phaseR: An R Package for Phase Plane Analysis of Autonomous ODE Systems

by Michael J. Grayling

Abstract When modelling physical systems, analysts will frequently be confronted by differential equations which cannot be solved analytically. In this instance, numerical integration will usually be the only way forward. However, for autonomous systems of ordinary differential equations (ODEs) in one or two dimensions, it is possible to employ an instructive qualitative analysis foregoing this requirement, using so-called phase plane methods. Moreover, this qualitative analysis can even prove to be highly useful for systems that can be solved analytically, or will be solved numerically anyway. The package **phaseR** allows the user to perform such phase plane analyses: determining the stability of any equilibrium points easily, and producing informative plots.

Introduction

Repeatedly, when a system of differential equations is written down, it cannot be solved analytically. This is particularly true in the non-linear case, which unfortunately habitually arises when modelling physical systems. As such, it is common that numerical integration is the only way for a modeller to analyse the properties of their system. Consequently, many software packages exist today to assist in this step. In R, for example, the package **deSolve** (Soetaert et al., 2010) deals with many classes of differential equation. It allows users to solve first-order stiff and non-stiff initial value problem ODEs, as well as stiff and non-stiff delay differential equations (DDEs), and differential algebraic equations (DAEs) up to index 3. Moreover, it can tackle partial differential equations (PDEs) with the assistance **ReacTran** (Soetaert and Meysman, 2012). For such PDE problems users also have available the package **rootSolve** (Soetaert et al., 2009), which is additionally particularly suited to analysing the equilibrium of differential equations. Finally, **bvpSolve** (Soetaert et al., 2013) can tackle boundary value problems of systems of ODEs, whilst **sde** (Iacus, 2009) is available for stochastic differential equations (SDEs). However, for autonomous ODE systems in either one or two dimensions, phase plane methods, as developed by mathematicians such as Henri Poincaré in the 19th Century (Farlow, 2006), allow for a qualitative analysis of the system's properties without the need for numerical integration. Specifically, it is usually possible to determine the long-term behaviour of the system for any initial condition, via a highly graphical procedure coupled with algorithmic mathematical analysis.

As a result of these considerations, phase plane analysis is an important device in any modeller's toolbox. It stands as a key technique not only for mathematicians, but biologists, physicists and engineers amongst others. See for example Jordan and Smith (2007), Barnes and Fulford (2009), Choudhury (2005), and Mooney (1999) for descriptions of uses in various fields. Since the method is extremely algorithmic in nature, software tools would clearly provide a powerful system by which to execute the techniques. Despite this, few programmes are available for implementing phase plane methods. Presently, a programme entitled pplane that employs a simple GUI system is available for Matlab (MATLAB, 2014; Polking, 2009). In addition, several java applets are available across the internet for executing phase plane methods. But, they vary widely in quality and often struggle when confronted with complex systems, whilst they also provide no simple way to export produced plots. Now, with R ever growing in popularity as an open-source general purpose scientific computing lenguage, it is logical that available code to implement phase plane methods would serve as a valuable resource. The first attempts in this direction were implemented by Kaplan and Flath (2004) at Macalester College who created several short programmes to analyse two dimensional ODEs and provided several example systems. **phaseR** extends their initial ideas in order to create a complete package; one providing phase plane novices with the opportunity for independent learning, instructors with a useful package for practical sessions, and experienced modellers with an easy means by which to produce quality plots.

Overall, **phaseR** maintains the original structure of Kaplan and Flath's code; with key functions available for each of the steps of a full phase plane analysis. Specifically, it can assist in identifying and classifying equilibrium points, allows direction fields as well as nullclines to be plotted, and allows trajectories to be added to plots for user specified initial conditions. **phaseR**, however, extends the functionality of the code to allow analysis of both one and two dimensional systems, whilst also allowing for far greater control over the final produced plot. Moreover, **phaseR** comes complete with an extensive guide detailing the techniques of phase plane analysis, also containing numerous examples and exercises. Finally, **phaseR** makes use of available R packages for finding numerical solutions to ODEs in order to ensure maximum stability of the required integration step. I will now proceed by briefly reviewing the techniques of phase plane analysis, before discussing the usage of the

package through several examples.

