Tarea 2

```r
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------------------------------

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ---------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(tidymodels)
```

```
## -- Attaching packages ------------------------------------------------------------------------------

## v broom     0.7.0      v recipes   0.1.13
## v dials     0.0.8      v rsample   0.0.7
## v infer     0.5.3      v tune      0.1.1
## v modeldata 0.0.2      v workflows 0.1.3
## v parsnip   0.1.3      v yardstick 0.0.7

## -- Conflicts ---------------------------------------------------------------------------------------
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
```

```r
install.packages("Deriv")
```

```
## Installing package into '/home/rstudio-user/R/x86_64-pc-linux-gnu-library/4.0'
## (as 'lib' is unspecified)
```

```r
library(Deriv)
```

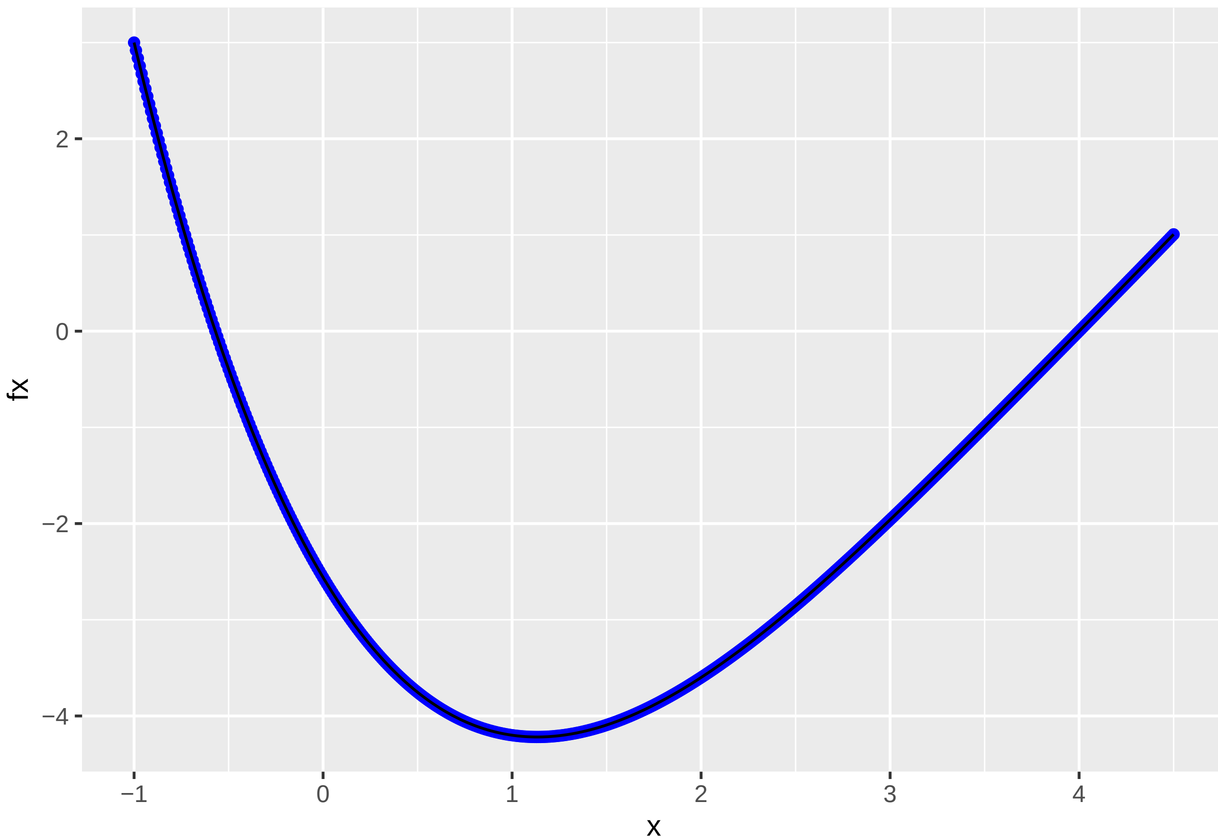
Parte 1: descenso en gradiente

Resolveremos un problema de minimización usando descenso en gradiente. Considera la siguiente función

```r
h <- function(x) {
  w <- x - 4
  (1/50) * (w^4 + w^2 + 100 * w)
}
```

Pregunta. grafica la función h. ¿Dónde está el mínimo? ¿Cuál es el valor que toma la función en el mínimo?

```r
## Creating values that will be evaluated in the function.
x <- seq(-1, 4.5, by=0.01)

## Evaluating created values in function
fx <- h(x)

## Merging results in dataframe
dfx <- data.frame(x, fx)
# print(dfx)

## Graphing the results
ggplot(
  data=dfx,
  aes(
    x=x,
    y=fx
  )
) +
  geom_point(color='blue') +
  geom_line(color='black')
```




```r
## Printing results for the minimum and the maximum value
print('El valor mínimo de la función se encuentra en: ')
```

```
## [1] "El valor mínimo de la función se encuentra en: "
```

```r
print(min(fx))
```

```
## [1] -4.218332
```

```r
print('Este valor se obtiene cuando la variable independiente toma el valor de: ')
```

```
## [1] "Este valor se obtiene cuando la variable independiente toma el valor de: "
```

```r
print(dfx$x[dfx$fx==min(fx)])
```

```
## [1] 1.13
```

