Spatial MLE: Gradient-Based and Gradient-Free Estimation

Contents


```
suppressPackageStartupMessages(library(spam))
```

1 Spatial dataset

We place $n = 500$ locations uniformly at random in $[0,1]^2$ and build a sparse distance matrix using `nearest.kdtree()` with neighbourhood radius $\delta = 0.33$. The covariance model is spherical with $\theta = (\text{range}, \text{sill}, \text{nugget}) = (0.3,\ 1.5,\ 0.1)$.

```
set.seed(42)
n     <- 500
locs  <- matrix(runif(2 * n), n, 2)
delta <- 0.35

h <- nearest.kdtree(locs, delta = delta, upper = NULL)
cat("Sparse distance matrix:", n, "x", n, " nnz =", nnz(h), "\n")
#> Sparse distance matrix: 500 x 500  nnz = 68464

theta.true <- c(0.3, 1.5, 0.1)       # range, sill, nugget
Sigma      <- cov.sph(h, theta.true)
Y          <- rmvnorm.spam(10, Sigma = Sigma)
y          <- Y[1L, ]
```

```
pal <- colorRampPalette(c("#4575b4", "#ffffbf", "#d73027"))(64)
col <- pal[cut(y, 64)]
plot(locs, pch = 19, col = col, cex = 0.9,
     xlab = "s1", ylab = "s2", main = "Simulated spatial data")
```

2 Estimation

We compare several approaches differing in solver, parametrization, gradient type, preconditioner, and optimizer. Natural-scale fits use box constraints; log-scale fits replace the box with a log link via `reparametrise()` and run unconstrained. A `ctrl1` and a `ctrl2` list control the L-BFGS-B and BFGS convergence criteria.

