



Revista Digital: Matemática, Educación e Internet

ISSN: 1659-0643

revistadigitalmatematica@itcr.ac.cr

Instituto Tecnológico de Costa Rica

Costa Rica

Mosquera López, Saulo
Los paquetes deSolve y phaseR para la resolución
numérica de Ecuaciones Diferenciales Ordinarias con R
Revista Digital: Matemática, Educación e Internet, vol. 24, núm. 2, 2024, Marzo-Agosto, pp. 1-33
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica

DOI: <https://doi.org/10.18845/rdmei.v24i2.7000>

Disponible en: <https://www.redalyc.org/articulo.oa?id=607975560007>

- Cómo citar el artículo
- Número completo
- Más información del artículo
- Página de la revista en redalyc.org

redalyc.org

Sistema de Información Científica Redalyc

Red de Revistas Científicas de América Latina y el Caribe, España y Portugal
Proyecto académico sin fines de lucro, desarrollado bajo la iniciativa de acceso
abierto

Los paquetes deSolve y phaseR para la resolución numérica de Ecuaciones Diferenciales Ordinarias con R

The deSolve and phaseR packages for numerical resolution of Ordinary Differential Equations with R

Saulo Mosquera López¹ *

¹Universidad de Nariño. XXX, Colombia.

*samolo@udenar.edu.co

18 junio de 2023 - 15 noviembre 2023

Resumen: En el lenguaje R se han desarrollado paquetes específicos que posibilitan el tratamiento numérico de problemas de valor inicial para ecuaciones y sistemas de ecuaciones diferenciales. En particular, el paquete **deSolve** permite, entre otros, la solución numérica y gráfica de problemas de valor inicial para ecuaciones diferenciales ordinarias y el paquete **phaseR** el análisis cualitativo de sistemas autónomos de ecuaciones diferenciales ordinarias, en una y dos dimensiones. El propósito de este artículo es el de ilustrar el uso del paquete deSolve con una modificación de la ecuación logística y el sistema de Chen y el paquete phaseR con un modelo que generaliza la ley de Hooke.

Palabras clave: Problemas de valor inicial, deSolve, phaseR, solución numérica, estabilidad, plano fase.

Abstract: In the R language, specific packages have been developed that enable the numerical treatment of initial value problems for equations and systems of differential equations. In particular, the **deSolve** package allows, among others, the numerical and graphical solution of initial value problems for ordinary differential equations and the **phaseR** package the qualitative analysis of Autonomous Systems of ordinary differential equations, in a and two dimensions. The purpose of this article is to illustrate the use of the deSolve package with a modification of the logistic equation and Chen's system and the phaseR package with a model that generalizes Hooke's law.

*Saulo Mosquera López. XXX.XXX. Código Postal: XXX. Correo electrónico: samolo@udenar.edu.co.

Keywords: Initial value problems, deSolve, phaseR, numerical solution, stability, phase plane.

1 Introducción

Gran variedad de problemas importantes y significativos de la biología, la química, la economía, la física y la ingeniería, se pueden modelar, evaluar y resolver en términos de ecuaciones diferenciales. Por ejemplo, las ecuaciones diferenciales pueden utilizarse para tratar problemas sobre crecimiento de poblaciones, desintegración radioactiva, reacciones químicas, transferencia de calor, propagación de ondas, problemas sobre oferta y demanda, meteorología y otros (Zill, 2001).

Usualmente no es posible expresar la solución de estos problemas, mediante una fórmula explícita o implícita, por lo que es necesario recurrir a métodos numéricos para hallar una aproximación a esta solución. En el lenguaje R, de amplia difusión y utilización en la actualidad, se han desarrollado herramientas que permiten resolver numéricamente, problemas de valor inicial para ecuaciones y sistemas de ecuaciones diferenciales mediante procedimientos adecuados.

De acuerdo a Bologna, 2020, R además de un lenguaje de programación, es un proyecto colaborativo, libre, independiente y de código abierto que evolucionó a partir del lenguaje S, fue creado en 1993 en Nueva Zelanda por Ross Ihaka y Robert Gentleman y está conformado por un conjunto de programas integrados para el manejo de datos, simulaciones, cálculos y aunque usualmente es utilizado para el análisis estadístico sus potencialidades en la investigación son muy amplias.

En R una de las posibilidades para tratar numéricamente ecuaciones diferenciales ordinarias es el paquete **deSolve** y consecuentemente, uno de los propósitos de este documento es el de ilustrar el uso de rutinas incorporadas en este paquete para resolver numéricamente una ecuación diferencial que modifica, **la ecuación logística y el sistema de Chen**.

El modelo de crecimiento logístico fue considerado por primera vez por Pierre François Verhulst en 1838 para simular el hecho de que una población de pequeño tamaño puede tener inicialmente crecimiento exponencial que se va acercando de manera asintótica a cierto valor, por encima del cual el tamaño de la población decrece hacia el mismo (Brauer et al., 2008). Estas suposiciones conducen al siguiente problema de valor inicial

$$\begin{aligned}\frac{dP}{dt} &= aP \left(1 - \frac{P}{K}\right) \\ P(t_0) &= P_0.\end{aligned}$$

para el que, $a > 0$ y $K > 0$ son parámetros característicos del modelo.

Cuando a las hipótesis anteriores se les incorpora el hecho de que si el tamaño de la población es pequeño la razón de crecimiento es negativa, se obtiene una modificación del modelo logístico que matemáticamente se expresa como

$$\frac{dP}{dt} = aP \left(\frac{P}{L} - 1 \right) \left(1 - \frac{P}{K} \right)$$

$$P(t_0) = P_0.$$

para el cual L es una nueva constante. En la subsección 3.1 analizaremos este modelo utilizando el paquete **deSolve**.

El sistema de Chen es sistema autónomo tridimensional de ecuaciones diferenciales ordinarias que para ciertos valores de sus parámetros exhibe una gran variedad de comportamiento dinámico que incluye caos (Sooraksa and Chen, 2018). De manera explícita el sistema de Chen está definido por las siguientes ecuaciones

$$\begin{aligned}\frac{dx}{dt} &= -ax + ay \\ \frac{dy}{dt} &= (c-a)x + cy - xz \\ \frac{dz}{dt} &= xy - bz\end{aligned}$$

donde a , b y c son parámetros del mismo.

En la subsección 3.2 utilizamos **deSolve** para trazar algunas trayectorias de este sistema para el que, siguiendo a Sooraksa and Chen, 2018, seleccionamos aquellos valores de sus parámetros para los cuales el sistema presenta un conjunto de estructura geométrica caótica conocido como el atractor de Chen.

Complementariamente, R también posee el paquete **phaseR**, el cual se puede utilizar para identificar y clasificar los puntos de equilibrio, representar el campo direccional, las isoclinas nulas así como trayectorias con diferentes condiciones iniciales, para ecuaciones diferenciales autónomas en una dimensión y sistemas de ecuaciones diferenciales autónomos en dos dimensiones. De esta manera el segundo objetivo de este documento es el de ilustrar el uso de este paquete con un modelo que permite generalizar la ley de Hooke.

La ley de Hooke fue propuesta en el Siglo XVII por R. Hooke y expresa el hecho de que dentro de ciertos límites, la fuerza requerida para estirar un objeto elástico, es directamente proporcional a la longitud del objeto (Cerón and Guerrero, 2008). Esta ley se modela a través de una ecuación diferencial lineal de segundo orden y en la subsección 5.1 analizamos, utilizando **phaseR**, la ecuación diferencial ordinaria no lineal que resulta al añadirle a la ecuación que modela la ley de Hooke un término cúbico. De manera explícita la ecuación diferencial resultante es

$$m \frac{d^2x}{dt^2} + kx + ax^3 = 0$$

para la cuál m , k y a son parámetros.

En las secciones 2 y 4 se describen las principales características de los paquetes **deSolve** y **phaseR** y en las secciones 3 y 5 la forma de utilizarlos.

El código de los archivos R considerados en este documento puede consultarse en el repositorio GitHub en la dirección <https://github.com/saulomosquera/samolo?sear>.

2 El paquete deSolve

El paquete **deSolve** fue desarrollado, alrededor del año 2010, por K. Soetaert, R. W. Setzer y T. Petzoldt (K. Soetaert et al., 2010) y permite resolver problemas de valor inicial para:

- Ecuaciones Diferenciales Ordinarias (EDO).
- Ecuaciones Diferenciales Parciales (EDP).
- Ecuaciones Diferenciales con Retardo (EDR).
- Ecuaciones Diferenciales Algebraicas (EDA).

Este paquete contiene diferentes funciones que resuelven numéricamente problemas de valor inicial para estos tipos de ecuaciones diferenciales, en particular, para ecuaciones diferenciales ordinarias la función que provee una interfaz de carácter más general para la mayor parte de los métodos incorporados es la función **ode()**. El paquete **deSolve** posee 17 rutinas de integración que se pueden llamar a través de esta función o se pueden utilizar de forma independiente (W. Soetaert and Petzoldt, n.d.).

La sintaxis básica de la función **ode()** es

```
ode(y, times, func, parms, method = c("nombre" ), ...)
```

y sus argumentos se describen en la tabla 1.

Dos observaciones son necesarias:

- La función *func* debe definirse al menos con el formato

```
func = function(times, y, parms)
```

en el que *times* es el intervalo de integración, y define las condiciones iniciales y *parms* es un vector o una lista que contiene los parámetros del sistema.

Table 1: Los argumentos de `ode()` y su descripción. Elaboración propia.

<i>Argumento</i>	<i>Descripción</i>
<code>y</code>	Define las condiciones iniciales.
<code>times</code>	Corresponde al intervalo de tiempo durante el cual se corre el proceso.
<code>func</code>	Es la función <code>R</code> que calcula los valores de las derivadas del sistema de EDO.
<code>parms</code>	Define los parámetros utilizados en el sistema de EDO o es <code>NULL</code> en el caso de no tenerlos.
<code>method</code>	Corresponde al nombre de la rutina de integración a utilizar.

- La función `ode()` utiliza más argumentos de los descritos y los valores que estos toman por defecto son suficientes para nuestros propósitos. Si Ud. desea analizarlos, una vez instalado el paquete **deSolve**, puede escribir, desde la consola de Rstudio,

```
help(ode)
```

y oprimir `control+ enter` con lo cual se despliega una ventana en la que se puede consultar una descripción de cada uno de ellos.

Esta función utiliza por defecto un método de integración denominado **lsoda** el cual proporciona resultados apropiados para la mayoría de los casos ya que esta implementación posee la propiedad de intercambiar automáticamente entre problemas rígidos y no rígidos, es decir, el usuario no tiene necesidad de conocer, con anterioridad el tipo de problema que está tratando si no que que el algoritmo selecciona de manera automática el método apropiado a aplicar.

La sintaxis básica de **lsoda** es semejante a la de **ode**, es decir

```
lsoda(y, times, func, parms, ...)
```

y la interpretación de sus opciones coincide con la de los argumentos de dicha función.

