Differential Equations in R

Tutorial useR conference 2011

Karline Soetaert, & Thomas Petzoldt

Centre for Estuarine and Marine Ecology (CEME)
Netherlands Institute of Ecology (NIOO-KNAW)
P.O.Box 140
4400 AC Yerseke
The Netherlands

k.soetaert@nioo.knaw.nl

Technische Universität Dresden
Faculty of Forest- Geo- and Hydrosciences
Institute of Hydrobiology
01062 Dresden
Germany

thomas.petzoldt@tu-dresden.de

September 15, 2011

Outline

- How to specify a model
- An overview of solver functions
- Plotting, scenario comparison,

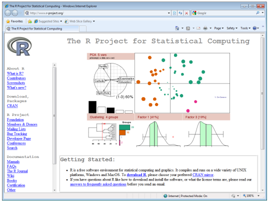
Outline

- How to specify a model
- An overview of solver functions
- Plotting, scenario comparison,

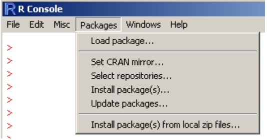
- Forcing functions and events
- Partial differential equations with ReacTran
- Speeding up

Installing the R Software and packages

Downloading R from the R-project website: `http://www.r-project.org`



Packages can be installed from within the R-software:

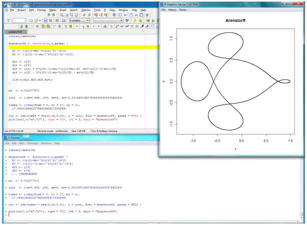


or via commandline

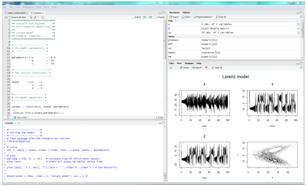
```
install.packages("deSolve", dependencies = TRUE)
```

Installing a suitable editor

Tinn-R is suitable (if you are a Windows adept)



Rstudio is very promising



Necessary packages

Several packages deal with differential equations

- deSolve: main integration package
- rootSolve: steady-state solver
- bvpSolve: boundary value problem solvers
- ReacTran: partial differential equations
- simecol: interactive environment for implementing models

All packages have at least one author in common
→ **Consistency** in interface

Getting help

- `?deSolve` opens the main help file
  - `Index` at bottom of this page opens an index page
- One main manual (or "vignette"):
  - `vignette("deSolve")`
  - `vignette("rootSolve")`
  - `vignette("bvpSolve")`
  - `vignette("ReacTran")`
  - `vignette("simecol-introduction")`
- Several dedicated vignettes:
  - `vignette("compiledCode")`
  - `vignette("bvpTests")`
  - `vignette("PDE")`
  - `vignette("simecol-Howto")`

Model specification

Let's begin ...

Logistic growth

Differential equation

$$\frac{dN}{dt} = r \cdot N \cdot \left(1 - \frac{N}{K}\right)$$

Analytical solution

$$N_t = \frac{K N_0 e^{rt}}{K + N_0 \left(e^{rt} - 1\right)}$$

R implementation

```
> logistic <- function(t, r, K, N0) {
+   K * N0 * exp(r * t) / (K + N0 * (exp(r * t) - 1))
+ }
> plot(0:100, logistic(t = 0:100, r = 0.1, K = 10, N0 = 0.1))
```

Numerical simulation in R

Why numerical solutions?

- Not all systems have an analytical solution,
- Numerical solutions allow discrete forcings, events, ...

Why R?

- If standard tool for statistics, why x$$$ for dynamic simulations?
- Other reasons will show up at this conference (useR!2011).

Numerical solution of the logistic equation

http://desolve.r-forge.r-project.org

```
library(deSolve)
model <- function (time, y, parms) {
  with(as.list(c(y, parms)), {
    dN <-   r * N * (1 - N / K)
    list(dN)
  })
}

y      <- c(N = 0.1)
parms  <- c(r = 0.1, K = 10)
times  <- seq(0, 100, 1)

out <- ode(y, times, model, parms)

plot(out)
```

Differential equation „similar to formula on paper"

Numerical methods provided by the deSolve package

Inspecting output

- Print to screen

```
> head(out, n = 4)

     time         N
[1,]    0 0.1000000
[2,]    1 0.1104022
[3,]    2 0.1218708
[4,]    3 0.1345160
```

- Summary

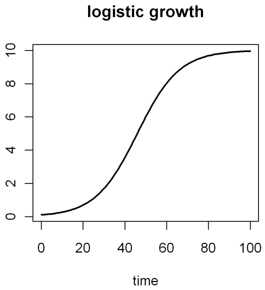
```
> summary(out)

                   N
Min.        0.100000
1st Qu.     1.096000
Median      5.999000
Mean        5.396000
3rd Qu.     9.481000
Max.        9.955000
N         101.000000
sd          3.902511
```

Inspecting output -ctd

- Plotting

```
> plot(out, main = "logistic growth", lwd = 2)
```



Inspecting output -ctd

- Diagnostic features of simulation

```
> diagnostics(out)
--------------------
lsoda return code
--------------------

  return code (idid) =  2
  Integration was successful.

--------------------
INTEGER values
--------------------

  1 The return code : 2
  2 The number of steps taken for the problem so far: 105
  3 The number of function evaluations for the problem so far: 211
  5 The method order last used (successfully): 5
  6 The order of the method to be attempted on the next step: 5
  7 If return flag =-4,-5: the largest component in error vector 0
  8 The length of the real work array actually required: 36
  9 The length of the integer work array actually required: 21
 14 The number of Jacobian evaluations and LU decompositions so far: 0
 15 The method indicator for the last succesful step,
         1=adams (nonstiff), 2= bdf (stiff): 1
 16 The current method indicator to be attempted on the next step,
         1=adams (nonstiff), 2= bdf (stiff): 1
```

Coupled ODEs: the rigidODE problem

Problem [3]

- Euler equations of a rigid body without external forces.
- Three dependent variables (y_1, y_2, y_3), the coordinates of the rotation vector,
- I_1, I_2, I_3 are the principal moments of inertia.

Coupled ODEs: the rigidODE equations

Differential equation

$$
\begin{aligned}
y_1' &= (I_2 - I_3)/I_1 \cdot y_2 y_3 \\
y_2' &= (I_3 - I_1)/I_2 \cdot y_1 y_3 \\
y_3' &= (I_1 - I_2)/I_3 \cdot y_1 y_2
\end{aligned}
$$

Parameters

$$I_1 = 0.5, I_2 = 2, I_3 = 3$$

Initial conditions

$$y_1(0) = 1, y_2(0) = 0, y_3(0) = 0.9$$

Coupled ODEs: the rigidODE problem

R implementation

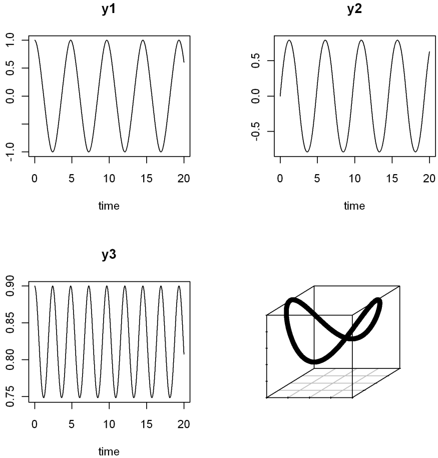
```
> library(deSolve)
> rigidode <- function(t, y, parms) {
+   dy1 <- -2   * y[2] * y[3]
+   dy2 <- 1.25* y[1] * y[3]
+   dy3 <- -0.5* y[1] * y[2]
+   list(c(dy1, dy2, dy3))
+ }
> yini  <- c(y1 = 1, y2 = 0, y3 = 0.9)
> times <- seq(from = 0, to = 20, by = 0.01)
> out   <- ode (times = times, y = yini, func = rigidode, parms = NULL)
```

```
> head (out, n = 3)
```

```
     time        y1         y2        y3
[1,] 0.00 1.0000000 0.00000000 0.9000000
[2,] 0.01 0.9998988 0.01124925 0.8999719
[3,] 0.02 0.9995951 0.02249553 0.8998875
```

