## *Generalized linear models*

*An introduction by Christoph Scherber*

What is the difference between transforming the response variable, and a generalized linear model? Up to now, we have dealt with models of the form

$$y = \beta_0 + \beta_1 x + \varepsilon$$

or, in matrix notation:

$y=X\beta + \varepsilon$, where $\beta=(\beta_0,\beta_1)$

$X\beta$ is called the **linear structure**. Several issues in data analysis cannot be resolved using this simple approach:

- if y assumes values only over a limited range, then the model $E(y) = \mu = X\beta + \varepsilon$ does not account for this
- the model does also not incorporate situations in which the variance is a function of the mean.

A **generalized linear model** has two main parts:
(1) a **link function** that describes how the mean depends on the linear predictor
(2) a **variance function** that describes how the variance of y depends on the mean.

If we transform our response variable, we may be lucky to account both for heteroscedasticity and for non-normality in the variance. However, transformations will sometimes resolve only one of both issues.

Generalized linear models are more flexible than transformations of the response, in that they allow a **separate modeling** of **linearity** and **variance** relationships.

**Further, if you transform the response variable, then you transform also the variance of your response variable. Hence, a log transformation will lead to the assumption that your variance is in fact log-normally distributed.**

If we transform our response variable, the resulting model may look like this:

$f_t(y)= X\beta + \varepsilon$, where $f_t(y)$ is the transformation function applied to y.

By contrast, a generalized linear model is expressed in a different way:

| | | |
|---|---|---|
| Linear predictor: | $\eta$ | $= X\beta$ |
| Link function: | $g(\mu)$ | $= \eta$, where $g(\mu)$ relates the mean to the linear predictor |
| Variance function: | var(y) | $= cV(\mu)$ , where c is a constant |

If we define the inverse link $f=g^{-1}$ to be the **mean function** relating $\mu$ to the predictors, we get

| | |
|---|---|
| Mean function: | $f(\eta) = f(X\beta) =\mu$ |

Thus, for every linear predictor η= Xß, we can shift backward and forward using the link function and the mean function:

Link function $g(\mu) = \eta$
Mean function $f(\eta) = \mu$

To put this in words,

> The value of η is obtained by transforming y using the link function, and the predicted value of y is obtained by applying the mean function to η.

Before all this becomes all too unfamiliar, let us consider an example. Let us suppose that in fact our data come from a normal distribution, with normally distributed variance and no restrictions on the mean.

In this case, we would set up our glm like this:

Link function: $g(\mu) = X\beta$
Variance function: $var(y) = cV(\mu)$

Now how do we arrive at $y=X\beta + \varepsilon$? Obviously, $g(\mu)$ has to be μ itself, and var(y) is 1:

$E(y) = \mu = g(\mu) = X\beta + \varepsilon$

> Hence, for data sampled from a normal distribution, the link function is μ and the variance function is 1.

## An example

To find out more about the differences between generalized linear models and the linear models we have worked with so far, let us now set up to artificial datasets; one with "normal" data, and a second one with count data that we have chosen from a poisson distribution:

