# THE GROUP LASSO FOR QUANTILE REGRESSION: AN R VINAIGRETTE

ROGER KOENKER

Abstract. Shrinkage is powerful, but a well-chosen basis expansion is sublime.

## 1. Introduction

In Koenker (2011) I have tried to describe some ideas for estimation and inference about additive quantile regression models based on total variation penalization. This approach has several advantages, not the least of which is the efficient linear programming formulation for the computation. But a sticking point has always been choice of smoothing parameters. An alternative approach to additive models has been proposed in Kato (2011) based on a group lasso formulation in which the Euclidean norm of the coefficients of the additive components are penalized. An innovative aspect of this approach is the choice of penalty parameters which are selected by simulating the gradient condition of the QR model, yielding a automatic, data-dependent smoothing strategy. There is a close connection to prior work of Belloni and Chernozhukov (2011) for the QR lasso problem, but the explicit link to additive models and regression smoothing of the Kato approach made it particularly appealing for me. In this note I will briefly describe some computational experience with this approach; as usual there are some positive and negative aspects.

## 2. The Model

We will consider a simple QR model of the form,

$$Q_{Y|z}(\tau|z) = g(z),$$

where $\tau \in (0,1)$ will be fixed, and the target function $g : \mathbb{R} \to \mathbb{R}$ will be assumed to be approximable, in some vague sense, by a B-spline expansion,

$$g(z) = \beta_0 + \sum_{j=1}^{p} \psi_j(z)\beta_j.$$

We will write the $n$ by $p$ design matrix, $X = (\psi_j(z_i))$, with rows, $x_i$, so we have a simple garden variety linear quantile regression model. Rather than assuming that we are clever enough to chose the basis expansion parsimoniously, we will rely instead on penalization to control the variability of the estimated parameters $\hat{\beta}$. It is presumably obvious that this setting can be expanded to incorporate more complicated additive models, but we will restrict attention to this simple one component model.

Version February 17, 2020. A manifesto for the Vinaigrette genre is available at `https://davoidofmeaning.blogspot.com/2016/12/r-vinaigrettes.html`. Comments on the genre-al idea, or on this particular instance would be most welcome.

## 3. Computation

There are potentially many schemes for penalization, but the approach we will consider here is the Euclidean norm penalty of Kato, $P(\beta) = \|S\beta\|_2$, where $S = (n^{-1}X^\top X)^{1/2}$ is intended to normalize the coordinates of $\beta$. The intercept parameter $\beta_0$ is left unpenalized. The resulting estimator solves,

$$\min \sum_{i=1}^{n} \rho_\tau(Y_i - \beta_0 + x_i^\top \beta) + \lambda P(\beta),$$

which can be reformulated as the second order cone programming problem,

$$\max\{y^\top a | 1^\top a = 0,\ \|b\| \leqslant \lambda,\ Sb = X^\top a,\ a \in [\tau - 1, \tau]^n\}.$$

This dual formulation can be efficiently solved with modern interior point methods. In Appendix A we illustrate an implementation for R and the optimization suite Mosek, Andersen (2010).

But how to we select $\lambda$? Kato, extending the proposal of Belloni and Chernozhukov (2011) for the QR lasso, suggested that $\lambda$ should be chosen so that the probability of the event $\{\lambda \geqslant c\Lambda\}$ is close to one, where $c$ is slightly larger than 1, and $\Lambda$ can be approximated by an upper quantile of the random variable,

$$\tilde{\Lambda} = \| \sum_{i=1}^{n} (\tau - I(U_i \leqslant \tau))(S^{-1/2} x_i / \sqrt{p}\|_2,$$

where $U_i$ are iid uniform $[0, 1]$. This quantile can be easily simulated and we will explore several choices of the quantile in the examples in the next section. In accordance with the suggestions in Kato (2011) we have chosen $c = 1.1$ for the implementation appearing in Appendix A. The implementation is structured so that one can specify a choice of $\lambda$ directly, or specify a quantile for the automated $\lambda$ selection. We will employ both options below.

## 4. Computational Experience

Our test problem is taken from Polson and Scott (2016): $z$ observations are taken as equally spaced on $[0, 1]$ and we contrast B-spline expansions with 10 and 50 basis functions. The target functions are given by,

$$Q_{Y|x}(\tau|x) = 5 * \sin(2 * \pi * x) + (0.5 + \exp(1.5 * \sin(4 * \pi * x)))\Phi^{-1}(\tau)$$

The target functions are all quite smooth, but the model is quite heteroscedastic so the conditional quantile functions are quite distinctive.

