Chapter 3
The oce Package

Abstract The oce package simplifies oceanographic analysis by handling the details of discipline-specific file formats, calculations and plots. Designed for real-world application and developed with open-source protocols, oce supports a broad range of practical work. Generic functions take care of general operations such as subsetting and plotting data, while specialized functions address more specific tasks such as hydrographic analysis, ADCP coordinate transformations, etc. It is easy to document work done with oce, because its functions automatically update processing logs stored within its data objects. Users are not limited to oce functions, however; data are extracted easily from oce objects, so that the thousands of other R packages may be used as needed.

3.1 Package Options

The oce package (Kelley and Richards 2018) has several options to control global behaviour, e.g.

```
options(oceDebug=3)
```

sets debugging to a high level for all oce function calls, as an alternative to setting debug argument in such calls. Some other options are listed in Table 3.1. It is typical to set such things in a startup file (see Sect. 2.2.4).

3.2 File Formats

Table 3.2 lists some of the data formats recognized by `read.oce()`, which uses `oceMagic()` to infer file type, and then calls a specialized function to read the data. These specialized functions may also be called directly. Either way, users are relieved from reading lengthy data-format specifications and writing complex

D. E. Kelley, *Oceanographic Analysis with R*,
https://doi.org/10.1007/978-1-4939-8844-0_3

Table 3.1 Some user-controllable `oce` startup options

Option	Default value	Meaning
`oceMar`	`c(3, 3, 2, 2)`	Value for `par(mar)`, controlling margin widths
`oceMgp`	`c(2.0, 0.7, 0)`	Value for `par(mgp)`, controlling axis label locations
`oceDrawTimeRange`	`TRUE`	Should `oce.plot.ts()` show the time range?
`oceAbbreviateTimeRange`	`TRUE`	Should `oce.plot.ts()` shorten time ranges?
`oceTimeFormat`	`"%Y-%m-%d %H:%M:%S"`	Format for time strings
`oceUnitBracket`	`"["`	Character to embrace units in plot labels; can also be `"("`
`oceEOS`	`"unesco"`	Preferred seawater equation of state; can also be `"gsw"`; see Sect. 5.2.1 and Appendix D

Table 3.2 Some of the oceanographic data formats recognized by `read.oce()` and its helper function, `oceMagic`

Class	Details
`adp`	Acoustic Doppler profiler, in RDI-Teledyne, Nortek or Sontek format
`adv`	Acoustic Doppler velocimeter, in Nortek or Sontek format
`amsr`	AMSR satellite data
`argo`	Argo float data
`bremen`	Data format used at Bremen
`cm`	Current meter, in Interocean format
`coastline`	Coastline shape, in mapgen, shapefile and other formats
`ctd`	CTD, in Seabird `*.cnv`, WOCE exchange, ODF or Ruskin format
`echosounder`	Biosonics scientific echosounder
`g1sst`	Global 1km SST satellite/model data
`gps`	Location data
`ladp`	Lowered Acoustic Doppler profiler
`landsat`	Landsat satellite data
`lisst`	Laser in situ scattering and transmissometry
`lobo`	Land/Ocean biogeochemistry Observatory
`met`	Meteorological data.
`oce`	Base of all classes in the oce package
`odf`	Data format used by Department of Fisheries and Oceans, Canada
`rsk`	RBR logging devices, e.g. temperature-depth recorders
`satellite`	Base of `amsr`, `g1sst` and `landsat` classes
`sealevel`	Sea-level elevation, in MEDS or Hawaii format
`section`	Section data
`tidem`	Tidal-model data
`topo`	Earth topography, in NOAA format
`windrose`	Wind rose data

code,[1] e.g. the specialized graphical representation of CTD data shown in Fig. 3.1 was constructed with a simple call to `plot()`, which tailors its action to the class of its first argument.

The ability to read a wide variety of data types is a good reason to try R and `oce` for oceanographic analysis. There are two main advantages over software provided by manufacturers. First, `oce` is open-source, and thus easy to inspect or modify. Second, manufacturers provide software for just their own instruments, which is of limited help in coordinating data from the typical oceanographic experiment, which employs a variety of instrument types.

