Motivation for Numerical Methods

- Sometimes an ODE may not be in one of the forms discussed in this course, so it might be difficult to find the exact solution.
- Sometimes an exact solution might not even exist.
- In such cases, a numerical approximation for the solution may be obtained.
- There are many different types of numerical methods to choose from (e.g. Euler's method, Modified Euler's method, Runge's method, Runge-Kutta methods).
- We will focus on the use of Euler's method.

Existence and Uniqueness Theorem

- Consider the 1st order IVP

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0.$$

If f and $\frac{\partial f}{\partial y}$ are continuous in the closed rectangle

$$R = \{(x, y) : |x - x_0| \le a, \ |y - y_0| \le b\}$$

then $\exists$ a unique solution $y(x)$ of the given IVP in an interval $|x - x_0| \le h \le a$.

Example

- Consider the IVP

$$\frac{dy}{dx} = \frac{2y}{x}, \qquad y(x_0) = y_0.$$

- $f(x,y) = \frac{2y}{x}$ and $\frac{\partial f}{\partial y} = \frac{2}{x}$. Both are undefined for $x = 0$.
- Using separation of variables, the general solution is

$$y(x) = Ax^2.$$

- The IVP has
  - a unique solution in an open interval containing x_0 if $x_0 \neq 0$,
  - no solution if $x_0 = 0$ and $y_0 \neq 0$,
  - infinite solutions if $x_0 = 0$ and $y_0 = 0$.

MATLAB Review

Starting MATLAB

- Run **MATLAB R2017a** from the Desktop
- The different MATLAB windows:
  - **Command Window** (centre): This is where you type commands.
  - **Workspace** (right): All variables currently stored are displayed here.
  - **Current Folder** (left): Lists files stored in the current directory.
  - **Command History** (accessed through the Layout tab in the Home menu): Stores all commands entered.
  - **Graphics Window** (pop-up): Displays plots/graphs.
  - **Editor Window**: Used to create/edit MATLAB codes.

MATLAB as a Calculator

- Arithmetic Operations

Addition	+	
Subtration	-	
Multiplication	*	.*
Division	/	./
Exponentiation	^	.^

- Unless otherwise specified, MATLAB assigns computed values to a variable called **ans**
- Variables can be assigned values using $=$
- ".*", "./" and ".^" are element-wise operations.

Built-In Constants and Functions

Name	**MATLAB syntax**	**Remarks**
π	pi	
10^x	1ex	Powers of 10
$\sqrt{x}$	sqrt(x)	Square root function
e^x	exp(x)	Exponential function
$\log(x)$	log10(x)	log base 10
$\ln(x)$	log(x)	log base e
$\sin(x)$	sin(x)	Measured in radians
	sind(x)	Measured in degrees
$\arcsin(x)$, $\sin^{-1}(x)$	asin(x)	Inverse trig functions
$\sinh(x)$	sinh(x)	Hyperbolic functions

MATLAB as a Calculator - Examples

- Click in the Command Window and type each of the following commands:

```
>> 3+4*(1+6/3)

>> pi^2-sqrt(ans)+cos(45)/cosh(45)

>> x=exp(3)*log10(3)+atan(3)/log(3)

>> y=20+log(csch(x-1)^2)

>> z=1+y/x

>> z*3e-2

>> ans
```

Variables

- A variable name may contain only letters, digits, and underscores, and must begin with a letter.
  - Punctuation marks are not allowed!
  - **Note**: MATLAB is case-sensitive!
- A variable can be assigned a numerical value, an array of numerical values or a string of characters.
- Use single quotation marks ' ' around any text that is to be entered as a character string.
- A variable cannot be given the same name as a MATLAB keyword, and also should not be given any name that already exists in MATLAB.

Useful Commands and Tools

- ***clear*** ***x*** ***y*** - deletes only variables **x** and **y**
- ***clear*** - deletes all current variables in the workspace
- ***clc*** - clears the Command Window
- ***quit*** - ends the session and closes MATLAB
- Hold down **Control** + **C** on the keyboard - this interrupts the current command execution, which is often useful for aborting commands that are taking too long to complete.
- While the Command Window is active, use the **Up** and **Down** arrow keys on the keyboard to access commands stored in the Command History.

