QUANTILE REGRESSION METHODS: AN R VINAIGRETTE

ROGER KOENKER

1. Introduction

In the beginning data was scarce and life was hard. Imagine estimating quantile regression models in 1975 for the infamous "stackloss" data, 21 observations with 4 covariates, on a CDC 7600 the size of a small New York kitchen with SAS linear programming algorithms and punch cards. Gradually, data became more plentiful and interacting with computers became easier, in the process methods of estimation and inference for quantile regression became more sophisticated. This note is intended to provide some guidance for selecting among the various methods offered by the author's R package **quantreg** depending on problem size and other characteristics. This is a prelude, I hope, to some constructive new development for the package. In the immortal words of Marx's last thesis on Feuerbach: "Philosphers have only tried to understand the world, our task is to change it." But it usually helps to understand it first.

Figure 1 offers an overview of the scope of the problem. Algorithms for point estimation of quantile regression parameters appear in blue, summary methods for assessing the precision of these estimates appear in pink, among these there are several distinct bootstrap methods appearing in brown, and for each bootstrap method there are one or more bootstrap algorithms available. Quantile regression models can be roughly characterized by four features: the sample size, n; the parametric dimension of the model, p; the degree of sparsity of the design matrix; and the number of quantiles to be estimated. Our objective is to offer some advice on how to select methods depending upon these features.

2. Algorithms for Point Estimation

Edgeworth described an algorithm for median regression in 1888, but it wasn't until the early 1950's that general median regression algorithms emerged with the development of linear programming. The Barrodale and Roberts (1974) dual bounded variables algorithm for median regression was ideally suited to the elaboration to quantile regression models that were linear in parameters, as described in Koenker and d'Orey (1987). This simplex approach was implemented in S in the late 1970's and remains an efficient method for problems of modest size. Interior point methods for linear programming that offered advantages for larger problems arrived in the 1990's and have been gradually incorporated into the **quantreg** package, first for unconstrained problems, then for linear inequality constrained problems and finally for problems with sparse designs. There are currently ten distinct algorithms for estimating parametric, linear quantile regression models:

January 13, 2021.A genre manifesto for R Vinaigrettes is available at http://davoidofmeaning.blogspot.com/2016/12/r-vinaigrettes.html.

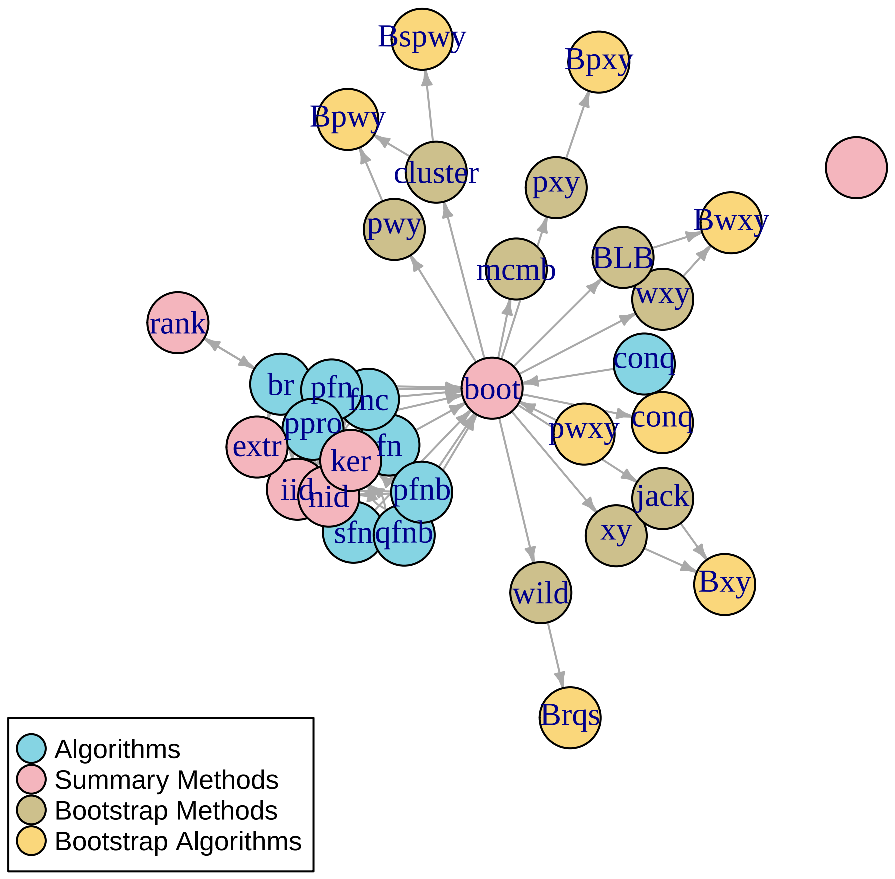

Figure 1. Network of Quantile Regression Methods

"br" The original simplex algorithm based on Barrodale and Roberts (1974). An unfortunate aspect of this algorithm is that the underlying fortran is highly unstructured and difficult to follow. At some point in 1998 I realized that the basic idea of the algorithm was quite close the essential idea of Edgeworth's 1888 paper, and I wrote a version in R that is only about 25 lines. This prototype version is available at `http://www.econ.uiuc.edu/~roger/research/rq/rqx.R`.

"fn" A basic interior point algorithm based on ideas of Mehrotra (1992) that I refer to as the Frisch-Newton method since it can be seen as a Newton-type method with log-barrier constraints as originally suggested by Ragnar Frisch.

