# Package 'intervals'

October 13, 2022

**Version** 0.15.2

**Type** Package

**Title** Tools for Working with Points and Intervals

**Author** Richard Bourgon <bourgon.richard@gene.com>

**Maintainer** Edzer Pebesma <edzer.pebesma@uni-muenster.de>

**Depends** R (>= 2.9.0)

**Imports** utils, graphics, methods

**Description** Tools for working with and comparing sets of points and intervals.

**License** Artistic-2.0

**LazyLoad** yes

**URL** http://github.com/edzer/intervals

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2020-04-04 10:00:02 UTC

## R topics documented:

---

`intervals-package` *Tools for working with points and intervals*

---

## Description

Tools for working with and comparing sets of points and intervals.

## Details

Index:

`Intervals-class` Classes `"Intervals"` and `"Intervals_full"`.

`Intervals_virtual-class` Class `"Intervals_virtual"`.

`Intervals_virtual_or_numeric-class` Class union `"Intervals_virtual_or_numeric"`.

`as.matrix` Coerce endpoints to a matrix.

`c` Concatenate different sets of intervals.

`close_intervals` Re-represent integer intervals with open or closed endpoints.

`closed` Accessor for `closed` slot: closure vector/matrix.

`clusters` Identify clusters in a collection of positions or intervals.

`contract` Contract sets.

`distance_to_nearest` Compute distance to nearest position in a set of intervals.

`empty` Identify empty interval rows.

`expand` Expand sets.

`interval_complement` Compute the complement of a set of intervals.

`interval_difference` Compute set difference.

`interval_included` Assess inclusion of one set of intervals with respect to another.

`interval_intersection` Compute the intersection of one or more sets of intervals.

`interval_overlap` Assess which query intervals overlap which targets.

`interval_union` Compute the union of intervals in one or more interval matrices.

`is.na` Identify interval rows with `NA` endpoints.

`plot` S3 plotting methods for intervals objects.

`reduce` Compactly re-represent the points in a set of intervals.

`sgd` Yeast gene model sample data.

`size` Compute interval sizes.

`split` Split an intervals object according to a factor.

`type` Accessor for `type` slot: Z or R.

`which_nearest` Identify nearest member(s) in a set of intervals.

Further information is available in the following vignettes:

`intervals_overview` Overview of the intervals package.

## Acknowledgments

Thanks to Julien Gagneur, Simon Anders, and Wolfgang Huber for numerous helpful suggestions about the package content and code.

## Author(s)

Richard Bourgon <bourgon@ebi.ac.uk>

## See Also

See the **genomeIntervals** package in Bioconductor, which extends the functionality of this package.

---

`as.matrix` *Extract matrix of endpoints*

---

## Description

S3 and S4 methods for extracting the matrix of endpoints from S4 objects.

## Usage

```
## S3 method for class 'Intervals_virtual'
as.matrix(x, ...)

## S4 method for signature 'Intervals_virtual'
as.matrix(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | `"Intervals"` or `"Intervals_full"` objects. |
| `...` | Unused, but required by the S3 generic. |

## Value

A two-column matrix, equivalent to `x@.Data` or `as(x, "matrix")`.

c *Combine different interval matrix objects*

## Description

S3 methods for concatenating sets of intervals into a single set.

## Usage

```
## S3 method for class 'Intervals'
c(...)
## S3 method for class 'Intervals_full'
c(...)
```

## Arguments

| | |
|---|---|
| ... | "Intervals" or "Intervals_full" objects. |

## Details

All objects are expected to have the same value in the type slot. If the closed slots differ for "Intervals" objects and type == "Z", the objects will be adjusted to have closed values matching that of x; if type == "R", however, then all objects must first be coerced to class "Intervals_full", with a warning. This coercion also occurs when a mixture of object types is passed in. A NULL in any argument is ignored.

## Value

A single "Intervals" or "Intervals_full" object. Input objects are concatenated in their order of appearance in the the argument list.

If any input argument is not a set of intervals, list(...) is returned instead.

## Note

These methods will be converted to S4 once the necessary dispatch on ... is supported.

## Examples

```
f1 <- Intervals( 1:2, type = "Z" )
g1 <- open_intervals( f1 + 5 )

# Combining Intervals objects over Z may require closure adjustment
c( f1, g1 )

f2 <- f1; g2 <- g1
type( f2 ) <- type( g2 ) <- "R"

# Combine Intervals objects over R which have different closure requires
```

```
# coercion

h <- c( f2, g2 )

# Coercion for mixed combinations as well
c( h, g2 + 10 )

## Not run:
# Combining different types is not permitted
c( h, g1 + 10 )

## End(Not run)
```

---

`close_intervals` *Re-represent integer intervals with open or closed endpoints*

---

## Description

Given an integer interval matrix, adjust endpoints so that all intervals have the requested closure status.

## Usage

```
## S4 method for signature 'Intervals_virtual'
close_intervals(x)

## S4 method for signature 'Intervals_virtual'
open_intervals(x)

## S4 method for signature 'Intervals'
adjust_closure(x, close_left = TRUE, close_right = TRUE)

## S4 method for signature 'Intervals_full'
adjust_closure(x, close_left = TRUE, close_right = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | An object of appropriate class, and for which `x@type == "Z"`. If `x@type == "R"`, an error is generated. |
| `close_left` | Should the left endpoints be closed or open? |
| `close_right` | Should the right endpoints be closed or open? |

## Value

An object of the same class as `x`, with endpoints adjusted as necessary and all `closed(x)` set to either `TRUE` or `FALSE`, as appropriate.

