UWEO StatR201 Lecture 4

Generalized Linear Models, and Splines

Assaf Oron, February 2013

Tonight's Menu

Tonight is a study in contrasts. If we look for a common theme, then it is **moderately extending the regression framework** to account for more dataset types and for stronger deviations from linearity and Normality.

First, **Generalized Linear Models (GLMs)**:

- Exponential Family distributions
- Generalized Linear Models and examples
- Overdispersion and Quasi-Likelihood GLMs, more examples

Then, **Splines:**

- What they are
- How to add them to an R regression.

Finally, our - R trick/annoyance

Exponential Family Distributions

The Normal is such a great distribution - it's the limit distribution of the sample mean, it's the "natural" distribution for regression, etc. Can we spread some of this goodness around, by identifying a wider class of distributions as "family relatives" of the Normal?

This paves the way to the **exponential family**, which includes any distribution with q parameters that can be written as:

$$f(y) = \exp\left[\sum_{j=1}^{q} a_j(y)b_j(\theta) + c(y) + d(\theta)\right],$$

with j the parameter index, and a, b, c, d algebraic functions (some of them may be degenerate, i.e., zero). Let's illustrate this via the Normal:

$$f(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(y-\mu)^2}{2\sigma^2}\right].$$

Say what? How can this fit into that? Just watch:

$$f(y) = \exp\left[\left(y\,\frac{\mu}{\sigma^2} - y^2\,\frac{1}{2\sigma^2}\right) - 0.5\log 2\pi - \left(2\log\sigma + \frac{\mu^2}{2\sigma^2}\right)\right].$$

Note that the all-important functions $b_1(\theta), b_2(\theta)$ do not have to be of the original "clean" parameters μ, σ. The actual arguments appearing instead are known as **the "natural parameters"** for this distribution.

Exponential Family Distributions

Is it worth the trouble? Big time. First, properties of exponential family MLEs can be easily derived. Second, this is the basis for GLMs.

But right now: try to force the Binomial (distribution shown in Lecture 3) into an exponential family format, and find its natural parameter.

Other common distributions belonging to the exponential family:

- Gamma (including the Exponential)
- Poisson
- Multinomial and ordered-Multinomial
- Negative-Binomial (what's *that*?)

Generalized Linear Models (GLMs)

Developed some 40 years ago (the original article is on the class website), GLMs focus on a single parameter θ, with the rest (in the multi-parameter case) becoming a **nuisance parameter(s) ϕ.** The GLM-friendly exponential family form is a slight variation on the one shown in the previous slide:

$$f(y) = \exp\left[\frac{a(y)b(\theta) + c(y) + d(\theta)}{h_1(\phi)} + h_2(y, \phi)\right],$$

with h_1, h_2 again algebraic functions. To make things even less fun notation-wise, this being regression we in fact have **a vector of observations, each of which has a different θ_i - but all obeying the same form.** How is that workable, and what is the relation to regression?

The relation is that the various θ_i's are functions of covariates, in the following manner (switching to matrix-vector notation):

$$\eta(\theta) = g(E[Y]) = \mathbf{X}\beta,$$

where g is (you guessed it) another algebraic function **-** but one that is **monotone and differentiable.** It is known as **the link function.** The most natural choice for g is one that mimics the function $b(\theta)$ in the distribution form as written above (e.g., if $b(\theta) = \log\theta$, then $g(E[Y]) = \log(E[Y])$ is the most natural choice).

So the θ_i are really a "shell", if you will, hiding within them the regression parameters of interest.

Important note: for this course's needs, you don't have to learn to manipulate or (heavens forbid) memorize this formula - just to be familiar with the general idea.

GLMs and Likelihood

If this format reminds you of "vanilla" linear regression, you're on the right track. The $\eta = \mathbf{X}\beta$ equation, with the underlying exponential family probablity model connecting it to the distribution of observations, is a Likelihood, and - using **Iterated Weighted Least Squares (IWLS)** - the MLEs $\hat{\beta}$ can be found (we've already encountered IWLS once: that's how robust regression is fit).

Notes:

- Of course, "vanilla" linear regression is a special case of a GLM, and it does not require IWLS b/c it has a closed-form least-squares solution.
- The MLEs for the "nuisance" parameters ϕ are found simultaneously "on the side", just like $\hat{\sigma^2}$ is found in linear regression.
- Because it's a likelihood, nested models can be compared using LRTs. To simplify matters, GLM's developers introduced the term **Deviance**, $D = -2 \log l$ (l being the log-likelihood) - so that differences in the deviance between nested models are (asymptotically) Chi-Square random variables under the Null.

Questions? (online/in-class)

GLM: Toy Example

```
library(lmtest)
glmod2 = glm(log10(accel) ~ log10(dist) + mag, data = attenu, family = gaussian)
summary(glmod2)
```

```
##
## Call:
## glm(formula = log10(accel) ~ log10(dist) + mag, family = gaussian,
##     data = attenu)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0343  -0.1852   0.0362   0.2153   0.6621
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.7161     0.1909   -3.75  0.00024 ***
## log10(dist)  -0.9047     0.0470  -19.24  < 2e-16 ***
## mag           0.1490     0.0337    4.42  1.7e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.091)
##
##     Null deviance: 50.911  on 181  degrees of freedom
## Residual deviance: 16.289  on 179  degrees of freedom
## AIC: 85.23
##
## Number of Fisher Scoring iterations: 2
```

- `gaussian` is the default family, so we didn't really have to explicitly specify it;
- Compare the summary to a summary of a similar `lm` run, and find the differences;
- The returned object inherits from the `lm` class, but has about twice as many components.

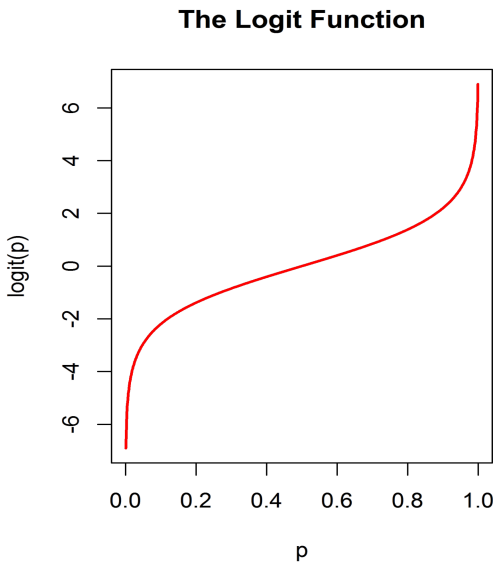
Logistic Regression and the Logit Link

Besides `glm`ing linear regression, by far the most widely used GLM is of course **logistic regression,** used for binary responses and assuming the Bernoulli/Binomial model.

