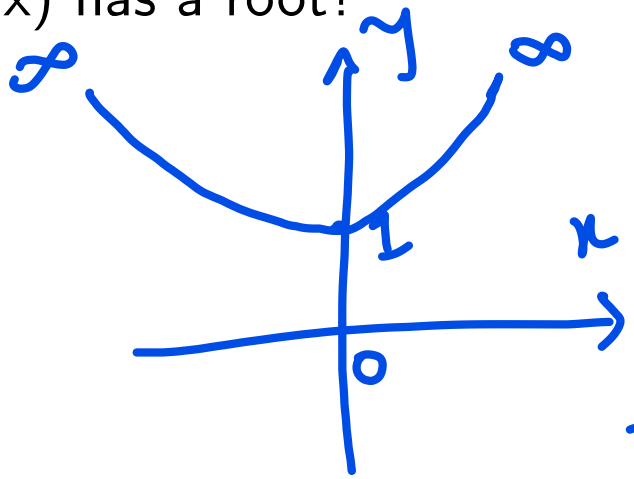
$\Rightarrow y = e^x \;\&\; y = x$

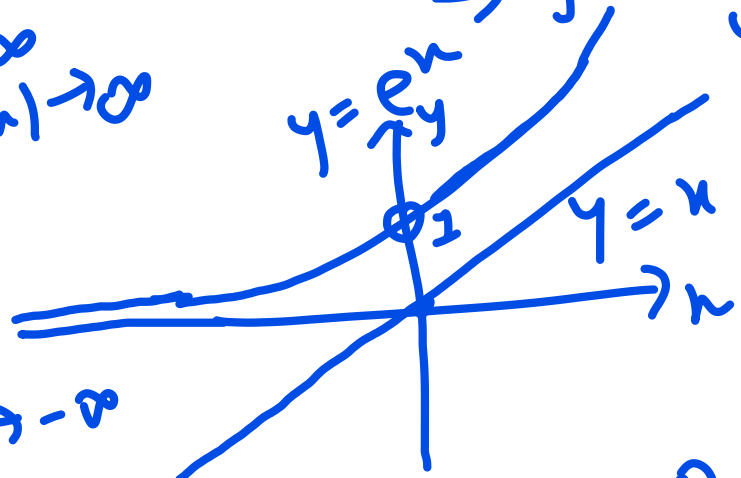
Consider the function $f(x) = e^x - x$.

$x \to \infty$

$f(x) \to \infty$

Does f(x) has a root?





$x \to -\infty$

$f'(x) = e^x - 1 = 0 \Rightarrow x = 0$

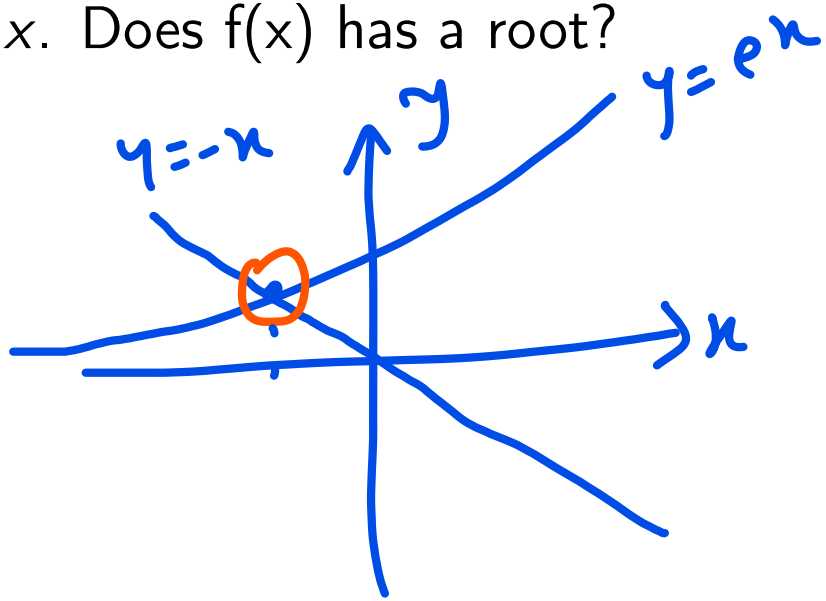
$f''(x) = e^x \;\&\; f''(0) = 1 > 0$

Introduction (contd.)

