# PBS Software:
# Maps, Spatial Analysis, and Other Utilities

Jon T. Schnute, Nicholas M. Boers, and Rowan Haigh

Fisheries and Oceans Canada
Science Branch, Pacific Region
Pacific Biological Station
3190 Hammond Bay Road
Nanaimo, British Columbia
V9T 6N7

2003

**Canadian Technical Report of
Fisheries and Aquatic Sciences 2496**

**Errata for *PBS Software: Maps, Spatial Analysis, and Other Utilities*, first printing**
(last updated January 9, 2004)

**Page 17,** fourth line, replaced

`(qcdexe.cpp, qcdexe.h)` → `(qcdexe.cpp)`

**Page 39,** before the 13th line from the bottom, inserted

If you have MiKTeX installed, you may need to edit its configuration file (e.g., `C:\Program Files\texmf\miktex\config\miktex.ini`). In it, find your version of LaTeX. To determine your version, issue the command `latex` in a DOS window. The output should resemble

```
This is e-TeX, Version 3.141592-2.1 (MiKTeX 2.4)
```

where in this case, the version of LaTeX is "e-TeX". At the end of the "Input Dirs" line, append a semicolon, the path to `Rd.sty`, and two forward slashes (e.g., `;C:\PROGRA~1\R\rw\share\texmf//`).

If, while running `Rcmd CHECK`, you receive errors involving a file named `MkRules` (e.g., `C:\Program Files\r\rw\src\gnuwin32\MkRules`), you may need to edit it. Prefix a hash (#) to the following lines (104 to 106 and 121 to 122):

```
.C.d:
     @echo "making $@ from $<"
     @$(CXX) $(DEPARG) $(CXXFLAGS) $($*-CXXFLAGS) $< -o $@

.C.o:
     $(CXX) $(CXXFLAGS) $($*-CXXFLAGS) -c $< -o $@
```

**Page 43,** lines 7 and 10, replaced

`bin` → `cmd`

**Page 47,** Table A1, indented rows 3 and 7 one additional space.

**Page 47,** Table A1, indented rows 4 to 6 and 8 to 11 two additional spaces.

**Page 47,** Table A1, row 19 column 2, replaced

Datasets → Data sets

**Page 47,** Table A1, below row 17, inserted

| | |
|---|---|
| `\AnalyticalTools` | Analytical tools for Windows |
| `\R` | R 1.8.1 |
| `\Scilab` | Scilab 2.7.2 |

Canadian Technical Report of

Fisheries and Aquatic Sciences 2496

2003

# PBS Software: Maps, Spatial Analysis, and Other Utilities

by

Jon T. Schnute, Nicholas M. Boers, and Rowan Haigh

Fisheries and Oceans Canada

Science Branch, Pacific Region

Pacific Biological Station

3190 Hammond Bay Road

Nanaimo, British Columbia

V9T 6N7

CANADA

Cat. No. Fs97-6/2496E ISSN 0706-6457

9 8 7 6 5 4 3 2 1 (First printing – October 31, 2003)

Correct citation for this publication:

Schnute, J.T., N.M. Boers, and R. Haigh. 2003. PBS software: maps, spatial analysis, and other utilities. Can. Tech. Rep. Fish. Aquat. Sci. 2496: viii + 82 p.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## ABSTRACT

Schnute, J.T., N.M. Boers, and R. Haigh. 2003. PBS software: maps, spatial analysis, and other utilities. Can. Tech. Rep. Fish. Aquat. Sci. 2496: viii + 82 p.

