Introducción a Métodos Econométricos en R

Professor: Horacio Larreguy
TA: Eduardo Zago

ITAM Investigación Aplicada 1 / Microeconometría Aplicada,
30/08/2023

General Perspective

Regresión Discontinua

Regresión Discontinua (RDD)

- La regresión discontinua explota un cambio brusco en una variable para explicar otra.
- Se puede examinar cómo una política, tratamiento o intervención afecta una población cuando se cruza un cierto umbral o punto de corte (discontinuity) en la variable independiente.
- Un ejemplo muy sencillo es cuando una política gubernamental esta condicionada en el ingreso de las personas.
- Solo individuos elegibles reciben el tratamiento.
- Individuos cerca del punto de quiebre, que no recibieron el tratamiento, nos brindan un buen *grupo de control*.

Regresión Discontinua: Especificación

- Consideramos el modelo

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 T_i + \delta X_i * T_i + \mu_i,$$

- Y_i denota la variable endógena, X_i una variable explicativa continua y T_i una variable por partes, discontinua, y función de X_i.

$$Z_i = \begin{cases} 0 & X_i \leq \bar{x} \\ 1 & X_i > \bar{x}, \end{cases}$$

- $\bar{x}$ es una constante que se conoce como el punto de quiebre.

RDD: LATE

- Todos los individuos tales que $X_i > \bar{x}$ reciben el tratamiento y aquellos que tienen un valor por debajo o igual al punto de quiebre, $X_i \leq \bar{x}$, no reciben el tratamiento.
- Si consideramos ahora:

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 T_i + \mu_i,$$

- El estimador asociado a T_i, β_2, es el *Average treatment effect* (ATE) de los individuos con $X_i = \bar{x}$.
- Dado que estamos recuperando el efecto cercano al punto de quiebre, también se le conoce como el *Local Average Treatment Effect* (LATE).

Supuesto de No Manipulación

- El supuesto más importante para obtener resultados causales del RDD es el supuesto de no manipulación de los agentes en cuestión.
- Supongan que al enterarse que el gobierno dara un apoyo condicional al ingreso, los individuos deciden disminuirlo o reportar uno menor para ser elegibles.
- Se genera **sesgo de selección**.
- ¿Cómo checamos que no haya manipulación?
- Tenemos que checar la continuidad de la variable explicativa.
- Para esto hay dos tests: **McCrary** (old school, pero informativo) y **CCT** (más reciente, *bias-corrected*).

