Inference and Simulation

EC 607, Set 04

Edward Rubin

Prologue

Schedule

Last time

The *CEF* and least-squares regression

Today

Inference

Read *MHE* 3.1

Upcoming

Lab: TBD

Problem set 002 coming soon.

Class project, step 1 due on May 1.

Inference

Inference

Why?

Q What's the big deal with inference?

Inference

Why?

Q What's the big deal with inference?

A We rarely know the CEF or the population (and its regression vector).

We *can* draw statistical inferences about the population using samples.

Inference

Why?

Q What's the big deal with inference?

A We rarely know the CEF or the population (and its regression vector).

We *can* draw statistical inferences about the population using samples.

Important The issue/topic of *statistical inference* is separate from *causality*.

Separate questions

1. How do we interpret the estimated coefficient $\hat{\beta}$?
2. What is the sampling distribution of $\hat{\beta}$?

Moving from population to sample

Recall The population-regression function gives us the best linear approximation to the CEF.

Moving from population to sample

Recall The population-regression function gives us the best linear approximation to the CEF.

We're interested in the (unknown) population-regression vector

$$\beta = \boldsymbol{E}\left[\mathbf{X}_i \mathbf{X}_i'\right]^{-1} \boldsymbol{E}[\mathbf{X}_i \mathbf{Y}_i]$$

Moving from population to sample

Recall The population-regression function gives us the best linear approximation to the CEF.

We're interested in the (unknown) population-regression vector

$$\beta = \boldsymbol{E}\left[\mathbf{X}_i \mathbf{X}_i'\right]^{-1} \boldsymbol{E}[\mathbf{X}_i \mathbf{Y}_i]$$

which we estimate via the ordinary least squares (OLS) estimator†

$$\hat{\beta} = \left(\sum_i \mathbf{X}_i \mathbf{X}_i' \right)^{-1} \left(\sum_i \mathbf{X}_i \mathbf{Y}_i \right)$$

† *MHE* presents a method-of-moments motivation for this derivation, where $\dfrac{1}{n} \sum_i \mathbf{X}_i \mathbf{X}_i'$ is our sample-based estimated for $\boldsymbol{E}\left[\mathbf{X}_i \mathbf{X}_i'\right]$. You've also seen others, *e.g.*, minimizing MSE of $\mathbf{Y}_i$ given $\mathbf{X}_i$.

A classic

However you write it, this OLS estimator

$$
\begin{aligned}
\hat{\beta} &= \left(\mathrm{X}'\mathrm{X}\right)^{-1}\mathrm{X}'\mathrm{y} \\
&= \left(\sum_i \mathrm{X}_i\mathrm{X}_i'\right)^{-1}\left(\sum_i \mathrm{X}_i\mathrm{Y}_i\right) \\
&= \beta + \left[\sum_i \mathrm{X}_i\mathrm{X}_i'\right]^{-1}\sum_i \mathrm{X}_i e_i
\end{aligned}
$$

is the same estimator you've been using since undergrad.

Inference

A classic

However you write it, this OLS estimator

$$
\begin{aligned}
\hat{\beta} &= \left(\mathrm{X}'\mathrm{X}\right)^{-1}\mathrm{X}'\mathrm{y} \\
&= \left(\sum_i \mathrm{X}_i\mathrm{X}_i'\right)^{-1}\left(\sum_i \mathrm{X}_i\mathrm{Y}_i\right) \\
&= \beta + \left[\sum_i \mathrm{X}_i\mathrm{X}_i'\right]^{-1}\sum_i \mathrm{X}_i e_i
\end{aligned}
$$

is the same estimator you've been using since undergrad.

Note I'm following *MHE* in defining $e_i = \mathbf{Y}_i - \mathbf{X}_i'\beta$.

A classic

As you've learned, the OLS estimator

$$\hat{\beta} = \left(\sum_i \mathrm{X}_i \mathrm{X}_i'\right)^{-1} \left(\sum_i \mathrm{X}_i \mathrm{Y}_i\right) = \beta + \left[\sum_i \mathrm{X}_i \mathrm{X}_i'\right]^{-1} \sum_i \mathrm{X}_i e_i$$

has asymptotic covariance

$$\boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i'\right]^{-1} \boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i' e_i^2\right] \boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i'\right]^{-1}$$

A classic

As you've learned, the OLS estimator

$$\hat{\beta} = \left(\sum_i \mathrm{X}_i \mathrm{X}_i' \right)^{-1} \left(\sum_i \mathrm{X}_i \mathrm{Y}_i \right) = \beta + \left[\sum_i \mathrm{X}_i \mathrm{X}_i' \right]^{-1} \sum_i \mathrm{X}_i e_i$$

has asymptotic covariance

$$\boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i' \right]^{-1} \boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i' e_i^2 \right] \boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i' \right]^{-1}$$

which we estimate by (**1**) replacing e_i with $\hat{e}_i = \mathrm{Y}_i - \mathrm{X}_i' \hat{\beta}$ and (**2**) replacing expectations with sample means, *e.g.*, $\boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i' e_i^2 \right]$ becomes $\dfrac{1}{n} \sum \left[\mathrm{X}_i \mathrm{X}_i' \hat{e}_i^2 \right]$.

Inference

A classic

As you've learned, the OLS estimator

$$\hat{\beta} = \left(\sum_i \mathrm{X}_i \mathrm{X}_i'\right)^{-1} \left(\sum_i \mathrm{X}_i \mathrm{Y}_i\right) = \beta + \left[\sum_i \mathrm{X}_i \mathrm{X}_i'\right]^{-1} \sum_i \mathrm{X}_i e_i$$

has asymptotic covariance

$$\boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i'\right]^{-1} \boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i' e_i^2\right] \boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i'\right]^{-1}$$

which we estimate by (**1**) replacing e_i with $\hat{e}_i = \mathrm{Y}_i - \mathrm{X}_i'\hat{\beta}$ and (**2**) replacing expectations with sample means, *e.g.*, $\boldsymbol{E}\left[\mathrm{X}_i \mathrm{X}_i' e_i^2\right]$ becomes $\dfrac{1}{n} \sum \left[\mathrm{X}_i \mathrm{X}_i' \hat{e}_i^2\right]$.

Standard errors of this flavor are known as heteroskedasticity-consistent (or -robust) standard errors (or Eicker-Huber-White).

Defaults

Statistical packages default to assuming homoskedasticity, *i.e.*, $E\left[e_i^2 \mid \mathrm{X}_i\right] = \sigma^2$ for all i.

Defaults

Statistical packages default to assuming homoskedasticity, *i.e.*, $E\left[e_i^2 \mid \mathrm{X}_i\right] = \sigma^2$ for all i. With homoskedasticity,

$$E\left[\mathrm{X}_i \mathrm{X}_i' e_i^2\right] = E\left[E\left[\mathrm{X}_i \mathrm{X}_i' e_i^2 \mid \mathrm{X}_i\right]\right] = E\left[\mathrm{X}_i \mathrm{X}_i' \, E\left[e_i^2 \mid \mathrm{X}_i\right]\right] = \sigma^2 \, E\left[\mathrm{X}_i \mathrm{X}_i'\right]$$

Defaults

Statistical packages default to assuming homoskedasticity, *i.e.*, $E\left[e_i^2 \mid \mathrm{X}_i\right] = \sigma^2$ for all i. With homoskedasticity,

$$E\left[\mathrm{X}_i\mathrm{X}_i' e_i^2\right] = E\left[E\left[\mathrm{X}_i\mathrm{X}_i' e_i^2 \mid \mathrm{X}_i\right]\right] = E\left[\mathrm{X}_i\mathrm{X}_i'\, E\left[e_i^2 \mid \mathrm{X}_i\right]\right] = \sigma^2\, E\left[\mathrm{X}_i\mathrm{X}_i'\right]$$

Now, returning to to the asym. covariance matrix of $\hat{\beta}$,

$$\begin{aligned} E\left[\mathrm{X}_i\mathrm{X}_i'\right]^{-1} E\left[\mathrm{X}_i\mathrm{X}_i' e_i^2\right] E\left[\mathrm{X}_i\mathrm{X}_i'\right]^{-1} &= E\left[\mathrm{X}_i\mathrm{X}_i'\right]^{-1} \sigma^2\, E\left[\mathrm{X}_i\mathrm{X}_i'\right] E\left[\mathrm{X}_i\mathrm{X}_i'\right]^{-1} \\ &= \sigma^2\, E\left[\mathrm{X}_i\mathrm{X}_i'\right]^{-1} \end{aligned}$$

Inference

Defaults

Angrist and Pischke argue we should probably change our default to heteroskedasticity.

