CS-7863: Scientific and Statistical Computing

Homework #5

Noah L. Schrick – 1492657

1.)

```
## 1. Systems of Equations in Matrix Form
```

a.)

```
# a. Use built-in R solve
A <- matrix(c(2,3,4,3,2,-3,4,4,2),nrow=3)
b <- matrix(c(3,5,9), nrow=3)

x <- solve(A,b)
x
```

```
> x
     [,1]
[1,]    1
[2,]   -1
[3,]    1
```

b.)

```
# b. Verify with matrix mult
all.equal(b, as.matrix(A %*% x))

# use all.equal instead of == (errors with tols and storage type)
# EX: identical(as.double(8), as.integer(8)) returns FALSE

> all.equal(b, as.matrix(A %*% x)) #
[1] TRUE
```

2.)

```
## 2. Matrix from polynomial
```

a.)

```
# a. Create vectors for x and y
xvec <- c(-3,-1.5,0.5,2,5)
yvec <- c(6.8,15.2,14.5,-21.2,10)
```

b.)

```
A.2 <- matrix(ncol=length(xvec), nrow=length(xvec))
for (i in 1:length(xvec)){
  A.2[,i] = xvec^(i-1)
}
A.2
```

```
> A.2
     [,1] [,2]  [,3]    [,4]     [,5]
[1,]    1 -3.0  9.00 -27.000  81.0000
[2,]    1 -1.5  2.25  -3.375   5.0625
[3,]    1  0.5  0.25   0.125   0.0625
[4,]    1  2.0  4.00   8.000  16.0000
[5,]    1  5.0 25.00 125.000 625.0000
```

c.)

```
# c. Solve
if (!require("matlib")) install.packages("matlib")
library(matlib)

# verify
bvec <- solve(A.2, yvec)
bvec
```

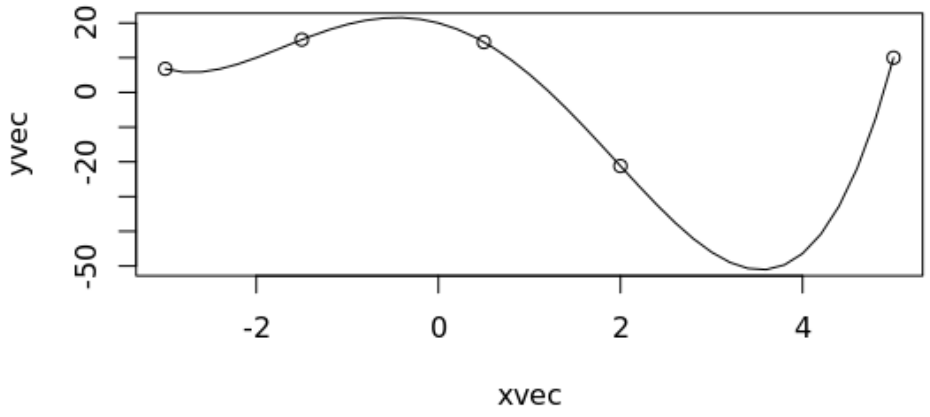
```
> bvec
[1] 20.0 -7.0 -8.0 -0.2  0.4
```

```
all.equal(as.matrix(yvec), A.2 %*% bvec)
```

```
[1] TRUE
```

d.)

```
# d. Plot
poly_predict <- function(x){
  # given input x, returns polynomial prediction
  sum(bvec*c(1,x,x^2,x^3,x^4))
}

xdomain <- seq(min(xvec),max(xvec),.2)    # predicted domain
y.predict <- sapply(xdomain,poly_predict) # predicted range
plot(xvec,yvec,ylim=c(-50,20))    # plot data
lines(xdomain,y.predict,type="l") # overlay solid line
```


yvec
20
0
-20
-50
-2
0
2
4
xvec

3.)

a.)

```
# a. Create vectors
# Resistance vec
Rvec <- c(5,10,5,15,10,20)
# Matrix A and vector b
A.3a <- matrix(c(Rvec[1],0,0,0,Rvec[5],Rvec[6],
                 0,Rvec[2],Rvec[3],Rvec[4],-Rvec[5],0,
                 1,-1,0,0,-1,0,
                 0,1,-1,0,0,0,
                 0,0,1,-1,0,0,
                 0,0,0,1,1,-1)
               ,nrow=6)
A.3a <- t(A.3a) # easier to conf that A is correct if def matrix as above then t()
V=200
b.3vec <- c(V,0,0,0,0,0)

# Solve
i.a <- solve(A.3a,b.3vec)
i.a

> i.a
[1] 6.153846 1.538462 1.538462 1.538462
[5] 4.615385 6.153846
```

b.)

```
# b. Repeat a. Use V=200, and R=(5,10,5,15,0,20)
Rvec.b <- c(5,10,5,15,0,20)
A.3b <- matrix(c(Rvec.b[1],0,0,0,Rvec.b[5],Rvec.b[6],
                 0,Rvec.b[2],Rvec.b[3],Rvec.b[4],-Rvec.b[5],0,
                 1,-1,0,0,-1,0,
                 0,1,-1,0,0,0,
                 0,0,1,-1,0,0,
                 0,0,0,1,1,-1)
               ,nrow=6)
A.3b <- t(A.3b) # easier to conf that A is correct if def matrix as above then t()

i.b <- solve(A.3b,b.3vec)
i.b

> i.b
[1] 8 0 0 0 8 8
```

