People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

UNIVERSITY ECHAHIDE HAMMA LAKHDAR-EL-OUED

FACULTY OF EXACT SCIENCES

Master's Thesis

ACADEMIC MASTER

Domain : Mathematics and Computer Science
Major : Mathematics
Speciality :Fundamental and Application Mathematics

Project

Solving Ordinary Differential Equations with Matlab

Presented by :Taiar Raihana and Moussaoui Hanadi
Under the supervision of: Mohammed Moumen Bekkouche

Supported before the jury:

M:Nadjet Doudi MA(A)	University El-Oued	President
M:Mohmmed Moumen Bekkouche.MC(A)	University El-Oued	Rapporteur
M:Mohammed Beggas. MA(A)	University El-Oued	Examiner

Academic Year : 2023 – 2024

الشكر والتقدير

قال تعالى :﴿ و لَئِنْ شَكَرْتُمْ لَأَزِيدَنَّكُمْ ﴾

سبحانك اللهم لا علم لنا إلا ما علمتنا , نشكر الله و نحمده فضل نعمه علينا, والصلاة و السلام على قدوة المربين نبينا محمد وعلى آله وصحبه أجمعين .

أتقدم بأسمى عبارات الشكر و التقدير الى كل من علمني علمنا ينتفع به و أدب يرتفع به.

بداء من معلمي الإبتدائي وصولا إلى أساتذة التعليم العالي و البحث العلمي في قسم الرياضيات التطبيقية بجامعة الشهيد حمى لخضر.

أما بعد يشرفنا أن نتقدم بالشكر الجزيل و العرفان إلى الأستاذ المشرف " الدكتور محمد مومن بكوش" الذي أفادني بنصائحه و توجيهاته طيلة إنجاز هذه المذكرة.

كما أشكر أعضاء لجنة المناقشة التي شرفتني بقبولها مناقشة مذكرتي ,اللذين لاشك أنهما سيفيضون عليا بتوجيهاتهما القيمة و ملاحظاتهما السديدة.

نتقدم بخالص الشكر والتقدير لكل من ساهم وبذل جهدًا لمساعدتنا، ونؤكد أن عطائكم وتفانيكم لن يمحى من ذاكرتنا. سنظل ممتنين إلى الأبد للبروفيسور "عبد اللطيف بالطيب" والأستاذة "لواتي فيروز" على مساهماتهما وجهودهما القيمة.

و في الختام، أعرب عن خالص شكري وامتناني لكل من قدم لي يد العون والمساعدة، سواء كانوا قريبين أو بعيدين. إن كلمة طيبة أو توجيه بسيط، حتى لو كانت دعوة صادقة في ظهر الغيب، تعني الكثير بالنسبة لي. لذا، أود أن أعبر عن جزيل شكري وامتناني لهم جميعًا على دعمهم الكبير وتشجيعهم المستمر.

ولكم مني فائق التقدير والاحترام.

إهداء

الحمد الله حبا و شكرا و امتنانا على البدء و الختام وَ آخِرُ دَعْوَاهُم أَنِ اَلْحَمْدُ لِلّٰهِ رَبّ اَلْعَالَمِين,

إلى من بلغ الرسالة و أدى الأمانة ..ونصح الأمة .. إلى نبي الرحمة و نور العالمين سيدنا محمد صلى الله عليه وسلم

إذا كان الإهداء يعبر ولو بجزء من الوفاء، فأهدي ثمرة جهدي :

الى من هو جزء من القلب و الفؤاد إلى أجمل وأروع الإنسان إلى قدوتي وخير مثال إلى من أحمل إسمه بكل فخر وعزة وشرف الى أبي.... ،اللهم إن ابي بين يديك فأرحمه واغفر له وبرد على قبره واجعله من الضاحكين المستبشرين , اللهم بقدر شوقي له ارحمه واغفر له واجمعني به في اعلى مراتب جنتك "محمد الطيب موساوي رحمه الله "

الى مصدر الأمان الذي استمد منه قوتي الى نور عيني و حظي الجيد و فوزي وفخري , الى من ابصرت بها طريق حياتي واعتزازي بذاتي الى القلب الحنون الى من كانت دعواتها تحيطني أمي الغالية " حياة ديدة"

إلى السند و الكتف الذي استند عليه دائما إخوتي " سعيد ,عبد الباسط , موسى ,رضا ,سفيان,ياسين " دمتم لي فخرا

الى الذين غمروني بالحب و أمدوني دائما بالقوة وكانوا موضع الأتكاء في كل عثراتي أخواتي " أمال ,إنتصار "

الى من قطعت معها هذا الدرب و شاركت معها كل لحظات , الى شقيقة الروح التى لم تلدها أمي بل ولدتها مواقف الحياة " ريحانة طيار"

الى ملائكة التي رزقني الله بهن لإعرف من خلالهن طعم الحياة بنات أختي و بنات أخي " يسرى , منال ,أسماء , ملاك ,إيثار "

الى من تحلو بالإخاء و تميزوا بالوفاء و العطاء و سعدت برفقتهم في دروب الحياة صديقاتي " دلال , عبير , كوثر , خيرة "

الى كل من كان لهم اثر جميل في حياتي و من يفرحون لنجاحي و كأنه نجاحهم....

★ ★ ★

" موساوي هنادي "

إهداء

الحمد الله حبا و شكرا و امتنانا على البدء و الختام وَ آخِرُ دَعْوَاهُم أَنِ اَلْحَمْدُ لِلّٰهِ رَبِّ اَلْعَالَمِين,

مما زادني فخرا وتيها...وكدت بأخمصي أطأ الثريا...دخولي تحت قولك ياعبادي...وآن صيرت أحمد لي نبيا...

إلهي لايطيب الليل إلا بقربك ولا يطيب النهار إلا بطاعتك ولاتطيب اللحظات السعيدة إلا بذكرك الله عز وجل

إلى من بلغ الرسالة وأدى الأمانة إلى صاحب الأثر العظيم في قلبي فدته روحي فما نفس تشابهه"سيدنا محمد صلى الله عليه وسلم"

إلى من ركع العطاء تحت قدميها إلى الغالية التي لانرى الأمل إلا من عينيها إلى من أفضلها على نفسي ولما لا فلقد ضحت من أجلي ولم تدخر جهدا في سبيل إسعادي على الدوام "أمي حبيبتي جميلة سحارة"

إلى اليد الطاهرة التي أزالت من أمامي أشواك الطريق ورسمت المستقبل بخطوط من الأمل والثقة للذي أعتمد عليه في كل صغيرة وكبيرة "والدي العزيز التجاني طيار"

إلى ملاكي الطاهر،وقوتي بعد الله،داعمتي الأولى و الأبدية "جدتي الغالية باهي عائشة"

يقال أن الخال والد أما خالي فلا كلام ينصفه ولا نص يكفي للحديث عنه،لأبطالي الثلاثة "العيد ،محمود ،ياسين"

إلى من هي أقرب إلي من روحي إلى من شاركتني حضن الأم وبها استمد قوتي وإصراري أختي ورفيقة دربي"حلومة"

إلى من كاتفتني ونحن نشق الطريق معا نحو النجاح في مسيرتنا العلمية ،إلى تلك التي كانت ولازالت أختي التي لم تلدها أمي "هنادي موساوي"

إلى اللواتي يفرحهن مايفرحني ويحزنهن مايحزني خالاتي وأقرب خلق الله إلى قلبي "زينب،نجاة،يمينة،سعاد،نورة"

لا أعرف كيف أشكر القدر الذي جمعني بكن ،لا أقول سوى دمتن لي صديقات مدى الحياة "آمنة،وسام، سامية"

★★★

"ريحانةطيار"

Table of contents

Introduction

In our current era, reliant on technology and science, it cannot be denied that ordinary differential equations play a crucial role in understanding natural phenomena and predicting the behaviour of complex systems. From the design of aircraft and spacecraft to the development of medicines and medical treatments, many engineering and scientific solutions rely on a precise understanding of the interactions and phenomena that can be represented by differential equations.

Ordinary differential equations, which are now an integral part of the daily lives of engineers and scientists, have deep roots in the development of mathematics, physics, and engineering. Their establishment is credited to eminent scientists such as Isaac Newton and Joseph-Louis Lagrange, who laid strong mathematical foundations for understanding motion and change in the natural world.

Despite the importance of ordinary differential equations, there are ongoing challenges facing researchers in this field. Among these challenges, the numerical and computational techniques used to solve these equations need continuous development to keep pace with technological advancements and growing scientific needs. Additionally, current challenges in science and engineering require innovative and creative applications of ordinary differential equations, opening the door for researchers to explore new areas of application.

This memorandum aims to provide a comprehensive study of solutions to ordinary differential equations using MATLAB as an effective and reliable computer tool. A variety of scientific problems will be analysed and solved using computational and analytical techniques, with the goal of understanding the phenomena and processes behind them and applying them to practical cases.

The memorandum consists of four chapters, with the first chapter presenting some basic concepts in MATLAB. The second chapter focuses on the concept of initial value problems (IVP) and classifications of ordinary differential equations, along with the

most important numerical methods for solving them using MATLAB. The third chapter addresses the numerical study of solutions to nth-order ordinary differential equations using MATLAB, and finally, the fourth chapter applies the numerical methods studied in the previous chapters to a physical and biological problem.

1 MATLAB basics

Introduction

MATLAB is considered the most popular program in scientific circles, used for a wide range of scientific and engineering problems. It simplifies the modelling and analysis of these problems through easy-to-use programming, with over 200 institutions in the United States alone using it. MATLAB is known for its high-performance programming language, tailored for technical computations, offering capabilities for calculations and visualization without requiring extensive expertise. Its name originates from " MATrix LABoratory", reflecting its focus on matrix operations, while its toolboxes provide specialized functions for various scientific applications, such as signal processing, control systems, simulations, neural networks, numerical analysis, finance, statistics, linear algebra, and optimization. MATLAB also offers a Graphical User Interface (GUI) for intuitive interaction, enhancing its usability as a sophisticated application tool.

1.1 Overview of MATLAB

1.1.1 Components of the MATLAB interfaces

Command Windows:

This screen is used to communicate with the MATLAB program by entering commands into it. The commands and instructions in MATLAB are divided into Bisections, which are: commands, statements and function.

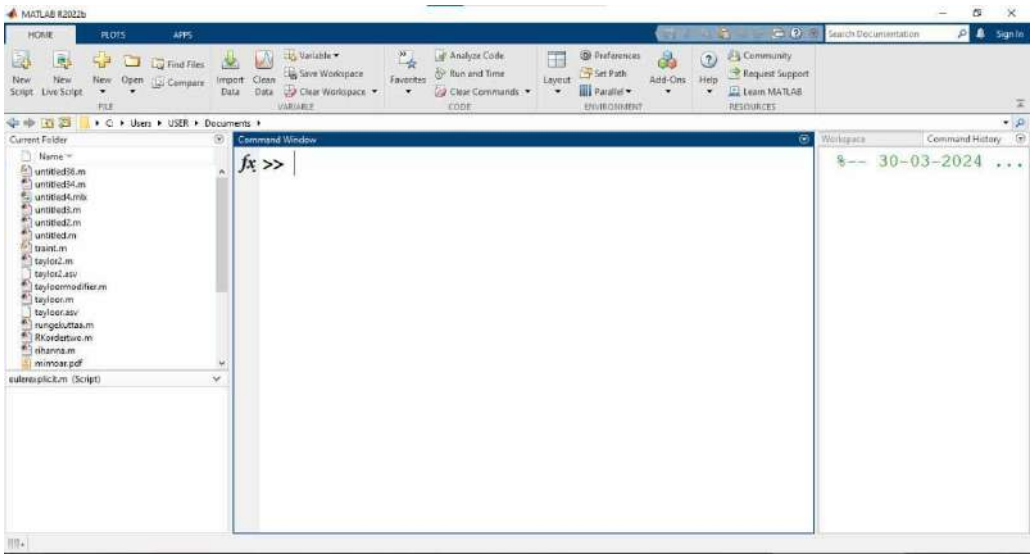
Current Directory (Current Folder):

This window is used to access files. Double-clicking on a file opens it in the editor, which will be discussed later.

Command History:

Through this window, you can view all the commands you've written, with the ability to revisit them by double-clicking.

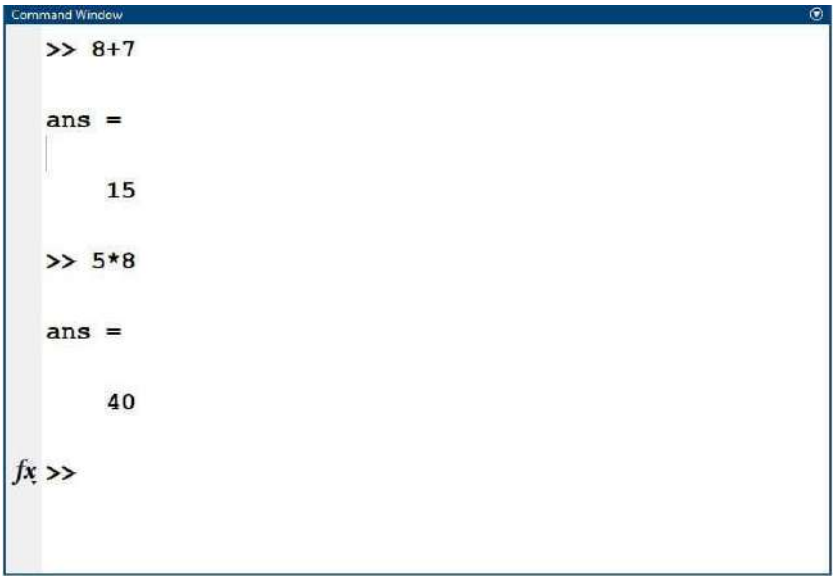
"All these windows are illustrated in the following figure"



1.1.2 Inputting instruction

The process of entering commands into MATLAB is very easy, and we will start by learning how to enter them through the command window. Later, we will become familiar with other methods so that we have comprehensive understanding. To get acquainted with the command window, we will initially use it as a calculator and then gradually we will learn all MATLAB commands.

1. The direct method for arithmetic operations:



2. The method of defining variables:

   Yes, MATLAB allows you to define a variable like X and assign it a specific value, which it will hold throughout the program unless changed. The variable is defined directly.

Example 1.1.

```
X = 11
Y = 8
X + Y
ans = 19
```

Notes on variables:

- *Small don't equal capital.*
- *The variable must start with a letter.*
- *The variable can rave a maximum length of 32 characters.*
- *The variable cannot be a reserved keyword for examples for, while, find …etc.*
- *Variables can be defined in a single line, separated by commas.*

- *There one variables with predefined values, meaning they can be changed best naturally defined in this way.*

  See the table

ans	*A variable always changes, and its value of the last answer that hasn't been defined*
pi	*it's* π
inf	*it means "infinity"*
NaN	*indicates an undefined numerical result*

1.1.3 Arithmetic operations

Here's table showing the expression of arithmetic operations in MATLAB and their precedence:

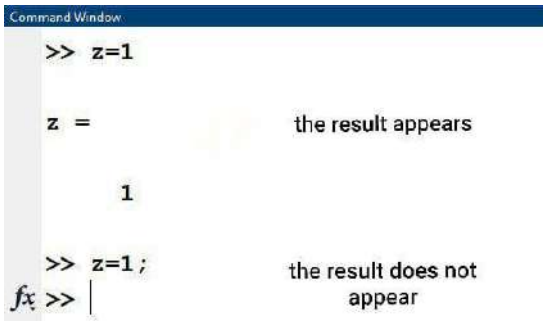
Matlab form	Operation	Symbol
$A \wedge B$	it means power or exponent	$\wedge$
$A * B$	multiplication	$*$
A/B	division	$/$
$A \setminus B = B \setminus A$	inverse division	$\setminus$
$A + B$	addition	$+$
$A - B$	subtraction	$-$

1.2 MATLAB commands and basics

We start with the most important commands, which are considered essential knowledge in the field, and we have abbreviated them as commands for ease and convenience, knowing that some of them are not just commands but rather combinations or phrases.

- The Semicolon:

  Her function is not display the result, see the example for distinction.



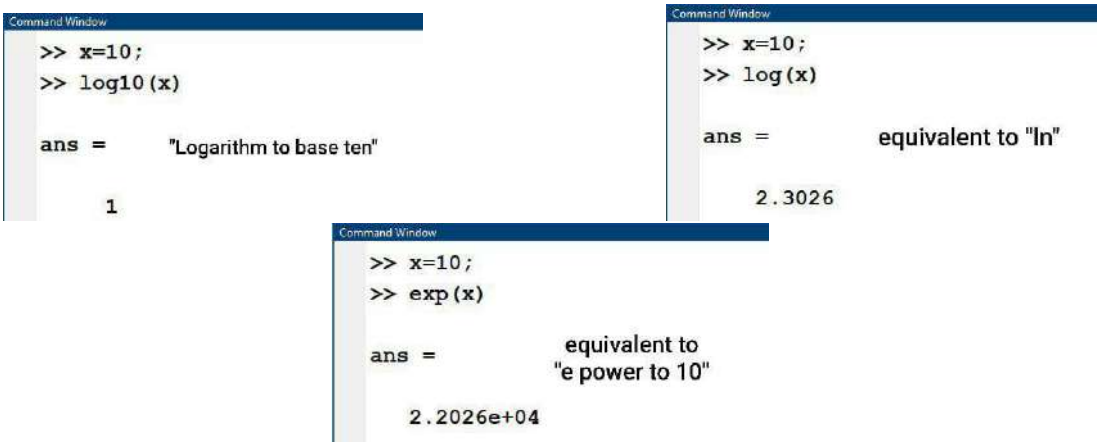
- The power e:

  The symbol "e" represents the tenth power

  Example 1.2. $6 * 10^4 = 6e4$

- log, in, exponential:

  See the example



- Trigonometric operations:

  $\cos x \longrightarrow \cos(x)$

  $\cos^{-1} x \longrightarrow a\cos(x)$

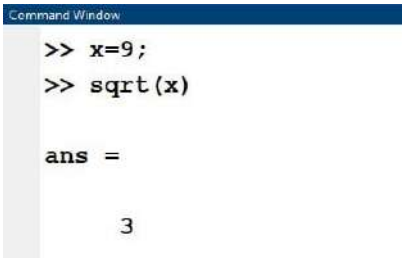
  and you can also write $\sin, \tan, \cosh \ldots$. in the same way

  Remark 1.1. *MATLAB deals with angles using" radian"*

- The command "sqrt":

  $sqrt(x) = \sqrt{x}$

Example 1.3.



