# Computational Physics with Maxima or R: Ch. 3, Two Point Boundary Value and Eigenvalue Problems *

Edwin (Ted) Woollett

August 24, 2015

## Contents



*The code examples use **R ver. 3.0.2** and **Maxima ver. 5.31** using **Windows 7**. This is a live document which will be updated when needed. Check **http://www.csulb.edu/~woollett/** for the latest version of these notes. Send comments and suggestions for improvements to **woollett@charter.net**

**Code files which accompany cp3.pdf are**
**1. cp3.mac**
**2. cp3.R**

Feedback from readers is the best way for this series of notes to become more helpful to users of **R** and **Maxima**. *All* comments and suggestions for improvements will be appreciated and carefully considered.

# 1 A Simple Shooting Solution for Linear 2nd order ODE BVPs

A simple solution is available for the linear (in y and its derivatives) 2nd order ode boundary value problem (BVP)

$$y''(x) = p(x)\,y'(x) + q(x)\,y(x) + r(x), \quad y(a) = A, \quad y(b) = B. \tag{1.1}$$

Let $u(x)$ be the solution of the initial value problem (IVP)

$$u''(x) = p(x)\,u'(x) + q(x)\,u(x) + r(x), \quad u(a) = A, \quad u'(a) = \delta_1 \tag{1.2}$$

and let $v(x)$ be the solution of the IVP

$$v''(x) = p(x)\,v'(x) + q(x)\,v(x) + r(x), \quad v(a) = A, \quad v'(a) = \delta_2, \tag{1.3}$$

in which $\delta_1$ and $\delta_2$ are two different guesses for the initial value of $y'(a)$ which will result in approximately satisfying $y(b) = B$.

We can then take an appropriate linear combination of these two IVP solutions

$$y(x) = \alpha\,u(x) + \beta\,v(x), \quad y'(x) = \alpha\,u'(x) + \beta\,v'(x), \quad y''(x) = \alpha\,u''(x) + \beta\,v''(x) \tag{1.4}$$

in which $\alpha$ and $\beta$ are constants chosen such that $y(a) = A$ (which implies $\alpha + \beta = 1$ since $u(a) = A$ and $v(a) = A$) and such that $y(b) = \alpha\,u(b) + \beta\,v(b) = B$. We then get

$$y(x) = \alpha\,u(x) + (1 - \alpha)\,v(x), \tag{1.5}$$

where

$$\alpha = \frac{B - v(b)}{u(b) - v(b)} \tag{1.6}$$

We provide code **`linear1`** (in both Maxima and **`R`**) which implements this solution method.

## 1.1 Example 1 using R

We use our homemade R functions **`myrk4`** and **`linear1`** to solve the linear 2nd order ode BVP

$$y'' = -\frac{\pi^2}{4}\,(1 + y), \quad \text{for} \quad y(0) = 0, \quad y(1) = 1. \tag{1.7}$$

The analytic solution is

$$y_{an}(x) = \cos(\frac{\pi\,x}{2}) + 2\,\sin(\frac{\pi\,x}{2}) - 1. \tag{1.8}$$

Just for practice, we can derive this analytic solution using the Maxima functions **`ode2`** for the general solution and **`bc2`** for the particular solution which satisfies the given boundary conditions. Maxima returns the general solution in terms of two constants which it calls **`%k1`** and **`%k2`**.

```
(%i1) de : 'diff(y,x,2) + %pi^2/4 * (1 + y);
(%o1) 'diff(y,x,2)+%pi^2*(y+1)/4
(%i2) gsoln : ode2(de,y,x);
(%o2) y = %k1*sin(%pi*x/2)+%k2*cos(%pi*x/2)-1
(%i3) psoln : bc2(gsoln,x=0,y=0,x=1,y=1);
(%o3) y = 2*sin(%pi*x/2)+cos(%pi*x/2)-1
(%i4) ysoln : rhs(%);
(%o4) 2*sin(%pi*x/2)+cos(%pi*x/2)-1
```

We check this analytic solution for boundary conditions and (again, just for practice) for satisfying the given differential equation. Note the use of **`ratsimp(expand(...))`** to massage an expression which should reduce to zero.

```
(%i5) ysoln, x=0;
(%o5) 0
(%i6) ysoln, x=1;
(%o6) 1
(%i7) yd2 : diff(ysoln,x,2);
(%o7) -%pi^2*sin(%pi*x/2)/2-%pi^2*cos(%pi*x/2)/4
(%i8) ratsimp(expand(yd2 + (ysoln + 1)*%pi^2/4));
(%o8) 0
(%i9) dy : diff(ysoln,x);
(%o9) %pi*cos(%pi*x/2)-%pi*sin(%pi*x/2)/2
(%i10) dy,x=0;
(%o10) %pi
```

Here is **R** code **linear1** from **cp3.R** which finds a solution for a 2nd order linear ode BVP (for which the boundary conditions are $y(a) = A$ and $y(b) = B$) by using two different values (input by the program user interactively) for the unknown $y'(a)$.

```
##    linear1 (bc, grid, func)
##    linear 2nd order ode BVP method over finite interval
##          for case:
##       y''(x) = p(x) y'(x) + q(x) y(x) + r(x), a <= x <= b
##       y1(x) = y(x),   y2(x) = y'(x)
##       y(a) = A, y(b) = B
##    Two guesses of y'(a) are asked for interactively.
##    The vector 'grid'  can be generated as xL = seq(a,b,h), for example.
##    linear1 calls homemade myrk4, which expects func to return
##       a R vector c(y[2], y'' as a function of x,y[1], and y[2]).
##    For the above problem, bc = c(A,B).
##    linear1 returns list(y1L, y2L), in which y1L is a R vector of y(x) grid
##       values, and y2L is a R vector of y'(x) grid values.

linear1 = function(bc, grid, func) {
    A = bc[1]
    B = bc[2]
##    get first guess for y'(a)
    del = as.numeric ( readline ( " first guess y'(a) = " ))

##    get first IVP solution ; myrk4 returns list(y1L, y2L)
    outL = myrk4 ( init = c(A, del), grid = grid, func = func        )
    uL = outL[[1]]
    upL = outL[[2]]

##    get second guess for y'(a)
    del = as.numeric ( readline ( " second guess y'(a) = " ))

##    get second IVP solution
    outL = myrk4 ( init = c(A, del), grid = grid, func = func        )
    vL = outL[[1]]
    vpL = outL[[2]]

    ub = last(uL)
    vb = last(vL)
    sdiff = ub - vb

##    check that sdiff is not "zero" so we can divide by it
    if  (abs(sdiff) < 1e-12)  {
        cat(" sdiff = ub - vb  is too small; choose different values for y'(a) guesses. \n ")
        return() }

##    form  numerical solution y1L and its first derivative y2L
    alpha = (B - vb)/sdiff    # scalar number needed to construct solution
    y1L = alpha*uL + (1 - alpha)*vL
    y2L =  alpha*upL + (1 - alpha)*vpL
    list ( y1L, y2L ) }
```

and here is an example of the use of **`linear1`** to solve the 2nd order linear ode BVP (1.7) .

```
> source("cp3.R")
> ex1 = function(x,y) { c( y[2], -(1 + y[1])*pi^2/4) }
> xL = seq(0, 1, 0.01)
> BC = c(0, 1)
> soln = linear1(bc=BC, grid = xL, func = ex1)
  first guess y'(a) = 1
  second guess y'(a) = 1.5
> yL = soln[[1]]
> fll(yL)
  0   1   101
> ypL = soln[[2]]
> fll(ypL)
  3.141593   -1.570796   101
> plot(xL,yL, type = "l", lwd = 2, col = "blue", ylab = "y(x)", xlab = "x")
> mygrid()
> curve ( cos(pi*x/2)+2*sin(pi*x/2) -1,0,1, type = "p", add = TRUE, col = "red")
> legend("bottomright", col = c("blue","red"), legend = c( "numerical", "exact"),
+            lwd = 2)
```

which shows agreement between the numerical solution and the exact solution.

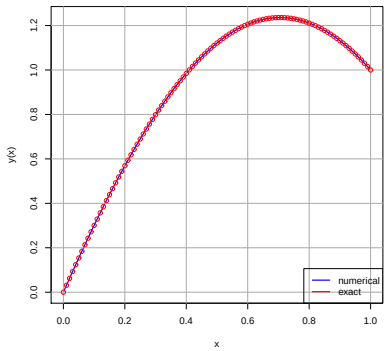


Figure 1: Example 1 Numerical $y(x)$ Compared with Analytic Solution

Next we compare the numerical results for the first derivative with the exact value of the second derivative.

```
> plot(xL,ypL, type = "l", lwd = 2, col = "blue", ylab = "dy/dx", xlab = "x")
> mygrid()
> curve ( pi*cos(pi*x/2) - pi*sin(pi*x/2)/2 ,0,1,type = "p",add = TRUE,col = "red")
> legend("topright", col = c("blue","red"), legend = c( "numerical", "exact"),
+                lwd = 2)
```

which shows agreement:

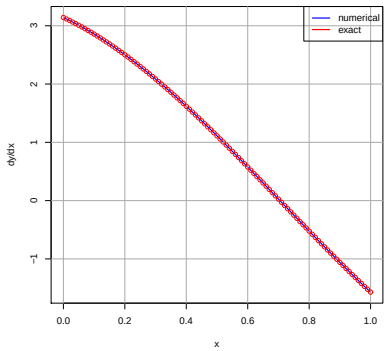


Figure 2: Example 1 Numerical $y'(x)$ Compared with Analytic Solution

## 1.2 Example 1 using Maxima

Here is Maxima code **linear1**, from **cp3.mac**, which can be used to solve a 2nd order linear ode BVP (for which the boundary conditions are $y(a) = A$ and $y(b) = B$).

```
/*    linear1 ([dy1dx,dy2dx], [y1,y2], [A,B], [x,a,b,h])
    2nd order linear ode BVP method
    Uses our homemade rk4 code from cp3.mac
        to solve the problem:
       y''(x) = p(x) y'(x) + q(x) y(x) + r(x), a <= x <= b
       y1(x) = y(x),   y2(x) = y'(x)
       y(a) = A, y(b) = B
    x is the independent variable, h is the grid spacing.
    Two guesses of y'(a) are asked for interactively.

    Returns a Maxima list with elements of the form [xi, y1i, y2i],
      which are approximations to [x, y(x), y'(x)] at the grid points.
*/
linear1(dy1dy2, ivar, bc, xgrid) :=
block([A,B,del, xL,uL,upL, ub, vL, vpL,vb,
               outL, sdiff, alpha, y1L, y2L,
               small : 1e-12, numer],numer:true,
    if length(dy1dy2) # 2 then (
        print(" need two first order ode derivatives"),
        return(done)),
    A : bc[1],
    B : bc[2],
/*     get first guess for y'(a)  */
    del : read( " first guess y'(a) = " ),

    /* get first IVP solution */
    outL : rk4(dy1dy2,ivar,[A,del],xgrid),
    xL : take(outL,1),
    uL : take(outL,2),
    upL : take(outL,3),

    /* get second guess for y'(a)  */
    del : read( " second guess y'(a) = " ),

    /* get second IVP solution */
    outL : rk4(dy1dy2,ivar,[A,del],xgrid),
    vL : take(outL,2),
    vpL : take(outL,3),

    ub : last(uL),
    vb : last(vL),
    sdiff : ub - vb,
    /* check that sdiff is not "zero" so we can divide by it */
    if abs(sdiff) < small then (
        print(" sdiff = ub - vb  is too small; choose different values for y'(a) guesses. "),
        return(done)),
    /*  form  numerical solution y1L and its first derivative y2L */
    alpha : (B - vb)/sdiff,    /* scalar number needed to construct solution */
    y1L : alpha*uL + (1 - alpha)*vL,
    y2L : alpha*upL + (1 - alpha)*vpL,
    makelist( [xL[j], y1L[j], y2L[j] ], j, 1, length(xL)))$
```

and here is an example of use to solve Example1 (1.7).

```
(%i1) load(cp3);
(%o1) "c:/k3/cp3.mac"
```

```
(%i14) soln : linear1( [y2, -(1+y1)*%pi^2/4], [y1,y2], [0,1], [x,0,1,0.01] )$
 first guess y'(a) =
1;
 second guess y'(a) =
 1.5;
(%i18) fll(soln);
(%o18) [[0.0,0.0,3.141593],[1.0,1.0,-1.570796],101]

(%i15) xL : take(soln,1)$
(%i16) yL : take(soln,2)$
(%i17)  ypL : take(soln,3)$
(%i18) plot2d([[discrete,xL,yL],cos(%pi*x/2)+2*sin(%pi*x/2) -1], [x,0,1],
                   [style,[lines,2,1], [points,2,2,2]],
                   [legend,"numerical","exact"],[ylabel,"y"],
                                   [gnuplot_preamble,"set grid"])$
```

which shows numerical agreement for $y(x)$ similar to Figure 1. Then

```
(%i19) plot2d([[discrete,xL,ypL], %pi*cos(%pi*x/2)-%pi*sin(%pi*x/2)/2], [x,0,1],
                   [style,[lines,2,1], [points,2,2,2]],
                   [legend,"numerical","exact"],[ylabel,"dy/dx"],
                                   [gnuplot_preamble,"set grid"])$
```

shows agreement for the first derivative $y'(x)$ similar to Figure 2.

### 1.3 Example 2 using R

We use our homemade R functions **`myrk4`** and **`linear1`** to solve the linear 2nd order ode

$$y''(x) = -\frac{3\,p\,y(x)}{(p+x^2)^2}, \quad p = 10^{-5}, \quad -0.1 \le x \le 0.1, \tag{1.9}$$

with the boundary conditions

$$y(-0.1) = -\frac{0.1}{\sqrt{p+0.01}}, \quad y(0.1) = \frac{0.1}{\sqrt{p+0.01}}. \tag{1.10}$$

The analytic solution is

$$y_{an}(x) = \frac{x}{\sqrt{p+x^2}}. \tag{1.11}$$

For small **p** the solution is an approximate step function.

```
> source("cp3.R")
> ex2 = function(x,y) { c( y[2], -3*p*y[1]/(p + x^2)^2) }
> xL = seq (-0.1, 0.1, 0.001)
> p = 1e-5
> BC = c( -0.1/sqrt(p + 0.01), 0.1/sqrt(p + 0.01))
> soln = linear1(bc=BC, grid = xL, func = ex2)
 first guess y'(a) = 1
 second guess y'(a) = 1.5
> yL = soln[[1]]
> fll(yL)
  -0.9995004   0.9995004   201
> ypL = soln[[2]]
> fll(ypL)
  0.06719535   0.07319558   201
> plot(xL,yL, type = "l", lwd = 2, col = "blue", ylab = "y(x)", xlab = "x")
> mygrid()
> curve ( x/sqrt(p + x*x), -0.1, 0.1, add = TRUE, type = "p", col = "red")
> legend("bottomright", col = c("blue","red"), legend = c( "numerical", "exact"),
+                         lwd = 2)
```

which produces the plot

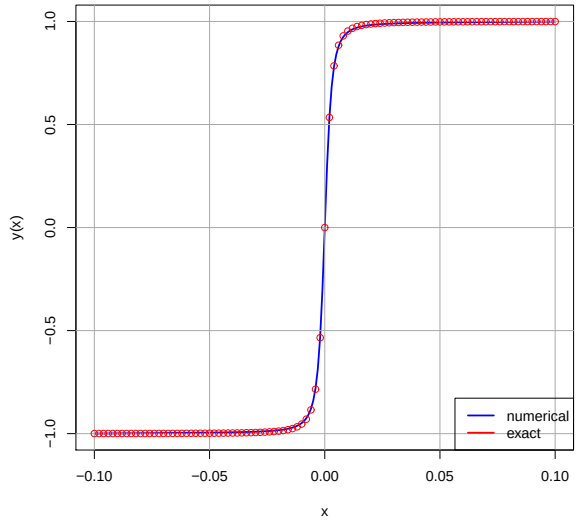


Figure 3: Example 2 Numerical $y(x)$ Compared with Analytic Solution

We can then plot the numerical first derivative returned by the solution.

```
> plot(xL,ypL, type = "l", lwd = 2, col = "blue", ylab = "dy/dx", xlab = "x")
> mygrid()
> curve ( p/(x^2+p)^(3/2), -0.1, 0.1, add = TRUE, type = "p", col = "red")
> legend("topright", col = c("blue","red"), legend = c( "numerical", "exact"),
+                           lwd = 2)
```

which produces the plot

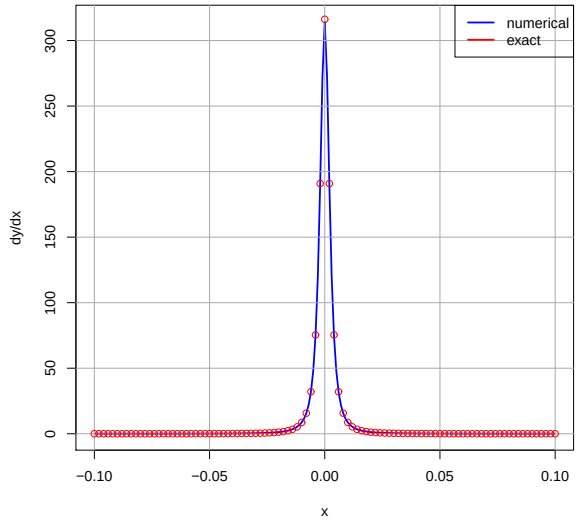


Figure 4: Example 2 Numerical $dy/dx$ Compared with Analytic Solution

### 1.4 Example 2 using Maxima

We use our homemade Maxima functions **rk4** and **linear1** to solve the Example 2 linear 2nd order ode BVP (1.9).

```
(%i1) load(cp3);
(%o1) "c:/k3/cp3.mac"
(%i2) p : 1e-5;
(%o2) 1.0E-5
(%i3) BC : [-0.1/sqrt(p + 0.01), 0.1/sqrt(p + 0.01)];
(%o3) [-0.9995,0.9995]
(%i4) soln : linear1( [y2, -3*p*y1/(p + x^2)^2], [y1,y2], BC, [x,-0.1,0.1,0.001] )$
 first guess y'(a) =
1;
 second guess y'(a) =
1.5;
(%i5) xL : take(soln,1)$
(%i6) fll(xL);
(%o6) [-0.1,0.1,201]
(%i7) yL : take(soln,2)$
(%i8) fll(yL);
(%o8) [-0.9995,0.9995,201]
(%i9) ypL : take(soln,3)$
(%i10) fll(ypL);
(%o10) [0.067195,0.073196,201]
(%i11) plot2d([[discrete,xL,yL], x/sqrt(p + x*x)], [x,-0.1,0.1],
            [style,[lines,2,1], [points,2,2,2]],
            [legend,"numerical","exact"],[ylabel,"y"],
                                [gnuplot_preamble,"set key bottom; set grid"])$
```

which produces a plot similar to that produced by **R** for $y(x)$, and we can then make a plot of $dy/dx$:

```
(%i12) plot2d([[discrete,xL,ypL], p/(x^2+p)^(3/2)], [x,-0.1,0.1],
            [style,[lines,2,1], [points,2,2,2]],
            [legend,"numerical","exact"],[ylabel,"dy/dx"],
                                [gnuplot_preamble,"set grid"])$
```

which produces a plot similar to that produced by **R** for $y'(x)$.

## 2 Shooting Methods for 2nd Order Linear or Nonlinear ODE BVPs

For the case of a 2nd order nonlinear ode BVP, we can use root finding methods to search for an initial value of $\delta = y'(a)$ such that both $y(a) = A$ and $y(b) = B$ to within some numerical tolerance.

### 2.1 Example 1 using rk4 and find_root with Maxima

Our example is

$$y''(x) = \frac{1}{8}\left(32 + 2\,x^3 - y(x)\,y'(x)\right), \quad 1 \le x \le 3 \tag{2.1}$$

subject to the boundary conditions

$$y(1) = 17, \quad y(3) = \frac{43}{3}. \tag{2.2}$$

The analytic solution is

$$y_{an}(x) = x^2 + \frac{16}{x}. \tag{2.3}$$

In addition to the Maxima function **find_root**, we use our homemade Runge-Kutta routine **rk4**.

Let $y_\delta(x)$ be the solution of the initial value problem for which $\delta = y'(1)$, and let $F(\delta)$, represented by **Fs(del)**, be the difference $y_\delta(3) - \frac{43}{3}$. We then seek a value of $\delta$ (represented by **del**) which results in a value of $Fs(\delta)$ close to zero (absolute value less than some prescribed tolerance). Recall that for a second order equation, **rk4** returns a list, each of whose elements has the form of the list **[x, y1(x), y2(x)]**, and we extract the second part of the last such list to get **y1(3)** produced by a given value of **del**.