We begin by illustrating performance of the estimator when the basis expansion is relatively parsimonious so there are only 10 basis function. Figure 1 a sample of 500 observations from this model. We have super-imposed the true conditional quantile curves in grey for $\tau \in \{.02, .15, .5, .85, .98\}$, and estimated conditional quantile curves in black with $\lambda$ chosen by the automated procedure described above with the default (Kato) choice of $\theta = 0.1$ yielding the 0.9 quantile of the simulated distribution with 500 realizations of the Bernoulli vector. One can always be suspicious about simulations of size one so diligent readers are encouraged to repeat this exercise with altered seeds, but I believe that it is fair to say that this performance is encouraging.

Encouraged, we now repeat the exercise with 50 basis functions rather than only 10. Same setup, same seed, but now the fit in Figure 2 is horribly undersmoothed. Is this the fault of our

automated $\lambda$ selection? Let's see, focusing on $\tau = 0.85$ we can try to increase $\lambda$ "gradually" and see what happens. In Figure 3 we have turned off the automated $\lambda$ selection and tried exploring with the well-known, but much maligned "eyeball method." Our automated selection was

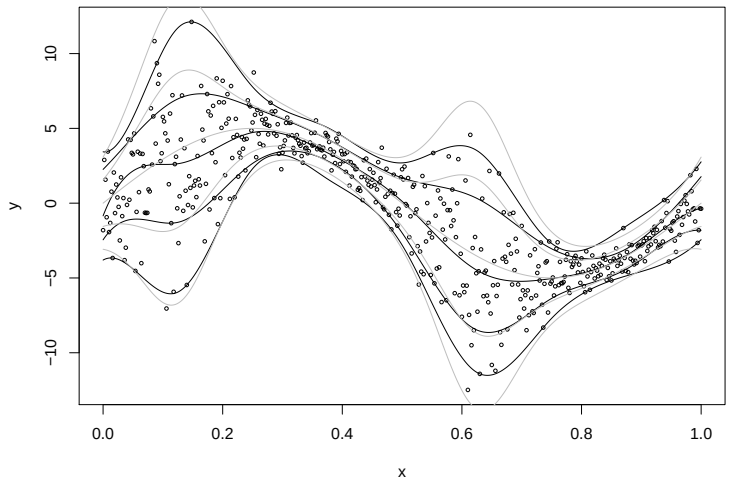


FIGURE 1. Sample of 500 points from the Polson-Scott model, with true conditional quantile curves for $\tau \in \{.02, .15, .5, .85, .98\}$, in grey, and fitted curves in black based on automated $\lambda$ selection with 10 B-spline basis functions.

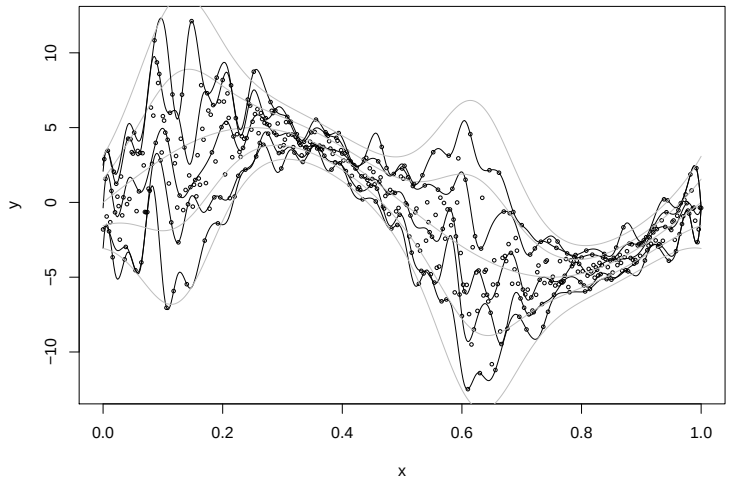


FIGURE 2. Sample of 500 points from the Polson-Scott model, with true conditional quantile curves for $\tau \in \{.02, .15, .5, .85, .98\}$, in grey, and fitted curves in black based on automated $\lambda$ selection with 50 B-spline basis functions.

