Package: vecvec (via r-universe)

November 21, 2026

Title Construct Mixed Type Data Structures with Vectors of Vectors

Version 1.5.0

Description Mixed type vectors are useful for combining semantically similar classes. Some examples of semantically related classes include time across different granularities (e.g. daily, monthly, annual) and probability distributions (e.g. Normal, Uniform, Poisson). These groups of vector types typically share common statistical operations which vary in results with the attributes of each vector. The 'vecvec' data structure facilitates efficient storage and computation across multiple vectors within the same object.

License MIT + file LICENSE

Imports cli, rlang, S7 (>= 0.2.2), stats, vctrs

Suggests lifecycle, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

URL https://pkg.mitchelloharawild.com/vecvec/, https://github.com/mitchelloharawild/vecvec

BugReports https://github.com/mitchelloharawild/vecvec/issues

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Mitchell O'Hara-Wild [aut, cre] (ORCID: <https://orcid.org/0000-0001-6729-7695>)

Maintainer Mitchell O'Hara-Wild <mail@mitchelloharawild.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-21 10:30:02 UTC

RemoteUrl https://github.com/cran/vecvec

RemoteRef HEAD

RemoteSha 1a638896b8d8a706afa8f51c80158b3102090af6

Contents

class_vecvec 2
is_vecvec 3
masked-cor 4
unvecvec 4
vecvec 5
vecvec_abind 6
vecvec_apply 7
vecvec_apply_fn 8
vecvec_mapply 9
vecvec_register 10

Index **12**

`class_vecvec` *S7 class for vecvec*

Description

`class_vecvec()` constructs a new vecvec object from a list of vectors. It is designed to be performant with minimal checks on the inputs. Direct use of `class_vecvec()` is useful for developers, but for users it is recommended to use `vecvec()` to create vecvec objects.

Usage

```
class_vecvec(x = list(), i = seq_len(sum(list_sizes(x))))
```

Arguments

`x`	An unnamed list of arbitrary vectors. The vectors can be of different types and lengths, but they must all be vectors (according to vctrs::vec_is). Adjacent vectors of the same type will be combined together into a single vector.
`i`	A vector of integers specifying the location of each element in `x` as if they were combined in order. The values in `i` must be between 1 and the total number of elements across all vectors in `x`, and can contain duplicates. If not provided, it defaults to a sequence from 1 to the total number of elements across all vectors in `x`.

Value

A vector of vectors of S7 object of class `class_vecvec`.

Examples

```
# Create a vecvec prototype
class_vecvec()

# Construct a vecvec from a list of vectors
class_vecvec(list(letters, rnorm(10)))

# Fully specify a vecvec with locations
class_vecvec(
  x = list(letters, rnorm(10)),
  i = c(1:3, 27:31, 26:4, 32:36)
)
```

---

is_vecvec *Test if an object is a vecvec*

Description

Test if an object is a vecvec

Usage

```
is_vecvec(x)
```

Arguments

x	Object to test.

Value

TRUE if x inherits from `class_vecvec`, FALSE otherwise.

See Also

`vecvec()` to create a vecvec object.

Examples

```
vv <- vecvec(1:3, letters)
is_vecvec(vv)
is_vecvec(1:3)
```

masked-cor *Correlation, Variance and Covariance*

Description

var, cov, and cor compute the variance of x and the covariance or correlation of x and y if these are vectors. If x and y are matrices then the covariances (or correlations) between the columns of x and the columns of y are computed.

These functions mask and wrap the stats::var, stats::cov, and stats::cor functions to add support for vecvec objects. More details can be found in the documentation for those functions.

Usage

```
var(x, ...)

cov(x, ...)

cor(x, ...)
```

Arguments

x	a numeric vector, matrix or data frame.
...	Additional arguments passed to stats::var, stats::cov, and stats::cor.

See Also

stats::var(), stats::cov(), stats::cor()

unvecvec *Coerce a vector of vectors to a single typed vector*

Description

[Stable]

Flattens a vecvec into a plain R vector by casting all elements to a common type. This is the inverse of vecvec().