If the CEF is nonlinear, then our linear approximation (linear regression) generates heteroskedasticity.

Defaults

Angrist and Pischke argue we should probably change our default to heteroskedasticity.

If the CEF is nonlinear, then our linear approximation (linear regression) generates heteroskedasticity.

$$E\left[\left(\mathrm{Y}_i - \mathrm{X}_i'\beta\right)^2 \mid \mathrm{X}_i\right]$$

Defaults

Angrist and Pischke argue we should probably change our default to heteroskedasticity.

If the CEF is nonlinear, then our linear approximation (linear regression) generates heteroskedasticity.

$$
\begin{aligned}
&E\left[\left(\mathrm{Y}_i - \mathrm{X}_i'\beta\right)^2 \mid \mathrm{X}_i\right] \\
&\quad = E\left[\left(\left\{\mathrm{Y}_i - E[\mathrm{Y}_i \mid \mathrm{X}_i]\right\} + \left\{E[\mathrm{Y}_i \mid \mathrm{X}_i] - \mathrm{X}_i'\beta\right\}\right)^2 \middle| \mathrm{X}_i\right]
\end{aligned}
$$

Defaults

Angrist and Pischke argue we should probably change our default to heteroskedasticity.

If the CEF is nonlinear, then our linear approximation (linear regression) generates heteroskedasticity.

$$
\begin{aligned}
&E\Big[\big(\mathrm{Y}_i - \mathrm{X}_i'\beta\big)^2 \mid \mathrm{X}_i\Big] \\
&\quad = E\Bigg[\bigg(\{\mathrm{Y}_i - E[\mathrm{Y}_i \mid \mathrm{X}_i]\} + \{\, E[\mathrm{Y}_i \mid \mathrm{X}_i] - \mathrm{X}_i'\beta\}\bigg)^2 \bigg| \mathrm{X}_i\Bigg] \\
&\quad = \mathrm{Var}(\mathrm{Y}_i \mid \mathrm{X}_i) + \big(E[\mathrm{Y}_i \mid \mathrm{X}_i] - \mathrm{X}_i'\beta\big)^2
\end{aligned}
$$

Defaults

Angrist and Pischke argue we should probably change our default to heteroskedasticity.

If the CEF is nonlinear, then our linear approximation (linear regression) generates heteroskedasticity.

$$
\begin{aligned}
&E\Big[\big(\mathrm{Y}_i - \mathrm{X}_i'\beta\big)^2 \mid \mathrm{X}_i\Big] \\
&\quad = E\left[\left(\{\mathrm{Y}_i - E[\mathrm{Y}_i \mid \mathrm{X}_i]\} + \{\, E[\mathrm{Y}_i \mid \mathrm{X}_i] - \mathrm{X}_i'\beta\}\right)^2 \middle| \mathrm{X}_i\right] \\
&\quad = \mathrm{Var}(\mathrm{Y}_i \mid \mathrm{X}_i) + \big(E[\mathrm{Y}_i \mid \mathrm{X}_i] - \mathrm{X}_i'\beta\big)^2
\end{aligned}
$$

Thus, even if $\mathbf{Y}_i \mid \mathbf{X}_i$ has contant variance, $e_i \mid \mathbf{X}_i$ is heteroskedastic.

Defaults

Angrist and Pischke argue we should probably change our default to heteroskedasticity.

If the CEF is nonlinear, then our linear approximation (linear regression) generates heteroskedasticity.

$$
\begin{aligned}
&\boldsymbol{E}\Big[\big(\mathrm{Y}_i - \mathrm{X}_i'\beta\big)^2 \mid \mathrm{X}_i\Big] \\
&\quad = \boldsymbol{E}\left[\left(\left\{\mathrm{Y}_i - \boldsymbol{E}[\mathrm{Y}_i \mid \mathrm{X}_i]\right\} + \left\{\boldsymbol{E}[\mathrm{Y}_i \mid \mathrm{X}_i] - \mathrm{X}_i'\beta\right\}\right)^2 \middle| \mathrm{X}_i\right] \\
&\quad = \mathbf{Var}(\mathrm{Y}_i \mid \mathrm{X}_i) + \big(\boldsymbol{E}[\mathrm{Y}_i \mid \mathrm{X}_i] - \mathrm{X}_i'\beta\big)^2
\end{aligned}
$$

Thus, even if $\mathbf{Y}_i \mid \mathbf{X}_i$ has contant variance, $e_i \mid \mathbf{X}_i$ is heteroskedastic. Unless you want to assume the CEF is *linear*.

Inference

Two notes

1. Heteroskedasticity is **not our biggest concern** in inference.

   > ...as an empirical matter, heteroskedasticity may matter very little... If heteroskedasticity matters a lot, say, more than a 30 percent increase or any marked decrease in standard errors, you should worry about possible programming errors or other problems. (*MHE*, p.47)

2. Notice that we've **avoided "standard" stronger assumptions**, *e.g.*, normality, fixed regressors, linear CEF, homoskedasticity.

Two notes

1. Heteroskedasticity is **not our biggest concern** in inference.

> ...as an empirical matter, heteroskedasticity may matter very little... If heteroskedasticity matters a lot, say, more than a 30 percent increase or any marked decrease in standard errors, you should worry about possible programming errors or other problems. (*MHE*, p.47)

2. Notice that we've **avoided "standard" stronger assumptions**, *e.g.*, normality, fixed regressors, linear CEF, homoskedasticity.

Following (2): We only have large-sample, asymptotic results (consistency) rather than finite-sample results (unbiasedness).

Inference

Warning

Because many of properties we care about for the inference are **large-sample** properties, they may not always apply to **small samples**.

Warning

Because many of properties we care about for the inference are **large-sample** properties, they may not always apply to **small samples**.

One practical way we can study the behavior of an estimator: **simulation**.

Warning

Because many of properties we care about for the inference are **large-sample** properties, they may not always apply to **small samples**.

One practical way we can study the behavior of an estimator: **simulation**.

Note You need to make sure your simulation can actually test/respond to the question you are asking (*e.g.*, bias *vs.* consistency).

Simulation

Let's compare false- and true-positive rates[†] for

1. **Homoskedasticity-assuming standard errors** $\left(\mathrm{Var}[e_i|\mathrm{X}_i] = \sigma^2\right)$
2. **Heteroskedasticity-robust standard errors**

† The false-positive rate goes by many names; another common name: *type-I error rate*.

Simulation

Let's compare false- and true-positive rates[†] for

1. **Homoskedasticity-assuming standard errors** $\left(\mathrm{Var}[e_i|\mathrm{X}_i] = \sigma^2\right)$
2. **Heteroskedasticity-robust standard errors**

Simulation outline

```
1. Define data–generating process (DGP).
2. Choose sample size n.
3. Set seed.
4. Run 10,000 iterations of
     a. Draw sample of size n from DGP.
     b. Conduct inference.
     c. Record inferences' outcomes.
```

† The false-positive rate goes by many names; another common name: *type-I error rate*.

Simulation

Simulation

Data-generating process

First, we'll define our DGP.

Simulation

Data-generating process

First, we'll define our DGP.

We've been talking a lot about nonlinear CEFs, so let's use one.

Let's keep the disturbances well behaved.

Simulation

Data-generating process

First, we'll define our DGP.

We've been talking a lot about nonlinear CEFs, so let's use one.

Let's keep the disturbances well behaved.

$$\mathbf{Y}_i = 1 + e^{0.5\mathbf{X}_i} + \varepsilon_i$$

where $\mathbf{X}_i \sim \mathbf{Uniform}(0, 10)$ and $\varepsilon_i \sim N(0, 1)$.

Simulation

Data-generating process

$$\mathrm{Y}_i = 1 + e^{0.5\mathrm{X}_i} + \varepsilon_i$$

where $\mathrm{X}_i \sim \mathrm{Uniform}(0, 10)$ and $\varepsilon_i \sim N(0, 15^2)$.

