Package 'matsketch'

September 21, 2026

Title Randomized Matrix Computations from Few Entries and Products

Version 0.1.0

Description Implements recent randomized algorithms that answer questions about a large positive-semidefinite matrix while touching only a small part of it. Randomly pivoted Cholesky builds a low-rank approximation from a few rows of a kernel matrix (Chen, Epperly, Tropp and Webber (2025) <doi:10.1002/cpa.22234>), with an accelerated variant based on rejection sampling (Epperly, Tropp and Webber (2025) <doi:10.1137/24m1699048>). The XTrace, XNysTrace and XDiag estimators recover the trace and diagonal of a matrix that is available only through matrix-vector products (Epperly, Tropp and Webber (2024) <doi:10.1137/23m1548323>), alongside the Hutch++ estimator of Meyer, Musco, Musco and Woodruff (2021) <doi:10.1137/1.9781611976496.16>. Randomized Nystrom preconditioning speeds up the conjugate gradient method for regularized linear systems (Frangella, Tropp and Udell (2023) <doi:10.1137/21m1466244>). These pieces are combined to fit restricted maximum likelihood variance-component models on genomic relationship matrices without forming or factorizing the covariance matrix.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports graphics, grDevices, stats

Suggests knitr, Matrix, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/mqfarooqi1/matsketch, https://mqfarooqi1.github.io/matsketch/

BugReports https://github.com/mqfarooqi1/matsketch/issues

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Muhammad Farooqi [aut, cre] (ORCID:
<https://orcid.org/0000-0003-4918-9791>)

Maintainer Muhammad Farooqi <mqfarooqi@gmail.com>

Repository CRAN

Date/Publication 2026-09-21 17:20:02 UTC

Contents

`diag_est` *Stochastic diagonal estimation*

Description

Estimates the diagonal of a matrix that is available only through products AX, using the XDiag estimator, which applies the same low-rank-plus-correction and leave-one-out ideas as `trace_est()`.

Usage

```
diag_est(A, m, n = NULL, adjoint = NULL)
```

Arguments

`A`	A square matrix, a function computing AX for a matrix X, or a lazy kernel from `kernel_matrix()`.
`m`	Number of matrix-vector products to spend.
`n`	Dimension of A. Required only when `A` is a function.
`adjoint`	For a non-symmetric `A` given as a function, a function computing $A^\top X$. When omitted, `A` is assumed symmetric.

Value

A numeric vector of length n.

References

Epperly, E. N., Tropp, J. A. & Webber, R. J. (2024) XTrace: making the most of every sample in stochastic trace estimation. SIAM Journal on Matrix Analysis and Applications 45, 1-23. doi:10.1137/23m1548323

See Also

`trace_est()`

Examples

```
set.seed(1)
U <- qr.Q(qr(matrix(rnorm(200 * 200), 200)))
A <- U %*% diag((1:200)^-1.5) %*% t(U)
est <- diag_est(A, m = 60)
cor(est, diag(A))
```

---

effective_dim *Effective dimension and recommended preconditioner rank*

Description

Estimates $d_{\mathrm{eff}}(\mu) = \sum_j \lambda_j/(\lambda_j + \mu)$ from the eigenvalues of a low-rank approximation, and the rank $2\lceil 1.5\, d_{\mathrm{eff}} \rceil + 1$ that Frangella, Tropp and Udell show is enough for a well-conditioned preconditioner.

Usage

```
effective_dim(approx, mu)
```

Arguments

`approx`	An object from `nystrom()` or `rpchol()`.
`mu`	Positive regularization parameter.

Details

The estimate uses only the retained eigenvalues, so it can only understate the true effective dimension. If the recommended rank exceeds the rank of `approx`, build a larger approximation and ask again.

Value

A list with `d_eff`, `recommended_rank`, and `sufficient`, which is `TRUE` when `approx` already has at least the recommended rank.

