Numerical Methods for Engineers: A practical approach

Jonathan Haydak

October 28, 2017

Contents

Part I
Differential Equations

0.1 Derivative approximations

Like so many things in math, we start by taking a function $f(x)$ and expanding it around some $x = x_0$ with a taylor series:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 \tag{0.1}$$

Neglecting the issue of convergence, we know that this expression is true for all x in a small neighborhood of x_0. Next, if we stay within a small neighborhood of x_0, we see that the $(x - x_0)^2$ will be much larger than the remaining terms due to the fact that a small number repeatedly raised to higher powers rapidly approaches zero. Given this, we truncate the Taylor series and make the approximation:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \mathcal{O}\left((x - x_0)^2\right) \tag{0.2}$$

It is worth taking a second to discuss what these equations means. Formally, it means that regardless of the choice of x_0, there exists a c such that

$$\begin{aligned} |f(x) - f_{approx}(x)| &\leq c|x - x_0|^2 \\ |f(x) - (f(x_0) + f'(x_0)(x - x_0))| &\leq c|x - x_0|^2 \end{aligned} \tag{0.3}$$

What this is saying is that the error in our approximation of $f(x)$ is proportional to $(x - x_0)^2$. Moreover, this says that we can make our approximation arbitrarily accurate by making $(x - x_0)^2$ small enough. In general, the exact functional form of $mathcalO(x - x_0)^2$ and we can play fast and dirty with algebraic manipulations involving this (and other error terms). Namely, we can multiply it by a constant, and smaller terms, and it still remains the same:

$$\begin{aligned} \mathcal{O}(x - x_0) &= \mathcal{O}(x_0 - x) \\ \mathcal{O}(x - x_0) &= -\mathcal{O}(x - x_0) \\ \mathcal{O}(x - x_0) &= 50\mathcal{O}(x - x_0) \\ \mathcal{O}(x - x_0) &= \mathcal{O}(x - x_0) + (x - x_0)^3 - (x - x_0)^4 \\ \frac{\mathcal{O}((x - x_0)^2)}{x - x_0} &= \mathcal{O}(x - x_0) \end{aligned}$$

and so on.

Now, suppose we want to approximate $f'(x_0)$ using (0.2). This gives us:

$$f'(x_0) \approx \frac{f(x) - f(x_0)}{x - x_0} + \mathcal{O}(x - x_0) \tag{0.4}$$

If we evaluate the above expression at $x = x_0 + \Delta t$, we end up with a formula for approximating the derivative at any point x_0 whose error is proportional to Δx.

$$f'(x_0) \approx \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} + \mathcal{O}(\Delta x) \tag{0.5}$$

(0.5) is called a **forward difference quotient**. We say it is first order accurate in Δx because its error is $\mathcal{O}(\Delta x)$. If instead we were to evaluate (0.2) at $x = x_0 - \Delta t$, we would get a **backward difference quotient**:

$$f'(x_0) \approx \frac{f(x_0) - f(x_0 - \Delta x)}{\Delta x} + \mathcal{O}(\Delta x) \tag{0.6}$$

Can we do better though? Yes, but we have to include more terms in the approximation. For example, what if we were to average the forward and backward difference quotient. Would we end up with a more accurate approximation. If you naively add up the terms in (0.5) and (0.5) and divide by 2, you might be tempted to say that the error on this type of scheme is still first order. However, there is cancellation going on in the error terms that you will miss if you do not expand out the difference quotients using more terms. Using a second order taylor series, we can write out the following expansions:

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{f''(x)}{2}\Delta x^2 + \mathcal{O}(\Delta x^3) \tag{0.7}$$

$$f(x - \Delta x) = f(x) - f'(x)\Delta x + \frac{f''(x)}{2}\Delta x^2 + \mathcal{O}(\Delta x^3) \tag{0.8}$$

Taking (0.7) - (0.8) and rearranging gives the following:

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2) \tag{0.9}$$

(0.9) is called a **central difference quotient**. It is the average of the forward and backward quotients and we see that it is second order accurate.

In a similar manner to how we obtained the central difference quotient, you can obtain higher order accuracy forward, backward, and central difference quotients by writing out taylor expansions of $f(x+\Delta x)$, $f(x-\Delta x)$, $f(x+2\Delta x)$, $f(x-2\Delta x)$ and adding the appropriate multiples of the corresponding equations to get terms to cancel. Some common difference schemes and their accuracy are summarized in the table below.

	Discrete approximation	Error
$f'(x)$	$\frac{f(x+h)-f(x)}{h}$	$\mathcal{O}(h)$
	$\frac{f(x)-f(x-h)}{h}$	$\mathcal{O}(h)$
	$\frac{f(x+h)-f(x-h)}{2h}$	$\mathcal{O}(h^2)$
	$\frac{-.5f(x+2h)+2f(x+h)-1.5f(x)}{h}$	$\mathcal{O}(h^2)$
	$\frac{.5f(x-2h)-2f(x-h)+1.5f(x)}{h}$	$\mathcal{O}(h^2)$
	$\frac{\frac{1}{12}f(x-2h)-\frac{2}{3}f(x-h)+\frac{2}{3}f(x+h)-\frac{1}{12}f(x+2h)}{h}$	$\mathcal{O}(h^4)$
$f''(x)$	$\frac{f(x+2h)-2f(x+h)+f(x)}{h^2}$	$\mathcal{O}(h)$
	$\frac{-f(x-2h)+2f(x-h)-f(x)}{h^2}$	$\mathcal{O}(h)$
	$\frac{f(x+h)-2f(x)+f(x-h)}{h^2}$	$\mathcal{O}(h^2)$
	$\frac{-\frac{1}{12}f(x+2h)+\frac{4}{3}f(x+h)-\frac{5}{2}f(x)+\frac{4}{3}f(x-h)-\frac{1}{12}f(x-2h)}{h^2}$	$\mathcal{O}(h^4)$

1 Methods

The most general form of a differential equation (O.D.E) takes the form:

$$f_n(x,y)\frac{\mathrm{d}^n y}{\mathrm{d}x^n}+f_{n-1}(x,y)\frac{\mathrm{d}^{n-1}y}{\mathrm{d}x^{n-1}}+\ldots+f_2(x,y)\frac{\mathrm{d}^2 y}{\mathrm{d}x^2 y}+f_1(x,y)\frac{\mathrm{d}y}{\mathrm{d}x}+f_0(x,y)=0 \tag{1.1}$$

If each f_i in (1.1) is a function of x only, that is, $f_i(x,y)=f_i(x)$, then we say that the differential equation is linear. We will see later on in this section that linear equations are very nice to deal with and that like so many things in math non-linearities can potentially make our lives very difficult. For (1.1) to be solvable, we must be additional information in the form of boundary conditions or initial conditions. The number of additional constraints we must be given for the equation to have a unique solution is equal to the order of the equation. A first order differential equation would require only one piece of additional information, but a second order equation would require two.

1.1 Initial Value Problems

Let us start out by considering a relatively easy class of differential equations: first order linear initial value problems. This type of equation will have the form:

$$\frac{\mathrm{d}y}{\mathrm{d}t} + f(y,t) = 0, \quad y(t = t_0) = y_0 \tag{1.2}$$

The basic idea is to discretize this differential equation and to come up with a scheme that allows us to solve for the next time level.

1.1.1 Explicit Methods

Forward Euler First, let us try approximating the derivative in (1.2) with a simple forward difference. We have previously shown in the first section that approximating the derivative in this way will give us $\mathcal{O}(\delta t))$ error, and we end up with the same error fo(1.1) when we approximate it with this type of difference. As you can imagine, we can achieve better errors by using higher order errors, and we will visit this concept shortly.

Suppose we create a grid of equally spaced mesh points for t such that $t_n = n\Delta t$ for $n = 1, 2, \dots, \frac{T}{\Delta t}$ where T is the total time interval for which we seek a solution. We will define y_n to be the calculated solution at time level n, that is, $y_n = y(t = t_n) = y(t = n\Delta t)$. To apply an explicit method, we approximate everything at the n_th time level (as opposed to the (n+1)th time level). This essentially means wherever we see $g(t)$ we replace it with g_n and wherever we see t we replace it with t_n. We can approximate (1.2) as

$$\frac{y_{n+1} - y_n}{\Delta t} + f(y_n, t_n) = 0 \tag{1.3}$$

isolating for y_{n+1},

$$y_{n+1} = y_n - \Delta t f(y_n, t_n), \quad y(t_0) = y_0 \tag{1.4}$$

This gives us an explicit algorithm to proceed forward in calculating y at the $n+1$ time level using information from the previous time step. Since we are given an initial condition, proceeding from the initial condition to later timesteps is relatively straightforward:

$$\begin{aligned} y_1 &= y_0 - \Delta t f(y_0, t_0) \\ y_2 &= y_1 - \Delta t f(y_1, t_1) \\ y_3 &= y_2 - \Delta t f(y_2, t_2) \\ &\vdots \\ y_N &= y_{N-1} - \Delta t f(y_{N-1}, t_{N-1}) \end{aligned}$$

The method we just described has a special name: Euler's method. Or, more precisely, Forward Euler's method. What step size Δt should we use? There is no one-size-fits-all answer to this and really depends on the problem. We will have much more to say about step sizes later on, but to start out with the idea is to basically just choose Δt small enough so that we get a converged solution. Keep in mind that this method approximate using a derivative with first order error in Δt, so we expect the solution we come up with to have first order error as well. This means that we should probably prefer small Δt.

Although you can think of Euler's method as a first order approximation of the derivative, an alternative and perhaps more useful view is in terms of linear approximations. For example, consider (1.1) again. We can write this equation in a slightly different form:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = g(y,t), \qquad y(t = t_0) = y_0$$

If we perform a first order taylor approximation about $t = t_0$ and ignore the remainder, then we have for small Δt that:

$$y(t_0 + \Delta t) = y(t_0) + \Delta t g(y_0, t_0)$$

From this, we see that Euler's method essentially estimates the function by calculating the current slope, and then taking a tiny step forward along this slope, and repeating this process. Solutions without sharp curvature are easier to converge than those that have a lot of curvature.

Ex 1.1. Given the differential equation

$$g'(t) + 2tg(t) = 0, \qquad g(0) = .5$$

Use the forward Euler's method to calculate g on the interval $0 \le t \le 2$. Investigate what happens for various choices of Δt.

