15
Non Linear Models

Non linear models cover a very vast area of science, including exponential, logistic and many other models.

15.1 Least squares regression for non linear models

The problem of minimizing a non−linear function over a space of parameters of the function, is usually referred as nonlinear fitting. These problems arise commonly in least−squares curve fitting and nonlinear programming. The problem is formulated as

$$\Phi(\beta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta)]^2 \tag{15.1}$$

where there are m empirical data pairs of (x_i, y_i). The objective function $\Phi(\beta)$ is minimized by optimizing the parameters β of the model $f(x, \beta)$, so that the sum of the squares of the deviations is minimized.

The Levenberg−Marquardt algorithm (**?**) is an iterative procedure where the user must provide an initial guess of the parameter vector β. Then, at each iteration, the parameter vector β is replaced by a new approximation $\beta + \delta$. This new parameter set will lead to a new sum of squares. The basis of the algorithm is a linear approximation of the function $f(x, \beta + \delta)$ in the neighborhood of β. Denoting J as the Jacobian matrix, a Taylor series expansion for small δ leads to

$$f(x, \beta + \delta) \approx f(x, \beta) + J_i \delta \tag{15.2}$$

where

$$J_i = \frac{\partial f(x_i, \beta)}{\partial \beta} \tag{15.3}$$

where J_i is the gradient of f with respect to β. The new objective function is now:

$$\Phi(\beta + \delta) = \sum_{i=1}^{m} [y - f(x_i, \beta) - J_i \delta]^2 \tag{15.4}$$

Therefore the sought δ that minimizes this new objective function, is the solution of the least−squares problem. The iteration ends when the function Φ is less than a threshold for the error. One of the limitation of the Levenberg−Marquardt algorithm is that the solution for the best fit parameters, may depend on the initial conditions of the parameters set, therefore the solution may not be unique. The problem of minimizing a non−linear function over a space of parameters of the function, is usually referred as nonlinear fitting. These problems arise commonly in least−squares curve fitting and nonlinear programming. The problem is formulated as

$$\Phi(\beta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta)]^2 \tag{15.5}$$

where there are m empirical data pairs of (x_i, y_i). The objective function $\Phi(\beta)$ is minimized by optimizing the parameters β of the model $f(x, \beta)$, so that the sum of the squares of the deviations is minimized.

The Levenberg−Marquardt algorithm (**?**) is an iterative procedure where the user must provide an initial guess of the parameter vector β. Then, at each iteration, the parameter vector β is replaced by a new approximation $\beta + \delta$. This new parameter set will lead to a new sum of squares. The basis of the algorithm is a linear approximation of the function $f(x, \beta + \delta)$ in the neighborhood of β. Denoting J as the Jacobian matrix, a Taylor series expansion for small δ leads to

$$f(x, \beta + \delta) \approx f(x, \beta) + J_i \delta \tag{15.6}$$

where

$$J_i = \frac{\partial f(x_i, \beta)}{\partial \beta} \tag{15.7}$$

The Jacobian matrix is shown below for three parameters

$$\begin{pmatrix} \frac{\partial J_1}{\partial \beta_1} & \frac{\partial J_1}{\partial \beta_2} & \frac{\partial J_1}{\partial \beta_3} \\ \frac{\partial J_2}{\partial \beta_1} & \frac{\partial J_2}{\partial \beta_2} & \frac{\partial J_2}{\partial \beta_3} \\ \frac{\partial J_3}{\partial \beta_1} & \frac{\partial J_3}{\partial \beta_2} & \frac{\partial J_3}{\partial \beta_3} \end{pmatrix} \tag{15.8}$$

where J_i is the gradient of f with respect to β. The new objective function is now:

$$\Phi(\beta + \delta) = \sum_{i=1}^{m} [y - f(x_i, \beta) - J_i \delta]^2 \tag{15.9}$$

Therefore the sought δ that minimizes this new objective function, is the solution of the least−squares problem. The iteration ends when the function Φ is less than a threshold for the error. One of the limitation of the Levenberg−Marquardt algorithm is that the solution for the best fit parameters, may depend on the initial conditions of the parameters set, therefore the solution may not be unique.

