STAT 513/413: Lecture 4
Mostly linear algebra

(first non-trivialities perhaps)

A tale of expert code I: floating point arithmetics

Floating-point arithmetics: numbers are represented as

base $* 10^{\text{exponent}}$ - which has inevitable consequences

```
> 0.000001*1000000
[1] 1
> x=0; for (k in (1:1000000)) x=x+0.000001
> x
[1] 1
> x-1
[1] 7.918111e-12

> x=1000000; for (k in (1:1000000)) x=x+0.000001
> x
[1] 1000001
> x-1000000
[1] 1.000008
> x-1000001
[1] 7.614493e-06
```


The moral here is: with floating-point arithmetics, adding works well if the added numbers are about of the same magnitude

A better algorithm thus does it

```
> x=0; for (k in (1:1000000)) x=x+0.000001; x=x+1000000
> x
[1] 1000001
> x-1000000
[1] 1
> x-1000001
[1] 0
```


Yeah, but what to do in general? The solution seems to be: use addition programmed by experts.


```
> sum
function (..., na.rm = FALSE)  .Primitive("sum")

> x=sum(c(1000000,rep(0.000001,1000000)))
> x
[1] 1000001
> x-1000000
[1] 1
> x-1000001
[1] -2.561137e-09
```

Vectorization alone does not do it

```
> x=rep(1,1000001) %*% c(1000000,rep(0.000001,1000000))
> x-1000000
         [,1]
[1,] 1.000008
> x-1000001
             [,1]
[1,] 7.614493e-06
> x=crossprod(rep(1,1000001),c(1000000,rep(0.000001,1000000)))
> x-1000000
         [,1]
[1,] 1.000008
> x-1000001
             [,1]
[1,] 7.614493e-06
```

A tale of expert code II: never invert a matrix...

The theory for a linear model $y \sim X\beta$ suggests that you obtain the least squares estimates via the formula

$$b = (X^\top X)^{-1} X^\top y$$

However, in computing you are never ever (well, every rule has an exception, but still) supposed to do


```
b <- solve(t(X) %*% X) %*% t(X) %*% y
```


Doing alternatively


```
b <- solve(crossprod(X)) %*% crossprod(X, y)
```

does not really save it

... but rather solve (a system of) equations

It is much better to get b via solving the system

$$(X^\top X)b = X^\top y$$

To this end,

```
b <- solve(crossprod(X), crossprod(X, y))
```


may work pretty well; but experts know that the best way is via a so-called QR decomposition (MATLAB "backslash" operator), which in R amounts to


```
b <- qr.solve(X, y)
```


This is correct - but many people do not need to know that much; unless they are in certain special situations), they may just do

```
b <- coef(lm(y ~ X-1))
```

and it amounts to the same thing!

Showing the difference is, however, a bit intricate...

...because the numerics of R is *very* good... The first attempt

```
> x = runif(10,0,10)
> X = cbind(rep(1,length(x)),x)
> y = 2 + 3*x + rnorm(length(x))
> cbind(X, y)
                x         y
 [1,] 1 5.088385 18.846518
 [2,] 1 1.875540  8.434781
 [3,] 1 4.509448 16.397015
 [4,] 1 7.366187 24.432547
 [5,] 1 4.914751 18.399520
 [6,] 1 9.296908 29.273038
 [7,] 1 8.083712 26.970036
 [8,] 1 5.210684 16.587565
 [9,] 1 5.028429 15.727741
[10,] 1 8.422086 28.772038
```

The first attempt actually does not show anything

```
>  solve(t(X) %*% X) %*% t(X) %*% y
      [,1]
  2.715395
x 2.954821
> solve(crossprod(X)) %*% crossprod(X, y)
      [,1]
  2.715395
x 2.954821
> solve(crossprod(X), crossprod(X, y))
      [,1]
  2.715395
x 2.954821
> qr.solve(X, y)
                 x
2.715395 2.954821
> coef(lm(y~X-1))
       X        Xx
2.715395 2.954821
```

