Stat 3701 Lecture Notes: Statistical Models

Charles J. Geyer

October 24, 2022

1 License

2 R

- The version of R used to make this document is 4.2.1.
- The version of the `rmarkdown` package used to make this document is 2.16.
- The version of the `pkgsearch` package used to make this document is 3.1.1.
- The version of the `MASS` package used to make this document is 7.3.58.1.
- The version of the `mgcv` package used to make this document is 1.8.40.

```
library("pkgsearch")
library("MASS")
library("mgcv")
```

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

```
## This is mgcv 1.8-40. For overview type 'help("mgcv-package")'.
```

3 Statistical Models of the R Kind

Statistical models come in two kinds, those that R calls models and those it doesn't. (Have you heard the joke: there are two kinds of people in this world, those who divide everything into two kinds, and those who don't? Or its geek version: there are 10 kinds of people in this world, those who know binary and those who don't?)

What R calls statistical models are fit by functions that work like the R function `lm` in many ways, make assumptions that are like those for linear models in many ways, and use many of the same generic functions to process their output, including `summary`, `predict`, and `anova`.

Of course these models are not *exactly* like linear models. They have to be different in *some* ways. But they are similar enough for some of the intuitions to carry over and some of the ways of working with models to carry over.

3.1 Regression Models

Linear models are a special case of "regression" models. They make the following assumptions.

- The data can be put in a data frame.

- The rows are *cases*, also called *individuals.* The columns are *variables.*
- One variable is special: the *response.* All other variables are called *predictors.*
- The job of *regression* is to estimate the conditional distribution of the response given the predictors for each case. If y_i is the response for case i, and x_i and z_i are the predictors for case i, then the job is to estimate the conditional distribution of y_i given x_i and z_i.

For those who have not had a theory course and don't know what a conditional distribution is, it is just a probability distribution like any other probability distribution — given by a probability mass function (PMF) if the response is a discrete random variable or by a probability density function (PDF) if the response is a continuous random variable — except that the parameters of the distribution are allowed to depend on the conditioning variables. The conditional distribution of the response given the predictors depends on what the values of the predictors are.

If you imagine new data (not part of the data used to fit the model), in which you do not get to see the response y_i for this new case but do get to see the predictors (say x_i and z_i), then the estimated conditional distribution allows *predictions* about y_i given the values of x_i and z_i.

There can be as many predictor variables as you like. There is only one response variable. (Exception: the R function `mlm` does linear models with *vector* response, but we won't talk about that.)

You can even make up new predictor variables from old ones.

- In polynomial regression, starting with predictors x_i and z_i, one can make up
  - quadratic predictors x_i^2 and $x_i z_i$ and z_i^2
  - cubic predictors x_i^3 and $x_i^2 z_i$ and $x_i z_i^2$ and z_i^3
  - and so forth for higher order polynomials.
- In regression with a categorical predictor vector x (what R calls a `factor`) one must make up *dummy variables.*
  - If x has k categories, then there are dummy variables d_1, d_2, $\ldots$, d_k.
  - A dummy variable is zero-or-one valued (an *indicator* variable).
  - The component of d_j for case i is equal to one if the value of x_i is the j-th category (in R `levels(x)[j]`).
  - If there is an *intercept* in the regression, then one of the dummy variables is "dropped" (left out of the model). Otherwise, there would be *collinearity* (and not all *coefficients* could be estimated).
  - Most R functions that fit models treat R variables of type `"character"` as if they were factors. They are automatically converted to factors in the process of model fitting.
- The *intercept* variable (if present) is itself a "made up" predictor. It is the vector all of whose components are equal to one.
- In general one can make up arbitrary predictors. Starting with a variable x one can make up $g_1(x)$, $g_2(x)$, $g_3(x)$ and so forth, where g_1 and g_2 and g_3 are *arbitrary* functions that act vectorwise.

The regression equation is

$$\theta_i = m_{i1}\beta_1 + m_{i2}\beta_2 + \cdots + m_{ip}\beta_p$$

where θ_i is some parameter (more on this later), the betas are unknown parameters to estimate (called the *coefficients* of the model). And each m_{ij} is the value of some predictor variable (either originally given or made-up) for case i.

If we are sophisticated we think of m_{ij} as as the components of a matrix M. Then the regression equation above can be rewritten as a matrix-vector equation

$$\theta = M\beta$$

If there are n cases and p coefficients, then

- the dimension of θ is n,
- the dimension of β is p,
- the dimension of M is n by p (n rows and p columns)

The matrix M is called the *model matrix* by some and the *design matrix* by others. But the name *design matrix* does not make much sense when the data do not come from a designed experiment, although this does not stop the people who like the term from using it for observational studies. Also R uses only the term *model matrix* in its documentation and in the name of the function `model.matrix` that turns R formulas into model matrices. So we will always say *model matrix*.

Weisberg makes a useful distinction. He calls the originally given predictor variables *predictors*, just like I have. He calls the columns of the model matrix *regressors*. This is shorter than and sounds more scientific than "either originally given or made-up predictors".

You know all of this from STAT 3032. This is just a reminder.

3.2 Linear Models (LM)

In order to know what stays the same and what changes, we briefly review linear models (those fit by the R function `lm`). Their assumptions are as follows.

- The parameter in the regression equation is the mean so we rewrite it
$$\mu = M\beta$$
- The components of the response vector are conditionally independent given the predictors.
- The conditional distributions of the components of the response vector given the predictors are *normal* and the variance does not depend on the predictors. If the response vector is y, then the conditional distribution of y_i given the predictors is
$$\text{Normal}(\mu_i, \sigma^2)$$
- the unknown parameters to be estimated are the regression coefficients (betas) and σ^2.

You may be in the habit of thinking of the regression equation as
$$y = M\beta + e$$
or written out in long form
$$y_i = m_{i1}\beta_1 + m_{i2}\beta_2 + \cdots + m_{ip}\beta_p + e_i$$
where e is "error". If so, forget that. That formulation does not generalize to so-called "generalized linear models". The description above does generalize.

Let's do a toy example, just to make the abstractions above concrete.

```
foo <- read.table("http://www.stat.umn.edu/geyer/5102/data/ex5-4.txt",
    header = TRUE)
names(foo)
```

```
## [1] "x"     "color" "y"
```

```
sapply(foo, class)
```

```
##           x       color           y
##   "integer" "character"   "numeric"
```

Suppose we want to fit parallel regression lines, regressing `y` on `x` for each color.

```r
lout <- lm(y ~ x + color, data = foo)
summary(lout)
```

```
##
## Call:
## lm(formula = y ~ x + color, data = foo)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.2398  -2.9939   0.1725   3.5555  11.9747
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 13.16989    1.01710  12.948  < 2e-16 ***
## x            1.00344    0.02848  35.227  < 2e-16 ***
## colorgreen   2.12586    1.00688   2.111   0.0364 *
## colorred     6.60586    1.00688   6.561  8.7e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.034 on 146 degrees of freedom
## Multiple R-squared:  0.898,  Adjusted R-squared:  0.8959
## F-statistic: 428.6 on 3 and 146 DF,  p-value: < 2.2e-16
```

