Statistical Analyses and Differential Privacy. An Empirical Analysis

Marcel Neunhoeffer - University of Mannheim

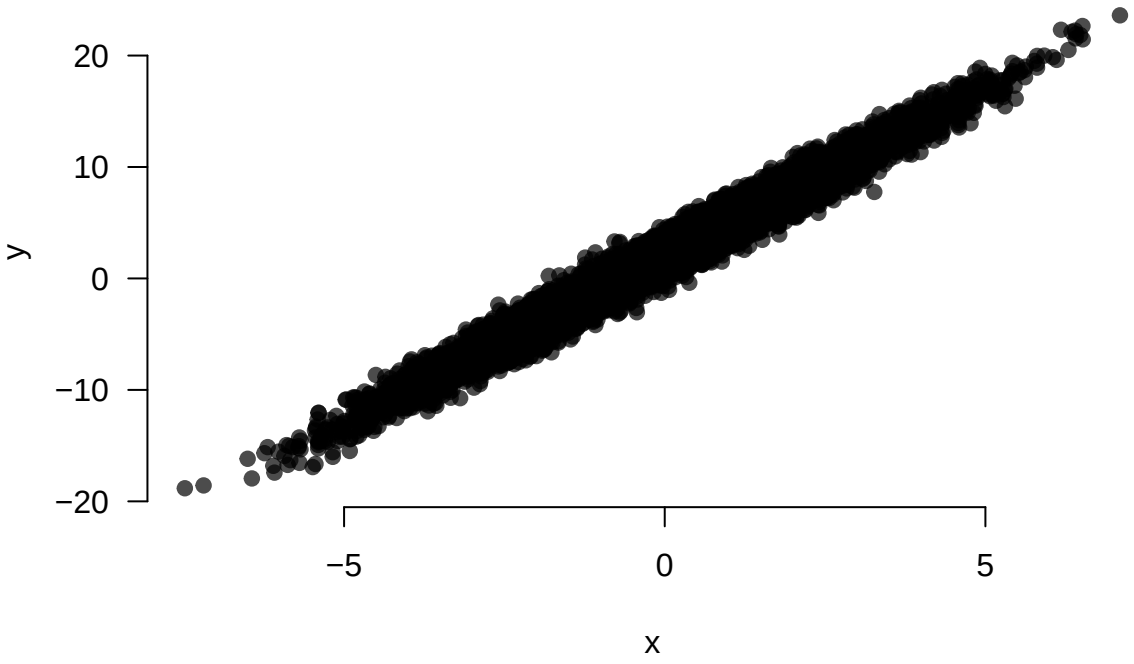
Last updated: 30 April, 2019

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

Load/create the data

```
n <- 10000
x <- rnorm(n, 0, 2)

y <- 2 + 3 * x + rnorm(n, 0, 1)

dat <- data.frame(y = y, x = x)
dat_outlier <- rbind(dat, c(10, -4))

plot(
  x,
  y,
  pch = 19,
  col = adjustcolor("black", alpha = 0.7),
  bty = "n",
  las = 1
)
```



Run Regressions

```
# Number of observations n
n <- nrow(dat)

# Number of regression coefficients
k <- 2
form <- 'y ~ x'
```

OLS

```
reg <- lm(as.formula(form), data = dat)
reg_outlier <- lm(as.formula(form), data = dat_outlier)
```

