1.2 Ordinary Differential Equations

QUESTION 1

The program used to output the results of Euler's method, the analytic value, error, and growth rate is **Code 1** and **Code 2** found on page 15, labelled as

```
estimation(h)
```

I used Code 1 for h=0.6, 0.4 and 0.2, and Code 2 for h=0.125 and 0.1 – Code 2 gives results to more decimal places, required as the values became smaller.

For the programs used in questions 1 – 4, each of them have a separate program defining f(x, y) written. This is written under **Code 1** too, simply just labelled as

```
f(x,y)
```

In all of my codes for this project, I have used variable 'Z_n' to represent the analytic solution, 'E_n' to represent the error, and G_n to represent the growth rate.

From the results, it can be seen that as the value of h decreases, the size of the error decreases, meaning that instability decreases, and growth rate inclusively decreases. Hence Euler's method becomes more accurate as h gets smaller.

Table 1: h = 0.6

x_n	Y_n	$y(x_n)$	E_n	$\gamma=\ln(\lvert E_n\rvert)/x_n$
0.6	9.0000	0.5487	8.4513	3.5572
1.2	-72.4607	0.3012	-72.7619	3.5727
1.8	625.8727	0.1653	625.7074	3.5772
2.4	-5381.0178	0.0907	-5381.1085	3.5794
3.0	46277.5692	0.0498	46277.5194	3.5808
3.6	-397986.6474	0.0273	-397986.6747	3.5817
4.2	3422685.4131	0.0150	3422685.3982	3.5824
4.8	-29435094.4181	0.0082	-29435094.4263	3.5829
5.4	253141812.0697	0.0045	253141812.0652	3.5832
6.0	-2177019583.7592	0.0025	-2177019583.7616	3.5835

For the following tables, I only presented the output for the starting values of x_n, and the end values, but for values of h that gave more output, I included some middle values too.

Table 2: h = 0.4

x_n	Y_n	y_n	E_n	$\gamma=\ln(\lvert E_n\rvert)/x_n$
0.4	6.0000	0.6687	5.3313	4.1840
0.8	-28.3781	0.4493	-28.8274	4.2017
1.2	155.9376	0.3012	155.6364	4.2063
1.6	-840.2559	0.2019	-840.4578	4.2087
2.0	4538.5932	0.1353	4538.4579	4.2102
4.4	112530699.7509	0.0123	112530699.7386	4.2133
4.8	-607665778.5809	0.0082	-607665778.5892	4.2136
5.2	3281395204.3864	0.0055	3281395204.3809	4.2138
5.6	-17719534103.6535	0.0037	-17719534103.6572	4.2139
6.0	95685484159.7509	0.0025	95685484159.7484	4.2141

Table 3: h = 0.2

x_n	Y_n	y_n	E_n	$\gamma=\ln(\lvert E_n\rvert)/x_n$
0.2	3.0000	0.7780	2.2220	3.9921
0.4	-4.1438	0.6687	-4.8125	3.9280
0.6	11.1273	0.5487	10.5786	3.9314
0.8	-22.8337	0.4493	-23.2830	3.9347
1.0	51.5821	0.3679	51.2143	3.9360
5.2	-794852559.3851	0.0055	-794852559.3906	3.9411
5.4	1748675630.6637	0.0045	1748675630.6592	3.9411
5.6	-3847086387.4466	0.0037	-3847086387.4503	3.9412
5.8	8463590052.3935	0.0030	8463590052.3905	3.9412
6.0	-18619898115.2567	0.0025	-18619898115.2592	3.9412

Table 4: h = 0.125

x_n	Y_n	y_n	E_n	$\gamma=\ln(\lvert E_n\rvert)/x_n$
0.125	1.8750000000	0.7471616	1.1278383807	0.962423
0.250	-0.2203183077	0.7604851	-0.9808034518	-0.077533
0.375	1.6805697759	0.6848105	0.9957592493	-0.011333
0.500	-0.3919023782	0.6061952	-0.9980975753	-0.003808
0.625	1.5291473651	0.5352160	0.9939313366	-0.009739
2.625	1.0681688462	0.0724398	0.9957290892	-0.001630
2.750	-0.9323443018	0.0639279	-0.9962721630	-0.001358
2.875	1.0522090415	0.0564161	0.9957929020	-0.001466
3.000	-0.9464287800	0.0497871	-0.9962158483	-0.001264
3.125	1.0397795332	0.0439369	0.9958425995	-0.001333
3.250	-0.9573977826	0.0387742	-0.9961719904	-0.001180
3.375	1.0300994223	0.0342181	0.9958813040	-0.001223
5.500	-0.9919470789	0.0040868	-0.9960338503	-0.000723
5.625	0.9996097753	0.0036066	0.9960032122	-0.000712
5.750	-0.9928474695	0.0031828	-0.9960302503	-0.000692
5.875	0.9988151835	0.0028088	0.9960063893	-0.000681
6.000	-0.9935486943	0.0024788	-0.9960274465	-0.000663

Table 5: h = 0.1

0.1	1.5000000000	0.7029409	0.7970591000	-2.268264
0.2	0.4572561271	0.7779685	-0.3207124220	-5.686052
0.3	0.9537424534	0.7325885	0.2211539798	-5.029654
0.4	0.5389818590	0.6686585	-0.1296766298	-5.106778
0.5	0.6820909537	0.6061952	0.0758957566	-5.156789
3.4	0.0332659604	0.0333733	-0.0001073096	-2.688174
3.5	0.0301003287	0.0301974	-0.0000970547	-2.640067
3.6	0.0272358779	0.0273237	-0.0000878445	-2.594428
3.7	0.0246440569	0.0247235	-0.0000794695	-2.551388

5.5	0.0040736342	0.0040868	-0.0000131372	-2.043647
5.6	0.0036859766	0.0036979	-0.0000118871	-2.025011
5.7	0.0033352096	0.0033460	-0.0000107559	-2.007028
5.8	0.0030178224	0.0030276	-0.0000097323	-1.989665
5.9	0.0027306387	0.0027394	-0.0000088062	-1.972891
6.0	0.0024707840	0.0024788	-0.0000079681	-1.956677

QUESTION 2

(i) We wish to find the analytic solution of the following Euler difference equation:

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

Rearrange this, and we get

$$Y_{n+1} + (16h - 1)Y_n = 15he^{-nh}$$

Complementary solution:

Auxiliary equation: $\lambda^2 + (16h - 1)\lambda = 0$ so $\lambda = 0\ or\ 1 - 16h$

Hence ${Y_n}^c = A(1 - 16h)^n$, where A is some constant.

