Solutions to Exercises from Chapter 08

Contents

The `comphy` package is loaded once at the beginning so to make all its functions available to this exercises session.

```
library(comphy)
```

1 Exercises on IVPs

1.1 Exercise 01

The solution to the following IVPs,

$$ty' + y = 2t, \quad y(1) = 0,$$

is

$$y(t) = t - \frac{1}{t}, \quad t \neq 0.$$

Solve this ODE numerically using `EulerODE` and compare the result visually with the exact solution fot $t \in [1, 3]$, when using step size $h = 0.4, 0.2, 0.1$. What error is expected for the solution at $t = 3$? Is this reasonable?

SOLUTION

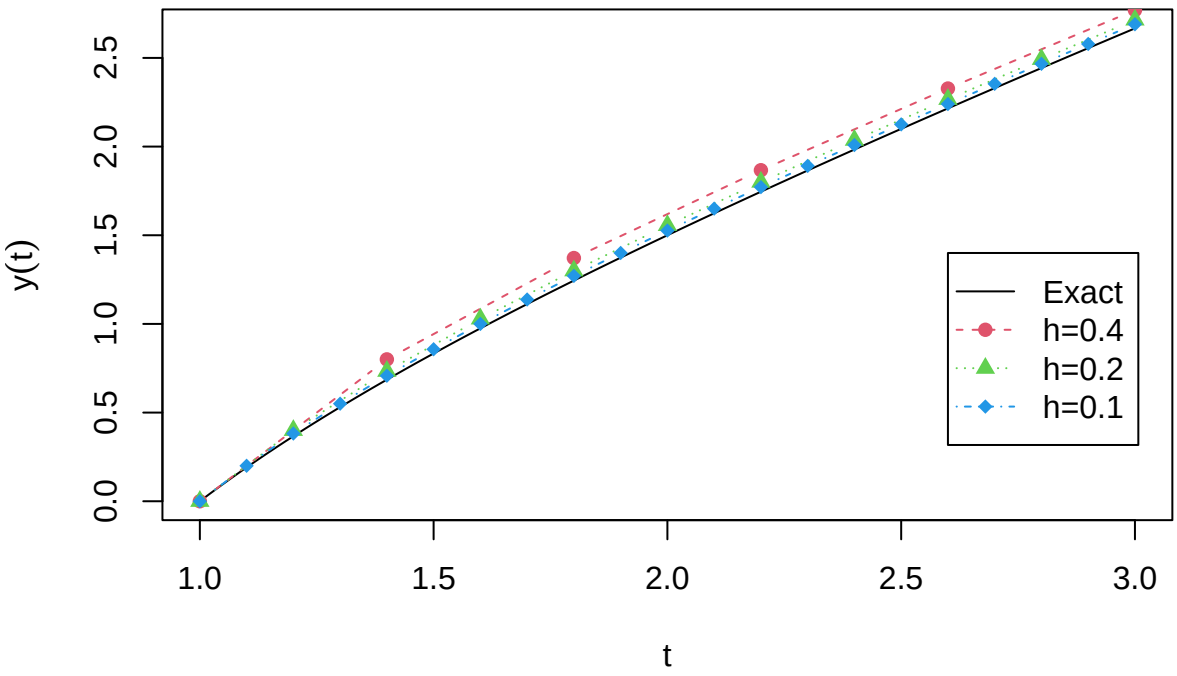
The application of the solver `EulerODE` is relatively straightforward, once the gradient of the ODE is defined. The ODE can, in fact, be re-written as

$$y' \equiv \frac{\mathrm{d}y}{\mathrm{d}t} = \frac{1}{t}\left(2t - y\right).$$

It is also important to remember that here $t_0 = 1$ (and not $t_0 = 0$, a common mistake) and $t_f = 3$.

```
# Define the gradient
f <- function(t,y) {ff <- (2*t-y)/t; return(ff)}
```

```r
# Solution interval
t0 <- 1
tf <- 3

# Initial conditions
y0 <- 0

# Step sizes
h1 <- 0.4; h2 <- 0.2; h3 <- 0.1

# Euler solver
ltmp1 <- EulerODE(f,t0,tf,y0,h1)
ltmp2 <- EulerODE(f,t0,tf,y0,h2)
ltmp3 <- EulerODE(f,t0,tf,y0,h3)

# Exact solution
tt <- seq(1,3,length.out=100)
yy <- tt-1/tt

# Visual comparisons
plot(tt,yy,type="l",
     xlab=expression(t),ylab=expression(y(t)))
points(ltmp1$t,ltmp1$y,type="b",pch=16,col=2,lty=2)
points(ltmp2$t,ltmp2$y,type="b",pch=17,col=3,lty=3)
points(ltmp3$t,ltmp3$y,type="b",pch=18,col=4,lty=4)
legend(2.6,1.4,legend=c("Exact","h=0.4","h=0.2","h=0.1"),
       pch=c(-1,16,17,18),col=c(1,2,3,4),lty=c(1,2,3,4))
```

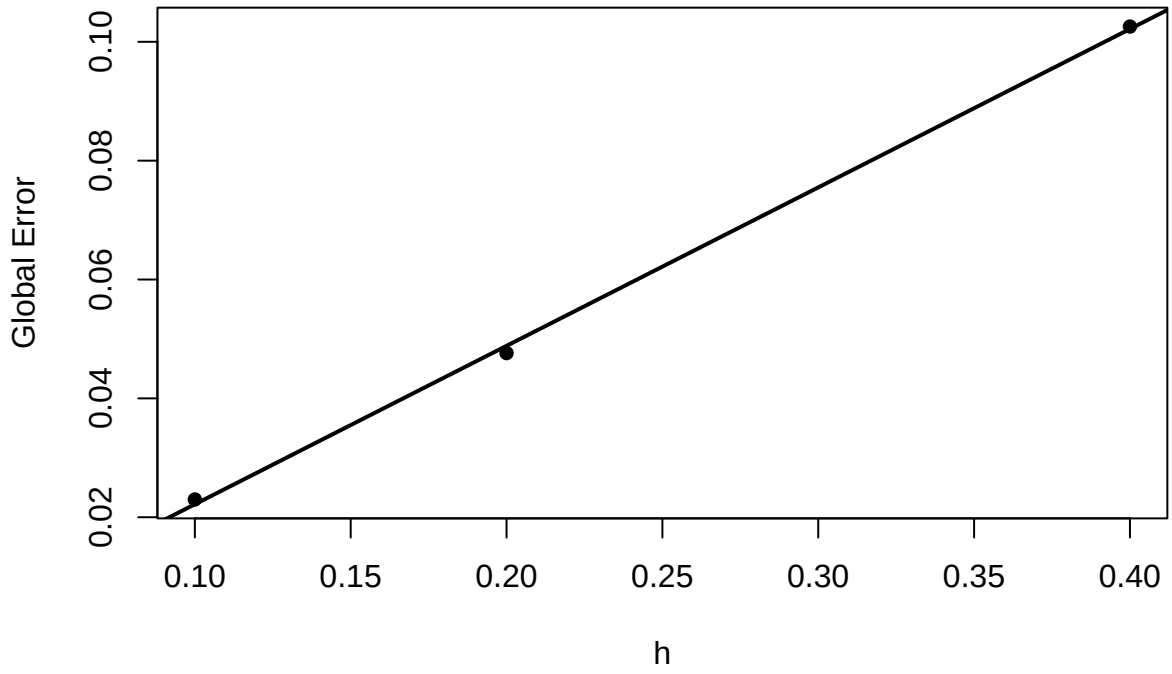


It is visually evident that the accuracy of the numerical solution increases with the decreasing size of h.

We know that the global error for the Euler method is $\mathrm{O}(h)$, which means that the difference $E = |y(t_f) - y_n|$ should decrease linearly with the decrease of h.

```
# Values at t=tf
yk <- yy[length(yy)]
yk1 <- ltmp1$y[length(ltmp1$y)]
yk2 <- ltmp2$y[length(ltmp2$y)]
yk3 <- ltmp3$y[length(ltmp3$y)]

# Errors
D1 <- abs(yk-yk1)
D2 <- abs(yk-yk2)
D3 <- abs(yk-yk3)

# Regression (linear dependency on h)
h <- c(h1,h2,h3)
D <- c(D1,D2,D3)
mdel <- lm(D ~ h)
print(summary(mdel))
#>
#> Call:
#> lm(formula = D ~ h)
#>
#> Residuals:
#>         1         2         3
```

```
#>  0.000406 -0.001218  0.000812
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -0.004484   0.001861   -2.41   0.2504
#> h            0.266605   0.007032   37.91   0.0168 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.001519 on 1 degrees of freedom
#> Multiple R-squared:  0.9993, Adjusted R-squared:  0.9986
#> F-statistic:  1437 on 1 and 1 DF,  p-value: 0.01679
plot(c(h1,h2,h3),c(D1,D2,D3),pch=16,xlab="h",ylab="Global Error")
abline(mdel,lwd=2)
```



The linear dependency of the global error on h is clearly visible.

1.2 Exercise 02

The solution to the following IVPs,

$$y' + \frac{1}{x}y = xy^2, \quad y(4) = -\frac{1}{4},$$

is

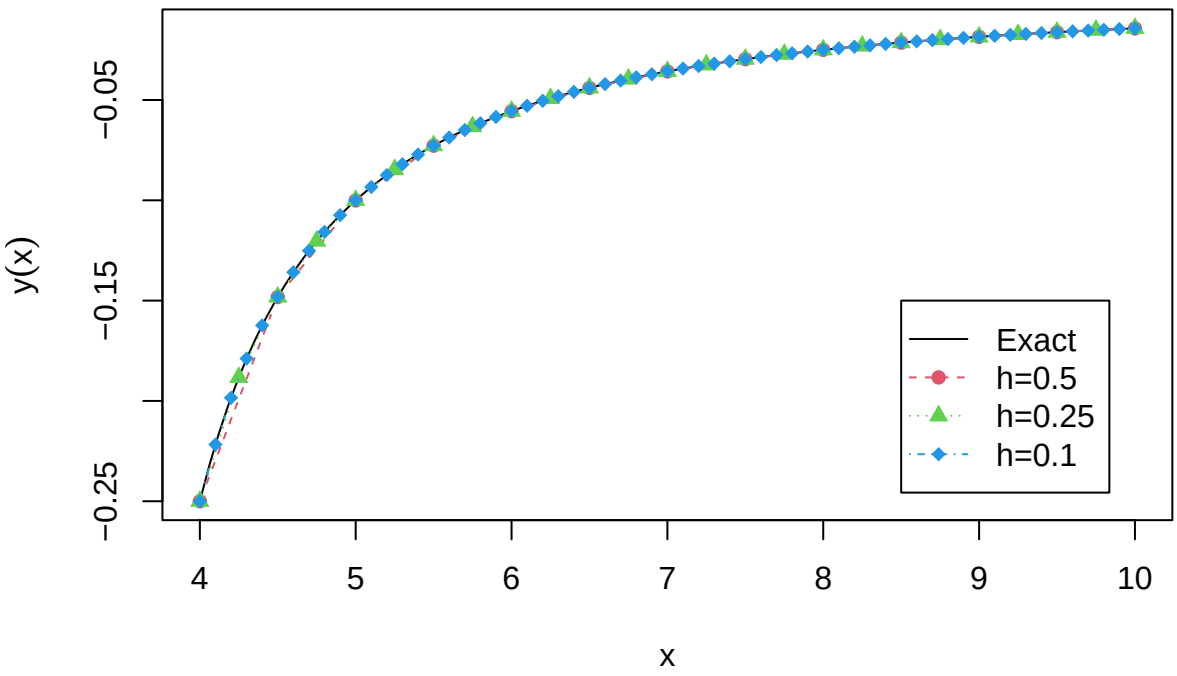
$$y(x) = \frac{1}{3x - x^2}.$$

Solve this ODE numerically using `RK4ODE` and compare the result visually with the exact solution fot $t \in [4, 10]$, when using step size $h = 0.5, 0.25, 0.1$. How can you extract the number of steps used by the method?

SOLUTION

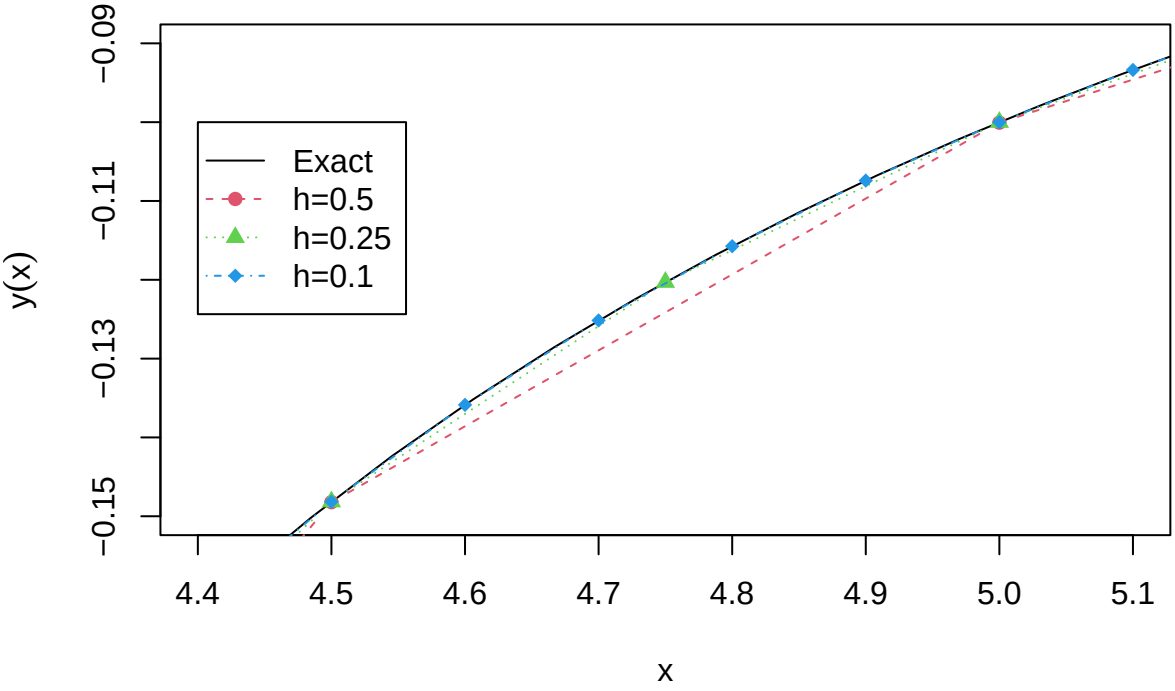
The exercise is straightforward. Functions, interval's extremes and initial conditions must be defined first. Then the algorithm is applied using `RK4ODE`.

```
# Define the gradient
f <- function(x,y) {ff <- x*y^2-y/x; return(ff)}

# Solution interval
t0 <- 4
tf <- 10

# Initial conditions
y0 <- -0.25

# Step sizes
h1 <- 0.5; h2 <- 0.25; h3 <- 0.1

# RK4 solver
ltmp1 <- RK4ODE(f,t0,tf,y0,h1)
ltmp2 <- RK4ODE(f,t0,tf,y0,h2)
ltmp3 <- RK4ODE(f,t0,tf,y0,h3)

# Exact solution
xx <- seq(4,10,length.out=100)
yy <- 1/(3*xx-xx^2)

# Visual comparisons
plot(xx,yy,type="l",
     xlab=expression(x),ylab=expression(y(x)))
points(ltmp1$t,ltmp1$y,type="b",pch=16,col=2,lty=2)
points(ltmp2$t,ltmp2$y,type="b",pch=17,col=3,lty=3)
points(ltmp3$t,ltmp3$y,type="b",pch=18,col=4,lty=4)
legend(8.5,-0.15,legend=c("Exact","h=0.5","h=0.25","h=0.1"),
       pch=c(-1,16,17,18),col=c(1,2,3,4),lty=c(1,2,3,4))
```



