# Package: nmathopencl (via r-universe)

June 5, 2026

**Type** Package

**Title** 'OpenCL'-Ported R 'Mathlib' for GPU-Accelerated Packages

**Version** 0.8.1

**Date** 2026-05-30

**Description** Ships statistical and mathematical routines from R internal 'nmath' ('Mathlib') as 'OpenCL' C sources under directory 'inst/cl/', with R wrappers that use the GPU when 'OpenCL' is available at compile time and fall back to 'stats' equivalents otherwise. Aimed at package developers building custom kernels (for example Bayesian GLMs via suggested package 'glmbayes') using 'opencltools' kernel loaders and related helpers. Contains translated shims, an illustrative GLM-related kernel subsystem, vignettes, and optional GPU acceleration.

**License** GPL-2

**URL** https://github.com/knygren/nmathopencl, https://knygren.r-universe.dev/nmathopencl

**BugReports** https://github.com/knygren/nmathopencl/issues

**Imports** stats, Rcpp (>= 1.1.1), RcppParallel, Rdpack (>= 0.11-0), opencltools (>= 0.8.1)

**RdMacros** Rdpack

**LinkingTo** Rcpp, RcppArmadillo, RcppParallel, opencltools

**Depends** MASS, R (>= 3.5.0)

**Suggests** glmbayes (>= 0.9.3), knitr, rmarkdown, testthat (>= 3.0.0), spelling

**SystemRequirements** Optional 'OpenCL' support. If available, GPU acceleration will be used; otherwise, computation runs on CPU.

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**LazyData** false

**Language** en-US

**Config/pak/sysreqs** make ocl-icd-opencl-dev

**Repository** https://knygren.r-universe.dev

**Date/Publication** 2026-06-05 17:50:41 UTC

**RemoteUrl** https://github.com/knygren/nmathopencl

**RemoteRef** HEAD

**RemoteSha** aee5bcdbe8a8f48db3a40bb4cebd67470ea5776e

# Contents

nmathopencl-package . . . 3
attach_cross_library_tags . . . 5
attach_kernel_call_tags . . . 7
attach_kernel_dependency_tags . . . 10
besselI_opencl . . . 11
dbeta_opencl . . . 12
dbinom_raw_opencl . . . 14
dcauchy_opencl . . . 17
dchisq_opencl . . . 18
dexp_opencl . . . 20
df_opencl . . . 22
dgamma_opencl . . . 24
dgeom_opencl . . . 26
dhyper_opencl . . . 28
dlnorm_opencl . . . 30
dlogis_opencl . . . 31
dnbinom_opencl . . . 33
dnorm_opencl . . . 36
dpois_raw_opencl . . . 38
dsignrank_opencl . . . 40
dt_opencl . . . 42
dunif_opencl . . . 44
dweibull_opencl . . . 46
dwilcox_opencl . . . 47
Ex_EnvelopeEval . . . 49
Ex_EnvelopeSize . . . 53
Ex_glmb_Standardize_Model . . . 55
Ex_glmbfamfunc . . . 56
extract_library_subset . . . 57
gammafn_opencl . . . 59
glmbayesEnvelopeExample . . . 61
imax2_opencl . . . 61
load_library_for_kernel . . . 63

nmathopencl_has_opencl . . . . . 65
norm_rand_opencl . . . . . 67
port_to_opencl_configure . . . . . 68
ptukey_opencl . . . . . 70
r_check_user_interrupt_opencl . . . . . 71
r_pow_opencl . . . . . 72
rmultinom_opencl . . . . . 74
stage_kernel_dependency_sort . . . . . 75
use_opencl_configure . . . . . 76
write_kernel_dependency_index . . . . . 77

**Index** **80**

---

`nmathopencl-package` *nmathopencl: OpenCL-Ported R Math Library for GPU-Accelerated Packages*

---

## Description

**nmathopencl** provides OpenCL-ported versions of R's internal `nmath` and `R_ext` math routines, enabling downstream R packages to build custom GPU-accelerated kernels that call the same statistical distribution functions available in base R. The package is intended as a **developer library**: users install it to gain access to the ported `.cl` source files, then write their own OpenCL kernels that `#include` those sources as needed.

## Details

The core deliverable is a collection of `.cl` files installed under `inst/cl/nmath/` that mirror the R `nmath` library (density, distribution, quantile, and random-variate functions). Downstream packages locate these files at runtime with `system.file("cl", package = "nmathopencl")` and assemble them into an OpenCL program using `opencltools::load_kernel_library(..., package = "nmathopencl")`.

The package also ships `Ex_EnvelopeEval` and its supporting functions (`Ex_glmbfamfunc`, `Ex_glmb_Standardize_Model`, `Ex_EnvelopeSize`) as a worked example of how a downstream package—here the **glmbayes** Bayesian GLM sampler—builds a custom kernel on top of the ported nmath routines. See `system.file("examples", "Ex_EnvelopeEval.R", package = "nmathopencl")` and `vignette("Chapter-10")` for a complete walkthrough.

Optional GPU acceleration is available wherever an OpenCL runtime is installed. Use `nmathopencl_has_opencl` to query compile-time OpenCL support, `nmathopencl_opencl_fp64_available` / `nmathopencl_opencl_device_info` for double-precision device selection used by kernels. Host/runtime diagnostics use `opencltools::diagnose_glmbayes()`.

The simulation theory underlying the envelope construction is described in (Nygren and Nygren 2006), with implementation details in (Nygren 2025, 2025). OpenCL GPU execution follows (Stone et al. 2010); package vignettes also discuss GPU workflows ((Nygren 2025, 2025)).

### C-callable API (C++ package developers)

GPU `*_opencl` routines are registered for `R_GetCCallable` on load. Downstream packages should `LinkingTo: nmathopencl, Imports: nmathopencl, opencltools`, and `#include <nmathopencl/nmathopencl_capi.h>` (see `system.file("include/nmathopencl/README.md", package = "nmathopencl")`). Call `nmathopencl_api_versior` for ABI compatibility. Kernel loading remains on **opencltools** (`opencltools_capi.h`).

### OpenCL startup checks

In interactive sessions, attaching the package with `library(nmathopencl)` may emit a `packageStartupMessage` comparing compile-time OpenCL support in **nmathopencl** and **opencltools**, noting that CPU fallbacks remain available, and summarizing whether an OpenCL runtime appears available on the host. Messages point to `?gpu_diagnostics`, `vignette("Chapter-01")` (OpenCL enablement), `vignette("Chapter-12")` (packaged `*_opencl` API), and this help page. Host-side OpenCL diagnostics use **opencltools**. Set `options(nmathopencl.quiet_opencl_startup = TRUE)` to suppress these notes (recommended for CI and `R CMD check`).

### Author(s)

Kjell Nygren

### References

Nygren K (2025). "Chapter A05: Simulation Methods – Likelihood Subgradient Densities." Vignette in the glmbayes R package. R vignette name: Chapter-A05.

Nygren K (2025). "Chapter A08: Overview of Envelope Related Functions." Vignette in the glmbayes R package. R vignette name: Chapter-A08.

Nygren K (2025). "Chapter 12: Large Models: GPU Acceleration using OpenCL." Vignette in the glmbayes R package. R vignette name: Chapter-12.

Nygren K (2025). "Chapter A10: Accelerated EnvelopeBuild Implementation using OpenCL." Vignette in the glmbayes R package. R vignette name: Chapter-A10.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

Stone JE, Gohara D, Shi G (2010). "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems." *Computing in Science & Engineering*, **12**(3), 66–72. doi:10.1109/MCSE.2010.69.

### See Also

Key developer entry points:

- `opencltools::load_kernel_library` — assemble the nmath `.cl` sources (`package = "nmathopencl"`).
- `nmathopencl_has_opencl` — check whether an OpenCL runtime is present.
- `nmathopencl_opencl_device_info` — which OpenCL device is used for fp64 kernels.

- Ex_EnvelopeEval — worked example of a custom kernel built on the ported nmath routines.

Useful links:

- GitHub: https://github.com/knygren/nmathopencl
- R-Universe: https://knygren.r-universe.dev/nmathopencl
- Related sampler package (Suggests): **glmbayes**

---

`attach_cross_library_tags` *Attach Cross-Library Dependency Tags to Kernel Files*

---

## Description

Given a set of user-facing kernel `.cl` files and a library directory, computes the full transitive dependency list for each kernel (using the library's pre-built dependency index) and writes the results back into the kernel files as annotation tags.

## Usage

```
attach_cross_library_tags(
  kernel_paths,
  library_dir,
  depends_tag = "depends_nmath",
  index = NULL,
  dry_run = FALSE
)
```

## Arguments

| | |
|---|---|
| `kernel_paths` | Character vector of paths to kernel `.cl` files. |
| `library_dir` | Path to the library directory containing `kernel_dependency_index.rds` and the library `.cl` files. |
| `depends_tag` | Name of the annotation tag in the kernel files that lists the direct library entry-point stems (e.g. `"depends_nmath"`). The function reads `@{depends_tag}` and writes `@all_{depends_tag}` and `@all_{depends_tag}_count`. |
| `index` | Optional dependency index (write_kernel_dependency_index, load via `readRDS`). If `NULL`, reads `file.path(library_dir, "kernel_dependency_index.rds")` with `message()`. |
| `dry_run` | Logical; if `TRUE`, compute tags but do not write any files. |

## Details

Cross-library analogue of attach_kernel_dependency_tags: it targets kernels that depend on a library through a `@{depends_tag}` tag (entry-point stems), instead of expanding `@depends` purely inside one library tree.

Typical usage for kernels that call nmath functions:

```
nmath_dir <- system.file(
  "cl", "nmath", package = "opencltools")
idx <- readRDS(file.path(nmath_dir, "kernel_dependency_index.rds"))

attach_cross_library_tags(
  kernel_paths = list.files("inst/cl/src", "\\\\.cl$", full.names = TRUE),
  library_dir  = nmath_dir,
  depends_tag  = "depends_nmath",
  index        = idx
)
```

This writes `@all_depends_nmath_count` and `@all_depends_nmath` into each kernel file that carries a `@depends_nmath` annotation.

## Value

A data frame (returned invisibly) with one row per kernel file and columns:

`file` Basename of the kernel file.

`direct_stems` Comma-separated direct entry-point stems read from `@{depends_tag}`.

`all_depends_count` Number of library files in the full transitive closure.

`all_depends` Comma-separated full transitive dependency list in load order.

`changed` `TRUE` if the file was (or would be, under `dry_run`) modified.

## See Also

attach_kernel_dependency_tags write_kernel_dependency_index load_library_for_kernel

## Examples

```
############################ Start of kernel_tagging_workflow example ########################

lib_dir <- system.file("cl/ex_glmbayes_nmath", package = "opencltools")
kernels <- list.files(
  system.file("cl/ex_glmbayes_draft_src", package = "opencltools"),
  pattern = "\\.cl$", full.names = TRUE
)

# Step 1: scan draft kernels for library calls (read-only dry run)
step1 <- attach_kernel_call_tags(
  kernel_paths = kernels,
  library_dir  = lib_dir,
```

```
  library_tag  = "nmath",
  dry_run      = TRUE
)
step1

# Step 2: expand transitive closure (small nmath library; runs on CRAN check)
nmath_small <- system.file("cl/nmath_small", package = "opencltools")
tagged      <- system.file("cl/src/dnorm_kernel.cl", package = "opencltools")
idx_small   <- write_kernel_dependency_index(library_dir = nmath_small, write = FALSE)

step2_small <- attach_cross_library_tags(
  kernel_paths = tagged,
  library_dir  = nmath_small,
  depends_tag  = "depends_nmath",
  index        = idx_small,
  dry_run      = TRUE
)
nrow(step2_small)


# Step 2: full nmath (slow)
nmath_dir <- system.file("cl/nmath", package = "opencltools")
idx       <- write_kernel_dependency_index(library_dir = nmath_dir, write = FALSE)

step2 <- attach_cross_library_tags(
  kernel_paths = tagged,
  library_dir  = nmath_dir,
  depends_tag  = "depends_nmath",
  index        = idx,
  dry_run      = TRUE
)
step2


################################################################################
## End of kernel_tagging_workflow example
################################################################################
```

---

`attach_kernel_call_tags` *Attach Library Call Tags to Kernel Files*

---

## Description

Scans kernel `.cl` source files for calls to functions provided by a pre-annotated library, then writes the discovered dependencies as annotation tags directly into the kernel files.

## Usage

```
attach_kernel_call_tags(
```

```
  kernel_paths,
  library_dir,
  library_tag,
  overwrite_existing = FALSE,
  dry_run = FALSE
)
```

## Arguments

| | |
|---|---|
| kernel_paths | Character vector of paths to kernel .cl files. |
| library_dir | Path to the pre-annotated library directory. Each .cl file in this directory must carry a @provides annotation listing the symbols it exports. |
| library_tag | String tag suffix used for annotation names, e.g. "nmath". Must not contain spaces or regex special characters. |
| overwrite_existing | Logical; if FALSE (default), skip files that already carry a @calls_<library_tag> annotation. Set to TRUE to re-scan and overwrite (also clears any existing @all_depends_<tag> so attach_cross_library_tags re-computes it cleanly). |
| dry_run | Logical; if TRUE, compute tags but do not write any files. |

## Details

For a library such as **nmathopencl**, each .cl shard carries a @provides annotation listing the symbols it defines. This function builds a **provides map** from those annotations (symbol → shard stem), scans each kernel's source (with comments and string literals stripped) for matching function calls, and writes four annotation tags at the top of each kernel file:

@library_deps: <library_tag> Library name (written if absent).

@calls_<library_tag>: sym1, sym2 Symbols from the library actually called in this kernel.

@depends_<library_tag>: stem1, stem2 Library shard stems that define the called symbols.

@calls_opencl_builtin: sym | (none) Detected OpenCL work-item and synchronization builtins (standard math builtins excluded).

### Two-step tagging workflow:

```
# Step 1 — this function: infer direct library calls from source
nmath_dir <- system.file("cl/nmath", package = "nmathopencl")
attach_kernel_call_tags(
  kernel_paths = list.files("inst/cl/src", "\\.cl$", full.names = TRUE),
  library_dir  = nmath_dir,
  library_tag  = "nmath"
)

# Step 2 — expand to full transitive closure
attach_cross_library_tags(
  kernel_paths = list.files("inst/cl/src", "\\.cl$", full.names = TRUE),
  library_dir  = nmath_dir,
  depends_tag  = "depends_nmath"
)
```

## Value

A data frame (returned invisibly) with one row per kernel file and columns:

`file` Basename of the kernel file.

`calls` Comma-separated library symbols detected in source.

`depends` Comma-separated shard stems for the detected symbols.

`opencl_builtins` Comma-separated OpenCL builtins detected, or empty string if none.

`changed` `TRUE` if the file was (or would be) modified. `NA` means the file was skipped (already tagged).

