Package 'oce'

July 2, 2014

Version 0.9-14

Date 2014-05-19

Title Analysis of Oceanographic data

Author Dan Kelley <Dan.Kelley@Dal.Ca>

Maintainer Dan Kelley <Dan.Kelley@Dal.Ca>

Depends R (>= 2.15), utils, methods, mapproj

Suggests ocedata, foreign, ncdf4, tiff

BugReports https://github.com/dankelley/oce/issues?sort=created&direction=desc&state=open

Description Supports the analysis of Oceanographic data,including ADP measurements, CTD measurements, sectional data, sea-level time series, coastline files, etc.
Provides functions for calculating seawater properties
such as potential temperature and density, as well as
derived properties such as buoyancy frequency and dynamic height.

License GPL (>= 2)

URL http://dankelley.github.com/oce/

LazyLoad yes

LazyData no

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-05-19 14:26:07

R topics documented:

abbreviateTimeLabels . . . 8
accessors . . . 9
addColumn . . . 10
adp . . . 11
adp-class . . . 12
adv . . . 14
adv-class . . . 15
airRho . . . 16
angleRemap . . . 17
applyMagneticDeclination . . . 18
approx3d . . . 19
as.coastline . . . 20
as.ctd . . . 21
as.drifter . . . 24
as.echosounder . . . 25
as.gps . . . 26
as.lisst . . . 27
as.lobo . . . 28
as.met . . . 29
as.sealevel . . . 30
as.section . . . 31
as.tdr . . . 32
as.topo . . . 34
as.windrose . . . 34
bcdToInteger . . . 36
beamName . . . 36
beamToXyz . . . 37
beamToXyzAdp . . . 38
beamToXyzAdv . . . 39
beamUnspreadAdp . . . 40
binApply . . . 41
binAverage . . . 42
binmapAdp . . . 44
binMean . . . 45
byteToBinary . . . 46
cm . . . 47
cm-class . . . 48
coastline-class . . . 48
coastlineBest . . . 49
coastlineWorld . . . 50
colormap . . . 51
coriolis . . . 54
ctd . . . 55
ctd-class . . . 56
ctdAddColumn . . . 57
ctdDecimate . . . 58

ctdFindProfiles . . . 60
ctdRaw . . . 61
ctdTrim . . . 62
ctdUpdateHeader . . . 64
ctimeToSeconds . . . 65
decimate . . . 66
decodeHeader . . . 67
decodeTime . . . 68
despike . . . 69
detrend . . . 71
drawDirectionField . . . 72
drawIsopycnals . . . 73
drawPalette . . . 74
drifter . . . 76
drifter-class . . . 77
echosounder . . . 78
echosounder-class . . . 79
eclipticalToEquatorial . . . 80
enuToOtherAdp . . . 81
enuToOtherAdv . . . 82
equatorialToLocalHorizontal . . . 83
errorbars . . . 84
extract . . . 85
fillGap . . . 86
findBottom . . . 87
findInOrdered . . . 88
formatCI . . . 88
formatPosition . . . 90
fullFilename . . . 91
geodDist . . . 91
geodGc . . . 93
geodXy . . . 94
GMTOffsetFromTz . . . 95
gps-class . . . 96
grad . . . 96
gravity . . . 97
head . . . 98
header . . . 99
imagep . . . 100
integerToAscii . . . 103
integrateTrapezoid . . . 104
interpBarnes . . . 105
is.beam . . . 107
julianCenturyAnomaly . . . 108
julianDay . . . 109
landsat-class . . . 110
landsatTrim . . . 111
latFormat . . . 112

latlonFormat . . . 112
lisst . . . 113
lisst-class . . . 114
lobo . . . 115
lobo-class . . . 115
loggerToc . . . 116
lonFormat . . . 117
magneticField . . . 118
makeFilter . . . 119
makeSection . . . 121
map2lonlat . . . 122
mapContour . . . 123
mapImage . . . 124
mapLines . . . 126
mapLocator . . . 127
mapLongitudeLatitudeXY . . . 128
mapMeridians . . . 129
mapPlot . . . 130
mapPoints . . . 132
mapPolygon . . . 133
mapScalebar . . . 135
mapText . . . 136
mapZones . . . 137
matchBytes . . . 138
matrixSmooth . . . 139
met . . . 140
met-class . . . 140
moonAngle . . . 141
nao . . . 143
numberAsHMS . . . 144
numberAsPOSIXct . . . 145
oce-class . . . 146
oce.as.POSIXlt . . . 147
oce.as.raw . . . 148
oce.axis.POSIXct . . . 148
oce.plot.ts . . . 150
oce.write.table . . . 152
oceApprox . . . 153
oceColors . . . 155
oceContour . . . 156
oceConvolve . . . 158
oceDebug . . . 159
oceEdit . . . 160
oceFilter . . . 161
oceMagic . . . 162
ocePmatch . . . 163
oceSmooth . . . 164
oceSpectrum . . . 165

parseLatLon . . . 166
plot.adp . . . 167
plot.adv . . . 172
plot.cm . . . 175
plot.coastline . . . 177
plot.ctd . . . 180
plot.drifter . . . 183
plot.echosounder . . . 185
plot.gps . . . 187
plot.landsat . . . 190
plot.lisst . . . 191
plot.lobo . . . 192
plot.met . . . 194
plot.sealevel . . . 195
plot.section . . . 197
plot.tdr . . . 201
plot.tidem . . . 203
plot.topo . . . 204
plot.windrose . . . 206
plotInset . . . 207
plotPolar . . . 209
plotProfile . . . 209
plotScan . . . 213
plotSticks . . . 214
plotTaylor . . . 215
plotTS . . . 216
predict.tidem . . . 219
prettyPosition . . . 220
processingLog . . . 221
pwelch . . . 222
rangeExtended . . . 224
rangeLimit . . . 224
read.adp . . . 225
read.adv . . . 229
read.cm . . . 234
read.coastline . . . 236
read.ctd . . . 238
read.drifter . . . 242
read.echosounder . . . 244
read.gps . . . 245
read.landsat . . . 246
read.lisst . . . 247
read.lobo . . . 249
read.met . . . 251
read.observatory . . . 252
read.oce . . . 253
read.sealevel . . . 254
read.section . . . 256

read.tdr . . . 258
read.topo . . . 259
rescale . . . 260
resizableLabel . . . 261
retime . . . 262
runlm . . . 263
sealevel . . . 265
sealevel-class . . . 266
sealevelTuktoyaktuk . . . 267
secondsToCtime . . . 268
section . . . 269
section-class . . . 270
sectionGrid . . . 271
sectionSmooth . . . 272
sectionSort . . . 273
showMetadataItem . . . 274
siderealTime . . . 275
soi . . . 276
standardDepths . . . 277
subset.adp . . . 278
subset.adv . . . 279
subset.cm . . . 280
subset.coastline . . . 281
subset.ctd . . . 282
subset.echosounder . . . 283
subset.lisst . . . 284
subset.oce . . . 285
subset.sealevel . . . 285
subset.section . . . 286
subset.tdr . . . 287
subset.topo . . . 288
subtractBottomVelocity . . . 289
summary.adp . . . 289
summary.adv . . . 290
summary.cm . . . 291
summary.coastline . . . 292
summary.ctd . . . 293
summary.drifter . . . 294
summary.echosounder . . . 295
summary.gps . . . 296
summary.landsat . . . 296
summary.lisst . . . 297
summary.lobo . . . 298
summary.met . . . 299
summary.sealevel . . . 300
summary.section . . . 301
summary.tdr . . . 302
summary.tidem . . . 303

summary.topo . . . 304
summary.windrose . . . 305
sunAngle . . . 306
swAbsoluteSalinity . . . 307
swAlpha . . . 309
swAlphaOverBeta . . . 310
swBeta . . . 311
swConductivity . . . 312
swConservativeTemperature . . . 313
swDepth . . . 314
swDynamicHeight . . . 315
swLapseRate . . . 316
swN2 . . . 317
swPressure . . . 319
swRho . . . 320
swRrho . . . 321
swSCTp . . . 322
swSigma . . . 323
swSigmaT . . . 324
swSigmaTheta . . . 325
swSoundAbsorption . . . 326
swSoundSpeed . . . 327
swSpecificHeat . . . 328
swSpice . . . 329
swSTrho . . . 330
swTFreeze . . . 331
swTheta . . . 332
swTSrho . . . 333
swViscosity . . . 334
tdr . . . 335
tdr-class . . . 336
tdrPatm . . . 337
tdrTrim . . . 338
teos . . . 339
teosSetLibrary . . . 341
threenum . . . 342
tidedata . . . 343
tidem . . . 344
tidem-class . . . 347
tidemAstron . . . 347
tidemVuf . . . 348
time.oce . . . 349
toEnuAdp . . . 350
toEnuAdv . . . 351
topo-class . . . 352
topoInterpolate . . . 352
topoWorld . . . 353
unabbreviateYear . . . 354

undriftTime 354
ungrid 355
unwrapAngle 356
useHeading 357
vectorShow 358
velocityStatistics 358
webtide 359
wind 361
window.oce 362
windrose-class 363
write.ctd 364
xyzToEnuAdp 365
xyzToEnuAdv 366

Index **369**

`abbreviateTimeLabels` *Abbreviate a list of times by removing commonalities (e.g. year)*

Description

Abbreviate a list of times by removing commonalities (e.g. year)

Usage

```
abbreviateTimeLabels(t, ...)
```

Arguments

`t`	vector of times.
`...`	optional arguments passed to the `format`, e.g. `format`.

Details

Abbreviates time labels for purposes such as saving repetition on axes.

Value

None.

Author(s)

Dan Kelley, with help from Clark Richards

See Also

This is used by various functions that draw time labels on axes, e.g. `plot.adp`.

accessors *Access or modify part of an Oce object*

Description

Access or modify part of an Oce object

Usage

```
elevation(x, time)
distance(x, time)
heading(x, time)
heading(x) <- value
latitude(x, time, byDepth=TRUE)
latitude(x) <- value
longitude(x, time, byDepth=TRUE)
longitude(x) <- value
pitch(x, time)
pitch(x) <- value
pressure(x, time)
pressure(x) <- value
roll(x, time)
roll(x) <- value
salinity(x, time)
salinity(x) <- value
sigmaTheta(x, time)
sigmaTheta(x) <- value
temperature(x, time)
temperature(x) <- value
oxygen(x, time)
oxygen(x) <- value
nitrate(x, time)
nitrate(x) <- value
nitrite(x, time)
nitrite(x) <- value
phosphate(x, time)
phosphate(x) <- value
silicate(x, time)
silicate(x) <- value
spice(x, time)
tritium(x, time)
time(x)
velocity(x)
```

Arguments

x	an oce object.

time	optional vector of POSIX times, or object the data slot of which contains times in a field named time or timeSlow. (If this argument is not provided, missing, values at all the times in x are returned.)
byDepth	flag usd only for "section" objects, which indicates whether to repeat the longitude or latitude values so that there is a value for each depth in each profile.
value	value to assign to the relevant item in the x object.

Details

These accessor functions provide a convenient way to discover, or set, data within oce objects. This prevents the user from having to know the details of storage, e.g. that Nortek Vector velocimeters record angles on slow timescales compared with velocities (stored within the data slot in entries named timeSlow, headingSlow, etc), whereas Sontek ADV velocimeters record them on the same timescale as velocity (stored in time, heading, etc.)

Value

Value of indicated portion of x.

Author(s)

Dan Kelley

See Also

Similar accessor functions are time, and velocity.

Examples

```
library(oce)
data(adp)
print(heading(adp))
heading(adp) <- 5 + heading(adp) # add 5 degrees to the heading
print(heading(adp))
```

---

addColumn *Add a column to an oce object*

Description

Add a column to an oce object's data.

Usage

```
addColumn(x, data, name)
```

Arguments

x	A ctd object, e.g. as read by read.ctd.
data	the data. The length of this item must match that of the existing data entries in the data slot).
name	the name of the column.

Details

If there is already a column with the given name, its contents are replaced by the new value.

Value

An object of class oce, with a new column.

Author(s)

Dan Kelley

See Also

ctdAddColumn does a similar thing for ctd objects, and is in fact called, if x is of class ctd.

Examples

```
library(oce)
data(ctd)
st <- swSigmaTheta(salinity(ctd), temperature(ctd), pressure(ctd))
new <- addColumn(ctd, st, "sigmaTheta")
```

---

adp *ADP (acoustic-doppler profiler) dataset*

Description

This is degraded subsample of measurements that were made with an upward-pointing ADP manufactured by Teledyne-RDI, as part of the St Lawrence Internal Wave Experiment (SLEIWEX).

Usage

```
data(adp)
```

Author(s)

Dan Kelley

Source

This file came from the SLEIWEX-2008 experiment.

See Also

The Oce documentation for adp-class explains the structure of ADP objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(adp)

# Velocity components.  (Note: we should probably trim some bins at top.)
plot(adp)

# Note that tides have moved the mooring.
plot(adp, which=15:18)

## End(Not run)
```

---

adp-class *Class to store acoustic Doppler profiler data*

Description

Class to store acoustic Doppler profiler data, holding the three slots used in all objects in Oce.

The processingLog is in standard form and needs little comment. The rest of this section discusses the metadata and data slots, and the notation used assumes an object of class adp that is named "adp."

The metadata slot contains various items relating to the dataset, including source file name, sampling rate, velocity resolution, velocity maximum value, and so on. Some of these are particular to particular instrument types, and prudent researchers will take a moment to examine the whole contents of the metdata, either in summary form (with str(adp[["metadata"]])) or in detail (with adp[["metadata"]]). Perhaps the most useful general properties are adp[["bin1Distance"]] (the distance, in metres, from the sensor to the bottom of the first bin), adp[["cellSize"]] (the cell height, in metres, in the vertical direction, *not* along the beam), and adp[["beamAngle"]] (the angle, in degrees, between beams and an imaginary centre line that bisects all beam pairs).

The diagram provided below indicates the coordinate-axis and beam-numbering conventions for three- and four-beam ADP devices, viewed as though the reader were looking towards the beams being emitted from the tranducers.

The bin geometry of a four-beam profiler is illustrated below, for adp[["beamAngle"]] equal to 20 degrees, adp[["bin1Distance"]] equal to 2m, and adp[["cellSize"]] equal to 1m. In the diagram, the viewer is in the plane containing two beams that are not shown, so the two visible beams are separated by 40 degrees. Circles indicate the centres of the range-gated bins within the beams. The lines enclosing those circles indicate the coverage of beams that spread plus and minus 2.5 degrees from their centreline.

Note that adp[["oceCoordinate"]] stores the present coordinate system of the object, and it has possible values "beam", "xyz" or "enu". (This should not be confused with adp[["originalCoordinate"]], which stores the coordinate system used in the original data file.)

In contrast to the metadata slot, which holds many items that are instrument-specific, the data slot enforces a single pattern on all instrument types. To begin with, adp[["v"]] is a three-dimensional numeric matrix of velocities in m/s. In this matrix, the first index indicates time, the second bin number, and the third beam number. The meanings of the beams depends on whether the object is in beam coordinates, frame coordinates, or earth coordinates.

Corresponding to the velocity matrix are two matrices of type raw, and identical dimension, accessed by adp[["a"]] and adp[["q"]], holding measures of signal strength and data quality quality, respectively. (The exact meanings of these depend on the particular type of instrument, and it is assumed that users will be familiar enough with instruments to know both the meanings and their practical consequences in terms of data-quality assessment, etc.)

In addition to the matrices, there are time-based vectors. The vector adp[["time"]] (of length equal to the first index of adp[["v"]], etc.) holds POSIXt times of observation. Depending on type of instrument and its configuration, there may also be corresponding vectors for sound speed (adp[["soundSpeed"]]), pressure (adp[["pressure"]]), temperature (adp[["temperature"]]), heading (adp[["heading"]]) pitch (adp[["pitch"]]), and roll (adp[["roll"]]), depending on the setup of the instrument.

The precise meanings of the data items depend on the instrument type. All instruments have v (for velocity), q (for a measure of data quality) and a (for a measure of backscatter amplitude). Devices from Teledyne-RDI profilers have an additional item g (for percent-good).

For RDI profilers, there are four three-dimensional arrays holding beamwise data. In these, the first index indicates time, the second bin number, and the third beam number (or coordinate number, for data in xyz, enu or other coordiante systems). In the list below, the quoted phrases are quantities as defined in Figure 9 of reference 1.

- v is "velocity" in m/s, inferred from two-byte signed integer values (multiplied by the scale factor that is stored in velocityScale in the metadata).
- q is "correlation magnitude" a one-byte quantity stored as type raw in the object. The values may range from 0 to 255.
- a is "echo intensity" a one-byte quantity stored as type raw in the object. The values may range from 0 to 255.
- g is "percent good" a one-byte quantity stored as raw in the object. The values may range from 0 to 100.

Finally, there is a vector adp[["distance"]] that indicates the bin distances from the sensor, meaasured in metres along an imaginary centre line bisecting beam pairs. The length of this vector equals dim(adp[["v"]])[2].

Methods

Extracting values: Matrix data may be accessed as illustrated above, e.g. or an adp object named adv, the data are provided by adp[["v"]], adp[["a"]], and adp[["q"]]. As a convenience, the last two of these can be accessed as numeric (as opposed to raw) values by e.g. adp[["a", "numeric"]]. The vectors are accessed in a similar way, e.g. adp[["heading"]], etc. Quantities in the metadata slot are also available by name, e.g. adp[["velocityResolution"]], etc.

Assigning values: This follows the standard form, e.g. to increase all velocity data by 1 cm/s, use `adp[["v"]] <- 0.01 + adp[["v"]]`.

Overview of contents: The `show` method (e.g. `show(d)`) displays information about an ADP object named `d`.

Dealing with suspect data

There are many possibilities for confusion with `adp` devices, owing partly to the flexibility that manufacturers provide in the setup. Prudent users will undertake many tests before trusting the details of the data. Are mean currents in the expected direction, and of the expected magnitude, based on other observations or physical constraints? Is the phasing of currents as expected? If the signals are suspect, could an incorrect scale account for it? Could the transformation matrix be incorrect? Might the data have exceeded the maximum value, and then "wrapped around" to smaller values? Time spent on building confidence in data quality is seldom time wasted.

Author(s)

Dan Kelley

References

1. Teledyne-RDI, 2007. *WorkHorse commands and output data format.* P/N 957-6156-00 (November 2007).

See Also

A file containing ADP data is usually recognized by Oce, and so `read.oce` will usually read the data. If not, one may use the general ADP function `read.adp` or specialized variants `read.adp.rdi`, `read.adp.nortek` or `read.adp.sontek` or `read.adp.sontek.serial`.

ADP data may be plotted with `plot.adp` function, which is a generic function so it may be called simply as `plot`.

Statistical summaries of ADP data are provided by the generic function `summary`, while briefer overviews are provided with `show`.

Conversion from beam to xyz coordinates may be done with `beamToXyzAdp`, and from xyz to enu (east north up) may be done with `xyzToEnuAdp`. `toEnuAdp` may be used to transfer either beam or xyz to enu. Enu may be converted to other coordinates (e.g. aligned with a coastline) with `enuToOtherAdp`.

---

adv *ADV (acoustic-doppler velocimeter) dataset*

Description

This is a sampling of measurements made with a Nortek Vector acoustic Doppler velocimeter deployed as part of the St Lawrence Internal Wave Experiment (SLEIWEX). Various identifying features have been redacted, and velocities have modified by adding random numbers chosen from a normal distribution with standard deviation of 2 percent of signal.

Usage

```
data(adv)
```

Author(s)

Dan Kelley

Source

This file came from the SLEIWEX-2008 experiment.

See Also

The documentation for `adv-class` in the Oce package explains the structure of ADV objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(adv)

# Velocity time-series
plot(adv)

# Spectrum of upward component of velocity, with ``turbulent'' reference line
s <- spectrum(adv[["v"]][,3],plot=FALSE)
plot(log10(s$freq), log10(s$spec), type='l')
for (a in seq(-20, 20, by=1))
    abline(a=a, b=-5/3, col='gray', lty='dotted')

## End(Not run)
```

---

adv-class *Class to store acoustic Doppler velocimeter data*

Description

Class to store acoustic Doppler velocimeter data, holding the three standard slots used in all objects in Oce. The `metadata` slot contains various items relating to the dataset, including source file name, sampling rate, velocity resolution and scale, etc. The `processingLog` is in standard form and needs little comment. The `data` slot holds a numeric matrix `v` of velocities in m/s, with the first index indicating time and the second indicating beam number. The meanings of the beams depends on whether the object is in beam coordinates, frame coordinates, or earth coordinates. The `data` slot also contains identically-dimensioned raw matrices `a` and `q`, holding measures of signal strength and data quality quality, respectively. It also contains a series of vectors, e.g. `time`, `temperature` and `pressure`, etc., depending on what sensors are included in the package. For all of these quantities, the details can be different for different instrument types, and it is assumed that the user will be familiar with the details.

Methods

Extracting values: Data may be accessed as e.g. for an object named d, the velocity matrix is retrieved by d[["v"]], the amplitude matrix by d[["a"]], the data-quality matrix by d[["q"]], etc. (The last two can be retrieved in numerical form, as opposed to raw form, by e.g. d[["a", "numeric"]].) Similarly, the vector quantities can be retrieved by name, e.g. d[["heading"]] (or "headingSlow", if appropriate), etc.

Assigning values: This follows the standard form, e.g. to increase all velocity data by 1 cm/s, use d[["v"]] <- 0.01 + d[["v"]].

Overview of contents: The show method (e.g. show(d)) displays information about an ADV object named d.

Author(s)

Dan Kelley

See Also

A file containing ADV data is usually recognized by Oce, and so read.oce will usually read the data. If not, one may use the general ADV function read.adv or specialized variants read.adv.nortek, read.adv.sontek.adr or read.adv.sontek.text.

ADV data may be plotted with plot.adv function, which is a generic function so it may be called simply as plot.

Statistical summaries of ADV data are provided by the generic function summary, while briefer overviews are provided with show.

Conversion from beam to xyz coordinates may be done with beamToXyzAdv, and from xyz to enu (east north up) may be done with xyzToEnuAdv. toEnuAdv may be used to transfer either beam or xyz to enu. Enu may be converted to other coordinates (e.g. aligned with a coastline) with enuToOtherAdv.

---

airRho *Air density*

Description

Compute, ρ, the *in-situ* density of air.

Usage

```
airRho(temperature, pressure, humidity)
```

Arguments

temperature	*in-situ* temperature [°C]
pressure	pressure in Pa (NOT kPa) – ignored at present
humidity	ignored at present

Details

This will eventually be a proper equation of state, but for now it's just returns something from wikipedia (i.e. not trustworthy), and not using humidity.

Value

In-situ air density [kg/m^3].

Author(s)

Dan Kelley

References

National Oceanographic and Atmospheric Agency, 1976. U.S. Standard Atmosphere, 1976. NOAA-S/T 76-1562. (Available as of 2010-09-30 at http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19770009539_1977009539.pdf).

Examples

```
degC <- seq(0,30,length.out=100)
p <- seq(98,102,length.out=100) * 1e3
contour(x=degC, y=p, z=outer(degC,p,airRho), labcex=1)
```

---

angleRemap *Convert angles from 0:360 to -180:180*

Description

Convert an angle from the 0:360 range to -180:180

Usage

```
angleRemap(theta)
```

Arguments

theta	an angle (in degrees) that is in the range from 0 to 360 degrees

Details

This is mostly used for instrument heading angles, in cases where the instrument is aligned nearly northward, so that small variations in heading (e.g. due to mooring motion) can yield values that swing from small angles to large angles, because of the modulo-360 cut point.

The method is to use the cosine and sine of the angle in order to find "x" and "y" values on a unit circle, and then to use atan2 to infer the angles.

Value

A vector of angles, in the range -180 to 180.

Author(s)

Dan Kelley

Examples

```
library(oce)
## fake some heading data that lie near due-north (0 degrees)
n <- 20
heading <- 360 + rnorm(n, sd=10)
heading <- ifelse(heading > 360, heading - 360, heading)
x <- 1:n
plot(x, heading, ylim=c(-10, 360), type='l', col='lightgray', lwd=10)
lines(x, angleRemap(heading))
```

---

applyMagneticDeclination

Earth magnetic declination

Description

Earth applyMagneticDeclination

Usage

```
applyMagneticDeclination(x, declination=0, debug=getOption("oceDebug"))
```

Arguments

x	an oce object.
declination	magnetic declination (to be added to the heading)
debug	a debugging flag, set to a positive value to get debugging.

Details

Instruments that use magnetic compasses to determine current direction need to have corrections applied for magnetic declination, to get currents with the y component oriented to geographic, not magnetic, north. Sometimes, and for some instruments, the declination is specified when the instrument is set up, so that the velocities as recorded are already. Other times, the data need to be adjusted. This function is for the latter case.

Value

Object, with velocity components ajusted to be aligned with geographic north and east.

Author(s)

Dan Kelley

References

http://www.ngdc.noaa.gov/IAGA/vmod/igrf.html

See Also

Use `magneticField` to determine the declination, inclination and intensity at a given spot on the world, at a given time.

Examples

```
library(oce)
```

`approx3d` *Trilinear interpolation in a 3D array*

Description

Interpolate within a 3D array, using the trilinear approximation.

Usage

```
approx3d(x, y, z, f, xout, yout, zout)
```

Arguments

`x`	vector of x values for grid (must be equi-spaced)
`y`	vector of y values for grid (must be equi-spaced)
`z`	vector of z values for grid (must be equi-spaced)
`f`	matrix of rank 3, with the gridd values mapping to the `x` values (first index of `f`), etc.
`xout`	vector of x values for output.
`yout`	vector of y values for output (length must match that of `xout`).
`zout`	vector of z values for output (length must match that of `xout`).

Details

Trilinear interpolation is used to interpolate within the `f` array, for those (`xout`, `yout` and `zout`) triplets that are inside the region specified by `x`, `y` and `z`. Triplets that lie outside the range of `x`, `y` or `z` result in `NA` values.

Value

A vector of interpolated values (or NA values), with length matching that of xout.

Author(s)

Dan Kelley and Clark Richards

Examples

```
## set up a grid
library(oce)
n <- 5
x <- seq(0, 1, length.out=n)
y <- seq(0, 1, length.out=n)
z <- seq(0, 1, length.out=n)
f <- array(1:n^3, dim=c(length(x), length(y), length(z)))
## interpolate along a diagonal line
m <- 100
xout <- seq(0, 1, length.out=m)
yout <- seq(0, 1, length.out=m)
zout <- seq(0, 1, length.out=m)
approx <- approx3d(x, y, z, f, xout, yout, zout)
## graph the results
plot(xout, approx, type='l')
points(xout[1], f[1,1,1])
points(xout[m], f[n,n,n])
```

---

as.coastline *Coerce data into coastline dataset*

Description

Coerces a sequence of longitudes and latitudes into a coastline dataset.

Usage

```
as.coastline(longitude, latitude, fillable)
```

Arguments

longitude	the longitude in decimal degrees, positive east of Greenwich, or a data frame with columns named latitude and longitude, in which case these values are extracted from the data frame and the second argument is ignored.
latitude	the latitude in decimal degrees, positive north of the Equator.
fillable	boolean indicating whether the coastline can be drawn as a filled polygon.

Details

This may be used when read.coastline cannot read a file, or when the data have been manipulated.

Value

An object of class "coastline" (for details, see read.coastline).

Author(s)

Dan Kelley

References

The NOAA site http://www.ngdc.noaa.gov/mgg/shorelines/shorelines.html is a good source for coastline data files.

See Also

The documentation for coastline-class explains the structure of coastline objects, and also outlines the other functions dealing with them.

---

as.ctd *Coerce data into ctd dataset*

Description

Coerces a dataset into a ctd dataset.

Usage

```
as.ctd(salinity, temperature, pressure,
       SA, CT,
       oxygen, nitrate, nitrite, phosphate, silicate,
       scan, other,
       missingValue,
       quality,
       filename="", type="", model="", serialNumber="",
       ship="", scientist="", institute="", address="", cruise="", station="",
       date="", startTime="", recovery="",
       longitude=NA, latitude=NA,
       waterDepth=NA, sampleInterval=NA, src="")
```

Arguments

salinity	Salinity through the water column, or a data frame containing columns named salinity, temperature, and pressure, in which case these values are extracted from the data frame, and the next two arguments are ignored. Otherwise, if it is numeric, it is first converted to a vector before proceeding.
temperature	Temperature through the water column. (This is converted to a vector, if it is not one already.
pressure	pressure through the water column. (If just a single value is given, then it is repeated to match the length of the temperature and salinity.
SA	absolute salinity (as in TEOS-10). If given, the supplied absolute salinity is converted internally to UNESCO-defined practical salinity.
CT	conservative temperature (as in TEOS-10). If given, the supplied conservative temperature is converted internally to UNESCO-defined in-situ temperature.
oxygen	optional oxygen concentration
nitrate	optional nitrate concentration [micromole/kg]
nitrite	optional nitrite concentration [micromole/kg]
phosphate	optional phosphate concentration [micromole/kg]
silicate	optional silicate concentration [micromole/kg]
scan	optional scan number. If not provided, this will be set to `1:length(salinity)`.
other	optional list of other data columns that are not in the standard list
missingValue	optional missing value, indicating data that should be taken as NA.
quality	quality flag, e.g. from the salinity quality flag in WOCE data. (In WOCE, `quality=2` indicates good data, `quality=3` means questionable data, and `quality=4` means bad data.
filename	filename to be stored in the object
type	type of CTD, e.g. "SBE"
model	model of instrument
serialNumber	serial number of instrument
ship	optional string containing the ship from which the observations were made.
scientist	optional string containing the chief scientist on the cruise.
institute	optional string containing the institute behind the work.
address	optional string containing the address of the institute.
cruise	optional string containing a cruise identifier.
station	optional string containing a station identifier.
date	optional string containing the date at which the profile was started.
startTime	optional string containing the start time.
recovery	optional string indicating the recovery time.
longitude	optional numerical value containing longitude in decimal degrees, positive in the eastern hemisphere.

latitude	optional numerical value containing the latitude in decimal degrees, positive in the northern hemisphere.
waterDepth	optional numerical value indicating the water depth in metres.
sampleInterval	optional numerical value indicating the time between samples in the profile.
src	optional string indicating data source

Details

This function assembles vectors of salinity, temperature, and pressure, to create a ctd object, e.g. so that plot.ctd can be used to make a standard four-panel plot, or so that a section can be constructed with makeSection. Normally, the input vectors will be of the same length, but as.ctd can also handle cases in which one or two of these is of unit length. For example, if only a temperature profile is available, as.ctd(35, T, p) could be used to construct a ctd object with constant salinity.

Value

An object of class "ctd" (for details, see read.ctd).

Author(s)

Dan Kelley

References

http://cchdo.ucsd.edu/CCHDO_DataSubmitGuide.pdf http://cchdo.ucsd.edu/manuals/pdf/90_1/chap4.pdf

See Also

The documentation for ctd-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
pressure <- 1:50
temperature <- 10 - tanh((pressure - 20) / 5) + 0.02*rnorm(50)
salinity <- 34 + 0.5*tanh((pressure - 20) / 5) + 0.01*rnorm(50)
ctd <- as.ctd(salinity, temperature, pressure)
summary(ctd)
plot(ctd)
```

`as.drifter` *Coerce data into drifter dataset*

Description

Coerces a dataset into a drifter dataset.

Usage

```
as.drifter(time, longitude, latitude, salinity, temperature, pressure,
           id, filename="", missingValue)
```

Arguments

`time`	time of observation
`longitude`	longitude of observation
`latitude`	latitude of observation
`salinity`	salinity of observation
`temperature`	temperature of observation
`pressure`	pressure of observation
`id`	drifter identifier
`filename`	source filename
`missingValue`	optional missing value, indicating data that should be taken as `NA`.

Details

This function assembles vectors into a `drifter` object, e.g. so that `plot.drifter` can be used.

Value

An object of `class` `"drifter"`.

Author(s)

Dan Kelley

See Also

The documentation for `drifter-class` explains the structure of drifter objects, and also outlines the other functions dealing with them.

`as.echosounder` *Coerce data into echosounder dataset*

Description

Coerces a dataset into a echosounder dataset.

Usage

```
as.echosounder(time, depth, a, src="",
               sourceLevel=220,
               receiverSensitivity=-55.4,
               transmitPower=0,
               pulseDuration=400,
               beamwidthX=6.5, beamwidthY=6.5,
               frequency=41800,
               correction=0)
```

Arguments

`time`	times of pings
`depth`	depths of samples within pings
`a`	matrix of amplitudes
`src`	optional string indicating data source
`sourceLevel`	source level, in dB(uPa@1m), denoted `sl` in [1 p15], where it is in units 0.1dB(uPa@1m)
`receiverSensitivity`	receiver sensivity of the main element, in dB(counts/uPa), denoted `rs` in [1 p15], where it is in units of 0.1dB(counts/uPa)
`transmitPower`	transmit power reduction factor, in dB, denoted `tpow` in [1 p10], where it is in units 0.1 dB.
`pulseDuration`	duration of transmited pulse in us
`beamwidthX`	x-axis -3dB one-way beamwidth in deg, denoted `bwx` in [1 p16], where the unit is 0.2 deg
`beamwidthY`	y-axis -3dB one-way beamwidth in deg, denoted `bwx` in [1 p16], where the unit is 0.2 deg
`frequency`	transducer frequency in Hz, denoted `fq` in [1 p16]
`correction`	user-defined calibration correction in dB, denoted `corr` in [1 p14], where the unit is 0.01dB.

Details

Creates an echosounder file. The defaults for e.g. `transmitPower` are taken from the `echosounder` dataset, and they are unlikely to make sense generally.

Value

An object of class "echosounder"; for details of this data type, see echosounder-class).

Author(s)

Dan Kelley

See Also

The documentation for echosounder-class explains the structure of echosounder objects, and also outlines the other functions dealing with them.

---

as.gps *Coerce data into a GPS dataset*

Description

Coerces a sequence of longitudes and latitudes into a GPS dataset.

Usage

```
as.gps(longitude, latitude, filename="")
```

Arguments

longitude	the longitude in decimal degrees, positive east of Greenwich, or a data frame with columns named latitude and longitude, in which case these values are extracted from the data frame and the second argument is ignored.
latitude	the latitude in decimal degrees, positive north of the Equator.
filename	name of file containing data (if applicable).

Details

This may be used when read.gps cannot read a file, or when the data have been manipulated.

Value

An object of class "gps" (for details, see read.gps).

Author(s)

Dan Kelley

References

The GPX format is described at http://www.topografix.com/GPX/1/1/.

See Also

The documentation for gps-class explains the structure of gps objects, and also outlines the other functions dealing with them.

`as.lisst` *Coerce data into a lisst object*

Description

Coerce data into a lisst object

Usage

```
as.lisst(data, filename="", year=0, tz="UTC", longitude=NA, latitude=NA)
```

Arguments

`data`	A table (or matrix) containing 42 columns, as in a LISST data file.
`filename`	Name of file containing the data.
`year`	Year in which the first observation was made. This is necessary because LISST timestamps do not indicate the year of observation. The default value is odd enough to remind users to include this argument.
`tz`	Timezone of observations. This is necessary because LISST timestamps do not indicate the timezone.
`longitude`	Longitude of observation.
`latitude`	Latitude of observation.

Details

If `data` contains fewer than 42 columns, an error is reported. If it contains more than 42 columns, only the first 42 are used. This is used by `read.lisst`, the documentation on which explains the meanings of the columns.

Value

An object of `class` `"lisst"` (for details, see `read.lisst`).

Author(s)

Dan Kelley

References

The LIST-100 users guide (version 4.65), which provided the information for this function, was downloaded in late May 2012, from http://www.sequoiasci.com/products/fam_LISST_100.aspx.

See Also

The documentation for `lisst-class` explains the structure of LISSTobjects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
```

`as.lobo` *Coerce data into lobo dataset*

Description

Coerce a dataset into a lobo dataset.

Usage

```
as.lobo(time, u, v, salinity, temperature, pressure, nitrate, fluorescence, filename="")
```

Arguments

`time`	vector of times of observation
`u`	vector of x velocity component observations
`v`	vector of y velocity component observations
`salinity`	vector of salinity observations
`temperature`	vector of temperature observations
`pressure`	vector of pressure observationss
`nitrate`	vector of nitrate observationss
`fluorescence`	vector of fluoresence observations
`filename`	source filename

Details

This function assembles vectors into a `lobo` object.

Value

An object of `class` `"lobo"`.

Author(s)

Dan Kelley

See Also

The documentation for `lobo-class` explains the structure of lobo objects, and also outlines the other functions dealing with them.

`as.met` *Coerce data into met dataset*

Description

Coerces a dataset into a met dataset.

Usage

```
as.met(time, temperature, pressure, u, v, filename="(constructed from data)")
```

Arguments

`time`	Vector of obseration times (or character strings that can be coerced into times).
`temperature`	vector of temperatures.
`pressure`	vector of pressures.
`u`	vector of eastward wind speed in m/s.
`v`	vector of northward wind speed in m/s.
`filename`	optional string indicating data source

Details

This function is used by `read.met`, and may be used to construct objects that behave as though read by that function.

Value

An object of `class` `"met"` (for details, see `met-class`).

Author(s)

Dan Kelley

See Also

The documentation for `met-class` explains the structure of `met` objects, and also outlines the other functions dealing with them.

`as.sealevel` *Coerce data into sea-level dataset*

Description

Coerces a dataset (minimally, a sequence of times and heights) into a sealevel dataset.

Usage

```
as.sealevel(elevation, time, header=NULL,
  stationNumber=NA, stationVersion=NA, stationName=NULL,
  region=NULL, year=NA, longitude=NA, latitude=NA, GMTOffset=NA,
  decimationMethod=NA, referenceOffset=NA, referenceCode=NA, deltat)
```

Arguments

`elevation`	a list of sea-level heights in metres, in an hourly sequence.
`time`	optional list of times, in POSIXct format. If missing, the list will be constructed assuming hourly samples, starting at 0000-01-01 00:00:00.
`header`	a character string as read from first line of a standard data file.
`stationNumber`	three-character string giving station number.
`stationVersion`	single character for version of station.
`stationName`	the name of station (at most 18 characters).
`region`	the name of the region or country of station (at most 19 characters).
`year`	the year of observation.
`longitude`	the longitude in decimal degrees, positive east of Greenwich.
`latitude`	the latitude in decimal degrees, positive north of the equator.
`GMTOffset`	offset from GMT, in hours.
`decimationMethod`	a coded value, with 1 meaning filtered, 2 meaning a simple average of all samples, 3 meaning spot readings, and 4 meaning some other method.
`referenceOffset`	?
`referenceCode`	?
`deltat`	optional interval between samples, in hours (as for the `ts` timeseries function). If this is not provided, and `t` can be understood as a time, then the difference between the first two times is used. If this is not provided, and `t` cannot be understood as a time, then 1 hour is assumed.

Details

The arguments are based on the standard data format, as described at `ftp://ilikai.soest.hawaii.edu/rqds/hourly.fmt`.

Value

An object of class "sealevel" (for details, see read.sealevel).

Author(s)

Dan Kelley

References

ftp://ilikai.soest.hawaii.edu/rqds/hourly.fmt.

See Also

The documentation for sealevel-class explains the structure of sealevel objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)

# Construct a year of M2 tide, starting at the default time
# 0000-01-01T00:00:00.
h <- seq(0, 24*365)
elevation <- 2.0 * sin(2*pi*h/12.4172)
sl <- as.sealevel(elevation)
summary(sl)

# As above, but start at the Y2K time.
time <- as.POSIXct("2000-01-01") + h * 3600
sl <- as.sealevel(elevation, time)
summary(sl)
```

---

as.section *Coerce ctd data into section dataset*

Description

Coerces a ctd dataset into a section dataset.

Usage

```
as.section(salinity, temperature, pressure, longitude, latitude,  station)
```

Arguments

salinity	Salinity, in a vector holding values for all stations.
temperature	Temperature, in a vector holding values for all stations.
pressure	Pressure, in a vector holding values for all stations.
longitude	Longitude, in a vector holding values for all stations.
latitude	Latitude, in a vector holding values for all stations.
station	Station identifier.

Details

Sometimes the data from an entire cruise will be combined into a single set. This function isolates individual stations from such data sets, and combines them into a section.

Value

An object of class "section" (for details, see read.section).

Author(s)

Dan Kelley

See Also

The documentation for section-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

---

as.tdr *Create a TDR object*

Description

Create a TDR (temperature-depth recorder) object.

Usage

```
as.tdr(time, temperature, pressure,
      filename="", instrumentType="rbr", serialNumber="", model="",
      pressureAtmospheric=NA, processingLog, debug=getOption("oceDebug"))
```

Arguments

time	a vector of times for the data.
temperature	temperatures at the give times.
pressure	pressures at the give times.
filename	optional name of file containing the data
instrumentType	type of instrument
serialNumber	serial number for instrument
model	instrument model type, e.g. "RBRduo"
pressureAtmospheric	optional atmospheric pressure, in the same unit as seawater pressure
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
debug	a flag that can be set to TRUE to turn on debugging.

Details

This is used by read.tdr to create tdr objects.

Value

An object of class "tdr", which is a list with elements detailed below.

data	a data table containing the time, temperature, and pressure data.
metadata	a list containing the following items
	header the header itself, as read from the input file.
	serialNumber serial number of instrument, inferred from first line of the header.
	loggingStart start time for logging, inferred from the header. Caution: this is often not the first time in the data, because the data may have been subsetted.
	samplePeriod seconds between samples, inferred from the header. Caution: this is often not the sampling period in the data, because the data may have been decimated.
processingLog	a processingLog of processing, in the standard oce format.

Author(s)

Dan Kelley

See Also

The documentation for tdr-class explains the structure of tdr objects, and also outlines the other functions dealing with them.

as.topo *Coerce data into topo dataset*

Description

Coerces a dataset into a topo (topographic) dataset.

Usage

```
as.topo(longitude, latitude, z, filename="")
```

Arguments

longitude	a vector of longitudes
latitude	a vector of latitudes
z	a matrix of heights (positive over land)
filename	name of data (used when called by read.topo)

Details

Mainly used by read.topo.

Value

An object of class "topo".

Author(s)

Dan Kelley

See Also

read.topo, which calls this.

as.windrose *Create a windrose object*

Description

Create a wind-rose object, typically for plotting with plot.windrose().

Usage

```
as.windrose(x, y, dtheta = 15, debug=getOption("oceDebug"))
```

Arguments

x	the x component of wind speed (or stress) *or* an object of class met (see met-class), in which case the u and v components of that object are used for the components of wind speed, and y here is ignored.
y	the y component of wind speed (or stress).
dtheta	the angle increment (in degrees) within which to classify the data
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

This is analagous to a histogram, but with breaks being angles.

Value

An object of class "windrose", which contains the standard oce slots named data, metadata and proxessingLog. The data slot contains

n the number of x values

x.mean the mean of the x values

y.mean the mean of the y values

theta the central angle (in degrees) for the class

count the number of observations in this class

mean the mean of the observations in this class

fivenum the fivenum vector for observations in this class (the min, the lower hinge, the median, the upper hinge, and the max)

Author(s)

Dan Kelley, with considerable help from Alex Deckmyn.

See Also

Use plot.windrose to produce a summary plot, and summary.windrose to produce a numerical summary.

Examples

```
library(oce)
xcomp <- rnorm(360) + 1
ycomp <- rnorm(360)
wr <- as.windrose(xcomp, ycomp)
summary(wr)
plot(wr)
```

bcdToInteger *Decode BCD to integer*

Description

Decode binary-coded-decimal to integer

Usage

```
bcdToInteger(x, endian=c("little", "big"))
```

Arguments

x	a raw value, or vector of raw values, coded in binary-coded decimal.
endian	character string indicating the endian-ness ("big" or "little"). The PC/intel convention is to use "little", and so most data files are in that format.

Value

An integer, or list of integers.

Author(s)

Dan Kelley

Examples

```
library(oce)
twenty.five <- bcdToInteger(as.raw(0x25))
thirty.seven <- as.integer(as.raw(0x25))
```

beamName *Name an acoustic-doppler beam.*

Description

Name a beam for an acoustic-doppler device, adjusting for beam number and for coordinate system.

Usage

```
beamName(x, which)
```

Arguments

x	an acoustic-doppler object, inheriting from class "adp" or "adv".
which	an integer indicating beam number.

Value

A character string containing a reasonable name for the beam, of the form "beam 1", etc., for beam coordinates, "east", etc. for enu coordinates, "u", etc. for "xyz", or "u'", etc., for "other" coordinates. The coordinate is determined by x@metadata$oce.coordinate.

Author(s)

Dan Kelley

See Also

This is used by read.oce.

---

beamToXyz *Change ADV or ADP coordinate systems*

Description

Convert velocity data from an acoustic-doppler velocimeter or acoustic-doppler profiler from one coordinate system to another.

Usage

```
beamToXyz(x, ...)
xyzToEnu(x, ...)
enuToOther(x, ...)
toEnu(x, ...)
```

Arguments

x	an object of class "adv" or "adp".
...	extra arguments that are passed on to the called function.

Details

Each of these functions checks the type of object, and calls the corresponding function, as appropriate. For example, beamToXyz calls beamToXyzAdp for an object that inhertis from "adp" or beamToXyzAdv for an object that inhertis from "adv".

Value

An object of the same type as x, but with x[["v"]] converted from beam coordinates to xyz coordinates, and with x[["oceCoordinate"]] changed from "beam" to "xyz".

Author(s)

Dan Kelley

See Also

The real work is done with specialized routines, beamToXyzAdp, beamToXyzAdv, xyzToEnuAdp, xyzToEnuAdv, enuToOtherAdp, enuToOtherAdv, toEnuAdp, and toEnuAdv.

---

beamToXyzAdp *Change ADP coordinate system*

Description

Convert ADP velocity components from a beam-based coordinate system to a xyz-based coordinate system.

Usage

```
beamToXyzAdp(x, debug=getOption("oceDebug"))
```

Arguments

x	an object of class "adp".
debug	a debugging flag, 0 for no debugging, and higher values for more and more debugging.

Details

The action depends on the type of object.

For a 3-beam aquadopp object, the beams are transformed into velocities using the matrix stored in the header.

For 4-beam rdi object, the beams are converted to velocity components using formulae from section 5.5 of *RD Instruments* (1998), viz. the along-beam velocity components B_1, B_2, B_3, and B_4 are used to calculate velocity components in a cartesian system referenced to the instrument using the following formulae: $u = ca(B_1 - B_2)$, $v = ca(B_4 - B_3)$, $w = -b(B_1 + B_2 + B_3 + B_4)$, and an estimate of the error in velocity is calculated using $e = d(B_1 + B_2 - B_3 - B_4)$

(Note that the multiplier on e is subject to discussion; RDI suggests one multiplier, but some oceanographers favour another.)

In the above, $c = 1$ if the beam geometry is convex, and $c = -1$ if the beam geometry is concave, $a = 1/(2\sin\theta)$, $b = 1/(4\cos\theta)$ and $d = a/\sqrt{2}$, where θ is the angle the beams make to the instrument "vertical".

Value

An object with the first 3 velocitiy indices having been altered to represent velocity components in xyz (or instrument) coordinates. (For rdi data, the values at the 4th velocity index are changed to represent the "error" velocity.)

To indicate the change, the value of metadata$oce.orientation is changed from beam to xyz.

Author(s)

Dan Kelley

References

1. R D Instruments, 1998. *ADP Coordinate Transformation, formulas and calculations.* P/N 951-6079-00 (July 1998).

2. WHOI/USGS-provided Matlab code for beam-enu transformation http://woodshole.er.usgs.gov/pubs/of2005-1429/MFILES/AQDPTOOLS/beam2enu.m

See Also

See `read.adp` for other functions that relate to objects of class `"adp"`.

---

beamToXyzAdv *Convert ADV from beam coordinates to xyz coordinates*

Description

Convert ADV velocity components from a beam-based coordinate system to a xyz-based coordinate system.

Usage

```
beamToXyzAdv(x, debug=getOption("oceDebug"))
```

Arguments

x	an object of class `"adv"`.
debug	a flag that, if non-zero, turns on debugging. Higher values yield more extensive debugging.

Details

The coordinate transformation is done using the transformation matrix contained in `x@metadata$transformation.matrix`, which is normally inferred from the header in the binary file. If there is no such matrix (e.g. if the data were streamed through a data logger that did not capture the header), `beamToXyzAdv` the user will need to store one in `x`, e.g. by doing something like the following: `x@metadata$transformation.matrix <- rbind(c( 291, 9716, -10002), c( 1409, 1409, 1409)) / 4096`.

Author(s)

Dan Kelley

References

http://www.nortek-as.com/lib/forum-attachments/coordinate-transformation

See Also

See read.adv for notes on functions relating to "adv" objects.

---

beamUnspreadAdp *Adjust ADP signal for spherical spreading*

Description

Compensate ADP signal strength for spherical spreading

Usage

```
beamUnspreadAdp(x, count2db=c(0.45, 0.45, 0.45, 0.45),
asMatrix=FALSE, debug=getOption("oceDebug"))
```

Arguments

x	an object of class "adp"
count2db	a set of coefficients, one per beam, to convert from beam echo intensity to decibels.
asMatrix	a boolean that indicates whether to return a numeric matrix, as opposed to returning an updated object (in which the matrix is cast to a raw value).
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

First, beam echo intensity is converted from counts to decibels, by multiplying by count2db. Then, the signal decrease owing to spherical spreading is compensated for by adding the term $20 \log 10(r)$, where r is the distance from the sensor head to the water from which scattering is occuring. r is given by x[["distance"]].

Value

An object of class "adp".

Author(s)

Dan Kelley

References

The coefficient to convert to decibels is a personal communication. The logarithmic term is explained in textbooks on acoustics, optics, etc.

See Also

See read.adp for other functions that relate to objects of class "adp".

Examples

```
library(oce)
data(adp)
plot(adp, which=5) # beam 1 echo intensity
adp.att <- beamUnspreadAdp(adp)
plot(adp.att, which=5) # beam 1 echo intensity
## Profiles
par(mar=c(4, 4, 1, 1))
a <- adp[["a", "numeric"]]            # second arg yields matrix return value
distance <- adp[["distance"]]
plot(apply(a,2,mean), distance, type='l', xlim=c(0,256))
lines(apply(a,2,median), distance, type='l',col='red')
legend("topright",lwd=1,col=c("black","red"),legend=c("original","attenuated"))
## Image
plot(adp.att, which="amplitude",col=oceColorsJet(100))
```

---

binApply *Apply a function to binned matrix data*

Description

Apply a function to binned matrix data

Usage

```
binApply1D(x, f, xbreaks, FUN)
binApply2D(x, y, f, xbreaks, ybreaks, FUN)
```

Arguments

x	a vector of numerical values.
y	a vector of numerical values.
f	a vector of data to which the elements of FUN may be supplied
xbreaks	values of x at the boundaries between bins; calculated using pretty if not supplied.
ybreaks	values of y at the boundaries between bins; calculated using pretty if not supplied.
FUN	function to apply to the data

Details

The function FUN is applied to f in bins specified by xbreaks and ybreaks. (If FUN is mean, consider using binMean2D instead, since it should be faster.)

Value

A list with the following elements: the breaks in x and y (xbreaks and ybreaks), the break midpoints (xmids and ymids), and a matrix containing the result of applying function FUN to f subsetted by these breaks.

Author(s)

Dan Kelley

Examples

```
library(oce)
## (a) 1D: salinity profile with median and quartile 1 and 3
data(ctd)
p <- ctd[["pressure"]]
S <- ctd[["salinity"]]
q1 <- binApply1D(p, S, pretty(p, 30), function(x) quantile(x, 1/4))
q3 <- binApply1D(p, S, pretty(p, 30), function(x) quantile(x, 3/4))
plotProfile(ctd, "salinity", col='gray', type='n')
polygon(c(q1$result, rev(q3$result)),
c(q1$xmids, rev(q1$xmids)), col='gray')
points(S, p, pch=20)

## (b) 2D: secchi depths in lat and lon bins
if (require(ocedata)) {
    data(secchi)
    col <- rev(oceColorsJet(100))[rescale(secchi$depth,
    xlow=0, xhigh=20,
    rlow=1, rhigh=100)]
    zlim <- c(0, 20)
    breaksPalette <- seq(min(zlim), max(zlim), 1)
    colPalette <- rev(oceColorsJet(length(breaksPalette)-1))
    drawPalette(zlim, "Secchi Depth", breaksPalette, colPalette)
    data(coastlineWorld)
    mapPlot(coastlineWorld, longitudelim=c(-5,20), latitudelim=c(50,66),
    grid=5, fill='gray', proj="lambert",
    parameters=c(lat0=50, lat1=65))
    bc <- binApply2D(secchi$longitude, secchi$latitude,
    pretty(secchi$longitude, 80),
    pretty(secchi$latitude, 40),
    f=secchi$depth, FUN=mean)
    mapImage(bc$xmids, bc$ymids, bc$result, zlim=zlim, col=colPalette)
    mapPolygon(coastlineWorld, col='gray')
}
```

---

binAverage *Bin-average a vector y, based on x values*

Description

Bin-average a vector y, based on x values

Usage

```
binAverage(x, y, xmin, xmax, xinc)
```

Arguments

x	a vector of numerical values.
y	a vector of numerical values.
xmin	x value at the lower limit of first bin; the minimum x will be used if this is not provided.
xmax	x value at the upper limit of last bin; the maximum x will be used if this is not provided.
xinc	width of bins, in terms of x value; 1/10th of xmax-xmin will be used if this is not provided.

Details

The y vector is averaged in bins defined for x. Missing values in y are ignored.

Value

A list with two elements: x, the mid-points of the bins, and y, the average y value in the bins.

Author(s)

Dan Kelley

Examples

```
library(oce)
## A. fake linear data
x <- seq(0, 100, 1)
y <- 1 + 2 * x
plot(x, y, pch=1)
ba <- binAverage(x, y)
points(ba$x, ba$y, pch=3, col='red', cex=3)

## B. fake quadratic data
y <- 1 + x ^2
plot(x, y, pch=1)
ba <- binAverage(x, y)
points(ba$x, ba$y, pch=3, col='red', cex=3)

## C. natural data
data(co2)
plot(co2)
```

```
avg <- binAverage(time(co2), co2, 1950, 2000, 2)
points(avg$x, avg$y, col='red')
```

---

binmapAdp *Bin-map an ADP object*

Description

Bin-map an ADP object, by interpolating velocities, backscatter amplitudes, etc., to uniform depth bins, thus compensating for the pitch and roll of the instrument. This only makes sense for ADP objects that are in beam coordinates.

Usage

```
binmapAdp(x, debug=getOption("oceDebug"))
```

Arguments

x	an object of class "adp"
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

This is a preliminary function that is still undergoing testing. Once the methods have been tested more, efforts may be made to speed up the processing, either by vectorizing in R or by doing some of the calculation in C.

Value

An object of class "adp".

Bugs

This only works for 4-beam RDI ADP objects.

Author(s)

Dan Kelley and Clark Richards

References

The method was devised by Clark Richards for use in his PhD work at Department of Oceanography at Dalhousie University.

See Also

See adp-class for a discussion of adp objects and notes on the many functions dealing with them.

Examples

```
## Not run:
library(oce)
beam <- read.oce("adp_rdi_2615.000",
                 from=as.POSIXct("2008-06-26", tz="UTC"),
                 to=as.POSIXct("2008-06-26 00:10:00", tz="UTC"),
                 longitude=-69.73433, latitude=47.88126)
beam2 <- binmapAdp(beam)
plot(enuToOther(toEnu(beam), heading=-31.5))
plot(enuToOther(toEnu(beam2), heading=-31.5))
plot(beam, which=5:8) # backscatter amplitude
plot(beam2, which=5:8)

## End(Not run)
```

---

binMean *Bin-count or bin-average f=f(x) or f=f(x,y), based on x or (x,y) values*

Description

Bin-count or bin-average f=f(x) or f=f(x,y), based on x or (x,y) values

Usage

```
binCount1D(x, xbreaks)
binCount2D(x, y, xbreaks, ybreaks)
binMean1D(x, f, xbreaks)
binMean2D(x, y, f, xbreaks, ybreaks)
```

Arguments

x	a vector of numerical values.
y	a vector of numerical values.
f	a vector of numerical values, f=f(x) for the 1D function and f=f(x,y) for the 2D function. If missing, the value component of the return value will consist entirely of NA values.
xbreaks	values of x at the boundaries between bins; calculated using pretty if not supplied.
ybreaks	values of y at the boundaries between bins; calculated using pretty if not supplied.

Details

The f vector is averaged in bins defined for x. Missing values in f are ignored.

Value

A list with the following elements: the breaks (`xbreaks`, along with `ybreaks` for the 2D case), the break mid-points (`xmids` along with `ymids` for the 2D case), the number of data points in each bin, `number`, and (for the "mean" case) the mean value of `f` value in the bins, `value`. For the 1D case, `number` and `mean` are vectors, whereas they are matrices for the 2D case. For plotting, the midpoints are more useful than the breaks, as shown in the examples.

Author(s)

Dan Kelley

Examples

```
library(oce)
## A. fake linear data
x <- seq(0, 100, 1)
f <- 1 + 2 * x
plot(x, f, pch=1)
m <- binMean1D(x, f)
points(m$xmids, m$result, pch=3, col='red', cex=3)

## B. fake quadratic data
f <- 1 + x ^2
plot(x, f, pch=1)
m <- binMean1D(x, f)
points(m$xmids, m$result, pch=3, col='red', cex=3)

## C. natural data
data(co2)
plot(co2, col='gray')
m <- binMean1D(time(co2), co2)
lines(m$xmids, m$result, type='o')

## D. 2D
x <- runif(500)
y <- runif(500)
f <- x - y
m <- binMean2D(x, y, f)
plot(x, y)
grid()
contour(m$xmids, m$ymids, m$result, add=TRUE)
```

---

byteToBinary *Format bytes as binary*

Description

Format bytes as binary

Usage

```
byteToBinary(x, endian=c("little", "big"))
```

Arguments

x	an integer to be interpreted as a byte.
endian	character string indicating the endian-ness ("big" or "little"). The PC/intel convention is to use "little", and so most data files are in that format.

Value

A character string representing the bit strings for the elements of x.

Author(s)

Dan Kelley

Examples

```
library(oce)
x <- 0:16
print(byteToBinary(x))
```

---

cm *Current-meter record*

Description

This is sample current meter record from an InterOcean S4 devide.

Usage

```
data(cm)
```

Source

This is a snippet from a time series recorded as part of the SLEIWEX experiment (http://myweb.dal.ca/kelley/SLEIWEX/index.php.

See Also

The documentation for cm-class in the Oce package explains the structure of cm objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(cm)
summary(cm)
plot(cm)

## End(Not run)
```

`cm-class` *Class to store current meter data*

Description

Class to store current meter data, with standard slots `metadata`, `data` and `processingLog`.

Methods

Extracting values: Data may be accessed as e.g. codecm[["time"]], where the string could also be e.g. `"u"` or `"v"` for column data, or `"longitude"` or `"latitude"` for scalars. (The names of the columns are displayed with `show()`. The name of the source file is found with `"filename"`.

Assigning values: Everything that may be accessed may also be assigned, e.g. `cm[["u"]] <- rep(35,100)`.

Overview of contents: The `show` method (e.g. `show(cm)`) displays information about the object.

Author(s)

Dan Kelley

See Also

A file containing CM profile data may be read with `read.cm`. Statistical summaries are provided with `summary.cm`, while overviews are provided by `show`. Plots are provided with `plot.cm`.

`coastline-class` *Class to store coastline data*

Description

Class to store coastline data, with standard slots `metadata` (containing `fillable` and `filename`), `data` (containing `longitude` and `latitude`) and `processingLog`.

Methods

Extracting values: Data may be accessed as e.g. `coastline[["longitude"]]` or `coastline[["latitude"]]`.

Assigning values: Latitude may be changed with e.g. `coastline[["longitude"]] <- value`, and of course the same can be done for `latitude`.

Overview of contents: The `show` method (e.g. `show(coastline)`) displays information about the object.

Author(s)

Dan Kelley

See Also

Use `as.coastline` to convert data to this form, `read.coastline` to read data in various formats, and `plot.coastline` to plot coastlines.

`coastlineBest` *Find the name of the best coastline file*

Description

Find the name of the most appropriate coastline for a given locale

Usage

```
coastlineBest(lonRange, latRange, span,
    debug=getOption("oceDebug"))
```

Arguments

`lonRange`	range of longitude for locale
`latRange`	range of latitude for locale
`span`	span of domain in km (if provided, previous two arguments are ignored).
`debug`	set to a positive value to get debugging information during processing.

Details

Checks `coastlineWorld`, `coastlineWorldFine` and `coastlineWorldCoarse`, in that order, to find the one most appropriate for the locale.

Value

The name of a coastline that can be loaded with `data()`.

Author(s)

Dan Kelley

`coastlineWorld` *World coastline*

Description

World coastline, in any of three resolutions

Usage

```
data(coastlineWorld)
```

Details

In each case, the longitudes are in the range from -180 to 180 degrees, i.e. western longitudes have negative values. Large lakes (particularly the Great Lakes) are missing from these datasets, since the intention is for use in ocean mapping. The resolutions of the three coastlines are listed below, along with typical applications.

- `coastlineWorld` is a coarse resolution 1:110M (with 10,696 points), suitable for world-scale plots plotted at a small size, e.g. inset diagrams
- `coastlineWorldMedium` resolution 1:50M (with 100,954 points), suitable for world- or basin-scale plots
- `coastlineWorldFine` resolution 1:10M (with 552,670 points), suitable for shelf-scale plots

Author(s)

Dan Kelley

Source

Downloaded from http://www.naturalearthdata.com, in `ne_110m_admin_0_countries.shp`.

See Also

The `ocedata` package provides two more coastlines with better resolution: `coastlineWorldMedium` and `coastlineWorldFine`.

The documentation for `coastline-class` explains the structure of coastline objects and discusses functions that deal with them.

The `maps` package provides a database named `world` that has 27221 points.

`colormap` *Calculate color map*

Description

Map values to colors, for use in palettes and plots.