Particular Integral:

Choose a particular integral of the form ${Y_n}^p = Be^{-hn}$:

$${Y_{n+1}}^p = Be^{-h(n+1)} = Be^{-h}e^{-hn}$$

Substitute into the difference equation:

$$Be^{-h}e^{-hn} + (16h - 1)Be^{-hn} = 15he^{-hn}$$

$$so\ Be^{-h} + (16h - 1)B = 15h$$

$$\boldsymbol{B = \frac{15h}{16h - 1 + e^{-h}}}$$

General solution:

$$Y_n = A(1 - 16h)^n + Be^{-hn}$$

At n = 0, we need:

$$0 = A + B\ so\ A = -B$$

$$so\ \boldsymbol{A = \frac{-15h}{16h - 1 + e^{-h}}}$$

(ii) As $n \to \infty$, the magnitude of $(1 - 16h)^n$ becomes larger and larger if $1 - 16h < -1$, which would require h > 1/8, whilst the e^{-hn} term just goes to zero. Overall, for h > 1/8, it would just diverge as $n \to \infty$, leading to instability. However, when 0 < h < 1/8, we have $-1 < 1 - 16h < 0$, meaning that as $n \to \infty$, that power will also tend to zero, leading to stability in this case.

To determine growth rate, like we did in question 1, we can write the error as

$$E_n = e^{\gamma x}$$

Taking logs on both sides and substituting x = nh, we get that

$$\gamma = \frac{1}{nh}\log(E_n)$$

For h > 1/8, the error comes from the first term, $\frac{-15h}{16h-1+e^{-h}}(1-16h)^n$, so we can just substitute that in place of E_n:

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

Then, when taking $n \to \infty$, we are just left with:

$$\gamma = \frac{1}{h}\log(1-16h)$$

(iii) Substituting $x_n = nh$ into the equation,

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

As h → 0, both $-15h$ and $16h-1+e^{-h}$ tend to zero too, so we have to use L'Hôpital's Rule to calculate the limit:

$$\lim_{h\to 0}\left(\frac{-15h}{16h-1+e^{-h}}\right) = \lim_{h\to 0}\left(\frac{-15}{16-e^{-h}}\right) = -1$$

Hence the coefficient of e^{-x_n} tends to 1.

We also have that:

$$y = \lim_{h\to 0}(1-16h)^{\frac{1}{h}}$$

$$\ln(y) = \ln\left(\lim_{h\to 0}(1-16h)^{\frac{1}{h}}\right)$$

$$\ln(y) = \lim_{h\to 0}\left(\ln(1-16h)^{\frac{1}{h}}\right) = \lim_{h\to 0}\left(\frac{1}{h}\ln(1-16h)\right)$$

$$= \lim_{h\to 0}\left(\frac{1}{h}\left(-16h - \frac{(-16)^2}{2} - \frac{(-16h)^3}{3} - \cdots\right)\right)$$

$$= \lim_{h\to 0}\left(-16 - \frac{16^2 h}{2} - \frac{16^3 h^2}{3} - \cdots\right) = -16$$

$$\ln(y) = -16, so\ y = e^{-16}$$

Hence, $\lim_{h\to 0}(1-16h)^{\frac{x_n}{h}} = e^{-16x_n}$.

Therefore, we get, overall, that as h → 0 ,

$$Y_n \to (-1)e^{-16x_n} + e^{-x_n} = -e^{-16x_n} + e^{-x_n}$$

So, we get that Y_n → $y(x_n)$, as required.

QUESTION 3

The programs used to integrate the ODE specified numerically with h = 0.05 from x = 0 to x = 4 using both the Euler and RK4 methods are **Code 3** and **Code 4** on page 16, labelled as

`euler(h)`

`& RK4(h)`

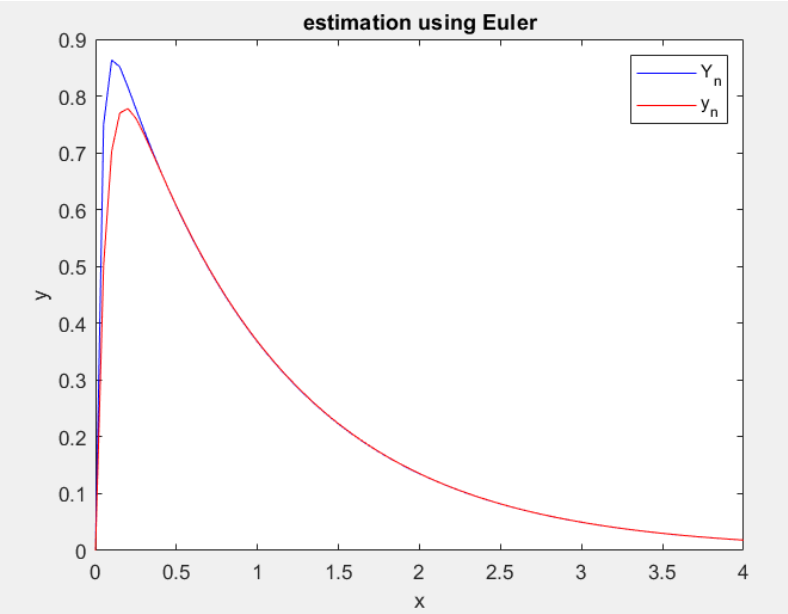

Figure 1: *A graph displaying the estimation of the differential equation* $\frac{dy}{dx} = -16y + 15e^{-x}$ *using Euler's method, alongside the analytical solution* $y = e^{-x} - e^{-16x}$.

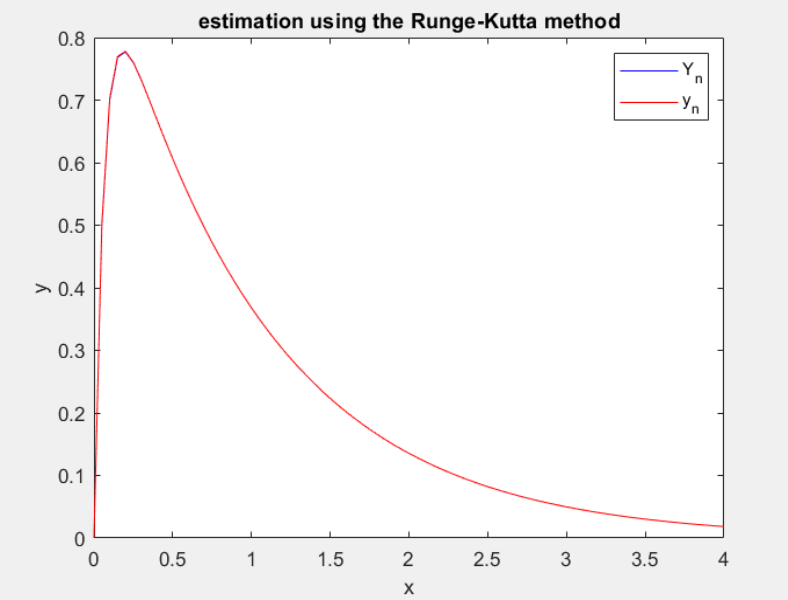

Figure 2: *A graph displaying the estimation of the differential equation* $\frac{dy}{dx} = -16y + 15e^{-x}$ *using the Runge-Kutta method, alongside the analytical solution* $y = e^{-x} - e^{-16x}$.

As expected, due to the error of Euler being $O(h^2)$ and the error of RK4 being $O(h^5)$, the estimation using RK4 aligns a lot more accurately with the analytic solution.

