Introduction to computational programming

Chapter 6 Exercise

Eigenvalues, eigenvectors, and phase planes

Solutions

David M. Rosenberg
University of Chicago
Committee on Neurobiology

Version control information:
Last changed date: 2009-11-19 11:47:22 -0600 (Thu, 19 Nov 2009)
Last changes revision:271
Version: Revision 271
Last changed by: David M. Rosenberg

November 19, 2009

1. *Finding eigenvalues.* There is an *R* function `eigen()` which calculates the eigenvalues and eigenvectors of a matrix. Without using this function:

   (a) Write a function that takes a 2×2 matrix and returns the eigenvalues of the matrix.

```
> calcEigenValues <- function(mat) {
+   a <- 1 + 0i;
+   b <- as.complex(mat[1, 1] + mat[2, 2]);
+   c <- as.complex(mat[1, 1] * mat[2, 2] - mat[2, 1] * mat[1, 2]);
+   result <- c( (b + sqrt(b^2 - 4 * a * c)) / (2 * a),
+                (b - sqrt(b^2 - 4 * a * c)) / (2 * a) );
+   if (Mod(result[1]) < Mod(result[2])) {
+     result <- c(result[2], result[1]);
+   }
+   return(result);
+ }
```

   (b) Use the provided function, `testEigenFunction()` to verify that your method produces *correct* results for any 2×2 matrix.

```
> testEigenFunction <- function(fun, nTests=50) {
+   for (ii in 1:nTests) {
+     A <- matrix(floor(runif(n=4, min=-100, max=100)), nrow=2);
+     sysResult <- as.complex(eigen(A)$values);
+     testResult <- as.complex(fun(A));
+     if (! (isTRUE(all.equal(sysResult, testResult)) ||
+            isTRUE(all.equal(sysResult, testResult[c(2, 1)])) ) ) {
```

```
+        cat(paste('Error encountered in test #', ii,
+                  ', further tests aborted.\n', sep='') );
+        return(list(testMatrix=A, systemResult=sysResult,
+                    testResult=testResult));
+      }
+    }
+    cat(paste('Success.  All ', nTests,
+              ' tests completed successfully.\n', sep=''));
+ }
```

```
> testEigenFunction(calcEigenValues, nTests=100);

Success.  All 100 tests completed successfully.
```

(c) Use your function to find the eigenvalues of the following matrices. Classify either eigenvalues by the sign of the real part and the existence / type of the imaginary part (e.g. positive real part, nonzero imaginary part).

i. $\begin{pmatrix} 0 & -2 \\ 1 & 0 \end{pmatrix}$

ii. $\begin{pmatrix} 3 & 2 \\ 4 & 1 \end{pmatrix}$

iii. $\begin{pmatrix} -4 & -2 \\ 3 & -1 \end{pmatrix}$

iv. $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

i. $\begin{pmatrix} 0 & -2 \\ 1 & 0 \end{pmatrix}$

```
> mat1 <- matrix(c(0, -2, 1, 0), nrow=2, byrow=TRUE);
> calcEigenValues(mat1);

[1] 0+1.414214i 0-1.414214i
```

Both eigenvalues have a nonzero imaginary part and a real part of zero.

ii. $\begin{pmatrix} 3 & 2 \\ 4 & 1 \end{pmatrix}$

```
> mat2 <- matrix(c(3, 2, 4, 1), nrow=2, byrow=TRUE);
> calcEigenValues(mat2);

[1]  5+0i -1+0i
```

Neither eigenvalue has a nonzero imaginary part. One eigenvalue has a positive real part and the other has a negative real part.

iii. $\begin{pmatrix} -4 & -2 \\ 3 & -1 \end{pmatrix}$

```
> mat3 <- matrix(c(-4, -2, 3, -1), nrow=2, byrow=TRUE);
> calcEigenValues(mat3);

[1] -2.5-1.936492i -2.5+1.936492i
```

Both eigenvalues have negative real parts and nonzero imaginary parts.

iv. $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

```
> mat4 <- matrix(c(2, 1, 1, 2), nrow=2, byrow=TRUE);
> calcEigenValues(mat4);

[1] 3+0i 1+0i
```

Both eigenvalues have positive real parts and zero imaginary parts.

