Project: 1.2 Ordinary Differential Equations

January 23, 2018

1. **Using the Euler method, starting with $Y_0 = 0$, compute Y_n for x up to $x = 6$ with $h = 0.6$, i.e. for n up to $6/h = 10$. Tabulate the values of x_n, the numerical solution Y_n, the analytic solution $y(x_n)$ from (6), and the global error $E_n = Y_n - y(x_n)$. You should find that the numerical result is unstable: the error oscillates with a magnitude that ultimately grows proportional to $e^{\gamma x}$, where the 'growth rate' γ is a positive constant which you should estimate.**

   Repeat with $h = 0.4$, 0.2, 0.125 and 0.1, presenting only a judicious selection of output to illustrate the behaviour. What effect does reducing h have on the instability and its growth rate?

   The function 'forwardeuler2' is specific for section 2 and uses $f(x, y) = -16y + 15e^{-x}$, $y(x) = e^{-x} - e^{-16x}$, as well as the intial condition $y(0) = 0$.

Key		
$x_n = x_0 + hn$	$Y_n = Y_{n-1} + hf(x_{n-1}, Y_{n-1})$	$y(x_n)$ = Actual Value of the function at x_n
$E_n = Y_n - y(x_n)$, i.e. Error in the value of Y_n		forwardeuler2(Number of Iterations = N, Step length = h)

Table 1: Key for what each input argument represents in the function as well as understanding what the headers mean in the table below.

Input into MATLAB: forwardeuler2(10,0.6). The table below shows the output of this function.

Iteration/n	x_n	Y_n	$y(x_n)$	E_n
1	0.6	9	0.548744	8.45126
2	1.2	-72.4607	0.301194	-72.7619
3	1.8	625.873	0.165299	625.707
4	2.4	-5381.02	0.090718	-5381.11
5	3	46277.6	0.0497871	46277.5
6	3.6	-397987	0.0273237	-397987
7	4.2	3.42269×10^6	0.0149956	3.42269×10^6
8	4.8	-2.94351×10^7	0.00822975	-2.94351×10^7
9	5.4	2.53142×10^8	0.00451658	2.53142×10^8
10	6	-2.17702×10^9	0.00247875	-2.17702×10^9

Table 2: Output for $h = 0.6$, showing the divergence of the the Forward Euler numerical method.

From Table 2, we can clearly see that this error oscillates and also grows in magnitude. From the question, we know the error grows with a magnitude proportional to $e^{\gamma x}$, as n gets large, and so we can estimate γ from the data we have obtained. We should consider larger values of n to get a more accurate estimate of γ, since the approximation is better for large n.

In general, given the information above, we can write:

$$|E_n| = ke^{\gamma x_n}$$

where k is the constant of proportionality.

We can then rearrange this equation and make γ the subject:

$$\begin{aligned} |E_n| &= ke^{\gamma x_n} \\ \Rightarrow \ln\left(\frac{|E_n|}{k}\right) &= \gamma x_n \\ \Rightarrow \qquad \gamma &= \frac{\ln\left(\frac{|E_n|}{k}\right)}{x_n} \end{aligned}$$

Below is an estimate of γ using $|E_9| = 2.53142 \times 10^8$, $x_9 = 5.4$, $|E_{10}| = 2.17702 \times 10^9$ and $x_{10} = 6$. We obtain two equations and solve them simultaneously.

$$\gamma = \frac{\ln\left(\frac{|E_9|}{k}\right)}{x_9} \Rightarrow k = \frac{|E_9|}{e^{\gamma x_9}} \tag{1}$$

$$\gamma = \frac{\ln\left(\frac{|E_{10}|}{k}\right)}{x_{10}} \Rightarrow k = \frac{|E_{10}|}{e^{\gamma x_{10}}} \tag{2}$$

Now equate both equations: (1) = (2)

$$\frac{|E_9|}{e^{\gamma x_9}} = \frac{|E_{10}|}{e^{\gamma x_{10}}}$$

$$\ln\left(\left(\frac{e^{x_{10}}}{e^{x_9}}\right)^{\gamma}\right) = \ln\left(\frac{|E_{10}|}{|E_9|}\right)$$

$$\gamma = \frac{\ln\left(\frac{|E_{10}|}{|E_9|}\right)}{x_{10} - x_9}$$

$$\gamma = \frac{\ln\left(\frac{2.17702\times10^9}{2.53142\times10^8}\right)}{6 - 5.4}$$

$$\gamma = 3.58627$$

I will now consider the Forward Euler method for values of h equal to 0.4, 0.2, 0.125 and 0.1. We can clearly see a pattern as h is varied. I will also calculate γ for each value of h, omitting the working out as it is the same as above.

Input into MATLAB: forwardeuler2(10,0.4). For $h = 0.4$, $\gamma = 4.21599$.

Iteration/n	x_n	Y_n	$y(x_n)$	E_n
⋮	⋮	⋮	⋮	⋮
9	3.6	3.85908×10^6	0.0273237	3.85908×10^6
10	4	-2.0839×10^7	0.0183156	-2.0839×10^7

Table 3: Output for $h = 0.4$, showing similar significant divergence of the the Forward Euler method, as was observed for when $h = 0.6$.

Input into MATLAB: forwardeuler2(10,0.2). For $h = 0.2$, $\gamma = 3.94228$.

Iteration/n	x_n	Y_n	$y(x_n)$	E_n
⋮	⋮	⋮	⋮	⋮
9	1.8	1199.94	0.165299	1199.78
10	2	-2639.38	0.135335	-2639.51

Table 4: Output for $h = 0.2$, showing the slower divergence of the the Forward Euler numerical method, compared to when $h = 0.4$

Input into MATLAB: forwardeuler2(10,0.125). For $h = 0.125$, $\gamma = 0.01955$.

Iteration/n	x_n	Y_n	$y(x_n)$	E_n
⋮	⋮	⋮	⋮	⋮
9	1.125	1.31938	0.324652	0.994725
10	1.25	-0.710654	0.286505	-0.997159

Table 5: Output for $h = 0.125$, showing a much slower divergence of the the Forward Euler numerical method, compared to $h = 0.2, 0.4, 0.6$.

Input into MATLAB: forwardeuler2(10,0.1). For $h = 0.1$, $\gamma = -1.92365$.

Iteration/n	x_n	Y_n	$y(x_n)$	E_n
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
9	0.9	0.415308	0.406569	0.00873891
10	1	0.36067	0.367879	−0.00720965

Table 6: Output for $h = 0.1$, showing the convergence of the the Forward Euler numerical method.

