1.2

1.2 Ordinary Differential Equations

Comparison of the numerical methods for solving ODEs

Note all code to generate figures will be at the end under the heading of the relevant figure(s)

Programming Task

I defined three functions, each to do one of the numerical integration algorithms `euler_method(x0, xN, N, func, Y0)`, `leapfrog_method(x0, xN, N, func, Y0)`, `rk4_method(x0, xN, N, func, Y0)` and is show on page 15. with the parameters being

- `x0` the starting x position
- `xN` the ending x position
- `N` number of points to be calculated
- `func` The function to be integrated
- `Y0` the value of $y(x)$ at $x = 0$

Stability

Question 1

n	x_n	Y_n	$y(x_n)$	E_n	$ln(\lvert E_n \rvert)$
0	0.0	0.0×10^{0}	0.000000	0.0×10^{0}	-inf
1	0.4	1.2×10^{0}	0.468424	7.316×10^{-1}	-0.312554
2	0.8	-2.231×10^{0}	0.408567	-2.64×10^{0}	0.970703
3	1.2	9.418×10^{0}	0.292964	9.125×10^{0}	2.211058
4	1.6	-3.165×10^{1}	0.200235	-3.185×10^{1}	3.460951
5	2.0	1.112×10^{2}	0.135000	1.11×10^{2}	4.709876
6	2.4	-3.871×10^{2}	0.090650	-3.872×10^{2}	5.958858
7	2.8	1.35×10^{3}	0.060796	1.35×10^{3}	7.207843
8	3.2	-4.707×10^{3}	0.040759	-4.707×10^{3}	8.456826
9	3.6	1.641×10^{4}	0.027323	1.641×10^{4}	9.705809
10	4.0	-5.723×10^{4}	0.018316	-5.723×10^{4}	10.954792
11	4.4	1.995×10^{5}	0.012277	1.995×10^{5}	12.203776
12	4.8	-6.958×10^{5}	0.008230	-6.958×10^{5}	13.452759
13	5.2	2.426×10^{6}	0.005517	2.426×10^{6}	14.701742
14	5.6	-8.459×10^{6}	0.003698	-8.459×10^{6}	15.950726
15	6.0	2.949×10^{7}	0.002479	2.949×10^{7}	17.199709
16	6.4	-1.028×10^{8}	0.001662	-1.028×10^{8}	18.448692
17	6.8	3.586×10^{8}	0.001114	3.586×10^{8}	19.697676
18	7.2	-1.25×10^{9}	0.000747	-1.25×10^{9}	20.946659
19	7.6	4.36×10^{9}	0.000500	4.36×10^{9}	22.195642
20	8.0	-1.52×10^{10}	0.000335	-1.52×10^{10}	23.444626
21	8.4	5.3×10^{10}	0.000225	5.3×10^{10}	24.693609
22	8.8	-1.848×10^{11}	0.000151	-1.848×10^{11}	25.942592
23	9.2	6.444×10^{11}	0.000101	6.444×10^{11}	27.191576
24	9.6	-2.247×10^{12}	0.000068	-2.247×10^{12}	28.440559
25	10.0	7.834×10^{12}	0.000045	7.834×10^{12}	29.689542

Figure 1: Graph tabulating information about Leapfrog method for $h = 0.4$

n	x_n	Y_n	$y(x_n)$	E_n	$ln(\lvert E_n \rvert)$
0	0.0	0.0×10^{0}	0.000000	0.0×10^{0}	-inf
5	1.0	4.321×10^{0}	0.349564	3.972×10^{0}	1.379151
10	2.0	-1.549×10^{2}	0.135000	-1.55×10^{2}	5.043487
15	3.0	6.044×10^{3}	0.049781	6.044×10^{3}	8.706828
20	4.0	-2.357×10^{5}	0.018316	-2.357×10^{5}	12.370170
25	5.0	9.189×10^{6}	0.006738	9.189×10^{6}	16.033511
30	6.0	-3.583×10^{8}	0.002479	-3.583×10^{8}	19.696852
35	7.0	1.397×10^{10}	0.000912	1.397×10^{10}	23.360193
40	8.0	-5.447×10^{11}	0.000335	-5.447×10^{11}	27.023535
45	9.0	2.124×10^{13}	0.000123	2.124×10^{13}	30.686876
50	10.0	-8.282×10^{14}	0.000045	-8.282×10^{14}	34.350217

Figure 2: Graph tabulating information about Leapfrog method for $h = 0.2$

n	x_n	Y_n	$y(x_n)$	E_n	$ln(\|E_n\|)$
0	0.0	0.0×10^0	0.000000	0.0×10^0	-inf
10	1.0	-1.305×10^0	0.349564	-1.654×10^0	0.503497
20	2.0	-8.158×10^1	0.135000	-8.171×10^1	4.403216
30	3.0	-4.038×10^3	0.049781	-4.038×10^3	8.303569
40	4.0	-1.996×10^5	0.018316	-1.996×10^5	12.203922
50	5.0	-9.863×10^6	0.006738	-9.863×10^6	16.104275
60	6.0	-4.874×10^8	0.002479	-4.874×10^8	20.004629
70	7.0	-2.409×10^{10}	0.000912	-2.409×10^{10}	23.904982
80	8.0	-1.19×10^{12}	0.000335	-1.19×10^{12}	27.805335
90	9.0	-5.883×10^{13}	0.000123	-5.883×10^{13}	31.705688
100	10.0	-2.907×10^{15}	0.000045	-2.907×10^{15}	35.606041

Figure 3: Graph tabulating information about Leapfrog method for $h = 0.1$

n	x_n	Y_n	$y(x_n)$	E_n	$ln(\|E_n\|)$
0	0.0	0.0×10^0	0.000000	0.0×10^0	-inf
20	1.0	-1.346×10^{-1}	0.349564	-4.842×10^{-1}	-0.725355
40	2.0	-2.56×10^1	0.135000	-2.574×10^1	3.247896
60	3.0	-1.369×10^3	0.049781	-1.369×10^3	7.221699
80	4.0	-7.28×10^4	0.018316	-7.28×10^4	11.195501
100	5.0	-3.872×10^6	0.006738	-3.872×10^6	15.169303
120	6.0	-2.059×10^8	0.002479	-2.059×10^8	19.143105
140	7.0	-1.095×10^{10}	0.000912	-1.095×10^{10}	23.116908
160	8.0	-5.826×10^{11}	0.000335	-5.826×10^{11}	27.090710
180	9.0	-3.098×10^{13}	0.000123	-3.098×10^{13}	31.064512
200	10.0	-1.648×10^{15}	0.000045	-1.648×10^{15}	35.038314

Figure 4: Graph tabulating information about Leapfrog method for $h = 0.05$

Using the tables above, we can approximate the growth rate by calculating the gradient of $ln(|E_n|)$ against x_n using the `Tabulate_leapfrog(n, table_length)` function where `n` is the number of data points to be generated and `table_length` is the length of the table generated.

```
table1, gr1 = Tabulate_leapfrog(25,25)
table2, gr2 = Tabulate_leapfrog(50,10)
table3, gr3 = Tabulate_leapfrog(100,10)
table4, gr4 = Tabulate_leapfrog(200,10)
print(gr1)
3.1252183037604206
print(gr2)
3.6634518016732462
print(gr3)
3.900282741901613
print(gr4)
3.973741002753291
```

So we can see as h decreases γ increases and seems to tend to 4 (which we will show is the case in Question 2). In terms of the instability, for the smaller values of x_n we see a decrease in stability as h decreases, but for larger values of x_n, we see an increase in instability before it starts to decrease.