Solution: First, we discretize the equation using a forward difference to approximate the time derivative.

$$\frac{g_{n+1} - g_n}{\Delta t} + 2t_n g_n = 0$$

$$g_{n+1} = g_n - \Delta t 2t_n g_n$$

$$g_{n+1} = g_n(1 - 2n\Delta t^2)$$

This gives us a straightforward algorithm for computing g at later time levels, and we can code up a solution using Matlab. We can even solve this equation analytically and compare the analytic solution to our approximations. Using an integration factor, we find that

$$g(t) = \frac{1}{2}e^{-t^2}$$

and we can check that this is correct by plugging it in

$$g'(t) + 2tg(t) = -te^{-t^2} + te^{-t^2} = 0$$

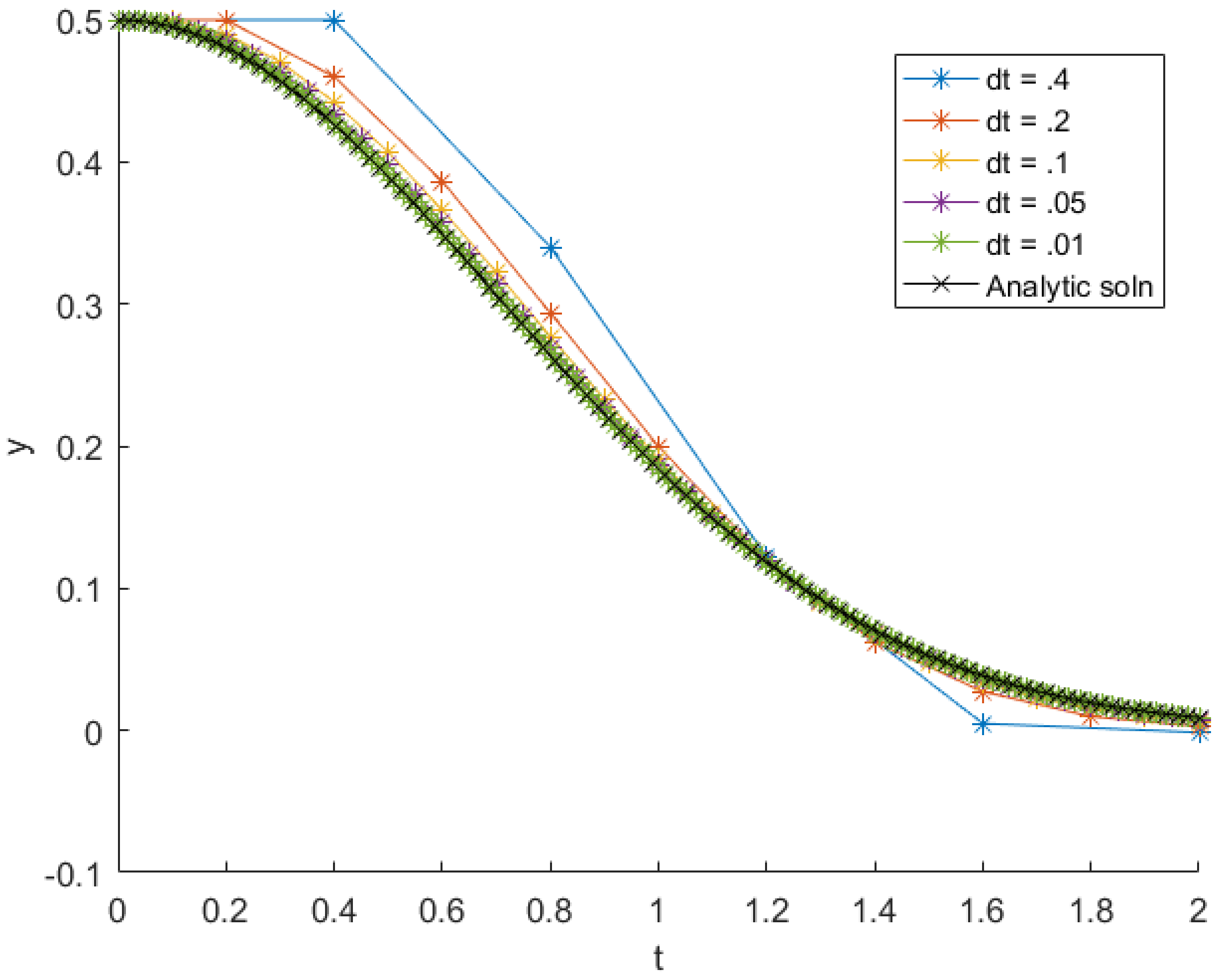


Figure 1: Forward Euler method with various timesteps.

From this, we see that we get relatively good qualitative agreement very quickly as the step size is decreased. We also see that as $\Delta t \to 0$, the solution converges to the analytic solution.

Summary To derive an explicit scheme for a differential equation, we can write the correspond discrete equation by approximating the derivative(s) using your favorite difference quotient and than evaluating all the other terms at the nth time level. You should be able to isolate the y_{n+1} term and write it explicitly in terms of other variables.

1.1.2 Improving Euler's Method

Using higher order derivative approximations We used a first order difference quotient in the previous example, but we know that has a first order truncation error. We can approve the accuracy of our scheme by using more accurate approximations of the derivative. For example, let us try using a centered difference quotient. In this case, (1.2) becomes:

$$\frac{y_{n+1} - y_{n-1}}{2\Delta t} + f(y_n, t_n) = 0 \tag{1.5}$$

$$y_{n+1} = y_{n-1} - f(y_n, t_n) \tag{1.6}$$

Wait! We only have one given initial condition, but it seems like to start this scheme we need to know two consecutive values of y_i. This is only a minor problem: we can estimate either y_{-1} or y_1 using Euler's method and then proceed with this scheme.

1.2 Implicit Methods

The central idea behind explicit methods is to estimate y_{n+1} by performing a taylor expansion around y_n. If instead we were to estimate y_{n+1} with a taylor expansion around y_{n+1}, we would end up with an **implicit** method. It is relatively straightforward to convert an explicit method to an implicit method: for every term that does not contain a derivative of y, replace the time step n with $n + 1$. For instance, to get the simplest implicit method, Implicit Euler, 1.2 becomes

$$\frac{y_{n+1} - y_n}{\Delta t} + f(y_{n+1}, t_{n+1}) = 0 \tag{1.7}$$

Notice now that isolating y_{n+1} is more tricky because it also appears in the $f(y_{n+1}, t_{n+1})$ term. If the differential equation is non-linear, implicit methods can be rather difficult and computationally expensive to solve depending on the nature of the non-linearity. In the case of linear equations and most ordinary differential equations that engineers encounter, the method is not too different from the normal euler method. At this point, you may well be wondering something along the lines of "Why bother?" It turns out that implicit methods have desirable stability and convergence properties for many differential equations in which achieving reasonable accuracy with an explicit method would require unrealistically small step sizes. These types of differential equations are called "stiff" and can be solved more efficiently using an implicit method and a larger step size.

To fully illustrate the different between explicit and implicit methods, consider the following example.

Ex 1.2. Solve $y'(x) + 4y(x) = 2x\cos(20x)$ on the interval $0 \leq x \leq 4$ given that $y(0) = 2$. Use the implicit Euler method and examine what happens to the convergence of the solution as the step size is varied.

Solution: Upon inspection of this differential equation, we see that that the cosine term has a frequency of $\frac{2\pi}{20} = .314159$. Because the period is significantly lower than the length of the interval we are solving y over, we should be cautious and consider that the solution might show significant curvature. Thus, an implicit method is appropriate for solving this problem.

First, we convert our differential equation into a discrete approximation:

$$\begin{aligned}
&\frac{y_{n+1} - y_n}{\Delta x} + 4y_{n+1} = 2x_{n+1}\cos(20x_{n+1}) \\
&y_{n+1} - y_n + 4\Delta x y_{n+1} = 2\Delta x x_{n+1}\cos(20x_{n+1}) \\
&(1 + 4\Delta x)y_{n+1} = y_n + 2\Delta x x_{n+1}\cos(20x_{n+1}) \\
&y_{n+1} = \frac{y_n + 2\Delta x x_{n+1}\cos(20x_{n+1})}{1 + 4\Delta x}
\end{aligned}$$

Since we are given $y_0 = 2$, we now have a straightforward way of calculating this scheme.

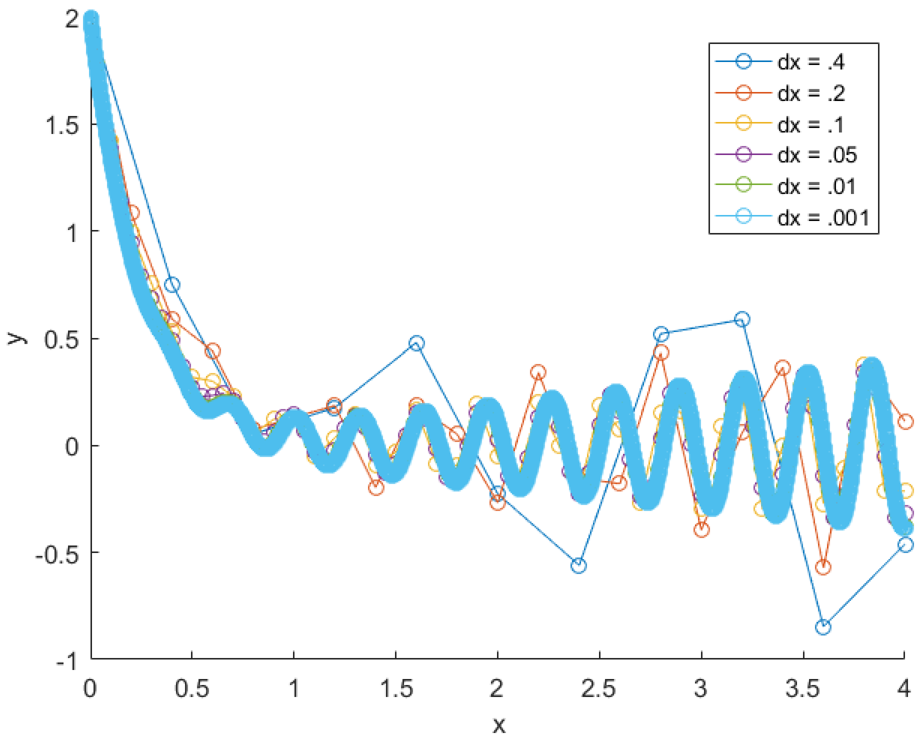


Figure 2: Implicit Euler with various timesteps.

1.3 Predictor-Corrector Methods

1.3.1 Modified Euler/Heun Method

Numerical methods for differential equations are a lot like ice cream. There are so many options to choose from and choosing a flavor can be very difficult. However, humans have a knack for ingenuity and discovered long ago that ice cream can be mixed so that I can get half chocolate and half vanilla or any combination that I want. In much the same way, the choice of a numerical scheme can seem overwhelming. Just like with ice cream, it turns out that combining implicit and explicit methods works out well. These schemes are called "Predictor-Corrector Methods". They combine the the best properties of both explicit and implicit methods and are very popular in practice.

The basic concept behind this method is to make a prediction using an explicit scheme expanded about y_i to "predict" a later point such as $y_{i+.5}$ or y_{i+1} Let's call this prediction y^*. The next step is to "correct" the estimate of y_{i+1} using a weighted average of explicit methods expanded about y_i and y^*. For higher order methods of this type, there can be more than one prediction step.

If that description sounds abstract, that is okay. Formally stating this algorithm should hopefully illustrate what is going on. The simplest predictor-corrector method is called the "modified Euler method" or "Heun's" method. Suppose we are attempting to solve

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(y, x), \qquad f(x = x_0) = a \tag{1.8}$$

with a being a constant. The first step is to use Euler's method to "predict" y_{n+1}:

$$y^*_{n+1} = y_n + \Delta x f(y_n, x_n) \tag{1.9}$$

The second step is to "correct" the estimate for y_{n+1} using the prediction:

$$y_{n+1} = y_n + \frac{\Delta x}{2}\left(f(y^*_{n+1}, x_{n+1}) + f(y_n, x_n)\right) \tag{1.10}$$

Although we are doing more computations with this algorithm than an implicit or euler, it turns out that the increase in the number of computations corresponds to a negligible increase in computation time. If you are familar with complexity analysis, this means that these methods have the same run time as the simpler implicit or explicit methods. To illustrate the utility of predictor corrector methods, let us consider applying this scheme to a solution with significant curvature.