Coupled ODEs: plotting the rigidODE problem

```
> plot(out)
> library(scatterplot3d)
> par(mar = c(0, 0, 0, 0))
> scatterplot3d(out[,-1], xlab = "", ylab = "", zlab = "", label.tick.marks = FALSE)
```



Exercise: the Rossler equations

Differential equation [12]

$$
\begin{aligned}
y_1' &= -y_2 - y_3 \\
y_2' &= y_1 + a \cdot y_2 \\
y_3' &= b + y_3 \cdot (y_1 - c)
\end{aligned}
$$

Parameters

$$a = 0.2, b = 0.2, c = 5$$

Initial Conditions

$$y_1 = 1, y_2 = 1, y_3 = 1$$

Exercise: the Rossler equations - ctd

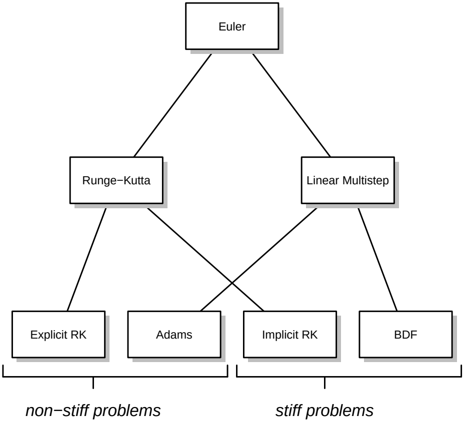
Tasks:

- Solve the ODEs on the interval $[0, 100]$
- Produce a 3-D phase-plane plot
- Use file `examples/rigidODE.R.txt` as a template

Solvers ...

Solver overview, stiffness, accuracy

Integration methods: package deSolve [20]



Solver overview: package deSolve

Function	Description
`lsoda` [9]	IVP ODEs, full or banded Jacobian, automatic choice for stiff or non-stiff method
`lsodar` [9]	same as `lsoda`; includes a root-solving procedure.
`lsode` [5], `vode` [2]	IVP ODEs, full or banded Jacobian, user specifies if stiff (bdf) or non-stiff (adams)
`lsodes` [5]	IVP ODEs; arbitrary sparse Jacobian, stiff
`rk4`, `rk`, `euler`	IVP ODEs; Runge-Kutta and Euler methods
`radau` [4]	IVP ODEs+DAEs; implicit Runge-Kutta method
`daspk` [1]	IVP ODEs+DAEs; bdf and adams method
`zvode`	IVP ODEs, like `vode` but for complex variables

adapted from [19].

Solver overview: package deSolve

Solver	Notes	stiff	y'=f(t,y)	My'=f(t,y)	F(y',t,y)=0	Roots	Events	Lags (DDE)	Nesting
lsoda/lsodar	automatic method selection	auto	x			x	x	x	
lsode	bdf, adams, …	user defined	x			x	x	x	
lsodes	sparse Jacobian	yes	x			x	x	x	
vode	bdf, adams, …	user defined	x				x	x	
zvode	complex numbers	user defined	x				x	x	
daspk	DAE solver	yes	x	x	x		x	x	
radau	DAE; implicit RK	yes	x	x		x	x	x	
rk, rk4, euler	euler, ode23, ode45, … rkMethod	no	x				x		x
iteration	returns state at t+dt	no	x				x		x

- ode, ode.band, ode.1D, ode.2D, ode.3D: top level functions (wrappers)
- red: functionality added by us

Options of solver functions

Top level function

```
> ode(y, times, func, parms,
+   method = c("lsoda", "lsode", "lsodes", "lsodar", "vode", "daspk",
+              "euler", "rk4", "ode23", "ode45", "radau",
+              "bdf", "bdf_d", "adams", "impAdams", "impAdams_d",
+              "iteration"), ...)
```

Workhorse function: the individual solver

```
> lsoda(y, times, func, parms, rtol = 1e-6, atol = 1e-6,
+   jacfunc = NULL, jactype = "fullint", rootfunc = NULL,
+   verbose = FALSE, nroot = 0, tcrit = NULL,
+   hmin = 0, hmax = NULL, hini = 0, ynames = TRUE,
+   maxordn = 12, maxords = 5, bandup = NULL, banddown = NULL,
+   maxsteps = 5000, dllname = NULL, initfunc = dllname,
+   initpar = parms, rpar = NULL, ipar = NULL, nout = 0,
+   outnames = NULL, forcings = NULL, initforc = NULL,
+   fcontrol = NULL, events = NULL, lags = NULL,...)
```

Arghhh, which solver and which options???

Problem type?

- ODE: use `ode`,
- DDE: use `dede`,
- DAE: `daspk` or `radau`,
- PDE: `ode.1D, ode.2D, ode.3D`,

...others for specific purposes, e.g. root finding, difference equations (`euler`, `iteration`) or just to have a comprehensive solver suite (`rk4, ode45`).

Stiffness

- default solver `lsoda` selects method automatically,
- `adams` or `bdf` may speed up a little bit if degree of stiffness is known,
- `vode` or `radau` may help in difficult situations.

Solvers for stiff systems

Stiffness

- Difficult to give a precise definition.

$\approx$ system where some components change more rapidly than others.

Sometimes difficult to solve:

- solution can be numerically unstable,
- may require very small time steps (slow!),
- deSolve contains solvers that are suitable for stiff systems,

But: "stiff solvers" slightly less efficient for "well behaving" systems.

- **Good news**: `lsoda` selects automatically between stiff solver (bdf) and nonstiff method (adams).

Van der Pol equation

Oscillating behavior of electrical circuits containing tubes [22].

2^{nd} order ODE

$$y'' - \mu(1 - y^2)y' + y = 0$$

... must be transformed into two 1^{st} order equations

$$y_1' = y_2$$
$$y_2' = \mu \cdot (1 - {y_1}^2) \cdot y_2 - y_1$$

- Initial values for state variables at $t = 0$: $y_{1_{(t=0)}} = 2, y_{2_{(t=0)}} = 0$
- One parameter: μ = large $\rightarrow$ stiff system; μ = small $\rightarrow$ non-stiff.

