

Nombre: Luis Alejandro Baena  
Código: 1023628877

Nombre: Jesús David Cantellon  
Código: 1013457170



## Estadística II

### Taller II

---

## Resumen

Este trabajo aborda el proceso de descomposición del valor singular (SVD, por sus siglas en inglés) y su aplicación en la obtención de componentes principales y biplots. Se explora la conceptualización y metodología detrás de la SVD, destacando su utilidad en el análisis de datos multidimensionales. Se proporcionan ejemplos prácticos utilizando software para ilustrar cómo se pueden obtener componentes principales y biplots a partir de la descomposición del valor singular. Este enfoque ofrece una herramienta efectiva para la reducción de dimensiones y la visualización de datos complejos, siendo de relevancia para diversas áreas de investigación y aplicación práctica.

## Índice

1	Descomposición del valor singular . . . . .	2
1.1	Definición del concepto . . . . .	2
1.2	Aplicación . . . . .	3
1.2.1	Compresión de imágenes . . . . .	3
1.2.2	Desarrollo en R . . . . .	4
1.2.3	Resultados de la compresión . . . . .	7
2	Calculo de la descomposición en valores singulares . . . . .	8
2.1	Algoritmo . . . . .	8
2.2	Implementación en R . . . . .	9
3	El gráfico Biplot . . . . .	13
3.1	Biplot a partir de la descomposición del valor singular . . . . .	13
3.2	Aplicación . . . . .	14
3.2.1	Código . . . . .	19
3.3	Conclusiones . . . . .	20

# 1. Descomposición del valor singular

Esta sección aborda la descomposición del valor singular (SVD), un proceso fundamental en el análisis de datos multidimensionales. Se explora el concepto y la aplicación de la SVD, destacando su utilidad para la reducción de dimensiones y la extracción de información clave. Se presenta un ejemplo práctico utilizando software, que ilustra cómo la SVD puede descomponer una matriz de datos en componentes singulares, facilitando así la comprensión y el análisis de conjuntos de datos complejos. Este enfoque demuestra la eficacia de la SVD en la exploración y la interpretación de datos, siendo una herramienta valiosa en diversas disciplinas y aplicaciones analíticas.

## 1.1. Definición del concepto

Por el teorema del valor espectral, sabemos que toda matriz simétrica  $A \in \mathbb{R}^{n \times n}$  se puede descomponer  $A = PDP^T$ , siendo  $P$  una matriz ortogonal ( $P^T = P^{-1}$ ),  $D$  una matriz diagonal que contiene los autovalores de  $A$ .

Cuando  $A$  no es simétrica pero sí cuadrada, si  $A$  es diagonalizable existe una descomposición de  $A = SDS^{-1}$ , siendo  $S$  no singular aunque no necesariamente ortogonal. Pero, cualquier matriz no es diagonalizable. Ahora veremos que toda matriz  $A \in \mathbb{R}^{m \times n}$ ,  $m \geq n$  (cuadrada o no, simétrica o no) tiene una factorización de la forma:  $A = UDV^T$ , donde  $U \in \mathbb{R}^{m \times m}$  con columnas ortogonales,  $V \in \mathbb{R}^{n \times n}$  matriz ortogonal y  $D$  una matriz diagonal ( $n \times n$ ). Este resultado se llama "descomposición en valores singulares" (DVS), y es una de las más importantes entre las descomposiciones de matrices. Explicaremos cómo obtenerla y haremos consideraciones sobre sus aplicaciones.

Sea  $A$  una matriz real  $m \times n$  de rango  $r \leq \min(m, n)$ . Una Descomposición en Valores Singulares (SVD, por sus siglas en inglés) es una manera de factorizar  $A$  como:

$$A = U\Sigma V^T, \quad (1)$$

donde  $U$  y  $V$  son matrices ortogonales tales que  $U^T U = I_m$  y  $V^T V = I_n$ . La matriz  $\Sigma$  contiene los valores singulares de  $A$  en su pseudo-diagonal, con ceros en otro lugar. Así,

$$A = U\Sigma V^T = \underbrace{\begin{bmatrix} u_1 & | & u_2 & | & \cdots & | & u_m \end{bmatrix}}_{U(m \times m)} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \cdots & \cdots & 0 & \cdots & 0 \\ 0 & \ddots & & & \vdots & & \vdots \\ \vdots & & \sigma_r & & & \vdots & \\ 0 & & 0 & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & \ddots & 0 & \cdots & 0 \end{bmatrix}}_{\Sigma(m \times n)} \underbrace{\begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{bmatrix}}_{V^T(n \times n)} \quad (2)$$

con  $u_1, \dots, u_m$  siendo las columnas ortonormales de  $U$ ,  $\sigma_1, \dots, \sigma_r$  siendo los valores singulares de  $A$  que satisfacen  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$ , y  $v_1, \dots, v_n$  siendo las columnas ortonormales de  $V^T$ . Los valores singulares se definen como las raíces cuadradas positivas de los autovalores de  $A^T A$ .

Nótese que como  $A^T A$  de tamaño  $n \times n$  es real y simétrica de rango  $r$ ,  $r$  de sus autovalores  $\sigma_i^2$ ,  $i = 1, \dots, r$ , son positivos, mientras que los  $n - r$  autovalores restantes son cero. En particular,

$$A^T A = V(\Sigma^T \Sigma)V^T, \quad A^T A v_i = \sigma_i^2 v_i, \quad i = 1, \dots, r, \quad A^T A v_i = 0, \quad i = r + 1, \dots, n. \quad (3)$$

Por lo tanto, los primeros  $r$  vectores  $v_i$  son los autovectores de  $A^T A$  con los autovalores  $\sigma_i^2$ . De manera similar, tenemos

$$AA^T = U(\Sigma\Sigma^T)U^T, \quad A^T A v_i = \sigma_i^2 v_i, \quad i = 1, \dots, r, \quad A^T A v_i = 0, \quad i = r + 1, \dots, m. \quad (4)$$

Por lo tanto, los primeros  $r$  vectores  $u_i$  son los autovectores de  $AA^T$  con los autovalores  $\sigma_i^2$ . Además,

$$Av_i = \sigma_i u_i, \quad i = 1, \dots, r, \quad Av_i = 0, \quad i = r + 1, \dots, n. \quad (5)$$

Si  $\text{rango}(A) = r < \min(m, n)$ , entonces hay  $n - r$  columnas y filas de ceros en  $\Sigma$ , haciendo innecesarias las columnas  $r + 1, \dots, m$  de  $U$  y las filas  $r + 1, \dots, n$  de  $V^T$  para hallar  $A$ . Por lo tanto,

$$\begin{aligned} A &= U\Sigma V^T = [u_1 | \dots | u_r | u_{r+1} | \dots | u_m] \begin{bmatrix} \sigma_1 & 0 & \dots & \dots & 0 & \dots & 0 \\ 0 & \ddots & & & \vdots & & \vdots \\ \vdots & & \sigma_r & & & \ddots & \\ 0 & & 0 & 0 & \vdots & & \vdots \\ 0 & 0 & \dots & \ddots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} v_1^T \\ \vdots \\ v_r^T \\ v_{r+1}^T \\ \vdots \\ v_n^T \end{bmatrix} \quad (6) \\ &= \underbrace{[u_1 | \dots | u_r]}_{m \times r} \underbrace{\begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & & \sigma_r \end{bmatrix}}_{r \times r} \underbrace{\begin{bmatrix} v_1^T \\ \vdots \\ v_r^T \end{bmatrix}}_{r \times n} \\ &= u_1\sigma_1v_1 + \dots + u_r\sigma_rv_r. \end{aligned}$$

## 1.2. Aplicacion

Al ser el SVD una técnica de reducción de dimensionalidad, su objetivo es, dado unos datos con cierta dimensión, proyectarlos a dimensiones menores sin perder mucha información, es decir, que la proyección que hagamos recolecte la mayor información relevante y significativa de los datos originales. Dada esta descomposición, podemos elegir cuantos valores de la matriz  $\Sigma$  conservar para que se tenga un buen representante de los datos pero en dimensiones menores, ya que estos valores estan organizados en orden descendente y de tal manera que su porcentaje de explicación de la varianza total de los datos también este en este orden.