Type resolution follows vctrs coercion rules: when ptype is NULL the common type is determined automatically from the slots of x via vctrs::vec_ptype_common(). If no common type can be found (e.g. all slots are empty), the result falls back to logical().

Usage

```
unvecvec(x, ptype = NULL)
```

Arguments

x	A vecvec object.
ptype	A prototype specifying the desired output type, e.g. character() or numeric(). If NULL (the default), the common type is inferred from the elements of x using vctrs::vec_ptype_common(), falling back to logical() when no common type can be determined. Passing an explicit ptype is useful when you need a guaranteed output type regardless of what x contains, or when automatic inference would pick an undesirable type.

Value

A vector of length length(x) and type ptype (or the inferred common type when ptype = NULL). Positions corresponding to NA indices in the underlying vecvec structure are filled with NA.

See Also

vecvec() to create a vecvec; vctrs::vec_ptype_common() for the type inference rules; vctrs::vec_cast() for the casting rules.

Examples

```
vv <- vecvec(1:3, c(4.5, 5.5))

# Automatic type inference: integer + double -> double
unvecvec(vv)

# Force a specific output type
unvecvec(vv, ptype = character())
```

---

vecvec *Create a vector of vectors*

Description

[Stable]

A vecvec is a vector that holds elements of different types without coercing them to a common type. Unlike a list of vectors, a vecvec behaves as a flat vector (hence vector of vectors). This means that you can operations (such as indexing, arithmetic, and statistics) apply across all elements of a vecvec as if they were combined into a single vector.

Mixed-type vectors can be useful if you need to store heterogeneous data in a vector-like structure. In most cases, I believe this is bad practice for data analytics, but this could be useful for tidying up messy data. The most valuable use case for vecvec is as a data structure for mixed-type semantic vectors. This package is used by mixtime and distributional to create vectors of time with different chronons and distributions with different shapes.

To convert a vecvec back to a plain typed vector, use unvecvec(), which casts all elements to a common type via vctrs::vec_cast().

Usage

```
vecvec(...)
```

Arguments

`...`	Vectors to combine. Each vector is stored as a separate typed slot; no type coercion is performed.

Value

A `vecvec` object whose length equals the total number of elements across all input vectors.

See Also

`unvecvec()` to coerce a `vecvec` to a single-typed vector; `is_vecvec()` to test whether an object is a `vecvec`.

Examples

```
# Mixed types are preserved without coercion
vv <- vecvec(Sys.Date(), rnorm(3), letters)
vv

# .[i] Indexing works like a flat vector
vv[c(1L, 3L, 7L)]

# .[[i]] drops to the original vector type
vv[[2L]]
```

`vecvec_abind` *Combine vecvec arrays along an arbitrary dimension*

Description

[Experimental]

`vecvec_abind()` generalises `cbind()`/`rbind()` to arbitrary-dimensional arrays, combining `...` along the dimension given by `along`. It underlies the `cbind()`/`rbind()` methods for `vecvec` (`along = 2` and `along = 1` respectively), and can also be used directly to stack higher-dimensional `vecvec` arrays, e.g. combining two matrices into a 3D array with `along = 3`.

Inputs that aren't already `vecvec` objects are first wrapped with `vecvec()`. An input with no `dim` (a plain vector) is treated as having a single dimension of length 1 at `along`, so a bare vector becomes a single column (`along = 2`), row (`along = 1`), or slice (`along > 2`) - matching how `cbind()`/`rbind()` treat plain vectors. Every dimension other than `along` must match (after this padding) across all inputs.

Usage

```
vecvec_abind(..., along = 1L)
```

Arguments

`...`	Vectors or dim-annotated `vecvec` arrays to combine; wrapped with `vecvec()` automatically if not already `vecvec` objects. Named arguments (e.g. `vecvec_abind(a = x, b = y)`) name the corresponding position(s) along `along`, similar to `cbind()`/`rbind()`.
`along`	The dimension along which to combine `...`. Defaults to 1 (row-wise, as `rbind()`). The result always has at least `max(2, along)` dimensions, so binding plain vectors produces a matrix (like `cbind()`/`rbind()`) rather than a flat vector - use `c()` for flat concatenation.

