Estimating Equations in R: `geex`

B. Saul

2017-05-10

M-estimation theory provides a framework for asympotic properties of estimators that are solutions to estimating equations. Regression methods such as Generalized Linear Models (GLM) and Generalized Estimating Equations (GEE) fit in this framework. Countless R packages implement specific applications of estimating equations. A common reason to use M-estimation is to compute the empirical sandwich variance estimator - an asymptotically Normal and "robust" covariance. Many packages compute this variance estimator automatically, and packages such as `sandwich` take the output of other modeling methods to compute this variance estimate.

`geex` aims to be provide a more general framework that any modelling method can use to compute point and variance estimates for parameters that are solutions to estimating equations. The basic idea:

- Analyst provides three things: (1) data, (2) instructions on how to split the data into independent units and (3) a function that takes unit-level data and returns a function in terms of parameters.
- `geex` computes point estimates and variance estimates for the parameters.

Basic Setup

I mostly follow the notation of Stefanski and Boos. I tried to keep notation in the code similar to mathematical notation.

Suppose we have m independent or nearly independent units of observations.

$$\sum_{i=1}^{m} \psi(O_i, \theta) = 0$$

Where ψ is vector of length p corresponding to the number of parameters in θ.

For notational ease, let $\psi(O_i, \theta) = \psi_i$ Let:

$$A_i = -\frac{\partial \psi(O_i, \theta)}{\partial \theta}$$

$$A = \sum_{i=1}^{m} A_i$$

$$B_i = \psi_i \psi_i^T$$

$$B = \sum_{i=1}^{m} B_i$$

$$\Sigma = A^{-1} B \{A^{-1}\}^T$$

Stefanski & Boos example 1

Example 1 illustrates calculation of sample mean and variance using estimating equations. I generate a data set with 100 observations drawn from a Normal(5, 2) distribution. Table translates the estimating equations into the `R` function needed for `geex`:

Table 1: Translating math to code

$$\psi(Y_i, \theta) = \begin{pmatrix} Y_i - \theta_1 \\ (Y_i - \theta_1)^2 - \theta_2 \end{pmatrix}$$

```
SB1_eefun <- function(data){
  function(theta){
    with(data,
      c(Y - theta[1],
       (Y - theta[1])^2 - theta[2] )
    )
  }
}
```

With the `eeFUN` function prepared, it is passed to `estimate_equations` along with the data, a character string naming the variable that identifies groups within the dataset, and starting values for the root finder.

```
estimates <- estimate_equations(eeFUN = SB1_eefun,
                                data  = dt,
                                units = 'id',
                                roots = c(1,1))
```

Table 2: " Comparing estimates from closed form versus geex "

	$\hat{\theta}$	$\hat{\Sigma}$
Closed form	$\begin{pmatrix} 5.0087 & 3.8493 \end{pmatrix}$	$\begin{pmatrix} 0.0385 & -0.0205 \\ -0.0205 & 0.2596 \end{pmatrix}$
geex	$\begin{pmatrix} 5.0087 & 3.8493 \end{pmatrix}$	$\begin{pmatrix} 0.0385 & -0.0205 \\ -0.0205 & 0.2596 \end{pmatrix}$
Decimal of difference	$\begin{pmatrix} Inf & 15 \end{pmatrix}$	$\begin{pmatrix} 14 & 14 \\ 14 & 12 \end{pmatrix}$

Stefanski & Boos example 2

Example 2 illustrates calculation of a ratio estimator. I generate a data set with 100 observations where $Y \sim N(5, 2)$ and $X \sim N(2, 0.2)$. Table translates the estimating equations into the `R` function needed for `geex`:

Table 3: Translating math to code

$$\psi(Y_i, \theta) = \begin{pmatrix} Y_i - \theta_1 \\ X_i - \theta_2 \\ \theta_1 - \theta_3\theta_2 \end{pmatrix}$$

```
SB2_eefun <- function(data){
  function(theta){
    with(data,
      c(Y - theta[1],
        X - theta[2],
        theta[1] - (theta[3] * theta[2]) )
    )
  }
}
```

```r
estimates <- estimate_equations(eeFUN = SB2_eefun,
                                data  = dt, units = 'id',
                                roots = c(1, 1, 1))
```

Table 4: " test "

