




oce: an R package for Oceanographic Analysis

Dan E. Kelley^{*1}, Clark Richards², and Chantelle Layton³

¹ Dan E. Kelley, Professor, Dalhousie University ² Clark Richards, Research Scientist, Bedford Institute of Oceanography, Department of Fisheries and Oceans, Canada; also Adjunct Professor, Dalhousie University ³ Chantelle Layton, Physical Scientist, Bedford Institute of Oceanography, Department of Fisheries and Oceans, Canada

DOI:

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Submitted:

Published:

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC-BY](#)).

Statement of Need

Oceanographic field experiments often employ a wide variety of instrument types, each of which reports data in an individual format. Many of these formats are quite complex, and decoding them can be a challenge, partly owing to weaknesses in the documentation provided by manufacturers. Although manufacturers tend to provide software for viewing data from their instruments, it is usually proprietary and closed-source, which does not help researchers who need to investigate the data in new ways, and to combine data from multiple instruments. The `oce` package (Kelley, Richards, & Layton, 2021) was developed to simplify data input, and to do so in an integrated way that facilitates open-ended analysis that employs multiple data types. It also provides tools for undertaking specialized oceanographic calculations (e.g., relating to seawater properties) and providing graphical displays that follow oceanographic conventions. This is all done in the R language (Ihaka & Gentleman, 1996; R Core Team, 2021), which provides wide-ranging data analysis tools that are needed for oceanographic research (Kelley, 2018).

Overview

The `oce` package has been hosted on CRAN (“The Comprehensive R Archive Network,” 2021) since the year 2009. The CRAN version, which is updated once or twice a year, may be installed by typing `install.packages("oce")` in an R console. Users who need newer features may use `remotes::install_github("dankelley/oce", ref="develop")` to download and build the development branch. Those wishing to investigate or participate in the development process are welcome to do so, at <https://github.com/dankelley/oce>.

The package has functions for decoding a long list of instrument-specific data formats. The return values of those functions are expressed in the S4 object-orientation scheme, with slots for (a) the actual data, (b) related metadata, and (c) a log of the `oce` function calls used in creating the object. This is illustrated by the results of executing the following in an R session, for the case of an object of the `"ctd"` subclass, which holds data acquired from a CTD (Conductivity-Depth-Pressure) instrument, a workhorse of oceanographic sampling.

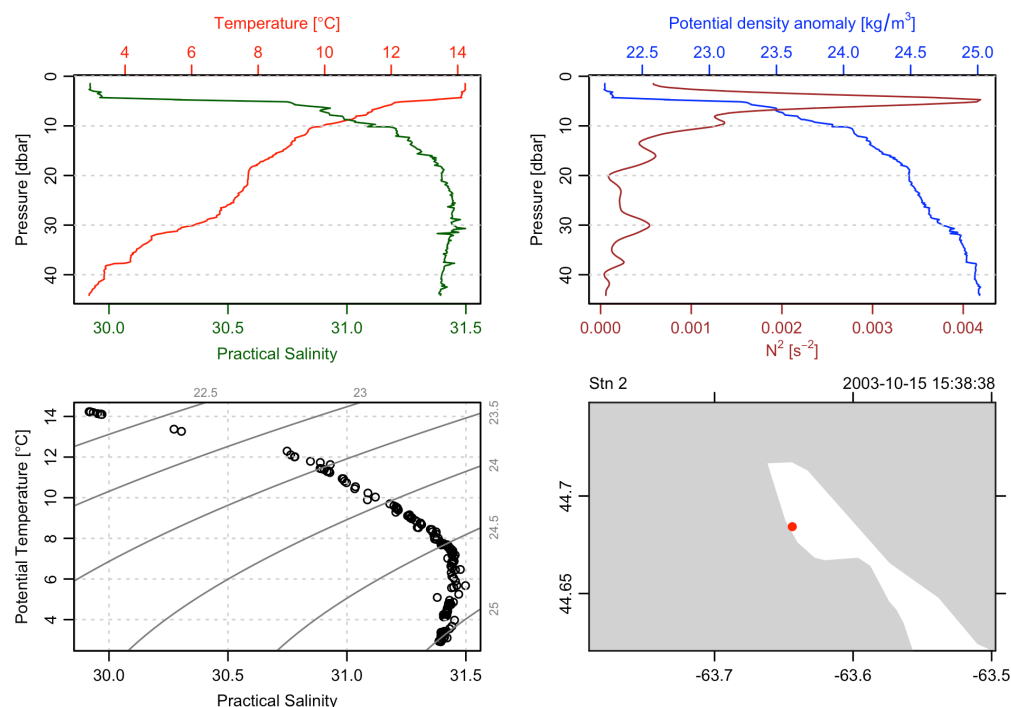
```
library(oce)           # load library
data(ctd)              # load a built-in sample file
slotNames(ctd)         # see 'slot' names
```

The last function call shown above is provided to explain the structure, but it is seldom used in practice. Rather, the next step after loading an object, or reading it from a data file, is often to view textual and graphical summaries of the data, with

^{*}Corresponding author.

```
summary(ctd)
plot(ctd)
```

results not shown here
results in Figure 1



the latter of which produces Figure 1. This diagram is good for an overview, but `oce` also provides more fine-grained control of graphical representations, with e.g.

```
plot(ctd, which="temperature")
```

plotting just temperature variation with depth (results not shown here). The variations of other properties may be shown by setting `which` appropriately, and this argument can also be used to specify other types of plots, in addition to the depth-variation form.

