Lecture 43, 44, 45 - Numerical solution of ordinary differential equations (ODE's)

(A) Theory
(B) Problems
(C) MATLAB Applications
Text: Supplementary notes from Instructor

43.1 Why is it important to be able to solve ordinary differential equations?

Ordinary differential equations are ubiquitous through-out chemical engineering. Your first exposure to ODE's in chemical engineering was probably a non steady-state mass or mole balance where we said

$$\text{accumulation} = \text{in} - \text{out} + \text{generation} - \text{consumption} \qquad (43.1)$$

For a system with a material entering a system (say the system is a reactor) and leaving the system and being consumed according to an irreversible first-order reaction, we have:

$$\frac{dN(t)}{dt} = \dot{N}_{in}(t) - \dot{N}_{out}(t) + k_1 \frac{N(t)}{V} \qquad (43.2)$$

where t is time, $N(t)$ is the moles of the material in the reactor at time t, V is the volume of the reactor, $\dot{N}_{in}(t)$ is the molar flowrate into the reactor, $\dot{N}_{out}(t)$ is the molar flowrate our of the reactor, and k_1 is the reaction rate constant.

If we want to find the solution to the non-steady-state problem, we need to be able to solve this type of equation. What is the solution to an ODE problem. Usually, the solution is not a number. Rather, it is a function. We want to find what function, $N(t)$, satisfies this ODE, given some initial condition.

If you look at this problem, you can see we need to integrate this ODE in order to solve. However we cannot use the techniques we have learned for numerical integration, like the Trapezoidal rule or Simpson's 1/3 rule because those techniques assume that you *already know* $N(t)$. In an ODE problem, we do not already know the function. We need new tools. This kind of problem is called an initial value problem because we will need some initial condition to fully solve it.

43.2 Euler's Method - Theory

The simplest method to solve an ODE is called Euler's Method. Looking at the example above, let's assume we know how much material is in the reactor initially. That is we have an initial condition. For a first-order differential equation like 43.2, we need one initial condition. Let's say that $N(t = 0) = N_0$. Now, we would like to know how much material is in the reactor at any time t.

Let's generalize this. Our equation is of the form:

$$\frac{dy(x)}{dx} = f(y,x) \tag{43.3}$$

with an initial condition $y(x = x_0) = y_0$. What Euler's method does is take steps through x (the independent variable) to obtain the value of y at other x. $f(y,x)$, by definition, is the derivative of y with respect to x. The first order approximation for the derivative at a point x_1 is

$$\left.\frac{dy(x)}{dx}\right|_{x_1} = f(y(x_1),x_1) \approx \frac{y(x_2) - y(x_1)}{x_2 - x_1} \tag{43.4}$$

You should notice that this is the exact same approximation we used to derive the Newton-Raphson root-finding method. There is no magic in the derivation of these techniques. This is straightforward calculus. We can rearrange equation (43.4) for $y(x_2)$

$$y(x_2) = y(x_1) + (x_2 - x_1)f(y(x_1),x_1) \tag{43.5}$$

This is Euler's method. If we set x_1 to our initial condition, we now have an approximation for y at x_2. We can then calculate y at x_3 using an analogous formula:

$$y(x_3) = y(x_2) + (x_3 - x_2)f(y(x_2),x_2) \tag{43.6}$$

In general we can calculate y at x_{i+1} as long as we know y at x_i.

$$y(x_{i+1}) = y(x_i) + (x_{i+1} - x_i)f(y(x_i),x_i) \tag{43.7}$$

This is the general equation for Euler's method. If we make the size of our steps from all x_i to $x_{i+1}$the same size, then we can that $\Delta x = x_{i+1} - x_i$. Our general formula for the Euler method is then:

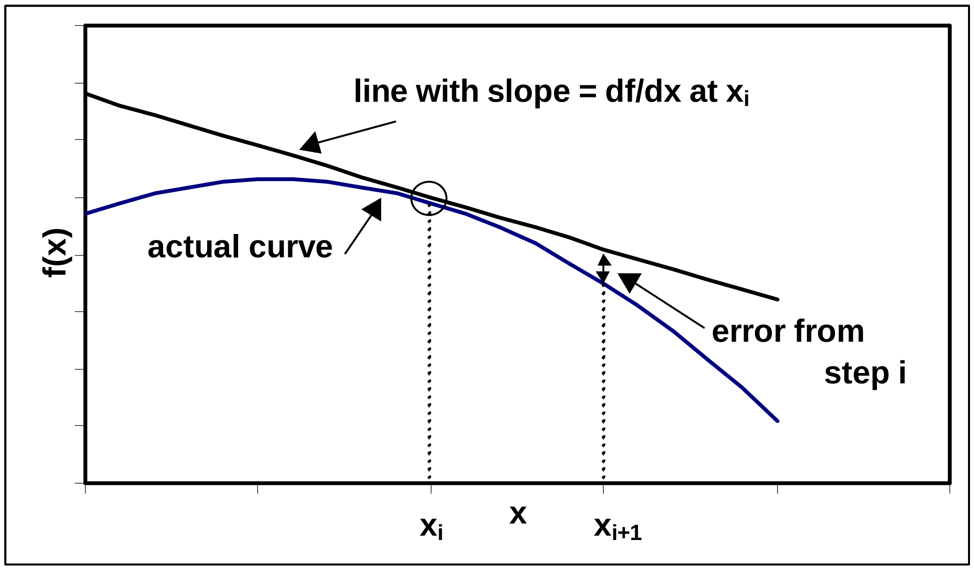
$$y(x_{i+1}) = y(x_i) + \Delta x * f(y(x_i),x_i) \tag{43.8}$$

The algorithm for Euler's method is as follows:

- Step One. Calculate $y(x)$ at point i
- Step Two. Calculate $f(y(x_i),x_i)$ at point i

- Step Three. Calculate new estimate for y $y(x_i)$ at point i (equation (43.8))
- Step Four. Loop back to step Two.

One step of Euler's method is graphical depicted below.



43.3 Euler's Method - Problem

For a system with a material entering a system (say the system is a reactor) and leaving the system and being consumed according to an irreversible first-order reaction, we have:

$$\frac{dN(t)}{dt} = \dot{N}_{in}(t) - \dot{N}_{out}(t) + k_1 \frac{N(t)}{V} \qquad (43.9)$$

where t is time, $N(t)$ is the moles of the material in the reactor at time t, V is the volume of the reactor, $\dot{N}_{in}(t)$ is the molar flowrate into the reactor, $\dot{N}_{out}(t)$ is the molar flowrate our of the reactor, and k_1 is the reaction rate constant.

