STAT 545 HW 5

Bowen Zheng

2026-02-18

Problem 1

A.

We can try to see how long it takes each of the parts to compute to find out.

Inverting D would be just taking reciprocal of p items, so it is $O(p)$ complexity.

Finding X^TX would be multiplying a $p \times n$ and $n \times p$ matrices. This would take $O(p^2n)$.

Adding two matrices $X^TX + D^{-1}$ would only take $p \times p$ computations, so $O(p^2)$.

Finally, the most complex step, the last one is inverting the $p \times p$ matrix so $O(p^3)$.

The biggest / dominating computational complexity is in this last step and thus our computational complexity for directly computing Σ is $O(p^3)$.

B.

We choose the obvious matrices to match up with the Woodbury matrix identity:

$A = D^{-1}$ (a $p \times p$ diagonal matrix)

$U = X^T$ (a $p \times n$ matrix)

$B = I_n$ (an $n \times n$ identity matrix)

$V = X$ (an $n \times p$ matrix)

Then we find

$$\Sigma = D - DX^T(I_n + XDX^T)^{-1}XD$$

.

Again, we can look at each step again to figure out the complexity.

Multiplying $ DXˆT$ or XD is simply one computation per thing so we get $O(np)$.

Computing the XDX^T will be $O(np)$, then multiplying the $n \times p$ by $p \times n$ will be $O(n^2p)$.

Adding $I_n + XDX^T$ will be adding two $n \times n$ so $O(n^2)$.

Inverting $I_n + XDX^T$ will be inverting $n \times n$ so $O(n^3)$.

Matrix multiplication of the parts together will be $n \times p$ by $p \times n$ or $p \times n$ by $n \times p$ so $O(np^2)$ or $O(n^2p)$.

Since $p > n$, our computational complexity $O(np^2)$ should mean it is less complex than in part (A.) where it was $O(p^3)$.

C.

First the distribution

In step 3, we know $(XDX^T + I_n)w = (y - v)$, then $w = (XDX^T + I_n)^{-1}(y - v)$, where $v = Xu + \delta$. Since $E[u] = 0$ and $E[\delta] = 0$, it follows that $E[v] = 0$. Then, $E[w] = (XDX^T + I_n)^{-1}(y - 0)$ and with step 4, $\theta = u + DX^T w$.

We can then find the expectation $E[\theta] = E[u] + DX^T E[w] = 0 + DX^T(XDX^T + I_n)^{-1}y$.

Now, We find that $X^T(XDX^T + I_n) = X^T XDX^T + X^T = (D^{-1} + X^T X)DX^T$. We multiply by $(D^{-1} + X^T X)^{-1}$ on the left and multiply by $(XDX^T + I_n)^{-1}$ on the right, to find

$$(D^{-1} + X^T X)^{-1}X^T(XDX^T + I_n)(XDX^T + I_n)^{-1} = (D^{-1} + X^T X)^{-1}(D^{-1} + X^T X)DX^T(XDX^T + I_n)^{-1}$$

$$(D^{-1} + X^T X)^{-1}X^T = DX^T(XDX^T + I_n)^{-1}$$

Note that this is a very useful identity we will use multiple times. Another one we will provide now:

$$\begin{aligned} \Sigma &= (D^{-1} + X^T X)^{-1} \\ \Sigma^{-1} &= D^{-1} + X^T X \\ I_p &= \Sigma D^{-1} + \Sigma X^T X \\ (I_p - \Sigma X^T X) &= \Sigma D^{-1} \end{aligned}$$

Rewrite $(D^{-1} + X^T X)^{-1}X^T = (X^T X + D^{-1})^{-1}X^T = \Sigma$. Then, from earlier

$$\begin{aligned} E[\theta] &= 0 + DX^T(XDX^T + I_n)^{-1}y \\ &= (X^T X + D^{-1})^{-1}X^T y \\ &= \Sigma X^T y \\ &= \mu \end{aligned}$$