Now consider the function $f(x) = e^x + x$. Does f(x) has a root?

$$e^x + x = 0$$

$$e^x = -x$$



Introduction (contd.)

Some root-finding methods:

- Bisection Method
- Newton-Raphson Method → same as Newton's method

Activity

Let f(x) be a function defined in the interval [a,b]. Suppose f(a)f(b)<0. Choose the correct statement.

f(a) & f(b) are of different signs

1. f(x) always has a root in (a,b).
2. f(x) has exactly one root in (a,b) if f is continuous on [a,b].
3. f(x) has at least one root in (a,b) if f is continuous on [a,b].
4. f(x) doesn't have a root in (a,b).

To answer, please go to the mentimeter link provided in the chat.

Bisection Method

Steps:

$[a, b]$ such that $r \in [a, b]$

1. Choose a guess interval which may contain the root.
2. Approximate the root by the mid-point of this interval.

   $a \quad m \quad b$

   $m = \frac{a+b}{2}$ (a, m) or (m, b)

3. Bisect the subsequent sub-intervals containing the root until the error is less than tolerance.

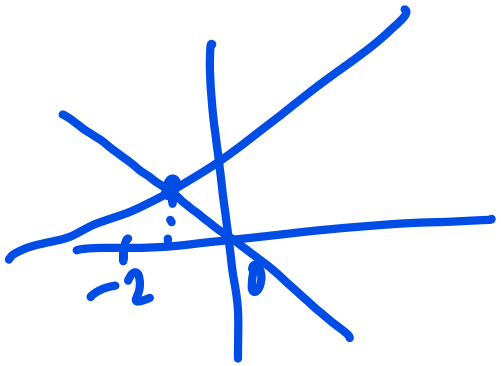
   tolerance is upper bound on your error.

4. The root is given by the midpoint of the last bisected interval.

$|f(x_i)| < tol$

$tol = 0.1, 0.01, 0.001$ $\quad f(x_i) \cong 0$

Bisection Method (contd.)



Consider again the function $f(x) = e^x + x$. Find its root using bisection method.

$a = -2, \; b = 0$

Bisection Method (contd.)

Write a function in MATLAB which takes the values of end points a and b of the guess interval, the function f, the tolerance tol and outputs the root of function f.

Homework: write bisection method function using while loop without recursion.

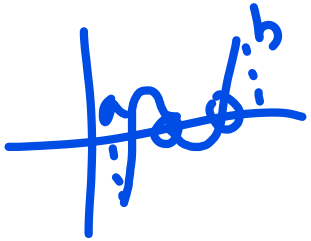
Bisection Method (contd.)

Demo

Bisection Method (contd.)

Limitations of bisection method:

- Requires knowledge of interval containing the root
- Cannot detect multiple roots → because (a) starting condition is $f(a) \cdot f(b) < 0$ (b) can converge to any root
- Fails if the function is discontinuous on the interval [a,b]



Newton-Raphson Method

Isaac Newton

Joseph Raphson

Simpson → extended this method for non-linear eq^ns

Steps:

1. Choose a guess value x_0 for the root.
2. Find a linear approximation of f(x) around $x = x_0$.

   → 1st order Taylor polynomial

   $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$ → straight line

3. Find the root (say x_1) of this linear approximation. The obtained x_1 is an improvement on the guess value x_0.

   $f(x) = f(x_0)$

4. Repeat the steps 2 and 3 to find improved guess values x_i $(i = 2, 3, ..., n)$ until the error is less than tolerance.

Newton-Raphson Method (contd.)