See Also

nystrom_precond()

Examples

```
set.seed(1)
X <- matrix(rnorm(1000), ncol = 2)
effective_dim(nystrom(kernel_matrix(X), l = 60), mu = 1e-2)
```

---

grm_matrix *Lazy genomic relationship matrix*

Description

Describes the genomic relationship matrix of VanRaden (2008),

$$G = \frac{ZZ^\top}{2\sum_j f_j(1-f_j)},$$

where Z holds the marker genotypes centred by twice the allele frequencies f_j, without forming it. A product with G costs two products with Z, and rows are computed on demand, so the functions in this package can work with G while only the $n \times p$ genotypes are held in memory.

Usage

```
grm_matrix(M, freq = NULL)
```

Arguments

M	Genotype matrix with one row per individual and one column per marker, coded as allele counts between 0 and 2. Missing genotypes must be imputed first.
freq	Allele frequencies used for centring. Defaults to the observed frequencies, colMeans(M) / 2.

Details

This matters once n is large: for 50,000 individuals the full matrix takes 20 GB, while the genotypes on a 10,000-marker panel take 4 GB.

Value

An object of class matsketch_grm, accepted wherever matsketch expects a matrix. as.matrix() forms the full matrix.

References

VanRaden, P. M. (2008) Efficient methods to compute genomic predictions. Journal of Dairy Science 91, 4414-4423. doi:10.3168/jds.20070980

See Also

reml_sketch(), sim_genomic()

Examples

```
set.seed(1)
M <- matrix(rbinom(200 * 500, 2, 0.3), 200)
G <- grm_matrix(M)
G
G$block(1:3, 1:3)
```

---

kernel_matrix *Lazy kernel matrix*

Description

Describes the kernel matrix K with entries $k(x_i, x_j)$ without computing it. Randomly pivoted Cholesky then evaluates only the entries it needs, roughly $(k+1)n$ of the n^2, which is what makes it practical when the full kernel matrix would not fit in memory.

Usage

```
kernel_matrix(
  X,
  kernel = c("gaussian", "laplace", "matern32", "matern52"),
  bandwidth = NULL,
  block = 1000L
)
```

Arguments

X	Numeric matrix with one row per point.
kernel	Kernel family: "gaussian" $\exp(-r^2/2h^2)$, "laplace" $\exp(-r/h)$, or the Matern kernels "matern32" and "matern52", where r is the Euclidean distance.
bandwidth	Length scale h. Defaults to the median distance between points, computed on at most the first 500 rows.
block	Number of rows computed at a time when multiplying by the whole matrix, which bounds memory use.

Details

Each product with the whole matrix recomputes the kernel, block by block. For methods that multiply many times, such as pcg(), it is faster to form the matrix once with as.matrix() whenever it fits in memory.

Value

An object of class `matsketch_kernel`, accepted wherever `matsketch` expects a matrix. `as.matrix()` forms the full matrix.

See Also

`rpchol()`, `grm_matrix()`

Examples

```
X <- matrix(rnorm(400), ncol = 2)
K <- kernel_matrix(X, "gaussian")
K
K$block(1:3, 1:3)
```

`nystrom` *Randomized Nystrom approximation*

Description

Computes a rank-ℓ approximation $A \approx U\hat{\Lambda}U^\top$ of a positive-semidefinite matrix from ℓ products with a random test matrix. A tiny shift keeps the Cholesky step stable and is removed from the eigenvalues afterwards.

Usage

```
nystrom(A, l, n = NULL)
```

Arguments

`A`	A positive-semidefinite matrix, a function computing AX, or a lazy matrix from `kernel_matrix()` or `grm_matrix()`.
`l`	Rank of the approximation, which is also the number of products used.
`n`	Dimension of A. Required only when `A` is a function.

Value

An object of class `nystrom` with the orthonormal eigenvectors `U` and eigenvalues `values`, in decreasing order.