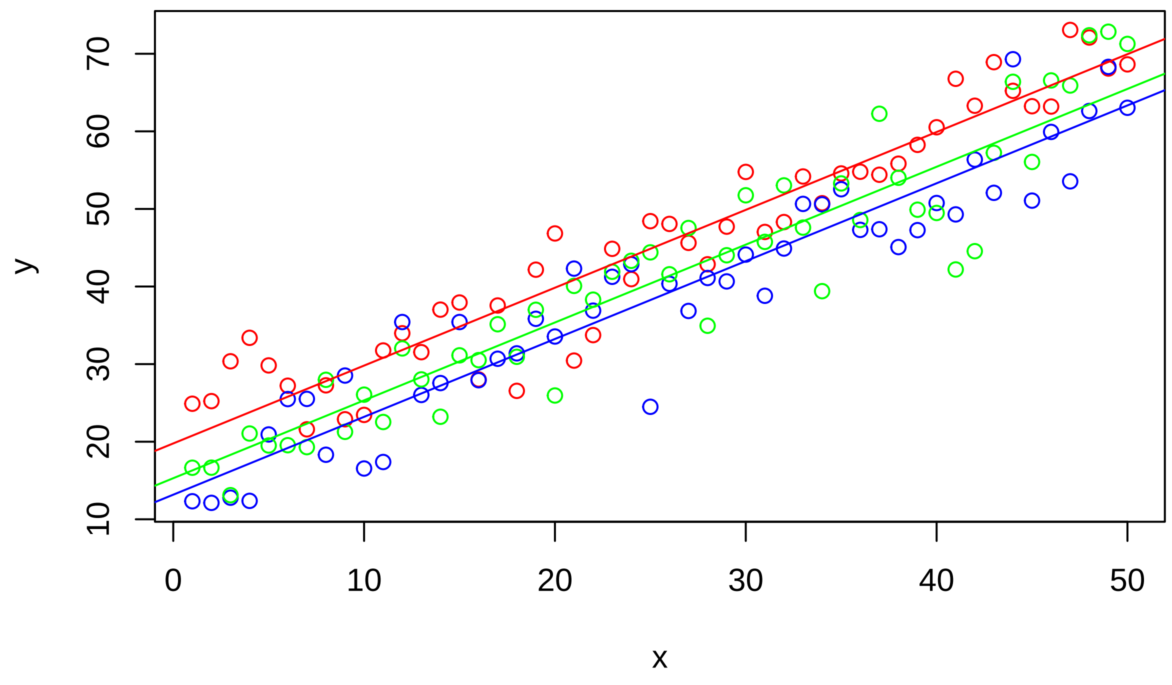
It will be easier to work with if we do not have an intercept, in which case R will not drop one dummy variable for `color` and the fitted model will be the same in the sense that it makes the same predictions about the conditional distribution of the response given the predictors, although the coefficients will change.

```r
lout <- lm(y ~ 0 + x + color, data = foo)
summary(lout)
```

```
##
## Call:
## lm(formula = y ~ 0 + x + color, data = foo)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.2398  -2.9939   0.1725   3.5555  11.9747
##
## Coefficients:
##            Estimate Std. Error t value Pr(>|t|)
## x           1.00344    0.02848   35.23   <2e-16 ***
## colorblue  13.16989    1.01710   12.95   <2e-16 ***
## colorgreen 15.29575    1.01710   15.04   <2e-16 ***
## colorred   19.77575    1.01710   19.44   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.034 on 146 degrees of freedom
## Multiple R-squared:  0.9875, Adjusted R-squared:  0.9872
## F-statistic:  2890 on 4 and 146 DF,  p-value: < 2.2e-16
```

Then we plot the results

```
coef <- coefficients(lout)
attach(foo)
plot(x, y, col = as.character(color))
for (col in unique(color)) {
    predname <- paste("color", col, sep="")
    abline(coef[predname], coef["x"], col = col)
}
```



```
detach(foo)
```

But that wasn't what we really wanted to illustrate. Where is the model matrix and the response vector? We don't see them. But we can!

```
lout <- lm(y ~ 0 + x + color, data = foo, x = TRUE, y = TRUE)
# model matrix
lout$x
```

```
##     x colorblue colorgreen colorred
## 1   1         0          0        1
## 2   1         1          0        0
## 3   1         0          1        0
## 4   2         0          0        1
## 5   2         1          0        0
## 6   2         0          1        0
## 7   3         0          0        1
## 8   3         1          0        0
## 9   3         0          1        0
## 10   4           0            0          1
## 11   4           1            0          0
## 12   4           0            1          0
## 13   5           0            0          1
## 14   5           1            0          0
## 15   5           0            1          0
## 16   6           0            0          1
## 17   6           1            0          0
## 18   6           0            1          0
## 19   7           0            0          1
## 20   7           1            0          0
## 21   7           0            1          0
## 22   8           0            0          1
## 23   8           1            0          0
## 24   8           0            1          0
## 25   9           0            0          1
## 26   9           1            0          0
## 27   9           0            1          0
## 28  10           0            0          1
## 29  10           1            0          0
## 30  10           0            1          0
## 31  11           0            0          1
## 32  11           1            0          0
## 33  11           0            1          0
## 34  12           0            0          1
## 35  12           1            0          0
## 36  12           0            1          0
## 37  13           0            0          1
## 38  13           1            0          0
## 39  13           0            1          0
## 40  14           0            0          1
## 41  14           1            0          0
## 42  14           0            1          0
## 43  15           0            0          1
## 44  15           1            0          0
## 45  15           0            1          0
## 46  16           0            0          1
## 47  16           1            0          0
## 48  16           0            1          0
## 49  17           0            0          1
## 50  17           1            0          0
## 51  17           0            1          0
## 52  18           0            0          1
## 53  18           1            0          0
## 54  18           0            1          0
## 55  19           0            0          1
## 56  19           1            0          0
## 57  19           0            1          0
## 58  20           0            0          1
## 59  20           1            0          0
## 60  20           0            1          0
## 61  21           0            0          1
## 62  21           1            0          0
## 63  21           0            1          0
## 64  22           0           0          1
## 65  22           1           0          0
## 66  22           0           1          0
## 67  23           0           0          1
## 68  23           1           0          0
## 69  23           0           1          0
## 70  24           0           0          1
## 71  24           1           0          0
## 72  24           0           1          0
## 73  25           0           0          1
## 74  25           1           0          0
## 75  25           0           1          0
## 76  26           0           0          1
## 77  26           1           0          0
## 78  26           0           1          0
## 79  27           0           0          1
## 80  27           1           0          0
## 81  27           0           1          0
## 82  28           0           0          1
## 83  28           1           0          0
## 84  28           0           1          0
## 85  29           0           0          1
## 86  29           1           0          0
## 87  29           0           1          0
## 88  30           0           0          1
## 89  30           1           0          0
## 90  30           0           1          0
## 91  31           0           0          1
## 92  31           1           0          0
## 93  31           0           1          0
## 94  32           0           0          1
## 95  32           1           0          0
## 96  32           0           1          0
## 97  33           0           0          1
## 98  33           1           0          0
## 99  33           0           1          0
## 100 34           0           0          1
## 101 34           1           0          0
## 102 34           0           1          0
## 103 35           0           0          1
## 104 35           1           0          0
## 105 35           0           1          0
## 106 36           0           0          1
## 107 36           1           0          0
## 108 36           0           1          0
## 109 37           0           0          1
## 110 37           1           0          0
## 111 37           0           1          0
## 112 38           0           0          1
## 113 38           1           0          0
## 114 38           0           1          0
## 115 39           0           0          1
## 116 39           1           0          0
## 117 39           0           1          0
## 118 40          0           0         1
## 119 40          1           0         0
## 120 40          0           1         0
## 121 41          0           0         1
## 122 41          1           0         0
## 123 41          0           1         0
## 124 42          0           0         1
## 125 42          1           0         0
## 126 42          0           1         0
## 127 43          0           0         1
## 128 43          1           0         0
## 129 43          0           1         0
## 130 44          0           0         1
## 131 44          1           0         0
## 132 44          0           1         0
## 133 45          0           0         1
## 134 45          1           0         0
## 135 45          0           1         0
## 136 46          0           0         1
## 137 46          1           0         0
## 138 46          0           1         0
## 139 47          0           0         1
## 140 47          1           0         0
## 141 47          0           1         0
## 142 48          0           0         1
## 143 48          1           0         0
## 144 48          0           1         0
## 145 49          0           0         1
## 146 49          1           0         0
## 147 49          0           1         0
## 148 50          0           0         1
## 149 50          1           0         0
## 150 50          0           1         0
## attr(,"assign")
## [1] 1 2 2 2
## attr(,"contrasts")
## attr(,"contrasts")$color
## [1] "contr.treatment"
```

```
# response vector
lout$y
```

```
##      1      2      3      4      5      6      7      8      9     10     11
## 24.894 12.323 16.645 25.231 12.119 16.654 30.366 12.784 13.108 33.389 12.377
##     12     13     14     15     16     17     18     19     20     21     22
## 21.053 29.832 20.933 19.510 27.217 25.493 19.547 21.594 25.516 19.300 27.266
##     23     24     25     26     27     28     29     30     31     32     33
## 18.319 27.977 22.893 28.526 21.272 23.453 16.545 26.064 31.757 17.384 22.537
##     34     35     36     37     38     39     40     41     42     43     44
## 33.977 35.425 32.021 31.540 26.024 28.029 37.023 27.558 23.217 37.941 35.422
##     45     46     47     48     49     50     51     52     53     54     55
## 31.131 28.019 27.925 30.516 37.553 30.690 35.140 26.554 31.384 30.943 42.170
##     56     57     58     59     60     61     62     63     64     65     66
## 35.838 37.001 46.840 33.557 25.956 30.454 42.309 40.079 33.739 36.892 38.306
##     67     68     69     70     71     72     73     74     75     76     77
## 44.858 41.251 41.904 40.955 42.880 43.316 48.426 24.497 44.400 48.069 40.356
##     78     79     80     81     82     83     84     85     86     87     88
## 41.578 45.626 36.850 47.543 42.866 41.102 34.943 47.719 40.658 44.029 54.769
##     89     90     91     92     93     94     95     96     97     98     99
## 44.117 51.758 47.033 38.805 45.756 48.309 44.897 53.034 54.183 50.649 47.593
##    100    101    102    103    104    105    106    107    108    109    110
## 50.734 50.592 39.404 54.573 52.541 53.288 54.794 47.300 48.573 54.407 47.381
##    111    112    113    114    115    116    117    118    119    120    121
## 62.267 55.833 45.085 54.018 58.259 47.267 49.899 60.527 50.764 49.482 66.771
##    122    123    124    125    126    127    128    129    130    131    132
## 49.294 42.197 63.300 56.363 44.555 68.916 52.074 57.232 65.223 69.296 66.387
##    133    134    135    136    137    138    139    140    141    142    143
## 63.237 51.071 56.056 63.198 59.924 66.563 73.067 53.556 65.907 72.092 62.624
##    144    145    146    147    148    149    150
## 72.380 68.096 68.308 72.840 68.632 63.036 71.266
```

