Ilustrating the estimation of the parameters of a non-linear model

Tiago A. Marques

November 21, 2018

Introduction

In this report we look at the use of `nls` to estimate the parameter values of a non-linear function. We then compare the approach with what would be the results using LMs, GLMs or GAMs. To use the GAMs we load package `mgcv`.

```
#load required library
library(mgcv)
```

```
## Loading required package: nlme
```

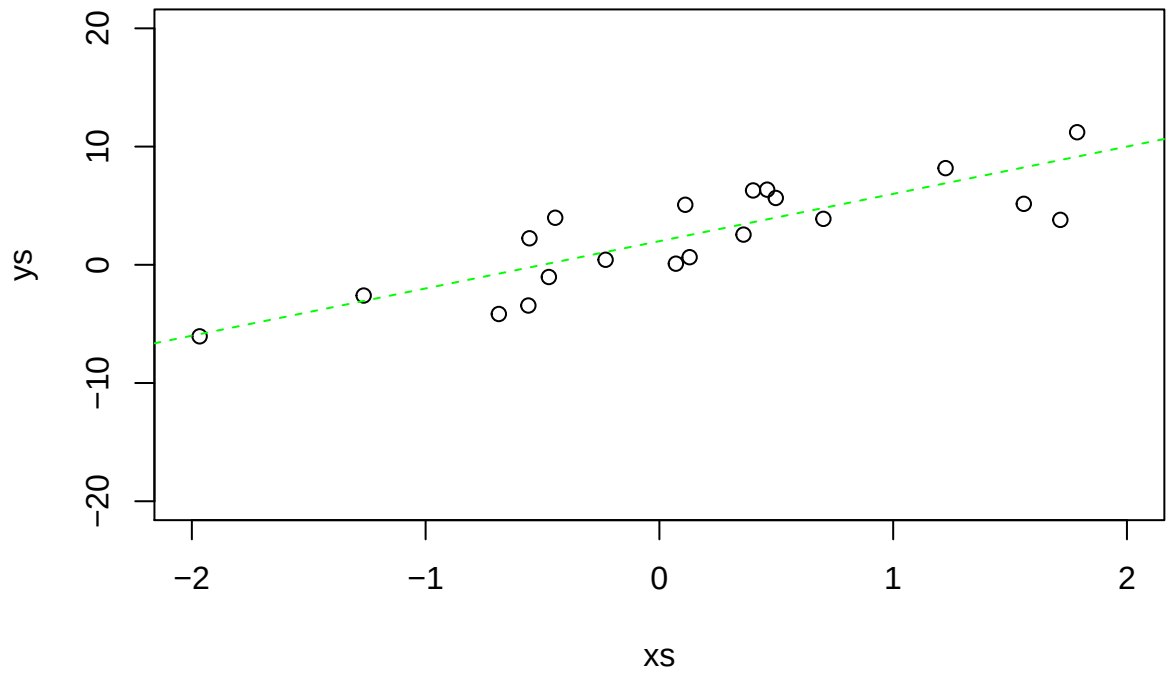
```
## This is mgcv 1.8-24. For overview type 'help("mgcv-package")'.
```

Comparing the nls with lm

Consider a linear model. We have learnt to estimate the parameter values for that model using function `lm`. But the truth is that we could have used a model fitting routine that is general enough such that we can specify the formula for our model, including a non-linear formula.

