mediations.R

rachel97

Mon Nov 14 11:43:30 2016

```
knitr::opts_chunk$set(tidy = TRUE)
```

mediate()

```
mediation::mediate
```

```
## function (model.m, model.y, sims = 1000, boot = FALSE, boot.ci.type = "perc",
##     treat = "treat.name", mediator = "med.name", covariates = NULL,
##     outcome = NULL, control = NULL, conf.level = 0.95, control.value = 0,
##     treat.value = 1, long = TRUE, dropobs = FALSE, robustSE = FALSE,
##     cluster = NULL, group.out = NULL, ...)
## {
##     cl <- match.call()
##     if (match("INT", names(cl), 0L)) {
##         warning("'INT' is deprecated - existence of interaction terms is now automatically detected fro
##     }
##     if (robustSE && boot) {
##         warning("'robustSE' is ignored for nonparametric bootstrap")
##     }
##     if (!is.null(cluster) && boot) {
##         warning("'cluster' is ignored for nonparametric bootstrap")
##     }
##     if (robustSE & !is.null(cluster)) {
##         stop("choose either 'robustSE' or 'cluster' option, not both")
##     }
##     if (boot.ci.type != "bca" & boot.ci.type != "perc") {
##         stop("choose either 'bca' or 'perc' for boot.ci.type")
##     }
##     if (dropobs) {
##         odata.m <- model.frame(model.m)
##         odata.y <- model.frame(model.y)
##         if (!is.null(cluster)) {
##             if (is.null(row.names(cluster)) & (nrow(odata.m) !=
##                 length(cluster) | nrow(odata.y) != length(cluster))) {
##                 warning("cluster IDs may not correctly match original observations due to missing data"
##             }
##             odata.y <- merge(odata.y, as.data.frame(cluster),
##                 sort = FALSE, by = "row.names")
##             rownames(odata.y) <- odata.y$Row.names
```

```
##             odata.y <- odata.y[, -1L]
##         }
##         newdata <- merge(odata.m, odata.y, sort = FALSE, by = c("row.names",
##             intersect(names(odata.m), names(odata.y))))
##         rownames(newdata) <- newdata$Row.names
##         newdata <- newdata[, -1L]
##         rm(odata.m, odata.y)
##         call.m <- getCall(model.m)
##         call.y <- getCall(model.y)
##         call.m$data <- call.y$data <- newdata
##         if (c("(weights)") %in% names(newdata)) {
##             call.m$weights <- call.y$weights <- model.weights(newdata)
##         }
##         model.m <- eval.parent(call.m)
##         model.y <- eval.parent(call.y)
##         if (!is.null(cluster)) {
##             cluster <- factor(newdata[, ncol(newdata)])
##         }
##     }
##     isGam.y <- inherits(model.y, "gam")
##     isGam.m <- inherits(model.m, "gam")
##     isGlm.y <- inherits(model.y, "glm")
##     isGlm.m <- inherits(model.m, "glm")
##     isLm.y <- inherits(model.y, "lm")
##     isLm.m <- inherits(model.m, "lm")
##     isVglm.y <- inherits(model.y, "vglm")
##     isRq.y <- inherits(model.y, "rq")
##     isRq.m <- inherits(model.m, "rq")
##     isOrdered.y <- inherits(model.y, "polr")
##     isOrdered.m <- inherits(model.m, "polr")
##     isSurvreg.y <- inherits(model.y, "survreg")
##     isSurvreg.m <- inherits(model.m, "survreg")
##     isMer.y <- inherits(model.y, "merMod")
##     isMer.m <- inherits(model.m, "merMod")
##     if (isMer.m && getCall(model.m)[[1]] == "glmer") {
##         m.family <- as.character(model.m@call$family)
##         if (m.family[1] == "binomial" && (is.na(m.family[2]) ||
##             m.family[2] == "logit")) {
##             M.fun <- binomial(link = "logit")
##         }
##         else if (m.family[1] == "binomial" && m.family[2] ==
##             "probit") {
##             M.fun <- binomial(link = "probit")
##         }
```

```
##         else if (m.family[1] == "binomial" && m.family[2] ==
##             "cloglog") {
##             M.fun <- binomial(link = "cloglog")
##         }
##         else if (m.family[1] == "poisson" && (is.na(m.family[2]) ||
##             m.family[2] == "log")) {
##             M.fun <- poisson(link = "log")
##         }
##         else if (m.family[1] == "poisson" && m.family[2] == "identity") {
##             M.fun <- poisson(link = "identity")
##         }
##         else if (m.family[1] == "poisson" && m.family[2] == "sqrt") {
##             M.fun <- poisson(link = "sqrt")
##         }
##         else {
##             stop("glmer family for the mediation model not supported")
##         }
##     }
##     if (isMer.y && getCall(model.y)[[1]] == "glmer") {
##         y.family <- as.character(model.y@call$family)
##         if (y.family[1] == "binomial" && (is.na(y.family[2]) ||
##             y.family[2] == "logit")) {
##             Y.fun <- binomial(link = "logit")
##         }
##         else if (y.family[1] == "binomial" && y.family[2] ==
##             "probit") {
##             Y.fun <- binomial(link = "probit")
##         }
##         else if (y.family[1] == "binomial" && y.family[2] ==
##             "cloglog") {
##             Y.fun <- binomial(link = "cloglog")
##         }
##         else if (y.family[1] == "poisson" && (is.na(y.family[2]) ||
##             y.family[2] == "log")) {
##             Y.fun <- poisson(link = "log")
##         }
##         else if (y.family[1] == "poisson" && y.family[2] == "identity") {
##             Y.fun <- poisson(link = "identity")
##         }
##         else if (y.family[1] == "poisson" && y.family[2] == "sqrt") {
##             Y.fun <- poisson(link = "sqrt")
##         }
##         else {
##             stop("glmer family for the outcome model not supported")
```

```
##         }
##     }
##     if (isGlm.m) {
##         FamilyM <- model.m$family$family
##     }
##     if (isMer.m && getCall(model.m)[[1]] == "glmer") {
##         FamilyM <- M.fun$family
##     }
##     if (isVglm.y) {
##         VfamilyY <- model.y@family@vfamily
##     }
##     if (!is.null(control) && !isGam.y) {
##         warning("'control' is only used for GAM outcome models - ignored")
##         control <- NULL
##     }
##     if (!is.null(outcome) && !(isSurvreg.y && boot)) {
##         warning("'outcome' is only relevant for survival outcome models with bootstrap - ignored")
##     }
##     m.data <- model.frame(model.m)
##     y.data <- model.frame(model.y)
##     if (!is.null(cluster)) {
##         row.names(m.data) <- 1:nrow(m.data)
##         row.names(y.data) <- 1:nrow(y.data)
##         if (!is.null(model.m$weights)) {
##             m.weights <- as.data.frame(model.m$weights)
##             m.name <- as.character(model.m$call$weights)
##             names(m.weights) <- m.name
##             m.data <- cbind(m.data, m.weights)
##         }
##         if (!is.null(model.y$weights)) {
##             y.weights <- as.data.frame(model.y$weights)
##             y.name <- as.character(model.y$call$weights)
##             names(y.weights) <- y.name
##             y.data <- cbind(y.data, y.weights)
##         }
##     }
##     if (isMer.y & !isMer.m) {
##         m.data <- eval(model.m$call$data, environment(formula(model.m)))
##         m.data <- na.omit(m.data)
##         y.data <- na.omit(y.data)
##     }
##     if (isMer.m && isMer.y) {
##         med.group <- names(model.m@flist)
##         out.group <- names(model.y@flist)
```

```
##         n.med <- length(med.group)
##         n.out <- length(out.group)
##         if (n.med > 1 || n.out > 1) {
##             stop("mediate does not support more than two levels per model")
##         }
##         else {
##             group.m <- med.group
##             group.y <- out.group
##             if (!is.null(group.out) && !(group.out %in% c(group.m,
##                 group.y))) {
##                 warning("group.out does not match group names used in merMod")
##             }
##             else if (is.null(group.out)) {
##                 group.out <- group.y
##             }
##         }
##     }
##     else if (!isMer.m && isMer.y) {
##         out.group <- names(model.y@flist)
##         n.out <- length(out.group)
##         if (n.out > 1) {
##             stop("mediate does not support more than two levels per model")
##         }
##         else {
##             group.m <- NULL
##             group.y <- out.group
##             group.out <- group.y
##         }
##     }
##     else if (isMer.m && !isMer.y) {
##         med.group <- names(model.m@flist)
##         n.med <- length(med.group)
##         if (n.med > 1) {
##             stop("mediate does not support more than two levels per model")
##         }
##         else {
##             group.m <- med.group
##             group.y <- NULL
##             group.out <- group.m
##         }
##     }
##     else {
##         group.m <- NULL
##         group.y <- NULL
```

```
##         group.out <- NULL
##     }
##     if (isMer.m) {
##         group.id.m <- m.data[, group.m]
##     }
##     else {
##         group.id.m <- NULL
##     }
##     if (isMer.y) {
##         group.id.y <- y.data[, group.y]
##     }
##     else {
##         group.id.y <- NULL
##     }
##     if (isMer.y && isMer.m) {
##         if (group.out == group.m) {
##             group.id <- m.data[, group.m]
##             group.name <- group.m
##         }
##         else {
##             group.id <- y.data[, group.y]
##             group.name <- group.y
##         }
##     }
##     else if (!isMer.y && isMer.m) {
##         group.id <- m.data[, group.m]
##         group.name <- group.m
##     }
##     else if (isMer.y && !isMer.m) {
##         if (!(group.y %in% names(m.data))) {
##             stop("specify group-level variable in mediator data")
##         }
##         else {
##             group.id <- y.data[, group.y]
##             group.name <- group.y
##             Y.ID <- sort(unique(group.id))
##             M.ID <- sort(as.vector(data.matrix(m.data[group.y])))
##             if (length(Y.ID) != length(M.ID)) {
##                 stop("groups do not match between mediator and outcome models")
##             }
##             else {
##                 if (FALSE %in% unique(Y.ID == M.ID)) {
##                   stop("groups do not match between mediator and outcome models")
##                 }
```

```
##             }
##         }
##     }
##     else {
##         group.id <- NULL
##         group.name <- NULL
##     }
##     n.m <- nrow(m.data)
##     n.y <- nrow(y.data)
##     if (!(isMer.y & !isMer.m)) {
##         if (n.m != n.y) {
##             stop("number of observations do not match between mediator and outcome models")
##         }
##         else {
##             n <- n.m
##         }
##         m <- length(sort(unique(model.frame(model.m)[, 1])))
##     }
##     weights.m <- model.weights(m.data)
##     weights.y <- model.weights(y.data)
##     if (!is.null(weights.m) && isGlm.m && FamilyM == "binomial") {
##         message("weights taken as sampling weights, not total number of trials")
##     }
##     if (!is.null(weights.m) && isMer.m && getCall(model.m)[[1]] ==
##         "glmer" && FamilyM == "binomial") {
##         message("weights taken as sampling weights, not total number of trials")
##     }
##     if (is.null(weights.m)) {
##         weights.m <- rep(1, nrow(m.data))
##     }
##     if (is.null(weights.y)) {
##         weights.y <- rep(1, nrow(y.data))
##     }
##     if (!(isMer.y & !isMer.m)) {
##         if (!all(weights.m == weights.y)) {
##             stop("weights on outcome and mediator models not identical")
##         }
##         else {
##             weights <- weights.m
##         }
##     }
##     else {
##         weights <- weights.y
##     }
```

```
##     if (is.character(m.data[, treat])) {
##         m.data[, treat] <- factor(m.data[, treat])
##     }
##     if (is.character(y.data[, treat])) {
##         y.data[, treat] <- factor(y.data[, treat])
##     }
##     if (is.character(y.data[, mediator])) {
##         y.data[, mediator] <- factor(y.data[, mediator])
##     }
##     isFactorT.m <- is.factor(m.data[, treat])
##     isFactorT.y <- is.factor(y.data[, treat])
##     if (isFactorT.m != isFactorT.y) {
##         stop("treatment variable types differ in mediator and outcome models")
##     }
##     else {
##         isFactorT <- isFactorT.y
##     }
##     if (isFactorT) {
##         t.levels <- levels(y.data[, treat])
##         if (treat.value %in% t.levels & control.value %in% t.levels) {
##             cat.0 <- control.value
##             cat.1 <- treat.value
##         }
##         else {
##             cat.0 <- t.levels[1]
##             cat.1 <- t.levels[2]
##             warning("treatment and control values do not match factor levels; using ",
##                 cat.0, " and ", cat.1, " as control and treatment, respectively")
##         }
##     }
##     else {
##         cat.0 <- control.value
##         cat.1 <- treat.value
##     }
##     isFactorM <- is.factor(y.data[, mediator])
##     if (isFactorM) {
##         m.levels <- levels(y.data[, mediator])
##     }
##     indexmax <- function(x) {
##         order(x)[length(x)]
##     }
##     getvcov <- function(dat, fm, cluster) {
##         cluster <- factor(cluster)
##         M <- nlevels(cluster)
```