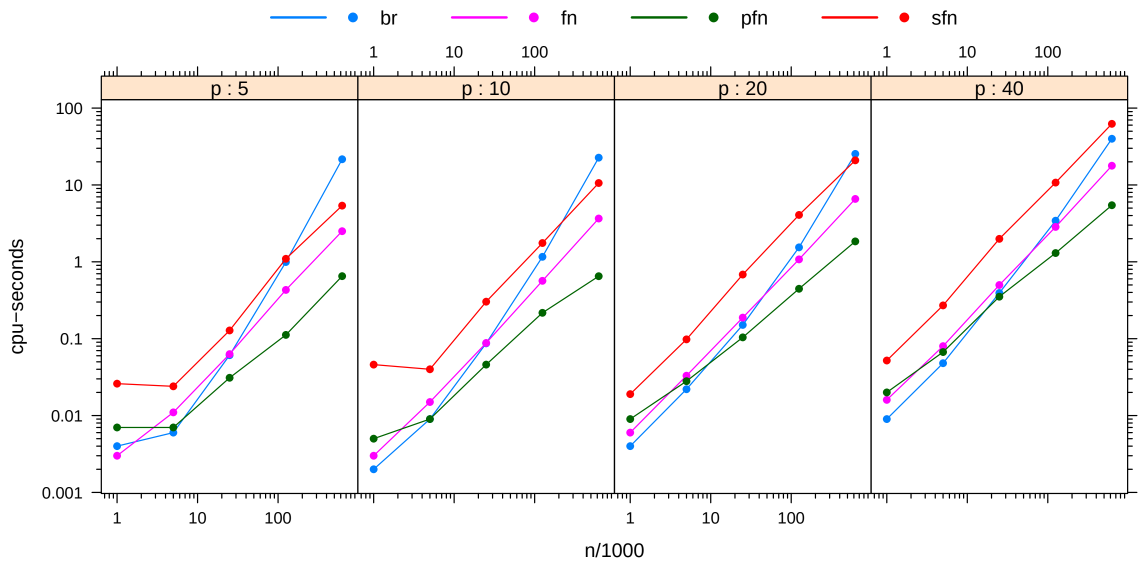

FIGURE 2. Comparison of Algorithm CPU Time for Dense Problems

"fnc"	A variant of the "fn" interior point algorithm that introduces linear inequality constraints.
"sfn"	A variant of the "fn" method that exploits sparse algebra for problems in which the design matrix is sparse.
"sfnc"	A variant of the "fnc" method that exploits sparse algebra for problems in which the design matrix is sparse.
"pfn"	A variant of the "fn" method that exploits a preprocessing strategy described in Portnoy and Koenker (1997) to reduce the effective sample size of the problem.
"ppro"	A variant of the "pfn" method that exploits preprocessing when multiple quantiles are being estimated.
"qfnb"	A variant of the "fnb" method designed for multiple quantiles all estimated within a fortran loop.
"pfnb"	A variant of the "ppro" method but implemented in fortran.
"lasso"	A variant of the "fn" method that incorporates a lasso, or scad penalty.

Further details about these methods is available from the package documentation for the function `rq` and the references provided therein.

Figure 2 compares cpu time in seconds for four of these algorithms: "br", "fn", "sfn" and "pfn." The model has p standard Gaussian covariates and Gaussian response. When n is less than 25,000 the simplex algorithm of Barrodale and Roberts is quite competitive, but for larger sample sizes the Frisch-Newton algorithm is quicker. Preprocessing is clearly advantageous beyond $n = 100,000$, or so, but its advantage dissipates somewhat for larger parametric dimension, p, of the model. For dense design matrices like these sparse algebra is not helpful The four methods all achieve the same precision to seven decimal digits.

Figure 3 compares performance in a similar setting except that a factor variable with q distinct levels has been added to 5 dense covariates of the model. Now Frisch-Newton is

