TRABAJO PROGRAMACION NUMERICA

Codigo en “R” hallar la funcion para llegar a la raíz

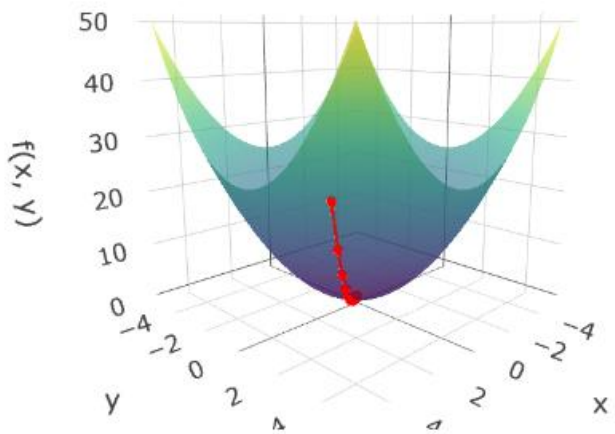
MÉTODO DEL GRADIENTE CON GRÁFICO

```
# Instalar librerías si no las tienes
if(!require(plotly)) install.packages("plotly", dependencies = TRUE)
library(plotly)

# --- Definir la función y el gradiente ---
f <- function(x, y) {
  return(x^2 + y^2)
}

grad_f <- function(x, y) {
  gx <- 2 * x
  gy <- 2 * y
  return(c(gx, gy))
}

# --- Parámetros iniciales ---
x <- 4
y <- 3
alpha <- 0.1   # tasa de aprendizaje
tol <- 1e-6
max_iter <- 200

# --- Listas para guardar el recorrido ---
trayectoria <- data.frame(x = x, y = y, z = f(x, y))
```

```
# --- Iteraciones ---
for (i in 1:max_iter) {
  grad <- grad_f(x, y)

  # Actualizar
  x_new <- x - alpha * grad[1]
  y_new <- y - alpha * grad[2]

  # Guardar punto
  trayectoria <- rbind(trayectoria, data.frame(x = x_new, y = y_new, z = f(x_new, y_new)))

  # Comprobar convergencia
  if (sqrt((x_new - x)^2 + (y_new - y)^2) < tol) {
    cat("Convergencia alcanzada en iteración", i, "\n")
    break
  }

  # Actualizar variables
  x <- x_new
  y <- y_new
}

cat("Mínimo aproximado en (x, y) =", round(x, 6), ",", round(y, 6), "\n")
cat("Valor mínimo f(x, y) =", round(f(x, y), 6), "\n")

# --- Crear superficie 3D ---
```

```
x_seq <- seq(-5, 5, length.out = 50)

y_seq <- seq(-5, 5, length.out = 50)

z_mat <- outer(x_seq, y_seq, f)


# --- Graficar con plotly ---

fig <- plot_ly() %>%

  add_surface(x = ~x_seq, y = ~y_seq, z = ~z_mat, opacity = 0.6, showscale = FALSE)
%>%

  add_trace(

    x = trayectoria$x,

    y = trayectoria$y,

    z = trayectoria$z,

    type = "scatter3d",

    mode = "lines+markers",

    line = list(color = "red", width = 4),

    marker = list(size = 3, color = "red"),

    name = "Recorrido del gradiente"

  ) %>%

  layout(

    title = "Método del Gradiente Descendente",

    scene = list(

      xaxis = list(title = "x"),

      yaxis = list(title = "y"),

      zaxis = list(title = "f(x, y)")

    )

  )


Fig
```

GRAFICO:



Ahora con OLS (mínimos cuadrados) y Gradiente Descendente (GD) en R con 100 000 datos

COMPARACIÓN DE OLS vs GRADIENTE DESCENDENTE (100000 datos)

Instalar librerías si no las tienes

if(!require(profvis)) install.packages("profvis", dependencies = TRUE)

if(!require(plotly)) install.packages("plotly", dependencies = TRUE)

library(profvis)

library(plotly)

set.seed(123)

n <- 100000 # 100 mil datos

x <- runif(n, 150, 200) # talla en cm

error <- rnorm(n, 0, 3)

y <- 0.5 * x - 60 + error # peso en kg (relación lineal simulada)