```
##         N <- sum(!is.na(cluster))
##         K <- fm$rank
##         dfc <- (M/(M - 1)) * ((N - 1)/(N - K))
##         uj <- apply(estfun(fm), 2, function(x) tapply(x, cluster,
##             sum))
##         dfc * sandwich(fm, meat. = crossprod(uj)/N)
##     }
##     if (!isOrdered.y) {
##         if (!boot) {
##             if (isGam.m | isGam.y | isRq.m) {
##                 stop("'boot' must be 'TRUE' for models used")
##             }
##             if (isSurvreg.m && is.null(survival::survreg.distributions[[model.m$dist]]$scale)) {
##                 MModel.coef <- c(coef(model.m), log(model.m$scale))
##                 scalesim.m <- TRUE
##             }
##             else if (isMer.m) {
##                 MModel.fixef <- lme4::fixef(model.m)
##                 MModel.ranef <- lme4::ranef(model.m)
##                 scalesim.m <- FALSE
##             }
##             else {
##                 MModel.coef <- coef(model.m)
##                 scalesim.m <- FALSE
##             }
##             if (isOrdered.m) {
##                 if (is.null(model.m$Hess)) {
##                   cat("Mediator model object does not contain 'Hessian';")
##                 }
##                 k <- length(MModel.coef)
##                 MModel.var.cov <- vcov(model.m)[(1:k), (1:k)]
##             }
##             else if (isSurvreg.m) {
##                 MModel.var.cov <- vcov(model.m)
##             }
##             else {
##                 if (robustSE & !isMer.m) {
##                   MModel.var.cov <- vcovHC(model.m, ...)
##                 }
##                 else if (robustSE & isMer.m) {
##                   MModel.var.cov <- vcov(model.m)
##                   warning("robustSE does not support mer class: non-robust SEs are computed for model.m
##                 }
##                 else if (!is.null(cluster)) {
```

```
##                   if (nrow(m.data) != length(cluster)) {
##                     warning("length of cluster vector differs from # of obs for mediator model")
##                   }
##                   dta <- merge(m.data, as.data.frame(cluster),
##                     sort = FALSE, by = "row.names")
##                   fm <- update(model.m, data = dta)
##                   MModel.var.cov <- getvcov(dta, fm, dta[, ncol(dta)])
##                 }
##                 else {
##                   MModel.var.cov <- vcov(model.m)
##                 }
##             }
##             if (isSurvreg.y && is.null(survival::survreg.distributions[[model.y$dist]]$scale)) {
##                 YModel.coef <- c(coef(model.y), log(model.y$scale))
##                 scalesim.y <- TRUE
##             }
##             else if (isMer.y) {
##                 YModel.fixef <- lme4::fixef(model.y)
##                 YModel.ranef <- lme4::ranef(model.y)
##                 scalesim.y <- FALSE
##             }
##             else {
##                 YModel.coef <- coef(model.y)
##                 scalesim.y <- FALSE
##             }
##             if (isRq.y) {
##                 YModel.var.cov <- summary(model.y, covariance = TRUE)$cov
##             }
##             else if (isSurvreg.y) {
##                 YModel.var.cov <- vcov(model.y)
##             }
##             else {
##                 if (robustSE & !isMer.y) {
##                   YModel.var.cov <- vcovHC(model.y, ...)
##                 }
##                 else if (robustSE & isMer.y) {
##                   YModel.var.cov <- vcov(model.y)
##                   warning("robustSE does not support mer class: non-robust SEs are computed for model.y
##                 }
##                 else if (!is.null(cluster)) {
##                   if (nrow(y.data) != length(cluster)) {
##                     warning("length of cluster vector differs from # of obs for outcome model")
##                   }
##                   dta <- merge(y.data, as.data.frame(cluster),
```

```
##                     sort = FALSE, by = "row.names")
##                   fm <- update(model.y, data = dta)
##                   YModel.var.cov <- getvcov(dta, fm, dta[, ncol(dta)])
##                 }
##                 else {
##                   YModel.var.cov <- vcov(model.y)
##                 }
##             }
##             se.ranef.new <- function(object) {
##                 se.bygroup <- lme4::ranef(object, condVar = TRUE)
##                 n.groupings <- length(se.bygroup)
##                 for (m in 1:n.groupings) {
##                   vars.m <- attr(se.bygroup[[m]], "postVar")
##                   K <- dim(vars.m)[1]
##                   J <- dim(vars.m)[3]
##                   names.full <- dimnames(se.bygroup[[m]])
##                   se.bygroup[[m]] <- array(NA, c(J, K))
##                   for (j in 1:J) {
##                     se.bygroup[[m]][j, ] <- sqrt(diag(as.matrix(vars.m[,
##                       , j])))
##                   }
##                   dimnames(se.bygroup[[m]]) <- list(names.full[[1]],
##                     names.full[[2]])
##                 }
##                 return(se.bygroup)
##             }
##             if (isMer.m) {
##                 MModel.fixef.vcov <- as.matrix(vcov(model.m))
##                 MModel.fixef.sim <- rmvnorm(sims, mean = MModel.fixef,
##                   sigma = MModel.fixef.vcov)
##                 Nm.ranef <- ncol(lme4::ranef(model.m)[[1]])
##                 MModel.ranef.sim <- vector("list", Nm.ranef)
##                 for (d in 1:Nm.ranef) {
##                   MModel.ranef.sim[[d]] <- matrix(rnorm(sims *
##                     nrow(lme4::ranef(model.m)[[1]]), mean = lme4::ranef(model.m)[[1]][,
##                     d], sd = se.ranef.new(model.m)[[1]][, d]),
##                     nrow = sims, byrow = TRUE)
##                 }
##             }
##             else {
##                 if (sum(is.na(MModel.coef)) > 0) {
##                   stop("NA in model coefficients; rerun models with nonsingular design matrix")
##                 }
##                 MModel <- rmvnorm(sims, mean = MModel.coef, sigma = MModel.var.cov)
```

```
##             }
##             if (isMer.y) {
##                 YModel.fixef.vcov <- as.matrix(vcov(model.y))
##                 YModel.fixef.sim <- rmvnorm(sims, mean = YModel.fixef,
##                   sigma = YModel.fixef.vcov)
##                 Ny.ranef <- ncol(lme4::ranef(model.y)[[1]])
##                 YModel.ranef.sim <- vector("list", Ny.ranef)
##                 for (d in 1:Ny.ranef) {
##                   YModel.ranef.sim[[d]] <- matrix(rnorm(sims *
##                     nrow(lme4::ranef(model.y)[[1]]), mean = lme4::ranef(model.y)[[1]][,
##                     d], sd = se.ranef.new(model.y)[[1]][, d]),
##                     nrow = sims, byrow = TRUE)
##                 }
##             }
##             else {
##                 if (sum(is.na(YModel.coef)) > 0) {
##                   stop("NA in model coefficients; rerun models with nonsingular design matrix")
##                 }
##                 YModel <- rmvnorm(sims, mean = YModel.coef, sigma = YModel.var.cov)
##             }
##             if (robustSE && (isSurvreg.m | isSurvreg.y)) {
##                 warning("‘robustSE’ ignored for survival models; fit the model with ‘robust’ option ins
##             }
##             if (!is.null(cluster) && (isSurvreg.m | isSurvreg.y)) {
##                 warning("‘cluster’ ignored for survival models; fit the model with ’cluster()’ term in
##             }
##             if (isMer.y & !isMer.m) {
##                 n <- n.m
##             }
##             pred.data.t <- pred.data.c <- m.data
##             if (isFactorT) {
##                 pred.data.t[, treat] <- factor(cat.1, levels = t.levels)
##                 pred.data.c[, treat] <- factor(cat.0, levels = t.levels)
##             }
##             else {
##                 pred.data.t[, treat] <- cat.1
##                 pred.data.c[, treat] <- cat.0
##             }
##             if (!is.null(covariates)) {
##                 for (p in 1:length(covariates)) {
##                   vl <- names(covariates[p])
##                   if (is.factor(pred.data.t[, vl])) {
##                     pred.data.t[, vl] <- pred.data.c[, vl] <- factor(covariates[[p]],
##                       levels = levels(m.data[, vl]))
```

```
##                   }
##                   else {
##                     pred.data.t[, vl] <- pred.data.c[, vl] <- covariates[[p]]
##                   }
##                 }
##             }
##             mmat.t <- model.matrix(terms(model.m), data = pred.data.t)
##             mmat.c <- model.matrix(terms(model.m), data = pred.data.c)
##             if (isGlm.m) {
##                 muM1 <- model.m$family$linkinv(tcrossprod(MModel,
##                   mmat.t))
##                 muM0 <- model.m$family$linkinv(tcrossprod(MModel,
##                   mmat.c))
##                 if (FamilyM == "poisson") {
##                   PredictM1 <- matrix(rpois(sims * n, lambda = muM1),
##                     nrow = sims)
##                   PredictM0 <- matrix(rpois(sims * n, lambda = muM0),
##                     nrow = sims)
##                 }
##                 else if (FamilyM == "Gamma") {
##                   shape <- gamma.shape(model.m)$alpha
##                   PredictM1 <- matrix(rgamma(n * sims, shape = shape,
##                     scale = muM1/shape), nrow = sims)
##                   PredictM0 <- matrix(rgamma(n * sims, shape = shape,
##                     scale = muM0/shape), nrow = sims)
##                 }
##                 else if (FamilyM == "binomial") {
##                   PredictM1 <- matrix(rbinom(n * sims, size = 1,
##                     prob = muM1), nrow = sims)
##                   PredictM0 <- matrix(rbinom(n * sims, size = 1,
##                     prob = muM0), nrow = sims)
##                 }
##                 else if (FamilyM == "gaussian") {
##                   sigma <- sqrt(summary(model.m)$dispersion)
##                   error <- rnorm(sims * n, mean = 0, sd = sigma)
##                   PredictM1 <- muM1 + matrix(error, nrow = sims)
##                   PredictM0 <- muM0 + matrix(error, nrow = sims)
##                 }
##                 else if (FamilyM == "inverse.gaussian") {
##                   disp <- summary(model.m)$dispersion
##                   PredictM1 <- matrix(SuppDists::rinvGauss(n *
##                     sims, nu = muM1, lambda = 1/disp), nrow = sims)
##                   PredictM0 <- matrix(SuppDists::rinvGauss(n *
##                     sims, nu = muM0, lambda = 1/disp), nrow = sims)
```

```
##                 }
##                 else {
##                   stop("unsupported glm family")
##                 }
##             }
##             else if (isOrdered.m) {
##                 if (model.m$method == "logistic") {
##                   linkfn <- plogis
##                 }
##                 else if (model.m$method == "probit") {
##                   linkfn <- pnorm
##                 }
##                 else {
##                   stop("unsupported polr method; use 'logistic' or 'probit'")
##                 }
##                 m.cat <- sort(unique(model.frame(model.m)[, 1]))
##                 lambda <- model.m$zeta
##                 mmat.t <- mmat.t[, -1]
##                 mmat.c <- mmat.c[, -1]
##                 ystar_m1 <- tcrossprod(MModel, mmat.t)
##                 ystar_m0 <- tcrossprod(MModel, mmat.c)
##                 PredictM1 <- matrix(, nrow = sims, ncol = n)
##                 PredictM0 <- matrix(, nrow = sims, ncol = n)
##                 for (i in 1:sims) {
##                   cprobs_m1 <- matrix(NA, n, m)
##                   cprobs_m0 <- matrix(NA, n, m)
##                   probs_m1 <- matrix(NA, n, m)
##                   probs_m0 <- matrix(NA, n, m)
##                   for (j in 1:(m - 1)) {
##                     cprobs_m1[, j] <- linkfn(lambda[j] - ystar_m1[i,
##                       ])
##                     cprobs_m0[, j] <- linkfn(lambda[j] - ystar_m0[i,
##                       ])
##                     probs_m1[, m] <- 1 - cprobs_m1[, m - 1]
##                     probs_m0[, m] <- 1 - cprobs_m0[, m - 1]
##                     probs_m1[, 1] <- cprobs_m1[, 1]
##                     probs_m0[, 1] <- cprobs_m0[, 1]
##                   }
##                   for (j in 2:(m - 1)) {
##                     probs_m1[, j] <- cprobs_m1[, j] - cprobs_m1[,
##                       j - 1]
##                     probs_m0[, j] <- cprobs_m0[, j] - cprobs_m0[,
##                       j - 1]
##                   }
```

