rcalcofi: ANALYSIS AND VISUALIZATION OF CALCOFI DATA IN R

EDWARD D. WEBER, SAM MCCLATCHIE
NOAA Fisheries
Southwest Fisheries Science Center
8604 La Jolla Shores Drive
La Jolla, California 92037-1508

ABSTRACT

We developed a package for the R software environment called "**rcalcofi**" designed to facilitate analysis and visualization of data related to the California Cooperative Fisheries Investigations program (CalCOFI) and other similar oceanographic data. The major tasks the package is designed to accomplish are to: convert spatial data between the CalCOFI line/station coordinate system and latitude/longitude or other projections; match nominal spatial locations and nearby points within a given radius; create enhanced spatial plots of multivariable or multivariate data using the lattice graphics system; grid data in the CalCOFI coordinate system; and easily download remotely-sensed data for integration into analyses. The package is freely available, and can be downloaded at ftp://swfscftp.noaa.gov/users/eweber/rcalcofi/.

INTRODUCTION

The R software environment (R Development Core Team 2008) is a tool for conducting statistical computing and graphics that is becoming increasingly popular in the fields of ecology, fisheries science, and oceanography, among others. Its freely available source code and active community of scientists contributing extensions to the base functionality of R make it a flexible tool for conducting oceanographic research. The capability to analyze spatial data in a variety of formats (Bivand et al. 2008) and produce publication-quality graphics in R make it an ideal tool for conducting analyses of data collected as part of the California Cooperative Fisheries Investigations (CalCOFI). Here we describe an R package designed to facilitate visualization and analysis of CalCOFI data and other similar oceanographic data. We created the **rcalcofi** package because we found during the course of our own research that we wanted to be able to perform several recurring tasks more conveniently than was possible using existing functionality in R. The major tasks that the package is designed to accomplish are to: convert spatial data between the CalCOFI line/station coordinate system and latitude/longitude or other projections; match nominal spatial locations and nearby points within a given radius; create enhanced spatial plots of multivariable or multivariate data using the lattice graphics system; grid data in the CalCOFI coordinate system; and easily download remotely-sensed data for integration into analyses.

In this article, we briefly describe the major functionality of the package; however, we do not attempt to document every detail of package use. The standard package documentation contains descriptions of all functions and options in **rcalcofi**, along with examples. Many more examples are contained in the vignette that is included with the package. It can be accessed directly from the "doc" folder of the package (rcalcofi.pdf), or by using the vignette command in R after the package has been installed.

```
> vignette("rcalcofi")
```

The standard documentation and vignette should serve as the primary references for the package. The source code for the package, and binary packages for the Apple OS X®, and Microsoft Windows® operating systems, can be downloaded at ftp://swfscftp.noaa.gov/users/eweber/rcalcofi. Users who are completely new to R may wish to access the Comprehensive R Archive Network (CRAN), www.cran.R-project.org, to view introductory material, documentation, and references pertaining to the general use of R before working with **rcalcofi**. Throughout this article the Courier font is used to indicate R package names, command names, or specific R code.

PACKAGE REQUIREMENTS

The general strategy in developing the package was to use as much of the capability of existing R packages as possible, thereby preserving conventions that are familiar to many users and avoiding duplication of effort. Thus, the package has a relatively long list of dependencies. They are the **fields** (Furrer et al. 2009), **grid** (R Development Core Team 2008), **lattice** (Sarkar 2009), **maptools** (Lewin-Koh et al. 2009), **methods** (R Development Core Team 2008), **rgdal** (Keitt et al. 2008), **sp** (Pebesma and Bivand 2005), and **spatstat** (Baddeley and Turner 2005) packages available from CRAN repositories. The **rgdal** package further requires the installation of the Geospatial Data Abstraction

Library and PROJ4 library for projection and transformation of spatial data. We also recommend installing the gshhs shoreline files (Wessels and Smith 1996) so that higher resolution shoreline images can be plotted. The package particularly relies on **lattice** graphics and spatial classes provided by the **sp** package. In addition to the standard documentation, **lattice** and **sp** are described in greater detail by Sarkar (2008) and Bivand et al. (2008), respectively.

MAJOR FUNCTIONALITY

Converting To and From the CalCOFI Line/Station Coordinate System

Converting data expressed in CalCOFI line and station coordinates to latitude and longitude is a common task when analyzing or plotting CalCOFI data. The **rcalcofi** package contains the function **station.to.latlon** to achieve this, and the reverse function **latlon.to.station**. These functions accept a matrix or two-column data frame of coordinates and return a matrix of converted coordinates using the algorithm described by Eber and Hewitt (1979). For example, a matrix of CalCOFI coordinates named **myCalCOFIdata** could be converted to a matrix of longitudes and latitudes as follows:

```
> station.to.latlon(myCalCOFIdata)
```

We note that repeated conversions between coordinate systems on the same data should be avoided because a small error is introduced with each conversion (cf., Eber and Hewitt 1979). That is, raw location data should probably be retained, and projection done once for each analysis or plot.

