Stat 243 Problem Set 7

Eugene Yedvabny

11/14/2014

Q1

The answers to the Chen *et al.* section questions were submitted on Monday but are also attached in hard copy to the end of this report.

Q2

a

By definition, if matrix A is eigen-decomposable, $A\vec{v} = \lambda\vec{v} = (\lambda I)\vec{v}$ or alternatively $(A - \lambda I)\vec{v} = 0$ for *any* non-zero $\vec{v}$ if λ is an eigenvalue of A. This condition is only true when $A - \lambda I$ is singular, which in turn has the property $det(A - \lambda I) = p(\lambda) = 0$, meaning the eigenvalues are *roots* of the characteristic polynomial $p(\lambda)$. For a generic value x, $p(x)$ can thus be written in terms of eigenvalues as $p(x) = (\lambda_1 - x)(\lambda_2 - x)...(\lambda_n - x)$. Putting it all together we get

$$det(A) = det(A - 0I) = p(0) = (\lambda_1 - 0)(\lambda_2 - 0)...(\lambda_n - 0) = \prod \lambda_i$$

b

$$\|A\| = sup\sqrt{(Az)^T Az} = sup\sqrt{(\Gamma\Lambda\Gamma^T z)^T(\Gamma\Lambda\Gamma^T z)} = sup\sqrt{(z^T\Gamma\Lambda\Gamma^T)(\Gamma\Lambda\Gamma^T z)} = sup\sqrt{(z^T\Gamma\Lambda^2\Gamma^T z)} = sup\sqrt{(y^T\Lambda^2 y)}$$

$$\|y\| = sup\sqrt{(\Gamma^T z)^T\Gamma^T z} = sup\sqrt{z^T\Gamma\Gamma^T z} = sup\sqrt{z^T z} = \|z\| = 1$$

The above transformation is possible since Γ is orthogonal, so $\Gamma^T = \Gamma^{-1}$ thus $\Gamma^T\Gamma = I$.

$$\|A\| = sup\sqrt{(y^T\Lambda^2 y)} = sup\sqrt{\sum \lambda_i^2 * y_i^2}$$

The norm of A is thus the max of the square root of the above sum. Since $\|y\| = 1$ the sum can be expressed simply as $\lambda_n^2 * 1 + \sum_{i \neq n} \lambda_i^2 * 0$ for any eigenvalue λ_n of A. The *maximum* of the sum is therefore the *largest* squared eigenvalue, meaning $\|A\|$ is the largest absolute value (square root of a square) eigenvalue of A.

Q3

First some naive calculations to check for validity:

```
library(microbenchmark)
n <- 1000
D <- diag(sample(1:9,n,replace=T)) # 10x10 diagonal random matrix
X <- replicate(n,sample(1:9,n,replace=T)) # 10x10 dense random matrix
summary(microbenchmark(DX <- D%*%X, unit='ms', times=3))$median
```

```
## [1] 76.84511
```

```
summary(microbenchmark(XD <- X%*%D, unit='ms', times=3))$median
```

```
## [1] 70.17527
```

In both cases these are $O(n^3)$ operations.

a

DX translates to multiplying every value in the nth *row* of X by the nth diagonal element of D. We can extract just the vector of the diagonal elements from D and do an element-by-element multiplication as $O(n^2)$

```
# diag(D) returns a vector of diagonal elements
summary(microbenchmark(DX_2 <- X * diag(D), unit='ms', times=3))$median
```

```
## [1] 4.853917
```

```
all.equal(DX_2,DX)
```

```
## [1] TRUE
```

b

XD is now a multiplication of every value in the nth *column* of X by the nth diagonal element of D. Since R is column-major, matrix*vector calculations won't work by default here. The easiest solution is to transpose X, multiply by diag vector of D and transpose again. The multiplication is again $O(n^2)$ and transpose is technically 'free' (but clearly from the timing below compared to DX above, reordering the matrix takes significant time).

```
summary(microbenchmark(XD_2 <- t(t(X)*diag(D)), unit='ms', times=3))$median
```

```
## [1] 11.10727
```

```
all.equal(XD_2,XD)
```

```
## [1] TRUE
```

Q4

a

The forward reduction of LU decomposition involves creating 0s below the diagonal of each columnn of A. There are (n-1) such 0s in the first column, (n-2) in the second column, etc until the (n-1)th column where there is one 0. Each 0 comes out from $A_{ij} - L_{ik} * A_{kj}$, where $L_{ik} = A_{ik}/A_{kk}$. Each zero of the reduced A thus requires (n-1) operations. The total flops are thus

$$\sum_{k=1}^{n-1}(n-k)(n-k) = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