- Complex numbers:

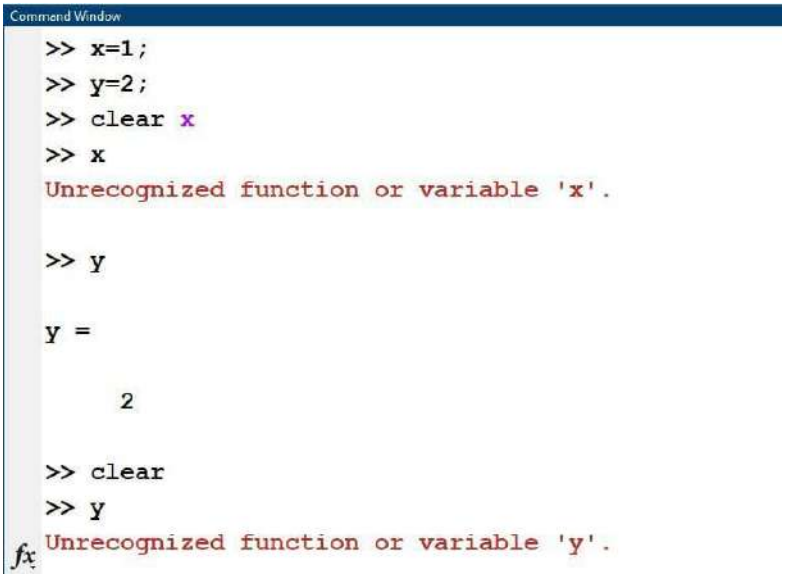
  The know fact is that the imaginary number is the square root of negative one, and it can be expressed in MATLAB by writing the number in this form $Z = X + iy$ or $Z = x + jy$.

  To determine the magnitude, we use the command $abs(z)$ and to determine the angle we use the command $angle(z)$ and $real(z)$ for real number,$imag(z)$ for imaginary number.

- clear and clc:

  - clear: As we previously defined in MATLAB, a variable like X is assigned a value and retains it until it is changed. The "clear" command's function is to delete the contents of a variable and make it undefined. It can be used to clear a specific variable or to clear all variables.

  See the example.



- clc: The purpose of the ”clc” is to clear command window

- The command ”exist”

  you can use the ” exist” function in MATLAB to check if a variable exists. It returns a value of 1 if the variable exists and 0 if it doesn't, here's how you can write it: exist('var')

- The command who, whos:

  The two commands ”who” and ”whos” are used in MATLAB to inquire about the variables currently stored in memory:
  -”who” provides a list of variable names.
  -”whos” provides a detailed table of variables including then sizes and types.

- The command …

  The three dots at the end of a sentence indicate that the sentence will be continued on the next line.

- The command quit:

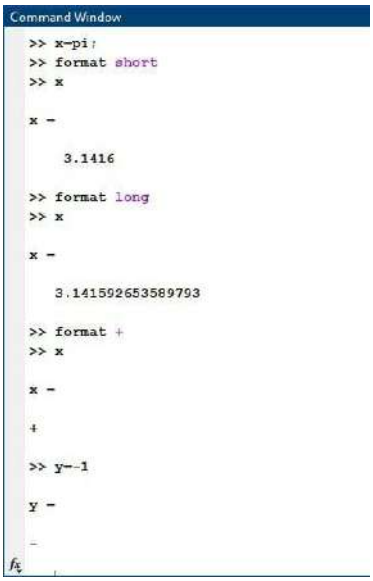
  Executing this command will close MATLAB and terminate the Program

- The commands of ”format”:

  These command's deal with the output formatting of numerical value, and we'll focus on the three most important:

  format short, format long, format+

  See the example to illustrate



- Ceil, floor and round:

  These commands are used for rounding floating -point numbers:
  ”ceil”: Returns the smallest integer greater than or equal to given number
  ”floor”: Returns the largest integer less than or equal to given member.
  ”round”: Rounds a given number to the meanest integer.

- Saving and leading variables:

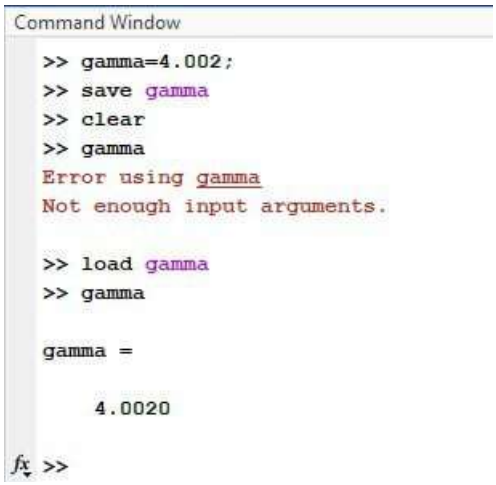
  In MATLAB, you can save a variable to persistently remain at the top-level workspace by following those steps:

- Define your variable.

- Save the variable from the MAT-file.

- Load the variable from the MAT-file when needed.

  See the example

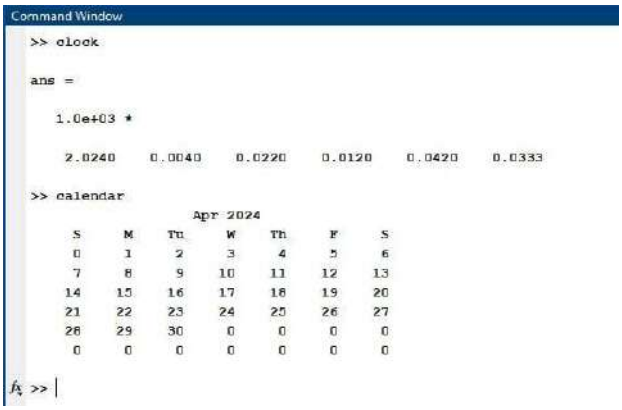


- Tab:

  Finally, the command "tab" will be recognized. Simply, if we have defined a variable, for example: Mathematics = March 14 th

  We can simply tape the first three or four letters and then press the " tab" key on the key board, if there is a variable like this one ,"Mathematics" will appear, and if there are multiple variables that start with the same letters, all similar variables will be displayed and you can choose any one of them by pressing"enter".

- Here are some additional commands:



  And thus, after clarifying the most important commands in MATLAB, any user of MATLAB regardless of their goal, should be familiar with these commands. with this knowledge, we will be able to handle MATLAB applications in the upcoming titles and chapters with ease and flexibility.

1.3 Vectors and matrices

1.3.1 Vectors

A vector is a sequence of elements. When the elements are arranged horizontally, the vector is called a row vector, while when the elements are arranged horizontally, it is called a column vector.

```
>> V = [4, -1, 6, 14]          % create a row vector V
V =
          4          -1          6          14
>> U = [6   -8   -1]            %create a row vector U
U =
          6          -8          -1
>> U = [6  ; -8 ; -1]           %create a column vector U
U =
           6
          -8
          -1
```

- Converting a row to a column vector (transported)

  ```
  >> U =  [6   -8   -1]'
  U =
             6
            -8
            -1
  ```

  If the elements of a vector are arranged in consecutive values but with a specific interval, it is written in the following forms:

$$X = \text{first-element} : \text{last-element}$$

```
>> X = 1:6          %Brackets are optional in this case (1,6)
```

```
X =
     1   2   3   4   5   6
```

if the elements of a vector are arranged in consecutive values but with a specific interval, it is written in the following forms:

$$X = \text{first-element} : \text{the-steps} : \text{last-element}$$

```
>> X = [0: 2: 10]
X =
     0   2   4   6   8   10
X = 0:0.2:1      %Brackets are optional in this case (0, 0.2, 1)
X =
     0   0.2000   0.4000   0.6000   0.8000   1.0000
```

Call an element or sequence of a vector:

The access to elements of a vector is done using the following general syntax:

$$\text{Vector-name (positions)}$$

Example 1.4.

```
>> V= [5, 2, -6, 14, 32, 0]
      V= 5   2   -6   14   32   0
>> V(3)                    %The third position
ans=
   -6
>> V(2: 4)          %from second position to fourth
ans=
  2   -6   14
```

Element-by- element operation for vectors:

$>>$ u $=$ $[0\,,\ \ 9\,,\ \ -1]$;

$>>$ v $=$ $[2\,,\ \ -2,\ \ -5]$;

The operation	Indication	Example
+	Addition of vectors	$>> u+3$ $ans =$ 3 12 2 $>> u+v$ $ans =$ 2 7 −6
-	Subtraction of vector	$>> u-1$ $ans =$ −1 8 −2 $>> u-v$ $ans =$ −2 11 4
.*	Element-by- element multiplication	$>> u*2$ $ans =$ 0 18 −2 $>> u.*2$ $ans =$ 0 18 −2 $>> u.*v$ $ans =$ 0 −18 5
./	Dividing element by element	$>> u./5$ $ans =$ 0 1.8000 0.2000 $>> u./v$ $ans =$ 0 −4.5000 0.2000
.^	Element-by- element power	$u.\hat{}2$ $ans =$ 0 81 1 $>> u.\hat{}v$ $ans =$ 0 0.0123 −1.0000

Notes: The writing of an expression such $u\hat{}2$ generates an error because this expression refers to a matrix multiplication ($u * u$ must be rewritten as $u * u'$ or $u' * u$ to be valid).

Vector length

The vector size (and number of segments) can be obtained with the following function length:

```
>> U = [0, -1, 5, 8, -14];
>> length (U)
ans=
        5
```

Special commands for vectors:

There are four famous orders for vectors:

- max(U) determines the largest element in U
- min(U) determines the smallest element in U
- $prod(U)$ calculates the product of the elements of U
- $sum(U)$ calculates total elements U

Example 1.5.

```
>> U = [8, -6, 11, 2, -4];
>> max (U)
ans=
        11
>> min (U)
ans=
        -6
>> prod (U)
ans=
```

```
        4224
>> Sum (U)
ans=
        11
```

1.3.2 Matrices

The matrix is the expression of value or information by a set of columns and rows (two-dimensional).

To express matrix in MATLAB the following must be respected:

- Elements must be placed in square brackets [ ].
- Items in one row are separated by space or comma (,)
- The columns are separated by either enter or semicolon (;)

Example 1.6. *to clarify this, we take the following matrix:*

$$A = \begin{bmatrix} 6 & 4 & -1 \\ 3 & -8 & 14 \\ 15 & 0 & 12 \end{bmatrix}$$

This matrix can be written into MATLAB using one of the following methods:

```
>> A = [6, 4, -1; 3, -8, 14; 15, 0, 12];
>> A = [6  4  -1; 3  -8  14; 15  0  12];
>> A = [6, 4, -1
        3, -8, 14
        15, 0, 12];
>> A= [[6; 3; 15], [4; -8; 0], [-1; 14; 12]];
```

Referencing and access to matrix elements:

Access to matrix elements is done using the following generic wording:

Matrix-Name (row-positions,column-position)

It is worth noting the following possibilities:

- The access to an element in row i and column j is by: $A(i, j)$
- The access to the entire row number i is done by: $A(i, :)$
- The access to the entire column number j is done by: $A(:, j)$

Example 1.7.

```
>> A = [1, 3, 5, 3; 7, -4, 13, -2; 0, 9, 10, -11]
A =
        1     3     5     3
        7    -4    13    -2
        0     9    10   -11
>>A(2,4)          %the element in the second  row at the fourth column
ans =
       -2
>>A(1,:)          %All the elements of the first row
ans =
        1   3   5   6
>>A(:,2)           %All the elements of the second column
ans =
        3
       -4
        9
>>A(:,3) = [ ]  %Delete third column
A =
        1     3     6
        7    -4    -2
        0     9   -11
>>A(2,:) = [ ]  %Delete second column
A =
      1     3     6
      0     9   -11
>>A = [A, [0; 0]] %Add a new column with A(:, 4)=[0; 0]
```

```
A =
     1    3    6    0
     0    9  -11    0
>> d =size(A)
d =
     3    4
>> d1 = size (A, 1)
d1 =
     3
>> d2 = size (A, 2)
d2 =
     4
```

Automatic generation of matrices:

In MATLAB, there are functions that automatically generate particular matrices, In the following table we present the most used:

Function	Indication.
zeros(n)	Generates matrix $n \times n$ with all elements= 0
zeros(m,n)	Generates matrix $m \times n$ with all elements = 0
ones(n)	Generates matrix $n \times n$ with all elements = 1
ones(m, n)	Generates matrix $m \times n$ with all elements = 1
eye (n)	Generates an identity matrix of dimension $n \times n$
magic(n)	Generates a magic matrix of dimension $n \times n$
rand(m, n)	Generates a matrix of dimension $m \times n$ of random values

Basic matrix operations:

practical	Indication
$+$	addition
$-$	subtraction
$\cdot*$	the multiplication element by element
$\cdot/$	the division element by element
$\cdot\backslash$	the inverse division element by element
$.\hat{}$	the power element by element
$*$	matrix multiplication
$/$	the matrix division. $(A/B) = (A * B^{-1})$

the element-by-element operations on matrices are the same as those for vectors (the only condition necessary to do an element-by-element operation is that the two matrices have the same dimensions). On the other hand, the multiplication or division of matrices requires some constraints.

Example 1.8.

```
$>> A= ones(3,2)
A =
     1     1
     1     1
     1     1

>> B=zeros(2,3)
B =
     0     0     0
     0     0     0

>> B = B+6
B =
     6     6     6
```

```
        6     6     6
```

```
>> A*B
ans =
        12    12    12
        12    12    12
        12    12    12
```

```
>> B.^2
ans =
        36    36    36
        36    36    36
```

Useful functions for matrix processing:

Some of the most commonly used functions for matrices are:

Function	the usefulness	Example
det	Determinant calculation of a matrix	>> A= [6, 2; −1, 8]; >> det(A) ans = 50
inv	Calculates the inverse of a matrix	>> inv(A) ans = 0.1600 −0.0400 0.0200 0.1200
rank	Calculates the rank of a matrix	>> rank (A) ans = 2
trace	Calculates the trace of a matrix	>> trace(A) ans = 14

eig	calculates the eigenvalues	>> eig(A) ans = 7.0000 + 1.0000i 7.0000 − 1.0000i
dot	Calculates the scalar product of 2 vectors	>> U= [2, 3, −4]; >> V= [1, 5, 7]; >> dot (U, V) ans = −11
norme	Calculates the standard of a Vector	>> norm(U) ans = 5.3852
cross	Calculates the vector product of 2 vectors	>> cross(U, V) ans = 41 −18 7
diag	Returns the diagonal of a matrix	>> diag(A) ans = 6 8
diag(V)	Creates a matrix With Vector V in the diagonal and 0 elsewhere	>> diag(V) ans = 1 0 0 0 5 0 0 0 7

tril	Returns the lower triangular part	>> A= [1, 1, 1; 2, 2, 2; 3, 3, 3]; >> tril(A) ans = 1 0 0 2 2 0 3 3 3 >> tril(A, −1) ans = 0 0 0 2 0 0 3 3 0
triu	Returns the upper triangular part	>> triu(A) ans = 1 1 1 0 2 2 0 0 3 >> triu(A, 1) ans = 0 1 1 0 0 2 0 0 0

1.4 MATLAB sentences, functions and subprograms

1.4.1 Input and output sentences

Input sentences:

To read a value given by the user, it is possible to use the **input** command, which has the following syntax:

```
Variable = input('indicative statement')
```

Example 1.9.

```
>> X= input('Enter X:')
    Enter X: 16
X=
        16
>> A= input ('Enter matrix A:')
     Enter matrix A: [1, 2, 3; 4, 5, 6]
A =
        1     2     3
        4     5     6
```

Output sentences:

The variable value can be displayed using the **disp** function which contains the following formula :

```
disp (object)
```

Example 1.10.

```
>> disp(X)
              16
>> disp(A)
        1     2     3
        4     5     6
```

1.4.2 Conditional sentences

if Statement:

They are characterized by having several forms:

```
if (condition)
        instruction_1
        instruction_2
        ...

        instruction_N
end
```

ou

```
if (condition)
        instruction set_1
else
        instruction set_2
end
```

```
if (condition_1)
        instruction set_1
elseif (condition_2)
        instruction set_2
else
        instruction set if all expressions are wrong
end
```

If the condition is evaluated to true, the instructions between the **if** and **end** will be executed; otherwise, they will not be (or if an else exists, the instructions between the **else** and **end** will be executed). If it is necessary to check multiple conditions instead of one,**elseif** clauses can be used for each new condition, and finally, an **else** can be put in the case where no condition has been evaluated to true.

Example 1.11. *Let's create a program that finds the roots of the quadratic equation indicated by:* $ax^2 + bx + c = 0$

```
1 %the program for solving the equation
2 %a*X^ 2 + b*X + c = 0
3 a= -2;
4 b= 1;
5 c= 3;
6 delta = b^2 - 4*a*c;
7 if(delta < 0)
8         disp('there is no solution')
9         elseif (delta == 0)
```

```
10                  X = b/ (2*a)
11          else
12                  X1 = (-b + sqrt(delta)) / (2*a)
13                  X2 = (-b - sqrt(delta)) / (2*a)
14  end
```

program results

$X1 =$

-1

$X2 =$

1.5000

1.4.3 Sentences of rotation and repetition for statement

For loops re-execute a set of commands a certain number of times and with a certain step, and the general formula of the for loop is given as follows

```
for i = X1: X3 :X2
        (commands)
end
```

Where the commands located between **for** and **end** statements are re-executed form the initial value X_1 to the final value X_2 and with an increase of X_3. as in the following example

Example 1.12.

```
for i=0:4
        A = i*2
end
```

Execution results

```
A =
     0
A =
     2
A =
     4
A =
     6
A =
     8
```

while statements:

while loops perform Calculations an indefinite number of times unlike for loops that perform a Certain number of passes, and the general formula of a **while** loop can be written as follows:

```
while expression

      (Commands)

      end
```

The set of commands located between the **while** and **end** statements will executes as long as all elements within expression have integer values, and the result of expressions is usually an odd number.

Example 1.13.

```
x = 1
while n < 10
   x= x + 1;
   disp(x);
end
```

Execution results:

2

3

1

5

⋮

10

note: If the break instruction is found within an inner loop that is located within larger loops, the program exits from the loop in which it encountered the instruction and does not exit from the larger loops.

1.5 Functions and subpragams

MATLAB contains a large number of predefined functions such as: sin, cos, *sqrt*, *Sum*, . . . *etc.* and it is possible to create our own function by writing their source codes in M-Files (with the same function name) respecting the syntax next

```
function [r1, r2, …, rn] =function_name(arg1, arg2, …, argn)

 r1 = . . . % the returned value for r1
 r2 = . . . % the returned value for r2
 r2 . . .
 rn = . . . % the returned value for rn
end
```

Where: $r_1 \dots r_n$ are the returned values, and $\arg_1 \dots \arg_n$ are the arguments.

Example 1.14. *Write a function that calculates the square root of a number by the method newton*

Solution:

```
1 function r = racine(number)
2     r= number./2;
3     perision = 6;
```

```
4       for i=1:perision
5           r=(r+number./r)/2;
6       end
7   end
```

The execution:

```
>>x =racine(9)
      x =
            3
```

1.6 Differentiation and integration

1.6.1 Symbolism and numerology

The symbolic system treats variables as symbols of numbers, and the numeric system treats variables as actual numbers; they should be given values in order to modify them. For example, if we write the following sentence directly in MATLAB.

$f(x) = x^4 - x$

The program will give error

- if something goes wrong, the solution is?

