1.2

1.2 Ordinary Differential Equations

The Euler, AB2 and RK4 methods on (6) are implemented in the programming class Section2.

Question 1

The results are computed in the class Q1. The table for $h = 0.5$ is shown below

```
x_n            Y_n                            y_e(x_n)                       E_n
0              0                              0                              0
0.5            3.0                            0.34956380228270817            2.650436197717292
1.0            -14.84454251472851             0.13499982060871019            -14.97954233533722
1.5            80.27990218645013              0.049780924155510616           80.23012126229462
2.0            -431.06755707890716            0.018315526353559458           -431.0858726052607
2.5            2315.9053295398835             0.006737944937931844           2315.8985915949456
3.0            -12441.658914554071            0.002478752138915013           -12441.66139330621
```

Table 1.1: Y_n, $y_e(x_n)$ and E_n against x_n for $h = 0.5$

To calculate γ, we attempt to find a best fit $y = Ae^{\gamma x}$ to the data excluding $(0, 0)$. We use the least-squares method for the linear relation $\ln y = \ln A + \gamma x$. This gives $\gamma = \frac{n\sum_{i=1}^{n} x_i \ln y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} \ln y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$, which is computed in Q1 to be 3.376.

The tables for $h = 0.375, 0.25, 0.125, 0.1, 0.05$ are shown below. For the last 4 values of h we only print the values when x is a multiple of 0.5.

```
h = 0.375
x_n            Y_n                            y_e(x_n)                       E_n
0              0                              0                              0
0.375          2.25                           0.42257948437315074            1.8274205156268493
0.75           -7.405762884499076             0.22065140797176347            -7.626414292470839
1.125          29.516822014414075             0.10527581475777766            29.411546199656296
1.5            -114.31282042446857            0.049780924155510616           -114.36260134862408
1.875          444.4195617353706              0.023517439953688605           444.3960442954169
2.25           -1726.9143347701497            0.011108981308262562           -1726.925443751458
2.625          6710.840549697807              0.005247517640925342           6710.835302180166
3.0            -26078.308213344048            0.002478752138915013           -26078.310692096187
```

```
γ = 3.6426090925760906

h = 0.25
x_n           Y_n                           y_e(x_n)                      E_n
0             0                             0                             0
0.5           -2.3853060156465746           0.34956380228270817           -2.7348698179292827
1.0           -15.446058294480025           0.13499982060871019           -15.581058115088736
1.5           -90.70830078055474            0.049780924155510616          -90.75808170471025
2.0           -528.9572466716199            0.018315526353559458          -528.9755621979735
2.5           -3083.0916488474104           0.006737944937931844          -3083.0983867923483
3.0           -17969.61342041709            0.002478752138915013          -17969.61589916923
γ = 3.5596822365582304

h = 0.125
x_n           Y_n                           y_e(x_n)                      E_n
0             0                             0                             0
0.5           0.015928185895433877          0.34956380228270817           -0.3336356163872743
1.0           -0.17814469643645559          0.13499982060871019           -0.31314451704516577
1.5           -0.2620359033388657           0.049780924155510616          -0.31181682749437634
2.0           -0.29367874011604866          0.018315526353559458          -0.31199426646960815
2.5           -0.30536830088063827          0.006737944937931844          -0.3121062458185701
3.0           -0.3096717006906251           0.002478752138915013          -0.3121504528295401
γ = -0.1277415920845518

h = 0.1
x_n           Y_n                           y_e(x_n)                      E_n
0             0                             0                             0
0.5           0.3909278741009339            0.34956380228270817           0.04136407181822571
1.0           0.12225301554451082           0.13499982060871019           -0.012746805064199368
1.5           0.05277916305875584           0.049780924155510616          0.002998238903245222
2.0           0.0178194402446741            0.018315526353559458          -4.960861088853588E-4
2.5           0.006916897269646114          0.006737944937931844          1.7895233171426935E-4
3.0           0.0024644935614499593         0.002478752138915013          -1.4258577465053705E-5
γ = -3.237937854371143

h = 0.05
x_n           Y_n                           y_e(x_n)                      E_n
0             0                             0                             0
0.5           0.3460754970275599            0.34956380228270817           -0.0034883052551482607
1.0           0.13499020026284192           0.13499982060871019           -9.620345868266433E-6
1.5           0.049847611921875756          0.049780924155510616          6.668776636514079E-5
2.0           0.01834249126873627           0.018315526353559458          2.6964915176810184E-5
2.5           0.00674793729921141           0.006737944937931844          9.992361279565626E-6
3.0           0.002482430135000681          0.002478752138915013          3.6779960856680484E-6
γ = -6.466082306416815
```

Table 1.2: $\mathrm{Y_n}$, $y_e(x_n)$ and E_n against x_n for $h = 0.375, 0.25, 0.125, 0.1, 0.05$

From the data, it appears that reducing h from 0.5 to 0.375 causes the growth rate to increase (thus more unstable). Further reducing h causes the growth rate to decrease. Reducing h beyond 0.125 gives negative growth rates (and thus stability).

Question 2

(i) We rewrite the equation into $Y_{n+1} + (12h - 1)Y_n - 4hY_{n-1} = 3h(e^{-2h})^n(3 - e^{2h})$.

The complementary solutions are $r_\pm^n$ where $r_\pm := \frac{1-12h\pm\sqrt{(12h-1)^2+4\cdot 4h}}{2} = \frac{1-12h\pm\sqrt{1-8h+144h^2}}{2}$.

For the particular solution, we try $Y_n = C(e^{-2h})^n$. This gives

$Ce^{-2h} + (12h - 1)C - 4hCe^{2h} = 3h(3 - e^{2h})$, so $C = \frac{3h(3-e^{2h})}{e^{-2h}+12h-1-4he^{2h}}$.

$\therefore Y_n = A\left(\frac{1-12h+\sqrt{1-8h+144h^2}}{2}\right)^n + B\left(\frac{1-12h-\sqrt{1-8h+144h^2}}{2}\right)^n + \frac{3h(3-e^{2h})}{e^{-2h}+12h-1-4he^{2h}}(e^{-2h})^n$,

where initial data gives $\begin{cases} A + B = -C \\ r_+A + r_-B = 6h - Ce^{-2h} \end{cases} \Rightarrow \begin{cases} A = \frac{-Cr_- - 6h + Ce^{-2h}}{-\sqrt{1-8h+144h^2}} \\ B = \frac{Cr_+ + 6h - Ce^{-2h}}{-\sqrt{1-8h+144h^2}} \end{cases}$.

(ii) Instability occurs when $|E_n| \to \infty$. As $y_e(x)$ is bounded, we must have $Y_n \to \infty$. This requires $|r_+|$ or $|r_-| > 1$, which solves to $h > \frac{1}{8}$.

In this case, the growth rate is $\max(|r_+|, |r_-|) = -r_- = \frac{-1+12h+\sqrt{1-8h+144h^2}}{2}$.

(iii) For $h \to 0$,

$r_+ \to 1$ and $r_- \to 0$;

$r_+^n = \left(\frac{1-12h+1-4h+O(h^2)}{2}\right)^{\frac{x_n}{h}} = (1-8h)^{\frac{x_n}{h}} \to e^{-8x_n}$, $r_-^n \to 0$ and $(e^{-2h})^n = e^{-2x_n}$;

$\frac{e^{-2h}+12h-1-4he^{2h}}{3h(3-e^{2h})} = \frac{1}{3(3-e^{2h})}\left(\frac{e^{-2h}-1}{h} + 12 - 4e^{2h}\right) \to \frac{1}{3(2)}(-2 + 12 - 4) = 1$, so $C \to 1$;

$A \to \frac{-1\cdot 0 - 6\cdot 0 + 1\cdot 1}{-\sqrt{1-8\cdot 0+144\cdot 0^2}} = -1$; and

$B \to \frac{1\cdot 1 - 6\cdot 0 - 1\cdot 1}{-\sqrt{1-8\cdot 0+144\cdot 0^2}} = 0$.

