Randomization Inference in Regression Models
`RRI` `R` Package

Panos Toulis*
University of Chicago
Booth School of Business

DRAFT (February 9, 2021)
Latest version here

Abstract

Residual randomization is a method for testing and inference in regression models based on invariance assumptions on the errors; e.g., inference assuming only exchangeability of errors, or sign symmetry, or both. Compared to standard normal OLS these assumptions are weaker, which adds robustness. Compared to bootstrap, residual randomization is more flexible, it does not rely on asymptotic normality, and addresses the inference problem in a unified way. In this technical report, we describe residual randomization in practice through the package `RRI`. This report is a companion to the main paper [7] that contains details on theory and methods.

* *email:* panos.toulis@chicagobooth.edu

Contents

1 Introduction

Consider the linear regression model of the form:

$$y = X\beta + \varepsilon,$$

where $y, \varepsilon \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times p}$, $\beta \in \mathbb{R}^p$ are the outcomes, errors, covariates, and parameters, respectively. The goal is to perform inference on β relying on few assumptions. Throughout we use plain OLS, $\hat{\beta}$, as our baseline estimator. To do inference on β we generally rely on normal OLS theory or bootstrap [1, 2, 3]. These approaches, however, rely on some implicit form of exhangeability in the errors. This requirement has created problems in practice, for example, when errors are heterogeneous or autocorrelated. Furthermore, most methods generally rely on asymptotic normality, which is sometimes a strong assumption, and certainly implausible in small-sample problems.

This technical report describes an alternative way for inference through *residual randomization*. The fundamental idea is to base inference on assuming an *invariance* of the errors:

$$\varepsilon \overset{d}{=} \mathsf{g}\varepsilon,$$

where $\mathsf{g} \in \mathcal{G}$ is a $n \times n$ matrix, and $\mathcal{G}$ is an algebraic group of operations such as permutations or random signs. For example, suppose the errors are exchangeable so that

$$(\varepsilon_1, \ldots, \varepsilon_n) \overset{d}{=} (\varepsilon_{\pi(1)}, \ldots, \varepsilon_{\pi(n)}),$$

where π denotes a permutation of n elements. In this case, $\mathsf{g} = \sum_{i=1}^n 1_i 1'_{\pi(i)}$, where 1_i is a n-length vector that is zero except at i-th element. The second key component of residual randomization is to use a statistic T_n such that $T_n \overset{H_0}{=} t_n(\varepsilon)$ for some function t_n under some null of interest H_0. In principle, testing (and inference) are possible by comparing T_n with values $\{t_n(\mathsf{g}\varepsilon) : \mathsf{g} \in \mathcal{G}\}$. In practice, we use residuals in t_n because the true errors are unknown.

For example, suppose that

$$y_i = \beta_1 + \beta_2 x_i + \varepsilon_i,$$

and we want to test $H_0 : \beta_2 = 0$. Consider the test statistic $T_n = \hat{\beta}_2$. The standard approach is to use some form of asymptotic normality for T_n to test H_0. But this is not necessary. Note that we can write $T_n = a'(X^\top X)^{-1} X^\top y$, where $a = (0, 1)$, and X is the $n \times 2$ covariate matrix with i-th row equal to $(1, x_i)$. If H_0 is true then $T_n = a'(X^\top X)^{-1} X^\top \varepsilon \equiv t_n(\varepsilon)$. What to compare T_n with? To test H_0, we would like to compare T_n with values $\{t_n(\pi\varepsilon) : \pi \in \mathsf{S}_n\}$, i.e., values of t_n for various permutations of ε (S_n is the symmetric n-group). But the errors are unknown, so we rely on the residuals. Thus, we

1. calculate restricted residuals $\hat{\varepsilon}_i^{\mathrm{r}} = y_i - \hat{\beta}_1^{\mathrm{r}}$, where $\hat{\beta}_1^{\mathrm{r}}$ is the OLS estimate β_1 with fixed $\beta_2 = 0$ (here, $\hat{\beta}_1^{\mathrm{r}} = \bar{y}$)
2. compare T_n with $\{t_n(\pi\hat{\varepsilon^{\mathrm{r}}})\}$ to get a p-value.

