

# Parameters and efficiencies of iterative solvers and preconditioners in spam

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

## Settings

Choose `setting` to control the computational load throughout this document. "**fast**" is suitable for a quick check (takes less than a minute); "**normal**" for routine use (a few minutes); "**extensive**" for publication-quality results (couple hours on my laptop). As we do not estimate parameters, we will set the range according to `ngamma`.

```
setting <- "fast"    # one of: "fast", "normal", "extensive"
setting <- "normal"
#setting <- "extensive"

if (setting=="fast") {
  n          = 1000    # single-problem size
  ngamma     = 25      # expected neighbours (controls taper radius)
  nprobe     = 10      # stochastic log-det probes
  lfil       = 10      # fill level for ildlt / ilu
  lfil_vec   = c(2, 5, 10, 20)*2    # sweep for lfil illustration
  nprobe_vec = c(5, 10, 20, 50)      # sweep for nprobe illustration
  omega_vec  = seq(0.25, 1.75, by = 0.25) # sweep for SSOR omega illustration
  miniter_vec = c(1, 5, 10, 20)      # sweep for miniter illustration
  n_vec      = c(1, 2, 5) * 1000    # sizes for scaling section
}
if (setting=="normal") {
  n          = 5000
  ngamma     = 45
  nprobe     = 50
  lfil       = 50
  lfil_vec   = 5*c(10, 20, 50, 80)
  nprobe_vec = c(10, 20, 50, 100)
  omega_vec  = seq(0.1, 1.9, by = 0.2)
  miniter_vec = c(1, 5, 10, 20, 50)
  n_vec      = c(1, 2, 5, 10, 20) * 1000
}
if (setting=="extensive") {
  n          = 20000
  ngamma     = 100
  nprobe     = 50
  lfil       = 75
  lfil_vec   = c(50, 75, 100, 150, 200, 250)
  nprobe_vec = c(10, 20, 50, 100, 200)
  omega_vec  = seq(0.1, 1.5, by = 0.1)
  miniter_vec = c(1, 5, 10, 20, 50, 100)
  n_vec      = c(1, 2, 5, 10, 20, 50, 100) * 1000
}
```

```
}

theta0 <- c(1.1, 0.1)  # without range!  sill, nugget only!!
```

We work with a single moderate problem for the first sections, then sweep parameters, and finally loop over `n_vec` for the scaling comparison.

```
set.seed(142)
locs <- cbind(runif(n), runif(n))
delta <- sqrt(ngamma / (n * pi))
D <- nearest.kdtree(locs, upper = NULL, delta = delta)
Sigma <- cov.sph(D, c(delta, theta0))
y <- c(rmvnorm.spam(1, Sigma = Sigma))
cat("n =", n, " | nnz =", nnz(Sigma), " | range =", round(delta, 5), "\n")
#> n = 5000 | nnz = 220480 | range = 0.05352
```

## Incomplete factorisation preconditioners

### Systematic study: `lfil`, `droptol`, ordering

We sweep `lfil_vec` (from the settings block), three drop tolerances, and the three available orderings for `ildlt`. For each combination we record: build time (s), factor memory (kB), and relative approximation quality  $\|M^{-1}b - A^{-1}b\|/\|A^{-1}b\|$ .

```
droptol_vec <- c(0, 1e-3, 1e-2)
orderings <- c("none", "MMD", "RCM")

combos <- expand.grid(lfil = lfil_vec,
                     droptol = droptol_vec,
                     ordering = orderings,
                     stringsAsFactors = FALSE)

res_pc <- data.frame(combos,
                    mem_kb = NA_real_,
                    t_build = NA_real_,
                    rel_err = NA_real_)

b <- rnorm(n)
z_exact <- solve(Sigma, b)

for (i in seq_len(nrow(combos))) {
  res_pc$t_build[i] <- system.time(
    pc <- ildlt(Sigma,
                lfil = combos$lfil[i],
                droptol = combos$droptol[i],
                ordering = combos$ordering[i],
                shift = 1)
  )["elapsed"]
  res_pc$mem_kb[i] <- as.numeric(object.size(pc)) / 1024
  z_pc <- solve(pc, b)
  res_pc$rel_err[i] <- sqrt(sum((z_exact - z_pc)^2)) / sqrt(sum(z_exact^2))
}
```

The two key trade-offs are visualized below. Left panel: factor memory vs. relative error — increasing `lfil` moves points left and down (better quality, more memory). Right panel: build time vs. relative error.

droptol primarily reduces memory and build time at the cost of quality; ordering mainly affects the fill pattern.

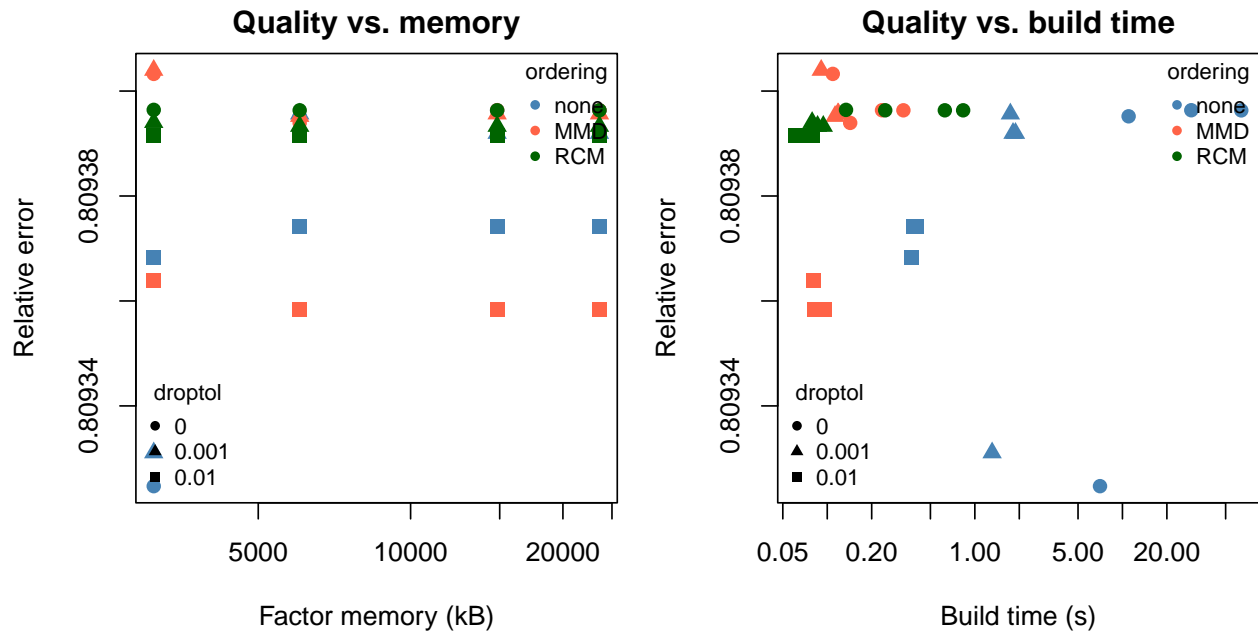
```
cols_ord <- c(none = "steelblue", MMD = "tomato", RCM = "darkgreen")
pch_drop <- c(16, 17, 15) # one symbol per droptol level

pc_col <- cols_ord[res_pc$ordering]
pc_pch <- pch_drop[match(res_pc$droptol, droptol_vec)]

par(mfrow = c(1, 2), mar = c(4, 4, 2, 1))

plot(res_pc$mem_kb, res_pc$rel_err, log = "xy",
     col = pc_col, pch = pc_pch, cex = 1.2,
     xlab = "Factor memory (kB)", ylab = "Relative error",
     main = "Quality vs. memory")
legend("topright", legend = names(cols_ord), col = cols_ord, pch = 16,
      title = "ordering", bty = "n", cex = 0.8)
legend("bottomleft", legend = as.character(droptol_vec), pch = pch_drop,
      title = "droptol", bty = "n", cex = 0.8)

plot(res_pc$t_build, res_pc$rel_err, log = "xy",
     col = pc_col, pch = pc_pch, cex = 1.2,
     xlab = "Build time (s)", ylab = "Relative error",
     main = "Quality vs. build time")
legend("topright", legend = names(cols_ord), col = cols_ord, pch = 16,
      title = "ordering", bty = "n", cex = 0.8)
legend("bottomleft", legend = as.character(droptol_vec), pch = pch_drop,
      title = "droptol", bty = "n", cex = 0.8)
```



