Exceedingly Simple Gram-Schmidt Code

Version 001, Februari 23, 2016

Contents

1 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
```

2 Example

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

```
## [1]  -6.893000  -4.123938 -12.897062
```

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

```
## $q
##             [,1]        [,2]       [,3]         [,4]
## [1,] -0.11844105 -0.69732184 -0.7069045 1.110223e-15
## [2,]  0.05799203 -0.71555822  0.6961418 8.881784e-16
## [3,]  0.99126618 -0.04145693 -0.1251906 4.718448e-16
##
## $r
##           [,1]       [,2]       [,3]       [,4]
## [1,] 0.6085186 0.03183012 -0.3059848 -0.3794801
## [2,] 0.0000000 2.37522932 -0.7144064 -0.5815552
## [3,] 0.0000000 0.00000000  0.1069499 -1.3793392
## [4,] 0.0000000 0.00000000  0.0000000  0.0000000
##
## $rank
## [1] 3
```

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

```
##             [,1]        [,2]       [,3]
## [1,] -0.11844105 -0.69732184 -0.7069045
## [2,]  0.05799203 -0.71555822  0.6961418
## [3,]  0.99126618 -0.04145693 -0.1251906
```

```
x[,4]
```

```
## [1]  1.4255382 -0.5660859 -0.1793760
```

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

```
## [1] 0.05505296 0.14101936 0.10694988
```

3 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 710.92550 754.75110 782.00936 769.88231 788.81639 1294.3305   100
##     C  66.88900  75.66938  85.77960  83.28075  91.91974  139.8222   100
##     Q  20.86567  25.89670  35.80789  32.65716  35.84932   72.5158   100
##  cld
##    c
##   b
##  a
```

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 = q
mb
```

```
## Unit: milliseconds
##  expr       min        lq      mean    median        uq      max neval cld
##     R 717.53407 755.53813 794.34557 779.96082 839.45376 992.5760   100   c
##     C  66.71474  80.18216  92.59452  87.86315  94.70657 181.7817   100 a
##     Q  71.25455  84.91014 108.42579  96.15216 113.06730 189.5604   100  b
```

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.

4 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;
        }
    }
}
```

5 NEWS

6 References