If we ever had a model in which the R formula mini-language — the mini-language whose interpreter is the R function `model.matrix` and whose documentation is `?formula` — is inadequate to describe the model matrix, then we can always just construct the response vector and model matrix without using a formula and tell `lm` to use them.

```
m <- as.matrix(lout$x) ### strip attributes
y <- as.vector(lout$y) ### strip names
lout.too <- lm(y ~ 0 + m)
summary(lout.too)
```

```
##
## Call:
## lm(formula = y ~ 0 + m)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.2398  -2.9939   0.1725   3.5555  11.9747
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## mx            1.00344    0.02848   35.23   <2e-16 ***
## mcolorblue   13.16989    1.01710   12.95   <2e-16 ***
## mcolorgreen 15.29575    1.01710   15.04   <2e-16 ***
## mcolorred   19.77575    1.01710   19.44   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.034 on 146 degrees of freedom
## Multiple R-squared:  0.9875, Adjusted R-squared:  0.9872
## F-statistic:  2890 on 4 and 146 DF,  p-value: < 2.2e-16
```

We need the `0 +` because the "intercept" regressor is already in `m`.

Except for changing the names of the coefficients by pasting `"m"` on the front, everything is the same.

```
identical(as.vector(coefficients(lout)), as.vector(coefficients(lout.too)))
```

```
## [1] TRUE
```

3.3 Generalized Linear Models (GLM)

3.3.1 Introduction

In generalized linear models we drop the normal distribution of the response given the predictors. We even drop the assumption that this distribution is continuous. It works for discrete response.

- In LM the response must be continuous, but the predictors can be anything (categorical predictors get turned into dummy variables).
- In GLM the response can be discrete or continuous.
- The R function `glm` fits GLM.
- The families that are allowed for the conditional distribution of response given predictor are documented on `?family`. The two most important are `binomial` and `poisson`.
- The regression equation has the general form

$$\theta = M\beta$$

  where θ_i is some parameter (usually not the mean) that specifies the conditional distribution of y_i given the predictors for case i within a specified family. The vector θ is called the *linear predictor* in GLM parlance.
- The relationship between mean values and linear predictor values is given by the *link function*. If g is the link function, then $\theta = g(\mu)$, where the link function g operates vectorwise.
  - For the binomial family, the default link function is the logit function (pronounced low-jit)
  - For the poisson family, the default link function is `log`
- The components of the response vector are conditionally independent given the predictors. (This assumption is the same for LM and GLM.)

R does not have the logit function but we can easily define it. We also define its inverse function.

```
logit <- function(p) log(p) - log1p(- p)
invlogit <- function(theta) 1 / (1 + exp(- theta))
```

3.3.2 Model Fitting

So let's try out GLM with Bernoulli response (a random variable is *Bernoulli* if it is zero-or-one-valued, in other words if it is binomial with sample size one).

```
foo <- read.table("http://www.stat.umn.edu/geyer/5102/data/ex6-1.txt",
    header = TRUE)
names(foo)
```

```
## [1] "x" "y"
```

```
sapply(foo, class)
```

```
##         x         y
## "integer" "integer"
```

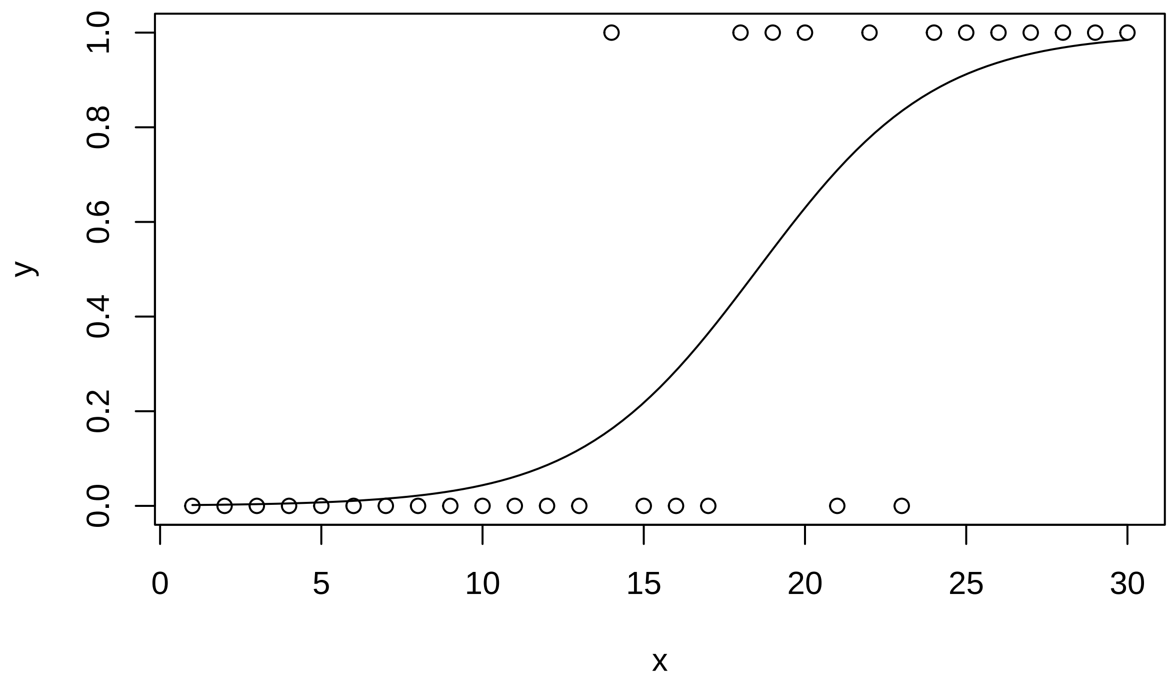
```
gout <- glm(y ~ x, data = foo, family = binomial)
summary(gout)
```

```
##
## Call:
## glm(formula = y ~ x, family = binomial, data = foo)
##
```

```
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.89589  -0.34210  -0.09359   0.34596   1.90606
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -6.7025     2.4554  -2.730  0.00634 **
## x             0.3617     0.1295   2.792  0.00524 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 40.381  on 29  degrees of freedom
## Residual deviance: 17.666  on 28  degrees of freedom
## AIC: 21.666
##
## Number of Fisher Scoring iterations: 6
```

A plot shows what we have done

```
with(foo, plot(x, y))
curve(predict(gout, newdata = data.frame(x = x), type = "response"),
    add = TRUE)
```



From the plot we see the reason for link function. The curve is the mean of the conditional Bernoulli distribution of y given x. It does not look like this relation should be linear. The curve given by the GLM looks reasonable. So whether or not the logit link is exactly the right thing, it is *a lot* more reasonable than the identity link (which would assume means are linear).

The R function `glm` does not even allow trying to use identity link here.