One dimensional autonomous ODE systems

Any one dimensional autonomous ODE, of a function $y = y(t) = y_t$, can be written in the following form

$$\frac{dy_t}{dt} = f(y_t).$$

Thus, autonomous systems are characterised by not explicitly depending on the independent variable t. For such systems, phase plane analysis begins by plotting at a range of values for both the dependent and the independent variable, a small arrow indicating the rate of change of y_t as provided by the ODE. This plot, commonly referred to as the direction field, is useful because solutions to the ODE must pass through the arrows in a tangential manner. Therefore, once produced, it provides an easy construction from which to draw trajectories for any initial condition without the need to solve the ODE.

Following this, so-called equilibrium points are determined. Defined as the points y_* where $f(y_*) = 0$, the reason for their name is easy to see: beginning at such a point, because of the lack of explicit dependence upon t in f, means the solution will remain there for all values of t. Moreover, these points are then classified as either stable or unstable, depending upon whether solutions converge towards, or diverge away, from them. Stability may here be determined from a 'phase portrait plot' of $f(y_t)$ against y_t ; on which arrows are placed indicating the direction of change of y_t with t. Arrows pointing towards each other on either side of an equilibrium point denote stability, whilst arrows pointing away from each other indicate the presence of an unstable point. Alternatively, a Taylor Series argument can be utilised to define a simple criterion. The argument proceeds by supposing the system begins a small distance δ_0 away from the fixed point y_*, i. e. $y_0 = y_* + \delta_0$. Then, writing our general expression for y_t as $y_t = y_* + \delta_t$, we use the Taylor Series expansion of f to form a differential equation for how δ_t changes with t

$$f(y_* + \delta_t) = f(y_*) + \delta_t \frac{\partial f}{\partial y_t}(y_*) + o(\delta_t),$$

where we have assumed higher order terms are negligible. Recalling $f(y_*) = 0$, our ODE becomes

$$\begin{aligned} \frac{d}{dt} f(y_* + \delta_t) &= \delta_t \frac{\partial f}{\partial y_t}(y_*), \\ \Rightarrow \frac{d\delta_t}{dt} &= \delta_t \frac{\partial f}{\partial y_t}(y_*) = k\delta_t. \end{aligned}$$

This autonomous ODE for δ_t can be solved easily to give $\delta_t = \delta_0 e^{kt}$. This analysis is useful to us since stability can be determined based upon whether δ_t grows, or decays, as t increases, i. e. using the simple criterion

$$k = \frac{\partial f}{\partial y_t}(y_*) = \begin{cases} > 0 & \text{Stable,} \\ < 0 & \text{Unstable} \end{cases}.$$

Two dimensional autonomous ODE systems

In the two dimensional case, for functions $x = x(t) = x_t$ and $y = y(t) = y_t$, any autonomous ODE system can be written as

$$\frac{dx_t}{dt} = f(x_t, y_t), \quad \frac{dy_t}{dt} = g(x_t, y_t).$$

Thus as before, these systems are characterised by a lack of explicit dependence upon the independent variable t in the functional forms of f and g.

The direction field, more commonly referred to here as the velocity field, is again produced to provide a powerful way by which to plot trajectories for any initial condition. However, here this is done so in the x_t-y_t plane, rather than that containing the independent variable. This visualisation proves to be the best way by which to examine an autonomous two dimensional system.

Commonly, to assist the plotting of the velocity field, nullclines are first computed. These are defined as the location where $f(x_t, y_t) = 0$ or $g(x_t, y_t) = 0$. Thus they define the curves across which either x_t or y_t does not change in t. Any velocity arrows produced will therefore either be perfectly horizontal or vertical. Since a velocity field must vary continuously across a particular nullcline, by the uniqueness of solutions to ODEs, nullclines can be used to check that no arrows have been plotted

incorrectly.