A confidence interval for β_2 is obtained by test inversion. Specifically, we can test a series of hypotheses $\ldots, H_0 : \beta_2 = -10, \ldots, H_0 : \beta_2 = 10, \ldots$ and report those endpoints for which the p-value exceeds the desired level. This is, in a nutshell, the basic residual randomization method.

We structure this paper as follows. In Section 2 we consider the basic testing problem solved by residual randomization, and present the related functionality in the `RRI` package. In Section 3 we consider the problem of inference. In Section 4 we illustrate on a real data example, and in Section 5 we briefly discuss more complex error structures, such as autocorrelated errors.

2 Testing using RRI

The main functionality in the RRI package is to test the following linear hypothesis:

$$H_0 : \lambda'\beta = \lambda_1\beta_1 + \ldots \lambda_p\beta_p = \lambda_0. \tag{2.1}$$

This includes the standard significance tests $\beta_1 = 0, \ldots, \beta_p = 0$. Inversion of the tests leads to confidence intervals (see next). There are two main functions in the package for testing H_0:

- `rrtest(model, g_invar, ...)`. Testing under generic invariance supplied by `g_invar`.
- `rtest_clust(model, type, clustering, ...)`. Testing under assumption of invariance of the specified `type` within clusters defined by `clustering`.

The key objects here are:

- `model`. This a list of `y, X, lam, lam0` that contains the data `y, X` and the coefficients λ, λ_0 as defined above.
- `g_invar`. The invariance function $\mathtt{g} : \mathbb{R}^n \to \mathbb{R}^n$ as defined earlier; e.g., `g_invar= function(e) sample(e)` encodes exchangeability.
- `clustering`. A list that splits the datapoint indexes $\{1, \ldots, n\}$ in clusters; e.g., `list(1:100)` defines only one cluster, whereas with `as.list(1:100)` every index has its own cluster.

2.1 Example

Consider a simple linear regression with two parameters:

```
> library(RRI)
> set.seed(123); n = 50
> X = cbind(rep(1, n), runif(n))
> beta = c(0, 0)
> y = X %*% beta + rnorm(n)
```

We wish to test $H_0 : \beta_2 = 0$. With standard OLS we can do this as follows:

```
> min(confint(fit)) > 0 | max(confint(fit)) < 0 # OLS t-stat reject?
[1] FALSE
```

The OLS test relies on an assumption that the errors are i.i.d. normal. However, we can use residual randomization to do inference only under an assumption of exchangeability (or some other form of invariance), without requiring independence or normality.

To do such inference using RRI we first need to consider what kind of invariance are we willing to posit. Suppose we only want to assume that errors are exchangeable. Then, we proceed as follows:

```
> model = list(y=y, X=X, lam=c(0, 1), lam0=0)  # test H0: beta2=0
> g_invar = function(e) sample(e)
> rrtest(model, g_invar)
[1] 0
```

The function `rrtest` can encode any generic residual randomization test with an invariance defined by `g_invar`. The test above is equivalent to the following code:

```
> rrtest_clust(model, type="perm")
[1] 0
```

The function `rrtest_clust` is more specific than `rrtest` in the sense it assumes that the invariance of the specified `type` holds in clusters. Suppose, for example, that the datapoints come from two hypothetical clusters (say split in half). Then we could do inference as follows:

```
> cl = list(1:25, 26:50)   # two clusters
> rrtest_clust(model, type="perm", clustering=cl)
[1] 0
```

In this code the test operates under the assumption that the errors are exchangeable only within the two specified clusters. Such flexibility is a key feature of `RRI`. In terms of invariances, there are three types:

- `type = "perm"`. Errors are exchangeable within clusters only. The default clustering is where all datapoints are in one single cluster (equivalent to plain exchangeability)
- `type = "sign"`. Errors are sign symmetric on the cluster level. The default clustering is where each datapoint has its own cluster (similar to "wild bootstrap")
- `type = "double"`. Errors are both exchangeable within clusters and sign-symmetric across. The default clustering is where each datapoint has its own cluster.

