QR: Un paquete para la factorización QR sin rotación.

Juan Claramunt González

Abstract El paquete QR contiene la función QR(). Esta función permite realizar una factorización QR de cualquier matriz real sin rotación en las columnas. Esto se contrapone a la función qr() disponible en base R, que realiza la factorización QR utilizando rotación de columnas. De este modo, la función QR se asegura que el producto de las matrices Q y R da como resultado la matriz factorizada, a diferencia de la función qr().

Palabras clave: QR - factorización - sin rotación

Traduciendo un método de Matlab a R nos dimos cuenta de que la función qr() de base R no retornaba el resultado esperado. El producto de las matrices Q y R obtenidas con qr() no era igual a la matriz inicial. El motivo es que qr() llama a la rutinas DQRDC(2) de LINPACK y DGEQP3 o ZGEQP3 de LAPACK. Estas rutinas llevan a cabo la descomposición QR de una matriz utilizando rotación de columnas y esto da lugar al problema. Afortunadamente, en LAPACK también encontramos rutinas que llevan a cabo la factorización QR sin rotación. En nuestro paquete hemos incluido la función QR() que llama a la rutina DGEQRF, que a su vez factoriza QR sin rotación.

Veamos a continuación como utilizar el nuevo paquete y su comparación con las funciones qr(), qr.Q() y qr.R() de base R.

Primero, definamos una matriz aleatoria para factorizar. El paquete QR acepta matrices reales de cualquier tamaño. Una vez definida la matriz, ya podemos utilizar la función QR().

```
#Definimos la matriz
A<-matrix(runif(25,min = -100, max = 100), 5, 5)

#Aplicamos la funcion QR y observamos el resultado
QRres<-QR(A)
QRres$Q
##             [,1]         [,2]       [,3]        [,4]        [,5]
## [1,] -0.42309448  0.234923652  0.6929345 -0.08446852  0.52773935
## [2,] -0.05026825 -0.784271484  0.3661381 -0.43580225 -0.24168291
## [3,] -0.29397211 -0.381323169  0.1177431  0.86461649 -0.08214574
## [4,] -0.84749158  0.001992615 -0.4634462 -0.23454854 -0.10935555
## [5,]  0.11748442 -0.429322475 -0.3964074 -0.01915120  0.80272909
QRres$R
##           [,1]      [,2]      [,3]       [,4]      [,5]
## [1,] -106.2881 -78.51503 -44.49275  12.876767  57.37270
## [2,]    0.0000 112.95345 -44.35744   3.440147 -32.31233
## [3,]    0.0000   0.00000  88.80717 100.827403  29.61217
## [4,]    0.0000   0.00000   0.00000  88.449295 -49.37597
## [5,]    0.0000   0.00000   0.00000   0.000000  69.72084
```

La nueva función QR realiza la factorización QR y también retorna Q y R a diferencia de las funciones de base R, en las que es necesario realizar primero la factorización y después reconstruir las matrices Q y R.

Veamos ahora como lo haríamos usando base R y comparemos los resultados:

```
qrres<-qr(A)
qrQ<-qr.Q(qrres)
qrR<-qr.R(qrres)
#Comprobamos si las matrices son iguales
all.equal(qrQ,QRres$Q)
## [1] TRUE
all.equal(qrR,QRres$R)
## [1] TRUE
```

En este caso vemos que los resultados son iguales, pues la función qr() no ha requerido la rotación de las columnas, sin embargo, este no es siempre el caso.

En el siguiente ejemplo definimos una matriz para que qr() use rotación. En este caso, si comparamos los metodos, podemos ver que los resultados no son iguales, y sólo la factorización obtenida usando QR() da como resultado la matriz original, X, cuando se realiza el producto Q*R.

```
X <- cbind(1, rep(1:0, each = 3), rep(0:1, each = 3),
           rep(c(1,0,0), 2),  rep(c(0,1,0), 2),  rep(c(0,0,1),2))
QRresX<-QR(X)
qrresX<-qr(X)
qrX_Q<-qr.Q(qrresX)
qrX_R<-qr.R(qrresX)

#Comprobamos si las matrices son iguales
all.equal(qrX_Q,QRresX$Q)
## [1] "Mean relative difference: 0.7358707"
all.equal(qrX_R,QRresX$R)
## [1] "Mean relative difference: 1.131095"

#Q obtenida con QR()
QRresX$Q
##            [,1]       [,2]          [,3]        [,4]       [,5]       [,6]
## [1,] -0.4082483 -0.4082483  7.925939e-01 -0.04580286 -0.1308926  0.1386750
## [2,] -0.4082483 -0.4082483 -5.661385e-01 -0.13740858 -0.3926777  0.4160251
## [3,] -0.4082483 -0.4082483 -2.264554e-01  0.18321144  0.5235703 -0.5547002
## [4,] -0.4082483  0.4082483  1.110223e-16 -0.79391626  0.1308926 -0.1386750
## [5,] -0.4082483  0.4082483  2.498002e-16  0.39695813 -0.5796671 -0.4160251
## [6,] -0.4082483  0.4082483  2.775558e-16  0.39695813  0.4487746  0.5547002

#Q obtenida con qr()
qrX_Q
##            [,1]       [,2]       [,3]          [,4]       [,5]        [,6]
## [1,] -0.4082483 -0.4082483  0.5773503 -2.775558e-17  0.5745653  0.05663922
## [2,] -0.4082483 -0.4082483 -0.2886751 -5.000000e-01 -0.3363337  0.46926857
## [3,] -0.4082483 -0.4082483 -0.2886751  5.000000e-01 -0.2382317 -0.52590779
## [4,] -0.4082483  0.4082483  0.5773503  1.029760e-16 -0.5745653 -0.05663922
## [5,] -0.4082483  0.4082483 -0.2886751 -5.000000e-01  0.3363337 -0.46926857
## [6,] -0.4082483  0.4082483 -0.2886751  5.000000e-01  0.2382317  0.52590779

#Comprobamos si el producto de Q y R es igual a la matrix original, X.
all.equal(X,QRresX$Q%*%QRresX$R)
## [1] TRUE
all.equal(X,qrX_Q%*%qrX_R)
## [1] "Mean relative difference: 1"
```

Por lo tanto, únicamente QR() es capaz de llevar a cabo la factorización QR sin necesidad de rotación. De esta manera, las matrices Q y R resultantes siempre tendrán como producto la matriz original a factorizar.

El proyecto, a parte de ser interesante por tratar de solucionar el problema inicial, incluye el hecho de extender R usando una libreria externa de Fortran sin necesidad de tener conocimientos en esta lengua. Esto se realiza gracias a la función .C() que nos permite utilizar código compilado de C. En este código de C hacemos una llamada a la rutina deseada de Fortran usando la funcion F77_CALL disponible en C. En este punto, únicamente debemos prestar atención a los argumentos de entrada y salida. Para ello, es de gran ayuda netlib.org, que contiene una guía con todas las rutinas disponibles en LAPACK así como sus archivos de ayuda (E. Anderson and Sorensen 1999). En dichos archivos podemos observar los argumentos de entrada y salida, sus tipos de datos y como deben ser utilizados.

Extender R reutilizando código de otras lenguas nos permite ampliar las capacidades de nuestra lengua de un modo óptimo y con menor esfuerzo.

El paquete está disponible en GitHub (jclaramunt/QR) y próximamente lo estará en CRAN.

10 E. Anderson, C. Bischof, Z. Bai, and D. Sorensen. 1999. *LAPACK Users' Guide*. Third Edition. SIAM. http://www.netlib.org/lapack/lug/lapack_lug.html.