## See Also

attach_cross_library_tags, attach_kernel_dependency_tags

## Examples

```
############################ Start of kernel_tagging_workflow example ########################

lib_dir <- system.file("cl/ex_glmbayes_nmath", package = "opencltools")
kernels <- list.files(
  system.file("cl/ex_glmbayes_draft_src", package = "opencltools"),
  pattern = "\\.cl$", full.names = TRUE
)

# Step 1: scan draft kernels for library calls (read-only dry run)
step1 <- attach_kernel_call_tags(
  kernel_paths = kernels,
  library_dir  = lib_dir,
  library_tag  = "nmath",
  dry_run      = TRUE
)
step1

# Step 2: expand transitive closure (small nmath library; runs on CRAN check)
nmath_small <- system.file("cl/nmath_small", package = "opencltools")
tagged      <- system.file("cl/src/dnorm_kernel.cl", package = "opencltools")
idx_small   <- write_kernel_dependency_index(library_dir = nmath_small, write = FALSE)

step2_small <- attach_cross_library_tags(
  kernel_paths = tagged,
  library_dir  = nmath_small,
  depends_tag  = "depends_nmath",
  index        = idx_small,
  dry_run      = TRUE
)
nrow(step2_small)


# Step 2: full nmath (slow)
nmath_dir <- system.file("cl/nmath", package = "opencltools")
```

```
idx       <- write_kernel_dependency_index(library_dir = nmath_dir, write = FALSE)

step2 <- attach_cross_library_tags(
  kernel_paths = tagged,
  library_dir  = nmath_dir,
  depends_tag  = "depends_nmath",
  index        = idx,
  dry_run      = TRUE
)
step2


################################################################################
## End of kernel_tagging_workflow example
################################################################################
```

---

`attach_kernel_dependency_tags` *Attach Dependency Tags to a Dependency-Ordered Kernel Library*

---

## Description

Compute and attach derived tags to each `.cl` file in a library: `@load_order`, `@all_depends`, and `@all_depends_count`.

## Usage

```
attach_kernel_dependency_tags(library_dir, dry_run = FALSE)
```

## Arguments

| | |
|---|---|
| `library_dir` | Directory containing `.cl` files with `@depends` tags. |
| `dry_run` | Logical; if `TRUE`, compute tags and reports without writing. |

## Details

Tags are written only if dependency sorting fully succeeds. If unresolved files remain, no files are modified and a cycle report is returned so source refactoring can be prioritized.

## Value

A list with:

**ok** Logical success flag.

**message** Human-readable summary.

**sorted** Sorted records data frame.

**unresolved** Unresolved records data frame (empty on success).

**cycles** Cycle report data frame (empty on success).

**tags** Per-file tag data frame with `source_origin`, `source_type`, `includes`, `depends`, `provides`, `load_order`, `all_depends`, and `all_depends_count` (present on success).

**header_functions** Functions declared in header files without `attribute_hidden`, including declaring header, inferred definition file, declaration signature, `define_alias`, and `all_depends` for the definition file.

An S3 class `"opencl_dependency_tags"` is attached for use with `print()`.

## Examples

```
############################ Start of attach_kernel_dependency_tags example ########################

lib_dir <- system.file("cl/ex_glmbayes_nmath", package = "opencltools")
res <- attach_kernel_dependency_tags(lib_dir, dry_run = TRUE)
res$ok
print(res)

################################################################################
## End of attach_kernel_dependency_tags example
################################################################################
```

---

`besselI_opencl` *Bessel Functions (OpenCL)*

---

## Description

OpenCL-backed wrappers for Bessel functions.

## Usage

```
besselI_opencl(x, nu, expon.scaled = FALSE, fallback = FALSE, verbose = FALSE)

besselJ_opencl(x, nu, fallback = FALSE, verbose = FALSE)

besselK_opencl(x, nu, expon.scaled = FALSE, fallback = FALSE, verbose = FALSE)

besselY_opencl(x, nu, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `x`, `nu` | Numeric vectors recycled together. |
| `expon.scaled` | Logical vector recycled with `x` and `nu` where applicable; maps to the `expon.scaled` flag in `besselI` / `besselK`. |
| `fallback` | When `TRUE` while `nmathopencl_has_opencl()` reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to `FALSE`. |
| `verbose` | Logical; print fallback/error diagnostics. |

## Value

Numeric vector of recycled common length.

## Known OpenCL limitations

Current Bessel OpenCL paths may require temporary-workspace allocation semantics equivalent to host `R_alloc`/`vmax*` behavior. On some GPU stacks this can fail at runtime; use `fallback = TRUE` if you must tolerate failures until device-side workspace handling is implemented.

## Examples

```
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  # Bessel OpenCL paths currently depend on temporary-workspace allocation
  # behavior (R_alloc/vmax* semantics) not yet fully implemented for device
  # execution. Keep these commented to avoid flaky CI/check failures:
  # n <- 1L
 # besselI_opencl(x = 2.0, nu = 1.5, expon.scaled = FALSE, fallback = FALSE, verbose = TRUE)
  # besselJ_opencl(x = 2.0, nu = 1.5, fallback = FALSE, verbose = TRUE)
 # besselK_opencl(x = 2.0, nu = 1.5, expon.scaled = FALSE, fallback = FALSE, verbose = TRUE)
  # besselY_opencl(x = 2.0, nu = 1.5, fallback = FALSE, verbose = TRUE)
} else {
  besselI(2.0, nu = 1.5)
  besselJ(2.0, nu = 1.5)
  besselK(2.0, nu = 1.5)
  besselY(2.0, nu = 1.5)
}
```

---

# dbeta_opencl *The Beta Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the beta distribution. These mirror the base `stats` beta family while adding OpenCL dispatch and optional CPU fallback behavior.

## Usage

```
dbeta_opencl(
  x,
  shape1,
  shape2,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)
```

```
dnbeta_opencl(
  x,
  shape1,
  shape2,
  ncp,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pbeta_opencl(
  q,
  shape1,
  shape2,
  ncp = 0,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qbeta_opencl(
  p,
  shape1,
  shape2,
  ncp = 0,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rbeta_opencl(n, shape1, shape2, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | (0,1) quantiles for central/non-central densities on this page. |
| shape1 | First shape parameter (must be > 0). |
| shape2 | Second shape parameter (must be > 0). |
| log | log density flags for density wrappers (stats semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports |

| | |
|---|---|
| | no OpenCL. All wrappers on this page default FALSE so OpenCL build/runtime faults surface unless you opt in (fallback = TRUE). Kernels overlapping 'inst/cl/nmath/pgamma_utils can still be brittle on some devices; see 'inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md' and 'inst/OPENCL_KERNEL_KNOWN_FAILURES.md'. |
| verbose | Logical; print fallback/error diagnostics. |
| ncp | Non-centrality parameter (must be >= 0). Used by dnbeta_opencl(), pbeta_opencl(), and qbeta_opencl(). |
| q | Numeric vector of quantiles for pbeta_opencl; recycled like stats::pbeta. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qbeta_opencl (like stats::qbeta). |
| n | Number of observations (non-negative integer scalar). Used only by rbeta_opencl. |

## Value

Numeric vector of length n.

## Known OpenCL limitations

**Non-central tails** qbeta_opencl with ncp > 0 may stress devices.

**Build quirks** Rf_lbeta linkage breaks on GPUs; skip GPU demos.
Tracker: 'inst/OPENCL_KERNEL_KNOWN_FAILURES.md'

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dbeta_opencl(rep(0.6, n), shape1 = 2.5, shape2 = 4, fallback = FALSE, verbose = TRUE)
 dnbeta_opencl(rep(0.6, n), shape1 = 2.5, shape2 = 4, ncp = 0.8, fallback = FALSE, verbose = TRUE)
 pbeta_opencl(q = 0.6, shape1 = 2.5, shape2 = 4, ncp = 0, fallback = FALSE, verbose = TRUE)
  ## qbeta_opencl: disabled — see inst/OPENCL_KERNEL_KNOWN_FAILURES.md
 # qbeta_opencl(rep(0.8, n), shape1 = 2.5, shape2 = 4, ncp = 0, fallback = FALSE, verbose = TRUE)
  rbeta_opencl(n, shape1 = 2.5, shape2 = 4, fallback = FALSE, verbose = TRUE)
} else {
  stats::dbeta(rep(0.6, n), shape1 = 2.5, shape2 = 4)
  stats::pbeta(0.6, shape1 = 2.5, shape2 = 4, ncp = 0)
  stats::rbeta(n, shape1 = 2.5, shape2 = 4)
}
```

---

# dbinom_raw_opencl *The Binomial Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the binomial distribution.

## Usage

```
dbinom_raw_opencl(
  x,
  size,
  prob,
  qprob = NULL,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

dbinom_opencl(
  x,
  size,
  prob,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pbinom_opencl(
  q,
  size,
  prob,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qbinom_opencl(
  p,
  size,
  prob,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rbinom_opencl(n, size, prob, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile. |
| size | Number of trials (must be >= 0). |
| prob | Probability of success in [0, 1]. |
| qprob | Complementary probability. If NULL, uses 1 - prob. |
| log | log density switch for dbinom* wrappers (stats semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. Density dbinom_* wrappers default FALSE; pbinom_opencl defaults TRUE temporarily ('inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md'); quantile and random wrappers default FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles for pbinom_opencl; recycled like stats::pbinom. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Probabilities for qbinom_opencl; ordering matches stats::qbinom. |
| n | Number of observations (non-negative integer scalar). Used only by rbinom_opencl. |

## Value

Numeric vector result from the corresponding binomial-family operation.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dbinom_raw_opencl(rep(6, n), size = 10, prob = 0.3, fallback = FALSE, verbose = TRUE)
  dbinom_opencl(rep(6, n), size = 10, prob = 0.3, fallback = FALSE, verbose = TRUE)
  pbinom_opencl(q = 6, size = 10, prob = 0.3, fallback = FALSE, verbose = TRUE)
  qbinom_opencl(rep(0.8, n), size = 10, prob = 0.3, fallback = FALSE, verbose = TRUE)
  rbinom_opencl(n, size = 10, prob = 0.3, fallback = FALSE, verbose = TRUE)
} else {
  stats::dbinom(rep(6, n), size = 10, prob = 0.3)
  stats::dbinom(rep(6, n), size = 10, prob = 0.3)
  stats::pbinom(6, size = 10, prob = 0.3)
  stats::qbinom(rep(0.8, n), size = 10, prob = 0.3)
  stats::rbinom(n, size = 10, prob = 0.3)
}
```

dcauchy_opencl *The Cauchy Distribution (OpenCL)*

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the Cauchy distribution.

## Usage

```
dcauchy_opencl(
  x,
  location = 0,
  scale = 1,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pcauchy_opencl(
  q,
  location = 0,
  scale = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qcauchy_opencl(
  p,
  location = 0,
  scale = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rcauchy_opencl(n, location = 0, scale = 1, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile. |

| | |
|---|---|
| location | Location parameter. |
| scale | Scale parameter (must be > 0). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | Quantiles (pcauchy_opencl; aligns with stats::pcauchy recycling). |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qcauchy_opencl (like stats::qcauchy). |
| n | Number of observations (non-negative integer scalar). Used only by rcauchy_opencl. |

## Value

Numeric vector result from the corresponding Cauchy-family operation.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dcauchy_opencl(rep(0.2, n), location = 0, scale = 1, fallback = FALSE, verbose = TRUE)
  pcauchy_opencl(q = 0.2, location = 0, scale = 1, fallback = FALSE, verbose = TRUE)
  qcauchy_opencl(rep(0.8, n), location = 0, scale = 1, fallback = FALSE, verbose = TRUE)
  rcauchy_opencl(n, location = 0, scale = 1, fallback = FALSE, verbose = TRUE)
} else {
  stats::dcauchy(rep(0.2, n), location = 0, scale = 1)
  stats::pcauchy(0.2, location = 0, scale = 1)
  stats::qcauchy(rep(0.8, n), location = 0, scale = 1)
  stats::rcauchy(n, location = 0, scale = 1)
}
```

---

# dchisq_opencl *The Chi-squared Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the chi-squared distribution, including non-central paths via ncp.

## Usage

```
dchisq_opencl(
  x,
  df,
  ncp = 0,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pchisq_opencl(
  q,
  df,
  ncp = 0,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qchisq_opencl(
  p,
  df,
  ncp = 0,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rchisq_opencl(n, df, ncp = 0, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of quantiles for dchisq_opencl. |
| df | Degrees of freedom (must be > 0). |
| ncp | Non-centrality parameter (must be >= 0). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. dchisq_opencl defaults FALSE; distribution/quantile and rchisq_opencl |

| | |
|---|---|
| | remain TRUE temporarily (‘inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md’); pass explicit fallback to override. |
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles for pchisq_opencl; recycled like stats::pchisq. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qchisq_opencl (like stats::qchisq). |
| n | Number of observations (non-negative integer scalar). Used only by rchisq_opencl; dchisq_opencl takes vector x first (like stats::dchisq). |

## Value

Numeric vector of length n.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dchisq_opencl(rep(4.5, n), df = 6, ncp = 0, fallback = FALSE, verbose = TRUE)
  pchisq_opencl(q = 4.5, df = 6, ncp = 0, fallback = FALSE, verbose = TRUE)
  qchisq_opencl(rep(0.8, n), df = 6, ncp = 0, fallback = FALSE, verbose = TRUE)
  rchisq_opencl(n, df = 6, ncp = 0, fallback = FALSE, verbose = TRUE)
} else {
  stats::dchisq(rep(4.5, n), df = 6, ncp = 0)
  stats::pchisq(4.5, df = 6, ncp = 0)
  stats::qchisq(rep(0.8, n), df = 6, ncp = 0)
  stats::rchisq(n, df = 6, ncp = 0)
}
```

---

# dexp_opencl *The Exponential Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the exponential distribution.

## Usage

```
dexp_opencl(
  x,
  rate = 1,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)
```

```
pexp_opencl(
  q,
  rate = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qexp_opencl(
  p,
  rate = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rexp_opencl(n, rate = 1, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile. |
| rate | Rate parameter (must be > 0). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles for pexp_opencl; recycled like stats::pexp. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qexp_opencl (like stats::qexp). |
| n | Number of observations (non-negative integer scalar). Used only by rexp_opencl. |

## Value

Numeric vector result from the corresponding exponential-family operation.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dexp_opencl(rep(1.2, n), rate = 1, fallback = FALSE, verbose = TRUE)
  pexp_opencl(q = 1.2, rate = 1, fallback = FALSE, verbose = TRUE)
  qexp_opencl(rep(0.8, n), rate = 1, fallback = FALSE, verbose = TRUE)
  rexp_opencl(n, rate = 1, fallback = FALSE, verbose = TRUE)
} else {
  stats::dexp(rep(1.2, n), rate = 1)
  stats::pexp(1.2, rate = 1)
  stats::qexp(rep(0.8, n), rate = 1)
  stats::rexp(n, rate = 1)
}
```

---

# df_opencl *The F Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the F distribution.

## Usage

```
df_opencl(
  x,
  df1,
  df2,
  ncp = 0,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pf_opencl(
  q,
  df1,
  df2,
  ncp = 0,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)
```

```
qf_opencl(
  p,
  df1,
  df2,
  ncp = 0,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rf_opencl(n, df1, df2, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of quantiles (df_opencl). |
| df1 | Numerator degrees of freedom (must be > 0). |
| df2 | Denominator degrees of freedom (must be > 0). |
| ncp | Non-centrality parameter (must be >= 0). Used by df_opencl(), pf_opencl(), and qf_opencl(). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. df_opencl defaults FALSE; pf_opencl, qf_opencl, and rf_opencl default TRUE temporarily ('inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md'). |
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles (pf_opencl); recycled like stats::pf. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qf_opencl (like stats::qf). |
| n | Number of observations (non-negative integer scalar). Used only by rf_opencl; df_opencl takes vector x first (like stats::df). |

## Value

For df_opencl, qf_opencl, rf_opencl: numeric vector result. For pf_opencl: numeric vector of recycled length (see stats::pf).

