Práctica 2: Algunas aplicaciones con matrices

1. Operaciones con matrices y vectores

En el Tema 2 (Sección 2.5.3) hemos visto que una matriz es un vector al que se le añade el atributo `dim`. En este sentido vectores y matrices son objetos de distinta clase en R y por ejemplo un vector no es igual a una matriz con una de sus dimensiones igual a 1 (vector fila o columna) ya que al vector le falta el atributo `dim`.
Cuando realizamos operaciones que involucran vectores y matrices en R a veces este realiza conversiones de los vector que permitan realizar el cálculo, y en algunos casos las operaciones no se corresponde con lo que el usuario podría esperar.
Por ejemplo, si A es una matriz $n \times m$ y x es un vector de dimensión m, el producto (matricial) Ax sería un vectores de dimensión n. Sin embargo en R, el producto `A%*%x` es un matriz (en general un array).

Ejecuta las siguientes sentencias y extrae conclusiones sobre el tipo de objeto que devuelven:

```
A<-matrix(1:9,3,3)
x<-1:3
A%*%x
A%*%t(x)
x%*%A
t(x)%*%A
t(x)%*%x
```

```
> A<-matrix(1:9,3,3)
> x<-1:3
> A%*%x # aqui x es tratado como una matriz 3x1

     [,1]
[1,]   30
[2,]   36
[3,]   42

> A%*%t(x) # t(x) sería una matriz 1x3 !

Error in A%*% t(x): argumentos no compatibles

> x%*%A # esto no debería funcionar
```

```
     [,1] [,2] [,3]
[1,]   14   32   50

>       # y sin embargo funciona (x es tratado como una matriz 1x3)
> t(x)%*%A

     [,1] [,2] [,3]
[1,]   14   32   50

> t(x)%*%x # el resultado es una matriz 1x1 en lugar de un vector

     [,1]
[1,]   14
```

La conversión de un objeto a vector o matriz se puede hacer usando las funciones `as.vector` y `as.matrix`. Estudiaremos estas y otras funciones de conversión en la Sección 2.6 del Tema 2.

2. Sistemas de ecuaciones lineales

Muchos cálculos en álgebra lineal y algunas aplicaciones en Estadística están relacionados con el problema básico de resolver un sistema de ecuaciones lineales del tipo:

$$Ax = b$$

En R hemos visto que la función estándar en el paquete *base* para resolver este tipo de sistemas es `solve`.

Ejecuta las siguientes sentencias en R y formula los sistemas se resuelven en cada caso:

```
# Sistema 1
solve(2,2)
# Sistema 2
A<-matrix(c(3,1,4,2),2,2)
b<-c(12,8)
solve(A,b)
# Sistema 3
solve(A,diag(2))
```

```
> # Sistema 1
> solve(2,2)

[1] 1
```

```
> # Sistema 2
> A<-matrix(c(3,1,4,2),2,2)
> b<-c(12,8)
> solve(A,b)

[1] -4  6

> # Sistema 3
> solve(A,diag(2))

     [,1] [,2]
[1,]  1.0 -2.0
[2,] -0.5  1.5

> # compara con solve(A):
> solve(A)

     [,1] [,2]
[1,]  1.0 -2.0
[2,] -0.5  1.5
```

Si consultamos la ayuda de la función `solve` podemos ver que el argumento `b` puede ser un vector (como en los sistemas 1 y 2 anteriores) o una matriz (como en el sistema 3). Sin embargo la función no proporciona mucha información directa sobre el algoritmo que implementa sino que nos redirige a rutinas de *LAPACK* que pueden consultarse en `https://www.netlib.org/lapack/lug/node38.html`. Si lo hacemos podemos ver que la forma en que la función resuelve el problema depende de las propiedades de la matriz A, usando factorizaciones adecuadas de la misma. Así para matrices generales usa una descomposición LU con pivoteo parcial, y para matrices simétricas y definidas positivas la factorización de Cholesky.

2.1. Sistemas mal condicionados y número de condición

El buen funcionamiento de un algoritmo puede en muchos casos depender de los datos. A los problemas donde los datos particulares pueden causar problemas computacionales se suele denominar mal condicionados (*ill-conditioned*), y en general hablar de la "condición de los datos".
El concepto de "condición" se entiende por tanto en el contexto de un conjunto particular de operaciones. Y de forma resumida datos mal condicionados son aquellos en los que un pequeño cambio en los mismos puede producir enormes cambios en la solución. En relación a un sistema de ecuaciones lineales como los que ilustramos antes, este concepto se referiría a los coeficientes en las ecuaciones, esto es las propiedades de la matriz A, definiendo lo que se llaman "sistemas bien condicionados" o "mal condicionados".