This report describes software written to facilitate the compilation and analysis of fishery data, particularly data referenced by spatial coordinates. Our research stems from experiences with information on Canada’s Pacific groundfish fisheries compiled at the Pacific Biological Station (PBS). Despite its origins in fishery data analysis, our software has broad applicability. The library *PBS Mapping* extends the languages R and S-PLUS to include two-dimensional plotting features similar to those commonly available in a Geographic Information System (GIS). Embedded C code speeds algorithms from computational geometry, such as finding polygons that contain specified point events or converting between longitude-latitude and Universal Transverse Mercator (UTM) coordinates. We also present a number of convenient utilities for the Microsoft Windows operating systems, including commands that support computational geometry outside the framework of R or S-PLUS. Tools to construct most of our software come freely from the Internet, as documented here in a guide to the packages available. Furthermore, we provide quick tutorials that address key technical issues relevant to our work, such as embedding C code into an R package and writing documentation that meets the R standard. Our results, which depend significantly on the work of students, illustrate the convergence of goals between academic training and applied research.

## RÉSUMÉ

Schnute, J.T., N.M. Boers, and R. Haigh. 2003. PBS software: maps, spatial analysis, and other utilities. Can. Tech. Rep. Fish. Aquat. Sci. 2496: viii + 82 p.

Ce rapport présente un logiciel conçu pour faciliter la compilation et l’analyse de données sur la pêche, particulièrement de données géoréférencées. Notre recherche découle de notre expérience de la compilation de données sur la pêche du poisson de fond dans le Pacifique à la Station biologique du Pacifique (SBP). Bien qu’il ait été conçu pour analyser des données sur la pêche, notre logiciel est d’application très générale. La cartothèque logicielle *PBS Mapping* permet de développer les langages R et S-PLUS afin qu’ils comprennent des capacités de traçage bidimensionnel semblables à celles disponibles couramment dans un système d’information géographique (SIG). Le langage C permet d’accélérer les algorithmes géométriques, comme ceux qui permettent de rechercher des polygones comportant des événements ponctuels précis ou de convertir des coordonnées géographiques en coordonnées de Mercator transverse universelle. Nous présentons également un certain nombre d’applications pratiques pour les systèmes d’exploitation Windows de Microsoft, y compris des commandes de géométrie algorithmique à l’extérieur du cadre des langages R et S-PLUS. Les outils utilisés pour mettre au point la plupart de nos logiciels sont disponibles gratuitement sur Internet, tel que souligné dans un guide des progiciels disponibles présenté dans ce document. De plus, nous offrons de courts didacticiels qui traitent des principales questions techniques liées à notre travail, comme l’intégration de code C dans un progiciel en langage R et la rédaction de documents qui satisfont à la norme du langage R. Nos résultats, qui dépendent grandement du travail d’étudiants, illustrent la convergence des objectifs de la formation académique et de la recherche appliquée.

## PREFACE

During the last several years, I’ve had the pleasure of directing work by computer science students from various local universities. My research as a mathematician in fish stock assessment requires an extensive software toolkit, including statistical languages, compilers, and operating system utilities. It helps greatly to have bright, adaptive students who can learn new languages quickly, investigate software possibilities, answer technical questions, and design programs that assist scientific analysis. Let me begin by acknowledging contributions from the following students:

- Robert Swan (University of Victoria), 1996;
- Mike Jensen (Malaspina University-College and Simon Fraser University), 1997 and 1999;
- Chris Grandin (Malaspina University-College), 2000 and 2001;
- Nick Henderson (Malaspina University-College), 2002;
- Nick Boers (Malaspina University-College), 2003.

Starting in 1998, I began a formal connection with the Computing Science Department at Malaspina University-College (MUC). My discussions with faculty members, particularly Dr. Peter Walsh and Dr. Jim Uhl, highlighted the convergence of goals between academic training and scientific research. Projects designed for fish stock assessment give students an opportunity to further their computer science careers while producing useful software. Both MUC and the Pacific Biological Station (PBS), where I work, are located in Nanaimo, British Columbia, Canada. This happy juxtaposition makes it easy to engage students in the exchange of ideas between academia and applied research. For example, Jim Uhl participated directly in Nick Boers’ PBS work term during the summer of 2003. Nick had completed a course in computer graphics taught by Jim in the fall of 2002. Algorithms in the textbook (Foley et al. 1996) proved invaluable for writing software to produce maps of the British Columbia coast with related fishery information.