## Note

The `close_intervals` and `open_intervals` are for convenience, and just call `adjust_closure` with the approriate arguments.

The `x` object may contain empty intervals, with at least one open endpoint, and still be valid. (Intervals are invalid if their second endpoint is less than their first.) The `close_intervals` method would, in such cases, create an invalid result; to prevent this, empty intervals are detected and removed, with a warning.

This package does not make a distinction between closed and open infinite endpoints: an interval with an infinite endpoint extends to (plus or minus) infinity regardless of the closure state. For example, `distance_to_nearest` will return a `0` when `Inf` is compared to both `"[0, Inf)"` and `"[0, Inf]"`.

## Examples

```
x <- Intervals(
               c( 1, 5, 10, 1, 6, 20 ),
               closed = c( TRUE, FALSE ),
               type = "Z"
               )

# Empties are dropped
close_intervals(x)
adjust_closure(x, FALSE, TRUE)

# Intervals_full
y <- as( x, "Intervals_full" )
closed(y)[1,2] <- TRUE
open_intervals(y)
```

---

`clusters` *Identify clusters in a collection of positions or intervals*

---

## Description

This function uses tools in the **intervals** package to quickly identify clusters – contiguous collections of positions or intervals which are separated by no more than a given distance from their neighbors to either side.

## Usage

```
## S4 method for signature 'numeric'
clusters(x, w, which = FALSE, check_valid = TRUE)

## S4 method for signature 'Intervals_virtual'
clusters(x, w, which = FALSE, check_valid = TRUE)
```

## Arguments

| | |
|---|---|
| x | An appropriate object. |
| w | Maximum permitted distance between a cluster member and its neighbors to either side. |
| which | Should indices into the x object be returned instead of actual subsets? |
| check_valid | Should validObject be called before passing to compiled code? Also see interval_overlap and reduce. |

## Details

A cluster is defined to be a maximal collection, with at least two members, of components of x which are separated by no more than w. Note that when x represents intervals, an interval must actually *contain a point* at distance w or less from a neighboring interval to be assigned to the same cluster. If the ends of both intervals in question are open and exactly at distance w, they will not be deemed to be cluster co-members. See the example below.

## Value

A list whose components are the clusters. Each component is thus a subset of x, or, if which == TRUE, a vector of indices into the x object. (The indices correspond to row numbers when x is of class "Intervals_virtual".)

## Note

Implementation is by a call to reduce followed by a call to interval_overlap. The clusters methods are included to illustrate the utility of the core functions in the **intervals** package, although they are also useful in their own right.

## Examples

```
# Numeric method
w <- 20
x <- sample( 1000, 100 )
c1 <- clusters( x, w )

# Check results
sapply( c1, function( x ) all( diff(x) <= w ) )
d1 <- diff( sort(x) )
all.equal(
          as.numeric( d1[ d1 <= w ] ),
          unlist( sapply( c1, diff ) )
          )

# Intervals method, starting with a reduced object so we know that all
# intervals are disjoint and sorted.
B <- 100
left <- runif( B, 0, 1e4 )
right <- left + rexp( B, rate = 1/10 )
y <- reduce( Intervals( cbind( left, right ) ) )
```

```
gaps <- function(x) x[-1,1] - x[-nrow(x),2]
hist( gaps(y), breaks = 30 )

w <- 200
c2 <- clusters( y, w )
head( c2 )
sapply( c2, function(x) all( gaps(x) <= w ) )

# Clusters and open end points. See "Details".
z <- Intervals(
               matrix( 1:4, 2, 2, byrow = TRUE ),
               closed = c( TRUE, FALSE )
               )
z
clusters( z, 1 )
closed(z)[1] <- FALSE
z
clusters( z, 1 )
```

---

`distance_to_nearest` *Compute distance to nearest position in a set of intervals*

---

## Description

For each point or interval in the `from` argument, compute the distance to the nearest position in the `to` argument.

## Usage

```
## S4 method for signature
## 'Intervals_virtual_or_numeric,Intervals_virtual_or_numeric'
distance_to_nearest(from, to, check_valid = TRUE)
```

## Arguments

| | |
|---|---|
| `from` | An object of appropriate type. |
| `to` | An object of appropriate type. |
| `check_valid` | Should `validObject` be called before passing to compiled code? Also see `interval_overlap`. |

## Value

A vector of distances, with one entry per point or interval in `from`. Any intervals in `from` which are either empty (see `empty`) or have `NA` endpoints produce a `NA` result.

## Note

This function is now just a wrapper for `which_nearest`.

## See Also

See `which_nearest`, which also returns indices for the interval or intervals (in case of ties) at the distance reported.

## Examples

```
# Point to interval

to <- Intervals( c(0,5,3,Inf) )
from <- -5:10
plot( from, distance_to_nearest( from, to ), type = "l" )
segments( to[,1], 1, pmin(to[,2], par("usr")[2]), 1, col = "red" )

# Interval to interval

from <- Intervals( c(-Inf,-Inf,3.5,-1,1,4) )
distance_to_nearest( from, to )
```

---

`empty` *Identify empty interval rows*

---

## Description

A valid interval matrix may contain empty intervals: those with common endpoints, at least one of which is open. The `empty` method identifies these rows.