The iterative formula for Newton-Raphson method is derived as

$$x_i = x_{i-1} - \frac{f(x_i)}{f'(x_{i-1})} \tag{1}$$

Newton-Raphson Method (contd.)

Find the root of the function $f(x) = e^x + x$ using Newton-Raphson method.

Newton-Raphson Method (contd.)

Write a function in MATLAB which takes the guess value x0, the function f and its derivative df, the tolerance tol and outputs the root of function f.

Demo

Newton-Raphson Method (contd.)

Limitations of Newton-Raphson method:

- Fails if $f'(x_i) = 0$ for some x_i.
  Example: $f(x) = x^3 - x^2 - x - 1$ with $x_0 = 1$.

- Fails if $f'(x_i)$ gets closer to zero for successive x_i values.
  Example: Consider $f(x) = \frac{1}{x} - e^x$ for $x_0 < 0$.

- In case of multiple roots, a guess values may converge to a different root than the one which is required.
  Example: $f(x) = \tan^{-1} x - x^2$ has two roots but for $x_0 < 0$, the method only gives the root $x = 0$.

End of Lecture 16

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 17
Numerical Differentiation

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

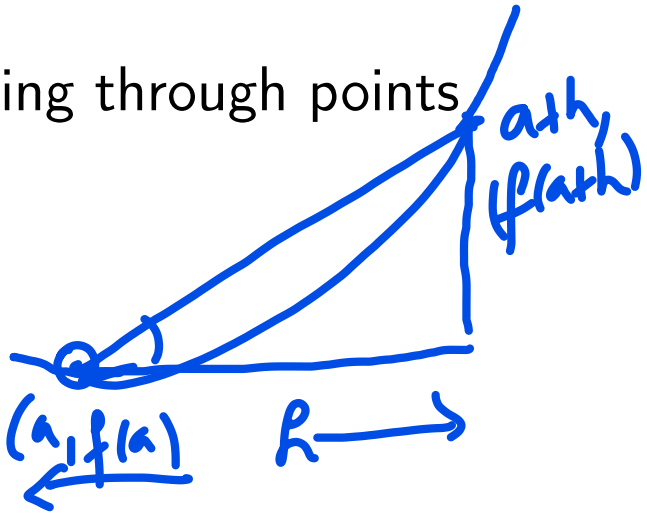
At the end of lecture, students will be able to

- finite difference schemes for derivatives
- understand Taylor series approximations for derivatives
- use MATLAB to find derivatives numerically

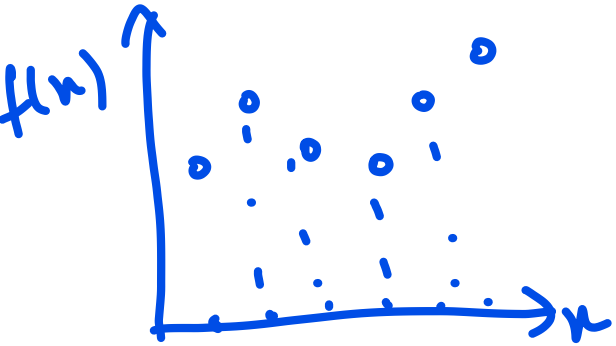
Introduction

For a function $f(x)$, the slope of a secant line passing through points $(a, f(a))$ and $(a+h, f(a+h))$ is

$$\underset{\text{of secant}}{slope} = \frac{f(a+h) - f(a)}{h}.$$



When $h \to 0$, this slope becomes the derivative of a function $f(x)$ at $x = a$:



$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}. \tag{1}$$

Introduction (contd.)