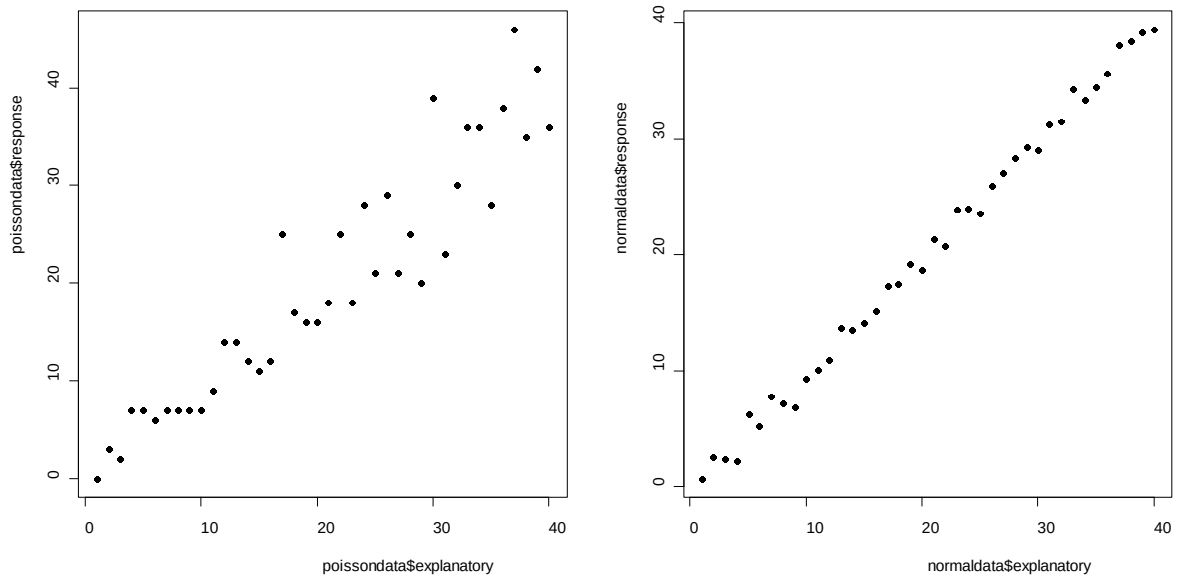
```
poissondata=data.frame(
response=rpois(40,1:40),
factor=gl(2,20),
explanatory=1:40)


normaldata=data.frame(
response=rnorm(40,1:40),
factor=gl(2,20),
explanatory=1:40)
```

The key difference between these two datasets is in the use of rpois vs. rnorm to generate poisson vs. normal random deviates. The vector of means is chosen to be 1:40.

"Response" is the response variable, "explanatory" is a numeric explanatory variable, and "factor" is another, categorical explanatory variable.

Let us first inspect these two datasets to see what the difference between them is:

```
par(mfrow=c(1,2))
plot(poissondata$explanatory,poissondata$response,pch=16)
plot(normaldata$explanatory,normaldata$response,pch=16)
```




You can see at once that the variance in the left-hand graph increases with the mean, while it is constant on the right-hand graph.

We conclude that:

**For data from a poisson distribution, the variance equals the mean. Hence, if the mean is 10, then the variance will also be approximately 10. By contrast, data from a normal distribution will always be assumed to have the same variance, regardless of the mean.**

We can check this also by explicitly calculating the variance for each of the two datasets. This requires us to split the values on the x axis into equal-sized segments (because the variance, obviously, can only be calculated for >2 values).

Hence, the idea is to convert our sequence of x values (1,2,3,4,....40) into a "pseudo factor" that consists of (1,1,1,1,2,2,2,2,3,3,3,3...), so that we can then calculate the variance of all the "levels" of this factor. This allows us to get a rough feeling for the relationship between the variance and the mean in our two artificial datasets:

```
newfac=gl(10,4)
```

This "new factor" now has 10 levels, each replicated 4 times, to give a total length of 40 values. We check if the result is as desired by typing

```
newfac
```

```
[1] 1  1  1  1  2  2  2  2  3  3  3  3  4  4  4  4  5  5  5  5  6  6  6  6  7
[26] 7  7  7  8  8  8  8  9  9  9  9  10 10 10 10
Levels: 1 2 3 4 5 6 7 8 9 10
```

Now how do we calculate the variance in our response variable along the levels of newfac? This is easy to do using tapply():