## Usage

```
## S4 method for signature 'Intervals'
empty(x)

## S4 method for signature 'Intervals_full'
empty(x)
```

## Arguments

| | |
|---|---|
| `x` | An `"Intervals"` or `"Intervals_full"` object. |

## Details

Intervals are deemed to be empty when their endpoints are equal and not both closed, or for `type == "Z"`, when their endpoints differ by 1 and both are open. The matrices `x` and `x[!empty(x),]` represent the same subset of the integers or the real line.

## Value

A boolean vector with length equal to `nrow(x)`.

### Warning

Exact equality (==) comparisons are used by `empty`. See the package vignette for a discussion of equality and floating point numbers.

### Note

Note that intervals of size 0 may not be empty over the reals, and intervals whose second endpoint is strictly greater than the first *may* be empty over the integers, if both endpoints are open.

### See Also

See `size` to compute the size of each interval in an object.

### Examples

```
z1 <- Intervals( cbind( 1, 1:3 ), type = "Z" )
z2 <- z1; closed(z2)[1] <- FALSE
z3 <- z1; closed(z3) <- FALSE

empty(z1)
empty(z2)
empty(z3)

r1 <- z1; type(r1) <- "R"
r2 <- z2; type(r2) <- "R"
r3 <- z3; type(r3) <- "R"

empty(r1)
empty(r2)
empty(r3)

s1 <- Intervals_full( matrix( 1, 3, 2 ), type = "Z" )
closed(s1)[2,2] <- FALSE
closed(s1)[3,] <- FALSE

empty(s1)
```

---

## expand *Expand or contract intervals*

---

### Description

It is often useful to shrink or grow each interval in a set of intervals: to smooth over small, uninteresting gaps, or to address possible imprecision resulting from floating point arithmetic. The `expand` and `contract` methods implement this, using either absolute or relative difference.

## Usage

```
## S4 method for signature 'Intervals_virtual'
expand(x, delta = 0, type = c("absolute", "relative"))

## S4 method for signature 'Intervals_virtual'
contract(x, delta = 0, type = c("absolute", "relative"))
```

## Arguments

| | |
|---|---|
| x | An "Intervals" or "Intervals_full" object. |
| delta | A non-negative adjustement value. A vector is permitted, and its entries will be recycled if necessary. |
| type | Should adjustment be based on relative or absolute difference. When type == "relative" intervals are expanded/contracted to include/exclude points for which a relative difference with respect to the nominal value is less than or equal to delta. (See the note below.) When type == "absolute", absolute rather than relative difference is used, i.e., all intervals are expanded or contracted by the same amount. |

## Value

A single object of appropriate class, with endpoint positions adjusted as requested. Expansion returns an object with the same dimension as x; contraction may lead to the elimination of now-empty rows.

## Note

Here, the relative difference between $x$ and $y$ is $|x - y|/\max(|x|, |y|)$.

## Examples

```
# Using adjustment to remove small gaps

x <- Intervals( c(1,10,100,8,50,200), type = "Z" )
close_intervals( contract( reduce( expand(x, 1) ), 1 ) )

# Finding points for which, as a result of possible floating point
# error, intersection may be ambiguous. Whether y1 intersects y2[2,]
# depends on precision.

delta <- .Machine$double.eps^0.5
y1 <- Intervals( c( .5, 1 - delta / 2 ) )
y2 <- Intervals( c( .25, 1, .75, 2 ) )

# Nominal

interval_intersection( y1, y2 )

# Inner limit
```

```
inner <- interval_intersection(
                              contract( y1, delta, "relative" ),
                              contract( y2, delta, "relative" )
                              )

# Outer limit

outer <- interval_intersection(
                              expand( y1, delta, "relative" ),
                              expand( y2, delta, "relative" )
                              )

# The ambiguous set, corresponding to points which may or may not be in
# the intersection -- depending on numerical values for endpoints
# which are, with respect to relative difference, indistinguishable from
# the nominal values.

interval_difference( outer, inner )
```

---

`Intervals-class` *Classes "Intervals" and "Intervals_full"*

---

## Description

`"Intervals"` objects are two-column matrices which represent sets, possibly non-disjoint and in no particular order, of intervals on either the integers or the real line. All intervals in each object have the same endpoint closure pattern. `"Intervals_full"` objects are similar, but permit interval-by-interval endpoint closure specification.

## Objects from the Class

Objects can be created by calls of the form `new("Intervals", ...)`, or better, by using the constructor functions `Intervals(...)` and `Intervals_full(...)`.

## Slots

**`.Data`:** See `"Intervals_virtual"`.

**`closed`:** For `"Intervals"` objects, a two-element logical vector. For `"Intervals_full"` objects, a two-column logical matrix with the same dimensions as `.Data`. If omitted in a `new` call, the `closed` slot will be initialized to an object of appropriate type and size, with all entries `TRUE`. If `closed` is a vector of length 1, or a vector of length 2 for the `"Intervals_full"` class, an appropriate object will be made by reusing the supplied values row-wise. See the example below.

**`type`:** See `"Intervals_virtual"`.

## Extends

Class "Intervals_virtual", directly.

Class "matrix", by class "Intervals_virtual", distance 2.