Searching for new ways to obtain a function's derivative

```r
funex <- function(x){
  x^2
}
# fnx <- expression(x^2)

xs <- seq(1, 10, 1)
print(xs)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
fs <- funex(xs)
print(fs)
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```r
# x <- seq(1, 5, 1)
# dfunex <- D(fnx, 'x')
# print(dfunex)
# eval(dfunex)

dfunex <- Deriv(funex)
print(dfunex)
```

```
## function (x)
## 2 * x
```

```r
print(dfunex(xs))
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

Pregunta: Ahora calcula la derivada en al siguiente función (rellena):

```r
# calcula la derivada
h_deriv <- Deriv(h)

# h_deriv <- function(x) {
#   # aquí tu calculo, debe regresar la derivada de h
#   # evaluada en x
#   # deriv <-
#   # deriv(x, 'x')
#
# }
```

Usaremos el código de la clase anterior:

```r
# función para descenso
descenso <- function(n, z_0, eta, h_deriv){
  z <- matrix(0, n, length(z_0))
  z[1, ] <- z_0
  for(i in 1:(n-1)){
    # paso de descenso
    z[i+1, ] <- z[i, ] - eta * h_deriv(z[i, ])
  }
  z
}
```

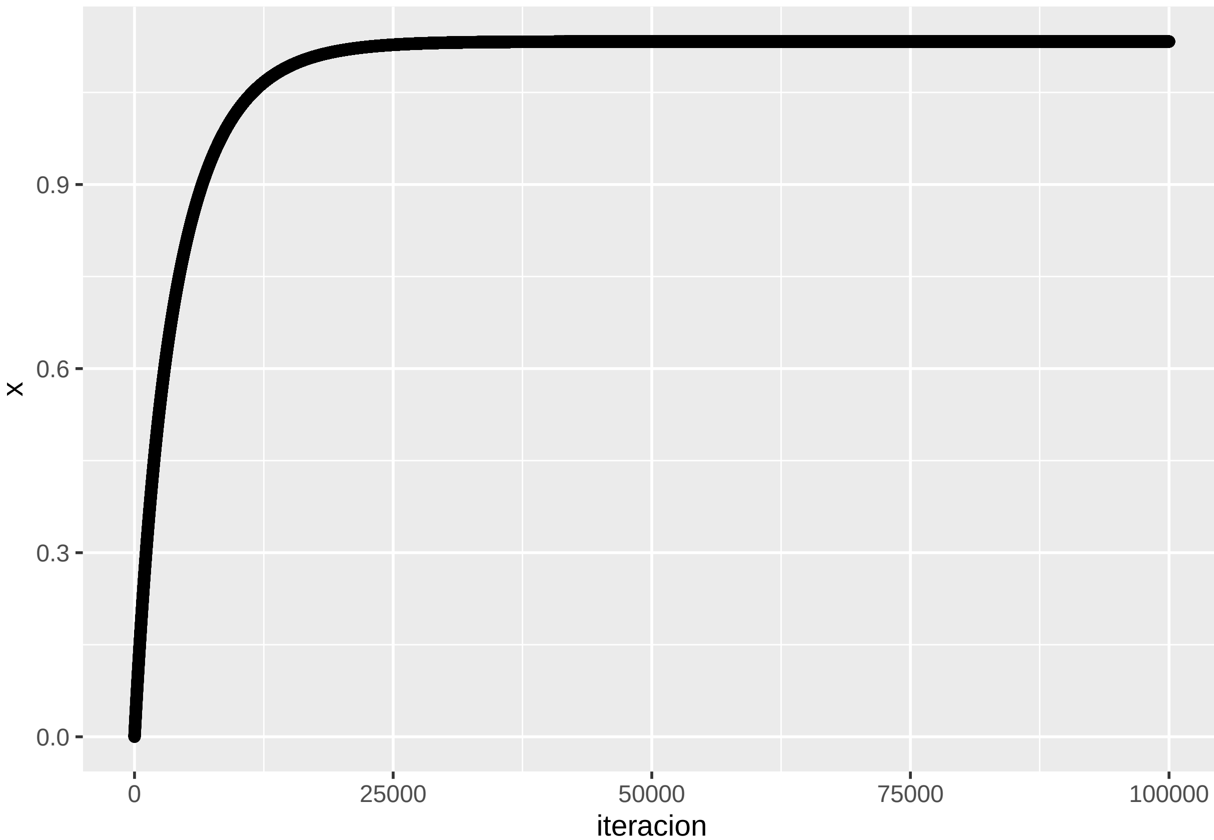
Pregunta: Empezamos las iteraciones en 0. En el siguiente código, escoge un tamaño de paso (*eta*) demasiado grande (diverge), demasiado chico (tarda mucho en converger) y uno intermedio donde alcances el mínimo en un número de iteraciones razonable. Puede ser necesario que ajustes el número de iteraciones, y puede ser que obtengas desbordes si pones tamaños de paso demasiado grandes:

Prueba con un valor de *eta* demasiado pequeño