A more flexible way to analyze spatial data is to use the spatial classes provided by the **sp** package. The **sp** package provides methods for projecting, gridding, overlaying, and converting spatial data (possibly with attributes) to and from data frames and matrices. We have adapted these methods in **rcalcofi** to treat the CalCOFI coordinate system as if it were any other map projection (e.g., Mercator, stereographic, etc.). Thus, the usual spatial methods can be applied to data associated with CalCOFI lines and stations, and they can be interconverted, imported, and exported using standard methods. The recommended means of working with CalCOFI spatial data is to convert them to an object of a spatial class (e.g., a **SpatialPointsDataFrame**), and then manipulate them using **sp** methods. In the **rcalcofi** package, the projection "**+proj=calcofi**" is used to get the means the CalCOFI line/station coordinate system. The hypothetical **myCalCOFI** data matrix could be converted to **SpatialPoints** and then transformed as follows:

```
> spdata <- SpatialPoints(data.frame
(myCalCOFIdata), CRS("+proj=calcofi"))
> spdata <- spTransform(spdata, CRS
("+proj=longlat"))
```

Converting to a spatial data type will also allow the data to be plotted using the high-level lattice plotting functions in **rcalcofi**, which we describe below.

Matching Locations

Another common problem when working with CalCOFI data is that measurements are recorded at locations near a nominal location (e.g., a station) and must be matched to the nominal record for further analysis. The function **determine.station** matches points that are within a specified radius of another set of nominal points. For example, a data frame called **stations** was created containing the locations of two near-shore core CalCOFI stations (line 76.7, stations 49 and 51; tab. 1). We matched four hypothetical points in a data frame called **samples** (tab. 2) to the nearest station within the default 4 km radius using the following call:

```
> determine.station(samples$lon,
samples$lat, stations$lon, stations$lat,
row.names(stations))
```

The result is a vector of row names from the station data:

```
> [1] 1 NA 2 2
```

The vector indicates the first sample point matches to station 1, and the last two points match to station 2 (fig. 1). The second result is an **NA**, indicating the second sample point was not located within 4 km of either

TABLE 1
The stations data frame containing the longitude and latitude of two core CalCOFI stations, line 76.7, stations 49 and 51. The four hypothetical sample points listed in Table 2 were matched to these data using the determine.station function, as illustrated in Figure 1.

	lon	lat
1	239.222	35.088
2	239.082	35.022

TABLE 2
The samples data frame containing the longitude and latitude of four hypothetical points that are near the CalCOFI stations listed in Table 1. These data were matched where they were within 4 km of a station using the determine.station function, as illustrated in Figure 1.

	lon	lat
1	239.205	35.080
2	239.203	35.124
3	239.051	35.020
4	239.082	35.043

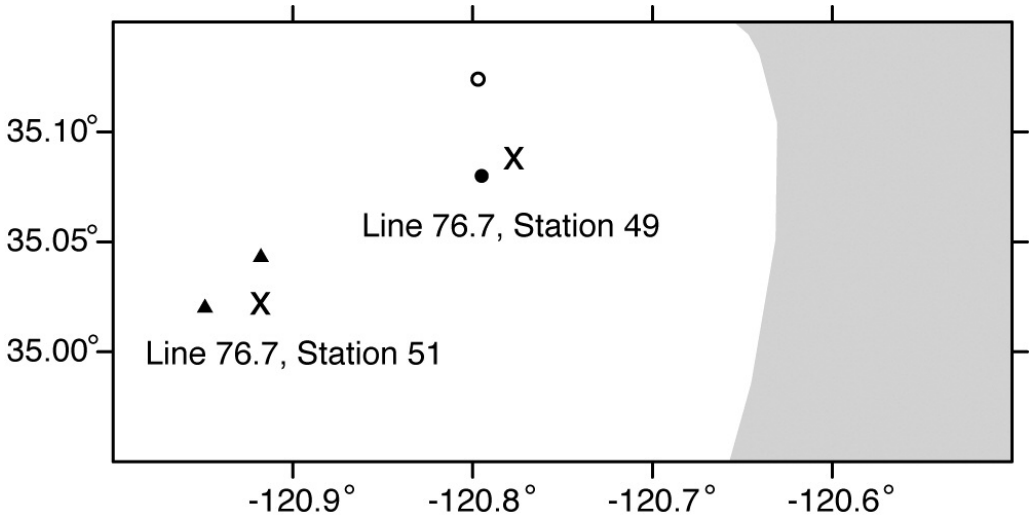

Figure 1. Four hypothetical sample locations (tab. 1) were matched to two core CalCOFI stations, line 76.7, stations 49 and 51 (tab. 2; designated by the "X" symbol), using the **`determine.station`** function if they were within a 4-km radius of a point. Closed circle indicates a sample location matched to the first station in Table 1 (line 76.7 station 49), and triangles indicate sample locations matched to the second station in Table 1 (line 76.7 station 51). The open circle indicates a location greater than 4 km from either station, so an NA was returned by the function. The gray area represents the shoreline.