McCrary Test

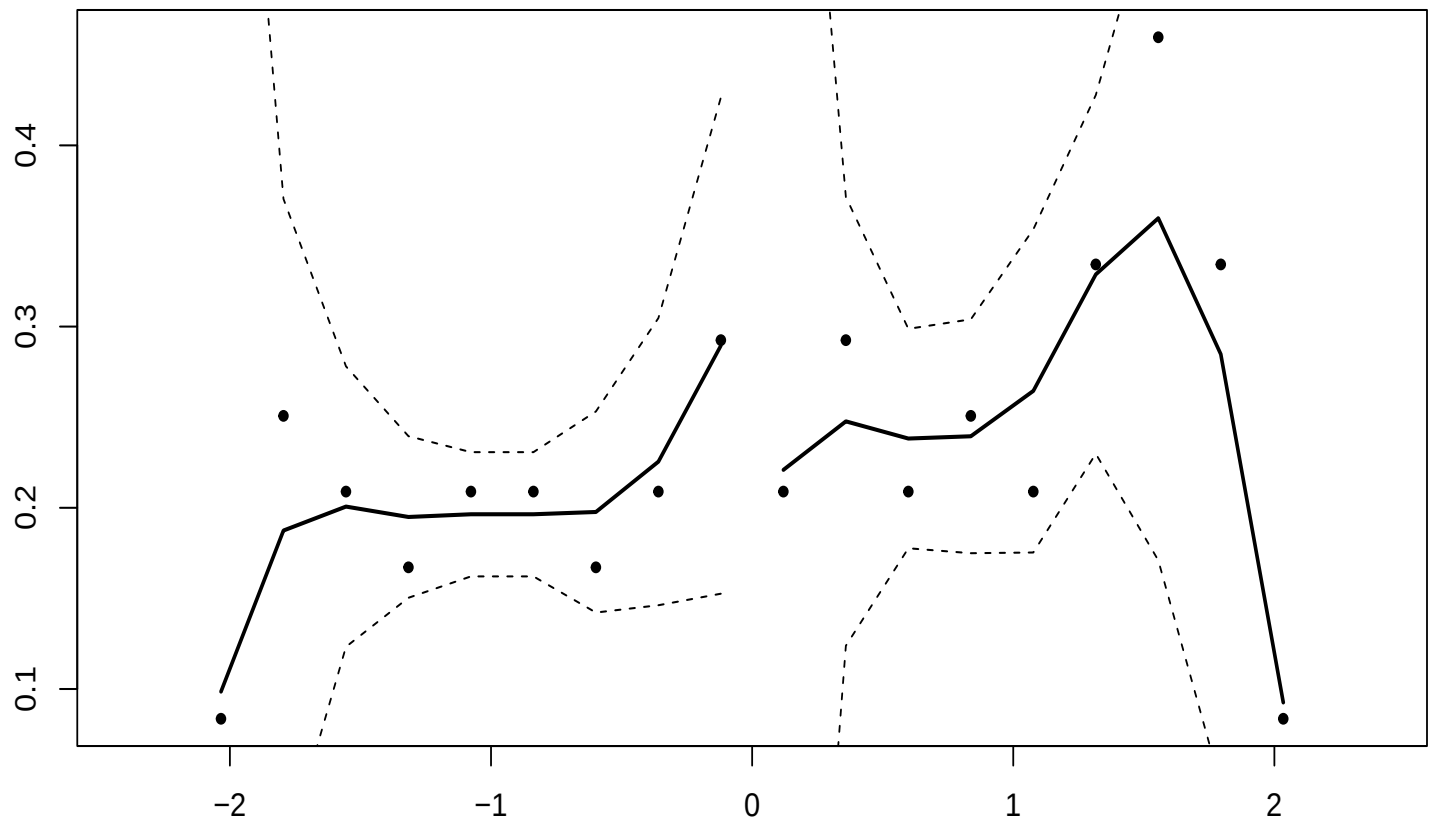
- McCrary desarrolló un *test* de densidad para probar la continuidad de la variable explicativa (*running variable*) en el punto de quiebre.

```
# Test de McCrary
library(rdd)
DCdensity(rd_data$x , 0, plot = F)
[1] 0.4241542
```

- Si el valor p es menor que .05, tenemos 95% de confianza de que ha ocurrido una manipulación.

McCrary Test: Gráfica

```
# Test de McCrary (gráfica)
DCdensity(rd_data$x , 0, plot = T)
```



Test de Continuidad en X: rddensity

- También esta disponible una forma de probar el supuesto de manipulación más actualizada, utilizando a Cataneo et al. (2019).
- El paquete rddensity, además de corregir por el sesgo, nos dara más flexibilidad a la hora de hacer las pruebas.
- En R:

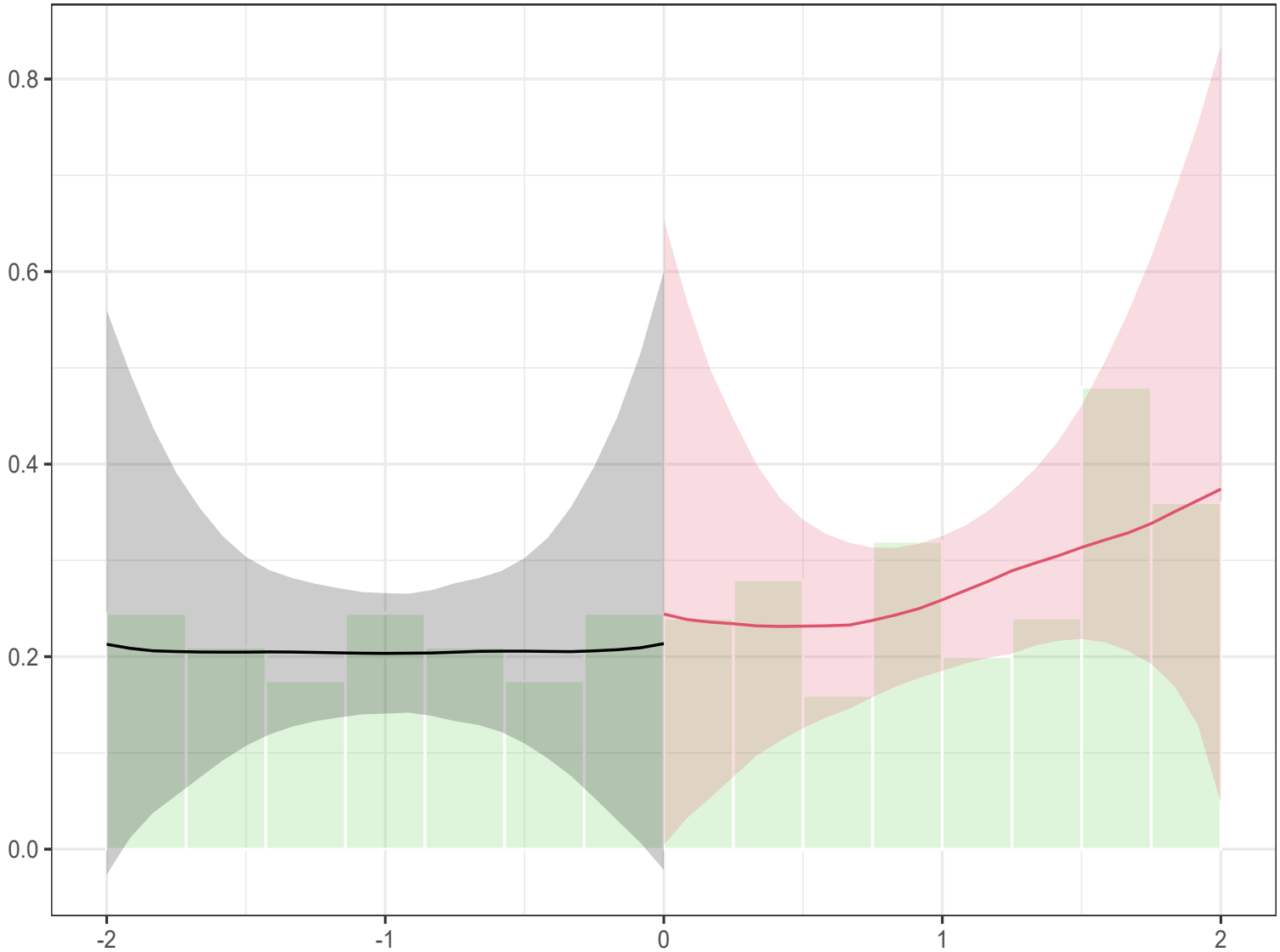
```
library(rddensity)
rdd2 <- rddensity(rd_data$x, c=0, p=1)
summary(rdd2)
####
#  Method                T                    P > |T|
#  Robust                0.2138               0.8307
```

CCT: Gráfica

- Podemos también graficar la distribución de la variable, y agregar la estimación del modelo encima de la misma.
- Si notamos, esta gráfica es mucho más informativa que la de McCrary:

```
set.seed(42) # fix the seed for simulating critical values
plot2 <- rdplotdensity(rdd2, x, plotRange = c(-2, 2),
                       plotN = 25, CIuniform = TRUE)
```

rddensity: Gráfica



RDD en R

- Podemos realizar la estimación más simple del local linear RDD utilizando el paquete felm.
- Seleccionamos un bandwidth cercano al cut-off, para realmente estimar el **local linear** RDD.

```
rd_data_filter <- rd_data |> filter(x < 1 & x > -1)
rd1 <- felm(y ~ z + z*x , data = rd_data_filter)
rd1$coefficients
```

```
                y
(Intercept) 0.8882418
z           4.7435245
x           0.8556126
z:x         0.6086382
```