Simulation

Data-generating process

$$\text{Y}_i = 1 + e^{0.5\text{X}_i} + \varepsilon_i$$

where $\text{X}_i \sim \text{Uniform}(0, 10)$ and $\varepsilon_i \sim N(0, 15^2)$.

```r
library(pacman)
p_load(dplyr)
# Choose a size
n = 1000
# Generate data
dgp_df = tibble(
  ε = rnorm(n, sd = 15),
  x = runif(n, min = 0, max = 10),
  y = 1 + exp(0.5 * x) + ε
)
```

Simulation

Data-generating process

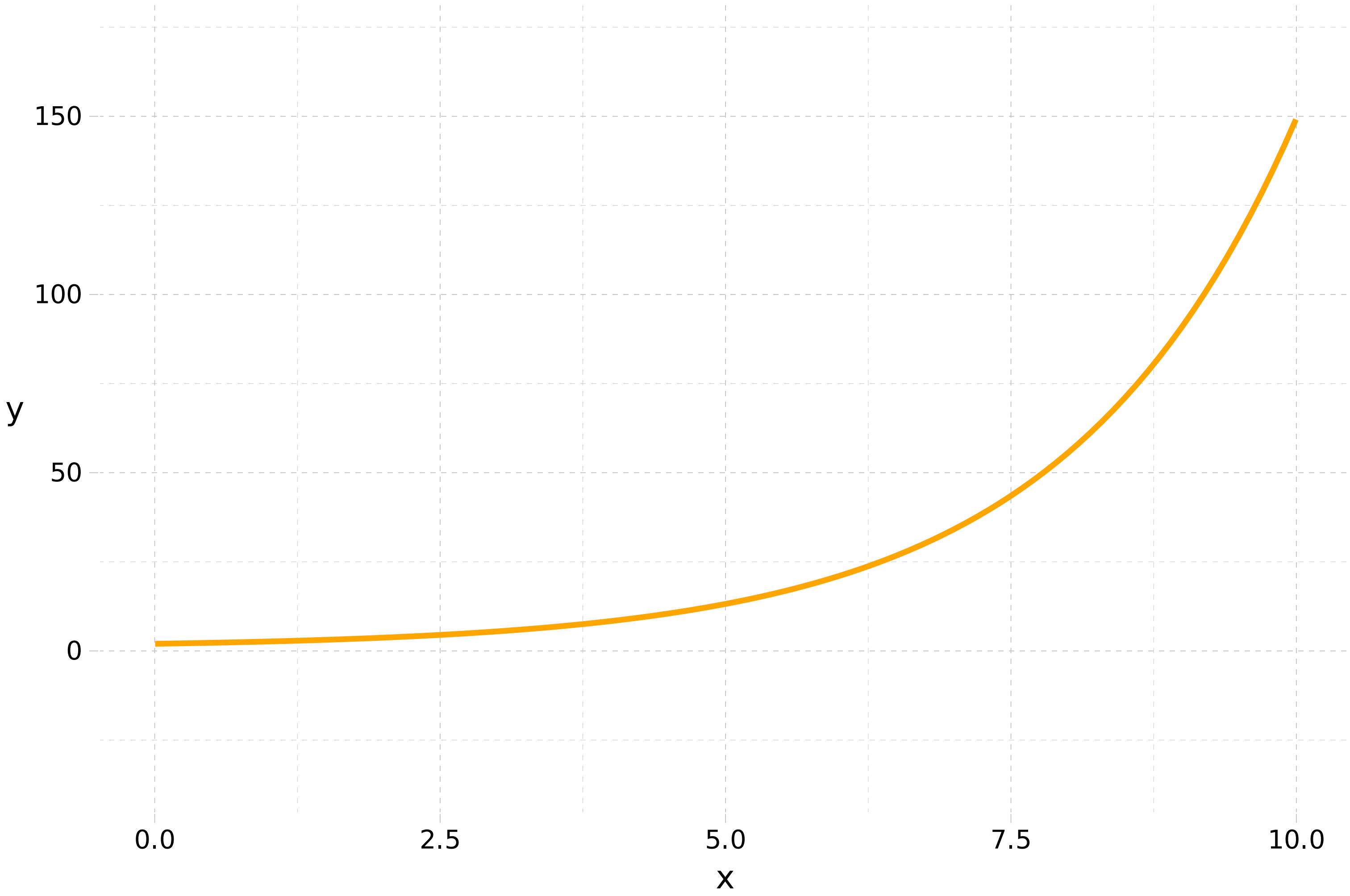
$$\text{Y}_i = 1 + e^{0.5\text{X}_i} + \varepsilon_i$$

where $\text{X}_i \sim \text{Uniform}(0, 10)$ and $\varepsilon_i \sim N(0, 15^2)$.

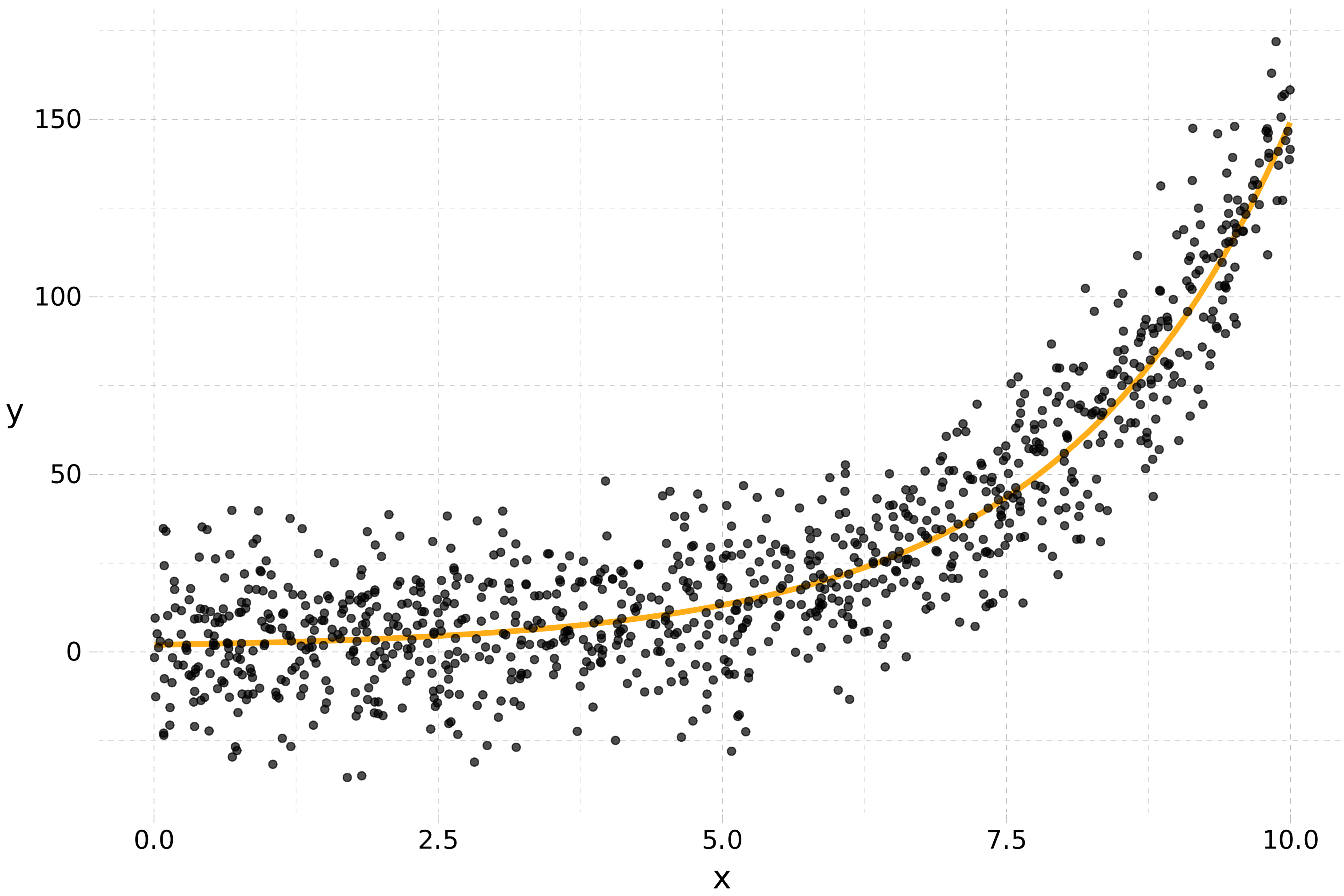
```r
library(pacman)
p_load(dplyr)
# Choose a size
n = 1000
# Generate data
dgp_df = tibble(
  ε = rnorm(n, sd = 15),
  x = runif(n, min = 0, max = 10),
  y = 1 + exp(0.5 * x) + ε
)
```

```
#> # A tibble: 1,000 × 3
#>          ε     x      y
#>      <dbl> <dbl>  <dbl>
#>  1   8.78   9.53 127.
#>  2  10.6    6.22  34.0
#>  3  -1.64   5.32  13.6
#>  4  -6.80   8.92  80.7
#>  5   9.09   1.96  12.8
#>  6 -27.3    8.84  57.0
#>  7   9.45   2.18  13.4
#>  8  -4.14   3.78   3.47
#>  9  -4.26   3.52   2.54
#> 10 -13.8    9.88 127.
#> # ℹ 990 more rows
```

