The OCE package

Dan E. Kelley

November 18, 2008

Abstract

The `oce` package makes it easy to read, summarize and plot data from a variety of Oceanographic instruments, isolating the researcher from the quirky data formats that are common in this field. It also provides functions for working with basic seawater properties such as the equation of state, and with derived quantities such as the buoyancy frequency. Although simple enough to be used in a teaching context, `oce` is powerful enough for a research setting. These things are illustrated here with practical examples.

1 Introduction

Oceanographers must deal with measurements made by a wide variety of instruments, a task that is complicated by the delight instrument manufacturers seem to take in inventing new data formats. The manufacturers often provide software for scanning the data files and producing some standard plots, but this is of limited use to researchers who work with several instrument types at the same time, and who need to carry the analysis beyond the first step.

The need to scan diverse data files was one motivation for the creation of `oce`, but an equal goal was to make it easy to work with the data once they are in the system. This was accomplished partly by the provision of functions to work with the data, and partly by developing a uniform object design that lets users reach inside without guesswork.

At the core of the `oce` design process is a policy of adding features according to the priorities of practical research. As a result, `oce` is a fairly comfortable tool today, and it should remain so as it grows.

2 Object design

As illustrated in Figure 1, each `oce` object is a list containing three elements: (a) `data`, a list or a data frame containing the actual data, for a CTD object, this will contain pressure, temperature, etc., (b) `metadata`, a list containing data such things as file headers, the location of a CTD cast, etc., and (c) `processing.log`, a list that documents how the file was created (often by a `read` or `as` method) and how it was changed thereafter (e.g. by decimating a CTD cast). Elements in the `metadata` can be edited with `oce.edit()`, a function that adds a line to the object's changelog file for every change that is made. The uniformity of the various `oce` objects makes it easy to build skill in examining and modifying objects.

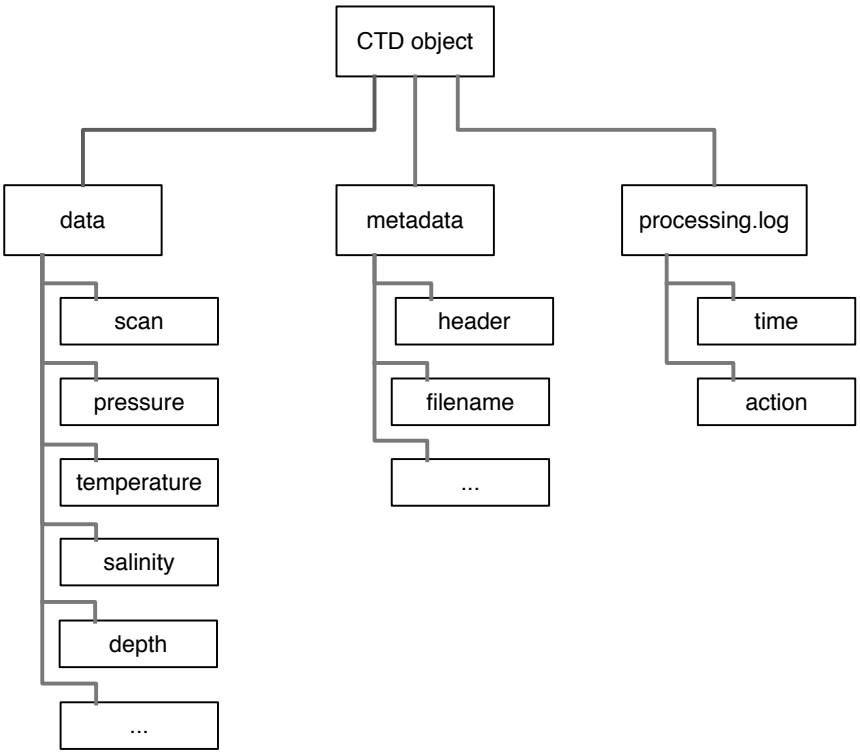

Figure 1: Sketch of the contents of a `ctd` object. The elements `data`, `metadata`, and `processing.log` are shared by all `oce` objects, although their contents vary from object to object.

3 Calculations of seawater properties

The `oce` package provides many functions for dealing with seawater properties. Probably the most used is `sw.rho(S,T,p)`, which computes seawater density ρ as a function of salinity S (PSU), in-situ temperature T (°C) and pressure p (decibar). (Note that this and similar functions starts with the letters `sw.` to designate that they relate to seawater properties. Future versions of `oce` may include the properties of air, so the prefix is an example of planning, rather than necessity.) The result is a number in the order of $1000\,\mathrm{kg/m^3}$. For many purposes, Oceanographers prefer to use the density anomaly $\sigma = \rho - 1000\,\mathrm{kg/m^3}$, provided with `sw.sigma(S,t,p)`, or its adiabatic cousin σ_θ, provided with `sw.sigma.theta(S,t,p)`.