QUESTION 4

The programs used to 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, . . ., 15 and plot a log–log graph of |En| against h over this range for both the Euler and RK4 methods is **Code 5** and **Code 6** on page 17, labelled as

```
euler_2(x_max)

& RK4_2(x_max)
```

Table 6: the error from Euler's method along with the value of h used.

Euler's Method	
h	E_n
0.1000000000000000	0.7970590999586960
0.0500000000000000	0.1604811683342310
0.0250000000000000	0.0716563659683888
0.0125000000000000	0.0338186499580592
0.0062500000000000	0.0164449388058929
0.0031250000000000	0.0081110427192849
0.0015625000000000	0.0040282337183389
0.0007812500000000	0.0020073650057734
0.0003906250000000	0.0010020032263450
0.0001953125000000	0.0005005828754077
0.0000976562500000	0.0002501868881529
0.0000488281250000	0.0001250673235322
0.0000244140625000	0.0000625271337331
0.0000122070312500	0.0000312619351167
0.0000061035156250	0.0000156305596561
0.0000030517578125	0.0000078151778747

Table 7: the error from the Runge-Kutta method along with the value of h used.

Runge-Kutta's Method	
h	E_n
0.1000000000000000	-0.0680922447919430
0.0500000000000000	-0.0021483082826023
0.0250000000000000	-0.0000954067299630
0.0125000000000000	-0.0000050373166091
0.0062500000000000	-0.0000002894705288
0.0031250000000000	-0.0000000173492359
0.0015625000000000	-0.0000000010618582
0.0007812500000000	-0.0000000000656750
0.0003906250000000	-0.0000000000040833
0.0001953125000000	-0.0000000000002546
0.0000976562500000	-0.0000000000000164
0.0000488281250000	-0.0000000000000001
0.0000244140625000	-0.0000000000000022
0.0000122070312500	-0.0000000000000023
0.0000061035156250	-0.0000000000000028
0.0000030517578125	0.0000000000000145

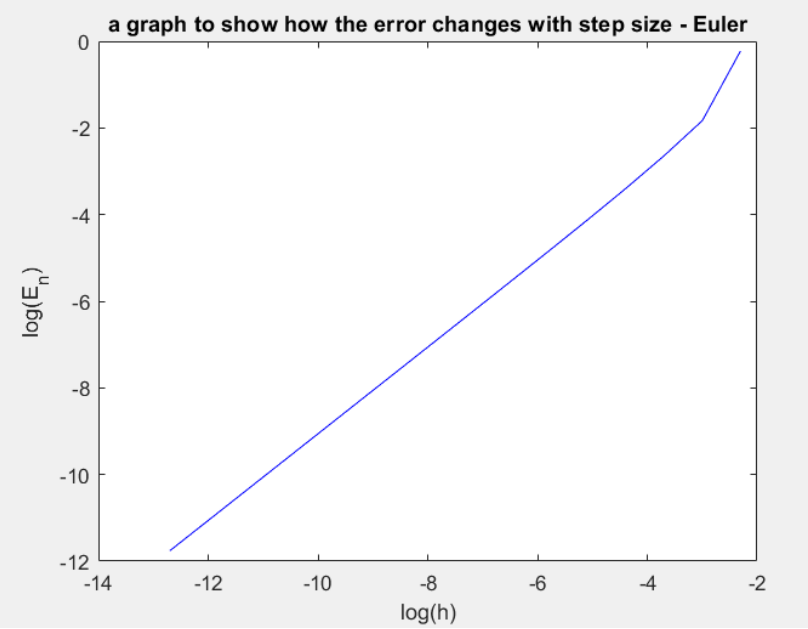

Figure 3: *A graph showing how the error from Euler's method changes with the value of h used.*

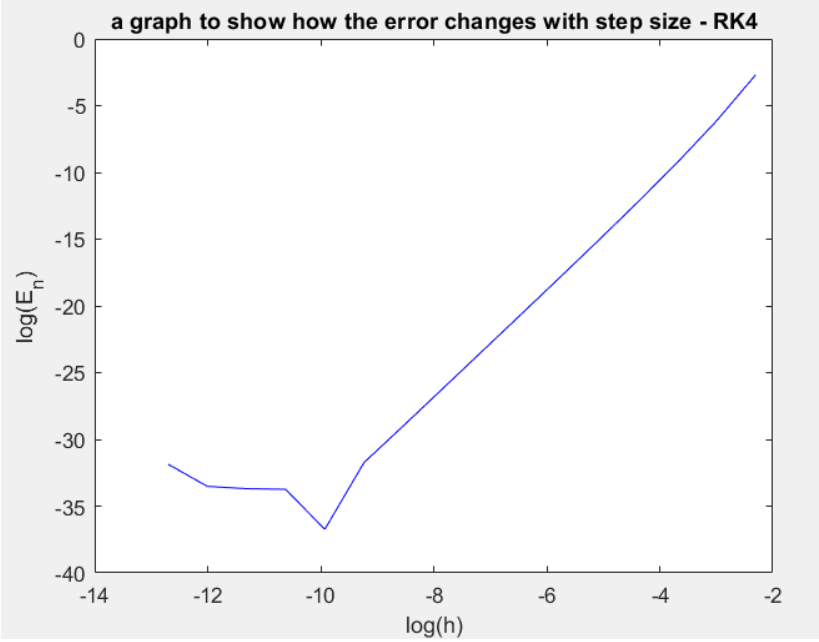

Figure 4: *A graph showing how the error from the Runge-Kutta method changes with the value of h used.*

Looking at the graph formed using the Euler method, it looks like there's a really strong linear relationship between $\log(E_n)$ and $\log(h)$, meaning we can think of the relationship between E_n and h as $E_n = Ah^m$, where A is a constant such that $\log(A)$ is the y-intercept of the graph presented above, and m is the gradient of the graph – looking at the graph, it looks like the graph has a gradient between 1 and 2.

On the other hand, whilst the RK4 method also gives a relationship between E_n and h as $E_n = Ah^m$, the value of m changes in this case, and is approximately between 4 and 5 instead. This aligns with what is expected, as it's known that the Euler method has first-order accuracy, and the Runge-Kutta method has fourth-order accuracy ($O(h^2)$ and $O(h^5)$).

QUESTION 5

$$\frac{d^2y}{dt^2} + \gamma\frac{dy}{dt} + y = \sin(\omega t)$$

, where γ and ω are non-negative real constants, and t and y are real variables.

To find the general solution of this second-order ODE, we need to find a 'complementary function' and a 'particular integral'; let's begin with the complementary function.