```
##                   draws_m1 <- matrix(NA, n, m)
##                   draws_m0 <- matrix(NA, n, m)
##                   for (ii in 1:n) {
##                     draws_m1[ii, ] <- t(rmultinom(1, 1, prob = probs_m1[ii,
##                       ]))
##                     draws_m0[ii, ] <- t(rmultinom(1, 1, prob = probs_m0[ii,
##                       ]))
##                   }
##                   PredictM1[i, ] <- apply(draws_m1, 1, indexmax)
##                   PredictM0[i, ] <- apply(draws_m0, 1, indexmax)
##                 }
##             }
##             else if (isLm.m) {
##                 sigma <- summary(model.m)$sigma
##                 error <- rnorm(sims * n, mean = 0, sd = sigma)
##                 muM1 <- tcrossprod(MModel, mmat.t)
##                 muM0 <- tcrossprod(MModel, mmat.c)
##                 PredictM1 <- muM1 + matrix(error, nrow = sims)
##                 PredictM0 <- muM0 + matrix(error, nrow = sims)
##                 rm(error)
##             }
##             else if (isSurvreg.m) {
##                 dd <- survival::survreg.distributions[[model.m$dist]]
##                 if (is.null(dd$itrans)) {
##                   itrans <- function(x) x
##                 }
##                 else {
##                   itrans <- dd$itrans
##                 }
##                 dname <- dd$dist
##                 if (is.null(dname)) {
##                   dname <- model.m$dist
##                 }
##                 if (scalesim.m) {
##                   scale <- exp(MModel[, ncol(MModel)])
##                   lpM1 <- tcrossprod(MModel[, 1:(ncol(MModel) -
##                     1)], mmat.t)
##                   lpM0 <- tcrossprod(MModel[, 1:(ncol(MModel) -
##                     1)], mmat.c)
##                 }
##                 else {
##                   scale <- dd$scale
##                   lpM1 <- tcrossprod(MModel, mmat.t)
##                   lpM0 <- tcrossprod(MModel, mmat.c)
```

```
##                 }
##                 error <- switch(dname, extreme = log(rweibull(sims *
##                   n, shape = 1, scale = 1)), gaussian = rnorm(sims *
##                   n), logistic = rlogis(sims * n), t = rt(sims *
##                   n, df = dd$parms))
##                 PredictM1 <- itrans(lpM1 + scale * matrix(error,
##                   nrow = sims))
##                 PredictM0 <- itrans(lpM0 + scale * matrix(error,
##                   nrow = sims))
##                 rm(error)
##             }
##             else if (isMer.m && getCall(model.m)[[1]] == "lmer") {
##                 M.RANEF1 <- M.RANEF0 <- 0
##                 for (d in 1:Nm.ranef) {
##                   name <- colnames(lme4::ranef(model.m)[[1]])[d]
##                   if (name == "(Intercept)") {
##                     var1 <- var0 <- matrix(1, sims, n)
##                   }
##                   else if (name == treat) {
##                     var1 <- matrix(1, sims, n)
##                     var0 <- matrix(0, sims, n)
##                   }
##                   else {
##                     var1 <- var0 <- matrix(data.matrix(m.data[name]),
##                       sims, n, byrow = T)
##                   }
##                   M.ranef <- matrix(NA, sims, n)
##                   MModel.ranef.sim.d <- MModel.ranef.sim[[d]]
##                   Z <- data.frame(MModel.ranef.sim.d)
##                   if (is.factor(group.id.m)) {
##                     colnames(Z) <- levels(group.id.m)
##                     for (i in 1:n) {
##                       M.ranef[, i] <- Z[, group.id.m[i] == levels(group.id.m)]
##                     }
##                   }
##                   else {
##                     colnames(Z) <- sort(unique(group.id.m))
##                     for (i in 1:n) {
##                       M.ranef[, i] <- Z[, group.id.m[i] == sort(unique(group.id.m))]
##                     }
##                   }
##                   M.RANEF1 <- M.ranef * var1 + M.RANEF1
##                   M.RANEF0 <- M.ranef * var0 + M.RANEF0
##                 }
```

```
##                 sigma <- attr(lme4::VarCorr(model.m), "sc")
##                 error <- rnorm(sims * n, mean = 0, sd = sigma)
##                 muM1 <- tcrossprod(MModel.fixef.sim, mmat.t) +
##                   M.RANEF1
##                 muM0 <- tcrossprod(MModel.fixef.sim, mmat.c) +
##                   M.RANEF0
##                 PredictM1 <- muM1 + matrix(error, nrow = sims)
##                 PredictM0 <- muM0 + matrix(error, nrow = sims)
##                 rm(error)
##             }
##             else if (isMer.m && getCall(model.m)[[1]] == "glmer") {
##                 M.RANEF1 <- M.RANEF0 <- 0
##                 for (d in 1:Nm.ranef) {
##                   name <- colnames(lme4::ranef(model.m)[[1]])[d]
##                   if (name == "(Intercept)") {
##                     var1 <- var0 <- matrix(1, sims, n)
##                   }
##                   else if (name == treat) {
##                     var1 <- matrix(1, sims, n)
##                     var0 <- matrix(0, sims, n)
##                   }
##                   else {
##                     var1 <- var0 <- matrix(data.matrix(m.data[name]),
##                       sims, n, byrow = T)
##                   }
##                   M.ranef <- matrix(NA, sims, n)
##                   MModel.ranef.sim.d <- MModel.ranef.sim[[d]]
##                   Z <- data.frame(MModel.ranef.sim.d)
##                   if (is.factor(group.id.m)) {
##                     colnames(Z) <- levels(group.id.m)
##                     for (i in 1:n) {
##                       M.ranef[, i] <- Z[, group.id.m[i] == levels(group.id.m)]
##                     }
##                   }
##                   else {
##                     colnames(Z) <- sort(unique(group.id.m))
##                     for (i in 1:n) {
##                       M.ranef[, i] <- Z[, group.id.m[i] == sort(unique(group.id.m))]
##                     }
##                   }
##                   M.RANEF1 <- M.ranef * var1 + M.RANEF1
##                   M.RANEF0 <- M.ranef * var0 + M.RANEF0
##                 }
##                 muM1 <- M.fun$linkinv(tcrossprod(MModel.fixef.sim,
```

```
##                   mmat.t) + M.RANEF1)
##                 muM0 <- M.fun$linkinv(tcrossprod(MModel.fixef.sim,
##                   mmat.c) + M.RANEF0)
##                 FamilyM <- M.fun$family
##                 if (FamilyM == "poisson") {
##                   PredictM1 <- matrix(rpois(sims * n, lambda = muM1),
##                     nrow = sims)
##                   PredictM0 <- matrix(rpois(sims * n, lambda = muM0),
##                     nrow = sims)
##                 }
##                 else if (FamilyM == "binomial") {
##                   PredictM1 <- matrix(rbinom(n * sims, size = 1,
##                     prob = muM1), nrow = sims)
##                   PredictM0 <- matrix(rbinom(n * sims, size = 1,
##                     prob = muM0), nrow = sims)
##                 }
##             }
##             else {
##                 stop("mediator model is not yet implemented")
##             }
##             if (isMer.y & !isMer.m) {
##                 J <- nrow(m.data)
##                 group.id.m <- as.vector(data.matrix(m.data[group.y]))
##                 v1 <- v0 <- matrix(NA, sims, length(group.id.y))
##                 num.m <- 1:J
##                 num.y <- 1:length(group.id.y)
##                 for (j in 1:J) {
##                   id.y <- unique(group.id.y)[j]
##                   NUM.M <- num.m[group.id.m == id.y]
##                   NUM.Y <- num.y[group.id.y == id.y]
##                   v1[, NUM.Y] <- PredictM1[, NUM.M]
##                   v0[, NUM.Y] <- PredictM0[, NUM.M]
##                 }
##                 PredictM1 <- v1
##                 PredictM0 <- v0
##             }
##             rm(mmat.t, mmat.c)
##             if (isMer.y & !isMer.m) {
##                 n <- n.y
##             }
##             effects.tmp <- array(NA, dim = c(n, sims, 4))
##             if (isMer.y) {
##                 Y.RANEF1 <- Y.RANEF2 <- Y.RANEF3 <- Y.RANEF4 <- 0
##                 for (d in 1:Ny.ranef) {
```

```
##                   name <- colnames(lme4::ranef(model.y)[[1]])[d]
##                   if (name == "(Intercept)") {
##                     var1 <- var2 <- var3 <- var4 <- matrix(1,
##                       sims, n)
##                   }
##                   else if (name == treat) {
##                     var1 <- matrix(1, sims, n)
##                     var2 <- matrix(1, sims, n)
##                     var3 <- matrix(0, sims, n)
##                     var4 <- matrix(0, sims, n)
##                   }
##                   else if (name == mediator) {
##                     var1 <- PredictM1
##                     var2 <- PredictM0
##                     var3 <- PredictM1
##                     var4 <- PredictM0
##                   }
##                   else {
##                     var1 <- var2 <- var3 <- var4 <- matrix(data.matrix(y.data[name]),
##                       sims, n, byrow = T)
##                   }
##                   Y.ranef <- matrix(NA, sims, n)
##                   YModel.ranef.sim.d <- YModel.ranef.sim[[d]]
##                   Z <- data.frame(YModel.ranef.sim.d)
##                   if (is.factor(group.id.y)) {
##                     colnames(Z) <- levels(group.id.y)
##                     for (i in 1:n) {
##                       Y.ranef[, i] <- Z[, group.id.y[i] == levels(group.id.y)]
##                     }
##                   }
##                   else {
##                     colnames(Z) <- sort(unique(group.id.y))
##                     for (i in 1:n) {
##                       Y.ranef[, i] <- Z[, group.id.y[i] == sort(unique(group.id.y))]
##                     }
##                   }
##                   Y.RANEF1 <- Y.ranef * var1 + Y.RANEF1
##                   Y.RANEF2 <- Y.ranef * var2 + Y.RANEF2
##                   Y.RANEF3 <- Y.ranef * var3 + Y.RANEF3
##                   Y.RANEF4 <- Y.ranef * var4 + Y.RANEF4
##                 }
##               }
##               for (e in 1:4) {
##                   tt <- switch(e, c(1, 1, 1, 0), c(0, 0, 1, 0),
```

```