Logistic regression uses its natural link function, which is (as you found earlier today) **the logit link:**

$$\text{logit}(p) = \log p - \log(1-p) = \log \frac{p}{1-p}$$

```
curve(log((x)/(1 - x)), col = 2, lwd = 2, main = "The Logit Function", n = 1000,
    xlab = "p", ylab = "logit(p)")
```



The logit curve is essentially the sigmoid logistic curve known from other fields - transposed. One useful "side effect" is that **on the model-fitting scale, the y response (usually denoted η) has an unrestricted range.**

Logistic Regression, Log-Odds and Odds-Ratio

If interpreting linear regression estimates, esp. with transformations and interactions, can be tricky - you can imagine how it is with logistic regression.

Fortunately, the $p/(1-p)$ at the link's core does have a real-world interpretation known to many: **the Odds** *(yes, just like in betting; check it out!)*. To clarify: the odds are the ratio between the probablities of the two options. For example, the odds for "heads" in a fair coin are 1.

In any case, the full logistic-regression link transformation is known either as logit, or **the log-odds.**

When interpreting effect estimates, we usually exponentiate the log-odds and report the effect as an **Odds Ratio (OR).**

For example, $\hat{\beta} = 0.2$ translates to an OR of $\exp 0.2 = 1.22$, or an increase by a factor of 1.22 in the odds of a "positive" response per unit increase in the corresponding predictor x.

Logistic Regression: A Not-Toy Example

```
challenger = read.csv("../Datasets/challenger.csv", as.is = TRUE)
dim(challenger)  # Data from Challenger launches and experiments before the tragic 1986 launch
```

```
## [1] 23  6
```

```
table(challenger$atrisk)  # Each of the 23 entries had 6 o-rings
```

```
##
##  6
## 23
```

```
table(challenger$Eroded)  # This variable counts how many were eroded
```

```
##
##  0  1  2
## 17  5  1
```

```
table(challenger$tempF)  # This is the critical variable. The temperature on the morning of the fatal launch was 31F.
```

```
##
## 53 57 58 63 66 67 68 69 70 72 73 75 76 78 79 81
##  1  1  1  1  1  3  1  1  4  1  1  2  2  1  1  1
```

Logistic Regression: A Not-Toy Example

With logistic regression, descriptives and diagnostics are often a challenge. With a categorical predictor, you can just cross-tabulate. With a continuous one, you can just **cut:**

```
challenger$tempcut = cut(challenger$tempF, c(50, 60, 69.5, 75.5))
tapply(challenger$Eroded, challenger$tempcut, sum)/tapply(challenger$atrisk,
    challenger$tempcut, sum)
```

```
##     (50,60]   (60,69.5] (69.5,75.5]
##     0.22222     0.02381     0.04167
```

```
tapply(challenger$Eroded, challenger$Pressure, sum)/tapply(challenger$atrisk,
    challenger$Pressure, sum)
```

```
##      50     100     200
## 0.02778 0.00000 0.06667
```

O-ring failure rates increase at low temperatures… remember not all 6 have to fail - it's enough for a few to fail to generate a disaster.

Not much seem to go on with pressure, though.

Logistic Regression: A Not-Toy Example

```
summary(glm(cbind(Eroded, Intact) ~ I(tempF - 70) + Pressure, data = challenger,
    family = binomial))$coef
```

```
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -4.109355   1.398492 -2.9384 0.003299
## I(tempF - 70) -0.169736   0.063915 -2.6557 0.007915
## Pressure       0.002585   0.008485  0.3046 0.760654
```

- In logistic regression, the y can be provided either as a pair of columns indicating *'yes', 'no'* responses - or as a single vector taking on values between 0 and 1 (most commonly just 1/0 or `TRUE/FALSE`).
- In the two-column case, the model automatically weights each row by the corresponding number of observations.

Logistic Regression: A Not-Toy Example

Let's repeat without pressure in the model, then interpret:

```
summary(glm(cbind(Eroded, Intact) ~ I(tempF - 70), data = challenger, family = binomial))$coef
```

```
##               Estimate Std. Error z value  Pr(>|z|)
## (Intercept)    -3.7475    0.64318  -5.827 5.659e-09
## I(tempF - 70)  -0.1795    0.05822  -3.083 2.049e-03
```

```
expit = function(x) exp(x)/(1 + exp(x))
expit(-3.7475)
```

```
## [1] 0.02303
```

At an ambient temperature of 70F, the probability that a Challenger shuttle o-ring would erode during takeoff is estimated as about 2.3%, based on data collected prior to the 1986 explosion. With each 1F decrease in temperature, the odds of erosion increase by a factor of 1.20 (1.07,1.34; p=0.002).

Find the estimated probability at 31F.

Questions? (online/in-class)

Over-Dispersion and Quasi-Likelihood

GLMs allow us to use (relatively) realistic Likelihood estimates for more problems. However, sometimes these models are still quite limited.

In particular, **both the Binomial and Poisson Likelihoods have a single parameter - meaning that the $E[Y]$ and $Var[Y]$ are algebraic functions of each other.**

That is often too restrictive, because in reality some noise often enters this relationship. Ignoring this noise might bias our estimate of the mean (i.e., the $\hat{\beta}$s), the SE, or both.

Solution: **invent an 'over-dispersion' parameter (=denominator in the GLM distribution formula), like the other distributions have.** You get an equation that looks just like a GLM equation, and you can solve it via IWLS just the same.

One obvious problem, is that **this is not a Likelihood anymore. It is known as Quasi-Likelihood. And the quasi-Likelihood GLM becomes a sermiparametric model.** This means that we only assume a form for the mean and variance, but not a full probability model for the observations.

Fortunately, under fairly general conditions (which boil down to assuming that the mean and variance behave reasonably well), the resulting estimates share the same asymotptic properties as 'plain vanilla' MLEs (consistency, Normality, etc.).