Class "array", by class "Intervals_virtual", distance 3.

Class "structure", by class "Intervals_virtual", distance 4.

Class "vector", by class "Intervals_virtual", distance 5, with explicit coerce.

## S3 methods

As of R 2.8.1, it still does not seem possible to write S4 methods for rbind or c. To concatenate sets of intervals into a single sets, the S3 methods c.Intervals and c.Intervals_full are provided. While rbind might seem more natural, its S3 dispatch is non-standard and it could not be used. Both methods are documented separately.

## S4 methods

**[** signature(x = "Intervals")

**[** signature(x = "Intervals_full")

**[<-** signature(x = "Intervals", i = "ANY", j = "missing", value = "Intervals_virtual")

**[<-** signature(x = "Intervals_full", i = "ANY", j = "missing", value = "Intervals_virtual")

**adjust_closure** signature(x = "Intervals")

**adjust_closure** signature(x = "Intervals_full")

**closed<-** signature(x = "Intervals")

**closed<-** signature(x = "Intervals_full")

**coerce** signature(from = "Intervals", to = "Intervals_full")

**coerce** signature(from = "Intervals_full", to = "Intervals")

**empty** signature(x = "Intervals")

**empty** signature(x = "Intervals_full")

**initialize** signature(.Object = "Intervals")

**initialize** signature(.Object = "Intervals_full")

**size** signature(x = "Intervals")

**size** signature(x = "Intervals_full")

## Warning

Validity checking takes place when, for example, using the type<- replacement accessor: if one attempts to set type to "Z" but the endpoint matrix contains non-integer values, an error is generated. Because accessors are not used for the endpoint matrix itself, though, it is possible to create invalid "Z" objects by setting endpoints to inappropriate values.

## Note

We do not currently permit an integer data type for the endpoints matrix, even when `type == "Z"`, because this creates complications when taking complements – which is most easily handled through the use of `-Inf` and `Inf`. This is particularly awkward for objects of class `"Intervals"`, since current endpoint closure settings may not permit inclusion of the minimal/maximal integer. This issue may be addressed, however, in future updates. (We do, however, check that endpoints are congruent to 0 mod 1 when `type == "Z"`.)

When creating object, non-matrix endpoint sources will be converted to a two-column matrix, for convenience. Recycling is supported for the `closed` slot when creating new objects.

## See Also

See `"Intervals_virtual"`.

## Examples

```
# The "Intervals" class

i <- Intervals(
               matrix(
                      c(1,2,
                        3,5,
                        4,6,
                        8,9
                        ),
                      byrow = TRUE,
                      ncol = 2
               ),
               closed = c( TRUE, TRUE ),
               type = "Z"
               )

# Row subsetting preserves class. Column subsetting causes coercion to
# "matrix" class.

i
i[1:2,]
i[,1:2]

# Full endpoint control

j <- as( i, "Intervals_full" )
closed(j)[ 3:4, 2 ] <- FALSE
closed(j)[ 4, 1 ] <- FALSE
j

# Rownames may be used

rownames(j) <- c( "apple", "banana", "cherry", "date" )
j
```

```
# Assignment preserves class, coercing if necessary

j[2:3] <- i[1:2,]
j
```

---

Intervals_virtual-class *Class "Intervals_virtual"*

---

## Description

A virtual class from which the "Intervals" and "Intervals_full" classes derive.

## Slots

.Data: Object of class "matrix". A two-column, numeric (see below) format is required. For a valid object, no value in the first column may exceed its partner in the second column. (Note that this *does* permit empty interval rows, when both endpoints are of equal value and not both closed.) Only integral (though not "integer" class) endpoints are permitted if type is "Z". See the note on this point in documentation for "Intervals".

type: Object of class "character". A one-element character vector with either "Z" or "R" is required.

## Extends

Class "matrix", from data part.

Class "array", by class "matrix", distance 2.

Class "structure", by class "matrix", distance 3.

Class "vector", by class "matrix", distance 4, with explicit coerce.

## Methods

**close_intervals** signature(x = "Intervals_virtual")

**closed** signature(x = "Intervals_virtual")

**clusters** signature(x = "Intervals_virtual")

**coerce** signature(from = "Intervals_virtual", to = "character")

**contract** signature(x = "Intervals_virtual")

**expand** signature(x = "Intervals_virtual")

**head** signature(x = "Intervals_virtual")

**initialize** signature(.Object = "Intervals_virtual")

**interval_complement** signature(x = "Intervals_virtual")

**interval_difference** signature(x = "Intervals_virtual", y = "Intervals_virtual")

**interval_intersection** `signature(x = "Intervals_virtual")`

**interval_union** `signature(x = "Intervals_virtual")`

**is.na** `signature(x = "Intervals_virtual")`

**open_intervals** `signature(x = "Intervals_virtual")`

**reduce** `signature(x = "Intervals_virtual")`

**show** `signature(object = "Intervals_virtual")`

**t** `signature(x = "Intervals_virtual")`

**tail** `signature(x = "Intervals_virtual")`

**type** `signature(x = "Intervals_virtual")`

**type<-** `signature(x = "Intervals_virtual")`

**which_nearest** `signature(from = "numeric", to = "Intervals_virtual")`

**which_nearest** `signature(from = "Intervals_virtual", to = "numeric")`

**which_nearest** `signature(from = "Intervals_virtual", to = "Intervals_virtual")`

## See Also

See the "Intervals" and "Intervals_full" classes.