The Maxima function **find_root** will abort if the supplied function whose zero is sought does not have opposite signs at the ends of the supplied interval **(del1,del2)**. By printing out a small table of values of the function **Fs**, we see that we can use the interval **(-15,-10)**. (We can also make a simple plot of **Fs** as a function of **del**; see below.)

```
(%i1) load(cp3);
(%o1) "c:/k3/cp3.mac"
(%i2) derivs : [y2,(32 + 2*x^3 - y1*y2)/8 ];
(%o2) [y2,(-y1*y2+2*x^3+32)/8]
(%i3) outL(del) := rk4(derivs,[y1,y2],[17,del],[x,1,3,0.01])$
(%i4) Fs(del0) := (second(last(outL(del0))) - 43/3)$
(%i5) for del:-20 thru 20 step 5 do
        print(del,"  ",Fs(del))$
-20    -4.027346
-15    -0.59963
-10    2.192145
-5    4.578477
0    6.685168
5    8.587376
10    10.33321
15    11.95519
20    13.47627
(%i6) delv : find_root(Fs, -15, -10);
(%o6) -14.0
(%i7) solnL : outL(delv)$
(%i8) fll(solnL);
(%o8) [[1.0,17.0,-14.0],[3.0,14.33333,4.222222],201]
(%i9) xL : take(solnL,1)$
(%i10) fll(xL);
(%o10) [1.0,3.0,201]
(%i11) y1L : take(solnL,2)$
(%i12) fll(y1L);
(%o12) [17.0,14.33333,201]
(%i13) plot2d([[discrete,xL,y1L], x^2 + 16/x], [x,1,3],
           [style,[lines,2,1], [points,2,2,2]],
           [legend,"numerical","exact"],[ylabel,"y"],
                              [gnuplot_preamble,"set grid"])$
```

which produces a plot in which the numerical solution visually agrees with the analytic solution:

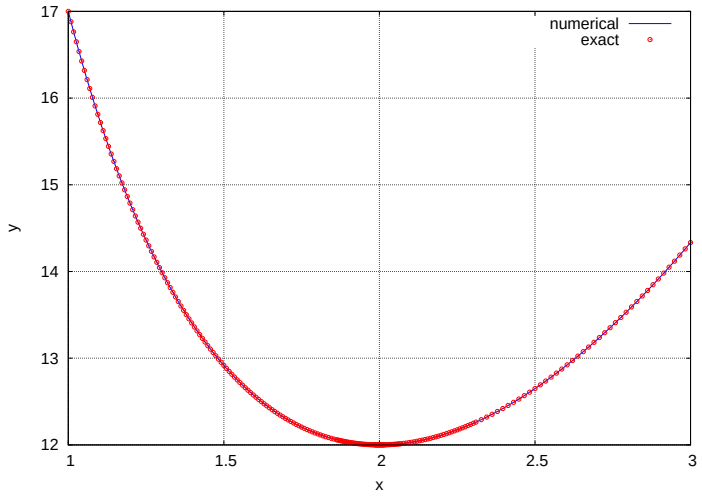


Figure 5: find_root Numerical $y(x)$ Compared with Analytic $y(x)$

Next we compare the numerical solution for the first derivative with the exact first derivative.

```
(%i14) y2L : take(solnL,3)$
(%i15) fll(y2L);
(%o15) [-14.0,4.222222,201]
(%i16) plot2d([[discrete,xL,y2L], 2*x - 16/x^2], [x,1,3],
           [style,[lines,2,1], [lines,2,2]],
           [legend,"numerical","exact"],[ylabel,"dydx"],
                              [gnuplot_preamble,"set key bottom;set grid"])$
```

which produces

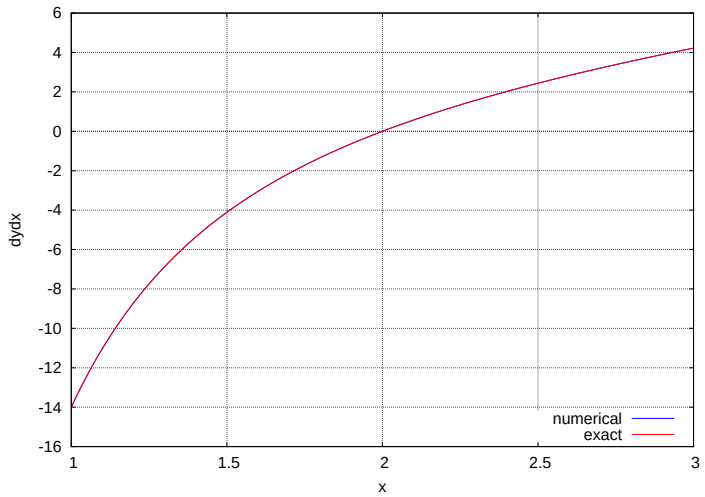


Figure 6: Numerical $y'(x)$ Compared with Analytic $y'(x)$

Here is an example of making a simple plot of **Fs** as a function of **del**:

```
(%i17) dL : makelist(del,del,-20,20,1)$
(%i18) FsL : map(Fs,dL)$
(%i19) time(%);
(%o19) [2.41]
(%i20) fll(FsL);
(%o20) [-4.027346,13.47627,41]
(%i21) plot2d([discrete,dL,FsL],[xlabel,"del"],[ylabel,"Fs"],
          [gnuplot_preamble,"set grid"])$
```

which produces the plot:

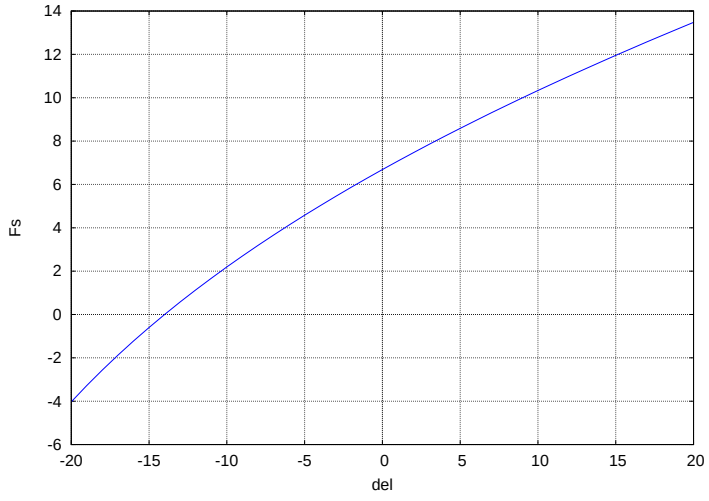


Figure 7: $F_s(\delta)$ versus $\delta$

## 2.2 Example 1 using myrk4 and uniroot with R

Our example is (repeating the Maxima introduction in the previous section):

$$y''(x) = \frac{1}{8}\left(32 + 2\,x^3 - y(x)\,y'(x)\right), \quad 1 \le x \le 3 \tag{2.4}$$

subject to the boundary conditions

$$y(1) = 17, \quad y(3) = \frac{43}{3}. \tag{2.5}$$

The analytic solution is

$$y_{an}(x) = x^2 + \frac{16}{x}. \tag{2.6}$$

We gave an example of using **uniroot** in Chapter 1. In addition to the **R** function **uniroot**, we use our homemade Runge-Kutta routine **myrk4(init,grid,func)**, which was introduced in Chapter 2, and which we have used in the linear method examples above. Recall that **myrk4** expects the output of the derivatives function **func** to be a R vector of the form **c(y[2], y'' as a function of x, y[1], and y[2])**. This derivatives function is called **derivs(x,y)**, and is defined by the line

```
derivs = function(x,y) {  c(y[2], (32 + 2*x^3 - y[1]*y[2])/8 ) }
```

below.

Let $y_\delta(x)$ be the solution of the initial value problem for which $\delta = y'(1)$, and let $F(\delta)$, represented by **Fs(del)**, be the difference $y_\delta(3) - \frac{43}{3}$. We then seek a value of $\delta$ (represented by **del**) which results in a value of $Fs(\delta)$ close to zero (absolute value less than some prescribed tolerance). Recall that for a second order equation, **myrk4** returns a R list, of the form **list( y1L, y2L)**, and we need **last(y1L)** to get **y1(3)** produced by a given value of **del**. The homemade function **last** (as well as **fll**) is in the code file **cp3.R**.

The **R** function **uniroot** will abort if the supplied function whose zero is sought does not have opposite signs at the ends of the supplied interval **(del1,del2)**. By printing out a small table of values of the function **Fs**, we see that we can use the interval **(-15,-10)**. (We can also make a simple plot of **Fs** as a function of **del**; see below.)

```
> source("cp3.R")
> derivs = function(x,y) {  c(y[2], (32 + 2*x^3 - y[1]*y[2])/8 ) }
> xL = seq(1, 3, 0.01)
> fll(xL)
  1   3   201
> outL = function(del) { myrk4( init = c(17, del), grid = xL, func = derivs ) }
> Fs = function(del) { last (outL (del) [[1]]) - 43/3 }
> for (del in seq(-20, 20, by = 5)) cat (del,"  ",Fs(del),"\n")
-20    -4.027346
-15    -0.5996257
-10    2.192145
-5    4.578477
0    6.685168
5    8.587376
10    10.33321
15    11.95519
20    13.47627
> delv = uniroot(Fs, c(-15, -10), tol = 1e-10 )$root
> delv
[1] -14
> solnL = outL(delv)
> yL = solnL[[1]]
> fll(yL)
  17   14.33333   201
> ypL = solnL[[2]]
> fll(ypL)
  -14   4.222222   201
> plot( xL, yL, type = "l", lwd = 2, col = "blue", xlab = "x", ylab = "y")
> mygrid()
> curve( x^2 + 16/x,1, 3,  type = "p", col = "red" , add = TRUE)
> legend("topright", col = c("blue","red"), legend = c( "numerical", "exact"),
+                            lwd = 2)
```

which produces the plot

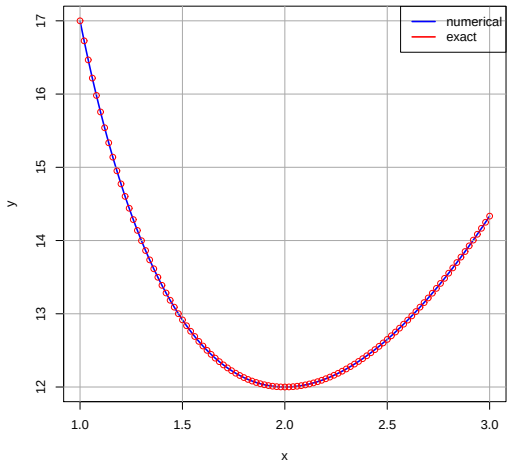


Figure 8: uniroot Numerical $y(x)$ Compared with Analytic $y(x)$

Next we plot the numerical first derivative:

```
> plot( xL, ypL, type = "l", lwd = 2, col = "blue", xlab = "x", ylab = "dydx")
> mygrid()
> curve( 2*x - 16/x^2,1, 3,  type = "p", col = "red" , add = TRUE)
> legend("bottomright", col = c("blue","red"), legend = c( "numerical", "exact"),
+                           lwd = 2)
```

which produces

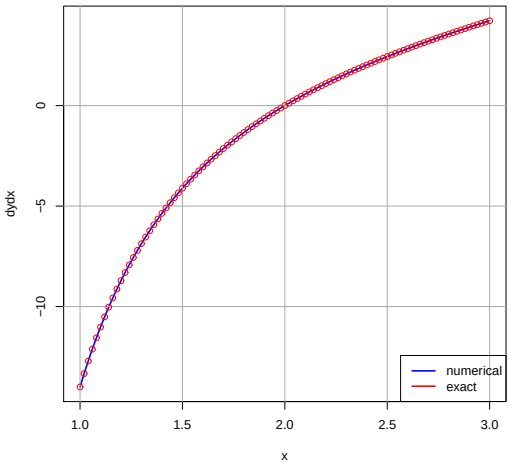


Figure 9: Numerical $y'(x)$ Compared with Analytic $y'(x)$

## 2.3 Example 2 using Secant Search with Maxima

We can use a secant search method, instead of **`find_root`**. For the example we consider here, the secant search method requires two guesses for the initial value of $y'(x)$. The example we consider is

$$y'' = 2\,y^3, \quad -1 \le x \le 0, \quad y(-1) = 1/2, \quad y(0) = 1/3. \tag{2.7}$$

The analytic solution is $y_{an}(x) = 1/(x+3)$.

We discussed the secant search method in Chapter 1. The Maxima code **shoot2(del0,del1,tol)** is available in **cp3.mac**, applies specifically to this particular example, and we list it here. We use the secant search method to find the value of $\delta = y'(-1)$ such that $F(\delta) = y_\delta(0) - 1/3 = 0$.

```
/* shoot2: uses secant search method
   example of method that works both for linear and nonlinear
   ODE's;  this example is specific to the nonlinear ode:
   y'' = 2*y^3,   -1 <= x <= 0,
   with boundary conditions:   y(-1) = 1/2,   y(0) = 1/3.
   Use variables:   y1 = y, y2 = y' .
   The grid size h = 0.01 is hardwired.
   Uses secant search method to
   seek a value of del = y'(-1)  such that F(del) = y1(del, x=0) -1/3 = 0 within tolerance tol .
   shoot2 returns the final Runge-Kutta solution list returned by our homemade rk4.
    The analytic solution is y(x) = 1/(3+x).
   */

shoot2(del0,del1,tol) :=
block([del2, F0, F1, F2, outL, j, jmax:10, h:0.01, yinit:1/2,
                    yfinal:1/3,  dy1dy2, ivar, xrange, numer],numer:true,

    dy1dy2 : [y2, 2*y1^3],
    ivar : [y1,y2],
    xrange : [x, -1, 0, h],

   /* get first trial solution */
    outL : rk4(dy1dy2,ivar,[yinit, del0], xrange),
    F0 : second(last(outL)) - yfinal,

    /* get second trial solution */
    outL : rk4(dy1dy2,ivar,[yinit, del1], xrange),
    F1 : second(last(outL)) - yfinal,

    /* update adjustable parameter del using secant rule */
    for j thru jmax do (
        del2 : del1 - F1*(del1 - del0)/(F1-F0),
        outL : rk4(dy1dy2,ivar,[yinit, del2], xrange),
        F2 : second(last(outL)) - yfinal,
        print(" j = ",j, " del2  = ",del2," F2 = ",F2),

        if abs(F2)   <   tol then return(),  /* escape from loop */
        F0 : F1,            /* rotate values */
        F1 : F2,
        del0 : del1,
        del1 : del2),
    outL)$
```

Here is the result we get using $\delta_0 = 0$, $\delta_1 = -1/2$, and using **tol = 1e-6**. The code prints out the iteration number, the current value of $\delta$, and the current value of $F(\delta)$. The code returns the complete output of **rk4** for the final value of $\delta$.

```
(%i1) load(cp3);
(%o1) "c:/k3/cp3.mac"
(%i2) soln : shoot2(0,-0.5,1e-6)$
 j =  1  del2  =  -0.26225  F2 =  -0.01433
 j =  2  del2  =  -0.24948  F2 =  6.084312E-4
 j =  3  del2  =  -0.25  F2 =  -1.4480274E-6
 j =  4  del2  =  -0.25  F2 =  -1.4686691E-10
(%i3) fll(soln);
(%o3) [[-1.0,0.5,-0.25],[0.0,0.33333,-0.11111],101]
(%i4) xL : take(soln,1)$
(%i5) fll(xL);
(%o5) [-1.0,0.0,101]
```

```
(%i6) y1L : take(soln,2)$
(%i7) fll(y1L);
(%o7) [0.5,0.33333,101]
(%i8) plot2d([ [discrete,xL,y1L], 1/(x+3)],[x, -1, 0],[xlabel,"x"],[ylabel,"y"],
               [style,[lines,2]], [legend,"numerical","analytic"],[gnuplot_preamble,"set grid"])$
```

which shows agreement of the numerical solution with the analytic:

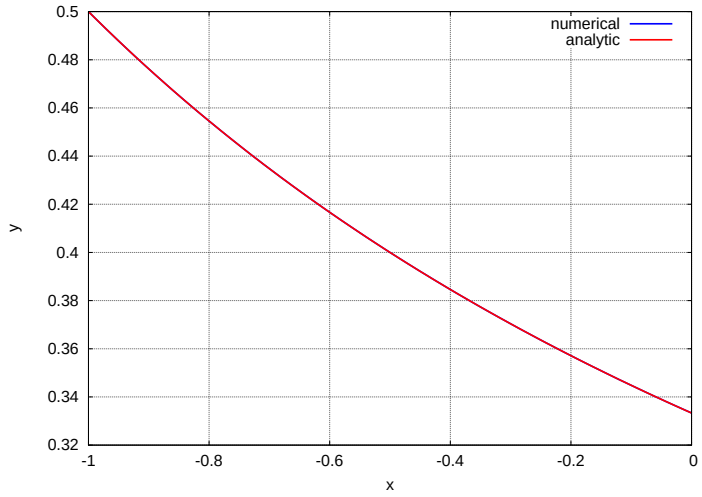


Figure 10: Secant Search $y(x)$ Compared with Analytic Solution

## 2.4 Example 2 using Secant Search with R

We can use a secant search method, instead of **uniroot**. For the example we consider here, the secant search method requires two guesses for the initial value of $y'(x)$. The example we consider is

$$y'' = 2\,y^3, \quad -1 \le x \le 0, \quad y(-1) = 1/2, \quad y(0) = 1/3. \tag{2.8}$$

The analytic solution is $y_{an}(x) = 1/(x+3)$.

We discussed the secant search method in Chapter 1. The R code **shoot2(del0,del1,tol)** is available in **cp3.R**, applies specifically to this particular example, and we list it here. We use the secant search method to find the value of $\delta = y'(-1)$ such that $F(\delta) = y_\delta(0) - 1/3 = 0$. **shoot2** needs a predefined derivatives function called **derivs**, and predefined grid values called **xL**.

```
## shoot2: uses secant search method
##   example of method that works both for linear and nonlinear
##   ODE BVPs;  this example is specific to the nonlinear ode:
##   y'' = 2*y^3, -1 <= x <= 0,
##   with boundary conditions:   y(-1) = 1/2,   y(0) = 1/3.
##     Use variables:   y[1] = y, y[2] = y' .
##   Uses secant search method to
##   seek a value of del = y'(-1)  such that F(del) = y1(del, x=0) -1/3 = 0 within tolerance tol .
##   shoot2 returns the final solution  list(yL, ypL) returned by our homemade myrk4.
##    The analytic solution is y(x) = 1/(3+x).
##   Needs predefined derivatives function called derivs, and predefined grid values called xL.

shoot2 = function(del0,del1,tol)  {
    jmax = 10
    yinit = 1/2
    yfinal = 1/3
```

```
  ## get first trial solution
   outL = myrk4(init = c(yinit,del0), grid = xL, func = derivs)
   F0 = last(outL[[1]]) - yfinal
   ## get second trial solution
   outL = myrk4(init = c(yinit,del1), grid = xL, func = derivs)
   F1 = last(outL[[1]]) - yfinal
   ## update adjustable parameter del using secant rule
   for ( j in seq(1,jmax,1) )  {
       del2 = del1 - F1*(del1 - del0)/(F1-F0)
       outL = myrk4(init = c(yinit,del2), grid = xL, func = derivs)
       F2 = last(outL[[1]]) - yfinal
       cat(" j = ",j, " del2  = ",del2," F2 = ",F2, "\n")
       if ( abs(F2) < tol ) break   ## escape from loop
       F0 = F1            ## rotate values
       F1 = F2
       del0 = del1
       del1 = del2}
   outL}
```

Here is the result we get using $\delta_0 = 0$, $\delta_1 = -1/2$, and **`tol = 1e-6`**. The code prints out the iteration number, the current value of $\delta$, and the current value of $F(\delta)$. The code returns the complete output of **`myrk4`** for the final value of $\delta$.

```
> source("cp3.R")
> derivs = function(x,y) {  c(y[2], 2*y[1]^3)  }
> xL = seq(-1, 0, 0.01)
> soln = shoot2(0,-0.5,1e-6)
 j =  1  del2  =  -0.2622537  F2 =  -0.01432995
 j =  2  del2  =  -0.249481  F2 =  0.0006084312
 j =  3  del2  =  -0.2500012  F2 =  -1.448027e-06
 j =  4  del2  =  -0.25  F2 =  -1.468672e-10
> is.list(soln)
[1] TRUE
> y1L = soln[[1]]
> fll(y1L)
  0.5   0.3333333   101
> plot(xL, y1L,type="l", col = "blue", lwd = 3,xlab = "x", ylab = "y")
> mygrid()
> curve(1/(x + 3),-1,0,col = "red", lwd = 3, add = TRUE)
> legend("topright", col = c("blue", "red"), lwd = 2,
+              legend = c("numerical", "analytic"), cex = 1.5)
```

which produces agreement between the numerical and analytical $y(x)$:

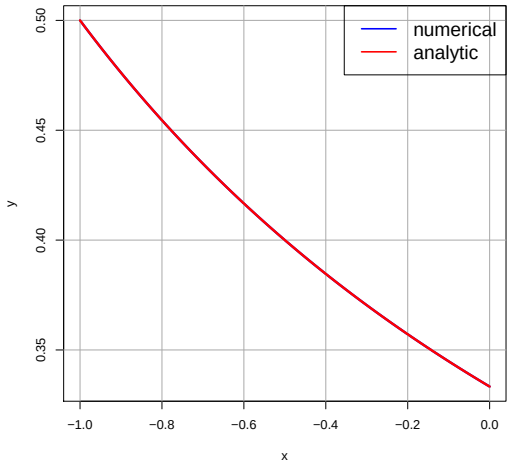


Figure 11: Secant Search $y(x)$ Compared with Analytic Solution

## 2.5 Example 3: Multiple Solutions, using rk4 and find_root with Maxima

The nonlinear ode boundary value problem

$$y'' = -e^{1+y}, \quad y(0) = 0, \quad y(1) = 0 \tag{2.9}$$

has two distinct solutions, each corresponding to a positive value of $y'(0)$. We use a simple shooting method, using Maxima's **rk4** and **find_root**.

```
(%i1) load(cp3);
(%o1) "c:/k3/cp3.mac"
(%i2) derivs : [y2, -exp(1 + y1) ];
(%o2) [y2,-%e^(y1+1)]
(%i3) outL(del) := rk4(derivs,[y1,y2],[0,del],[x,0,1,0.01])$
(%i4) ylast(del0) := second(last(outL(del0)))$
(%i5) delL : makelist(x,x,1,10,0.5);
(%o5) [1,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5,7.0,7.5,8.0,8.5,9.0,9.5,10.0]
(%i6) ylastL : map(ylast,delL);
(%o6) [-0.48292,-0.20965,0.023678,0.21304,0.35546,0.44936,0.49475,0.49319,
       0.44753,0.3615,0.23932,0.085257,-0.096623,-0.30263,-0.52955,-0.77461,
       -1.035488,-1.310241,-1.597251]
(%i7) plot2d( [[discrete,delL,ylastL], [discrete,delL,ylastL]], [xlabel,"del"],[ylabel,"y(1)"],
         [style,[lines,2], [points,1,1,1]],[gnuplot_preamble,"set grid"],
                 [legend,false])$
```

which produces the plot

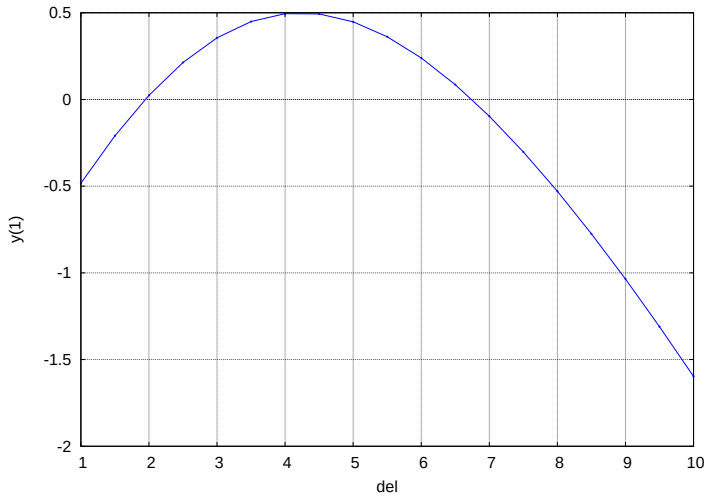


Figure 12: Dependence of $y(1)$ on $\delta = y'(0)$

In addition we can look for sign changes in $y(1)$ by making a simple table:

```
(%i8) for del thru 10 step 0.5 do
         print(del,"  ",ylast(del))$
1    -0.48292
1.5    -0.20965
2.0    0.023678
2.5    0.21304
3.0    0.35546
3.5    0.44936
4.0    0.49475
4.5    0.49319
5.0    0.44753
5.5    0.3615
6.0    0.23932
6.5    0.085257
7.0    -0.096623
7.5    -0.30263
8.0    -0.52955
8.5    -0.77461
```

```
9.0    -1.035488
9.5    -1.310241
10.0    -1.597251
```

We can then find the values of $y'(0)$, using **find_root**, which satisfy both boundary conditions, and then plot the corresponding solutions.

```
(%i9) del1 : find_root(ylast,1.5,2.5);
(%o9) 1.944773
(%i10) del2 : find_root(ylast,6.5,7);
(%o10) 6.743274
(%i11) outL1 : outL(del1)$
(%i12) yL1 : take(outL1,2)$
(%i13) fll(yL1);
(%o13) [0.0,1.7347235E-17,101]
(%i14) xL1 : take(outL1,1)$
(%i15) fll(xL1);
(%o15) [0.0,1.0,101]
(%i16) outL2 : outL(del2)$
(%i17) yL2 : take(outL2,2)$
(%i18) fll(yL2);
(%o18) [0.0,-2.220446E-16,101]
(%i19) xL2 : take(outL2,1)$
(%i20) fll(xL2);
(%o20) [0.0,1.0,101]
(%i21) plot2d([ [discrete,xL1,yL1], [discrete,xL2,yL2]],
                  [style,[lines,2]],[xlabel,"x"],
          [ylabel,"y"],[legend,"del = 1.94","del = 6.74"],
           [gnuplot_preamble,"set grid"])$
```

which produces the plot

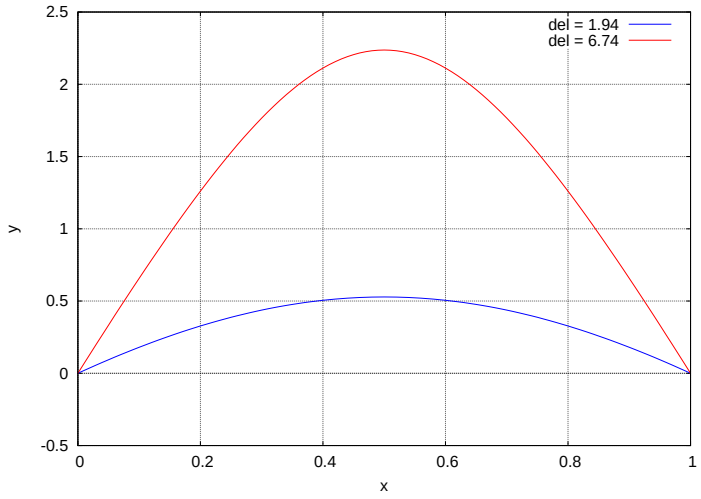


Figure 13: Example 1: Two Solutions

## 2.6 Example 4, Multiple Solutions, using rk4 and find_root with Maxima

Ascher, Mattheij, and Russell (Numerical Solution of Boundary Value Problems for Ordinary Differential Equations, SIAM, 1995) on page 89 consider the problem

$$y'' + \beta\, e^{y} = 0, \quad y(0) = y(1) = 0. \tag{2.10}$$

They assert two solutions exist if $\beta < \beta_c$, one solution exists if $\beta = \beta_c$, and no solutions exist if $\beta > \beta_c$, where

$$\beta_c = 3.51383. \tag{2.11}$$

They also assert that if $\beta = 1$, then solutions exist for $y'(0) = 0.549$ and for $y'(0) = 10.909$. We look at this case ($\beta = 1$) using Maxima with **rk4** and **find_root**, repeating the steps used in Example 1.

```
(%i1) load(cp3);
(%o1) "c:/k3/cp3.mac"
(%i2) derivs : [y2, -exp(y1) ];
(%o2) [y2,-%e^y1]
(%i3) outL(del) := rk4(derivs,[y1,y2],[0,del],[x,0,1,0.01])$
(%i4) ylast(del0) := second(last(outL(del0)))$
(%i5) delL : makelist(del,del,0.2,15,0.25)$
(%i6) fll(delL);
(%o6) [0.2,14.95,60]
(%i7) ylastL : map(ylast,delL)$
(%i8) fll(ylastL);
(%o8) [-0.29219,-2.798501,60]
(%i9) plot2d( [[discrete,delL,ylastL], [discrete,delL,ylastL]], [xlabel,"del"],[ylabel,"y(1)"],
          [style,[lines,2], [points,1,1,1]],[gnuplot_preamble,"set grid"],
                  [legend,false])$
```

which produces the plot

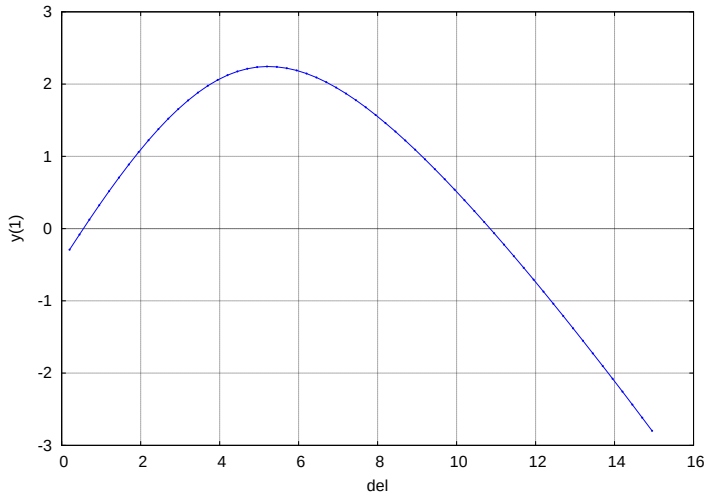


Figure 14: Values of $y(1)$ vs. $y'(0)$ for $\beta = 1$

Placing the cursor over the intersections of the curve with the $y(1) = 0$ line, we see that solutions exist for $y'(0) \approx 0.52, \quad 10.8$.

```
(%i10) for del:0.2 thru 0.8 step 0.1 do
         print(del,"  ",ylast(del))$
0.2    -0.29219
0.3    -0.20767
0.4    -0.12384
0.5    -0.040738
0.6    0.04161
0.7    0.12317
0.8    0.20389
(%i11) for del:10.5 thru 11 step 0.1 do
         print(del,"  ",ylast(del))$
10.5    0.21286
10.6    0.15201
10.7    0.090745
10.8    0.029066
10.9    -0.033014
11.0    -0.095484
(%i12) del1 : find_root(ylast,0.5,0.6);
(%o12) 0.54935
(%i13) del2 : find_root(ylast,10.8,10.9);
(%o13) 10.8469
```

We note that the use of **rk4** produced $\delta_2 = 10.8469$, instead of the value Ascher, et. al. assert (10.909). We can use the **R** package **deSolve** (using, say, ode) which would allow a more accurate integration, and produce a more accurate value of $\delta_2$, later. (As we

see in the next section, **ode** plus **uniroot** produces the same values as we found here using Maxima.)

We plot the solutions corresponding to the two values of $\delta = y'(0)$ found above.

```
(%i14) outL1 : outL(del1)$
(%i15) yL1 : take(outL1,2)$
(%i16) fll(yL1);
(%o16) [0.0,-2.6020852E-18,101]
(%i17) xL1 : take(outL1,1)$
(%i18) fll(xL1);
(%o18) [0.0,1.0,101]
(%i19) outL2 : outL(del2)$
(%i20) yL2 : take(outL2,2)$
(%i21) fll(yL2);
(%o21) [0.0,2.3731017E-15,101]
(%i22) xL2 : take(outL2,1)$
(%i23) fll(xL2);
(%o23) [0.0,1.0,101]
(%i24) plot2d([ [discrete,xL1,yL1], [discrete,xL2,yL2]],
                  [style,[lines,2]],[xlabel,"x"],
          [ylabel,"y"],[legend,"del = 0.549","del = 10.85"],
           [gnuplot_preamble,"set grid"])$
```

which produces the plot

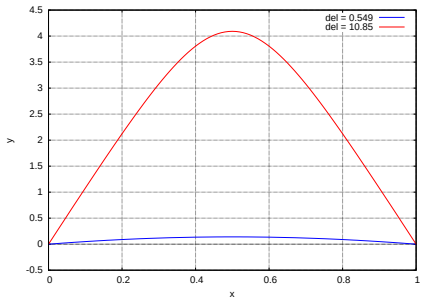


Figure 15: Example 2: Two Solutions for $\beta = 1$

We next consider the case $\beta = 5$, which should correspond to no solution.

```
(%i25) derivs : [y2, -5*exp(y1) ];
(%o25) [y2,-5*%e^y1]
(%i26) ylastL : map(ylast,delL)$
(%i27) fll(ylastL);
(%o27) [-1.745155,-6.228789,60]
(%i28) plot2d( [[discrete,delL,ylastL], [discrete,delL,ylastL]], [xlabel,"del"],[ylabel,"y(1)"],
          [style,[lines,2], [points,1,1,1]],[gnuplot_preamble,"set grid"],
                  [legend,false])$
```

which produces the plot

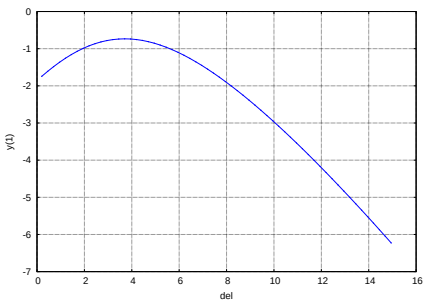


Figure 16: Example 2: Values of $y(1)$ vs. $y'(0)$ for $\beta = 5$

We see that there is no value of $y'(0)$ for which $y(1) = 0$ and hence no solution to the given BVP for $\beta = 5$.

## 2.7 Example 4, Multiple Solutions, using myrk4 and uniroot with R

By using the **deSolve** package, we can employ more accurate integrators than fourth order Runge-Kutta. The default method used by **ode** is **lsoda**. For ease of comparison with the Maxima method used above, we use similar names. We repeat the statement of the problem here.

$$y'' + \beta\, e^{y} = 0, \quad y(0) = y(1) = 0. \tag{2.12}$$

Ascher, et.al., assert two solutions exist if $\beta < \beta_c$, one solution exists if $\beta = \beta_c$, and no solutions exist if $\beta > \beta_c$, where

$$\beta_c = 3.51383. \tag{2.13}$$

They also assert that if $\beta = 1$, then solutions exist for $y'(0) = 0.549$ and for $y'(0) = 10.909$.

When using the **ode** integrator, the function used for **func** should return a R list containing a R vector of derivatives, as we discussed in Chapter 2. The **ode** integration routine returns a matrix object. You can view the first few rows using **head(outL,n=3)**. We first look at the case $\beta = 1$.

```
> source("cp3.R")
> derivs = function(x,y,p) {  list( c(y[2], -beta*exp( y[1] )))}
> ylast = function(del) last( outL(del) [,2])
> library(deSolve)
> xL = seq(0,1,0.01)
> beta = 1
> outL = function(del) { ode(y = c(0,del),times = xL, func = derivs, parms = NULL) }
> ylast(1)
[1] 0.3626952
> delL = seq(0.2, 15, 0.25)
> fll(delL)
  0.2   14.95   60
> ylastL = sapply(delL, ylast)
> fll(ylastL)
  -0.2921862   -2.79849   60
> plot( delL, ylastL, type = "l", lwd = 2, col = "blue",xlab = "del",
+                ylab = "y(1)")
> points( delL,ylastL,pch = 19,cex=0.5, col = "blue")
> mygrid()
```

which produces the plot

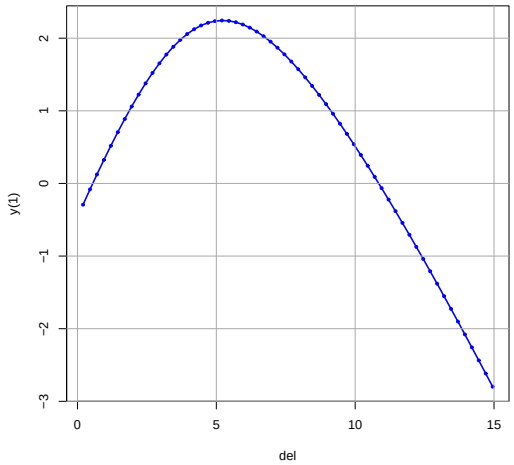


Figure 17: Values of $y(1)$ vs. $y'(0)$ for $\beta = 1$

which again shows two values of $y'(0)$ for which $y(1) = 0$. We see that solutions exist for $y'(0) \approx (0.5, 11)$.

We can print values of $y(1)$ versus $y'(0)$ as before to look for sign changes.

```
> for (del in seq(0.2,0.8,0.1)) cat(del,"  ",ylast(del),"\n")
0.2    -0.2921862
0.3    -0.2076695
0.4    -0.1238423
0.5    -0.04073761
0.6    0.04161019
0.7    0.1231653
0.8    0.2038904
> for (del in seq(10.5,11,0.1)) cat(del,"  ",ylast(del),"\n")
10.5    0.2128662
10.6    0.1520208
10.7    0.09075283
10.8    0.02907364
10.9    -0.03300593
11    -0.09547605
> del1 = uniroot(ylast,c(0.5,0.6))$root
> del1
[1] 0.5493529
> del2 = uniroot(ylast,c(10.8,10.9))$root
> del2
[1] 10.84691
```

which returns the same values found above using Maxima. We can now plot $y(x)$ corresponding to these two solutions for the value of $y'(0)$ for the case $\beta = 1$.

```
> outL1 = outL(del1)
> yL1 = outL1[,2]
> fll(yL1)
  0   2.223776e-07   101
> outL2 = outL(del2)
> yL2 = outL2[,2]
> fll(yL2)
  0   -6.625572e-10   101
> plot( xL, yL2, type = "l", lwd = 2, col = "blue",xlab = "x", ylab = "y")
> lines (xL, yL1, lwd = 2, col = "red")
> legend("topright", col = c("red", "blue"), legend = c("del=0.549","del=10.85"),cex=1.5)
> mygrid()
```

which produces the plot

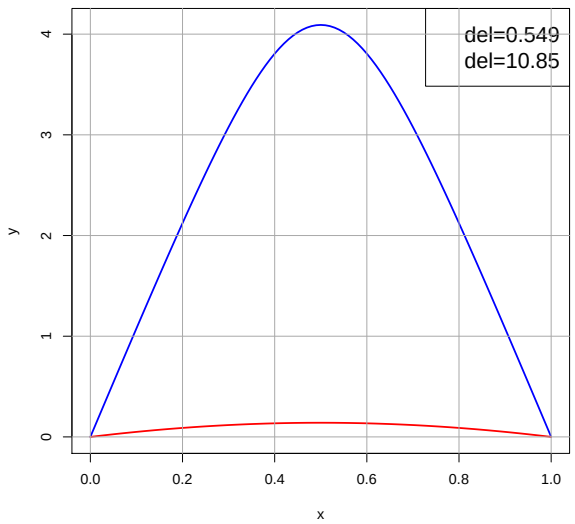


Figure 18: red:$v'(0) = 0.549$, blue:$v'(0) = 10.85$, for $\beta = 1$

The default integrator used by **ode** is **lsoda**, which uses as defaults **rtol = 1e-6** and **atol = 1e-6**. As an experiment, we define the function **outL** using **1e-15** instead, which requires we add two more arguments to **ode** in our definition of **outL(del)**. We find below that the value of **del1** and **del2** are unchanged, and the values of $y(1)$ are no closer to $0$.

```
> outL = function(del) { ode(y = c(0,del),times = xL, func = derivs,
+                            parms = NULL,rtol=1e-15,atol=1e-15) }
> del1 = uniroot(ylast,c(0.5,0.6))$root
> del1
[1] 0.5493529
> del2 = uniroot(ylast,c(10.8,10.9))$root
> del2
[1] 10.8469
> outL1 = outL(del1)
> yL1 = outL1[,2]
> fll(yL1)
  0   2.223765e-07   101
> outL2 = outL(del2)
> yL2 = outL2[,2]
> fll(yL2)
  0   -5.849995e-10   101
```

We next look at the case $\beta = 5$ which is larger than $\beta_c$.

```
> beta = 5
> ylastL = sapply(delL, ylast)
> fll(ylastL)
  -1.745155   -6.228776   60
> max(ylastL)
[1] -0.7350473
> plot( delL, ylastL, type = "l", lwd = 2, col = "blue",xlab = "del",
+                  ylab = "y(1)")
> points( delL,ylastL,pch = 19,cex=0.5, col = "blue")
> mygrid()
```

which produces the plot

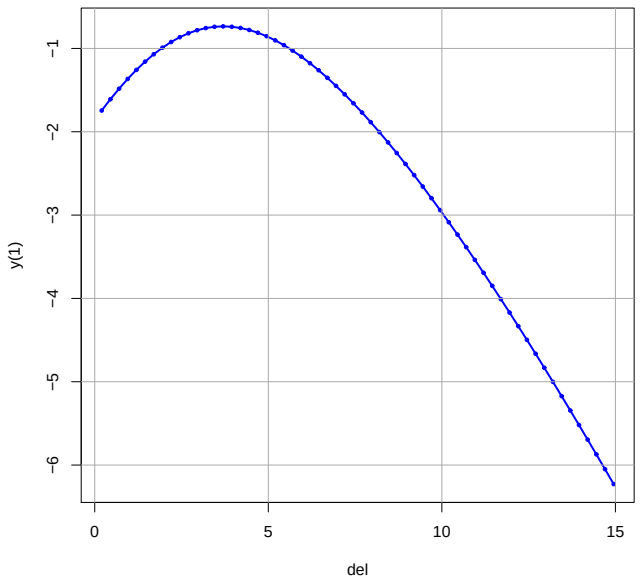


Figure 19: $y(1)$ vs. $y'(0)$ for $\beta = 5$; $\max(y(1)) = -0.735$

We see that there is no value of $y'(0)$ for which $y(1) = 0$ and hence no solution to the given BVP for $\beta = 5$.