Question 2

(i) First rewrite the difference equation as

$$Y_{n+1} + 8hY_n - Y_{n-1} = 6h\left(e^{-h}\right)^n$$

solving the homogeneous equations gives us

$$Y_n = A\left(-4h + \sqrt{16h^2+1}\right)^n + B\left(-4h - \sqrt{16h^2+1}\right)^n$$

considering a particular integral of the form $Y_n = C(e^{-h})^n$ we get

$$Y_n = \frac{3h}{4h - \sinh h}(e^{-h})^n$$

Imposing our initial conditions we get the full solution

$$Y_n = -\frac{3h}{4h-\sinh h}\left[\frac{1}{2}\left(1 + \frac{\cosh h}{\sqrt{16h^2+1}}\right)\left(-4h + \sqrt{16h^2+1}\right)^n + \frac{1}{2}\left(1 - \frac{\cosh h}{\sqrt{16h^2+1}}\right)\left(-4h - \sqrt{16h^2+1}\right)^n - \left(e^{-h}\right)^n\right]$$

(ii) Note that for $h > 0$, $\left|-4h - \sqrt{16h^2+1}\right| = \left|4h + \sqrt{16h^2+1}\right| > 1$ so for fixed h, $\left|\left(-4h - \sqrt{16h^2+1}\right)^n\right| \to \infty$ as $n \to \infty$ and does so at an exponential rate while all other terms tend to 0 as $n \to \infty$ so the term $\left(-4h - \sqrt{16h^2+1}\right)^n$ causes the instability. If we consider $x_n = nh$ and let $n \to \infty$, $h \to 0$ then the term $\left(-4h - \sqrt{16h^2+1}\right)^n = \left(-\frac{4x_n}{n} - \sqrt{16\left(\frac{x_n}{n}\right)^2+1}\right)^n$ is asymptotic to $\left(-1 - \frac{4x_n}{n}\right)^n$, noting that the other terms converge to the analytic solution, we get for large n, where $x_n = nh$

$$|E_n| \approx \frac{1}{2}\left|\frac{3h}{4h-\sinh h}\cdot\left(1 - \frac{\cosh h}{\sqrt{16h^2+1}}\right)\right| e^{4x_n} = D(h)e^{4x_n}$$

(iii) Now if we consider the Taylor expansion of relevant formulas at 0 with respect to h, and using the fact $x_n \equiv nh$ we get

$$\begin{aligned}
\lim_{\substack{h\to 0\\ n\to\infty}} Y_n &= -\frac{3h}{4h-h}\left[\frac{1}{2}\left(1 + \frac{1+O(h^2)}{1+O(h^2)}\right)\left(1 - 4h + O(h^2)\right)^n + \frac{1}{2}\left(1 - \frac{1+O(h^2)}{1+O(h^2)}\right)\left(1 - 4h - O(h^2)\right)^n - e^{-hn}\right] \\
&= -\left[\left(1 - \frac{4x_n}{n} + O(\frac{1}{n^2})\right)^n - e^{-hn}\right] \\
&= -\left[\left(1 - \frac{4x_n}{n}\right)^n + O(\frac{1}{n}) - e^{-x_n}\right] \\
&= e^{-x_n} - e^{-4x_n}
\end{aligned}$$

The point-wise convergence shown above doesn't necessarily show that the instability can be suppressed in any finite number of steps, however consider $x_N = Nh \geq 10$ for N sufficiently large then for all $n \leq N$, there exists n large such that $x_n = nh$ so

$$|E_n| = D(h)e^{4x_n} \leq D(h)e^{40} = |E_N|$$

So it suffices to show that $Y_N \to y(x_N)$ for all $x_N = Nh \geq 10$ finite, which is true by the above, so instability can be suppressed for $x \in [0, 10]$

Accuracy

Question 3

x	Runge-Kutta	Euler	Analytical
0.0	0.000000	0.000000	0.000000
0.4	0.405856	1.200000	0.468424
0.8	0.381797	0.084384	0.408567
1.2	0.285601	0.488564	0.292964
1.6	0.199468	0.068294	0.200235
2.0	0.135877	0.201299	0.135000
2.4	0.091668	0.041623	0.090650
2.8	0.061605	0.083888	0.060796
3.2	0.041338	0.022639	0.040759
3.6	0.027721	0.035331	0.027323
4.0	0.018585	0.011590	0.018316