	$\hat{\theta}$	$\hat{\Sigma}$
Closed form	$\begin{pmatrix} 5.2667 & 1.9984 & 2.6355 \end{pmatrix}$	$\begin{pmatrix} 0.0381 & -0.0004 & 0.0196 \\ -0.0004 & 0.0003 & -0.0007 \\ 0.0196 & -0.0007 & 0.0107 \end{pmatrix}$
geex	$\begin{pmatrix} 5.2667 & 1.9984 & 2.6355 \end{pmatrix}$	$\begin{pmatrix} 0.0377 & -0.0004 & 0.0194 \\ -0.0004 & 0.0003 & -0.0006 \\ 0.0194 & -0.0006 & 0.0106 \end{pmatrix}$
Decimal of difference	$\begin{pmatrix} Inf & Inf & 16 \end{pmatrix}$	$\begin{pmatrix} 4 & 6 & 4 \\ 6 & 6 & 6 \\ 4 & 6 & 4 \end{pmatrix}$

Stefanski & Boos example 3

Example 3 illustrates calculation of a ratio estimator. I generate a data set with 100 observations where $Y \sim N(5, 4)$. Table translates the estimating equations into the `R` function needed for `geex`:

Table 5: Translating math to code

$\psi(Y_i, \theta) = \begin{pmatrix} Y_i - \theta_1 \\ X_i - \theta_2 \\ \sqrt{\theta_2} - \theta_3 \\ log(\theta_2) - \theta_4 \end{pmatrix}$	`SB3_eefun <- function(data){`
	`  function(theta){`
	`    with(data,`
	`      c(Y - theta[1],`
	`       (Y - theta[1])^2 - theta[2],`
	`       sqrt(theta[2]) - theta[3],`
	`       log(theta[2]) - theta[4])`
	`    )`
	`  }`
	`}`

```r
estimates <- estimate_equations(eeFUN= SB3_eefun,
                                data  = dt, units = 'id',
                                roots = c(1, 1, 1, 1))
```

Table 6: " Example 3 "

	$\hat{\theta}$	$\hat{\Sigma}$
Closed form	$\begin{pmatrix} 5.0117 & 16.5029 & 4.0624 & 2.8035 \end{pmatrix}$	$\begin{pmatrix} 0.1650 & 0.1116 & 0.0137 & 0.0068 \\ 0.1116 & 5.3752 & 0.6616 & 0.3257 \\ 0.0137 & 0.6616 & 0.0814 & 0.0401 \\ 0.0068 & 0.3257 & 0.0401 & 0.0197 \end{pmatrix}$
geex	$\begin{pmatrix} 5.0117 & 16.5029 & 4.0624 & 2.8035 \end{pmatrix}$	$\begin{pmatrix} 0.1650 & 0.1116 & 0.0137 & 0.0068 \\ 0.1116 & 5.3752 & 0.6616 & 0.3257 \\ 0.0137 & 0.6616 & 0.0814 & 0.0401 \\ 0.0068 & 0.3257 & 0.0401 & 0.0197 \end{pmatrix}$
Decimal of difference	$\begin{pmatrix} Inf & Inf & Inf & 16 \end{pmatrix}$	$\begin{pmatrix} 13 & 12 & 13 & 13 \\ 12 & 10 & 12 & 12 \\ 13 & 12 & 12 & 13 \\ 13 & 12 & 13 & 13 \end{pmatrix}$

Stefanski & Boos example 4

Example 4 illustrates calculation of an instumental variable estimator. I generate a data set with 100 observations where INPUT data generation. Table translates the estimating equations into the `R` function needed for `geex`:

Table 7: Translating math to code

$$\psi(Y_i, T_i, W_i\theta) = \begin{pmatrix} \theta_1 - T_i \\ \theta_2 - W_i \\ (Y_i - \theta_3 W_i)(\theta_2 - W_i) \\ (Y_i - \theta_4 W_i)(\theta_1 - T_i) \end{pmatrix}$$

```
SB_eefun <- function(data){
  function(theta){
    with(data,
      c(theta[1] - T_,
        theta[2] - W,
        (Y - (theta[3] * W)) * (theta[2] - W),
        (Y - (theta[4] * W)) * (theta[1] - T_))
    )
  }
}
```

```
estimates <- estimate_equations(eeFUN = SB4_eefun,
                                data  = dt, units = 'id',
                                roots = c(1, 1, 1, 1))
```

```
YW_model <- lm(Y ~ W, data = dt)
YT_model <- lm(Y ~ T_, data = dt)

WT_model <- lm(W ~ T_, data = dt)
## closed form roots
theta_cls <- c(theta1 = mean(dt$T_),
  theta2 = mean(dt$W),
  theta3 = coef(YW_model)[2],
  theta4 = coef(YT_model)[2]/coef(WT_model)[2])

## closed form covariance
# Not sure how compute SB's closed form since it depends on X, which is
# supposed to be unobserved.
# sigma2_e <- var(residuals(YW_model))
```

```
# sigma2_W <- var(dt$W)
# sigma2_U <- var(residuals(YT_model))

# ((sigma_e^2 * (1 + sigma_U^2 * sigma_e^2) + beta^2 * (sigma_U^2 * 1))/(1 + sigma_U^2 * sigma_e^2)^2) /
# ((sigma_e^2 * (1 + sigma_U^2 * sigma_e^2) + beta^2 * (sigma_U^2 * 1))/(1 + sigma_U^2 * sigma_e^2)^2) /
Sigma_cls <- matrix(NA, nrow = 2, ncol = 2)
```