We need to properly pose the problem first, before we can numerically solve it. We need to specify all of the parameters or functions in equation (43.9).

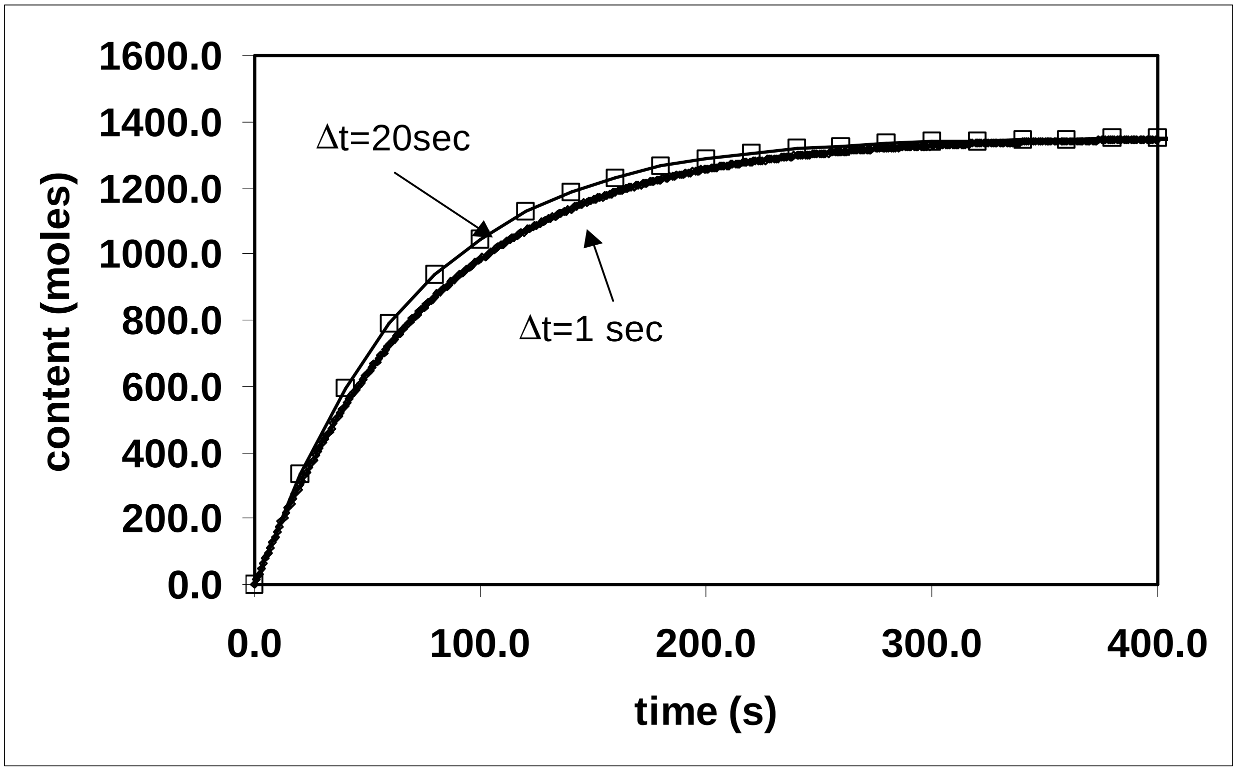
- The initial condition is $N(t = 0) = 0$; the reactor is initially empty.
- The volume of the reactor is 1.0 m^3.
- The input flow rate is constant at $\dot{N}_{in}(t) = \dot{N}_{in} = 16.67\ mol/s$.
- The reaction rate constant at the operating temperature is $k_1 = 0.01 m^3/s$.
- The output flowrate is a function that varies with time.
- The reactor is an upright cylinder, 2 meters in height with radius, $r = .399\ m$

- The material has a molecular weight of $MW = 0.100 \text{ kg/mol}$
- The material has a density of $\rho = 1000 \text{ kg/m}^3$ (water)
- The outflow is gravity driven. That means the out flowrate is determined by the height of the fluid in the reactor via the Bernoulli equation.
- The system is frictionless.
- The outflow leaves via a horizontal pipe of cross-sectional area, $A = 0.001 \text{m}^2$.
- The Bernoulli equation then states the linear velocity, u, is $u = \sqrt{2gh(t)}$ where g is gravity (10.0 m/s) and $h(t)$ is the height of the water in the reactor at time t
- The volumetric flowrate, $\dot{V}$, is $\dot{V} = Au$
- The out flowrate then is $\dot{N}_{out}(t) = \frac{N(t)}{V}\dot{V} = \frac{N(t)}{V}Au = \frac{N(t)}{V}A\sqrt{2gh(t)}$
- $h(t)$ is a function of $N(t)$; $h(t) = N(t) \cdot MW / \rho / \left(\pi r^2\right)$
- The out flowrate then is $\dot{N}_{out}(t) = \frac{N(t)}{V}A\sqrt{\frac{2gN(t)MW}{\rho\pi r^2}}$
- for our values, $\dot{N}_{out}(t) = 6.32 \cdot 10^{-5} \cdot N(t)^{3/2}$

So our ODE becomes:

$$\frac{dN(t)}{dt} = 16.67 - 6.32 \cdot 10^{-5} \cdot N(t)^{3/2} - 0.01N(t) \tag{43.10}$$

This equation is a non-linear, first-order, ordinary differential equation. Below, we show the plot of using Euler's method to solve this ODE.



For the molecular weight, density, and reactor volume that we used, the maximum reactor capacity is 10000 moles. We see that the reactor does not overflow. We also see that it takes about 400 seconds for the reactor to reach steady state.

The two plots shown were both generated using Euler's formula. One used a time step of 20 seconds and the other a time step of 1 sec. The discrepancy is not great but this may be due to the fact that the solution approaches a steady state. If this were not the case, we might expect that the error between the methods would continue to grow.

43.4 Classical Fourth-Order Runge-Kutta Method - Theory

Just like with all numerical problems, there is a whole body of methods that have been developed. The varying methods have different strengths and weaknesses. One of the weaknesses of the Euler's formula is that we use the derivative at point i to be the derivative across the entire interval between point i and point i+1. There are a family of methods which are basically modifications of Euler's method, which remove this deficiency.