# 3 Using the R Package bvpSolve

We quote loosely from parts of the beginning of Ch. 11, "Solving Boundary Value Problems in R", of the text **Solving Differential Equations in R**, by Karline Soetaert, Jeff Cash, and Francesca Mazzia, Springer-Verlag, 2012 (SCM).

> Boundary Value Problems (BVP) can be solved in **R** using shooting methods (**bvpshoot**), which use the solvers from the **R** package **deSolve**, MIRK (Mono-implicit Runge-Kutta) methods (**bvptwp**), and collocation methods (**bvpcol**). Their simplified syntax is
>
> ```
> bvpshoot ( yini, x, func, yend, parms, order, ...)
> byptwp ( yini, x, func, yend, parms, order, ... )
> bvpcol ( yini, x, func, yend, parms, order, ... ).
> ```
>
> where **func** is the derivative function and **parms** is the parameter vector or list. These two arguments are similar to the case of initial value problem solvers from the **R** package **deSolve**. However, the independent variable is called **x** here, rather than **times** for IVPs. The boundary conditions are specified by **yini** and **yend**, for the first and last point respectively. They are both a vector with length equal to the number of dependent variables, and having **NA** where the boundary value is not known.
>
> ...The default tolerances are $10^{-8}$ for all the codes. ...Functions **bvpshoot** and **bvptwp** can solve two-point boundary value problems only while **bvpcol** also finds solutions for multipoint problems.
>
> As **bvptwp** and **bvpcol** generally lead to more accurate solutions, and provide solutions where **bvpshoot** fails, the latter should normally not be used, unless to find good initial guesses for **bvptwp** or **bvpcol**.

Some indications of the theoretical basis of various methods (and references) are supplied by SCM in ch. 10, "Boundary Value Problems." They often refer to the authoritative text **Numerical Solution of Boundary Value Problems for Ordinary Differential Equations**, by Uri M. Ascher, Robert M. M. Mattheij, and Robert D. Russell, SIAM, 1995.

To use the powerful **R** tools for boundary value problems, you first need to install the **R** package **bvpSolve**, using, say the command

```
install.packages("bvpSolve")
```

and you also need the supporting packages **rootSolve** and **deSolve**.

## 3.1 Using bvpSolve's bvpshoot

An important argument in **bvpshoot** is **guess**, and the documentation (use **?bvpshoot** to see the html page of documentation) has the information:

```
guess:
guess for the value(s) of the unknown initial conditions;

if initial and final conditions are specified by yini and yend, then guess should
 contain one value for each NA in yini. The length of guess should thus equal
  the number of unknown initial conditions (=NAs in yini). If guess is not
  provided, a value = 0 is assumed for each NA in yini and a warning is printed.
```

In this section, we use **bvpshoot** to solve the linear BVP (1.9) we previously solved using **linear1** and **myrk4**:

$$y''(x) = -\frac{3\,p\,y(x)}{(p+x^2)^2}, \quad p = 10^{-5}, \quad -0.1 \le x \le 0.1, \tag{3.1}$$

with the boundary conditions

$$y(-0.1) = -\frac{0.1}{\sqrt{p+0.01}}, \quad y(0.1) = \frac{0.1}{\sqrt{p+0.01}}. \tag{3.2}$$

The analytic solution is

$$y_{an}(x) = \frac{x}{\sqrt{p+x^2}}. \tag{3.3}$$

For small **p** the solution is an approximate step function.

In the following work, we use **y[1]** to represent $y(x)$ and **y[2]** to represent $y'(x)$ inside the derivatives function **derivs**.

```
> source("cp3.R")
> library(bvpSolve)
Loading required package: rootSolve
Loading required package: deSolve
Attaching package: bvpSolve
> derivs = function(x,y,pars) { list( c(y[2], -3*p*y[1]/ (p + x^2)^2) ) }
> xL = seq (-0.1, 0.1, 0.001)
> p = 1e-5
> # initial and final condition; second conditions unknown
> init =  c(y = -0.1 / sqrt(p + 0.01), dy = NA)
> end  =  c(  0.1 / sqrt(p + 0.01), NA)
> # Solve bvp
> soln  =  bvpshoot(yini = init, x = xL, func = derivs, yend = end, guess = 1)
> is.matrix(soln)
[1] TRUE
> head(soln, n = 3)
          x          y          dy
[1,] -0.100 -0.9995004 0.009983301
[2,] -0.099 -0.9994902 0.010288639
[3,] -0.098 -0.9994798 0.010606540
> plot(soln, lwd = 2,col = "blue")
```

which produces the plots of $y(x)$ and $dy/dx$:

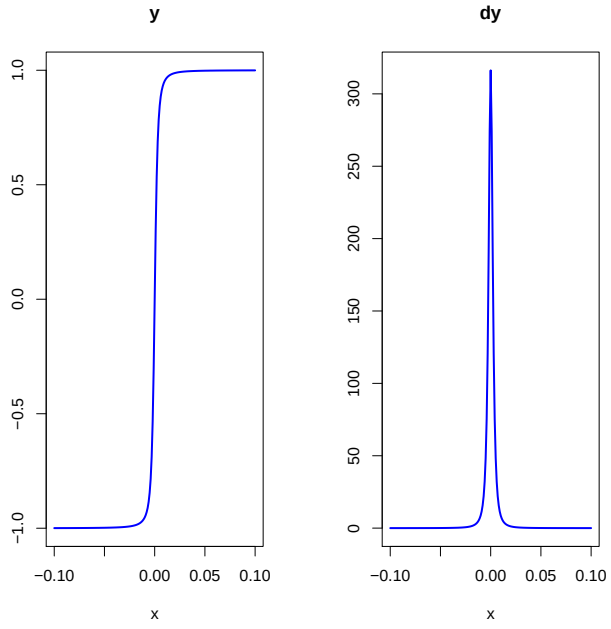


Figure 20: bvpshoot Default Plot of $y(x)$ and $dy/dx$

The first element of **yini** is the value of $y(a)$ and the second is the value of $y'(a)$, and likewise the first element of **yend** is the value of $y(b)$ and the second is the value of $y'(b)$. For any unknown values, use NA (not available).

You can provide names for the dependent variables in the specification of **yini** (as above) which will be used when making a table of values (using **head**, etc) and when making plots.

The derivatives function used for 'func' must return **list(c(y[2],y'' as a function of x,y[1], and y[2]))**, since the derivatives function must return the derivatives in the same order as that in which the initial and end conditions are defined.

Loading the package **bvpSolve** automatically changes some of the default behavior of **plot**: The plots of the dependent variables are by default plotted side by side, two to a row, using a **type="l"** style and with no grid.

Since we have provided names for the two dependent variables in the specification of **yinit**, we can use the following syntax to make a separate plot of $y(x)$, adding a grid (from **cp3.R**) and the exact solution, and a legend.

```
> plot(soln,which = "y", lwd = 2,col = "blue")
> mygrid()
> curve(x/sqrt(p + x^2), -0.1, 0.1,type = "p", col = "red", add = TRUE)
> legend("bottomright", col = c("blue","red"), legend = c( "numerical", "exact"),
+                           lwd = 2)
```

which produces the plot

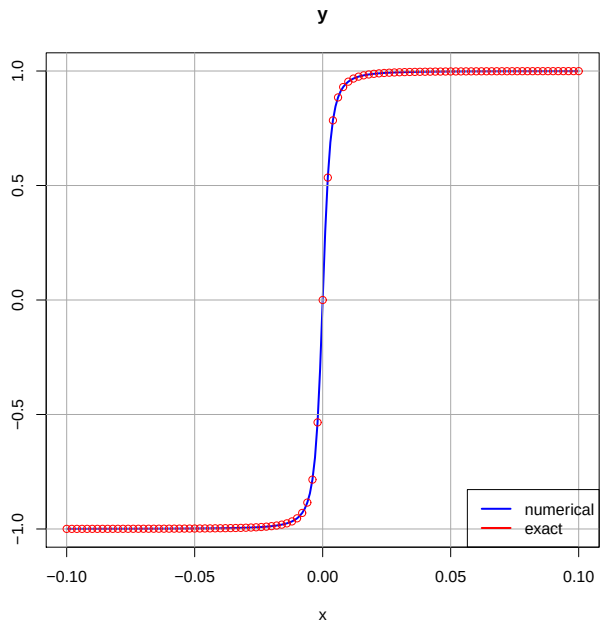


Figure 21: bvpshoot Plot of Numerical and Exact $y(x)$

In a similar manner, we can make a separate plot of $y'(x)$.

```
> plot(soln,which = "dy", lwd = 2,col = "blue")
> mygrid()
> curve ( p/(x^2+p)^(3/2), -0.1, 0.1, add = TRUE, type = "p", col = "red")
> legend("topright", col = c("blue","red"), legend = c( "numerical", "exact"),
+                           lwd = 2)
```

which produces the plot

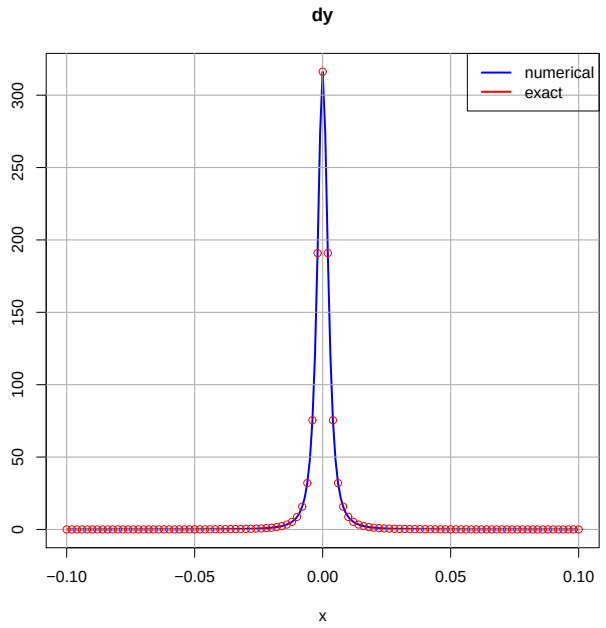


Figure 22: bvpshoot Plot of Numerical and Exact $dy/dx$

If you have not given names to the dependent variables in the **yinit** specification, you can revert to plotting the second and third columns of the matrix **soln** against the first column (the **x** values). You have two choices of syntax using numbers:

```
> plot(soln[,1],soln[,2],type = "l",lwd = 2,col = "blue",xlab = "x", ylab = "y(x)")
```

is equivalent to

```
> plot(soln,which = 1,type = "l",lwd = 2,col = "blue",xlab = "x", ylab = "y(x)")
```

and

```
> plot(soln[,1],soln[,3],type = "l",lwd = 2,col = "blue", xlab = "x", ylab = "dydx")
```

is equivalent to

```
> plot(soln,which = 2,type = "l",lwd = 2,col = "blue", xlab = "x", ylab = "dydx")
```

An alternative approach, if you have not assigned names using **yinit**, is to use **colnames(soln) = c("x","y","dy")** (after **soln** has been calculated) and then use **plot(soln, which = "y",...)** to select the column to be plotted. These names will then also be used with the **head** command.

## 3.2 Why You Should Use bvptwp Instead of bvpshoot

Instead of using $p = 10^{-5}$, as we did in the preceding problem, we use $p = 1$ over the domain $-1 \le x \le 1$, and find that **bvpshoot** cannot return the correct solution even if the starting guess for $y'(-1)$ is the exact value implied by the analytic solution.

Thus, we solve the problem

$$y''(x) = -\frac{3\,y(x)}{(1+x^2)^2}, \quad -1 \le x \le 1, \tag{3.4}$$

with the boundary conditions

$$y(-1) = -\frac{1}{\sqrt{2}}, \quad y(1) = \frac{1}{\sqrt{2}}. \tag{3.5}$$

The analytic solution is

$$y_{an}(x) = \frac{x}{\sqrt{1+x^2}}. \tag{3.6}$$

We begin, as before, with the use of **bvpshoot**.

```
> source("cp3.R")
> library(bvpSolve)
Loading required package: rootSolve
Loading required package: deSolve
Attaching package: bvpSolve
> derivs = function(x,y,pars) { list( c(y[2], -3*y[1]/ (1 + x^2)^2) ) }
> xL = seq (-1, 1, 0.01)
> init = c(y = -1/sqrt(2), dy = NA)
> end = c( 1/sqrt(2), NA)
> #  bvpshoot with guess = 1
> soln  =  bvpshoot(yini = init, x = xL, func = derivs, yend = end, guess = 1)
> is.matrix(soln)
[1] TRUE
> head(soln, n = 3)
         x          y       dy
[1,] -1.00 -0.7071068 1.000000
[2,] -0.99 -0.6970802 1.005319
[3,] -0.98 -0.6870003 1.010667
> plot(soln,which = "y", lwd = 2,col = "blue")
> mygrid()
> curve(x/sqrt(1 + x^2), -1, 1,type = "p", col = "red", add = TRUE)
> #  bvpshoot with guess = exact value of y'(-1) = 1/2^(3/2) = 0.3535...
> soln2  =  bvpshoot(yini = init, x = xL, func = derivs, yend = end, guess = 1/2^(3/2))
> head(soln2, n = 3)
         x          y        dy
[1,] -1.00 -0.7071068 0.8808722
[2,] -0.99 -0.6982715 0.8861953
[3,] -0.98 -0.6893827 0.8915575
> #  bvpshoot with guess = exact, returns soln2 with y'(-1) = 0.881
> lines( soln2[,"x"], soln2[,"y"] ,lwd = 2, col = "green")
> #  use bvptwp instead of bvpshoot, call it soln3
> soln3 = bvptwp(yini = init, x = xL, func = derivs, yend = end)
> head(soln3,n=3)
         x          y        dy
[1,] -1.00 -0.7071068 0.3535500
[2,] -0.99 -0.7035446 0.3588932
[3,] -0.98 -0.6999286 0.3643166
> #  bvptwp gets correct value for  y'(-1)
> lines( soln3[,"x"], soln3[,"y"] ,lwd = 2, col = "black")
> legend("bottomright", col = c("blue", "green", "red", "black"),
+        legend = c( "guess = 1", "guess = exact", "analytic", "bvptwp"), lwd = 2,cex = 1.4)
```

which produces the plot

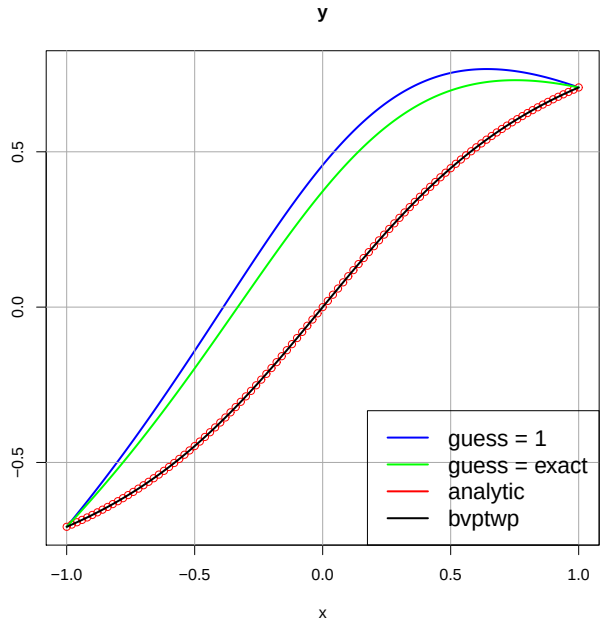


Figure 23: Example Showing bvptwp Is Better than bvpshoot

### 3.3 Poisson's Equation $\nabla^2\Phi = -4\,\pi\,\rho$ with Spherically Symmetric Charge Distribution $\rho(r)$

To find the electrostatic potential $\Phi$ generated by a given localized charge distribution $\rho(r)$, one uses Poisson's equation (we use Gaussian c.g.s. units):

$$\nabla^2\Phi = -4\,\pi\,\rho, \tag{3.7}$$

which, for a spherically symmetric $\rho$ and $\Phi$, simplifies to

$$\frac{1}{r^2}\,\frac{d}{dr}\left(r^2\,\frac{d\,\Phi}{dr}\right) = -4\,\pi\,\rho. \tag{3.8}$$

The substitution

$$\Phi(r) = r^{-1}\,\phi(r) \tag{3.9}$$

results in an ODE for $\phi$

$$\frac{d^2\phi}{dr^2} = -4\,\pi\,r\,\rho. \tag{3.10}$$

Let's assume the charge distribution is (using dimensionless variables ):

$$\rho(r) = \frac{1}{8\,\pi}\,e^{-r} \tag{3.11}$$

which corresponds to a total charge

$$Q = \int \rho(r)\,d^3r = \int_0^\infty \rho(r)\,4\,\pi\,r^2 dr = 1 \tag{3.12}$$

We can use Maxima to check this integral:

```
(%i1) rho : exp(-r)/8/%pi$
(%i2) integrate(rho*4*%pi*r^2,r,0,inf);
(%o2) 1
```

Since $\rho$ has no singular behavior at the origin ( there is no point charge there), $\Phi$ is regular there, which implies that $\phi = r\,\Phi$ vanishes at $r = 0$. And since the total charge $Q = 1$, the potential $\Phi$ behaves as $\Phi \to 1/r$ for large $r$. This then determines the large $r$ behavior of $\phi$: $\phi \to 1$.

The boundary value problem to be solved is then

$$\frac{d^2\phi}{dr^2} = -\frac{1}{2}\,r\,e^{-r}, \quad \phi(0) = 0, \quad \phi(\infty) = 1. \tag{3.13}$$

The exact solution to this problem (including the correct boundary conditions) is

$$\phi(r) = 1 - \frac{1}{2}\,(r+2)\,e^{-r}. \tag{3.14}$$

We can use Maxima to derive this exact solution. Maxima returns solutions to the inhomogeneous ODE which contain constants called **%k1** and **%k2**, and **%k1** is evaluated below using $\phi(0) = 0$.

```
(%i3) de : 'diff(phi,r,2) + r*exp(-r)/2;
(%o3) r*%e^-r/2+'diff(phi,r,2)
(%i4) gsoln : ode2(de,phi,r);
(%o4) phi = -(r+2)*%e^-r/2+%k2*r+%k1
(%i5) phi0 : rhs(gsoln),r=0;
(%o5) %k1-1
(%i6) solve(phi0,%k1);
(%o6) [%k1 = 1]
(%i7) gsoln : gsoln,%;
(%o7) phi = -(r+2)*%e^-r/2+%k2*r+1
```

Imposing the condition that for large $r$ the solution be bounded implies that the constant **%k2** must be zero.