Model implementation

```
> library(deSolve)
> vdpol <- function (t, y, mu) {
+   list(c(
+          y[2],
+          mu * (1 - y[1]^2) * y[2] - y[1]
+   ))
+ }

> yini <- c(y1 = 2, y2 = 0)

> stiff <- ode(y = yini, func = vdpol, times = 0:3000, parms = 1000)
> nonstiff <- ode(y = yini, func = vdpol, times = seq(0, 30, 0.01), parms = 1)

> head(stiff, n = 5)

     time       y1            y2
[1,]    0 2.000000  0.0000000000
[2,]    1 1.999333 -0.0006670373
[3,]    2 1.998666 -0.0006674088
[4,]    3 1.997998 -0.0006677807
[5,]    4 1.997330 -0.0006681535
```

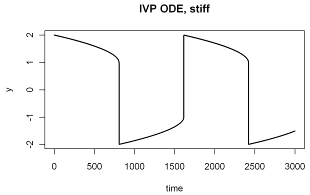
Interactive exercise

- The following link opens in a web browser. It requires a recent version of Firefox, Internet Explorer or Chrome, ideal is Firefox 5.0 in full-screen mode. Use Cursor keys for slide transition:
- **Left cursor** guides you through the full presentation.
- **Mouse** and **mouse wheel** for full-screen panning and zoom.
- **Pos1** brings you back to the first slide.
  - `examples/vanderpol.svg`
- The following opens the code as text file for life demonstrations in R
  - `examples/vanderpol.R.txt`

Plotting

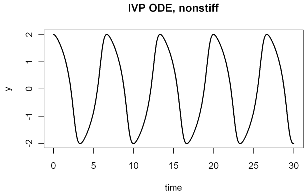
Stiff solution

```
> plot(stiff, type = "l", which = "y1",
+      lwd = 2, ylab = "y",
+      main = "IVP ODE, stiff")
```



Nonstiff solution

```
> plot(nonstiff, type = "l", which = "y1",
+      lwd = 2, ylab = "y",
+      main = "IVP ODE, nonstiff")
```



Default solver, lsoda:

```
> system.time(
+   stiff <- ode(yini, 0:3000, vdpol, parms = 1000)
+ )

   user  system elapsed
   0.59    0.00    0.61

> system.time(
+   nonstiff <- ode(yini, seq(0, 30, by = 0.01), vdpol, parms = 1)
+ )

   user  system elapsed
   0.67    0.00    0.69
```

Implicit solver, bdf:

```
> system.time(
+   stiff <- ode(yini, 0:3000, vdpol, parms = 1000,  method = "bdf")
+ )

   user  system elapsed
   0.55    0.00    0.60

> system.time(
+   nonstiff <- ode(yini, seq(0, 30, by = 0.01), vdpol, parms = 1, method = "bdf")
+ )

   user  system elapsed
   0.36    0.00    0.36
```

⇒ Now use other solvers, e.g. `adams`, `ode45`, `radau`.

Results

Timing results; your computer may be faster:

solver	non-stiff	stiff
ode23	0.37	271.19
lsoda	0.26	0.23
adams	0.13	616.13
bdf	0.15	0.22
radau	0.53	0.72

Comparison of solvers for a stiff and a non-stiff parametrisation of the van der Pol equation (time in seconds, mean values of ten simulations on an AMD AM2 X2 3000 CPU).

Accuracy and stability

- Options `atol` and `rtol` specify accuracy,
- Stability can be influenced by specifying `hmax` and `maxsteps`.

Accuracy and stability - ctd

atol (default 10^{-6}) defines absolute threshold,

- select appropriate value, depending of the size of your state variables,
- may be between $\approx 10^{-300}$ (or even zero) and $\approx 10^{300}$.

rtol (default 10^{-6}) defines relative threshold,

- It makes no sense to specify values $< 10^{-15}$ because of the limited numerical resolution of double precision arithmetics ($\approx$ 16 digits).

hmax is automatically set to the largest difference in times, to avoid that the simulation possibly ignores short-term events. Sometimes, it may be set to a smaller value to improve robustness of a simulation.

hmin should normally not be changed.

Example: Setting rtol and atol: `examples/PCmod_atol_0.R.txt`

Plotting, scenario comparison, observations

Plotting and printing

Methods for plotting and extracting data in deSolve

- `subset` extracts specific variables that meet certain constraints.
- `plot`, `hist` create one plot per variable, in a number of panels
- `image` for plotting 1-D, 2-D models
- `plot.1D` and `matplot.1D` for plotting 1-D outputs

- `?plot.deSolve` opens the main help file

rootSolve has similar functions

- `subset` extracts specific variables that meet certain constraints.
- `plot` for 1-D model outputs, `image` for plotting 2-D, 3-D model outputs

- `?plot.steady1D` opens the main help file

Chaos

The Lorenz equations

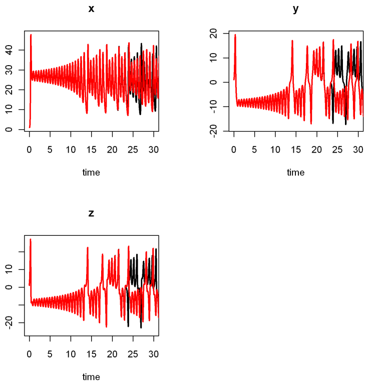
- chaotic dynamic system of ordinary differential equations
- three variables, X, Y and Z represent idealized behavior of the earth's atmosphere.

```
> chaos <- function(t, state, parameters) {
+   with(as.list(c(state)), {
+
+     dx     <- -8/3 * x + y * z
+     dy     <- -10 * (y - z)
+     dz     <- -x * y + 28 * y - z
+
+     list(c(dx, dy, dz))
+   })
+ }
> yini  <- c(x = 1, y = 1, z = 1)
> yini2 <- yini + c(1e-6, 0, 0)
> times <- seq(0, 100, 0.01)
> out  <- ode(y = yini, times = times, func = chaos, parms = 0)
> out2 <- ode(y = yini2, times = times, func = chaos, parms = 0)
```

Plotting multiple scenarios

- The default for plotting one or more objects is to draw a line plot
- We can plot as many objects of class `deSolve` as we want.
- By default different outputs get different colors and line types

```
> plot(out, out2, xlim = c(0, 30), lwd = 2, lty = 1)
```



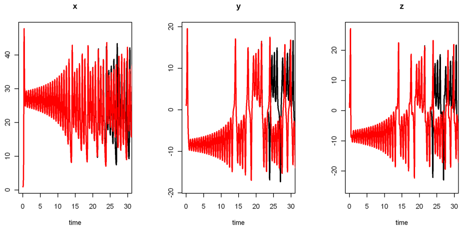
Changing the panel arrangement

Default

The number of panels per page is automatically determined up to 3 x 3 (`par(mfrow = c(3, 3))`).

Use mfrow() or mfcol() to overrule

```
> plot(out, out2, xlim = c(0,30), lwd = 2, lty = 1, mfrow = c(1, 3))
```



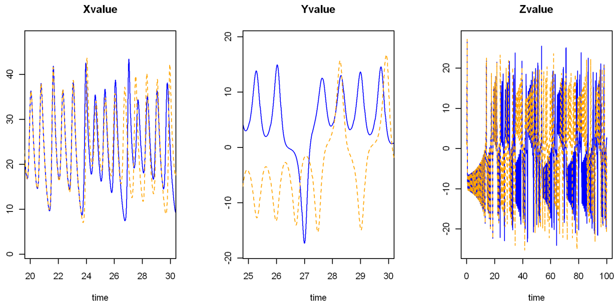
Important:

upon returning the default `mfrow` is **NOT** restored

Changing the defaults

- We can change the defaults of the *dataseries*, (`col`, `lty`, etc.)
  - will be effective for all figures
- We can change the default of each *figure*, (`title`, labels, etc.)
  - vector input can be specified by a list; `NULL` will select the default