$f(x) = \sqrt{\sin x} \times \log x \times e^{-x^2}$

$f'(x)$ vs x

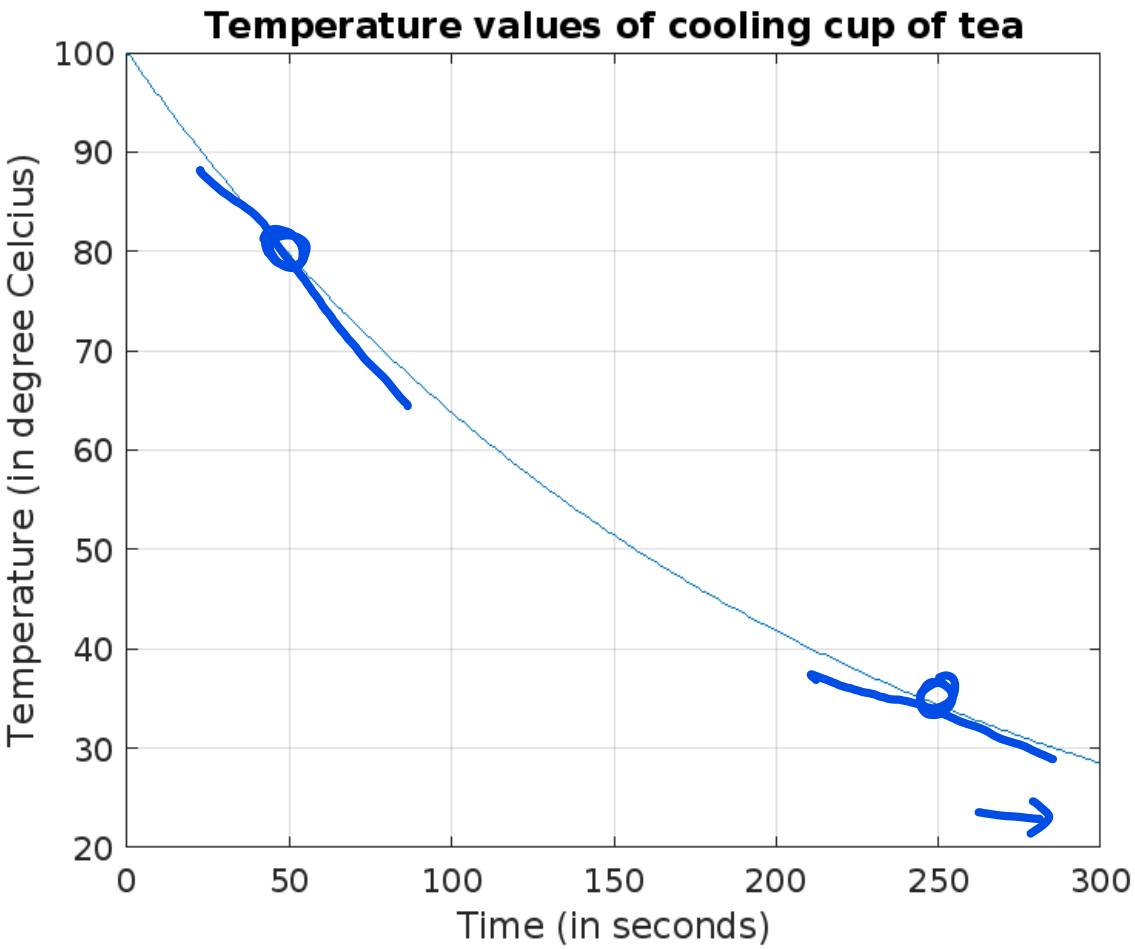
Points to note ☛

- (1) is helpful if analytical form of $f(x)$ is known explicitly.
- Even if $f(x)$ is known, sometimes analytical form of f'(x) can be too complicated.

Introduction (contd.)

Example: Suppose you go outside with a cup of tea heated at 100 °C. The outside temperature is 8 °C. What is the rate of cooling at any time instant?

rate of change of temp. with time $= \frac{dT}{dt}$



Finite-difference schemes

- The domain of a function $f(x)$ can be represented by a **numerical grid** which contains points x_i evenly spaced by fixed distance called **spacing** or **step size**.

- A **finite difference** is the difference of values of function $f(x)$ at two grid points. MATLAB's diff() operator finds the finite differences $f(x_{i+1}) - f(x_i)$.

or method

- A **finite-difference scheme** for derivative provides a formula for estimating derivative of a function on the numerical grid.