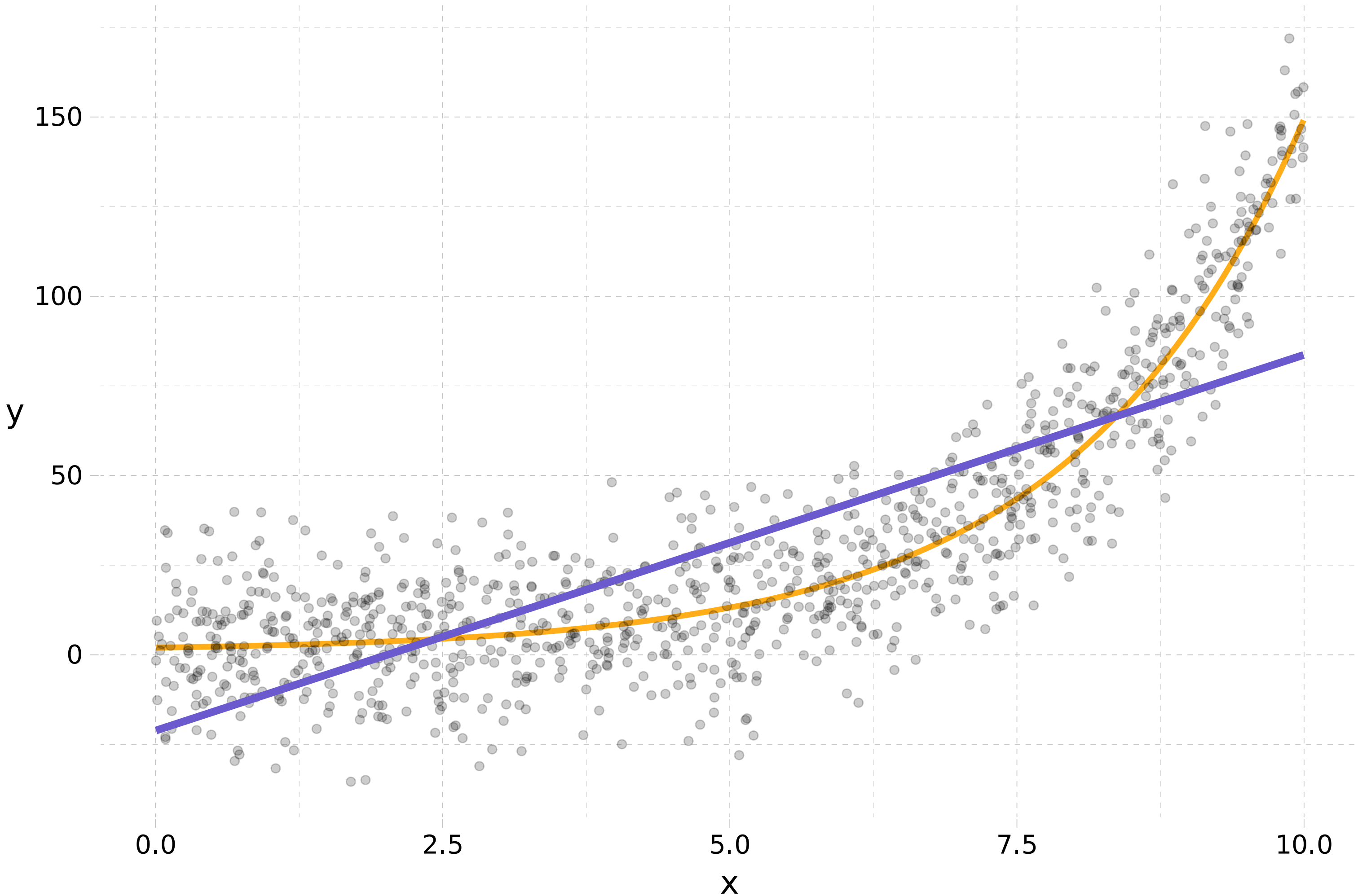
Our CEF



Our population



The population least-squares regression line



Iterating

To make iterating easier, let's wrap our DGP in a function.

```
fun_iter = function(iter, n = 30) {
  # Generate data
  iter_df = tibble(
    ε = rnorm(n, sd = 15),
    x = runif(n, min = 0, max = 10),
    y = 1 + exp(0.5 * x) + ε
  )
}
```

We still need to run a regression and draw some inferences.

Note We're defaulting to size-30 samples.

Simulation

We will use `lm_robust()` from the `estimatr` package for OLS and inference.†

- `se_type = "classical"` provides homoskedasticity-assuming SEs
- `se_type = "HC2"` provides heteroskedasticity-robust SEs

```
lm_robust(y ~ x, data = dgp_df, se_type = "classical") %>% tidy() %>% select(1:5)
```

```
#>          term  estimate std.error statistic       p.value
#> 1 (Intercept) -21.14183  1.473496 -14.34807  1.383951e-42
#> 2           x  10.48074  0.257810  40.65294 6.560626e-214
```

```
lm_robust(y ~ x, data = dgp_df, se_type = "HC2") %>% tidy() %>% select(1:5)
```

```
#>          term  estimate std.error statistic       p.value
#> 1 (Intercept) -21.14183 1.4335274 -14.74812  1.112039e-44
#> 2           x  10.48074 0.3097606  33.83495 8.788638e-168
```

† `lm()` works for "spherical" standard errors but cannot calculate het.-robust standard errors.

Simulation

Inference

Now add these estimators to our iteration function…

```
fun_iter = function(iter, n = 30) {
  # Generate data
  iter_df = tibble(
    ε = rnorm(n, sd = 15),
    x = runif(n, min = 0, max = 10),
    y = 1 + exp(0.5 * x) + ε
  )
  # Estimate models
  lm1 = lm_robust(y ~ x, data = iter_df, se_type = "classical")
  lm2 = lm_robust(y ~ x, data = iter_df, se_type = "HC2")
  # Stack and return results
  bind_rows(tidy(lm1), tidy(lm2)) %>%
    select(1:5) %>% filter(term == "x") %>%
    mutate(se_type = c("classical", "HC2"), i = iter)
}
```

Run it

Now we need to actually run our `fun_iter()` function 10,000 times.

Run it

Now we need to actually run our `fun_iter()` function 10,000 times.

There are a lot of ways to run a single function over a list/vector of values.

- `lapply()`, *e.g.*, `lapply(X = 1:3, FUN = sqrt)`
- `for()`, *e.g.*, `for (x in 1:3) sqrt(x)`
- `map()` from `purrr`, *e.g.*, `map(1:3, sqrt)`

Simulation

Run it

Now we need to actually run our `fun_iter()` function 10,000 times.

There are a lot of ways to run a single function over a list/vector of values.

- `lapply()`, *e.g.*, `lapply(X = 1:3, FUN = sqrt)`
- `for()`, *e.g.*, `for (x in 1:3) sqrt(x)`
- `map()` from `purrr`, *e.g.*, `map(1:3, sqrt)`

We're going to go with `map()` from the `purrr` package because it easily parallelizes across platforms using the `furrr` package.

Simulation

Run it!

Run our function 10,000 times

```
# Packages
p_load(purrr)
# Set seed
set.seed(12345)
# Run 10,000 iterations
sim_list = map(1:1e4, fun_iter)
```

Run it!