Figure 5: Y_n using both Runge-Kutta and Euler Method $h = 0.4$

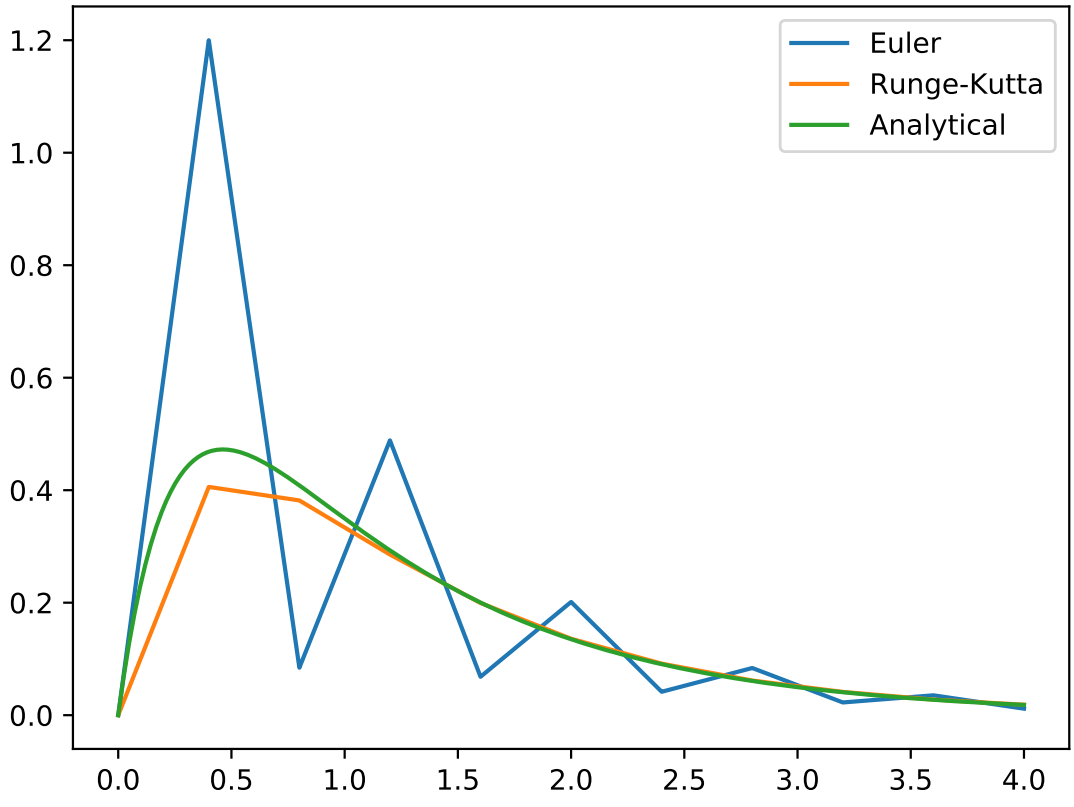

Figure 6: Y_n using both Runge-Kutta and Euler Method $h = 0.4$

Question 4

h	Euler	Leapfrog	Runge-Kutta
0.400000	0.731576	7.315765e-01	-6.256753e-02
0.200000	0.142815	-4.459466e-01	-1.923277e-03
0.100000	0.063716	-1.601633e-01	-8.420115e-05
0.050000	0.030039	-4.479847e-02	-4.413362e-06
0.025000	0.014599	-1.153981e-02	-2.526839e-07
0.012500	0.007198	-2.906985e-03	-1.511655e-08
0.006250	0.003574	-7.281358e-04	-9.243528e-10
0.003125	0.001781	-1.821210e-04	-5.714440e-11
0.001563	0.000889	-4.553569e-05	-3.551992e-12
0.000781	0.000444	-1.138426e-05	-2.214895e-13
0.000391	0.000222	-2.846087e-06	-1.387779e-14
0.000195	0.000111	-7.115230e-07	-9.992007e-16
0.000098	0.000055	-1.778808e-07	0.000000e+00
0.000049	0.000028	-4.447022e-08	-4.440892e-16
0.000024	0.000014	-1.111755e-08	1.221245e-15
0.000012	0.000007	-2.779388e-09	-4.440892e-16

Figure 7: Table of E_n at $x_n = 0.4$ using the three different methods for different values of h

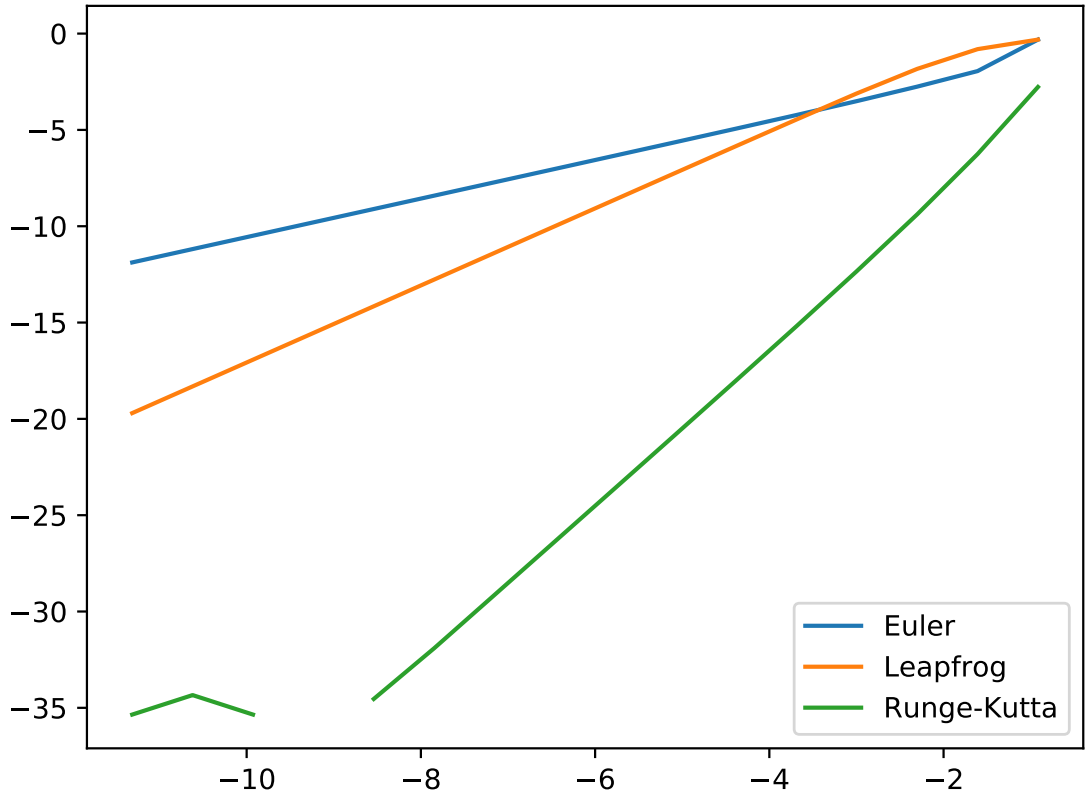

Figure 8: log-log graph of $|E_n|$ against h

We can see from Dahlquist's Equivalence Theorem as stated in *An Introduction to Numerical Analysis by Andre Suli and David Mayers*, page 340 that if an algorithm has truncation error $O(h^k)$ then it has global error $O(h^{k-1})$, and so taking logs we can deduce that $\log|E_n| \sim (k-1)\log h$ as $h \to 0$ which is what we see on the graph above

Numerical solutions of second-order ODEs

Question 5

The general analytic solution to the ODE

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

therefore

$$y \to -\frac{a}{\left(\omega^2-\Omega^2\right)^2+\left(\omega\gamma\right)^2}\left(\omega\gamma\cos\omega t+\left(\omega^2-\Omega^2\right)\sin\omega t\right) = A_s\sin\left(\omega t-\phi_s\right)$$

where

$$A_s = \frac{a}{\sqrt{\left(\omega^2-\Omega^2\right)^2+\left(\omega\gamma\right)^2}}$$
$$\phi_s = \arctan\frac{\omega\gamma}{\Omega^2-\omega^2}$$

Programming Task

The function to perform the RK4 method on the described coupled system is called `coupled_rk4_method(t0, tN, N, gamma, delta, omega, a, w)` and is show on page 18. with the parameters being