## Known OpenCL limitations

qf_opencl() can fail on some GPU/driver combinations with CL_OUT_OF_RESOURCES. This has been observed in both central and non-central settings, with non-central paths typically more fragile.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  df_opencl(rep(2, n), df1 = 5, df2 = 9, ncp = 0, fallback = FALSE, verbose = TRUE)
  df_opencl(rep(2, n), df1 = 5, df2 = 9, ncp = 1.1, fallback = FALSE, verbose = TRUE)
  pf_opencl(q = 2, df1 = 5, df2 = 9, ncp = 0, fallback = FALSE, verbose = TRUE)
  pf_opencl(q = 2, df1 = 5, df2 = 9, ncp = 1.1, fallback = FALSE, verbose = TRUE)
  rf_opencl(n, df1 = 5, df2 = 9, fallback = FALSE, verbose = TRUE)
  ## qf_opencl: disabled — see inst/OPENCL_KERNEL_KNOWN_FAILURES.md
} else {
  stats::df(rep(2, n), df1 = 5, df2 = 9, ncp = 0)
  stats::df(rep(2, n), df1 = 5, df2 = 9, ncp = 1.1)
  stats::pf(2, df1 = 5, df2 = 9, ncp = 0)
  stats::pf(2, df1 = 5, df2 = 9, ncp = 1.1)
  stats::rf(n, df1 = 5, df2 = 9)
}
```

---

dgamma_opencl *The Gamma Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the gamma distribution. These mirror the base `stats` gamma family while adding OpenCL dispatch and optional CPU fallback behavior.

## Usage

```
dgamma_opencl(
  x,
  shape,
  scale = 1,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pgamma_opencl(
  q,
  shape,
  rate = 1,
  scale = 1/rate,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
```

```
)

qgamma_opencl(
  p,
  shape,
  scale = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rgamma_opencl(n, shape, scale = 1, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of quantiles for dgamma_opencl. |
| shape | Shape parameter (must be > 0). |
| scale | Scale parameter (must be > 0). For pgamma_opencl, combined with rate like stats::pgamma. |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for pgamma_opencl/qgamma_opencl; parallel dispatch reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. dgamma_opencl defaults FALSE; distribution/quantile wrappers follow 'inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md'. Pass explicit fallback where needed. |
| verbose | Logical; print informational fallback messages. |
| q | Numeric vector of quantiles for pgamma_opencl (same role as stats::pgamma). |
| rate | Optional rate for pgamma_opencl; see pgamma. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qgamma_opencl (like stats::qgamma). |
| n | Number of observations (non-negative integer scalar). Used only by rgamma_opencl; dgamma_opencl takes vector x first (like stats::dgamma). |

## Details

pgamma_opencl follows pgamma argument names and rate/scale handling (including the error when both are supplied inconsistently). Recycling of q, shape, and scale follows stats::pgamma. Vector lower.tail and log.p are recycled row-wise with those arguments (like pnorm_opencl); a single-vector stats::pgamma() call does not apply tail flags element-wise. There is no leading n argument for pgamma_opencl. On the GPU path each recycled row runs pgamma_kernel once with

n_out = 1. Missing or non-finite values after recycling, or non-positive shape/scale, use row-wise stats::pgamma.

## Value

Numeric vector result from the corresponding gamma-family operation.

## Known OpenCL limitations

Compilation of qgamma_kernel can fail (ptxas: unresolved stirlerr_cycle_free). Runnable examples omit GPU qgamma_opencl until resolved. See 'inst/OPENCL_KERNEL_KNOWN_FAILURES.md'.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dgamma_opencl(rep(1.2, n), shape = 2, scale = 1, fallback = FALSE, verbose = TRUE)
  pgamma_opencl(q = 1.2, shape = 2, scale = 1, fallback = FALSE, verbose = TRUE)
  ## qgamma_opencl: disabled — see inst/OPENCL_KERNEL_KNOWN_FAILURES.md
  # qgamma_opencl(rep(0.8, n), shape = 2, scale = 1, fallback = FALSE, verbose = TRUE)
  rgamma_opencl(n, shape = 2, scale = 1, fallback = FALSE, verbose = TRUE)
} else {
  stats::dgamma(rep(1.2, n), shape = 2, scale = 1)
  stats::pgamma(1.2, shape = 2, scale = 1)
  stats::rgamma(n, shape = 2, scale = 1)
}
```

---

# dgeom_opencl *The Geometric Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the geometric distribution.

## Usage

```
dgeom_opencl(
  x,
  prob,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pgeom_opencl(
  q,
  prob,
```

```
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qgeom_opencl(
  p,
  prob,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rgeom_opencl(n, prob, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile. |
| prob | Probability of success in [0, 1]. |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. dgeom_opencl defaults FALSE. pgeom_opencl, qgeom_opencl, and rgeom_opencl default FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles for pgeom_opencl; recycled like stats::pgeom. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qgeom_opencl (like stats::qgeom). |
| n | Number of observations (non-negative integer scalar). Used only by rgeom_opencl. |

## Value

Numeric vector result from the corresponding geometric-family operation.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dgeom_opencl(rep(4, n), prob = 0.3, fallback = FALSE, verbose = TRUE)
```

```
  pgeom_opencl(q = 4, prob = 0.3, fallback = FALSE, verbose = TRUE)
  qgeom_opencl(rep(0.8, n), prob = 0.3, fallback = FALSE, verbose = TRUE)
  rgeom_opencl(n, prob = 0.3, fallback = FALSE, verbose = TRUE)
} else {
  stats::dgeom(rep(4, n), prob = 0.3)
  stats::pgeom(4, prob = 0.3)
  stats::qgeom(rep(0.8, n), prob = 0.3)
  stats::rgeom(n, prob = 0.3)
}
```

---

dhyper_opencl *The Hypergeometric Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the hypergeometric distribution.

## Usage

```
dhyper_opencl(
  x,
  m,
  n_black,
  k,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

phyper_opencl(
  q,
  m,
  n_black,
  k,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qhyper_opencl(
  p,
  m,
  n_black,
  k,
```

```
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rhyper_opencl(n, m, n_black, k, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile. |
| m | Number of white balls in the urn (must be >= 0). |
| n_black | Number of black balls in the urn (must be >= 0). |
| k | Number of draws (must be >= 0). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. dhyper_opencl defaults FALSE; phyper_opencl and rhyper_opencl default TRUE temporarily ('inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md'); qhyper_opencl defaults FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles for phyper_opencl; recycled like stats::phyper. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qhyper_opencl (like stats::qhyper). |
| n | Number of observations (non-negative integer scalar). Used only by rhyper_opencl. |

## Value

Numeric vector result from the corresponding hypergeometric-family operation.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dhyper_opencl(rep(3, n), m = 10, n_black = 12, k = 8, fallback = FALSE, verbose = TRUE)
  phyper_opencl(q = 3, m = 10, n_black = 12, k = 8, fallback = FALSE, verbose = TRUE)
  qhyper_opencl(rep(0.8, n), m = 10, n_black = 12, k = 8, fallback = FALSE, verbose = TRUE)
  rhyper_opencl(n, m = 10, n_black = 12, k = 8, fallback = FALSE, verbose = TRUE)
} else {
  stats::dhyper(rep(3, n), m = 10, n = 12, k = 8)
  stats::phyper(3, m = 10, n = 12, k = 8)
  stats::qhyper(rep(0.8, n), m = 10, n = 12, k = 8)
```

```
  stats::rhyper(n, m = 10, n = 12, k = 8)
}
```

---

dlnorm_opencl *The Lognormal Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the lognormal distribution.

## Usage

```
dlnorm_opencl(
  x,
  meanlog = 0,
  sdlog = 1,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

plnorm_opencl(
  q,
  meanlog = 0,
  sdlog = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qlnorm_opencl(
  p,
  meanlog = 0,
  sdlog = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rlnorm_opencl(n, meanlog = 0, sdlog = 1, fallback = FALSE, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| x | Numeric scalar quantile (must be >= 0). |
| meanlog | Mean of the distribution on the log scale. |
| sdlog | Standard deviation on the log scale (must be > 0). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles for plnorm_opencl; recycled like stats::plnorm. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qlnorm_opencl (like stats::qlnorm). |
| n | Number of observations (non-negative integer scalar). Used only by rlnorm_opencl. |

### Value

Numeric vector of length n.

### Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dlnorm_opencl(rep(1.2, n), meanlog = 0.1, sdlog = 0.8, fallback = FALSE, verbose = TRUE)
  plnorm_opencl(q = 1.2, meanlog = 0.1, sdlog = 0.8, fallback = FALSE, verbose = TRUE)
  qlnorm_opencl(rep(0.8, n), meanlog = 0.1, sdlog = 0.8, fallback = FALSE, verbose = TRUE)
  rlnorm_opencl(n, meanlog = 0.1, sdlog = 0.8, fallback = FALSE, verbose = TRUE)
} else {
  stats::dlnorm(rep(1.2, n), meanlog = 0.1, sdlog = 0.8)
  stats::plnorm(1.2, meanlog = 0.1, sdlog = 0.8)
  stats::qlnorm(rep(0.8, n), meanlog = 0.1, sdlog = 0.8)
  stats::rlnorm(n, meanlog = 0.1, sdlog = 0.8)
}
```

---

## dlogis_opencl *The Logistic Distribution (OpenCL)*

---

### Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the logistic distribution.

## Usage

```
dlogis_opencl(
  x,
  location = 0,
  scale = 1,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

plogis_opencl(
  q,
  location = 0,
  scale = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qlogis_opencl(
  p,
  location = 0,
  scale = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rlogis_opencl(n, location = 0, scale = 1, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile. |
| location | Location parameter. |
| scale | Scale parameter (must be > 0). |
| log | log density switch for dlogis_opencl (stats semantics). |
| opencl_parallel | Dispatch hint (TRUE, FALSE, NA) for plogis_opencl and qlogis_opencl; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. See 'inst/OPENCL_KERNEL_KNOWN_FAILURES.md'. |

| | |
|---|---|
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles for plogis_opencl; recycled like stats::plogis. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qlogis_opencl (like stats::qlogis). |
| n | Number of observations (non-negative integer scalar). Used only by rlogis_opencl. |

## Value

Numeric vector result from the corresponding logistic-family operation.

## Known OpenCL limitations

Some platforms fail to link qlogis_kernel (ptxas unresolved Rf_qlogis). Runnable examples omit GPU qlogis_opencl until resolved. See 'inst/OPENCL_KERNEL_KNOWN_FAILURES.md'.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dlogis_opencl(rep(0.2, n), location = 0, scale = 1, fallback = FALSE, verbose = TRUE)
  plogis_opencl(q = 0.2, location = 0, scale = 1, fallback = FALSE, verbose = TRUE)
  ## qlogis_opencl: disabled — see inst/OPENCL_KERNEL_KNOWN_FAILURES.md
  rlogis_opencl(n, location = 0, scale = 1, fallback = FALSE, verbose = TRUE)
} else {
  stats::dlogis(rep(0.2, n), location = 0, scale = 1)
  stats::plogis(0.2, location = 0, scale = 1)
  stats::rlogis(n, location = 0, scale = 1)
}
```

---

# dnbinom_opencl *The Negative Binomial Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the negative binomial distribution, with variants parameterized by prob and by mu.

## Usage

```
dnbinom_opencl(
  x,
  size,
  prob,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
```

```
  verbose = FALSE
)

pnbinom_opencl(
  q,
  size,
  prob,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qnbinom_opencl(
  p,
  size,
  prob,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rnbinom_opencl(n, size, prob, fallback = FALSE, verbose = FALSE)

dnbinom_mu_opencl(
  x,
  size,
  mu,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pnbinom_mu_opencl(
  q,
  size,
  mu,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)
```

```
qnbinom_mu_opencl(
  p,
  size,
  mu,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rnbinom_mu_opencl(n, size, mu, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile (must be >= 0). |
| size | Dispersion/size parameter (must be >= 0). |
| prob | Probability of success in [0, 1]. |
| log | log density switch for density wrappers (stats semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. Density wrappers (dnbinom_opencl, dnbinom_mu_opencl) default FALSE; pnbinom_opencl/pnbinom_mu_opencl default TRUE temporarily ('inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md'); quantiles and random wrappers default FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | Quantiles for pnbinom* wrappers (stats::pnbinom recycling). |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Probabilities for qnbinom* wrappers (stats::qnbinom semantics). |
| n | Observations scalar; used by negative-binomial r* wrappers only. |
| mu | Mean parameter (must be >= 0). |

## Value

Numeric vector result from the corresponding negative-binomial operation.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dnbinom_opencl(rep(4, n), size = 7, prob = 0.4, fallback = FALSE, verbose = TRUE)
  pnbinom_opencl(q = 4, size = 7, prob = 0.4, fallback = FALSE, verbose = TRUE)
```

```
  qnbinom_opencl(rep(0.8, n), size = 7, prob = 0.4, fallback = FALSE, verbose = TRUE)
  rnbinom_opencl(n, size = 7, prob = 0.4, fallback = FALSE, verbose = TRUE)

  dnbinom_mu_opencl(rep(4, n), size = 7, mu = 5, fallback = FALSE, verbose = TRUE)
  pnbinom_mu_opencl(q = 4, size = 7, mu = 5, fallback = FALSE, verbose = TRUE)
  qnbinom_mu_opencl(rep(0.8, n), size = 7, mu = 5, fallback = FALSE, verbose = TRUE)
  rnbinom_mu_opencl(n, size = 7, mu = 5, fallback = FALSE, verbose = TRUE)
} else {
  stats::dnbinom(rep(4, n), size = 7, prob = 0.4)
  stats::pnbinom(4, size = 7, prob = 0.4)
  stats::qnbinom(rep(0.8, n), size = 7, prob = 0.4)
  stats::rnbinom(n, size = 7, prob = 0.4)

  stats::dnbinom(rep(4, n), size = 7, mu = 5)
  stats::pnbinom(4, size = 7, mu = 5)
  stats::qnbinom(rep(0.8, n), size = 7, mu = 5)
  stats::rnbinom(n, size = 7, mu = 5)
}
```

---

dnorm_opencl *The Normal Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the normal distribution. These mirror the base `stats` normal family while adding OpenCL dispatch and optional CPU fallback behavior.

## Usage