3.3.3 Hypothesis Tests

Suppose we try out some polynomial regressions (polynomial on the linear predictor scale).

```
gout2 <- glm(y ~ x + I(x^2), data = foo, family = binomial)
summary(gout2)
```

```
##
## Call:
## glm(formula = y ~ x + I(x^2), family = binomial, data = foo)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.89479  -0.31789  -0.06147   0.37706   1.90034
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.047585   7.036637  -1.144    0.253
## x            0.518776   0.758023   0.684    0.494
## I(x^2)      -0.004293   0.019942  -0.215    0.830
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 40.381  on 29  degrees of freedom
## Residual deviance: 17.618  on 27  degrees of freedom
## AIC: 23.618
##
## Number of Fisher Scoring iterations: 7
```

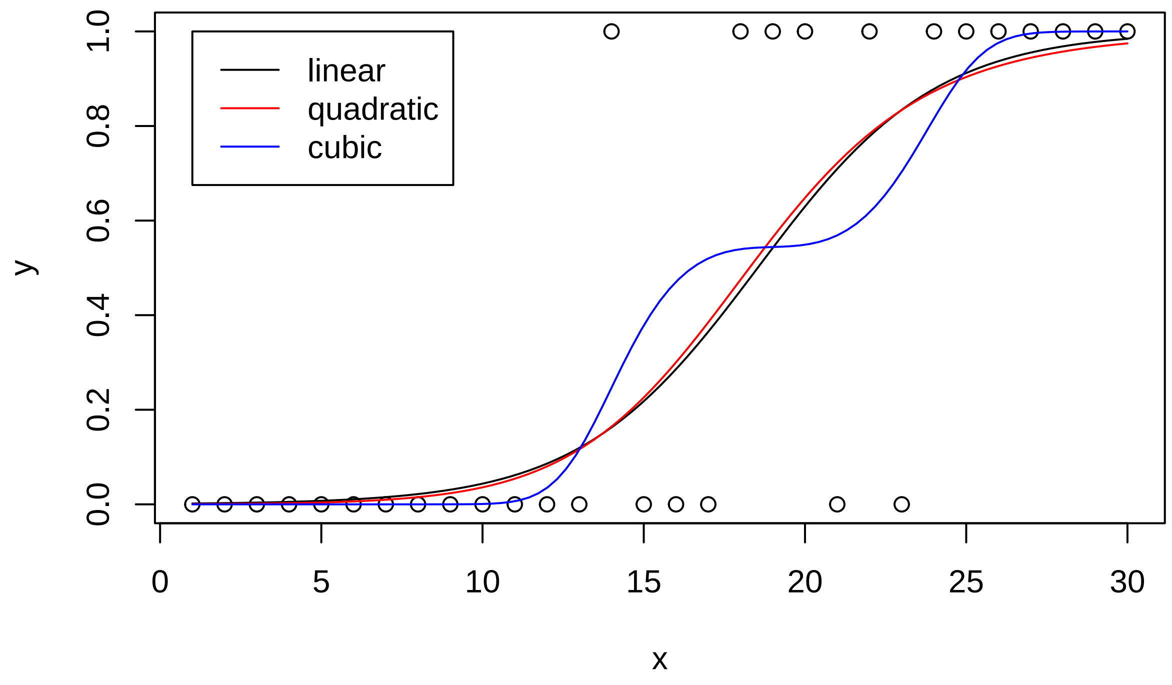
```
gout3 <- glm(y ~ x + I(x^2) + I(x^3), data = foo, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(gout3)
```

```
##
## Call:
## glm(formula = y ~ x + I(x^2) + I(x^3), family = binomial, data = foo)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.55860  -0.09545   0.00000   0.18845   1.67327
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -68.90383   79.12749  -0.871    0.384
## x            10.89649   13.00993   0.838    0.402
## I(x^2)       -0.57339    0.69988  -0.819    0.413
## I(x^3)        0.01007    0.01233   0.816    0.414
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 40.381  on 29  degrees of freedom
## Residual deviance: 16.183  on 26  degrees of freedom
## AIC: 24.183
##
## Number of Fisher Scoring iterations: 11
```

```
plot(foo)
curve(predict(gout, newdata = data.frame(x = x), type = "response"),
    add = TRUE)
curve(predict(gout2, newdata = data.frame(x = x), type = "response"),
    add = TRUE, col = "red")
curve(predict(gout3, newdata = data.frame(x = x), type = "response"),
    add = TRUE, col = "blue")
legend(1, 1, legend = c("linear", "quadratic", "cubic"),
    col = c("black", "red", "blue"), lty = 1)
```



Another example of TIMTOWTDI. Different ways to make the same plot.

```
plot(foo$x, foo$y, xlab = "x", ylab = "y")
attach(foo); plot(x, y); detach(foo)
with(foo, plot(x, y))
plot(foo)
```

(The last only works because the names are `x` and `y`.)

As we know from our experience with `lm`, which produces objects of class `"lm"`, and all of the generic functions that have methods for such objects, there are two ways to do hypothesis tests involving such fits.

- If the test involves whether one coefficient should be in or out of the model, we just look at the P-values printed by the R function `summary`. If we want to know whether the coefficient for `"I(x^3)"` is statistically significantly different from zero in the fit `gout3`, the P-value 0.414 says it is not.

Then, having decided that the quadratic model is to be preferred on grounds of parsimony to the cubic model, we look in the fit of that model to see whether the coefficient for `"I(x^2)"` is statistically significantly different from zero, and the P-value 0.830 says it is not.

- If the test involves more than one parameter then we use the R function `anova` to do the comparison. And it can do several different, but asymptotically equivalent (meaning they give nearly the same result when the sample size is large). Let's try them.

```
anova(gout, gout3, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x
## Model 2: y ~ x + I(x^2) + I(x^3)
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1        28     17.666
## 2        26     16.183  2   1.4831   0.4764
```

```
anova(gout, gout3, test = "LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x
## Model 2: y ~ x + I(x^2) + I(x^3)
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1        28     17.666
## 2        26     16.183  2   1.4831   0.4764
```

```
anova(gout, gout3, test = "Rao")
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x
## Model 2: y ~ x + I(x^2) + I(x^3)
##   Resid. Df Resid. Dev Df Deviance    Rao Pr(>Chi)
## 1        28     17.666
## 2        26     16.183  2   1.4831 0.4122   0.8138
```

Somewhat bizarrely, `"Chisq"` and `"LRT"` name the same test: the likelihood ratio test (LRT) whose asymptotic distribution under the null hypothesis is chi-square with degrees of freedom that is the difference in the number of parameters of the model being tested. The option `"Rao"` names a different test: the Rao test, also called the "score" test, also called the "Lagrange multiplier" test. Its asymptotic distribution under the null hypothesis is the same as the LRT.

- These P-values 0.4764 and 0.8138 would be nearly the same if we were in asymptopia (large sample size territory). They are not because the sample size is not large.
- Either of them, though, says that the larger model (the cubic one) fits these data no better than the smaller model (the linear one), hence the latter is preferred on grounds of parsimony.

In short

- to compare models that differ in one parameter, use the P-values given by the R function `summary` or perhaps by the R functions `drop1` and `add1` (which we do not illustrate), and
- to compare models that differ in more than one parameter, use the P-values given by the R function `anova`. Of course, `anova` compares any two models, even those that differ in one parameter.
- Hence one can always use `anova`, but only sometimes use `summary` or `add1` or `drop1`.

Warning: R function `anova` is actually a footgun. One condition is required for the test to be valid (other than large sample size), and that is that the models be *nested*, which means that the little model (null hypothesis) is a special case of the big model (alternative hypothesis). R function `anova` does not check this condition, so you have to know what you are doing to use it. Usually it is obvious when the models are nested (every term in the formula for the little model is also a term in the formula for the big model). But not always obvious. The real condition is that every *probability distribution* in the little model is also in the big model.

3.3.4 Confidence Intervals

Statistical inference for a GLM is slightly different from that for LM in that the sampling distributions of estimators and test statistics are approximate, "large n" distributions, where LM has exact t and F distributions, GLM have approximate normal and chi-square distributions. But that is the only difference. Everything else works the same.

For an example, here is a confidence interval for the mean for predictor value $x = 25$.

```r
conf.level <- 0.95
xnew <- 25

crit <- qnorm((1 + conf.level) / 2)
pout <- predict(gout, newdata = data.frame(x = xnew),
    se.fit = TRUE, type = "response")
pout$fit + c(-1,1) * crit * pout$se.fit
```

```
## [1] 0.7468731 1.0772870
```

This confidence interval does not work very well. Notice that its upper end point is well above the range of possible values of the parameter (the response vector is zero-or-one-valued so the means range between zero and one). There is nothing wrong with this. The confidence interval is based on "large sample approximation" and our data are not numerous.