$\therefore Y_n \to -1 \cdot e^{-8x_n} + 0 \cdot 0 + 1 \cdot e^{-2x_n} = e^{-2x_n} - e^{-8x_n} = y_e(x_n)$ as required.

The conclusions in (ii) will not be altered if a more accurate method is used in the first step, because as seen in (ii), the instability ultimately depends only on the (roots of the) recurrence relation arisen solely from the AB2 method, not the initial data Y_0 and Y_1.

Question 3

The class Q3 computes the required data. The results are shown below.

```
x_n    By Euler              By AB2                By RK4                Exact
0.0    0.0                   0.0                   0.0                   0.0
0.08   0.48                  0.48                  0.3241070783102275    0.3248513649231628
0.16   0.5818290187037815    0.39274352805567214   0.4473467299405445    0.4481117366204968
0.24   0.558009984528733     0.4876225384633936    0.47159377932986374   0.47217642967579065
0.32   0.49789962249729147   0.4164311037190864    0.44960067663427744   0.44998768359974883
0.4    0.4323442276396882    0.40383898773457055   0.4083332523661637    0.40856676013885534
0.48   0.3713218247265541    0.34637818509355833   0.36127142300144444   0.36139928463002213
0.56   0.317464442169613     0.3109275299927524    0.31488567526279976   0.31494638146837206
0.64   0.27090150060011975   0.26630527992421027   0.2720406656715109    0.2720612775581882
0.72   0.2309824444335763    0.23202872641141947   0.23377848912576604   0.23377664708367732
0.8    0.1968790041635059    0.19835787277456512   0.2002483463398119    0.20023496072148145
0.88   0.16778677013629673   0.1706863382350795    0.17118721023057526   0.1711687372607923
0.96   0.14298477188413108   0.1457191104511431    0.146164896484783     0.1461449872315685
1.04   0.12184585970085526   0.12471463806959114   0.12470601260323641   0.12468661633435679
1.12   0.10383101134762746   0.10638278273884474   0.10634799137116642   0.10633005812551842
1.2    0.08847924618718724   0.09083086771722519   0.09066631204307055   0.09065022455292165
1.28   0.07539714620630542   0.07744261050247567   0.07728319567425081   0.0772690275936581
```

```
1.36    0.06424924804705383    0.0660506864193279     0.0658682568348689     0.06585592331134593
1.44    0.05474961142161279    0.05630034829818351    0.05613549008711835    0.05612483332982787
1.52    0.046654546272164796   0.047995321764351846   0.04783881730713741    0.04782965374180339
1.6     0.03975638361519454    0.04090470168162353    0.040767298040945726   0.04075944320579417
1.68    0.03387815601108581    0.034863104417673596   0.034740522409569555   0.034733803210276824
1.76    0.028869060457465404   0.02971048620023035    0.029604407616938506   0.029598667570138903
1.84    0.024600590645275706   0.02531974403580975    0.02522746937769848    0.025222570086747293
1.92    0.020963240553208317   0.021576822786575613   0.021497567222105347   0.02149338792428282
2.0     0.017863695244798156   0.018387270010932356   0.018319090144704946   0.018315526353559458
```

Table 3.1: Numerical solution Y_n by Euler, AB2 and RK4 methods, and exact solution $y_e(x_n)$, all against x_n

The graph is shown below.

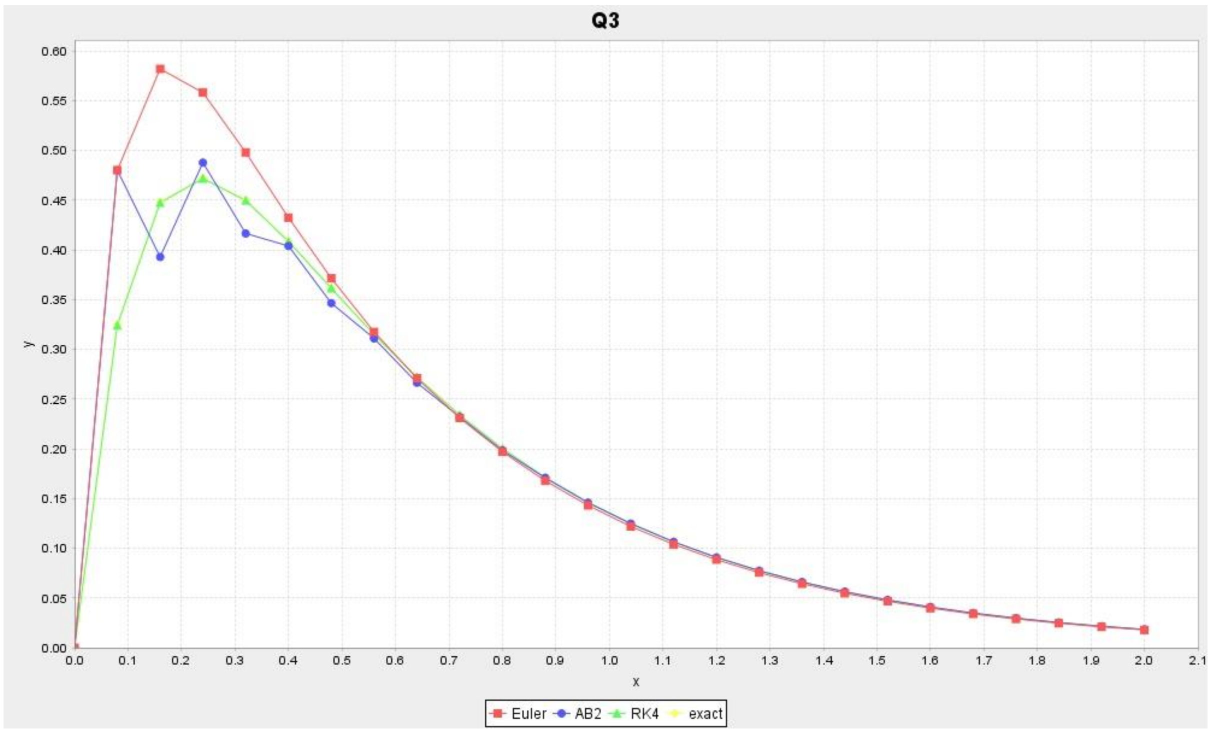

Figure 3.1: Numerical solution Y_n by Euler, AB2 and RK4 methods, and exact solution $y_e(x_n)$, all against x_n

Question 4

The class Q4 computes the required error. The results are shown below.

```
k      h               By Euler                  By AB2                    By RK4
0      0.16            0.5118882633795032        0.5118882633795032        -0.021515851003511943
1      0.08            0.13371728208328465       -0.05536820856482466      -7.650066799523292E-4
2      0.04            0.057656256051365096      -1.9884485800331086E-4    -3.6098200158984906E-5
3      0.02            0.027035576908105652      -1.6995554713622463E-4    -1.962196212401679E-6
4      0.01            0.013116459859215446      -6.619719641753896E-5     -1.1439171604399334E-7
5      0.005           0.006462991019929976      -1.9560276233898666E-5    -6.905267468937382E-9
6      0.0025          0.0032082817613272585     -5.261418713375932E-6     -4.241488826828288E-10
7      0.00125         0.0015984095782144991     -1.361233063712497E-6     -2.6280255749355774E-11
8      6.25E-4         7.97780837878348E-4       -3.460027303558988E-7     -1.6353030041216243E-12
9      3.125E-4        3.9853553168323064E-4     -8.720977362486337E-8     -1.0202949596305189E-13
10     1.5625E-4       1.9917918105427646E-4     -2.1890904255972288E-8    -5.828670879282072E-15
11     7.8125E-5       9.956746142480988E-5      -5.4837727692103044E-9    1.1102230246251565E-16
12     3.90625E-5      4.977820057294746E-5      -1.372323643611395E-9     1.27675647831893E-15
13     1.953125E-5     2.4887718018618E-5        -3.4325320363848277E-10   -6.106226635438361E-16
```

14	9.765625E-6	1.2443513475290935E-5	-8.583411759133242E-11	1.1657341758564144E-15
15	4.8828125E-6	6.221670358075304E-6	-2.146077759945797E-11	3.3306690738754696E-16

Table 4.1: Error E_n by Euler, AB2 and RK4 methods against h (and k)

The graph is shown below.

