AMD EPYC 9454 48-Core Processor

Machine `epyc-9454`. The tables with a cost column measured on this machine.

4 table(s) in this directory. Each is shown under the file that produces it, at the manuscript's column width, so a table that does not fit is visible here.

`tab-likelihood-loglik.tex`

Table 1: Effect of each implementation's covariance errors on the log-likelihood, T, beside the error of double-precision linear algebra on exact covariances, F, over 32 simulated data sets. $T = |\ell_x(\text{impl}) - \ell_x(\text{ref})|$, both log-likelihoods from a Cholesky factorisation and solve in extended precision (long double): of the implementation's double covariance matrix, and of the arbitrary-precision reference covariances, so T is the effect of the covariance errors up to the rounding of the extended-precision computation. $F = |\ell_d(\text{ref}) - \ell_x(\text{ref})|$, ℓ_d the double-precision LAPACK computation (`dpotrf`, `dpotrs`) on the reference covariances rounded to double: the error a double-precision likelihood carries whichever covariances enter it. Each cell: median / maximum over the data sets, in the unit its column head gives. At $\nu = 2.5$ the library evaluates the closed form of the half-integer order, and the row *closed form (libm exp)* is that closed form written by hand with the C library's `exp`. The last column counts, of the (n, ν) measured, those at which every covariance is finite and the matrix positive definite. Results were obtained on the AMD CPU.

	$n = 256$		$n = 2048$		
implementation	$\nu = 1.3$ (10^{-14})	$\nu = 2.5$ (10^{-13})	$\nu = 1.3$ (10^{-12})	$\nu = 2.5$ (10^{-11})	finite
double LAPACK, F	8.7 / 32	2.7 / 9.7	4.8 / 16	2.1 / 5.5	
libmatern (avx512)	5.2 / 18	1.4 / 5.0	1.2 / 4.2	0.40 / 1.2	10 of 10
libmatern (avx2)	4.8 / 22	1.4 / 5.0	1.0 / 4.9	0.40 / 1.2	10 of 10
libmatern (matched_fma)	4.7 / 22	1.4 / 5.2	0.98 / 4.6	0.35 / 1.3	10 of 10
libmatern (matched)	3.8 / 18	1.5 / 5.3	1.4 / 4.0	0.47 / 1.4	10 of 10
libmatern (scalar)	3.4 / 16	1.1 / 3.7	0.73 / 3.2	0.50 / 1.2	10 of 10
bessels.jl (scaled)	5.9 / 27	2.9 / 7.8	1.8 / 6.4	0.61 / 2.1	10 of 10
bessels.jl-old (scaled)	17 / 57	6.5 / 24	6.1 / 20	1.4 / 5.1	10 of 10
boost (exp-log)	3.4 / 11	3.6 / 7.9	1.0 / 3.0	0.66 / 2.3	10 of 10
gsl (exp-log)	15 / 33	5.3 / 18	1.7 / 6.6	1.2 / 4.5	10 of 10
std-cyl_bessel_k (exp-log)	14 / 41	5.0 / 11	3.3 / 8.3	1.1 / 2.8	10 of 10
closed form (libm exp)	–	0.98 / 3.5	–	0.41 / 1.7	5 of 5

The data: n sites in $[0, 1]^3$, $s = 0.25$, a nugget 10^{-2}; data set k is $y_k = Lw_k$, L the Cholesky factor of the closed form of order 1.5, w_k standard normal (Box–Muller, seed $11+k$), L and Lw_k formed in long double and rounded once, so the data do not depend on the machine's LAPACK (their bit hashes are in `likelihood_accuracy.csv`); the reference covariances are those of the covariance-matrix sets at $n = 256$ and 2048, whose sha256 the run records (`likelihood_refs.sha256`). –: not measured (the size exceeded the external's time budget), or not applicable (the closed form exists only at a half-integer order).