Open-source alternatives are available in Matlab and Python, and readers will likely find themselves using these from time to time. A weakness of many such systems is that they tend to be specialized to particular instruments. By contrast, `oce` handles many instruments in a uniform way, which can be helpful to analysts who work with several data types at once. Much of the `oce` uniformity stems from its object-orientation design, discussed in the next section.

3.3 Object Orientation

The `oce` package uses the S4 scheme of object orientation[2] with a hierarchical collection of object classes that inherit from a common base class named "`oce`". This inheritance scheme simplifies the internal coding of `oce`, reducing the chance of bugs and also making it easier for users to add new objects.

All `oce` classes have three S4 "slots," with contents as follows (Fig. 3.2).

- `metadata`, a list describing the object. The contents vary with the object type, perhaps including the name of a data file, the sampling location, etc.
- `data`, a list containing the actual data. Again, the contents depend on the object. For example, CTD objects contain vectors for hydrographic quantities, ADP objects contain vectors for time and distance in addition to arrays for velocity components, etc. (This combination of vectors and arrays explains why a list is used instead of a data frame; see Sect. 2.3.6.)
- `processinglog`, a list containing items named `time` and `value` that record the processing steps that led to present state of the object.

There are two ways to access data within `oce` objects. It is possible to use the `@` symbol to access information stored in an object's `data` or `metadata` slots. However, the recommended method is to the access operator "`[[`", e.g.

[1]The effort of decoding oceanographic data files can be significant, e.g. Teledyne-RDI (2007) devotes nearly 30 pages to byte-level format of ADCP files, and following that format requires several hundred lines of R and C/C++ code.

[2]There is no need to understand S4 in order to use `oce`, but curious readers can get the gist from `help(Classes)` or Chapter 9 of Chambers (2008).

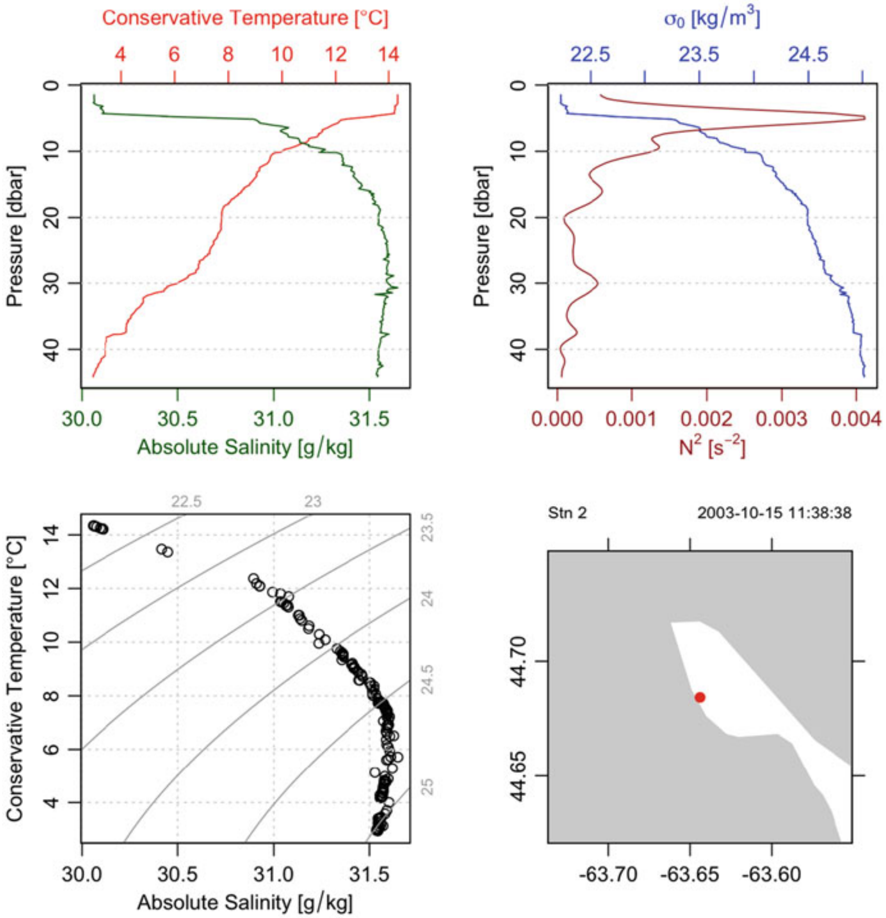

Fig. 3.1 Hydrographic diagram of a CTD cast made in Halifax Harbour by students in the author's Physical Oceanography class at Dalhousie University. This diagram was produced with just two oce function calls: one to read the data, another to plot them

```
data(ctd, package="oce")
head(ctd[["temperature"]])
 [1] 14.22109 14.22649 14.22509 14.22219 14.22669
14.23318
```