## CG solver

Systematic study: preconditioner, lfil, omega, tolerance

Three sub-sweeps on the same (Sigma, b) pair.

**Sweep 1 — SSOR, varying  $\omega$ .** SSOR requires no explicit build step; the relaxation parameter  $\omega$  controls convergence. The expected behaviour is a U-curve in iteration count with a minimum near the optimal  $\omega$ .

```
res_ssor <- data.frame(
  omega = omega_vec,
  iter = NA_integer_,
  relres = NA_real_,
  t_solve = NA_real_
)
for (i in seq_along(omega_vec)) {
  res_ssor$t_solve[i] <- system.time(
    r <- cg(Sigma, b,
      control = get.iter.control(precond = "ssor", omega = omega_vec[i]))
  )["elapsed"]
  res_ssor$iter[i] <- r$iter
  res_ssor$relres[i] <- r$relres
}
```

**Sweep 2 — ildlt (MMD ordering), varying  $l_{fil}$ .** Build and solve are timed separately so we can distinguish preconditioner construction cost from solve cost.

```
res_ildlt <- data.frame(
  lfil = lfil_vec,
  iter = NA_integer_,
  relres = NA_real_,
  t_build = NA_real_,
  t_solve = NA_real_
)
for (i in seq_along(lfil_vec)) {
  res_ildlt$t_build[i] <- system.time(
    pc <- ildlt(Sigma, lfil = lfil_vec[i], ordering = "MMD", shift = 1)
  )["elapsed"]
  res_ildlt$t_solve[i] <- system.time(
    r <- cg(Sigma, b, control = get.iter.control(precond = pc))
  )["elapsed"]
  res_ildlt$iter[i] <- r$iter
  res_ildlt$relres[i] <- r$relres
}
```

**Sweep 3 — tolerance, no preconditioner.**

```
tol_vec <- c(1e-3, 1e-5, 1e-7, 1e-10)
res_tol <- data.frame(
  tol = tol_vec,
  iter = NA_integer_,
  relres = NA_real_,
  t_solve = NA_real_
)
for (i in seq_along(tol_vec)) {
  res_tol$t_solve[i] <- system.time(
    r <- cg(Sigma, b, control = get.iter.control(tol = tol_vec[i]))
  )["elapsed"]
  res_tol$iter[i] <- r$iter
  res_tol$relres[i] <- r$relres
}
```

```
res_tol
#>      tol iter      relres t_solve
#> 1 1e-03  42 8.879821e-04  0.011
#> 2 1e-05  68 8.831115e-06  0.020
#> 3 1e-07  94 9.007160e-08  0.023
#> 4 1e-10 131 9.010342e-11  0.032
```

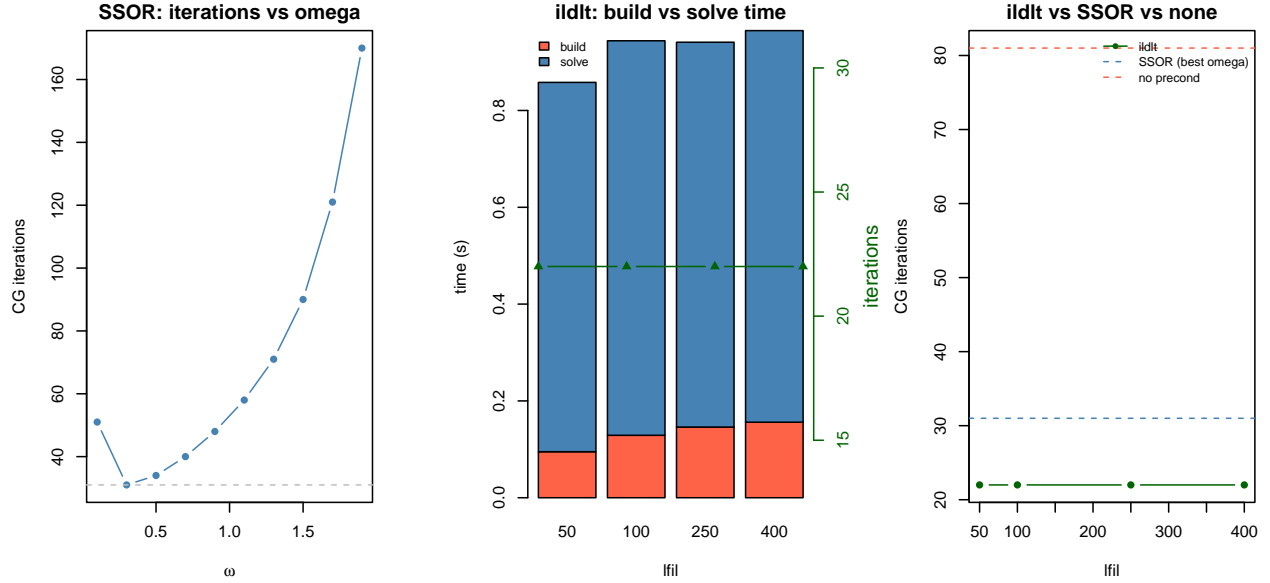
Left panel: SSOR iteration count vs.  $\omega$  — the U-curve identifies the optimal relaxation. Right panel: for ildlt, build time and solve time stacked by lfil, with iteration count on a secondary axis.

```
par(mfrow = c(1, 3), mar = c(4, 4, 2, 4))

## Panel 1: SSOR omega vs iterations
plot(res_ssor$omega, res_ssor$iter, type = "b", pch = 16, col = "steelblue",
     xlab = expression(omega), ylab = "CG iterations",
     main = "SSOR: iterations vs omega")
abline(h = res_ssor$iter[which.min(res_ssor$iter)], lty = 2, col = "gray")

## Panel 2: ildlt lfil - stacked build + solve time
bp <- barplot(rbind(res_ildlt$t_build, res_ildlt$t_solve),
             beside = FALSE,
             names.arg = lfil_vec,
             col = c("tomato", "steelblue"),
             xlab = "lfil", ylab = "time (s)",
             main = "ildlt: build vs solve time")
legend("topleft", legend = c("build", "solve"), fill = c("tomato", "steelblue"),
     bty = "n", cex = 0.8)
par(new = TRUE)
plot(seq_along(lfil_vec), res_ildlt$iter, type = "b", pch = 17, col = "darkgreen",
     axes = FALSE, xlab = "", ylab = "")
axis(4, col = "darkgreen", col.axis = "darkgreen")
mtext("iterations", side = 4, line = 2.5, col = "darkgreen", cex = 0.8)

## Panel 3: ildlt iterations vs lfil with SSOR and no-precond as references
iter_none <- cg(Sigma, b)$iter
iter_ssor <- res_ssor$iter[which.min(res_ssor$iter)]
plot(lfil_vec, res_ildlt$iter, type = "b", pch = 16, col = "darkgreen",
     xlab = "lfil", ylab = "CG iterations",
     main = "ildlt vs SSOR vs none",
     ylim = range(c(res_ildlt$iter, iter_none, iter_ssor)))
abline(h = iter_none, lty = 2, col = "tomato")
abline(h = iter_ssor, lty = 2, col = "steelblue")
legend("topright",
     legend = c("ildlt", "SSOR (best omega)", "no precondition"),
     col = c("darkgreen", "steelblue", "tomato"),
     pch = c(16, NA, NA), lty = c(1, 2, 2),
     bty = "n", cex = 0.8)
```