Table 8: " Example 4 "

	$\hat{\theta}$	$\hat{\Sigma}$
Closed form	$\begin{pmatrix} 9.4961 & 4.9928 & 3.0057 & 3.0210 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$
geex	$\begin{pmatrix} 9.4961 & 4.9928 & 3.0057 & 3.0210 \end{pmatrix}$	$\begin{pmatrix} 0.0001 & 0.0001 \\ 0.0001 & 0.0001 \end{pmatrix}$
Decimal of difference	$\begin{pmatrix} Inf & Inf & 16 & 15 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$

Stefanski & Boos example 5

Example 5 illustrates calculation of an instumental variable estimator. I generate a data set with 100 observations where $X \sim N(2, 1)$. Let $\theta_0 = 0$. Table translates the estimating equations for the Hodges-Lehmann location estimation and the sample mean into the `R` function needed for `geex`:

Table 9: Translating math to code

$\psi(Y_i, \theta) = \begin{pmatrix} IC_{\hat{\theta}_{HL}}(X; \theta_0) - (\theta_1 - \theta_0) \\ X_i - \theta_2 \end{pmatrix}$	`SB5_eefun <- function(data, theta0 = 0){`
	`  Xi <- data$X`
	`  IC_HL <- (1/IC_denom) * (F0(Xi, theta0) - 0.5)`
	`  function(theta){`
	`     c(IC_HL - (theta[1] - theta0),`
	`       Xi - theta[2])`
	`  }`
	`}`

```
F0 <- function(y, theta0, distrFUN = pnorm){
  distrFUN(y - theta0, mean = 0)
}

f0 <- function(y, densFUN){
  densFUN(y, mean = 0)
}

integrand <- function(y, densFUN = dnorm){
  f0(y, densFUN = densFUN)^2
}

IC_denom <- integrate(integrand, lower = -Inf, upper = Inf)$value

SB5_eefun <- function(data){
  Xi <- data$X
  function(theta){
     IC_HL <- (1/IC_denom) * (F0(Xi, theta[1]) - 0.5)
     c(IC_HL,
```

```
        Xi - theta[2])
  }
}
```

```
estimates <- estimate_equations(eeFUN = SB5_eefun,
                                data  = dt, units = 'id',
                                roots = c(2, 1))
```

```
theta_cls <- c(hl.loc(dt$X), mean(dt$X))

## closed form covariance
# Not sure how compute SB's closed form since it depends on X, which is
# supposed to be unobserved.
Sigma_cls <- matrix(c(1/(12 * IC_denom^2) / n, NA, NA, var(dt$X)/100),
                    nrow = 2, ncol = 2, byrow = TRUE)
```

Table 10: " Example 5 "

	$\hat{\theta}$	$\hat{\Sigma}$
Closed form	$\begin{pmatrix} 1.9729 & 1.9422 \end{pmatrix}$	$\begin{pmatrix} 0.0105 & \\ & 0.0118 \end{pmatrix}$
geex	$\begin{pmatrix} 1.9729 & 1.9422 \end{pmatrix}$	$\begin{pmatrix} 0.0124 & 0.0117 \\ 0.0117 & 0.0117 \end{pmatrix}$
Decimal of difference	$\begin{pmatrix} 6 & Inf \end{pmatrix}$	$\begin{pmatrix} 3 & \\ & 4 \end{pmatrix}$

Stefanski & Boos example 6

Example 6 illustrates calculation of the Huber estimator of the center of symmetric distributions. I generate a data set with 100 observations where $Y \sim N(2, 1)$. Table translates the estimating equations for the Huber estimator for the center of symmetric distributions into the `R` function needed for `geex`:

Table 11: Translating math to code

$\psi_k(Y_i, \theta) = \big((Y_i - \theta) * I(\lvert(Y_i - \theta)\rvert \leq k) + k * sgn(Y_i - \theta)\big)$	`SB6_eefun <- function(data, k = 1.5){`
	`  function(theta){`
	`    x <- data$Y - theta[1]`
	`    if(abs(x) <= k) x else sign(x) * k`
	`  }`
	`}`

```
estimates <- estimate_equations(eeFUN = SB6_eefun,
                                data  = dt, units = 'id',
                                roots = 1)
```

```
theta_cls <- MASS::huber(dt$Y, tol = 1e-10)$mu

psi_k <- function(x, k = 1.5){
  if(abs(x) <= k) x else sign(x) * k
}

A <- lapply(dt$Y, function(y){
  x <- y - theta_cls
  -numDeriv::grad(psi_k, x = x)
```

```
}) %>% unlist() %>% mean()

B <- lapply(dt$Y, function(y){
  x <- y - theta_cls
  psi_k(x = x)^2
}) %>% unlist() %>% mean()

## closed form covariance
Sigma_cls <- matrix(1/A * B * 1/A / n)
```

Table 12: " Example 6 "

	$\hat{\theta}$	$\hat{\Sigma}$
Closed form	$\begin{pmatrix}1.9368\end{pmatrix}$	$\begin{pmatrix}0.0130\end{pmatrix}$
geex	$\begin{pmatrix}1.9301\end{pmatrix}$	$\begin{pmatrix}0.0130\end{pmatrix}$
Decimal of difference	$\begin{pmatrix}3\end{pmatrix}$	$\begin{pmatrix}5\end{pmatrix}$

Stefanski & Boos example 7

Example 7 illustrates calculation of sample quantiles using M-estimation. I generate a data set with 100 observations where $Y \sim N(2, 1)$. Table translates the estimating equations for two sample quantiles (median and 65th percentile) into the `R` function needed for `geex`:

Table 13: Translating math to code

$$\psi_k(Y_i, \theta) = \begin{pmatrix} 0.5 - I(Y_i \leq \theta_1) \\ 0.65 - I(Y_i \leq \theta_2) \end{pmatrix}$$

```
SB7_eefun <- function(data){
  function(theta){
    with(data,
      c(0.5  - (Y <= theta[1]),
        0.65 - (Y <= theta[2]))
    )
  }
}
```

This example is *under development*. The core functions need to be modified in order approximate the ψ and ψ_i functions. I propose to add an additional API that allows users to manipulate these functions. The example below illustate.

I begin with a nondifferentiable ψ function of a single dimension.

```
SB7_eefun <- function(data){
  function(theta){
    0.5  - (data$Y <= theta[1])
  }
}
```

Here, I modify the `eeroot` function in order to approximate the $\psi = \sum_i \psi_i$ function.

```
eeroot_mod <- function(geex_list,
                   start         = NULL,
                   rootsolver    = rootSolve::multiroot,
                   root_options  = NULL,
                   apprx_fun     = NULL,
                   apprx_options = NULL,
```

```
                   ...){

  # Create estimating equation functions per group
  psi_i <- lapply(geex_list$splitdt, function(data_i){
    geex_list$eeFUN(data = data_i, ...)
  })

  # Create psi function that sums over all ee funs
  psi <- function(theta){
    psii <- lapply(psi_i, function(f) {
      do.call(f, args = append(list(theta = theta), geex_list$ee_args))
    })
    apply(check_array(simplify2array(psii)), 1, sum)
  }

  # apprx_fun is a function that manipulates the psi function and returns a new psi function
  if(!is.null(apprx_fun)){
    psi <- do.call(apprx_fun, args = append(list(psi = psi), apprx_options))
  }
  # Find roots of psi
  rargs <- append(root_options, list(f = psi, start = start))
  do.call(rootsolver, args = rargs)
}
```

Here's an example of how it would work using `splinefun`.

```
spline_approx <- function(psi, eval_theta){
  ### Use splinefun ####
  psi2 <- Vectorize(psi)
  psi <- function(theta) splinefun(x = eval_theta, psi2(eval_theta))(theta)
  psi
}


myList <- list(eeFUN = SB7_eefun, splitdt = split(dt, f = dt$id))

root_spline1 <- eeroot_mod(
  geex_list = myList,
  start     = 2,
  apprx_fun = spline_approx,
  apprx_options = list(eval_theta = seq(1, 5, by = .5))
)

# Compare to the truth
median(dt$Y) - root_spline1$root
```

```
## [1] -0.08006065
```

```
# But notice that the basis of the spline matters too
root_spline2 <- eeroot_mod(
  geex_list = myList,
  start     = 2,
  apprx_fun = spline_approx,
  apprx_options = list(eval_theta = seq(1, 5, by = .1))
)
```

```r
# Compare to the truth
median(dt$Y) - root_spline2$root
```

```
## [1] 0.02921017
```

```r
# But notice that the basis of the spline matters too
root_spline3 <- eeroot_mod(
  geex_list = myList,
  start     = 2,
  apprx_fun = spline_approx,
  apprx_options = list(eval_theta = seq(-5, 5, by = 1))
)

# Compare to the truth
median(dt$Y) - root_spline3$root
```

```
## [1] -0.08048269
```

The above method works, but (a) it's not clear how to choose `eval_theta` and (b) it would not work with a ψ function of greater than 1 dimension since `splinefun` only takes univariate arguments.