Quantitative fishery science requires a strong connection between theory and practice. In his book on computing theory, Michael Sipser (1997, p. xii) tells students that:

> “. . . theory is good for you because studying it expands your mind. Computer technology changes quickly. Specific technical knowledge, though useful today, becomes outdated in just a few years. Consider instead the abilities to think, to express yourself clearly and precisely, to solve problems, and to know when you haven't solved a problem. These abilities have lasting value. Studying theory trains you in these areas.”

While dealing with the issues addressed here, I found myself asking simple questions that have numerically interesting answers. How do you locate fishing events within management areas or other polygons? How should regional boundaries on maps be clipped to lie within a smaller rectangle? I soon realised that I had touched upon the emerging field of computational geometry, where people have devised clever and efficient algorithms for addressing such questions.

Remarkably effective software can now be obtained freely from the Internet. I’m particularly fond of R, a version of the powerful statistical language S (and later S-PLUS) devised by Becker et al. (1988). Although written originally for Unix, R has also been implemented for Microsoft's Windows operating systems. The web site http://cran.r-project.org/ describes R as GNU S, "a freely available language and environment for statistical computing and graphics". The GNU project (http://www.gnu.org/), where the recursive acronym GNU means "GNU's Not Unix", offers a wealth of free software including compilers for C/C++, Fortran, and Pascal. Code can be written in these compiled languages to speed computations that would otherwise run more slowly in R or S. Nick Boers has used such linkages intelligently to bring fast computational geometry into our map drawing package for R/S.

The Internet also makes various scripting languages freely available, including Perl (http://www.perl.com/) and Python (http://www.python.org/). Personally, I prefer the consistent mathematical design of Python, but many important tasks have been written in Perl. For example, R uses extensive Perl scripts to build a complete library package from files containing source code, data, and documentation. Similarly, Nick Boers used Perl to convert high-resolution shoreline data (http://www.ngdc.noaa.gov/mgg/shorelines/gshhs.html) into the format used here.

Work described in this report began in 1996, when I wanted to develop fishery databases that would facilitate stock assessments and give perspective to the information available (Schnute et al. 1996). Some of the tools described here, notably a set of convenient macros for Microsoft Access, address questions raised at that time. Since then, our databases have improved substantially, particularly for Pacific groundfish. These include detailed spatial information on the catch of every species, recorded for every tow by observers aboard trawl vessels. Polygons that define management regions and coastal boundaries have also become important components of the data. Statistical analyses require graphical tools to portray information on coastal maps, as with the software for R/S presented here.

The collective work of several students and various PBS staff members has followed a path from designing databases to producing visual images on maps. Various other products appeared along the way, including

- handy operating system tools,
- investigations into various options for free software, and
- resolution of technical questions raised by these tasks.

This report covers rather heterogeneous material, in an attempt to document results from all this work. Readers can take from it whatever appears useful, most notably the PBS Mapping software package for R/S. I've played mostly a design role in developing these tools, trusting the students to deal with implementation. It's been a highly collaborative process, often with heated debates about how things should work. Occasionally, I've taken the first author's prerogative of resolving the debate by expressing my own view.

Jon Schnute

# 1. INTRODUCTION

This report describes software written to facilitate the compilation and analysis of fishery data, particularly data referenced by spatial coordinates. Our work developed from experiences constructing databases that capture information from Canada's Pacific groundfish fisheries. Fishing events take place across a broad range of coastal waters and result in the capture of many species. Initially, we focused on issues related to database design and development, as described in previous reports by Schnute et al. (1996), Haigh and Schnute (1999), Rutherford (1999), Schnute et al. (2001, Section 2 and Appendix A), and Sinclair and Olsen (2002). Analyses of these databases shifted our attention to the problem of portraying and understanding such complex information. Maps with statistical information proved especially useful, and we found ourselves facing questions commonly addressed by Geographic Information Systems (GIS).

