

ChE 400: Applied Chemical Engineering Calculations
Tutorial 6: Numerical Solution of ODE Using Excel and Matlab

Gerardine G. Botte

This handout contains information about:

- How to solve ODEs using Euler's method and Runge Kutta's method using Excel
- How to solve systems of ODEs using Euler's method and Runge Kutta's method using Excel
- How to solve ODEs and systems of ODEs using build up subroutines available in Matlab

1. Solve the following differential equation using Euler's method:

$$\frac{dy}{dx} = -2x^3 + 12x - 20x + 8.5$$

with the initial value $y = 1$ @ $x = 0$. Stop your solution when $x = 1$.

Compare your solution with the exact solution. Discretize your function for $\Delta x = 0.1$. Is your solution stable for $\Delta x = 0.1$? What is the Δx for a stable solution if you want to be accurate for 2 decimals?

Solution:

1. The analytical solution is given by:

$$y = -\frac{x^4}{2} + 4x^3 - 10x^2 + 8.5x + 1$$

2. The value of the dependent variable can be calculated using Euler's method by:

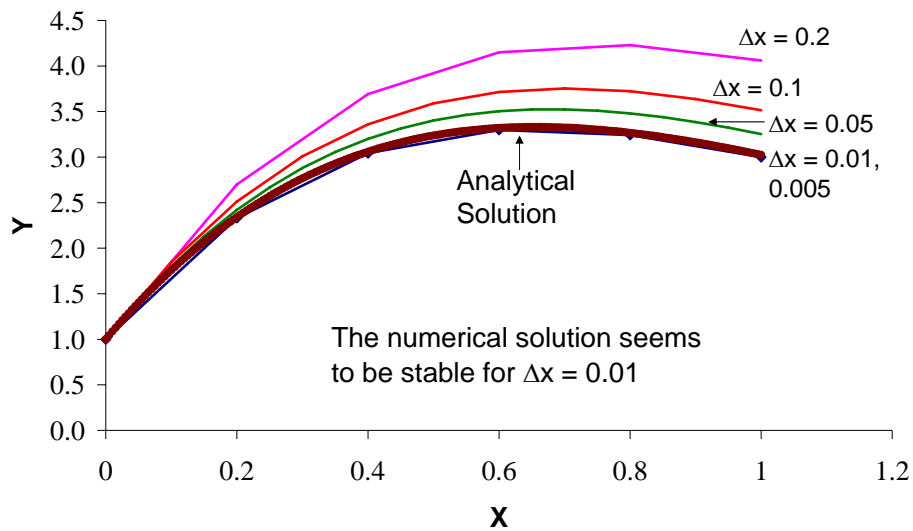
$$y_{i+1} = y_i + \Delta x f_i(x, y)$$

3. The solution is given by:

| node | x | Exact sol. | Numerical |
|------|-------|------------|-----------|
| 1 | 0.000 | 1.00 | 1.00 |
| 2 | 0.100 | 1.75 | 1.85 |
| 3 | 0.200 | 2.33 | 2.51 |
| 4 | 0.300 | 2.75 | 3.01 |
| 5 | 0.400 | 3.04 | 3.36 |
| 6 | 0.500 | 3.22 | 3.59 |
| 7 | 0.600 | 3.30 | 3.72 |
| 8 | 0.700 | 3.30 | 3.75 |
| 9 | 0.800 | 3.24 | 3.72 |
| 10 | 0.900 | 3.14 | 3.64 |
| 11 | 1.000 | 3.00 | 3.52 |

The solution is stable for $\Delta x = 0.01$, see results given below

Solution Example 1 Using Euler's method



2. Solve exercise 1 using second order Runge-Kutta method (midpoint method).

Solution:

- The solution using second order RK method is given by:

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2) \Delta x$$

where

$$k_1 = f(x_i, y_i)$$

- For the midpoint method $a_1 = 0$, $a_2 = 1$, $k_2 = f(x_i + \frac{\Delta x}{2}, y_i + \frac{k_1}{2} \Delta x)$. The solution is given below:

Midpoint Method: $\Delta x = 0.1$

| node | x | k1 | k2 | y numerical | y exact | Error |
|------|-------|--------|--------|-------------|---------|-------|
| 1 | 0 | | | 1 | | |
| 2 | 0.100 | 8.500 | 7.530 | 1.753 | 1.754 | 0.001 |
| 3 | 0.200 | 6.618 | 5.763 | 2.329 | 2.331 | 0.002 |
| 4 | 0.300 | 4.964 | 4.219 | 2.751 | 2.754 | 0.003 |
| 5 | 0.400 | 3.526 | 2.884 | 3.040 | 3.043 | 0.004 |
| 6 | 0.500 | 2.292 | 1.748 | 3.214 | 3.219 | 0.004 |
| 7 | 0.600 | 1.250 | 0.797 | 3.294 | 3.299 | 0.005 |
| 8 | 0.700 | 0.388 | 0.021 | 3.296 | 3.302 | 0.006 |
| 9 | 0.800 | -0.306 | -0.594 | 3.237 | 3.243 | 0.006 |
| 10 | 0.900 | -0.844 | -1.058 | 3.131 | 3.138 | 0.007 |
| 11 | 1.000 | -1.238 | -1.385 | 2.993 | 3.000 | 0.007 |

3. Solve the following system of ODE using Euler's method, second order R-K (midpoint):

$$\frac{dy_1}{dx} = -0.5y_1$$

$$\frac{dy_2}{dx} = 4 - 0.3y_2 - 0.1y_1$$

Initial conditions: $y_1 = 4$, $y_2 = 6$ @ $x = 0$. Use $\Delta x = 0.5$

Solution:

