Lecture 8: Calculus and Differential Equations

Dr. Mohammed Hawa
Electrical Engineering Department
University of Jordan

EE201: Computer Applications. See Textbook Chapter 9.

Numerical Methods

- MATLAB provides many functions that support numerical solutions to common math problems:
  - Integration and Differentiation (Calculus)
  - Finding zeros of a function
  - Solving ordinary differential equations
  - Many others
- Numerical analysis provides answers as numbers, not closed-form solutions as in analytical solutions (see *next* lecture for symbolic math in MATLAB).

The integral of *f(x)* is the area *A* under the curve of *f* (*x*) from *x* = *a* to *x* = *b*.

$$A = \int_a^b f(x)dx$$

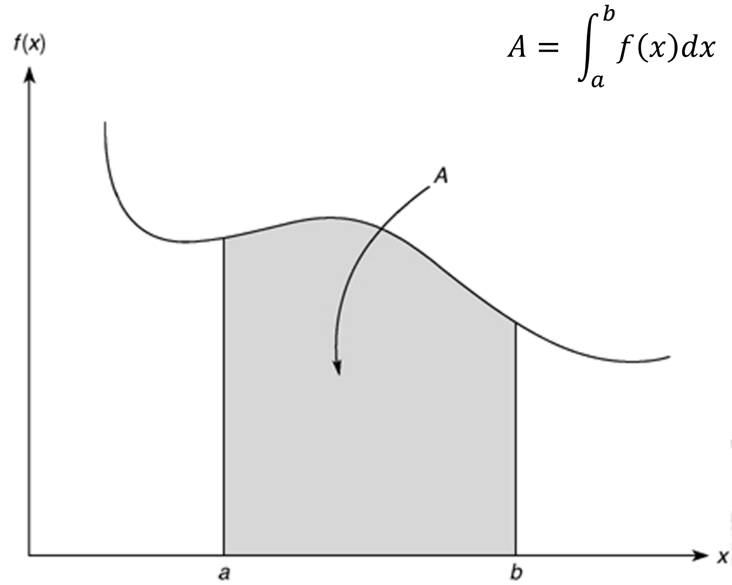
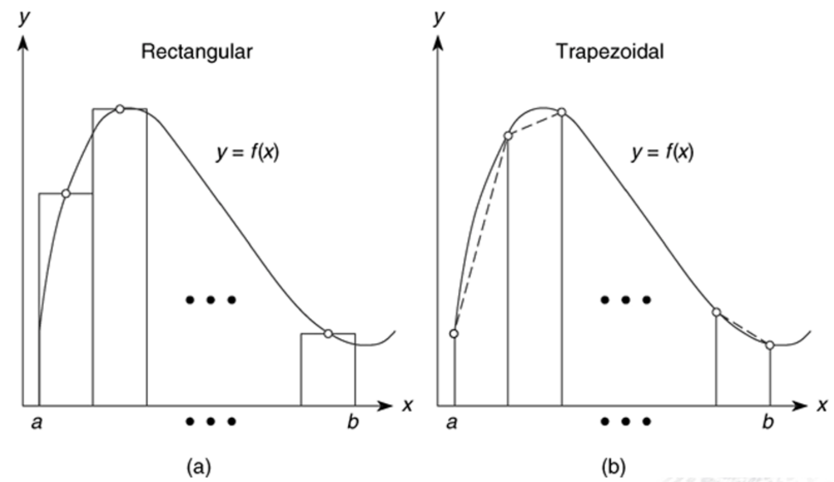


Illustration of Numerical Integration: (a) rectangular method and (b) more accurate trapezoidal method.



Example

$$A = \int_0^{\pi} \sin(x)\,dx = [-\cos(x)]_0^{\pi} = 1 - (-1) = 2$$

`trapz(x,y)`

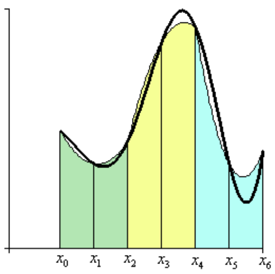
Uses trapezoidal integration to compute the integral of `y` with respect to `x`, where the array `y` contains the function values at the points contained in the array `x`.