The classical fourth-order Runge-Kutta method is

$$y(x_{i+1}) = y(x_i) + \left[\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\right]h \tag{43.11}$$

where

$$\begin{aligned}
k_1 &= f(x_i, y(x_i)) \\
k_2 &= f\left(x_i + \frac{h}{2}, y(x_i) + \frac{h}{2}k_1\right) \\
k_3 &= f\left(x_i + \frac{h}{2}, y(x_i) + \frac{h}{2}k_2\right) \\
k_4 &= f(x_i + h, y(x_i) + hk_3)
\end{aligned} \tag{43.12}$$

Without a rigorous derivation, we can see that the Runge-Kutta method improves over the estimate of the Euler method by constructing an estimate for the derivative which is more representative of the real derivative over the course of the interval.

43.5 Classical Runge-Kutta Fourth-Order Method - Problems

Let's compare the accuracy of the Euler Method and the Classical Runge-Kutta Fourth-Order Method (RK4). In order to do this we need to solve an ODE that we know the analytical answer to. Let's look at an ODE from the family:

$$\frac{dy(x)}{dx} = f(y,x) = cy(x) \tag{43.13}$$

where c is a constant. To solve this, separate y(x) from x and integrate.

$$\frac{dy(x)}{y(x)} = cdx \tag{43.14}$$

$$\int_{y(x=x_0)}^{y(x)} \frac{dy'(x)}{y'(x)} = \int_{x_0}^{x} cdx \tag{43.15}$$

$$\ln(y(x))\Big|_{y(x_0)}^{y(x)} = cx\Big|_{x_0}^{x} \tag{43.16}$$

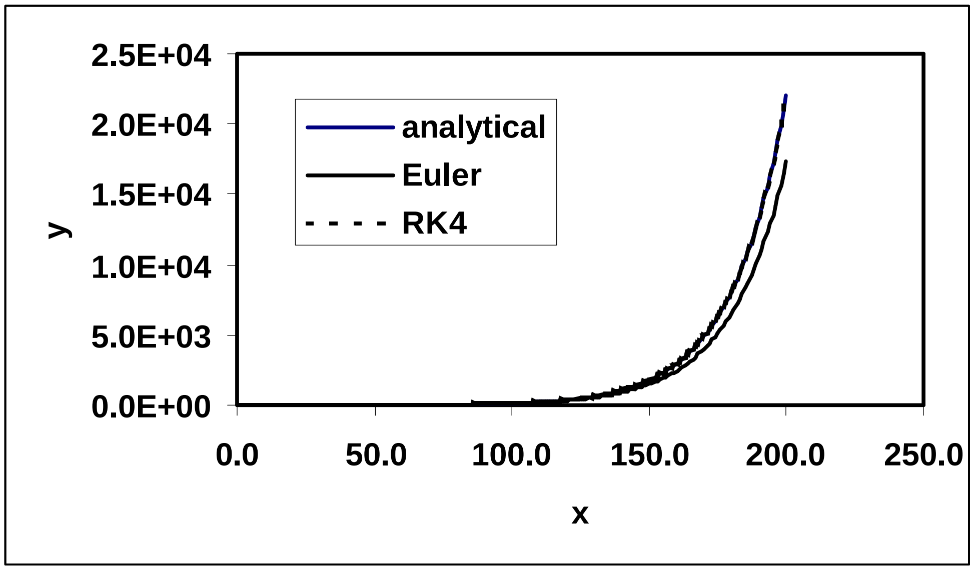
$$\ln\left(\frac{y(x)}{y(x_0)}\right) = c(x - x_0) \tag{43.17}$$

$$y(x) = y(x_0)e^{c(x-x_0)} \tag{43.18}$$

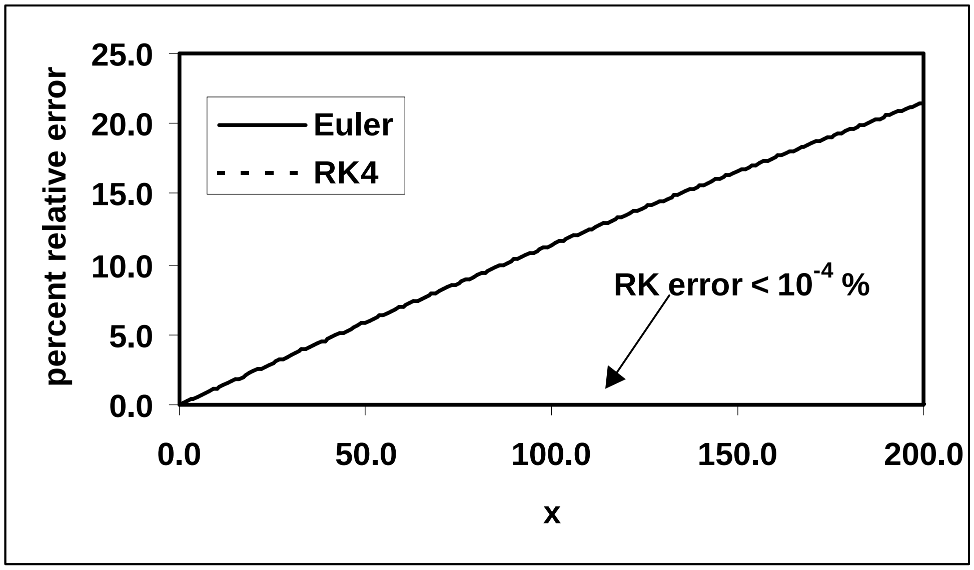
So, that's the analytical result in equation (43.18). Let's compare Euler and RK4 with the analytical result when $c = 0.05$ and when we have the initial condition $y(x_0 = 0) = 1$. The equation becomes:

$$y(x) = e^x \tag{43.19}$$

The graphical result is shown below for x up to 200.



As you can see, the RK4 is right on top of the analytical solution. The Euler method is beginning to substantially deviate from the analytical solution. We can plot the percent error for the two methods.