Por esto, el SVD es ampliamente utilizada para diversos procedimientos tales como el cálculo del pseudoinverso, el ajuste de datos por mínimos cuadrados, el control multivariable, la aproximación de matrices y la determinación del rango, el rango y el espacio nulo de una matriz y/o la compresión de imágenes, entre otras. Profundizaremos en el uso del SVD para comprimir imágenes.

### 1.2.1. Compresión de imágenes

**Definición:** Comprimir una imagen es reducir los datos redundantes e irrelevantes de la imagen con la menor pérdida posible para permitir su almacenamiento o transmisión de forma eficiente.

Para entender como podemos aplicar el SVD a este proceso, es necesario comprender el funcionamiento del color en las imágenes digitales: Los colores (tales como el blanco, cafe o morado por ejemplo) que vemos en nuestras pantallas no son exactamente lo que parecen, de hecho, no hay pigmentos de esos colores en la pantalla. Lo que vemos realmente es una mezcla del Rojo, Verde y Azul esparridos por medio de pequeños pixeles en toda la pantalla, de allí viene la abreviación RGB (Red - Green - Blue). Estos pixeles rojos, verdes y azules tienen una saturación (intensidad) en una escala de 0 a 255, siendo 0 completamente apagado y 255 completamente encendido. A continuación una representación de ello:

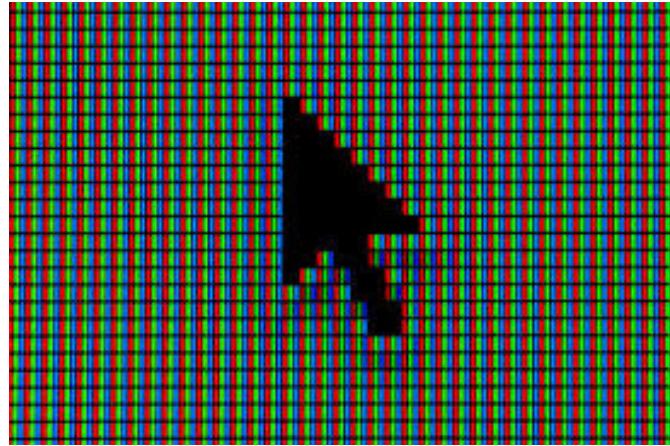


Figura 1: Pixeles dibujando una flecha - Tomada de <https://shorturl.at/jpAM5>

Teniendo en cuenta esto, una imagen es basicamente una fusión de tres matrices correspondientes a la capa o canal del color rojo, verde y azul, todas de dimensión  $m \times n$  (medido en pixeles). La entrada de cada matriz es un valor de 0 a 255 indicando la intensidad del color en esa posición. Dicho todo esto, el procedimiento que seguiremos para la descomposición de imagenes será este:

1. Seleccionar una imagen, no importa la dimensión o resolución.
2. Hallar sus matrices RGB y hacer un mapeo desde sus valores actuales al intervalo  $[0, 1]$  dividiendolos todos por 255. Hacemos esto debido a la facilidad de traducir los resultados de la descomposición resultante a matrices que describe la imagen.
3. Calculamos el SVD de las matrices correspondientes a los canales rojo, verde y azul.
4. Hacemos un análisis de la varianza explicada por cada valor singular y determinamos cuantos conservamos.
5. Dada las nuevas matrices RGB, las fusionamos y generamos la nueva imagen.

### 1.2.2. Desarrollo en R

A continuación se muestra el código en R que implementa el proceso de reducción en una imagen dado el procedimiento que se acaba de presentar:

```
# Instalamos la librería "magick" que es util para procesamiento de gráficos e
# imágenes en R.
install.packages("magick")

# Importamos la librería "magick".
library(magick)

# Importamos y leemos la imagen a comprimir.
imagen <- image_read("C:/Users/David Espitia/Desktop/canada.jpg")

# Obtenemos el conjunto de matrices RGB de nuestra imagen.
matrix_rgb <- image_data(imagen)

# Hallar la anchura medida en pixeles de la imagen (output = 1600px).
```

```

numero_columnas = dim(matrix_rgb)[2]
# Hallar la altura medida en pixeles de la imagen (output = 900px).
numero_filas = dim(matrix_rgb)[3]

# Tomar la matrix del canal "red" convertida a valores decimales y estandarizada
# (porque en un principio estan en base hexagesimal y queremos que esten en
# un rango de 0 a 1 para la fusión final de los canales).
red <- matrix(data = as.integer(matrix_rgb[1,,]), nrow = numero_filas,
               ncol = numero_columnas)/255
# Tomar la matrix del canal "green" convertida a valores decimales y estandarizada.
green <- matrix(data = as.integer(matrix_rgb[2,,]), nrow = numero_filas,
                  ncol = numero_columnas)/255
# Tomar la matrix del canal "blue" convertida a valores decimales y estandarizada.
blue <- matrix(data = as.integer(matrix_rgb[3,,]), nrow = numero_filas,
                 ncol = numero_columnas)/255

# Calcular SVD para "red".
svd_red <- svd(red)

# Calcular SVD para "green".
svd_green <- svd(green)

# Calcular SVD para "blue".
svd_blue <- svd(blue)

# Graficamos la varianza total explicada de los datos por cada valor singular para
# los canales RGB.

plot(svd_red$d^2/sum(svd_red$d^2),pch=19,xlab="Valor singular",
      ylab="Varianza explicada",
      main="Varianza explicada por cada valor singular - Red")
plot(svd_green$d^2/sum(svd_green$d^2),pch=19,xlab="Valor singular",
      ylab="Varianza explicada",
      main="Varianza explicada por cada valor singular - Green")
plot(svd_blue$d^2/sum(svd_blue$d^2),pch=19,xlab="Valor singular",
      ylab="Varianza explicada",
      main="Varianza explicada por cada valor singular - Blue")

# Ciclo para obsevar como luce la imagen con 2, 10, 20, 90 y 900 de sus valores
# singulares.
for (i in c(2,10,20,90,900)) {
  canal_red <- svd_red$u[,1:i] %*% diag(svd_red$d[1:i]) %*% t(svd_red$v[,1:i])
  canal_green <- svd_green$u[,1:i] %*% diag(svd_green$d[1:i]) %*% t(svd_green$v[,1:i])
  canal_blue <- svd_blue$u[,1:i] %*% diag(svd_blue$d[1:i]) %*% t(svd_blue$v[,1:i])

  imagen <- array(c(canal_red,canal_green,canal_blue),
                   dim = c(numero_filas,numero_columnas,3))

  saveRDS(imagen, paste("imagen",i,".rds",sep=""))
}

