```
Finger Exercises to be discussed Wed Sept 25 2002
-------------------------------------------------

Consider the following data sets:

A. The Longley data (Monahan, p. 86), which is highly collinear.

 * Splus:
      X <- longley.x
      y <- longley.y
 * R:
      data(longley)
      X <- longley[,1:6]
      y <- longley[,7]


B. The Steel and Torre data (Monahan, p. 109 ex 5.2)
      X <- 1:6
      y <- c(4.5, 5.5, 6.5, 8.0, 10.0, 12.0)

-----------------------------------------------------------------------

1. Compute the least-squares regression coefficients for
   the Longley and Steel/Torre datasets, using each of the following
   methods.  Nb., don't forget to say X <- cbind(1,X)) to get the
   intercept in the model, before using methods (a), (b), (c).

    (a) Inverting XTX directly:
          XTX _ t(X) %*% X
          XTXinv _ solve(XTX)
    (b) Cholesky decomposition
          [use chol() to find the cholesky decomposition]
    (c) QR decomposition
          [use qr(), qr.Q() and qr.R() to find the QR decomposition]
    (d) lm in SPLUS or R
          [don't say X _ cbind(1,X) in this case]

   and compare the results.  [There will not be too much difference
   because Splus does all its calculations in double precision, and
   typically shows only 7 or so significant decimal digits.  How could
   you find differences in the lower-order digits?  Single precision
   calculations might not be able to handle the longley data, for
   example]

LONGLEY:
-------

(a) XTXinv:

The strategy is simple.  From X'Xb = X'y (' = transpose) we simply
want to write b = (X'X)^{-1} X'y.  So we convert X'X, invert it, etc...

> X _ cbind(1,X)
> XTX _ t(X) %*% X
> XTXinv _ solve(XTX)
Problem in solve.qr(a): apparently singular matrix
Use traceback() to see the call stack

It didn't work!  Apparently XTX is not (numerically) of full
rank---Splus does not think it possesses an inverse!

Since we will see below that other methods are able to produce a b-hat
here, we have to ask what's wrong with inverting XTX?

Although X has highly collinear columns, it is---mathematically---of
full rank.  Therefore XTX is mathematically full rank and has an
inverse.

However, computing XTX in machine floating point (even double
precision) can introduce roundoff, truncation and underflow errors
that make XTX "look" singular.  In particular,

> eigen(XTX)$values
[1] 6.665301e+07 2.090730e+05 1.053550e+05 1.803976e+04 2.455730e+01
[6] 2.015117e+00 1.172178e-07

shows that the smallest eigenvalue of the computed XTX is nearly 0;
and we could believe from this that XTX has only 6 nonzero
eigenvalues.  But then XTX is of rank 6, not 7, and doesn't have an
inverse.  An analogous check in obtaining the QR decomposition of XTX
as a prelude to computing XTXinv makes the same conclusion, and stops
cold.


(b) Cholesky:

Note: Monahan discusses the Cholesky decomposition A = LL' (' =
transpose), but checking help(chol) in Splus or R shows that the
decomposition that chol() produces is A = U'U, where U is
upper-triangular.  The solution strategy is the same though:

(i)   Change X'Xb = X'y into U'Ub = X'y
(ii)  Solve U'w = X'y for w
(iii) Solve Ub = w    for w

For (ii) and (iii) you either have to program your own backsolving
routine (like we did on the blackboard) or note the solve() and
backsolve() routines referenced at the bottom of help(chol).  We will
use solve() here.

> X _ cbind(1,X)
> XTX _ t(X) %*% X
> U _ chol(XTX)
> w _ solve(t(U),t(X) %*% y)
> b _ solve(U,w)
> b
                      [,1]
              -3.482259e+03
GNP deflator  1.506187e-02
         GNP -3.581918e-02
  Unemployed -2.020230e-02
Armed Forces -1.033227e-02
  Population -5.110411e-02
        Year  1.829151e+00

> round(b,7)
                      [,1]
              -3482.2585593
GNP deflator     0.0150619
         GNP    -0.0358192
  Unemployed    -0.0202023
Armed Forces    -0.0103323
  Population    -0.0511041
        Year     1.8291514

So using this approach it seems that XTX is, effectively,
invertible...  Hmmm...

(c) QR:

Here the strategy is as follows:

(i)   Change X'Xb = X'y into R'Q'QRb = R'Q'y, i.e. R'Rb = R'Q'y
(ii)  Note that if Rb = Q'y then also R'Rb = R'Q'y, so only need to...
(iii) ...Solve Rb = Q'y
```

```
> X _ cbind(1,X)
> X.qr _ qr(X)
> Q _ qr.Q(X.qr)
> R _ qr.R(X.qr)
> b _ solve(R , t(Q) %*% y)
> round(b,7)
                      [,1]
             -3482.2586346
GNP deflator     0.0150619
         GNP    -0.0358192
  Unemployed    -0.0202023
Armed Forces    -0.0103323
  Population    -0.0511041
        Year     1.8291515
```

