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 <- 0.33

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 = 61920

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

```
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")
```

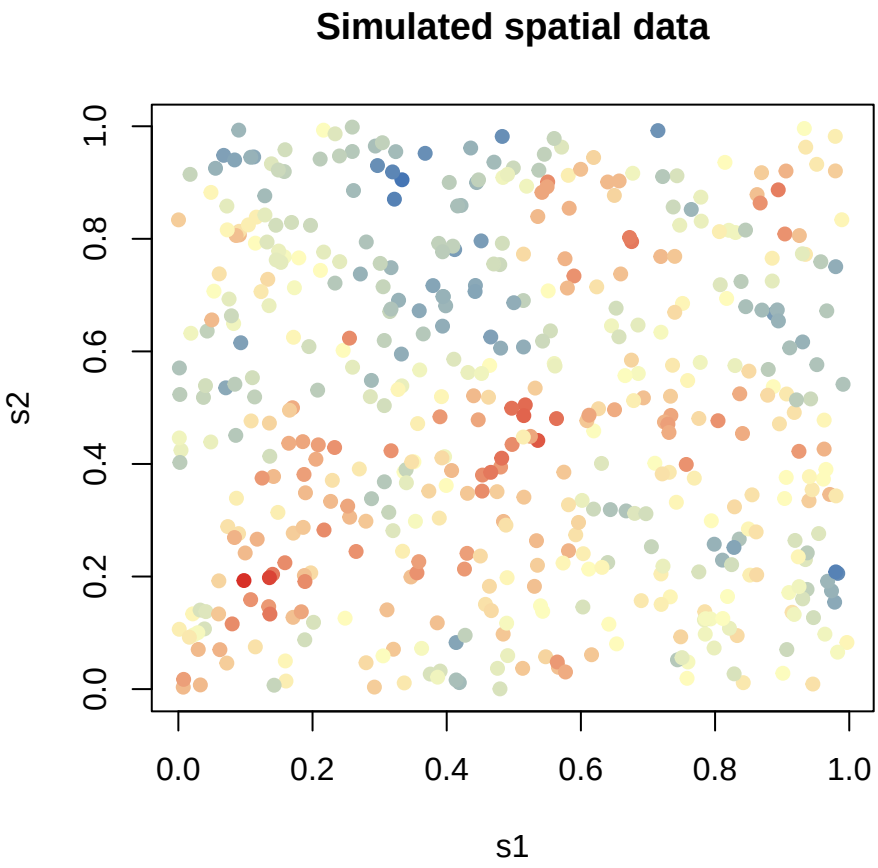

Figure 1: Simulated observations; colour encodes the response value.

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.

```
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)
sph_log <- reparametrise(cov.sph, grad.cov.sph)
phi0    <- log(theta0)
```

2.1 Cholesky — numerical gradient (L-BFGS-B, natural scale)

`optim` approximates the gradient by finite differences at each step.

```
t.num <- system.time(
  fit.num <- mle.spam(y, h, cov.sph, theta0,
                      thetalower = thetalower,
                      thetaupper = thetaupper,
                      control    = ctrl1)
)
```

2.2 Cholesky — numerical gradient (L-BFGS-B, log scale)

The same fit on the log scale via `reparametrise()`. With the true optimum interior to the box the projected gradient equals the full gradient, so L-BFGS-B behaves identically to unconstrained optimisation; the function evaluation count should match the natural-scale fit.

```
t.log.num <- system.time(
  fit.log.num <- mle.spam(y, h, sph_log$Covariance, phi0,
                          thetalower = rep(-Inf, 3),
                          thetaupper = rep( Inf, 3),
                          control    = ctrl1)
)
```

2.3 Cholesky — analytical gradient (L-BFGS-B, natural scale)

`grad.cov.sph` supplies exact derivatives $\partial\Sigma/\partial\theta_k$; `gradneg2loglik` computes the gradient via n Cholesky back-solves (one per row of each derivative matrix).

```
t.grad <- system.time(
  fit.grad <- mle.spam(y, h, cov.sph, theta0,
                       thetalower     = thetalower,
                       thetaupper     = thetaupper,
                       gradCovariance = grad.cov.sph,
                       control        = ctrl1)
)
```

2.4 Cholesky — numerical gradient (BFGS, log scale)

BFGS does not support box constraints. The log parametrisation removes the need for them: all $\phi_k \in \mathbb{R}$ automatically give $\theta_k = e^{\phi_k} > 0$.

```
t.bfgs.num <- system.time(
  fit.bfgs.num <- mle.spam(y, h, sph_log$Covariance, phi0,
                           thetalower = rep(-Inf, 3),
                           thetaupper = rep( Inf, 3),
                           method     = "BFGS",
                       control        = ctrl2)
)
#> Warning in (function (object, x, ...) : Singularity problem when updating a Cholesky Factor.
#> 'object' not updated.
#> Warning in (function (object, x, ...) : Singularity problem when updating a Cholesky Factor.
#> 'object' not updated.
```