`tab-scipy-z_nu.tex`

Table 2: SciPy as one more external, the (z, ν) datasets: `scipy.special.kv` and `kve` of SciPy 1.18.1 from its own `xsf` headers (AMOS `zbesk`), compiled with `-ffp-contract=off` as the SciPy wheel is built (`make scipy-check` compares their values with the wheel's); the other compiled externals keep contraction on. The covariance is assembled per point by the harness with the C library's `pow`, `tgamma` and `lgamma`, as for every external; SciPy called through NumPy assembles vectorised, which can cost less (Python comparison). Against libmatern (avx512) of the same job. ULP error as median / maximum over the finite results; cost in ns per element. Each external implementation is shown in its best assembly for each dataset, the one with the lowest maximum ULP error among those with the fewest non-finite results, direct on an exact tie. Results were obtained on the AMD CPU.

dataset	assembly	finite (%)	scipy med / max	lib med / max	scipy ns	lib ns	speed-up	rank
Grid	scaled‡	100	17.9 / 733	1.10 / 12.5	265	20.1	13.2×	4 of 6
Random	direct‡	100	4.90 / 1001	1.02 / 21.8	471	22.4	21.0×	5 of 6
Small z	direct	87.6	10.0 / 200†	0.483 / 10.4	401	19.2	20.8×	5 of 6
Near-integer ν	direct	100	11.5 / 186	0.864 / 17.7	456	17.6	25.8×	5 of 6
Half-integer ν	scaled	100	39.8 / 907	0.600 / 10.4	208	4.67	44.5×	4 of 6
Large z	scaled	100	125 / 1458	0.707 / 14.7	404	20.7	19.5×	4 of 6
Large ν	logtrick	67.3	175 / 2.04×10^4†	0.729 / 25.1	340	16.7	20.3×	3 of 6
Router seams	scaled	100	16.6 / 871	1.06 / 22.4	327	27.9	11.7×	4 of 6
Size / random, $N = 2^{25}$	logtrick	–	–	–	357	23.7⋆	15.1×	–
Size / half-integer, $N = 2^{25}$	logtrick	–	–	–	133	1.31⋆	101×	–

†: maximum over a reduced population, non-finite results excluded. *finite*: share of the points with a finite result in the shown assembly; the median and maximum are over those points. *speed-up*: SciPy's median cost over the library's; ≈ where their interquartile ranges overlap. *rank*: SciPy's position by median cost among the externals timed in the same job, each in its best assembly (bessels.jl, bessels.jl-old, boost, gsl, std). ‡: another assembly's maximum is within 1 ULP of the shown one (Grid: direct median 1.37 ULP, 340 ns; logtrick median 22.1 ULP, 279 ns; Random: scaled median 20.5 ULP, 387 ns; logtrick median 21.9 ULP, 396 ns). ⋆: from the library ladder of another run ('2026-10-30T03-23Z cpu'), not this job; on the accuracy datasets this job's library costs differ from that run's by -0.5 to +0.6 %. Size / random at $N = 2^{25}$: cost per element of the cumulative pass over the first N points, for both rows; Size / half-integer at $N = 2^{25}$: cost per element of the cumulative pass over the first N points, for both rows.

`tab-throughput-h_nu_s.tex`

Table 3: Single-threaded runtime cost per element at $N = 33\,554\,432$ for the size-scaling datasets with (h, ν, s) supplied and z computed. The cost columns give the cost in nanoseconds per element for the random and half-integer ν datasets, for the library rows of the `avx512` and `avx2` backends, which use a workspace, including its creation and destruction in each call; the ratio columns give the speedup of the `avx512` backend over each external implementation, computed from the corresponding costs. At this N a timed repetition is one pass made of four calls, over $[0, 2^{22})$, $[2^{22}, 2^{23})$, $[2^{23}, 2^{24})$ and $[2^{24}, 2^{25})$, with the clock read at the end, each call of the avx512 and avx2 backends creating and destroying its workspace (for an external implementation the pass is one loop over the same elements); the cost is the time of the pass divided by N. Below $N = 2^{22}$ the ladder times one call over the first N elements instead. On the random size-scaling dataset, at the switch, from $N = 2^{21}$ to 2^{22}, no series moves by more than 2 % (the largest change is bessels.jl-old (scaled), +0.7 %). This run times no N both ways, so it does not separate an effect of the timing mode from one of N. On the half-integer ν size-scaling dataset, at the switch, from $N = 2^{21}$ to 2^{22}, the cost of scalar rises by 16.3 %; it stays within 1 % from $N = 2^{19}$ to 2^{21} and from $N = 2^{22}$ to 2^{25}; every other series moves by at most 2 %. This run times no N both ways, so it does not separate an effect of the timing mode from one of N. Each cost is the median of the 25 timed repetitions of the pass over the first N points. *IQR*: the largest interquartile range of these repetitions over the datasets of the row, in per cent of the median. Results were obtained on the AMD CPU.