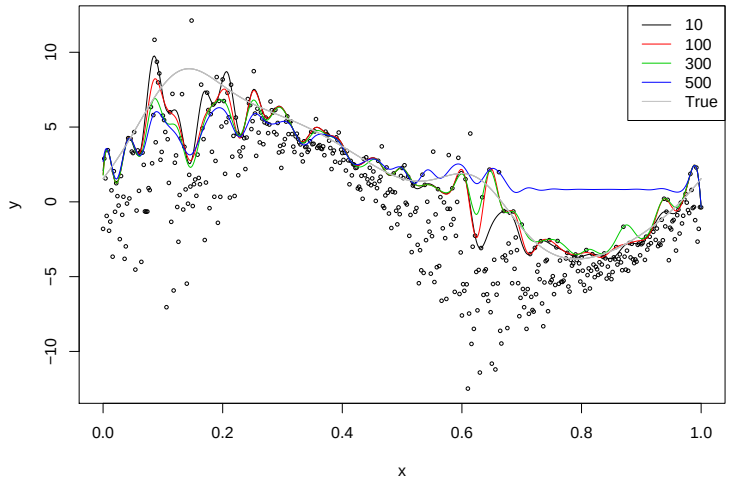


FIGURE 3. Sample of 500 points from the Polson-Scott model, with true conditional quantile curve for $\tau = .85$, in grey, and fitted curves in various colors for several $\lambda$'s with 50 B-spline basis functions.

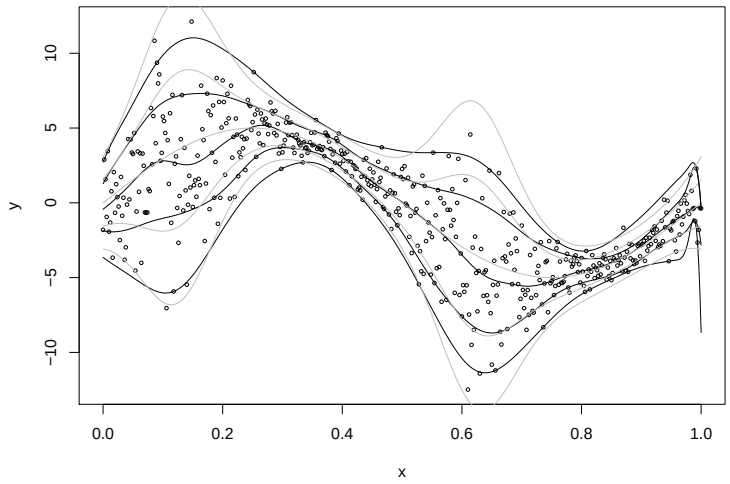


FIGURE 4. Sample of 500 points from the Polson-Scott model, with true conditional quantile curves for $\tau \in \{.02, .15, .5, .85, .98\}$, in grey, and fitted curves in black based on semi-automated $\lambda$ selection with 50 B-spline basis functions and second-differenced Euclidean norm penalty.

$\hat{\lambda} = 9.97$, so we started by turning up the knob to $\lambda = 100$. Surprisingly, this made almost no visible difference, still the estimate was drastically undersmoothed. So what about $\lambda = 300$? Same story. At $\lambda = 500$ we do get something quite different, but even worse. Instead of producing a nice smooth curve throughout the relevant domain, we see the same wiggles for small $x$, and then there is a range of $x$ from about 6 to 9.5 in which the estimate flattens out and as a result completely misses the dip in the target function. So apparently there is no $\lambda$ that can successfully capture the slowly undulating structure of our test problem when the B-spline basis expansion is so expansive.

What went wrong? When $p$ was 10 everything looked splendid. If we think back to our first lesson in B-splines it will be recalled that their main virtue is that they are locally supported, so with $p = 50$ have lots of basis functions all with small support, and this seems to inevitably lead to a highly oscilliatory fit. When there are fewer basis functions we can expect to get something much smoother, and we do. Unfortunately, the penalization applied here isn't clever enough to find a good linear combination of 50 basis functions that would produce reasonable fit. What we would need is a penalty that understood that adjacent basis functions should have coefficients that didn't differ too much. This is precisely the feature of the familiar $L_2$ smoothing spline penalty, which boils down to shrinking *second differences* in adjacent spline coefficients. This can be adapted quite easily to the QR setting as already suggested in much earlier work of Bosch, Ye, and Woodworth (1995) and more recently by Koenker and Mizera (2014) and Fasiolo, Goude, Nedellec, and Wood (2017).

Indeed, it is relatively easy to adapt the Euclidean penalty described above so that it penalized second differences simply by modifying the matrix $S$. In Appendix A we also include a modified version of the original code that replaces the scaled Euclidean penalty of Kato (2011) with a version that penalizes the Euclidean norm of the second differences of the B-spline coefficients. Unfortunately, automatic $\lambda$ selection still produces a severely undersmoothed fit, but when $\lambda$ is dialed up to produce a smoother fit we do get the expectedly smoother fit. How to automatically select $\lambda$ in this formulation is still an open question however.

