Reproducibility Code for "The Residual Randomization Method for Invariant Inference"

This document describes the code implementing all simulations and applications of residual randomization in the paper. Every section of the paper corresponds to an `R` script. Typically, every script has the full simulation code, and also some separate code for quick reproducibility.

1 Section 4.1, Table 1 (Simulation with one-way clustering)

In this example, we use `Section_4.1_clustered.R`.

- `DGP()` – Generates data.
- `main_sim()` – Produces Table 1.

This code is computationally intensive and it is more practical to run it on a cluster. To generate some key results from Table 1 much faster, we can use the following code:

```
> source("Section_4.1_clustered.R")
> quick_examples()

"Example with log-normal Xg, no cluster effects, homoskedastic error."
....
[1] "j= 990  /  999"
       OLS ClusterOLS    RR-sign  RR-double
     4.343      8.586      4.646      4.343

...
"Example with log-normal Xg, no cluster effects, heteroskedastic error."
...
"j= 990  /  999"
       OLS ClusterOLS    RR-sign  RR-double
    49.192     12.727      5.455     44.747
```

As in the paper (Table 1) we see that OLS and cluster robust OLS over-reject. Residual randomization with cluster sign symmetry nearly maintains the correct level. The test with double invariance over-rejects because the assumption of within-cluster exchangeability is not true.

2 Section 4.2, Table 2 (Simulation on dyadic regression)

In this example, we use `Sec_4.2_dyadic.R`.

- `DGP()` – Generates data.
- `main_sim()` – Produces Table 2.

In the paper, we used a cluster with 400 nodes, where each node run the main simulation for 50 replications. For some quick results, the following code reproduces the part of the table with $(\varepsilon, X) =$ (normal, lognormal) and $n = 190$.

```
> source("Section_4.2_dyadic.R")
> quick_examples()

"Example with log-normal Xg, no cluster effects, homoskedastic error."
....
> quick_examples()
           HC2           2way random-effects        dyad-CL      RR-dyadic
           100              0              0              0              0
           HC2           2way random-effects        dyad-CL      RR-dyadic
           100              0              0              0              0


...
       47.959         11.224          7.143         11.224          7.143
           HC2           2way random-effects        dyad-CL      RR-dyadic
        47.475         11.111          7.071         11.111          7.071
           HC2           2way random-effects        dyad-CL      RR-dyadic
            48             11              7             11              7
```

As in the paper, we see that the residual randomization performs better than alternatives, and nearly maintains the correct level.

3 Section 4.3, Table 3 (Behrens–Fisher example)

In this example, we use `Sec_4.3_BehrensFisher.R`.

- `DGP()` – Generates data.
- `main_sim()` – Produces Table 2.
- `test_exact()` – Implementation of the exact randomization test of Section 4.3.

In the paper, we used a cluster with 400 nodes, where each node run the main simulation for 250 replications. For some quick results, the following code will reproduce the part of Table 3 with $(\varepsilon, X) =$ (normal, lognormal) and $n = 190$.

```
> source("Sec_4.3_BehrensFisher.R")
> quick_examples()
[1] "iter= 100"
       BM wild boot     exact
        1         0         7
[1] "iter= 200"
       BM wild boot     exact
      0.5       0.0       5.0
...
1] "iter= 1000"
       BM wild boot     exact
      1.6       0.0       5.1
```

We see that the exact test nearly achieves the nominal level, while the alternatives significantly under-reject (they are under-powered).

4 Section 5.1, Table 4 (Hormone data example)

In this example we use `Sec_5.1_hormone.R`.

- `main_sim()` produces Table 4.
- `plot_pvals()` produces Figure 1.

5 Section 5.2 (Honeybees example)

In this example we can simple source `Sec_5.2_honeybees.R`. This will print the randomization-based 95% CI.

6 Section 5.3, Table 5 (Trade example)

In this example we use `Sec_5.3_trade.R`.

- `main_sim()` – produces the lower-part of Table 5 referring to the dyadic invariances.
- `main_sim_no_dyadic()` – produces the middle-part of Table 5 pertaining to non-dyadic invariances.
- `replicate_rose_cameron()` – replicates some results in previous papers.
- `sensitivity_analysis()` – presents additional analysis indicating that β_{CU} may be zero.

We run the full simulations in a cluster. For a quick illustration, the following code shows that $beta_{\text{CU}} = 0$ is rejected by simple invariance structures (e.g., full exchangeability). The same hypothesis is not rejected by dyadic exchangeability. This invariance is more plausible in this setting.

```
> quick_examples()

[1] " Centering..."
[1] "> Generating decomposition..."
[1] "> Total observed country pairs =  9230"
[1] ">>> Applying Filter by  comcont"
[1] ">>> Total observed country pairs =  2194"
[1] ">>> Total cliques= 13 total countries= 81 ( 64.29 %)"
[1] "> Statistics on clique size:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   2.000   3.000   6.231   8.000  18.000

# analyze trade data under full exchangeability invariance
>   RRI_trade_no_structure(beta_0=0, out, type = "perm", num_r=3000)

[1] 0.03805093  # pval< 0.05, simple exchangeability rejects Beta_CU=0

# analyze data under dyadic exchangeability
>   RRI_trade(beta_0=0, out, num_r = 3000)

[1] 0.9227976    #pval > 0.05 -- dyadic exchangeability does not reject Beta_CU=0
```

7 Section 6.1, Table 6 (Joint hypotheses)

We use script `Sec_6.1_jointH0.R`.

- `main_sim()` – produces the residual randomization part of Table 6 in the paper.

8 Section 6.2, Table 7 (Autocorrelated errors)

We use script `Sec_6.2_hac.R`.

- `DGP()` — the data generating process.
- `main_sim()` — produces Table 7 in the paper for a given ρ (see start of script).

In the paper, we used a cluster with 400 nodes, where each node run the main simulation for 250 replications. For some quick results, the following code reproduces the part of Table 7 with $\rho = 0.8$, autocorrelated normal ε and autocorrelated normal x.

```
> source("Sec_6.2_hac.R")   # set rho=0.8
> quick_examples()
[1] "samples= 50 Rejection rates %="
          OLS           HAC RR-reflection
           24            12             8


.....
[1] "samples= 1000 Rejection rates %="
          OLS           HAC RR-reflection
         33.6          17.6           3.9
```

As in the paper, we see that OLS significantly over-rejects. HAC errors are better but still over-reject. Residual randomization maintains the correct level, but it rejects a bit lower than the nominal level.