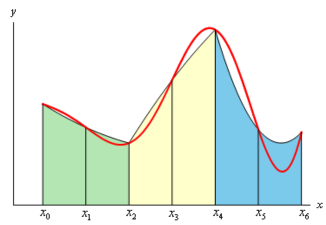
```
>> x = linspace(0,pi,10);
>> y = sin(x);
>> A = trapz(x,y)
A =
    1.9797

>> x = linspace(0,pi,100);
>> y = sin(x);
>> A = trapz(x,y)
A =
    1.9998
```


Simpson's Rule

- Another approach to numerical integration is Simpson's Rule, which divides the integration range [a, b] into an even number of sections and uses a different quadratic function to represent the integrand for each panel.





Important numerical integration functions:

`quad(fun, a, b)` `quad(fun, a, b, tol)`	Uses an adaptive Simpson's rule to compute the integral of the function whose handle is `fun`, with `a` the lower limit and `b` the upper limit. The function `fun` must accept a vector argument. The parameter `tol` is optional, and indicates the specified error tolerance.
`quadl(fun,a,b)`	Uses Lobatto quadrature to compute the integral of the function `fun`. The rest of the syntax is identical to `quad`.
`dblquad(fun, a, b, c, d)`	computes the integral of f(x,y) from x = a to b, and y = c to d. The function `fun` must accept a vector argument x and scalar y, and it must return a vector result.
`triplequad(fun,a,b,c,d,e,f)`	computes the integral of f(x,y,z) from x = a to b, y = c to d, and z = e to f. The function must accept a vector x, and scalar y and z.

Although the `quad` and `quadl` functions are more accurate than `trapz`, they are restricted to computing the integrals of functions and cannot be used when the integrand is specified by a set of points. For such cases, use the `trapz` function.

MATLAB function `quad` implements an adaptive version of Simpson's rule, while the `quadl` function is based on an adaptive Lobatto integration algorithm.

To compute the integral of sin(*x*) from 0 to π, type

```
>> A = quad(@sin,0,pi)
```

The answer given by MATLAB is 2.0000, which is correct. We use `quadl` the same way; namely,

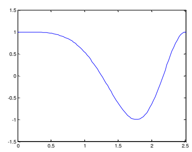
```
>>A = quadl(@sin,0,pi).
```


To integrate cos(x^2) from 0 to $\sqrt{2\pi}$, create the function in an m-file:

```
function yy = cossq(x)
yy = cos(x.^2);
```

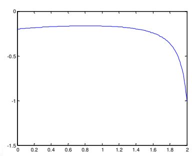
Note that we must use array exponentiation. Then `quad` function is called as follows:

```
>> quad(@cossq, 0, sqrt(2*pi))
ans =
    0.6119
```



Or you can use an anonymous function:

```
>> f = @(x)(1./(x.^3 - 2*x - 5));
>> quad(f, 0, 2)
ans =
    -0.4605
```



Double and Triple Integrals

`A = dblquad(fun, a, b, c, d)` computes the integral of *f(x,y)* from *x* = *a* to *b*, and *y* = *c* to *d*. Example: $f(x,y) = xy^2$.

```
>> fun = @(x,y) x.*y^2;
>> A = dblquad(fun, 1, 3, 0, 1)
A =
     1.3333
```

$$\int_c^d \int_a^b f(x,y)dxdy$$

`A = triplequad(fun, a, b, c, d, e, f)` computes the triple integral of *f(x,y, z)* from *x* = *a* to *b*, *y* = *c* to *d*, and *z* = *e* to *f*. Example: $f(x,y,z) = (xy - y^2)/z$.

```
>> fun = @(x,y,z)(x*y - y^2)/z;
>> A = triplequad(fun, 1,3, 0,2, 1,2)
A =
     1.8484
```

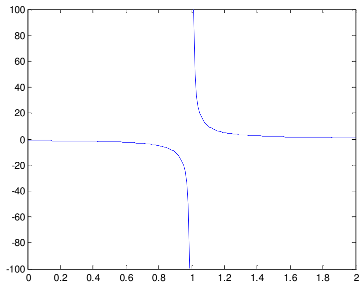
$$\int_e^f \int_c^d \int_a^b f(x,y,z)dxdydz$$

Note: The function must accept a vector *x*, but scalar *y* and *z*.

