tauBayesW

tauBayesW Package

2025-09-12

Contents

Introduction

The **tauBayesW** package implements four approaches for Bayesian weighted quantile regression:

- **Asymmetric Laplace Distribution (ALD)**: Uses the ALD as a working likelihood, incorporating survey weights in the scale parameter. Posterior inference is carried out via Gibbs sampling.
- **Score**: Builds a working likelihood from estimating equations (score function), following Huang, Xu and Tashnev (2015). It accounts for survey weights and uses adaptive Metropolis–Hastings for inference.
- **Approximate**: Proposed by Wang, Kim and Yang (2018), this method approximates the posterior using the sampling distribution of summary statistics. The Score-based method is a special case, also using adaptive Metropolis–Hastings.
- **Expectation–Maximization (EM) Algorithm**: Implemented for multivariate quantile regression. It leverages the ALD mixture representation to update latent variables and parameters iteratively, targeting posterior modes with much lower computational cost than MCMC.

Also, this vignette compares the performance of **bayesQR** with the three MCMC methods implemented in **tauBayesW** using a simulated dataset and the classic Prostate cancer dataset included in the `bayesQR` package.
We estimate the 25th, 50th, and 75th quantile and compare coefficient estimates across all methods.

tauBayesW usage examples

Let's begin showing how to use tauBayesW methods:

```
library(tauBayesW)
```

Creating Priors

The `prior()` function provides a unified interface to define prior distributions for both **univariate** (`bqr.svy`) and **multivariate** (`mo.bqr.svy`) Bayesian quantile regression models.

- For univariate models, it creates an object of class `bqr_prior`.
- For multivariate models, it creates an object of class `mo_bqr_prior`.

Both objects are **S3 classes** that store the prior information in a structured way.

> The `prior()` function creates an S3 object of class `bqr_prior` that contains the prior information. If the object is omitted in a call to `bqr.svy()` or `mo.bqr.svy()`, a **standard vague prior** is used by default.

Doing an example using tauBayesW

```
my_prior <- tauBayesW::prior(
  p = 5,                                    # Number of regression parameters (including intercept)
  type = "MCMC",                            # Prior type: "MCMC" or "EM"
  beta_mean = rep(0, 5),                    # Vector of prior means for regression coefficients
  beta_cov = diag(1000, 5),                 # Prior covariance matrix (large values = vague prior)
  sigma_shape = 0.001,                      # Shape parameter of Inverse-Gamma prior on variance (ALD o
  sigma_rate = 0.001,                       # Rate parameter of Inverse-Gamma prior on variance (ALD on
  names = c("Int", "x1", "x2", "x3", "x4")  # Optional coefficient names
)
print(my_prior)
#> $b0
#> Int  x1  x2  x3  x4
#>   0   0   0   0   0
#>
#> $B0
#>      Int   x1   x2   x3   x4
#> Int 1000    0    0    0    0
#> x1     0 1000    0    0    0
#> x2     0    0 1000    0    0
#> x3     0    0    0 1000    0
#> x4     0    0    0    0 1000
#>
#> $c0
#> [1] 0.001
#>
#> $C0
#> [1] 0.001
#>
#> attr(,"class")
#> [1] "bqr_prior"
```

Fitting models with different methods

We use the **mtcars** dataset to fit three models — **ALD**, **Score**, and **Approximate** — each at the 0.5 quantile.

```r
set.seed(123)

data(mtcars)

mtcars_scaled <- mtcars
mtcars_scaled[, c("wt", "hp", "cyl")] <- scale(mtcars[, c("wt", "hp", "cyl")])

form <- mpg ~ wt + hp + cyl

p <- length(attr(terms(form), "term.labels")) + 1
prior_mean <- rep(0, p)
prior_cov  <- diag(1000, p)

fit_ald <- bqr.svy(
  form,
  data      = mtcars_scaled,
  quantile  = 0.5,
  method    = "ald",
  niter     = 20000,
  burnin    = 10000,
  thin      = 5,
  prior     = list(b0 = prior_mean, B0 = prior_cov),
  print_progress = 5000
)
#> Iteration 5000 of 20000
#> Iteration 10000 of 20000
#> Iteration 15000 of 20000

fit_score <- bqr.svy(
  form,
  data      = mtcars_scaled,
  quantile  = 0.5,
  method    = "score",
  niter     = 20000,
  burnin    = 10000,
  thin      = 5,
  print_progress = 10000
)
#> Iteration 10000 of 20000
#> Iteration 20000 of 20000

fit_approx <- bqr.svy(
  form,
  data      = mtcars_scaled,
  quantile  = 0.5,
  method    = "approximate",
  niter     = 20000,
  burnin    = 10000,
  thin      = 5,
```

```
  print_progress = 5000
)
#> Iteration 0 of 20000
#> Iteration 5000 of 20000
#> Iteration 10000 of 20000
#> Iteration 15000 of 20000
```

For the EM algorithm, the model is fitted by projecting the multivariate response onto a set of directions. These directions can be supplied by the user through an object `U`; if not provided, the algorithm will automatically generate them.

```
data(mtcars)

Y <- cbind(mtcars$mpg, mtcars$hp)

set.seed(123)
d <- ncol(Y)
n_dir <- 3
U <- matrix(NA_real_, d, n_dir)
for (k in 1:n_dir) {
  u_k <- rnorm(d)
  U[, k] <- u_k / sqrt(sum(u_k^2))
}

p <- 3
prior_mo <- prior(
  p     = p,
  type  = "EM",
  beta_mean   = rep(0, p),
  beta_cov    = diag(1e6, p),
  sigma_shape = 0.001,
  sigma_rate  = 0.001
)

fit_mo <- mo.bqr.svy(
  cbind(mpg, hp) ~ wt + cyl,
  data     = mtcars,
  quantile = c(0.25, 0.5, 0.75),
  U        = U,
  prior    = prior_mo,
  n_dir    = n_dir,
  max_iter = 5000,
  verbose  = FALSE
)
```

We can also fit several quantiles at once by passing a vector of probabilities with `c()`, for example `quantile = c(0.25, 0.5, 0.75)`.

Model Output: Print and Summary Methods

There are also print() and summary() created for the output objects of tauBayesW

We can expore the return of the object bqr.svy

```r
# Print method - shows basic model information
knitr::kable(
  data.frame(
    Information = c("Method", "Quantiles", "Draws", "Burn-in", "Thin"),
    Value = c(fit_ald$method,
              paste(fit_ald$quantile, collapse = ", "),
              length(fit_ald$draws),
              fit_ald$warmup,
              fit_ald$thin)
  ),
  caption = "Model Information - ALD Method"
)
```

Table 1: Model Information - ALD Method

Information	Value
Method	ald
Quantiles	0.5
Draws	10000
Burn-in	10000
Thin	5

```r
# Summary method - detailed convergence diagnostics will be shown in summary section

# Access posterior draws - show structure
if (!is.null(fit_ald$draws)) {
  knitr::kable(
    head(as.data.frame(fit_ald$draws), 6),
    caption = "First 6 posterior draws (ALD Method)",
    digits = 4
  )
}
```

Table 2: First 6 posterior draws (ALD Method)

(Intercept)	wt	hp	cyl	sigma
19.5491	-3.4376	-1.0179	-1.5954	0.7238
19.4543	-2.5142	-0.7504	-2.4742	0.7828
19.5965	-2.6978	-0.5379	-2.8920	0.6449
19.5514	-2.7159	-0.3566	-2.6491	0.5281
19.4124	-3.1352	-0.5566	-2.1270	0.8342
19.4756	-2.9804	-0.8045	-2.3303	0.7001

For the `mo.bqr.svy` it is recommended to use the `summary()` method in order to obtain a cleaner and more structured view of the output. This provides posterior mode estimates, EM convergence information, and scale parameters for each quantile and direction.