Aunque para cumplir los propósitos de este escrito es suficiente utilizar la función `ode()`, como ilustración de otro método de integración incorporado en el paquete **deSolve**, en la subsección 5.1, para el caso del resorte suave, usaremos la función `rk()` que es una implementación que puede utilizarse para resolver problemas de valor inicial para sistemas de EDO de primer orden no rígidos. La función `rk` es una función de alto nivel que proporciona interfaces a una colección de algoritmos explícitos de un solo paso, de la familia Runge-Kutta con paso de tiempo fijo o variable. La sintaxis de esta función es análoga a la de las funciones `ode()` y `lsoda()` y la interpretación de sus argumentos es exactamente la misma,

aunque puede seleccionarse el método utilizar, dentro de la familia de algoritmos Runge-Kutta implementados.

Características adicionales de estas funciones pueden verse utilizando la ayuda del paquete **deSolve** a través de Rstudio o en W. Soetaert and Petzoldt, [n.d.](#)

Una vez instalado R y Rstudio, debemos instalar el paquete **deSolve**, una manera de realizar esto es escribir, en la interfaz de Rstudio:

```
install.packages ("deSolve")
```

y oprimir `control+ enter`. Después de unos segundos se muestra un mensaje análogo al siguiente

```
package 'deSolve' successfully unpacked and MD5 sums  
checked
```

que nos pone de manifiesto la instalación exitosa del mismo. Una vez cargado el paquete se requiere activar las librerías internas del mismo, esto se consigue escribiendo en Rstudio

```
library(deSolve)
```

y oprimiendo `control+ enter`.

De acuerdo con K. Soetaert et al., [2016](#) la definición en R de un problema de valor inicial requiere de dos etapas básicas que se pueden definir como:

La **explicitación del modelo**, que, en general, incluye los siguientes pasos:

- La definición de los parámetros del modelo y sus correspondientes valores.
- La definición de las variables de estado del modelo y de sus condiciones iniciales.
- La implementación de las ecuaciones del modelo las cuales brindan la razón de cambio de las variables de estado.

La **carga del modelo** que consta de:

- La inclusión de los tiempos en los que se desea conocer los estados del modelo.
- La integración de las ecuaciones del modelo.
- La representación tabular o gráfica de los resultados del mismo.

El siguiente ejemplo muestra de manera concisa cómo aplicar este esquema para resolver en R el siguiente problema.

Ejemplo 1. Resolver en R el problema de valor inicial

$$\frac{dx}{dt} = x^2 + xt$$
$$x(0) = 0.1$$

en el intervalo $[0, 1]$, con paso 0.01, mostrar las primeras 4 iteraciones y construir la gráfica de la solución.

El código R que resuelve este problema, con las condiciones establecidas, es el siguiente

```
# Figura 1
install.packages("deSolve")
library(deSolve)
Ecu1= function(t, x, parms){
  dx = x^2+x*t
  list(dx)}
Cini= 0.1
tiempos =seq(from = 0, to = 1, by = 0.01)
solu=ode( y= Cini, times = tiempos, func = Ecu1, parms =
  NULL)
head(solu, n=4)
plot(solu, main = "_", lwd = 2, xlab="t", ylab="x(t)",
  col=c("magenta3"))
```

La tabla 2 muestra los resultados numéricos de las primeras 4 iteraciones.

Table 2: Primeras 4 iteraciones. Elaboración propia.

<i>Iter.</i>	<i>t</i>	<i>x(t)</i>
[1,]	0,00	0.1000000
[2,]	0,01	0.1001051
[3,]	0,02	0.1002205
[4,]	0,03	0.1003461

y la figura 1 muestra la gráfica de la solución del problema de valor inicial considerado.

En las siguientes secciones se ilustran algunas de las potencialidades del paquete **deSolve** y la función **ode()** para resolver numéricamente problemas de valor inicial para ecuaciones diferenciales ordinarias, así como para sistemas de ecuaciones diferenciales.

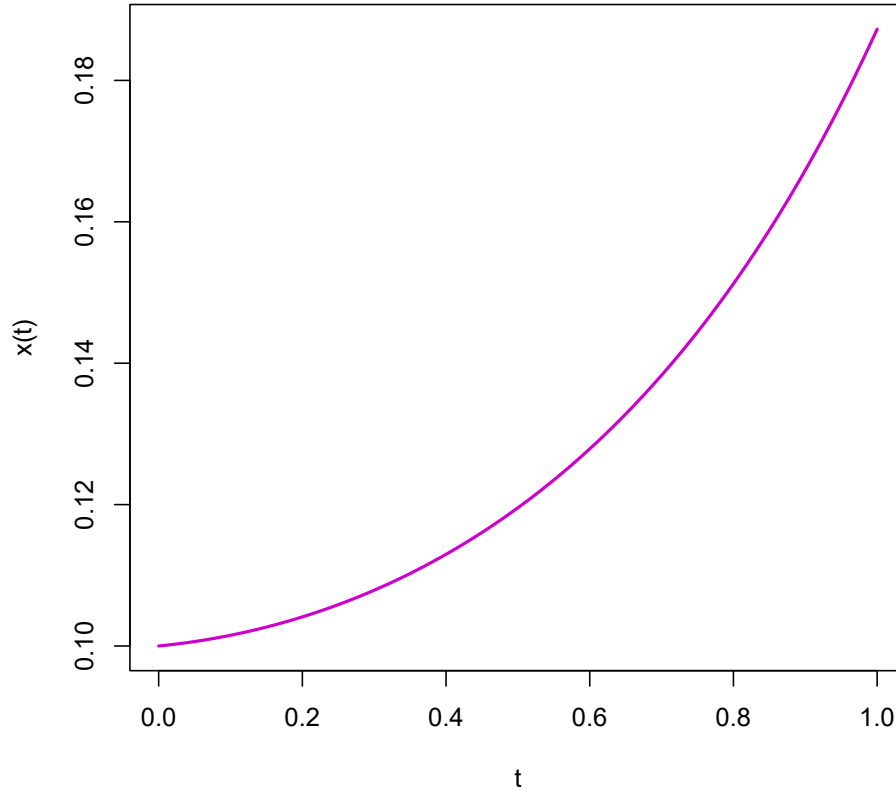


Figure 1: Simulación numérica del PVI $\frac{dx}{dt} = x^2 + xt$, $x(0) = 0.1$.
Elaboración propia.

3 Dos aplicaciones del paquete deSolve a problemas de la vida real

En los siguientes ejemplos se ilustra este proceso, el cual utiliza la función `ode()`, del paquete `deSolve`, así como algunas de sus características y opciones.

3.1 Una modificación de la ecuación logística

El modelo de crecimiento logístico fue publicado por primera vez por Pierre François Verhulst en 1838 y lo utilizó para describir el crecimiento auto-limitado de una población biológica. Las suposiciones básicas de este modelo son (Blanchard et al., 1999):

- Si la población es pequeña la tasa de crecimiento de la población es proporcional a su tamaño.
- Si la población es demasiado grande para ser soportada por su entorno y recursos, el tamaño de la población disminuye.

Un modelo matemático que recoge estas observaciones está definido por el siguiente problema de valor inicial:

$$\begin{aligned}\frac{dP}{dt} &= aP \left(1 - \frac{P}{K}\right) \\ P(t_0) &= P_0.\end{aligned}$$

En él, $a > 0$ y $K > 0$ son parámetros característicos del modelo; el valor de a se conoce como **la tasa intrínseca de crecimiento**, el valor K es **la capacidad de carga del sistema** y representa la población máxima con la que el medio se satura.

Autores como Blanchard et al., 1999 afirman que existen poblaciones en las que si el número de individuos es elevado, la razón de cambio decrece y puede llegar a ser negativa y si el tamaño de la población es demasiado pequeño esta razón también decrece. Un modelo que incorpora estas observaciones es el siguiente:

$$\begin{aligned}\frac{dP}{dt} &= aP \left(\frac{P}{L} - 1\right) \left(1 - \frac{P}{K}\right) \\ P(t_0) &= P_0.\end{aligned}$$

en el cual L es una constante que se conoce como **el factor de escasez** y $0 < L < K$.

De acuerdo con Blanchard et al., 1999, la ardilla negra es un pequeño mamífero nativo de las montañas rocallosas cuya población es muy territorial y satisface el modelo de crecimiento logístico modificado; utilizaremos el paquete **deSolve** para analizar el comportamiento de las soluciones del modelo para lo cual, debido a su territorialidad, consideramos una tasa intrínseca de crecimiento moderada con valor $a = 1.5$, $L = 6$ como valor del factor de escasez y $K = 14$ como la capacidad de carga del sistema. Adicionalmente consideramos como tamaños iniciales de la población, los valores $P(0) = 3$, $P(0) = 5$, $P(0) = 7$, $P(0) = 9$, $P(0) = 15$ y $P(0) = 20$ que nos permiten sugerir conclusiones acerca del comportamiento de las soluciones del modelo.

3.1.1 La implementación en R.

En esta sección se desarrollan en R las etapas descritas en la sección 2, para resolver numéricamente la modificación de la ecuación logística con las condiciones iniciales consideradas anteriormente para lo cual utilizamos la función **ode()** del paquete **deSolve**, así mismo inicialmente se cargan las librerías internas del paquete.

Una vez cargado el paquete **deSolve**, las siguientes líneas muestran cómo se cargan sus librerías y se definen los parámetros del modelo.

```
# Figura 2
# install.packages("deSolve")
library(deSolve) # carga de las librerías
# La explicitación del modelo
# Vector que contiene los valores de los parámetros
params1=c(a=1.5,L=6, K=14)
```

Aquí definimos las condiciones iniciales para el modelo, cada una de ellas como un vector.

```
# Condiciones iniciales
cini1=c(3)
cini2=c(5)
cini3=c(7)
cini4=c(9)
cini5=c(15)
cini6=c(20)
```

La ecuación del modelo se incluye a través de una función con nombre Logis que calcula la razón de cambio de la variable de estado P, la devuelve como una lista y usa como argumentos los tiempos de corrida, la variable de estado y los parámetros.

```
# Ecuaciones
# P es la variable de estado
Logis= function(t, P, parms) {
  Salida1=with(as.list(c(P, parms)),{
    dP=a*P*(P/L-1)*(1-P/K)
    list(c(dP))})
  return(Salida1)}
```

El modelo se corre en el intervalo de tiempo $[0, 2]$ con paso 0.1 y se resuelve utilizando la función **ode()** que contiene la rutina **lsoda** como método de integración predeterminado y que tiene como argumentos básicos las condiciones iniciales, el tiempo de corrida, la función Logis y los parámetros.

```
# La carga del modelo
# Tiempos
tiempos1=seq(0,2,0.1)
# La integración utiliza la función ode.
sol1=ode(y=cini1, t=tiempos1, func=Logis, parms=params1)
sol2=ode(y=cini2, t=tiempos1, func=Logis, parms=params1)
sol3=ode(y=cini3, t=tiempos1, func=Logis, parms=params1)
sol4=ode(y=cini4, t=tiempos1, func=Logis, parms=params1)
sol5=ode(y=cini5, t=tiempos1, func=Logis, parms=params1)
sol6=ode(y=cini6, t=tiempos1, func=Logis, parms=params1)
```

La función `ode()` tiene como salida un objeto de la clase `deSolve` que es una matriz que contiene los valores de la variable de estado en los tiempos proporcionados. Las primeras 10 iteraciones de este proceso, para la solución, `sol1`, se obtienen con:

```
# La representación tabular
head(sol1, n=10)
```

Lo que genera una lista con los valores de los tiempos y el correspondiente valor de la variable de estado que se muestran en la tabla 3.