```r
# Pon un valor grande y uno chico para eta. ¿Qué pasa?
z_0 <- 0
eta <- 0.0001 # ajusta aquí
z <- descenso(100000, z_0, eta, h_deriv)

## Creación de tabla con valores numéricos
dat_iteraciones <- tibble(iteracion = 1:nrow(z),
                          x = z[, 1], y = h(z[, 1]))
print(tail(dat_iteraciones))
```

```
## # A tibble: 6 x 3
##   iteracion     x     y
##       <int> <dbl> <dbl>
## 1     99995  1.13 -4.22
## 2     99996  1.13 -4.22
## 3     99997  1.13 -4.22
## 4     99998  1.13 -4.22
## 5     99999  1.13 -4.22
## 6    100000  1.13 -4.22
```

```r
## Grafica las iteraciones
ggplot(dat_iteraciones, aes(x = iteracion, y = x)) +
  geom_point() + geom_line()
```



En esta iteración podemos observar que toma un aproximado de 40k iteraciones llegar al resultado que estamos buscando.

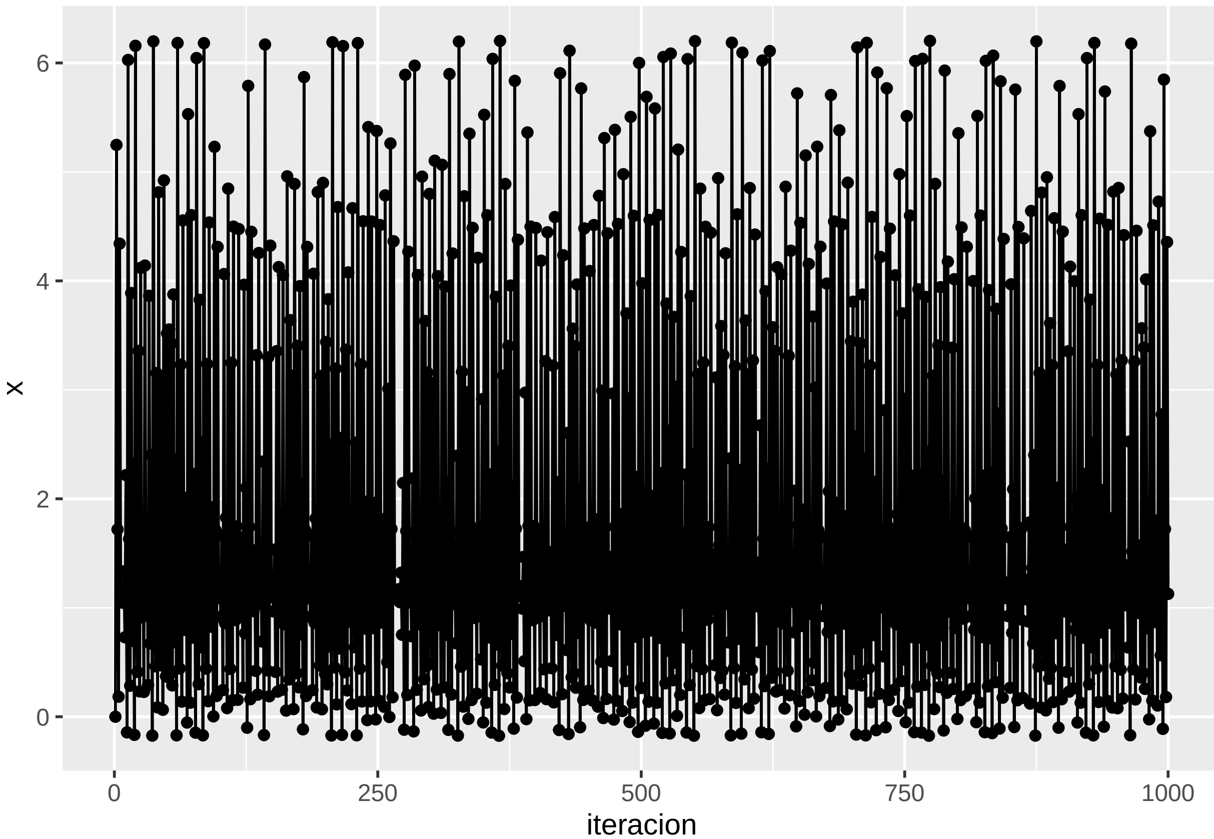
Prueba con un valor de *eta* demasiado grande

```
# Pon un valor grande y uno chico para eta. ¿Qué pasa?
z_0 <- 0
eta <- 1.6 # ajusta aquí
z <- descenso(1000, z_0, eta, h_deriv)

## Creación de tabla con valores numéricos
dat_iteraciones <- tibble(iteracion = 1:nrow(z),
                          x = z[, 1], y = h(z[, 1]))
print(tail(dat_iteraciones))
```