Ex 1.3. Solve $y'(x) + (y(x))^3 = 2x\cos^2(25x)$ on the interval $0 \leq x \leq 10$ given that $y(0) = 2$. Use the modified Euler method and examine how the convergence depends on Δx.

Solution: The first step consists of a prediction:

$$y^*_{n+1} = y_n + \Delta x\left(2x_n\cos^2(25x_n) - (y_n)^3\right)$$

The second step is the correction:

$$y_{n+1} = y_n + \frac{\Delta x}{2}\left(2x_{n+1}\cos^2(25x_{n+1}) - (y^*_{n+1})^3 + 2x_n\cos^2(25x_n) - (y_n)^3\right)$$

From which we can now code up the scheme. As an illustration of how one might go about coding this up, a sample Matlab code is given.

```matlab
hold all
[x1,y1] = solve_for_y(10,.4);
[x2,y2] = solve_for_y(10,.2);
[x3,y3] = solve_for_y(10,.1);
[x4,y4] = solve_for_y(10,.05);
[x5,y5] = solve_for_y(10,.01);
[x6,y6] = solve_for_y(10,.001);
plot(x1,y1,'-o',x2,y2,'-o',x3,y3,'-o',x4,y4,'-',x5,y5,'-',x6,y6,'-')
xlabel('x')
ylabel('y')
legend('dx = .4', 'dx = .2', 'dx = .1', 'dx = .05', 'dx = .01', 'dx = .001' )
axis([0 10 .8 2.6])
function [xvec,y] = solve_for_y(T,dx)
%T is the total time to solve over, ie, (0,T)
%dx is step size
%initial condition is hard-coded in
y0 = 2;

N = ceil(T/dx);
xvec = 0:dx:N*dx;
y = zeros(1,N+1); %value of y at index i corresponds to y_{i-1}, ie,
% y = [y_0 y_1 y_2 ... y_N]
y(1) = y0;
for i = 1:N
    %matlab starts indexing at 1, so have to shift by 1 here
    y_p = y(i) + dx * (2*xvec(i)*cos(25*xvec(i))^2 - y(i)^3);
    y(i+1) = y(i) + (dx/2) * (2*xvec(i+1)*cos(25*xvec(i+1))^2 - y_p^3 + 2*xvec(i) * ...
        cos(25*xvec(i))^2 - y(i)^3);
end
end
```

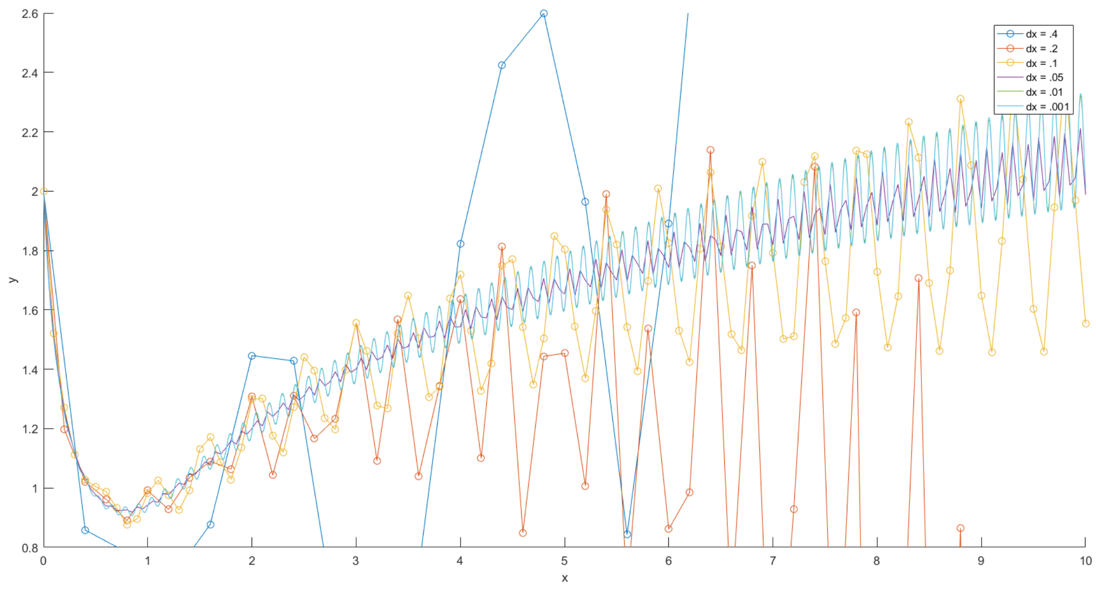


Figure 3: Heun's method with various step sizes. For functions with a lot of curvature, larger step sizes are unstable.

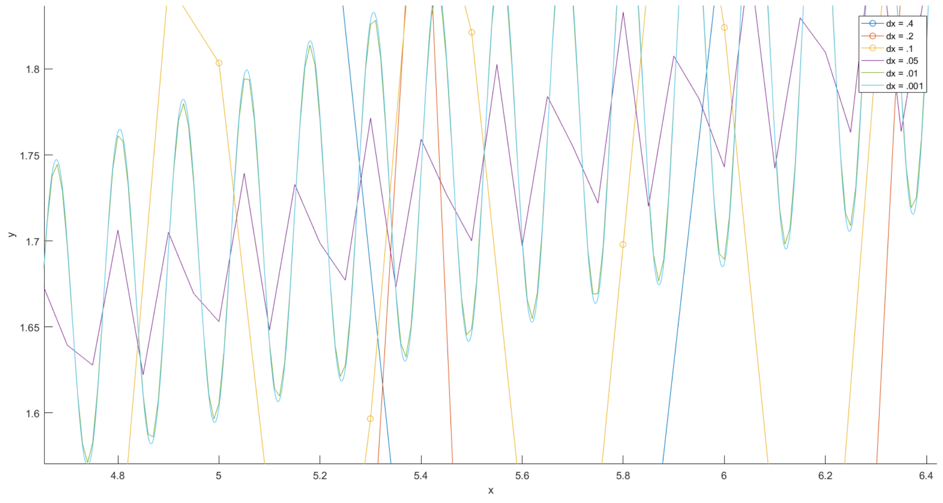


Figure 4: Zoomed-in section of previous plot

1.3.2 Runge-Kutta methods

Runge-Kutta (RK) methods are a family of techniques that use predictor-corrector steps to integrate ordinary differential equations. While there are actually many RK techniques, the most famous one is probably the

RK4 technique. The reason being it is relatively easy to implement, but gives an astonishing cumulative error of $\mathcal{O}(\Delta t^4)$. We will first describe the second order RK2 technique, and then the popular RK4. Suppose we are again given (1.8):

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x,y), \qquad f(x = x_0) = a \tag{1.11}$$

The approach with the RK2 method is to predict the value of y at the midpoint of the interval being considered, that is a step size of $\frac{1}{2}\Delta x$. We then approximate the slope of y at the midpoint using this prediction. Finally, we calculate the value for y_{n+1} using the value of the slope at this predicted point. Formally, tis can be described as:

$$\begin{aligned} h_1 &= \Delta x f(x_n, y_n) \\ h_2 &= \Delta x f(x_{n+\frac{1}{2}}, y_n + \frac{1}{2}h_1) \\ y_{n+1} &= y_n + h_2 \end{aligned}$$

This method gives an accumulated error of $\mathcal{O}(\Delta x^2)$. The more popular RK4 technique, which yields a global error of $\mathcal{O}(\Delta x^4)$ involves more prediction and correction steps, but can be formally described as:

$$\begin{aligned} h_1 &= \Delta x f(x_n, y_n) \\ h_2 &= \Delta x f(x_{n+\frac{1}{2}}, y_n + \frac{1}{2}h_1) \\ h_3 &= \Delta x f(x_{n+\frac{1}{2}}, y_n + \frac{1}{2}h_2) \\ h_4 &= \Delta x f(x_{n+1}, y_n + h_3) \\ y_{n+1} &= y_n + \frac{1}{6}\left(h_1 + 2h_2 + 2h_3 + h_4\right) \end{aligned}$$

In general, the coefficients are chosen so as to minimize the error that the method produces. However, if you find yourself in a pitch and you can't remember the correct coefficients for averaging or what points to predict, you can probably make up your own method and still get good accuracy (good, not optimal, but as an engineer all you should care about is good enough!) as long as enough prediction and correction steps are included and the coefficients add up to one. For example consider the following whimsical RK method that we shall call **Jon's method**:

$$\begin{aligned} h_1 &= \Delta x f(x_n, y_n) \\ h_2 &= \Delta x f(x_{n+\frac{1}{4}}, y_n + \frac{1}{4}h_1) \\ h_3 &= \Delta x f(x_{n+\frac{1}{2}}, y_n + \frac{1}{2}h_2) \\ h_4 &= \Delta x f(x_{n+\frac{3}{4}}, y_n + \frac{3}{4}h_3) \\ h_5 &= \Delta x f(x_{n+1}, y_n + h_4) \\ y_{n+1} &= y_{n+1} + \frac{1}{9}(h_1 + 2h_2 + 3h_3 + 2h_4 + h_5) \end{aligned}$$

Although the above method uses prediction/correction steps, we should not expect it to be fifth order accurate seeing as the points and weights were chosen arbitrarily rather than to minimize error. We should, however, expect this scheme to perform quite well because it is averaging over several intervals rather than at a single point.

Ex 1.4. Given the differential equaiton $x^4 y' + (x^3 + 2x)y = \sin(x) + 2$, solve for $y(10)$ given that $y(1) = 0$. Compare the accuracy of the RK2 method, the RK4 method, and Jon's method.

Solution: The analytic solution to this differential equation would most likely be very messy to find. To compare accuracies of these methods, we will calculate $y(x)$ using an RK4 scheme with a very small step size. It is worthwhile to point out that the RK4 method is actually robust and accurate enough that we can essentially use it as a gold standard in this case to approximate the true solution with "good enough" accuracy.