43.6 Numerical Solution of Higher-order ODE IVP's - Theory

Thus far, we have learned how to solve initial value problems (IVP) when we have one ordinary first-order differential equation. We are going to make three extension to that. We are going to solve

- the IVP for one ordinary nth-order differential equation
- the IVP for a system of ordinary first order differential equations
- the BVP for a one ordinary 2nd-order differential equation

There is a simple trick to solving the IVP for an ordinary nth-order differential equation. You make a substitution that transforms the nth-order differential equation into n first-order differential equations. The general form for a second-order differential equation is

$$\frac{d^2y(x)}{dx^2} = f\left(x, y(x), \frac{dy(x)}{dx}\right) \tag{43.20}$$

with the following initial conditions:

$$\left.\frac{dy(x)}{dx}\right|_{x=x_0} = y_0' \\ y(x = x_0) = y_0 \tag{43.21}$$

We make the substitution

$$z(x) = \frac{dy(x)}{dx} \tag{43.22}$$

and we obtain a system of 2 ordinary first-order differential equations, each with its own initial condition. Equation (43.20) becomes

$$\frac{dz(x)}{dx} = f(x, y(x), z(x)) \tag{43.23}$$

with the initial condition

$$z(x = x_0) = y'_0 \tag{43.24}$$

The second differential equation is just equation (43.22)

$$\frac{dy(x)}{dx} = z(x) \tag{43.22}$$

with the initial condition

$$y(x = x_0) = y_0 \tag{43.25}$$

In general any n-th order ODE can be converted into a system of n first-order ODE's using this technique. Remember, the problem must be an IVP, in order to use this technique.

So, we can solve any nth order ODE IVP if we can solve a system of n first-order ODE IVP's. Let's find out how to do that.

43.7 Numerical Solution of Systems of First-Order ODE IVP's -Theory

If you recall when we learned how to single-variable linear regression, there was simple extension to multivariable linear regression. When we learned how to do single-equation Newton-Raphson root-finding, there was a simple extension to multivariate Newton-Raphson root-finding. Similarly, there is a simple extension from solving a single first-order ODE IVP to solving a system of n first-order ODE IVP's.

We have the general problem of n ODE's:

$$\begin{gathered}
\frac{dy_1(x)}{dx} = f_1(x, y_1, y_2, y_3, \ldots y_{n-1}, y_n) \\
\frac{dy_2(x)}{dx} = f_2(x, y_1, y_2, y_3, \ldots y_{n-1}, y_n) \\
\ldots \\
\frac{dy_n(x)}{dx} = f_n(x, y_1, y_2, y_3, \ldots y_{n-1}, y_n)
\end{gathered} \tag{43.26}$$

with n initial conditions:

$$y_j(x = x_0) = y_{j,0} \quad \text{for } j = 1 \text{ to } n \tag{43.27}$$

Note carefully, that there are n ODE's, thus n functions, f_j, and n unknown functions, y_j, but there is only one independent variable x. This is what makes this system a system of ODE's rather than PDE's, partial differential equations. These techniques you are learning will not solve PDE's.

Euler's method for a system of ODE's

$$y_j(x_{i+1}) = y_j(x_i) + \Delta x * f_j\left(x_i, y_1(x_i), y_2(x_i), y_3(x_i), \ldots y_{n-1}(x_i), y_n(x_i)\right) \quad (43.28)$$

for j = 1 to n. There is no difference between this equation and the equation for the single system ODE using Euler's method (equation (43.8)). In this case, remember that the subscript j attached to the y and the f denotes different functions. The subscript i attached to the x variable denotes steps (or iterations).

The algorithm is to compute all f_j at iteration i, then compute all y_j at iteration i+1, and repeat the process.

Runge-Kutta's 4-order method for a system of ODE's

The extension of RK4 to systems of ODE's is just as simple as the extension of Euler's method. However, because RK4 has a little more sophistication, the extension looks more complicated, when it is really not. The RK4 equation for a system of equations is given by

$$y_j(x_{i+1}) = y_j(x_i) + \left[\frac{1}{6}\left(k_{1,j} + 2k_{2,j} + 2k_{3,j} + k_{4,j}\right)\right]h \quad (43.29)$$

for j = 1 to n, where

$$\begin{aligned}
k_{1,j} &= f_j\left(x_i, \{y_m(x_i)\}\right) \\
k_{2,j} &= f_j\left(x_i + \frac{h}{2}, \left\{y_m(x_i) + \frac{h}{2}k_{1,m}\right\}\right) \\
k_{3,j} &= f_j\left(x_i + \frac{h}{2}, \left\{y_m(x_i) + \frac{h}{2}k_{2,m}\right\}\right) \\
k_{4,j} &= f_j\left(x_i + h, \{y_m(x_i) + hk_{3,m}\}\right)
\end{aligned} \quad (43.30)$$

The algorithm for the RK4 method is to first calculate all four k functions for all n equations, giving 4n function evaluations. Then calculate new values of y at the next x value, using equation 43.29.

for j = 1 to n. There is no difference between this equation and the equation for the single system ODE using Euler's method (equation (43.8)). In this case, remember that the subscript j attached to the y and the f denotes different functions. The subscript i attached to the x variable denotes steps (or iterations).

The algorithm is to compute all f_j at iteration i, then compute all y_j at iteration i+1, and repeat the process.

43.8 Numerical Solution of Systems of First-Order ODE IVP's - Problem

Let's say we a second order ODE of the form:

$$\frac{d^2y(x)}{dx^2} = f\left(x, y(x), \frac{dy(x)}{dx}\right) = ax + by(x) + c\frac{dy(x)}{dx} \tag{43.31}$$

where a, b, and c are constants and with the following initial conditions:

$$\begin{aligned} &\left.\frac{dy(x)}{dx}\right|_{x=x_0} = y_0' \\ &y(x = x_0) = y_0 \end{aligned} \tag{43.32}$$

We make the substitutions

$$\begin{aligned} &y_1(x) = y(x) \\ &y_2(x) = \frac{dy_1(x)}{dx} \end{aligned} \tag{43.33}$$

Then we have the system of first-order ODE IVP's:

$$\frac{dy_1(x)}{dx} = f_1(x, y_1(x), y_2(x)) = y_2(x) \tag{43.34}$$

$$\frac{dy_2(x)}{dx} = f_2(x, y_1(x), y_2(x)) = ax + by_1(x) + cy_2(x) \tag{43.35}$$