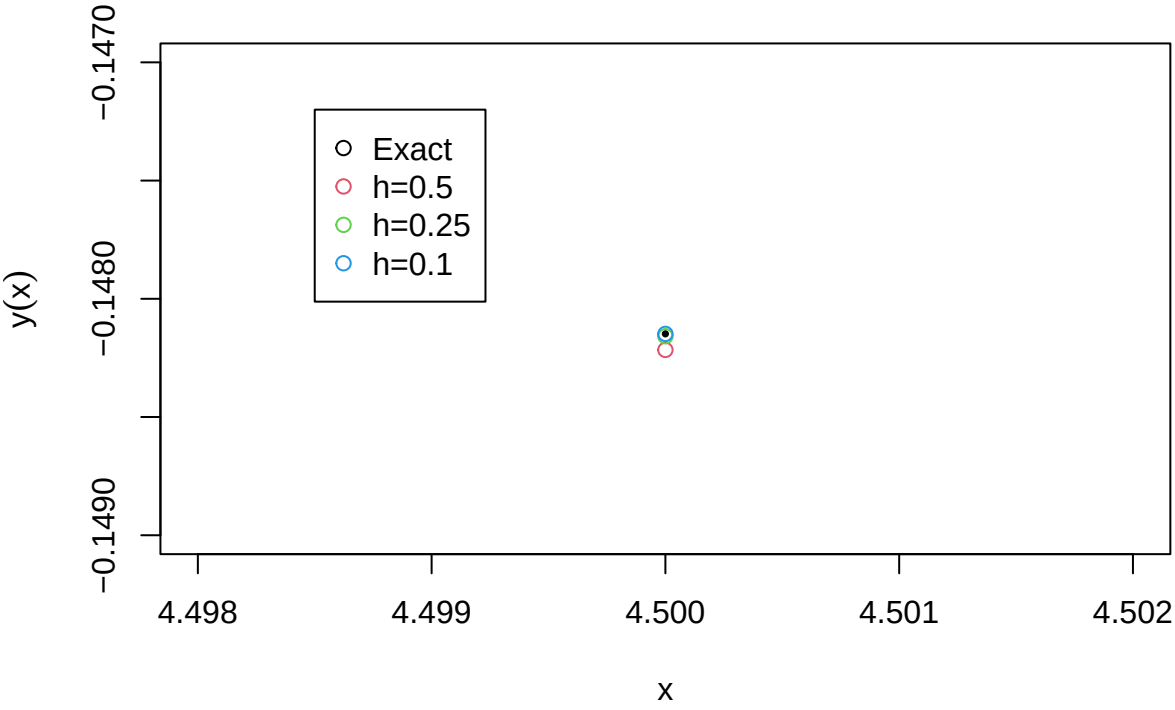
Details on the actual distance of the numerical solution from the exact one can be better appreciated when the plot is zoomed in, for example close to the elbow of the curve.

```
plot(xx,yy,type="l",xlim=c(4.4,5.1),ylim=c(-0.15,-0.09),
     xlab=expression(x),ylab=expression(y(x)))
points(ltmp1$t,ltmp1$y,type="b",pch=16,col=2,lty=2)
points(ltmp2$t,ltmp2$y,type="b",pch=17,col=3,lty=3)
points(ltmp3$t,ltmp3$y,type="b",pch=18,col=4,lty=4)
legend(4.4,-0.1,legend=c("Exact","h=0.5","h=0.25","h=0.1"),
       pch=c(-1,16,17,18),col=c(1,2,3,4),lty=c(1,2,3,4))
```



It is not a sign of poor accuracy that the lines connecting the different points stick out of the curve more evidently for the solution with $h = 0.5$, because these points are far apart. This is less evident, but still the case, for the solution with $h = 0.25$ and $h = 0.1$. But the points itself are relatively close to the correct value of the solution, as shown, for example, in the following plot around the value $x = 4.5$.

```
# Exact value at x=4.5
plot(4.5,1/(3*4.5-4.5^2),pch=16,cex=0.5,
     xlim=c(4.498,4.502),ylim=c(-0.149,-0.147),
     xlab=expression(x),ylab=expression(y(x)))
points(ltmp1$t,ltmp1$y,pch=1,col=2,lty=2)
points(ltmp2$t,ltmp2$y,pch=1,col=3,lty=3)
points(ltmp3$t,ltmp3$y,pch=1,col=4,lty=4)
legend(4.4985,-0.1472,legend=c("Exact","h=0.5","h=0.25","h=0.1"),
       pch=c(1,1,1,1),col=c(1,2,3,4))
```



All points, even the one corresponding to $h = 0.5$, are close to the exact solution. That gives a measure of the accuracy of the RK4 solver.

The number of steps required by a method can be simply extracted as the size of any of the arrays returned by the solvers. For example:

```
# Number of steps when h=0.5
print(length(ltmp1$t))
#> [1] 13

# Number of steps when h=0.25
print(length(ltmp2$t))
#> [1] 25

# Number of steps when h=0.1
print(length(ltmp3$t))
#> [1] 61
```

1.3 Exercise 03

In a closed environment, a biological population may grow quickly at first, but then slow down as resources become limited. This process can be modelled by the *logistic growth equation*:

$$\frac{dn}{dt} = rn\left(1 - \frac{n}{K}\right),$$

where:

- $n(t)$ is the population size at time t,

- $r > 0$ is the intrinsic growth rate,
- $K > 0$ is the *carrying capacity* (the maximum population that the environment can sustain).

The equation assumes that the population grows approximately exponentially when small, $n \ll K$, but that growth slows and eventually stops as n approaches K.

In this exercise, we consider the case:

- Growth rate: $r = 0.5$,
- Carrying capacity: $K = 1000$,
- Initial population: $n(0) = 50$.

The analytical solution to the logistic equation is:

$$n(t) = \frac{K n_0 e^{rt}}{K + n_0(e^{rt} - 1)}.$$

1. Use the analytical solution to compute the exact population values at $t = 5$, $t = 10$, and $t = 20$.
2. Implement Euler's method to solve the equation numerically on the interval $t \in [0, 20]$ using step size $h = 1$. Compare your numerical results with the exact values from part 1.
3. Repeat the numerical solution using the Heun method and the classical Runge-Kutta method (RK4). Report and compare the results at $t = 5$, $t = 10$, and $t = 20$.
4. What do your numerical methods predict for large t? Does the population approach the expected limiting value K?

SOLUTION

1. It is worth defining the analytical solution as an R function and then use it to calculate the values at the suggested times.

```
# Solution of the logistic eqn
nsol <- function(t,r,K,n0) {
  nn <- K*n0*exp(r*t)/(K+n0*(exp(r*t)-1))

  return(nn)
}

# Given parameters
r <- 0.5
K <- 1000
n0 <- 50

# Growth at t=5, 10, 20
stmp <- sprintf("Population at t=5: %8.0f\n",nsol(5,r,K,n0))
cat(stmp)
#> Population at t=5:      391
stmp <- sprintf("Population at t=10: %8.0f\n",nsol(10,r,K,n0))
cat(stmp)
#> Population at t=10:      887
stmp <- sprintf("Population at t=20: %8.0f\n",nsol(20,r,K,n0))
cat(stmp)
#> Population at t=20:      999
```

2.

```r
# Gradient
f <- function(t,n,r,K) {
  ff <- r*n*(1-n/K)

  return(ff)
}

# Interval
t0 <- 0
tf <- 20

# Initial conditions
n0 <- 50

# Step size
h <- 1

# Euler method
ltmp <- EulerODE(f=f,t0=t0,tf=tf,y0=n0,h=h,r=r,K=K)

# Output all values as a table
tble <- data.frame(t=ltmp$t,n=ltmp$y)
print(tble)
#>     t        n
#> 1   0  50.0000
#> 2   1  73.7500
#> 3   2 107.9055
#> 4   3 156.0364
#> 5   4 221.8809
#> 6   5 308.2058
#> 7   6 414.8133
#> 8   7 536.1849
#> 9   8 660.5303
#> 10  9 772.6453
#> 11 10 860.4776
#> 12 11 920.5055
#> 13 12 957.0931
#> 14 13 977.6260
#> 15 14 988.5627
#> 16 15 994.2160
#> 17 16 997.0912
#> 18 17 998.5414
#> 19 18 999.2696
#> 20 19 999.6346
#> 21 20 999.8172
```

The solution at $t = 5, 10, 20$ is not exactly the correct one, which is not surprising, given the nature of the approximation. Note how the parameters `r` and `K` could be passed by `EulerODE` to the gradient function because *ellipses*, `...`, were appropriately included in the `comphy` code.

3. The same calculations can be repeated using the Heun and 4th-order Runge-Kutta methods.

```r
# Gradient already defined

# Heun
```

```
ltmpH <- HeunODE(f=f,t0=t0,tf=tf,y0=n0,h=h,r=r,K=K)

# 4th order Runge-Kutta
ltmpR <- RK4ODE(f=f,t0=t0,tf=tf,y0=n0,h=h,r=r,K=K)

# Comparison (Here ltmpH$t=ltmpR$t)
tble <- data.frame(t=ltmpR$t,nH=ltmpH$y,nR=ltmpR$y)
print(tble)
#>     t        nH        nR
#> 1   0  50.00000  50.00000
#> 2   1  78.95273  79.83664
#> 3   2 122.63638 125.13478
#> 4   3 185.86194 190.80878
#> 5   4 271.97315 279.92730
#> 6   5 379.81223 390.58414
#> 7   6 501.19950 513.77727
#> 8   7 622.21761 635.31901
#> 9   8 729.11342 741.74559
#> 10  9 814.11600 825.62985
#> 11 10 876.46637 886.43212
#> 12 11 919.67992 927.88028
#> 13 12 948.52296 954.96747
#> 14 13 967.31250 972.18497
#> 15 14 979.36554 982.93679
#> 16 15 987.02252 989.57690
#> 17 16 991.85722 993.64965
#> 18 17 994.89826 996.13719
#> 19 18 996.80652 997.65261
#> 20 19 998.00216 998.57436
#> 21 20 998.75060 999.13448
```

The values produced with the Heun and 4th-order Runge-Kutta methods are closer to the correct values than those produced with the Euler method.

4. Clearly, the analytical expression $n(t)$ goes to K when $t \to \infty$. Let us check this is also the case with the numerical solution provided by, say, `RK4ODE`. It will suffice to extend tf to a large value, say 1000.

```
# Just change tf
tf <- 1000

# Apply RK4
ltmpR <- RK4ODE(f,t0,tf,n0,h,r=r,K=K)

# Only show last 10 values
# (992 to 1001 - because the first value is 0)
tble <- data.frame(t=ltmpR$t[992:1001],
                   nR=ltmpR$y[992:1001])
print(tble)
#>       t   nR
#> 1   991 1000
#> 2   992 1000
#> 3   993 1000
#> 4   994 1000
#> 5   995 1000
#> 6   996 1000
```

```
#> 7   997 1000
#> 8   998 1000
#> 9   999 1000
#> 10 1000 1000
```

The numerical algorithm behaves well because it displays the asymptotic value. In fact, this is reached quite soon during growth.

1.4 Exercise 04

Consider the classical *Lotka–Volterra system*, modelling the interaction between a prey population $x(t)$ and a predator population $y(t)$. The governing equations are:

$$\mathrm{d}x/\mathrm{d}t = \alpha x - \beta xy$$
$$\mathrm{d}y/\mathrm{d}t = \delta xy - \gamma y$$

The meaning of the variables and parameters is as follows:

- $x(t)$: number of *prey* (e.g., rabbits) at time t
- $y(t)$: number of *predators* (e.g., foxes) at time t
- α: natural *growth rate of prey* in the absence of predators
- β: *predation rate coefficient* (how often predators encounter and eat prey)
- δ: *growth rate of predators* per prey eaten
- γ: *natural death rate of predators* in the absence of prey

The task to be carried out in this exercise are:

1. Implement a gradient function `f(t,u)` where `u=c(x,y)` and `f` returns the derivatives $\mathrm{d}x/\mathrm{d}t$ and $\mathrm{d}y/\mathrm{d}t$.
2. Use the `RK4ODE` solver to solve the system numerically over the interval $t \in [0, 30]$, where the parameters are
$$\alpha = 1.0, \quad \beta = 0.1, \quad \delta = 0.075, \quad \gamma = 1.5,$$
and the initial conditions are
$$x(0) = 40, \quad y(0) = 9.$$
Use a step size $h = 0.1$.
3. Plot both populations as functions of time.
4. Plot the *phase portrait*, i.e., a plot of $y(t)$ versus $x(t)$.
5. Try changing the parameters and observe whether the solution remains periodic or tends to a steady state.

SOLUTION

1. It is convenient to implement the gradient with generic parameters, as this can be then used for as many models as needed.

```
# Gradient (two ODEs)
f <- function(t,u,alpha,beta,gamma,delta) {
  dx <- alpha*u[1]-beta*u[1]*u[2]
  dy <- delta*u[1]*u[2]-gamma*u[2]

  return(c(dx,dy))
}
```

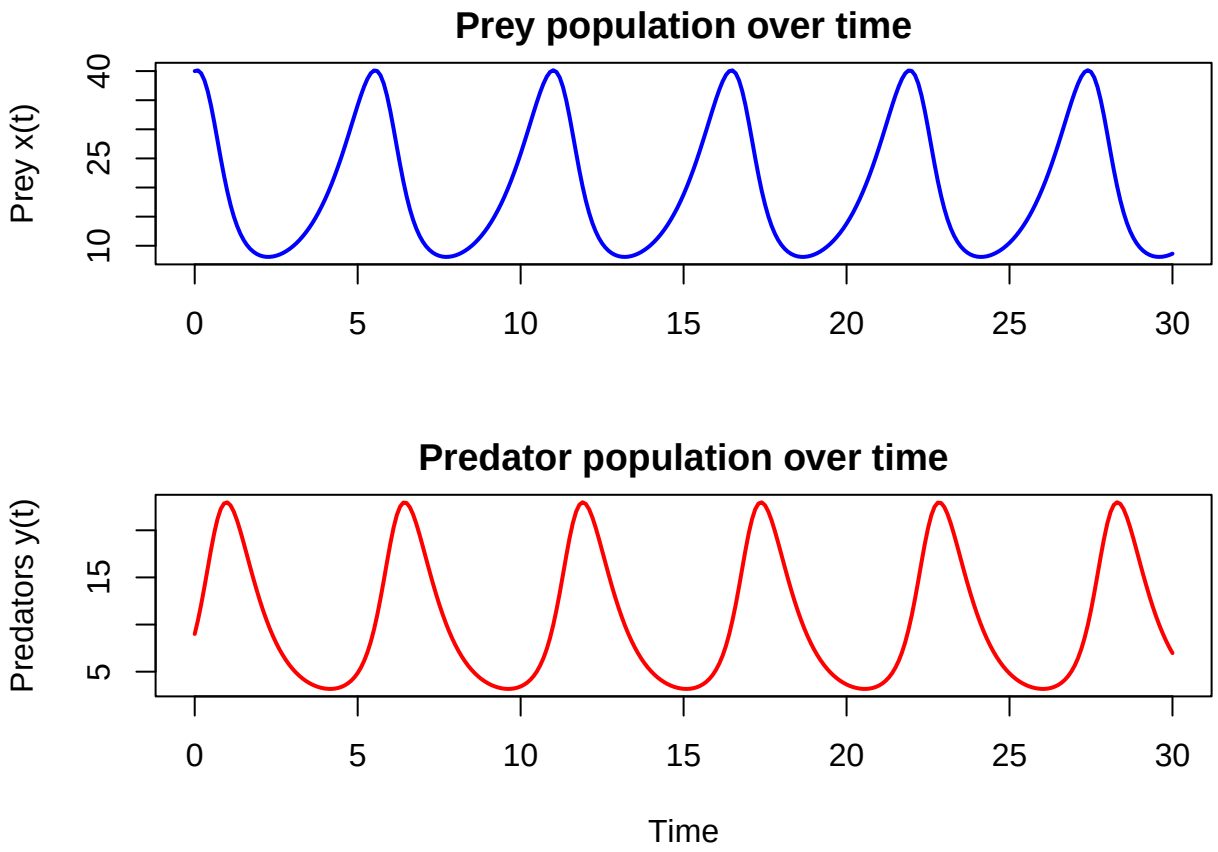
```r
# Test
print(f(t=0,u=c(40,9),alpha=1,beta=0.1,gamma=1.5,delta=0.075))
#> [1]  4.0 13.5
```

2. Solution using `RK4ODE`.

```r
# Gradient already prepared

# Use assigned values of parameters
alpha <- 1.0
beta <- 0.1
gamma <- 1.5
delta <- 0.075

# Solution interval
t0 <- 0
tf <- 30

# Initial conditions
u0 <- c(40,9)

# Step size
h <- 0.1

# 4th order Runge-Kutta
ltmp <- RK4ODE(f,t0,tf,u0,h,
               alpha=alpha,beta=beta,gamma=gamma,delta=delta)

# Find out number of steps
n <- length(ltmp$t)-1

# Check initial and final values
print(ltmp$t[1])
#> [1] 0
print(ltmp$t[n+1])
#> [1] 30
print(ltmp$y[1,])
#> [1] 40  9
print(ltmp$y[n+1,])
#> [1] 8.632400 6.971652
```

3. Let's plot the populations one on top of the other, for better comparison.

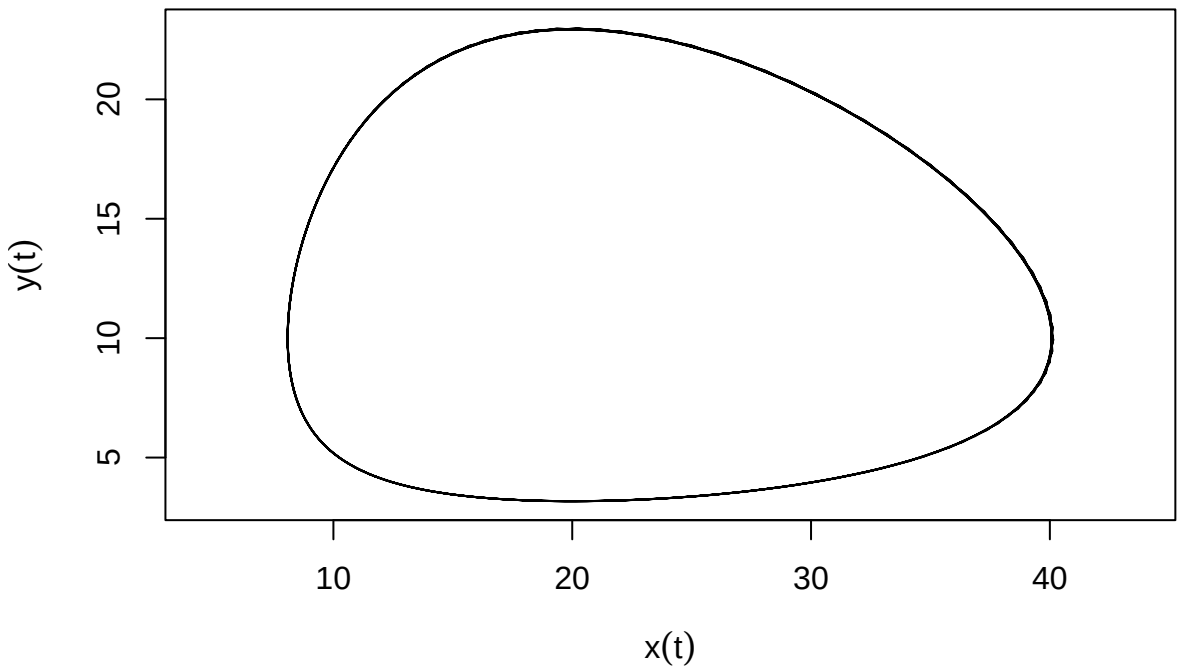
```r
# Two stacked plots
par(mfrow=c(2, 1),mar=c(4,4,2,1))

# Prey plot (at the top)
plot(ltmp$t,ltmp$y[,1],type="l",col="blue",lwd=2,
     xlab="",ylab="Prey x(t)",
     main="Prey population over time")

# Predator plot (at the bottom)
plot(ltmp$t,ltmp$y[,2],type="l",col="red",lwd=2,
     xlab="Time",ylab="Predators y(t)",
     main="Predator population over time")
```



We can admire in the plots above the typical dynamics of the predator-prey model. First, abundance of preys happens because there were not many predators to hunt and kill preys. But then prey abundance means that more food is available for predators and they can prosper. But too many predators means competition and less food (preys) available. So predators starve and begin to die. Then preys can breed more easily, their number starts increasing, and the cycle repeats.