of the stations. The resulting vector could be assigned as a column in the **`samples`** data frame and used to merge data sets using the standard R methods.

Enhanced Plotting of Spatial Data in Lattice

It is often useful to visualize multivariable or multivariate data by constructing graphics with a panel for each level of one or more variables. For example, we frequently plot species abundance separately for each year, or at different levels of one or more environmental variables. The R **`lattice`** graphics system was designed for plotting these types of data. The **`rcalcofi`** package adds several high-level lattice functions for plotting spatial data using lattice graphics. These functions have several convenient features for plotting spatial data, including formula methods similar to standard lattice functions, automatic conversion and plotting of spatial data from different projections (including CalCOFI), and default panel functions that were designed to be integrated into more complex graphics. All of the high-level plotting functions in **`rcalcofi`** accept shoreline maps that can be created easily using the **`get.map`** function. Nearly all of the standard lattice functions will work as expected with these functions, thereby allowing users who are familiar with lattice graphics to customize plots even further.

The new high-level plotting functions included in **`rcalcofi`** are **`spxyplot`**, **`splevelplot`**, **`spcontourplot`**, **`spgridplot`**, and **`spstickplot`**. The **`spxyplot`**, **`splevelplot`**, and **`spcontourplot`** functions are analogous to the standard lattice **`xyplot`**, **`levelplot`**, and **`contourplot`** functions, except that they accept spatial data types instead of data frames, and plot in any desired projection. The **`splevelplot`** and **`spcontourplot`** functions actually call the same underlying function with different options, similar to **`levelplot`** and **`contourplot`**. A few other adjustments have been made to **`splevelplot`** to accommodate data projected from a different coordinate system. If data that were regularly spaced in the original coordinate system are plotted in a different projection, the function will plot polygons with corners at the appropriate projected coordinates rather than rectangles. This is useful for creating a **`levelplot`** on the CalCOFI grid but plotting it in longitude and latitude, for example. A second option for plotting irregularly-spaced data in **`splevelplot`** is to use the **`krig`** option. This will call the **`Krig`** function (with user arguments, if specified) from the **`fields`** package to create an interpolated surface, and then plot the surface as a standard **`levelplot`** in the specified coordinate system.

The **`spgridplot`** function plots gridded spatial information as a standard grid if it is plotted in its native projection or as a grid with curved lines that reflect distortion if it has been projected to a new coordinate system. It accepts gridded (i.e., regularly spaced) data as objects of gridded spatial classes from the **`sp`** package (**`SpatialGrid`**, **`SpatialGridDataFrame`**, **`SpatialPixels`**, or **`SpatialPixelsDataFrame`**).

The **`spstickplot`** function plots data associated with spatial locations as solid bars or filled frames similar to "thermometers" in the standard graphics **`symbols`** function. It is useful for creating three-dimensional bar plots.

High-level functions in **`rcalcofi`** can be called with only one or two arguments for basic use. For example, a **`SpatialPoints`** object named **`mySpatialPoints`**