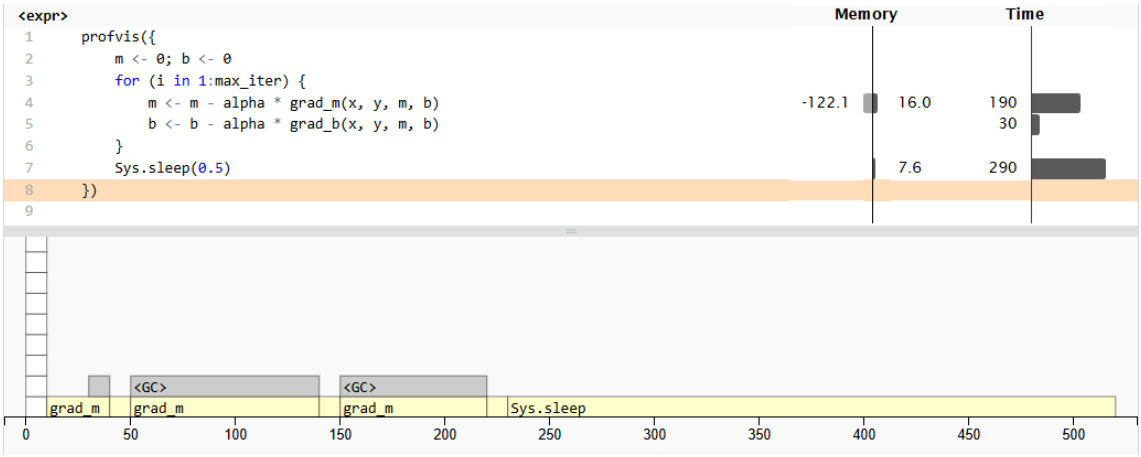
```
cat("\n--- MÉTODO OLS ---\n")

t_ols <- system.time({
  modelo_ols <- lm(y ~ x)
})

summary(modelo_ols)
cat("Tiempo OLS:", round(t_ols["elapsed"], 4), "segundos\n")

# Perfilado con profvis
profvis({
  modelo_ols_temp <- lm(y ~ x)
  Sys.sleep(0.5)  # ayuda a que profvis registre los datos
})

cat("\n--- MÉTODO DEL GRADIENTE DESCENDENTE ---\n")

# Función de costo
coste <- function(x, y, m, b) {
  n <- length(y)
  pred <- m * x + b
  return(sum((y - pred)^2) / (2 * n))
}

# Gradientes parciales
grad_m <- function(x, y, m, b) {
  n <- length(y)
```

```
  return(-sum(x * (y - (m * x + b))) / n)
}
grad_b <- function(x, y, m, b) {
  n <- length(y)
  return(-sum(y - (m * x + b)) / n)
}

# Parámetros iniciales
m <- 0
b <- 0
alpha <- 0.0000001   # tasa de aprendizaje (ajustada para 100000 datos)
max_iter <- 200

# Guardar historia
historia <- data.frame(iter = 0, m = m, b = b, costo = coste(x, y, m, b))

# Medir tiempo de ejecución
t_gd <- system.time({
  for (i in 1:max_iter) {
    m <- m - alpha * grad_m(x, y, m, b)
    b <- b - alpha * grad_b(x, y, m, b)
    historia <- rbind(historia, data.frame(iter = i, m = m, b = b, costo = coste(x, y, m, b)))
  }
})

cat("Parámetros finales del GD:\n")
cat("m =", round(m, 4), ", b =", round(b, 4), "\n")
cat("Costo final =", round(coste(x, y, m, b), 4), "\n")
```

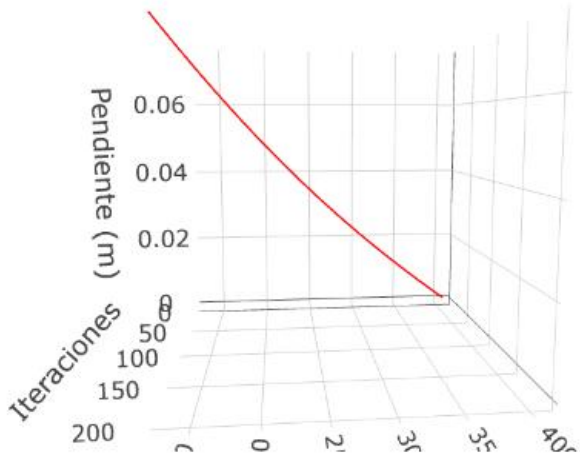
```
cat("Tiempo GD:", round(t_gd["elapsed"], 4), "segundos\n")


# Perfilado con profvis
profvis({
  m <- 0; b <- 0
  for (i in 1:max_iter) {
    m <- m - alpha * grad_m(x, y, m, b)
    b <- b - alpha * grad_b(x, y, m, b)
  }
  Sys.sleep(0.5)
})


plot(x, y, col = rgb(0.7, 0.7, 0.7, 0.5), pch = 16,
     main = "Comparación OLS vs Gradiente Descendente",
     xlab = "Talla (cm)", ylab = "Peso (kg)")
abline(modelo_ols, col = "blue", lwd = 2)
abline(a = b, b = m, col = "red", lwd = 2)
legend("topleft", legend = c("OLS", "Gradiente Descendente"), col = c("blue", "red"),
lwd = 2)


fig <- plot_ly(historia, x = ~iter, y = ~costo, z = ~m,
               type = 'scatter3d', mode = 'lines',
               line = list(color = 'red')) %>%
  layout(title = "Convergencia del Gradiente Descendente",
         scene = list(
           xaxis = list(title = "Iteraciones"),
           yaxis = list(title = "Costo"),
           zaxis = list(title = "Pendiente (m)")
```