```
## # A tibble: 6 x 3
##   iteracion      x      y
##       <int>  <dbl>  <dbl>
## 1       995 -0.111 -2.17
## 2       996  5.85   4.00
## 3       997  1.72  -3.91
## 4       998  0.181 -3.09
## 5       999  4.36   0.715
## 6      1000  1.13  -4.22
```

```
## Grafica las iteraciones
ggplot(dat_iteraciones, aes(x = iteracion, y = x)) +
```

```r
  geom_point() + geom_line()
```



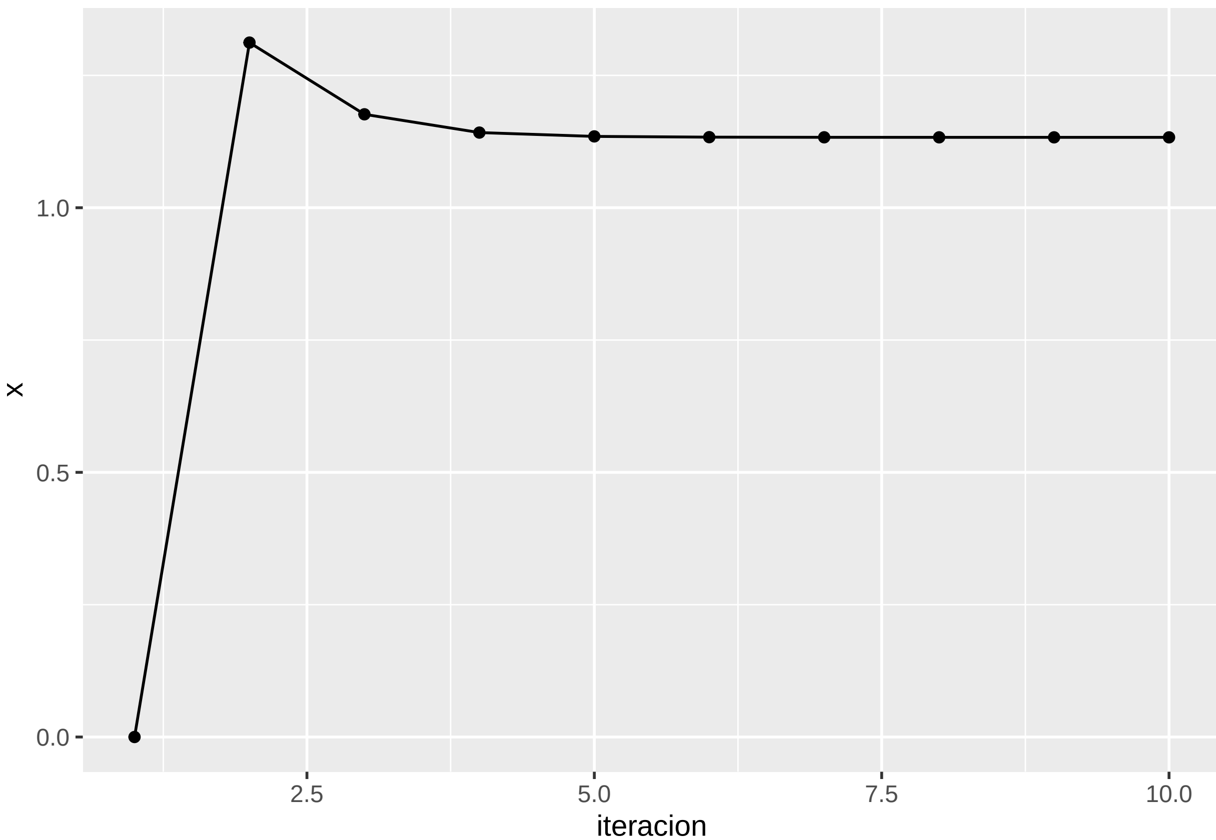
Aquí observamos un comportamiento divergente de los resultados y no se logra ver una estabilización después de las 1k iteraciones.

Prueba con un valor de *eta* demasiado grande

```r
# Pon un valor grande y uno chico para eta. ¿Qué pasa?
z_0 <- 0
eta <- 0.4 # ajusta aquí
z <- descenso(10, z_0, eta, h_deriv)

## Creación de tabla con valores numéricos
dat_iteraciones <- tibble(iteracion = 1:nrow(z),
                          x = z[, 1], y = h(z[, 1]))
print(tail(dat_iteraciones))
```

```
## # A tibble: 6 x 3
##   iteracion     x     y
##       <int> <dbl> <dbl>
## 1         5  1.13 -4.22
## 2         6  1.13 -4.22
## 3         7  1.13 -4.22
## 4         8  1.13 -4.22
## 5         9  1.13 -4.22
## 6        10  1.13 -4.22
```

```r
## Grafica las iteraciones
ggplot(dat_iteraciones, aes(x = iteracion, y = x)) +
  geom_point() + geom_line()
```



A diferencia de los últimos dos ejemplos, aquí vemos que el modelo logra llegar al resultado final en aproximadamente 5 iteraciones.

Parte 2: descenso en gradiente para regresión

Pregunta: en el siguiente ejercicio ajustamos un modelo de regresión para el problema de precio de casas que vimos en clase. Escoge

1. Un número de iteraciones y tamaño de paso adecuados para encontrar los coeficientes,
2. ¿Qué pasa si pones un tamaño de paso muy chico? ¿Qué pasa si pones un tamaño de paso demasiado grande?

Cargamos los datos:

```r
casas_receta <- read_rds("casas_receta.rds")
print(casas_receta)
```

```
## Data Recipe
##
## Inputs:
```

```
##
##        role #variables
##        none          3
##     outcome          1
##   predictor          2
```

```
x_ent <- casas_receta %>%
  prep %>%
  juice %>%
  select(tiene_piso_2, calidad) %>%
  as.matrix()
print(head(x_ent))
```

```
##      tiene_piso_2 calidad
## [1,]            1       1
## [2,]            1       1
## [3,]            0       2
## [4,]            0       0
## [5,]            0       0
## [6,]            0      -2
```

```
y_ent <- casas_receta %>%
  prep %>%
  juice %>%
  pull(precio_m2_miles)
print(head(y_ent))
```

```
## [1] 1.3124423 1.3469961 1.9507213 1.3487108 1.5974336 0.7474941
```

Pregunta: qué contienen x_ent y y_ent? - La variable 'x_ent' contiene las dos variables que componen el modelo de predicción: - Presencia de un segundo piso en la construcción. - Calidad de la construcción.

- La variable 'y_ent' contiene los precios por metro cuadrado de la construcción.

Ambas variables corresponden a los datos de entrenamiento para la construcción de nuestro modelo de regresión.

Definimos un modelo de regresión lineal en keras (**nota**: si quieres hacer corridas desde cero empieza con estas líneas que siguen. De otra manera vas a iterar desde el punto donde te hayas quedado en la corrida anterior):

```
library(keras)
```

```
##
## Attaching package: 'keras'

## The following object is masked from 'package:yardstick':
##
##     get_weights
```

```
n_entrena <- nrow(x_ent)
## correr desde aquí para empezar desde cero
modelo_casas <-
    keras_model_sequential() %>%
    layer_dense(units = 1,  # una sola respuesta,
    activation = "linear",# combinar variables linealmente
    kernel_initializer = initializer_constant(0), #inicializamos coefs en 0
    bias_initializer = initializer_constant(0))   #inicializamos ordenada en 0
```

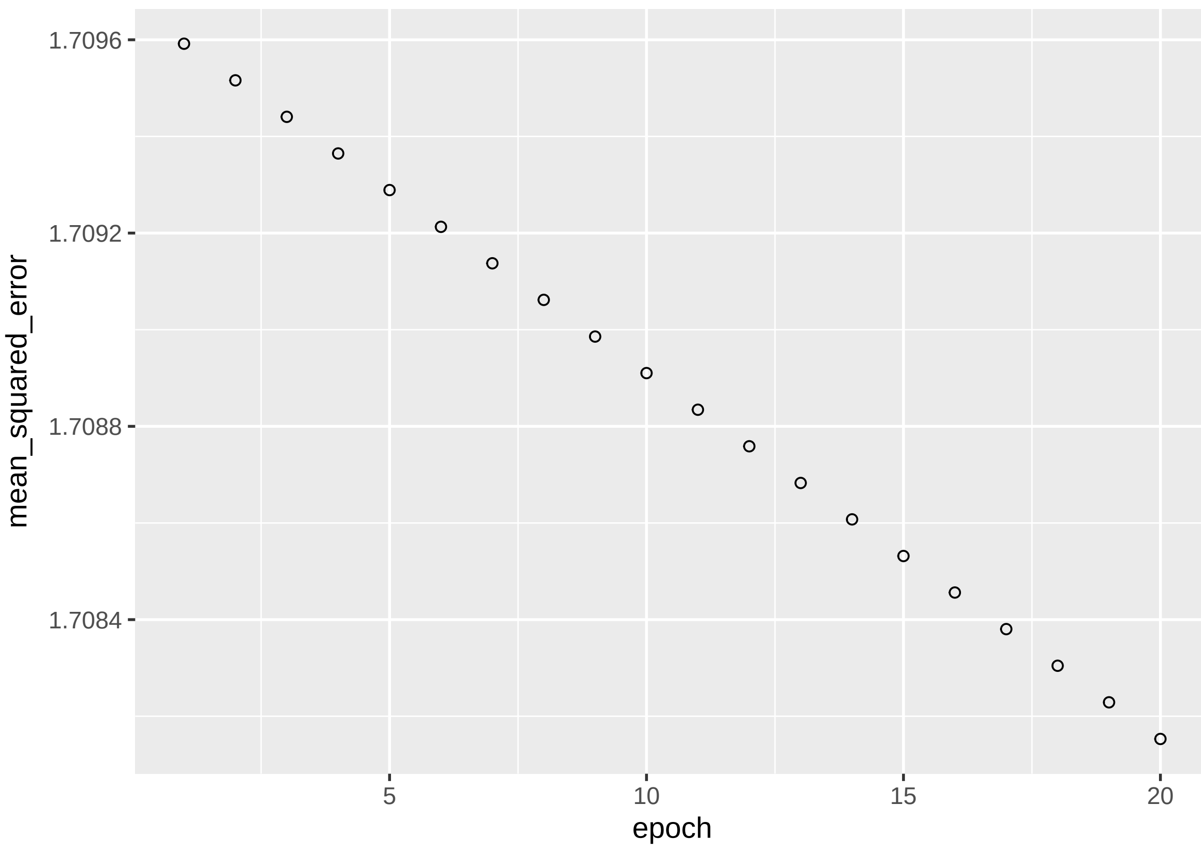
```r
# ajusta este valor, es le tamaño de paso o tasa de aprendizaje
lr <- 0.00001
```