First, we express differential equation in terms of the slope.

$$y' = \frac{\sin(x)}{x^4} + \frac{2}{x^4} - \frac{y}{x} - \frac{2y}{x^3}$$

```
hold all
x0 = .75; y0=0; xn = 10;
[x1,y1] = myRK2(.25,x0,y0,xn);
[x2,y2] = myRK4(.25,x0,y0,xn);
[x3,y3] = my_jon(.25,x0,y0,xn);
[x4,y4] = myRK2(.05,x0,y0,xn);
[x5,y5] = myRK4(.05,x0,y0,xn);
[x6,y6] = my_jon(.05,x0,y0,xn);
[x_c,y_c] = myRK4(.001,x0,y0,xn);
plot(x1,y1,'r-o',x2,y2,'c-o',x3,y3,'b-o')
plot(x4,y4,'r-o','MarkerFaceColor','r','MarkerSize',3)
plot(x5,y5,'c-o', 'MarkerFaceColor','c','MarkerSize',3)
plot(x6,y6,'b-o', 'MarkerFaceColor','b','MarkerSize',3)
plot(x_c,y_c,'k-x','MarkerSize',2)
xlabel('x')
ylabel('y')
legend('dx = .25, RK2', 'dx = .25, RK4', 'dx = .25, jon', 'dx = .05, RK2', ...
    'dx = .05, RK4', 'dx = .05, jon','dx = .001, RK4')

function [x,y] = myRK2(dx,x0,y0,xn)
%dx - timestep, x0 - initial point, y0- initial point, xn - final x to be
%integrated over
x = x0:dx:xn;
y = zeros(size(x));
y(1) = y0;
numX = length(x);
for i = 2:numX
   h1 = dx * dydx(x(i-1),y(i-1));
   h2 = dx * dydx(x(i-1)+.5*dx,y(i-1)+.5*h1);
   y(i) = y(i-1) + h2;
end
end

function [x,y] = myRK4(dx,x0,y0,xn)
%dx - timestep, x0 - initial point, y0- initial point, xn - final x to be
%integrated over
x = x0:dx:xn;
y = zeros(size(x));
y(1) = y0;
numX = length(x);
for i = 2:numX
   h1 = dx * dydx(x(i-1),y(i-1));
   h2 = dx * dydx(x(i-1)+.5*dx,y(i-1)+.5*h1);
   h3 = dx * dydx(x(i-1)+.5*dx,y(i-1)+.5*h2);
   h4 = dx * dydx(x(i),y(i-1)+h3);
   y(i) = y(i-1) + (1/6) * (h1 + 2*h2 + 2*h3 + h4);
end
end

function [x,y] = my_jon(dx,x0,y0,xn)
```

```
%dx - timestep, x0 - initial point, y0- initial point, xn - final x to be
%integrated over
x = x0:dx:xn;
y = zeros(size(x));
y(1) = y0;
numX = length(x);
for i = 2:numX
   h1 = dx * dydx(x(i-1),y(i-1));
   h2 = dx * dydx(x(i-1)+.25*dx,y(i-1)+.25*h1);
   h3 = dx * dydx(x(i-1)+.5*dx,y(i-1)+.5*h2);
   h4 = dx * dydx(x(i-1)+.75*dx,y(i-1)+.75*h3);
   h5 = dx * dydx(x(i),y(i-1)+h4);
   y(i) = y(i-1) + (1/9) * (h1 + 2*h2 + 3*h3 + 2*h4 + h5);
end
end

function out = dydx(x,y)
out = sin(x)/x^4 + 2/x^4 - y/x - 2*y/x^3;
end
```

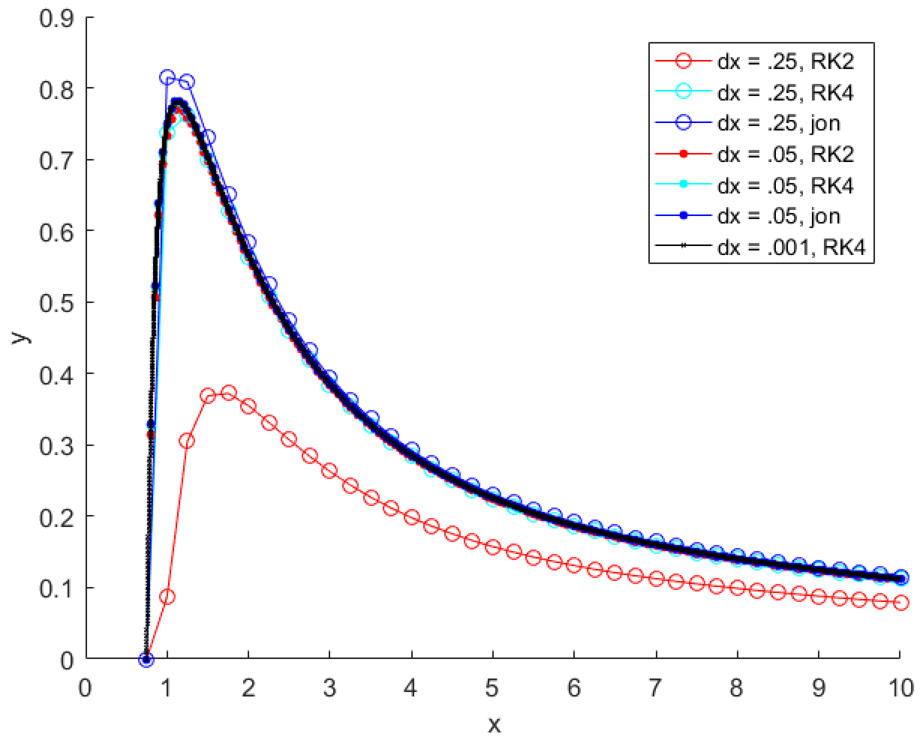


Figure 5: Comparison of several RK techniques.

Notice that the RK4 scheme performs relatively well, even with a large step size of .25. This is the power of predictor corrector techniques - it overcomes inaccuracies caused by sharp curvature by considering the slope at various points. In this example, we see that the RK2 technique does not have enough prediction/correction steps to overcome the curvature. The jon's method is better than the RK2 method but worse than RK4.

This is not surprising - it has one more prediction and correction step than RK4, but the sampling points and weighting was chosen arbitrarily instead of so as to minimize the error.

1.4 Adaptive Step Sizes

For most of the methods considered so far, it was assumed that a uniform step size was used. In most cases, this is a very natural and reasonable choice to make. However, for problems that are very computationally expensive and take a long period of time to run, efficiency becomes very important so as to reduce the number of steps needed. One of the best ways to do this is to use a step size that changes. While there are various rules for choosing the step size, the basic idea is to check the error at each step and to make it smaller if the error is to large.

1.4.1 Richardson Extrapolation

In this context, suppose that we are trying to solve a given IVP with a method that is k order accurate. Let A^* represent the exact solution. We have

$$\begin{aligned} A^* &= A(h) + Bh^k + Ch^{k+1} \\ &= A(h) + Bh^k + \mathcal{O}(h^{k+1}) \end{aligned} \tag{1.12}$$

where $A(h)$ is the approximate solution from the scheme with step size h and B is a constant. Going forward, we will sometimes neglect writing the higher order terms because they are (hopefully) sufficiently small, but do not forget that they are always there. Multiplying both sides by 2^k and using a step size of $\frac{h}{2}$

$$2^k A^* = 2^k A\left(\frac{h}{2}\right) + Bh^k + \mathcal{O}(h^{k+1}) \tag{1.13}$$

subtracting (1.13) - (1.12)

$$\begin{aligned} (2^k - 1)A^* &= 2^k A\left(\frac{h}{2}\right) - A(h) + \mathcal{O}(h^{k+1}) \\ A^* &= \frac{2^k A\left(\frac{h}{2}\right) - A(h)}{(2^k - 1)} + \mathcal{O}(h^{k+1}) \end{aligned} \tag{1.14}$$

what this says is that when integrating a differential equation, if we make two predictions for y_{n+1} with a step size h (one step) and $\frac{h}{2}$ respectively, we can use (1.14) to get an approximation that is $k+1$ accurate, an entire order more accurate than $A(h)$ and $A(\frac{h}{2})$. This is useful, but we still need to determine if it is *good enough*. To determine this, we can compute the error of $A(\frac{h}{2})$, which we know will be *less* accurate than (1.14). If this error is good enough, than the error from (1.14) will certainly be good enough.

$$\begin{aligned} E_{rr}[A(h/2)] &= A^* - A(h/2) \\ &= B(h/2)^k + \mathcal{O}(h)^{k+1} \\ &= B(h/2)^k \end{aligned} \tag{1.15}$$

Now solving for B

$$\begin{aligned} A^* &= A(h) + Bh^k + Ch^{k+1} \\ A^* &= A(h/2) + B(h/2)^k + C(h/2)^{k+1} \end{aligned}$$

subtracting these two equations

$$\begin{aligned} 0 &= A(h) - A(h/2) + Bh^k(1 - \frac{1}{2^k}) + \mathcal{O}(h^{k+1}) \\ B &= \frac{A(h) - A(h/2)}{h^k(1 - \frac{1}{2^k})} + \mathcal{O}(h^{k+1}) \end{aligned} \tag{1.16}$$

plugging (1.16) into (1.15)

$$E_{rr}[A(h/2)] = \frac{A(h) - A(h/2)}{\frac{1}{2^k} - 1} \tag{1.17}$$

With all this, we can now describe a method for integrating with an adaptive step size:

1. Start with a set step size h. Compute $A(h)$ and $A(h/2)$. In other words, compute y_{n+1} twice: once with step size h, and again with step size $(h/2)$ (this one will take two steps to reach y_{n+1}.
2. Calculate the error using (1.17) and compare it to your error tolerance. If the error is less than the tolerance, calculate y_{n+1} from $A(h)$ and $A(h/2)$ using (1.14).
3. If the error is greater than the tolerance, the integration will need to be repeated with a new timestep. If the error was within the tolerance, a new timestep for the next step should be predicted. In either care, the new step size can be estimated by solving

$$E_{rr}[A(h)] = Bh^k_{new,1}$$

4. Either repeat the previous integration or move onto the next one using

$$h_{new,2} = .9h_{new,1}$$

where the factor of .9 is added to help ensure that the desired accuracy is reached.

Ex 1.5. Given $y'(x) = (.01x^2 - 2)y^{.5} + e^{-x^2}y + x^2\sin^2(x)$ and the initial condition $y(2) = 0$, use an adaptive step to calculate $y(10)$. Compare the results to using a uniform step size.

Solution

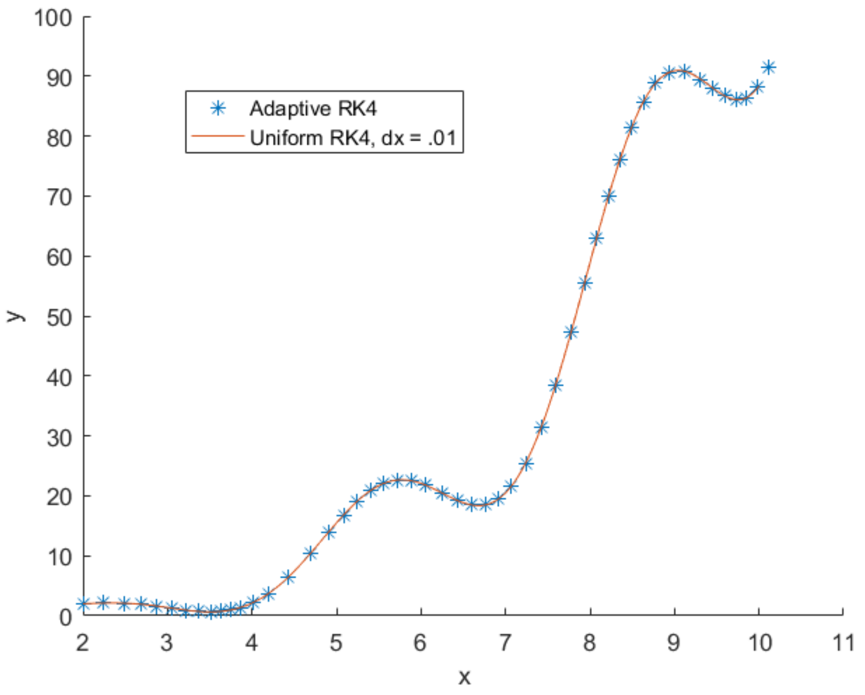


Figure 6: Comparison of adaptive and uniformly spaced RK4 method.