Run our function 10,000 times

```
# Packages
p_load(purrr)
# Set seed
set.seed(12345)
# Run 10,000 iterations
sim_list = map(1:1e4, fun_iter)
```

Parallelized 10,000 iterations

```
# Packages
p_load(purrr, furrr)
# Set options
set.seed(123)
# Tell R to parallelize
plan(multisession)
# Run 10,000 iterations
sim_list = future_map(
  1:1e4, fun_iter,
  .options = furrr_options(seed = T)
)
```

Simulation

Run it!

Run our function 10,000 times

```
# Packages
p_load(purrr)
# Set seed
set.seed(12345)
# Run 10,000 iterations
sim_list = map(1:1e4, fun_iter)
```

Parallelized 10,000 iterations

```
# Packages
p_load(purrr, furrr)
# Set options
set.seed(123)
# Tell R to parallelize
plan(multisession)
# Run 10,000 iterations
sim_list = future_map(
  1:1e4, fun_iter,
  .options = furrr_options(seed = T)
)
```

The `furrr` package (`future` + `purrr`) makes parallelization **easy and fun**!😸

Simulation

Run it!

Run our function 10,000 times

```r
# Packages
p_load(purrr)
# Set seed
set.seed(12345)
# Run 10,000 iterations
sim_list = map(1:1e4, fun_iter)
```

Parallelized 10,000 iterations

```r
# Packages
p_load(purrr, furrr)
# Set options
set.seed(123)
# Tell R to parallelize
plan(multisession)
# Run 10,000 iterations
sim_list = future_map(
  1:1e4, fun_iter,
  .options = furrr_options(seed = T)
)
```

The `furrr` package (`future` + `purrr`) makes parallelization **easy and fun**!😸

Note Use `multisession` or `multicore` instead of `multiprocess`.

Run it!!

Our `fun_iter()` function returns a `data.frame`, and `future_map()` returns a `list` (of the returned objects).

So `sim_list` is going to be a `list` of `data.frame` objects. We can bind them into one `data.frame` with `bind_rows()`.

```
# Bind list together
sim_df = bind_rows(sim_list)
```

Simulation

Run it!!

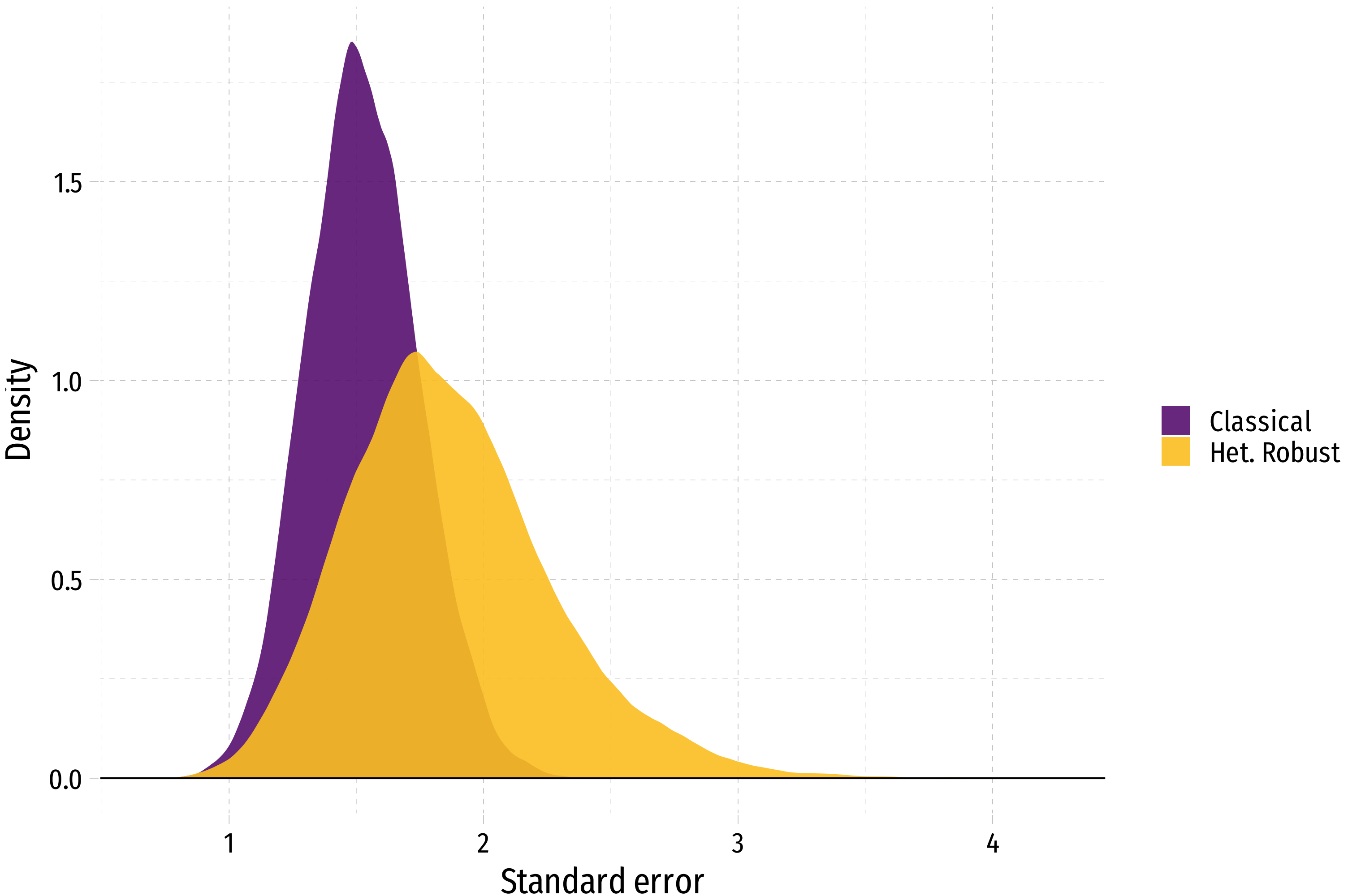
Our `fun_iter()` function returns a `data.frame`, and `future_map()` returns a `list` (of the returned objects).

So `sim_list` is going to be a `list` of `data.frame` objects. We can bind them into one `data.frame` with `bind_rows()`.

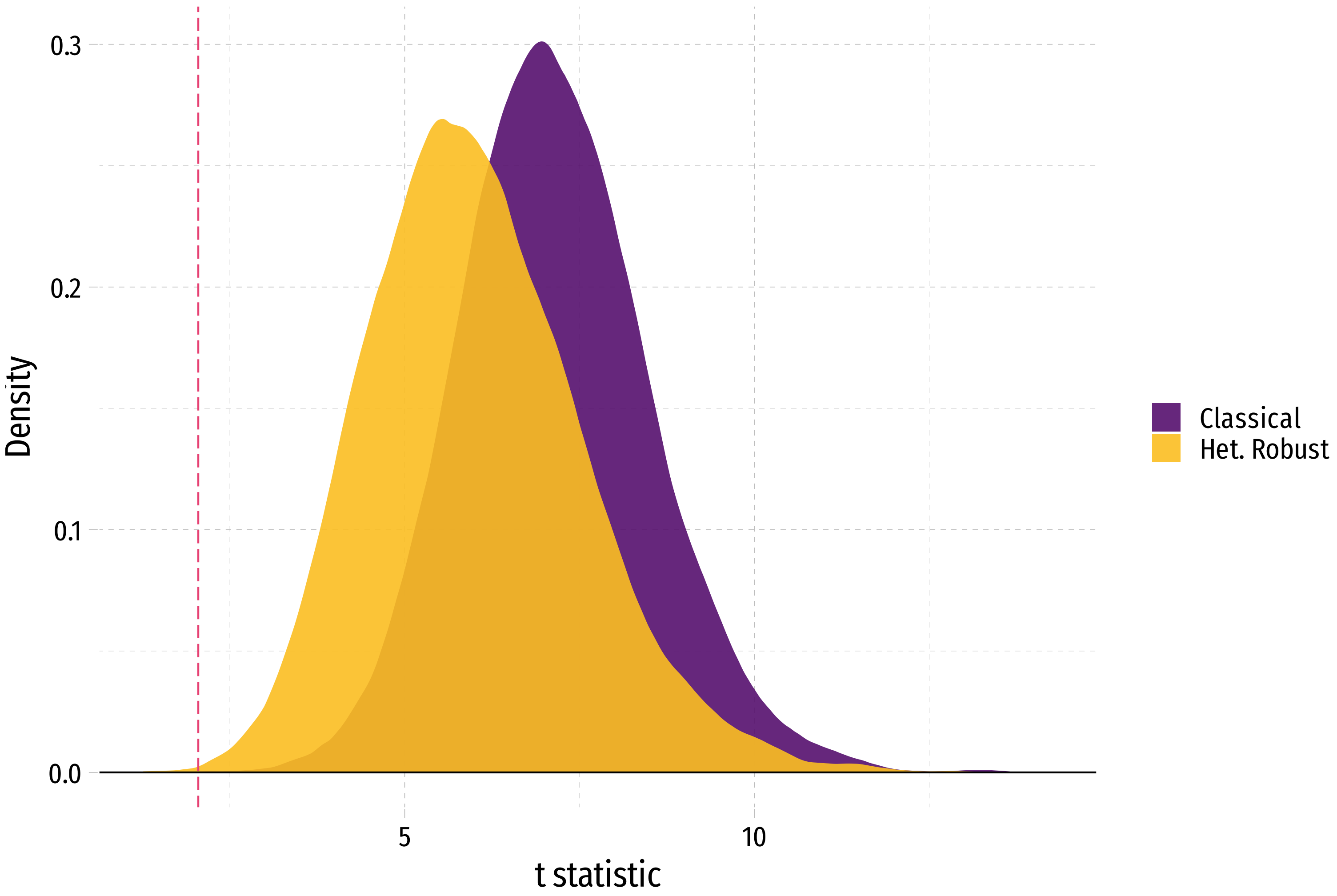
```
# Bind list together
sim_df = bind_rows(sim_list)
```