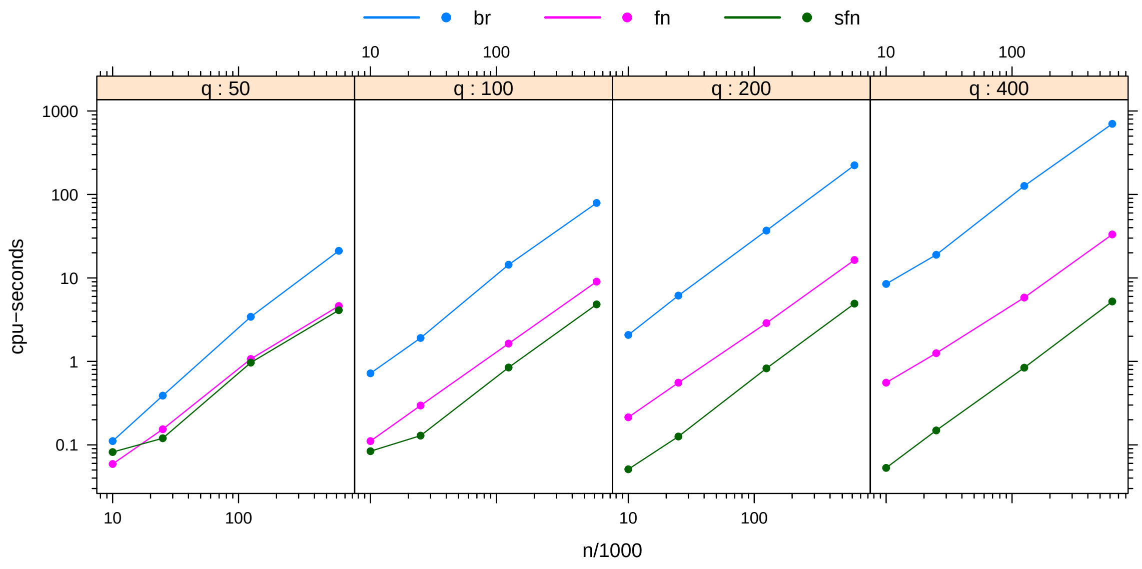

FIGURE 3. Comparison of Algorithm CPU Time for Sparse Problems

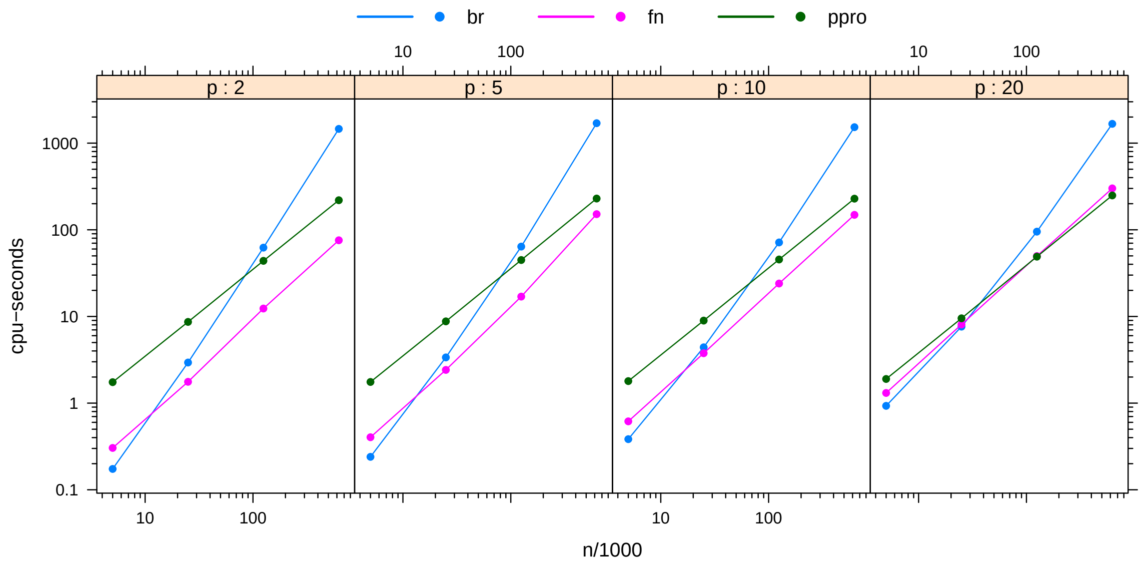

FIGURE 4. Comparison of Algorithm CPU Time for Multiple taus

clearly superior to simplex, but the sparse algebra version of Frisch-Newton is much faster especially when q is large.

2.1. **Estimation with Many τ's.** Thus far we have only considered estimating a single conditional quantile model, but many applications involve estimating several quantiles as an ensemble. This is relatively easy to do by specifying, for example for the deciles, `tau = 1:9/10` in the call to `rq`. A little reflection will suggest that adjacent quantiles should have somewhat similar solutions, so some form of preprocessing might be helpful rather solving for each value of τ, *de novo*. This idea is closely related to the preprocessing proposed in Portnoy and Koenker (1997); given a solution at some τ_0 if we want a solution at some nearby τ_1 we can presume that the signs of the residuals for τ_0 should be predictive of the sign of the residuals for τ_1. This being the case we can combine observations with large negative residuals into one new pseudo-observation with a very negative response, and likewise combine observations with large positive residuals into another pseudo-observation with a very positive response and thereby effectively reduce the effective sample size of the estimation problem. In Portnoy and Koenker (1997) this process was based on a preliminary fit for a model with a fixed τ, but the idea for which we coined the highly sophisticated technical term "globbing" is very similar. Globbing for adjacent quantiles was implemented by Blaise Melly and described in further detail in Chernozhukov et al. (2020), and is now implemented in **quantreg** with the `method = "ppro"` option for `rq` function.

Figure 4 compares CPU effort for "br", "fn" and "ppro" for our dense model (without the factor variable) with $\tau \in \{0.02, 0.04, \ldots, 0.98\}$, for 49 distinct, equally spaced τ's. The results are quite surprising, so much so that I suspected that I'd blundered somewhere. There is no performance gain from preprocessing, indeed except for the largest of the models fitted, with $n = 625,000$ and $p = 20$, "fn" is always quicker than "ppro". Profiling the preprocessing code I discovered that most of cpu time was being spent computing sums rather than actually doing the "real work" of optimization. This finding led to a extended detour into the wonderful world of ratfor and fortran.[1]

Figure 5 compares performance of the "ppro" R implementation with two new fortran implementations for the same problems considered in Figure 4. The "qfnb" method was an intermediate prototype to see whether I could still remember how to write ratfor; it simply loops over a vector of τ's solving the full n by p problem at each step. The "pfnb" method implements the preprocessing approach of "ppro" but all of the computation stays inside the fortran call. The preprocessing strategy when "fortranized" shows a clear performance advantage. There are several possible refinements, or tuning parameters to choose for the preprocessing; the most crucial of these is the choice of the initial sample size that I have provisionally set to be $n^{2/3}p^{1/2}$. Further experimentation could well lead to even better performance.

There are hints scattered around the R ecosystem, including the authoritative R Core Team (2020) suggesting that the standard R interface with Fortran, i.e. `.Fortran`, is inefficient compared to the more recently introduced `.Call` interface, although I couldn't find any quantitative evidence for the magnitude of this "overhead" penalty. This prompted me to write an inquiry to the newsgroup R-devel that provoked several responses including the suggestion that the package **dotCall64** might offer a relatively easy way to explore the magnitude of such effects. So in the spirit of scientific exploration I wrote a new version of `rq.fit.pfnb` that simply replaced the call to `.Fortran` with a call to `.C64` in this package

[1]Ratfor is a dialect of fortran (rational fortran) developed at Bell Labs in the late 1970's providing better control structures and more flexible formatting. Like many ancient languages it is somewhat endangered, but after some extensive googology I found an updated C version maintained by Brian Gaeke that generated fully functional fortran 77. This had the valuable side benefit that it allowed me to clean up some incompatibilities in my earlier ratfor code as well as allowing me to write new code. I've added links on my Reproducibility webpage to a tar image of the source files for Gaeke's version of ratfor and to Brian Kernighan's brilliant 10 page exposition of the ratfor language.