From Figure 6, we see that the adaptive RK4 method is able to ensure very high accuracy while making much larger step sizes. Notice that as the curvature increases, the step size becomes smaller, and when the curvature decreases the step size becomes bigger. Although each step of the adaptive algorithm involves more computations than a single step of a uniform RK4 method, the total number of computations is much lower overall.

```
close all
clc
clear
dx_init = .01;
dx_max = .5;
tol = 1e-6;
x0 =2; y0 = 2;
x_final = 10;
thisx = x0; thisy = y0; thisdx = .01;
k  = 5; %one iteration of RK4 is 5th order accurate
xvec = [x0];
yvec = [y0];
while thisx < x_final
    %calculate A(h)
    Ah = RK4_step(thisdx,thisx,thisy);
    Ah2 = RK4_step(thisdx/2,thisx,thisy);
    Ah2 = RK4_step(thisdx/2,thisx+thisdx/2,Ah2);
    B = (Ah - Ah2) / thisdx^k;
    B = B/(1 - (1/2^k));
    Err = B * (thisdx/2)^k;
    if abs(Err) <= tol %then accept
        thisy = ((2^k)*Ah2-Ah)/(2^k-1);
        thisx = thisx + thisdx;
        xvec = [xvec thisx];
        yvec = [yvec thisy];
    end
    thisdx = abs((tol*10/B))^(1/k);
    %thisdx = thisdx*.9;
end
hold all
plot(xvec,yvec,'*')
[x1,y1] = myRK4(.01,x0,y0,x_final);
plot(x1,y1,'-')
xlabel('x')
ylabel('y')
legend('Adaptive RK4','Uniform RK4, dx = .01')

function [x,y] = myRK4(dx,x0,y0,xn)
%dx - timestep, x0 - initial point, y0- initial point, xn - final x to be
%integrated over
x = x0:dx:xn;
y = zeros(size(x));
y(1) = y0;
numX = length(x);
for i = 2:numX
   h1 = dx * dydx(x(i-1),y(i-1));
   h2 = dx * dydx(x(i-1)+.5*dx,y(i-1)+.5*h1);
   h3 = dx * dydx(x(i-1)+.5*dx,y(i-1)+.5*h2);
   h4 = dx * dydx(x(i),y(i-1)+h3);
   y(i) = y(i-1) + (1/6) * (h1 + 2*h2 + 2*h3 + h4);
end
end


function [yn] = RK4_step(dx,x0,y0)
%dx - timestep, x0 - initial point, y0- initial point, xn - final x to be
%integrated over
```

```
h1 = dx * dydx(x0,y0);
h2 = dx * dydx(x0+.5*dx,y0+.5*h1);
h3 = dx * dydx(x0+.5*dx,y0+.5*h2);
h4 = dx * dydx(x0+dx,y0+h3);
yn = y0 + (1/6) * (h1 + 2*h2 + 2*h3 + h4);
end

function out = dydx(x,y)
out = (.01*x^2-2)*y^.5 + exp(-x^2)*y+x^2*sin(x)^2;
end
```

2 Higher Order Differential Equations

2.1 Direct explicit methods

Many of the lessons from previous sections are still for higher order differential equations. We can convert the equation into a discrete scheme with derivative approximations, and then either solve it with an explicit or implicit method. Since there is essentially nothing new to learn, it is best to start with an example.

Ex 2.1. Solve $x^2y''(x) + xy'(x) + (x^2 - 4)y = 0$ on the interval $1 \leq x \leq 10$ subject to the conditions $y(1) = .75$, $y'(1) = 1$. Use centered difference quotients and an explicit scheme.

Solution: First, we write the discrete form of the equation and isolate for y_{n+1}

$$\begin{aligned}
&x^2\frac{y_{n+1} - 2y_n + y_{n-1}}{\Delta x^2} + x\frac{y_{n+1} - y_{n-1}}{2\Delta x} + (x^2 - 4)y_n = 0 \\
&y_{n+1} - 2y_n + y_{n-1} + \left(\frac{\Delta x}{x}\right)\frac{y_{n+1} - y_{n-1}}{2} + \left(\frac{\Delta x^2}{x^2}\right)(x^2 - 4)y_n = 0 \\
&\left(1 + \frac{\Delta x}{2x}\right)y_{n+1} = \left(\frac{4\Delta x^2}{x^2} - \Delta x^2 + 2\right)y_n + \left(\frac{\Delta x}{2x} - 1\right)y_{n-1} \\
&y_{n+1} = \frac{\frac{4\Delta x^2}{x^2} - \Delta x^2 + 2}{1 + \frac{\Delta x}{2x}}y_n + \frac{\frac{\Delta x}{2x} - 1}{1 + \frac{\Delta x}{2x}}y_{n-1}
\end{aligned}$$

We are given y_0, but we also need y_1 or y_{-1}. The second boundary condition gives us an additional equation if we use a forward quotient:

$$\begin{aligned}
&\frac{y_1 - y_0}{\Delta x} = y'(0) \\
&y_1 = y_0 + \Delta x y'(0) \\
&y_1 = .75 + \Delta x
\end{aligned}$$

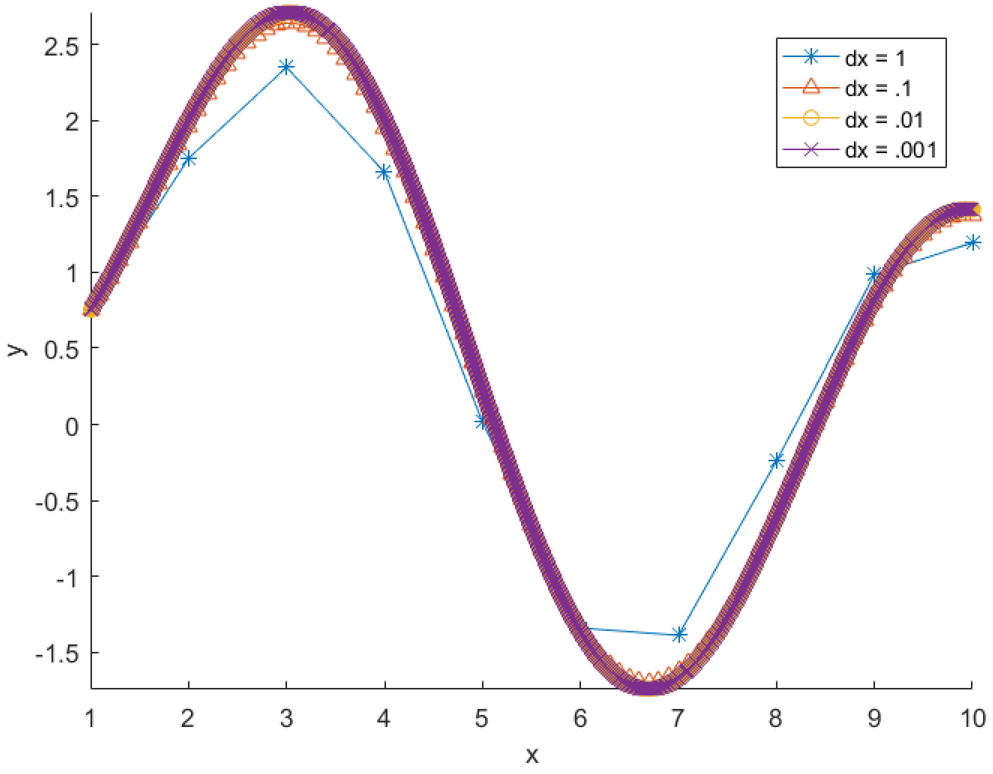


Figure 7: Centered scheme with various timesteps.

From Figure 7, we see that convergence is achieved rather quickly. Although the final expression is rather cumbersome, the process was virtually identical to that used for a first order ODE.

Ex 2.2. Given the equation $\frac{y''(x)}{1+x^2} - \sin(x)y'(x) + (x^2-1)y(x) = 0$ and the boundary conditions $y(1) = y(4) = 0$, find $y(x)$ on the interval $1 \leq x \leq 4$.

Solution We begin by writing the corresponding discrete equation. For simplicity, we will use centered difference schemes.

$$\frac{y_{n+1} - 2y_n + y_{n-1}}{(1+x_n^2)\Delta x^2} - \sin(x_n)\frac{y_{n+1} - y_{n-1}}{2\Delta x} + (x_n^2 - 1)y_n = 0$$

$$\left(\frac{1}{(1+x_n^2)\Delta x^2} - \frac{\sin(x_n)}{2\Delta x}\right) y_{n+1} + \left(x_n^2 - 1 - \frac{2}{(1+x_n^2)\Delta x^2}\right) y_n + \left(\frac{1}{(1+x_n^2)\Delta x^2} + \frac{\sin(x_n)}{2\Delta x}\right) y_{n-1} = 0$$

$$y_{n+1} = \frac{\left(-x_n^2 + 1 + \frac{2}{(1+x_n^2)\Delta x^2}\right)}{\left(\frac{1}{(1+x_n^2)\Delta x^2} - \frac{\sin(x_n)}{2\Delta x}\right)} y_n - \frac{\left(\frac{1}{(1+x_n^2)\Delta x^2} + \frac{\sin(x_n)}{2\Delta x}\right)}{\left(\frac{1}{(1+x_n^2)\Delta x^2} - \frac{\sin(x_n)}{2\Delta x}\right)} y_{n-1}$$

Because the boundary conditions do not allow us to get two consecutive values for y_k and y_{k+1}, we will have to solve this system as a matrix. For convenience, suppose

$$y_{n+1} = f_1(x_n)y_n - f_2(x_n)y_{n-1}$$

with f_1, f_2 being the appropriate functions just derived. We see that we if we divide the interval $[1, 4]$ into N points, we can write $N - 2$ equations:

$$\begin{aligned} f_2(x_1)y_0 - f_1(x_1)y_1 + y_2 &= 0 \\ f_2(x_2)y_1 - f_1(x_2)y_2 + y_3 &= 0 \\ f_2(x_3)y_2 - f_1(x_3)y_3 + y_4 &= 0 \\ &\vdots \\ f_2(x_{N-2})y_{N-3} - f_1(x_{N-2})y_{N-2} + y_{N-1} &= 0 \end{aligned}$$

Notice that we only have $N - 2$ equations but N variables. The final two equations from the boundary condition give us enough information to solve the system of equation. Given this, we can rewrite the system of equations as a matrix equation:

$$\begin{bmatrix} f_2(x_1) & -f_1(x_1) & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & f_2(x_2) & -f_1(x_2) & 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & f_2(x_3) & -f_1(x_3) & 1 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Thus, solving the problem now reduces to solving this matrix equation.