So what are the results?

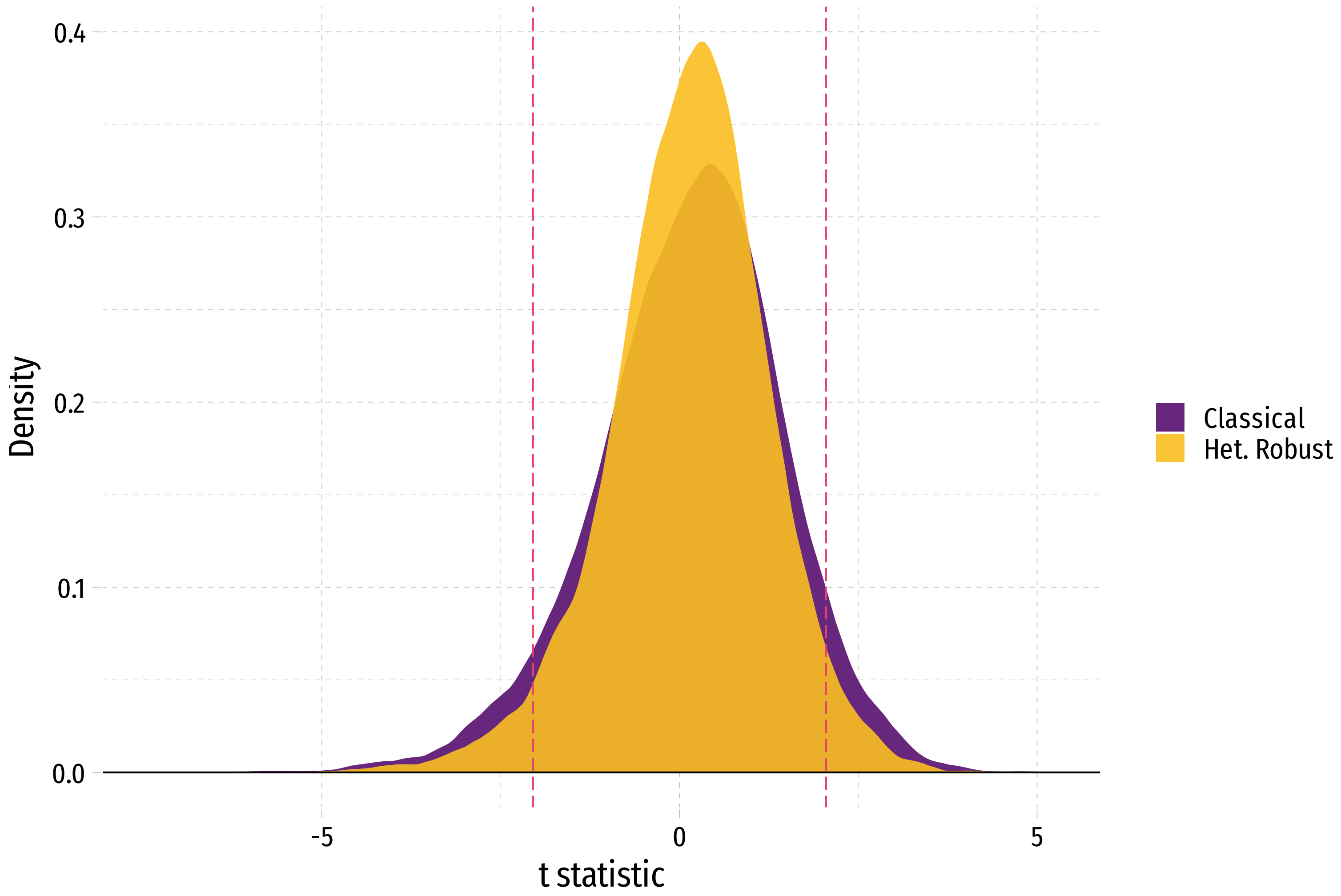
Comparing the distributions of standard errors for the coefficient on x



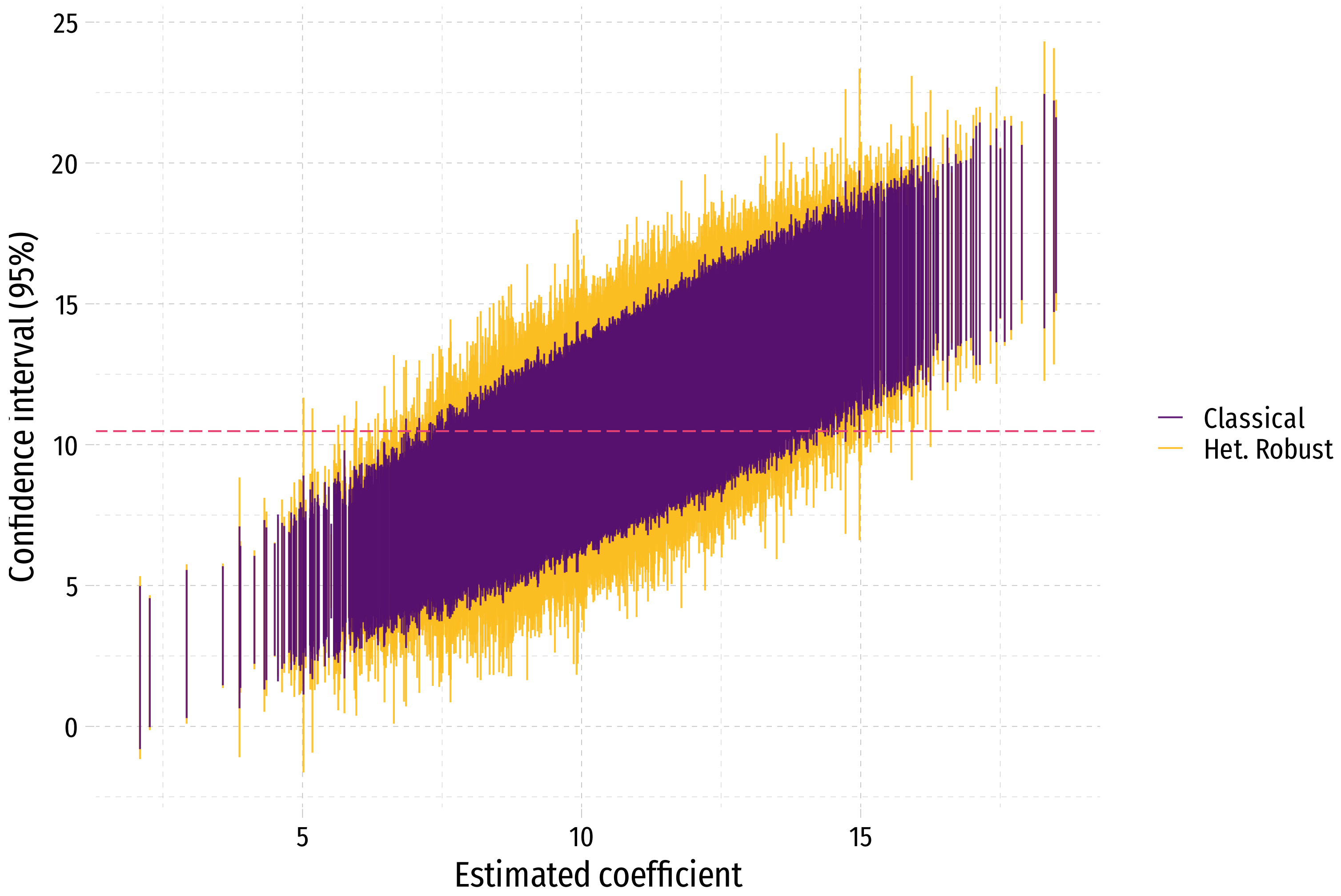
Comparing distributions of t stats for the coefficient on x $(H_o: \ \beta_1 = 0)$