- `t0` the starting time
- `tN` the ending time
- `N` number of points to be calculated
- `gamma` γ in the given formula
- `delta` δ in the given formula
- `omega` Ω in the given formula
- `a` a in the given formula
- `w` ω in the given formula

Question 6

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

to get the analytical solution I defined a function `get_y_ana(gamma, w)` as shown on page 19 to return the analytical solution given appropriate γ and ω. I also defined another function `Tabulate_rk4(t0, tN, N, gamma, w, num_output)` as shown on page 19 in order to tabulate all the relevant information given start and end times, number of data points to be found using Runge-Kutta, γ and ω, and the number of rows for the table to have.

n	x_n	Y_n	$y(x_n)$	E_n
0	0.0	0.000000	0.000000	0.0×10^{0}
1	0.4	0.016297	0.016228	6.929×10^{-5}
2	0.8	0.107096	0.107052	4.409×10^{-5}
3	1.2	0.277351	0.277357	-6.187×10^{-6}
4	1.6	0.463176	0.463193	-1.747×10^{-5}
5	2.0	0.570018	0.569982	3.616×10^{-5}
6	2.4	0.525149	0.525017	1.324×10^{-4}
7	2.8	0.319065	0.318849	2.157×10^{-4}
8	3.2	0.016016	0.015786	2.293×10^{-4}
9	3.6	-0.271422	-0.271568	1.462×10^{-4}
10	4.0	-0.431911	-0.431897	-1.339×10^{-5}
11	4.4	-0.405897	-0.405708	-1.889×10^{-4}
12	4.8	-0.213414	-0.213108	-3.066×10^{-4}
13	5.2	0.054085	0.054399	-3.132×10^{-4}
14	5.6	0.274493	0.274696	-2.024×10^{-4}
15	6.0	0.349891	0.349910	-1.878×10^{-5}
16	6.4	0.250400	0.250239	1.609×10^{-4}
17	6.8	0.026938	0.026677	2.613×10^{-4}
18	7.2	-0.212968	-0.213211	2.428×10^{-4}
19	7.6	-0.355165	-0.355284	1.185×10^{-4}
20	8.0	-0.331653	-0.331601	-5.153×10^{-5}
21	8.4	-0.151895	-0.151707	-1.88×10^{-4}
22	8.8	0.101712	0.101941	-2.288×10^{-4}
23	9.2	0.312064	0.312220	-1.565×10^{-4}
24	9.6	0.381580	0.381587	-6.894×10^{-6}
25	10.0	0.277415	0.277266	1.489×10^{-4}

Figure 9: Graph tabulating information about Runge-Kutta 4 method for $h = 0.4$

n	x_n	Y_n	$y(x_n)$	E_n
0	0.0	0.000000	0.000000	0.0×10^{0}
5	1.0	0.184971	0.184971	6.039×10^{-7}
10	2.0	0.569988	0.569982	5.906×10^{-6}
15	3.0	0.173025	0.173011	1.414×10^{-5}
20	4.0	-0.431903	-0.431897	-5.64×10^{-6}
25	5.0	-0.081188	-0.081169	-1.891×10^{-5}
30	6.0	0.349913	0.349910	3.651×10^{-6}
35	7.0	-0.098861	-0.098876	1.451×10^{-5}
40	8.0	-0.331608	-0.331601	-6.936×10^{-6}
45	9.0	0.219683	0.219693	-9.522×10^{-6}
50	10.0	0.277278	0.277266	1.17×10^{-5}

Figure 10: Graph tabulating information about Runge-Kutta 4 method for $h = 0.2$

n	x_n	Y_n	$y(x_n)$	E_n
0	0.0	0.000000	0.000000	0.0×10^{0}
10	1.0	0.184971	0.184971	4.175×10^{-8}
20	2.0	0.569983	0.569982	4.775×10^{-7}
30	3.0	0.173012	0.173011	8.43×10^{-7}
40	4.0	-0.431898	-0.431897	-4.909×10^{-7}
50	5.0	-0.081170	-0.081169	-1.114×10^{-6}
60	6.0	0.349910	0.349910	3.611×10^{-7}
70	7.0	-0.098875	-0.098876	8.193×10^{-7}
80	8.0	-0.331602	-0.331601	-5.272×10^{-7}
90	9.0	0.219692	0.219693	-4.801×10^{-7}
100	10.0	0.277267	0.277266	7.8×10^{-7}

Figure 11: Graph tabulating information about Runge-Kutta 4 method for $h = 0.1$

We can see that within each run, the global error remains relatively constant, but as we decrease h, the size of the global error decreases.