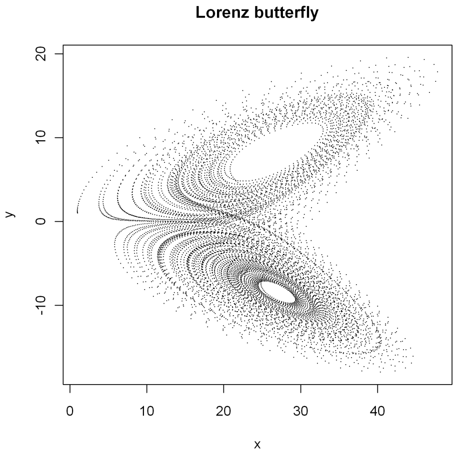
```
> plot(out, out2, col = c("blue", "orange"),  main = c("Xvalue", "Yvalue", "Zvalue"),
+      xlim = list(c(20, 30), c(25, 30), NULL), mfrow = c(1, 3))
```



R's default plot

- If we select x and y-values, R's default plot will be used

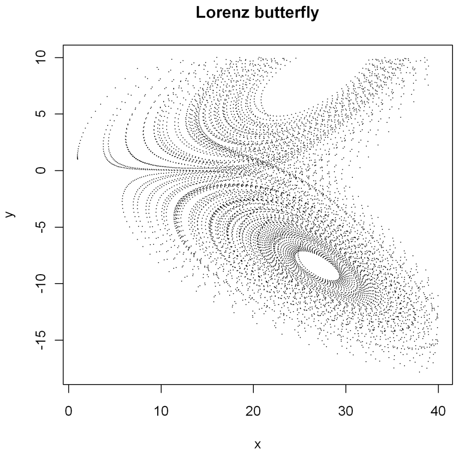
```
> plot(out[,"x"], out[,"y"], pch = ".", main = "Lorenz butterfly",
+      xlab = "x", ylab = "y")
```



R's default plot

- Use subset to select values that meet certain conditions:

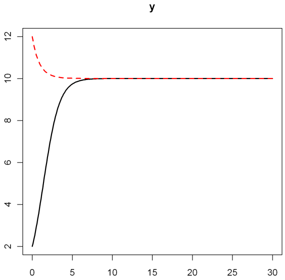
```
> XY <- subset(out, select = c("x", "y"), subset = y < 10 & x < 40)
> plot(XY, main = "Lorenz butterfly", xlab = "x", ylab = "y", pch = ".")
```



Plotting multiple scenarios

Simple if number of outputs is known

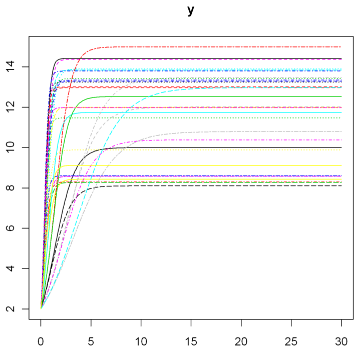
```
> derivs <- function(t, y, parms)
+    with (as.list(parms), list(r * y * (1-y/K)))
> parms <- c(r = 1, K = 10)
> yini  <- c(y = 2)
> yini2 <- c(y = 12)
> times <- seq(from = 0, to = 30, by = 0.1)
> out   <- ode(y = yini,  parms = parms, func = derivs, times = times)
> out2  <- ode(y = yini2, parms = parms, func = derivs, times = times)
> plot(out, out2, lwd = 2)
```



Plotting multiple scenarios

Use a list if many or unknown number of outputs

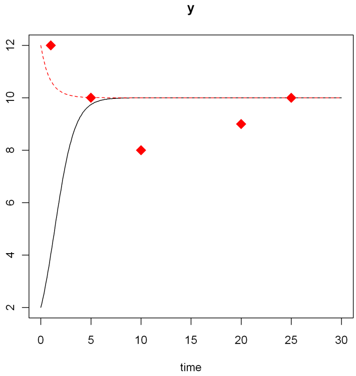
```
> outlist <- list()
> plist <- cbind(r = runif(30, min = 0.1, max = 5),
+                K = runif(30, min = 8, max = 15))
> for (i in 1:nrow(plist))
+   outlist[[i]] <- ode(y = yini, parms = plist[i,], func = derivs, times = times)
> plot(out, outlist)
```



Observed data

Arguments obs and obspar are used to add observed data

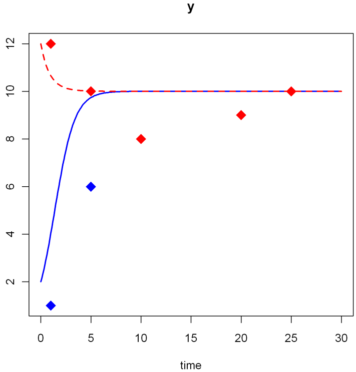
```
> obs2 <- data.frame(time = c(1, 5, 10, 20, 25), y =  c(12, 10, 8, 9, 10))
> plot(out, out2, obs = obs2, obspar = list(col = "red", pch = 18, cex = 2))
```



Observed data

A list of observed data is allowed

```
> obs2 <- data.frame(time = c(1, 5, 10, 20, 25), y =  c(12, 10, 8, 9, 10))
> obs1 <- data.frame(time = c(1, 5, 10, 20, 25), y =  c(1, 6, 8, 9, 10))
> plot(out, out2, col = c("blue", "red"), lwd = 2,
+      obs = list(obs1, obs2),
+      obspar = list(col = c("blue", "red"), pch = 18, cex = 2))
```



Under control: Forcing functions and events

Discontinuities in dynamic models

Most solvers assume that dynamics is *smooth*
However, there can be several types of discontinuities:

- Non-smooth *external variables*
- Discontinuities in the *derivatives*
- Discontinuites in the *values of the state variables*

A solver does not have large problems with first two types of discontinuities, but changing the values of state variables is much more difficult.

External variables in dynamic models

. . . also called forcing functions

Why external variables?

- Some important phenomena are not explicitly included in a differential equation model, but imposed as a *time series*. (e.g. sunlight, important for plant growth is never "modeled").
- Somehow, during the integration, the model needs to know the value of the external variable at each time step!

External variables in dynamic models

Implementation in R

- R has an ingenious function that is especially suited for this task: function `approxfun`
- It is used in two steps:
  - First an interpolating function is constructed, that contains the data. This is done before solving the differential equation.

    ```
    afun <- approxfun(data)
    ```

  - Within the derivative function, this interpolating function is called to provide the interpolated value at the requested time point (`t`):

    ```
    tvalue <- afun(t)
    ```

`?forcings` will open a help file

Example: Predator-Prey model with time-varying input

This example is from [15]

Create an artificial time-series

```
> times <- seq(0, 100, by = 0.1)
> signal <- as.data.frame(list(times = times, import = rep(0, length(times))))
> signal$import <- ifelse((trunc(signal$times) %% 2 == 0), 0, 1)
```

```
> signal[8:12,]
```

```
   times import
8    0.7      0
9    0.8      0
10   0.9      0
11   1.0      1
12   1.1      1
```

Create the interpolating function, using approxfun

```
> input <- approxfun(signal, rule = 2)
```

```
> input(seq(from = 0.98, to = 1.01, by = 0.005))
```

```
[1] 0.80 0.85 0.90 0.95 1.00 1.00 1.00
```

A Predator-Prey model with time-varying input

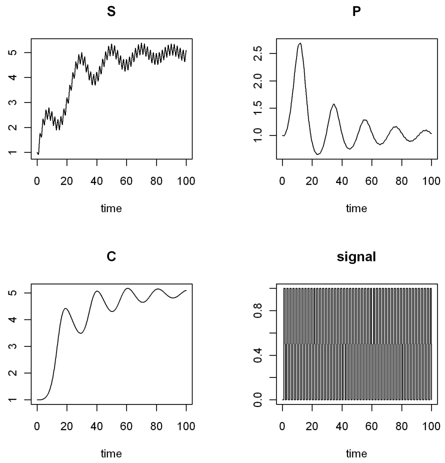
Use interpolation function in ODE function

```
> SPCmod <- function(t, x, parms) {
+    with(as.list(c(parms, x)), {
+
+    import <- input(t)
+
+    dS <- import - b * S * P + g * C
+    dP <- c * S * P - d * C * P
+    dC <- e * P * C - f * C
+    res <- c(dS, dP, dC)
+    list(res, signal = import)
+  })
+ }

> parms <- c(b = 0.1, c = 0.1, d = 0.1, e = 0.1, f = 0.1, g = 0)
> xstart <- c(S = 1, P = 1, C = 1)
> out <- ode(y = xstart, times = times,  func = SPCmod, parms)
```

Plotting model output

```
> plot(out)
```



Discontinuities in dynamic models: Events

What?