Here's an example of how it would work using `mgcv::gam`.

```r
library(mgcv)
gam_approx <- function(psi, eval_theta){
  ### Use splinefun ####
  psi2 <- Vectorize(psi)
  Y    <- psi2(eval_theta)
  gam_basis <- gam(Y ~ s(eval_theta))
  psi <- function(theta) predict(gam_basis, newdata = data.frame(eval_theta = theta))
  psi
}

root_gam1 <- eeroot_mod(
  geex_list = myList,
  start     = 2,
  apprx_fun = gam_approx,
  apprx_options = list(eval_theta = seq(1, 5, by = .3))
)

# Compare to the truth
median(dt$Y) - root_gam1$root
```

```
## [1] -0.02318684
```

```r
# Still, the basis of the spline influences the result
root_gam2 <- eeroot_mod(
  geex_list = myList,
  start     = 2,
  apprx_fun = gam_approx,
  apprx_options = list(eval_theta = seq(1.5, 3.5, length.out = 20))
)

# Compare to the truth
median(dt$Y) - root_gam2$root
```

```
## [1] -0.05344775
```

What happens when the n increases from 100 to 10000?

```r
dt2 <- data.frame(Y = rnorm(n, mean = theta_tru, sd = sigma), id = 1:10000)
myList2 <- list(eeFUN = SB7_eefun, splitdt = split(dt2, f = dt2$id))

root_spline4 <- eeroot_mod(
  geex_list = myList2,
  start     = 2,
  apprx_fun = spline_approx,
  apprx_options = list(eval_theta = seq(0, 5, by = .1))
)

median(dt2$Y) - root_spline4$root
```

```
## [1] -0.01297451
```

```r
root_gam3 <- eeroot_mod(
  geex_list = myList2,
  start     = 2,
  apprx_fun = gam_approx,
  apprx_options = list(eval_theta = seq(0, 5, by = .1))
)

# Compare to the truth
median(dt$Y) - root_gam3$root
```

```
## [1] -0.1546932
```

In order to compute A_i, we need to approximate ψ_i, so I'll make a similar modification to `compute_matrices`.

```r
compute_matrices_mod <- function(geex_list,
                             theta,
                             numDeriv_options = list(method = 'Richardson'),
                             silent = TRUE,
                             apprx_fun     = NULL,
                             apprx_options = NULL,
                             ...){
  if(is.null(geex_list$ee_args)){
    ee_args <- NULL
  }

  with(geex_list, {

    # Create list of estimating eqn functions per unit
    psi_i <- lapply(splitdt, function(data_i){
      f <- eeFUN(data = data_i, ...)
      if(!is.null(apprx_fun)){
        f <- do.call(apprx_fun, args = append(list(psi = f), apprx_options))
      }
      f
    })

    # Compute the negative of the derivative matrix of estimating eqn functions
    # (the information matrix)
    A_i <- lapply(psi_i, function(ee){
      args <- append(list(fun = ee, x = theta), numDeriv_options)
```

```r
      val  <- do.call(numDeriv::jacobian, args = append(args, ee_args))
      -val
    })
    A_i_array <- check_array(simplify2array(A_i))
    A   <- apply(A_i_array, 1:2, sum)

    # Compute outer product of observed estimating eqns
    B_i <- lapply(psi_i, function(ee) {
      ee_val <- do.call(ee, args = append(list(theta = theta), ee_args))
      ee_val %*% t(ee_val)
    })
    B   <- apply(check_array(simplify2array(B_i)), 1:2, sum)

    list(A = A, A_i = A_i, B = B, B_i = B_i)
  })
}
```

```r
spline_matrices1 <- compute_matrices_mod(
  geex_list = myList,
  theta     = median(dt$Y),
  apprx_fun = spline_approx,
  apprx_options = list(eval_theta = seq(0, 4, by = .5))
)

compute_sigma(A = spline_matrices1$A, B = spline_matrices1$B)
```

```
##             [,1]
## [1,] 0.01925883
```

```r
gam_matrices1 <- compute_matrices_mod(
  geex_list = myList,
  theta     = median(dt$Y),
  apprx_fun = gam_approx,
  apprx_options = list(eval_theta = seq(0, 4, by = .3))
)

compute_sigma(A = gam_matrices1$A, B = gam_matrices1$B)
```

```
##             [,1]
## [1,] 0.01164746
```

```r
# V(theta) from Stefanski and Boos
((0.5 * (1 -0.5))/((dnorm(median(dt2$Y), mean = mean(dt2$Y), sd = sd(dt2$Y)))^2) )
```

```
## [1] 1.552629
```