Get OLS statistics from "Covariance" Matrix

All OLS estimates (coefficients and standard errors) can be obtained from the "covariance matrix" of the data.

```
# Model Formula ff
ff <-
  y ~ x

# Create model frame from formula and data
mf <- model.frame(ff, data = dat)

# Transform model frame to numeric matrix with y in the first column and
# Intercept in the second column.
```

```r
data_y <- model.response(mf, "numeric")
data_x <- model.matrix(ff, data = mf)
df <- cbind(data_y, data_x)

# Calculate "covariance matrix" t(data) %*% data
covariance <- t(df) %*% df

# Which column contains y?
y_col <- 1

# Which column contains the column of 1s?
int_col <- which(apply(df, 2, function(x)
  all(x == 1)))

# Create ols_cov function
ols_cov <-
  function(covariance,
           dp = FALSE,
           y_col = 1,
           int_col = 2) {
    ##########################################
    # Get some helpful indices
    ##########################################

    # Number of rows/columns
    n_cov <- nrow(covariance)

    # Number of independent variables
    k <- n_cov - 1

    ##########################################
    # Subset the matrix to important parts
    ##########################################

    # Number of observations in the data
    n <- covariance[int_col, int_col]

    # t(X) %*% X
    XtX <- covariance[-y_col, -y_col, drop = F]

    # t(X) %*% y
    Xty <- covariance[-y_col, y_col, drop = F]

    # t(y) %*% y
    yty <- covariance[y_col, y_col, drop = F]

    ##########################################
    # Calculate statistics of interest
    ##########################################

    # The sum of the squared residuals
    ete <- yty - (t(Xty) %*% solve(XtX) %*% Xty)
```

```
    # Solve linear system for vector of betas
    betas <- solve(XtX, Xty)

    # Calculate residual variance
    res_var <- as.numeric(ete / (n - k))

    # Calculate covaraince matrix of the coefficients
    cov_betas <- res_var * solve(XtX)

    # Calculate standard errors
    ses <- sqrt(diag(cov_betas))

    res <- cbind(betas, ses)
    colnames(res) <- c("Coefficients", "Standard Errors")
    return(res)
  }

ols_cov(covariance)
```

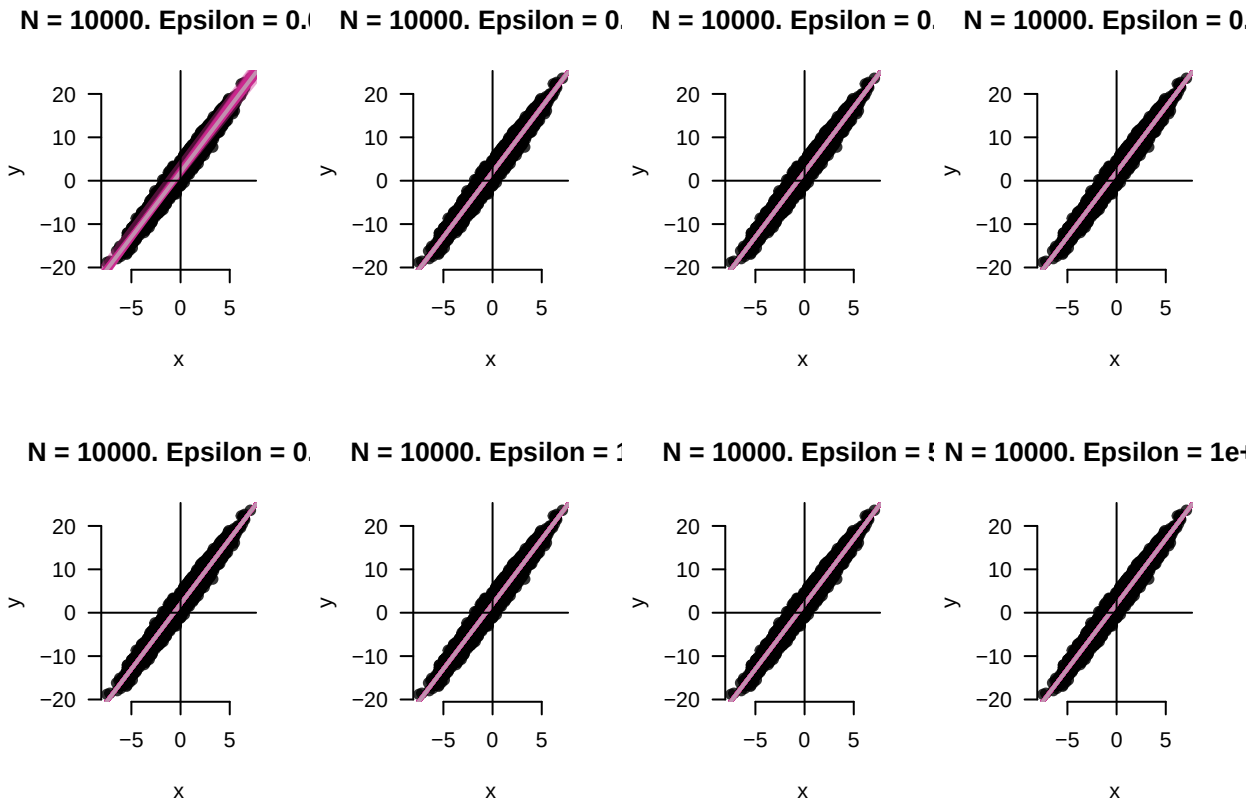
```
##             Coefficients Standard Errors
## (Intercept)     2.004393     0.010027699
## x               2.994346     0.005005753
```

