Capítulo 10
Álgebra lineal con aplicaciones de programación

La R El lenguaje tiene varios comandos de álgebra matricial incorporados. Esto resulta útil para los analistas que desean escribir sus propios estimadores o tienen otros problemas en álgebra lineal que desean calcular usando software. En algunos casos, es más fácil aplicar una fórmula para predicciones, errores estándar o alguna otra cantidad directamente en lugar de buscar un programa enlatado para calcular la cantidad, si existe. El álgebra de matrices hace que sea sencillo calcular estas cantidades usted mismo. Este capítulo presenta la sintaxis y los comandos disponibles para realizar álgebra matricial enR.

El capítulo continúa describiendo primero cómo un usuario puede ingresar datos originales a mano, como un medio para crear vectores, matrices y marcos de datos. Luego presenta varios de los comandos asociados con el álgebra lineal. Finalmente, trabajamos a través de un ejemplo aplicado en el que estimamos un modelo lineal con mínimos cuadrados ordinarios programando nuestro propio estimador.

Como datos de trabajo a lo largo del capítulo, consideramos un ejemplo simple de las elecciones al Congreso de Estados Unidos de 2010. Modelamos la participación del candidato republicano en el voto bipartidista en las elecciones para la Cámara de Representantes (**y)** en 2010. La entrada las variables son una constante ($\mathbf{X_1}$), La participación de Barack Obama en el voto presidencial bipartidista en 2008 ($\mathbf{X_2}$), y la situación financiera del candidato republicano en relación con el Demócrata en cientos de miles de dólares ($\mathbf{X_3}$). Para simplificar, modelamos las nueve carreras de la Cámara en el estado de Tennessee. Los datos se presentan en la Tabla10.1.

Electrónico suplementario material: La en línea versión de esto capítulo (doi: 10.1007 / 978-3-319-23446-5_10) contiene material complementario destinado a disponible usuarios autorizados.

JE Monogan III, *Análisis político con R,* Utilice R !,
DOI 10.1007 / 978-3-319-23446-5_10

Cuadro 10.1 Del congreso datos electorales de Tennessee en 2010

Distrito	Republicano**y)**	Constante (X_1)	Obama (X_2)	Financiamiento (X_3)
1	0,808	1	0,290	4.984
2	0,817	1	0.340	5.073
3	0.568	1	0.370	12.620
4	0.571	1	0.340	6.443
5	0.421	1	0.560	5.758
6	0,673	1	0.370	15.603
7	0,724	1	0.340	14.148
8	0.590	1	0,430	0.502
9	0,251	1	0,770	9.048

Nota: Datos de Monogan (2013a)

10.1 Creación de vectores y matrices

Como primera tarea, al asignar valores a un vector o matriz, debemos usar el comando de asignación tradicional (<-). El comandoC **C**Combina varios elementos componentes en un *vector* objeto.[1] Entonces para crear un vector a con los valores específicos de 3, 4 y 5:

```
a <- c (3,4,5)
```

Dentro C, todo lo que tenemos que hacer es separar cada elemento del vector con una coma. Como un ejemplo más interesante, supongamos que quisiéramos ingresar tres de las variables de Table 10.1: La participación de los republicanos en el voto bipartidista en 2010, la participación de Obama en el voto bipartidista en 2008 y la ventaja financiera republicana, todos como vectores. Teclearíamos:

```
Y <-c (.808, .817, .568, .571, .421, .673, .724, .590, .251) X2 <-c (.29, .34, .37, .34 ,. 56,
.37, .34, .43, .77) X3 <-c (4.984,5.073,12.620, -6.443, -5.758,15.603,14.148,0.502,

    - 9.048)
```

Siempre que ingrese datos en forma vectorial, generalmente debemos asegurarnos de haber ingresado el número correcto de observaciones. Lalargo El comando devuelve cuántas observaciones hay en un vector. Para verificar las tres entradas de nuestros vectores, escribimos:

```
largo); longitud (X2); longitud (X3)
```

Los tres de nuestros vectores deberían tener una longitud de 9 si se ingresaron correctamente. Observe aquí el uso del punto y coma (;). Si un usuario prefiere colocar varios comandos en una sola línea de texto, el usuario puede separar cada comando con un punto y coma en lugar de poner cada comando en una línea separada. Esto permite que los comandos simples se almacenen de forma más compacta.

[1] Alternativamente, cuando los elementos combinados son objetos complejos, C en su lugar crea un lista objeto.

Si queremos crear un vector que siga una **reps**patrón de alimentación, podemos usar el reps mando. Por ejemplo, en nuestro modelo de resultados electorales de Tennessee, nuestro término constante es simplemente un9 1 vector de 1s:

```
X1 <- rep (1, 9)
```

El primer término dentro reps es el término que se debe repetir y el segundo término es el número de veces que debe repetirse.

Los vectores secuenciales también son fáciles de crear. Un signo de dos puntos (:) indicaR para listar números enteros secuenciales desde el punto de inicio hasta el final. Por ejemplo, para crear un índice para el distrito del Congreso de cada observación, necesitaríamos un vector que contenga valores de 1 a 9:

```
índice <- c (1: 9)
```

Como acotación al margen, un comando más general es el seq comando, que nos permite definir los intervalos de un **seq**uencia, así como valores iniciales y finales. Por ejemplo, si por alguna razón quisiéramos crear una secuencia a partir de2 a 1 en incrementos de 0,25, escribiríamos:

```
e <- seq (-2, 1, por = 0.25)
```

Cualquier vector que creamos se puede imprimir en la pantalla simplemente escribiendo el nombre del vector. Por ejemplo, si simplemente escribimosY en el símbolo del sistema, nuestro vector de participación de votos republicanos se imprime en la salida:

```
[1] 0,808 0,817 0,568 0,571 0,421 0,673 0,724
[8] 0,590 0,251
```

En este caso, los nueve valores del vector Y están impresos. El número impreso entre llaves al comienzo de cada fila ofrece el índice del primer elemento de esa fila. Por ejemplo, la octava observación deY es el valor 0.590. Para vectores más largos, esto ayuda al usuario a realizar un seguimiento de los índices de los elementos dentro del vector.