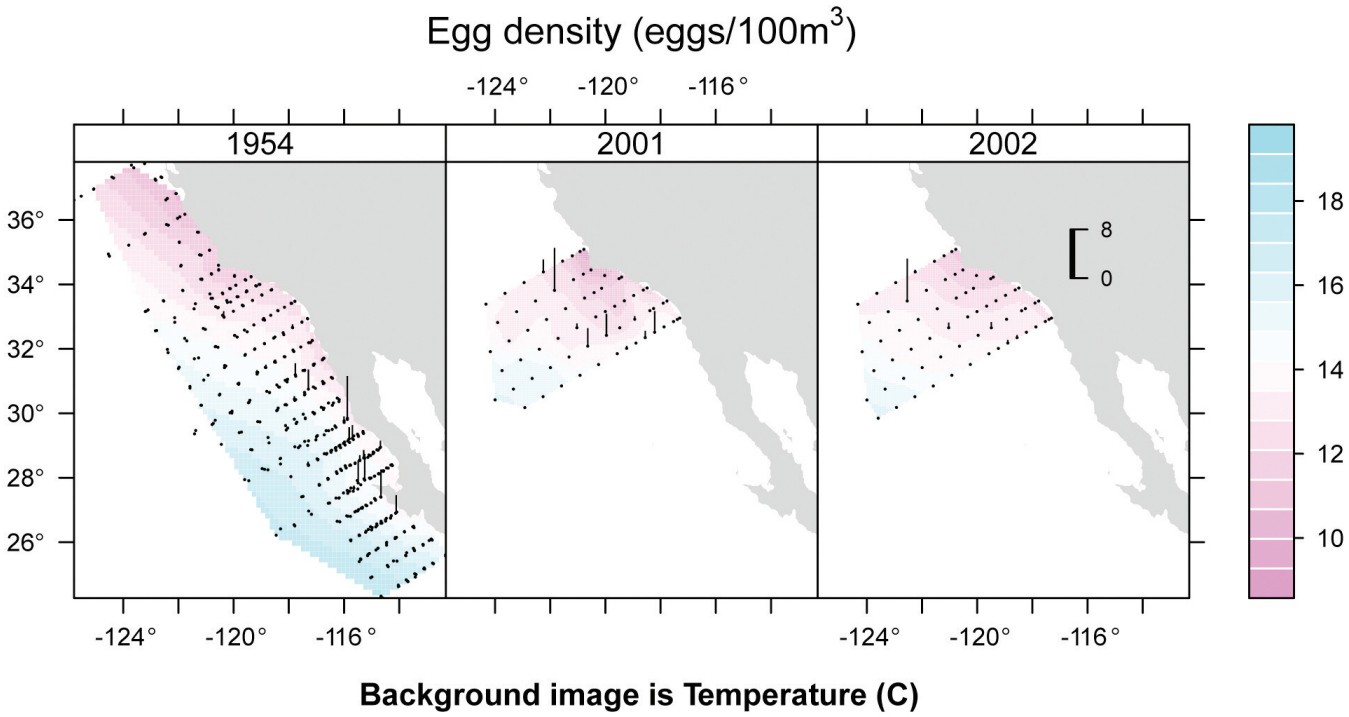

Figure 2. Density of sardine (*Sardinops sagax*) eggs (bars) sampled at CalCOFI stations in spring of 1954, 2001, and 2002 plotted over an interpolated surface of mean water temperature in the upper 50 m, as sampled at CalCOFI stations (color image). Black dots indicate sample locations. The plot was created using **splevelplot** as the base high-level plotting function. A custom panel function was used to over-plot points using **panel.spxyplot**, and bars using **panel.spstickplot**. The figure is based on an example data set named **calcofiDat** that is included in the package.

can be plotted in its native coordinate system as follows:

```
> spxyplot(mySpatialPoints)
```

We provide somewhat more complex examples to illustrate multiple features in a single graphic. These examples also demonstrate how custom panel functions can be used to combine graph types. Following the standard lattice convention, each high-level plotting function in **rcalcofi** has a default panel function that performs the actual plotting of symbols. For example, the default panel function for **spxyplot** is **panel.spxyplot**. More complex graphs can be created by replacing the default panel function with a custom panel function that combines several default types. Figure 2 illustrates the use of **splevelplot** as the base high-level plotting function. A custom panel function was used to over-plot points using **panel.spxyplot**, and bars using **panel.spstickplot**. The figure is based on an example data set called **calcofiDat** that is included in the package. The data describe CalCOFI samples collected during spring in 1954, 2001, and 2002. The color image depicts mean water temperature in the upper 50 m, as interpolated from data collected at each station using the **krig** option in **splevelplot**. Sample locations are indicated by black. Bars indicate densities of sardine eggs captured at each station.

Several other features of high-level plotting functions in **rcalcofi** are also illustrated in Figure 2. The shoreline polygons (technically a **gList** of polygons; cf., Murrell 2006) were passed to the function for plotting in each panel. The shoreline plotted was from the sample file named **shoreline** included with the package but could have been created using the **get.map** function. The figure was conditioned on years using the formula method (e.g., **temperature ~ coordinates | year**). For convenience, high-level functions in **rcalcofi** that accept a formula method can accept the word "coordinates" in place of the coordinate names (e.g., instead of **temperature ~ longitude * latitude | year**). The **splevelplot** function also calculated the correct aspect ratio and placed degree symbols on axis labels automatically. These could have been overridden by providing an **aspect** argument or a custom axis.