```

```

# Obtenemos las matrices RGB de las imágenes con números diferentes de valores
# singulares.
imagen2_matrix <- readRDS("imagen2.rds")
imagen10_matrix <- readRDS("imagen10.rds")
imagen20_matrix <- readRDS("imagen20.rds")
imagen90_matrix <- readRDS("imagen90.rds")
imagen900_matrix <- readRDS("imagen900.rds")

# Instalamos la librería "grid" que provee un sistema de graficos de bajo nivel
# para crear plots, tablas y otras visualizaciones.
install.packages("grid")
# Importamos la librería "grid".
library(grid)

# Generamos la imagen con 2 valores singulares, establecemos sus dimensiones e
# imprimos el plot de la imagen. Observece que en la función "rgb" utilizamos
# las funciones "pmax" y "pmin" y esto es debido a que estas matrices tienen
# valores negativos o mayores a 1, y la función rgb acepta entradas de matrices
# en el intervalo [0,1], en esta configuración por lo tanto ajustamos estos
# valores de tal forma que: Si es negativo se convierta en 0 y si es mayor que 1
# que se convierta en 1. Para efectos visuales esto no afecta en el resultado final.
imagen2 <- rgb(pmax(pmin(imagen2_matrix[,,1],1),0),
                pmax(pmin(imagen2_matrix[,,2],1),0),
                pmax(pmin(imagen2_matrix[,,3],1),0))
dim(imagen2) <- c(1600,900)
grid.raster(imagen2, interpolate=FALSE)

# Generamos la imagen con 10 valores singulares, establecemos sus dimensiones e
# imprimos el plot de la imagen.
imagen10 <- rgb(pmax(pmin(imagen10_matrix[,,1],1),0),
                  pmax(pmin(imagen10_matrix[,,2],1),0),
                  pmax(pmin(imagen10_matrix[,,3],1),0))
dim(imagen10) <- c(1600,900)
grid.raster(imagen10, interpolate=FALSE)

# Generamos la imagen con 20 valores singulares, establecemos sus dimensiones e
# imprimos el plot de la imagen.
imagen20 <- rgb(pmax(pmin(imagen20_matrix[,,1],1),0),
                  pmax(pmin(imagen20_matrix[,,2],1),0),
                  pmax(pmin(imagen20_matrix[,,3],1),0))
dim(imagen20) <- c(1600,900)
grid.raster(imagen20, interpolate=FALSE)

# Generamos la imagen con 90 valores singulares, establecemos sus dimensiones e
# imprimos el plot de la imagen.
imagen90 <- rgb(pmax(pmin(imagen90_matrix[,,1],1),0),
                  pmax(pmin(imagen90_matrix[,,2],1),0),
                  pmax(pmin(imagen90_matrix[,,3],1),0))
dim(imagen90) <- c(1600,900)

```

```

grid.raster(imagen90, interpolate=FALSE)

# Generamos la imagen con 900 valores singulares, establecemos sus dimensiones e
# imprimos el plot de la imagen.
imagen900 <- rgb(pmax(pmin(imagen900_matrix[, , 1], 1), 0),
                  pmax(pmin(imagen900_matrix[, , 2], 1), 0),
                  pmax(pmin(imagen900_matrix[, , 3], 1), 0))
dim(imagen900) <- c(1600, 900)
grid.raster(imagen900, interpolate=FALSE)

```

### 1.2.3. Resultados de la compresión

Como se mostró en el código, escogimos una imagen llamada 'canada.jpg' de altura 900px y anchura 1600px tomada de <https://rb.gy/8ak0dw>. Al realizar el calculo SVD se obtuvieron las siguientes gráficas que nos muestra la varianza explicada de cada valor singular para los diferentes canales:

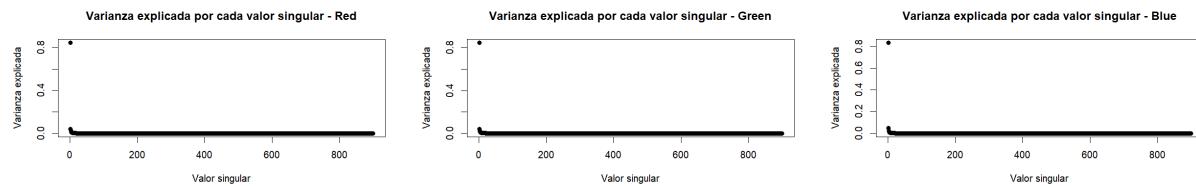


Figura 2: Varianza explicada de los valores singulares - RGB

Se observa que los primeros (los número 1) valores singulares de la descomposición SVD de cada canal explican mas del 80 % de la variación total de los datos. Tambien es relevante mencionar que los segundos mayores valores singulares de canal explica aproximadamente un 2 % de la variación total de los datos. De esto podemos decir con mucha certeza que si aproximamos las matrices RGB de la imagen a su descomposición SVD tomando solo los 2 primeros valores singulares, se entenderá el bosquejo de la imagen o por lo menos se puede intuir el proposito de esta. Es claro que si consideramos todos los valores singulares que en nuestro caso son 900, la aproximación será prácticamente igual a la imagen original, pero noteese también que a medida que consideramos mas valores singulares, la explicación de la varianza que estos proporcionan se va haciendo cada vez mas pequeña tendiendo a 0, por lo que si consideramos solo los componentes principales (es decir, los valores que mas explican la varianza de los datos), tendríamos una aproximación muy satisfactoria de la imagen original. A continuación, visualizaremos los resultados de las imágenes considerando 2, 10, 20, 90 y 900 valores singulares y tambien anexaremos la imagen original:

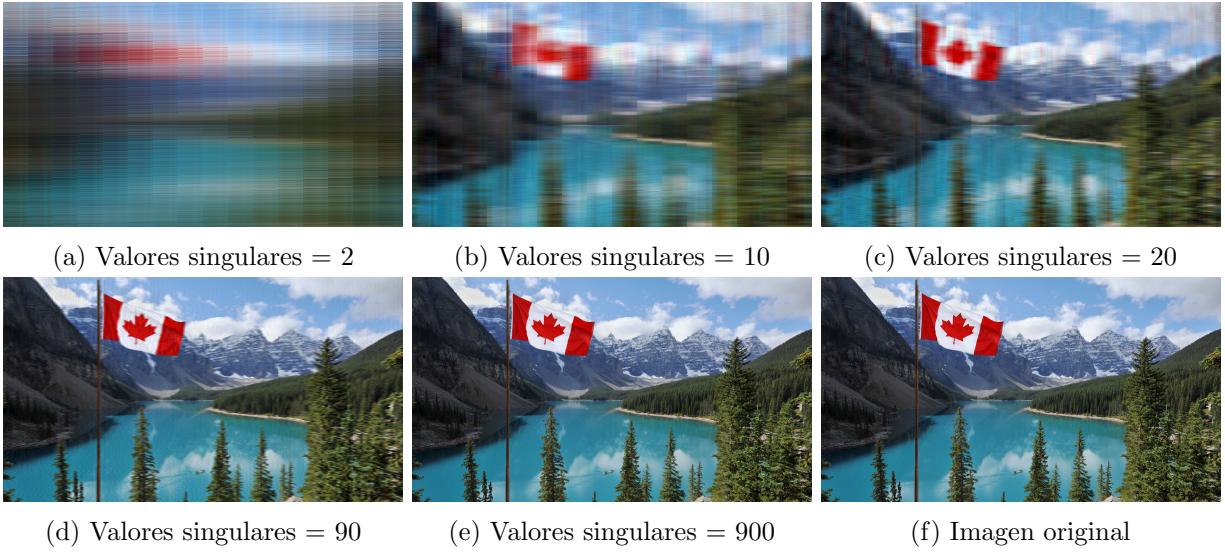


Figura 3: Comparación de la imagen con diferentes valores singulares

Notese que cuando tenemos 2 valores singulares se entiende la intención de la imagen tal como lo pronosticamos. Se observa tambien una mejora decreciente cuando vamos considerando mas valores singulares y terminamos resaltando que tomando 90 valores singulares, podemos practicamente reemplazar la imagen original. Lo util de todo esto es que 90 valores singulares significan que esta imagen de dimensiones  $900 \times 1600$  pasa a ser una aproximación muy fiel contando solo con 90 componentes principales siendo cada una una matriz de rango 1, esto implica que la dimensión se redujo considerablemente, pasando de 900 a 90 (10 %) y para efectos computacionales, la imagen se vuelve mas tratable.

## 2. Calculo de la descomposición en valores singulares

### 2.1. Algoritmo

A continuación, se presentan los pasos a seguir para calcular el SVD:

1. Calculamos  $A^T A$  de la matriz  $A$  de rango  $r$  de dimensión  $m \times n$ , siguiendo con la matriz considerada en 1.1.
2. Resolvemos la ecuación caracteristica  $\Delta_{A^T A}(\lambda) = \left| A^T A - \lambda I \right| = 0$  de  $A^T A$  para los valores propios  $\lambda_1, \dots, \lambda_r$  de  $A^T A$ . Estos valores propios serán positivos. Despues tomamos sus raices cuadradas para obtener  $\sigma_1, \dots, \sigma_r$  que son los valores singulares de  $A$ , esto es,  $\sigma_i = +\sqrt{\lambda_i}$  para todo  $i = 1, \dots, r$ .
3. Ordenamos los valores singulares, posiblemente renombrandolos, tal que  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$ .
4. Construimos la matriz  $\Sigma$  de tamaño  $m \times n$  tal que  $\Sigma_{ii} = \sigma_i$  para  $i = 1, \dots, r$ , y  $\Sigma_{ij} = 0$  cuando  $i \neq j$ .
5. Encontramos una base para  $\text{Null}(A^T A - \lambda_i I)$  (recordemos que el espacio nulo de una matriz  $M$  es el conjunto de todas las soluciones a la ecuación homegena  $M\mathbf{x} = \mathbf{0}$ , es decir,  $\text{Null}(M) = \{\mathbf{x} : \mathbf{x} \in \mathbf{R}^n, M\mathbf{x} = \mathbf{0}\}$ ). Es decir, resolvemos  $(A^T A - \lambda_i I)s_i = 0$  para  $s_i$ , un vector propio de  $A$  correspondiente a  $\lambda_i$ , para cada valor  $\lambda_i$ . Ya que  $A^T A$  es simetrica, sus vectores propios correspondientes a diferentes valores propios son ya ortogonales (pero probablemente no ortonormales).

6. Calculamos los vectores singulares derechos  $v_1, \dots, v_r$  normalizando cada vector propio  $s_i$  multiplicandolo por  $\frac{1}{\|s_i\|}$ . Es decir,  $v_i = \frac{1}{\|s_i\|} s_i$  para todo  $i = 1, \dots, r$ . Si  $n > r$ , los  $n - r$  vectores  $v_{r+1}, \dots, v_n$  necesitan ser escogidos como una base ortonormal en  $Null(A)$ . Notese que ya que  $Av_i = \sigma_i u_i$  para  $i = 1, \dots, r$ , los vectores  $v_1, \dots, v_r$  proveen una base ortonormal para  $Row(A)$  (el espacio generado por los vectores fila de  $A$ ) mientras que los vectores  $u_1, \dots, u_r$  proveen una base ortonormal para  $Col(A)$  (el espacio generado por los vectores columna de  $A$ ). En particular,  $R^n = Row(A) \perp Null(A) = span\{v_1, \dots, v_r\} \perp span\{v_{r+1}, \dots, v_n\}$  (recordando que  $span$  de un conjunto de vectores es el conjunto de todas las posibles combinaciones lineales de esos vectores y que  $\mathbf{E}_1 \perp \mathbf{E}_2$  significa que cada vector en  $\mathbf{E}_1$  es ortogonal a cada vector de  $\mathbf{E}_2$ ).
  7. Construimos la matriz ortogonal  $V = [v_1 | \dots | v_n]$ .
  8. Verificamos  $V^T V = I$ .
  9. Calculamos los vectores singulares izquierdos  $u_1, \dots, u_r$  como  $Av_i = \sigma_i u_i \Rightarrow u_i = \frac{Av_i}{\sigma_i}$ , para todo  $i = 1, \dots, r$ . En este metodo,  $u_1, \dots, u_r$  son ortogonales. Si  $m > r$ , los  $m - r$  vectores  $u_{r+1}, \dots, u_m$  necesitan ser escogidos como una base ortonormal en  $Null(A^T)$ . Notese que ya que  $Av_i = \sigma_i u_i$  para  $i = 1, \dots, r$ , los vectores  $u_1, \dots, u_r$  proveen una base ortonormal para  $Col(A)$  (el espacio generado por los vectores fila de  $A$ ) mientras que los vectores  $u_{r+1}, \dots, u_m$  proveen una base ortonormal para el espacio izquierdo  $Null(A^T)$ . En particular,  $R^m = Col(A) \perp Null(A^T) = span\{u_1, \dots, u_r\} \perp span\{u_{r+1}, \dots, u_m\}$ .
  10. Construimos  $U = [u_1 | \dots | u_m]$ .
  11. Verificar  $U^T U = I$ .
  12. Verificar  $A = U \Sigma V^T$ .
  13. Construimos la descomposición diádica de  $A$  como  $A = U \Sigma V^T = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \dots + \sigma_r u_r v_r^T$ . Recordemos que una diáda es un producto de un vector columna  $n \times 1$  con otro vector fila  $1 \times n$ , resultando en una matrix  $n \times n$  cuyo rango es 1.