Most of the functions use the UNESCO formulations of seawater properties, but new formulations may be added as they come into use in the literature. A partial list of seawater functions is as follows: `sw.dynamic.height` (dynamic height), `sw.N2` (buoyancy freqency), `sw.S.C.T.p` (salinity S from C, T and p), `sw.S.T.rho` (S from T and ρ), `sw.T.S.rho` (T from S and ρ), `sw.T.freeze` (freezing temperature), `sw.alpha` (thermal expansion coefficient $\alpha = -\rho_0^{-1}\partial\rho/\partial T$), `sw.beta` (haline compression coefficient $\beta = \rho_0^{-1}\partial\rho/\partial S$), `sw.alpha.over.beta` ($\alpha/\beta$), `sw.conductivity` (conductivity from S, T and p), `sw.depth` (depth from p and latitude), `sw.lapse.rate` (adiabatic lapse rate), `sw.rho` (density ρ from S, T and p), `sw.sigma` ($\rho - 1000\,\mathrm{kg/m^3}$), `sw.sigma.t` ($\sigma$ with p set to zero and temperature unaltered), `sw.sigma.theta` (σ with p set to zero and temperature altered adiabatically), `sw.sound.speed` (speed of sound in m/s), `sw.specific.heat` (specific heat in J/kg/°C), `sw.spice` (a quantity used in double-diffusive research), `sw.theta` (potential temperature in °C), and `sw.viscosity` (viscosity). Details and examples are provided in the documentation of these functions.

Exercise 1. (a) What is the density of a seawater parcel at pressure 100 dbar, with 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 100m, increasing the pressure to 200dbar? (e) Draw a blank TS diagram with S from 30 to 40 PSU and T from -2 to 20°C. (Answers are provided at the end of this document.)

4 CTD data

4.1 Example with pre-trimmed data

To get you started with CTD data, `oce` provides a sample data set that has been trimmed to just the downcast portion of the sampling. (See the next section to learn how to do this trimming.). The commands

```
> library(oce)
> data(ctd)
> plot(ctd)
```

produce Figure 2. You may also get a summary of the data with

```
> summary(ctd)
```

The object used to hold CTD data stores not just the data, but also the raw header sequence, and whatever has been discovered about the dataset by parsing the header; use

```
> names(ctd)
```

to learn about these metadata, and use

```
> names(ctd$data)
```

to find out what sensors were attached to the instrument, thus providing data columns.

Of course, you may apply any `R` techniques to the data in `oce` objects, e.g. `hist(ctd$data$temperature)` would produce a histogram of temperature for the `ctd` object. It is always worth checking, though, to see if `oce` has already defined a function that you may be applying, e.g. `plot.TS` will produce a lovely temperature-salinity diagram, with isopycnals and proper units on the axes.

The package provides facilities for some common operations with oceanographic data, such as trimming CTD profiles with `ctd.trim()`, but of course you may do that sort of work by acting on the data directly, if necessary. Just make sure you realize that the metadata will not be altered if you do that. Also, it is a good idea to add log entries to any objects that you change, by using the `processing.log.append()` function. (You can see an example of this in action with `?section`.)

Exercise 2. Plot a profile of σ_θ and N^2, for just the data in the pycnocline.

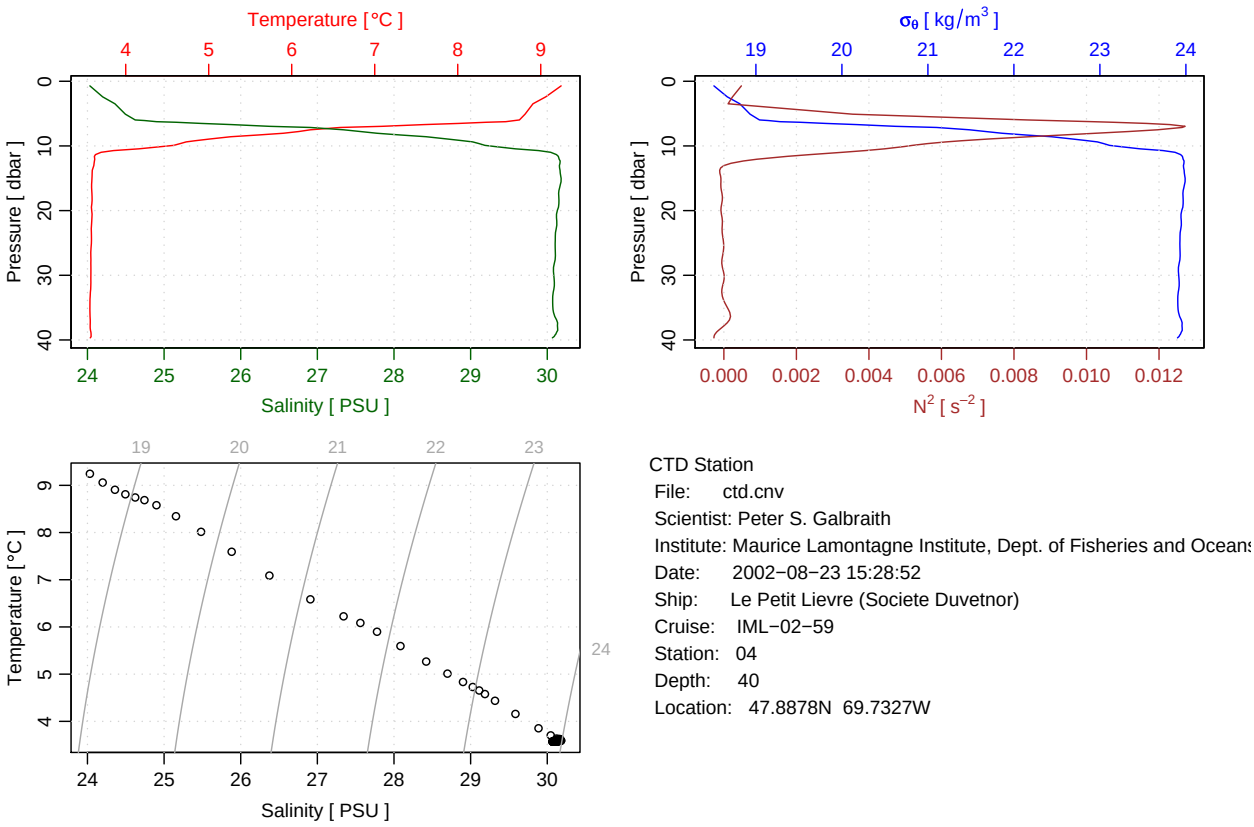

Figure 2: Overview graph of the sample CTD dataset `ctd`, acquired in the St Lawrence Estuary Internal Wave Experiment. (This dataset has been trimmed to the downcast; see the text and Figure 3.)