))

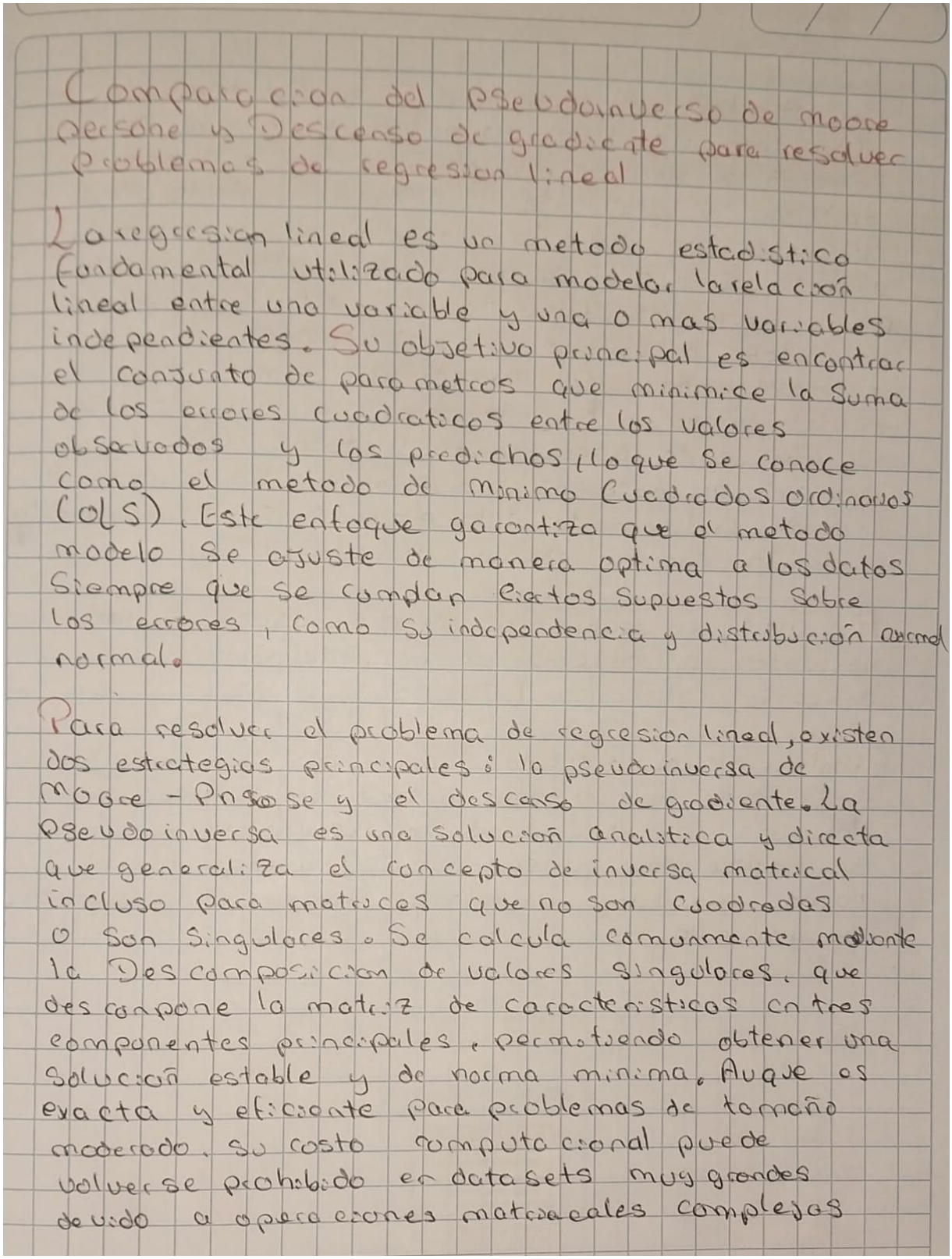
Fig


<expr>
Memory
Time
profvis({
m <- 0; b <- 0
for (i in 1:max_iter) {
m <- m - alpha * grad_m(x, y, m, b)
b <- b - alpha * grad_b(x, y, m, b)
}
Sys.sleep(0.5)
})
-122.1
16.0
190
30
7.6
290
<GC>
<GC>
grad_m
grad_m
grad_m
Sys.sleep
0
50
100
150
200
250
300
350
400
450
500