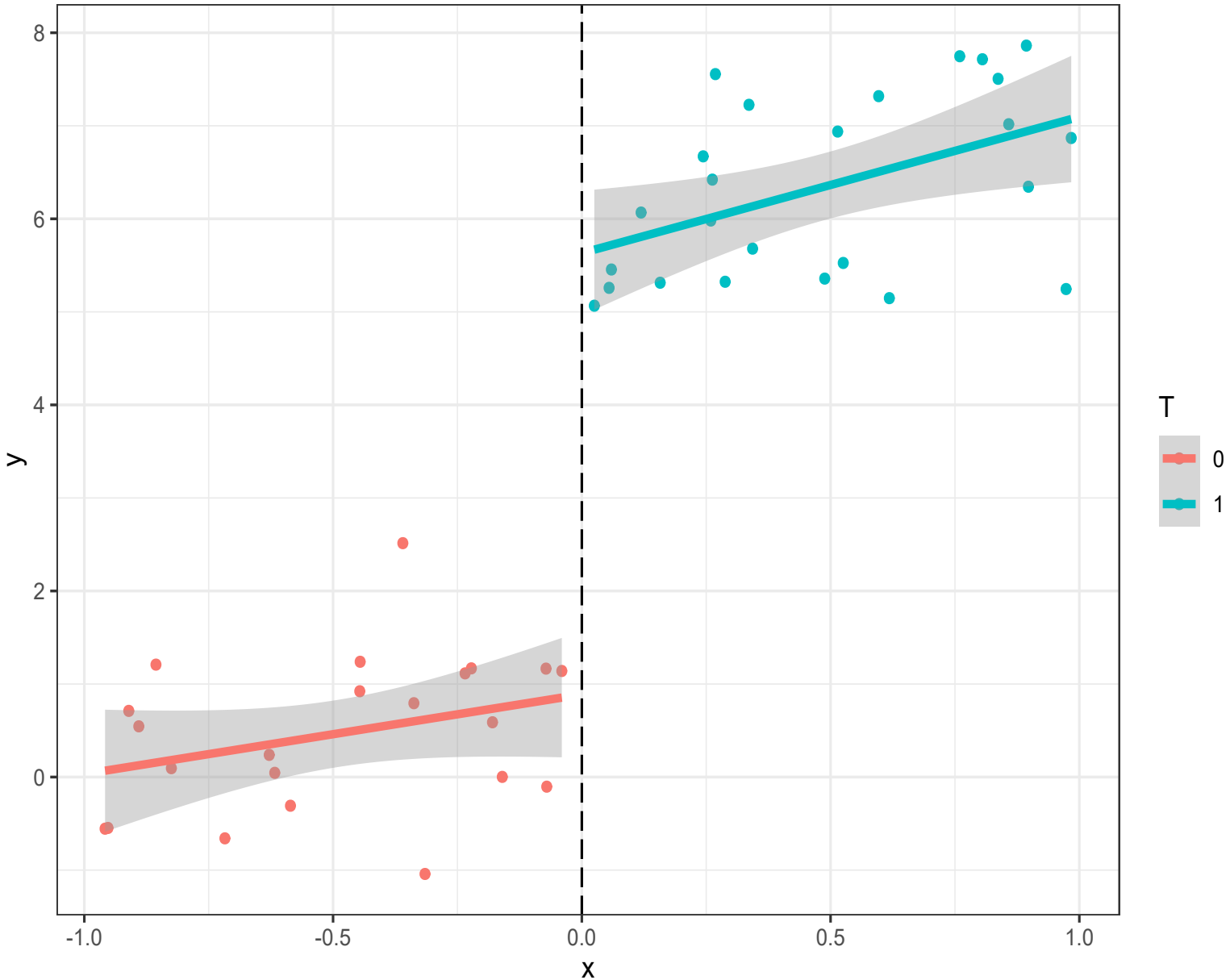
- Es también muy común presentar los resultados utilizando una gráfica.

RDD en R: Gráfica

- Para la gráfica, utilizamos ggplot()

```
ggplot(rd_data_filter, aes(x, y, color = factor(z))) +
            geom_point() +
            geom_smooth(method = 'lm', size = 1.5) +
            geom_vline(xintercept=0, linetype="longdash") +
            theme_bw() +
            xlab("x") +
            ylab("y") +
            scale_color_discrete(name = "T")
```

RDD en R: Gráfica



Paquetes de RDD en R

- Noten que felm() no nos permite cambiar los parámetros de nuestro modelo, ya que la función esta hecha para estimar modelos con efectos fijos.
- La función no nos permite modificar los parámetros de acuerdo a lo que queremos.
- Lo mas importante es que no estima de forma óptima el bandwidth.
- Por lo tanto, recurrimos a paquetes ya documentados que contienen funciones ya establecidas para esto: rddtools y rdrobust.
- Cada uno nos proporciona distintas formas de estimar nuestros modelos, proporcionándonos robustez a la hora de presentar los resultados.