Question 7

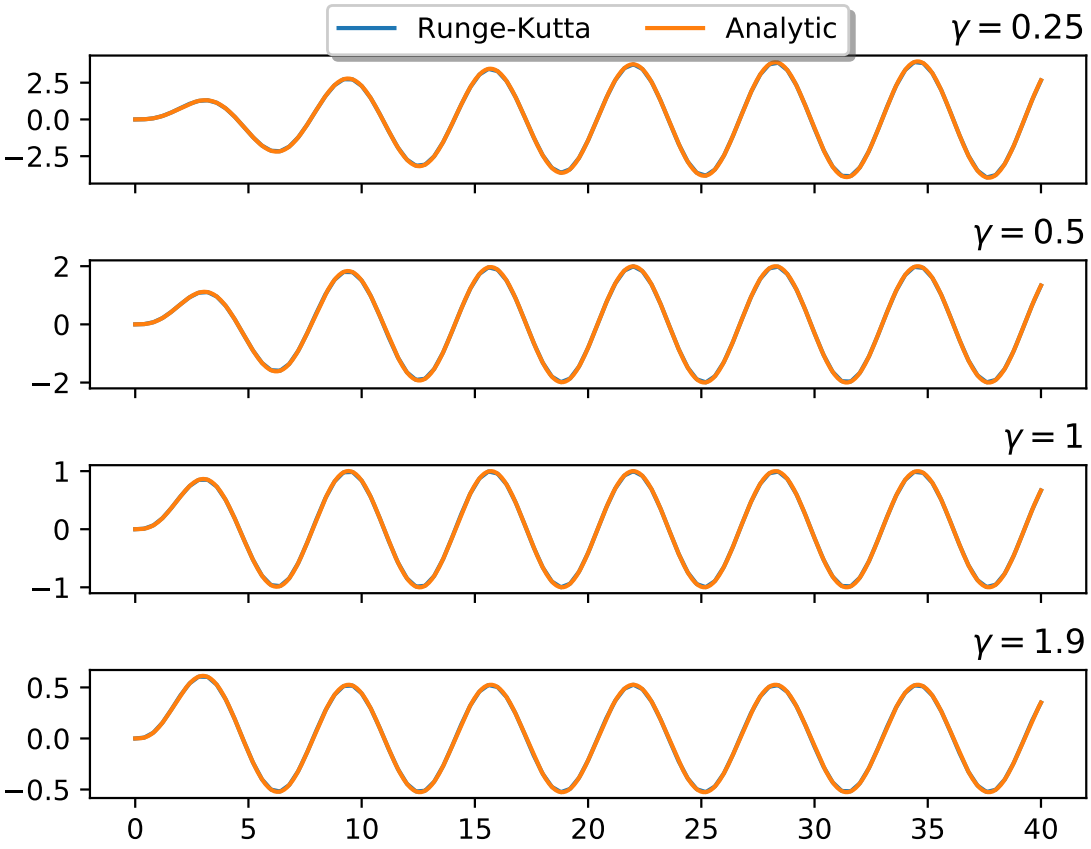

Figure 12: Runge-Kutta using $h = 0.4$ where $\omega = 1$

In the case of $\omega = 1$ we can see that we have a general analytical solution

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

so we can see that for all values of γ that the solution will tend to $\frac{1}{\gamma}\cos(t)$, the functions differ at the beginning due to a transient term but this disappears as $t \to \infty$. Physically This is due to the fact that at the beginning, energy is provided to the system, then as we get to later times, the constant driving force provides into the system the same amount of energy which is lost due to damping.

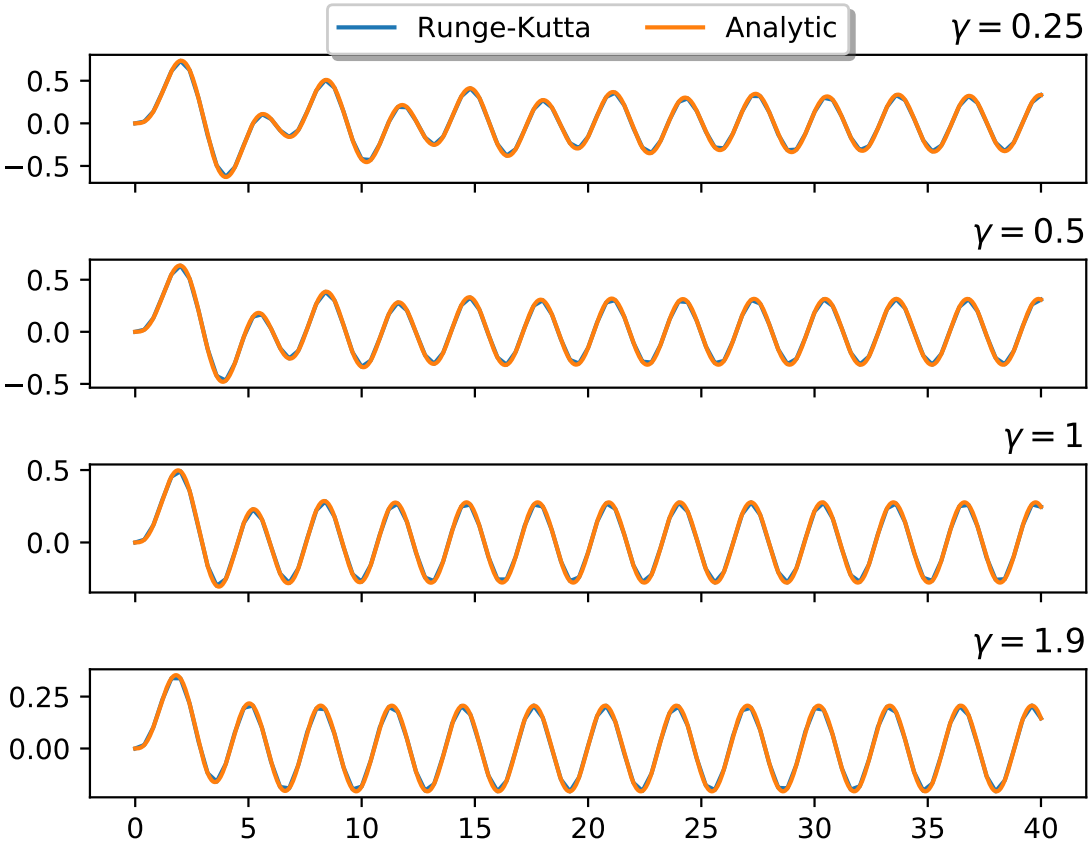

Figure 13: Runge-Kutta using $h = 0.4$ where $\omega = 2$

In the case of $\omega = 2$ we can see that we have a general analytical solution

$$y = \frac{1}{9-4\gamma^2}\left[e^{-\frac{\gamma t}{2}}\left(2\gamma\cos\left(\frac{\sqrt{4-\gamma^2}}{2}t\right) + \frac{2\left(\gamma^2+6\right)}{\sqrt{4-\gamma^2}}\sin\left(\frac{\sqrt{4-\gamma^2}}{2}t\right)\right) - 3\sin 2t - 2\gamma\cos 2t\right]$$

so we can see that for all values of γ that the solution will tend to $-\frac{1}{9-4\gamma^2}\left(3\sin 2t + 2\gamma\cos 2t\right)$, the functions differ at the beginning due to a transient term $\frac{1}{\gamma}e^{-\frac{\gamma t}{2}}\cos\left(\frac{\sqrt{4-\gamma^2}}{2}t\right)$ but this disappears as $t \to \infty$. Physically This is due to the fact that at the beginning, energy is provided to the system, then as we get to later times, the constant driving force provides into the system the same amount of energy which is lost due to damping.
There are two main differences between the cases $\omega = 1$ and $\omega = 2$. The first is the rate of oscillation of the steady state solution, this is different because the driving force oscillates at a different rate for the two cases. The second difference being the max amplitude of the steady state solution, this is different as larger values of ω mean there is a larger damping on the system, so more energy is lost via the damping, so the maximum amplitude would be lower.