## Stochastic log-determinant

The exact log-determinant serves as the reference throughout.

```
ld.exact <- 2 * sum(log(diag(chol(Sigma))))
```

### Systematic study: nprobe, preconditioner, seed variability

**Sweep 1 — nprobe.** More probes reduce variance at linear cost in time. The estimator is unbiased so the expected error decays as  $C/\sqrt{n_{\text{probe}}}$ .

```
res_nprobe <- data.frame(
  nprobe = nprobe_vec,
  logdet = NA_real_,
  se      = NA_real_,
  rel_err = NA_real_,
  t_solve = NA_real_
)
for (i in seq_along(nprobe_vec)) {
  res_nprobe$t_solve[i] <- system.time(
    r <- logdet.cg(Sigma,
      control = get.iter.control(nprobe = nprobe_vec[i], seed = 1))
  )["elapsed"]
  res_nprobe$logdet[i] <- r$logdet
  res_nprobe$se[i] <- r$logdet.se
  res_nprobe$rel_err[i] <- abs(r$logdet - ld.exact) / abs(ld.exact)
}
res_nprobe
#>   nprobe  logdet      se  rel_err t_solve
#> 1     10 -3241.106 32.31597 0.006496114 0.209
#> 2     20 -3252.344 19.11801 0.009985921 0.410
#> 3     50 -3246.296 15.72023 0.008107876 1.083
#> 4    100 -3245.612 10.04387 0.007895431 2.100
```

**Sweep 2 — preconditioner and solver at fixed nprobe.** A better preconditioner yields a better-conditioned Lanczos recurrence and tighter Ritz-value approximations. `logdet.cg` and `logdet.minres` are

called with identical `get.iter.control()` parameters — the interface is the same for both solvers.

```
pc_ildlt <- ildlt(Sigma, lfil = lfil, ordering = "MMD")

## identical control objects for both solvers
ic_none <- get.iter.control(nprobe = nprobe, seed = 1)
ic_ssor <- get.iter.control(nprobe = nprobe, seed = 1, precondition = "ssor")
ic_ildlt <- get.iter.control(nprobe = nprobe, seed = 1, precondition = pc_ildlt)

## combine: (label, solver function, control)
cases <- list(
  list(label = "cg.none", fn = logdet.cg, ic = ic_none),
  list(label = "cg.ssor", fn = logdet.cg, ic = ic_ssor),
  list(label = "cg.ildlt", fn = logdet.cg, ic = ic_ildlt),
  list(label = "mr.none", fn = logdet.minres, ic = ic_none),
  list(label = "mr.ssor", fn = logdet.minres, ic = ic_ssor)
)

res_lpc <- data.frame(
  label = sapply(cases, `[`, "label"),
  logdet = NA_real_,
  se = NA_real_,
  rel_err = NA_real_,
  t_solve = NA_real_,
  stringsAsFactors = FALSE
)

for (i in seq_along(cases)) {
  res_lpc$t_solve[i] <- system.time(
    r <- cases[[i]]$fn(Sigma, control = cases[[i]]$ic)
  )["elapsed"]
  res_lpc$logdet[i] <- r$logdet
  res_lpc$se[i] <- r$logdet.se
  res_lpc$rel_err[i] <- abs(r$logdet - ld.exact) / abs(ld.exact)
}

res_lpc
#>      label      logdet      se      rel_err t_solve
#> 1 cg.none -3246.296 15.72023 0.008107876 1.064
#> 2 cg.ssor -3246.296 15.72023 0.008107876 1.059
#> 3 cg.ildlt -3246.296 15.72023 0.008107876 1.891
#> 4 mr.none -3246.296 15.72023 0.008107876 1.037
#> 5 mr.ssor -3246.296 15.72023 0.008107876 1.046
```

**Sweep 3 — seed variability.** Multiple independent runs at fixed `nprobe` show the distribution of the estimator around the exact value.

```
n_seeds <- 20
ld_seeds <- sapply(seq_len(n_seeds), function(s)
  logdet.cg(Sigma,
    control = get.iter.control(nprobe = nprobe, seed = s))$logdet
)
```