##                   c(1, 0, 1, 1), c(1, 0, 0, 0))
##                 Pr1 <- matrix(, nrow = n, ncol = sims)
##                 Pr0 <- matrix(, nrow = n, ncol = sims)
##                 for (j in 1:sims) {
##                   pred.data.t <- pred.data.c <- y.data
##                   if (!is.null(covariates)) {
##                     for (p in 1:length(covariates)) {
##                       vl <- names(covariates[p])
##                       if (is.factor(pred.data.t[, vl])) {
##                         pred.data.t[, vl] <- pred.data.c[, vl] <- factor(covariates[[p]],
##                           levels = levels(y.data[, vl]))
##                       }
##                       else {
##                         pred.data.t[, vl] <- pred.data.c[, vl] <- covariates[[p]]
##                       }
##                     }
##                   }
##                   cat.t <- ifelse(tt[1], cat.1, cat.0)
##                   cat.c <- ifelse(tt[2], cat.1, cat.0)
##                   cat.t.ctrl <- ifelse(tt[1], cat.0, cat.1)
##                   cat.c.ctrl <- ifelse(tt[2], cat.0, cat.1)
##                   if (isFactorT) {
##                     pred.data.t[, treat] <- factor(cat.t, levels = t.levels)
##                     pred.data.c[, treat] <- factor(cat.c, levels = t.levels)
##                     if (!is.null(control)) {
##                       pred.data.t[, control] <- factor(cat.t.ctrl,
##                         levels = t.levels)
##                       pred.data.c[, control] <- factor(cat.c.ctrl,
##                         levels = t.levels)
##                     }
##                   }
##                   else {
##                     pred.data.t[, treat] <- cat.t
##                     pred.data.c[, treat] <- cat.c
##                     if (!is.null(control)) {
##                       pred.data.t[, control] <- cat.t.ctrl
##                       pred.data.c[, control] <- cat.c.ctrl
##                     }
##                   }
##                   PredictMt <- PredictM1[j, ] * tt[3] + PredictM0[j,
##                     ] * (1 - tt[3])
##                   PredictMc <- PredictM1[j, ] * tt[4] + PredictM0[j,
##                     ] * (1 - tt[4])
##                   if (isFactorM) {
```

```
##                   pred.data.t[, mediator] <- factor(PredictMt,
##                     levels = 1:m, labels = m.levels)
##                   pred.data.c[, mediator] <- factor(PredictMc,
##                     levels = 1:m, labels = m.levels)
##                 }
##                 else {
##                   pred.data.t[, mediator] <- PredictMt
##                   pred.data.c[, mediator] <- PredictMc
##                 }
##                 ymat.t <- model.matrix(terms(model.y), data = pred.data.t)
##                 ymat.c <- model.matrix(terms(model.y), data = pred.data.c)
##                 if (isVglm.y) {
##                   if (VfamilyY == "tobit") {
##                     Pr1.tmp <- ymat.t %*% YModel[j, -2]
##                     Pr0.tmp <- ymat.c %*% YModel[j, -2]
##                     Pr1[, j] <- pmin(pmax(Pr1.tmp, model.y@misc$Lower),
##                       model.y@misc$Upper)
##                     Pr0[, j] <- pmin(pmax(Pr0.tmp, model.y@misc$Lower),
##                       model.y@misc$Upper)
##                   }
##                   else {
##                     stop("outcome model is in unsupported vglm family")
##                   }
##                 }
##                 else if (scalesim.y) {
##                   Pr1[, j] <- t(as.matrix(YModel[j, 1:(ncol(YModel) -
##                     1)])) %*% t(ymat.t)
##                   Pr0[, j] <- t(as.matrix(YModel[j, 1:(ncol(YModel) -
##                     1)])) %*% t(ymat.c)
##                 }
##                 else if (isMer.y) {
##                   if (e == 1) {
##                     Y.RANEF.A <- Y.RANEF1
##                     Y.RANEF.B <- Y.RANEF2
##                   }
##                   else if (e == 2) {
##                     Y.RANEF.A <- Y.RANEF3
##                     Y.RANEF.B <- Y.RANEF4
##                   }
##                   else if (e == 3) {
##                     Y.RANEF.A <- Y.RANEF1
##                     Y.RANEF.B <- Y.RANEF3
##                   }
##                   else {
```

```
##                       Y.RANEF.A <- Y.RANEF2
##                       Y.RANEF.B <- Y.RANEF4
##                     }
##                     Pr1[, j] <- t(as.matrix(YModel.fixef.sim[j,
##                       ])) %*% t(ymat.t) + Y.RANEF.A[j, ]
##                     Pr0[, j] <- t(as.matrix(YModel.fixef.sim[j,
##                       ])) %*% t(ymat.c) + Y.RANEF.B[j, ]
##                   }
##                   else {
##                     Pr1[, j] <- t(as.matrix(YModel[j, ])) %*%
##                       t(ymat.t)
##                     Pr0[, j] <- t(as.matrix(YModel[j, ])) %*%
##                       t(ymat.c)
##                   }
##                   rm(ymat.t, ymat.c, pred.data.t, pred.data.c)
##                 }
##                 if (isGlm.y) {
##                   Pr1 <- apply(Pr1, 2, model.y$family$linkinv)
##                   Pr0 <- apply(Pr0, 2, model.y$family$linkinv)
##                 }
##                 else if (isSurvreg.y) {
##                   dd <- survival::survreg.distributions[[model.y$dist]]
##                   if (is.null(dd$itrans)) {
##                     itrans <- function(x) x
##                   }
##                   else {
##                     itrans <- dd$itrans
##                   }
##                   Pr1 <- apply(Pr1, 2, itrans)
##                   Pr0 <- apply(Pr0, 2, itrans)
##                 }
##                 else if (isMer.y && getCall(model.y)[[1]] ==
##                   "glmer") {
##                   Pr1 <- apply(Pr1, 2, Y.fun$linkinv)
##                   Pr0 <- apply(Pr0, 2, Y.fun$linkinv)
##                 }
##                 effects.tmp[, , e] <- Pr1 - Pr0
##                 rm(Pr1, Pr0)
##             }
##             if (!isMer.m && !isMer.y) {
##                 rm(PredictM1, PredictM0, YModel, MModel)
##             }
##             else if (!isMer.m && isMer.y) {
##                 rm(PredictM1, PredictM0, YModel.ranef.sim)
```

```
##             }
##             else {
##                 rm(PredictM1, PredictM0, MModel.ranef.sim)
##             }
##             et1 <- effects.tmp[, , 1]
##             et2 <- effects.tmp[, , 2]
##             et3 <- effects.tmp[, , 3]
##             et4 <- effects.tmp[, , 4]
##             delta.1 <- t(as.matrix(apply(et1, 2, weighted.mean,
##                 w = weights)))
##             delta.0 <- t(as.matrix(apply(et2, 2, weighted.mean,
##                 w = weights)))
##             zeta.1 <- t(as.matrix(apply(et3, 2, weighted.mean,
##                 w = weights)))
##             zeta.0 <- t(as.matrix(apply(et4, 2, weighted.mean,
##                 w = weights)))
##             rm(effects.tmp)
##             tau <- (zeta.1 + delta.0 + zeta.0 + delta.1)/2
##             nu.0 <- delta.0/tau
##             nu.1 <- delta.1/tau
##             delta.avg <- (delta.1 + delta.0)/2
##             zeta.avg <- (zeta.1 + zeta.0)/2
##             nu.avg <- (nu.1 + nu.0)/2
##             d0 <- mean(delta.0)
##             d1 <- mean(delta.1)
##             z1 <- mean(zeta.1)
##             z0 <- mean(zeta.0)
##             tau.coef <- mean(tau)
##             n0 <- median(nu.0)
##             n1 <- median(nu.1)
##             d.avg <- (d0 + d1)/2
##             z.avg <- (z0 + z1)/2
##             n.avg <- (n0 + n1)/2
##             if (isMer.y | isMer.m) {
##                 if (!is.null(group.m) && group.name == group.m) {
##                   G <- length(unique(group.id.m))
##                   delta.1.group <- matrix(NA, G, sims)
##                   delta.0.group <- matrix(NA, G, sims)
##                   zeta.1.group <- matrix(NA, G, sims)
##                   zeta.0.group <- matrix(NA, G, sims)
##                   for (g in 1:G) {
##                     0
##                     delta.1.group[g, ] <- t(apply(matrix(et1[group.id.m ==
##                       unique(group.id.m)[g], ], ncol = sims),
```

```
##                     2, weighted.mean, w = weights[group.id.m ==
##                       unique(group.id.m)[g]]))
##                   delta.0.group[g, ] <- t(apply(matrix(et2[group.id.m ==
##                     unique(group.id.m)[g], ], ncol = sims),
##                     2, weighted.mean, w = weights[group.id.m ==
##                       unique(group.id.m)[g]]))
##                   zeta.1.group[g, ] <- t(apply(matrix(et3[group.id.m ==
##                     unique(group.id.m)[g], ], ncol = sims),
##                     2, weighted.mean, w = weights[group.id.m ==
##                       unique(group.id.m)[g]]))
##                   zeta.0.group[g, ] <- t(apply(matrix(et4[group.id.m ==
##                     unique(group.id.m)[g], ], ncol = sims),
##                     2, weighted.mean, w = weights[group.id.m ==
##                       unique(group.id.m)[g]]))
##                 }
##               }
##               else {
##                 G <- length(unique(group.id.y))
##                 delta.1.group <- matrix(NA, G, sims)
##                 delta.0.group <- matrix(NA, G, sims)
##                 zeta.1.group <- matrix(NA, G, sims)
##                 zeta.0.group <- matrix(NA, G, sims)
##                 for (g in 1:G) {
##                   delta.1.group[g, ] <- t(apply(matrix(et1[group.id.y ==
##                     unique(group.id.y)[g], ], ncol = sims),
##                     2, weighted.mean, w = weights[group.id.y ==
##                       unique(group.id.y)[g]]))
##                   delta.0.group[g, ] <- t(apply(matrix(et2[group.id.y ==
##                     unique(group.id.y)[g], ], ncol = sims),
##                     2, weighted.mean, w = weights[group.id.y ==
##                       unique(group.id.y)[g]]))
##                   zeta.1.group[g, ] <- t(apply(matrix(et3[group.id.y ==
##                     unique(group.id.y)[g], ], ncol = sims),
##                     2, weighted.mean, w = weights[group.id.y ==
##                       unique(group.id.y)[g]]))
##                   zeta.0.group[g, ] <- t(apply(matrix(et4[group.id.y ==
##                     unique(group.id.y)[g], ], ncol = sims),
##                     2, weighted.mean, w = weights[group.id.y ==
##                       unique(group.id.y)[g]]))
##                 }
##               }
##               tau.group <- (zeta.1.group + delta.0.group +
##                 zeta.0.group + delta.1.group)/2
##               nu.0.group <- delta.0.group/tau.group
```

```
##                 nu.1.group <- delta.1.group/tau.group
##                 delta.avg.group <- (delta.1.group + delta.0.group)/2
##                 zeta.avg.group <- (zeta.1.group + zeta.0.group)/2
##                 nu.avg.group <- (nu.1.group + nu.0.group)/2
##                 d0.group <- apply(delta.0.group, 1, mean)
##                 d1.group <- apply(delta.1.group, 1, mean)
##                 z1.group <- apply(zeta.1.group, 1, mean)
##                 z0.group <- apply(zeta.0.group, 1, mean)
##                 tau.coef.group <- apply(tau.group, 1, mean)
##                 n0.group <- apply(nu.0.group, 1, median)
##                 n1.group <- apply(nu.1.group, 1, median)
##                 d.avg.group <- (d0.group + d1.group)/2
##                 z.avg.group <- (z0.group + z1.group)/2
##                 n.avg.group <- (n0.group + n1.group)/2
##             }
##         }
##         else {
##             if (isMer.m | isMer.y) {
##                 stop("'boot' must be 'FALSE' for models used")
##             }
##             Call.M <- getCall(model.m)
##             Call.Y <- getCall(model.y)
##             delta.1 <- matrix(NA, sims, 1)
##             delta.0 <- matrix(NA, sims, 1)
##             zeta.1 <- matrix(NA, sims, 1)
##             zeta.0 <- matrix(NA, sims, 1)
##             tau <- matrix(NA, sims, 1)
##             for (b in 1:(sims + 1)) {
##                 index <- sample(1:n, n, replace = TRUE)
##                 if (b == sims + 1) {
##                   index <- 1:n
##                 }
##                 if (isSurvreg.m) {
##                   if (ncol(model.m$y) > 2) {
##                     stop("unsupported censoring type")
##                   }
##                   mname <- names(m.data)[1]
##                   if (substr(mname, 1, 4) != "Surv") {
##                     stop("refit the survival model with 'Surv' used directly in model formula")
##                   }
##                   nc <- nchar(mediator)
##                   eventname <- substr(mname, 5 + nc + 3, nchar(mname) -
##                     1)
##                   if (nchar(eventname) == 0) {
```

```