4. The phase portrait is readily done by plotting $x(t)$ on the horizontal axis and $y(t)$ on the vertical one. These quantities were calculated in the previous snippet.

```
# Back to one graphical window
par(mfrow=c(1,1),mar=c(5.1,4.1,4.1,2.1))

# Phase portrait
plot(ltmp$y[,1],ltmp$y[,2],type="l",asp=1,
     xlab=expression(x(t)),ylab=expression(y(t)))
```

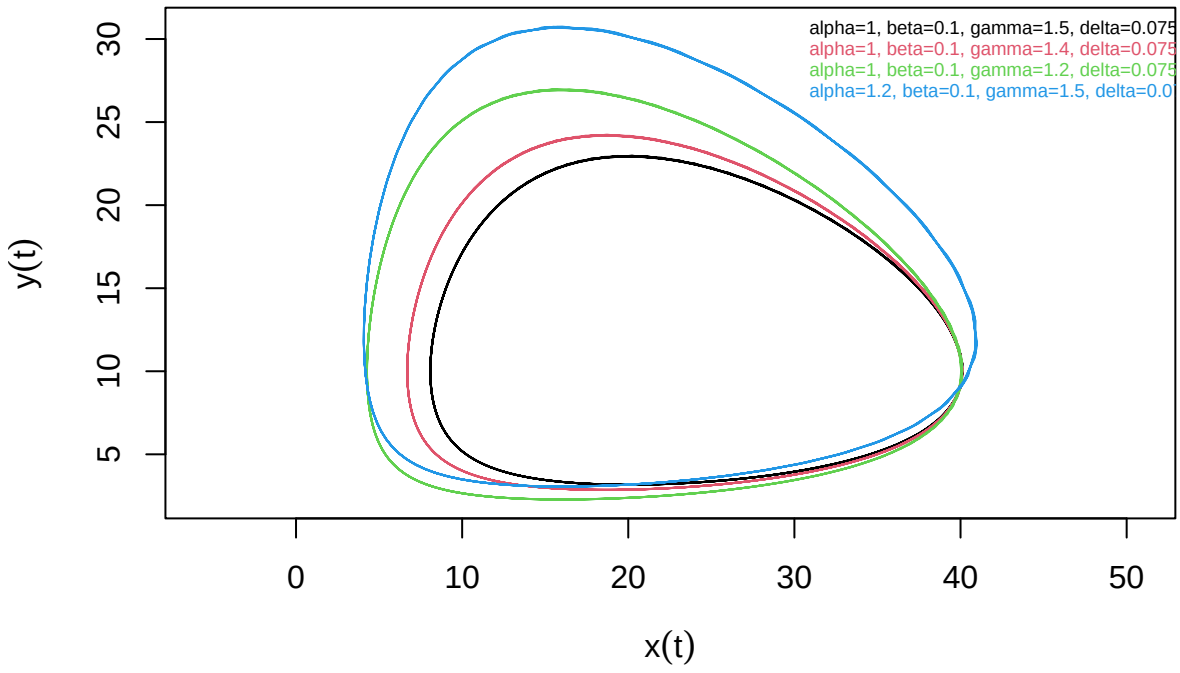


The closed orbit observed in the above phase portrait is the signature of a cyclic dynamics.

5. We can try and modify parameters slightly. The phase portrait, in our case will still contain closed orbits, but different from each other.

```
# First choice of parameters
ltmp0 <- ltmp

# Second choice of parameters
gamma <- 1.4

# Recalculate dynamics
ltmp1 <- RK4ODE(f,t0,tf,u0,h,
                alpha=alpha,beta=beta,gamma=gamma,delta=delta)

# Third choice of parameters
gamma <- 1.2

# Recalculate dynamics
ltmp2 <- RK4ODE(f,t0,tf,u0,h,
                alpha=alpha,beta=beta,gamma=gamma,delta=delta)

# Fourth choice of parameters
alpha <- 1.2

# Recalculate dynamics
ltmp3 <- RK4ODE(f,t0,tf,u0,h,
```

```
                alpha=alpha,beta=beta,gamma=gamma,delta=delta)

# Find extents of plot
rangeX <- range(ltmp0$y[,1],ltmp1$y[,1],ltmp2$y[,1],ltmp3$y[,1])
rangeY <- range(ltmp0$y[,2],ltmp1$y[,2],ltmp2$y[,2],ltmp3$y[,2])

# Phase portraits
plot(ltmp0$y[,1],ltmp0$y[,2],type="l",asp=1,
     xlab=expression(x(t)),ylab=expression(y(t)),
     xlim=rangeX,ylim=rangeY)
lines(ltmp1$y[,1],ltmp1$y[,2],col=2)
lines(ltmp2$y[,1],ltmp2$y[,2],col=3)
lines(ltmp3$y[,1],ltmp3$y[,2],col=4)

# Legend
legend(29,32,legend=c("alpha=1, beta=0.1, gamma=1.5, delta=0.075",
                      "alpha=1, beta=0.1, gamma=1.4, delta=0.075",
                      "alpha=1, beta=0.1, gamma=1.2, delta=0.075",
                      "alpha=1.2, beta=0.1, gamma=1.5, delta=0.075"),
       text.col=c(1,2,3,4),bty="n",cex=0.55)
```



The choice of values is obviously arbitrary. Changing these systematically can help connecting them with how they determine the dynamics of the model.

1.5 Exercise 05

In mechanics and engineering, the third derivative of position with respect to time is known as *jerk*, and it plays an important role in the modelling of motion where smoothness or mechanical stress is a concern. For instance, jerk arises in motion planning for vehicles or robotic arms, where sudden changes in acceleration must be avoided.

Consider the following third-order differential equation:

$$\frac{\mathrm{d}^3x}{\mathrm{d}t^3} + 4\frac{\mathrm{d}x}{\mathrm{d}t} + x = 0,$$

with initial conditions:

$$x(0) = 1, \quad \frac{\mathrm{d}x(0)}{\mathrm{d}t} = 0, \quad \frac{\mathrm{d}^2x(0)}{\mathrm{d}t^2} = 0.$$

In this exercise, you must:

1. Introduce new variables
$$y_1 = x, \quad y_2 = \frac{\mathrm{d}x}{\mathrm{d}t}, \quad y_3 = \frac{\mathrm{d}^2x}{\mathrm{d}t^2},$$
and rewrite the equation as a system of three coupled first-order differential equations.
2. Write down the corresponding initial conditions for the variables y_1, y_2, y_3.
3. Use a numerical method (e.g. Euler or Runge-Kutta of order 4) to solve the system on the interval $t \in [0, 10]$ with a step size of your choice.
4. Plot the numerical solution for position $x(t)$, velocity $\mathrm{d}x/\mathrm{d}t$, and acceleration $\mathrm{d}^2x/\mathrm{d}t^2$. Discuss the overall behaviour of the system.
5. What role does the jerk term (third derivative) appear to play in the motion over time?

SOLUTION

1. With the new suggested variables, the system is built in a cascading fashion. As $x = y_1$, the assignment

$$y_2 = \frac{\mathrm{d}x}{\mathrm{d}t}$$

becomes

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = y_2.$$

The assignment

$$y_3 = \frac{\mathrm{d}^2x}{\mathrm{d}t^2} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}x}{\mathrm{d}t}\right)$$

becomes

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = y_3.$$

The last equation of the system is found replacing all variables in the original ODE:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}^2x}{\mathrm{d}t^2}\right) + 4\frac{\mathrm{d}x}{\mathrm{d}t} + x = 0$$

$$\Downarrow$$

$$\frac{\mathrm{d}y_3}{\mathrm{d}t} + 4y_2 + y_1 = 0 \;\Rightarrow\; \frac{\mathrm{d}y_3}{\mathrm{d}t} = -y_1 - 4y_2.$$

The original third order ODE is thus transformed into the following system of first order ODEs:

$$\begin{cases} \mathrm{d}y_1/\mathrm{d}t = y_2 \\ \mathrm{d}y_2/\mathrm{d}t = y_3 \\ \mathrm{d}y_3/\mathrm{d}t = -y_1 - 4y_2. \end{cases}$$

2. The associated initial conditions are:

$$y_1(0) = 1, \quad y_2(0) = 0, \quad y_3(0) = 0.$$

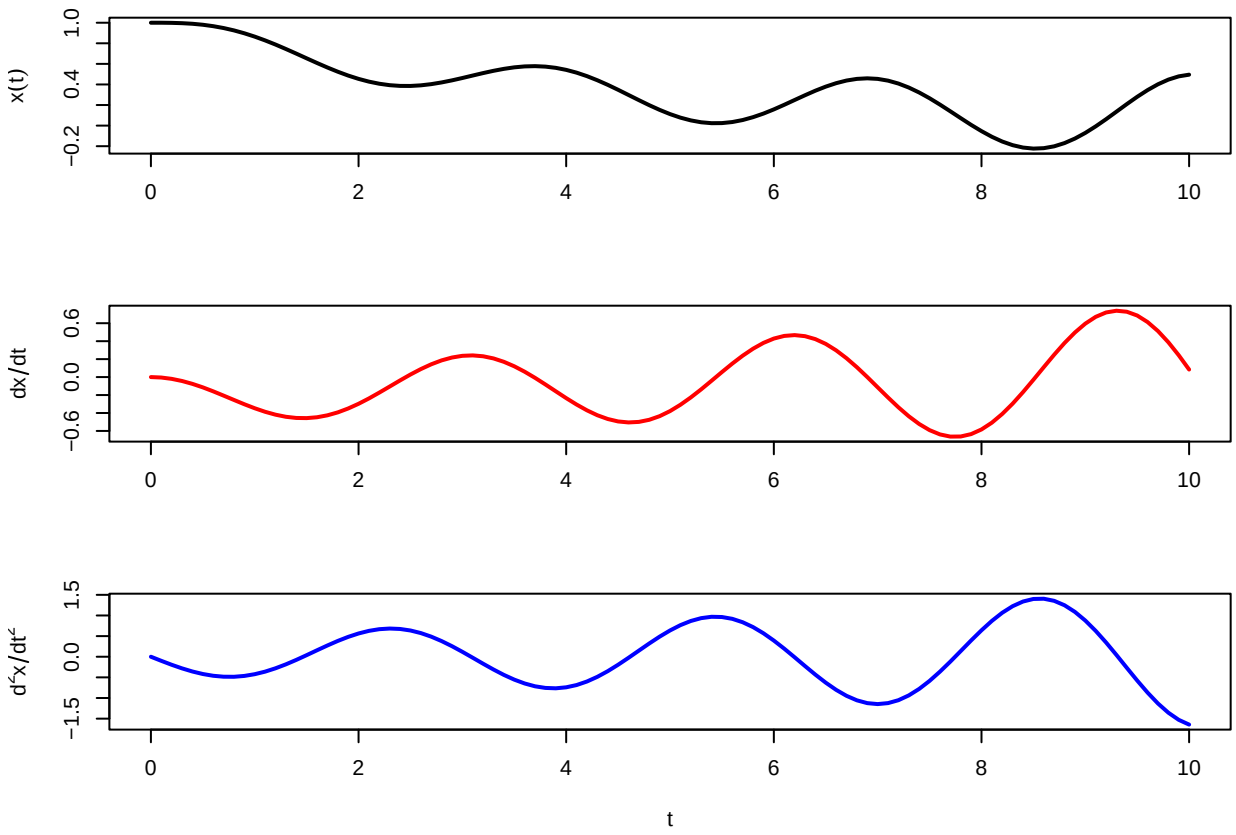
3. We apply the 4th-order Runge-Kutta method with step $h = 0.1$ to find the solutions for y_1, y_2, y_3 in the suggested range, $t \in [0, 10]$.

```
# Gradient
f <- function(t,y) {
  dy1 <- y[2]
  dy2 <- y[3]
  dy3 <- -y[1]-4*y[2]

  return(c(dy1,dy2,dy3))
}

# Solution interval
t0 <- 0
tf <- 10

# Initial conditions
y0 <- c(1,0,0)

# Step size
h <- 0.1

# Solver
ltmp <- RK4ODE(f,t0,tf,y0,h)

# Check
print(ltmp$t[1])
#> [1] 0
print(ltmp$y[1,])
#> [1] 1 0 0
```

The application seems to have been written in the right way. Always take care in counting the components of the initial conditions (three in this case).

4. Let's plot the three components on top of each other, for comparison.

```
# Three stacked plots
par(mfrow=c(3, 1),mar=c(4,4,2,1))

# x plot (at the top)
plot(ltmp$t,ltmp$y[,1],type="l",col="black",lwd=2,
     xlab="",ylab=expression(x(t)))
plot(ltmp$t,ltmp$y[,2],type="l",col="red",lwd=2,
     xlab="",ylab=expression(dx/dt))
plot(ltmp$t,ltmp$y[,3],type="l",col="blue",lwd=2,
     xlab=expression(t),ylab=expression(d^2 * x / dt^2))
```



The numerical solutions for position $x(t)$, velocity $\mathrm{d}x/\mathrm{d}dt$, and acceleration $\mathrm{d}^2x/\mathrm{d}t^2$ all exhibit a wavy, oscillatory behaviour over the interval $t \in [0, 10]$. This is consistent with the structure of the differential equation:

$$\frac{\mathrm{d}^3x}{\mathrm{d}t^3} + 4\frac{\mathrm{d}x}{\mathrm{d}t} + x = 0,$$

which combines inertial effects with a form of damping acting on the velocity.

- The *position* $x(t)$ oscillates around zero, with smooth curves and alternating concavity, indicating repeated acceleration and deceleration.
- The *velocity* $\mathrm{d}x/\mathrm{d}t$ also oscillates, with a phase shift relative to position. Its peaks and troughs align with points where position curvature changes.
- The *acceleration* $\mathrm{d}^2x/\mathrm{d}t^2$ shows sharper oscillations, reflecting the rapid changes in the rate of velocity due to the influence of jerk.

The plots reveal a system that does not settle to equilibrium or diverge, but instead exhibits sustained oscillations. The interplay of the damping and jerk terms produces motion that is smooth but highly dynamic.

5. The third derivative term, known as *jerk*, plays a critical role in shaping the system's dynamics. While a second-order equation would describe motion based solely on position and velocity, the presence of jerk introduces an additional layer of inertia, the system now responds not only to forces and velocities, but also to how rapidly the acceleration is changing. This manifests as:

- Smoother, more continuous transitions in motion,
- Greater resistance to abrupt changes in acceleration,
- More complex oscillatory behaviour, involving higher-order phase interactions between $x(t)$, $\mathrm{d}x/\mathrm{d}t$, and $\mathrm{d}^2x/\mathrm{d}t^2$.

Physically, such models are used in systems where motion needs to be smooth and controlled, for example, in robotics, vehicle suspension systems, or precision mechanisms. The oscillations seen in the plots reflect the fact that the system tries to damp motion through velocity, but is slowed down in how quickly it can adjust due to the influence of jerk.