Value

A `vecvec` array whose dimension at `along` is the sum of the `along` dimensions of `...` (after padding), and whose other dimensions match `...`.

See Also

`vecvec()`; `c()` for flat (dim-less) concatenation of `vecvec` objects.

Examples

```
# cbind()-like: bind columns
vecvec_abind(vecvec(1:3), vecvec(4:6), along = 2)

# rbind()-like: bind rows (the default)
vecvec_abind(vecvec(1:3), vecvec(4:6))

# Stack two matrices into a 3D array
m1 <- array(vecvec(1:6), dim = c(2, 3))
m2 <- array(vecvec(7:12), dim = c(2, 3))
vecvec_abind(m1, m2, along = 3)
```

`vecvec_apply` *Apply a function to each vector of the vecvec*

Description

The `vecvec_apply()` function applies a function `.f` to each vector in the `vecvec` vectors.

Usage

```
vecvec_apply(x, .f, ...)
```

Arguments

x	A vecvec object
.f	A function to apply to each vector
...	Additional arguments passed to .f

Value

A vecvec data type with the same structure as x but with each vector transformed by .f.

---

vecvec_apply_fn *Function factory for vecvec_apply*

Description

The vecvec_apply_fn() function is a function factory that creates applies the function .f to each vector and optionally simplifies the result with [unvecvec()]. The function matches the forms of the original function .f and can be used to define methods for generic functions that apply to vecvec objects. If .f is a primitive function, the resulting function will have a apply over an argument x and pass through ....

Usage

```
vecvec_apply_fn(.f, ptype = NULL, SIMPLIFY = !is.null(ptype))
```

Arguments

.f	A function to apply to each vector
ptype	A prototype to simplify to. If NULL, the result will be a vecvec object. If not NULL, the result will be simplified to the type of ptype if possible.
SIMPLIFY	If TRUE, the [unvecvec()] will be applied to the result of vecvec_apply(), using ptype as the target type. If FALSE, the result will always be a vecvec object.

Value

A function that applies .f to each vector of a vecvec object and optionally simplifies the result.

vecvec_mapply *Apply a vectorised function across multiple vectors of a vecvec, batched by shared underlying storage slots*

Description

The `vecvec_mapply()` function applies a function `.f` across the vectors in `.l`, combining the results into a vecvec that preserves the original (possibly heterogeneous) type of each element of any `vecvec` inputs. Unlike `mapply()`/`Map()`, `.f` is not called once per logical element - it is called once per contiguous run of positions that draw from the same combination of underlying storage slots across every vector in `.l` (the same slot-grouping used internally for `Ops.vecvec::vecvec`, e.g. `+`/`==`). Each call therefore receives one vector per element of `.l` (that group's values) rather than one scalar per element of `.l`, so `.f` must itself be vectorised - an ordinary operator or vectorised function (e.g. `+`, `paste0`) qualifies automatically, but `.f` must return a result the same length as its inputs, or `vecvec_mapply()` errors.

Usage

```
vecvec_mapply(.l, .f, ..., ptype = NULL, SIMPLIFY = !is.null(ptype))
```

Arguments

`.l`	A list of vectors to apply `.f` over. Elements of `.l` may be `vecvec` objects or plain vectors; plain vectors are treated as if wrapped with `vecvec()`. All vectors in `.l` are recycled to a common size following `vctrs::vec_recycle_common()` rules.
`.f`	A vectorised function, applied once per group of positions that share the same underlying storage slots across every vector in `.l`. Called with one vector per element of `.l` (that group's values) plus `...`.
`...`	Additional arguments passed to every call of `.f`.
`ptype`	A prototype to simplify the result to with `unvecvec()`. If `NULL` and `SIMPLIFY` is `TRUE`, the common type is inferred from the result as in `unvecvec()`.
`SIMPLIFY`	If `TRUE`, `unvecvec()` is applied to the result, using `ptype` as the target type. If `FALSE`, the result is always a `vecvec` object.