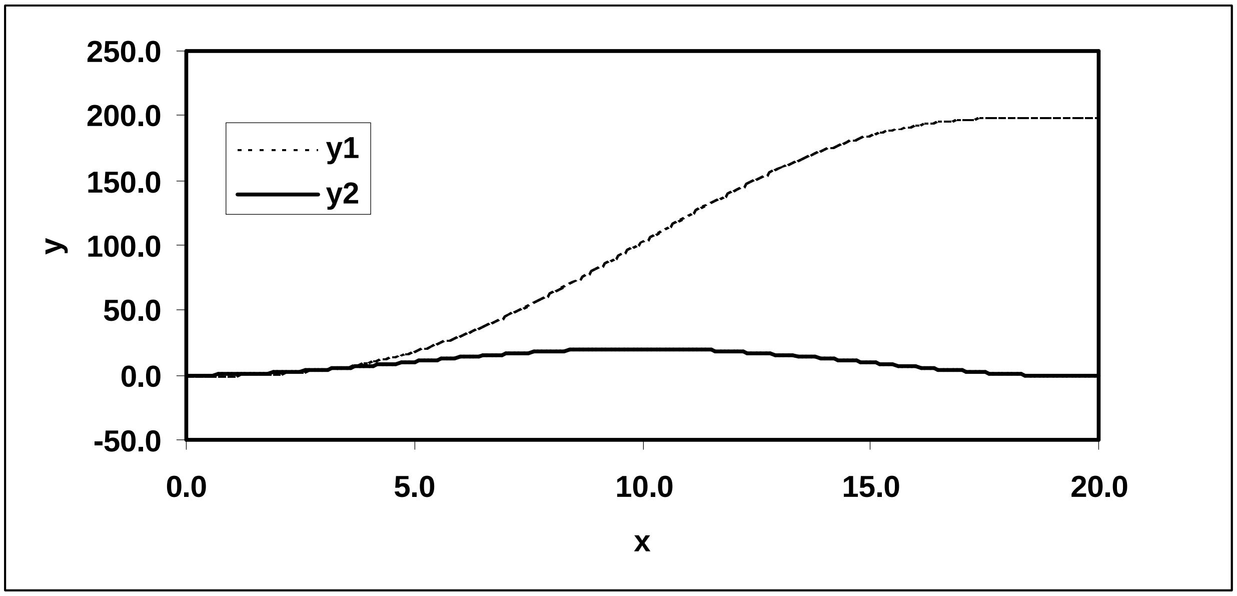
with the initial conditions

$$\begin{aligned} &y_2(x = x_0) = y_0' \\ &y_1(x = x_0) = y_0 \end{aligned} \tag{43.36}$$

For simplicity, let's make a=1.0, b=-0.1, c=0.01, $x_0 = 0$, $y_0 = 0$, $y_0' = 0$. Also let's choose a step size, h = 0.1. Using the RK4 method, we have for the first five iterations. (All these numbers are rounded to the nearest thousandth decimal place).

x	y1	y2	k1,1	k2,1	k3,1	k4,1	k1,2	k2,2	k3,2	k4,2
0.0	0.000	0.000	0.000	0.000	0.003	0.005	0.000	0.050	0.050	0.100
0.1	0.000	0.005	0.005	0.010	0.013	0.020	0.100	0.150	0.150	0.200
0.2	0.001	0.020	0.020	0.030	0.033	0.045	0.200	0.250	0.250	0.300
0.3	0.005	0.045	0.045	0.060	0.063	0.080	0.300	0.350	0.350	0.400
0.4	0.011	0.080	0.080	0.100	0.102	0.125	0.400	0.450	0.449	0.499
0.5	0.021	0.125	0.125	0.150	0.152	0.180	0.499	0.549	0.549	0.598

The curves for 200 iterations are shown below:



43.9 Numerical Solution of ODE BVP's (The Shooting Method) - Theory

In some cases the conditions that accompany a second-order differential equation are not of the initial condition form,

$$\left.\frac{dy(x)}{dx}\right|_{x=x_0} = y_0'$$
$$y(x = x_0) = y_0 \tag{43.32}$$

Instead the are of a boundary condition form. One such set of boundary conditions is

$$\begin{aligned} y(x = x_0) &= y_0 \\ y(x = x_F) &= y_F \end{aligned} \tag{43.37}$$

Note that only one of the boundary conditions (BC's) is at the initial time. The second BC is at some other, final time.

One technique used to solve this boundary value problem (BVP) is called the "shooting method" and relies completely on techniques we used for the initial value problem (IVP). First, we recast the second-order ODE as a system of 2 first-order ODE's (as we did in section 43.7), so that

$$\frac{d^2y(x)}{dx^2} = f\left(x, y(x), \frac{dy(x)}{dx}\right) \tag{43.31}$$

becomes

$$\begin{aligned} y_1(x) &= y(x) \\ y_2(x) &= \frac{dy_1(x)}{dx} \end{aligned} \tag{43.33}$$

We then make a guess for our missing initial condition, so that we have 2 initial conditions

$$\begin{aligned} y_1(x = x_0) &= y_{0,1} \ \text{(given as a BC)} \\ y_2(x = x_0) &= y_{0,2} \ \text{(guessed as an IC)} \end{aligned} \tag{43.38}$$

Now we have a system of ODE's exactly as we solved in Section 43.8. We then solve these ODE's up to the point $x = x_F$, where we check to see if the second BC of equation (43.32)

$$y(x = x_F) = y_F \tag{43.39}$$

is satisfied. If this BC is satisfied we have found the correct solution. If not, we need to pick a new guess for our second IC of equation (43.38). Picking a new guess can be tricky. One method is to use linear interpolation.

When we use linear interpolation to pick new guesses, first we make two guesses and solve the system of ODE's twice. Then we guess a third time with our third guess being

$$y_{0,2}^{(3)} = y_{0,2}^{(1)} + \left(y_{0,2}^{(2)} - y_{0,2}^{(1)}\right)\left[\frac{y_F - y(x_F)^{(1)}}{y(x_F)^{(2)} - y(x_F)^{(1)}}\right] \tag{43.40}$$

43.10 Numerical Solution of ODE BVP's (The Shooting Method) - Problem

Let's take the same ODE from section 43.8 and solve a BVP with it.

$$\frac{d^2y(x)}{dx^2} = f\left(x, y(x), \frac{dy(x)}{dx}\right) = ax + by(x) + c\frac{dy(x)}{dx} \tag{43.31}$$

with the boundary conditions

$$\begin{aligned} y(x = x_0) &= y_0 \\ y(x = x_F) &= y_F \end{aligned} \tag{43.37}$$

where a=1.0, b=-0.1, c=0.01, $x_0 = 0$, $y_0 = 0$, $y(x = 17.0) = y_F = 197.4585$. Also let's choose a step size, h = 0.1.