Usage

```
colormap(z,
         zlim, zclip=FALSE,
         breaks, col=oceColorsJet,
         name, x0, x1, col0, col1, blend=0,
         missingColor,
         debug=getOption("oceDebug"))
```

Arguments

`z`	an optional vector or other set of numerical values to be examined. If `z` is given, the return value will contain an item named `zcol` that will be a vector of the same length as `z`, containing a color for each point. If `z` is not given, `zcol` will contain just one item, the color `"black"`.
`zlim`	optional vector containing two numbers that specify the `z` limits for the colorscale. If not given, this will be determined from the other arguments, as follows. If `name` is given, then the `range` of numerical values contained therein will be used for `zlim`. Otherwise, if `z` is given, then its `rangeExtended` sets `zlim`. Otherwise, if `x0` and `x1` are given, then their `range` sets `zlim`. Otherwise, there is no way to infer `zlim` and indeed there is no way to construct a colormap, so an error is reported. It is an error to specify both `zlim` and `breaks`, if the length of the latter does not equal 1.
`zclip`	logical, indicating whether to clip the colors to those corresponding to `zlim`, if the latter is provided. Clipped regions will be colored with `missingColor`.
`breaks`	an optional indication of break points between color levels (see `image`). If this is provided, the arguments `name` through `blend` are all ignored (see "Details"). If it is provided, then it may either be a vector of break points, or a single number indicating the desired number of break points to be computed with `pretty(z, breaks)`. In either case of non-missing `breaks`, the resultant break points must number 1 plus the number of colors (see `col`).
`col`	either a vector of colors or a function taking a numerical value as its single argument and returning a vector of colors. The value of `col` is ignored if `name` is provided, or if `x0` through `col1` are provided.
`name`	an optional string naming a built-in colormap (one of `"gmt_relief"`, `"gmt_ocean"`, `"gmt_globe"` or `"gmt_gebco"`) or the name of a file or URL that contains a color map specification in GMT format, e.g. `"http://www.beamreach.org/maps/gmt/share/cpt/GMT_glob` If `name` is provided, then `x0`, `x1`, `col0` and `col1` are all ignored.

`x0, x1, col0, col1`
: Vectors that specify a color map. They must all be the same length, with `x0` and `x1` being numerical values, and `col0` and `col1` being colors. The colors may be strings (e.g. `"red"`) or colors as defined by `rgb` or `hsv`.

`blend`
: a number indicating how to blend colors within each band. This is ignored except when `x0` through `col1` are supplied. A value of 0 means to use `col0[i]` through the interval `x0[i]` to `x1[i]`. A value of 1 means to use `col1[i]` in that interval. A value between 0 and 1 means to blend between the two colors according to the stated fraction. Values exceeding 1 mean to break up the domain into `blend` sub-intervals; making this a large number yields a palette that blends smoothly between the fixed colors specified in `col0` and `col1`.

`missingColor`
: color to use for missing values. If not provided, this will be `"gray"`, unless `name` is given, in which case it comes from that color table.

`debug`
: a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

This is a multi-purpose function that generally links ("maps") numerical values to colors. The return value can specify colors for points on a graph, or `breaks` and `col` vectors that are suitable for use by `drawPalette`, `imagep` or `image`.

There are three ways of specifying color schemes, and `colormap` works by checking for each condition in turn.

- *Case A*. Supply `z` but nothing else. In this case, `breaks` will be set to `pretty(z, 10)` and things are otherwise as in case 2.
- *Case B*. Supply `breaks`. In this case, `breaks` and `col` are used together to specify a color scheme. If `col` is a function, then it is expected to take a single numerical argument that specifies the number of colors, and this number will be set to `length(breaks)-1`. Otherwise, `col` may be a vector of colors, and its length must be one less than the number of breaks. (NB. if `breaks` is given, then all other arguments except `col` and `missingColor` are ignored.)
- *Case C*. Do not supply `breaks`, but supply `name` instead. This `name` may be the name of a pre-defined color palette (`"gmt_relief"`, `"gmt_ocean"`, `"gmt_globe"` or `"gmt_gebco"`), or it may be the name of a file (including a URL) containing a color map in the GMT format (see "References"). (NB. if `name` is given, then all other arguments except `z` and `missingColor` are ignored.)
- *Case D*. Do not supply either `breaks` or `name`, but instead supply each of `x0`, `x1`, `col0`, and `col1`. These values are specify a value-color mapping that is similar to that used for GMT color maps. The method works by using `seq` to interpolate between the elements of the `x0` vector. The same is done for `x1`. Similarly, `colorRampPalette` is used to interpolate between the colors in the `col0` vector, and the same is done for `col1`.

Value

A list containing the following (not necessarily in this order)

- `zcol`, a vector of colors for `z`, if `z` was provided, otherwise `"black"`

- zlim, a two-element vector suitable as the argument of the same name supplied to image or imagep
- breaks and col, vectors of breakpoints and colors, suitable as the same-named arguments to image or imagep
- x0 and x1, numerical vectors of the sides of color intervals, and col0 and col1, vectors of corresponding colors. The meaning is the same as on input. The purpose of returning these four vectors is to permit users to alter color mapping, as in example 3 in "Examples".
- missingColor, a color that could be used to specify missing values, e.g. as the same-named argument to imagep. If this is supplied as an argument, its value is repeated in the return value. Otherwise, its value is either "gray" or, in the case of name being given, the value in the GMT color map specification.

Author(s)

Dan Kelley

References

Information on GMT software is given at http://gmt.soest.hawaii.edu. Diagrams showing the GMT color schemes are at http://www.geos.ed.ac.uk/it/howto/GMT/CPT/palettes.html, and numerical specifications for some color maps are at http://www.beamreach.org/maps/gmt/share/cpt, http://soliton.vm.bytemark.co.uk/pub/cpt-city, and other sources.

Examples

```
library(oce)

## Example 1. color scheme for points on xy plot
x <- seq(0, 1, length.out=40)
y <- sin(2 * pi * x)
par(mar=c(3, 3, 1, 1))
mar <- par('mar') # prevent margin creep by drawPalette()
## First, default breaks
c <- colormap(y)
drawPalette(c$zlim, col=c$col, breaks=c$breaks)
plot(x, y, bg=c$zcol, pch=21, cex=1)
grid()
par(mar=mar)
## Second, 100 breaks, yielding a smoother palette
c <- colormap(y, breaks=100)
drawPalette(c$zlim, col=c$col, breaks=c$breaks)
plot(x, y, bg=c$zcol, pch=21, cex=1)
grid()
par(mar=mar)

## Example 2. topographic image with a standard color scheme
par(mfrow=c(1,1))
data(topoWorld)
cm <- colormap(name="gmt_globe")
imagep(topoWorld, breaks=cm$breaks, col=cm$col)
```

```
## visualize color map
# plot(seq_along(cm$x0), cm$x0, pch=21, bg=cm$col0)
# grid()
# points(seq_along(cm$x1), cm$x1, pch=21, bg=cm$col1)

## Example 3. topographic image with modified colors
cm <- colormap(name="gmt_globe")
deep <- cm$x0 < -4000
cm$col0[deep] <- 'black'
cm$col1[deep] <- 'black'
cm <- colormap(x0=cm$x0, x1=cm$x1, col0=cm$col0, col1=cm$col1)
imagep(topoWorld, breaks=cm$breaks, col=cm$col)

## Example 4. image of world topography with water colorized
##            smoothly from violet at 8km depth to blue
##            at 4km depth, then blending in 0.5km increments
##            to white at the coast, with tan for land.
cm <- colormap(x0=c(-8000, -4000, 0, 100),
               x1=c(-8000, -4000, 0, 100),
               col0=c("violet","blue","white","tan"),
               col1=c("violet","blue","white","tan"),
               blend=c(100, 8, 0))
lon <- topoWorld[['longitude']]
lat <- topoWorld[['latitude']]
z <- topoWorld[['z']]
imagep(lon, lat, z, breaks=cm$breaks, col=cm$col)
contour(lon, lat, z, levels=0, add=TRUE)
```

`coriolis` *Coriolis parameter on rotating earth*

Description

Compute f, the Coriolis parameter as a function of latitude.

Usage

```
coriolis(lat, degrees=TRUE)
```

Arguments

`lat`	Latitude in °N or radians north of the equator.
`degrees`	Flag indicating whether degrees are used for latitude; if set to `FALSE`, radians are used.

Value

Coriolis parameter [radian/s].

Author(s)

Dan Kelley

References

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

Examples

```
C <- coriolis(45) # 1e-4
```

`ctd` *A CTD profile in Halifax Harbour*

Description

A CTD profile measured in Halifax Harbour in 2003, by students enrolled in the author's Physical Oceanography class at Dalhousie University.

Usage

```
data(ctd)
```

Source

This is a station within the `section` dataset, and details are provided in the documentation for the latter.

See Also

The documentation for `ctd-class` in the Oce package explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(ctd)
plot(ctd, which=c(1,2,3,5), coastline="coastlineWorldFine")

## End(Not run)
```

`ctd-class` *Class to store hydrographic data*

Description

Class to store hydrographic data, with standard slots `metadata`, `data` and `processingLog`.

Methods

Consider a CTD object named `ctd`.

Accessing column values.

Column data may be accessed as e.g. `ctd[["salinity"]]`, `ctd[["temperature"]]`, `ctd[["pressure"]]`, etc. There may be other columns also, depending on the CTD configuration.

Depth is accessed with e.g. `ctd[["depth"]]`, while its negative, the vertical coordinate, is accessed with e.g. `ctd[["z"]]`; note that these are calculated using `swDepth` and `swZ`, and that any values that may have been read in a data file are ignored.

Potential temperature is calculated with `ctd[["potential temperature"]]`. The TEOS-10 defined quantity called "absolute salinity" is retrieved with `ctd[["absolute salinity"]]`, `ctd[["absoluteSalinity"]]`, or `ctd[["SA"]]`, while "conservative temperature" is retrieved with `ctd[["conservative temperature"]]`, `ctd[["conservativeTemperature"]]`, or `ctd[["CT"]]`. (None of the TEOS-10 quantities are stored in the data; rather, they are computed if requested.)

Accessing scalar values.

Various scalar quantities are also available, e.g. `ctd[["longitude"]]`, etc.

Accessing derived values.

The square of buoyancy frequency N is retrieved with `ctd[["N2"]]` or `swN2`, density ratio with `ctd[["Rrho"]]` and spiciness with `ctd[["spice"]]`.

Assigning values.

Items stored in the object may be altered with e.g. `ctd[["salinity"]] <- rep(35,10)`. Note that this method will not work with derived quantities such as conservative temperature, etc.

Overview of contents.

The `show` method (e.g. `show(ctd)`) displays information about the object.

Author(s)

Dan Kelley

See Also

A file containing CTD profile data may be read with `read.ctd`, and a CTD object can also be created with `as.ctd`. See `read.ctd` for references on data formats used in CTD files.

Statistical summaries are provided by `summary.ctd`, while `show` displays an overview.

CTD objects may be plotted with plot.ctd, which does much of its work by calling plotProfile or plotTS, both of which can also be called by the user, to get fine control over the plots.

A CTD profile can be isolated from a larger record with ctdTrim, a task made easier when plotScan is used to examine the results. Towyow data can be split up into sets of profiles (ascending or descending) with ctdFindProfiles. CTD data may be smoothed and/or cast onto specified pressure levels with ctdDecimate.

Low-level manipulation may be done with functions such as ctdAddColumn and ctdUpdateHeader. Additionally, many of the contents of CTD objects may be altered with the [[]] scheme discussed above, and skilled uses may also manipulate the contents directly.

---

ctdAddColumn *Add a column to a CTD file*

Description

Add a column to a ctd file, updating the header as appropriate.

Usage

```
ctdAddColumn(x, column, name, label, unit, debug=getOption("oceDebug"))
```

Arguments

x	A ctd object, e.g. as read by read.ctd.
column	A column of data to be inserted, in the form of a numeric vector, whose length matches that of columns in the objecct.
name	Character string indicating the name this column is to have in the data slot of x.
label	Optional character string or expression indicating the name of the column, as it will appear in plot labels. (If not given, name will be used.)
unit	Optional character string indicating unit.
debug	set to a positive value to get debugging information during processing.

Details

This function adds a column to the object's data slot.

Value

An object of class "ctd", with a new column.

Author(s)

Dan Kelley

See Also

The documentation for ctd-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(ctd)
SS <- ctd[["salinity"]]^2
ctdNew <- ctdAddColumn(ctd, SS, "ss",
expression(paste(S^2, " [", PSU^2, " ]")))
plotProfile(ctdNew, "ss")
```

---

ctdDecimate *Decimate a CTD profile*

Description

Interpolate a CTD profile to specified pressure values.

Usage

```
ctdDecimate(x, p=1, method="approx", e=1.5, debug=getOption("oceDebug"))
```

Arguments

x	a ctd object, e.g. as read by read.ctd.
p	pressure increment, or vector of pressures. In the first case, pressures from 0dbar to the rounded maximum pressure are used, incrementing by p dbars. If a vector of pressures is given, interpolation is done to these pressures.
method	the method to be used for calculating decimated values. This may be a function or a string naming a built-in method. The built-in methods are "boxcar" (the default method, based on a local average), "approx" (based on linear interpolation between neighboring points), "lm" (based on local regression, with e setting the size of the local region), "rr" (for the Reineger and Ross method, carried out with oceApprox) and "unesco" (for the UNESCO method, carried out with. oceApprox. If method is a function, then it must take three arguments, the first being pressure, the second being an arbitrary variable in another column of the data, and the third being a vector of target pressures at which the calculation is carried out, and the return value must be a vector. See "Examples".
e	is an expansion coefficient used to calculate the local neighbourhoods for the "boxcar" and "lm" methods. If e=1, then the neighbourhood for the i-th pressure extends from the (i-1)-th pressure to the (i+1)-th pressure. At the endpoints it is assumed that the outside bin is of the same pressure range as the first inside bin. For other values of e, the neighbourhood is expanded linearly in each direction. If the "lm" method produces warnings about "prediction from a rank-deficient fit", a larger value of "e" should be used.
debug	a Boolean, set to TRUE to debug the reading process.

Details

The "approx" method is best for bottle data, in which the usual task is to interpolate from a coarse sampling grid to a finer one. For CTD data, the "boxcar" method may be the best choice, because the task is normally to sub-sample, and some degree of smoothing is usually desired. (The "lm" method is quite slow, and the results are similar to those of the boxcar method.)

NB. A sort of numerical cabeling effect can result from this procedure, but it can be avoided as follows

```
xd <- ctdDecimate(x)
xd[["sigmaTheta"]] <- swSigmaTheta(xd[["salinity"]],xd[["temperature"]],xd[["pressure"]])
```

Value

An object of class "ctd", with pressures that are as set by the "p" parameter and all other properties modified appropriately.

Author(s)

Dan Kelley

References

R.F. Reiniger and C.K. Ross, 1968. A method of interpolation with application to oceanographic data. *Deep Sea Research*, **15**, 185-193.

See Also

The documentation for ctd-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(ctd)
plotProfile(ctd, "salinity", ylim=c(10, 0))
p <- seq(0, 45, 1)
ctd2 <- ctdDecimate(ctd, p=p)
lines(ctd2[["salinity"]], ctd2[["pressure"]], col="blue")
p <- seq(0, 45, 1)
ctd3 <- ctdDecimate(ctd, p=p, method=function(x,y,xout)
                    predict(smooth.spline(x, y, df=30), p)$y)
lines(ctd3[["salinity"]], ctd3[["pressure"]], col="red")
```

ctdFindProfiles *Find profiles within a towyow CTD record*

Description

Examine the pressure record looking for extended periods of either ascent or descent, and return either indices to these events or a vector of CTD records containing the events.

Usage

```
ctdFindProfiles(x,
  cutoff=0.5, minLength=10, minHeight=0.1*diff(range(x[["pressure"]])),
  direction=c("descending", "ascending"), arr.ind=FALSE,
  debug=getOption("oceDebug"), ...)
```

Arguments

x	A ctd object, as read by read.ctd or created by as.ctd.
cutoff	criterion on pressure difference; see "Details".
minLength	lower limit on number of points in candidate profiles.
minHeight	lower limit on height of candidate profiles.
direction	string indicating the travel direction to be selected
arr.ind	should array indices be returned, or a vector of ctd objects?
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	extra arguments that are passed to smooth.spline.

Details

The method works by examining the pressure record. First, this is smoothed using smooth.spline, which is provided with any extra arguments as supplied to the present function, e.g. ctdFindProfiles(..., df=10) uses a spline with 10 degrees of freedom. The spline is then first differenced with diff. Median values of the positive and negative first-difference values are then multiplied by cutoff. This establishes criteria for any given point to be in an ascending profile, a descending profile, or a non-profile. Contiguous regions are then found, and those that have fewer than minLength points are discarded. Then, those that have pressure ranges less than minHeight are discarded.

It is often necessary to pass the resultant profiles through ctdTrim, to remove artifacts such as an equilibration phase, etc.

Value

If arr.ind=TRUE, a data frame with columns start and end, the indices of the downcasts. Otherwise, a vector of ctd objects.

Author(s)

Dan Kelley

See Also

The documentation for `ctd-class` explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
d <- read.csv("towyow.csv", header=TRUE)
towyow <- as.ctd(d$salinity, d$temperature, d$pressure)

casts <- ctdFindProfiles(towyow)
par(mfrow=c(length(casts), 3))
for (cast in casts) {
  plotProfile(cast, "salinity")
  plotProfile(cast, "temperature")
  plotTS(cast, type='o')
}

## End(Not run)
```

---

ctdRaw *Seawater CTD profile, without trimming of extraneous data*

Description

This is sample CTD profile provided for testing. It includes not just the (useful) portion of the dataset during which the instrument was being lowered, but also data from the upcast and from time spent near the surface. Spikes are also clearly evident in the pressure record. With such real-world wrinkles, this dataset provides a good example of data that need trimming with the Oce function `ctdTrim`.

Usage

```
data(ctdRaw)
```

Author(s)

Dan Kelley

Source

The data were acquired near the centre of the Bedford Basin of the Halifax Harbour, during an October 2003 field trip of Dalhousie University's Oceanography 4120/5120 class.

See Also

The documentation for `ctd-class` in the Oce package explains the structure of CTD objects, and also outlines the other functions dealing with them.

`ctdTrim` *Trim start/end portions of a CTD cast*

Description

Trim start/end portions of a CTD cast.

Usage

```
ctdTrim(x, method=c("downcast","index","range"),
inferWaterDepth=TRUE, removeDepthInversions=FALSE,
parameters, debug=getOption("oceDebug"))
```

Arguments

`x` A `ctd` object, e.g. as read by `read.ctd`.

`method` Various methods exist, some of which use parameters:

"downcast" Select only data for which the CTD is descending (or ascending, if that is the overall trend). This is done in stages.

1. Step 1.The pressure data are despiked with a smooth() filter with method "3R". This removes wild spikes that arise from poor instrument connections, etc.
2. Step 2.If no `parameters` are given, then any data with negative pressures are deleted. If there is a parameter named `pmin`, then that pressure (in decibars) is used instead as the lower limit. This is a commonly-used setup, e.g. `ctdTrim(ctd, parameters=list(pmin=1))` removes the top decibar (roughly 1m) from the data.
3. Step 3.The maximum pressure is determined, and data acquired subsequent to that point are deleted. This removes the upcast and any subsequent data.
4. Step 4.An initial equilibrium phase is removed by a regression of pressure on scan number. The model has zero pressure for some initial portion, and then a constant increase with scan number. Then this initial zero-pressure portion is deleted. (The regression may fail, and if so, a warning is printed, and this step is skipped.)

"index" Select values only in the list of indices specified in `parameters`. The indices may be integers, *e.g.* `parameters=10:30` selects data points with indices 10, 11, ... 30, or logicals, *e.g.* `parameters=c(TRUE,TRUE, ...)`.

"range" Select data based on the value of the column named `parameters$item`. This may be by range or by critical value. By range: select values between `parameters$from` (the lower limit) and `parameters$to` (the upper limit) By critical value: select if the named column exceeds the value. For example, `ctd2 <- ctdTrim(ctd, "range", parameters=list(item="scan", from=5))` starts at scan number 5 and continues to the end, while `ctdTrim(ctd,"range",parameters=list(i` also starts at scan 5, but extends only to scan 100.

`inferWaterDepth`
: boolean, set to `TRUE` if the water depth (`metadata$water.depth`) is to be determined from the maximum of the trimmed pressure. (This is ignored if the original data file contained the water depth in the header.)

`removeDepthInversions`
: Boolean value indicating whether to remove any levels at which depth is less than, or equal to, a depth above. (This is needed if the object is to be assembled into a section, unless `ctdDecimate` will be used, which will remove the inversions.

`parameters`
: A list whose elements depend on the method; see above.

`debug`
: a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

For normal CTD profiling, the goal is to isolate only the downcast, discarding measurements made in the air, in an equilibration phase in which the device is held below the water surface, and then the upcast phase that follows the downcast. This is handled reasonably well by `ctdTrim` with `method="downcast"`. (The datasets provided with `oce` were produced this way.)

However, for detailed work it makes sense to do things semi-manually. The eye is simply better at handling exceptional cases. The process is simple: use `plotScan()` to get an idea of the scan indices of the downacast, and then use `ctdTrim` with `method="index"`. A few trials will normally identify the downcast very well.

Value

An object of `class` `"ctd"`, with data having been trimmed in some way.

Author(s)

Dan Kelley

References

The Seabird CTD instrument is described at `http://www.seabird.com/products/spec_sheets/19plusdata.htm`.

See Also

The documentation for `ctd-class` explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(ctdRaw)
plot(ctdRaw) # barely recognizable, due to pre- and post-cast junk
plot(ctdTrim(ctdRaw)) # looks like a real profile ...
plot(ctdDecimate(ctdTrim(ctdRaw),method="boxcar")) # ... smoothed
```

---

ctdUpdateHeader *Update a CTD header*

Description

Update the header of a `ctd` object

Usage

```
ctdUpdateHeader(x, debug=FALSE)
```

Arguments

`x`	A `ctd` object, e.g. as read by `read.ctd`.
`debug`	Set to `TRUE` for debugging.

Details

Update the header of a `ctd` object, e.g. adjusting `nvalues` and the `span` of each column. This is done automatically by `ctdTrim`, for example.

Value

A new `ctd` object.

Author(s)

Dan Kelley

References

The Seabird CTD instrument is described at http://www.seabird.com/products/spec_sheets/19plusdata.htm.

See Also

The documentation for `ctd-class` explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(ctd)
ctd[["pressure"]] <- ctd[["pressure"]] + 3
ctdNew <- ctdUpdateHeader(ctd)
```

`ctimeToSeconds` *Interpret a character string as a time interval*

Description

Interpret a character string as a time interval

Usage

```
ctimeToSeconds(ctime)
```

Arguments

`ctime`	a character string (see 'Details'.

Details

Strings are of the form MM:SS or HH:MM:SS.

Value

A numeric value, the number of seconds represented by the string.

Author(s)

Dan Kelley

See Also

See `secondsToCtime`, the inverse of this.

Examples

```
library(oce)
cat("10      = ", ctimeToSeconds("10"), "s\n", sep="")
cat("01:04   = ", ctimeToSeconds("01:04"), "s\n", sep="")
cat("1:00:00 = ", ctimeToSeconds("1:00:00"), "s\n", sep="")
```

`decimate` *Smooth and decimate an oce object*

Description

Smooth and decimate, or subsample, an oce object.

Usage

```
decimate(x, by=10, to, filter, debug=getOption("oceDebug"))
```

Arguments

`x`	an `oce` object containing a `data` element.
`by`	an indication of the subsampling. If this is a single number, then it indicates the spacing between elements of `x` that are selected. If it is two numbers (a condition only applicabile if `x` is an `echosounder` object, at present), then the first number indicates the time spacing and the second indicates the depth spacing.
`to`	Indices at which to subsample. If given, this over-rides `by`.
`filter`	optional list of numbers representing a digital filter to be applied to each variable in the `data` slot of `x`, before decimation is done. If not supplied, then the decimation is done strictly by sub-sampling.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

Later on, other methods will be added, and `ctdDecimate` will be retired in favour of this, a more general, function. The filtering is done with the `filter` function of the stats package.

Value

An object of `class` `"oce"` that has been subsampled appropriately.

Bugs

Only a preliminary version of this function is provided in the present package. It only works for objects of class `echosounder`, for which the decmation is done after applying a running median filter and then a boxcar filter, each of length equal to the corresponding component of `by`.

Author(s)

Dan Kelley

See Also

Filter coefficients may be calculated using makeFilter. (Note that ctdDecimate will be retired when the present function gains equivalent functionality.)

Examples

```
library(oce)
data(adp)
plot(adp)
adp.dec <- decimate(adp,by=2,filter=c(1/4,1/2,1/4))
plot(adp.dec)
```

---

decodeHeader *Decode a Nortek header*

Description

Decode data in a Nortek ADV or ADP header.

Usage

```
decodeHeaderNortek(buf,
                   type=c("aquadoppHR", "aquadoppProfiler", "aquadopp", "vector"),
                     debug=getOption("oceDebug"), ...)
```

Arguments

buf	a "raw" buffer containing the header
type	type of device
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	additional arguments, passed to called routines.

Details

Decodes the header in a binary-format Nortek ADV/ADP file. This function is designed to be used by read.adp and read.adv, but can be used directly as well. The code is based on information in the Nortek System Integrator Guide (2008) and on postings on the Nortek "knowledge center" discussion board. One might assume that the latter is less authorative than the former. For example, the inference of cell size follows advice found at http://www.nortekusa.com/en/knowledge-center/forum/hr-profilers/736804717 (downloaded June 2012)), which contains a typo in an early posting that is corrected later on.

Value

A list containing elements hardware, head, user and offset. The easiest way to find the contents of these is to run this function with debug=3.

Author(s)

Dan Kelley and Clark Richards

References

1. Information on Nortek profilers (including the System Integrator Guide, which explains the data format byte-by-byte) is available at http://www.nortekusa.com/. (One must join the site to see the manuals.)

2. The Nortek Knowledge Center http://www.nortekusa.com/en/knowledge-center may be of help if problems arise in dealing with data from Nortek instruments.

See Also

Most users should employ the functions read.adp and read.adv instead of this one.

---

decodeTime *Decode a time, similar to as.POSIXct()*

Description

Decode a time, similar to as.POSIXct()

Usage

```
decodeTime(time, timeFormats, tz="UTC")
```

Arguments

time	a character string with an indication of the time
timeFormats	optional vector of time formats to use, as for as.POSIXct
tz	time zone

Details

Each format in timeFormats is used in turn as the format argument to as.POSIXct, and the first that produces a non-NA result is used. If timeFormats is missing, the following formats are tried, in the stated order:

- "%b %d %Y %H:%M:%S" (e.g. "Jul 1 2013 01:02:03") and "%b %d %Y" (e.g. "Jul 1 2013")
- "%B %d %Y %H:%M:%S" (e.g. "July 1 2013 01:02:03") and "%B %d %Y" (e.g. "July 1 2013")
- "%d %b %Y %H:%M:%S" (e.g. "1 Jul 2013 01:02:03") and "%d %b %Y" (e.g. "1 Jul 2013")
- "%d %B %Y %H:%M:%S" (e.g. "1 July 2013 01:02:03") and "%d %B %Y" (e.g. "1 July 2013")
- "%Y-%m-%d %H:%M:%S" (e.g. "2013-07-01 01:02:03") and "%Y-%m-%d" (e.g. "2013-07-01")
- "%Y-%b-%d %H:%M:%S" (e.g. "2013-July-01 01:02:03") and "%Y-%b-%d" (e.g. "2013-Jul-01")
- "%Y-%B-%d %H:%M:%S" (e.g. "2013-July-01 01:02:03") and "%Y-%B-%d" (e.g. "2013-July-01")

- "%d-%b-%Y %H:%M:%S" (e.g. "01-Jul-2013 01:02:03") and "%d-%b-%Y" (e.g. "01-Jul-2013")
- "%d-%B-%Y %H:%M:%S" (e.g. "01-July-2013 01:02:03") and "%d-%B-%Y" (e.g. "01-July-2013")
- "%Y/%b/%d %H:%M:%S" (e.g. "2013/Jul/01 01:02:03") and "%Y/%b/%d" (e.g. "2013/Jul/01")
- "%Y/%B/%d %H:%M:%S" (e.g. "2013/July/01 01:02:03") and "%Y/%B/%d" (e.g. "2013/July/01")
- "%Y/%m/%d %H:%M:%S" (e.g. "2013/07/01 01:02:03") and "%Y/%m/%d" (e.g. "2013/07/01")

Value

A time as returned by `as.POSIXct`.

Author(s)

Dan Kelley

Examples

```
decodeTime("July 1 2013 01:02:03")
decodeTime("Jul 1 2013 01:02:03")
decodeTime("1 July 2013 01:02:03")
decodeTime("1 Jul 2013 01:02:03")
decodeTime("2013-07-01 01:02:03")
decodeTime("2013/07/01 01:02:03")
decodeTime("2013/07/01")
```

---

despike *Remove spikes from a time series*

Description

Remove spikes from a time series

Usage

```
despike(x, reference=c("median","smooth", "trim"), n=4, k=7, min, max,
        replace=c("reference", "NA"))
```

Arguments

x	a vector of values, interpreted as a time series
reference	indication of the type of reference time series to be used in the detection of spikes; see 'Details'.
n	an indication of the limit to differences between x and the reference time series, used for reference="median" or reference="smooth"; see 'Details.'
k	length of running median used with reference="median", and ignored for other values of reference.
min	minimum non-spike value of x, used with reference="trim".

max	maximum non-spike value of x, used with reference="trim".
replace	an indication of what to do with spike values, with "reference" indicating to replace them with the reference time series, and "NA" indicating to replace them with NA.

Details

The method identifies spikes with respect to a "reference" time-series, and replaces these spikes with the reference value, or with NA according to the value of action.

For reference="median", the first step is to linearly interpolate across any gaps, in which x==NA. Then the reference time series is constructed using runmed as a running median of k elements. Then, the standard deviation of the difference between x and the reference is calculated. Any x values that differ from the reference by more than n times this standard deviation are considered to be spikes. If replace="reference", these x values are replaced with the reference series, and the resultant time series is returned. If replace="NA", the spikes are replaced with NA in the returned time series.

For reference="smooth", the processing is the same as for "median", except that smooth is used to calculate the reference time series.

For reference="trim", the reference time series is constructed by linear interpolation across any regions in which x<min or x>max. In this case, the value of n is ignored, and the return value either uses the reference time series for spikes, or NA, according to the value of replace.

Value

A new vector in which spikes are replaced as described above.

Author(s)

Dan Kelley

Examples

```
n <- 50
x <- 1:n
y <- rnorm(n=n)
y[n/2] <- 10                    # 10 standard deviations
plot(x, y, type='l')
lines(x, despike(y), col='red')
lines(x, despike(y, reference="smooth"), col='darkgreen')
lines(x, despike(y, reference="trim", min=-3, max=3), col='blue')
legend("topright", lwd=1, col=c("black", "red", "darkgreen", "blue"),
       legend=c("raw", "median", "smooth", "trim"))
```

detrend *Detrend a set of observations*

Description

Detrend a set of observations, which can be useful in operations that are thrown off by endpoints, e.g. digital filtering.

Usage

```
detrend(x, y)
```

Arguments

x	a vector of numerical values. If y is not given, then x is taken for y.
y	an optional vector

Details

Detrends y by subtracting a linear trend in x, to create Y that has Y[1]=0 and Y[length(Y)]=0. If y is not given, then y is taken from x, and x is set to the series of integers from 1 to length{x}.

Value

A list containing Y, the detrended version of y, and the intercept a and slope b of the linear function of x that is subtracted from y to yield Y.

Author(s)

Dan Kelley

Examples

```
x <- seq(0, 0.9 * pi, length.out=50)
y <- sin(x)
plot(x, y)
d <- detrend(x, y)
points(x, d$Y, pch=20)
abline(h=0, lty='dotted')
abline(d$a, d$b, col='red')
points(x, d$Y + d$a + d$b * x, col='blue', pch='+')
```

drawDirectionField *Draw a direction field*

Description

Draw a direction field

Usage

```
drawDirectionField(x, y, u, v, scalex, scaley, add=FALSE, type=1,
debug=getOption("oceDebug"), ...)
```

Arguments

x,y	coordinates at which velocities are specified
u,v	velocity components in the x and y directions
scalex,scaley	scale to be used for the velocity arrows. Exactly one of these must be specified. Arrows that have u^2+v^2=1 will have length scalex along the x axis, or scaley along the y axis, according to which argument is given.
add	if TRUE, the arrows are added to an existing plot; otherwise, a new plot is started, with asp set to 1.
type	type of the arrow-like indication of the direction.
debug	debugging value; set to a positive integer to get debugging information.
...	extra graphical parameters, supplied to functions called by drawDirectionField. Try adjusting cex, pch, and lwd, and col.

Details

The direction field is indicated variously, depending on the value of type.

For type=1, each indicator is drawn with a symbol, according to the value of pch (either supplied globally, or as an element of the ... list) and of size cex, and colour col. Then, a line segment is drawn for each, and for this lwd and col may be set globally or in the ... list.

For type=2, the points are not drawn, but arrows are drawn instead of the line segments. Again, lwd and col control the type of the line.

Value

None.

Author(s)

Dan Kelley

Examples

```
library(oce)
plot(c(-1.5, 1.5), c(-1.5, 1.5), xlab="", ylab="", type='n')
drawDirectionField(x=rep(0, 2), y=rep(0, 2), u=c(1,1), v=c(1, -1), scalex=0.5, add=TRUE)
plot(c(-1.5, 1.5), c(-1.5, 1.5), xlab="", ylab="", type='n')
drawDirectionField(x=rep(0, 2), y=rep(0, 2), u=c(1,1), v=c(1, -1), scalex=0.5, add=TRUE,
                   type=2)
plot(c(-1.5, 1.5), c(-1.5, 1.5), xlab="", ylab="", type='n')
drawDirectionField(x=rep(0, 2), y=rep(0, 2), u=c(1,1), v=c(1, -1), scalex=0.5, add=TRUE,
                   type=2, length=0.1)
```

`drawIsopycnals` *Add isopycnal curves to TS plot*

Description

Draw isopycnal curves on an existing temperature-salinity plot

Usage

```
drawIsopycnals(nlevels=6, levels, rotate=TRUE, rho1000=FALSE, digits=2,
               eos=getOption("eos", default="unesco"),
            cex=0.75*par('cex'), col="darkgray", lwd=par("lwd"), lty=par("lty"))
```

Arguments

`nlevels`	suggested number of density levels (i.e. isopycnal curves); ignored if `levels` is supplied.
`levels`	optional density levels to draw.
`rotate`	boolean, set to `TRUE` to write all density labels horizontally.
`rho1000`	boolean, set to `TRUE` to write isopycnal labels as e.g. 1024 instead of 24.
`digits`	number of decimal digits to use in label (supplied to `round`).
`eos`	name of equation of state to be used, either `"unesco"` or `"teos"`. If the latter, then the computer must have the TEOS library installed; see `teos`.
`cex`	size for labels.
`col`	colour for lines and labels.
`lwd`	line width for isopcynal curves
`lty`	line type for isopcynal curves

Details

Adds isopycnal lines to an existing temperature-salinity plot. This is called by `plotTS`, and may be called by the user also, e.g. if an image plot is used to show TS data density.

Value

None.

Author(s)

Dan Kelley

See Also

plotTS, which calls this.

---

drawPalette *Draw palette on RHS of plot device*

Description

Draw a palette on the right-hand side of plot device

Usage

```
drawPalette(zlim, zlab="",
    breaks, col, colormap,
    mai, cex.axis=par("cex.axis"), pos=4,
    labels=NULL, at=NULL,
    levels, drawContours=FALSE,
    plot=TRUE, fullpage=FALSE, drawTriangles=FALSE,
    axisPalette, tformat,
    debug=getOption("oceDebug"), ...)
```

Arguments

zlim	two-element vector containing the lower and upper limits of z. This may also be a vector of any length exceeding 1, in which case its range is used.
zlab	label for the palette scale.
breaks	the z values for breaks in the colour scheme.
col	either a vector of colours corresponding to the breaks, of length 1 plus the number of breaks, or a function specifying colours, e.g. oceColorsJet for a rainbow.
colormap	a color map as created by colormap. If provided, this takes precedence over breaks and col.
mai	margins for palette, as defined in the usual way; see par. If not given, reasonable values are inferred from the existence of a non-blank zlab.
cex.axis	character-expansion value for text labels

pos	an integer indicating the location of the palette within the plotting area, 1 for near the bottom, 2 for near the left-hand side, 3 for near the top side, and 4 (the default) for near the right-hand side.
labels	optional vector of labels for ticks on palette axis (must correspond with at)
at	optional vector of positions for the labels
levels	optional contour levels, in preference to breaks values, to be added to the image if drawContours is TRUE.
drawContours	logical value indicating whether to draw contours on the palette, at the colour breaks.
plot	logical value indicating whether to plot the palette, the default, or whether to just alter the margins to make space for where the palette would have gone. The latter case may be useful in lining up plots, as in example 1 of "Examples".
fullpage	logical value indicating whether to draw the palette filling the whole plot width (apart from mai, of course). This can be helpful if the palette panel is to be created with layout, as illustrated in the "Examples".
drawTriangles	logical value indicating whether to draw triangles on the top and bottom of the palette. If a single value is provide, it applies to both ends of the palette. If a pair is provided, the first refers to the lower range of the palette, and the second to the upper range.
axisPalette	optional replacement function for axis(), e.g. for exponential notation on large or small values.
tformat	optional format for axis labels, if the variable is a time type (ignored otherwise).
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions.

Details

In the normal use, drawPalette draws an image palette near the right-hand side of the plotting device, and then adjusts the global margin settings in such a way as to cause the next plot to appear (with much larger width) to the left of the palette. The function can also be used, if zlim is not provided, to adjust the margin without drawing anything; this is useful in lining up the x axes of a stack of plots, some some of which will have palettes and others not.

The plot positioning is done entirely with margins, not with par(mfrow) or other R schemes for multi-panel plots. This means that the user is free to use those schemes without worrying about nesting or conflicts.

Value

None.

Use with multi-panel plots

An important consequence of the margin adjustment is that multi-panel plots require that the initial margin be stored prior to the first call to drawPalette, and reset after each palette-plot pair. This method is illustrated in "Examples".

Author(s)

Dan Kelley, with help from Clark Richards

See Also

This is used by imagep.

Examples

```
library(oce)
par(mgp=getOption("oceMgp"))

## 1. A three-panel plot
par(mfrow=c(3,1), mar=c(3, 3, 1, 1))
omar <- par('mar')                # save initial margin

## 1a. top panel: simple case
drawPalette(zlim=c(0,1), col=oceColorsJet(10))
plot(1:10, 1:10, col=oceColorsJet(10)[1:10],pch=20,cex=3,xlab='x', ylab='y')
par(mar=omar)                     # reset margin

## 1b. middle panel: colormap
cm <- colormap(name="gmt_globe")
drawPalette(colormap=cm)
icol <- seq_along(cm$col)
plot(icol, cm$breaks[icol], pch=20, cex=2, col=cm$col,
     xlab="Palette index", ylab="Palette breaks")
par(mar=omar)                     # reset margin

## 1c. bottom panel: space for palette (to line up graphs)
drawPalette(plot=FALSE)
plot(1:10, 1:10, col=oceColorsJet(10)[1:10],pch=20,cex=3,xlab='x', ylab='y')
par(mar=omar)                     # reset margin

# 2. Use layout to mimic the action of imagep(), with the width
# of the palette region being 14 percent of figure width.
d <- 0.14
layout(matrix(1:2,nrow=1), widths=c(1-d,d))
image(volcano, col=oceColorsJet(100), zlim=c(90, 200))
contour(volcano, add=TRUE)
drawPalette(c(90, 200), fullpage=TRUE, col=oceColorsJet)
```

---

drifter *ARGO drifter dataset*

Description

This is an ARGO drifter data object, for drifter 6900388, downloaded in July 2011. These data were collected and made freely available by the International Argo Program and the national programs that contribute to it. Salinity, temperature and pressure are each matrices with first index corresponding to depth, e.g. `drifter[["temperature"]][1,]` is the surface temperature.

Usage

```
data(drifter)
```

Author(s)

Dan Kelley

Source

http://www.usgodae.org/ftp/outgoing/argo/dac/bodc/6900388/6900388_prof.nc

See Also

See `drifter-class` in the Oce package for notes on the contents of `drifter` objects, and for functions to work with them.

Examples

```
## Not run:
library(oce)
data(drifter)
summary(drifter)
data(coastlineWorld)
plot(drifter, which="trajectory", coastline=coastlineWorld)

## End(Not run)
```

---

drifter-class *Class for drifter data*

Description

Class to store drifter data, with standard slots `metadata`, `data` and `processingLog`.

Methods

Data may be accessed as e.g. `pt[["time"]]`, where the string could also be `"longitude"` or `"latitude"`. The profile data are available in `"salinity"`, `"temperature"` and `"pressure"`, each of which is a matrix with first index corresponding to depth and second index corresponding to profile number. Assignment to these can be made with e.g. `drifter[["latitude"]] <- value`, etc. Indeed, any quantity in the `metadata` slot or the `data` slot can be retrieved or updated in this way.

Author(s)

Dan Kelley

See Also

A drifter object may be read with read.drifter or created with as.drifter. Plots can be made with plot.drifter, while summary.drifter produces statistical summaries and show produces overviews.

---

echosounder *echosounder dataset*

Description

This is degraded subsample of measurements that were made with a Biosonics scientific echousounder, as part of the St Lawrence Internal Wave Experiment (SLEIWEX).

Usage

```
data(echosounder)
```

Author(s)

Dan Kelley

Source

This file came from the SLEIWEX-2008 experiment.

See Also

The documentation for echosounder-class in the Oce package explains the structure of echosounder objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(echosounder)
plot(echosounder)
plot(echosounder, drawBottom=TRUE)
plot(echosounder, which=2, drawBottom=TRUE, atTop=pretty(echosounder[["time"]]))

## End(Not run)
```

`echosounder-class` *Class to store echosounder data*

Description

Class to store echosounder data, with standard slots `metadata`, `data` and `processingLog`. The `data` slot is a list containing

- An infrequently updated record of the intrument position, in `timeSlow`, `longitudeSlow` and `latitudeSlow`. These are used in plotting maps with `plot.echosounder`.
- An interpolated record of the instrument position, in `time`, `longitude`, and `latitude`. Linear interpolation is used to infer the longitude and latitude from the variables listed above.
- `depth`, vector of depths of echo samples (measured positive downwards in the water column). This is calculated from the inter-sample time interval and the sound speed provided as the `soundSpeed` argument to `read.echosounder`, so altering the value of the latter will alter the echosounder plots provided by `plot.echosounder`.
- The echosounder signal amplitude `a`, a matrix whose number of rows matches the length of `time`, etc., and number of columns equal to the length of `depth`. Thus, for example, `a[100,]` represents the depth-dependent amplitude at the time of the 100th ping.
- A matrix named `b` exists for dual-beam and split-beam cases. For dual-beam data, this is the wide-beam data, whereas `a` is the narrow-beam data. For split-beam data, this is the x-angle data.
- A matrix named `c` exists for split-beam data, containing the y-angle data.
- In addition to these matrices, ad-hoc calculated matrices named `Sv` and `TS` may be accessed as explained in the next section.

Methods

Accessing values. Data may be accessed as e.g. `echosounder[["time"]]`, where the string could also be e.g. FIXME (add more). The names of the columns are displayed with `show()`. The name of the source file is found with `"filename"`.

Derived data are also available: `"distance"` calls `geodDist` to compute locations, and calculates distance along the ship track, `"Sv"` returns a matrix of backscatter strength in DB, and `"TS"` returns a matrix of target strength in dB.

Assigning values. Everything that may be accessed may also be assigned, e.g. `echosounder[["time"]] <- 3600 + echosounder[["time"]]` adds an hour to time.

Overview of contents. The `show` method (e.g. `show(echosounder)`) displays information about the object.

Author(s)

Dan Kelley

See Also

A file containing echosounder data may be read with `read.echosounder`, and a echosounder object can also be created with `as.echosounder`.

Statistical summaries are provided by `summary.echosounder`, while `show` displays an overview. The `findBottom` function infers the ocean bottom from tracing the strongest reflector from ping to ping.

Echosounder objects may be plotted with `plot.echosounder`.

The contents of `echosounder` objects may be altered with `subset.echosounder`, or with the the `[[]]` scheme discussed in the previous section; skilled users may also manipulate the contents directly, but this is not recommended because it is brittle to changes in the data structure.

`eclipticalToEquatorial` *Convert ecliptical to equatorial coordinate*

Description

Convert from ecliptical to equatorial coordinates

Usage

```
eclipticalToEquatorial(lambda, beta, epsilon)
```

Arguments

`lambda`	longitude, in degrees, or a data frame containing `lambda`, `beta`, and `epsilon`, in which case the next to arguments are ignored.
`beta`	geocentric latitude, in degrees
`epsilon`	obliquity of the ecliptic, in degrees

Details

The method is taken from equations 8.3 and 8.4 of [1], or, equivalently, from equations 12.3 and 12.4 of [2].

Value

A data frame containing columns `rightAscension` and `declination` both in degrees.

Author(s)

Dan Kelley, based on formulae in [1] and [2].

References

1. Meeus, Jean, 1982. Astronomical formuae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages.

2. Meeus, Jean, 1991. Astronomical algorithms. Willmann-Bell, Richmond VA, USA. 429 pages.

`enuToOtherAdp` *Convert east-north-up to other coordinate*

Description

Convert ADP velocity components from an enu-based coordinate system to another system, perhaps to align axes with the coastline.

Usage

```
enuToOtherAdp(x, heading=0, pitch=0, roll=0)
```

Arguments

`x`	an object of class `"adp"`.
`heading`	number or vector of numbers, giving the angle, in degrees, to be added to the heading. See "Details".
`pitch`	as `heading` but for pitch.
`roll`	as `heading` but for roll.

Details

The supplied angles specify rotations to be made around the axes for which heading, pitch, and roll are defined. For example, an eastward current will point southeast if `heading=45` is used.

The returned value has heading, pitch, and roll matching those of `x`, so these angles retain their meaning as the instrument orientation.

NOTE: this function works similarly to `xyzToEnuAdp`, except that in the present function, it makes no difference whether the instrument points up or down, etc.

Value

An object with `data$v[,1:3,]` altered appropriately, and `metadata$oce.coordinate` changed from `enu` to `other`.

Author(s)

Dan Kelley

References

RD Instruments, 1998. *ADP Coordinate Transformation, formulas and calculations.* P/N 951-6079-00 (July 1998)

See Also

See `read.adp` for other functions that relate to objects of class `"adp"`.

Examples

```
library(oce)
data(adp)
o <- enuToOtherAdp(adp, heading=-31.5)
plot(o, which=1:3)
```

`enuToOtherAdv` *Convert east-north-up to other coordinate*

Description

Convert ADV velocity components from an enu-based coordinate system to another system, perhaps to align axes with the coastline.

Usage

```
enuToOtherAdv(x, heading=0, pitch=0, roll=0, debug=getOption("oceDebug"))
```

Arguments

`x`	an object of class `"adv"`.
`heading`	number or vector of numbers, giving the angle, in degrees, to be added to the heading. See "Details".
`pitch`	as `heading` but for pitch.
`roll`	as `heading` but for roll.
`debug`	a flag that turns on debugging. Set to 1 to get a debugging information.

Details

The supplied angles specify rotations to be made around the axes for which heading, pitch, and roll are defined. For example, an eastward current will point southeast if `heading=45` is used.

The returned value has heading, pitch, and roll matching those of `x`, so these angles retain their meaning as the instrument orientation.

NOTE: this function works similarly to `xyzToEnuAdv`, except that in the present function, it makes no difference whether the instrument points up or down, etc.

Author(s)

Dan Kelley

See Also

See `read.adv` for other functions that relate to objects of class `"adv"`

`equatorialToLocalHorizontal` *Convert equatorial to local horizontal coordinate*

Description

Convert from equatorial coordinates to local horizontal coordinates, i.e. azimuth and altitude.

Usage

```
equatorialToLocalHorizontal(rightAscension, declination, t, longitude, latitude)
```

Arguments

`rightAscension`	right ascension, e.g. calculated with `eclipticalToEquatorial`.
`declination`	declination, e.g. calculated with `eclipticalToEquatorial`.
`t`	time of observation.
`longitude`	longitude of observation, positive in eastern hemisphere.
`latitude`	latitude of observation, positive in northern hemisphere.

Details

The method is taken from equations 8.5 and 8.6 of [1], or, equivalently, from equations 12.5 and 12.6 of [2].

Value

A data frame containing columns `altitude` (angle above horizon, in degrees) and `azimuth` (angle anticlockwise from south, in degrees).

Author(s)

Dan Kelley, based on formulae in [1] and [2].

References

1. Meeus, Jean, 1982. Astronomical formuae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages.

2. Meeus, Jean, 1991. Astronomical algorithms. Willmann-Bell, Richmond VA, USA. 429 pages.

errorbars *Draw error bars on an existing xy diagram*

Description

Draw error bars on an existing xy diagram

Usage

```
errorbars(x, y, xe, ye, percent=FALSE, style=0, length=0.025, ...)
```

Arguments

x	x coordinates of points on the existing plot.
y	y coordinates of points on the existing plot.
xe	error on x coordinates of points on the existing plot, either a single number or a vector of length identical to that of y.
ye	as xe but for y coordinate.
percent	boolean flag indicating whether xe and ye are in terms of percent of the corresponding x and y values.
style	indication of the style of error bar. Using style=0 yields simple line segments (drawn with segments) and style=1 yields line segments with short perpendicular endcaps.
length	length of endcaps, for style=1 only; it is passed to arrows, which is used to draw that style of error bars.
...	graphical parameters passed to the code that produces the error bars, e.g. to segments for style=0.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
S <- ctd[["salinity"]]
T <- ctd[["temperature"]]
plot(S, T)
errorbars(S, T, 0.05, 0.5)
```

`extract` *Extract data from an oce object*

Description

Extract data from an oce object

Usage

```
extract(x, names)
```

Arguments

`x`	an oce object.
`names`	a vector of character values, indicating names of items to be extracted.

Details

This is a convenience function that extracts data from an oce object, providing the user with isolation from the internal storage scheme used in oce objects.

Value

A list of the values found.

Author(s)

Dan Kelley

Examples

```
library(oce)
## ctd cast
data(ctd)
TS <- extract(ctd, c("temperature", "salinity"))
print(summary(lm(temperature~salinity, data=TS)))

## section
data(section)
TSlon <- extract(section, c("temperature","salinity","longitude"))
med <- factor(-20 > TSlon$longitude, labels=c("med","other"))
plotTS(TSlon, col=c("black","gray")[med])
```

`fillGap` *Fill a gap in an oce object*

Description

Fill a gap in an oce object

Usage

```
fillGap(x, method=c("linear"), rule=1)
```

Arguments

`x`	an `oce` object.
`method`	to use; see "Details".
`rule`	integer controlling behaviour at start and end of `x`. If `rule=1`, `NA` values at the ends are left in the return value. If `rule=2`, they are replaced with the nearest non-NA point.

Details

Sequences of `NA` values, are filled by linear interpolation between the non-`NA` values that bound the gap.

Value

A new `oc` object, with gaps removed.

Bugs

1. Eventually, this will be expanded to work with any `oce` object. But, for now, it only works for vectors that can be coerced to numeric.
2. If the first or last point is `NA`, then `x` is returned unaltered.
3. Only method `linear` works now, and that argument is ignored at present.

Author(s)

Dan Kelley

Examples

```
library(oce)
# Integers
x <- c(1:2, NA, NA, 5:6)
y <- fillGap(x)
print(data.frame(x,y))
# Floats
```

```
x <- x + 0.1
y <- fillGap(x)
print(data.frame(x,y))
```

`findBottom` *Find the ocean bottom in an echosounder object*

Description

Find the ocean bottom in an echosounder object

Usage

```
findBottom(x, ignore=5, clean=despike)
```

Arguments

`x`	an object of class `echosounder`
`ignore`	number of metres of data to ignore, near the surface
`clean`	a function to clean the inferred depth of spikes

Details

Finds the depth in a Biosonics echosounder file, by finding the strongest reflector and smoothing its trace.

Value

A list with elements: the `time` of a ping, the `depth` of the inferred depth in metres, and the `index` of the inferred bottom location, referenced to the object's `depth` vector.

Author(s)

Dan Kelley

See Also

The documentation for `echosounder-class` explains the structure of `echosounder` objects, and also outlines the other functions dealing with them.

findInOrdered *Find indices of tiems in an ordered vector*

Description

Find indices of tiems in an ordered vector

Usage

```
findInOrdered(x, f)
```

Arguments

x	a numeric vector, in increasing order by value.
f	a numeric vector of items whose indices are sought.

Details

The indices point to the largest items in `x` that are less than or equal the values in `f`. The method uses a bisection search, so the time taken is proportional to `length(f) * log2(length(x))`.

Value

A numerical vector indicating the indices of left-sided neighbors.

Author(s)

Dan Kelley

Examples

```
findInOrdered(seq(0,10,1), c(1.2,7.3))
```

formatCI *Confidence interval in parenthetic notation*

Description

Format a confidence interval in parenthetic notation

Usage

```
formatCI(ci, style=c("+/-", "parentheses"), model, digits=NULL)
```

Arguments

ci	optional vector of length 2 or 3.
style	string indicating notation to be used.
model	optional regression model, e.g. returned by lm or nls.
digits	optional number of digits to use; if not supplied, getOption("digits") is used.

Details

If a model is given, then ci is ignored, and a confidence interval is calculated using confint with level set to 0.6914619. This level corresponds to a range of plus or minus one standard deviation, for the t distribution and a large number of degrees of freedom (since qt(0.6914619, 100000) is 0.5).

If model is missing, ci must be provided. If it contains 3 elements, then first and third elements are taken as the range of the confidence interval (which by convention should use the level stated in the previous paragraph), and the second element is taken as the central value. Alternatively, if ci has 2 elements, they are taken to be bounds of the confidence interval and their mean is taken to be the central value.

In the +/- notation, e.g. $a \pm b$ means that the true value lies between $a - b$ and $a + b$ with a high degree of certainty. Mills et al. (1993, section 4.1 on page 83) suggest that b should be set equal to 2 times the standard uncertainty or standard deviation. JCGM (2008, section 7.2.2 on pages 25 and 26), however, suggest that b should be set to the standard uncertainty, while also recommending that the $\pm$ notation be avoided altogether.

The parentheses notation is often called the compact notation. In it, the digits in parenthese indicate the uncertainty in the corresponding digits to their left, e.g. 12.34(3) means that the last digit (4) has an uncertainty of 3. However, as with the $\pm$ notation, different authorities offer different advice on defining this uncertainty; Mills et al. (1993, section 4.1 on page 83) provide an example in which the parenthetic notation has the same value as the $\pm$ notation, while JCM (2008, section 7.2.2 on pages 25 and 26) suggest halving the number put in parentheses.

The foramtci function is based on the JCM (2008) notation, i.e. formatCI(ci=c(8,12), style="+/-") yields "10+-2", and formatCI(ci=c(8,12), style="parentheses") yields "10(2)".

Note: if the confidence range exceeds the value, the parentheses format reverts to +/- format.

Value

If ci is given, the result is a character string with the estimate and its uncertainty, in plus/minus or parenthetic notation. If model is given, the result is a 1-column matrix holding character strings, with row names corresponding to the parameters of the model.

Author(s)

Dan Kelley

References

JCGM, 2008. *Evaluation of measurement data - Guide to the expression of uncertainty in measurement (JCGM 100:2008)*, published by the Joint Committee for Guides in Metrology. [http:

//www.bipm.org/en/publications/guides/gum.html] (See section 7.2.2 for a summary of notation, which shows equal values to the right of a +- sign and in parentheses.)

I. Mills, T. Cvitas, K. Homann, N. Kallay, and K. Kuchitsu, 1993. *Quantities, Units and Symbols in Physical Chemistry*, published Blackwell Science for the International Union of Pure and Applied Chemistry. [http://old.iupac.org/publications/books/gbook/index.html] (See section 4.1, page 83, for a summary of notation, which shows that a value to the right of a +- sign is to be halved if put in parentheses.)

Examples

```
x <- seq(0, 1, length.out=300)
y <- rnorm(n=300, mean=10, sd=1) * x
m <- lm(y~x)
print(formatCI(model=m))
```

---

formatPosition *Geographical position in degrees and minutes*

Description

Format geographical positions to degrees, minutes, and hemispheres

Usage

```
formatPosition(latlon, isLat=TRUE, type=c("list", "string", "expression"), showHemi=TRUE)
```

Arguments

latlon	a vector of latitudes or longitudes
isLat	a boolean that indicates whether the quantity is latitude or longitude
type	a string indicating the type of return value (see below)
showHemi	a boolean that indicates whether to indicate the hemisphere

Details

This function is in an early stage of development.

Value

A list containing degrees, minutes, seconds, and hemispheres, or a vector of strings or (broken) a vector of expressions.

Author(s)

Dan Kelley

Examples

```
library(oce)
formatPosition(10+1:10/60+2.8/3600)
formatPosition(10+1:10/60+2.8/3600, type="string")
```

`fullFilename` *full name of file, including path*

Description

Return full name of file, including path

Usage

```
fullFilename(filename)
```

Arguments

`filename`	name of file

Details

Determines the full name of a file, including the path. Used by many `read.X` routines, where `X` is the name of a class of object.

Value

Full file name

Author(s)

Dan Kelley

`geodDist` *Geodesic distance on earth*

Description

Compute geodesic distance on surface of earth.

Usage

```
geodDist(lon1, lat1=NULL, lon2=NULL, lat2=NULL, alongPath=FALSE)
```

Arguments

lon1	longitude or a vector of longitudes, **or** a section object, from which longitude and latitude are extracted and used instead of the next three arguments
lat1	latitude or vector of latitudes (ignored if lon1 is a section object)
lon2	optional longitude or vector of longitudes (ignored if alongPath=TRUE
lat2	optional latitude or vector of latitudes (ignored if alongPath=TRUE
alongPath	boolean indicating whether to compute distance along the path, as opposed to distance from the reference point. If alongPath=TRUE, any values provided for lat2 and lon2 will be ignored.

Details

This calculates distance between points on the earth, measured along the surface. The method involves the solution of the geodetic inverse problem, using T. Vincenty's modification of Rainsford's method with Helmert's elliptical terms.

The function may be used in several different ways.

Case 1: lon1 is a section object. The values of lat1, lon2, and lat2 arguments are ignored, and the behaviour depends on the value of the alongPath argument. If alongPath=FALSE, the return value contains the geodetic distances of each station from the first one. If alongPath=TRUE, the return value is the geodetic distance along the path connecting the stations, in the order in which they are stored in the section.

Case 2: lon1 is a vector. If lon2 and lat2 are not given, then the return value is a vector containing the distances of each point from the first one, *or* the distancce along the path connecting the points, according to the value of alongPath. On the other hand, if both lon2 and lat2 are specified, then the return result depends on the length of these arguments. If they are each of length 1, then they are taken as a reference point, from which the distances to lon1 and lat1 are calculated (ignoring the value of alongPath). However, if they are of the same length as lon1 and lat1, then the return value is the distance between corresponding (lon1,lat1) and (lon2,lat2) values.

Value

Vector of distances in kilometres.

Author(s)

Dan Kelley based this on R code sent to him by Darren Gillis, who in 2003 had modified Fortran code that, according to comments in the source, had been written in 1974 by L. Pfeifer and J. G. Gergen.

References

T. Vincenty, "Direct and Inverse Solutions of Ellipsoid on the Ellipsoid with Application of Nested Equations", *Survey Review*, April 1975. (As of early 2009, this document is available at http://www.ngs.noaa.gov/PUBS_LIB/inverse.pdf.)

See Also

geodXy

Examples

```
library(oce)
km <- geodDist(100, 45, 100, 46)
data(section)
geodDist(section)
geodDist(section, alongPath=TRUE)
```

---

geodGc *Great-circle segments between points on earth*

Description

Find great-circle segments between two points on earth

Usage

```
geodGc(longitude, latitude, dmax)
```

Arguments

longitude	vector of longitudes, in degrees east
latitude	vector of latitudes, in degrees north
dmax	maximum angular separation to tolerate between sub-segments, in degrees.

Details

Each pair in the longitude and latitude vectors is considered in turn. For long vectors, this may be slow.

Value

Data frame of longitude and latitude.

Author(s)

Dan Kelley, based on code from Clark Richards, in turn based on formulae provided by Ed Williams.

Source

http://williams.best.vwh.net/avform.htm#Intermediate

See Also

geodDist and geodXy

Examples

```
library(oce)
data(coastlineWorld)
mapPlot(coastlineWorld, type='l',
longitudelim=c(-80,10), latitudelim=c(35,80),
projection="orthographic", orientation=c(35, -35, 0))
## Great circle from New York to Paris (Lindberg's flight)
l <- geodGc(c(-73.94,2.35), c(40.67,48.86), 1)
mapLines(l$longitude, l$latitude, col='red', lwd=2)
```

---

geodXy *Convert lon/lat to x/y on earth*

Description

Convert lon/lat to x/y on earth

Usage

```
geodXy(lon, lat, lon.ref, lat.ref, rotate=0)
```

Arguments

lon	vector of longitudes
lat	vector of latitudes
lon.ref	numeric, reference longitude
lat.ref	numeric, reference latitude
rotate	numeric, counterclockwise angle, in degrees, by which to rotate the (x, y) coordinates about the reference point. This is useful in rotating the coordinate system to align with a coastline, a mean current, etc.

Details

This is based on the same geodesic calculations used in geodDist, the documentation of which explains the method and provides literature citations.

Value

Data frame of x and y, measured in metres.

Author(s)

Dan Kelley

See Also

geodDist

Examples

```
library(oce)
lat <- c(0, 1/60,    0, 1/60)
lon <- c(0,    0, 1/60, 1/60)
plot(geodXy(lat, lon, 0, 0, 0) / 1852)
points(geodXy(lat, lon, 0, 0, 1) / 1852, col='red')
```

GMTOffsetFromTz *Determine time offset from timezone*

Description

Determine time offset from timezone

Usage

```
GMTOffsetFromTz(tz)
```

Arguments

tz	a timezone, e.g. UTC.

Details

The data are from http://www.timeanddate.com/library/abbreviations/timezones/ and were hand-edited to develop this code, so there may be errors. Also, note that some of these contradict; if you examine the code, you'll see some commented-out portions that represent solving conflicting definitions by choosing the more common timezone abbreviation over a the less common one.

Value

Number of hours in offset, e.g. AST yields 4.

Author(s)

Dan Kelley

Examples

```
library(oce)
cat("Atlantic Standard Time is ", GMTOffsetFromTz("AST"), "hours after UTC")
```

gps-class *Class to store gps data*

Description

Class to store gps data, with standard slots `metadata` (`filename`), `data` (a list containing `longitude` and `latitude`) and `processingLog`.

Methods

Extracting values: Data may be accessed as e.g. `gps[["longitude"]]` or `gps[["latitude"]]`.

Assigning values: Latitude may be changed with e.g. `gps[["longitude"]] <- value`, and of course the same can be done for `latitude`.

Overview of contents: The `show` method (e.g. `show(gps)`) displays information about the object.