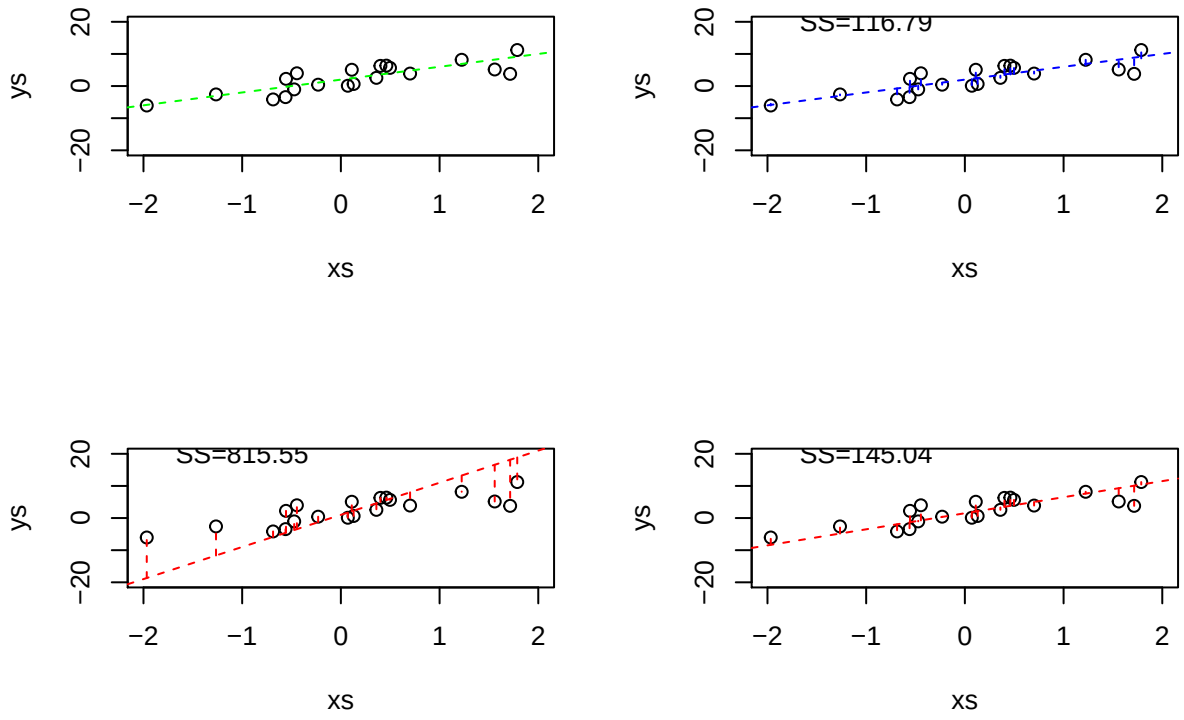
Lets start by considering the linear case. Here's some simulated data from a linear model.

```
set.seed(123)
xs=rnorm(20)
ys=2+4*xs+rnorm(20,sd=3)
par(mfrow=c(1,1))
plot(xs,ys,ylim=c(-20,20),xlim=c(-2,2))
abline(2,4,col="green",lty=2)
```



Note that in general we could calculate the distance between the observaed values and the predicted values (and)given our model), for any set of parameters. Assume a comparison between the true parameter values (green line below), those estimated by `lm` (blue line below) and two other sets of values (red lines below)

```
set.seed(123)
xs=rnorm(20)
ys=2+4*xs+rnorm(20,sd=3)
par(mfrow=c(2,2))
plot(xs,ys,ylim=c(-20,20),xlim=c(-2,2))
abline(2,4,col="green",lty=2)
#estimated by lm
plot(xs,ys,ylim=c(-20,20),xlim=c(-2,2))
abline(2,4,col="blue",lty=2)
yspreds=predict(lm(ys~xs))
segments(x0=xs,y0=yspreds,x1=xs,y1=ys,lty=2,col="blue")
text(-1,20,paste0("SS=",round(sum((ys-yspreds)^2),2)))
#bad 1
plot(xs,ys,ylim=c(-20,20),xlim=c(-2,2))
abline(1,10,col="red",lty=2)
segments(x0=xs,y0=1+10*xs,x1=xs,y1=ys,lty=2,col="red")
text(-1,20,paste0("SS=",round(sum((ys-(1+10*xs))^2),2)))
#bad 2
plot(xs,ys,ylim=c(-20,20),xlim=c(-2,2))
abline(1.5,5,col="red",lty=2)
segments(x0=xs,y0=1.5+5*xs,x1=xs,y1=ys,lty=2,col="red")
text(-1,20,paste0("SS=",round(sum((ys-(1.5+5*xs))^2),2)))
```



When the true model is a line, in practice, we can't destinguish the estimates from the `lm` and the corresponding `nls` estimates

```
set.seed(123)
xs=rnorm(20)
ys=2+4*xs+rnorm(20,sd=3)
plot(xs,ys)
abline(2,4,col="green",lty=2)
lm1=lm(ys~xs)
abline(lm1,col="red",lty=3)
summary(lm1)
```

```
##
## Call:
## lm(formula = ys ~ xs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5369 -1.8056 -0.5572  2.4926  3.7773
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.8795     0.5759   3.264  0.00431 **
## xs            3.7652     0.6008   6.267 6.55e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 2.547 on 18 degrees of freedom
## Multiple R-squared:  0.6857, Adjusted R-squared:  0.6683
## F-statistic: 39.27 on 1 and 18 DF,  p-value: 6.554e-06
```