We transform the second-order ODE into 2 first-order ODE's.

$$\frac{dy_1(x)}{dx} = f_1(x, y_1(x), y_2(x)) = y_2(x) \tag{43.34}$$

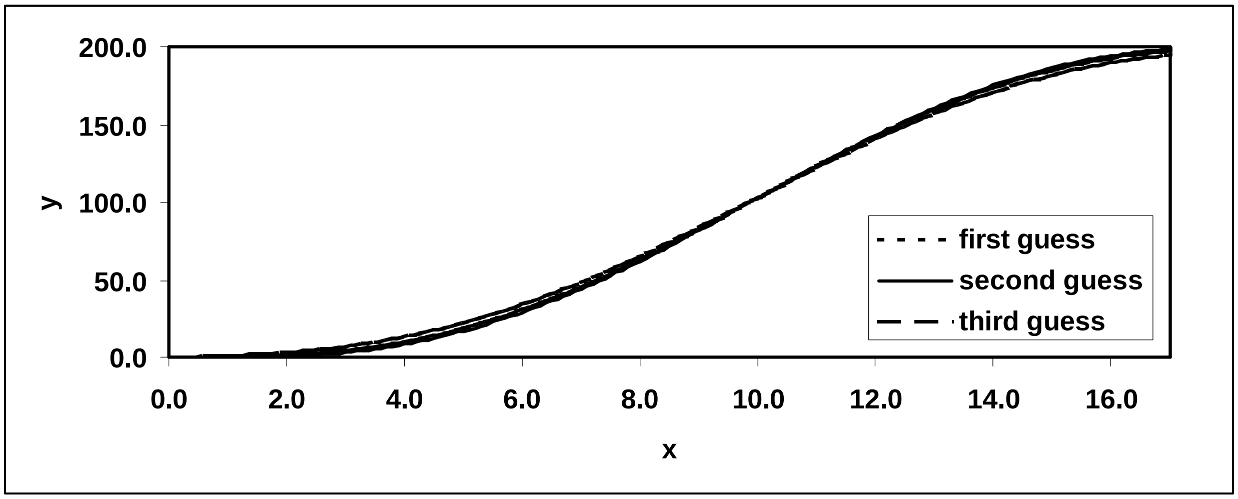
$$\frac{dy_2(x)}{dx} = f_2(x, y_1(x), y_2(x)) = ax + by_1(x) + cy_2(x) \tag{43.35}$$

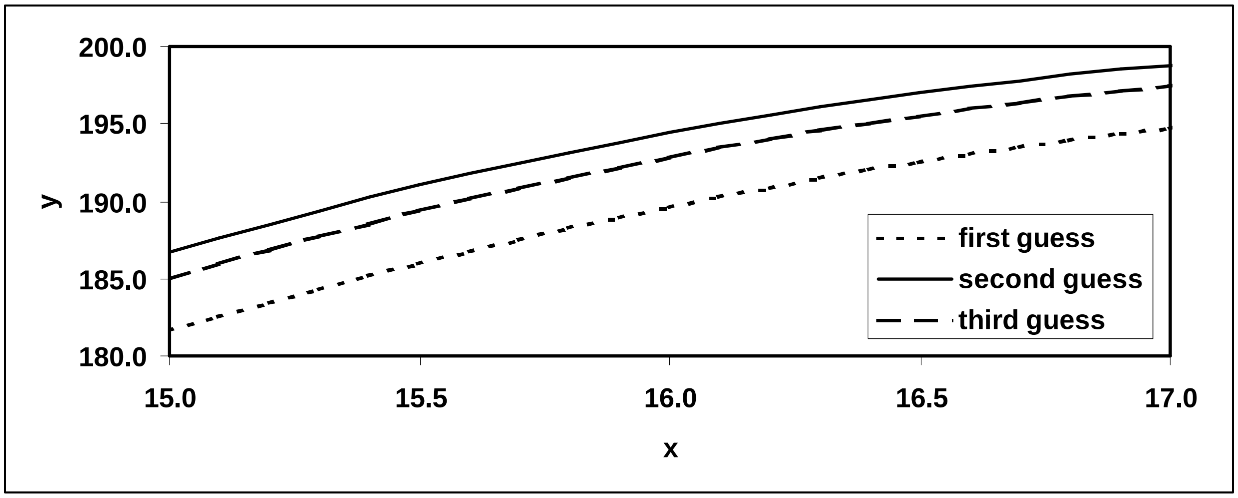
with the initial conditions

$$\begin{aligned} y_2(x = x_0) &= y'_0 \\ y_1(x = x_0) &= y_0 \end{aligned} \tag{43.36}$$

where we have to guess y'_0. Let's make our first guess of y'_0, ${y'_0}^{(1)} = 1.0$

We then solve the system of ODE's up to $x = x_F = 17.0$ using a technique like RK4. Doing so yields, $y(x_F)^{(1)} = y(x = x_F) = 194.7443$. For our second guess, let's try a value of y'_0, ${y'_0}^{(2)} = -0.5$. We then again solve the system of ODE's up to $x = x_F = 17.0$ using a technique like RK4. Doing so yields, $y(x_F)^{(2)} = y(x = x_F) = 198.8156$. Let's find a third guess using equation (43.40). This equation, yields an initial guess of ${y'_0}^{(3)} = 0.000$. We then again solve the system of ODE's up to $x = x_F = 20.0$ using a technique like RK4. Doing so yields, $y(x_F)^{(2)} = y(x = x_F) = 197.4585$. This is precisely the answer we where shooting for. We have solved the ODE BVP problem. Here, in graphical form, are the functions y, from our three guesses. (First the whole function and then a close-up of the point we were shooting for.)





We hit the answer precisely on the third point because the ODE we were solving was linear. If the ODE was non-linear, the interpolation would not be exact and we would need more (possibly many more) iterations of the shooting method.

43.10 Numerical Solution of ODE's - MATLAB

So far in this section, we have learned how to

- solve a single first-order ODE IVP using Euler or RK4 methods.
- solve a single nth-order ODE IVP using Euler or RK4 methods.
- solve a system of first-order ODE IVP's using Euler or RK4 methods

- solve a 2nd-order ODE BVP using the shooting method with either Euler or RK4 methods.

All of these numerical techniques are computation intensive. They can not be done efficiently by hand. MATLAB can be used to solve all four types of problems.