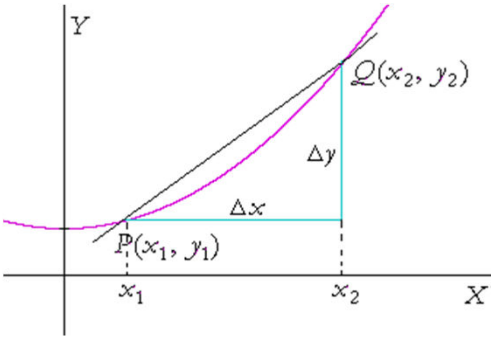
Be careful: function singularity

```
>> f = @(x) ( 1./(x-1));
>> quad(f, 0, 2)
Warning: Infinite or Not-
a-Number function value
encountered.
> In quad at 113
ans =
   NaN
```

$$\int_0^2 \frac{1}{1-x} dx$$



Numerical differentiation: Illustration of estimating the derivative *dy*/*dx*.



$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$$

$$\frac{dy}{dx} \approx \frac{y_2 - y_1}{x_2 - x_1}$$

MATLAB provides the `diff` function to use for computing derivative estimates.

`d = diff(y)`, where `y` is a vector of *n* elements, the result is a vector `d` containing *n* – 1 elements that are the differences between adjacent elements in `y`. That is:

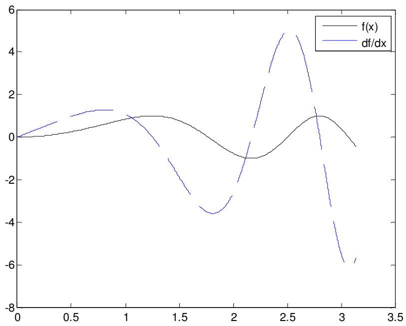
d=[*y*(2)-*y*(1), *y*(3)-*y*(2),..., *y*(*n*)-*y*(*n*-1)]

For example:

```
>> y = [5, 7, 12, -20];
>> diff(y)
ans =
      2      5    -32
```

Example

```
step = 0.001;
x = 0 : step : pi;
y = sin(x.^2);
d = diff(y)/step;
% an approximation
% to derivative
% 2.*x.*cos(x.^2)
plot(x,y,'k',x(2:end),d,'--');
legend('f(x)', 'df/dx');
```



Ordinary Differential Equations

- An ordinary differential equation (ODE) is an equation containing ordinary derivatives of the dependent variable.
- An equation containing partial derivatives with respect to two or more independent variables is a partial differential equation (PDE).
- We limit ourselves to ODE that must be solved for a given set of initial conditions.
- Solution methods for PDEs are an advanced topic, and we do not look at them.

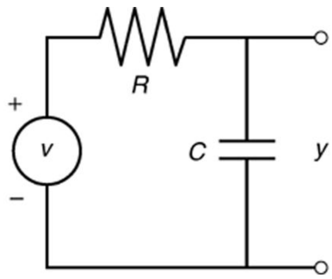
Several Methods

- Several numerical methods to solve ODEs.
- Examples include:
  - **Euler and Backward Euler methods**
  - Predictor-Corrector method
  - First-order exponential integrator method
  - **Runge-Kutta methods**
  - Adams-Moulton methods
  - Gauss-Radau methods
  - Adams-Bashforth methods
  - Hermite–Obreschkoff methods
  - Fehlberg methods
  - Parker–Sochacki methods
  - Nyström methods
  - Quantized State Systems methods

Multiple Solvers

- MATLAB offers multiple ODE solvers, each uses different methods.
- `Ode23`: Solves non-stiff differential equations, low order method.
- `ode45`: Solves non-stiff differential equations, medium order method: *uses a combination of fourth- and fifth-order Runge-Kutta methods*.
- `ode23s`: Solves stiff differential equations, low order method.
- `ode15i`: Solves fully implicit differential equations, variable order method.
- And so on.
- We will limit ourselves to the `ode45` solver.