These results indicate how the reduction in h decreases the instability of the Forward Euler Method. We can clearly see how the magnitude of the errors are reduced substantially for every smaller value of h tested. The growth rate γ increases initially for smaller values of h. However, in the limit as $n \Rightarrow \infty$, the growth rate for a particular value of h will be less than the growth rate for a larger h. Due to the reduction in the magnitude of the error, the growth rate γ is reduced and becomes negative when the Forward Euler method converges.

2. **(i) Find the analytic solution of the Euler *difference* equation**

$$Y_{n+1} = Y_n + h(-16Y_n + 15(e^{-h})^n) \textbf{ with } Y_0 = 0.$$

(ii) Hence explain why and when instability occurs, and with what growth rate.

(iii) Show that in the limit $h \to 0$, $n \to \infty$ with $x_n \equiv nh$ fixed, the solution of the difference-equation problem (7) converges to the solution (6) of the differential-equation problem specifed by (1a), (5a) and (5b).

We can work through solving this difference equation to find the values of h for which Y_n is valid. This will give us an indication of what values of h are valid for the Forward Euler method to be stable.

(i) Our solution Y_n is made up of a complimentary function and a particular integral, $Y_n = Y_n^{(c)} + Y_n^{(p)}$. We can find the complimentary function by solving the homogenous difference equation. We can choose the ansatz to be $Y_n^{(c)} = Ak^n$. First we rearrange our initial equation, and then work out each term above separately.

$$\begin{aligned}
Y_{n+1} &= Y_n + h(-16Y_n + 15(e^{-h})^n)\\
Y_{n+1} + 16hY_n - Y_n &= 15h(e^{-hn})\\
\text{Homogeonous (unforced) difference equation:}&\\
Y_{n+1}^{(c)} + 16hY_n^{(c)} - Y_n^{(c)} &= 0\\
\text{Using the ansat, } Y_n^{(c)} = Ak^n\text{, we obtain:}&\\
Ak^{n+1} + 16hAk^n - Ak^n &= 0\\
k\cancel{Ak^n} + (16h-1)\cancel{Ak^n} &= 0\\
k &= (1-16h)
\end{aligned}$$

Hence:

$$\Rightarrow Y_n^{(c)} = A(1-16h)^n$$

We can work out the particular integral of this difference equation that gives us the forcing, using the ansatz $Y_n^{(p)} = Be^{-hn}$.

We again consider the original equation and solve for the particular integral:

$$\begin{aligned}
Y_{n+1} + 16hY_n - Y_n &= 15h(e^{-hn})\\
\text{Inhomogeonous (forced) difference equation:}&\\
Y_{n+1}^{(p)} + 16hY_n^{(p)} - Y_n^{(p)} &= 15h(e^{-hn})\\
Be^{-h(n+1)} + 16hBe^{-hn} - Be^{-hn} &= 15h(e^{-hn})\\
Be^{-h}\cancel{(e^{-hn})} + 16hB\cancel{(e^{-hn})} - B\cancel{(e^{-hn})} &= 15h\cancel{(e^{-hn})}\\
B(e^{-h} + 16h - 1) &= 15h\\
B &= \frac{15h}{e^{-h} + 16h - 1}
\end{aligned}$$

Therefore we have found a solution, $Y_n^{(p)}$, for this difference equation:

$$Y_n^{(p)} = \frac{15he^{-hn}}{e^{-h} + 16h - 1}$$

The general solution is:

$$Y_n = Y_n^{(c)} + Y_n^{(p)} = A(1-16h)^n + \frac{15he^{-hn}}{e^{-h} + 16h - 1}$$

Applying the initial condition that $Y_0 = 0$, we obtain:

$$Y_0 = 0 = A + \frac{15h}{e^{-h} + 16h - 1} \Rightarrow A = -\frac{15h}{e^{-h} + 16h - 1}$$

Hence we can write the final solution as:

$$Y_n = -\frac{15h(1-16h)^n}{e^{-h} + 16h - 1} + \frac{15he^{-hn}}{e^{-h} + 16h - 1}$$

(ii) We can now consider the leading term, $(1-16h)^n$. Clearly convergence occurs for values of h such that $|1-16h| < 1$. i.e. for $0 < h < 0.125$. Hence instability occurs for all $h > 0.125$, as this term does not converge for these values. The growth rate can be calculated by equating the modulus of the asymptotic behaviour (since we consider the magnitude of the error) to the predicted asymptotic behaviour in the question and solving for γ.

$$\left| -\frac{15h(1-16h)^n}{e^{-h} + 16h - 1} \right| = ke^{\gamma x_n}$$

$$\left| -\frac{15h}{e^{-h} + 16h - 1} \right| \left| (1-16h)^n \right| = ke^{\gamma hn}$$

$$\text{We can let: } k = \left| -\frac{15h}{e^{-h} + 16h - 1} \right|$$

$$\text{And hence obtain: } \left|(1-16h)^n\right| = |1-16h|^n = e^{\gamma hn}$$

$$\Rightarrow |1-16h| = e^{\gamma h}$$

$$\Rightarrow \gamma = \frac{\ln|1-16h|}{h}$$

(iii) Considering the case as $h \to 0$ and $n \to \infty$, with $x_n = hn$ is fixed, we can see that:

$$\lim_{h\to 0}\lim_{n\to\infty} Y_n = \lim_{h\to 0}\lim_{n\to\infty} \left(\frac{15h}{e^{-h} + 16h - 1}\left(e^{-hn} - (1-16h)^n\right) \right) = (e^{-x_n} - e^{-16x_n}),$$

since

$$\lim_{h\to 0} \left(\frac{15h}{e^{-h} + 16h - 1} \right) = \lim_{h\to 0} \left(\frac{15}{-e^{-h} + 16} \right) = 1, \text{ by L'Hôpitals Rule,}$$

and

$$\lim_{h\to 0}\lim_{n\to\infty} \left(e^{-hn} - (1-16h)^n\right) = \lim_{h\to 0}\lim_{n\to\infty} \left(e^{-hn} - \left(1 + \frac{-16x_n}{n}\right)^n\right) = (e^{-x_n} - e^{-16x_n}).$$

This is obtained by using the known limit:

$$\lim_{n\to\infty} \left(1 + \frac{k}{n}\right)^n = e^k.$$

3. **Integrate the ODE numerically with $h = 0.05$ from $x = 0$ to $x = 4$ using both the Euler and RK4 methods, in each case tabulating the numerical solution Y_n against x_n (although not necessarily at every step), and plotting it with the exact solution (6) superposed.**

Input into MATLAB: $[Xf, Yf, Ef]$=forwardeuler2_3(80,0.05)

n	0	1	2	3	4	5	...	77	78	79	80
x_n	0	0.05	0.10	0.15	0.20	0.25	...	3.85	3.90	3.95	4.00
Y_n	0	0.75000	0.863422	0.851312	0.815793	0.777207	...	0.021245	0.020209	0.019223	0.018286

Table 7: Table showing how x_n and Y_n vary as n increases for the forward Euler Method.