There is a command in MATLAB through which variables are defined based on symbols, and then the length of the program is treated as a number, and its value is computed when requested. The command is: `syms` Var_1 Var_2

Example 1.15.

```
>> f=x^2-x
        ???Unrecognized function or variable 'x'.
>> syms x
>> f=x^2-x
f =
        x^2 - x
>> x=1
```

```
x =
        1
```

More than one code can also be defined using the command

Example 1.16.

```
>> syms x y z
>> f = x*z-y^3
f =
        x*z- y^3
```

1.6.2 Differential

is used for derivation by the MATLAB command diff. the differential (derivation) in MATLAB is done in more than one way.

The direct method:

The derivation in the direct method is carried out in the following forms `diff(var)`

Example 1.17.

```
>> diff(x^4-x)
ans =
        4*x^3-1
```

Notes

- A citation should be placed when the variable is not encoded.
- Using the eval command to give a value to the derivative.

Example 1.18.

```
>> diff(x^4-x)
ans =
        4*x^3-1
>> x= 0; ans
```

```
ans =
        4*x^3-1
>> eval(ans)
ans =
        -1
```

Semi-direct method:

The same direct method, but to get rid of to the quotation mark, we work on encoding the variable before the derivation prefix.

Example 1.19.

```
>> syms x y
>> f = x^2 - 3*y;
>> diff(f)
ans =
        2*x
>> diff(f,y)    %Derivation relation to y
ans =
        -3
```

Definition of conjugation and then derivation:

Example 1.20.

```
>> syms x y z
>> f = x^2-1/y-sin(z);
>> f=inline(f);
>>f(1,2,(60*pi/180))
ans =
        -0.3660
```

1.6.3 Integration

Integration is like differentiation. there is no difference between them except using the **int** Command instead of the **diff** command.

Example 1.21. *Find the value of the integral* $\frac{1}{x}$ *from* 1 *to* 4

```
>>syms x
>>int(1/x,1,4)
   ans=
          -log(1)+log(4)
```

note: The period is not mandatory

```
>>int(1/x)
   ans=
        log(x)
```

1.7 Graphs and data visualization in MATLAB

Based on the principle that an image is better than a long speech, MATLAB offers a powerful visualization system that allows presentation and graphic display data in a way that is both efficient and easy.

In this part of the course, we will present the basic principles essential for draw curves in MATLAB.

1.7.1 The plot function

The plot function can be used with vectors or matrices. She draws lines by connecting points with coordinates defined in its arguments, and it to several forms:

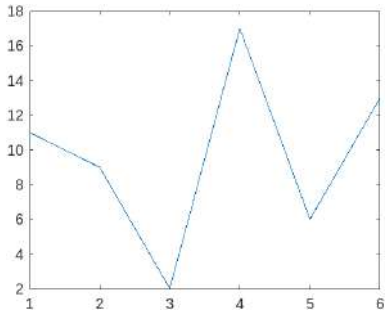
Even it contains two vectors of the same size as arguments:

it considers:

- The values of the first vector as the elements of the X axis (the abscissa).
- The values of the second vector as the elements of the Y axis (the ordinates).

Example 1.22.

```
>> A = [1,2,3,4,5,6];
>> B = [11,9,2,17,6,13];
>> plot(A,B)
```



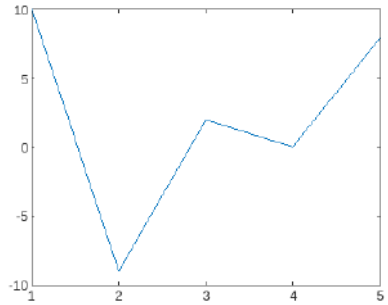
Format(1-1)

If it contains a single vector like argument:

She considers the values of the vector as the elements of the Y axis (the ordinates), and their relative positions will define the X axis (the abscissas).

Example 1.23.

```
>> A = [10, -9, 2, 0, 8];
>> plot(A)
```



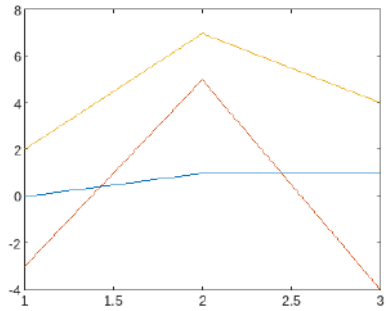
Format(1-2)

If it contains a single matrix as an argument:

It considers the values of each column as the elements of the Y axis, and their relative positions (the line number) as the values of the x axis. So, it will give several curves (one for each column).

Example 1.24.

```
>> M = [0, -3, 2
        1,  5, 7
        1, -4, 4];
>> plot(M)
```



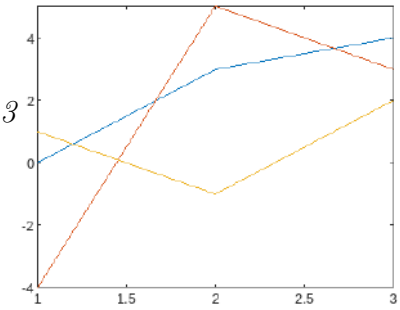
Format(1-3)

If it contains two matrices as argument:

It considers the values of each column of the first matrix as the elements of the X axis, and the values of each column of the second matrix as the values of the Y axis

Example 1.25.

```
>> A = [1, 1, 1; 2, 2, 2; 3, 3,
>> B = [0, -4, 1; 3, 5, -1; 4, 3
>> plot(A,B)
```



Format(1-4)

It is obvious that the more the number of coordinates increases, the more precise the curve becomes.

For example, to draw the curve of the function $y = \cos(x)$ on $[0; 2\pi]$ we can write:

Example 1.26.

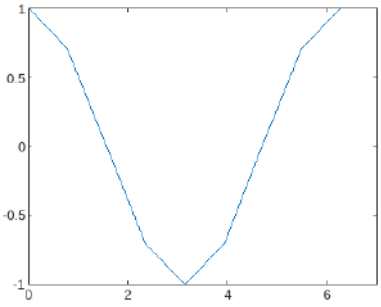
- *The first figure*

```
>> x=0:pi/4:2*pi;
>> y = cos(x);
>> plot(x, y)
```

- *The second figure*

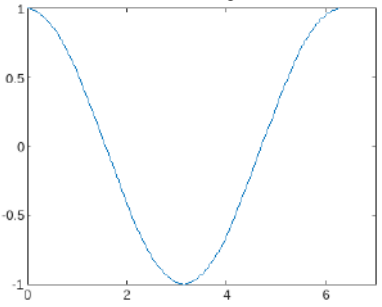
```
>> x=0:pi/16:2*pi;
>> y = cos(x);
>> plot(x, y)
```

not: $\frac{\pi}{4}$



Format(1-5)

not: $\frac{\pi}{16}$



Format(1-6)

1.7.2 Change the appearance of a curve

It is possible to manipulate the appearance of a curve by changing the color of the curve, the shape of the coordinate points, and the type of line connecting the points. To do this, we add a new argument (which we can call a marker) of type string of character to the plot function like this:

`plot(X,Y, 'marker')`

The content of the marker is a combination of a set of special characters collected in the following table:

curve color		Representation of points	
the character	its effect	the character	its effect
b or blue	Curve in blue	·	one point·
g or green	Curve in green	∘	a circle ●
r or red	Curve in red	×	the symbol ×
c ir cyan	between green and blue	+	the symbol +
m or magenta	in bright purplish red	∗	one star∗
y or yellow	curve in yellow	S	a Square ■
K or black	curve in black	d	a rhombus ♦
Style of the Curve		∨	lower triangle ▼
the character	its effect	∧	upper triangle ▲
−	full line	<	left triangle ◀
:	dotted line …	>	right triangle ▶
−.	in point dash −. − . − .	P	Pentagram ⋆
−−	in dash - - - -	h	hexagram ✶

2 Ordinary differential equations

Introdution

A differential equation is an equation involving an unknown function and its derivatives. In this chapter, we start by introducing the solvability concept of initial value problems (IVP) associated with first-order ordinary differential equations (ODE).

Example 2.1.

- $\dfrac{dy}{dx} = 4x - 1$
- $e^y \dfrac{d^2y}{dx^2} + 2\left(\dfrac{dy}{dx}\right)^2 = 0$

Notation 2.1. *The symbols $y', y'', y''', \ldots, y^{(n)}$ are commonly used to denote the first, second, third, fourth, up to $n-$th derivatives of y with respect to the independent variable under consideration. Therefore, y' represents $\dfrac{d^2y}{dx^2}$ if the independent variables is x, but represent $\dfrac{d^2y}{dp^2}$ if the independent variable is p. It's worth nothing that parentheses are used in y^n to distinguish it from the $n-$th power y^n. If the independent variable is time, typically denoted by t, primes are often replaced by dots. Thus, y', y'', and y''' represent $\dfrac{dy}{dt}, \dfrac{d^3y}{dt^3}$, respectively.*

2.1 Initial-value and boundary-value problems

An initial-value problem consists of a differential equation along with initial conditions, all specified at the same point in the independent variable. These initial conditions are subsidiary conditions that determine the behavior of the unknown function and its derivatives at the starting point. On the other hand, if subsidiary conditions

are provided at multiple points in the independent variable, the problem becomes a boundary-value problem, and these conditions are referred to as boundary conditions.

Example 2.2. *The problem* $y''+2y' = e^x; y(\pi) = 1, y'(\pi) = 2$ *is an initial-value problem, because the two subsidiary conditions are both given at* $x = \pi$.
The problem $y'' + 2y' = e^x; y(0) = 1, y(1) = 1$ *is a boundary-value problem,Because the two subsidiary conditions are given at the different values* $x = 0$ *and* $x = 1$.

2.2 First order differential equations

2.2.1 Classification of 1^{st} order differential equation

2.2.1.1 Standard form and differential form

The standard form for a first-order differential equation in the unknown function $y(x)$ by

$$y' = f(x, y) \tag{2.1}$$

Here, the derivative $\dfrac{dy}{dx}$ appears only on the left side of the equation. While not all, many first-order differential equations can be expressed in standard form by algebraically solving for $\dfrac{dy}{dx}$ and setting it equal to the right side of the resulting equation.

The right side, denoted by $f(x, y)$, can always be represented as the quotient of two other functions $M(x, y)$ and $N(x, y)$. In this case, the standard form (2.1) can be rewritten as

$$\frac{dy}{dx} = \frac{M(x, y)}{N(x, y)} \tag{2.2}$$

This expression is equivalent to the differential form of the equation.

2.2.1.2 Linear equation

Consider a first-order linear differential equation in standard form (2.1). If the function $f(x, y)$ can be expressed as

$$f(x, y) = -p(x)y + q(x) \tag{2.3}$$

Where $p(x)$ and $q(x)$ are functions of x, then the differential equation is linear. Linear first-order differential equation can always be expressed as

$$y' + p(x)y = q(x) \tag{2.4}$$

2.2.1.3 Bernoulli equation

A Bernoulli differential equation is an equation of the form

$$\frac{dy}{dx} + p(x)y = q(x)y^n \tag{2.5}$$

Where n denotes a real number. When $n = 1$ or $n = 0$, a Bernoulli equation reduces to a linear equation.

2.2.1.4 Homogeneous equation

A first-order differential equation in standard form (2.1) is considered homogeneous if it satisfies the condition

$$f(tx, ty) = tf(x, y) \tag{2.6}$$

for every real number t. It's important to note that within the broader framework of differential equation, the term "homogeneous" carries a different meaning. However, when discussing first-order differential equations specifically, "homogeneous" refers to the condition mentioned above.

2.2.1.5 Separable equation

Consider a differential equation in differential form (2.2). If $M(x, y) = A(x)$ (a function only of x) and $N(x, y) = B(y)$ (a function only of y), the differential equation is separable, or has its variables separated.

2.2.1.6 Exact equation

A differential equation:

$$M(x, y)dx + N(x, y)dy = 0 \tag{2.7}$$

is considered exact if then exists a function $g(x, y)$ such that:

$$dg(x, y) = M(x, y)dx + N(x, y)dy \tag{2.8}$$

-The test for exactness that if $M(x, y)$ and $N(x, y)$ are continuous an have continuous first partial derivatives on some rectangle of the XY- plane, so $(1, 6, 1)$ is exact if and only if

$$\frac{dM(x, y)}{dy} = \frac{dN(x, y)}{dx} \tag{2.9}$$

Définition 2.1. *.[8] A function $f(x, y)$ is said to satisfy a **Lipschitz condition** in the variable y on a set $D \subset \mathbb{R}^2$ if a constant $L > 0$ exists with:*

$$|f(x, y_1) - f(x, y_2)| \leq L|y_1 - y_2| \tag{2.10}$$

*whenever (x, y_1) and (x, y_2) are in D. The constant L is called a **Lipschitz constant** for f.*

Théorème 2.1. *[8]Suppose that $D = \{(x, y)|a \leq x \leq band - \infty < y < +\infty\}$ and that $f(x, y)$ is continuous on D. If f satisfies a Lipschitz condition on D in the variable y, then the initial-value problem:*

$$y'(x) = f(x, y); \quad a \leq x \leq b, y(a) = \alpha \tag{2.11}$$

has a unique solution $y(x)$ for $a \leq x \leq b$.

Example 2.3. *Use theorem **??** to show that there is a unique solution to the initial-value problem*

$$y' = 1 + x\sin(xy); 0 \leq x \leq 2; y(0) = 0 \tag{2.12}$$

Solutions

When holding x constant and applying the Mean value theorem to the function

$$f(x, y) = 1 + x\sin(xy)$$

We find that for $y_1 < y_2$, there exists a number ϵ in the interval (y_1, y_2) such that:

$$\frac{f(x,y_2)-f(x,y_1)}{y_2-y_1}=\frac{d}{dy}f(x,\epsilon)=x^2\cos(\epsilon x)$$

Therefore

$$|f(x,y_2)-f(x,y_2)|=|y_2-y_1||x^2\cos(\epsilon x)|\leq u|y_2-y_1|$$

Thus, the function of $f(x,y)$ satisfies a Lipschitz condition with respect to the variable y with a Lipschitz constant $L=4$.

Additionally, the function $f(x,s)$ is continuous for $0\leq x\leq 2$ and $-\infty<y<+\infty$. Therefore, by theorem **??**, a unique solution exists for this initial-value problem.

2.2.2 Numerical methods for solving 1^{st} order differential equation with MATLAB

When the term "numerical solution" is mentioned, it immediately brings to mind a vast and complex subject known as "numerical analysis." Therefore, it can be stated that numerical solution is a crucial aspect within the broader domain of numerical analysis. Numerical solution involves finding specific values of the dependent variable at particular points of the independent variable. It necessitates considerations of related subjects such as stability, convergence, and sensitivity analysis of multiple variables, known as sensitivity analysis."we prefer to begin our study of numerical solutions for ordinary differential equations by examining the various methods used in this field.

2.2.2.1 Taylor series methods

The Taylor series method, in fact is a technique for numerically finding the solution to ordinary differential equations (ODEs). It cannot be strictly classified as a purely numerical method as it is more closely related to approximation techniques. It can be considered as an algorithm coupled with numerical methods.

To understand this method , let's assume that we are dealing with an ordinary differential equation in the standard form: $y'=f(x,y)$

The given equation has the following condition:

$$y(x_0) = y_0 \tag{2.13}$$

We observe from the given condition $y(x_0) = y_0$ that the solution to the ordinary differential equation $y' = f(x, y)$ is known at the initial point, denoted as $x = x_0$. The goal is to find the solution to this equation at any other point x. Now, let's recall the Taylor series expansion of a function f(x) around a point of expansion x. The expansion has the following form:

$$f(x) = f(x_0) + \frac{x - x_0}{1!} f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \dots \tag{2.14}$$

Assuming that the difference between two consecutive points is a constant

$$h = x - x_0 \tag{2.15}$$

By substituting from equation (2.13) into equation (2.12), the latter takes the following form:

$$f(x) = f(x_0) + \frac{h}{1!} f'(x_0) + \frac{h^2}{2!} f''(x_0) + \frac{h^3}{3!} f''(x_0) + \dots \tag{2.16}$$

And now, considering the expansion in equation (2.13) we can logically infer the steps to numerically find the solution to the given ordinary differential equation in equation (2.1) Based on this, the following are the main steps for finding the numerical solution to an ordinary differential equation using the Taylor series expansion:

1. The differential equation to be solved must be formulated in the general form mentioned earlier, known by the number (2.1)

2. Calculate the higher-order derivatives that appear in the expansion (2.13)

3. Find the values of these derivatives at the point of expansion, which is defined by the initial condition $y(x_0) = y_0$, along with the differential equation to be solved.

4. In the final step, substitute the initial condition and the derivatives at the expansion point into the general expansion equation.

Example 2.4. *Solve the following boundry value problem* $y' = \sin(x)$ *at* $x = 0.2$ *given that* $y(0) = -1$

solution:

In the beginning, and since we are going to Taylor series method to numerically solve the given differential equation, the first step in the solution is to ensure that the equation is in the standard form, as defined in equation (2.1).Looking at the given equation and comparing it with the standard in the standard form,we find that the provided equation is indeed in the standard form. There for:

$$\begin{aligned} y' &= \sin(x) \\ y'' &= \cos(x) \\ y''' &= -\sin(x) \\ &\vdots \\ y^{(n)} &= \sin\left(x + n\frac{\pi}{2}\right) \end{aligned}$$

The next step is to compute the equations at the expansion point, and this point is determined by the given initial condition $y(0) = 1$.

This means that the expansion point is at $x = 0$.

$$\begin{aligned} y(0) &= -1 \\ y'(0) &= \sin(0) = 0 \\ y''(0) &= 1 \\ y'''(0) &= -1 \end{aligned}$$

in this method, we point out an important point, which is when Solving an equation, regardless of the accuracy required for the Solution, it is sufficient to stop with the expansion after 3 terms. the length of the step has a dived impact on accuracy.

To illustrate this, we will find the solution twice : first with a period length of 0.2, and second with a period length 0.1 .

Case (1) : $h = 0.2$

assume the point of expansion Zero, therefore:

$$h = x - x_0 = 0.2 - 0 = 0.2$$

By substituting the equations' images at the expansion point into the given Taylor series expansion equation(2.13)

$$f(x) = f(x_0) + \frac{h}{1!}f'(x_0) + \frac{h^2}{2!}f''(x_0) + \frac{h^3}{3!}f'''(x_0) + \dots$$
$$f(x) = -1 + \frac{(0.2)}{1!}(0) + \frac{(0.2)^2}{2!}(1) + \frac{(0.2)^3}{3!}(0) + \cdots = -0.98$$

Case (2) : $h = 0.1$

in this case, to find the required value, we should apply Taylor series twice time. Considering the expansion point to be Zero, we continue solving in the same manner.

$$h = x - x_0 = 0.1 - 0 = 0.1$$

$$f(x) = -1 + \frac{(0.1)}{1!}(0) + \frac{(0.1)^2}{2!}(1) + \frac{(0.1)^3}{3!}(0) + \cdots = -0.995 \tag{2.17}$$

the next step is to consider the point of expansion to be 0.1. Therefore it is necessary to evaluate all derivatives at this new value, i.e,

$$y(0.1) = -0.9950$$
$$y'(0.1) = 0.09983$$
$$y''(0.1) = 0.9950$$
$$y'''(0.1) = -0.09983$$

again, Substitute into Taylor series, Yields:

$$f(0.2) = -0.980042$$

The following tables compare the numerical results with the exact solution for this problem.