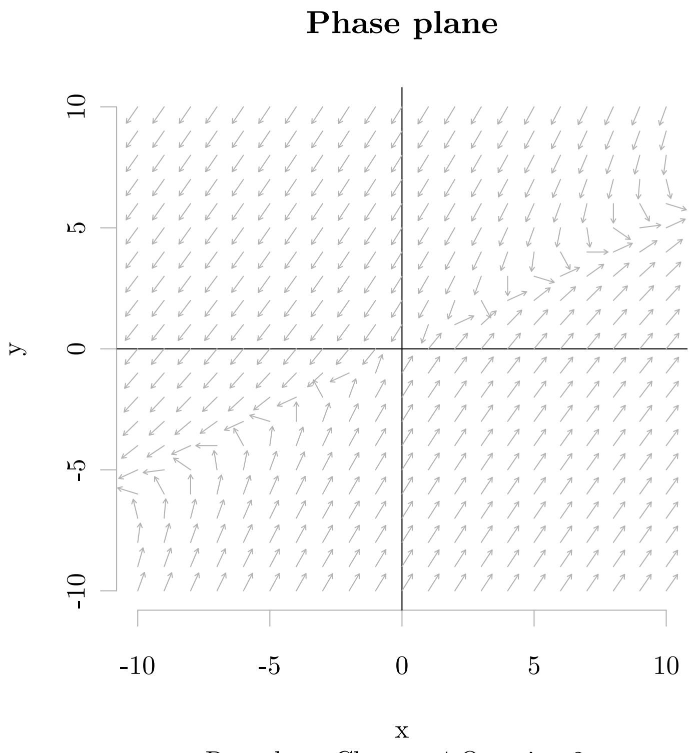
2. *Phase planes.*

   (a) Write a function which takes as arguments a 2×2 matrix, a range of x values and a range of y values and draws a phase plane for the system over the specified range. You may use the provided function `normalizeVector()`. The resulting function should generate plots similar to the example below.

```
> normalizeVector <- function(vec, total=0.8) {
+   iLength <- sqrt(sum(vec ^ 2));
+   return(vec * total / iLength)
+ }
```

```
> drawPhasePlane <- function(mat, xRange, yRange, main='Phase plane',
+                            question='') {
+   xVals <- seq(from=xRange[1], to=xRange[2], by=diff(xRange)/20);
+   yVals <- seq(from=yRange[1], to=yRange[2], by=diff(yRange)/20);
+
+   plot(xRange, yRange, type='n', bty='n', fg=grey(0.7),
+        main=main, xlab='x', ylab='y',
+        sub=sprintf('Rosenberg %s', question));
+   abline(h=0, v=0);
+
+   for (xx in xVals) {
+     for (yy in yVals) {
+       xy1 <- mat %*% matrix(c(xx, yy), nrow=2)
+       xy1 <- normalizeVector(xy1)
+       arrows(xx, yy, xx + xy1[1], yy + xy1[2], length=0.025,
+              col=grey(0.7));
+     }
+   }
+ }
```

```
> mat <- matrix(c(3, −4, 4, −7), nrow=2, byrow=TRUE)
> drawPhasePlane(mat, c(−10, 10), c(−10, 10),
+                  question='Chapter 4 Question 2a');
```