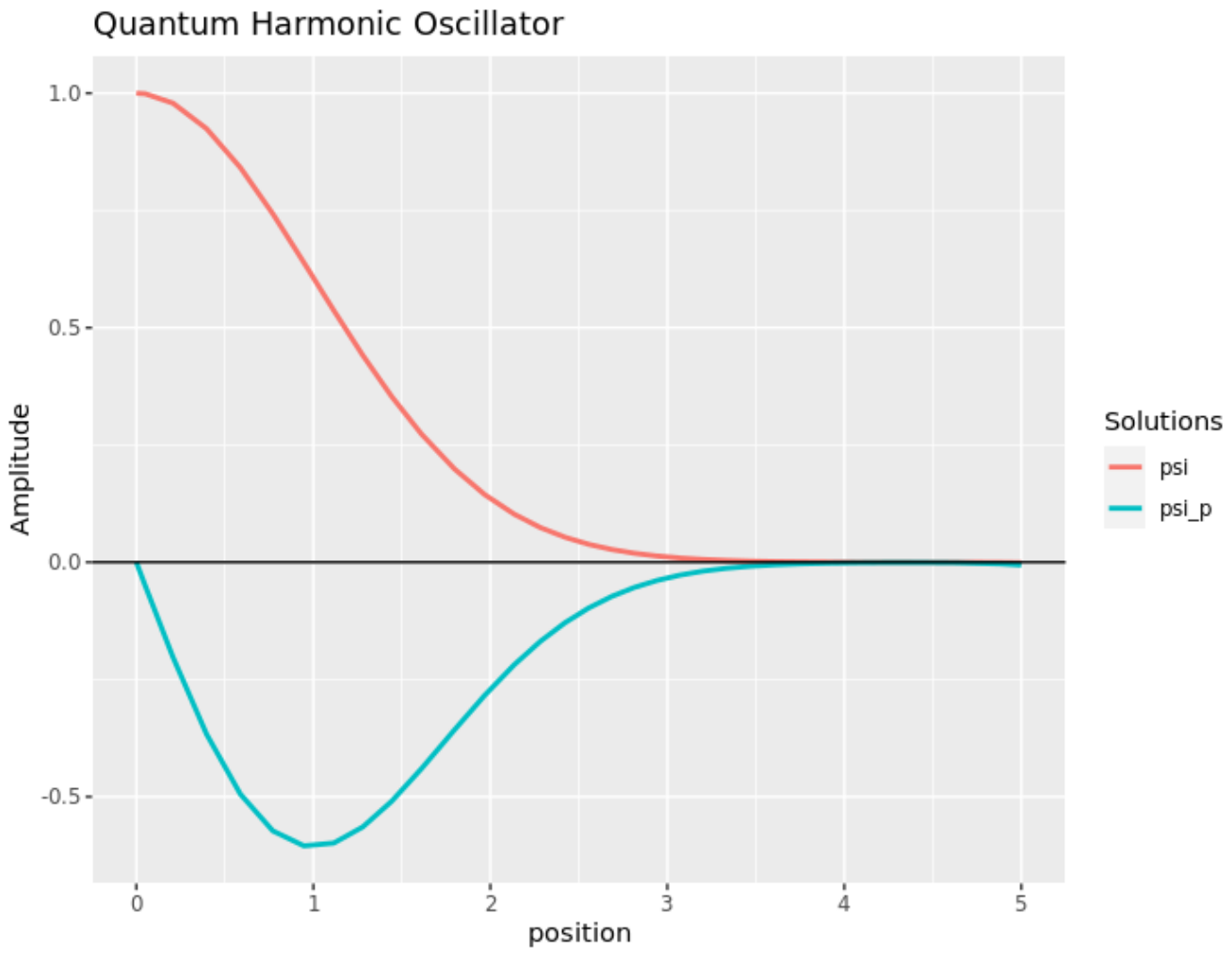
Since R5 is 0, the current will take the path of least resistance. From the voltage source, the current will flow through R1 and only flow through R5 and R6. Since the current through R5 is “infinite” (due to 0 resistance), no current will flow through R2, R3, or R4 (the left half of the circuit).

4.)

```
if (!require("pracma")) install.packages("pracma")
library(pracma)
```

a.)