Example: Find the response of the first-order RC circuit .



$$\tau \frac{dy}{dt} + y = 0$$

$$y(0) = V_c \quad (I.C.)$$

$$y(t) = y(0)e^{-t/\tau} \quad (natural\ response)$$

$$\tau \frac{dy}{dt} + y = V_s$$

$$y(0) = V_c \quad (I.C.)$$

$$y(t) = V_s + (y(0) - V_s)e^{-t/\tau} \quad (total\ response)$$

$$\dot{y}(t) = \frac{dy}{dt}$$

$$\ddot{y}(t) = \frac{d^2y}{dt^2}$$

Solving First-Order Differential Equations

First write the equation as *dy/dt = f(t,y)* then solve it using this syntax:

```
[t,y] = ode45(@f,tspan,y0)
```

where `@f` is the handle of the function file whose inputs must be *t* and *y*, and whose output must be a column vector representing *dy/dt*; that is, *f(t,y)*. The number of rows in the output column vector must equal the order of the equation.

The array `tspan` contains the starting and ending values of the independent variable *t*, and optionally any intermediate values.

The array `y0` contains the initial values of *y*. If the equation is first order, then `y0` is a scalar.

The circuit model for zero input voltage V_s and $\tau = 0.1$ is:

$$0.1 \times \frac{dy}{dt} + y = 0$$

And the i.c. is $y(0) = 2$ V.

First re-write the equation in the required format:

$$\frac{dy}{dt} = -10y$$

Next define the following function file. Note that the order of the input arguments must be *t* and *y*.

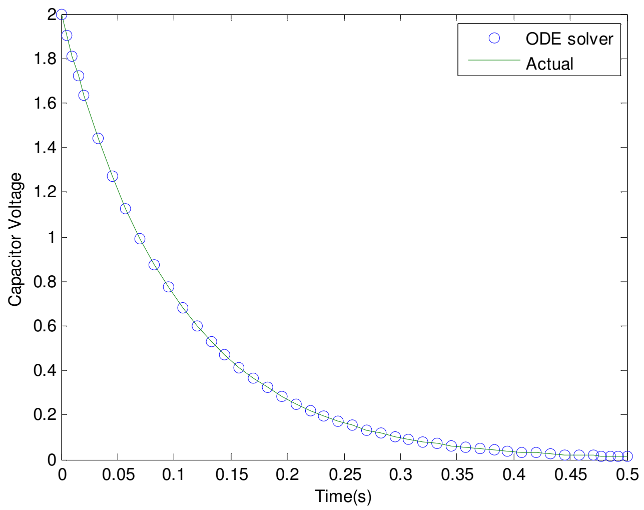
```
f = @(t,y) -10*y;
```


The solver is called as follows, and the solution plotted along with the analytical solution `y_true`. The initial condition is $y(0) = 2$.

```
f = @(t,y) -10*y;
[t, y] = ode45(f, [0 0.5], 2);
y_analytical = 2*exp(-10*t);
plot(t,y,'o', t, y_analytical);
legend('ODE solver', 'Actual');
xlabel('Time(s)');
ylabel('Capacitor Voltage');
```

Note that we need not generate the array `t` to evaluate `y_analytical`, because `t` is generated by the `ode45` function.
The plot is shown on the next slide.

Free (natural) response of an RC circuit (decaying exponential).



 Electrical Engineering Department, University of Jordan 23

The circuit model for input voltage $V_s = 10V$ and $\tau = 0.1$:

$$0.1 \times \frac{dy}{dt} + y = 10$$

And the i.c. is $y(0) = 2$ V.