2.5 Cholesky — analytical gradient (BFGS, log scale)

The chain-rule correction $\partial/\partial\phi_k = e^{\phi_k}\,\partial/\partial\theta_k$ is applied automatically by the `reparametrise()` wrapper.

```
t.bfgs.grad <- system.time(
  fit.bfgs.grad <- mle.spam(y, h, sph_log$Covariance, phi0,
                            thetalower     = rep(-Inf, 3),
                            thetaupper     = rep( Inf, 3),
                            gradCovariance = sph_log$gradCovariance,
                            method         = "BFGS",
                       control        = ctrl2)
)
#> Warning in (function (object, x, ...) : Singularity problem when updating a Cholesky Factor.
#> 'object' not updated.
#> Warning in (function (object, x, ...) : Singularity problem when updating a Cholesky Factor.
#> 'object' not updated.
```

2.6 CG — stochastic log-determinant, numerical gradient

The log-determinant is estimated via stochastic Lanczos quadrature with `nprobe = 30` Rademacher probe vectors fixed before `optim` begins (deterministic objective).

```
t.cg.num <- system.time(
  fit.cg.num <- mle.spam(y, h, cov.sph, theta0,
                         thetalower   = thetalower,
                         thetaupper   = thetaupper,
                         solver       = "cg",
                         iter.control = list(nprobe = 30, seed = 1),
                         control      = ctrl1)
)
```

2.7 CG — stochastic log-determinant, analytical gradient

The Hutchinson estimator approximates the trace term in the gradient using the same probe vectors as the objective — both then optimise a consistent stochastic approximation.

```
t.cg.grad <- system.time(
  fit.cg.grad <- mle.spam(y, h, cov.sph, theta0,
                          thetalower     = thetalower,
```

```
                          thetaupper     = thetaupper,
                          gradCovariance = grad.cov.sph,
                          solver         = "cg",
                          iter.control   = list(nprobe = 30, seed = 1),
                          control        = ctrl1)
)
```

2.8 CG — stochastic log-determinant, analytical gradient, preconditioner

As before, but with the SSOR preconditioner ($\omega = 0.25$).

```
t.cg.prec <- system.time(
  fit.cg.prec <- mle.spam(y, 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)
)
```

3 Comparison

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

```
res <- rbind(
  "truth"                        = c(theta.true,                  NA,                        NA),
  "chol / num  / L-BFGS-B"       = c(fit.num$theta,               fit.num$counts[1],         fit.num$counts
  "chol / num  / L-BFGS-B / log" = c(exp(fit.log.num$theta),  fit.log.num$counts[1],     fit.log.num$cou
  "chol / anal / L-BFGS-B"       = c(fit.grad$theta,              fit.grad$counts[1],        fit.grad$counts
  "chol / num  / BFGS / log"     = c(exp(fit.bfgs.num$theta), fit.bfgs.num$counts[1],    fit.bfgs.num$co
  "chol / anal / BFGS / log"     = c(exp(fit.bfgs.grad$theta),fit.bfgs.grad$counts[1],   fit.bfgs.grad$
  "CG   / num  / L-BFGS-B"       = c(fit.cg.num$theta,        fit.cg.num$counts[1],      fit.cg.num$coun
  "CG   / anal / L-BFGS-B"       = c(fit.cg.grad$theta,       fit.cg.grad$counts[1],     fit.cg.grad$cou
  "CG   / anal / L-BFGS-B prec" = c(fit.cg.prec$theta,       fit.cg.prec$counts[1],     fit.cg.prec$cou
)
colnames(res) <- c("range", "sill", "nugget", "fn evals", "gr evals")
knitr::kable(round(res, 4), caption = "MLE estimates and optimiser call counts")
```

Table 1: MLE estimates and optimiser call counts

	range	sill	nugget	fn evals	gr evals
truth	0.3000	1.5000	0.1000	NA	NA
chol / num / L-BFGS-B	0.2963	1.6727	0.0929	19	19
chol / num / L-BFGS-B / log	0.2963	1.6727	0.0929	14	14
chol / anal / L-BFGS-B	0.2963	1.6726	0.0929	19	19
chol / num / BFGS / log	0.2963	1.6726	0.0929	35	5
chol / anal / BFGS / log	0.2963	1.6726	0.0929	34	5
CG / num / L-BFGS-B	0.2963	1.7033	0.0887	82	82
CG / anal / L-BFGS-B	0.2963	1.7033	0.0887	64	64
CG / anal / L-BFGS-B prec	0.2963	1.7033	0.0887	64	64

```r
times <- c(
  "chol / num  / L-BFGS-B"       = t.num["elapsed"],
  "chol / num  / L-BFGS-B / log" = t.log.num["elapsed"],
  "chol / anal / L-BFGS-B"       = t.grad["elapsed"],
  "chol / num  / BFGS / log"     = t.bfgs.num["elapsed"],
  "chol / anal / BFGS / log"     = t.bfgs.grad["elapsed"],
  "CG   / num  / L-BFGS-B"       = t.cg.num["elapsed"],
  "CG   / anal / L-BFGS-B"       = t.cg.grad["elapsed"]
)
knitr::kable(data.frame(elapsed = round(times, 2)),
             caption = "Elapsed time (seconds)")
```