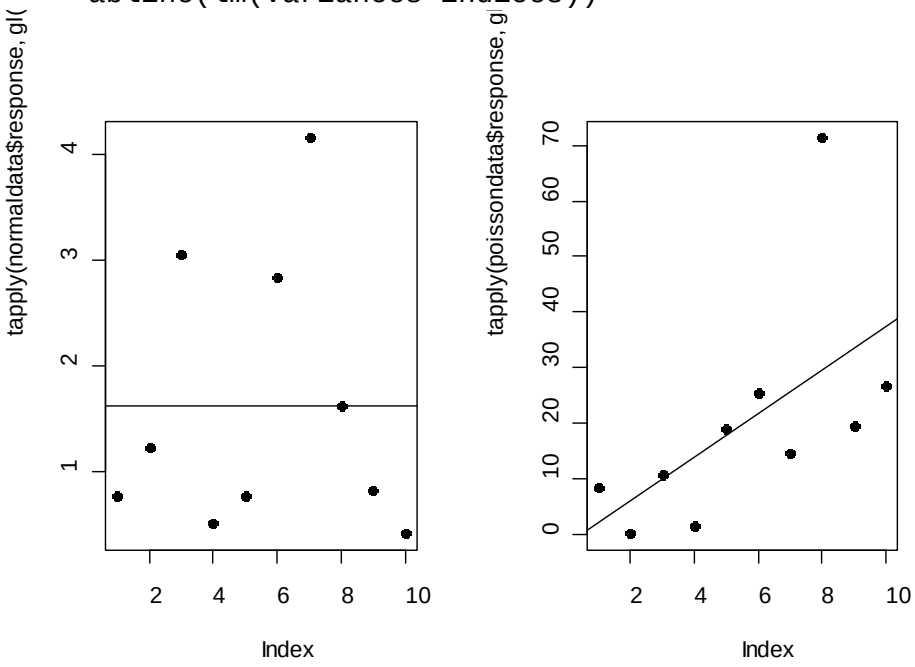
```
tapply(normaldata$response,newfac,var)
```

```
      1         2         3         4         5         6         7         8
0.7695754 1.2290859 3.0454138 0.5218952 0.7728920 2.8394325 4.1523744 1.6278420
        9        10
0.8241246 0.4157948
```

So there you go. The only thing we need to do is convert this list into a numeric vector using as.numeric(), and bind all these things together into a single set of commands.

We set up a plotting region with 1 row and 2 columns, using par(mfrow...)); then, we can plot our results and additionally add some regression lines to indicate the trends in our variances:

```
par(mfrow=c(1,2))

plot(tapply(normaldata$response,gl(10,4),var),pch=16)
     variances=as.numeric(tapply(normaldata$response,gl(10,4),var))
     indices=as.numeric(gl(10,1))
     abline(lm(variances~indices))

plot(tapply(poissondata$response,gl(10,4),var),pch=16)
     variances=as.numeric(tapply(poissondata$response,gl(10,4),var))
     indices=as.numeric(gl(10,1))
     abline(lm(variances~indices))
```




It is plain to see that indeed for the **normally distributed** data, there is **no relationship** between the mean and the variance, while for the **poisson** data, the variance equals the mean – or, in our case, the variance even increases with the mean (which is an indication of **overdispersion**).

In our case, the variance-to-mean ratio is 7.63:

```
var(response)/mean(response)
[1] 7.638342
```

The variance-to-mean ratio for poisson data should be 1. If it is greater than that, this is likely indication of overdispersion.

What does this mean for our overall data analysis strategy? Let us inspect this using our variable "factor" from the two datasets.

```
tapply(poissondata$response,poissondata$factor,var)
       1        2
35.31316 70.22105

tapply(normaldata$response,normaldata$factor,var)
       1        2
34.53792 36.46460
```

You can see that the variance doubles in the poisson data, whereas it is more or less unchanged in the normal data.

Let us now suppose that we set out to analyse our data using a small linear regression model each;

```
model1=lm(response~explanatory,poissondata)
model2=lm(response~explanatory,normaldata)
```

**summary(model1)**

```
Call:
lm(formula = response ~ explanatory, data = poissondata)

Residuals:
    Min      1Q  Median      3Q     Max
-8.2365 -2.6062 -0.5176  2.8208  9.8468

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.46154    1.36939  -0.337    0.738
explanatory  0.98959    0.05821  17.001   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.249 on 38 degrees of freedom
Multiple R-squared: 0.8838,     Adjusted R-squared: 0.8808
F-statistic:   289 on 1 and 38 DF,  p-value: < 2.2e-16
```