2 Exercises on BVPs

2.1 Exercise 06

Consider the following linear BVP:

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2} - \frac{2}{x}\frac{\mathrm{d}y}{\mathrm{d}x} + y(x) = x, \qquad y(1) = 0,\ y(2) = 1.$$

1. Use BVPlinshoot2 to compute the solution on $[1, 2]$. Try with step sizes $h = 0.01$, $h = 0.005$, $h = 0.0025$ and record the values of $y(1.5)$.
2. Plot the solution for the three different step sizes, using different colours/line traits.

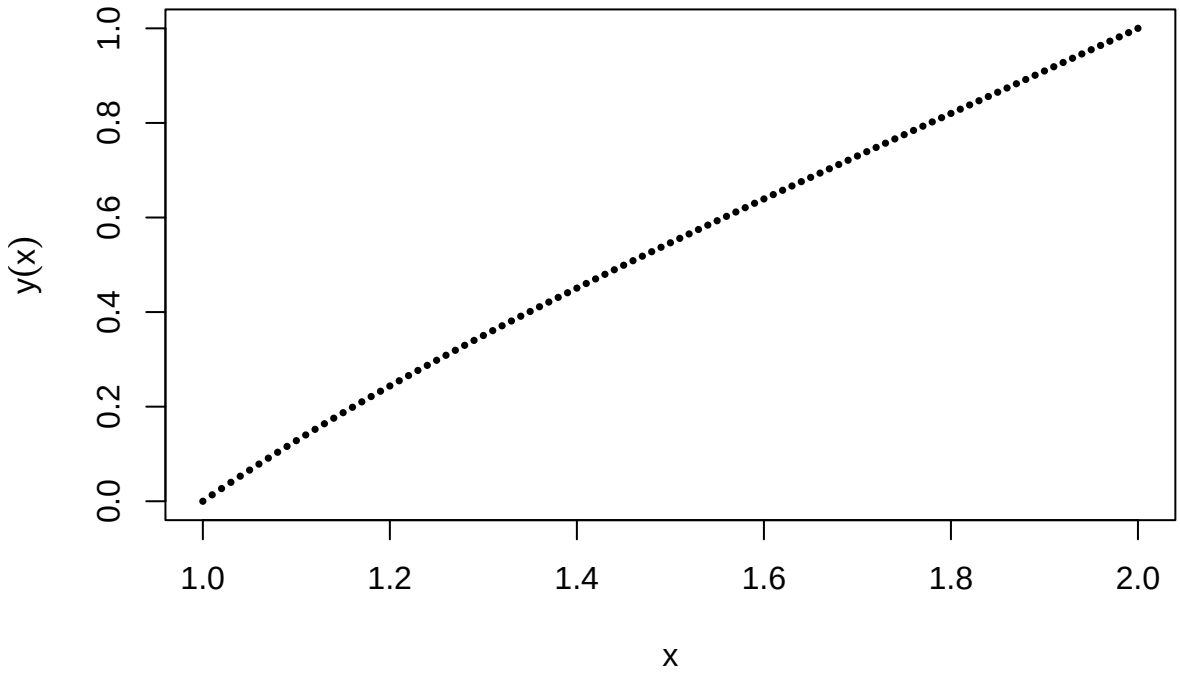
SOLUTION

1. The function $f(t, y, y')$ needed in `BVPlinshoot2` uses x rather than t:

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2} = f(x, y, y') = x - y + \frac{2}{x}y'.$$

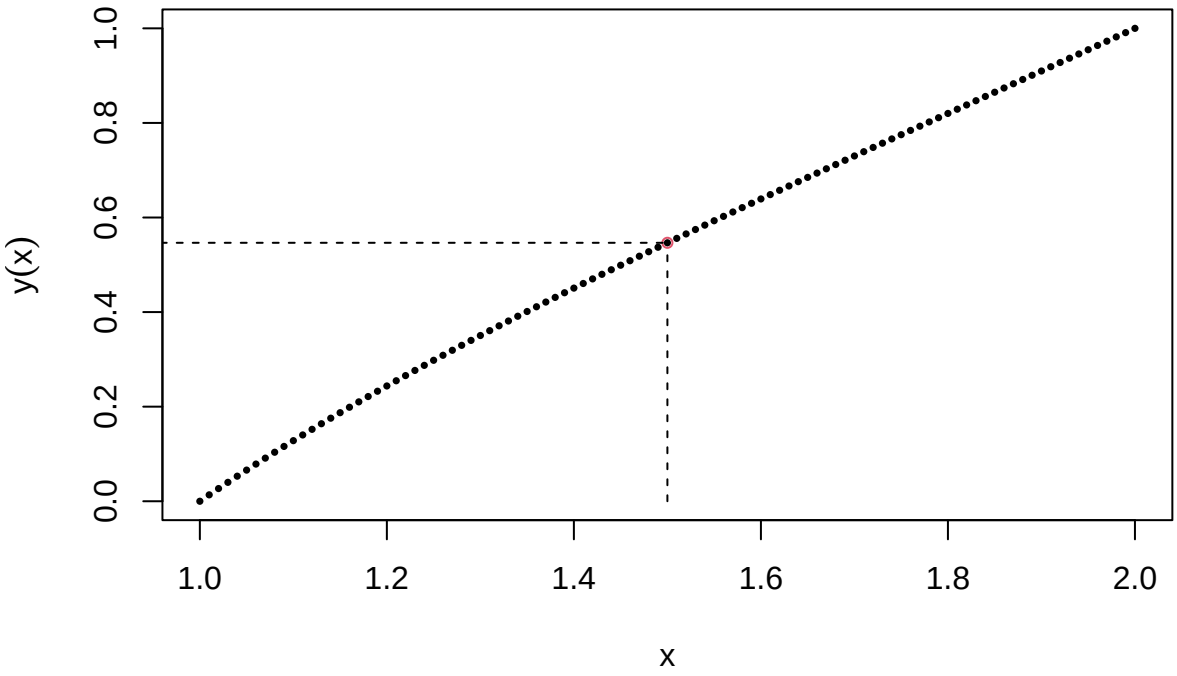
The values of x are away from 0, so we do not have to worry about the presence of $2/x$ in the equation. Let us try with $h = 0.01$ first.

```
# Make sure comphy is loaded in memory
require(comphy)

# Define "gradient" f(x,y,y')
f <- function(x,y,y1) {
  ff <- x-y-2*y1/x

  return(ff)
}

# Interval
x0 <- 1
xf <- 2

# BVPs
y0 <- 0
yf <- 1

# Step size
hA <- 0.01

# Solution. Linear shooting method
ltmpA <- BVPlinshoot2(f,x0,xf,y0,yf,hA)

# Plot
plot(ltmpA$t,ltmpA$y[,1],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y(x)))
```



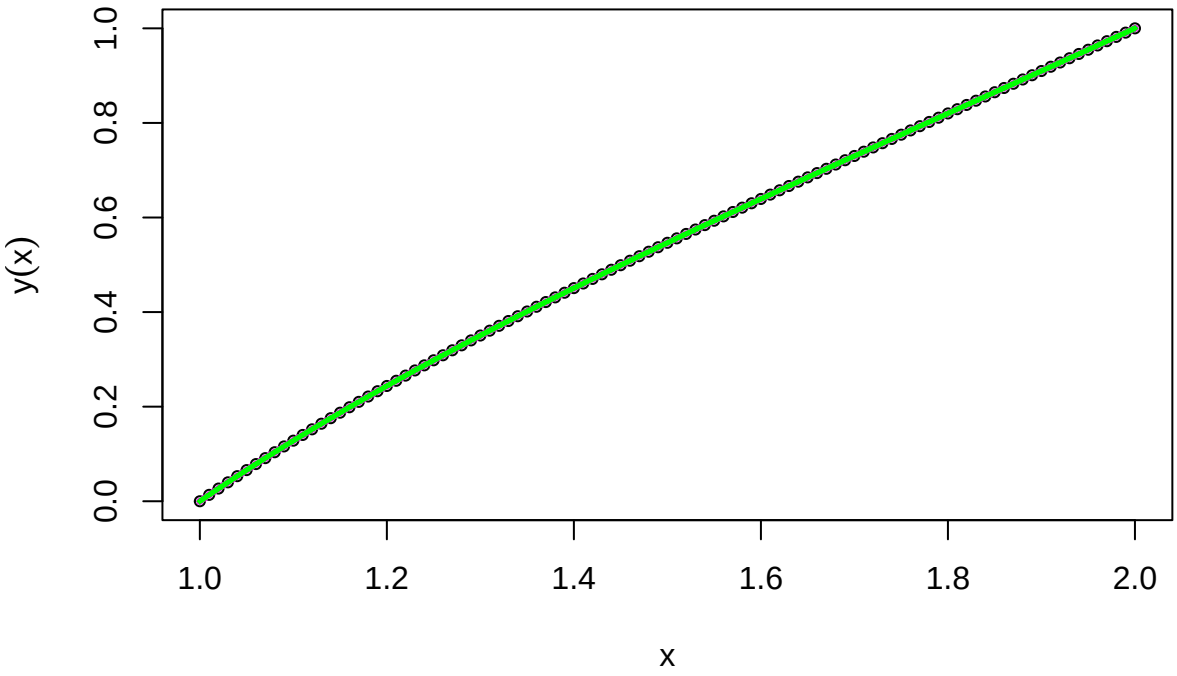
To extract the value at $x = 1.5$ we identify the position corresponding to 1.5 in the array `ltmpA$t`.

```
# Which index corresponds to x=1.5?
idxA <- which(abs(ltmpA$t-1.5) < 0.000001)
xA15 <- ltmpA$t[idxA]
print(idxA)
#> [1] 51
print(xA15)
#> [1] 1.5


# Value y(1.5) requested
yA15 <- ltmpA$y[idxA,1]
print(yA15) # y_combined is the name assigned to first column by BVPlinshoot2
#> y_combined
#>   0.546498

# Indicate point in the plot
plot(ltmpA$t,ltmpA$y[,1],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y(x)))
points(xA15,yA15,pch=1,col=2,cex=0.7)
segments(x0=xA15,x1=xA15,y0=0,y1=yA15,lty=2)
segments(x0=0,x1=xA15,y0=yA15,y1=yA15,lty=2)
```



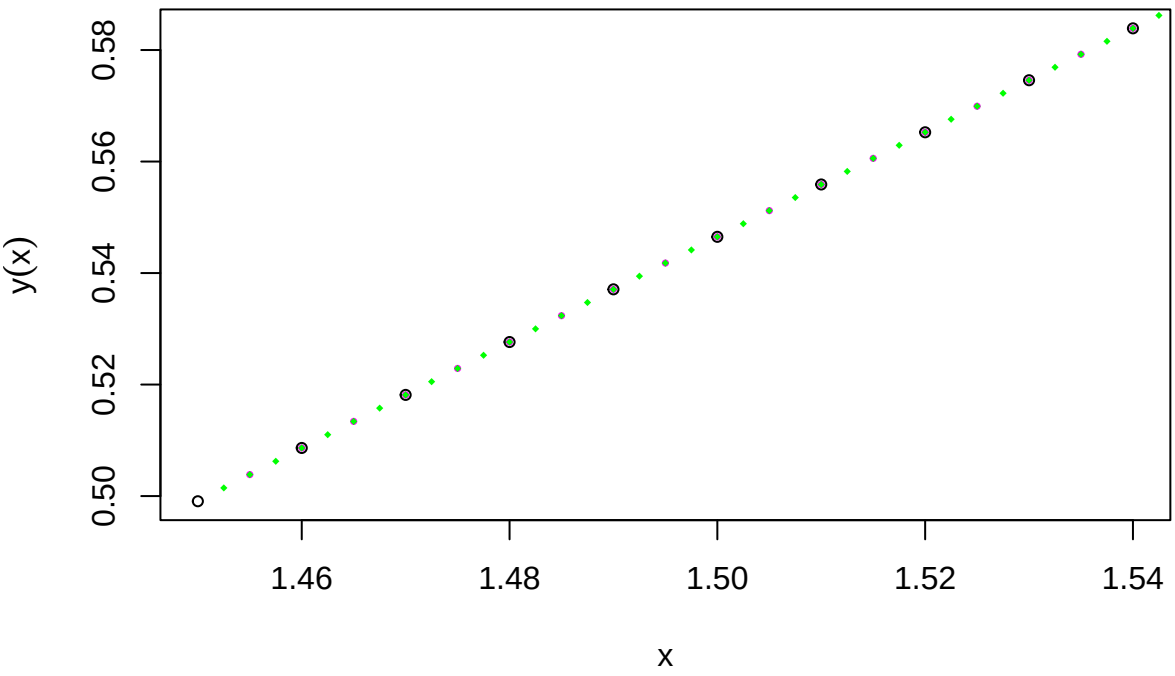
We can now repeat solutions for the other two step sizes and carry out a comparison of results.

```
# Step size 0.005
hB <- 0.005

# Solution. Linear shooting method
ltmpB <- BVPlinshoot2(f,x0,xf,y0,yf,hB)

# Step size 0.0025
hC <- 0.0025

# Solution. Linear shooting method
ltmpC <- BVPlinshoot2(f,x0,xf,y0,yf,hC)

# Comparison
plot(ltmpA$t,ltmpA$y[,1],type="b",pch=1,cex=0.7,
     xlab=expression(x),ylab=expression(y(x)))
points(ltmpB$t,ltmpB$y[,1],type="b",pch=16,cex=0.5,col="magenta")
points(ltmpC$t,ltmpC$y[,1],type="b",pch=18,cex=0.5,col="green")
```



It is difficult to see how solutions compare, although it is clear that they are very close with each other. We can *zoom in* by restricting the x range to the interval $[1.45, 1.55]$.

```
# Identify indices corresponding to x in [1.4,1.6]
jdxA <- which(ltmpA$t >= 1.45 & ltmpA$t <= 1.55)
jdxB <- which(ltmpB$t >= 1.45 & ltmpB$t <= 1.55)
jdxC <- which(ltmpC$t >= 1.45 & ltmpC$t <= 1.55)

# Plot to compare
plot(ltmpA$t[jdxA],ltmpA$y[jdxA,1],pch=1,cex=0.7,
     xlab=expression(x),ylab=expression(y(x)))
points(ltmpB$t[jdxB],ltmpB$y[jdxB,1],pch=16,cex=0.5,col="magenta")
points(ltmpC$t[jdxC],ltmpC$y[jdxC,1],pch=18,cex=0.5,col="green")
```



It is clear that the numerical solutions are very close with each other, which means that already $h = 0.01$ is good enough to approximate the solution.

Let us conclude with comparing the values at $x = 1.5$.

```
# Which index corresponds to x=1.5?
idxB <- which(abs(ltmpB$t-1.5) < 0.000001)
xB15 <- ltmpB$t[idxB]
print(idxB)
#> [1] 101
print(xB15)
#> [1] 1.5
idxC <- which(abs(ltmpC$t-1.5) < 0.000001)
xC15 <- ltmpC$t[idxC]
print(idxC)
#> [1] 201
print(xC15)
#> [1] 1.5


# Value y(1.5) requested
yB15 <- ltmpB$y[idxB,1]
print(yB15)
#> y_combined
#>   0.546498
yC15 <- ltmpC$y[idxC,1]
print(yC15)
```

```
#> y_combined
#>   0.546498

# Comparison
cmp <- cbind(yA15,yB15,yC15)
print(cmp)
#>                yA15     yB15     yC15
#> y_combined 0.546498 0.546498 0.546498
```

The values at $x = 1.5$ coincide, which corresponds to the experience we have had with the accuracy of the linear shooting method in the text book. In fact ,even though we cannot compare the numerical results with the analytic solution in this case, we can calculate the *mean local error* (see main text book) between the numerical solutions with different setp sizes. Care must be taken, though, to match indices as each step size correspond to a different number of points.

```
# Tolerance for matching points
tol <- 1e-12

# Mean local error between A and B
# Matching points. Row i, Column j equal TRUE means there's a match
M <- abs(outer(ltmpA$t,ltmpB$t,`-`)) <= tol

# Transform in index pairs. Matrix with two columns: (i "matches" j)
pairs <- which(M,arr.ind=TRUE)

# Use columns
MLerrorAB <- mean(abs(ltmpA$y[pairs[,1],1]-ltmpB$y[pairs[,2],1]))
print(MLerrorAB)
#> [1] 5.226654e-11

# The result should correspond with the difference between O(h^5)
OA <- hA^5
OB <- hB^5
DAB <- abs(OA-OB)
print(DAB)
#> [1] 9.6875e-11
```

As we can see we have an agreement in orders of magnitude. The student can repeat the same reasoning for the other two comparisons (between A and C and between B and C).

2.2 Exercise 07

Solve the following BVP analytically,

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = -e^x, \qquad y(0) = 0,\ y(1) = 0,$$

and compare your solution with the numerical solution obtained with the linear shooting method. Do errors behave according to expectation?