Convergence Diagnostics

```
methods <- c("ALD", "Score", "Approximate")
fits <- list(fit_ald, fit_score, fit_approx)

summ_multi <- lapply(fits, summary)

summ_df <- do.call(rbind, lapply(seq_along(summ_multi), function(i) {
  s <- summ_multi[[i]]
  if (!is.null(s$posterior_summary)) {
    df <- as.data.frame(s$posterior_summary)
  } else if (!is.null(s$per_tau)) {
    df <- as.data.frame(s$per_tau[[1]]$coef_summary)
  } else {
    stop("Unexpected summary object structure")
  }
  df$Method <- methods[i]
  df
}))

summ_df <- summ_df[, c("Method", setdiff(names(summ_df), "Method"))]

knitr::kable(
  summ_df,
  caption = "Posterior summary for all methods (Quantile = 0.5)",
  digits = 3
)
```

Table 3: Posterior summary for all methods (Quantile = 0.5)

Method	variable	mean	median	sd	rhat	ess_bulk	ess_tail	q2.5	q97.5	lower_ci	upper_ci
ALD	(Intercept)	19.469	19.462	0.191	1.000	1284.050	1346.805	19.115	19.900	19.115	19.900
ALD	wt	-2.812	-2.813	0.286	1.000	1358.121	1665.570	-3.369	-2.268	-3.369	-2.268
ALD	hp	-0.843	-0.790	0.478	1.000	933.397	994.624	-1.932	0.084	-1.932	-0.084
ALD	cyl	-2.272	-2.277	0.553	1.001	1059.654	1155.988	-3.292	-1.142	-3.292	-1.142
ALD	sigma	0.721	0.708	0.120	1.000	1494.400	1758.235	0.528	0.993	0.528	0.993
Score	(Intercept)	19.452	19.439	0.069	1.010	64.450	178.875	19.360	19.647	19.360	19.647
Score	wt	-2.793	-2.797	0.109	1.007	51.126	113.136	-2.977	-2.599	-2.977	-2.599
Score	hp	-0.689	-0.727	0.157	1.015	49.899	92.489	-0.898	0.301	-0.898	-0.301
Score	cyl	-2.360	-2.310	0.210	1.007	46.176	63.541	-2.982	-2.106	-2.982	-2.106
Approximate	(Intercept)	19.495	19.469	0.092	1.011	65.881	276.033	19.371	19.702	19.371	19.702
Approximate	wt	-2.795	-2.796	0.103	1.001	299.787	409.059	-2.979	-2.597	-2.979	-2.597
Approximate	hp	-0.667	-0.693	0.167	1.020	88.145	225.328	-0.937	0.297	-0.937	-0.297

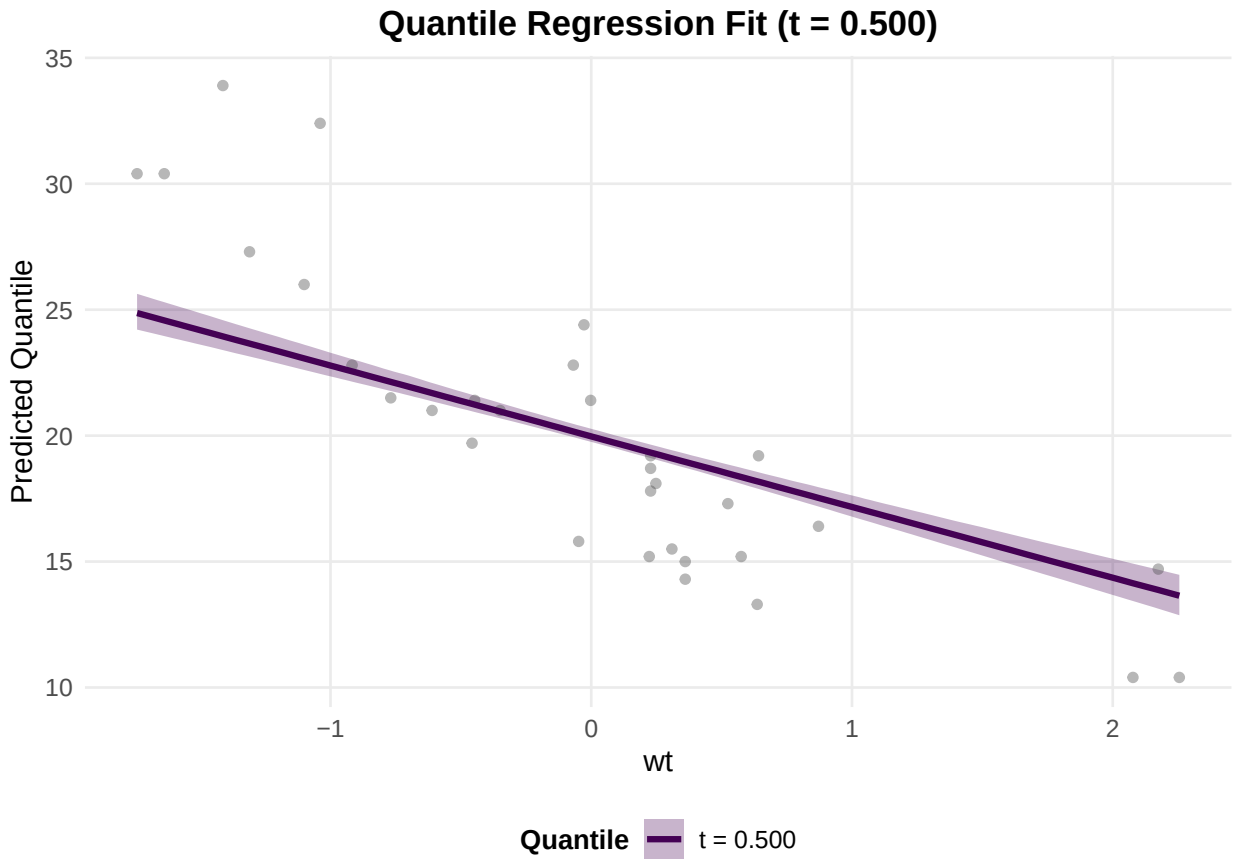
Method	variable	mean	median	sd	rhat	ess_bulk	ess_tail	q2.5	q97.5	lower_ci	upper_ci
Approximateyl		-2.420	-2.372	0.247	1.009	73.988	117.333	-3.022	-2.073	-3.022	-2.073

Visualization functions

The `tauBayesW` package provides comprehensive visualization capabilities through its `plot()` method. The plotting system supports both **base R graphics** (default) and **ggplot2** (when `use_ggplot = TRUE`).