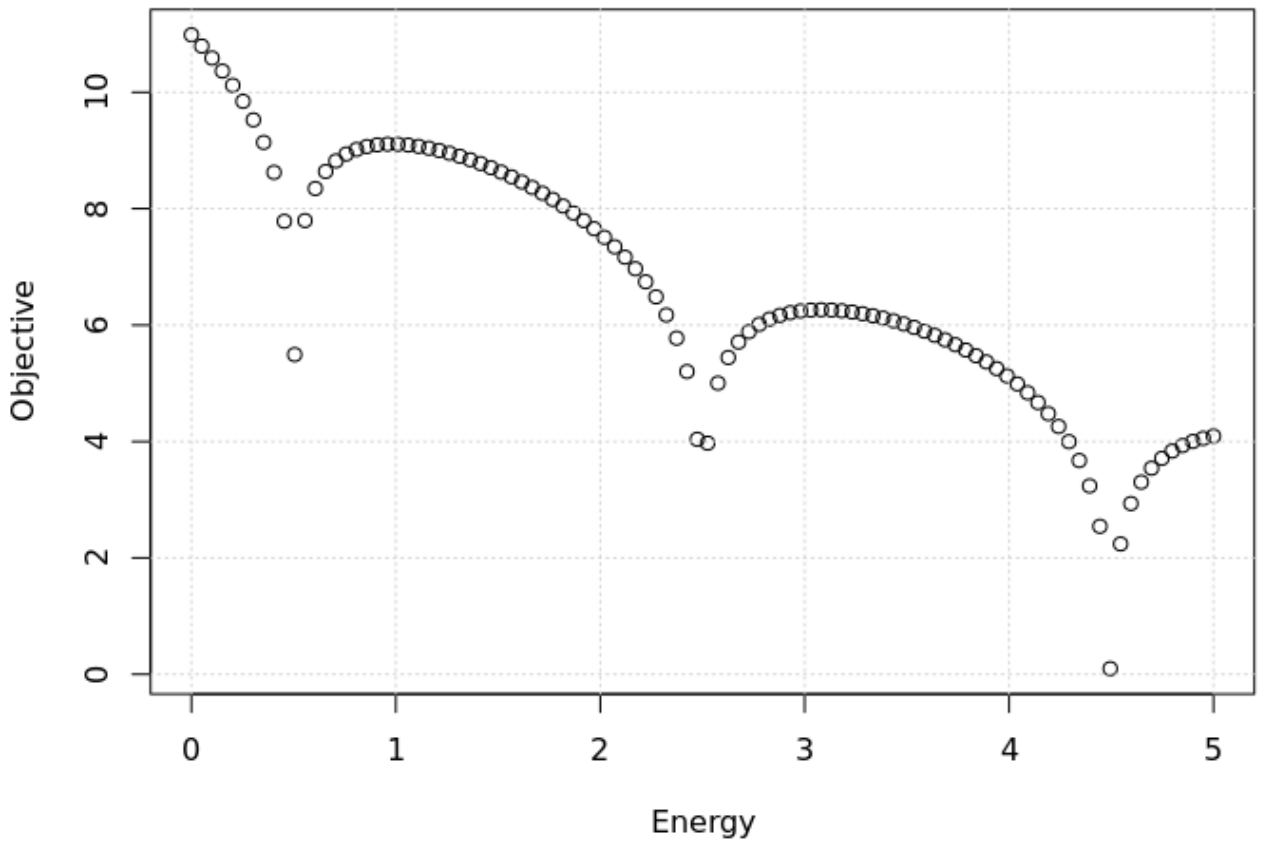
```
#a. Solve and plot the 1d quantum harmonic oscillator wave function
quantumHO <- function(x,y,E){
  psi     <- y[1]
  y1p <- y[2]   # y1' fill in blank
  y2p <- (x^2)*y[1]-2*E*y[1]  # y2' fill in blank

  as.matrix(c(y1p,y2p))
}

if (!require("reshape")) install.packages("reshape")
library(reshape)
if (!require("ggplot2")) install.packages("ggplot2")
library(ggplot2)
plotQHO <- function(E){
  # plot the wavefunction for a given energy
  QHO.sol <- ode45(function(x,y){quantumHO(x,y,E)},
                   y0=c(1,0),t0=0, tfinal=5)
  QHO.melted <- melt(data.frame(position=QHO.sol$t,
                                psi=QHO.sol$y[,1],
                                psi_p=QHO.sol$y[,2]),
                     id = "position")
  colnames(QHO.melted)[2] <- "Solutions"  # rename "variable"

  g <- ggplot(data = QHO.melted, aes(x=position, y=value, color=Solutions))
  g <- g + geom_line(size=1) + geom_abline(slope=0, intercept=0)
  g <- g + xlab("position") +  ylab("Amplitude")
  g <- g + ggtitle("Quantum Harmonic Oscillator")
  show(g)
}