Differential equation	exact solution	$h = 0.2$	$h = 0.1$
$y' = \sin(x)$	$y = -\cos(x)$ $y = -\cos(0.2) = 0.980066$	-0.98	-0.980042

MATLAB code

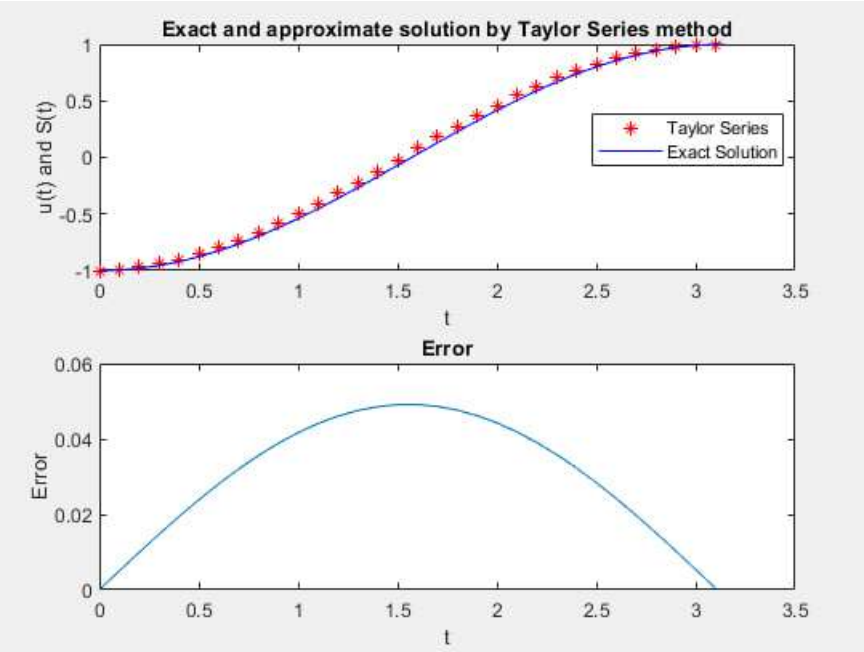
```
% Taylor Series method(second 3)
% y'=f(x,y)
```

```
% x=[a b]
% y(a)= u(1)=alpha
%S is the Exact Solution of y'
clc
clear
syms t y
f(t,y) =sin(t);
S(t) = -cos(t);
a = 0;
b = pi;
x = a:0.1:b;
n=length(x);
h=x(2)-x(1);
u(1) =-1;
g(t,y)=f(t,y)+h*diff(f,t,1)+(h^2/2)*diff(f,t,2)+(h^3/6)*diff(
   f,t,3);
for i = 1:n-1
    u(i+1) = u(i) +h*g(x(i),u(i));
    er(i) = abs(S(x(i)) - u(i));
end
er(n) = abs(S(x(n)) - u(n));
subplot(2,1,1)
plot(x, u, 'r*')
hold on
fplot(S, [a b], 'b')
title('Exact and approximate solution by Taylor Series method
   ')
xlabel('t')
ylabel('u(t) and S(t)')
legend('Taylor Series', 'Exact Solution')

```

```
subplot(2,1,2)
plot(x, er)
title('Error')
xlabel('t')
ylabel('Error')
```

Program results:



Format(2-1)

2.2.2.2 Euler method

Explicit Euler:

Previously, we defined the numerical solution of an ordinary differential equation as finding the values of the dependent variable, regardless of its name, at specific values of the independent variable. Let's assume that we have a differential equation, and the numerical solution for this equation has been found at specific points as follows:

$$x_0 < x_1 < x_2 < \cdots < x_n \leq b \tag{2.18}$$

Let's assume the following:

$$x_n = x_0 + nh, \qquad n = 0, 1, 2, 3, \ldots \tag{2.19}$$

Based on that, the solution takes the following form:

$$y(x_n) = y_h(x_n) = y_n \tag{2.20}$$

Now we will derive the general form of Euler's formula of derivation. Let's start by assuming that the first-order differential equation is as follows:

$$\frac{dy(x)}{dx} = f(x, y(x)), \qquad x_0 \leq x \leq b \tag{2.21}$$

with

$$y(x_0) = y_0 \tag{2.22}$$

Let's assume that $y(x)$ is the true solution to the differential equation we want to solve. Based on that, the equation (**??**) can be approximated as follows:

$$y(x_0) = y_0 \tag{2.23}$$

At the point $x = x_n$, the derivative $\dfrac{dy(x)}{dx}$ can be approximated as follows:

$$\frac{dy(x)}{dx} \simeq f(x_n, y(x_n)) \tag{2.24}$$

By substituting equation (**??**) into equation (**??**), the latter takes the following form:

$$\frac{1}{h}\left[y(x_{n+1}) - y(x_n)\right] \simeq f(x_n, y(x_n)) \tag{2.25}$$

Equation (**??**) in a more simplified form is as follows:

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{2.26}$$

The equation (**??**) represents the general solution for a first-order differential equation using Euler's general formula.

Implicit Euler:

The implicit Euler algorithm is given by:

$$\begin{cases} y_0 \qquad given \\ y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}) \end{cases} \tag{2.27}$$

Example 2.5. *given*

$$y' = \sin(x) \tag{2.28}$$

with

$$y(0) = -1 \tag{2.29}$$

obtain y at $x = 0.02$*,* 0.04*,* 0.06 *and* 0.08*.*

solution:

We have a first-order differential equation given by equation (**??**), and the solution to the equation at $x = 0$ is given as a boundary condition in equation (**??**). The task is to find the Solution to the equation, i.e., determine the values of y at a specific points the solution begins with writing the general form of Euler's method as follows:

$$y_{n+1} + hf(x_n, y_n) \tag{2.30}$$

and

$$h = x_{n+1} - x_n \tag{2.31}$$

The numerical solution for the differential equation at the point $x = 0.02$ corresponding to the value $x = 0.04$ and the step size is as follows:

$$h = x_1 - x_0 = 0.02 - 0.0 = 0.02 \tag{2.32}$$

Based on that, we calculate the value of $y(0.02)$ follows :

$$y_1 = y_0 + hf(x_o, y_0) = -1.0 + 0.02 \times (sin(0)) = 1.0 \tag{2.33}$$

The numerical solution for the differential equation at the point $x = 0.04$ corresponding to the value n=1, and the step size is as follows:

$$h = x_2 - x_i = 0.04 - 0.02 = 0.02 \tag{2.34}$$

Therefore

$$y_2 = y_1 + hf(x_1, y_1) = 1.0 + 0.02 \times (sin(0.02)) = -0.9998 \tag{2.35}$$

By following the same previous steps, we can obtain the solution at the remaining points for the independent variable, and we can compile all the results in the following table:

n	x_n	y_n
0	0.00	-1.0000
1	0.02	-1.0000
2	0.04	-0.9998
3	0.06	-0.9996
4	0.08	-0.9994

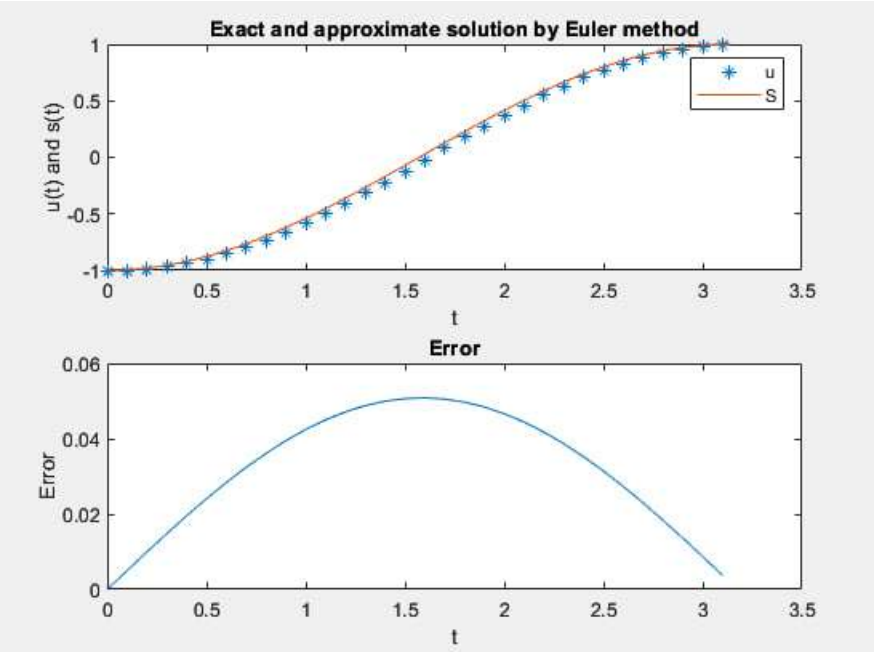
Table 2.1: Caption

MATLAB code:

```
% Euler Method
% y'=f(x,y)
% x=[a b]
% y(a)= u(1)=alpha
%S is the Exact Solution of y'
clc
clear
```

```
syms t y
f(t,y)=sin(t);
S(t)=-cos(t);
u(1)=-1;
a=0;
b=pi;
x=[a:0.1:b];
n=length(x);
h=x(2)-x(1);
for i=2:n
   u(i)=u(i-1)+h*f(x(i-1),u(i-1));
   er(i)=abs(S(x(i))-u(i));
end
subplot(2,1,1)
plot(x,u,'*')
hold on
fplot(S,[a b])
legend('u','S')
title('Exact and approximate solution by Euler method')
xlabel('t');
ylabel('u(t) and s(t)')
subplot(2,1,2)
plot(x,er)
title('Error')
xlabel('t');
ylabel('Error')
```

Program results:



Format(2-2)

2.2.2.3 Modified Euler method

The method you are describing is known as the Modified Euler method or Improved Euler method. It relies on successive improvement for each calculated value of the dependent variable. The improvement process continues until a specific accuracy is achieved, either predetermined or after a certain number of iterations.

The steps of the solution are as follows:

- We start with the ordinary Euler's formula to find the initial approximation of the solution.
- Calculate the average of the first derivative in the first step and the derivative of the first approximation
- Using the average calculated from the second step to find the approximation for the second iteration.
- We continue repeating the three previous steps until we achieve the desired accuracy or until the specified number is reached to calculate the solution at each point.
- The following example will illustrate these steps in detail.

Example 2.6. *Given*

$$y' = \sin(x) \tag{2.36}$$

With

$$y(0) = (1) \tag{2.37}$$

Use modified Euler to obtain y at $x = 0.01$ up to the third approximation

Solution: The current example involves solving a first-order differential equation with a boundary condition at point $x = 0$. the goal is to find the solution of this equation at point $x = 0.1$ using the modified euler method for three consecutive approximations.

First, we begin by writing the general equation of the modified euler method, updating the notation to indicate iteration. Instead of writing y_1, for example, it is modified as follows $y_1^{(1)}$ represents the first approximation, $y_2^{(2)}$ represents the second approximation, and so on..

$$y_1^{(1)} = y_0 + hf(x_0, y_0) \tag{2.38}$$

and

$$\begin{aligned} h &= X_1 - X_0 \\ &= 0.1 - 0.0 \\ &= 0.1 \end{aligned} \tag{2.39}$$

$$y_1^{(1)} = -1.0 + (0.1)x\sin(0) = -1 \tag{2.40}$$

The next step is to write the general formula for the modified euler method as follows:

$$y_1^{(n+1)} = y_{i-1} + \frac{h}{2}\left[f(x_{i-1}, y_{i-1}) + f\left(x_{i-1} + h, y_i^{(n)}\right)\right], \tag{2.41}$$

We notice the presence of two indices in the new equation (6). The first one, "n", is used to determine the number of approximations. The second one, 'i', is used to differentiate between the values of the dependent variable being used. We use two

different values, one of which remains constant at the same point we calculate, while the other varies successively. Therefore, if we take "$i = 1$" and "$n = 1$", we obtain the second approximation as follows:

$$y_1^{(2)} = y_0 + \frac{h}{2}\left[f(x_0, y_0) + f\left((x_0 + h), y_1^{(1)}\right)\right]$$

by observing the final limit of equation (7), We find that it represents the first approximation that we used in Euler's equation. Non, by Calculating equation (7), we can obtain the second approximation.

$$y_1^{(2)} = -1.0 + \frac{(0.1)}{2}\left[\sin(0) + \sin(0.0 + 0.1)\right] = -0.9004 \tag{2.42}$$

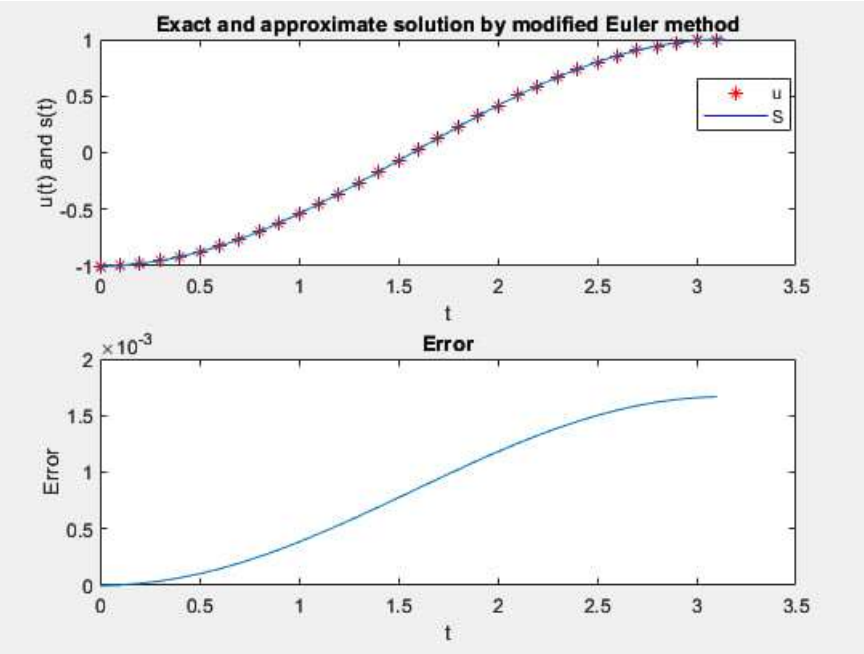
$$\begin{aligned} y_1^{(3)} &= y_0 + \frac{h}{2}\left[f(x_0, y_0) + f(x_0 + h), y_1^{(2)}\right] \\ y_1^{(3)} &= 1.0 + \frac{(0.1)}{2}\left[\sin(0) + \sin(0.0 + 0.1)\right] = -0.9004 \end{aligned} \tag{2.43}$$

MATLAB code:

```
% Modified Euler Method
% y'=f(x,y)
% x=[a b]
% y(a)= u(1)=alpha
%S is the Exact Solution of y'
clc
clear
syms t y
f(t,y)=sin(t);
S(t)=-cos(t);
u(1)=-1;
 a=0;
 b=pi;
```

```
x=[a:0.1:b];
n=length(x);
 h=x(2)-x(1);
for i=2:n
   u(i)=u(i-1)+h*f(x(i-1)+h,u(i-1))/2+h*(f(x(i-1),u(i-1))/2);
   er(i)=abs(S(x(i))-u(i));
end
subplot(2,1,1)
plot(x,u,'*')
hold on
fplot(S,[a b])
legend('u','S')
title('Exact and approximate solution by modified Euler
   method')
xlabel('t')
ylabel('u(t) and s(t)')
subplot(2,1,2)
plot(x,er)
title('Error')
xlabel('t')
ylabel('Error')
```

Program results:



Format(2-3)

2.2.2.4 Runge kutta method

Previously, we extensively discussed the solution of a first-order differential equation using Taylor's expansion, Euler's method, and Modified Euler's method. In the following, we will present the Runge Kutta method for solving a first-order differential equation in the form:

$$\frac{dy}{dx} = f(x, y)$$

Given a boundary condition:

$$y(x_0) = y_0$$

Now, let's delve into the Runge Kutta method for solving this first-order differential equation. The required solution for the equation at successive points starting from the initial point x_0 follows the strategy outlined in the steps.

$$y_1 = y_0 + K_{ave}$$

Where

$$K_{ave} = \frac{1}{6}\{K_1 + 2 \cdot (K_2 + K_3) + K_4\} \tag{2.44}$$

and

$$\begin{aligned} K_1 &= hf(x_0, y_0) \\ K_2 &= hf\left(x_0 + \frac{h}{2}, y_0 + \frac{K_1}{2}\right) \\ K_3 &= hf\left(x_0 + \frac{h}{2}, y_0 + \frac{K_2}{2}\right) \\ K_4 &= hf\left(x_0 + h, y_0 + K_3\right) \end{aligned}$$

Example 2.7. *we apply this in method. to the same previous example*

$$y' = f(x, y) = \sin(x) \tag{2.45}$$

$$y(x_0 = 0) = y_0 = -1 \tag{2.46}$$

use the Runge-Kutte method to obtain the values of y *at* $x = 0.1$ *.*

Solution:

1^{st} step: determine the step length as follow:

$$\begin{aligned} h &= x - x_0 \\ &= 0.1 - 0 = 0.1 \end{aligned}$$

2^{nd} step: calculate the different value of K_i:

$$\begin{aligned} K_1 &= hf(x_0, y_0) = (0.1)(0) = 0 \\ K_2 &= hf(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}) = (0.1) \cdot (\sin(0.05)) = 0.0000872665 \\ K_3 &= hf(x_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}) = (0.1) \cdot (\sin(0.05)) = 0.0000872665 \\ K_4 &= hf(x_0 + h, y_0 + k_3) = (0.1) \cdot (\sin(0.1)) = 0.0001745328 \end{aligned}$$

Now:

$$K_{ave} = \frac{1}{6}\left(K_1 + 2\left(K_2 + K_3\right) + K_4\right)$$
$$= K_{ave} = 0.04996666$$