## 2.2. Implementación en R

A continuación se presenta el código en R del algoritmo anteriormente presentado:

```

svd_algorithm <- function(A) {
  # Paso 1: Calcular  $A^T * A$ 
  ATA <- t(A) %*% A

  # Paso 2: Calcular los valores singulares de  $A^T * A$ 
  singular_values <- sqrt(eigen(ATA)$values)

  # Paso 3: Ordenar los valores singulares
  sorted_singular_values <- sort(singular_values, decreasing = TRUE)

  # Paso 4: Construir la matriz Sigma
  Sigma <- diag(sorted_singular_values)

  # Paso 5: Calcular los autovectores de  $A^T * A$ 
  eigenvectors <- eigen(ATA)$vectors

  # Paso 6: Calcular los vectores singulares derechos
}

```

```

U <- eigenvectors
V <- matrix(0, nrow = ncol(A), ncol = length(sorted_singular_values))

for (i in 1:length(sorted_singular_values)) {
  V[, i] <- eigenvectors[, i] / sqrt(sum(eigenvectors[, i]^2))
}

# Paso 7: Completar los vectores singulares derechos si es necesario
if (nrow(A) > length(sorted_singular_values)) {
  additional_V <- qr.Q(qr(t(A)))[, (length(sorted_singular_values) + 1):nrow(A)]
  V <- cbind(V, additional_V)
}

# Paso 8: Verificar que  $V^T * V = I$ 
verify_V <- all.equal(t(V) %*% V, diag(ncol(V)))

# Paso 9: Calcular los vectores singulares izquierdos
U <- matrix(0, nrow = nrow(A), ncol = length(sorted_singular_values))

for (i in 1:length(sorted_singular_values)) {
  U[, i] <- A %*% V[, i] / sorted_singular_values[i]
}

# Paso 10: Completar los vectores singulares izquierdos si es necesario
if (nrow(A) > length(sorted_singular_values)) {
  additional_U <- qr.Q(qr(A))[, (length(sorted_singular_values) + 1):nrow(A)]
  U <- cbind(U, additional_U)
}

# Paso 11: Verificar que  $U^T * U = I$ 
verify_U <- all.equal(t(U) %*% U, diag(nrow(U)))

# Paso 12: Verificar que  $A = U * \Sigma * V^T$ 
verify_A <- all.equal(A, U %*% Sigma %*% t(V))

# Paso 13: Construir la descomposición dyádica de A
dyadic_decomposition <- list()
for (i in 1:length(sorted_singular_values)) {
  dyadic_decomposition[[i]] <- sorted_singular_values[i] * outer(U[, i], V[, i])
}

# Resultados
result <- list(Sigma = Sigma,
                 U = U,
                 V = V,
                 verify_V = verify_V,
                 verify_U = verify_U,
                 verify_A = verify_A,
                 dyadic_decomposition = dyadic_decomposition)

```

```

    return(result)
}

# Ejemplo de uso:
# Crear una matriz de ejemplo
A <- matrix(c(1,0,1,2,2,1,1,1,4,2,2,2), nrow = 3, ncol = 4)

# Calcular la descomposición en valores singulares
result <- svd_algorithm(A)

# Verificar los resultados
print("Sigma:")
print(result$Sigma)
print("U:")
print(result$U)
print("V:")
print(result$V)
print("Verificación de V^T * V = I:")
print(result$verify_V)
print("Verificación de U^T * U = I:")
print(result$verify_U)
print("Verificación de A = U * Sigma * V^T:")
print(result$verify_A)
print("Descomposición Dyádica:")
print(result$dyadic_decomposition)

```

Los resultados son los siguientes:

```

> print("Sigma:")
[1] "Sigma:"
> print(result$Sigma)
     [,1]      [,2]      [,3]      [,4]
[1,] 5.934925  0.000000  0.0000000 0.00000e+00
[2,] 0.000000  2.302153  0.0000000 0.00000e+00
[3,] 0.000000  0.000000  0.6904711 0.00000e+00
[4,] 0.000000  0.000000  0.0000000 3.46472e-08
> print("U:")
[1] "U:"
> print(result$U)
     [,1]      [,2]      [,3]      [,4]
[1,] -0.4871646  0.5145251  0.70564479 -1.281746e-08
[2,] -0.4521216  0.5427055 -0.70785366  2.563492e-08
[3,] -0.7471658 -0.6638785 -0.03175896  0.000000e+00
> print("V:")
[1] "V:"
> print(result$V)
     [,1]      [,2]      [,3]      [,4]
[1,] -0.2079774 -0.06487552  0.97597980 -1.421308e-15
[2,] -0.4424214  0.63009820 -0.05239423 -6.359987e-01
[3,] -0.6618363 -0.69425587 -0.18718340 -2.119996e-01
[4,] -0.5683145  0.34172535 -0.09839031  7.419985e-01

```

```

> print("Verificación de V\AT * V = I:")
[1] "Verificación de V\AT * V = I:"
> print(result$verify_V)
[1] TRUE
> print("Verificación de U\AT * U = I:")
[1] "Verificación de U\AT * U = I:"
> print(result$verify_U)
[1] "Attributes: < Component "dim": Mean relative difference: 0.25 >"
[2] "Numeric: lengths (16, 9) differ"
> print("Verificación de A = U * Sigma * V\AT:")
[1] "Verificación de A = U * Sigma * V\AT:"
> print(result$verify_A)
[1] TRUE

```

```

> print("Descomposición Dyadica:")
[1] "Descomposición Dyadica:"
> print(result$dyadic_decomposition)
[[1]]
      [,1]     [,2]     [,3]     [,4]
[1,] 0.6013221 1.279167 1.913558 1.643159
[2,] 0.5580675 1.187153 1.775911 1.524963
[3,] 0.9222494 1.961862 2.934829 2.520118

[[2]]
      [,1]     [,2]     [,3]     [,4]
[1,] -0.07684608 0.7463612 -0.8223570 0.4047790
[2,] -0.08105491 0.7872392 -0.8673972 0.4269486
[3,] 0.09915252 -0.9630107 1.0610660 -0.5222760

[[3]]
      [,1]     [,2]     [,3]     [,4]
[1,] 0.47552402 -0.025527900 -0.091200865 -0.047938447
[2,] -0.47701255 0.025607810 0.091486350 0.048088508
[3,] -0.02140192 0.001148935 0.004104678 0.002157566

[[4]]
      [,1]     [,2]     [,3]     [,4]
[1,] 6.311874e-31 2.824402e-16 9.414672e-17 -3.295135e-16
[2,] -1.262375e-30 -5.648803e-16 -1.882934e-16 6.590271e-16
[3,] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00

```

Cabe mencionar que R tiene una función integrada muy eficiente llamada `svd` que calcula de manera muy rápida la descomposición SVD de cualquier matriz que le pasemos retornando efectivamente el vector  $d$ , la matriz  $u$  y la matriz  $v$ . Pero para efectos educativos se decidió usar una implementación desde cero.

### 3. El gráfico Biplot

En esta sección correspondiente al biplot generado a partir de la descomposición del valor singular, se abordará la técnica de visualización que permite representar tanto las observaciones como las variables en un mismo espacio bidimensional o tridimensional. Se explicará el proceso para obtener un biplot a partir de la descomposición del valor singular, destacando como calcular la matriz de covarianza o correlación, así como la descomposición en sí misma. Además, se describirá cómo se proyectan las observaciones y las variables en el espacio de componentes principales y cómo se interpretan las coordenadas resultantes en el gráfico final. Este enfoque visual facilita la identificación de patrones, relaciones y agrupamientos entre las observaciones y las variables, ofreciendo una comprensión intuitiva de la estructura de los datos.