The figure was created from the **calcofiDat** data frame as follows. First, a year variable was added to the data frame based on the **datetime** column. Then, the data frame was converted to a **SpatialPoints DataFrame**.

```
> data(calcofiDat)
> ccdat <- calcofiDat
> ccdat$year <- substring
(ccdat$datetime, 1, 4)
> ccdat <- SpatialPointsDataFrame
(SpatialPoints(ccdat[c("longitude",
"latitude")], proj4string = CRS
("+proj=longlat")), ccdat)
```

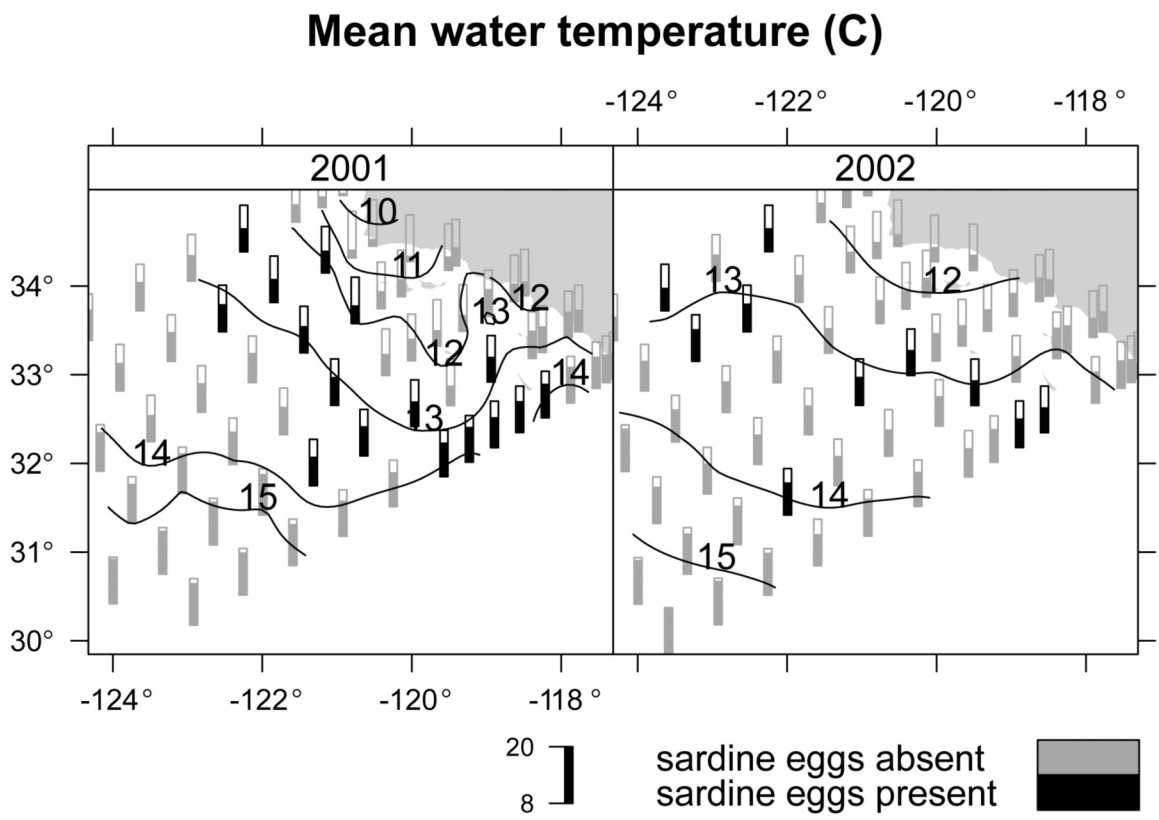

Figure 3. Mean water temperature in the upper 50 m, as measured at CalCOFI stations during spring 2001 and 2002. Data are plotted in two ways. First, filled bars indicate mean temperature at each station similarly to "thermometers" symbols in the standard **symbols** plotting function. An interpolated temperature surface is also over-plotted as a contour plot. The graph was created using **spstickplot** as the high-level plotting function with a custom panel function that called **panel.spcontourplot** to plot contours. A groups argument was used to plot stations where sardine eggs were present in black and stations where sardine eggs were absent in gray. Data are in the **calcofiDat** example data frame that is included in the package.

Then the plot was created as follows:

```
> splevelplot(temperature ~ coordinates
| year, ccdat, krig = TRUE, as.table =
TRUE, xlab = "", ylab = "", main =
expression(paste('Egg density
(eggs/100', m^3, ')', sep = '')),
sub = 'Background image is Temperature
(C)', strip = strip.custom(bg =
"transparent"), layout = c(3, 1),
panel = function(...)
{
  panel.splevelplot(...)
  panel.map(map = shoreline,
proj4string = "+proj=longlat")
  subscripts <- list(...)$subscripts
  panel.spxyplot(ccdat[subscripts, ],
proj4string = "+proj=longlat", col =
"black", pch = 20, cex = 0.1)
  panel.spstickplot(ccdat[subscripts,
"eggs"], proj4string = "+proj=longlat",
filledBars = FALSE, col = "black",
width = unit(0.01, "npc"))
  })
```