Useful Commands and tools

- A semicolon (;) is used to suppress the output - the output is still stored in the Workspace, but it is not displayed in the Command Window.
- An ellipsis (...) is used to enter input on an additional line - this is useful when typing long lines of MATLAB code.
- % is used for commenting. MATLAB does not run any commands appearing after a % in a line - this is useful for documenting and explaining what your code does.
- ***help*** - lists all the help topics
- ***help*** ***xyz*** - provides help on topic xyz
- Note: Online support for MATLAB can be found at: www.mathworks.com/help/matlab/

Formatting the Command Window Display

- MATLAB uses double-precision floating point arithmetic - values are stored accurate to 15 decimal places.
- However, MATLAB displays only 4 decimal places by default.
  - To display 15 decimal places, type **format long**
  - To display only 4 decimal places, type **format short** (default)
  - To display numbers as fractions, type **format rat**
- By default, MATLAB displays a blank line between each line of text in the Command Window.
  - To disable this feature, type **format compact**
  - To enable it, type **format loose**

Inputting Arrays

- **Row vector** >> x=[0 1 2 3] or x=[0,1,2,3]
- **Column vector** >> y=[0;1;2;3]
- **Matrix** >> Z=[0 1 2 3; 4 5 6 7]
- Other ways of entering row vectors
  >> x=1:2:6
  >> x=linspace(1,6,3)
- **Special Matrices**
  >> zeros(m,n) - zero matrix
  >> ones (m,n) - matrix of ones
  >> eye(m,n) - ones on leading diagonal; zeros elsewhere

Indexing

- You can ask MATLAB for the number stored in row 2, column 3 of a matrix
  >> Z(2,3)
- Or for all the elements stored in row 2
  >> Z(2,:)
- Or for all the elements stored in column 3
  >> Z(:,3)
- Or change the value stored in a position
  >> Z(2,3)=2

Matrix Operations

- Elementwise addition or subtraction (must be of the same dimensions)
  >> **A+B** or **A-B**
- Multiplication (must have the same inner dimensions)
  >> **A*B**
- Element-wise multiplication or division
  >> **A.*B** or **A./B**
- Element-wise powers
  >> **A.^m**

Matrix Operations

- Transpose of a matrix Z (denoted Z^T)
  >> **Z'**
- Dimensions of a matrix Z
  >> **size(Z)**
- Number of elements in a matrix Z
  >> **numel(Z)**
- Length of a vector x
  >> **length(x)**

Function Handles - Self Reading

- A function handle is a data class that stores an association to a function.
- It allows you to indirectly call a function by using its associated handle rather than the actual function name.
- Function handles are entered in the format
  fun_handle=@fun_name

  ```
  >> f=@sqrt
  >> f(9)
  >> g=@cos
  ```
- Function handles are treated as scalars (or 1×1 arrays).

Anonymous Functions - Self Reading

- MATLAB has several built-in functions (e.g. cosine, square root, etc.)
- We can also program user-defined functions using
  - Function files (discussed later)
  - **Anonymous function** definitions using function handles
- An anonymous function is one that does not explicitly exist in MATLAB, but is associated with a function handle.

```
>> dotsquare=@(x) x.^2
>> dotsquare(1:5)
>> z=@(x,y) x.^2+y.^2
>> z(1:5,[2 3 5 7 11])
```

2D Plots

- Plotting a curve of y versus x
  >> **plot(x,y)**

- Adding an x- or y-axis label
  >> **xlabel('text'); ylabel('text')**

- Inserting a title
  >> **title('text')**

- Example:
  >> **t=linspace(-pi,pi); u=t.^2;**
  >> **plot(t,u)**
  >> **xlabel('t'); ylabel('u')**
  >> **title('Plot of u versus t')**

2D Plots

- You can change the line color, line style or marker style
  >> **plot(x,y,'m+-')** produces a solid magenta curve with plus sign markers

Line Color		Line Style		Marker Style	
yellow	y	solid (default)	-	plus	+
magenta	m	dashed	- -	circle	o
cyan	c	dotted	:	asterisk	*
red	r	dash-dot	-.	point	.
green	g			cross	x
blue	b			square	s
black	k			diamond	d

2D Plots

- You can plot in different figures by calling a figure before plotting
  >> **figure(2)**
- You can overlay multiple plots on the same graph by using the hold command
  >> **plot(x,y)**
  >> **hold on**
  >> **plot(x,z,'g')**
  >> **hold off**
- Or by using the plot command
  >> **plot(x,y,x,z,'g')**

Script Files

- A **script** is a file containing a valid set of MATLAB commands.
- Scripts are used to type multiple command lines which can then all be run at once.
- A script can be run by
  - typing its name in the Command Window and pressing enter, or
  - clicking the Run icon in the Editor window.
- The script must be saved, and the folder in which the file is saved must be displayed in the "Current Folder" window for the script to run.
- Variables created when a script is run are saved in the Workspace (referred to as gobal variables).