Let's consider one more simulated example with heteroskedastic variance.

```
> set.seed(123); n = 200
> X = cbind(rep(1, n), 1:n/n)
> beta = c(-1, 0.2)
> ind = c(rep(0, 0.9*n), rep(1, .1*n))  # cluster indicator
> y = X %*% beta + rnorm(n, sd= (1-ind) * 0.1 + ind * 5) # heteroskedastic
```

This example is set up such that there are two clusters, and error is heteroskedastic across clusters. Because the variance in one cluster is much larger than the other, normal OLS is misled to think that β_2 is not significant:

```
> confint(lm(y ~ X + 0))  # normal OLS does not reject H0: beta2 = 0
        2.5 %     97.5 %
X1 -1.2681800 -0.4711499
X2 -0.8789845  0.4963558
```

In fact, OLS inference goes the opposite direction since true $\beta_2 = 0.2$ and is positive. We can now do inference assuming within-cluster exchangeability (which here is a correct assumption) using `RRI`:

```
> cl = list(which(ind==0), which(ind==1))
> model = list(y=y, X=X, lam=c(0, 1), lam0=0)
> rrtest_clust(model, "perm", cl)  # errors are exchangeable within clusters
[1] 1
```

We see that residual randomization correctly rejects the null.

3 Inference using RRI

As mentioned before we can invert the tests of the previous section to produce confidence intervals. There are two main functions in RRI for that:

- `rrinf(y, X, g_invar, ...)`. Inference under generic invariance supplied by `g_invar`.
- `rrinf_clust(y, X, type, clustering, ...)`. Inference under invariance of the specified `type` within clusters defined by `clustering`.

The objects in the arguments were explained in Section 2.

3.1 Example

Let's start with a simple simulated example to illustrate inference with RRI:

```
> set.seed(123); n = 100
> X = cbind(rep(1, n), runif(n))
> beta = c(-1, 1)
> y = X %*% beta + rnorm(n)
> confint(lm(y ~ X + 0))
        2.5 %     97.5 %
X1 -1.3980246 -0.6198951
X2  0.2318356  1.5885017
```

To produce 95% (by default) confidence intervals for β with RRI we have to decide on the underlying invariance. Suppose we want to assume plain exchangeability. We can use the generic `rrinf` that expects a user-defined invariance function `g_invar`:

```
> g_invar = function(e) sample(e)  # Assume exchangeable errors.
> rrinf(y, X, g_invar)
     midpoint estimate       2.5%      97.5%
[1,]        -1.0089599 -1.4276536 -0.5902661
[2,]         0.9449301  0.2497005  1.6401597
```

We see that this is similar to OLS. This is not surprising since the errors are indeed iid normal and there are no clusters.

To illustrate the difference between the methods let's go back to the cluster example of the previous section. We initialize as follows:

```
> set.seed(123); n = 200
> X = cbind(rep(1, n), 1:n/n)
> beta = c(-1, 0.2)
> ind = c(rep(0, 0.9*n), rep(1, .1*n))  # cluster indicator
> y = X %*% beta + rnorm(n, sd= (1-ind) * 0.1 + ind * 5) # heteroskedastic
```

We can do `plot(y, X[, 2])` to see the cluster heteroskedastic effect. Here are the OLS intervals:

```
> confint(lm(y ~ X + 0))
        2.5 %     97.5 %
X1 -1.2681800 -0.4711499
X2 -0.8789845  0.4963558
```

We see that the OLS confidence interval for β_2 is imprecise in the sense that it is centered on negative values even though $\beta_2 = 0.2$. We can get confidence intervals from `RRI` assuming exchangeability within clusters as follows:

```
> cl = list(which(ind==0), which(ind==1))  #  define the clustering
> rrinf_clust(y, X, "perm", cl)  # improved CI through clustered errors
     midpoint estimate        2.5%      97.5%
[1,]        -1.0957261 -1.13682813 -1.0546241
[2,]         0.1278477  0.05692279  0.1987726
```

We see a clear improvement now as the confidence intervals are better centered.