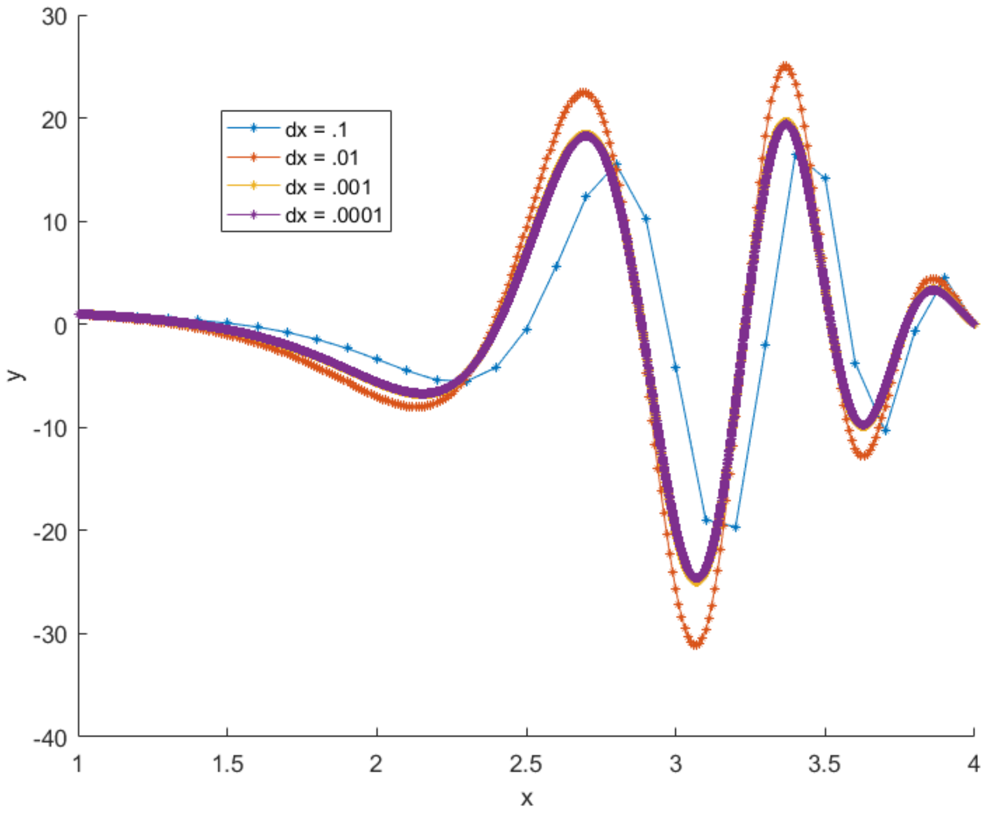


Figure 8: Centered scheme with various timesteps.

```
close all
clc
clear

y0 = 1;
yN = 0;

[x1,y1] = mysolver(.1,y0,yN);
[x2,y2] = mysolver(.01,y0,yN);
[x3,y3] = mysolver(.001,y0,yN);
[x4,y4] = mysolver(.0005,y0,yN);

hold all
plot(x1,y1,'*-','MarkerSize',3)
plot(x2,y2,'*-','MarkerSize',3)
plot(x3,y3,'*-','MarkerSize',3)
plot(x4,y4,'*-','MarkerSize',3)
xlabel('x')
ylabel('y')
legend('dx = .1','dx = .01','dx = .001','dx = .0001')


```

```
function [xn,yn] = mysolver(dx,y0,yN)
xn = 1:dx:4;
yn = zeros(size(xn));
N = length(xn);
A = zeros(N,N);
b = zeros(N,1);
for i=2:N-1
    A(i-1,i-1) = f2(xn(i-1),dx);
    A(i-1,i) = -f1(xn(i-1),dx);
    A(i-1,i+1) = 1;
end
A(N-1,1) = 1;
b(N-1) = y0;
A(N,N) = 1;
b(N) = yN;
yn = A\b;

end

function out = f1(x,dx)
num = -x^2 + 1 + 2/(1+x^2)/(dx^2);
denom = 1/(1+x^2)/(dx^2) - sin(x)/2/dx;
out = num/denom;
end

function out  = f2(x,dx)
num = 1/(1+x^2)/(dx^2) + sin(x)/2/dx;
denom = 1/(1+x^2)/(dx^2) - sin(x)/2/dx;
out = num / denom;
end
```

2.2 Conversion into a system of equations

The more standard way of solving higher order linear ordinary differential equation is to use a transformation of variables. Suppose we are given the equation

$$f_0(x)\frac{\mathrm{d}^n y}{\mathrm{d}x^n} + f_1(x)\frac{\mathrm{d}^{n-1} y}{\mathrm{d}x^{n-1}} + \ldots + f_{n-2}(x)\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + f_{n-1}(x)\frac{\mathrm{d}y}{\mathrm{d}x} + f_n(x) = 0 \tag{2.1}$$

Consider the transformation

$$\begin{aligned} v_0 &= y \\ v_1 &= \frac{\mathrm{d}y}{\mathrm{d}x} \\ v_2 &= \frac{\mathrm{d}^2 y}{\mathrm{d}x^2} \\ &\vdots \\ v_{n-2} &= \frac{\mathrm{d}^{n-2} y}{\mathrm{d}x^{n-2}} \\ v_{n-1} &= \frac{\mathrm{d}^{n-1} y}{\mathrm{d}x^{n-1}} \end{aligned}$$

Upon taking the derivative of each v_i term, we see that the problem is now transformed into solving a system of first order differential equations. Observe that the differential equation for each v_i term is almost trivial to write except for the last one. For this derivative, we use the original differential equaiton itself to obtain the corresponding term.

$$\begin{aligned}
v_0' &= v_1 \\
v_1' &= v_2 \\
v_2' &= v_3 \\
&\vdots \\
v_{n-2}' &= v_{n-1} \\
v_{n-1}' &= \frac{-f_n(x)}{f_0(x)} - \frac{f_{n-1}(x)}{f_0(x)} v_1 - \frac{f_{n-2}(x)}{f_0(x)} v_2 - \ldots - \frac{f_1(x)}{f_0(x)} v_{n-1}
\end{aligned}$$

Or, expressed in matrix form

$$\underbrace{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{n-2} \\ v_{n-1} \end{bmatrix}'}_{\mathbf{v}'} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & 0 & 0 & 0 & \ldots & 1 \\ \frac{-f_{n-1}}{f_0} & \frac{-f_{n-2}}{f_0} & \ldots & \ldots & \ldots & & \frac{-f_1}{f_0} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{n-2} \\ v_{n-1} \end{bmatrix}}_{\mathbf{v}} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \frac{-f_n}{f_0} \end{bmatrix}}_{\mathbf{b}}$$

From here, any appropriate method can be used to solve for v_0, which will also be a solution to the original differential equation.

Ex 2.3. Given the IVP problem $y''' - 2x^2 y'' + xy' = 0$ and the initial conditions $y(0) = 0$, $y'(0) = 6$, $y''(0) = -5.5$, solve for y(12).

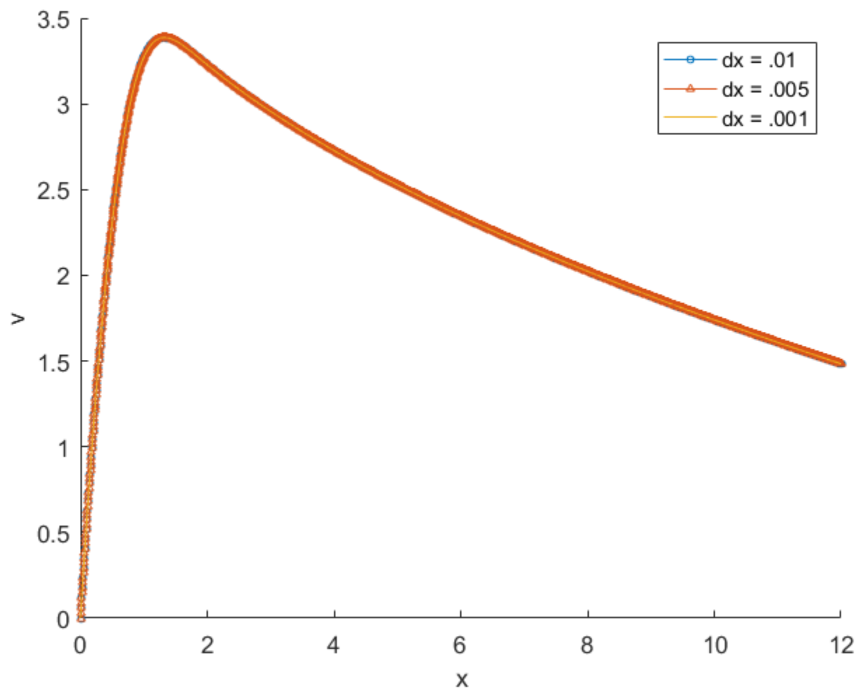
Solution: Using the transformations

$$\begin{aligned}
v_0 &= y \\
v_1 &= y' \\
v_2 &= y''
\end{aligned}$$

recasting the equation using the new variables

$$\begin{aligned}
v_0' &= v_1, \quad v_0(0) = 0 \\
v_1' &= v_2, \quad v_1(0) = 0 \\
v_2' &= 2x^2 v_2 - x v_1, \quad v_2(0) = 0
\end{aligned}$$

which can be easily integrated using an RK4 method.



```
close all
x0 = 0;
v0 = [0;6;-5.5];
xn = 12;
[x1,v1] = my_RK4(.01,x0,v0,xn);
[x2,v2] = my_RK4(.005,x0,v0,xn);
[x3,v3] = my_RK4(.001,x0,v0,xn);
hold all
plot(x1,v1,'-o','MarkerSize',3)
plot(x2,v2,'-^','MarkerSize',3)
plot(x3,v3,'-','MarkerSize',3)
xlabel('x')
ylabel('v')
legend('dx = .01','dx = .005','dx = .001')
function [xvec, v] = my_RK4(dx,x0,v0,xn)
%dx - timestep, x0 - initial point, v0- initial point, xn final time
xvec = x0:dx:xn;
v = [v0(1)];
for x0 = xvec(1:end-1)
    h1 = dx * dvdx(x0,v0);
    h2 = dx * dvdx(x0+.5*dx,v0+.5*h1);
    h3 = dx * dvdx(x0+.5*dx,v0+.5*h2);
    h4 = dx * dvdx(x0+dx,v0+h3);
    v0 = v0 + (1/6) * (h1 + 2*h2 + 2*h3 + h4);
    v = [v v0(1)];
end

end

function out = dvdx(x,v)
out = zeros(3,1);
```

```
out(1) = v(2);
out(2) = v(3);
out(3) = -2*x^2 * v(3) - x*v(2);
end
```

The previous example had initial data that was nice - we did not have to jump through any hoops to incorporate it. However, this is not always the case, as the following example will illustrate. If we are given initial data located at different points, we end up having to solve a matrix equation rather than just a recurrence relation.

Ex 2.4. Given $y'' + \frac{1}{x}y' + (x+1)y = 0$ and the boundary conditions $y(0) = 1$, $y(20) = 25$, find $y(5)$.

Solution: We begin by making the transformation $v_0 = y$ and $v_1 = y'$ to obtain

$$v_0' = v_1$$
$$v_1' = -(x+1)v_0 - \frac{1}{x}v_1$$
$$\underbrace{\begin{bmatrix} v_0 \\ v_1 \end{bmatrix}'}_{\mathbf{v}'} = \underbrace{\begin{bmatrix} 0 & 1 \\ -(x+1) & \frac{-1}{x} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} v_0 \\ v_1 \end{bmatrix}}_{\mathbf{v}}$$

Under this transformation, we obtain new boundary conditions as well: $v_0(1) = 1$, $v_0(20) = 25$.