SOLUTION

This simple ODE can be integrated twice to yield

$$y(x) = -e^x + k_1 x + k_2,$$

where k_1, k_2 are integration constants to be found using the boundary conditions. This is readily done:

$$y(0) = 0 \ \Leftrightarrow\ -1 + k_2 = 0 \ \Rightarrow\ k_2 = 1,$$

$$y(1) = 0 \;\Leftrightarrow\; -e + k_1 + 1 = 0 \;\Rightarrow\; k_1 = e - 1.$$

The analytic solution is therefore:

$$y(x) = 1 + (e-1)x - e^x.$$

We will use this solution to compare the accuracy of the numerical solution.

Let us use a step size $h = 0.01$ in the linear shooting method. This should correspond to an accuracy $\mathrm{O}(h^5) = 10^{-10}$.

```
# Define "gradient" f(x,y,y')
f <- function(x,y,y1) {
  ff <- -exp(x)

  return(ff)
}

# Interval
x0 <- 0
xf <- 1

# BVPs
y0 <- 0
yf <- 0

# Step size
h <- 0.01

# Solution. Linear shooting method
ltmp <- BVPlinshoot2(f,x0,xf,y0,yf,h)

# Analytic solution at same grid points as the numerical solution
xx <- ltmp$t
yy <- 1+(exp(1)-1)*xx-exp(xx)

# Compare solutions using the mean local error
MLerror <- mean(abs(ltmp$y[,1]-yy))
print(MLerror)
#> [1] 1.452272e-12
```

The mean local error does not exceed expectation, which means that the numerical solution is very close to the analytic one.

2.3 Exercise 08

A rod of length $L = 10$ m has its ends held at temperatures

$$T(0) = T_A = 300 \text{ K}, \qquad T(L) = T_B = 350 \text{ K}.$$

Assume steady one–dimensional conduction with

$$\kappa(T) = 1 + \alpha T, \qquad \alpha = 10^{-3} \text{ K}^{-1},$$

where κ is the variable *thermal conductivity* of the rod (measured in $\mathrm{Wm^{-1}K^{-1}}$) and where no internal heat sources are present. The steady state equation is

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(\kappa(T)\,\frac{\mathrm{d}T}{\mathrm{d}x}\right) = 0,$$

which, written in the form $\mathrm{d}^2T/\mathrm{d}x^2 = f(x, T, T')$, becomes

$$\frac{\mathrm{d}^2T}{\mathrm{d}x^2} = -\frac{\alpha}{1+\alpha T}\left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)^2.$$

1. Define $f(x, T, T') = -\alpha/(1+\alpha T)\,(T')^2$ and solve on $[0, 10]$ with `BVPshoot2`, using $T_A = 300$, $T_B = 350$.
2. Compute and report the numerical value of $T(5)$.
3. Produce a line plot of $T(x)$ on $[0, 10]$.
4. Repeat with $\alpha = 0$ (constant conductivity). Plot both solutions together and comment briefly on how the temperature–dependent conductivity bends the profile away from the straight line.

SOLUTION

1,2.

```
# Define "gradient" f(x,T,T')
f <- function(x,T,T1,alpha) {
  ff <- -alpha/(1+alpha*T)*T1^2

  return(ff)
}

# Interval
x0 <- 0
xf <- 10

# BVPs
T0 <- 300
Tf <- 350

# Step size
h <- 0.01

# Solution. Nonlinear shooting method
ltmpN <- BVPshoot2(f,x0,xf,T0,Tf,h,alpha=0.001)

# Value of T in the middle of the rod
idxN <- which(abs(ltmpN$t-5) < 0.000001)
x5N <- ltmpN$t[idxN]
T5N <- ltmpN$y[idxN,1]
print(cbind(idxN,x5N,T5N))
#>      idxN x5N      T5N
#> [1,]  501   5 325.2358
```

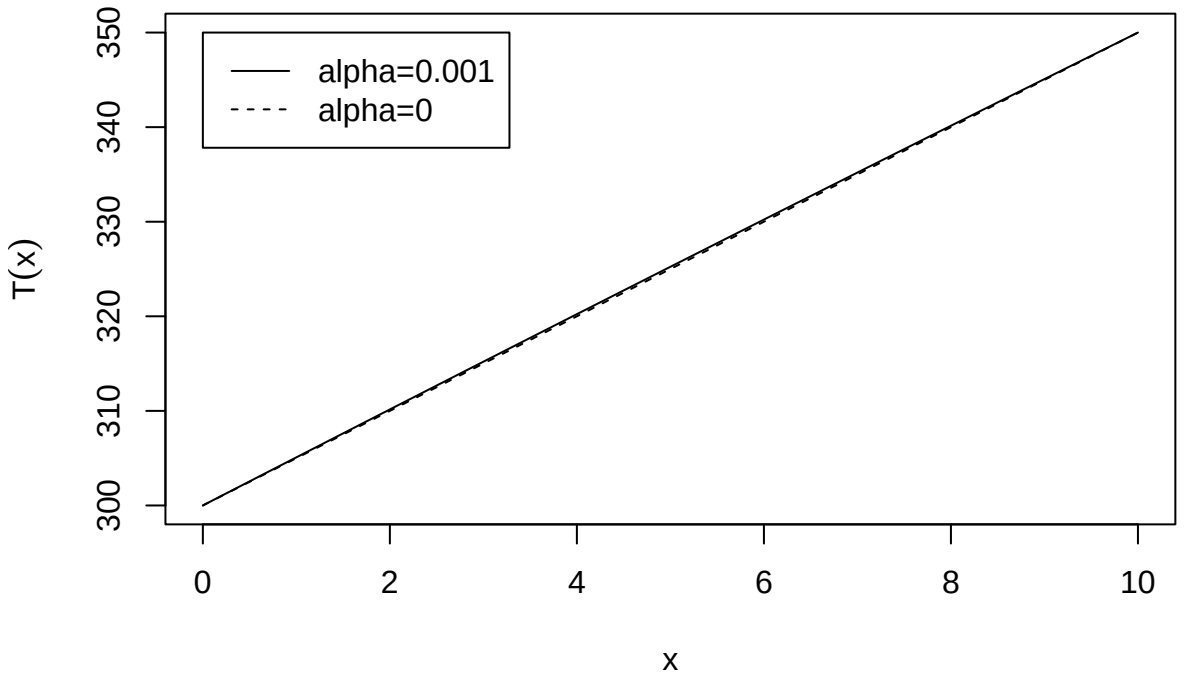
The value of the temperature in the steady state is slightly more than half way the temperature at the two extremes. It should be exactly equal to 325 if $\kappa(T) = 1$ ($\alpha = 0$):

```
# Solution. Nonlinear shooting method
ltmp0 <- BVPshoot2(f,x0,xf,T0,Tf,h,alpha=0)

# Value of T in the middle of the rod
idx0 <- which(abs(ltmp0$t-5) < 0.000001)
x50 <- ltmp0$t[idx0]
T50 <- ltmp0$y[idx0,1]
print(cbind(idx0,x50,T50))
```

```
#>      idx0 x50 T50
#> [1,]  501   5 325
```

3,4. Let us compare the two solutions graphically.

```
# First plot alpha=0.001
plot(ltmpN$t,ltmpN$y[,1],type="l",
     xlab=expression(x),ylab=expression(T(x)))

# Second plot alpha=0
points(ltmp0$t,ltmp0$y[,1],type="l",lty=2)

# Legend
legend(x=0,y=350,legend=c("alpha=0.001","alpha=0"),lty=c(1,2))
```



There seems to be very little difference between the two curves. In fact, α is nudging the temperature profile in a minimal way, i.e. the profile is not significantly sensible to changes in α. We can reason quantitatively on this. First of all, the ODE states that a derivative is equal to zero; this means that its argument is a constant, say A:

$$\frac{\mathrm{d}}{\mathrm{d}x}\left((1+\alpha T)\frac{\mathrm{d}T}{\mathrm{d}x}\right)=0 \;\Rightarrow\; (1+\alpha T)\frac{\mathrm{d}T}{\mathrm{d}x}=A.$$

The differential equation is with separable variables:

$$(1+\alpha T)\mathrm{d}T=A\mathrm{d}x \;\Rightarrow\; T+\frac{\alpha}{2}T^2=Ax+B.$$

The two constants can be found using $T(0)=300$ and $T(10)=350$. It is then left as an exercise to verify

that this leads to:

$$\frac{\alpha}{2}T^2 + T - (1625\alpha + 5)x - \frac{\alpha}{2}300^2 - 300 = 0.$$

This second degree equation in the variable T has two roots and only one is physical because the temperature can never be negative:

$$T(x,\alpha) = \frac{-1 + \sqrt{(1+300\alpha)^2 + 2\alpha(1625\alpha+5)x}}{\alpha}, \quad \alpha \neq 0.$$

This is, in fact, the analytical solution of the BVP. We can use it, for example, to compare the effectiveness of the numerical method. But we can also use it to verify the variability of the curves for various values of α. For example, it is clear that the second degree equation becomes

$$T - 5x - 300 = 0 \;\Rightarrow\; T(x,0) = 300 + 5x,$$

when $\alpha \to 0$, which is the straight line profile already observed. But we can also think at how the physical root defining $T(x,\alpha)$ changes for $\alpha \to \infty$. The square root becomes the significant quantity at the numerator so that the whole expression is dominated by

$$\sqrt{\frac{300^2\alpha^2 + 3250\alpha^2 x}{\alpha^2}} = \sqrt{300^2 + 3250x}.$$

So, when $\alpha \to \infty$, the temperature profile becomes

$$T(x,\infty) = \sqrt{300^2 + 3250x}.$$

This still obeys the boundary conditions:

$$T(0,\infty) = \sqrt{300^2} = 300, \qquad T(10,\infty) = \sqrt{300^2 + 32500} = 350.$$

And, when $x = 5$ we have $T(5,\infty) = \sqrt{300^2 + 3250(5)} \approx 325.96$. We can verify this using a large value for α in the numerical solution:

```
# Large alpha (infty)
ltmp <- BVPshoot2(f,x0,xf,T0,Tf,h,alpha=1000)

# Value of T in the middle of the rod
idx <- which(abs(ltmp$t-5) < 0.000001)
x5 <- ltmp$t[idx]
T5 <- ltmp$y[idx,1]
print(cbind(idx,x5,T5))
#>      idx x5       T5
#> [1,] 501  5 325.9601
```

The numerical value matches the theoretical one.

3 Exercises on EPs

3.1 Exercise 09

The stationary Schrödinger equation for a particle of mass m in one dimension is

$$-\frac{\hbar^2}{2m}\frac{\mathrm{d}^2\psi}{\mathrm{d}x^2} + V(x)\psi = E\,\psi.$$

Consider the *infinite square well* potential

$$V(x) = \begin{cases} 0, & 0 < x < L, \\ +\infty, & \text{otherwise}, \end{cases} \qquad \psi(0) = 0,\ \psi(L) = 0.$$

Inside the well ($0 < x < L$), this reduces to the Sturm–Liouville problem

$$\frac{\mathrm{d}^2\psi}{\mathrm{d}x^2} + \lambda\,\psi = 0, \qquad \psi(0) = \psi(L) = 0,$$

with parameter $\lambda = 2mE/\hbar^2$ (λ has dimensions of inverse length squared, L^{-2}; physical energies are recovered via $E = (\hbar^2/2m)\lambda$).

It is known that the eigenpairs are

$$\lambda_n = \left(\frac{n\pi}{L}\right)^2, \quad \psi_n(x) = \sqrt{\frac{2}{L}}\sin\left(\frac{n\pi x}{L}\right), \quad E_n = \frac{\hbar^2\pi^2}{2mL^2}n^2, \qquad n = 1, 2, 3, \ldots$$

1. Use `EPSturmLiouville2` to compute the first four numerical eigenvalues and eigenfunctions on $[0, L]$ with $L = 1$.
2. Compare the numerical eigenvalues with the exact values $\lambda_n = (n\pi)^2$. Report the relative errors for $n = 1, \ldots, 4$.
3. Plot the first four eigenfunctions obtained numerically and overlay the exact sine functions for visual comparison.

SOLUTION

1. We have seen a similar EP in the text book. The Sturm–Liouville problem represented has

$$p(x) = 1,\ q(x) = 0,\ w(x) = 1.$$

It is therefore straightforward to set function `EPStrurmLiouville2` up as done in the following snippet. The stepsize can be $h = 0.01$, which means that $[0, 1]$ will have $n = 1 = 101$.

```
# Make sure comphy is loaded in memory
require(comphy)

# Define the interval and number of grid points
a <- 0
b <- 1
n <- 100
x <- seq(a,b,length.out=n+1)

# Define constant coefficient functions
p <- function(s) 1
q <- function(s) 0
w <- function(s) 1

# Solve the Sturm-Liouville eigenproblem (4 eigenvalues wanted)
ep <- EPSturmLiouville2(p,q,w,x,nev=4,
                        normalize=TRUE,return_matrices=TRUE)

# Print the eigenvalues
print(round(ep$values,3))
#> [1]   9.869  39.465  88.761 157.706
```

2. The theoretical values λ_n, given that $L = 1$, are $\lambda_n = n^2\pi^2$. Let us look at the numerical comparison and the relative error.

```
# Theoretical quantities
lbdas <- ((1:4)*pi)^2
```

```
# Compare numerical and theoretical eigenvalues
print(cbind(numerical=ep$values,theoretical=lbdas))
#>       numerical theoretical
#> [1,]   9.868793    9.869604
#> [2,]  39.465431   39.478418
#> [3,]  88.760708   88.826440
#> [4,] 157.705974  157.913670

# Relative errors
Rers <- abs(lbdas-ep$values)/abs(lbdas)
print(cbind(n=1:4,RelErr=Rers))
#>      n       RelErr
#> [1,] 1 0.0000822440
#> [2,] 2 0.0003289435
#> [3,] 3 0.0007400012
#> [4,] 4 0.0013152548
```

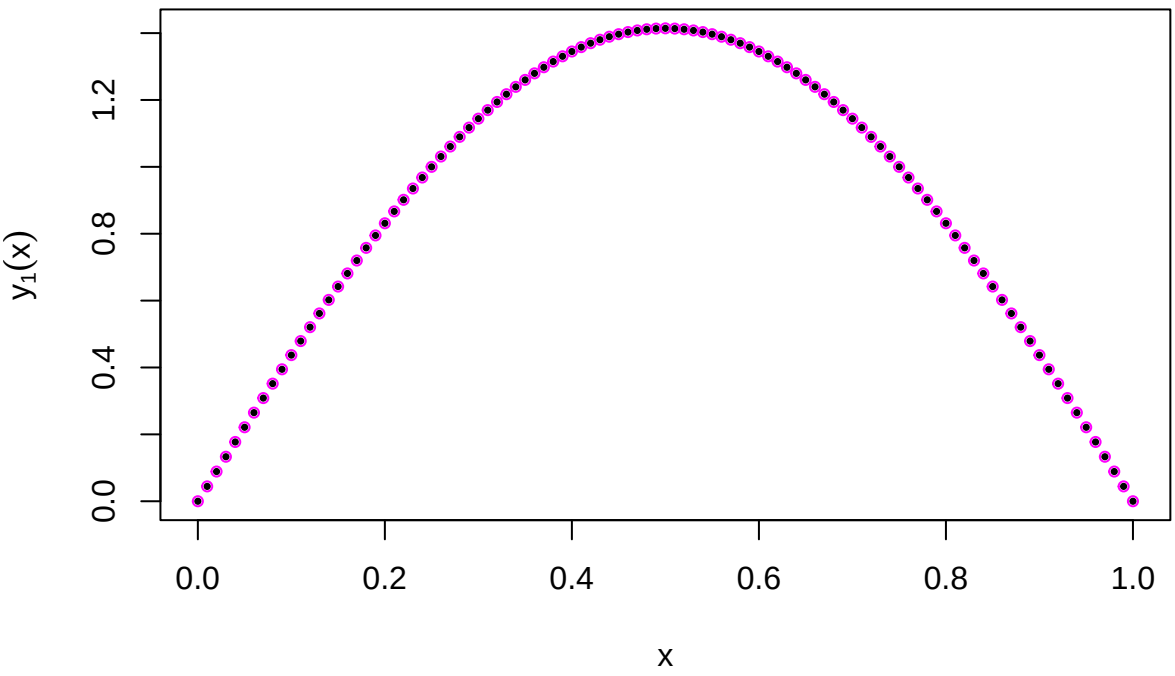
The relative errors increase with increasing n. This is to be expected (see main text).

