Using R to analyze continuous models (lab 4)

Steve Walker

October 6, 2014

Version: 2014-10-06 12:07:38

1 Solving linear models

We start simple, with the following bivariate linear model,

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{A}\boldsymbol{x} \tag{1}$$

where the coefficient matrix, $\boldsymbol{A}$, is,

```
set.seed(2)
A <- matrix(rnorm(4), 2, 2, byrow = TRUE)
A

##         [,1]    [,2]
## [1,] -0.8969  0.1848
## [2,]  1.5878 -1.1304
```

We see that this model has a stable fixed point at the origin because all eigenvalues are less than zero,

```
eigen(A)$values

## [1] -1.5678 -0.4594
```

However, how is this equilibrium approached? Let us find a time-dependent solution. We know that the solution is

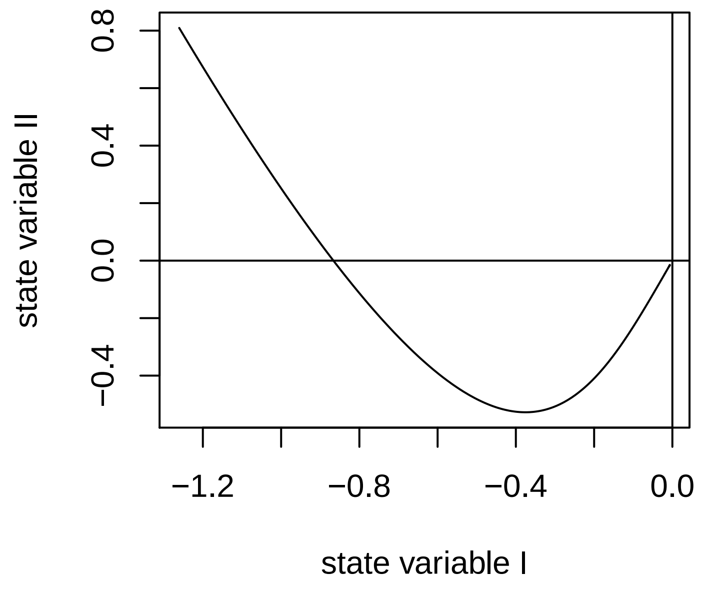
$$\boldsymbol{x}(t) = \phi_1 e^{t\lambda_1}\boldsymbol{v}_1 + \phi_2 e^{t\lambda_2}\boldsymbol{v}_2 \tag{2}$$

where the λ's and $\boldsymbol{v}$'s are eigenvalues and eigenvectors, and the ϕ's are components of the vector,

$$\boldsymbol{\phi} = \boldsymbol{V}^{-1}\boldsymbol{x}(0) \tag{3}$$

where $\boldsymbol{V}$ is the matrix with eigenvectors for columns. Therefore, we can find a solution by,

```
set.seed(2)
x0 <- runif(2, -2, 2)
eigA <- eigen(A)
V <- eigA$vectors
lambda <- eigA$values
phi <- solve(V) %*% x0
tt <- seq(0, 10, length = 1000)
x <- phi[1] * outer(exp(tt*lambda[1]), V[,1]) +
     phi[2] * outer(exp(tt*lambda[2]), V[,2])
plot(x, type = "l",
     xlab = "state variable I",
     ylab = "state variable II")
abline(h = 0, v = 0)
```



Exercise 1.1: Choose different initial conditions, and use this code to explore the resulting approach to equilibrium.