DP-OLS

```
# Functions taken from PSIlence library

dpUnif <- function(n, seed = NULL) {
  if (!is.null(seed)) {
    set.seed(seed)
    return(runif(n))
  }
  return(openssl::rand_num(n))
}

dpNoise <- function(n,
                    scale,
                    dist,
                    shape = NULL,
                    seed = NULL) {
  u <- dpUnif(n, seed)
  if (dist == 'laplace') {
    return(qlap(u, b = scale))
  } else if (dist == 'gaussian') {
    return(qnorm(u, sd = scale))
  } else if (dist == 'gamma') {
    return(qgamma(u, scale = scale, shape = shape))
  } else {
    stop(sprintf('Distribution "%s" not understood', dist))
  }
}
```

```r
qlap <- function(p, mu = 0, b = 1) {
  q <- ifelse(p < 0.5, mu + b * log(2 * p), mu - b * log(2 - 2 * p))
  return(q)
}

# Dimensions of "covariance matrix"
d <- nrow(covariance)

# Number of observations is 1'1
n <- covariance[2, 2]

# Set interesting epsilon values
epsilons <- c(0.01, 0.1, 0.3, 0.5, 0.7, 1, 5, 100000)

# Set number of repetitions
n_rep <- 100

# Set up arrays to hold the results
# res will hold the regression coefficients
res <- array(NA, dim = c(n_rep, d - 1, length(epsilons)))
# res_se will hold the standard errors of the regression coefficients
# (if X'X is invertable, else NA)
res_se <- array(NA, dim = c(n_rep, d - 1, length(epsilons)))

# Loop over epsilon values and number of repetitions.
for (epsilon in 1:length(epsilons)) {
  for (rep in 1:n_rep) {
    # Laplace Noise according to Jian, Xie & Zhang 2015

    # b is the scale parameter for the Laplace distribution
    b <- (2 * d) / (n * epsilons[epsilon])

    # Draw (d^2 + d) / 2 values (lower/upper triangle plus diagonal)
    # from the Laplace distribution with b.
    raw_L <- dpNoise((d ^ 2 + d) / 2, b, dist = "laplace")

    # Put the vector of drawn values into the right positions of a d x d matrix
    L <- matrix(NA, nrow = d, ncol = d)
    L[upper.tri(L, diag = T)] <- raw_L
    L[lower.tri(L)] <- t(L)[lower.tri(L)]

    # Is this ok? Calculate the noisy version of the "covariance matrix".
    dp_cov <- (covariance / n + L) * n

    # Get regression coefficients from noisy covariance matrix
    res[rep, , epsilon] <- ols_cov(dp_cov)[, 1]

    # Get standard errors of the regression coefficients from the noisy
    # covariance matrix (if possible, throws a warning if X'X is not invertable.)
    res_se[rep, , epsilon] <- ols_cov(dp_cov)[, 2]
  }
}
```

```r
# Put it all in a plot
par(mfrow = c(2, 4))
for (epsilon in 1:length(epsilons)) {
  plot(
    dat$x,
    dat$y,
    pch = 19,