##                     m.data.tmp <- data.frame(m.data, as.numeric(m.data[,
##                       1L][, 1L]))
##                     names(m.data.tmp)[c(1L, ncol(m.data) + 1)] <- c(mname,
##                       mediator)
##                   }
##                   else {
##                     m.data.tmp <- data.frame(m.data, as.numeric(m.data[,
##                       1L][, 1L]), as.numeric(model.m$y[, 2]))
##                     names(m.data.tmp)[c(1L, ncol(m.data) + (1:2))] <- c(mname,
##                       mediator, eventname)
##                   }
##                   Call.M$data <- m.data.tmp[index, ]
##                 }
##                 else {
##                   Call.M$data <- m.data[index, ]
##                 }
##                 if (isSurvreg.y) {
##                   if (ncol(model.y$y) > 2) {
##                     stop("unsupported censoring type")
##                   }
##                   yname <- names(y.data)[1]
##                   if (substr(yname, 1, 4) != "Surv") {
##                     stop("refit the survival model with ‘Surv’ used directly in model formula")
##                   }
##                   if (is.null(outcome)) {
##                     stop("‘outcome’ must be supplied for survreg outcome with boot")
##                   }
##                   nc <- nchar(outcome)
##                   eventname <- substr(yname, 5 + nc + 3, nchar(yname) -
##                     1)
##                   if (nchar(eventname) == 0) {
##                     y.data.tmp <- data.frame(y.data, as.numeric(y.data[,
##                       1L][, 1L]))
##                     names(y.data.tmp)[c(1L, ncol(y.data) + 1)] <- c(yname,
##                       outcome)
##                   }
##                   else {
##                     y.data.tmp <- data.frame(y.data, as.numeric(y.data[,
##                       1L][, 1L]), as.numeric(model.y$y[, 2]))
##                     names(y.data.tmp)[c(1L, ncol(y.data) + (1:2))] <- c(yname,
##                       outcome, eventname)
##                   }
##                   Call.Y$data <- y.data.tmp[index, ]
##                 }
```

```
##                 else {
##                   Call.Y$data <- y.data[index, ]
##                 }
##                 Call.M$weights <- m.data[index, "(weights)"]
##                 Call.Y$weights <- y.data[index, "(weights)"]
##                 if (isOrdered.m && length(unique(y.data[index,
##                   mediator])) != m) {
##                   stop("insufficient variation on mediator")
##                 }
##                 new.fit.M <- eval.parent(Call.M)
##                 new.fit.Y <- eval.parent(Call.Y)
##                 pred.data.t <- pred.data.c <- m.data
##                 if (isFactorT) {
##                   pred.data.t[, treat] <- factor(cat.1, levels = t.levels)
##                   pred.data.c[, treat] <- factor(cat.0, levels = t.levels)
##                 }
##                 else {
##                   pred.data.t[, treat] <- cat.1
##                   pred.data.c[, treat] <- cat.0
##                 }
##                 if (!is.null(covariates)) {
##                   for (p in 1:length(covariates)) {
##                     vl <- names(covariates[p])
##                     if (is.factor(pred.data.t[, vl])) {
##                       pred.data.t[, vl] <- pred.data.c[, vl] <- factor(covariates[[p]],
##                         levels = levels(m.data[, vl]))
##                     }
##                     else {
##                       pred.data.t[, vl] <- pred.data.c[, vl] <- covariates[[p]]
##                     }
##                   }
##                 }
##                 if (isGlm.m) {
##                   muM1 <- predict(new.fit.M, type = "response",
##                     newdata = pred.data.t)
##                   muM0 <- predict(new.fit.M, type = "response",
##                     newdata = pred.data.c)
##                   if (FamilyM == "poisson") {
##                     PredictM1 <- rpois(n, lambda = muM1)
##                     PredictM0 <- rpois(n, lambda = muM0)
##                   }
##                   else if (FamilyM == "Gamma") {
##                     shape <- gamma.shape(new.fit.M)$alpha
##                     PredictM1 <- rgamma(n, shape = shape, scale = muM1/shape)
```

```
##                     PredictM0 <- rgamma(n, shape = shape, scale = muM0/shape)
##                   }
##                   else if (FamilyM == "binomial") {
##                     PredictM1 <- rbinom(n, size = 1, prob = muM1)
##                     PredictM0 <- rbinom(n, size = 1, prob = muM0)
##                   }
##                   else if (FamilyM == "gaussian") {
##                     sigma <- sqrt(summary(new.fit.M)$dispersion)
##                     error <- rnorm(n, mean = 0, sd = sigma)
##                     PredictM1 <- muM1 + error
##                     PredictM0 <- muM0 + error
##                   }
##                   else if (FamilyM == "inverse.gaussian") {
##                     disp <- summary(new.fit.M)$dispersion
##                     PredictM1 <- SuppDists::rinvGauss(n, nu = muM1,
##                       lambda = 1/disp)
##                     PredictM0 <- SuppDists::rinvGauss(n, nu = muM0,
##                       lambda = 1/disp)
##                   }
##                   else {
##                     stop("unsupported glm family")
##                   }
##                 }
##                 else if (isOrdered.m) {
##                   probs_m1 <- predict(new.fit.M, newdata = pred.data.t,
##                     type = "probs")
##                   probs_m0 <- predict(new.fit.M, newdata = pred.data.c,
##                     type = "probs")
##                   draws_m1 <- matrix(NA, n, m)
##                   draws_m0 <- matrix(NA, n, m)
##                   for (ii in 1:n) {
##                     draws_m1[ii, ] <- t(rmultinom(1, 1, prob = probs_m1[ii,
##                       ]))
##                     draws_m0[ii, ] <- t(rmultinom(1, 1, prob = probs_m0[ii,
##                       ]))
##                   }
##                   PredictM1 <- apply(draws_m1, 1, indexmax)
##                   PredictM0 <- apply(draws_m0, 1, indexmax)
##                 }
##                 else if (isRq.m) {
##                   call.new <- new.fit.M$call
##                   call.new$tau <- runif(n)
##                   newfits <- eval.parent(call.new)
##                   tt <- delete.response(terms(new.fit.M))
```

```
##                   m.t <- model.frame(tt, pred.data.t, xlev = new.fit.M$xlevels)
##                   m.c <- model.frame(tt, pred.data.c, xlev = new.fit.M$xlevels)
##                   X.t <- model.matrix(tt, m.t, contrasts = new.fit.M$contrasts)
##                   X.c <- model.matrix(tt, m.c, contrasts = new.fit.M$contrasts)
##                   rm(tt, m.t, m.c)
##                   PredictM1 <- rowSums(X.t * t(newfits$coefficients))
##                   PredictM0 <- rowSums(X.c * t(newfits$coefficients))
##                   rm(newfits, X.t, X.c)
##                 }
##                 else if (isLm.m) {
##                   sigma <- summary(new.fit.M)$sigma
##                   error <- rnorm(n, mean = 0, sd = sigma)
##                   PredictM1 <- predict(new.fit.M, type = "response",
##                     newdata = pred.data.t) + error
##                   PredictM0 <- predict(new.fit.M, type = "response",
##                     newdata = pred.data.c) + error
##                   rm(error)
##                 }
##                 else if (isSurvreg.m) {
##                   dd <- survival::survreg.distributions[[new.fit.M$dist]]
##                   if (is.null(dd$itrans)) {
##                     itrans <- function(x) x
##                   }
##                   else {
##                     itrans <- dd$itrans
##                   }
##                   dname <- dd$dist
##                   if (is.null(dname)) {
##                     dname <- new.fit.M$dist
##                   }
##                   scale <- new.fit.M$scale
##                   lpM1 <- predict(new.fit.M, newdata = pred.data.t,
##                     type = "linear")
##                   lpM0 <- predict(new.fit.M, newdata = pred.data.c,
##                     type = "linear")
##                   error <- switch(dname, extreme = log(rweibull(n,
##                     shape = 1, scale = 1)), gaussian = rnorm(n),
##                     logistic = rlogis(n), t = rt(n, df = dd$parms))
##                   PredictM1 <- as.numeric(itrans(lpM1 + scale *
##                     error))
##                   PredictM0 <- as.numeric(itrans(lpM0 + scale *
##                     error))
##                   rm(error)
##                 }
```

```
##                 else {
##                   stop("mediator model is not yet implemented")
##                 }
##                 effects.tmp <- matrix(NA, nrow = n, ncol = 4)
##                 for (e in 1:4) {
##                   tt <- switch(e, c(1, 1, 1, 0), c(0, 0, 1, 0),
##                     c(1, 0, 1, 1), c(1, 0, 0, 0))
##                   pred.data.t <- pred.data.c <- y.data
##                   if (!is.null(covariates)) {
##                     for (p in 1:length(covariates)) {
##                       vl <- names(covariates[p])
##                       if (is.factor(pred.data.t[, vl])) {
##                         pred.data.t[, vl] <- pred.data.c[, vl] <- factor(covariates[[p]],
##                           levels = levels(y.data[, vl]))
##                       }
##                       else {
##                         pred.data.t[, vl] <- pred.data.c[, vl] <- covariates[[p]]
##                       }
##                     }
##                   }
##                   cat.t <- ifelse(tt[1], cat.1, cat.0)
##                   cat.c <- ifelse(tt[2], cat.1, cat.0)
##                   cat.t.ctrl <- ifelse(tt[1], cat.0, cat.1)
##                   cat.c.ctrl <- ifelse(tt[2], cat.0, cat.1)
##                   if (isFactorT) {
##                     pred.data.t[, treat] <- factor(cat.t, levels = t.levels)
##                     pred.data.c[, treat] <- factor(cat.c, levels = t.levels)
##                     if (!is.null(control)) {
##                       pred.data.t[, control] <- factor(cat.t.ctrl,
##                         levels = t.levels)
##                       pred.data.c[, control] <- factor(cat.c.ctrl,
##                         levels = t.levels)
##                     }
##                   }
##                   else {
##                     pred.data.t[, treat] <- cat.t
##                     pred.data.c[, treat] <- cat.c
##                     if (!is.null(control)) {
##                       pred.data.t[, control] <- cat.t.ctrl
##                       pred.data.c[, control] <- cat.c.ctrl
##                     }
##                   }
##                   PredictM1.tmp <- PredictM1
##                   PredictM0.tmp <- PredictM0
```

```