Besides this "ctd" subclass, `oce` supports over two dozen other subclasses that cover a wide range of oceanographic instrumentation. In every case, the same "`summary()`" and "`plot()`" function calls provide textual and graphical representations of the contents of the object. This specialization of these two generic functions simplifies analysis considerably. For example, if `PATTERN` is a regular expression that specifies a set of data files, whether of a single instrument type or multiple instrument types, then

```
for (file in list.files(PATTERN)) {
  d <- read.oce(file)
  summary(d)
  plot(d)
}
```

will provide information about each data file of interest, forming a good first stage of analysis. More detailed plot calls are usually required for the second stage of analysis, but a feature of `oce` (and R) is that it is often quite easy to undertake such analyses.

`Oce` provides other generic functions, such as `subset()` for focusing on subsets of data, `handleFlags()` for processing data-quality flags, and `[[` for accessing data. This accessor operation is particularly worthy of note, for several reasons, of which two stand out.

1. `[[` lets users access information regardless of where it occurs in the object. For example, a CTD does not measure longitude and latitude, but these things are

often streamed into data files separately, so they are stored in the object's `metadata` slot, not the `data` slot. Users do not need to know this detail, though, because e.g. `ctd[["longitude"]]` will extract longitude from whichever slot holds it. For other data types, e.g. from a GPS instrument, the `[[` operation would get it from the `data` slot. Importantly, a single line of code would work for both types of data, so, again, code written for one object type will work for another.

2. `[[` can access not just information stored within the object, but also things that can be calculated from that information. For example, CTD files typically hold temperature along with other properties from which seawater density may be computed (McDougall & Barker, 2011; Millero, 2010). Density is an important dynamical factor, and so `oce` is set up so that e.g. `ctd[["density"]]` will compute it, if the object named `ctd` contains the required elements. This also applies for other elements that are commonly needed for oceanographic analysis, but not measured directly.

The significance of `[[` is that it forms a bridge from the oceanographic realm to the general R realm, letting users extract the data into whatever form will facilitate further analysis. With tens of thousands of well-vetted R packages covering a wide array of statistical and other methods, this lets oceanographers focus on their work, not on tool generation.

Example: Tidal Analysis

A more detailed example may help to solidify some of the key aspects of `oce`. Many readers will have an interest in tides, so we will work with a year-long record of sea level, $\eta = \eta(t)$ in Halifax Harbour, in the year 2003, during which the city was struck by Hurricane Juan.

Consider the code given below, which produces Figure 2. Comments made within the code ought to be reasonably self-explanatory. A built-in sealevel file is used, to make a reproducible example, but replacing the `data()` call with a `read.sealevel()` call will handle data files in standard formats. Note that the `tidem()` function is quite sophisticated, covering over 500 lines of R code in order to apply specialized procedures developed in the tidal-research community (Foreman, Cherniawsky, & Ballantyne, 2009; Godin, 1972; Pawlowicz, Beardsley, & Lentz, 2002). Readers with R experience may notice that the function name evoke the `lm()` function for linear models, and they may not be surprised that `oce` provides a function named `predict()`, for generating tidal predictions from tidal models.

```
library(oce)                                # load library
data(sealevel)                              # use built-in example dataset
t <- sealevel[["time"]]                     # extract time
eta <- sealevel[["elevation"]]              # extract sea level
m <- tidem(sealevel)                        # fit tidal model
etaDetided <- eta - predict(m)              # de-tide observations
par(mfrow=c(2, 1))                         # set up a two-panel plot
oce.plot.ts(t, eta, xaxs="i",               # top: observed sea level
            grid=TRUE, ylab="Sea level [m]")
oce.plot.ts(t, etaDetided, xaxs="i",        # bottom: de-tided sea level
            grid=TRUE, ylab="De-tided sea level [m]")
```

A comparison of the panels of Figure 2 reveals that tides explain much of the sea level variation in Halifax Harbour. The lower panel illustrates an increase of detided variance during the winter months, as is expected for a site at a northern mid-latitude. More surprising is the large spike towards the end of September. This is a result of Hurricane Juan, which swept over Halifax at that time, causing a storm surge of in sea level amounting