3. The exact eigenfunctions are

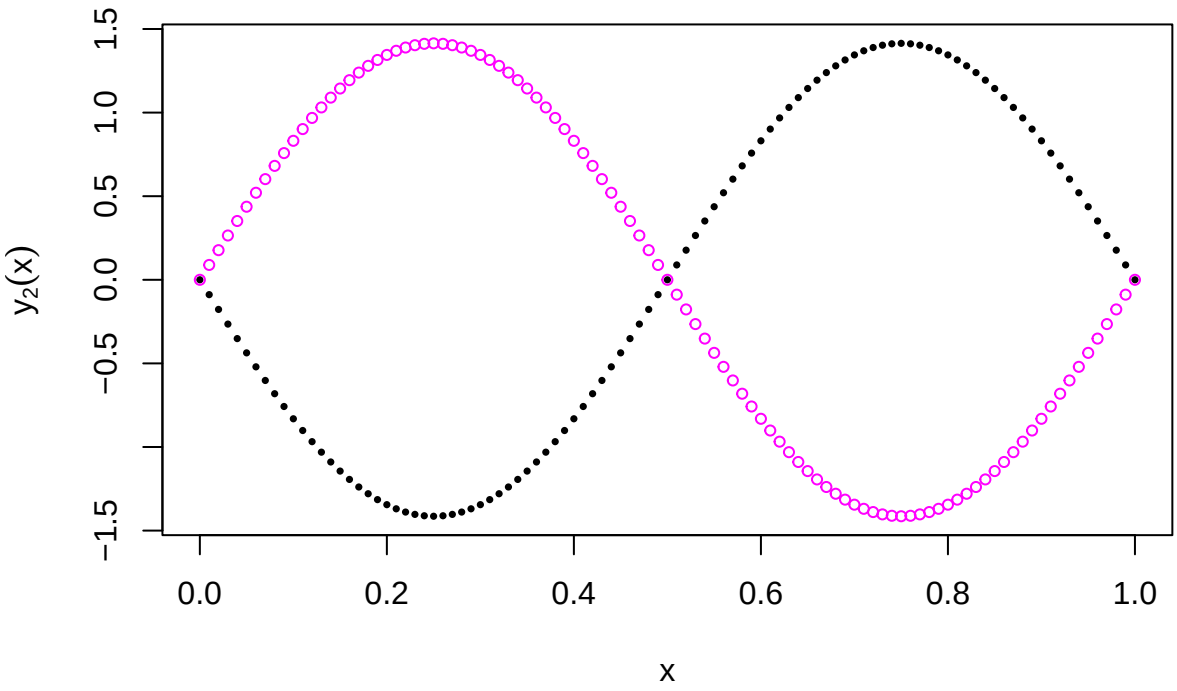
$$\psi_n(x) = \sqrt{2}\sin(n\pi x), \qquad n = 1, 2, 3, 4.$$

For the graphical comparison, let us look at the following code.

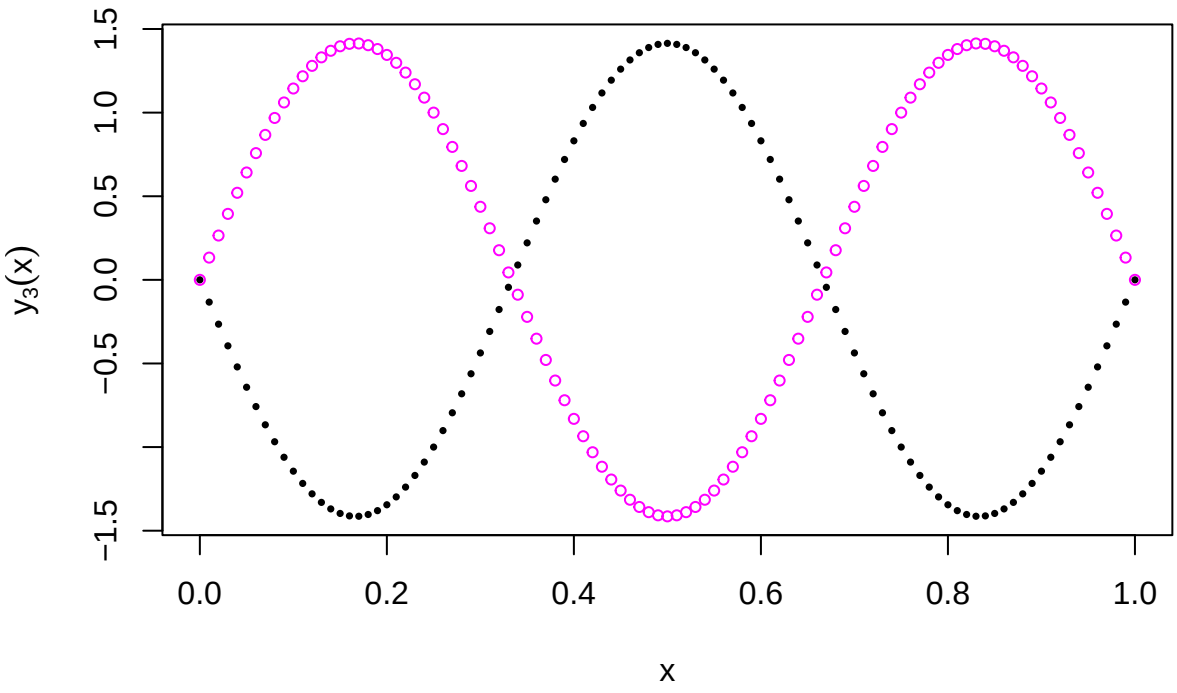
```
# Evaluate true eigenfunctions at n+1 points
xx <- seq(0,1,length.out=n+1)
yy1 <- sqrt(2)*sin(pi*xx)
yy2 <- sqrt(2)*sin(2*pi*xx)
yy3 <- sqrt(2)*sin(3*pi*xx)
yy4 <- sqrt(2)*sin(4*pi*xx)

# First comparison plot
plot(xx,ep$vectors_full[,1],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y[1](x)))
points(xx,yy1,type="b",pch=1,col="magenta",cex=0.7)
```



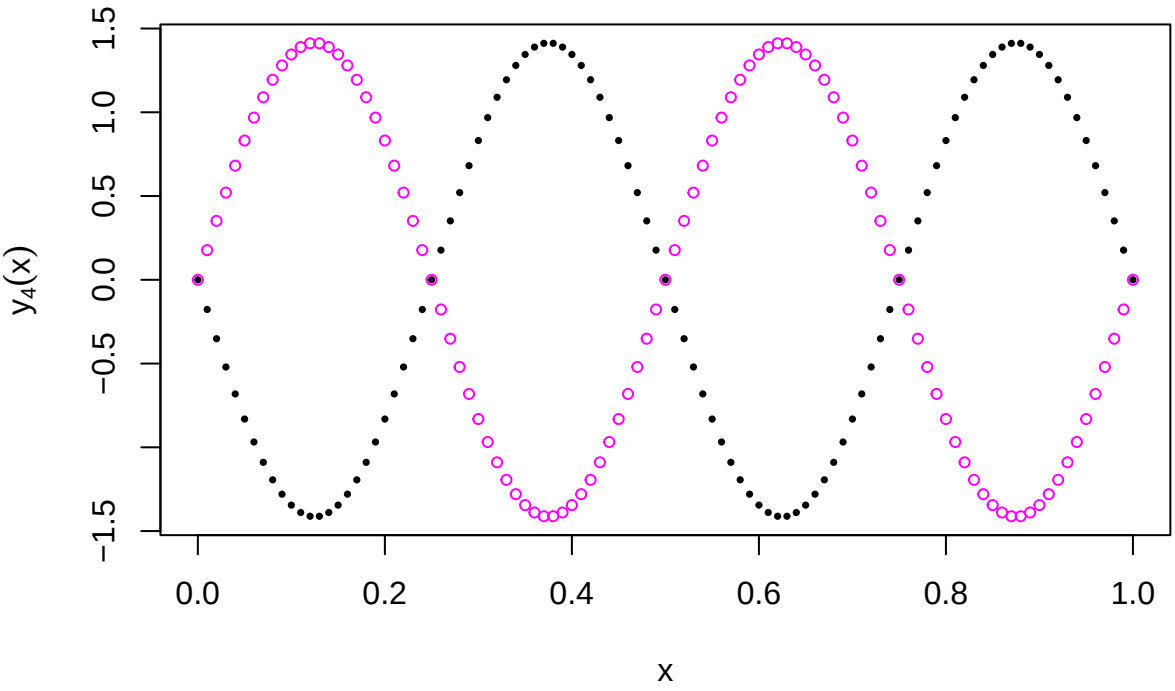

```
# Secons comparison plot
plot(xx,ep$vectors_full[,2],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y[2](x)))
points(xx,yy2,type="b",pch=1,col="magenta",cex=0.7)
```




```
# Third comparison plot
plot(xx,ep$vectors_full[,3],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y[3](x)))
points(xx,yy3,type="b",pch=1,col="magenta",cex=0.7)
```




```
# Fourth comparison plot
plot(xx,ep$vectors_full[,4],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y[4](x)))
points(xx,yy4,type="b",pch=1,col="magenta",cex=0.7)
```



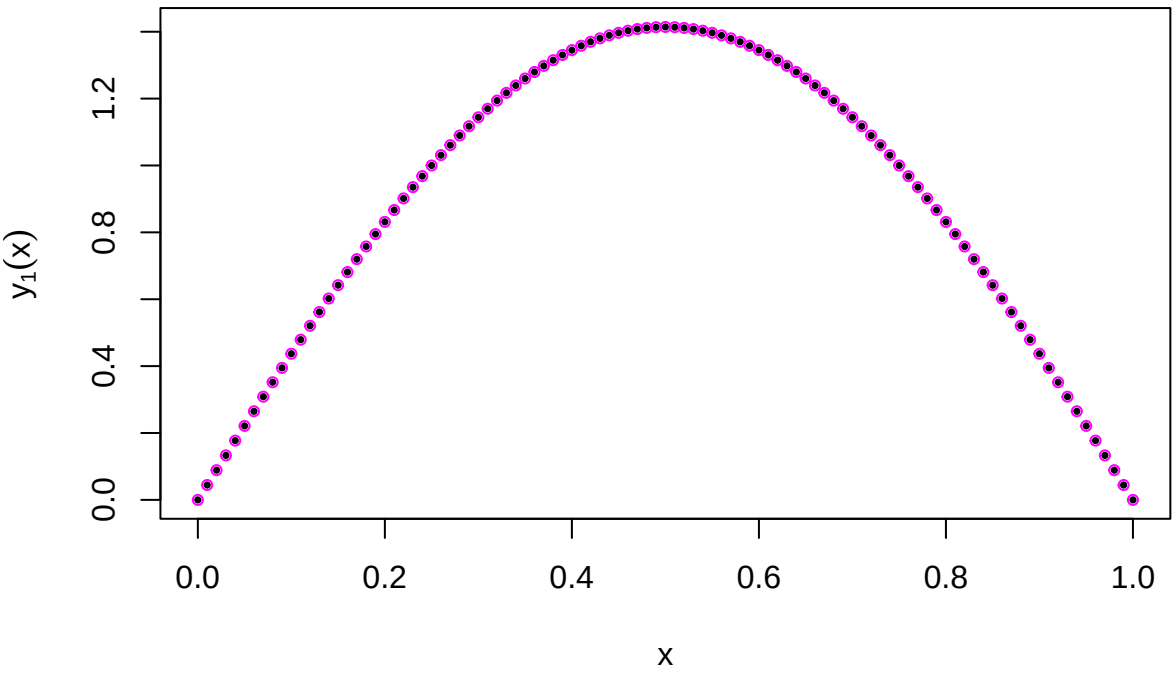
While the fundamental eigenfunctions matches exactly, the other modes appear "inverted'': why? Eigenvalue problems determine eigenfunctions only *up to a nonzero scalar multiple*. In our real, self–adjoint setting (Sturm–Liouville with homogeneous Dirichlet conditions and its standard centred–difference discretisation), each simple eigenpair (λ_k, ϕ_k) can be represented equally well by $(\lambda_k, -\phi_k)$. The continuous eigenfunctions are thus unique only up to a global sign, and the same holds for the discrete eigenvectors of the symmetric tridiagonal matrix.

Numerical eigensolvers enforce a norm (e.g. $\|\phi_k\| = 1$) but do not enforce a sign. Implementation details and roundoff decide whether the solver returns ϕ_k or $-\phi_k$. As a result, when we overlay numerical eigenfunctions with the analytic ones (e.g. $\sin(n\pi x)$ on $[0, 1]$), some curves may appear upside–down. This is not an error; physical quantities (like $|\psi|^2$) and orthogonality are unchanged by a global sign.

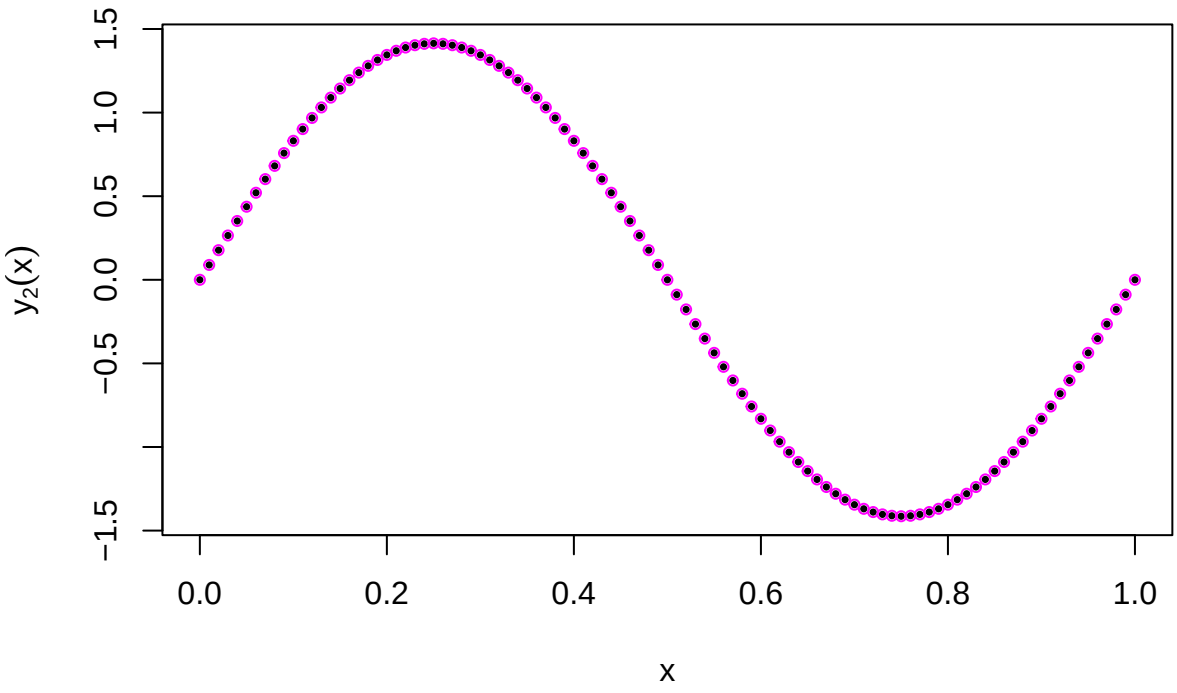
Thus, before plotting or comparing with a reference function ϕ_k^{ref}, one should choose the orientation of the numerical eigenvector $y^{(k)}$ by a sign that maximises their correlation with respect to the appropriate inner product. Equivalently, in the absence of a reference curve, one could enforce a simple rule such as "make the component of largest magnitude positive''. Either convention removes the apparent inversions and yields consistent plots across modes and mesh refinements. For the case under study we can, for example, use the following:

```
# Align by correlation with known eigenfunctions
Ytrue <- cbind(yy1,yy2,yy3,yy4)
V <- ep$vectors_full
for (k in 1:4) {
  if (sum(V[,k]*Ytrue[,k]) < 0) V[,k] <- -V[,k]
}

# Plot again
```

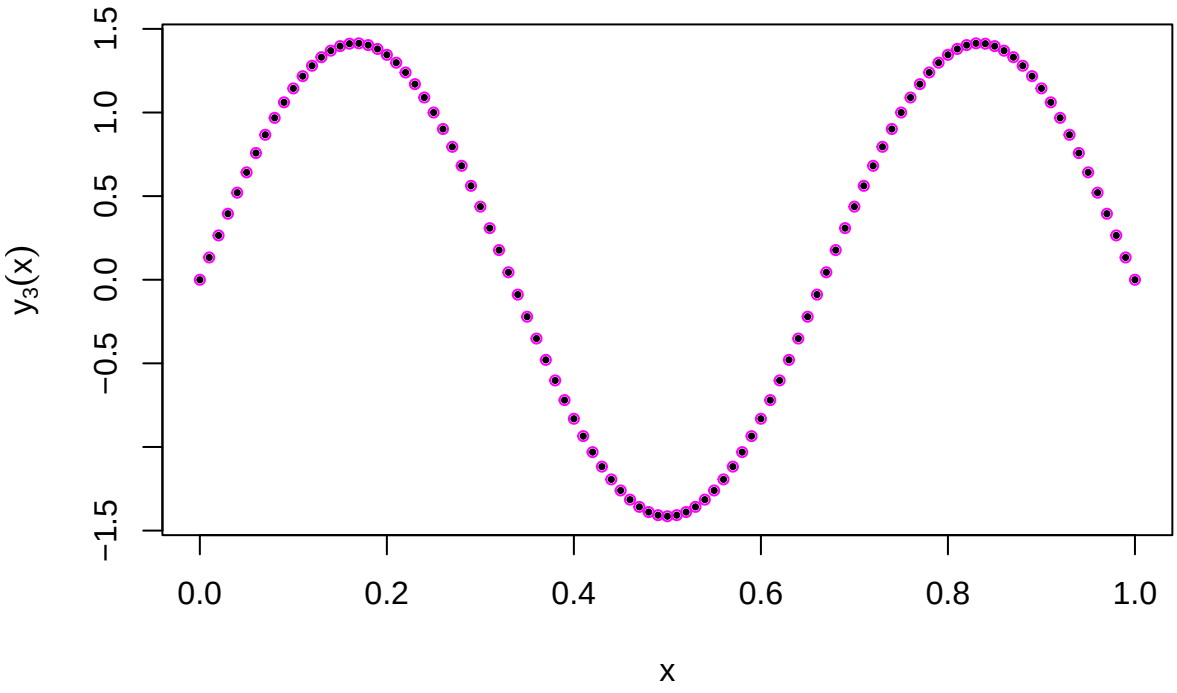
```r
# First comparison plot
plot(xx,V[,1],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y[1](x)))
points(xx,yy1,type="b",pch=1,col="magenta",cex=0.7)
```