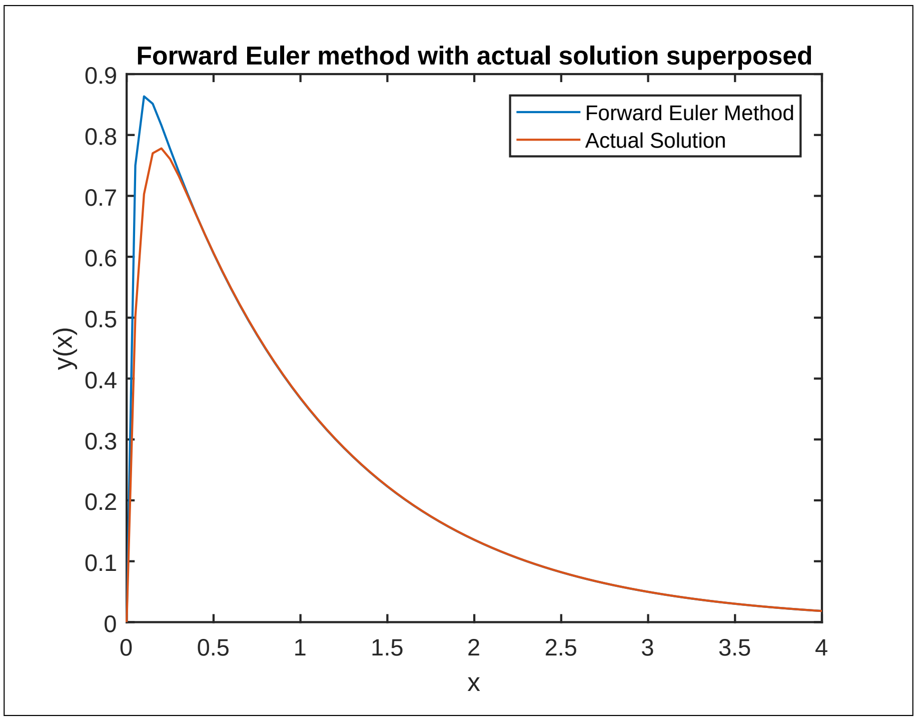

Figure 1: Graph showing the forward Euler method compared to the actual solution of the ODE.

Input into MATLAB: $[Xr, Yr, Er]$=rungekutta2_3(80,0.05)

n	0	1	2	3	4	5	...	77	78	79	80
x_n	0	0.05	0.10	0.15	0.20	0.25	...	3.85	3.90	3.95	4.00
Y_n	0	0.499509	0.700793	0.768546	0.777109	0.760010	...	0.021280	0.020242	0.019255	0.018316

Table 8: Table showing how x_n and Y_n vary as n increases for the Runge-Kutta method.

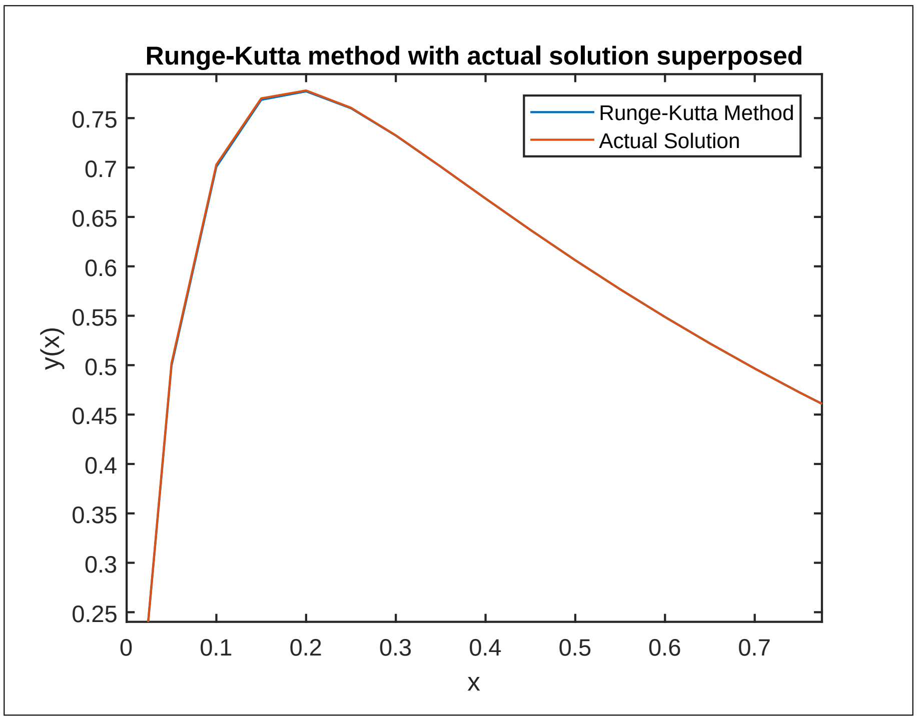

Figure 2: Graph showing the Runge-Kutta method compared to the actual solution of the ODE. We can clearly see how close the iterative solution is to the actual solution.

4. **For both the Euler and the RK4 methods, tabulate the global error E_n at $x_n = 0.1$ against $h \equiv 0.1/n$ for $n = 2^k$ with $k = 0, 1, 2, \ldots, 15$, and plot a log-log graph of $|E_n|$ against h over this range.**

   Comment on the relationship of your results to the theoretical accuracy of the methods.

   Using both functions: 'forwardeuler2_4' and 'rungekutta2_4', we can obtain the log-log graph below indicating the relationship between the modulus of the error, $|E_n|$, and the step size, h, for n as described in the question above.

   Since this graph (Figure 3) gives a straight line for both methods, we can conclude that the error, $|E_n|$, obtained in the forward Euler method decreases proportional to h^α, for some α. Similarly, the error $|E_n|$ for the Runge-Kutta method decreases proportional to h^β, for some β. Since the Runge-Kutta method has a steeper slope in the log-log graph, we expect $\alpha < \beta$. To work out α and β, we can use the formula for the gradient of a log-log graph in relation to the equation of the graph.

$$|E_n| = Ah^m \Rightarrow \ln|E_n| = m\ln(h) + \ln(A) \Rightarrow m = \frac{\ln(|E_i/E_j|)}{\ln(h_i/h_j)},$$

   where $(h_i, |E_i|)$ and $(h_j, |E_j|)$ are points on the log-log graph. Solving for α and β using the data obtained to create the graph, we get that $\alpha \approx 1.0147$ and $\beta \approx 4.0453$. These results agree with the theoretical accuracy of the forward Euler method and the Runge-Kutta method. Since the forward Euler method is $O(h^2)$ as $h \to 0$, we would expect $\alpha = 1$ which is very close to our calculated value. Since the Runge-Kutta method is $O(h^5)$ as $h \to 0$, we would expect $\beta = 4$, which is again very close to our calculated value. Hence there is a strong similarity between the theoretical accuracy and the results obtained.