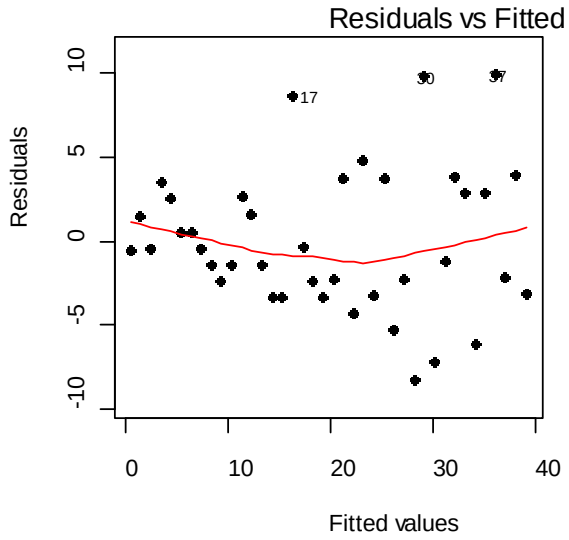
**summary(model2)**

```
Call:
lm(formula = response ~ explanatory, data = normaldata)

Residuals:
     Min       1Q   Median       3Q      Max
-1.72649 -0.55533 -0.09112  0.47061  1.73682
```

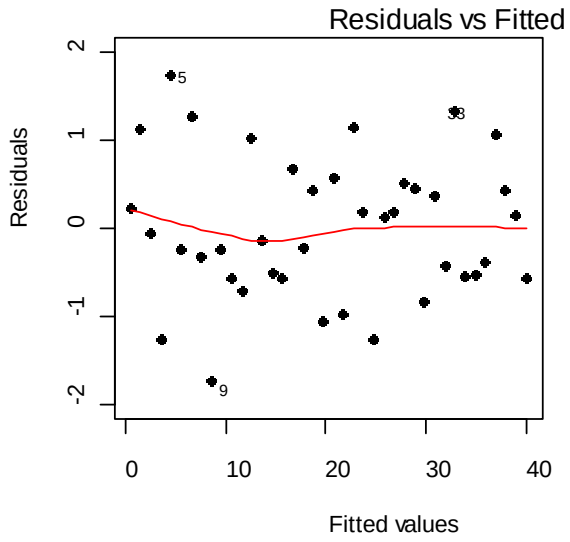
```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.55638    0.26097  -2.132   0.0395 *
explanatory  1.01414    0.01109  91.424   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.8098 on 38 degrees of freedom
Multiple R-squared: 0.9955,     Adjusted R-squared: 0.9954
F-statistic:  8358 on 1 and 38 DF,  p-value: < 2.2e-16
```

The residuals in the poisson model are slightly asymmetric, while they are perfectly symmetric in the normal model. The intercept in the poisson model is non-significant, while it is significant in the normal model. The standard error for the parameter estimates is higher in the poisson model compared with the normal model. The residual standard error in the poisson model is much greater than in the normal model. Finally, the adjusted R squared value for the poisson model is worse compared with the normal model.

The plot of model1 shows the following graphs:


Residuals vs Fitted
Residuals
Fitted values
10
5
0
-5
-10
0
10
20
30
40
17
30
37

Let us compare these plots with the ones from model2:




You can clearly see the following things:

(1) The variance increases with the mean in model1 (the residuals look “trumped shaped”)

(2) the square root of the residuals against the fitted values increases with the mean in the poisson data