---

# Intervals_virtual_or_numeric-class

*Class "Intervals_virtual_or_numeric"*

---

## Description

A class union combining "Intervals_virtual" and "numeric". Used by, e.g., distance_to_nearest and which_nearest.

## Methods

**distance_to_nearest** `signature(from = "Intervals_virtual_or_numeric", to = "Intervals_virtual_or_numeric")`

**interval_overlap** `signature(from = "Intervals_virtual_or_numeric", to = "Intervals_virtual_or_numeric")`

## interval_complement *Compute the complement of a set of intervals*

### Description

Compute the complement of a set of intervals.

### Usage

```
## S4 method for signature 'Intervals_virtual'
interval_complement(x, check_valid = TRUE)
```

### Arguments

| | |
|---|---|
| x | An "Intervals" or "Intervals_full" object. |
| check_valid | Should validObject be called before passing to compiled code? Also see interval_overlap. |

### Value

An object of the same class as x, compactly representing the complement of the intervals described in x.

### Note

For objects of class "Intervals", closure on -Inf or Inf endpoints is set to match that of all the intervals with finite endpoints. For objects of class "Intervals_full", non-finite endpoints are left open (although in general, this package does not make a distinction between closed and open infinite endpoints).

## interval_difference *Compute set difference*

### Description

Compute the set difference between two objects.

### Usage

```
## S4 method for signature 'Intervals_virtual,Intervals_virtual'
interval_difference(x, y, check_valid = TRUE)
```

### Arguments

| | |
|---|---|
| x | An "Intervals" or "Intervals_full" object. |
| y | An "Intervals" or "Intervals_full" object, with a type slot matching that of x. |
| check_valid | Should validObject be called on x and y before passing to compiled code? Also see interval_overlap. |

### Value

An object representing the subset of the integers or real line, as determined by type(x), found in x but not in y.

### See Also

These methods are just wrappers for interval_intersection and interval_complement.

---

## interval_included *Assess inclusion of one set of intervals with respect to another*

---

### Description

Determine which intervals in the one set are completely included in the intervals of a second set.

### Usage

```
## S4 method for signature 'Intervals,Intervals'
interval_included(from, to, check_valid = TRUE)
## S4 method for signature 'Intervals_full,Intervals_full'
interval_included(from, to, check_valid = TRUE)
```

### Arguments

| | |
|---|---|
| from | An "Intervals" or "Intervals_full" object, or a vector of class "numeric". |
| to | An "Intervals" or "Intervals_full" object, or a vector of class "numeric". |
| check_valid | Should validObject be called before passing to compiled code? This, among other things, verifies that endpoints are of data type "numeric" and the closed vector/matrix is appropriately sized and of the correct data type. (Compiled code does no further checking.) |

### Value

A list, with one element for each row/component of from. The elements are vectors of indices, indicating which to rows (or components, for the "numeric" method) are completely included within each interval in from. A list element of length 0 indicates no included elements. Note that empty to elements are not included in anything, and empty from elements do not include anything.

## See Also

See interval_overlap for partial overlaps – i.e., at at least a point.

## Examples

```
# Note that 'from' and 'to' contain valid but empty intervals.

to <- Intervals(
               matrix(
                      c(
                        2,  6,
                        2,  8,
                        2,  9,
                        4,  4,
                        6,  8
                        ),
                      ncol = 2, byrow = TRUE
                      ),
               closed = c( TRUE, FALSE ),
               type = "Z"
               )

from <- Intervals(
               matrix(
                      c(
                         2,  8,
                         8,  9,
                         6,  9,
                        11, 12,
                         3,  3
                        ),
                      ncol = 2, byrow = TRUE
                      ),
               closed = c( TRUE, FALSE ),
               type = "Z"
               )
rownames(from) <- letters[1:nrow(from)]

from
to
interval_included(from, to)

closed(to) <- TRUE
to
interval_included(from, to)

# Intervals_full

F <- FALSE
T <- TRUE

to <- Intervals_full(
```

```
                    rep( c(2,8), c(4,4) ),
                    closed = matrix( c(F,F,T,T,F,T,F,T), ncol = 2 ),
                    type = "R"
                    )

type( from ) <- "R"
from <- as( from, "Intervals_full" )

from
to
interval_included(from, to)

# Testing

B <- 1000

x1 <- rexp( B, 1/1000 )
s1 <- runif( B, max=5 )
x2 <- rexp( B, 1/1000 )
s2 <- runif( B, max=3 )

from <- Intervals_full( cbind( x1, x1 + s1 ) )
to <- Intervals_full( cbind( x2, x2 + s2 ) )

ii <- interval_included( from, to )
ii_match <- which( sapply( ii, length ) > 0 )

from[ ii_match[1:3], ]
lapply( ii[ ii_match[1:3] ], function(x) to[x,] )

included <- to[ unlist( ii ), ]
dim( included )

interval_intersection( included, interval_complement( from ) )
```

---

`interval_intersection` *Compute the intersection of one or more sets of intervals*

---

## Description

Given one or more sets of intervals, produce a new set compactly representing points contained in at least one interval of each input object.