Veamos un ejemplo de un sistema que está mal condicionado y comprobemos el problema que supondrían pequeños cambios en los coeficientes. Se trata del sistema:

$$\begin{aligned} 10x + 7y + 8z + 7w &= 32 \\ 7x + 5y + 6z + 5w &= 23 \\ 8x + 6y + 10z + 9w &= 33 \\ 7x + 5y + 9z + 10w &= 31 \end{aligned}$$

Crea objetos `A` (matriz de coeficientes) y `b` (vector de términos independientes) para resolver en R el sistema anterior. Resuelve el sistema con `solve`. Después perturba el vector `b` sumándole 0.05 a cada uno de sus elementos y busca la nueva solución, ¿se parece a la anterior? Repite la operación con un incremento de 0.1. Comenta los resultados.

```
> A<-matrix(c(10,7,8,7,7,5,6,5,8,6,10,9,7,5,9,10),4,4)
> b<-c(32,23,33,31)
> solve(A,b)

[1] 1 1 1 1

> # cambio pequeño en los coeficientes
> b<-c(32,23,33,31)+0.05
> solve(A,b)

[1] 0.40 2.00 0.75 1.15

> # cambio pequeño en los coeficientes
> b<-c(32,23,33,31)+0.1
> solve(A,b)

[1] -0.2  3.0  0.5  1.3
```

La condición de los datos se cuantifica mediante lo que se llama el número de condición (*condition number*). Se trata de un valor positivo de modo que valores muy grandes indican que los datos están mal condicionados. Para el caso de un sistema lineal con matriz de coeficientes A no singular, el número de condición se define en relación a la inversa[1] de A como:

$$\kappa_p(A) = \|A\|_p \|A^{-1}\|_p,$$

[1]Para más detalles sobre este tema puedes estudiar el libro Gentle, J.E. (2009).*Computational Statistics*. Statistics & Computing. Springer.

donde su definición depende de la norma ($\|\cdot\|_p$) considerada. Para la norma L_2 ($p = 2$) este valor coincide con el cociente entre el máximo y el mínimo autovalor[2] (en valor absoluto) de la matriz.
En R el número de condición se calcula con la función `kappa` (con norma L_2 por defecto). Un valor relacionado y fácil de interpretar es el número de condición recíproco (*reciprocal condition number*) que se define como la inversa del número de condición con valores entre 0 y 1. En R se calcula con la función `rcond` y se interpreta como una medida de lo cerca que está una matriz A de "ser singular". Cuanto más cerca esté de 0 más cerca estará de ser singular y el mal condicionamiento será más severo.

Calcula el número de condición de la matriz `A` del sistema anterior así como su recíproco. Realiza primero el cálculo con las funciones `kappa` y `rcond` y después comprueba que coinciden con su definición (`kappa(A)` como cociente entre máximo y mínimo autovalor en valor absoluto y `rcond(A)` como su inversa). Comenta el resultado.

```
> kappa(A)

[1] 3341.215

> rcond(A)  ## la matriz está muy cerca de ser singular,

[1] 0.0002228164

>           #  el sistema está mal condicionado
>
> # comprobación de la función kappa:
> A.autov<-eigen(A)$values
> abs(max(A.autov)/min(A.autov))  # ¿=? kappa

[1] 2984.093

> kappa(A, exact=TRUE)  ## este sí coincide

[1] 2984.093

> # comprobación de la función rcond:
> 1/kappa(A) # ¿=? rcond

[1] 0.0002992923

> 1/kappa(A, exact=TRUE)

[1] 0.0003351102
```

[2]En R los autovalores de un objeto de tipo matriz `A` los puedes obtener con la función `eigen`. Puedes calcularlos y acceder a ellos desde el vector de devuelve la sentencia `eigen(A)$values`.