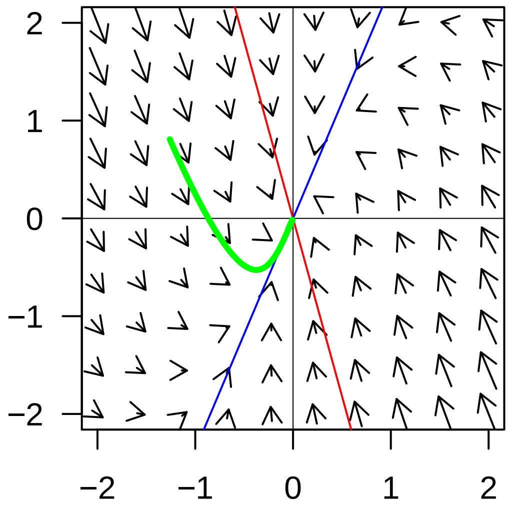
2 Vector field plots

Another way to get more insight into the model that is not dependent on specific initial conditions is to construct a vector field plot, which represents the magnitude and direction of the time derivative at various points in the state space. To construct a vector field plot we use the `plotrix` package,

```
install.package("plotrix")
library(plotrix)
```

```
library(plotrix)
xx <- seq(-2, 2, length = 10)
X <- as.matrix(expand.grid(xx, xx))
Ax <- X %*% t(A)
```

```
par(mar = c(3, 3, 1, 1))
plot(c(-2, 2), c(-2, 2), type = "n",
     las = 1)
abline(h = 0, v = 0, lwd = 0.5)
vectorField(Ax[,1], Ax[,2],
            X[,1], X[,2])
eigA <- eigen(A)
eVec <- eigA$vectors
eVecInv <- solve(eVec)
eVal <- eigA$values
eigSlopes <- eVec[2,]/eVec[1,]
abline(a = 0,
       b = eigSlopes[1],
       col = "red")
abline(a = 0,
       b = eigSlopes[2],
       col = "blue")
lines(x, col = "green", lwd = 3)
```



We have also plotted the two eigenvectors (in red and blue) and the trajectory (in green). Note that the vectors along the eigenvectors point to the fixed point, indicating stability.

Exercise 2.1: Choose different values for `A` and explore the range of behaviour that is possible with bivariate linear models. Choose values untill you have found (1) models with cycles, (2) models with only one linearly independent eigenvector, and (3) models with one positive and one negative eigenvalue.

3 Numerical solutions to differential equations

R can also be used to solve differential equations. You are required to be able to use R for this purpose. We will be using the `deSolve` add-on package for R, which can be obtained by,

```
install.package("deSolve")
library(deSolve)
```

To illustrate `deSolve` we will use the logistic model,

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right) \tag{4}$$

The first task is to construct a function that computes the gradient (i.e. the time-derivative of the state variable),

```
gradfun <- function(t, N, params) {
    with(c(as.list(N), as.list(params)),
         list(N = r * N * (1 - N/K)))
}
```

This function takes three arguments, `t`, which accepts a time point at which to evaluate the differential equation, `N`, which accepts the state variable (e.g. N), and `params`, which accepts the model parameters (e.g. r and K). The arguments must be in this order, but the names do not matter. Note also that we have used the `with` function, which at this point might seem a bit magical, but allows us to refer to the state variables and parameters by name. The output of this function is an R list with the gradient as the first argument. The `NULL` element must be present as a placeholder.

To solve this function, we use the `lsoda` function,

```
desol <- lsoda(y = c(N = 0.1),
               times = seq(0, 10, by = 0.1),
               func = gradfun,
               parms = c(r = 1, K = 5))
```

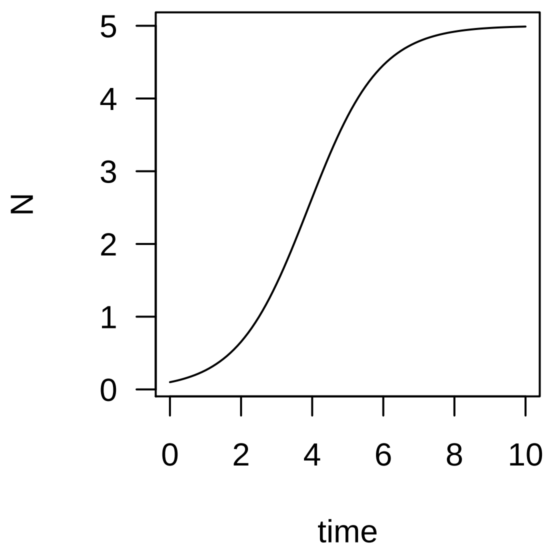
which takes the initial values, `y`, the times at which to evaluate, `times`, the gradient function, `func`, and numerical values for the parameters, `parms`. Let's look at the solutions for the first few time points,

```
desol <- as.data.frame(desol)
head(desol)

##   time      N
## 1  0.0 0.1000
## 2  0.1 0.1103
## 3  0.2 0.1216
## 4  0.3 0.1340
## 5  0.4 0.1477
## 6  0.5 0.1628
```

We may plot the results using `with` as well,

```
par(mar = c(4, 4, 1, 1))
with(desol, plot(time, N,
                 las = 1,
                 type = "l"))
```



The `deSolve` package may also be used to solve multivariate differential equations. Please consult the `?lsoda` help file for examples.

Exercise 3.1 *: Use the techniques you have learnt to complete Project 5.9 on pp.320-1 in MS. For part b, contruct a vector field plot instead of a phase plane analysis.