   There is clearly an issue with the Runge-Kutta plot for values of $h < 10^{-4}$. This is due to the floating point precision of MATLAB being 2.2204×10^{-16}. To get the errors at $x_n = 0.1$, $|E_n|$, for $h < 10^{-4}$, MATLAB would have had to use values beyond this precision and hence the data obtained would be inaccurate, leading to the graph not being linear for all values of h used.

5. **Use the Euler method with $h = 0.001$ to integrate from $x = 0$ to $x = 10$. You should find that the magnitude of the global error ultimately grows *exponentially*. Solve the Euler difference-equation problem analytically; hence explain the reason for this behaviour, and identify the growth rate. Could the error be suppressed by using a smaller value of h? Why, or why not? What if RK4 were used instead of Euler?**

   Input into MATLAB: forwardeuler2_5(10000,0.001)

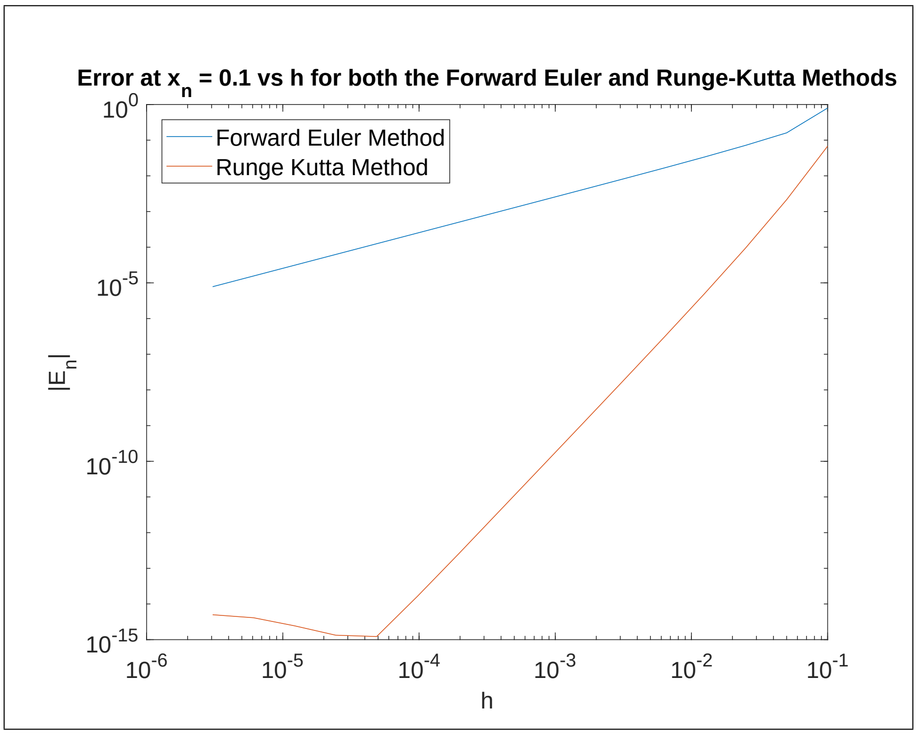

Figure 3: Graph showing how the error, $|E_n|$, decreases proportional to h^α (forward Euler method) and h^β (Runge-Kutta method), for $h > 10^{-4}$.

Iteration/n	x_n	Y_n	$y(x_n)$	$\|E_n\|$
1	0.001	0.999	0.999	4.99833×10^{-7}
2	0.002	0.998001	0.998002	1.00117×10^{-6}
3	0.003	0.997003	0.997004	1.50401×10^{-6}
4	0.004	0.996006	0.996008	2.00836×10^{-6}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
9997	9.997	-2.14697×10^{13}	4.55363×10^{-5}	2.14697×10^{13}
9998	9.998	-2.15556×10^{13}	4.54908×10^{-5}	2.15556×10^{13}
9999	9.999	-2.16418×10^{13}	4.54454×10^{-5}	2.16418×10^{13}
10000	10	-2.17284×10^{13}	4.53999×10^{-5}	2.17284×10^{13}

Table 9: Table showing how rapidly the magnitude of the error increases from being of the order 10^{-7} to 10^{13}

We can see that the global error ultimately grows exponentially, say $E_n \sim e^{\gamma_2 x_n}$, and once we have solved the difference equation, we can work out an equation for the growth rate (γ_2).

We can solve the difference equation (3) analytically using the same method as before. The solution is of the form $Y_n = Y_n^{(c)} + Y_n^{(p)}$, and so we proceed in the same way.

We can work out the complimentary function using the same ansatz as before: $Y_n^{(c)} = Ak^n$.

$$Y_{n+1} = Y_n + h(4Y_n - 5(e^{-x_n})) \tag{3}$$

$$Y_{n+1} - 4hY_n - Y_n = -5h(e^{-hn})$$

Homogeonous (unforced) difference equation:

$$Y_{n+1}^{(c)} - 4hY_n^{(c)} - Y_n^{(c)} = 0$$

$$Ak^{n+1} - 4hAk^n - Ak^n = 0$$

$$k\cancel{Ak^n} - (4h+1)\cancel{Ak^n} = 0$$

$$k = (4h+1) \Rightarrow Y_n^{(c)} = A(4h+1)^n$$

We do the same thing for the particular integral, i.e. let $Y_n^{(p)} = Be^{-hn}$.

$$Y_{n+1} - 4hY_n - Y_n = -5h(e^{-hn})$$

Inhomogeonous (forced) difference equation:

$$\begin{aligned}
Y_{n+1}^{(p)} - 4hY_n^{(p)} - Y_n^{(p)} &= -5h(e^{-hn}) \\
Be^{-h(n+1)} - 4hBe^{-hn} - Be^{-hn} &= -5h(e^{-hn}) \\
Be^{-h}\cancel{(e^{-hn})} - 4hB\cancel{(e^{-hn})} - B\cancel{(e^{-hn})} &= -5h\cancel{(e^{-hn})} \\
B(e^{-h} - 4h - 1) &= -5h \\
B = \frac{-5h}{e^{-h} - 4h - 1} &\Rightarrow Y_n^{(p)} = \frac{-5he^{-hn}}{e^{-h} - 4h - 1}
\end{aligned}$$