Note that there is a function qr.coef (referenced in help(qr)) that does this all automatically:

```
>  qr.coef(qr(X),y)
           GNP deflator         GNP Unemployed Armed Forces  Population
 -3482.259   0.01506187 -0.03581918 -0.0202023  -0.01033227 -0.05110411
```

(d) USING lm:

```
> lm(y ~ X)
Call:
lm(formula = y ~ X)

Coefficients:
 (Intercept) XGNP deflator        XGNP XUnemployed XArmed Forces XPopulation
   -3482.259    0.01506187 -0.03581918  -0.0202023   -0.01033227 -0.05110411

    XYear
 1.829151

Degrees of freedom: 16 total; 9 residual
Residual standard error: 0.3048541
```

STEEL/TORRE:

(a) XTXinv:

```
> X _ cbind(1,X)
> XTX _ t(X) %*% X
> XTXinv _ solve(XTX)
> b _ XTXinv %*% t(X) %*% y
```

(b) Cholesky:

```
> X _ cbind(1,X)
> XTX _ t(X) %*% X
> U _ chol(XTX)
> w _ solve(t(U),t(X) %*% y)
> b _ solve(U,w)
> round(b,7)
  [,1]
   2.5
X  1.5
```

(c) QR:

```
> X _ cbind(1,X)
> X.qr _ qr(X)
> Q _ qr.Q(X.qr)
> R _ qr.R(X.qr)
> b _ solve(R , t(Q) %*% y)
> round(b,7)
  [,1]
   2.5
X  1.5
```

(d) USING lm:

```
> lm(y ~ X)
Call:
lm(formula = y ~ X)

Coefficients:
 (Intercept)   X
         2.5 1.5

Degrees of freedom: 6 total; 4 residual
Residual standard error: 0.5
```

[if you are using Splus, explore help(lm.fit) as well. This is the fitting function that lm uses. lm.fit in R is less versatile.]

help(lm.fit) shows that lm.fit is a switcher among the various numerical fitting functions (lm.fit.qr, lm.fit.chol, lm.fit.svd). The argument method does the switching: "qr" for lm.fit.qr, etc.

help(lm.fit.qr) gives help on each of these three methods for solving the normal equations for linear regression.

2. As described on pp. 84--85 of Monahan, the condition number for a linear regression bounds relative changes in the estimated regression coefficients in terms of relative changes in the response variables Y, or in the predictors X, in a regression problem.

   Computing the exact condition number is not hard for small problems:

   If X is the matrix of regressors (without the intercept), the condition number may be computed as:

```
XTX _ cbind(1,X)%*%cbind(1,X)
v _ eigen(XTX)$values
k _ sqrt(max(v)/min(v))
```

   Two alternatives:

```
XTX _ cbind(1,X)%*%cbind(1,X)
v _ svd(XTX)$d
k _ sqrt(max(v)/min(v))

v _ svd(cbind(1,X))
k _ max(v)/min(v)
```

   Argue that these are all the same (cf. Monahan p. 123).

   (There is also a function kappa() that computes a fast approximate condition number, but the exact calculation is usually fast enough.)