```r
# Compilar seleccionando cantidad a minimizar, optimizador y métricas
modelo_casas %>% compile(
    loss = "mean_squared_error",  # pérdida cuadrática
    optimizer = optimizer_sgd(lr = lr), # descenso en gradiente
    metrics = list("mean_squared_error"))
```

Ahora iteramos: Primero probamos con un número bajo de iteraciones (epochs):

```r
historia <- modelo_casas %>% fit(
  x_ent, # x entradas
  y_ent,
  batch_size = nrow(x_ent), # para descenso en gradiente
  epochs = 20 # número de iteraciones
)
```

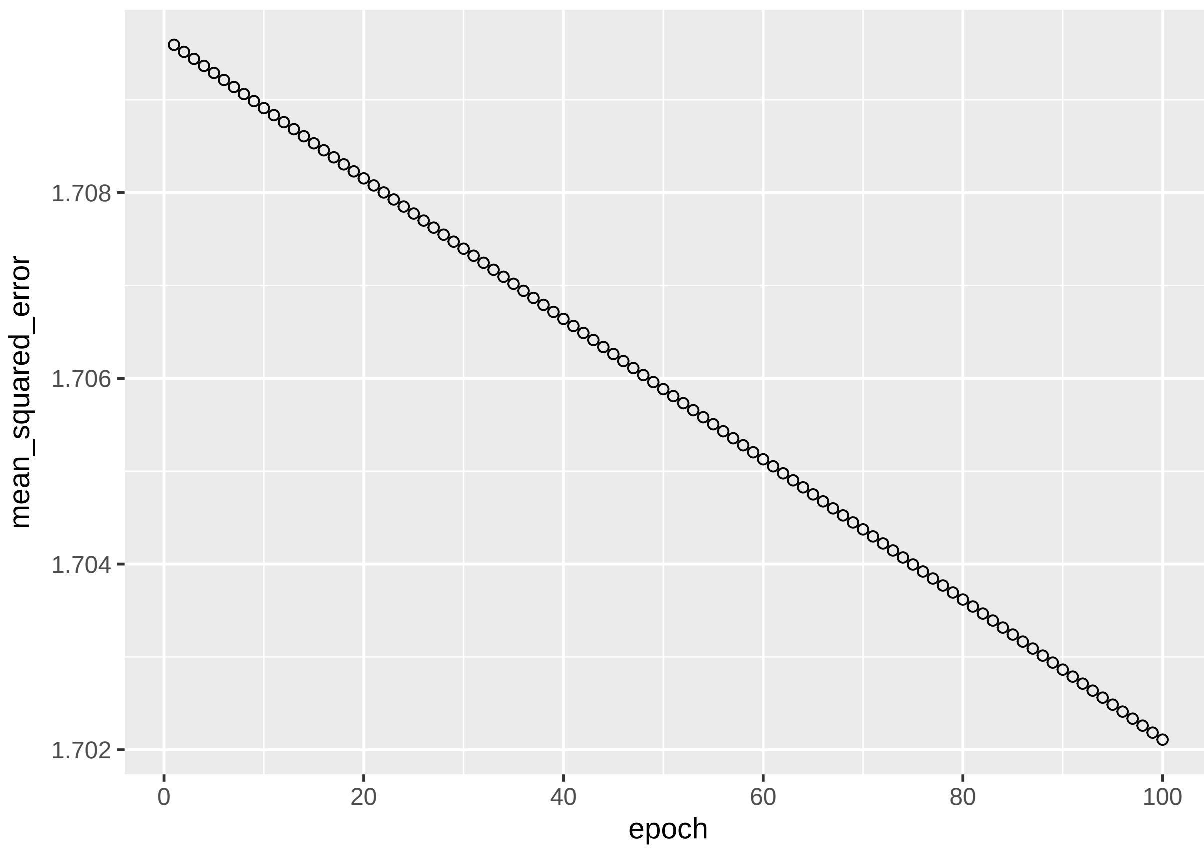
```r
plot(historia, metrics = "mean_squared_error", smooth = FALSE)
```



Prueba 1: utilizando un tamaño de paso demasiado chico

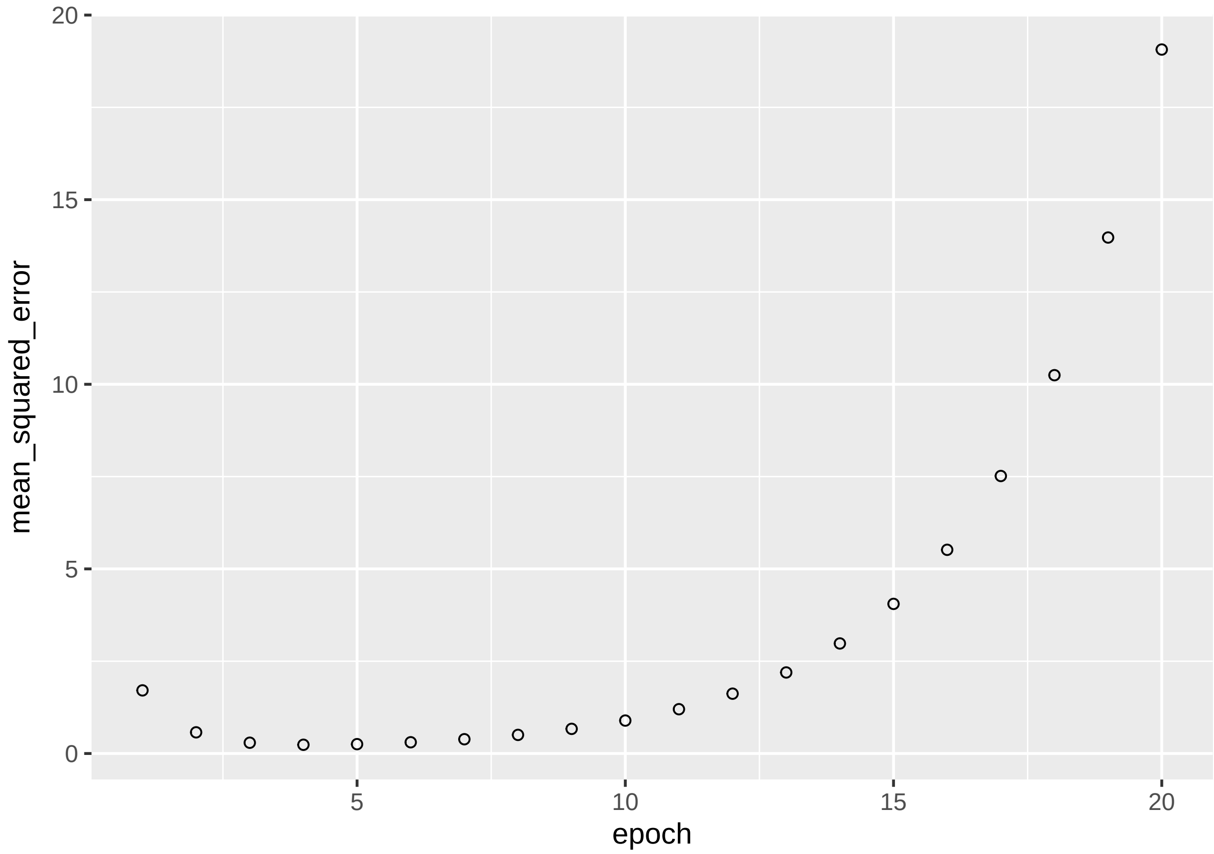
```r
library(keras)
n_entrena <- nrow(x_ent)
## correr desde aquí para empezar desde cero
modelo_casas <-
    keras_model_sequential() %>%
```