References

Frangella, Z., Tropp, J. A. & Udell, M. (2023) Randomized Nystrom preconditioning. SIAM Journal on Matrix Analysis and Applications 44, 718-752. doi:10.1137/21m1466244

See Also

`nystrom_precond()`, `rpchol()`

Examples

```
set.seed(1)
X <- matrix(rnorm(600), ncol = 3)
nys <- nystrom(kernel_matrix(X), l = 30)
head(nys$values)
```

`nystrom_precond` *Nystrom preconditioner*

Description

Builds the preconditioner for the regularized system $(A+\mu I)x = b$ from a low-rank approximation $A \approx U\hat{\Lambda}U^\top$:

$$P^{-1} = (\hat{\lambda}_\ell + \mu)U(\hat{\Lambda} + \mu I)^{-1}U^\top + (I - UU^\top),$$

where $\hat{\lambda}_\ell$ is the smallest retained eigenvalue.

Usage

```
nystrom_precond(approx, mu)
```

Arguments

`approx`	An object from `nystrom()` or `rpchol()`.
`mu`	Positive regularization parameter.

Details

The preconditioned system has a small condition number once the rank reaches about the effective dimension $d_{\text{eff}}(\mu) = \operatorname{tr}(A(A+\mu I)^{-1})$: Frangella, Tropp and Udell show that a rank of $2\lceil 1.5\, d_{\text{eff}}(\mu)\rceil + 1$ keeps the expected condition number below 28, whatever the size of the matrix. `effective_dim()` estimates that rank.

The approximation can come from `nystrom()` or from `rpchol()`. Because only μ enters the formula after the approximation is built, one approximation serves any number of values of μ, which is what makes it cheap to re-use inside an iterative fit.

Value

An object of class `nystrom_precond`, to pass to `pcg()`.

References

Frangella, Z., Tropp, J. A. & Udell, M. (2023) Randomized Nystrom preconditioning. SIAM Journal on Matrix Analysis and Applications 44, 718-752. doi:10.1137/21m1466244

See Also

`pcg()`, `effective_dim()`

Examples

```
set.seed(1)
X <- matrix(rnorm(1000), ncol = 2)
K <- kernel_matrix(X)
pre <- nystrom_precond(nystrom(K, l = 40), mu = 1e-3)
pre
```

---

pcg *Preconditioned conjugate gradients*

Description

Solves $(A + \mu I)x = b$ for a positive-semidefinite A using only products with A. Without a preconditioner this is the ordinary conjugate gradient method, whose iteration count grows with the condition number; with a preconditioner from `nystrom_precond()` the count stays small and nearly independent of the size of the problem.

Usage

```
pcg(A, b, mu = 0, precond = NULL, tol = 1e-08, maxit = 1000L, x0 = NULL)
```

Arguments

`A`	A positive-semidefinite matrix, a function computing AX, or a lazy matrix from `kernel_matrix()` or `grm_matrix()`. With `mu = 0`, A must be positive definite.
`b`	Right-hand side: a vector, or a matrix with one system per column.
`mu`	Non-negative regularization parameter.
`precond`	`NULL` for no preconditioning, an object from `nystrom_precond()`, or a function applying the inverse preconditioner to a matrix of residuals, column by column.
`tol`	Stop each system when its residual norm falls below `tol` times the norm of its right-hand side.
`maxit`	Maximum number of iterations.
`x0`	Optional starting value, the same shape as `b`.

Details

When `b` is a matrix, each column is solved as a separate system, but all of them advance together, so every step multiplies A by a block of vectors. That is much faster in R than solving the columns one at a time.

Value

An object of class `pcg_result` with the solution `x` (the same shape as `b`), the `iterations` and whether each system `converged`, and the relative `residuals` after each iteration.