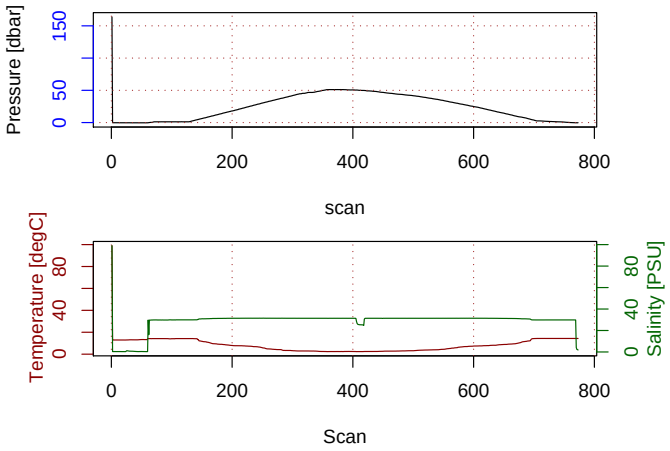

Figure 3: Scanwise plot of the `ctd.raw` sample data set. Note the wild spike at the start, the equilibration phase before the downcast, and the spurious freshening signal near the start of the upcast. See the text for a discussion of how inspection of such graphs can help in trimming CTD data.

4.2 Example with raw data

Practicing Oceanographers may be wondering how the CTD cast used in the preceding section was trimmed of equilibration-phase and upcast-phase data. Data from these sections are spurious and must be trimmed as a first step in processing. For example, consider the following code.

```
> data(ctd.raw)
> plot.ctd.scan(ctd.raw)
```

This produces a two-panel plot (Figure 3) of the data as a time-series, revealing not just the (useful) downcast, but also the subsequent upcast sequence. The x-axis in this plot is the scan number, which is a convenient index for extraction of the downcast portion of the profile by an essentially manual method, e.g. proceeding with a sequence of commands such as

```
> plot.ctd.scan(ctd.trim(ctd.raw, "scan", c(140, 250)))
> plot.ctd.scan(ctd.trim(ctd.raw, "scan", c(150, 250)))
```

This is the "gold standard" method, which is recommended for detailed work. However, for quick work, you may find that the automatic downcast detection scheme works adequately, e.g.

```
> ctd.trimmed <- ctd.trim(ctd.raw)
```

It should be noted that `ctd.trim` inserts entries into the object's log file, so that you (or anyone else with whom you share the object) will be able to see the details of how the trimming was done.

Once the profile has been trimmed, you may wish to use `ctd.decimate()` to smooth the data and interpolate the smoothed results to uniformly-spaced pressure values. For example, a quick examination of a file might be done by the following:

```
> plot(ctd.decimate(ctd.trim(read.ctd("stn123.cnv"))))
```

4.3 Example with WOCE archive data

The package has a harder time scanning the headers of data files in the WOCE archive format than it does in the Seabird format illustrated in the previous examples. This is mainly because front-line researchers tend to work in the Seabird format, and partly because the WOCE format is odd. For example, the first line of a WOCE file is of the form `CTD,20060609WHPOSIODAM` (or `BOTTLE,...`). Scanning the item to the left of the comma is not difficult (although there are variants to the two shown, e.g. `CTDO` sometimes occurs). The part to the right of the comma is more difficult. The first part is a date (yyyymmdd) so that's no problem. But then things start to get tricky. In the example provided, this text contains the division of the institute (WHPO), the institute itself (SIO), and initial of the investigator (DAM). The problem is that no dividers separate these items, and that there seem to be no standards for the item lengths. Rather than spend a great deal of time coding special cases (e.g. scanning to see if the string `WHOI` occurs in the header line), the approach taken with `oce` is to ignore such issues relating to quirky headers. This frees up time to work on more important things, such as plotting the data.

Of course, R provides access to object constituents, so that you are free to do such things as

```
> x <- read.ctd("nnsa_00934_00001_ct1.csv", type = "WOCE")
> x$metadata$institute <- "SIO"
> x$metadata$scientist <- "DAM"
```

but it is *bad practice* to alter metadata in such a way, because doing so alters the object without altering its documentation. The preferred scheme is to do as follows.

```
> x <- read.ctd("nnsa_00934_00001_ct1.csv", type = "WOCE")
> x <- oce.edit(x, "institute", "SIO")
> x <- oce.edit(x, "scientist", "DAM")
```

which will store a note about the changes in the object's log. Even better, provide a reason for the change, and sign the change with your name:

```
> x <- read.ctd("nnsa_00934_00001_ct1.csv", type = "WOCE")
> x <- oce.edit(x, "institute", "SIO", "human-parsed", "Dan Kelley")
> x <- oce.edit(x, "scientist", "DAM", "human-parsed", "Dan Kelley")
```