GRAFICA


Pendiente (m)
0.06
0.04
0.02
0
Iteraciones
0
50
100
150
200

RESUMEN :



Comparación del pseudoinverso de moore penrose vs Descenso de gradiente para resolver problemas de regresion lineal

La regresion lineal es un metodo estadistico fundamental utilizado para modelar la relación lineal entre una variable y una o mas variables independientes. Su objetivo principal es encontrar el conjunto de parametros que minimice la suma de los errores cuadraticos entre los valores observados y los predichos, lo que se conoce como el metodo de minimo cuadrados ordinarios (OLS). Este enfoque garantiza que el metodo modelo se ajuste de manera optima a los datos siempre que se cumplan ciertos supuestos sobre los errores, como su independencia y distribución normal normal.

Para resolver el problema de regresion lineal, existen dos estrategias principales: la pseudoinversa de Moore-Penrose y el descenso de gradiente. La pseudoinversa es una solución analitica y directa que generaliza el concepto de inversa matricial incluso para matrices que no son cuadradas o son singulares. Se calcula comunmente mediante la Descomposicion de valores singulares, que descompone la matriz de caracteristicas en tres componentes principales, permitiendo obtener una solución estable y de norma minima. Aunque es exacta y eficiente para problemas de tamaño moderado, su costo computacional puede volverse prohibido en datasets muy grandes debido a operaciones matriciales complejas

Comparación del pseudoinverso de Moore-Penrose y Descenso de gradiente para resolver problemas de regresión lineal

La regresión lineal es un método estadístico fundamental utilizado para modelar la relación lineal entre una variable y una o más variables independientes. Su objetivo principal es encontrar el conjunto de parámetros que minimice la suma de los errores cuadráticos entre los valores observados y los predichos, lo que se conoce como el método de mínimo cuadrados ordinarios (OLS). Este enfoque garantiza que el modelo se ajuste de manera óptima a los datos siempre que se cumplan ciertos supuestos sobre los errores, como su independencia y distribución normal.

Para resolver el problema de regresión lineal, existen dos estrategias principales: la pseudoinversa de Moore-Penrose y el descenso de gradiente. La pseudoinversa es una solución analítica y directa que generaliza el concepto de inversa matricial incluso para matrices que no son cuadradas o son singulares. Se calcula comúnmente mediante la Descomposición de valores singulares, que descompone la matriz de características en sus componentes principales, permitiendo obtener una solución estable y de norma mínima. Aunque es exacta y eficiente para problemas de tamaño moderado, su costo computacional puede volverse prohibido en datasets muy grandes debido a operaciones matriciales complejas

En los modelos de regresión lineal, el propósito principal es determinar los coeficientes que describen de la mejor manera la relación entre una variable dependiente (y) y una o varias variables independientes (x).
El método OLS (mínimos cuadrados ordinarios) es un enfoque analítico y exacto, mientras que el descenso del gradiente (GD) es un procedimiento numérico e iterativo.

En términos de precisión, **OLS** suele ofrecer resultados más exactos, ya que obtiene los coeficientes de forma directa mediante fórmulas matemáticas. En cambio, **GD** depende del número de iteraciones y del valor elegido para la tasa de aprendizaje, lo que puede afectar su exactitud.

Respecto a la velocidad, OLS es más eficiente con pequeños volúmenes de datos o modelos simples, pero su rendimiento disminuye cuando aumenta el tamaño del conjunto de datos, debido a que requiere la inversión de matrices grandes. El método del gradiente, por otro lado, puede manejar conjuntos de datos de gran escala sin necesidad de almacenar ni invertir matrices completas, aunque su proceso de convergencia sea más lento.

El costo computacional del OLS crece de manera cúbica conforme aumenta el tamaño de los datos, mientras que el del gradiente está ligado al número de iteraciones y a la cantidad de observaciones procesadas.

La fortaleza principal del OLS radica en su simplicidad y precisión, mientras que la del gradiente es su capacidad de adaptación y escalabilidad, lo que lo hace ideal para modelos extensos o complejos, como los usados en aprendizaje automático y redes neuronales.

No obstante, el OLS se vuelve poco eficiente con grandes volúmenes de información o datos dispersos, y el gradiente puede fallar en converger si la tasa de aprendizaje no es adecuada.

En resumen, el método de mínimos cuadrados ordinarios es más conveniente para conjuntos de datos pequeños o medianos, donde se requiere una solución rápida y exacta. En cambio, el descenso del gradiente resulta más apropiado para problemas con enormes cantidades de datos o modelos complejos, donde la inversión matricial es demasiado costosa o impracticable. En experimentos con datos simulados, OLS tiende a ser más veloz, pero el gradiente, aunque más lento, es más eficiente para trabajar con sistemas de gran escala y recursos limitados.