An Introducion to QCD

Anonymous Authors

December 11, 2025

Contents

Introduction

`QCD` is a package that solves penalized quantile regression via exact coordinate descent method. The penalties considered are LASSO, SCAD, and MCP. Note that QCD algorithms for SCAD and MCP are experimental.

This vignette describes basic usage of functions related to ℓ_1 penalized quantile regression model in `QCD` package in R.

`QCD` mainly solves the following problem. Given data points $(x_1, y_1), \ldots, (x_n, y_n)$, where $y_i \in \mathbb{R}$ is a numerical response variable, and $x_i \in \mathbb{R}^p$ is a p-dimensional covariate,

$$\arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n} \rho_\tau \left(y_i - x_i^\top \beta\right) + \lambda \sum_{j=1}^{p} |\beta_j|$$

where $\rho_\tau(u) := u\left(\tau - \mathbf{I}(u < 0)\right)$ is the check loss function, and λ is a penalty parameter to be chosen in a data-driven fashion. λ could be a grid of values covering the entire range of possible solutions.

Installation

You can download the `QCD` package from Github.

```
library(devtools)
install_github("anonQCD/QCD")
```

Quick Start: QCD

The purpose of this section is to give users a general sense of the package regarding QCD. We will briefly go over the main functions, basic operations and outputs. `QCD` can be loaded using the `library` command in the standard way:

```
library(QCD)
```

QCD solves for the exact solution of penalized quantile regression under high-dimensional settings.

High-Dimensional Data Generation

The function generate.data() simulates data for penalized quantile regression, based on Peng and Wang (2015). It creates a sparse linear model with correlated covariates and heteroskedastic noise. User can freely adjust signal-to-noise ratio and autocorrelation rate in the covariance function. Note that in our work, we do not consider intercept and we can change the value of true beta through signal argument.

We generate a high-dimensional dataset for illustration, where the number of predictors exceeds the number of observations with n = 50, p = 150. Reader can also specify different configurations of data generating process in the following code chunk.

```
n <- 50
p <- 150
set.seed(9)
data <- generate.data(n = n, p = p, tau = 0.3,
                      signal = 1, autocorrelation = 0.5)
x <- data$X
y <- data$Y
```

QCD Under Lasso Penalty

With single value of λ, we can fit the ℓ_1 penalized quantile regression using 'qcd.lasso.fit'.

```
qr.lasso <- qcd.lasso.fit(x = x, y = y,
tau = 0.3, lambda = 0.01,
thresh = 1e-06, maxit = 100)
```

This function uses exact coordinate-wise minimization for the quantile check loss, which attains similar accuracy over approximate methods. It is particularly suitable for high-dimensional problems (i.e., when `p >> n`), where traditional solvers like linear programming (LP) become computationally expensive.

QCD Under Non-Convex Penalties

We can fit the SCAD and MCP penalized quantile regression using 'qcd.scad.fit' and 'qcd.mcp.fit' with single value of λ. These functions are experimental.

```
qr.scad <- qcd.scad.fit(x = x, y = y,
                        tau = 0.3, lambda = 0.01,
                        a = 2.2,
                        thresh = 1e-06, maxit = 100)
qr.scad
```

```
qr.mcp <- qcd.mcp.fit(x = x, y = y,
                      tau = 0.3, lambda = 0.01,
                      a = 2.2,
                      thresh = 1e-06, maxit = 100)
qr.mcp
```

Across a grid of lambdas

The `qcd.path()` function solves penalized quantile regression over a grid of `lambda` values using **pathwise coordinate descent algorithm**. It supports LASSO, SCAD, and MCP penalties, and can optionally apply warm start and nudge to improve numerical stability and computation time along the solution path.

```
## Create lambda grid
upper <- 5; lower <- -7
```