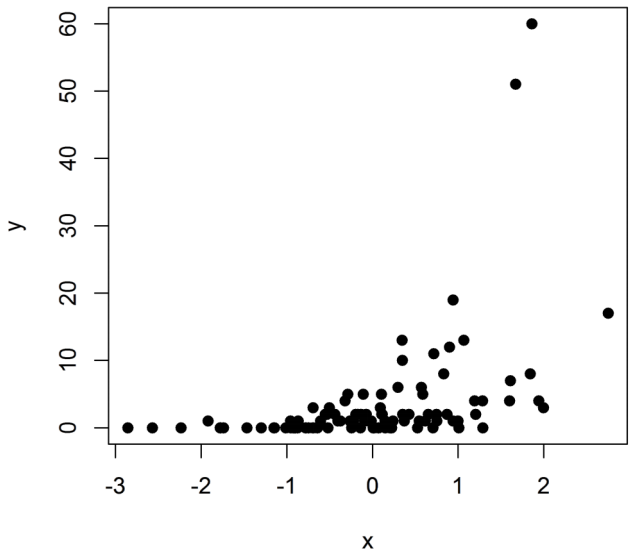
The Assumptions-Properties-Behavior monster strikes again…

Over-Dispersion and Quasi-Likelihood: Toy Example

```
x = rnorm(100)  # log-normal
y = rpois(100, lambda = exp(x + rnorm(100)))  # 2nd layer of randomization
summary(glm(y ~ x, family = poisson))$coef
```

```
##             Estimate Std. Error z value  Pr(>|z|)
## (Intercept)   0.6729     0.0790   8.517 1.637e-17
## x             1.0818     0.0541  19.994 6.170e-89
```

```
plot(y ~ x, pch = 19)
```




```
summary(glm(y ~ x, family = quasipoisson))  # only SE changes here...
```

```
##
## Call:
## glm(formula = y ~ x, family = quasipoisson)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.197  -1.503  -0.834   0.286   8.840
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.673      0.187    3.59  0.00052 ***
## x              1.082      0.128    8.43  3.1e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 5.63)
##
##     Null deviance: 860.79  on 99  degrees of freedom
## Residual deviance: 453.11  on 98  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 5
```

GLMs: Pros and Cons

Pros:

- GLMs extend the advantages of linear regression (asymptotic Normality, efficiency, etc.) into a wider set of modeling challenges, while providing relatively realistic models and remaining within a likelihood framework.
- GLMs enable the modeling of outcomes that might be otherwise hard to deal with (binary, count, etc.)
- GLMs enable the modeling of various mean-variance relationships, with great flexibility with both the link and the dispersion parameter. You can see all the options under `?family`. In particular, the option named just `quasi` is a free-for-all **semiparametric** regression rather than a 'proper' Likelihood one.

Cons:

- As the actual outcome becomes less smooth, diagnostics become harder (the worst is logistic regression).
- Sometimes it's simpler just to transform the y and remain within the framework of linear regression. Comparing the two is often worth the extra effort.

Questions? (online/in-class)

Splines

We already saw how both x and y can be transformed to fit an obviously nonlinear relationship: log, polynomial, any algebraic function.

If the relationship is complicated, in principle we can represent x as a higher- and higher-order polynomial. However, polynomials are **universal:** the same polynomial would have to hold over the entire range of x. Even in a deterministic setting, finding a higher-order polyomial to fit the entire range is a headache. When you enter noise into the picture, it becomes essentially impossible.

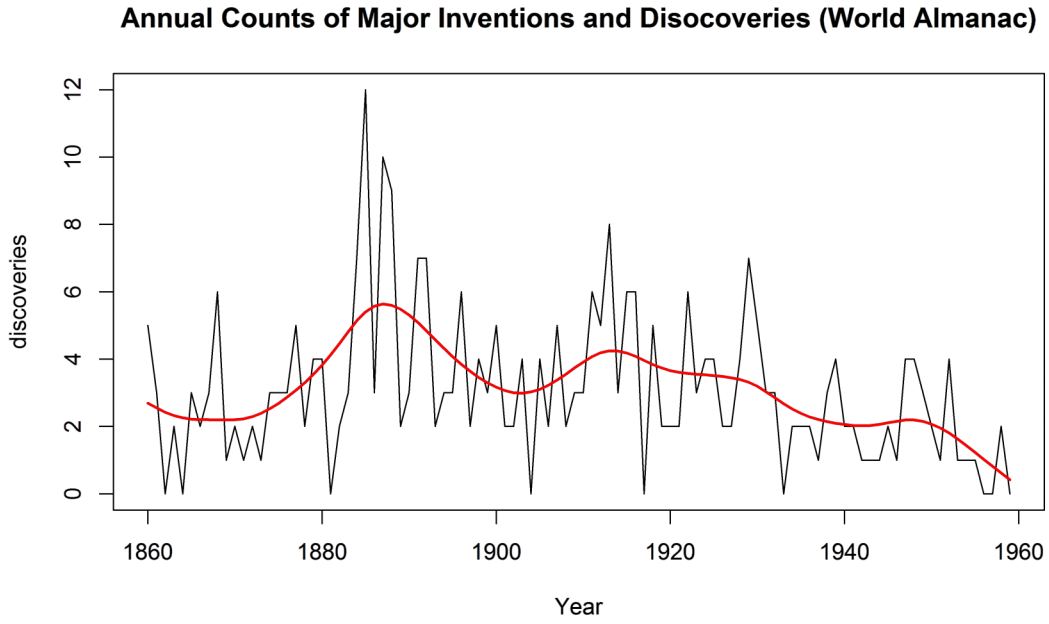
This is where splines (and other similar systems) come to our aid. **Splines are systems of piecewise-polynomial curves.** The user can control the polynomial order, as well as the number of pieces and the way each piece connects.

Splines are used perhaps most often for effects such as age and time, when we know there is a “typical” curve, but it does not have to follow some specific functional form.

The first part of Hastie et al. Chapter 5 is a great introduction to splines.

Splines

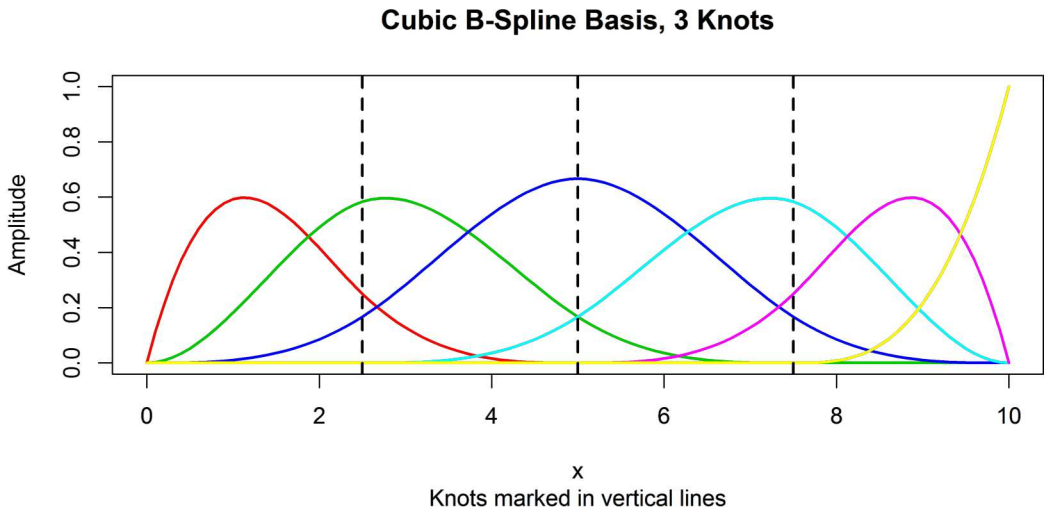
```
plot(discoveries, main = "Annual Counts of Major Inventions and Disocoveries (World Almanac)",
    xlab = "Year")
sline = smooth.spline(discoveries ~ time(discoveries))
lines(sline$x, sline$y, col = 2, lwd = 2)
```



The smoothing spline is an interesting and unusual beast, available right on R's standard installation. However, being unusual, it is **not** the basic one available for regression. Rather, we use functions that form a **spline basis**, available in the library `splines` – which like `MASS` and `lattice` is installed with R but not loaded automatically into each session.