The meaning of "large $r$" is determined by the source term behavior as a function of $r$:

```
> source("cp3.R")
> curve(-r*exp(-r)/2, 0, 20, xname = "r",  ylab = "d2phi" ,lwd = 2)
> mygrid()
```

which produces the plot

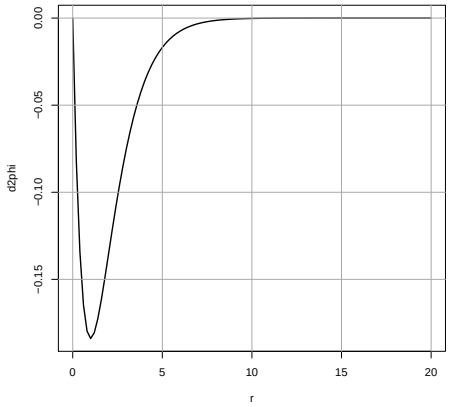


Figure 24: $\phi''(r) = -r\,exp(-r)/2$

We should be able to produce good results if we impose the large $r$ boundary condition at $r = 20$.

```
> library(bvpSolve)
Loading required package: rootSolve
Loading required package: deSolve
Attaching package: bvpSolve

> derivs = function(r,phi,pars) { list( c(phi[2], -r*exp(-r)/2 )) }
> rL = seq (0, 20, 0.1)
> fll(rL)
  0   20   201
> init = c(phi = 0, dphi = NA)
> end = c( 1, NA)
> soln = bvptwp(yini = init, x = rL, func = derivs, yend = end)
> colnames(soln) = c("r", "phi", "dphi")
> head(soln, n = 3)
      r         phi      dphi
[1,] 0.0 0.00000000 0.4999999
[2,] 0.1 0.04992073 0.4976605
[3,] 0.2 0.09939620 0.4912384
> plot(soln,which = "phi", lwd = 2,col = "blue", xlab = "r")
> mygrid()
> curve(1 - (r+2)*exp(-r)/2, 0, 20, xname = "r", type = "p", col = "red", add = TRUE)
> legend("bottomright", col = c("blue",  "red"),
+         legend = c( "bvptwp",  "analytic"), lwd = 2,cex = 1.4)
```

which produces the plot

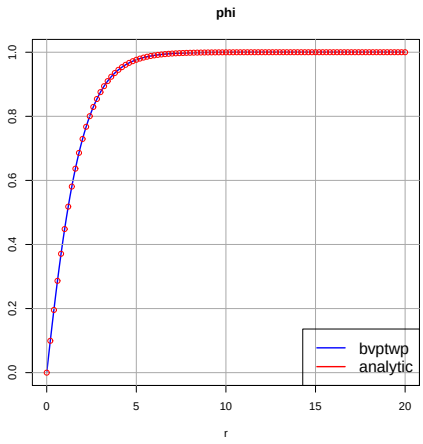


Figure 25: Example of Solving Poisson's Equation in One Dimension

## 3.4 $(\nabla^2 - a^2)\,\Phi = -4\,\pi\,\rho$ with Spherically Symmetric $\rho(r)$

Spherically symmetric solutions of the equation

$$\left(\nabla^2 - a^2\right)\,\Phi = -4\,\pi\,\rho \tag{3.15}$$

which, for a spherically symmetric $\rho$ and $\Phi$, and with the substitution

$$\Phi(r) = r^{-1}\,\phi(r) \tag{3.16}$$

results in an ODE for $\phi$

$$\phi'' - a^2\,\phi = -4\,\pi\,r\,\rho(r), \quad \phi(0) = 0, \quad \phi(\infty) = 0 \tag{3.17}$$

We again assume (using dimensionless variables ):

$$\rho(r) = \frac{1}{8\,\pi}\,e^{-r} \tag{3.18}$$

We change our notation: $\phi(r) \to y(x)$, and use **`bvpSolve`** with the BVP

$$y'' - a^2\,y = -\frac{1}{2}\,x\,e^{-x}, \quad y(0) = 0, \quad y(\infty) = 0 \tag{3.19}$$

```
> source("cp3.R")
> library(bvpSolve)
Loading required package: rootSolve
Loading required package: deSolve

> derivs = function(x,y,pars) {
+         list (  c(y[2], y[1] - x*exp(-x)/2) ) }
> xL = seq (0, 20, 0.1)
> init = c(y=0,dy=NA)
> end = c(0,NA)
> soln = bvptwp(yini = init, x = xL, func = derivs, yend = end)
> head(soln,n=3)
      x          y        dy
[1,] 0.0 0.00000000 0.1249999
[2,] 0.1 0.01244153 0.1232840
[3,] 0.2 0.02456195 0.1187159
> plot(soln,which = "y", lwd = 3,col = "blue")
> mygrid()
> curve(x*(1+x)*exp(-x)/8, 0, 20,  type = "p", col = "red", add = TRUE)
> legend("topright", col = c("blue", "red"), lwd=2,
+           legend = c("numerical", "analytic"),cex = 1.5)
```

which shows agreement with the analytic solution:

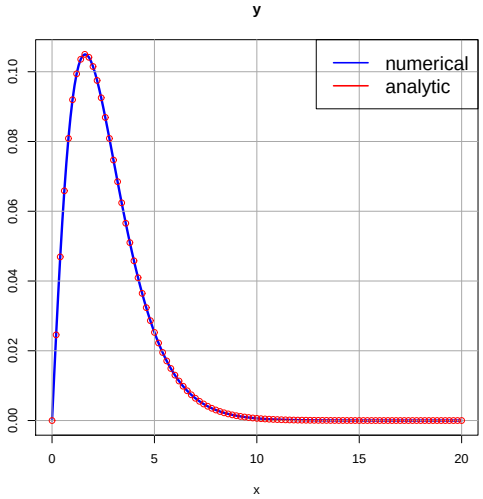


Figure 26: Numerical and Analytic $y(x)$

## 3.5 Solving a Potentially Stiff ODE BVP

Here we illustrate the use of **bvptwp** to solve a potentially stiff problem for two small values of $\varepsilon$.

```
> setwd("c:/k3")
> getwd()
[1] "c:/k3"
> source("cp3.R")
> library(bvpSolve)
Loading required package: rootSolve
Loading required package: deSolve
> stiff1 = function(x, y, pars) {
+     list(c(  y[2],
+            1/eps * ( -x*y[2] + y[1] - (1 + eps*pi*pi) *
+              cos(pi*x)  - pi*x*sin(pi*x)))) }
> eps = 0.01
> soln1 = bvptwp( yini = c (y1 = -1, y2 = NA), yend = c(y1 = 1, y2 = NA),
+                       func = stiff1,x = seq(-1, 1, by = 0.01))
> head(soln1, n=3)
         x        y1          y2
[1,] -1.00 -1.000000 2.45475e-11
[2,] -0.99 -0.999507 9.86798e-02
[3,] -0.98 -0.998027 1.97262e-01
> plot(soln1,lwd = 2)
```

The **plot** command above, by default, produces the plot of both $u(x)$ (represented by **y1**) and $u'(x)$ (represented by **y2**) for $\varepsilon = 0.01$:

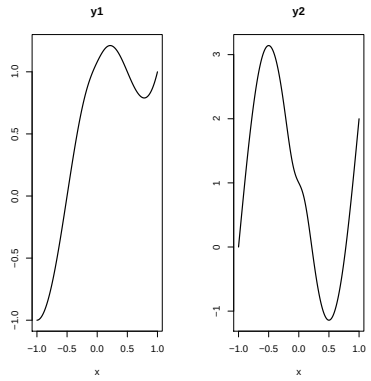


Figure 27: bvptwp Numerical Solution for $\varepsilon = 0.01$

To show a plot of just $u(x) = y_1(x)$ we use

```
> plot(soln1,which = "y1", lwd = 2, col = "blue")
> mygrid()
```

which produces the plot

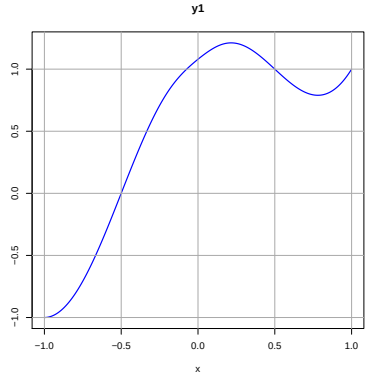


Figure 28: bvptwp Numerical Solution $u(x) = y_1(x)$ for $\varepsilon = 0.01$

**soln2** uses $\varepsilon = 0.0001$. The first attempt uses the default tolerance $10^{-8}$, and a mesh point storage error is returned. The second attempt uses the flag **atol = 1e-5** to override the default tolerance and succeeds. Both **soln1** and **soln2** are then displayed in the same plots.

```
> eps = 0.0001
> soln2 = bvptwp( yini = c (y1 = -1, y2 = NA), yend = c(y1 = 1, y2 = NA),
+                       func = stiff1,x = seq(-1, 1, by = 0.01))
Error in bvpsolver(1, yini, x, func, yend, parms, order, ynames, xguess,  :
  The Expected No. Of mesh points Exceeds Storage Specifications.
> soln2 = bvptwp( yini = c (y1 = -1, y2 = NA), yend = c(y1 = 1, y2 = NA),
+                       func = stiff1,x = seq(-1, 1, by = 0.01), atol = 1e-5)
> head(soln2, n=3)
            x        y1          y2
[1,] -1.000000 -1.000000 8.32662e-09
[2,] -0.996667 -0.999945 3.28981e-02
[3,] -0.993333 -0.999781 6.57926e-02
> plot(soln1, soln2, col = "black", lty = c("solid", "dashed"),lwd = 2)
```

which produces the plot

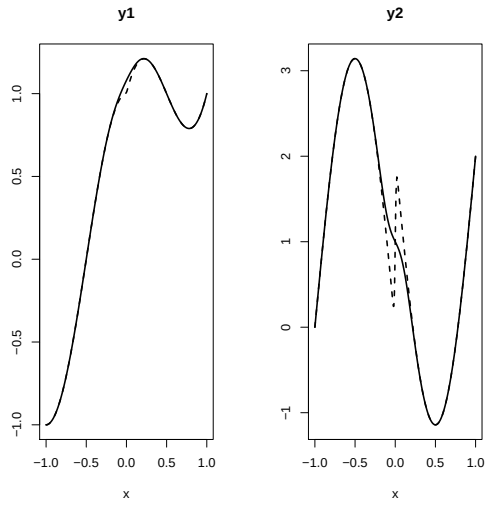


Figure 29: Solutions for $\varepsilon = 0.01$ (solid line) and $\varepsilon = 0.0001$ (dotted line)

# 4 Wave Equation with One Cartesian Spatial Dimension $\phi_{t\,t} = v^2\,\phi_{x\,x}$

## 4.1 Separation of Variables

The hyperbolic partial differential equation (the "wave equation")

$$\frac{\partial^2\,\phi(x,t)}{\partial\,t^2} = v^2\,\frac{\partial^2\,\phi(x,t)}{\partial\,x^2} \tag{4.1}$$

has "standing wave" solutions ("normal mode" solutions) with the separated variable product form

$$\phi(x,t) = y(x)\,T(t). \tag{4.2}$$

Insertion of this product form into (4.1) leads to a

$$\frac{y''(x)}{y(x)} = \frac{1}{v^2}\,\frac{T''(t)}{T(t)} = -k^2 \tag{4.3}$$

in which $k^2$ must be a constant, since the function of $x$ must equal the function of $t$ for arbitrary values of $x$ and $t$. We then need to solve

$$y''(x) + k^2\,y(x) = 0 \tag{4.4}$$

subject to physically imposed spatial boundary conditions, and

$$T''(t) + k^2\,v^2\,T(t) = 0 \tag{4.5}$$

to find the fundamental solutions with the separated product form.

## 4.2 Eigenvalues of the BVP: $y''(x) + k^2\,y(x) = 0$ for $y(0) = y(1) = 0$

The "normal modes" ("standing waves") of the transverse motions of a stretched string of uniform linear mass density, and with fixed ends, after separation of variables, and after introducing dimensionless, scaled, physical variables, must be solutions of the boundary value problem

$$\frac{d^2\,y}{d\,x^2} + k^2\,y(x) = 0, \quad y(0) = 0, \quad y(1) = 0, \tag{4.6}$$

in which $k$ is the positive, constant, scaled wavenumber, linearly related to the frequency of vibration, $y(x)$ is the scaled transverse displacement of the string, and $x$ is the scaled coordinate along the string.

This boundary value problem is also an eigenvalue problem, since a nontrivial solution (the trivial solution $y = 0$ exists for most values of $k$) exists only for particular values of $k$: $k_n$, which must be discovered. The normalization of the corresponding eigenfunctions $y_n(x)$ is not fixed by the above conditions, but can be chosen for convenience.

### 4.2.1 Analytic Solution Using Maxima

We can use Maxima's **ode2** and **ic2** to get the analytic solution for $y(0) = 0$ but arbitrary $k$ and $y'(0)$.

```
(%i1) de : 'diff(y,x,2) + k^2*y;
(%o1) 'diff(y,x,2)+k^2*y
(%i2) assume(k>0);
(%o2) [k > 0]
(%i3) gsoln : ode2(de,y,x);
(%o3) y = %k1*sin(k*x)+%k2*cos(k*x)
(%i4) psoln : ic2(gsoln,x=0,y=0,'diff(y,x)= yp0);
(%o4) y = sin(k*x)*yp0/k
(%i5) ysoln : rhs(%);
(%o5) sin(k*x)*yp0/k
```

We then have

$$y_{an}(x) = \frac{y'(0)\,\sin(k\,x)}{k}, \tag{4.7}$$

so the analytic end value is

$$y_{an}(1) = \frac{y'(0)\,\sin(k)}{k}. \tag{4.8}$$

The condition that $y(1) = 0$ then implies the discrete normal modes

$$k_n = n\,\pi, \quad y_n(x) \sim \sin(k_n\,x), \quad n = 1, 2, 3, \ldots \tag{4.9}$$

The fundamental standing wave solutions of the one dimensional wave equation (4.1) for these spatial boundary conditions are then

$$u_n(x,t) = \sin(k_n\,x)\,\sin(k_n\,v\,t), \quad k_n = n\,\pi, \quad n = 1, 2, 3, \ldots \tag{4.10}$$

and

$$v_n(x,t) = \sin(k_n\,x)\,\cos(k_n\,v\,t), \quad k_n = n\,\pi, \quad n = 1, 2, 3, \ldots \tag{4.11}$$

The behavior at time $t = 0$, for example, can be used to determine a particular linear combination of the fundamental solutions which describes the initial and subsequent behavior.

### 4.2.2 Eigenvalues using rk4 and find_root with Maxima

We use **rk4** and **find_root** to find the roots $k_n$ for small $n$ for the BVP (4.6). We search for the value of $k$ such that both $y(0) = 0$ and $y(1) = 0$. The routine **ylast(k,yp0,h)** returns the value of $y(1)$ produced for given values of $k$, $y'(0)$ (represented by **yp0**), and step size $h$. (We guess some reasonable value of $y'(0)$ – see below).

```
(%i6) load(cp3);
(%o6) "c:/k3/cp3.mac"
(%i7) ylast(k,yp0,h) :=
block([solnL, numer],numer:true,
     solnL : rk4([y2,-k^2*y1],[y1,y2],[0,yp0],[x,0,1,h]),
     last(take(solnL,2)))$
(%i8) ylast(1,1,0.01);
(%o8) 0.84147
(%i9) sin(1),numer;
(%o9) 0.84147
```

We can use **`ylast`** to make a plot of the numerical values of $y(1)$ as a function of $k$, assuming (here) the value $y'(0) = 1$.

```
(%i10) kL : makelist (k,k,1,15,0.5);
(%o10) [1,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5,7.0,7.5,8.0,8.5,9.0,9.5,
       10.0,10.5,11.0,11.5,12.0,12.5,13.0,13.5,14.0,14.5,15.0]
(%i11) ylastL : map (lambda ([k], ylast (k,1,0.01)), kL);
(%o11) [0.84147,0.665,0.45465,0.23939,0.04704,-0.10022,-0.1892,-0.21723,
       -0.19178,-0.12828,-0.046569,0.033095,0.093855,0.12507,0.12367,0.09394,
       0.045791,-0.00791,-0.054401,-0.08378,-0.090908,-0.076127,-0.044716,
       -0.0053078,0.032318,0.059538,0.070757,0.064476,0.043355]
(%i12) plot2d([ [discrete,kL,ylastL], [discrete,kL,ylastL]],  [xlabel,"k"],[ylabel , "y(1)"],
           [style, [lines,2,1],[points,1,1,1]],[gnuplot_preamble,"set grid"],
                   [legend, false])$
```

which produces the plot

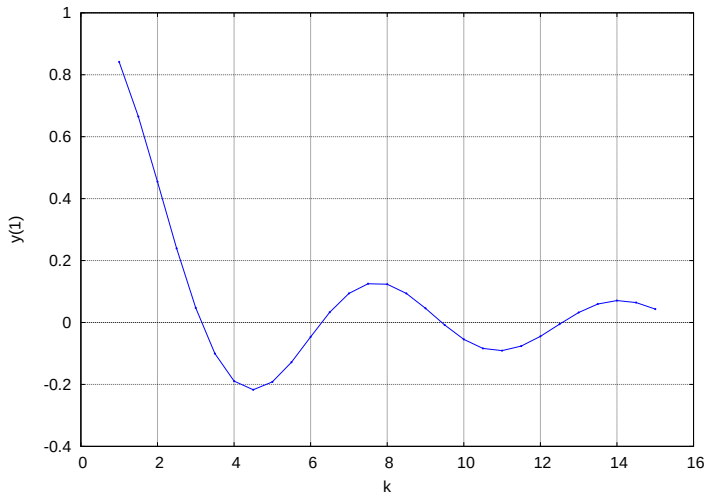


Figure 30: Numerical Values $y(1)$ vs. $k$ for $y'(0) = 1$

The numerical value $y(1)$ generated will depend on the value we use for $y'(0)$. If $y(x)$ is a solution of our linear homogeneous ode, then so is $c\,y(x)$ for any constant $c$. One would need to apply some normalization convention to fix the constant $c$, and this should not affect the discrete, isolated, values of $k$ for which the boundary condition $y(1) = 0$ is satisfied. This means that we have some freedom in the value of $y'(0)$ we use in our search for the eigenvalues $k_n$.

We can compare the effect of different choices of $y'(0)$ by plotting the values of $y(1)$ for both $y'(0) = 1$ and $y'(0) = 0.5$.

```
(%i13) ylastL2 : map (lambda ([k], ylast(k,0.5,0.01)), kL);
(%o13) [0.42074,0.3325,0.22732,0.11969,0.02352,-0.050112,-0.0946,-0.10861,
        -0.095892,-0.06414,-0.023285,0.016548,0.046928,0.062533,0.061835,
        0.04697,0.022896,-0.003955,-0.027201,-0.04189,-0.045454,-0.038063,
        -0.022358,-0.0026539,0.016159,0.029769,0.035378,0.032238,0.021678]
(%i14) plot2d( [  [discrete, kL, ylastL],   [discrete, kL, ylastL],
             [ discrete, kL, ylastL2], [discrete, kL, ylastL2]  ],
             [  style, [ lines,2,1], [ points,1,1,1],
             [ lines,2,2], [ points,1,2,1]  ],
                        [ xlabel,"k"], [ ylabel , "y(1)"], [ gnuplot_preamble, "set grid"],
             [ legend, "y'(0) = 1", "y'(0) = 1", "y'(0) = 0.5", "y'(0) = 0.5"  ] )$
```

which produces the plot

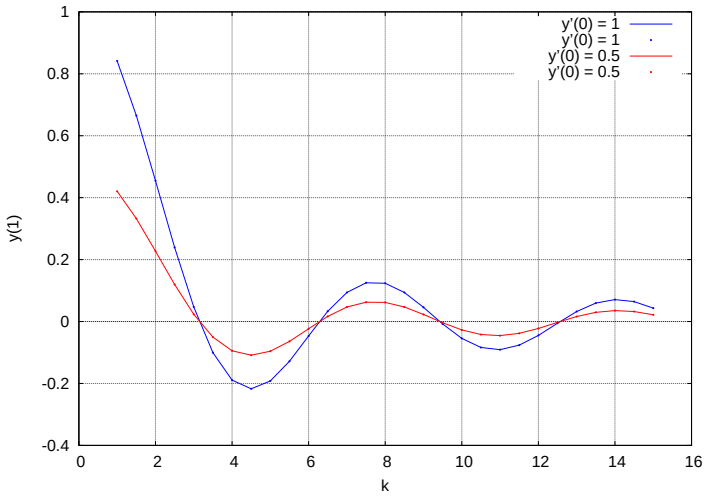


Figure 31: Numerical Values $y(1)$ vs. $k$ for Two Values of $y'(0)$

We see that the values of $k$ for which $y(1) = 0$ do not change if we vary $y'(0)$ somewhat.

For larger $k$, the integration done by **rk4** must cope with more oscillation in the interval $(0, 1)$, and the errors in $y(1)$ should increase. To test the sensitivity of the numerical value produced for $y(1)$ to the value of $k$ we define **wave1(k, yp0, h)**, which prints $k, y_{num}(1), y_{num}(1) - y_{an}(1)$.

```
(%i15) wave1(k,yp0,h) :=
block ( [yf, solnL, y1L, ylast, numer], numer:true,
    yf : yp0*sin(k)/k,     /* analytic y(1) */
    solnL : rk4( [y2, -k^2*y1], [y1, y2], [0, yp0], [x,0,1,h] ),
    y1L : take (solnL,2),
    ylast : last (y1L),      /* numerical y(1) */
    print ("  ", k, "  ", ylast, "  ", ylast - yf ) )$
(%i16) wave1(1, 1, 0.01)$
   1    0.84147    -4.560774E-11
(%i17) for k thru 15 do
           wave1(k, 1, 0.01)$
   1    0.84147    -4.560774E-11
   2    0.45465    5.345776E-10
   3    0.04704    6.6564902E-9
   4    -0.1892    1.4474494E-8
   5    -0.19178    -1.2680528E-8
   6    -0.046569    -1.0205692E-7
   7    0.093855    -1.582428E-7
   8    0.12367    2.7054689E-8
   9    0.045791    4.7983707E-7
   10    -0.054401    7.3446406E-7
   11    -0.090908    1.0630237E-7
   12    -0.044716    -1.3581221E-6
   13    0.032318    -2.2548933E-6
   14    0.070757    -8.0382239E-7
   15    0.043355    2.8371473E-6
```

As expected, the errors in $y_{num}(1)$ grow as $k$ increases.