References

Frangella, Z., Tropp, J. A. & Udell, M. (2023) Randomized Nystrom preconditioning. SIAM Journal on Matrix Analysis and Applications 44, 718-752. doi:10.1137/21m1466244

See Also

`nystrom_precond()`

Examples

```
set.seed(1)
X <- matrix(rnorm(1000), ncol = 2)
K <- as.matrix(kernel_matrix(X))
b <- rnorm(500)
plain <- pcg(K, b, mu = 1e-3)
pre <- nystrom_precond(rpchol(K, k = 60), mu = 1e-3)
fast <- pcg(K, b, mu = 1e-3, precond = pre)
c(plain = plain$iterations, preconditioned = fast$iterations)
```

---

plot.pcg_result *Plot the convergence of a conjugate gradient solve*

Description

Draws the relative residual after each iteration on a logarithmic scale, one line per system solved.

Usage

```
## S3 method for class 'pcg_result'
plot(x, tol = NULL, ...)
```

Arguments

`x`	An object from `pcg()`.
`tol`	Optional tolerance to mark with a horizontal line.
`...`	Passed to `graphics::matplot()`.

Value

`x`, invisibly.

Examples

```
set.seed(1)
X <- matrix(rnorm(1000), ncol = 2)
K <- as.matrix(kernel_matrix(X))
plot(pcg(K, rnorm(500), mu = 1e-3), tol = 1e-8)
```

plot.reml_sketch *Plot the path of a REML fit*

Description

Draws the genetic and residual variance estimates at each iteration, which shows whether the fit settled or was still moving when it stopped.

Usage

```
## S3 method for class 'reml_sketch'
plot(x, ...)
```

Arguments

x	An object from reml_sketch() or reml_exact().
...	Passed to graphics::matplot().

Value

x, invisibly.

Examples

```
set.seed(1)
dat <- sim_genomic(n = 200, p = 400, h2 = 0.5)
plot(reml_exact(dat$y, dat$G))
```

plot.rpchol *Plot the error of a randomly pivoted Cholesky approximation*

Description

Draws the relative trace error after each pivot on a logarithmic scale, which shows how quickly the approximation improves with its rank.

Usage

```
## S3 method for class 'rpchol'
plot(x, ...)
```

Arguments

x	An object from rpchol().
...	Passed to graphics::plot().

Value

x, invisibly.

Examples

```
set.seed(1)
X <- matrix(rnorm(1000), ncol = 2)
plot(rpchol(kernel_matrix(X), k = 60))
```

---

reml_exact *Variance components by exact dense REML*

Description

Fits the same model as reml_sketch() with the same average-information updates, but forms and factorizes V and computes every trace exactly. It is meant for checking reml_sketch() on problems small enough to factorize.

Usage

```
reml_exact(y, G, X = NULL, start = NULL, tol = 1e-08, maxit = 100L)
```

Arguments

y	Numeric response.
G	A positive-semidefinite relationship matrix, or a lazy matrix from grm_matrix() or kernel_matrix(), which is formed in full.
X	Fixed-effect design matrix. Defaults to an intercept.
start	Starting values c(genetic, residual). Defaults to half the residual variance of a least-squares fit for each.
tol	Stop when no variance component changes by more than this fraction between iterations.
maxit	Maximum number of REML iterations.

Value

An object of class reml_sketch whose approx is "exact".

References

Gilmour, A. R., Thompson, R. & Cullis, B. R. (1995) Average information REML: an efficient algorithm for variance parameter estimation in linear mixed models. Biometrics 51, 1440-1450. doi:10.2307/2533274

See Also

reml_sketch()

Examples

```
set.seed(1)
dat <- sim_genomic(n = 300, p = 600, h2 = 0.5)
reml_exact(dat$y, dat$G)
```

`reml_sketch` *Variance components by sketched REML*

Description

Fits $y = X\beta + g + e$, with $g \sim N(0, \sigma_g^2 G)$ and $e \sim N(0, \sigma_e^2 I)$, by average-information REML, without forming or factorizing the covariance matrix $V = \sigma_g^2 G + \sigma_e^2 I$.