Finite-difference schemes (contd.)

Some finite difference schemes:

$\frac{1}{2}(a+b)$

- Forward difference

$$f'(a) \approx \frac{f(a+h) - f(a)}{h}. \quad \to (a) \tag{2}$$

- Backward difference

$$f'(a) \approx \frac{f(a) - f(a-h)}{h}. \quad \to (b) \tag{3}$$

- Central difference

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}. \tag{4}$$

Finite-difference schemes (contd.)

Example: Write a script file which uses finite difference schemes to plot the temperature gradient curve for the cooling cup of tea example.

Demo

Finite-difference schemes (contd.)

The analytical form of $T(t)$ comes from Physics. For our problem, it is taken as

$$T = 8 + 92e^{-0.005t} \tag{5}$$

Newton's law of cooling

The exact solution for temperature gradient is given by

$$\frac{dT}{dt} = -0.46e^{-0.005t} \tag{6}$$

$$T = \overset{8}{\overset{\shortparallel}{T_a}} + \underbrace{(T_0 - T_a)}_{(100-8)} e^{-\gamma t}$$

heat transfer coefficient

Example: Write a script file to plot the exact and approximate temperature gradient of cup of tea example.

Demo

Taylor series approximations of derivatives

The finite difference schemes discussed before can also be derived using Taylor series. Consider the Taylor series of a function $f(x)$ at $x = a$:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots \tag{7}$$

which for the point $x = a + h$ gives

$$f(a+h) = f(a) + \frac{f'(a)}{1!}h + \frac{f''(a)}{2!}h^2 + \cdots . \tag{8}$$

From (8), we have

$$f'(a) = \frac{f(a+h) - f(a)}{h} - \frac{f''(a)}{2!}h - \frac{f'''(a)}{3!}h^2 + \cdots . \tag{9}$$

Taylor series approximations of derivatives (contd.)

For very small h, (9) gives the approximation

$$f'(a) \approx \frac{f(a+h) - f(a)}{h} \tag{10}$$

Exercise: Derive backward and central difference schemes for $f'(a)$ using Taylor series of $f(x)$.

The Big O notation

Consider equation (10) again.

$$f'(a) = \frac{f(a+h) - f(a)}{h} - \frac{f''(a)}{2!}h - \frac{f'''(a)}{3!}h^2 + \cdots . \tag{11}$$

This can be compactly written as

order

$$f'(a) = \frac{f(a+h) - f(a)}{h} + O(h). \tag{12}$$

Big 'O' notation

Let's now study what $O(h)$ means.

The Big O notation (contd.)

Definition

For two functions $\phi(x)$ and $\psi(x)$, ϕ is of order of ψ as $x \to x_0$

$$\phi(x) = O(\psi(x)) \quad as \quad x \to x_0 \tag{13}$$

x_0 can be finite, ∞, $-\infty$

if

$$\lim_{x \to x_0} \frac{\phi(x)}{\psi(x)} = C \tag{14}$$

where C is a finite constant.

The Big O notation (contd.)

Let $\phi(h) = -\frac{f''(a)}{2!}h - \frac{f'''(a)}{3!}h^2 + \cdots$.

Then

$$\lim_{h \to 0} \frac{\phi(h)}{h} = -\frac{f''(a)}{2!} = C(say) \tag{15}$$

which means

$$\phi(h) = O(h) \quad \text{as} \quad h \to 0 \tag{16}$$

Thus, we say that forward difference scheme (12) is $O(h)$.

Order of accuracy

For a $O(h^p)$ finite difference scheme, p is called the **order of accuracy**.

Example: The forward difference scheme (12) is first order accurate.

Exercise: Show that the central difference scheme for $f'(a)$ can be written as

$$f'(a) = \frac{f(a+h) - f(a-h)}{2h} + O(h^2) \tag{17}$$

and therefore is second order accurate.

Higher order derivatives

We can again use Taylor series to approximate higher order derivatives of $f(x)$.

Example: Find finite-difference scheme for $f''(a)$.