Details

Because the number of calls to `.f` scales with the number of distinct slot combinations rather than with `length(.l[[1]])`, this is cheap for structured or replicated inputs (few distinct slot combinations) and degrades toward one call per element only in the worst case (every position drawing from a different combination of slots) - the same cost profile as `vecvec_apply()` and `Ops.vecvec::vecvec`, and unlike a plain `mapply()`/`Map()` call, which always calls `.f` once per element.

Value

A vecvec object combining the results of calling .f on each group of positions of .l, or (if SIMPLIFY is TRUE) that result simplified to a single type with unvecvec(). Positions where any input has a missing (NA) index form their own group and are not passed to .f at all - they come back as NA directly, matching how missing elements are treated elsewhere (e.g. vecvec_apply(), [<-, is.na<-).

See Also

vecvec_apply() for the single-input equivalent, applying .f once per underlying storage vector of a single vecvec. Ops.vecvec::vecvec uses the same slot-grouping strategy for arithmetic/comparison operators.

---

vecvec_register *Register methods for a vecvec subclass*

Description

Call vecvec_register() inside your package's .onLoad() function to register the S3 methods that allow a custom vecvec subclass to participate in the vctrs vector system.

Usage

```
vecvec_register(
  x,
  f_ptype2 = vec_ptype2_vecvec,
  f_cast_to = vec_cast_to_vecvec,
  f_cast_from = vec_cast_from_vecvec
)
```

Arguments

x	An S7 class object that extends class_vecvec. Typically the class object created by S7::new_class() in your package, e.g. class_myvec.
f_ptype2	The function to register as the vec_ptype2 method for this class. Defaults to vec_ptype2_vecvec, guarantees that the common type of two vectors is a vecvec vector.
f_cast_to	The function to register as the vec_cast.<class>.* methods for this class. Defaults to vec_cast_to_vecvec, which attempts to cast other types to vecvec by calling vecvec().
f_cast_from	The function to register as the vec_cast.*.<class> methods for this class. Defaults to vec_cast_from_vecvec, which attempts to cast vecvec to other types by calling unvecvec().

Details

Specifically, `vecvec_register()` registers:

- `vec_cast.<class>.*` methods, so that other types can be cast *to* your vecvec subclass.
- `vec_cast.*.<class>` methods, so that your vecvec subclass can be cast *from* other types.

Value

`NULL`, invisibly. Called for its side effects.

Examples

```
## Not run:
# In your package, define a vecvec subclass:
class_myvec <- S7::new_class(
  "myvec",
  parent = vecvec::class_vecvec
)

# Then register it in .onLoad():
.onLoad <- function(libname, pkgname) {
  S7::methods_register()
  vecvec::vecvec_register(class_myvec)
}

## End(Not run)
```

Index

c(), *7*
cbind(), *6, 7*
class_vecvec, 2, *10*
cor *(*masked-cor*)*, 4
cov *(*masked-cor*)*, 4

is_vecvec, 3
is_vecvec(), *6*

masked-cor, 4

rbind(), *6, 7*

S7::new_class(), *10*
stats::cor, *4*
stats::cor(), *4*
stats::cov, *4*
stats::cov(), *4*
stats::var, *4*
stats::var(), *4*

unvecvec, 4
unvecvec(), *5, 6*, *9, 10*

var *(*masked-cor*)*, 4
vctrs::vec_cast(), *5*
vctrs::vec_is, *2*
vctrs::vec_ptype_common(), *4, 5*
vctrs::vec_recycle_common(), *9*
vecvec, 5, *10*
vecvec(), *2–7*, *9*
vecvec_abind, 6
vecvec_apply, 7
vecvec_apply(), *9, 10*
vecvec_apply_fn, 8
vecvec_mapply, 9
vecvec_register, 10