plotQHO(0.5)
```



b.)

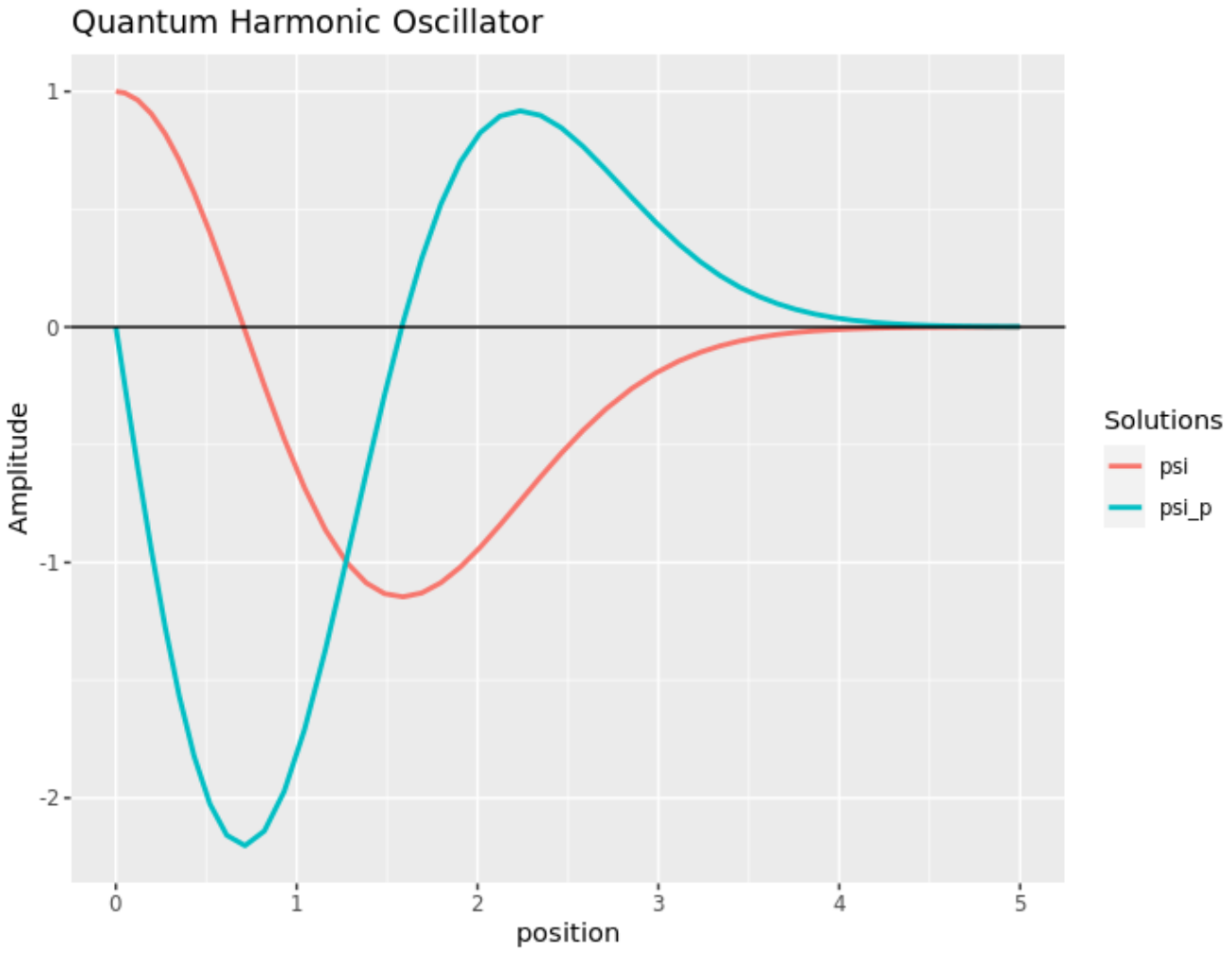
```
# b. Solve with Shooting Method
## Shooting Method Objective Function
QHO_shoot <- function(E){
  # E is an energy guess
  ho.sol <- ode45(function(x,y){quantumHO(x,y,E)},
                  y0=c(1,0),t0=0, tfinal=5)
  last_idx <- length(ho.sol$t)
  psi_right <- ho.sol$y[,1][last_idx] # psi at last x
  # We want the wave function at the right boundary to be
  # exponentially small. So, the log-abs value will be large
  # and negative for the correct solution.
  return(log(abs(psi_right)))
}

## The objective function plot informs the energy guesses below.
energy_vals <- seq(0,5.,len=100)
```

```
objective_vals <- sapply(energy_vals, QHO_shoot)
plot(energy_vals,objective_vals, xlab="Energy", ylab="Objective")
grid()
```




```
# “optimize” finds min or max of function in an interval.
# Find ground state energy E0.
E0<-optimize(QHO_shoot,interval=c(0,1),tol=1e-8)$minimum
E0
> E0
[1] 0.5

# Find first excited state E1.
E1<-optimize(QHO_shoot,interval=c(2,3),tol=1e-8)$minimum
E1
> E1
[1] 2.5
```

```
plotQHO(E1)
```