- An event is when the values of state variables change abruptly.

Events in Most Programming Environments

- When an event occurs, the simulation needs to be restarted.
- Use of loops etc. ...
- Cumbersome, messy code

Events in R

- Events are part of a model; no restart necessary
- Separate dynamics inbetween events from events themselves
- Very neat and efficient!

Discontinuities in dynamic models, Events

Two different types of events in R

- Events occur at *known times*
  - Simple changes can be specified in a `data.frame` with:
    - name of state variable that is affected
    - the time of the event
    - the magnitude of the event
    - event method ("replace", "add", "multiply")
  - More complex events can be specified in an `event function` that returns the changed values of the state variables `function(t, y, parms, ...)`.
- Events occur when certain *conditions* are met
  - Event is triggered by a `root` function
  - Event is specified in an `event function`

`?events` will open a help file

A patient injects drugs in the blood

Problem Formulation

- Describe the concentration of the drug in the blood
- Drug injection occurs at known times $\rightarrow$ `data.frame`

Dynamics inbetween events

- The drug decays with rate `b`
- Initially the drug concentration $= 0$:

```
> pharmaco <- function(t, blood, p) {
+   dblood <-  - b * blood
+   list(dblood)
+ }
```

```
> b    <- 0.6
> yini <- c(blood = 0)
```

A patient injects drugs in the blood

Specifying the event

- Daily doses, at same time of day
- Injection makes the concentration in the blood increase by 40 units.
- The drug injections are specified in a special event `data.frame`

```
> injectevents <- data.frame(var = "blood",
+                            time =  0:20,
+                            value = 40,
+                            method = "add")
```

```
> head(injectevents)
```

```
    var time value method
1 blood    0    40    add
2 blood    1    40    add
3 blood    2    40    add
4 blood    3    40    add
5 blood    4    40    add
6 blood    5    40    add
```

A patient injects drugs in the blood

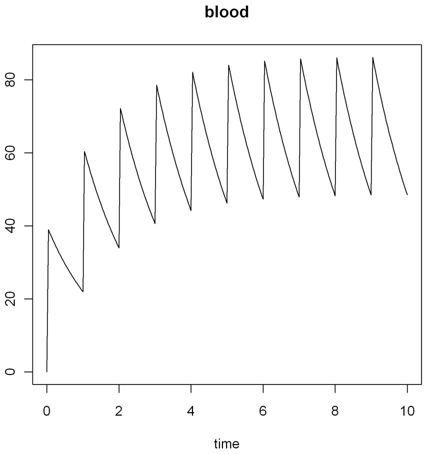
Solve model

- Pass events to the solver in a list
- All solvers in `deSolve` can handle events
- Here we use the "implicit Adams" method

```
> times <- seq(from = 0, to = 10, by = 1/24)
> outDrug <- ode(func = pharmaco, times = times, y = yini,
+              parms = NULL, method = "impAdams",
+              events = list(data = injectevents))
```

plotting model output

```
> plot(outDrug)
```



An event triggered by a root: A Bouncing Ball

Problem formulation [13]

- A ball is thrown vertically from the ground ($y(0) = 0$)
- Initial velocity (y') $= 10\ m\ s^{-1}$; acceleration $g = 9.8\ m\ s^{-2}$
- When ball hits the ground, it bounces.

ODEs describe height of the ball above the ground (y)

Specified as 2^{nd} order ODE

$$\begin{aligned} y'' &= -g \\ y(0) &= 0 \\ y'(0) &= 10 \end{aligned}$$

Specified as 1^{st} order ODE

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= -g \\ y_1(0) &= 0 \\ y_2(0) &= 10 \end{aligned}$$

A Bouncing Ball

Dynamics inbetween events

```
> library(deSolve)

> ball <- function(t, y, parms) {
+   dy1 <- y[2]
+   dy2 <- -9.8
+
+   list(c(dy1, dy2))
+ }

> yini  <- c(height = 0, velocity = 10)
```

The Ball Hits the Ground and Bounces

Root: the Ball hits the ground

- The ground is where height $= 0$
- Root function is 0 when $y_1 = 0$

```
> rootfunc <- function(t, y, parms) return (y[1])
```

Event: the Ball bounces

- The velocity changes sign (-) and is reduced by 10%
- Event function returns changed values of both state variables

```
> eventfunc <- function(t, y, parms) {
+   y[1] <- 0
+   y[2] <- -0.9*y[2]
+   return(y)
+ }
```

An event triggered by a root: the bouncing ball

Solve model

- Inform solver that event is triggered by root (`root = TRUE`)
- Pass event function to solver
- Pass root function to solver

```
> times <- seq(from = 0, to = 20, by = 0.01)
> out <- ode(times = times, y = yini, func = ball,
+              parms = NULL, rootfun = rootfunc,
+              events = list(func = eventfunc, root = TRUE))
```

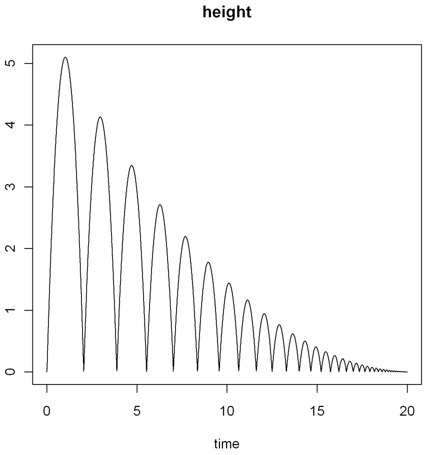
Get information about the root

```
> attributes(out)$troot
```

```
 [1]  2.040816  3.877551  5.530612  7.018367  8.357347  9.562428 10.647001 11.623117
 [9] 12.501621 13.292274 14.003862 14.644290 15.220675 15.739420 16.206290 16.626472
[17] 17.004635 17.344981 17.651291 17.926970 18.175080 18.398378 18.599345 18.780215
[25] 18.942998 19.089501 19.221353 19.340019 19.446817 19.542935 19.629441 19.707294
[33] 19.777362 19.840421 19.897174 19.948250 19.994217
```

An event triggered by a root: the bouncing ball

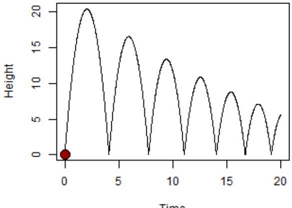
```
> plot(out, select = "height")
```



An event triggered by a root: the bouncing ball

Create Movie-like output

```
for (i in seq(1, 2001, 10)) {
  plot(out, which = "height", type = "l", lwd = 1,
       main = "", xlab = "Time", ylab = "Height"
  )
  points(t(out[i,1:2]), pch = 21, lwd = 1, col = 1, cex = 2,
         bg = rainbow(30, v = 0.6)[20-abs(out[i,3])+1])
  Sys.sleep(0.01)
}
```



Exercise: Add events to a logistic equation

Problem formulation, ODE

The logistic equation describes the growth of a population:

$$y' = r \cdot y \cdot (1 - \frac{y}{K})$$

$$r = 1, K = 10, y_0 = 2$$

Events

This population is being harvested according to several strategies:

- There is no harvesting
- Every 2 days the population's density is reduced to 50%
- When the species has reached 80% of its carrying capacity, its density is halved.