    col = adjustcolor("black", 0.8),
    xlab = "x",
    ylab = "y",
    main = paste0("N = ", n, ". Epsilon = ", epsilons[epsilon]),
    las = 1,
    bty = "n"
  )

  for (rep in 1:n_rep) {
    abline(
      a = res[rep, , epsilon][1],
      b = res[rep, , epsilon][2],
      col = adjustcolor("maroon3", alpha = 0.4),
      lwd = 2
    )
  }
  abline(
    a = ols_cov(covariance)[1, 1],
    b = ols_cov(covariance)[2, 1],
    col = adjustcolor("grey", alpha = 0.7),
    lwd = 2
  )
  abline(v = 0, h = 0)
}
```




```
ols_cov(covariance)
```

```
##             Coefficients Standard Errors
## (Intercept)     2.004393     0.010027699
## x               2.994346     0.005005753
```

```
# N is 1'1
n <- covariance[2, 2]

# Function to calculate the sensitivity of the covariance matrix
# slightly adapted from PSIlence Library
covariance_sensitivity <- function(n, rng, intercept = 2) {
  diffs <- apply(rng, 1, diff)
  if (!is.null(intercept)) {
    diffs[intercept] <- 1
  }
  sensitivity <- c()
  for (i in 1:length(diffs)) {
    for (j in i:length(diffs)) {
      s <- ((n - 1) / n) * diffs[i] * diffs[j]
      sensitivity <- c(sensitivity, s)
    }
  }
  return(sensitivity)
}

# Set ranges for every data column
rng <- matrix(NA,
              nrow = k + 1,
```

```
              ncol = 2,
              byrow = TRUE)

# Calculate ranges from min and max
for (i in 1:ncol(df)) {
  rng[i,] <- c(min(df[, i]), max(df[, i]))
}

# Calculate sensitivities
sens <- covariance_sensitivity(n = n, rng = rng)

# Set interesting values of epsilon
epsilons <- c(0.01, 0.1, 0.3, 0.5, 0.7, 1, 5, 100)

# Set number of repetitions
n_rep <- 100

# Create arrays to collect results
res <- array(NA, dim = c(n_rep, d - 1, length(epsilons)))
res_se <- array(NA, dim = c(n_rep, d - 1, length(epsilons)))

# Loop over values of epsilon and repetitions
for (epsilon in 1:length(epsilons)) {
  for (rep in 1:n_rep) {
    # Laplace Noise as implemented in the PSIlence R-library
    noise_scale <- sens / epsilons[epsilon]
    raw_L <-
      dpNoise(n = length(noise_scale),
              scale = noise_scale,
              dist = 'laplace')

    # Put the noise values in the right places in a d x d matrix
    L <- matrix(NA, nrow = d, ncol = d)
    L[lower.tri(L, diag = T)] <- raw_L
    L[upper.tri(L)] <- t(L)[upper.tri(L)]

    # Calculate noise covariance matrix
    dp_cov <- (covariance + L)

    # Run regression on noisy covariance matrix
    res[rep, , epsilon] <- ols_cov(dp_cov)[, 1]
    res_se[rep, , epsilon] <- ols_cov(dp_cov)[, 2]
  }
}
```

