Differentiating the QR Decomposition

Jan de Leeuw - University of California Los Angeles

Started March 13 2023, Version of March 15, 2023

Abstract

We derive formulas and compute the Jacobian of the QR decomposition. Code in R is given. Analytical and numerical derivatives are compared.

Contents

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome.

1 Introduction

For the convergence analysis of multivariate methods that iteratively use the QR decomposition $X = QR$ we need the derivatives of both Q and R with respect to X. There is no claim of originality here, I am sure the results have been derived and published many times before.

2 Perturbation

We assume X is $n \times m$ with $n \geq m$, and of full column rank $r = m$. If $r < m$ the QR decomposition is not uniquely defined, and differentiability becomes problematic.

The Gram-Schmidt algorithm, without pivoting, shows that the QR decomposition is indeed differentiable. If we perturb X to $X + (dX)$, then to the first order Q gets perturbed to $Q + (dQ)$ and

R to $R + (dR)$. In order to find dQ and dR we most solve the equations

$$
\begin{aligned}
[X + (dX)] &= [Q + (dQ)][R + (dR)], &(1)\\
[Q + (dQ)]'[Q + (dQ)] &= I, &(2)\\
\text{lt}(R + (dR)) &= 0. &(3)
\end{aligned}
$$

Here $\text{lt}(A)$ operator replaces the upper triangular part of a square matrix A by zeroes. These equations simplify to

$$
\begin{aligned}
(dX) &= Q(dR) + (dQ)R, &(4)\\
(dQ)'Q + Q'(dQ) &= 0, &(5)\\
\text{lt}(dR) &= 0. &(6)
\end{aligned}
$$

Equation (5) says that $(dQ)'Q$ is anti-symmetric, and (6) says dR is upper triangular. Write $dQ = QA + Q_\perp B$, with $Q_\perp$ an orthonormal basis for the null space of X.

If we premultiply both sides of equation (4) by Q' and postmultiply by R^{-1} we have

$$A + (dR)R^{-1} = Q'(dX)R^{-1}, \tag{7}$$

where A is anti-symmetric. It follows that

$$\text{lt}(A) = \text{lt}(Q'(dX)R^{-1}), \tag{8}$$

which gives the lower-triangular part of A and by anti-symmetry the upper-triangular part as well. Subtraction A from both sides of (7) gives $(dR)R^{-1}$ and thus dR.

Finally premultiplying (4) by $Q'_\perp$ and postmultiplying by R^{-1} gives

$$B = Q'_\perp(dX)R^{-1}, \tag{9}$$

and thus dQ.

The computations of dQ and dR are implemented in the R (R Core Team (2022)) function d_qr(), which takes arguments X and Y to form the perturbation $Z = X + Y$. Thus $dX = Y$ and the differentials are evaluated at X.

Here is a small example with some random matrices.

```r
set.seed(12345)
x <- matrix(rnorm(30), 10, 3)
y <- matrix(rnorm(30), 10, 3) / 100
h <- d_qr(x, y)
```

To show the quality of the linear approximation we compare QR decomposition $Z = Q_Z R_Z$ with $X = Q_X R_X$. But first the approximation of order zero. The sum of the absolute values of $Q_Z - Q_X$ is 0.1316906 and that of $R_Z - R_X$ is 0.0698008. For the linear approximation the sum of the absolute values of $Q_Z - (Q_X + dQ)$ is 0.0014707 and that of $R_Z - (R_X + dR)$ is 0.0014424.

3 Jacobian

To compute partial derivatives of Q and R with respect to X we use $Y = dX$ with a single element equal to one, and the rest zero. By taking each of the nm elements in turn, we find the partials and we can collect them in the Jacobian. For our small example there are 30 elements in X and there are 39 elements in R and Q. Thus the Jacobian is 39×30. Of course the partials of the lower triangle of R are always zero.