## 5. Conclusion

Admittedly, the foregoing examples don't really *prove* anything, but they offer *prima facie* evidence that the Euclidean penalty. unless paired with a carefully selected basis expansion, is difficult to tune automatically. In contrast, the total variation smoothing penalty, because it is tied more closely to the data points via the linear programming structure, seems better capable of accomplishing this.

## Appendix A. R Coda

```
# Mosek implementation of (dual) group lasso method for QR a la Kato (2011)
glasso = function(X, y, G, lambda = NULL, tau = 0.5, theta = NULL, rtol = 1e-6, verb = 0, ...){
    # G is a list of group subsets of the columns of X.
    # By convention G[[1]] contains un-penalized columns
    # No sanity check for G (yet!)
    q = length(G)
    p = NCOL(X)
    n = NROW(X)
    Pk = 0
    lams = rep(NA, q)
    P = list(sense = "max")
```

```
    P$c = c(y, rep(0, p+2))
    P$bc = matrix(0,2,p)
    P$cones = matrix(list(),2,q)
    P$cones[1,1:q] = "QUAD"
    P$A = Matrix(0, p, p)
    for(k in 1:q){
        Xk = X[,G[[k]], drop = FALSE]
        pk = NCOL(Xk)
        Sk = chol(crossprod(Xk)/n)/sqrt(pk)
        lams[k] = lamhat(Xk, tau = tau, theta = theta)
        P$A[(Pk + 1):(Pk + pk),(Pk + 1):(Pk + pk)] = Sk
        P$cones[2,k] = list(c(n + p + k, (n + Pk + 1):(n + Pk + pk)))
        Pk = Pk + pk
    }
    if(!length(lambda)) lambda = max(lams[-1])
    P$bx = rbind(c(rep(tau-1,n),rep(-Inf,p),0,lambda),c(rep(tau,n),rep(Inf,p), 0, lambda))
    P$A = cbind(t(X), -P$A,0,0)
    P$dparam$intpnt_co_tol_rel_gap <- rtol
    z <- mosek(P, opts = list(verbose = verb))
    status <- z$sol$itr$solsta
    coef = z$sol$itr$slc - z$sol$itr$suc
    resid = y - X %*% coef
    list(coef = coef, resid = resid, lambda = lambda, status = status)
}
lamhat = function(X, tau, R = 500, c = 1.1, theta = 0.1){
    n = NROW(X)
    p = NCOL(X)
    S = backsolve(chol(crossprod(X)/n), diag(p)) %*% t(X)/sqrt(p)
    B = tau - (matrix(runif(n * R), n, R) < tau)
    v = apply((S %*% B)^2, 2, sum)
    c * quantile(sqrt(v), 1 - theta)
}
# fig code for the glasso vinaigrette
require(Rmosek)
require(splines)
source("glasso.R")
set.seed(1848)
# Polson-Scott Model
dgp <- function(n) {
     x <- 0:n/n
     y <- rnorm(n+1, 5 * sin(2 * pi * x), 0.5 + exp(1.5 * sin(4 * pi * x)))
     data.frame(x = x, y = y)
}
M = dgp(500)
x = M$x
y = M$y
X = cbind(1,bs(x, df = 10))
taus = c(.02, .15, .5, .85, .98)
pdf(file = "fig1.pdf", height = 6, width = 8)
plot(x,y,cex = .5)
for(i in 1:length(taus)){
    F = glasso(X,y, G = list(1,2:NCOL(X)), tau = taus[i], theta = .1)
    yF = X %*% F$coef
    lines(x, yF)
    lines(x, 5 * sin(2 * pi * x) +  (0.5 + exp(1.5 * sin(4 * pi * x))) * qnorm(taus[i]), col = "grey")
```

```
}
dev.off()
X = cbind(1,bs(x, df = 50))
taus = c(.02, .15, .5, .85, .98)
pdf(file = "fig2.pdf", height = 6, width = 8)
plot(x,y,cex = .5)
for(i in 1:length(taus)){
    F = glasso(X,y, G = list(1,2:NCOL(X)), tau = taus[i], theta = .1)
    yF = X %*% F$coef
    lines(x, yF)
    lines(x, 5 * sin(2 * pi * x) +  (0.5 + exp(1.5 * sin(4 * pi * x))) * qnorm(taus[i]), col = "grey")
}
dev.off()
X = cbind(1,bs(x, df = 50))
lambdas = c(10, 100, 300, 500)
pdf(file = "fig3.pdf", height = 6, width = 8)
plot(x,y,cex = .5)
for(i in 1:length(lambdas)){
    F = glasso(X,y, G = list(1,2:NCOL(X)), tau = 0.85, lambda = lambdas[i], theta = .1)
    lambdas[i] = F$lambda
    yF = X %*% F$coef
    lines(x, yF, col = i)
    lines(x, 5 * sin(2 * pi * x) +  (0.5 + exp(1.5 * sin(4 * pi * x))) * qnorm(0.85), col = "grey")
}
legend("topright", c(expression(lambda = 10), expression(lambda = 100), expression(lambda = 300),
     expression(lambda = 500), expression(True)), col = c(1:4,"grey"), lty = 1)
dev.off()



# NB.  This version replaces Kato's standardization with smoothing penalty.
glasso = function(X, y, G, lambda = 1, tau = 0.5, rtol = 1e-6, verb = 0, ...){
# G is a list of group subsets of the columns of X.
# By convention G[[1]] contains un-penalized columns
# No sanity check for G (yet!)
q = length(G)
p = NCOL(X)
n = NROW(X)
Pk = 0
P = list(sense = "max")
P$c = c(y, rep(0, p+2))
P$bc = matrix(0,2,p)
P$cones = matrix(list(),2,q)
P$cones[1,1:q] = "QUAD"
P$A = Matrix(0, p, p)
for(k in 1:q){
    Xk = X[,G[[k]], drop = FALSE]
    pk = NCOL(Xk)
    if(k == 1){
Sk = diag(pk)
P$A[(Pk + 1):(Pk + pk),(Pk + 1):(Pk + pk)] = Sk
    }
    else {
Sk = diff(diff(diag(pk)))
P$A[(Pk + 1):(Pk + pk - 2),(Pk + 1):(Pk + pk)] = Sk
    }
```

```
    P$cones[2,k] = list(c(n + p + k, (n + Pk + 1):(n + Pk + pk)))
    Pk = Pk + pk
}
P$bx = rbind(c(rep(tau-1,n),rep(-Inf,p),0,lambda),c(rep(tau,n),rep(Inf,p), 0, lambda))
P$A = cbind(t(X), -P$A,0,0)
P$dparam$intpnt_co_tol_rel_gap <- rtol
z <- mosek(P, opts = list(verbose = verb))
status <- z$sol$itr$solsta
coef = z$sol$itr$slc - z$sol$itr$suc
resid = y - X %*% coef
list(coef = coef, resid = resid, status = status)
}

# New fig2 with smoother penalty
require(Rmosek)
require(splines)
source("glassos.R")
set.seed(1848)
# Polson-Scott Model
dgp <- function(n) {
    x <- 0:n/n
    y <- rnorm(n+1, 5 * sin(2 * pi * x), 0.5 + exp(1.5 * sin(4 * pi * x)))
    data.frame(x = x, y = y)
}
M = dgp(500)
x = M$x
y = M$y
X = cbind(1,bs(x, df = 50))
taus = c(.02, .15, .5, .85, .98)
pdf(file = "fig4.pdf", height = 6, width = 8)
plot(x,y,cex = .5)
for(i in 1:length(taus)){
    F = glasso(X,y, G = list(1,2:NCOL(X)), tau = taus[i], lambda = 6, verb = 5)
    yF = X %*% F$coef
    lines(x, yF)
    lines(x, 5 * sin(2 * pi * x) + (0.5 + exp(1.5 * sin(4 * pi * x))) * qnorm(taus[i]), col = "grey")
}
dev.off()
```