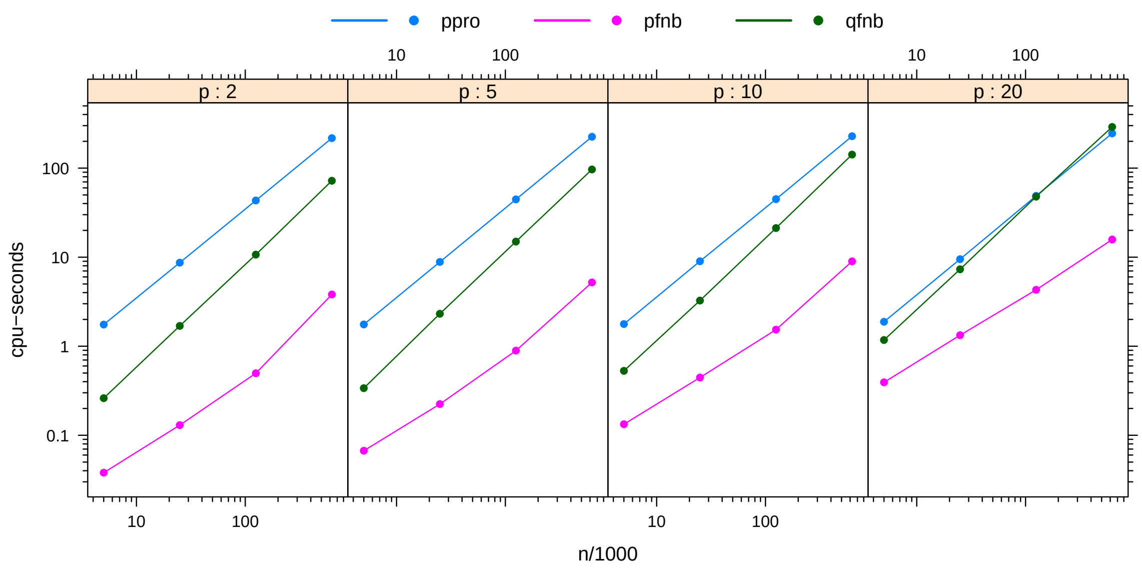

FIGURE 5. Comparison of New Fortran Algorithms for Multiple taus

and then running a moderately large problem with $n = 500,000$ and $p = 5$ for 49 equally spaced quantiles.

In this example there is a slight performance advantage for the new `.C64` code, which requires 4.37, versus 4.505 seconds for the `.Fortran` version; for somewhat smaller problems the `.Fortran` was actually quicker. My provisional conclusion from this exercise was that this performance gain didn't justify the additional layer of coding.[2]

3. Summary Methods

In R parlance a "summary" method can be anything that elaborates or coagulates the salient features of an R object. For fitted objects from linear models this typically means that they provide some form of inference about the reliability of the parameters of the fitted model. Like quantile regression fitting methods there are quite a few summary method options. The earliest of these offered direct estimates of the asymptotic covariance (sandwich) matrix, which takes the form,

$$V = \tau(1-\tau)H_n^{-1}J_nH_n^{-1},$$

where $J_n = n^{-1}\sum_i x_i x_i^\top$, $H_n = n^{-1}\sum_i x_i x_i^\top f_{Y|x}(\xi_i(\tau)$ and $f_i(\xi_i(\tau)) = f_i(x_i^\top \beta(\tau))$ is the conditional density of the response given covariates.

These original methods are:

- iid estimates a scalar conditional density value from fitted residuals assuming that H_n and J_n are the same up to a factor of proportionality,
- nid estimates a vector of conditional density values for H_n by differencing fitted models at $\tau \pm h$, for some bandwidth h,
- ker estimates H_n by the kernel method of Powell (1991).

[2]Kaspar Daniel Hansen suggested that perhaps it would also be worthwhile to compare the memory requirements of the two methods and indeed this does reveal a significant gain: 146Mb versus 262Mb for the .Fortran version.

Two other methods emerged slightly later:

rank estimates a confidence interval for each parameter by inverting a rank test derived from the general theory of Gutenbrunner and Jurečková (1992) using parametric linear programming methods. This procedure is quite reliable, however it is crucially dependent upon simplex steps in the implementation of "br" fitting routine and consequently is only practical in sample sizes up to a few thousand observations.

boot as indicated in the earlier network diagram there are now a plethora of bootstrapping methods designed for various circumstances,

The main objective of the present note is to provide some guidance about which of these methods to use under various circumstances, and possibly to suggest directions for future development.

Given a fitted object constructed by one of the foregoing estimation methods any of the summary methods can, in principle, be used by simply invoking the R command `summary(fit, se = meth)` where `meth` is the name of one of these summary methods. As a general rule, for small problems, that is problems with less than 5000 observations, it is advisable to use the `se = "rank"` approach, which produces asymmetric "percentile intervals based on the rank inversion procedure. However, for larger problems this can be rather slow since many simplex pivoting operations are required. There are various other options to this approach documented in the help file for `rq.fit.br`. These options can be passed along via the `...` argument of the `summary` function. The basic theory of these confidence intervals is sketched in Section 3.5.5 of Koenker (2005) with further references cited there. To illustrate we can consider a log transformed version of the classical Engel food expenditure model.

```
data(engel)
f <- rq(log(foodexp) ~ log(income), tau = 0.5, data = engel)
summary(f)
##
## Call: rq(formula = log(foodexp) ~ log(income), tau = 0.5, data = engel)
##
## tau: [1] 0.5
##
## Coefficients:
##             coefficients lower bd upper bd
## (Intercept) 0.41833      0.06966  0.90053
## log(income) 0.87659      0.80514  0.93016
```

Another limitation of the rank method is that it operates coefficient by coefficient so it doesn't know how to cope with covariance matrix estimation for the vector of coefficients as would be required,for example, by the `anova` test procedures. By default if no `se` option is specified, the rank method is used if the sample size is less that 1001 and the covariance option for `summary` is not requested, otherwise by default the `se = "nid"` option is invoked. For historical reasons, the "nid" option is treated as a default in such situations even though in retrospect it might have been preferable to use the `se = "ker"` option. The `se = "iid"` option should be avoided except in rare cases where one is very confident about the iid assumption.

When multiple τ's are specified in the fitting function then a list of fitted objects are returned of class `rqs`, and summary then acts on this list sequentially returning a list of tables of coefficients and their confidence intervals in the case that the `se = "rank"` option is used, or standard errors, t-statistics and p-values in the case that one of the other options is used.