Author(s)

Dan Kelley

See Also

Use `as.gps` to convert data to this form, `read.gps` to read GPX-format data, and `plot.gps` to plot data.

grad *Gradient of a matrix*

Description

Calculate the grad of a matrix by first differences

Usage

```
grad(h, x, y)
```

Arguments

h	a matrix
x	x values
y	y values

Details

In the interior of the matrix, centred second-order differences are used to infer the components of the grad. Along the edges, first-order differences are used.

Value

A list containing gx and gy, matrices of the same dimension as h.

Author(s)

Dan Kelley, based on advice of Clark Richards, and mimicking a matlab function.

Examples

```
## Geostrophic flow around an eddy
library(oce)
dx <- 5e3
dy <- 10e3
x <- seq(-200e3, 200e3, dx)
y <- seq(-200e3, 200e3, dy)
R <- 100e3
h <- outer(x, y, function(x, y) 500*exp(-(x^2+y^2)/R^2))
grad <- grad(h, x, y)
par(mfrow=c(2,2), mar=c(3, 3, 1, 1), mgp=c(2, 0.7, 0))
contour(x,y,h,asp=1, main=expression(h))
f <- 1e-4
gprime <- 9.8 * 1 / 1024
u <- -(gprime / f) * grad$gy
v <-  (gprime / f) * grad$gx
contour(x, y, u, asp=1, main=expression(u))
contour(x, y, v, asp=1, main=expression(v))
contour(x, y, sqrt(u^2+v^2), asp=1, main=expression(speed))
```

---

gravity *Acceleration due to earth gravity*

Description

Compute g, the acceleration due to gravity, as a function of latitude.

Usage

```
gravity(latitude=45, degrees=TRUE)
```

Arguments

latitude	Latitude in °N or radians north of the equator.
degrees	Flag indicating whether degrees are used for latitude; if set to FALSE, radians are used.

Details

Value not verified yet, except roughly.

Value

Acceleration due to gravity [m^2/s].

Author(s)

Dan Kelley

References

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

Caution: Fofonoff and Millard (1983 UNESCO) use a different formula.

Examples

```
g <- gravity(45) # 9.8
```

---

head *Ends of* oce *objects.*

Description

Returns the first or last parts of an oce object.

Usage

```
## S3 method for class 'adp'
head(x, n = 6L, ...)
## S3 method for class 'adp'
tail(x, n = 6L, ...)
```

Arguments

x	an object
n	a single integer. If positive, size for the resulting object: number of elements for a vector (including lists), rows for a matrix or data frame or lines for a function. If negative, all but the n last/first number of elements of x.
...	arguments to be passed to or from other methods.

Details

For adp objects, returns first or last profiles, as indicated.

Value

An object like x but generally smaller.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(adp)
start <- head(adp)
plot(start)
```

---

header *Return the header for an Oce data object*

Description

Return the header for an oce data object.

Usage

```
header(x)
```

Arguments

x	an oce object.

Details

Returns header, taken from the metadata of x.

Value

The header, often as a list of character strings.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
header(ctd)
```

imagep *Plot an image with a color palette*

Description

Plot an image with a color palette

Usage

```
imagep(x, y, z,
    xlim, ylim, zlim,
    zclip=FALSE,
    flipy=FALSE,
    xlab="", ylab="", zlab="", zlabPosition=c("top", "side"),
    breaks, col, colormap,
    labels=NULL, at=NULL,
    drawContours=FALSE,
    drawPalette=TRUE,
    drawTriangles=FALSE,
    tformat,
    drawTimeRange=getOption("oceDrawTimeRange"),
    filledContour=FALSE,
    missingColor=NULL,
    mgp=getOption("oceMgp"),
    mar, mai.palette,
    xaxs="i", yaxs="i",
    cex=par("cex"),
    adorn,
    axes=TRUE,
    main="",
    axisPalette,
    debug=getOption("oceDebug"),
    ...)
```

Arguments

x,y	optional locations of grid lines along which values of z are measured. The values must be finite and distinct. If they are out of order, they will be ordered (and the z data will be ordered in a matching way). If x and y are not provided, they are constructed to indicate the indices of the matrix, in contrast to the range of 0 to 1, as is the case for image. If x is a list, its components x$x and x$y are used for x and y, respectively. If the list has component z this is used for z. (NOTE: these arguments are meant to mimic those of image, which explains the same description here.) There are also some special cases, e.g. if x is a topographic object such as can be created with read.topo or as.topo, then longitude and latitude are used for axes, and topographic height is drawn.

z	a matrix containing the values to be plotted (NAs are allowed). Note that x can be used instead of z for convenience. (NOTE: these arguments are meant to mimic those of image, which explains the same description here.)
xlim	limits on x axis.
ylim	limits on y axis.
zlim	either a pair of numbers giving the limits for the colorscale, or "histogram" to have a flattened histogram (i.e. to maximally increase contrast throughout the domain.)
zclip	logical, indicating whether to clip the colors to those corresponding to zlim, if the latter is provided. Clipped regions will be colored with missingColor.
flipy	logical, with TRUE indicating that the image should be flipped top to bottom (e.g. to produce a profile image for a downward-looking acoustic-doppler profile).
xlab,ylab,zlab	names for x axis, y axis, and the image values.
zlabPosition	string indicating where to put the label for the z axis, either at the top-right of the main image, or on the side, in the axis for the palette.
breaks	the z values for breaks in the color scheme. If this is of length 1, the value indicates the desired number of breaks, which is supplied to pretty, in determining clean break points.
col	either a vector of colors corresponding to the breaks, of length 1 plus the number of breaks, or a function specifying colors, e.g. oceColorsJet for a rainbow.
colormap	a color map as created by colormap. If provided, this takes precedence over breaks and col.
labels	optional vector of labels for ticks on palette axis (must correspond with at)
at	optional vector of positions for the labels
drawContours	logical value indicating whether to draw contours on the image, and palette, at the color breaks. Images with a great deal of high-wavenumber variation look poor with contours.
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
drawTimeRange	logical, only used if the x axis is a time. If TRUE, then an indication of the time range of the data (not the axis) is indicated at the top-left margin of the graph. This is useful because the labels on time axes only indicate hours if the range is less than a day, etc.
drawPalette	indication of the type of palette to draw, if any. If drawPalette=TRUE, a palette is drawn at the right-hand side of the main image. If drawPalette=FALSE, no palette is drawn, and the right-hand side of the plot has a thin margin. If drawPalette="space", then no palette is drawn, but space is put in the right-hand margin to occupy the region in which the palette would have been drawn. This last form is useful for producing stacked plots with uniform left and right margins, but with palettes on only some of the images.
drawTriangles	logical value indicating whether to draw triangles on the top and bottom of the palette. This is passed to drawPalette.
filledContour	boolean value indicating whether to use filled contours to plot the image.

missingColor	a color to be used to indicate missing data, or NULL to avoid making the indication.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar"). If not given, a reasonable value is calculated based on whether xlab and ylab are empty strings.
mai.palette	palette margin corrections (in inches), added to the mai value used for the palette. Use with care.
xaxs	character indicating whether image should extend to edge of x axis (with value "i") or not; see par("xaxs").
yaxs	as xaxs but for y axis.
cex	size of labels on axes and palette; see par("cex").
adorn	optional expression to be performed immediately after drawing the data panel.
axes	logical, set TRUE to get axes on the main image.
main	title for plot.
axisPalette	optional replacement function for axis(), passed to drawPalette.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions.

Details

Creates an image with a color palette to the right. The effect is similar to filled.contour except that with imagep it is possible to set the layout outside the function, which enables the creation of plots with many image-palette panels. Note that the contour lines may not coincide with the color transitions, in the case of coarse images.

Note that this does not use layout or any of the other screen splitting methods. It simply manipulates margins, and draws two plots together. This lets users employ their favourite layout schemes.

The palette is drawn before the image, so that further drawing can be done on the image if desired, if the user prefers not to use the adorn argument.

NOTE: imagep is an analogue of image, and so it borrows a somewhat odd convention: the number of rows in the matrix corresponds to the x axis, not the y axis. (Actually, image permits the length of x to match either nrow(z) or 1+nrow(z), but here only the first is permitted.)

Value

None.

Author(s)

Dan Kelley, with help from Clark Richards

See Also

This uses drawPalette.

Examples

```
library(oce)

## 1. simplest use
imagep(volcano)

## 2. something oceanographic (internal-wave speed)
h <- seq(0, 50, length.out=100)
drho <- seq(1, 3, length.out=200)
speed <- outer(h, drho, function(drho, h) sqrt(9.8 * drho * h / 1024))
imagep(h, drho, speed, xlab="Equivalent depth [m]",
ylab=expression(paste(Delta*rho, " [kg/m^3]")),
zlab="Internal-wave speed [m/s]")

## 3. fancy labelling on atan() function
x <- seq(0, 1, 0.01)
y <- seq(0, 1, 0.01)
angle <- outer(x,y,function(x,y) atan2(y,x))
imagep(x, y, angle, filledContour=TRUE, breaks=c(0, pi/4, pi/2),
       col=c("lightgray", "darkgray"),
       at=c(0, pi/4, pi/2),
       labels=c(0, expression(pi/4), expression(pi/2)))

## 4. a colormap case
data(topoWorld)
cm <- colormap(name="gmt_globe")
imagep(topoWorld, colormap=cm)
```

`integerToAscii` *Decode integer to corresponding ASCII code*

Description

Decode integer to corresponding ASCII code

Usage

```
integerToAscii(i)
```

Arguments

`i`	an integer, or integer vector.

Value

A character, or character vector.

Author(s)

Dan Kelley

Examples

```
library(oce)
A <- integerToAscii(65)
cat("A=", A, "\n")
```

`integrateTrapezoid` *Use trapezoidal integration*

Description

Estimate the integral of one-dimensional function using the trapezoidal rule.

Usage

```
integrateTrapezoid(x, y, type=c("A", "dA", "cA"))
```

Arguments

`x`	x values, or a single value that is taken as the (constant) difference between x values.
`y`	y values, with length (n, say) that must match that of `x`. If `y` is not given, the integration is done with the `x` values taken to be y, and dx taken to be 1.
`type`	Flag indicating the desired return value (see "Value").

Value

If `type="A"` (the default), a single value is returned, containing the estimate of the integral of `y=y(x)`. If `type="dA"`, a numeric vector of the same length as `x`, of which the first element is zer0, the second element is the integral between `x[1]` and `x[2]`, etc. If `type="cA"`, the result is the cumulative sum (as in `cumsum`) of the values that would be returned for `type="dA"`. See "Examples".

Bugs

There is no handling of `NA` values.

Author(s)

Dan Kelley

Examples

```
x <- seq(0, 1, length.out=10) # try larger length.out to see if area approaches 2
y <- 2*x + 3*x^2
A <- integrateTrapezoid(x, y)
dA <- integrateTrapezoid(x, y, "dA")
cA <- integrateTrapezoid(x, y, "cA")
print(A)
print(sum(dA))
print(tail(cA, 1))
print(integrateTrapezoid(diff(x[1:2]), y))
print(integrateTrapezoid(y))
```

---

interpBarnes *Grid data using Barnes algorithm*

Description

Grid data using Barnes algorithm.

Usage

```
interpBarnes(x, y, z, w,
    xg, yg, xgl, ygl,
    xr, yr, gamma=0.5, iterations=2, trim=0,
    debug=getOption("oceDebug"))
```

Arguments

x, y	a vector of x and ylocations.
z	a vector of z values, one at each (x,y) location.
w	a optional vector of weights at the (x,y) location. If not supplied, then a weight of 1 is used for each point, which means equal weighting. Higher weights give data points more influence.
xg, yg	optional vectors defining the x and y grids. If not supplied, these values are inferred from the data, using e.g. pretty(x, n=50).
xgl, ygl	optional lengths of the x and y grids, to be constructed with seq spanning the data range. These values xgl are only examined if xg and yg are not supplied.
xr,yr	optional values defining the width of the radius ellipse in the x and y directions. If not supplied, these are calculated as the span of x and y over the square root of the number of data.
gamma	grid-focussing parameter. At each iteration, xr and yr are reduced by a factor of sqrt(gamma).
iterations	number of iterations.

trim	a number between 0 and 1, indicating the quantile of data weight to be used as a criterion for blanking out the gridded value (using NA). If 0, the whole zg grid is returned. If >0, any spots on the grid where the data weight is less than the trim-th quantile are set to NA. See examples.
debug	a flag that turns on debugging. Set to 0 for no debugging information, to 1 for more, etc; the value is reduced by 1 for each descendent function call.

Details

The algorithm follows that described by Koch et al. (1983), with the addition of the ability to blank out the grid in spots where data are sparse, using the trim argument.

Value

A list containing: xg, a vector holding the x-grid); yg, a vector holding the y-grid; zg, a matrix holding the gridded values; wg, a matrix holding the weights used in the interpolation at its final iteration; and zd, a vector of the same length as x, which holds the interpolated values at the data points.

Author(s)

Dan Kelley

References

S. E. Koch and M. DesJardins and P. J. Kocin, 1983. "An interactive Barnes objective map anlaysis scheme for use with satellite and conventional data," *J. Climate Appl. Met.*, vol 22, p. 1487-1503.

See Also

See wind.

Examples

```
library(oce)

# 1. contouring example, with wind-speed data from Koch et al. (1983)
data(wind)
u <- interpBarnes(wind$x, wind$y, wind$z)
contour(u$xg, u$yg, u$zg, labcex=1)
text(wind$x, wind$y, wind$z, cex=0.7, col="blue")
title("Numbers are the data")

# 2. As 1, but blank out spots where data are sparse
u <- interpBarnes(wind$x, wind$y, wind$z, trim=0.1)
contour(u$xg, u$yg, u$zg, level=seq(0, 30, 1))
points(wind$x, wind$y, cex=1.5, pch=20, col="blue")

# 3. As 1, but interpolate back to points, and display the percent mismatch
u <- interpBarnes(wind$x, wind$y, wind$z)
contour(u$xg, u$yg, u$zg, labcex=1)
```

```
mismatch <- 100 * (wind$z - u$zd) / wind$z
text(wind$x, wind$y, round(mismatch), col="blue")
title("Numbers are percent mismatch between grid and data")


# 4. As 3, but contour the mismatch
mismatchGrid <- interpBarnes(wind$x, wind$y, mismatch)
contour(mismatchGrid$xg, mismatchGrid$yg, mismatchGrid$zg, labcex=1)

# 5. One-dimensional example, smoothing a salinity profile
data(ctd)
p <- pressure(ctd)
y <- rep(1, length(p)) # fake y data, with arbitrary value
S <- salinity(ctd)
pg <- pretty(p, n=100)
g <- interpBarnes(p, y, S, xg=pg, xr=1)
plot(S, p, cex=0.5, col="blue", ylim=rev(range(p)))
lines(g$zg, g$xg, col="red")
```

`is.beam` *Determine coordinate system*

Description

Determine coordinate system for acoustic-doppler device

Usage

```
is.beam(x)
is.xyz(x)
is.enu(x)
coordinate(x)
```

Arguments

`x`	an `oce` object that inherits from either `adp` or `adv`.

Details

These functions work by checking the value of `x@metadata$oce.coordinate`, the main purpose being to prevent users from having to know the difference between that item and others of similar names.

Value

The `is.` functions return `TRUE` if the object inherits from `adv` or `adp`, and is of the stated coordinate type, or `FALSE` otherwise. `coordinate` return a string indicating the coordinate system, `beam` (velocities oriented along acoustic beams), `xyz` (velocities in a cartesian coordinate system tied to the instrument, or its pressure case), or `enu` (a cartesian coordinate system with one component pointing east, the second north, and the third up.

Author(s)

Dan Kelley

See Also

`read.adp` and `read.adv` read such files; consult their documentation for information about other related functions.

Examples

```
data(adp)
print(is.beam(adp))
print(is.xyz(adp))
print(is.enu(adp))
print(coordinate(adp))
```

---

julianCenturyAnomaly *Julian-Day number to Julian century*

Description

Convert a Julian-Day number to a time in julian centuries since noon on January 1, 1900.

Usage

```
julianCenturyAnomaly(jd)
```

Arguments

jd	a julian day number, e.g. as given by `julianDay`.

Details

The method follows Meese (1982 equation 15.1). The example reproduces the example provided by Meeuse (1982 example 15.a), with fractional error 3e-8.

Value

Julian century since noon on January 1, 1900.

Author(s)

Dan Kelley

References

Meeus, Jean, 1982. Astronomical formuae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages

Examples

```
t <- ISOdatetime(1978, 11, 13, 4, 35, 0, tz="UTC")
jca <- julianCenturyAnomaly(julianDay(t))
cat(format(t), "is Julian Century anomaly", format(jca, digits=8), "\n")
```

`julianDay` *Convert a POSIXt time to a Julian day*

Description

Convert a POSIXt time to a Julian day

Usage

```
julianDay(t, year, month, day, hour, min, sec, tz="UTC")
```

Arguments

`t`	a time, in POSIXt format, e.g. as created by `as.POSIXct`, `as.POSIXlt`, or `numberAsPOSIXct`. If this is provided, the other arguments are ignored.
`year`	year, to be provided along with `month`, etc., if `t` is not provided.
`month`	month, numbered with January being 1.
`day`	day in month, starting at 1.
`hour`	hour of day.
`min`	minute of hour
`sec`	second of hour
`tz`	timezone

Details

The method is taken from Chapter 3 of Meeus (1982). It should be noted that Meeus and other astronomical treatments use fractional days, whereas the present code follows the R convention of specifying days in whole numbers, with hours, minutes, and seconds also provided as necessary. Conversion is simple, as illustrated in the example for 1977 April 26.4, for which Meeus calculates julian day 2443259.9. Note that the R documentation for `julian` suggests another formula, but the point of the present function is to match the other Meeus formulae, so that suggestion is ignored here.

Value

A Julian-Day number, in astronomical convention as explained in Meeus.

Author(s)

Dan Kelley

References

Meeus, Jean, 1982. Astronomical formuae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages

Examples

```
t <- ISOdatetime(1977, 4, 26, hour=0, min=0, sec=0, tz="ET")+0.4*86400
jd <- julianDay(t)
cat(format(t), "is Julian Day", format(jd, digits=14), "\n")
```

`landsat-class` *Class to store landsat data*

Description

Class to store landsat data, with standard slots `metadata`, `data` and `processingLog`.

Methods

Consider a landsat object named `landsat`.

Accessing data values. The data may be accessed with e.g. `landsat[["band", 8]]`, for band number 8, or e.g. `landsat[["time"]]` for the time at which the image was acquired.

Accessing metadata. Anything in the metadata can be accessed by name, e.g. `landsat[["time"]]`. Note that some items are simply copied over from the source data file and are not altered by e.g. decimation. An exception is the lat-lon box, which is altered by `landsatTrim`.

Overview of contents. The `summary` method displays information about the object.

Author(s)

Dan Kelley

References

1. http://glovis.usgs.gov

2. http://landsat.gsfc.nasa.gov/?page_id=5377

See Also

A file containing Landsat data may be read with `read.landsat` or `read.oce`, and one such file is provided by the `ocedata` package as a dataset named `landsat`.

Plots may be made with `plot.landsat`. Since plotting can be quite slow, decimation is available both in the plotting function and as the separate function `link{decimate}`. Images may be subsetted with `landsatTrim`.

Landsat data are available at several websites; see e.g. [1].

See [2] for information on the bands for Landsat 8, paying particular attention to the resolutions and wavelengths. Band 8 is panchromatic, and has the highest resolution, so this is the default used in `plot.landsat`.

`landsatTrim` *Trim a landsat image to a lat-lon box*

Description

Trim a landsat image to a lat-lon box.

Usage

```
landsatTrim(x, ll, ur, debug=getOption("oceDebug"))
```

Arguments

`x`	A `landsat` object, e.g. as read by `read.landsat`.
`ll`	A list containing `longitude` and `latitude`, for the lower-left corner of the portion of the image to retain.
`ur`	A list containing `longitude` and `latitude`, for the upper-right corner of the portion of the image to retain.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

An error results if there is no intersection between the trimming box and the image box. Note that since the image is not bounded by a rectangular box, this method is only approximate.

Value

An object of `class` `"landsat"`, with data having been trimmed in some way.

Author(s)

Dan Kelley

See Also

The documentation for `landsat-class` explains the structure of landsat objects, and also outlines the other functions dealing with them.

latFormat *Format a latitude*

Description

Format a latitude, using "S" for negative latitude.

Usage

```
latFormat(lat, digits=max(6, getOption("digits") - 1))
```

Arguments

lat	latitude in °N north of the equator.
digits	the number of significant digits to use when printing.

Value

A character string.

Author(s)

Dan Kelley

See Also

lonFormat and latlonFormat.

latlonFormat *Format a latitude-longitude pair*

Description

Format a latitude-longitude pair, using "S" for negative latitudes, etc.

Usage

```
latlonFormat(lat, lon, digits=max(6, getOption("digits") - 1))
```

Arguments

lat	latitude in °N north of the equator.
lon	longitude in °N east of Greenwich.
digits	the number of significant digits to use when printing.

Value

A character string.

Author(s)

Dan Kelley

See Also

latFormat and lonFormat.

---

lisst *LISST dataset*

Description

LISST (Laser in-situ scattering and transmissometry) dataset, constructed artificially.

Usage

```
data(lisst)
```

Author(s)

Dan Kelley

Source

This was constructed artifically, to roughly match the sort of values that might be measured in the field.

See Also

The documentation for `lisst-class` in the Oce package explains the structure of LISST objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(lisst)
plot(lisst)

## End(Not run)
```

`lisst-class` *Class to store LISST data*

Description

Class to store LISST data.

Methods

The sections below assume a LISST object named `d`. Generally, these are created by `as.lisst` or by `read.lisst`, and the documentation for the latter should be consulted for information on the object contents.

Extracting values: The time-series data may be accessed as e.g. `d[["c1"]]` for size-class numbrer 1. There are 32 size classes. Also available are `d[["temperature"]]` and `d[["pressure"]]` and `d[["time"]]`. See `read.lisst` for further details.

Assigning values: This follows the standard form, e.g. to smooth data in size class 1 over time, do `d[["c1"]] <- smooth(d[["c1"]]`.

Overview of contents: The `show` method (e.g. `show(d)`) displays information about a lisst object named `d`.

Author(s)

Dan Kelley

References

The LIST-100 users guide (version 4.65), which provided the information for this function, was downloaded in late May 2012, from http://www.sequoiasci.com/products/fam_LISST_100.aspx.

See Also

One may read lisst objets with `read.lisst`, plot them with `plot.lisst`, and summarize them with `summary`.

lobo *lobo dataset*

Description

This is sample lobo dataset obtained in the Northwest Arm of Halifax by Satlantic.

Usage

```
data(ctd)
```

Author(s)

Dan Kelley

Source

The data were downloaded from a web interface at Satlantic LOBO web server and then read with `read.lobo` in the Oce package.

See Also

The documentation for `lobo-class` in the Oce package explains the structure of LOBO objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(lobo)
summary(lobo)
plot(lobo)

## End(Not run)
```

lobo-class *Class to store LOBO data*

Description

Class to store LOBO data, with standard slots `metadata`, `data` and `processingLog`.

Methods

Data may be accessed as e.g. `pt[["time"]]`, where the string could also be `"pressure"` or `"temperature"`. Assignment to these can be made with e.g. `pt[["pressure"]] <- value`, etc. Indeed, any quantity in the `metadata` slot or the `data` slot can be retrieved or updated in this way.

Author(s)

Dan Kelley

See Also

A lobo object may be read with read.lobo or created with as.lobo. Plots can be made with plot.lobo, while summary.lobo produces statistical summaries and show produces overviews.

---

loggerToc *Decode table-of-contents file from a datalogger*

Description

Decode table-of-contents file from a datalogger, of the sort used by some researchers at Dalhousie University.

Usage

```
loggerToc(dir, from, to, debug=getOption("oceDebug"))
```

Arguments

dir	name of a directory containing a single table-of-contents file, with .TBL at the end of its file name.
from	optional POSIXct time, indicating the beginning of a data interval of interest. This must have timezone "UTC".
to	optional POSIXct time, indicating the end of a data interval of interest. This must have timezone "UTC".
debug	optional integer to control debugging, with positive values indicating to print information about the processing.

Details

It is assumed that the .TBL file contains lines of the form "File \day179\SL08A179.023 started at Fri Jun 27 22:00:0 2008" The first step is to parse these lines to get day and hour information, i.e. 179 and 023 in the line above. Then, recognizing that it is common to change the names of such files, the rest of the file-name information in the line is ignored, and instead a new file name is constructed based on the data files that are found in the directory. (In other words, the "\day179\SL08A" portion of the line is replaced.) Once the file list is complete, with all times put into R format, then (optionally) the list is trimmed to the time interval indicated by from and to. It is important that from and to be in the UTC time zone, because that time zone is used in decoding the lines in the .TBL file.

Value

A list with two elements: filename, a vector of file names, and startTime, a vector of POSIXct times indicating the (real) times of the first datum in the corresponding files.

Author(s)

Dan Kelley

Examples

```
## Not run:
table <- loggerToc("/data/archive/sleiwex/2008/moorings/m05/adv/sontek_202h/raw",
from=as.POSIXct("2008-07-01 00:00:00", tz="UTC"),
    to=as.POSIXct("2008-07-01 12:00:00", tz="UTC"))
print(table)

## End(Not run)
```

---

lonFormat *Format a longitude*

Description

Format a longitude, using "W" for west longitude.

Usage

```
lonFormat(lon, digits=max(6, getOption("digits") - 1))
```

Arguments

lon	longitude in °N east of Greenwich.
digits	the number of significant digits to use when printing.

Value

A character string.

Author(s)

Dan Kelley

See Also

latFormat and latlonFormat.

`magneticField` *Earth magnetic declination, inclination, and intensity*

Description

Earth magnetic declination, inclination, and intensity

Usage

```
magneticField(longitude, latitude, time)
```

Arguments

`longitude`	longitude in degrees east (negative for degrees west). The dimensions must conform to lat.
`latitude`	latitude in degrees north, a number, vector, or matrix.
`time`	either a decimal year or a POSIX time corresponding to the `longitude` and `latitude` values, or a vector or matrix matching these location values.

Details

Implements the International geomagnetic reference field, based on a reworked version of a Fortran program downloaded from a NOAA website. The code seems to have been written by Susan Macmillon of the British Geological Survey. Comments in the code indicate that it employs coefficients agreed to in December 2009 by the IAGA Working Group V-MOD. The code also suggests that the valid time interval is 1900.0 to 2015.0, with values up to 2020 having reduced accuracy. And, finally, values for dates 1945 to 2005.0 are said to be "non-definitative," with other values in the range being declared "definitative."

Value

A list containing `declination`, `inclination`, and `intensity`.

Caution

This code has not been adequately checked. The hope is to get some independent test values sometime in Oct 2013.

Author(s)

Dan Kelley

References

1. The underlying Fortran code is from a NOAA website (http://www.ngdc.noaa.gov/IAGA/vmod/igrf.html).

2. A website with a calculator for these fields is at http://www.geomag.bgs.ac.uk/data_service/models_compass/wmm_calc.html, and, as of October 7 2013, this site produces values that agree with those calculated by the present function to three digits after the decimal.

Examples

```
library(oce)
# Halifax NS
magneticField(-(63+36/60), 44+39/60, 2013)

## map of North American values
data(coastlineWorld)
mapPlot(coastlineWorld, longitudelim=c(-130,-55), latitudelim=c(35,60),
proj="lambert", parameters=c(lat0=40,lat1=60), orientation=c(90,-100,0))
lon <- seq(-180, 180, 1)
lat <- seq(-90, 90)
lonm <- rep(lon, each=length(lat))
latm <- rep(lat, times=length(lon))
## Note the counter-intuitive nrow argument
decl <- matrix(magneticField(lonm, latm, 2013)$declination,
               nrow=length(lon), byrow=TRUE)
mapContour(lon, lat, decl, col='red', levels=seq(-90, 90, 5))
incl <- matrix(magneticField(lonm, latm, 2013)$inclination,
               nrow=length(lon), byrow=TRUE)
mapContour(lon, lat, incl, col='blue', levels=seq(-90, 90, 5))
```

---

makeFilter *Make a digital filter*

Description

make a digital filter

Usage

```
makeFilter(type=c("blackman-harris","rectangular", "hamming", "hann"),
           m, asKernel=TRUE)
```

Arguments

type	a string indicating the type of filter to use. (See Harris (1978) for a comparison of these and similar filters.)

- `"blackman-harris"` yields a modified raised-cosine filter designated as "4-Term (-92 dB) Blackman-Harris" by Harris (1978; coefficients given in the table on page 65). This is also called "minimum 4-sample Blackman Harris" by that author, in his Table 1, which lists figures of merit as follows: highest side lobe level -92dB; side lobe fall off -6 db/octave; coherent gain 0.36; equivalent noise bandwidth 2.00 bins; 3.0-dB bandwidth 1.90 bins; scallop loss 0.83 dB; worst case process loss 3.85 dB; 6.0-db bandwidth 2.72 bins; overlap correlation 46 percent for 75% overlap and 3.8 for 50% overlap. Note that the equivalent noise bandwidth is the width of a spectral peak, so that a value of 2 indicates a cutoff frequency of `1/m`, where `m` is as given below.
- `"rectangular"` for a flat filter. (This is just for convenience. Note that `kernel("daniell",....)` gives the same result, in kernel form.) `"hamming"` for a Hamming filter (a raised-cosine that does not taper to zero at the ends)
- `"hann"` (a raised cosine that tapers to zero at the ends).

`m` length of filter. This should be an odd number, for any non-rectangular filter.

`asKernel` boolean, set to `TRUE` to get a smoothing kernel for the return value.

Details

The filter is suitable for use by `filter`, `convolve` or (for the `asKernal=TRUE` case) with `kernapply`. Note that `convolve` should be faster than `filter`, but it cannot be used if the time series has missing values. For the Blackman-Harris filter, the half-power frequency is at `1/m` cycles per time unit, as shown in the "Examples" section. When using `filter` or `kernapply` with these filters, use `circular=TRUE`.

Value

If `asKernel` is `FALSE`, this returns a list of filter coefficients, symmetric about the midpoint and summing to 1. These may be used with `filter`, which should be provided with argument `circular=TRUE` to avoid phase offsets. If `asKernel` is `TRUE`, the return value is a smoothing kernel, which can be applied to a timeseries with `kernapply`, whose bandwidth can be determined with `bandwidth.kernel`, and which has both print and plot methods.

Author(s)

Dan Kelley

References

F. J. Harris, 1978. On the use of windows for harmonic analysis with the discrete Fourier Transform. *Proceedings of the IEEE*, 66(1), 51-83 (http://web.mit.edu/xiphmont/Public/windows.pdf.)

Examples

```
library(oce)
y <- c(rep(1,10), rep(-1,10))
x <- seq_along(y)
plot(x, y, type='o', ylim=c(-1.05, 1.05))
```

```
BH <- makeFilter("blackman-harris", 11, asKernel=FALSE)
H <- makeFilter("hamming", 11, asKernel=FALSE)
yBH <- stats::filter(y, BH)
points(x, yBH, col=2, type='o')
yH <- stats::filter(y, H)
points(yH, col=3, type='o')
legend("topright", col=1:3, cex=2/3, pch=1,
       legend=c("input", "Blackman Harris", "Hamming"))
```

---

makeSection *Bind CTD profiles together into a cross section*

Description

Combine a series of CTD profiles together to create a section.

Usage

```
makeSection(item, ...)
section + station
```

Arguments

item	either (1) a ctd object, in which case the rest of the arguments are other ctd objects, (2) a list of ctd objects, (3) a list of names of ctd objects, or (4) a list of names of files containing ctd data.
...	one or more ctd objects, either given as separate arguments, or a list of such objects.
section	a section to which a station is to be added.
station	a station to be added to a section.

Details

The stations are stored in order of the station identification number (stored as metadata$station in the ctd object), if possible. The ctd stations must share identical pressure values; use sectionGrid to do that.

Value

An object of class "section" (for details, see read.section).

Author(s)

Dan Kelley

See Also

read.ctd reads CTD data. section is a sample data set.

Examples

```
library(oce)
data(ctd)
ctdWarmed <- ctd
ctdWarmed[["temperature"]] <- ctdWarmed[["temperature"]] + 0.5
ctdWarmed[["latitude"]] <- ctdWarmed[["latitude"]] + 0.1
section <- makeSection(ctd, ctdWarmed)
summary(section)

s2 <- makeSection(ctd)
s2 <- s2 + ctdWarmed
plot(s2)
# Below is how to create a section from well-named CSV files
# that contain consecutive stations
## Not run:
    plot(sectionSmooth(sectionGrid(makeSection(dir("*.csv")))))

## End(Not run)
```

---

map2lonlat *Find lon-lat coordinates of a point on a map*

Description

Find lon-lat coordinates of a point on a map

Usage

```
map2lonlat(xusr, yusr, tolerance=1e-4)
```

Arguments

xusr, yusr	coordinates of points on the map.
tolerance	tolerance for misfit.

Details

The location is inferred by trying to invert the results of the mapproject function. The inversion is done by using optim to find the best match between a point in geographical space and plot space, and this match is not always good, being dependent on a starting guess.

Value

A list containing longitude and latitdue.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

---

mapContour *Plot contours on a existing map*

Description

Plot contours on an existing map

Usage

```
mapContour(longitude=seq(0, 1, length.out=nrow(z)),
latitude=seq(0, 1, length.out=ncol(z)),
z,
nlevels=10, levels=pretty(range(z, na.rm=TRUE), nlevels),
col=par("fg"), lty=par("lty"), lwd=par("lwd"))
```

Arguments

longitude	longitudes of points to be plotted, or an object of class topo (see topo-class), in which case longitude, latitude and z are inferred from that object.
latitude	latitudes of points to be plotted
z	matrix to be contoured
nlevels	number of contour levels, if and only if levels is not supplied
levels	list of contour levels
col	colour of lines
lty	type of lines
lwd	width of lines

Details

Adds contour lines to an existing map, using mapLines. The arguments are based on those to contour and contourLines.

Bugs

As with mapLines, long lines should be subdivided into multiple segments so that e.g. great circle lines will be curved.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
par(mar=rep(1, 4))
## Arctic 100m, 2km, 3km isobaths, showing shelves and ridges.
mapPlot(coastlineWorld,
        latitudelim=c(60,120), longitudelim=c(-130,-50),
        proj="stereographic", orientation=c(90, -90, 0),
axes=FALSE, fill='lightgray')
data(topoWorld)
lon <- topoWorld[['longitude']]
lat <- topoWorld[['latitude']]
z <- topoWorld[['z']]
mapContour(lon, lat, z, levels=c(-100, -2000, -3000),col=1:3,lwd=2)

## End(Not run)
```

---

mapImage *Plot an image on a existing map*

Description

Plot an image on an existing map

Usage

```
mapImage(longitude, latitude, z, zlim, zclip=FALSE,
         breaks, col, colormap, border=NA,
         lwd=par("lwd"), lty=par("lty"),
         filledContour=FALSE, missingColor=NA, debug=getOption("oceDebug"))
```

Arguments

longitude	longitudes of grid lines
latitude	latitudes of grid lines
z	matrix for image
zlim	limit for z (colour)
zclip	logical, indicating whether to clip the colours to those corresponding to zlim, if the latter is provided. Clipped regions will be coloured with missingColor.
breaks	the z values for breaks in the colour scheme. If this is of length 1, the value indicates the desired number of breaks, which is supplied to pretty, in determining clean break points.

col	either a vector of colours corresponding to the breaks, of length 1 plus the number of breaks, or a function specifying colours, e.g. oceColorsJet for a rainbow.
colormap	optional colormap, as created by colormap. If provided, this takes precedence over breaks, codecol and missingColor.
border	colour used for borders of patches (passed to polygon); the default NA means no border.
lwd	line width, used if borders are drawn
lty	line type, used if borders are drawn
filledContour	boolean value indicating whether to use filled contours to plot the image. This is ignored at present.
missingColor	a color to be used to indicate missing data, or NA to skip the drawing of such regions (which will retain whatever material has already been drawn at the regions).
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more. (Temporary note: setting this to 99 forces the use of a drawing method that draws polygons one by one, which is possibly less error prone in terms of pruning data, but is much slower than the default method. This 99 option will be removed without notice, as it is really to be used mainly by the developers and testers.)

Details

Adds an image to an existing map, by analogy to image.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
mapPlot(coastlineWorld, type='l',
longitudelim=c(-70,-50), latitudelim=c(40,50),
proj="polyconic", orientation=c(90, -60,0), grid=TRUE)
data(topoWorld)
mapImage(topoWorld, col=oceColorsGebco)
mapMeridians(10, lty='dotted', col='darkgray')
mapZones(10, lty='dotted', col='darkgray')
mapLines(coastlineWorld)

## Northern polar region, with colour-coded bathymetry
```

```
drawPalette(c(-5000,0), zlim=c(-5000, 0), col=oceColorsJet)
mapPlot(coastlineWorld, type='l',
longitudelim=c(-180,180), latitudelim=c(60,120),
proj="stereographic", grid=TRUE)
data(topoWorld)
mapImage(topoWorld, zlim=c(-5000, 0), col=oceColorsJet)
mapMeridians()
mapZones()
mapLines(coastlineWorld[['longitude']], coastlineWorld[['latitude']])

# Levitus SST (requires dataset from http://www.esrl.noaa.gov)
con <- open.ncdf("/data/oar/levitus/temperature_annual_1deg.nc")
##con <- open.ncdf("/data/oar/levitus/temperature_annual_5deg.nc")
lon <- get.var.ncdf(con, "lon")
lat <- get.var.ncdf(con, "lat")
SST <- get.var.ncdf(con, "t_an")[,,1]
##SST <- get.var.ncdf(con, "t_mn")[,,1]
par(mar=rep(1, 4))
Tlim <- c(-2, 30)
drawPalette(Tlim, col=oceColorsJet)
mapPlot(coastlineWorld, projection='mollweide', grid=FALSE)
mapImage(lon, lat, SST, col=oceColorsJet, zlim=Tlim, debug=99)
mapPolygon(coastlineWorld, col='gray')

## End(Not run)
```

`mapLines` *Plot lines on a existing map*

Description

Plot lines on an existing map

Usage

```
mapLines(longitude, latitude, greatCircle=FALSE, ...)
```

Arguments

`longitude`	longitudes of points to be plotted, or an object from which longitude and latitude can be inferred (e.g. a coastline file, or the return value from `mapLocator`), in which case the following two arguments are ignored.
`latitude`	latitudes of points to be plotted
`greatCircle`	boolean indicating whether to render line segments as great circles. (Ignored at present.)
`...`	optional arguments passed to `lines`.

Details

Adds lines to an existing map, by analogy to `lines`.

Author(s)

Dan Kelley

See Also

See `mapPlot` for general information on plotting maps, including other functions.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
longitude <- coastlineWorld[['longitude']]
latitude <- coastlineWorld[['latitude']]
mapPlot(longitude, latitude, type='l',
longitudelim=c(-80,10), latitudelim=c(0,120),
projection="orthographic", orientation=c(45,-100,0))
lon <- c(-63.5744, 0.1062)             # Halifax CA to London UK
lat <- c(44.6479, 51.5171)
mapLines(lon, lat, col='red')

## End(Not run)
```

---

mapLocator *Locate points on a existing map*

Description

Locate points on an existing map

Usage

```
mapLocator(n=512, type='n', ...)
```

Arguments

`n`	number of points to locate; see `locator`.
`type`	type of connector for the points; see `locator`.
`...`	extra arguments passed to `locator` (and either `mapPoints` or `mapLines`, if appropriate) if `type` is not `'n'`.

Details

This uses `map2lonlat` to infer the location in geographical space; see the documentation for that function on its limitations.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

---

mapLongitudeLatitudeXY

Convert from longitude and latitude to x and y

Description

Find (x,y) values corresponding to (longitude, latitude) values, using the present projection.

Usage

```
mapLongitudeLatitudeXY(longitude, latitude)
```

Arguments

longitude	longitudes of points, or an object from which latitude and longitude can be inferred (e.g. a coastline file, or the return value from mapLocator), in which case the following two arguments are ignored.
latitude	latitudes of points, needed only if they cannot be inferred from from the first argument

Details

This is mainly a wrapper around mapproject.

Value

A list containing x and y.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
xy <- mapLongitudeLatitudeXY(coastlineWorld)
plot(xy, type='l', asp=1)

## End(Not run)
```

`mapMeridians` *Plot meridians on an existing map*

Description

Plot meridians (lines of constant latitude) on a existing map

Usage

```
mapMeridians(latitude, lty='solid', lwd=0.5*par('lwd'), col='darkgray', ...)
```

Arguments

`latitude`	either a boolean indicating whether to draw a meridian grid, or a vector of latitudes at which to draw meridians.
`lty`	line type
`lwd`	line width
`col`	colour
`...`	optional arguments passed to `lines`.

Details

Meridians that will not fit in the plotting space are ignored.

Bugs

This should use `approx` to fill in multiple segments within the line, so that e.g. great circle lines will be curved.

Author(s)

Dan Kelley

See Also

See `mapPlot` for general information on plotting maps, including other functions.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
longitude <- coastlineWorld[['longitude']]
latitude <- coastlineWorld[['latitude']]
mapPlot(longitude, latitude, type='l',
longitudelim=c(-130,-50), latitudelim=c(30,60),
proj="polyconic", orientation=c(90, -90, 0))
mapMeridians()

## End(Not run)
```

`mapPlot` *Plot a map*

Description

Plot a map

Usage

```
mapPlot(longitude, latitude, longitudelim, latitudelim, grid=TRUE,
    bg, fill=NULL, type='l', axes=TRUE, drawBox=TRUE, showHemi=TRUE,
    polarCircle=0,
    projection="mollweide", parameters=NULL, orientation=NULL,
    debug=getOption("oceDebug"),
    ...)
```

Arguments

`longitude`	longitudes of points to be plotted, or an object with a slot named `data` that contains items named `longitude` and `latitude` (e.g. of class `coastline`; see `coastline-class`), in which case position is inferred from the object and the second argument to this function is ignored. If `longitude` is missing, it is set to `coastlineWorld`.
`latitude`	latitudes of points to be plotted
`longitudelim`	optional limit of longitudes to plot
`latitudelim`	optional limit of latitudes to plot
`grid`	either a number (or pair of numbers) indicating the spacing of longitude and latitude lines, in degrees, or a logical value indicating whether to draw a default grid with 15 degree spacing.
`bg`	background colour for plot (ignored at present).
`fill`	colour to be used to fill land regions, or `NULL` to avoid filling. For some map projections (particularly if the view includes the dateline), the filled region may not line up with coastlines, making it sensible to set `fill=NULL`; see "Examples".

type	value to indicate type of plot, as with par("plot").
axes	logical value indicating whether to draw longitude and latitude values in the lower and left margin, respectively. This may not work well for some projections or scales.
drawBox	logical value indicating whether to draw a box around the plot. This is helpful for many projections at sub-global scale.
showHemi	logical value indicating whether to show the hemisphere in axis tick labels.
polarCircle	a number indicating the number of degrees of latitude extending from the poles, within which zones are not drawn.
projection	projection; see mapproject.
parameters	parameters for projection; see mapproject.
orientation	orientation for projection; see mapproject.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions.

Details

Creates a map using the indicated projection. The available projections can be found in the documentation for mapproject, which also provides information on the arguments named parameters and orientation. Further details on these and other projections are provided by Snyder (1987), an exhaustive treatment that includes many illustrations, an overview of the history of the topic, and some notes on the strengths and weaknesses of the various formulations. See especially pages 2 through 7, which define terms and provide recommendations.

Author(s)

Dan Kelley

References

Snyder, John P., 1987. Map Projections: A Working Manual. USGS Professional Paper: 1395 (available at pubs.usgs.gov/pp/1395/report.pdf).

Jenny, B., 2012. Adaptive composite map projections. IEEE Transactions on Visualization and Computer Graphics (Proceedings Scientific Visualization / Information Visualization 2012), 18-22, p 2575-2582.

XKCD guide to map projections (http://xkcd.com/977/).

See Also

Points may be added to a map with mapPoints, lines with mapLines, and text with mapText. Points on a map may be found with mapLocator. Great circle paths can be calculated with geodGc. Scale bars may be added with mapScalebar.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)

## Mollweide is an equal-area projection that works well for
## whole-globe views, below shown in default and Pacific-focus
## views.  See Snyder (1987 page 54).
mapPlot(coastlineWorld, proj="mollweide")
mapPlot(coastlineWorld, proj="mollweide", orientation=c(90,-180,0))

## Orthographic projections resemble a globe, making them
## attractive for non-technical use, but they are neither conformal
## nor equal-area, so they are somewhat limited for serious
## use on large scales.  See Snyder (1987 section 20).  The
## examples given below illustrate use for low-latitude
## Pacific and high-latitude Atlantic/Arctic.
mapPlot(coastlineWorld, projection="orthographic",
        orientation=c(0,-180,0))
mapPlot(coastlineWorld, projection="orthographic",
        orientation=c(60,-40,0))

## The Lambert conformal conic projection is an equal-area
## projection recommended by Snyder for regions of large
## east-west extent away from the equator, here illustrated
## for the USA and Canada.  Readers should compare the results
## with those with a polygonic projection, which is also popular.
mapPlot(coastlineWorld, longitudelim=c(-130,-55), latitudelim=c(35,60),
        proj="lambert", parameters=c(lat0=40,lat1=60),
        orientation=c(90,-100,0))

## The stereographic projection (Snyder 1987 page 120) is conformal,
## used below for an Arctic view with a Canadian focus.
mapPlot(coastlineWorld, longitudelim=c(-130,-50), latitudelim=c(70,110),
        proj="stereographic", orientation=c(90, -135, 0), fill='gray')

## End(Not run)
```

---

mapPoints *Plot points on a existing map*

Description

Plot points on an existing map

Usage

```
mapPoints(longitude, latitude, ...)
```

Arguments

longitude	longitudes of points to be plotted, or an object from which longitude and latitude can be inferred (e.g. a coastline file, or the return value from mapLocator), in which case the following two arguments are ignored.
latitude	latitudes of points to be plotted
...	optional arguments passed to points.

Details

Adds points to an existing map, by analogy to points.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
mapPlot(coastlineWorld, type='l', grid=TRUE,
longitudelim=c(-80,0), latitudelim=c(20,50),
projection="mercator", orientation=c(90,-40,0))
lon <- section[["longitude", 'byStation']]
lat <- section[["latitude", 'byStation']]
mapPoints(lon, lat, pch=20, col='red')

## End(Not run)
```

---

mapPolygon *Plot a polygon on a existing map*

Description

Plot a polygon on an existing map

Usage

```
mapPolygon(longitude, latitude, density=NULL, angle=45,
    border=NULL, col=NA, lty=par('lty'), ..., fillOddEven=FALSE)
```

Arguments

longitude	longitudes of points to be plotted, or an object from which longitude and latitude can be inferred (e.g. a coastline file, or the return value from mapLocator), in which case the following two arguments are ignored.
latitude	latitudes of points to be plotted
density	as for polygon
angle	as for polygon
border	as for polygon
col	as for polygon
lty	as for polygon
...	as for polygon
fillOddEven	as for polygon

Details

Adds a polygon to an existing map, by analogy to polygon. Used by mapImage.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
mapPlot(coastlineWorld, type='l',
longitudelim=c(-70,-50), latitudelim=c(40,50),
proj="polyconic", orientation=c(90, -90,0), grid=TRUE)
data(topoWorld)
tlon <- topoWorld[['longitude']][550:650]
tlat <- topoWorld[['latitude']][240:300]
z <- topoWorld[['z']][550:650, 240:300]
mapImage(tlon, tlat, z)
mapLines(coastlineWorld[['longitude']], coastlineWorld[['latitude']])

## End(Not run)
```

mapScalebar *Draw a scalebar on an existing map*

Description

Draw a scalebar on an existing map

Usage

```
mapScalebar(x, y=NULL, length,
            lwd=4*par("lwd"), cex=1.2*par("cex"), col="black")
```

Arguments

x, y	position of the scalebar. Eventually this should be similar to the corresponding arguments in legend, but at the moment y must be NULL and x must be "topleft".
length	length to indicate, in kilometres. If not provided, a possibly reasonable choice is made.
lwd	scalebar line width
col	colour
cex	character size for label

Details

The scale is appropriate to the centre of the plot, and will become increasingly inaccurate away from that spot, with the error depending on the projection and the fraction of the earth that is shown.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

Examples

```
## Not run:
library(oce)
## Arctic Ocean
data(coastlineWorld)
latlim <- 90 + 25 * c(-1, 1)
lonlim <- c(-130, -50)
orientation <- c(90, -90, 0)
proj <- "stereographic"
fill <- "lightgray"
```

```
mapPlot(coastlineWorld, latitudelim=latlim, longitudelim=lonlim,
        proj=proj, orientation=orientation,
        axes=FALSE, fill=fill)
mapScalebar()

## End(Not run)
```

---

mapText *Plot text on a existing map*

Description

Plot text on an existing map

Usage

```
mapText(longitude, latitude, labels, ...)
```

Arguments

longitude	longitudes of text to be plotted
latitude	latitudes of text to be plotted
labels	labels of text to be plotted
...	optional arguments passed to text, e.g. adj, pos, etc.

Details

Adds text to an existing map, by analogy to text.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

Examples

```
library(oce)
data(coastlineWorld)
longitude <- coastlineWorld[['longitude']]
latitude <- coastlineWorld[['latitude']]
mapPlot(longitude, latitude, type='l',
longitudelim=c(-70,-50), latitudelim=c(45,50),
projection="mercator", grid=1)
lon <- -63.5744 # Halifax
lat <- 44.6479
mapPoints(lon, lat, pch=20, col="red")
mapText(lon, lat, "Halifax", col="red", pos=1, offset=1)
```

mapZones *Plot zones on an existing map*

Description

Plot zones (lines of constant longitude) on a existing map

Usage

```
mapZones(longitude, polarCircle=0, lty='solid', lwd=0.5*par('lwd'), col='darkgray', ...)
```

Arguments

longitude	either a boolean indicating whether to draw a zonal grid, or a vector of longitudes at which to draw zones.
polarCircle	a number indicating the number of degrees of latitude extending from the poles, within which zones are not drawn.
lty	line type
lwd	line width
col	colour
...	optional arguments passed to lines.

Details

Zones that will not fit in the plotting space are ignored.

Bugs

This should use approx to fill in multiple segments within the line, so that e.g. great circle lines will be curved.

Author(s)

Dan Kelley

See Also

See mapPlot for general information on plotting maps, including other functions.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
longitude <- coastlineWorld[['longitude']]
latitude <- coastlineWorld[['latitude']]
mapPlot(longitude, latitude, type='l', grid=FALSE,
longitudelim=c(-80,10), latitudelim=c(0,120),
projection="orthographic", orientation=c(45,-100,0))
mapZones()

## End(Not run)
```

---

matchBytes *Locate byte sequences in a raw vector*

Description

Find spots in a raw vector that match a given byte sequence.

Usage

```
matchBytes(input, b1, ...)
```

Arguments

input	a vector of raw (byte) values.
b1	a vector of bytes to match (must be of length 2 or 3 at present; for 1-byte, use which).
...	additional bytes to match for (up to 2 permitted)

Value

List of the indices of input that match the start of the bytes sequence (see example).

Author(s)

Dan Kelley

Examples

```
buf <- as.raw(c(0xa5, 0x11, 0xaa, 0xa5, 0x11, 0x00))
match <- matchBytes(buf, 0xa5, 0x11)
print(buf)
print(match)
```

matrixSmooth *Smooth a matrix.*

Description

Smooth a matrix.

Usage

```
matrixSmooth(m, passes=1)
```

Arguments

m	a matrix to be smoothed.
passes	an integer specifying the number of times the smoothing is to be applied.

Details

The values on the edge of the matrix are unaltered. For interior points, the result is defined in terms in terms of the original as follows. $r_{i,j} = (2m_{i,j} + m_{i-1,j} + m_{i+1,j} + m_{i,j-1} + m_{i,j+1})/6$. Note that missing values propagate to neighbours.

Value

A smoothed matrix.

Author(s)

Dan Kelley

Examples

```
library(oce)
opar <- par(no.readonly = TRUE)
m <- matrix(rep(seq(0, 1, length.out=5), 5), nrow=5, byrow=TRUE)
m[3,3] <- 2
m1 <- matrixSmooth(m)
m2 <- matrixSmooth(m1)
m3 <- matrixSmooth(m2)
par(mfrow=c(2,2))
image(m,  col=rainbow(100), zlim=c(0,4), main="original image")
image(m1, col=rainbow(100), zlim=c(0,4), main="smoothed 1 time")
image(m2, col=rainbow(100), zlim=c(0,4), main="smoothed 2 times")
image(m3, col=rainbow(100), zlim=c(0,4), main="smoothed 3 times")
par(opar)
```

met *Sample meteorological object*

Description

This is sample `met` object, provided for testing, created by the code given in "Examples".

Usage

```
data(met)
```

Source

Downloaded from the Environment Canada website http://climate.weatheroffice.gc.ca/climateData/bulkdata_e.html?timeframe=1&Prov=XX&StationID=6358&Year=2011&Month=7&Day=15&format=csv on November 17, 2011.

See Also

The documentation for `met-class` in the Oce package explains the structure of `met` objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(met)
plot(met)

## End(Not run)
```

met-class *Class to store meteorological data*

Description

Class to store meteorological data, with standard slots `metadata`, `data` and `processingLog`.

Methods

Accessing values. For an object named `m`, temperature may be accessed as `m[["time"]]`, which yields a vector. Similarly pressure is `m[["pressure"]]`. The components of wind are stored in *oceanographic* convention, i.e. the eastward component is available as `m[["u"]]` and the northward component is `m[["v"]]`. `m[["u"]]`, and wind direction is `m[["direction"]]`. The filename from which the data came (if any) may be found with `m[["filename"]]`.

Assigning values. Everything that may be accessed may also be assigned, e.g. `m[["temperature"]] <- 1 + m[["temperatu` increases temperature by 1C.

Overview of contents. The `show` method (e.g. `show(met)`) displays information about the object.

Author(s)

Dan Kelley

See Also

A file containing CTD profile data may be read with `read.met`, and a CTD object can also be created with `as.met`. A sample object is available with `data(met)`. Statistical summaries are provided by `summary.met`, while `show` displays an overview. Plotting may be handled with `plot.met`.

`moonAngle` *Lunar angle as function of space and time.*

Description

Lunar angle as function of space and time.

Usage

```
moonAngle(t, longitude=0, latitude=0, useRefraction=TRUE)
```

Arguments

`t`	time, a POSIXt object (must be in timezone UTC), or a numeric value that corresponds to such a time.
`longitude`	observer longitude in degrees east
`latitude`	observer latitude in degrees north
`useRefraction`	boolean, set to `TRUE` to apply a correction for atmospheric refraction. (Ignored at present.)

Details

The calculations are based on formulae provided by Meeus [1982], primarily in chapters 6, 18, and 30. The first step is to compute sidereal time as formulated in Meeus [1982] chapter 7, which in turn uses Julian day computed according to as formulae in Meeus [1982] chapter 3. Using these quantities, formulae in Meeus [1982] chapter 30 are then used to compute geocentric longitude ($lambda$, in the Meeus notation), geocentric latitude ($beta$), and parallax. Then the `obliquity` of the ecliptic is computed with Meeus [1982] equation 18.4. Equatorial coordinates (right ascension and declination) are computed with equations 8.3 and 8.4 from Meeus [1982], using `eclipticalToEquatorial`. The hour angle (H) is computed using the unnumbered equation preceding Meeus's [1982] equation 8.1. Finally, Meeus [1982] equations 8.5 and 8.6 are used to calculate the local azimuth and altitude of the moon, using `equatorialToLocalHorizontal`.

Value

A list containing the following.

`time`	time
`azimuth`	moon azimuth, in degrees eastward of north, from 0 to 360. Note: this is not the convention used by Meeus, who uses degrees westward of South. (See diagram below.)
`altitude`	moon altitude, in degrees from -90 to 90. (See diagram below.)
`rightAscension`	right ascension, in degrees
`declination`	declination, in degrees
`lambda`	geocentric longitude, in degrees
`beta`	geocentric latitude, in degrees
`diameter`	lunar diameter, in degrees.
`distance`	earth-moon distance, in kilometers)
`illuminatedFraction`	fraction of moon's visible disk that is illuminated
`phase`	phase of the moon, defined in equation 32.3 of Meeus [1982]. The fractional part of which is 0 for new moon, 1/4 for first quarter, 1/2 for full moon, and 3/4 for last quarter.

Alternate formulations

Formulae provide by Meeus [1982] are used for all calculations here. Meeus [1991] provides formulae that are similar, but that differ in the 5th or 6th digits. For example, the formula for ephemeris time in Meeus [1991] differs from that in Meeus [1992] at the 5th digit, and almost all of the approximately 200 coefficients in the relevant formulae also differ in the 5th and 6th digits. Discussion of the changing formulations is best left to members of the astronomical community. For the present purpose, it may be sufficient to note that `moonAngle`, based on Meeus [1982], reproduces the values provided in example 45.a of Meeus [1991] to 4 significant digits, e.g. with all angles matching to under 2 minutes of arc.

Author(s)

Dan Kelley, based on formulae in Meeus [1982].

References

Meeus, Jean, 1982. Astronomical formulae for calculators. Willmann-Bell. Richmond VA, USA. 201 pages.

Meeus, Jean, 1991. Astronomical algorithms. Willmann-Bell, Richmond VA, USA. 429 pages.

See Also

The equivalent function for the sun is `sunAngle`.

Examples

```
library(oce)
par(mfrow=c(3,2))
y <- 2012
m <- 4
days <- 1:3
## Halifax sunrise/sunset (see e.g. http://www.timeanddate.com/worldclock)
rises <- ISOdatetime(y, m, days,c(13,15,16), c(55, 04, 16),0,tz="UTC") + 3 * 3600 # ADT
sets <- ISOdatetime(y, m,days,c(3,4,4), c(42, 15, 45),0,tz="UTC") + 3 * 3600
azrises <- c(69, 75, 82)
azsets <- c(293, 288, 281)
latitude <- 44.65
longitude <- -63.6
for (i in 1:3) {
    t <- ISOdatetime(y, m, days[i],0,0,0,tz="UTC") + seq(0, 24*3600, 3600/4)
    ma <- moonAngle(t, longitude, latitude)
    oce.plot.ts(t, ma$altitude, type='l', mar=c(2, 3, 1, 1), cex=1/2, ylab="Altitude")
    abline(h=0)
    points(rises[i], 0, col='red', pch=3, lwd=2, cex=1.5)
    points(sets[i], 0, col='blue', pch=3, lwd=2, cex=1.5)
    oce.plot.ts(t, ma$azimuth, type='l', mar=c(2, 3, 1, 1), cex=1/2, ylab="Azimuth")
    points(rises[i], -180+azrises[i], col='red', pch=3, lwd=2, cex=1.5)
    points(sets[i], -180+azsets[i], col='blue', pch=3, lwd=2, cex=1.5)
}
```

---

nao *North Atlantic Oscillation Index*

Description

This is the North Oscillation Index, downloaded in May 2014 and processed as follows.

```
d <- scan("http://www.cpc.ncep.noaa.gov/products/precip/CWlink/pna/norm.nao.monthly.b5001.current.as
isYear <- d > 1900
index <- d[!isYear]
year <- 1/24 + seq(d[isYear][1], by=1/12, length.out=length(index))
nao <- data.frame(year=year, index=index)
```

Usage

```
data(nao)
```

Author(s)

Dan Kelley

Source

http://www.cpc.ncep.noaa.gov/products/precip/CWlink/pna/norm.nao.monthly.b5001.current.ascii.table

Examples

```
## Not run:
library(oce)
data(nao)
plot(nao$year, nao$index, type='l')

## End(Not run)
```

---

numberAsHMS *Convert a numeric time to hour, minute, and second*

Description

Convert a numeric time to hour, minute, and second

Usage

```
numberAsHMS(t, default=0)
```

Arguments

t	a vector of factors or character strings, in the format 1200 for 12:00, 0900 for 09:00, etc.
default	value to be used for the returned hour, minute and second if there is something wrong with the input value (e.g. its length exceeds 4 characters, or it contains non-numeric characters)

Value

A list containing hour, minute, and second, the last of which is always zero.

Author(s)

Dan Kelley

See Also

numberAsHMS

Examples

```
t <- c("0900", "1234")
numberAsHMS(t)
```

numberAsPOSIXct *Convert a numeric time to a POSIXct time*

Description

Convert a numeric time to a POSIXct time

Usage

```
numberAsPOSIXct(t, type=c("unix", "matlab", "gps", "argo",
                          "sas", "spss", "yearday"), tz="UTC")
```

Arguments

t	an integer corresponding to a time, in a way that depends on type (see "Details").
type	the type of time (see "Details").
tz	a string indicating the time zone, used only for unix and matlab times, since GPS times are always referenced to the UTC timezone.

Details

Unix times, indicated by type="unix", are measured in seconds since the start of the year 1970. Matlab times, indicated by type="matlab", are measured in years since the start of the year 1899. Argo times, indicated by type="argo", are measured in days since the start of the twentieth century. SAS times, indicated by type="sas", have origin at the start of 1960. SPSS times, indicated by type="spss", have origin at 1582-10-14.

Note that in these cases, t is a vector. In the yearday and GPS cases, however, t must be a two-column matrix. For type="gps", the first column is the GPS "week" (referenced to 1999-08-22) and the second column is GPS "second" (i.e. the second within the week). For type="yearday", the first column is the year, and the second is the yearday (starting at 1 for the first second of January 1, to match the convention used by Sea-Bird CTD software).

Value

A POSIXct time.

Author(s)

Dan Kelley

See Also

numberAsHMS

Examples

```
numberAsPOSIXct(0)                      # unix time 0
numberAsPOSIXct(1, type="matlab")       # matlab time 1
numberAsPOSIXct(cbind(566,345615), type="gps") # Canada Day
numberAsPOSIXct(cbind(2013, 0), type="yearday") # start of 2013
```

`oce-class` *base class of oce package*

Description

This is class from which all objects in the oce package inherit.

Slots

`metadata` List containing metadata, such as file names, instrument configurations, etc.

`data` List containing the actual data.

`processingLog` List containing a time-stamped record of changes that have been made to the object, either automatically recorded by functions in the package, or added by the user with `oceEdit`.

Methods

All classes in the Oce package inherit from this base class, so they all possess the following methods. Note that some classes extend the methods, and these extensions are discusssed in the detailed documentation files. For example, see `section-class` to see how latitudes of sections may be extracted either with one value per station, or with a value for each of the pressures within each of the stations.

[[Provides read-only access to an item in the object's `metadata` or `data`. The item is sought first in the meta-data, then in the data. For example, `ctd[["filename"]]` yields the name of the file from which a CTD object named `ctd` had been read.