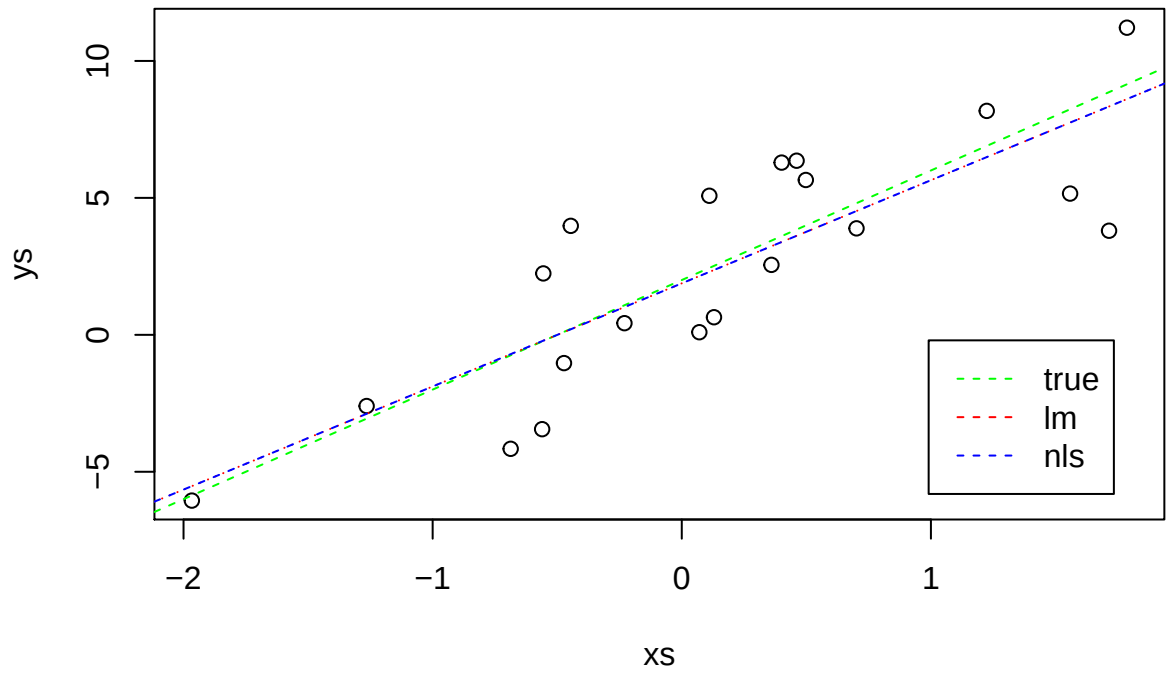
```
#Now, using nls for the estimation
#fitting the TRUE model using nls
nlslm1=nls(y~a+b*x,data=data.frame(y=ys,x=xs),start=list(a=3,b=5))
snlslm1=summary(nlslm1)
nlslm1
```

```
## Nonlinear regression model
##   model: y ~ a + b * x
##    data: data.frame(y = ys, x = xs)
##     a     b
## 1.879 3.765
##  residual sum-of-squares: 116.8
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 5.245e-09
```

```
coef(nlslm1)
```

```
##        a        b
## 1.879480 3.765209
```

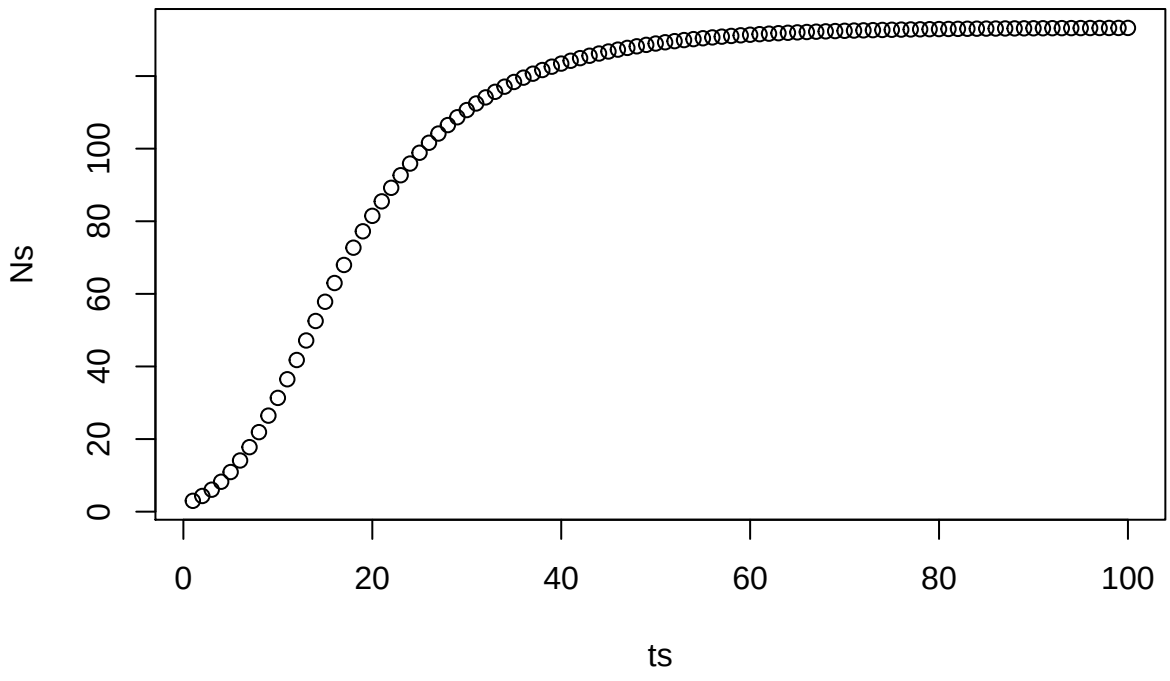
```
abline(coef(nlslm1)[1],coef(nlslm1)[2],col="blue",lty=2)
legend("bottomright",inset=0.05,legend=c("true","lm","nls"),
       lwd=1,col=c("green","red","blue"),lty=2)
```