RDD en R: rddtools

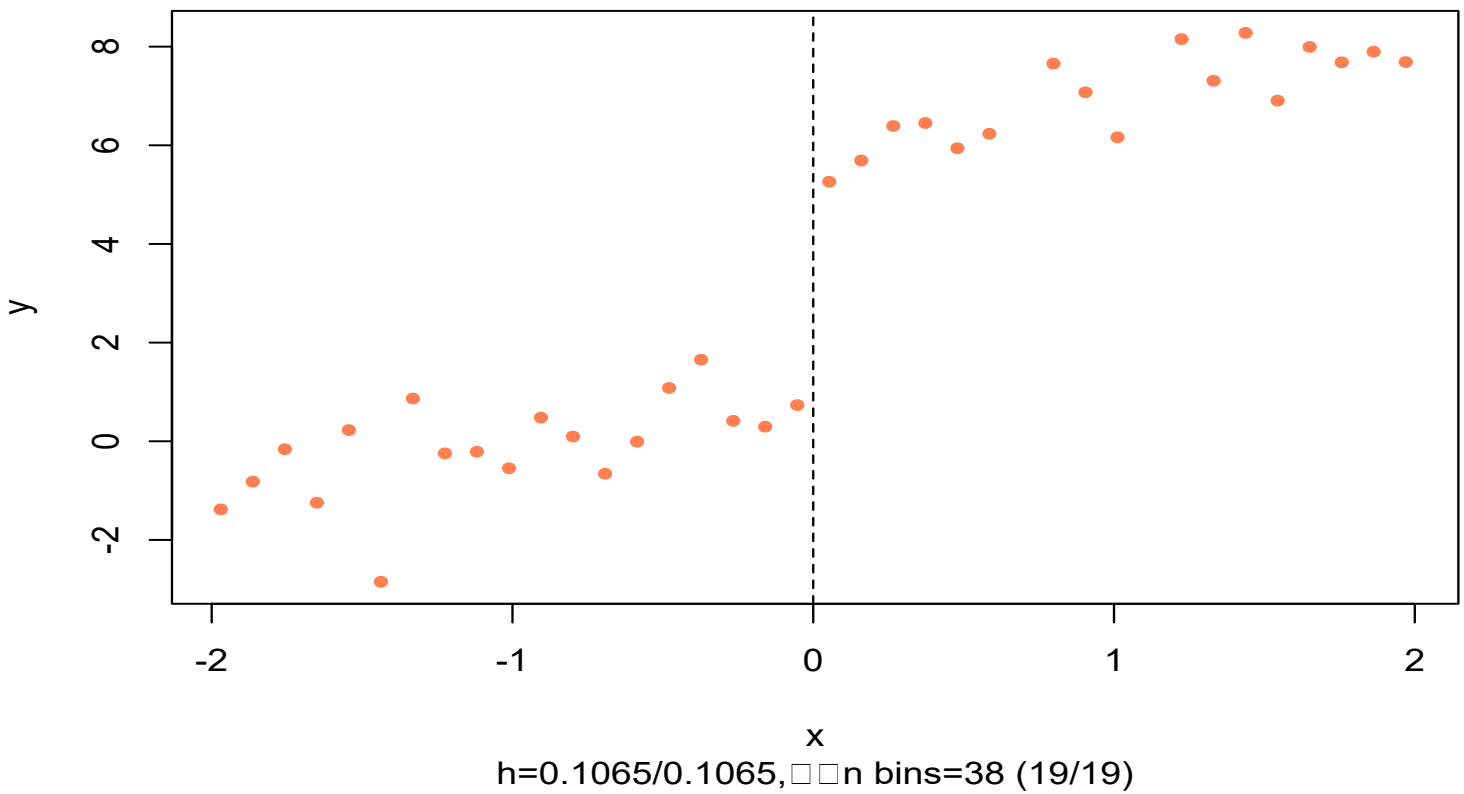
- El paquete rddtools nos da mucha más flexibilidad al dejarnos elegir distintos parámetros que pudieran ser de nuestro interés.
- Empecemos viendo el modelo más simple, sin cambiarle ningún parámetro:

```
# Sin optimal bandwidth
rd_data1 <- rdd_data(rd_data_filter$y, rd_data_filter$x, cutpoint = 0)
rd2$coefficients
(Intercept)           D           x     x_right
0.8882418   4.7435245   0.8556126   0.6086382
```

- Nos da exactamente los mismos resultados que usando felm().