## Usage

```
## S4 method for signature 'Intervals_virtual'
interval_intersection(x, ..., check_valid = TRUE)

## S4 method for signature 'missing'
interval_intersection(x, ..., check_valid = TRUE)
```

### Arguments

| | |
|---|---|
| x | An "Intervals" or "Intervals_full" object. |
| ... | Additional objects of the same classes permitted for x. |
| check_valid | Should validObject be called before passing to compiled code? Also see interval_overlap. |

### Value

A single object representing points contained in each of the objects supplied in the x and ... arguments.

### See Also

See interval_union and interval_complement, which are used to produce the results.

---

## interval_overlap *Assess overlap from one set of intervals to another*

---

### Description

Assess overlap from intervals in one set to intervals in another set, and return the relevant indices.

### Usage

```
## S4 method for signature
## 'Intervals_virtual_or_numeric,Intervals_virtual_or_numeric'
interval_overlap(from, to, check_valid = TRUE)
```

### Arguments

| | |
|---|---|
| from | An "Intervals" or "Intervals_full" object, or a vector of class "numeric". *Note!* Prior to v. 0.11.1, this argument was called target. |
| to | An "Intervals" or "Intervals_full" object, or a vector of class "numeric". *Note!* Prior to v. 0.11.1, this argument was called query. |
| check_valid | Should validObject be called before passing to compiled code? This, among other things, verifies that endpoints are of data type "numeric" and the closed vector/matrix is appropriately sized and of the correct data type. (Compiled code does no further checking.) |

### Details

Intervals which meet at endpoints overlap only if both endpoints are closed. Intervals in to with NA endpoints are ignored, with a warning; in from, such intervals produce no matches. Intervals in either to or from which are actually empty have their endpoints set to NA before proceeding, with warning, and so do not generate matches. If eith to or from is a vector of class "numeric", overlap will be assess for the corresponding set of points.

## Value

A list, with one element for each row/component of `from`. The elements are vectors of indices, indicating which `to` rows (or components, for the `"numeric"` method) overlap each interval in `from`. A list element of length 0 indicates no overlapping elements.

## Note

If you want real (`type == "R"`) intervals that overlap in a set of positive measure — not just at endpoints — set all endpoints to open (i.e., `close(from) <- FALSE; closed(to) <- FALSE`) first.

This function is now just a wrapper for `which_nearest`.

## See Also

See `which_nearest` for details on nearby as well as overlapping intervals in `to`.

## Examples

```
# Note that 'from' contains a valid but empty interval.

to <- Intervals(
               matrix(
                      c(
                        2,  8,
                        3,  4,
                        5, 10
                        ),
                      ncol = 2, byrow = TRUE
                      ),
               closed = c( TRUE, FALSE ),
               type = "Z"
               )

from <- Intervals(
               matrix(
                      c(
                         2,  8,
                         8,  9,
                         6,  9,
                        11, 12,
                         3,  3
                        ),
                      ncol = 2, byrow = TRUE
                      ),
               closed = c( TRUE, FALSE ),
               type = "Z"
               )
rownames(from) <- letters[1:nrow(from)]

empty(to)
empty(from)
```

```
interval_overlap(from, to)

# Non-empty real intevals of size 0 can overlap other intervals.

u <- to
type(u) <- "R"

v <- Intervals_full( rep(3,4) )
closed(v)[2,] <- FALSE
v
empty(v)
size(v)

interval_overlap(v, u)

# Working with points

interval_overlap( from, c( 2, 3, 6, NA ) )
```

---

`interval_union` *Compute the union of intervals in one or more interval matrices*

---

## Description

Compute the union of intervals in one or more interval matrices. The intervals contained in a single interval matrix object need not, in general, be disjoint; `interval_union`, however, always returns a matrix with sorted, disjoint intervals.

## Usage

```
## S4 method for signature 'Intervals_virtual'
interval_union(x, ..., check_valid = TRUE)

## S4 method for signature 'missing'
interval_union(x, ..., check_valid = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | An `"Intervals"` or `"Intervals_full"` object. |
| `...` | Optionally, additional objects which can be combined with `x`. See `c.Intervals` for details on mixing different types of objects. |
| `check_valid` | Should `validObject` be called before passing to compiled code? Also see `interval_overlap`. |

## Details

All supplied objects are combined using `c` and then then passed to `reduce`. The `missing` method is only to permit use of `do.call` with named list, since no named element will typically match `x`.

## Value

A single object of appropriate class, compactly representing the union of all intervals in `x`, and optionally, in `...` as well. For class `"Intervals"`, the result will have the same `closed` values as `x`.

## See Also

See `reduce`, which is used to produce the results.

---

`plot.Intervals` *Plotting methods for interval objects*

---

## Description

S3 methods for plotting `"Intervals"` and `"Intervals_full"` objects.

## Usage

```
## S3 method for class 'Intervals'
plot(x, y, ...)
## S3 method for class 'Intervals_full'
plot(
     x, y = NULL,
     axes = TRUE,
     xlab = "", ylab = "",
     xlim = NULL, ylim = NULL,
     col = "black", lwd = 1,
     cex = 1,
     use_points = TRUE,
     use_names = TRUE,
     names_cex = 1,
     ...
     )