Still, we can do somewhat better. The "large sample approximation" assumes, among other things, that the inverse link function is approximately linear in the region of the confidence interval, and we can just see from the picture that that's not correct. So if we make a confidence interval for the linear predictor and then map that to the mean value parameter scale, that should be better. At least our new confidence interval will have to be in the range of possible parameter values (because the inverse link function maps in there).

```r
tout <- predict(gout, newdata = data.frame(x = xnew), se.fit = TRUE)
invlogit(tout$fit + c(-1,1) * crit * tout$se.fit)
```

```
## [1] 0.5693274 0.9878655
```

The fact that these two confidence intervals are so different, when "large sample theory" says they are "asymptotically equivalent" (should be very close for large sample sizes) says that "large sample approximation" is not working very well for these data.

3.4 Nonparametric Regression

Nonparametric refers to statistical models that cannot be described using a finite set of parameters. They are models that are too big to be parametric.

There does not seem to be any useful way to be totally nonparametric about regression: to say that there is some relationship between response and predictors and we want to proceed making no assumptions whatsoever. There is just not enough structure to get going. Or at least no one has ever had any idea about how to proceed in this manner AFAIK.

So nonparametric regression divides in two:

- being parametric about the regression equation but nonparametric about the error distribution and

- being nonparametric about the regression equation but being parametric about the error distribution, usually making the same error distribution assumptions as LM or GLM.

3.4.1 Being Nonparametric About the Error Distribution

There isn't a way to be nonparametric about zero-or-one-valued response. A zero-or-one-valued random variable is automatically Bernoulli. So this is about continuous response. In this section we are looking at competitors that make all the same assumptions as LM except they

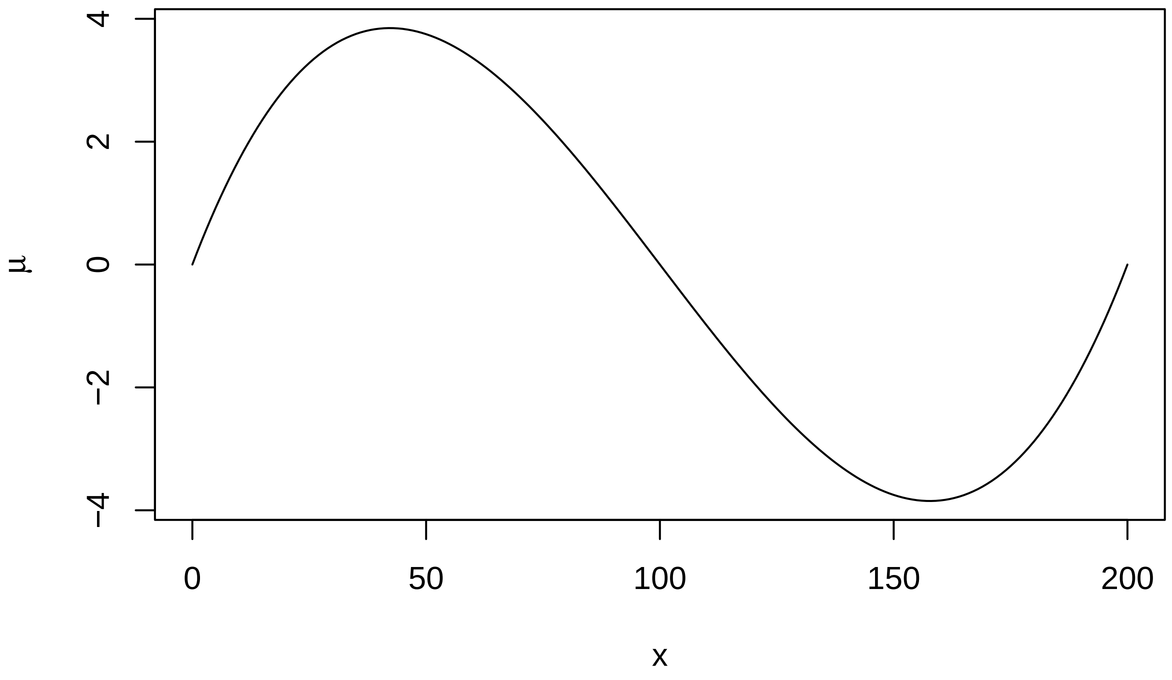
- drop the assumption that the error distribution is homoscedastic (same variance) normal and
- instead assume the error distribution is symmetric about zero and the same for all components of the response vector.

The normal distribution is symmetric. Hence we have changed normal distribution symmetric about zero into any distribution symmetric about zero. That's nonparametric because "any distribution" is too large a family to be described by a finite set of parameters.

Already in the notes about correcting errors in data we tried out two R functions that approach this problem using two different methodologies. Recall that these were the R functions `lqs` and `rlm` both of which are in the R package `MASS`, which is a "recommended" package that comes with every installation of R.

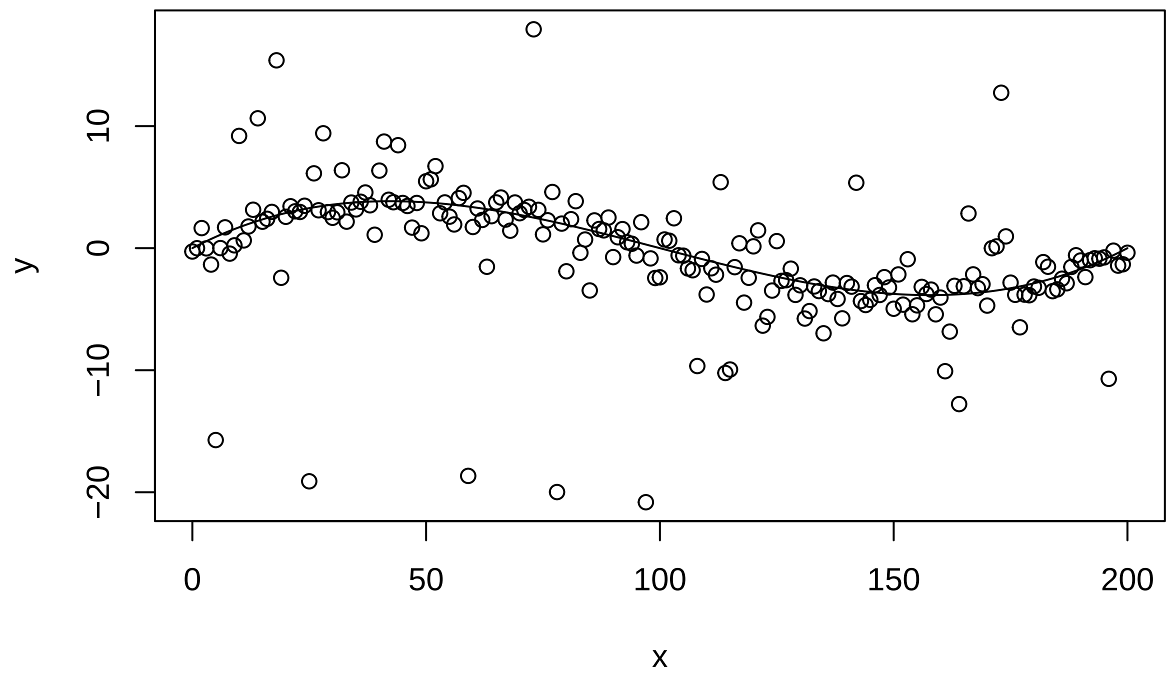
Let us make up some data that is a challenge for LM but these functions handle well. First, being a bit bored with "simple" linear regression, let's try means are a cubic function of the predictor.

```r
x <- 0:200
mu.true <- x * (x - 100) * (x - 200) * 1e-5
plot(x, mu.true, type = "l", xlab = "x", ylab = expression(mu))
```



So now we want to add "errors" having a horrible distribution. The worst distribution we know for errors (at least worst for LM) is the Cauchy distribution, which is very heavy tailed. If you have Cauchy errors, and are expecting normal errors, then it looks like there are lots of "outliers". But they aren't mistakes. The errors just have a different distribution.

```r
set.seed(42)
y <- mu.true + rcauchy(length(x))
plot(x, y)
lines(x, mu.true)
```



So now we have some data to try out methods on. But we have a problem.

> It's hard to know what lessons you are supposed to draw from a toy problem.
>
> — Me

We now have to imagine that we don't know how the data were created and just want to do regression.