A B-Spline Basis

```
library(splines)
x = seq(0, 10, 0.1)
mybase = bs(x, knots = c(2.5, 5, 7.5))
plot(c(0, 10), c(0, 1), type = "n", main = "Cubic B-Spline Basis, 3 Knots",
     ylab = "Amplitude", xlab = "x", sub = "Knots marked in vertical lines")
for (a in 1:6) lines(x, mybase[, a], col = a + 1, lwd = 2)
abline(v = c(2.5, 5, 7.5), lty = 2, lwd = 2)
```

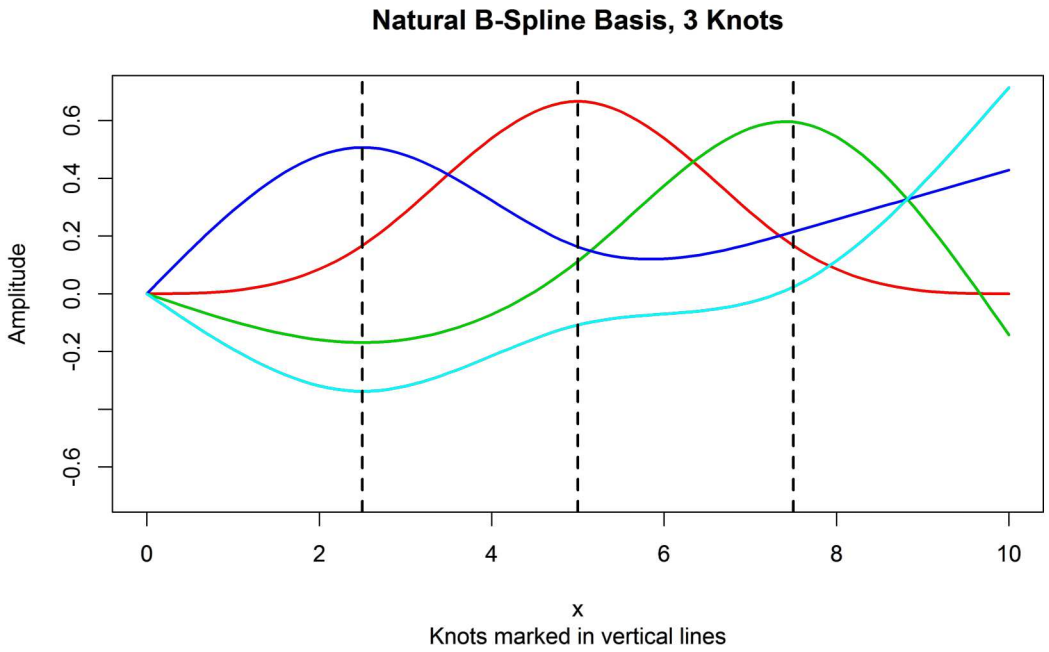


As you can see, B-spline basis has a logic and symmetry to it. It is a **complete basis** just like, say, the Fourier transform basis or (more closely resembling) any polynomial basis. The actual fitted spline curve will be a **linear combination** of these basis functions.

- **Knots** are the points dividing the range of x into pieces
- The **degree:** even though some books e.g. Hastie et al., insist that the degree is 1 for piecewise-constant and hence 4 for cubic splines - R mercifully follows common-sense and uses the same convention as for ordinary polynomials (so cubic is 3rd degree).
- The **smoothness** is how the pieces connect at the knots. For B-splines, derivatives up to `degree-1` are continuous at the knots. The most commonly used splines are cubic, which means they are second-derivative continuous everywhere. This is R's default.
- In regression, you would just place a `bs(varname,knots=(whatever))` on the RHS of the formula. That will generate a matrix. Each column will show up as a row in the regression `summary`. This is why it is more common to specify the `df` argument rather than the `knots`.

A *Natural* B-Spline Basis

```
mybase = ns(x, knots = c(2.5, 5, 7.5))
plot(c(0, 10), c(0, 1), type = "n", main = "Natural B-Spline Basis, 3 Knots",
    ylab = "Amplitude", xlab = "x", sub = "Knots marked in vertical lines",
    ylim = c(-0.7, 0.7))
for (a in 1:4) lines(x, mybase[, a], col = a + 1, lwd = 2)
abline(v = c(2.5, 5, 7.5), lty = 2, lwd = 2)
```



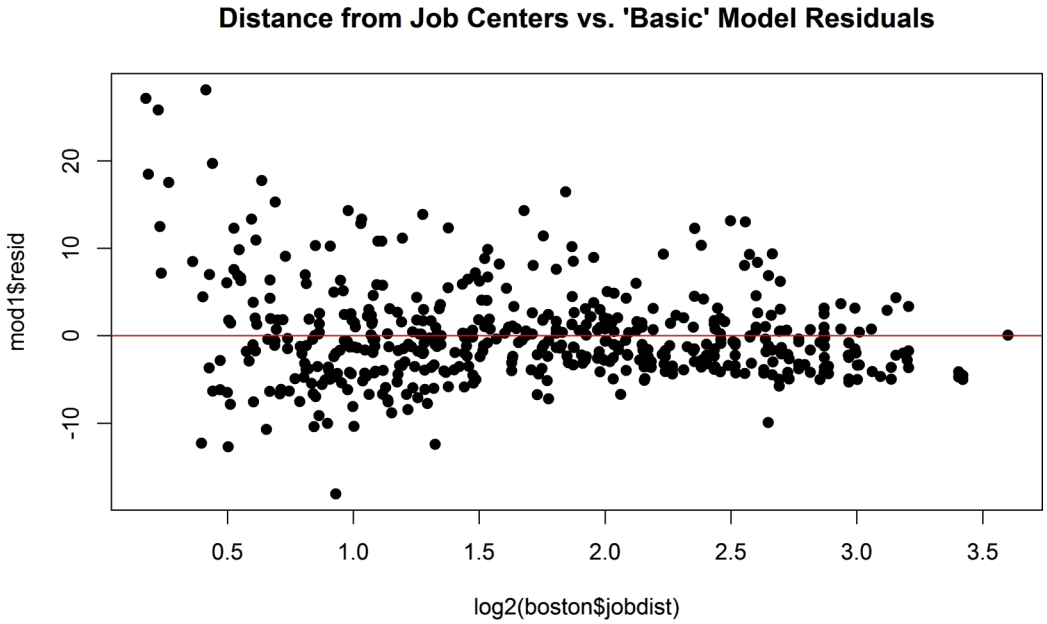
When using cubic splines, most people prefer the variant known as **natural splines**, called by `ns`. It assumes the curve is linear on the domain's extreme outer boundary – thereby saving 4 degrees of freedom (and also somewhat stabilizing the ends of the range).