The key for the stickplot was added using the function **spstickplotKey**:

```
> spstickplotKey(width = unit(0.01,
"npc"), minval = min(ccdat$eggs,
na.rm = TRUE), maxval = max(ccdat$eggs,
na.rm = TRUE), col = 'black', border =
'black', vp = viewport(x = unit(0.8,
"npc"), y = unit(0.6, "npc")),
draw = TRUE)
```

We used a subset of the same data, years 2001 and 2002, to plot mean water temperature in the upper 50 m (fig. 3). The data were plotted in two ways. The **spstickplot** function was used to indicate measurements at each station, with framed bars indicating the minimum and maximum values of bars. An interpolated

surface of temperature was over-plotted as contours on each panel using **panel.spcontourplot**. The plot was conditioned on years, and grouped by the logical variable **presence** to indicate stations where sardine eggs were captured in black, and stations where no sardine eggs were captured in gray. The data were prepared and the basic graph was plotted using the following commands:

```
> ccdat2 <- ccdat[ccdat$year %in%
c('2001', '2002'), ]
> ccdat$presence <- ifelse(ccdat$eggs >
0, "present", "absent")
> spstickplot(temperature ~ coordinates
| year, ccdat2, as.table = TRUE,
groups = presence, xlab = "",
ylab = "", width = unit(0.015, "npc"),
col = c('darkgray', 'black'), border =
c('darkgray', 'black'), strip =
strip.custom(bg = "transparent"),
layout = c(2, 1), main = 'Mean water
temperature (C)',
panel = function(...)
{
  subscripts <- list(...)$subscripts
  panel.map(shoreline, '+proj=longlat')
  panel.spstickplot(...)
  panel.spcontourplot(ccdat2[subscripts,
'temperature'], krig = TRUE,
proj4string = '+proj=longlat',
cex = 0.5)
})
```

Two additional commands were used to generate the group key and the bar key:

```
> draw.key(list(text=list(lab =
c('sardine eggs absent', 'sardine eggs
present')), rect = list(col =
c('darkgray', 'black'))),
vp = viewport(x = unit(0.75, "npc"),
y = unit(0.22, "npc")), draw = TRUE)
> spstickplotKey(width = unit(0.015,
"npc"), minval = min(ccdat$temperature,
na.rm = TRUE), maxval =
max(ccdat$temperature, na.rm = TRUE),
col = 'black', border = 'black',
vp = viewport(x = unit(0.5, "npc"),
y = unit(0.22, "npc")), main = TRUE,
draw = TRUE)
```

Gridding Data in the CalCOFI Coordinate System

Gridding data can be accomplished using the overlay method exactly as described in the **sp** package. Basic gridding of data can be accomplished in **rcalcofi** even more conveniently using the **pixelize.spatialdat** function. This calls the **overlay** function to match spatial data to grid cells, and then creates a new grid with average values for the data in each grid cell (or another specified function such as the median). For example, a grid in the form of a **SpatialPixels** object from line 30 to line 120, with default cell sizes 3-1/3 lines by 10 stations, was created using the **calcofi.grid** function:

```
> ccgrd <- calcofi.grid(lineRange =
c(30, 120))
```

We overlayed the grid onto the example data set named **TPH_ssta_8day_20060416** that is included in the package. The data set consists of remotely-sensed sea surface temperature measured at 5.5 km resolution by the Pathfinder mission. The data are expressed in longitude and latitude for geographic range 28° to 42°N and -135° to -112°W for the eight-day period centered on 16 April 2006. These data were downloaded from the NOAA Coastwatch server using the **get.dap.data** function described below. The data were converted to a **SpatialPointsDataFrame** named **llimagedat** using the standard **sp** method:

```
> data(TPH_ssta_8day_20060416)
> llimagedat <- TPH_ssta_8day_20060416
> llimagedat <- SpatialPointsDataFrame
(cords = llimagedat[c("x", "y")], data
= llimagedat["z"],proj4string =
CRS("+proj=longlat"))
```

A new grid named **ccimage**, containing mean sea surface temperatures for each cell of the CalCOFI grid, was created using the **pixelize.spatialdat** function. The function handled the different projections of the two data sets automatically:

```
> ccimage <- pixelize.spatialdat
(llimagedat, ccgrd)
```

The gridded data are plotted in Figure 4. The plot was further customized before plotting with a different shoreline map. We created **shoreline2** using the **get.map** function. The map used a high-resolution gshhs shoreline file similar to the **shoreline** data, but light yellow as a fill color:

```
> gshhsPath <-
'/Users/eweber/calcofi/gshhs/gshhs_h.b'
> shoreline2 <- get.map(gshhsPath =
gshhsPath, col = 'lightyellow', border
= 'transparent', xlim = c(-136, -114),
ylim = c(22, 45))
```