Panel 1: estimate  $\pm 2$  SE vs. `nprobe` with the exact value as reference. Panel 2: distribution of estimates over seeds (`nprobe` fixed) — visually confirms unbiasedness. Panel 3: preconditioner and solver comparison (CG and MINRES with the same control parameters), showing estimates  $\pm 2$  SE against the exact value; solve times are annotated on the x-axis.

```

par(mfrow = c(1, 3), mar = c(4, 4, 2, 1))

## Panel 1: estimate +/- 2 SE vs nprobe, exact value as reference
ylim1 <- range(c(res_nprobe$logdet + 2 * res_nprobe$se,
                 res_nprobe$logdet - 2 * res_nprobe$se,
                 ld.exact))
plot(res_nprobe$nprobe, res_nprobe$logdet,
     type = "b", pch = 16, col = "steelblue",
     xlab = "nprobe", ylab = "log-det estimate",
     main = "Estimate +/- 2 SE vs. nprobe",
     ylim = ylim1)
arrows(res_nprobe$nprobe,
       res_nprobe$logdet - 2 * res_nprobe$se,
       res_nprobe$nprobe,
       res_nprobe$logdet + 2 * res_nprobe$se,
       length = 0.05, angle = 90, code = 3, col = "steelblue")
abline(h = ld.exact, lty = 2, col = "tomato", lwd = 2)
legend("topright", legend = c("estimate +/- 2 SE", "exact"),
      col = c("steelblue", "tomato"), pch = c(16, NA),
      lty = c(1, 2), lwd = c(1, 2), bty = "n", cex = 0.8)

## Panel 2: distribution over seeds - confirms unbiasedness
hist(ld_seeds, breaks = 10, col = "lightblue", border = "white",
     xlab = "log-det estimate", main = "Seed variability (fixed nprobe)")
abline(v = ld.exact, col = "tomato", lwd = 2)
abline(v = mean(ld_seeds), col = "steelblue", lwd = 2, lty = 2)
legend("topright", legend = c("exact", "mean over seeds"),
      col = c("tomato", "steelblue"), lwd = 2, lty = c(1, 2),
      bty = "n", cex = 0.8)

## Panel 3: solver x preconditioner - estimate +/- 2 SE, exact as reference.
## CG: filled symbols; MINRES: open symbols.
pc_x <- seq_along(res_lpc$label)
is_cg <- grepl("^cg", res_lpc$label)
pc_pch <- ifelse(is_cg, 16L, 17L)
pc_col <- ifelse(grepl("none$", res_lpc$label), "gray50",
                ifelse(grepl("ssor$", res_lpc$label), "steelblue", "tomato"))
ylim3 <- range(c(res_lpc$logdet + 2 * res_lpc$se,
                 res_lpc$logdet - 2 * res_lpc$se,
                 ld.exact))
plot(pc_x, res_lpc$logdet,
     pch = pc_pch, col = pc_col,
     xlab = "", ylab = "log-det estimate",
     main = "Solver x preconditioner",
     xlim = c(0.5, length(pc_x) + 0.5), ylim = ylim3,
     xaxt = "n")
axis(1, at = pc_x, labels = res_lpc$label, cex.axis = 0.8)
arrows(pc_x,
       res_lpc$logdet - 2 * res_lpc$se,
       pc_x,
       res_lpc$logdet + 2 * res_lpc$se,
       length = 0.05, angle = 90, code = 3, col = pc_col)
abline(h = ld.exact, lty = 2, lwd = 2, col = "gray30")

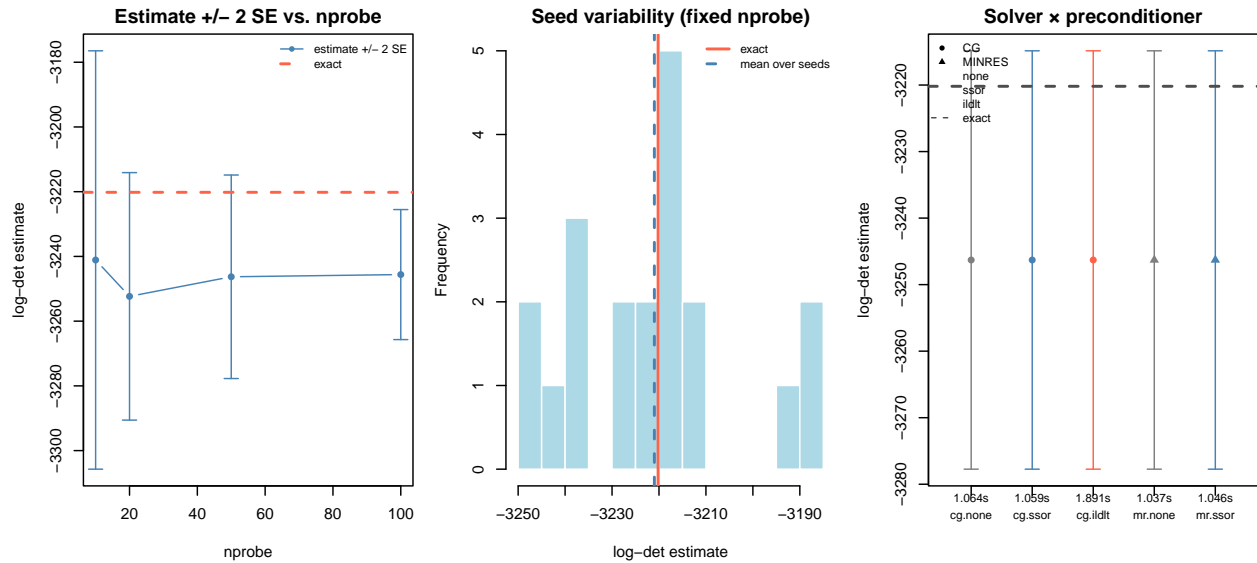
```



```

text(pc_x, par("usr")[3], labels = paste0(round(res_lpc$t_solve, 3), "s"),
     pos = 1, xpd = TRUE, cex = 0.7)
legend("topleft",
      legend = c("CG", "MINRES", "none", "ssor", "ildlt", "exact"),
      pch = c(16, 17, NA, NA, NA, NA),
      lty = c(NA, NA, NA, NA, NA, 2),
      col = c("black", "black", "gray50", "steelblue", "tomato", "gray30"),
      bty = "n", cex = 0.75)

```



#### Sweep 4 — miniter: minimum Lanczos steps for accurate spectral coverage.

`miniter` is relevant only for `cg.lanczos`, `minres.lanczos`, `logdet.cg`, and `logdet.minres`. It forces at least that many Lanczos recurrence steps regardless of solver convergence. The motivation: a good preconditioner (e.g., SSOR) can cause the solver to converge in very few steps, leaving the Lanczos tridiagonal matrix  $T_k$  too small to cover the spectrum accurately — and hence giving a poor log-det estimate. Increasing `miniter` trades extra Lanczos steps for better spectral coverage.

We first illustrate the mechanism on a single probe solve (how many steps are actually performed), then sweep `miniter` for the full estimator. Both `logdet.cg` and `logdet.minres` are called with the same `get.iter.control()` object.

```

## Single probe: show how iter grows with miniter under SSOR vs no precondition.
set.seed(1)
v_demo <- sample(c(-1.0, 1.0), n, replace = TRUE)
iter_free_none <- cg.lanczos(Sigma, v_demo,
                           control = get.iter.control(miniter = 1L))$iter
iter_free_ssor <- cg.lanczos(Sigma, v_demo,
                           control = get.iter.control(precond = "ssor",
                                                         miniter = 1L))$iter
iter_min_ssor <- cg.lanczos(Sigma, v_demo,
                           control = get.iter.control(precond = "ssor",
                                                         miniter = max(miniter_vec)))$iter

cat("Free convergence: no precondition = ", iter_free_none,
    "iter | SSOR = ", iter_free_ssor, "iter\n")
#> Free convergence: no precondition = 81 iter | SSOR = 34 iter
cat("SSOR + miniter = ", max(miniter_vec), ":", iter_min_ssor, "iter\n")
#> SSOR + miniter = 50 : 50 iter

```

With SSOR the solver converges quickly (few natural steps), so miniter is the binding constraint. Without preconditioning the solver runs many steps regardless.