1. Euler's method ($\Delta x=0.5$):

| node | x | y1 | y2 |
|------|-------|-------|-------|
| 1 | 0.000 | 4.000 | 6.000 |
| 2 | 0.500 | 3.000 | 6.900 |
| 3 | 1.000 | 2.250 | 7.715 |
| 4 | 1.500 | 1.688 | 8.445 |
| 5 | 2.000 | 1.266 | 9.094 |

2. R-K Midpoint ($\Delta x=0.5$):

| node | x | k1,1 | k2,1 | y1 | k1,2 | k2,2 | y2 |
|------|-------|--------|--------|-------|-------|-------|-------|
| 1 | 0.000 | | | 4.000 | | | 6.000 |
| 2 | 0.500 | -2.000 | -1.750 | 3.125 | 1.800 | 1.620 | 6.810 |
| 3 | 1.000 | -1.563 | -1.367 | 2.441 | 1.645 | 1.480 | 7.550 |
| 4 | 1.500 | -1.221 | -1.068 | 1.907 | 1.491 | 1.342 | 8.221 |
| 5 | 2.000 | -0.954 | -0.834 | 1.490 | 1.343 | 1.209 | 8.825 |

4. Solving an ODE or system of ODEs using Matlab:

ODE Solver Basic Syntax: All of the ODE solver functions share a syntax that makes it easy to try any of the different numerical methods if it is not apparent which is the most appropriate. To apply a different method to the same problem, simply change the ODE solver function name. The simplest syntax, common to all the solver functions, is:

$$[t,y] = \text{solver}(\text{odefun}, \text{tspan}, y_0)$$

where solver is one of the ODE solver functions listed below: ODE45, ODE23, etc. See Matlab help for more details about each of them. The ODE solvers are based on Runge Kutta method. ODE23 uses a second order Runge-Kutta type, while ODE45 uses a fourth order R-K type.

The basic input arguments are:

odefun: Function that evaluates the system of ODEs. It has the form $dydt = \text{odefun}(t,y)$ where t is a scalar, and $dydt$ and y are column vectors.

tspan: Vector specifying the interval of integration. The solver imposes the initial conditions at tspan(1), and integrates from tspan(1) to tspan(end). For tspan vectors with two elements [t0 tf], the solver returns the solution evaluated at every integration step. For tspan vectors with more than two elements, the solver returns solutions evaluated at the given time points. The time values must be in order, either increasing or decreasing. Specifying tspan with more than two elements does not affect the internal time steps that the solver uses to traverse the interval from tspan(1) to tspan(end). All solvers in the MATLAB ODE suite obtain output values by means of continuous extensions of the basic formulas. Although a solver does not necessarily step precisely to a time point specified in tspan, the solutions produced at the specified time points are of the same order of accuracy as the solutions computed at the internal time points. Specifying tspan with more than two elements has little effect on the efficiency of computation, but for large systems, affects memory management.

y0: Vector of initial conditions for the problem

The output arguments are:

t: Column vector of time points

y: Solution array. Each row in y corresponds to the solution at a time returned in the corresponding row of t.

Additional ODE Solver Arguments

For more advanced applications, you can also specify as input arguments solver options and additional problem parameters.

Options: Structure of optional parameters that change the default integration properties. This is the fourth input argument.

$$[t,y] = \text{solver}(\text{odefun},\text{tspan},\text{y0},\text{options})$$

Improving ODE Solver Performance tells you how to create the structure and describes the properties you can specify.

p1,p2...: Parameters that the solver passes to odefun.

$$[t,y] = \text{solver}(\text{odefun},\text{tspan},\text{y0},\text{options},\text{p1},\text{p2}\dots)$$

IMPORTANT: It is recommended to define your function odefun in a separate file. When the function is called by the solver the symbol @ should be used as: @odefun, instead of the regular 'odefun' used in the past.

5. Solve the system of ODE given in exercise 3 using Matlab. Use ODE23 for the solution.

The first step is to define the function file that contains the system of ODEs:

```

C:\MATLAB6p1\work\system1.m
File Edit View Text Debug Breakpoints Web Window Help
Stack Base
1 function dy = system(t,y)
2 %This function contains the system of ODE to be solved.
3 %call: system(x,y)
4 %x: x vector
5 %y: vector with the function
6 %Developed by GGB on 11/06/02. Last modified on 11/06/02
7 %-----
8 dy = zeros(2,1); % a column vector. The vector is initialize with all the values equal to zero just to make
9 % sure that it does not contain other type of garbage
10 dy(1) = -0.5*y(1);
11 dy(2) = 4-0.3*y(2)-0.1*y(1);
system1.m odeh6.m
Ready

```

The second step is to build the program using the function defined above:

```

C:\MATLAB6p1\work\odeh6.m
File Edit View Text Debug Breakpoints Web Window Help
Stack Base
1 %This program solves the system of ODEs
2 %-----
3 clc;
4 [X,Y] = ode23(@system,[0 2],[4 6]); % use of the ODE solver
5 plot(X,Y(:,1),'-',X,Y(:,2),'-.');
6 title('Solution of the System of ODE'); % Plot details, title
7 xlabel('X '); % Plot labeling of x
8 ylabel('Y');
9 legend('Y1','Y2');
10 [X,Y] = ode23(@system,[0 0.5 1 1.5 2],[4 6]) % if we want the solution at the specific times used before
system1.m odeh6.m HW2.m
Ready

```

Solutions:

```
Command Window
File Edit View Web Window Help
>>
X =
    0
    0.5000
    1.0000
    1.5000
    2.0000

Y =
    4.0000    6.0000
    3.1152    6.8577
    2.4261    7.6321
    1.8894    8.3269
    1.4715    8.9468

>>
Ready
```