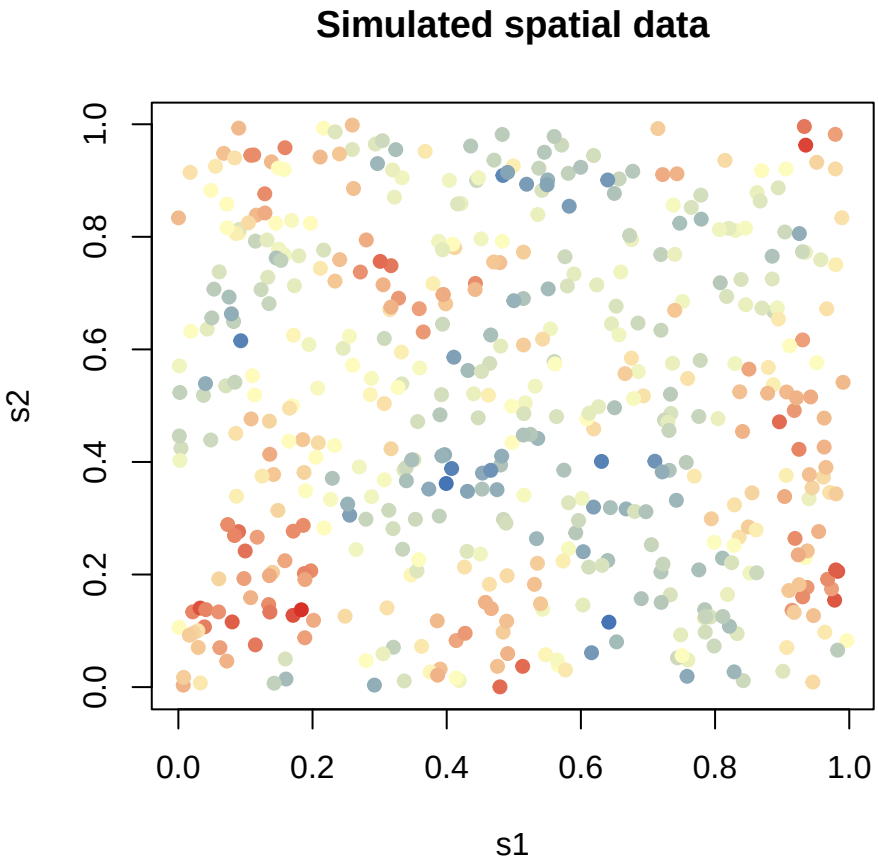

Figure 1: Simulated observations; colour encodes the response value.

```
theta0     <- theta.true
thetalower <- c(0.01, 0.01, 0)      # lower bounds on natural scale
thetaupper <- c(delta, Inf, Inf)    # range capped at neighbourhood radius

ctrl1 <- list(factr = 1e4,    # tighter than default 1e7
              pgtol = 1e-3)  # matches somewhat factr
ctrl2 <- list(reltol= 1e-4)  # matches somewhat factr

## log-scale wrappers: phi = log(theta), theta = exp(phi)
link <- function(phi) {
      p <- plogis(phi[1])
      c(delta * p, exp(phi[2]), exp(phi[3]))
    }
dlink <- function(phi) {
      p <- plogis(phi[1])
      c(delta * p * (1 - p), exp(phi[2]), exp(phi[3]))
    }
sph_log <- reparametrise(cov.sph, grad.cov.sph,
                         link=link, grad.link= dlink)
phi0 <- c(qlogis(theta0[1] / delta), log(theta0[2:3]))  # inverse transform

mleout <- mle.spam(y, h, cov.sph, theta0,
                         thetalower = thetalower,
                         thetaupper = thetaupper,
                         control    = ctrl1, loginfo=TRUE)
```

```
run_all <- function(yi) {
  list(
    num       = mle.spam(yi, h, cov.sph, theta0,
                         thetalower = thetalower,
                         thetaupper = thetaupper,
                         control    = ctrl1),
    log.num   = mle.spam(yi, h, sph_log$Covariance, phi0,
                         thetalower = rep(-Inf, 3),
                         thetaupper = rep( Inf, 3),
                         control    = ctrl1),
    grad      = mle.spam(yi, h, cov.sph, theta0,
                         thetalower     = thetalower,
                         thetaupper     = thetaupper,
                         gradCovariance = grad.cov.sph,
                         control        = ctrl1),
    bfgs.num  = mle.spam(yi, h, sph_log$Covariance, phi0,
                         thetalower = rep(-Inf, 3),
                         thetaupper = rep( Inf, 3),
                         method     = "BFGS",
                         control    = ctrl2),
    bfgs.grad = mle.spam(yi, h, sph_log$Covariance, phi0,
                         thetalower     = rep(-Inf, 3),
                         thetaupper     = rep( Inf, 3),
                         gradCovariance = sph_log$gradCovariance,
                         method         = "BFGS",
                         control        = ctrl2),
    cg.num    = mle.spam(yi, h, cov.sph, theta0,
                         thetalower   = thetalower,
```

```
                        thetaupper   = thetaupper,
                        solver       = "cg",
                        iter.control = list(nprobe = 30, seed = 1),
                        control      = ctrl1),
    cg.grad   = mle.spam(yi, h, cov.sph, theta0,
                        thetalower     = thetalower,
                        thetaupper     = thetaupper,
                        gradCovariance = grad.cov.sph,
                        solver         = "cg",
                        iter.control   = list(nprobe = 30, seed = 1),
                        control        = ctrl1),
    cg.prec   = mle.spam(yi, h, cov.sph, theta0,
                        thetalower     = thetalower,
                        thetaupper     = thetaupper,
                        gradCovariance = grad.cov.sph,
                        solver         = "cg",
                        iter.control   = list(nprobe = 30, seed = 1,
                                              precond='ssor', omega = 0.25),
                        control        = ctrl1)
  )
}

all_fits <- lapply(seq_len(nrow(Y)), function(i) run_all(Y[i, ]))
```