How do you find the degrees of freedom?

- For `bs()`, `df=degree+knots`, so `knots=df-degree` (when you need to figure out how many knots your domain will be cut into).
- For `ns()`, `df=degree+knots-2=knots+1`, so `knots=df-1`.
- These calculations assume that the constant term is omitted from the splines, meaning that like for `factor` variables, it is "swallowed" into the general intercept (one can insist on having a separate spline intercept, and pay 1 additional d.f. for it).
- Note however, that since `ns()` is constrained to be linear on the edges, 1 d.f. is **identical** to a linear covariate, while 2 d.f. is quadratic with 1 knot.

Splines in Regression: Example (a.k.a. Boston is still alive!)

```
library(lmtest)
mod1 = lm(homeval ~ lowSES + rooms, data = boston)
jobperm = order(boston$jobdist)  ### This will be useful soon...
plot(log2(boston$jobdist), mod1$resid, pch = 19, main = "Distance from Job Centers vs. 'Basic' Model Residuals")
abline(h = 0, col = 2)
```



Once distance from job centers is log-transformed, a strong association seems apparent: for a given poverty rate and home size, home value residuals decrease with increasing distance from job centers - at first sharply, then much more slowly. **A highly nonlinear relationship, it seems.**

Splines in Regression: Naive Approach

```
mod2 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), df = 2), data = boston)
mod3 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), df = 3), data = boston)
mod4 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), df = 4), data = boston)
mod5 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), df = 5), data = boston)
mod6 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), df = 6), data = boston)
mod7 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), df = 7), data = boston)
mod8 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), df = 8), data = boston)
mod9 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), df = 9), data = boston)
lrtest(mod1, mod2, mod3, mod4, mod5, mod6, mod7, mod8, mod9)
```

```
## Likelihood ratio test
##
## Model 1: homeval ~ lowSES + rooms
## Model 2: homeval ~ lowSES + rooms + ns(log2(jobdist), df = 2)
## Model 3: homeval ~ lowSES + rooms + ns(log2(jobdist), df = 3)
## Model 4: homeval ~ lowSES + rooms + ns(log2(jobdist), df = 4)
## Model 5: homeval ~ lowSES + rooms + ns(log2(jobdist), df = 5)
## Model 6: homeval ~ lowSES + rooms + ns(log2(jobdist), df = 6)
## Model 7: homeval ~ lowSES + rooms + ns(log2(jobdist), df = 7)
## Model 8: homeval ~ lowSES + rooms + ns(log2(jobdist), df = 8)
## Model 9: homeval ~ lowSES + rooms + ns(log2(jobdist), df = 9)
##   #Df LogLik Df Chisq Pr(>Chisq)
## 1   4  -1583
## 2   6  -1572  2 22.21    1.5e-05 ***
## 3   7  -1555  1 33.42    7.4e-09 ***
## 4   8  -1544  1 21.89    2.9e-06 ***
## 5   9  -1539  1 10.88    0.00097 ***
## 6  10  -1537  1  2.25    0.13364
## 7  11  -1537  1  1.78    0.18251
## 8  12  -1534  1  4.63    0.03139 *
## 9  13  -1533  1  1.93    0.16460
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Splines in Regression: Naive Approach

The previous slide's LRTs behave a bit strangely (between the 5-6-7-8 d.f. models), for a good reason.

Strictly speaking, just increasing the d.f. of a spline does not generate a series of nested models. The tests are legit only if the simpler model's knots are a subset of the larger model's, or if the only difference is that the larger model has a higher degree or less constraints (e.g., `ns` vs. `bs` on the same set of knots).

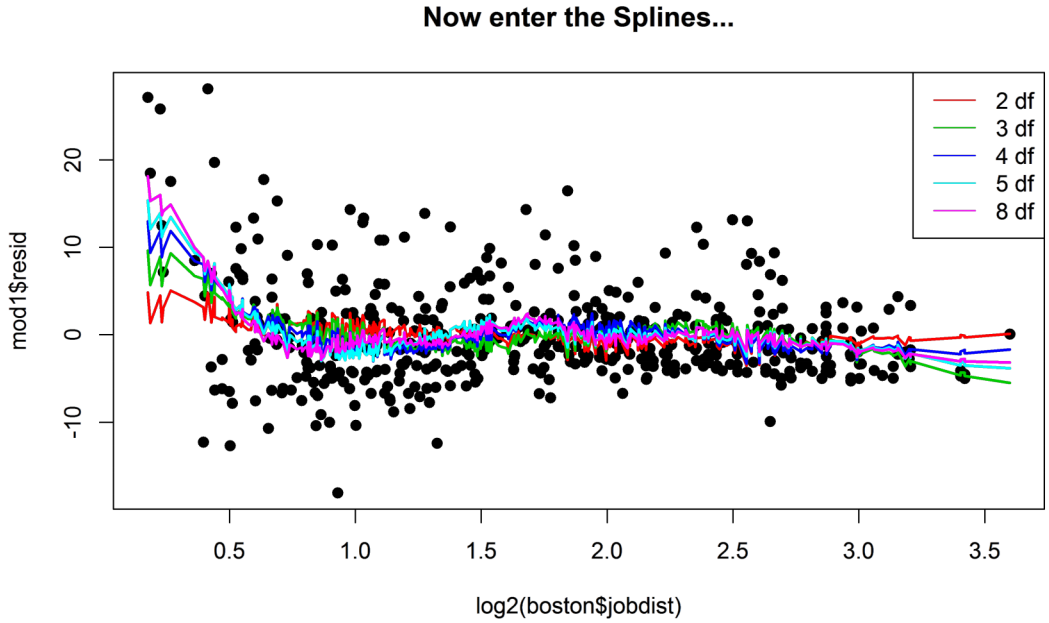
R's default when provided the d.f.'s, is to create the appropriate number of knots at **evenly-spaced quantiles of X.** Natural splines have `df-1` knots, splitting the domain into `df` parts. So the model with 2 default d.f. is nested inside the 4 and 6 d.f. - but not inside the 3 and 5 d.f. models. And the 5 d.f. model is not nested in any of the larger one it was tested against.

Also, **just letting the knots be chosen by the default is wasteful, if we know that the nonlinearity is concentrated in certain regions.**

Let's see how the model fit looks like for the cascade we generated.

Splines in Regression: Example (a.k.a. Boston is still alive!)

