Rinterference Vignette

Bradley Saul

November 3, 2014

This is a VERY brief description of how to use the `rinterference` package.

```
library(rinterference)
```

```
## Loading required package: numDeriv
## Loading required package: lme4
## Loading required package: Matrix
## Loading required package: Rcpp
```

I have included a simple dataset generated from `rinterferenceSim` package: 3000 units, 250 groups, 2 covariates, 21 allocations (0 to 1 by 0.05), using the parameters in the code below. See Perez 2014 for details on how the simulations were structured.

```
alphaz = seq(0, 1, by=.05) # needed to compute truth
theta.base <- c(.5, -0.788, -2.953, -0.098, -0.145, 0.351)
theta.sim <- c(0.2727, -0.0387, .2719, 1.0859)
sample_sim <- sim_interference(n = 3000, N = 250, nsims = 1,
                               base.parameters = theta.base,
                               parameters = theta.sim,
                               alphas = alphaz)
sample_data <- sample_sim$sims[[1]]
#save(sample_data, file = "data/sample_data.rda")
```

Here's what the data looks like:

```
head(sample_data)
```

```
##   y     X1    X2 A B group
## 1 1 5.3607 1.716 0 0     1
## 2 0 0.1965 1.731 0 1     1
## 3 0 0.4846 1.770 1 1     1
## 4 0 0.8013 1.716 0 1     1
## 5 0 2.1427 1.772 1 1     1
## 6 0 1.2861 1.716 0 1     1
```

y is the outcome. X1 and X2 are covariates. A is the randomized treatment indicator. B is an indicator of participation in the trial. Group is obvious.

The `rinterference` package has two main steps. First is to calculate all the pieces of the IPW estimators. `run_interference()` does this. Using these pieces, you want to compute the effects and associated variances. `calc_effect()` does this. `direct_effect()`, `indirect_effect()`, `total_effect()` and `overall_effect()` are convenient wrappers for `calc_effect` to get.

Here's a quick sample of `run_interference()` based on the data provided:

```
sample_run <- run_interference(
  data = sample_data, # name of the data frame
  groups = 'group',   # quoted string with group variable
  outcome = 'y',      # quoted string with outcome variable
  predictors = c('X1', 'X2'), # vector of strings with predcitor variables
  # At this time at least one predictor must be defined
  treatment = 'A', # quote string of treatment variable
  propensityB = 'B', # quoted string for 'first stage' variable. OPTIONAL.
  allocations = c(.3, .45, .6))
```

```
## [1] "Run_interference complete"
```

Now we can compute the effects:

```
#Compute DE(.3)
direct_effect(sample_run, .3)
```

```
##    point variance     ll   ul
## 1 0.3418 0.003635 0.2237 0.46
```

```
#Compute IE(.3, .6)
indirect_effect(sample_run, .3, .6)
```

```
##    point variance     ll     ul
## 1 0.4275  0.00411 0.3019 0.5532
```

```
#Compute IE(.3, .9)
#Oops we didn't include that allocation scheme in run_interence
```

Let's say we're running simulations and we know the true parameters. In the sample above, the parameters were estimated by `glmer`, but we can tell `run_interference` that we know the truth via the `known_params` argument. Be careful that the number of parameters line up with the number of parameters expected by the integrand function. In the case of `logit_integrand`, the default integrand function, it expects an intercept parameter, one for each covariate, and one for the random effect.

```
sample_run_truth <- run_interference(
  data = sample_data, # name of the data frame
  groups = 'group',   # quoted string with group variable
  outcome = 'y',      # quoted string with outcome variable
  predictors = c('X1', 'X2'), # vector of strings with predcitor variables
  # At this time at least one predictor must be defined
  treatment = 'A', # quote string of treatment variable
  propensityB = 'B', # quoted string for 'first stage' variable. OPTIONAL.
  known_params = c(0.2727, -0.0387, .2719, 1.0859),
  allocations = c(.3, .45, .6))
```

```
## [1] "Run_interference complete"
```

Now we can compute the effects:

```r
#Compute DE(.3)
direct_effect(sample_run_truth, .3)
```

```
##    point variance     ll     ul
## 1 0.3907 0.003694 0.2716 0.5098
```

```r
#Compute IE(.3, .6)
indirect_effect(sample_run_truth, .3, .6)
```

```
##    point variance     ll     ul
## 1 0.5142 0.004063 0.3892 0.6391
```

You can define your own integrand as a function and pass it as an argument to `run_interference` if desired. It would be helpful to look at the code for `logit_integrand` to get started.