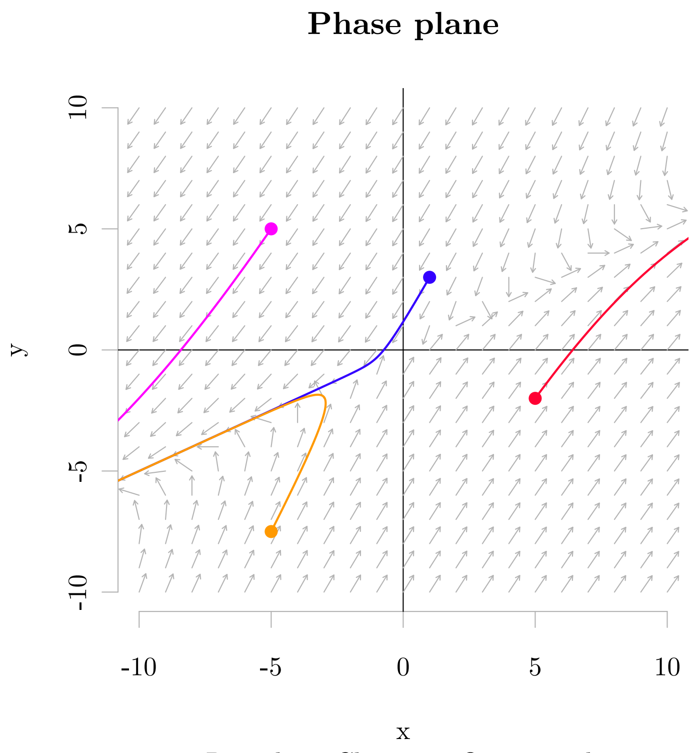
Rosenberg Chapter 4 Question 2a

(b) Modify your function so that it takes an additional argument specifying a set of initial values and plots the phase plane *and* the solution line for the given initial conditions. You may use the provided function `solveOdeSystem()`. Your function should be take similar arguments and produce similar results as the example below.

```
> solveOdeSystem <- function(mat, x, y) {
+   e <- eigen(mat);
+   eVectors <- e$vectors
+   eVals <- e$values;
+   ab <- solve(eVectors, matrix(c(x, y), nrow=2));
+   A <- ab[1];
+   B <- ab[2];
+   t <- seq(0, 100, by=0.01);
+   xx <- (A * eVectors[1, 1]) * exp(eVals[1] * t) +
+          (B  * eVectors[1, 2]) * exp(eVals[2] * t);
+   yy <- (A * eVectors[2, 1]) * exp(eVals[1] * t) +
+          (B  * eVectors[2, 2]) * exp(eVals[2] * t);
+   return(list(x=xx, y=yy));
+ }
```

```
> drawPhasePlane2 <- function(mat, xRange, yRange, initialValues=NULL,
+                            main='Phase plane', question='') {
+   drawPhasePlane(mat, xRange, yRange, main, question);
+
+   if(!is.null(initialValues)) {
+     cols <- rainbow(n=length(initialValues), start=0.7, end=0.1)
+     for (ii in 1:length(initialValues)) {
+       soln <- solveOdeSystem(mat, initialValues[[ii]][1],
+                              initialValues[[ii]][2] );
+       lines(soln$x, soln$y, col=cols[ii], lwd=2);
+       points(soln$x[1], soln$y[1], col=cols[ii], pch=19);
+     }
+   }
+ }
```

```
> initialValues <- list(c(1, 3), c(-5, 5), c(5, -2), c(-5, -7.5))
> drawPhasePlane2(mat, c(-10, 10), c(-10, 10), initialValues,
+                 question='Chapter 6 Question 2b');
```