Naming Script Files

- A script name
  - must begin with a letter
  - can contain only letters, digits, and underscores
  - **punctuation marks are not allowed!**
  - should be no longer than 63 characters
- Never give a script file the same name as
  - one of the variables it computes, or
  - a built-in function.
- The **exist** command can be used to check if a name, say "**xyx**", can be used for a script.
  - Type **exist('xyz')** in the Command Window.
  - If MATLAB returns 0 then you can use the name.

Function Files

- A function file takes user-defined inputs each time it is run.
- The first line in a function file must be of the form:

  function [out1,out2,...] = fun_name(in1,in2,...)
- out1,out2,... are the names of the output variables. The square brackets can be omitted if there is only one output variable.
- in1,in2,... are the names of the input variables. The values of these variables must be specified each time the function is run.
- The same rules for naming script files also apply to function files. Additionally, the file must be saved with the same name as the function, i.e. fun_name.m, or else it will not run.

Function Files

- Variables used within a function file are not stored in the Workspace; only the output variables are stored.
- A function can be called with a command of the form

>> [out1,out2,...] = fun_name(in1val,in2val,...)

- in1val,in2val,... are the values specified for the input variables.
- If "[out1,out2,...]" is omitted, then only the first output is stored in the variable **ans**.
- The **feval** command is another way to evaluate functions

  >> **feval(fun,x1,x2,...,xn)** - evaluates the function **fun** with input arguments **x1**, **x2**, ..., **xn**. This is equivalent to **fun(x1,x2,...,xn)**

Control Structures

- Loops are used to perform a series of commands as many times as necessary.
  - A stopping criterion determines when a loop stops.
- Some other control structures perform conditional checks and execute only the commands associated with certain conditions.

for Loop

- Used to execute a series of commands a fixed number of times.
- Basic format:

```
for i=a:k:b
        commands to be executed in each loop
end
```

- Example:

```
function S=sums(n)
S=0;
for k=1:n
        S=S+k;
end
end
```

while Loop

- Execute a series of commands while a specified condition is true.
- Basic format:

```
while expression
        commands to be executed in each loop
end
```

- Example:

```
function [S,n]=sums2(x)
S=0; n=0;
while S<x
        S=S+rand(1);
        n=n+1;
end
end
```

if Statements (if-elseif-else)

- An **if** statement evaluates logical expressions and executes the specified commands when the answer is true.
- Basic format:

if *expression1*
    commands to be executed if expression1 is true
elseif *expression2*
    commands to be executed if expression2 is true
else
    commands to be executed otherwise
end

Relational and Logical Operators

Relational Operators		**Logical Operators**	
equal to	==	and	&
greater than	>	or	|
less than	<	not	~
greater than or equal to	>=		
less than or equal to	<=		
not equal to	~=		

if Statements (if-elseif-else)

- Example:

```
x=-3:0.1:3;
y=zeros(1,length(x));
for i=1:length(x)
    if x(i)>-1 & x(i)<1
        y(i)=x(i)^2+1;
    else
        y(i)=2;
    end
end
plot(x,y)
```

Euler's Method

Derivation of Euler's Method

- Consider a general IVP

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0, \qquad x \in [a, b]$$

that satisfies the existence and uniqueness criteria.

- We can approximate $\frac{dy}{dx}$ using

$$\frac{dy}{dx} = \lim_{h \to 0} \frac{y(x+h) - y(x)}{h} \approx \frac{y(x+h) - y(x)}{h}.$$

- Therefore

$$y(x+h) \approx y(x) + h \cdot f(x, y) \tag{1}$$

Derivation of Euler's Method

- Let's discretize the interval $[a, b]$ into N subintervals.
- The total number of points is $N + 1$
- The size of each subinterval is $h = \frac{b-a}{N}$
- Denote the points $x_j = a + h \cdot j$, for $j = 0, 1, 2, \ldots, N$

$$x_0 = a,$$
$$x_1 = a + h,$$
$$x_2 = a + 2h,$$
$$\vdots$$
$$x_{N-1} = a + (N-1) \cdot h,$$
$$x_N = b$$

Derivation of Euler's Method

- Taking $x = x_j$ in equation (1) gives

$$y\left(x_j + h\right) \approx y\left(x_j\right) + h \cdot f\left(x_j, y\left(x_j\right)\right)$$

