Fast matrix calculations

Stephen Coleman

2022-05-26

Rich structure

GPs with the squared exponential kernel and consistent timepoints across observations have very rich structure in their covariance functions. We can use this to reduce the number of calculations we have to do.

Let $X = (X_1, \ldots, X_N)$ be our exchangeable, observed data with each $X_n = (X_{n,1}, \ldots, X_{n,P})$ having the same P time points/locations. We assume:

$$X_{n,p} = \mu_p + \epsilon_i, X_n \sim \mathcal{N}(\mu, \sigma^2 I), \mu \sim GP(0, C(\cdot, \cdot)), A(p, p') = a^2 \exp\left(-\frac{(p-p')^2}{l}\right).$$

Given the data, X, the covariance function of the GP is a $(NP \times NP)$ block matrix composed of N^2 $P \times P$ subblocks, A.

We are interested in two matrix calculations, $\Xi = C(\{1, \ldots, P\}, \cdot) \times (\sigma^2 I + C)^{-1}$, and $\Xi \times C(\cdot, \{1, \ldots, P\})$, where $A(\cdot, \{i, j, k\})$ is the i, j and kth columns of the matrix A. Let $\tau = \{1, \ldots, P\}$ for conciseness.

There already exists a fast calculation of the inverse, $(\sigma^2 I + C)^{-1}$ which I will state here, so we do not need to consider this problem.

$$Z = (I + \sigma^{-2} N A)^{-1}, (\sigma^2 I + C)^{-1} = \sigma^{-2}\left(I - n^{-1} J \otimes (I - Z)\right).$$

For ease of notation, let $B = \sigma^{-2} n^{-1} J \otimes (I - Z)$ and $C_{inv} = (\sigma^2 I + C)^{-1}$.

It is important to note that both C and C_{inv} have block structure. We aim to use this to avoid N identical calculations and effectively consider only $P \times P$ matrices in our calculations.

$$\Xi(\tau, kP + \tau) = \frac{1}{\sigma^2}(A - A \times B),$$

for any $k \in \{0, \ldots, N\}$, by which I mean each $P \times P$ block of Ξ has this form.

Then, the second product is simply:

$$\Xi \times C(\cdot, \tau) = N\, \Xi(\tau, \tau) \times C(\tau, \tau).$$

```r
approxEqual <- function(A, B, tol = 1e-10) {
  all(((A - B)**2) < tol)
}

squaredExponentialFunction <- function(amplitude, length, delta){
  amplitude * exp(- delta / length)
}
```

```r
kernelSubBlock <- function(P, amplitude, length) {
  A <- matrix(0, P, P)
  for(ii in seq(1, P)) {
    for(jj in seq(1, ii)) {
      delta <- ii - jj
      x <- squaredExponentialFunction(amplitude, length, delta)
      A[ii, jj] <- A[jj, ii] <- x
    }
  }
  A
}

smallerInversion <- function(sigma, subblock) {
  Q <- I_D + sigma**(-2) * n * subblock
  Z <- solve(Q)
  Z
}

fastInversion <- function(sigma, subblock) {
  Z <- smallerInversion(sigma, subblock)
  sigma**(-2) * (I_nD - (1 / n) * kronecker(J, I_D - Z))
}

my_product <- function(sigma, subblock) {
  Z <- smallerInversion(sigma, subblock)
  B <- I_D - Z
  output <- subblock - subblock %*% B
  sigma**(-2) * output
}

n <- 3
D <- 30

D_inds <- seq(1, D)
ND_inds <- seq(1, n * D)

I_n <- diag(1, nrow = n, ncol = n)
I_D <- diag(1, nrow = D, ncol = D)
I_nD <- diag(1, nrow = n * D, ncol = n * D)

J <- matrix(1, n, n)

sigma <- 0.85
ampltiude <- 1.25
length <- 1.6667

A <- kernelSubBlock(D, ampltiude, length)
C_part_1 <- cbind(A, A, A)
C <- rbind(C_part_1, C_part_1, C_part_1)
Q <- (I_D + sigma**(-2) * n * A)
Z <- solve(Q)

C_inv <- fastInversion(sigma, A)
```

```r
C_inv_naive <- solve(sigma**2 * I_nD + C)

approxEqual(C_inv, C_inv_naive)
```

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

```r
reduced_mat <- my_product(sigma, A)
naive_answer <- C[1:D, ] %*% C_inv
my_answer <- cbind(reduced_mat, reduced_mat, reduced_mat)

approxEqual(naive_answer, my_answer)
```

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

```r
final_answer_1 <- n * (my_answer[1:D, 1:D] %*% C[1:D, 1:D])
final_answer_2 <- n * (reduced_mat %*% C[1:D, 1:D])
naive_answer <- naive_answer %*% C[, 1:D]

approxEqual(final_answer_2, naive_answer)
```

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