10.1.1 Crear matrices

Pasando a las matrices, podemos crear un objeto de clase *matriz* de varias formas posibles. Primero, podríamos usar elmatriz comando: En el caso más simple posible, suponga que queremos crear una matriz con todos los valores iguales. Para crear un4 4 matriz B con cada valor igual a 3:

```
b <- matriz (3, ncol = 4, nrow = 4, byrow = FALSE)
```

La sintaxis del matriz El comando primero llama a los elementos que definirán la matriz: En este caso, enumeramos un solo escalar, por lo que este número se repitió para todas las celdas de la matriz. Alternativamente, podríamos listar un vector en su lugar que incluye suficientes entradas para llenar toda la matriz. Luego necesitamos especificar el número de columnas (ncol) y filasnrow). Por último, el byrow el argumento se establece en FALSO por defecto. Con elFALSO configuración, R llena la matriz columna por columna. ACIERTO rellenos de ajuste

la matriz fila por fila en su lugar. Como regla, siempre que cree una matriz, escriba el nombre de la matriz (B en este caso) en la consola de comandos para ver si el camino R Ingrese los datos que coincidan con lo que pretendía.

Una segunda opción simple es crear una matriz a partir de vectores que ya se han ingresado. Podemos**unir** los vectores juntos como **C**vectores de columna usando el cbind comando y como **r**ow vectores usando el rbind mando. Por ejemplo, en nuestro modelo de resultados electorales de Tennessee, necesitaremos crear una matriz de todas las variables de entrada en la que las variables definen las columnas y las observaciones definen las filas. Como hemos de fi nido nuestros tres vectores variables (y cada vector está ordenado por observación), podemos simplemente crear dicha matriz usandocbind:

```
X <-cbind (1, X2, X3)
X
```

La cbind El comando trata nuestros vectores como columnas en una matriz. Esto es lo que queremos, ya que una matriz de predicción define filas con observaciones y columnas con variables. La1 en el cbind El comando asegura que todos los elementos de la primera columna sean iguales a la constante 1. (Por supuesto, la forma en que diseñamos X1, también podríamos haber incluido ese vector). X en la consola, obtenemos la impresión:

```
     X2     X3
[1,] 1 0,29   4.984
[2,] 1 0,34   5.073
[3,] 1 0,37  12.620
[4,] 1 0,34  - 6.443
[5,] 1 0,56  - 5.758
[6,] 1 0,37  15.603
[7,] 1 0,34  14.148
[8,] 1 0,43   0.502
[9,] 1 0,77  - 9.048
```

Estos resultados coinciden con los datos que tenemos en la Tabla 10.1, por lo que nuestra matriz de covariables debería estar lista cuando estemos listos para estimar nuestro modelo.

Solo para ilustrar el rbind mando, R fácilmente combinaría los vectores como filas de la siguiente manera:

```
T <-rbind (1, X2, X3)
T
```

No formateamos datos como este, pero para ver cómo se ven los resultados, escriba T en el resultados de la consola en la impresión:

```
     [, 1]  [, 2]  [, 3]  [, 4]  [, 5]  [, 6]  [, 7]
1    1.000  1.000  1,00   1.000  1.000  1.000  1.000
X2   0.290  0.340  0,37   0.340  0.560  0.370  0.340
X3   4.984  5.073  12,62 -6,443 -5,758 15,603 14,148
     [, 8]  [, 9]
1    1.000  1.000
X2   0.430  0,770
X3   0,502 - 9.048
```

Por tanto, vemos que cada vector variable forma una fila y cada observación forma una columna. En la impresión, cuandoR carece de espacio para imprimir una fila completa en una sola línea, envuelve todas las filas a la vez, presentando así las columnas 8 y 9 más adelante.

En tercer lugar, podríamos crear una matriz utilizando subíndices. (Los detalles adicionales sobre los subíndices de vectores y matrices se presentan en la Sec.10.1.3.) A veces, al crear una matriz, el usuario conocerá todos los valores por adelantado, pero en otras ocasiones un usuario debe crear una matriz y luego completar los valores más tarde. En el último caso, es una buena idea crear "espacios en blanco" para completar designando cada celda como faltante (oN / A).
La característica interesante de esto es que un usuario puede identificar fácilmente una celda que nunca se llenó. Por el contrario, si se crea una matriz con algún valor numérico predeterminado, digamos 0, luego es imposible distinguir una celda que tiene un 0 predeterminado de una con un valor real de 0. Entonces, si quisiéramos crear un3 3 matriz nombrada blanco para completar más tarde, escribiríamos:

```
blanco <- matriz (NA, ncol = 3, nrow = 3)
```

Si luego quisiéramos asignar el valor de 8 a la primera fila, tercer elemento de columna, escribiríamos:

```
en blanco [1,3] <- 8
```

Si luego quisiéramos insertar el valor (D 3: 141592:: :) en la segunda fila, entrada de la primera columna, escribiríamos:

```
en blanco [2,1] <- pi
```

Si quisiéramos usar nuestro vector previamente definido **a** D .3; 4; 5 /0 para definir la segunda columna, escribiríamos:

```
en blanco [, 2] <- a
```

Luego podríamos verificar nuestro progreso simplemente escribiendo blanco en el símbolo del sistema, que imprimiría:

```
          [, 1] [, 2] [, 3]
[1,]       N / A     3     8
[2,] 3,141593       4   N / A
[3,] NA             5   N / A
```

A la izquierda de la matriz, el Se definen los términos de fila. En la parte superior de la matriz, el se definen los términos de la columna. Observe que todavía hay cuatro elementos codificadosN / A porque nunca se ofreció un valor de reemplazo.