So,

$y_1 = y_0 + K_{ave} \Longrightarrow y_1 \simeq -1 + 0.008327776$

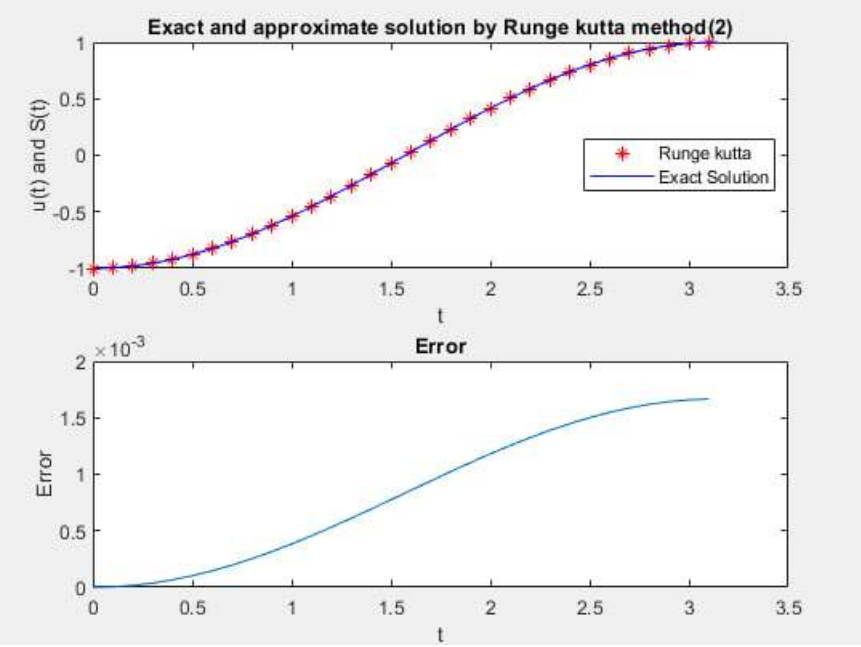
$\Rightarrow y_1 \simeq -0.991672224$

MATLAB code

```
% Runge kutta method(second order)
% y'=f(x,y)
% x=[a b]
% y(a)= u(1)=alpha
%S is the exact solution of y'
clc
clear
syms t y
f(t,y)=sin(t);
S(t)=-cos(t);
a=0;
b=pi;
x=[a:0.1:b];
n=length(x);
u(1)=-1;
h=x(2)-x(1);
for i=2:n
k1=h*f(x(i-1),u(i-1));
k2=h*f(x(i-1)+h,u(i-1)+k1);
kave=(k1+k2)/2;
u(i)=u(i-1)+kave;
 er(i)=abs(S(x(i))-u(i));
end
```

```
subplot(2,1,1)
plot(x,u,'r*')
hold on
fplot(S,[a b],'b')
title('Exact and approximate solution by Runge kutta method
   (2)');
xlabel('t');
ylabel('u(t) and S(t)');
legend('Runge kutta', 'Exact Solution');
subplot(2,1,2)
plot(x,er)
title('Error');
xlabel('t');
ylabel('Error');
```

Program results:

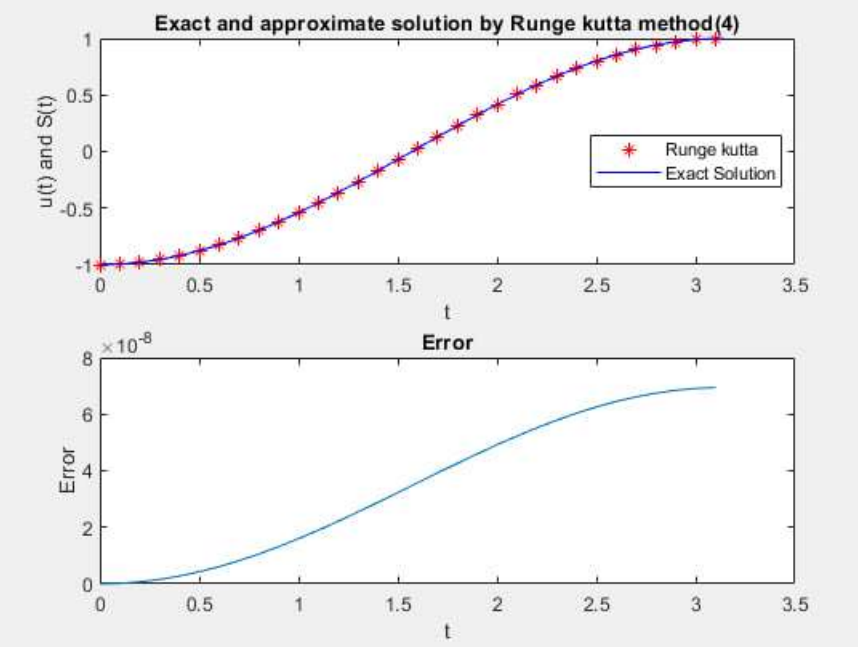


Format(2-4)

```
% Runge kutta method(fourth order)
% y'=f(x,y)
% x=[a b]
% y(a)= u(1)=alpha
%S is the original function y'
clc
clear
syms t y
f(t,y)=sin(t);
S(t)=-cos(t);
a=0;
b=pi;
x=[a:0.1:b];
n=length(x);
u(1)=-1;
h=x(2)-x(1);
for i=2:n
```

```
k1=h*f(x(i-1),u(i-1));
k2=h*f(x(i-1)+h/2,u(i-1)+k1/2);
k3=h*f(x(i-1)+h/2,u(i-1)+k2/2);
k4=h*f(x(i-1)+h,u(i-1)+k3);
kave=(k1+2.*(k2+k3)+k4)/6;
u(i)=u(i-1)+kave;
 er(i)=abs(S(x(i))-u(i));
end
subplot(2,1,1)
plot(x,u,'r*')
hold on
fplot(S,[a b],'b')
title('Exact and approximate solution by Runge kutta method
   (4)')
xlabel('t')
ylabel('u(t) and S(t)')
legend('Runge kutta', 'Exact Solution')
subplot(2,1,2)
plot(x,er)
title('Error')
xlabel('t')
ylabel('Error')
```

Program results:



Format(2-5)

2.2.2.5 Multistep methods

An **m-step multistep method** for solving the initial-value problem

$$y' = f(x, y), \ a \leq x \leq b, \tag{2.47}$$

$$y(a) = \alpha \tag{2.48}$$

has a difference equation for finding the approximation ω_{i+1} at the mesh point x_{i+1} represented by the following equation, where m is an integer greater than 1:

$$\begin{aligned} \omega_{i+1} &= a_{m-1}\omega_i + a_{m-2}\omega_{i-1} + \cdots + a_0\omega_{i+1-m} \\ &+ h\left[b_m f(x_{i+1}, \omega_{i+1}) + b_{m-1} f(x_i, \omega_i)\right. \\ &\left. + \cdots + b_0 f(x_{i+1-m}, \omega_{i+1-m})\right] \end{aligned} \tag{2.49}$$

for $i = m-1, m \ldots N-1$, where $h = (b-a)/N$, the $a_0, a_1, \ldots, a_{m-1}$ and $b_0, b_1 \ldots b_m$ are constants, and the starting values

$$\omega_0 = \alpha, \quad \omega_1 = \alpha_1, \quad \omega_2 = \alpha_2, \quad \omega_{m-1} = \alpha_{m-1}$$

are specified.

When $b_m = 0$ the method is called **explicit**, or **open**, because Eq. (2.44) then gives ω_{i+1} explicitly in terms of previously determined values. When $b_m \neq 0$ the method is called

implicit, or **closed**, because ω_{i+1} occurs on both sides of Eq. (2.44), so ω_{i+1} is specified only implicitly.

For example, the equations

$$\begin{aligned}
&\omega_0 = \alpha, \quad \omega_1 = \alpha_1, \quad \omega_2 = \alpha_2, \quad \omega_3 = \alpha_3, \\
&\omega_{i+1} = \omega_i + \frac{h}{24}\left[55f(x_i, \omega_i) - 59f(x_{i-1}, \omega_{i-1}) + 37f(x_{i-2}, \omega_{i-2}) - 9f(x_{i-3}, \omega_{i-3})\right].
\end{aligned} \tag{2.50}$$

for each $i = 3, 4, \ldots, N-1$, define an explicit four-step method known as the **fourth-order Adams-Bashforth technique**. The equations

$$\begin{aligned}
&\omega_0 = \alpha, \quad \omega_1 = \alpha_1, \quad \omega_2 = \alpha_2, \\
&\omega_{i+1} = \omega_i + \frac{h}{24}\left[9f(x_{i+1}, \omega_{i+1}) + 19f(x_i, \omega_i) - 5f(x_{i-1}, \omega_{i-1}) + f(x_{i-2}, \omega_{i-2})\right]
\end{aligned} \tag{2.51}$$

for each $i = 2, 3, ...N-1$, define an implicit three-step method known as **the fourth-order Adams-Moulton technique.**

Adams-Bashforth explicit methods

Some of the explicit multistep methods together with their required starting values and local truncation errors are as follows.

- Adams-Bashforth two-step explicit method:

$$\begin{aligned}
&\omega_0 = \alpha, \quad \omega_1 = \alpha_1, \\
&\omega_{i+1} = \omega_i + \frac{h}{2}\left[3f(x_i; \omega_i) - f(x_{i-1}, \omega_{i-1})\right]
\end{aligned} \tag{2.52}$$

  where $i = 1, 2, ..., N-1$. The local truncation error is $\tau_{i+1}(h) = \frac{5}{12}y'''(\mu_i)^2$, for some $\mu_i \in (x_{i-1}, x_{i+1})$.

- Adams-Bashforth three-step explicit method:

$$\omega_0 = \alpha, \omega_1 = \alpha_1, \omega_2 = \alpha_2,$$
$$\omega_{i+1} = \omega_i + \frac{h}{12}\left[23f(x_i, \omega_i) - 16f(x_{i-1}, \omega_{i-1}) + 5f(x_{i-2}, \omega_{i-2})\right]$$

where $i = 2, 3.....N-1$. The local truncation error is $\tau_{i+1}(h) = \frac{3}{8}y^{(4)}(\mu_i)h^3$, for some $\mu_i \in (x_{i-2}, x_{i+1})$.

- Adams-Bashforth four-step explicit method:

$$\omega_0 = \alpha \quad \omega_1 = \alpha_1, \quad \omega_2 = \alpha_2, \quad \omega_3 = \alpha_3,$$
$$\omega_{i+1} = \omega_i + \frac{h}{24}\left[55f(x_i, \omega_i) - 59f(x_{i-1}, \omega_{i-1}) + 37f(x_{i-2}, \omega_{i-2}) - 9f(x_{i-3}, \omega_{i-3})\right].$$

where $i = 3, 4.....N-1$. The local truncation error is $\tau_{i+1}(h) = \frac{251}{720}y^{(5)}(\mu_i)h^4$, for some $\mu_i \in (x_{i-3}, x_{i+1})$.

- Adams-Bashforth five-step explicit method :

$$\omega_0 = \alpha, \quad \omega_1 = \alpha_1, \quad \omega_2 = \alpha_2, \quad \omega_3 = \alpha_3, \quad \omega_4 = \alpha_4,$$
$$\omega_{i+1} = \omega_i + \frac{h}{720}\left[1901f(x_i, \omega_i) - 2774f(x_{i-1}, \omega_{i-1}) + 2616f(x_{i-2}, \omega_{i-2}) - 1274f(x_{i-3}, \omega_{i-3})\right.$$
$$\left. +251f(x_{i-4}, \omega i - 4)\right].$$

where $i = 4, 5.....N-1$. The local truncation error is $\tau_{i+1}(h) = \frac{95}{288}y^{(6)}(\mu_i)h^5$, for some $\mu_i \in (x_{i-4}, x_{i+1})$.

Adams-Moulton implicit methods

Implicit methods are derived by using $(x_{i+1}, f(x_{i+1}, y(x_{i+1})))$ as an additional interpolation node in the approximation of the integral

$$\int_{x_1}^{x_{i+1}} f(x, y(x))dx$$

Some of the more common implicit methods are as follows.

- Adams-Moulton two-step implicit method:

$$\begin{aligned} &\omega_0 = \alpha, \quad \omega_1 = \alpha_1, \\ &\omega_{i+1} = \omega_i + \frac{h}{12}\left[5f(x_{i+1}, \omega_{i+1}) + 8f(x_i, \omega_i) - f(x_{i-1}, \omega_{i-1})\right] \end{aligned} \tag{2.53}$$

  where $i = 1, 2, ..., N-1$. The local truncation error is $\tau_{i+1}(h) = -\frac{1}{24}y^{(4)}(\mu_i)h^3$, for some $\mu_i \in (x_{i-1}, x_{i+1})$

- Adams-Moulton three-step implicit method:

$$\begin{aligned} &\omega_0 = \alpha, \quad \omega_1 = \alpha_1, \quad \omega_2 = \alpha_2, \\ &\omega_{i+1} = \omega_i + \frac{h}{24}\left[9f(x_{i+1}, \omega_{i+1}) + 19f(x_i, \omega_i) - 5f(x_{i-1}, \omega_{i-1}) + f(x_{i-2}, \omega_{i-2})\right] \end{aligned}$$

  where $i = 2, 3, ...N-1$. The local truncation error is $\tau_{i+1}(h) = -\frac{19}{720}y^{(5)}(\mu_i)h^4$, for some $\mu_i \in (x_{i-2}, x_{i+1})$

- Adams-Moulton four-step implicit method:

$$\begin{aligned} &\omega_0 = \alpha, \quad \omega_1 = \alpha_1, \quad \omega_2 = \alpha_2, \quad \omega_3 = \alpha_3, \\ &\omega_{i+1} = \omega_i + \frac{h}{720}\left[251f(x_{i+1}, \omega_{i+1}) + 646f(x_i, \omega_i) - 264f(x_{i-1}, \omega_{i-1}) + 106f(x_{i-2}, \omega_{i-2})\right. \\ &\left. -19f(x_{i-3}, \omega_{i-3})\right] \end{aligned}$$

  where $i = 3, 4.....N-1$. The local truncation error is $\tau_{i+1}(h) = -\frac{3}{160}y^{(6)}(\mu_i)h^5$, for some $\mu_i \in (x_{i-3}, x_{i+1})$

  It is interesting to compare an m-step Adams-Bashforth explicit method with an $(m-1)$-step Adams-Moulton implicit method. Both involve m evaluations of f per step, and both have the terms $y^{m+1}(\mu_i)h^m$ in their local truncation errors. In general, the coefficients of the terms involving f in the local truncation error are smaller for the implicit methods than for the explicit methods. This leads to greater stability and smaller round-off errors for the implicit methods.

Example 2.8. *Consider the initial-value problem*

$$y' = \sin(x), 0 \leq x \leq \pi, y(0) = -1$$

Use the exact values given from $y(x) = -\cos(x)$ *as Starting Values and* $h = 0.2$ *to compare the approximations from* (a) *by the explicit Adams-Bashforh four-step method and* (b) *the implicit Adams-Moultion Three-Step method.*

Solutions:

1. the Adams-Bashforth method has the difference equation

   $W_{i+1} = W_i + \frac{h}{24}[55f(x_i, W_i) - 59f(x_{i-1}), W_{i-1}) + 37f(x_{i-2}, W_{i-2}) - 9f(x_{i-3}, W_{i-3})].$

   for $i = 3, 4, \ldots, 9$. When Simplified using $f(x, y) = \sin(x) h = 0.2$, and $x_i = ih$, it becomes

   $W_{i+1} = W_i + \frac{h}{24}[55\sin(x_i) - 59\sin(x_{i-1}) + 37\sin(x_{i-1}) - 9\sin(x_{i-3})]$

2. the Adams-Moulton method has the difference equation

   $W_{i+1} = W_i + \frac{h}{24}[9f(x_{i+1}, W_{i+1}) + 19f(x_i, w_i) - f(x_{i-1}, w_{i-1}) + f(x_{i-2}, w_{i-2})]$

   for $i = 2, 3, \ldots, 9$ this reduces to

   $W_{i+1} = W_i + \frac{h}{24}[9\sin(9\sin(x_{i+1}) + 19\sin(x_i) - 5\sin(x_{i-1}) + \sin(x_{i-2})]$

   The results in table were obtained using the exact Values form $y(x) = -\cos(x)$ for $\alpha, \alpha_1, \alpha_2$ and α_3 in the implicit Adams-Bashforth Case for α, α_1 and α_2 in the implicit Adams - Moulton Case. Note that the implicit Adams-Moulton method gives consistently better results.

x_i	Exact	Adams-Bashforth w_i	Error	Adams Moulton w_i	Error
0.0	−1.000000				
0.2	−1.980066				
0.4	−0.921061				
0.8	−0.825335	−0.82533561	0.00004207	−0.825333	0.00000282
1.0	−0.696706	−0.696748	0.0001037	−0.69670	0.00000716
1.2	−0.540302	−0.540406	0.0001824	−0.540289	0.00001284
1.4	−0.169967	−0.362540	0.000275	−0.362338	0.000019648
1.6	0.029199	−0.170242	0.0003779	−0.16994	0.0000272
1.8	0.227202	0.02888215	0.000487	0.029235	0.0000354
2	0.4161468	0.226715	0.0005978	0.227246	0.00005218

MATLAB code:

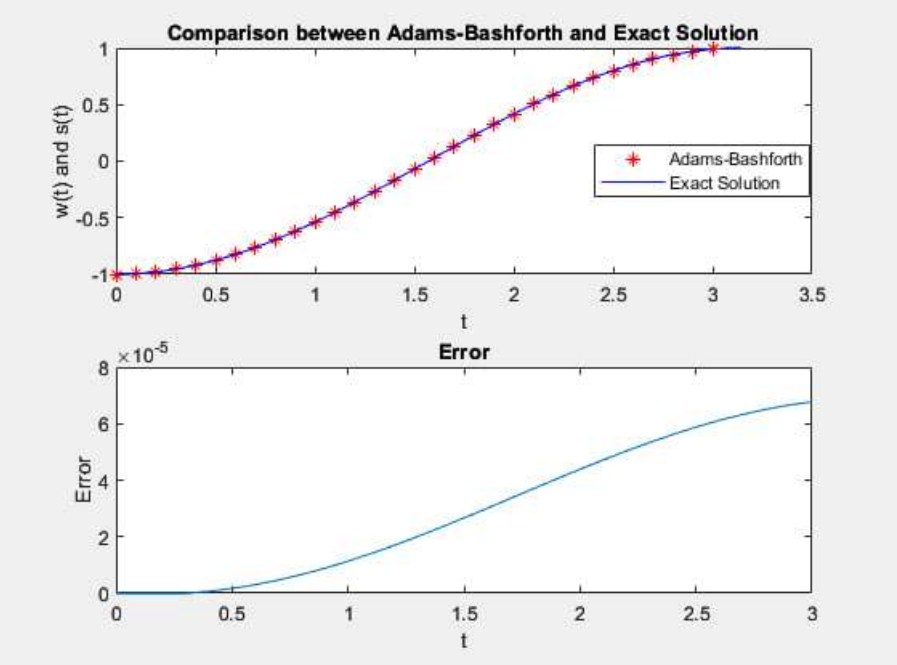
```
% Adams-Bashforth method
% y'=f(x,y)
% x=[a b]
% y(a)= u(1)=alpha
%S is the Exact Solution of y'
clc
clear
syms t y
f(t,y) = sin(t);
s(t) = -cos(t);
a=0;
b=3;
x = [a:0.2:b];
h = x(2) - x(1);
w(1) = -1;
w(2) = s(x(2));
w(3) = s(x(3));
w(4) = s(x(4));
n = length(x);
```

```
for i = 4:n-1
    w(i+1) = w(i) + (h/24) * (55*f(x(i), w(i)) - 59*f(x(i-1),
        w(i-1)) + 37*f(x(i-2), w(i-2)) - 9*f(x(i-3), w(i-3)))
       ;

end
er=abs(w-s(x));
subplot(2,1,1);
plot(x, w, 'r*');
hold on
fplot(s, [a b], 'b')
title('Exact and approximate solution by Adams-Bashforth
   method')
xlabel('t')
ylabel('w(t) and s(t)')
legend('Adams-Bashforth', 'Exact Solution')

subplot(2,1,2);
plot(x, er);
title('Error')
xlabel('t')
ylabel('Error')
```