```r
    layer_dense(units = 1,  # una sola respuesta,
    activation = "linear",# combinar variables linealmente
    kernel_initializer = initializer_constant(0), #inicializamos coefs en 0
    bias_initializer = initializer_constant(0))   #inicializamos ordenada en 0


# ajusta este valor, es le tamaño de paso o tasa de aprendizaje
lr <- 0.00001


# Compilar seleccionando cantidad a minimizar, optimizador y métricas
modelo_casas %>% compile(
    loss = "mean_squared_error",  # pérdida cuadrática
    optimizer = optimizer_sgd(lr = lr), # descenso en gradiente
    metrics = list("mean_squared_error"))


historia <- modelo_casas %>% fit(
  x_ent, # x entradas
  y_ent,
  batch_size = nrow(x_ent), # para descenso en gradiente
  epochs = 100 # número de iteraciones
)


plot(historia, metrics = "mean_squared_error", smooth = FALSE)
```



Prueba 2: utilizando un tamaño de paso demasiado grande

```
library(keras)
n_entrena <- nrow(x_ent)
## correr desde aquí para empezar desde cero
modelo_casas <-
    keras_model_sequential() %>%
    layer_dense(units = 1,  # una sola respuesta,
    activation = "linear",# combinar variables linealmente
    kernel_initializer = initializer_constant(0), #inicializamos coefs en 0
    bias_initializer = initializer_constant(0))   #inicializamos ordenada en 0