```
plot(log2(boston$jobdist), mod1$resid, pch = 19, main = "Now enter the Splines...")
lines(log2(boston$jobdist[jobperm]), mod2$fitted[jobperm] - mod1$fitted[jobperm],
    col = 2, lwd = 2)
lines(log2(boston$jobdist[jobperm]), mod3$fitted[jobperm] - mod1$fitted[jobperm],
    col = 3, lwd = 2)
lines(log2(boston$jobdist[jobperm]), mod4$fitted[jobperm] - mod1$fitted[jobperm],
    col = 4, lwd = 2)
lines(log2(boston$jobdist[jobperm]), mod5$fitted[jobperm] - mod1$fitted[jobperm],
    col = 5, lwd = 2)
lines(log2(boston$jobdist[jobperm]), mod8$fitted[jobperm] - mod1$fitted[jobperm],
    col = 6, lwd = 2)
legend("topright", legend = paste(c(2:5, 8), "df"), lty = 1, col = 2:6)
```



Splines in Boston... More Carefully Now

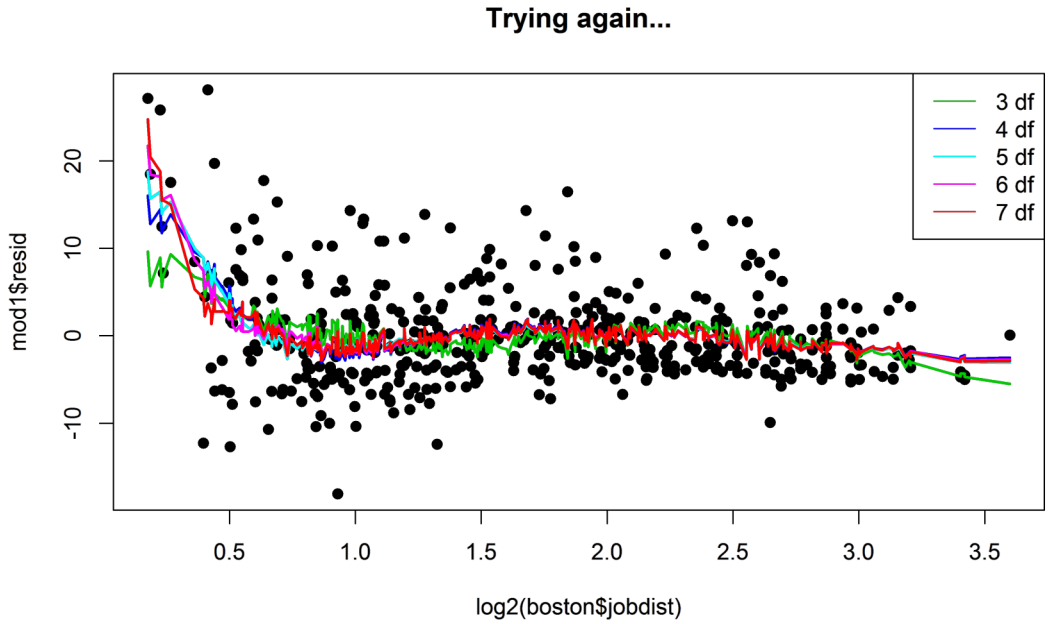
Instead, we can specify the knots, adding one at a time at the region we suspect to be most sensitive.

```
mod44 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
    c(1/6, 1/3, 2/3))), data = boston)
mod55 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
    c(1/12, 1/6, 1/3, 2/3))), data = boston)
mod66 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
    c(1/24, 1/12, 1/6, 1/3, 2/3))), data = boston)
mod77 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
    c(1/48, 1/24, 1/12, 1/6, 1/3, 2/3))), data = boston)
mod88 = lm(homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
    c(1/96, 1/48, 1/24, 1/12, 1/6, 1/3, 2/3))), data = boston)
lrtest(mod3, mod44, mod55, mod66, mod77, mod88)
```

```
## Likelihood ratio test
##
## Model 1: homeval ~ lowSES + rooms + ns(log2(jobdist), df = 3)
## Model 2: homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
##     c(1/6, 1/3, 2/3)))
## Model 3: homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
##     c(1/12, 1/6, 1/3, 2/3)))
## Model 4: homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
##     c(1/24, 1/12, 1/6, 1/3, 2/3)))
## Model 5: homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
##     c(1/48, 1/24, 1/12, 1/6, 1/3, 2/3)))
## Model 6: homeval ~ lowSES + rooms + ns(log2(jobdist), knots = quantile(log2(jobdist),
##     c(1/96, 1/48, 1/24, 1/12, 1/6, 1/3, 2/3)))
##   #Df LogLik Df Chisq Pr(>Chisq)
## 1   7  -1555
## 2   8  -1538  1 34.05    5.4e-09 ***
## 3   9  -1536  1  4.21      0.040 *
## 4  10  -1533  1  5.69      0.017 *
## 5  11  -1530  1  5.42      0.020 *
## 6  12  -1530  1  0.07      0.792
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Splines in Boston... More Carefully Now

```
plot(log2(boston$jobdist), mod1$resid, pch = 19, main = "Trying again...")
lines(log2(boston$jobdist[jobperm]), mod3$fitted[jobperm] - mod1$fitted[jobperm],
    col = 3, lwd = 2)
lines(log2(boston$jobdist[jobperm]), mod44$fitted[jobperm] - mod1$fitted[jobperm],
    col = 4, lwd = 2)
lines(log2(boston$jobdist[jobperm]), mod55$fitted[jobperm] - mod1$fitted[jobperm],
    col = 5, lwd = 2)
lines(log2(boston$jobdist[jobperm]), mod66$fitted[jobperm] - mod1$fitted[jobperm],
    col = 6, lwd = 2)
lines(log2(boston$jobdist[jobperm]), mod77$fitted[jobperm] - mod1$fitted[jobperm],
    col = 2, lwd = 2)
legend("topright", legend = paste(3:7, "df"), lty = 1, col = c(3:6, 2))
```