Let's try LM first (even though the data are wildly inappropriate for it, which we aren't supposed to know).

```r
lout <- lm(y ~ poly(x, degree = 3))
summary(lout)
```

```
##
## Call:
## lm(formula = y ~ poly(x, degree = 3))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21.9021  -0.7643   0.3190   1.1957  16.3198
```

```
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -0.4871     0.3206  -1.519    0.130
## poly(x, degree = 3)1 -30.3337     4.5448  -6.674 2.46e-10 ***
## poly(x, degree = 3)2   4.0652     4.5448   0.894    0.372
## poly(x, degree = 3)3  20.4835     4.5448   4.507 1.13e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.545 on 197 degrees of freedom
## Multiple R-squared:   0.25,  Adjusted R-squared:  0.2386
## F-statistic: 21.89 on 3 and 197 DF,  p-value: 2.822e-12
```

Comparing with the "simulation truth", which of course we are pretending we don't know, this doesn't look good. The true regression function is cubic with coefficients

- coefficient of constant term (intercept): 0
- coefficient of first degree term: $(0 \cdot (-100) + 0 \cdot (-200) + (-100)(-200)) \times 10^{-5} = 0.2$
- coefficient of second degree term: $(0 + (-100) + (-200)) \times 10^{-5} = -0.003$
- coefficient of third degree term: 10^{-5}.

So now let's try the others.

```
qout <- lqs(y ~ poly(x, degree = 3))
summary(qout)
```

```
##               Length Class      Mode
## crit            1    -none-     numeric
## sing            1    -none-     character
## coefficients    4    -none-     numeric
## bestone         4    -none-     numeric
## fitted.values 201    -none-     numeric
## residuals     201    -none-     numeric
## scale           2    -none-     numeric
## terms           3    terms      call
## call            2    -none-     call
## xlevels         0    -none-     list
## model           2    data.frame list
```

How annoying! There is no summary method for class `"lqs"`

```
qout$coefficients
```

```
##          (Intercept) poly(x, degree = 3)1 poly(x, degree = 3)2
##           -0.3292728          -34.0221175            0.5423177
## poly(x, degree = 3)3
##           25.3250799
```

This doesn't seem to have done any better than `lm` in coming close to the simulation truth. (There are a lot of options to play with, but we won't bother.)

On to `rlm`.

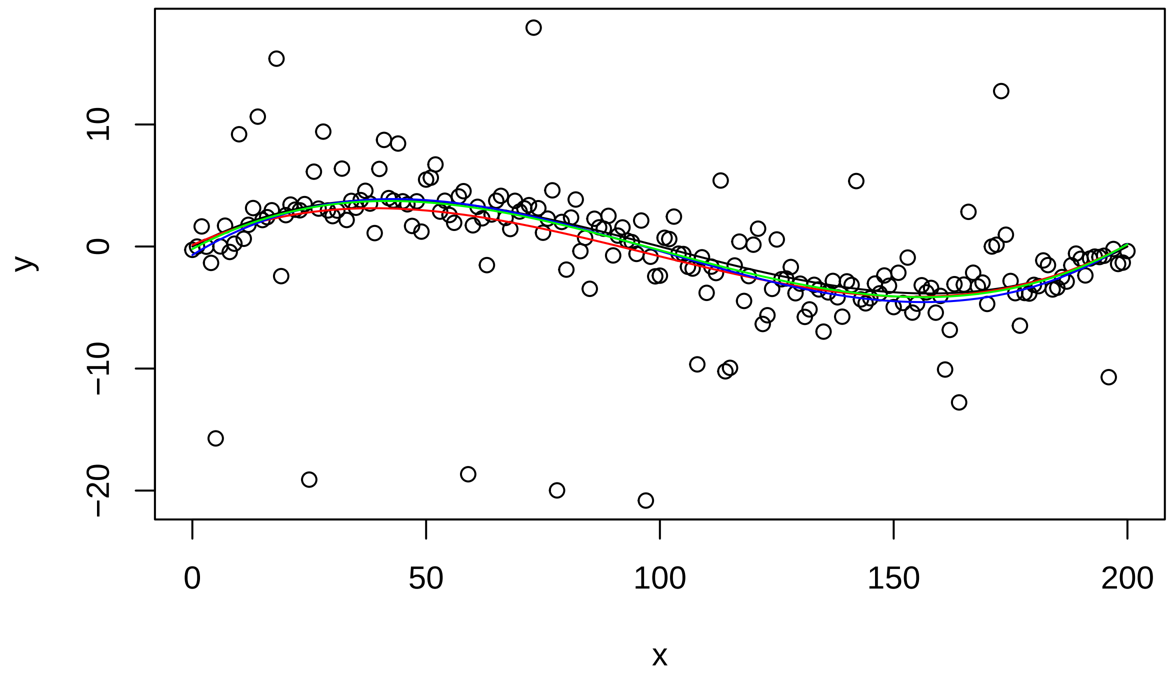
```
rout <- rlm(y ~ poly(x, degree = 3))
summary(rout)
```

```
##
```

```
## Call: rlm(formula = y ~ poly(x, degree = 3))
## Residuals:
##       Min        1Q    Median        3Q       Max
## -22.26643  -0.88096   0.05205   0.92375  16.33379
##
## Coefficients:
##                      Value    Std. Error t value
## (Intercept)           -0.2081   0.1244    -1.6736
## poly(x, degree = 3)1 -32.5148   1.7630   -18.4428
## poly(x, degree = 3)2   1.2372   1.7630     0.7017
## poly(x, degree = 3)3  22.8929   1.7630    12.9851
##
## Residual standard error: 1.354 on 197 degrees of freedom
```

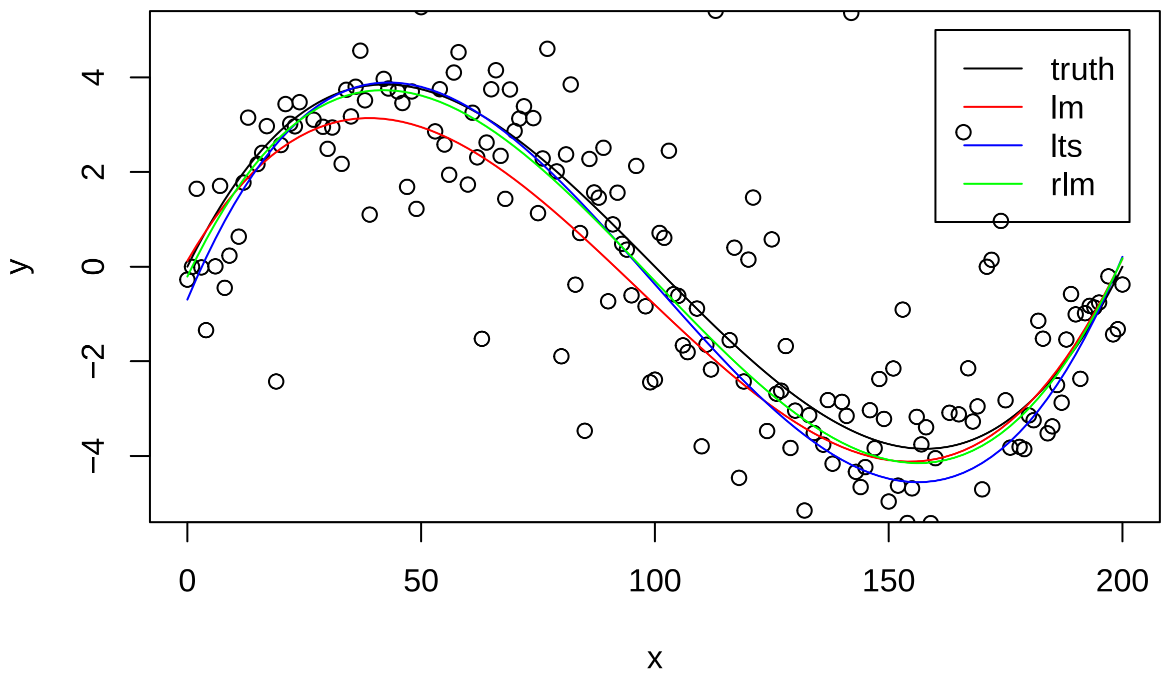
I have no idea why none of these "work". Let's look at what they did.

```
plot(x, y)
lines(x, mu.true)
curve(predict(lout, newdata = data.frame(x = x)), col = "red", add = TRUE)
curve(predict(qout, newdata = data.frame(x = x)), col = "blue", add = TRUE)
curve(predict(rout, newdata = data.frame(x = x)), col = "green", add = TRUE)
```



Oh. There isn't that much difference. We can't really see very well what is going on with the compressed range of variation of the regression curve. Let's zoom in.

```
plot(x, y, ylim = c(-5, 5))
lines(x, mu.true)
curve(predict(lout, newdata = data.frame(x = x)), col = "red", add = TRUE)
curve(predict(qout, newdata = data.frame(x = x)), col = "blue", add = TRUE)
curve(predict(rout, newdata = data.frame(x = x)), col = "green", add = TRUE)
legend(160, 5, legend = c("truth", "lm", "lts", "rlm"),
    col = c("black", "red", "blue", "green"), lty = 1)
```



I guess we have to say that `rlm` is the winner on this one toy problem, because it is closest to the simulation truth at both ends, whereas `lm` is quite a bit off near $x = 50$ and `lts` (`lqs` using the default method `lts`) is quite a bit off near $x = 150$. But, as said above, it is hard to know what lessons to draw from a toy problem.