1-. Fitted regression plot Example plotting the fitted regression line using **ggplot2**, including the credible interval as a shaded band around the fitted line.

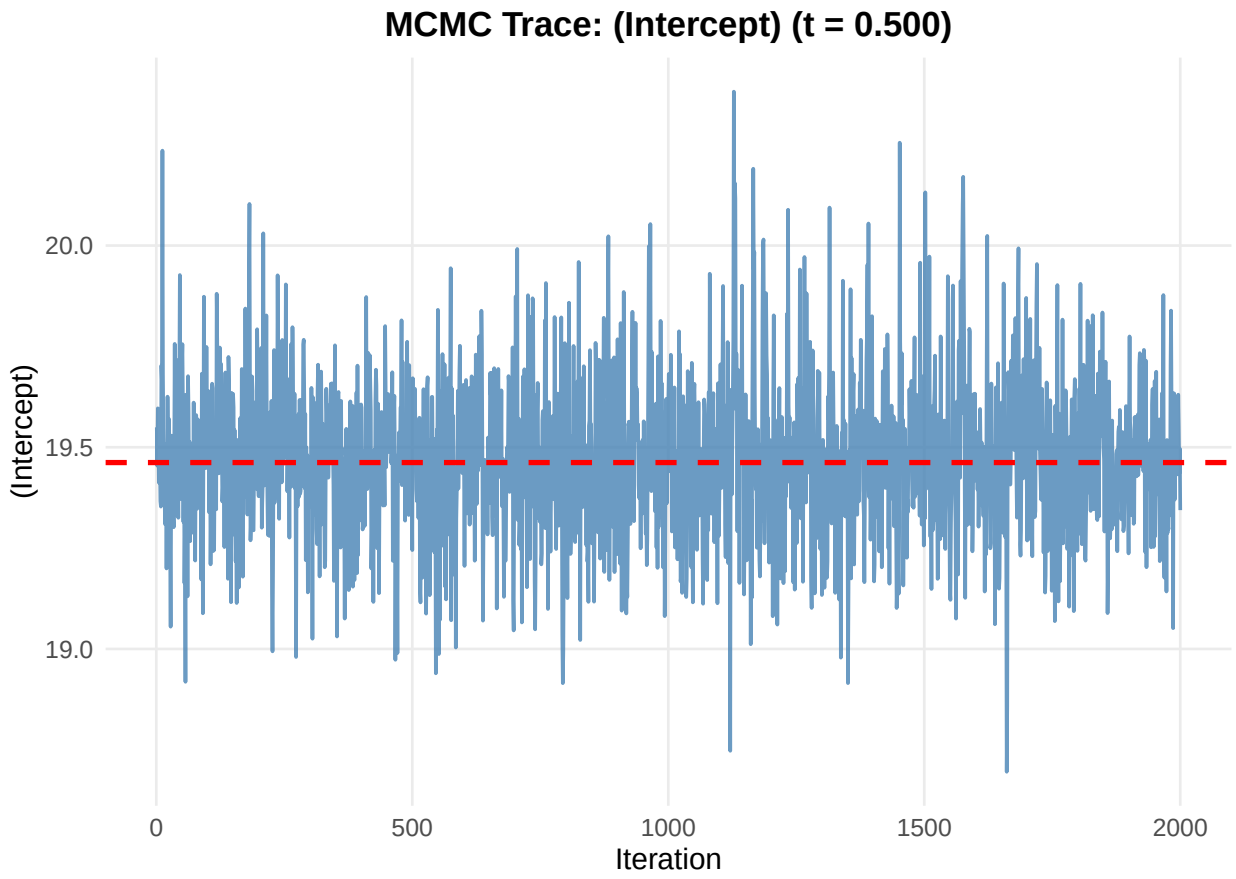
```
plot(fit_ald, use_ggplot = TRUE, show_ci  =TRUE)
```



It is possible to hide the points from the plot by setting the argument `add_points = FALSE`.

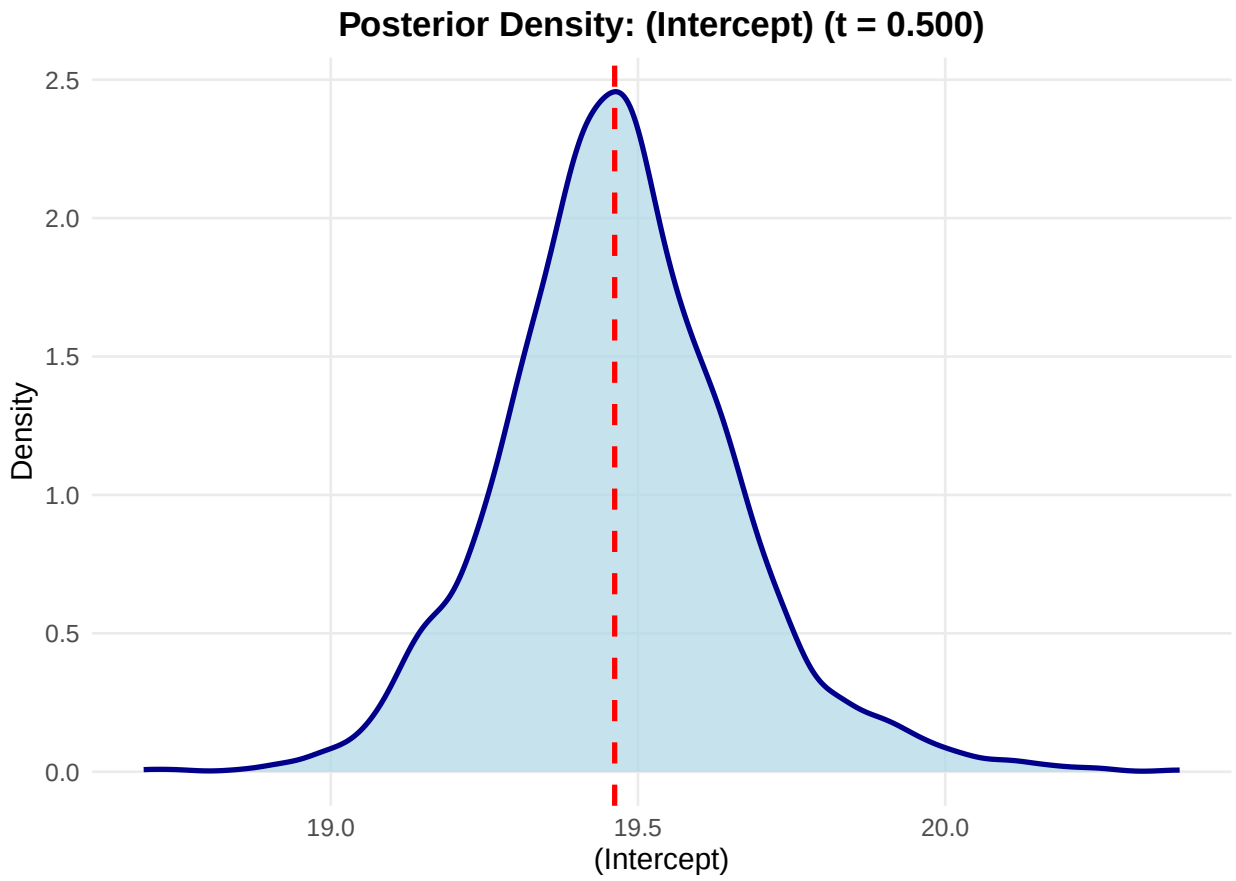
2. Trace Plots Example plotting the MCMC trace of a selected coefficient to assess convergence and mixing across iterations.

```
# Basic quantile plot (requires x_var)
plot(fit_ald, type = "trace", x_var = "hp")
```