[[<- Alters the named item in the object's `metadata` or `data`, the former being examined before the latter. For example, to add 0.005 to the salinity of a CTD object named `ctd`, one might write `ctd[["salinity"]] <- 0.005 + ctd[["salinity"]]`.

show Displays brief information about the object named as an argument. For example, `show(ctd)` would provide such an overview for a CTD object as discussed above.

Author(s)

Dan Kelley

See Also

Information on the classes that derive from this base class are found at the following links: `adp-class`, `adv-class`, `cm-class`, `coastline-class`, `ctd-class`, `drifter-class`, `echosounder-class`, `lisst-class`, `lobo-class`, `met-class`, `sealevel-class`, `section-class`, `tdr-class`, `tidem-class`, `topo-class`, and `windrose-class`.

`oce.as.POSIXlt` *More general form of as.POSIXlt*

Description

A more general form of as.POSIXlt.

Usage

```
oce.as.POSIXlt(x, tz = "")
```

Arguments

`x`	a date, as for `as.POSIXlt`, but also including forms in which the month name appears.
`tz`	the timezone, as for `as.POSIXlt`

Details

Used in parsing headers, this function is built on the standard `as.POSIXlt` function. the only difference is that this also recognizes dates of forms such as `"2002 100 1430"` (year day hhmm), `"Aug 23 2002"`, `"August 23 2002"`, `"2002 Aug 23"`, and `"2002 23 Aug"`. (The month may appear in abbreviated form or written in full, and may be capitalized or not.)

Value

A POSIXlt object.

Author(s)

Dan Kelley

See Also

`as.POSIXlt`, from which this is derived.

oce.as.raw *Version of as.raw() that clips data*

Description

A version of as.raw() that clips data to prevent warnings

Usage

```
oce.as.raw(x)
```

Arguments

x	values to be converted to raw

Details

Negative values are clipped to 0, while values above 255 are clipped to 255; the result is passed to as.raw and returned.

Value

Raw values corresponding to x.

Author(s)

Dan Kelley

Examples

```
x <- c(-0.1, 0, 1, 255, 255.1)
data.frame(x, oce.as.raw(x))
```

oce.axis.POSIXct *Modified version of axis.POSIXct*

Description

Modified version of axis.POSIXct

Usage

```
oce.axis.POSIXct(side, x, at, tformat, labels = TRUE,
  drawTimeRange=TRUE, abbreviateTimeRange=FALSE, drawFrequency=FALSE,
  cex=par("cex"), cex.axis=par("cex.axis"), cex.main=par("cex.main"),
  mar=par("mar"), mgp=par("mgp"),
  main="", debug=getOption("oceDebug"), ...)
```

Arguments

side	as for axis.POSIXct.
x	as for axis.POSIXct.
at	as for axis.POSIXct.
tformat	as format for axis.POSIXct for now, but eventually will have new features for multiline labels, e.g. day on one line and month on another.
labels	as for axis.POSIXct.
drawTimeRange	boolean, TRUE to draw a time range on the opposite side of the plot.
drawFrequency	boolean, TRUE to show the frequency of sampling in the data
abbreviateTimeRange	boolean, TRUE to abbreviate the second number in the time range, e.g. dropping the year if it is the same in the first number.
cex	size of labels on axes; see par("cex").
cex.axis	see par("cex.axis").
cex.main	see par("cex.main").
mar	value for par(mar) for axis
mgp	value for par(mgp) for axis
main	title of plot
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	as for axis.POSIXct.

Details

As axis.POSIXct but with axis labels obeying the timezone of x. This will not be needed for R 2.9 and later, but is included so that oce will work even with earlier versions.

Value

The locations on the axis scale at which tick marks were drawn.

Author(s)

Dan Kelley

See Also

This is used mainly by oce.plot.ts.

`oce.plot.ts` *Plot a time-series, obeying the timezone*

Description

Plot a time-series, obeying the timezone and possibly drawing the range in the top-left margin

Usage

```
oce.plot.ts(x, y, type="l", xlim, ylim, xlab, ylab,
            drawTimeRange=TRUE, adorn=NULL, fill=FALSE,
            cex=par("cex"), cex.axis=par("cex.axis"), cex.main=par("cex.main"),
            mgp=getOption("oceMgp"),
          mar=c(mgp[1]+if(nchar(xlab)>0) 1.5 else 1, mgp[1]+1.5, mgp[2]+1, mgp[2]+3/4),
            main="",
            despike=FALSE,
            axes=TRUE, tformat,
            marginsAsImage=FALSE,
            grid=FALSE, grid.col="darkgray", grid.lty="dotted", grid.lwd=1,
            debug=getOption("oceDebug"),
            ...)
```

Arguments

`x`	the times of observations.
`y`	the observations.
`type`	plot type, `"l"` for lines, `"p"` for points.
`xlim`	optional limit for x axis.
`ylim`	optional limit for y axis.
`drawTimeRange`	a boolean, set to `TRUE` to indicate the range of times in the top-left margin.
`adorn`	optional `expression` to be performed immediately after drawing the panel. (See `plot.adp` for an example.)
`fill`	boolean, set `TRUE` to fill the curve to zero (which it does incorrectly if there are missing values in y).
`xlab`	name for x axis; defaults to `""`.
`ylab`	name for y axis; defaults to the plotted item.
`cex`	size of labels on axes; see `par`("cex").
`cex.axis`	see `par`("cex.axis").
`cex.main`	see `par`("cex.main").
`mgp`	3-element numerical vector to use for `par(mgp)`, and also for `par(mar)`, computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.

mar	value to be used with par("mar") to set margins. THe default value uses significantly tighter margins than is the norm in R, which gives more space for the data. However, in doing this, the existing par("mar") value is ignored, which contradicts values that may have been set by a previous call to drawPalette. To get plot with a palette, first call drawPalette, then call oce.plot.ts with mar=par("mar").
main	title of plot.
despike	boolean flag that can turn on despiking with despike.
axes	boolean, set to TRUE to get axes plotted
tformat	optional format for labels on the time axis
marginsAsImage	boolean indicatingn whether to set the right-hand margin to the width normally taken by an image drawn with imagep.
grid	if TRUE, a grid will be drawn for each panel. (This argument is needed, because calling grid after doing a sequence of plots will not result in useful results for the individual panels.
grid.col	color of grid
grid.lty	line type of grid
grid.lwd	line width of grid
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	graphical parameters passed down to plot.

Details

Depending on the version of R, the standard plot and plot.ts routines will not obey the time zone of the data. This routine gets around that problem. It can also plot the time range in the top-left margin, if desired; this string includes the timezone, to remove any possible confusion.

The time axis is drawn with oce.axis.POSIXct.

Author(s)

Dan Kelley

Examples

```
library(oce)
t.start <- as.POSIXct("2008-01-01", tz="UTC")
t <- seq(t.start, length.out=48, by="30 min")
y <- sin(as.numeric(t - t.start) * 2 * pi / (12 * 3600))
oce.plot.ts(t, y, type='l', xaxs='i')

## Not run:
weatherplot <- function(id=6358, time=as.POSIXlt(Sys.Date()))
{
    ## The default stationID is Halifax, NS, Canada
    site <- "http://www.climate.weatheroffice.gc.ca/climateData/bulkdata_e.html"
```

```
    time <- as.POSIXlt(time)
    f <- paste(site,
               "?timeframe=1&StationID=", id,
               "&&Year=", time$year+1900,
               "&Month=", time$mon+1,
               "&Day=", time$mday,
               "&format=csv", sep="")
    ## Read lines first, to see where header ends.  Cannot parse header, owing
    ## to multi-byte strings there.
    ll <- readLines(f)
    skip <- which(grepl("Date", ll, perl=TRUE))
    d <- read.csv(f, skip=skip, header=FALSE)
    t <- as.POSIXct(d[,1])
    p <- d[,19]
   oce.plot.ts(t, p, type='l', ylab="Pressure [dbar]", cex=2/3, main=paste("Station", id))
}
weatherplot()

## End(Not run)
```

`oce.write.table` *Write the data portion of object to a file*

Description

Write the data portion of object to a file

Usage

```
oce.write.table(x, file="", ...)
```

Arguments

`x`	an `oce` object that contains a `data` table.
`file`	file name, as passed to `write.table`. Use `""` to get a listing in the terminal window.
`...`	optional arguments passed to `write.table`.

Details

The output has a line containing the names of the columns in `x$data`, each enclosed in double quotes. After that line are lines for the data themselves. The default is to separate data items by a single space character, but this can be altered by using a `sep` argument in the `...` list (see `write.table`).

This function is little more than a thin wrapper around `write.table`, the only difference being that row names are ommitted here, making for a file format that is more conventional in Oceanography.

Value

The value of write.table is returned.

Author(s)

Dan Kelley

See Also

write.table, which does the actual work.

---

oceApprox *Interpolate 1D data with Unesco or Reiniger-Ross algorithm*

Description

Interpolate one-dimensional data using schemes that permit curvature but tends minimize extrema that are not well-indicated by the data.

Usage

```
oceApprox(x, y, xout, method=c("rr", "unesco"))
```

Arguments

x	the independent variable (z or p, usually).
y	the dependent variable.
xout	the values of the independent variable at which interpolation is to be done.
method	method to use. See "Details".

Details

Setting method="rr" yields the weighted-parabola algorithm of Reiniger and Ross (1968). For procedure is as follows. First, the interpolant for any xout value that is outside the range of x is set to NA. Next, linear interpolation is used for any xout value that has only one smaller neighboring x value, or one larger neighboring value. For all other values of xout, the 4 neighboring points x are sought, two smaller and two larger. Then two parabolas are determined, one from the two smaller points plus the nearest larger point, and the other from the nearest smaller point and the two larger points. A weighted sum of these two parabolas provides the interpolated value. Note that, in the notation of Reiniger and Ross (1968), this algorithm uses m=2 and n=1. (A future version of this routine might provide the ability to modify these values.)

Setting method="unesco" yields the method that is used by the U.S. National Oceanographic Data Center. It is described in pages 48-50 of reference 2; reference 3 presumably contains the same information but it is not as easily accessible. The method works as follows.

- If there are data above 5m depth, then the surface value is taken to equal to the shallowest recorded value.

- Distance bounds are put on the four neighboring points, and the Reiniger-Ross method is used for interpolated points with sufficiently four close neighbors. The bounds are described in table 15 of reference 2 only for so-called standard depths; in the present instance they are transformed to the following rules. Inner neighbors must be within 5m for data above 10m, 50m above 250m 100m above 900m, 200m above 2000m, or within 1000m otherwise. Outer neighbors must be within 200m above 500m, 400m above 1300m, or 1000m otherwise. If two or more points meet these criteria, Lagrangian interpolation is used. If not, `NA` is used as the interpolant.

After these rules are applied, the interpolated value is compared with the values immediately above and below it, and if it is outside the range, simple linear interpolation is used.

Value

A vector of interplated values, corresponding to the `xout` values and equal in number.

Author(s)

Dan Kelley

References

1. R.F. Reiniger and C.K. Ross, 1968. A method of interpolation with application to oceanographic data. *Deep Sea Research*, **15**, 185-193.
2. Daphne R. Johnson, Tim P. Boyer, Hernan E. Garcia, Ricardo A. Locarnini, Olga K. Baranova, and Melissa M. Zweng, 2011. World Ocean Database 2009 Documentation. NODC Internal report 20. Ocean Climate Laboratory, National Oceanographic Data Center. Silver Spring, Maryland.
3. UNESCO, 1991. Processing of oceanographic station data, 138 pp., Imprimerie des Presses Universitaires de France, United Nations Educational, Scientific and Cultural Organization, France.

Examples

```
library(oce)
if (require(ocedata)) {
    data(RRprofile)
    zz <- seq(0,2000,5)
    plot(RRprofile$temperature, RRprofile$depth, ylim=c(500,0), xlim=c(2,11))
    ## Contrast two methods
    a1 <- oceApprox(RRprofile$depth, RRprofile$temperature, zz)
    a2 <- oceApprox(RRprofile$depth, RRprofile$temperature, zz, 'rr')
    lines(a1, zz)
    lines(a2, zz, col='red')
    legend("bottomright", lwd=1, col=1:2,
    legend=c("Unesco", "Reiniger-Ross"), cex=3/4)
}
```

`oceColors` *Create a palette of colours*

Description

Create a palette of colours

Usage

```
oceColorsPalette(n, which=1)

oceColorsGebco(n=9, region=c("water", "land", "both"),
type=c("fill","line"))

oceColorsJet(n)

oceColors9A(n)

oceColors9B(n)

oceColorsTwo(n, low=2/3, high=0, smax=1, alpha = 1)
```

Arguments

`n`	the number of colors (≥ 1) to be in the palette.
`which`	an identifier for the palette (see "Details").
`region`	the region characteristic, for `oceColorsGebco`.
`type`	the type of item drawn, for `oceColorsGebco`.
`low`	the hue, in [0,1], for the low end of a `oceColorsTwo` scale.
`high`	the hue, in [0,1], for the high end of a `oceColorsTwo` scale.
`smax`	the maximum saturation, in [0,1], for the colors of `oceColorsTwo`.
`alpha`	the alpha value, in [0,1], for the colors of `oceColorsTwo`.

Details

`oceColorsPalette` provides a variety of pre-defined palettes. `which=1` yields the ColorBrewer diverging red/blue scheme while `which=2` yields the ColorBrewer diverging RYB scheme. (Each is interpolated from the 11-class schemes provided on this site.)

A family of nine-color schemes is as follows: `which="jet"` (or `which="9A"` or `which=9.01` for the Jet scheme; `which="9B"` or `which=9.02` for a scheme similar to Jet but omitting the green, and somewhat desaturating the yellow and cyane.

`oceColorsGebco` provides palettes that mimic the GEBCO altas colours, with shades of blue for water and of brown for land. The blue values go from dark to light, and the brown ones from light

to dark; in this way, topographic images have light values near sea-level, and get darker in either deeper water or higher terrain.

`oceColorsJet` provides a palette similar to the Matlab "jet" palette.

`oceColorsTwo` provides a two-tone palette that fades to white at central values.

Author(s)

Dan Kelley

References

Color Brewer. http://colorbrewer2.org/

Light, A., and P. J. Bartlein, 2004. The End of the Rainbow? Color Schemes for Improved Data Graphics. *Eos Trans. AGU*, 85(40), doi:10.1029/2004EO400002.

Martin Jakobsson, Ron Macnab, and Members of the Editorial Board, IBCAO. Selective comparisons of GEBCO (1979) and IBCAO (2000) maps. http://www.ngdc.noaa.gov/mgg/bathymetry/arctic/ibcao_gebco_comp.html

Stephenson, David B., 2005. Comment on "Color schemes for improved data graphics," by A. Light and P. J. Bartlein. *Eos Trans. AGU*, 86(20). (See also http://geography.uoregon.edu/datagraphics/color_scales.htm)

Examples

```
library(oce)
opar <- par(no.readonly = TRUE)
x <- array(1:1000, dim=c(1,1000))
par(mfrow=c(1,4))
image(x, col=oceColorsTwo(200), main="oceColorsTwo")
image(x, col=oceColorsJet(200), main="oceColorsJet")
image(x, col=oceColorsGebco(200), main="oceColorsGebco")
image(x, col=oceColorsPalette(200), main="oceColorsPalette")
par(opar)

# real-world example, with acoustic-doppler profiler data
data(adp)
par(mfrow=c(3,1))
plot(adp, which='u1')
plot(adp, which='u1', col=oceColorsJet)
plot(adp, which='u1', col=oceColors9B)
```

---

oceContour *Contour with ability to flip x and y*

Description

This provides something analagous to `contour`, but with the ability to flip x and y.

Usage

```
oceContour(x, y, z, revx=FALSE, revy=FALSE, add=FALSE,
           tformat, drawTimeRange=getOption("oceDrawTimeRange"),
           debug=getOption("oceDebug"), ...)
```

Arguments

x	values for x grid.
y	values for y grid.
z	matrix for values to be contoured. The first dimension of z must equal the number of items in x, etc.
revx	set to TRUE to reverse the order in which the labels on the x axis are drawn
revy	set to TRUE to reverse the order in which the labels on the y axis are drawn
add	logical value indicating whether the contours should be added to a pre-existing plot.
tformat	time format; if not supplied, a reasonable choice will be made by oce.axis.POSIXct, which draws time axes.
drawTimeRange	logical, only used if the x axis is a time. If TRUE, then an indication of the time range of the data (not the axis) is indicated at the top-left margin of the graph. This is useful because the labels on time axes only indicate hours if the range is less than a day, etc.
debug	a flag that turns on debugging; set to 1 to information about the processing.
...	optional arguments passed to plotting functions.

Details

Setting revy=TRUE can be helpful if the y data represent pressure or depth below the surface.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(topoWorld)
## coastline now, and in last glacial maximum
lon <- topoWorld[["longitude"]]
lat <- topoWorld[["latitude"]]
z <- topoWorld[["z"]]
oceContour(lon, lat, z, levels=0, drawlabels=FALSE)
oceContour(lon, lat, z, levels=-130, drawlabels=FALSE, col='blue', add=TRUE)
```

`oceConvolve` *Convolve two time series*

Description

Convolve two time series, using a backward-looking method

Usage

```
oceConvolve(x, f, end=2)
```

Arguments

`x`	a numerical vector of observations.
`f`	a numerical vector of filter coefficients.
`end`	a flag that controls how to handle the points of the `x` series that have indices less than the length of `f`. If `end=0`, the values are set to 0. If `end=1`, the original `x` values are used there. If `end=2`, that fraction of the `f` values that overlap with `x` are used.

Details

This function provides a straightforward convolution, which may be useful to those who prefer not to use `convolve` and `filter` in the `stats` package.

Value

A vector of the convolution output.

Author(s)

Dan Kelley

Examples

```
library(oce)
t <- 0:1027
n <- length(t)
signal <- ifelse(sin(t * 2 * pi / 128) > 0, 1, 0)
tau <- 10
filter <- exp(-seq(5*tau, 0) / tau)
filter <- filter / sum(filter)
observation <- oceConvolve(signal, filter)
plot(t, signal, type='l')
lines(t, observation, lty='dotted')
```

oceDebug *Print a debugging message*

Description

Print a debugging message

Usage

```
oceDebug(debug=0, ..., unindent=0)
```

Arguments

debug	an integer, less than or equal to zero for no message, and greater than zero for increasing levels of debugging. Values greater than 4 are treated like 4.
...	items to be supplied to cat, which does the printing. Almost always, this should include a trailing newline.
unindent	Number of levels to un-indent, e.g. for start and end lines from a called function.

Details

Indentation is used, with 8 spaces for debug=4 or higher, 6 for debug=3, 4 for debug=2, and 2 for debug=1. Normally, functions decrease the debug level by 1 when they call other functions, so the effect is a nesting, with more space for deeper function level.

Author(s)

Dan Kelley

Examples

```
foo <- function(debug)
{
   oceDebug(debug, "in function foo\n")
}
debug <- 1
oceDebug(debug, "in main")
foo(debug=debug-1)
```

`oceEdit` *Edit an oce object*

Description

Edit an element of a oce object

Usage

```
oceEdit(x, item, value, action,
        reason="", person="", debug=getOption("oceDebug"))
```

Arguments

`x`	an `oce` object. The exact action of `oceEdit` depends on the `class` of `x`; see "Details".
`item`	if supplied, a character string naming an item in the object's metadata (see "Details").
`value`	new value for item, if both supplied.
`action`	optional character string containing R code to carry out some action on the object.
`reason`	character string giving the reason for the change.
`person`	character string giving the name of person making the change.
`debug`	an integer that specifies a level of debugging, with 0 or less indicating no debugging, and 1 or more indicating debugging.

Details

There are two ways to use this function.

1. If both an `item` and `value` are supplied, then the object's metadata entry named `item` is updated to the supplied `value`.

2. If `item` and `value` are not supplied, then `action` must be supplied. This is a character string specifying some action to be performed on the object, e.g. a manipulation of a column. The action must refer to the object as `x`; see Examples.

3. Applied to an `adv` object (i.e. data from an acoustic velocimeter), `oceEdit` treats items named `heading`, `pitch`, `roll` appropriately, depending on the type of `adv` instrument used. (This is necessary because different manufacturers produce different forms of these items, i.e. Nortek reports them on a time base that is different from the velocity reporting, while Sontek reports them on the same time base.)

In each case, a log entry is stored in the object, to document the change. Indeed, this is the main benefit to using this function, instead of altering the object directly. The log entry will be most useful if it contains a brief note on the `reason` for the change, and the name of the `person` doing the work.

Value

An object of class "oce", altered appropriately, and with a log item indicating the nature of the alteration.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
ctd2 <- oceEdit(ctd, item="latitude", value=47.8879,
                reason="illustration", person="Dan Kelley")
ctd3 <- oceEdit(ctd,action="x@data$pressure<-x@data$pressure-1")
```

---

oceFilter *Filter a time-series, possibly recursively*

Description

Filter a time-series, possibly recursively

Usage

```
oceFilter(x, a=1, b, zero.phase=FALSE)
```

Arguments

x	a vector of numeric values, to be filtered as a time series.
a	a vector of numeric values, giving the a coefficients (see "Details").
b	a vector of numeric values, giving the b coefficients (see "Details").
zero.phase	boolean, set to TRUE to run the filter forwards, and then backwards, thus removing any phase shifts associated with the filter.

Details

The filter is defined as e.g. $y[i] = b[1] * x[i] + b[2] * x[i-1] + b[3] * x[i-2] + ... - a[2] * y[i-1] - a[3] * y[i-2] - a[4] * y[i-3] - ...$, where some of the illustrated terms will be omitted if the lengths of a and b are too small, and terms are dropped at the start of the time series where the index on x would be less than 1.

By contrast with the filter function of R, oceFilter lacks the option to do a circular filter. As a consequence, oceFilter introduces a phase lag. One way to remove this lag is to run the filter forwards and then backwards, as in the "Examples". However, the result is still problematic, in the sense that applying it in the reverse order would yield a different result. (Matlab's filtfilt shares this problem.)

Value

A numeric vector of the filtered results, y, as denoted in "Details".

Note

The first value in the a vector is ignored, and if `length(a)` equals 1, a non-recursive filter results.

Author(s)

Dan Kelley

Examples

```
library(oce)
par(mar=c(4,4,1,1))
b <- rep(1,5)/5
a <- 1
x <- seq(0, 10)
y <- ifelse(x == 5, 1, 0)
f1 <- oceFilter(y, a, b)
plot(x, y, ylim=c(-0, 1.5), pch="o", type='b')
points(x, f1, pch="x", col="red")

# remove the phase lag
f2 <- oceFilter(y, a, b, TRUE)
points(x, f2, pch="+", col="blue")

legend("topleft", col=c("black","red","blue"), pch=c("o","x","+"),
       legend=c("data","normal filter", "zero-phase filter"))
mtext("note that normal filter rolls off at end")
```

---

oceMagic *Find the type of an oceanographic data file*

Description

Find the type of an oceanographic data file.

Usage

```
oceMagic(file, debug=getOption("oceDebug"))
```

Arguments

`file`	a connection or a character string giving the name of the file to be checked.
`debug`	an integer, set non-zero to turn on debugging. Higher values indicate more debugging.

Details

This function tries to infer the file type, based on either the data within the file or, more rarely, based on the file name.

Value

A character string indicating the file type, or `"unknown"`, if the type cannot be determined. If the result contains `"/"` characters, these separate a list describing the file type, with the first element being the general type, the second element being the manufacturer, and the third element being the manufacturer's name for the instrument. For example, `"adp/nortek/aquadopp"` indicates a acoustic-doppler profiler made by NorTek, of the model type called AquaDopp.

Author(s)

Dan Kelley

See Also

This is used mainly by `read.oce`.

`ocePmatch` *Partial matching of strings or numbers*

Description

An extended version of `pmatch` that allows `x` to be numeric or string-based. As with `pmatch`, partial string matches are handled.

Usage

```
ocePmatch(x, table, nomatch=NA_integer_, duplicates.ok=FALSE)
```

Arguments

`x`	a code, or vector of codes. This may be numeric, in which case it is simply returned without further analysis of the other arguments, or it may be string-based, in which case `pmatch` is used to find numeric matches.
`table`	a list that maps strings to numbers; `pmatch` is used on `names(table)`. If the name contains characters that are normally not permitted in a variable name, use quotes, e.g. `list(salinity=1, temperature=2, "salinity+temperature"=3)`.
`nomatch`	value to be returned for cases of no match (passed to `pmatch`.
`duplicates.ok`	code for the handling of duplicates (passed to `pmatch`).

Details

This is a wrapper that is useful mainly for `which` arguments to plotting functions.

Value

A number, or vector of numbers, corresponding to the matches. Non-matches are indicated with NA values, or whatever value is given by the NA argument.

Author(s)

Dan Kelley

See Also

Since pmatch is used for the actual matching, its documentation should be consulted.

Examples

```
library(oce)
ocePmatch(c("s", "at", "te"), list(salinity=1, temperature=3.1))
```

---

oceSmooth *Smooth an oce object*

Description

Smooth an oce object.

Usage

```
oceSmooth(x, ...)
```

Arguments

x	an oce object.
...	parameters to be supplied to smooth, which does the actual work.

Details

Each data element is smoothed as a timeseries. For ADP data, this is done along time, not distance. Time vectors, if any, are not smoothed. A good use of oceSmooth is for despiking noisy data.

Value

An object of class "oce" that has been smoothed appropriately.

Author(s)

Dan Kelley

See Also

The work is done with smooth, and the ... arguments are handed to it directly by oceSmooth.

Examples

```
library(oce)
data(ctd)
d <- oceSmooth(ctd)
plot(d)
```

---

oceSpectrum *Wrapper to give normalized spectrum*

Description

This is a wrapper around the R spectrum function, which returns spectral values that are adjusted so that the integral of those values equals the variance of the input x.

Usage

```
oceSpectrum(x, ...)
```

Arguments

x	As for spectrum, a univariate or multivariate time series.
...	extra arguments passed on to spectrum.

Value

A spectrum that has values that integrate to the variance.

Author(s)

Dan Kelley

See Also

spectrum.

Examples

```
x <- rnorm(1e3)
s <- spectrum(x, plot=FALSE)
ss <- oceSpectrum(x, plot=FALSE)
cat("variance of x=", var(x), "\n")
cat("integral of     spectrum=", sum(s$spec)*diff(s$freq[1:2]), "\n")
cat("integral of oceSpectrum=", sum(ss$spec)*diff(ss$freq[1:2]), "\n")
```

parseLatLon *Parse a latitude or longitude string*

Description

Parse a latitude or longitude string, e.g. as in the header of a CTD file

Usage

```
parseLatLon(line, debug=getOption("oceDebug"))
```

Arguments

line	a character string containing an indication of latitude or longitude.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

The following formats are understood (for, e.g. latitude)

```
* NMEA Latitude = 47 54.760 N
** Latitude:      47 53.27 N
```

Value

A numerical value of latitude or longitude.

Author(s)

Dan Kelley

See Also

Used by read.ctd.

plot.adp *Plot ADP data*

Description

Plot ADP data

Usage

```
## S4 method for signature 'adp'
plot(x, which=1:dim(x@data$v)[3],
     mode=c("normal", "diagnostic"),
     col, breaks,
     zlim,
     titles,
     lwd=par('lwd'),
     type='l',
     ytype=c("profile", "distance"),
     adorn=NULL,
     drawTimeRange=getOption("oceDrawTimeRange"),
     useSmoothScatter,
     missingColor="gray",
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1.5,mgp[1]+1.5,1.5,1.5),
     mai.palette=rep(0, 4),
     tformat,
     marginsAsImage=FALSE,
     cex=par("cex"), cex.axis=par("cex.axis"), cex.main=par("cex.main"),
     xlim, ylim,
     control,
     useLayout=FALSE,
     coastline="coastlineWorld",
     main="",
     grid=FALSE, grid.col="darkgray", grid.lty="dotted", grid.lwd=1,
     debug=getOption("oceDebug"),
     ...)
```

Arguments

x	an adp object, e.g. as read by read.adp.
which	list of desired plot types. These are graphed in panels running down from the top of the page. See "Details" for the meanings of various values of which.
mode	a string indicating whether to plot the conventional signal (normal) or or, in the special case of Aquadopp single-bin profilers, possibly the diagnostic signal. This argument is ignored except in the case of Aquadopp instruments. Perhaps a third option will become available in the future, for the burst mode that some instruments provide.

col	optional indication of colour(s) to use. If not provided, the default for images is oceColorsPalette(128,1), and for lines and points is black.
breaks	optional breaks for color scheme
zlim	a range to be used as the zlim parameter to the imagep call that is used to create the image. If omitted, zlim is set for each panel individually, to encompass the data of the panel and to be centred around zero. If provided as a two-element vector, then that is used for each panel. If provided as a two-column matrix, then each panel of the graph uses the corresponding row of the matrix; for example, setting zlim=rbind(c(-1,1),c(-1,1),c(-.1,.1)) might make sense for which=1:3, so that the two horizontal velocities have one scale, and the smaller vertical velocity has another.
titles	optional vector of character strings to be used as labels for the plot panels. For images, these strings will be placed in the right hand side of the top margin. For timeseries, these strings are ignored. If this is provided, its length must equal that of which.
lwd	if the plot is of a time-series or scattergraph format with lines, this is used in the usual way; otherwise, e.g. for image formats, this is ignored.
type	if the plot is of a time-series or scattergraph format, this is used in the usual way, e.g. "l" for lines, etc.; otherwise, as for image formats, this is ignored.
ytype	character string controlling the type of the y axis for images (ignored for time series). If "distance", then the y axis will be distance from the sensor head, with smaller distances nearer the bottom of the graph. If "profile", then this will still be true for upward-looking instruments, but the y axis will be flipped for downward-looking instruments, so that in either case, the top of the graph will represent the sample nearest the sea surface.
adorn	list of expressions to be executed for the panels in turn, e.g. to adorn the plots. If the number matches the number of panels, then the strings are applied to the appropriate panels, as they are drawn from top-left to bottom-right. If only a single expression is provided, it is used for all panels. (See "Examples".)
drawTimeRange	boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.
useSmoothScatter	boolean that indicates whether to use smoothScatter in various plots, such as which="uv". If not provided a default is used, with smoothScatter being used if there are more than 2000 points to plot.
missingColor	colour used to indicate NA values in images (see imagep); set to NULL to avoid this indication.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
mai.palette	margins, in inches, to be added to those calculated for the palette; alter from the default only with caution
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)

marginsAsImage	boolean, TRUE to put a wide margin to the right of time-series plots, even if there are no images in the which list. (The margin is made wide if there are some images in the sequence.)
cex	size of labels on axes; see par("cex").
cex.axis	see par("cex.axis").
cex.main	see par("cex.main").
xlim	optional 2-element list for xlim, or 2-column matrix, in which case the rows are used, in order, for the panels of the graph.
ylim	optional 2-element list for ylim, or 2-column matrix, in which case the rows are used, in order, for the panels of the graph.
control	optional list of parameters that may be used for different plot types. Possibilities are drawBottom (a boolean that indicates whether to draw the bottom) and bin (a numeric giving the index of the bin on which to act, as explained in "Details").
useLayout	set to FALSE to prevent using layout to set up the plot. This is needed if the call is to be part of a sequence set up by e.g. par(mfrow).
coastline	a coastline object, or a character string naming one. This is used only for which="map". See notes at plot.ctd for more information on built-in coastlines.
main	main title for plot, used just on the top panel, if there are several panels.
grid	if TRUE, a grid will be drawn for each panel. (This argument is needed, because calling grid after doing a sequence of plots will not result in useful results for the individual panels.
grid.col	color of grid
grid.lty	line type of grid
grid.lwd	line width of grid
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions. For example, supplying despike=TRUE will cause time-series panels to be de-spiked with despike. Another common action is to set the colour for missing values on image plots, with the argument missingColor (see imagep). Note that it is an error to give breaks in ..., if the formal argument zlim was also given, because they could contradict each other.

Details

Creates a summary plot of data measured by an acoustic doppler profiler. This may have one or more panels, with the content being controlled by the which argument.

- which=1:4 (or which="u1" to "u4") yield a distance-time image plot of a velocity component. If x is in beam coordinates (signalled by x@metadata$oce.coordinate=="beam"), this will be the beam velocity, labelled b[1] etc. If x is in xyz coordinates (sometimes called frame coordinates, or ship coordinates), it will be the velocity component to the right of the frame or ship (labelled u etc). Finally, if x is in "enu" coordinates, the image will show the the eastward component (labelled east). If x is in "other" coordinates, it will be component corresponding to east, after rotation (labelled u\'). Note that the coordinate is set by read.adp, or by beamToXyzAdp, xyzToEnuAdp, or enuToOtherAdp.

- which=5:8 (or which="a1" to "a4") yield distance-time images of backscatter intensity of the respective beams. (For data derived from Teledyn-RDI instruments, this is the item called "echo intensity.")
- which=9:12 (or which="q1" to "q4") yield distance-time images of signal quality for the respective beams. (For RDI data derived from instruments, this is the item called "correlation magnitude.")
- which=70:73 (or which="g1" to "g4") yield distance-time images of percent-good for the respective beams. (For data derived from Teledyne-RDI instruments, which are the only instruments that yield this item, it is called "percent good.")
- which=13 (or which="salinity") yields a time-series plot of salinity.
- which=14 (or which="temperature") yields a time-series plot of temperature.
- which=15 (or which="pressure") yields a time-series plot of pressure.
- which=16 (or which="heading") yields a time-series plot of instrument heading.
- which=17 (or which="pitch") yields a time-series plot of instrument pitch.
- which=18 (or which="roll") yields a time-series plot of instrument roll.
- which=19 yields a time-series plot of distance-averaged velocity for beam 1, rightward velocity, eastward velocity, or rotated-eastward velocity, depending on the coordinate system.
- which=20 yields a time-series of distance-averaged velocity for beam 2, foreward velocity, northward velocity, or rotated-northward velocity, depending on the coordinate system.
- which=21 yields a time-series of distance-averaged velocity for beam 3, up-frame velocity, upward velocity, or rotated-upward velocity, depending on the coordinate system.
- which=22 yields a time-series of distance-averaged velocity for beam 4, for beam coordinates, or velocity estimate, for other coordinates. (This is ignored for 3-beam data.)
- which=23 yields a progressive-vector diagram in the horizontal plane, plotted with asp=1. Normally, the depth-averaged velocity components are used, but if the control list contains an item named bin, then the depth bin will be used (with an error resulting if the bin is out of range).
- which=24 yields a time-averaged profile of the first component of velocity (see which=19 for the meaning of the component, in various coordinate systems).
- which=25 as for 24, but the second component.
- which=26 as for 24, but the third component.
- which=27 as for 24, but the fourth component (if that makes sense, for the given instrument).
- which=28 or "uv" yields velocity plot in the horizontal plane, i.e. u[2] versus u[1]. If the number of data points is small, a scattergraph is used, but if it is large, smoothScatter is used.
- which=29 or "uv+ellipse" as the "uv" case, but with an added indication of the tidal ellipse, calculated from the eigen vectors of the covariance matrix.
- which=30 or "uv+ellipse+arrow" as the "uv+ellipse" case, but with an added arrow indicating the mean current.
- which=40 or "bottom range" for average bottom range from all beams of the instrument.
- which=41 to 44 (or "bottom range1" to "bottom range1") for bottom range from beams 1 to 4.

- which=50 or "bottom velocity" for average bottom velocity from all beams of the instrument.
- which=51 to 54 (or "bottom velocity1" to "bottom velocity4") for bottom velocity from beams 1 to 4.
- which=55 (or "heaving") for time-integrated, depth-averaged, vertical velocity, i.e. a time series of heaving.
- which=100 (or "soundSpeed") for a time series of sound speed.

In addition to the above, there are some groupings defined:

- which="velocity" equivalent to which=1:3 (velocity components)
- which="amplitude" equivalent to which=5:7 (backscatter intensity components)
- which="quality" equivalent to which=9:11 (quality components)
- which="hydrography" equivalent to which=14:15 (temperature and pressure)
- which="angles" equivalent to which=16:18 (heading, pitch and roll)

The color scheme for image plots (which in 1:12) is provided by the col argument, which is passed to image to do the actual plotting. See "Examples" for some comparisons.

A common quick-look plot to assess mooring movement is to use which=15:18 (pressure being included to signal the tide, and tidal currents may dislodge a mooring or cause it to settle).

By default, plot.adp uses a zlim value for the image that is constructed to contain all the data, but to be symmetric about zero. This is done on a per-panel basis, and the scale is plotted at the top-right corner, along with the name of the variable being plotted. You may also supply zlim as one of the ... arguments, but be aware that a reasonable limit on horizontal velocity components is unlikely to be of much use for the vertical component.

A good first step in the analysis of measurements made from a moored device (stored in d, say) is to do plot(d, which=14:18). This shows time series of water properties and sensor orientation, which is helpful in deciding which data to trim at the start and end of the deployment, because they were measured on the dock or on the ship as it travelled to the mooring site.

Author(s)

Dan Kelley

See Also

The documentation for adp-class explains the structure of ADP objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(adp)
plot(adp, which=1:3)
plot(adp, which=5, missingColor='gray',
adorn=expression({
    lines(x[["time"]], x[["pressure"]], lwd=3, col='blue')
    }))
plot(adp, which='temperature', tformat='%H:%M')
```

plot.adv *Plot ADV data*

Description

Plot ADV data

Usage

```
## S4 method for signature 'adv'
plot(x,
    which=c(1:3,14,15),
    col,
    titles,
    type="l",
    lwd=par('lwd'),
    adorn=NULL,
    drawTimeRange=getOption("oceDrawTimeRange"),
    drawZeroLine=FALSE,
    useSmoothScatter,
    mgp=getOption("oceMgp"),
    mar=c(mgp[1]+1.5,mgp[1]+1.5,1.5,1.5),
    tformat,
    marginsAsImage=FALSE,
    cex=par("cex"), cex.axis=par("cex.axis"), cex.main=par("cex.main"),
    xlim, ylim,
    brushCorrelation, colBrush="red",
    main="",
    debug=getOption("oceDebug"),
    ...)
```

Arguments

x	an adv object, e.g. as read by read.adv.
which	list of desired plot types. These are graphed in panels running down from the top of the page. See "Details" for the meanings of various values of which.
col	optional indication of colour(s) to use. If not provided, the default for images is oceColorsPalette(128,1), and for lines and points is black.
titles	optional vector of character strings to be used as labels for the plot panels. For images, these strings will be placed in the right hand side of the top margin. For timeseries, these strings are ignored. If this is provided, its length must equal that of which.
lwd	if the plot is of a time-series or scattergraph format with lines, this is used in the usual way; otherwise, e.g. for image formats, this is ignored.
type	type of plot, as for plot.

adorn	optional list of expressions to be performed immediately after drawing the panels. (See plot.adp for an example.)
drawTimeRange	boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.
drawZeroLine	boolean that indicates whether to draw zero lines on velocities.
useSmoothScatter	boolean that indicates whether to use smoothScatter in various plots, such as which="uv". If not provided a default is used, with smoothScatter being used if there are more than 2000 points to plot.,
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
marginsAsImage	boolean, TRUE to put a wide margin to the right of time-series plots, matching the space used up by a palette in an imagep plot.
cex	size of labels on axes; see par("cex").
cex.axis	see par("cex.axis").
cex.main	see par("cex.main").
xlim	optional 2-element list for xlim, or 2-column matrix, in which case the rows are used, in order, for the panels of the graph.
ylim	optional 2-element list for ylim, or 2-column matrix, in which case the rows are used, in order, for the panels of the graph.
brushCorrelation	optional number between 0 and 100, indicating a per-beam correlation threshhold below which data are to be considered suspect. If the plot type is p, the suspect points (velocity, backscatter amplitude, or correlation) will be coloured red; otherwise, this argument is ignored.
colBrush	colour to use for brushed (bad) data, if brushCorrelation is active.
main	main title for plot, used just on the top panel, if there are several panels.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions.

Details

Creates a multi-panel summary plot of data measured by an ADV. The panels are controlled by the which argument. (Note the gaps in the sequence, e.g. 4 and 8 are not used.)

- which=1 to 3 (or "u1" to "u3")
  yield timeseries of the first, second, and third components of velocity (in beam, xyz or enu coordinates).
- which=4 is not permitted (since ADV are 3-beam devices)

- which=5 to 7 (or "a1" to "a3") yield timeseries of the amplitudes of beams 1 to 3. (Note that the data are called data$a[,1], data$a[,2] and data$a[,3], for these three timeseries.)
- which=8 is not permitted (since ADV are 3-beam devices)
- which=9 to 11 (or "q1" to "q3") yield timeseries of correlation for beams 1 to 3. (Note that the data are called data$c[,1], data$c[,2] and data$c[,3], for these three timeseries.)
- which=12 is not permitted (since ADVs are 3-beam devices)
- which=13 is not permitted (since ADVs do not measure salinity)
- which=14 or which="temperature" yields a timeseries of temperature.
- which=15 or which="pressure" yields a timeseries of pressure.
- which=16 or which="heading" yields a timeseries of heading.
- which=17 or which="pitch"yields a timeseries of pitch.
- which=18 or which="roll"yields a timeseries of roll.
- which=19 to 21 yields plots of correlation versus amplitude, for beams 1 through 3, using smoothScatter.
- which=22 is not permitted (since ADVs are 3-beam devices)
- which=23 or "progressive vector" yields a progressive-vector diagram in the horizontal plane, plotted with asp=1, and taking beam1 and beam2 as the eastward and northward components of velocity, respectively.
- which=28 or "uv" yields velocity plot in the horizontal plane, i.e. u[2] versus u[1]. If the number of data points is small, a scattergraph is used, but if it is large, smoothScatter is used.
- which=29 or "uv+ellipse" as the "uv" case, but with an added indication of the tidal ellipse, calculated from the eigen vectors of the covariance matrix.
- which=30 or "uv+ellipse+arrow" as the "uv+ellipse" case, but with an added arrow indicating the mean current.
- which=50 or "analog1" plots a time series of the analog1 signal, if there is one.
- which=51 or "analog2" plots a time series of the analog2 signal, if there is one.
- which=100 or "voltage" plots the voltage as a timeseries, if voltage exists in the dataset.

In addition to the above, there are some groupings defined:

- which="velocity" equivalent to which=1:3 (three velocity components)
- which="amplitude" equivalent to which=5:7 (three amplitude components)
- which="backscatter" equivalent to which=9:11 (three backscatter components)
- which="hydrography" equivalent to which=14:15 (temperature and pressure)
- which="angles" equivalent to which=16:18 (heading, pitch and roll)

Author(s)

Dan Kelley

See Also

The documentation for adv-class explains the structure of ADV objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(adv)
plot(adv)
```

---

plot.cm *Plot cm (current meter) data*

Description

Plot cm (current meter) data

Usage

```
## S4 method for signature 'cm'
plot(x,
     which=c(1, 2, 7, 9),
     type="l",
     adorn=NULL,
     drawTimeRange=getOption("oceDrawTimeRange"),
     drawZeroLine=FALSE,
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1.5,mgp[1]+1.5,1.5,1.5),
     small=2000,
     main="",
     tformat,
     debug=getOption("oceDebug"),
     ...)
```

Arguments

x	an cm object, e.g. as read by read.cm.
which	list of desired plot types. These are graphed in panels running down from the top of the page. See "Details" for the meanings of various values of which.
type	type of plot, as for plot.
adorn	optional list of expressions to be performed immediately after drawing the panels. (See plot.adp for an example.)
drawTimeRange	boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.
drawZeroLine	boolean that indicates whether to draw zero lines on velocities.

mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
small	an integer indicating the size of data set to be considered "small", to be plotted with points or lines using the standard plot function. Data sets with more than small points will be plotted with smoothScatter instead.
main	main title for plot, used just on the top panel, if there are several panels.
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions.

Details

Creates a multi-panel summary plot of data measured by a current meter. The panels are controlled by the which argument, as follows.

- which=1 or which="u" for a time-series graph of eastward velocity, u, as a function of time.
- which=2 or which="v" for a time-series graph of northward velocity, u, as a function of time.
- which=3 or "progressive vector" for progressive-vector plot
- which=4 or "uv" for a plot of v versus u. (Dots are used for small datasets, and smoothScatter for large ones.)
- which=5 or "uv+ellipse" as the "uv" case, but with an added indication of the tidal ellipse, calculated from the eigen vectors of the covariance matrix.
- which=6 or "uv+ellipse+arrow" as the "uv+ellipse" case, but with an added arrow indicating the mean current.
- which=7 or "depth" for depth
- which=8 or "salinity" for salinity
- which=9 or "temperature" for temperature
- which=10 or "heading" for heading
- which=11 or "TS" for a TS diagram

Author(s)

Dan Kelley

See Also

The documentation for cm-class explains the structure of CM objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(cm)
summary(cm)
plot(cm)
```

`plot.coastline` *Plot a coastline*

Description

Plot a coastline

Usage

```
## S4 method for signature 'coastline'
plot(x,
     xlab="", ylab="", showHemi=TRUE,
     asp,
     clongitude, clatitude, span,
     projection=NULL, parameters=NULL, orientation=NULL,
     expand=1,
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1,mgp[1]+1,1,1),
     bg,
     fill='lightgray',
     axes=TRUE, cex.axis=par('cex.axis'),
     add=FALSE, inset=FALSE,
     geographical=0,
     debug=getOption("oceDebug"),
     ...)
```

Arguments

`x`	A `coastline` object, as read by `read.coastline` or created by `as.coastline`, or a list containing items named `longitude` and `latitude`.
`xlab`	label for x axis
`ylab`	label for y axis
`showHemi`	logical indicating whether to show the hemisphere in axis tick labels.
`asp`	Aspect ratio for plot. The default is for `plot.coastline` to set the aspect ratio to give natural latitude-longitude scaling somewhere near the centre latitude on the plot. Often, it makes sense to set `asp` yourself, e.g. to get correct shapes at 45N, use `asp=1/cos(45*pi/180)`. Note that the land mass is not symmetric about the equator, so to get good world views you should set `asp=1` or set `ylim` to be symmetric about zero. Any given value of `asp` is ignored, if `clongitude` and `clatitude` are given.

clongitude,clatitude
: optional center latitude of map, in decimal degrees. If both clongitude and clatitude are provided, then any provided value of asp is ignored, and instead the plot aspect ratio is computed based on the center latitude. If clongitude and clatitude are provided, then span must also be provided.

span
: optional suggested span of plot, in kilometers. The suggestion is an upper limit on the scale; depending on the aspect ratio of the plotting device, the radius may be smaller than span. A value for span must be supplied, if clongitude and clatitude are supplied.

projection
: optional map projection to use (see mapPlot); if not given, a cartesian frame is used, scaled so that coastline shapes near the centre of the plot are preserved. If a projection is provided, the coordinate system will bear an indirect relationship to longitude and longitude, and further adornment of the plot must be done with e.g. mapPoints instead of points.

parameters
: optional parameters to map projection (see mapPlot.

orientation
: optional orientation of map projection (see mapPlot.

expand
: numerical factor for the expansion of plot limits, showing area outside the plot, e.g. if showing a ship track as a coastline, and then an actual coastline to show the ocean boundary. The value of expand is ignored if either xlim or ylim is given.

mgp
: 3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.

mar
: value to be used with par("mar").

bg
: optional colour to be used for the background of the map. This comes in handy for drawing insets (see "details").

fill
: colour to be used to fill land regions. This is ignored unless the coastline object is fillable (e.g. for a "shapefile" read by read.coastline). Set to NULL to turn off filling.

axes
: boolean, set to TRUE to plot axes.

cex.axis
: value for axis font size factor.

add
: boolean, set to TRUE to draw the coastline on an existing plot. Note that this retains the aspect ratio of that existing plot, so it is important to set that correctly, e.g. with asp=1/cos(lat * pi / 180), where clat is the central latitude of the plot.

inset
: set to TRUE for use within plotInset. The effect is to prevent the present function from adjusting margins, which is necessary because margin adjustment is the basis for the method used by plotInset.

geographical
: flag indicating the style of axes. If geographical=0, the axes are conventional, with decimal degrees as the unit, and negative signs indicating the southern and western hemispheres. If geographical=1, the signs are dropped, with axis values being in decreasing order within the southern and western hemispheres. If geographical=2, the signs are dropped and the axes are labelled with degrees, minutes and seconds, as appropriate.

debug	set to TRUE to get debugging information during processing.
...	optional arguments passed to plotting functions. For example, set yaxp=c(-90,90,4) for a plot extending from pole to pole.

Details

This function plots a coastline. An attempt is made to fill the space of the plot, and this is done by limiting either the longitude range or the latitude range, as appropriate, by modifying the eastern or northern limit, as appropriate.

To get an inset map inside another map, draw the first map, do par(new=TRUE), and then call plot.coastline with a value of mar that moves the inset plot to a desired location on the existing plot, and with bg="white".

Value

None.

Author(s)

Dan Kelley

See Also

The documentation for coastline-class explains the structure of coastline objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(coastlineWorld)
plot(coastlineWorld)
plot(coastlineWorld, clongitude=-63.6, clatitude=44.6, span=1000)

## Canada in Lambert projection
plot(coastlineWorld, clongitude=-90, clatitude=60, span=3000,
     fill=NULL, grid=10,
     projection='lambert', parameters=c(lat0=50,lat1=70),
     orientation=c(90,-90,0))

## End(Not run)
```

plot.ctd *Plot seawater CTD data*

Description

Plot CTD data, by default in a four-panel display showing (a) profiles of salinity and temperature, (b) profiles of density and the square of buoyancy frequency, (c) a TS diagram and (d) a coastline diagram indicating the station location.

Usage

```
## S4 method for signature 'ctd'
plot(x, which=c(1, 2, 3, 5), col=par("fg"),
     eos=getOption("eos", default='unesco'),
     ref.lat=NaN, ref.lon=NaN,
     grid=TRUE,
     coastline="best",
     Slim, Tlim, plim, densitylim, N2lim, Rrholim,
     dpdtlim, timelim,
     lonlim, latlim,
     clongitude, clatitude, span, showHemi=TRUE,
     projection=NULL, parameters=NULL, orientation=NULL,
     latlon.pch=20, latlon.cex=1.5, latlon.col="red",
     cex=1, cex.axis=par('cex.axis'),
     pch=1,
     useSmoothScatter=FALSE,
     df,
     keepNA=FALSE,
     type='l',
     adorn=NULL,
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1.5,mgp[1]+1.5,mgp[1]+1.5,mgp[1]+1),
     inset=FALSE,
     add=FALSE,
     debug=getOption("oceDebug"),
     ...)
```

Arguments

x	A ctd object, e.g. as read by read.ctd, or a list containing items named salinity and temperature.
which	list of desired plot types.

- which=1 or which="salinity+temperature" gives a combined profile of temperature and salinity
- which=2 or which="density+N2" gives a combined profile of σ_θ and N^2
- which=3 or which="TS" gives a TS plot

- which=4 or which="text" gives a textual summary of some aspects of the data
- which=5 or which="map" gives a map, with a dot for the station location. Notes near the top boundary of the map give the station number, the sampling date, and the name of the chief scientist, if these are known.
- which=5.1 as for which=5, except that the file name is drawn above the map
- which=6 or which="density+dpdt" gives a profile of density and dP/dt, which is useful for evaluating whether the instrument is dropping properly through the water column
- which=7 or which="density+time" gives a profile of density and time
- which=8 or which="index" gives a profile of index number (especially useful for ctdTrim)
- which=9 or which="salinity" gives a salinity profile
- which=10 or which="temperature" gives a temperature profile
- which=11 or which="density" gives a density profile
- which=12 or which="N2" gives an N^2 profile
- which=13 or which="spice" gives a spiciness profile
- which=14 or which="tritium" gives a tritium profile

col — colour of lines or symbols

eos — either "unesco" or "teos". If the latter, then the computer must have the TEOS library installed; see teos. The effect of using "teos" is to us "absolute salinity" on the x axis and "conservative temperature" on the y axis. Typically, the temperature values will be very similar to those with unesco, but the salinities will be increased by 0.1 to 0.2 units. The isopycnals will still run through the same points as for the unesco style. See teos for more information.

ref.lat — Latitude of reference point for distance calculation

ref.lon — Longitude of reference point for distance calculation

grid — Set TRUE to get a grid on all plots.

coastline — a specification of the coastline to be used for which="map". This may be a coastline object, whether built-in or supplied by the user, or a character string. If the later, it may be the name of a built-in coastline ("coastlineWorld", "coastlineWorldFine", or "coastlineWorldCoarse"), or "best", to choose a suitable coastline for the locale, or "none" to prevent the drawing of a coastline. There is a speed penalty for providing coastline as a character string, because it forces plot.coastline to load it on every call. So, if plot.coastline is to be called several times for a given coastline, it makes sense to load it in before the first call, and to supply the object as an argument, as opposed to the name of the object.

Slim — optional limits of salinity axes

Tlim — optional limits of temperature axes

plim — optional limits of pressure axes

densitylim — optional limits of density axis

N2lim — optional limits of N^2 axis

Rrholim	optional limits of R_rho axis
dpdtlim	optional limits of dP/dt axis
timelim	optional limits of delta-time axis
lonlim	optional limits of longitude axis of map (ignored if no map plotted) DEPRECATED 2014-01-07
latlim	optional limits of latitude axis of map (ignored if no map plotted) DEPRECATED 2014-01-07
clongitude	center longitude
clatitude	center latitude
span	optional span of map, in km. If not given, the map scale is chosen to include some points on the nearby coastline.
showHemi	logical indicating whether to show hemisphere in axis tick labels.
projection	projection for map, passed to mapproject by plot with a coastline object; NULL for none.
parameters	parameters for map, as for projection.
orientation	orientation for map, as for projection.
latlon.pch	pch for sample location (ignored if no map plotted)
latlon.cex	cex for sample location (ignored if no map plotted)
latlon.col	col for sample location (ignored if no map plotted)
cex	size to be used for plot symbols (see par)
cex.axis	size factor for axis labels (see par)
pch	code for plotting symbol (see par).
useSmoothScatter	boolean, set to TRUE to use smoothScatter instead of plot to draw the plot.
df	optional argument that is ignored except for plotting buoyancy frequency; in that case, it is passed to swN2 as the argument named df.
keepNA	flag indicating whether to keep NA values in linegraphs, which will yield breaks in the lines.
type	type of plot to draw, using the same scheme as plot.
adorn	list of expressions to be executed for the panels in turn, e.g. to adorn the plots. If the number matches the number of panels, then the strings are applied to the appropriate panels, as they are drawn from top-left to bottom-right. If only a single expression is provided, it is used for all panels. (See "Examples".)
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
inset	set to TRUE for use within plotInset. The effect is to prevent the present function from adjusting margins, which is necessary because margin adjustment is the basis for the method used by plotInset.

add	add to an existing plot. This only works if `length(which)=1`, and it will yield odd results if the value of `which` does not match that in the previous plots.
debug	set to a positive value to get debugging information during processing.
...	optional arguments passed to plotting functions. A common example is to set `df`, for use in swN2 calculations.

Details

Creates a multi-panel summary plot of data measured in a CTD cast. The panels are controlled by the `which` argument. Normally, 4 panels are specified with the `which`, but it can also be useful to specify less than 4 panels, and then to draw other panels after this call.

If only 2 panels are requested, they will be drawn side by side.

If more than one panel is drawn, then on exit from `plot.ctd`, the value of `par` will be reset to the value it had before the function call. However, if only one panel is drawn, the adjustments to `par` made within `plot.ctd` are left in place, so that further additions may be made to the plot.

Author(s)

Dan Kelley

See Also

The documentation for `ctd-class` explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(ctd)
plot(ctd)
```

---

plot.drifter *Plot drifter data*

Description

Plot a summary diagram for drifter data.

Usage

```
## S4 method for signature 'drifter'
plot(x, which = 1, level,
     coastline=c("best", "coastlineWorld", "coastlineWorldMedium",
                 "coastlineWorldFine", "none"),
     cex=1, pch=1, type='p', col,
     adorn=NULL,
     mgp=getOption("oceMgp"),
```

```
        mar=c(mgp[1]+1.5, mgp[1]+1.5, 1.5, 1.5),
        tformat,
        debug=getOption("oceDebug"), ...)
```

Arguments

x	A drifter object, e.g. as read by read.drifter.
which	list of desired plot types. • which=1 or which="trajectory" gives a plot of the drifter trajectory, with the coastline, if one is provided. • which=2 or "salinity ts" gives a time series of salinity at the indicated level(s) • which=3 or "temperature ts" gives a time series of temperature at the indicated level(s) • which=4 or "TS" gives a TS diagram at the indicated level(s) • which=5 or "salinity profile" gives a salinity profile of all the data (with S and p trimmed to the 1 and 99 percentiles) • which=6 or "temperature profile" gives a temperature profile (with T and p trimmed to the 1 and 99 percentiles)
level	level to plot, for e.g. which=2 and higher. May be an integer, in which case it refers to an index of depth (1 being the top) or it may be the string "all" which means to plot all data.
coastline	string giving the coastline to be used in a drifter-location map, or "best" to pick the one with highest resolution, or "none" to avoid drawing the coastline.
cex	size of plotting symbols to be used if type='p'.
pch	type of plotting symbols to be used if type='p'.
type	plot type, either "l" or "p".
col	optional list of colours for plotting.
adorn	list of expressions to be executed for the panels in turn, e.g. to adorn the plots. If the number matches the number of panels, then the strings are applied to the appropriate panels, as they are drawn from top-left to bottom-right. If only a single expression is provided, it is used for all panels. (See "Examples".)
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
debug	debugging flag
...	optional arguments passed to plotting functions.

Details

Creates a summary plot for a drifter dataset.

Value

None.

Author(s)

Dan Kelley

References

http://www.argo.ucsd.edu/

See Also

The documentation for drifter-class explains the structure of drifter objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(drifter)
plot(drifter, which="trajectory")
```

---

plot.echosounder *Plot echosounder data*

Description

Plot echosounder data

Usage

```
## S4 method for signature 'echosounder'
plot(x,
     which=1, beam="a",
     newx,
     xlab, ylab,
     xlim, ylim, zlim,
     type="l", col=oceColorsJet, lwd=2,
     despike=FALSE,
     drawBottom, ignore=5,
     drawTimeRange=FALSE,
     radius, coastline,
     adorn=NULL,
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1,mgp[1]+1,mgp[1]+1,mgp[1]+1),
     atTop, labelsTop,
     tformat,
     debug=getOption("oceDebug"),
     ...)
```

Arguments

x	An echosounder object, e.g. as read by read.echosounder, or created by as.echosounder.
which	list of desired plot types: which=1 or which="zt image" gives a z-time image, which=2 or which="zx image" gives a z-distance image, and which=3 or which="map" gives a map showing the cruise track. In the image plots, the display is of log10 of amplitude, trimmed to zero for any amplitude values less than 1 (including missing values, which equal 0). Add 10 to the numeric codes to get the secondary data (non-existent for single-beam files,
beam	a more detailed specification of the data to be plotted. For single-beam data, this may only be "a". For dual-beam data, this may be "a" for the narrow-beam signal, or "b" for the wide-beam signal. For split-beam data, this may be "a" for amplitude, "b" for x-angle data, or "c" for y-angle data.
newx	optional vector of values to appear on the horizontal axis if which=1, instead of time. This must be of the same length as the time vector, because the image is remapped from time to newx using approx.
xlab, ylab	optional labels for the horizontal and vertical axes; if not provided, the labels depend on the value of which.
xlim	optional range for x axis.
ylim	optional range for y axis.
zlim	optional range for colorscale.
type	type of graph, "l" for line, "p" for points, or "b" for both.
col	colorscale for image, a function
lwd	line width (ignored if type="p")
atTop	optional vector of time values, for labels at the top of the plot produced with which=2. If labelsTop is provided, then it will hold the labels. If labelsTop is not provided, the labels will be constructed with the format function, and these may be customized by supplying a format in the ... arguments.
labelsTop	optional vector of character strings to be plotted above the atTop times. Ignored unless atTop was provided.
tformat	optional argument passed to imagep, for plot types that call that function. (See strptime for the format used.)
despike	remove vertical banding by using smooth to smooth across image columns, row by row.
drawBottom	optional flag used for section images. If TRUE, then the bottom is inferred as a smoothed version of the ridge of highest image value, and data below that are grayed out after the image is drawn. If drawBottom is a colour, then that colour is used, instead of white. The bottom is detected with findBottom, using the ignore value described next.
ignore	optional flag specifying the thickness in metres of a surface region to be ignored during the bottom-detection process. This is ignored unless drawBottom=TRUE.

drawTimeRange	if TRUE, the time range will be drawn at the top. Ignored except for which=2, i.e. distance-depth plots.
radius	radius to use for maps; ignored unless which=3 or which="map".
coastline	coastline to use for maps; ignored unless which=3 or which="map".
adorn	list of expressions to be executed for the panels in turn, e.g. to adorn the plots. If the number matches the number of panels, then the strings are applied to the appropriate panels, as they are drawn from top-left to bottom-right. If only a single expression is provided, it is used for all panels. (See "Examples".)
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
debug	set to an integer exceeding zero, to get debugging information during processing.
...	optional arguments passed to plotting functions. For example, for maps, it is possible to specify the radius of the view in kilometres, with radius.

Details

Simple linear approximation is used when a newx value is specifie with the which=2 method, but arguably a gridding method should be used, and this may be added in the future.