We might be tempted to try and use an RK4 scheme. In this case, these boundary conditions do not allow us to easily do that. Notice now that we are working with vectors instead of scalars. To get the correct number of equations and unknowns to solve, we need to write out the equation at all $n-2$ points between the boundaries and solve simultaneously Letting $v_0^n = v_0(x_n)$ and $v_1^n = v_1(x_n)$, the discretized version of the differential equation using a forward Euler method becomes

2.3 Boundary Value Problems

In the worst case scenario, Initial Value Problems can usually be reduced to solving a system of linear algebraic equations. Boundary value problems are often a bit more involved. Generally, a boundary value problem (BVP) can be posed in the following manner: we are seeking to find a differentiable function $f(x)$ that satisfies

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = g\left(x, f, \frac{\mathrm{d}f}{\mathrm{d}x}\right) \qquad f(a) = \alpha, \quad f(b) = \beta \tag{2.2}$$

with α, β being scalars. For a boundary value problem, we cannot have a first order differential equation. This is because a first order ODE requires one piece of additional information to be unique. Two conditions would overspecify the system.

2.3.1 Guess and Check: Shooting method

The first method we will present for solving a BVP is the Shooting method. Despite the fancy name, it really amounts to guessing and checking for a solution. It is probably best illustrated with an example, but the idea is to solve an IVP that solves the BVP conditions. Instead of considering (2.2), consider the IVP problem

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = g\left(x, f, \frac{\mathrm{d}f}{\mathrm{d}x}\right) \qquad f(a) = \alpha, \quad \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=a} = w \tag{2.3}$$

where w is now a variable. Let $F(w, b)$ denote $f(b)$ that solves (2.3) for that value of w. We can solve (**??**) by finding the roots of $F(w, b) - \beta$. Clearly, this method is a lot of work. We have encountered examples of differential equations that are significantly difficult to solve once, but now we are going to have to solve an IVP possibly many times to get a clear picture of what $F(w, b) - \beta$ looks. If necessary, we could optimize this process using root finding methods. However, unless the differential equation is extremely stiff and costly to integrate, this is usually overkill.

Ex 2.5. Consider the differential equation $x^2y'' + x^2y' + (x^2 - 5)y = 0$ satisfying the boundary conditions $y(1) = 0$ and $y(5) = 1$.

Solution First, we need to solve the corresponding IVP. For the initial conditions, we can choose to work with either boundary, but we will arbitrarily choose to use the initial conditions $y(1) = 0$ and $y'(1) = w$. Using the transformation $v_0 = f$, $v_1 = f'$, we can express this equation as

$$\underbrace{\begin{bmatrix} v_0 \\ v_1 \end{bmatrix}'}_{\mathbf{v}'} = \underbrace{\begin{bmatrix} 0 & 1 \\ \left(\frac{5}{x^2} - 1\right) & -1 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} v_0 \\ v_1 \end{bmatrix}}_{\mathbf{v}}$$

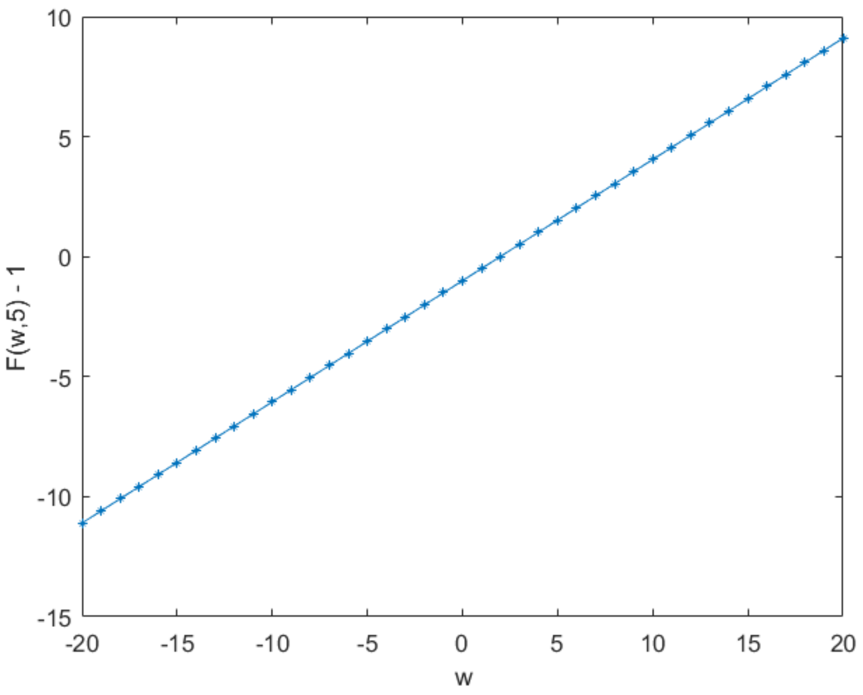
Because we will be solving this equation multiple times, we will start out using a relatively large step size. Later on, we can refine our solution. Consider splitting up the interval $[1, 5]$ into N equally spaced points x_i for $i = 1, 2, \ldots, N$. Letting $\mathbf{A}^n = \mathbf{A}|_{x=x_n}$ and $\mathbf{v}^n = \mathbf{v}|_{x=x_n}$

$$\mathbf{h}_1 = \Delta x \mathbf{A}^n \mathbf{v}^n$$
$$\mathbf{h}_2 = \Delta x \mathbf{A}^{n+.5}(\mathbf{v}^n + .5\mathbf{h}_1)$$
$$\mathbf{h}_3 = \Delta x \mathbf{A}^{n+.5}(\mathbf{v}^n + .5\mathbf{h}_2)$$
$$\mathbf{h}_4 = \Delta x \mathbf{A}^{n+1}(\mathbf{v}^n + \mathbf{h}_3)$$
$$\mathbf{v}^{n+1} = \mathbf{v}^n + \frac{1}{6}(\mathbf{h}_1 + 2\mathbf{h}_2 + 2\mathbf{h}_3 + \mathbf{h}_4)$$

and the initial conditions become

$$\mathbf{v}^1 = \begin{bmatrix} 0 \\ w \end{bmatrix}$$

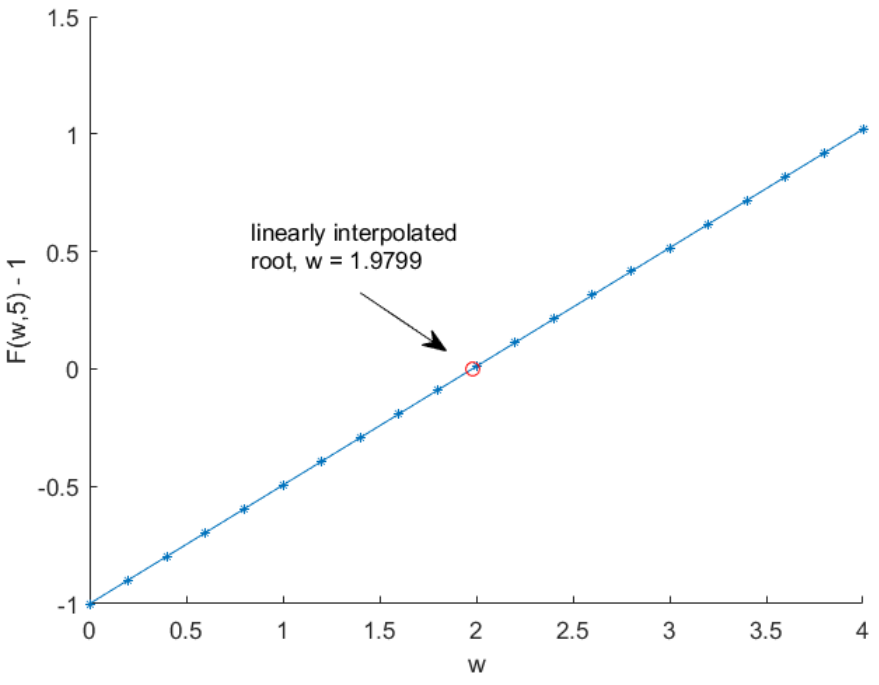
Let $F(w, 5)$ denote the solution so this IVP for $y'(0) = w$ at $x = 5$. Our first task is to find $F(w, 5) - 1$ for enough w to find a root. To start with, we will consider $w = -20, -19, -18, \ldots, 19, 20$.



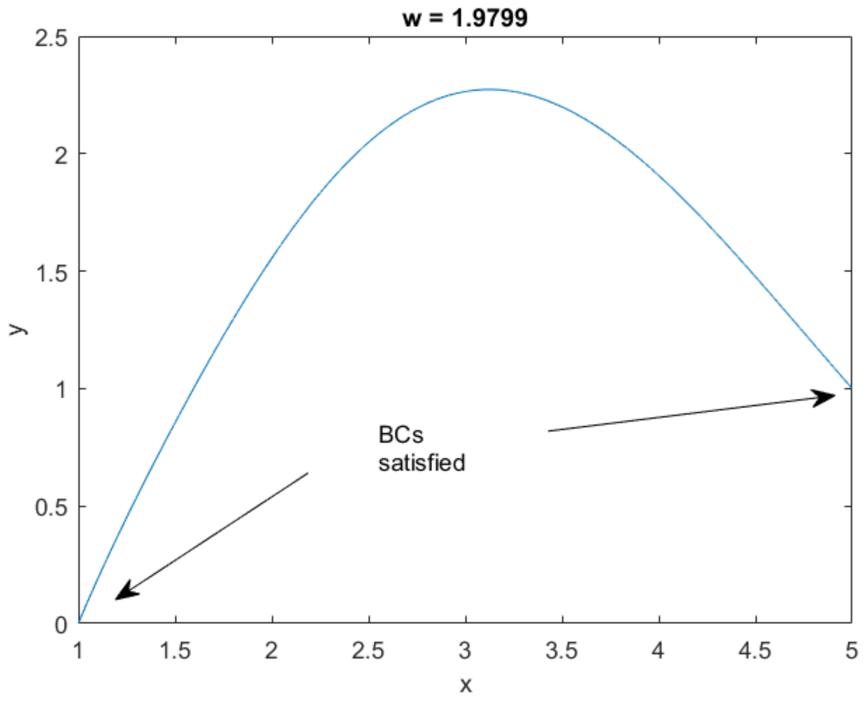
```matlab
dx = 1e-3;
x0 = 1; xN = 5;
F = [];
wrange = -20:20;
for w = wrange
    IC = [0; w];
    [x,y] = myRK4(x0,IC,xN,dx);
    F = [F y(end)];
end
plot(wrange,F-1,'*-','MarkerSize',3)
xlabel('w')
ylabel('F(w,5) - 1')



function [x,y] = myRK4(x0,IC,xN,dx)
x = x0:dx:xN;
numx = length(x);
v = zeros(2,numx);
v(1,1) = IC(1);
v(2,1) = IC(2);

for i = 1:numx-1
    vn = v(:,i);
    h1 = dx*A(x(i))*vn;
    h2 = dx*A(x(i)+.5*dx)*(vn+.5*h1);
    h3 = dx*A(x(i)+.5*dx)*(vn+.5*h2);
    h4 = dx*A(x(i+1))*(vn+h3);
    v(:,i+1) = vn + (1/6)*(h1+2*h2+2*h3+h4);
end
y = v(1,:);
end

function out =  A(x)
out = [0 1; (5/x^2-1) -1];
end
```

From this, we see that the correct initial solution is around $w = 3$. Now we can refine both w and dx by using smaller spacing as well as doing a linear interpolation of w.

```matlab
    g = @(xq) interp1(wrange,F-1,xq);
    w_sol = fzero(@(x) g(x),2);
```



and so using a value of $w = 1.9799$ and a step size of $\Delta x = 1 \times 10^{-4}$



In the previous example, we saw that $F(w, 5) - 1$ was linear in w. This is a result of the linearity of the differential equation. For BV problems, if the differential equation is linear, there is a unique solution and using linear interpolation can be used to leverage serious savings in computational time. Instead of solving the equation for many values of w, only two different values need to be evaluated as long as there is a sign change between the two values. From here, the root can be linearly interpolated.