There are two advantages of the accessor approach. First, it isolates users from the details of internal storage, letting users write code that is resistent to any changes in the internal structure of oce objects that may be necessitated by changes in instrumentation or analysis methodologies. Second, accessors make it easy for users to infer derived quantities that are not actually stored in the data object, e.g. potential temperature for a CTD or attenuation-corrected backscatter strength for an ADCP.

The "[[" operator works for assignment as well as access, e.g. temperature might be increased with

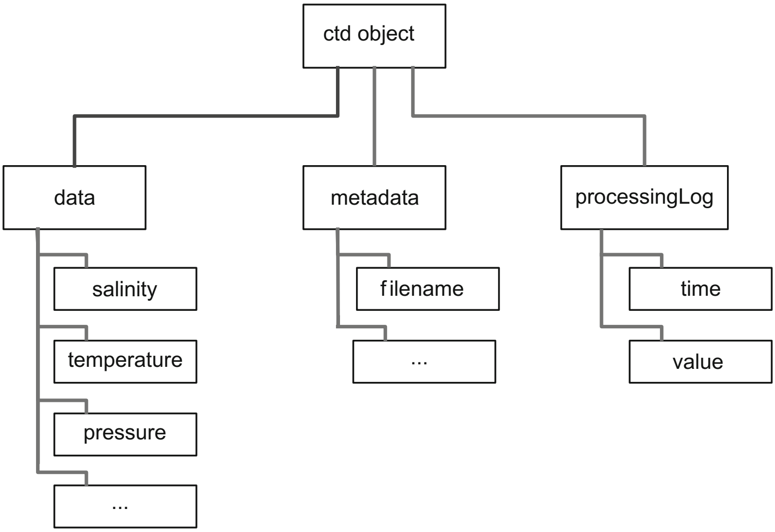

Fig. 3.2 Structure of a CTD object

```
ctd[["temperature"]] <- 0.001 + ctd[["temperature"]]
```

but this scheme should be used only for quantities actually stored in the object, not for derived quantities. (For nontrivial changes, however, it is recommended to use `oceSetData()` and `oceSetMetadata()`, since these record changes within the object's processing log.)

Internally, "`[[`" is a function that searches through the object for the named quantity. It first examines the `metadata` slot, returning the value (for access or assignment) if found, otherwise moving on to the `data` slot and repeating the test. This scheme permits a uniform notation, no matter which slot holds the information. This is useful because the appropriate slot depends on the object class, e.g. `latitude` is mandatory for a `coastline` object so it belongs in the `data` slot, but it is an optional addition for CTD instruments, so it belongs in the `metadata` slot. The code fragment `a[["latitude"]]` works for both types of object, taking values from different places depending on the class of `a`. This scheme makes it easy for an analyst to work with a wide range of data types without case-by-case tailoring of code.

3.4 Datasets

Several datasets are provided with `oce` and `ocedata`, some of which are listed in Tables 3.3 and 3.4. The dataset documentation can be a useful adjunct to the help on its related class; e.g. compare the output of `help("ctd")` with the more detailed information that `help("ctd-class")` provides.