- Since $x_j = a + h \cdot j$ we have

$$y\left(a + hj + h\right) \approx y\left(x_j\right) + h \cdot f\left(x_j, y\left(x_j\right)\right)$$

$$\therefore \quad y\left(a + h(j+1)\right) \approx y\left(x_j\right) + h \cdot f\left(x_j, y\left(x_j\right)\right)$$

$$\therefore \quad y\left(x_{j+1}\right) \approx y\left(x_j\right) + h \cdot f\left(x_j, y\left(x_j\right)\right)$$

Algorithm for Euler's Method

$$y\left(x_{j+1}\right) \approx y\left(x_j\right) + h \cdot f\left(x_j, y\left(x_j\right)\right), \quad j = 0, 1, \ldots N-1 \quad (2)$$

- h is the step-size
- $N+1$ is the number of discretized points in the interval $[a, b]$
- $x_0 = a$, $x_1 = a + h$, $x_2 = a + 2h$, ...
  $x_{N-1} = a + (N-1) \cdot h$ and $x_N = b$
- Since we have $y\left(x_0\right) = y_0$, we can solve for $y\left(x_1\right)$, then use this to solve for $y\left(x_2\right)$ and so on.
- Thus, we obtain approximate values for y at the discretized values of x.
- This method is sometimes too slow (if h is small) or too inaccurate (if h is large).

Example

- Consider the IVP

$$\frac{dy}{dx} = x + y - 1, \qquad y(0) = 2, \qquad x \in [0, 0.75]$$

- The exact solution is

$$y_{exact} = 2e^{x} - x.$$

- Let's apply Euler's Method to find an approximate solution.
- Take $N = 3$ subintervals.
- There are 4 points; the step-size is $h = \frac{0.75-0}{3} = 0.25$
- The points are $x_0 = 0$, $x_1 = 0.25$, $x_2 = 0.5$ and $x_3 = 0.75$

Example

- Apply Euler's method:
  $y\left(x_{j+1}\right) \approx y\left(x_j\right) + h \cdot \left(x_j + y\left(x_j\right)\right)$
  - $y\left(x_1\right) \approx y\left(x_0\right) + h \cdot \left(x_0 + y\left(x_0\right) - 1\right)$
  - $y\left(0.25\right) \approx 2 + 0.25 \times \left(0 + 2 - 1\right) = 2.25$
  - $y\left(0.5\right) \approx 2.25 + 0.25 \times \left(0.25 + 2.25 - 1\right) = 2.625$
  - $y\left(0.75\right) \approx 2.625 + 0.25 \times \left(0.5 + 2.625 - 1\right) = 3.15625$
- The exact value of $y\left(0.75\right)$ is given by

$$y_{exact}(0.75) = 2e^{0.75} - 0.75 \approx 3.484$$

- The error is significant because h is too large.

MATLAB Code for Euler's Method

```
function [x,y] = Euler(a, b, N, y0, f)
% MATLAB code for using Euler's Method to solve the general 1st order IVP
% dy/dx=f(x,y) on the interval [a,b] with  y(a)=y0

% Inputs:    a and b define the interval [a,b]
%            N is the number of subintervals
%            y0 is the value of y(a)
%            f is the function f(x,y) which must be defined separately

% Outputs:   x stores the discretized values of x in the interval [a,b]
%            y stores the corresponding approximate values of the solution

h=(b-a)/N;  % determines the step-size needed
x=linspace(a,b,N+1);  % discretizes the interval for x
y=zeros(1,N+1);   % initializes y as a vector of zeros
y(1)=y0;    % stores the value of y(a) as the first entry in the y vector
for i=1:N   % using a for loop to compute the remaining y values
    y(i+1)=y(i)+h*feval(f,x(i),y(i));   % Algorithm for Euler's Method
end
plot(x,y)   % displays a plot of y versus x
end
```

Underestimates/Overestimates

- Euler's Method will normally produce **either** an underestimate **or** an overestimate of the actual solution.
- An underestimate can be identified by
  - a graph that is concave upward (the gradient increases as x increases), or
  - an increase in the approximate solution as the step-size is decreased (as h tends to zero, the approximate solution increases towards the actual solution).
- An overestimate can be identified by
  - a graph that is concave downward (the gradient decreases as x increases), or
  - a decrease in the approximate solution as the step-size is decreased (as h tends to zero, the approximate solution decreases towards the actual solution).