Table 3: Primeras 10 iteraciones. Elaboración propia.

<i>Iter.</i>	<i>t</i>	<i>P(t)</i>
[1,]	0,0	3.000000
[2,]	0,1	2.840869
[3,]	0,2	2.679054
[4,]	0,3	2.515542
[5,]	0,4	2.351415
[6,]	0,5	2.187841
[7,]	0,6	2.026040
[8,]	0,7	1.867238
[9,]	0,8	1.712629
[10,]	0,9	1.563335

Las gráficas de las seis soluciones y algunos textos ilustrativos, se obtienen con el siguiente código y se muestran en la figura 2.

```
# La representación gráfica
plot(sol1,sol2,sol3, sol4,sol5, sol6, lwd=2,
      xlab="t", ylab="P(t)", col=c("red", "green", "blue",
      "deeppink3", "brown4", "orange"),
      main= "Crecimiento_logístico_modificado")
legend("topright", legend=c("P(0)=3", "P(0)=5", "P(0)=7", "
P(0)=9",
"P(0)=15", "P(0)=20"),
col=c("red", "green", "blue", "deeppink3", "brown4", "
orange"),
lwd=c(2,2), bg="aliceblue")
```

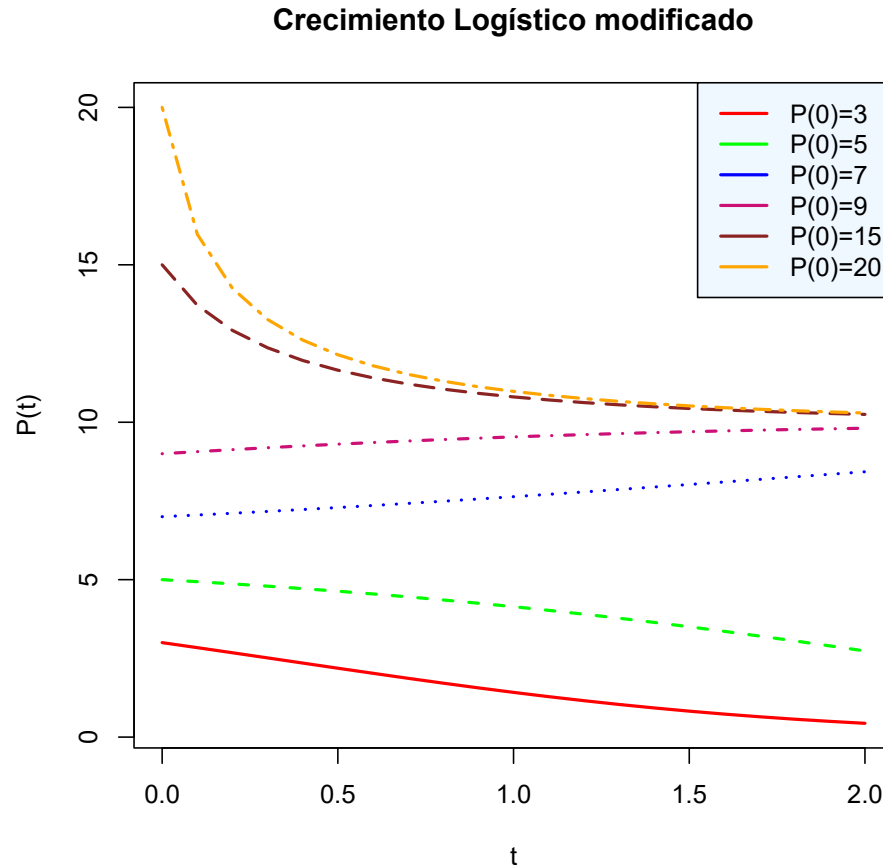


Figure 2: Simulaciones numéricas del modelo logístico modificado. Elaboración propia.

Esta gráfica sugiere que si el tamaño de la población inicial de ardillas negras es menor que la dada por el factor de escasez la población tiende a extinguirse, que si la población de ardillas negras está entre el factor de escasez y la capacidad de soporte el tamaño de la población crece acercándose a la capacidad de carga y que si el tamaño de la población es mayor que la capacidad de carga esta disminuye acercándose a este valor.

Modificamos el código anterior para incluir las soluciones constantes, $P = 0$, $P = 6$ y $P = 14$, denominadas **Puntos de equilibrio**, así como otras condiciones iniciales.

```
# figura 3
# install.packages("deSolve")
library(deSolve)
# Solución de la ecuación logística
tiempos2=seq(from = 0, to = 3, by = 0.01)
param2 = c(a=1.5, L=6, K=14)
logis = function(t,y,param){
```

```

salida = with(as.list(c(y,param)), {
  dP=a*P*(P/L-1)*(1-P/K)
  list(c(dP))})
return(salida)}

oncecur=seq(0,20,2)

listadatos=lapply(oncecur,function(i){ode(y=c(P=i), times
=tiempos2,func=logis,parms = param2)})

curvas= rbind(listadatos[[1]],listadatos[[2]],listadatos
[[3]],
  listadatos[[4]],listadatos[[5]],listadatos
  [[6]],
  listadatos[[7]],listadatos[[8]],listadatos
  [[9]],
  listadatos[[10]],listadatos[[11]])

curvas2=data.frame(curvas)
curvas2$group=c(rep("P(0)=0",301),rep("P(0)=2",301),
  rep("P(0)=4",301),rep("P(0)=6",301),rep("P(0)
=8",301),
  rep("P(0)=10",301),rep("P(0)=12",301),rep("P
(0)=14",301),
  rep("P(0)=16",301),rep("P(0)=18",301),rep("P
(0)=20",301))

curvas2

logis2=curvas2
# Modificar los datos
data_label = logis2
data_label$label = NA
data_label$label[which(data_label$time == min(data_label$
time)))] <- data_label$group[which(data_label$time ==
min(data_label$time)))]

install.packages("ggplot2")
require(ggplot2)

colcur = c("red", "#999999", "#E69F00", "red",
  "deeppink4", "green",
  "#000000",
  "red", "#0072B2", "#009E73" , "#CC79A7")

```

```

install.packages("ggrepel")
require(ggrepel)

# ggplot2 traza el gráfico con etiquetas
ggplot(data_label, aes(time, P, group = forcats::fct_inorder(group))) +
  geom_line(aes(colour= forcats::fct_inorder(group)), size
    =0.8) +
  scale_colour_manual(values=cbp2)+
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        panel.border = element_rect(colour = "black", size
          =0.5,
                                     fill=NA))+
  geom_label_repel(aes(label = label,color=group),
    nudge_x = -0.15,
    size=2,
    label.size=NA,
    fill="honeydew2",
    na.rm = TRUE,segment.color=NA) +
  theme(legend.position = "none") +
  labs(x = "t", y="P(t)")

```

El proceso anterior genera la gráfica que se muestra en la figura 3. Observe que todas las soluciones cercanas a la solución $P = 6$ se "alejan" de ella y que, en general, las soluciones "tienden" hacia la solución de equilibrio $P = 0$ o $P = 14$, es decir, $P = 6$ es una fuente y $P = 0$, $P = 14$ son sumideros.

3.2 El sistema de Chen

El concepto de **atractor extraño** surge a raíz de los trabajos de E. Lorenz, quien en 1963, escribió un notable artículo en el cual en un intento, por estudiar la posibilidad de predecir el comportamiento del flujo atmosférico a largo plazo, es decir, conocido el estado del flujo atmosférico en un instante dado determinar su comportamiento en tiempos futuros distantes, simuló experimentalmente el proceso convectivo atmosférico y observó que el flujo resultante podía ser muy irregular con trayectorias no periódicas inestables. Matemáticamente este proceso puede ser descrito utilizando un modelo de Rayleigh - Benard el cual es un sistema de ecuaciones en derivadas parciales que se deduce de las ecuaciones de Navier - Stokes para el que no se conocen soluciones exactas a excepción de las triviales. Suponiendo una expansión de Fourier de cierto tipo Lorenz reduce el sistema de

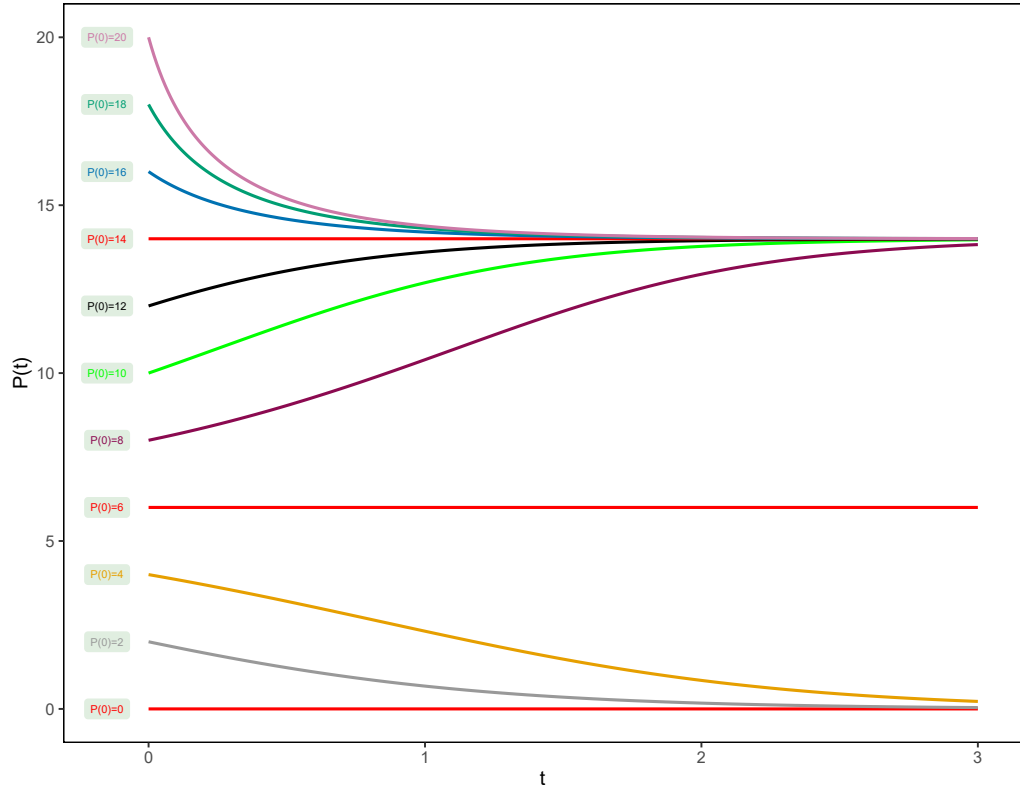


Figure 3: Simulaciones numéricas del modelo logístico modificado. Elaboración propia.

ecuaciones en derivadas parciales a un sistema autónomo de ecuaciones diferenciales ordinarias que se conoce como el sistema de Lorenz (Mosquera, 1992).