3 Comparison

Log-scale estimates are back-transformed via $\hat{\theta} = e^{\hat{\phi}}$ for comparison with natural-scale fits.

```
## helper: mean estimates + mean call counts over all realisations
mrow <- function(key, log.scale = FALSE) {
  th <- sapply(all_fits, function(s) if (log.scale) exp(s[[key]]$theta) else s[[key]]$theta)
  fn <- sapply(all_fits, function(s) s[[key]]$counts[1])
  gr <- sapply(all_fits, function(s) s[[key]]$counts[2])
  c(rowMeans(th), mean(fn), mean(gr))
}
res <- rbind(
  "truth"                        = c(theta.true,                   NA,                        NA),
  "chol / num  / L-BFGS-B"      = mrow("num"),
  "chol / num  / L-BFGS-B / log" = mrow("log.num",  log.scale = TRUE),
  "chol / anal / L-BFGS-B"      = mrow("grad"),
  "chol / num  / BFGS / log"     = mrow("bfgs.num",  log.scale = TRUE),
  "chol / anal / BFGS / log"     = mrow("bfgs.grad", log.scale = TRUE),
  "CG   / num  / L-BFGS-B"      = mrow("cg.num"),
  "CG   / anal / L-BFGS-B"      = mrow("cg.grad"),
  "CG   / anal / L-BFGS-B prec" = mrow("cg.prec")
)
colnames(res) <- c("range", "sill", "nugget", "fn evals", "gr evals")
knitr::kable(round(res, 4), caption = paste("Mean MLE estimates and optimiser call counts over",
                                            nrow(Y), "realisations"))
```

Table 1: Mean MLE estimates and optimiser call counts over 10 realisations

	range	sill	nugget	fn evals	gr evals
truth	0.3000	1.5000	0.1000	NA	NA
chol / num / L-BFGS-B	0.3027	1.4863	0.1112	25.8	25.8
chol / num / L-BFGS-B / log	6.5987	1.4864	0.1112	10.6	10.6
chol / anal / L-BFGS-B	0.3027	1.4864	0.1112	16.1	16.1
chol / num / BFGS / log	6.5360	1.4666	0.1126	25.5	4.1
chol / anal / BFGS / log	6.5360	1.4666	0.1126	25.0	4.1
CG / num / L-BFGS-B	0.3032	1.5190	0.1069	70.0	70.0
CG / anal / L-BFGS-B	0.3031	1.5186	0.1069	68.8	68.8
CG / anal / L-BFGS-B prec	0.3031	1.5186	0.1069	68.8	68.8

```
keys      <- c("num","log.num","grad","bfgs.num","bfgs.grad","cg.num","cg.grad","cg.prec")
log.keys  <- c("log.num","bfgs.num","bfgs.grad")
xlabs     <- c("ch/n/L","ch/n/L/lg","ch/a/L","ch/n/B/lg","ch/a/B/lg","CG/n","CG/a","CG/a+p")
metrics   <- c("range","sill","nugget","neg2ll","fn_evals","gr_evals")
truth_val <- c(theta.true, NA, NA, NA)

## build [method x metric x realisation] array
res_arr <- array(NA, dim = c(length(keys), 6, nrow(Y)))
for (i in seq_len(nrow(Y)))
  for (j in seq_along(keys)) {
    f  <- all_fits[[i]][[keys[j]]]
    th <- if (keys[j] %in% log.keys) exp(f$theta) else f$theta
    res_arr[j,, i] <- c(th, f$value, f$counts[1], f$counts[2])
  }

par(mfrow = c(2, 3), mar = c(5, 4, 2, 1))
for (j in seq_along(metrics)) {
  boxplot(t(res_arr[, j, ]), names = xlabs, main = metrics[j],
          col = "#91bfdb", las = 2, cex.axis = 0.75)
  if (!is.na(truth_val[j]))
    abline(h = truth_val[j], col = "red", lwd = 2)
}
```

All L-BFGS_B approaches recover the true parameters reasonably well. BFGS struggles, if the range is too close to delta. is consistent, BFGS slightly more off. The natural-scale and log-scale L-BFGS-B fits (rows 1–2) show matching function evaluation counts: when the optimum is interior to the box the projected gradient equals the full gradient, so the two parametrisations traverse the same steps. The analytical gradient reduces function evaluations (each gradient evaluation replaces $p + 1$ finite-difference calls, $p = \text{length}(\theta) = 3$). The CG solver is attractive for large n where Cholesky becomes the bottleneck.