Cuarto, en contraste con el llenado de matrices después de la creación, también podemos conocer los valores que queremos en el momento de crear la matriz. Como ejemplo simple, para crear un 2 2 matriz **W** en el que enumeramos el valor de cada celda columna por columna:

```
W <- matriz (c (1,2,3,4), ncol = 2, nrow = 2)
```

Observe que nuestro primer argumento ahora es un vector porque queremos proporcionar elementos únicos para cada una de las celdas de la matriz. Con cuatro elementos en el vector, tenemos el número correcto de entradas para un2 2 matriz. Además, en este caso, ignoramos labyrow argumento porque el valor predeterminado es completar por columnas. Por el contrario, si quisiéramos enumerar los elementos de la celda fila por fila en la matriz**Z,** simplemente estableceríamos byrow a CIERTO:

```
Z <- matriz (c (1,2,3,4), ncol = 2, nrow = 2, byrow = TRUE)
```

Tipo W y Z en la consola y observe cómo se ha reorganizado el mismo vector de entradas de celda pasando de una celda a otra.

Alternativamente, suponga que quisiéramos crear un 10 10 matriz **norte** donde cada entrada de celda era un **r**andom dibujar de un **norma**una distribución con media 10 y desviación estándar 2, o *N*.10; 4 /:

```
N <- matriz (rnorm (100, media = 10, sd = 2), nrow = 10, ncol = 10)
```

Porque rnorm devuelve un objeto de la clase de vector, nuevamente estamos listando un vector para crear nuestras entradas de celda. Larnorm El comando se basa en la distribución normal con nuestras especificaciones 100 veces, lo que proporciona el número de entradas de celda que necesitamos para un 10 10 matriz.

Quinto, en muchas ocasiones, querremos crear un *matriz diagonal* que contiene solo elementos en su principal **diag**onal (de arriba a la izquierda a abajo a la derecha), con ceros en todas las demás celdas. El comandodiag facilita la creación de dicha matriz:

```
D <- diag (c (1: 4))
```

Escribiendo D en la consola, ahora vemos cómo aparece esta matriz diagonal:

```
     [, 1] [, 2] [, 3] [, 4]
[1,]     1     0     0     0
[2,]     0     2     0     0
[3,]     0     0     3     0
[4,]     0     0     0     4
```

Además, si se inserta una matriz cuadrada en el diag comando, devolverá un vector de los elementos diagonales de la matriz. Por ejemplo, en una matriz de varianza-covarianza, los elementos diagonales son varianzas y se pueden extraer rápidamente de esta manera.

10.1.2 Conversión de matrices y tramas de datos

Un medio final de crear un objeto de matriz es con el como.matriz mando. Este comando puede tomar un objeto del*marco de datos* clase y convertirlo en un objeto de la *matriz* clase. Para un marco de datos llamadomis datos, por ejemplo, el comando mydata.2 <- como.matrix (mydata) coaccionaría los datos en forma matricial, haciendo que todas las operaciones matriciales subsiguientes fueran aplicables a los datos. Además, escribiendomydata.3 <- as.matrix (mydata [, 4: 8]) solo tomaría las columnas cuatro a ocho del marco de datos y las coaccionaría en una matriz, lo que permitiría al usuario subconjuntar los datos mientras crea un objeto de matriz.

De manera similar, supongamos que quisiéramos tomar un objeto del *matriz* clase y crea un objeto de la *marco de datos* clase, tal vez nuestros datos electorales de Tennessee de Table 10.1. En este caso, elas.data.frame el comando funcionará. Si simplemente quisiéramos crear un marco de datos a partir de nuestra matriz de covariables**X,** podríamos escribir:

```
X.df <- como.data.frame (X)
```

Si quisiéramos crear un marco de datos utilizando todas las variables de la Tabla, 10.1, incluida la variable dependiente y el índice, podríamos escribir:

```
tennessee <- como.data.frame (cbind (index, Y, X1, X2, X3))
```

En este caso, el cbind comando incrustado en as.data.frame de fi ne una matriz, que se convierte inmediatamente en un marco de datos. En total, entonces, si un usuario desea ingresar sus propios datos enR, la forma más sencilla será de fi nir vectores variables, unirlos como columnas y convertir la matriz en un marco de datos.

10.1.3 Subíndice

Como se mencionó en la sección de creación de matrices, llamar a elementos de vectores y matrices por sus subíndices puede ser útil para extraer la información necesaria o para realizar asignaciones. Para llamar a un valor específico, podemos indexar un vector**y** por el *norte*th elemento, usando Y [n]. Entonces, el tercer elemento del vector **y,** o y_3, es llamado por:

```
Y [3]
```

Para indexar un *nk* matriz **X** por el valor $X_{ij,}$ dónde I representa la fila y j representa la columna, usa la sintaxis X [i, j]. Si queremos seleccionar todos los valores del *j*a columna de **X,** nosotros podemos usar X [, j]. Por ejemplo, para devolver la segunda columna de la matriz **X,** tipo:

```
X [, 2]
```

Alternativamente, si una columna tiene un nombre, entonces el nombre (entre comillas) también se puede usar para llamar a la columna. Por ejemplo:

```
X [, "X2"]
```

Del mismo modo, si deseamos seleccionar todos los elementos del *I*la fila de **X,** nosotros podemos usar X [i,]. Para la primera fila de la matriz **X,** tipo:

```
X [1,]
```

Alternativamente, también podríamos usar un nombre de fila, aunque la matriz **X** no tiene nombres asignados a las filas.