El Sistema de Lorenz posee dependencia sensitiva con respecto a las condiciones iniciales, consecuentemente exhibe comportamiento caótico y muestra lo que se conoce como un atractor extraño que corresponde, en su diagrama de fase, a una estructura geométrica con forma poco usual en la que las trayectorias del sistema nunca se cortan y son líneas de longitud infinita encerradas en un área finita, que describen órbitas no periódicas (Mosquera, 1992).

A partir del trabajo de Lorenz se han propuesto varios sistemas de ecuaciones diferenciales que contienen atractores extraños, entre ellos, el atractor de Rossler, el atractor de Chen, el atractor de Chua, el atractor de Liu y otros. En esta sección implementaremos en R el sistema de ecuaciones diferenciales ordinarias que produce **el atractor de Chen**.

El sistema de Chen (Sooraksa and Chen, 2018) fue propuesto en 1999 y puede considerarse como un modelo de clima controlado bajo la perspectiva anticontrol. Desde el punto de vista matemático, es el dual del sistema de Lorenz a través de la inversión en el tiempo y corresponde a un sistema autónomo de ecuaciones

diferenciales ordinarias definido por

$$\begin{aligned}\frac{dx}{dt} &= -ax + ay \\ \frac{dy}{dt} &= (c-a)x + cy - xz \\ \frac{dz}{dt} &= xy - bz\end{aligned}$$

Las variables x y z representan, el promedio espacial de la velocidad hidrodinámica y la diferencia de temperatura entre la corriente ascendente y la descendente y la variable y el gradiente de temperatura. a , b y c son parámetros característicos del sistema y dependiendo de sus valores se tienen diversas situaciones; por ejemplo, de acuerdo a Sooraksa and Chen, 2018 con los valores de los parámetros $a = 35$, $b = 3$ y $c = 28$ y las condiciones iniciales $x(0) = -3$, $y(0) = 2$, $z(0) = 20$ el sistema presenta comportamiento impredecible y se obtiene un conjunto de estructura geométrica caótica que se conoce cómo **el atractor de Chen**.

3.2.1 La implementación en R.

El sistema de Chen presenta una variada dinámica, sin embargo, presentamos el caso más importante de esta, su atractor extraño.

El siguiente código implementa la descripción realizada en la sección 2, para resolver numéricamente el Sistema de Chen, para lo cual nuevamente se utiliza la función `ode()` del paquete `deSolve`. Siguiendo a Sooraksa and Chen, 2018 consideramos los valores de los parámetros $a = 35$, $b = 3$ y $c = 28$ para los cuales se produce **el atractor de Chen**.

Con base en lo expuesto en K. Soetaert et al., 2016 para construir un gráfico de los resultados numéricos obtenidos es posible utilizar un método diseñado explícitamente para objetos de la clase `deSolve`, que ordena las figuras en dos filas y dos columnas. En este caso en ellas se presentan las gráficas de las variables dependientes contra la independiente y puesto que queda un espacio en el margen inferior derecho en él, se muestra el gráfico del promedio espacial de la velocidad hidrodinámica contra la diferencia de temperatura entre la corriente ascendente y la descendente, que en este caso corresponde a la proyección del atractor de Chen en el plano xz .

```
# Figura 4
# El Sistema de Chen
# Las ecuaciones del modelo
Chen = function (t, y, parms) {
  with(as.list(y), {
    dX = -a * X+a*Y
```

```

dY = (c-a)*X+c*Y - X*Z
dZ =X*Y - b*Z
list(c(dX, dY, dZ)) )})

# Valores de los parámetros y condiciones iniciales
param=c(a = 35, b = 3, c = 28)
Eini = c(X = -3, Y = 2, Z = 20)

# la integración se realiza por 100 veces cada 0.001 vez
library(deSolve)
tiempos = seq(from = 0, to = 100, by = 0.001)
salida = ode(y = Eini, times = tiempos, func = Chen,
             parms = param)

# La gráfica del modelo
plot(salida,xlab="tiempo", col= "blue", lwd = 1)
plot(salida[, "X"], salida[, "Z"], col= "red", type = "l",
     xlab = "X",
     ylab = "Z", main = "Proyección_plano_XZ")

```

La gráfica resultante de este proceso se muestra en la figura 4.

Las siguientes líneas de R, generan una gráfica 3D que ilustra el denominado **Atractor de Chen** y que se muestra en la figura 5.

```

# Figura 5
# gráfica 3D
library(scatterplot3d)
scatterplot3d(salida[, -1], type = "l", lwd = 1, xlab = "X",
              ylab = "Y", zlab = "Z", main = "El_atractor_de
              _Chen",
              col.main="red", color=("darkorchid"), box=FALSE, angle
              =45)

```

En esta sección se ha presentado únicamente una muestra del variado comportamiento que presenta este sistema para diversos valores de sus parámetros, en Sooraksa and Chen, 2018 puede encontrarse una descripción más completa de este sistema.

4 El paquete phaseR

El paquete **phaseR** fue desarrollado, alrededor del año 2012, por Michael J. Grayling, Gerhard Burger, Stephen P. Ellner y John M. Guckenheimer y es un

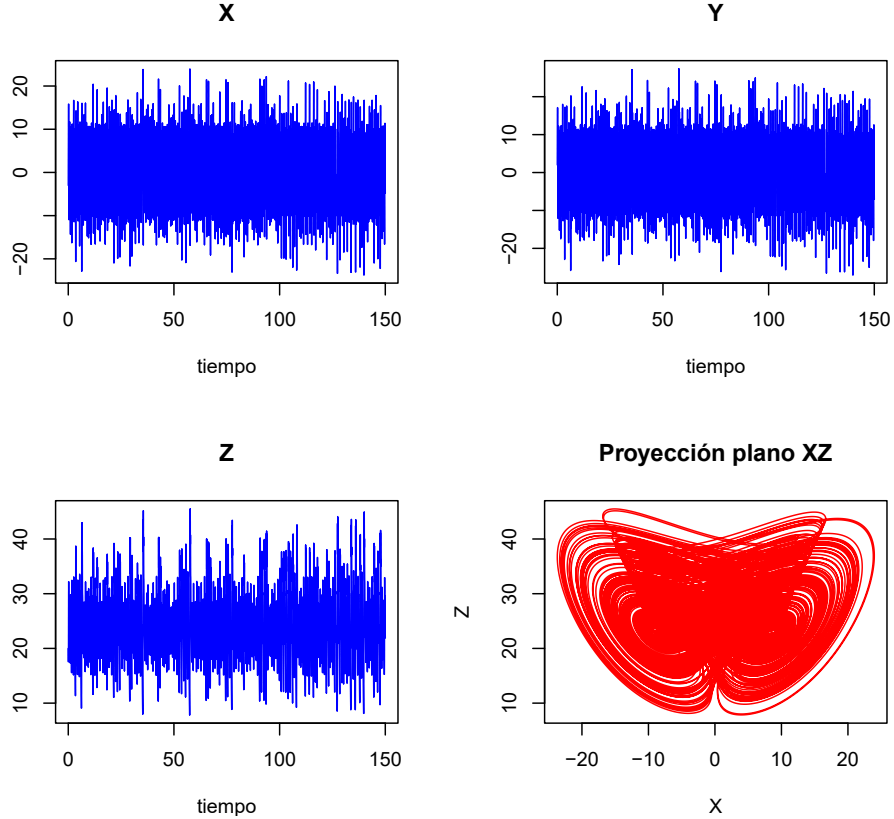


Figure 4: Simulaciones numéricas del sistema de Chen. Elaboración propia.

paquete R para el análisis cualitativo de sistemas autónomos de ecuaciones diferenciales ordinarias en una y dos dimensiones, para lo cual utiliza el método del plano fase. En la tabla 4 se muestra una descripción de las funciones incorporadas en este paquete (Grayling, 2022).

Su instalación se realiza de la manera usual, internamente el paquete utiliza la función `ode()` y el propósito central de esta sección es la de utilizar algunas de las funciones descritas en la tabla 4 para ilustrar las potencialidades del paquete para tratar desde el punto de vista cualitativo, problemas de valor inicial definidos para sistemas autónomos en $\mathbb{R}$ y en $\mathbb{R}^2$.

De acuerdo a Grayling, 2022 presentamos la sintaxis básica de cada una de las funciones del paquete `phaseR()` que utilizaremos en este documento.

Para la función `flowField` la sintaxis es:

```
flowField(func, xlim, ylim, parameters, points, system,
          add, ...)
```

y una descripción de sus argumentos se presenta en la tabla 5

El atractor de Chen

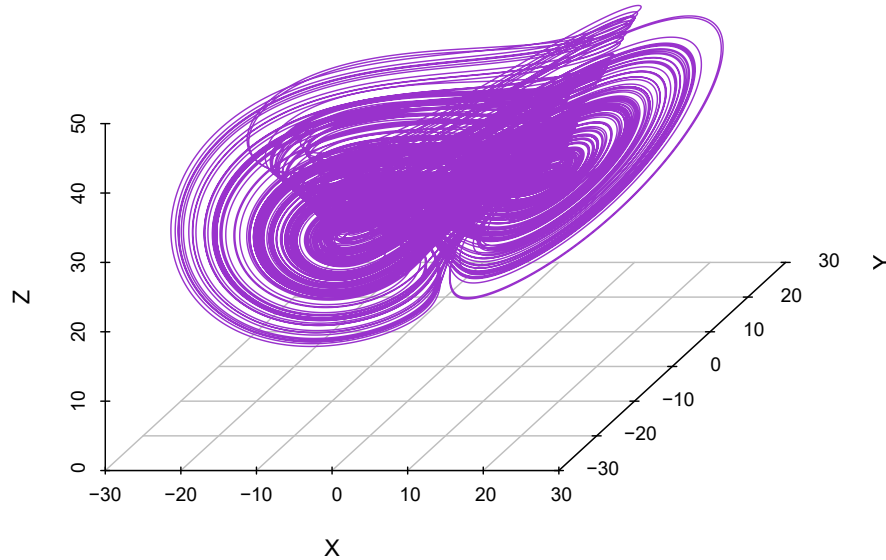


Figure 5: Una vista 3D del atractor de Chen. Elaboración propia.

Table 4: Descripción de las funciones del paquete phaseR. Elaboración propia.

<i>Función</i>	<i>Descripción</i>
flowField	Grafica el campo direccional de un sistema autónomo de EDO
nullclines	Traza las isoclinas nulas de un sistema autónomo de EDO en
numericalSolution	Resuelve numéricamente un sistema autónomo de EDO bidimensional
trajectory	Resuelve numéricamente un sistema autónomo, uno o dos dimensional
stability	Realiza un análisis de estabilidad para clasificar los puntos de equilibrio
drawManifolds	Traza la gráfica de las variedades estable e inestable de un punto de equilibrio
findEquilibrium	Identifica un punto de equilibrio de un sistema autónomo de EDO
phasePlaneAnalysis	Proporciona un método simple para realizar un análisis del plano de fase
phasePortrait	Traza el retrato fase de un sistema autónomo unidimensional

Para la función *nullclines* la sintaxis es:

```
nullclines(func, xlim, ylim, parameters, points, system,
```

Table 5: Los argumentos de **flowField()** y su descripción. Elaboración propia.