4 Effect of convergence tolerances

The L-BFGS-B stopping criteria `factr` (relative function decrease) and `pgtol` (projected gradient norm) control when the optimiser terminates. Tighter tolerances increase the number of iterations but have diminishing returns once the tolerance is below the noise floor of the objective. Here we vary both simultaneously using the Cholesky solver with numerical gradient.

```
factr_vals <- c(1e6,5e6,1e7, 5e7, 1e8, 5e8, 1e9, 5e9, 1e10, 5e10, 1e11, 5e11, 1e12)   # 1e7 is the R de
pgtol_vals <- c(0, 1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 5e-1, 1e0, 5e0) # This default
```

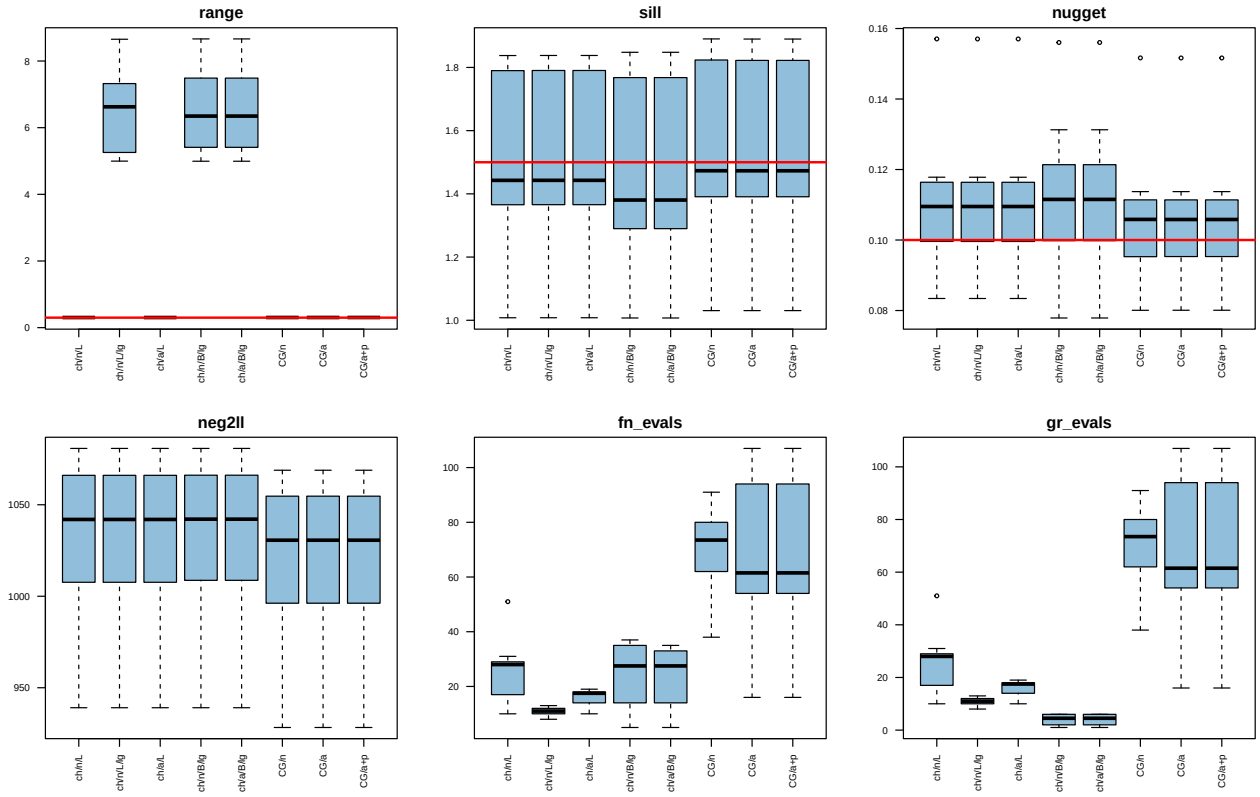

Figure 2: Distribution of estimates, neg2ll, and iteration counts across realisations; red lines = truth.

```
grid <- data.frame(factr = factr_vals, pgtol = pgtol_vals)

tol_fits <- lapply(seq_len(nrow(grid)), function(i) {
  mle.spam(y, h, cov.sph, theta0,
           thetalower = thetalower,
           thetaupper = thetaupper,
           control    = list(factr = grid$factr[i],
                             pgtol = grid$pgtol[i]))
})

tol_res <- do.call(rbind, lapply(seq_len(nrow(grid)), function(i) {
  fit <- tol_fits[[i]]
  c(factr     = grid$factr[i],
    pgtol     = grid$pgtol[i],
    range     = fit$theta[1],
    sill      = fit$theta[2],
    nugget    = fit$theta[3],
    norm      = norm(fit$theta - theta.true),
    deltaN2LL = fit$value,
    fn_evals  = unname(fit$counts[1]),
    gr_evals  = unname(fit$counts[2]))
}))
tol_res[,"deltaN2LL"] <- tol_res[,"deltaN2LL"]- tol_res[1,"deltaN2LL"]
```

```
knitr::kable(round(as.data.frame(tol_res), 4),
             caption = paste("Effect of factr and pgtol on chol/num.grad estimates.",
                             "True theta =", paste(theta.true, collapse = ", ")))
```