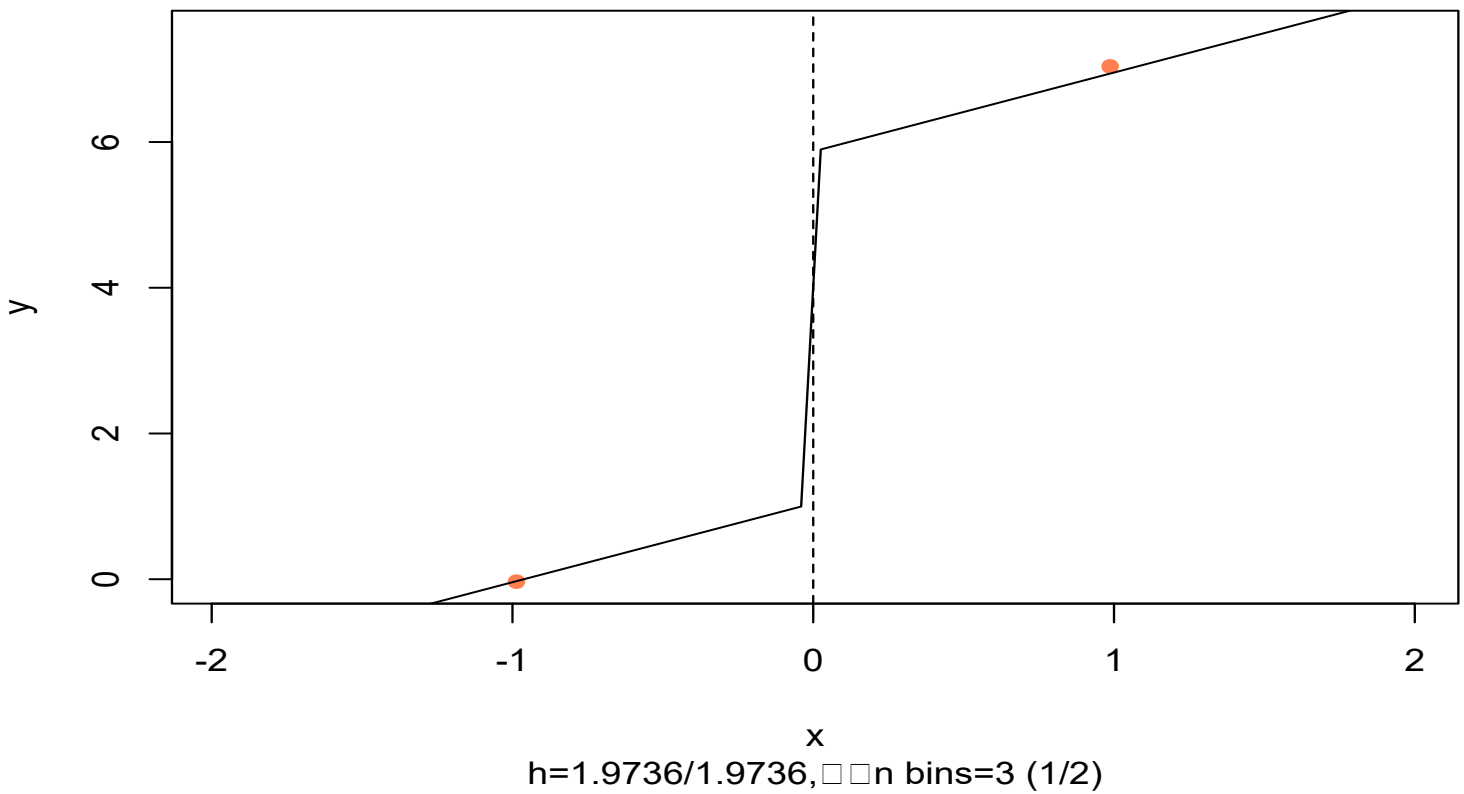
Gráfica de los Datos

```r
# Graficamos los datos muestra
plot(rd_data, col = c("coral"), xlab = "x", ylab = "y")
```