Sin embargo, si lo deseamos, podemos crear **nombres de filas** para matrices. Por ejemplo:

```
nombres de fila (X) <- c ("Dist. 1", "Dist. 2", "Dist. 3", "Dist. 4",
      "Dist. 5", "Dist. 6", "Dist. 7", "Dist. 8", "Dist. 9")
```

Del mismo modo, el comando colnames permite al usuario definir **columna**umn **nombres.** Simplemente escribir nombres de fila (X) o colnames (X) sin hacer un encargo, R imprimirá los nombres de filas o columnas guardados para una matriz.

10.2 Comandos vectoriales y matriciales

Ahora que tenemos la sensación de crear vectores y matrices, recurrimos a comandos que extraen información de estos objetos o nos permiten realizar álgebra lineal con ellos. Como se mencionó antes, después de ingresar un vector o matriz en R, Además de imprimir el objeto en la pantalla para una inspección visual, también es una buena idea verificar y asegurarse de que las dimensiones del objeto sean correctas. Por ejemplo, para obtener el**largo** de un vector **a:**

```
longitud (a)
```

Del mismo modo, para obtener el **oscuro**ensiones de una matriz, escriba:

```
tenue (X)
```

La oscuro El comando primero imprime el número de filas, luego el número de columnas. La verificación de estas dimensiones ofrece una garantía adicional de que los datos de una matriz se han introducido correctamente.

En el caso de los vectores, los elementos se pueden tratar como datos y se pueden extraer cantidades resumidas. Por ejemplo, para sumar el**suma** de los elementos de un vector:

```
suma (X2)
```

Del mismo modo, para tomar el **significar** de los elementos de un vector (en este ejemplo, la media del voto de Obama por distrito en 2008):

```
media (X2)
```

Y tomar el **var**ianza de un vector:

```
var (X2)
```

Otra opción que tenemos para matrices y vectores es que podemos tomar muestras de un objeto dado con el comando muestra. Suponga que queremos una muestra de diez números de **NORTE,** nuestro 10 10 matriz de extracciones aleatorias de una distribución normal:

```
set.seed (271828183)
N <- matriz (rnorm (100, media = 10, sd = 2), nrow = 10, ncol = 10) s <- muestra
(N, 10)
```

El primer comando, set.seed hace que esta simulación sea más replicable. (Ver Cap.11 para obtener más detalles sobre este comando.) Esto nos da un vector llamado **s** de diez elementos aleatorios de **NORTE.** También tenemos la opción de aplicar el muestra comando a los vectores.[2]

La solicitar El comando es a menudo la forma más eficiente de hacer cálculos vectorizados. Por ejemplo, para calcular las medias de todas las columnas de nuestra matriz de datos de Tennessee**X:**

```
aplicar (X, 2, media)
```

[2]Para los lectores interesados en bootstrapping, que es una de las aplicaciones más comunes de muestreo a partir de datos, el enfoque más eficiente será instalar el bota package y pruebe algunos de los ejemplos que ofrece la biblioteca.

En este caso, el primer argumento enumera la matriz a analizar. El segundo argumento,2 dice R aplicar una función matemática a lo largo del *columnas* de la matriz. El tercer argumento,significar es la función que queremos aplicar a cada columna. Si quisiéramos la media del*filas* en lugar de las columnas, podríamos usar un 1 como segundo argumento en lugar de un 2. Cualquier función definida en R se puede utilizar con el solicitar mando.

10.2.1 Álgebra de matrices

También se encuentran disponibles comandos que son más específicos del álgebra matricial. Siempre que el usuario desee guardar la salida de una operación vectorial o matricial, se debe utilizar el comando de asignación. Por ejemplo, si por alguna razón necesitáramos la diferencia entre cada participación republicana del voto bipartidista y la participación de Obama en el voto bipartidista, podríamos asignar un vector**metro** a la diferencia de los dos vectores escribiendo:

```
m <- Y - X2
```

Con operaciones aritméticas, R es bastante fl exible, pero una palabra sobre cómo funcionan los comandos puede evitar confusiones en el futuro. Para suma (+) y resta (-): si los argumentos son dos vectores de la misma longitud (como es el caso en el cálculo de vectores **metro),** luego R calcula la suma o resta de vectores regulares donde cada elemento se suma o se resta de su elemento correspondiente en el otro vector. Si los argumentos son dos matrices del mismo tamaño, se aplica la suma o resta de matrices cuando cada entrada se combina con su entrada correspondiente en la otra matriz. Si un argumento es un escalar, entonces el escalar se agregará a cada elemento del vector o matriz.

Nota: Si se agregan dos vectores de longitud desigual o dos matrices de diferente tamaño, se genera un mensaje de error, ya que estos argumentos no son conformes. Para ilustrar esto, si intentamos sumar nuestra muestra de diez números de antes,**s,** a nuestro vector de La participación de Obama en los votos en 2008, **X2**, como sigue:

```
s + X2
```

En este caso recibiríamos una salida que deberíamos ignorar, seguida de un error mensaje en rojo:

```
[1] 11,743450        8.307068 11.438161 14.251645 10.828459
[6] 10.336895        9.900118 10.092051 12.556688 9.775185
Mensaje de advertencia:
En s + X2: la longitud del objeto más larga no es un múltiplo de la longitud del objeto
más corta
```

Este mensaje de advertencia nos dice que la salida es una tontería. Dado que el vector s tiene longitud 10, mientras que el vector X2 tiene longitud 9, que R ha hecho aquí se agrega el *décimo* elemento de s hacia *primero* elemento de X2 y usó esto como el décimo elemento de la salida. El mensaje de error que sigue sirve como recordatorio de que una entrada basura que rompe las reglas del álgebra lineal produce una salida basura que nadie debería usar.