4 Data Example

Now we work on a real-world data example. We use the `Duncan` dataset[1] that contains data from 1950 on income and education levels across various occupations, and information on the perceived "prestige" of every occupation. Since the data are proportions we will make a logit transformation:

```
> data("Duncan")
> for(j in 2:4) { Duncan[, j] = log(Duncan[, j] / (100.5 - Duncan[, j])) }
> n = nrow(Duncan)
> y = Duncan$prestige; X = cbind(rep(1, n), Duncan$income, Duncan$education)
> colnames(X) = c("intercept", colnames(Duncan[, 2:3]))
```

Here are the OLS confidence intervals:

```
> ols = lm(prestige ~ income + education, data=Duncan)
> confint(ols)
                 2.5 %    97.5 %
(Intercept) -0.2158245 0.3722788
income       0.4600268 1.0283294
education    0.2481290 0.6509819
```

We see that both income and education are significant for prestige.

Next, we can use (residual) bootstrap to try a nonparametric approach to inference—see Appendix A for a description of bootstrap methods in regression, and [4] for some examples on this dataset. We use the `boot` function in the `boot` R package:

```
> yhat = fitted(ols)
> e = residuals(ols)
> boot.fn <- function(data, ind){
+   yboot = yhat + e[ind]
+   coef(lm(yboot ~ X + 0))
+ }
> out = boot(Duncan, boot.fn, R = 999, maxit=100)
> out
      original       bias    std. error
t1* 0.07822712 0.0059865963  0.14424626
t2* 0.74417811 0.0020797617  0.13613217
t3* 0.44955543 0.0005607954  0.09723125
```

[1] https://vincentarelbundock.github.io/Rdatasets/doc/car/Duncan.html

```
> confint(out)
Bootstrap bca confidence intervals

       2.5 %    97.5 %
1 -0.2236692 0.3747571
2  0.4873804 1.0099769
3  0.2721077 0.6562315
```

We see that the bootstrap package reports the original OLS estimates and the bootstrap standard errors. Here, the bootstrap errors and intervals are quite similar to OLS. Note that the `boot` function performs inference assuming exhangeability since it is permuting the datapoint indexes. It is therefore less flexible than residual randomization.

Now we can use use the `RRI` package for residual randomization inference. We start with an assumption of exchangeability (as in bootstrap):

```
> rrinf_clust(y, X, type="perm", num_R = 5000)
          midpoint estimate       2.5%     97.5%
intercept        0.07822712 -0.3811251 0.5375793
income           0.74417811  0.5007566 0.9875997
education        0.44955543  0.2567006 0.6424103
```

The main difference with OLS and bootstrap is that randomization reports a wider interval for the intercept. However, it reports similar errors for the other parameters. To produce intervals under error sign symmetry we just need to change the `type`:

```
> rrinf_clust(y, X, type="sign", num_R=5000)
          midpoint estimate       2.5%     97.5%
intercept         0.1078627 -0.1736757 0.3894012
income            0.8157727  0.4434809 1.1880645
education         0.4394052  0.1754985 0.7033118
```

The intervals here are even wider but not qualitatively different than before. This adds robustness and a capability for useful sensitivity analysis through "plug-and-play" of different invariances — this is a key feature of residual randomization.

4.1 Clustered Errors

In the Duncan dataset there is one additional variable for the type of occupation: professional (`prof`), blue collar (`bc`), and white collar (`wc`). Suppose we think that it is implausible that errors are exchangeable across these types, but they are plausibly exchangeable *within* a type. Residual randomization offers a simple way to deal with this more complex cluster structure.

To illustrate suppose that errors are exchangeable only within the occupation type cluster:

```
> cl = sapply(levels(Duncan$type), function(t) which(Duncan$type==t))
> rrinf_clust(y, X, type = "perm", cl, num_R=5000)
          midpoint estimate       2.5%     97.5%
intercept        -0.2181291 -0.5885744 0.1523162
income            0.5866700  0.3862052 0.7871349
education         0.5307575  0.4191047 0.6424103
```

The intervals are now considerably sharper compared to unclustered exchangeability. This is reasonable since we put more structure into the inference.