For the the quantiles in particular, Stefanski and Boos suggest directly approximating f(theta) by a kernel density. Let's try that.

```r
dens <- density(dt$Y)
ff <- splinefun(x = dens$x, y = dens$y)
density_apprx <- function(psi){
  function(theta) ff(theta)
}

density_matrices1 <- compute_matrices_mod(
```

```r
  geex_list = myList,
  theta     = median(dt$Y),
  apprx_fun = density_apprx
)

compute_sigma(A = density_matrices1$A, B = density_matrices1$B)
```

```
##          [,1]
## [1,] 6.785466
```

What would happen if instead of approximating ψ for finding roots, we approximated ψ_i? I'm sure it would be slower, but if making same approximations for both situations (finding roots and derivatives of A_i) seems more principled plus the code would be modified in the same location in both `eeroot` and `compute_matrices`.

```r
# Put apprx_fun code in same place as compute_matrices
eeroot_mod2<- function(geex_list,
                   start         = NULL,
                   rootsolver    = rootSolve::multiroot,
                   root_options  = NULL,
                   apprx_fun     = NULL,
                   apprx_options = NULL,
                   ...){

  # Create estimating equation functions per group
  psi_i <- lapply(geex_list$splitdt, function(data_i){
      f <- geex_list$eeFUN(data = data_i, ...)
      if(!is.null(apprx_fun)){
        f <- do.call(apprx_fun, args = append(list(psi = f), apprx_options))
      }
      f
  })

  # Create psi function that sums over all ee funs
  psi <- function(theta){
    psii <- lapply(psi_i, function(f) {
      do.call(f, args = append(list(theta = theta), geex_list$ee_args))
    })
    apply(check_array(simplify2array(psii)), 1, sum)
  }
  # Find roots of psi
  rargs <- append(root_options, list(f = psi, start = start))
  do.call(rootsolver, args = rargs)
}
```

```r
# But notice that the basis of the spline matters too
root_spline5 <- eeroot_mod2(
  geex_list = myList,
  start     = 2,
  apprx_fun = spline_approx,
  apprx_options = list(eval_theta = seq(-5, 5, by = 1))
)

# Compare to the truth
median(dt$Y) - root_spline5$root
```

```
## [1] -0.08048269
```

```r
root_gam5 <- eeroot_mod2(
  geex_list = myList,
  start     = 2,
  apprx_fun = gam_approx,
  apprx_options = list(eval_theta = seq(0, 5, by = .1))
)

# Compare to the truth
median(dt$Y) - root_gam5$root
```

```
## [1] -0.03741613
```

```r
# But notice that the basis of the spline matters too
root_spline6 <- eeroot_mod2(
  geex_list = myList2,
  start     = 2,
  apprx_fun = spline_approx,
  apprx_options = list(eval_theta = seq(-5, 5, by = 1))
)

# Compare to the truth
median(dt$Y) - root_spline6$root
```

```
## [1] -0.197103
```

```r
root_gam6 <- eeroot_mod(
  geex_list = myList2,
  start     = 2,
  apprx_fun = gam_approx,
  apprx_options = list(eval_theta = seq(0, 5, by = .1))
)

# Compare to the truth
median(dt$Y) - root_gam6$root
```

```
## [1] -0.1546932
```

Not really.

Now try to approximate a multidimensional ψ function.

```r
SB7_eefun2 <- function(data){
  function(theta){
    with(data,
    c(0.25  - (Y <= theta[1]),
      0.5   - (Y <= theta[2]),
      0.75  - (Y <= theta[3]))
    )
  }
}
```