3. Density Plots Example plotting the posterior density of a selected coefficient, including the fitted quantile as a reference line.

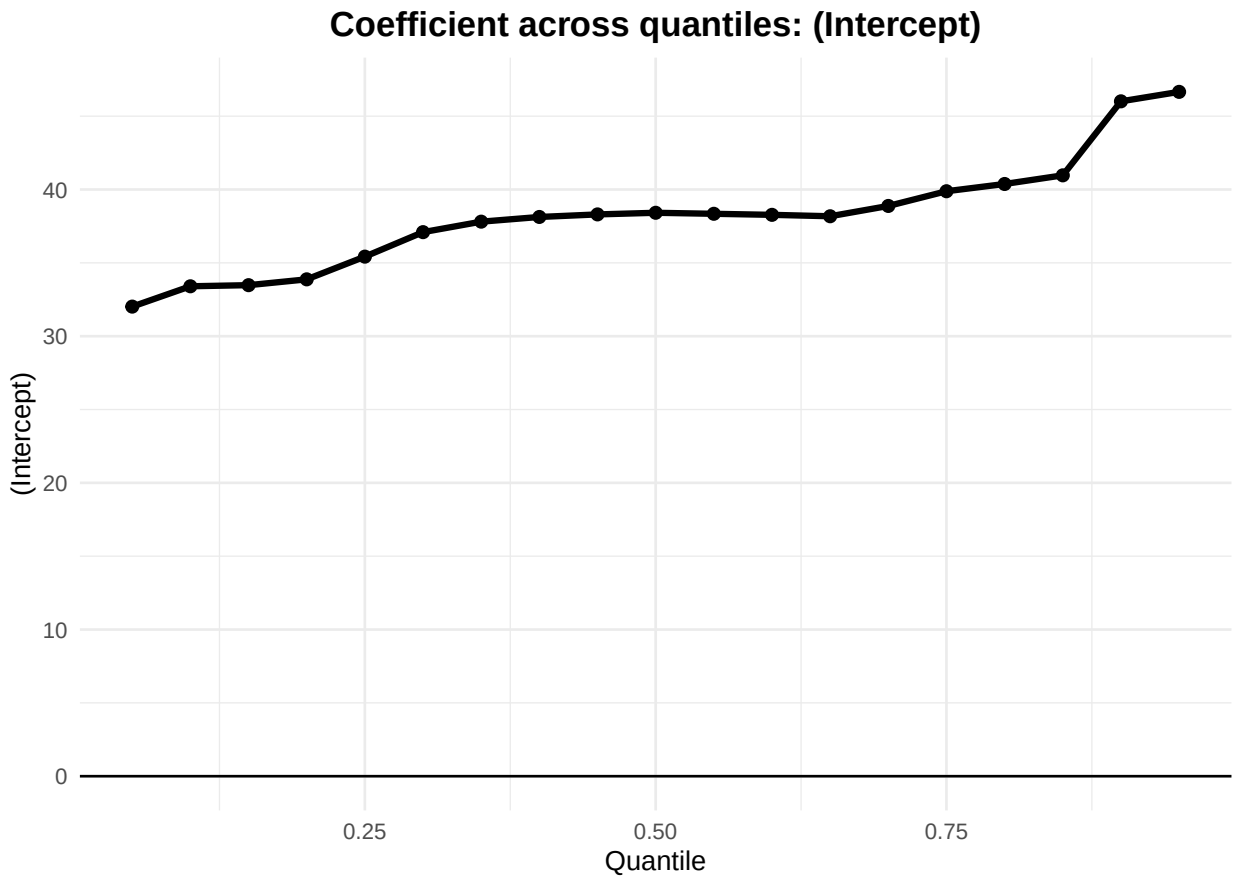
```
# Combined view
plot(fit_ald, type = "density", x_var = "wt")
```



4. Quantile Plots Example plotting the estimated coefficient of a predictor across multiple quantiles.

```
fit_multi <- bqr.svy(
  mpg ~ wt + hp + cyl,
  data = mtcars,
  quantile = seq(0.05, 0.95, 0.05),
  method = "ald",
  niter = 10000,
  burnin = 5000,
  thin = 5,
  print_progress = 10000
)

plot(fit_multi, type = "quantile", which = "(Intercept)")
```