4. Bootstrap Methods for Quantile Regression

As is evident in the network diagram there are many flavors of the bootstrap for quantile regression and the task in this section is to try to make some sense out of which of them are appropriate in various settings. Let's begin with a brief description of the candidates, in approximate order of introduction into the **quantreg** package:

xy	The classical progenitor of all these methods is the "xy" bootstrap studied by Bickel and Freedman (1981) for the mean regression model. Pairs (x_i, y_i) are drawn at random with replacement, the model is reestimated for each of the R bootstrap samples and a covariance matrix is estimated from the R by p estimates of the coefficients.
pwy	the Parzen et al. (1994) bootstrap exploits the fact that the subgradient condition for optimality of the quantile regression estimator is a pivotal statistic, so one can resample Bernoulli random variables and again construct R new estimates of the coefficients to produce a covariance matrix.
spwy	a variant of the "pwy" method that uses the sparse Frisch-Newton method, intended specially for the "cluster" bootstrap method and used automatically when the design matrix of the fitted object is stored in sparse form.
wxy	instead of the multinomial sampling of the "xy" method one can resample exponential weights and construct weighted quantile regression estimates; this has the advantage that all the observations appear in each replication, albeit with different weights, so it avoids singularities of the resampled design matrices that can sometimes cause problems.
jack	a variant of the "xy" method that operates on subsamples proposed by Portnoy (2014).
wild	a variant of the wild bootstrap that has proven valuable in other settings adapted to the quantile regression and proposed by Feng et al. (2011).
MCMB	a Markov chain resampling scheme proposed by Kocherginsky et al. (2004) that updates one coefficient at a time to accelerate inference for large sample sizes,
BLB	the "bag of little bootstraps" proposed by Kleiner et al. (2014) intended for large samples.
cluster	a variant of the wild bootstrap adapted to clustered observations proposed by Hagemann (2017)
conquer	a variant of the "wxy" method implemented in the **conquer** package of He et al. (2020) and using the smoothing methods developed there, intended for large problems.
extr	a variant of the "pwy" bootstrap adapted for inference for extreme quantiles and proposed by Chernozhukov et al. (2018)
pwxy	A variant of the "wxy" bootstrap implemented in fortran and using preprocessing to accelerate computation, suitable for large applications.

To compound the choice problem each of these options has several sub-options or tuning parameters and potentially more than one implementation in terms of lower level languages. The typical implementation draws random variables required to construct the bootstrap samples, and passes this information to a lower level, either Fortran or in the case of MCMB a C routine, where the matrix of bootstrap coefficients is computed inside a loop. This strategy keeps all the random number generation in R where it is subject to the usual reproducibility mechanisms and avoids potential slow looping constructs in R. In large samples, however, it is subject to memory constraints and overlooks potential opportunities for parallelization.

The selection of a bootstrap method is controlled by the `bsmethod` argument to the `boot.rq` function and can be passed as a argument to `summary`. One of the earliest innovations in quantile regression bootstrapping was introduced in Buchinsky (1994) who proposed computing bootstrap replications on subsamples of the observations rather than on samples of the original size n. This was later justified more formally in Bickel and Sakov (2008) and became the subject of considerable subsequent research. I enjoy the irony that for Moshe it was a case of "necessity as the mother of invention," a way to speed up progress on his thesis research. This "m out of n" version of the bootstrap is implemented with the `mofn` argument of the `boot.rq` function. The number of bootstrap replications is also an important influence on both the speed and reliability of the method; the default of $R = 200$ is probably somewhat smaller than would be ideally specified.

An important aspect of the bootstrapping ecosystem in **quantreg** is the relation of bootstrap methods and fitting methods. Early on it was recognized that doing bootstrap replications in an R loop was painfully slow as soon as samples became moderately large, so most of the early methods were implemented by passing sampling weights to a wrapper function that called an implementation of a simplified "br" method inside a fortran loop. This is undesirable since if users have –in their wisdom – estimated their model with the "fn" option or the "sfn" method they should be entitled to expect that the bootstrapping that is done in with the summary method would use the same fitting procedure. An exception to this state of affairs is the "cluster" bootstrap method, which uses either "fn" or "sfn" depending upon whether the design matrix associated with the fitted object is an ordinary R matrix, or special sparse matrix. There is also a prototype "pxy" bootstrap method that implements a variant of the preprocessing strategy used for the "ppro" and "pfnb" fitting methods. It would seem to be desirable to develop a fortran version of this intended for large problems.

To this end, I've coded a new `boot.rq.pwxy` option that implements the preprocessing method within a fortran loop. Unlike earlier fortranizations of bootstrap methods that passed a n by R array of data to construct the bootstrap samples, this implementation draws bootstrap weights inside the fortran loop one replication at a time. The former strategy is fine for small sample sizes, but is obviously impractical in terms of memory requirements when n is large. One may worry that generating random variates from within fortran would disrupt the usual reproducibility mechanisms of `set.seed`, etc. Fortunately, R provides functionality to resolve this quite easily. In contrast to the prototype `boot.rq.ppro` which uses multinomial sampling, this version uses the weighted bootstrap with standard exponential weights, which should be more robust especially in settings with factor covariates.

Figure 6 compares timings for three bootstrap methods: "wxy" the fortran simplex version using exponential weighting, "pxy" the Frisch-Newton R version using preprocessing and multinomial sampling, and "pwxy" the fortran Frisch-Newton version using preprocessing and exponential weighting. Timings are reported in seconds for $R = 200$ replications of the bootstrap procedure. The simplex version is clearly not competitive except perhaps at lowest sample size, $n = 500$. In the largest sample size of the experiment, $n = 625,000$ the simplex procedure is between 100 and 500 times slower than the preprocessed procedures. Somewhat surprisingly, at least to me, the fortran preprocessing version is only slightly faster than the R preprocessing version, and then only when the model dimension is moderate. Whether this can be improved by better choice of tuning parameters remains an open question. There is an obvious advantage in multinomial sampling in that it yields a further reduction in sample