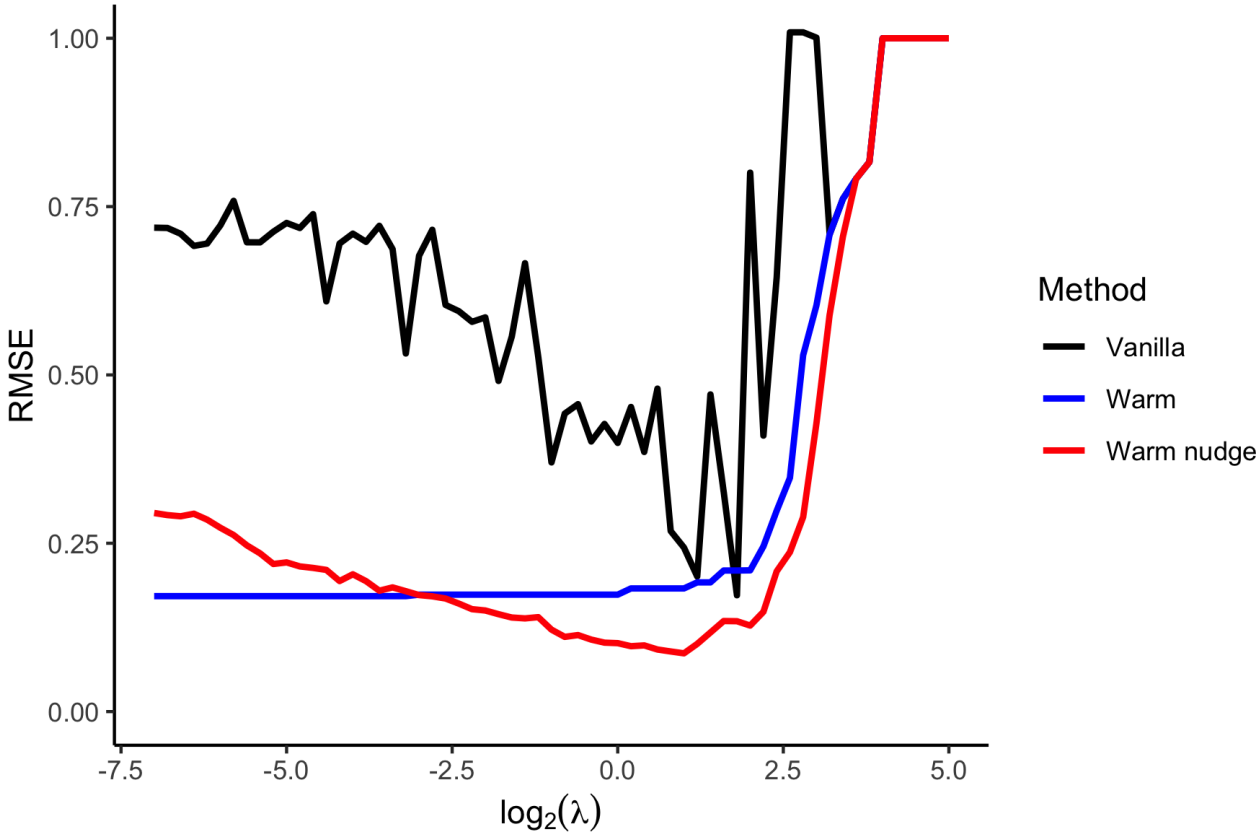
```r
lambda.grid <-  2^seq(upper, lower, by = -0.2)

## warm start version
qr.lasso.warm = qcd.path(x = x, y = y, tau = 0.3,
                         funname = "LASSO", lambda = lambda.grid,
                         nudge = FALSE,
                         thresh = 1e-06, maxit = 1000)

qr.lasso.warm

## warm start and nudge version
set.seed(1)
qr.lasso.warm.nudge = qcd.path(x = x, y = y, tau = 0.3,
                               funname = "LASSO", lambda = lambda.grid,
                               nudge = TRUE, nudgesd = 0.01,
                               thresh = 1e-06, maxit = 1000)

qr.lasso.warm.nudge
```