We have to be more extreme

```
> x = rep(1,1000)+rnorm(1000,0,.000001)
> X = cbind(rep(1,length(x)),x)
> y = 2 + 3*x + rnorm(length(x))
> det(crossprod(X))
[1] 1.133003e-06
> solve(t(X) %*% X) %*% t(X) %*% y
       [,1]
   20737.20
x -20732.22
> solve(crossprod(X), crossprod(X, y))
       [,1]
   20737.19
x -20732.21
> qr.solve(X, y)
                   x
 20733.83 -20728.85
> coef(lm(y ~ X-1))
        X         Xx
 20733.83 -20728.85
```

Vector and matrix algebra

`*`	is for componentwise multiplication (components better match!)
`%*%`	vector/matrix multiplication
`crossprod(A,B)`	$A^\top B$ (uses dedicated algorithm)
`crossprod(A)`	in particular $A^\top A$
`rep()`	a repetition function, very flexible
`solve(A, y)`	finds b such that $Ab = y$
`solve(A)`	finds A^{-1} (if needed be)
`c()`	concatenation of vectors, flexible too
`matrix()`	setting up matrices
`rbind(A,B)`	matrices are merged by rows (must match)
`cbind(A,B)`	matrices are merged by columns (must match)
`length()`	returns the length of a vector
`dim()`	returns the dimension of a matrix

Type conversions

General format `as.`*type*

```
> qr.solve(X, y)
                  x
 20733.83 -20728.85
> as.vector(qr.solve(X, y))
[1]  20733.83 -20728.85
> as.vector(coef(lm(y~X-1)))
[1]  20733.83 -20728.85
> as.vector(solve(crossprod(X), crossprod(X, y)))
[1]  20737.19 -20732.21
> as.vector(solve(t(X) %*% X) %*% t(X) %*% y)
[1]  20737.20 -20732.22
```


Note: in R, vectors are interpreted not linewise or columnwise, but in an "ambiguous manner": whatever suits more for a multiplication to succeed. In other words, the same square matrix can be multiplied by the same vector from both sides: `X %*% a` or `a %*% X` - which creates usually no problem, until we have an expression `a %*% a` which is always a number, $a^\top a$ for column vectors. If we want to obtain $a a^\top$, a matrix, we need to write `a %*% t(a)`

Potpourri

```
> numeric(4)
[1] 0 0 0 0
> rep(0,4)
[1] 0 0 0 0
> rep(c(0,1),4)
[1] 0 1 0 1 0 1 0 1
> rep(c(0,1),c(3,2))
[1] 0 0 0 1 1
> X=matrix(0,nrow=2,ncol=2)
> X=matrix(1:4,nrow=2,ncol=2)
> X
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> as.vector(X)
[1] 1 2 3 4
> as.matrix(1:4)
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
```

Finally, reminder

Inverse of a matrix should never *be computed, unless:*

- it is absolutely necessary to compute standard errors

- the number of right-hand sides is so much larger than n that the extra cost is insignificant

(this one is based on the following: solving *two* systems, $Ax = b_1$ and $Ax = b_2$ costs exactly that much as solving *one* system $Ax = b$ by first calculating A^{-1} and then $A^{-1}b$)

- the size of n is so small that the costs are irrelevant

(yeah, in the toy setting we don't care)

(*John F. Monahan, Numerical Methods of Statistics*)

(remarks by I.M.)

Some reminders from linear algebra

Useful formulae: $(AB)^\top = B^\top A^\top$

$\det(AB) = \det(A)\det(B) \qquad \det(A^\top) = \det(A)$

Useful definitions: we say that matrix A is
nonnegative definite (or positive semidefinite): $x^\top A x \geqslant 0$ for every x
positive definite: $x^\top A x > 0$ for every $x \neq 0$

The definitions imply that A is a square matrix; some automatically require that it is also symmetric, so better check (in statistics it is almost always symmetric matrices the definitions are applied to)

Useful habit in theory (albeit not observed by R in practice): consider vectors as $n \times 1$ columns (in statistics, it is always like this)

Useful caution: if a is an $n \times 1$ vector, then $a^\top a$ is a number (which we did denote by $\|a\|_2^2$), but $aa^\top$ is an $n \times n$ matrix. In general, matrix multiplication is not commutative: AB is in general different from BA

Useful principle: block matrices are multiplied in a same way as usual matrices, only blocks are itself matrices, thus multiplied as such, and hence the dimensions must match

Useful practice: check dimensions