Begin with the auxiliary equation:

$$\lambda^2 + \gamma\lambda + 1 = 0$$

$$\lambda = \frac{-\gamma \pm \sqrt{\gamma^2 - 4}}{2}$$

As we are dealing with the case where $0 < \gamma < 2$ (lightly-damped case), we get complex λ:

$$\lambda = -\frac{\gamma}{2} \pm \frac{\sqrt{4-\gamma^2}}{2} i$$

And so the general complementary solution is

$$y_c = e^{-\frac{\gamma}{2}t}\left(A\sin\left(\frac{\sqrt{4-\gamma^2}}{2}t\right) + B\cos\left(\frac{\sqrt{4-\gamma^2}}{2}t\right)\right)$$

To get the particular integral, we take a general form

$$y_p = \alpha\sin(\omega t) + \beta\cos(\omega t)$$

$$\dot{y}_p = -\beta\omega\sin(\omega t) + \alpha\omega\cos(\omega t)$$

$$\ddot{y}_p = -\alpha\omega^2\sin(\omega t) - \beta\omega^2\cos(\omega t)$$

Substitute this into the differential equation, and we get

$$-\alpha\omega^2\sin(\omega t) - \beta\omega^2\cos(\omega t) - \gamma\beta\omega\sin(\omega t) + \gamma\alpha\omega\cos(\omega t) + \alpha\sin(\omega t) + \beta\cos(\omega t) = \sin(\omega t)$$

Comparing sin and cos coefficients, we get:

$$-\beta\omega^2 + \gamma\alpha\omega + \beta = 0$$

$$-\alpha\omega^2 - \gamma\beta\omega + \alpha = 1$$

$$so\ \alpha = \frac{1-w^2}{\gamma^2w^2 + (1-w^2)^2}\ ,\beta = \frac{-\gamma\omega}{\gamma^2w^2 + (1-w^2)^2}$$

So, the full general solution is:

$$y = e^{-\frac{\gamma}{2}t}\left(A\sin\left(\frac{\sqrt{4-\gamma^2}}{2}t\right) + B\cos\left(\frac{\sqrt{4-\gamma^2}}{2}t\right)\right) + \frac{1-w^2}{\gamma^2w^2 + (1-w^2)^2}\sin(\omega t) + \frac{-\gamma\omega}{\gamma^2w^2 + (1-w^2)^2}\cos(\omega t)$$

As t → ∞, the complementary part of the general solution tends towards zero, due to the exponential decaying factor, so only $\frac{1-w^2}{\gamma^2w^2+(1+w^2)^2}\sin(\omega t) + \frac{-\gamma\omega}{\gamma^2w^2+(1+w^2)^2}\cos(\omega t)$ remains. We can write this as $A_s\sin(\omega t - \varphi_s)$, where $A_s = \sqrt{\left(\frac{\gamma\omega}{\gamma^2w^2+(1+w^2)^2}\right)^2 + \left(\frac{1-w^2}{\gamma^2w^2+(1+w^2)^2}\right)^2}$, and $\tan(\varphi_s) = \frac{\gamma\omega}{1-w^2}$

When we have initial conditions $y = \frac{dy}{dt} = 0$ at t = 0, we get $A = \frac{2}{\sqrt{4-\gamma^2}}\left(\frac{\gamma}{2}B - \frac{\omega(1-\omega^2)}{(\gamma\omega)^2+(1-\omega^2)^2}\right)$ and $B = \frac{\gamma\omega}{(\gamma\omega)^2+(1-W^2)^2}$. Use v for γ and w for ω in the program.

PROGRAMMING TASK: The program to solve equation (8) using the RK4 method, subject to the initial conditions given in the question, is **Code 7** found on page 18, labelled as

```
vector_RK4(t,Y,v,w)
```

For the programs used in questions 5 – 7, each of them have a separate program defining **f**(t, **y**):

```
vector_F(t,Y,v,w)
```

QUESTION 6

Completed using Code 7, with input parameters (h, t_max, v, w) = (h, 10, 1, $\sqrt{3}$)

The following tables show the estimation of equations (8) and (14) with RK4 alongside the analytic solution and the error, based on the value of h used.

Table 8: h = 0.4

x_n	Y_n	$y(x_n)$	E_n
0.4	0.0162971236	0.0162278362	0.000069287412672
0.8	0.1070963120	0.1070522266	0.000044085383481
1.2	0.2773505163	0.2773567032	-0.000006186865723
1.6	0.4631758946	0.4631933612	-0.000017466593964
2.0	0.5700184208	0.5699822604	0.000036160404415
2.4	0.5251494621	0.5250170916	0.000132370458332
2.8	0.3190649415	0.3188491953	0.000215746258683
3.2	0.0160156095	0.0157863441	0.000229265347967
3.6	-0.2714218078	-0.2715679761	0.000146168323902
4.0	-0.4319107011	-0.4318973151	-0.000013386054430
4.4	-0.4058967620	-0.4057078188	-0.000188943208563
4.8	-0.2134144126	-0.2131078366	-0.000306575937985
5.2	0.0540853311	0.0543985779	-0.000313246767075
5.6	0.2744934239	0.2746958609	-0.000202436944524
6.0	0.3498907332	0.3499095087	-0.000018775531686
6.4	0.2503999126	0.2502390569	0.000160855668998
6.8	0.0269379097	0.0266765955	0.000261314223600
7.2	-0.2129684880	-0.2132112673	0.000242779284817
7.6	-0.3551652579	-0.3552837544	0.000118496516928
8.0	-0.3316529253	-0.3316014002	-0.000051525026423
8.4	-0.1518953440	-0.1517073016	-0.000188042433758
8.8	0.1017122210	0.1019409735	-0.000228752484986
9.2	0.3120639354	0.3122204786	-0.000156543167716
9.6	0.3815799323	0.3815868268	-0.000006894474980
10.0	0.2774153301	0.2772664187	0.000148911446708

For h = 0.2 and 0.1, I've chosen to only present the output of the starting and ending values of x_n.

Table 9: h = 0.2

x_n	Y_n	$y(x_n)$	E_n
0.2	0.0021829778	0.0021807243	0.000002253537015
0.4	0.0162307799	0.0162278362	0.000002943648518
0.6	0.0501524419	0.0501499043	0.000002537562163
0.8	0.1070538055	0.1070522266	0.000001578827502
1.0	0.1849714771	0.1849708731	0.000000603938541
1.2	0.2773567706	0.2773567032	0.000000067476383
1.4	0.3741344280	0.3741341443	0.000000283702529
1.6	0.4631947516	0.4631933612	0.000001390445182

1.8	0.5321285944	0.5321252560	0.000003338415716
2.0	0.5699881665	0.5699822604	0.000005906079256
8.2	-0.2574334017	-0.2574228363	-0.000010565336770
8.4	-0.1517200932	-0.1517073016	-0.000012791629935
8.6	-0.0269912530	-0.0269778917	-0.000013361286625
8.8	0.1019287529	0.1019409735	-0.000012220635315
9.0	0.2196831393	0.2196926609	-0.000009521559330
9.2	0.3122148761	0.3122204786	-0.000005602516008
9.4	0.3684440131	0.3684449606	-0.000000947473490
9.6	0.3815906992	0.3815868268	0.000003872336120
9.8	0.3499865824	0.3499783155	0.000008266914180
10.0	0.2772781168	0.2772664187	0.000011698099165