<i>Argumento</i>	<i>Descripción</i>
<code>func</code>	Es una función, definida en el formato de deSolve que calcula la derivada.
<code>xlim</code>	Para una ecuación diferencial en dimensión uno proporciona los límites de la variable independiente.
<code>ylim</code>	Para una ecuación diferencial en dimensión uno proporciona los límites de la variable dependiente.
<code>parameters</code>	Define los parámetros utilizados en el sistema de EDO o es NULL en el caso de no tenerlos.
<code>points</code>	Corresponde a un número que define la densidad de los segmentos que se muestran.
<code>add</code>	Es un valor lógico. Si es TRUE el campo vectorial se agrega a un gráfico existente.
<code>system</code>	Corresponde a "one.dim" o "two.dim" en dependencia del tipo de sistema.

add, ...)

en ella se observa que su sintaxis coincide con la de la función *flowFields* en consecuencia la interpretación de sus argumentos es exactamente la misma.

Para la función *numericalSolution* la sintaxis es:

```
numericalSolution(func, y0, tlim, type, parameters, col, ...)
```

y la descripción de sus argumentos se presenta en la tabla 6

Table 6: Los argumentos de **numericalSolution()** y su descripción. Elaboración propia.

<i>Argumento</i>	<i>Descripción</i>
<code>func</code>	Es una función, definida en el formato de deSolve que calcula la derivada.
<code>y0</code>	Es un vector numérico en dimensión dos que define la condición inicial.
<code>tlim</code>	Proporciona los límites de la variable independiente para los cuales se traza.
<code>parameters</code>	Define los parámetros utilizados en el sistema de EDO o es NULL en el caso de no tenerlos.
<code>type</code>	Únicamente toma los valores "one" o "two". Si se establece en "one", la variable dependiente es una sola.
<code>col</code>	Establece los colores de las trayectorias de las variables dependientes. E

Para la función *trajectory* la sintaxis es:

```
trajectory(func, y0, tlim, parameters, system, col, ...)
```

y una descripción de sus argumentos se presenta en la tabla 7.

Table 7: Los argumentos de **trajectory()** y su descripción. Elaboración propia.

<i>Argumento</i>	<i>Descripción</i>
<code>func</code>	Es una función, definida en el formato de deSolve que calcula la derivada.
<code>y0</code>	Define las condiciones iniciales. Para una EDO en dimensión uno, es un vector de longitud uno.
<code>tlim</code>	Proporciona el intervalo de valores de la variable independiente para la cual se calcula la trayectoria.
<code>parameters</code>	Define los parámetros utilizados en el sistema de EDO o es NULL en caso de no haberlos.
<code>system</code>	Toma dos valores "one.dim" o "two.dim" en dependencia del tipo de sistema de ecuaciones.
<code>col</code>	Es un vector de caracteres para dar color a las gráficas de las trayectorias.

Para la función *stability* la sintaxis es:

```
stability(func, ystar, parameters, system, summary, ...)
```

y una descripción de sus argumentos se presenta en la tabla 8

Table 8: Los argumentos de **stability()** y su descripción. Elaboración propia.

<i>Argumento</i>	<i>Descripción</i>
<code>func</code>	Es una función, definida en el formato de deSolve que calcula la derivada.
<code>ystar</code>	Corresponde al punto de equilibrio para el cual se desea analizar su estabilidad.
<code>parameters</code>	Define los parámetros utilizados en el sistema de EDO o es NULL en caso de no haberlos.
<code>system</code>	Toma dos valores "one.dim" o "two.dim" en dependencia del tipo de sistema de ecuaciones.
<code>summary</code>	Es un valor lógico. Si es TRUE se devuelve un resumen del análisis de estabilidad.

Cada una de estas funciones posee argumentos adicionales que no son necesarios para nuestros objetivos sin embargo una descripción de estos puede verse en Grayling, 2022.

El siguiente ejemplo muestra de manera sintética cómo utilizar algunas de estas funciones para realizar un análisis cualitativo del siguiente problema.

Ejemplo 2. *Utilizar R para generar el campo direccional, construir las isoclinas nulas, trayectorias con diversas condiciones iniciales y un análisis de estabilidad*

para los puntos de equilibrio de ecuación diferencial

$$\frac{dP}{dt} = aP \left(\frac{P}{L} - 1 \right) \left(1 - \frac{P}{K} \right)$$
$$P(0) = P_0.$$

Este es el modelo de la ecuación logística modificada considerada en la subsección 3.1 por lo que como valores de los parámetros consideramos los seleccionados en aquella sección. El código R que genera lo solicitado, es el siguiente

```
# figura 6
# install.packages("phaseR")
library(phaseR)
# Análisis cualitativo de la ecuación logística
# modificada
tiempos2=seq(from = 0, to = 5, by = 0.01)
param2 = c(a=1.5, L=6, K=14)
logis.mod = function(t,P,param){
  salida = with(as.list(c(P,param)),{
    dP=a*P*(P/L-1)*(1-P/K)
    list(c(dP))})
  return(salida)}

# El campo vectorial
logis.mod.flowField =
  flowField(logis.mod, xlim = c(0, 5), ylim = c(-1, 20),
    parameters = param2, points = 21,
    system = "one.dim", add = FALSE, xlab = "t",
    ylab = "P(t)")
grid()

# Las isoclinas nulas
logis.mod.nullclines =
  nullclines(logis.mod, xlim = c(0, 5), ylim = c(-1, 20),
    parameters = param2, system = "one.dim")

# Trayectorias con diversas condiciones iniciales
logis.mod.trajectory =
  trajectory(logis.mod, y0 = c(1.5, 2.5, 7.5, 11.5, 15, 18),
    tlim = c(0, 5),
    parameters = param2, system = "one.dim", col =
      rep("red", 6))

# Estabilidad de los puntos de equilibrio
```

```
logis.mod.Estab1=stability(logis.mod,ystar = c(0),
  parameters = param2, system = "one.dim")
logis.mod.Estab2=stability(logis.mod,ystar = c(6),
  parameters = param2, system = "one.dim")
logis.mod.Estab3=stability(logis.mod,ystar = c(14),
  parameters = param2, system = "one.dim")
```

El código anterior genera la gráfica de la figura 6 que muestra el campo direccional, las isoclinas nulas y trayectorias para algunas condiciones iniciales.

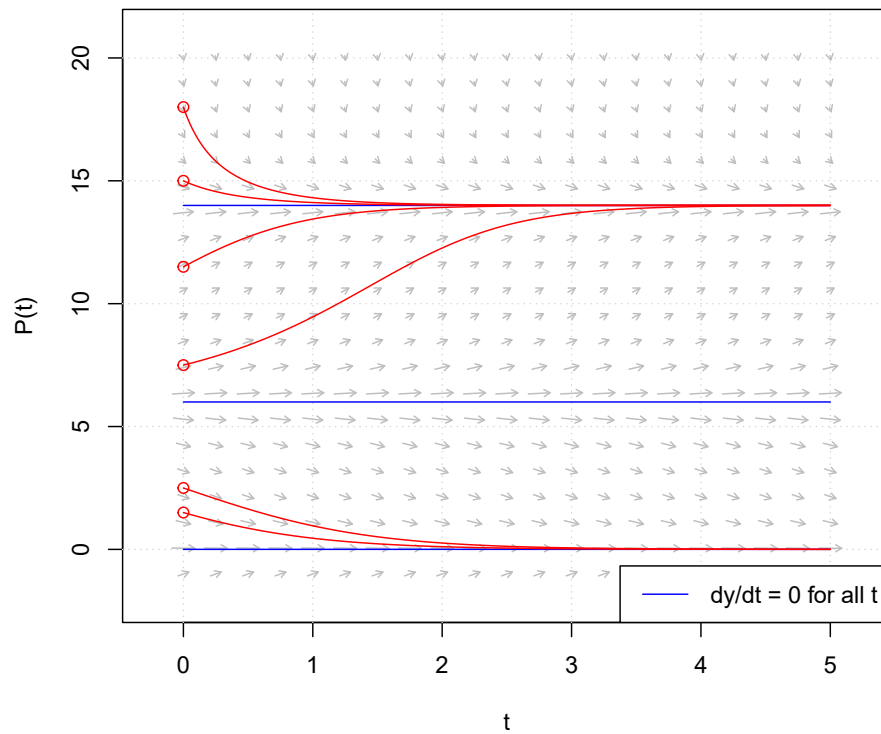


Figure 6: Campo vectorial, isoclinas nulas, trayectorias de la ecuación logística modificada. Elaboración propia.

En cuanto a estabilidad de los puntos de equilibrio los resultados son los siguientes

discriminant = -1.5, classification = Stable

discriminant = 0.85714, classification = Unstable

discriminant = -2, classification = Stable

y que por tanto clasifican a $P = 6$ como una el punto de equilibrio inestable y a $P = 0$ y $P = 14$ como puntos de equilibrio asintóticamente estables.

5 Una aplicación del paquete phaseR a un problema de la vida real

En los siguientes ejemplos se ilustra el uso de los paquetes, **deSolve** y **phaseR** los cuales utilizan la función **ode()**, así como algunas de las características y opciones del mismo con una ecuación diferencial de segundo orden no lineal.

5.1 Un modelo que generaliza la ley de Hooke.

La **Ley de Hooke** fue formulada en 1768 por Robert Hooke y básicamente establece que:

”La fuerza que devuelve un resorte a su posición de equilibrio es proporcional a la distancia que se desplaza de esta posición” (Cerón and Guerrero, 2008).

Cuando este enunciado se expresa matemáticamente produce una ecuación diferencial lineal de segundo orden que se estudia, usualmente, en un curso básico de ecuaciones diferenciales ordinarias. En este aparte se analiza la ecuación diferencial no lineal

$$m \frac{d^2x}{dt^2} + kx + ax^3 = 0$$

que describe el movimiento de un cuerpo de masa m que cuelga de un resorte y fue propuesta en Cerón and Guerrero, 2008 como un modelo que generaliza la ley de Hooke. Físicamente x representa el desplazamiento de un cuerpo de masa m en el extremo del resorte no lineal sin fricción, k es un parámetro positivo que se denomina elasticidad del resorte y cuya rigidez $a \neq 0$ varía con el desplazamiento. Si $a > 0$ la rigidez aumenta con el desplazamiento y se tiene un resorte duro, pero para $a < 0$ la rigidez disminuye con el desplazamiento y se tiene un resorte suave (Cerón and Guerrero, 2008).