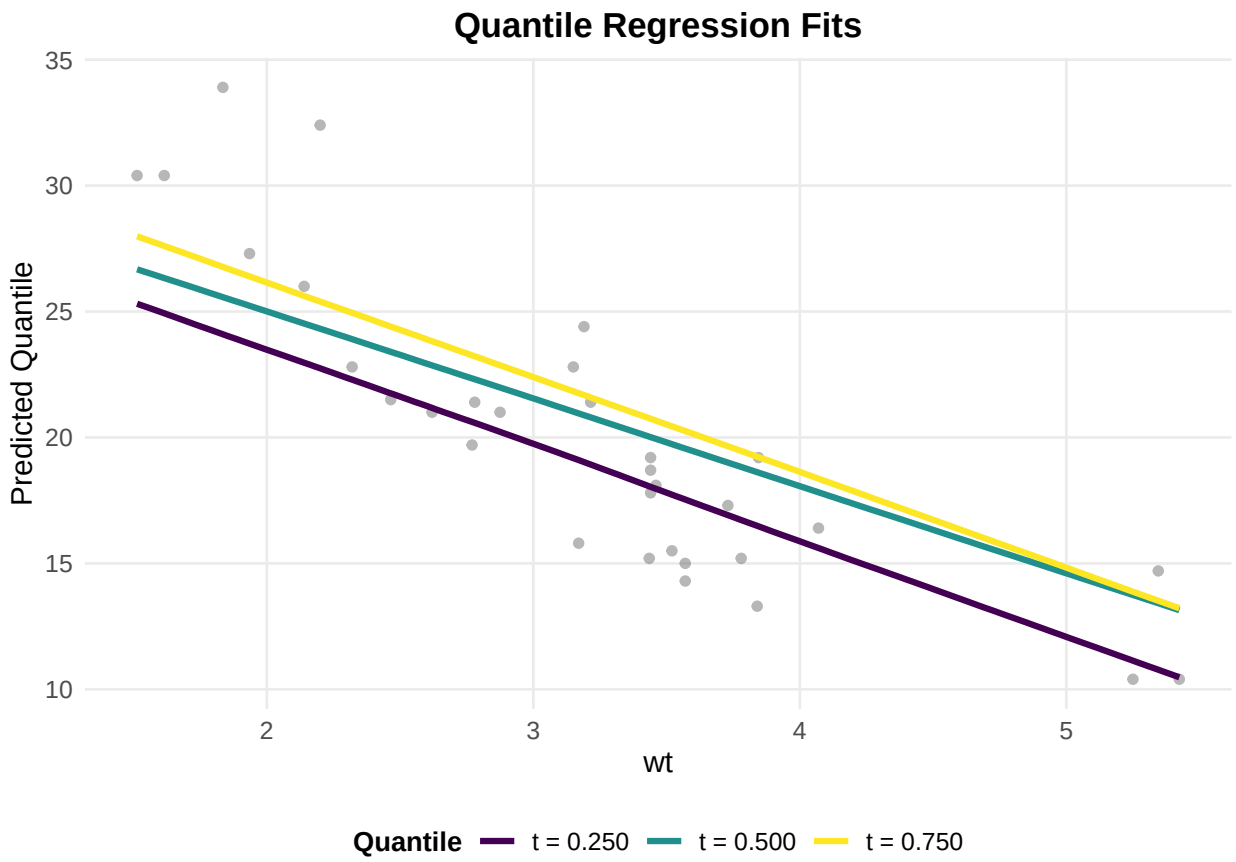


5. Multi-Panel Layouts When working with multiple quantiles, it is often insightful to visualize how the fitted regression lines vary across different levels of the conditional distribution of the response.

```
# Fit model with multiple quantiles
fit_multi <- bqr.svy(mpg ~ wt + hp, data = mtcars,
                     quantile = c(0.25, 0.5, 0.75), method = "ald",
                     niter = 5000, burnin = 2500)
#> Iteration 1000 of 5000
#> Iteration 2000 of 5000
#> Iteration 3000 of 5000
#> Iteration 4000 of 5000
#> Iteration 1000 of 5000
#> Iteration 2000 of 5000
#> Iteration 3000 of 5000
#> Iteration 4000 of 5000
#> Iteration 1000 of 5000
#> Iteration 2000 of 5000
#> Iteration 3000 of 5000
#> Iteration 4000 of 5000
```

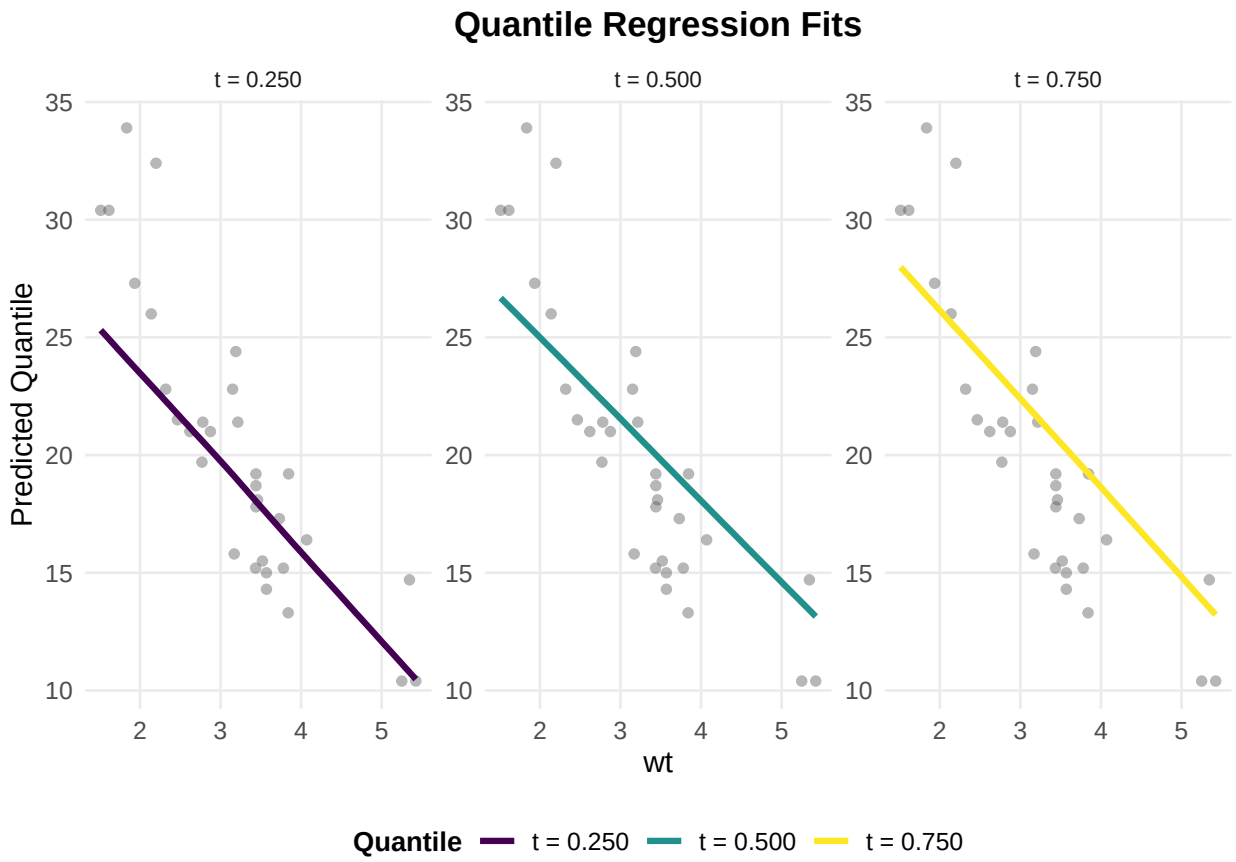
The `plot()` method in **tauBayesW** makes this straightforward: by passing several quantiles in the `tau` argument, the function can overlay multiple fitted curves in the same plot.

```r
# Multi-panel coefficient plot
plot(fit_multi, type = "fit", use_ggplot = TRUE, ncol = 3, combine = TRUE)
```



Alternatively, the user may choose to display separate panels for each quantile by setting `combine = FALSE`.

```r
# Multi-panel coefficient plot
plot(fit_multi, type = "fit", use_ggplot = TRUE, ncol = 3, combine = FALSE)
```



Comparison with bayesQR for the MCMC algorithms and EM algorithm

Simulation Study

We conduct a simulation study to illustrate **tauBayesW** capabilities and compare with **bayesQR** using three different sampling designs: Poisson sampling, stratified sampling, and systematic sampling. Each design simulates complex survey data with known population parameters.