Table 10: h = 0.1

x_n	Y_n	$y(x_n)$	E_n
0.1	0.00028110657	0.00028103514	0.0000000714296214
0.2	0.00218084223	0.00218072427	0.0000001179554627
0.3	0.00711216239	0.00711201960	0.0000001427963473
0.4	0.01622798588	0.01622783622	0.0000001496598164
0.5	0.03039044079	0.03039029821	0.0000001425792103
0.6	0.05015003008	0.05014990433	0.0000001257495614
0.7	0.07573517702	0.07573507366	0.0000001033668770
0.8	0.10705230612	0.10705222664	0.0000000794752200
0.9	0.14369630561	0.14369624778	0.0000000578257059
1.0	0.18497091490	0.18497087315	0.0000000417511301
9.1	0.2698891181	0.2698894729	-0.000000354782555
9.2	0.3122202609	0.3122204786	-0.000000217694665
9.3	0.3454073686	0.3454074417	-0.000000073094070
9.4	0.3684450351	0.3684449606	0.000000074549966
9.5	0.3806309685	0.3806307478	0.000000220679970
9.6	0.3815871876	0.3815868268	0.000000360787739
9.7	0.3712715006	0.3712710100	0.000000490550222
9.8	0.3499789214	0.3499783155	0.000000605959737
9.9	0.3183329818	0.3183322784	0.000000703444629
10.0	0.2772671986	0.2772664187	0.000000779976764

Generally, we can see that as h gets closer to 0, the error gets, meaning the RK4 method becomes more accurate. More specifically, we can see that when h decreases by a factor of 2, the error decreases, on average, by a factor of 10, supporting the known fact that the error of RK4 is $O(h^5)$.

QUESTION 7

The program used for this question is **Code 8** on page 18, labelled as

```
vector_RK4_plot(h,t_max,v,w)
```

The only difference between **Code 7** and **Code 8** is that Code 8 has additional code written at the end to plot the graphs, whereas Code 7 does not have that.

Use h = 0.1 for this question, as it's the most accurate out of the three values of h we tested in question 6.

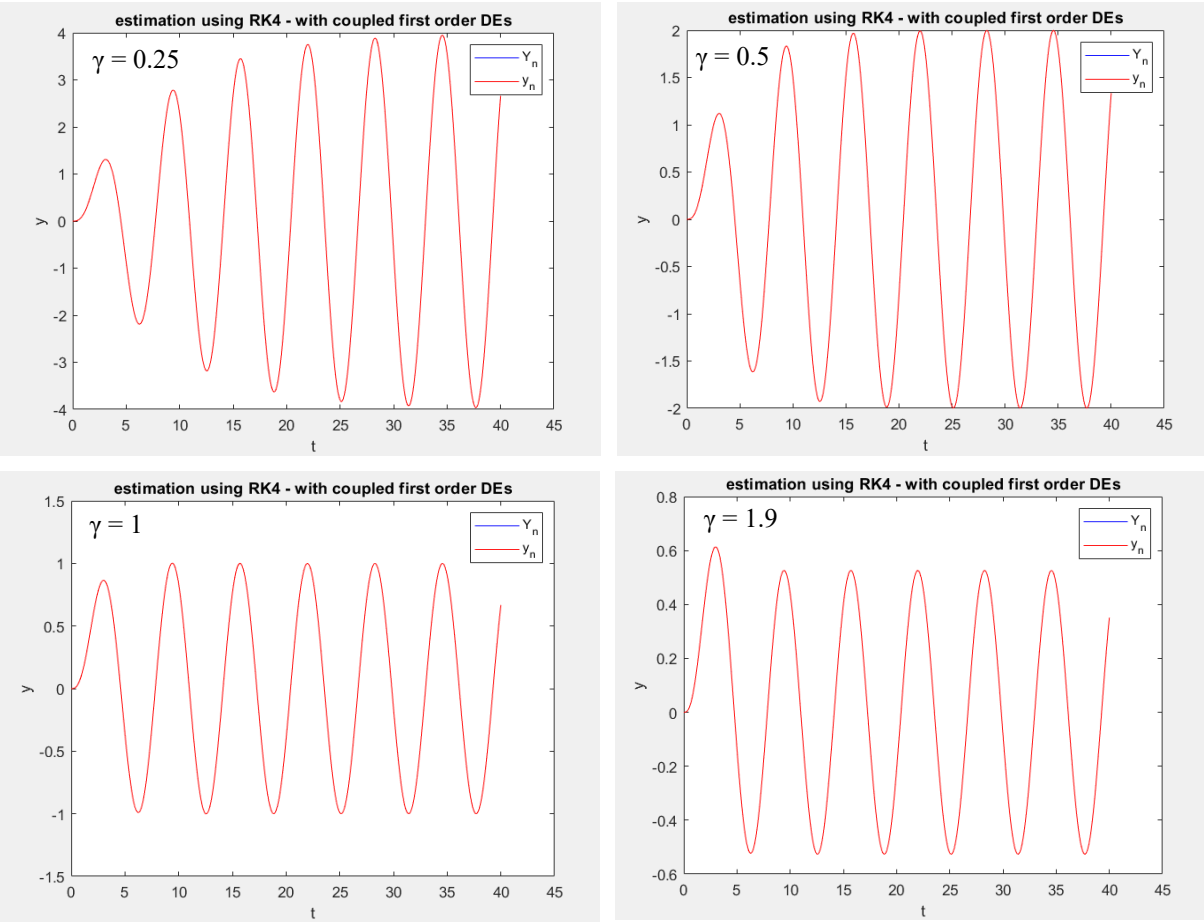

Figure 5: *Four graphs showing how the estimation of the Runge-Kutta method alongside the analytic solution changes with various γ for equations (8) and (14), with ω = 1*

