Introduction to computational programming
Chapter 6 Exercise
Eigenvalues, eigenvectors, and phase planes

David M. Rosenberg
University of Chicago
Committee on Neurobiology

Version control information:
Last changed date: 2009-11-18 19:44:07 -0600 (Wed, 18 Nov 2009)
Last changes revision:262
Version: Revision 262
Last changed by: David M. Rosenberg

November 19, 2009

Overview

In this section you will continue to explore the meaning and use of eigenvalues as they pertain to solving (systems) of differential equations.

Part I

Tutorial

1 Eigenvalues

Although it may seem at first like reinventing the wheel, it is not uncommon for a programmer to decide to *rewrite* or *replace* an existing or built-in function. The first problem on this assignment asks you to do precisely that: to write your own functions to calculate the eigenvalues and eigenvectors of a 2×2 matrix.[1]

Ultimately, the key to writing these functions is to see that the calculation of a 2×2 matrix's eigenvalues amounts to nothing more than solving a carefully formed quadratic equation. Once you have the eigenvalues, calculation of the corresponding eigenvectors should proceed naturally.

[1] *R* already provides a function for this purpose, `eigen()`, which returns the eigenvalues and eigenvectors for a square matrix.

A few pointers and suggestions in writing these functions:

1. Remember, the quadratic equation $ax^2 + bx + c = 0$ has solutions $\frac{b \pm \sqrt{b^2 - 4ac}}{2a}$.
2. Don't forget about the possibility of complex values or intermediates. In *R* , if you anticipate the occurrance of complex numbers anywhere in the function, it is best to coerce *all* values in the function to type `complex()`.[2]

```
> a <- -3;
> sqrt(a);

[1] NaN

> a2 <- -3 + 0i;
> sqrt(a2);

[1] 0+1.732051i

> a3 <- as.complex(a);
> sqrt(a3);

[1] 0+1.732051i
```

3. Check your work! Since there already exists a function in *R* which should work the same as your functions, it should be easy to test whether or not your function works properly.

```
> 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);    ## _my_ solution

Success.  All 50 tests completed successfully.

> exFun2 <- function(mat) {
+   return(abs(eigen(mat)$values));
+ }
> testEigenFunction(exFun2);
```

[2] Use the function `as.complex()` for variables and use the `a + bi` notation for actual values.

```
Error encountered in test #1, further tests aborted.
$testMatrix
     [,1] [,2]
[1,]   59  -47
[2,]  -87  -86

$systemResult
[1] -110.17083+0i   83.17083+0i

$testResult
[1] 110.17083+0i  83.17083+0i
```

2 Phase planes

Another task assigned in this week's homework is to write a function similar to the "phase-plane" applet Dmitry demonstrated in class. In order to facilitate this task, several *auxiliary* or helper functions will be provided. This is similar to problem from chapter 1 where you were provided the `parsePolynomial()` and `deparsePolynomial()` functions. The first function, `normalizeVector()` takes a vector of length 2 of arbitrary euclidean length and "shortens" or "lengthens" it so that its length is euclidean length is 1. [3] This function is useful for the first "part" of phase plane construction. Most phase plane diagrams (including those shown in class and the example below) normalize the length of the field arrows to enhance legibility.

The second function you are provided with, `solveOdeSystem()` numerically calculates the solution to a specified linear system of ODEs and returns a table of x, and y values representing the solution curve. This function may be useful to you in the second part of the "phase-plane" question.

3 Functions defined by optional arguments

The second part of the "phase-plane" exercise involves adding functionality to your function so that it takes an (optional) list of initial values and draws the phase plane plot with solution curves for each set of initial values specified.

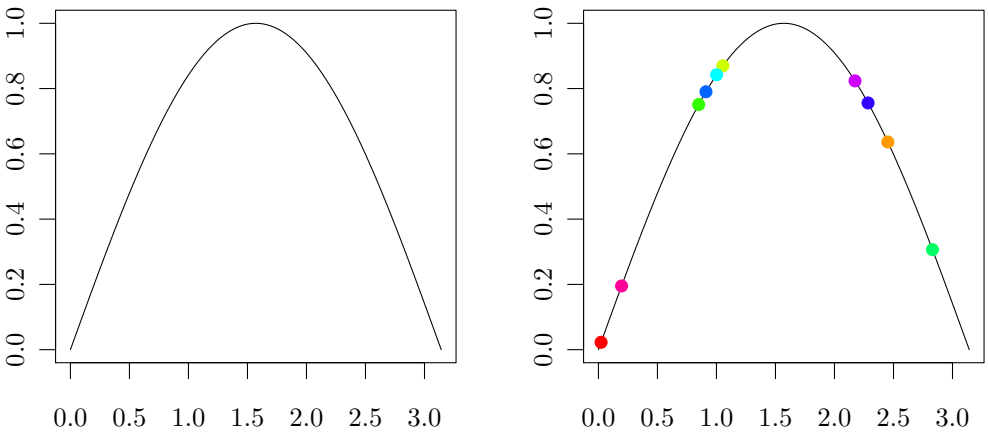
In order to help you see how this might be done, consider the following (trivial) example.