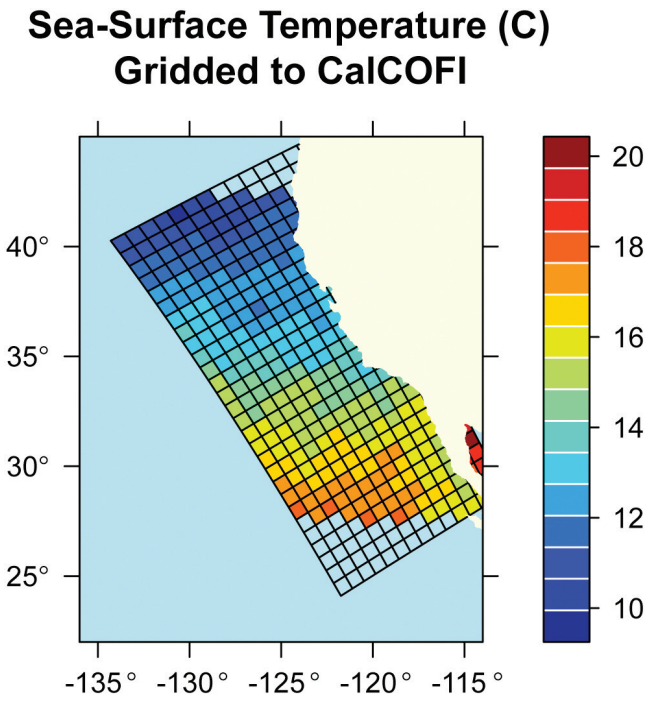

Figure 4. Mean sea surface temperature for the eight-day period centered on 16 April 2006. Original data were Pathfinder measurements expressed in longitude and latitude at 5.5 km resolution, and contained in the example data set named **TPH_ssta_8day_20060416**. The data were gridded to cell sizes of 3-1/3 lines by 10 stations in the CalCOFI coordinate system using the **pixelize.spatialdat** function. Data were plotted using the **spimageplot** function with a custom panel that included the **panel.spgridplot** function to create the black grid that is superimposed on the image.

The graph was plotted using the following commands:

```
> proj4string <- '+proj=longlat'
> trellis.par.set(axis.text = list
(cex = 0.9))
> splevelplot(ccimage, proj4string =
proj4string, xlim = c(-136, -114),
ylim = c(22, 45), proj4string.limits =
proj4string, xlab = '', ylab = '',
main = 'Sea-Surface Temperature
(C)\nGridded to CalCOFI', col.regions =
tim.colors(100),
panel = function(...)
{
  panel.fill(col = 'lightblue')
  panel.splevelplot(...)
  panel.spgridplot(ccimage,
proj4string)
  panel.map(shoreline2, proj4string)
})
```

Note that the grid was expressed in the CalCOFI coordinate system, but plotted in longitude and latitude by providing the **proj4string** argument **+proj=longlat**. The **panel.fill** function was used to create the light blue background behind the image and the shoreline. A different color palette for the image was also specified using the **col.regions** argument.

Downloading Remotely Sensed Data

The **rcalcofi** package includes functions to download freely available remotely sensed data from the NOAA Coastwatch server (http://coastwatch.pfel.noaa.gov/). The package includes a summary table of available sensors and data types called **dapDat.Rda**. The **get.dap.info** function can be used to print these data or construct a call to the **get.dap.data** function, which is used to download data. For example, the following code will download SEAWIFS monthly chlorophyll data for the given date and range, which is summarized as row 63 in **dapDat.Rda**:

```
> dapcall <- get.dap.info(63, c(28,
29), c(-135, -134), "2006-04-16",
saveFile = FALSE)
> dapcall
> eval(dapcall)
```

The **get.dap.data** function can also be called directly:

```
> fileType <-"xyz"
> latRange <- c(28, 29)
> lonRange <- c(-135, -134)
> satellite <- "TPH"
> variable <- "ssta"
> dte <- "2006-04-16"
> timePeriod <- "8day"
> get.dap.data(satellite, variable,
timePeriod, dte, lonRange, latRange)
```

The function will automatically save the file in the default directory using a file name that indicates the sensor, date, and time period downloaded (e.g., the **TPH_ssta_8day_20060416** data frame was downloaded using this function), unless the **saveFile** argument is set to FALSE. The **check.dap.dates** function queries the NOAA Coastwatch servers to find available dates for data from a given sensor. It may be called manually, and is called automatically if a **get.dap.data** call fails. These functions are adapted (with permission) from the **xtractomatic** R program provided by Dave Foley and Cindy Bessey, NOAA Southwest Fisheries Science Center, Environmental Research Division, Pacific Grove, California. The **xtractomatic** program provides an alternative method of retrieving satellite data. It can be downloaded at http://coastwatch.pfel.noaa.gov/coastwatch/CWBrowserWW360.jsp?get.