Quantum Harmonic Oscillator
Amplitude
position
Solutions
psi
psi_p
1
0
-1
-2
0
1
2
3
4
5

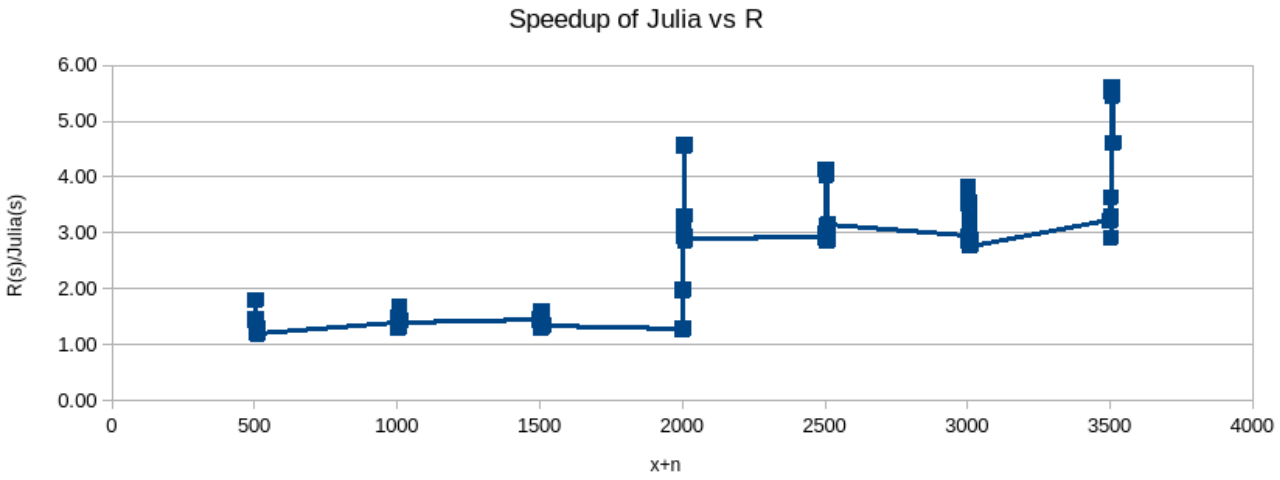
5.)

a.)

```
# a. Solve with R and Julia
if (!require("tictoc")) install.packages("tictoc")
library(tictoc)

solve_QHO <- function(n, xL, xR){
  #n<-1000
  #xL <- -10  # need to change for very excited state
  #xR <- 10
  h<- (xR-xL)/(n-2)  # step size
  x <- seq(xL, xR, by=h) # grid of x values

  H <- diag(1/h^2 + .5*x^2) # makes the diagnonal
  dim(H)
  #   diag       upper diag              lower diag
  H <- H + Diag(rep(-1/(2*h^2),n-2),1) + Diag(rep(-1/(2*h^2),n-2),-1)

  H.eigs <- eigen(H)
  vals <- H.eigs$values
  vecs <- H.eigs$vectors

  return(list(x=x,vecs=vecs,vals=vals))
}

print_QHO_sol <- function(sol){
  print("Smallest eigvalue, lowest energy:")
  print(vals[n-1])
  #vals[n-1]  # smallest eigvalue, lowest energy
  plot(x,vecs[,n-1], type="l", main="Ground state wave function") # ground state
wave function

  print("2nd smallest eigvalue, first excited energy:")
  print(vals[n-2])
  #vals[n-2]  # 2nd smallest eigvalue, first excited energy
  plot(x,vecs[,n-2], type="l", main="First excited wave function") # first excited
wave function

  print("3rd smallest eigvalue, second excited energy:")
  print(vals[n-3])
  #vals[n-3]  # 3rd smallest eigvalue, second excited energy
  plot(x,vecs[,n-3], type="l", main="Second excited wave function") # second
excited wave function
}
```

```
# data for timing comparison
timing.table <- matrix(nrow = 0, ncol = 4)
for (n in seq(500, 3500, by=500)){
  for (x in seq(1,10)){
    tic(quiet=TRUE)
    QHO_sol <- solve_QHO(n, -x, x)
    t <- toc(quiet=TRUE)
    timing.table <- rbind(timing.table, c(-x, x, n, t$callback_msg))
  }
}
```