3. Compute exact (as in #2) and approximate (using kappa()) condition numbers for the following X's [don't forget to cbind with 1!]:

```
   (a) X _ rnorm(6)         # cbind(1,X) full rank

> X _ cbind(1,rnorm(6))
> v _ svd(X)$d
> max(v)/min(v)
[1] 1.445677
> kappa(X)
[1] 1.322598


   (b) X _ 1:6              # cbind(1,X) full rank, very mild collinearity

> v _ svd(cbind(1,1:6))$d
> max(v)/min(v)
[1] 9.359386
> kappa(cbind(1,1:6))
[1] 11.37986


   (c) X _ cbind(1:6,2:7)   # cbind(1,X) not full rank

>  v _ svd(cbind(1,1:6,2:7))$d
> max(v)/min(v)
[1] Inf
> v
[1] 15.318538  1.158612  0.000000

Note: v has 3 columns but only two nonzero singular values -- i.e. it
is of rank 2; So, it's not of full rank and the condition number is infinity!

> kappa(cbind(1,1:6,2:7))
[1] 8.296808e+17


   (d) X _ longley.x        # cbind(1,X) full rank, highly collinear

> X _ longley.x
> v _ svd(cbind(1,X))$d
> max(v)/min(v)
[1] 23845862
> kappa(X)
[1] 22219266


4. For 3(a), (b) and (c), use the y vector for the Steel/Torre data;
   for 3(d) use the y vector from the Longley data.  For all four
   examples, compare the regression coefficients in

   lm(y ~ X)       vs     y1 _ y + rnorm(length(y)); lm(y1 ~ X)

   and relate to the results in #3.

(a) Random X's

> X _ X[,-1]  # (starting from X _ cbind(1,rnorm(6)) in #3)
> y _ c(4.5, 5.5, 6.5, 8.0, 10.0, 12.0)
> y1 _ y + rnorm(length(y))
> lm(y ~ X)$coef
 (Intercept)         X
    7.776017 0.3681481
> lm(y1 ~ X)$coef
 (Intercept)          X
    7.486244 -0.2373551

We can directly compare this with the bound on the relative change in
b-hat vs relative change in y-hat discussed in class:

> b0 _ lm(y~X)$coef
> b1 _ lm(y1~X)$coef
> sqrt(sum((b1-b0)^2)/sum(b0^2))
[1] 0.08622899
> yh0 _ lm(y~X)$fitted
> yh1 _ lm(y1~X)$fitted
> sqrt(sum((yh1-yh0)^2)/sum(yh0^2))
[1] 0.0631556
> 0.08622899/0.0631556
[1] 1.365342

So, not too far off the condition number (1.446)...



(b) STEEL/TORRE: (X _ 1:6)

> X _ 1:6
> y _ c(4.5, 5.5, 6.5, 8.0, 10.0, 12.0)
> y1 _ y + rnorm(length(y))
> lm(y ~ X)$coef
 (Intercept)   X
         2.5 1.5
> lm(y1 ~ X)$coef
 (Intercept)        X
    2.118812 1.622246

> b0 _ lm(y~X)$coef
> b1 _ lm(y1~X)$coef
> sqrt(sum((b1-b0)^2)/sum(b0^2))
[1] 0.1373052
> yh0 _ lm(y~X)$fitted
> yh1 _ lm(y1~X)$fitted
> sqrt(sum((yh1-yh0)^2)/sum(yh0^2))
[1] 0.02620884
> 0.1373052/0.02620884
[1] 5.238889

This is a little less than we would expect, from the condition number
(9.359).  Further experimentation suggests that the condition number
is a little pessimistic in this case:

> kap _ NULL
> for (i in 1:100) {
+ y1 _ y + rnorm(length(y))
+ b0 _ lm(y~X)$coef
+ b1 _ lm(y1~X)$coef
+ num _ sqrt(sum((b1-b0)^2)/sum(b0^2))
+ yh0 _ lm(y~X)$fitted
+ yh1 _ lm(y1~X)$fitted
+ den _ sqrt(sum((yh1-yh0)^2)/sum(yh0^2))
+ kap _ c(kap, num/den)
+ }
> stem(kap)

N = 100   Median = 4.578734
Quartiles = 2.18714, 5.65403

Decimal point is at the colon

   0 : 7777888999
   1 : 14
   1 : 567888899
   2 : 1112223444
   2 : 8
   3 : 0
   3 : 556889
   4 : 01122244
```

```
   4 : 5556777888999
   5 : 00112223334
   5 : 5566778999
   6 : 11111124444
   6 : 55555666
```