#### 3.1. Biplot a partir de la descomposición del valor singular

La obtención del biplot a partir de la descomposición del valor singular es una técnica comúnmente utilizada en análisis multivariado y análisis de componentes principales (PCA). Un biplot es una representación gráfica que muestra tanto las observaciones (individuos o casos) como las variables en el mismo espacio. Este tipo de representación es útil para visualizar la estructura de los datos y las relaciones entre las variables y las observaciones.

La base de la representación en el biplot es la descomposición de valores singulares. Si tenemos una matriz  $Y$  de tamaño  $n \times p$  con rango  $r$ , donde  $r \leq p \leq n$ , entonces la matriz puede descomponerse como:

$$Y = U\Lambda V^T$$

donde  $U$  ( $n \times p$ ) y  $V$  ( $p \times p$ ) son matrices de vectores singulares y  $\Lambda$  ( $p \times p$ ) es una matriz diagonal de valores singulares.  $U$  es la matriz cuyas columnas corresponden a los  $p$  autovectores ortogonales de  $YY'$  y  $V$  es la matriz ortogonal que corresponde a los autovectores de  $Y'Y$ .  $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_p)$ , donde  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_r > \lambda_{r+1} = \lambda_{r+2} = \dots = \lambda_m = 0$ . Los valores singulares son las raíces cuadradas positivas de los autovalores de  $Y'Y$ .

Normalmente, la matriz se aproxima utilizando los primeros valores y vectores singulares dominantes. La representación del biplot consiste en el gráfico de los marcadores de fila  $G$  (puntos que representan las observaciones o datos individuales) y los marcadores de columna  $H$  (flechas que representan las variables originales), donde:

$$\begin{aligned} G &= U_k \Lambda_k^\alpha \\ H &= V_k \Lambda_k^{1-\alpha} \end{aligned}$$

El valor de  $k$  determina la dimensión de la aproximación (generalmente  $k = 2$ ), es decir, el término se refiere a una matriz que contiene los primeros  $k$  autovectores de una matriz de datos, y el parámetro  $\alpha$  especificado por el usuario ( $0 \leq \alpha \leq 1$ ) determina si se enfatizan las filas o las columnas de  $Y$ . Luego, la matriz  $Y$  se aproxima por  $Y_k = GH' = U_k \Lambda_k^\alpha \Lambda_k^{1-\alpha} V_k'$ . Los gráficos de las coordenadas asociadas con  $G$  superpuestas sobre las coordenadas asociadas con  $H$  forman la representación visual del **Biplot**.

Aunque muchos valores de  $\alpha$  son posibles, tres son comúnmente utilizados: 1,  $\frac{1}{2}$  y 0. Cuando se selecciona el valor 1, el resultado se llama biplot JK o RMP (preservación de la métrica de fila). En esta representación, las distancias entre pares de filas se conservan (después de realizar cualquier centrado y escalamiento) y la visualización es útil para estudiar objetos (por ejemplo, interpretar matrices de distancia). Cuando se selecciona el valor 0, el resultado es un biplot GH o CMP (preservación de la

métrica de columna). Esta representación conserva las distancias entre las columnas y es útil para interpretar la varianza y las relaciones entre variables (por ejemplo, interpretar una matriz de covarianza o correlación). El otro valor de  $\alpha$ ,  $\frac{1}{2}$ , otorga escalamiento o peso igual a las filas y columnas. Es útil para interpretar la interacción en experimentos de dos factores.

Para construir el biplot, se proyectan las observaciones y las variables en el espacio de las primeras dos o tres componentes principales (las que explican la mayor cantidad de varianza). Las coordenadas resultantes se utilizan para ubicar tanto las observaciones como las variables en el mismo gráfico. Las observaciones se representan como puntos y las variables como flechas, donde la dirección y la longitud de las flechas indican la relación y la importancia de cada variable en las componentes principales.

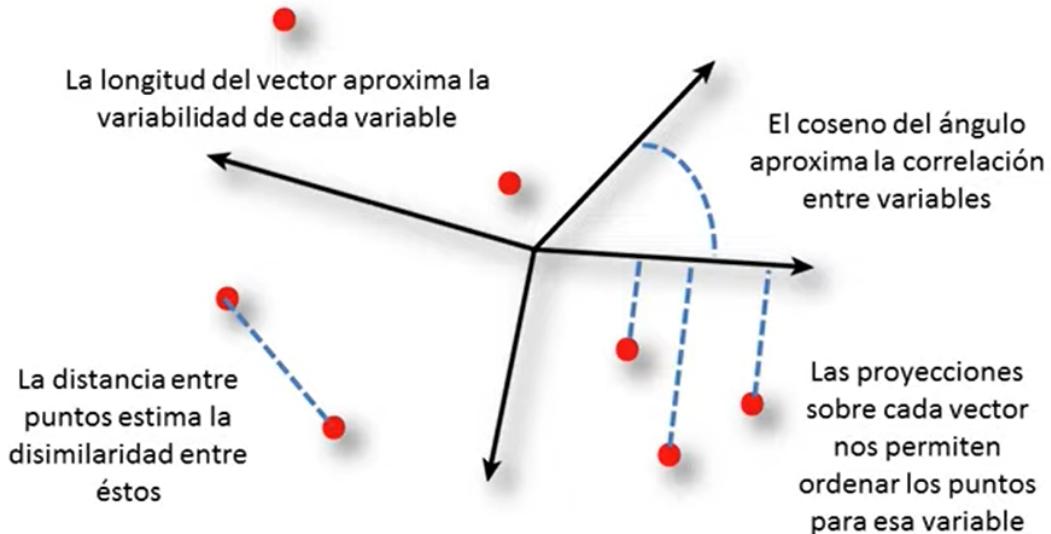


Figura 4: Interpretación de un Biplot

La interpretación de un Biplot sigue una regla sencilla, según el ángulo entre 2 vectores:

1. Si es mayor que  $90^\circ$ , las variables que representa están positivamente relacionadas.
2. Si es de aproximadamente  $90^\circ$ , las variables que representa no están relacionadas.
3. Si es menor que  $90^\circ$ , las variables que representa están negativamente relacionadas.

### 3.2. Aplicación

En esta sección de aplicación, se realizó un análisis de componentes principales (PCA) sobre el conjunto de datos Iris utilizando el paquete ‘vegan’ y ‘ggbioplot’ en R. El objetivo principal fue explorar la estructura de los datos y visualizar las relaciones entre las variables y las observaciones.