As one would expect, equilibrium points retain their great importance in two dimensions. Here, their definition generalises as the locations (x_*, y_*) where $f(x_*, y_*) = g(x_*, y_*) = 0$. Consequently, the intersection of the x_t and y_t nullclines can be used to determine their location. Stability here is defined in an analogous manner to the one dimensional case, whilst a Taylor Series argument again allows the formulation of a simple rule for determining said stability. As before, it supposes that we have an equilibrium point, now given by (x_*, y_*), and that the system initially lies slightly away from this point at $(x_* + \delta_0, y_* + \epsilon_0)$, and in general we have $(x_t, y_t) = (x_* + \delta_t, y_* + \epsilon_t)$. Then using the Taylor expansion for f, the differential equation for x_t becomes

$$\begin{aligned}\frac{d\delta_t}{dt} &= f(x_* + \delta_t, y_* + \epsilon_t), \\ &= f(x_*, y_*) + \delta_t \frac{\partial f}{\partial x_t}(x_*, y_*) + \epsilon_t \frac{\partial f}{\partial y_t}(x_*, y_*) + o(\delta_t) + o(\epsilon_t), \\ &= \delta_t \frac{\partial f}{\partial x_t}(x_*, y_*) + \epsilon_t \frac{\partial f}{\partial y_t}(x_*, y_*) + o(\delta_t) + o(\epsilon_t).\end{aligned}$$

Similarly, the differential equation for y_t becomes

$$\frac{d\epsilon_t}{dt} = \delta_t \frac{\partial g}{\partial x_t}(x_*, y_*) + \epsilon_t \frac{\partial g}{\partial y_t}(x_*, y_*) + o(\delta_t) + o(\epsilon_t).$$

In both equations we have again assumed terms of second order and higher are negligible. Now if we write this system in matrix form we acquire

$$\frac{d\boldsymbol{\delta}_t}{dt} = \left.\begin{pmatrix} \frac{\partial f}{\partial x_t} & \frac{\partial f}{\partial y_t} \\ \frac{\partial g}{\partial x_t} & \frac{\partial g}{\partial y_t} \end{pmatrix}\right|_{(x_*, y_*)} \boldsymbol{\delta}_t = J\boldsymbol{\delta}_t.$$

Here J is called the Jacobian of the system, and $\boldsymbol{\delta} = (\delta, \epsilon)^\top$. It is consideration of the characteristic equation of J, and the values of its eigenvalues it provides, that allows equilibrium points to be classified, not only as either stable or unstable, but also in to sub-categories. Full details of this can be found in the guide that accompanies **phaseR**, available in the documentation folder of its installation. However, classification ultimately, and simply, depends upon the values of the determinant and trace of J.

Thus, in both the one and two dimensional cases, phase plane methods allow trajectories to be easily plotted, and the long-term behaviour of a system identified by classifying the equilibria. We shall next see how **phaseR** can be used to perform such analyses using several simple commands.

Key functions

phaseR employs six key functions for the employment of phase plane analysis.

- `flowField`: Plots the direction or velocity field of one or two dimensional autonomous ODE systems.
- `nullclines`: Plots the nullclines of two dimensional autonomous ODE systems. Alternatively, it can be used to plot horizontal lines at equilibrium points for one dimensional autonomous ODE systems.
- `numericalSolution`: For two dimensional systems, this function numerically solves the ODEs for a given initial condition via **deSolve**, and then plots the dependent variables against the independent variable. Thus, it behaves as a wrapper for the user to **deSolve**, allowing for easier implementation and subsequent plotting.
- `phasePortrait`: For one dimensional autonomous ODEs, it plots the phase portrait i. e. the derivative against the dependent variable. In addition, it adds arrows to the dependent variable axis from which the stability of any equilibrium points can be determined.
- `stability`: Classifies equilibrium points, using the criteria that result from the aforementioned Taylor Series arguments.
- `trajectory`: Plots trajectories in the x_t-y_t plane by performing numerical integration of the chosen ODE system, again via **deSolve**. Initial conditions can be specified as an argument, or by clicking with the cursor on locations in a pre-existing plot.