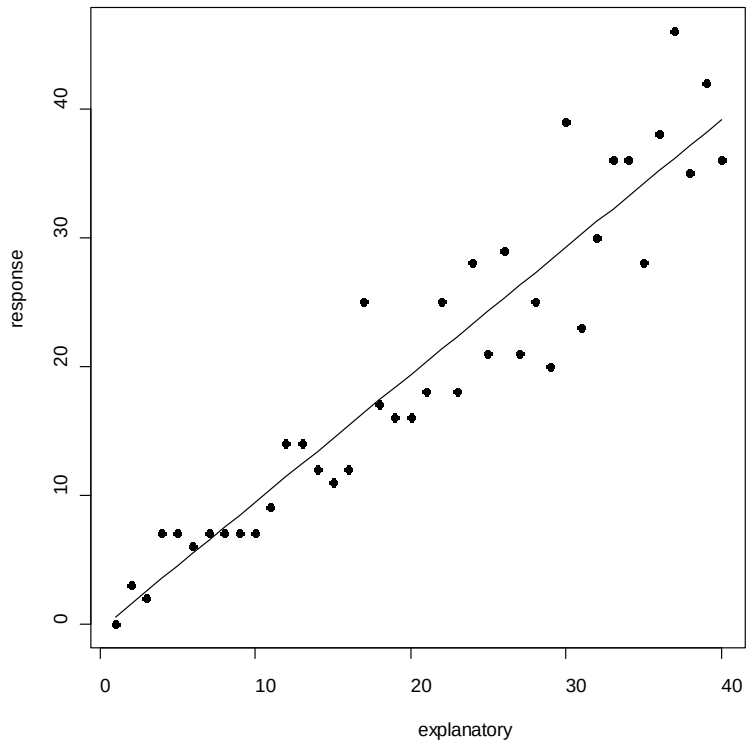
Now what is wrong with the analysis of poisson data we have done so far? Well, basically and first of all it violates several assumptions that we make in linear models. Most importantly, linear models assume constancy of variance, which is, as we have seen several times now, clearly not the case.

The usual way to cure this would be to transform the data. Let us concentrate our work, therefore, on the poisson data from now on:

```
attach(poissondata)

model1=lm(response~explanatory,poissondata)
model2=lm(sqrt(response+0.5)~explanatory,poissondata)
model3=lm(log(response+1)~explanatory,poissondata)
```

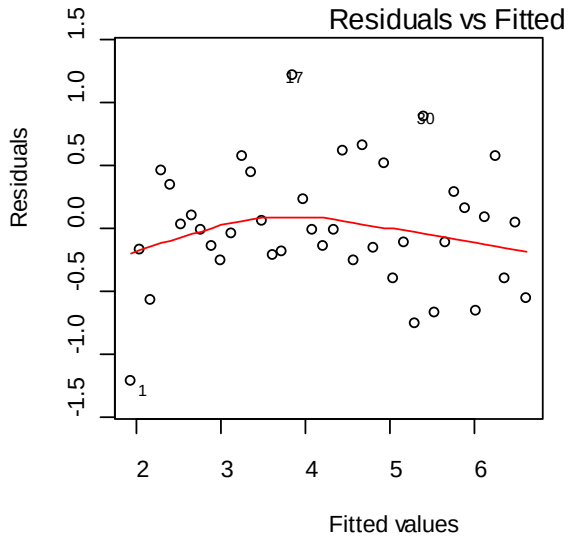
Let us inspect graphically if these models fit the data well:

```
par(mfrow=c(1,1))
plot(explanatory,response,pch=16)

lines(explanatory,predict(model1,explanatory=explanatory))
lines(explanatory,(predict(model2,explanatory=explanatory))^2-
0.5,lty=2)
lines(explanatory,exp(predict(model3,explanatory=explanatory))-
1,lty=3)
```

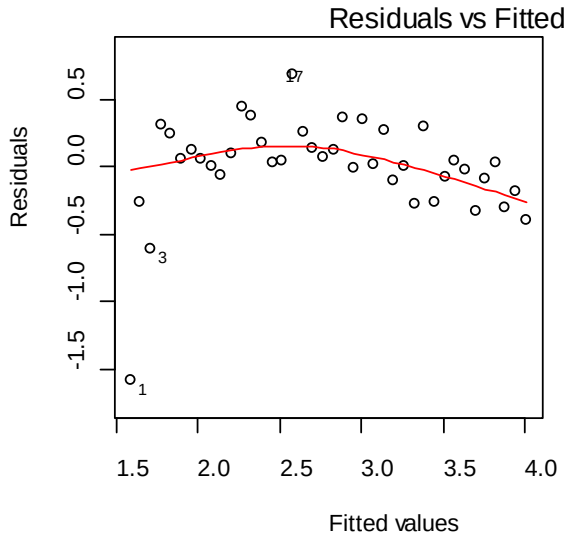



The dashed line shows the predictions from model2, while the dotted line shows the predictions from model3. Of these models, the linear model and the model with square-root transformed response variable perform best. Let us see what the residual plots show us:

First the square-root transformed data:




These plots are not too bad, and we may do well with this model already . Now for the log-transformed data:




The transformation has introduced an “n”-shape into the plot of the residuals against the fitted values, and the other diagnostic plots also do not look too well.