Comparing distributions of t stats for the coefficient on x $(H_o:\ \beta_1 = \beta)$



Comparing the confidence intervals for the coefficient on x



Simulation

How did it go?

For a 5% test the **classical** SEs

- reject the **true value** in 11.38% of samples
- reject **zero** in 99.98% of samples

For a 5% test the **het.-robust** SEs

- reject the **true value** in 6.97% of samples
- reject **zero** in 99.93% of samples

All of these test are for a false H_0.

Q How would the simulation change to enforce a *true* null hypothesis?

Simulation

Updating to enforce the null

Let's update our simulation function to take arguments `γ` and `δ` such that

$$\mathrm{Y}_i = 1 + e^{\gamma \mathrm{X}_i} + \varepsilon_i$$

where $\varepsilon_i \sim \mathrm{N}(0, \sigma^2 \mathrm{X}_i^\delta)$.

Simulation

Updating to enforce the null

Let's update our simulation function to take arguments `γ` and `δ` such that

$$\mathbf{Y}_i = 1 + e^{\gamma \mathbf{X}_i} + \varepsilon_i$$

where $\varepsilon_i \sim \mathrm{N}(0, \sigma^2 \mathrm{X}_i^{\delta})$.

In other words,

- $\gamma = 0$ implies no relationship between $\mathbf{Y}_i$ and $\mathbf{X}_i$.
- $\delta = 0$ implies homoskedasticity.

Updating to enforce the null

Updating the function…

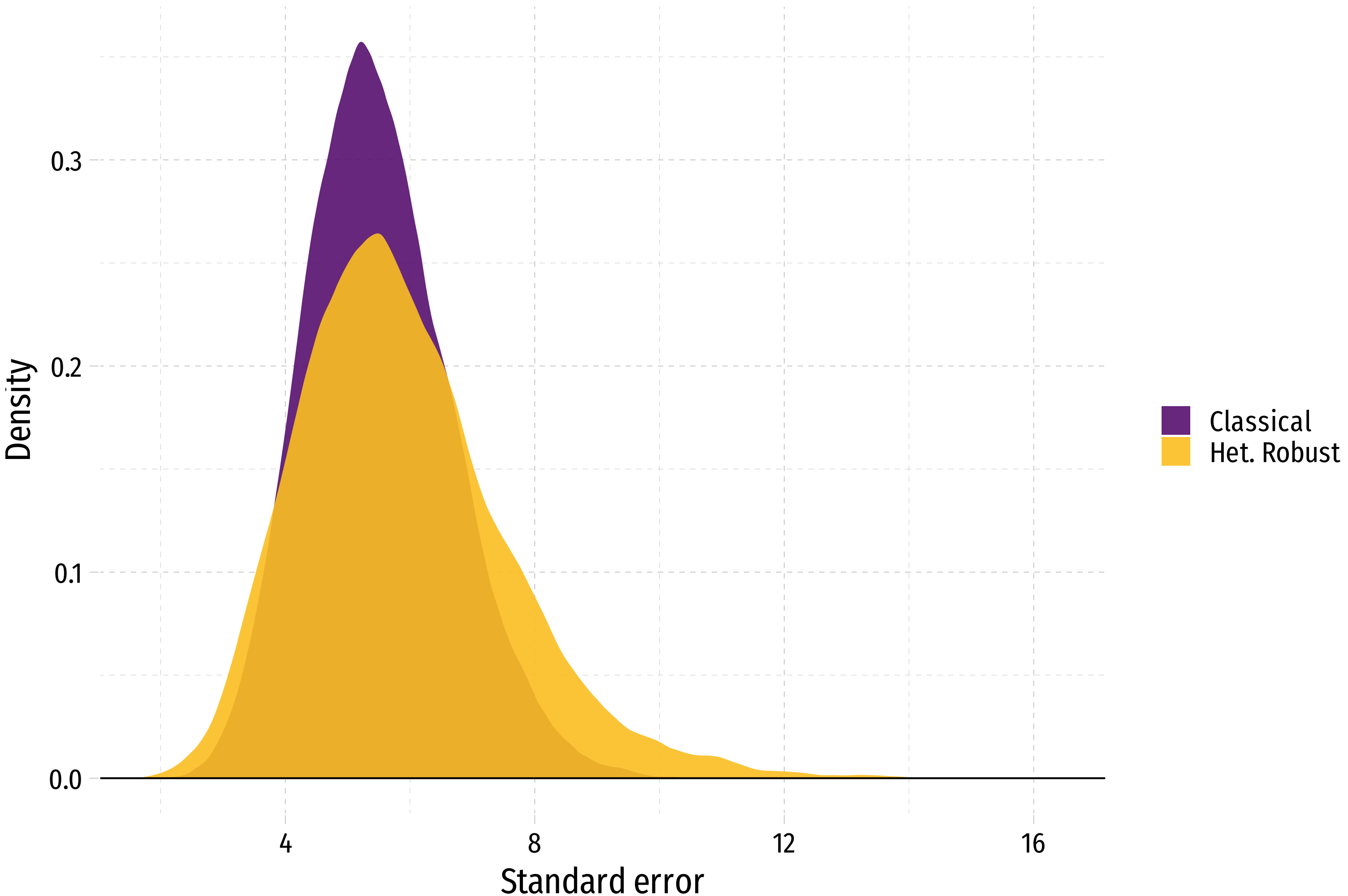
```
flex_iter = function(iter, γ = 0, δ = 1, n = 30) {
  # Generate data
  iter_df = tibble(
    x = runif(n, min = 0, max = 10),
    ε = rnorm(n, sd = 15 * x^δ),
    y = 1 + exp(γ * x) + ε
  )
  # Estimate models
  lm1 = lm_robust(y ~ x, data = iter_df, se_type = "classical")
  lm2 = lm_robust(y ~ x, data = iter_df, se_type = "HC2")
  # Stack and return results
  bind_rows(tidy(lm1), tidy(lm2)) %>%
    select(1:5) %>% filter(term == "x") %>%
    mutate(se_type = c("classical", "HC2"), i = iter)
}
```

Run again!