Usage

```
reml_sketch(
  y,
  G,
  X = NULL,
  rank = 100L,
  m = 40L,
  approx = c("auto", "rpchol", "nystrom"),
  estimator = c("xtrace", "hutchinson"),
  start = NULL,
  tol = 1e-04,
  maxit = 50L,
  cg_tol = 1e-06
)
```

Arguments

`y`	Numeric response.
`G`	Relationship or kernel matrix: a positive-semidefinite matrix, a lazy matrix from `grm_matrix()` or `kernel_matrix()`, or a function computing GX for a matrix X.
`X`	Fixed-effect design matrix. Defaults to an intercept.
`rank`	Rank of the approximation of `G` used as a preconditioner.
`m`	Matrix-vector products per trace estimate. Each costs one linear solve at every iteration.
`approx`	How to approximate `G`. `"auto"` uses `rpchol()` when entries of `G` can be read and `nystrom()` when `G` is a function.
`estimator`	Trace estimator, `"xtrace"` or `"hutchinson"`. XTrace is much more accurate when PG has a few dominant eigenvalues, as under population structure; when the spectrum is flat the two are close.

`start`	Starting values `c(genetic, residual)`. Defaults to half the residual variance of a least-squares fit for each.
`tol`	Stop when no variance component changes by more than this fraction between iterations.
`maxit`	Maximum number of REML iterations.
`cg_tol`	Relative residual tolerance for each linear solve.

Details

Three ideas from the package do the work.

- **Solves.** A system in V is the system $(G + \mu I)x = b/\sigma_g^2$ with $\mu = \sigma_e^2/\sigma_g^2$. It is solved by conjugate gradients, preconditioned with a low-rank approximation of G that is built once, by `rpchol()` or `nystrom()`, and reused for every value of μ the fit visits. The right-hand sides needed at each step are solved together, each starting from its solution at the previous step.
- **One stochastic trace.** The score needs $\mathrm{tr}(PG)$, which is estimated by XTrace or by Hutchinson's estimator. The test matrix is drawn once and kept for the whole fit, so successive iterations see the same sketch and converge smoothly rather than jittering.
- **One exact trace.** PV is idempotent with rank $n - \mathrm{rank}(X)$, so $\mathrm{tr}(P) = (n - \mathrm{rank}(X) - \sigma_g^2 \mathrm{tr}(PG))/\sigma_e^2$ holds exactly, and the second trace costs nothing.

Here $P = V^{-1} - V^{-1}X(X^\top V^{-1} X)^{-1} X^\top V^{-1}$. The average-information matrix needs only quadratic forms in P, and these are computed from solves rather than estimated.

Because the test matrix is fixed, the fit converges to the exact solution of a slightly perturbed set of REML equations. The `trace_se` column of `history` shows the size of the perturbation, which shrinks as `m` grows. `reml_exact()` fits the same model by dense linear algebra, for checking results on problems small enough to factorize.

Value

An object of class `reml_sketch` with the variance components `sigma2`, the heritability `h2`, their standard errors `se`, the fixed effects `beta`, the iteration `history`, the number of linear systems solved `solves`, and the total conjugate gradient iterations `cg_iterations` they took.

References

Gilmour, A. R., Thompson, R. & Cullis, B. R. (1995) Average information REML: an efficient algorithm for variance parameter estimation in linear mixed models. Biometrics 51, 1440-1450. doi:10.2307/2533274

Bermann, M., Legarra, A., Aguilar, I., Alvarez-Munera, A., Misztal, I. & Lourenco, D. (2025) Estimation of (co)variance components for very large datasets and complex single-step genomic models. Genetics Selection Evolution 57. doi:10.1186/s12711025010069

See Also

`reml_exact()`, `grm_matrix()`, `sim_genomic()`

Examples

```
set.seed(1)
dat <- sim_genomic(n = 300, p = 600, h2 = 0.5)
fit <- reml_sketch(dat$y, dat$G, rank = 60, m = 30)
fit

# the relationship matrix need never be formed
reml_sketch(dat$y, grm_matrix(dat$M), rank = 60, m = 30)$h2
```

---

rpchol *Randomly pivoted Cholesky*

Description

Builds a rank-k approximation $A \approx FF^\top$ of a positive-semidefinite matrix by choosing k pivot columns at random, each with probability proportional to the diagonal of the part of A not yet explained.