```
dnorm_opencl(
  x,
  mean = 0,
  sd = 1,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pnorm_opencl(
  q,
  mean = 0,
  sd = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
```

```
  verbose = FALSE
)

qnorm_opencl(
  p,
  mean = 0,
  sd = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rnorm_opencl(n, mean = 0, sd = 1, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of quantiles. |
| mean | Location parameter (rnorm: scalar). For pnorm_opencl and qnorm_opencl, recycled like the corresponding stats function. |
| sd | Scale parameter (rnorm: scalar, sd >= 0). For pnorm_opencl and qnorm_opencl, recycled like the corresponding stats function. The GPU path runs only when every recycled value is strictly positive (sd > 0); sd == 0 is evaluated with stats::pnorm. |
| log | log flag for dnorm_opencl (stats semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE. |
| verbose | Logical; print informational fallback messages. |
| q | Numeric vector of quantiles for pnorm_opencl (same role as stats::pnorm). |
| lower.tail, log.p | Recycling for pnorm_opencl; see Details for contrasts with some vector stats::pnorm calls. |
| p | Numeric vector of probabilities for qnorm_opencl (like stats::qnorm). |
| n | Number of observations (non-negative integer scalar). Used only by rnorm_opencl; not an argument to pnorm_opencl / qnorm_opencl. |

## Details

pnorm_opencl matches pnorm on the first five statistical arguments (q, mean, sd, lower.tail, log.p) and the usual defaults. It also accepts opencl_parallel, fallback, and verbose. Only rnorm_opencl uses a leading n.
Recycling follows pnorm once arguments are aligned. On the GPU path, each recycled row calls

the scalar pnorm_kernel once (len submissions).
Vector q, mean, sd, lower.tail, and log.p yield one OpenCL evaluation per output index.
Length-zero q returns numeric(0) as in pnorm.
Missing or non-finite values (after recycling), or any sd == 0, use CPU pnorm. Zero-length recycling errors and negative sd still error.

## Value

Numeric vector result from the corresponding normal-family operation.

## Examples

```
n <- 5L
x <- c(-1, 0, 1)
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dnorm_opencl(x, mean = 0, sd = 1, fallback = FALSE, verbose = TRUE)
  pnorm_opencl(q = 0.2, mean = 0, sd = 1, fallback = FALSE, verbose = TRUE)
  qnorm_opencl(rep(0.8, n), mean = 0, sd = 1, fallback = FALSE, verbose = TRUE)
  rnorm_opencl(n, mean = 0, sd = 1, fallback = FALSE, verbose = TRUE)
} else {
  stats::dnorm(x, mean = 0, sd = 1)
  stats::pnorm(0.2, mean = 0, sd = 1)
  stats::qnorm(rep(0.8, n), mean = 0, sd = 1)
  stats::rnorm(n, mean = 0, sd = 1)
}
```

---

# dpois_raw_opencl *The Poisson Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the Poisson distribution.

## Usage

```
dpois_raw_opencl(
  x,
  lambda,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

dpois_opencl(
  x,
  lambda,
```

```
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

ppois_opencl(
  q,
  lambda,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qpois_opencl(
  p,
  lambda,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rpois_opencl(n, lambda, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile (must be >= 0). |
| lambda | Mean/rate parameter (must be >= 0). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. Density dpois_* wrappers default FALSE; ppois_opencl defaults TRUE temporarily ('inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md'); qpois_opencl and rpois_opencl default FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles for ppois_opencl; recycled like stats::ppois. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qpois_opencl (like stats::qpois). |
| n | Number of observations (non-negative integer scalar). Used only by rpois_opencl. |

## Value

Numeric vector result from the corresponding Poisson-family operation.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dpois_raw_opencl(rep(4, n), lambda = 4, fallback = FALSE, verbose = TRUE)
  dpois_opencl(rep(4, n), lambda = 4, fallback = FALSE, verbose = TRUE)
  ppois_opencl(q = 4, lambda = 4, fallback = FALSE, verbose = TRUE)
  qpois_opencl(rep(0.8, n), lambda = 4, fallback = FALSE, verbose = TRUE)
  rpois_opencl(n, lambda = 4, fallback = FALSE, verbose = TRUE)
} else {
  stats::dpois(rep(4, n), lambda = 4)
  stats::dpois(rep(4, n), lambda = 4)
  stats::ppois(4, lambda = 4)
  stats::qpois(rep(0.8, n), lambda = 4)
  stats::rpois(n, lambda = 4)
}
```

---

# dsignrank_opencl *The Wilcoxon Signed Rank Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the Wilcoxon signed rank distribution.

## Usage

```
dsignrank_opencl(
  x,
  nsize,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

psignrank_opencl(
  q,
  nsize,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)
```

```
qsignrank_opencl(
  p,
  nsize,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rsignrank_opencl(n, nsize, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile (must be >= 0). |
| nsize | Number of observations used by signed-rank routines (must be > 0). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | p*-wrapper quantiles (stats::psignrank semantics). |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | q*-wrapper probabilities (stats::qsignrank semantics). |
| n | Draw-count scalar (r* path only). |

## Value

Numeric vector result from the corresponding signed-rank operation.

## Known OpenCL limitations

Signed-rank kernels can fail to build on some GPU toolchains due to unresolved runtime allocation symbols (for example R_chk_calloc). Use fallback = TRUE only if you need to tolerate those failures until device-safe shims are complete.

## Examples

```
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  # Signed-rank OpenCL kernels are currently known to fail on some GPU stacks
  # due to unresolved runtime allocation symbols (e.g., R_chk_calloc).
  # Keeping these commented avoids flaky check failures:
```

```
  # n <- 5L
  # dsignrank_opencl(n, x = 6, nsize = 8, fallback = FALSE, verbose = TRUE)
  # psignrank_opencl(q = 6, nsize = 8, fallback = FALSE, verbose = TRUE)
  # qsignrank_opencl(rep(0.8, n), nsize = 8, fallback = FALSE, verbose = TRUE)
  # rsignrank_opencl(n, nsize = 8, fallback = FALSE, verbose = TRUE)
} else {
  n <- 5L
  stats::dsignrank(rep(6, n), n = 8)
  stats::psignrank(6, n = 8)
  stats::qsignrank(rep(0.8, n), n = 8)
  stats::rsignrank(n, n = 8)
}
```

---

dt_opencl *The Student t Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the Student t distribution.

## Usage

```
dt_opencl(
  x,
  df,
  ncp = 0,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pt_opencl(
  q,
  df,
  ncp = 0,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qt_opencl(
  p,
  df,
  ncp = 0,
```

```
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rt_opencl(n, df, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of quantiles (dt_opencl). |
| df | Degrees of freedom (must be > 0). |
| ncp | Non-centrality parameter. |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. dt_opencl defaults FALSE; pt_opencl, qt_opencl, and rt_opencl default TRUE temporarily ('inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md'). |
| verbose | Logical; print fallback/error diagnostics. |
| q | Numeric vector of quantiles (pt_opencl); recycled like stats::pt. |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qt_opencl (like stats::qt). |
| n | Number of observations (non-negative integer scalar). Used only by rt_opencl; dt_opencl takes vector x first (like stats::dt). |

## Value

For dt_opencl, qt_opencl, rt_opencl: numeric vector result. For pt_opencl: numeric vector of recycled length (see stats::pt).

## Known OpenCL limitations

Some platforms fail to link qnt_kernel (ptxas unresolved Rf_qnt). Runnable examples omit GPU qt_opencl until resolved. See 'inst/OPENCL_KERNEL_KNOWN_FAILURES.md'.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dt_opencl(rep(1.5, n), df = 8, ncp = 0, fallback = FALSE, verbose = TRUE)
  dt_opencl(rep(1.5, n), df = 8, ncp = 1.2, fallback = FALSE, verbose = TRUE)
  pt_opencl(q = 1.5, df = 8, ncp = 0, fallback = FALSE, verbose = TRUE)
```

```
  pt_opencl(q = 1.5, df = 8, ncp = 1.2, fallback = FALSE, verbose = TRUE)
  ## qt_opencl: disabled — see inst/OPENCL_KERNEL_KNOWN_FAILURES.md
  rt_opencl(n, df = 8, fallback = FALSE, verbose = TRUE)
} else {
  stats::dt(rep(1.5, n), df = 8, ncp = 0)
  stats::dt(rep(1.5, n), df = 8, ncp = 1.2)
  stats::pt(1.5, df = 8, ncp = 0)
  stats::pt(1.5, df = 8, ncp = 1.2)
  stats::rt(n, df = 8)
}
```

---

dunif_opencl *The Uniform Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the uniform distribution. These mirror the base `stats` uniform family while adding OpenCL dispatch and optional CPU fallback behavior.

## Usage

```
dunif_opencl(
  x,
  min = 0,
  max = 1,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

punif_opencl(
  q,
  min = 0,
  max = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qunif_opencl(
  p,
  min = 0,
  max = 1,
  lower.tail = TRUE,
```

```
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

runif_opencl(n, min = 0, max = 1, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile for dunif_opencl. |
| min | Lower limit of the distribution. |
| max | Upper limit of the distribution. |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. See 'inst/OPENCL_KERNEL_KNOWN_FAILURES.md'. |
| verbose | Logical; print informational fallback messages. |
| q | Numeric quantiles (punif_opencl; like stats::punif). |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Probabilities for qunif_opencl (stats::qunif semantics). |
| n | Draw count (r*); density wrappers still lead with x. |

## Value

Numeric vector result from the corresponding uniform-family operation.

## Known OpenCL limitations

Some platforms fail to link qunif_kernel (ptxas unresolved Rf_qunif). Runnable examples omit GPU qunif_opencl until resolved. See 'inst/OPENCL_KERNEL_KNOWN_FAILURES.md'.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dunif_opencl(rep(0.4, n), min = 0, max = 1, fallback = FALSE, verbose = TRUE)
  punif_opencl(q = 0.4, min = 0, max = 1, fallback = FALSE, verbose = TRUE)
  ## qunif_opencl: disabled — see inst/OPENCL_KERNEL_KNOWN_FAILURES.md
  runif_opencl(n, min = 0, max = 1, fallback = FALSE, verbose = TRUE)
} else {
  stats::dunif(rep(0.4, n), min = 0, max = 1)
  stats::punif(0.4, min = 0, max = 1)
  stats::runif(n, min = 0, max = 1)
}
```

dweibull_opencl *The Weibull Distribution (OpenCL)*

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the Weibull distribution.

## Usage

```
dweibull_opencl(
  x,
  shape,
  scale = 1,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pweibull_opencl(
  q,
  shape,
  scale = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qweibull_opencl(
  p,
  shape,
  scale = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rweibull_opencl(n, shape, scale = 1, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile (must be >= 0). |

| | |
|---|---|
| shape | Shape parameter (must be > 0). |
| scale | Scale parameter (must be > 0). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. Defaults TRUE only for qweibull_opencl temporarily (‘inst/OPENCL_PGAMMA_UTILS_KERNEL_ density/survival/rand stay FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| q | p*-wrapper quantiles (stats::pweibull semantics). |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | Numeric vector of probabilities for qweibull_opencl (like stats::qweibull). |
| n | Number of observations (non-negative integer scalar). Used only by rweibull_opencl. |

## Value

Numeric vector result from the corresponding Weibull-family operation.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  dweibull_opencl(rep(1.2, n), shape = 2, scale = 1.5, fallback = FALSE, verbose = TRUE)
  pweibull_opencl(q = 1.2, shape = 2, scale = 1.5, fallback = FALSE, verbose = TRUE)
  qweibull_opencl(rep(0.8, n), shape = 2, scale = 1.5, fallback = FALSE, verbose = TRUE)
  rweibull_opencl(n, shape = 2, scale = 1.5, fallback = FALSE, verbose = TRUE)
} else {
  stats::dweibull(rep(1.2, n), shape = 2, scale = 1.5)
  stats::pweibull(1.2, shape = 2, scale = 1.5)
  stats::qweibull(rep(0.8, n), shape = 2, scale = 1.5)
  stats::rweibull(n, shape = 2, scale = 1.5)
}
```

---

# dwilcox_opencl *The Wilcoxon Rank Sum Distribution (OpenCL)*

---

## Description

OpenCL-backed density, distribution, quantile, and random generation wrappers for the Wilcoxon rank sum distribution.

## Usage

```
dwilcox_opencl(
  x,
  m,
  nn,
  log = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

pwilcox_opencl(
  q,
  m,
  nn,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qwilcox_opencl(
  p,
  m,
  nn,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

rwilcox_opencl(n, m, nn, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric scalar quantile (must be >= 0). |
| m | Number of observations in one sample (must be > 0). |
| nn | Number of observations in the other sample (must be > 0). |
| log | log flag for densities (stats *d*-family semantics). |
| opencl_parallel | Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE. |

| | |
|---|---|
| verbose | Logical; print fallback/error diagnostics. |
| q | p*-wrapper quantiles (stats::pwilcox semantics). |
| lower.tail, log.p | Tail/log-*p* inputs (stats meanings). |
| p | q*-wrapper probabilities (stats::qwilcox semantics). |
| n | Draw-count scalar (r* path only). |

## Value

Numeric vector result from the corresponding Wilcoxon-family operation.

## Known OpenCL limitations

Wilcoxon kernels can still hit runtime-shim gaps depending on device and driver stack (for example unresolved runtime symbols in some builds). Defaults follow fallback = FALSE: OpenCL failures surface unless you pass fallback = TRUE.

## Examples

```
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  # Wilcoxon OpenCL kernels are currently known to fail on some GPU stacks due
  # to unresolved runtime symbols. Keeping these commented avoids flaky checks:
  # n <- 1L
  # dwilcox_opencl(n, x = 5, m = 5, nn = 7, fallback = FALSE, verbose = TRUE)
  # pwilcox_opencl(q = 5, m = 5, nn = 7, fallback = FALSE, verbose = TRUE)
  # qwilcox_opencl(rep(0.8, n), m = 5, nn = 7, fallback = FALSE, verbose = TRUE)
  #
  # Known allocator/runtime fragility on some stacks:
  # rwilcox_opencl(n, m = 5, nn = 7, fallback = FALSE, verbose = TRUE)
} else {
  n <- 1L
  stats::dwilcox(5, m = 5, n = 7)
  stats::pwilcox(5, m = 5, n = 7)
  stats::qwilcox(rep(0.8, n), m = 5, n = 7)
  stats::rwilcox(n, m = 5, n = 7)
}
```

---

# Ex_EnvelopeEval *Evaluate Negative Log-Likelihood and Gradients*

---

## Description

These functions implement the grid evaluation logic used in envelope construction for rejection sampling. They make use of the theory described in (Nygren and Nygren 2006) and the general implementation outlined in (Nygren 2025).