The general solution is:

$$Y_n = Y_n^{(c)} + Y_n^{(p)} = A(4h+1)^n - \frac{5he^{-hn}}{e^{-h} - 4h - 1}$$

Applying the initial condition that $Y_0 = 0$, we obtain the final solution:

$$Y_n = \frac{5h(4h+1)^n}{e^{-h} - 4h - 1} - \frac{5he^{-hn}}{e^{-h} - 4h - 1}$$

In the limit as $n \to \infty$, the magnitude of the error $|E_n|$ will always increase exponentially since for no values of $h > 0$ will the term $\frac{5h(4h+1)^n}{e^{-h}-4h-1}$ converge, since the leading term $(4h+1) > 1 \; \forall h > 0$. The growth rate is calculated in the same way as before. As $n \to \infty$, $-\frac{5he^{-hn}}{e^{-h}-4h-1} \to 0$. Therefore the error is only a result of the first term. We can, as before, equate the magnitude of this error term to the predicted asymptotic behaviour and solve for γ_2.

$$\begin{aligned}
\left|\frac{5h(4h+1)^n}{e^{-h} - 4h - 1}\right| &= ke^{\gamma_2 x_n} \\
\left|\frac{5h}{e^{-h} - 4h - 1}\right|\left|(4h+1)^n\right| &= ke^{\gamma_2 hn} \\
\text{We can let: } k &= \left|\frac{5h}{e^{-h} - 4h - 1}\right| \\
\text{And hence obtain: } \left|(4h+1)^n\right| = |4h+1|^n &= e^{\gamma_2 hn} \\
\Rightarrow |4h+1| &= e^{\gamma_2 h} \\
\Rightarrow \gamma_2 &= \frac{\ln|4h+1|}{h} \\
\text{For } h &= 0.001: \\
\gamma_2 = \frac{\ln|4 \times 0.001 + 1|}{0.001} &= 3.99202
\end{aligned}$$

Since

$$\gamma_2 = \frac{\ln|4h+1|}{h} \text{ for all } h,$$

we can see that the magnitude of the error in this case will always get exponentially bigger, as the growth rate $\gamma_2 > 0$.

If a smaller value of h is used, then the error would initially be very small (and hence suppressed) since the leading term $(4h+1)^n$ would not grow as fast. However as n increases, the magnitude of the error, as shown above, always grows exponentially and so eventually the error will get larger in size.

Using the Runge-Kutta method, compared to the Euler method, would result in a much more accurate numerical approximation for all values of h. This is evident in Table 10 below where we can see that the initial error is of the order 10^{-16} whereas the Euler method has initial errors of 10^{-7}. More so, the approximations would still be accurate even as n increases due to the slow divergence of the magnitude of the error. We can also see from Table 10 that the error for the Runge-Kutta method does not increase as fast as n gets large, which means that the error is suppressed by using the Runge-Kutta method. However, similar to the Euler method, the error still eventually increases exponentially and in the limit will get very large.

Input into MATLAB: rungekutta2_5(10000,0.001)

Iteration/n	x_n	Y_n	$y(x_n)$	$\lvert E_n \rvert$
1	0.001	0.999	0.999	7.46917×10^{-16}
2	0.002	0.998002	0.998002	1.43108×10^{-16}
3	0.003	0.997004	0.997004	2.19593×10^{-16}
4	0.004	0.996008	0.996008	2.98552×10^{-16}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
9997	9.997	−33893.2	4.55363×10^{-5}	33893.2
9998	9.998	−34029.1	4.54908×10^{-5}	34029.1
9999	9.999	−34165.5	4.54454×10^{-5}	34165.5
10000	10	−34302.4	4.53999×10^{-5}	34302.4

Table 10: Table showing how small the magnitude of the error is for the Runge-Kutta method compared to the Euler method, when $h = 0.001$.

6. **When $\alpha = 4$, equation (12a) with $p \neq 0$ has general solution**

$$y = A(1+x)\sin\left(p(1+x)^{-1} - \phi\right)$$

where A and ϕ are arbitrary constants. What if $p = 0$? Write down the particular solution (for all p) satisfying

$$y = 0,\ dy/dx = 1 \textbf{ at } x = 0.$$

Deduce that the smallest (non-negative) eigenvalue of (12a)−(12b) with $\alpha = 4$ is $p = 2\pi$, and write down the other eigenvalues and the corresponding eigenfunctions, $y(x)$ (there is no need to normalise the eigenfunctions).

I will first address the question of when $p = 0$. In this case, equation (12a) becomes:

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = 0,$$

which is satisfied by the general solution above:

$$y(x) = A(1+x)\sin(-\phi) = B(1+x), \text{ where } B = A\sin(-\phi), \text{ a constant.}$$

If we apply the initial conditions to the case $p = 0$, we get:

$$\left.\begin{array}{l} 0 = B \\ 1 = B \end{array}\right\} \implies \text{There is no solution for } p = 0.$$

For all $p > 0$, we can find the particular solution given the initial conditions in the question.

$$y(x) = A(1+x)\sin\left(p(1+x)^{-1} - \phi\right)$$

Using $y = 0$ at $x = 0$:

$$0 = A\sin(p - \phi) \tag{4}$$

Using $\dfrac{\mathrm{d}y}{\mathrm{d}x} = 1$ at $x = 0$:

$$1 = A\sin(p - \phi) - Ap\cos(p - \phi) \tag{5}$$

We can solve equations (4) and (5) simultaneously to work out the values of A and ϕ. Solving equation (4) gives us:

$$0 = A\sin(p-\phi) \Rightarrow \sin(p-\phi) = 0, \text{ since } A \neq 0.$$
$$\Rightarrow p - \phi = n\pi \Rightarrow \phi = p - n\pi, \text{ where } n \in \mathbb{Z}.$$

We can then substitute this into equation (5) and obtain a value for A.

$$1 = A\sin(p-\phi) - Ap\cos(p-\phi)$$
$$1 = 0 - Ap(-1)^n = Ap(-1)^{n+1}$$
$$\Rightarrow A = \frac{1}{p(-1)^{n+1}}, \text{ where } n \in \mathbb{Z}.$$

Therefore the particular solution is:

$$y(x) = \frac{(1+x)}{p(-1)^{n+1}} \sin\big(p\big((1+x)^{-1} - 1\big) + n\pi\big)$$

After some simplification, we obtain:

$$y(x) = \frac{1+x}{p} \sin\left(\frac{px}{1+x}\right), \text{ for } p > 0.$$

To find the smallest non-negative eigenvalue of (12a)-(12b), we must solve (12a) analytically using (13) and obtain possible values for p.