As an example, standard phase plane analyses for a two dimensional ODE system would proceed by using `flowField`, `nullclines` and then `trajectory` to create a summarising plot, and finally `stability` to classify the equilibria.

Example 1: logistic growth model

To describe how **phaseR** can be used to analyse a one dimensional autonomous system of interest, we will consider the logistic growth model. Originally proposed by Verhulst (1838), this ODE is frequently used in Biology to model the growth of a population under density dependence. It is given in its most general case by

$$\frac{dy_t}{dt} = \beta y_t \left(1 - \frac{y_t}{K}\right).$$

To analyse any system using **phaseR**, a function must first be created to return the value of the derivative at any point (t, y), in a style compatible with **deSolve**. For this model such a function is already present in the package as `logistic`, which comprises the following code

```
logistic <- function(t, y, parameters){
    beta <- parameters[1]
    K    <- parameters[2]
    dy   <- beta*y*(1 - y/K)
    list(dy)
}
```

The general format of this derivative function should be a common one: given `t`, `y` and a vector `parameters`, it should simply return a list with first element the value of the derivative.

With this, we can then proceed with our desired analysis. From here we shall consider the particular case $beta = 1$ and $K = 2$. The following code creates Figure 1; adding the direction field, several trajectories and horizontal lines at any equilibrium points.

```
> logistic.flowField  <-
+     flowField(logistic, x.lim = c(0, 5), y.lim = c(-1, 3),
+               parameters = c(1, 2), points = 21,
+               system = "one.dim", add = FALSE, xlab = "t")
> grid()
> logistic.nullclines <-
+     nullclines(logistic, x.lim = c(0, 5), y.lim = c(-1, 3),
+                parameters = c(1, 2), system = "one.dim")
> logistic.trajectory <-
+     trajectory(logistic, y0 = c(-0.5, 0.5, 1.5, 2.5), t.end = 5,
+                parameters = c(1, 2), system = "one.dim", colour = rep("black", 4))
```

Here we have tailored our result by setting a suitable range for the dependent and independent variables, whilst by default suitable axes labels have been added to the plot. These are just several input variables that can be changed to suit the needs of your particular system.

Now, we can observe from Figure 1 that two equilibrium points have been identified; appearing to be $y_* = 0$ and $y_* = 2$. We can confirm their location analytically however by setting the right hand side of the ODE to zero

$$\begin{aligned} y_* \left(1 - \frac{y_*}{2}\right) &= 0, \\ \Rightarrow y_* &= \{0, 2\}. \end{aligned}$$

It appears from the trajectories in Figure 1 that the point $y_* = 2$ is stable, and $y_* = 0$ unstable. We can confirm this by plotting the phase portrait with the following code, producing Figure 2

```
> logistic.phasePortrait <- phasePortrait(logistic, y.lim = c(-0.5, 2.5),
+                                         parameters = c(1, 2), points = 10)
> grid()
```

Alternatively, if we use our Taylor Series method we can draw this same conclusion

$$\frac{d}{dy}\left(\frac{dy}{dt}\right)\bigg|_{y=y_*} = 1 - y_* = \begin{cases} +1 & y_* = 0, \\ -1 & y_* = 2 \end{cases}.$$