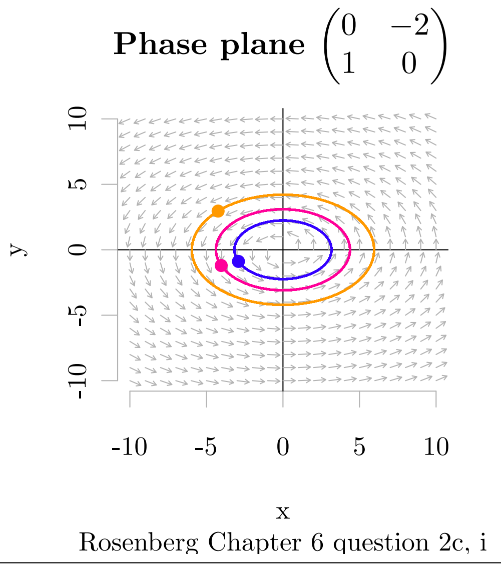


(c) Use this function to produce plots of the phase plane flow for the system

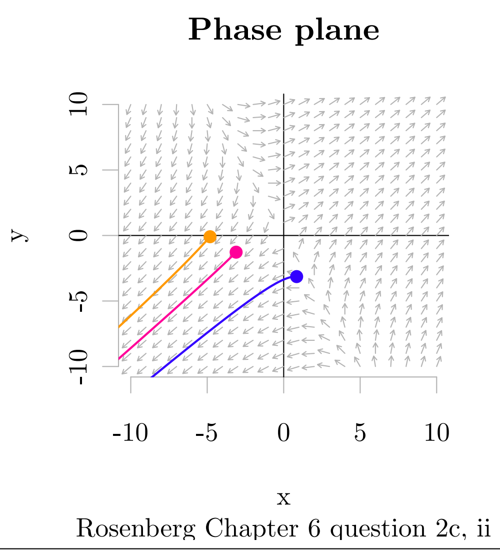
$$\frac{d\vec{x}}{dt} = \mathbf{A}\vec{x}$$

for each of the matrices **A** in problem 1. Plot two different solution trajectories for each system.

```
> layout(matrix(1:2, nrow=1));
> xrange <- yrange <- c(-10, 10);
> iVals <- runif(n=6, min=-5, max=5);
> drawPhasePlane2(mat1, xrange, yrange,
+                 initialValues=list(iVals[1:2], iVals[3:4], iVals[5:6]),
+                 question='Chapter 6 question 2c, i',
+         main='Phase plane $\\begin{pmatrix} 0 & -2 \\\\ 1 & 0 \\end{pmatrix}$')
    ;
> iVals <- runif(n=6, min=-5, max=5);
> drawPhasePlane2(mat2, xrange, yrange,
+                 initialValues=list(iVals[1:2], iVals[3:4], iVals[5:6]),
+                 question='Chapter 6 question 2c, ii');
```