Let's look at some confidence intervals.

```
xxx <- seq(0, 200, 50)
# simulation truth
mu.true[x %in% xxx]
```

```
## [1]  0.00  3.75  0.00 -3.75  0.00
```

```
predict(lout, newdata = data.frame(x = xxx), interval = "confidence")
```

```
##          fit       lwr       upr
## 1  0.1219031 -2.360356  2.604162
## 2  2.9500900  1.824345  4.075835
## 3 -0.8076797 -1.755966  0.140607
## 4 -4.0893201 -5.215066 -2.963575
## 5  0.1672551 -2.315004  2.649514
```

Three out of five 95% confidence intervals miss the simulation truth. No surprise because the assumptions for LM are badly violated. The only surprise is that LM doesn't do even worse.

The R function predict.lqs does not do confidence intervals. We would have to bootstrap or something of the sort to do any inference after using lqs. More on the bootstrap later.

```
predict(rout, newdata = data.frame(x = xxx), interval = "confidence")
```

```
##          fit        lwr         upr
## 1 -0.2098263 -0.9449440  0.52529147
## 2  3.6161838  3.2827957  3.94957183
## 3 -0.3056834 -0.5865173 -0.02484953
## 4 -4.0826497 -4.4160377 -3.74926165
## 5  0.1780630 -0.5570547  0.91318073
```

The R function predict.rlm apparently does confidence intervals, but I say "apparently" because this function is undocumented. When you do ?predict.rlm you are taken to the help page for the rlm function, but nothing is said about the rlm method of the generic function predict. We managed to use it above by assuming it works just like predict.lm when it is just doing predictions (no confidence intervals).

If we look at the definition of this function

```
getS3method("predict", "rlm")
```

```
## function (object, newdata = NULL, scale = NULL, ...)
## {
##     object$qr <- qr(sqrt(object$weights) * object$x)
##     NextMethod(object, scale = object$s, ...)
## }
## <bytecode: 0x55cce1900950>
## <environment: namespace:MASS>
```

We see something very mysterious. What does NextMethod do? It calls the next method for the next class, but what is that?

```
class(rout)
```

```
## [1] "rlm" "lm"
```

Oh. The next class is "lm", so it actually calls the function predict.lm but with whatever it uses from the object of class "lm" is different when the object is made by rlm.

This is not how to document code IMHO. If you have to Use the source, Luke!, that's not documentation.

3.4.2 Being Nonparametric About the Regression Equation

3.4.2.1 A Plethora of Packages

Now we are back to assuming normal errors, but we want to be nonparametric about the regression function. Suppose in our toy problem we don't know the regression equation is cubic? Since we need normal errors, let's go back and put normal errors on the same simulation truth regression function.

```
y <- mu.true + rnorm(length(x))
```

There are several nonparametric regression functions in the "core" and "recommended" packages that come with every R installation

- smooth.spline in the stats package (core)
- ksmooth in the stats package (core)
- locpoly in the KernSmooth package (recommended)
- gam in the mgcv package (recommended)
- smooth.spline in the stats package (core)

- `loess` in the `stats` package (core)
- `supsmu` in the `stats` package (core)

And there's lots more on CRAN

```
foo <- advanced_search("smooth|kernel|spline", size = 1000)
dim(foo)
```

```
## [1] 636  14
```

```
names(foo)
```

```
##  [1] "score"                "package"              "version"
##  [4] "title"                "description"          "date"
##  [7] "maintainer_name"      "maintainer_email"     "revdeps"
## [10] "downloads_last_month" "license"              "url"
## [13] "bugreports"           "package_data"
```

```
foo$package[1:50]
```

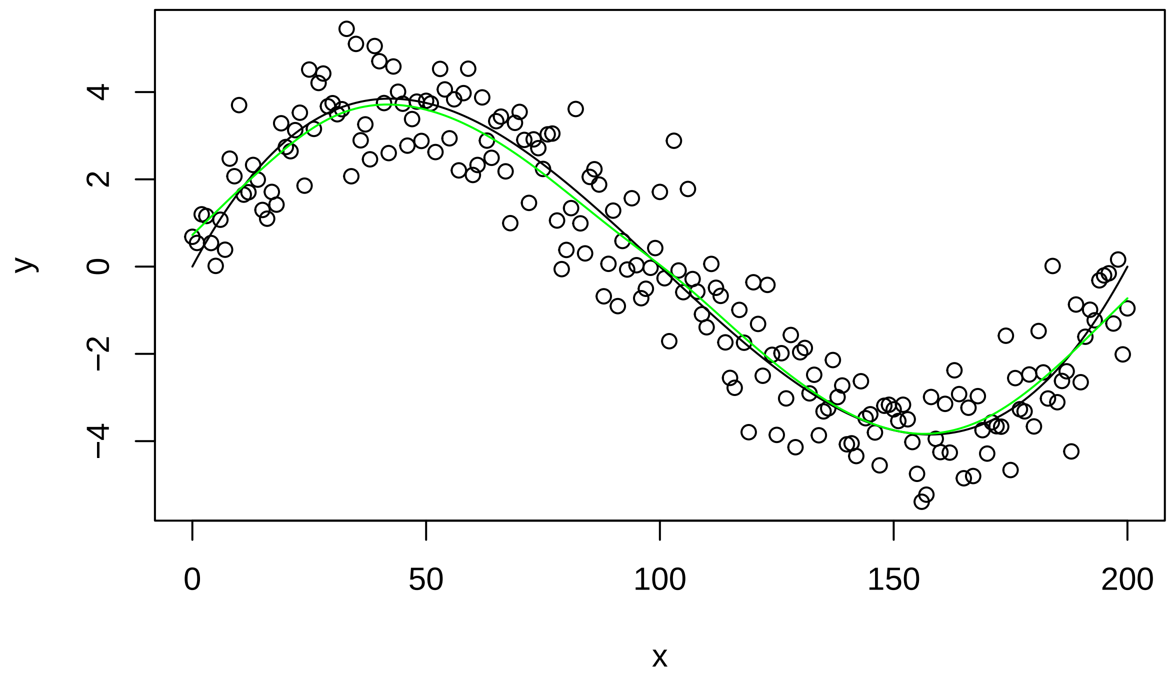
```
##  [1] "ibr"             "lmeSplines"     "KernSmooth"     "gss"
##  [5] "pspline"         "ks"             "KernSmoothIRT"  "npbr"
##  [9] "pendensity"      "GoFKernel"      "cobs"           "kernelboot"
## [13] "lokern"          "KENDL"          "bigsplines"     "adass"
## [17] "JOPS"            "assist"         "tvReg"          "mda"
## [21] "smoothAPC"       "kzs"            "kernelPhil"     "eks"
## [25] "npreg"           "sm"             "qualypsoss"     "kdecopula"
## [29] "npregfast"       "binsmooth"      "HQM"            "mgss"
## [33] "eBsc"            "smoothROCtime"  "btb"            "bshazard"
## [37] "FastGaSP"        "snfa"           "sparr"          "SOP"
## [41] "Conake"          "iosmooth"       "kequate"        "pgam"
## [45] "quantregGrowth"  "BTSPAS"         "boostmtree"     "rstpm2"
## [49] "np"              "DCSmooth"
```

I have no idea how many of these are true positives (really do nonparametric regression).

3.4.2.2 R function `smooth.spline`

Rather than tackle what is obviously a huge subject. Let's just illustrate one.

```
plot(x, y)
lines(x, mu.true)
sout <- smooth.spline(x, y, all.knots = TRUE)
lines(sout, col = "green")
```



Good job! Note that we did not tell `smooth.spline` that the regression function was cubic. We didn't tell it anything about the regression function. Nevertheless, it comes pretty close.

