Homework 1

Abhisek Banerjee

2026-02-08

Problem 1

a)

We have a matrix $\Sigma = \sigma I + KK'$ where K is an $N \times M$ matrix having $i.i.d$ $\mathcal{N}(0,1)$ random variables in its element. As given in question, we take $\sigma = 0.2$ and fix M at 10.

```r
M = 10
sig = 0.2
```

Looking at my computer condition, I will go with the following grid for N.

```r
ns = c(50, 100, 200, 500, 1000, 2000, 3000, 5000)
```

We set a seed for reproducibility

```r
set.seed(12345678)
```

Two time points storing variable to note the time for the two methods.

```r
t1 = rep(0, length(ns))
t2 = rep(0, length(ns))
```

From slide 25 we get the Sherman-Morrison Woodbury formula as

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

For our problem, $\Sigma = \sigma I + KK^\top$, we note:

$$A = \sigma I, \quad U = K, \quad C = I_M, \quad V = K^\top$$

This gives us:

$$A^{-1} = \frac{1}{\sigma}I, \quad C^{-1} = I_M, \quad VA^{-1}U = \frac{1}{\sigma}K^\top K$$

Substituting into the SMW formula:

$$\Sigma^{-1} = \frac{1}{\sigma}I - \frac{1}{\sigma}I \cdot K\left(I_M + \frac{1}{\sigma}K^\top K\right)^{-1}K^\top \cdot \frac{1}{\sigma}I$$

Simplifying the above we obtain:

$$\Sigma^{-1} = \frac{1}{\sigma}I - \frac{1}{\sigma^2}K\left(I_M + \frac{1}{\sigma}K^\top K\right)^{-1}K^\top$$

Now,we do matrix inversion wuth method 1 and 2:

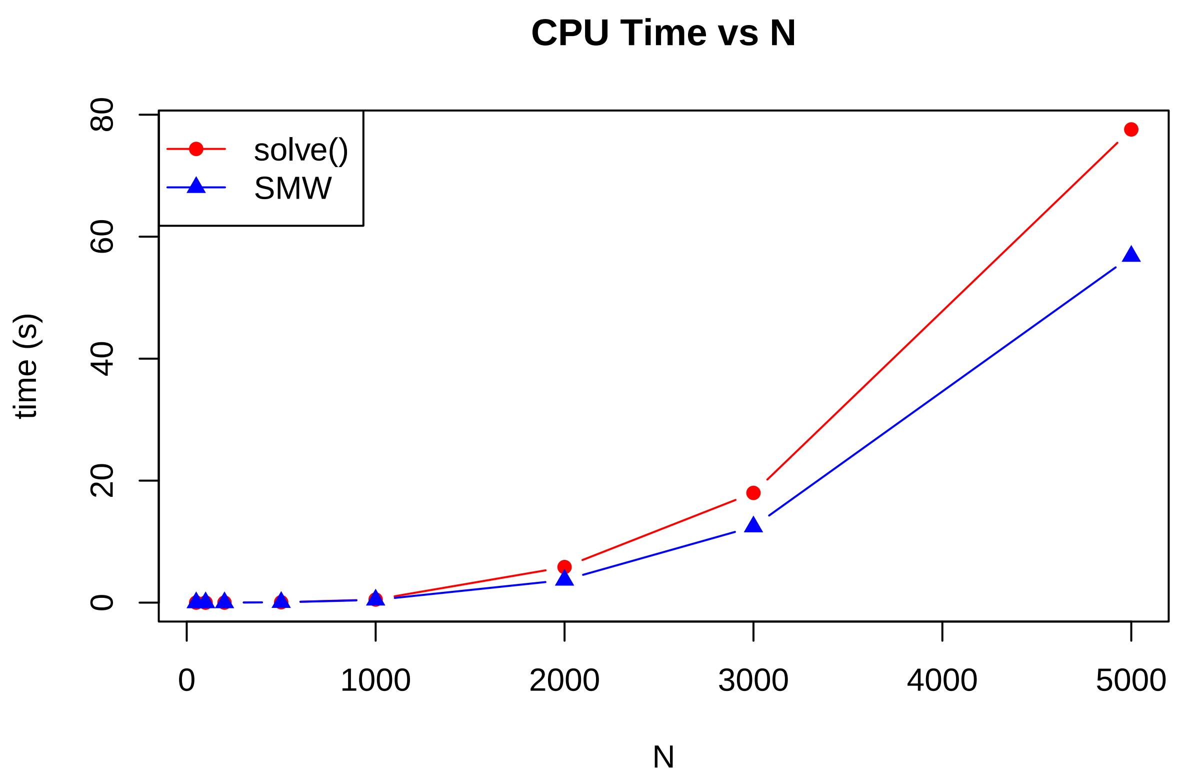
```r
for (i in 1:length(ns)) {
  n = ns[i]
  K = matrix(rnorm(n * M), nrow = n, ncol = M)
  S = sig * diag(n) + K %*% t(K)
  t1[i] = system.time(solve(S))[3]
  t2[i] = system.time({
    Ainv = (1/sig) * diag(n)
    Cinv = diag(M)
    VAU = t(K) %*% Ainv %*% K
    mid = solve(Cinv + VAU)
    ans = Ainv - Ainv %*% K %*% mid %*% t(K) %*% Ainv
  })[3]

  cat("n =", n, "| solve:", t1[i], "| smw:", t2[i], "\n")
}
```

```
## n = 50 | solve: 0.003 | smw: 0.001
## n = 100 | solve: 0.001 | smw: 0.001
## n = 200 | solve: 0.004 | smw: 0.005
## n = 500 | solve: 0.065 | smw: 0.058
## n = 1000 | solve: 0.506 | smw: 0.446
## n = 2000 | solve: 5.83 | smw: 3.703
## n = 3000 | solve: 17.99 | smw: 12.446
## n = 5000 | solve: 77.582 | smw: 56.802
```