Primero, se realizó un análisis de componentes principales utilizando la función ‘rda()’ del paquete ‘vegan’. Se generó un gráfico de sedimentación (screeplot) para evaluar la importancia relativa de cada componente principal en el modelo.

```
> summary(df.pca)

Call:
rda(X = df, scale = TRUE)

Partitioning of correlations:
              Inertia Proportion
Total             4           1
Unconstrained     4           1

Eigenvalues, and their contribution to the correlations

Importance of components:

PC1    PC2    PC3    PC4
Eigenvalue 2.9185 0.9140 0.14676 0.020715
Proportion Explained 0.7296 0.2285 0.03669 0.005179
Cumulative Proportion 0.7296 0.9581 0.99482 1.000000
```

Figura 5: Análisis de componentes principales

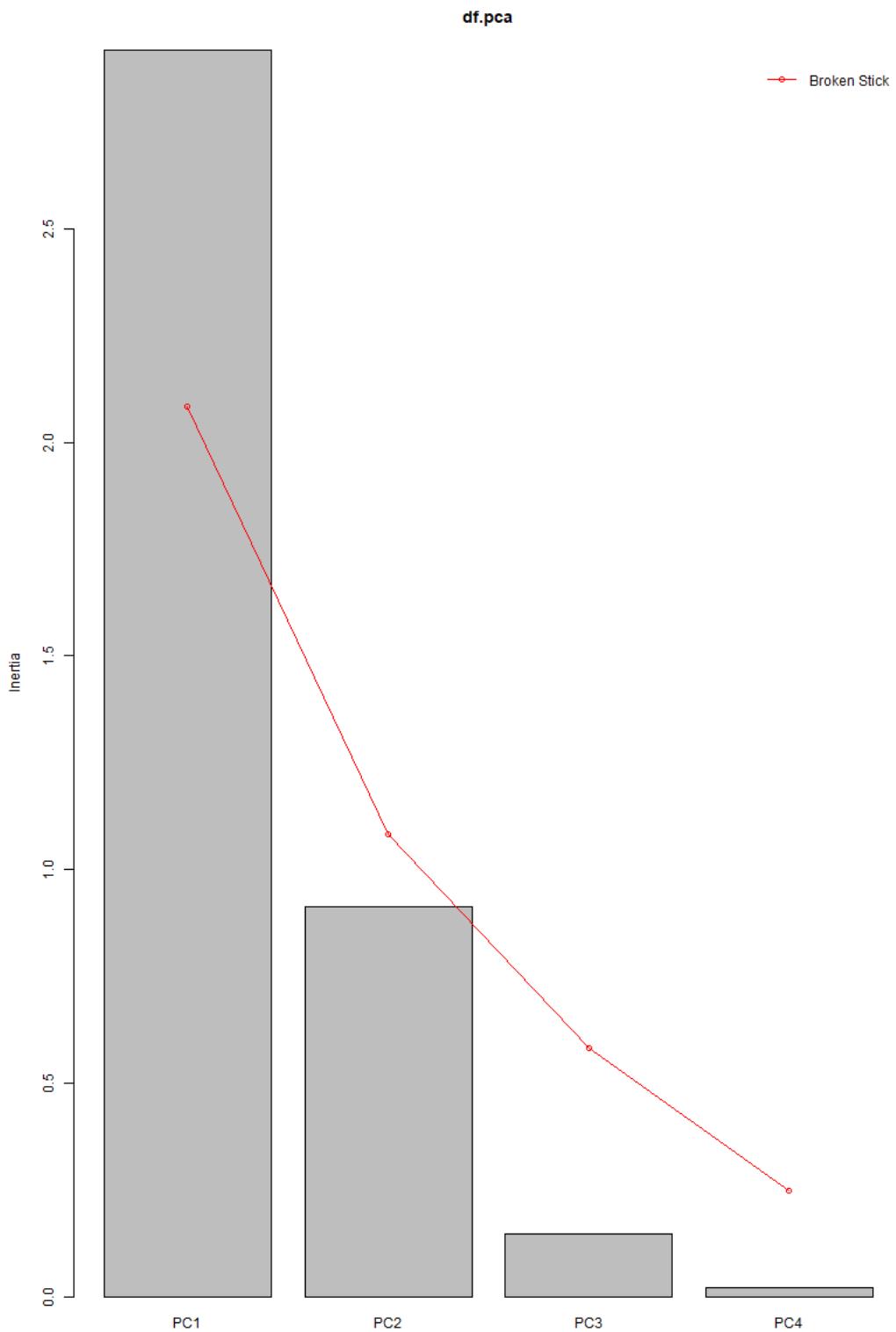


Figura 6: Gráfico complementario para el análisis de componentes principales

Este resumen del análisis de componentes principales (PCA) proporciona información sobre la cantidad de variabilidad en los datos explicada por cada componente principal, así como la contribución relativa de cada componente a la variabilidad total.

Los eigenvalores representan la cantidad de varianza explicada por cada componente principal. En este caso, el primer componente principal (PC1) tiene un eigenvalor de 2.9185, y explica el 72.96 % de la varianza total en los datos. El segundo componente principal (PC2) tiene un eigenvalor de 0.9140, lo que explica el 22.85 % de la varianza total. Los eigenvalores restantes para PC3 y PC4 son 0.14676 y 0.020715, respectivamente. La columna de "Proporción acumulada" indica la acumulación de varianza explicada a medida que se agregan más componentes principales. Por ejemplo, los dos primeros componentes principales (PC1 y PC2) explican conjuntamente el 95.81 % de la varianza total en los datos, lo cual es casi completamente los datos.

Posteriormente, se ejecutó un PCA adicional utilizando la función 'prcomp()' para obtener los componentes principales con el paquete 'stats'.

```
> pc
Standard deviations (1, ..., p=4):
[1] 1.7083611 0.9560494 0.3830886 0.1439265

Rotation (n x k) = (4 x 4):
          PC1        PC2        PC3        PC4
Sepal.Length 0.5210659 -0.37741762 0.7195664 0.2612863
Sepal.Width -0.2693474 -0.92329566 -0.2443818 -0.1235096
Petal.Length 0.5804131 -0.02449161 -0.1421264 -0.8014492
Petal.Width  0.5648565 -0.06694199 -0.6342727 0.5235971
```

Figura 7: Análisis de componentes principales

Esto muestra las desviaciones estándar de los primeros cuatro componentes principales, indicando que el primer componente (PC1) captura la mayor cantidad de variabilidad en los datos. La matriz de rotación revela cómo cada variable original se relaciona con cada componente principal.

Luego, se generó un biplot utilizando 'ggbioplot()' para visualizar las relaciones entre las observaciones (flores) y las variables (longitud y ancho de los sépalos y pétalos) en un espacio bidimensional.

El biplot resultante incluyó elipses alrededor de los puntos de los datos para representar la dispersión de las observaciones dentro de cada grupo de especies de flores. Además, se agregaron círculos para representar la correlación entre las variables originales en el espacio de los componentes principales. Este círculo es una representación visual de la correlación entre las variables originales en el espacio de los componentes principales. Cuando las variables están altamente correlacionadas en el espacio de los componentes principales, el círculo tendrá un radio grande. En cambio, si las variables están poco correlacionadas, el círculo será más pequeño. Esto proporciona una referencia visual adicional para interpretar las relaciones entre las variables en el biplot.