Table 3.3 Datasets provided in the oce package

Name	Description
adp	SLEIWEX ADCP measurements
adv	SLEIWEX ADV measurements
argo	Argo float #3900388 measurements
cm	SLEIWEX S4 current meter measurements
coastlineWorld	Default (1:50M) world coastline
colours	Colours used in some oce palettes
ctd	CTD profile collected in Halifax Harbour
ctdRaw	Raw CTD data, including calibration and upcast
echosounder	SLEIWEX echosounder measurements
landsat	Data from a Landsat image
lisst	LISST dataset, constructed artificially
lobo	LOBO measurements made in Halifax Harbour
met	Meteorological observations at Halifax Int'l Airport
rsk	SLEIWEX temperature-depth recorder data (RBR logger)
sealevel	Sea-level variation within Halifax Harbour during 2003
sealevelTuktoyaktuk	Sea-level variation near Tuktoyaktuk, from Foreman (1977)
section	WOCE hydrographic section designated A03
tidedata	Data on tidal constituents, used by tidem()
topoWorld	World topography data on a 12-minute grid
wind	Wind data in Koch et al. (1983)

3.5 Functions

The oce package provides generic functions (Sect. 2.3.11.6) to handle common tasks, including the access operator mentioned above, along with subset(), summary() and plot(). This scheme lets users ignore the internal structure of the data, e.g. if d is an oce object including time, then

```
dd <- subset(d, time < mean(range(d[["time"]],
na.rm=TRUE)))
```

retrieves data from the early portion of the sampling interval, no matter the object's class. Plotting is also done with generic functions, e.g. Fig. 3.1 was produced with

```
data(ctd, package="oce")
plot(ctd)
```

where the plot details are obtained with either of the following:

```
help("plot,ctd-method")
?"plot,ctd-method"
```

In addition to generic functions, oce provides a long list of functions for specialized oceanographic tasks, including (with * representing several function names)

- map*() functions for drawing maps with projections (see Sect. 3.6)

Table 3.4 Datasets provided in the `ocedata` package

Name	Description
`RRprofile`	Hydrographic profile from Reiniger and Ross (1968)
`beaufort`	A CTD profile in the Beaufort Sea
`buoy`	Measurements made by a buoy off Halifax
`coastlineWorldFine`	Fine-resolution (1:10M) world coastline
`coastlineWorldMedium`	Medium-resolution (1:50M) world coastline
`conveyor`	Some points on the Broecker (1991) "conveyor belt"
`drag`	Air-sea drag coefficients from Garratt (1977)
`endeavour`	Path of HMS Endeavour
`geosecs235`	GEOSECS tritium station 235
`giss`	Goddard Institute for Space Studies temperature timeseries
`gs`	Gulf Stream position, from Drinkwater et al. (1994)
`levitus`	"Levitus" World Ocean Atlas SSS and SST
`munk`	Pacific temperature profile examined by Munk (1966)
`nao`	North Atlantic Oscillation timeseries
`oceans`	Geometry of some oceans
`papa`	Measurements at Ocean Weather Station P
`redfieldNC`	Nitrate-carbon data in Figure 3 of Redfield (1934)
`redfieldNP`	Nitrate-phosphate data in Figure 1 of Redfield (1934)
`redfieldPlankton`	Plankton data in Table II of Redfield (1934)
`riley`	Plankton data in Figure 21 of Riley (1946)
`schmitt`	Temperature-salinity data in Figure 1 of Schmitt (1981)
`secchi`	Sechhi-disk measurements in North and Baltic Seas
`soi`	Southern Oscillation Index from 1866
`topo2`	World topography data on a 2-degree grid
`turbulence`	Turbulence measurements by Grant et al. (1962)
`wilson`	Seafloor-spreading data in Table 1 of Wilson (1963)

- `oce.plot.ts()`, an alternative to `plot.ts()` for time-series data
- `imagep()` and `drawPalette()` for colour palettes in images and generally
- `pwelch()` for averaged spectra as discussed by Welch (1967)
- `atm*()` functions relating to atmospheric properties
- `sw*()` functions relating to seawater properties (see Table 3.5 and Sect. 5.2.1)

Exercise 3.1 Use the generic `plot()` for CTD objects, to produce a version of Fig. 3.1 using the UNESCO equation of state instead of the default TEOS-10 version. (See page 209 for a solution.)