Author(s)

Dan Kelley, with extensive help from Clark Richards

See Also

The documentation for echosounder-class explains the structure of echosounder objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(echosounder)
plot(echosounder, which=c(1,2), drawBottom=TRUE)
```

---

plot.gps *Plot a gps object*

Description

Plot a gps object

Usage

```
## S4 method for signature 'gps'
plot(x,
     xlab="", ylab="",
     asp,
     clongitude, clatitude, span,
     projection, parameters=NULL, orientation=NULL,
     expand=1,
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1,mgp[1]+1,1,1),
     bg,
     axes=TRUE, cex.axis=par('cex.axis'),
     add=FALSE, inset=FALSE,
     geographical=0,
     debug=getOption("oceDebug"),
     ...)
```

Arguments

x	A gps object, as read by read.gps or created by as.gps, or a list containing items named longitude and latitude.
xlab	label for x axis
ylab	label for y axis
asp	Aspect ratio for plot. The default is for plot.gps to set the aspect ratio to give natural latitude-longitude scaling somewhere near the centre latitude on the plot. Often, it makes sense to set asp yourself, e.g. to get correct shapes at 45N, use asp=1/cos(45*pi/180). Note that the land mass is not symmetric about the equator, so to get good world views you should set asp=1 or set ylim to be symmetric about zero. Any given value of asp is ignored, if clongitude and clatitude are given.
clongitude,clatitude	optional center latitude of map, in decimal degrees. If both clongitude and clatitude are provided, then any provided value of asp is ignored, and instead the plot aspect ratio is computed based on the center latitude. If clongitude and clatitude are provided, then span must also be provided.
span	optional suggested span of plot, in kilometers. The suggestion is an upper limit on the scale; depending on the aspect ratio of the plotting device, the radius may be smaller than span. A value for span must be supplied, if clongitude and clatitude are supplied.
projection	optional map projection to use (see mapPlot); if not given, a cartesian frame is used, scaled so that gps shapes near the centre of the plot are preserved. If a projection is provided, the coordinate system will bear an indirect relationship to longitude and longitude, and further adornment of the plot must be done with e.g. mapPoints instead of points.
parameters	optional parameters to map projection (see mapPlot.
orientation	optional orientation of map projection (see mapPlot.

expand	numerical factor for the expansion of plot limits, showing area outside the plot, e.g. if showing a ship track as a gps, and then an actual gps to show the ocean boundary. The value of expand is ignored if either xlim or ylim is given.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
bg	optional colour to be used for the background of the map. This comes in handy for drawing insets (see "details").
axes	boolean, set to TRUE to plot axes.
cex.axis	value for axis font size factor.
add	boolean, set to TRUE to draw the gps on an existing plot. Note that this retains the aspect ratio of that existing plot, so it is important to set that correctly, e.g. with asp=1/cos(lat * pi / 180), where clat is the central latitude of the plot.
inset	set to TRUE for use within plotInset. The effect is to prevent the present function from adjusting margins, which is necessary because margin adjustment is the basis for the method used by plotInset.
geographical	flag indicating the style of axes. If geographical=0, the axes are conventional, with decimal degrees as the unit, and negative signs indicating the southern and western hemispheres. If geographical=1, the signs are dropped, with axis values being in decreasing order within the southern and western hemispheres. If geographical=2, the signs are dropped and the axes are labelled with degrees, minutes and seconds, as appropriate.
debug	set to TRUE to get debugging information during processing.
...	optional arguments passed to plotting functions. For example, set yaxp=c(-90,90,4) for a plot extending from pole to pole.

Details

This function plots a gps object. An attempt is made to use the whole space of the plot, and this is done by limiting either the longitude range or the latitude range, as appropriate, by modifying the eastern or northern limit, as appropriate.

To get an inset map inside another map, draw the first map, do par(new=TRUE), and then call plot.gps with a value of mar that moves the inset plot to a desired location on the existing plot, and with bg="white".

Value

None.

Author(s)

Dan Kelley

See Also

The documentation for gps-class explains the structure of gps objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
gps <- read.oce("~/Dropbox/dalwhoi.gpx")
plot(gps)
lines(coastlineWorld[["longitude"]], coastlineWorld[["latitude"]])

## End(Not run)
```

---

plot.landsat *Plot landsat data*

Description

Plot landsat data.

Usage

```
## S4 method for signature 'landsat'
plot(x, which=1, band, decimate=1, zlim, col=oceColorsPalette,
    debug=getOption("oceDebug"), ...)
```

Arguments

x	A landsat object, e.g. as read by read.landsat.
which	desired plot type; 1=image, 2=histogram.
zlim	either a pair of numbers giving the limits for the colourscale, or "histogram" to have a flattened histogram (i.e. to maximally increase contrast throughout the domain.) If not given, the 1 and 99 percent quantiles are calculated and used as limits.
col	a function yielding colours, taking a single integer argument with the desired number of colours.
band	if given, the number of the band to plot (between 1 and 11 for Landsat-8). If not given, 8 will be used if band8 exists in the object data, or otherwise the first band will be used.
decimate	decimation interval for images plots; e.g. setting to 10 means to show only every 10-th point in both directions, for a plot simplification by a factor 100. The purpose is mainly to speed up plotting, which can be slow on large landsat images.
debug	set to a positive value to get debugging information during processing.
...	optional arguments passed to plotting functions.

Details

Since landsat images are very detailed, it is sensible to use a fast plotting device, e.g. x11, and to use perhaps decimate=10 for initial images. The histogram plot can be handy in setting scales, e.g. when an image has a fair bit of land, the histogram will be double-lobed, and so quick examination can help in setting a good value for zlim. Using zlim="histogram" is probably the fastest way to explore an image for detail, but it is important to bear in mind that it yields a nonlinear colourscale.

Author(s)

Dan Kelley

See Also

The documentation for landsat-class explains the structure of landsat objects, and also outlines the other functions dealing with them. The ocedata package provides a dataset named landsat.

Examples

```
## Not run:
library(oce)
library(ocedata)
data(landsat)
plot(landsat, which=1, band=8, xlim=c(0.10, 0.15), breaks=200) # suggests z limits
plot(landsat, which=2)
plot(landsat, which=2, zlim=c(0.105, 0.130))
plot(landsat, which=2, zlim="histogram") # ugly but shows details best

## End(Not run)
```

---

plot.lisst *Plot LISST data*

Description

Plot LISST data

Usage

```
## S4 method for signature 'lisst'
plot(x, which=c(16, 37, 38), tformat, debug=getOption("oceDebug"), ...)
```

Arguments

x	a lisst object, e.g. as read by read.lisst.
which	list of desired plot types. These are graphed in panels running down from the top of the page. See "Details" for the meanings of various values of which.
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)

debug	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies.
...	optional arguments passed to plotting functions.

Details

Creates a multi-panel summary plot of data measured by LISST instrument. The panels are controlled by the `which` argument, as follows.

- `which=1` to `32`, or `which="C1"` to `"C32"` for a time-series graph of the named column (a size class).
- `which=33` or `which="lts"` for a time-series plot of laser transmission sensor.
- `which=34` or `which="voltage"` for a time-series plot of instrument voltage.
- `which=35` or `which="aux"` for a time-series plot of the external auxiliary input.
- `which=36` or `which="lrs"` for a time-series plot of the laser reference sensor.
- `which=37` or `which="pressure"` for a time-series plot of pressure.
- `which=38` or `which="temperature"` for a time-series plot of temperature.
- `which=41` or `which="transmission"` for a time-series plot of transmission, in percent.
- `which=42` or `which="beam"` for a time-series plot of beam-C, in 1/metre.

Author(s)

Dan Kelley

See Also

The documentation for `lisst-class` explains the structure of lisst objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(lisst)
plot(lisst)
```

`plot.lobo` *Plot lobo data*

Description

Plot a summary diagram for lobo data.

Usage

```
## S4 method for signature 'lobo'
plot(x,
     which=c(1,2,3),
     adorn=NULL,
     mgp=getOption("oceMgp"),
     mar=c(mgp[2]+1, mgp[1]+1, 1, mgp[1]+1.25),
     debug=getOption("oceDebug"),
     ...)
```

Arguments

x	A lobo object, e.g. as read by read.lobo.
which	A vector of numbers or character strings, indicating the quantities to plot. These are stacked in a single column. The possible values for which are as follows: 1 or "temperature" for a time series of temperature; 2 or "salinity" for salinity; 3 or "TS" for a TS diagram; 4 or "u" for a timeseries of the u component of velocity; 5 or "v" for a timeseries of the v component of velocity; 6 or "nitrate" for a timeseries of nitrate concentration; 7 or "fluorescence" for a timeseries of fluorescence value.
adorn	list of expressions to be executed for the panels in turn, e.g. to adorn the plots. If the number matches the number of panels, then the strings are applied to the appropriate panels, as they are drawn from top-left to bottom-right. If only a single expression is provided, it is used for all panels. (See "Examples".)
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions.

Details

Creates a summary plot for a lobo dataset.

Value

None.

Author(s)

Dan Kelley

References

http://lobo.satlantic.com/ and http://www.mbari.org/lobo/

See Also

The documentation for lobo-class explains the structure of LOBO objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(lobo)
plot(lobo)
```

---

plot.met *Plot meteorological data*

Description

Plot meteorological data

Usage

```
## S4 method for signature 'met'
plot(x,
     which=1:4,
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1,mgp[1]+1,mgp[1]+1,mgp[1]+1),
     tformat,
     debug=getOption("oceDebug"),
     ...)
```

Arguments

x	A cdt object, e.g. as read by read.met, or a list containing items named salinity and temperature.
which	list of desired plot types. • which=1 gives a time-series plot of temperature • which=2 gives a time-series plot of pressure • which=3 gives a time-series plot of the x (eastward) component of velocity • which=4 gives a time-series plot of the y (northward) component of velocity
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
debug	set to TRUE to get debugging information during processing.
...	optional arguments passed to plotting functions. A common example is to set df, for use in swN2 calculations.

Details

Creates a multi-panel summary plot of data measured in a meteorological data set. cast. The panels are controlled by the `which` argument. Normally, 4 panels are specified with the `which`, but it can also be useful to specify less than 4 panels, and then to draw other panels after this call.

If more than one panel is drawn, then on exit from `plot.met`, the value of `par` will be reset to the value it had before the function call. However, if only one panel is drawn, the adjustments to `par` made within `plot.met` are left in place, so that further additions may be made to the plot.

Author(s)

Dan Kelley

See Also

The documentation for `met-class` explains the structure of `met` objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(met)
plot(met, which=3:4)
```

`plot.sealevel` *Plot sealevel data*

Description

Plot a summary diagram for sealevel data.

Usage

```
## S4 method for signature 'sealevel'
plot(x, which=1:3, adorn=NULL,
drawTimeRange=getOption("oceDrawTimeRange"),
mgp=getOption("oceMgp"),
mar=c(mgp[1]+0.5,mgp[1]+1.5,mgp[2]+1,mgp[2]+3/4),
marginsAsImage=FALSE,
debug=getOption("oceDebug"),
...)
```

Arguments

x	an object of class "sealevel", e.g. as read by read.sealevel.
which	a numerical or string vector indicating desired plot types, with possibilities 1 or "all" for a time-series of all the data, 2 or "month" for a time-series of just the first month, 3 or "spectrum" for a power spectrum (truncated to frequencies below 0.1 cycles per hour, or 4 or "cumulativespectrum" for a cumulative integral of the power spectrum.
adorn	vector of expressions to be executed for the panels in turn, e.g. to adorn the plots. If the number matches the number of panels, then the strings are applied to the appropriate panels, as they are drawn from top-left to bottom-right. If only a single expression is provided, it is used for all panels. (See "Examples".)
drawTimeRange	boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
marginsAsImage	boolean, TRUE to put a wide margin to the right of time-series plots, matching the space used up by a palette in an imagep plot.
debug	a flag that turns on debugging, if it exceeds 0.
...	optional arguments passed to plotting functions.

Details

Creates a plot for a sea-level dataset, in one of two varieties. Depending on the length of which, either a single-panel or multi-panel plot is drawn. If there is just one panel, then the value of par used in plot.sealevel is retained upon exit, making it convenient to add to the plot. For multi-panel plots, par is returned to the value it had before the call, and so adorn must be used to add to individual panels.

Value

None.

Author(s)

Dan Kelley

References

The example refers to Hurricane Juan, which caused a great deal of damage to Halifax in 2003. Since this was in the era of the digital photo, a casual web search will uncover some spectacular images of damage, from both wind and storm surge. The wikipedia entry http://en.wikipedia.org/wiki/Hurricane_Juan provides a good entry to the topic, with the Canadian Hurricane Centre's http://www.atl.ec.gc.ca/weather/hurricane/juan/summary_e.html filling in some more technical details.

See Also

The documentation for sealevel-class explains the structure of sealevel objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(sealevel)
plot(sealevel)
plot(sealevel, which=1,
     xlim=as.POSIXct(c("2003-09-24","2003-10-05"), tz="UTC"))
juan <- as.POSIXct("2003-09-29 04:00:00", tz="UTC")
abline(v=juan, col="red")
mtext("Hurricane\nJuan", juan, col="red")
```

---

plot.section *Plot a CTD section*

Description

Plot a CTD section.

Usage

```
## S4 method for signature 'section'
plot(x,
     which=c(1, 2, 3, 99),
     eos=getOption("eos", default='unesco'),
     at=NULL,
     labels=TRUE,
     grid=FALSE,
     contourLevels=NULL,
     contourLabels=NULL,
     stationIndices,
     coastline=c("best", "coastlineWorld", "coastlineWorldMedium",
                 "coastlineWorldFine", "none"),
     xlim=NULL,
     ylim=NULL,
     map.xlim=NULL,
     map.ylim=NULL,
     xtype=c("distance", "track", "longitude", "latitude"),
     ytype=c("depth", "pressure"),
     ztype=c("contour", "image", "points"),
     zbreaks=NULL, zcol=NULL,
     legend.loc="bottomright",
     adorn=NULL,
     showStations=FALSE,
```

```
showBottom=TRUE,
mgp=getOption("oceMgp"),
mar=c(mgp[1]+1, mgp[1]+1, mgp[2], mgp[2] + 0.5),
col=par("col"), cex=par("cex"), pch=par("pch"),
debug=getOption("oceDebug"),
...)
```

Arguments

x	a section object, e.g. as created by makeSection.
which	a list of desired plot types, as explained in "Details". There may be up to four panels in total, and the desired plots are placed in these panels, in reading order. If only one panel is plotted, par is not adjusted, which makes it easy to add to the plot with subsequent plotting commands.
eos	either "unesco" or "teos". If the latter, then the computer must have the TEOS library installed; see teos. The effect of using "teos" is to plot "absolute salinity" instead of practical salinity, and conservative temperature instead of in-situ temperature. Typically, the temperature values will be very similar to those with unesco, but the salinities will be increased by 0.1 to 0.2 units. See teos for more information.
at	if NULL (the default), the x axis will indicate the distance of the stations from the first in the section. (This may give errors in the contouring routine, if the stations are not present in a geographical order.) If a list, then it indicates the values at which stations will be plotted.
labels	either a logical, indicating whether to put labels on the x axis, or a vector that is a list of labels to be placed at the x positions indicated by at.
grid	if TRUE, points are drawn at data locations.
contourLevels	optional contour levels
contourLabels	optional contour labels
stationIndices	optional list of the indices of stations to use. Note that an index is *not* a station number, e.g. to show the first 4 stations, use station.indices=1:4.
coastline	string giving the coastline to be used in a station map, or "best" to pick the one with highest resolution, or "none" to avoid drawing the coastline.
xlim	optional limit for x axis (only in sections, not map)
ylim	optional limit for y axis (only in sections, not map)
map.xlim	optional xlim for station location, which can be helpful in ensuring that a recognizable coastline can be seen. (This value is used instead of map.ylim, if both are supplied.)
map.ylim	optional ylim for station location, which can be helpful in ensuring that a recognizable coastline can be seen
xtype	type of x axis, for contour plots, either "distance" for distance (in km) to the first point in the section, "track" for distance along the cruise track, or "longitude" or "latitude". Note that if the x values are not in order, they will be put in order (which may make no sense) and a warning will be printed.

ytype	type of y axis for contour plots, either "pressure" for pressure (in dbar, with zero at the surface) or "depth" for depth (in m below the surface, calculated from pressure with swDepth).
ztype	string indicating whether to use contours, an image, or points. In the first two cases, the data must be gridded, with identical pressures at each station.
zbreaks, zcol	breaks and colours to be used if ztype="points" or "image". If not provided, a reasonable default is chosen. If zcol is a function, it will be invoked with an argument equal to 1+length(zbreaks). If zbreaks is not given, it defaults to a vector of length 200, with values spanning the data range.
legend.loc	location of legend, as supplied to legend.
adorn	list of expressions to be executed for the panels in turn, e.g. to adorn the plots. If the number matches the number of panels, then the strings are applied to the appropriate panels, as they are drawn from top-left to bottom-right. If only a single expression is provided, it is used for all panels. (See "Examples".)
showStations	logical indicating whether to draw station numbers on maps.
showBottom	logical indicating whether to draw the bottom, or a character string indicating the method for plotting the bottom. The allowed methods are: polygon, which fills the space to the bottom, or lines, which draws lines from stations to the bottom, or points, which draws points at the bottom.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
col	colour, as in par("col").
cex	value to be used with par("cex"), for any use of points, e.g. for which="data".
pch	value to be used with par("pch") (see cex, above.)
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to the contouring function, e.g. using labcex=1 will increase the size of contour labels.

Details

Creates a summary plot for a CTD section, with one panel for each value of which. The codes are as follows.

- which=1 or "temperature" for temperature contours (the default)
- which=2 or "salinity" for salinity contours
- which=3 or "sigmaTheta" for sigma-theta contours
- which=4 or "nitrate" for nitrate concentration contours
- which=5 or "nitrite" for nitrite concentration contours
- which=6 or "oxygen" for oxygen concentration contours
- which=7 or "phosphate" for phosphate concentration contours

- which=8 or "silicate" for silicate concentration contours
- which=20 or "data" for a dot for each data location
- which=99 or "map" for a location map

The y-axis for the contours is pressure, plotted in the conventional reversed form, so that the water surface appears at the top of the plot. The x-axis is more complicated. If at is not supplied, then the routine calculates x as the distance between the first station in the section and each of the other stations. (This will produce an error if the stations are not ordered geographically, because the contour routine cannot handle non-increasing axis coordinates.) If at is specified, then it is taken to be the location, in arbitrary units, along the x-axis of labels specified by labels; the way this works is designed to be the same as for axis.

Value

None.

Author(s)

Dan Kelley

See Also

The documentation for section-class explains the structure of section objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(section)
sg <- sectionGrid(section)

## AO3 section
plot(sg, which="salinity", ztype="points", pch=20, cex=1.5)

## Gulf Stream
GS <- subset(section, 109<=stationId&stationId<=129)
GSg <- sectionGrid(GS, p=seq(0,2000,100))
plot(GSg, which=c(1,99), map.ylim=c(34,42))

par(mfrow=c(2,1))
plot(GS, which=1, ylim=c(2000, 0), ztype='points',
zbreaks=seq(0,30,2), pch=20, cex=3)
plot(GSg, which=1, ztype='image', zbreaks=seq(0,30,2))

## image, with coloured dots to show if grid smoothing was OK
plot(GSg, which=1, ztype='image')
T <- GS[['temperature']]
col <- oceColorsJet(100)[rescale(T, rlow=1, rhigh=100)]
points(GS[['distance']],GS[['depth']],pch=20,cex=3,col='white')
points(GS[['distance']],GS[['depth']],pch=20,cex=2.5,col=col)
```

plot.tdr *Plot tdr (temperature-depth recorder) data*

Description

Plot tdr (temperature-depth recorder) data

Usage

```
## S4 method for signature 'tdr'
plot(x,
     which=1:4, title="", adorn=NULL,
     tlim, plim, Tlim,
     xlab, ylab,
     tformat,
     drawTimeRange=getOption("oceDrawTimeRange"),
     abbreviateTimeRange=getOption("oceAbbreviateTimeRange"),
     useSmoothScatter=FALSE,
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1.5,mgp[1]+1.5,1.5,1.5),
     main="",
     debug=getOption("oceDebug"),
     ...)
```

Arguments

x	tdr object, typically result of read.tdr.
which	list of desired plot types. These are graphed in panels running down from the top of the page. See "Details" for the meanings of various values of which.
title	character string to be used in the text-summary panel (which=2)
adorn	list of expressions to be executed for the panels in turn, e.g. to adorn the plots. If the number matches the number of panels, then the strings are applied to the appropriate panels, as they are drawn from top-left to bottom-right. If only a single expression is provided, it is used for all panels. (See "Examples".)
tlim	optional limits for time axis. If not provided, the value will be inferred from the data.
plim	optional limits for pressure axis. If not provided, the value will be inferred from the data. (It is helpful to specify this, if the auto-scaled value will be inappropriate, e.g. if more lines are to be added later.)
Tlim	optional limits for temperature axis. If not provided, the value will be inferred from the data. (It is helpful to specify this, if the auto-scaled value will be inappropriate, e.g. if more lines are to be added later.)
xlab	optional label for x axis.
ylab	optional label for y axis.

tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
drawTimeRange	boolean that applies to panels with time as the horizontal axis, indicating whether to draw the time range in the top-left margin of the plot.
abbreviateTimeRange	boolean that applies to panels with time as the horizontal axis, indicating whether to abbreviate the second time in the time range (e.g. skipping the year, month, day, etc. if it's the same as the start time).
useSmoothScatter	a boolean to cause smoothScatter to be used for profile plots, instead of plot.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
main	main title for plot, used just on the top panel, if there are several panels.
debug	a flag that turns on debugging, if it exceeds 0.
...	optional arguments passed to plotting functions.

Details

Several plots are available.

- which=1 or "temperature" for a time-series plot of temperature
- which=2 or "text" for textual information about the dataset
- which=3 or "pressure" for a time-series plot of pressure
- which=4 or "profile" for a temperature "profile"

Author(s)

Dan Kelley

See Also

The documentation for tdr-class explains the structure of PT objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(tdr)
t <- as.POSIXct('2008-07-04 20:00:00',tz='UTC')
plot(tdr, which=c(1,3), adorn=expression({abline(v=t)}))
```

plot.tidem *Plot a tidal fit*

Description

Plot a summary diagram for a tidal fit.

Usage

```
## S4 method for signature 'tidem'
plot(x,
     which=1,
     labelIf=NULL,
     log="",
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1,mgp[1]+1,mgp[2]+0.25,mgp[2]+1),
     ...)
```

Arguments

x	a tidem object, as created by tidem.
which	integer flag indicating plot type, 1 for stair-case spectral, 2 for spike spectral.
labelIf	if NULL, the function will indicate some particular tidal constituents; if a value is provided, labels will be given for any constituent with amplitude exceeding the value provided.
log	if set to "x", the frequency axis will be logarithmic.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
...	optional arguments passed to plotting functions.

Details

Creates a plot for a "tidem" object. See tidem for examples.

Author(s)

Dan Kelley

See Also

tidem fits a "tidem" object, and summary.tidem summarizes such an object.

Examples

```
## Not run:
library(oce)
data(sealevel)
tide <- tidem(sealevel)
plot(tide)

## End(Not run)
```

`plot.topo` *Plot topography data*

Description

Plot contours of topographic data.

Usage

```
## S4 method for signature 'topo'
plot(x,
     xlab="", ylab="",
     asp,
     clongitude, clatitude, span,
     expand=1.5,
     water.z,
     col.water,
     lty.water,
     lwd.water,
     land.z,
     col.land,
     lty.land,
     lwd.land,
     geographical=FALSE,
     location="topright",
     mgp=getOption("oceMgp"),
     mar=c(mgp[1]+1,mgp[1]+1,1,1),
     debug=getOption("oceDebug"),
     ...)
```

Arguments

x	an topo object, e.g. as read by `read.topo`.
xlab	label for x axis
ylab	label for y axis

asp	Aspect ratio for plot. The default is for plot.coastline to set the aspect ratio to give natural latitude-longitude scaling somewhere near the centre latitude on the plot. Often, it makes sense to set asp yourself, e.g. to get correct shapes at 45N, use asp=1/cos(45*pi/180). Note that the land mass is not symmetric about the equator, so to get good world views you should set asp=1 or set ylim to be symmetric about zero. Any given value of asp is ignored, if clongitude and clatitude are given.
clatitude	optional center latitude of map, in degrees north. If this and clongitude are provided, then any provided value of asp is ignored, and instead the plot aspect ratio is computed based on the center latitude. Also, if clongitude and clatitude are provided, then span must be, also.
clongitude	optional center longitude of map, in degrees east; see clatitude.
span	optional suggested span of plot, in kilometers (must be supplied, if clongitude and clatitude are supplied).
expand	numerical factor for the expansion of plot limits, showing area outside the plot, e.g. if showing a ship track as a coastline, and then an actual coastline to show the ocean boundary. The value of expand is ignored if either xlim or ylim is given.
water.z	depths at which to plot water contours. If not provided, these are inferred from the data.
col.water	colors corresponding to water.z values. If not provided, these will be "fill" colors from oceColorsGebco.
lty.water	line type(s) for water contours
lwd.water	line width(s) for water contours
land.z	depths at which to plot land contours. If not provided, these are inferred from the data. If set to NULL, no land contours will be plotted.
col.land	colors corresponding to land.z values. If not provided, these will be "fill" colors from oceColorsGebco.
lty.land	line type(s) for land contours
lwd.land	line width(s) for land contours
geographical	set to TRUE to get latitudes and longitudes without minus signs.
location	location for a legend (or "none", for no legend).
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
debug	boolean value, set to TRUE to get some debugging information.
...	additional arguments passed on to plotting functions

Details

The plot aspect ratio is set based on the middle latitude in the plot. Be aware that the ETOPO2 data are on a 2-minute (roughly 2-mile) spacing, and coastlines created with such data can be quite inaccurate on scales of 100km or less.

The line properties, such as land.lwd, may either be a single item, or a vector; in the latter case, the length must match the length of the corresponding properties, e.g. land.z.

Author(s)

Dan Kelley

See Also

TOPO data can be created with read.topo and summarized with summary.topo.

Examples

```
library(oce)
data(topoWorld)
plot(topoWorld)
plot(topoWorld, clongitude=300, clatitude=45, span=10000)
```

---

plot.windrose *Plot a wind rose diagram*

Description

Plot a wind-rose diagram

Usage

```
## S4 method for signature 'windrose'
plot(x, type=c("count","mean","median","fivenum"),
              convention=c("meteorological", "oceanographic"),
              mgp=getOption("oceMgp"),
              mar=c(mgp[1], mgp[1], 1+mgp[1], mgp[1]), col, ...)
```

Arguments

x	a windrose object, e.g. as created by as.windrose.
type	the thing to be plotted, either the number of counts in the angle interval, the mean of the values in the interval, the median of the values, or a fivenum representation of the values.
convention	string indicating whether to use meteorological convention or oceanographic convention for the arrows that emanate from the centre of the rose. In meteorological convenction, an arrow emanates towards the right on the diagram if the wind is from the east; in oceanographic convention, such an arrow indicates flow *to* the east.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.

mar	value to be used with par("mar").
col	an optional list of colours to use. If not set, the colours will be c("red", "pink", "blue", "lightgray' For the first three types of plot, the first colour in this list is used to fill in the rose, the third is used for the petals of the rose, and the fourth is used for grid lines. For the "fivenum" type, the first colour is used for the inter-quartile range, the second is used outside this range, the third is used for the median, and the fourth is, again, used for the grid lines.
...	optional arguments passed to plotting functions.

Details

Creates a wind-rose diagram.

Author(s)

Dan Kelley

See Also

as.windrose creates a wind-rose object, and summary.windrose produces a numerical summary.

Examples

```
library(oce)
opar <- par(no.readonly = TRUE)
xcomp <- rnorm(360) + 1
ycomp <- rnorm(360)
wr <- as.windrose(xcomp, ycomp)
par(mfrow=c(1,2))
plot(wr)
plot(wr, "fivenum")
par(opar)
```

---

plotInset *Plot an inset diagram*

Description

Plot an inset diagram

Usage

```
plotInset(xleft, ybottom, xright, ytop, expr,
    mar=c(2, 2, 1, 1),
    debug=getOption("oceDebug"))
```

Arguments

xleft	location of left-hand of the inset diagram, in the existing plot units. (PROVISIONAL FEATURE: this may also be "bottomleft", to put the inset there. Eventually, other positions may be added.)
ybottom	location of bottom side of the inset diagram, in the existing plot units.
xright	location of right-hand side of the inset diagram, in the existing plot units.
ytop	location of top side of the inset diagram, in the existing plot units.
expr	An expression that draws the inset plot. This may be a single plot command, or a sequence of commands enclosed in curly braces.
mar	margins, in line heights, to be used at the four sides of the inset diagram. (This is often helpful to save space.)
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

Adds an inset diagram to an existing plot. Note that if the inset is a map or coastline, it will be necessary to supply inset=TRUE to prevent the inset diagram from occupying the whole device width. After plotInset() has been called, any further plotting will take place within the inset, so it is essential to finish a plot before drawing an inset.

Author(s)

Dan Kelley

Examples

```
library(oce)
## power law in linear and log form
x <- 1:10
y <- x^2
plot(x, y, log='xy',type='l')
plotInset(3, 1, 10, 8,
          expr=plot(x,y,type='l',cex.axis=3/4,mgp=c(3/2,1/2,0)),
          mar=c(2.5,2.5,1,1))

## CTD data with location
data(ctd)
plot(ctd, which="TS")
plotInset(29.9, 2.7, 31, 10,
          expr=plot(ctd, which='map',
          coastline="coastlineWorld",
          span=5000, mar=NULL, cex.axis=3/4))
```

plotPolar *Draw a polar plot.*

Description

Draw a polar plot.

Usage

```
plotPolar(r, theta, debug=getOption("oceDebug"), ...)
```

Arguments

r	radii of points to plot.
theta	angles of points to plot, in degrees.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to the lower-level plotting functions.

Details

Creates a crude polar plot.

Author(s)

Dan Kelley

Examples

```
library(oce)
r <- rnorm(50, mean=2, sd=0.1)
theta <- runif(50, 0, 360)
plotPolar(r, theta)
```

plotProfile *Plot a CTD profile of various quantities*

Description

Plot a CTD profile, in any of several common formats.

Usage

```
plotProfile(x,
    xtype="salinity+temperature",
    ytype=c("pressure", "z", "depth", "sigmaTheta"),
    eos=getOption("eos", default='unesco'),
    xlab=NULL, ylab=NULL,
    col="black",
    col.salinity="darkgreen",
    col.temperature="red",
    col.rho="blue",
    col.N2="brown",
    col.dpdt="darkgreen",
    col.time="darkgreen",
    pt.bg="transparent",
    grid=TRUE,
    col.grid="lightgray",
    lty.grid="dotted",
    Slim, Tlim, densitylim, N2lim, Rrholim, dpdtlim, timelim, ylim,
    lwd=par("lwd"),
    xaxs="r", yaxs="r",
    cex=1, pch=1,
    useSmoothScatter=FALSE,
    df,
    keepNA=FALSE,
    type='l',
    mgp=getOption("oceMgp"),
   mar=c(1 + if (length(grep('\\+', xtype))) mgp[1] else 0, mgp[1]+2, mgp[1] + 2, 2),
    add=FALSE, inset=FALSE,
    debug=getOption("oceDebug"),
    ...)
```

Arguments

x	A cdt object, e.g. as read by read.ctd.
xtype	Item(s) plotted on the x axis, either a vector of length equal to that of x@data$pressure or a text code from the list below.
	"salinity" Profile of salinity.
	"temperature" Profile of *in-situ* temperature.
	"theta" Profile of *potential* temperature.
	"density" Profile of density (expressed as σ_θ).
	"index" Index of sample (very useful for working with ctdTrim).
	"salinity+temperature" Profile of salinity and temperature within a single axis frame.
	"N2" Profile of square of buoyancy frequency N^2, calculated with swN2 with an optional argument setting of df=length(x[["pressure"]])/4 to do some smoothing.

"density+N2" Profile of sigma-theta and the square of buoyancy frequency within a single axis frame.

"density+dpdt" Profile of sigma-theta and dP/dt for the sensor. The latter is useful in indicating problems with the deployment. It is calculated by first differencing pressure and then using a smoothing spline on the result (to avoid grid-point wiggles that result because the SBE software only writes 3 decimal places in pressure). Note that dP/dt may be off by a scale factor; this should not be a problem if there is a time column in the data slot, or a sample.rate in the metadata slot.

"spice" Profile of spice

"Rrho" Profile of Rrho, defined in the diffusive sense

"RrhoSF" Profile of Rrho, defined in the salt-finger sense

an expression an expression to be evaluated, in the calling environment, for some quantity; in this case, it makes sense to specify also xlab.

ytype variable to use on y axis; note that z is the negative of depth.

eos name of equation of state to be used, either "unesco" or "teos". If the latter, then the computer must have the TEOS library installed; see teos.

xlab optional label for x axis (at top of plot).

ylab optional label for y axis. Set to "" to prevent labelling the axis.

col colour for a general profile.

col.salinity color for salinity profile (see "Details").

col.temperature color for temperature (see "Details").

col.rho color for density (see "Details").

col.N2 color for square of buoyancy frequency (see "Details").

col.dpdt color for dP/dt.

col.time color for delta-time.

pt.bg inside color for symbols with pch in 21:25

grid logical, set to TRUE to get a grid.

col.grid colour for grid.

lty.grid line type for grid.

Slim Optional limit for S axis

Tlim Optional limit for T axis

densitylim Optional limit for density axis

N2lim Optional limit for N2 axis

Rrholim Optional limit for Rrho axis

dpdtlim Optional limit for dp/dt axis

timelim Optional limit for delta-time axis

ylim Optional limit for y axis

lwd lwd value for data line

xaxs	value of par xaxs to use
yaxs	value of par yaxs to use
cex	size to be used for plot symbols (see par)
pch	code for plotting symbol (see par).
useSmoothScatter	boolean, set to TRUE to use smoothScatter instead of plot to draw the plot.
df	optional argument, passed to swN2 if provided, and if a plot using N^2 is requested.
keepNA	FALSE
type	type of plot to draw, using the same scheme as plot.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
add	a flag that controls whether to add to an existing plot. (It makes sense to use add=TRUE in the panel argument of a coplot, for example.)
inset	set to TRUE for use within plotInset. The effect is to prevent the present function from adjusting margins, which is necessary because margin adjustment is the basis for the method used by plotInset.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to other functions. A common example is to set df, for use in swN2 calculations.

Details

The colours (col.salinity, etc.) are ony used if two profiles appear on a plot.

Value

None.

Author(s)

Dan Kelley

See Also

read.ctd scans ctd information from a file, and plotTS plots a temperature-salinity diagram.

Examples

```
library(oce)
data(ctd)
plotProfile(ctd, xtype="temperature")
```

plotScan *Plot seawater data in a low-level fashion*

Description

Plot CTD data as time-series against scan number, to help with trimming extraneous data from a CTD cast.

Usage

```
plotScan(x, name = "scan", adorn=NULL, mgp=getOption("oceMgp"), type='l', ...)
```

Arguments

x	A ctd object, e.g. as read by read.ctd.
name	name of variable for x axis
adorn	list of character strings containing commands to be executed for the panels. If the number matches the number of panels, then the strings are applied to the appropriate panels, as they are drawn from top-left to bottom-right. If only a single string is provided, it is used for all panels. (See "Examples".)
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
type	line type
...	optional arguments passed to plotting functions.

Details

Plots ctd data as time-series against the scan number, as an aide to trimming to downcasts, *etc*.

Author(s)

Dan Kelley

See Also

summary.ctd summarizes a ctd object plot.ctd plot summary diagram of ctd object. read.ctd scans ctd object from a file.

Examples

```
library(oce)
data(ctd)
plotScan(ctd)

# demonstrate adding elements to plots, e.g. to narrow
# down on good start end points
plotScan(ctd, adorn="abline(v=240,col='red')")
```

plotSticks *Draw a stick-plot diagram*

Description

Draw a stick-plot diagram

Usage

```
plotSticks(x, y, u, v, yscale=1, add=FALSE, length=1/20,
           mgp=getOption("oceMgp"),
           mar=c(mgp[1]+1,mgp[1]+1,1,1+par("cex")),
           ...)
```

Arguments

x	x coordinates of stick origins.
y	y coordinates of stick origins. If not supplied, 0 will be used; if length is less than that of x, the first number is repeated and the rest are ignored.
u	x component of stick length.
v	y component of stick length.
yscale	scale from u and v to y (see "Details").
add	boolean, set TRUE to add to an existing plot.
length	value to be provided to arrows; here, we set a default that is smaller than normally used, because these plots tend to be crowded in oceanographic applications.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar").
...	graphical parameters passed down to arrows. It is common, for example, to use smaller arrow heads than arrows uses; see "Examples".

Details

The arrows are drawn with directions on the graph that match the directions indicated by the u and v components. The arrow size is set relative to the units of the y axis, according to the value of yscale, which has the unit of v divided by the unit of y.

The interpretation of diagrams produced by plotSticks can be difficult, owing to overlap in the arrows. For this reason, it It is often a good idea to smooth u and v before using this function.

Author(s)

Dan Kelley

Examples

```
library(oce)

# Flow from a point source
n <- 16
x <- rep(0, n)
y <- rep(0, n)
theta <- seq(0, 2*pi, length.out=n)
u <- sin(theta)
v <- cos(theta)
plotSticks(x, y, u, v, xlim=c(-2, 2), ylim=c(-2, 2))
rm(n, x, y, theta, u, v)

# Oceanographic example
if (require(ocedata)) {
    data(airHalifax)
    t <- airHalifax$t
    u <- airHalifax$u
    v <- airHalifax$v
    temperature <- airHalifax$temperature
    oce.plot.ts(t, temperature, type='l', ylim=c(0,25))
    plotSticks(t, 5, u, v, yscale=2, add=TRUE)
}
```

---

plotTaylor *Plot a model-data comparison diagram.*

Description

Plot a model-data comparison diagram as described by Taylor (2001).

Usage

```
plotTaylor(x, y, scale, pch, col, labels, pos, ...)
```

Arguments

x	a vector of reference values of some quantity, e.g. measured over time or space.
y	a matrix whose columns hold values of values to be compared with those in x. (If y is a vector, it is converted first to a one-column matrix).
scale	optional scale, interpreted as the maximum value of standard deviation.
pch	list of characters to plot, one for each column of y.
col	list of colours for points on the plot, one for each column of y.
labels	optional vector of strings to use for labelling the points.
pos	optional vector of positions for labelling strings. If not provided, labels will be to the left of the symbols.
...	optional arguments passed by plotTaylor to more child functions.

Details

Creates a diagram as described by Taylor (2001). The graph is in the form of a semicircle, with radial lines and spokes connecting at a focus point on the flat (lower) edge. The radius of a point on the graph indicates the standard deviation of the corresponding quantity, i.e. x and the columns in y. The angle connecting a point on the graph to the focus provides an indication of correlation coefficient with respect to x. The "east" side of the graph indicates $R = 1$, while $R = 0$ is at the "north edge" and $R = -1$ is at the "west" side. The x data are indicated with a bullet on the graph, appearing on the lower edge to the right of the focus at a distance indicating the standard deviation of x. The other points on the graph represent the columns of y, coded automatically or with the supplied values of pch and col.

The example shows two tidal models of the Halifax sealevel data, computed with tidem with just the M2 component and the S2 component; the graph indicates that the M2 model is much better than the S2 model.

Author(s)

Dan Kelley

References

Taylor, Karl E., 2001. Summarizing multiple aspects of model performance in a single diagram, *J. Geophys. Res.*, 106:D7, 7183–7192.

Examples

```
library(oce)
data(sealevel)
x <- elevation(sealevel)
M2 <- predict(tidem(sealevel, constituents="M2"))
S2 <- predict(tidem(sealevel, constituents=c("S2")))
plotTaylor(x, cbind(M2, S2))
```

---

plotTS *Plot temperature-salinity diagram*

Description

Plot temperature-salinity diagram for seawater (CTD) data.

Usage

```
plotTS(x,
    inSitu=FALSE,
    type="p",
    referencePressure=0,
    nlevels=6, levels,
    grid=TRUE,
```

```
    col.grid="lightgray",
    lty.grid="dotted",
    rho1000=FALSE,
    eos=getOption("eos", default='unesco'),
    cex=par("cex"), col=par("col"), pch=par("pch"),
    bg, pt.bg="transparent",
    col.rho="darkgray",
    cex.rho=3/4*par("cex"),
    rotate=FALSE,
    useSmoothScatter=FALSE,
    xlab,
    ylab,
    Slim,
    Tlim,
    mgp=getOption("oceMgp"),
    mar=c(mgp[1]+1.5,mgp[1]+1.5,mgp[1],mgp[1]),
    lwd=par('lwd'), lty=par('lty'),
    lwd.rho=par("lwd"), lty.rho=par("lty"),
    add=FALSE, inset=FALSE,
    debug=getOption("oceDebug"),
    ...)
```

Arguments

x	An object containing salinity and temperature data, typically a cdt object or section object.
inSitu	A boolean indicating whether to use in-situ temperature or (the default) potential temperature, calculated with reference pressure given by referencePressure. This is ignored if eos="teos", because in that case the y axis is necessarily the conservative formulation of temperature.
type	representation of data, "p" for points, "l" for connecting lines, or "n" for no indication.
referencePressure	reference pressure, to be used in calculating potential temperature, if inSitu is FALSE.
nlevels	Number of automatically-selected isopycnal levels (ignored if levels is supplied).
levels	Optional vector of desired isopycnal levels.
grid	a flag that can be set to TRUE to get a grid.
col.grid	colour for grid.
lty.grid	line type for grid.
rho1000	if TRUE, label isopycnals as e.g. 1024; if FALSE, label as e.g. 24
eos	either "unesco" or "teos". If the latter, then the computer must have the TEOS library installed; see teos. The effect of using "teos" is to us "absolute salinity" on the x axis and "conservative temperature" on the y axis. Typically, the temperature values will be very similar to those with unesco, but the salinities

	will be increased by 0.1 to 0.2 units. The isopycnals will still run through the same points as for the unesco style. See teos for more information.
cex	character-expansion factor for symbols, as in par("cex").
pch	symbol type, as in par("pch").
bg	optional colour to be painted under plotting area, before plotting. (This is useful for cases in which inset=TRUE.)
pt.bg	inside color for symbols with pch in 21:25
col	colour for symbols.
col.rho	colour for isopycnal lines.
cex.rho	size of isopycnal labels.
rotate	if TRUE, labels in right-hand margin are written vertically
useSmoothScatter	if TRUE, use smoothScatter to plot the points.
xlab	optional label for the x axis, with default "Salinity [PSU]".
ylab	optional label for the y axis, with default "Temperature [C]".
Slim	optional limits for salinity axis, otherwise inferred from data.
Tlim	optional limits for temperature axis, otherwise inferred from data.
mgp	3-element numerical vector to use for par(mgp), and also for par(mar), computed from this. The default is tighter than the R default, in order to use more space for the data and less for the axes.
mar	value to be used with par("mar"). If set to NULL, then par("mar") is used. A good choice for a TS diagram with a palette to the right is mar=par("mar")+c(0, 0, 0, 1)).
lwd	line width of lines or symbols.
lty	line type of lines or symbols.
lwd.rho	line width for density curves.
lty.rho	line type for density curves.
add	a flag that controls whether to add to an existing plot. (It makes sense to use add=TRUE in the panel argument of a coplot, for example.)
inset	set to TRUE for use within plotInset. The effect is to prevent the present function from adjusting margins, which is necessary because margin adjustment is the basis for the method used by plotInset.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional arguments passed to plotting functions.

Details

Creates a temperature-salinity plot for a CTD cast, with labeled isopycnals.

Value

None.

Author(s)

Dan Kelley

See Also

summary.ctd summarizes the information, while read.ctd scans it from a file.

Examples

```
library(oce)
data(ctd)
plotTS(ctd)
```

---

predict.tidem *Predict a time series from a tidal model*

Description

Predict a time series from a tidal model.

Usage

```
## S3 method for class 'tidem'
predict(object, newdata, ...)
```

Arguments

object	a tidem object.
newdata	optional vector of POSIXt times at which to make the prediction. If not present, predict.tidem uses the times that were provided in the original call to tidem.
...	optional arguments passed on to children.

Details

This is a wrapper around the predict method for object$model.

Value

A vector of predictions.

Author(s)

Dan Kelley

See Also

tidem fits a tidal model.

Examples

```
## Not run:
library(oce)
# 1. tidal anomaly
data(sealevelTuktoyaktuk)
time <- sealevelTuktoyaktuk[["time"]]
elevation <- sealevelTuktoyaktuk[["elevation"]]
oce.plot.ts(time, elevation, type='l', ylab="Height [m]", ylim=c(-2,6))
tide <- tidem(sealevelTuktoyaktuk)
lines(time, elevation - predict(tide), col="red")
abline(h=0, col="red")

# 2. prediction at specified times
data(sealevel)
m <- tidem(sealevel)
## Check fit over 2 days (interpolating to finer timescale)
look <- 1:48
time <- sealevel[["time"]]
elevation <- sealevel[["elevation"]]
oce.plot.ts(time[look], elevation[look])
# 360s = 10 minute timescale
t <- seq(from=time[1], to=time[max(look)], by=360)
lines(t, predict(m,newdata=t), col='red')
legend("topright", col=c("black","red"),
legend=c("data","model"),lwd=1)

## End(Not run)
```

---

prettyPosition *Pretty lat/lon in deg, min, sec*

Description

Round a geographical positions in degrees, minutes, and seconds

Usage

```
prettyPosition(x, debug=getOption("oceDebug"))
```

Arguments

x	a series of one or more values of a latitude or longitude, in decimal degrees
debug	set to a positive value to get debugging information during processing.

Details

Depending on the range of values in x, rounding is done to degrees, half-degrees, minutes, etc.

Value

A vector of numbers that will yield good axis labels if formatPosition is used.

Author(s)

Dan Kelley

Examples

```
library(oce)
formatPosition(prettyPosition(10+1:10/60+2.8/3600))
```

---

processingLog *Add an item to object processingLog*

Description

Add an item to object processingLog

Usage

```
processingLog(x) <- value
processingLog(h, value="")
processingLogItem(value="")
processingLogShow(x)
```

Arguments

h	an oce object, a processingLog, or NULL. If h is an oce object, then processingLog returns a summary of the processing log, and value is ignored. If h is a processing log, then value is added to it. If h is NULL, then a new processing log is created, with value as the single item.
value	a character string describing the action.
x	an oce object.

Details

The processingLog function is designed for the user to alter the processingLog of an object. The processingLogItem function is used internally within the package.

Value

The updated processingLog.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
ctd@metadata$latitude <- ctd@metadata$latitude + 1
processingLog(ctd) <- "correct the latitude"
```

`pwelch` *Welch periodogram*

Description

Compute periodogram using the Welch (1967) method

Usage

```
pwelch(x, window, noverlap, nfft, fs, spectrumtype, esttype,
     plot=TRUE, debug=getOption("oceDebug"), ...)
```

Arguments

`x`	a vector or timeseries to be analyzed. If a timeseries, then there is no need to specify `fs`.
`window`	window specification, either a single value giving the number of windows to use, or a vector of window coefficients. If not specified, then 8 windows are used, each with a Hamming (raised half-cosine) window.
`noverlap`	number of points to overlap between windows. If not specified, this will be set to half the window length.
`nfft`	length of FFT. This cannot be given if `window` is given, and the latter is a single integer.
`fs`	frequency of time-series. If `x` is a time-series, and if `fs` is supplied, then time-series is altered to have frequency `fs`.
`spectrumtype`	not used (yet)
`esttype`	not used (yet)
`plot`	logical, set to `TRUE` to plot the spectrum.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`...`	optional extra arguments to be passed to `spectrum`. Unless specified in this list, `spectrum` is called with `plot=FALSE` to prevent plotting the separate spectra, and with `taper=0`, which is not needed with the default Hanning window. However, the other defaults of `spectrum` are used, e.g. `detrend=TRUE`.

Details

The Welch (1967) method is used. First, x is broken up into chunks, overlapping as specified by noverlap. These chunks are then detrended with detrend, multiplied by the window, and then passed to spectrum. The resulting spectra are then averaged, with the results being stored in spec of the return value. Other entries of the return value mimic those returned by spectrum.

Value

List mimicking the return value from spectrum, containing frequency freq, spectral power spec, degrees of freedom df, bandwidth bandwidth, etc.

Bugs

Both bandwidth and degrees of freedom are just copied from the values for one of the chunk spectra, and are thus incorrect. That means the cross indicated on the graph is also incorrect.

Author(s)

Dan Kelley

References

Welch, P. D., 1967. The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms. *IEEE Transactions on Audio Electroacoustics*, AU-15, 70–73.

Examples

```
library(oce)
Fs <- 1000
t <- seq(0, 0.296, 1/Fs)
x <- cos(2 * pi * t * 200) + rnorm(n=length(t))
xts <- ts(x, frequency=Fs)
s <- spectrum(xts, spans=c(3,2), main="random + 200 Hz", log='no')
w <- pwelch(xts, plot=FALSE)
lines(w$freq, w$spec, col="red")
w2 <- pwelch(xts, nfft=75, plot=FALSE)
lines(w2$freq, w2$spec, col='green')
abline(v=200, col="blue", lty="dotted")
cat("Checking spectral levels with Parseval's theorem:\n")
cat("var(x)                              = ", var(x), "\n")
cat("2 * sum(s$spec) * diff(s$freq[1:2]) = ", 2 * sum(s$spec) * diff(s$freq[1:2]), "\n")
cat("sum(w$spec) * diff(s$freq[1:2])     = ", sum(w$spec) * diff(w$freq[1:2]), "\n")
cat("sum(w2$spec) * diff(s$freq[1:2])    = ", sum(w2$spec) * diff(w2$freq[1:2]), "\n")
## co2
par(mar=c(3,3,2,1), mgp=c(2,0.7,0))
s <- spectrum(co2, plot=FALSE)
plot(log10(s$freq), s$spec * s$freq,
     xlab=expression(log[10]*Frequency), ylab="Power*Frequency", type='l')
title("Variance-preserving spectrum")
pw <- pwelch(co2, nfft=256, plot=FALSE)
```

```
lines(log10(pw$freq), pw$spec * pw$freq, col='red')
```

`rangeExtended` *Calculate range, extended a little*

Description

Calculate range, extended a little, as is done for axes.

Usage

```
rangeExtended(x, extend=0.04)
```

Arguments

`x`	a numeric vector.
`extend`	fraction to extend on either end

Details

This is analogous to what is done as part of the R axis range calculation, in the case where `xaxs="r"`.

Value

A two-element vector with the extended range of `x`.

Author(s)

Dan Kelley

`rangeLimit` *Substitute NA for data outside a range*

Description

Substitute NA for data outside a range, e.g. to remove wild spikes in data.

Usage

```
rangeLimit(x, min, max)
```

Arguments

`x`	vector of values
`min`	minimum acceptable value. If not supplied, and if `max` is also not supplied, a `min` of the 0.5 percentile will be used.
`max`	maximum acceptable value. If not supplied, and if `min` is also not supplied, a `min` of the 0.995 percentile will be used.

Author(s)

Dan Kelley

Examples

```
ten.to.twenty <- rangeLimit(1:100, 10, 20)
```

`read.adp` *Read an ADP data file*

Description

Read an ADP data file, producing an object of type adp.

Usage

```
read.adp(file, from=1, to, by=1, tz=getOption("oceTz"),
  longitude=NA, latitude=NA,
  manufacturer=c("rdi", "nortek", "sontek"),
  monitor=FALSE, despike=FALSE, processingLog,
  debug=getOption("oceDebug"),
  ...)
read.adp.rdi(file, from=1, to, by=1, tz=getOption("oceTz"),
  longitude=NA, latitude=NA,
  type=c("workhorse"),
  monitor=FALSE, despike=FALSE, processingLog,
  testing=FALSE,
  debug=getOption("oceDebug"),
  ...)

read.aquadopp(file, from=1, to, by=1, tz=getOption("oceTz"),
  longitude=NA, latitude=NA,
  orientation, distance,
  monitor=FALSE, despike=FALSE, processingLog,
  debug=getOption("oceDebug"),
  ...)

read.aquadoppHR(file, from=1, to, by=1, tz=getOption("oceTz"),
  longitude=NA, latitude=NA,
  orientation, distance,
  monitor=FALSE, despike=FALSE, processingLog,
  debug=getOption("oceDebug"),
  ...)
read.aquadoppProfiler(file, from=1, to, by=1, tz=getOption("oceTz"),
  longitude=NA, latitude=NA,
  orientation, distance,
  monitor=FALSE, despike=FALSE, processingLog,
```

```
  debug=getOption("oceDebug"),
  ...)

read.adp.nortek(file, from=1, to, by=1, tz=getOption("oceTz"),
  longitude=NA, latitude=NA,
  type=c("aquadoppHR", "aquadoppProfiler", "aquadopp"),
  orientation, distance,
  monitor=FALSE, despike=FALSE, processingLog,
  debug=getOption("oceDebug"),
  ...)

read.adp.sontek(file, from=1, to, by=1, tz=getOption("oceTz"),
  longitude=NA, latitude=NA,
  type=c("adp","pcadp"),
  monitor=FALSE, despike=FALSE, processingLog,
  debug=getOption("oceDebug"),
  ...)

read.adp.sontek.serial(file, from=1, to, by=1, tz=getOption("oceTz"),
  longitude=NA, latitude=NA,
  type=c("adp", "pcadp"),
  beamAngle=25, orientation,
  monitor=FALSE, processingLog,
  debug=getOption("oceDebug"))
```

Arguments

file	a connection or a character string giving the name of the file to load. (For read.adp.sontek.serial, this is generally a list of files, which will be concatenated.
from	indication of the first profile to read. This can be an integer, the sequence number of the first profile to read, or a POSIXt time before which profiles should be skipped, or a character string that converts to a POSIXt time (assuming UTC timezone). See "Examples", and make careful note of the use of the tz argument.
to	if supplied, an indication of the last profile to read, in a format as described for from. If not supplied, the whole file will be read.
by	an indication of the stride length to use while walking through the file. If this is an integer, then by-1 profiles are skipped between each pair of profiles that is read. If this is a string representing a time interval, in colon-separated format (MM:SS), then this interval is divided by the sampling interval, to get the stride length.
manufacturer	a character string indicating the manufacturer, used by the general function read.adp to select a subsidiary function to use, such as read.adp.nortek.
type	a character string indicating the type of instrument.
orientation	optional character string specifying the orientation of the sensor, provided for those cases in which it cannot be inferred from the data file. The valid choices are "upward", "downward", and "sideward".

distance	optional vector holding the distances of bin centres from the sensor. This argument is ignored except for Nortek profilers, and need not be given if the function determines the distances correctly from the data. The problem is that the distance is poorly documented in the Nortek System Integrator Guide (2008 edition, page 31), so the function must rely on word-of-mouth formulae that do not work in all cases.
tz	character string indicating time zone to be assumed in the data.
longitude	optional signed number indicating the longitude in degrees East.
latitude	optional signed number indicating the latitude in degrees North.
beamAngle	angle between instrument axis and beams, in degrees.
monitor	boolean, set to TRUE to provide an indication (with numbers and dots) of every profile read.
despike	if TRUE, despike will be used to clean anomalous spikes in heading, etc.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
testing	a flag that applies only to RDI units. If this is TRUE, then the time-varying device orientation is inferred from the per-profile header information, and the boolean result is stored in an integer vector named upward within the data slot.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	additional arguments, passed to called routines.

Details

Reads a binary-format ADP file. Several file types can be handled. Some of these functions are wrappers that map to device names, e.g. read.aquadoppProfiler does its work by calling read.adp.nortek; in this context, it is worth noting that the "aquadopp" instrument is a one-cell profiler that might just as well have been documented under the heading read.adv.

Value

An object of class "adp", which contains measurements made with an ADP device. The value of metadata$coordinate is set to "beam", a fact that is used in other steps in processing. For information on data stored in the object, see "Details".

There are three types of element stored in the result's data, namely space-series, time-series, and matrix. These are contained within a list named data, as follows:

distance A space-series that stores the distance of cells from the transducer head, in the vertical (not slant-wise) direction.

time Times of the profiles, in POSIXct format.

pressure Pressure, in decibars.

temperature The temperature, in deg C.

salinity The salinity, in PSU. (This may be measured, or just a repeated constant value specified when the equipment was set up.)

`depth.of.transducer` The depth of the transducer.

`heading` The heading of the instrument, in degrees.

`pitch` The pitch of the instrument, in degrees.

`roll` The roll of the instrument, in degrees.

`v` A 3-D matrix of velocity. The first index corresponds to profile number, the second to cell number, and the third to beam number.

`a` A 3-D matrix of backscatter amplitude, corresponding to `v`.

`q` A 3-D matrix of a measure of the quality of the data, corresponding to `v`.

`br` Depth to bottom in each of 4 beams (only for Teledyne-RDI devices that have bottom tracking.

`bv` Bottom velocity in each of 4 beams (only for Teledyne-RDI devices that have bottom tracking.

In ADP data files, velocities are coded to signed 2-byte integers, with a scale factor being used to convert to velocity in metres per second. These two facts control the maximum recordable velocity and the velocity resolution, values that may be retrieved for an ADP object name `d` with `d[["velocityMaximum"]]` and `d[["velocityResolution"]]`.

Implementation notes

- **Teledyne-RDI files.** If a heading bias had been set with the `EB` command during the setup for the deployment, then a heading bias will have been stored in the file's header. This value is stored in the object's metadata as `metadata$heading.bias`. **Importantly**, this value is subtracted from the headings stored in the file, and the result of this subtraction is stored in the objects heading value (in `data$heading`). It should be noted that `read.adp.rdi()` was tested for firmware version 16.30. For other versions, there may be problems. For example, the serial number is not recognized properly for version 16.28.
- **Nortek Aquadopp files.** The R code is based on information in the Nortek System Integrator Guide (2008) and on postings on the Nortek "knowledge center" discussion board. One might assume that the latter is less authorative than the former. For example, the inference of cell size follows advice found at http://www.nortekusa.com/en/knowledge-center/forum/hr-profilers/736804717 (downloaded June 2012)), which contains a typo in an early posting that is corrected later on.

A warning for RDI files

The upward/downward orientation is inferred from the "fixed" header of the first profile in the data file, so it will be incorrect if the deployment orientation is different. This poses a problem for data recorded in beam coordinates, because the orientation is used by e.g. `beamToXyz` in converting to xyz coordinates. The solution is to alter the value of `@metadata$orientation` prior to doing any such coordinate transformation.

Author(s)

Dan Kelley and Clark Richards

References

1. Teledyne-RDI, 2007. *WorkHorse commands and output data format.* P/N 957-6156-00 (November 2007). (Section 5.3 h details the binary format, e.g. the file should start with the byte `0x7f` repeated twice, and each profile starts with the bytes `0x80`, followed by `0x00`, followed by the sequence number of the profile, represented as a little-endian two-byte short integer. `read.adp.rdi()` uses these sequences to interpret data files.)

2. Information on Nortek profilers (including the System Integrator Guide, which explains the data format byte-by-byte) is available at http://www.nortekusa.com/. (One must join the site to see the manuals.)

3. Information about Sontek profilers is available at http://www.sontek.com.

4. The Nortek Knowledge Center http://www.nortekusa.com/en/knowledge-center may be of help if problems arise in dealing with data from Nortek instruments.

See Also

The documentation for adp-class explains the structure of ADP objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
# A day sampled at 1/2 hour interval.  Note the timezone.
dir <- "/data/archive/sleiwex/2008/moorings/"
f <- paste(dir, "m09/adp/rdi_2615/raw/adp_rdi_2615.000", sep="")
d <- read.oce(f, from=as.POSIXct("2008-06-26", tz="UTC"),
                 to=as.POSIXct("2008-06-27", tz="UTC"), by="30:00")
summary(d)
plot(d)

## End(Not run)
```

---

read.adv *Read an ADV data file*

Description

Read an ADV data file, producing an object of type adv.

Usage

```
read.adv(file, from=1, to, by=1, tz=getOption("oceTz"),
  type=c("nortek", "sontek", "sontek.adr", "sontek.text"),
  header=TRUE,
  longitude=NA, latitude=NA,
  start, deltat,
  debug=getOption("oceDebug"), monitor=FALSE, processingLog)
```

```
read.adv.nortek(file, from=1, to, by=1, tz=getOption("oceTz"),
  header=TRUE,
  longitude=NA, latitude=NA,
  type=c("vector", "aquadopp"),
  haveAnalog1=FALSE, haveAnalog2=FALSE,
  debug=getOption("oceDebug"), monitor=FALSE, processingLog)

read.adv.sontek.serial(file, from=1, to, by=1, tz=getOption("oceTz"),
  longitude=NA, latitude=NA,
  start, deltat,
  debug=getOption("oceDebug"), monitor=FALSE, processingLog)

read.adv.sontek.adr(file, from=1, to, by=1, tz=getOption("oceTz"),
  header=TRUE,
  longitude=NA, latitude=NA,
  debug=getOption("oceDebug"), monitor=FALSE, processingLog)

read.adv.sontek.text(basefile, from=1, to, by=1, tz=getOption("oceTz"),
  originalCoordinate="xyz",
  transformationMatrix,
  longitude=NA, latitude=NA,
  debug=getOption("oceDebug"), monitor=FALSE, processingLog)
```

Arguments

file	a connection or a character string giving the name of the file to load. It is also possible to give file as a vector of filenames, to handle the case of data split into files by a data logger. In the multi-file case, header must be FALSE, start must be a vector of times, and deltat must be provided.
basefile	character string naming the base of filenames to load (used only by read.adv.sontek.text). The actual filenames are constructed by appending ".hd1" and ".ts1" to the base name.
from	index number of the first profile to be read, or the time of that profile, as created with as.POSIXct (hint: use tz="UTC"). This argument is ignored if header==FALSE. See "Examples".
haveAnalog1	optional argument, only for Nortek devices: indicates whether to extract the "analog1" (8-bit) data.
haveAnalog2	optional argument, only for Nortek devices: indicates whether to extract the "analog2" (16-bit) data.
to	indication of the last profile to read, in a format matching that of from. This is ignored if header==FALSE.
by	an indication of the stride length to use while walking through the file. This is ignored if header==FALSE. Otherwise, if this is an integer, then by-1 profiles are skipped between each pair of profiles that is read. This may not make much sense, if the data are not equi-spaced in time. If by is a string representing a time interval, in colon-separated format, then this interval is divided by the sampling

interval, to get the stride length. *BUG:* by only partially works; see the "Bugs" section below.

`header` a boolean indicating whether the file contains a header at the start. (This will not be the case for files that are created by data loggers that chop the raw data up into a series of sub-files, e.g. once per hour.)

`longitude` optional signed number indicating the longitude in degrees East.

`latitude` optional signed number indicating the latitude in degrees North.

`start` the time of the first sample, typically created with `as.POSIXct`. This is mandatory if `header` is `FALSE`. This may be a vector of times, if `filename` is a vector of file names.

`deltat` the time between samples. (This is mandatory if `header=FALSE`.)

`originalCoordinate`
character string indicating coordinate system, one of `"beam"`, `"xyz"`, `"enu"` or `"other"`. (This is needed for the case of multiple files that were created by a data logger, because the header information is normally lost in such instances.)

`transformationMatrix`
transformation matrix to use in converting beam coordinates to xyz coordinates. This will over-ride the matrix in the file header, if there is one. An example is `rbind(c(2.710, -1.409, -1.299), c(0.071, 2.372, -2.442), c(0.344, 0.344, 0.344))`.

`type` character string indicating type of file, and used by `read.adv` to dispatch to one of the speciality functions.

`tz` character string indicating time zone to be assumed in the data.

`debug` a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies. For example, `read.adv.nortek()` calls `read.header.nortek()`, so that `read.adv.nortek(...,debug=2)` provides information about not just the main body of the data file, but also the details of the header.

`monitor` boolean, set to `TRUE` to provide an indication of every data burst read.

`processingLog` if provided, the action item to be stored in the log. This parameter is typically only provided for internal calls; the default that it provides is better for normal calls by a user.

Details

Reads a binary-format ADV file. This is straightforward for files with headers, since the headers contain all the information required for further processing.

Files *without* headers may be created in experiments in which a data logger was set up to monitor the serial data stream from an instrument. The lack of header information places a burden on the user, who must supply such basic information as the times of observations, the instrument orientation, the instrument coordinate system, etc. Example 3 below shows how to deal with such files. Three things should be noted.