##                   PredictMt <- PredictM1 * tt[3] + PredictM0 *
##                     (1 - tt[3])
##                   PredictMc <- PredictM1 * tt[4] + PredictM0 *
##                     (1 - tt[4])
##                   if (isFactorM) {
##                     pred.data.t[, mediator] <- factor(PredictMt,
##                       levels = 1:m, labels = m.levels)
##                     pred.data.c[, mediator] <- factor(PredictMc,
##                       levels = 1:m, labels = m.levels)
##                   }
##                   else {
##                     pred.data.t[, mediator] <- PredictMt
##                     pred.data.c[, mediator] <- PredictMc
##                   }
##                   if (isRq.y) {
##                     pr.1 <- predict(new.fit.Y, type = "response",
##                       newdata = pred.data.t, interval = "none")
##                     pr.0 <- predict(new.fit.Y, type = "response",
##                       newdata = pred.data.c, interval = "none")
##                   }
##                   else {
##                     pr.1 <- predict(new.fit.Y, type = "response",
##                       newdata = pred.data.t)
##                     pr.0 <- predict(new.fit.Y, type = "response",
##                       newdata = pred.data.c)
##                   }
##                   pr.mat <- as.matrix(cbind(pr.1, pr.0))
##                   effects.tmp[, e] <- pr.mat[, 1] - pr.mat[,
##                     2]
##                   rm(pred.data.t, pred.data.c, pr.1, pr.0, pr.mat)
##                 }
##                 if (b == sims + 1) {
##                   d1 <- weighted.mean(effects.tmp[, 1], weights)
##                   d0 <- weighted.mean(effects.tmp[, 2], weights)
##                   z1 <- weighted.mean(effects.tmp[, 3], weights)
##                   z0 <- weighted.mean(effects.tmp[, 4], weights)
##                 }
##                 else {
##                   delta.1[b] <- weighted.mean(effects.tmp[, 1],
##                     weights)
##                   delta.0[b] <- weighted.mean(effects.tmp[, 2],
##                     weights)
##                   zeta.1[b] <- weighted.mean(effects.tmp[, 3],
##                     weights)
```

```
##                 zeta.0[b] <- weighted.mean(effects.tmp[, 4],
##                   weights)
##               }
##           }
##           tau.coef <- (d1 + d0 + z1 + z0)/2
##           n0 <- d0/tau.coef
##           n1 <- d1/tau.coef
##           d.avg <- (d1 + d0)/2
##           z.avg <- (z1 + z0)/2
##           n.avg <- (n0 + n1)/2
##           tau <- (delta.1 + delta.0 + zeta.1 + zeta.0)/2
##           nu.0 <- delta.0/tau
##           nu.1 <- delta.1/tau
##           delta.avg <- (delta.0 + delta.1)/2
##           zeta.avg <- (zeta.0 + zeta.1)/2
##           nu.avg <- (nu.0 + nu.1)/2
##       }
##       low <- (1 - conf.level)/2
##       high <- 1 - low
##       if (boot & boot.ci.type == "bca") {
##           BC.CI <- function(theta) {
##               z.inv <- length(theta[theta < mean(theta)])/sims
##               z <- qnorm(z.inv)
##               U <- (sims - 1) * (mean(theta) - theta)
##               top <- sum(U^3)
##               under <- 6 * (sum(U^2))^{
##                 3/2
##               }
##               a <- top/under
##               lower.inv <- pnorm(z + (z + qnorm(low))/(1 -
##                 a * (z + qnorm(low))))
##               lower2 <- lower <- quantile(theta, lower.inv)
##               upper.inv <- pnorm(z + (z + qnorm(high))/(1 -
##                 a * (z + qnorm(high))))
##               upper2 <- upper <- quantile(theta, upper.inv)
##               return(c(lower, upper))
##           }
##           d0.ci <- BC.CI(delta.0)
##           d1.ci <- BC.CI(delta.1)
##           tau.ci <- BC.CI(tau)
##           z1.ci <- BC.CI(zeta.1)
##           z0.ci <- BC.CI(zeta.0)
##           n0.ci <- BC.CI(nu.0)
##           n1.ci <- BC.CI(nu.1)
```

```
##             d.avg.ci <- BC.CI(delta.avg)
##             z.avg.ci <- BC.CI(zeta.avg)
##             n.avg.ci <- BC.CI(nu.avg)
##         }
##         else {
##             d0.ci <- quantile(delta.0, c(low, high), na.rm = TRUE)
##             d1.ci <- quantile(delta.1, c(low, high), na.rm = TRUE)
##             tau.ci <- quantile(tau, c(low, high), na.rm = TRUE)
##             z1.ci <- quantile(zeta.1, c(low, high), na.rm = TRUE)
##             z0.ci <- quantile(zeta.0, c(low, high), na.rm = TRUE)
##             n0.ci <- quantile(nu.0, c(low, high), na.rm = TRUE)
##             n1.ci <- quantile(nu.1, c(low, high), na.rm = TRUE)
##             d.avg.ci <- quantile(delta.avg, c(low, high), na.rm = TRUE)
##             z.avg.ci <- quantile(zeta.avg, c(low, high), na.rm = TRUE)
##             n.avg.ci <- quantile(nu.avg, c(low, high), na.rm = TRUE)
##         }
##         d0.p <- pval(delta.0, d0)
##         d1.p <- pval(delta.1, d1)
##         d.avg.p <- pval(delta.avg, d.avg)
##         z0.p <- pval(zeta.0, z0)
##         z1.p <- pval(zeta.1, z1)
##         z.avg.p <- pval(zeta.avg, z.avg)
##         n0.p <- pval(nu.0, n0)
##         n1.p <- pval(nu.1, n1)
##         n.avg.p <- pval(nu.avg, n.avg)
##         tau.p <- pval(tau, tau.coef)
##         if (isMer.y | isMer.m) {
##             QUANT <- function(object) {
##                 z <- quantile(object, c(low, high), na.rm = TRUE)
##                 return(z)
##             }
##             d0.ci.group <- t(apply(delta.0.group, 1, QUANT))
##             d1.ci.group <- t(apply(delta.1.group, 1, QUANT))
##             tau.ci.group <- t(apply(tau.group, 1, QUANT))
##             z1.ci.group <- t(apply(zeta.1.group, 1, QUANT))
##             z0.ci.group <- t(apply(zeta.0.group, 1, QUANT))
##             n0.ci.group <- t(apply(nu.0.group, 1, QUANT))
##             n1.ci.group <- t(apply(nu.1.group, 1, QUANT))
##             d.avg.ci.group <- t(apply(delta.avg.group, 1, QUANT))
##             z.avg.ci.group <- t(apply(zeta.avg.group, 1, QUANT))
##             n.avg.ci.group <- t(apply(nu.avg.group, 1, QUANT))
##             d0.p.group <- rep(NA, G)
##             d1.p.group <- rep(NA, G)
##             d.avg.p.group <- rep(NA, G)
```

```
##             z0.p.group <- rep(NA, G)
##             z1.p.group <- rep(NA, G)
##             z.avg.p.group <- rep(NA, G)
##             n0.p.group <- rep(NA, G)
##             n1.p.group <- rep(NA, G)
##             n.avg.p.group <- rep(NA, G)
##             tau.p.group <- rep(NA, G)
##             for (g in 1:G) {
##                 d0.p.group[g] <- pval(delta.0.group[g, ], d0.group[g])
##                 d1.p.group[g] <- pval(delta.1.group[g, ], d1.group[g])
##                 d.avg.p.group[g] <- pval(delta.avg.group[g, ],
##                   d.avg.group[g])
##                 z0.p.group[g] <- pval(zeta.0.group[g, ], z0.group[g])
##                 z1.p.group[g] <- pval(zeta.1.group[g, ], z1.group[g])
##                 z.avg.p.group[g] <- pval(zeta.avg.group[g, ],
##                   z.avg.group[g])
##                 n0.p.group[g] <- pval(nu.0.group[g, ], n0.group[g])
##                 n1.p.group[g] <- pval(nu.1.group[g, ], n1.group[g])
##                 n.avg.p.group[g] <- pval(nu.avg.group[g, ], n.avg.group[g])
##                 tau.p.group[g] <- pval(tau.group[g, ], tau.coef.group[g])
##             }
##         }
##         INT <- paste(treat, mediator, sep = ":") %in% attr(terms(model.y),
##             "term.labels") | paste(mediator, treat, sep = ":") %in%
##             attr(terms(model.y), "term.labels")
##         if (!INT & isGam.y) {
##             INT <- !isTRUE(all.equal(d0, d1))
##         }
##         if (long && !isMer.y && !isMer.m) {
##             out <- list(d0 = d0, d1 = d1, d0.ci = d0.ci, d1.ci = d1.ci,
##                 d0.p = d0.p, d1.p = d1.p, d0.sims = delta.0,
##                 d1.sims = delta.1, z0 = z0, z1 = z1, z0.ci = z0.ci,
##                 z1.ci = z1.ci, z0.p = z0.p, z1.p = z1.p, z0.sims = zeta.0,
##                 z1.sims = zeta.1, n0 = n0, n1 = n1, n0.ci = n0.ci,
##                 n1.ci = n1.ci, n0.p = n0.p, n1.p = n1.p, n0.sims = nu.0,
##                 n1.sims = nu.1, tau.coef = tau.coef, tau.ci = tau.ci,
##                 tau.p = tau.p, tau.sims = tau, d.avg = d.avg,
##                 d.avg.p = d.avg.p, d.avg.ci = d.avg.ci, d.avg.sims = delta.avg,
##                 z.avg = z.avg, z.avg.p = z.avg.p, z.avg.ci = z.avg.ci,
##                 z.avg.sims = zeta.avg, n.avg = n.avg, n.avg.p = n.avg.p,
##                 n.avg.ci = n.avg.ci, n.avg.sims = nu.avg, boot = boot,
##                 boot.ci.type = boot.ci.type, treat = treat, mediator = mediator,
##                 covariates = covariates, INT = INT, conf.level = conf.level,
##                 model.y = model.y, model.m = model.m, control.value = control.value,
```

```
##                 treat.value = treat.value, nobs = n, sims = sims,
##                 call = cl, robustSE = robustSE, cluster = cluster)
##             class(out) <- "mediate"
##         }
##         if (!long && !isMer.y && !isMer.m) {
##             out <- list(d0 = d0, d1 = d1, d0.ci = d0.ci, d1.ci = d1.ci,
##                 d0.p = d0.p, d1.p = d1.p, z0 = z0, z1 = z1, z0.ci = z0.ci,
##                 z1.ci = z1.ci, z0.p = z0.p, z1.p = z1.p, n0 = n0,
##                 n1 = n1, n0.ci = n0.ci, n1.ci = n1.ci, n0.p = n0.p,
##                 n1.p = n1.p, tau.coef = tau.coef, tau.ci = tau.ci,
##                 tau.p = tau.p, d.avg = d.avg, d.avg.p = d.avg.p,
##                 d.avg.ci = d.avg.ci, z.avg = z.avg, z.avg.p = z.avg.p,
##                 z.avg.ci = z.avg.ci, n.avg = n.avg, n.avg.p = n.avg.p,
##                 n.avg.ci = n.avg.ci, boot = boot, boot.ci.type = boot.ci.type,
##                 treat = treat, mediator = mediator, covariates = covariates,
##                 INT = INT, conf.level = conf.level, model.y = model.y,
##                 model.m = model.m, control.value = control.value,
##                 treat.value = treat.value, nobs = n, sims = sims,
##                 call = cl, robustSE = robustSE, cluster = cluster)
##             class(out) <- "mediate"
##         }
##         if (long && (isMer.y || isMer.m)) {
##             out <- list(d0 = d0, d1 = d1, d0.ci = d0.ci, d1.ci = d1.ci,
##                 d0.p = d0.p, d1.p = d1.p, d0.sims = delta.0,
##                 d1.sims = delta.1, z0 = z0, z1 = z1, z0.ci = z0.ci,
##                 z1.ci = z1.ci, z0.p = z0.p, z1.p = z1.p, z0.sims = zeta.0,
##                 z1.sims = zeta.1, n0 = n0, n1 = n1, n0.ci = n0.ci,
##                 n1.ci = n1.ci, n0.p = n0.p, n1.p = n1.p, n0.sims = nu.0,
##                 n1.sims = nu.1, tau.coef = tau.coef, tau.ci = tau.ci,
##                 tau.p = tau.p, tau.sims = tau, d.avg = d.avg,
##                 d.avg.p = d.avg.p, d.avg.ci = d.avg.ci, d.avg.sims = delta.avg,
##                 z.avg = z.avg, z.avg.p = z.avg.p, z.avg.ci = z.avg.ci,
##                 z.avg.sims = zeta.avg, n.avg = n.avg, n.avg.p = n.avg.p,
##                 n.avg.ci = n.avg.ci, n.avg.sims = nu.avg, d0.group = d0.group,
##                 d1.group = d1.group, d0.ci.group = d0.ci.group,
##                 d1.ci.group = d1.ci.group, d0.p.group = d0.p.group,
##                 d1.p.group = d1.p.group, d0.sims.group = delta.0.group,
##                 d1.sims.group = delta.1.group, z0.group = z0.group,
##                 z1.group = z1.group, z0.ci.group = z0.ci.group,
##                 z1.ci.group = z1.ci.group, z0.p.group = z0.p.group,
##                 z1.p.group = z1.p.group, z0.sims.group = zeta.0.group,
##                 z1.sims.group = zeta.1.group, n0.group = n0.group,
##                 n1.group = n1.group, n0.ci.group = n0.ci.group,
##                 n1.ci.group = n1.ci.group, n0.p.group = n0.p.group,
```

```