Miscellaneous Functions

Several other simple miscellaneous functions that are included in **rcalcofi** are likely to be useful to other

researchers working with CalCOFI data. The most frequently used of these are probably **calculate.mld**, **calculate.distance.offshore**, **cols.to.zmatrix**, and **zmatrix.to.cols**. The **calculate.mld** function returns an estimate of mixed-layer depth, given density, and water depth, following the procedure described by Kara et al. (2000). The **calculate.distance.offshore** function calculates the nearest distance from shore (as represented by a **SpatialPolygons** object) for each point in a matrix of locations. The **cols.to.zmatrix**, and **zmatrix.to.cols** functions convert spatial data between column (matrix or data frame) formats and the list format used by functions such as **image** and **contour**.

DISCUSSION

The **rcalcofi** package has reached a level of development where it may be useful to other researchers, despite relatively limited testing as part of our own research. We anticipate that additional refinement will be needed and welcome any bug reports as they are discovered. Although the package was developed with analysis of CalCOFI data in mind, much of the functionality is likely to be useful for other types of data. The spatial plotting functions may be particularly useful for other areas of fisheries and oceanographic research.

Some of the functionality of the package can be accomplished more computationally efficiently using GIS software or other programs. Although it is often convenient to conduct analyses using pure R solutions rather than working interactively between R and other programs, alternative approaches may be preferable when working with very large data sets. For such cases, the Spatial task view on the CRAN website provides an up-to-date list of applications that work well with R.

We consider **rcalcofi** to be primarily a package of convenience functions rather than one that introduces major new functionality. The authors of the core software and packages upon which **rcalcofi** relies have done most of the difficult programming. Users should not overlook existing functionality outside of the package when analyzing CalCOFI data. For example, standard graphics or standard **lattice** graphics may be as simple to use as our plotting functions in some cases when all data are in the same coordinate system.

ACKNOWLEDGMENTS

We thank E. Archer for reviewing the manuscript. This work was supported in part by a grant from the U.S. Integrated Ocean Observing System program. We thank C. Oliver of NOAA and J. Everett of Ocean Associates, Inc. for administering funding.

LITERATURE CITED

Baddeley, A., and R. Turner. 2005. Spatstat: an R package for analyzing spatial point patterns. J. Statist. Software. 12:1–42.

Bivand, R. S., Pebesma, E. J., and Gómez-Rubio, V. 2008. Applied spatial data analysis with R. Springer, New York, 374 pp.

Eber, L. E., and R. P. Hewitt. 1979. Conversion algorithms for the CALCOFI station grid. Calif. Coop. Oceanic Fish. Invest. Rep. 20:135–137.

Furrer, R. D. Nychka, and S. Sain. 2009. fields: Tools for spatial data. R package version 5.02. http://www.image.ucar.edu/Software/Fields.

Kara, A. B., P. A. Rochford, and H. E. Hurlburt. 2000. An optimal definition for ocean mixed layer depth. J. Geophys. Res. Oceans 105: 16803–16821.

Keitt, T. H., R. S. Bivand, E. J. Pebesma, and B. Rowlingson. 2008. rgdal: Bindings for the Geospatial Data Abstraction Library. R package version 0.5-30. http://www.gdal.org, http://rgdal.sourceforge.net/, http://sourceforge.net/projects/rgdal/.

Lewin-Koh, N. J., R. S. Bivand, E. J. Pebesma, E. Archer, A. Baddeley, H. Bibiko, S. Dray, D. Forrest, P. Giraudoux, D. Golicher, V. Gómez Rubio, P. Hausmann, T. Jagger, S. P. Luque, D. MacQueen, A. Niccolai, and T. Short. 2009. maptools: Tools for reading and handling spatial objects. R package version 0.7–21.

Murrell, P. 2006. R graphics. Taylor and Francis, Boca Raton, Florida, 301 pp.

Pebesma, E. J., and R. S. Bivand. 2005. Classes and methods for spatial data in R. R News 5. http://cran.r-project.org/doc/Rnews.

R Development Core Team. 2008. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Sarkar, D. 2008. Lattice: multivariate data visualization with R. Springer, New York. http://lmdvr.r- forge.r- project.org/.

Sarkar, D. 2009. lattice: Lattice Graphics. R package version 0.17-20.

Wessel, P., and W. H. F. Smith. 1996. A global self-consistent, hierarchical, high-resolution shoreline database. J. Geophys. Rsch. 101:8741-8743. http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html.