The used simulation functions can be found in the data-raw folder simulation.R file

Simulation Execution We run the simulation function and obtain the following values

```
# Simulation parameters
set.seed(42)
N <- 10000  # Population size
n <- 500    # Sample size
n_est <- 5  # Number of strata for stratified sampling
quantiles <- c(0.25, 0.5, 0.75)

# True population parameters
true_beta <- c(2, 1.5)  # Intercept and slope

# Create simulation parameters table
sim_params <- data.frame(
```

```
  Parameter = c("Population size", "Sample size", "Number of strata", "Quantiles", "True Beta0", "True B
  Value = c(N, n, n_est, paste(quantiles, collapse = ", "), true_beta[1], true_beta[2])
)

knitr::kable(sim_params, caption = "Simulation Study Parameters")
```

Table 4: Simulation Study Parameters

Parameter	Value
Population size	10000
Sample size	500
Number of strata	5
Quantiles	0.25, 0.5, 0.75
True Beta0	2
True Beta1	1.5

```
# Generate data for each sampling design
data_poi <- artificial_data_poi(N, n)
data_est <- artificial_data_est(N, n, n_est)
data_sys <- artificial_data_sys(N, n)

# Create data frames for analysis
create_dataframe <- function(sim_data) {
  data.frame(
    y = sim_data$sample_data,
    x = sim_data$x_matrix[, 2],  # Remove intercept column
    weights = sim_data$weights
  )
}

df_poi <- create_dataframe(data_poi)
df_est <- create_dataframe(data_est)
df_sys <- create_dataframe(data_sys)

# Sample sizes summary
sample_sizes <- data.frame(
  Design = c("Poisson", "Stratified", "Systematic"),
  Sample_Size = c(nrow(df_poi), nrow(df_est), nrow(df_sys))
)

knitr::kable(sample_sizes, caption = "Sample Sizes After Sampling", col.names = c("Design", "Sample Size
```

Table 5: Sample Sizes After Sampling

Design	Sample Size
Poisson	534
Stratified	500
Systematic	500

```r
# Function to fit models and extract coefficients
fit_and_compare <- function(data, design_name) {
  # Common MCMC settings (reduced for vignette)
  niter <- 5000
  burnin <- 2500

  # bayesQR
  fit_bqr <- bayesQR::bayesQR(
    y ~ x,
    data = data,
    quantile = quantiles,
    ndraw = niter - burnin,
    keep = 1
  )

  # tauBayesW - ALD method
  fit_ald <- bqr.svy(
    y ~ x,
    data = data,
    weights = data$weights,
    quantile = quantiles,
    method = "ald",
    niter = niter,
    burnin = burnin,
    print_progress = 5000
  )

  # tauBayesW - Score method
  fit_score <- bqr.svy(
    y ~ x,
    data = data,
    weights = data$weights,
    quantile = quantiles,
    method = "score",
    niter = niter,
    burnin = burnin,
    print_progress = 5000
  )

  # tauBayesW - Approximate method
  fit_approx <- bqr.svy(
    y ~ x,
    data = data,
    weights = data$weights,
    quantile = quantiles,
    method = "approximate",
    niter = niter,
    burnin = burnin,
    print_progress = 5000
  )

  # Extract coefficient estimates
```

```
  coef_bqr <- sapply(1:3, function(i) colMeans(fit_bqr[[i]]$betadraw))
  coef_ald <- fit_ald$beta
  coef_score <- fit_score$beta
  coef_approx <- fit_approx$beta

  # Calculate bias and RMSE
  calculate_metrics <- function(estimates, true_vals) {
    bias <- estimates - matrix(rep(true_vals, 3), nrow = 2, ncol = 3)
    rmse <- sqrt(colMeans(bias^2))
    list(bias = bias, rmse = rmse, estimates = estimates)
  }

  metrics_bqr <- calculate_metrics(coef_bqr, true_beta)
  metrics_ald <- calculate_metrics(coef_ald, true_beta)
  metrics_score <- calculate_metrics(coef_score, true_beta)
  metrics_approx <- calculate_metrics(coef_approx, true_beta)

  return(list(
    bayesQR = metrics_bqr,
    ALD = metrics_ald,
    Score = metrics_score,
    Approximate = metrics_approx
  ))
}

# Run comparison for each design
results_poi <- fit_and_compare(df_poi, "Poisson")
results_est <- fit_and_compare(df_est, "Stratified")
results_sys <- fit_and_compare(df_sys, "Systematic")
```

Method Comparison

```


# Poisson Sampling
for (q_idx in seq_along(quantiles)) {
  tau_val <- quantiles[q_idx]
  table_result <- create_results_table(
    results_poi, "Poisson", q_idx, tau_val
  )
  print(knitr::kable(
    table_result, digits = 4,
    caption = paste("Poisson Sampling =", tau_val)
  ))
}
```

Simulation Results

Table 6: Poisson Sampling = 0.25

Method	Intercept	Slope
True Values	2.0000	1.5000
bayesQR	1.9092	1.3008
ALD	1.6628	1.3651
Score	1.6760	1.2313
Approximate	1.6748	1.2321

Table 7: Poisson Sampling = 0.5

Method	Intercept	Slope
True Values	2.0000	1.5000
bayesQR	2.3707	1.4352
ALD	2.1408	1.4983
Score	2.1237	1.4820
Approximate	2.1234	1.4825

Table 8: Poisson Sampling = 0.75

Method	Intercept	Slope
True Values	2.0000	1.5000
bayesQR	3.0358	1.3903
ALD	2.6539	1.4852
Score	2.6986	1.4824
Approximate	2.6968	1.4838

```r
# Stratified Sampling
for (q_idx in seq_along(quantiles)) {
  tau_val <- quantiles[q_idx]
  table_result <- create_results_table(
    results_est, "Stratified", q_idx, tau_val
  )
  print(knitr::kable(
    table_result, digits = 4,
    caption = paste("Stratified Sampling =", tau_val)
  ))
}
```

Table 9: Stratified Sampling = 0.25

Method	Intercept	Slope
True Values	2.0000	1.5000
bayesQR	1.4433	1.5779
ALD	1.3217	1.5691
Score	1.2925	1.5599
Approximate	1.2793	1.5668

Table 10: Stratified Sampling = 0.5

Method	Intercept	Slope
True Values	2.0000	1.5000
bayesQR	2.2061	1.4838
ALD	1.8633	1.6120
Score	1.8448	1.6100
Approximate	1.8426	1.6111

Table 11: Stratified Sampling = 0.75

Method	Intercept	Slope
True Values	2.0000	1.5000
bayesQR	2.8812	1.4801
ALD	2.5164	1.5360
Score	2.5585	1.5515
Approximate	2.5563	1.5502

```r
# Systematic Sampling
for (q_idx in seq_along(quantiles)) {
  tau_val <- quantiles[q_idx]
  table_result <- create_results_table(
    results_sys, "Systematic", q_idx, tau_val
  )
  print(knitr::kable(
    table_result, digits = 4,
    caption = paste("Systematic Sampling =", tau_val)
  ))
}
```