$$y(x) = A(1+x)\sin\big(p(1+x)^{-1} - \phi\big)$$

Apply boundary conditions, $y(0) = y(1) = 0$, we obtain:

$$y(0) = 0 = A\sin(p-\phi) \qquad\qquad y(1) = 0 = 2A\sin\left(\frac{p}{2} - \phi\right)$$

$$\sin(p-\phi) = 0 \qquad\qquad \sin\left(\frac{p}{2} - \phi\right) = 0$$

$$p - \phi = n\pi$$

We can substitute ϕ into $\sin\left(\frac{p}{2} - \phi\right)$, since ϕ is a constant, and obtain:

$$\sin\left(\frac{p}{2} - p + n\pi\right) = \sin\left(n\pi - \frac{p}{2}\right) = (-1)^{n+1}\sin\left(\frac{p}{2}\right) = 0 \implies p = 2\pi k, \text{ where } k \in \mathbb{Z}.$$

Therefore the smallest non-negative eigenvalue p would be when $k = 1$, i.e. $p = 2\pi$, since $k = 0 \Rightarrow p = 0$ (for which there are no solutions as shown above).

The other eigenvalues would be

$$p = 2k\pi, \text{ where } k \in \mathbb{Z},$$

with corresponding eigenfunctions

$$y(x) = A(-1)^{n+1}(1+x)\sin\left(\frac{2k\pi x}{1+x}\right) = B(1+x)\sin\left(\frac{2k\pi x}{1+x}\right).$$

7. **Taking $\alpha = 4$, run your program with $p = 6$ and $h = 0.1/2^k$ for $k = 0, 1, 2, \ldots, 12$ in turn, tabulating the numerical solution Y_n at $x_n = 1$ and the global error $Y_n - y(1)$ against $h \equiv 1/n$. Repeat with $p = 7$. Do the errors behave as expected?**

   The analytic solution for the scenario $\alpha = 4$, given the initial conditions (14), has been calculated in question 6. Therefore, we can quote this in this question, substituting in the value of p appropriately. The general solution is

$$y(x) = \frac{1+x}{p}\sin\left(\frac{px}{1+x}\right).$$

   First we consider the case when $p = 6$. This gives us the particular solution

$$y(x) = \frac{1+x}{6}\sin\left(\frac{6x}{1+x}\right).$$

   I have used this analytic solution to work out the global error and I have obtained the following results: Input into MATLAB: rungekutta7_1(12,6)

k	h	Y_n	$y(x_n)$	$\lvert E_n\rvert$
0	0.1	0.04711817458	0.04704000269	$7.817189807 \times 10^{-5}$
1	0.05	0.04704681924	0.04704000269	$6.816557345 \times 10^{-6}$
2	0.025	0.04704046843	0.04704000269	$4.657396744 \times 10^{-7}$
3	0.0125	0.04704003266	0.04704000269	$2.997232081 \times 10^{-8}$
4	0.00625	0.04704000458	0.04704000269	$1.894094904 \times 10^{-9}$
5	0.003125	0.04704000281	0.04704000269	$1.189339617 \times 10^{-10}$
6	0.0015625	0.04704000269	0.04704000269	$7.449161112 \times 10^{-12}$
7	0.00078125	0.04704000269	0.04704000269	$4.656431812 \times 10^{-13}$
8	0.000390625	0.04704000269	0.04704000269	$2.898552668 \times 10^{-14}$
9	0.0001953125	0.04704000269	0.04704000269	$1.639345801 \times 10^{-15}$
10	9.765625×10^{-5}	0.04704000269	0.04704000269	$3.31300068 \times 10^{-16}$
11	4.8828125×10^{-5}	0.04704000269	0.04704000269	$3.348337467 \times 10^{-16}$
12	$2.44140625 \times 10^{-5}$	0.04704000269	0.04704000269	$-7.45609936 \times 10^{-17}$

Table 11: Table showing how the accuracy of the Runge-Kutta method for this second order ODE increases for smaller values of h, at $x_n = 1$, for $p = 6$.

Input into MATLAB: rungekutta7_1(12,7)

k	h	Y_n	$y(x_n)$	$\lvert E_n\rvert$
0	0.1	-0.09994976666	-0.1002237793	0.000274012676
1	0.05	-0.1002052535	-0.1002237793	$1.85258815 \times 10^{-5}$
2	0.025	-0.100222638	-0.1002237793	$1.14134082 \times 10^{-6}$
3	0.0125	-0.1002237095	-0.1002237793	6.9846124×10^{-8}
4	0.00625	-0.100223775	-0.1002237793	$4.303880926 \times 10^{-9}$
5	0.003125	-0.1002237791	-0.1002237793	$2.668388916 \times 10^{-10}$
6	0.0015625	-0.1002237793	-0.1002237793	$1.660654289 \times 10^{-11}$
7	0.00078125	-0.1002237793	-0.1002237793	$1.035276388 \times 10^{-12}$
8	0.000390625	-0.1002237793	-0.1002237793	$6.488595329 \times 10^{-14}$
9	0.0001953125	-0.1002237793	-0.1002237793	$4.101242697 \times 10^{-15}$
10	9.765625×10^{-5}	-0.1002237793	-0.1002237793	$5.624068058 \times 10^{-16}$
11	4.8828125×10^{-5}	-0.1002237793	-0.1002237793	$1.575485316 \times 10^{-15}$
12	$2.44140625 \times 10^{-5}$	-0.1002237793	-0.1002237793	$1.950185587 \times 10^{-15}$

Table 12: Table showing how the accuracy of the Runge-Kutta method for this second order ODE increases for smaller values of h, at $x_n = 1$, for $p = 7$.