Question 8

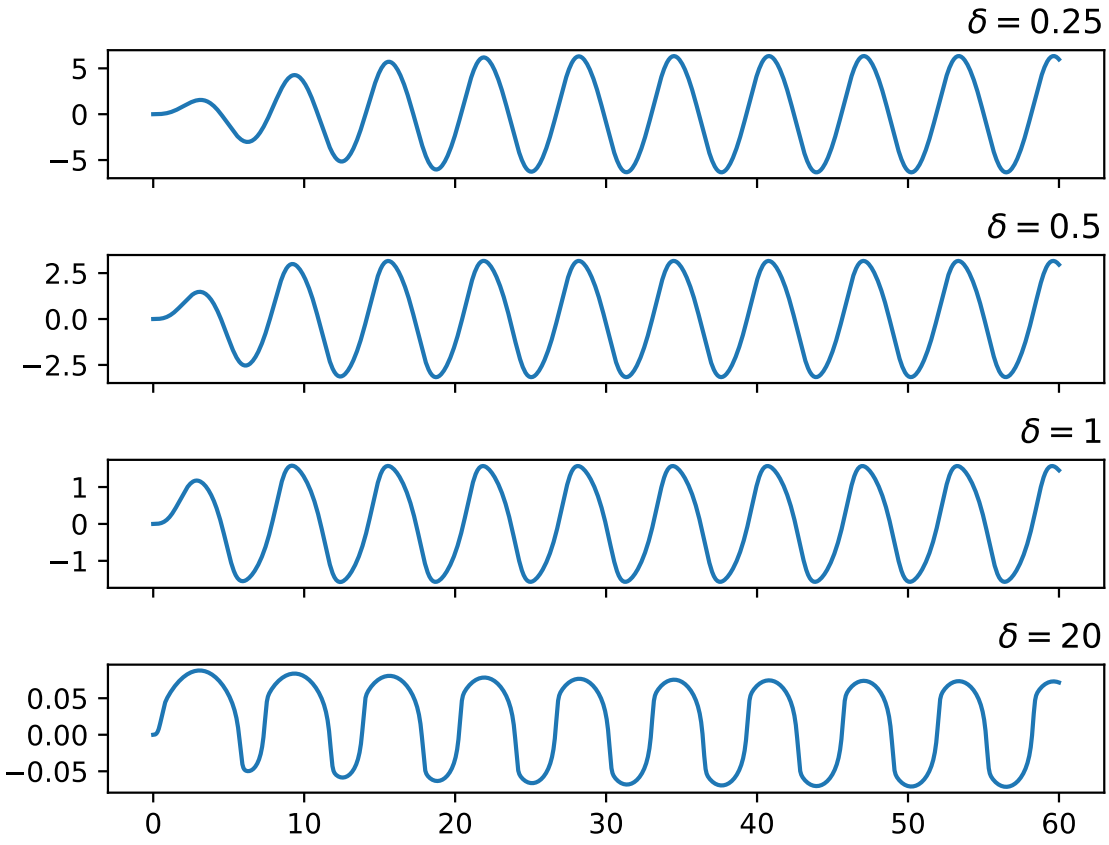

Figure 14: Runge-Kutta for non-linear damping

We expect the solutions to be oscillatory in nature and be bounded, which we can see in the plots above, so the values of h used are satisfactory.

For all the plots in this question (shown in figure 14), we can see that they all have the same period, however as δ increases, the amplitude of the solution decreases, and the transients at the beginning become comparatively larger than the steady state solution.

consider $Y = \delta y$ then the ODE transforms into

$$\frac{d^2Y}{dt^2} + \delta \frac{d}{dt}\left(\frac{1}{3}Y^3\right) + Y = \delta \sin t$$

and let $Y = \sum_{n=-1}^{\infty} \delta^{n+1} y_n(t)$ and consider $y_{-1}(t) = A\cos t$ then if we consider δ small enough such that all other terms are negligible then we get

$$A^3\left(-\frac{1}{4}\sin t - \frac{1}{4}\sin 3t\right) = \sin t$$

so using $A = -\sqrt[3]{4}$ gives us the best fit for this, then if we assume δ slightly larger but ignore all terms of order δ^2 or larger and using what we have for y_{-1} we get

$$\frac{d^2y_0}{dt^2} + \delta \frac{d}{dt}\left(y_{-1}^2 y_0\right) + y_0 = -\sin 3t$$

and using similar reasoning as above we get

$$B = -C = -\frac{1}{8}$$

So putting this all together we get

$$Y \approx -\sqrt[3]{4}\cos t - \delta\frac{1}{8}\left(\sin t - \sin 3t\right)$$

therefore

$$y \approx -\frac{\sqrt[3]{4}}{\delta}\cos t - \frac{1}{8}\left(\sin t - \sin 3t\right)$$

For large δ we can say

$$\frac{d}{dt}\left(\frac{1}{3}Y^3\right) = \sin t$$

solving this we get

$$Y = -\sqrt[3]{3\cos t} \implies y = -\frac{\sqrt[3]{3\cos t}}{\delta}$$

So we can see that for large δ the change in velocity of the particles is much greater than for small δ. We can also see via comparison to question 7 that although the steady state solutions oscillate at the same frequency, that changing the coefficient of linear damping impacts the steady state amplitude less than cubic dampening (from $\gamma = 0.25$ to $\gamma = 1$ the steady state amplitude decreased by a factor of approximately 2.5 whereas for $\delta = 0.25$ to $\delta = 1$ the damping decreased by approximately a factor of 5).

Programs

Programming Task

```
def euler_method(x0, xN, N, func, Y0):
    h = (xN-x0)/N
    Y = [Y0]
    for n in range(1, N+1):
        Yn_1 = Y[-1]
        xn_1 = x0 + (n-1)*h
        Yn = Yn_1 + h * func(xn_1, Yn_1)
        Y.append(Yn)
    return Y

def leapfrog_method(x0, xN, N, func, Y0):
    h = (xN-x0)/N
    Y = [Y0, Y0 + h*func(x0, Y0)]
    for n in range(2, N+1):
        Yn_1 = Y[-1]
        Yn_2 = Y[-2]
        xn_1 = x0 + (n-1)*h
        Y.append(Yn_2+2*h*func(xn_1, Yn_1))
    return Y

def rk4_method(x0, xN, N, func, Y0):
    h = (xN-x0)/N
    Y = [Y0]
    for n in range(1, N+1):
        Yn_1 = Y[-1]
        xn_1 = x0 + (n-1)*h
        k1 = h*func(xn_1, Yn_1)
        k2 = h*func(xn_1+0.5*h, Yn_1+0.5*k1)
        k3 = h*func(xn_1+0.5*h, Yn_1+0.5*k2)
        k4 = h*func(xn_1+h, Yn_1+k3)
        Yn = Yn_1+ 1/6*(k1+2*k2+2*k3+k4)
        Y.append(Yn)
    return Y
```