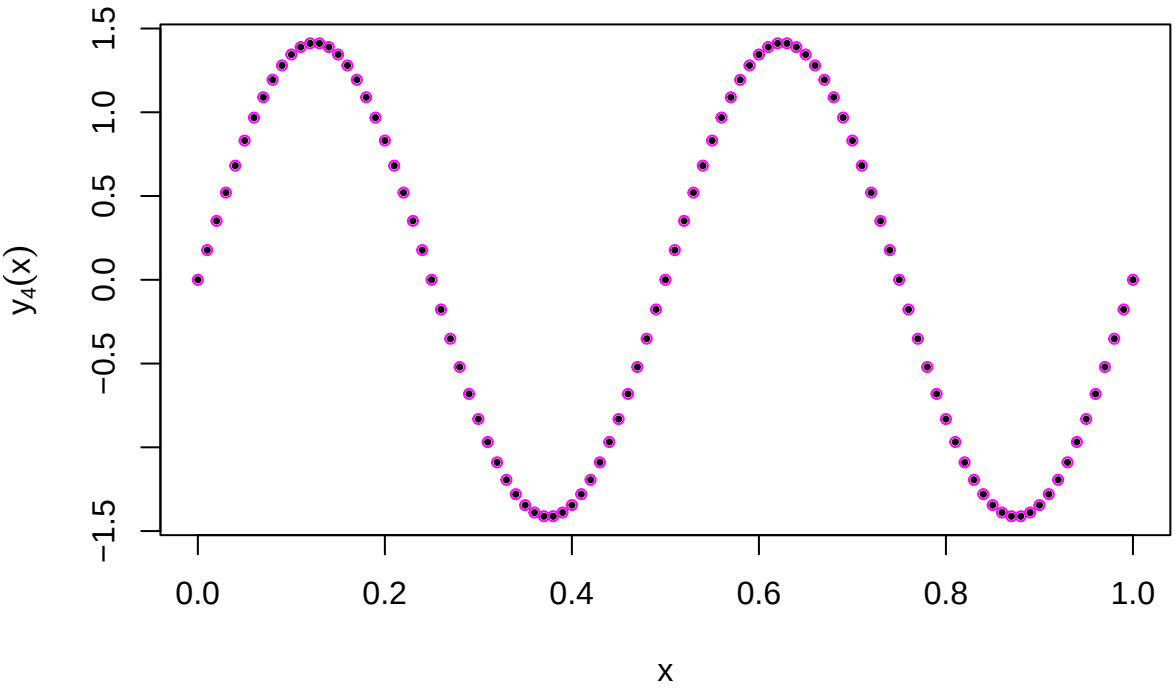

```r
# Secons comparison plot
plot(xx,V[,2],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y[2](x)))
points(xx,yy2,type="b",pch=1,col="magenta",cex=0.7)
```




```
# Third comparison plot
plot(xx,V[,3],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y[3](x)))
points(xx,yy3,type="b",pch=1,col="magenta",cex=0.7)
```




```
# Fourth comparison plot
plot(xx,V[,4],type="b",pch=16,cex=0.5,
     xlab=expression(x),ylab=expression(y[4](x)))
points(xx,yy4,type="b",pch=1,col="magenta",cex=0.7)
```



Remark. For problems with degenerate eigenvalues (not the case here), any orthonormal basis of the eigenspace is acceptable; numerical solvers may return different orthonormal combinations. A sign (or, in complex settings, a phase) alignment step against a chosen reference basis restores visual consistency.

3.2 Exercise 10

The BVP related to the *modified Legendre equation*,

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left((1-x^2)\frac{\mathrm{d}y}{\mathrm{d}x}\right) = \lambda(1-x^2)y, \qquad y(-1) = y(1) = 0.$$

is a Sturm–Liouville problem with homogeneous Dirichlet conditions but nonconstant weight function:

$$p(x) = 1 - x^2, \qquad q(x) = 0, \qquad w(x) = 1 - x^2.$$

1. Use `EPSturmLiouville2` to find the first four eigenvalues and eigenvectors of the problem, using a step size $h = 0.01$.
2. Plot the eigenvectors in the interval $[-1, 1]$.
3. Repeat using a coarser grid with $h = 0.1$. Compare eigenvalues and eigenvectors with those of the first numerical solution.

SOLUTION

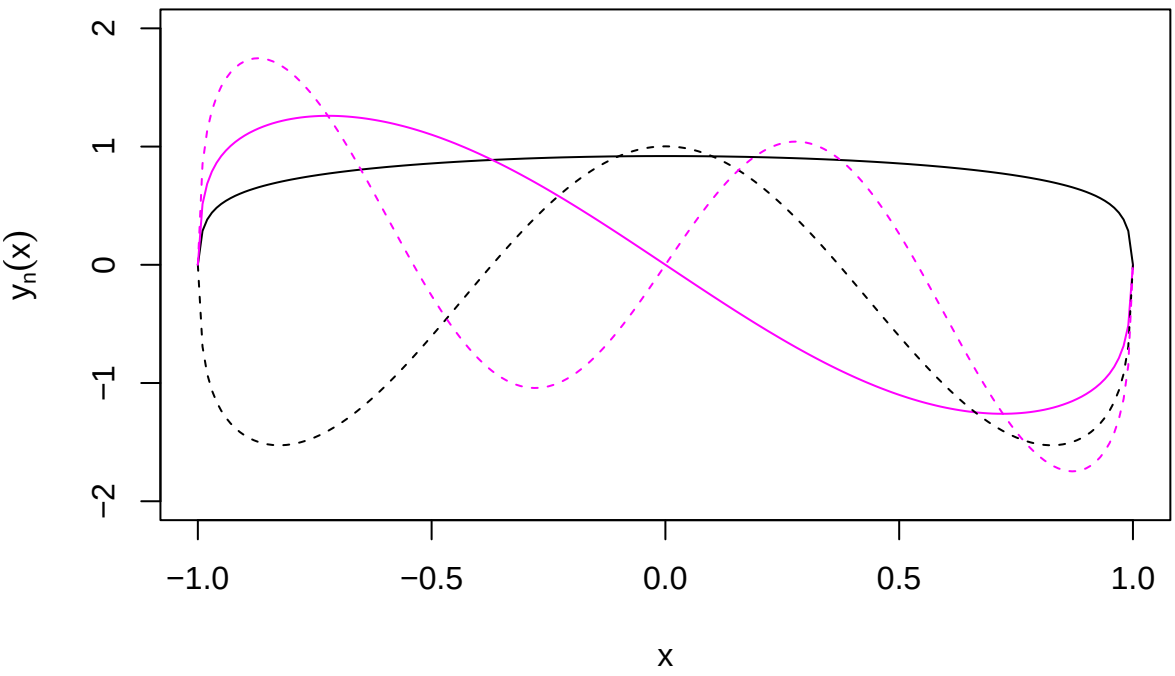
1. For the numerical solution we can follow the same structure used in the main text.

```
# Define the interval and grid (h=0.01)
a <- -1
b <- 1
```

```
h <- 0.01
x <- seq(a,b,by=h) # Should have n+1=201 grid points

# Define functions
p <- function(s) 1-s^2  # p(x)
q <- function(s) 0  # q(x)
w <- function(s) 1-s^2  # w(x)

# Solve the Sturm-Liouville eigenproblem
ep <- EPSturmLiouville2(p,q,w,x,nev=4,normalize=TRUE)

# Print the eigenvalues
print(round(ep$values,3))
#> [1]  0.497  6.125 16.924 32.774
```

2. We will plot the four eigenfunctions using a vertical range $[-2, 2]$, in a same window. An interesting observation is that not all modes are symmetric.

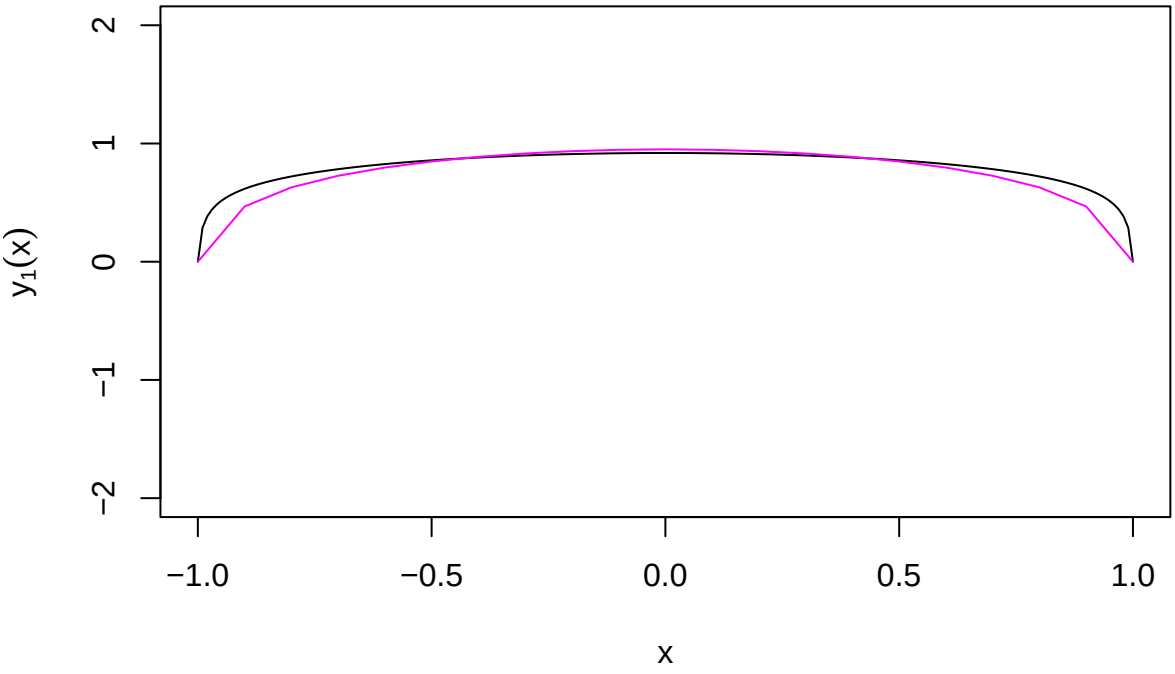
```
# Copy the four eigenfunctions unto separate vectors
y1 <- ep$vectors_full[,1]
y2 <- ep$vectors_full[,2]
y3 <- ep$vectors_full[,3]
y4 <- ep$vectors_full[,4]

# Plot using different colours/line traits
plot(x,y1,type="l",ylim=c(-2,2),
     xlab=expression(x),ylab=expression(y[n](x)))
points(x,y2,type="l",col="magenta")
points(x,y3,type="l",lty=2)
points(x,y4,type="l",col="magenta",lty=2)
```



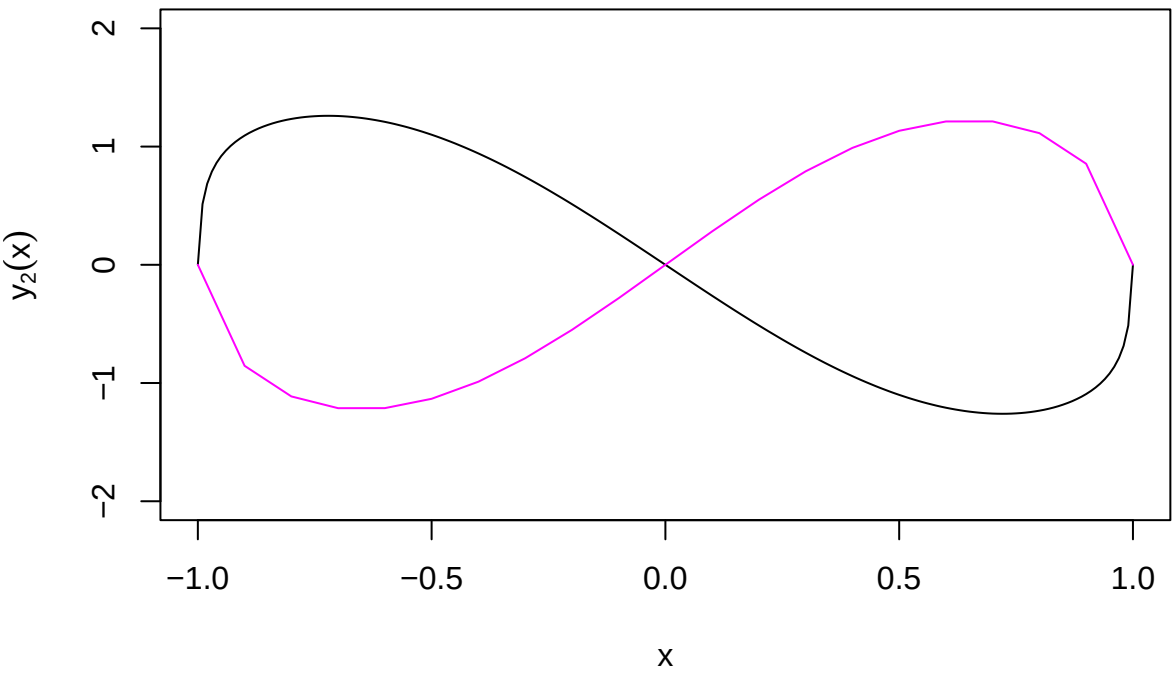
3. Use $h = 0.1$. The steps are the same and we can compare both eigenvalues (nuerically) and eigenvectors (graphically into four separate plots).

```
# Need only to modify the x grid
h <- 0.1
x2 <- seq(a,b,by=h) # Should have n+1=21 grid points

# Solve the Sturm-Liouville eigenproblem
ep2 <- EPSturmLiouville2(p,q,w,x2,nev=4,normalize=TRUE)

# Print the eigenvalues
print(round(ep2$values,3))
#> [1]  0.796  7.070 18.484 34.611

# Compare eigenvalues
cmp <- cbind(n=1:4,h001=ep$values,h01=ep2$values)
print(round(cmp,3))
#>      n   h001    h01
#> [1,] 1  0.497  0.796
#> [2,] 2  6.125  7.070
#> [3,] 3 16.924 18.484
#> [4,] 4 32.774 34.611

# Eigenfunctions
y2_1 <- ep2$vectors_full[,1]
y2_2 <- ep2$vectors_full[,2]
y2_3 <- ep2$vectors_full[,3]
```

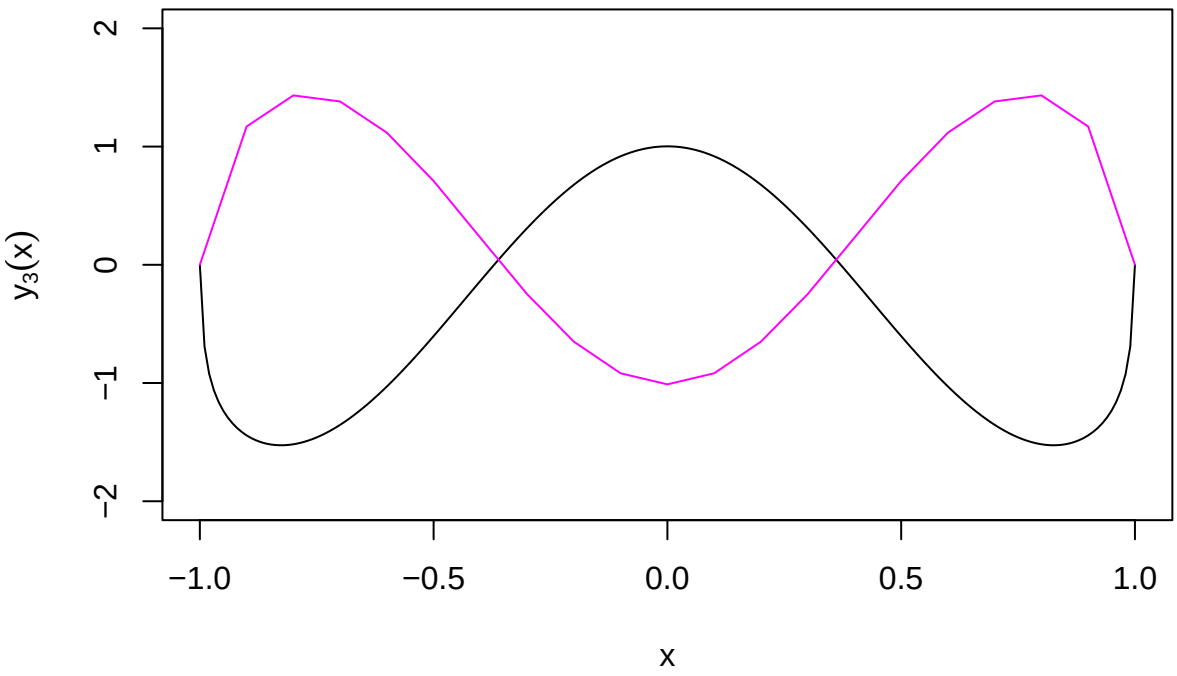
```
y2_4 <- ep2$vectors_full[,4]

# Compare eigenfunctions
plot(x,y1,type="l",ylim=c(-2,2),
     xlab=expression(x),ylab=expression(y[1](x)))
points(x2,y2_1,type="l",col="magenta")
```



