

Package: GiottoKernels (via r-universe)

September 28, 2026

Title Optional Compiled Kernels for the Giotto Suite

Version 0.2.0

Description Small compiled compute kernels for the Giotto suite. Each takes plain vectors or an Arrow C stream and returns plain vectors, and knows nothing about Giotto's classes. Every kernel is optional: the calling package checks `has_kernel()` and otherwise runs its own R implementation, so installing this package only changes speed, never results beyond floating-point rounding.

License MIT + file LICENSE

URL <https://github.com/giotto-suite/GiottoKernels>

BugReports <https://github.com/giotto-suite/GiottoKernels/issues>

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.65

Depends R (>= 4.2)

Imports nanoarrow

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libzstd-dev libclang-dev

Repository <https://giotto-suite.r-universe.dev>

Date/Publication 2026-09-28 21:46:53 UTC

RemoteUrl <https://github.com/giotto-suite/GiottoKernels>

RemoteRef HEAD

RemoteSha 7bfa0d688bdf8af0073b178bc2d924bf1772a6de

Contents

gram_stream	2
has_kernel	3
kernel_threads	4
project_stream	4
Index	6

gram_stream	<i>Gram matrix and feature sums from a stream of triplets</i>
-------------	---

Description

Computes $G = \sum_c x_c x_c^T$ and $s = \sum_c x_c$ over cells c , where x_c is cell c 's feature vector, reading sparse (row_id, col_id, value) triplets from an Arrow stream: row = cell, col = feature. Only nonzero pairs within a cell are multiplied, so the cost is the sum over cells of nnz_c^2 rather than $n_cells * n_features^2$, and memory stays at one batch plus the $n_features \times n_features$ result.

Usage

```
gram_stream(stream, n_features, n_threads = NULL)
```

Arguments

stream	anything <code>nanoarrow::as_nanoarrow_array_stream()</code> accepts: an arrow <code>RecordBatchReader</code> , a <code>nanoarrow_array_stream</code> , or a data frame. It must have columns <code>row_id</code> (int32), <code>col_id</code> (int32, 1-based feature index) and <code>value</code> (float64), without nulls; other columns are ignored. The stream is consumed.
n_features	number of features P ; every <code>col_id</code> must be in $1..n_features$.
n_threads	worker threads, resolved by <code>kernel_threads()</code> : NULL (default) uses <code>options(gkernels.n_threads)</code> , else 1. A fixed count gives bit-identical results across runs. Each thread keeps its own $n_features \times n_features$ accumulator, so the count is lowered, if needed, to keep all of them within <code>options(gkernels.memory_gb)</code> (default 4).

Details

This is the first pass of Gram-eigen PCA: centering, scaling and the eigendecomposition follow from G , s and the cell count.

Value

a list with G , the $n_features \times n_features$ symmetric Gram matrix, and s , the length- $n_features$ feature sums.

Layout the stream must have

Batch boundaries may fall anywhere, including inside a cell. What is required is that no cell interior to one batch also appears in another. A stream sorted by `row_id` always satisfies this. Two checks enforce it, and the call errors rather than returning a wrong result if either fails: each batch must be sorted by `row_id`, and batches' `row_id` ranges may overlap only at one shared edge cell.

Features within a cell need not be sorted, but each (`row_id`, `col_id`) pair must appear at most once, as in any sparse store: a repeated pair is not summed first, so its diagonal term would be wrong.

Examples

```
tri <- data.frame(
  row_id = c(1L, 1L, 2L),
  col_id = c(1L, 2L, 2L),
  value = c(1, 2, 3)
)
gram_stream(tri, n_features = 2L)
```

has_kernel

Is a kernel available in this build?

Description

Checks the kernels compiled into the installed package, so a caller can fall back on an older install instead of calling a kernel it does not have:

Usage

```
has_kernel(name)
```

Arguments

name kernel name, e.g. "gram_stream".

Details

```
use_kernel <- requireNamespace("GiottoKernels", quietly = TRUE) &&
  GiottoKernels::has_kernel("gram_stream")
```

Value

TRUE or FALSE.

Examples

```
has_kernel("gram_stream")
```

kernel_threads	<i>Resolve the thread count for a kernel call</i>
----------------	---

Description

An explicit `n_threads` is used as given. `NULL` falls back to `options(gkernels.n_threads)`, and to 1 when that is unset, so a kernel runs single-threaded unless the caller or the session asks for more.

Usage

```
kernel_threads(n_threads = NULL)
```

Arguments

`n_threads` `NULL`, or a positive whole number.

Value

a positive integer.

Examples

```
kernel_threads() # 1, or options(gkernels.n_threads)
kernel_threads(4) # 4
```

project_stream	<i>Sparse x dense product from a stream of triplets</i>
----------------	---

Description

Computes $A \% \% V$, where A is the `n_rows` x `nrow(V)` sparse matrix encoded by `(row_id, col_id, value)` triplets in an Arrow stream (`row` = cell, `col` = feature) and V is a dense matrix, one column per component. Only stored entries are touched, so the cost is `nnz * ncol(V)`, and memory is one batch plus the `n_rows` x `ncol(V)` result.

Usage

```
project_stream(stream, V, n_rows, n_threads = NULL)
```

Arguments

stream	anything <code>nanoarrow::as_nanoarrow_array_stream()</code> accepts, with columns <code>row_id</code> (int32, 1-based, at most <code>n_rows</code>), <code>col_id</code> (int32, 1-based row of <code>V</code>) and <code>value</code> (float64), without nulls. Other columns are ignored. The stream is consumed.
V	numeric matrix, one row per feature.
n_rows	number of rows in the result; rows with no stored values are 0.
n_threads	worker threads, resolved by <code>kernel_threads()</code> : NULL (default) uses <code>options(gkernels.n_threads)</code> , else 1.

Details

This is the second pass of Gram-eigen PCA: cell coordinates from the loadings. Centering is left to the caller, since it is one subtraction per component: $(A - 1 \mu^T) V = A V - 1 (\mu^T V)$.

Value

the `n_rows` x `ncol(V)` matrix `A %*% V`.

Layout the stream must have

The same as `gram_stream()`: batch boundaries may fall anywhere, but no cell interior to one batch may also appear in another, which a stream sorted by `row_id` always satisfies. The call checks this and errors rather than return a wrong result. Each `(row_id, col_id)` pair must appear at most once.

Every output row is built by one thread from its entries in stream order, so the result is bit-identical at any thread count.

Examples

```
tri <- data.frame(
  row_id = c(1L, 1L, 3L),
  col_id = c(1L, 2L, 2L),
  value  = c(1, 2, 3)
)
project_stream(tri, V = diag(2), n_rows = 3L)
```

Index

`gram_stream`, [2](#)

`gram_stream()`, [5](#)

`has_kernel`, [3](#)

`kernel_threads`, [4](#)

`kernel_threads()`, [2](#), [5](#)

`nanoarrow::as_nanoarrow_array_stream()`,
[2](#), [5](#)

`project_stream`, [4](#)