5 Other Error Structures

Residual randomization can handle even more complex types of error invariance. For example, we can use a certain reflective symmetry of time series to do inference with autocorrelated errors [7, Section 6.1], or use penalized estimators within residual randomization for high-dimensional regression [7, Section 6.2]. There are empirical evaluations showing that residual randomization can outperform state-of-art methods in these settings as well. The caveat is that there are no theoretical guarantees (as of now) regarding validity. We will focus on such more complex error structures in future versions of this technical report.

6 Concluding remarks

Residual randomization is a new method for testing and inference in regression models with complex error structure. The method works in a variety of error structures, particularly clustered errors. The package `RRI` provides core residual randomization functionality through the functions `rrtest, rrtest_clust, rrinf, rrinf_clust`. Using the package we can consider different invariances for the same problem, and such "plug-and-play" format makes residual randomization more flexible and data-adaptive than bootstrap.

References

[1] B Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, pages 1–26, 1979.

[2] Bradley Efron. Nonparametric estimates of standard error: the jackknife, the bootstrap and other methods. *Biometrika*, 68(3):589–599, 1981.

[3] Bradley Efron and Robert Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75, 1986.

[4] John Fox. Bootstrapping regression models. *An R and S-PLUS Companion to Applied Regression: A Web Appendix to the Book. Sage, Thousand Oaks, CA. URL http://cran. r-project. org/doc/contrib/Fox-Companion/appendix-bootstrapping. pdf*, 2002.

[5] David A Freedman et al. Bootstrapping regression models. *The Annals of Statistics*, 9(6):1218–1228, 1981.

[6] SC Peters and DA Freedman. Some notes on the bootstrap in regression problems. *Journal of Business & Economic Statistics*, 2(4):406–09, 1984.

[7] Panos Toulis. Life after bootstrap: residual randomization inference in regression models. 2019.

A Bootstrap

The pairs bootstrap resamples y and (the rows of) X to create a new dataset, (y^*, X^*), and then obtains a new OLS estimate, $\hat{\beta}^*$. The residual bootstrap [5, 6] operates conditionally on X, calculates residuals $\hat{\varepsilon} = y - X\hat{\beta}$ and defines the bootstrap dataset as (y^*, X) to obtain $\hat{\beta}^*$, where $y^* = X\hat{\beta} + \hat{\varepsilon}^*$, and $\hat{\varepsilon}^*$ is a sample with replacement from $\hat{\varepsilon}$.

B Details on Testing Algorithm

Here, we present in detail the main testing algorithm implemented in the `RRI` package (function `rrtest`). As mentioned before, the basic test of the `RRI` package is the hypothesis:

$$H_0 : \lambda'\beta = \lambda_1\beta_1 + \ldots \lambda_p\beta_p = \lambda_0.$$

To test H_0 we proceed as follows (below we use the `model` list elements without the specifier `model$..` for simplicity).

1. Define function $t_n(e) = \lambda'(X^\top X)^{-1}X^\top e$.
2. Estimate OLS, $\hat{\beta}$, by regressing y on X:
3. Calculate observed test statistic $T_n = \lambda'\hat{\beta} - \lambda_0$: Note that under H_0 we have

$$T_n = \lambda'\hat{\beta} - \lambda_0 = t_n(\varepsilon) + \lambda'\beta - \lambda_0 \overset{H_0}{=} t_n(\varepsilon).$$

4. Calculate the restricted residuals, $\hat{\varepsilon^{\mathrm{r}}} = y - X\hat{\beta}^{\mathrm{r}}$, where

$$\hat{\beta}^{\mathrm{r}} = \arg \min_{\beta:\lambda'\beta=\lambda_0} ||y - X\beta||^2$$

   are the OLS estimated restricted under the null. See `restricted_OLS_c` function in the `RRI` package:

5. *Repeat:* Transform the residuals according to random `g` from $\mathcal{G}$ and calculate $t_n(\mathtt{g}\hat{\varepsilon^{\mathrm{r}}})$.

   This forms the randomization distribution of the test statistic.

6. Compare T_n with the randomization distribution, e.g., through a two-sided p-value. We can reject at α-level if the p-value is smaller than $(1 - \alpha)/2$.