Exceedingly Simple Gram-Schmidt Code

Version 001, Februari 23, 2016

Contents

#Problem

The *QR decomposition* of a rectangular $n \times m$ matrix X of rank m is of the form $X = QR$, with Q $n \times m$ orthonormal and R non-singular and square upper-triangular of order m. If X has rank $r < m$ we can still make the decomposition, but we allow some columns of Q and some rows of R to be zero.

There are various ways to compute the QR decomposition. In this note we implement the *Gram-Schmidt* or *GS* method in both `R` and `C`. GS operates on each of the columns of X in turn, and replaces them by the columns of Q.

```
##      [,1] [,2] [,3]
## [1,]    1    1    0
## [2,]    2    1    1
## [3,]    3    0    3
## [4,]    4    1    3
## [5,]    5    1    4
```

```
##        [,1] [,2] [,3]
## [1,] 0.1348    1    0
## [2,] 0.2697    1    1
## [3,] 0.4045    0    3
## [4,] 0.5394    1    3
## [5,] 0.6742    1    4
```

1 Example

```r
x<-matrix (rnorm(12), 3, 4)
print (b <- solve (x[,1:3], x[,4]))
```

```
## [1] -0.4277343 -0.3265584 -0.7450780
```

```r
print(h <- gsrc(x))
```

```
## $q
##            [,1]      [,2]        [,3]          [,4]
## [1,] -0.9771127 0.2004259  0.07127587  2.775558e-17
## [2,]  0.1839479 0.6278182  0.75631178  0.000000e+00
## [3,]  0.1068362 0.7521129 -0.65031703 -5.551115e-17
##
## $r
##         [,1]      [,2]      [,3]       [,4]
## [1,] 2.29933 -1.954527 0.2748604 -0.5500276
## [2,] 0.00000  1.354862 1.0898730 -1.2544821
## [3,] 0.00000  0.000000 1.3501702 -1.0059821
## [4,] 0.00000  0.000000 0.0000000  0.0000000
##
## $rank
## [1] 3
```

```r
h$q[,1:3]
```

```
##            [,1]      [,2]        [,3]
## [1,] -0.9771127 0.2004259  0.07127587
## [2,]  0.1839479 0.6278182  0.75631178
## [3,]  0.1068362 0.7521129 -0.65031703
```

```r
x[,4]
```

```
## [1]  0.2143061 -1.6495992 -0.3480677
```

```r
colSums(x[,4]*h$q[,1:3])/b
```

```
## [1] 1.285910 3.841524 1.350170
```

2 Timing

```
set.seed (12345)
x<-matrix (rnorm (1000000L), 10000L, 100L)
library (microbenchmark)
mb<-microbenchmark(R = gs(x), C = gsrc(x), Q = qr(x), times = 100L)
mb
```

```
## Unit: milliseconds
##  expr       min        lq      mean    median        uq       max neval
##     R 901.55286 1070.5833 1129.0989 1123.8111 1186.3891 1562.5058   100
##     C  90.77601  113.7328  133.6758  129.8622  143.9768  280.4501   100
##     Q  88.92902  104.5294  115.5634  113.3968  123.7876  183.0908   100
```

Thus for this example the `C` code is about 8-10 times as fast as the `R` code. The QR decomposition that comes with `R`, based on Householder transformations, is again twice as fast.

In a personal communication Bill Venables pointed out (01/18/16) that the above timing comparisons are somewhat unfavorable to our routines, because the standard `qr` routines in `R` still have to dig Q and R out of the qr structure. So an alternative, and perhaps more suitable comparison, is

```
mb<-microbenchmark(R = gs(x), C = gsrc(x), Q = {qrx <- qr(x); list(q = qr.Q(qrx), r = qr
mb
```

```
## Unit: milliseconds
##  expr      min       lq      mean    median        uq       max neval
##     R 880.3235 987.0325 1100.4779 1086.1389 1201.6151 1573.2712   100
##     C  87.9257 112.7299  131.5274  124.2346  144.5255  244.5871   100
##     Q 263.1983 300.5889  348.3349  334.6637  384.1192  584.5522   100
```

Now `gsrc` is faster than `qr`, which now includes the cost of the copies and assignments. So a completely fair comparison will be somewhere in between the two benchmark results.

#Appendix: Code

```
dyn.load("gs.so")

gs <- function (x, eps = 1e-10) {
  n <- nrow (x)
  m <- ncol (x)
  q <- matrix (0, n, m)
  r <- matrix (0, m, m)
  h <- .C("gsc", x = as.double(x), q = as.double(q), r = as.double(r), n = as.integer(n)
  return (list (q = matrix(h$q, n, m), r = matrix (h$r, m ,m), rank = h$rank))
}
```

```c
#include <math.h>

void
gsc (double *x, double *q, double *r, int *n, int *m, int *rank, double *eps)
{
    int             i, j, l, jn, ln, jm, imax = *n, jmax = *m;
    double      s = 0.0;
    *rank = 0;
    for (i = 0; i < imax; i++)
        s += *(x + i) * *(x + i);
    if (s  >  *eps) {
        *rank = 1;
        s = sqrt(s);
        *r = s;
        for (i = 0; i < imax; i++)
            *(q + i) = *(x + i) / s;
    }
    for (j = 1; j < jmax; j++) {
        jn = j * imax;
        jm = j * jmax;
        for (l = 0; l < j; l++) {
            ln = l * imax;
            s = 0.0;
            for (i = 0; i < imax; i++)
                s += *(q + ln + i) * *(x + jn + i);
            *(r + jm + l) = s;
            for (i = 0; i < imax; i++)
                *(q + jn + i) += s * *(q + ln + i);
        }
        for (i = 0; i < imax; i++)
            *(q + jn + i) = *(x + jn + i) - *(q + jn + i);
        s = 0.0;
        for (i = 0; i < imax; i++)
            s += *(q + jn + i) * *(q + jn + i);
        if (s  > *eps) {
            s = sqrt(s);
            *rank = *rank + 1;
            *(r + jm + j) = s;
            for (i = 0; i < imax; i++)
                *(q + jn + i) /= s;
        }
    }
}
```

#NEWS

#References