## References


Andersen, E. D. (2010): "The Mosek Optimization Tools Manual, Version 6.0," Available from https://www.mosek.com.

Belloni, A., and V. Chernozhukov (2011): "$\ell_1$-penalized quantile regression in high-dimensional sparse models," *The Annals of Statistics*, 39, 82–130.

Bosch, R. J., Y. Ye, and G. G. Woodworth (1995): "A Convergent Algorithm for Quantile Regression With Smoothing Splines," *Computational Statistics and Data Analysis*, 19, 613–630.

Fasiolo, M., Y. Goude, R. Nedellec, and S. N. Wood (2017): "Fast calibrated additive quantile regression," Available from https://arxiv.org/abs/1707.03307.

Kato, K. (2011): "Group Lasso for high dimensional sparse quantile regression models," Available from https://arxiv.org/abs/1103.1458.

Koenker, R. (2011): "Additive models for quantile regression: Model selection and confidence bandaids," *Brazilian Journal of Probability and Statistics*, 25, 239–262.

Koenker, R., and I. Mizera (2014): "Convex Optimization in R," *Journal of Statistical Software, Articles*, 60, 1–23.

Polson, N. G., and J. Scott (2016): "Mixtures, envelopes and hierarchical duality," *J. of the Royal Statistical Society (B)*, 78, 701–727.