Usage

```
rpchol(
  A,
  k,
  method = c("accelerated", "simple", "greedy", "uniform"),
  block = NULL,
  tol = 0
)
```

Arguments

`A`	A symmetric positive-semidefinite matrix, or a lazy matrix from `kernel_matrix()` or `grm_matrix()`.
`k`	Target rank.
`method`	Pivoting rule; see Details.
`block`	Proposals per round for the accelerated method. Defaults to `max(10, ceiling(k / 10))`.
`tol`	Stop early once the unexplained trace falls below `tol` times the trace of `A`. A floor of `1e-13` always applies: below it the residual is rounding error, and further pivots would be chosen from noise.

Details

Sampling in proportion to the residual diagonal is what separates the method from its predecessors. Greedy pivoting always takes the largest residual and can fixate on outliers; uniform sampling ignores where the matrix actually has mass. Randomly pivoted Cholesky balances the two, and

reaches near-optimal approximations while reading only about $(k+1)n$ entries of the matrix, so it never needs A in full.

Four pivoting rules are available:

- `"accelerated"` (default) proposes a block of pivots at once and accepts each by rejection sampling. Its output has the same distribution as `"simple"`, but most of its work is done in block operations, which is much faster when k is large.
- `"simple"` draws one pivot at a time in proportion to the residual diagonal.
- `"greedy"` always takes the largest residual diagonal, which is the classical pivoted partial Cholesky decomposition.
- `"uniform"` takes pivots uniformly at random, which gives the column-sampling Nystrom approximation.

The last two are included as baselines, so that the gain from random pivoting can be measured on the problem at hand.

The relative trace error reported is exact, not estimated: the residual $A - FF^\top$ is positive semidefinite, so its trace norm is the sum of the residual diagonal the algorithm maintains anyway.

Value

An object of class `rpchol` containing the factor `F`, whose number of columns is the rank achieved; the chosen `pivots`; the relative trace error `trace_error`, and `trace_path`, its value after each pivot; and `entries`, the number of matrix entries read.

References

Chen, Y., Epperly, E. N., Tropp, J. A. & Webber, R. J. (2025) Randomly pivoted Cholesky: practical approximation of a kernel matrix with few entry evaluations. Communications on Pure and Applied Mathematics 78, 995-1041. doi:10.1002/cpa.22234

Epperly, E. N., Tropp, J. A. & Webber, R. J. (2025) Embrace rejection: kernel matrix approximation by accelerated randomly pivoted Cholesky. SIAM Journal on Matrix Analysis and Applications 46, 2527-2557. doi:10.1137/24m1699048

See Also

`kernel_matrix()`, `nystrom_precond()`

Examples

```
set.seed(1)
X <- matrix(rnorm(1000), ncol = 2)
K <- kernel_matrix(X)
fit <- rpchol(K, k = 40)
fit

# the same budget spent on greedy or uniform pivots
rpchol(K, k = 40, method = "greedy")$trace_error
rpchol(K, k = 40, method = "uniform")$trace_error
```

`sim_genomic` *Simulate a genomic data set*

Description

Draws biallelic marker genotypes, forms the genomic relationship matrix of VanRaden (2008), and simulates a trait with the requested heritability, so that `reml_sketch()` can be checked against known variance components.

Usage

```
sim_genomic(
  n = 1000L,
  p = 2000L,
  h2 = 0.5,
  pops = 1L,
  fst = 0.05,
  form_G = TRUE
)
```

Arguments

`n`	Number of individuals.
`p`	Number of markers.
`h2`	Heritability, between 0 and 1.
`pops`	Number of subpopulations.
`fst`	Fixation index between subpopulations, used when `pops > 1`.
`form_G`	Return the relationship matrix itself. Set to `FALSE` for large `n` and pass `grm_matrix(M)` to the fitting functions instead.