For points $x = a + h$ and $x = a - h$ from (8) we have

$$f(a+h) = f(a) + \frac{f'(a)}{1!}h + \frac{f''(a)}{2!}h^2 + \cdots . \tag{18}$$

$$f(a-h) = f(a) - \frac{f'(a)}{1!}h + \frac{f''(a)}{2!}h^2 + \cdots . \tag{19}$$

$$f(a+h) + f(a-h) - 2f(a) = f''(a)h^2 + \cdots$$

Higher order derivatives (contd.)

Adding equations (18) and (19) and solving for $f''(a)$ gives

$$f''(a) \approx \frac{f(a+h) - 2f(a) + f(a-h)}{h^2} \tag{20}$$

$O(h)$?

$O(h^2)$?

$O(h^4)$?

$f(a-h)$ $f(a)$ $f(a+h)$

$f'''(a) = ?$

$f''''(a) = ?$

End of Lecture 17

Please provide your feedback here

MA2252 Introduction to Computing

Lecture 18
Numerical Integration

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

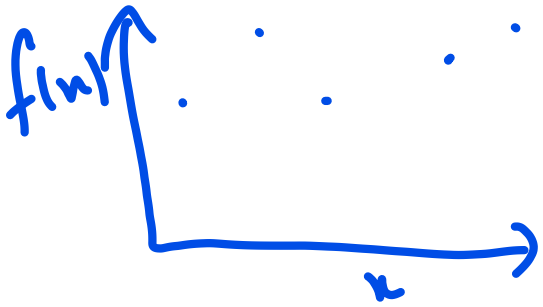
At the end of lecture, students will be able to

- apply numerical methods to evaluate integrals
- understand geometrical interpretation of these methods
- implement these methods in MATLAB
- use MATLAB built-in integration functions

Introduction

Why study numerical integration?

- The anti-derivatives of many functions cannot be represented in terms of elementary functions. **Examples:** $\frac{\sin x}{x}$, e^{-x^2} and $\frac{1}{\ln x}$
- Analytical form of the integrand function(say $f(x)$) may be unknown. **Example:** The values of $f(x)$ are only known at a set of data points x_i.



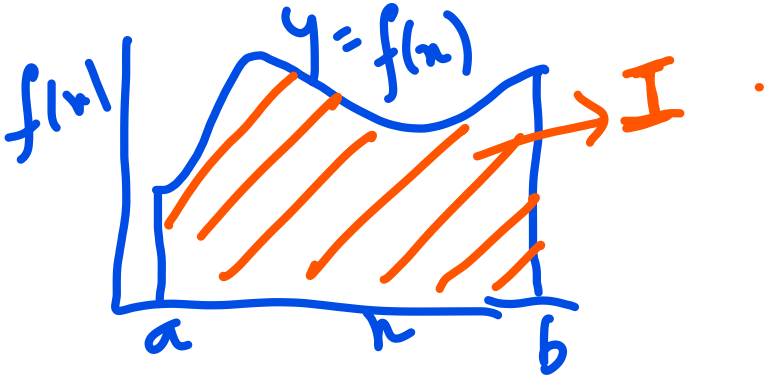
Problem statement

Consider a function $f(x)$ defined over a interval $[a, b]$. We want to evaluate

$$I = \int_a^b f(x)dx. \tag{1}$$

definite integral

This integral can be geometrically seen as area under the curve $y = f(x)$ for $x \in [a, b]$.



Problem statement (contd.)