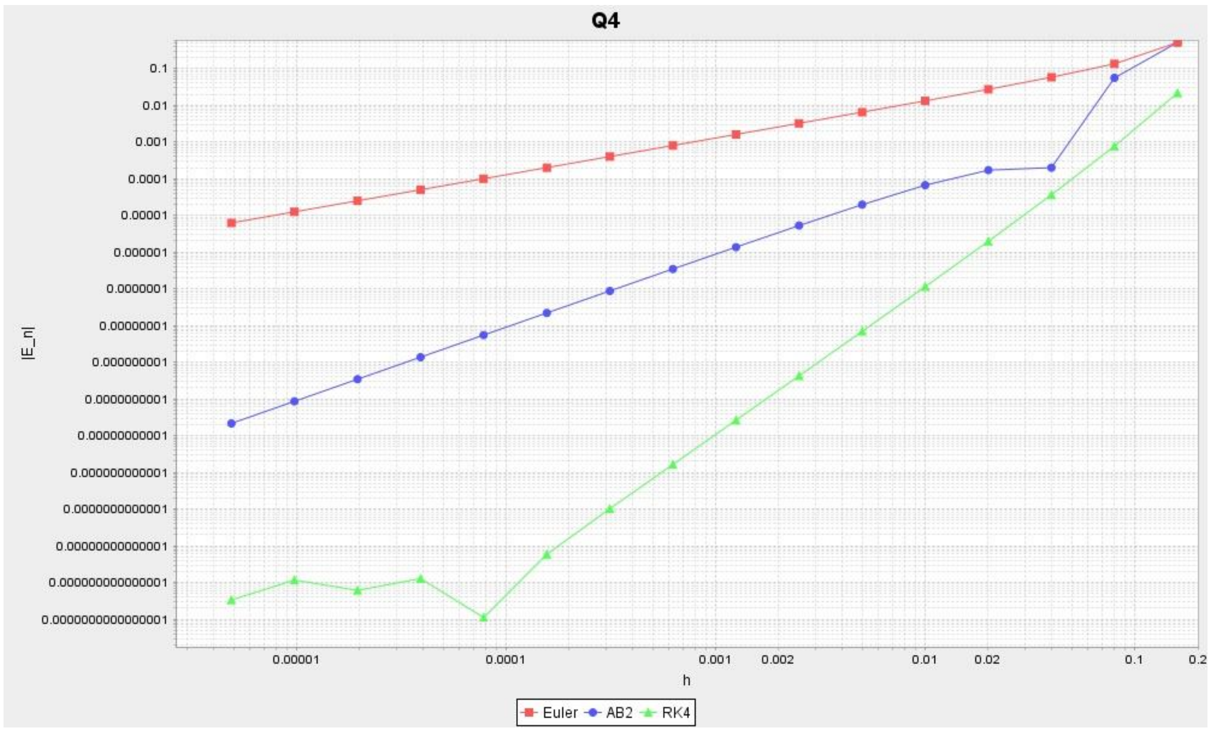

Figure 4.1: Error E_n by Euler, AB2 and RK4 methods against h

We notice that for $k \geq 13$ the error of RK4 becomes so small $(\sim 10^{-15})$ it reaches the limits of double precision, so the data there is not accurate.

The rest of the data looks linear, so we find the best linear fit to $\log|E_n|$ against $\log h$, for which we obtain the slopes 1.0414, 2.0942 and 4.1767, which implies $|E_n| = O(h^{1.0414})$ for the Euler method, $O(h^{2.0942})$ for AB2, and $O(h^{4.1767})$ for RK4.

We note that the slopes round off (therefore are close) to the respective theoretical accuracy orders of the methods. This can be explained if we regard $|E_n|$ to have the same order as the sum of local errors $|e_n|$, which is $O(h^{p+1}) \times n = O(h^{p+1}) \times O\left(\frac{1}{h}\right) = O(h^p)$.

Question 5

We consider the equation $\frac{d^2y}{dx^2} + p^2(1+x)^{-2}y = 0$.

Case 1: $p = 0$

Then $y = Ax + B$. Applying boundary conditions (ICs) gives the solution $y = x$.

Case 2: $p \geq 0$

The equation is equidimensional in $1 + x$, so we try $y = (1 + x)^r$. This gives $r(r-1) + p^2 = 0$ with roots $r_\pm = \frac{1 \pm \sqrt{1-4p^2}}{2}$. Then

$$y = \begin{cases} A(1+x)^{r_+} + B(1+x)^{r_-} & \text{if } p \neq \frac{1}{2} \\ (A + B\ln(1+x))(1+x)^{\frac{1}{2}} & \text{otherwise} \end{cases}.$$

Case 2.1: $p = \frac{1}{2}$

Applying ICs: $\begin{cases} 0 = A \\ 1 = B + \frac{A}{2} \end{cases}$, so $y = \ln(1+x)\,(1+x)^{\frac{1}{2}}$.

Case 2.2: $p \neq \frac{1}{2}$

Applying ICs: $\begin{cases} 0 = A + B \\ 1 = Ar_+ + Br_- \end{cases} \Rightarrow \begin{cases} A = \frac{1}{\sqrt{1-4p^2}} \\ B = -\frac{1}{\sqrt{1-4p^2}} \end{cases}$, so $y = \frac{1}{\sqrt{1-4p^2}}\left((1+x)^{r_+} - (1+x)^{r_-}\right)$.

Now we apply the boundary condition $y(1) = 0$. Cases 1 and 2.1 are hence rejected.

For case 2.2 we have $0 = 2^{r_+} - 2^{r_-} \Rightarrow 1 = 2^{r_+ - r_-} = e^{(\ln 2)\sqrt{1-4p^2}}$, which gives $p = \sqrt{\frac{1}{4} + \left(\frac{n\pi}{\ln 2}\right)^2}$ where $n \in \mathbb{Z}_{\geq 0}$.

As $p \neq \frac{1}{2}$, we have $n \geq 1$, the smallest eigenvalue is $p^{(1)} = \sqrt{\frac{1}{4} + \left(\frac{\pi}{\ln 2}\right)^2}$.

For $p = p^{(k)} = \sqrt{\frac{1}{4} + \left(\frac{k\pi}{\ln 2}\right)^2}$, we have

$$y^{(k)}(x) = \frac{1}{i\frac{2k\pi}{\ln 2}}\sqrt{1+x}\left((1+x)^{i\frac{k\pi}{\ln 2}} - (1+x)^{-i\frac{k\pi}{\ln 2}}\right) = \frac{\ln 2}{k\pi}\sqrt{1+x}\sin\frac{\ln(1+x)\cdot k\pi}{\ln 2}.$$

Therefore the eigenvalue-eigenfunction pairs are

$$\left(p^{(k)}, y^{(k)}(x)\right) = \left(\sqrt{\frac{1}{4} + \left(\frac{k\pi}{\ln 2}\right)^2}, \frac{\ln 2}{k\pi}\sqrt{1+x}\sin\frac{\ln(1+x)\cdot k\pi}{\ln 2}\right).$$

The RK4 method on (18) is implemented in the programming class Section3.