Table 2: Effect of factr and pgtol on chol/num.grad estimates. True theta = 0.3, 1.5, 0.1

factr	pgtol	range	sill	nugget	norm	deltaN2LL	fn_evals	gr_evals
1e+06	0e+00	0.3005	1.5296	0.1164	0.0465	0.0000	17	17
5e+06	0e+00	0.3005	1.5296	0.1164	0.0465	0.0000	17	17
1e+07	0e+00	0.3005	1.5296	0.1164	0.0465	0.0000	17	17
5e+07	1e-04	0.3005	1.5296	0.1164	0.0465	0.0000	17	17
1e+08	5e-04	0.3005	1.5297	0.1164	0.0465	0.0000	16	16
5e+08	1e-03	0.3001	1.5023	0.1191	0.0215	0.0149	7	7
1e+09	5e-03	0.3001	1.5023	0.1191	0.0215	0.0149	7	7
5e+09	1e-02	0.3001	1.5022	0.1191	0.0215	0.0149	6	6
1e+10	5e-02	0.3004	1.5022	0.1193	0.0220	0.0159	5	5
5e+10	1e-01	0.3004	1.5022	0.1193	0.0220	0.0159	5	5
1e+11	5e-01	0.3004	1.5022	0.1193	0.0220	0.0159	5	5
5e+11	1e+00	0.3000	1.5022	0.1199	0.0221	0.0176	4	4
1e+12	5e+00	0.3000	1.5022	0.1199	0.0221	0.0176	4	4

Looser tolerances converge in fewer function evaluations with estimates that are practically indistinguishable from those at tight tolerances, provided the tolerance stays well above numerical noise. For the CG solver the objective itself has noise of order $O(s^{-1/2})$ (where s = nprobe), so `factr = 1e2`, `pgtol = 1e-1` is the best choice here. In general, we presume that the values should be similar.

5 Effect of the number of probe vectors

The stochastic log-determinant has variance decreasing with `nprobe`. We compare $s \in \{5, 30, 100\}$ over `nrep = 10` independent data realisations; the optimiser settings (`ctrl1`) are held fixed.

```
nrep  <- 10
seeds <- 1:nrep

run_nprobe <- function(seed_data, nprobe) {
  set.seed(seed_data)
  yi  <- c(rmvnorm.spam(1, Sigma = Sigma))
  fit <- mle.spam(yi, h, cov.sph, theta0,
                  thetalower   = thetalower,
                  thetaupper   = thetaupper,
                  solver       = "cg",
                  iter.control = list(nprobe = nprobe, seed = 1),
                  control      = ctrl1)
  fit$theta
}
```

```
res5   <- t(sapply(seeds, run_nprobe, nprobe =   5))
res30  <- t(sapply(seeds, run_nprobe, nprobe =  30))
res100 <- t(sapply(seeds, run_nprobe, nprobe = 100))
```

```r
make_row <- function(mat, s) {
  c(nprobe = s,
    range_mean = mean(mat[, 1]), range_sd = sd(mat[, 1]),
    sill_mean  = mean(mat[, 2]), sill_sd  = sd(mat[, 2]),
    nugget_mean= mean(mat[, 3]), nugget_sd= sd(mat[, 3]))
}
nprobe_res <- rbind(make_row(res5,   5),
                    make_row(res30,  30),
                    make_row(res100, 100))
knitr::kable(round(as.data.frame(nprobe_res), 4),
             caption = paste("Mean and SD of CG/num.grad estimates over", nrep,
                             "data realisations by nprobe"))
```