For a real-world example (with warts!), visit `http://cchdo.ucsd.edu/data_access?ExpoCode=58JH199410` and download the zip file containing the Arctic section called "CARINA", measured in 1994. Expand the zip file, enter the directory, and run the code below.

```
> library(oce)
> files <- system("ls *.csv", intern = TRUE)
> for (i in 1:length(files)) {
+     cat(files[i], "\n")
+     x <- read.ctd(files[i])
+     if (i == 1) {
+         plot.TS(x, xlim = c(31, 35.5), ylim = c(-1, 10), type = "l",
+             col = "red")
+     }
+     else {
+         lines(x$data$salinity, x$data$temperature, col = "red")
+     }
+ }
```

What you'll see is an overall T-S diagram for the entire dataset. It may take a while, since the dataset contains over 90,000 observations. You may note that, even though this is an official, quality-controlled dataset, it is not without problems. The graph that is produced by this code has several spurious lines oriented horizontally (indicating spurious salinity) and vertically (indicating spurious temperature). One way to find such values is to put the lines

```
> print(range(x$data$temperature))
> print(range(x$data$salinity))
```

after the `read.ctd()` command. One thing you'll find is that station 987 has a minimum salinity range of 0.0009 to 987. These values are clearly in error, as are the temperatures at this spot in the file. (It is perhaps revealing that the spurious salinity is equal to the station number.) Indeed, at this spot in the file it can be seen that the pressure jumps from 1342 to 0, and then starts increasing again; the file contains two profiles, or the same profile twice. This is not the only flaw that is revealed by the graph, and by `range` commands; a generous user would spend a week tracking down such issues, and would then contact the data provider (or the chief scientist of the field work) with specific suggestions for correcting the files. The point here is to highlight how this package can be used with real-world data.

4.4 Section plots

The commands

```
> data(section)
> data(coastline.hal)
> plot(section, coastline = coastline.hal)
```

will plot a summary diagram containing sections of T, S, and σ_θ, along with a chart indicating station locations. In addition to such overview diagrams, `plot` can also create individual plots of individual properties.

Exercise 3. Draw a TS diagram for the section data, colour-coded by station

The Halifax section is supplied in a pre-gridded format, but some datasets have different pressure levels at each station. For such cases, the `section.grid` function may be used, e.g.

```
> data(a03)
> Gulf.Stream <- section.subset(a03, 124:102)
> Gulf.Stream.gridded <- section.grid(Gulf.Stream, p = seq(0, 1600,
+     25))
> data(coastline.world)
> plot(Gulf.Stream.gridded, coastline = coastline.world, map.xlim = c(-80,
+     -60))
```

produces Figure 4. The ship doing the sampling was travelling westward from the Mediterranean towards North America, taking 124 stations in total; the `station.indices` value selects the last few stations of the section, during which the ship heading was changed to run in a northwesterly direction, to cross isobaths (and perhaps, the Gulf Stream) at right angles.

Exercise 4. Plot dynamic height across the Gulf Stream, and show the corresponding geostrophic velocity.

5 Coastline and topographic data

Coastline data are available from a variety of sources. The NOAA site `http://www.ngdc.noaa.gov/mgg_coastline/` is particularly popular, and it has the advantage of providing data in Splus format. The function `read.coastline` can handle reading that format (plus some other formats), and `plot` on the resulting object will produce a simple coastline map. The only real advantage over plotting things yourself is that latitude and longitude are scaled to give natural shapes near the centre of the plot.

Bathymetric charts, or more generally topographic maps, can be produced easily. A sample data set is provided, so that

```
> library(oce)
> data(topo.maritimes)
> plot(topo.maritimes, xlim = c(-66, -58), ylim = c(44, 50), water.z = c(-50,
+     -100, -150, -200, -300, -400, -500, -1000, -2000), water.lwd = c(1,
+     1, 1, 1, 1, 1, 1.5, 1.5, 1.5))
```

will produce a chart of the waters embraced by the Maritime Provinces of Canada (Figure 5).