Question 6

The class Q6 computes the required data. The results for $p = 4$ are shown below.

```
p = 4
k      h              By RK4                 Exact                  Error
0      0.1            0.13576482860337882    0.13572667829769547    3.815030568335431E-5
1      0.05           0.1357294471282426     0.13572667829769547    2.768830547117407E-6
2      0.025          0.13572686124314215    0.13572667829769547    1.8294544668062684E-7
3      0.0125         0.1357266900048542     0.13572667829769547    1.1707158742435553E-8
4      0.00625        0.13572667903734328    0.13572667829769547    7.396478152177366E-10
5      0.003125       0.13572667834416274    0.13572667829769547    4.64672744726613E-11
6      0.0015625      0.13572667830060722    0.13572667829769547    2.9117541711087824E-12
7      7.8125E-4      0.13572667829787774    0.13572667829769547    1.8227086506783508E-13
8      3.90625E-4     0.13572667829770657    0.13572667829769547    1.1102230246251565E-14
9      1.953125E-4    0.13572667829769527    0.13572667829769547    -1.942890293094024E-16
10     9.765625E-5    0.13572667829769516    0.13572667829769547    -3.0531133177191805E-16
11     4.8828125E-5   0.13572667829769455    0.13572667829769547    -9.159339953157541E-16
12     2.44140625E-5  0.13572667829769108    0.13572667829769547    -4.385380947269368E-15
```

Table 6.1: Numerical solution Y_n RK4 method with $h = \frac{1}{2^k}$ for $p = 4$

Similar to Question 5, we throw away terms that are too small $(< 10^{-15})$ and fit the rest of the data, in which we obtain $|E_n| = O(h^{3.9595})$.

The results for $p = 5$ are shown below.

```
p = 5
k      h              By RK4                 Exact                  Error
0      0.1            -0.08569906425614551   -0.08584370354077758   1.4463928463206988E-4
1      0.05           -0.08583461818041233   -0.08584370354077758   9.085360365257422E-6
2      0.025          -0.08584314406303951   -0.08584370354077758   5.594777380685256E-7
3      0.0125         -0.08584366898613817   -0.08584370354077758   3.455463941370862E-8
4      0.00625        -0.08584370139635134   -0.08584370354077758   2.144426247685516E-9
5      0.003125       -0.08584370340726342   -0.08584370354077758   1.3351415806273081E-10
6      0.0015625      -0.08584370353244951   -0.08584370354077758   8.328074341257263E-12
7      7.8125E-4      -0.08584370354025772   -0.08584370354077758   5.198619312807296E-13
8      3.90625E-4     -0.08584370354074475   -0.08584370354077758   3.2834845953289005E-14
9      1.953125E-4    -0.08584370354077565   -0.08584370354077758   1.9290125052862095E-15
10     9.765625E-5    -0.08584370354077682   -0.08584370354077758   7.632783294297951E-16
11     4.8828125E-5   -0.08584370354078033   -0.08584370354077758   -2.7478019859472624E-15
12     2.44140625E-5  -0.08584370354077969   -0.08584370354077758   -2.1094237467877974E-15
```

Table 6.2: Numerical solution Y_n RK4 method with $h = \frac{1}{2^k}$ for $p = 4$

And we obtain $|E_n| = O(h^{4.009})$.

Both errors behave as expected, i.e. are near $O(h^4)$.

Question 7

The class Q7 implements the false position method. There are two main sources of error, the error of RK4, and the error due to the false position method accepting $|\phi(p)| < \epsilon$.

The first error has order Ch^4 for some constant C, so if we want error of $p^{(1)}$ within $\pm 5 \times 10^{-6}$ we should use $h \ll \left(\frac{5\times10^{-6}}{C}\right)^{\frac{1}{4}} = 0.04729C^{-\frac{1}{4}}$. We use the class Q6 to obtain C for $p = 5$ and we obtain 1.4797. As we expect the error to increase when $\vec{f}(x, \vec{y})$ in (18) fluctuates more, e.g. when p is larger, so we can assume that C for $p = p^{(1)}$ is at most 1.4797, so we can safely choose $h = \frac{1}{2^8}$.

The second error has size around $\frac{\epsilon}{|\phi'(p)|}$, so we want $\epsilon \leq \left(5 \times 10^{-6} - 1.4797 \left(\frac{1}{2^8}\right)^4\right) |\phi'(p)| = 4.9997 \times 10^{-6} \times |\phi'(p)|$. While I have failed to numerically estimate $\phi'(p)$, the graph generated by the class Rubbish seems to suggest that $\phi(p)$ is convex in $(4, 5)$ and $-0.2216 < \phi'(p) < -0.08584$.The graph is shown below.

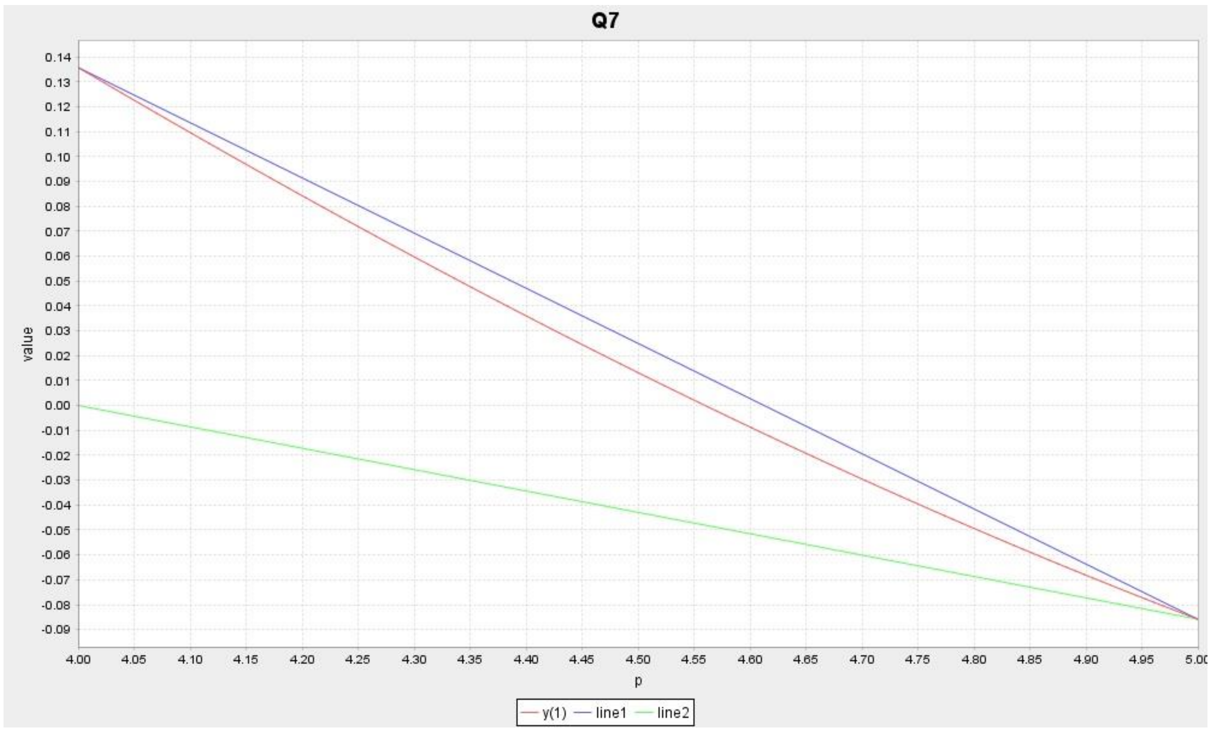

Figure 7.1: $y(1)$ ($= \phi(p)$; red curve) compared to straight lines (blue and green) with slopes –0.22 and –0.086

Therefore we can choose $\epsilon = 10^{-8}$.
The table below shows the iterates for $h = \frac{1}{2^8}$ and $\epsilon = 10^{-8}$.

```
p                       y(1)
4.0                     0.135726678410996
5.0                     -0.08584370321449729
4.612566884685908       -0.011333445629095757
4.565358346497272       -0.0011957529279220556
4.560421033512551       -1.2289009501231102E-4
4.559914074272174       -1.2595197429531718E-5
4.559862120050514       -1.2905392634209555E-6
4.559856796725826       -1.3222847117851794E-7
4.5598562512992595      -1.3548070849217497E-8
4.5598561954151         -1.3881287803246822E-9
p^(1) = 4.5598561954151
```

Table 7.1: Iterates of the false position method when finding y(1)

Question 8

We first want to locate roughly where $p^{(k)}$ is. While I have failed to numerically estimate them, the graph generated by the class Rubbish seems to suggest that the five smallest eigenvalues are in $(6, 20)$, $(20, 34)$, $(34, 48)$, $(48, 62)$ and $(62, 76)$. The graph is shown below.