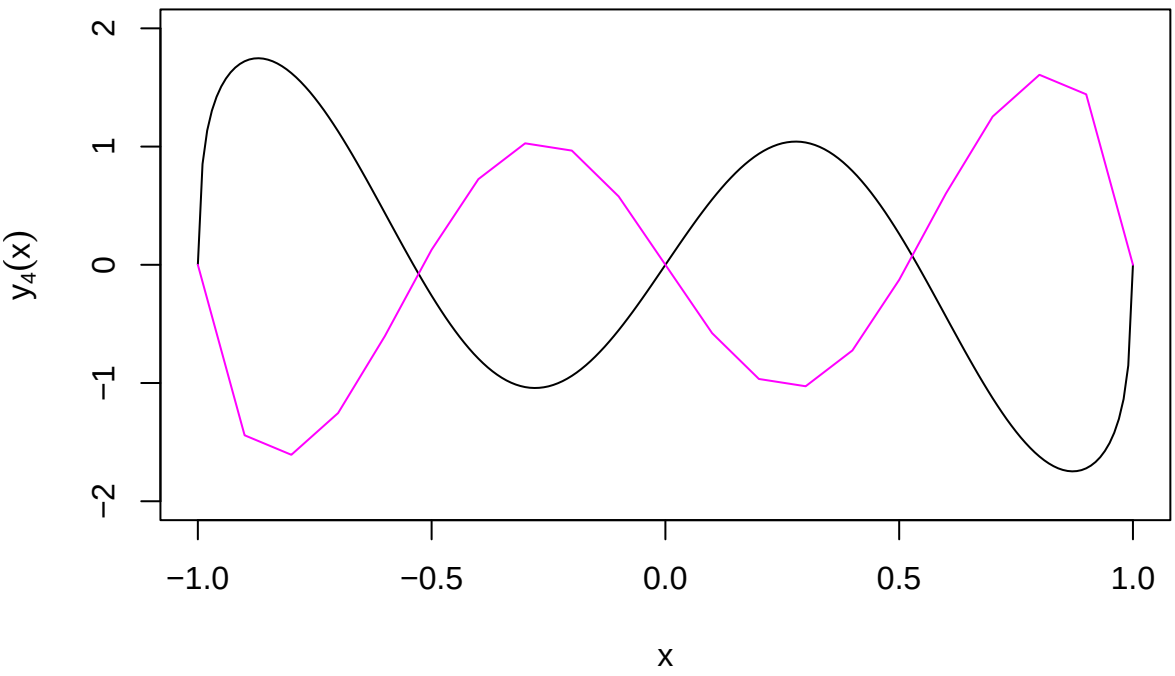

```
plot(x,y2,type="l",ylim=c(-2,2),
     xlab=expression(x),ylab=expression(y[2](x)))
points(x2,y2_2,type="l",col="magenta")
```




```
plot(x,y3,type="l",ylim=c(-2,2),
     xlab=expression(x),ylab=expression(y[3](x)))
points(x2,y2_3,type="l",col="magenta")
```

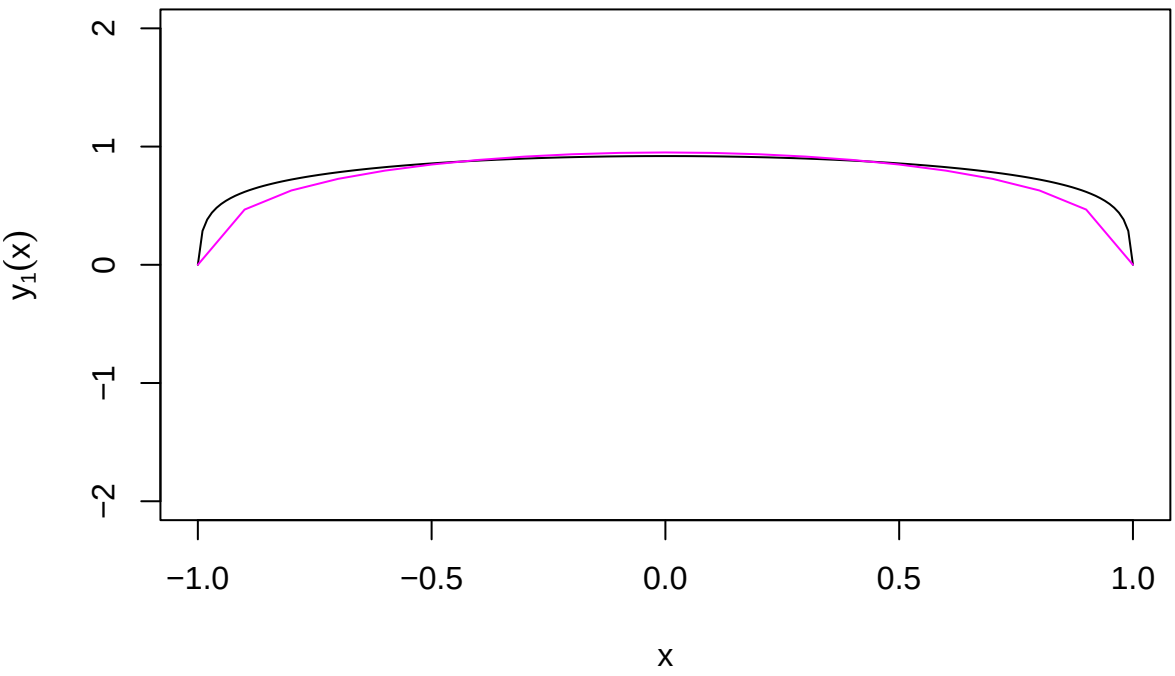



```
plot(x,y4,type="l",ylim=c(-2,2),
     xlab=expression(x),ylab=expression(y[4](x)))
points(x2,y2_4,type="l",col="magenta")
```

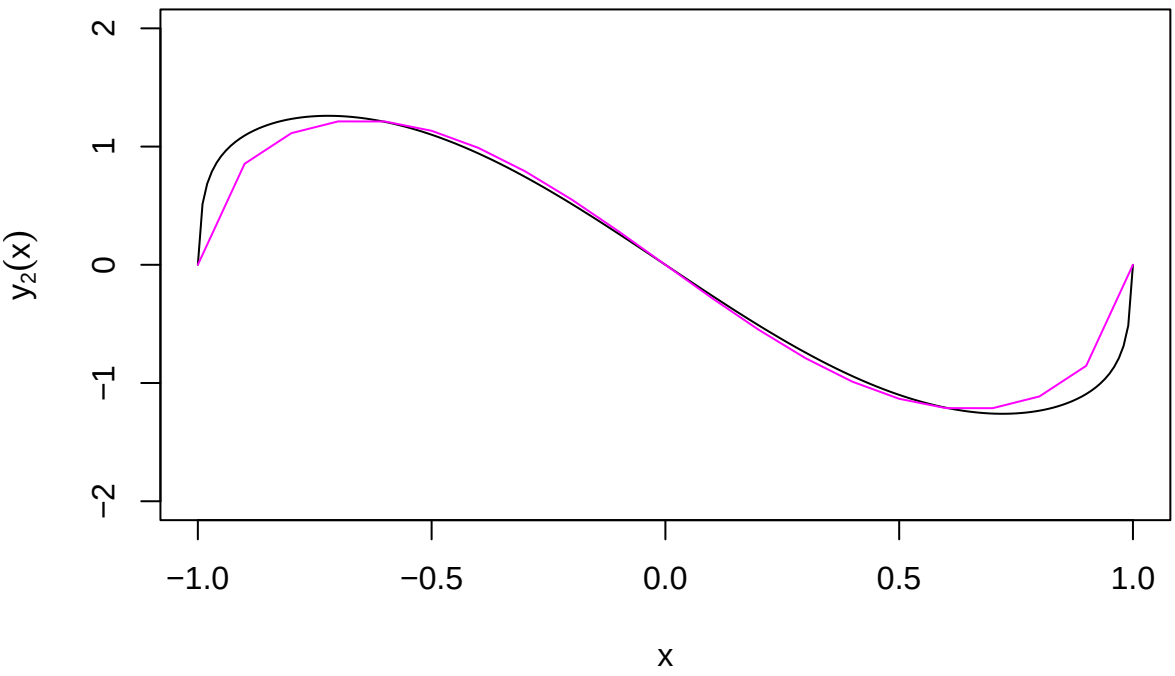


The same problem of inverted sign for eigenfunctions, has presented itself here. Let us *cure* the problem as done in the previous exercise, with the extra complication that we will have to match points for the correlation, as the grids are different (see Exercise 06).

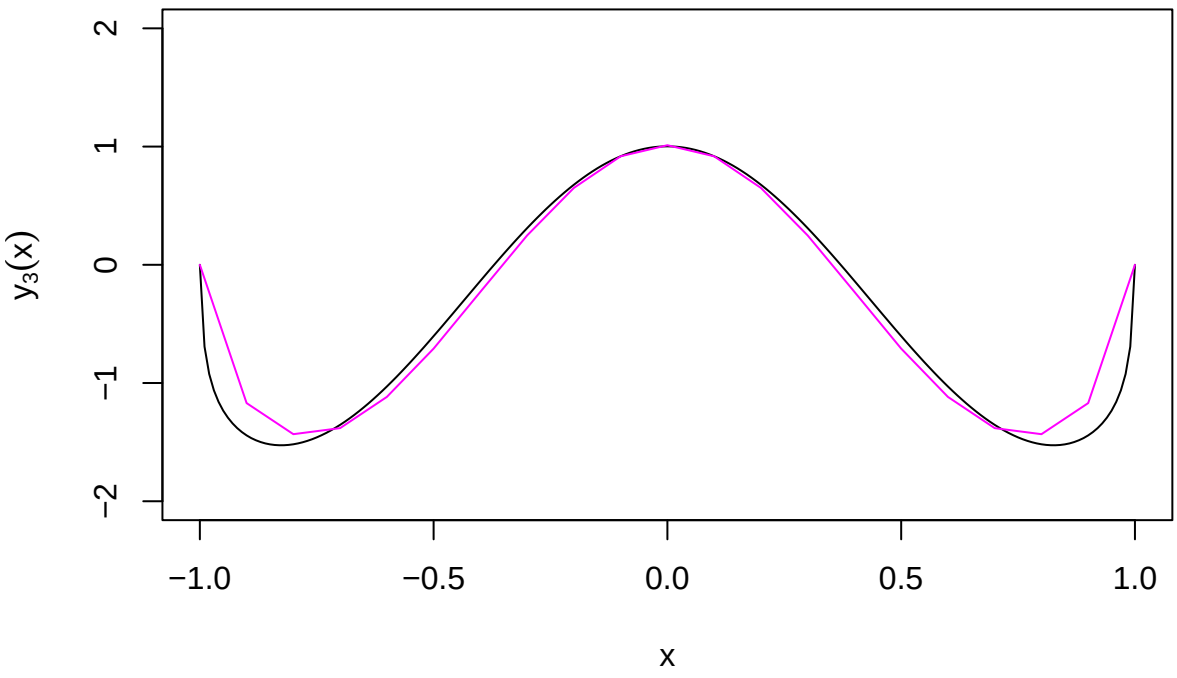
```
# Matching points. Row i, Column j equal TRUE means there's a match
tol <- 1e-12
M <- abs(outer(x,x2,`-`)) <= tol

# Transform in index pairs. Matrix with two columns: (i "matches" j)
pairs <- which(M,arr.ind=TRUE)

# Align by correlation with known eigenfunctions
Ytrue <- cbind(y1[pairs[,1]],y2[pairs[,1]],
               y3[pairs[,1]],y4[pairs[,1]])
V <- ep2$vectors_full
for (k in 1:4) {
  if (sum(V[,k]*Ytrue[,k]) < 0) V[,k] <- -V[,k]
}

# Now plot comparisons again.
plot(x,y1,type="l",ylim=c(-2,2),
     xlab=expression(x),ylab=expression(y[1](x)))
points(x2,V[,1],type="l",col="magenta")
```



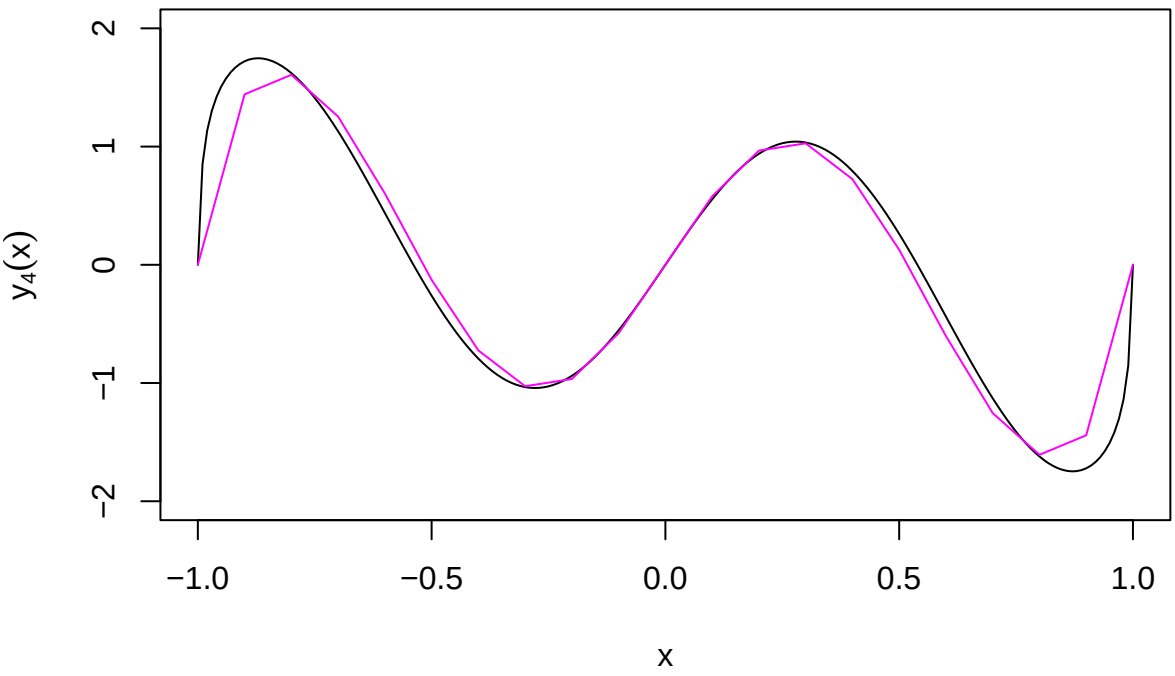

```
plot(x,y2,type="l",ylim=c(-2,2),
     xlab=expression(x),ylab=expression(y[2](x)))
points(x2,V[,2],type="l",col="magenta")
```




```
plot(x,y3,type="l",ylim=c(-2,2),
     xlab=expression(x),ylab=expression(y[3](x)))
points(x2,V[,3],type="l",col="magenta")
```




```
plot(x,y4,type="l",ylim=c(-2,2),
     xlab=expression(x),ylab=expression(y[4](x)))
points(x2,V[,4],type="l",col="magenta")
```



We can observe, as it is generally the case in all problems with a discretised integration interval, that the accuracy decreases with the coarser grid.

We can also try and observe empirically the difference in eigenvalues when the grid is refined further.

```
# Need only to modify the x grid
h <- 0.005
x3 <- seq(a,b,by=h) # Should have n+1=401 grid points

# Solve the Sturm-Liouville eigenproblem
ep3 <- EPSturmLiouville2(p,q,w,x3,nev=4,normalize=TRUE)

# Need only to modify the x grid
h <- 0.0025
x4 <- seq(a,b,by=h) # Should have n+1=801 grid points

# Solve the Sturm-Liouville eigenproblem
ep4 <- EPSturmLiouville2(p,q,w,x4,nev=4,normalize=TRUE)

# Compare eigenvalues
cmp <- cbind(n=1:4,h01=ep2$values,h001=ep$values,h0005=ep3$values,
             h00025=ep4$values)
print(round(cmp,3))
#>      n    h01   h001  h0005 h00025
#> [1,] 1  0.796  0.497  0.446  0.405
#> [2,] 2  7.070  6.125  5.963  5.832
#> [3,] 3 18.484 16.924 16.633 16.399
```

```
#> [4,] 4 34.611 32.774 32.338 31.991
```

In this case we can see that values are converging from above. The appropriate step size can be decided based, for example, on a minimal drop in value between corresponding eigenvalues for different step sizes.