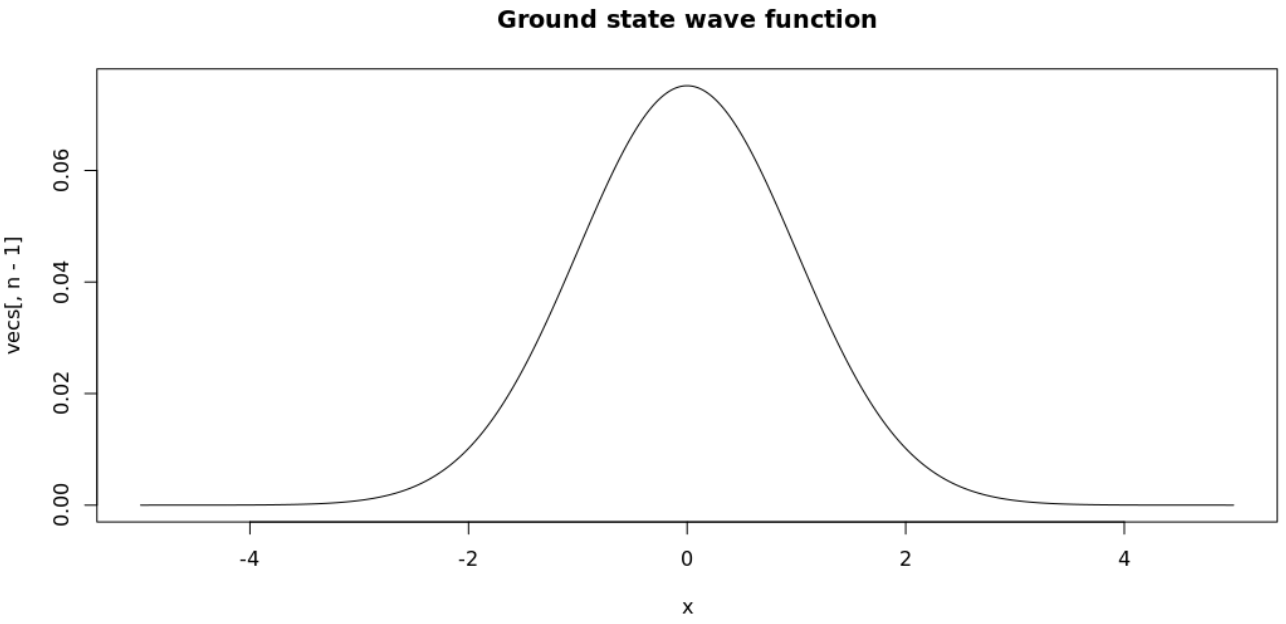
xL	xR	n	x+n	R (s)	Julia (s)	R/Julia
-1	1	500	501	0.214	0.148029	1.45
-2	2	500	502	0.22	0.150261	1.46
-3	3	500	503	0.278	0.154602	1.80
-4	4	500	504	0.216	0.14716	1.47
-5	5	500	505	0.22	0.155556	1.41
-6	6	500	506	0.214	0.150252	1.42
-7	7	500	507	0.206	0.158266	1.30
-8	8	500	508	0.191	0.157053	1.22
-9	9	500	509	0.203	0.162109	1.25
-10	10	500	510	0.201	0.16775	1.20
-1	1	1000	1001	1.585	1.13648	1.39
-2	2	1000	1002	1.732	1.15683	1.50
-3	3	1000	1003	1.559	1.16679	1.34
-4	4	1000	1004	1.626	1.10247	1.47
-5	5	1000	1005	1.459	1.11232	1.31
-6	6	1000	1006	1.916	1.13689	1.69
-7	7	1000	1007	1.571	1.13103	1.39
-8	8	1000	1008	1.656	1.16069	1.43
-9	9	1000	1009	1.632	1.15089	1.42
-10	10	1000	1010	1.654	1.17742	1.40
-1	1	1500	1501	5.336	3.67248	1.45
-2	2	1500	1502	5.255	3.7029	1.42
-3	3	1500	1503	4.949	3.82769	1.29
-4	4	1500	1504	4.887	3.74227	1.31
-5	5	1500	1505	5.155	3.9338	1.31
-6	6	1500	1506	5.975	3.76876	1.59
-7	7	1500	1507	5.325	3.69101	1.44
-8	8	1500	1508	5.384	3.80243	1.42
-9	9	1500	1509	5.266	3.83773	1.37
-10	10	1500	1510	5.146	3.86154	1.33
-1	1	2000	2001	12.23	9.59251	1.27
-2	2	2000	2002	12.674	6.38925	1.98
-3	3	2000	2003	13.436	4.29539	3.13
-4	4	2000	2004	12.904	4.31519	2.99
-5	5	2000	2005	12.595	4.25795	2.96
-6	6	2000	2006	16.838	5.11118	3.29
-7	7	2000	2007	19.558	4.27903	4.57
-8	8	2000	2008	12.563	4.30127	2.92
-9	9	2000	2009	11.918	4.1489	2.87
-10	10	2000	2010	12.697	4.3977	2.89
-1	1	2500	2501	24.332	8.31164	2.93
-2	2	2500	2502	24.879	8.32744	2.99
-3	3	2500	2503	35.73	8.63275	4.14
-4	4	2500	2504	25.098	8.42029	2.98
-5	5	2500	2505	24.981	8.41784	2.97
-6	6	2500	2506	33.57	8.34807	4.02
-7	7	2500	2507	25.033	8.36885	2.99
-8	8	2500	2508	24.515	8.60565	2.85
-9	9	2500	2509	26.322	8.42938	3.12
-10	10	2500	2510	26.865	8.5208	3.15
-1	1	3000	3001	42.765	14.5234	2.94
-2	2	3000	3002	54.857	14.3747	3.82
-3	3	3000	3003	50.735	14.3307	3.54
-4	4	3000	3004	42.273	14.6435	2.89
-5	5	3000	3005	41.84	14.6837	2.85
-6	6	3000	3006	46.412	14.2687	3.25
-7	7	3000	3007	45.485	14.6746	3.10
-8	8	3000	3008	41.874	14.4266	2.90
-9	9	3000	3009	41.92	14.633	2.86
-10	10	3000	3010	39.972	14.4246	2.77
-1	1	3500	3501	73.341	22.7882	3.22
-2	2	3500	3502	82.248	22.588	3.64
-3	3	3500	3503	65.727	22.5377	2.92
-4	4	3500	3504	75.375	22.8299	3.30
-5	5	3500	3505	125.22	22.5725	5.55
-6	6	3500	3506	126.6	22.6165	5.60
-7	7	3500	3507	124.844	22.8842	5.46
-8	8	3500	3508	125.837	22.9572	5.48
-9	9	3500	3509	126.342	23.0635	5.48
-10	10	3500	3510	128.125	27.8173	4.61