Modelo Ajustado a los Datos

```
plot(rd2, col = "coral",  xlab = "x", ylab = "y")
```



rddtools: Ajustando el Bandwidth

- Lo más importante del paquete de rddtools es que nos permitira elegir cómo estimar el badwidth.
- Este parámetro depende enteramente de los datos y varios autores han propuesto distintas formas de estimar el bandwidth óptimo.
- rddtools nos permite utilizar la estimación de Imbens y Kalyanaraman (2012).

```
rd3_bw <- rdd_reg_lm(rdd_object = rd_data, slope = "separate",
            bw = rdd_bw_ik(rd_data, kernel = "Triangular"))

rd3_bw$coefficients
(Intercept)           D           x     x_right
  0.8891591   4.8656633   0.9065084   0.2972173
```

---

rdrobust: Bandwidth de Calonico et al. (2015)

- El método más reciente para estimar el *bandwidth óptimo* es el de Calonico et al. (2015).
- El diseño de los autores utiliza tanto estimaciones de polinomios locales como inferencia utilizando el error cuadrático medio (MSE) para generar el bandwidth óptimo.
- Además, proponen un estimador robusto y bias-corrected, que hace que el bandwidth sea más pequeño para así reducir la tasa de error de cobertura a la hora de contruir intervalos de confianza.
- Este lo podemos encontrar en su propio paquete, llamado rdrobust.

rdrobust: Bandwidth de Calonico et al. (2015)

- Podemos observar los coeficientes utilizando el summary().

```
# RD con optimal bandwidth usando Calonico et al. 2015
reg_cct = rdrobust(rd_data$y, rd_data$x, all=T)
summary(reg_cct)
```

```
=============================================================================
        Method     Coef. Std. Err.         z     P>|z|      [ 95% C.I. ]
=============================================================================
  Conventional     4.638     0.508     9.132     0.000     [3.642 , 5.633]
Bias-Corrected     4.662     0.508     9.181     0.000     [3.667 , 5.658]
        Robust     4.662     0.632     7.383     0.000     [3.425 , 5.900]
=============================================================================
```

- O mejor aún, graficarlos utilizando funciones del mismo paquete.

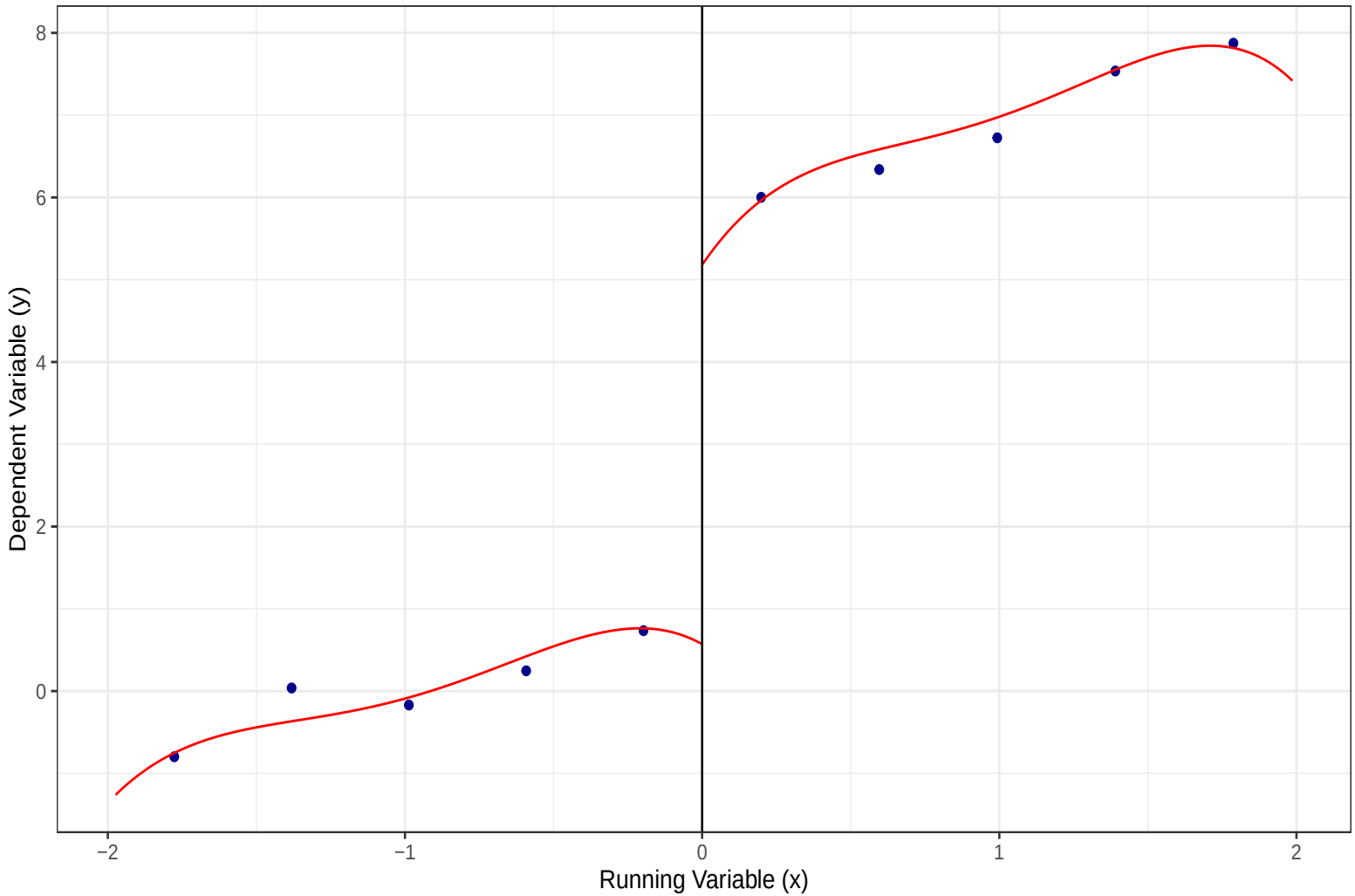
rdrobust: Bandwidth de Calonico et al. (2015)