But how does our analysis change if we use a generalized linear model to analyze these data?

```
model4=glm(response~explanatory,poissondata,family=poisson)
summary(model4)

Call:
glm(formula = response ~ explanatory, family = poisson, data =
poissondata)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-3.40352  -0.61009  -0.08195   0.41912   2.74581

Coefficients:
```

```
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 1.702845   0.099586   17.10   <2e-16 ***
explanatory 0.053627   0.003427   15.65   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 324.820  on 39  degrees of freedom
Residual deviance:  52.118  on 38  degrees of freedom
AIC: 237.09

Number of Fisher Scoring iterations: 4
```

The first thing we have to account for is overdispersion. We see that the dispersion parameter of the poisson family is taken to be 1. Hence, the residual deviance, divided by the residual degrees of freedom, should also be 1. As it happens, 52.118/38=1.37, which is a (mild) indication of overdispersion. We might want to correct for this by using quasipoisson errors in our glm:

```
model5=glm(response~explanatory,poissondata,family=quasipoisson)
```

The next thing we note is that the fitted values are on the log scale, so we have to back-transform them to compare the model with our previous ones:

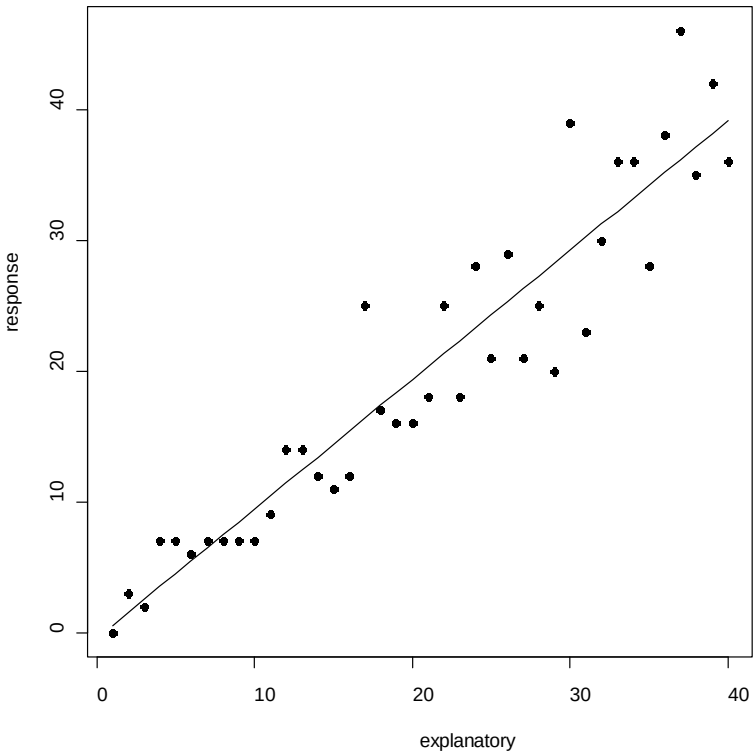
```
exp(coef(model5))
```

```
     (Intercept) explanatory
     5.489545    1.055091
```

This should caution us: Why is the intercept 5.48, given the fact that it is 1 (as we know, because we simulated the data on our own)?

Now let us see how well our glm fits the data:

```
par(mfrow=c(1,1))
plot(explanatory,response,pch=16)
lines(explanatory,predict(model1,data.frame(explanatory=explanatory)))
lines(explanatory,(predict(model2, data.frame(explanatory=explanatory)))^2-
0.5,lty=2)
lines(explanatory,exp(predict(model3, data.frame(explanatory=explanatory)))-1,lty=3)
lines(explanatory,exp(predict(model5,
data.frame(explanatory=explanatory))),lty=1,col="red")
```




We can see clearly that the glm has produced results that are qualitatively similar to the log transformation. So, in our case, the standard glm options (log link, variance=mean) seem not to resolve our problem.