## Usage

```
Ex_EnvelopeEval(G4, y, x, mu, P, alpha, wt,
                    family, link,
                    use_opencl = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| G4 | Numeric matrix of parameter values (parameters * grid points). |
| y | Numeric response vector. |
| x | Numeric design matrix. |
| mu | Numeric matrix of offsets or prior means. |
| P | Numeric matrix representing the portion of the prior precision shifted into the likelihood. |
| alpha | Numeric offset vector of length m |
| wt | Numeric vector of weights. |
| family | Character string; model family (e.g. "gaussian"). |
| link | Character string; link function (e.g. "identity"). |
| use_opencl | Logical; if TRUE, attempt OpenCL acceleration. |
| verbose | Logical; if TRUE, print diagnostic output. |

## Details

Compact overview.
Derivations:\cr vignettes/equations ((Nygren and Nygren 2006)).

- Dispatcher fans out to CPU and OpenCL kernel runners.
- NegLL vectors plus cbars matrices feed envelopes in rNormal_reg internals.
- Acceptance inequalities mirror the vignette "Simulation execution" chapter.

## Value

**Ex_EnvelopeEval** NegLL: negatives logLik; cbars: tangent gradients.

**f2_f3_non_opencl** qf/grad from CPU kernels.

**f2_f3_opencl** qf/grad from OpenCL.

**run_opencl_pilot** Pilot runtime estimate (seconds).

## References

Nygren K (2025). "Chapter A05: Simulation Methods – Likelihood Subgradient Densities." Vignette in the glmbayes R package. R vignette name: Chapter-A05.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

## See Also

Ex_EnvelopeSize (Nygren 2025) (Nygren 2025) (Nygren 2025) (Nygren 2025)

## Examples

```
################################ Start of Ex_EnvelopeEval example ###################

# This example demonstrates Ex_EnvelopeEval in isolation. Ex_EnvelopeEval evaluates
# the negative log-likelihood and gradients at a grid of parameter values.
# It is called internally by EnvelopeBuild. Here we build the same inputs
# (grid G4, standardized model) using Ex_EnvelopeSize and expand.grid, then
# call Ex_EnvelopeEval directly. The setup mirrors Ex_EnvelopeBuild through
# the standardization step.

data(menarche, package = "MASS")
Age2 <- menarche$Age - 13

x <- matrix(as.numeric(1.0), nrow = length(Age2), ncol = 2)
x[, 2] <- Age2

y <- menarche$Menarche / menarche$Total
wt <- menarche$Total

mu <- matrix(as.numeric(0.0), nrow = 2, ncol = 1)
mu[2, 1] <- (log(0.9 / 0.1) - log(0.5 / 0.5)) / 3

V1 <- 1 * diag(as.numeric(2.0))
V1[1, 1] <- ((log(0.9 / 0.1) - log(0.5 / 0.5)) / 2)^2
V1[2, 2] <- (3 * mu[2, 1] / 2)^2

famfunc <- Ex_glmbfamfunc(binomial(logit))
f2 <- famfunc$f2
f3 <- famfunc$f3

dispersion2 <- as.numeric(1.0)
start <- mu
offset2 <- rep(as.numeric(0.0), length(y))
P <- solve(V1)
n <- 1000

wt2 <- wt / dispersion2
alpha <- x %*% as.vector(mu) + offset2
mu2 <- 0 * as.vector(mu)
P2 <- P
x2 <- x

parin <- start - mu
opt_out <- optim(parin, f2, f3,
  y = as.vector(y), x = as.matrix(x), mu = as.vector(mu2),
  P = as.matrix(P), alpha = as.vector(alpha), wt = as.vector(wt2),
  method = "BFGS", hessian = TRUE
)
```

```
bstar <- opt_out$par
A1 <- opt_out$hessian

Standard_Mod <- Ex_glmb_Standardize_Model(
  y = as.vector(y), x = as.matrix(x), P = as.matrix(P),
  bstar = as.matrix(bstar, ncol = 1), A1 = as.matrix(A1)
)

bstar2 <- Standard_Mod$bstar2
A <- Standard_Mod$A
x2 <- Standard_Mod$x2
mu2 <- Standard_Mod$mu2
P2 <- Standard_Mod$P2

############################################################################
# Build grid G4 via Ex_EnvelopeSize and expand.grid (as EnvelopeBuild does)
############################################################################
a <- diag(A)
omega <- (sqrt(2) - exp(-1.20491 - 0.7321 * sqrt(0.5 + a))) / sqrt(1 + a)
b2 <- as.vector(bstar2)
G1 <- rbind(b2 - omega, b2, b2 + omega)

size_info <- Ex_EnvelopeSize(a, G1, Gridtype = 3L, n = n)
G2 <- size_info$G2

G3 <- as.matrix(do.call(expand.grid, G2))
G4 <- t(G3)

############################################################################
# Ex_EnvelopeEval: negative log-likelihood and gradients at grid points
############################################################################
eval_out <- Ex_EnvelopeEval(
  G4 = G4,
  y = y,
  x = as.matrix(x2),
  mu = as.matrix(mu2, ncol = 1),
  P = as.matrix(P2),
  alpha = as.vector(alpha),
  wt = as.vector(wt2),
  family = "binomial",
  link = "logit",
  use_opencl = FALSE,
  verbose = FALSE
)

eval_out$NegLL
eval_out$cbars

############################################################################
# End of Ex_EnvelopeEval example
############################################################################
```

Ex_EnvelopeSize *Envelope Sizing and Optimization*

## Description

`Ex_EnvelopeSize()` is the high-level entry point that constructs per-dimension grids and expected draw counts, while `Ex_EnvelopeOpt()` performs the adaptive optimization used when `Gridtype = 2`.

## Usage

```
Ex_EnvelopeSize(a, G1, Gridtype = 2L, n = 1000L, n_envopt = -1,
                    use_opencl = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `a` | Numeric vector of diagonal precisions for the log-likelihood (posterior precision is $1 + a_i$). |
| `G1` | Numeric matrix of candidate grid points (3 * l1). |
| `Gridtype` | Integer code controlling grid sizing logic:<br>• 1 = static threshold test<br>• 2 = adaptive optimization via `Ex_EnvelopeOpt()`<br>• 3 = always three-point grid<br>• 4 = always single-point grid |
| `n` | Integer; number of posterior draws to generate (used for grid sizing). |
| `n_envopt` | Integer; effective sample size passed to `Ex_EnvelopeOpt`. Defaults to -1, which means "use n". |
| `use_opencl` | Logical; if `TRUE`, attempt GPU acceleration. |
| `verbose` | Logical; if `TRUE`, print progress messages. |

## Details

These functions implement the grid sizing logic used in envelope construction for rejection sampling. They make use of the theory described in (Nygren and Nygren 2006) and the general implementation outlined in (Nygren 2025).

`Ex_EnvelopeSize()` returns the constructed grid (`G2`), index vectors (`GIndex1`), expected draw count (`E_draws`), and the per-dimension grid index.

`Ex_EnvelopeOpt()` implements the adaptive optimization used in `Gridtype = 2`, ranking dimensions by posterior variance and promoting them to three-point tangents when the tradeoff is favorable.

## Value

`Ex_EnvelopeSize()` A list with components `G2`, `GIndex1`, `E_draws`, and `gridindex`.

`Ex_EnvelopeOpt()` An integer vector of length $l1$ with entries 1 (single-point) or 3 (three-point).

### Gridtype Logic and Candidates per Draw

The envelope sizing logic follows the analysis of (Nygren and Nygren 2006).

**Gridtype 1: Static Threshold** For each dimension $i$, if $\sqrt{1+a_i} \leq 2/\sqrt{\pi} \approx 1.128379$, then a single tangent at the posterior mode suffices. Expected candidates per draw in that dimension: $\sqrt{1+a_i}$. Otherwise, a symmetric three-point envelope is used at $(\theta_i^\star - \omega_i, \theta_i^\star, \theta_i^\star + \omega_i)$, with expected candidates per draw bounded above by $2/\sqrt{\pi}$.

**Gridtype 2: Adaptive Optimization** Each dimension is assigned either a single-point or three-point envelope by minimizing

$$T_{\text{total}}(g_i) = T_{\text{build}}(g_i) + T_{\text{sample}}(n, acc_i(g_i)).$$

The optimizer balances build cost (grows with number of tangents) against sampling cost (decreases as acceptance improves). Expected candidates per draw: $\prod_j \text{scaleest}_{i,j}$, where each factor is either $\sqrt{1+a_j}$ (single-point) or $2/\sqrt{\pi}$ (three-point), depending on the optimization outcome.

**Gridtype 3: Always Three-Point** Every dimension uses a symmetric three-point envelope. Expected candidates per draw:

$$\left(\tfrac{2}{\sqrt{\pi}}\right)^k$$

for $k$ dimensions, as shown in Theorem 3 of (Nygren and Nygren 2006).

**Gridtype 4: Always Single-Point** Every dimension uses a single tangent at the posterior mode. Expected candidates per draw:

$$\prod_{i=1}^{k} \sqrt{1+a_i}$$

(Example 1 in (Nygren and Nygren 2006)).

### References

Nygren K (2025). "Chapter A05: Simulation Methods – Likelihood Subgradient Densities." Vignette in the glmbayes R package. R vignette name: Chapter-A05.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

### See Also

`Ex_EnvelopeEval` for evaluating these grids. (Nygren 2025) (Nygren 2025)

`Ex_glmb_Standardize_Model` *Standardize A Non-Gaussian Model*

## Description

Standardizes a Non-Gaussian Model prior to Envelope Creation

## Usage

```
Ex_glmb_Standardize_Model(y, x, P, bstar, A1)
```

## Arguments

| | |
|---|---|
| `y` | a vector of observations of length m |
| `x` | a design matrix of dimension m*p |
| `P` | Positive-definite prior precision ($m timesm$). |
| `bstar` | a matrix containing the posterior mode from an optimization step |
| `A1` | a matrix containing the posterior precision matrix at the posterior mode |

## Details

Starts from the posterior mode quantities and proceeds in three transformations.
Step 1: posterior-precision eigendecomp.
Interim model gets identity posterior precision.
Step 2: isolate diagonal `epsilon`; remainder behaves like likelihood information.
Step 3: another eigendecomp diagonalizes data precision `A`.
The prior tied to `epsilon` becomes identity.
(Nygren and Nygren 2006)

## Value

A list with the following components

| | |
|---|---|
| `bstar2` | Standardized Posterior Mode |
| `A` | Standardized Data Precision Matrix |
| `x2` | Standardized Design Matrix |
| `mu2` | Standardized Prior Mean vector |
| `P2` | Standardized Precision Matrix Added to log-likelihood |
| `L2Inv` | A matrix used when undoing the first step in standardization described below |
| `L3Inv` | A matrix used when undoing the second step in standardization described below |

## References

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

`Ex_glmbfamfunc` *Return family functions used during simulation and post processing*

## Description

This function takes as input a `family` object and returns a set of functions that are used during simulation and summarization of models using the simulation functions in this package.

## Usage

```
Ex_glmbfamfunc(family)

## S3 method for class 'Ex_glmbfamfunc'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `family` | an object of class `family` |
| `x` | an object of class `"Ex_glmbfamfunc"` for which a printed output is desired. |
| `...` | additional optional arguments |

## Details

Compiled paths dominate;
retain `Ex_glmbfamfunc` closures for scripted workflows.

## Value

A list (class `"Ex_glmbfamfunc"`) whose first four components are **always** present for every supported `family` and `link`. The names `f1`–`f4` are stable: they mean the same roles across families (only the internal formulas change).

`f1` Neg log-likelihood in coefficients `b` (usual data args).

`f2` Neg log-posterior: likelihood plus Normal(`mu`, `P`) quadratic penalty.

`f3` Gradient of `f2` w.r.t.\ `b` (argument pattern mirrors `f2`).

`f4` Deviance gap vs saturation; honors `dispersion`;
quasi / DIC helper.

`f7` Weighted curvature / Hessian proxy at `b`.

Slots `f5` and `f6` are **not** returned: they were reserved for alternate or C++-aligned likelihood/posterior routines and remain commented out in the implementation (only `f1`, `f2`, `f3`, `f4`, and `f7` are assigned in the returned list).

`extract_library_subset` *Extract a Minimal Library Subset for a Set of Kernels*

## Description

Given one or more kernel `.cl` files and a library directory, determines the minimal set of library files required (union of all kernels' dependency annotations) and copies them — in dependency order — to a destination directory.

## Usage

```
extract_library_subset(
  kernel_paths,
  library_dir,
  dest_dir,
  depends_tag = "all_depends",
  index = NULL,
  overwrite = FALSE
)
```

## Arguments

| | |
|---|---|
| `kernel_paths` | Character vector of paths to kernel `.cl` files. Each file is scanned for the annotation tag given by `depends_tag`. |
| `library_dir` | Path to the source library directory. |
| `dest_dir` | Path where files would be written. Must already exist for any copying to occur. If absent, a warning is issued, no directories are created, and nothing is copied; the returned `data.frame` still describes the planned subset (sources, intended destinations, `copied = FALSE`). In addition to the `.cl` files, `kernel_dependency_index.rds` is copied here when copying runs so the extracted subset can be used with a pre-loaded index. |
| `depends_tag` | Name of the annotation tag listing library file stems. Defaults to `"all_depends"`. Pass `"all_depends_nmath"` for kernels that annotate their nmath dependencies with that tag. |
| `index` | Pre-loaded dependency index. If `NULL`, read from `file.path(library_dir, "kernel_dependency_index.rds")` with a `message()` nudging toward the recommended pattern. |
| `overwrite` | Logical; if `FALSE` (default) existing files in `dest_dir` are not overwritten — the copy is skipped and `copied = FALSE` in the returned data frame. |

## Details

Use this to populate a project-local copy containing only the library files actually needed by your kernels. The result can then be committed alongside your kernel files, removing a runtime dependency on the full library.

Bundled libraries such as `inst/cl/nmath` ship `kernel_dependency_index.rds` next to their `.cl` shards. Pass `index =` when you pass a pre-loaded index object to avoid redundant reads; regenerate the files with write_kernel_dependency_index, for example after porting via `nmathtools/port_inst_cl_nmath_from_src.` in the `openclport` package source tree.