ODE INITIAL VALUE PROBLEMS IN MATLAB

On the website, I have provided a routine called sysode.m that can solve systems of first order ordinary differential equations. It will work for a single ODE as well.

The description for how to use the file can be obtained by opening MATLAB, moving to the directory where you have downloaded the sysode.m file, and typing

```
help sysode
```

This yields:

```
sysode(m,n,xo,xf,yo)
    This routine solves one non-linear first-order ordinary differential
    equation initial value problem.

   m = 1 for Euler's method
   m = 2 for Classical Runge-Kutta 4rth order method
   n = number of steps
   xo = starting value of x
   xf = ending value of x
   o = number of first order ordinary differential equations
   yo = initial condition at xo

    The differential equation must appear in the file 'sysodeinput.m'
   This program creates an output data file 'sysode.out'

   Author:  David Keffer   Date:  October 23, 1998
```

Hopefully, it is clear from this information how to use sysode.m at the command line.
For example,

```
sysode(2,100,0,10,0)
```

would use the Runge-Kutta method to solve a single ODE IVP over the range $0 \le x \le 10$ with 100 intervals, and the initial condition $y(x = 0) = 0$.

For a system of 3 equations, sysode.m would be started by typing at the MATLAB prompt:

```
sysode(2,100,0,10,[4,5,6])
```

which would use the Runge-Kutta method to solve a system of 3 ODE IVP's over the range $0 \le x \le 10$ with 100 intervals, and the initial condition $y_1(x = 0) = 4$, $y_2(x = 0) = 5$, $y_3(x = 0) = 6$.

The ODE's to be solved by the code sysode.m must be entered in the file sysodeinput.m. This file can be as simple as:

```
function dydt = sysodeinput(x,y,nvec);
dydt = -0.25*y;
```

if you wanted to solve the ODE

$$\frac{dy}{dt} = -\frac{y}{4}$$

This file can also be more complicated, for example:

```
function dydt = sysodeinput(x,y,nvec);
% L is pipe length (cm)
L= 300.0;
% dp is pipe diameter (cm)
dp = 0.48;
% g is gravity (cm/s^2)
g = 980.67;
% mu is viscosity (g/(cm*s))
mu = 0.0103;
% rho is density (g/cm^3)
rho = 0.9970;
% dta = tank diameter (cm)
dta = 15.24;
%
dydt = -(g*rho^0.25*dp^4.75*(1+y/L)/(2*0.0791*mu^0.25*dta^3.5))^(4/7);
```

Or, for a case of a system of 3 ODE's it might look like:

```
function dydt = sysodeinput(x,y,nvec);
c=[-1.0,-2.0,3.0,-4.0,5.0];
dydt(1) = y(2);
dydt(2) = y(3);
dydt(3) = c(1)*y(3) + c(2)*y(2) + c(3)*y(1) + c(4)*x + c(5);
```

The input variable nvec in the routine sysodeinput.m allows for certain Runge-Kutta variables to be transmitted from sysode.m to sysodeinput.m. It needs to remain as an argument.

In order to use sysode.m to solve a higher order ODE, you must first analytically transform the nth-order ODE to a system of n first-order ODEs. Then this system can be solved using sysode.m.

ODE BOUNDARY VALUE PROBLEMS IN MATLAB

On the website, I have provided a routine called shooting.m that can solve a second order ordinary differential equation boundary value problem. Shooting.m takes the guts of sysode.m and adds a little extra manipulation on top. Basically the two codes solve ODEs by the same method.

The description for how to use the shooting.m file can be obtained by opening MATLAB, moving to the directory where you have downloaded the shooting.m file, and typing
```
help shooting
```
This yields:

```
shooting(m,n,xo,xf,yo,yf)
   This routine solves one non-linear first-order ordinary differential
   equation boundary value problem using the shooting method.

   m = 1 for Euler's method
   m = 2 for Classical Runge-Kutta 4rth order method
   n = number of steps
   xo = starting value of x
   xf = ending value of x
   yo(1) = initial condition at xo
   yo(2) = estimated initial derivative at xo
   yf = final condition at xf

   The differential equation must appear in the file 'shootinput.m'
   This program creates an output data file 'shoot.out'

   Author:  David Keffer   Date:  October 23, 1998
```

Hopefully, it is clear from this information how to use sysode.m at the command line.
For example,

```
shooting(2,100,0,10,[2,3],4)
```

uses the shooting method to solve the ODE BVP over the range $0 £ x £ 10$ with 100 intervals, and the boundary condition $y_1(x = 0) = 2$ and $y_2(x = 10) = 4$. The second initial condition in the brackets is *just a guess*. As with any method, the better the initial guess, the more likely you are to converge to the correct solution.

The ODE-BVP to be solved by the code shooting.m must be entered in the file shootinput.m. This file can be as simple as:

```
function dydt = odeivpn(x,y,nvec);
   c=[-1.0,-2.0,2.0,0.0];
   dydt(1) = y(2);
   dydt(2) = c(1)*y(2) + c(2)*y(1) + c(3)*sin(x) + c(4);
```

43.11 Combining Runge-Kutta and Newton-Raphson Methods

Let's consider solving the system of a fluid draining from a tank through a pipe:

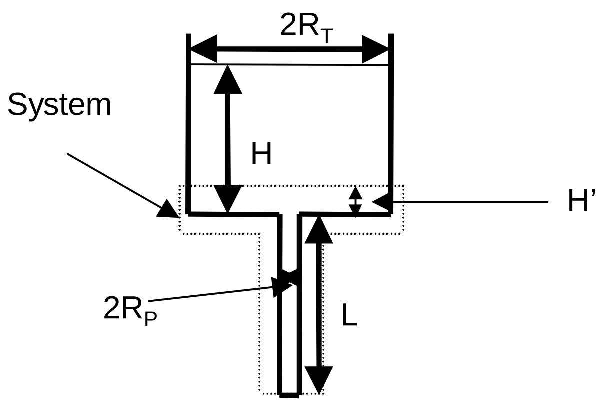


Figure One. Efflux from a Tank

If we neglect all terms in the mechanical energy balance except (1) pressure drop, (2) potential energy, and (3) frictional head loss due to flow in the pipe (assuming turbulent flow in the pipe), we have the ODE:

$$\frac{dH}{dt} = -\left(\frac{g\rho^{0.25}D_P^{4.75}\left(1+\frac{H}{L}\right)}{2(0.0791)\mu^{0.25}D_T^{3.5}}\right)^{4/7}$$

with the initial condition (the initial height of water in the tank)

$$\mathbf{H(t=0) = H_o}$$

which can be solved using sysode.m as shown above. (In fact the second example sysodeinput.m file in the ODE IVP problems given above is intended to solve precisely this case.) Note that this equation is non-linear. Our function, H(t), is raised to the 4/7 power. In the Runge-Kutta method, we would know $\mathbf{H(t)}$ and we could calculate $\left(\frac{dH}{dt}\right)$ directly.