15.2 Applications in R

The implementation of the Levenberg−Marquardt Nonlinear Least−Squares Algorithm is in the library *minpack.lm*. The function `nls.lm` is used to perform the non−linear fitting. The function `nls.lm(par, lower=NULL, upper=NULL, fn, jac = NULL, control = nls.lm.control(), ...)` has the following arguments:

- `par` A list or numeric vector of starting estimates. If par is a list, then each element must be of length 1.
- `lower` A numeric vector of lower bounds on each parameter. If not given, the default lower bound for each parameter is set to -Inf.
- `upper` A numeric vector of upper bounds on each parameter. If not given, the default upper bound for each parameter is set to Inf.
- `fn` A function that returns a vector of residuals, the sum square of which is to be minimized. The first argument of `fn` must be par.
- `jac` A function to return the Jacobian for the `fn` function.

The example below in code `Ch12_1_1` was presented as an example in the **R** documentation for the package *minpack.lm*.

```
#Code Ch12_1_1
library(minpack.lm)
## values over which to simulate data
x <- seq(0,5,length=100)
## model based on a list of parameters
getPred <- function(parS, xx) parS$a * exp(xx * parS$b) + parS$c
## parameter values used to simulate data
pp <- list(a=9,b=-1, c=6)
## simulated data, with noise
simDNoisy <- getPred(pp,x) + rnorm(length(x),sd=.1)

## plot data
plot(x,simDNoisy, main="data")

## residual function
residFun <- function(p, observed, xx) observed - getPred(p,xx)

## starting values for parameters
parStart <- list(a=3,b=-.001, c=1)

## perform fit
nls.out <- nls.lm(par=parStart, fn = residFun, observed = simDNoisy,
xx = x, control = nls.lm.control(nprint=1))

## plot model evaluated at final parameter estimates
lines(x,getPred(as.list(coef(nls.out)), x), col=2, lwd=2)

# summary information on parameter estimates
summary(nls.out)
```

A modification of the code above is presented here, where the data are not generated, by read from a file for a simple exponential function. The output is presented below with the significance on the right end of the parameter.

```
Parameters:
Estimate Std. Error t value Pr(>|t|)
a 9.10027 0.04628 196.62 <2e-16 ***
b -0.99743 0.01112 -89.66 <2e-16 ***
c 5.97631 0.02097 285.02 <2e-16 ***

Residual standard error: 0.1078 on 97 degrees of freedom
Number of iterations to termination: 8
```

Figure 15.1 depicts the results of the fitting procedure, with the generated data (points) and the fitted line.

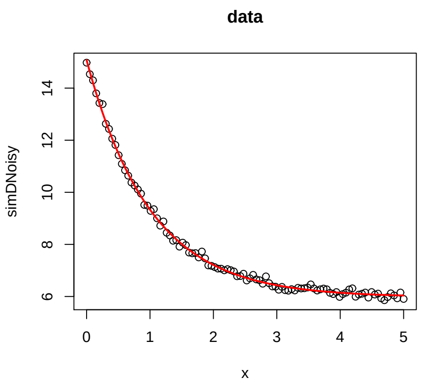

Fig. 15.1 Example of non−linear fitting for an exponential decay

```
#Code Ch12_1_1
library(minpack.lm)
setwd("~/Didattica/R_class_4/exercises/Ch13_nonlinear_regression")

exp_df <- read.table(file = "expon.dat", header = TRUE,sep = "\t")
exp_df
attach(exp_df)

## model based on a list of parameters
getPred <- function(parS, xx) parS$a * exp(xx * parS$b)
## plot data
plot(obs_x,obs_y, main="data")

## residual function
residFun <- function(p, observed, xx) observed - getPred(p,xx)
```

```
## starting values for parameters
parStart <- list(a=2,b=2)

## perform fit
nls.out <- nls.lm(par=parStart, fn = residFun, observed = obs_y,
xx = obs_x, control = nls.lm.control(nprint=1))

## plot model evaluated at final parameter estimates
lines(obs_x,getPred(as.list(coef(nls.out)), obs_x), col=2, lwd=2)

# summary information on parameter estimates
summary(nls.out)
```