Table 3: Mean and SD of CG/num.grad estimates over 10 data realisations by nprobe

nprobe	range_mean	range_sd	sill_mean	sill_sd	nugget_mean	nugget_sd
5	0.2936	0.0217	1.3671	0.265	0.1171	0.0341
30	0.2996	0.0209	1.4818	0.272	0.1042	0.0320
100	0.3011	0.0205	1.4470	0.273	0.1103	0.0323

```r
par(mfrow = c(1, 3))
nms <- c("range", "sill", "nugget")
for (j in seq_along(nms)) {
  boxplot(list("s=5"   = res5[, j],
               "s=30"  = res30[, j],
               "s=100" = res100[, j]),
          main = nms[j], ylab = nms[j],
          col  = c("#91bfdb", "#fee090", "#fc8d59"))
  abline(h = theta.true[j], col = "red", lwd = 2)
}
```

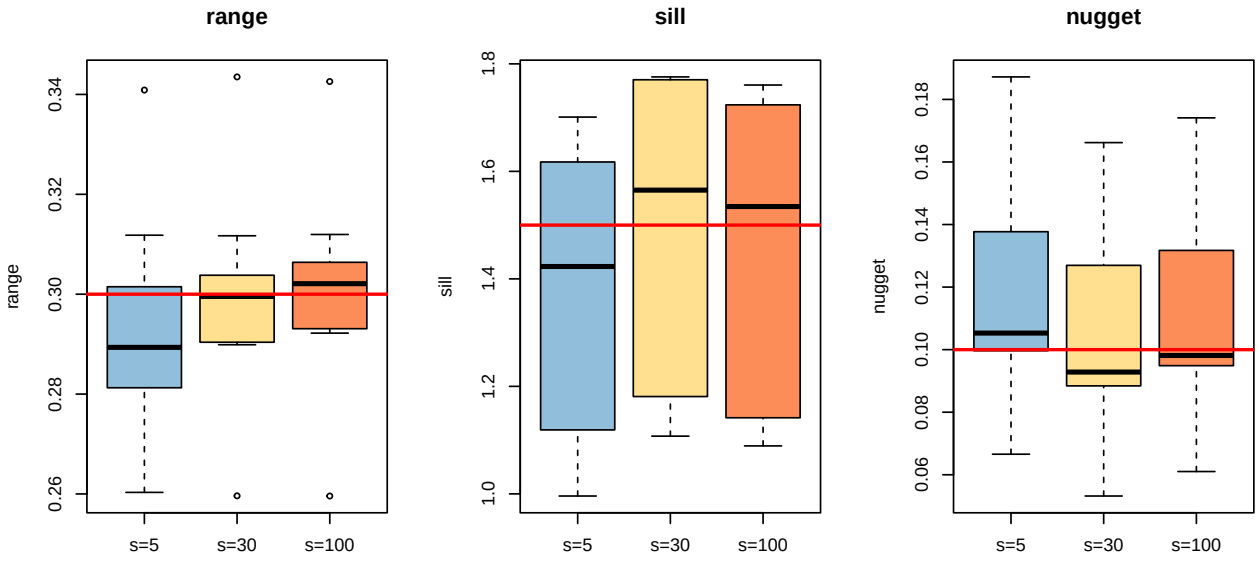

Figure 3: Distribution of estimates by nprobe; red line = truth.

The standard deviation of the estimates does not decrease as `nprobe` increases. The reason is that the replications vary the *data* (different simulated realisations of y), not the probe vectors (the probe seed is fixed at 1 throughout). The spread in estimates therefore reflects the **statistical variance of the MLE** across datasets, which is governed by the Fisher information and is independent of `nprobe`. Increasing `nprobe` reduces the approximation error of the stochastic log-determinant for a *fixed* dataset; it has no effect on the dataset-to-dataset variability of the estimator.