Figures 1,2,3, and 4

```
def x_axis_gen(x0, xN, n):
    h = (xN-x0)/n
    X = [x0]
    for i in range(1, n+1):
        Xn = X[-1]
        X.append(Xn+h)
    return X

def latex_format(num):
    parts = "{0:.3e}".format(num).split("e")
    if len(parts) == 1:
        part = parts[0]
        part = part.replace(r"inf", r"\infty")
        return "$"+part+"$"
    base = parts[0]
```

```
    exp = parts[1]
    latex_str = r"${0} \times 10^{{{1}}}$".format(str(float(base)), str(int(exp)))
    return latex_str

def Tabulate_leapfrog(n, table_length):
    Yl_values = leapfrog_method(0, 10, n, f, 0)
    X_values = x_axis_gen(0, 10, n)
    tick = round(n/table_length)

    ln_E_n = []

    table = {"$n$":[],
             "$x_n$":[],
             "$Y_n$":[],
             "$y(x_n)$":[],
             "$E_n$":[],
             "$ln(|E_n|)$":[]}
    for i in range(len(X_values)):
        if i%tick!=0:
            continue
        x_n = X_values[i]
        Y_n = Yl_values[i]
        y_n = y_ana(x_n)
        E_n = Y_n - y_n
        ln_mag_E_n = log(abs(E_n))
        table["$n$"].append(i)
        table["$x_n$"].append(x_n)
        table["$Y_n$"].append(latex_format(Y_n))
        table["$y(x_n)$"].append(y_n)
        table["$E_n$"].append(latex_format(E_n))
        table["$ln(|E_n|)$"].append(ln_mag_E_n)
        ln_E_n.append(ln_mag_E_n)

    ln_E_N = ln_E_n[-1]
    ln_E_0 = ln_E_n[1]
    x_N = float(table["$x_n$"][-1])
    x_0 = float(table["$x_n$"][1])
    growth_rate = (ln_E_N-ln_E_0)/(x_N-x_0)
    return table, growth_rate


# can see E against ln_mag_E has gradient approx 3

table1, gr1 = Tabulate_leapfrog(25,25)
table2, gr2 = Tabulate_leapfrog(50,10)
table3, gr3 = Tabulate_leapfrog(100,10)
table4, gr4 = Tabulate_leapfrog(200,10)

def draw_table(table, index, file_name, column_format):
    template = r'''\documentclass[preview]{{standalone}}
    \usepackage{{booktabs}}
    \begin{{document}}
    {}
```

```
    \end{{document}}
    '''
    dataframe = pd.DataFrame.from_dict(table).set_index(index)
    with open(file_name+".tex", 'w') as file:
        file.write(template.format(dataframe.to_latex(
                escape=False,
                column_format = column_format)))



draw_table(table1, "$n$", "table1", "|c||c|r|c|r|r|")
draw_table(table2, "$n$", "table2", "|c||c|r|c|r|r|")
draw_table(table3, "$n$", "table3", "|c||c|r|c|r|r|")
draw_table(table4, "$n$", "table4", "|c||c|r|c|r|r|")
```

Figures 5 and 6

```
Yr_values = rk4_method(0, 4, 10, f, 0)
Ye_values = euler_method(0, 4, 10, f, 0)
X_values = x_axis_gen(0, 4, 10)
Xa_values = x_axis_gen(0, 4, 1000)
Ya_values = [y_ana(x) for x in Xa_values]

table5 = {"$x$": X_values,
        "Runge-Kutta" : Yr_values,
        "Euler": Ye_values,
        "Analytical": [Ya_values[i*100] for i in range(0,11)]}

draw_table(table5, "$x$", "table5", "|c||c|c|c|")

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot(X_values, Ye_values)
ax.plot(X_values, Yr_values)
ax.plot(Xa_values, Ya_values)
ax.legend(['Euler', 'Runge-Kutta', 'Analytical'], loc='upper right')

plt.show()

fig.savefig("Q3.pdf")
```

Figures 7 and 8

```
y_ana_04 = y_ana(0.4)
euler = []
leapfrog = []
rk4 = []
h_vals = []
for k in range(0,16):
    n = 2**k
    h_vals.append(0.4/n)
    Ye_n = euler_method(0,0.4,n,f,0)[n]
    Yl_n = leapfrog_method(0,0.4,n,f,0)[n]
```

```
    Yr_n = rk4_method(0,0.4,n,f,0)[n]
    euler.append(Ye_n-y_ana_04)
    leapfrog.append(Yl_n-y_ana_04)
    rk4.append(Yr_n-y_ana_04)

table6 = {"$h$" : h_vals,
          "Euler" : euler,
          "Leapfrog" : leapfrog,
          "Runge-Kutta" : rk4}

draw_table(table6, "$h$", "table6", "|c||c|c|c|")

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot([log(abs(x)) for x in h_vals], [log(abs(x)) for x in euler])
ax.plot([log(abs(x)) for x in h_vals], [log(abs(x)) for x in leapfrog])
ax.plot([log(abs(x)) for x in h_vals], [log(abs(x)) for x in rk4])
ax.legend(['Euler', 'Leapfrog', 'Runge-Kutta'], loc='lower right')

plt.show()

fig.savefig("Q4.pdf")
```

Programming Task

```
def coupled_rk4_method(t0, tN, N, gamma, delta, omega, a, w):
    f1 = lambda t, y1, y2: y2
    f2 = lambda t, y1, y2: -gamma*y2 - (delta**3)*(y1**2)*y2 - (omega**2)*y1+a*sin(w*t)
    h = (tN-t0)/N
    Y = [[0,0]]

    for n in range(1,N+1):
        tn_1 = t0 +(n-1)*h
        Y1n_1= Y[-1][0]
        Y2n_1= Y[-1][1]

        k11 = h*f1(tn_1, Y1n_1, Y2n_1)
        k12 = h*f2(tn_1, Y1n_1, Y2n_1)

        k21 = h*f1(tn_1+0.5*h, Y1n_1+0.5*k11, Y2n_1+0.5*k12)
        k22 = h*f2(tn_1+0.5*h, Y1n_1+0.5*k11, Y2n_1+0.5*k12)

        k31 = h*f1(tn_1+0.5*h, Y1n_1+0.5*k21, Y2n_1+0.5*k22)
        k32 = h*f2(tn_1+0.5*h, Y1n_1+0.5*k21, Y2n_1+0.5*k22)

        k41 = h*f1(tn_1+h, Y1n_1+k31, Y2n_1+k32)
        k42 = h*f2(tn_1+h, Y1n_1+k31, Y2n_1+k32)

        Y1n = Y1n_1 + 1/6*(k11+2*k21+2*k31+k41)
        Y2n = Y2n_1 + 1/6*(k12+2*k22+2*k32+k42)

        Y.append([Y1n, Y2n])
```

```
    soln = [y[0] for y in Y]
    return soln
```