The code next is work in progress. No good, so far.

```r
gam_approx_multi <- function(psi, eval_theta1, eval_theta2, eval_theta3){
  ### Use splinefun ####
  # psi2 <- Vectorize(psi)
```

```r
  Y <- matrix(NA, nrow = length(eval_theta1), ncol = 3)
  for(i in 1:length(eval_theta1)){
      Y[i, ]  <- psi(c(eval_theta1[i], eval_theta2[i], eval_theta3[i]))
  }

  gam_data <- data.frame(
    Y1 = Y[, 1],
    Y2 = Y[, 2],
    Y3 = Y[, 3],
    x1 = eval_theta1,
    x2 = eval_theta2,
    x3 = eval_theta3
  )

  gam_basis <- gam(list(Y1 ~ s(x1),
                        Y2 ~ s(x2),
                        Y3 ~ s(x3)),
                   family = mvn(d = 3),
                   data = gam_data)
  print(gam_basis)
  psi <- function(theta) {
    predict(gam_basis, newdata = data.frame(eval_theta1 = theta[1],
                                            eval_theta2 = theta[2],
                                            eval_theta3 = theta[3]))
  }
  psi
}

SB7_eefun2(dt[1, ])(c(1, 2, 3))

myList3 <- list(eeFUN = SB7_eefun2, splitdt = split(dt, f = dt$id))

root_gam1 <- eeroot_mod(
  geex_list = myList3,
  start     = c(1, 2, 3),
  apprx_fun = gam_approx_multi,
  apprx_options = list(eval_theta1 = seq(-5, 5, by = .1),
                       eval_theta2 = seq(-5, 5, by = .1),
                       eval_theta3 = seq(-5, 5, by = .1))
)

quantile(dt$Y, probs = c(.25, .5, .75))
```

```r
estimates <- estimate_equations(eeFUN = SB7_eefun,
                                data  = dt, units = 'id',
                                findroots = FALSE,
                                roots = c(2))
```

```r
theta_cls <- c(quantile(dt$Y, 0.5), quantile(dt$Y, 0.65))
```

Stefanski & Boos example 8

Example 8 illustrates robust regression. I generate a data set with 50 observations where half of the observation have $X_i = 1$ and the others have $X_i = 0$. \$Y = \$ 0.5 + 2 $X_i + \epsilon_i$ and $\epsilon_i \sim N(0, 1)$. Table translates the estimating equations for robust regression into the R function needed for geex:

Table 14: Translating math to code

$\psi_k(Y_i, \theta) = \left(\psi_k(Y_i - \mathbf{x}_i^T \beta)\mathbf{x}_i\right)$	`SB8_eefun <- function(data){`
	`  Yi <- data$Y`
	`  xi <- model.matrix(Y ~ X, data = data)`
	`  function(theta){`
	`    r <- Yi - xi %*% theta`
	`    c(psi_k(r) %*% xi)`
	`  }`
	`}`

```
estimates <- estimate_equations(eeFUN = SB8_eefun,
                               data  = dt, units = 'id',
                               roots = c(1, 1))
```

```
m <- MASS::rlm(Y ~ X, data = dt, method = 'M')
theta_cls <- coef(m)
Sigma_cls <- vcov(m)
```

Table 15: " Example 8 "

	$\hat{\theta}$	$\hat{\Sigma}$
Closed form	$\begin{pmatrix} 0.5681 & 1.6957 \end{pmatrix}$	$\begin{pmatrix} 0.0438 & -0.0438 \\ -0.0438 & 0.0876 \end{pmatrix}$
geex	$\begin{pmatrix} 0.5705 & 1.6913 \end{pmatrix}$	$\begin{pmatrix} 0.0381 & -0.0381 \\ -0.0381 & 0.0814 \end{pmatrix}$
Decimal of difference	$\begin{pmatrix} 3 & 3 \end{pmatrix}$	$\begin{pmatrix} 3 & 3 \\ 3 & 3 \end{pmatrix}$

Stefanski & Boos example 9

Example 9 illustrates estimation of a generalized linear model. I generate a data set with 100 observations where half of the observation have $X_{1i} = 1$ and the others have $X_{1i} = 0$. $Y_i \sim Bern[\text{logit}^{-1}($ 0.5 + 2 $X_{1i}+$ 0.1 $X_{2i})]$. Table translates the estimating equations for logistic regression into the R function needed for geex:

Table 16: Translating math to code

$\psi_k(Y_i, \theta) = \left(D_i(\beta)\frac{Y_i - \mu_i(\beta)}{V_i(\beta)\tau}\right)$	`SB9_eefun <- function(data){`
	`  Yi <- data$Y`
	`  xi <- model.matrix(Y ~ X1 + X2, data = data, drop = FALSE)`
	`  function(theta){`
	`    lp <- xi %*% theta`
	`    mu <- plogis(lp)`
	`    D  <- t(xi) %*% dlogis(lp)`
	`    V  <- mu * (1 - mu)`
	`    D %*% solve(V) %*% (Yi - mu)`
	`  }`
	`}`

```r
estimates <- estimate_equations(eeFUN = SB9_eefun,
                                data  = dt, units = 'id',
                                roots = c(1, 1, 1))
```

```r
m <- glm(Y ~ X1 + X2, data = dt, family = binomial(link = 'logit'))
theta_cls <- coef(m)
Sigma_cls <- sandwich(m)
```

Table 17: " Example 9 "