Now we make the plot asked in the question:

```r
plot(ns, t1, type = "b", col = "red", pch = 16,
     xlab = "N", ylab = "time (s)",
     main = "CPU Time vs N",
     ylim = range(c(t1, t2)))
lines(ns, t2, type = "b", col = "blue", pch = 17)
legend("topleft", legend = c("solve()", "SMW"),
       col = c("red", "blue"), pch = c(16, 17), lty = 1)
```



b)

From the pdf on FLOPS uploaded in canvas, “flops of different operations” we calculate the flops for the Sherman Woodbury algorithm as follows:

We compute $\Sigma^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$ where $A = \sigma I$, $U = K$, $C = I_M$, $V = K^\top$.

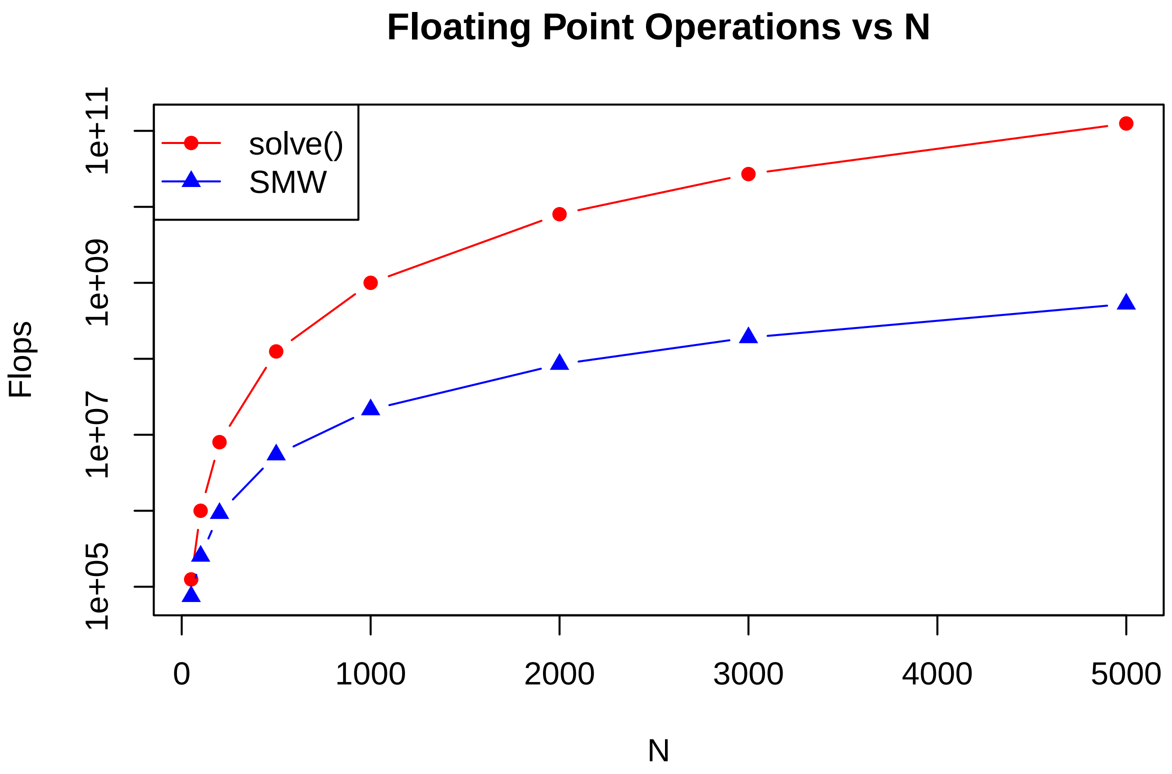
- $A^{-1} = \frac{1}{\sigma}I$: N flops (diagonal inverse)
- $VA^{-1} = K^\top \cdot \frac{1}{\sigma}I$: NM flops (scale each entry)
- $VA^{-1}U = (K^\top)(K)$, $(M \times N)(N \times M)$: $2NM^2$ flops
- $C^{-1} + VA^{-1}U$, $M \times M$ addition: M^2 flops
- Solve $(C^{-1} + VA^{-1}U)^{-1}(VA^{-1})$, i.e. $M \times M$ system applied to $M \times N$ matrix: $M^3 + 2M^2N$ flops
- $A^{-1}U = \frac{1}{\sigma}K$: NM flops
- $(A^{-1}U) \cdot$ mid result, $(N \times M)(M \times N)$: $2N^2M$ flops
- Subtract from A^{-1}: N^2 flops

Total: $N + 2NM + 2NM^2 + M^2 + M^3 + 2M^2N + 2N^2M + N^2$. With $M = 10$ fixed, the dominant term is $2N^2M = 20N^2$, so the cost is $O(N^2)$.

For the the algorithm with `solve()` function, we solve $N \times N$ giving us N^3. The following code in R gives the plot:

```r
f1 = ns^3
f2 = ns + 2*ns*M + 2*ns*M^2 + M^2 + M^3 + 2*M^2*ns + 2*ns^2*M + ns^2
plot(ns, f1, type = "b", col = "red", pch = 16, log = "y",
     xlab = "N", ylab = "Flops",
     main = "Floating Point Operations vs N",
     ylim = range(c(f1, f2)))
lines(ns, f2, type = "b", col = "blue", pch = 17)
```

```r
legend("topleft", legend = c("solve()", "SMW"),
       col = c("red", "blue"), pch = c(16, 17), lty = 1)
```



1c

Looking at the two plots, it seems like, for small N, for example untill 2000 the difference in computation time by the two methods is nit that significant. However as N increases, the difference in computation time increases for both and the difference in time between the two algorithms starts to increase more and more. A similar observation can be seen for the second graph as well, increasing N leads to a steady increase in floating point operations in both, however for the solve function, just like computation time, the floating point operations are way more than the SMW algorithm. This shows that algebric manipulation can significantly reduce computation time and can beat already optimized inbuilt codes in programming languages.