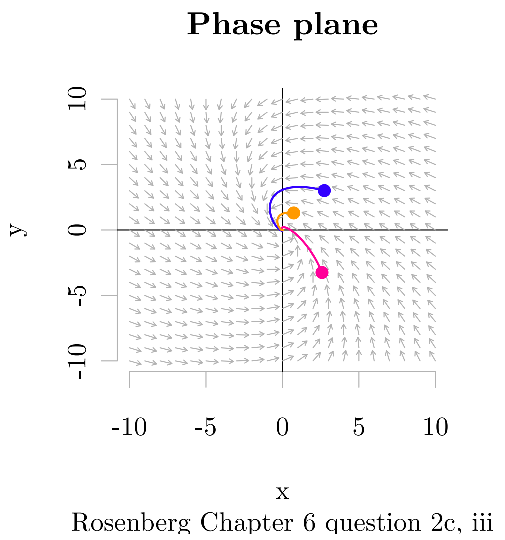


Rosenberg Chapter 6 question 2c, i

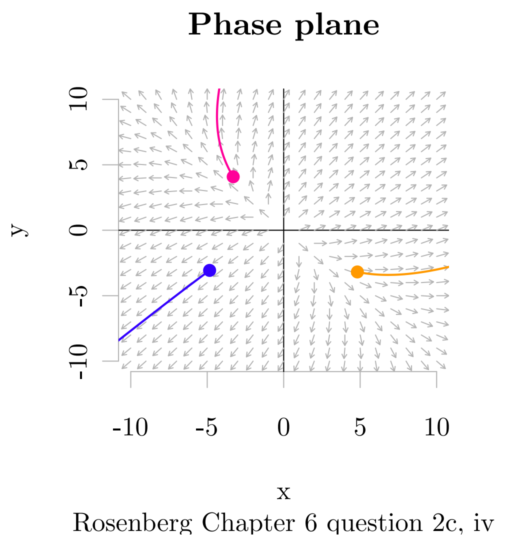


Rosenberg Chapter 6 question 2c, ii

```
> layout(matrix(1:2, nrow=1));
> iVals <- runif(n=6, min=-5, max=5);
> drawPhasePlane2(mat3, xrange, yrange,
+                 initialValues=list(iVals[1:2], iVals[3:4], iVals[5:6]),
+                 question='Chapter 6 question 2c, iii');
> iVals <- runif(n=6, min=-5, max=5);
> drawPhasePlane2(mat4, xrange, yrange,
+                 initialValues=list(iVals[1:2], iVals[3:4], iVals[5:6]),
+                 question='Chapter 6 question 2c, iv');
```



Rosenberg Chapter 6 question 2c, iii



Rosenberg Chapter 6 question 2c, iv

3. *Model of relationships: symmetric lovers.* Let us consider the model of relationships between two lovers, with respective feelings quantified by variables X and Y. Consider the following situation:

$$\begin{pmatrix} \dot{X} \\ \dot{Y} \end{pmatrix} = \begin{pmatrix} a & b \\ b & a \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix}$$

(a) Explain what a and b represent, and what their possible range is.

In this scenario, the a variables represent the response of both individuals towards to their own emotional state and the b variables represent the reciprocal effect of each individual's emotional state on the other.

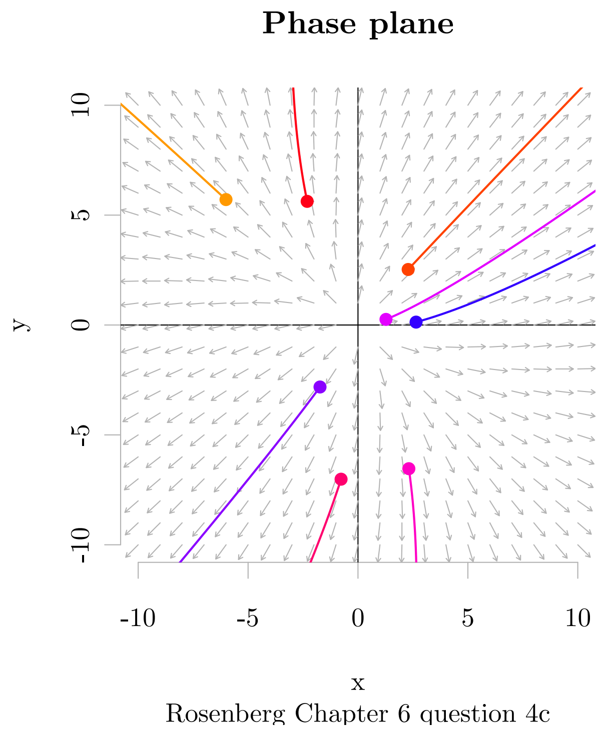
(b) Find the eigenvalues of the matrix by using the eigenvalue formula, and classify the qualitatively different types of behavior possible for varying values of a and b.

$$\begin{aligned}
\tau &= a + a = 2a \\
\Delta &= a * a - b * b = a^2 - b^2 \\
\lambda &= \frac{\tau \pm \sqrt{\tau^2 - 4\Delta}}{2} \\
&= \frac{2a \pm \sqrt{4a^2 - 4(a^2 - b^2)}}{2} \\
&= \frac{2a \pm 2b}{2} \\
&= a \pm b
\end{aligned}$$