Considering a non-linear function

We are going to use as an example the logistic function, which is a function of 3 parameters, `N0`, `K` and `r`. `N0` is also known as the initial population size, `K` the carrying capacity and `r` the growth rate. More details about this function and its relation to other models to describe population growth are available here (https://en.wikipedia.org/wiki/Beverton%E2%80%93Holt_model). We can plot the function below, for some arbitrarily chosen values of the parameters (`N0`=2,`K`=200,`r`=0.08, with `t` from 1 to 100).

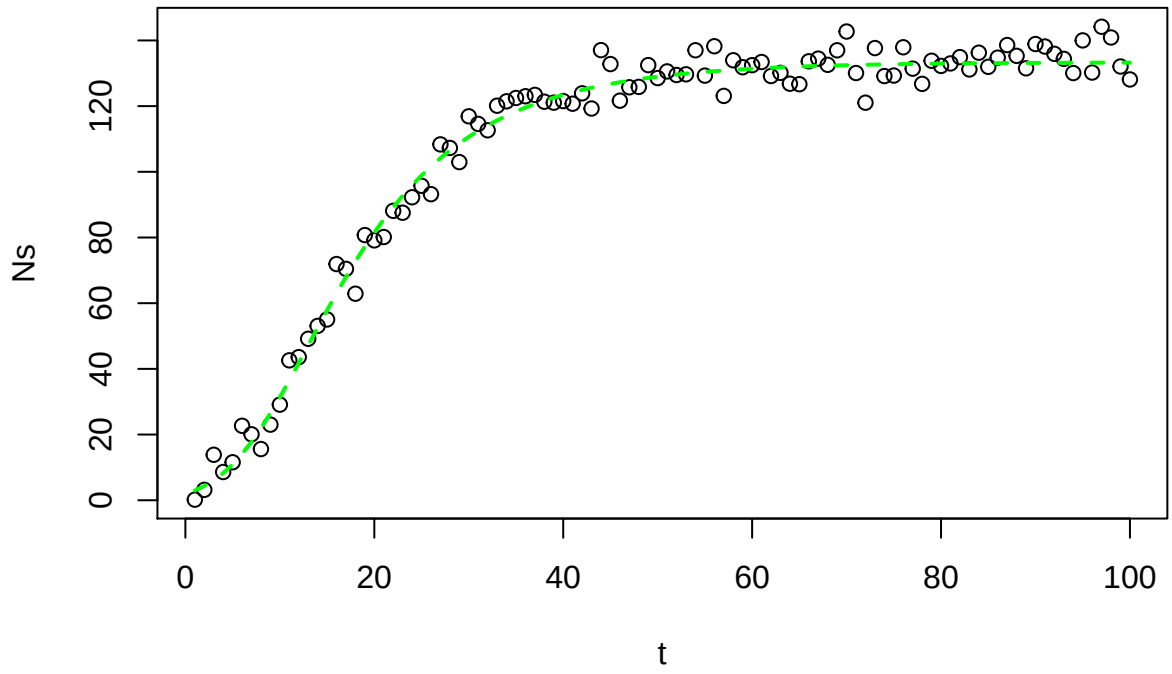
```
getNt=function(t,N0,K,r){
  #logistic model
  # see e.g.
  #
  N=(K*N0)/(N0+(K-N0)^exp(-r*t))
return(N)
}

#plot true model
ts=1:100
Ns=getNt(t=ts,N0=2,K=200,r=0.08)
plot(ts,Ns)
```