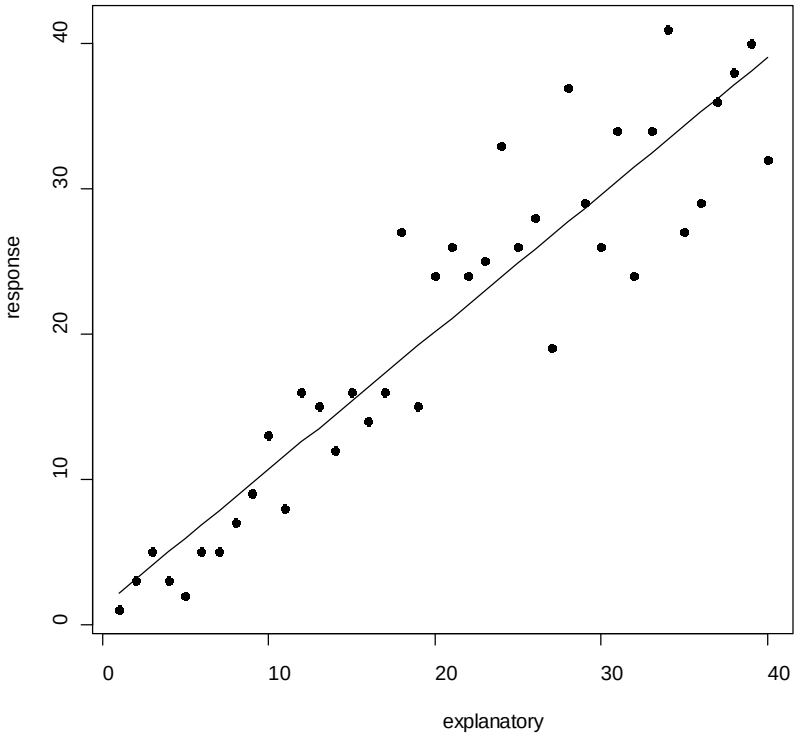
What we are left with now is a problem common to statistical analyses: We have several models, each of them performing more or less well; we also have residual plots to guide our decisions. And now we have to choose which model we use to describe our data.

We have seen that the main problem is the log transformation of the response. So why don´t we try a glm with variance=mean, but an identity link instead of the log link? Thus, we model only the variance, but leave the mean unchanged.

```
model6=glm(response~explanatory,
     quasi(variance="mu",link="identity"))
```

Let us plot the predictions from our three candidate models: The “no transformation” model, the “square root” model, and model 6.

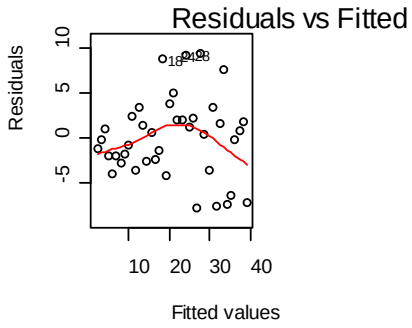
```
plot(explanatory,response,pch=16)
lines(explanatory,predict(model1,data.frame(explanatory=explanatory)))
lines(explanatory,(predict(model2,data.frame(explanatory=explanatory)))^2-
0.5,lty=2)
lines(explanatory,predict(model7,data.frame(explanatory=explanatory),type="response
"),lty=1,col="red")
```




We see that model1 (the “no transformation” model) and model6 (our most complicated glm model) produce a fit that is equally suitable, with the “square root” model slightly off.

Our final decision will be based on residual plots again:

```
par(mfrow=c(3,4))
plot(model1)
plot(model2)
plot(model6)
```




The first row is for model1, the second for model2 and the third for model6. It is clear that model6 is best: All three plots look reasonably well behaved. The residuals are nicely normally distributed, and there is only an almost invisible pattern in the residuals vs. fitted values.