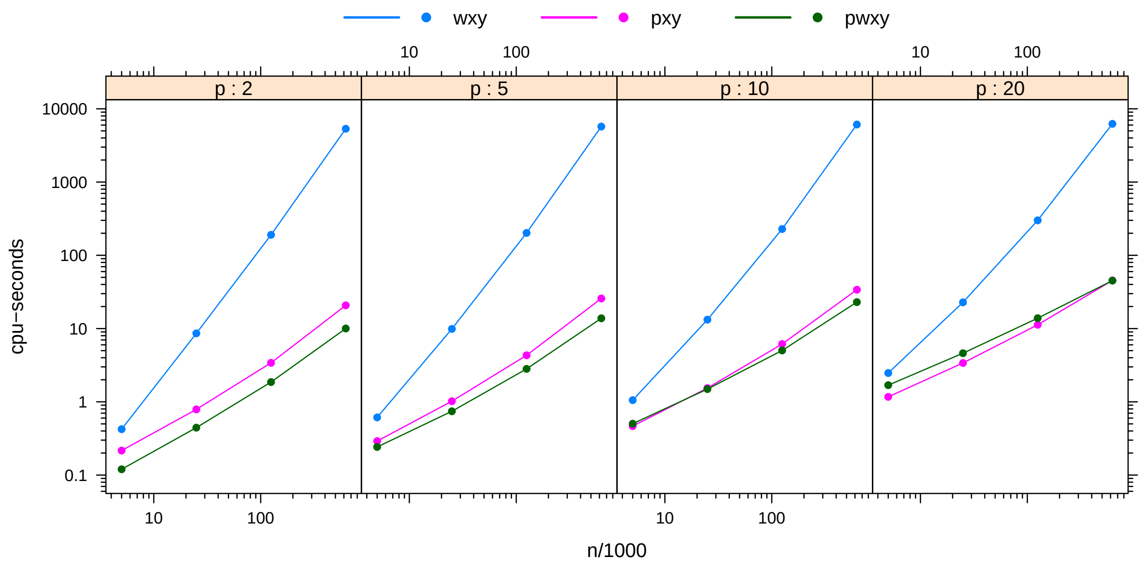

Figure 6. Comparison of Three Bootstrapping Methods

size, but this is offset – in my view – by the concomitant increase in risk of design matrix singularities. There is a rather delicate balance in setting the width of the initial confidence band in any implementation of the preprocessing approach; in our bootstrap experiment I've used $\tau \pm \sqrt(np)/2n$. This choice trades off the advantage of reduced sample size against the likelihood of violations of the predicted signs condition that require repeatedly solving the globbed problem. Further investigation of this trade-off is clearly warranted. Some further experimentation with the form of the confidence band has also been suggested by Portnoy (1997). Clearly as p becomes larger the problem becomes more challenging; in many large p applications one might hope that the design matrices would be sufficiently sparse to justify using the sparse Frisch-Newton algorithm with preprocessing. This suggests yet another option to add to the todo list.

5. Smoothing, Gradient Descent and Other Perturbations

Steve Portnoy organized a session at the 2010 JSM in Vancouver with Laurie Davies, Emmanuel Candès and I. At lunch after the session Candès made a very persuasive case for the obsolescence of interior point methods for "big data" applications, arguing that in high dimensional problems it wasn't practical to do repeated Cholesky factorization of large Hessian matrices. Instead, one needed to rely on first order methods like gradient descent. In my usual lackadaisical fashion I began reading and experimenting with such methods originally motivated by Parikh and Boyd (2013) and the elegant R implementation of Fougner (2014), I implemented a proximal gradient method for fitting quantile regression models. Some experiments with this approach are reported in Koenker (2018). This was my introduction to what we might call "spurious precision in statistical computing." At the time I was very conscious of precision in convex optimization because I was beginning to compare interior point methods for computing the NPMLE for various mixture problems with solutions from the EM algorithm. EM was very slow to converge, while interior point methods produced

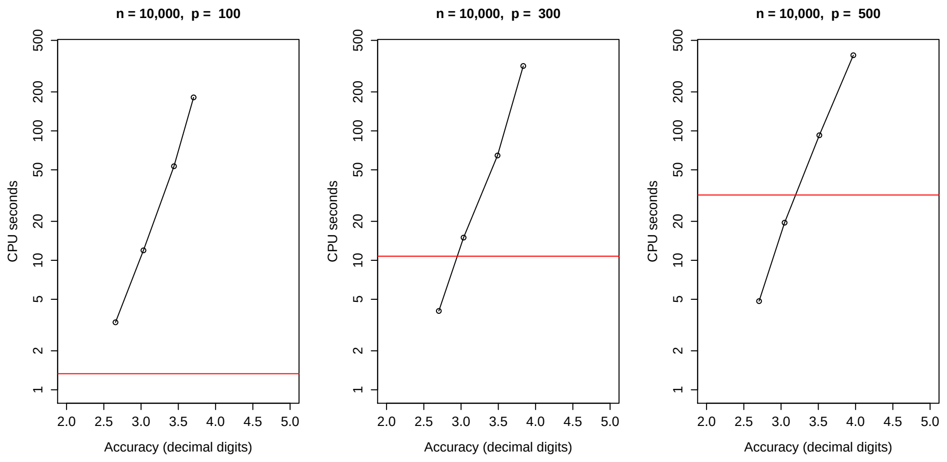

Figure 7. Accuracy vs. Computational Effort: CPU effort (in seconds) is plotted against accuracy in the number of correct decimal digits, averaged over the p coefficients for the POGS GPU solutions to the primal quantile regression problem. Baseline accuracy is determined by the interior point solution depicted by the horizontal (red) line, which is accurate to at six decimal digits. Although the POGS procedure is quite quick to produce a solution with two to three digit accuracy, the effort required to produce better accuracy increases rapidly. In contrast, there is little advantage observed in the interior point timings when the convergence tolerance is relaxed.

much more accurate solutions very quickly. In the mixture setting since one was estimating a distribution function on a relatively fine grid of a few hundred points, the differences in precision were very apparent visually: EM even after a few thousand iterations still produced a rather smooth approximation of the mixing distribution, while the interior point method rapidly converged to solutions with just a few point masses, as expected from the theory.

Something similar happens with the QR computing in the sense that simplex and interior point methods converge to essentially identical solutions, say up to 7 or 8 decimal digits, whereas the proximal method was quite quick to achieve 2 or 3 digits of precision, but very slow to make further progress. This is depicted in Figure 5 taken from Koenker (2018). However, in the case of the mixture problems accuracy seems important because it reveals a qualitative difference in the nature of the solutions, in the QR setting it is less clear. If we are plotting results, or even if we are reporting tables of results, it is unclear that we need more than a few “significant” digits. After all the original data is rarely more accurate than that. Maybe accuracy is just a fetish? It should be noted, especially so since it wasn’t noted in the original source, that the problem dimensions of Figure 5 are modest by modern standards, and much greater computational gains could be made in larger problems with better GPU hardware and thus more parallelism. I recall Candès mentioning daily CCTV footage from Beijing.