Now, consider the case where we also include kinetic energy in our mechanical energy balance. Then we obtain one first-order ODE of the form:

$$-g(L+H)+\left(\frac{dH}{dt}\right)^{1.75}\left[\frac{2(0.0791)\mu^{0.25}LD_T^{3.5}}{\rho^{0.25}D_P^{4.75}}\right]+\left(\frac{dH}{dt}\right)^{2}\left[\frac{\left(\frac{D_T^4}{D_P^4}\right)-1}{2}\right]=0$$

where we now have a third term (the kinetic energy term). We can't use sysode.m as it is given to solve this problem for a specific reason. What is this reason? Well, examine the equation; now $\left(\frac{dH}{dt}\right)$ appears in a non-linear fashion within the equation. In the Runge-Kutta method, we would know $H(t)$ but we cannot calculate $\left(\frac{dH}{dt}\right)$ directly because we can't arrange the equation into the form

$$\left(\frac{dH}{dt}\right)=f(t,H(t)).$$

For a given value of t and $H(t)$, we must use the Newton-Raphson method (or an equivalent method for finding the roots of a non-linear algebraic equation) to find the value of $\left(\frac{dH}{dt}\right)$. So, what must we do?

Remember, in the Runge-Kutta method, the k's are values of the derivative evaluated at different point. In order to get these k's, we need to find $\left(\frac{dH}{dt}\right)$. So, in the Runge-Kutta fourth-order method, we would have to use Newton-Raphson 4 times (since there are 4 k's.) If we were going to divide our time range of interest into 100 Runge-Kutta intervals, we would have to use the Newton-Raphson method, 400 times.

In MATLAB, that is done easily enough, we could just call rootfinder or some other non-linear equation solving routine, in sysodeinput.m. Thus, we would be returning the value of $\left(\frac{dH}{dt}\right)$ just as sysode.m requires. However, the problem arises, that we need good initial guesses when we use a technique like Newton-Raphson. If we have to use the Newton-Raphson method 400 times, then we need 400 different good initial guesses. The initial guess for $\left(\frac{dH}{dt}\right)$ at time t = 0, may not be a good guess at time t = 100. However, not to despair, we are in luck. If we can get a good initial guess for $\left(\frac{dH}{dt}\right)$at t = 0, then we can solve for $\left(\frac{dH}{dt}\right)$at t = 0. If our Runge-

Kutta step is small enough, then we can use the converged solution for $\left(\frac{dH}{dt}\right)$ at t = 0 as our initial guess for $\left(\frac{dH}{dt}\right)$ at t = 1. Then, we can use the Runge-Kutta method to determine $H(t)$ at t = 1. We can then repeat the process and use our converged solution for $\left(\frac{dH}{dt}\right)$ at t = 1 as our initial guess for $\left(\frac{dH}{dt}\right)$ at t = 2. This technique frequently works and does work for the mechanical energy balance shown above.

There is an example of a code, based very closely on sysode.m that solves this equation on my website for the ChE 310 Efflux from a tank experiment.

The gist of the code is as follows:

(1) the main program in sysode.m doesn't change.
(2) The Runge-Kutta function inside the sysode.m file changes as follows:

```
%
%  This routine computes the 4rth order R-K method
%
function h = rk4eval310(t,dt,n,ho,dhdto)
%
h = zeros(n+1,1);
h(1) = ho;
k = zeros(4,n+1);
for i =  1:n
   k(1,i) = odeivp310(t(i),h(i),dhdto);
   dhdto = -k(1,i);
   k(2,i) = odeivp310(t(i)+dt/2,h(i)+dt/2*k(1,i),dhdto);
   dhdto = -k(2,i);
   k(3,i) = odeivp310(t(i)+dt/2,h(i)+dt/2*k(2,i),dhdto);
   dhdto = -k(3,i);
   k(4,i) = odeivp310(t(i)+dt,h(i)+dt*k(3,i),dhdto);
   dhdto = -k(4,i);
   h(i+1) = h(i)+dt/6*(k(1,i)+2*k(2,i)+2*k(3,i)+k(4,i));
end
```

Note, that each time we call the function odeivp.m (used instead of sysodeinput.m but serving the same purpose), we pass the initial guess for the derivative, dhdto. Also note that the value of dhdto is updated each time to reflect our latest value of the derivative.

The function, odeivp.m just returns the value of the derivative, but in this case, calls fzero to get it.

```
function dhdt = odeivp310(tt,hh,dhdto);
% L is pipe length (cm)
```

```
L= 15.24;
% dp is pipe diameter (cm)
dp = 0.47752;
%
% parameters that are constant for all runs
%
% g is gravity (cm/s^2)
g = 980.67;
% mu is viscosity (g/(cm*s))
mu = 0.008817;
% rho is density (g/cm^3)
rho = 0.9970;
% dta = tank diameter (cm)
dta = 15.24;
% hp is h prime, the amount left in the tank at the end of the run (cm)
hp = 2.54;
%
%  solve the ODE for dhdt
%
   dhdt = -fzero('f310b',dhdto,1.e-6,0,g,L,mu,dta,rho,dp,hh,hp);
```

Because the MATLAB instrinsic function, fzero, requires the function to be in a different file, we actually place the function in the file: f310b.m, shown below.

```
function f=f310b(dhdt,g,L,mu,dta,rho,dp,h,hp);
f=-g*(L+h)+dhdt^1.75*(2*0.0791*mu^0.25*L*dta^3.5)/
              (rho^0.25*dp^4.75)+dhdt^2*0.5*(dta^4/dp^4-1);
```

In this way, we can combine the Runge-Kutta method for solving ODE IVPs and the Newton-Raphson method for finding the roots of non-linear equations, to solve ODE IVPs where the derivative is in a non-linear functional form, such that we cannot manipulate it to yield the traditional

$$\left(\frac{dH}{dt}\right) = f(t,H(t)).$$