Create Plots

Produce plots from the results. The plots will be stored to the figures folder in the project directory.

```
pdf(
  file = paste0("figures/dp_ols_n", n, ".pdf"),
  width = 16,
  height = 9
```

```
)
par(mfrow = c(2, 4))
for (epsilon in 1:length(epsilons)) {
  plot(
    dat$x,
    dat$y,
    pch = 19,
    col = adjustcolor("black", 0.8),
    xlab = "x",
    ylab = "y",
    main = paste0("N = ", n, ". Epsilon = ", epsilons[epsilon]),
    las = 1,
    bty = "n"
  )

  for (rep in 1:n_rep) {
    abline(
      a = res[rep, , epsilon][1],
      b = res[rep, , epsilon][2],
      col = adjustcolor("maroon3", alpha = 0.2),
      lwd = 2
    )
  }
  abline(
    a = ols_cov(covariance)[1, 1],
    b = ols_cov(covariance)[2, 1],
    col = adjustcolor("grey", alpha = 0.7),
    lwd = 2
  )
  abline(v = 0,
         h = 0,
         col = "grey",
         lty = "dashed")
}
dev.off()
```

```
## pdf
##   2
```

Misc

This could be interesting to look at.

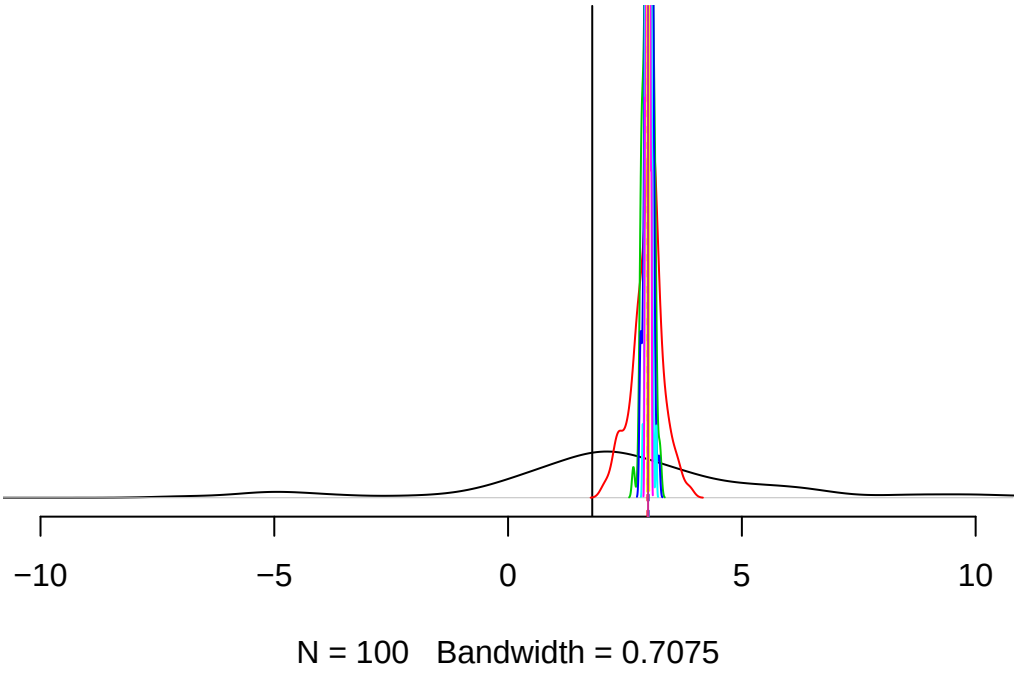
```
ols_cov(covariance)
```

```
##             Coefficients Standard Errors
## (Intercept)     2.004393     0.010027699
## x               2.994346     0.005005753
```

```
cbind(res[rep, , 7], res_se[rep, , 7])
```

```
##          [,1]        [,2]
## [1,] 2.004090 0.009428808
## [2,] 3.013166 0.004721204
```

```r
par(mfrow = c(1, 1))
plot(
  density(res[, 2, 1], na.rm = T),
  ylim = c(0, 2),
  xlim = c(-10, 10),
  main = "",
  bty = "n",
  yaxt = "n",
  ylab = ""
)
abline(v = mean(res[, 2, 1]))
for (i in 2:length(epsilons)) {
  lines(density(res[, 2, i], na.rm = T), col = i)
  abline(v = mean(res[, 2, i]),
         col = i,
         lty = "dashed")
}
abline(v = ols_cov(covariance)[2, 1], col = "maroon3")
```



Old code / Trying DP-OLS with PSIlence library

`rng` Numeric, a $p \times 2$ matrix where each row is a 2-tuple giving an a priori estimate of the lower and upper bounds of the p^{th} variable in the data frame, including the response variable. Ranges should be entered in the order that they appear in the `formula`, such that for a formula `'y ~ x'`, the first row of `rng` should provide the lower and upper bounds on `y` and the second row should provide the lower and upper bounds on `x`.

`rng` should be based in substantive knowledge of the data. If

```r
# Privacy parameter epsilon
epsilon <- 0.03
```

```
# Ranges of data columns
rng <- matrix(NA,
              nrow = k + 1,
              ncol = 2,
              byrow = TRUE)

for (i in 1:ncol(dat)) {
  rng[i, ] <- c(min(dat[, i]), max(dat[, i]))
}
```

```
model <-
  dpGLM$new(
    mechanism = 'mechanismObjective',
    var.type = 'numeric',
    n = n,
    rng = rng,
    epsilon = epsilon,
    formula = form,
    objective = 'ols'
  )

model$release(dat)
```