Now, we can actually generate data from that model, adding a bit of noise to it to make it realistically. Here we use Gaussian noise. Then we plot the data, with the true data generating model overlaid.

```
#generate data with noise
set.seed(123)
Ns=getNt(t=ts,N0=2,K=200,r=0.08)+rnorm(100,mean=0,sd=5)
```

```
t=1:100;plot(t,Ns)
lines(t,getNt(t,N0=2,K=200,r=0.08),col="green",lwd=2,lty=2)
```



As an aside, we output this file for students to work on it.

```
#write a file for FT11
write.table(file="data4FT11.txt",data.frame(Ts=ts,Ns=Ns),row.names=FALSE)
```

Finally, we can use function `nls` to estimate the parameter values

```
#fitting the TRUE model using nls
nls1=nls(Ns~(K*N0)/(N0+(K-N0)^exp(-r*t)),data=data.frame(Ns=Ns,t=t),start=list(N0=3,K=180,r=0.05))
snls1=summary(nls1)
snls1
```

```
##
## Formula: Ns ~ (K * N0)/(N0 + (K - N0)^exp(-r * t))
##
## Parameters:
##     Estimate Std. Error t value Pr(>|t|)
## N0 2.520e+00  5.343e-01   4.717 8.03e-06 ***
## K  1.879e+02  1.081e+01  17.381  < 2e-16 ***
## r  7.338e-02  5.364e-03  13.682  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.572 on 97 degrees of freedom
##
```

```
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 4.954e-06
```

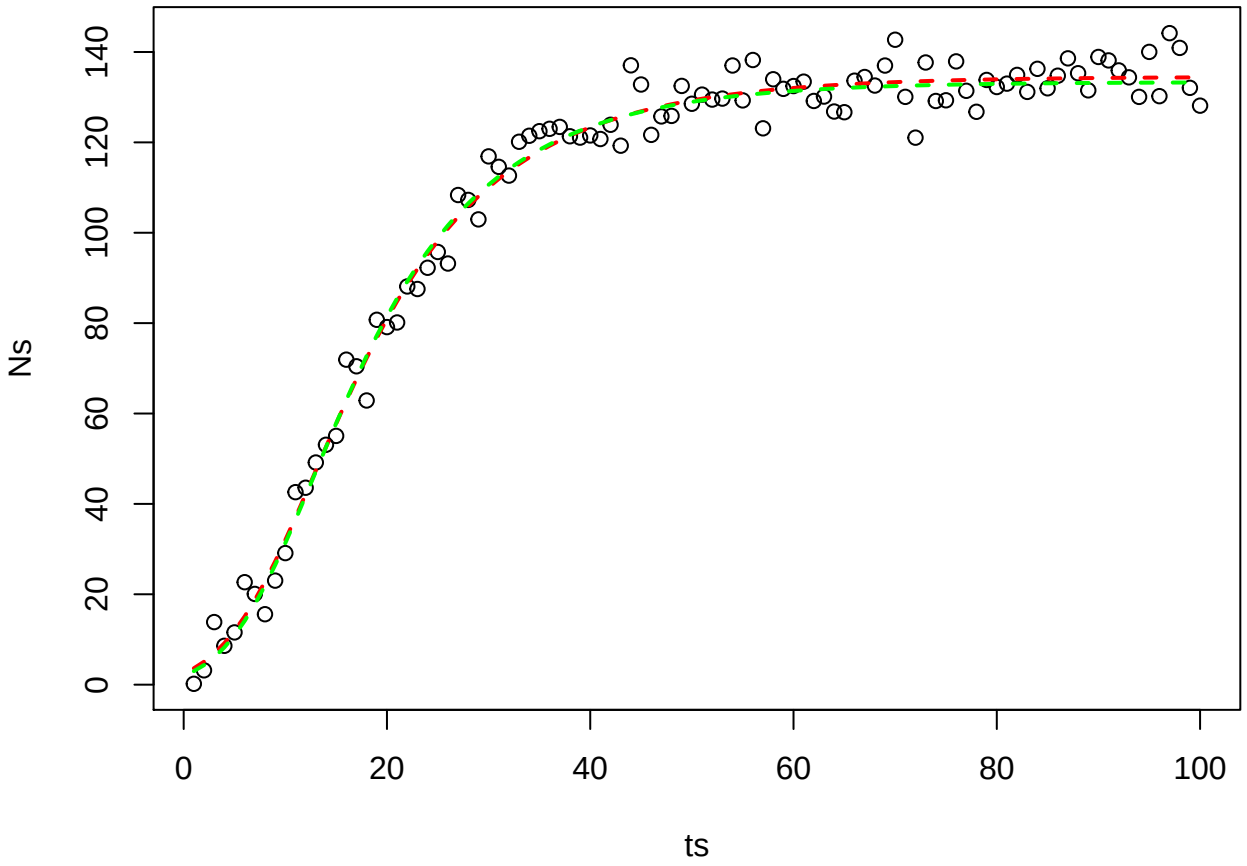
```
coef(snls1)
```

```
##        Estimate   Std. Error   t value     Pr(>|t|)
## N0   2.52014793  0.534256001  4.717117 8.027388e-06
## K  187.88742284 10.810088596 17.380748 1.489282e-31
## r    0.07338263  0.005363502 13.681849 2.259608e-24
```