(c) DEPENDENT X's (X _ cbind(1:6,2:7))

```
> X _ cbind(1:6,2:7)
> y _ c(4.5, 5.5, 6.5, 8.0, 10.0, 12.0)
> y1 _ y + rnorm(length(y))
> lm(y ~ X)$coef
Problem in lm.fit.qr(unset(x), unset(y)): computed fit is singular, rank 2
Use traceback() to see the call stack
> lm(y1 ~ X)$coef
Problem in lm.fit.qr(unset(x), unset(y)): computed fit is singular, rank 2
Use traceback() to see the call stack
```

Since X is (really!) non-full-rank, lm can't go on. To get some sort of fit anyway, we could try the "svd" method of lm:

```
> lm(y ~ X,method="svd")$coef
 (Intercept)        X1       X2
    1.166667 0.1666667 1.333333

> lm(y1 ~ X,method="svd")$coef
 (Intercept)         X1       X2
     1.69464 -0.1774483 1.517192
```

It would be tempting to compare these changes in b-hat with the condition number (infinity!) again, but this comparison doesn't make sense, since it is the condition number for a different problem (essentially, the full-rank regression problem)! Note that condition=infinity correctly characterizes the idea that b-hat should be in some linear subspace (and hence values of b0 and b1 can wander infinitely far from each other, regardless of the (unique) values of yh0 and yh1!).

(d) LONGLEY:

```
> lm(y~X)$coef
 (Intercept) XGNP deflator        XGNP XUnemployed XArmed Forces XPopulation
   -3482.259    0.01506187 -0.03581918  -0.0202023   -0.01033227 -0.05110411

    XYear
 1.829151


>  y1 _ y + rnorm(length(y)); lm(y1 ~ X)$coef
 (Intercept) XGNP deflator      XGNP XUnemployed XArmed Forces XPopulation
   -3755.022    -0.2557951 0.0521333 -0.01156115   -0.01268914    -1.10314

   XYear
 2.02731


> b0 _ lm(y~X)$coef
> b1 _ lm(y1~X)$coef
> sqrt(sum((b1-b0)^2)/sum(b0^2))
[1] 0.07833003
> yh0 _ lm(y~X)$fitted
> yh1 _ lm(y1~X)$fitted
> sqrt(sum((yh1-yh0)^2)/sum(yh0^2))
[1] 0.007769028
```

So the condition number for the Longley data provides a very crude bound on the sensitivity of b-hat to this (small) change in y-hat. Experimentation suggests that here too the condition number is pessimistic (though there is a suggestion of a long right tail that might warrant further investigation!):

```
> kap _ NULL
> for (i in 1:100) {
+ y1 _ y + rnorm(length(y))
+ b0 _ lm(y~X)$coef
+ b1 _ lm(y1~X)$coef
+ num _ sqrt(sum((b1-b0)^2)/sum(b0^2))
+ yh0 _ lm(y~X)$fitted
+ yh1 _ lm(y1~X)$fitted
+ den _ sqrt(sum((yh1-yh0)^2)/sum(yh0^2))
+ kap _ c(kap, num/den)
+ }
> stem(kap)

N = 100   Median = 70.6323
Quartiles = 41.47714, 101.1353

Decimal point is 1 place to the right of the colon

    0 : 2356667
    1 : 01123
    2 : 245
    3 : 45589
    4 : 0000122233567
    5 : 003458
    6 : 0002468899
    7 : 01123445689
    8 : 1124589
    9 : 01226679
   10 : 3499
   11 : 02336
   12 : 0379
   13 : 4
   14 : 3
   15 : 03458
   16 : 29
   17 :
   18 : 03
   19 : 1
```

However the large condition number (23,845,862) does suggest that numerical problems are likely, as we saw when we tried to compute (X'X)^{-1} directly.