1. The use must choose the appropriate `read.adv` variant corresponding to the instrument in question. (This is necessary because `oceMagic`, which is used by the generic `read.oce` routine, cannot determine the type of instrument by examining a file that lacks a header.)

2. The call to the read function must include a start time (start) and the number of seconds between data (deltat), again, because the instrument data stream may lack those things when the device is set to a serial mode. Also, of course, it is necessary to set header=FALSE in the function call.
3. Once the file has been read in, the user will be obliged to specify other information, for the object to be well-formed. For example, the read function will have no way of knowing the instrument orientation, the coordinate system being used, the transformation matrix to go from "beam" to "xyz" coordinates, or the instrument heading, pitch, and roll, to go from "xyz" coordinates to "enu" coordinates. Such things are illustrated in example 3 below.

In ADV data files, velocities are coded to signed 2-byte integers, with a scale factor being used to convert to velocity in metres per second. These two facts control the maximum recordable velocity and the velocity resolution, values that may be retrieved for an ADV object name d with d[["velocityMaximum"]] and d[["velocityResolution"]].

Value

An object of class "adv", which contains measurements made with an ADV device. This is a list containing lists named metadata, data, and processingLog.

The metadata contains information as given in the following table. The "Nortek name" is the name used in the Nortek System Integrator Guide [reference 1] and the "Sontek name" is the name used in the relevant Sontek documentation. References are given in square brackets.

metadata **name**	**Nortek name**	**Sontek name**	**Meaning**
manufacturer	-	-	Either "nortek" or "sontek"
instrumentType	-	-	Either "vector" or "adv"
filename	-	-	Name of data file(s)
latitude	-	-	Latitude of mooring (if applicable)
longitude	-	-	Longitude of mooring (if applicable)
numberOfSamples	-	-	Number of data samples in file
numberOfBeams	NBeams [1 p18]	-	Number of beams (always 3)
numberOfBeamSequencesPerBurst	NPings	-	number of beam sequences per burst
measurementInterval	MeasInterval [1 p31]	-	
samplingRate	512/(AvgInterval) [1 p30; 4]	-	

The data list contains items with names corresponding to adp objects, with an exception for Nortek data. Nortek instruments report some things at a time interval that is longer than the velocity sampling, and these are stored in data as timeSlow, headingSlow, pitchSlow, rollSlow, and temperatureSlow; if burst sampling was used, there will also be items recordsBurst and timeBurst.

The processingLog is in the standard format.

Special considerations for NorTek files

The data format is inferred from the System Integrator Guide [1]. This document lacks clarity in spots, and so read.adv.nortek contains some assumptions that are noted here, so that users will be aware of possible problems.

A prominent example is the specification of the sampling rate, stored in `metadata$sampingRate` in the return value. Repeated examination of the System Integrator Guide [1] failed to indicate where this value is stored in the various headers contained in Vector datasets. After some experimentation with a few data files, `read.adv.nortek` was set up to calculate `metadata$samplingRate` as `512/AvgInterval` where `AvgInterval` is a part of the "User Configuration" header [1 p30], where the explanation is "average interval in seconds"). This formula was developed through trial and error, but it was confirmed later on the Nortek discussion group, and it should appear in upcoming versions of [1].

Another basic issue is the determination of whether an instrument had recorded in continuous mode or burst mode. One might infer that `TimCtrlReg` in the "User Configuration" header [1 p30] determines this, in bits 1 and 2. However, this was the case in test files available to the author. For this reason, `read.adv.nortek` infers the mode by reverse engineering of data files of known configuration. The present version of `read.adv.nortek` determines the sampling mode from the "`NRecords`" item of the "Vector Velocity Data" header, which seems to be 0 for data collected continuously, and non-zero for data collected in bursts.

Taking these things together, we come upon the issue of how to infer sampling times for Nortek instruments. There do not seem to be definitive documents on this, and so `read.adv.nortek` is based partly on information (of unknown quality) found on Nortek discusson boards. The present version of `read.adv.nortek` infers the times of velocity observations differently, depending on whether the instrument was set to record in burst mode or continuous mode. For burst mode, times stated in the burst headers are used, but for continous mode, times stated in the "vector system data" are used. On the advice found on a Nortek discussion board, the burst-mode times are offset by 2 seconds to allow for the instrument warm-up period.

Special considerations for Sontek files

The binary format is inferred from Appendix 2.2.3 of the Sontek ADV operation Manual [3], with the following exceptions and notes.

1. The documentation says sampling rate is in units of 0.1Hz, but a test file indicates that it is in 0.01 Hz.
2. Bursts are recognized by byte sequences [2 p95]. In each case, a signalling byte is to be followed by a certain number of bytes, and so this code checks for two-byte sequences. The are as follows:
   - `c(0x81,0x12)` for an ADV with no optional sensors installed.
   - `c(0x83,0x18)` if a compass/tilt sensor is installed, but no temperature or pressure sensors.
   - `c(0x85,0x16)` if temperature and/or pressure sensors are installed, but no compass/tilt sensor.
   - `c(0x87,0x1c)` if a compass/tilt sensor is installed in addition to temperature and/or pressure sensors.

   Bug: only the second-last of these is handled in the present version of the package.

Bugs

The `by` argument only has an effect on quickly-varying variables, such as the fast timescale, velocity, backscatter, and amplitude It has no effect on slowly-varying variables, such as heading,

temperature, etc. And, for the Nortek case, it also has no effect on the burst information. The reason for all of this is that it is not altogether clear what by *should* mean, for those variables. Indeed, this is an argument for deleting by from the argument list, and this may be done in a future version of `read.adv`.

Author(s)

Dan Kelley

References

1. Nortek AS. System Integrator Guide (paradopp family of products). June 2008. (Doc No: PSI00-0101-0608). (Users may find it helpful to also examine newer versions of the guide.)

2. SonTek/YSI ADVField/Hydra Acoustic Doppler Velocimeter (Field) Technical Documentation (Sept 1, 2001).

3. Appendix 2.2.3 of the Sontek ADV operation Manual, Firmware Version 4.0 (Oct 1997).

4. Nortek Knowledge Center http://www.nortekusa.com/en/knowledge-center

See Also

The documentation for `adv-class` explains the structure of ADV objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
# A nortek Vector file
d <- read.oce("/data/archive/sleiwex/2008/moorings/m05/adv/nortek_1943/raw/adv_nortek_1943.vec",
              from=as.POSIXct("2008-06-26 00:00:00", tz="UTC"),
              to=as.POSIXct("2008-06-26 00:00:10", tz="UTC"))
plot(d, which=c(1:3,15))

## End(Not run)
```

---

read.cm *Read a current-meter data file*

Description

Read a current-meter data file, producing an object of type `cm`.

Usage

```
read.cm(file, from=1, to, by=1, tz=getOption("oceTz"),
        type=c("s4"),
        longitude=NA, latitude=NA,
        debug=getOption("oceDebug"), monitor=FALSE, processingLog, ...)
```

Arguments

file	a connection or a character string giving the name of the file to load.
from	index number of the first measurement to be read, or the time of that measurement, as created with as.POSIXct (hint: use tz="UTC").
to	indication of the last measurement to read, in a format matching that of from.
by	an indication of the stride length to use while walking through the file. If this is an integer, then by-1 measurements are skipped between each pair of profiles that is read. This may not make much sense, if the data are not equi-spaced in time. If by is a string representing a time interval, in colon-separated format, then this interval is divided by the sampling interval, to get the stride length. *BUG:* if the data are not equi-spaced, then odd results will occur.
longitude	optional signed number indicating the longitude in degrees East.
latitude	optional signed number indicating the latitude in degrees North.
type	character string indicating type of file (ignored at present).
tz	character string indicating time zone to be assumed in the data.
debug	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies.
monitor	ignored at present.
processingLog	if provided, the action item to be stored in the log. This parameter is typically only provided for internal calls; the default that it provides is better for normal calls by a user.
...	optional arguments passed to plotting functions.

Details

This function has been tested on only a single file, and the data-scanning algorithm was based on visual inspection of that file. Whether it will work generally is an open question.

Value

An object of class "cm", which contains measurements made with a current-meter device. For information on data stored in the object, see "Details".

Author(s)

Dan Kelley

See Also

The documentation for cm-class explains the structure of CM objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
cm <- read.oce("cm_interocean_0811786.s4a.tab")
summary(cm)
plot(cm)

## End(Not run)
```

`read.coastline` *Scan a coastline data file*

Description

Read a coastline file in mapgen, matlab, Splus, or shapefile format.

Usage

```
read.coastline(file, type=c("R","S","mapgen","shapefile","openstreetmap"),
  debug=getOption("oceDebug"), monitor=FALSE, processingLog)
read.coastline.shapefile(file, lonlim=c(-180,180), latlim=c(-90,90),
  debug=getOption("oceDebug"), monitor=FALSE, processingLog)
read.coastline.openstreetmap(file, lonlim=c(-180,180), latlim=c(-90,90),
  debug=getOption("oceDebug"), monitor=FALSE, processingLog)
```

Arguments

`file`	name of file containing coastline data.
`type`	type of file, one of `"R"`, `"S"`, `"mapgen"`, `"shapefile"` or `"openstreetmap"`.
`debug`	set to TRUE to print information about the header, etc.
`latlim`	range of (signed) latitudes, used only for shapefiles. Regions that do not intersect this range are skipped.
`lonlim`	as `latlim`, but a signed longitude.
`monitor`	print a dot for every coastline segment read (ignored except for reading "shapefile" type)
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)

Details

The S and R formats are identical, and consist of two columns, lon and lat, with land-jump segments separated by lines with two NAs.

The MapGen format is of the form

```
# -b
-16.179081 28.553943
-16.244793 28.563330
```

BUG: the 'arc/info ungenerate' format is not yet understood.

Value

An object of `class` `"coastline"`, which is a `list` containing

`data`	a list containing
	longitude the longitude in decimal degrees positive east of Greenwich.
	latitude the latitude in decimal degrees positive north of the equator.
`metadata`	a NULL item that may be used in a future version.
`processingLog`	A processingLog of processing, in the standard `oce` format.

A hack for depth contours

The following demonstrates that this code is getting close to working with depth contours. But this should be handled more internally, and a new object for depth contours should be constructed, of which coastlines could be a subset.

```
 library(oce)
d <- read.coastline.shapefile("~/Dropbox/DepthContours/DepthContours.shp")
 isna <- is.na(d[["latitude"]])
 idx<-1+cumsum(isna)
 lat<-split(d[["latitude"]][!isna], idx[!isna])
 lon<-split(d[["longitude"]][!isna], idx[!isna])
 depths <- d[["depths"]]
 n <- length(depths)
 D<-200
 plot(c(-180,180),c(-90,90), xlab="", ylab="", asp=1, type='n')
 for (i in 1:n) {
     if (depths[i]==D)
         lines(lon[[i]],lat[[i]])
 }
```

Author(s)

Dan Kelley

References

The NOAA site http://www.ngdc.noaa.gov/mgg/coast/ is a popular source for coastline data files.

Information about Canadian coastlines in "shapefile" format is provided at http://coastalmap.marine.usgs.gov/GISdata/basemaps/canada/shoreline/canada_wvs_geo_wgs84.htm (link tested

Dec 2011 and Sep 2012). Data may be downloaded, after registration, from http://www.geobase.ca/geobase/en/search.do?produit=nrn&language=en (link tested December 2011). The "shapefile" format is described in *ESRI Shapefile Technical Description*, March 1998, available at http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf (link tested Dec 2011 and Sep 2012).

See Also

The documentation for coastline-class explains the structure of coastline objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
cl <- read.coastline("7404.dat")
# If no plot yet:
plot(cl)
# To add to an existing plot:
lon <- longitude(cl)
lat <- latitude(cl)
lines(lon, lat)
# Note: another trick is to do something like the following,
# to get issues of whether longitude is defined in (-180,180)
# or (0,360)
lines(lon, lat)
lines(lon-360, lat)

## End(Not run)
```

---

read.ctd *Read a CTD data file*

Description

Read a CTD data file, producing an object of type ctd.

Usage

```
read.ctd(file, type=NULL, columns=NULL, station=NULL,
  monitor=FALSE, debug=getOption("oceDebug"), processingLog, ...)
read.ctd.sbe(file, columns=NULL, station=NULL, missing.value,
  monitor=FALSE, debug=getOption("oceDebug"), processingLog, ...)
read.ctd.woce(file, columns=NULL, station=NULL, missing.value=-999,
  monitor=FALSE, debug=getOption("oceDebug"), processingLog, ...)
read.ctd.woce.other(file, columns=NULL, station=NULL, missing.value=-999,
  monitor=FALSE, debug=getOption("oceDebug"), processingLog, ...)
read.ctd.odf(file, columns=NULL, station=NULL, missing.value=-999,
  monitor=FALSE, debug=getOption("oceDebug"), processingLog, ...)
```

```
read.ctd.itp(file, columns=NULL, station=NULL, missing.value=-999,
  monitor=FALSE, debug=getOption("oceDebug"), processingLog, ...)
```

Arguments

file	a connection or a character string giving the name of the file to load. For read.ctd.sbe() and read.ctd.woce(), this may be a wildcard (e.g. "*.cnv" or "*.csv") in which case the return value is a vector containing CTD objects created by reading the files from list.files with pattern set to the specified wildcard pattern.
type	if NULL, then the first line is studied, in order to determine the file type. If type="SBE19", then a *Seabird 19* (or similar) CTD format is assumed. If type="WOCE" then a WOCE-exchange file is assumed. If type="ITP" then an ice-tethered profiler file is assumed.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
columns	if NULL, then read.ctd tries to infer column names from the header. For SBE files only, the column argument can control the column selection. It is a list that names data types and the columns containing them, starting at 1. The list must include "pressure", "temperature" and "salinity", with any other values being ignored (in this version of the function). Note that SBE headers count the "name" from zero, e.g. if the header line says # name 1 = prDM: Pressure, Digiquartz [db] then the correct specification to get this pressure would be columns=list(pressure=2, ...).
station	optional character string containing an identifying name (or number) for the station. (This can be useful if the routine cannot determine the name automatically, or if another name is preferred.)
missing.value	optional missing-value flag; data matching this value will be set to NA upon reading.
monitor	boolean, set to TRUE to provide an indication of progress. This is useful if filename is a wildcard.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
...	additional arguments, passed to called routines.

Details

These functions read CTD datasets that have been stored in common formats, and could be extended to accommodate other formats if needed. The basic function is read.ctd, which analyzes some of the file contents, and then calls one of the following, any of which can be called directly.

- read.ctd.sbe() reads files files created by Seabird CTD instruments. These are recognized by a first line with first ten characters "* Sea-Bird."
- read.ctd.woce() reads files stored in the exchange format used by the World Ocean Circulation Experiment (WOCE) (first 4 characters of the first line being "CTD,"), and also in a rarer format with the first 3 characters being "CTD" followed by a blank or the end of the line).

- `read.ctd.woce.other()` reads the format called "CTD" in the section of the archive websites named "Other formats." These data are stored in filenames ending `.WCT`, and they do not have a great deal of metadata (e.g. longitude), so the user is forced to infer such things from a separate file. Support for this data type is limited, e.g. requiring a header of a certain length and data columns in a certain order. Improvements are unlikely to be added to the function, since this data type seems to offer no advantages over the type handled by `read.ctd.woce()`.
- `read.ctd.odf()` reads files stored in Ocean Data Format, used in some Canadian hydrographic databases.

Different file types provide different meta-information. For example, the WOCE exchange format binds together the institute name and the initials of the chief scientist into a single string that `read.ctd` cannot parse, so both `object@metadata$institute` and `object@metadata$scientist` are left blank for WOCE files.

Value

An object of `class` `"ctd"`, which is a list with elements detailed below. The most important elements are the station name and position, along with the profile data that are contained in the data frame named `data`. (Other elements in the list may be deleted in future versions of the package, if they prove to be of little use in practice, or if they prove to have been idiosyncratic features of the particular files used in early development of `oce`.)

`data` a data table containing the profile data. The column names are discovered from the header, and may differ from file to file. For example, some CTD instruments may have a fluorometer connected, others may not. The order of the columns may vary from case to case, and so it is important to refer to them by name. The following vectors are normally present: `data$pressure`, `data$salinity`, `data$temperature`, and `data$sigmatheta`. (σ_θ is calculated using `swSigmaTheta`.)

`metadata` a list containing the following items

`header` the header itself, normally containing several dozen lines of information.

`filename` name of the file passed to `read.ctd`.

`filename.orig` name of the original file saved by the instrument (normally a hex file).

`system.upload.time` system upload time.

`ship` name of vessel from which the CTD was deployed.

`scientist` name of the scientist leading the work at sea.

`institute` name of the institute behind the work.

`address` the address of the institute where the scientist works.

`cruise` name of cruise.

`station` station number or name.

`date` date of lowering of CTD into the water.

`startTime` time when instrument started recording data.

`latitude` latitude, in decimal degrees, positive north of equator.

`longitude` longitude, in decimal degrees, positive if east of Greenwich and west of dateline.

`recovery` date of recovery of CTD.

`waterDepth` the water depth at the site.

`sampleInterval` time interval between samples, in seconds.

`processingLog` a processingLog of processing, in the standard `oce` format.

Implementation and extension

The functions attempt to infer a wide range of meta-information from file headers, but variations in these headers limit general application. For example, `read.ctd.sbe` handles water depths in any of the following forms, but ostensibly similar forms may not work.

- `"** DEPTH = 100"`
- `"** Water Depth:   40 m"`
- `"** Depth (m): 3447 "`
- `"** Depth: 16"`
- `"** Profondeur: 92"`

If water depth cannot be inferred from the header, `read.ctd` sets it to the maximum recorded pressure, and issues a warning to that effect.

Similar issues come up for essentially everything stored in CTD headers, and so if odd values are found (e.g. a station in the wrong hemisphere), there is a good chance that the format is not being handled correctly. Given the expense of collecting data, users are well-advised to check inferred values against the values in the data files, for at least on profile within a given cruise. Modifying the `read.ctd` code is not particularly difficult, and users are encouraged to examine the source code (in `R/ctd.R`) to see whether modification can help. Some experience with regular expressions and string manipulation may be needed; see `regexpr` and `sub`. Three sample files are provided with the package, in

- `system.file("extdata", "ctd.cnv", package="oce")`
- `system.file("extdata", "d200321-001.ctd", package="oce")`
- `system.file("extdata", "CTD_BCD2010666_01_01_DN.ODF", package="oce")`

and an examination of these in relationship with the existing code should help users to understand the present implementation, providing insights on extending it for their own data.

In many cases, CTD instruments are set up to report dates in English. This can cause a problem for users running in different locales, since e.g month names differ. Therefore, if you know your datafile is written in American-English notation, you might want to do `Sys.setlocale("LC_TIME", "en_US")` before you try to read the data.

Author(s)

Dan Kelley

References

The Sea-Bird SBE 19plus profiler is described at http://www.seabird.com/products/spec_sheets/19plusdata.htm. The company recommends the use of their own software, and perhaps for this reason it is difficult to find a specification for the data files. Inspection of data files led to most of the code used in Oce. If the company ever publishes standards for the data formats, of course Oce will be adjusted. In the meantime, it does a reasonable job in many instances.

The WOCE-exchange format is described at http://woce.nodc.noaa.gov/woce_v3/wocedata_1/whp/exchange/exchange_format_desc.htm, and a sample file is at http://woce.nodc.noaa.gov/woce_v3/wocedata_1/whp/exchange/example_ct1.csv

The ODF format, used by the Canadian Department of Fisheries and Oceans, is described at http://slgo.ca/app-sgdo/en/docs_reference/documents.html, and this was used as a base for `read.ctd.odf`. However, it was only a starting point, for examination of data files revealed many variants in the names of the data columns. If anything odd happens with ODF files (e.g. if they cannot be plotted), the first thing to do is to reread the files with `debug=1`, to see if column names were converted properly.

Ice-tethered profile (ITF) data are available at www.whoi.edu/itf; note that the present version only handles data in profiler-mode, not fixed-depth mode.

See Also

The documentation for `ctd-class` explains the structure of `ctd` objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
## Not run:
## Labrador Sea data, file 0001919.tar.gz from website
## http://www.nodc.noaa.gov/cgi-bin/OAS/prd/accession/download
d <- read.ctd.woce("*.csv")
data(coastlineWorld)
plot(coastlineWorld, clat=55, clon=-50, span=5000)
longitude <- sapply(d, function(stn) stn[['longitude']])
latitude <- sapply(d, function(stn) stn[['latitude']])
points(longitude, latitude, col='red')

## End(Not run)
```

`read.drifter` *Read a drifter data file*

Description

Read a drifter file, producing an object of type `drifter`.

Usage

```
read.drifter(file, debug=getOption("oceDebug"), processingLog, ...)
```

Arguments

file	a connection or a character string giving the name of the file to load.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
...	additional arguments, passed to called routines.

Details

This reads netCDF formatted ARGO data.

Value

An object of class "drifter".

Author(s)

Dan Kelley

References

http://www.argo.ucsd.edu/

See Also

The documentation for drifter-class explains the structure of drifter objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
d <- read.drifter("/data/OAR/6900388_prof.nc")
summary(d)
plot(d)

## End(Not run)
```

`read.echosounder` *Read an echosounder data file*

Description

Read an echosounder data file, producing an object of type `echosounder`.

Usage

```
read.echosounder(file, channel=1, soundSpeed=swSoundSpeed(35,10,50),
                 tz=getOption("oceTz"), debug=getOption("oceDebug"),
                 processingLog)
```

Arguments

`file`	a connection or a character string giving the name of the file to load.
`channel`	sequence number of channel to extract, for multi-channel files.
`soundSpeed`	sound speed, in m/s. (The documents on Biosonics instruments suggest that this could be inferred from values in the file header, but test files proved this to be false.)
`tz`	character string indicating time zone to be assumed in the data.
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`processingLog`	if provided, the action item to be stored in the log, typically only provided for internal calls.

Details

Reads a biosonics echosounder file. This function was written for and tested with single-beam, dual-beam, and split-beam Biosonics files of type V3, and may not work properly with other file formats.

Value

An object of `class` `"echosounder"` with standard slots `metadata`, `data` and `processingLog` that are described in the documentation for the object `echosounder-class`.

Bugs

Only the amplitude information (in counts) is determined. A future version of this funciton may provide conversion to dB, etc. The handling of dual-beam and split-beam files is limited. In the dual-beam cse, only the wide beam signal is processed (I think ... it could be the narrow beam, actually, given the confusing endian tricks being played). In the split-beam case, only amplitude is read, with the x-axis and y-axis angle data being ignored.

Author(s)

Dan Kelley, with help from Clark Richards

References

Various echousounder instruments provided by BioSonics are described at the company website, http://www.biosonicsinc.com/. The document listed as [1] below was provided to the author of this function in November 2011, which suggests that the data format was not changed since July 2010.

[1] Biosonics, 2010. DT4 Data File Format Specification. BioSonics Advanced Digital Hydroacoustics. July, 2010. SOFTWARE AND ENGINEERING LIBRARY REPORT BS&E-2004-07-0009-2.0.

See Also

The documentation for echosounder-class explains the structure of ctd objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
## Not run:
e <- read.echosounder('data.dt4')
plot(e)

## End(Not run)
```

---

read.gps *Scan a gps data file*

Description

Read a gps file in gpx format format.

Usage

```
read.gps(file, type, debug=getOption("oceDebug"), processingLog)
```

Arguments

file	name of file containing gps data.
type	type of file, which will be inferred from examination of the data if not supplied. In the present version, the only choice for type is "gpx".
debug	set to TRUE to print information about the header, etc.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)

Details

Reads GPX format files by simply finding all longitudes and latitudes; in other words, information on separate tracks, or waypoints, etc., is lost.

Value

An object of class "gps".

Author(s)

Dan Kelley

References

The GPX format is described at http://www.topografix.com/GPX/1/1/.

See Also

The documentation for gps-class explains the structure of gps objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
gps <- read.gps("~/Dropbox/dalwhoi.gpx")

## End(Not run)
```

---

read.landsat *Read a landsat data file*

Description

Read a landsat data file, producing an object of type landsat.

Usage

```
read.landsat(file, band=1:11, debug=getOption("oceDebug"))
```

Arguments

file	a connection or a character string giving the name of the file to load. This is a directory name containing the data.
band	the bands to read, by default all of the Landsat-8 bands. See 'Value' for the names of the bands.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

The `tiff` package must be installed for this to work.

The function follows a strict file naming code that works for Landsat-8 datasets but may fail otherwise. Also, it only works for detailed data (in TIF format), not for preview data (in JPG format).

Landsat data are provided on several websites, e.g. [1]. Reference [2] provides information on the bands.

The bands are stored in the object with names "aerosol" (band 1), "blue" (band 2), "green" (band 3), "red" (band 4), "nir" (band 5), "swir1" (band 6), "swir2" (band 7), "panchromatic" (band 8), "cirrus" (band 9), "tirs1" (band 10), and "tirs2" (band 11). See [2] for information on these bands and their typical uses.

Value

An object of class `landsat`, with the conventional Oce slots `metadata`, `data` and `processingLog`. The `data` slot holds matrices of the data in the requested bands.

Author(s)

Dan Kelley

References

1. http://glovis.usgs.gov

2. http://landsat.gsfc.nasa.gov/?page_id=5377

See Also

See `landsat-class` for more information on `landsat` objects. A sample dataset named `landsat` is available in the `ocedata` package.

`read.lisst` *Read a LISST data file*

Description

Read a LISST data file, producing an object of type `lisst`.

Usage

```
read.lisst(file, year=0, tz="UTC", longitude=NA, latitude=NA)
```

Arguments

file	a connection or a character string giving the name of the file to load.
year	year in which the measurement of the series was made.
tz	time zone.
longitude	longitude of observation (stored in metadata)
latitude	latitude of observation (stored in metadata)

Details

The file should contain 42 columns, with no header. If there are fewer than 42 columns, an error results. If there are more, only the first 42 are used. Note that read.oce can recognize LISST files by their having a name ending in ".asc" and by having 42 elements on the first line. Even so, it is preferred to use the present function, because it gives the opportunity to specify the year and timezone, so that times can be calculated properly.

Value

An object of class "lisst", which is a list with slots as detailed below.

data	a data frame containing columns named C1 to C32, which are volume concentrations (in microliters per liter) for size classes 1 through 32, along with lts (laser transmission sensor, column 33), voltage (battery voltage in calibrated units, column 34), aux (external auxiliary input 1 in calibrated units, column 35), lrs (laster reference sensor in calibrated units, column 36), pressure (pressure in dbar, column 37), temperature (in degrees C, column 38), dayhour (day*100 +hour, column 39), minutesecond (minute*100+second, column 40), transmission (computed percent optical transmission over path, column 41), beam (computed beam-C in units of 1/m, column 42), and timea POSIXct time calculated from dayhour and minuteseconds, using the year specified as an argument, or the present year, if the year was not given, and using timezone tz.
metadata	a list containing the following items filename, the name of the file holding the data, latitude, the latitude of the observations, and longitude, the longitude of the observations.
processingLog	a processingLog of processing, in the standard oce format.

Author(s)

Dan Kelley

References

The LIST-100 users guide (version 4.65), which provided the information for this function, was downloaded in late May 2012, from http://www.sequoiasci.com/products/fam_LISST_100.aspx.

See Also

The documentation for `lisst-class` explains the structure of `lisst` objects, and also outlines the other functions dealing with them. In particular, `read.lisst` uses `as.lisst` to create the LISST object that is returned.

`read.lobo` *Read a lobo data file*

Description

Read a data file created by a LOBO instrument.

Usage

```
read.lobo(file, cols=7, processingLog)
```

Arguments

`file`	a connection or a character string giving the name of the file to load.
`cols`	number of columns in dataset.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)

Details

This version of read.lobo is really quite crude, having been developed mainly for a "predict the Spring bloom" contest at Dalhousie University. In particular, the function assumes that the data columns are exactly as specified in the Examples section; if you reorder the columns or add new ones, this function is unlikely to work correctly. Furthermore, it should be noted that the file format was inferred simply by downloading files; the supplier makes no claims that the format will be fixed in time.

It is also worth noting that there is no `read.oce` equivalent to `read.lobo`, because the file format has no recognizable header.

Value

An object of `class` `"lobo"`, which has the following slots:

`data`	a `list` containing the following vectors:
	`time` the times of observation
	`u` one horizontal component of velocity (necessarily eastward) [m/s]
	`v` an orthogonal component of horizontal velocity [m/s]
	`salinity` the salinity [PSU]
	`temperature` the in-situ temperature [degC]
	`p` the pressure [dbar]

	`nitrate` the nitrate concentration [unit?]
	`fluorescence` [unit?]
`metadata`	a list containing `header`, the header from the data file.
`processingLog`	a processingLog of processing, in the standard `oce` format.

Note

The `oce` author was unable to find a description of the data format, and so `read.lobo` makes some restrictive assumptions about the data format, e.g. trying to find columns whose names contain the following strings `"date"`, `"current across"`, `"current along"`, `"nitrate"`, `"fluorescence"`, `"salinity"`, `"temperature"` and `"Air temperature"`. The `data` slot in the return value will be filled in with these data, or with vectors with `NA` values, for unfound columns.

Author(s)

Dan Kelley

Source

The file was created as given in the example.

References

http://lobo.satlantic.com and http://www.mbari.org/lobo/

See Also

The documentation for `lobo-class` explains the structure of LOBO objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
uri <- paste("http://lobo.satlantic.com/cgi-bin/nph-data.cgi?",
  "min_date=20070220&max_date=20070305",
  "&x=date&",
  "y=current_across1,current_along1,nitrate,fluorescence,salinity,temperature&",
  "data_format=text",sep="")
lobo <- read.lobo(uri)

## End(Not run)
```

`read.met` *Read a meteorological data file*

Description

Read a meteorological data file

Usage

```
read.met(file, type=NULL, skip, tz=getOption("oceTz"),
         debug=getOption("oceDebug"), processingLog, ...)
```

Arguments

`file`	a connection or a character string giving the name of the file to load.
`type`	if `NULL`, then the first line is studied, in order to determine the file type. If `type="msc"`, then a file as formatted by the Meteorological Service of Canada is assumed.
`skip`	optional number of lines of header that occur before the actual data. If this is not supplied, `read.met` scans the file until it finds a line starting with `"Date/Time"`, and considers all lines above that to be header.
`tz`	timezone assumed for time data
`debug`	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
`processingLog`	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
`...`	additional arguments, passed to called routines.

Details

Reads a comma-separated value file in the format used by the Meterological Service of Canada (MSC). The agency does not publish a format for these files, so this function was based on a study of a few sample files, and it may fail for other files, if MSC changes the format.

Value

An object of `class` `"met"`, of which the `data` slot contains vectors `time`, `temperature`, `pressure`, `u`, and `v`.

Note

There seem to be several similar formats in use, so this function may not work in all cases.

Author(s)

Dan Kelley

See Also

The documentation for met-class explains the structure of meteorological objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
met <- read.met("ile-rouge-eng-hourly-06012008-06302008.csv")
plot(met, which=3:4)

## End(Not run)
```

---

read.observatory *Read observatory data file*

Description

Read observatory data file, e.g. produced by VENUS

Usage

```
read.observatory(file, type=c("ctd"),
                 debug=getOption("oceDebug"), processingLog, ...)
read.observatory.ctd(file,
                     debug=getOption("oceDebug"), processingLog, ...)
```

Arguments

file	a connection or a character string giving the name of the file to load.
type	type of data; must be "ctd", the only type handled at present.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
...	additional arguments, passed to called routines.

Details

These are a preliminary versions of functions that may be extended or deleted in a future version of Oce. Part of the uncertainty is that the format was inferred from inspection of files, and this is always a dangerous path (compared to following lear documentation on file formats) because the data archiver might alter formats in future.

Value

An object of class "ctd".

Author(s)

Dan Kelley

References

http://venus.uvic.ca/data/data-archive/

See Also

The documentation for ctd-class explains the structure of ctd objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
## Not run:
ctd <- read.oce("data.txt")
## mooring data are not profiles, so show timeseries
par(mfrow=c(3,1))
oce.plot.ts(ctd[["time"]], ctd[["pressure"]])
oce.plot.ts(ctd[["time"]], ctd[["salinity"]])
oce.plot.ts(ctd[["time"]], ctd[["temperature"]])

## End(Not run)
```

---

read.oce *Read an oceanographic data file*

Description

Read an oceanographic data file, auto-discovering the file type from the first line of the file.

Usage

```
read.oce(file, ...)
```

Arguments

file	a connection or a character string giving the name of the file to load.
...	arguments to be handed to whichever instrument-specific reading function is selected, based on the header.

Details

This function tries to infer the file type from the first line, using oceMagic. If it can be discovered, then an instrument-specific file reading function is called, with the file and with any additional arguments being supplied.

Value

An object of base class "oce", and also with a class that signifies the type of data, e.g. "ctd", "sealevel", etc.

Author(s)

Dan Kelley

See Also

The file type is determined by oceMagic. If the file type can be determined, then one of the following is called: read.ctd, read.coastline read.lobo, read.tdr, read.sealevel, etc.

Examples

```
library(oce)
x <- read.oce(system.file("extdata", "ctd.cnv", package="oce"))
plot(x) # summary with TS and profiles
plotTS(x) # just the TS
```

---

read.sealevel *Read a sea-level data file*

Description

Read a data file holding sea level data. BUG: the time vector assumes GMT, regardless of the GMT.offset value.

Usage

```
read.sealevel(file, tz=getOption("oceTz"),
              processingLog, debug=getOption("oceDebug"))
```

Arguments

file	a connection or a character string giving the name of the file to load. See Details for the types of files that are recognized.
tz	time zone. The default value, oceTz, is set to UTC at setup. (If a time zone is present in the file header, this will supercede the value given here.)
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
debug	set to TRUE to get debugging information during processing.

Details

This function starts by scanning the first line of the file, from which it determines whether the file is in one of two known formats: type 1, the format used at the Hawaii archive centre, and type 2, the comma-separated-value format used by the Marine Environmental Data Service. (The file type is inferred by checking for the existence of the string `Station_Name` on the first line of the file, indicating type 2.) If the file is in neither of these formats, the user might wish to scan it directly, and then to use `as.sealevel` to create a `sealevel` object.

Value

An object of `class` `"sealevel"`, which is a `list` containing

`data` A `data.frame` containing

- `time` time
- `eta` sea level [m]

`metadata` A list containing

- `header` the header line or lines. (This may prove helpful if detail extraction fails.)
- `year` year in which the observations were made.
- `station.number` identifier for the station.
- `station.version` see online docs at site mentioned in References.
- `station.name` name of station
- `region` a region code.
- `latitude` latitude, decimal degrees, positive north of equator.
- `longitude` longitude, decimal degrees, positive east of Greenwich.
- `GMT.offset` offset from GMT time. This is read directly from files of type 1. For files of type 2, the offset is inferred from the timezone, but this is a somewhat risky business, since timezone names are not standard.
- `decimation.method` see online docs at sites mentioned in References.
- `reference.offset` a reference offset; see online docs at sites mentioned in References.
- `reference.code` a reference code; see online docs at site mentioned in References.
- `units` unit of observations in the original file (normally "MM")
- `n` number of observations.
- `sampling.interval` hours between samples in the timeseries.

`processingLog` a processingLog of processing, in the standard `oce` format.

Author(s)

Dan Kelley

References

The Hawaii archive site at http://ilikai.soest.hawaii.edu/uhslc/datai.html provides a graphical interface for downloading sealevel data in Type 1 (the format is described at ftp://ilikai.soest.hawaii.edu/rqds/hourly.fmt). The MEDS repository provides Type 2 data, at http://www.meds-sdmm.dfo-mpo.gc.ca/meds/Databases/TWL/TWL_inventory_e.htm.

See Also

The documentation for sealevel-class explains the structure of sealevel objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
# this yields the sealevel dataset
sl <- read.oce("h275a96.dat")
summary(sl)
plot(sl)
m <- tidem(sl)
plot(m)

## End(Not run)
```

---

read.section *Read a section containing multiple CTD profiles*

Description

Read a file that contains a series of ctd profiles that make up an oceanographic section.

Usage

```
read.section(file, directory, sectionId="", flags,
             ship="", scientist="", institute="",
             missingValue=-999,
             debug=getOption("oceDebug"),
  processingLog)
```

Arguments

file	a file containing a set of CTD observations. At present, only the *exchange BOT* format is accepted (see Details).
directory	a directory that contains a set of CTD files that hold individual stations in the section.
sectionId	optional string indicating the name for the section. If not provided, the section ID is determined by examination of the file header.

ship	name of the ship carrying out the sampling
scientist	name of chief scientist aboard ship
institute	name of chief scientist's institute
flags	optional list of salinity-quality flags that will be demanded of the salinity data. If not given, an appropriate flag will be chosen based on the data type. For example, for WOCE data, any data with salinity flag not equal to 2 will be rejected. read.section tries to use the salinity from the CTD (with WOCE header name CTDSAL), but if its flag is unacceptable, the function tries SALNTY (yes, that is the spelling!) instead. But if neither has an acceptable flag, the datum will not be read.
missingValue	numerical value used to indicate missing data
debug	logical. If TRUE, print some information that might be helpful during debugging.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)

Details

Only *exchange BOT* comma-separated value format is permitted at this time, but other formats may be added later. It should also be noted that the parsing scheme was developed after inspection of the A03 data set (see Examples). This may cause problems if the format is not universal. For example, the header must name the salinity column "CTDSAL"; if not, salinity values will not be read from the file.

Value

An object of class "section", which is a list containing

data	a list containing just one item, station, which is a list of ctd objects, one per station.
metadata	a list containing: header, the header from the data file; sectionId, a name for the section; stationId, a vector of station IDs, one per station; latitude, a vector of station latitudes, decimal and positive in northern hemisphere; and longitude, a vector of station latitudes, decimal and positive in eastern hemisphere.
processingLog	a processingLog of processing, in the standard oce format.

Author(s)

Dan Kelley

References

Several repository sites provide section data. An example that is perhaps likely to exist for years is http://cchdo.ucsd.edu, but a search on "WOCE bottle data" should turn up other sites, if this one ceases to exist. Only the so-called *exchange BOT* data format can be processed by read.section() at this time.

See Also

The documentation for section-class explains the structure of section objects, and also outlines the other functions dealing with them.

---

read.tdr *Read temperature-depth recorder file*

Description

Read an tdr temperature-depth recorder file, producing an object of type tdr.

Usage

```
read.tdr(file, from=1, to, by=1, type, tz=getOption("oceTz"),
         processingLog, debug=getOption("oceDebug"))
```

Arguments

file	a connection or a character string giving the name of the file to load.
from	indication of the first datum to read. This can a positive integer to indicate sequence number, the POSIX time of the first datum, or a character string that can be converted to a POSIX time. (For POSIX times, be careful about the tz argument.)
to	an indication of the last datum to be read, in the same format as from. If to is missing, data will be read to the end of the file.
by	an indication of the stride length to use while walking through the file. If this is an integer, then by-1 profiles are skipped between each pair of profiles that is read. If this is a string representing a time interval, in colon-separated format (HH:MM:SS or MM:SS), then this interval is divided by the sampling interval, to get the stride length.
type	optional file type, presently only permitted to be rsk.
tz	time zone. The default value, oceTz, is set to UTC at setup.
processingLog	if provided, the action item to be stored in the log. (Typically only provided for internal calls; the default that it provides is better for normal calls by a user.)
debug	a flag that can be set to TRUE to turn on debugging.

Details

Read a TDR (temperature-depth recorder) file. At the moment, four styles are understood: (1) text file with columns for temperature and pressure (with sampling times indicated in the header); (2) text file with four columns, in which the date the time of day are given in the first two columns, followed by the temperature, and pressure; (3) text file with five columns, in which depth in the water column is given after the pressure; (4) a new (and possibly still changeable) SQLite-based database format.

Value

An object of class "tdr", which is a list with elements detailed below.

data	a data table containing the time, temperature, and pressure data.
metadata	a list containing the following items
	header the header itself, as read from the input file.
	serial.number serial number of instrument, inferred from first line of the header.
	loggingStart start time for logging, inferred from the header. Caution: this is often not the first time in the data, because the data may have been subsetted.
	sample.period seconds between samples, inferred from the header. Caution: this is often not the sampling period in the data, because the data may have been decimated.
processingLog	a processingLog of processing, in the standard oce format.

Implementation notes

The end time for measurement (metadata$measurement.start) is inferred from the Logger time field in the header, not from Logging end. The datasets available to the author suggest this is the proper scheme when the recorders are turned off manually before the end time that was programmed in. In other cases, the assumption may or may not be correct. Still, the end time for subsampling (metdata$subsample.end) should be correct.

Author(s)

Dan Kelley

See Also

The documentation for tdr-class explains the structure of PT objects, and also outlines the other functions dealing with them.

---

read.topo *Read an topography file*

Description

Read a file that contains topographic data in the ETOPO2 dataset, as provided by the NOAA website.

Usage

```
read.topo(file, ...)
```

Arguments

file	name of a file containing an ETOPO2 dataset.
...	additional arguments, passed to called routines.

Value

An object of type class "topo", which is a list containing the following items.

data	a data frame containing lat, lon, and z
metadata	a list containing the source filename
processingLog	a processingLog, in the standard oce format.

Author(s)

Dan Kelley

References

http://www.ngdc.noaa.gov/mgg/gdas/gd_designagrid.html

Examples

```
## Not run:
library(oce)
topoMaritimes <- read.topo("topoMaritimes.asc")
plot(topographyMaritimes)

## End(Not run)
```

---

rescale *Rescale values to lie in a given range*

Description

Rescale values to lie in a given range

Usage

```
rescale(x, xlow, xhigh, rlow=0, rhigh=1, clip=TRUE)
```

Arguments

x	a numeric vector.
xlow	x value to correspond to rlow. If not given, it will be calculated as the minimum value of x
xhigh	x value to correspond to rhigh. If not given, it will be calculated as the maximum value of x
rlow	value of the result corresponding to x equal to xlow.
rhigh	value of the result corresponding to x equal to xhigh.
clip	logical, set to TRUE to clip the result to the range spanned by rlow and rhigh.

Details

This is helpful in e.g. developing a color scale for an image plot. It is not necessary that rlow be less than rhigh, and in fact reversing them is a good way to get a reversed colorscale for a plot.

Value

A new vector, which has minimum lim[1] and maximum lim[2].

Author(s)

Dan Kelley

Examples

```
library(oce)
# Fake tow-yow data
t <- seq(0, 600, 5)
x <- 0.5 * t
z <- 50 * (-1 + sin(2 * pi * t / 360))
T <- 5 + 10 * exp(z / 100)
palette <- oceColorsJet(100)
zlim <- range(T)
drawPalette(zlim=zlim, col=palette)
plot(x, z, type='p', pch=20, cex=3,
     col=palette[rescale(T, xlow=zlim[1], xhigh=zlim[2], rlow=1, rhigh=100)])
```

---

resizableLabel *Provide axis names in adjustable sizes*

Description

Provide axis names in adjustable sizes, e.g. using T instead of Temperature, and including units as appropriate.

Usage

```
resizableLabel(item=c("S", "T", "theta", "sigmaTheta",
  "conservative temperature", "absolute salinity",
  "nitrate", "nitrite", "oxygen", "phosphate", "silicate",
  "tritium", "spice", "fluorescence",
  "p", "z",
  "distance", "distance km", "along-track distance km",
  "heading", "pitch", "roll",
  "u", "v", "w", "speed", "direction",
  "eastward", "northward", "depth", "elevation",
  "latitude", "longitude",
  "frequency cph", "spectral density m2/cph"),
              axis=c("x", "y"))
```

Arguments

item	code for the label.
axis	which axis to use.

Details

Used by e.g. plot.ctd.

Value

A character string or expression, in either a long or a shorter format, for use in the indicated axis at the present plot size. Whether the unit is enclosed in parentheses or square brackets is determined by the value of getOption("oceUnitBracket"), which may be "[" or "(".

Author(s)

Dan Kelley

---

retime *Adjust the time within Oce object*

Description

Adjust the time within Oce object

Usage

```
retime(x, a, b, t0, debug=getOption("oceDebug"))
```

Arguments

x	an oce object (presently, this must be of class adv)
a	intercept [in seconds] in linear model of time drift (see "Details").
b	slope [unitless] in linear model of time drift [unitless] (see "Details").
t0	reference time [in POSIXct format] used in linear model of time drift (see "Details").
debug	a flag that, if nonzero, turns on debugging.

Details

This function compensates for drifting instrument clocks, according to $t' = t + a + b(t - t0)$, where t' is the true time and t is the time stored in the object. A single check on time mismatch can be described by a simple time offset, with a non-zero value of a, a zero value of b, and an arbitrary value of t0. Checking the mismatch before and after an experiment yields sufficient information to specify a linear drift, via a, b, and t0. Note that t0 is just a convenience parameter, which avoids the user having to know the "zero time" used in R and clarifies the values of the other two parameters. It makes sense for t0 to have the same timezone as the time within x.

The returned object is computed by linear interpolation, using approx with rule=2, to avoid NA values at the start or end. The new time will be as given by the formula above. Note that if the drift is large enough, the sampling rate will be changed. It is a good idea to start with an object that has an extended time range, so that, after this is called, subset can be used to trim to a desired time range.

Value

A new object, with time and other data adjusted.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(adv)
adv2 <- retime(adv,0,1e-4,as.POSIXct("2008-07-01 00:00:00",tz="UTC"))
plot(adv[["time"]], adv2[["time"]]-adv[["time"]], type='l')
```

---

runlm *Calculate running linear models*

Description

Fit linear models y=y(x) in local regions of x space, possibly with distance-based Hanning weightings.

Usage

```
runlm(x, y, xout, window=c("hanning", "boxcar"), L, deriv)
```

Arguments

x	a vector holding x values.
y	a vector holding y values.
xout	optional vector of x values at which the derivative is to be found. If not provided, x is used.
window	type of weighting function used to weight data within the window; see 'Details'.
L	width of running window, in x units. If not provided, a reasonable default will be used.
deriv	an optional indicator of the desired return value; see 'Examples'.

Details

The linear model is calculated from the slope of a localized least-squares regression model y=y(x). The localization is defined by the x difference from the point in question, with data at distance exceeding L/2 being ignored. With a `boxcar` window, all data within the local domain are treated equally, while with a `hanning` window, a raised-cosine weighting function is used; the latter produces smoother derivatives, which can be useful for noisy data. The function is based on internal calculation, not on `lm`.

Value

If `deriv` is not specified, a list containing vectors of output values `y` and `y`, derivative (`dydx`), along with the scalar length scale `L`. If `deriv=0`, a vector of values is returned, and if `deriv=1`, a vector of derivatives is returned.

Author(s)

Dan Kelley

Examples

```
 library(oce)

# Case 1: smooth a noisy signal
x <- 1:100
y <- 1 + x/100 + sin(x/5)
yn <- y + rnorm(100, sd=0.1)
L <- 4
calc <- runlm(x, y, L=L)
plot(x, y, type='l', lwd=7, col='gray')
points(x, yn, pch=20, col='blue')
lines(x, calc$y, lwd=2, col='red')

# Case 2: square of buoyancy frequency
data(ctd)
```

```
par(mfrow=c(1,1))
plot(ctd, which="N2")
rho <- swRho(ctd)
z <- swZ(ctd)
zz <- seq(min(z), max(z), 0.1)
N2 <- -9.8 / mean(rho) * runlm(z, rho, zz, deriv=1)
lines(N2, -zz, col='red')
legend("bottomright", lwd=2, bg="white",
       col=c("black", "red"),
       legend=c("swN2()", "using runlm()"))
```

`sealevel` *Sea-level data set, from Halifax Harbour*

Description

This sample sea-level dataset is the 2003 record from Halifax Harbour in Nova Scotia, Canada. Note that Hurricane Juan hit Halifax a half hour before midnight on September 28th of 2003, causing a strong storm surge; see example at `plot.sealevel` in the Oce package. For reasons that are not mentioned on the data archive website, the record ends on the 8th of October.

Usage

```
data(sealevel)
```

Author(s)

Dan Kelley

Source

The data were downloaded from MEDS, and read with `read.oce` in the Oce package. The sign of the longitude was flipped before storing.

References

The MEDS archive is at http://www.meds-sdmm.dfo-mpo.gc.ca/meds/Databases/TWL/TWL_inventory_e.htm. An entry to the literature about Hurricane Juan is at http://www.atl.ec.gc.ca/weather/hurricane/juan/.

See Also

The documentation for `sealevel-class` in the Oce package explains the structure of sealevel objects, and also outlines the other functions dealing with them. Data for Tuktoyaktuk are stored in `sealevelTuktoyaktuk`.

Examples

```
## Not run:
library(oce)
data(sealevel)
plot(sealevel)
plot(sealevel, which=1, xlim=as.POSIXct(c("2003-01-01","2003-02-01")))

## End(Not run)
```

`sealevel-class` *Class to store sealevel data*

Description

Class to store sealevel data, e.g. from a tide gauge, with standard slots `metadata`, `data` and `processingLog`.

Methods

Data may be accessed as e.g. `sealevel[["time"]]`, where the string could also be e.g. `"elevation"` for the corresponding sea-level elevation, or e.g. `"longitude"` or `"latitude"` for scalars. (The names of the columns are displayed with `show()`.) The name of the source file is found with `"filename"`.

Everything that may be accessed may also be assigned, e.g. `sealevel[["elevation"]] <- value`.

The `show` method displays information about the object, while `summary.sealevel` provides a statistical summary.

Author(s)

Dan Kelley

See Also

The documentation for `sealevel-class` explains the structure of sealevel objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(sealevel)
show(sealevel)
plot(sealevel)
```

`sealevelTuktoyaktuk` *Sea-level data set acquired in 1975 at Tuktoyaktuk*

Description

This sea-level dataset is provided with in Appendix 7.2 of Foreman (1977) and also with the `T_TIDE` package (Pawlowicz et al., 2002). It results from measurements made in 1975 at Tuktoyaktuk, Northwest Territories, Canada.

The data set contains 1584 points, some of which have NA for sea-level height.

Although Foreman's Appendix 7.2 states that times are in Mountain standard time, the timezone is set to `UTC` in the present case, so that the results will be similar to those he provides in his Appendix 7.3.

Usage

```
data(sealevelTuktoyaktuk)
```

Source

The data were based on the `T_TIDE` dataset, which in turn seems to be based on Appendix 7.2 of Foreman (1977). Minor editing was on file format, and then the `sealevelTuktoyaktuk` object was created using `as.sealevel` in the Oce package.

References

Foreman, M. G. G., 1977. Manual for tidal heights analysis and prediction. Pacific Marine Science Report 77-10, Institute of Ocean Sciences, Patricia Bay, Sidney, BC, 58pp.

Pawlowicz, Rich, Bob Beardsley, and Steve Lentz, 2002. Classical tidal harmonic analysis including error estimates in MATLAB using `T_TIDE`. Computers and Geosciences, 28, 929-937.

See Also

The documentation for `sealevel-class` in the Oce package explains the structure of sea-level objects, and also outlines the other functions dealing with them. Data for Halifax harbour are stored in `sealevel`.

Examples

```
## Not run:
library(oce)
data(sealevelTuktoyaktuk)
time <- sealevelTuktoyaktuk[["time"]]
elevation <- sealevelTuktoyaktuk[["elevation"]]
oce.plot.ts(time, elevation, type='l', ylab="Height [m]", ylim=c(-2,6))
legend("topleft", legend=c("Tuktoyaktuk (1975)","Detided"),
       col=c("black","red"),lwd=1)
tide <- tidem(sealevelTuktoyaktuk)
```

```
detided <- elevation - predict(tide)
lines(time, detided, col="red")

## End(Not run)
```

---

secondsToCtime *Time interval as colon-separated string*

Description

Convert a time interval to a colon-separated string

Usage

```
secondsToCtime(sec)
```

Arguments

sec	length of time interval in seconds.

Details

Strings are

Value

A string with a colon-separated time interval.

Author(s)

Dan Kelley

See Also

See ctimeToSeconds, the inverse of this.

Examples

```
library(oce)
cat("   10 s = ", secondsToCtime(10), "\n", sep="")
cat("   61 s = ", secondsToCtime(61), "\n", sep="")
cat("86400 s = ", secondsToCtime(86400), "\n", sep="")
```

`section` *Hydrographic section*

Description

This is line A03 (ExpoCode 90CT40_1, with nominal sampling date 1993-09-11). The chief scientist was Tereschenkov of SOI, working aboard the Russian ship Multanovsky, undertaking a westward transect from the Mediterranean outflow region across to North America, with a change of heading in the last few dozen stations to run across the nominal Gulf Stream axis.

Usage

```
data(section)
```

Author(s)

Dan Kelley

Source

This is based on a WOCE file downloaded from http://cchdo.ucsd.edu/data/onetime/atlantic/a03/a03_hy1.csv.

See Also

The documentation for `section-class` in the Oce package explains the structure of section objects, and also outlines the functions dealing with them.

Examples

```
## Not run:
library(oce)
# Gulf Stream
data(section)
GS <- subset(section, 109<=stationId&stationId<=129)
GSg <- sectionGrid(GS, p=seq(0, 5000, 100))
plot(GSg, map.xlim=c(-80,-60))

## End(Not run)
```

`section-class` *Class to store hydrographic section data*

Description

Class to store hydrographic section data, with standard slots `metadata`, `data` and `processingLog`.

Methods

The examples given below use `s` to represent the name of a section.

Extracting values: A `list` of stations is retrieved by `s[["station"]]`. Individual stations are retrieved by providing a station number as a second argument in the index, e.g. the first station is `s[["station", 1]]` (which is a `ctd-class` object).

Aggregated values of the quantities measured at each level of the CTD profiles contained within the section may be accessed as e.g. `section[["salinity"]]`. This works for any quantity whose name is present in the constituent profiles.

Since it is often useful to pair such quantities with locations, `section[["longitude"]]` and `section[["latitude"]]` return vectors with values repeated for each level in each CTD (see the `pairs()` call in the example section). If just one latitude or longitude is desired per station, e.g. `section[["latitude", "byStation"]]` may be used. Station-by-station values of dynamic height are provided by e.g. `section[["dynamic height"]]`. The depths (actually pressures) of all data are obtained from e.g. `section[["depth"]]`, and the distances along the transect (measured from the first station) are obtained from e.g. `section[["distance"]]`.

Assigning values: (This feature has yet to be added.)

Overview of contents: The `show` method (e.g. `show(section)`) displays information about the object.

Author(s)

Dan Kelley

See Also

Sections can be read with `read.section` or created with `read.section` or created from CTD objects by using `makeSection` or by combining an existing section with a CTD object with the + operator.

Sections may be sorted with `sectionSort`, subsetted with `subset.section`, smoothed with `sectionSmooth`, and gridded with `sectionGrid`. Gridded sections may be plotted with `plot.section`.

Statistical summaries are provided by `summary.section`, while overviews are provided by `show.section`.

The sample dataset `section` contains data along WOCE line A03.

Examples

```
library(oce)
data(section)
plot(section[['station', 1]])
pairs(cbind(z=-section[["pressure"]],T=section[["temperature"]],S=section[["salinity"]]))
## T profiles for first few stations in section, at common scale
par(mfrow=c(3,3))
Tlim <- range(section[["temperature"]])
ylim <- rev(range(section[["pressure"]]))
for (stn in section[["station",1:9]])
plotProfile(stn, xtype='temperature', ylim=ylim, Tlim=Tlim)
```

---

sectionGrid *Grid a section*

Description

Grid a section, by interpolating to fixed pressure levels.

Usage

```
sectionGrid(section, p, method="approx", debug=getOption("oceDebug"), ...)
```

Arguments

section	a section object containing the section to be gridded.
p	optional indication of the pressure levels to which interpolation should be done. If this is not supplied, the pressure levels will be calculated based on the typical spacing in the ctd profiles stored within section. If a single numerical value is provided, it is taken as a step for a seq() running from 0 to the maximum pressure (rounded to that step). If p="levitus", then the levels are set to be those of Levitus atlas, given by standardDepths, trimmed to the maximum pressure in section. If a list of numerical values is provided, then it is used as is.
method	the method to use to decimate data within the stations; see ctdDecimate, which is used for the decimation.
debug	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies. For example, read.adv.nortek() calls read.header.nortek(), so that read.adv.nortek(...,debug=2) provides information about not just the main body of the data file, but also the details of the header.
...	optional arguments to be supplied to ctdDecimate.

Details

The "approx", "boxcar" and "lm" methods are described in the documentation for ctdDecimate, which is used to do this processing. The default "approx" method is best for bottle data, the "boxcar" is best for ctd data, and the "lm" method is probably too slow to recommend for exploratory work, in which it is common to do trials with a variety of "p" values.

Value

An object of class "section" that contains stations whose pressure values match identically.

Author(s)

Dan Kelley

See Also

The documentation for section-class explains the structure of section objects, and also outlines the other functions dealing with them.

Examples

```
# Gulf Stream
library(oce)
data(section)
GS <- subset(section, 109<=stationId&stationId<=129)
GSg <- sectionGrid(GS, p=seq(0, 5000, 100))
plot(GSg, map.xlim=c(-80,-60))
```

---

sectionSmooth *Smooth a section*

Description

Smooth a section in the lateral (alpha version that may change)

Usage

```
sectionSmooth(section, method=c("spline", "barnes"), debug=getOption("oceDebug"), ...)
```

Arguments

section	a section object containing the section to be smoothed. For method="spline", the pressure levels must match for each station in the section.
method	specifies the method to use; see 'Details'.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.
...	optional extra arguments, passed to either smooth.spline or interpBarnes.

Details

This function should be used with caution, as should any operation that changes data. Although smoothing may be desirable to produce aesthetically-pleasing plots, it can also introduce artifacts that can lead to erroneous conclusions. The prudent analyst starts by comparing plots of the raw data with plots of the smoothed data.

For `method="spline"`, the section is smoothed using `smooth.spline` on individual pressure levels, with any parameters listed in `parameters` being passed to that function. If `df` is not present in `parameters`, then this function sets it to the number of stations divided by 5. Smoothing is done separately for temperature, salinity, and sigma-theta.

For the (much slower) `method="barnes"` method, smoothing is done across both horizontal and vertical coordinates, using `interpBarnes`. Any arguments in `...` being passed to that function; see 'Examples'.

Value

An object of `class` `"section"` that is smoother than the input section.

Author(s)

Dan Kelley

See Also

The documentation for `section-class` explains the structure of section objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(section)
gs <- subset(section, 109<=stationId&stationId<=129)
gsg <- sectionGrid(gs, p=seq(0, 5000, 150))
gss1 <- sectionSmooth(gsg, "spline", df=16)
plot(gss1)
gss2 <- sectionSmooth(gsg, "barnes", xr=24, yr=100)
plot(gss2)
```

---

sectionSort *Sort a section*

Description

Sort a section

Usage

```
sectionSort(section, by=c("stationId", "distance"))
```

Arguments

section	a section object containing the section whose stations are to be sorted
by	a string indicating how to reorder. Note that the "distance" method does not yet work ... also, eventually there will be other possibilities, e.g. "latitude".

Details

Sections created with makeSection have "stations" that are in the order of the CTD objects (or filenames for such objects) provided. Sometimes, this is not the desired order, e.g. if file names discovered with dir are in an order that makes no sense. (For example, a practioner might name stations "stn1", "stn2", etc., not realizing that this will yield an unhelpful ordering, by file name, if there are more than 9 stations.)

The purpose of sectionSort is to permit reordering the constituent stations in sensible ways.

Value

An object of class "section" that has less lateral variation than the input section.

Author(s)

Dan Kelley

See Also

The documentation for section-class explains the structure of section objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
# Eastern North Atlantic, showing Mediterranean water.
library(oce)
s <- subset(section, 1<=stationId&stationId<=60)
ss <- sectionGrid(s, p=seq(00, 5000, 100))
ss <- sectionSort(ss, by="stationId")
plot(ss, which=c(2,99), map.xlim=c(-75,0))

## End(Not run)
```

---

showMetadataItem *Show metadata item*

Description

Show metadata item

Usage

```
showMetadataItem(object, name, label="", postlabel="", isdate=FALSE, quote=FALSE)
```

Arguments

object	an object inheriting from the base oce class.
name	name of item
label	label to print before item
postlabel	label to print after item
isdate	boolean indicating whether the item is a time
quote	boolean indicating whether to enclose the item in quotes

Details

This is a helper function for various summary functions.

Value

None.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
showMetadataItem(ctd, "ship", "ship")
```

---

siderealTime *Convert a POSIXt time to a sidereal time*

Description

Convert a POSIXt time to a sidereal time

Usage

```
siderealTime(t)
```

Arguments

t	a time, in POSIXt format, e.g. as created by as.POSIXct, as.POSIXlt, or numberAsPOSIXct. If this is provided, the other arguments are ignored.

Details

The method is taken from Chapter 7 of Meeus (1982). The small correction that he discusses after his equation 7.1 is not applied here.

Value

A sidereal time, in hours in the range from 0 to 24.

Author(s)

Dan Kelley

References

Meeus, Jean, 1982. Astronomical formuae for Calculators. Willmann-Bell. Richmond VA, USA. 201 pages

Examples

```
t <- ISOdatetime(1978, 11, 13, 0, 0, 0, tz="UTC")
print(siderealTime(t))
```

---

soi *Southern Oscillation Index*

Description

This is the Southern Oscillation Index, downloaded in May 2014 and processed as follows.

```
d <- scan("http://www.cgd.ucar.edu/cas/catalog/climind/SOI.signal.annstd.ascii")
isYear <- d > 1800
index <- d[!isYear]
year <- 1/24 + seq(d[isYear][1], by=1/12, length.out=length(index))
## Trim -99.99 values
missing <- index < -90
year <- year[!missing]
index <- index[!missing]
soi <- data.frame(year=year, index=index)
```

Usage

```
data(soi)
```

Author(s)

Dan Kelley

Source

http://www.cgd.ucar.edu/cas/catalog/climind/SOI.signal.annstd.ascii

Examples

```
## Not run:
library(oce)
data(soi)
plot(soi$year, soi$index, type='l')

## End(Not run)
```

`standardDepths` *Standard oceanographic depths*

Description

Standard oceanographic depths

Usage

```
standardDepths()
```

Details

This returns so-called standard deptsh 0m, 10m, etc. below the sea surface.

Value

A vector of depths, c(0, 10, ...).

Author(s)

Dan Kelley

subset.adp *Subset an adp object*

Description

Subset an adp (acoustic Doppler profile) object

Usage

```
## S4 method for signature 'adp'
subset(x, subset, ...)
```

Arguments

x	a adp object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Details

This function is somewhat analogous to subset.data.frame. Subsetting can be by time or distance, but these may not be combined; use a sequence of calls to subset by both.