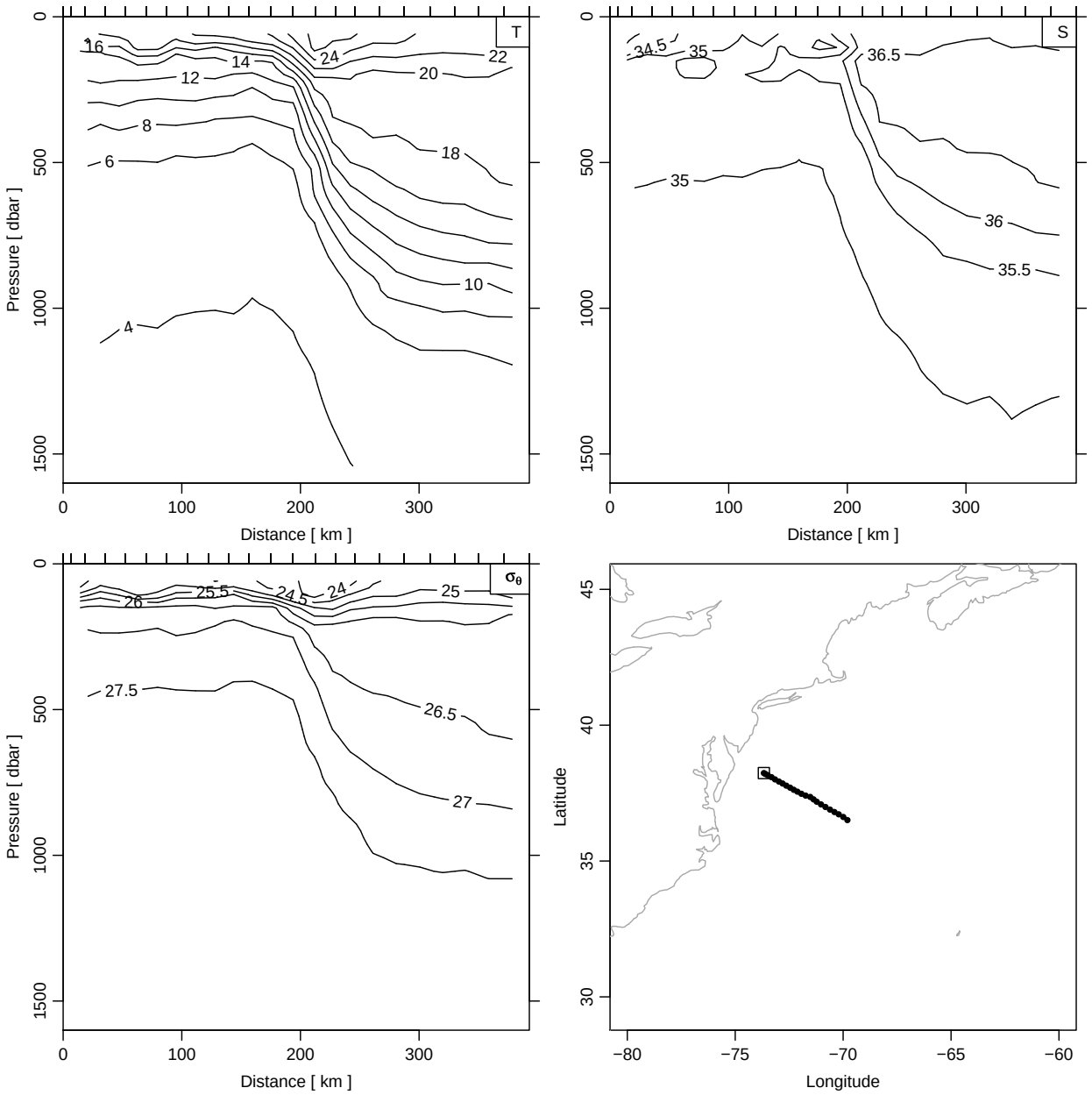

Figure 4: Portion of the CTD section designated A03 (Cruise chief scientist: Tereschenkov, SOI.), showing the region of the Gulf Stream.