Exercise: Add events to a logistic equation - ctd

Tasks:

- Run the model for 20 days
- Implement first strategy in a `data.frame`
- Second strategy requires root and event function
- Use file `examples/logisticEvent.R.txt` as a template

Delay Differential Equations

What?

Delay Differential Equations are similar to ODEs except that they involve *past* values of variables and/or derivatives.

DDEs in R: R-package deSolve

- `dede` solves DDEs
- `lagvalue` provides lagged values of the state variables
- `lagderiv` provides lagged values of the derivatives

Example: Chaotic Production of White Blood Cells

Mackey-Glass Equation [8]:

- y: current density of white blood cells,
- y_τ is the density τ time-units in the past,
- first term equation is production rate
- b is destruction rate

$$
\begin{aligned}
y' &= a y_\tau \frac{1}{1+y_\tau^c} - by \\
y_\tau &= y(t-\tau) \\
y_t &= 0.5 \qquad \text{for } t \leq 0
\end{aligned} \tag{1}
$$

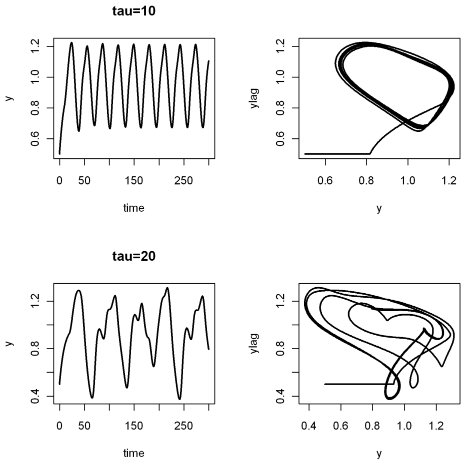
- For $\tau = 10$ the output is periodic,
- For $\tau = 20$ cell densities display a chaotic pattern

Solution in R

```
> library(deSolve)
> retarded <- function(t, y, parms, tau) {
+   tlag <- t - tau
+   if (tlag <= 0)
+     ylag <- 0.5
+   else
+     ylag <- lagvalue(tlag)
+
+   dy <- 0.2 * ylag * 1/(1+ylag^10) - 0.1 * y
+   list(dy = dy, ylag = ylag)
+ }
> yinit <- 0.5
> times <- seq(from = 0, to = 300, by = 0.1)
> yout1 <- dede(y = yinit, times = times, func = retarded, parms = NULL, tau = 10)
> yout2 <- dede(y = yinit, times = times, func = retarded, parms = NULL, tau = 20)
```

Solution in R

```
> plot(yout1, lwd = 2, main = "tau=10", ylab = "y", mfrow = c(2, 2), which = 1)
> plot(yout1[,-1], type = "l", lwd = 2, xlab = "y")
> plot(yout2, lwd = 2, main = "tau=20", ylab = "y", mfrow = NULL, which = 1)
> plot(yout2[,-1], type = "l", lwd = 2, xlab = "y")
```



Exercise: the Lemming model

A nice variant of the logistic model is the DDE lemming model [14]:

$$y' = r \cdot y(1 - \frac{y(t-\tau)}{K}) \tag{2}$$

Use file examples/ddelemming.R.txt as a template to implement this model

- initial condition $y(t=0) = 19.001$
- parameter values $r = 3.5$, $\tau = 0.74$, $K = 19$
- history $y(t) = 19$ for $t < 0$
- Generate output for t in [0, 40].

Diffusion, advection and reaction:
Partial differential equations (PDE) with ReacTran

Partial Differential Equations

PDEs as advection-diffusion problems

Many second-order PDEs can be written as advection-diffusion problems:

$$\frac{\partial C}{\partial t} = -v\frac{\partial C}{\partial x} + D\frac{\partial^2 C}{\partial x^2} + f(t, x, C)$$

same for 2-D and 3-D

Example: wave equation in 1-D

$$\frac{\partial^2 u}{\partial t^2} = c^2\frac{\partial^2 u}{\partial x^2} \tag{3}$$

can be written as:

$$\begin{aligned} \frac{du}{dt} &= v \\ \frac{\partial v}{\partial t} &= c^2\frac{\partial^2 u}{\partial x^2} \end{aligned} \tag{4}$$

Three packages for solving PDEs in R

ReacTran: methods for numerical approximation of PDEs [16]

- tran.1D(C, C.up, C.down, D, v, ...)
- tran.2D, tran.3D

deSolve: special solvers for time-varying cases [20]

- ode.1D(y, times, func, parms, nspec, dimens, method, names, ...)
- ode.2D, ode.3D

rootSolve: special solvers for time-invariant cases [19]

- steady.1D(y, time, func, parms, nspec, dimens, method, names, ...)
- steady.2D, steady.3D

Numerical solution of the wave equation

```
library(ReacTran)

wave <- function (t, y, parms) {
  u <- y[1:N]
  v <- y[(N+1):(2*N)]
  du <- v
  dv <- tran.1D(C = u, C.up = 0, C.down = 0, D = 1,
                dx = xgrid)$dC
list(c(du, dv))
}
xgrid <- setup.grid.1D(-100, 100, dx.1 = 0.2)
x     <- xgrid$x.mid
N     <- xgrid$N
uini  <- exp(-0.2*x^2)
vini  <- rep(0, N)
yini  <- c(uini, vini)
times <- seq (from = 0, to = 50, by = 1)

out <- ode.1D(yini, times, wave, parms, method = "adams",
              names = c("u", "v"), dimens = N)

image(out, grid = x)
```

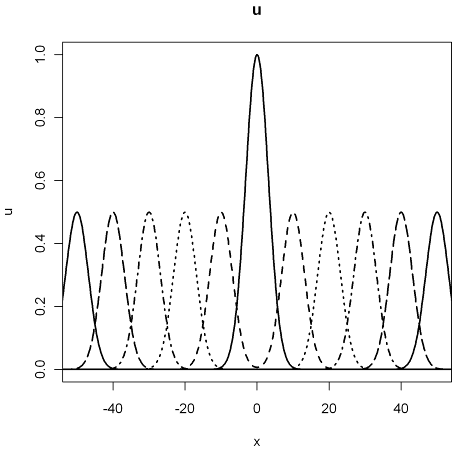
http://desolve.r-forge.r-project.org

Methods from ReacTran

Numerical method provided by the deSolve package

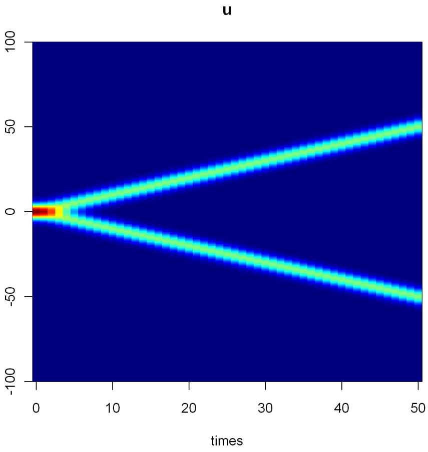
Plotting 1-D PDEs: matplot.1D

```
> outtime <- seq(from = 0, to = 50, by = 10)
> matplot.1D(out, which = "u", subset = time %in% outtime, grid = x,
+      xlab = "x", ylab = "u", type = "l", lwd = 2, xlim = c(-50, 50), col="black")
```