3. Regresión lineal y mínimos cuadrados

3.1. Contexto

Los modelos de regresión lineal se utilizan para describir la relación entre una variable (observable) de respuesta (Y) y un grupo de variables (observables) predictoras ($X_1, \ldots, X_k$). Dado un conjunto de n observaciones de dichas variables la ecuación del modelo para la observación i-ésima se expresa como

$$y_i = \beta_0 + \sum_{j=1}^{k} \beta_j x_{ij} + \epsilon_i \quad (i = 1, \ldots, n), \tag{1}$$

donde x_{ij} denota la observación i-ésima de la variable X_j, $\beta_0, \ldots, \beta_k$ son los coeficientes de regresión (parámetros desconocidos), y ϵ_i son errores aleatorios (no observables) con media 0, incorrelados y varianza constante[3].
Usando notación matricial el modelo anterior para las n observaciones se escribe como:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

donde se definen:

- El vector de respuesta $\mathbf{y} = (y_1, \ldots, y_n)'$.
- La matriz de regresión: $\mathbf{X} = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,k} \\ 1 & x_{2,1} & \cdots & x_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,k} \end{pmatrix}$
- El vector de coeficientes: $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_k)'$.
- El vector de errores: $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)'$,

El problema de la regresión consiste en estimar $\boldsymbol{\beta}$ a partir de los datos, buscando un estimador $\widehat{\boldsymbol{\beta}}$ tal que $X\widehat{\boldsymbol{\beta}}$ esté "lo más próximo posible" a $\mathbf{y}$.
Observa que este problema está relacionado con la resolución del sistema lineal

$$X\boldsymbol{\beta} = \mathbf{y},$$

y que dado que en la práctica el problema se resuelve con un número de observaciones n bastante superior al número de variables k (o coeficientes β_j), se trata de un sistema con más ecuaciones que incógnitas que por lo general no tendrá solución "exacta"[4]. Por tanto el objetivo es resolver este otro sistema:

$$X\boldsymbol{\beta} \approx \mathbf{y},$$

[3]Estas tres condiciones se denominan condiciones de Gauss-Markov y el modelo con ellas se denomina modelo de regresión lineal simple de Gauss-Markov

[4]Observa que si tuviera solución exacta estaríamos diciendo que las observaciones de la variable de respuesta Y son una combinación lineal de las observaciones de las variables predictoras. Esto no se corresponderá con la situación real en la práctica.

buscando la "mejor" aproximación.
El método óptimo de estimación en este contexto nos lo da el criterio de los mínimos cuadrados, el cual busca la solución al sistema anterior resolviendo el siguiente problema de minimización:

$$\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2 = \min_{\boldsymbol{\beta}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}). \tag{2}$$

La solución a dicho problema se puede obtener fácilmente de manera analítica y está dada por la expresión

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}, \tag{3}$$

asumiendo que dicha inversa exista.

3.2. Implementación computacional directa

A partir de la expresión (3) la implementación computacional de la regresión parece sencilla en R. Observa que se trata de operaciones básicas matriciales que ya hemos practicado. A continuación te propongo que las realices para un ejemplo con datos generados de manera artificial correspondiente al caso de la regresión lineal simple (una sola variable predictora, esto es, $k = 1$).

Utiliza el código que aparece a continuación de este cuadro para generar una muestra de observaciones de las variables Y y X de tamaño $n = 5$ de la forma siguiente: las observaciones x_i corresponden a valores aleatorios desde una distribución normal estándar; y las observaciones de y_i se obtienen a partir de la ecuación del modelo $y_i = 1 + x_i + \epsilon_i$, donde ϵ_i son valores aleatorios generados desde una normal con media 0 y desviación típica 0.1. A partir de esos datos:

- Crea la matriz de regresión $\mathbf{X}$ y el vector de respuesta $\mathbf{y}$.
- Calcula $(\mathbf{X}'\mathbf{X})^{-1}$.
- Usando el resultado anterior calcula $\widehat{\boldsymbol{\beta}}$ a partir de la expresión (3).
- Observa que los datos se han generado verificando exactamente el modelo de regresión lineal (1) con $\boldsymbol{\beta} = (1, 1)'$. ¿Se parece el estimador por mínimos cuadrados que has obtenido a partir de los datos al verdadero $\boldsymbol{\beta}$ del modelo?
- Representa el modelo lineal desde el que se han generado los datos escribiendo `curve(1+x,-3,3)`. Después añade los datos que has generado escribiendo `points(x,y)` y finalmente la estimación del modelo que has calculado, usando de nuevo la función `curve` con argumento `add=TRUE` (y si quieres distinto color, por ejemplo rojo, con el argumento `col=2`).
- Repite el ejercicio para $n = 50$ y $n = 500$. ¿Qué diferencias observas?
- ¿Qué problema(s) puede tener en la práctica este tipo de implementación directa?

```
n<-5
set.seed(22)
x<-rnorm(n)
y<-1+x+rnorm(n,0,0.1)
```