*## Full estimator: vary miniter with SSOR, identical ic for CG and MINRES.*

```
res_miniter <- data.frame(
  miniter = rep(miniter_vec, 2L),
  solver = rep(c("CG", "MINRES"), each = length(miniter_vec)),
  logdet = NA_real_,
  se = NA_real_,
  rel_err = NA_real_,
  t_solve = NA_real_
)
for (i in seq_along(miniter_vec)) {
  ic <- get.iter.control(nprobe = nprobe, seed = 1, precondition = "ssor",
    miniter = miniter_vec[i])
  j <- i + length(miniter_vec)

  res_miniter$t_solve[i] <- system.time(
    r <- logdet.cg(Sigma, control = ic))["elapsed"]
  res_miniter$logdet[i] <- r$logdet
  res_miniter$se[i] <- r$logdet.se
  res_miniter$rel_err[i] <- abs(r$logdet - ld.exact) / abs(ld.exact)

  res_miniter$t_solve[j] <- system.time(
    r <- logdet.minres(Sigma, control = ic))["elapsed"]
  res_miniter$logdet[j] <- r$logdet
  res_miniter$se[j] <- r$logdet.se
  res_miniter$rel_err[j] <- abs(r$logdet - ld.exact) / abs(ld.exact)
}
res_miniter
#>      miniter solver      logdet      se      rel_err t_solve
#> 1          1    CG -3246.296 15.72023 0.008107876 0.993
#> 2          5    CG -3246.296 15.72023 0.008107876 0.995
#> 3         10    CG -3246.296 15.72023 0.008107876 0.985
#> 4         20    CG -3246.296 15.72023 0.008107876 0.988
#> 5         50    CG -3246.296 15.72023 0.008107876 0.988
#> 6          1 MINRES -3246.296 15.72023 0.008107876 1.006
#> 7          5 MINRES -3246.296 15.72023 0.008107876 1.008
#> 8         10 MINRES -3246.296 15.72023 0.008107876 1.008
#> 9         20 MINRES -3246.296 15.72023 0.008107876 1.015
#> 10        50 MINRES -3246.296 15.72023 0.008107876 1.010
```

```
par(mfrow = c(1, 2), mar = c(4, 4, 2, 1))
```

```
cg_rows <- res_miniter$solver == "CG"
mr_rows <- res_miniter$solver == "MINRES"
```

*## Panel 1: estimate +/- 2 SE vs miniter*

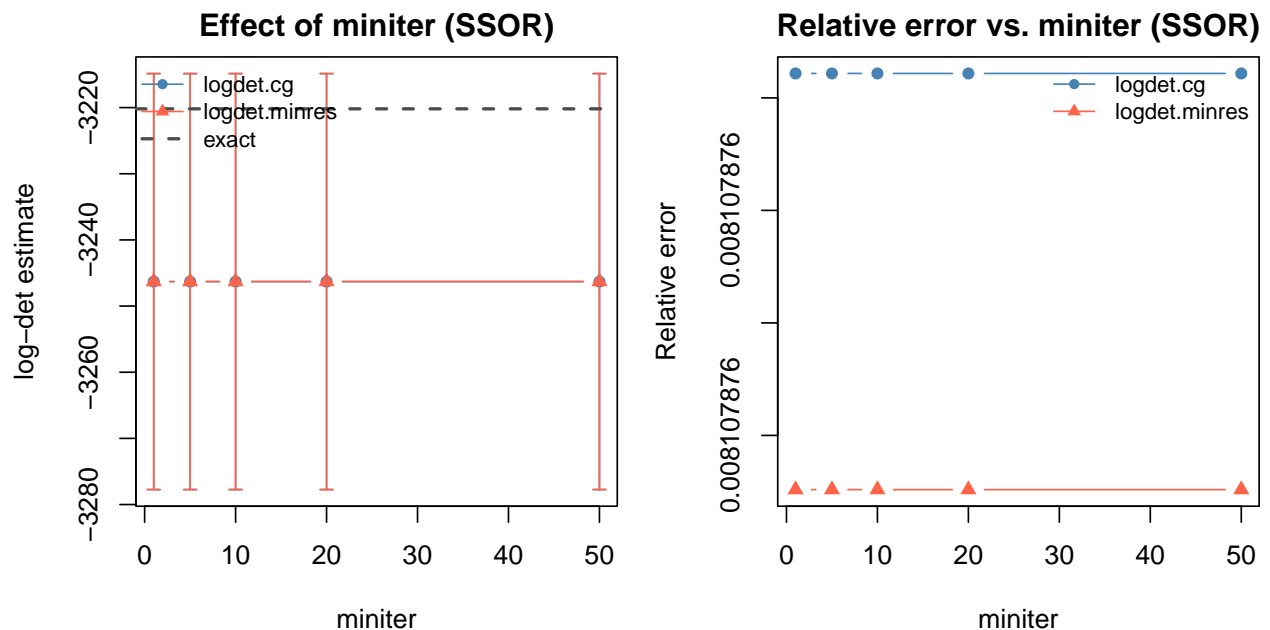
```
ylim1 <- range(c(res_miniter$logdet + 2 * res_miniter$se,
  res_miniter$logdet - 2 * res_miniter$se,
  ld.exact), na.rm = TRUE)
plot(res_miniter$miniter[cg_rows], res_miniter$logdet[cg_rows],
  type = "b", pch = 16, col = "steelblue", ylim = ylim1,
  xlab = "miniter", ylab = "log-det estimate",
  main = "Effect of miniter (SSOR)")
```

```

lines(res_miniter$miniter[mr_rows], res_miniter$logdet[mr_rows],
      type = "b", pch = 17, col = "tomato")
arrows(res_miniter$miniter[cg_rows],
       res_miniter$logdet[cg_rows] - 2 * res_miniter$se[cg_rows],
       res_miniter$miniter[cg_rows],
       res_miniter$logdet[cg_rows] + 2 * res_miniter$se[cg_rows],
       length = 0.04, angle = 90, code = 3, col = "steelblue")
arrows(res_miniter$miniter[mr_rows],
       res_miniter$logdet[mr_rows] - 2 * res_miniter$se[mr_rows],
       res_miniter$miniter[mr_rows],
       res_miniter$logdet[mr_rows] + 2 * res_miniter$se[mr_rows],
       length = 0.04, angle = 90, code = 3, col = "tomato")
abline(h = ld.exact, lty = 2, lwd = 2, col = "gray30")
legend("topleft",
      legend = c("logdet.cg", "logdet.minres", "exact"),
      col = c("steelblue", "tomato", "gray30"),
      pch = c(16, 17, NA), lty = c(1, 1, 2), lwd = c(1, 1, 2),
      bty = "n", cex = 0.85)

## Panel 2: relative error vs miniter (log scale)
ylim2 <- range(res_miniter$rel_err, na.rm = TRUE)
plot(res_miniter$miniter[cg_rows], res_miniter$rel_err[cg_rows],
     type = "b", pch = 16, col = "steelblue",
     log = "y", ylim = ylim2,
     xlab = "miniter", ylab = "Relative error",
     main = "Relative error vs. miniter (SSOR)")
lines(res_miniter$miniter[mr_rows], res_miniter$rel_err[mr_rows],
      type = "b", pch = 17, col = "tomato")
legend("topright",
      legend = c("logdet.cg", "logdet.minres"),
      col = c("steelblue", "tomato"), pch = c(16, 17), lty = 1,
      bty = "n", cex = 0.85)