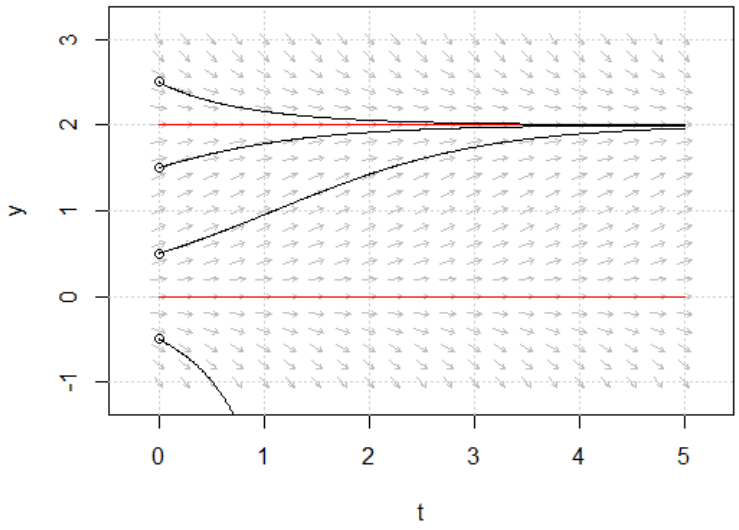

Figure 1: The direction field and several trajectories for the logistic growth model with $\beta = 1$ and $K = 2$. It can be seen that two equilibrium points have been located.

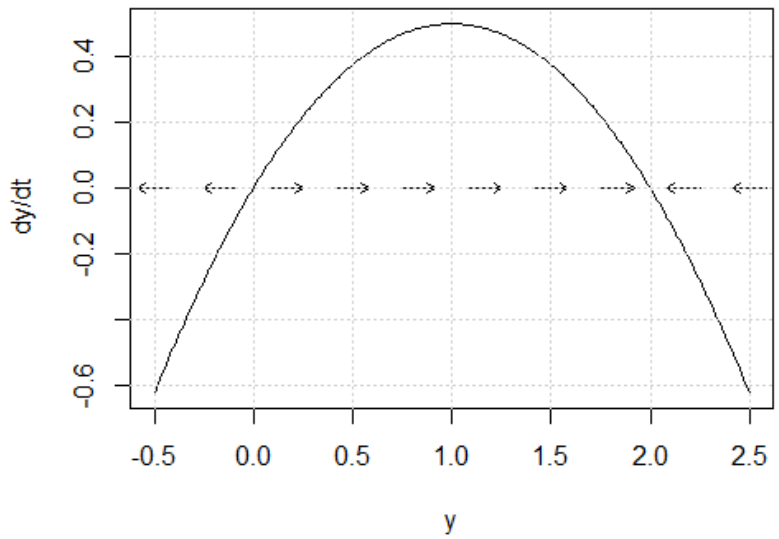

Figure 2: The phase portrait for the logistic growth model with $\beta = 1$ and $K = 2$. The line $y = 0$ has been identified as unstable, and the line $y = 2$ as stable.

Thus we have drawn the same conclusion; $y_* = 0$ is indeed unstable, and $y_* = 2$ stable. The following code can perform this analysis for us using `stability`

```
> logistic.stability.1 <-
+     stability(logistic, y.star = 0, parameters = c(1, 2), system = "one.dim")

Discriminant: 1   Classification: Unstable

> logistic.stability.2 <-
+     stability(logistic, y.star = 2, parameters = c(1, 2), system = "one.dim")

Discriminant: -1   Classification: Stable
```

This completes our phase plane analysis of the logistic growth model. We have successfully identified that for this model, if $y_0 > 0$, which it must be for the model to make physical sense, the system will eventually approach the line $y_t = 2$, regardless of the exact initial condition. **phaseR** clearly provided a capacity by which to either check, or perform, the analyses that would take time by hand using several short lines of code.

Example 2: Lotka-Volterra model

As an example of the capabilities of **phaseR** for a two dimensional system, we will study a particular case of the Lotka-Volterra model (Lotka, 1925; Volterra, 1926) for interacting predator and prey. This model can be written in its general form as

$$\frac{dx_t}{dt} = \lambda x_t - \epsilon x_t y_t,$$
$$\frac{dy_t}{dt} = \eta x_t y_t - \delta y_t.$$

Again, we must first specify a derivative function. As before, this is already provided in the package as `lotkaVolterra`, which consists of the following code

```
lotkaVolterra <- function(t, y, parameters) {
    x <- y[1]
    y <- y[2]
    lambda  <- parameters[1]
    epsilon <- parameters[2]
    eta     <- parameters[3]
    delta   <- parameters[4]
    dy    <- numeric(2)
    dy[1] <- lambda*x - epsilon*x*y
    dy[2] <- eta*x*y - delta*y
    list(dy)
}
```

deSolve again requires us here to create a function taking values for `t`, `y` and `parameters`, but here returning a list whose first element is a vector of the two derivatives, and also accepting `y` as a vector argument.