6 Scaling to n = 20000

For large n the Cholesky factorisation dominates. We generate a fresh dataset with $n = 2000$, approximately m datapoints within the range, and compare three approaches: exact Cholesky with numerical gradient, CG with numerical gradient, and CG with analytical gradient, both with the SSOR preconditioner ($\omega = 0.25$).

```
set.seed(7)
n2     <- 20000
m2     <- 100
locs2  <- matrix(runif(2 * n2), n2, 2)
delta2 <- sqrt(m2 / (n2 * pi))
theta.true2 <- c(signif(delta2*.9, 2), theta.true[-1] )

h2     <- nearest.kdtree(locs2, delta = delta2)
h2 <- h2 + t(h2)

cat("Large distance matrix:", n2, "x", n2, " nnz =", nnz(h2), " density =",
    round(nnz(h2)/n2^2*100,2),"%\n")
#> Large distance matrix: 20000 x 20000  nnz = 1948398  density = 0.49 %

Sigma2 <- cov.sph(h2, theta.true2)
y2     <- c(rmvnorm.spam(1, Sigma = Sigma2))
```

```
theta0.2     <- theta.true2 * 0.9
thetaupper2 <- c(delta2, Inf, Inf)
ctrl <- list(factr = 5e4, pgtol = 5e-2)
ic <- list(nprobe = 30, seed = 1, tol = 1e-3, maxiter=250)

t2.chol.num <- system.time(
  fit2.chol.num <- mle.spam(y2, h2, cov.sph, theta0.2,
                            thetalower = thetalower,
                            thetaupper = thetaupper2,
                            control    = ctrl)
)


t2.cg.prec <- system.time(
  fit2.cg.prec <- mle.spam(y2, h2, cov.sph, theta0.2,
                           thetalower     = thetalower,
                           thetaupper     = thetaupper2,
                           gradCovariance = grad.cov.sph,
                           solver         = "cg",
                           iter.control   = ic,
                           control        = ctrl)
)

t2.cg.grad <- system.time(
```

```
  fit2.cg.grad <- mle.spam(y2, h2, cov.sph, theta0.2,
                           thetalower   = thetalower,
                           thetaupper   = thetaupper2,
                           gradCovariance = grad.cov.sph,
                            solver       = "cg",
                           iter.control = list( ic, precond='ssor', omega= 0.3),
                            control      = ctrl)
)
```

```
res2 <- rbind(
  "truth"                  = c(theta.true2,  rep(NA,3)),
  "chol / num  / L-BFGS-B" = c(fit2.chol.num$theta, fit2.chol.num$value,  fit2.chol.num$counts[1], fit2
  "CG   / prec / L-BFGS-B" = c(fit2.cg.prec$theta,   fit2.cg.prec$value, fit2.cg.prec$counts[1],   fit2
  "CG   / anal / L-BFGS-B" = c(fit2.cg.grad$theta,  fit2.cg.grad$value, fit2.cg.grad$counts[1],  fit2.cg
)
colnames(res2) <- c("range", "sill", "nugget", "neg2ll", "fn evals", "gr evals")
knitr::kable(round(res2, 4), caption = paste("MLE estimates for n =",n2 ))
```

Table 4: MLE estimates for n = 20000

	range	sill	nugget	neg2ll	fn evals	gr evals
truth	0.0360	1.5000	0.1000	NA	NA	NA
chol / num / L-BFGS-B	0.0363	1.4786	0.1047	44073.35	43	43
CG / prec / L-BFGS-B	0.0363	1.4735	0.1056	44118.45	74	74
CG / anal / L-BFGS-B	0.0363	1.4735	0.1056	44118.45	74	74

```
times2 <- rbind(
  "chol / num"  = c(t2.chol.num["elapsed"],perIteration=unname(t2.chol.num["elapsed"]/fit2.chol.num$cou
  "CG   / prec"  = c( t2.cg.prec["elapsed"], t2.cg.prec["elapsed"]/fit2.cg.prec$counts[1]),
  "CG   / anal" = c(t2.cg.grad["elapsed"],t2.cg.grad["elapsed"]/fit2.cg.grad$counts[1])
)
knitr::kable(data.frame(round(times2, 2)),
             caption = paste("Elapsed time (seconds) for n =",n2))
```

Table 5: Elapsed time (seconds) for n = 20000

	elapsed	perIteration
chol / num	216.58	5.04
CG / prec	540.46	7.30
CG / anal	544.19	7.35

At $n = 20000$ the Cholesky factorisation is still the fastest one. However, one can observe cases where the iterative approach outperforms direct ones with $n = 10000$ only. See other RMarkdown illustrations where we further increase n, until at each Cholesky objective evaluation becomes more expensive than the iterative CG solve. The CG solver with an analytical gradient further reduces the number of objective evaluations needed, combining the benefits of a cheap per-iteration cost and fewer iterations overall.