Exercise 3.2 (a) Calculate the density of seawater at pressure 100 dbar, salinity 34 PSU, and temperature 10 °C. (b) What temperature would the parcel have if raised adiabatically to the surface? (c) What density would it have if raised adiabatically to the surface? (d) What density would it have if lowered about 100 m,

Table 3.5 Some functions relating to seawater properties

Function	Description
`swAbsoluteSalinity()`	Absolute salinity, S_A
`swAlpha()`	Thermal expansion coefficient, $\alpha = -\rho_0^{-1}\partial\rho/\partial T$
`swAlphaOverBeta()`	Ratio of thermal and haline coefficients, α/β
`swBeta()`	Haline contraction coefficient, $\beta = \rho_0^{-1}\partial\rho/\partial S$
`swConductivity()`	Electrical conductivity, C
`swConservativeTemperature()`	Conservative temperature, Θ
`swDepth()`	Depth, $-z$, inferred from p and latitude
`swDynamicHeight()`	Dynamic height
`swLapseRate()`	Adiabatic lapse rate
`swN2()`	Square of buoyancy frequency, N^2
`swRho()`	Density, $\rho = \rho(S, T, p)$
`swSCTp()`	S inferred from conductivity, T and p
`swSTrho()`	S inferred from T and ρ
`swSigma()`	$\sigma = \rho - 1000\,\mathrm{kg/m^3}$
`swSigmaT()`	$\sigma(S, T, 0)$
`swSigmaTheta()`	$\sigma(S, \theta, 0)$
`swSoundAbsorption()`	Sound absorption
`swSoundSpeed()`	Sound speed
`swSpecificHeat()`	Specific heat
`swSpice()`	Spiciness, a property orthogonal to density
`swTFreeze()`	Freezing temperature
`swTSrho()`	T inferred from S and ρ
`swTheta()`	Potential temperature, θ
`swViscosity()`	Dynamic viscosity, μ
`swZ()`	Vertical coordinate, z, inferred from p and latitude

Here, C represents electrical conductivity, p pressure, S salinity and T in situ temperature

increasing the pressure to 200 dbar? (e) Draw a blank T-S diagram with S from 30 to 40 PSU and T from -2 to 20 °C. (See page 209 for a solution.)

Exercise 3.3 Use `propagate` from the `propagate` package to estimate typical CTD salinity uncertainty. (See page 209 for a solution.)

3.6 A Practical Example

Figure 3.3 shows annual-mean world sea-surface temperature (SST) from the 2009 version of the World Ocean Atlas dataset (Locarnini et al. 2010; Antonov et al. 2010). A detailed explanation of the construction this diagram provides the chance to highlight some important `oce` functions. The first step is to access the data,

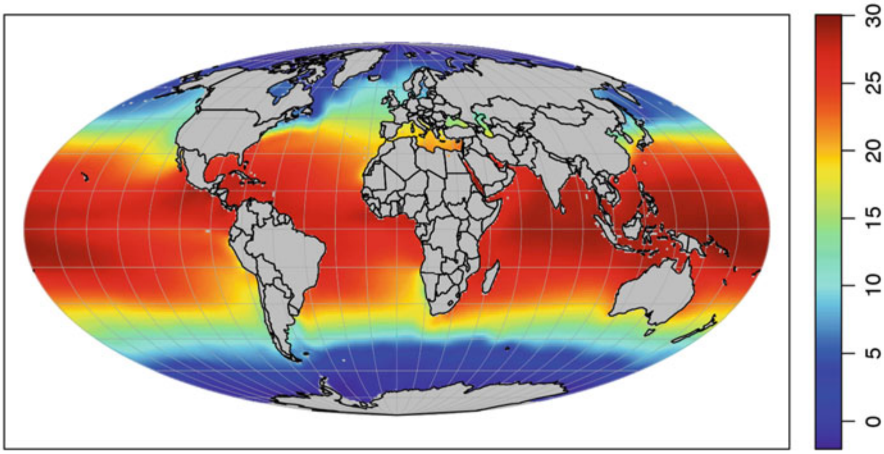

Fig. 3.3 Annual-mean sea surface temperature shown in Mollweide projection

a convenient (but spatially coarse) form of which is provided by the `ocedata` package:

```
data(levitus, package="ocedata")
```

Although `oce` can easily select a colour-scale for the image, analysts usually prefer to set such things to achieve uniformity across plots, and this may be done with

```
cm <- colormap(zlim=c(-2, 30), col=oceColorsJet)
```

which uses the "jet" color mapping (see Sect. 2.4.14). Then a palette is drawn with

```
drawPalette(colormap=cm)
```