Per the derivation in class, each steps of the Cholesky decomposition is more expensive, but there are a lot fewer steps: $\frac{n^3}{6} + O(n)$. The inclusion of $b \rightarrow b^*$ calculation is a matrix * vector calculation and we already have all the L elements from factorization above. There is one less dimension, so flops $= \sum_{k=0}^{n-1} n - k = \frac{n(n-1)}{2} = \frac{n^2}{2} + O(n)$

b

Backward elimination is $\sum_{k=1}^{n-1} k = \frac{n(n+1)}{2} = \frac{n^2}{2} + O(n)$

c

The additional dimension does not affect the decomposition, only the forward and backward eliminations. The calculation would have to be performed on every column of B, so the final flop cost is p * *the calculated values above.*

d

The LU decomposition cost is $\frac{n^3}{3} + O(n^2)$ and yields U and components of L. Finding A^{-1} is then a matter of applying backwards elimination (an $O(n^2)$ operation) n times. So we get $\frac{n^3}{3} + n * n^2 \approx n^3$

e

VB is just plain matrix multiplication and both matrices are dense, so the calculation is (# of mult per value in product)(number of values in product) $= n * n * p = n^2 p$

f

From **c** we have cost $A^{-1}B = \frac{p*n^3}{3} + \frac{p*3*n^2}{2} + O(n)$

From **e** we have cost $A^{-1}B = n^3 + pn^2 + O(n^2)$

Since the n^3 component is independent of p, the invert-and-multiply approach is more efficient if $p > 3$.

g

```
test_decomp <- function(n,p){
  A <- crossprod(matrix(rnorm(n^2), n))
  B <- matrix(rnorm(n*p),n)

  # Using LU
  m1 <- summary(microbenchmark({
    S1 <- solve(A,B)
  },unit='ms', times=3))$median

  # Using inversion
  m2 <- summary(microbenchmark({
    A_inv <- solve(A)
    S2 <- A_inv %*% B
  },unit='ms', times=3))$median

  # Using Cholesky
  m3 <- summary(microbenchmark({
    U <- chol(A)
    S3 <- backsolve(U,B)
  },unit='ms', times=3))$median
```

```
  print(sprintf("A: %d x %d, B: %d x %d",n,n,n,p))
  print(sprintf("LU: %f, Inv: %f, Chol: %f ms",m1,m2,m3))
  all.equal(S1,S2,S3)
}

test_decomp(100,1)
```

```
## [1] "A: 100 x 100, B: 100 x 1"
## [1] "LU: 0.193955, Inv: 0.248173, Chol: 0.109886 ms"

## [1] TRUE
```

```
test_decomp(100,100)
```

```
## [1] "A: 100 x 100, B: 100 x 100"
## [1] "LU: 0.198170, Inv: 0.275316, Chol: 0.127304 ms"

## [1] TRUE
```

```
test_decomp(100,3000)
```

```
## [1] "A: 100 x 100, B: 100 x 3000"
## [1] "LU: 1.796815, Inv: 1.811395, Chol: 1.035625 ms"

## [1] TRUE
```

```
test_decomp(3000,1)
```

```
## [1] "A: 3000 x 3000, B: 3000 x 1"
## [1] "LU: 444.339321, Inv: 1749.267582, Chol: 215.136574 ms"

## [1] "Mean relative difference: 1.593766e-14"
```

```
test_decomp(3000,100)
```

```
## [1] "A: 3000 x 3000, B: 3000 x 100"
## [1] "LU: 480.707831, Inv: 1576.328340, Chol: 223.994262 ms"

## [1] TRUE
```

```
test_decomp(3000,3000)
```

```
## [1] "A: 3000 x 3000, B: 3000 x 3000"
## [1] "LU: 1485.710283, Inv: 2996.795274, Chol: 788.117143 ms"

## [1] "Mean relative difference: 2.294081e-15"
```

Interestingly enough, the above timings suggest I am *waaaay* off on my predictions. LU is always faster than invert+multiply, regardless of p. The Cholesky approach always wins the race and is, expectedly, 2x the speed of LU by virtue of having half of the operations.

h

For many different Bs, LU decomposition would have to be done for each B, while A^-1 is a single expensive computation that can be substituted into many Bs. Matrix multiplication grows linearly with the number of columns of B. LU and Cholesky likewise grow linearly as the number of backward eliminations changes with each B. But bacward eliminations only operate on one half of the matrix, so the prefactor is smaller. Thus computing $(LU)^{-1}B$ is still probably going to be faster.