First re-write the equation in the required format:

$$\frac{dy}{dt} = -10y + 100$$

Next define the following function file. Note that the order of the input arguments must be *t* and *y*.

```
f = @(t,y) -10*y+100;
```

 Electrical Engineering Department, University of Jordan 24

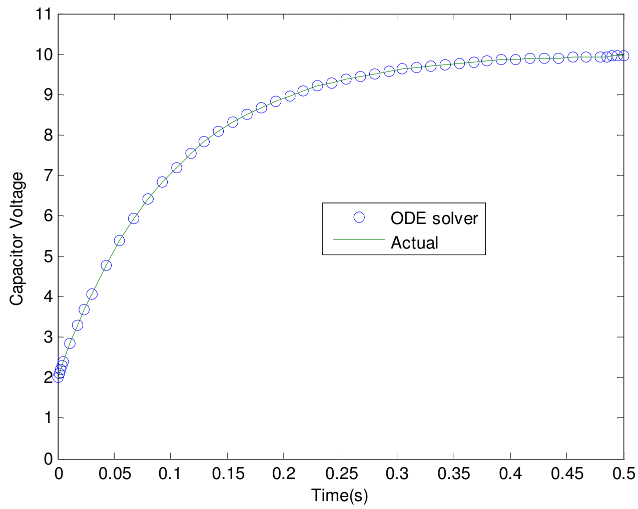
The solver is called as follows, and the solution plotted along with the analytical solution `y_true`. The initial condition is $y(0) = 2$.

```
f = @(t,y) -10*y+100;
[t, y] = ode45(f, [0 0.5], 2);
y_analytical = 10+(2-10)*exp(-10*t);
plot(t,y,'o', t, y_analytical);
legend('ODE solver', 'Actual');
xlabel('Time(s)');
ylabel('Capacitor Voltage');
```

Note that we need not generate the array `t` to evaluate `y_analytical`, because `t` is generated by the `ode45` function.
The plot is shown on the next slide.

 Electrical Engineering Department, University of Jordan 25

Natural plus forced (total) response of an RC circuit (increasing exponential).



 Electrical Engineering Department, University of Jordan 26

The circuit model for input voltage $V_s = 10e^{-t/0.3} \sin\left(\frac{2\pi t}{0.03}\right)$ and $\tau = 0.1$:

$$0.1 \times \frac{dy}{dt} + y = 10e^{-t/0.3} \sin\left(\frac{2\pi t}{0.03}\right)$$

And assume the i.c. is $y(0) = 0$ V.

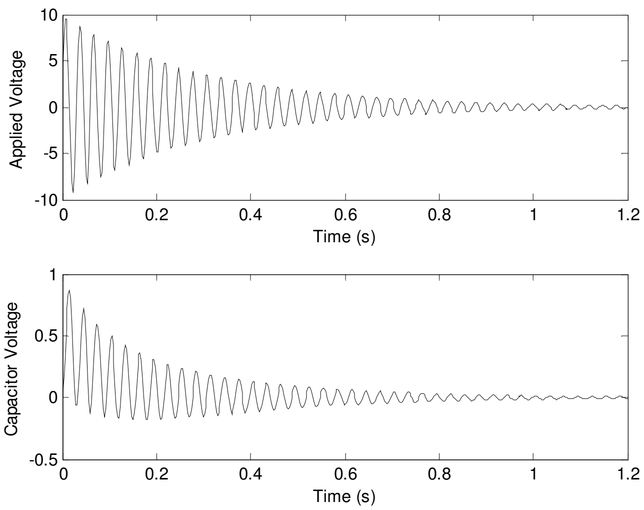
First re-write the equation in the required format:

$$\frac{dy}{dt} = -10y + 100e^{-t/0.3} \sin\left(\frac{2\pi t}{0.03}\right)$$

Next define the following function file. Note that the order of the input arguments must be *t* and *y*.

```
f = @(t,y) -10*y+100* ...
exp(-1*t/0.3).*sin(2*pi*t/0.03);
```


Result



Extension to Higher-Order Equations

To use the ODE solvers to solve an equation of 2nd order or higher, you must first write the equation as a set of first-order equations.