Now we need to solve for the covariance matrix. Again, we use step 4 identity: $\theta = u + DX^T w$. We substitute in our w, our formula we just solved for the Σ, remember that $u\delta$ independent and find

$$\begin{aligned} \theta &= u + DX^T(XDX^T + I_n)^{-1}(y - v) \\ &= u + DX^T(XDX^T + I_n)^{-1}(y - Xu - \delta) \\ &= (I_p - DX^T(XDX^T + I_n)^{-1}X)u + (DX^T(XDX^T + I_n)^{-1})(y - \delta) \\ &= (I_p - (D^{-1} + X^T X)^{-1}X^T X)u + (D^{-1} + X^T X)^{-1}X^T(y - \delta) \\ &= (I_p - \Sigma X^T X)u + \Sigma X^T(y - \delta) \\ \implies Var(\theta) &= (I_p - \Sigma X^T X)Var(u)(I_p - \Sigma X^T X)^T + (\Sigma X^T)I_n(\Sigma X^T)^T \\ &= (\Sigma D^{-1})D(\Sigma D^{-1})^T + (\Sigma X^T)I_n(\Sigma X^T)^T \\ &= \Sigma D^{-1}\Sigma + \Sigma X^T X\Sigma \\ &= \Sigma(D^{-1} + X^T X)\Sigma \\ &= \Sigma\Sigma^{-1}\Sigma \\ &= \Sigma \end{aligned}$$

θ is a linear combination of normal distributions with mean μ and variance Σ. Thus, $\theta \sim N(\mu, \Sigma)$.

Computational complexity

Step 1: We first sample p and then n, so $O(p+n)$ complexity there.

Step 2: We have to do matrix vector multiplication, $n \times p$ by $p \times 1$ so $O(np)$ and then add the two vectors $O(n)$.

Step 3: To solve for w, we have to take the inverse of the $n \times n$ $(XDX^T + I_n)$ and then multiply it by $(y - v)$, which is $O(n^2 p)$ and $O(n^3)$. Since $p > n$, then $O(n^2 p)$ is more.

Step 4: This final step just involves addition and multiplication of matrices. Adding is $O(n)$ and multiplication is $O(np)$.

The most dominant term is $O(n^2 p)$ coming from step 3 that takes $O(n^2 p + n^3)$ computational complexity.

D.

> Take n = 50, p = 100. Simulate X and ϵ from i.i.d. standard normals. Take β to be a vector of all ones and set $y = X\beta + \epsilon$. Take D = I and implement the sampling schemes of parts (a) , (b) and (c) to generate 1,000 samples each. Report the computational times. You may use the Cholesky decomposition function in R.

```r
# Data Generation
set.seed(218)
n <- 50
p <- 100
n_samples <- 1000


X <- matrix(rnorm(n * p), n, p)
epsilon <- rnorm(n)
beta_true <- rep(1, p)
y <- X %*% beta_true + epsilon
D_diag <- rep(1, p)
sqrt_D_diag <- sqrt(D_diag)
D <- diag(D_diag)

# Part A: Doing steps then sampling
time_a <- system.time({
  D_inv <- solve(D)
  Precision <- t(X) %*% X + D_inv
  Sigma <- solve(Precision)
  mu <- Sigma %*% t(X) %*% y
  L <- t(chol(Sigma))
  samples_a <- replicate(n_samples, mu + L %*% rnorm(p))
})

# Part B: Woodbury for Sigma
time_b <- system.time({
  In <- diag(n)
  D_inv <- solve(D)
  inner_inv <- solve(In + X %*% D_inv %*% t(X))
  Sigma_w <- D_inv - D_inv %*% t(X) %*% inner_inv %*% X %*% D_inv
  mu_w <- Sigma_w %*% t(X) %*% y
  L_w <- t(chol(Sigma_w))
```

```
  samples_b <- replicate(n_samples, mu_w + L_w %*% rnorm(p))
})
# Part C: Algorithm
time_c <- system.time({

  In <- diag(n)
  inner_mat <- X %*% D %*% t(X) + In

  samples_c <- replicate(n_samples, {
    u <- sqrt_D_diag * rnorm(p)
    delta <- rnorm(n)
    v <- X %*% u + delta
    w <- solve(inner_mat, y-v)
    theta <- u + t(X) %*% w
    theta
  })
})


# Reporting Times
cat("Part (A) Direct:   ", time_a["elapsed"], "s\n")
```

```
## Part (A) Direct:    0.04 s
```

```
cat("Part (B) Woodbury: ", time_b["elapsed"], "s\n")
```

```
## Part (B) Woodbury:  0.02 s
```

```
cat("Part (C) Algorithm: ", time_c["elapsed"], "s\n")
```

```
## Part (C) Algorithm:  0.07 s
```

Each run returns a different amount of time taken. In theory, part B and C should take the least time. However, it seems that from fastest to slowest is B A C. Of course, it swings wildly, but after running the same chunk of code many many times, this seems to be the correct pattern. In fact, the order in which I generate / time the sampling procedure seems to matter, but if I generate A B C, then B A C is the order I get generally. This is not the case if I generate the samples in a different order. Perhaps it would be more consistent if the sample size was not so small.