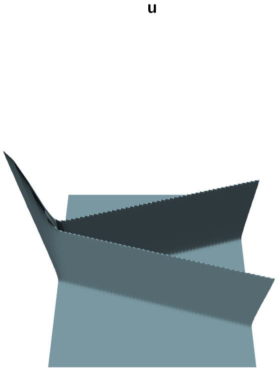
Plotting 1-D PDEs: image

```
> image(out, which = "u", grid = x)
```



Plotting 1-D PDEs: persp plots

```
> image(out, which = "u", grid = x, method = "persp", border = NA,
+        col = "lightblue", box = FALSE, shade = 0.5, theta = 0, phi = 60)
```



Exercise: the Brusselator

Problem formulation [6]

The Brusselator is a model for an auto-catalytic chemical reaction between two products, A and B, and producing also C and D in a number of intermediary steps.

$$
\begin{aligned}
A &\xrightarrow{k_1} X_1 \\
B + X_1 &\xrightarrow{k_2} X_2 + C \\
2X_1 + X_2 &\xrightarrow{k_3} 3X_1 \\
X_1 &\xrightarrow{k_4} D
\end{aligned}
$$

where the k_i are the reaction rates.

Exercise: Implement the Brusselator in 1-D

Equations for X1 and X2

$$\frac{\partial X_1}{\partial t} = D_{X_1} \frac{\partial^2 X_1}{\partial x^2} + 1 + X_1^2 X_2 - 4X_1$$

$$\frac{\partial X_2}{\partial t} = D_{X_2} \frac{\partial^2 X_2}{\partial x^2} + 3X_1 - X_1^2 X_2$$

Tasks

- The grid x extends from 0 to 1, and consists of 50 cells.
- Initial conditions:

$$X_1(0) = 1 + sin(2 * \pi * x), X_2(0) = 3$$

- Generate output for $t = 0, 1, \ldots 10$.
- Use file implementing the wave equation as a template: `examples/wave.R.txt`

2-D wave equation: Sine-Gordon

Problem formulation

The Sine-Gordon equation is a non-linear hyperbolic (wave-like) partial differential equation involving the sine of the dependent variable.

$$\frac{\partial^2 u}{\partial t^2} = D\frac{\partial^2 u}{\partial x^2} + D\frac{\partial^2 u}{\partial y^2} - \sin u \tag{5}$$

Rewritten as two first order differential equations:

$$\begin{aligned} \frac{du}{dt} &= v \\ \frac{\partial v}{\partial t} &= D\frac{\partial^2 u}{\partial x^2} + D\frac{\partial^2 u}{\partial y^2} - \sin u \end{aligned} \tag{6}$$

2-D Sine-Gordon in R

grid:

```
> Nx  <- Ny <- 100
> xgrid <- setup.grid.1D(-7, 7, N = Nx); x <- xgrid$x.mid
> ygrid <- setup.grid.1D(-7, 7, N = Ny); y <- ygrid$x.mid
```

derivative function:

```
> sinegordon2D <- function(t, C, parms) {
+    u <- matrix(nrow = Nx, ncol = Ny, data = C[1 : (Nx*Ny)])
+    v <- matrix(nrow = Nx, ncol = Ny, data = C[(Nx*Ny+1) : (2*Nx*Ny)])
+    dv <- tran.2D (C = u, C.x.up = 0, C.x.down = 0, C.y.up = 0, C.y.down = 0,
+                   D.x = 1, D.y = 1, dx = xgrid, dy = ygrid)$dC  -   sin(u)
+    list(c(v, dv))
+ }
```

initial conditions:

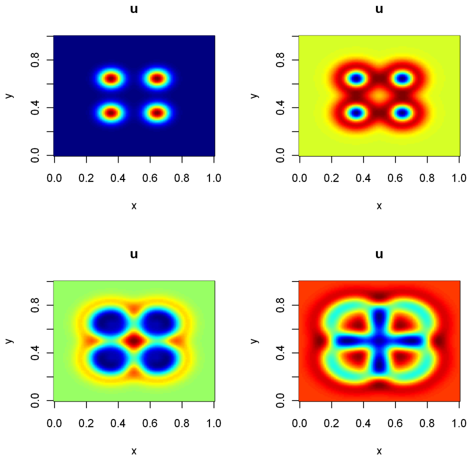
```
> peak <- function (x, y, x0, y0)  return(exp(-( (x-x0)^2 + (y-y0)^2)))
> uini <- outer(x, y, FUN = function(x, y) peak(x, y, 2,2) + peak(x, y,-2,-2)
+                                        + peak(x, y,-2,2) + peak(x, y, 2,-2))
> vini <- rep(0, Nx*Ny)
```

solution:

```
> out  <- ode.2D (y = c(uini,vini), times = 0:3, parms = 0, func = sinegordon2D,
+          names = c("u", "v"), dimens = c(Nx, Ny), method = "ode45")
```

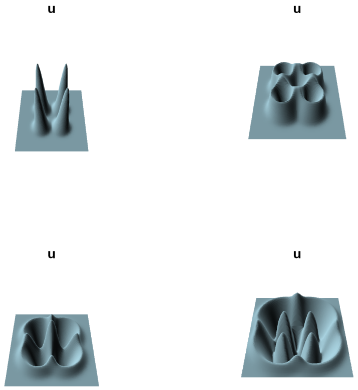
Plotting 2-D PDEs: image plots

```
> image(out, which = "u", grid = list(x, y), mfrow = c(2,2), ask = FALSE)
```



Plotting 2-D PDEs: persp plots

```
> image(out, which = "u", grid = list(x, y), method = "persp", border = NA,
+       col = "lightblue", box = FALSE, shade = 0.5, theta = 0, phi = 60,
+       mfrow = c(2,2), ask = FALSE)
```



Movie-like output of 2-D PDEs

```
out  <- ode.2D (y = c(uini, vini), times = seq(0, 3, by = 0.1),
                parms = NULL, func = sinegordon2D,
                names=c("u", "v"), dimens = c(Nx, Ny),
                method = "ode45")
image(out, which = "u", grid = list(x = x, y = y),
     method = "persp", border = NA,
     theta = 30, phi = 60, box = FALSE, ask = FALSE)
```

Exercise: Implement the Brusselator in 2-D

Equations

$$\begin{aligned}
\frac{\partial X_1}{\partial t} &= D_{X_1}\frac{\partial^2 X_1}{\partial x^2} + D_{X_1}\frac{\partial^2 X_1}{\partial y^2} + 1 + X_1^2 X_2 - 4X_1 \\
\frac{\partial X_2}{\partial t} &= D_{X_2}\frac{\partial^2 X_1}{\partial x^2} + D_{X_2}\frac{\partial^2 X_1}{\partial y^2} + 3X_1 - X_1^2 X_2
\end{aligned}$$

Tasks

- The grids x and y extend from 0 to 1, and consist of 50 cells.
- Parameter settings: diffusion coefficient:

$$D_{X_1} = 2; D_{X_2} = 8 * D_{X_1}$$

- Initial condition for X_1, X_2: random numbers inbetween 0 and 1.
- Generate output for t = 0, 1, . . . 8
- Use the file implementing the Sine-Gordon equation as a template: `examples/sinegordon.R.txt`

Speeding up: Matrices and compiled code

Methods for speeding up

- Use matrices,
- Implement essential parts in compiled code (Fortran, C),
- Implement the full method in compiled code.