# ajusta este valor, es le tamaño de paso o tasa de aprendizaje
lr <- 0.6


# Compilar seleccionando cantidad a minimizar, optimizador y métricas
modelo_casas %>% compile(
    loss = "mean_squared_error",  # pérdida cuadrática
    optimizer = optimizer_sgd(lr = lr), # descenso en gradiente
    metrics = list("mean_squared_error"))


historia <- modelo_casas %>% fit(
  x_ent, # x entradas
  y_ent,
  batch_size = nrow(x_ent), # para descenso en gradiente
  epochs = 20 # número de iteraciones
)


plot(historia, metrics = "mean_squared_error", smooth = FALSE)
```

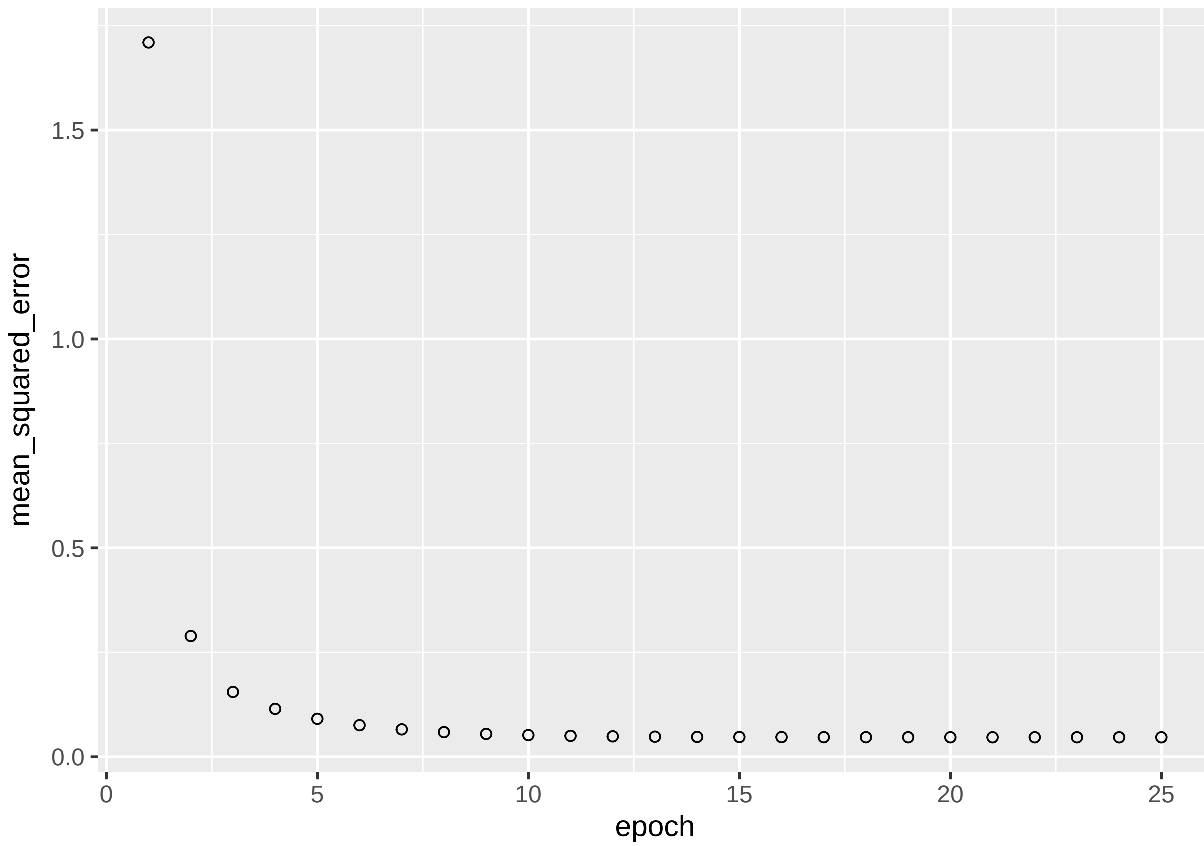


Con estos parámetros podemos ver que el error comienza a diverger después de como la 5ta época.

Prueba 3: ajustando a parámetros "adecuados"

```r
library(keras)
n_entrena <- nrow(x_ent)
## correr desde aquí para empezar desde cero
modelo_casas <-
    keras_model_sequential() %>%
    layer_dense(units = 1,  # una sola respuesta,
    activation = "linear",# combinar variables linealmente
    kernel_initializer = initializer_constant(0), #inicializamos coefs en 0
    bias_initializer = initializer_constant(0))   #inicializamos ordenada en 0


# ajusta este valor, es le tamaño de paso o tasa de aprendizaje
lr <- 0.5


# Compilar seleccionando cantidad a minimizar, optimizador y métricas
modelo_casas %>% compile(
    loss = "mean_squared_error",  # pérdida cuadrática
    optimizer = optimizer_sgd(lr = lr), # descenso en gradiente
    metrics = list("mean_squared_error"))
```

```r
historia <- modelo_casas %>% fit(
  x_ent, # x entradas
  y_ent,
  batch_size = nrow(x_ent), # para descenso en gradiente
  epochs = 25 # número de iteraciones
)


# print(tail(historia))


plot(historia, metrics = "mean_squared_error", smooth = FALSE)
```



Los resultados de esta prueba logran converger el error a un valor muy cercano a cero.