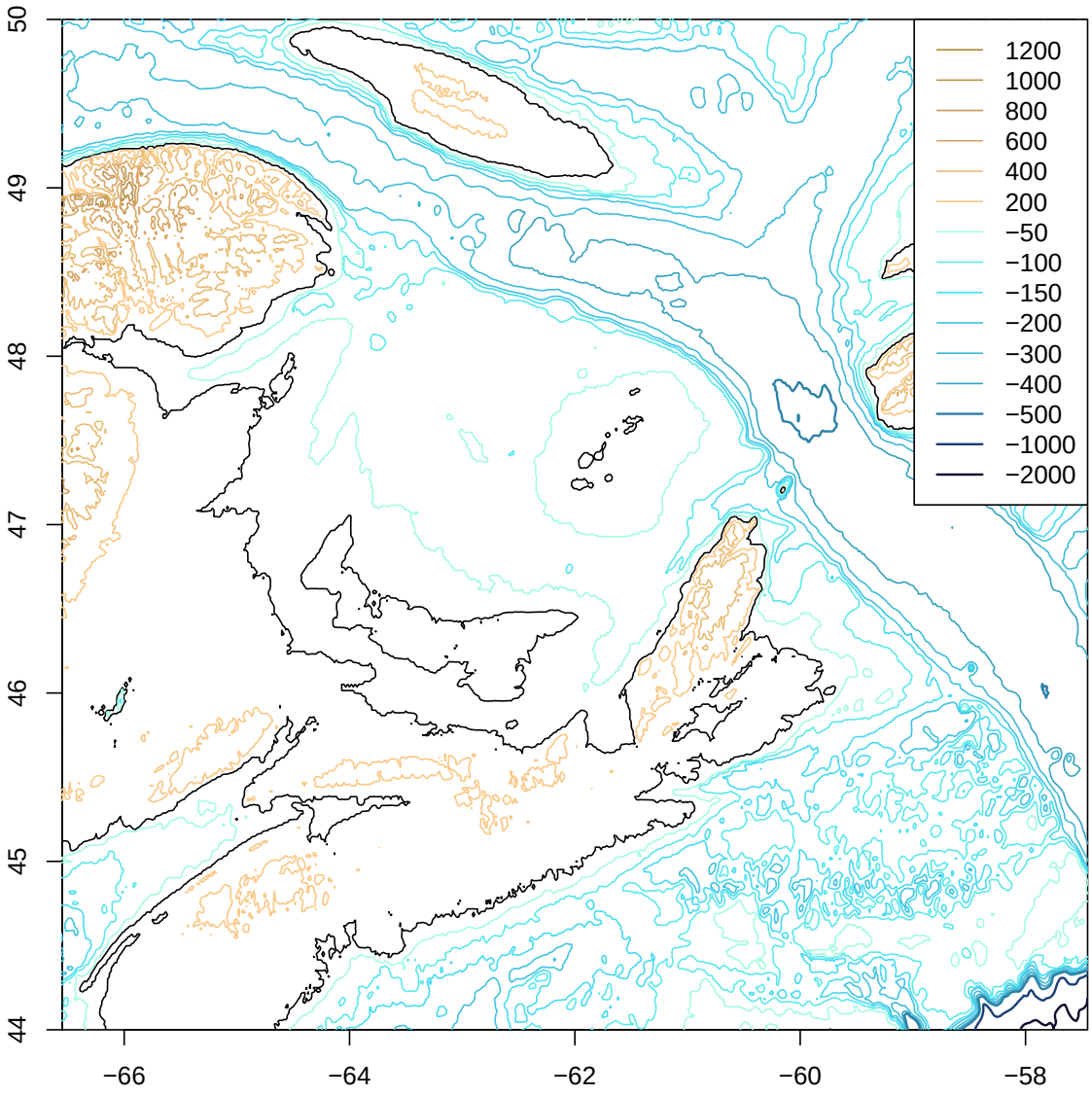

Figure 5: Topography of eastern Canada, centred on Les îles de la Madeleine, north of Prince Edward Island.

6 Sea-level data

6.1 Time-domain analysis

The commands

```
> library(oce)
> data(sealevel.hal)
> plot(sealevel.hal)
```

load and graph a build-in dataset of sea-level timeseries. The result, shown in Figure 6, is a four-panel plot. The top panel is a timeseries view that provides an overview of the entire data set. The second panel is narrowed to the most recent month, which should reveal spring-neap cycles if the tide is mixed. The third panel is a spectrum, with a few tidal constituents indicated. At the bottom is a cumulative spectrum, which makes these narrow-banded constituents quite visible.

Exercise 5. Illustrate Halifax sealevel variations during Hurricane Juan

Exercise 6. Draw a spectrum of sea-level variation, with the M2 tidal component indicated.

6.2 Tidal analysis

In a future version, tidal analysis will be provided, along the lines of the t-tide package in Matlab. A preliminary version of tidal analysis is provided by the `tidem` function provided in this version of the package, but readers are cautioned that the results are certain to change in a future version. (The problems involve phase, and the inference of satellite nodes.)

7 Lobo data

The commands

```
> library(oce)
> data(lobo)
> plot(lobo)
```

produce a plot (Figure 7) of lobo data from the Northwest Arm of Halifax Harbour. Note the relationship between decreasing nutrients and increasing fluorescence, as well as the diurnal signal in the latter.

The reader should note that the `lobo` part of `oce` is somewhat preliminary. In particular, the package requires that certain data columns be present, and in a certain order. Also, the function `read.oce` does not understand `lobo` files. Why these limitations, you ask? Well, the `lobo` code was really only written as an aside, for the author's contribution to a "predict the spring bloom" contest held at Dalhousie University.

Exercise 7. Draw a T-S plot for these data, using a colour coding to indicate time, and using plotting tricks to reduce the obscuring of this time signal.