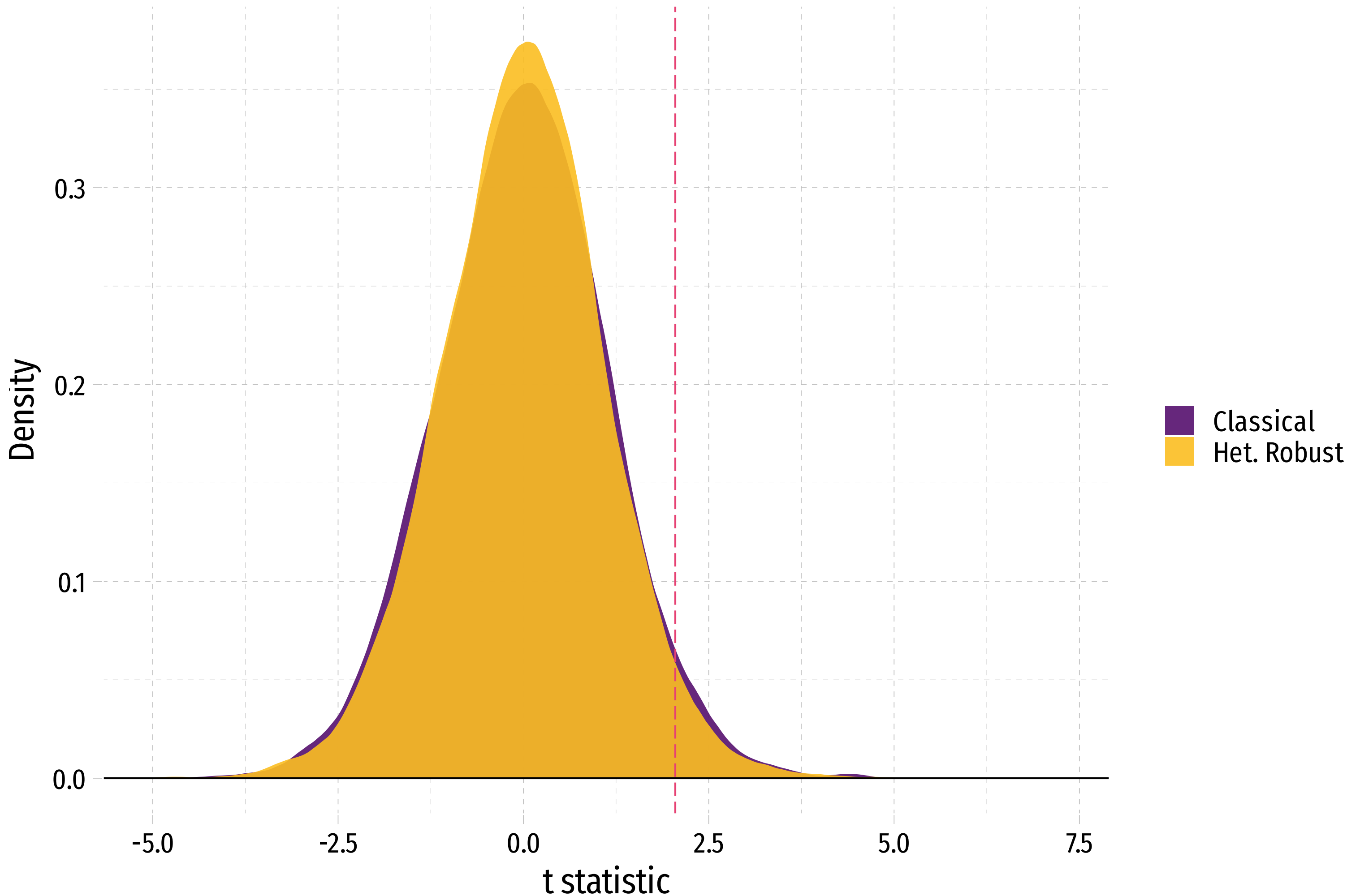
Now we run our new function `flex_iter()` 10,000 times

```
# Packages
p_load(purrr, furrr)
# Set options
set.seed(123)
# Tell R to parallelize
plan(multisession)
# Run 10,000 iterations
null_df = future_map(
  1:1e4, flex_iter,
  # Enforce the null hypothesis
  γ = 0,
  # Specify heteroskedasticity
  δ = 1,
  .options = furrr_options(seed = T)
) %>% bind_rows()
```

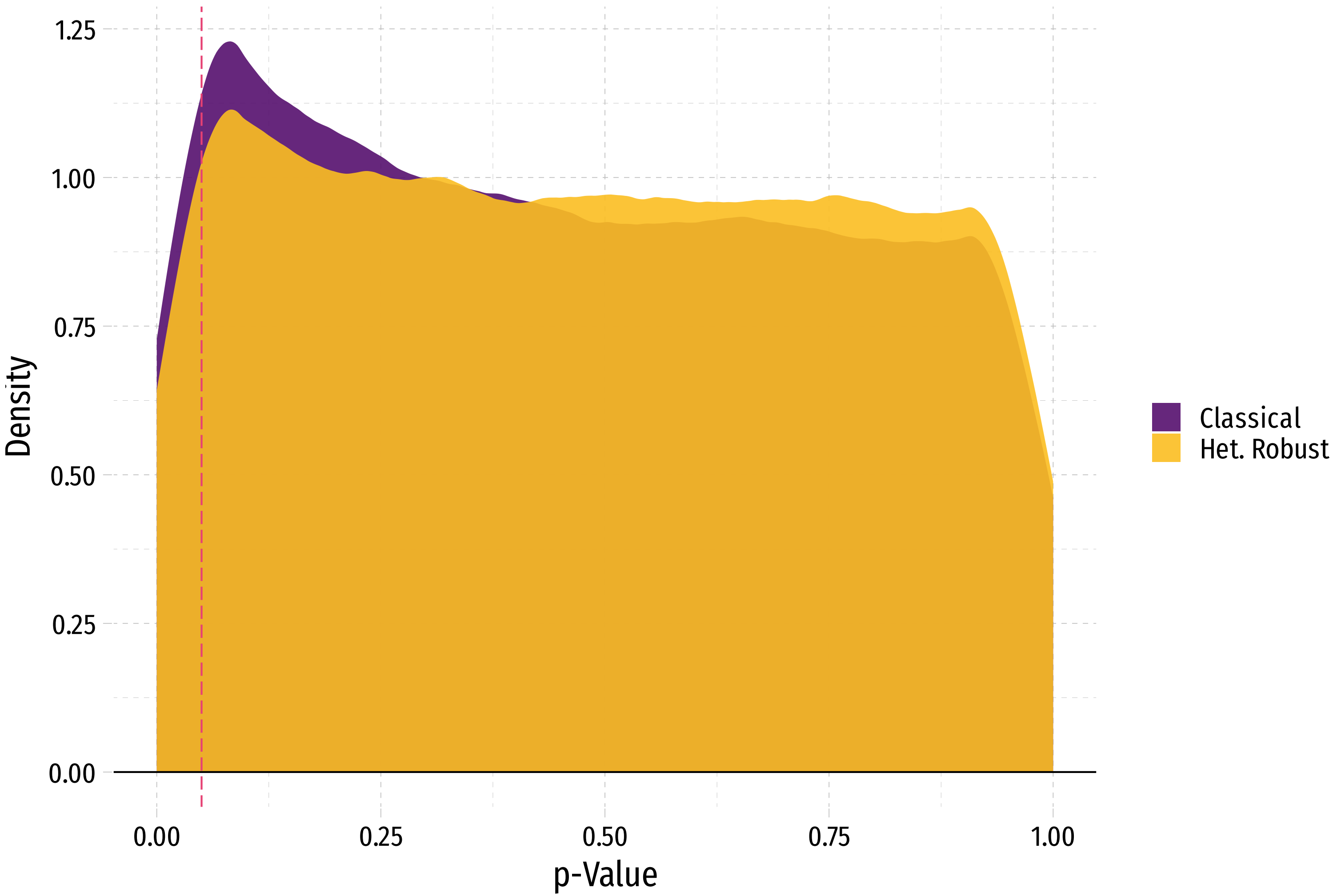
Comparing the distributions of standard errors for the coefficient on x



Comparing the distributions of t statistics for the coefficient on x



Distributions of *p*-values: both methods slightly over-reject the (true) null



Simulation

How did it go? (The sequel)

For a 5% test

- the **classical** SEs reject the **true value (zero)** in 7.73% of samples;
- the **het.-robust** SEs reject the **true value (zero)** in 6.68% of samples.

In this setting,

- **over-rejection** of the **true null** is a bit worse with IID SE estimator;
- **false precision** is *much worse*.

Summary

Wrapping up

While research often ignores it,
inference is just as important as identification.

Without understanding our **uncertainty** and the **population** onto which we draw inference, how can we learn anything from point estimates?

Summary

Wrapping up

While research often ignores it,
inference is just as important as identification.

Without understanding our **uncertainty** and the **population** onto which we draw inference, how can we learn anything from point estimates?

(Enter simulation)

Simulation is a fantastic tool for understanding estimators' behaviors.

Keep in mind: Simulation results impose (more) assumptions.

Table of contents