##                 n1.p.group = n1.p.group, n0.sims.group = nu.0.group,
##                 n1.sims.group = nu.1.group, tau.coef.group = tau.coef.group,
##                 tau.ci.group = tau.ci.group, tau.p.group = tau.p.group,
##                 tau.sims.group = tau.group, d.avg.group = d.avg.group,
##                 d.avg.p.group = d.avg.p.group, d.avg.ci.group = d.avg.ci.group,
##                 d.avg.sims.group = delta.avg.group, z.avg.group = z.avg.group,
##                 z.avg.p.group = z.avg.p.group, z.avg.ci.group = z.avg.ci.group,
##                 z.avg.sims.group = zeta.avg.group, n.avg.group = n.avg.group,
##                 n.avg.p.group = n.avg.p.group, n.avg.ci.group = n.avg.ci.group,
##                 n.avg.sims.group = nu.avg.group, boot = boot,
##                 boot.ci.type = boot.ci.type, treat = treat, mediator = mediator,
##                 covariates = covariates, INT = INT, conf.level = conf.level,
##                 model.y = model.y, model.m = model.m, control.value = control.value,
##                 treat.value = treat.value, nobs = n, sims = sims,
##                 call = cl, group.m = group.m, group.y = group.y,
##                 group.name = group.name, group.id.m = group.id.m,
##                 group.id.y = group.id.y, group.id = group.id,
##                 robustSE = robustSE, cluster = cluster)
##             class(out) <- "mediate.mer"
##         }
##         if (!long && (isMer.y || isMer.m)) {
##             out <- list(d0 = d0, d1 = d1, d0.ci = d0.ci, d1.ci = d1.ci,
##                 d0.p = d0.p, d1.p = d1.p, z0 = z0, z1 = z1, z0.ci = z0.ci,
##                 z1.ci = z1.ci, z0.p = z0.p, z1.p = z1.p, n0 = n0,
##                 n1 = n1, n0.ci = n0.ci, n1.ci = n1.ci, n0.p = n0.p,
##                 n1.p = n1.p, tau.coef = tau.coef, tau.ci = tau.ci,
##                 tau.p = tau.p, d.avg = d.avg, d.avg.p = d.avg.p,
##                 d.avg.ci = d.avg.ci, z.avg = z.avg, z.avg.p = z.avg.p,
##                 z.avg.ci = z.avg.ci, n.avg = n.avg, n.avg.p = n.avg.p,
##                 n.avg.ci = n.avg.ci, d0.group = d0.group, d1.group = d1.group,
##                 d0.ci.group = d0.ci.group, d1.ci.group = d1.ci.group,
##                 d0.p.group = d0.p.group, d1.p.group = d1.p.group,
##                 z0.group = z0.group, z1.group = z1.group, z0.ci.group = z0.ci.group,
##                 z1.ci.group = z1.ci.group, z0.p.group = z0.p.group,
##                 z1.p.group = z1.p.group, n0.group = n0.group,
##                 n1.group = n1.group, n0.ci.group = n0.ci.group,
##                 n1.ci.group = n1.ci.group, n0.p.group = n0.p.group,
##                 n1.p.group = n1.p.group, tau.coef.group = tau.coef.group,
##                 tau.ci.group = tau.ci.group, tau.p.group = tau.p.group,
##                 d.avg.group = d.avg.group, d.avg.p.group = d.avg.p.group,
##                 d.avg.ci.group = d.avg.ci.group, z.avg.group = z.avg.group,
##                 z.avg.p.group = z.avg.p.group, z.avg.ci.group = z.avg.ci.group,
##                 n.avg.group = n.avg.group, n.avg.p.group = n.avg.p.group,
##                 n.avg.ci.group = n.avg.ci.group, boot = boot,
```

```
##                 boot.ci.type = boot.ci.type, treat = treat, mediator = mediator,
##                 covariates = covariates, INT = INT, conf.level = conf.level,
##                 model.y = model.y, model.m = model.m, control.value = control.value,
##                 treat.value = treat.value, nobs = n, sims = sims,
##                 call = cl, group.m = group.m, group.y = group.y,
##                 group.name = group.name, group.id.m = group.id.m,
##                 group.id.y = group.id.y, group.id = group.id,
##                 robustSE = robustSE, cluster = cluster)
##             class(out) <- "mediate.mer"
##         }
##         out
##     }
##     else {
##         if (boot != TRUE) {
##             warning("ordered outcome model can only be used with nonparametric bootstrap - option force
##             boot <- TRUE
##         }
##         if (isMer.m) {
##             stop("merMod class is not supported for ordered outcome")
##         }
##         n.ycat <- length(unique(model.response(y.data)))
##         delta.1 <- matrix(NA, sims, n.ycat)
##         delta.0 <- matrix(NA, sims, n.ycat)
##         zeta.1 <- matrix(NA, sims, n.ycat)
##         zeta.0 <- matrix(NA, sims, n.ycat)
##         tau <- matrix(NA, sims, n.ycat)
##         for (b in 1:(sims + 1)) {
##             index <- sample(1:n, n, replace = TRUE)
##             if (b == sims + 1) {
##                 index <- 1:n
##             }
##             Call.M <- model.m$call
##             Call.Y <- model.y$call
##             if (isSurvreg.m) {
##                 if (ncol(model.m$y) > 2) {
##                   stop("unsupported censoring type")
##                 }
##                 mname <- names(m.data)[1]
##                 if (substr(mname, 1, 4) != "Surv") {
##                   stop("refit the survival model with ‘Surv’ used directly in model formula")
##                 }
##                 nc <- nchar(mediator)
##                 eventname <- substr(mname, 5 + nc + 3, nchar(mname) -
##                   1)
```

```
##                 if (nchar(eventname) == 0) {
##                   m.data.tmp <- data.frame(m.data, as.numeric(m.data[,
##                     1L][, 1L]))
##                   names(m.data.tmp)[c(1L, ncol(m.data) + 1)] <- c(mname,
##                     mediator)
##                 }
##                 else {
##                   m.data.tmp <- data.frame(m.data, as.numeric(m.data[,
##                     1L][, 1L]), as.numeric(model.m$y[, 2]))
##                   names(m.data.tmp)[c(1L, ncol(m.data) + (1:2))] <- c(mname,
##                     mediator, eventname)
##                 }
##                 Call.M$data <- m.data.tmp[index, ]
##             }
##             else {
##                 Call.M$data <- m.data[index, ]
##             }
##             Call.Y$data <- y.data[index, ]
##             Call.M$weights <- m.data[index, "(weights)"]
##             Call.Y$weights <- y.data[index, "(weights)"]
##             new.fit.M <- eval.parent(Call.M)
##             new.fit.Y <- eval.parent(Call.Y)
##             if (isOrdered.m && length(unique(y.data[index, mediator])) !=
##                 m) {
##                 coefnames.y <- names(model.y$coefficients)
##                 coefnames.new.y <- names(new.fit.Y$coefficients)
##                 new.fit.Y.coef <- rep(0, length(coefnames.y))
##                 names(new.fit.Y.coef) <- coefnames.y
##                 new.fit.Y.coef[coefnames.new.y] <- new.fit.Y$coefficients
##                 new.fit.Y$coefficients <- new.fit.Y.coef
##             }
##             pred.data.t <- pred.data.c <- m.data
##             if (isFactorT) {
##                 pred.data.t[, treat] <- factor(cat.1, levels = t.levels)
##                 pred.data.c[, treat] <- factor(cat.0, levels = t.levels)
##             }
##             else {
##                 pred.data.t[, treat] <- cat.1
##                 pred.data.c[, treat] <- cat.0
##             }
##             if (!is.null(covariates)) {
##                 for (p in 1:length(covariates)) {
##                   vl <- names(covariates[p])
##                   if (is.factor(pred.data.t[, vl])) {
```

```
##                     pred.data.t[, vl] <- pred.data.c[, vl] <- factor(covariates[[p]],
##                       levels = levels(m.data[, vl]))
##                   }
##                   else {
##                     pred.data.t[, vl] <- pred.data.c[, vl] <- covariates[[p]]
##                   }
##                 }
##             }
##             if (isGlm.m) {
##                 muM1 <- predict(new.fit.M, type = "response",
##                   newdata = pred.data.t)
##                 muM0 <- predict(new.fit.M, type = "response",
##                   newdata = pred.data.c)
##                 if (FamilyM == "poisson") {
##                   PredictM1 <- rpois(n, lambda = muM1)
##                   PredictM0 <- rpois(n, lambda = muM0)
##                 }
##                 else if (FamilyM == "Gamma") {
##                   shape <- gamma.shape(model.m)$alpha
##                   PredictM1 <- rgamma(n, shape = shape, scale = muM1/shape)
##                   PredictM0 <- rgamma(n, shape = shape, scale = muM0/shape)
##                 }
##                 else if (FamilyM == "binomial") {
##                   PredictM1 <- rbinom(n, size = 1, prob = muM1)
##                   PredictM0 <- rbinom(n, size = 1, prob = muM0)
##                 }
##                 else if (FamilyM == "gaussian") {
##                   sigma <- sqrt(summary(model.m)$dispersion)
##                   error <- rnorm(n, mean = 0, sd = sigma)
##                   PredictM1 <- muM1 + error
##                   PredictM0 <- muM0 + error
##                 }
##                 else if (FamilyM == "inverse.gaussian") {
##                   disp <- summary(model.m)$dispersion
##                   PredictM1 <- SuppDists::rinvGauss(n, nu = muM1,
##                     lambda = 1/disp)
##                   PredictM0 <- SuppDists::rinvGauss(n, nu = muM0,
##                     lambda = 1/disp)
##                 }
##                 else {
##                   stop("unsupported glm family")
##                 }
##             }
##             else if (isOrdered.m) {
```

```
##                 probs_m1 <- predict(new.fit.M, type = "probs",
##                   newdata = pred.data.t)
##                 probs_m0 <- predict(new.fit.M, type = "probs",
##                   newdata = pred.data.c)
##                 draws_m1 <- matrix(NA, n, m)
##                 draws_m0 <- matrix(NA, n, m)
##                 for (ii in 1:n) {
##                   draws_m1[ii, ] <- t(rmultinom(1, 1, prob = probs_m1[ii,
##                     ]))
##                   draws_m0[ii, ] <- t(rmultinom(1, 1, prob = probs_m0[ii,
##                     ]))
##                 }
##                 PredictM1 <- apply(draws_m1, 1, indexmax)
##                 PredictM0 <- apply(draws_m0, 1, indexmax)
##             }
##             else if (isRq.m) {
##                 call.new <- new.fit.M$call
##                 call.new$tau <- runif(n)
##                 newfits <- eval.parent(call.new)
##                 tt <- delete.response(terms(new.fit.M))
##                 m.t <- model.frame(tt, pred.data.t, xlev = new.fit.M$xlevels)
##                 m.c <- model.frame(tt, pred.data.c, xlev = new.fit.M$xlevels)
##                 X.t <- model.matrix(tt, m.t, contrasts = new.fit.M$contrasts)
##                 X.c <- model.matrix(tt, m.c, contrasts = new.fit.M$contrasts)
##                 rm(tt, m.t, m.c)
##                 PredictM1 <- rowSums(X.t * t(newfits$coefficients))
##                 PredictM0 <- rowSums(X.c * t(newfits$coefficients))
##                 rm(newfits, X.t, X.c)
##             }
##             else if (isLm.m) {
##                 sigma <- summary(new.fit.M)$sigma
##                 error <- rnorm(n, mean = 0, sd = sigma)
##                 PredictM1 <- predict(new.fit.M, type = "response",
##                   newdata = pred.data.t) + error
##                 PredictM0 <- predict(new.fit.M, type = "response",
##                   newdata = pred.data.c) + error
##                 rm(error)
##             }
##             else if (isSurvreg.m) {
##                 dd <- survival::survreg.distributions[[new.fit.M$dist]]
##                 if (is.null(dd$itrans)) {
##                   itrans <- function(x) x
##                 }
##                 else {
```

```
##                 itrans <- dd$itrans
##               }
##               dname <- dd$dist
##               if (is.null(dname)) {
##                 dname <- new.fit.M$dist
##               }
##               scale <- new.fit.M$scale
##               lpM1 <- predict(new.fit.M, newdata = pred.data.t,
##                 type = "linear")
##               lpM0 <- predict(new.fit.M, newdata = pred.data.c,
##                 type = "linear")
##               error <- switch(dname, extreme = log(rweibull(n,
##                 shape = 1, scale = 1)), gaussian = rnorm(n),
##                 logistic = rlogis(n), t = rt(n, df = dd$parms))
##               PredictM1 <- as.numeric(itrans(lpM1 + scale *
##                 error))
##               PredictM0 <- as.numeric(itrans(lpM0 + scale *
##                 error))
##               rm(error)
##           }
##           else {
##               stop("mediator model is not yet implemented")
##           }
##           effects.tmp <- array(NA, dim = c(n, n.ycat, 4))
##           for (e in 1:4) {
##               tt <- switch(e, c(1, 1, 1, 0), c(0, 0, 1, 0),
##                 c(1, 0, 1, 1), c(1, 0, 0, 0))
##               pred.data.t <- pred.data.c <- y.data
##               if (!is.null(covariates)) {
##                 for (p in 1:length(covariates)) {
##                   vl <- names(covariates[p])
##                   if (is.factor(pred.data.t[, vl])) {
##                     pred.data.t[, vl] <- pred.data.c[, vl] <- factor(covariates[[p]],
##                       levels = levels(y.data[, vl]))
##                   }
##                   else {
##                     pred.data.t[, vl] <- pred.data.c[, vl] <- covariates[[p]]
##                   }
##                 }
##               }
##               cat.t <- ifelse(tt[1], cat.1, cat.0)
##               cat.c <- ifelse(tt[2], cat.1, cat.0)
##               cat.t.ctrl <- ifelse(tt[1], cat.0, cat.1)
##               cat.c.ctrl <- ifelse(tt[2], cat.0, cat.1)
```