The computation of the Jacobian is in the R function p_qr(). To check our results we have also written p_qr_num(), which computes the Jacobian by using the numerical differentiation from the numDeriv package (Gilbert and Varadhan (2019)). As figure 1 shows, both numerical and analytic Jacobians are the same.

```r
par(pty="s")
pfor <- p_qr(x)
pnum <- p_qr_num(x)
plot(pnum, pfor)
```

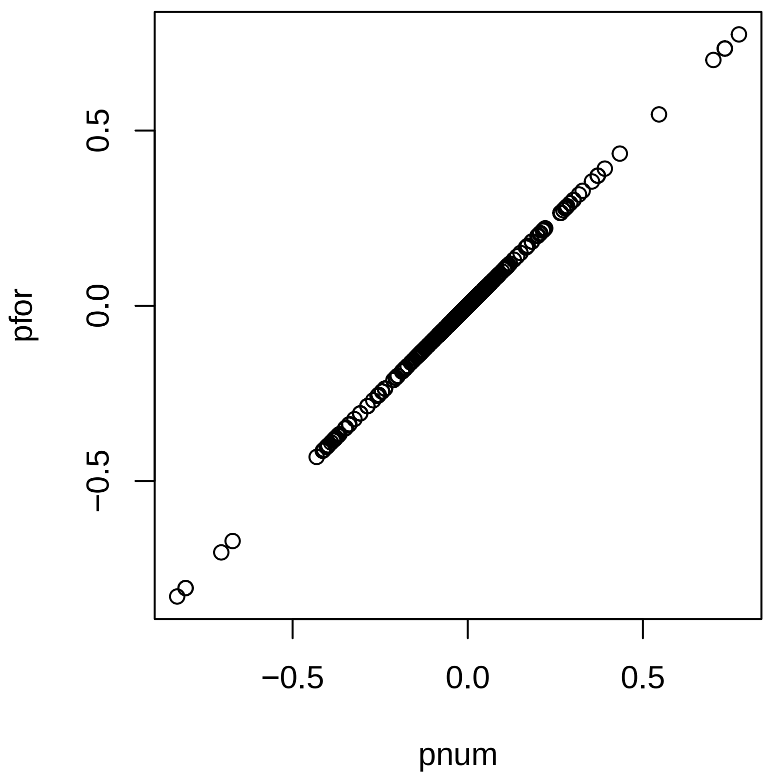

Figure 1: Numerical and Analytic Jacobians

4 Appendix: Code

4.1 d_qr.R

```
lt <- function(x) {
  n <- nrow(x)
  x[outer(1:n, 1:n, "<")] <- 0
  return(x)
}

d_qr <- function(x, y) {
  n <- nrow(x)
  m <- ncol(x)
  z <- x + y
  qrx <- qr(x)
  qx <- qr.Q(qrx)
  rx <- qr.R(qrx)
  qrz <- qr(z)
  qz <- qr.Q(qrz)
  rz <- qr.R(qrz)
  qp <- qr.Q(qr(cbind(qx, diag(n))))[, -(1:m)]
  ri <- solve(rx)
  v <- crossprod(qx, y %*% ri)
  a <- lt(v) - t(lt(v))
  b <- crossprod(qp, y %*% ri)
  dq <- qx %*% a + qp %*% b
  dr <- (v - a) %*% rx
  return(list(
    qx = qx,
    rx = rx,
    qz = qz,
    rz = rz,
    dq = dq,
    dr = dr
  ))
}

p_qr <- function(x) {
  n <- nrow(x)
  m <- ncol(x)
  qrx <- qr(x)
  qx <- qr.Q(qrx)
  rx <- qr.R(qrx)
  ri <- solve(rx)
```

```r
  qp <- qr.Q(qr(cbind(qx, diag(n))))[, -(1:m)]
  g <- matrix(0, (n * m) + m ^ 2, n * m)
  for (i in 1:n) {
    for (j in 1:m) {
      k <- i + (j - 1) * n
      v <- outer(qx[i, ], ri[j, ])
      a <- lt(v) - t(lt(v))
      b <- outer(qp[i, ], ri[j, ])
      dq <- qx %*% a + qp %*% b
      dr <- (v - a) %*% rx
      g[, k] <- c(as.vector(dq), as.vector(dr))
    }
  }
  return(g)
}



p_qr_num <- function(x) {
  n <- nrow(x)
  m <- ncol(x)
  f <- function(x, n = n, p = m) {
    xm <- matrix(x, n, p)
    qx <- qr(xm)
    q <- as.vector(qr.Q(qx))
    r <- as.vector(qr.R(qx))
    return(c(q, r))
  }
  g <- jacobian(f, as.vector(x), n = n, p = m)
  return(g)
}
```

References

Gilbert, P., and R. Varadhan. 2019. *numDeriv: Accurate Numerical Derivatives*. https://CRAN.R-project.org/package=numDeriv.

R Core Team. 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.