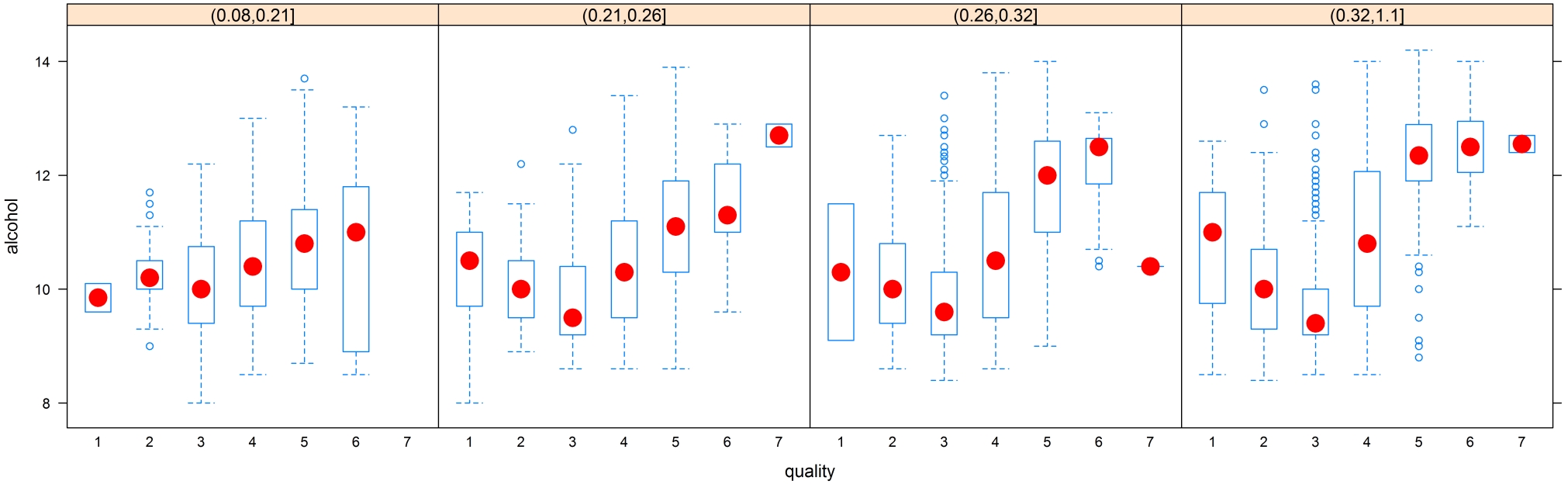


It's unlikely that 7 d.f. for this variable will withstand the full model - but at least we are spending the d.f. more carefully. *Questions? (online/in-class)* Try to see whether *linear* `bs` splines (`degree=1`) can match this performance on the same variable.

R Annoyance/Workaround: Taking Control of `lattice`'s `bwplot`

`lattice` is great for visualizing multi-way relationships. But as some of you have noticed, to make really nice plots usually requires a lot of tedious lists. `bwplot` is perhaps the worst in this. For example, what do the usual modifiers `col, cex, pch` control?

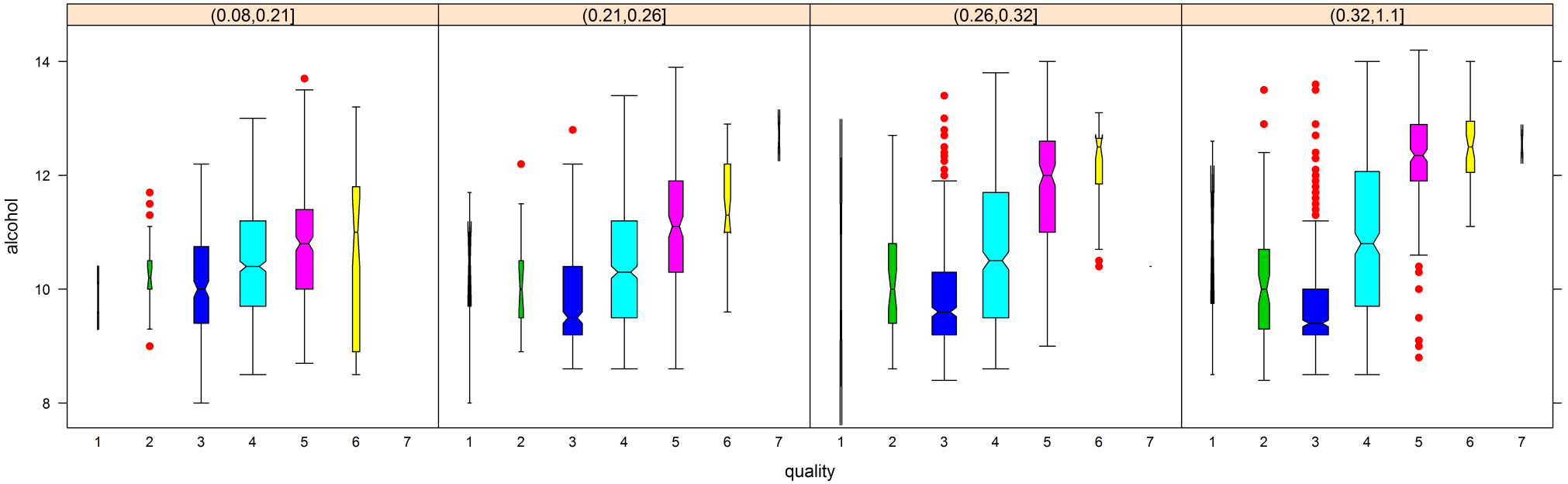
```
wine = read.csv("../Datasets/winequality-white.csv", as.is = T)
bwplot(alcohol ~ quality | cut(volatile.acidity, quantile(volatile.acidity)),
    data = wine, col = 2, pch = 19, cex = 2, horiz = F)
```



It controls the median symbol. Even more irritating, if you forget `horiz=F` the boxplot will go the wrong way!