From here we shall focus on the particular case $\lambda = 2$, $\epsilon = 1$, $\eta = 3$ and $\delta = 2$. We begin by determining the nullclines; by setting the two derivatives to zero in turn, starting with x_t

$$2x_t - x_t y_t = 0 \Rightarrow x_t(2 - y_t) = 0 \Rightarrow x_t = 0,\ y_t = 2,$$

and then y_t

$$3x_t y_t - 2y_t = 0 \Rightarrow y_t(3x_t - 2) = 0 \Rightarrow y_t = 0,\ x_t = 2/3.$$

Requiring that both derivatives are zero identifies the equilibrium points of the model; clearly given here by $(0, 0)$ and $(2/3, 2)$. To classify them we turn to the Jacobian

$$J = \begin{pmatrix} 2 - y_* & -x_* \\ 3y_* & 3x_* - 2 \end{pmatrix}.$$

Thus for $(0, 0)$; $\det(J) = -4$, which classifies the point as a saddle. Additionally, for $(2/3, 2)$ we have that $\det(J) = 4$, and $\mathrm{tr}(J) = 0$, which classifies the point as a centre. Readers are again pointed to the package guide for further details of classification using the Jacobian.

We can repeat all of the above analyses easily using **phaseR**; plotting the nullclines along with the velocity field and several trajectories with the following code that produces Figure 3

```
> lotkaVolterra.flowField <-
+     flowField(lotkaVolterra, x.lim = c(0, 5), y.lim = c(0, 10),
+               parameters = c(2, 1, 3, 2), points = 19, add = FALSE)
> grid()
> lotkaVolterra.nullclines <-
+     nullclines(lotkaVolterra, x.lim = c(-1, 5), y.lim = c(-1, 10),
+                parameters = c(2, 1, 3, 2), points = 500)
> y0 <- matrix(c(1, 2, 2, 2, 3, 4), ncol = 2, nrow = 3, byrow = TRUE)
> lotkaVolterra.trajectory <-
+     trajectory(lotkaVolterra, y0 = y0, t.end = 10,
+                parameters = c(2, 1, 3, 2), colour = rep("black", 3))
```

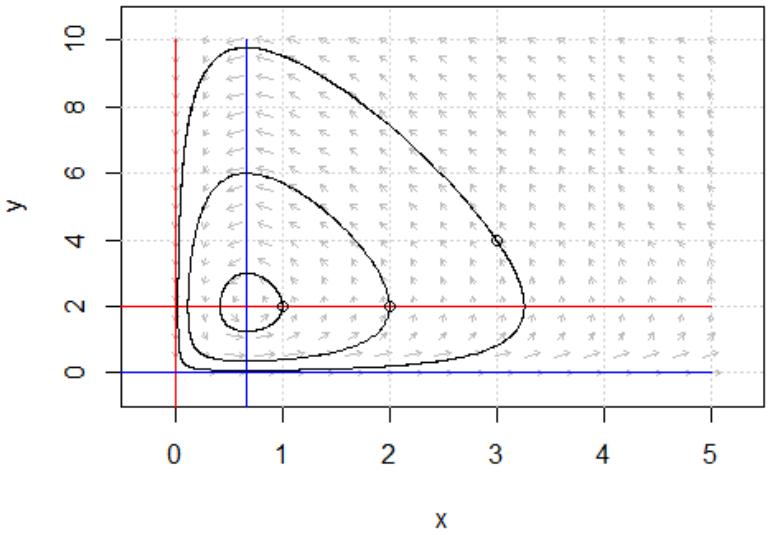

Figure 3: The velocity field, nullclines and several trajectories for the Lotka-Volterra model with $\lambda = 2$, $\epsilon = 1$, $\eta = 3$ and $\delta = 2$. Trajectories can be seen to traverse around the point $(2/3, 2)$.