```



## Quadratic form

`quadform.cg` solves  $Az = y$  via CG then returns  $y^T z$ . Accuracy is governed entirely by the CG solve: `tol`, `maxiter`, and the preconditioner. `nprobe` is ignored.

```
qf.exact <- as.double(y %*% solve(Sigma, y))
```

### Systematic study: `tol`, preconditioner, `maxiter`

**Sweep 1 — `tol`.** Tighter tolerance means more CG iterations and a more accurate solve, directly reducing the error in the quadratic form.

```
tol_vec <- c(1e-2, 1e-4, 1e-6, 1e-8, 1e-10)

res_qftol <- data.frame(
  tol      = tol_vec,
  quad     = NA_real_,
  rel_err  = NA_real_,
  iter     = NA_integer_,
  t_solve  = NA_real_
)
for (i in seq_along(tol_vec)) {
  res_qftol$t_solve[i] <- system.time(
    r <- quadform.cg(Sigma, y, control = get.iter.control(tol = tol_vec[i]))
  )["elapsed"]
  res_qftol$quad[i] <- r$quad
  res_qftol$rel_err[i] <- abs(r$quad - qf.exact) / abs(qf.exact)
  res_qftol$iter[i] <- r$iter
}
res_qftol
#>      tol      quad      rel_err iter t_solve
#> 1 1e-02 5023.824 9.286153e-05  25  0.007
#> 2 1e-04 5024.290 7.173785e-09  52  0.014
#> 3 1e-06 5024.290 3.333596e-09  78  0.020
#> 4 1e-08 5024.290 2.966910e-13 103  0.025
#> 5 1e-10 5024.290 2.624783e-14 128  0.031
```

**Sweep 2 — preconditioner at default `tol`.** A better preconditioner reduces iterations and typically also reduces the quadratic form error at fixed tolerance.

```
qf_precond_cases <- list(
  none = get.iter.control(),
  ssor = get.iter.control(precond = "ssor"),
  ildlt = get.iter.control(precond = ildlt(Sigma, lfil = lfil, ordering = "MMD"))
)

res_qfpc <- data.frame(
  precondition = names(qf_precond_cases),
  quad        = NA_real_,
  rel_err     = NA_real_,
  iter        = NA_integer_,
  t_solve     = NA_real_
)
for (i in seq_along(qf_precond_cases)) {
  res_qfpc$t_solve[i] <- system.time(
    r <- quadform.cg(Sigma, y, control = qf_precond_cases[[i]])
  )
}
```

```

) ["elapsed"]
res_qfpc$quad[i] <- r$quad
res_qfpc$rel_err[i] <- abs(r$quad - qf.exact) / abs(qf.exact)
res_qfpc$iter[i] <- r$iter
}
res_qfpc
#>   precondition   quad      rel_err iter t_solve
#> 1      none 5024.29 3.333596e-09   78   0.019
#> 2      ssor 5024.29 8.055370e-13   32   0.026
#> 3     idlt 5024.29 2.311620e-13    5   0.754

```

**Sweep 3 — maxiter.** When `maxiter` is too small CG stops before convergence, leaving a residual that contaminates the quadratic form. The error curve drops sharply once `maxiter` exceeds the iterations needed for convergence.

```

maxiter_vec <- c(5, 10, 20, 50, 100, 200)

res_qfmax <- data.frame(
  maxiter = maxiter_vec,
  quad    = NA_real_,
  rel_err = NA_real_,
  iter    = NA_integer_,
  ierr    = NA_integer_
)
for (i in seq_along(maxiter_vec)) {
  r <- quadform.cg(Sigma, y,
    control = get.iter.control(maxiter = maxiter_vec[i]))
  res_qfmax$quad[i] <- r$quad
  res_qfmax$rel_err[i] <- abs(r$quad - qf.exact) / abs(qf.exact)
  res_qfmax$iter[i] <- r$iter
  res_qfmax$ierr[i] <- r$ierr
}
#> Warning: cg: did not converge in 5 iterations; relative residual =
#> 0.469525948680046
#> Warning: cg: did not converge in 10 iterations; relative residual =
#> 0.155051239809467
#> Warning: cg: did not converge in 20 iterations; relative residual =
#> 0.0241583110295742
#> Warning: cg: did not converge in 50 iterations; relative residual =
#> 0.000121452469582937
res_qfmax
#>   maxiter   quad      rel_err iter ierr
#> 1      5 4228.124 1.584634e-01    5    1
#> 2     10 4913.417 2.206741e-02   10    1
#> 3     20 5021.387 5.777876e-04   20    1
#> 4     50 5024.290 1.493030e-08   50    1
#> 5    100 5024.290 3.333596e-09   78    0
#> 6    200 5024.290 3.333596e-09   78    0

```

```

par(mfrow = c(1, 3), mar = c(4, 4, 2, 1))

## Panel 1: tol vs relative error, iterations on right axis
plot(res_qftol$tol, res_qftol$rel_err, log = "xy",
  type = "b", pch = 16, col = "steelblue",