- Para graficar utilizamos la función rdplot()

```
# Y observar el plot:
rdplot(rd_data$y, rd_data$x,
       x.lab="Running Variable (x)",
       y.lab="Dependent Variable (y)", title = "")
```

---

rdrobust: Gráfica



Efectos Heterogéneos

- Supongamos que quisieramos observar la heterogeneidad de nuestros efectos utilizando otras variables.
- Por ejemplo, supongamos que quisieramos ver la diferencia entre el efecto que tiene X en Y, antes y después de un evento.
- Para eso generamos los siguientes datos:

```
set.seed(5) # Semilla  aleatoria
# Generamos los datos
x <- runif(100, -2, 2)
y <- 1 +  x + 8*(x >= 0) + rnorm(100,0,1)
z <- ifelse(x > 0, 1, 0) # creamos la variable indicadora
# Concatenamos los vectores
rd_data_post <- cbind.data.frame(y,x,z) |> mutate(post = 1)
rd_data <- rd_data |> mutate(post = 0)

final <- rbind(rd_data, rd_data_post)
```

Efectos Heterogéneos: Sample Split

- La forma de analizar estos efectos es realizar un sample split, y estimamos el modelo para cada set de datos.
- Suponiendo que empezabamos con la base final, haríamos:

```
final_pre <- final |> filter(post == 0)
final_post <- final |> filter(post == 1)

reg_pre = rdrobust(final_pre$y, final_pre$x, all=T)
summary(reg_pre)
```

```
=============================================================================
        Method     Coef. Std. Err.         z     P>|z|      [ 95% C.I. ]
=============================================================================
  Conventional     4.638     0.508     9.132     0.000     [3.642 , 5.633]
Bias-Corrected     4.662     0.508     9.181     0.000     [3.667 , 5.658]
        Robust     4.662     0.632     7.383     0.000     [3.425 , 5.900]
=============================================================================
```

Efectos Heterogéneos: Sample Split

- Y el post:

```
reg_post = rdrobust(final_post$y, final_post$x, all=T)
summary(reg_post)
```

```
=============================================================================
        Method     Coef. Std. Err.         z     P>|z|      [ 95% C.I. ]
=============================================================================
  Conventional     8.652     0.862    10.037     0.000     [6.963 , 10.342]
Bias-Corrected     8.702     0.862    10.095     0.000     [7.013 , 10.392]
        Robust     8.702     1.178     7.385     0.000     [6.392 , 11.012]
=============================================================================
```