Finally, we can verify that our stability analysis was indeed correct

```
> lotkaVolterra.stability.1 <-
+     stability(lotkaVolterra, y.star = c(0, 0), parameters = c(2, 1, 3, 2))

T: 0   Delta: -4   Discriminant: 16   Classification: Saddle

> lotkaVolterra.stability.2 <-
+     stability(lotkaVolterra, y.star = c(2/3, 2), parameters = c(2, 1, 3, 2))

T: 0   Delta: 4   Discriminant: -16   Classification: Centre
```

Biologically, we have seen that no matter what the exact initial starting values are, according to this model the numbers of predator and prey will oscillate around the centre $(2/3, 2)$. Alternatively, this oscillation can be seen by plotting the dependent variables against the independent using the following code, producing Figure 4

```
> lotkaVolterra.numericalSolution <-
+     numericalSolution(lotkaVolterra, y0 = c(3, 4), t.end = 10, type = "one",
+                       parameters = c(2, 1, 3, 2), colour = c("green", "orange"),
+                       ylab = "x, y", ylim = c(0, 10))
> legend("bottomright", col = c("green", "orange"), legend = c("x", "y"), lty = 1)
```

This concludes our analysis of this two dimensional system. Again **phaseR** allowed for the phase plane analysis to be performed quickly and easily, bypassing the time consuming process of drawing by hand.

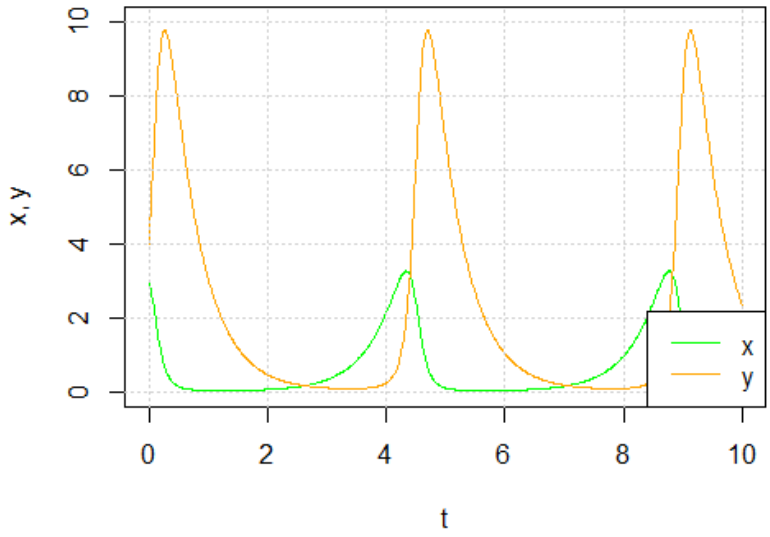

Figure 4: The numerical solution of the Lotka-Volterra mode with $\lambda = 2$, $\epsilon = 1$, $\eta = 3$ and $\delta = 2$ and $(x_0, y_0) = (3, 4)$. The number of each species can be seen to oscillate.