## S4 method for signature 'Intervals,missing'
plot(x, y, ...)
## S4 method for signature 'Intervals_full,missing'
plot(x, y, ...)
## S4 method for signature 'Intervals,ANY'
plot(x, y, ...)
## S4 method for signature 'Intervals_full,ANY'
plot(x, y, ...)
```

## Arguments

| | |
|---|---|
| x | An "Intervals" or "Intervals_full" object. |
| y | Optional vector of heights at which to plot intervals. If omitted, y will be automatically computed to generate a compact plot but with no overlap. |
| axes | As for plot.default. |
| xlab | As for plot.default. |
| ylab | As for plot.default. |
| xlim | As for plot.default. |
| ylim | If not explicitly supplied, ylim is set to the maximum value required for intervals which are visible for the given xlim. |
| col | Color used for segments and endpoint points and interiors. Recycled if necessary. |
| lwd | Line width for segments. See par. |
| cex | Endpoint magnification. Only relevant if use_points = TRUE. See par. |
| use_points | Should points be plotted at interval endpoints? |
| use_names | Should rownames(x) by used for segment labels in the plot? |
| names_cex | Segment label magnification. Only relevant if use_names = TRUE. |
| ... | Other arguments for plot.default. |

## Details

Intervals with NA for either endpoint are not plotted. Vertical placement is on the integers, beginning with 0.

## Value

None.

## Examples

```
# Note plot symbol for empty interval in 'from'.

from <- Intervals(
                  matrix(
                         c(
                            2,  8,
                            8,  9,
                            6,  9,
                           11, 12,
                            3,  3
                           ),
                         ncol = 2, byrow = TRUE
                         ),
                  closed = c( FALSE, TRUE ),
                  type = "Z"
                  )
```

```
rownames(from) <- c("a","b","c","d","e")

to <- Intervals(
                matrix(
                       c(
                         2,  8,
                         3,  4,
                         5, 10
                         ),
                       ncol = 2, byrow = TRUE
                       ),
                closed = c( FALSE, TRUE  ),
                type = "Z"
                )

rownames(to) <- c("x","y","z")

empty(from)

plot(
     c(from,to),
     col = rep(1:2, c(nrow(from), nrow(to)))
     )

legend("topright", c("from","to"), col=1:2, lwd=1)

# More intervals. The maximal height shown is adapted to the plotting
# window.

B <- 10000
left <- runif( B, 0, 1e5 )
right <- left + rexp( B, rate = 1/10 )
x <- Intervals( cbind( left, right ) )

plot(x, use_points=FALSE)
plot(x, use_points=FALSE, xlim = c(0, 500))
```

---

reduce *Compactly re-represent the points in a set of intervals*

---

## Description

In general, "Intervals" and "Intervals_full" objects may be redundant, the intervals they contain may be in arbitrary order, and they may contain non-informative intervals for which one or both endpoints are NA. The reduce function re-represents the underlying subsets of the integers or the real line in the unique, minimal form, removing intervals with NA endpoints (with warning).

### Usage

```
## S4 method for signature 'Intervals_virtual'
reduce( x, check_valid = TRUE )
```

### Arguments

| | |
|---|---|
| x | An "Intervals" or "Intervals_full" object. |
| check_valid | Should validObject be called before passing to compiled code? Also see interval_overlap. |

### Value

A single object of appropriate class, compactly representing the union of all intervals in x. All intervals in reduce(x) have numeric (i.e., not NA) endpoints.

### See Also

See interval_union, which is really just concatenates its arguments and then calls reduce.

---

## sgd *Yeast gene model sample data*

---

### Description

This data set contains a data frame describing a subset of the chromosome feature data represented in Fall 2007 version of 'saccharomyces_cerevisiae.gff', available for download from the *Saccharomyces* Genome Database (http://www.yeastgenome.org).

### Usage

```
data(sgd)
```

### Format

A data frame with 14080 observations on the following 8 variables.

SGDID SGD feature ID.

type Only four feature types have been retatined: "CDS", "five_prime_UTR_intron", "intron", and "ORF". Note that "ORF" correspond to a whole gene while "CDS", to an exon. *S. cerevisae* does not, however, have many multi-exonic genes.

feature_name A character vector

parent_feature_name The feature_name of the a larger element to which the current feature belongs. All retained "CDS" entries, for example, belong to an "ORF" entry.

chr The chromosome on which the feature occurs.

start Feature start base.

stop Feature stop base.

strand Is the feature on the Watson or Crick strand?

## Examples

```
# An example to compute "promoters", defined to be the 500 bases
# upstream from an ORF annotation, provided these bases don't intersect
# another orf. See documentation for the sgd data set for more details
# on the annotation set.

use_chr <- "chr01"

data( sgd )
sgd <- subset( sgd, chr == use_chr )

orf <- Intervals(
                subset( sgd, type == "ORF", c( "start", "stop" ) ),
                type = "Z"
                )
rownames( orf ) <- subset( sgd, type == "ORF" )$feature_name

W <- subset( sgd, type == "ORF", "strand" ) == "W"

promoters_W <- Intervals(
                        cbind( orf[W,1] - 500, orf[W,1] - 1 ),
                        type = "Z"
                        )

promoters_W <- interval_intersection(
                                    promoters_W,
                                    interval_complement( orf )
                                    )

# Many Watson-strand genes have another ORF upstream at a distance of
# less than 500 bp

hist( size( promoters_W ) )