Program results:



Format(2-6)

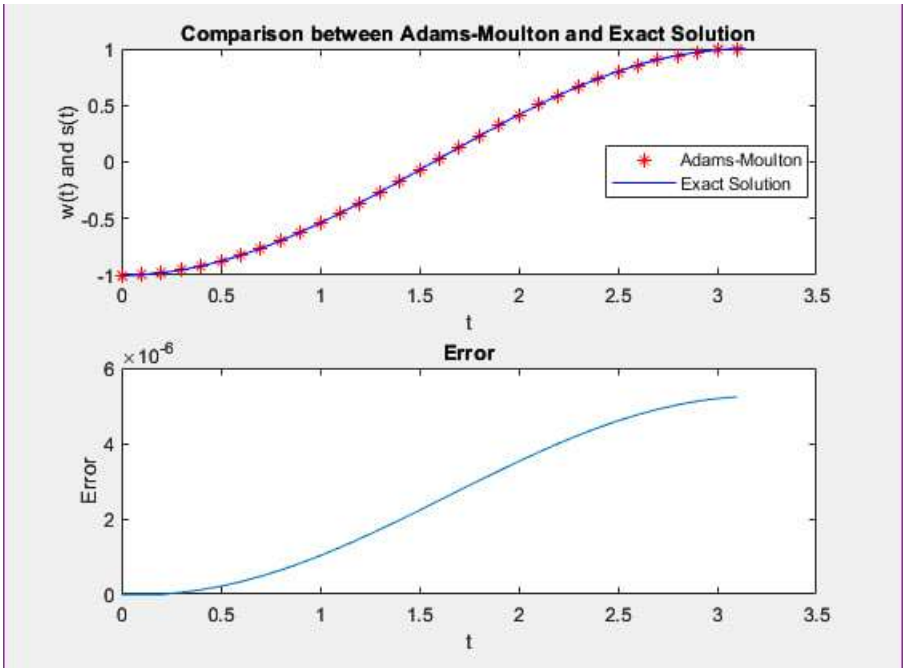
```
% Adams-Moulton method
% y'=f(x,y)
% x=[a b]
% y(a)= u(1)=alpha
%S is the original function y'
clc
clear
syms t y
f(t,y) = sin(t);
s(t) = -cos(t);
a=0;
b=pi;
x = [a:0.2:b];
h = x(2) - x(1);
w(1) = -1;
w(2) = s(x(2));
w(3) = s(x(3));
```

```
n = length(x);
for i = 3:n-1
   % w(i+1) = w(i) + (h/24) * (9*f(x(i+1), w(i+1)) + 19*f(x(i
      ), w(i)) - 5*f(x(i-1), w(i-1)) + f(x(i-2), w(i-2)))
   w(i+1) = w(i) + (h/24) * (9*sin(x(i+1)) + 19*sin(x(i)) -
      5*sin(x(i-1)) + sin(x(i-2)));

end
er = abs(w - s(x));
subplot(2,1,1);
plot(x, w, 'r*');
hold on;
fplot(s, [a b], 'b');
title('Comparison between Adams-Moulton and Exact Solution');
xlabel('t');
ylabel('w(t) and s(t)');
legend('Adams-Moulton', 'Exact Solution');

subplot(2,1,2);
plot(x, er);
title('Error');
xlabel('t');
ylabel('Error');
```

Program results:



Format(2-7)

2.3 Second-order differential equations

A second-order differential equation involves the second derivative of an unknown function. In general form, a second-order ordinary differential equation (ODE) can be written as:

$$\frac{d^2y}{dx^2} = f\left(x, y, \frac{dy}{dx}\right)$$

Here, y is the dependent variable, x is the independent variable, and f is a function that may depend on x, the first derivative $\frac{dy}{dx}$, and y.

The solution to a second-order ODE typically involves finding a function $y(x)$ that satisfies the given differential equation along with specified initial or boundary conditions.

There are many numerical methods for solving second-order differential equations, and we will mention them: finite difference method, Runge-Kutta method, Euler's method, spectral method, and finite element method.

In this title , discuss the most famous methods, including: finite difference Method and finite element method.

2.3.1 Numerical methods for solving second order differential equations

The principle of the finite difference method (DF) involves selecting a certain number of points in the domain, denoted as $(x_1, x_2, \ldots, x_{\mathbb{N}})$ in $\mathbb{R}^N$. The differential operator is approximated using quotients, leading to the derivation of a system of equations in terms of discrete unknowns. These discrete unknowns represent approximations of the unknown function u at the discretization points.

Let's perform a Taylor expansion around x_i up to the third order:

- $y(x_{i+1}) = y(x_i + h) = y(x_i) + h\frac{dy}{dx}(x_i) + \frac{h^2}{2}\frac{d^2y^2}{d^2x^2}(x_i) + 0(h^3)$
- $y(x_{i-1}) = y(x_i - h) = y(x_i) - h\frac{dy}{dx}(x_i) + \frac{h^2}{2}\frac{d^2y}{d^2x^2}(x_i) + 0(h^3)$

The subtraction of these two expressions yields:

$$y(x_{i+1}) - y(x_{i-1}) = 2h\frac{dy}{dx}(x_i) + 0(h^3)$$

2.3.1.1 Finite difference method

Let us consider a model problem:

$$u''(x) = f(x), \quad 0 < x < 1, \quad u(0) = u_a, \quad u(1) = u_b$$

to illustrate the general procedure using a finite difference method as follows.

1. Creation of net work. the network is a limited collection of points where we seek the values of a function that represents an approximate solution to the differential equation. for instance, when we take the positive integer $n > 0$, We can use an equidistant interval network.

$$x_i = ih, \quad i = 0, 1, \ldots, n, \quad h = \frac{1}{n}$$

If we aim to obtain four significant lines in the approximate solution, we can choose the parameter n according to the accuracy requirement. For example, we can choose $n = 100$ or a larger value.

2. Represents the result of the finite difference thread at each grid point. Where the solution is unknown in obtain an algebraic system of equations. note which we have for the doubly unique function $\Theta(x)$

$$\Theta''(x) = \lim_{\Delta x \to 0} \frac{\Theta(x - \Delta x) - 2\Theta(x) + \Theta(x + \Delta x)}{(\Delta x)^2}$$

therefore, we can approximate the nearest function values to $u''(x_i)$ to obtain the finite difference formula for the second order derivative at the grid point x_i is

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x_i) + u(x_i + h)}{h^2}$$

with some errors in the approximation, in the finite difference method, we replace the differential equation for each grid point x_i with:

$$\frac{u(x_i - h) - 2u(x_i) + u(x_i + h)}{h^2} = f(x_i) + \text{error}$$

the local truncation fault will be reviewed later.we therefore define a finite difference (FD) Solution (approximation) for $u(x)$ over all x_i as a solution u_i (if any) with the following system of algebraic equations:

$$\frac{u_a - 2u_1 + u_2}{h^2} = f(x_1)$$
$$\frac{u_1 - 2u_2 + u_3}{h^2} = f(x_2)$$
$$\cdots\cdots\cdots$$
$$\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = f(x_i)$$
$$\frac{u_{n-2} - 2u_{n-1} - u_b}{h^2} = f(x_{n-1})$$

Please note that the finite difference equations for each grid point contain the solution values for the three grid points, namely x_{i-1}, x_i, and x_{i+1}. An array of three grid points is called a finite difference stencil.

3. Solve a system of algebraic equations to obtain an approximation of the solution

at each grid point. Systems of algebraic equations can be written in matrix and vector form.

$$\begin{bmatrix} \frac{-2}{h^2} & \frac{1}{h^2} & & & & & & \\ \frac{1}{h^2} & \frac{-2}{h^2} & \frac{1}{h^2} & & & & & \\ & \frac{1}{h^2} & \frac{-2}{h^2} & \frac{1}{h^2} & & & & \\ & \cdots & \cdots & \cdots & & & & \\ & & \cdots & \cdots & \cdots & & & \\ & & & \cdots & \cdots & \cdots & & \\ & & & & \cdots & \cdots & \cdots & \\ & & & & & \frac{1}{h^2} & \frac{-2}{h^2} & \frac{1}{h^2} \\ & & & & & & \frac{1}{h^2} & \frac{-2}{h^2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} f(x_1) - \frac{u_a}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_b}{h^2} \end{bmatrix}$$

Example 2.9. *Let's consider the following differential equation:*

$$\begin{cases} -u''(x) = \dfrac{1}{x^2}; \quad x \in]1;2[\\ u(1) = 0; \quad u(2) = ln(2) \qquad \textit{non-homogeneons dirichlet condition} \end{cases}$$

with $f \in C\left([1;2]\right);\mathbb{R})$.

Solution:

Let $h = \dfrac{1}{N+1}, \quad N \in \mathbb{N}^*, \quad w_h = \{x = ih; \quad i = 1, \ldots, N+1\}$

We have: $u''(x) = \dfrac{u_{i+1} - 2u_i + u_{i+1}}{h^2} + 0(h^*)$

so, Approaching $u''(x)$ *by differential quotient by Taylor developement, we obtain:*

$$\textit{the equation diserete} \begin{cases} -\dfrac{1}{h^2}(u_{i+1} - 2u_i + u_{i+1}) = f_i \quad 1 \le i \le N \\ u_0 = 0 \\ u_{N+1} = \ln 2 \end{cases}$$

$$\begin{aligned} \textit{for} \quad i = 1: \ & \frac{1}{h^2}\left(-u_0 + 2u_1 - u_2\right) = f_1 \qquad (u_0 = 0) \\ i = 2: \ & \frac{1}{h^2}\left(-u_1 + 2u_2 - u_3\right) = f_2 \\ i = N-1: \ & \frac{1}{h^2}\left(-u_{N-2} + 2u_{N-1} - u_N\right) = f_{N-1} \\ i = N: \ & \frac{1}{h^2}\left(-u_{N-1} + 2u_N - u_{N+1}\right) = f_N \quad (u_{N+1} = \ln 2) \end{aligned}$$

we can write an matrix for $A_h u_h = b_h$ with:

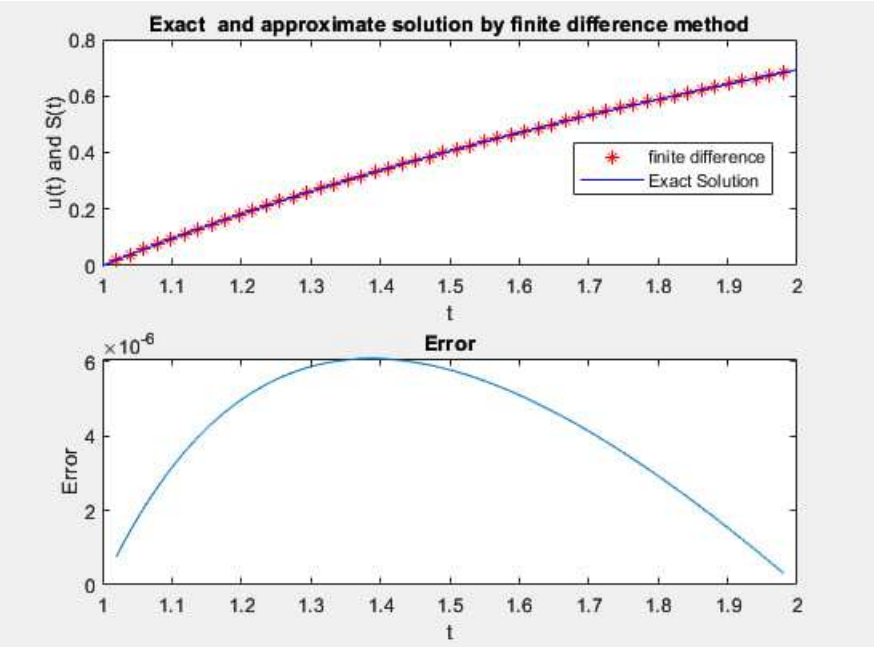
$$A_h = \begin{bmatrix} +2 & -1 & 0 & \dots & \dots & \dots & 0 \\ -1 & +2 & -1 & \dots & \dots & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & & \ddots & 0 \\ 0 & \dots & \dots & -1 & \ddots & & -1 \\ 0 & \dots & \dots & 0 & -1 & & +2 \end{bmatrix}, u_h = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix}, b_h = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \\ f_N - \frac{ln(2)}{h^2} \end{bmatrix}$$

MATLAB code:

```
% finite difference method
% y"=f
% x=[a b]
%S is the exact solution of y"
clear
clc
N=50;
a=1;
b=2;
c=log(1);
d=log(2);
syms t
f(t)=1/t^2;
S(t) =log(t);
h=(b-a)/(N+1);
A=diag(2*ones(1,N),0)+diag(-1*ones(1,N-1),-1)+diag(-1*ones(1,
   N-1),+1);
for i=1:N
x(i)=a+i*h;
bh(i)=vpa(h^2*f(x(i)));
end
bh(1)=vpa(h^2*(f(x(1))+c/h^2));
bh(N)=vpa(h^2*(f(x(N))+d/h^2));
```

```
Uh=inv(A)*bh'
er = zeros(1, N);
for i = 1:N
    er(i) = abs(S(x(i)) - Uh(i));
end
subplot(2,1,1)
plot(x,Uh,'r*')
hold on
fplot(S,[a b],'b')
title('Exact  and approximate solution by finite difference
   method')
xlabel('t')
ylabel('u(t) and S(t)')
legend('finite difference', 'Exact Solution')
subplot(2,1,2)
plot(x,er)
title('Error')
xlabel('t')
ylabel('Error')
hold off
```

Program results:



Format(2-8)

2.3.1.2 Finite element method

We have a second-order ordinary differential equation (ODE) in the form:

$$-u''(x) = f(x) \qquad \text{where} \qquad 0 < x < 1$$

the boundary conditions are $u(0) = 0$ and $u(1) = 0$

Let's rephrase the steps for illustrating the finite element method for the 1D two-point boundary value problem (BVP) using the Galerkin finite element method:

1. to constract a variational or weak formulation, we start by multiplying both sides of the given differential equation by a test function $v(x)$ that satisfies the boundary conditions $v(0) = 0$ and $v(1) = 0$. this yields:

$$u''(x)v(x) = f(x)v(x)$$

integrating both sides over the domain $0 < x < 1$, we have:

$$\int_0^1 -u''(x)v(x)dx = \int_0^1 f(x)v(x)dx$$

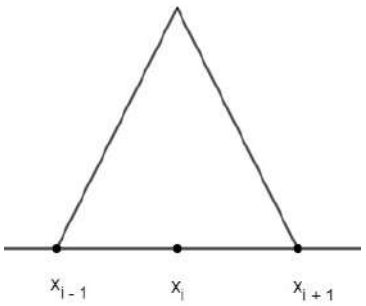
Now, integrating the left-hand side by parts, we get:

$$[-u'(x)v(x)]_0^1 + \int_0^1 u'(x)v'(x)dx = \int_0^1 f(x)v(x)dx$$

2. Generate a mesh, Such as a uniform cartesian mesh where $x_i = ih$ for $i = 0, 1, \ldots, n$ with $h = \dfrac{1}{n}$ defining intervals (x_{i-1}, x_i) for $i = 1, 2, \ldots, n$.

3. Formulate a set of basis functions derived from the mesh, such as the piecewise linear functions indexed from 1 to $n - 1$.

$$\psi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{h} & \text{if } \ x_{i-1} \le x < x_i \\ \dfrac{x_{i+1} - x}{h} & \text{if } \ x_i \le x < x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$



often called the hat functions, see the right diagram for a hat function.

4. Express the finite element (FE) solution as a linear combination of the basis functions:

$$u_h(x) = \sum_{j=1}^{n-1} cj\psi_j(x)$$

where the coefficients c_j are the unknowns to be determined when employing hat basis functions, it is evident that $u_h(x)$ is also a piecewise linear function, although this may not typically hold true for the true solution $u(x)$. Subsequently, we establish a linear system of equations for the coefficients by substituting the approximate solution $u_h(x)$ for the exact solution $u(x)$ in the weak form: $\int_0^1 u'v'dx = \int_0^1 fvdx$, i. e,

$$\begin{aligned} \int_0^1 u_h'v'dx &= \int_0^1 fvdx \\ \Longrightarrow \int_0^1 \sum_{j=1}^{n-1} c_j\psi_j'v'dx &= \int_0^1 \sum_{j=1}^{n-1} c_j \int_0^1 \psi_j'v'dx \\ &= \int_0^1 fvdx \end{aligned}$$

Next, select the test function $v(x)$ as $\psi_1, \psi_2, \ldots, \psi_{n-1}$ successively to generate the system of liners equation, taking into account that additional errors are introduced.

$$\left(\int_0^1 \psi_1'\psi_1' dx\right) c_1 + \cdots + \left(\int_0^1 \psi_1'\psi_{n-1}' dx\right) c_{n-1} = \int_0^1 f\psi_1 dx$$
$$\left(\int_0^1 \psi_2'\psi_1' dx\right) c_1 + \cdots + \left(\int_0^1 \psi_2'\psi_n' dx\right) c_{n-1} = \int_0^1 f\psi_2 dx$$
$$\cdots\cdots\cdots\cdots\cdots\cdots$$
$$\left(\int_0^1 \psi_i'\psi_1' dx\right) c_1 + \cdots + \left(\int_0^1 \psi_i'\psi_{n-1}' dx\right) c_{n-1} = \int_0^1 f\psi_i dx$$
$$\cdots\cdots\cdots\cdots\cdots\cdots$$
$$\left(\int_0^1 \psi_{n-1}'\psi_1' dx\right) c_1 + \cdots + \left(\int_0^1 \psi_{n-1}'\psi_{n-1}' dx\right) = \int_0^1 f\psi_{n-1} dx$$

Or in the matrix-vector form

$$\begin{bmatrix} a(\psi_1,\psi_1) & a(\psi_1,\psi_2) & \ldots & a(\psi_1,\psi_{n-1}) \\ a(\psi_2,\psi_1) & a(\psi_1,\psi_2) & \ldots & a(\psi_2,\psi_{n-1}) \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ a(\psi_{n-1},\psi_1) & a(\psi_{n-1},\psi_2) & \ldots & a(\psi_{n-1},\psi_{n-1}) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} (f,\psi_1) \\ (f,\psi_2) \\ \\ \\ (f,\psi_{n-1}) \end{bmatrix}$$

where:

$$a(\psi_i,\psi_j) = \int_0^1 \psi_i'\psi_j' dx, \qquad (f,\psi_i) = \int_0^1 f\psi_i dx$$