```
##                 if (isFactorT) {
##                   pred.data.t[, treat] <- factor(cat.t, levels = t.levels)
##                   pred.data.c[, treat] <- factor(cat.c, levels = t.levels)
##                   if (!is.null(control)) {
##                     pred.data.t[, control] <- factor(cat.t.ctrl,
##                       levels = t.levels)
##                     pred.data.c[, control] <- factor(cat.c.ctrl,
##                       levels = t.levels)
##                   }
##                 }
##                 else {
##                   pred.data.t[, treat] <- cat.t
##                   pred.data.c[, treat] <- cat.c
##                   if (!is.null(control)) {
##                     pred.data.t[, control] <- cat.t.ctrl
##                     pred.data.c[, control] <- cat.c.ctrl
##                   }
##                 }
##                 PredictM1.tmp <- PredictM1
##                 PredictM0.tmp <- PredictM0
##                 PredictMt <- PredictM1 * tt[3] + PredictM0 *
##                   (1 - tt[3])
##                 PredictMc <- PredictM1 * tt[4] + PredictM0 *
##                   (1 - tt[4])
##                 if (isFactorM) {
##                   pred.data.t[, mediator] <- factor(PredictMt,
##                     levels = 1:m, labels = m.levels)
##                   pred.data.c[, mediator] <- factor(PredictMc,
##                     levels = 1:m, labels = m.levels)
##                 }
##                 else {
##                   pred.data.t[, mediator] <- PredictMt
##                   pred.data.c[, mediator] <- PredictMc
##                 }
##                 probs_p1 <- predict(new.fit.Y, newdata = pred.data.t,
##                   type = "probs")
##                 probs_p0 <- predict(new.fit.Y, newdata = pred.data.c,
##                   type = "probs")
##                 effects.tmp[, , e] <- probs_p1 - probs_p0
##                 rm(pred.data.t, pred.data.c, probs_p1, probs_p0)
##             }
##             if (b == sims + 1) {
##                 d1 <- apply(effects.tmp[, , 1], 2, weighted.mean,
##                   w = weights)
```

```
##                 d0 <- apply(effects.tmp[, , 2], 2, weighted.mean,
##                   w = weights)
##                 z1 <- apply(effects.tmp[, , 3], 2, weighted.mean,
##                   w = weights)
##                 z0 <- apply(effects.tmp[, , 4], 2, weighted.mean,
##                   w = weights)
##             }
##             else {
##                 delta.1[b, ] <- apply(effects.tmp[, , 1], 2,
##                   weighted.mean, w = weights)
##                 delta.0[b, ] <- apply(effects.tmp[, , 2], 2,
##                   weighted.mean, w = weights)
##                 zeta.1[b, ] <- apply(effects.tmp[, , 3], 2, weighted.mean,
##                   w = weights)
##                 zeta.0[b, ] <- apply(effects.tmp[, , 4], 2, weighted.mean,
##                   w = weights)
##             }
##         }
##         tau.coef <- (d1 + d0 + z1 + z0)/2
##         tau <- (zeta.1 + zeta.0 + delta.0 + delta.1)/2
##         low <- (1 - conf.level)/2
##         high <- 1 - low
##         if (boot.ci.type == "bca") {
##             BC.CI <- function(theta) {
##                 z.inv <- length(theta[theta < mean(theta)])/sims
##                 z <- qnorm(z.inv)
##                 U <- (sims - 1) * (mean(theta) - theta)
##                 top <- sum(U^3)
##                 under <- 6 * (sum(U^2))^{
##                   3/2
##                 }
##                 a <- top/under
##                 lower.inv <- pnorm(z + (z + qnorm(low))/(1 -
##                   a * (z + qnorm(low))))
##                 lower2 <- lower <- quantile(theta, lower.inv)
##                 upper.inv <- pnorm(z + (z + qnorm(high))/(1 -
##                   a * (z + qnorm(high))))
##                 upper2 <- upper <- quantile(theta, upper.inv)
##                 return(c(lower, upper))
##             }
##             d0.ci <- BC.CI(delta.0)
##             d1.ci <- BC.CI(delta.1)
##             tau.ci <- BC.CI(tau)
##             z1.ci <- BC.CI(zeta.1)
```

```
##             z0.ci <- BC.CI(zeta.0)
##         }
##         else {
##             CI <- function(theta) {
##                 return(quantile(theta, c(low, high), na.rm = TRUE))
##             }
##             d0.ci <- apply(delta.0, 2, CI)
##             d1.ci <- apply(delta.1, 2, CI)
##             tau.ci <- apply(tau, 2, CI)
##             z1.ci <- apply(zeta.1, 2, CI)
##             z0.ci <- apply(zeta.0, 2, CI)
##         }
##         d0.p <- d1.p <- z0.p <- z1.p <- tau.p <- rep(NA, n.ycat)
##         for (i in 1:n.ycat) {
##             d0.p[i] <- pval(delta.0[, i], d0[i])
##             d1.p[i] <- pval(delta.1[, i], d1[i])
##             z0.p[i] <- pval(zeta.0[, i], z0[i])
##             z1.p[i] <- pval(zeta.1[, i], z1[i])
##             tau.p[i] <- pval(tau[, i], tau.coef[i])
##         }
##         INT <- paste(treat, mediator, sep = ":") %in% attr(model.y$terms,
##             "term.labels") | paste(mediator, treat, sep = ":") %in%
##             attr(model.y$terms, "term.labels")
##         if (long) {
##             out <- list(d0 = d0, d1 = d1, d0.ci = d0.ci, d1.ci = d1.ci,
##                 d0.p = d0.p, d1.p = d1.p, d0.sims = delta.0,
##                 d1.sims = delta.1, tau.coef = tau.coef, tau.ci = tau.ci,
##                 tau.p = tau.p, z0 = z0, z1 = z1, z0.ci = z0.ci,
##                 z1.ci = z1.ci, z0.p = z0.p, z1.p = z1.p, z1.sims = zeta.1,
##                 z0.sims = zeta.0, tau.sims = tau, boot = boot,
##                 boot.ci.type = boot.ci.type, treat = treat, mediator = mediator,
##                 covariates = covariates, INT = INT, conf.level = conf.level,
##                 model.y = model.y, model.m = model.m, control.value = control.value,
##                 treat.value = treat.value, nobs = n, sims = sims,
##                 call = cl, robustSE = robustSE, cluster = cluster)
##         }
##         else {
##             out <- list(d0 = d0, d1 = d1, d0.ci = d0.ci, d1.ci = d1.ci,
##                 d0.p = d0.p, d1.p = d1.p, tau.coef = tau.coef,
##                 tau.ci = tau.ci, tau.p = tau.p, z0 = z0, z1 = z1,
##                 z0.ci = z0.ci, z1.ci = z1.ci, z0.p = z0.p, z1.p = z1.p,
##                 boot = boot, boot.ci.type = boot.ci.type, treat = treat,
##                 mediator = mediator, covariates = covariates,
##                 INT = INT, conf.level = conf.level, model.y = model.y,
```

```
##                 model.m = model.m, control.value = control.value,
##                 treat.value = treat.value, nobs = n, sims = sims,
##                 call = cl, robustSE = robustSE, cluster = cluster)
##         }
##         class(out) <- "mediate.order"
##         out
##     }
## }
## <environment: namespace:mediation>
```

polr()

```
MASS::polr
```

```
## function (formula, data, weights, start, ..., subset, na.action,
##     contrasts = NULL, Hess = FALSE, model = TRUE, method = c("logistic",
##         "probit", "loglog", "cloglog", "cauchit"))
## {
##     m <- match.call(expand.dots = FALSE)
##     method <- match.arg(method)
##     if (is.matrix(eval.parent(m$data)))
##         m$data <- as.data.frame(data)
##     m$start <- m$Hess <- m$method <- m$model <- m$... <- NULL
##     m[[1L]] <- quote(stats::model.frame)
##     m <- eval.parent(m)
##     Terms <- attr(m, "terms")
##     x <- model.matrix(Terms, m, contrasts)
##     xint <- match("(Intercept)", colnames(x), nomatch = 0L)
##     n <- nrow(x)
##     pc <- ncol(x)
##     cons <- attr(x, "contrasts")
##     if (xint > 0L) {
##         x <- x[, -xint, drop = FALSE]
##         pc <- pc - 1L
##     }
##     else warning("an intercept is needed and assumed")
##     wt <- model.weights(m)
##     if (!length(wt))
##         wt <- rep(1, n)
##     offset <- model.offset(m)
##     if (length(offset) <= 1L)
##         offset <- rep(0, n)
##     y <- model.response(m)
##     if (!is.factor(y))
##         stop("response must be a factor")
##     lev <- levels(y)
##     llev <- length(lev)
##     if (llev <= 2L)
##         stop("response must have 3 or more levels")
##     y <- unclass(y)
##     q <- llev - 1L
##     if (missing(start)) {
##         q1 <- llev%/%2L
##         y1 <- (y > q1)
##         X <- cbind(Intercept = rep(1, n), x)
```

```
##         fit <- switch(method, logistic = glm.fit(X, y1, wt, family = binomial(),
##             offset = offset), probit = glm.fit(X, y1, wt, family = binomial("probit"),
##             offset = offset), loglog = glm.fit(X, y1, wt, family = binomial("probit"),
##             offset = offset), cloglog = glm.fit(X, y1, wt, family = binomial("probit"),
##             offset = offset), cauchit = glm.fit(X, y1, wt, family = binomial("cauchit"),
##             offset = offset))
##         if (!fit$converged)
##             stop("attempt to find suitable starting values failed")
##         coefs <- fit$coefficients
##         if (any(is.na(coefs))) {
##             warning("design appears to be rank-deficient, so dropping some coefs")
##             keep <- names(coefs)[!is.na(coefs)]
##             coefs <- coefs[keep]
##             x <- x[, keep[-1L], drop = FALSE]
##             pc <- ncol(x)
##         }
##         logit <- function(p) log(p/(1 - p))
##         spacing <- logit((1L:q)/(q + 1L))
##         if (method != "logistic")
##             spacing <- spacing/1.7
##         gammas <- -coefs[1L] + spacing - spacing[q1]
##         start <- c(coefs[-1L], gammas)
##     }
##     else if (length(start) != pc + q)
##         stop("'start' is not of the correct length")
##     ans <- polr.fit(x, y, wt, start, offset, method, hessian = Hess,
##         ...)
##     beta <- ans$coefficients
##     zeta <- ans$zeta
##     deviance <- ans$deviance
##     res <- ans$res
##     niter <- c(f.evals = res$counts[1L], g.evals = res$counts[2L])
##     eta <- if (pc)
##         offset + drop(x %*% beta)
##     else offset + rep(0, n)
##     pfun <- switch(method, logistic = plogis, probit = pnorm,
##         loglog = pgumbel, cloglog = pGumbel, cauchit = pcauchy)
##     cumpr <- matrix(pfun(matrix(zeta, n, q, byrow = TRUE) - eta),
##         , q)
##     fitted <- t(apply(cumpr, 1L, function(x) diff(c(0, x, 1))))
##     dimnames(fitted) <- list(row.names(m), lev)
##     fit <- list(coefficients = beta, zeta = zeta, deviance = deviance,
##         fitted.values = fitted, lev = lev, terms = Terms, df.residual = sum(wt) -
##             pc - q, edf = pc + q, n = sum(wt), nobs = sum(wt),
```

```
##         call = match.call(), method = method, convergence = res$convergence,
##         niter = niter, lp = eta)
##     if (Hess) {
##         dn <- c(names(beta), names(zeta))
##         H <- res$hessian
##         dimnames(H) <- list(dn, dn)
##         fit$Hessian <- H
##     }
##     if (model)
##         fit$model <- m
##     fit$na.action <- attr(m, "na.action")
##     fit$contrasts <- cons
##     fit$xlevels <- .getXlevels(Terms, m)
##     class(fit) <- "polr"
##     fit
## }
## <bytecode: 0x7ff02df88108>
## <environment: namespace:MASS>
```