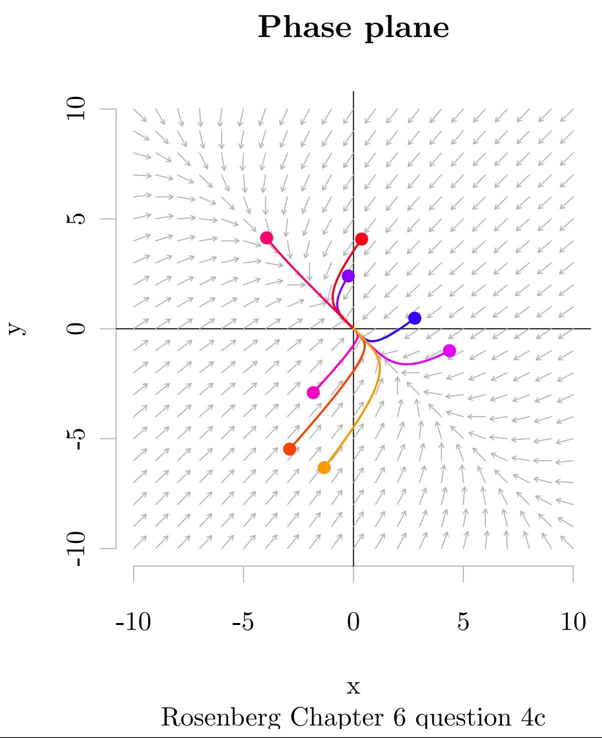
In this scenario, only real eigenvalues are possible, so no spiral / cyclic behavior will be observed. All three possible combinations of signs for these (real) eigenvalues are possible, however. If $a < b < 0$, then both eigenvalues will be negative and the relationship will move to extinction. Conversely, if $a > b > 0$, then both eigenvalues will be positive and the "emotional content" of the couple will increase without bound. If neither of these conditions is met, then one eigenvalue will be positive and one negative, resulting in "saddle-point" behavior at the origin. In this case, the long-term outcome for the couple is dependent upon the initial conditions.

(c) Pick representative values of a and b and produce phase plane plots using the function from problem 2. Explain what long-term predictions you can make from the phase plane plot for the relationship.

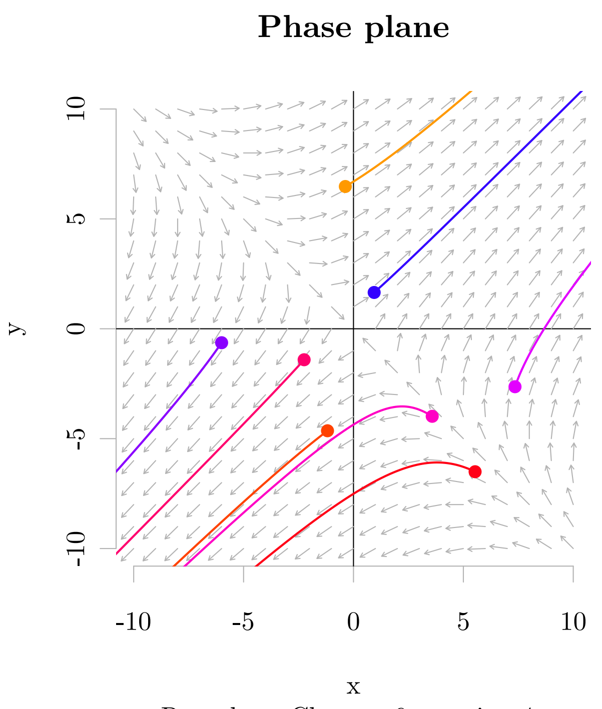
```
> layout(1);
> ab <- c(0.9, 0.2);
> rMat1 <- matrix(c(ab[1], ab[2], ab[2], ab[1]), nrow=2, byrow=TRUE);
> iVals <- runif(n=16, min=-7.5, max=7.5)
> initVals <- list();
> for (ii in 1:8) { initVals[[ii]] <- iVals[c(ii, ii + 8)] }
> drawPhasePlane2(rMat1, xrange, yrange, initVals,
+                   question='Chapter 6 question 4c');
```



```
> ab <- c(-0.6, -0.4);
> rMat2 <- matrix(c(ab[1], ab[2], ab[2], ab[1]), nrow=2, byrow=TRUE);
> iVals <- runif(n=16, min=-7.5, max=7.5)
> initVals <- list();
> for (ii in 1:8) { initVals[[ii]] <- iVals[c(ii, ii + 8)] }
> drawPhasePlane2(rMat2, xrange, yrange, initVals,
+                 question='Chapter 6 question 4c');
```



```
> ab <- c(0.5, 0.8);
> rMat3 <- matrix(c(ab[1], ab[2], ab[2], ab[1]), nrow=2, byrow=TRUE);
> iVals <- runif(n=16, min=-7.5, max=7.5)
> initVals <- list();
> for (ii in 1:8) { initVals[[ii]] <- iVals[c(ii, ii + 8)] }
> drawPhasePlane2(rMat3, xrange, yrange, initVals,
+                  question='Chapter 6 question 4c');
```