	$\hat{\theta}$	$\hat{\Sigma}$
Closed form	$\begin{pmatrix} 0.3545 & 2.2680 & 0.2747 \end{pmatrix}$	$\begin{pmatrix} 0.1421 & -0.0336 & -0.1225 \\ -0.0336 & 0.4264 & -0.1087 \\ -0.1225 & -0.1087 & 0.2687 \end{pmatrix}$
geex	$\begin{pmatrix} 0.3545 & 2.2680 & 0.2747 \end{pmatrix}$	$\begin{pmatrix} 0.1421 & -0.0336 & -0.1225 \\ -0.0336 & 0.4264 & -0.1087 \\ -0.1225 & -0.1087 & 0.2687 \end{pmatrix}$
Decimal of difference	$\begin{pmatrix} 12 & 10 & 12 \end{pmatrix}$	$\begin{pmatrix} 8 & 7 & 7 \\ 7 & 7 & 6 \\ 7 & 6 & 7 \end{pmatrix}$

Stefanski & Boos example 10

Example 10 illustrates testing equality of success probablities. Table translates the estimating equations into the R function needed for geex:

Table 18: Translating math to code

$\psi_k(Y_i, n_i, \theta) = \begin{pmatrix} \frac{(Y_i - n_i\theta_2)^2}{n_i\theta_2(1-\theta_2)} - \theta_1 \\ Y_i - n_i\theta_2 \end{pmatrix}$	`SB10_eefun <- function(data){ Y <- data$ft_made n <- data$ft_attp function(theta){ p <- theta[2] c(((Y - (n * p))^2)/(n * p * (1 - p)) - theta[1], Y - n * p) } }`

```r
estimates <- estimate_equations(eeFUN = SB10_eefun,
                                data  = shaq, units = 'game',
                                roots = c(.5, .5))
```

```r
V11 <- function(p) {
  k <- length(nrow(shaq))
  sumn <- sum(shaq$ft_attp)
  sumn_inv <- sum(1/shaq$ft_attp)
  term2_n <- 1 - (6 * p) + (6 * p^2)
  term2_d <- p * (1 - p)
  term2 <- term2_n/term2_d
  print(term2)
  term3 <- ((1 - 2 * p)^2)/( (sumn/k) * p * (1 - p))
  print(term3)
  2 + (term2 * (1/k) * sumn_inv)  - term3
```

```r
}

### ???? I keep getting a negative value for V11

p_tilde <- sum(shaq$ft_made)/sum(shaq$ft_attp)
V <- V11(.45)
```

```
## [1] -1.959596
## [1] 0.0001365001
```

```r
V
```

```
## [1] -2.497375
```

```r
pnorm(estimates$parameters[1], mean = 1, sd = sqrt(V))
```

```
## Warning in sqrt(V): NaNs produced
```

```
## [1] NaN
```

Small Sample Corrections of Fay (2001)

Bias correction

$$H_i = \{1 - min(b, \{A_i A\}_{jj})\}^{-1/2}$$

Where b is a constant chosen by the analyst. Fay lets $b = 0.75$. Note that H_i is a diagonal matrix.

$$B_i^{bc} = H_i \psi_i \psi_i^T H_i$$

$$B^{bc} = \sum_{i=1}^{m} B_i^{bc}$$

$$\Sigma^{bc} = A^{-1} B^{bc} \{A^{-1}\}^T$$

Degrees of Freedom corrections

Let L be the contrast of interest (e.g.) $(0, \ldots, 0, 1, -1)$ for a causal difference when the last two elements of the estimating equations are the counterfactual means.

$$\mathcal{I} = [I_p \cdots I_p]$$

where I_p is a $p \times p$ identity matrix.

$$G = I_{pm} - \begin{bmatrix} A_1^{bc} \\ \vdots \\ A_m \end{bmatrix} A^{-1} \mathcal{I}$$

$$M = diag\{H_i A^{-1} L L^T (A^{-1})^T H_i\}$$

$$C = G^T M G$$

$$w_i = L^T \left[\left\{ \sum_{j \neq i} A_i \right\}^{-1} - A^{-1} \right] L$$

$$\bar{w} = \sum_{i=1}^{m} w_i$$

$$A_i^{bc} = \frac{w_i}{\bar{w}} B^{bc}$$

$$\hat{df}_1 = \frac{\{Tr(diag(A_i)C)\}^2}{Tr(diag(A_i)Cdiag(A_i)C)}$$

$$\hat{df}_2 = \frac{\left\{Tr(diag(A_i^{bc})C)\right\}^2}{Tr(diag(A_i^{bc})Cdiag(A_i^{bc})C)}$$