```
lib_dir <- system.file("cl/nmath", package = "opencltools")
kernel_paths <- system.file(
  c("cl/src/dnorm_kernel.cl", "cl/src/pnorm_kernel.cl"),
  package = "opencltools"
)
dest_dir <- tempfile("ex_subset"); dir.create(dest_dir)
## on.exit(unlink(dest_dir, recursive = TRUE), add = TRUE)
result <- extract_library_subset(
  kernel_paths, lib_dir, dest_dir,
  depends_tag = "all_depends_nmath")
```

`extract_library_subset()` evaluates `inst/extdata/opencl_known_failures.json` against the union of `depends_tag` annotations and launcher paths.

## Value

A `nmathopencl_lib_extract_df` subclass of `data.frame` with one row per library shard (`.cl` files in dependency order, followed by companion index files when planned or copied) and columns:

`stem` Stem name (filename without `.cl`), or index filenames.

`source` Full path to the source file under `library_dir`.

`dest` Intended destination path under `dest_dir`.

`copied` `TRUE` if copied; otherwise `FALSE` including when `dest_dir` is missing (`copied = FALSE` for every row), a source path is missing, or an existing destination was skipped.

## See Also

printing methods

load_library_for_kernel

write_kernel_dependency_index

Other OpenCL kernel library subsets: `load_library_for_kernel()`, `write_kernel_dependency_index()`

## Examples

```
############################ Start of extract_library_subset example ########################

## Small library (fast; runs on CRAN check)
lib_small <- system.file("cl/nmath_small", package = "opencltools")
kpath     <- system.file("cl/src/dnorm_kernel.cl", package = "opencltools")
idx_small <- write_kernel_dependency_index(library_dir = lib_small, write = FALSE)
dest_small <- file.path(tempdir(), "opencltools_extract_small")
if (dir.exists(dest_small)) unlink(dest_small, recursive = TRUE)
dir.create(dest_small, recursive = TRUE)
on.exit(unlink(dest_small, recursive = TRUE), add = TRUE)
```

```
df_small <- extract_library_subset(
  kpath, lib_small, dest_small,
  depends_tag = "all_depends_nmath",
  index       = idx_small
)
sum(df_small$copied)


## Full nmath (slow)
lib_dir <- system.file("cl/nmath", package = "opencltools")
kernel_paths <- system.file(
  c("cl/src/dnorm_kernel.cl", "cl/src/pnorm_kernel.cl"),
  package = "opencltools"
)
idx <- write_kernel_dependency_index(library_dir = lib_dir, write = FALSE)
dest_dir <- file.path(tempdir(), "opencltools_extract_example")
if (dir.exists(dest_dir)) unlink(dest_dir, recursive = TRUE)
dir.create(dest_dir, recursive = TRUE)
on.exit(unlink(dest_dir, recursive = TRUE), add = TRUE)
df <- extract_library_subset(
  kernel_paths, lib_dir, dest_dir,
  depends_tag = "all_depends_nmath",
  index       = idx
)
print(df)
sum(df$copied)


################################################################################
## End of extract_library_subset example
################################################################################
```

---

gammafn_opencl *Special Functions (OpenCL)*

---

## Description

OpenCL-backed wrappers for selected special functions from R Mathlib.

## Usage

```
gammafn_opencl(x, fallback = FALSE, verbose = FALSE)

lgammafn_opencl(x, fallback = FALSE, verbose = FALSE)

digamma_opencl(x, fallback = FALSE, verbose = FALSE)

trigamma_opencl(x, fallback = FALSE, verbose = FALSE)
```

```
tetragamma_opencl(x, fallback = FALSE, verbose = FALSE)

pentagamma_opencl(x, fallback = FALSE, verbose = FALSE)

psigamma_opencl(x, deriv, fallback = FALSE, verbose = FALSE)

beta_opencl(a, b, fallback = FALSE, verbose = FALSE)

lbeta_opencl(a, b, fallback = FALSE, verbose = FALSE)

choose_opencl(n, k, fallback = FALSE, verbose = FALSE)

lchoose_opencl(n, k, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric vector (and additional vectors where listed); arguments are recycled to a common length like the corresponding base functions. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| deriv | Derivative order for psigamma_opencl (recycled with x). |
| a, b | Parameters for beta_opencl / lbeta_opencl (recycled together). |
| n, k | Arguments for choose_opencl / lchoose_opencl, like base::choose(n, k) (recycled together). |

## Value

Numeric vector of recycled common length.

## Examples

```
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  gammafn_opencl(x = 2.5, fallback = FALSE, verbose = TRUE)
  lgammafn_opencl(x = 2.5, fallback = FALSE, verbose = TRUE)
  digamma_opencl(x = 2.5, fallback = FALSE, verbose = TRUE)
  trigamma_opencl(x = 2.5, fallback = FALSE, verbose = TRUE)
  tetragamma_opencl(x = 2.5, fallback = FALSE, verbose = TRUE)
  pentagamma_opencl(x = 2.5, fallback = FALSE, verbose = TRUE)
  psigamma_opencl(x = 2.5, deriv = 1, fallback = FALSE, verbose = TRUE)

  beta_opencl(a = 2.5, b = 3.0, fallback = FALSE, verbose = TRUE)
  lbeta_opencl(a = 2.5, b = 3.0, fallback = FALSE, verbose = TRUE)
  choose_opencl(n = 10, k = 4, fallback = FALSE, verbose = TRUE)
  lchoose_opencl(n = 10, k = 4, fallback = FALSE, verbose = TRUE)
} else {
  base::gamma(2.5)
  base::lgamma(2.5)
```

```
  base::digamma(2.5)
  base::trigamma(2.5)
  base::psigamma(2.5, deriv = 2L)
  base::psigamma(2.5, deriv = 3L)
  base::psigamma(2.5, deriv = 1L)
  base::beta(2.5, 3.0)
  base::lbeta(2.5, 3.0)
  base::choose(10, 4)
  base::lchoose(10, 4)
}
```

---

`glmbayesEnvelopeExample` *Envelope Evaluation Utilities*

---

## Description

Core utilities for envelope-based posterior simulation using GPU-accelerated OpenCL kernels. These functions support the construction and evaluation of log-likelihood envelopes used in rejection sampling, and serve as an example of how downstream packages can build custom OpenCL kernels on top of the ported nmath routines in **nmathopencl**.

## References

There are no references for Rd macro `\insertAllCites` on this help page.

---

`imax2_opencl` *Math Support Functions (OpenCL)*

---

## Description

OpenCL-backed wrappers for miscellaneous scalar support functions.

## Usage

```
imax2_opencl(x, y, fallback = FALSE, verbose = FALSE)

imin2_opencl(x, y, fallback = FALSE, verbose = FALSE)

fmax2_opencl(x, y, fallback = FALSE, verbose = FALSE)

fmin2_opencl(x, y, fallback = FALSE, verbose = FALSE)

sign_opencl(x, fallback = FALSE, verbose = FALSE)

fprec_opencl(x, digits, fallback = FALSE, verbose = FALSE)
```

```
fround_opencl(x, digits, fallback = FALSE, verbose = FALSE)

fsign_opencl(x, y, fallback = FALSE, verbose = FALSE)

ftrunc_opencl(x, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x, y | Numeric vectors recycled together (where both appear). |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| digits | Numeric vector recycled with x for precision/rounding helpers. |

## Value

Numeric vector of recycled common length.

## Examples

```
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  imax2_opencl(x = 7, y = 3, fallback = FALSE, verbose = TRUE)
  imin2_opencl(x = 7, y = 3, fallback = FALSE, verbose = TRUE)
  fmax2_opencl(x = 7.2, y = 3.1, fallback = FALSE, verbose = TRUE)
  fmin2_opencl(x = 7.2, y = 3.1, fallback = FALSE, verbose = TRUE)
  sign_opencl(x = -2.5, fallback = FALSE, verbose = TRUE)
  fprec_opencl(x = 123.456, digits = 4, fallback = FALSE, verbose = TRUE)
  fround_opencl(x = 123.456, digits = 2, fallback = FALSE, verbose = TRUE)
  fsign_opencl(x = -2.5, y = 4.0, fallback = FALSE, verbose = TRUE)
  ftrunc_opencl(x = 123.456, fallback = FALSE, verbose = TRUE)
} else {
  as.double(pmax(7L, 3L))
  as.double(pmin(7L, 3L))
  pmax(7.2, 3.1)
  pmin(7.2, 3.1)
  base::sign(-2.5)
  signif(123.456, digits = 4)
  base::round(123.456, digits = 2)
  base::sign(-2.5) * abs(4.0)
  base::trunc(123.456)
}
```

load_library_for_kernel *Load a Minimal OpenCL Library Subset for a Single Kernel*

## Description

Given a single kernel `.cl` file and a library directory (with an associated dependency index), reads the annotation tag that lists needed library files and returns their source code concatenated in the correct dependency order.

## Usage

```
load_library_for_kernel(
  kernel_path,
  library_dir,
  depends_tag = "all_depends",
  index = NULL
)
```

## Arguments

| | |
|---|---|
| `kernel_path` | Path to a single `.cl` kernel file. The file is scanned for the annotation tag given by `depends_tag`. |
| `library_dir` | Path to the library directory containing the `.cl` source files (e.g. `system.file("cl/nmath", package = "opencltools")`). |
| `depends_tag` | Name of the annotation tag in the kernel file that lists the required library file stems. Defaults to `"all_depends"`. For kernels annotated with `@all_depends_nmath`, pass `depends_tag = "all_depends_nmath"`. |
| `index` | Optional RDS list; `NULL` triggers lazy reads. |

## Details

For repeated calls in R code, loading `kernel_dependency_index.rds` once and passing `index =` avoids redundant disk reads. The bundled `cl/nmath` directory ships `kernel_dependency_index.rds` beside the `.cl` files; use write_kernel_dependency_index to regenerate it after porting (for example via `nmathtools/port_inst_cl_nmath_from_src.R` in the `openclport` package source tree).

```
lib_dir <- system.file("cl/nmath", package = "opencltools")
kpath <- system.file("cl/src/dnorm_kernel.cl", package = "opencltools")
src <- load_library_for_kernel(
  kpath, lib_dir,
  depends_tag = "all_depends_nmath")
```

Subsets come only from each launcher file's transitive `@all_depends_nmath`-style annotations (use `attach_cross_library_tags` where dependency graphs span libraries). Deliberately loading every `.cl` under `cl/nmath` remains available via `load_kernel_library(..., "nmath")` where appropriate.

When `inst/extdata/opencl_known_failures.json` matches the launcher path or the declared / loaded stems, `warning(...)` points to fragile ports.

## Value

A character vector subclass `nmathopencl_concatenated_lib` holding concatenated sources (often length 1; blank annotations yield length-zero concatenation). Attachments describe requested and loaded library stems, paths, and byte size; see print methods.

## See Also

extract_library_subset

printing methods

write_kernel_dependency_index

Other OpenCL kernel library subsets: `extract_library_subset()`, `write_kernel_dependency_index()`

## Examples

```
############################ Start of load_library_for_kernel example ########################

## Small library (fast; runs on CRAN check)
lib_small <- system.file("cl/nmath_small", package = "opencltools")
kpath     <- system.file("cl/src/dnorm_kernel.cl", package = "opencltools")
idx_small <- write_kernel_dependency_index(library_dir = lib_small, write = FALSE)
src_small <- load_library_for_kernel(
  kpath, lib_small,
  depends_tag = "all_depends_nmath",
  index       = idx_small
)
nzchar(src_small)


## Full nmath (slow; rebuilds index over all shards)
lib_dir <- system.file("cl/nmath", package = "opencltools")
idx   <- write_kernel_dependency_index(library_dir = lib_dir, write = FALSE)
src <- load_library_for_kernel(
  kpath, lib_dir,
  depends_tag = "all_depends_nmath",
  index       = idx
)
print(src)
nzchar(src)


################################################################################
## End of load_library_for_kernel example
################################################################################
```

nmathopencl_has_opencl *GPU and OpenCL diagnostics for* **nmathopencl**

## Description

Compile-time and device-selection probes for **nmathopencl**.

Low-level workstation probes (GPU vendor detection, driver and ICD checks, PATH validation, `verify_opencl_runtime()`, PATH helpers, combined diagnostic reports, etc.) live in **opencltools**; call them as `opencltools::detect_environment_and_gpus()`, `opencltools::diagnose_glmbayes()`, and related topics documented under `?opencltools`.

## Usage

```
nmathopencl_has_opencl()

nmathopencl_opencl_device_info(force = FALSE, details = FALSE)

nmathopencl_opencl_fp64_available(force = FALSE)

nmathopencl_opencl_reset_device_selection()
```

## Arguments

| | |
|---|---|
| `force` | If `TRUE`, rerun discovery even when a previous selection is cached. |
| `details` | If `TRUE`, include a `candidates` list describing every platform/device pair (extension flag and probe result per device). |

## Details

GPU acceleration uses OpenCL kernels and `*_opencl` wrappers when `nmathopencl_has_opencl()` is `TRUE` and a suitable device is available ((Stone et al. 2010)). CPU fallbacks apply for many routines when OpenCL is absent at compile time or runtime.

For a readable host/runtime report, use `opencltools::diagnose_glmbayes()`; use `nmathopencl_has_opencl()` for this package's compile-time flag. Setup: `vignette("Chapter-01", package = "nmathopencl")`; packaged GPU API: `vignette("Chapter-12", package = "nmathopencl")`.

## Value

A list with ok (logical), `reason` (character), indices, vendor/name strings, `device_type`, `extension_cl_khr_fp64`, `probe_fp64_ok`, `selection_policy`, and optionally `candidates`.

### Functions

- `nmathopencl_opencl_device_info()`: Cached OpenCL device selection for double-precision (`cl_khr_fp64`) kernels in **nmathopencl**: enumerates platforms and devices, prefers GPU, checks the extension token, then verifies with a tiny `clBuildProgram` probe. Override with environment variables `NMATHOPENCL_PLATFORM_INDEX` and/or `NMATHOPENCL_DEVICE_INDEX` (0-based; device index is within the platform's device list). Use `nmathopencl_opencl_reset_device_selection()` to clear the cache (e.g. after driver changes).
- `nmathopencl_opencl_fp64_available()`: Returns `TRUE` if a cached OpenCL device passes the `cl_khr_fp64` extension check and build probe used for double kernels in **nmathopencl** (uses this package's compile-time OpenCL build and device cache, not **opencltools**).
- `nmathopencl_opencl_reset_device_selection()`: Clears the process-local OpenCL device selection cache so the next kernel or `nmathopencl_opencl_device_info()` run re-enumerates devices.

### Diagnostics exported from nmathopencl

- `nmathopencl_has_opencl()` — `TRUE` if this build was compiled with OpenCL support.
- `nmathopencl_opencl_device_info()`, `nmathopencl_opencl_fp64_available()` — cached double-precision device selection for kernels.
- `nmathopencl_opencl_reset_device_selection()` — clear device cache.
- `opencltools::get_opencl_core_count()` — compute units on the opencltools default device.

### Host / runtime checks (opencltools)

- `detect_environment_and_gpus()`
- `detect_compute_runtimes()`
- `verify_opencl_runtime()`
- `check_runtime_env()`
- `opencltools::diagnose_glmbayes()`
- `add_to_path_windows()` and related PATH helpers

### References

Stone JE, Gohara D, Shi G (2010). "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems." *Computing in Science & Engineering*, **12**(3), 66–72. doi:10.1109/MCSE.2010.69.

### See Also

`nmathopencl_has_opencl`, `nmathopencl_opencl_device_info`, **opencltools**.

norm_rand_opencl *OpenCL-backed RNG Core linkage checks*

## Description

Linkage wrappers for core RNG primitives used by translated Mathlib paths.

## Usage