Table 2: Elapsed time (seconds)

	elapsed
chol / num / L-BFGS-B.elapsed	0.44
chol / num / L-BFGS-B / log.elapsed	0.29
chol / anal / L-BFGS-B.elapsed	5.60
chol / num / BFGS / log.elapsed	0.17
chol / anal / BFGS / log.elapsed	1.55
CG / num / L-BFGS-B.elapsed	101.90
CG / anal / L-BFGS-B.elapsed	21.16

All approaches recover the true parameters reasonably well. L-BFGS_B 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.

```r
factr_vals <- c(1e7, 5e6, 1e6, 5e5, 1e5, 5e4, 1e4, 5e3, 1e3, 5e2, 1e2,5e1, 1e1)   # 1e7 is the R defaul
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

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 3: 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+07	0e+00	0.2963	1.6726	0.0929	0.1833	0.0000	18	18
5e+06	0e+00	0.2963	1.6726	0.0929	0.1833	0.0000	18	18
1e+06	0e+00	0.2963	1.6727	0.0929	0.1834	0.0000	20	20
5e+05	1e-04	0.2963	1.6727	0.0929	0.1834	0.0000	20	20
1e+05	5e-04	0.2963	1.6727	0.0929	0.1834	0.0000	19	19
5e+04	1e-03	0.2963	1.6727	0.0929	0.1834	0.0000	19	19
1e+04	5e-03	0.2963	1.6727	0.0929	0.1834	0.0000	19	19
5e+03	1e-02	0.2963	1.6727	0.0929	0.1834	0.0000	19	19
1e+03	5e-02	0.2963	1.6726	0.0929	0.1833	0.0000	18	18
5e+02	1e-01	0.2963	1.6726	0.0929	0.1833	0.0000	18	18
1e+02	5e-01	0.2925	1.6298	0.0966	0.1406	0.1669	12	12
5e+01	1e+00	0.2925	1.6298	0.0966	0.1406	0.1669	12	12
1e+01	5e+00	0.2914	1.5664	0.1025	0.0775	0.4194	11	11

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 (`ctrl`) 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      = ctrl)
  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))
```

```
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 4: 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	0.2528	0.0095	1.4471	0.2651	0.1141	0.0372
30	0.2502	0.0110	1.5093	0.2607	0.1010	0.0357
100	0.2494	0.0122	1.4593	0.2562	0.1085	0.0360

```
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)
}
```

Variance seems constant here. Why? The cost scales linearly with `nprobe`.

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
```

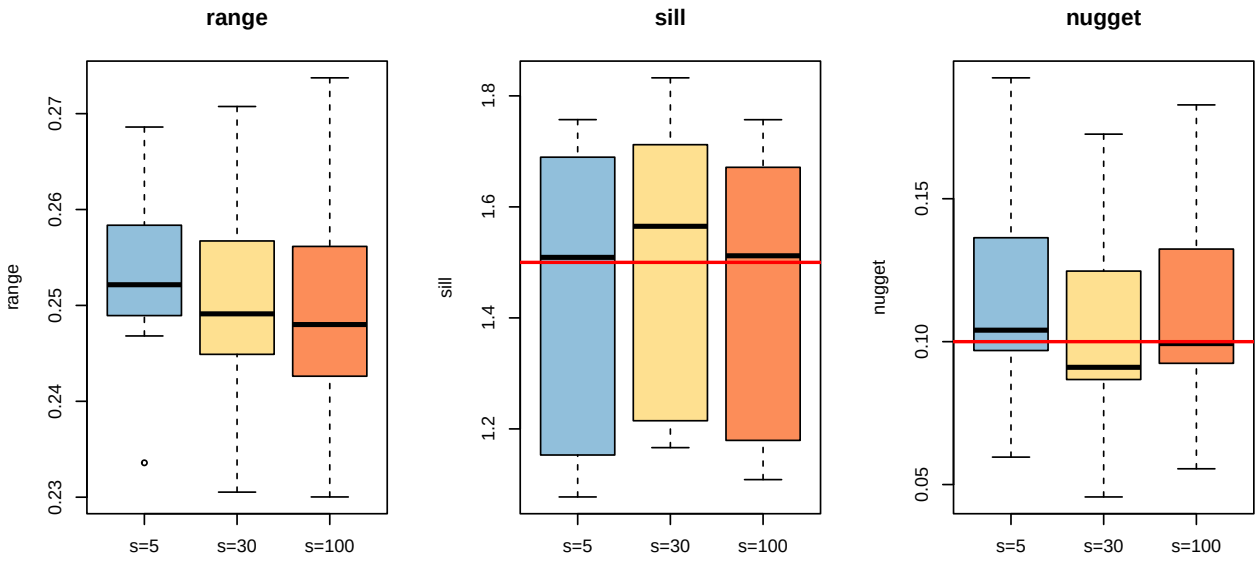

Figure 2: Distribution of estimates by nprobe; red line = truth.

```
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 5: 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 6: Elapsed time (seconds) for n = 20000

	elapsed	perIteration
chol / num	308.79	7.18
CG / prec	944.20	12.76
CG / anal	624.32	8.44

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.