Replication: The Structure of Inequality and the Politics of Redistribution

Filippo Teoldi, Zara Riaz and Julian Gerez

October 23rd, 2018

Design declaration

First we start by loading in the `DeclareDesign` package and defining the elements of the design.

- `declare_population` refers to the sample size of the study. The study concerns country-year units. In this case, there are 858 observations.
- `delcare_potential_oucomes` refers to

```
library('DeclareDesign')

design <-
  declare_population(N = 858) +
  declare_potential_outcomes() +
  declare_estimand() +
  declare_assignment() +
  declare_estimator()
```

Replication

Data cleaning

First we open the dataset with the `haven` package, which allows us to open .dta files.

```
library('haven')
directory <- "/Users/juliangerez/Google Drive/Semester_Fall_2018/Political Economy of Development/Lupu-
data <- read_dta(paste0(directory, "LupPon_APSR.dta"))
```

Next, the authors redefine invert disproportionality measures, `disp_gall` as such.

```
data$disp_gall <- data$disp_gall*-1
```

Then the variables female participation, `fempar`, and annual net union density, `union` are multiplied by 100 so that they are rescaled.

```
data$fempar <- data$fempar*100
data$union <- data$union*100
```

The variables `pjoint` and `disp_gall`, are partisanship and disproportionality, respectively. These are standardized from [0,1]. To do so, we are defining a function, `range01`, which standardizes the range of a variable such that it takes on values from 0 to 1.

```
range01 <- function(x){(x-min(x))/(max(x)-min(x))}

data$stdpjoint <- range01(data$pjoint)
data$stdpdisp_gall <- range01(data$disp_gall)
```

Next, we interpolate missing values. But before we can do so, first we define as all the variables that need to be interpolated: `pratio9050`, `pratio5010`, `pratio9050s`, `pratio5010s`, `pforeign`, and `pvoc`. However, we need to interpolate missing values for each country, not for the dataset as a whole. So we write a loop to define the object `data_countries` as a list of the data (with these aforementioned new variables) subsetted by each country.

```
data$pratio9050 <- NA
data$pratio5010 <- NA
data$pratio9050s <- NA
data$pratio5010s <- NA
data$pforeign <- NA
data$pvoc <- NA

data_countries <- lapply(unique(data$country), function(x)
  subset(data, data$country==x)
)
```

At this point, we can interpolate missing values for each variable. The `zoo` package allows use to use the function `na.approx` to linearly interpolate missing values. We use a set of loops that interpolates missing values indexed for each country, `i`, in our list of `data.frames`, `data_countries`, *for each variable.*[1] Finally, we can use `rbind` to bind this new list into a single `data.frame`, and remove our list of `data.frames`.

```
library('zoo')

# Interpolate pratio9050 (data_countries[[i]][,24]) using ratio9050 (data_countries[[i]][,5])

for (i in 1:length(data_countries)){
data_countries[[i]][,24] <- na.approx(data_countries[[i]][,5], x = index(data_countries[[i]][,3], data_
}

# Interpolate pratio5010 (data_countries[[i]][,25]) using ratio5010 (data_countries[[i]][,6])

for (i in 1:length(data_countries)){
data_countries[[i]][,25] <- na.approx(data_countries[[i]][,6], x = index(data_countries[[i]][,3], data_
}

# Interpolate pratio9050s (data_countries[[i]][,26]) using ratio9050s (data_countries[[i]][,7])

for (i in 1:length(data_countries)){
data_countries[[i]][,26] <- na.approx(data_countries[[i]][,7], x = index(data_countries[[i]][,3], data_
}

# Interpolate pratio5010s (data_countries[[i]][,27]) using ratio9050 (data_countries[[i]][,8])

for (i in 1:length(data_countries)){
data_countries[[i]][,27] <- na.approx(data_countries[[i]][,8], x = index(data_countries[[i]][,3], data_
}

# Interpolate pforeign (data_countries[[i]][,28]) using foreign (data_countries[[i]][,16])
```

[1] This is what `data_countries[[i]][,y>23]` refers to, where `i` is each country and `y` represents of the new variables. The 24th column is `pratio9050`, the 25th column `pratio5010`, and so on. Each of these are interpolated using the original variables, which is represented in `data_countries[[i]][,z>5]`, where `z` represents the original variables corresponding the new variables (i.e. `pratio9050` is interpolated using `ratio9050`, which is in the 5th column, and so on). Note that the index along which the function is operating is by year (`data_contries[[i]][,3]`) for *every* variable. In other words, we are replacing the variables of interest in each country for missing years.

```
for (i in 1:length(data_countries)){
data_countries[[i]][,28] <- na.approx(data_countries[[i]][,16], x = index(data_countries[[i]][,3], data_
}

# Interpolate pvoc (data_countries[[i]][,29]) using ratio9050 (data_countries[[i]][,19])

for (i in 1:length(data_countries)){
data_countries[[i]][,29] <- na.approx(data_countries[[i]][,19], x = index(data_countries[[i]][,3], data_
}

data <- do.call("rbind", data_countries)
rm(data_countries)
```

We generate an immigration measure, `fpop` which reflects the percentage of the population that is foreign-born by using our interpolated measure `pforeign`, multiplying it by 1000, and dividin this result by `pop`, which is total population.

```
data$pforeign <- data$pforeign*1000
data$fpop <- data$pforeign/data$pop
```

Our last data cleaning step before moving on to generating the averages for the redistribution models is to generate additional measures of inequality as defined by manipulations to our existing measures of inequality: `ratio9010`, `ratio9010s`, `skew`, and `skews`.

```
data$ratio9010 <- data$pratio9050*data$pratio5010
data$ratio9010s <- data$pratio9050s*data$pratio5010s
data$skew <- data$pratio9050/data$pratio5010
data$skews <- data$pratio9050s/data$pratio5010s
```

Let's generate the averages for the redistribution models by using a series of loops. First we generate the `since` variable, which represents the years since the last redistribution, `redist`, for each country. We remake our list of the subset of countries as before and define `since` (`data_countries[[i]][35]`) accordingly by creating a new logical vector, `nona`, that tells us when the `redist` variable is and is not defined for each country.

```
data_countries <- lapply(unique(data$country), function(x)
  subset(data, data$country==x)
)

for (i in 1:length(data_countries)){
  data_countries[[i]] <- cbind(data_countries[[i]], NA)
  nona <- !is.na(data_countries[[i]][,4])
  data_countries[[i]][,35][nona] <- c(NA, diff(data_countries[[i]][,3][nona]))
}

data <- do.call("rbind", data_countries)
names(data)[35] <- "since"
rm(data_countries)
```