Figures 9, 10, and 11

```
def get_y_ana(gamma, w):
    A = w*gamma/((1-w**2)**2+(gamma*w)**2)
    B = w*(gamma**2 - 2*(1-w**2))/(((1-w**2)**2 +(gamma*w)**2)*(4-gamma**2)**0.5)
    C = (1-w**2)/((1-w**2)**2 +(gamma*w)**2)
    D = -(gamma*w)/((1-w**2)**2 +(gamma*w)**2)
    k = ((4-gamma**2)**0.5)/2

    y_ana = lambda t: A*(e**(-gamma*t/2))*cos(k*t) + B*(e**(-gamma*t/2))*sin(k*t) + C*sin(w*t)

    return y_ana

def Tabulate_rk4(t0, tN, N, gamma, w, num_output):
    Yr_values = coupled_rk4_method(t0, tN, N, gamma, 0, 1, 1, w)
    X_values = x_axis_gen(t0, tN, N)
    tick = round(N/num_output)

    y_ana = get_y_ana(gamma, w)

    table = {"$n$":[],
             "$x_n$":[],
             "$Y_n$":[],
             "$y(x_n)$":[],
             "$E_n$":[]}
    for i in range(N+1):
        if i % tick != 0:
            continue
        x_n = X_values[i]
        Y_n = Yr_values[i]
        y_n = y_ana(x_n)
        E_n = Y_n - y_n
        table["$n$"].append(i)
        table["$x_n$"].append(x_n)
        table["$Y_n$"].append(Y_n)
        table["$y(x_n)$"].append(y_n)
        table["$E_n$"].append(latex_format(E_n))
    return table

table7 = Tabulate_rk4(0, 10, 25, 1, 3**(1/2), 25)
table8 = Tabulate_rk4(0, 10, 50, 1, 3**(1/2), 10)
table9 = Tabulate_rk4(0, 10, 100, 1, 3**(1/2), 10)

draw_table(table7, "$n$", "table7", "|c||c|r|c|r|")
draw_table(table8, "$n$", "table8", "|c||c|r|c|r|")
draw_table(table9, "$n$", "table9", "|c||c|r|c|r|")
```

Figures 12 and 13

```
X_ana = x_axis_gen(0, 40, 4000)
```

```
X_40 = x_axis_gen(0, 40, 100)
Y_1 = coupled_rk4_method(0, 40, 100, 0.25, 0, 1, 1, 1)
y_ana_1 = get_y_ana(0.25, 1)
Y_ana_1 = [y_ana_1(x) for x in X_ana]


Y_2 = coupled_rk4_method(0, 40, 100, 0.5, 0, 1, 1, 1)
y_ana_2 = get_y_ana(0.5 , 1)
Y_ana_2 = [y_ana_2(x) for x in X_ana]


Y_3 = coupled_rk4_method(0, 40, 100, 1, 0, 1, 1, 1)
y_ana_3 = get_y_ana(1   , 1)
Y_ana_3 = [y_ana_3(x) for x in X_ana]


Y_4 = coupled_rk4_method(0, 40, 100, 1.9, 0, 1, 1, 1)
y_ana_4 = get_y_ana(1.9 , 1)
Y_ana_4 = [y_ana_4(x) for x in X_ana]

fig, ax = plt.subplots(4, sharex=True)

ax[0].plot(X_40, Y_1)
ax[0].plot(X_ana, Y_ana_1)
ax[0].legend(['Runge-Kutta', 'Analytic'], loc='upper center',
              bbox_to_anchor=(0.5, 1.5),
              ncol = 2, fancybox=True, shadow=True)
ax[0].set_title(r"$\gamma=0.25$", loc="right")

ax[1].plot(X_40, Y_2)
ax[1].plot(X_ana, Y_ana_2)
ax[1].set_title(r"$\gamma=0.5$", loc="right")

ax[2].plot(X_40, Y_3)
ax[2].plot(X_ana, Y_ana_3)
ax[2].set_title(r"$\gamma=1$", loc="right")

ax[3].plot(X_40, Y_4)
ax[3].plot(X_ana, Y_ana_4)
ax[3].set_title(r"$\gamma=1.9$", loc="right")

fig.subplots_adjust(hspace=0.6)

plt.show()

fig.savefig("Q7a.pdf")


Y_5 = coupled_rk4_method(0, 40, 100, 0.25, 0, 1, 1, 2)
y_ana_5 = get_y_ana(0.25, 2)
Y_ana_5 = [y_ana_5(x) for x in X_ana]

Y_6 = coupled_rk4_method(0, 40, 100, 0.5, 0, 1, 1, 2)
```

```
y_ana_6 = get_y_ana(0.5 , 2)
Y_ana_6 = [y_ana_6(x) for x in X_ana]

Y_7 = coupled_rk4_method(0, 40, 100, 1, 0, 1, 1, 2)
y_ana_7 = get_y_ana(1   , 2)
Y_ana_7 = [y_ana_7(x) for x in X_ana]

Y_8 = coupled_rk4_method(0, 40, 100, 1.9, 0, 1, 1, 2)
y_ana_8 = get_y_ana(1.9 , 2)
Y_ana_8 = [y_ana_8(x) for x in X_ana]

fig, ax = plt.subplots(4, sharex=True)

ax[0].plot(X_40, Y_5)
ax[0].plot(X_ana, Y_ana_5)
ax[0].legend(['Runge-Kutta', 'Analytic'], loc='upper center',
             bbox_to_anchor=(0.5, 1.5),
             ncol = 2, fancybox=True, shadow=True)
ax[0].set_title(r"$\gamma=0.25$", loc="right")

ax[1].plot(X_40, Y_6)
ax[1].plot(X_ana, Y_ana_6)
ax[1].set_title(r"$\gamma=0.5$", loc="right")

ax[2].plot(X_40, Y_7)
ax[2].plot(X_ana, Y_ana_7)
ax[2].set_title(r"$\gamma=1$", loc="right")

ax[3].plot(X_40, Y_8)
ax[3].plot(X_ana, Y_ana_8)
ax[3].set_title(r"$\gamma=1.9$", loc="right")

fig.subplots_adjust(hspace=0.6)

plt.show()

fig.savefig("Q7b.pdf")
```

Figure 14

```
X_60 = x_axis_gen(0, 60, 1000)
Y_9 = coupled_rk4_method(0,60,1000,0,0.25,1,1,1)

Y_10 = coupled_rk4_method(0,60,1000,0,0.5,1,1,1)

Y_11 = coupled_rk4_method(0,60,1000,0,1,1,1,1)


X_60_1 = x_axis_gen(0, 60, 30000)
Y_12 = coupled_rk4_method(0,60,30000,0,20,1,1,1)

fig, ax = plt.subplots(4, sharex=True)

ax[0].plot(X_60, Y_9)
```

```
ax[0].set_title(r"$\delta=0.25$", loc="right")

ax[1].plot(X_60, Y_10)
ax[1].set_title(r"$\delta=0.5$", loc="right")

ax[2].plot(X_60, Y_11)
ax[2].set_title(r"$\delta=1$", loc="right")

ax[3].plot(X_60_1, Y_12)
ax[3].set_title(r"$\delta=20$", loc="right")

fig.subplots_adjust(hspace=0.6)

plt.show()

fig.savefig("Q8.pdf")
```