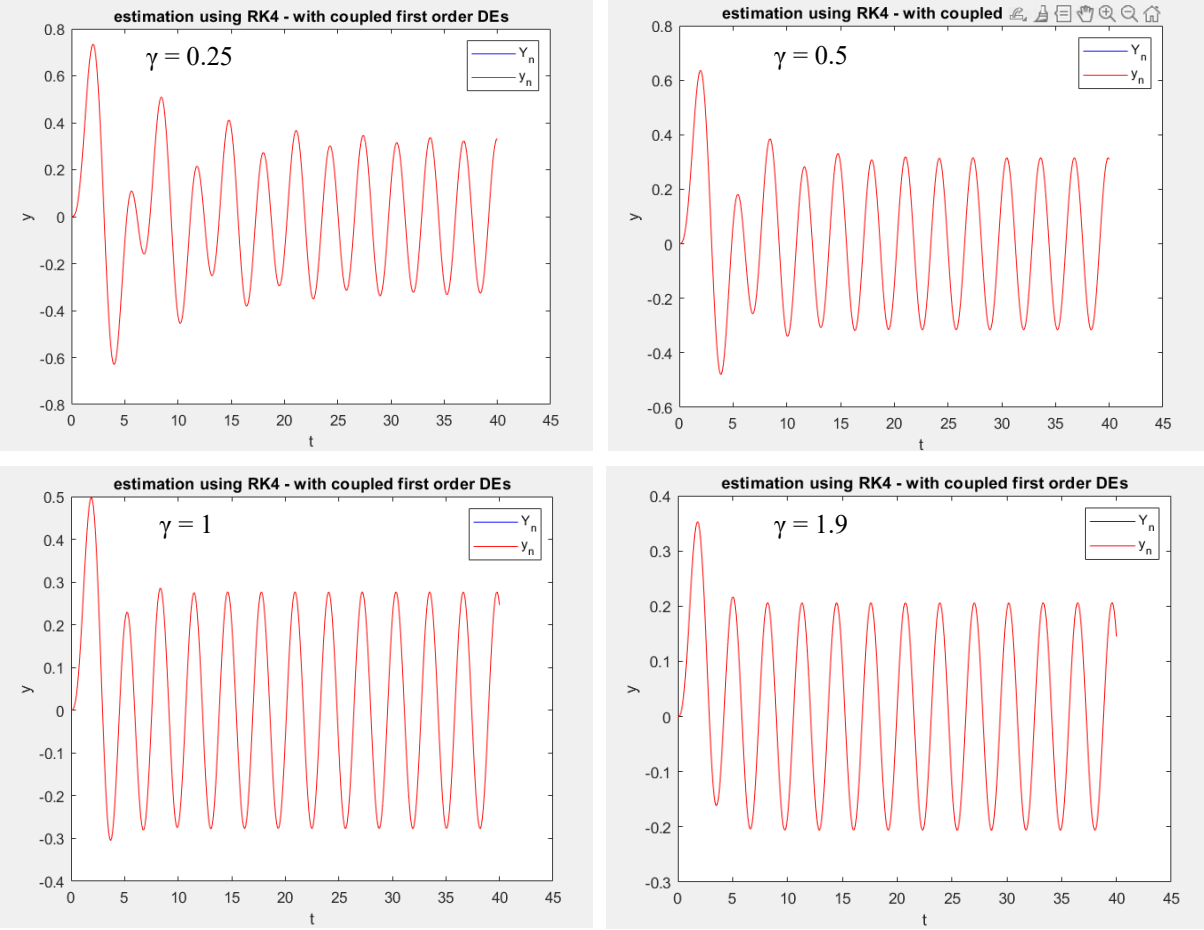

Figure 6: *Four graphs showing how the estimation of the Runge-Kutta method alongside the analytic solution changes with various γ for equations (8) and (14), with ω = 2*

Firstly, we can see that the graph formed by the estimation using RK4 aligns extremely well with the graph of the analytic solution, so the numerical solutions do align with the analytic solution.

Secondly, we can notice that as the value of γ increases, the magnitude of the y-value at the peaks decreases. Looking at the $\omega = 1$ case, we see more specifically that as γ increases by a factor of 2, the magnitude of the y-value at the peaks halves.

- To be able to explain this, refer back to the analytic solution:

$$y = e^{-\frac{\gamma}{2}t}\left(A\sin\left(\frac{\sqrt{4-\gamma^2}}{2}t\right) + B\cos\left(\frac{\sqrt{4-\gamma^2}}{2}t\right)\right) + \frac{1-\omega^2}{\gamma^2\omega^2+(1-\omega^2)^2}\sin(\omega t) + \frac{-\gamma\omega}{\gamma^2\omega^2+(1-\omega)^2}\cos(\omega t)$$

  , where $A = \frac{2}{\sqrt{4-\gamma^2}}\left(\frac{\gamma}{2}B - \frac{\omega(1-\omega^2)}{(\gamma\omega)^2+(1-\omega^2)^2}\right)$ and $B = \frac{\gamma\omega}{(\gamma\omega)^2+(1-\omega^2)^2}$.
- As the value of γ increases, the exponential factor in front of the complementary solution decreases, and so does the factor of $\frac{\sqrt{4-\gamma^2}}{2}$ (although the exponential factor will have much more of an impact in this case).
- Furthermore, the denominator of the coefficients in the particular solution, $\gamma^2\omega^2 + (1-\omega^2)^2$, increases, meaning the coefficients altogether decrease (the coefficient of $\cos(\omega t)$ does have a γ in the numerator, but the higher order γ^2 in the denominator would have more of an impact).

- Finally, looking back at the original differential equation, we see that γ is the damping coefficient in the equation, hence it's easy to conclude that as the damping coefficient increases, the magnitude of y at the peaks would decrease.

We can also notice that when ω increases by a factor of 2, the width of the oscillations inclusively increase by approximately a factor of 2.

- This is due to the frequency of the forcing term in the differential equation changing. When an external force is applied to a damped oscillator (or any oscillator), the oscillator is forced to shift from its natural frequency to the external force's frequency.
- Initially, $\omega = 1$, so the oscillator's frequency becomes that of $\sin(t)$: 2π. Then, when $\omega = 2$, the frequency becomes around π for the same reason.

QUESTION 8

The program used to generate and plot numerical solutions to equations (15) and (16) for t up to 6, using h = 0.1 again as it's the most accurate out of the various h used in question 6, is **Code 9** on page 19**,** labelled as

```
RK4_vector(h,t_max,delta)
```

This program inclusively implements a separately written code for **f(**t, **y**) in this problem. This is also under **Code 9**, labelled as

```
F_vector(t,Y,delta)
```