```

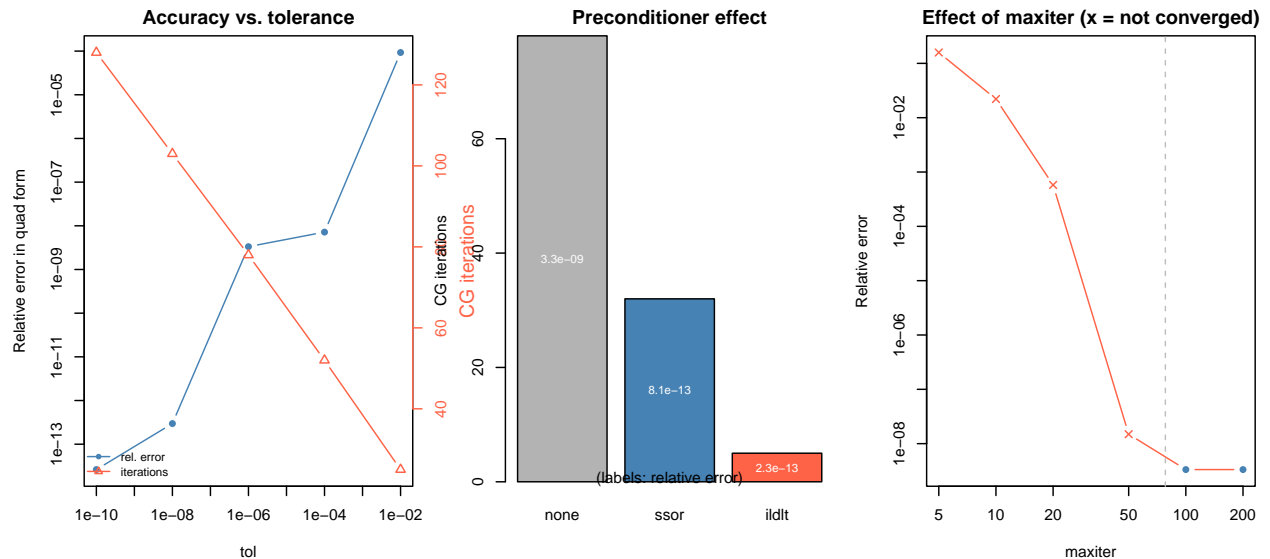
```

    xlab = "tol", ylab = "Relative error in quad form",
    main = "Accuracy vs. tolerance")
par(new = TRUE)
plot(res_qftol$tol, res_qftol$iter, log = "x", type = "b",
     pch = 2, col = "tomato", axes = FALSE, xlab = "", ylab = "")
axis(4, col = "tomato", col.axis = "tomato")
mtext("CG iterations", side = 4, line = 2.5, col = "tomato", cex = 0.8)
legend("bottomleft", legend = c("rel. error", "iterations"),
      col = c("steelblue", "tomato"), pch = c(16, 2), lty = 1,
      bty = "n", cex = 0.8)

## Panel 2: preconditioner comparison
cols3 <- c("gray70", "steelblue", "tomato")
bp <- barplot(res_qfpc$iter, names.arg = res_qfpc$precond,
             col = cols3, ylab = "CG iterations",
             main = "Preconditioner effect")
text(bp, res_qfpc$iter / 2,
     labels = format(res_qfpc$rel_err, digits = 2, scientific = TRUE),
     cex = 0.75, col = "white")
mtext("(labels: relative error)", side = 1, line = -1, cex = 0.7)

## Panel 3: maxiter vs relative error - convergence cliff
conv_iter <- res_qfmax$iter[which(res_qfmax$ierr == 0)[1]]
plot(res_qfmax$maxiter, res_qfmax$rel_err, log = "xy",
     type = "b", pch = ifelse(res_qfmax$ierr == 0, 16, 4),
     col = ifelse(res_qfmax$ierr == 0, "steelblue", "tomato"),
     xlab = "maxiter", ylab = "Relative error",
     main = "Effect of maxiter (x = not converged)")
if (!is.na(conv_iter))
  abline(v = conv_iter, lty = 2, col = "gray")

```



## Scaling comparison

For each  $n$  in `n_vec` (from the settings block) we generate a fresh random problem, build all preconditioners, and time each step separately. Methods compared:

label	preconditioner	build step?
chol	Cholesky (symbolic + numeric solve)	yes
cg_none	none	—
ssor	SSOR ( $\omega = 1$ , default)	—
ildlt	ildlt, MMD ordering, lfil from settings	yes

lfil is fixed at the single-problem value from the settings block. `system.time()` performs a garbage collection before each timed expression.

```
set.seed(12)

col_names <- c("n", "prep",
               "chol_sym", "chol_solve",
               "cg_none",
               "ssor_solve",
               "ildlt_build", "ildlt_solve")
sc_times <- matrix(NA_real_, nrow = length(n_vec), ncol = length(col_names),
                  dimnames = list(NULL, col_names))
sc_iter <- matrix(NA_integer_, nrow = length(n_vec), ncol = 3,
                 dimnames = list(NULL, c("cg_none", "ssor", "ildlt")))

for (i in seq_along(n_vec)) {
  ni <- n_vec[i]
  deltai <- sqrt(ngamma / (ni * pi))

  sc_times[i, "n"] <- ni

  sc_times[i, "prep"] <- system.time({
    loci <- cbind(runif(ni), runif(ni))
    Di <- nearest.kdtree(loci, delta = deltai, upper = TRUE,
                        memory = list(nnz = ngamma * ni))

    Di <- t(Di) + Di
    Si <- cov.sph(Di, c(deltai, theta0))
    yi <- rnorm(ni)
  })["elapsed"]

  ## Cholesky
  sc_times[i, "chol_sym"] <- system.time(
    Ri <- chol(Si)
  )["elapsed"]
  sc_times[i, "chol_solve"] <- system.time(
    solve.spam(Ri, yi)
  )["elapsed"]

  ## CG, no preconditioner
  sc_times[i, "cg_none"] <- system.time(
    r <- cg(Si, yi)
  )["elapsed"]
  sc_iter[i, "cg_none"] <- r$iter

  ## CG + SSOR (no build step)
  sc_times[i, "ssor_solve"] <- system.time(
    r <- cg(Si, yi, control = get.iter.control(precond = "ssor"))
  )["elapsed"]
}
```

```

) ["elapsed"]
sc_iter[i, "ssor"] <- r$iter

## CG + ildlt (build and solve separate)
sc_times[i, "ildlt_build"] <- system.time(
  pc_i <- ildlt(Si, lfil = lfil, ordering = "MMD")
) ["elapsed"]
sc_times[i, "ildlt_solve"] <- system.time(
  r <- cg(Si, yi, control = get.iter.control(precond = pc_i))
) ["elapsed"]
sc_iter[i, "ildlt"] <- r$iter
}

cbind(n = n_vec, sc_iter)
#>      n cg_none ssor ildlt
#> [1,] 1000      71   32    5
#> [2,] 2000      79   33    5
#> [3,] 5000      80   35    5
#> [4,] 10000     83   36    5
#> [5,] 20000     84   36    5
sc_times[, -1, drop = FALSE]
#>      prep chol_sym chol_solve cg_none ssor_solve ildlt_build ildlt_solve
#> [1,] 0.006  0.026    0.001  0.004    0.005    0.014    0.031
#> [2,] 0.012  0.015    0.001  0.008    0.011    0.031    0.122
#> [3,] 0.030  0.052    0.004  0.020    0.029    0.092    0.751
#> [4,] 0.069  0.157    0.008  0.041    0.058    0.214    3.086
#> [5,] 0.150  0.549    0.021  0.100    0.148    0.524   12.290
n_vec
#> [1] 1000 2000 5000 10000 20000

cols <- c(prep      = "gray70",
          chol_sym  = "darkblue",
          chol_solve = "steelblue",
          cg_none   = "tomato",
          ssor_solve = "orange",
          ildlt_build = "darkgreen",
          ildlt_solve = "green3")

## strip the n column for plotting
plot_mat <- sc_times[, -1, drop = FALSE]
ylim     <- range(plot_mat[plot_mat!=0], na.rm = TRUE)

matplot(n_vec, plot_mat, type = "b",
        pch = seq_along(cols), col = cols,
        log = "xy", xaxt = "n",
        xlab = "n", ylab = "time (s)",
        main = "Scaling: Cholesky vs CG vs SSOR vs ildlt",
        ylim = ylim)
axis(1, at = n_vec, labels = paste0(n_vec / 1000, "k"))
legend("topleft",
       legend = c("matrix prep",
                  "Cholesky symbolic", "Cholesky solve",
                  "CG no precondition",
                  "SSOR solve"),