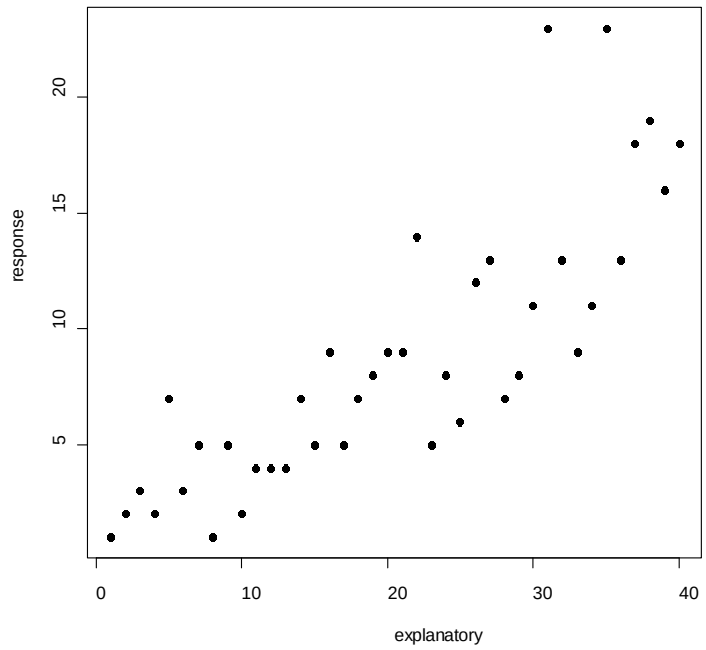
We summarize with the following statements:

1. Always look at your data from many sides
2. Understand the underlying mechanism that generated the data (e.g. a poisson process)
3. Distrust every model and check it carefully
4. Use residual plots as often as possible
5. The default options of GLM´s do not always produce nice results

Now, for a last example, let´s try out what happens if we have both non-normal errors and non-linearity in our data.

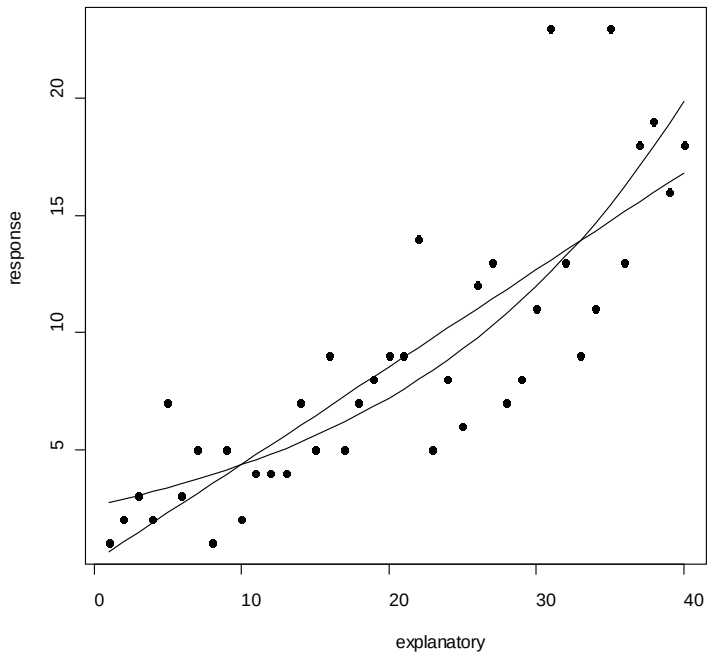
First, we set up the data frame as usual:

```
poissondata2=data.frame(
response=rpois(40, exp(seq(1,3, length.out=40))),
factor=gl(2,20),
explanatory=1:40)

attach(poissondata2)
plot(response~explanatory,pch=16)
```




Now, we start modelling using a classical linear model, and a generalized linear model.

```
model1=lm(response~explanatory)
model2=glm(response~explanatory,poisson)

AIC(model1,model2)

summary(model2)

plot(response~explanatory,pch=16)
lines(explanatory,predict(model2,list(explanatory=explanatory),type=
"response"))
lines(explanatory,predict(model1,list(explanatory=explanatory),type=
"response"))
```




In this case, the GLM definitely has produced a better fit than the linear model.