```
norm_rand_opencl(n, fallback = FALSE, verbose = FALSE)

unif_rand_opencl(n, fallback = FALSE, verbose = FALSE)

r_unif_index_opencl(n, dn, fallback = FALSE, verbose = FALSE)

exp_rand_opencl(n, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| n | Number of observations. Non-negative integer scalar. |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE. |
| verbose | Logical; print fallback/error diagnostics. |
| dn | Positive upper bound used by r_unif_index_opencl. |

## Value

Numeric vector of length n.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  norm_rand_opencl(n, fallback = FALSE, verbose = TRUE)
  unif_rand_opencl(n, fallback = FALSE, verbose = TRUE)
  r_unif_index_opencl(n, dn = 10, fallback = FALSE, verbose = TRUE)
  exp_rand_opencl(n, fallback = FALSE, verbose = TRUE)
} else {
  stats::rnorm(n)
  stats::runif(n)
  floor(stats::runif(n, min = 0, max = 10))
  stats::rexp(n)
}
```

port_to_opencl_configure *Port an existing static src/Makevars to use OpenCL configure scripts*

## Description

Migrates a package that already has a static committed `src/Makevars` (and optionally `src/Makevars.win`) to the configure-script pattern required for CRAN-safe OpenCL support.

The function renames the existing `src/Makevars` to `src/Makevars.in` (the maintained source template) and generates `configure` (Linux/macOS) and `configure.win` (Windows) scripts that read `src/Makevars.in` at `R CMD INSTALL` time, run OpenCL detection, and write the final `src/Makevars` with OpenCL flags merged in – or copied verbatim from `src/Makevars.in` for CPU-only builds.

The generated scripts **always succeed**: if no OpenCL SDK is found the package installs cleanly as CPU-only. This is the property that makes packages safe for CRAN submission on build machines without a GPU SDK.

For packages with no existing `src/Makevars`, use `use_opencl_configure` instead. This function is for *migrating* an existing static Makevars.

## Usage

```
port_to_opencl_configure(path = ".", backup = TRUE, overwrite = FALSE)
```

## Arguments

| | |
|---|---|
| `path` | Character. Root directory of the target package. Defaults to the current working directory (`"."`). |
| `backup` | Logical. If `TRUE` (default), rename `src/Makevars` to `src/Makevars.in` before writing the configure scripts. If `FALSE`, only the configure scripts are written (the existing `src/Makevars` is left in place). |
| `overwrite` | Logical. If `TRUE`, overwrite existing configure scripts. Defaults to `FALSE`. |

## Value

Invisibly returns a character vector of the file paths written.

## src/Makevars.in workflow

After porting, **maintain** `src/Makevars.in` for your base build flags (`OpenMP`, **RcppParallel**, LAPACK, etc.). The configure script reads it at install time and appends (or omits) the OpenCL flags. The generated `src/Makevars` is a build artifact – add it to `.gitignore` and never commit it. To update base flags, edit `src/Makevars.in` and reinstall.

**configure → USE_OPENCL → has_opencl()**

```
configure / configure.win
  -> reads src/Makevars.in for base flags
  -> detects CL/cl.h + libOpenCL (+ runtime probe on Linux)
  -> writes -DUSE_OPENCL into Makevars  (or copies .in verbatim)

#ifdef USE_OPENCL in C++ source
  -> guards all GPU code; package compiles cleanly either way

has_opencl() in R
  -> mirrors the compile-time flag; TRUE only if USE_OPENCL was set
```

## Limitations

- `+=` (append) assignments in `src/Makevars` are detected and trigger a warning; review the generated configure carefully.
- If `src/Makevars` looks like a generated file (contains absolute paths, `-lOpenCL`, or `-DUSE_OPENCL`), the function warns. Run on the static committed file, not a build artifact.
- Packages that already have `configure` or `configure.win` are refused unless `overwrite = TRUE`. Users with existing configure scripts should integrate the OpenCL block manually; see `system.file("configure-templates", "README.md", package = "opencltools")`.

## See Also

`use_opencl_configure` for packages without an existing `src/Makevars`. `vignette("Chapter-02", package = "nmathopencl")` in **nmathopencl** for a full guide when building on the ported nmath kernel library.

## Examples

```
############################ Start of port_to_opencl_configure example ######################

tmp <- tempfile("port_opencl_pkg")
dir.create(tmp)
dir.create(file.path(tmp, "src"))
on.exit(unlink(tmp, recursive = TRUE), add = TRUE)

writeLines(
  c(
    "PKG_CXXFLAGS = $(SHLIB_OPENMP_CXXFLAGS)",
    "PKG_LIBS = $(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS)"
  ),
  file.path(tmp, "src", "Makevars")
)

written <- port_to_opencl_configure(path = tmp, backup = TRUE)
written
file.exists(file.path(tmp, "configure"))
file.exists(file.path(tmp, "src", "Makevars.in"))
```

```
## Regenerate configure scripts after editing Makevars.in (same temp tree)
written2 <- port_to_opencl_configure(path = tmp, backup = FALSE, overwrite = TRUE)
length(written2)

################################################################################
## End of port_to_opencl_configure example
################################################################################
```

---

`ptukey_opencl` *The Studentized Range Distribution (OpenCL)*

---

## Description

OpenCL-backed distribution and quantile wrappers for the studentized range (Tukey) distribution.

## Usage

```
ptukey_opencl(
  q,
  nmeans,
  df,
  nranges = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)

qtukey_opencl(
  p,
  nmeans,
  df,
  nranges = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  opencl_parallel = NA,
  fallback = FALSE,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `q` | Numeric vector of quantiles for `ptukey_opencl`; recycled like `stats::ptukey`. |
| `nmeans` | Number of means in each range (must be >= 2). |
| `df` | Degrees of freedom (must be > 0). |
| `nranges` | Number of groups whose maxima/minima define the range (must be >= 1). |

lower.tail, log.p
: Tail/log-*p* inputs (stats meanings).

opencl_parallel
: Dispatch hint (TRUE,FALSE,NA) for *p*/*q* wrappers on this page; parallel kernels reserved.

fallback
: When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL. Defaults TRUE temporarily ('inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBACK.md'); pass FALSE to surface failures.

verbose
: Logical; print fallback/error diagnostics.

p
: Numeric vector of probabilities for qtukey_opencl (like stats::qtukey).

## Value

Numeric vector result from ptukey_opencl or qtukey_opencl.

## Examples

```
n <- 1L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  ptukey_opencl(q = 3.4, nmeans = 5, df = 10, nranges = 1, fallback = FALSE, verbose = TRUE)
  qtukey_opencl(rep(0.8, n), nmeans = 5, df = 10, nranges = 1, fallback = FALSE, verbose = TRUE)
} else {
  stats::ptukey(3.4, nmeans = 5, df = 10, nranges = 1)
  stats::qtukey(rep(0.8, n), nmeans = 5, df = 10, nranges = 1)
}
```

---

# r_check_user_interrupt_opencl

*OpenCL-backed R_ext runtime utility linkage checks*

---

## Description

Wrappers for utility hooks used by translated R_ext-dependent kernels.

## Usage

```
r_check_user_interrupt_opencl(n, fallback = FALSE, verbose = FALSE)

r_check_stack_opencl(n, fallback = FALSE, verbose = FALSE)
```

## Arguments

n
: Number of observations. Non-negative integer scalar.

fallback
: When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to FALSE.

verbose
: Logical; print fallback/error diagnostics.

## Value

Numeric vector of length n.

## Known OpenCL limitations

On some builds, r_check_stack_opencl() can fail in device compilation or runtime due to missing host/runtime stack symbols. Use as linkage smoke only, and keep CPU fallback enabled unless explicitly debugging OpenCL failures.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  r_check_user_interrupt_opencl(n, fallback = FALSE, verbose = TRUE)

  # Known linkage/runtime gap on some setups (stack hook symbol availability):
  # r_check_stack_opencl(n, fallback = FALSE, verbose = TRUE)
} else {
  as.numeric(seq_len(n))
}
```

---

# r_pow_opencl *OpenCL-backed R Math runtime linkage checks*

---

## Description

Wrappers for low-level Mathlib runtime helpers used by translated kernels.

## Usage

```
r_pow_opencl(x, y, fallback = FALSE, verbose = FALSE)

r_pow_di_opencl(x, n_exp, fallback = FALSE, verbose = FALSE)

log1pmx_opencl(x, fallback = FALSE, verbose = FALSE)

log1pexp_opencl(x, fallback = FALSE, verbose = FALSE)

log1mexp_opencl(x, fallback = FALSE, verbose = FALSE)

lgamma1p_opencl(x, fallback = FALSE, verbose = FALSE)

pow1p_opencl(x, y, fallback = FALSE, verbose = FALSE)

logspace_add_opencl(logx, logy, fallback = FALSE, verbose = FALSE)

logspace_sub_opencl(logx, logy, fallback = FALSE, verbose = FALSE)

logspace_sum_opencl(logx, logy, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | Numeric vector(s): primary input, recycled together like stats family functions. Length-zero returns numeric(0). |
| y | Numeric vector for r_pow_opencl and pow1p_opencl (recycled against x). |
| fallback | When TRUE while nmathopencl_has_opencl() reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). For log1pmx_opencl, lgamma1p_opencl, pow1p_opencl, and the logspace_* wrappers, defaults to TRUE temporarily while pgamma_utils-stitching kernels are stabilized; see 'inst/OPENCL_PGAMMA_UTILS_KERNEL_FALLBA |
| verbose | Logical; print fallback/error diagnostics. |
| n_exp | Integer vector for r_pow_di_opencl, recycled against x. |
| logx, logy | Numeric vectors for log-space combination helpers (recycled together). |

## Details

On the GPU path, arguments are recycled to a common length len (maximum argument length, R recycling rules). Each output index runs one scalar kernel launch.

## Value

Numeric vector of the recycled common length (see Details).

## Examples

```
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  r_pow_opencl(x = 1.2, y = 2, fallback = FALSE, verbose = TRUE)
  r_pow_di_opencl(x = 1.2, n_exp = 3L, fallback = FALSE, verbose = TRUE)
  log1pmx_opencl(x = 0.2, fallback = FALSE, verbose = TRUE)
  log1pexp_opencl(x = 0.2, fallback = FALSE, verbose = TRUE)
  log1mexp_opencl(x = 0.5, fallback = FALSE, verbose = TRUE)
  lgamma1p_opencl(x = 0.2, fallback = FALSE, verbose = TRUE)
  pow1p_opencl(x = 0.2, y = 3, fallback = FALSE, verbose = TRUE)
  logspace_add_opencl(logx = -2, logy = -3, fallback = FALSE, verbose = TRUE)
  logspace_sub_opencl(logx = -2, logy = -3, fallback = FALSE, verbose = TRUE)
  logspace_sum_opencl(logx = -2, logy = -3, fallback = FALSE, verbose = TRUE)
  log1pmx_opencl(x = seq(-0.5, 0.5, by = 0.25), fallback = FALSE, verbose = TRUE)
} else {
  (1.2 + seq_len(1L) * 1e-3)^2
  1.2^3
  log1p(0.2) - 0.2
  ifelse(0.2 > 0, 0.2 + log1p(exp(-0.2)), log1p(exp(0.2)))
  ifelse(0.5 <= log(2), log(-expm1(-0.5)), log1p(-exp(-0.5)))
  lgamma(1.2)
  exp(3 * log1p(0.2))
  {
    m <- max(-2, -3)
    m + log1p(exp(min(-2, -3) - m))
  }
  -2 + log1p(-exp(-3 - (-2)))
```

```
  {
    m <- max(-2, -3)
    m + log1p(exp(min(-2, -3) - m))
  }
  {
    xv <- seq(-0.5, 0.5, by = 0.25)
    log1p(xv) - xv
  }
}
```

---

`rmultinom_opencl` *The Multinomial Distribution (OpenCL linkage subset)*

---

## Description

OpenCL-backed linkage wrapper for multinomial sampling. The current OpenCL kernel supports a 2-category multinomial parameterization via scalar `prob`, and returns a `2 x n` count matrix.