# All CDS entries are completely within their corresponding ORF entry.

cds_W <- Intervals(
                subset( sgd, type == "CDS" & strand == "W", c( "start", "stop" ) ),
                type = "Z"
                )
rownames( cds_W ) <- NULL

interval_intersection( cds_W, interval_complement( orf[W,] ) )
```

---

`size` *Compute interval sizes*

---

## Description

Compute the size, in either Z or R as appropriate, for each interval in an interval matrix.

## Usage

```
## S4 method for signature 'Intervals'
size(x, as = type(x))

## S4 method for signature 'Intervals_full'
size(x, as = type(x))
```

## Arguments

| | |
|---|---|
| x | An "Intervals" or "Intervals_full" object. |
| as | Should the intervals be thought of as in Z or R? This is usually determined automatically from the type slot, but because changing type may cause object copying, it is sometimes convenient to temporarily override this slot without actually resetting it. |

## Details

For type "Z" objects, counting measure; for type "R" objects, Lebesgue measure. For type "Z" objects, intervals of form (*a*,*a*] and [*a*,*a*) are both of length 0.

## Value

A numeric vector with length equal to nrow(x).

## See Also

See empty to identify empty intervals. Note that when type(x) == "R", a size of 0 does not imply that an interval is empty.

## Examples

```
z1 <- Intervals( cbind( 1, 1:3 ), type = "Z" )
z2 <- z1; closed(z2)[1] <- FALSE
z3 <- z1; closed(z3) <- FALSE

size(z1)
size(z2)
size(z3)

r1 <- z1; type(r1) <- "R"
r2 <- z2; type(r2) <- "R"
r3 <- z3; type(r3) <- "R"

size(r1)
size(r2)
size(r3)
```

```
s1 <- Intervals_full( matrix( 1, 3, 2 ), type = "Z" )
closed(s1)[2,2] <- FALSE
closed(s1)[3,] <- FALSE

size(s1)
```

---

`split` *Split an intervals object according to a factor*

---

## Description

S3 and S4 methods for splitting `"Intervals"` or `"Intervals_full"` objects.

## Usage

```
## S3 method for class 'Intervals_virtual'
split(x, f, drop = FALSE, ...)

## S4 method for signature 'Intervals_virtual'
split(x, f, drop = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | `"Intervals"` or `"Intervals_full"` objects. |
| `f` | Passed to `split.data.frame`. |
| `drop` | Passed to `split.data.frame`. |
| `...` | Passed to `split.data.frame`. |

## Value

A list of objects of the same class as `x`, split by the levels of `f`. Until R 2.15, special handling was not required. Subsequent changes to the **base** package `split` function required an explicit method here, but code already provided by `split.data.frame` was sufficient.

---

`which_nearest` *Identify nearest member(s) in a set of intervals*

---

## Description

For each point or interval in the `from` argument, identify the nearest member or members (in case of ties) of the interval set in the `to` argument.

## Usage

```
## S4 method for signature 'numeric,Intervals_virtual'
which_nearest(from, to, check_valid = TRUE)

## S4 method for signature 'Intervals_virtual,numeric'
which_nearest(from, to, check_valid = TRUE)

## S4 method for signature 'Intervals_virtual,Intervals_virtual'
which_nearest(from, to, check_valid = TRUE)

## S4 method for signature 'numeric,numeric'
which_nearest(from, to, check_valid = TRUE)
```

## Arguments

| | |
|---|---|
| from | An object of appropriate type. |
| to | An object of appropriate type. |
| check_valid | Should validObject be called before passing to compiled code? Also see interval_overlap. |

## Value

A data frame with three columns: distance_to_nearest, which_nearest, and which_overlap. The last two are actually lists, since there may be zero, one, or more nearest/overlapping intervals in the to object for any given interval in from.

Empty intervals in to, or intervals with NA endpoints, produce a NA distance result, and no nearest or overlapping hits.

## Note

(v. 0.11.0) The code used for the distance_to_nearest column here is completely distinct from that used for the original distance_to_nearest function. For the moment, they will co-exist for testing purposes, but this function's code will eventually replace the older code.

Note that a naive way of implementing which_nearest would be to use the simpler, old implementation of distance_to_nearest, use expand to grow all intervals by the correspnoding amount, and then use interval_overlap to identify target. This approach, however, will miss a small fraction of targets due to floating point issues.

## Examples

```
# Point to interval. Empty rows, or those with NA endpoints, do not
# generate hits. Note that distance_to_nearest can be 0 but without
# overlap, depending on endpoint closure.

to <- Intervals_full( c(-1,0,NA,5,-1,3,10,Inf) )
closed(to)[1,] <- FALSE
closed(to)[2,2] <- FALSE
from <- c( NA, -3:5 )
```

```
to
cbind( from, which_nearest( from, to ) )

# Completely empty to object

which_nearest( from, to[1,] )

# Interval to interval

from <- Intervals( c(-Inf,-Inf,3.5,-1,1,4) )
from
which_nearest( from, to )

# Checking behavior with ties

from <- Intervals_full( c(2,2,4,4,3,3,5,5) )
closed( from )[2:3,] <- FALSE
to <- Intervals_full( c(0,0,6,6,1,1,7,8) )
closed( to )[2:3,] <- FALSE

from
to
which_nearest( from, to )

from <- Intervals_full( c(1,3,6,2,4,7) )
to <- Intervals_full( c(4,4,5,5) )
closed( to )[1,] <- FALSE

from
to
which_nearest( from, to )
```

# Index