Formulating a model with matrices and vectors can lead to a considerable speed gain – and compact code – while retaining the full flexibility of R. The use of compiled code saves even more CPU time at the cost of a higher development effort.

Use of matrices

A Lotka-Volterra model with 4 species

```
>   model <- function(t, n, parms) {
+     with(as.list(c(n, parms)), {
+       dn1 <- r1 * n1 - a13 * n1 * n3
+       dn2 <- r2 * n2 - a24 * n2 * n4
+       dn3 <- a13 * n1 * n3 - r3 * n3
+       dn4 <- a24 * n2 * n4 - r4 * n4
+     return(list(c(dn1, dn2, dn3, dn4)))
+   })
+ }
> parms <- c(r1 = 0.1, r2 = 0.1, r3 = 0.1, r4 = 0.1, a13 = 0.2, a24 = 0.1)
> times = seq(from = 0, to = 500, by = 0.1)
> n0  = c(n1 = 1, n2 = 1, n3 = 2, n4 = 2)

> system.time(out <- ode(n0, times, model, parms))

   user  system elapsed
   3.02    0.00    3.02
```

Source: examples/lv-plain-or-matrix.R.txt

Use of matrices

A Lotka-Volterra model with 4 species

```
> model <- function(t, n, parms) {
+   with(parms, {
+     dn <- r * n  + n * (A %*% n)
+     return(list(c(dn)))
+   })
+ }
> parms <- list(
+   r = c(r1 = 0.1, r2 = 0.1, r3 = -0.1, r4 = -0.1),
+   A = matrix(c(0.0, 0.0, -0.2, 0.0,      # prey 1
+                0.0, 0.0, 0.0, -0.1,      # prey 2
+                0.2, 0.0, 0.0, 0.0,       # predator 1; eats prey 1
+                0.0, 0.1, 0.0, 0.0),      # predator 2; eats prey 2
+                nrow = 4, ncol = 4, byrow = TRUE)
+ )
```

```
> system.time(out <- ode(n0, times, model, parms))
```

```
   user  system elapsed
   1.66    0.00    1.70
```

Source: examples/lv-plain-or-matrix.R.txt

Results

- `plot(out)` will show the results.
- Note that the "plain" version has only 1 to 1 connections, but the matrix model is already full connected (with most connections are zero). The comparison is insofar unfair that the matrix version (despite faster execution) is more powerful.
- Exercise: Create a fully connected model in the plain version for a fair comparison.
- A parameter example (e.g. for weak coupling) can be found on: http://tolstoy.newcastle.edu.au/R/e7/help/09/06/1230.html

Using compiled code

All solvers of deSolve

- allow direct communication between solvers and a compiled model.

See `vignette ("compiledCode")` [15]

Principle

- Implement core model (and only this) in C or Fortran,
- Use data handling, storage and plotting facilities of R.

`examples/compiled_lorenz/compiledcode.svg`

The End

Thank you!

More Info:

`http://desolve.r-forge.r-project.org`

Acknowledgments

Citation

A lot of effort went in creating this software; please cite it when using it.

- to cite deSolve: [20], rootSolve [19], ReacTran [16],
- Some complex examples can be found in [18],
- A framework to fit differential equation models to data is FME [17],
- A framework for ecological modelling is simecol [10].

Acknowledgments

- None of this would be possible without the splendid work of the R Core Team [11],
- This presentation was created with Sweave [7],
- Creation of the packages made use of Rforge [21].

Bibliography I

[1] K E Brenan, S L Campbell, and L R Petzold.
Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.
SIAM Classics in Applied Mathematics, 1996.

[2] P N Brown, G D Byrne, and A C Hindmarsh.
Vode, a variable-coefficient ode solver.
SIAM Journal on Scientific and Statistical Computing, 10:1038–1051, 1989.

[3] E Hairer, S. P. Norsett, and G Wanner.
Solving Ordinary Differential Equations I: Nonstiff Problems. Second Revised Edition.
Springer-Verlag, Heidelberg, 2009.

[4] E Hairer and G Wanner.
Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems. Second Revised Edition.
Springer-Verlag, Heidelberg, 2010.

[5] A. C. Hindmarsh.
ODEPACK, a systematized collection of ODE solvers.
In R. Stepleman, editor, *Scientific Computing, Vol. 1 of IMACS Transactions on Scientific Computation*, pages 55–64. IMACS / North-Holland, Amsterdam, 1983.

[6] R. Lefever, G. Nicolis, and I. Prigogine.
On the occurrence of oscillations around the steady state in systems of chemical reactions far from equilibrium.
Journal of Chemical Physics, 47:1045–1047, 1967.

[7] Friedrich Leisch.
Dynamic generation of statistical reports using literate data analysis.
In W. Härdle and B. Rönz, editors, *COMPSTAT 2002 – Proceedings in Computational Statistics*, pages 575–580, Heidelberg, 2002. Physica-Verlag.

[8] M. C. Mackey and L. Glass.
Oscillation and chaos in physiological control systems.
Science, 197:287–289, 1977.

Bibliography II

[9] Linda R. Petzold.
Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations.
SIAM Journal on Scientific and Statistical Computing, 4:136–148, 1983.

[10] Thomas Petzoldt and Karsten Rinke.
simecol: An object-oriented framework for ecological modeling in R.
Journal of Statistical Software, 22(9):1–31, 2007.

[11] R Development Core Team.
R: A Language and Environment for Statistical Computing.
R Foundation for Statistical Computing, Vienna, Austria, 2011.
ISBN 3-900051-07-0.

[12] O.E. Rossler.
An equation for continous chaos.
Physics Letters A, 57 (5):397–398, 1976.

[13] L. F. Shampine, I. Gladwell, and S. Thompson.
Solving ODEs with MATLAB.
Cambridge University Press, Cambridge, 2003.

[14] L.F Shampine and S. Thompson.
Solving ddes in matlab.
App. Numer. Math., 37:441–458, 2001.

[15] K Soetaert, T Petzoldt, and RW Setzer.
R-package deSolve, Writing Code in Compiled Languages, 2009.
package vignette.

[16] Karline Soetaert and Filip Meysman.
Reactive transport in aquatic ecosystems: rapid model prototyping in the open source software R.
Environmental modelling and software, page in press, 2011.

Bibliography III

[17] Karline Soetaert and Thomas Petzoldt.
Inverse modelling, sensitivity and monte carlo analysis in R using package FME.
Journal of Statistical Software, 33(3):1–28, 2010.

[18] Karline Soetaert and Thomas Petzoldt.
Solving ODEs, DAEs, DDEs and PDEs in R.
Journal of Numerical Analysis, Industrial and Applied Mathematics, in press, 2011.

[19] Karline Soetaert, Thomas Petzoldt, and R. Woodrow Setzer.
Solving Differential Equations in R.
The R Journal, 2(2):5–15, December 2010.

[20] Karline Soetaert, Thomas Petzoldt, and R. Woodrow Setzer.
Solving differential equations in R: Package deSolve.
Journal of Statistical Software, 33(9):1–25, 2010.

[21] Stefan Theußl and Achim Zeileis.
Collaborative Software Development Using R-Forge.
The R Journal, 1(1):9–14, May 2009.

[22] B. van der Pol and J. van der Mark.
Frequency demultiplication.
Nature, 120:363–364, 1927.