## Usage

```
rmultinom_opencl(n, size, prob, fallback = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `n` | Number of observations. Non-negative integer scalar. |
| `size` | Number of trials per draw (non-negative integer scalar). |
| `prob` | Probability of the first category in `[0, 1]`. |
| `fallback` | When `TRUE` while `nmathopencl_has_opencl()` reports OpenCL present, recover with CPU if the OpenCL call fails. Ignored when the runtime reports no OpenCL (CPU path is chosen automatically). Defaults to `FALSE`. |
| `verbose` | Logical; print fallback/error diagnostics. |

## Value

Integer matrix with 2 rows and `n` columns.

## Examples

```
n <- 5L
if (!nmathopencl_has_opencl() || identical(Sys.getenv("NOT_CRAN"), "true")) {
  rmultinom_opencl(n, size = 12L, prob = 0.4, fallback = FALSE, verbose = TRUE)
} else {
  stats::rmultinom(n, size = 12L, prob = c(0.4, 0.6))
}
```

stage_kernel_dependency_sort *Stage Kernel Library Dependency Sort Results*

## Description

Run the file-level @depends sort without requiring full success. Files that can be sorted are copied in order, and files blocked by unresolved dependencies are copied into a separate folder with CSV reports.

## Usage

```
stage_kernel_dependency_sort(library_dir, output_dir, overwrite = FALSE)
```

## Arguments

| | |
|---|---|
| library_dir | Directory containing .cl files with @depends tags. |
| output_dir | Directory where sorted and unresolved files/reports should be written. |
| overwrite | Logical; remove and recreate output_dir if it exists. |

## Value

A named list with:

sorted Data frame of files placed in dependency order (order, pass, file, source_type, depends).

unresolved Data frame of files blocked by missing dependencies.

sorted_dir, unresolved_dir Output directory paths; CSV reports 'sorted_files.csv' and 'unresolved_files.csv' are written under output_dir.

## Examples

```
############################ Start of stage_kernel_dependency_sort example ########################

lib_dir    <- system.file("cl/ex_glmbayes_nmath", package = "opencltools")
output_dir <- tempfile("stage_sort")
on.exit(unlink(output_dir, recursive = TRUE), add = TRUE)

res <- stage_kernel_dependency_sort(lib_dir, output_dir, overwrite = TRUE)
nrow(res$sorted)
length(list.files(output_dir, recursive = TRUE))

##############################################################################
## End of stage_kernel_dependency_sort example
##############################################################################
```

use_opencl_configure *Set up OpenCL configure scripts in a downstream R package*

## Description

Copies generic OpenCL `configure` and `configure.win` scripts to the root directory of a package. The scripts detect `CL/cl.h` and `libOpenCL` at compile time and generate `src/Makevars` (Linux / macOS) or `src/Makevars.win` (Windows) with or without `-DUSE_OPENCL`, depending on what is found.

The scripts **always succeed**. When no OpenCL SDK is present they produce a CPU-only Makevars with no `-lOpenCL`. This is the key property that makes packages safe for CRAN submission without requiring a GPU SDK on the build machine.

## Usage

```
use_opencl_configure(path = ".", overwrite = FALSE)
```

## Arguments

| | |
|---|---|
| `path` | Character. Root directory of the target package. Defaults to the current working directory (`"."`). |
| `overwrite` | Logical. If `TRUE`, overwrite existing configure scripts. Defaults to `FALSE` to avoid accidentally replacing a customized script. |

## Value

Invisibly returns a character vector of the file paths that were written (empty if all files were skipped).

## Why configure scripts are necessary

A package that references `-lOpenCL` or `CL/cl.h` in a static `src/Makevars` will **fail to compile** on `'CRAN'` build machines (which have no GPU SDK installed), and no binary will be produced. The configure scripts here avoid this by probing for the SDK at install time and falling back to a CPU-only build when it is absent. The relationship is:

```
configure / configure.win
  -> detects CL/cl.h + libOpenCL
  -> writes -DUSE_OPENCL into Makevars   (or omits it)

#ifdef USE_OPENCL in C++ source
  -> guards all GPU code; package compiles cleanly either way

has_opencl() in R
  -> mirrors the compile-time flag; returns TRUE only if USE_OPENCL was set
```

## See Also

port_to_opencl_configure for packages with an existing static src/Makevars. opencltoolsLdFlags for linking against this package from C++. vignette("Chapter-02", package = "nmathopencl") in **nmathopencl** for a full downstream package guide when using the ported nmath kernel library. Template source: system.file("configure-templates", package = "opencltools").

## Examples

```
############################ Start of use_opencl_configure example ######################

tmp <- tempfile("use_opencl_pkg")
dir.create(tmp)
on.exit(unlink(tmp, recursive = TRUE), add = TRUE)

written <- use_opencl_configure(path = tmp)
written
file.exists(file.path(tmp, "configure"))
file.exists(file.path(tmp, "configure.win"))

## Overwrite path (same temp tree; safe when run.dontest = TRUE)
written2 <- use_opencl_configure(path = tmp, overwrite = TRUE)
length(written2)

################################################################################
## End of use_opencl_configure example
################################################################################
```

---

# write_kernel_dependency_index

*Build and save a kernel dependency index*

---

## Description

Writes two companion index files next to the .cl files in the kernel library directory:

## Usage

```
write_kernel_dependency_index(
  library_dir = NULL,
  tags = NULL,
  output_path = NULL,
  write = TRUE,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| library_dir | Library root with annotated .cl files.<br>When tags is supplied, must agree with tags$library_dir. |
| tags | Optional attachment result from attach_kernel_dependency_tags.<br>Must have ok = TRUE when reused to skip resorting. |
| output_path | Path for the RDS file. Defaults to file.path(<library_dir>, "kernel_dependency_index.rds").<br>The .tsv file is always written to the same directory with the .tsv extension. |
| write | If FALSE, builds the index object and returns it without writing either file. |
| verbose | If TRUE, emits short messages with the output paths. |

## Details

- RDS helper shard for loaders such as load_library_for_kernel.
- Tab-separated (.tsv) stem map for C++ (rows stem<TAB>dependencies, pre-sorted).

Both files encode the same information and are always written together so they remain in sync.

## Value

Invisibly, the index list() (version schema):

- version: integer schema version.
- generated_at: timestamp from Sys.time().
- library_dir, library_name: resolved library path / basename.
- stems_ordered: stems in global load order.
- load_order: named integer vector stem -> rank.
- depends: named list stem -> character() (direct @depends).
- all_depends: named list stem -> character() (transitive dependencies, order consistent with global load order).
- n_files: file count.

## See Also

load_library_for_kernel

extract_library_subset

Other OpenCL kernel library subsets: extract_library_subset(), load_library_for_kernel()

## Examples

```
############################ Start of write_kernel_dependency_index example ########################

lib_dir <- system.file("cl/ex_glmbayes_nmath", package = "opencltools")
idx <- write_kernel_dependency_index(library_dir = lib_dir, write = FALSE)
names(idx)
```

```
length(idx$stems_ordered)
idx$n_files

################################################################################
## End of write_kernel_dependency_index example
################################################################################
```

# Index


∗ **diagnostics**
    nmathopencl_has_opencl, 65
∗ **environment**
    nmathopencl_has_opencl, 65
∗ **gpu**
    nmathopencl_has_opencl, 65
∗ **opencl**
    nmathopencl_has_opencl, 65

add_to_path_windows, *66*
attach_cross_library_tags, 5, *8, 9*, *63*
attach_kernel_call_tags, 7
attach_kernel_dependency_tags, *6*, *9*, 10, *78*

besselI, *11*
besselI_opencl, 11
besselJ_opencl *(*besselI_opencl*)*, 11
besselK, *11*
besselK_opencl *(*besselI_opencl*)*, 11
besselY_opencl *(*besselI_opencl*)*, 11
beta_opencl *(*gammafn_opencl*)*, 59

check_runtime_env, *66*
choose_opencl *(*gammafn_opencl*)*, 59

dbeta_opencl, 12
dbinom_opencl *(*dbinom_raw_opencl*)*, 14
dbinom_raw_opencl, 14
dcauchy_opencl, 17
dchisq_opencl, 18
detect_compute_runtimes, *66*
detect_environment_and_gpus, *66*
dexp_opencl, 20
df_opencl, 22
dgamma_opencl, 24
dgeom_opencl, 26
dhyper_opencl, 28
digamma_opencl *(*gammafn_opencl*)*, 59
dlnorm_opencl, 30
dlogis_opencl, 31
dnbeta_opencl *(*dbeta_opencl*)*, 12
dnbinom_mu_opencl *(*dnbinom_opencl*)*, 33
dnbinom_opencl, 33
dnorm_opencl, 36
dpois_opencl *(*dpois_raw_opencl*)*, 38
dpois_raw_opencl, 38
dsignrank_opencl, 40
dt_opencl, 42
dunif_opencl, 44
dweibull_opencl, 46
dwilcox_opencl, 47

Ex_EnvelopeEval, *3*, *5*, 49, *54*
Ex_EnvelopeSize, *51*, 53
Ex_glmb_Standardize_Model, 55
Ex_glmbfamfunc, 56
exp_rand_opencl *(*norm_rand_opencl*)*, 67
extract_library_subset, 57, *64*, *78*

family, *56*
fmax2_opencl *(*imax2_opencl*)*, 61
fmin2_opencl *(*imax2_opencl*)*, 61
fprec_opencl *(*imax2_opencl*)*, 61
fround_opencl *(*imax2_opencl*)*, 61
fsign_opencl *(*imax2_opencl*)*, 61
ftrunc_opencl *(*imax2_opencl*)*, 61

gammafn_opencl, 59
glmbayesEnvelopeExample, 61
gpu_diagnostics *(*nmathopencl_has_opencl*)*, 65

imax2_opencl, 61
imin2_opencl *(*imax2_opencl*)*, 61

lbeta_opencl *(*gammafn_opencl*)*, 59
lchoose_opencl *(*gammafn_opencl*)*, 59
lgamma1p_opencl *(*r_pow_opencl*)*, 72
lgammafn_opencl *(*gammafn_opencl*)*, 59
load_kernel_library, *63*

load_library_for_kernel, *6*, *58*, 63, *78*
log1mexp_opencl *(*r_pow_opencl*)*, 72
log1pexp_opencl *(*r_pow_opencl*)*, 72
log1pmx_opencl *(*r_pow_opencl*)*, 72
logspace_add_opencl *(*r_pow_opencl*)*, 72
logspace_sub_opencl *(*r_pow_opencl*)*, 72
logspace_sum_opencl *(*r_pow_opencl*)*, 72

nmathopencl *(*nmathopencl-package*)*, 3
nmathopencl-package, 3
nmathopencl_has_opencl, *3, 4*, *11*, *13*, *16*, *18, 19*, *21*, *23*, *25*, *27*, *29*, *31, 32*, *35*, *37*, *39*, *41*, *43*, *45*, *47, 48*, *60*, *62*, 65, *65–67*, *71*, *73, 74*
nmathopencl_opencl_device_info, *3, 4*, *66*
nmathopencl_opencl_device_info *(*nmathopencl_has_opencl*)*, 65
nmathopencl_opencl_fp64_available, *3*, *66*
nmathopencl_opencl_fp64_available *(*nmathopencl_has_opencl*)*, 65
nmathopencl_opencl_reset_device_selection, *66*
nmathopencl_opencl_reset_device_selection *(*nmathopencl_has_opencl*)*, 65
norm_rand_opencl, 67

opencltools::diagnose_glmbayes, *3*, *65, 66*
opencltools::get_opencl_core_count, *66*
opencltools::load_kernel_library, *4*
opencltoolsLdFlags, *77*

packageStartupMessage, *4*
pbeta_opencl *(*dbeta_opencl*)*, 12
pbinom_opencl *(*dbinom_raw_opencl*)*, 14
pcauchy_opencl *(*dcauchy_opencl*)*, 17
pchisq_opencl *(*dchisq_opencl*)*, 18
pentagamma_opencl *(*gammafn_opencl*)*, 59
pexp_opencl *(*dexp_opencl*)*, 20
pf_opencl *(*df_opencl*)*, 22
pgamma, *25*
pgamma_opencl, *25*
pgamma_opencl *(*dgamma_opencl*)*, 24
pgeom_opencl *(*dgeom_opencl*)*, 26
phyper_opencl *(*dhyper_opencl*)*, 28
plnorm_opencl *(*dlnorm_opencl*)*, 30
plogis_opencl *(*dlogis_opencl*)*, 31
pnbinom_mu_opencl *(*dnbinom_opencl*)*, 33
pnbinom_opencl *(*dnbinom_opencl*)*, 33
pnorm, *37, 38*
pnorm_opencl, *25*
pnorm_opencl *(*dnorm_opencl*)*, 36
port_to_opencl_configure, 68, *77*
pow1p_opencl *(*r_pow_opencl*)*, 72
ppois_opencl *(*dpois_raw_opencl*)*, 38
print methods, *64*
print(), *11*
print.Ex_glmbfamfunc *(*Ex_glmbfamfunc*)*, 56
printing methods, *58*, *64*
psigamma_opencl *(*gammafn_opencl*)*, 59
psignrank_opencl *(*dsignrank_opencl*)*, 40
pt_opencl *(*dt_opencl*)*, 42
ptukey_opencl, 70
punif_opencl *(*dunif_opencl*)*, 44
pweibull_opencl *(*dweibull_opencl*)*, 46
pwilcox_opencl *(*dwilcox_opencl*)*, 47

qbeta_opencl *(*dbeta_opencl*)*, 12
qbinom_opencl *(*dbinom_raw_opencl*)*, 14
qcauchy_opencl *(*dcauchy_opencl*)*, 17
qchisq_opencl *(*dchisq_opencl*)*, 18
qexp_opencl *(*dexp_opencl*)*, 20
qf_opencl *(*df_opencl*)*, 22
qgamma_opencl *(*dgamma_opencl*)*, 24
qgeom_opencl *(*dgeom_opencl*)*, 26
qhyper_opencl *(*dhyper_opencl*)*, 28
qlnorm_opencl *(*dlnorm_opencl*)*, 30
qlogis_opencl *(*dlogis_opencl*)*, 31
qnbinom_mu_opencl *(*dnbinom_opencl*)*, 33
qnbinom_opencl *(*dnbinom_opencl*)*, 33
qnorm_opencl *(*dnorm_opencl*)*, 36
qpois_opencl *(*dpois_raw_opencl*)*, 38
qsignrank_opencl *(*dsignrank_opencl*)*, 40
qt_opencl *(*dt_opencl*)*, 42
qtukey_opencl *(*ptukey_opencl*)*, 70
qunif_opencl *(*dunif_opencl*)*, 44
qweibull_opencl *(*dweibull_opencl*)*, 46
qwilcox_opencl *(*dwilcox_opencl*)*, 47

r_check_stack_opencl *(*r_check_user_interrupt_opencl*)*, 71
r_check_user_interrupt_opencl, 71
r_pow_di_opencl *(*r_pow_opencl*)*, 72
r_pow_opencl, 72

r_unif_index_opencl *(*norm_rand_opencl*)*, 67
rbeta_opencl *(*dbeta_opencl*)*, 12
rbinom_opencl *(*dbinom_raw_opencl*)*, 14
rcauchy_opencl *(*dcauchy_opencl*)*, 17
rchisq_opencl *(*dchisq_opencl*)*, 18
rexp_opencl *(*dexp_opencl*)*, 20
rf_opencl *(*df_opencl*)*, 22
rgamma_opencl *(*dgamma_opencl*)*, 24
rgeom_opencl *(*dgeom_opencl*)*, 26
rhyper_opencl *(*dhyper_opencl*)*, 28
rlnorm_opencl *(*dlnorm_opencl*)*, 30
rlogis_opencl *(*dlogis_opencl*)*, 31
rmultinom_opencl, 74
rnbinom_mu_opencl *(*dnbinom_opencl*)*, 33
rnbinom_opencl *(*dnbinom_opencl*)*, 33
rnorm_opencl, *37*
rnorm_opencl *(*dnorm_opencl*)*, 36
rpois_opencl *(*dpois_raw_opencl*)*, 38
rsignrank_opencl *(*dsignrank_opencl*)*, 40
rt_opencl *(*dt_opencl*)*, 42
runif_opencl *(*dunif_opencl*)*, 44
rweibull_opencl *(*dweibull_opencl*)*, 46
rwilcox_opencl *(*dwilcox_opencl*)*, 47

sign_opencl *(*imax2_opencl*)*, 61
stage_kernel_dependency_sort, 75
Sys.time(), *78*

tetragamma_opencl *(*gammafn_opencl*)*, 59
trigamma_opencl *(*gammafn_opencl*)*, 59

unif_rand_opencl *(*norm_rand_opencl*)*, 67
use_opencl_configure, *68, 69*, 76

verify_opencl_runtime, *66*

warning, *64*
write_kernel_dependency_index, *5, 6*, *58*, *63, 64*, 77