Comparison of RMSE path

We will evaluate the estimation accuracy of QCD algorithm using three different strategies: vanilla, warm start, and warm nudge. We will measure the performance using RMSE.

```r
# true coefficient
true_beta <- data$true_beta

# Helper function for Relative Squared Error
calc_rmse <- function(beta_hat, beta_true) {
  sum((beta_hat - beta_true)^2) / sum(beta_true^2)
}

# Vanilla
rmse.vanilla <- numeric(length(lambda.grid))
b0 <- rep(0, p)

for (l in seq_along(lambda.grid)) {
  fit <- qcd.lasso.fit(x = x, y = y, tau = 0.3,
                       lambda = lambda.grid[l],
                       warm = NULL, # No warm start
                       thresh = 1e-4, maxit = 100)

  rmse.vanilla[l] <- calc_rmse(fit$beta, true_beta)
}

# Warm Start
fit.warm <- qcd.path(x = x, y = y, tau = 0.3,
                     funname = "LASSO",
                     lambda = lambda.grid,
                     nudge = FALSE,
                     thresh = 1e-4, maxit = 100)

rmse.warm <- apply(fit.warm$beta, 2, calc_rmse, beta_true = true_beta)
```

```r
# Warm Nudge
set.seed(1)
fit.nudge <- qcd.path(x = x, y = y, tau = 0.3,
                      funname = "LASSO",
                      lambda = lambda.grid,
                      nudge = TRUE, nudgesd = 0.01,
                      thresh = 1e-4, maxit = 100)

rmse.warm.nudge <- apply(fit.nudge$beta, 2, calc_rmse, beta_true = true_beta)

# Plot the RMSE path
QCD_df <- data.frame(
  lambda = log2(lambda.grid),
  rmse = c(rmse.vanilla, rmse.warm, rmse.warm.nudge),
  Method = rep(c('Vanilla', 'Warm', 'Warm nudge'), each = length(lambda.grid))
)

library(ggplot2)
ggplot(QCD_df, aes(x = lambda, y = rmse)) +
  geom_line(aes(colour = Method), linewidth = 1) +
  labs(x = expression(log[2](lambda)), y = "RMSE") +
  theme_classic() +
  scale_colour_manual(values = c("black", "blue", "red")) +
  coord_cartesian(xlim = c(-7, 5), ylim = c(0, 1))
```



From above RMSE plot, vanilla QCD displays bumpy regularization path. Once warm start is implemented, the regularization path smoothens but the RMSE remains constant after a specific λ, indicating it is stuck at a suboptimal point. If a nudge is added to the warm start, the regularization path shows a U shape,

suggesting that nudge helps our QCD algorithm converge to an optimal point.

Runtime Comparison with LP and QICD

We compare the runtime of QCD with the benchmark methods: LP and Quantile Iterative Coordinate Descent (QICD). QCD runs faster than LP and QICD.

```
library(quantreg)
library(QICD)

p <- 300
n <- 50
tau <- 0.3
upper <- 5; lower <- -5
lambda.grid <- 2^seq(upper, lower, by = -0.2)

data <- generate.data(p = p, n = n, signal = 1, tau = tau)
x <- data$X
y <- data$Y

# Method 1: QICD
t.qicd <- NA
ptm <- proc.time()
for (l in seq_along(lambda.grid)) {
  fit.qicd <- QICD(x = x, y = y, tau = tau,
                   lambda = lambda.grid[l], intercept = FALSE, funname = 'lasso')
}
t.qicd <- proc.time() - ptm

# Method 2: QCD
set.seed(9)
ptm <- proc.time()
fit.qcd <- qcd.path(x = x, y = y, tau = tau,
                    funname = "LASSO",
                    lambda = lambda.grid,
                    nudge = TRUE, nudgesd = 0.01, # Warm start with Nudge
                    thresh = 1e-4, maxit = 100)
t.qcd <- proc.time() - ptm

# Method 3: Linear Programming (quantreg)
ptm <- proc.time()
for (l in seq_along(lambda.grid)) {
  fit.lp <- rq.fit.lasso(x = x, y = y, tau = tau,
                         lambda = rep(lambda.grid[l]*2, p))
}
t.lp <- proc.time() - ptm

print(t.lp)
#>    user  system elapsed
#>   4.275   0.019   4.302
print(t.qicd)
#>    user  system elapsed
#>   1.339   0.009   1.351
print(t.qcd)
#>    user  system elapsed
```

```
#>   0.288   0.002   0.290
```