Value

A new adp object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(adp)
plot(adp)
#plot(subset(adp, distance < 10))
#plot(subset(adp, time < mean(range(adp[['time']]))))
```

subset.adv *Subset an adv object*

Description

Subset an adv (acoustic Doppler profile) object

Usage

```
## S4 method for signature 'adv'
subset(x, subset, ...)
```

Arguments

x	a adv object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Details

This function is somewhat analogous to subset.data.frame, except that subsets can only be specified in terms of time.

Value

A new adv object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(adv)
plot(adv)
plot(subset(adv, time < mean(range(adv[['time']]))))
```

subset.cm *Subset a cm object*

Description

Subset a cm (current meter) object

Usage

```
## S4 method for signature 'cm'
subset(x, subset, ...)
```

Arguments

x	a cm object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Details

This function is somewhat analogous to subset.data.frame.

Value

A new cm object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(cm)
plot(cm)
plot(subset(cm, time < mean(range(cm[['time']]))))
```

`subset.coastline` *Subset a coastline object*

Description

Subset a coastline object

Usage

```
## S4 method for signature 'coastline'
subset(x, subset, ...)
```

Arguments

`x`	a `coastline` object.
`subset`	a condition to be applied to the `data` portion of `x`. See 'Details'.
`...`	ignored.

Details

This function is somewhat analogous to `subset.data.frame`. Points on the coastline that are outside the subset region are set to `NA`. If the results are to be plotted with `plot.coastline`, it will be necessary to set the argument `fill=NULL` to avoid filling land areas.

Value

A new `coastline` object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(coastlineWorld)
plot(coastlineWorld)
a <- subset(coastlineWorld, longitude < 0)
points(a[['longitude']], a[['latitude']], col='red', pch=20, cex=1/2)
```

subset.ctd *Subset a ctd object*

Description

Subset a ctd object

Usage

```
## S4 method for signature 'ctd'
subset(x, subset, ...)
```

Arguments

x	a ctd object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Details

This function is somewhat analogous to subset.data.frame, but only one independent variable may be used in subset in any call to the function, which means that repeated calls will be necessary to, subset based on more than one independent variable (e.g. time and distance).

Value

A new ctd object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
plot(ctd)
plot(subset(ctd, pressure<10))
```

`subset.echosounder` *Subset an echosounder object*

Description

Subset an echosounder object

Usage

```
## S4 method for signature 'echosounder'
subset(x, subset, ...)
```

Arguments

`x`	a `echosounder` object.
`subset`	a condition to be applied to the `data` portion of `x`. See 'Details'.
`...`	ignored.

Details

This function is somewhat analogous to `subset.data.frame`. Subsetting can be by `time` or `depth`, but these may not be combined; use a sequence of calls to subset by both.

Value

A new `echosounder` object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(echosounder)
plot(echosounder)
plot(subset(echosounder, depth < 10))
plot(subset(echosounder, time < mean(range(echosounder[['time']]))))
```

`subset.lisst` *Subset a lisst object*

Description

Subset a lisst object

Usage

```
## S4 method for signature 'lisst'
subset(x, subset, ...)
```

Arguments

`x`	a `lisst` object.
`subset`	a condition to be applied to the `data` portion of `x`. See 'Details'.
`...`	ignored.

Details

This function is somewhat analogous to `subset.data.frame`, but subsetting is only permitted by time.

Value

A new `lisst` object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(lisst)
plot(lisst)
plot(subset(lisst, time < mean(range(lisst[['time']]))))
```

subset.oce *Subset an oce object*

Description

Subset an oce object

Usage

```
## S4 method for signature 'oce'
subset(x, subset, ...)
```

Arguments

x	a oce object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Details

This function is somewhat analogous to subset.data.frame.

Value

A new oce object.

Author(s)

Dan Kelley

subset.sealevel *Subset a sealevel object*

Description

Subset a sealevel object

Usage

```
## S4 method for signature 'sealevel'
subset(x, subset, ...)
```

Arguments

x	a sealevel object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Details

This function is somewhat analogous to subset.data.frame, but subsetting is only permitted by time.

Value

A new sealevel object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(sealevel)
plot(sealevel)
plot(subset(sealevel, time < mean(range(sealevel[['time']]))))
```

---

subset.section *Subset a section object*

Description

Subset a section object

Usage

```
## S4 method for signature 'section'
subset(x, subset, ...)
```

Arguments

x	a section object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	may include debug, to set a debugging level.

Details

This function is somewhat analogous to subset.data.frame. The condition set by subset may be in terms of stationId or any combination of longitude, latitude and time. However, stationId may not be combined with the others; to get that effect, call this function more than once.

Value

A new section object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(section)
GS <- subset(section, 109<=stationId&stationId<=129)
```

---

subset.tdr *Subset a tdr object*

Description

Subset a tdr (temperature-depth recorder) object

Usage

```
## S4 method for signature 'tdr'
subset(x, subset, ...)
```

Arguments

x	a tdr object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Details

This function is somewhat analogous to subset.data.frame, but subsetting is only permitted by time.

Value

A new tdr object.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(tdr)
plot(tdr)
plot(subset(tdr, time < mean(range(tdr[['time']]))))
```

---

subset.topo *Subset a topo object*

Description

Subset a topo (topography) object

Usage

```
## S4 method for signature 'topo'
subset(x, subset, ...)
```

Arguments

x	a topo object.
subset	a condition to be applied to the data portion of x. See 'Details'.
...	ignored.

Details

This function is somewhat analogous to subset.data.frame. Subsetting can be by time or distance, but these may not be combined; use a sequence of calls to subset by both.

Value

A new topo object.

Author(s)

Dan Kelley

Examples

```
## northern hemisphere
library(oce)
data(topoWorld)
plot(subset(topoWorld, latitude > 0))
```

subtractBottomVelocity *Subtract bottom velocity from ADP velocity*

Description

Subtract bottom velocity from ADP velocity

Usage

```
subtractBottomVelocity(x, debug=getOption("oceDebug"))
```

Arguments

x	an object of class "adp", which is in *beam* coordinates.
debug	a flag that, if non-zero, turns on debugging. Higher values yield more extensive debugging. This is passed to called functions, after subtracting 1.

Details

For now, the data must be in *beam* coordinates. This will be changed when requests are made for other coordinate systems.

Author(s)

Dan Kelley

See Also

See read.adp for notes on functions relating to "adp" objects, and adp-class for notes on the ADP object class.

summary.adp *Summarize an ADP object*

Description

Summarize data in an adp object.

Usage

```
## S4 method for signature 'adp'
summary(object, ...)
```

Arguments

object	an object of class "adp", usually, a result of a call to read.oce, read.adp.rdi, or read.adp.nortek.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented.

Value

An object of class "summary.adp", which contains pertinent information about the ADP record and its processing.

Author(s)

Dan Kelley

See Also

The documentation for adp-class explains the structure of ADP objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(adp)
summary(adp)
```

---

summary.adv *Summarize an ADV object*

Description

Summarize data in an adv object.

Usage

```
## S4 method for signature 'adv'
summary(object, ...)
```

Arguments

object	an object of class "adv", usually, a result of a call to read.adv.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented.

Value

An object of class `"summary.adv"`, which contains pertinent information about the ADV record and its processing.

Author(s)

Dan Kelley

See Also

The documentation for `adv-class` explains the structure of ADV objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(adv)
summary(adv)
```

`summary.cm` *Summarize a cm (current meter) object*

Description

Summarize a `cm` (current meter) object

Usage

```
## S4 method for signature 'cm'
summary(object, ...)
```

Arguments

`object`	an object of class `"cm"`, as read by `read.cm.s4`, perhaps called automatically by `read.cm` or `read.oce`.
`...`	further arguments passed to or from other methods.

Details

Pertinent summary information is presented.

Value

summary.cm returns an object of class "summary.cm", containing pertinent information about the cm object and its processing.

Author(s)

Dan Kelley

See Also

The documentation for cm-class explains the structure of CM objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(cm)
summary(cm)
```

---

summary.coastline *Summarize a coastline data object*

Description

Summarizes coastline length, bounding box, etc.

Usage

```
## S4 method for signature 'coastline'
summary(object, ...)
```

Arguments

object	an object of class "coastline", usually, a result of a call to read.coastline or read.oce.
...	further arguments passed to or from other methods.

Value

An object of class "summary.coastline", which contains pertinent information about the Lobo record and its processing.

Author(s)

Dan Kelley

References

http://www.ngdc.noaa.gov/mgg/shorelines/shorelines.html

See Also

The documentation for `coastline-class` explains the structure of coastline objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(coastlineWorld)
plot(coastlineWorld)
```

`summary.ctd` *Summarize a CTD object*

Description

Summarizes some of the data in a `ctd` object.

Usage

```
## S4 method for signature 'ctd'
summary(object, ...)
```

Arguments

`object`	an object of class `"ctd"`, usually, a result of a call to `read.ctd`, `read.oce`, or `as.ctd`.
`...`	further arguments passed to or from other methods.

Details

Pertinent summary information is presented, including the station name, sampling location, data ranges, etc.

Value

An object of class `"summary.ctd"`, which when printed shows pertinent information about the CTD record and its processing.

Author(s)

Dan Kelley

See Also

The documentation for ctd-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(ctd)
summary(ctd)
```

---

summary.drifter *Summarize a drifter object*

Description

Summarizes some of the data in a drifter object.

Usage

```
## S4 method for signature 'drifter'
summary(object, ...)
```

Arguments

object	an object of class "drifter", usually, a result of a call to read.drifter.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented.

Value

An object of class "summary.drifter", which contains pertinent information about the CTD record and its processing.

Author(s)

Dan Kelley

References

http://www.argo.ucsd.edu/

See Also

The documentation for drifter-class explains the structure of drifter objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(drifter)
summary(drifter)
```

`summary.echosounder` *Summarize an echosounder object*

Description

Summarizes some of the data in an `echosounder` object.

Usage

```
## S4 method for signature 'echosounder'
summary(object, ...)
```

Arguments

`object`	an object of class `"echosounder"`, usually, a result of a call to `read.echosounder`, `read.oce`, or `as.echosounder`.
`...`	further arguments passed to or from other methods.

Details

Pertinent summary information is presented, including the sampling location, data ranges, etc.

Value

`NULL`

Author(s)

Dan Kelley

See Also

The documentation for `echosounder-class` explains the structure of echosounder objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(echosounder)
summary(echosounder)
```

summary.gps *Summarize a gps object*

Description

Summarize a gps object

Usage

```
## S4 method for signature 'gps'
summary(object, ...)
```

Arguments

object	an object of class "gps"
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented.

Value

NULL

Author(s)

Dan Kelley

See Also

The documentation for gps-class explains the structure of GPS objects, and also outlines the other functions dealing with them.

summary.landsat *Summarize a landsat object*

Description

Summarizes some of the data in a landsat object.

Usage

```
## S4 method for signature 'landsat'
summary(object, ...)
```

Arguments

object	an object of class "landsat", usually, a result of a call to read.landsat.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented, including the station name, sampling location, data ranges, etc.

Value

An object of class "summary.landsat", which when printed shows pertinent information about the landsat record and its processing.

Author(s)

Dan Kelley

See Also

The documentation for landsat-class explains the structure of landsat objects, and also outlines the other functions dealing with them.

---

summary.lisst *Summarize a LISST object*

Description

Summarizes some of the data in a lisst object.

Usage

```
## S4 method for signature 'lisst'
summary(object, ...)
```

Arguments

object	an object of class "lisst", usually, a result of a call to read.lisst, read.oce, or as.lisst.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented, including the sampling times, data ranges, etc.

Value

NULL

Author(s)

Dan Kelley

References

The LIST-100 users guide (version 4.65), which provided the information for this function, was downloaded in late May 2012, from http://www.sequoiasci.com/products/fam_LISST_100.aspx.

See Also

The documentation for lisst-class explains the structure of LISST objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(lisst)
summary(lisst)
```

---

summary.lobo *Summarize a lobo data object*

Description

Summarizes some of the data in a lobo object.

Usage

```
## S4 method for signature 'lobo'
summary(object, ...)
```

Arguments

object	an object of class "lobo", usually, a result of a call to read.lobo or read.oce.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented, including the sampling interval, data ranges, etc.

Value

NULL

Author(s)

Dan Kelley

References

http://lobo.satlantic.com http://www.mbari.org/lobo/

See Also

The documentation for lobo-class explains the structure of LOBO objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(lobo)
summary(lobo)
```

---

summary.met *Summarize a met object*

Description

Summarizes some of the data in a met object.

Usage

```
## S4 method for signature 'met'
summary(object, ...)
```

Arguments

object	an object of class "met", usually, a result of a call to read.met, or as.met.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented, including the sampling location, data ranges, etc.

Value

NULL

Author(s)

Dan Kelley

See Also

The documentation for met-class explains the structure of met objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(met)
summary(met)
```

`summary.sealevel` *Summarize a sealevel object*

Description

Summarizes some of the data in a sealevel object.

Usage

```
## S4 method for signature 'sealevel'
summary(object, ...)
```

Arguments

`object`	an object of class `"sealevel"`, usually, a result of a call to `read.sealevel`, `read.oce`, or `as.sealevel`.
`...`	further arguments passed to or from other methods.

Details

Pertinent summary information is presented.

Value

An object of class `"summary.sealevel"`, which contains pertinent information about the sealevel record and its processing.

Author(s)

Dan Kelley

See Also

The documentation for `sealevel-class` explains the structure of sealevel objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(sealevel)
summary(sealevel)
```

summary.section *Summarize a CTD section*

Description

Summarize a CTD section.

Usage

```
## S4 method for signature 'section'
summary(object, ...)
```

Arguments

object	an object of class "section", usually, a result of a call to read.section, read.oce, or makeSection.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented, including station locations, distance along track, etc.

Author(s)

Dan Kelley

See Also

The documentation for section-class explains the structure of section objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(section)
summary(section)
```

summary.tdr *Summarize a tdr object*

Description

Summarize a tdr object

Usage

```
## S4 method for signature 'tdr'
summary(object, ...)
```

Arguments

object	an object of class "tdr", usually, a result of a call to read.tdr.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented, including the start and stop times, as well as statistics of temperature and pressure.

Value

NULL

Author(s)

Dan Kelley

See Also

The documentation for tdr-class explains the structure of PT objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(tdr)
summary(tdr)
```

summary.tidem *Summarize a tide-model object*

Description

summary method for class "tidem".

Usage

```
## S4 method for signature 'tidem'
summary(object, p, constituent, ...)
```

Arguments

object	an object of class "tidem", usually, a result of a call to tidem.
p	optional value of the maximum p value for the display of an individual coefficient. If not given, all coefficients are shown.
constituent	optional name of constituent on which to focus.
...	further arguments passed to or from other methods.

Details

By default, all fitted constituents are plotted, but it is quite useful to set e.g. p=0.05 To see just those constituents that are significant at the 5 percent level.

Note that the p values are estimated as the average of the p values for the sine and cosine components at a given frequency.

Value

NULL

Author(s)

Dan Kelley

See Also

tidem fits a tidal model to data (and describes the nature of tide objects), and plot.tidem plots a tide object.

Examples

```
## Not run:
library(oce)
data(sealevel)
tide <- tidem(sealevel)
summary(tide)

## End(Not run)
```

`summary.topo` *Summarize a topography data object*

Description

Summarizes some of the data in an `topo` object.

Usage

```
## S4 method for signature 'topo'
summary(object, ...)
```

Arguments

`object`	an object of class `"topo"`, usually, a result of a call to `read.topo` or `read.oce`.
`...`	further arguments passed to or from other methods.

Details

Pertinent summary information is presented, including the longitude and latitude range, and the range of elevation.

Value

`NULL`

Author(s)

Dan Kelley

See Also

A `topo` object may be read with `read.topo`.

Examples

```
library(oce)
data(topoWorld)
summary(topoWorld)
```

summary.windrose *Summarize a windrose data object*

Description

Summarizes some of the data in a windrose object.

Usage

```
## S4 method for signature 'windrose'
summary(object, ...)
```

Arguments

object	an object of class "windrose", usually, a result of a call to as.windrose.
...	further arguments passed to or from other methods.

Details

Pertinent summary information is presented.

Value

NULL

Author(s)

Dan Kelley

See Also

A windrose object may be created with as.windrose or plotted with plot.windrose.

Examples

```
library(oce)
xcomp <- rnorm(360) + 1
ycomp <- rnorm(360)
wr <- as.windrose(xcomp, ycomp)
summary(wr)
```

sunAngle *Solar angle as function of space and time.*

Description

Solar angle as function of space and time.

Usage

```
sunAngle(t, longitude=0, latitude=0, useRefraction=FALSE)
```

Arguments

t	Time, a POSIXt object (must be in timezone UTC), or a numeric value that corresponds to such a time.
longitude	observer longitude in degrees east
latitude	observer latitude in degrees north
useRefraction	boolean, set to TRUE to apply a correction for atmospheric refraction

Details

Based on NASA-provided Fortran program, in turn (according to comments in the code) based on "The Astronomical Almanac".

Value

A list containing the following.

time	time
azimuth	azimuth, in degrees eastward of north, from 0 to 360. (See diagram below.)
altitude	altitude, in degrees above the horizon, ranging from -90 to 90. (See diagram below.)
diameter	solar diameter, in degrees
distance	distance to sun, in astronomical units

Author(s)

Dan Kelley

References

Based on Fortran code retrieved from ftp://climate1.gsfc.nasa.gov/wiscombe/Solar_Rad/SunAngles/sunae.f on 2009-11-1. Comments in that code list as references:

Michalsky, J., 1988: The Astronomical Almanac's algorithm for approximate solar position (1950-2050), Solar Energy 40, 227-235

The Astronomical Almanac, U.S. Gov't Printing Office, Washington, D.C. (published every year).

The code comments suggest that the appendix in Michalsky (1988) contains errors, and declares the use of the following formulae in the 1995 version the Almanac:

- p. A12: approximation to sunrise/set times;
- p. B61: solar altitude (AKA elevation) and azimuth;
- p. B62: refraction correction;
- p. C24: mean longitude, mean anomaly, ecliptic longitude, obliquity of ecliptic, right ascension, declination, Earth-Sun distance, angular diameter of Sun;
- p. L2: Greenwich mean sidereal time (ignoring T^2, T^3 terms).

The code lists authors as Dr. Joe Michalsky and Dr. Lee Harrison (State University of New York), with modifications by Dr. Warren Wiscombe (NASA Goddard Space Flight Center).

See Also

The equivalent function for the moon is `moonAngle`.

Examples

```
rise <- as.POSIXct("2011-03-03 06:49:00", tz="UTC") + 4*3600
set <- as.POSIXct("2011-03-03 18:04:00", tz="UTC") + 4*3600
mismatch <- function(lonlat)
{
   sunAngle(rise, lonlat[1], lonlat[2])$altitude^2 + sunAngle(set, lonlat[1], lonlat[2])$altitude^2
}
result <- optim(c(1,1), mismatch)
lon.hfx <- (-63.55274)
lat.hfx <- 44.65
dist <- geodDist(result$par[1], result$par[2], lon.hfx, lat.hfx)
cat(sprintf("Infer Halifax latitude %.2f and longitude %.2f; distance mismatch %.0f km",
            result$par[2], result$par[1], dist))
```

---

swAbsoluteSalinity *Seawater absolute salinity, in TEOS-10*

Description

Compute seawater absolute salinity, according to TEOS-10.

Usage

```
swAbsoluteSalinity(salinity, pressure, longitude, latitude)
```

Arguments

salinity	either salinity (in which case pressure, etc. must be provided) **or** a ctd object (in which case salinity, etc. are determined from the object, and must not be provided in the argument list).
pressure	pressure in dbar.
longitude	longitude of observation.
latitude	latitude of observation.

Details

If the first argument is a ctd object, then values of salinity, etc, are extracted from it, and used for the calculation.

The absolute salinity is calculated using the TEOS-10 function gsw_sa_from_sp. Typically, this is a fraction of a unit higher than practical salinity as defined in the UNESCO formulae from the 1980s.

Value

Absolute salinity in g/kg.

Author(s)

Dan Kelley

References

[1] McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

[2] The TEOS-10 library is provided at www.teos-10.org.

[3] A programming interface to the matlab version of the TEOS library is provided at http://www.teos-10.org/pubs/gsw/html/gsw_contents.html.

See Also

The related TEOS-10 quantity "conservative temperature" may be computed with swConservativeTemperature. For a ctd object, absolute salinity may also be recovered by indexing as e.g. ctd[["absoluteSalinity"]] or ctd[["SA"]].

Examples

```
## Not run:
sa <- swAbsoluteSalinity(35.5, 300, 260, 16)
stopifnot(abs(35.671358392019094 - sa) < 00.000000000000010)

## End(Not run)
```

`swAlpha` *Seawater thermal expansion coefficient*

Description

Compute α, the seawater thermal expansion coefficient, as the product of α/β and β, each calculated from McDougall's (1987) algorithm.

Usage

```
swAlpha(salinity, temperature=NULL, pressure=NULL, isTheta=FALSE)
```

Arguments

`salinity`	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, `temperature` and `pressure` are determined from the object, and must not be provided in the argument list).
`temperature`	*in-situ* temperature [°C]
`pressure`	pressure [dbar]
`isTheta`	Set TRUE if t is theta or FALSE if t is in-situ

Details

If the first argument is a `ctd` object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Value

Value in 1/degC.

Author(s)

Dan Kelley

References

McDougall, T.J. 1987. "Neutral Surfaces" Journal of Physical Oceanography vol 17 pages 1950-1964

Examples

```
# 2.5060e-4 (inferred from p1964 of McDougall 1987)
a <- swAlpha(40, 10, 4000)
```

`swAlphaOverBeta` *Ratio of seawater thermal expansion coefficient to haline contraction coefficient*

Description

Compute α/β using McDougall's (1987) algorithm.

Usage

```
swAlphaOverBeta(salinity, temperature=NULL, pressure=NULL,
                  isTheta=FALSE)
```

Arguments

`salinity`	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, `temperature` and `pressure` are determined from the object, and must not be provided in the argument list).
`temperature`	*in-situ* temperature [°C]
`pressure`	pressure [dbar]
`isTheta`	Set TRUE if the temperature a potential temperature, FALSE otherwise.

Details

If the first argument is a `ctd` object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Value

Value in psu/°C.

Author(s)

Dan Kelley

References

McDougall, T.J. 1987. "Neutral Surfaces" Journal of Physical Oceanography vol 17 pages 1950-1964

Examples

```
# 0.3476 (from p1964 of McDougall 1987)
ab <- swAlphaOverBeta(40, 10, 4000, isTheta=TRUE)
```

swBeta *Seawater haline contraction coefficient*

Description

Compute β using McDougall's (1987) algorithm. NOTE: this conflicts with base::beta, so be sure to prefix as swBeta().

Usage

```
swBeta(salinity, temperature, pressure, isTheta=FALSE)
```

Arguments

salinity	either salinity [PSU] (in which case temperature and pressure must be provided) **or** a ctd object (in which case salinity, temperature and pressure are determined from the object, and must not be provided in the argument list).
temperature	*in-situ* temperature [°C]
pressure	seawater pressure [dbar]
isTheta	Set TRUE if temperature is theta or FALSE if temperature is in-situ

Details

If the first argument is a ctd object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Value

Value in 1/psu.

Author(s)

Dan Kelley

References

McDougall, T.J. 1987. "Neutral Surfaces" Journal of Physical Oceanography vol 17 pages 1950-1964

Examples

```
# 0.72088e-3 (from p1964 of McDougall 1987)
swBeta(40, 10, 4000, isTheta=TRUE)
```

swConductivity *Seawater thermal conductivity*

Description

Compute seawater thermal conductivity, in $Wm^{-1\circ}C^{-1}$

Usage

```
swConductivity(salinity, temperature=NULL, pressure=NULL)
```

Arguments

salinity	salinity [PSU], or a ctd object, in which case temperature and pressure will be ignored.
temperature	*in-situ* temperature [°C]
pressure	pressure [dbar]

Details

If the first item is ctd object, then the salinity, temperature and pressure values will be extracted from it, and used in the calculation.

Note that this is provisional code only!

To do: Fill in a summary of Caldwell's technique.

To do: Compare this with Caldwell's stated uncertainty.

Caution. The results differ from Fofonoff's (1962) table 5 by 0.1 percent at 35PSU, and by under 1 percent for fresh water.

Value

Conductivity of seawater in $Wm^{-1\circ}C^{-1}$.

To calculate thermal diffusivity in m^2/s, divide by the product of density and specific heat, as in the example.

Author(s)

Dan Kelley

References

Caldwell, Douglas R., 1974. Thermal conductivity of seawater, *Deep-sea Research*, **21**, 131-137.

Fofonoff, N. P., 1962. Physical properties of sea-water, *The Sea*, **1**, 3-30.

Examples

```
library(oce)
cond <- swConductivity(10,35,100); # 0.618569
diffusivity <- cond / (swRho(10,35,100) * swSpecificHeat(10,35,100))
```

`swConservativeTemperature` *Seawater conservative temperature, in TEOS-10.*

Description

Compute seawater conservative temperature, according to TEOS-10.

Usage

```
swConservativeTemperature(salinity, temperature, pressure)
```

Arguments

`salinity`	either salinity (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, etc. are determined from the object, and must not be provided in the argument list).
`temperature`	*in-situ* temperature [°C]
`pressure`	pressure [dbar]

Details

If the first argument is a `ctd` object, then values of salinity, etc, are extracted from it, and used for the calculation.

The conservative temperature is calculated using the TEOS-10 function `gsw_ct_from_t`.

Value

Conservative temperature in degrees Celcius.

Author(s)

Dan Kelley

References

[1] McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

[2] The TEOS-10 library is provided at www.teos-10.org.

[3] A programming interface to the matlab version of the TEOS library is provided at http://www.teos-10.org/pubs/gsw/html/gsw_contents.html.

See Also

The related TEOS-10 quantity "absolute salinity" may be computed with swAbsoluteSalinity.
For a ctd object, conservative temperature may also be recovered by indexing as e.g. ctd[["conservativeTemperature"]] or ctd[["CT"]].

Examples

```
## Not run:
ct <- swConservativeTemperature(35.7, 15, 300)
stopifnot(abs(14.930280459895560 - ct) < 00.000000000000010)

## End(Not run)
```

---

swDepth *Water depth*

Description

Compute depth from pressure and latitude.

Usage

```
swDepth(pressure, latitude=45, degrees=TRUE)
swZ(pressure, latitude=45, degrees=TRUE)
```

Arguments

pressure	either pressure [dbar], in which case lat must also be given, or a ctd object, in which case lat will be inferred from the object.
latitude	Latitude in °N or radians north of the equator.
degrees	Flag indicating whether degrees are used for latitude; if set to FALSE, radians are used.

Details

Depth is calculated from pressure using Saunders and Fofonoff's method, with the formula refitted for 1980 UNESCO equation of state (UNESCO, 1983).

Check value: 9712.653 m for 10,000 dbar at 30 deg latitude.

Value

For swDepth, depth below the ocean surface, in metres. For swZ, the negative of the depth, i.e. the distance above the surface.

Author(s)

Dan Kelley

References

Unesco 1983. Algorithms for computation of fundamental properties of seawater, 1983. *Unesco Tech. Pap. in Mar. Sci.*, No. 44, 53 pp.

Examples

```
d <- swDepth(10, 45)
```

---

swDynamicHeight *Dynamic height of seawater profile*

Description

Compute the dynamic height of a column of seawater.

Usage

```
swDynamicHeight(x, referencePressure=2000)
```

Arguments

`x`	a `section` object, **or** a `ctd` object.
`referencePressure`	reference pressure [dbar]

Details

If the first argument is a `section`, then dynamic height is calculated for each station within a section, and returns a list containing distance along the section along with dynamic height.

If the first argument is a `ctd`, then this returns just a single value, the dynamic height.

Stations are analysed in steps. First, a piecewise-linear model of the density variation with pressure is developed using `approxfun`. (The option `rule=2` is used to extrapolate the uppermost density up to the surface, preventing a possible a bias for bottle data, in which the first depth may be a few metres below the surface.) A second function is constructed as the density of water with salinity 35PSU, temperature of 0°C, and pressure as in the `ctd`. The reciprocal difference of these densities is then integrated using `integrate` with pressure limits `0` to `referencePressure`, and divided by the acceleration due to gravity, to calculate the dynamic height.

If the water column is too short to have bottom pressure exceeding `referencePressure`, a missing value is reported.

Value

In the first form, a list containing `distance`, the distance [km] from the first station in the section and `height`, the dynamic height [m].

In the second form, a single value, containing the dynamic height [m].

Author(s)

Dan Kelley

References

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

Examples

```
    ## Not run:
library(oce)

# Dynamic height and geostrophy
par(mfcol=c(2,2))
par(mar=c(4.5,4.5,2,1))

# Left-hand column: whole section
# (The smoothing lowers Gulf Stream speed greatly)
westToEast <- subset(section, 1<=stationId&stationId<=123)
dh <- swDynamicHeight(westToEast)
plot(dh$distance, dh$height, type='p', xlab="", ylab="dyn. height [m]")
ok <- !is.na(dh$height)
smu <- supsmu(dh$distance, dh$height)
lines(smu, col="blue")
f <- coriolis(section[["station", 1]][["latitude"]])
g <- gravity(section[["station", 1]][["latitude"]])
v <- diff(smu$y)/diff(smu$x) * g / f / 1e3 # 1e3 converts to m
plot(smu$x[-1], v, type='l', col="blue", xlab="distance [km]", ylab="velocity [m/s]")

# right-hand column: gulf stream region, unsmoothed
gs <- subset(section, 102<=stationId&stationId<=124)
dh.gs <- swDynamicHeight(gs)
plot(dh.gs$distance, dh.gs$height, type='b', xlab="", ylab="dyn. height [m]")
v <- diff(dh.gs$height)/diff(dh.gs$distance) * g / f / 1e3
plot(dh.gs$distance[-1], v, type='l', col="blue",
  xlab="distance [km]", ylab="velocity [m/s]")

## End(Not run)
```

---

swLapseRate *Seawater lapse rate*

Description

Compute adiabatic lapse rate

Usage

```
swLapseRate(salinity, temperature=NULL, pressure=NULL)
```

Arguments

salinity	either salinity [PSU] (in which case temperature and pressure must be provided) **or** a ctd object (in which case salinity, temperature and pressure are determined from the object, and must not be provided in the argument list).
temperature	*in-situ* temperature [°C]
pressure	pressure [dbar]

Details

If the first argument is a ctd object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Compute lapse rate using Fofonoff and Millard's (1983) formula.

Value

Lapse rate [degC/m].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp. (Section 7, pages 38-40)

Examples

```
lr <- swLapseRate(40, 40, 10000) # 3.255976e-4
```

---

swN2 *Squared buoyancy frequency for seawater*

Description

Compute N^2, the square of the buoyancy frequency for a seawater profile.

Usage

```
swN2(pressure, sigmaTheta=NULL, derivs, df, ...)
```

Arguments

pressure	either pressure [dbar] (in which case sigmaTheta must be provided) **or** an object of class ctd object (in which case sigmaTheta is inferred from the object.
sigmaTheta	Surface-referenced potential density minus 1000 [kg/m^3]
derivs	optional argument to control how the derivative $d\sigma_\theta/dp$ is calculated. This may be a character string or a function of two arguments. See "Details".
df	optional argument, passed to smooth.spline if this function is used for smoothing.
...	additional argument, passed to smooth.spline, in the case that derivs="smoothing". See "Details".

Details

If the first argument is a ctd object, then density is is inferred from it, and the sigmaTheta argument is ignored.

The core of the method involves differentiating density with respect to pressure, and the derivs argument is used to control how this is done, as follows.

- if derivs is not supplied, the action is as though it were given as the string "smoothing"
- if derivs equals "simple", then the derivative of density with respect to pressure is calculated as the ratio of first-order derivatives of density and pressure, each calculated using diff. (A zero is appended at the top level.)
- if derivs equals "smoothing", then the processing depends on the number of data in the profile, and on whether df is given as an optional argument. When the number of points exceeds 4, and when df exceeds 1, smooth.spline is used to calculate smoothing spline representation the variation of density as a function of pressure, and derivatives are extracted from the spline using predict. Otherwise, density is smoothed using smooth, and derivatives are calculated as with the "simple" method.
- if derivs is a function taking two arguments (first pressure, then density) then that function is called directly to calculate the derivative, and no smoothing is done before or after that call.

Given the derivative, the square of the buoyancy frequency is calculated as $10^{-4}g^2 d\sigma_\theta/dp$.

Value

Square of buoyancy frequency [$radian^2/s^2$].

Author(s)

Dan Kelley

Examples

```
library(oce)
data(ctd)
# Illustrate effect of changing df
p <- pressure(ctd)
```

```
plot(swN2(ctd), p, ylim=rev(range(p)), xlab="N2",ylab="p", type='l')
lines(swN2(ctd, df=3), p, col="blue")
grid()
```

`swPressure` *Water pressure*

Description

Compute seawater pressure from depth by inverting `swDepth` using `uniroot`.

Usage

```
swPressure(depth , latitude=45)
```

Arguments

`depth`	distance below the surface in metres.
`latitude`	Latitude in °N or radians north of the equator.

Details

Numerical inversion of `swDepth` is done using `uniroot`.

Value

Pressure in dbar.

Author(s)

Dan Kelley

References

Unesco 1983. Algorithms for computation of fundamental properties of seawater, 1983. *Unesco Tech. Pap. in Mar. Sci.*, No. 44, 53 pp.

Examples

```
swPressure(9712.653, 30) # should be 10km (check value in UNESCO 1983).
```

swRho *Seawater density*

Description

Compute, ρ, the *in-situ* density of seawater.

Usage

```
swRho(salinity, temperature=NULL, pressure=NULL, eos=c("unesco", "teos"),
      longitude, latitude)
```

Arguments

salinity	either salinity (in which case temperature and pressure must be provided) **or** a ctd object (in which case salinity, temperature and pressure are determined from the object, and must not be provided in the argument list).
temperature	*in-situ* temperature [°C]
pressure	pressure [dbar]
eos	equation of state to use, either unesco for the UNESCO formulation [1,2], or teos for the TEOS-10 formulation [3].
longitude	longitude of observation (only used if eos="teos").
latitude	latitude of observation (only used if eos="teos").

Details

If the first argument is a ctd object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

The density is calculated using the UNESCO equation of state for seawater, assuming that input variables are defined according to modern calibrations. In conventional Oceanographic notation, the calculated quantity is $\rho(S, t, p)$

Value

In-situ density [kg/m^3].

Author(s)

Dan Kelley

References

[1] Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

[2] Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

[3] McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

See Also

Related density routines include swSigma, swSigmaT, and swSigmaTheta.

Examples

```
library(oce)
rho <- swRho(35, 13, 1000)
```

---

swRrho *Density ratio*

Description

Compute density ratio

Usage

```
swRrho(ctd, sense=c("diffusive", "finger"), smoothingLength=10, df)
```

Arguments

ctd	an object of class ctd
sense	an indication of the sense of double diffusion under study and therefore of the definition of Rrho; see 'Details'
smoothingLength	ignored if df supplied, but otherwise the latter is calculated as the number of data points, divided by the number within a depth interval of smoothingLength metres.
df	if given, this is provided to smooth.spline.

Details

This computes Rrho (density ratio) from a ctd object, calculating derivatives from smoothing splines whose properties are governed by smoothingLength or df. If sense="diffusive" the definition is $(beta * dS/dz)/(alpha * d(theta)/dz)$ and the reciprocal for "finger".

If the default arguments are acceptable, ctd[["Rrho"]] may be used instead of swRrho(ctd).

The function returns NA if there are fewer than 4 depths in the profile, because it is not reasonable to fit a spline to such short profiles of salinity and temperature.

Value

Density ratio defined in either the "diffusive" or "finger" sense.

Author(s)

Dan Kelley and Chantelle Layton

Examples

```
library(oce)
data(section)
stn <- section[["station"]][[20]]
par(mfrow=c(1,3))
plotProfile(stn,"salinity")
plotProfile(stn,"temperature")
plotProfile(stn, "RrhoSF")
```

---

swSCTp *Salinity from electrical conductivity, temperature and pressure*

Description

Compute salinity based on electrical conductivity, temperature, and pressure.

Usage

```
swSCTp(conductivity, temperature, pressure, conductivityUnit=c("ratio", "mS/cm", "S/m"))
```

Arguments

`conductivity`	conductivity ratio [unitless]
`temperature`	*in-situ* temperature [°C]
`pressure`	pressure [dbar]
`conductivityUnit`	string indicating the unit used for conductivity; `ratio` means actual conductivity divided by 4.2914 S/m.

Details

Calculate salinity from what is actually measured by the CTD, *i.e.* conductivity, the *in-situ* temperature and pressure. Often this is done by the CTD processing software, but sometimes it is helpful to do this directly, *e.g.* when there is a concern about mismatches in sensor response times. Salinity is calculated using the UNESCO algorithm described by Fofonoff and Millard (1983).

Value

Practical salinity [PSU].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

See Also

For thermal (as opposed to electrical) conductivity, see `swConductivity`.

Examples

```
print(swSCTp(1,15,2000))   # 34.25045
print(swSCTp(1.2,20,2000)) # 37.24563
print(swSCTp(0.65,5,1500)) # 27.99535
```

`swSigma` *Seawater density anomaly*

Description

Compute σ, the *in-situ* density of seawater, minus 1000 kg/m^3.

Usage

```
swSigma(salinity, temperature=NULL, pressure=NULL)
```

Arguments

`salinity`	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, `temperature` and `pressure` are determined from the object, and must not be provided in the argument list).
`temperature`	*in-situ* temperature [°C]
`pressure`	pressure [dbar]

Details

If the first argument is a `ctd` object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Definition: $\rho(S, t, p)$ - 1000 kg/m^3.

Value

In-situ density anomaly [kg/m^3].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

See Also

Related density routines include: swRho to calculate in-situ density, swSigmaT to calculate an archaic quantity found in old textbooks, and swSigmaTheta to calculate potential density anomaly relative to the surface pressure. If you are thinking of using swSigma, you probably want swSigmaTheta.

Examples

```
sig <- swSigma(35, 13, 1000) # 30.818
```

---

swSigmaT *Seawater quasi-potential density anomaly*

Description

Compute σ_t, a rough estimate of potential density of seawater, minus 1000 kg/m^3.

Usage

```
swSigmaT(salinity, temperature=NULL, pressure=NULL)
```

Arguments

salinity	either salinity [PSU] (in which case temperature and pressure must be provided) **or** a ctd object (in which case salinity, temperature and pressure are determined from the object, and must not be provided in the argument list).
temperature	*in-situ* temperature [°C]
pressure	pressure [dbar]

Details

If the first argument is a ctd object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Definition: $\sigma_t = \rho(S, t, 0)$ - 1000 kg/m^3.

Value

Quasi-potential density anomaly [kg/m^3].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

See Also

Related density routines include: swRho, swSigma, and swSigmaTheta.

Examples

```
sigmat <- swSigmaT(35, 13, 1000)
```

---

swSigmaTheta *Seawater potential density anomaly*

Description

Compute σ_θ, the potential density of seawater, minus 1000 kg/m^3.

Usage

```
swSigmaTheta(salinity, temperature=NULL, pressure=NULL)
```

Arguments

salinity	either salinity [PSU] (in which case temperature and pressure must be provided) **or** a ctd object (in which case salinity, temperature and pressure are determined from the object, and must not be provided in the argument list).
temperature	*in-situ* temperature [°C]
pressure	pressure [dbar]

Details

If the first argument is a ctd object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Definition: $\sigma_\theta = \rho(S, \theta(S, t, p), 0$ - 1000 kg/m^3.

Value

Potential density anomaly [kg/m^3].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

See Also

Related density routines include: swRho, swSigma, and swSigmaT.

Examples

```
sigmaTheta <- swSigmaTheta(35, 13, 1000) #
```

---

swSoundAbsorption *Seawater sound absorption in dB/m*

Description

Compute the sound absorption of seawater, in dB/m

Usage

```
swSoundAbsorption(frequency, salinity, temperature, pressure, pH=8,
    formulation=c("fisher-simmons", "francois-garrison"))
```

Arguments

frequency	frequency of sound in Hz
salinity	salinity in PSU
temperature	*in-situ* temperature [°C]
pressure	water pressure in dbar
pH	seawater pH
formulation	specify the formulation to use; see references

Details

Salinity and pH are ignored in this formulation. Several formulae may be found in the literature, and they give results differing by 10 percent, as shown at [3] for example. For this reason, it is likely that more formulations will be added to this function, and entirely possible that the default may change.

Value

Sound absorption in dB/m.

Author(s)

Dan Kelley

References

[1] F. H. Fisher and V. P. Simmons, 1977. Sound absorption in sea water. J. Acoust. Soc. Am., 62(3), 558-564.

[2] R. E. Francois and G. R. Garrison, 1982. Sound absorption based on ocean measurements. Part II: Boric acid contribution and equation for total absorption. J. Acoust. Soc. Am., 72(6):1879-1890.

[3] http://resource.npl.co.uk/acoustics/techguides/seaabsorption/

Examples

```
## Fisher & Simmons (1977 table IV) gives 0.52 dB/km for 35 PSU, 5 degC, 500 atm
## (4990 dbar of water)a and 10 kHz
alpha <- swSoundAbsorption(35, 4, 4990, 10e3)

## reproduce part of Fig 8 of Francois and Garrison (1982 Fig 8)
f <- 1e3 * 10^(seq(-1,3,0.1)) # in KHz
plot(f/1000, 1e3*swSoundAbsorption(f, 35, 10, 0, formulation='fr'),
     xlab=" Freq [kHz]", ylab=" dB/km", type='l', log='xy')
lines(f/1000, 1e3*swSoundAbsorption(f, 0, 10, 0, formulation='fr'), lty='dashed')
legend("topleft", lty=c("solid", "dashed"), legend=c("S=35", "S=0"))
```

---

swSoundSpeed *Seawater sound speed*

Description

Compute the seawater speed of sound.

Usage

```
swSoundSpeed(salinity, temperature=NULL, pressure=NULL)
```

Arguments

`salinity`	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, `temperature` and `pressure` are determined from the object, and must not be provided in the argument list).
`temperature`	*in-situ* temperature [°C]
`pressure`	pressure [dbar]

Details

If the first argument is a `ctd` object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

The sound speed is calculated using the formulation in section 9 of Fofonoff and Millard (1983).

Value

Sound speed [m/s].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp. (See section 9.)

Examples

```
s <- swSoundSpeed(40, 40, 10000) # 1731.995 (p48 of Fofonoff + Millard 1983)
```

---

swSpecificHeat *Seawater specific heat*

Description

Compute specific heat of seawater.

Usage

```
swSpecificHeat(salinity, temperature=NULL, pressure=NULL)
```

Arguments

`salinity`	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, `temperature` and `pressure` are determined from the object, and must not be provided in the argument list).
`temperature`	*in-situ* temperature [°C]
`pressure`	seawater pressure [dbar]

Details

If the first argument is a `ctd` object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Based on matlab code on `ftp://acoustics.whoi.edu/pub/Matlab/oceans`, which was in turn based on Millero et al (1973, 1981).

Value

Specific heat $Jkg^{-1}\,{}^{\circ}C^{-1}$

Author(s)

Dan Kelley

References

Millero et. al., J. Geophys. Res. 78 (1973), 4499-4507

Millero et. al., UNESCO report 38 (1981), 99-188.

Examples

```
CP <- swSpecificHeat(40, 40, 10000) # 3949.500
```

`swSpice` *Seawater spiciness*

Description

Compute seawater "spice" (a variable orthogonal to density in TS space)

Usage

```
swSpice(salinity, temperature=NULL, pressure=NULL)
```

Arguments

`salinity`	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, `temperature` and `pressure` are determined from the object, and must not be provided in the argument list).
`temperature`	*in-situ* temperature [°C]
`pressure`	seawater pressure [dbar]

Details

If the first argument is a `ctd` object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

Roughly speaking, seawater with a high spiciness is relatively warm and salty compared with less spicy water. Another interpretation is that spice is a variable measuring distance orthogonal to isopycnal lines on TS diagrams (if the diagrams are scaled to make the isopycnals run at 45 degres). The definition used here is that of Pierre Flament. (Other formulations exist.) Note that pressure is ignored in the definition. Spiceness is sometimes denoted $\pi(S, t, p)$.

Value

Spice [kg/m^3].

Author(s)

Dan Kelley

References

P. Flament, 2002. A state variable for characterizing water masses and their diffusive stability: spiciness. *Progr. Oceanog.*, **54**, 493-501.

Examples

```
spice <- swSpice(35, 10, 0) # 1.1 by eye, from Flament's fig2
```

---

swSTrho *Seawater salinity from temperature and density*

Description

Compute *in-situ* salinity, given temperature, density, and pressure.

Usage

```
swSTrho(temperature, density, pressure, eos=getOption("eos", default="unesco"))
```

Arguments

`temperature`	*in-situ* temperature [°C]
`density`	*in-situ* density or sigma value [kg/m^3]
`pressure`	*in-situ* pressure [dbar]
`eos`	either `"unesco"` or `"teos"`. If the latter, then the computer must have the TEOS library installed; see `teos`.

Details

Finds the salinity that yields the given density, with the given temperature and pressure. The method is a bisection search with a salinity tolerance of 0.001. The isopycnal lines on temperature-salinity diagrams (`plotTS`) are computed with this function.

Value

In-situ salinity [PSU].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

See Also

swTSrho

Examples

```
S <- swSTrho(10, 22, 0) # 28.651
```

`swTFreeze` *Seawater freezing temperature*

Description

Compute freezing temperature of seawater.

Usage

```
swTFreeze(salinity, pressure=NULL)
```

Arguments

`salinity`	either salinity [PSU] or a `ctd` object from which salinity will be inferred.
`pressure`	seawater pressure [dbar]

Details

In the first form, the argument is a `ctd` object, from which the salinity and pressure values are extracted and used to for the calculation.

Value

Temperature [°C]

Author(s)

Dan Kelley

References

UNESCO tech. papers in the marine science no. 28. 1978 eighth report JPOTS Annex 6 freezing point of seawater F.J. Millero pp.29-35.

Examples

```
Tf <- swTFreeze(40, 500) # -2.588567 degC
```

---

swTheta *Seawater potential temperature*

Description

Compute θ, the potential temperature of seawater.

Usage

```
swTheta(salinity, temperature=NULL, pressure=NULL, referencePressure=0,
        method=c("unesco","bryden"))
```

Arguments

salinity	either salinity [PSU] (in which case temperature and pressure must be provided) **or** a ctd object (in which case salinity, temperature and pressure are determined from the object, and must not be provided in the argument list).
temperature	*in-situ* temperature [°C]
pressure	pressure [dbar]
referencePressure	reference pressure [dbar]
method	algorithm to be used (see details)

Details

If the first argument is a ctd object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

The potential temperature is defined to be the temperature that a water parcel of salinity S, *in-situ* temperature t and pressure p would have if were to be moved adiabatically to a location with pressure referencePressure. This quantity is commonly denoted θ in the oceanographic literature.

The "bryden" method, based on Bryden (1973), calculates potential temperature referenced to the surface, ignoring the reference pressure. The "unesco" method, based on Fofonoff *et al.* (1983), generalizes Bryden's method, permitting calculation for arbitrary reference pressure. For normal use, the "unesco" method is preferred. (The example given below illustrates that the two methods yield similar results for surface-referenced calculations.)

Value

Potential temperature [°C] of seawater.

Author(s)

Dan Kelley

References

Bryden, H. L., 1973. New polynomials for thermal expansion, adiabatic temperature gradient and potential temperature of seawater, *Deep-Sea Res.*, **20**, 401-408.

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp.

See Also

The corresponding potential density anomaly σ_θ can be calculated with swSigmaTheta.

Examples

```
library(oce)
cat("unesco example:",swTheta(35, 13, 1000),"\n")
cat("bryden example:",swTheta(35, 13, 1000, "bryden"),"\n")

print(swTheta(40,40,10000,0,"unesco")) # 36.89073 (Fofonoff et al., 1983)

# Demonstrate that the methods agree to a couple of
# millidegrees over a typical span of values
S <- c(30,30,38,38)
T <- c(-2,-2,30,30)
p <- 1000 * runif(n=4)
print(max(abs(swTheta(S,T,p) - swTheta(S,T,p,0,"bryden"))))


# Example from a cross-Atlantic section
data(section)
stn <- section[['station', 70]]
plotProfile(stn, 'theta', ylim=c(6000, 1000))
lines(stn[['temperature']], stn[['pressure']], lty=2)
legend("topleft", lty=1:2,
       legend=c("potential", "in-situ"),
       bg='white', title="Station 70")
```

---

swTSrho *Seawater temperature from salinity and density*

Description

Compute *in-situ* temperature, given salinity, density, and pressure.

Usage

```
swTSrho(salinity, density, pressure, eos=getOption("eos", default='unesco'))
```

Arguments

salinity	*in-situ* salinity [PSU]
density	*in-situ* density or sigma value [kg/m^3]
pressure	*in-situ* pressure [dbar]
eos	name of equation of state to be used, either "unesco" or "teos". If the latter, then the computer must have the TEOS library installed; see teos.

Details

Finds the temperature that yields the given density, with the given salinity and pressure. The method is a bisection search with temperature tolerance 0.001 $^\circ C$.

Value

In-situ temperature [$^\circ C$].

Author(s)

Dan Kelley

References

Fofonoff, P. and R. C. Millard Jr, 1983. Algorithms for computation of fundamental properties of seawater. *Unesco Technical Papers in Marine Science*, **44**, 53 pp

Gill, A.E., 1982. *Atmosphere-ocean Dynamics*, Academic Press, New York, 662 pp.

See Also

swSTrho

Examples

```
temperature <- swTSrho(35, 23, 0)
```

---

swViscosity *Seawater viscosity*

Description

Compute viscosity of seawater, in $Pa \cdot s$

Usage

```
swViscosity(salinity, temperature=NULL)
```

Arguments

`salinity`	either salinity [PSU] (in which case `temperature` and `pressure` must be provided) **or** a `ctd` object (in which case `salinity`, `temperature` and `pressure` are determined from the object, and must not be provided in the argument list).
`temperature`	*in-situ* temperature [°C]

Details

If the first argument is a `ctd` object, then salinity, temperature and pressure values are extracted from it, and used for the calculation.

The result is determined from a regression of the data provided in Table 87 of Dorsey (1940). The fit matches the table to within 0.2 percent at worst, and with average absolute error of 0.07 percent. The maximum deviation from the table is one unit in the last decimal place.

No pressure dependence was reported by Dorsey (1940).

Value

Viscosity of seawater in $Pa \cdot s$. Divide by density to get kinematic viscosity in m^2/s.

Author(s)

Dan Kelley

References

N. Ernest Dorsey (1940), *Properties of ordinary Water-substance*, American Chemical Society Monograph Series. Reinhold Publishing.

Examples

```
v <- swViscosity(30,10); # 0.001383779
```

`tdr` *tdr (temperature-data recorder) dataset*

Description

A sample object of Oce class `tdr-class`.

Usage

```
data(tdr)
```

Author(s)

Dan Kelley

Source

The SLEIWEX experiment.

References

http://myweb.dal.ca/kelley/SLEIWEX/index.php

See Also

The documentation for `tdr-class` in the Oce package explains the structure of `tdr` objects, and also outlines the other functions dealing with them.

Examples

```
## Not run:
library(oce)
data(tdr)
plot(tdr)

## End(Not run)
```

`tdr-class` *Class to store temperature-depth recorder data*

Description

Class to store temperature-depth recorder data, with standard slots `metadata`, `data` and `processingLog`.

Methods

Data may be accessed as e.g. `tdr[["time"]]`, where the string could also be `"pressure"` or `"temperature"`. Assignment to these can be made with e.g. `tdr[["pressure"]] <- value`, etc. Indeed, any quantity in the `metadata` slot or the `data` slot can be retrieved or updated in this way.

Author(s)

Dan Kelley

See Also

A `tdr` object may be read with `read.tdr` or created with `as.tdr`. Plots can be made with `plot.tdr`, while `summary.tdr` produces statistical summaries and `show` produces overviews. If atmospheric pressure has not been removed from the data, the functions `tdrPatm` may provide guidance as to its value. Similarly, if the record contains periods when the instrument was in the air, `tdrTrim` may prove useful in isolating the times when it was in the water. However, these last two functions are no equal to decent record-keeping at sea.

`tdrPatm` *Estimate atmospheric pressure in tdr record*

Description

Estimate atmospheric pressure in tdr record

Usage

```
tdrPatm(x, dp=0.5)
```

Arguments

`x`	A `tdr` object, or a list of pressures (in decibars).
`dp`	Half-width of pressure window to be examined (in decibars).

Details

Pressures must be in decibars for this to work. First, a subset of pressures is created, in which the range is `sap-dp` to `sap+dp`. Here, `sap`=10.1325 dbar is standard sealevel atmospheric pressure. Within this window, three measures of central tendency are calculated: the median, the mean, and a weighted mean that has weight given by $exp(-2 * ((p - sap)/dp)^2)$.

Value

A list of four estimates: `sap`, the median, the mean, and the weighted mean.

Author(s)

Dan Kelley

See Also

The documentation for `tdr-class` explains the structure of PT objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(tdr)
print(tdrPatm(tdr))
```

tdrTrim *Trim start/end portions of a tdr cast*

Description

Trim start/end portions of a tdr cast.

Usage

```
tdrTrim(x, method="water", parameters=NULL, debug=getOption("oceDebug"))
```

Arguments

x	A tdr object, e.g. as read by read.tdr.
method	Various methods exist, some of which use parameters: "water" Trim data at start and end that seem to be in air (i.e. that have pressure very near 10db, i.e 100kPa. "time" Select values only in indicated range of times, provided in POSIX format. "index" Select data only if the range of indices given in the two-element parameters list
parameters	Depends on method; see above.
debug	a flag that turns on debugging. The value indicates the depth within the call stack to which debugging applies. For example, read.adv.nortek() calls read.header.nortek(), so that read.adv.nortek(...,debug=2) provides information about not just the main body of the data file, but also the details of the header.

Details

The "water" method is mainly for quick and dirty work. In many cases, the user will be working with several files, and so it will make sense to use the "time" method, to synchronize the time series. Normally, notes will have been taken in the field, so no guessing need be done about the time the instruments went in the water. In some cases, though, the time will have to be inferred from the data, and so it might make sense to start by trimming with the "water" method, after which summary.tdr is used to find the probably time when the instrument was put into the water.

After the data are trimmed in time, the pressure record is modified by subtracting 10.1325~dbar, the average sea-level pressure.

Value

An object of class "tdr", with data having been trimmed in some way, and with the pressure being reduced by mean sea-level pressure.

Author(s)

Dan Kelley

See Also

The documentation for `tdr-class` explains the structure of PT objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(tdr)
tdrTrimmed <- tdrTrim(tdr)
plot(tdrTrimmed)
```

---

teos *Interface to TEOS-10 library*

Description

Interface to TEOS-10 library

Usage

```
teos(name, a1, a2, a3, a4, lib=getOption("libteos"))
```

Arguments

`name`	the name of a TEOS-10 function. Note that the function names in the TEOS-10 documentation are in mixed case, but those in the actual TEOS-10 library are *lower case*. Thus, for example, the function to computer absolute salinity must be named as `"gsw_sa_from_sp"`, not `"gsw_SA_from_SP"`, as the documentation specifies.
`a1`	the first argument to the TEOS-10 function
`a2`	the second argument to the TEOS-10 function
`a3`	the third argument to the TEOS-10 function
`a4`	the fourth argument to the TEOS-10 function
`lib`	the name of the shared library that provides TEOS-10.

Details

For teos to work, the user must download the C version of TEOS-10 from the website listed in the references, then type make install at the command line, and finally copy the resultant libgswteos-10.so file to the directory /usr/local/lib. For example, the steps listed below were used on the author's (OSX-based) machine.

```
curl -OL http://www.teos-10.org/software/gsw_c_v3_0.zip
unzip gsw_c_v3_0.zip
cd gsw-_v3.0
make library
sudo mv libgswteos-10.so /usr/local/lib
```

If the library is installed elsewhere ("/opt/lib/libgswteos-10.so", to take an example), it will be necessary to note that fact, by performing two actions:

```
option(libteos="/opt/lib/libgswteos10.so")
.C("set_libteos", "/opt/lib/libgswteos10.so")
```

Value

Whatever the TEOS-10 function returns, with dimensions matching those of a1. TEOS-10 functions use the value 9e15 to indicate invalid conditions (e.g. latitude exceeding 90 degrees), and these are converted by teos() to NA.

Author(s)

Dan Kelley

References

[1] McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127, ISBN 978-0-646-55621-5.

[2] The TEOS-10 library is provided at www.teos-10.org.

[3] A programming interface to the matlab version of the TEOS library is provided at http://www.teos-10.org/pubs/gsw/html/gsw_contents.html.

See Also

swAbsoluteSalinity may be used to compute absolute salinity, and swConservativeTemperature may be used to compute conservative temperature. These values are used by plotTS if the eos argument is set to "teos".

Examples

```
## Not run:
## do not run this in checks, because it will fail unless TEOS-10 is installed
library(oce)
data(ctd)
sp <- ctd[["salinity"]]
```

```
temperature <- ctd[["temperature"]]
n <- length(sp)
p <- ctd[["pressure"]]
lat <- rep(ctd[["latitude"]], n)
lon <- rep(ctd[["longitude"]], n)
sa <- teos("gsw_sa_from_sp", sp, p, lon, lat)
par(mfrow=c(1,2))
plot(ctd, which="salinity")
lines(sa, p, col='red')
plot(ctd, which="density")
rhopot <- teos("gsw_pot_rho_t_exact", sa, temperature, p, rep(0,length(p)))
lines(rhopot - 1000, p, col='red')


## The following demonstrations are from test values from the TEOS-10 supplied
## program "gsw_check_functions.c".  (These, with more digits, are part of the
## test sequence that is run when Oce is built; see the tests/eos.R in the source
## code.)