Por el contrario, *, /, ^ (exponentes), Exp (función exponencial), y Iniciar sesión todos aplican la operación relevante a cada entrada escalar en el vector o matriz. Para nuestra matriz de predictores de Tennessee, por ejemplo, pruebe el comando:

```
X.sq <- X ^ 2
```

Esto devolvería una matriz que cuadra cada celda separada en la matriz **X.** Por un token similar, la operación de multiplicación simple (*) realiza la multiplicación elemento por elemento. Por tanto, si dos vectores tienen la misma longitud o dos matrices del mismo tamaño, la salida será un vector o matriz del mismo tamaño donde cada elemento es el producto de los elementos correspondientes. A modo de ilustración, multipliquemos la participación de Obama en el voto bipartidista por la ventaja financiera republicana de varias formas. (Las cantidades pueden ser tontas, pero esto ofrece un ejemplo de cómo el código funciona.) Pruebe, por ejemplo:

```
x2.x3 <- X2 * X3
```

El vector de salida es:

```
[1] 1.44536 1.72482          4.66940 -2.19062 -3.22448
[6] 5.77311 4.81032          0.21586 -6.96696
```

Esto corresponde a la multiplicación escalar elemento por elemento.

Más a menudo, el usuario realmente querrá realizar una multiplicación de matrices adecuada. En R, la multiplicación de matrices se realiza mediante la operación% *%. Entonces, si hubiéramos multiplicado nuestros dos vectores de participación de votos de Obama y ventaja financiera republicana De este modo:

```
x2.x3.inner <- X2% *% X3
```

R ahora devolvería el *producto Interno,* o producto escalar ($\mathbf{X}_2 \mathbf{X}_3$). Esto equivale a 6.25681 en nuestro ejemplo. Otra cantidad útil para la multiplicación de vectores es la *exterior producto,* o producto tensorial ($\mathbf{X}_2 \otimes \mathbf{X}_3$). Podemos obtener esto por *transponer* el segundo vector en nuestro código:

```
x2.x3.outer <- X2% *% t (X3)
```

La salida de este comando es un 9 9 matriz. Para obtener esta cantidad, el **t**comando ransposet) se utilizó.[3] En álgebra matricial, es posible que necesitemos convertir los vectores de fila en vectores de columna o viceversa, y el **t**La operación ranspose logra esto. De manera similar, cuando se aplica a una matriz, cada fila se convierte en una columna y cada columna se convierte en una fila.

Como un ejemplo más de multiplicación de matrices, la variable de entrada matriz **X** tiene tamaño 9 3, y la matriz **T** que permite a las variables definir filas es un 3 9 matriz. En este caso, podríamos crear la matriz**PAG** D **TX** porque el número de columnas de **T** es el mismo que el número de filas de **X.** Por tanto, podemos decir que las matrices son *conforme a la multiplicación* y resultará en una matriz de tamaño 3 3. Podemos calcular esto como:

[3] Una sintaxis alternativa habría sido X2% o% X3.

```
P <- T% *% X
```

Tenga en cuenta que si nuestras matrices no son aptas para la multiplicación, entonces R devolverá un mensaje de error.[4]

Más allá de la multiplicación de matrices y la operación de transposición, otra cantidad importante que es exclusiva del álgebra de matrices es la **det**erminante de una matriz cuadrada (o una matriz que tiene el mismo número de filas que de columnas). Nuestra matriz**PAG** es cuadrado porque tiene tres filas y tres columnas. Una razón para calcular el determinante es que una matriz cuadrada tiene una inversa solo si el determinante es distinto de cero.[5] Para calcular el determinante de **PAG,** escribimos:

```
det (P)
```

Esto arroja un valor de 691,3339. Por tanto, sabemos que**PAG** tiene una inversa.

Para una matriz cuadrada que se puede invertir, el *inverso* es una matriz que se puede multiplicar por la original para producir la matriz de identidad. Con elresolver mando, R bien **resolver** para la inversa de la matriz o transmitir que la matriz no es reversible. Para probar este concepto, escriba:

```
invP <- resolver (P)
invP% *% P
```

En la primera línea, creamos **PAG** 1, que es la inversa de **pag.** En la segunda línea, multiplicar **PAG** 1**PAG** y la impresión es:

```
                                X2                X3
     1.000000e + 00  - 6.106227e-16  - 6.394885e-14
X2 1.421085e-14       1.000000e + 00    1.278977e-13
X3 2.775558e-17  - 2.255141e-17         1.000000e + 00
```

Esta es la forma básica de la matriz identidad, con valores de 1 a lo largo de la diagonal principal (desde la parte superior izquierda a la inferior derecha) y valores de 0 fuera de la diagonal. Si bien los elementos fuera de la diagonal no se enumeran exactamente como cero, esto se puede atribuir a un error de redondeo enRparte. La notación científica para la segunda fila, primer elemento de la columna, por ejemplo, significa que los primeros 13 dígitos después del lugar decimal son cero, seguidos de un 1 en el decimocuarto dígito.

[4]Una variedad única de multiplicación de matrices se llama producto Kronecker (**H˝L).** El producto Kronecker tiene aplicaciones útiles en el análisis de datos de panel. Ver elkronecker comando en R para más información.

[5]Como otra aplicación en estadística, la función de verosimilitud para una distribución normal multivariante también recurre al determinante de la matriz de covarianza.

10.3 Ejemplo aplicado: Programación de regresión OLS

Para ilustrar las diversas operaciones de álgebra matricial que R tiene disponible, en esta sección trabajaremos un ejemplo aplicado calculando el estimador de mínimos cuadrados ordinarios (MCO) con nuestro propio programa utilizando datos reales.