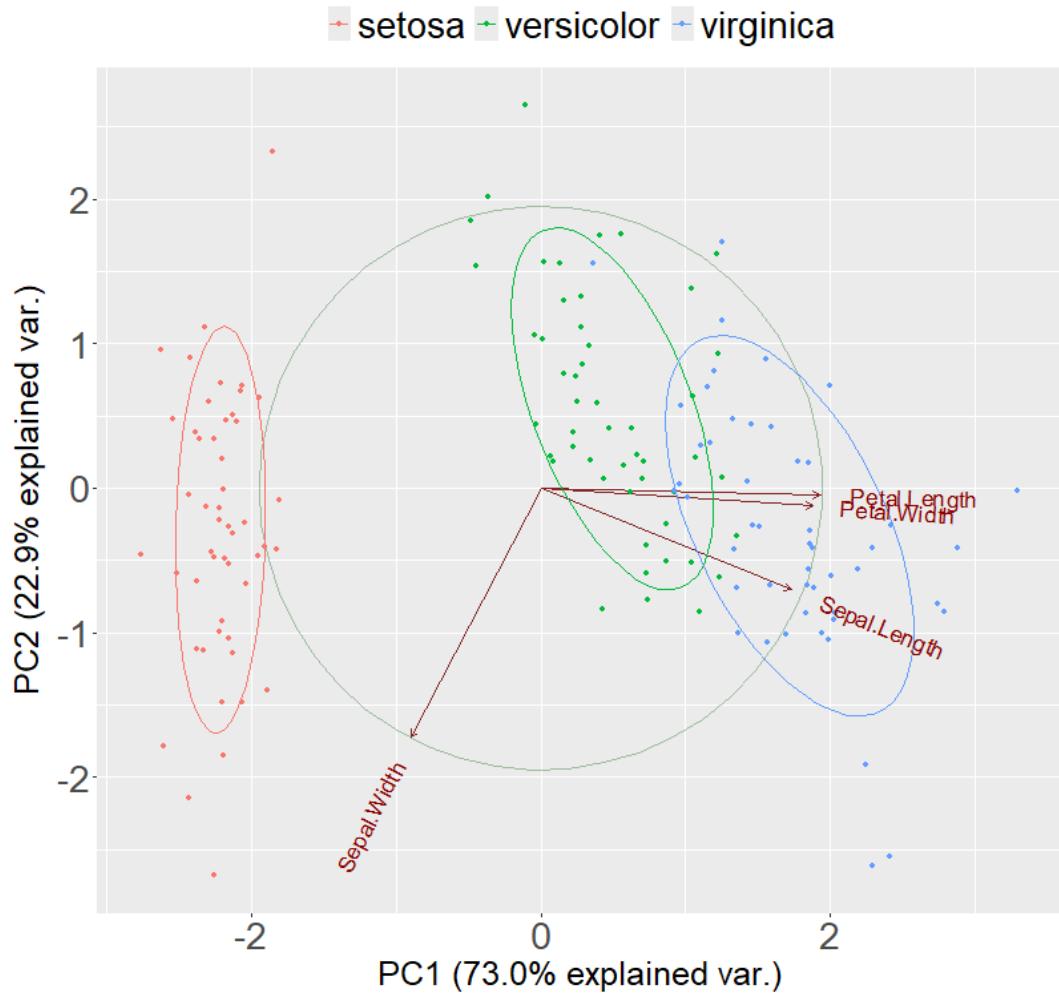


Figura 8: Biplot del dataset “iris”

### 3.2.1. Código

```
# Cargar las librerías necesarias
library(vegan)
library(ggbiplot)

# Leer el dataframe
df <- iris[, 1:4] # Seleccionar las columnas del dataframe

# Análisis de componentes principales
df.pca <- rda(df, scale = TRUE)
summary(df.pca)

# Modelo de "Broken stick" para los valores propios
(inventory <- df.pca$CA$eig)
screeplot(df.pca, bstick = TRUE, npcs = length(df.pca$CA$eig))

par(mfrow = c(1,1))
# Análisis de componentes principales
pc <- prcomp(df,
               center = TRUE, # Queremos usar el centro como promedio para las variables
               scale. = TRUE) # Normalización

# Gráfico biplot
g <- ggbiplot(pc,
               obs.scale = 1,
               var.scale = 1,
               groups = iris$Species,
               ellipse = TRUE, # Elipse alrededor del punto de los datos
               circle = TRUE, # Círculo en el gráfico
               varname.size = 6) # Tamaño de los nombres de las variables

# Ajustar el tamaño del título y los ejes
g <- g + theme(axis.title = element_text(size = 25), # Tamaño de los títulos de los ejes
                text = element_text(size = 35),
                legend.direction = "horizontal",
                legend.position = "top") # Tamaño del texto en general

g <- g + scale_color_discrete(name = "")

print(g)
```

### **3.3. Conclusiones**

La obtención del biplot a partir de la descomposición del valor singular en el análisis de componentes principales (PCA) es una herramienta valiosa para visualizar la relación entre observaciones y variables en un espacio. Este enfoque proporciona una representación intuitiva de la estructura de los datos, permitiendo identificar patrones, relaciones y agrupamientos de manera eficaz. Además, al asignar colores a las variables, se facilita la interpretación de la contribución de cada una al conjunto de datos. El biplot resultante no solo simplifica la comprensión de la distribución de los datos, sino que también ofrece insights importantes para análisis posteriores.

## Referencias

- [1] *Biplot and Singular Value Decomposition Macros for Excel*. URL: [https://www.researchgate.net/publication/5142803\\_Biplot\\_and\\_Singular\\_Value\\_Decomposition\\_Macros\\_for\\_ExcelC](https://www.researchgate.net/publication/5142803_Biplot_and_Singular_Value_Decomposition_Macros_for_ExcelC).
- [2] *Biplot in practice*. URL: [https://www.fbbva.es/wp-content/uploads/2017/05/dat/DE\\_2010\\_biplots\\_in\\_practice.pdf](https://www.fbbva.es/wp-content/uploads/2017/05/dat/DE_2010_biplots_in_practice.pdf).
- [3] *Descomposición en Valores singulares(SVD)*. URL: [https://www.mate.unlp.edu.ar/practicas/70\\_18\\_0911201012951.pdf](https://www.mate.unlp.edu.ar/practicas/70_18_0911201012951.pdf).
- [4] *Explicación detallada de la DVS en Wikipedia*. URL: [https://es.wikipedia.org/wiki/Descomposici%C3%B3n\\_en\\_valores\\_singulares](https://es.wikipedia.org/wiki/Descomposici%C3%B3n_en_valores_singulares).
- [5] Andrew Lounsbury. «Singular Value Decomposition». En: *Department of Mathematics, Tennessee Technological University* (sep. de 2018). URL: <https://www.tntech.edu/cas/pdf/math/techreports/TR-2018-2.pdf>.
- [6] *PCA VS RDA*. URL: [https://pmassicotte.github.io/stats-denmark-2019/07\\_rda.html#/pca-vs-rda](https://pmassicotte.github.io/stats-denmark-2019/07_rda.html#/pca-vs-rda).
- [7] Sergio A. Pernice. «Descomposición en valores singulares y análisis de factores en ciencias humanas y sociales». En: *Revista de Métodos Cuantitativos para la Economía y la Empresa* 37 (2024), págs. 1-29. DOI: 10.46661/revmetodoscuanteconempresa.8004.
- [8] *Principal Component Analysis (PCA) - Iris Dataset*. URL: <https://www.kaggle.com/code/kemalgunay/principal-component-analysis-pca-iris-dataset>.
- [9] ALVIN C. RENCHER. *Methods of Multivariate Analysis*. Second edition. ISBN: 0-471-41889-7. 2002. URL: <https://onlinelibrary.wiley.com/doi/book/10.1002/9781118391686>.
- [10] *Serrano.Academy en Español. Descomposición en valores singulares [Vídeo]*. YouTube. URL: <https://www.youtube.com/watch?v=wp3AJ0ZJCjw>.
- [11] Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. URL: <https://egrcc.github.io/docs/math/all-of-statistics.pdf>.