The term $a(u,v)$ is called a bilinear form due to its linearity with respect to each variable (function), while (f,v) is denoted a linear form. if ψ_i are the hat functions. then, in particular we get:

$$\begin{bmatrix} \frac{2}{h} & \frac{-1}{h} & & & & & \\ \frac{-1}{h} & \frac{2}{h} & \frac{-1}{h} & & & & \\ & \frac{-1}{h} & \frac{2}{h} & \frac{-1}{h} & & & \\ & & \vdots & \vdots & \vdots & & \\ & & & \vdots & \vdots & \vdots & \\ & & & & \frac{-1}{h} & \frac{2}{h} & \frac{-1}{h} \\ & & & & & \frac{-1}{h} & \frac{2}{h} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} \int_0^1 f\psi_1 dx \\ \int_0^1 f\psi_2 dx \\ \int_0^1 f\psi_3 dx \\ \vdots \\ \vdots \\ \int_0^1 f\psi_{n-2} dx \\ \int_0^1 f\psi_{n-1} dx \end{bmatrix}$$

5. Solve the system of linear equations to determine the coefficients, and hence obtain the approximate solution:

$$u_h(x) = \sum_{j=1}^{n-1} c_j \psi_j(x)$$

Example 2.10. *We propose to search for an approximat solution to the same previous problems.So, we find* $u(x)$ *in the domain* $\Omega =]1, 2[$ *Satisfying the differential equation:*

$$-u''(x) = \frac{1}{x^2} \quad \dots \qquad (M)$$

With boundary conditions on Ω *:*

$$u(1) = \ln 1 \quad \textit{and } u(2) = \ln 2$$

Solutions:

In this problem, the domain is $1-$*dimentional and there is no time dimension involved.To describe the points in the domain, we only need a Single variable* x*. Therefore, the equation to be solved is an ordinary differential equation.*

The exact solution to this problem is:

$$u(x) = \ln(x)$$

the variational formulation of the problem

We have :

$$V = \Big\{ u \in H^1(\Omega); \quad u(1) = 0, u(2) = \ln 2 \Big\}$$

Let's denote $\Omega =]1, 2[$

Solving the differentiation is equivalent to seeking $u(x) \in V$ *such that, by multiplying the two side in the test function,* $v(x)$ *that satisfying the boundary condition with integration.*

We find: $\int_\Omega -u''(x)v(x)dx = \int_\Omega \frac{1}{x^2} v(x)dx$

this formulation is the variational formulation of the Problem (M)*.*

Another variational formulation Can be obtained by using integration by Parts and the boundary conditions $u(1) = 0$ *and* $u(2) = \ln 2$

So:

$$\int_1^2 u'(x)v'(x)dx - [u'(x)v(x)]_1^2 = \int_1^2 \frac{1}{x^2}\cdot v(x)dx$$
$$= -\int_1^2 u'(x)v'(x)dx - \ln(2)u'(2) = \int_1^2 \frac{1}{x^2}\cdot v(x)dx$$

we have:

$$\psi_i(x) = \begin{cases} \frac{1}{h}x + 1 - i - \frac{a}{h} & x \in [x_{i-1}, x_i[\\ \frac{-1}{h}x + 1 + i + \frac{a}{h} & x \in [x_i, x_{i+1}[\\ 0 \qquad \textit{Other wise} \end{cases}$$

then

$$\psi_i'(x) = \begin{cases} \frac{1}{h} & x \in [x_{i-1}, x_i[\\ \frac{-1}{h} & x \in [x_i, x_{i+1}[\\ 0 \quad \textit{Other wise} \end{cases}$$

we've got

$$u(x) = \sum_{i=1}^{n} c_i\psi_i(x) \Longrightarrow u'(x) = \sum_{i=1}^{n} c_i\psi_i'(x)$$

we take $v = \psi_i$

$$-\int_1^2 u'(x)\psi_i'(x)dx = \int_1^2 \frac{1}{x^2}\psi_i(x)dx$$

$$\Rightarrow -\int_1^2 \left(\sum_{j=1}^{n} c_j\psi_j'(x)\right)\psi_i'(x)dx = \int_1^2 \frac{1}{x^2}\psi_i(x)dx$$

$$\Rightarrow \int_{x_{i-1}}^{x_i} c_{i-1}\psi_{i-1}'(x)\psi_i'(x)dx + \int_{x_{i-1}}^{x_{i+1}} c_i\left(\psi_i(x)\right)^2 dx + \int_{x_i}^{x_{i+1}} c_{i+1}\psi_{i-1}'(x)\psi_i'(x)dx = \int_1^2 \frac{1}{x^2}\psi_i(x)dx$$

$$\Rightarrow \int_{x_i}^{x_{i-1}} -\frac{c_{i-1}}{h^2}dx + \int_{x_{i-1}}^{x_{i+1}} \frac{c_i}{h^2}dx + \int_{x_i}^{x_{i+1}} -\frac{c_{i+1}}{h^2}dx = \int_1^2 \psi_i(x)dx$$

by integration

$$-\frac{c_{i-1}}{h} + \frac{2c_i}{h} - \frac{c_{i+1}}{h} = \int_1^2 \frac{1}{x^2}\psi_i(x)dx$$

by calculate the following integral $\int_1^2 \frac{1}{x^2}\psi_i(x)dx$

$$\int_1^2 \frac{1}{x^2}\psi_i(x)dx = \frac{1}{h}\ln\left(\frac{{x_i}^2}{x_{i+1}x_{i-1}}\right) + \left(\frac{-1}{x_{i+1}} + \frac{1}{x_{i-1}}\right) + i\left(\frac{-1}{x_{i+1}} + \frac{2}{x_i} - \frac{1}{x_{i-1}}\right) = g(x_i)$$

by substitute

$$-\frac{c_{i-1}}{h} + \frac{2c_i}{h} - \frac{c_{i+1}}{h} = g(x_i)$$

So, for

$$
\begin{aligned}
&i = 1 = \frac{-c_0}{h} + \frac{2c_1}{h} - \frac{c_2}{h} = g\left(x_1\right) \\
&i = 2 : -\frac{c_1}{h} + \frac{2c_2}{h} - \frac{c_3}{h} = g\left(x_2\right) \\
&\vdots \\
&\vdots \\
&i = N : \frac{-c_{N-1}}{h} + \frac{2c_N}{h} - \frac{c_{N+1}}{h} = g(x_N)
\end{aligned}
$$

in the matrix form

$$
\begin{bmatrix}
2 & -1 & & & \\
-1 & 2 & -1 & & \\
\ddots & \ddots & \ddots & & \\
 & \ddots & \ddots & \ddots & \\
 & & \ddots & \ddots & -1 \\
 & & & -1 & 2
\end{bmatrix}
\begin{bmatrix}
c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{N-2} \\ c_{N-1}
\end{bmatrix}
=
\begin{bmatrix}
g(x_1) + \frac{c_0}{h} \\ g(x_2) + \\ \vdots \\ \vdots \\ g(x_{N-2}) \\ g(x_{N-1}) + \frac{c_{N+1}}{h}
\end{bmatrix}
$$

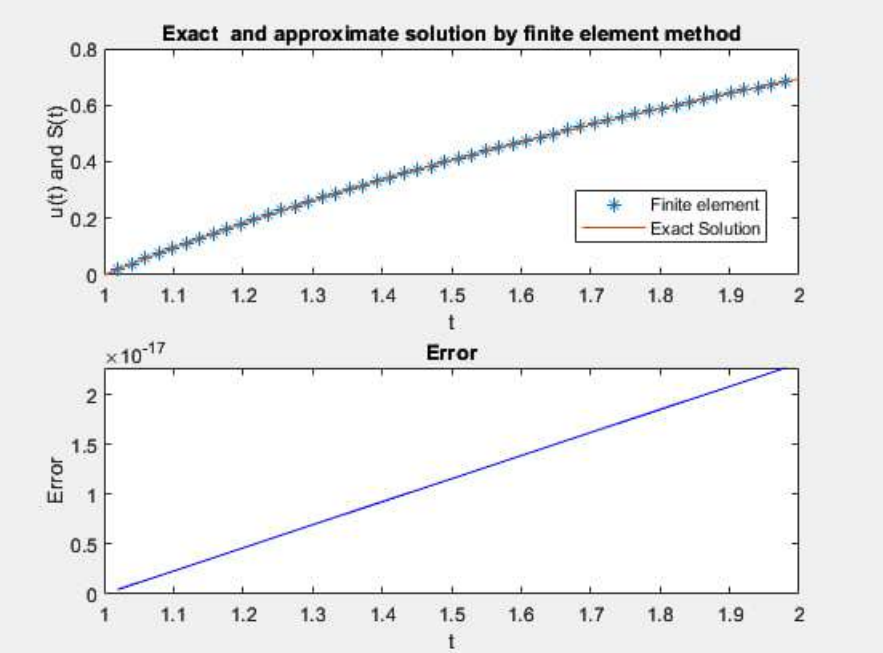
MATLAB code:

```
% finite element method
% y"=f(x,y)
% x=[a b]
%S is the exact solution of y"
clc
clear
N=10;
a=1;
b=2;
c=log(1);
d=log(2);
syms t s
f(t)=1/t^2;
S(t)=log(t);
h=(b-a)/(N+1);
```

```
A=-(diag(-2*ones(1,N),0)+diag(1*ones(1,N-1),-1)+diag(1*ones
    (1,N-1),+1));
for i=1:N
    x(i)=a+i*h;
end
 B(1)=h*int(f(s)*(s/h-a/h),a,x(1))+h*int(f(s)*(-s/h+2+a/h),x
     (1),x(2));
for i=2:N-1
 B(i)=h*int(f(s)*(s/h-(i-1)-a/h),x(i-1),x(i))+h*int(f(s)*(-s/
     h+(i+1)+a/h),x(i),x(i+1));
end
 B(N)=h*int(f(s)*(s/h-N+1-a/h),x(N-1),x(N))+h*int(f(s)*(-s/h
     +(N+1)+a/h),x(N),b)+log(2);
c=inv(A)*B';
for i=1:N
     er(i)=abs(S(x(i))-c(i));
end
subplot(2,1,1)
plot(x,c,'*')
hold on
fplot(S,[a b])
title('Exact  and approximate solution by finite element
    method')
xlabel('t')
ylabel('u(t) and S(t)')
legend('Finite element','Exact Solution')
hold off
subplot(2,1,2)
plot(x,er,'b')
title('Error')
xlabel('t')
```

```
42 ylabel('Error')
```

Program results:



Format(2-9)

3 Solving higher order differential equations

3.1 Taylor series method

The method used to solve a first-order differential equation using Taylor expansion is essentially the same as when dealing with higher-order equations, we will illustrate this through the following example:

Example 3.1. *Solve the following boundary value problem* $y'' = y - 3x$ *at* $x = 0.1$ *given that* $y(0) = 1$ *and* $y'(0) = 4$

Solution:

We begin with the calculation of higher differentials of the differential equation given as the following:

$$y'' = y - 3x$$
$$y''' = y' - 3$$
$$y^{(4)} = y''$$

The next step is to compute the previous derivatives at the point of the expansion, considering the note we mentioned in the example about the first-degree Taylor expansion, specifically regarding the point of the expansion.

$$y(0) = 1 \qquad condition$$
$$y'(0) = 4 \qquad condition$$
$$y''(0) = 1$$
$$y'''(0) = 1$$
$$y^{(4)}(0) = 1$$

Take:

$$h = x - x_0 = 0.1 - 0.0 = 0.1$$

By substituting the equations' images at the expansion point into the given Taylor series expansion equation(2.13).

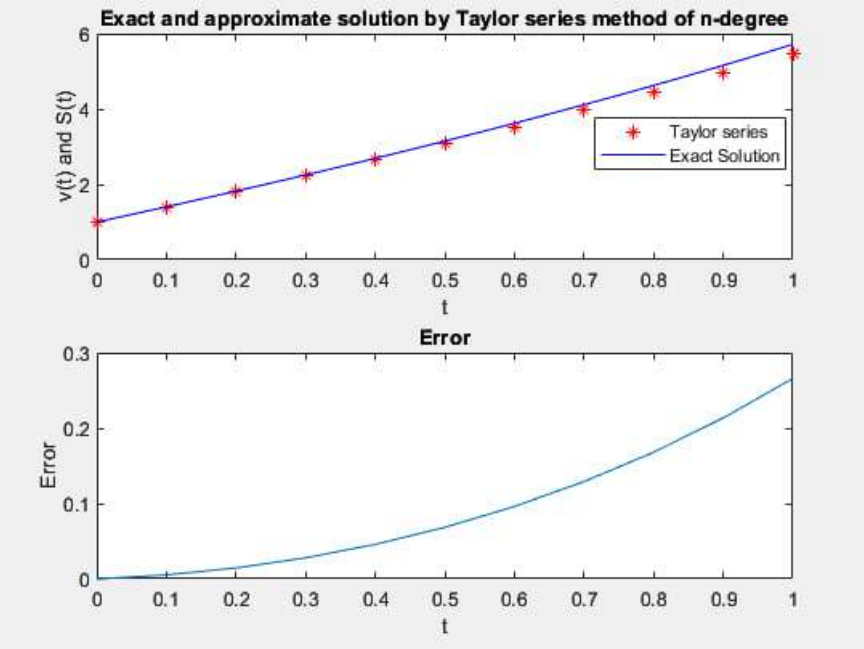
$$f(x) = f(x_0) + \frac{h}{1!}f'(x_0) + \frac{h^2}{2!}f''(x_0) + \frac{h^3}{3!}f^{(3)}(x_0) + \dots$$
$$f(x) = 1 + \frac{0.1}{1!}(1) + \frac{(0.1)^2}{2!}(4) + \frac{(0.1)^3}{3!}(1) + \frac{(0.1)^4}{4!}(1)\dots$$
$$= 1.1814$$

MATLAB code:

```
%Taylor series method n-degree
%f(t,y)=y"
%x=[a b]
%v(1)=y(a)and u(1)=y'(a)
%S is the Exact Solution of y"
clc
clear
syms t y
f(t,y)=y-3*t;
S(t)=exp(t)+3*t;
v(1)=1;
u(1)=4;
a=0;
b=1;
x=[a:0.1:b];
n=length(x);
h=x(2)-x(1);
g(t,y)=f(t,y)+h*diff(f,t,1)+(h^2/2)*diff(f,t,2)+(h^3/6)*diff(
   f,t,3)+(h^4/24)*diff(f,t,4);
for i=2:n
    v(i)=v(i-1)+h*u(i-1);
```

```
21      u(i)=u(i-1)+h*g(x(i-1),v(i-1));
22       er(i)=abs(S(x(i))-v(i));
23  end
24  subplot(2,1,1)
25  plot(x,v,'r*')
26  hold on
27  fplot(S, [a b], 'b')
28  title('Exact and approximate solution by Taylor series method
        of n-degree')
29  xlabel('t');
30  ylabel('v(t) and S(t)')
31  legend('Taylor series','Exact Solution')
32  subplot(2,1,2)
33  plot(x,er)
34  title('Error')
35  xlabel('t');
36  ylabel('Error')
```

Program results:



Format(3-1)

3.2 Euler method

When looking at Euler's formula for solving a first-order differential equation, we find that it relates the values of the dependent variable at two consecutive points with the first derivative present in the differential equation we aim to solve. This observation marks the starting point in solving higher-order differential equations, where appropriate substitution is made to reduce the order of the equation until it becomes first-order. Then, we can directly use Euler's general formula for solving it. This will become clearer through the following example.

Example 3.2. *Given*

$$y'' = y - 3x \tag{3.1}$$

with

$$y'(0) = 4, \quad y(0) = 1 \tag{3.2}$$

obtain $\frac{dy}{dx}$ *at* $x = 0.6$, 0.8 *and* 1. *hint take* $h = 0.1$

Solution:

When looking at the given differential equation that needs to be solved, we find that it is a second-order equation. To solve it, we need to reduce the equation's order to first-order. Additionally, the given initial condition in terms of the first derivative needs to be transformed into a value for the new dependent variable instead of the value of the derivative of the dependent variable. This transformation can be achieved through the following substitution:

$$u = y' \implies u' = y'' \tag{3.3}$$

by substituting equation (**??**) into the given differential equation and condition, it takes the following form:

$$\begin{cases} u' = y - 3x \\ u(0.40) = 0.41 \end{cases} \tag{3.4}$$

the next step would be to write down the Euler's equation corresponding to the differential equation in equation(**??**)

$$u_{n+1} = u_n + hf(x_n, u_n) = u_n + h\left(u_n - 3x_n\right) \tag{3.5}$$

The final step is to perform the some calculations, was discussed in chapter 2 to explain this method at specific points, and the results one shown in the following table.

n	x_n	u_n	u_{n+1}
0	0.0	4.0000	4.0700
1	0.1	4.0700	4.2470
2	0.2	4.2470	4.3429
3	0.3	4.3429	4.4575
4	0.4	4.4575	4.5855
5	0.5	4.5855	4.7282
6	0.6	4.7282	4.8866

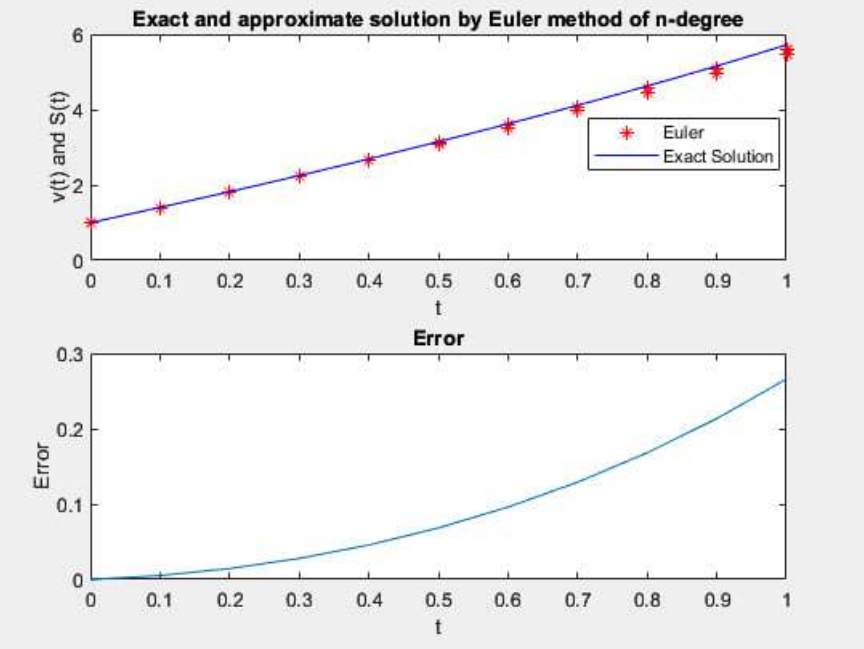
MATLAB code:

```
1        %Euler's method of n-degree
```

```
%f(t,y)=y"
%x=[a b]
%v(1)=y(a) and u(1)=y'(a)
%S is the Exact Solution of y"
clc
clear
syms t y
f(t,y)=y-3*t;
S(t)=exp(t)+3*t;
v(1)=1;
u(1)=4;
a=0;
b=1;
x=[a:0.1:b];
n=length(x);
h=x(2)-x(1);
for i=2:n
    v(i)=v(i-1)+h*u(i-1);
   u(i)=u(i-1)+h*f(x(i),v(i-1));
    er(i)=abs(S(x(i))-v(i));
end
subplot(2,1,1)
plot(x,v,'r*')
hold on
fplot(S, [a b], 'b')
title('Exact and approximate solution by Euler method of n-
   degree')
xlabel('t');
ylabel('v(t) and S(t)')
legend('Euler','Exact Solution')
subplot(2,1,2)
```

```
32 plot(x,er)
33 title('Error')
34 xlabel('t');
35 ylabel('Error')
```

Program results:



Format(3-2)

3.3 Modified Euler method

Solution Strategy

The general approach remains constant, but the main idea lies in utilizing an appropriate substitution. This substitution is employed to reduce the order of the equation, thereby converting it to a first-order equation. Subsequently, we gain the ability to apply the general relationship for the modified euler method.