Steps to evaluate (1) numerically:

- Create a numerical grid $x_i \quad (i = 0, 1, 2, \cdots, n)$ such that $x_0 = a, x_n = b$ and $x_{i+1} - x_i = h(\text{say})$.
- Using some appropriate method, calculate the area A_i under $f(x)$ for each sub-interval $[x_i, x_{i+1}] \quad (i = 0, 1, 2, \cdots, n-1)$.
- Compute the sum of the areas A_i over the interval $[a, b]$ i.e.

$$I \approx \sum_{i=0}^{n-1} A_i \tag{2}$$

→ sum the areas of the A_i strips

Numerical integration methods

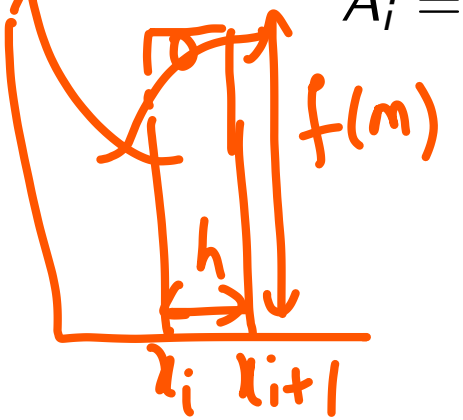
- Midpoint rule
- Trapezoidal rule
- Simpson's rule

Midpoint rule

Steps:

- The value of function in a subinterval $[x_i, x_{i+1}]$ is interpolated by a constant function with the value $f(\frac{x_i+x_{i+1}}{2})$.
- The area A_i is calculated by area of rectangle under the constant function.

$A_i = f(m) \times h$



$$A_i = h * f(\frac{x_i + x_{i+1}}{2}) \tag{3}$$

$$I \approx \sum_{i=0}^{n-1} A_i$$

$$m = \frac{x_i + x_{i+1}}{2}$$

Midpoint rule (contd.)

Example: Write a script file which uses Midpoint rule to approximate $\int_0^{\pi} \sin x \, dx$.

$$\int_0^{\pi} \sin x \, dx = 2$$

Midpoint rule (contd.)

Demo

Trapezoidal rule

trapezoid = trapezium

Steps:

- Here, the function in the subinterval $[x_i, x_{i+1}]$ is approximated using a straight line joining points $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$ (linear interpolation).

- The area A_i is calculated by the area of trapezium formed under this straight line.

$$A_i = \frac{1}{2}(f(x_i) + f(x_{i+1}))h \tag{4}$$

Trapezoidal rule (contd.)

Example: Write a script file which uses Trapezoidal rule to approximate $\int_0^{\pi} \sin x \, dx$.

Trapezoidal rule (contd.)

Demo

Simpson's rule

Steps:

- Here, the function $f(x)$ is approximated on two subintervals $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$ taken together. The interpolating function is a quadratic passing through points $(x_{i-1}, f(x_{i-1}))$, $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$.
- The area B_i over interval $[x_{i-1}, x_{i+1}]$ is derived as

$$B_i = \frac{h}{3}(f(x_{i-1}) + 4f(x_i) + f(x_{i+1})) \tag{5}$$

- The integral I is given by

$$I \approx \sum_{i=1, i=odd}^{n-1} B_i \tag{6}$$

Simpson's rule (contd.)

(6) can also be expressed in the form:

weights

$$I \approx \frac{h}{3}\left[f(x_0) + 4\left(\sum_{i=1, i=odd}^{n-1} f(x_i)\right) + 2\left(\sum_{i=2, i=even}^{n-2} f(x_i)\right) + f(x_n)\right] \quad (7)$$

Note: Since B_i is calculated for two consecutive subintervals taken together, Simpson's rule requires even number of subintervals i.e. n should be even.

Simpson's rule (contd.)

Example: Write a script file which uses Simpson's rule to approximate $\int_0^{\pi} \sin x \, dx$.

Simpson's rule (contd.)

Demo

MATLAB's built-in integration functions

Two useful functions are trapz() and integral().

- trapz(x,f) takes of numerical grid x and function f as vector arguments and computes the value of integral I using trapezoidal rule.
- integral(fun,xmin,xmax) integrates the function fun from lower limit xmin to upper limit xmax.

MATLAB's built-in integration functions (contd.)

Write a script file using trapz() and integral() functions to approximate $\int_0^{\pi} \sin x \, dx$.

MATLAB's built-in integration functions (contd.)

Demo

End of Lecture 18

Please provide your feedback here