We estimate `N0` to be 2.52, and `K` to be 187.887 and r to be 0.073, which is not bad given the initial values (`N0`=2,`K`=200,`r`=0.08).

We can see what that corresponds to by comparing the true model with the estimated model

```
par(mfrow=c(1,1),mar=c(4,4,0.2,0.2))
plot(ts,Ns)
lines(t,getNt(t,N0=coef(nls1)[1],K=coef(nls1)[2],r=coef(nls1)[3]),col="red",lwd=2,lty=2)
lines(t,getNt(t,N0=2,K=200,r=0.08),col="green",lwd=2,lty=2)
```



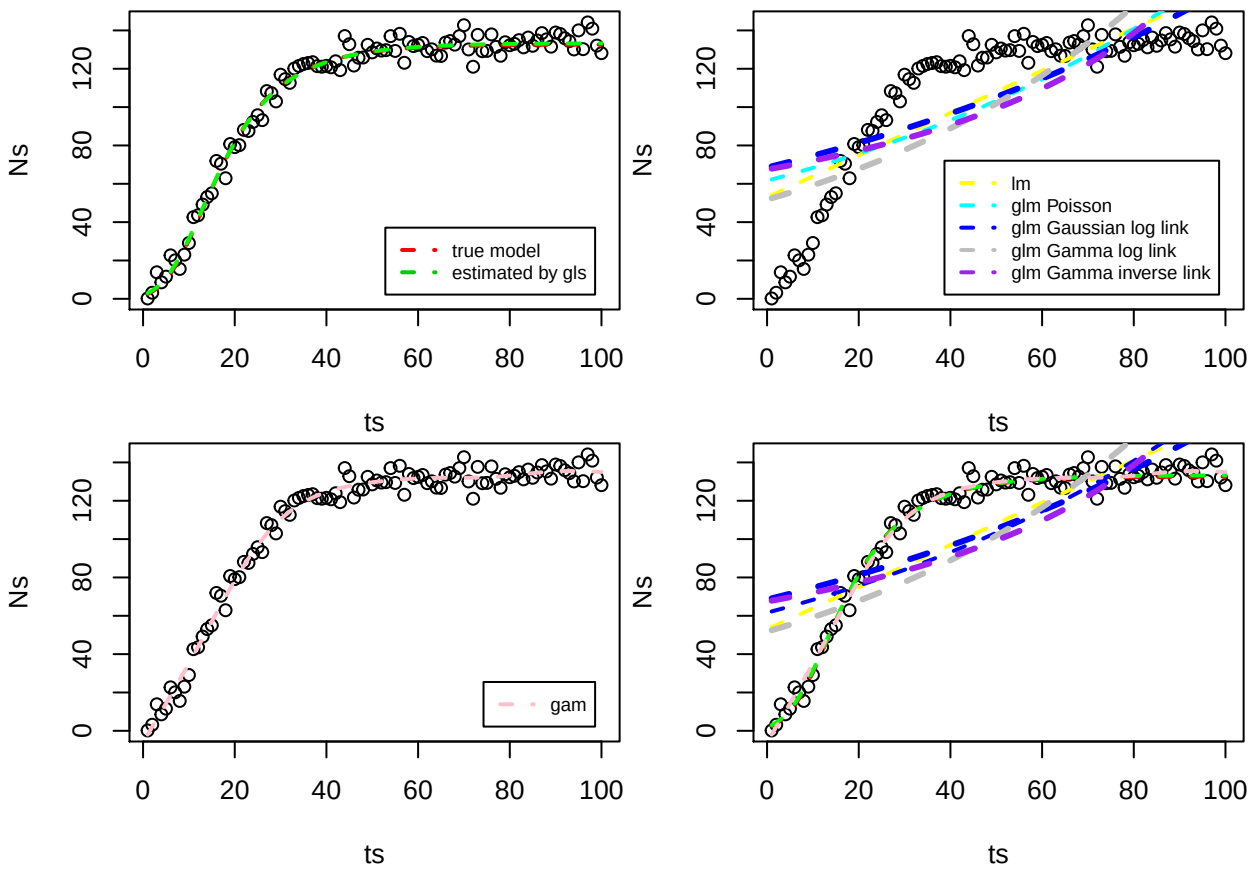
Finally, we could actually try to estimate the model without knowing the functional form of the model. To do so we could use a linear model, a GLM and a GAM. This is implemented next, but note that neither the use of the LM or the GLM would make sense here, since the relation is not linear - we are just doing it to show they are bad models!

```
#fitting a LM
lm1=lm(Ns~ts)
#fitting GLMs - poor performance expected
glm1=glm(Ns~ts,family=poisson(link="log"))
```

```
glm2=glm(Ns~ts,family=gaussian(link="log"))
glm3=glm(Ns~ts,family=Gamma(link="log"))
glm4=glm(Ns~ts,family=Gamma(link="inverse"))
#fitting a GAM
gam1=gam(Ns~s(ts))
```