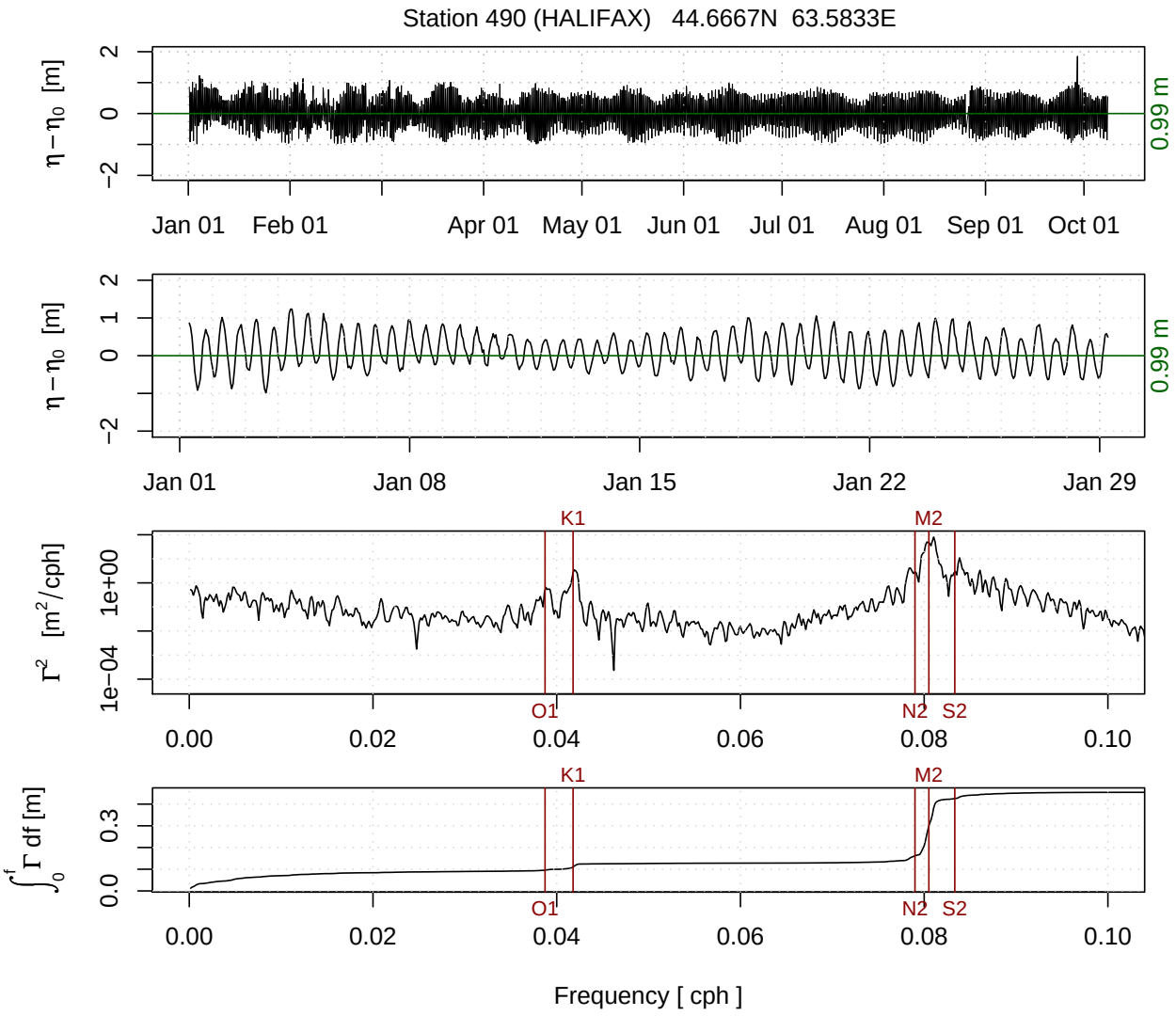

Figure 6: Sea-level timeseries measured in 2003 in Halifax Harbour. (The spike in September is the storm surge associated with Hurricane Juan, regarded by the Canadian Hurricane Centre to be one of the most powerful and damaging hurricanes to ever hit Canada.

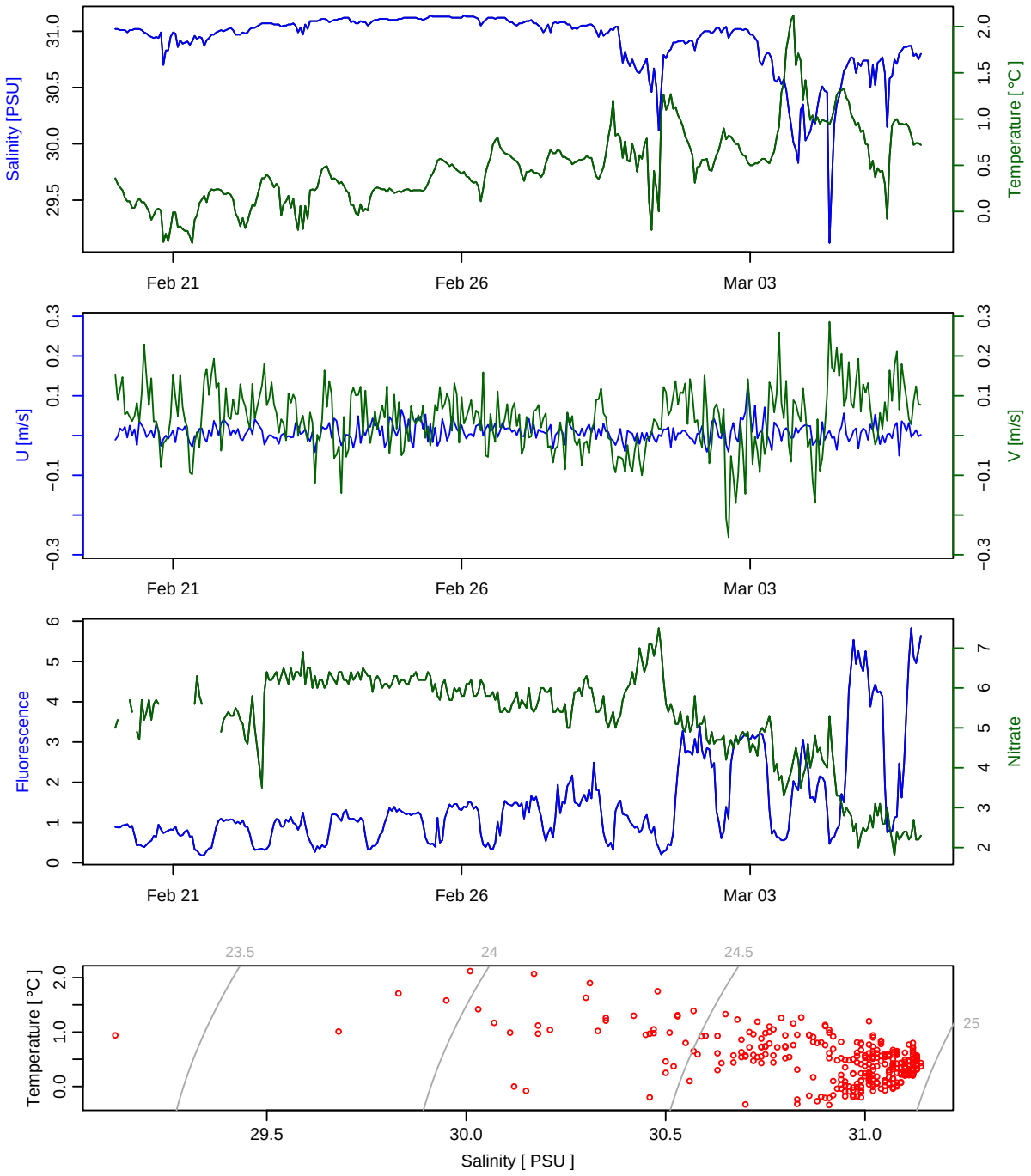

Figure 7: Lobo measurements in Northwest Arm of Halifax Harbour, at the time of the 2007 Spring bloom.