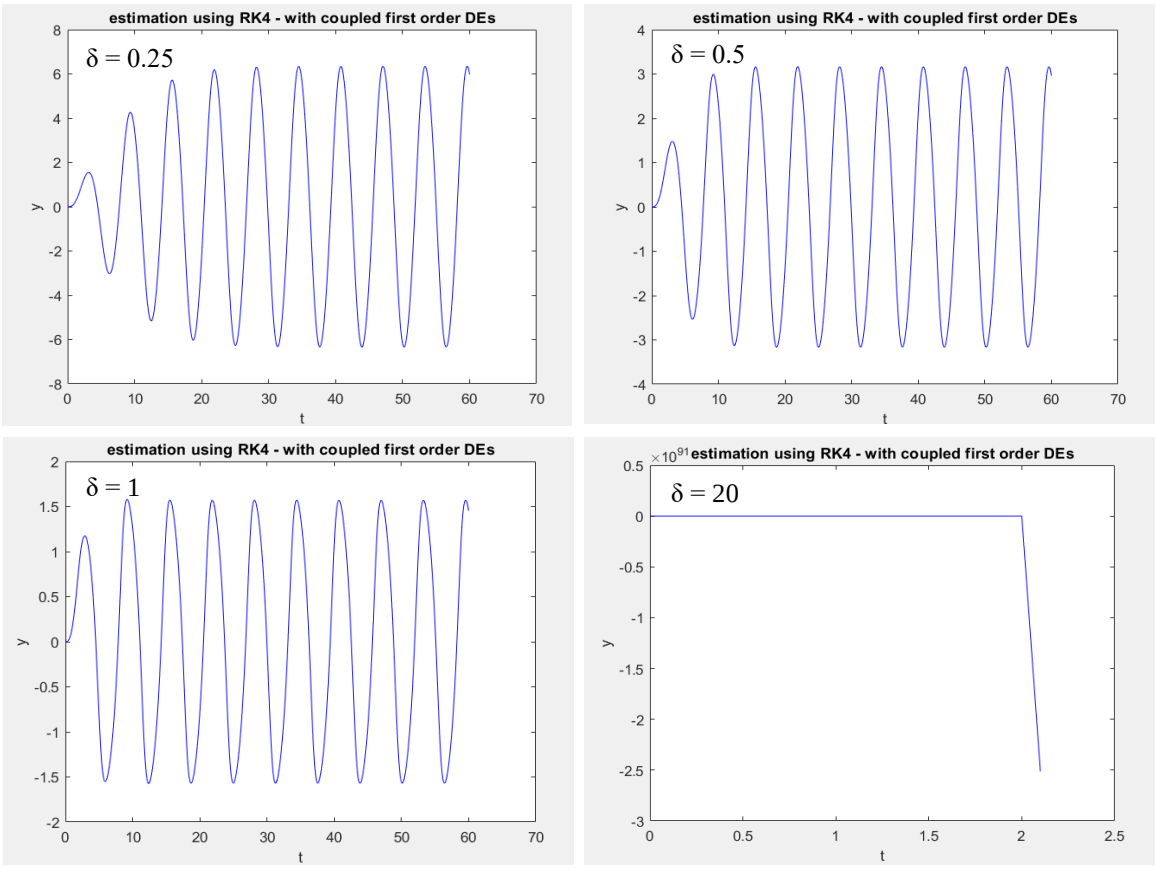

Figure 6: *Four graphs showing how the estimation of the Runge-Kutta method changes with various δ for equations (15) and (16)*

For small δ (0.25, 0.5 and 1), they are all of period 2π. One way to see why this is, is because for small δ, the middle term in the differential equation is approximately zero, so the differential equation becomes:

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

, which resembles the differential equation in questions 5 – 7, but with $\gamma = 0$ and $\omega = 1$, hence why those graphs' behaviours resemble those in question 7.

In the case where δ is large (20), the damping term in the differential equation becomes very large (coefficient of $20^3 = 8000$), hence oscillations disappear, and the graph just rapidly declines in value downwards.

Programs

CODE 1

Code for f(x,y):

```
function [Derivative] = f(x,y)
Derivative=-16*y+15*exp(-x);
end
```

full code to output the table (up to 4 decimal places):

```
function [TABLEAU] = estimation(h)
Y_n=0;
x_n=0;
TABLEAU=zeros(6/h,5);
for n=1:6/h
    Y_n=Y_n + h*f(x_n,Y_n);
    x_n=x_n+h;
    Z_n=exp(-x_n)-exp(-16*x_n);
    E_n=Y_n-Z_n;
    G_n=log(abs(E_n))/x_n;
    TABLEAU(n,1)=x_n;
    TABLEAU(n,2)=Y_n;
    TABLEAU(n,3)=Z_n;
    TABLEAU(n,4)=E_n;
    TABLEAU(n,5)=G_n;
    fprintf('x_n=%.1f, Y_n=%17.4f, y(x_n)=%.4f, E_n=%17.4f, G_n=%.4f\n',x_n, Y_n,
Z_n, E_n, G_n)
end
```

CODE 2

full code to output the table (up to 10 decimal places):

```
function [TABLEAU] = estimation(h)
Y_n=0;
x_n=0;
TABLEAU=zeros(6/h,5);
for n=1:6/h
    Y_n=Y_n + h*f(x_n,Y_n);
    x_n=x_n+h;
    Z_n=exp(-x_n)-exp(-16*x_n);
    E_n=Y_n-Z_n;
    G_n=log(abs(E_n))/x_n;
    TABLEAU(n,1)=x_n;
    TABLEAU(n,2)=Y_n;
    TABLEAU(n,3)=Z_n;
    TABLEAU(n,4)=E_n;
    TABLEAU(n,5)=G_n;
    fprintf('x_n=%.3f, Y_n=%17.10f, y(x_n)=%.7f, E_n=%17.10f, G_n=%.6f\n',x_n,
Y_n, Z_n, E_n, G_n)
end
```

CODE 3

```
function [TABLEAU] = euler(h)
Y_n=0;
x_n=0;
TABLEAU=zeros(4/h,3);
vector_X=zeros(1,1+(4/h));
vector_Y=zeros(1,1+(4/h));
vector_Z=zeros(1,1+(4/h));
for n=1:4/h
    Y_n=Y_n + h*f(x_n,Y_n);
    x_n=x_n+h;
    Z_n=exp(-x_n)-exp(-16*x_n);
    vector_X(n+1)=x_n;
    vector_Y(n+1)=Y_n;
    vector_Z(n+1)=Z_n;
    TABLEAU(n,1)=x_n;
    TABLEAU(n,2)=Y_n;
    TABLEAU(n,3)=Z_n;
    fprintf('x_n=%.2f, Y_n=%17.10f, y(x_n)=%.7f\n',x_n, Y_n, Z_n)
end
plot(vector_X,vector_Y,'b-')
hold on;
plot(vector_X,vector_Z,'r-')
xlabel('x')
ylabel('y')
title('estimation using Euler')
legend('Y_n','y_n','Location','northeast')
```

CODE 4

```
function [RK4_results] = RK4(h)
Y_n=0;
x_n=0;
RK4_results=zeros(4/h,3);
vector_X=zeros(1,1+(4/h));
vector_Y=zeros(1,1+(4/h));
vector_Z=zeros(1,1+(4/h));
for n=1:4/h
    k1=h*f(x_n,Y_n);
    k2=h*f((x_n+(h/2)), (Y_n+((k1)/2)));
    k3=h*f((x_n+(h/2)), (Y_n+((k2)/2)));
    k4=h*f((x_n+h), (Y_n+k3));
    Y_n=Y_n + (k1+2*k2+2*k3+k4)/6;
    x_n=x_n+h;
    Z_n=exp(-x_n)-exp(-16*x_n);
    vector_X(n+1)=x_n;
    vector_Y(n+1)=Y_n;
    vector_Z(n+1)=Z_n;
    RK4_results(n,1)=x_n;
    RK4_results(n,2)=Y_n;
    RK4_results(n,3)=Z_n;
    fprintf('x_n=%.2f, Y_n=%17.10f, y(x_n)=%.7f\n',x_n, Y_n, Z_n)
end
plot(vector_X,vector_Y,'b-')
hold on;
plot(vector_X,vector_Z,'r-')
xlabel('x')
```

```
ylabel('y')
title('estimation using the Runge-Kutta method')
legend('Y_n','y_n','Location','northeast')
```