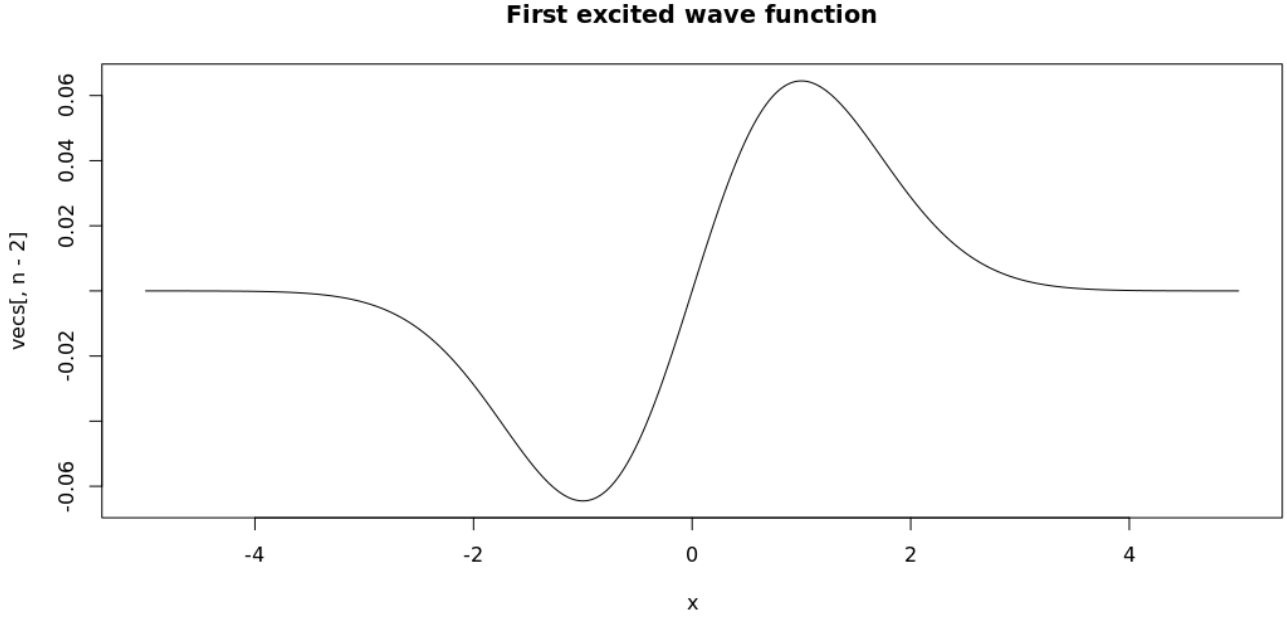


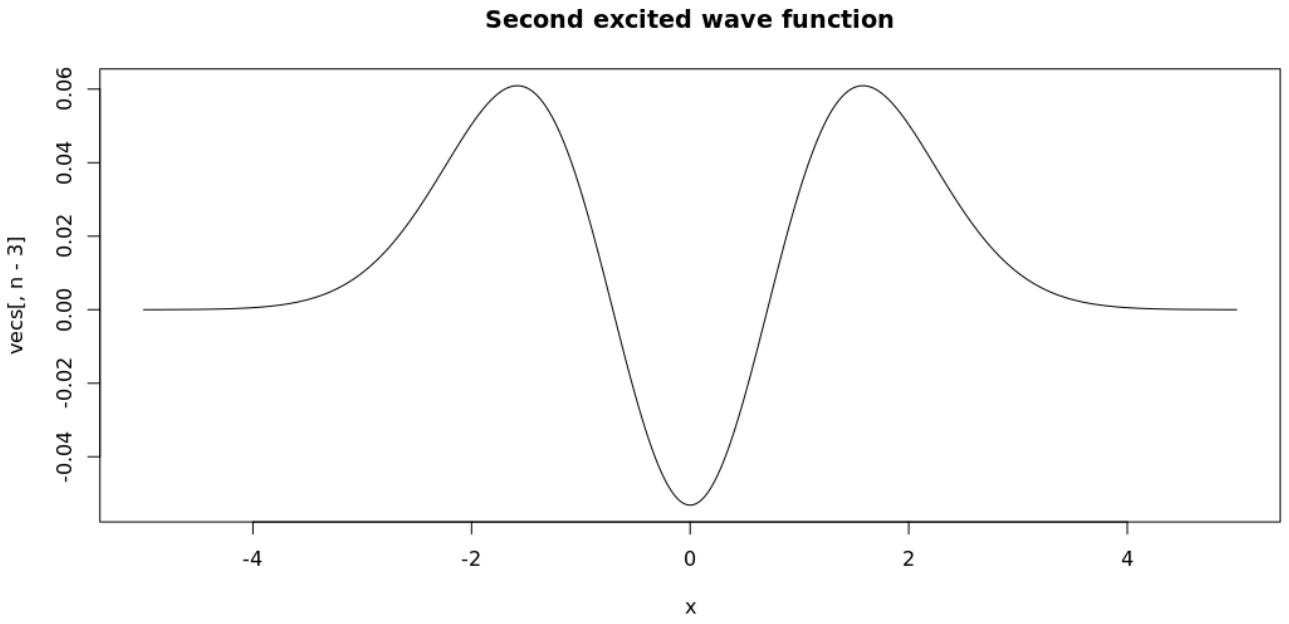
An inner for loop was used to iterate through values of xL and xR. For simplicity, xL=-xR. "n" was looped from 500 to 3500 in an outer loop with an increment of 500. Timing was collected for both R and Julia. For displaying the data, the x-axis is xR+n. Even with low values of n, the speedup obtained from Julia was noticeable, reaching almost 2.0 when xR=10.

```
# answer to main question
QHO_sol <- solve_QHO(1000, -5, 5)
print_QHO_sol(QHO_sol$x, QHO_sol$vecs,QHO_sol$vals,1000)
```

```
> print_QHO_sol(QHO_sol$x, QHO_sol$vecs,QHO_sol$vals,1000)
[1] "Smallest eigvalue, lowest energy:"
[1] 0.4999969
[1] "2nd smallest eigvalue, first excited energy:"
[1] 1.499984
[1] "3rd smallest eigvalue, second excited energy:"
[1] 2.499959
```


Ground state wave function
vecs[, n - 1]
0.00
0.02
0.04
0.06
-4
-2
0
2
4
x

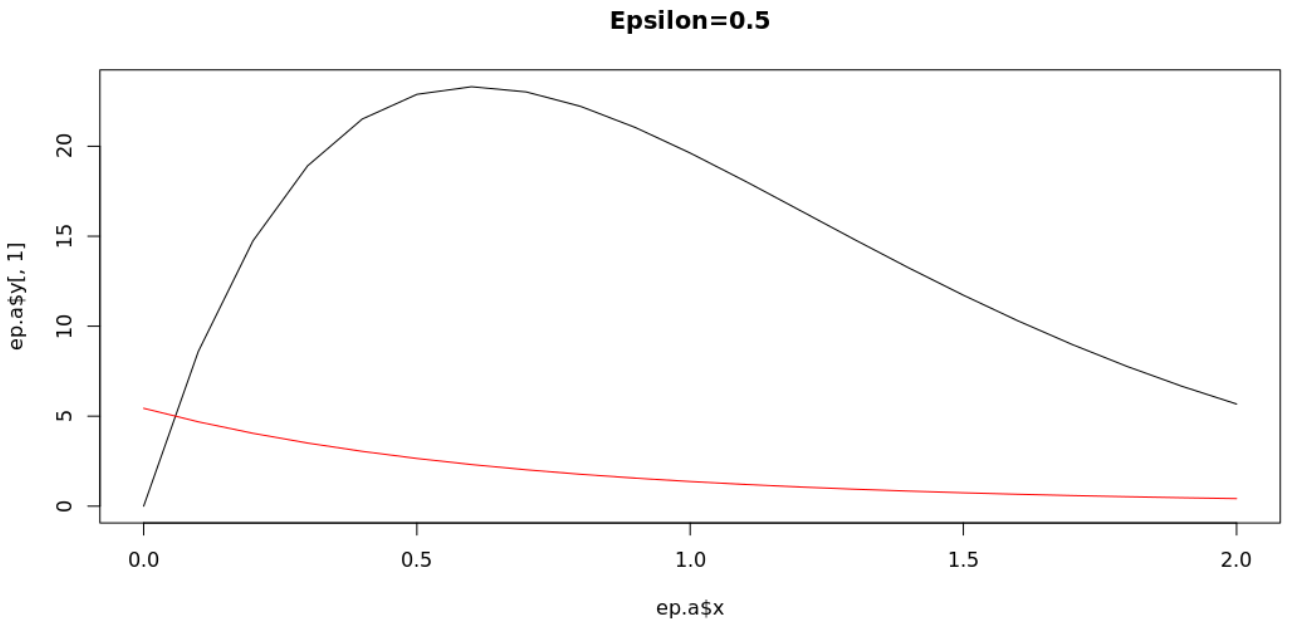

First excited wave function
vecs[, n - 2]
-0.06
-0.02
0.02
0.04
0.06
-4
-2
0
2
4
x