Example 3.3. *Given*

$$y'' = f(x, y) = \ y - 3x \tag{3.6}$$

with

$$y'(0) = 4, \quad y(0) = 1 \tag{3.7}$$

obtain $\frac{dy}{dx}$ *at* $x = 0.6$, 0.8 *and* 1. *hint take* $h = 0.1$

Solutions

The current example involves a second-order ordinary differential equation, and since we will utilize the modified euler method, this will be achieved through the use of an appropriate substitution.This substitution will reduce the equation's order, converting it from second-order to first-order. The substitution is as follows:

$$u = y' \Longrightarrow u' = y'' \tag{3.8}$$

By reformulating the differential equation and the accompanying condition in terms of the new variable, it takes the following form:

$$\begin{cases} u' = \ y - 3x \\ u(0) = 4 \end{cases} \tag{3.9}$$

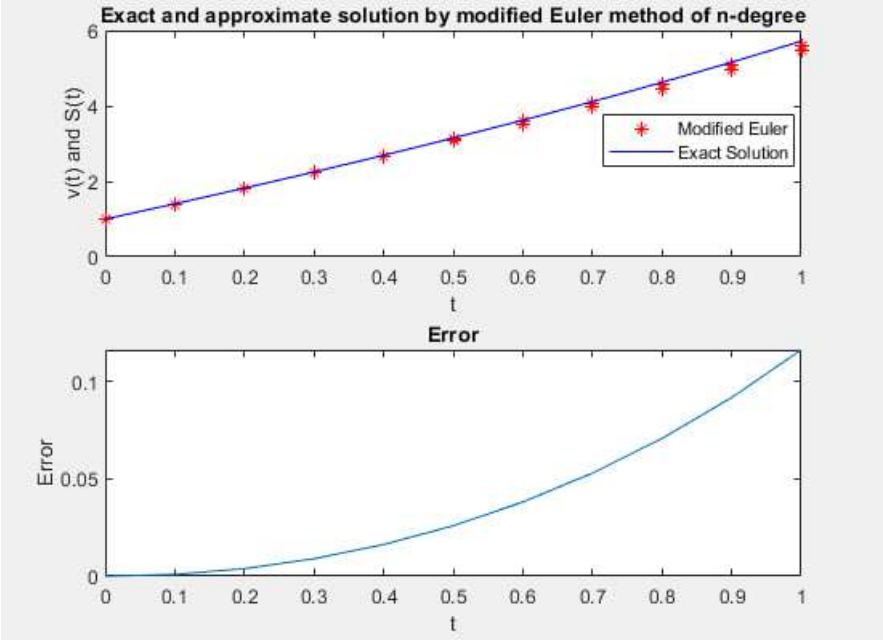
Therefore, the given differential equation takes the following new form in terms of the new variable. The steps to solve this issue, which we discussed in the second chapter to explain this method at the specified points.

MATLAB code:

```
1         %Modified Euler method of n_degree
2 %y"=f(t,y)
3 %x=[a b]
4 %v(1)=y(a) and u(1)=y'(a)
5 %S is the Exact Solution of y"
6 clc
7 clear
8 syms t y
9 f(t,y)=y-3*t;
10 S(t)=exp(t)+3*t;
11 v(1)=1;
12 u(1)=4;
13 a=0;
```

```
b=1;
x=[a:0.1:b];
n=length(x);
h=x(2)-x(1);
for i=2:n
   u(i)=u(i-1)+h/2*(f(x(i-1)+h,v(i-1))+(f(x(i-1),v(i-1))));
   v(i)=v(i-1)+h/2*(u(i-1)+u(i));
   er(i)=abs(S(x(i))-v(i));
end
subplot(2,1,1)
plot(x,v,'r*')
hold on
fplot(S, [a b], 'b')
title('Exact and approximate solution by modified Euler
   method of n-degree')
xlabel('t')
ylabel('v(t) and S(t)')
legend('Modified Euler','Exact Solution')
subplot(2,1,2)
plot(x,er)
title('Error')
xlabel('t')
ylabel('Error')
```

Program results:



Format(3-3)

4 Application of ordinary differential equations

Applications of differential equations extend to a wide range of fields, from physics and engineering to chemistry, computer science, biology, and economics.These equations describe changes in time and space and are used to understand natural phenomena and predict their behavior.

4.1 The first model

The basic equation governing the amount of current I (in amperes) in a simple RL circuit **fig(4-1)** consisting of a resistance R (in ohms), an inductor L (in henries), and an electromotive force (abbreviated emf) E (in volts) is

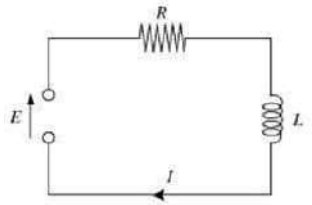
$$\frac{dl}{dt} + \frac{R}{L}I = \frac{E}{L} \tag{4.1}$$

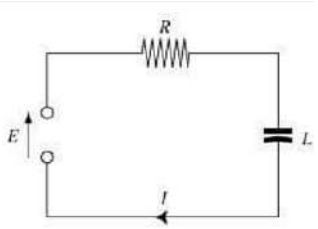
For an RC circuit consisting of a resistance, a capacitance C (in farads), an emf, and no inductance **fig(4-2)** the equation governing the amount of electrical charge q (in coulombs) on the capacitor is

$$\frac{dq}{dt} + \frac{1}{RC}q = \frac{E}{R}$$

The relationship between q and I is

$$I = \frac{dq}{dt}$$





fig(4-1) **fig(4-2)**

Example 4.1. *An RL Circuit has an emf of 5 volts, a resistance initial of* 50 *ohms, an inductance of 1 henry, and no initial current. find the current in the circuit at any time t here* $E = 5$, $R = 50$, *and* $L = 1$

$$\begin{cases} \dfrac{dI}{dt} + 50I = 5 \\ I(0) = 0 \end{cases}$$

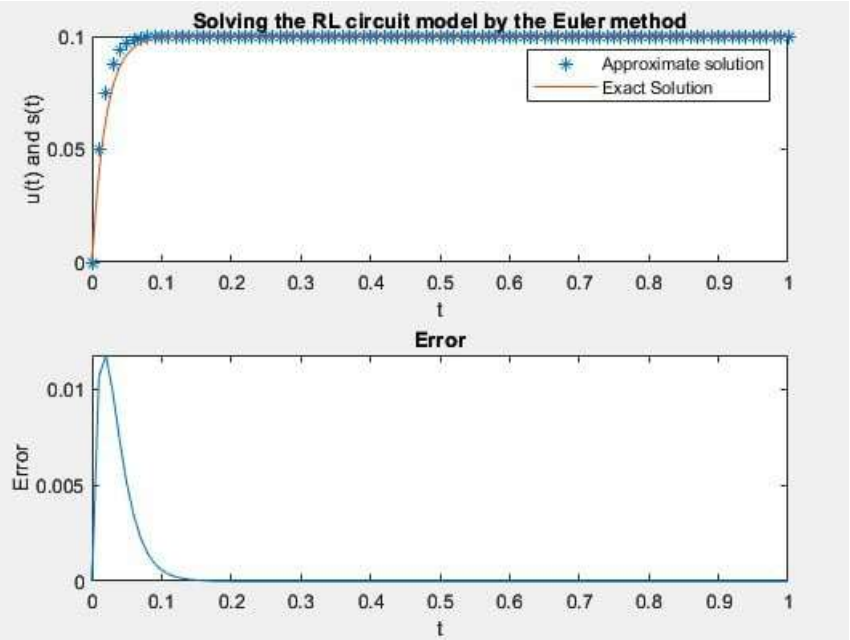
its solution is $I = -\dfrac{1}{10}e^{-50t} + \dfrac{1}{10}$

MATLAB code:

```
        % RL circuit model
%Remark:in order to converge,h must be small enough
clc
clear
syms t y
f(t,y)=5-50*y;
S(t)=(1/10)*(-exp(-50*t)+1);
alpha=0;
a=0;
b=1;
N=100;
h=(b-a)/N;
x(1)=a;
u(1)=alpha;
for i=2:N+1
```

```
    x(i)=a+(i-1)*h;
    u(i)=u(i-1)+h*f(x(i-1),u(i-1));
    er(i)=abs(S(x(i))-u(i));
end
subplot(2,1,1)
plot(x,u,'*')
hold on
fplot(S,[a b])
title('Solving the RL circuit model by the Euler method')
xlabel('t');
ylabel('u(t) and s(t)');
legend('Approximate solution', 'Exact Solution');
subplot(2,1,2)
plot(x, er)
title('Error')
xlabel('t')
ylabel('Error')
```

Program results:



Format(4-1)

4.2 The second model

Introduction

The objective of this model is to study the prediction of the growth of a non-vascularized lung tumor (the first stage of cancer), before and during chemotherapy treatment. This model will be represented by a system of ordinary differential equations that describe, among other things, tumor volume and proliferating cells.

Tumor angiogenesis model:

In the context of our work, we have mainly considered formulations based on ODEs, which only describe the dynamics of cancer growth. In this context, the tumor growth model by Hahnfeldt et al. is generally preferred. Indeed, the formulation by Hahnfeldt et al. is based on experimental observations. The tumor angiogenesis model according to Hahnfeldt et al. is formalized as follows. Let: C(t) be the volume of cancer cells, and E(t) be the volume of endothelial cells that supply the tumor with oxygen and nutrients. The tumor angiogenesis evolution is described by the following system of ODEs.

$$\begin{cases} \dfrac{dc}{dt} = -\lambda_c c(t) \log\left(\dfrac{c(t)}{E(t)}\right) \\ \dfrac{dE}{dt} = bc(t) - dc(t)^{\frac{2}{3}} E(t) \end{cases} \tag{4.2}$$

λ_c: The tumor growth rate

b: The birth rate of vascular endothelial celles.

d: The death rate of endothelial celles.

In particular, the tumor follows a Gompertzian function implying that its growth saturates at a maximum volume. The birth (b) and death (d) rates of endothelial cells primarily depend on the type of tumor and the patient.

Example 4.2. *The original model of physiological data by Hahnfeldt et al. was initially established to describe the vascular stage of the tumor. The authors were able to validate their formulation after observing Lewis lung carcinoma in mice . They then identified the parameters c, b, and d of their model from experimental data. The identified values are summarized in the following table:*

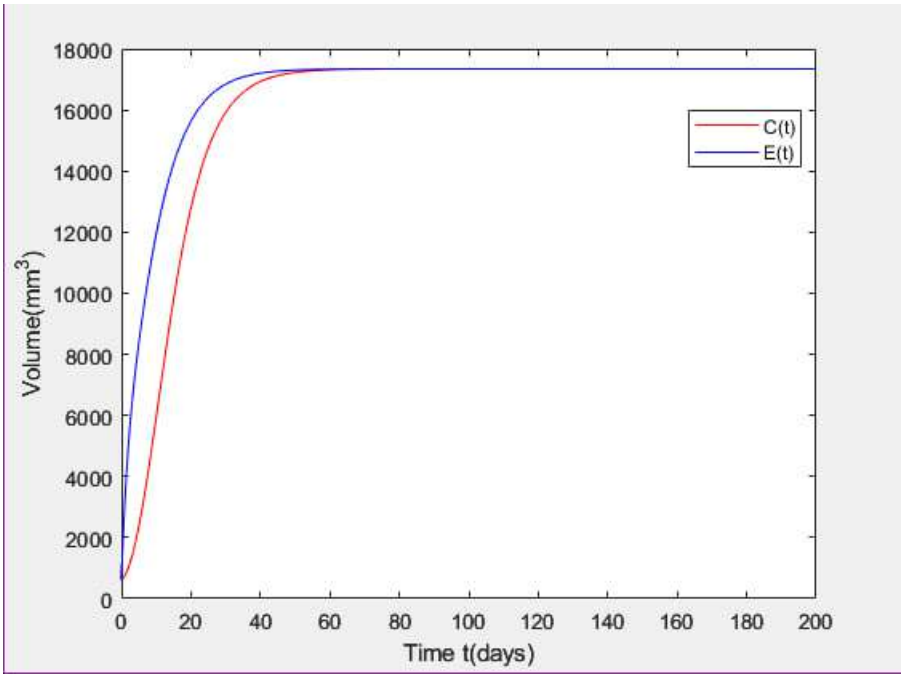
parameters	*Values*	*Unit*
λc	$\frac{0.192}{log(10)}$	$1/day$
b	0.00873	$1/(day.mm^2)$
d	5.85	$1/day$

Parameters of tumor growth by angiogenesis according to the observations of Hahnfeldt et al. of a tumor in mice.

MATLAB code

```
% Tumor angiogenesis model
%Remark:in order to converge,h must be small enough
clc
clear
lambdac=0.1921;
b=5.85;
d=0.00873;
t=0:0.001:200;
h=t(2)-t(1);
n=length(t);
y0=[625,625];
y(1,:)=y0;
f=@(t,y)[(-lambdac)*y(1)*log(y(1)/y(2));b*y(1)-d*y(1)^(2/3)*y
   (2)];
for i=1:n-1
    k1=h*f(t(i),y(i,:))';
    k2=h*f(t(i)+0.5*h,y(i,:)+0.5*k1)';
    k3=h*f(t(i)+0.5*h,y(i,:)+0.5*k1)';
    k4=h*f(t(i)+h,y(i,:)+k3)';
    y(i+1,:)=y(i,:)+(1/6)*(k1+2*k2+2*k3+k4);
end
plot(t, y(:,1), 'r')
hold on
plot(t, y(:,2), 'b')
```

```
24 hold off
25 xlabel('Time t(days)')
26 ylabel('Volume(mm^3)')
27 legend('C(t)','E(t)')
```



Format(4-2)

Conclusion

This study aims to explore and analyze the use of Ordinary Differential Equations (ODEs) in numerical computation using the MATLAB environment. By applying advanced concepts and techniques in MATLAB, the mathematical and practical foundations for solving a variety of engineering and physics problems using ODEs have been elucidated.

The accuracy and efficiency of several methods for solving ODEs in MATLAB have been analyzed, including numerical integration techniques such as the Euler method and the Runge-Kutta method, in addition to using analytical solutions for differential equations. The results have shown the impact of the time step and method selection on the accuracy and efficiency of the solution.

Based on the obtained results, the importance of using MATLAB in analyzing and solving ordinary differential equations can be emphasized. Moreover, attention can be directed towards exploring and developing new techniques to improve the performance and accuracy of ODE solutions using MATLAB.

Bibliography

[1] B. Ahlem, B. Cheima, F. Boutheina, T. Souad, *mathématique du cancer et ses traitements*, Université Fréres Mentouri Constantine, 2019.

[2] D. Bachir, *Outils de programmation 2*, Université Echahid Hamma Lakhdar El-Oued, 2018.

[3] D. Nadjet, *Méthodes numériques pour EDO et EDP*, Université Echahid Hamma Lakhdar El-Oued, 2021.

[4] H. Adalet, *Méthode Numériques pour les ODEs et EDP*, Univesité de Batna 02, 2020.

[5] K. Hayder, *MATLAB(The Language of Technical Computing)*, Mathematics, 2002.

[6] R. A. Mohamed, *Applications géomètriques de MATLAB*, Université Appliquée de Balqa, 2010.

[7] R. Bronson, G. B. Costa, *Differential Equations*, Schaum's outline series, Doi: 10.1036/0071456872.

[8] R. L. Burden, J. D. Faires, *Numerical Analysis*, cengage learning.

[9] S. J. Ahmed, *Equations differentielles Ordinaires*, Maison de livre universitaire, E.A.U. , 2012.

[10] Y. Baziz, *Cours Outils de programation pour les mathématiques*, Centre Universitaire Relizane, 2014.

[11] Z. Li, Z. Qiao, T. Tang, *Numerical Solution of Differential Equations*, Cambridge University Press, doi: 978-1-107-16322-5.

[12] Z. Li, Z. Qiao, T. Tang, *Numerical Solution of Differential Equations(Introduction to Finite Difference and Finite Element Methods)*, Prelims, 2017.

Résumé

Dans cette thèse intitulée "Résolution des équations différentielles ordinaires à l'aide de MAT- LAB", nous visons à fournir une étude complète et organisée. Nous avons commencé par aborder les fondamentaux de MATLAB, puis nous avons étudié les solutions numériques pour les équations différentielles ordinaires de différents degrés à l'aide de MATLAB. Tout au long de cette étude, nous avons mis l'accent sur la simplification de la compréhension et sur l'explication des méthodes considérées comme plus précises dans la résolution. Enfin, nous avons documenté des applications pratiques, notamment l'étude des circuits RL et un modèle de croissance des vaisseaux sanguins dans les tumeurs.

Mots-clés : équations différentielles - solutions numériques aux équations différentielles

ملخص

حاولنا في هذه المذكرة المعنونة بـ ' حل معادلات التفاضل العادية باستخدام MATLAB ' إلى تقديم دراسة شاملة ومنظمة. بدأنا بالتطرق إلى أساسيات MATLAB ومن ثم قمنا بدراسة الحلول العددية لمعادلات التفاضل العادية من الدرجات المختلفة باستخدام خلال هذه الدراسة، ركزنا على تبسيط الفهم وتوضيح الطرق التي تعتبر أكثر دقة في الحل، وفي النهاية، قمنا بتوثيق ومن تطبيقات عملية تشمل دراسة الدارة RL ونموذج لنمو الأوعية الدموية في الأورام.

كلمات المفتاحية: المعادلات التفاضلية – حلول عدادية للمعادلات التفاضلية

.Abstract

In this thesis titled "Solving Ordinary Differential Equations Using MATLAB", we aim to provide a comprehensive and organized study. We started by covering the fundamentals of MATLAB, then proceeded to study numerical solutions for ordinary differential equations of various degrees using MATLAB. Throughout this study, we focused on simplifying understanding and elucidating the methods that are considered more precise in solving. Finally, we documented practical applications including the study ofRL circuits and a model for the growth of blood vessels in tumors.

Keywords: differential equations - numerical solutions to differential equations