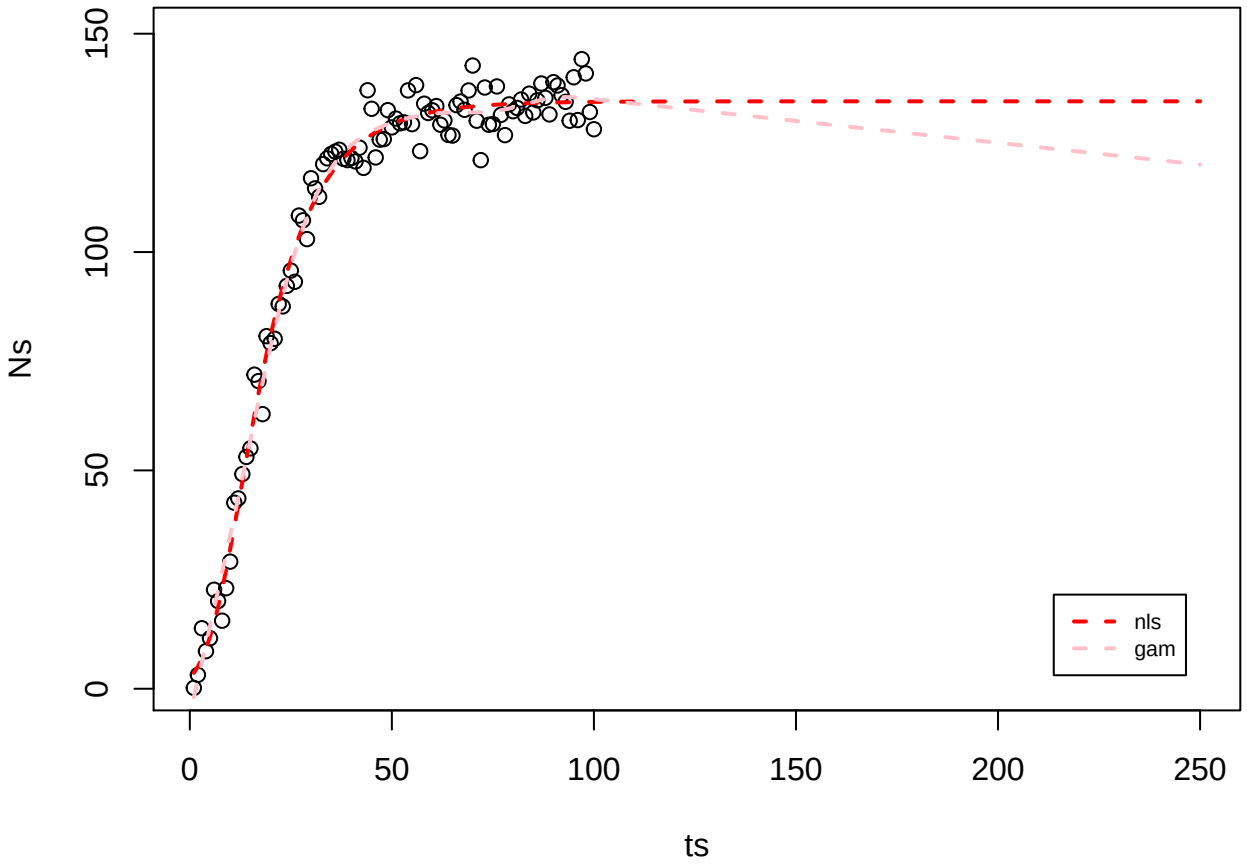
Then, we can get predictions for each one of the estimated models and plot them

```
#predictions
#LM
predlm1=predict(lm1)
#GLM2
predglm1=predict(glm1,newdata=data.frame(ts=ts),type="response")
predglm2=predict(glm2,newdata=data.frame(ts=ts),type="response")
predglm3=predict(glm3,newdata=data.frame(ts=ts),type="response")
predglm4=predict(glm4,newdata=data.frame(ts=ts),type="response")
#GAMs
predgam1=predict(gam1,newdata=data.frame(ts=ts),type="response")
#------------------------------------------------------------
#plot
par(mfrow=c(2,2),mar=c(4,4,0.2,0.2))
plot(ts,Ns)
lines(t,getNt(t,N0=1.63,K=214,r=0.0853),col="red",lwd=2,lty=2)
lines(t,getNt(t,N0=2,K=200,r=0.08),col="green",lwd=2,lty=2)
legend("bottomright",cex=0.7,inset=0.05,legend=c("true model","estimated by gls"),lwd=2,col=2:3,lty=2)
plot(ts,Ns)
lines(t,predlm1,col="yellow",lwd=2,lty=2)
lines(t,predglm1,col="cyan",lwd=2,lty=2)
lines(t,predglm2,col="blue",lwd=3,lty=2)
lines(t,predglm3,col="gray",lwd=3,lty=2)
lines(t,predglm4,col="purple",lwd=3,lty=2)
legend("bottomright",inset=0.05,legend=c("lm","glm Poisson","glm Gaussian log link","glm Gamma log link
lwd=2,cex=0.7,col=c("yellow","cyan","blue","gray","purple"),lty=2)
plot(ts,Ns)
lines(t,predgam1,col="pink",lwd=2,lty=2)
legend("bottomright",inset=0.05,legend="gam",
       lwd=2,cex=0.7,col="pink",lty=2)
plot(ts,Ns)
lines(t,getNt(t,N0=1.63,K=214,r=0.0853),col="red",lwd=2,lty=2)
lines(t,getNt(t,N0=2,K=200,r=0.08),col="green",lwd=2,lty=2)
lines(t,predlm1,col="yellow",lwd=2,lty=2)
lines(t,predglm1,col="blue",lwd=2,lty=2)
lines(t,predglm2,col="blue",lwd=3,lty=2)
lines(t,predglm3,col="gray",lwd=3,lty=2)
lines(t,predglm4,col="purple",lwd=3,lty=2)
lines(t,predgam1,col="pink",lwd=2,lty=2)
```