R Annoyance/Workaround: Taking Control of `lattice`'s `bwplot`

```
bwplot(alcohol ~ quality | cut(volatile.acidity, quantile(volatile.acidity)),
    data = wine, pch = "|", varwidth = T, par.settings = list(box.umbrella = list(col = 1,
        lty = 1, coef = 3.5), box.rectangle = list(col = 1, fill = c(1, 3:8)),
        plot.symbol = list(pch = 19, col = 2)), horiz = F, notch = T)
```

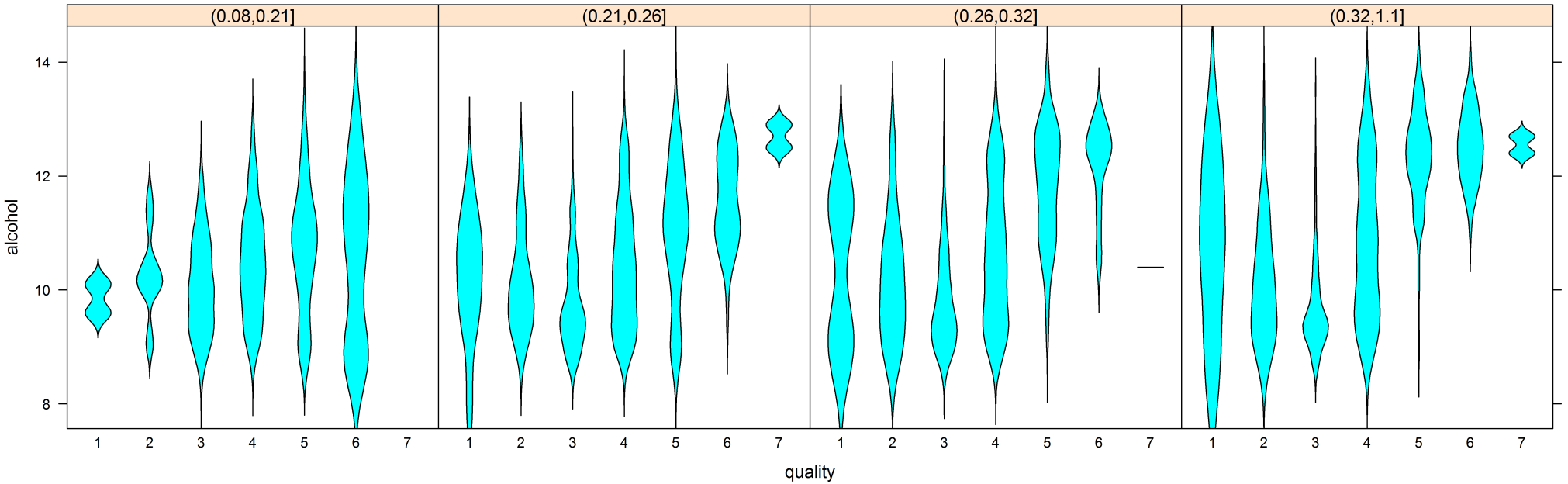


The `varwidth` option exists in the regular `boxplot` function as well. It makes the box width proportional to $\sqrt{n_g}$, with n_g being the group size. The `notch` option marks approximate 95% CI's for the median, helping identify whether group differences might be significant.

Done? Not quite…

R Annoyance/Workaround: Taking Control of `lattice`'s `bwplot`

```
bwplot(alcohol ~ quality | cut(volatile.acidity, quantile(volatile.acidity)),
    data = wine, panel = panel.violin, horiz = F)
```



This is a **violin plot**, which replaces the boxes, whiskers and symbols by two mirror-images of a density plot.

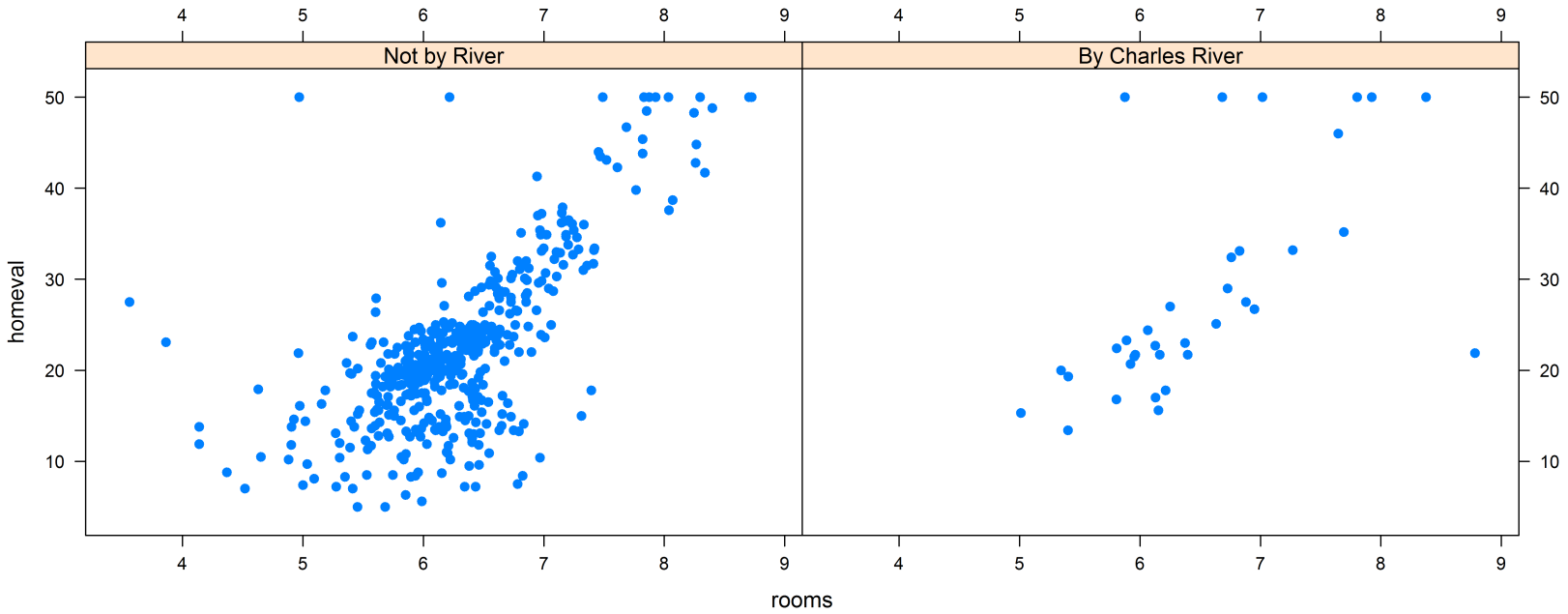
Modifying it (color, etc.) is even harder than for plain-vanilla `bwplot` (some examples can be found online). And `varwidth` seems to operate in the wrong direction.

But some people like it.

Cool R Trick, *"Things I Learned from my Students"* Corner

Remembered the awkward, 3-command way to get better labels for `lattice` panel headings?

```
xyplot(homeval ~ rooms | factor(river, labels = c("Not by River", "By Charles River")),
    pch = 19, data = boston, scales = list(alternating = 3))
```



Courtesy of Alan Bruce.

`label` is just an attribute of `factor`, not part of `lattice`. You can also specify it as an argument of a `cut` command (worked for me). Try it out!