Statistical inference after "smoothing", which is what this is often called rather than "nonparametric regression" is also problematic. We would have to bootstrap to get confidence intervals. Some smoothing methods do come with confidence intervals, but those intervals assume the degree of smoothness of the regression function is known rather than estimated from the data. So they are bogus when, as here, we don't specify a degree of smoothness (by using the optional argument `df`).

3.4.2.3 R function `gam` in package `mgcv`

Let's try one more. The R package `mgcv`, which is a "recommended" package that comes with every R installation, fits much more general models than `smooth.spline`. It fits, so-called additive models, where the mean vector has the form

$$\mu = g_1(x_1) + g_2(x_2) + \cdots + g_k(x_k)$$

where x_1, x_2, and x_k are different predictors and g_1, g_2, and g_k are different smooth functions that are to be figured out by the computer. So it does nonparametric regression with multiple predictors.

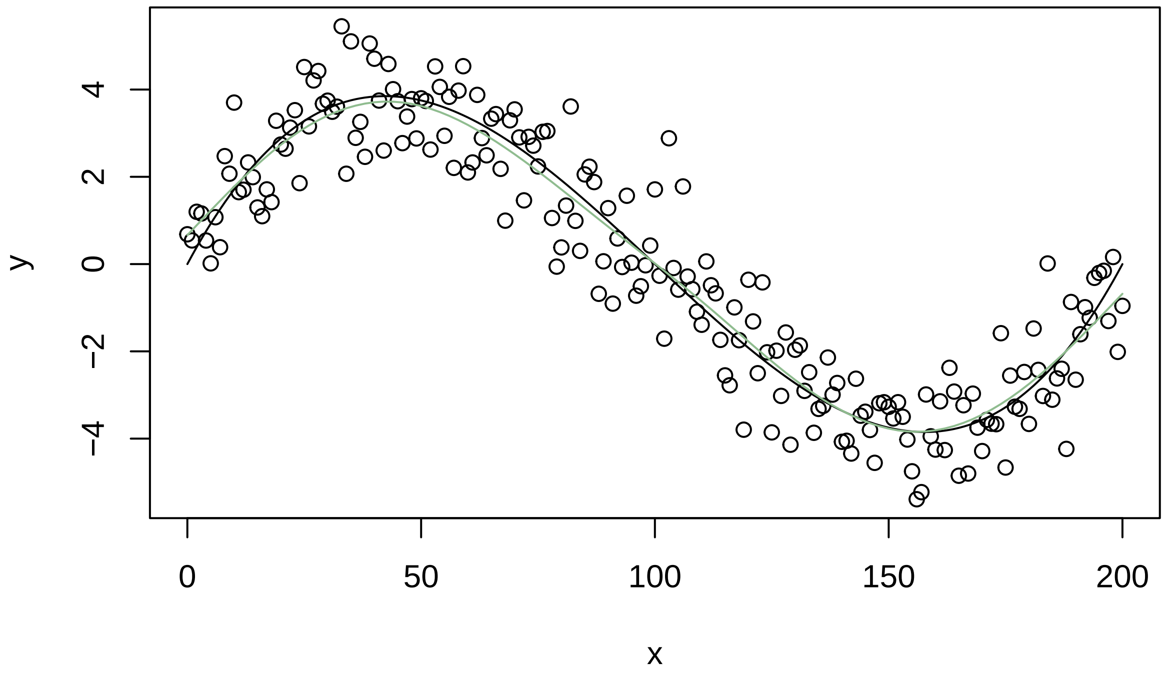
But we are just going to illustrate one predictor using the same data as in the preceding section.

```r
plot(x, y)
lines(x, mu.true)
aout <- gam(y ~ s(x, bs="cr"))
summary(aout)
```

```
##
## Family: gaussian
## Link function: identity
```

```
##
## Formula:
## y ~ s(x, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.03357    0.06625  -0.507    0.613
##
## Approximate significance of smooth terms:
##        edf Ref.df     F p-value
## s(x) 6.335  7.472 219.6  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.892   Deviance explained = 89.5%
## GCV = 0.91575  Scale est. = 0.88233   n = 201
```

```r
curve(predict(aout, newdata = data.frame(x = x), type = "response"),
    add = TRUE, col = "darkseagreen")
```



Unlike, some other functions we have looked at, the R function produces objects of class

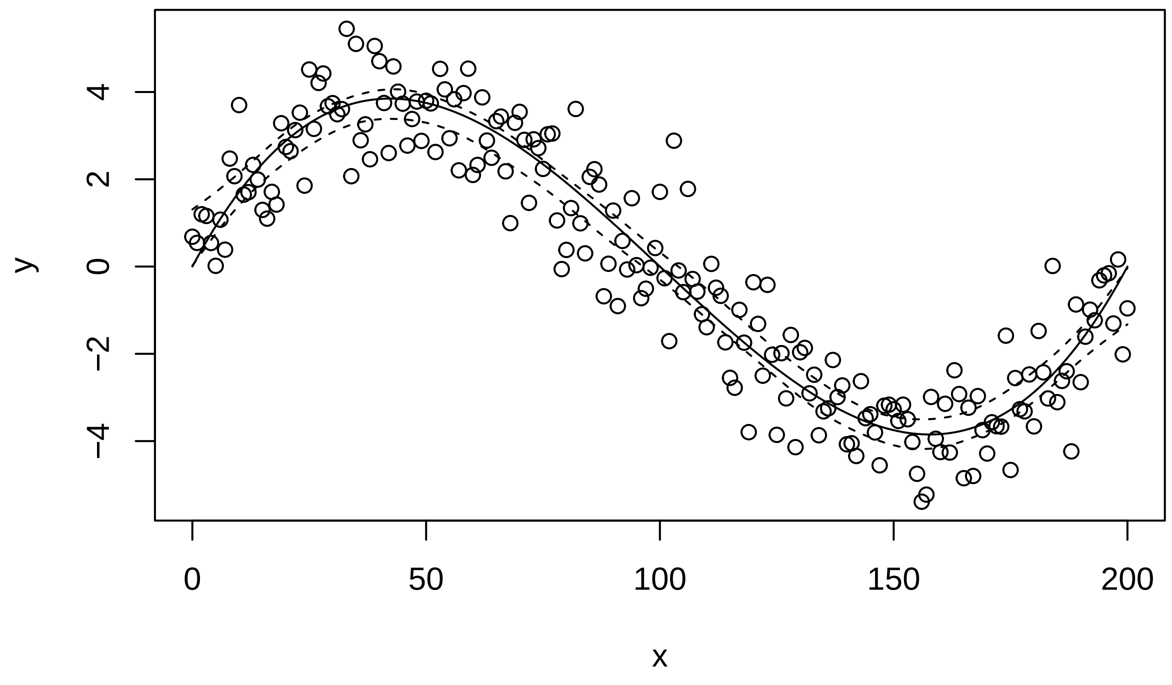
```r
class(aout)
```

```
## [1] "gam" "glm" "lm"
```

so there is a rich class of methods for generic functions. We can make confidence intervals.

```
conf.level <- 0.95

plot(x, y)
lines(x, mu.true)
fred <- function(x) {
    foo <- predict(aout, newdata = data.frame(x = x),
        type = "response", se.fit = TRUE)
    crit <- qnorm((1 + conf.level) / 2)
    foo$fit + crit * foo$se.fit
}
curve(fred, lty = "dashed", add = TRUE)
fred <- function(x) {
    foo <- predict(aout, newdata = data.frame(x = x),
        type = "response", se.fit = TRUE)
    crit <- qnorm((1 + conf.level) / 2)
    foo$fit - crit * foo$se.fit
}
curve(fred, lty = "dashed", add = TRUE)
```



This seems pretty good. Even though `gam` and `predict.gam` know nothing about the true unknown regression function, and even though these confidence intervals

- assume that the degree of smoothness is known rather than estimated, and
- are not corrected for simultaneous coverage,

they mostly (nearly everywhere) cover the simulation truth.

4 Statistical Models of Other Kinds

Not all statistical models are regression like, with a response and a predictor. You know from intro stats that the simplest models just have one variable. But there is still statistical inference.

There can also be very complicated models with many parameters to estimate (but no single variable picked out as being a "response" to "predict").

More on these later. First we get some other subjects out of the way.