# Absolute salinity is sa=35.671 g/kg, for practical salinity 35.5 PSU, pressure
# 300 dbar, long 300 degE, and lat 16 degN.
teos("gsw_sa_from_sp", 35.5, 300, 260, 16)

# Conservative temperature is ct=14.930, for abs sal=35.7 g/kg, in-situ temp 15
# degC, and press 300 dbar.
teos("gsw_ct_from_t", 35.7, 15, 300)

# Density is 1026.456 kg/m^3 for sa=35.7 g/kg, ct = 20 degC and p=300
teos("gsw_rho", 35.7, 20, 300)


## End(Not run)
```

`teosSetLibrary` *Set location of TEOS-10 library*

Description

Set location of TEOS-10 library

Usage

```
teosSetLibrary(path)
```

Arguments

`path`	full path to the TEOS-10 shared library.

Details

This need not be called if the library is in the file `/usr/local/lib/libgswteos-10.so`, which is set by default when Oce starts up.

Value

None.

Author(s)

Dan Kelley

See Also

See `teos` for more information on the TEOS-10 library.

`threenum` *Calculate min, mean, and max values*

Description

Calculate min, mean, and max values

Usage

```
threenum(x)
```

Arguments

`x`	a vector or matrix of numerical values.

Details

This is a faster cousin of the standard `fivenum` function.

Value

A vector of three values, the minimum, the mean, and the maximum.

Author(s)

Dan Kelley

Examples

```
library(oce)
threenum(1:10)
```

`tidedata` *Tidal constituent information*

Description

The `tidedata` dataset contains Tide-constituent information that is use by `tidem` in the Oce package to fit tidal models. `tidedata` is a list containing

const a list containing vectors

- `name` a string with constituent name
- `freq` the frequency, in cycles per hour
- `kmpr` a string naming the comparison constituent, blank if there is none
- `ikmpr` index of comparison constituent, or `0` if there is none
- `df` frequency difference betwee constituent and its comparison, used in the Rayleigh criterion
- `d1` first Doodson number
- `d2` second Doodson number
- `d3` third Doodson number
- `d4` fourth Doodson number
- `d5` fifth Doodson number
- `d6` sixth Doodson number
- `semi` (fill in some info later)
- `nsat` number of satellite constituents
- `ishallow` (fill in some info later)
- `nshallow` (fill in some info later)
- `doodsonamp` (fill in some info later)
- `doodsonspecies` (fill in some info later)

`sat` a list containing vectors

- `deldood` (fill in some info later)
- `phcorr` (fill in some info later)
- `amprat` (fill in some info later)
- `ilatfac` (fill in some info later)
- `iconst` (fill in some info later)

`shallow` a list containing vectors

- `iconst` (fill in some info later)
- `coef` (fill in some info later)
- `iname` (fill in some info later)

Apart from the use of `d1` through `d6`, the naming and content follows `T_TIDE`. All of this is based on Foreman (1977), to which the reader is referred for details.

Usage

```
data(tidedata)
```

Author(s)

Dan Kelley

Source

The data come from the `tide3.dat` file of the `T_TIDE` package (Pawlowicz et al., 2002), and derive from Appendices provided by Foreman (1977). The data are scanned using '`tests/tide.R`' in this package, which also performs some tests using `T_TIDE` values as a reference.

References

Foreman, M. G. G., 1977. Manual for tidal heights analysis and prediction. Pacific Marine Science Report 77-10, Institute of Ocean Sciences, Patricia Bay, Sidney, BC, 58pp.

Pawlowicz, Rich, Bob Beardsley, and Steve Lentz, 2002. Classical tidal harmonic analysis including error estimates in MATLAB using `T_TIDE`. Computers and Geosciences, 28, 929-937.

See Also

`tidem` in the Oce package is used to fit tidal models using `tidedata`.

`tidem` *Fit a tidal model to a timeseries*

Description

Fit a tidal model to a timeseries.

Usage

```
tidem(x, t, constituents, latitude=NULL, rc=1, regress=lm,
      debug =getOption("oceDebug"))
```

Arguments

`x`	Either a `sealevel` object (e.g. produced by `read.sealevel` or `as.sealevel`) or a list of numbers. In the former case, time is part of the object, so `t` may not be given here. In the latter case, `tidem` needs a way to determine time, so `t` must be given.
`t`	An indication of times, to be given only if `x` is numeric. There are two styles: `t` may be a POSIXt vector of times at which the `x` observations were made, or `t` may be a single number indicating the number of seconds between those observations. (In the latter case, a sealevel object is created, starting at `as.POSIXct("2000-01-01 00:00:00", tz="UTC")`. The starting time is irrelevant to the fitted coefficients, but it *does* matter if `predict.tidem` is to be used.)
`constituents`	an optional list of tidal constituents to which the fit is done (see "Details".)

latitude	if provided, the latitude of the observations. If not provided, tidem will try to infer this from sl.
rc	the value of the coefficient in the Rayleigh criterion.
regress	function to be used for regression, by default lm, but could be for example rlm from the MASS package.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

The fit is done in terms of sine and cosine components at the indicated tidal frequencies, with the amplitude and phase being calculated from the resultant coefficients on the sine and cosine terms.

The tidal constituents to be used in the analysis are specified as follows.

1. Case 1.If constituents is not provided, then the constituent list will be made up of the 69 constituents regarded by Foreman as standard. These include astronomical frequencies and some shallow-water frequencies, and are as follows: c("Z0", "SA", "SSA", "MSM", "MM", "MSF", "MF", "ALP1",
2. Case 2.If the first item in constituents is the string "standard", then a provisional list is set up as in Case 1, and then the (optional) rest of the elements of constituents are examined, in order. Each of these constituents is based on the name of a tidal constituent in the Foreman (1977) notation. (To get the list, execute data(tideData) and then execute cat(tideData$name).) Each named constituent is added to the existing list, if it is not already there. But, if the constituent is preceeded by a minus sign, then it is removed from the list (if it is already there). Thus, for example, constituents=c("standard", "-M2", "ST32") would remove the M2 constituent and add the ST32 constituent.
3. Case 3.If the first item is not "standard", then the list of constituents is processed as in Case 2, but without starting with the standard list. As an example, constituents=c("K1", "M2") would fit for just the K1 and M2 components. (It would be strange to use a minus sign to remove items from the list, but the function allows that.)

   In each of the above cases, the list is reordered in frequency prior to the analysis, so that the results of summary.tidem will be in a familiar form.

   Once the constituent list is determined, tidem prunes the elements of the list by using the Rayleigh criterion, according to which two constituents of frequencies f_1 and f_2 cannot be resolved unless the time series spans a time interval of at least $rc/(f_1 - f_2)$. The value rc=1 yields nominal resolution.

   A list of constituent names is created by the following:

```
data(tidedata)
print(tidedata$const$name)
```

   The text should include discussion of the (not yet performed) nodal correction treatement.

Value

An object of class "tide", consisting of

const	constituent number, e.g. 1 for Z0, 1 for SA, etc.
model	the regression model
name	a vector of constituent names, in non-subscript format, e.g. "M2".
frequency	a vector of constituent frequencies, in inverse hours.
amplitude	a vector of fitted constituent amplitudes, in metres.
phase	a vector of fitted constituent phase. NOTE: The definition of phase is likely to change as this function evolves. For now, it is phase with respect to the first data sample.
p	a vector containing a sort of p value for each constituent. This is calculated as the average of the p values for the sine() and cosine() portions used in fitting; whether it makes any sense is an open question.

Bugs

1. 1.This function is not fully developed yet, and both the form of the call and the results of the calculation may change.
2. 2.Nodal correction is not done.
3. 3.The reported p value may make no sense at all, and it might be removed in a future version of this function. Perhaps a significance level should be presented, as in the software developed by both Foreman and Pawlowicz.

Author(s)

Dan Kelley

References

Foreman, M. G. G., 1977. Manual for tidal heights analysis and prediction. Pacific Marine Science Report 77-10, Institute of Ocean Sciences, Patricia Bay, Sidney, BC, 58pp.

Foreman, M. G. G., Neufeld, E. T., 1991. Harmonic tidal analyses of long time series. International Hydrographic Review, 68 (1), 95-108.

Leffler, K. E. and D. A. Jay, 2009. Enhancing tidal harmonic analysis: Robust (hybrid) solutions. Continental Shelf Research, 29(1):78-88.

Pawlowicz, Rich, Bob Beardsley, and Steve Lentz, 2002. Classical tidal harmonic analysis including error estimates in MATLAB using T_TIDE. Computers and Geosciences, 28, 929-937.

See Also

summary.tidem summarizes a "tide" object, plot.tidem plots one, and predict.tidem makes predictions from one. As for the input, sealevel objects may be created with as.sealevel or read.sealevel. See notes at sealevelTuktoyaktuk, which is test data set.

Examples

```
library(oce)
# The demonstration time series from Foreman (1977),
# also used in T_TIDE (Pawlowicz, 2002).
data(sealevelTuktoyaktuk)
tide <- tidem(sealevelTuktoyaktuk)
summary(tide)

# AIC analysis
extractAIC(tide[["model"]])

# Fake data at M2
t <- seq(0, 10*86400, 3600)
eta <- sin(0.080511401 * t * 2 * pi / 3600)
sl <- as.sealevel(eta)
m <- tidem(sl)
summary(m)
```

---

tidem-class *Class to store tidal-constituent models*

Description

Class to store tidal-constituent models, with standard slots `metadata`, `data` and `processingLog`.

Author(s)

Dan Kelley

---

tidemAstron *Do ephemeris calculations for tidem*

Description

Do ephemeris calculations for tidem.

Usage

```
tidemAstron(t)
```

Arguments

`t`	time in `POSIXct` format. (It is **very** important to use `tz="GMT"` in constructing `t`.)

Details

Based directly on `t_astron`, from the `T_TIDE` package.

Value

A `data.frame` containing

`astro`	(fill in later)
`ader`	(fill in later)

Author(s)

Dan Kelley, based directly on `t_astron` from the `T_TIDE` package.

See Also

`tidem`, which uses this.

Examples

```
tidemAstron(as.POSIXct("2008-01-22 18:50:24"))
```

`tidemVuf` *Do ephemeris calculations for tidem*

Description

Do ephemeris calculations for tidem.

Usage

```
tidemVuf(t, j, lat=NULL)
```

Arguments

`t`	time in `POSIXct` format. (It is **very** important to use `tz="GMT"` in constructing `t`.)
`j`	indices of tidal constituents to use
`lat`	latitude (if missing, something is done for that)

Details

Based directly on `t_vuf`, from the `T_TIDE` package.

Value

A data.frame containing

astro	(fill in later)
ader	(fill in later)

Author(s)

Dan Kelley, based directly on t_vuf from the T_TIDE package.

See Also

tidem, which uses this.

Examples

```
tidemVuf(as.POSIXct("2008-01-22 18:50:24"), 43, 45.0)
```

---

time.oce *Extract time from an oce object*

Description

Extract time from an oce object

Usage

```
## S3 method for class 'oce'
time(x, ...)
```

Arguments

x	an oce object.
...	possibly contains an argument named which, as explained in "Details".

Details

This is a convenience function that isolates the user from the details of details of storage. The behaviour depends on the type of object.

- adv objects: the times for the velocity observations are returned as a vector of POSIX times.
- adp objects: by default, the times of velocity observations are returned. This is also the case if which=1 is supplied in the ... list. If which=2 is supplied in the ... list, the times at which heading was measured are returned; for some instruments, these are not the same as the velocity times.

Value

A vector of times.

Author(s)

Dan Kelley

See Also

Similar accessor functions are heading, pitch, roll, and velocity.

Examples

```
library(oce)
data(adp)
cat("the first time is:", format(time(adp)[1]), "\n")
```

---

toEnuAdp *Convert from ENU coordinates*

Description

Convert ADP velocities to enu coordinates, from any other coordinates

Usage

```
toEnuAdp(x, declination=0, debug=getOption("oceDebug"))
```

Arguments

x	an object of class "adp".
declination	magnetic declination to be added to the heading, to get ENU with N as "true" north.
debug	a flag that, if non-zero, turns on debugging. Higher values yield more extensive debugging. This is passed to called functions, after subtracting 1.

Details

This is mostly a wrapper that calls beamToXyzAdp if needed, and then calls xyzToEnuAdp.

Author(s)

Dan Kelley

References

http://www.nortek-as.com/lib/forum-attachments/coordinate-transformation

See Also

See read.adp for notes on functions relating to "adp" objects. Also, see beamToXyzAdp and xyzToEnuAdp.

---

toEnuAdv *Convert from ENU coordinates*

Description

Convert ADV velocities to enu coordinates, from any other coordinates

Usage

```
toEnuAdv(x, declination=0, debug=getOption("oceDebug"))
```

Arguments

x	an object of class "adv".
declination	magnetic declination to be added to the heading, to get ENU with N as "true" north.
debug	a flag that, if non-zero, turns on debugging. Higher values yield more extensive debugging. This is passed to called functions, after subtracting 1.

Details

This is mostly a wrapper that calls beamToXyzAdv if needed, and then calls xyzToEnuAdv.

Author(s)

Dan Kelley

References

http://www.nortek-as.com/lib/forum-attachments/coordinate-transformation

See Also

See read.adv for notes on functions relating to "adv" objects. Also, see beamToXyzAdv and xyzToEnuAdv.

topo-class *Class to store topographic data*

Description

Class to store topographic data, with standard slots `metadata`, `data` and `processingLog`.

Methods

Data may be accessed as e.g. `topo[["latitude"]]`, where the string could also be `"longitude"` or `"z"`, or abbreviations matching uniquely to any of these.

Author(s)

Dan Kelley

topoInterpolate *Interpolate within a topography dataset*

Description

Interpolate within a topography dataset

Usage

```
topoInterpolate(longitude, latitude, topo)
```

Arguments

`longitude`	longitude, or vector of longitudes (in the same sign convention as used in `topo`).
`latitude`	latitude, or vector of latitudes (in the same sign convention as used in `topo`).
`topo`	topography object, as e.g. loaded with `data(topoWorld)`.

Details

Bilinear interpolation is used so that values will vary smoothly within a longitude-latitude grid cell. Note that the sign convention for `longitude` and `latitude` must match that in `topo`.

Value

A height (or vector of heights) giving the elevation of the earth above means sea level at the indicated location on the earth.

Author(s)

Dan Kelley

See Also

A topo object may be read with read.topo.

Examples

```
library(oce)
data(topoWorld)
# "The Gully", approx. 400m deep, connects Gulf of St Lawrence with North Atlantic
topoInterpolate(45, -57, topoWorld)
```

---

topoWorld *Global topographic dataset at half-degree resolution*

Description

Global topographic dataset at half-degree resolution, created by decimating the ETOPO5 dataset.

Usage

```
data(topoWorld)
```

Author(s)

Dan Kelley

Source

The ETOPO5 dataset was downloaded in late 2009 from the NOAA website, and decimated from 1/12th degree resolution to 1/2 degree resolution.

References

http://www.ngdc.noaa.gov/mgg/global/relief/ETOPO5/TOPO/ETOPO5/

See Also

Created with as.topo in the Oce package.

Examples

```
## Not run:
library(oce)
data(topoWorld)
plot(topoWorld, location=NULL)

## End(Not run)
```

unabbreviateYear *Determine year from various abbreviations*

Description

Determine year from various abbreviations

Usage

```
unabbreviateYear(year)
```

Arguments

year	a year, or vector of years, possibly abbreviated

Details

Various data files may contain various abbreviations for years. For example, 99 refers to 1999, and 8 refers to 2008. Sometimes, even 108 refers to 2008 (the idea being that the "zero" year was 1900). This function deals with the three cases mentioned. It will fail if someone supplies 60, meaning year 2060 as opposed to 1960.

Author(s)

Dan Kelley

Examples

```
fullYear <- unabbreviateYear(c(99, 8, 108))
```

undriftTime *Correct for drift in instrument clock*

Description

Correct for drift in instrument clock

Usage

```
undriftTime(x, slowEnd=0, tname="time")
```

Arguments

x	an object of class "oce".
slowEnd	number of seconds to add to final instrument time, to get the correct time of the final sample. This will be a positive number, for a "slow" instrument clock.
tname	name of time column.

Details

It is assumed that the instrument clock matches the real time at the start of the sampling, and that the clock drifts linearly (i.e. is uniformly fast or slow) over the sampling interval. Linear interpolation is used to infer the values of all variables in the `data` slot. The data length is altered in this process, e.g. a slow instrument clock (positive `slowEnd`) takes too few samples in a given time interval, so `undriftTime` will increase the number of data.

Value

An object of the same class as `x`, with the `data` slot adjusted appropriately.

Author(s)

Dan Kelley

Examples

```
## Not run:
library(oce)
rbr011855 <- read.oce(
 "/data/archive/sleiwex/2008/moorings/m08/pt/rbr_011855/raw/pt_rbr_011855.dat")
d <- subset(rbr011855, time < as.POSIXct("2008-06-25 10:05:00"))
x <- undriftTime(d, 1)   # clock lost 1 second over whole experiment
summary(d)
summary(x)

## End(Not run)
```

---

ungrid *Extract (x, y, z) from (x, y, grid)*

Description

Extract the grid points from a grid, returning columns.

Usage

```
ungrid(x, y, grid)
```

Arguments

`x`	a vector holding the x coordinates of grid.
`y`	a vector holding the y coordinates of grid.
`grid`	a matrix holding the grid.

Details

This is useful for e.g. gridding large datasets, in which the first step might be to use binMean2D, followed by interpBarnes.

Value

A list containing three vectors: x, the grid x values, y, the grid y values, and grid, the grid values.

Author(s)

Dan Kelley

Examples

```
library(oce)
data(wind)
u <- interpBarnes(wind$x, wind$y, wind$z)
contour(u$xg, u$yg, u$zg)
U <- ungrid(u$xg, u$yg, u$zg)
points(U$x, U$y, col=oceColorsJet(100)[rescale(U$grid, rlow=1, rhigh=100)], pch=20)
```

---

unwrapAngle *Unwrap an angle that suffers modulo-360 problems*

Description

Unwrap an angle that suffers modulo-360 problems

Usage

```
unwrapAngle(angle)
```

Arguments

angle	an angle (in degrees) that is thought be near 360 degrees, with added noise

Details

This is mostly used for instrument heading angles, in cases where the instrument is aligned nearly northward, so that small variations in heading (e.g. due to mooring motion) can yield values that swing from small angles to large angles, because of the modulo-360 cut point.

The method is to use the cosine and sine of the angle, to construct "x" and "y" values on a unit circle, then to find means and medians of x and y respectively, and finally to use atan2 to infer the angles.

Value

A list with two estimates: mean is based on an arithmetic mean, and median is based on the median. Both are mapped to the range 0 to 360.

Author(s)

Dan Kelley

Examples

```
library(oce)
true <- 355
a <- true + rnorm(100, sd=10)
a <- ifelse(a > 360, a - 360, a)
a2 <- unwrapAngle(a)
par(mar=c(3,3,5,3))
hist(a, breaks=360)
abline(v=a2$mean, col="blue", lty="dashed")
abline(v=true, col="blue")
mtext("true (solid)\n estimate (dashed)", at=true, side=3, col="blue")
abline(v=mean(a), col="red")
mtext("mean", at=mean(a), side=3, col="red")
```

---

useHeading *Replace the heading for one instrument with the heading for another*

Description

Replace the heading for one instrument with the heading for another.

Usage

```
useHeading(b, g, add=0)
```

Arguments

b	object holding data from an instrument whose heading is bad, but whose other data are good.
g	object holding data from an instrument whose heading is good, and should be interpolated to the time base of b.
add	an angle, in degrees counterclockwise, to be added to the heading.

Value

A copy of b, but with b$data$heading replaced with an angle that result from linear interpolation of the headings in g, and then adding the angle add.

Author(s)

Dan Kelley

vectorShow *Show some values from a vector*

Description

Show some values from a vector

Usage

```
vectorShow(v, msg, digits=5)
```

Arguments

v	the vector.
msg	a message to show, introducing the vector. If not provided, then a label is created from v.
digits	for numerical values of v, this is the number of digits to use, in formatting the numbers with format; otherwise, digits is ignored.

Details

This is similar to str, but it shows data at the first and last of the vector, which is quite helpful in debugging.

Value

A string, suitable for using in cat or oceDebug.

Author(s)

Dan Kelley

velocityStatistics *Report statistics of ADP or ADV velocities.*

Description

Report statistics of ADP or ADV velocities, such as means and variance ellipses.

Usage

```
velocityStatistics(x, control, ...)
```

Arguments

x	an ADV or ADP object.
control	an optional list used to specify more information. In this version, this is ignored for ADV objects. For ADP objects, if control$bin is an integer, it is taken as the bin to be selected (otherwise, an average across bins is used).
...	additional arguments that are used in the call to mean.

Details

This function has been tested on only a single file, and the data-scanning algorithm was based on visual inspection of that file. Whether it will work generally is an open question.

Value

A list containing items the major and minor axes of the covariance ellipse (ellipseMajor and ellipseMinor), the angle of the major axis anticlockwise of the horizontal axis (ellipseAngle), and the x and y components of the mean velocity (uMean and vMean).

Author(s)

Dan Kelley

Examples

```
library(oce)
data(adp)
a <- velocityStatistics(adp)
print(a)
t <- seq(0, 2*pi, length.out=100)
theta <- a$ellipseAngle * pi / 180
y <- a$ellipseMajor * cos(t) * sin(theta) + a$ellipseMinor * sin(t) * cos(theta)
x <- a$ellipseMajor * cos(t) * cos(theta) - a$ellipseMinor * sin(t) * sin(theta)
plot(adp, which="uv+ellipse+arrow")
lines(x, y, col='blue', lty="dashed", lwd=5)
arrows(0, 0, a$uMean, a$vMean, lwd=5, length=1/10, col='blue', lty="dashed")
```

---

webtide *Get a tidal prediction from a WebTide database*

Description

Get a tidal prediction from a WebTide database

Usage

```
webtide(action=c("map", "predict"),
        longitude, latitude, node, time,
        basedir=getOption("webtide"),
        region="nwatl", plot=TRUE,
        tformat, ...)
```

Arguments

action	An indication of the action, either action="map" to draw a map or actoin="predict" to get a prediction; see 'Details'.
longitude	longitude at which prediction is required (ignored if node is given).
latitude	latitude at which prediction is required (ignored if node is given).
node	node to look up; only needed if longitude and latitude are not given.
time	times at which prediction is to be made. If not supplied, this will be the week starting at the present time, incrementing by 15 minutes.
basedir	directory containing the WebTide application.
region	database region, given as a directory name in the WebTide directory. For example, h3o is for Halifax Harbour, nwatl is for the northwest Atlantic, and sshelf is for the Scotian Shelf and Gulf of Maine.
plot	boolean indicating whether to plot.
tformat	optional argument passed to oce.plot.ts, for plot types that call that function. (See strptime for the format used.)
...	optional arguments passed to plotting functions. A common example is to set xlim and ylim, to focus a map region.

Details

If action="map" then a map is drawn, with a dot for the lower-left corner of each triangle used in the finite-element tidal simulation upon which WebTide predictions are based. If node is missing, then locator is called, so that the user can indicate a spot of interest on the map, and this point is indicated on the map (and in the return value). If node is provided, however, the point is indicated but locator is not called. (This second style is of use in documenting interactive work after the fact.)

If action="predict" then either a node number or the longitude and latitude must be specified. If plot=TRUE (the default) then a plot is drawn, but no plot is produced otherwise. In either case, the (silent) return value is a list as described in the next section. The times used for prediction are specified with the time argument, and if this is not specified then a week following the present time is used.

Naturally, webtide will not work unless WebTide has been installed on the computer.

Value

If action="map" and plot=TRUE, the return value is a list containing the index of the nearest node, along with the latitude and longitude of that node. If action="map" and plot=FALSE, the return value is a list of all nodes, longitude, and latitudes.

If action="predict", the return value is a list containing a vector of times (time), as well as vectors of the predicted elevation in metres and the predicted horizontal components of velocity, u and v, along with the node number, and the basedir and region as supplied to this function.

Author(s)

Dan Kelley

References

The WebTide software may be downloaded for free at http://www.bio.gc.ca/science/research-recherche/ocean/webtide/index-eng.php, along with a suite of databases for various geographical regions. Note that WebTide is not an open-source application, as it consists mainly of compiled Java code, which precludes examination of the source.

Examples

```
## Not run:
library(oce)
prediction <- webtide("predict", lon=-69.61, lat=48.14)

## End(Not run)
```

---

wind *Wind dataset*

Description

Wind data.

Usage

```
data(wind)
```

Format

Columns wind$x and wind$y are location, while wind$z is the wind speed, in m/s.

Author(s)

Dan Kelley

Source

The data are taken from Figure 5 of Koch et al. (1983).

References

S. E. Koch and M. DesJardins and P. J. Kocin, 1983. "An interactive Barnes objective map anlaysis scheme for use with satellite and conventional data," *J. Climate Appl. Met.*, vol 22, p. 1487-1503.

See Also

See interpBarnes in the Oce package.

---

window.oce *Window an oce object by time or distance*

Description

Window an oce object by time or distance

Usage

```
## S3 method for class 'oce'
window(x, start = NULL, end = NULL,
     frequency = NULL, deltat = NULL, extend = FALSE,
     which=c("time","distance"), indexReturn=FALSE,
     debug=getOption("oceDebug"), ...)
```

Arguments

x	an oce object.
start	the start time (or distance) of the time (or space) region of interest. This may be a single value or a vector.
end	the end time (or distance) of the time (or space) region of interest. This may be a single value or a vector.
frequency	not permitted yet.
deltat	not permitted yet
extend	not permitted yet
which	string containing the name of the quantity on which sampling is done. Possibilities are "time", which applies the windowing on the time entry of the data slot, and "distance", for the distance entry (for those objects, such as adp, that have this entry).
indexReturn	boolean flag indicating whether to return a list of the "kept" indices for the time entry of the data slot, as well as the timeSlow entry, if there is one.. Either of these lists will be NULL, if the object lacks the relevant items.
debug	a flag that turns on debugging.
...	ignored

Details

Windows x on either time or distance, depending on the value of which. In each case, values of start and end may be integers, to indicate a portion of the time or distance range. If which is "time", then the start and end values may also be provided as POSIX times, or character strings indicating times (in time zone given by the value of getOption("oceTz")).

Value

Normally, this is new oce object. However, if indexReturn=TRUE, the return value is two-element list containing items named index and indexSlow, which are the indices for the time entry of the data slot (and the timeSlow, if it exists).

Author(s)

Dan Kelley

See Also

subset provides more flexible selection of subsets.

Examples

```
library(oce)
data(adp)
plot(adp)
early <- window(adp,
                start="2008-06-26 00:00:00",
                end="2008-06-26 12:00:00")
plot(early)
bottom <- window(adp, start=0, end=20, which="distance")
plot(bottom)
```

---

windrose-class *Class to store windrose data*

Description

Class to store windrose data, with standard slots metadata, data and processingLog.

Methods

Data may be accessed as e.g. windrose[["theta"]], where the string could also be "count", or "fives", or abbreviations matching uniquely to any of these.

Author(s)

Dan Kelley

write.ctd *Write a CTD data object as a .csv file*

Description

Write a CTD data object as a .csv file.

Usage

```
write.ctd(object, file=stop("'file' must be specified"))
```

Arguments

object	A ctd object, e.g. as read by read.ctd.
file	Either a character string (the file name) or a connection. This is a mandatory argument.

Details

Writes a comma-separated file containing the data frame stored in object@data. The file is suitable for reading with a spreadsheet, or with read.csv. Note that the output file will retain none of the meta-data stored in object.

Author(s)

Dan Kelley

See Also

The documentation for ctd-class explains the structure of CTD objects, and also outlines the other functions dealing with them.

Examples

```
library(oce)
data(ctd)
write.ctd(ctd, "ctd.csv")
d <- read.csv("ctd.csv")
plot(as.ctd(d$salinity, d$temperature, d$pressure))
```

xyzToEnuAdp *Convert XYZ to ENU coordinates*

Description

Convert ADP or ADV velocity components from a xyz-based coordinate system to an enu-based coordinate system, by using the instrument's recording of heading, pitch, and roll.

Usage

```
xyzToEnuAdp(x, declination=0, debug=getOption("oceDebug"))
```

Arguments

x	an object of class "adp".
declination	magnetic declination to be added to the heading after "righting" (see below), to get ENU with N as "true" north.
debug	a flag that turns on debugging. Set to 1 to get a moderate amount of debugging information, or to 2 to get more.

Details

The first step is to convert the (x,y,z) velocity components (stored in the three columns of `x[["v"]][,,1:3]`) into what RDI [1, pages 11 and 12] calls "ship" (or "righted") components. For example, the z coordinate, which may point upwards or downwards depending on instrument orientation, is mapped onto a "mast" coordinate that points more nearly upwards than downward. The other ship coordinates are called "starboard" and "forward", the meanings of which will be clear to mariners. Once the (x,y,z) velocities are converted to ship velocities, the orientation of the instrument is extracted from heading, pitch, and roll vectors stored in the object. These angles are defined differently for RDI and Sontek profilers.

The code handles every case individually, based on the table given below. The table comes from Clark Richards, a former PhD student at Dalhousie University [2], who developed it based on instrument documentation, discussion on user groups, and analysis of measurements acquired with RDI and Sontek acoustic current profilers in the SLEIWEX experiment [3]. In the table, (X, Y, Z) denote instrument-coordinate velocities, (S, F, M) denote ship-coordinate velocities, and (H, P, R) denote heading, pitch, and roll.

Case	Mfr.	Instr.	Orient.	H	P	R	S	F	M	
1	RDI	ADCP	up	H	arctan(tan(P)*cos(R))	R	-X	Y	-Z	
2	RDI	ADCP	down	H	arctan(tan(P)*cos(R))	-R	X	Y	Z	
3	Nortek	ADP	up	H-90	R	-P	X	Y	Z	
4	Nortek	ADP	down	H-90	R	-P	X	-Y	-Z	
5	Sontek	ADP	up	H-90	-P	-R	X	Y	Z	
6	Sontek	ADP	down	H-90	-P	-R	X	Y	Z	
7	Sontek	PCADP	up	H-90	R	-P	X	Y	Z	
8	Sontek	PCADP	down	H-90	R	-P	X	Y	Z	

Finally, a standardized rotation matrix is used to convert from ship coordinates to earth coordinates. As described in the RDI coordinate transformation manual [1, pages 13 and 14], this matrix is based on sines and cosines of heading, pitch, and roll If `CH` and `SH` denote cosine and sine of heading (after adjusting for declination), with similar terms for pitch and roll using second letters `P` and `R`, the rotation matrix is

```
rbind(c( CH*CR + SH*SP*SR,   SH*CP,    CH*SR - SH*SP*CR),
      c(-SH*CR + CH*SP*SR,   CH*CP,   -SH*SR - CH*SP*CR),
      c(           -CP*SR,      SP,               CP*CR))
```

This matrix is left-multiplied by a matrix with three rows, the top a vector of "starboard" values, the middle a vector of "forward" values, and the bottom a vector of "mast" values. Finally, the columns of `data$v[,,1:3]` are filled in with the result of the matrix multiplication.

Value

An object with `data$v[,,1:3]` altered appropriately, and `metadata$oce.orientation` changed from `xyz` to `enu`.

Author(s)

Dan Kelley

References

1. RD Instruments, 1998. *ADCP Coordinate Transformation, formulas and calculations.* P/N 951-6079-00 (July 1998).
2. Clark Richards, 2012, PhD Dalhousie University Department of Oceanography.
3. The SLEIWEX experiment (http://myweb.dal.ca/kelley/SLEIWEX/index.php).

See Also

See `read.adp` for other functions that relate to objects of class `"adp"`.

`xyzToEnuAdv` *Convert XYZ to ENU coordinates*

Description

Convert ADP or ADV velocity components from a xyz-based coordinate system to an enu-based coordinate system.

Usage

```
xyzToEnuAdv(x, declination=0,
            cabled=FALSE, horizontalCase, sensorOrientation,
            debug=getOption("oceDebug"))
```

Arguments

x	an object of class "adv".
declination	magnetic declination to be added to the heading, to get ENU with N as "true" north.
cabled	boolean value indicating whether the sensor head is connected to the pressure case with a cable. If cabled=FALSE, then horizontalCase is ignored. See "Details".
horizontalCase	optional boolean value indicating whether the sensor case is oriented horizontally. Ignored unless cabled is TRUE. See "Details".
sensorOrientation	optional string indicating the direction in which the sensor points. The value, which must be "upward" or "downward", over-rides the value of x@metadata$orientation, which will have been set by read.adv, *provided* that the data file contained the full header. See "Details".
debug	a flag that, if non-zero, turns on debugging. Higher values yield more extensive debugging.

Details

The coordinate transformation is done using the heading, pitch, and roll information contained within x. The algorithm is similar to that used for Teledyne/RDI ADCP units, taking into account the different definitions of heading, pitch, and roll as they are defined for the velocimeters.

Generally, the transformation must be done on a time-by-time basis, which is a slow operation. However, this function checks whether the vectors for heading, pitch and roll, are all of unit length, and in that case, the calculation is altered, resulting in shorter executation times. Note that the angles are held in (data$timeSlow, data$headingSlow, ...) for Nortek instruments and (data$time, data$heading, ...) for Sontek instruments.

Since the documentation provided by instrument manufacturers can be vague on the coordinate transformations, the method used here had to be developed indirectly. (This is in contrast to the RDI ADCP instruments, for which there are clear instructions.) documents that manufacturers provide. If results seem incorrect (e.g. if currents go east instead of west), users should examine the code in detail for the case at hand. The first step is to set debug to 1, so that the processing will print a trail of processing steps. The next step should be to consult the table below, to see if it matches the understanding (or empirical tests) of the user. It should not be difficult to tailor the code, if needed.

The code handles every case individually, based on the table given below. The table comes from Clark Richards, a former PhD student at Dalhousie University [2], who developed it based on instrument documentation, discussion on user groups, and analysis of measurements acquired with Nortek and Sontek velocimeters in the SLEIWEX experiment [3].

The column labelled "Cabled" indicates whether the sensor and the pressure case are connected with a flexible cable, and the column labelled "H.case" indicates whether the pressure case is oriented horizontally. These two properties are not discoverable in the headers of the data files, and so they must be supplied with the arguments cabled and horizontalCase. The source code refers to the information in this table by case numbers. (Cases 5 and 6 are not handled.) Angles are abbreviated as follows:: heading "H," pitch "P," and roll "R". Entries X, Y and Z refer to instrument coordinates of the same names. Entries S, F and M refer to so-called ship coordinates starboard, forward, and

mast; it is these that are used together with a rotation matrix to get velocity components in the east, north, and upward directions.

Case	Mfr.	Instr.	Cabled	H. case	Orient.	H	P	R	S	F	M
1	Nortek	vector	no	-	up	H-90	R	-P	X	-Y	-Z
2	Nortek	vector	no	-	down	H-90	R	-P	X	Y	Z
3	Nortek	vector	yes	yes	up	H-90	R	-P	X	Y	Z
4	Nortek	vector	yes	yes	down	H-90	R	P	X	-Y	-Z
5	Nortek	vector	yes	no	up	-	-	-	-	-	-
6	Nortek	vector	yes	no	down	-	-	-	-	-	-
7	Sontek	adv	-	-	up	H-90	R	-P	X	-Y	-Z
8	Sontek	adv	-	-	down	H-90	R	-P	X	Y	Z

Author(s)

Dan Kelley, in collaboration with Clark Richards

References

1. http://www.nortek-as.com/lib/forum-attachments/coordinate-transformation
2. Clark Richards, 2012, PhD Dalhousie University Department of Oceanography.
3. The SLEIWEX experiment (http://myweb.dal.ca/kelley/SLEIWEX/index.php).

See Also

See `read.adv` for notes on functions relating to `"adv"` objects.

Index

∗Topic **classes**
adp-class, 12
adv-class, 15
cm-class, 48
coastline-class, 48
ctd-class, 56
drifter-class, 77
echosounder-class, 79
gps-class, 96
landsat-class, 110
lisst-class, 114
lobo-class, 115
met-class, 140
oce-class, 146
sealevel-class, 266
section-class, 270
tdr-class, 336
tidem-class, 347
topo-class, 352
windrose-class, 363
∗Topic **color**
oceColors, 155
∗Topic **datasets**
adp, 11
adv, 14
cm, 47
coastlineWorld, 50
ctd, 55
ctdRaw, 61
drifter, 76
echosounder, 78
lisst, 113
lobo, 115
met, 140
nao, 143
sealevel, 265
sealevelTuktoyaktuk, 267
section, 269
soi, 276
tdr, 335
tidedata, 343
topoWorld, 353
wind, 361
∗Topic **hplot**
errorbars, 84
mapContour, 123
mapImage, 124
mapLines, 126
mapLocator, 127
mapLongitudeLatitudeXY, 128
mapMeridians, 129
mapPlot, 130
mapPoints, 132
mapPolygon, 133
mapScalebar, 135
mapText, 136
mapZones, 137
oce.plot.ts, 150
plot.adp, 167
plot.adv, 172
plot.cm, 175
plot.coastline, 177
plot.ctd, 180
plot.drifter, 183
plot.echosounder, 185
plot.gps, 187
plot.landsat, 190
plot.lisst, 191
plot.lobo, 192
plot.met, 194
plot.sealevel, 195
plot.section, 197
plot.tdr, 201
plot.tidem, 203
plot.topo, 204
plot.windrose, 206
plotInset, 207
plotPolar, 209

plotProfile, 209
plotScan, 213
plotSticks, 214
plotTaylor, 215
plotTS, 216
∗Topic **manip**
head, 98
∗Topic **misc**
abbreviateTimeLabels, 8
accessors, 9
addColumn, 10
airRho, 16
angleRemap, 17
applyMagneticDeclination, 18
approx3d, 19
as.coastline, 20
as.ctd, 21
as.drifter, 24
as.echosounder, 25
as.gps, 26
as.lisst, 27
as.lobo, 28
as.met, 29
as.sealevel, 30
as.section, 31
as.tdr, 32
as.topo, 34
as.windrose, 34
bcdToInteger, 36
beamName, 36
beamToXyz, 37
beamToXyzAdp, 38
beamToXyzAdv, 39
beamUnspreadAdp, 40
binApply, 41
binAverage, 42
binmapAdp, 44
binMean, 45
byteToBinary, 46
coastlineBest, 49
colormap, 51
coriolis, 54
ctdAddColumn, 57
ctdDecimate, 58
ctdFindProfiles, 60
ctdTrim, 62
ctdUpdateHeader, 64
ctimeToSeconds, 65
decimate, 66
decodeHeader, 67
decodeTime, 68
despike, 69
detrend, 71
drawDirectionField, 72
drawIsopycnals, 73
drawPalette, 74
eclipticalToEquatorial, 80
enuToOtherAdp, 81
enuToOtherAdv, 82
equatorialToLocalHorizontal, 83
extract, 85
fillGap, 86
findBottom, 87
findInOrdered, 88
formatCI, 88
formatPosition, 90
fullFilename, 91
geodDist, 91
geodGc, 93
geodXy, 94
GMTOffsetFromTz, 95
grad, 96
gravity, 97
header, 99
imagep, 100
integerToAscii, 103
integrateTrapezoid, 104
interpBarnes, 105
is.beam, 107
julianCenturyAnomaly, 108
julianDay, 109
landsatTrim, 111
latFormat, 112
latlonFormat, 112
loggerToc, 116
lonFormat, 117
magneticField, 118
makeFilter, 119
makeSection, 121
matchBytes, 138
matrixSmooth, 139
moonAngle, 141
numberAsHMS, 144
numberAsPOSIXct, 145
oce.as.POSIXlt, 147
oce.as.raw, 148

oce.axis.POSIXct, 148
oce.write.table, 152
oceApprox, 153
oceContour, 156
oceConvolve, 158
oceDebug, 159
oceEdit, 160
oceFilter, 161
oceMagic, 162
ocePmatch, 163
oceSmooth, 164
oceSpectrum, 165
parseLatLon, 166
predict.tidem, 219
prettyPosition, 220
processingLog, 221
pwelch, 222
rangeExtended, 224
rangeLimit, 224
read.adp, 225
read.adv, 229
read.cm, 234
read.coastline, 236
read.ctd, 238
read.drifter, 242
read.echosounder, 244
read.gps, 245
read.landsat, 246
read.lisst, 247
read.lobo, 249
read.met, 251
read.observatory, 252
read.oce, 253
read.sealevel, 254
read.section, 256
read.tdr, 258
read.topo, 259
rescale, 260
resizableLabel, 261
retime, 262
runlm, 263
secondsToCtime, 268
sectionGrid, 271
sectionSmooth, 272
sectionSort, 273
showMetadataItem, 274
siderealTime, 275
standardDepths, 277
subset.adp, 278
subset.adv, 279
subset.cm, 280
subset.coastline, 281
subset.ctd, 282
subset.echosounder, 283
subset.lisst, 284
subset.oce, 285
subset.sealevel, 285
subset.section, 286
subset.tdr, 287
subset.topo, 288
subtractBottomVelocity, 289
summary.adp, 289
summary.adv, 290
summary.cm, 291
summary.coastline, 292
summary.ctd, 293
summary.drifter, 294
summary.echosounder, 295
summary.gps, 296
summary.landsat, 296
summary.lisst, 297
summary.lobo, 298
summary.met, 299
summary.sealevel, 300
summary.section, 301
summary.tdr, 302
summary.tidem, 303
summary.topo, 304
summary.windrose, 305
sunAngle, 306
swAbsoluteSalinity, 307
swAlpha, 309
swAlphaOverBeta, 310
swBeta, 311
swConductivity, 312
swConservativeTemperature, 313
swDepth, 314
swDynamicHeight, 315
swLapseRate, 316
swN2, 317
swPressure, 319
swRho, 320
swRrho, 321
swSCTp, 322
swSigma, 323
swSigmaT, 324

swSigmaTheta, 325
swSoundAbsorption, 326
swSoundSpeed, 327
swSpecificHeat, 328
swSpice, 329
swSTrho, 330
swTFreeze, 331
swTheta, 332
swTSrho, 333
swViscosity, 334
tdrPatm, 337
tdrTrim, 338
teos, 339
teosSetLibrary, 341
threenum, 342
tidem, 344
tidemAstron, 347
tidemVuf, 348
time.oce, 349
toEnuAdp, 350
toEnuAdv, 351
topoInterpolate, 352
unabbreviateYear, 354
undriftTime, 354
ungrid, 355
unwrapAngle, 356
useHeading, 357
vectorShow, 358
velocityStatistics, 358
webtide, 359
window.oce, 362
write.ctd, 364
xyzToEnuAdp, 365
xyzToEnuAdv, 366
∗Topic **oce**
adp-class, 12
adv-class, 15
cm-class, 48
coastline-class, 48
ctd-class, 56
drifter-class, 77
echosounder-class, 79
gps-class, 96
landsat-class, 110
lisst-class, 114
lobo-class, 115
met-class, 140
oce-class, 146
sealevel-class, 266
section-class, 270
tdr-class, 336
tidem-class, 347
topo-class, 352
windrose-class, 363
+.section *(*makeSection*)*, 121
[[,adp-method *(*adp-class*)*, 12
[[,adv-method *(*adv-class*)*, 15
[[,cm-method *(*cm-class*)*, 48
[[,coastline,string,missing,ANY-method *(*accessors*)*, 9
[[,coastline-method *(*coastline-class*)*, 48
[[,ctd-method *(*ctd-class*)*, 56
[[,drifter-method *(*drifter-class*)*, 77
[[,echosounder-method *(*echosounder-class*)*, 79
[[,gps-method *(*gps-class*)*, 96
[[,landsat-method *(*landsat-class*)*, 110
[[,lisst-method *(*lisst-class*)*, 114
[[,lobo-method *(*lobo-class*)*, 115
[[,met-method *(*met-class*)*, 140
[[,oce-method *(*oce-class*)*, 146
[[,sealevel-method *(*sealevel-class*)*, 266
[[,section-method *(*section-class*)*, 270
[[,tdr-method *(*tdr-class*)*, 336
[[,tidem-method *(*tidem-class*)*, 347
[[,topo-method *(*topo-class*)*, 352
[[,windrose-method *(*windrose-class*)*, 363
[[<-,adp-method *(*adp-class*)*, 12
[[<-,adv-method *(*adv-class*)*, 15
[[<-,cm-method *(*cm-class*)*, 48
[[<-,coastline-method *(*coastline-class*)*, 48
[[<-,ctd-method *(*ctd-class*)*, 56
[[<-,echosounder-method *(*echosounder-class*)*, 79
[[<-,gps-method *(*gps-class*)*, 96
[[<-,landsat-method *(*landsat-class*)*, 110
[[<-,lisst-method *(*lisst-class*)*, 114
[[<-,lobo-method *(*lobo-class*)*, 115
[[<-,met-method *(*met-class*)*, 140
[[<-,oce-method *(*oce-class*)*, 146
[[<-,sealevel-method *(*sealevel-class*)*, 266
[[<-,section-method *(*section-class*)*, 270
[[<-,tdr-method *(*tdr-class*)*, 336

abbreviateTimeLabels, 8
accessors, 9
addColumn, 10
adp, 11
adp-class, 12
adv, 14
adv-class, 15
airRho, 16
angleRemap, 17
applyMagneticDeclination, 18
approx, *129*, *137*, *186*, *263*
approx3d, 19
approxfun, *315*
arrows, *84*, *214*
as.coastline, 20, *49*, *177*
as.ctd, 21, *56*, *60*, *293*
as.drifter, 24, *78*
as.echosounder, 25, *80*, *186*, *295*
as.gps, 26, *96*, *188*
as.lisst, 27, *114*, *249*, *297*
as.lobo, 28, *116*
as.met, 29, *141*, *299*
as.POSIXct, *68, 69*, *109*, *230, 231*, *235*, *275*
as.POSIXlt, *109*, *147*, *275*
as.raw, *148*
as.sealevel, 30, *255*, *300*, *344*, *346*
as.section, 31
as.tdr, 32, *336*
as.topo, 34, *100*
as.windrose, 34, *206, 207*, *305*
atan2, *17*, *356*
axis, *200*
axis.POSIXct, *149*

bandwidth.kernel, *120*
bcdToInteger, 36
beamName, 36
beamToXyz, 37, *228*
beamToXyzAdp, *14*, *37, 38*, 38, *169*, *350, 351*
beamToXyzAdv, *16*, *37, 38*, 39, *351*
beamUnspreadAdp, 40
binApply, 41
binApply1D *(*binApply*)*, 41
binApply2D *(*binApply*)*, 41
binAverage, 42
binCount *(*binMean*)*, 45
binCount1D *(*binMean*)*, 45
binCount2D *(*binMean*)*, 45
binmapAdp, 44
binMean, 45
binMean1D *(*binMean*)*, 45
binMean2D, *41*, *356*
binMean2D *(*binMean*)*, 45
byteToBinary, 46

cat, *159*, *358*
class, *11*, *21*, *23, 24*, *26–29*, *31–35*, *40*, *44*, *57*, *59*, *63*, *66*, *111*, *121*, *160, 161*, *164*, *227*, *232*, *235*, *237*, *240*, *243, 244*, *246*, *248, 249*, *251, 252*, *254, 255*, *257*, *259, 260*, *272–274*, *338*, *345*, *354*
cm, 47
cm-class, 48
coastline-class, 48
coastlineBest, 49
coastlineWorld, 50, *130*
colormap, 51, *74*, *101*, *125*
colorRampPalette, *52*
confint, *89*
contour, *123*, *156*, *200*
contourLines, *123*
convolve, *120*, *158*
coordinate *(*is.beam*)*, 107
coplot, *212*, *218*
coriolis, 54
ctd, 55
ctd-class, 56
ctdAddColumn, *11*, *57*, 57
ctdDecimate, *57*, 58, *63*, *66, 67*, *271, 272*
ctdFindProfiles, *57*, 60
ctdRaw, 61
ctdTrim, *57*, *60*, 62, *181*, *210*
ctdUpdateHeader, *57*, 64
ctimeToSeconds, 65, *268*
cumsum, *104*

data.frame, *255*, *348, 349*
decimate, 66
decodeHeader, 67
decodeHeaderNortek *(*decodeHeader*)*, 67
decodeTime, 68
despike, 69, *151*, *169*, *227*
detrend, 71, *223*
diff, *60*, *318*
dir, *274*
distance *(*accessors*)*, 9
drawDirectionField, 72

drawIsopycnals, 73
drawPalette, *52*, 74, *101*, *102*, *151*
drifter, 76
drifter-class, 77

echosounder, 78
echosounder-class, 79
eclipticalToEquatorial, 80, *83*, *141*
elevation *(*accessors*)*, 9
enuToOther *(*beamToXyz*)*, 37
enuToOtherAdp, *14*, *38*, 81, *169*
enuToOtherAdv, *16*, *38*, 82
equatorialToLocalHorizontal, 83, *141*
errorbars, 84
expression, *102*, *150*, *173*, *175*
extract, 85

filled.contour, *102*
fillGap, 86
filter, *66*, *120*, *161*
findBottom, *80*, 87, *186*
findInOrdered, 88
fivenum, *35*, *206*, *342*
format, *8*, *186*, *358*
formatCI, 88
formatPosition, 90, *221*
fullFilename, 91

geodDist, *79*, 91, *93*, *94*
geodGc, 93, *131*
geodXy, *93*, 94
getOption, *89*
GMTOffsetFromTz, 95
gps-class, 96
grad, 96
gravity, 97
grid, *151*, *169*

head, 98
header, 99
heading, *350*
heading *(*accessors*)*, 9
heading<- *(*accessors*)*, 9
hsv, *52*

image, *51–53*, *100–102*, *125*, *171*
imagep, *52*, *53*, *76*, 100, *151*, *168*, *169*, *173*, *186*, *196*
integerToAscii, 103
integrate, *315*
integrateTrapezoid, 104
interpBarnes, 105, *272*, *273*, *356*
is.beam, 107
is.enu *(*is.beam*)*, 107
is.xyz *(*is.beam*)*, 107

julian, *109*
julianCenturyAnomaly, 108
julianDay, *108*, 109

kernapply, *120*
kernel, *120*

landsat-class, 110
landsatTrim, *110*, 111
latFormat, 112, *113*, *117*
latitude *(*accessors*)*, 9
latitude<- *(*accessors*)*, 9
latlonFormat, *112*, 112, *117*
layout, *75*, *102*, *169*
legend, *135*, *199*
lines, *126*, *127*, *129*, *137*
lisst, 113
lisst-class, 114
list, *237*, *249*, *255*, *257*, *270*
list.files, *239*
lm, *89*, *264*, *345*
lobo, 115
lobo-class, 115
locator, *127*, *360*
log10, *186*
loggerToc, 116
lonFormat, *112*, *113*, 117
longitude *(*accessors*)*, 9
longitude<- *(*accessors*)*, 9

magneticField, *19*, 118
makeFilter, *67*, 119
makeSection, *23*, 121, *198*, *270*, *274*, *301*
map2lonlat, 122, *127*
mapContour, 123
mapImage, 124, *134*
mapLines, *123*, 126, *127*, *131*
mapLocator, *126*, 127, *128*, *131*, *133*, *134*
mapLongitudeLatitudeXY, 128
mapMeridians, 129
mapPlot, *123–125*, *127–129*, 130, *133–137*, *178*, *188*

mapPoints, *127*, *131*, 132, *178*, *188*
mapPolygon, 133
mapproject, *122*, *128*, *131*, *182*
mapScalebar, *131*, 135
mapText, *131*, 136
mapZones, 137
matchBytes, 138
matrixSmooth, 139
mean, *41*, *359*
met, 140, *141*
met-class, *35*, 140
moonAngle, 141, *307*

nao, 143
nitrate *(*accessors*)*, 9
nitrate<- *(*accessors*)*, 9
nitrite *(*accessors*)*, 9
nitrite<- *(*accessors*)*, 9
nls, *89*
numberAsHMS, *144*, 144, *145*
numberAsPOSIXct, *109*, 145, *275*

oce *(*oce-class*)*, 146
oce-class, 146
oce.as.POSIXlt, 147
oce.as.raw, 148
oce.axis.POSIXct, 148, *151*, *157*
oce.plot.ts, *101*, *149*, 150, *168*, *173*, *176*, *184*, *191*, *194*, *202*, *360*
oce.write.table, 152
oceApprox, *58*, 153
oceColors, 155
oceColors9A *(*oceColors*)*, 155
oceColors9B *(*oceColors*)*, 155
oceColorsGebco, *205*
oceColorsGebco *(*oceColors*)*, 155
oceColorsJet, *74*, *101*, *125*
oceColorsJet *(*oceColors*)*, 155
oceColorsPalette *(*oceColors*)*, 155
oceColorsTwo *(*oceColors*)*, 155
oceContour, 156
oceConvolve, 158
oceDebug, 159, *358*
oceEdit, *146*, 160
oceFilter, 161
oceMagic, 162, *231*, *253*, *254*
ocePmatch, 163
oceSmooth, 164
oceSpectrum, 165

optim, *122*
oxygen *(*accessors*)*, 9
oxygen<- *(*accessors*)*, 9

par, *74*, *102*, *131*, *149–151*, *168*, *169*, *173*, *176*, *178*, *182*, *184*, *187*, *189*, *193*, *194*, *196*, *199*, *202*, *203*, *205*, *207*, *212*, *214*, *218*
parseLatLon, 166
phosphate *(*accessors*)*, 9
phosphate<- *(*accessors*)*, 9
pitch, *350*
pitch *(*accessors*)*, 9
pitch<- *(*accessors*)*, 9
plot, *151*, *172*, *175*, *176*, *182*, *202*, *212*
plot,adp,missing-method *(*plot.adp*)*, 167
plot,adp-method *(*plot.adp*)*, 167
plot,adv,missing-method *(*plot.adv*)*, 172
plot,adv-method *(*plot.adv*)*, 172
plot,cm,missing-method *(*plot.cm*)*, 175
plot,cm-method *(*plot.cm*)*, 175
plot,coastline,missing-method *(*plot.coastline*)*, 177
plot,coastline-method *(*plot.coastline*)*, 177
plot,ctd,missing-method *(*plot.ctd*)*, 180
plot,ctd-method *(*plot.ctd*)*, 180
plot,drifter,missing-method *(*plot.drifter*)*, 183
plot,drifter-method *(*plot.drifter*)*, 183
plot,echosounder,missing-method *(*plot.echosounder*)*, 185
plot,echosounder-method *(*plot.echosounder*)*, 185
plot,gps,missing-method *(*plot.gps*)*, 187
plot,gps-method *(*plot.gps*)*, 187
plot,landsat,missing-method *(*plot.landsat*)*, 190
plot,landsat-method *(*plot.landsat*)*, 190
plot,lisst,missing-method *(*plot.lisst*)*, 191
plot,lisst-method *(*plot.lisst*)*, 191
plot,lobo,missing-method *(*plot.lobo*)*, 192
plot,lobo-method *(*plot.lobo*)*, 192
plot,met,missing-method *(*plot.met*)*, 194
plot,met-method *(*plot.met*)*, 194
plot,sealevel,missing-method *(*plot.sealevel*)*, 195

plot,sealevel-method (plot.sealevel), 195
plot,section,missing-method (plot.section), 197
plot,section-method (plot.section), 197
plot,tdr,missing-method (plot.tdr), 201
plot,tdr-method (plot.tdr), 201
plot,tidem,missing-method (plot.tidem), 203
plot,tidem-method (plot.tidem), 203
plot,topo,missing-method (plot.topo), 204
plot,topo-method (plot.topo), 204
plot,windrose,missing-method (plot.windrose), 206
plot,windrose-method (plot.windrose), 206
plot.adp, *8*, *14*, *150*, 167, *173*, *175*
plot.adv, *16*, 172
plot.cm, *48*, 175
plot.coastline, *49*, 177, *181*, *281*
plot.ctd, *23*, *57*, *169*, 180, *213*, *262*
plot.drifter, *24*, *78*, 183
plot.echosounder, *79, 80*, 185
plot.gps, *96*, 187
plot.landsat, *110*, 190
plot.lisst, *114*, 191
plot.lobo, *116*, 192
plot.met, *141*, 194
plot.sealevel, 195
plot.section, 197, *270*
plot.tdr, 201, *336*
plot.tidem, 203, *303*, *346*
plot.topo, 204
plot.ts, *151*
plot.windrose, *35*, 206, *305*
plotInset, *178*, *182*, *189*, 207, *212*, *218*
plotPolar, 209
plotProfile, *57*, 209
plotScan, *57*, 213
plotSticks, 214
plotTaylor, 215
plotTS, *57*, *73, 74*, *212*, 216, *330*, *340*
pmatch, *163, 164*
points, *133*, *178*, *188*, *199*
polygon, *125*, *134*
POSIXct, *116*, *145*
predict.tidem, 219, *344*, *346*
pressure (accessors), 9
pressure<- (accessors), 9
pretty, *41*, *45*, *51, 52*, *101*, *124*
prettyPosition, 220
processingLog, 221
processingLog<- (processingLog), 221
processingLogItem (processingLog), 221
processingLogShow (processingLog), 221
pwelch, 222

quantile, *106*

range, *51*
rangeExtended, *51*, 224
rangeLimit, 224
read.adp, *14*, *39*, *41*, *67, 68*, *82*, *108*, *167*, *169*, 225, *289*, *351*, *366*
read.adp.nortek, *14*, *290*
read.adp.rdi, *14*, *290*
read.adp.sontek, *14*
read.adp.sontek.serial, *14*
read.adv, *16*, *40*, *67, 68*, *83*, *108*, *172*, *227*, 229, *290*, *351*, *367, 368*
read.adv.nortek, *16*
read.adv.sontek.adr, *16*
read.adv.sontek.text, *16*
read.aquadopp (read.adp), 225
read.aquadoppHR (read.adp), 225
read.aquadoppProfiler (read.adp), 225
read.cm, *48*, *175*, 234, *291*
read.cm.s4, *291*
read.coastline, *21*, *49*, *177, 178*, 236, *254*, *292*
read.csv, *364*
read.ctd, *11*, *23*, *56–58*, *60*, *62*, *64*, *121*, *166*, *180*, *210*, *212, 213*, *219*, 238, *254*, *293*, *364*
read.drifter, *78*, *184*, 242, *294*
read.echosounder, *79, 80*, *186*, 244, *295*
read.gps, *26*, *96*, *188*, 245
read.landsat, *110, 111*, *190*, 246, *297*
read.lisst, *27*, *114*, *191*, 247, *297*
read.lobo, *116*, *193*, 249, *254*, *298*
read.met, *29*, *141*, *194*, 251, *299*
read.observatory, 252
read.oce, *14*, *16*, *37*, *110*, *163*, *231*, *248, 249*, 253, *290–293*, *295*, *297, 298*, *300, 301*, *304*

read.sealevel, *31*, *196*, *254*, 254, *300*, *344*, *346*
read.section, *32*, *121*, 256, *270*, *301*
read.tdr, *33*, *201*, *254*, 258, *336*, *338*
read.topo, *34*, *100*, *204*, *206*, 259, *304*, *353*
regexpr, *241*
rescale, 260
resizableLabel, 261
retime, 262
rgb, *52*
roll, *350*
roll *(*accessors*)*, 9
roll<- *(*accessors*)*, 9
round, *73*
runlm, 263
runmed, *70*

salinity *(*accessors*)*, 9
salinity<- *(*accessors*)*, 9
sealevel, 265, *267*
sealevel-class, 266
sealevelTuktoyaktuk, *265*, 267, *346*
secondsToCtime, *65*, 268
section, *55*, *121*, 269, *270*
section-class, 270
sectionGrid, *121*, *270*, 271
sectionSmooth, *270*, 272
sectionSort, *270*, 273
segments, *84*
seq, *52*, *105*
show, *56*, *80*, *141*
show,adp-method *(*adp-class*)*, 12
show,adv-method *(*adv-class*)*, 15
show,cm-method *(*cm-class*)*, 48
show,coastline-method *(*coastline-class*)*, 48
show,ctd-method *(*ctd-class*)*, 56
show,echosounder-method *(*echosounder-class*)*, 79
show,gps-method *(*gps-class*)*, 96
show,landsat-method *(*landsat-class*)*, 110
show,lisst-method *(*lisst-class*)*, 114
show,lobo-method *(*lobo-class*)*, 115
show,met-method *(*met-class*)*, 140
show,oce-method *(*oce-class*)*, 146
show,sealevel-method *(*sealevel-class*)*, 266
show,section-method *(*section-class*)*, 270
show,tdr-method *(*tdr-class*)*, 336
showMetadataItem, 274
siderealTime, 275
sigmaTheta *(*accessors*)*, 9
sigmaTheta<- *(*accessors*)*, 9
silicate *(*accessors*)*, 9
silicate<- *(*accessors*)*, 9
smooth, *70*, *164*, *165*, *186*, *318*
smooth.spline, *60*, *272*, *273*, *318*, *321*
smoothScatter, *168*, *170*, *173*, *174*, *176*, *182*, *202*, *212*, *218*
soi, 276
spectrum, *165*, *222*, *223*
spice *(*accessors*)*, 9
standardDepths, *271*, 277
str, *358*
strptime, *101*, *168*, *173*, *176*, *184*, *186*, *191*, *194*, *202*, *360*
sub, *241*
subset, *263*, *363*
subset,adp,missing-method *(*subset.adp*)*, 278
subset,adp-method *(*subset.adp*)*, 278
subset,adv,missing-method *(*subset.adv*)*, 279
subset,adv-method *(*subset.adv*)*, 279
subset,cm,missing-method *(*subset.cm*)*, 280
subset,cm-method *(*subset.cm*)*, 280
subset,coastline,missing-method *(*subset.coastline*)*, 281
subset,coastline-method *(*subset.coastline*)*, 281
subset,ctd,missing-method *(*subset.ctd*)*, 282
subset,ctd-method *(*subset.ctd*)*, 282
subset,echosounder,missing-method *(*subset.echosounder*)*, 283
subset,echosounder-method *(*subset.echosounder*)*, 283
subset,lisst,missing-method *(*subset.lisst*)*, 284
subset,lisst-method *(*subset.lisst*)*, 284
subset,oce,missing-method *(*subset.oce*)*, 285
subset,oce-method *(*subset.oce*)*, 285
subset,sealevel,missing-method *(*subset.sealevel*)*, 285
subset,sealevel-method

(subset.sealevel), 285
subset,section,missing-method (subset.section), 286
subset,section-method (subset.section), 286
subset,tdr,missing-method (subset.tdr), 287
subset,tdr-method (subset.tdr), 287
subset,topo,missing-method (subset.topo), 288
subset,topo-method (subset.topo), 288
subset.adp, 278
subset.adv, 279
subset.cm, 280
subset.coastline, 281
subset.ctd, 282
subset.data.frame, *278–288*
subset.echosounder, *80*, 283
subset.lisst, 284
subset.oce, 285
subset.sealevel, 285
subset.section, *270*, 286
subset.tdr, 287
subset.topo, 288
subtractBottomVelocity, 289
summary, *114*
summary,adp,missing-method (summary.adp), 289
summary,adp-method (summary.adp), 289
summary,adv,missing-method (summary.adv), 290
summary,adv-method (summary.adv), 290
summary,cm,missing-method (summary.cm), 291
summary,cm-method (summary.cm), 291
summary,coastline,missing-method (summary.coastline), 292
summary,coastline-method (summary.coastline), 292
summary,ctd,missing-method (summary.ctd), 293
summary,ctd-method (summary.ctd), 293
summary,drifter,missing-method (summary.drifter), 294
summary,drifter-method (summary.drifter), 294
summary,echosounder,missing-method (summary.echosounder), 295
summary,echosounder-method (summary.echosounder), 295
summary,gps,missing-method (summary.gps), 296
summary,gps-method (summary.gps), 296
summary,landsat,missing-method (summary.landsat), 296
summary,landsat-method (summary.landsat), 296
summary,lisst,missing-method (summary.lisst), 297
summary,lisst-method (summary.lisst), 297
summary,lobo,missing-method (summary.lobo), 298
summary,lobo-method (summary.lobo), 298
summary,met,missing-method (summary.met), 299
summary,met-method (summary.met), 299
summary,sealevel,missing-method (summary.sealevel), 300
summary,sealevel-method (summary.sealevel), 300
summary,section,missing-method (summary.section), 301
summary,section-method (summary.section), 301
summary,tdr,missing-method (summary.tdr), 302
summary,tdr-method (summary.tdr), 302
summary,tidem,missing-method (summary.tidem), 303
summary,tidem-method (summary.tidem), 303
summary,topo,missing-method (summary.topo), 304
summary,topo-method (summary.topo), 304
summary,windrose,missing-method (summary.windrose), 305
summary,windrose-method (summary.windrose), 305
summary.adp, 289
summary.adv, 290
summary.cm, *48*, 291
summary.coastline, 292
summary.ctd, *56*, *213*, *219*, 293
summary.drifter, *78*, 294
summary.echosounder, *80*, 295

summary.gps, 296
summary.landsat, 296
summary.lisst, 297
summary.lobo, *116*, 298
summary.met, *141*, 299
summary.sealevel, *266*, 300
summary.section, *270*, 301
summary.tdr, 302, *336*, *338*
summary.tidem, *203*, 303, *345, 346*
summary.topo, *206*, 304
summary.windrose, *35*, *207*, 305
sunAngle, *142*, 306
swAbsoluteSalinity, 307, *314*, *340*
swAlpha, 309
swAlphaOverBeta, 310
swBeta, 311
swConductivity, 312, *323*
swConservativeTemperature, *308*, 313, *340*
swDepth, *56*, *199*, 314, *319*
swDynamicHeight, 315
swLapseRate, 316
swN2, *56*, *182, 183*, *194*, *210*, *212*, 317
swPressure, 319
swRho, 320, *324–326*
swRrho, 321
swSCTp, 322
swSigma, *321*, 323, *325, 326*
swSigmaT, *321*, *324*, 324, *326*
swSigmaTheta, *240*, *321*, *324, 325*, 325, *333*
swSoundAbsorption, 326
swSoundSpeed, 327
swSpecificHeat, 328
swSpice, 329
swSTrho, 330, *334*
swTFreeze, 331
swTheta, 332
swTSrho, *331*, 333
swViscosity, 334
swZ, *56*
swZ *(*swDepth*)*, 314

tail *(*head*)*, 98
tdr, 335
tdr-class, 336
tdrPatm, *336*, 337
tdrTrim, *336*, 338
temperature *(*accessors*)*, 9
temperature<- *(*accessors*)*, 9
teos, *73*, *181*, *198*, *211*, *217, 218*, *330*, *334*, 339, *342*
teosSetLibrary, 341
text, *136*
threenum, 342
tidedata, 343
tidem, *203*, *216*, *219*, *303*, 344, *348, 349*
tidem-class, 347
tidemAstron, 347
tidemVuf, 348
time, *10*
time *(*accessors*)*, 9
time.oce, 349
toEnu *(*beamToXyz*)*, 37
toEnuAdp, *14*, *38*, 350
toEnuAdv, *16*, *38*, 351
topo-class, 352
topoInterpolate, 352
topoWorld, 353
tritium *(*accessors*)*, 9
ts, *30*

unabbreviateYear, 354
undriftTime, 354
ungrid, 355
uniroot, *319*
unwrapAngle, 356
useHeading, 357

vectorShow, 358
velocity, *10*, *350*
velocity *(*accessors*)*, 9
velocityStatistics, 358

webtide, 359
which, *138*
wind, *106*, 361
window.oce, 362
windrose-class, 363
write.ctd, 364
write.table, *152, 153*

x11, *191*
xyzToEnu *(*beamToXyz*)*, 37
xyzToEnuAdp, *14*, *38*, *81*, *169*, *350, 351*, 365
xyzToEnuAdv, *16*, *38*, *82*, *351*, 366