Details

With `pops > 1` the individuals come from that many subpopulations whose allele frequencies have drifted apart under the Balding-Nichols model with fixation index `fst`. Population structure gives the relationship matrix a few large eigenvalues, as real breeding and human cohorts do, and those are exactly what a low-rank preconditioner captures.

Genetic values are sums of marker effects on the centred genotypes, with variance chosen so that the genetic and residual variance components on the scale of the relationship matrix are `h2` and `1 - h2`.

The function draws random numbers but does not set the seed; call `base::set.seed()` first for a reproducible data set.

Value

A list with the phenotype `y`, an intercept design matrix `X`, the genotype matrix `M`, the relationship matrix `G` (when `form_G = TRUE`), the subpopulation of each individual `pop`, and the true `h2`.

References

VanRaden, P. M. (2008) Efficient methods to compute genomic predictions. Journal of Dairy Science 91, 4414-4423. doi:10.3168/jds.20070980

Balding, D. J. & Nichols, R. A. (1995) A method for quantifying differentiation between populations at multi-allelic loci and its implications for investigating identity and paternity. Genetica 96, 3-12. doi:10.1007/bf01441146

See Also

`grm_matrix()`, `reml_sketch()`

Examples

```
set.seed(1)
dat <- sim_genomic(n = 200, p = 500, h2 = 0.4, pops = 3)
dim(dat$G)
table(dat$pop)
```

`trace_est` *Stochastic trace estimation*

Description

Estimates $\mathrm{tr}(A)$ for a matrix that is available only through products AX, spending about `m` such products.

Usage

```
trace_est(
  A,
  m,
  method = c("xtrace", "xnystrace", "hutchpp", "hutchinson"),
  n = NULL
)
```

Arguments

`A`	A square matrix, a function computing AX for a matrix X, or a lazy kernel from `kernel_matrix()`.
`m`	Number of matrix-vector products to spend.
`method`	Estimator; see Details.
`n`	Dimension of A. Required only when `A` is a function.

Details

- `"xtrace"` (default) combines a low-rank approximation of A with a correction for what it misses, and uses every product twice through a leave-one-out construction. It is typically far more accurate than the older estimators at the same cost, and it reports its own standard error. It works for any square matrix.
- `"xnystrace"` is the counterpart for positive-semidefinite matrices. It uses a Nystrom approximation, which lets it spend all `m` products on a single sketch.
- `"hutchpp"` is Hutch++, the estimator XTrace improves on.
- `"hutchinson"` is the classical Girard-Hutchinson estimator, the average of $\omega^\top A\omega$ over random sign vectors. It is included as the baseline the others are measured against.

Value

An object of class `trace_est` holding the `estimate`, its `std_error` (not available for Hutch++), the `method`, and the number of products actually used, `matvecs`.

References

Epperly, E. N., Tropp, J. A. & Webber, R. J. (2024) XTrace: making the most of every sample in stochastic trace estimation. SIAM Journal on Matrix Analysis and Applications 45, 1-23. doi:10.1137/23m1548323

Meyer, R. A., Musco, C., Musco, C. & Woodruff, D. P. (2021) Hutch++: optimal stochastic trace estimation. Symposium on Simplicity in Algorithms, 142-155. doi:10.1137/1.9781611976496.16

Hutchinson, M. F. (1989) A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. Communications in Statistics - Simulation and Computation 18, 1059-1076. doi:10.1080/03610918908812806

See Also

`diag_est()`

Examples

```
set.seed(1)
U <- qr.Q(qr(matrix(rnorm(300 * 300), 300)))
A <- U %*% diag((1:300)^-2) %*% t(U)
sum(diag(A))
trace_est(A, m = 40)
trace_est(A, m = 40, method = "hutchinson")
```

Index