	ns / element		× this library			
implementation	Size / rand. 2^{25}	Size / half 2^{25}	Size / rand. 2^{25}	Size / half 2^{25}	med−min	IQR
libmatern (scalar)	175	7.56	–	–	3.6 %	0.58 %
libmatern (matched)	232	31.4	–	–	0.1 %	0.22 %
libmatern (matched_fma)	119	8.52	–	–	0.1 %	0.097 %
libmatern (avx2)	34.2	2.43	–	–	0.1 %	0.12 %
libmatern (avx512)	25.3	1.70	–	–	0.1 %	0.059 %
bessels.jl (scaled)	181	95.3	7.16×	55.9×	0.1 %	0.088 %
bessels.jl-old (scaled)	157	124	6.20×	72.5×	0.3 %	0.037 %
boost (exp-log)	949	538	37.5×	316×	0.0 %	0.011 %
gsl (exp-log)	245	189	9.69×	111×	0.0 %	0.018 %
std-cyl_bessel_k (exp-log)	306	195	12.1×	114×	0.0 %	0.018 %

med−min: largest excess of the median over the fastest repetition over the datasets of the row, in per cent. –: not applicable.

`tab-throughput-z_nu.tex`

Table 4: Single-threaded runtime cost per element at $N = 33\,554\,432$ for the size-scaling datasets with (z, ν) supplied. The cost columns give the cost in nanoseconds per element for the random and half-integer ν datasets, for the library rows of the `avx512` and `avx2` backends, which use a workspace, including its creation and destruction in each call; the ratio columns give the speedup of the `avx512` backend over each external implementation, computed from the corresponding costs. At this N a timed repetition is one pass made of four calls, over $[0, 2^{22})$, $[2^{22}, 2^{23})$, $[2^{23}, 2^{24})$ and $[2^{24}, 2^{25})$, with the clock read at the end, each call of the avx512 and avx2 backends creating and destroying its workspace (for an external implementation the pass is one loop over the same elements); the cost is the time of the pass divided by N. Below $N = 2^{22}$ the ladder times one call over the first N elements instead. On the random size-scaling dataset, at the switch, from $N = 2^{21}$ to 2^{22}, no series moves by more than 2 % (the largest change is bessels.jl-old (scaled), +0.9 %). This run times no N both ways, so it does not separate an effect of the timing mode from one of N. On the half-integer ν size-scaling dataset, at the switch, from $N = 2^{21}$ to 2^{22}, the cost of scalar rises by 9.0 % and that of matched_fma rises by 4.5 %; each stays within 1 % from $N = 2^{19}$ to 2^{21} and from $N = 2^{22}$ to 2^{25}; every other series moves by at most 2 %. This run times no N both ways, so it does not separate an effect of the timing mode from one of N. Each cost is the median of the 25 timed repetitions of the pass over the first N points. *IQR*: the largest interquartile range of these repetitions over the datasets of the row, in per cent of the median. Results were obtained on the AMD CPU.

	ns / element		× this library			
implementation	Size / rand. 2^{25}	Size / half 2^{25}	Size / rand. 2^{25}	Size / half 2^{25}	med−min	IQR
libmatern (scalar)	168	6.49	–	–	0.1 %	0.12 %
libmatern (matched)	211	15.0	–	–	0.1 %	0.028 %
libmatern (matched_fma)	112	7.69	–	–	3.5 %	0.027 %
libmatern (avx2)	31.8	2.02	–	–	0.1 %	0.067 %
libmatern (avx512)	23.7	1.31	–	–	0.1 %	0.15 %
bessels.jl (scaled)	180	94.1	7.60×	71.6×	0.1 %	0.022 %
bessels.jl-old (scaled)	156	119	6.57×	90.2×	0.0 %	0.030 %
boost (exp-log)	948	536	39.9×	408×	0.0 %	0.0074 %
gsl (exp-log)	244	188	10.3×	143×	0.0 %	0.030 %
std-cyl_bessel_k (exp-log)	304	193	12.8×	147×	0.1 %	0.053 %

med−min: largest excess of the median over the fastest repetition over the datasets of the row, in per cent. –: not applicable.