Table 12: Systematic Sampling = 0.25

Method	Intercept	Slope
True Values	2.0000	1.5000
bayesQR	1.7098	1.4487
ALD	1.4035	1.5937
Score	1.3301	1.6013
Approximate	1.3515	1.5869

Table 13: Systematic Sampling = 0.5

Method	Intercept	Slope
True Values	2.0000	1.5000
bayesQR	2.3127	1.4671
ALD	2.1208	1.5018
Score	2.0686	1.5094
Approximate	2.0847	1.4998

Table 14: Systematic Sampling = 0.75

Method	Intercept	Slope
True Values	2.0000	1.5000
bayesQR	2.9191	1.4580
ALD	2.5350	1.5787
Score	2.5622	1.5866
Approximate	2.5398	1.6011

For the EM algorithm

```
library(pracma)
library(dplyr)
library(knitr)

#if (!exists("rbern")) rbern <- function(n, p) rbinom(n, 1, p)

make_mo_dataset <- function(design = c("poi","est","sys"),
                            N = 5000, n = 400, n_est = 4,
                            beta1 = c(2, 1.5),
                            beta2 = c(-1, 0.8),
                            sd2   = 1) {
  design <- match.arg(design)
  if (design == "poi") {
    s <- artificial_data_poi(N, n)
    X <- s$x_matrix
    y1 <- as.numeric(s$sample_data)
    w  <- as.numeric(s$weights)
  } else if (design == "est") {
    s <- artificial_data_est(N, n, n_est)
    X <- s$x_matrix
    y1 <- as.numeric(s$sample_data)
    w  <- as.numeric(s$weights)
  } else {
    s <- artificial_data_sys(N, n)
    X <- s$x_matrix
    y1 <- as.numeric(s$sample_data)
    w  <- as.numeric(s$weights)
  }

  y2 <- as.numeric(X %*% beta2 + rnorm(nrow(X), sd = sd2))

  df <- data.frame(
    y1 = y1,
    y2 = y2,
    x1 = X[, 2]
  )

  B <- cbind(beta1, beta2)

  list(data = df, w = w, X = X, B = B)
}
```

```r
mo_prior_default <- function(p,
                             beta_mean   = rep(0, p),
                             beta_cov    = diag(1e6, p),
                             sigma_shape = 0.001,
                             sigma_rate  = 0.001,
                             names       = NULL) {
  if (length(beta_mean) == 1L) beta_mean <- rep(beta_mean, p)
  if (is.numeric(beta_cov) && length(beta_cov) == 1L) beta_cov <- diag(beta_cov, p)
  if (is.numeric(beta_cov) && is.null(dim(beta_cov)) && length(beta_cov) == p)
    beta_cov <- diag(beta_cov, p)
  if (!is.null(names)) {
    names(beta_mean) <- names
    dimnames(beta_cov) <- list(names, names)
  }
  structure(list(
    beta_mean   = beta_mean,
    beta_cov    = beta_cov,
    sigma_shape = sigma_shape,
    sigma_rate  = sigma_rate
  ), class = "mo_bqr_prior")
}

run_mo_sim <- function(design = "poi",
                       N = 5000, n = 400, n_est = 4,
                       taus = c(0.25, 0.5, 0.75),
                       n_dir = 10,
                       max_iter = 500, verbose = FALSE,
                       keep_intercept = FALSE) {
  sim <- make_mo_dataset(design = design, N = N, n = n, n_est = n_est)
  df  <- sim$data
  w   <- sim$w
  B   <- sim$B
  d   <- 2
  p   <- 2

  set.seed(123)
  U <- matrix(NA_real_, d, n_dir)
  for (k in 1:n_dir) {
    uk <- rnorm(d); U[, k] <- uk / sqrt(sum(uk^2))
  }

  pr_mo <- mo_prior_default(p = p, beta_cov = diag(1e6, p))

  fit <- mo.bqr.svy(
    cbind(y1, y2) ~ x1,
    data     = df,
    weights  = w,
    quantile = taus,
    U        = U,
    max_iter = max_iter,
    verbose  = verbose,
    prior    = pr_mo
  )
```

```
  qlab <- paste0("q", taus)
  tau_pick <- "q0.5"
  stopifnot(tau_pick %in% qlab)

  comp <- do.call(rbind, lapply(1:n_dir, function(k) {
    Bu <- as.numeric(B %*% U[, k])
    beta_hat <- fit$fit[[tau_pick]]$directions[[k]]$beta[1:p]

    if (keep_intercept) {
      data.frame(
        dir      = k,
        u1       = round(U[1, k], 4),
        u2       = round(U[2, k], 4),
        true_Int = Bu[1],
        true_x1  = Bu[2],
        hat_Int  = beta_hat[1],
        hat_x1   = beta_hat[2],
        row.names = NULL
      )
    } else {
      data.frame(
        dir     = k,
        u1      = round(U[1, k], 4),
        u2      = round(U[2, k], 4),
        true_x1 = Bu[2],
        hat_x1  = beta_hat[2],
        row.names = NULL
      )
    }
  }))

  list(fit = fit, U = U, B = B, comparison_tau0.5 = comp)
}

out_poi2 <- run_mo_sim(design = "poi", N = 5000, n = 400, n_est = 4,
                       taus = c(0.25, 0.5, 0.75), n_dir = 10,
                       max_iter = 400, keep_intercept = TRUE)
```

```
kable(out_poi2$comparison_tau0.5,
      digits = 3,
      caption = "Directional comparison (tau = 0.5): true projected coefficients Bu vs. estimates from m
```

Table 15: Directional comparison (tau = 0.5): true projected coefficients Bu vs. estimates from mo.bqr.svy.

dir	u1	u2	true_Int	true_x1	hat_Int	hat_x1
1	-0.925	-0.380	-1.470	-1.691	-1.839	-1.340
2	0.999	0.045	1.953	1.535	2.237	1.330
3	0.075	0.997	-0.847	0.910	-0.472	0.876
4	0.342	-0.940	1.624	-0.238	1.367	-0.279
5	-0.839	-0.544	-1.133	-1.694	-1.441	-1.357

dir	u1	u2	true_Int	true_x1	hat_Int	hat_x1
6	0.959	0.282	1.637	1.665	2.050	1.353
7	0.964	0.266	1.662	1.659	2.046	1.361
8	-0.297	0.955	-1.549	0.318	-1.189	0.301
9	0.245	-0.969	1.460	-0.407	1.156	-0.365
10	0.829	-0.559	2.217	0.797	2.290	0.705