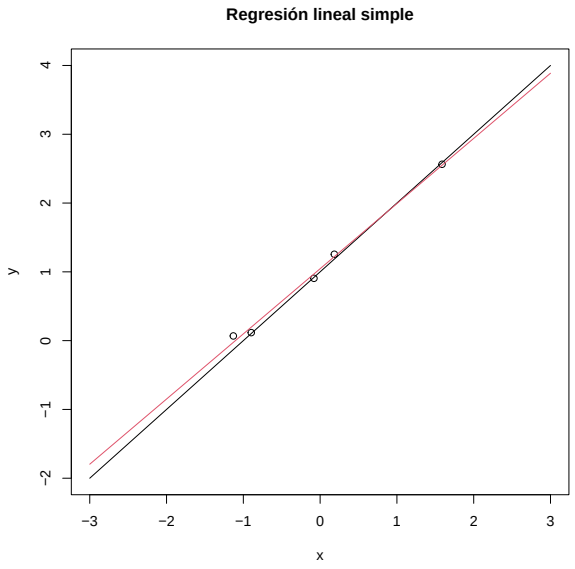
```
> n<-5
> set.seed(2)
> x<-rnorm(n)
> y<-1+x+rnorm(n,0,0.1)
> X<-cbind(1,x)
> XX<-crossprod(X)
> XXinv<-solve(XX)
> XXinv

                      x
  0.20097042 0.01449041
x 0.01449041 0.21637328
```

```
> beta<-XXinv%*%t(X)%*%y
> beta

       [,1]
  1.0453963
x 0.9472755

> curve(1+x,-3,3,main='Regresión lineal simple',ylab='y')
> points(x,y)
> curve(beta[1]+beta[2]*x,add=T,col=2)
```



3.3. Implementación usando la descomposición QR

> *La forma de una expresión matemática y el modo en que dicha expresión debe evaluarse en la práctica real puede ser bastante diferente.* [Gentle, 2009]

La factorización QR resulta muy útil para los cálculos en sistemas que involucran matrices no cuadradas $(n \times p)$ como la que aparece en el problema de la regresión. Dicha descomposición escribe la matriz como el producto de una matriz cuadrada ortogonal, Q $(n \times n)$, y una matriz triangular superior, R $(n \times p)$. Estas matrices tienen propiedades que permiten en muchos casos simplificar los cálculos, por ejemplo las matrices ortogonales verifican que $Q'Q = I$ y $\|Q'z\|_2 = \|z\|_2$.
Usando una descomposición QR de la matriz de regresión $\mathbf{X} = \mathbf{QR}$, el problema de mínimos cuadrados (2) se puede reescribir como sigue:

$$\begin{aligned} \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2 &= \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{QR}\boldsymbol{\beta}\|_2 = \min_{\boldsymbol{\beta}} \|\mathbf{Q}'\mathbf{y} - \mathbf{Q}'\mathbf{QR}\boldsymbol{\beta}\|_2 \\ &= \min_{\boldsymbol{\beta}} \|\mathbf{Q}'\mathbf{y} - \mathbf{R}\boldsymbol{\beta}\|_2. \end{aligned}$$

Observa que con esto hemos reescrito nuestro problema de minimización en términos de $\mathbf{Q}'\mathbf{y}$, un vector con n elementos, y el producto $\mathbf{R}\boldsymbol{\beta}$, donde $\mathbf{R}$ es una matriz triangular superior. Para el ejemplo con $n = 5$ observaciones que ejecutamos antes el problema se escribiría como:

$$\min_{\boldsymbol{\beta}} \left\| \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix} - \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} \right\|_2 ,$$

donde hemos usado la notación $(b_1, \ldots, b_n)' = \mathbf{Q}'\mathbf{y}$. Ahora observa que solo b_1 y b_2 dependen de la elección de $\boldsymbol{\beta}$. Si la matriz $\mathbf{X}$ es de rango completo entonces existe un único $\widehat{\boldsymbol{\beta}}$ que verifica:

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} - \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix} \begin{pmatrix} \widehat{\beta}_1 \\ \widehat{\beta}_2 \end{pmatrix} = 0 \tag{4}$$

y que por tanto nos dará la solución al problema de mínimos cuadrados anterior.
Vamos a usar ahora esta reformulación del problema para obtener el estimador por mínimos cuadrados (3). Para ello disponemos en R de la funciones `qr` que nos da la descomposición QR, junto con `qr.Q` y `qr.R` que extraen de Q y R, respectivamente, y la función `backsolve` que permite resolver un sistema de ecuaciones lineal con matriz de coeficientes triangular superior como es el sistema (4).