Evaluate

```
summary(reg)
```

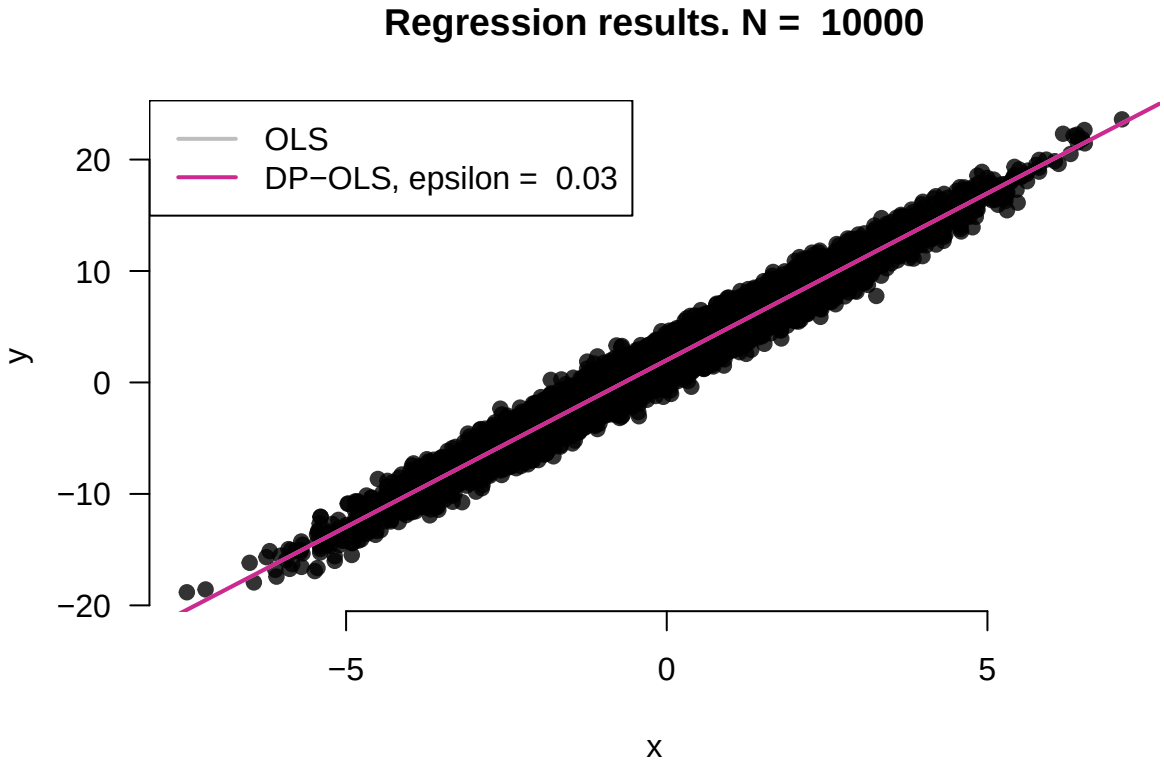
```
##
## Call:
## lm(formula = as.formula(form), data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0284 -0.6758  0.0051  0.6710  3.6624
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.004393   0.010028   199.9   <2e-16 ***
## x           2.994346   0.005006   598.2   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.003 on 9998 degrees of freedom
## Multiple R-squared:  0.9728, Adjusted R-squared:  0.9728
## F-statistic: 3.578e+05 on 1 and 9998 DF,  p-value: < 2.2e-16
```

```
print(model$result)
```

```
## $release
##              estimate
## intercept    2.004349
## x            2.994297
## variance  -552.566650
```

```
##
## $variable
## [1] "y" "x"
##
## $accuracy
## [1] 0.009985774
##
## $epsilon
## [1] 0.03
```

```r
plot(
  dat$x,
  dat$y,
  pch = 19,
  col = adjustcolor("black", 0.8),
  xlab = "x",
  ylab = "y",
  main = paste("Regression results. N = ", n),
  las = 1,
  bty = "n"
)
abline(reg, col = "grey", lwd = 2)
abline(
  a = model$result$release[1, ],
  b = model$result$release[2, ],
  col = "maroon3",
  lwd = 2
)
legend(
  "topleft",
  legend = c("OLS", paste("DP-OLS, epsilon = ", epsilon)),
  lty = "solid",
  lwd = 2,
  col = c("grey", "maroon3")
)
```


Regression results. N = 10000
OLS
DP−OLS, epsilon = 0.03
20
10
0
−10
−20
y
−5
0
5
x