We can use a similar analysis to question 4 and obtain a relationship between the magnitude of the error, $|E_n|$, and the step length, h. By calculating the gradient, γ, of a log-log graph of $|E_n|$ vs h in Table 12, we can see that $|E_n|$ decreases proportional to h^γ, where $\gamma \approx 4.0116$. This implies the error is approximately $O(h^5)$ as $h \to 0$ and hence the error does behave as expected since we know the Runge-Kutta method should be $O(h^5)$ as $h \to 0$.

```
function [x_Final,Y_Final] = forwardeuler2(N,h)
% Forward Euler method to solve the first order Ordinary Differential Equations
% y'=(-16*y+15*exp(-x)).
% Input f(x,y), N is the number of iterations, h is the step length, x_0 is
% the initial x value and Y_0 is the corresponding initial y
% value.(x_0,Y_0) gives us a point on the solution curve.
syms x
syms y
f(x,y)=(-16*y+15*exp(-x))
x_0=0;
Y_0=0;
disp('Exact solution is m(x):')
m(x)=(exp(-x)-exp(-16*x))
count=1;
x_1=x_0;
Y_1=Y_0;
for count=1:N
    Y_2=Y_1+h*f(x_1,Y_1);
    x_1=x_0+count*h;
    E_1=double(Y_2-m(x_1));
    Y_1=Y_2;
    fprintf('$ %3g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ \\ hline \n',count,x_1,
        Y_2,m(x_1),E_1)
    %This output is here to make it easier to make tables from the data
    %obtained in MATLAB.
    count=count+1;
end
x_Final=x_1;
Y_Final=double(Y_1);
Actual_Value = double(m(x_1));
end
```

```matlab
function [x_vector,Y_vector,Error_vector] = forwardeuler2_3(N,h)
% Forward Euler method to solve the first order Ordinary Differential Equations
% y'=(-16*y+15*exp(-x)).
% Input f(x,y), N is the number of iterations, h is the step length, x_0 is
% the initial x value and Y_0 is the corresponding initial y
% value.(x_0,Y_0) gives us a point on the solution curve.
syms x
syms y
f(x,y)=(-16*y+15*exp(-x))
x_0=0;
Y_0=0;
disp('Exact solution is m(x):')
m(x)=(exp(-x)-exp(-16*x))
count=1;
x_1=x_0;
Y_1=Y_0;
E_1=0;
x_vector = zeros(1,N+1);
Y_vector = zeros(1,N+1);
Error_vector = zeros(1,N+1);
for count=1:N
    x_vector(1,count)=x_1;
    Y_vector(1,count)=Y_1;
    Error_vector(1,count)=E_1;
    Y_2=Y_1+h*f(x_1,Y_1);
    x_1=x_0+count*h;
    E_1=double(Y_2-m(x_1));
    Y_1=Y_2;
    fprintf('$ %3g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ \\ hline \n',count,x_1,
        Y_1,m(x_1),E_1)
    %This output is here to make it easier to make tables from the data
    %obtained in MATLAB.
    count=count+1;
end
x_vector(1,N+1)=x_1;
Y_vector(1,N+1)=Y_1;
Error_vector(1,N+1)=E_1;
x_Final=x_1;
Y_Final=double(Y_1);
Actual_Value = double(m(x_1));
end
```

```
function [x_vector,Y_vector,Error_vector] = rungekutta2_3(N,h)
% Runge Kutta method to solve first order Ordinary Differential Equations
% of the form y'=f(x,y).
% Input f(x,y), N is the number of iterations, h is the step length, x_0 is
% the initial x value and Y_0 is the corresponding initial y
% value.(x_0,Y_0) gives us a point on the solution curve.
syms x
syms y
f(x,y)=(-16*y+15*exp(-x))
x_0=0;
Y_0=0;
disp('Exact solution is m(x):')
m(x)=(exp(-x)-exp(-16*x))
count=1;
x_1=x_0;
Y_1=Y_0;
E_1=0;
x_vector = zeros(1,N+1);
Y_vector = zeros(1,N+1);
Error_vector = zeros(1,N+1);
for count=1:N
    k_1=h*f(x_1,Y_1);
    k_2=h*f(x_1+(1/2)*h,Y_1+(1/2)*k_1);
    k_3=h*f(x_1+(1/2)*h,Y_1+(1/2)*k_2);
    k_4=h*f(x_1+h,Y_1+k_3);
    x_vector(1,count)=x_1;
    Y_vector(1,count)=Y_1;
    Error_vector(1,count)=E_1;
    Y_2=Y_1+(1/6)*(k_1+2*k_2+2*k_3+k_4);
    x_1=x_0+count*h;
    E_1=double(Y_2-m(x_1));
    Y_1=Y_2;
    fprintf('$ %3g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ \\ hline \n',count,x_1,
        Y_2,m(x_1),E_1)
    count=count+1;
end
x_vector(1,N+1)=x_1;
Y_vector(1,N+1)=Y_1;
Error_vector(1,N+1)=E_1;
x_Final=x_1;
Y_Final=double(Y_1);
Actual_Value = double(m(x_1));
end
```

```
function [Error_vector,h_vector] = forwardeuler2_4(k)
% Forward Euler method to solve the first order Ordinary Differential Equations
% y'=(-16*y+15*exp(-x)).
% Input f(x,y), N is the number of iterations, h is the step length, x_0 is
% the initial x value and Y_0 is the corresponding initial y
% value.(x_0,Y_0) gives us a point on the solution curve.
syms x
syms y
f(x,y)=(-16*y+15*exp(-x))
x_0=0;
Y_0=0;
disp('Exact solution is m(x):')
m(x)=(exp(-x)-exp(-16*x))
count=1;
power=0;
Error_vector=zeros(1,k+1);
h_vector=zeros(1,k+1);
for power=0:k
    N=2^(power);
    h=(0.1)/(N);
    x_1=x_0;
    Y_1=Y_0;
    for count=1:N
    Y_2=double(Y_1+h*f(x_1,Y_1));
    x_1=x_0+count*h;
    E_1=double(Y_2-m(x_1));
    Y_1=Y_2;
    %This output is here to make it easier to make tables from the data
    %obtained in MATLAB.
    count=count+1;
    end
fprintf('$ %3g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ \\ hline \n',
    power,h,x_1,Y_2,m(x_1),E_1);
Error_vector(1,power+1)=E_1;
h_vector(1,power+1)=h;
power=power+1;
end
Error_vector;
h_vector;
x_Final=x_1;
Y_Final=double(Y_1);
Actual_Value = double(m(x_1));
end
```

```
function [Error_vector,h_vector] = rungekutta2_4(k)
% Runge Kutta method to solve first order Ordinary Differential Equations