Clearly, while fitting the true model was the best approach, and gives us parameter estimates that we can interpret, the GAM was actually able to provide a good estimate of the model, but without the functional form of the model being there. What that means is that most likely the GAM model would be horrible to predict the future. We can check that by predicting also for times up to 120, say.

```
ts=1:100
par(mfrow=c(1,1),mar=c(4,4,0.2,0.2))
plot(ts,Ns,xlim=c(1,250),ylim=c(1,150))
ts=1:250
lines(ts,getNt(ts,N0=coef(nls1)[1],K=coef(nls1)[2],r=coef(nls1)[3]),col="red",lwd=2,lty=2)
#GAMs
predgam120=predict(gam1,newdata=data.frame(ts=ts),type="response")
lines(ts,predgam120,col="pink",lwd=2,lty=2)
legend("bottomright",inset=0.05,legend=c("nls","gam"),
       lwd=2,cex=0.7,col=c("red","pink"),lty=2)
```



Interestingly, AIC would be able to spot the best model as being the true model, with the GAM being the only potential contender. This is somewhat reassuring. When we do not know the true functional form of the model, all that we might do is to fit the GAM and try to guess the model from the fitted relationship.

```
AIC(nls1,gam1,lm1,glm2,glm3,glm4)
```

```
##            df       AIC
## nls1 4.000000  592.7270
## gam1 8.847954  601.8163
## lm1  3.000000  912.7391
## glm2 3.000000  938.8170
## glm3 3.000000 1062.5258
## glm4 3.000000 1072.7032
```

Notice that I have not calculated the AIC for the Poisson glm. This is because since the observations were non integers, the model fits but the likelihood cannot be evaluated properly (as the Poisson expects the results to be integers).