Confining ourselves to values $k \leq 15$, we can make a simple table of the values of $y_{num}(1)$ as a function of $k$, looking for sign changes.

```
(%i18) for k thru 15 do print(" ",k," ", ylast(k, 1, 0.01))$
  1   0.84147
  2   0.45465
  3   0.04704
  4   -0.1892
```

```
  5   -0.19178
  6   -0.046569
  7   0.093855
  8   0.12367
  9   0.045791
  10   -0.054401
  11   -0.090908
  12   -0.044716
  13   0.032318
  14   0.070757
  15   0.043355
```

As also implied by our plots, we search for roots in the intervals $k \in [3,4], [6,7], [9,10], [12,13]$.

```
(%i19) k1 : find_root( lambda([k], ylast(k, 1, 0.01)), 3, 4);
(%o19) 3.141593
(%i20) k2 : find_root( lambda([k], ylast(k, 1, 0.01)), 6, 7);
(%o20) 6.283186
(%i21) k3 : find_root( lambda([k], ylast(k, 1, 0.01)), 9, 10);
(%o21) 9.424784
(%i22) k4 : find_root( lambda([k], ylast(k, 1, 0.01)), 12, 13);
(%o22) 12.5664
```

We can make a plot of the eigenfunction $y_1(x)$ and its derivative $y_1'(x)$ implied by the eigenvalue $k_1$. The numerical solution here assumes a normalization such that $y'(0) = 1$.

```
(%i23) solnL1 : rk4([y2,-k1^2*y1],[y1,y2],[0,1],[x,0,1,0.01])$
(%i24) yL1 : take(solnL1,2)$
(%i25) fll(yL1);
(%o25) [0.0,8.8470897E-17,101]
(%i26) ypL1 : take(solnL1,3)$
(%i27) fll(ypL1);
(%o27) [1.0,-1.0,101]
(%i28) xL1 : take(solnL1,1)$
(%i29) fll(xL1);
(%o29) [0.0,1.0,101]
```

Here we plot the numerical and exact values of $y(x)$ for $k = k_1$.

```
(%i30) plot2d ([[discrete, xL1,yL1], sin(k1*x)/k1],[x,0,1],
              [xlabel,"x"],[ylabel,"y"],[style,[lines,2]],
              [legend, "numerical","exact"],[gnuplot_preamble,"set grid"])$
```

which produces the plot

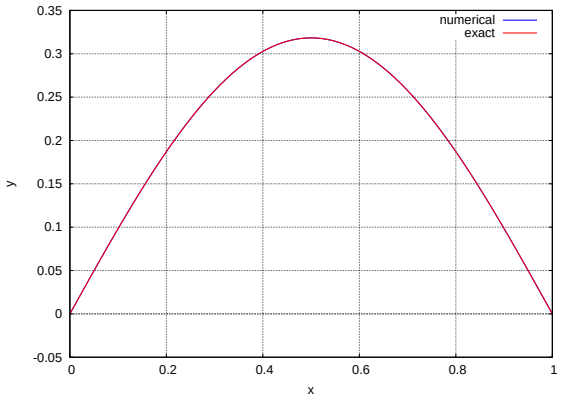


Figure 32: Numerical $y_1(x)$ for $k = k_1$ and $y'(0) = 1$ with Analytical Soln.

Next we plot the numerical and exact values of $y_1'(x)$ for $k = k_1$, again assuming $y'(0) = 1$.

```
(%i31) plot2d ([[discrete, xL1,ypL1], cos(k1*x)],[x,0,1],
              [xlabel,"x"],[ylabel,"dydx"],[style,[lines,2]],
              [legend, "numerical","exact"],[gnuplot_preamble,"set grid"])$
```

which produces the plot

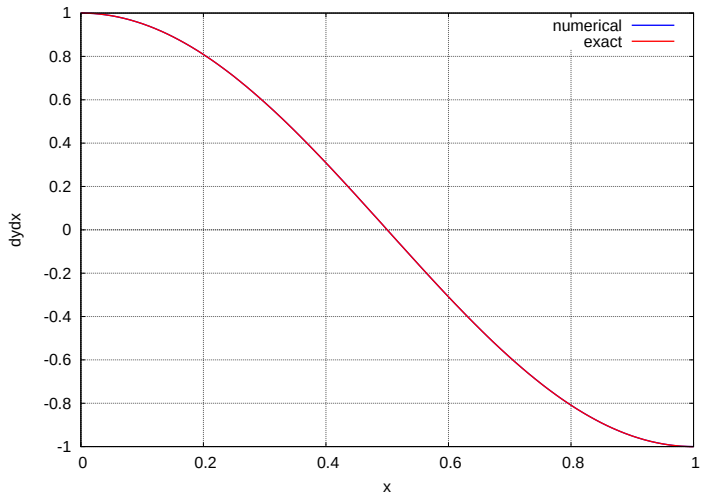


Figure 33: Numerical $y_1'(x)$ for $k = k_1$ and $y'(0) = 1$.

### 4.2.3 Eigenvalues using myrk4 and uniroot with R

We use **myrk4** and **uniroot** to find the roots $k_n$ for small $n$. We search for the value of $k$ such that both $y(0) = 0$ and $y(1) = 0$. The routine **ylast(k1)** returns the value of $y(1)$ produced for given values of **k1** and $y'(0)$ (represented by global parameter **del**). (We guess some reasonable value of $y'(0)$ – see below).

Recall that **myrk4(init, grid, func)** expects **func** to return a R vector of derivatives, and returns a list of the form **list(yL, ypL)**, in which **yL** and **ypL** are R vectors. Our global **derivs** function (below) depends on a global value of the parameter **k**. To get the **derivs** function to work with the function **ylast(k1)** below, we need to use the construct **k <<- k1** (inside **ylast**) to change the global setting of **k**. In the approach here, both **k** and **del** are global parameters.

```
> source("cp3.R")
> derivs = function(x,y) { c(y[2], -k^2* y[1] )}
> xL = seq(0,1,0.01)
> fll(xL)
  0   1   101
> del = 1
> k = 1
> out = myrk4(c(0,del),xL,derivs)
> yL = out[[1]]
> fll(yL)
  0   0.841471   101
> last(yL)
[1] 0.841471
> ylast = function(k1) {
+        k <<-  k1
+        last (myrk4(c(0,del),xL,derivs) [[1]] )}
> ylast(3)
[1] 0.04704001
> k
[1] 3
> ylast(1)
[1] 0.841471
> k
[1] 1
> kL = seq(1,15,0.5)
> ylastL = sapply(kL, ylast)
> fll(ylastL)
  0.841471   0.04335536   29
> plot(kL,ylastL,type="l",lwd = 2, col = "blue",xlab = "k",
+                         ylab = "y(1)")
> points(kL,ylastL,pch=19,col="blue")
> mygrid()
```

which produces the plot

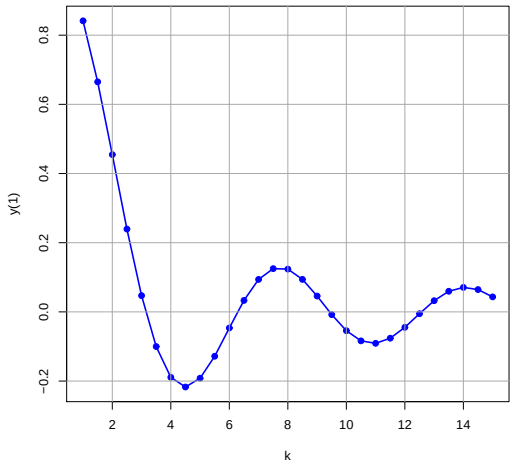


Figure 34: Numerical Values $y(1)$ vs. $k$ for $y'(0) = 1$

The numerical value $y(1)$ generated will depend on the value we use for $y'(0)$. If $y(x)$ is a solution of our linear homogeneous ode, then so is $c\,y(x)$ for any constant $c$. One would need to apply some normalization convention to fix the constant $c$, and this should not affect the discrete, isolated, values of $k$ for which the boundary condition $y(1) = 0$ is satisfied. This means that we have some freedom in the value of $y'(0)$ we use in our search for the eigenvalues $k_n$.

We can compare the effect of different choices of $y'(0)$ by plotting the values of $y(1)$ for both $y'(0) = 1$ and $y'(0) = 0.5$.

```
> del = 0.5
> ylastL2 = sapply(kL, ylast)
> fll(ylastL2)
  0.4207355   0.02167768   29
> lines(kL,ylastL2,col = "red",lwd = 2)
> points(kL,ylastL2,pch=19,col = "red")
> legend("topright", col = c("blue", "red"),lwd=2,
+              legend = c("del = 1", "del = 2"))
```

which produces the plot

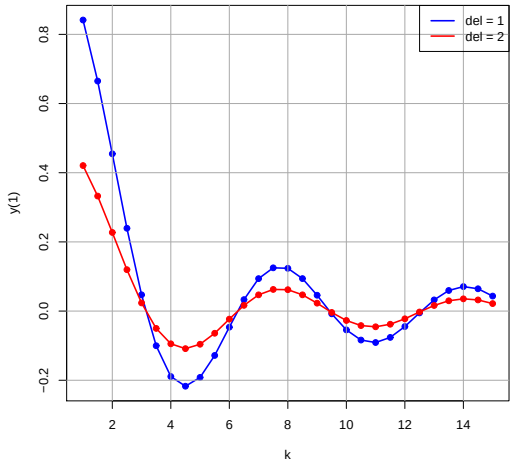


Figure 35: Numerical Values $y(1)$ vs. $k$ for Two Values of $y'(0)$

We see that the values of $k$ for which $y(1) = 0$ do not change if we vary $y'(0)$ somewhat.

For larger $k$, the integration done by **myrk4** must cope with more oscillation in the interval $(0, 1)$, and the errors in $y(1)$ should increase. To test the sensitivity of the numerical value produced for $y(1)$ to the value of $k$ we define **wave1(k1)**, which prints $k, y_{num}(1), y_{num}(1) - y_{an}(1)$.

```
> del = 1
> wave1 = function(k1) {
+      yan = del*sin(k1)/k1    #   analytic y(1)
+  ynum = ylast(k1)      #  numerical
+      cat("  ",k1,"  ",ynum,"  ",ynum - yan,"\n") }
> wave1(1)
   1    0.841471    -4.560774e-11
> for (k1 in seq(1,15,1) )   wave1(k1)
   1    0.841471    -4.560774e-11
   2    0.4546487    5.345776e-10
   3    0.04704001    6.65649e-09
   4    -0.1892006    1.447449e-08
   5    -0.1917849    -1.268053e-08
   6    -0.04656935    -1.020569e-07
   7    0.09385507    -1.582428e-07
   8    0.1236698    2.705469e-08
   9    0.04579142    4.798371e-07
   10    -0.05440138    7.344641e-07
   11    -0.09090809    1.063024e-07
   12    -0.04471577    -1.358122e-06
   13    0.03231829    -2.254893e-06
   14    0.07075686    -8.038224e-07
   15    0.04335536    2.837147e-06
```

As expected, the errors in $y_{num}(1)$ grow as $k$ increases.

Confining ourselves to values $k \leq 15$, we can make a simple table of the values of $y_{num}(1)$ as a function of $k$, looking for sign changes.

```
> for (k1 in seq(1,15,1)) cat(k1,"  ",ylast(k1),"\n")
1    0.841471
2    0.4546487
3    0.04704001
4    -0.1892006
5    -0.1917849
6    -0.04656935
7    0.09385507
8    0.1236698
9    0.04579142
10    -0.05440138
11    -0.09090809
12    -0.04471577
13    0.03231829
14    0.07075686
15    0.04335536
```

As also implied by our plots, we search for roots in the intervals $k \in [3, 4], [6, 7], [9, 10], [12, 13]$.

```
> k1 = uniroot(ylast,c(3,4))$root; k1
[1] 3.141588
> k2 = uniroot(ylast,c(6,7))$root; k2
[1] 6.283182
> k3 = uniroot(ylast,c(9,10))$root; k3
[1] 9.424783
> k4 = uniroot(ylast,c(12,13))$root; k4
[1] 12.5664
```

We can make a plot of the eigenfunction $y_1(x)$ and its derivative $y_1'(x)$ implied by the eigenvalue $k_1$. The numerical solution here assumes a normalization such that $y'(0) = 1$.

```
> del = 1
> k = k1
> k
[1] 3.141588
> solnL1 = myrk4(c(0,del),xL,derivs)
> yL1 = solnL1[[1]]
> fll(yL1)
  0   1.645898e-06   101
> ypL1 = solnL1[[2]]
> fll(ypL1)
  1   -1   101
> plot(xL,yL1,type="l",lwd=2,col="blue",xlab = "x",ylab = "y")
> curve(sin(k1*x)/k1,0,1,add=TRUE,col = "red",lwd=2)
> mygrid()
> legend("topright",col = c("blue","red"),lwd=2,
+        legend = c("numerical", "analytic"))
```

which produces the plot

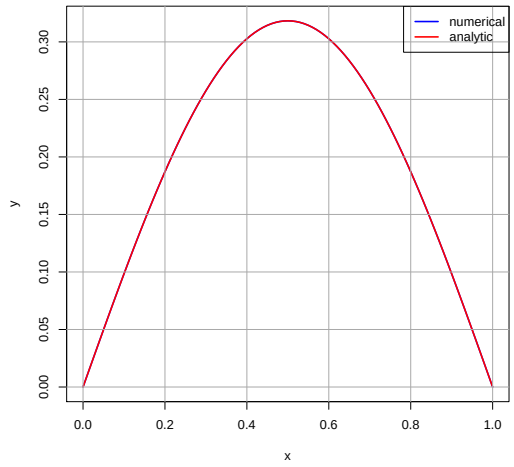


Figure 36: Numerical $y_1(x)$ for $k = k_1$ and $y'(0) = 1$ with Analytical Soln.

Next we plot the numerical and exact values of $y_1'(x)$ for $k = k_1$, again assuming $y'(0) = 1$.

```
> plot(xL,ypL1,type="l",lwd=2,col="blue",xlab = "x",ylab = "dydx")
> curve(cos(k1*x), 0, 1, add=TRUE, col = "red", lwd=2)
> mygrid()
> legend("topright",col = c("blue","red"),lwd=2,
+        legend = c("numerical", "analytic"))
```

which produces the plot

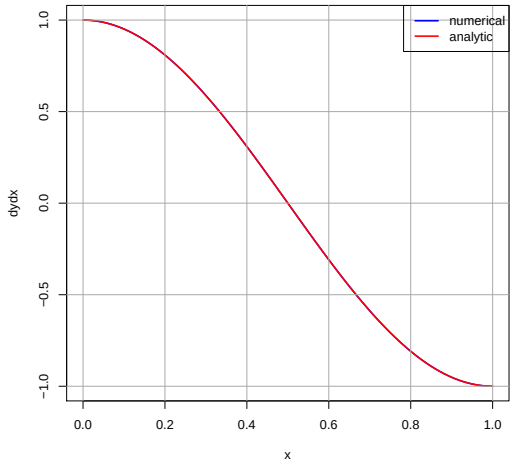


Figure 37: Numerical $y_1'(x)$ for $k = k_1$ and $y'(0) = 1$ with Analytical Soln.

## 4.3 Eigenvalues of the BVP: $y''(x) + k^2 y(x) = 0$ for $y'(0) = y(1) = 0$

We seek positive values of $k$ such that nontrivial solutions of

$$y''(x) + k^2 y(x) = 0, \quad y'(0) = 0, \quad y(1) = 0 \tag{4.12}$$

exist.

### 4.3.1 Analytic Solution Using Maxima

We can use Maxima's **ode2** and **ic2** functions.

```
(%i1) de : 'diff(y,x,2) + k^2*y;
(%o1) 'diff(y,x,2)+k^2*y
(%i2) assume(k>0);
(%o2) [k > 0]
(%i3) gsoln : ode2(de,y,x);
(%o3) y = %k1*sin(k*x)+%k2*cos(k*x)
(%i4) psoln : ic2(gsoln,x=0,y=y0,'diff(y,x)=0);
(%o4) y = cos(k*x)*y0
```

The analytic solution for any $y(0)$ and any positive $k$ is then

$$y_{an}(x) = y(0)\ \cos(k\,x), \tag{4.13}$$

and the analytic end value is

$$y_{an}(1) = y(0)\ \cos(k). \tag{4.14}$$

The condition that $y(1) = 0$ then implies the discrete normal modes

$$k_n = n\,\frac{\pi}{2}, \quad y_n(x) \sim \sin(k_n\,x), \quad n = 1, 3, 5, \ldots \tag{4.15}$$

For example, using Maxima's **solve** function:

```
(%i5) solve(cos(k),k);
solve: using arc-trig functions to get a solution.
Some solutions will be lost.
(%o5) [k = %pi/2]
```

The fundamental standing wave solutions of the one dimensional wave equation (4.1) for these spatial boundary conditions are then

$$u_n(x,t) = \cos(k_n\,x)\ \sin(k_n\,v\,t), \quad k_n = n\,\frac{\pi}{2}, \quad n = 1, 3, 5, \ldots \tag{4.16}$$

and

$$v_n(x,t) = \cos(k_n\,x)\ \cos(k_n\,v\,t), \quad k_n = n\,\frac{\pi}{2}, \quad n = 1, 3, 5, \ldots \tag{4.17}$$

The behavior at time $t = 0$, for example, can be used to determine a particular linear combination of the fundamental solutions which describes both the initial and subsequent behavior.

### 4.3.2 Eigenvalues using rk4 and find_root with Maxima

We use **rk4** and **find_root** to find the roots $k_n$ for small $n$ for the BVP (4.12). We search for the value of $k$ such that both $y'(0) = 0$ and $y(1) = 0$. The routine **ylast(k,yp0,h)** returns the value of $y(1)$ produced for given values of $k$, $y(0)$ (represented by **y0**), and step size $h$. (We guess some reasonable value of $y(0)$ – see below).

```
(%i6) load(cp3);
(%o6) "c:/k3/cp3.mac"
(%i7) ylast(k,y0,h) :=
block([solnL, numer],numer:true,
    solnL : rk4([y2,-k^2*y1],[y1,y2],[y0,0],[x,0,1,h]),
    last(take(solnL,2)))$
(%i8) ylast(1,1,0.01);
(%o8) 0.5403
(%i9) cos(1),numer;
(%o9) 0.5403
```

We can use **ylast** to make a plot of the numerical values of $y(1)$ as a function of $k$, assuming (here) the value $y(0) = 1$.

```
(%i10) kL : makelist(k,k,1,15,0.5);
(%o10) [1,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5,7.0,7.5,8.0,8.5,9.0,9.5,
       10.0,10.5,11.0,11.5,12.0,12.5,13.0,13.5,14.0,14.5,15.0]
(%i11) ylastL : map(lambda([k],ylast(k,1,0.01)),kL);
(%o11) [0.5403,0.070737,-0.41615,-0.80114,-0.98999,-0.93646,-0.65364,-0.2108,
       0.28366,0.70867,0.96017,0.97659,0.7539,0.34664,-0.1455,-0.60201,
       -0.91113,-0.99717,-0.83908,-0.47555,0.0044123,0.48329,0.84384,0.99779,
       0.90746,0.59495,0.13678,-0.35487,-0.75964]
(%i12) plot2d([ [discrete,kL,ylastL], [discrete,kL,ylastL]],  [xlabel,"k"],[ylabel , "y(1)"],
           [style, [lines,2,1],[points,1,1,1]],[gnuplot_preamble,"set grid"],
                   [legend, false])$
```

which produces the plot

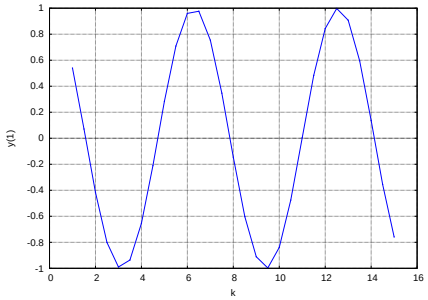


Figure 38: Numerical Values $y(1)$ vs. $k$ for $y(0) = 1$

The numerical value $y(1)$ generated will depend on the value we use for $y(0)$. If $y(x)$ is a solution of our linear homogeneous ode, then so is $c\,y(x)$ for any constant $c$. One would need to apply some normalization convention to fix the constant $c$, and this should not affect the discrete, isolated, values of $k$ for which the boundary condition $y(1) = 0$ is satisfied. This means that we have some freedom in the value of $y(0)$ we use in our search for the eigenvalues $k_n$.