% of the form y'=f(x,y).
% Input f(x,y), N is the number of iterations, h is the step length, x_0 is
% the initial x value and Y_0 is the corresponding initial y
% value.(x_0,Y_0) gives us a point on the solution curve.
syms x
syms y
f(x,y)=(-16*y+15*exp(-x))
x_0=0;
Y_0=0;
disp('Exact solution is m(x):')
m(x)=(exp(-x)-exp(-16*x))
count=1;
power=0;
Error_vector=zeros(1,k+1);
h_vector=zeros(1,k+1);
for power=0:k
    N=2^(power);
    h=(0.1)/(N);
    x_1=x_0;
    Y_1=Y_0;

for count=1:N
    k_1=double(h*f(x_1,Y_1));
    k_2=double(h*f(x_1+(1/2)*h,Y_1+(1/2)*k_1));
    k_3=double(h*f(x_1+(1/2)*h,Y_1+(1/2)*k_2));
    k_4=double(h*f(x_1+h,Y_1+k_3));
    Y_2=Y_1+(1/6)*(k_1+2*k_2+2*k_3+k_4);
    x_1=x_0+count*h;
    E_1=double(Y_2-m(x_1));
    Y_1=Y_2;
    %fprintf('$ %3g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ \\ hline \n
        ',count,h,x_1,Y_2,m(x_1),E_1)
    count=count+1;
end
fprintf('$ %3g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ \\ hline \n',
    power,h,x_1,Y_2,m(x_1),E_1);
Error_vector(1,power+1)=E_1;
h_vector(1,power+1)=h;
power=power+1;
end
Error_vector;
h_vector;
x_Final=x_1;
Y_Final=double(Y_1);
Actual_Value = double(m(x_1));
end
```

```
function [x_Final,Y_Final] = forwardeuler2_5(N,h)
% Forward Euler method to solve the first order Ordinary Differential Equations
% y'=(-16*y+15*exp(-x)).
% Input f(x,y), N is the number of iterations, h is the step length, x_0 is
% the initial x value and Y_0 is the corresponding initial y
% value.(x_0,Y_0) gives us a point on the solution curve.
syms x
syms y
f(x,y)=4*y-5*exp(-x)
x_0=0;
Y_0=1;
disp('Exact solution is m(x):')
m(x)=(exp(-x))
count=1;
x_1=x_0;
Y_1=Y_0;
for count=1:N
    Y_2=double(Y_1+h*f(x_1,Y_1));
    x_1=x_0+count*h;
    E_1=abs(double(Y_2-m(x_1)));
    Y_1=Y_2;
    fprintf('$ %3g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ \\ hline \n',count,x_1,
        Y_2,m(x_1),E_1)
    %This output is here to make it easier to make tables from the data
    %obtained in MATLAB.
    count=count+1;
end
x_Final=x_1;
Y_Final=double(Y_1);
Actual_Value = double(m(x_1));
end
```

```
function [x_Final,Y_Final,Actual_Value] = rungekutta2_5(N,h)
% Runge Kutta method to solve first order Ordinary Differential Equations
% of the form y'=f(x,y).
% Input f(x,y), N is the number of iterations, h is the step length, x_0 is
% the initial x value and Y_0 is the corresponding initial y
% value.(x_0,Y_0) gives us a point on the solution curve.
syms x
syms y
f(x,y)=4*y-5*exp(-x)
x_0=0;
Y_0=1;
disp('Exact solution is m(x):')
m(x)=(exp(-x))
count=1;
x_1=x_0;
Y_1=Y_0;

for count=1:N
    k_1=double(h*f(x_1,Y_1));
    k_2=double(h*f(x_1+(1/2)*h,Y_1+(1/2)*k_1));
    k_3=double(h*f(x_1+(1/2)*h,Y_1+(1/2)*k_2));
    k_4=double(h*f(x_1+h,Y_1+k_3));

    Y_2=Y_1+(1/6)*(k_1+2*k_2+2*k_3+k_4);
    x_1=x_0+count*h;
    E_1=abs(double(Y_2-m(x_1)));
    Y_1=Y_2;

    fprintf('$ %3g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ \\ hline \n',count,x_1,
        Y_2,m(x_1),E_1)
    count=count+1;
end
x_Final=x_1;
Y_Final=double(Y_1);
Actual_Value = double(m(x_1));
end
```

```
function [x_Final,Y_Final,Actual_Value] = rungekutta7_1(k,p)
% Runge Kutta method to solve first order Ordinary Differential Equations
% of the form y'=f_1(x,y,z) and z'=f_2(x,y,z)
% Input f(x,y), N is the number of iterations , h is the step length , x_0 is
% the initial x value and Y_0 is the corresponding initial y
% value.(x_0,Y_0) gives us a point on the solution curve.
syms x
syms y
syms z
f_1(x,y,z)= z
%This is (dy/dx) = z
f_2(x,y,z)= -p^(2)*((1+x)^(-4))*y;
%This is (dz/dx) = y''
x_0=0
Y_0=0
Z_0=1
disp('Exact solution y is m_1(x):')
m_1(x)= ((1+x)/p)*sin((p*x)/(1+x))
%disp('Exact solution dy/dx is m_2(x):')
%m_2(x)=(1/p)*sin((p*x)/(1+x))+(1/(1+x))*cos((p*x)/(1+x))
count=1;
x_1=x_0;
Z_1=Z_0;
Y_1=Y_0;
Y_2=[Y_1,Z_1]';
power=0;
for power=0:k
    h =(0.1)/(2^(power));
    N=1/h;
    x_1=x_0;
    Y_1=Y_0;
    Z_1=Z_0;
    Y_2=[Y_1,Z_1]';
for count=1:N
    k_11=double(h*f_1(x_1,Y_1,Z_1));
    k_12=double(h*f_2(x_1,Y_1,Z_1));
    k_21=double(h*f_1(x_1+(1/2)*h,Y_1+(1/2)*k_11,Z_1+(1/2)*k_12));
    k_22=double(h*f_2(x_1+(1/2)*h,Y_1+(1/2)*k_11,Z_1+(1/2)*k_12));
    k_31=double(h*f_1(x_1+(1/2)*h,Y_1+(1/2)*k_21,Z_1+(1/2)*k_22));
    k_32=double(h*f_2(x_1+(1/2)*h,Y_1+(1/2)*k_21,Z_1+(1/2)*k_22));
    k_41=double(h*f_1(x_1+h,Y_1+k_31,Z_1+k_32));
    k_42=double(h*f_2(x_1+h,Y_1+k_31,Z_1+k_32));

    k_1=[k_11,k_12]';
    k_2=[k_21,k_22]';
    k_3=[k_31,k_32]';
    k_4=[k_41,k_42]';
    Y_3=Y_2+(1/6)*(k_1+2*k_2+2*k_3+k_4);
    x_1=x_0+count*h;
    Y_2=Y_3;
    Y_1=Y_2(1,1);
    Z_1=Y_2(2,1);
    E_1=double(Y_2(1,1)-m_1(x_1));
    count=count+1;
    %fprintf('$ %3g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ & $ %2.6g $ \\ hline \n
        ',count,h,x_1,Y_2(1,1),m_1(x_1),E_1)
end
   fprintf('$ %3g $ & $ %2.10g $ & $ %2.10g $ & $ %2.10g $ & $ %2.10g $ \\ hline \n',power,h,
       Y_2(1,1),m_1(x_1),E_1)
   power=power+1;
end
x_Final=x_1;
Y_Final=double(Y_1);
Actual_Value = double(m_1(x_1));
end
```