There have been several proposals for implementation of the ADMM (alternating direction method of multipliers) and related first order approaches for quantile regression, but none really attracted my attention until Xuming He sent a preprint of He et al. (2020). Drawing on prior work of Fernandes et al. (2021) and Barzilai and Borwein (1988), Xuming and his

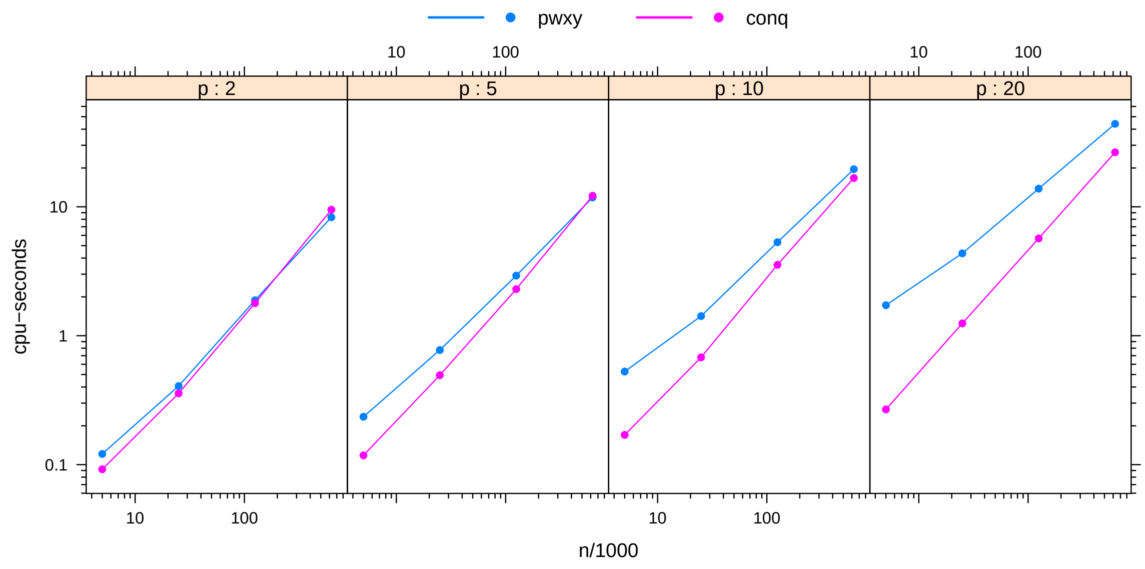

Figure 8. Comparison of Two More Bootstrapping Methods

coauthors develop a scalable gradient descent based algorithm that performs very efficiently on large problems. Their approach has been implemented in the R package, **conquer** and I have created a hook in **quantreg** so that both fitting and summary methods can easily access it. In contrast to the interior point methods implemented in **quantreg** that can be seen as taking full Newton steps for a log-barrier formulation of the underlying linear programming problem, their approach adopts a quasi-Newton, gradient descent approach that avoids the full Cholesky factorization at each iteration. Step sizes are chosen according to a proposal of Barzilai and Borwein (1988). A crucial aspect of the new method is the smoothing of the usual quantile objective function as recently proposed by Fernandes et al. (2021). In contrast to other proposed strategies for smoothing their convolution method preserves the convexity of the objective function and leads to improved performance. Of course a consequence of this smoothing, is that the original problem is somewhat perturbed, but this too may have a silver lining. Provided that the conditional density of the response with respect to the covariates satisfies weak smoothness conditions, smoothing the objective function results in a modest efficiency gain. This is analogous to other methods of smoothing as discussed in Koenker (2005) Section 5.2. The experiments in Fernandes et al. (2021) and He et al. (2020) bear out this theoretical finding.

The improvement in computational speed is particularly important in accelerating bootstrap computations, so I was obviously interested to see how this new gradient descent approach compared to the preprocessing approach described above as "pwxy." Figure 8 reveals that, indeed, gradient descent as implemented in the **conquer** package is substantially quicker than "pwxy," although it appears that for very large sample sizes perhaps it is catching up.

A final comparison for this section involves the "bag of little bootstraps" of Kleiner et al. (2014) designed for "big data" applications. The implementation of this approach in **quantreg** is somewhat cheesy in the sense that there is no real parallelism as one might expect given the original description of the method. On the other hand, I was curious to see how it might

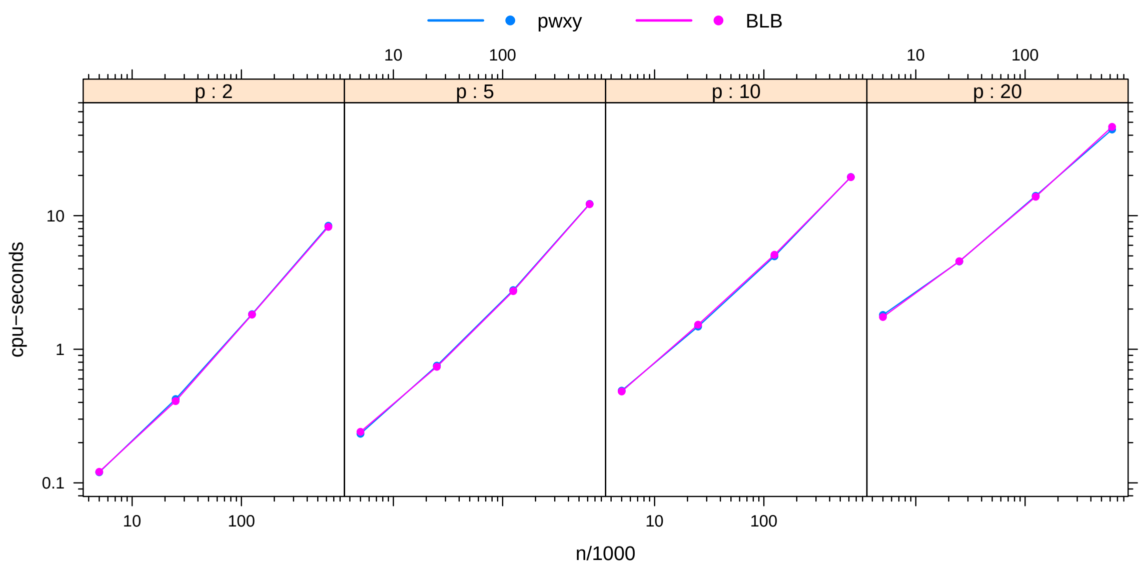

Figure 9. Comparison of Two More Bootstrapping Methods

compare with the preprocessing approach of `boot.rq.pwxy`. Figure 9 illustrates comparison. The "BLB" method adopts the suggested default using groups of size $\lfloor n^{0.7} \rfloor$ and for each group generating a bootstrap sample using `boot.rq.pwxy`, so effectively we are comparing doing many smaller sample size preprocessed bootstraps and then aggregating them, to doing one larger preprocessed bootstrap. Curiously, at least over our problem domain, the timings are almost identical! Of course, this suggests that in a world where one has access to many cores the "BLB" option seems quite promising. If/when I decide to upgrade to a M1 Apple Mini, I may try to revisit this comparison.