Example:

$$5\frac{d^2y}{dt^2} + 7\frac{dy}{dt} + 4y = f(t)$$

By re-arranging to get the highest derivative:

$$\frac{d^2y}{dt^2} = \frac{1}{5}f(t) - \frac{4}{5}y - \frac{7}{5}\frac{dy}{dt}$$

 Electrical Engineering Department, University of Jordan 29

Example (*Continue*)

$$\frac{d^2y}{dt^2} = \frac{1}{5}f(t) - \frac{4}{5}y - \frac{7}{5}\frac{dy}{dt}$$

We then change variables: $x_2 = dy/dt$
Hence: $dx_2/dt = d^2y/dt^2$
Also: $x_1 = y$. Hence we have two equations:

$$\frac{dx_1}{dt} = x_2$$

$$\frac{dx_2}{dt} = \frac{1}{5}f(t) - \frac{4}{5}x_1 - \frac{7}{5}x_2$$

 Electrical Engineering Department, University of Jordan 30

Example (*Continue*)

$$\frac{dx_1}{dt} = x_2$$
$$\frac{dx_2}{dt} = \frac{1}{5}f(t) - \frac{4}{5}x_1 - \frac{7}{5}x_2$$

This form is sometimes called the Cauchy form or the state-variable form.

We now define a function that accepts two values of x and then computes the values of dx_1/dt and dx_2/dt and stores them in a column vector.

Example (Code)

$$\frac{dx_1}{dt} = x_2$$
$$\frac{dx_2}{dt} = \frac{1}{5}\sin(t) - \frac{4}{5}x_1 - \frac{7}{5}x_2$$

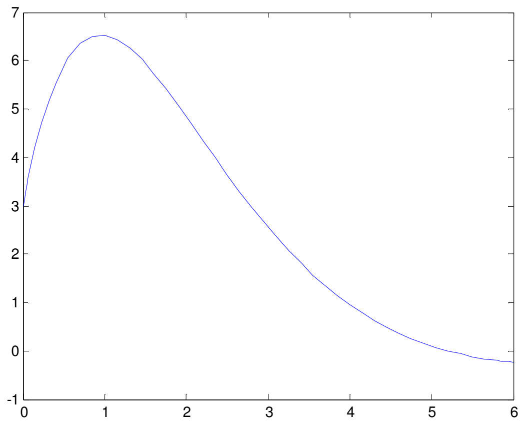
```
d = @(t,x) [x(2); sin(t)/5-4*x(1)/5-7*x(2)/5];
[t, x] = ode45(d, [0 6], [3 9]);
```

Here $x(0) = 3$ and $\dot{x}(0) = 9$, and we solve for $0 \le t \le 6$. Also $f(t) = \sin(t)$.

Note `x` is a matrix with two columns. The first column contains the values of x_1 at the various times generated by the solver; the second column contains the values of x_2.

If you type `plot(t, x)`, you will obtain a plot of both x_1 and x_2 versus `t`. Thus, type `plot(t, x(:,1))` to see the result for `y`.

Result



HW: Alternative Solution

Define the function in an m-file:

```
function xdot = d(t, x)
xdot(1) = x(2);
xdot(2) = (1/5)*(sin(t)-4*x(1)-7*x(2));
xdot = [xdot(1); xdot(2)];
```

Use the function to solve the ODE:

```
[t, x] = ode45(@d, [0 6], [3 9]);
% notice the need to use handles
plot(t, x(:,1));
```

Homework

- Solve as many problems from Chapter 9 as you can
- Suggested problems:
- Solve: 9.1, 9.4, 9.14, 9.16, 9.23, 9.27, 9.31, 9.34.

 Electrical Engineering Department, University of Jordan 35