10.3.1 Cálculo manual de OLS

Primero, para motivar el trasfondo del problema, considere la formulación del modelo y cómo lo estimaríamos a mano. Nuestro modelo de regresión lineal poblacional para la participación de votos en Tennessee es:**y** D **X** C **u.** En este modelo, **y** es un vector de la proporción de votos republicanos en cada distrito, **X** es una matriz de predictores (incluida una constante, la participación de Obama en los votos en 2008 y la ventaja financiera del republicano en relación con el demócrata), ˇ consiste en el coeficiente parcial para cada predictor, y **tu** es un vector de perturbaciones. Estimamosˇ̃ D .**X**0**X/** 1**X**0**y,** obteniendo la función de regresión muestral **y**sobredosis **X**ˇ̃.

Para empezar a calcular a mano, tenemos que definir **X.** Tenga en cuenta que debemos incluir un vector de 1 para estimar un término de intersección. En forma escalar, nuestra población modelo es: y_ID ˇ1X_{1I}C ˇ2X_{2I}C ˇ3X_{3I}C tu_I, dónde X_{1I}D 1 para todos *I.* Esto nos da la matriz de predictores:

X D 6

2 3
1 0:29 4: 984
6 5: 0737
6 7
61 0:34 7
6 7
61 0:37 12: 6207 6 7
6 7
1 0:34 6: 4437 61 7
6 7
0:56 5: 7587 61 7
6 7
6:37 15: 6037 7
6 7
6 7
61 0:34 14: 1487 41 0:43 0: 5025

1 0:77 9: 048

A continuación, premultiplicar **X** por su transposición:

X0**X** D 6

2 3
1 0:29 4: 984
6 5: 073 7
6 7
6 1 0:34 7
6 7
2 6 1 0:37 12: 620 7
6 7
1: 000 1: 000 1: 000 1: 000 1: 000 0: 1: 000 6 1 0:34 6: 443 7
6 7
4: 0291: 0841: 0370 0: 340 560 0: 370 0: 340 0: 430 0: 770 5 6 1 0:34 5: 758 7
6 7
4: 984 5: 073 12: 620 6: 443 5: 758 15: 603 14: 148 0: 502 9: 048 6 1 0:56 15: 603 7
6 7
6 1 0:37 7
6 7
6 1 0:34 14: 148 7
6 7
4 1 0:43 0: 502 5

1 0:77 9: 048

que funciona para:

$$\mathbf{X}_0\mathbf{X} \text{ D } \begin{matrix} 9\text{: } 00000 & 3\text{: } 81000 & 31\text{: } 68100 \\ 3\text{: } 81000 & 1\text{: } 79610 & 6\text{: } 256815 \\ 31\text{: } 68100 & 6\text{: } 25681 & 810\text{: } 24462 \end{matrix}$$

También necesitamos $\mathbf{X}_0\mathbf{y}$:

$$\mathbf{X}_0\mathbf{y} \text{ D } \begin{matrix} 1\text{: } 000 & 1\text{: } 000 & 1\text{: } 000 \\ 4\text{: } 984 & 5\text{: } 073 & 12\text{: } 620 \end{matrix} \quad \begin{matrix} 1\text{: } 000 \\ 6\text{: } 443 \end{matrix} \quad \begin{matrix} 1\text{: } 000 & 1\text{: } 000 & 1\text{: } 000 & 1\text{: } 000 \\ 5\text{: } 758 & 15 & \text{: } 603 & 14\text{: } 148 \end{matrix} \quad \begin{matrix} 1\text{: } 000 \\ 0\text{: } 502 \end{matrix} \quad \begin{matrix} 0\text{: } 808 \\ 9\text{: } 048 \end{matrix}$$

$$\begin{matrix} 0\text{: } 290 \\ 0\text{: } 340 \\ 0\text{: } 370 \end{matrix} \quad \begin{matrix} 0\text{: } 340 \end{matrix} \quad \begin{matrix} 0\text{: } 560 & 0\text{: } 370 \\ 0\text{: } 340 & 0\text{: } 430 \end{matrix} \quad \begin{matrix} 0\text{: } 770 \end{matrix} \quad \begin{matrix} 60\text{: } 8177 \\ 60\text{: } 5687 \\ 60\text{: } 5717 \\ 60\text{: } 4217 \\ 60\text{: } 6737 \\ 60\text{: } 7247 \\ 40\text{: } 5905 \end{matrix}$$

$$0\text{: } 251$$

que funciona para:

$$\mathbf{X}_0\mathbf{y} \text{ D } \begin{matrix} 5\text{: } 4230 \\ 2\text{: } 09435 \\ 28\text{: } 0059 \end{matrix}$$

Invertir a mano

La última cantidad que necesitamos es la inversa de $\mathbf{X}_0\mathbf{X}$. Haciendo esto a mano, podemos resolver por eliminación de Gauss-Jordan:

$$\text{Œ}\mathbf{X}_0\mathbf{X}\mathbf{j}\mathbf{I} \text{ D } \left[\begin{array}{ccc|ccc} 9\text{: } 00000 & 3\text{: } 81000 & 31\text{: } 68100 & 1\text{: } 00000 & 0\text{: } 00000 & 0\text{: } 00000 \\ 3\text{: } 81000 & 1\text{: } 79610 & 6\text{: } 25681 & 0\text{: } 00000 & 1\text{: } 00000 & 0\text{: } 000005 \\ 31\text{: } 68100 & 6\text{: } 25681 & 810\text{: } 24462 & 0\text{: } 00000 & 0\text{: } 00000 & 1\text{: } 00000 \end{array}\right]$$