We can compare the effect of different choices of $y(0)$ by plotting the values of $y(1)$ for both $y(0) = 1$ and $y(0) = 0.5$.

```
(%i13) ylastL2 : map(lambda([k],ylast(k,0.5,0.01)),kL);
(%o13) [0.27015,0.035369,-0.20807,-0.40057,-0.495,-0.46823,-0.32682,-0.1054,
       0.14183,0.35433,0.48009,0.48829,0.37695,0.17332,-0.072749,-0.301,
       -0.45556,-0.49859,-0.41954,-0.23777,0.0022062,0.24164,0.42192,0.4989,
       0.45373,0.29747,0.06839,-0.17744,-0.37982]
(%i14) plot2d( [  [discrete, kL, ylastL],   [discrete, kL, ylastL],
             [ discrete, kL, ylastL2], [discrete, kL, ylastL2]  ],
             [  style, [ lines,2,1], [ points,1,1,1],
                                     [ lines,2,2], [ points,1,2,1]  ],
                        [ xlabel,"k"], [ ylabel , "y(1)"], [ gnuplot_preamble, "set grid"],
             [ legend, "y(0) = 1", "y(0) = 1", "y(0) = 0.5", "y(0) = 0.5"  ] )$
```

which produces the plot

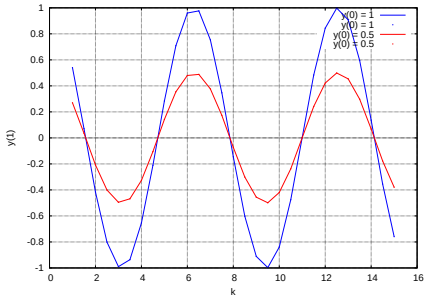


Figure 39: Numerical Values $y(1)$ vs. $k$ for Two Values of $y(0)$

We see that the values of $k$ for which $y(1) = 0$ do not change if we vary $y(0)$ somewhat.

For larger $k$, the integration done by **rk4** must cope with more oscillation in the interval $(0, 1)$, and the errors in $y(1)$ should increase. To test the sensitivity of the numerical value produced for $y(1)$ to the value of $k$ we define **wave1(k, y0, h)**, which prints $k, y_{num}(1), y_{num}(1) - y_{an}(1)$.

```
(%i15) wave1(k,y0,h) :=
block ( [yexact, solnL, y1L, ynum, numer], numer:true,
     yexact : y0*cos(k),     /* analytic y(1) */
     solnL : rk4( [y2, -k^2*y1], [y1, y2], [y0, 0], [x,0,1,h] ),
         y1L : take (solnL,2),
         ynum : last (y1L),      /* numerical y(1) */
         print ("  ", k, "  ", ynum, "  ", ynum - yexact ) )$
(%i16) wave1(1, 1, 0.01)$
   1    0.5403    6.9745654E-11
(%i17) for k thru 15 do
           wave1(k, 1, 0.01)$
   1    0.5403    6.9745654E-11
   2    -0.41615    2.4429408E-9
   3    -0.98999    3.357889E-9
   4    -0.65364    -6.2684697E-8
   5    0.28366    -2.5257391E-7
   6    0.96017    -2.1192422E-7
   7    0.7539    8.5699728E-7
   8    -0.1455    2.7219014E-6
   9    -0.91113    2.3579966E-6
   10    -0.83908    -3.9353366E-6
   11    0.0044123    -1.3368254E-5
   12    0.84384    -1.2816029E-5
   13    0.90746    9.8862842E-6
   14    0.13678    4.3373724E-5
   15    -0.75964    4.6814335E-5
```

As expected, the errors in $y_{num}(1)$ grow as $k$ increases.

Confining ourselves to values $k \leq 15$, we can make a simple table of the values of $y_{num}(1)$ as a function of $k$, looking for sign changes.

```
(%i18) ylast(k,y0,h) :=
block([solnL, numer],numer:true,
     solnL : rk4([y2,-k^2*y1],[y1,y2],[y0,0],[x,0,1,h]),
         last(take(solnL,2)))$
(%i19) for k thru 15 do print(" ",k," ",ylast(k, 1, 0.01))$
  1   0.5403
  2   -0.41615
  3   -0.98999
  4   -0.65364
  5   0.28366
  6   0.96017
  7   0.7539
  8   -0.1455
  9   -0.91113
  10   -0.83908
  11   0.0044123
  12   0.84384
  13   0.90746
  14   0.13678
  15   -0.75964
```

As also implied by our plots, we search for roots in the intervals $k \in [1, 2], [4, 5], [7, 8], [10, 11], [14, 15]$.

```
(%i20) k1 : find_root( lambda([k], ylast(k, 1, 0.01)), 1, 2);
(%o20) 1.570796
(%i21) k2 : find_root( lambda([k], ylast(k, 1, 0.01)), 4, 5);
(%o21) 4.712389
(%i22) k3 : find_root( lambda([k], ylast(k, 1, 0.01)), 7, 8);
(%o22) 7.853984
(%i23) k4 : find_root( lambda([k], ylast(k, 1, 0.01)), 10, 11);
(%o23) 10.99559
```

```
(%i24) k5 : find_root( lambda([k], ylast(k, 1, 0.01)), 14, 15);
(%o24) 14.13721
```

We can make a plot of the eigenfunction $y_1(x)$ and its derivative $y_1'(x)$ implied by the eigenvalue $k_1$. The numerical solution here assumes a normalization such that $y(0) = 1$.

```
(%i25) solnL1 : rk4([y2,-k1^2*y1],[y1,y2],[1,0],[x,0,1,0.01])$
(%i26) yL1 : take(solnL1,2)$
(%i27) fll(yL1);
(%o27) [1.0,-3.1225023E-17,101]
(%i28) ypL1 : take(solnL1,3)$
(%i29) fll(ypL1);
(%o29) [0.0,-1.570796,101]
(%i30) xL1 : take(solnL1,1)$
(%i31) fll(xL1);
(%o31) [0.0,1.0,101]
```

Here we plot the numerical and exact values of $y_1(x)$ for $k = k_1$.

```
(%i32) plot2d ([[discrete, xL1,yL1], cos(k1*x)],[x,0,1],
               [xlabel,"x"],[ylabel,"y"],[style,[lines,2]],
               [legend, "numerical","exact"],[gnuplot_preamble,"set grid"])$
```

which produces the plot

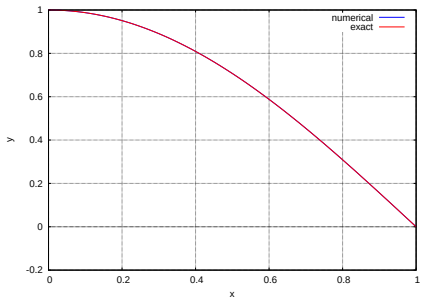


Figure 40: Numerical $y_1(x)$ for $k = k_1$ and $y(0) = 1$ with Analytic Soln.

Next we plot the numerical and exact values of $y_1'(x)$ for $k = k_1$, again assuming $y(0) = 1$.

```
(%i33) plot2d ([[discrete, xL1,ypL1], -k1*sin(k1*x)],[x,0,1],
               [xlabel,"x"],[ylabel,"dydx"],[style,[lines,2]],
               [legend, "numerical","exact"],[gnuplot_preamble,"set grid"])$
```

which produces the plot

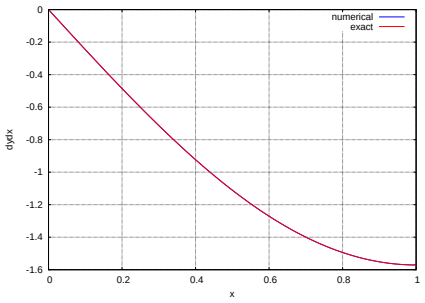


Figure 41: Numerical $y_1'(x)$ for $k = k_1$ and $y(0) = 1$ with Analytic Soln.

### 4.3.3 Eigenvalues using myrk4 and uniroot with R

We use **myrk4** and **uniroot** to find the roots $k_n$ for small $n$. We search for the values of $k$ such that $y'(0) = 0$ and $y(1) = 0$. The routine **ylast(k1)** returns the value of $y(1)$ produced for given values of **k1** and $y(0)$ (represented by global parameter **del**). (We guess some reasonable value of $y(0)$ – see below).

Recall that **myrk4(init, grid, func)** expects **func** to return a R vector of derivatives, and returns a list of the form **list(yL, ypL)**, in which **yL** and **ypL** are R vectors. Our global **derivs** function (below) depends on a global value of the parameter **k**. To get the **derivs** function to work with the function **ylast(k1)** below, we need to use the syntax **k <<- k1** (inside **ylast**) to change the global setting of **k**. In the approach here, both **k** and **del** are global parameters.

```
> source("cp3.R")
> derivs = function(x,y) { c(y[2], -k^2* y[1] )}
> xL = seq(0,1,0.01)
> fll(xL)
  0   1   101
> del = 1
> k = 1
> out = myrk4(c(del,0),xL,derivs)
> yL = out[[1]]
> fll(yL)
  1   0.5403023   101
> last(yL)
[1] 0.5403023
> ylast = function(k1) {
+             k <<-  k1
+             last (myrk4(c(del,0), xL, derivs) [[1]] )}
> ylast(3)
[1] -0.9899925
> k
[1] 3
> ylast(1)
[1] 0.5403023
> k
[1] 1
> kL = seq(1,15,0.5)
> ylastL = sapply(kL, ylast)
> fll(ylastL)
  0.5403023   -0.7596411   29
> plot(kL,ylastL,type="l",lwd = 2, col = "blue",xlab = "k",
+                         ylab = "y(1)")
> points(kL,ylastL,pch=19,col="blue")
> mygrid()
```

which produces the plot

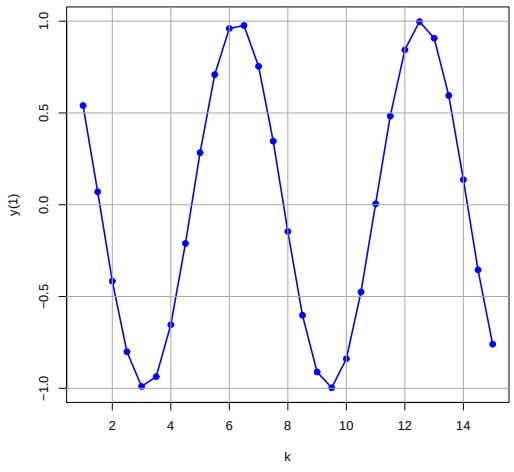


Figure 42: Numerical Values $y(1)$ vs. $k$ for $y(0) = 1$

The numerical value $y(1)$ generated will depend on the value we use for $y(0)$. If $y(x)$ is a solution of our linear homogeneous ode, then so is $c\,y(x)$ for any constant $c$. One would need to apply some normalization convention to fix the constant $c$, and this should not affect the discrete, isolated, values of $k$ for which the boundary condition $y(1) = 0$ is satisfied. This means that we have some freedom in the value of $y(0)$ we use in our search for the eigenvalues $k_n$.

We can compare the effect of different choices of $y(0)$ by adding to this plot the values of $y(1)$ for $y(0) = 0.5$.

```
> del = 0.5
> ylastL2 = sapply(kL, ylast)
> fll(ylastL2)
  0.2701512   -0.3798205   29
> lines(kL, ylastL2, lwd = 2, col = "red")
> points(kL, ylastL2, pch=19, col="red")
> legend("topleft",col = c("blue","red"),lwd=2,
+       legend = c("y(0) = 1", "y(0) = 0.5"))
```

which produces the plot

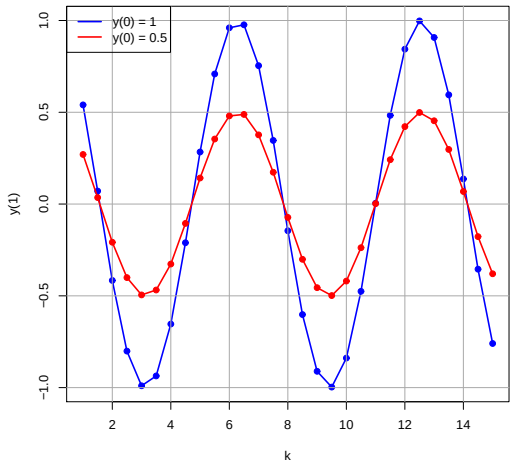


Figure 43: Numerical Values $y(1)$ vs. $k$ for Two Values of $y(0)$

We see that the values of $k$ for which $y(1) = 0$ do not change if we vary $y(0)$ somewhat.

For larger $k$, the integration done by **myrk4** must cope with more oscillation in the interval $(0, 1)$, and the errors in $y(1)$ should increase. To test the sensitivity of the numerical value produced for $y(1)$ to the value of $k$ we define **wave1(k1)**, which prints $k, y_{num}(1), y_{num}(1) - y_{an}(1)$.

```
> del = 1
> wave1 = function(k1) {
+         yan = del*cos(k1)    #   analytic y(1)
+         ynum = ylast(k1)      #  numerical
+         cat("  ",k1,"  ",ynum,"  ",ynum - yan,"\n") }
> wave1(1)
   1    0.5403023    6.974565e-11
> for (k1 in seq(1,15,1) )   wave1(k1)
   1    0.5403023    6.974565e-11
   2    -0.4161468    2.442941e-09
   3    -0.9899925    3.357889e-09
   4    -0.6536437    -6.26847e-08
   5    0.2836619    -2.525739e-07
   6    0.9601701    -2.119242e-07
   7    0.7539031    8.569973e-07
   8    -0.1454973    2.721901e-06
   9    -0.9111279    2.357997e-06
   10    -0.8390755    -3.935337e-06
   11    0.00441233    -1.336825e-05
   12    0.8438411    -1.281603e-05
   13    0.9074567    9.886284e-06
   14    0.1367806    4.337372e-05
   15    -0.7596411    4.681434e-05
```

As expected, the errors in $y_{num}(1)$ grow as $k$ increases.

Confining ourselves to values $k \leq 15$, we can make a simple table of the values of $y_{num}(1)$ as a function of $k$, looking for sign changes.

```
> for (k1 in seq(1,15,1)) cat(k1,"  ",ylast(k1),"\n")
1    0.5403023
2    -0.4161468
3    -0.9899925
4    -0.6536437
5    0.2836619
6    0.9601701
7    0.7539031
8    -0.1454973
9    -0.9111279
10    -0.8390755
11    0.00441233
12    0.8438411
13    0.9074567
14    0.1367806
15    -0.7596411
```

As also implied by our plots, we search for roots in the intervals $k \in [1,2], [4,5], [7,8], [10,11], [14,15]$.

```
> k1 = uniroot(ylast,c(1,2))$root; k1
[1] 1.570796
> k2 = uniroot(ylast,c(4,5))$root; k2
[1] 4.712389
> k3 = uniroot(ylast,c(7,8))$root; k3
[1] 7.853973
> k4 = uniroot(ylast,c(10,11))$root; k4
[1] 10.99559
> k5 = uniroot(ylast,c(14,15))$root; k5
[1] 14.13723
```

We can make a plot of the eigenfunction $y_1(x)$ and its derivative $y_1'(x)$ implied by the eigenvalue $k_1$. The numerical solution here assumes a normalization such that $y(0) = 1$.

```
> del = 1
> k = k1
> k
[1] 1.570796
> solnL1 = myrk4(c(del,0),xL,derivs)
> yL1 = solnL1[[1]]
> fll(yL1)
  1   1.021841e-09   101
> ypL1 = solnL1[[2]]
> fll(ypL1)
  0   -1.570796   101
> plot(xL,yL1,type="l",lwd=2,col="blue",xlab = "x",ylab = "y")
> curve(cos(k1*x),0,1,add=TRUE,col = "red",lwd=2)
> mygrid()
> legend("topright",col = c("blue","red"),lwd=2,
+        legend = c("numerical", "analytic"))
```

which produces the plot

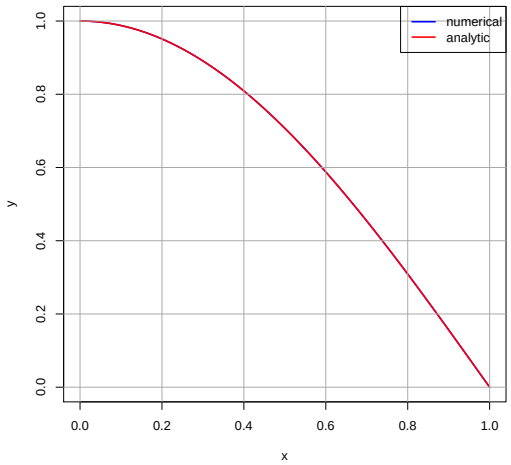


Figure 44: Numerical $y_1(x)$ for $k = k_1$ and $y(0) = 1$ with Analytical Soln.

Next we plot the numerical and exact values of $y_1'(x)$ for $k = k_1$, again assuming $y(0) = 1$.

```
> plot(xL,ypL1,type="l",lwd=2,col="blue",xlab = "x",ylab = "dydx")
> curve(-k1*sin(k1*x), 0, 1, add=TRUE, col = "red", lwd=2)
> mygrid()
> legend("topright",col = c("blue","red"),lwd=2,
+        legend = c("numerical", "analytic"))
```

which produces the plot

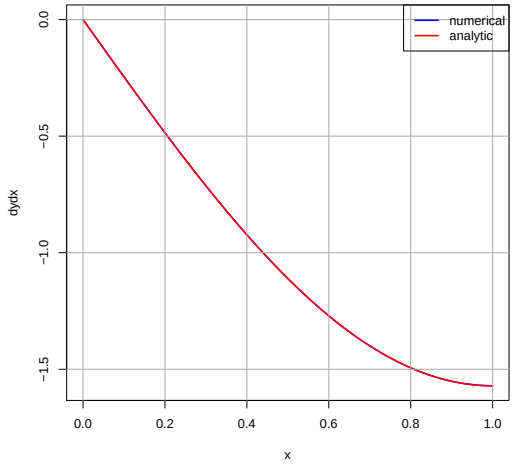


Figure 45: Numerical $y_1'(x)$ for $k = k_1$ and $y(0) = 1$ with Analytical Soln.

# 5 Wave Equation $u_{tt} = v^2 \nabla^2 u$ in Plane Polar Coordinates

Using plane polar coordinates $r$ and $\theta$ to treat the vibrations of a circular drumhead with given boundary and initial conditions, with $u$ the height of the drumhead from the plane, we scale the radial distance to $0 \le r \le 1$ and assume $u = 0$ when $r = 1$. We seek $u(r, \theta, t)$ that satisfies

$$u_{tt} = v^2 \nabla^2 u = v^2 \left(u_{rr} + \frac{1}{r} u_r + \frac{1}{r^2} u_{\theta\theta}\right), \quad 0 \le r \le 1 \tag{5.1}$$

such that

$$u = 0 \quad \text{when} \quad r = 1 \quad \text{for} \quad 0 \le t < \infty, \tag{5.2}$$

subject to initial conditions on $u$ and $u_t$ when $t = 0$.

In this section we generally use a notation which agrees with that of Stanley J. Farlow, Partial Differential Equations for Scientists and Engineers, John Wiley, 1982, Lesson 30, “The Vibrating Drumhead; (also in the Dover reprint, 1993).

## 5.1 Separation of Variables

We look for fundamental standing wave solutions of the form

$$u(r, \theta, t) = R(r)\, \Theta(\theta)\, T(t) \tag{5.3}$$

by using the method of separation of variables. We first separate out the dependence on time $t$, using

$$u(r, \theta, t) = U(r, \theta)\, T(t). \tag{5.4}$$

Substitution into (5.1) leads to

$$\frac{1}{v^2} \frac{T''(t)}{T(t)} = \frac{\nabla^2 U}{U} = -k^2. \tag{5.5}$$

The sign of the separation constant is chosen to arrive at simple harmonic motion in time

$$T''(t) + k^2 v^2 T(t) = 0, \tag{5.6}$$

and the spatial dependence is governed by the Helmholtz equation

$$\nabla^2 U(r,\theta) + k^2\, U(r,\theta) = 0. \tag{5.7}$$

We need $U(r=1,\theta)=0$ to satisfy the spatial boundary condition at $r=1$. We next substitute $U(r,\theta) = R(r)\,\Theta(\theta)$ in (5.7), divide by $R\,\Theta$, and multiply by $r^2$ to get

$$\frac{r^2}{R}\,R''(r) + \frac{r}{R}\,R'(r) + r^2\,k^2 = -\frac{1}{\Theta}\,\Theta''(\theta) = n^2 \tag{5.8}$$

The $\theta$ dependence, governed by

$$\Theta''(\theta) + n^2\,\Theta(\theta) = 0, \tag{5.9}$$

is then of the form

$$A\,\sin(n\,\theta) + B\,\cos(n\,\theta), \tag{5.10}$$

for $n = 0, 1, 2, 3, \ldots$ and for $n=0$ becomes a constant independent of $\theta$. The radial dependence is then governed by

$$R''(r) + \frac{1}{r}\,R'(r) + \left(k^2 - \frac{n^2}{r^2}\right)\,R(r) = 0 \tag{5.11}$$

with the boundary condition

$$R(1) = 0. \tag{5.12}$$

## 5.2 The $n=0$ and $R(0)=1$ Case, with $R(r) \to y(x)$

With the replacement $R(r) \to y(x)$ in the radial equation, and assuming $R(0)=1$, we seek values of $k$ such that the BVP

$$y''(x) + \frac{1}{x}\,y'(x) + k^2\,y(x) = 0, \quad y(0)=1, \quad y(1)=0 \tag{5.13}$$

is satisfied. We can sidestep the problem of the singularity $\frac{1}{x}$ at $x=0$ in the second term by starting the integration at $x_0 = 10^{-15}$.