Figure 15.2 depicts the results of the fitting procedure, with the generated data (points) and the fitted line.

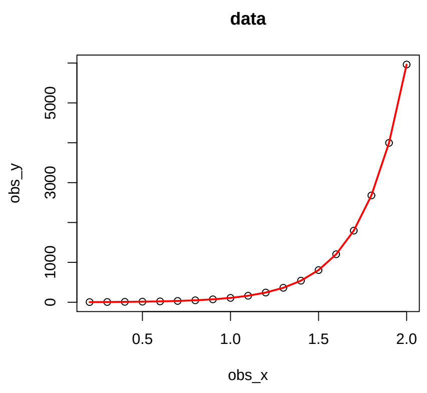

Fig. 15.2 Example of non−linear fitting for an exponential function

The code below is also an example presented in the **R** documentation for the package *minpack.lm.* In the example below the derivatives of the function `D` with respective to named parameters are explicitely computed. This computation also performed outside the function are useful to check if the function is differentiable with respect to the parameters and therefore the Jacobian matrix can be successfully created and therefore the algorithm will successfully converge to a solution.

```
#Code Ch12_1_4
library(minpack.lm)

#==============GENERATE DATA===================================
#define function
f <- function(T, tau, N0, a, f0)
{
       expr <- expression(N0 * exp(-T/tau) * (1 + a*cos(f0*T)))
       eval(expr)
}
#Values over which to simulate data: 500 intervals between 0 and 8
T <- seq(0, 8, length=501)
#Parameter values underlying simulated data
p <- c(tau = 2.2, N0 = 1000, a = 0.25, f0 = 8)
#Define vector with simulated data to fit
Ndet <- do.call("f", c(list(T = T), as.list(p)))
#Add noise. Set random number seed so example is reproducible.
set.seed(17)
N <- Ndet + rnorm(length(Ndet), mean=Ndet, sd=.01*max(Ndet))
head(N)
plot(N)
#===============================================================
#================FIT EQUATION TO DATA==========================

#Function D computes derivatives with respective to named parameter.
j <- function(T, tau, N0, a, f0)
{
       expr <- expression(N0*exp(-T/tau)*(1 + a*cos(f0*T)))
       c(eval(D(expr, "tau")),
       eval(D(expr, "N0" )),
       eval(D(expr, "a" )),
       eval(D(expr, "f0" )))
}

#Define a residual function
fcn <- function(p, T, N, fcall, jcall)
(N - do.call("fcall", c(list(T = T), as.list(p))))

#Define analytical expression for the gradient
fcn.jac <- function(p, T, N, fcall, jcall)
{
       -do.call("jcall", c(list(T = T), as.list(p)))
}

#Initial guess starting values
guess <- c(tau = 2.2, N0 = 1500, a = 0.25, f0 = 10)

#Non-linear curve fit
out <- nls.lm(par = guess, fn = fcn, jac = fcn.jac,
fcall = f, jcall = j,
T = T, N = N, control = nls.lm.control(nprint=1))
#get the fitted values
```

```
N1 <- do.call("f", c(list(T = T), out$par))

#Plots
plot(T, N, cex = 0.5, main="Simulated Data to Fit")
grid()
lines(T, N1, col="blue", lwd=2) # Fitted solution
```

Figure 15.3 depicts the results of the fitting procedure, with the generated data (points) and the fitted line.