La utilización de los paquetes **deSolve** y **PhaseR** para analizar esta ecuación diferencial requiere transformarla en un sistema dos-dimensional de ecuaciones diferenciales; para lo cual utilizamos el cambio de variable $\frac{dx}{dt} = y$ con lo que se obtiene el sistema no lineal bidimensional

$$\frac{dx}{dt} = y$$

$$\frac{dy}{dt} = -\frac{k}{m}x - \frac{a}{m}x^3$$

en el cual, llamando $c^2 = \frac{k}{m} > 0$ y $b = \frac{a}{2k} \neq 0$ se puede escribir como

$$\frac{dx}{dt} = y$$

$$\frac{dy}{dt} = -c^2(x + 2bx^3)$$

El resorte duro

Cerón et al (2008) demuestran que, en el caso del resorte duro, el comportamiento de las trayectorias es independiente del valor de b , por ello para considerar este caso, consideramos los valores $c^2 = 1$ y $b = 2$ e implementamos el siguiente código, que utiliza la función **ode** y genera las gráficas del desplazamiento y la velocidad en ventanas diferentes.

```
# Figura 7
library(deSolve)
#Definimos las condiciones iniciales
Cini = c(X=1, Y=0)
# Definimos los tiempos en los cuales queremos construir
  la solución
tiempos = seq(from=0, to=8, by=0.01)
# Definimos los valores de los parámetros del modelo
params = c(c=1, b=2)
# Definimos la función que expresa nuestro sistema de
  ecuaciones
hookel= function(t,y,params){
  Salida = with(as.list(c(y,params)),{
    dX = Y
    dY = -c^2*(X+2*b*X^3)
    list(c(dX,dY))
  })
  return(Salida)
}
# Se llama la función ode para construir la solución
Salida= ode(y = Cini, times = tiempos, func = hookel,
  params = params)
# Se construye la gráfica de la solución
plot(Salida,xlab="tiempo", col="red",type="l")
```

Este proceso que proporciona la imagen que se muestra en la figura 7.

Este gráfico no muestra toda la interacción del sistema, consecuentemente construimos un gráfico más acorde a nuestras necesidades para lo cual generamos,

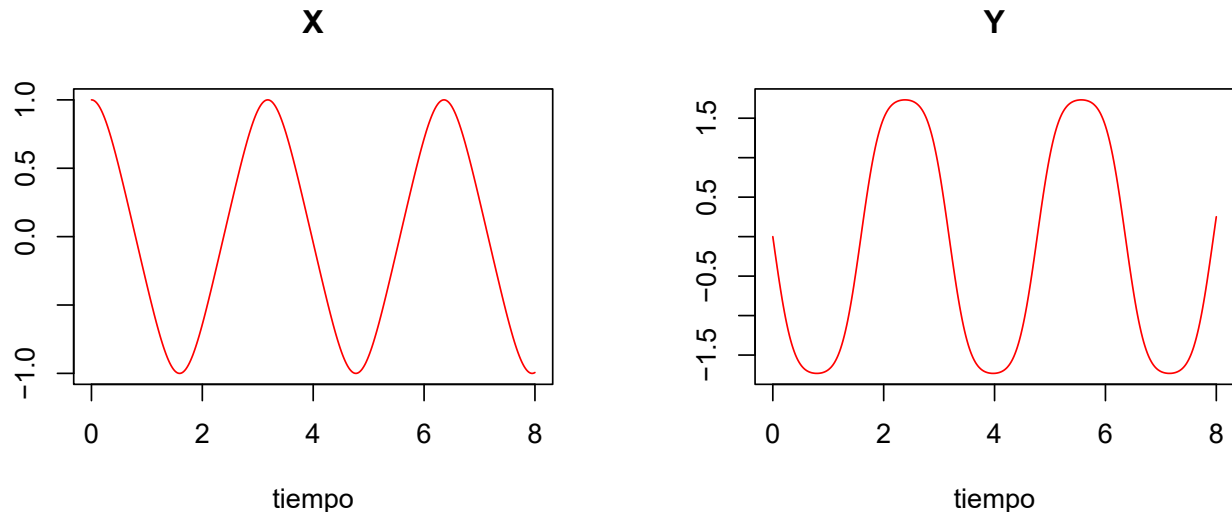


Figure 7: A la izquierda, Posición y a la derecha, Velocidad.
Elaboración propia.

en primer lugar, el gráfico del desplazamiento del resorte y en la misma ventana el de su velocidad.

```
# Figura 8
par(mfrow=c(1,1))
# Posicion X contra t
plot(Salida[, "time"], Salida[, "X"],
     ylim=c(-2, max(Salida[, "X"], 3)),
     type="l",
     col="red",
     lwd=2,
     xlab="Tiempo",
     ylab="",
     main="Un_modelo_que_generaliza_la_ley_de_Hooke")
grid()
# velocidad Y contra t
lines(Salida[, "time"], Salida[, "Y"],
      type="l",
      col="blue",
      lwd=2)
legend("topright",
      legend=c("Desplazamiento", "Velocidad"),
      col=c("red", "blue"),
      lwd=c(2, 2),
      bg="snow1")
```

Este proceso que genera la gráfica de la figura 8, la cual sugiere que las soluciones

de la ecuación diferencial tienen carácter periódico cómo se verifica teóricamente en Cerón and Guerrero, 2008.

Modelo de Hooke generalizado

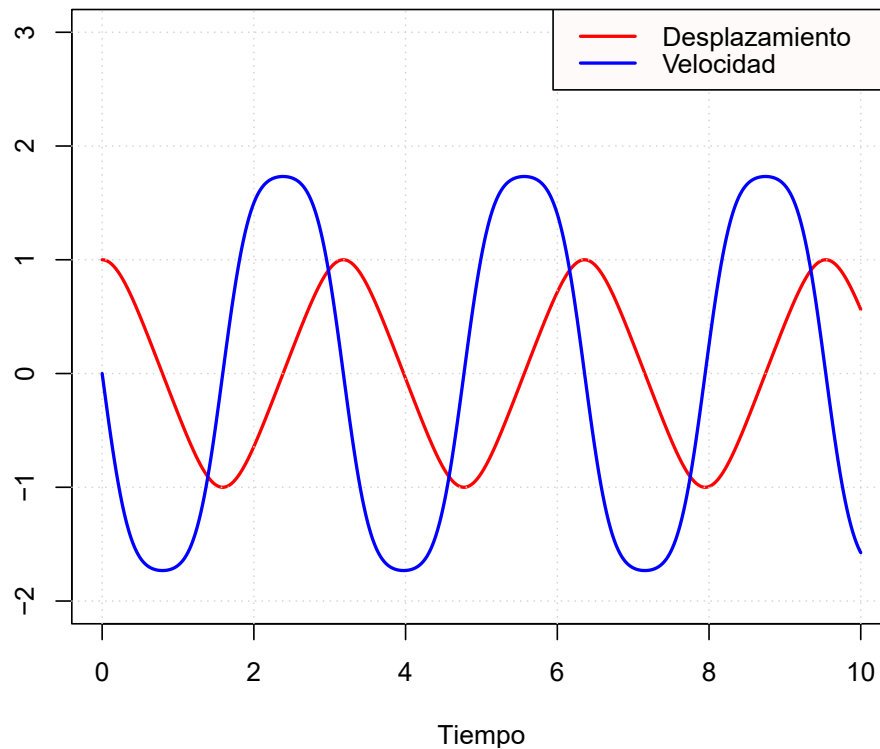


Figure 8: Desplazamiento y velocidad contra tiempo. Elaboración propia.

Una manera diferente de visualizar la dinámica de un sistema autónomo bidimensional, es utilizar el gráfico de las trayectorias en el plano fase, es decir, un gráfico del desplazamiento contra la velocidad, lo cual en R se puede obtener utilizando algunas de las funciones del paquete **phaseR**, ilustramos este hecho a continuación, para lo cual en primera instancia instalamos el correspondiente paquete y activamos sus librerías.

```
install.package{ "phaseR" }  
library(phaseR)
```

En el caso del resorte duro se demuestra en Cerón and Guerrero, 2008 que el sistema tiene un único punto de equilibrio en el punto $(0, 0)$ y que las trayectorias del sistema corresponden a órbitas cerradas para todo $b > 0$, consecuente con este hecho para el análisis de la ecuación seguimos considerando como valores de los parámetros $c = 1$ y $b = 2$.

El siguiente código nos ilustra, de una manera diferente, el comportamiento del

sistema en el plano fase, para el cual graficamos trayectorias con diferentes condiciones iniciales, las isoclinas nulas y el campo direccional.

```
# Figura 9
library(phaseR)
Hooke=function(t, y, param){
# Variables
  x =y[1]
  y =y[2]
# Parámetros
  c =param[1]
  b =param[2]
# Ecuaciones diferenciales
  dy =numeric(2)
  dy[1]=y
  dy[2]= -c^2*(x+2*b*x^3)
  list(dy)}
# Campo vectorial
Hooke.flowField=
  flowField(Hooke, xlim = c(-3, 5), ylim = c(-12, 12.5),
    param = c(1, 2), points =21 , add = FALSE)
# Isoclinas nulas
Hooke.nullclines =
  nullclines(Hooke, xlim = c(-5, 5), ylim = c(-12, 12.5),
    param = c(1, 2), points = 500)
Cini = matrix(c(1, 2, 1.5, 2, 2.5, 2, 3,5), ncol = 2,
  nrow = 4, byrow = TRUE)
# Curvas solución
Hooke.trajectory =
  trajectory(Hooke, y0 = Cini, tlim= c(0,10),
    param = c(1, 2), col = rep("red", 4))
```

En este proceso produce la gráfica que ilustra la figura 9 que verifica que para esta condición inicial y estos valores de los parámetros el sistema posee órbitas cerradas.

El siguiente script utiliza la función *numericalSolution* para graficar en una misma ventana, el desplazamiento y la velocidad contra el tiempo. La gráfica resultante se muestra en la figura 10.

```
# Figura 10
Hooke.num.sol=
  numericalSolution(Hooke, y0 = c(1, 2), tlim=c(0,10),
    type = "one",
    parameters = c(1, 2), col = c("red", "
```

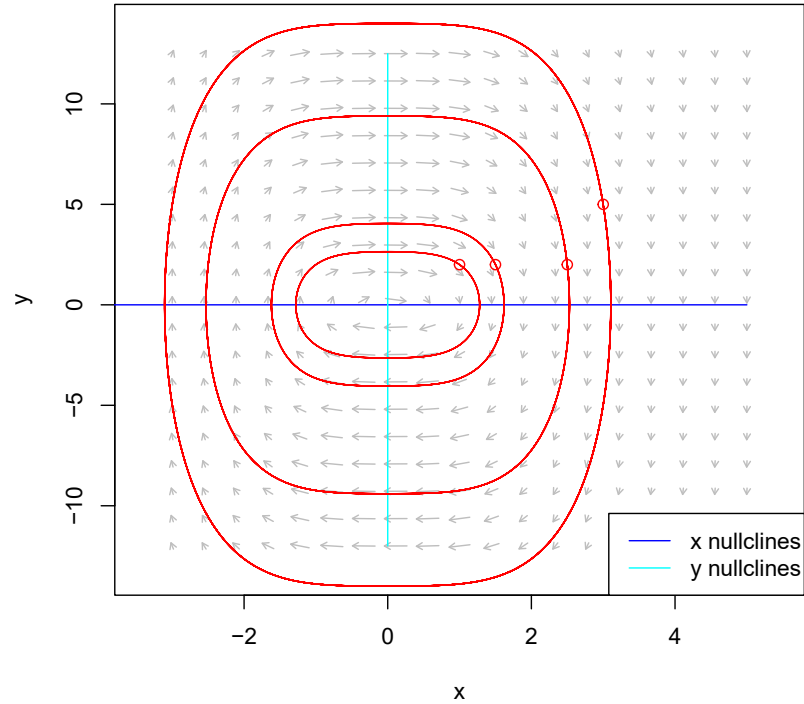


Figure 9: Campo direccional, isoclinas nulas y trayectorias, resorte duro. Elaboración propia.

```

        green"),
        ylab = "X, _Y", ylim = c(-3, 3))
legend("topright", legend=c("X", "Y"), col=c("red", "green"
    ),
        lwd=c(2,2), bg="aliceblue")