On the other hand, nonlinear BVPs are significantly more involved. For these problems, $F(w, b) - \beta$ is *not* linear. Moreover, the solution is not even unique. There can be multiple roots. Interpolation can still be used, but not over large ranges like in the linear case. The following example will demonstrate this.

Ex 2.6. Find a solution to the differential equation $y'' + y^2y' + xy = 0$ with $y(0) = 2$ and $y(10) = \frac{1}{2}$.

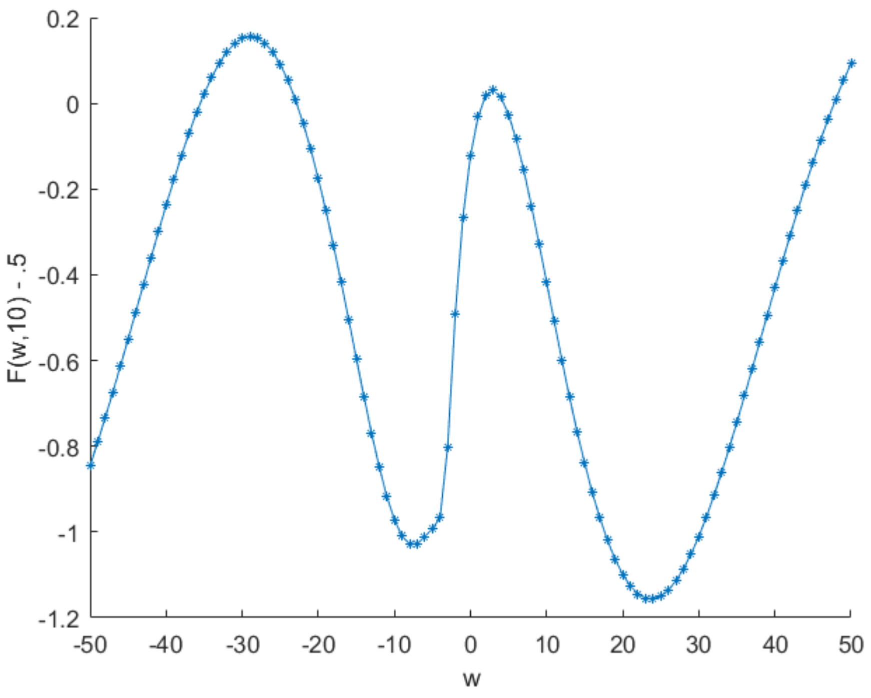
Like the previous example, we will use the transformation $v_0 = y$ and $v_1 = y'$. Unlike the previous example, the matrix equation will now have an additional term representing the nonlinearity.

$$\underbrace{\begin{bmatrix} v_0 \\ v_1 \end{bmatrix}'}_{\mathbf{v}'} = \underbrace{\begin{bmatrix} 0 & 1 \\ -x & 0 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} v_0 \\ v_1 \end{bmatrix}}_{\mathbf{v}} + \underbrace{\begin{bmatrix} 0 \\ -v_0^2 v_1 \end{bmatrix}}_{\mathbf{b}}$$

Again, consider splitting up the interval $[0, 10]$ into N equally spaced points x_i for $i = 1, 2, \dots, N$ and let $\mathbf{A}^n = \mathbf{A}|_{x=x_n}$, $\mathbf{v}^n = \mathbf{v}|_{x=x_n}$, and $\mathbf{b}_=\mathbf{b}|_{x=x_n}$. Using an RK4 scheme,

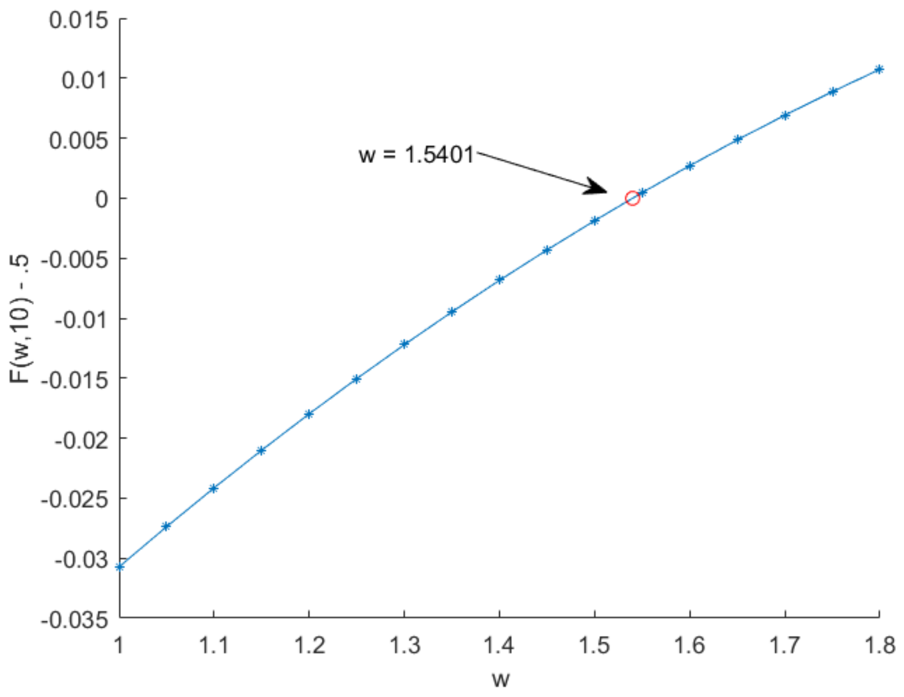
$$\begin{aligned}
\mathbf{h}_1 &= \Delta x \mathbf{A}^n \mathbf{v}^n + \Delta x \mathbf{b}|_{\mathbf{v}^n} \\
\mathbf{h}_2 &= \Delta x \mathbf{A}^{n+.5}(\mathbf{v}^n + .5\mathbf{h}_1) + \Delta x \mathbf{b}|_{\mathbf{v}^n + .5\mathbf{h_1}} \\
\mathbf{h}_3 &= \Delta x \mathbf{A}^{n+.5}(\mathbf{v}^n + .5\mathbf{h}_2) + \Delta x \mathbf{b}|_{\mathbf{v}^n + .5\mathbf{h_2}} \\
\mathbf{h}_4 &= \Delta x \mathbf{A}^{n+1}(\mathbf{v}^n + \mathbf{h}_3) + \Delta x \mathbf{b}|_{\mathbf{v}^n + \mathbf{h_3}} \\
\mathbf{v}^{n+1} &= \mathbf{v}^n + \frac{1}{6}(\mathbf{h}_1 + 2\mathbf{h}_2 + 2\mathbf{h}_3 + \mathbf{h}_4)
\end{aligned}$$

The initial conditions for the corresponding IVP become $\mathbf{v}^1 = \begin{bmatrix} 2 & w \end{bmatrix}^T$. Let $F(w, 10)$ denote $y(10)$ obtained by solving the above IVP with initial condition w. We seek to find a root of $F(w, 10) - \frac{1}{2}$. To start with, we will consider $w = -50, -49, \dots, 49, 50$ and a larger step size of $\Delta x = .01$.

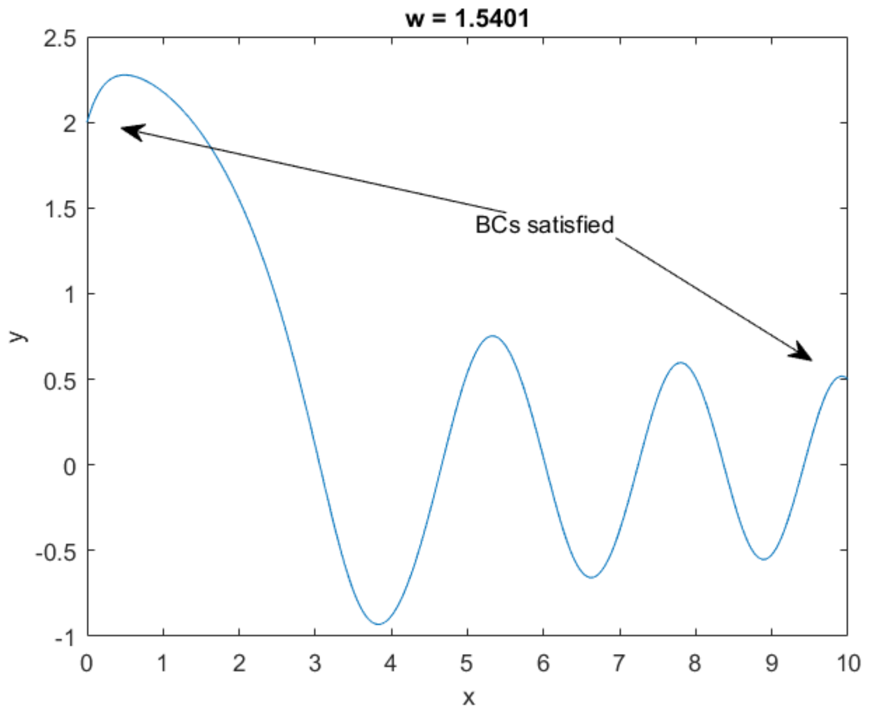


```
dx = 1e-2;
x0 = 0; xN = 10;
F = [];
wrange = -50:1:50;
beta = .5; alpha = 2;
for w = wrange
    IC = [alpha; w];
    [x,y] = myRK4(x0,IC,xN,dx);
    F = [F y(end)];
end
g = @(xq) interp1(wrange,F-beta,xq);
hold on
plot(wrange,F-beta,'*-','MarkerSize',3)
xlabel('w')
ylabel('F(w,10) - .5')


function [x,y] = myRK4(x0,IC,xN,dx)
x = x0:dx:xN;
numx = length(x);
v = zeros(2,numx);
v(1,1) = IC(1);
v(2,1) = IC(2);

for i = 1:numx-1
    vn = v(:,i);
    h1 = dx*dV(x(i),vn);
    h2 = dx*dV(x(i)+.5*dx,vn+.5*h1);
    h3 = dx*dV(x(i)+.5*dx,vn+.5*h2);
    h4 = dx*dV(x(i+1),vn+h3);
    v(:,i+1) = vn + (1/6)*(h1+2*h2+2*h3+h4);
end
y = v(1,:);
end

function out =  dV(x,v)
out = zeros(2,1);
out(1) = v(2);
out(2) = -x*v(1) -v(1)^2*v(2);
end
```

The nonlinearity of the equation gives rise to multiple solutions. We only seek a single solution, so let's focus on the one near $w = 0$. Refining the interval to search for w and performing linear interpolation yields $w = 1.5401$.



Solving the DE with this initial condition yields



2.3.2 Eigenvalue Problems