8 The future of oce

The present version of `oce` can only handle data types that the author has been using lately in his research. New data types will be added as the need arises in that work, but the author would also be happy to add other data types that are likely to prove useful to the Oceanographic community. (The data types need not be restricted to Physical Oceanography, but the author will need some help in dealing with other types of data, given his research focus.)

As for algorithms, there are plenty of gaps in `oce`. Dealing with measurements of turbulence is a high priority for the author, for example, and it is also clear that there are some methods of ADP processing that should be provided by the package.

Two principles will guide the addition of data types and functions: (a) the need, as perceived by the author or by other contributors and (b) the ease with which the additions can be made.

9 Development website

The site `http://code.google.com/p/r-oce/` provides a window on the development that goes on between the CRAN releases of the package. Please visit the site to report bugs, to suggest new features, or just to see how `oce` development is coming along.

Oce versions are only made official, i.e. released to the CRAN package system, when they are considered to be of a high quality. However, you may also get the bleeding-edge version from the development website, perhaps to see whether the author has successfully addressed a deficiency that you reported. You will need to have subversion installed on you machine, and then can do e.g.

```
svn checkout http://r-oce.googlecode.com/svn/trunk/ -r 441 r-oce-read-only
```

to retrieve the intermediate version number 441. Naturally, you should rename the directory from `r-oce-read-only` to `oce` before attempting to check, build, or install the package.

Answers to exercises

Exercise 1 – Seawater properties.

```
> library(oce)
> sw.rho(S = 34, t = 10, p = 100)

[1] 1026.624

> sw.theta(S = 34, t = 10, p = 100)

[1] 9.988598

> sw.rho(S = 34, t = sw.theta(S = 34, t = 10, p = 100), p = 0)

[1] 1026.173

> sw.rho(S = 34, t = sw.theta(S = 34, t = 10, p = 100, pref = 200),
+     p = 200)

[1] 1027.074

> plot.TS(as.ctd(c(30, 40), c(-2, 20), rep(0, 2)), grid = TRUE,
+     col = "white")
```

Exercise 2 – Profile plots. Although one may argue as to the limits of the pycnocline, for illustration let us say it is in 5bar to 12dbar range.

```
> library(oce)
> data(ctd)
> pycnocline <- ctd.trim(ctd, "pressure", c(5, 12))
> plot.profile(pycnocline, type = "density+N2")
```

Exercise 3 – TS diagram for section data.

```
> library(oce)
> data(section)
> SS <- TT <- pp <- id <- NULL
> n <- length(section$data$station)
> for (i in 1:n) {
+     stn <- section$data$station[[i]]
+     SS <- c(SS, stn$data$salinity)
+     TT <- c(TT, stn$data$temperature)
+     pp <- c(pp, stn$data$pressure)
+     id <- c(id, rep(i, length(stn$data$pressure)))
+ }
> ctd <- as.ctd(SS, TT, pp)
> plot.TS(ctd, col = hsv(0.7 * id/n), cex = 2, pch = 21)
```

Exercise 4 – Gulf Stream. (Try `?sw.dynamic.height` for hints on smoothing.)

```
> library(oce)
> data(a03)
> Gulf.Stream <- section.subset(a03, 124:102)
> dh <- sw.dynamic.height(Gulf.Stream)
> par(mfrow = c(2, 1))
> plot(dh$distance, dh$height, type = "b", xlab = "", ylab = "Dyn. Height [m]")
> grid()
> f <- coriolis(Gulf.Stream$data$station[[1]]$metadata$latitude)
> g <- gravity(Gulf.Stream$data$station[[1]]$metadata$latitude)
> v <- diff(dh$height)/diff(dh$distance) * g/f/1000
> plot(dh$distance[-1], v, type = "l", col = "blue", xlab = "Distance [km]",
+     ylab = "Velocity [m/s]")
> grid()
> abline(h = 0)
```

Exercise 5 – Halifax sealevel during Hurricane Juan. A web search will tell you that Hurricane Juan hit about midnight, 2003-sep-28. The author can verify that the strongest winds occurred a bit after midnight – that was the time he moved to a room without windows, in fear of flying glass.

```
> library(oce)
> data(sealevel.hal)
> plot(sealevel.hal, focus.time = c("2003-09-23", "2003-10-05"))
> abline(v = as.POSIXct("2003-09-28 23:30:00"), col = "red", lty = "dotted")
> mtext("Hurricane\nJuan", at = as.POSIXct("2003-09-28 23:30:00"),
+     col = "red")
```

Exercise 6 – Sealevel spectrum. Notice the use of `(object)$data$(item)` here. All `oce` objects are lists, and all of them contain a `data` element of a similar form to this.

```
> library(oce)
> data(sealevel.hal)
> spectrum(sealevel.hal$data$eta, spans = c(3, 7))
> abline(v = 1/12.42)
> mtext("M2", at = 1/12.42, side = 3)
```

Exercise 7 – Lobo plot. The resampling with `i` is to avoid obscuring colours by overplotting. Note the use of `as.ctd` to assemble the data into something that `plot.TS` can handle. This is an example of the practicality of `oce`; eventually, `plot.TS` may be altered to take simple columns of data, but for now it seems reasonable to require the user to assemble these data into a CTD object, and to spend development time on something that will pay off better.

```
> library(oce)
> data(lobo)
> i <- sample(length(lobo$data$temperature))
> a <- as.numeric(lobo$data$time[i] - lobo$data$time[1])
> col <- hsv(0.5 * a/max(a), 1, 1)
> plot.TS(as.ctd(lobo$data$salinity[i], lobo$data$temperature[i],
+     0), col = col, pch = 1)
```

Index