Rosenberg Chapter 6 question 4c

4. *Bonus question. Solving a linear system.* Write a function which takes as arguments two functions in x and y representing the linear functions $\dot{x}$ and $\dot{y}$ and an initial set of values x_0 and y_0 and returns a function representing the solution to the system. For example

```
> dotX <- function(x, y) {
+   return(3 * x - 4 * y);
+ }
> dotY <- function(x, y) {
+   return(4 * x - 7 * y);
+ }
> solveSystem(dotX, dotY, x0=1, y0=3);

$x
function (t)
{
    return(5/3 * exp(-5 * t) - 2/3 * exp(t))
}
<environment: 0x1156b0660>

$y
function (t)
{
    return(10/3 * exp(-5 * t) + -1/3 * exp(t))
}
<environment: 0x1156b0660>
```

```
> solveSystemSoln <- function(dotX, dotY, x0, y0) {
+   mat <- matrix(c(dotX(1, 0), dotX(0, 1), dotY(1, 0), dotY(0, 1)),
+                 nrow=2, byrow=TRUE)
+   e <- eigen(mat);
+   ab <- solve(e$vectors, matrix(c(x0, y0), nrow=2));
+   a <- ab[1];
+   b <- ab[2];
+   y <- x <- function(t) { }
+   body(x) <- parse(text=paste(a * e$vectors[1, 1],
+                     " * exp(t * ", e$values[1], ") + ",
+                     b * e$vectors[1, 2], "* exp(t * ",
+                     e$values[2], ")" ))
+   body(y) <- parse(text=paste(a * e$vectors[2, 1],
+                     " * exp(t * ", e$values[1], ") + ",
+                     b * e$vectors[2, 2], "* exp(t * ",
+                     e$values[2], ")" ))
+   return(list(x=x, y=y));
+ }
```

```
> soln <- solveSystemSoln(dotX, dotY, x0=1, y0=3)
> soln;

$x
function (t)
1.66666666666667 * exp(t * -5) + -0.666666666666667 * exp(t *
    1)
<environment: 0x102738cb0>

$y
function (t)
3.33333333333333 * exp(t * -5) + -0.333333333333333 * exp(t *
    1)
<environment: 0x102738cb0>
```

5. *Bonus question #2 (not required). Calculating eigenvectors.*

   (a) In problem 1 you created and tested a function which calculates the eigenvalues of a 2×2 matrix. Write a companion function which calculates both the eigenvalues *and* the eigenvectors of a 2×2 matrix.

```
> solveSystemSoln <- function(dotX, dotY, x0, y0) {
+   mat <- matrix(c(dotX(1, 0), dotX(0, 1), dotY(1, 0), dotY(0, 1)),
+                 nrow=2, byrow=TRUE)
+   e <- eigen(mat);
+   ab <- solve(t(e$vectors), matrix(c(x0, y0), nrow=2));
+   a <- ab[1];
+   b <- ab[2];
+   y <- x <- function(t) { }
+   body(x) <- parse(text=paste(a * e$vectors[1, 1],
+                     " * exp(t * ", e$values[1], ") + ",
+                     b * e$vectors[2, 1], "* exp(t * ",
+                     e$values[2], ")" ))
+   body(y) <- parse(text=paste(a * e$vectors[1, 2],
+                     " * exp(t * ", e$values[1], ") + ",
+                     b * e$vectors[2, 2], "* exp(t * ",
+                     e$values[2], ")" ))
+   return(list(x=x, y=y));
+ }
```