### 5.2.1 Analytic Solution in Terms of Bessel Functions Using Maxima

The general solution of the linear ode

$$u''(z) + \frac{1}{z}\,u'(z) + u(z) = 0 \tag{5.14}$$

is a linear combination of the Bessel functions $J_0(z)$ and $Y_0(z)$. For example, using the identity $J_2 = 2\,J_1/z - J_0$,

```
(%i1) u : bessel_j(0,z);
(%o1) bessel_j(0,z)
(%i2) up : diff(u,z);
(%o2) -bessel_j(1,z)
(%i3) upp : diff(up,z);
(%o3) -(bessel_j(0,z)-bessel_j(2,z))/2
(%i4) eqn : z*upp + up + z*u,bessel_j(2,z)=2*bessel_j(1,z)/z - bessel_j(0,z),expand;
(%o4) 0
```

and using the identity $Y_2 = 2\,Y_1/z - Y_0$,

```
(%i5) u : bessel_y(0,z);
(%o5) bessel_y(0,z)
(%i6) up : diff(u,z);
(%o6) -bessel_y(1,z)
(%i7) upp : diff(up,z);
(%o7) -(bessel_y(0,z)-bessel_y(2,z))/2
(%i8) eqn : z*upp + up + z*u,bessel_y(2,z)=2*bessel_y(1,z)/z - bessel_y(0,z),expand;
(%o8) 0
```

If we let $z = k\,x$ (for positive $k$) and $y(x) = u(z) = u(k\,x)$, then (5.14) takes the form of (5.13), so that the solution of (5.13) (without the boundary conditions) is $y(x) = A\,J_0(k\,x) + B\,Y_0(k\,x)$, in which $A$ and $B$ are constants chosen to fit the boundary conditions. Since $Y_0(k\,x) \to -\infty$ as $x \to 0$, $B=0$ and $y(x) = A\,J_0(k\,x)$.

We have the property $J_0(0) = 1$,

```
(%i9) bessel_j(0,0);
(%o9) 1
```

so if we require $y(0) = 1$ (for our BVP), then $y(x) = J_0(k\,x)$ The further requirement that $y(1) = 0$ then implies that $J_0(k) = 0$, which is only satisfied for a discrete set of positive values of $k$, which can be found using Maxima. We first make a plot of $J_0(k)$ versus $k$.

```
(%i10) plot2d(bessel_j(0,k),[k,0,20],[xlabel,"k"],[ylabel,"J0(k)"],
          [style,[lines,2]],[gnuplot_preamble,"set grid"])$
```

which produces the plot

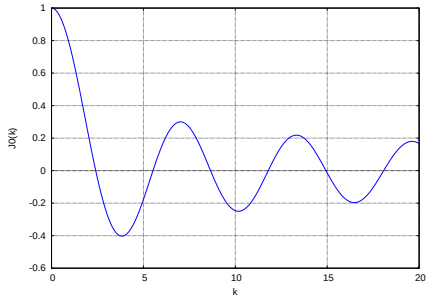


Figure 46: Values of $J_0(k)$ versus $k$

We can put the cursor at the intersections of the curve with the $y = 0$ axis to find approximate values of the set of eigenvalues $k_j$.

We can also make a simple table of $J_0(k)$ as a function of $k$, looking for sign changes.

```
(%i11) fpprintprec:8$
(%i12) for k thru 15 do
          print(" ",k," ",float(bessel_j(0,k)))$
  1   0.765198
  2   0.223891
  3   -0.260052
  4   -0.39715
  5   -0.177597
  6   0.150645
  7   0.300079
  8   0.171651
  9   -0.0903336
  10   -0.245936
  11   -0.17119
  12   0.0476893
  13   0.206926
  14   0.171073
  15   -0.0142245
```

We can then use **`find_root`** with **`bessel_j(0,k)`** to find the first four roots of $J_0(k) = 0$, using the intervals $k \in (2,3), (5,6), (8,9), (11,12)$. The "0" stands for $n = 0$, the "a" stands for "analytic".

```
(%i13) k01a : find_root(bessel_j(0,k),k,2,3);
(%o13) 2.404826
(%i14) k02a : find_root(bessel_j(0,k),k,5,6);
(%o14) 5.520078
(%i15) k03a : find_root(bessel_j(0,k),k,8,9);
(%o15) 8.653728
(%i16) k04a : find_root(bessel_j(0,k),k,11,12);
(%o16) 11.79153
```

### 5.2.2 Why We Need to Use $y'(0) = 0$

Multiply (5.13) through by $x$, to get

$$x\,y''(x) + y'(x) + x\,k^2\,y(x) = 0. \tag{5.15}$$

Take the limit of this ode as $x \to 0$, and assume both $y(0)$ and $y''(0)$ are finite. The first and third terms approach zero, leaving $y'(0) = 0$.

### 5.2.3 Numerical Solution using rk4 and find_root with Maxima

We define **ylast(k)** to return the value of $y(1)$ for a given value of $k$. (Note the value of **x0** which avoids the singularity at $x = 0$.)

```
(%i1) load(cp3);
(%o1) "c:/k3/cp3.mac"
(%i2) ylast(k) :=
block([solnL,x0:1e-15,h:0.01, numer],numer:true,
     solnL : rk4([y2, -k^2*y1 - y2/x],[y1,y2],[1,0],[x,x0,1,h]),
     last(take(solnL,2)))$
(%i3) ylast(2);
(%o3) 0.22388
(%i4) ylast(3);
(%o4) -0.26003
```

We can use **ylast** to make a plot of the numerical values of $y(1)$ as a function of $k$.

```
(%i5) kL : makelist (k,k,1,15,0.5)$
(%i6) fll(kL);
(%o6) [1,15.0,29]
(%i7) ylastL : map (ylast,kL)$
(%i8) fll(ylastL);
(%o8) [0.76519,-0.014197,29]
(%i9) plot2d([ [discrete,kL,ylastL], [discrete,kL,ylastL]],  [xlabel,"k"],[ylabel , "y(1)"],
           [style, [lines,2,1],[points,1,1,1]],[gnuplot_preamble,"set grid"],
                   [legend, false])$
```

which produces the plot

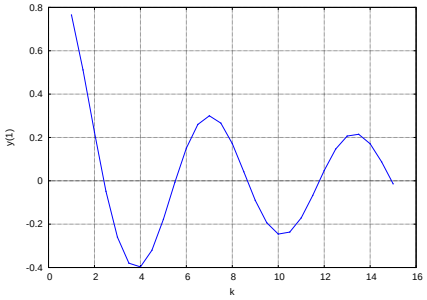


Figure 47: Values of $y(1)$ versus $k$

and make a table, looking for sign changes

```
(%i10) for j thru length(kL) do print(" ",kL[j]," ",ylastL[j])$
  1   0.76519
  1.5   0.51182
  2.0   0.22388
  2.5   -0.048381
  3.0   -0.26003
  3.5   -0.38009
  4.0   -0.3971
  4.5   -0.32049
  5.0   -0.17756
  5.5   -0.0068419
  6.0   0.1506
  6.5   0.26
  7.0   0.29996
  7.5   0.26621
  8.0   0.17156
  8.5   0.041913
  9.0   -0.090273
  9.5   -0.19378
  10.0   -0.24573
  10.5   -0.23643
  11.0   -0.17101
  11.5   -0.067577
  12.0   0.047632
  12.5   0.14669
  13.0   0.20663
```

```
  13.5   0.21466
  14.0   0.17079
  14.5   0.087388
  15.0   -0.014197
```

We can then look for the eigenvalues $k_{0\,j}$ using **ylast** with **find_root** with $k \in (2, 2.5), (5.5, 6), (8.5, 9), (11.5, 12)$, and compare the numerical values with the "analytic" values.

```
(%i11) k01 : find_root(ylast,2,2.5);
(%o11) 2.404826
(%i12) k01 - k01a;
(%o12) -3.2908436E-8
(%i13) k02 : find_root(ylast,5.5,6);
(%o13) 5.520077
(%i14) k02 - k02a;
(%o14) -7.2644058E-7
(%i15) k03 : find_root(ylast,8.5,9);
(%o15) 8.653725
(%i16) k03 - k03a;
(%o16) -2.9313481E-6
(%i17) k04 : find_root(ylast,11.5,12);
(%o17) 11.79153
(%i18) k04 - k04a;
(%o18) -4.8401604E-6
```

We can compare the numerical solution for $y(x)$ with the analytic solution for the lowest $k = k_1$ eigenvalue using a plot.

```
(%i19) soln(k) :=
block([x0:1e-15,h:0.01, numer],numer:true,
     solnL : rk4([y2, -k^2*y1 - y2/x],[y1,y2],[1,0],[x,x0,1,h]))$
(%i20) soln1 : soln(k01)$
(%i21) xL : take(soln1,1)$
(%i22) fll(xL);
(%o22) [1.0E-15,1.0,101]
(%i23) yL1 : take(soln1,2)$
(%i24) fll(yL1);
(%o24) [1.0,-5.8980598E-17,101]
(%i25) plot2d([[discrete, xL,yL1], bessel_j(0, k01a * x) ], [x,0,1],[xlabel,"x"],
           [ylabel,"y"], [style,[lines,2]], [legend,"numerical","analytic"],
                   [gnuplot_preamble,"set grid"])$
```

which produces the plot showing agreement:

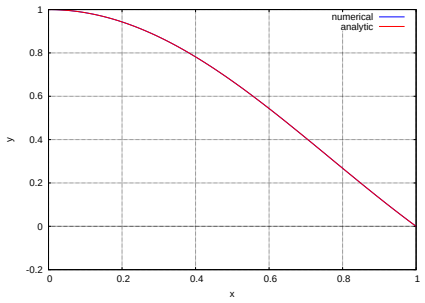


Figure 48: $k_1$ Numerical Solution compared with $J_0(k_1\, x)$

Likewise for $k = k_2$:

```
(%i26) soln2 : soln(k02)$
(%i27) yL2 : take(soln2,2)$
(%i28) fll(yL2);
(%o28) [1.0,-1.0755286E-16,101]
(%i29) plot2d([[discrete, xL,yL2], bessel_j(0, k02a * x) ], [x,0,1],[xlabel,"x"],
           [ylabel,"y"], [style,[lines,2]], [legend,"numerical","analytic"],
                   [gnuplot_preamble,"set grid"])$
```

which again shows agreement:

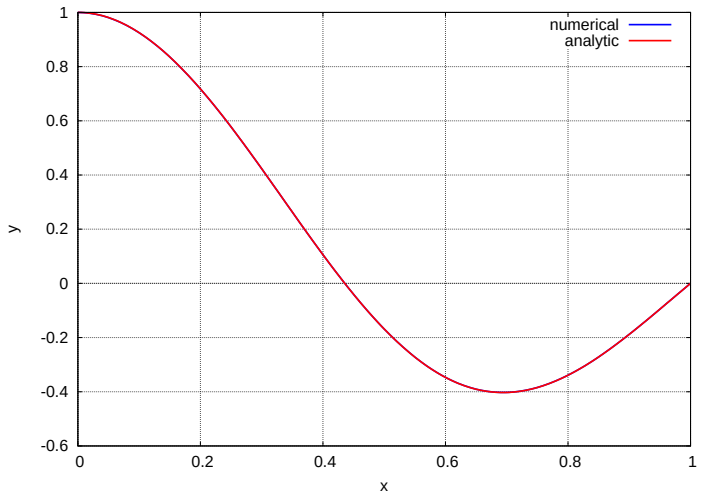


Figure 49: $k_2$ Numerical Solution compared with $J_0(k_2\,x)$

### 5.2.4 Numerical Solution using myrk4 and uniroot with R

We seek values of $k$ such that the BVP

$$y''(x) + \frac{1}{x}\,y'(x) + k^2\,y(x) = 0, \quad y(0) = 1, \quad y(1) = 0 \tag{5.16}$$

is satisfied. We can sidestep the problem of the singularity $\frac{1}{x}$ at $x = 0$ in the second term by starting the integration at $x_0 = 10^{-15}$.

As we have discussed above, the analytic solutions are $y_k(x) = J_0(k\,x)$, in which the discrete set of eigenvalues $k$ are the solutions of $y_k(1) = J_0(k) = 0$.

If you type **`?Bessel`** inside R, you will find a description of the syntax for Bessel functions in R. In particular, $J_n(x)$ is represented by the R function **`besselJ(x, n)`**.

Here we use R to numerically evaluate $J_0(0) = 1$, and then make a plot of $J_0(k)$ as a function of $k$, to see visually where the curve crosses the $y = 0$ axis; these locate the eigenvalues for which $y(1) = J_0(k) = 0$.

```
> besselJ(0,0)
[1] 1
> source("cp3.R")
> curve(besselJ(k,0),0,15,xname="k",col="blue",lwd=3,
+       ylab="J0(k)")
> mygrid()
```

which produces the plot

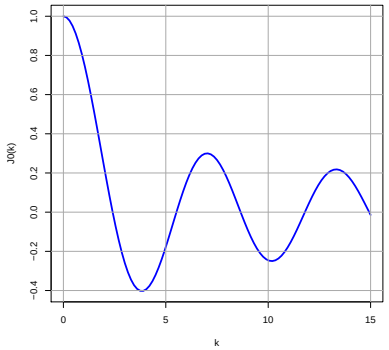


Figure 50: $J_0(k)$ versus $k$ using besselJ(k,0)

We can make a simple table of values of $k$ and the corresponding values of $J_0(k)$, looking for sign changes.

```
> for (k in seq(1,15,1)) cat (k," ",besselJ(k,0), "\n")
1   0.7651977
2   0.2238908
3   -0.260052
4   -0.3971498
5   -0.1775968
6   0.1506453
7   0.3000793
8   0.1716508
9   -0.09033361
10   -0.2459358
11   -0.1711903
12   0.04768931
13   0.2069261
14   0.1710735
15   -0.01422447
```

We can then use **uniroot** to find eigenvalues for $k \in (2,3),(5,6),(8,9),(11,12),(14,15)$.

```
> k01a = uniroot(function(k) besselJ(k,0), c(2,3), tol = 1e-10)$root
> k01a
[1] 2.404826
> k02a = uniroot(function(k) besselJ(k,0), c(5,6), tol = 1e-10)$root
> k02a
[1] 5.520078
> k03a = uniroot(function(k) besselJ(k,0), c(8,9), tol = 1e-10)$root
> k03a
[1] 8.653728
> k04a = uniroot(function(k) besselJ(k,0), c(11,12), tol = 1e-10)$root
> k04a
[1] 11.79153
> k05a = uniroot(function(k) besselJ(k,0), c(14,15), tol = 1e-10)$root
> k05a
[1] 14.93092
```

We next define a R function **ylast(k1)** which uses our homemade R function **myrk4** (as used above) to return the value of $y(1)$ for given $k = k_1$ value. In our global **derivs** function, **k** is a global parameter which is changed inside **ylast** using the syntax **k <<- k1**.

```
> derivs = function(x,y) {
+          c(y[2], -k^2*y[1] - y[2]/x ) }
> xL = seq(1e-15,1,0.01)
> fll(xL)
  1e-15   1   101
> head(xL)
[1] 1e-15 1e-02 2e-02 3e-02 4e-02 5e-02
> ylast = function(k1) {
+         k  <<-   k1
+         last (myrk4(c(1,0), xL, derivs) [[1]] )}
> ylast(2)
[1] 0.2238833
> ylast(3)
[1] -0.2600324
> kL = seq(1,15,0.5)
> fll(kL)
  1   15   29
> ylastL = sapply(kL, ylast)
> fll(ylastL)
  0.7651913   -0.01419687   29
> plot(kL, ylastL, type = "l", col = "blue", lwd = 3, xlab = "k",
+          ylab = "y(1)")
> mygrid()
```

which produces the plot

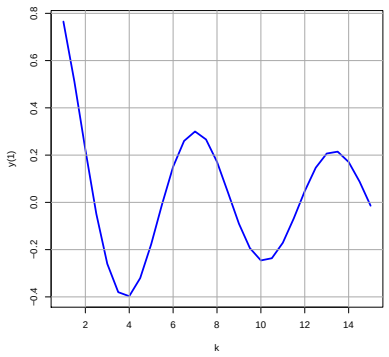


Figure 51: Numerical $y_k(1)$ versus $k$ using myrk4

We next make a small table of $y(1)$ values as a function of $k$, as implied by the numerical integration.

```
> kL = seq(1,15,1)
> fll(kL)
  1   15   15
> ylastL = sapply(kL, ylast)
> fll(ylastL)
  0.7651913   -0.01419687   15
> for (j in kL) cat(" ",kL[j]," ",ylastL[j],"\n")
  1   0.7651913
  2   0.2238833
  3   -0.2600324
  4   -0.3970967
  5   -0.1775594
  6   0.1506001
  7   0.2999558
  8   0.1715578
  9   -0.09027275
  10   -0.2457283
  11   -0.1710143
  12   0.04763221
  13   0.2066296
  14   0.1707881
  15   -0.01419687
```

We can then look for the eigenvalues $k_{0\,j}$ using **`ylast`** with **`uniroot`** with $k \in (2,3),(5,6),(8,9),(11,12)$, and compare the numerical values with the "analytic" values.

```
> k01 = uniroot(ylast, c(2,3), tol = 1e-10)$root
> k01
[1] 2.404826
> k01 - k01a
[1] -3.290844e-08
> k02 = uniroot(ylast, c(5,6), tol = 1e-10)$root
> k02
[1] 5.520077
> k02 - k02a
[1] -7.264406e-07
> k03 = uniroot(ylast, c(8,9), tol = 1e-10)$root
> k03
[1] 8.653725
> k03 - k03a
[1] -2.931348e-06
> k03 = uniroot(ylast, c(8,9), tol = 1e-12)$root
> k03
[1] 8.653725
> k03 - k03a
[1] -2.931348e-06
> k04 = uniroot(ylast, c(11,12), tol = 1e-12)$root
> k04
[1] 11.79153
> k04 - k04a
[1] -4.84016e-06
```

We can now make a plot comparing the numerical solution with the "analytic" solution for $k = k_1$.

```
> soln = function(k1) {
+           k  <<-  k1
+   myrk4( c(1,0), xL, derivs ) }
> soln1 = soln(k01)
> yL1 = soln1[[1]]
> fll(yL1)
  1   -1.834644e-14   101
> plot(xL,yL1,type = "l", col = "blue", lwd = 3, xlab = "x",ylab = "y(x)")
> mygrid()
> curve(besselJ(k01a*x, 0),0,15,col="red",lwd=3,add = TRUE,n=200)
> legend("topright", col = c("blue", "red"), lwd = 2,
+         legend = c("numerical", "analytic"), cex = 1.5)
```

which shows agreement

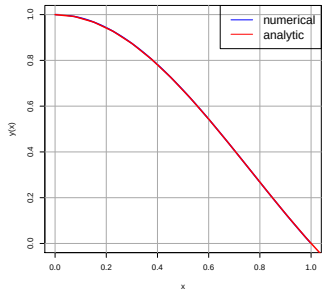


Figure 52: Numerical vs. Analytic $y(x)$ for $k = k_1$

We next compare the numerical vs. the "analytical" solution for $k = k_2$.

```
> soln2 = soln(k02)
> yL2 = soln2[[1]]
> fll(yL2)
  1   -2.081668e-17   101
> plot(xL,yL2,type = "l", col = "blue", lwd = 3, xlab = "x",ylab = "y(x)")
> mygrid()
> curve(besselJ(k02a*x, 0),0,15,col="red",lwd=3,add = TRUE,n=400)
> legend("topright", col = c("blue", "red"), lwd = 2,
+            legend = c("numerical", "analytic"), cex = 1.5)
```

which also shows agreement

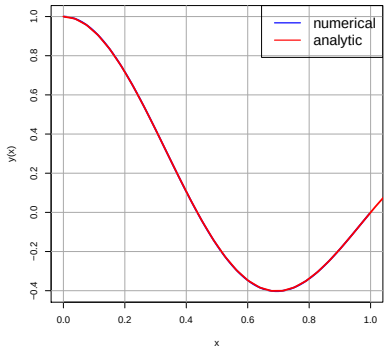


Figure 53: Numerical vs. Analytic $y(x)$ for $k = k_2$

# 6 1D Solutions of Schroedinger's Equation

Two examples of numerical solutions of Schroedinger's equation in 1D are worked out as examples in the file **`example3.pdf`**. These examples are a quantum particle in a finite rectangular potential well and a quantum particle in a potential well described by the Lennard-Jones 6/12 potential (energy).

In addition to **`cp3.mac`** and **`cp3.R`**, code files **`FW.mac`**, **`FW.R`**, **`LJ6-12.mac`**, and **`LJ6-12.R`** provide the needed method (both Runge-Kutta and Numerov methods are used) codes for these examples.

# 7 References

In addition to the references already mentioned above, the author has made use of the recent text

**Computational Physics: A Practical Introduction to Computational Physics and Scientific Computing, Athens, 2014, by Konstantinos Anagnostopoulos.**
**Free copies in various formats are available on the webpage:**
**http://www.physics.ntua.gr/~konstant/ComputationalPhysics/**

**The author's webpage is:**
**http://www.physics.ntua.gr/~konstant**