Prostate Dataset

These data come from a study that examined the correlation between the level of prostate specific antigen and a number of clinical measures in men who were about to receive a radical prostatectomy. It is a data frame with 97 rows and 9 columns and is loaded form the bayesQR package.

```
library(tauBayesW)
library(bayesQR)

# Load Prostate dataset
data(Prostate, package = "bayesQR")

# Center covariates (subtract mean, don't scale)
covars <- Prostate[, colnames(Prostate) != "lpsa"]
covars_centered <- as.data.frame(scale(covars, center = TRUE, scale = FALSE))

# Combined dataset with centered covariates
Prostate_centered <- cbind(lpsa = Prostate$lpsa, covars_centered)

# Design matrix (includes intercept automatically)
X <- model.matrix(lpsa ~ ., data = Prostate_centered)
y <- Prostate_centered$lpsa
w <- rep(1, length(y))  # Equal weights

# Display dataset summary
dataset_info <- data.frame(
  Information = c("Observations", "Coefficients", "Response Variable", "Equal Weights"),
  Value = c(nrow(Prostate_centered), ncol(X), "lpsa (log prostate-specific antigen)", "Yes")
)

knitr::kable(dataset_info, caption = "Prostate Dataset Summary")
```

Table 16: Prostate Dataset Summary

Information	Value
Observations	97
Coefficients	9
Response Variable	lpsa (log prostate-specific antigen)
Equal Weights	Yes

```
predictors_info <- data.frame(
  Predictor = colnames(X),
  Description = c("Intercept", paste("Centered", colnames(X)[-1]))
```

```
)

knitr::kable(predictors_info, caption = "Predictor Variables")
```

Table 17: Predictor Variables

Predictor	Description
(Intercept)	Intercept
lcavol	Centered lcavol
lweight	Centered lweight
age	Centered age
lbph	Centered lbph
svi	Centered svi
lcp	Centered lcp
gleason	Centered gleason
pgg45	Centered pgg45

```
# Prior for tauBayesW (all methods)
prior_tbw <- tauBayesW::prior(
  p = ncol(X),
  type = "MCMC",
  beta_mean = rep(0, ncol(X)),
  beta_cov = diag(1e6, ncol(X)),  # Very vague prior
  sigma_shape = 0.001,            # For ALD method
  sigma_rate = 0.001,
  names = colnames(X)
)

# Prior for bayesQR
prior_bqr <- bayesQR::prior(
  lpsa ~ ., data = Prostate_centered,
  beta0 = rep(0, ncol(X)),
  V0 = diag(1e6, ncol(X))  # Same vague prior
)
```

```
set.seed(12345)

fit_bqr <- bayesQR(
  lpsa ~ .,
  data = Prostate_centered,
  quantile = c(0.25, 0.5, 0.75),
  ndraw = 5000,
  prior = prior_bqr,
  keep = 5
)

taus <- c(0.25, 0.5, 0.75)

coef_mat <- do.call(rbind, lapply(seq_along(taus), function(i) {
```

```
  draws_i <- fit_bqr[[i]][["betadraw"]]
  colnames(draws_i) <- colnames(X)
  as.numeric(colMeans(draws_i, na.rm = TRUE))  # vector numérico
}))

rownames(coef_mat) <- paste0("tau = ", taus)
colnames(coef_mat) <- colnames(X)
```

bayesQR (Reference Method)

```
set.seed(12345)

fit_ald <- bqr.svy(
  lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
  data     = Prostate_centered,
  quantile = c(0.25, 0.5, 0.75),
  method   = "ald",
  prior    = prior_tbw,
  niter    = 50000,
  burnin   = 25000,
  thin     = 5,
  print_progress = 25000
)

coef_ald <- fit_ald$beta
```

tauBayesW Method 1: ALD (Asymmetric Laplace Distribution)

```
set.seed(12345)

fit_score <- bqr.svy(
  lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
  data     = Prostate_centered,
  quantile = c(0.25, 0.5, 0.75),
  method   = "score",
  prior    = prior_tbw,
  niter    = 50000,
  burnin   = 25000,
  thin     = 5,
  print_progress = 25000
)

coef_score <- fit_score$beta
```

tauBayesW Method 2: Score Likelihood

```r
set.seed(12345)

fit_approximate <- bqr.svy(
  lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
  data     = Prostate_centered,
  quantile = c(0.25, 0.5, 0.75),
  method   = "approximate",
  prior    = prior_tbw,
  niter    = 50000,
  burnin   = 25000,
  thin     = 5,
  print_progress = 25000
)

coef_approximate <- fit_approximate$beta
```

tauBayesW Method 3: Approximate Method

```r
make_results_table <- function(q_index, q_label) {
  coef_bqr   <- coef_mat[q_index, ]
  coef_ald   <- fit_ald[["beta"]][, q_index]
  coef_score <- fit_score[["beta"]][, q_index]
  coef_approx<- fit_approximate[["beta"]][, q_index]

  vars <- names(coef_bqr)

  results <- data.frame(
    Method = c("bayesQR", "tauBayesW - ALD", "tauBayesW - Score", "tauBayesW - Approximate"),
    t(sapply(list(coef_bqr, coef_ald, coef_score, coef_approx), function(coefs) coefs[vars]))
  )

  colnames(results) <- c("Method", vars)

  knitr::kable(results, digits = 3, caption = paste("Coefficient Estimates at quantile", q_label))
}
```

Results comparison

```r
make_results_table(1, 0.25)
```

25th quantile

Table 18: Coefficient Estimates at quantile 0.25

Method	(Intercept)	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45
bayesQR	1.927	0.629	0.604	-0.016	0.115	0.688	-0.212	-0.021	0.009
tauBayesW - ALD	2.111	0.695	0.683	-0.014	0.114	0.780	-0.237	-0.060	0.011
tauBayesW - Score	2.072	0.703	0.690	-0.015	0.117	0.695	-0.234	-0.018	0.011
tauBayesW - Approximate	2.077	0.692	0.708	-0.012	0.106	0.793	-0.231	-0.038	0.011

```
make_results_table(2, 0.50)
```

50th quantile

Table 19: Coefficient Estimates at quantile 0.5

Method	(Intercept)	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45
bayesQR	2.473	0.567	0.516	-0.022	0.134	0.806	-0.141	0.042	0.007
tauBayesW - ALD	2.432	0.525	0.554	-0.026	0.156	0.832	-0.116	0.208	0.003
tauBayesW - Score	2.438	0.528	0.541	-0.024	0.149	0.862	-0.113	0.231	0.002
tauBayesW - Approximate	2.441	0.524	0.534	-0.024	0.150	0.869	-0.109	0.231	0.002

```
make_results_table(3, 0.75)
```

75th quantile

Table 20: Coefficient Estimates at quantile 0.75

Method	(Intercept)	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45
bayesQR	3.033	0.549	0.398	-0.020	0.112	0.893	-0.041	0.065	0.003
tauBayesW - ALD	2.894	0.565	0.146	-0.013	0.111	0.864	-0.065	-0.109	0.007
tauBayesW - Score	2.943	0.479	0.478	-0.021	0.075	0.947	0.000	-0.059	0.004
tauBayesW - Approximate	2.931	0.563	0.115	-0.015	0.115	0.941	-0.057	-0.100	0.006