Dividir la fila 1 entre 9:

$$\left[\begin{array}{ccc|ccc} 1\text{: } 00000 & 0\text{: } 42333 & 3\text{: } 52011 & 0\text{: } 11111 & 0\text{: } 00000 & 0\text{: } 00000 \\ 4 & 3\text{: } 81000 & 1\text{: } 79610 & 6\text{: } 25681 & 0\text{: } 00000 & 1\text{: } 00000 \\ 31\text{: } 68100 & 6\text{: } 25681 & 810\text{: } 24462 & 0\text{: } 00000 & 0\text{: } 00000 & 1\text{: } 00000 \end{array}\right] \quad \begin{matrix} 0\text{: } 00000 \\ 0\text{: } 000005 \end{matrix}$$

Reste 3,81 multiplicado por la fila 1 de la fila 2:

$$\left[\begin{array}{ccc|ccc} 1\text{: } 00000 & 0\text{: } 42333 & 3\text{: } 52011 & 0\text{: } 11111 & 0\text{: } 00000 & 0\text{: } 00000 \\ 4 & 0\text{: } 00000 & 0\text{: } 18320 & 7\text{: } 15481 & 0\text{: } 42333 & 1\text{: } 00000 \\ 31\text{: } 68100 & 6\text{: } 25681 & 810\text{: } 24462 & 0\text{: } 00000 & 0\text{: } 00000 & 1\text{: } 00000 \end{array}\right] \quad \begin{matrix} 0\text{: } 000005 \end{matrix}$$

Reste 31.681 veces la fila 1 de la fila 3:

2 3
1: 00000 0: 42333 3: 52011 0: 11111 0: 00000 0: 00000
40: 00000 0: 18320 7: 15481 0: 42333 1: 00000 0: 000005
0: 00000 7: 15481 698: 72402 3: 52011 0: 00000 1: 00000

Divida la fila 2 entre 0,1832:

2 3
1: 00000 0: 42333 3: 52011 0: 11111 0: 00000 0: 00000
40: 00000 1: 00000 39: 05464 2: 31077 5: 45852 0: 000005
0: 00000 7: 15481 698: 72402 3: 52011 0: 00000 1: 00000

Suma 7,15481 veces la fila 2 a la fila 3:

2 3
1: 00000 0: 42333 3: 52011 0: 11111 0: 00000 0: 00000 2:
40: 00000 1: 00000 39: 05464 31077 5: 45852 0: 000005
0: 00000 0: 00000 419: 29549 20: 05323 39: 05467 1: 00000

Dividir la fila 3 entre 419.29549:

2 3
1: 00000 0: 42333 3: 52011 0: 11111 0: 00000 0: 00000
40: 00000 1: 00000 39: 05464 2: 31077 5: 45852 0: 000005
0: 00000 0: 00000 1: 00000 0: 04783 0: 09314 0: 00239

Sumar 39.05464 veces la fila 3 a la fila 2:

2 3
1: 00000 0: 42333 3: 52011 0: 11111 0: 00000 0: 00000
40: 00000 1: 00000 0: 00000 4: 17859 9: 09607 0: 093345
0: 00000 0: 00000 1: 00000 0: 04783 0: 09314 0: 00239

Restar 3,52011 multiplicado por la fila 3 de la fila 1:

2 3
1: 00000 0: 42333 0: 00000 0: 27946 0: 32786 0: 00841
40: 00000 1: 00000 0: 00000 4: 17859 0: 9: 09607 0: 093345
00000 0: 00000 1: 00000 0: 04783 0: 09314 0: 00239

Reste .42333 multiplicado por la fila 2 de la fila 1:

2 3
1: 00000 0: 00000 0: 00000 2: 04838 4: 17849 0: 04792
40: 00000 1: 00000 0: 00000 4: 17859 9: 09607 0: 0: 093345 D ŒIj.X0X/ 1
00000 0: 00000 1: 00000 0: 04783 0: 09314 0: 00239

Como una pequeña arruga en estos cálculos manuales, podemos mira eso .X0X/ 1 está un poco apagado debido a error de redondeo. En realidad, debería ser una matriz simétrica.

Respuesta final a mano

Si posmultiplicamos.$\mathbf{X}_0\mathbf{X}/_1$ por $\mathbf{X}_0\mathbf{y}$ obtenemos:

$$\mathbf{.X}_0\mathbf{X}/_1\mathbf{X}_0\mathbf{y} \text{ D } \begin{matrix} 2 \\ 4 \\ 3 \end{matrix} \begin{matrix} 2: 04838 & 4: 17849 & 0: 04792 \\ 9: 09607 & 0: 0933454 & 2: 09435 \\ 0: 04783 & 0: 09314 & 0: 00239 \end{matrix} \begin{matrix} 32 \\ 1 \\ 28 \end{matrix} \begin{matrix} 5:4230 \\ 0: 0059 \end{matrix} \begin{matrix} 3 \\ 3 \\ 0 \end{matrix} \begin{matrix} 2 \\ 4 \\ 0 \end{matrix} \begin{matrix} 1:0178 \\ 1: 00185 \\ 0: 0025 \end{matrix} \begin{matrix} 3 \\ \text{D} \\ \end{matrix} \check{\mathbf{O}}$$

O en forma escalar: yO_ID 1: 0178 1: 0018X_{2I}C 0: 0025X_{3I}.

10.3.2 Escribir un estimador de MCO en R

Dado que invertir la matriz requirió tantos pasos e incluso se encontró con algún error de redondeo, será más fácil tener R haz parte del trabajo pesado por nosotros. (A medida que aumenta el número de observaciones o variables, encontraremos la ayuda computacional aún más valiosa). Para programar nuestro propio estimador MCO enR, los comandos de teclado que requerimos son:

• Multiplicación de matrices:% *%
• Transponer: t
• Matriz inversa: resolver

Sabiendo esto, es fácil programar un estimador para $\check{\mathbf{O}}$ D.$\mathbf{X}_0\mathbf{X}/_1\mathbf{X}_0\mathbf{y}$.