   (b) Your new function should produce result nearly identical to those returned by the `eigen()` function. Use the provided `testEigenFunction2()` function to validate your new function.

```
> testEigenFunction2 <- function(fun, nTests) {
+   for (ii in 1:nTests) {
+     mat <- matrix(floor(runif(n=4, min=-100, max=100)), nrow=2);
+
+     sSol <- eigen(mat);
+     sSol$vectors[, 1] <- as.complex(normalizeVector(sSol$vectors[, 1]));
+     sSol$vectors[, 2] <- as.complex(normalizeVector(sSol$vectors[, 2]));
+     sSol$values <- as.complex(sSol$values);
+
+     mySol <- fun(mat);
+     mySol$vectors[, 1] <- as.complex(normalizeVector(mySol$vectors[, 1]));
+     mySol$vectors[, 2] <- as.complex(normalizeVector(mySol$vectors[, 2]));
+     mySol$values <- as.complex(mySol$values);
+
+     if (!( isTRUE(all.equal(sSol$values, mySol$values)) ||
+            isTRUE(all.equal(sSol$values, mySol$values[c(2, 1)] )) )) {
+       cat(sprintf('Error: eigenvalue result mismatch on iteration %d.\n',
+                   ii));
+       return(list(matrix=mat, sSolution=sSol, mySolution=mySol));
+     }
+     if (!(isTRUE(all.equal(sSol$values, mySol$values)))) {
+       sSol$vectors <- sSol$vectors[, c(2, 1)];
+     }
+     if (    (!isTRUE(all.equal(sSol$vectors[,1], mySol$vectors[,1]))) &&
+             (!isTRUE(all.equal(-1 * sSol$vectors[,1], mySol$vectors[,1]))) ) {
+       cat(sprintf('Error: eigenvector result mismatch on iteration %d.\n',
+           ii));
+       return(list(matrix=mat, sSolution=sSol, mySolution=mySol));
+     } else if ( (!isTRUE(all.equal(sSol$vectors[,2], mySol$vectors[,2]))) &&
+                 (!isTRUE(all.equal(-1 * sSol$vectors[,2],
+                                    mySol$vectors[,2])))) {
+       cat(sprintf('Error: eigenvector result mismatch on iteration %d.\n',
+                   ii));
+       return(list(matrix=mat, systemSolution=sSol, mySolution=mySol));
+     }
+   }
+   cat(sprintf('Success over %d trials.\n', nTests));
+ }
> ## As an example....
> testEigenFunction2(eigen, 50);                # Test the test function

Success over 50 trials.

> badFunction <- function(mat) {
+   result <- list(values=as.complex(mat[, 1]),
+                  vectors=matrix(as.complex(mat), nrow=2))
+   return(result);
+ }
> testEigenFunction2(badFunction, 50);          # This should fail

Error: eigenvalue result mismatch on iteration 1.
$matrix
     [,1] [,2]
[1,]  -58   88
[2,]  -31  -92

$sSolution
$sSolution$values
[1] -75+49.38623i -75-49.38623i

$sSolution$vectors
                      [,1]                  [,2]
[1,]  0.9114064+0.1338372i  0.9114064-0.1338372i
[2,] -0.2511775+0.4856329i -0.2511775-0.4856329i


$mySolution
$mySolution$values
[1] -58+0i -31+0i

$mySolution$vectors
```

```
               [,1]           [,2]
[1,] -0.7055453+0i  0.5529781+0i
[2,] -0.3771018+0i -0.5781135+0i
```

```
> soln <- solveSystemSoln(dotX, dotY, x0=1, y0=3)
> soln;

$x
function (t)
1.66666666666667 * exp(t * -5) + -0.666666666666667 * exp(t *
    1)
<environment: 0x1156379c8>

$y
function (t)
3.33333333333333 * exp(t * -5) + -0.333333333333333 * exp(t *
    1)
<environment: 0x1156379c8>
```