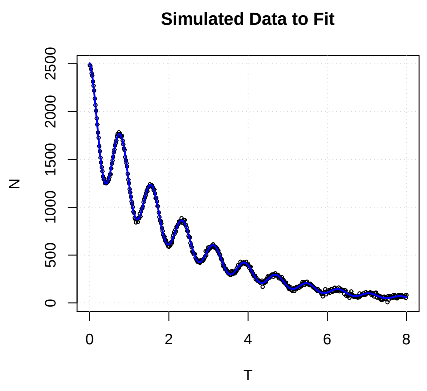

Fig. 15.3 Example of non−linear fitting for a periodic exponential decay

In the next example, experimental data for a soil water retention curve (SWR) are described by using the (**?**) equation, which has the following form:

$$S_e(\psi) = \frac{\theta - \theta_r}{\theta_s - \theta_r} = \left(\frac{1}{[1 + (\alpha\psi)^n]^m}\right) \tag{15.10}$$

which solving for θ can be written as:

$$\theta = \theta_r + (\theta_s - \theta_r)\left(\frac{1}{[1 + (\alpha\psi)^n]^m}\right) \tag{15.11}$$

where S_e is the degree of saturation $[0, 1]$ and α, n, m, θ_s θ_r are fitting parameters. Different restrictions can be imposed on the parameters n and m depending on the shape of the curve. In particular, when only a limited range of water retention values are available (usually in the 'wet' range of the curve) it might be necessary to restrict the parameters n and m. More stable results are generally obtained when the restriction $m = 1 - 1/n$ is implemented for incomplete data sets.

The code is shown below.

```
#Code Ch12_1_5
library(minpack.lm)
```

```
setwd("~/Didattica/R_class_4/exercises/Ch13_nonlinear_regression")

WRC_df <- read.table(file = "clay.txt", header = TRUE,sep = "\t")
WRC_df
attach(WRC_df)
plot(wp,wc)
wp_log<-log(wp, base = exp(1))
plot(wp_log,wc,ylim=c(0,0.45))

#detach(WRC_df)
## model based on a list of parameters
## Definition of the van Genuchten equation
##theta_r,theta_s,alpha,n,m (list of parameters)

getPred <- function(parS, xx) parS$theta_r+(parS$theta_s-parS$theta_r)*
(1/(1+(wp_log*parS$alpha)^parS$n)^parS$m)
## plot data


## residual function
residFun <- function(p, observed, xx) observed - getPred(p,xx)

## starting values for parameters
parStart <- list(theta_r=0.01,theta_s=0.5,alpha=0.5,n=1,m=1)
lower_limit <- c(theta_r=0.01,theta_s=0.01, alpha=0.01,n=0.01,m=0.01)
upper_limit <- c(theta_r=0.1,theta_s=0.4,alpha=2,n=3,m=3)

## perform fit
nls.out <- nls.lm(par=parStart,lower=lower_limit,upper= upper_limit,
fn = residFun, observed = wc, xx = wp_log, control = nls.lm.control(nprint=1))

## plot model evaluated at final parameter estimates
lines(wp_log,getPred(as.list(coef(nls.out)), wp_log), ylim=c(0,0.45),col=2, lwd=2)

# summary information on parameter estimates
summary(nls.out)
```

The results of the fitting procedure is shown in Figure

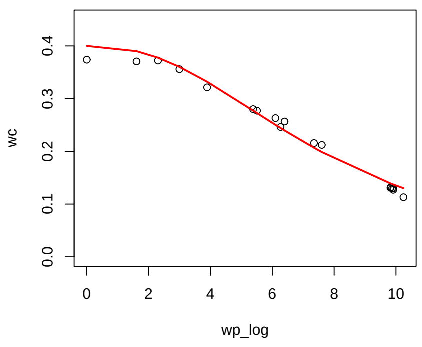

Fig. 15.4 Example of non−linear fitting for the van Genuchten (1980) equation.