Concluding remarks and future developments

Rarely can a system of differential equations be solved in closed form. Consequently, numerical solutions are often required and therefore much theory exists around integration techniques for each class of differential equations. In R many of these methods have been implemented in packages such as **deSolve**, **bvpSolve**, **sde**, **ReacTran** and **rootSolve**. Between them, these packages allow modellers to deal with many classes of ODEs, DDEs, DAEs, SDEs and PDEs. However, for a one or two dimensional system of autonomous ODEs, an alternative approach to direct numerical solution is available. This so-called phase plane analysis allows much important inference to be made about the system using relatively simplistic graphical and mathematical techniques. The algorithmic nature of the phase plane approach lends itself well to software tools, and **phaseR** provides the first such complete package in R for its implementation. I have demonstrated how **phaseR** allows users to easily perform this qualitative analysis with little initial system set-up, and few commands for execution, required. Sometimes, phase plane analysis is criticised on two grounds; that it is limited to systems of two or less dimension (Mutambara, 1999), and that it is highly labour intensive. Whilst the former still stands, and thus only problems that can be at least well approximated by a second order system may be analysed using the methods discussed above, the latter issue is at least relieved further by the creation of **phaseR**. Whilst to provide greater detail on the dimensional limitation future development of the package will seek to incorporate further example systems from nature, including approximation techniques for higher order problems. In addition, package development will see the one dimensional tools extended to non-autonomous ODEs. Therefore, given all these considerations, in conjunction with the extensive accompanying guide describing in increased detail both the above techniques and providing many worked examples and exercises, **phaseR** should hopefully serve as a useful package for both independent and group led learning, as well as for those simply seeking to produce high-quality figures.

Bibliography

B. Barnes and G. R. Fulford. *Mathematical Modelling with Case Studies: A Differential Equations Approach Using Maple and MATLAB*, pages 159–258. CRC Press, Florida, United States, second edition, 2009. ISBN 978-1420083484. [p43]

D. R. Choudhury. *Modern Control Engineering*, page 755. Prentice-Hall, New Delhi, India, 2005. ISBN 978-81-203-2196-0. [p43]

S. J. Farlow. *An Introduction to Differential Equations and Their Applications*, page 452. Dover Publications, 2006. ISBN 978-0486445953. [p43]

S. M. Iacus. *sde: Simulation and Inference for Stochastic Differential Equations*, 2009. URL http://CRAN.R-project.org/package=sde. R package version 2.0.10. [p43]

D. W. Jordan and P. Smith. *Nonlinear Ordinary Differential Equations: An Introduction for Scientists and Engineers*, pages 49–88. Oxford University Press, New York, United States, fourth edition, 2007. ISBN 978-0-19-920824. [p43]

D. Kaplan and D. Flath. Visualizing the Phase Plane using R, 2004. URL http://www.macalester.edu/~kaplan/math135/pplane.pdf. [p43]

A. J. Lotka. *Elements of Physical Biology*. Williams and Wilkins, Baltimore, United States, 1925. [p48]

MATLAB. *2014a*. The MathWorks Inc., Natick, Massachusetts, United States, 2014. [p43]

D. Mooney. *A Course in Mathematical Modeling*, page 283. The Mathematical Association of America, 1999. ISBN 978-0-88385-7120. [p43]

A. G. O. Mutambara. *Design and Analysis of Control Systems*, page 722. CRC Press, 1999. ISBN 978-0849318986. [p50]

J. C. Polking. pplane8, 2009. URL http://math.rice.edu/~dfield/. [p43]

K. Soetaert and F. Meysman. Reactive transport in aquatic ecosystems: Rapid model prototyping in the open source software R. *Environmental Modelling & Software*, 32:49–60, 2012. [p43]

K. Soetaert, T. Petzoldt, and R. W. Setzer. *A Practical Guide to Ecological Modelling. Using R as a Simulation Platform*. Springer, 2009. ISBN 978-1402086236. [p43]

K. Soetaert, T. Petzoldt, and R. W. Setzer. Solving Differential Equations in R: Package deSolve. *Journal of Statistical Software*, 33(9):1–25, 2010. [p43]

K. Soetaert, J. Cash, and F. Mazzia. *bvpSolve: Solvers for boundary value problems of ordinary differential equations*, 2013. URL http://CRAN.R-project.org/package=bvpSolve. R package version 1.2.4. [p43]

P.-F. Verhulst. Notice sur la loi que la population poursuit dans son accroissement. *Correspondance mathamatique et physique*, 10:113–121, 1838. [p46]

V. Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Mem Acad Lincei Roma*, 2:31–113, 1926. [p48]

Michael J. Grayling
MRC Biostatistics Unit
Cambridge
CB2 0SR
United Kingdom
mjg211@cam.ac.uk