```

Para finalizar esta sección utilizamos la función *stability*, de este paquete, para determinar, la estabilidad del punto de equilibrio en el origen, lo que obtenemos con la siguiente línea.

```

Hooke.estabilidad.d=stability(Hooke, ystar=c(0,0),
    parameters=c(1,2))

```

R genera como respuesta la línea siguiente, que nos dice que el punto de equilibrio $(0,0)$ corresponde a un centro como se verifica teóricamente en Cerón and Guerrero, 2008.

tr = 0, Delta = 1, discriminant = -4, classification = Centre

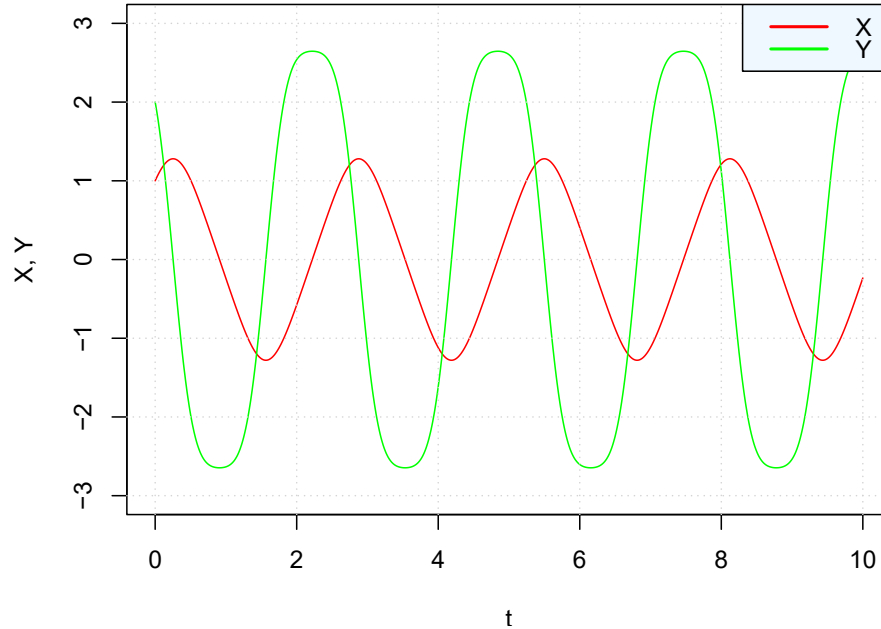


Figure 10: Desplazamiento y Velocidad contra el tiempo, resorte duro. Elaboración propia.

El resorte suave.

En este caso el sistema posee tres puntos de equilibrio, $(0, 0)$, $(\frac{1}{\sqrt{-2b}}, 0)$ y $(-\frac{1}{\sqrt{-2b}}, 0)$ y puesto que en Cerón at el (2008) se demuestra que el sistema posee órbitas cerradas en la región

$$D = \left\{ (x, y) / |x| < \frac{1}{\sqrt{-2b}}, y \in \mathbb{R} \right\}$$

para el análisis de la ecuación consideramos como valores de los parámetros $c = 1$ y $b = -2$ e inicialmente una condición inicial en el interior de esta región. En el primer script utilizamos, únicamente con fines ilustrativos, la función **rk** la cual usa directamente la rutina de integración Runge-Kutta5 como un método de integración por defecto y proporciona, en ventanas diferentes, las gráficas del desplazamiento y la velocidad contra el tiempo.

```
# Figura 11
library(deSolve)
Hooke2 = function(t, x, parms) {
  Salida1=with(as.list(c(parms, x)), {
    dX = Y
    dY = -c^2*(X+2*b*X^3)
    list(c(dX, dY))
  })
  return(Salida1) }
```

```

tiempos2 = seq(0, 20, 0.01)
params = c(c = 1, b = -2)
Cini2 = c(X = 0.3, Y = 0.2)
# La función rk usa ode45 como un método por defecto
Solucion= rk(y = Cini2, times = tiempos2, func = Hooke2,
  parms = params)
plot(Solucion, xlab="tiempo", col="red", type="l")

```

Este código produce la gráfica que se ilustra en la figura 11. Nótese que hemos utilizado como condición inicial el punto (0.3, 0.2) como valores de los parámetros $c = 1$ y $b = -2$ y que esta gráfica sugiere la existencia en el sistema de trayectorias periódicas.

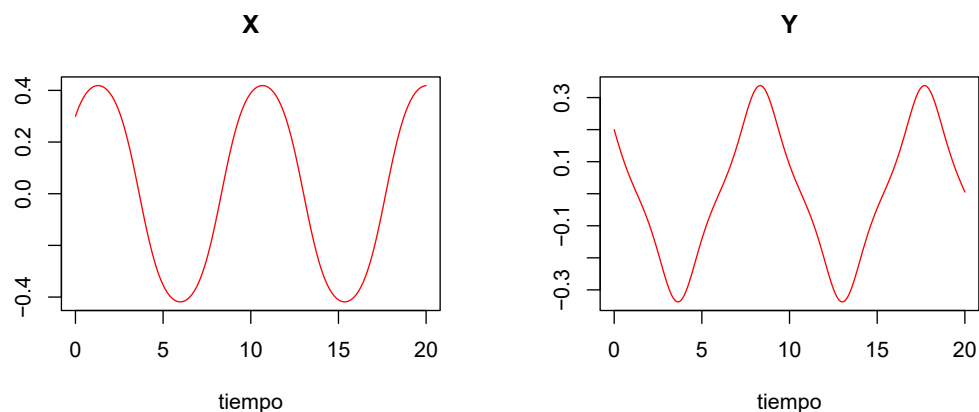


Figure 11: Posición y Velocidad, resorte suave, Cini=(0.3, 0.2).
Elaboración propia.

Si conservamos los valores de los parámetros, modificamos la condición inicial a (0.7, 0.5), el tiempo al intervalo [0, 1.2] y corremos nuevamente el código anterior, obtenemos la gráfica que se ilustra en la figura 12, la cual nos pone de manifiesto la posibilidad de la existencia de trayectorias no periódicas.

Si en el caso del resorte suave, utilizamos las funciones *trajectory*, *flowField* y *nullclines* para generar trayectorias con diversas condiciones iniciales, el campo vectorial y las isoclinas nulas, el código que genera la figura 13, es el siguiente.

```

# Figura 13
library(phaseR)
Hooke=function(t, y, parameters){
  #variables
  x =y[1]
  y =y[2]
  # parametros
  c =parameters[1]
  b =parameters[2]

```

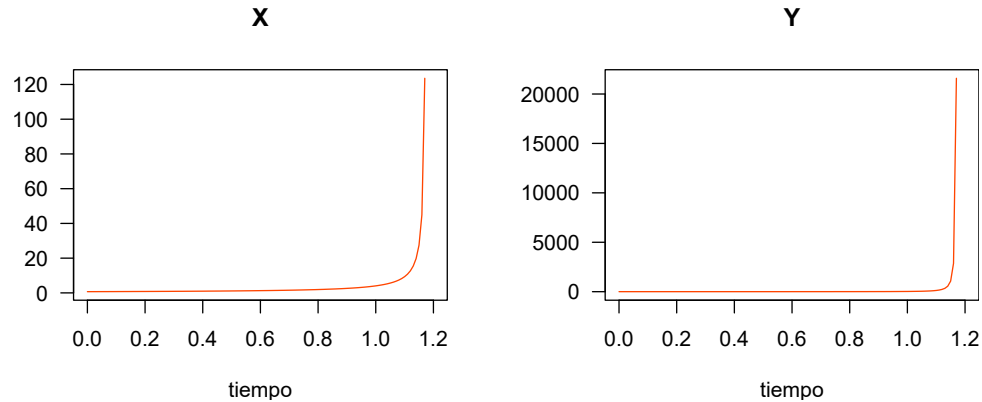


Figure 12: Posición y Velocidad, resorte suave, Cini=(0.7,0.5).
Elaboración propia.

```
# definimos las ecuaciones diferenciales
dy =numeric(2)
dy[1]=y
dy[2]= -c^2*(x+2*b*x^3)
list(dy) }
Hooke.flowField=
  flowField(Hooke, xlim = c(-2, 2), ylim = c(-2, 2),
            parameters = c(1, -2), points = 21, add = FALSE)
Hooke.nullclines=
  nullclines(Hooke, xlim = c(-2, 2), ylim = c(-2, 2),
            parameters = c(1, -2), points = 500)
Cini=matrix(c(0.1,0, 0.3,0, 0.2,0.1, 0.3, 0.2, -0.2,
             -0.3,0.2, 0.3,-1,1, 1,-1,-0.8,0.8,0.8,-0.8,
             -1,1.5,1,-1.5,-1.5,1.5,1.5,-1.5),
            ncol = 2, nrow = 14, byrow = TRUE)
Hooke.trajectory =
  trajectory(Hooke, y0 = Cini, tlim= c(0,10),
            parameters = c(1, -2), col = rep("
            mediumvioletred", 6))
```

El siguiente script, genera la gráfica de la figura 14, y codifica dos situaciones. En primer lugar, el desplazamiento y la velocidad contra el tiempo, para la condición inicial (0.3,0.2) la cual está en una región en la cual existen órbitas periódicas y en segundo lugar, el desplazamiento y la velocidad contra el tiempo, para la condición inicial (0.7,0.5) que está en una región en la que no existen órbitas periódicas. El script utiliza la función *numericalSolution* del paquete **phaseR**.

```
# figura 14
par(mfrow=c(1,2))
Hooke.num.sol=
```