```

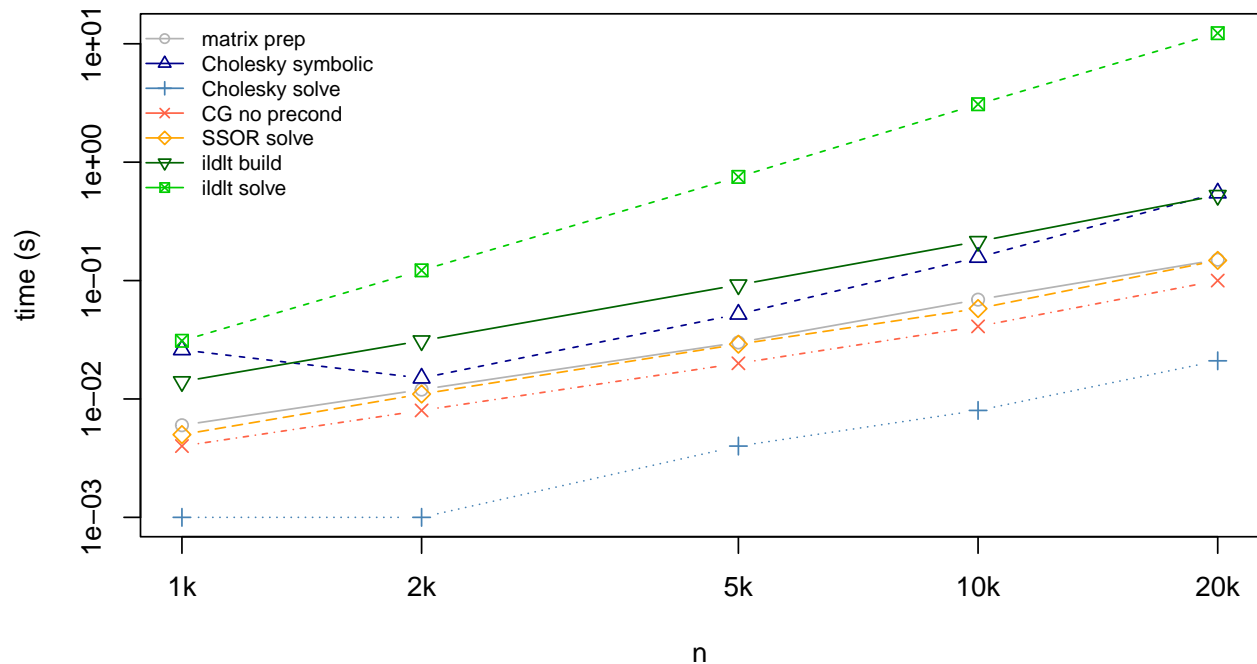


```

      "ildlt build", "ildlt solve"),
col = cols, pch = seq_along(cols), lty = 1,
bty = "n", cex = 0.75)

```

### Scaling: Cholesky vs CG vs SSOR vs ildlt



### Likelihood evaluation

For each  $n$  in  $n\_vec$  we generate a fresh dataset and time one `neg2loglik()` evaluation at `c(delta, theta0)`. Methods: Cholesky, CG with no preconditioner, CG + SSOR, CG + ildlt (MMD). The ildlt build is included inside the CG + ildlt timing since `neg2loglik` rebuilds the preconditioner at each call; the SSOR column has no such cost. `nprobe` is taken from the settings block.

```

set.seed(12)

ll_col_names <- c("n", "chol", "cg_none", "ssor", "ildlt")
ll_times <- matrix(NA_real_, nrow = length(n_vec), ncol = length(ll_col_names),
  dimnames = list(NULL, ll_col_names))

for (i in seq_along(n_vec)) {
  ni <- n_vec[i]
  deltai <- sqrt(ngamma / (ni * pi))
  thetai <- c(deltai, theta0)
  loci <- cbind(runif(ni), runif(ni))
  Di <- nearest.kdtree(loca, delta = deltai, upper = TRUE,
    memory = list(nnz = ngamma * ni))

  Di <- t(Di) + Di
  Si <- cov.sph(Di, thetai)
  yi <- c(rmvnorm.spam(1, Sigma = Si))

  ll_times[i, "n"] <- ni

```

```

ll_times[i, "chol"] <- system.time(
  neg2loglik(theta = thetai, y = yi, distmat = Di, cov.sph,
    solver = "chol")
)["elapsed"]

ll_times[i, "cg_none"] <- system.time(
  neg2loglik(theta = thetai, y = yi, distmat = Di, cov.sph,
    solver = "cg",
    iter.control = get.iter.control(nprobe = nprobe))
)["elapsed"]

ll_times[i, "ssor"] <- system.time(
  neg2loglik(theta = thetai, y = yi, distmat = Di, cov.sph,
    solver = "cg",
    iter.control = get.iter.control(nprobe = nprobe,
      precond = "ssor"))
)["elapsed"]

ll_times[i, "ildlt"] <- system.time(
  neg2loglik(theta = thetai, y = yi, distmat = Di, cov.sph,
    solver = "cg",
    iter.control = get.iter.control(nprobe = nprobe,
      precond = ildlt(Si, lfil = lfil,
        ordering = "MMD")))
)["elapsed"]
}

ll_cols <- c(chol = "darkblue",
  cg_none = "tomato",
  ssor = "orange",
  ildlt = "darkgreen")

plot_ll <- ll_times[, -1, drop = FALSE]
ylim_ll <- range(plot_ll, na.rm = TRUE)

matplot(n_vec, plot_ll, type = "b",
  pch = seq_along(ll_cols), col = ll_cols,
  log = "xy", xaxt = "n",
  xlab = "n", ylab = "time (s)",
  main = "neg2loglik evaluation time",
  ylim = ylim_ll)
axis(1, at = n_vec, labels = paste0(n_vec / 1000, "k"))
legend("topleft",
  legend = c("Cholesky", "CG no precond", "CG + SSOR", "CG + ildlt"),
  col = ll_cols, pch = seq_along(ll_cols), lty = 1,
  bty = "n", cex = 0.8)

```

## neg2loglik evaluation time