Utilizando la misma muestra de $n = 5$ observaciones que has generado antes calcula el estimador $\widehat{\boldsymbol{\beta}}$ usando la descomposición QR. Para ello puedes seguir los siguientes pasos:

- Obtén la descomposición QR de la matriz $\mathbf{X}$ con la función `qr`.
- Extrae la matriz Q aplicando la función `qr.Q` a la descomposición anterior. Observa que la matriz que te devuelve tiene dimensión $n \times 2$ (de hecho solo estas dos columnas son necesarias para los cálculos).
- Calcula el vector $\mathbf{Q}'\mathbf{y}$.
- Extrae la matriz R aplicando la función `qr.R` a la descomposición anterior. Observa que la matriz que te devuelve tiene dimensión 2×2.
- Resuelve el sistema (4) usando `backsolve`. Compara la solución con la que obtuviste con la implementación directa.

```
> QR<-qr(X)
> Q<-qr.Q(QR)
> Q

            [,1]        [,2]
[1,] -0.4472136  0.38605676
[2,] -0.4472136 -0.11713583
[3,] -0.4472136 -0.76975270
[4,] -0.4472136  0.49465340
[5,] -0.4472136  0.00617837

> b<-t(Q)%*%y
> R<-qr.R(QR)
> R

                         x
[1,] -2.236068  0.1497483
[2,]  0.000000 -2.1498006

> backsolve(R,b)

          [,1]
[1,] 1.0453963
[2,] 0.9472755
```

Veremos en el Tema 4 que en R la función estándar en el paquete *base* para calcular el estimador $\widehat{\boldsymbol{\beta}}$ es `lm`. Dicha función utiliza este tipo de implementación basado en la descomposición QR. Su uso lo veremos con más detalle en dicho tema si bien ahora puedes usarla para comprobar las soluciones que has dado antes con el conjunto de 5 observaciones. Para ello escribirías `lm(y~x)`, lo que te devolvería como resultado los coeficientes estimados en $\widehat{\boldsymbol{\beta}}$.

4. Ejercicio propuesto

Considera el sistema $Ax = b$ definido como sigue: Para $n = 3$ crea la matriz de coeficientes A como un matriz cuadrada `A` de dimensión n cuya primera columna sea $1, 2, \ldots, n$, la segunda $1^2, 2^2, \ldots, n^2$, hasta la última $1^n, 2^n, \ldots, n^n$. Crea un vector `b` como resultado del producto (matricial) de `A` por un vector de n unos. Con dichos objetos:

- Resuelve el sistema usando la función `solve`.
- Observa que tal y como hemos definido el sistema la solución x es un vector de n unos. Calcula el máximo de las diferencias $x - 1$ en valor absoluto.

- Repite el ejercicio para $n = 4, 5, \ldots 12$. Comenta en cada caso las dificultades del problema en relación a la condición de la matriz de coeficientes.

```
> n<-3
> x<-1:n
> A<-matrix(rep(1:n,times=n,each=n)^rep(1:n,times=n),ncol=n,nrow=n)
> b<-A%*%rep(1,n)
> solve(A,b)

     [,1]
[1,]    1
[2,]    1
[3,]    1

> rcond(A)

[1] 0.005128205

> ## Repetir para n=4,..., el mal condicionamiento se agrava hasta:
> n<-12
> x<-1:n
> A<-matrix(rep(1:n,times=n,each=n)^rep(1:n,times=n),ncol=n,nrow=n)
> b<-A%*%rep(1,n)
> solve(A,b)

Error in solve.default(A, b): sistema es computacionalmente singular: número
de condición recíproco = 3.95621e-17

> rcond(A)

[1] 3.95621e-17

> ## el mal condicionamiento es muy severo, aún así podemos resolverlo
> #  pero disminuyendo el argumento tolerance (tol) de la función solve
> cA<-rcond(A)
> solve(A,b,tol=cA)

          [,1]
 [1,] 0.9995348
 [2,] 1.0019621
 [3,] 0.9947548
 [4,] 1.0098035
 [5,] 0.9866145
 [6,] 1.0136050
```

```
 [7,] 0.9896751
 [8,] 1.0057876
 [9,] 0.9976687
[10,] 1.0006394
[11,] 0.9998929
[12,] 1.0000083
```