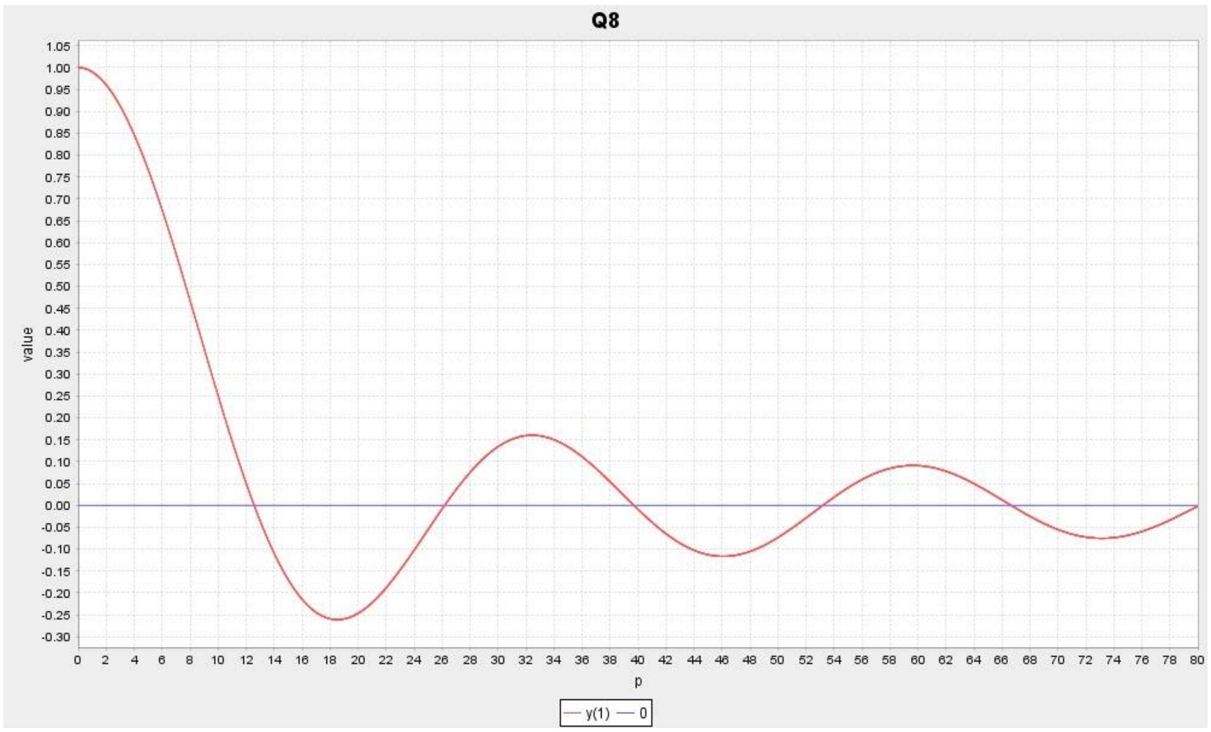

Figure 8.1: Graph of y(1) against p

The class Q8 finds p for $\alpha = 10$ by binary search instead false position because I have failed to deduce any accurate information about $\phi'(p)$ for calculating ϵ. We can circumvent the problem of determining ϵ by declaring $p = p_1$ when we know there is a root in the interval (p_1, p_2) (i.e. $\phi(p_1)$ and $\phi(p_2)$ have opposite signs), and $p_2 - p_1 \leq 5 \times 10^{-6} - Ch^4$.

Now we determine C and then h. Similar to Question 7 we expect the error to increase when $\vec{f}(x, \vec{y})$ in (18) fluctuates more, e.g. when p is larger or α is smaller, so this C should be bounded by the C when $p = 80$ and $\alpha = 2$. We calculate it by using the data (E_n against $h = \frac{1}{2^k}$) from the class Q6. The graph and table is shown below.

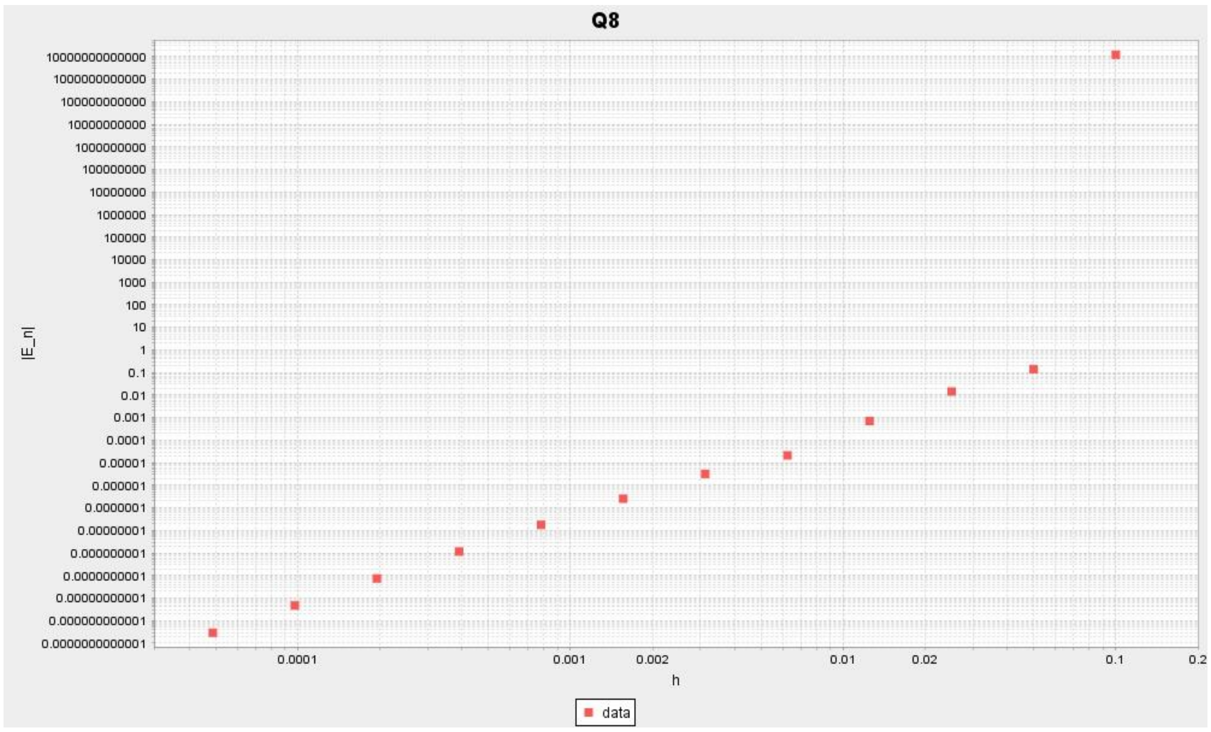

Figure 8.2: Log-log graph of $|E_n|$ against $h = \frac{1}{2^k}$

The data for $h = 1$ is out of place, and all data is at least 10^{-14}, so we fit all data except for the $h = 1$ one. This gives $C = 16507.8824$, so we can choose $h = 1/2^{10}$ and require $p_2 - p_1 < 10^{-6}$. Then the class Q8 gives the results $p^{(1)} = 12.57660$, $p^{(2)} = 26.18278$, $p^{(3)} = 39.71110$, $p^{(4)} = 53.19890$ and $p^{(5)} = 66.66330$.

Q8 also computes the normalisation constant by Simpson's rule and graph the normalised eigenfunctions. The results are shown below.