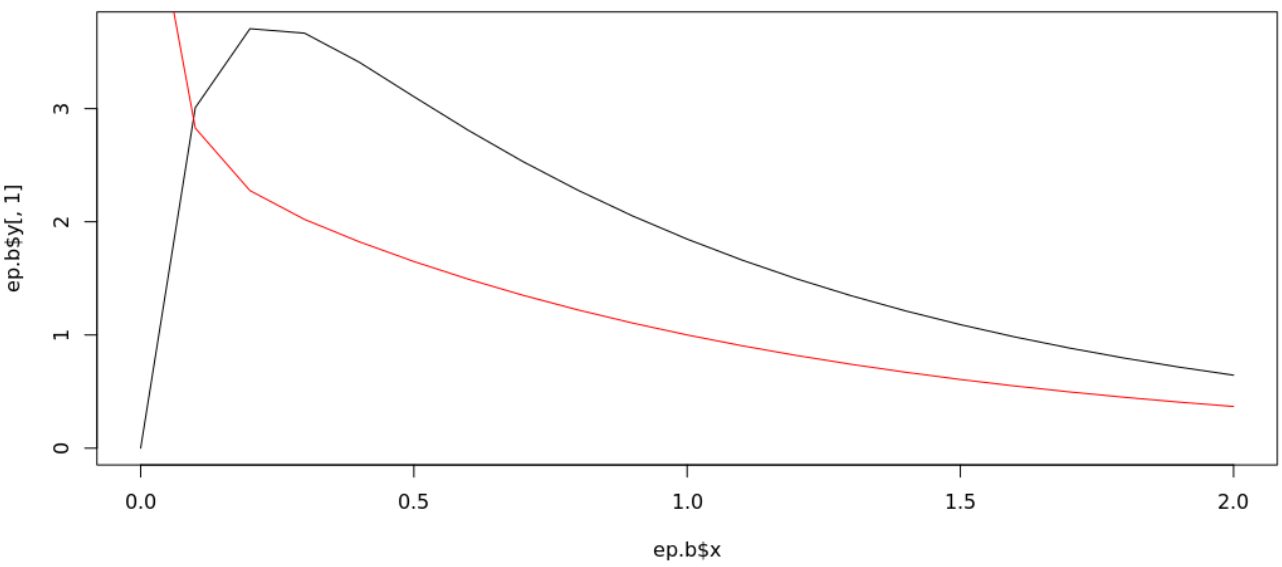


b.)

```
# b. Solve damped oscillator with perturbation param
yo <- 0
y1 <- 1
tmin <- 0
tfinal <- 2

dho.pert.encap.f <- function(ep){
  dho.pert.f <- function(t, y){
    yn <- y[1]
    ydot <- y[2] - yn
    ypp <- (-(1+ep)*yn - ydot)/ep
    as.matrix(c(ydot,ypp))
  }

  proj.obj <- function(v0, y0=yo, tfinal){
    # minimize w.r.t. v0
    proj.sol <- ode45(dho.pert.f,
                      y=c(yo, y1), t0=tmin, tfinal=tfinal)
    final_index <- length(proj.sol$t)
    yf <- proj.sol$y[final_index,1]  # want equal to right boundary
    log(abs(yf)) # minimize this
  }
```

```
  # user specifies tfinal and yfinal for BVP
  ydot_best <- optimize(proj.obj,
                        interval=c(1,100), #bisect-esque interval
                        tol=1e-10,
                        y0=0, tfinal=2) # un-optimized obj params
  ydot_best$minimum  # best ydot

  best.sol <- rk4sys(dho.pert.f, a=0, b=2, y0=c(0, ydot_best$minimum),
                     n=20)  # 20 integration stepstmax
}

approx <- function(x){
  exp(1-(x/ep)) + exp(1-x)
}

ep <- 0.5
ep.a <- dho.pert.encap.f(ep)
plot(ep.a$x, ep.a$y[,1], type="l", main="Epsilon=0.5")
par(new=T)
lines(ep.a$x, approx(seq(0,2,len=21)), type="l", col="red")

ep <- 0.05
ep.b <- dho.pert.encap.f(ep)
plot(ep.b$x, ep.b$y[,1], type="l", main="Epsilon=0.05")
par(new=T)
lines(ep.b$x, approx(seq(0,2,len=21)), type="l", col="red")
```




Epsilon=0.05
ep.b$y[, 1]
0
1
2
3
0.0
0.5
1.0
1.5
2.0
ep.b$x

6.)

```
function Laplace_time(;alpha, L, T, h, u_tblr)
  # ; lets you ref arguments by name
  # Laplace_time(alpha=20, L=25, T=)
  # u_tblr = [100, 0, 0, 0] # boundary conditions
  dt = h^2/(4*alpha+1);  # ensures numerical stability
  n = floor(Int64, L/h); # coerce to integer, spatial loops
  m = floor(Int64, T/dt); # number of time steps
  u = zeros(n,n,m) # intialize solution 3d array

  function impose_bc(u_tblr, u, n, k)
      u_top    = u_tblr[1];   u[1,:,k] .= u_top; # top bc at time k
      u_bottom = u_tblr[2];   u[n,:,k] .= u_bottom; # top bc at time k
      u_left   = u_tblr[3];   u[:,1,k] .= u_left; # top bc at time k
      u_right  = u_tblr[4];   u[:,n,k] .= u_right; # top bc at time k
      return u
  end
  u = impose_bc(u_tblr, u, n, 1)  # intial u at time k=1
  for k in range(2,m-1)          # time
    for i in range(2,n-1)      # x
        for j in range(2,n-1) # y
          u[i,j,k+1] = (1-(4*alpha*dt)/(h^2))*u[i,j,k] +
((alpha*dt)/(h^2))*(u[i+1,j,k]+u[i-1,j,k]+u[i,j+1,k]+u[i,j-1,k]) # method of lines
        end
    end
    u = impose_bc(u_tblr, u, n, k) # re-impose the bc
  end
  return u
end
```

The heatmaps and GIFs were saved, but were unable to be viewed. The Julia code was ran on a Linux machine, and there are open bug reports regarding animations on a plot window for Julia. I performed a moderate amount of bug-fixes, but ultimately started spending too much time trying to get the plot window to function properly.

The GIFs are saved, and I am able to send or submit them as needed.