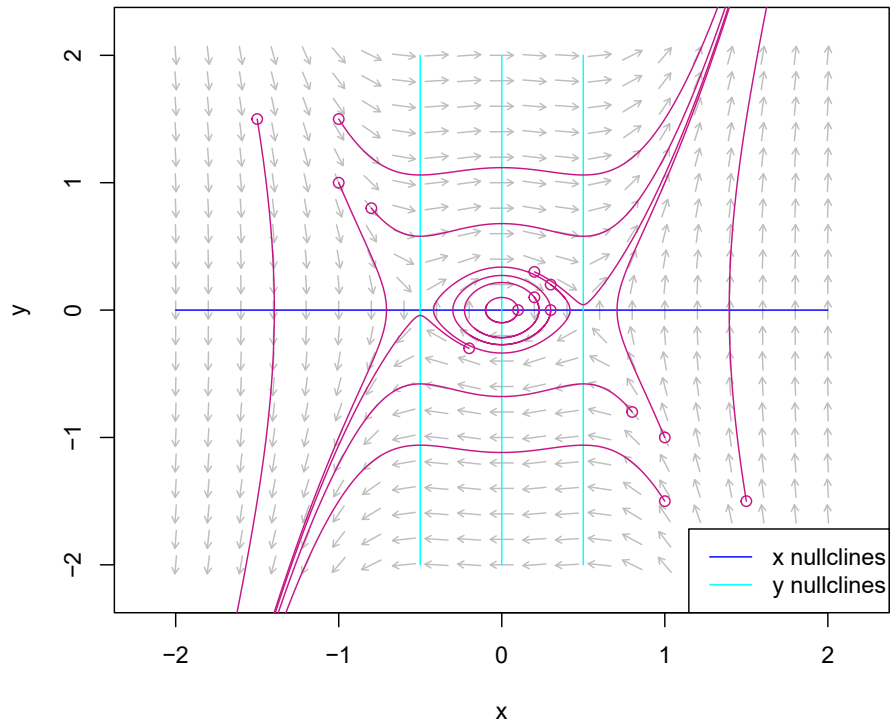


Figure 13: Campo direccional, isoclinas nulas y trayectorias, resorte suave. Elaboración propia.

```
numericalSolution(Hooke, y0 = c(0.3, 0.2), tlim=c(0,15),
  type = "one",
  parameters = c(1, -2), col = c("red", "
    green"),
  ylab = "X,_Y", xlab = "tiempo", ylim = c
    (-0.5, 0.5))
legend("topright",legend=c("X", "Y"),col=c("red", "green"
  ),
  lwd=c(2,2),
  bg="aliceblue")

Hooke.num.sol=
  numericalSolution(Hooke, y0 = c(0.7, 0.5), tlim=c
    (0,1.5),type = "one",
    parameters = c(1, -2), col = c("red", "
      green"),
    ylab = "X,_Y", xlab = "tiempo", ylim = c
      (0, 10))
legend("topright",legend=c("X", "Y"),col=c("red", "green"
  ),
```

```
lwd=c(2,2),
bg="aliceblue")
```

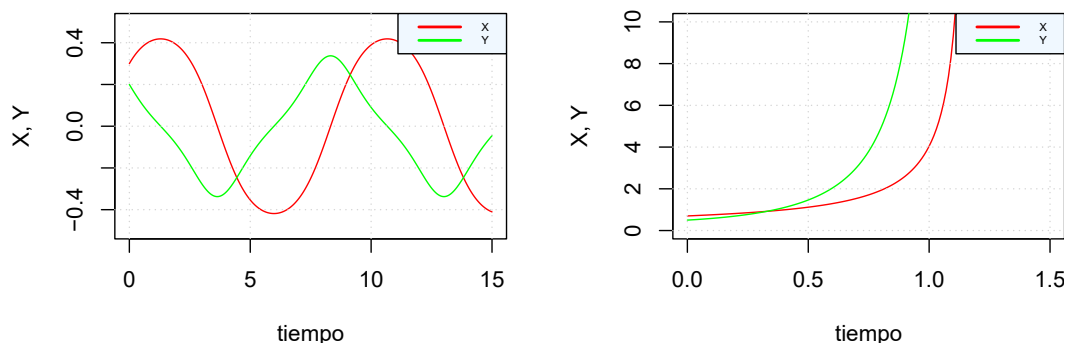


Figure 14: Desplazamiento y Velocidad contra el tiempo. Resorte suave: izquierda $C_{ini}=(0.3, 0.2)$, derecha $C_{ini}=(0.7, 0.5)$. Elaboración propia.

Estas gráficas que nos ilustran, en el caso del resorte suave, la diversidad de comportamiento que pueden tener las trayectorias del modelo que generaliza la ley de Hooke.

Cerramos esta sección utilizando la función *stability* para determinar la estabilidad de los puntos de equilibrio $(0, 0)$, $(1, 0)$ y $(-1, 0)$.

```
Hooke.Estab1=stability(Hooke, ystar = c(0, 0), parameters =
  c(1, -2))
Hooke.Estab2=stability(Hooke, ystar = c(1, 0), parameters =
  c(1, -2))
Hooke.Estab3=stability(Hooke, ystar = c(-1, 0), parameters =
  c(1, -2))
```

líneas de código que dan como resultado

```
tr = 0, Delta = 1, discriminant = -4, classification = Centre
tr = 0, Delta = -11, discriminant = 44, classification = Saddle
tr = 0, Delta = -11, discriminant = 44, classification = Saddle
```

y que por tanto clasifican, el punto de equilibrio $(0, 0)$ como un centro y los puntos de equilibrio $(1, 0)$ y $(-1, 0)$ como puntos silla.

6 Conclusiones

En esta sección final se presentan algunos comentarios que complementan lo considerado en el desarrollo de este documento.

- El propósito central de este escrito ha sido el de ilustrar la manera de utilizar los paquetes `deSolve` y `phaseR`, para tratar, desde el punto de vista numérico, problemas de valor inicial para ecuaciones diferenciales ordinarias y a lo largo del mismo se ha hecho énfasis en ello, sin embargo, no hemos comentado el uso de algunos detalles técnicos tales como: la ubicación de textos, la malla, el color y otros. Esperamos que un análisis de los correspondientes códigos le proporcione una idea de cómo incluirlos, o en su defecto, en la bibliografía, por ejemplo, en Wei, 2006 se puede encontrar una lista completa de los colores o en Chang, 2018 se ilustra la forma de utilizar muchos comandos y funciones de R para el manejo de gráficos.
- Al revisar la lista de paquetes que muestra, en Rstudio, la instalación inicial de R, se observa que los paquetes **deSolve**, **phaseR**, **scatterplot3d** y **ggplot2** no están incorporados, por ello, si usted desea experimentar con el código de este documento, o su propio código, para resolver problemas de valor inicial, se sugiere en primera instancia instalarlos y activar sus correspondientes librerías.
- Se debe ser cuidadoso al utilizar los códigos R proporcionados en este documento pues, en general, estos no son independientes unos de otros. Por ejemplo, si se corren el script que produce las Figuras 10, no se produce el resultado esperado pues este es dependiente del script que genera la Figura 9. De la misma manera el código de la figura 14 es dependiente del código de la figura 13.
- Los paquetes de R, **deSolve** y **phaseR**, además de ser herramientas que se utilizan en el ámbito científico con propósitos investigativos, también pueden ser utilizados como recurso didáctico, en el aula de clase, donde pueden ser empleados para ilustrar y afianzar los conocimientos adquiridos por los estudiantes en cursos de ecuaciones diferenciales ordinarias o métodos numéricos.
- Finalmente nos permitimos insistir en el hecho de que el código de los archivos R considerados en este trabajo se encuentran en el repositorio GitHub en la dirección <https://github.com/saulomosquera/samolo?sear>.

7 Bibliografía

Blanchard, P., Devaney, R., & Hall, G. (1999). *Ecuaciones diferenciales*. International Thomson Editores, S.A. de C.V.

Bologna, E. (2020). Un recorrido por los métodos cuantitativos en ciencias sociales a bordo de r. *Prueba de hipótesis: las aplicaciones*. <https://estadisticacienciasocialesr.rbind.io/>

- Brauer, F., Van den Driessche, P., Wu, J., & Allen, L. J. (2008). *Mathematical epidemiology* (Vol. 1945). Springer.
- Cerón, J., & Guerrero, L. (2008). *Análisis cualitativo de una generalización de la ley de hooke*. [Trabajo de grado para optar el título de Licenciado en Matemáticas]. Universidad de Nariño.
- Chang, W. (2018). *R graphics cookbook practical recipes for visualizing data* (2nd edition). <https://r-graphics.org/index.html>
- Grayling, M. (2022). *Package phaser: Phase plane analysis of one and two dimensional autonomous*. (tech. rep.). <https://cran.r-project.org/web/packages/phaseR/phaseR.pdf>
- Mosquera, S. (1992). *El sistema de lorenz* [Tesis de maestría]. Universidad del Valle.
- Soetaert, K., Petzoldt, T., & R.W., S. (2010). Solving differential equations en r: Package desolve. *Journal of Statistical Software*, 33(9). <https://doi.org/10.18637/jss.v033.i09>
- Soetaert, K., Petzoldt, T., & W., S. R. (2016). *R package desolve: Solving initial value differential equations in [internet]*. (tech. rep.). <http://cran.r-project.org/web/packages/deSolve/vignettes/deSolve.pdf>
- Soetaert, W., K.and Setzer, & Petzoldt, T. (n.d.). *Package desolve: Solvers for initial value problems of differential equations*. (tech. rep.). <https://cran.r-project.org/web/packages/deSolve/deSolve.pdf>
- Sooraksa, P., & Chen, G. (2018). Chen system as a controlled weather model — physical principle, engineering design and real applications. *International Journal of Bifurcation and Chaos*.
- Wei, Y. (2006). *Color in r*. (tech. rep.). <https://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>
- Zill, D. (2001). *Ecuaciones diferenciales con aplicaciones de modelado*. Thomson Learning.

References

- [1] [1] Blanchard, P., Devaney, R., & Hall, G. (1999). Ecuaciones Diferenciales. International Thomson Editores, S.A. de C.V.
- [2] [2] Bologna, E. (2020). Un recorrido por los métodos cuantitativos en Ciencias Sociales a bordo de R. Prueba de hipótesis: las aplicaciones. <https://estadisticacienciasocialesr.rbind.io>.
- [3] [3] Brauer, F., Van den Driessche, P., Wu, J., & Allen, L. J. (2008). Mathematical epidemiology (Vol. 1945). Springer.
- [4] [4] Cerón, J. & Guerrero, L. (2008). Análisis cualitativo de una generalización de la Ley de Hooke. [Trabajo de grado para optar el título de Licenciado en Matemáticas]. Universidad de Nariño
- [5] [5] Chang, W. (2018). R graphics cookbook practical recipes for visualizing data (2nd edition). [https : / / r - graphics.org/index.html](https://r-graphics.org/index.html)
- [6] [6] Grayling, M. (2022). Package phaseR: Phase Plane Analysis of One and Two Dimensional Autonomous. (inf.téc.). <https://cran.r-project.org/web/packages/phaseR/phaseR.pdf>
- [7] [7] Mosquera, S. (1992). El sistema de Lorenz [Tesis de maestría]. Universidad del Valle.
- [8] [8] Soetaert, K., Petzoldt, T., & R.W., S. (2010). Solving Differential Equations en R: Package deSolve. Journal of Statistical Software, 33(9). <https://doi.org/10.18637/jss.v033.i09>.
- [9] [9] Soetaert, K., Petzoldt, T., & W., S. R. (2016). R package deSolve: Solving initial value differential equations in [Internet]. (inf. téc.). <http://cran.r-project.org/web/packages/deSolve/vignettes/deSolve.pdf>
- [10] [10] Soetaert, W., K. and Setzer, & Petzoldt, T. (s.f.). Package deSolve: Solvers for Initial Value Problems of Differential Equations. (inf. téc.). <https://cran.r-project.org/web/packages/deSolve/deSolve.pdf>
- [11] [11] Sooraksa, P., & Chen, G. (2018). Chen System as a Controlled Weather Model — Physical Principle, Engineering Design and Real Applications. International Journal of Bifurcation and Chaos.
- [12] [12] Wei, Y. (2006). Color in R. (inf. téc.). <https://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>.
- [13] [13] Zill, D. (2001). Ecuaciones diferenciales con aplicaciones de modelado. Thomson Learning.