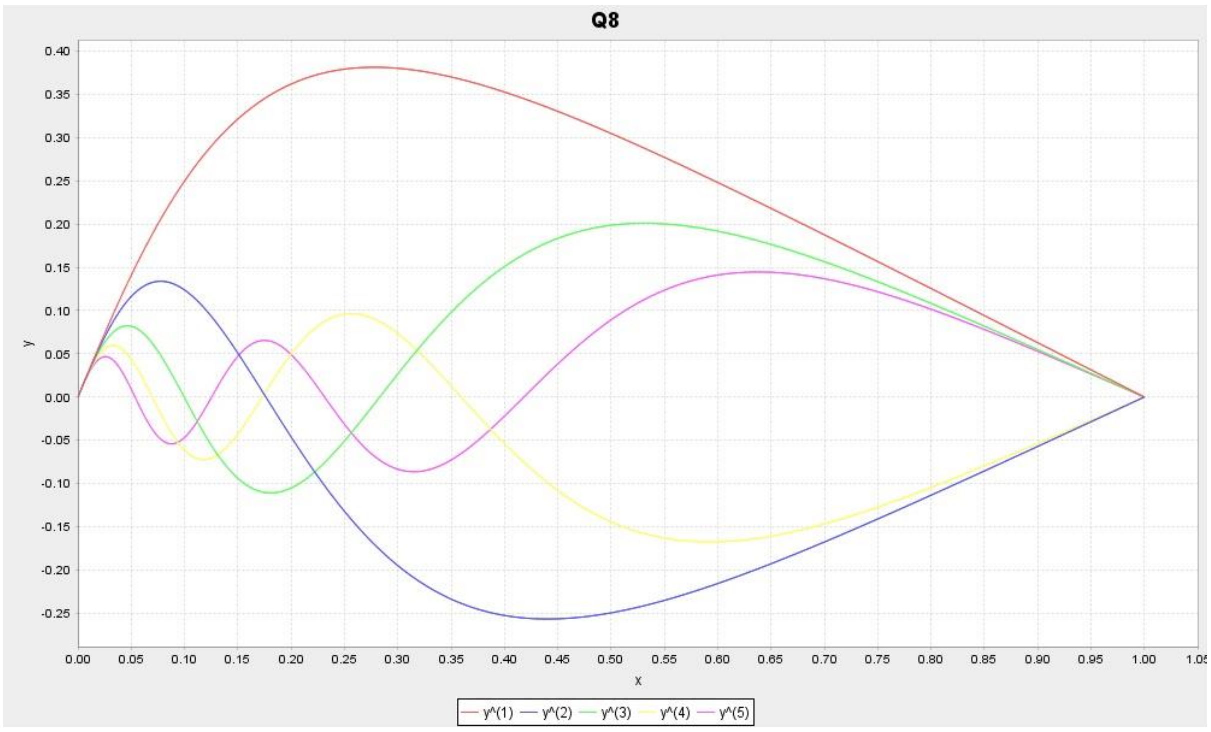

Figure 8.3: Normalised eigenfunctions $y^{(k)}(x)$

Mathematically the eigenfunctions look like sine waves of different cycle lengths scaled in a way such that the amplitudes and the cycle lengths are larger when x is larger.

Physically the sine waves are fundamental modes of the wave equation with uniform density. The scaling is due to lower density of the string as x increases. Lower density implies lighter, and thus easier to move, so the string oscillates at a larger amplitude. Lighter also implies less force is required to oscillate the string. As tension is uniform in the string, and force depends positively on tension and curvature, the curvature of the lighter part of the string must be lower. This explains the shape of the curve observed above.

Appendix

Main classes

Section2.java

```java
package project;

class Section2 {

	static double f(double x, double y) {

		return -8 * y + 6 * Math.exp(-2 * x);

	}

	static double y_e(double x) {

		return Math.exp(-2 * x) - Math.exp(-8 * x);

	}

	static double[] euler(double h, int n) {

		double Y[] = new double[n+1];
		Y[0] = 0;

		for (int i = 0; i < n; i++)
			Y[i+1] = Y[i] + h * f(i*h, Y[i]);

		return Y;

	}

	static double[] ab2(double h, int n) {

		double Y[] = new double[n+1];
		Y[0] = 0;

		Y[1] = Y[0] + h * f(0, Y[0]);

		for (int i = 1; i < n; i++)
			Y[i+1] = Y[i] + h * (1.5d * f(i*h, Y[i]) - 0.5d * f((i-1)*h, Y[i-1]));

		return Y;

	}

	static double[] rk4(double h, int n) {

		double Y[] = new double[n+1];
		Y[0] = 0;

		for (int i = 0; i < n; i++) {
			double k1 = f(i*h, Y[i]);
			double k2 = f(i*h + h/2, Y[i] + h/2*k1);
			double k3 = f(i*h + h/2, Y[i] + h/2*k2);
			double k4 = f(i*h + h, Y[i] + h*k3);
			Y[i+1] = Y[i] + h / 6 * (k1 + 2*k2 + 2*k3 + k4);
		}

		return Y;

	}

}
```

Q1.java

```java
package project;

public class Q1 {

	public static void main(String[] args) {

		double[] hList = {0.5, 0.375, 0.25, 0.125, 0.1, 0.05};
		int[] printStep = {1, 1, 2, 4, 5, 10};

		for (int j = 0; j < 6; j++) {

			double h = hList[j];
			int n = (int) (3d / h);

			System.out.println("h = " + h);
			String format = "%-15s%-30s%-30s%-30s\n";
			System.out.printf(format, "x_n", "Y_n", "y_e(x_n)", "E_n");
			System.out.printf(format, 0, 0, 0, 0);

			double Y[] = Section2.ab2(h, n);
			double y[] = new double[99];
			y[0] = 0;

			for (int i = 0; i < n; i++) {
				y[i+1] = Section2.y_e((i+1)*h);
				if((i+1) % printStep[j] == 0)
					System.out.printf(format, (i+1)*h, Y[i+1], y[i+1], (Y[i+1]-y[i+1]));
			}

			double tmp1 = 0, tmp2 = 0, tmp3 = 0, tmp4 = 0;
			for (int i = 1; i < 7; i++) {
				tmp1 += i*h * Math.log(Math.abs(Y[i]-y[i]));
				tmp2 += i*h;
				tmp3 += Math.log(Math.abs(Y[i]-y[i]));
				tmp4 += i*i*h*h;
			}
```

```java
                        System.out.println("\u03b3 = " + (6*tmp1 - tmp2*tmp3) / (6*tmp4 - tmp2*tmp2));

                        System.out.println();

                }

        }

}
```

Q3.java

```java
package project;

import java.io.IOException;

public class Q3 {

	public static void main(String[] args) throws IOException {

		double[] x = new double[26];
		for (int i = 0; i < 26; i++)
			x[i] = 0.08d * i;
		double[] a = Section2.euler(0.08, 25);
		double[] b = Section2.ab2(0.08, 25);
		double[] c = Section2.rk4(0.08, 25);
		double[] exact = new double[26];
		for (int i = 0; i < 26; i++) exact[i] = Section2.y_e(0.08d * i);

		String format = "%-7s%-22s%-22s%-22s%-22s\n";
		System.out.printf(format, "x_n", "By Euler", "By AB2", "By RK4", "Exact");
		for (int i = 0; i < 26; i++)
			System.out.printf(format, 0.08d * i, a[i], b[i], c[i], exact[i]);

		graphs.XYDataset d = new graphs.XYDataset("Q3", "x", "y");
		d.addSeries(x, a, 26, "Euler");
		d.addSeries(x, b, 26, "AB2");
		d.addSeries(x, c, 26, "RK4");
		d.addSeries(x, exact, 26, "exact");
		d.plotLine(true);
	//	d.display();
		d.save("F3.1.jpg");

	}

}
```