CODE 5

```
function [EULER] = euler_2(x_max)
vector_h=zeros(1,16);
vector_E=zeros(1,16);
EULER=zeros(16,2);
for k=0:15
    n=2^k;
    h=x_max/n;
    Y_n=0;
    x_n=0;
    for i=1:n
        Y_n=Y_n + h*f(x_n,Y_n);
        x_n=x_n+h;
    end
    Z_n=exp(-0.1)-exp(-16*0.1);
    E_n=Y_n-Z_n;
    EULER(k+1,1)=h;
    EULER(k+1,2)=E_n;
    vector_h(k+1)=log(h);
    vector_E(k+1)=log(abs(E_n));
    fprintf('h=%.16f, E_n=%.16f\n',log(h), log(abs(E_n)))
end
plot(vector_h,vector_E,'b-')
xlabel('x')
ylabel('y')
title('a graph to show how the error changes with step size')
```

CODE 6

```
function [RUNGE_KUTTA] = RK4_2(x_max)
vector_h=zeros(1,16);
vector_E=zeros(1,16);
RUNGE_KUTTA=zeros(16,2);
for k=0:15
    n=2^k;
    h=x_max/n;
    Y_n=0;
    x_n=0;
    for i=1:n
        k1=h*f(x_n,Y_n);
        k2=h*f((x_n+(h/2)), (Y_n+((k1)/2)));
        k3=h*f((x_n+(h/2)), (Y_n+((k2)/2)));
        k4=h*f((x_n+h), (Y_n+k3));
        Y_n=Y_n + (k1+2*k2+2*k3+k4)/6;
        x_n=x_n+h;
    end
    Z_n=exp(-0.1)-exp(-16*0.1);
    E_n=Y_n-Z_n;
    RUNGE_KUTTA(k+1,1)=h;
    RUNGE_KUTTA(k+1,2)=E_n;
    vector_h(k+1)=log(h);
    vector_E(k+1)=log(abs(E_n));
    fprintf('h=%.16f, E_n=%.16f\n',h, E_n)
```

```
end
plot(vector_h,vector_E,'b-')
xlabel('x')
ylabel('y')
title('a graph to show how the error changes with step size')
```

CODE 7

Code for f(x,y):

```
function [derivative_vector] = vector_F(t,Y,v,w)
derivative_vector=[Y(2), (-v*Y(2)-Y(1)+sin(w*t))];
end
```

full code to output the table:

```
function [RK4_output] = vector_RK4(h,t_max,v,w)
RK4_output = zeros(t_max/h,4);
Y_n=zeros(1,2);
t_n=0;
k=(sqrt(4-v^2))/2;
d=(v*w)^2 +(1-w^2)^2;
B=(v*w)/((v*w)^2 + (1-w^2)^2);
A=((v*B)/2 - (w*(1-w^2))/d)/k;
for n=1:t_max/h
    k1=h*vector_F(t_n,Y_n,v,w);
    k2=h*vector_F(t_n+(h/2), Y_n+((k1)/2),v,w);
    k3=h*vector_F(t_n+(h/2), Y_n+((k2)/2),v,w);
    k4=h*vector_F(t_n+h, Y_n+k3,v,w);
    Y_n=Y_n + (k1+2*k2+2*k3+k4)/6;
    t_n=t_n + h;
    Z_n=exp(-(v*t_n)/2)*(A*sin(k*t_n)+B*cos(k*t_n))+(1-w^2)*sin(w*t_n)/d -
(v*w)*cos(w*t_n)/d;
    E_n=Y_n(1)-Z_n;
    RK4_output(n,1)=t_n;
    RK4_output(n,2)=Y_n(1);
    RK4_output(n,3)=Z_n;
    RK4_output(n,4)=E_n;
    fprintf('t_n=%.1f, Y_n=%17.10f, Z_n=%17.10f, E_n=%17.15f\n',t_n, Y_n(1), Z_n,
E_n)
end
```

CODE 8

```
function [RK4_output] = vector_RK4_plot(h,t_max,v,w)
RK4_output = zeros(t_max/h,4);
Y_n=zeros(1,2);
t_n=0;
k=(sqrt(4-v^2))/2;
d=(v*w)^2 +(1-w^2)^2;
B=(v*w)/((v*w)^2 + (1-w^2)^2);
A=((v*B)/2 - (w*(1-w^2))/d)/k;
vector_T=zeros(1,1+(4/h));
vector_Y=zeros(1,1+(4/h));
vector_Z=zeros(1,1+(4/h));
for n=1:t_max/h
    k1=h*vector_F(t_n,Y_n,v,w);
    k2=h*vector_F(t_n+(h/2), Y_n+((k1)/2),v,w);
```

```
    k3=h*vector_F(t_n+(h/2), Y_n+((k2)/2),v,w);
    k4=h*vector_F(t_n+h, Y_n+k3,v,w);
    Y_n=Y_n + (k1+2*k2+2*k3+k4)/6;
    t_n=t_n + h;
    Z_n=exp(-(v*t_n)/2)*(A*sin(k*t_n)+B*cos(k*t_n))+(1-w^2)*sin(w*t_n)/d -
(v*w)*cos(w*t_n)/d;
    E_n=Y_n(1)-Z_n;
    vector_T(n+1)=t_n;
    vector_Y(n+1)=Y_n(1);
    vector_Z(n+1)=Z_n;
    RK4_output(n,1)=t_n;
    RK4_output(n,2)=Y_n(1);
    RK4_output(n,3)=Z_n;
    RK4_output(n,4)=E_n;
    fprintf('t_n=%.1f, Y_n=%17.10f, Z_n=%17.10f, E_n=%17.10f\n',t_n, Y_n(1), Z_n,
E_n)
end
plot(vector_T,vector_Y,'b-')
hold on;
plot(vector_T,vector_Z,'r-')
xlabel('t')
ylabel('y')
title('estimation using RK4 - with coupled first order DEs')
legend('Y_n','y_n','Location','northeast')
```

CODE 9

Code for f(x,y):

```
function [vector_derivative] = F_vector(t,Y,delta)
vector_derivative=[Y(2), (-((delta)^3)*((Y(1))^2)*Y(2)-Y(1)+sin(t))];
end
```

full code to output the table and graph:

```
function [RUNGE_KUTTA_2] = RK4_vector(h,t_max,delta)
RUNGE_KUTTA_2 = zeros(t_max/h,2);
Y_n=zeros(1,2);
t_n=0;
vector_T=zeros(1,1+(t_max/h));
vector_Y=zeros(1,1+(t_max/h));
for n=1:t_max/h
    k1=h*F_vector(t_n,Y_n,delta);
    k2=h*F_vector(t_n+(h/2), Y_n+((k1)/2),delta);
    k3=h*F_vector(t_n+(h/2), Y_n+((k2)/2),delta);
    k4=h*F_vector(t_n+h, Y_n+k3,delta);
    Y_n=Y_n + (k1+2*k2+2*k3+k4)/6;
    t_n=t_n + h;
    vector_T(n+1)=t_n;
    vector_Y(n+1)=Y_n(1);
    RUNGE_KUTTA_2(n,1)=t_n;
    RUNGE_KUTTA_2(n,2)=Y_n(1);
    fprintf('t_n=%.1f, Y_n=%17.10f\n',t_n, Y_n(1))
end
plot(vector_T,vector_Y,'b-')
xlabel('t')
ylabel('y')
title('estimation using RK4 - with coupled first order DEs')
```