I am going to create a function, *pointCurve*, that functions similarly to the `curve()` function but takes an additional argument, `xList`, that specifies a set of x values for which to add colored dots to the function curve.

```
> pointCurve <- function(expr, from=0, to=1, xList=NULL, ...) {
+   sexpr <- substitute(expr);            ## Don't worry about this
+   curve(expr, from=from, to=to, ...)
+   if(!is.null(xList)) {                 ## This is KEY - Identifying whether
+                                         ##   or not to do the optional
+                                         ##   'thing'
+     y <- numeric(length=length(xList));
+     for (ii in 1:length(xList)) {
+       y[ii] <- eval(call(eval(as.character(sexpr)), xList[ii]));
```

[3] The euclidean length of a a vector $\bar{v} = (a, b)$ is represented by $|\bar{v}|$ and given by the formula $|\bar{v}| = \sqrt{a^2 + b^2}$.

```
+     }
+     points(xList, y, col=rainbow(n=length(xList)), pch=19);
+   }
+ }
> layout(matrix(c(1, 2), nrow=1));
> par(mar=c(2,2,2,2));
> pointCurve(sin, 0, pi);                    ## One behavior without the arg.
> pointCurve(sin, 0, pi, xList=runif(n=10, ## And one with it.
+                          min=0, max=pi));
```



Part II

Exercises

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.

   (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=''));
+ }
```

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

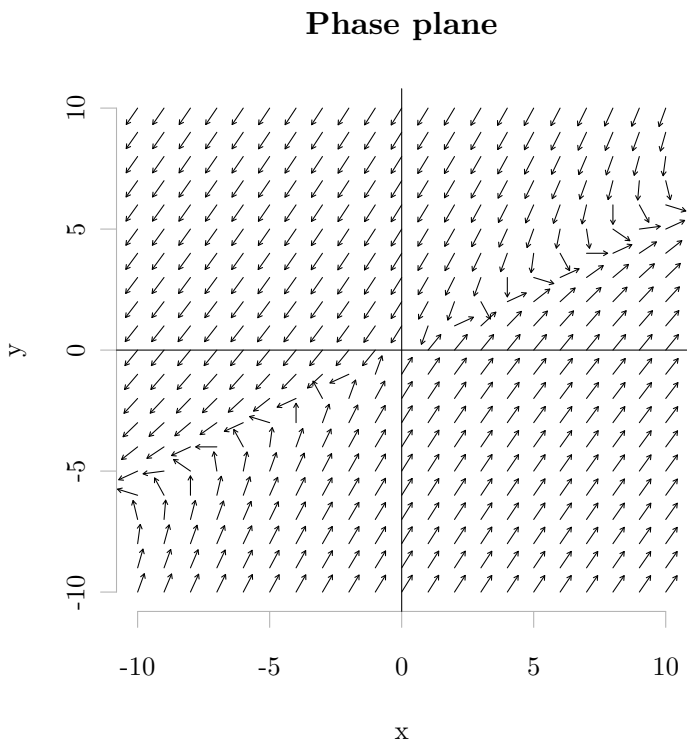
iv. $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

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)
+ }
```

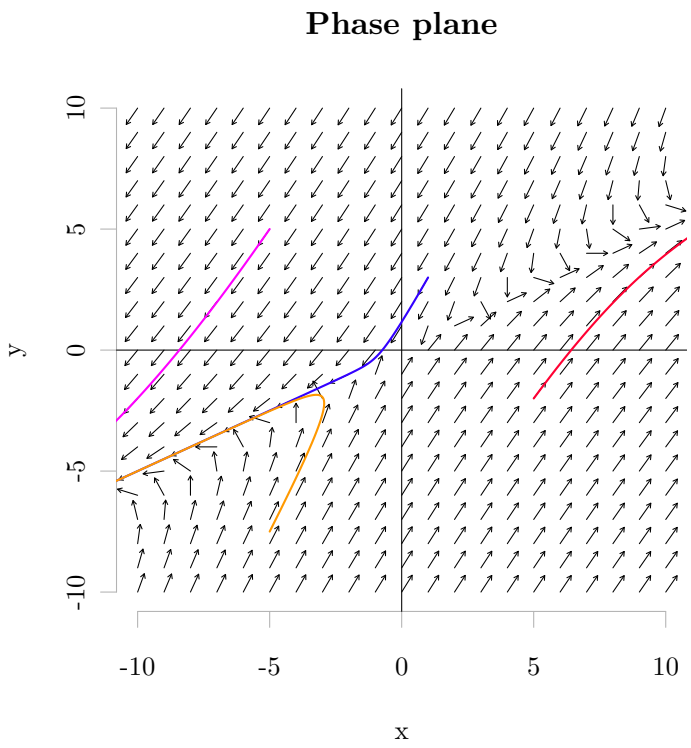
```
> mat <- matrix(c(3, -4, 4, -7), nrow=2, byrow=TRUE)
> drawPhasePlane(mat, c(-10, 10), c(-10, 10))
```



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$vectors / min(e$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));
+ }
```

```
> initialValues <- list(c(1, 3), c(-5, 5), c(5, -2), c(-5, -7.5))
> drawPhasePlane2(mat, c(-10, 10), c(-10, 10), initialValues);
```



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 $\mathbf{A}$ in problem 1. Plot two different solution trajectories for each system.

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.

(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.

(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.

4. *Bonus question (not required). 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: 0x100bb4540>

$y
function (t)
{
    return(10/3 * exp(-5 * t) + -1/3 * exp(t))
}
<environment: 0x100bb4540>
```

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.

   (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,]   38    5
[2,]  -86   -4

$sSolution
$sSolution$values
[1] 20.31662+0i 13.68338+0i

$sSolution$vectors
              [,1]           [,2]
[1,]  0.2176674+0i -0.1611256+0i
[2,] -0.7698188+0i  0.7836061+0i


$mySolution
$mySolution$values
[1]  38+0i -86+0i

$mySolution$vectors
              [,1]          [,2]
[1,]  0.3233311+0i  0.624695+0i
[2,] -0.7317493+0i -0.499756+0i
```