6. Conclusion

What have we learned? When the sample size is less than about 10,000, the ancient simplex machinery is quite serviceable even for conventional bootstrapping. However, for larger sample sizes it is advantageous to turn to interior point methods especially when one wants solutions for a large number of quantiles, or when bootstrapping is desired. In both cases preprocessing can help to significant speed up the computations. Further exploration of the tuning parameter selection for preprocessing is certainly warranted. When the parametric dimension of the model is large, but the design matrix is sparse as in our experiment for Figure 3, there is a large gain to using the sparse version of the Frisch-Newton method. Unfortunately, there is (as yet) no preprocessed sparse version for multiple taus, or for bootstrapping so this is something that should probably be on the todo list. Finally, there is significant scope for further exploration of first order methods along the lines of the **conquer** package.

Some progress has been made with the introduction of the fortran versions called by `rq.fit.pfnb` and `boot.rq.pwxy`, although this has the obvious downside that it adds to the clutter of options for both fitting and inference. It is difficult with only a handful of experimental settings to be precise about problem dimension boundaries and choice of methods.

As always, I would appreciate hearing about user experience, both good and bad. This is main mechanism for improvement.

References

Barrodale, I. and Roberts, F. (1974), 'Solution of an overdetermined system of equations in the ℓ_1 norm', *Communications of the ACM* **17**, 319–320.

Barzilai, J. and Borwein, J. M. (1988), 'Two-point step size gradient methods', *IMA J. Numerical Analysis* **8**, 141–48.

Bickel, P. J. and Freedman, D. A. (1981), 'Some asymptotic theory for the bootstrap', *The Annals of Statistics* **9**, 1196–1217.

Bickel, P. J. and Sakov, A. (2008), 'On the choice of m in the m out of n bootstrap and confidence bounds for extrema', *Statistica Sinica* **18**, 967–985.

Buchinsky, M. (1994), 'Changes in US wage structure 1963-87: An application of quantile regression', *Econometrica* **62**, 405–458.

Chernozhukov, V., Fernandez-Val, I. and Kaji, T. (2018), Extremal quantile regression, *in* R. Koenker, V. Chernozhukov, X. He and L. Peng, eds, 'Handbook of Quantile Regression', CRC Press.

Chernozhukov, V., Fernandez-Val, I. and Melly, B. (2020), 'Fast algorithms for the quantile regression process', *Empirical Economics* **?**, ?–?

Feng, X., He, X. and Hu, J. (2011), 'Wild bootstrap for quantile regression', *Biometrika* **98**, 995–999.

Fernandes, M., Guerre, E. and Horta, E. (2021), 'Smoothing quantile regressions', *Journal of Business & Economic Statistics* **39**, 338–357.

Fougner, C. (2014), *POGS: Proximal Operator Graph Solver*. `https://github.com/foges/pogs`.

Gutenbrunner, C. and Jurečková, J. (1992), 'Regression quantile and regression rank score process in the linear model and derived statistics', *Ann. Statist.* **20**, 305–330.

Hagemann, A. (2017), 'Cluster-robust bootstrap inference in quantile regression models', *Journal of the American Statistical Association* **112**, 446–456.

He, X., Pan, X., Tan, K. M. and Zhou, W.-X. (2020), *conquer: Convolution-Type Smoothed Quantile Regression*. `https://CRAN.R-project.org/package=conquer`.

Kleiner, A., Talwalkar, A., Sarkar, P. and Jordan, M. (2014), 'A scalable bootstrap for massive data', *J. Royal Statistical Society (B)* **76**, 795–816.

Kocherginsky, M., He, X. and Mu, Y. (2004), 'Practical confidence intervals for regression quantiles', *J. of Comp. and Graphical Stat.* . forthcoming.

Koenker, R. (2005), *Quantile Regression*, Cambridge U. Press.

Koenker, R. (2018), Computational methods for quantile regression, *in* R. Koenker, V. Chernozhukov, X. He and L. Peng, eds, 'Handbook of Quantile Regression', CRC Press.

Koenker, R. and d'Orey, V. (1987), 'Computing regression quantiles', *Applied Statistics* **36**, 383–393.

Mehrotra, S. (1992), 'On the implementation of a primal-dual interior point method', *SIAM J. of Optimization* **2**, 575–601.

Parikh, N. and Boyd, S. (2013), 'Proximal algorithms', *Foundations and Trends in Optimization* **1**, 123–221.

Parzen, M. I., Wei, L. and Ying, Z. (1994), 'A resampling method based on pivotal estimating functions', *Biometrika* **81**, 341–350.

Portnoy, S. (1997), On computation of regression quantiles: Making the laplacian tortoise faster, *in* Y. Dodge, ed., 'L_1-statistical procedures and related topics', Institute of Mathematical Statistics, Hayward, CA, pp. 187–200.

Portnoy, S. (2014), 'The jackknife's edge: Inference for censored regression quantiles', *Computational Statistics & Data Analysis* **72**, 273–281.

Portnoy, S. and Koenker, R. (1997), 'The Gaussian hare and the Laplacian tortoise: Computability of squared-error versus absolute-error estimators, with discusssion', *Statistical Science* **12**, 279–300.

Powell, J. L. (1991), Estimation of monotonic regression models under quantile restrictions, *in* W. Barnett, J. Powell and G. Tauchen, eds, 'Nonparametric and Semiparametric Methods in Econometrics', Cambridge U. Press: Cambridge.

R Core Team (2020), *Writing R Extensions*, R Foundation for Statistical Computing, Vienna, Austria. `https://www.R-project.org/`.