Q4.java

```java
package project;

import java.io.IOException;

public class Q4 {

	public static void main(String[] args) throws IOException {

		double[] a = new double[16];
		double[] b = new double[16];
		double[] c = new double[16];
		double[] x = new double[16];
		String format = "%-6s%-16s%-25s%-25s%-25s\n";

		System.out.printf(format, "k", "h", "By Euler", "By AB2", "By RK4");
		for (int k = 0; k < 16; k++) {
			int n = 1 << k;
			double exact = Section2.y_e(0.16d);
			a[k] = Section2.euler(0.16d / n, n)[n] - exact;
			b[k] = Section2.ab2(0.16d / n, n)[n] - exact;
			c[k] = Section2.rk4(0.16d / n, n)[n] - exact;
			x[k] = 0.16d / n;
			System.out.printf(format, k, x[k], a[k], b[k], c[k]);
			a[k] = Math.abs(a[k]);
			b[k] = Math.abs(b[k]);
			c[k] = Math.abs(c[k]);
		}

		graphs.XYDataset d = new graphs.XYDataset("Q4", "h", "|E_n|");
		d.addSeries(x, a, 16, "Euler");
		d.addSeries(x, b, 16, "AB2");
		d.addSeries(x, c, 16, "RK4");
		d.useLogAxis('x');
		d.useLogAxis('y');
		d.plotLine(true);
	//	d.display();
```

```
            d.save("F4.1.jpg");

            System.out.println((new graphs.XYData(x, a, 16)).powerFit1()[1]);
            System.out.println((new graphs.XYData(x, b, 16)).powerFit1()[1]);
            System.out.println((new graphs.XYData(x, c, 12)).powerFit1()[1]);

      }

}
```

Section3.java

```java
package project;

class Section3 {

	double p, a;

	Section3(double pVal, double aVal) {

		p = pVal;
		a = aVal;

	}

	private double[] add(double[] vector1, double[] vector2) {

		int n = vector1.length;
		if (n != vector2.length) return null;
		else {
			double[] v = new double[n];
			for (int i = 0; i < n; i++) v[i] = vector1[i] + vector2[i];
			return v;
		}

	}

	private double[] multiply(double scalar, double[] vector) {

		int n = vector.length;
		double[] v = new double[n];
		for (int i = 0; i < n; i++) v[i] = vector[i] * scalar;
		return v;

	}

	double[] f(double x, double[] y) {

```

```java
            return new double[] {y[1], -p * p * Math.pow(1 + x, -a) * y[0]};

        }

        double[][] rk4(double h, int n) {

            double Y[][] = new double[n+1][2];
            Y[0][0] = 0;
            Y[0][1] = 1;

            for (int i = 0; i < n; i++) {
                double[] k1 = f(i*h, Y[i]);
                double[] k2 = f(i*h + h/2, add(Y[i], multiply(h/2, k1)));
                double[] k3 = f(i*h + h/2, add(Y[i], multiply(h/2, k2)));
                double[] k4 = f(i*h + h, add(Y[i], multiply(h, k3)));
                Y[i+1] = add(Y[i], multiply(h/6, add(add(add(k1, multiply(2, k2)), multiply(2, k3)), k4)));
            }

            return Y;

        }

        static double y_e2(double x, double p) {

            double C = Math.sqrt(4*p*p - 1) / 2;
            return Math.sqrt(1+x) / C * Math.sin(Math.log(1+x) * C);

        }

}
```

Q6.java

```java
package project;

import java.io.IOException;

public class Q6 {

	public static void main(String[] args) throws IOException {

		double p = 4, a = 2;

		Section3 s = new Section3(p, a);
		String format = "%-6s%-15s%-24s%-24s%-24s\n";
		System.out.printf(format, "k", "h", "By RK4", "Exact", "Error");

		double[] xData = new double[13];
		double[] yData = new double[13];

		for (int k = 0; k < 13; k++) {
			double c = s.rk4(1d / (10 << k), (10 << k))[10 << k][0];
			double exact = Section3.y_e2(1, p);
			System.out.printf(format, k, 1d / (10 << k), c, exact, c - exact);
			xData[k] = 1d / (10 << k);
			yData[k] = Math.abs(c - exact);
		}

		graphs.XYDataset d = new graphs.XYDataset("Q6", "h", "|E_n|");
		d.addSeries(xData, yData, 12, "data");
		d.useLogAxis('x');
		d.useLogAxis('y');
		d.plotScatter();
		d.display();
	//	d.save("F8.2.jpg");

		double[] fit = (new graphs.XYData(xData, yData, 0, 8)).powerFit1();
		System.out.println(fit[0] + "    " + fit[1]);

```

```
        }

}
```

Q7.java

```java
package project;

public class Q7 {

	private static int a = 2, n = 256;
	private static String format = "%-24s%-34s\n";

	private static double falsePosition(double p1, double p2, double epsilon) {

		if (p1 < p2) {
			double q1 = (new Section3(p1, a)).rk4(1d/n, n)[n][0];
			System.out.printf(format, p1, q1);
			if (Math.abs(q1) < epsilon) return p1;
			else {
				double q2 = (new Section3(p2, a)).rk4(1d/n, n)[n][0];
				System.out.printf(format, p2, q2);
				if (Math.abs(q2) < epsilon) return p2;
				else if (q1 * q2 < 0) return falsePosition(p1, q1, p2, q2, epsilon);
				else return search(p1, q1, p2, epsilon, (byte) Math.signum(q1), 0);
			}
		} else return Double.NaN;

	}

	private static double search(double p1, double q1, double p2, double epsilon, byte sign, int stack) {

		double p = (p1 + p2) / 2;
		double q = (new Section3(p, a)).rk4(1d/n, n)[n][0];
		System.out.printf(format, p, q);

		if (stack > 15) return Double.NaN;
		else if (q * sign > 0) {
			double q7 = search(p1, q1, p, epsilon, sign, stack + 1);
			if (Double.isNaN(q7)) return search(p, q, p2, epsilon, sign, stack + 1);
			else return q7;
		} else return falsePosition(p1, q1, p, q, epsilon);

	}

	private static double falsePosition(double p1, double q1, double p2, double q2, double epsilon) {

		double p = (q2*p1 - q1*p2) / (q2 - q1);
		double q = (new Section3(p, a)).rk4(1d/n, n)[n][0];
		System.out.printf(format, p, q);

		if (Math.abs(q) < epsilon) return p;
		else if (q * q1 < 0) return falsePosition(p1, q1, p, q, epsilon);
		else return falsePosition(p, q, p2, q2, epsilon);

	}

	public static void main(String[] args) {

		System.out.printf(format, "p", "y(1)");
		double p = falsePosition(4, 5, 0.00000001);
		System.out.println("p^(1) = " + p);

	}

}
```

Q8.java

```java
package project;

import java.io.IOException;

public class Q8 {

	private static int a = 10, n = 1024;

	private static double binarySearch(double p1, double p2, double delta) {

		if (p1 < p2) {
			double q1 = (new Section3(p1, a)).rk4(1d/n, n)[n][0];
			double q2 = (new Section3(p2, a)).rk4(1d/n, n)[n][0];
			if (q1 * q2 < 0) {
				if (p2 - p1 < delta) return p1;
				else return binarySearch(p1, q1, p2, q2, delta);
			}
			else return search(p1, q1, p2, delta, (byte) Math.signum(q1), 0);
		} else return Double.NaN;

	}

	private static double search(double p1, double q1, double p2, double delta, byte sign, int stack) {

		double p = (p1 + p2) / 2;
		double q = (new Section3(p, a)).rk4(1d/n, n)[n][0];

		if (stack > 15) return Double.NaN;
		else if (q * sign > 0) {
			double q7 = search(p1, q1, p, delta, sign, stack + 1);
			if (Double.isNaN(q7)) return search(p, q, p2, delta, sign, stack + 1);
			else return q7;
		} else return binarySearch(p1, q1, p, q, delta);

	}

	private static double binarySearch(double p1, double q1, double p2, double q2, double delta) {

		double p = (p1 + p2) / 2;
		double q = (new Section3(p, a)).rk4(1d/n, n)[n][0];

		if (q * q1 < 0) {
			if (p - p1 < delta) return p1;
			else return binarySearch(p1, q1, p, q, delta);
		} else {
			if (p2 - p < delta) return p;
			else return binarySearch(p, q, p2, q2, delta);
		}

	}

	public static void main(String[] args) throws IOException {

		double[] p = {
			binarySearch(6, 20, 0.000001),
			binarySearch(20, 34, 0.000001),
			binarySearch(34, 48, 0.000001),
			binarySearch(48, 62, 0.000001),
			binarySearch(62, 76, 0.000001)
		};

		graphs.XYDataset d = new graphs.XYDataset("Q8", "x", "y");
		double xData[] = new double[n + 1];
		for (int j = 0; j <= n; j++) xData[j] = (double) j / n;

		for (int i = 0; i < 5; i++) {

			System.out.println("p^(" + (i+1) + ") = " + p[i]);
			double yData[] = new double[n + 1];
			for (int j = 0; j <= n; j++)
				yData[j] = (new Section3(p[i], 10)).rk4(1d/n, n)[j][0];

			double tmp[] = new double[n + 1];
			for (int j = 0; j <= n; j++)
```

```java
                        tmp[j] = Math.pow(1 + xData[j], -10) * yData[j] * yData[j];
                    double normaliser = tmp[0] - tmp[n];
                    for (int k = 1; k <= n / 2; k++)
                        normaliser += 4 * tmp[2*k - 1] + 2 * tmp[2*k];
                    normaliser = p[i] * Math.sqrt(normaliser / 3 / n);
                    for (int j = 0; j <= n; j++)
                        yData[j] /= normaliser;

                    d.addSeries(xData, yData, n + 1, "y^(" + (i+1) + ")");
                }

                d.plotLine(false);
        //      d.display();
                d.save("F8.3.jpg");

        }

}
```