Primero debemos ingresar nuestros vectores variables usando los datos de la Tabla 10.1 y combine las variables de entrada en una matriz. Si aún no los ha introducido, escriba:

```
Y <-c (.808, .817, .568, .571, .421, .673, .724, .590, .251) X1 <- rep (1, 9)

X2 <-c (.29, .34, .37, .34, .56, .37, .34, .43, .77) X3 <-c (4.984,5.073,12.620, -6.443,
-5.758,15.603 , 14.148,0.502,
        - 9.048)
X <-cbind (X1, X2, X3)
```

Para estimar MCO con nuestro propio programa, simplemente necesitamos traducir el estimador $\mathbf{.X}_0\mathbf{X}/_1\mathbf{X}_0\mathbf{y}$ dentro R sintaxis:

```
beta.hat <-solve (t (X)% *% X)% *% t (X)% *% Y
beta.hat
```

Desglosando esto un poco: El resolver El comando comienza porque la primera cantidad es inversa,.$\mathbf{X}_0\mathbf{X}/_1$. Dentro de la llamada a resolver, el primer argumento debe ser transpuesto (de ahí el uso de t) y luego se posmultiplica por el no transpuesto matriz de covariables (de ahí el uso de% *%). Seguimos postmultiplicando la transposición de **X,** luego postmultiplicando el vector de resultados (**y).** Cuando imprimimos nuestros resultados, obtenemos:

```
                [, 1]
      1.017845630
X2  - 1.001809341
X3    0,002502538
```

Estos son los mismos resultados que obtuvimos a mano, a pesar de las discrepancias de redondeo que encontramos al invertir por nuestra cuenta. Escribiendo el programa enR, sin embargo, simultáneamente nos dio un control total sobre el estimador, siendo mucho más rápido que las operaciones manuales.

Por supuesto, una opción aún más rápida sería utilizar lata lm comando que usamos en el Cap. 6:

```
tennessee <- as.data.frame (cbind (Y, X2, X3)) lm (Y ~ X2 + X3,
data = tennessee)
```

Esto también produce exactamente los mismos resultados. En la práctica, siempre que se calculan las estimaciones de MCO, este es casi siempre el enfoque que queremos adoptar. Sin embargo, este procedimiento nos ha permitido verificar la utilidad deRcomandos de álgebra matricial. Si el usuario encuentra la necesidad de programar un estimador más complejo para el que no existe un comando predefinido, éste debería ofrecer herramientas relevantes para lograrlo.

10.3.3 Otras aplicaciones

Con estas herramientas básicas en la mano, los usuarios ahora deberían poder comenzar a programar con matrices y vectores. Algunas aplicaciones intermedias de esto incluyen: calcular errores estándar a partir de modelos estimados conoptimver Cap. 11) y valores predichos para formas funcionales complicadas (consulte el código que produjo la Fig. 9.3). Algunas de las aplicaciones más avanzadas que pueden beneficiarse del uso de estas herramientas incluyen: mínimos cuadrados generalizados factibles (incluidos mínimos cuadrados ponderados), optimización de funciones de verosimilitud para distribuciones normales multivariadas y generación de datos correlacionados utilizando**Chol**descomposición de esky (ver el chol mando). Muy pocos programas ofrecen la flexibilidad de estimación y programación queR lo hace siempre que el álgebra matricial es esencial para el proceso.

10.4 Problemas de práctica

Para estos problemas de práctica, considere los datos sobre las elecciones al Congreso en Arizona en 2010, presentados a continuación en la Tabla 10,2:

Cuadro 10.2 Del congreso datos electorales de Arizona en 2010

Distrito	Republicano**y)**	Constante (**X**1)	Obama (**X**2)	Financiamiento (X3)
1	0,50	1	0,44	9.11
2	0,65	1	0,38	8.31
3	0,52	1	0,42	8.00
4	0,28	1	0,66	8,50
5	0,52	1	0,47	7.17
6	0,67	1	0,38	5.10
7	0,45	1	0,57	5.32
8	0,47	1	0,46	18,64

Nota: Datos de Monogan (2013a)

1. Cree un vector que consista en la proporción republicana del voto bipartidista (**y)** en los ocho distritos electorales de Arizona. Usando cualquier medio que prefiera, cree una matriz,**X,** en el que las ocho filas representan los ocho distritos de Arizona, y las tres columnas representan un vector constante de unidades (**X**1), La participación de Obama en los votos en 2008 (**X**2), y el equilibrio financiero del republicano (**X**3). Imprima su vector y matriz, y pregunte R para calcular la longitud del vector y las dimensiones de la matriz. Están todos estos resultados son consistentes con los datos que ve en la Tabla 10,2?
2. Usando su matriz, **X,** calcular **X**0**X** utilizando Rcomandos de. Cual es tu resultado?
3. Utilizando Rcomandos, encuentre la inversa de su respuesta anterior. Es decir, calcular.**X**0**X/** 1. Cual es tu resultado?
4. Usando los datos de la tabla 10,2, considere el modelo de regresión *y*/D ˇ1*X*1/C ˇ2*X*2/C ˇ3*X*3/C *tu*/. Utilizando Rcomandos de álgebra matricial, calcular ˇÖ D .**X**0**X/** 1**X**0**y.** ¿Cuáles son sus estimaciones de los coeficientes de regresión, ˇÖ1, ˇÖ2, y 3? ¿Cómo se comparan sus resultados si calcula estos utilizando la lm comando del Cap. 6?