At a global scale, the coastline provided with `oce` provides sufficient detail and the Mollweide projection may be a good choice (see Appendix C for more on projections); with these choices,

```
data(coastlineWorld, package="oce")
mapPlot(coastlineWorld, projection="+proj=moll",
grid=FALSE)
```

draws the gray land area in Fig. 3.3. Finally,

```
mapImage(levitus$longitude, levitus$latitude,
         levitus$SST, colormap=cm)
```

adds the sea-surface temperature. Readers who are following along will notice that some of the image grid elements are painting over the land. This problem is alleviated by redrawing that land, after first drawing lines of longitude and latitude:

```
mapGrid()
mapLines(coastlineWorld)
```

thus completing Fig. 3.3.

Readers might wish to examine the documentation of the relevant functions to understand this example fully, but the above should indicate the potential of `oce` to produce useful specialized oceanographic plots, in addition to those offered by its generic functions. The main thing to realize is that oce is built with R base graphics, which means that a painting model is employed, with new graphical elements being put on top of existing ones.

Exercise 3.4 Map ocean-surface density. (See page 210 for a solution.)

Exercise 3.5 Use mapPlot() to draw a world coastline with the Robinson projection, and trace the 1700s H.M.S. Endeavour cruise. (See page 210 for a solution.)

3.7 Evolution of oce

The oce package began with ad hoc code to read CTD files stored in the ".cnv" format (see Exercise 2.44). This was a main program that consisted of little more than a call to read.table() to read a specified file, with the value of the skip argument chosen after inspection of the lines at the start of that particular file.

Headers in .cnv files are of variable length, and it is tedious to alter skip for each case, so the next step was to determine the header length by using grep() to detect the end of the header. As more files were considered, it became desirable to infer data columns from the header, instead of specifying them manually. Other features were added as applications widened, and to avoid confusion the code was recast as a function that returned not just columnar data, but also other (meta-data) quantities, such as station number, sampling location, etc., which are sometimes present in CTD headers. With such additions, formal documentation became necessary, because even the author found it difficult to remember the features without examining the code. For oce, as perhaps for other packages, this was the time when the effort of packaging was seen to be worthwhile, in order not just to bind documentation and code together, but also to take advantage of the checks of code and documentation that are involved in R packaging and, importantly, to create a system that would benefit colleagues.

From the early stages, a version control system was used to track changes to the oce source code. Between 2007 and 2010, the subversion system was used, but then a switch was made to git. The code was originally hosted on a website on the author's desktop computer, but as the user base grew, it was moved to Google,[3] where it was called r-oce because the name oce had been taken. Then, in 2010, oce was moved to GitHub,[4] where it resides today, benefiting from the collaboration of additional authors and the advice and bug reports of users from around the world.

[3] code.google.com/p/r-oce.

[4] github.com/dankelley/oce.

The official version of `oce` is available on the CRAN[5] servers and may be installed with `install.packages()`. The Github website provides updates between official releases, and it is also used by those requesting new features, reporting bugs or otherwise helping with `oce` development.

It is worth noting that additions to `oce` are always based on the practical needs of the authors and their colleagues, never on some imagined needs. For example, support for CTD data was followed quickly by support for oceanographic sections, with the `section` class being added as a second child to the parent `oce` class. As the authors started working with acoustical instruments, support was added for acoustic-Doppler profilers (`adp`) and velocimeters (`adv`). This continued, one instrument at a time, with `oce` gradually growing to offer support for most instruments in common use today.

Generally, `oce` functions were developed to work on data in the authors' possession, often data under active study in a research program. An advantage of this (beyond satisfying individual research needs) was the early detection of coding errors or poor design. Through time, new features were increasingly based on requests from users, often as articulated on the development website[4]. By design, this scheme directs coding effort first and foremost to issues of high relevance to the oceanographic community.

At this point in the text, readers should be able to apply R to their own work, relying on `oce` to handle quirky data formats and produce diagrams in the oceanographic convention. However, as with any tool, there are dangers in forming habits based on success in early tests. For this reason, the remainder of this book addresses practical aspects to using R for oceanographic analysis, starting with a re-analysis of the data in some classic research articles, and then turning attention to more modern and technical issues.

[5] cran.r-project.org.