Rubbish.java

```java
package project;

import java.io.IOException;

public class Rubbish {

	public static void main(String[] args) throws IOException {

		int a = 2;
		int n = 256;
		double[] xData = new double[129];
		double[] yData = new double[129];

		for (int i = 0; i < 129; i++) {
			xData[i] = 4 + (double) i / 128;
			yData[i] = (new Section3(xData[i], a)).rk4(1d / n, n)[n][0];
		}

		graphs.XYDataset d = new graphs.XYDataset("Q7", "p", "value");
		d.addSeries(xData, yData, 129, "y(1)");
		d.addSeries(new double[] {4, 5}, new double[] {yData[0], yData[128]}, 2, "line1");
		d.addSeries(new double[] {4, 5}, new double[] {0, yData[128]}, 2, "line2");
		d.setRange('x', 4, 5);
		d.plotLine(false);
	//	d.display();
		d.save("F7.1.jpg");

		System.out.println(yData[128] - yData[0]);
		System.out.println(yData[128]);

		a = 10;
		xData = new double[8001];
		yData = new double[8001];

		for (int i = 0; i < 8001; i++) {
			xData[i] = (double) i / 100;
```

```
                yData[i] = (new Section3(xData[i], a)).rk4(1d / n, n)[n][0];
            }

            d = new graphs.XYDataset("Q8", "p", "value");
            d.addSeries(xData, yData, 8001, "y(1)");
            d.addSeries(new double[] {0, 80}, new double[] {0, 0}, 2, "0");
            d.setRange('x', 0, 80);
            d.plotLine(false);
//          d.display();
            d.save("F8.1.jpg");

        }

}
```

Libraries

I used the external libraries JCommon v1.0.23 (http://www.jfree.org/jcommon/) and JFreeChart v1.0.19 (http://www.jfree.org/jfreechart/) , and some classes from my own library MyLibrary as shown below.

XYData.java

```java
package graphs;

public class XYData {

	public double[] x, y;
	int n;

	public XYData(double[][] data, int number) {

		n = number;

		x = new double[n];
		for (int i = 0; i < number; i++) x[i] = data[i][0];

		y = new double[n];
		for (int i = 0; i < number; i++) y[i] = data[i][1];


	}

	public XYData(double[] xData, double[] yData, int number) {

		n = number;
		x = xData;
		y = yData;

	}

	private double[] linearFit(double[] x, double[] y) {

		double sumX = 0, sumX2 = 0, sumXY = 0, sumY = 0;
		for (int i = 0; i < n; i++) {
			sumX += x[i];
			sumX2 += x[i] * x[i];
			sumXY += x[i] * y[i];
			sumY += y[i];
		}
		return new double[] {(sumY*sumX2 - sumX*sumXY) / (n*sumX2 - sumX*sumX), (n*sumXY - sumX*sumY) / (n*sumX2 - sumX*sumX)};

	}

	public double[] linearFit() {

		return linearFit(x, y);

	}

	private double[] log(double[] a) {

		double[] b = new double[n];
		for (int i = 0; i < n; i++) b[i] = Math.log(a[i]);
		return b;

	}

	public double[] exponentialFit1() {

		double[] b = linearFit(x, log(y));
		return new double[] {Math.exp(b[0]), b[1]};

	}

	public double[] powerFit1() {

		double[] b = linearFit(log(x), log(y));
		return new double[] {Math.exp(b[0]), b[1]};
```

```

        }

}
```

XYDataset.java

```java
package graphs;

import java.io.File;
import java.io.IOException;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.axis.LogarithmicAxis;
import org.jfree.chart.axis.NumberTickUnit;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.*;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

public class XYDataset {

	public XYDataset(String chartTitle, String xLabel, String yLabel) {

		title = chartTitle;
		xLbl = xLabel;
		yLbl = yLabel;
		xAxis.setLabel(xLabel);
		yAxis.setLabel(yLabel);

	}

	private XYSeriesCollection collection = new XYSeriesCollection();
	private String title, xLbl, yLbl;
	private NumberAxis xAxis = new NumberAxis();
	private NumberAxis yAxis = new NumberAxis();
	private JFreeChart chart;

	public void addSeries(double[][] data, int number, String key) {

        XYSeries series = new XYSeries(key);
        for (int i = 0; i < number; i++) series.add(data[i][0], data[i][1]);
        collection.addSeries(series);

    }

    public void addSeries(double[] xData, double[] yData, int number, String key) {

        XYSeries series = new XYSeries(key);
        for (int i = 0; i < number; i++) series.add(xData[i], yData[i]);
        collection.addSeries(series);

    }

    public void addSeries(XYData d1, String key) {

        XYSeries series = new XYSeries(key);
        for (int i = 0; i < d1.n; i++) series.add(d1.x[i], d1.y[i]);
        collection.addSeries(series);

    }

    public void useNumberAxis(char axis) {

        if (axis == 'x') xAxis = new NumberAxis(xLbl);
        if (axis == 'y') yAxis = new NumberAxis(yLbl);

    }

    public void useLogAxis(char axis) {

        if (axis == 'x') xAxis = new LogarithmicAxis(xLbl);
        if (axis == 'y') yAxis = new LogarithmicAxis(yLbl);

    }

    public void setRange(char axis, double lower, double higher) {

          if (axis == 'x') xAxis.setRange(lower, higher);
          if (axis == 'y') yAxis.setRange(lower, higher);

    }

    public void setTickUnit(char axis, double tick) {

          if (axis == 'x') xAxis.setTickUnit(new NumberTickUnit(tick));
          if (axis == 'y') yAxis.setTickUnit(new NumberTickUnit(tick));

    }

    public void display() {

          ApplicationFrame af = new ApplicationFrame(chart.getTitle().getText());

          ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(1000, 600));

        af.setContentPane(chartPanel);
          af.pack();
          RefineryUtilities.centerFrameOnScreen(af);
          af.setVisible(true);

    }

    public void save(String file) throws IOException {

          ChartUtilities.saveChartAsJPEG(new File(file), chart, 1000, 600);

    }

    public void plotLine(boolean showShape) {

          chart = new JFreeChart(title, new XYPlot(collection, xAxis, yAxis, new XYLineAndShapeRenderer(true, showShape)));

    }
```

```java

	public void plotScatter() {

		chart = new JFreeChart(title, new XYPlot(collection, xAxis, yAxis, new XYLineAndShapeRenderer(false, true)));

	}

}
```