Commercial GIS packages can be expensive, with an additional requirement for specialised training. Because analysts who deal with Pacific groundfish data usually have experience using the statistical languages R (available freely on the Internet) or S-PLUS (available commercially), we began by writing functions in this context to produce the maps required. As described in the Preface to this report, students played an important role in these developments. This year we completely redesigned and rewrote the software package, now called PBS Mapping. Along the way, we devised other pieces of software and investigated various technical issues. This report compiles a somewhat diverse body of work by various students since 1996.

Section 2 covers the mapping software itself, including a description of the public domain Global Self-consistent, Hierarchical, High-resolution Shoreline (GSHHS) database used here. We also discuss the Universal Transverse Mercator (UTM) projection that gives a particularly accurate flat projection of the earth's surface. Our software supports conversions between longitude-latitude and UTM coordinates.

Section 3 documents a number of convenient utilities, both for Microsoft Access and the Windows operating systems. Where possible, we have used free software to construct these utilities, as discussed more completely in Section 4 and Appendix B. Section 5 provides quick tutorials on key technical issues relevant to our work. Readers have access to our software and documentation from a CD (Appendix A), including an Adobe Acrobat PDF file with this report. Our figures, generated from the mapping software, appear best in colour. We recommend a colour copy printed from the PDF file, if possible. Appendices C and D provide a complete technical manual for the mapping library.

We anticipate that our software will change for the better, due to bug fixes and other improvements. This report documents version 1.0. We may produce revisions as future versions of the software become available.

## 2. MAPS AND SPATIAL ANALYSIS

Niklaus Wirth, the author of Pascal and Modula-2, summarises the essence of software design in the title of his book (Wirth 1975): *Algorithms + Data Structures = Programs*. Our software package PBS Mapping begins with data structures that embody two essential concepts. First, polygons define boundaries, such as shorelines and fishery management areas. Second, fishing events occur at specific locations defined by two coordinates, such as longitude and latitude. The languages R and S conveniently support such structures through the concept of a *data frame*, essentially a database table in which rows and columns define records and fields, respectively. Objects in R/S can also have *attributes* that assign properties, such as the projection used in defining a coordinate system.

### 2.1. Data Structures for Maps

#### <u>PolySet</u>

In our software, a *PolySet* data frame defines a collection of polygons, based on four or five numerical fields:

- `PID` – the primary identification number for a polygon;
- `SID` (optional) – the secondary identification number for a polygon;
- `POS` – the position number associated with a vertex;
- `X` – the horizontal coordinate at a vertex;
- `Y` – the vertical coordinate at a vertex.

The simplest PolySet has no `SID`, and each `PID` corresponds to a different polygon. By analogy with a child's "follow the dots" game, the `POS` field enumerates the vertices to be connected by straight lines. Coordinates (`X`, `Y`) specify the location of each vertex. Thus, in familiar mathematical notation, a polygon consists of *n* points $(x_i, y_i)$ with $i = 1, \ldots, n$, where $i$ corresponds to the `POS` index. A PolySet has two potential interpretations. The first associates a line segment with each successive pair of points from 1 to *n*, giving a *polyline* (in GIS terminology) composed of the sequential line segments. The second includes a final line segment joining points *n* and 1, thus closing the polygon.

The secondary ID field allows us to define regions as composites of polygons. From this point of view, each primary ID identifies a collection of polygons distinguished by secondary IDs. For example, a single management area (`PID`) might consist of two fishing areas, each defined by a unique `SID`. A secondary polygon can also correspond to an inner boundary, like the hole in a doughnut. We adopt the convention that `POS` goes from 1 to *n* along an outer boundary, but from *n* to 1 along an inner boundary. In GIS software, these normally correspond to clockwise and counter-clockwise directions, respectively, so that a clockwise direction always corresponds to `POS` moving from 1 to *n*. Our software uses ordering of the `POS` index to distinguish between inner and outer boundaries, but makes no explicit requirement about the directional sense (clockwise or counter-clockwise).

A PolySet with a secondary ID field must have indices `SID` that appear in ascending order within each `PID`. Furthermore, inner boundaries must follow the outer boundary that encloses them. The `POS` field for each polygon (`PID`, `SID`) must similarly appear in strictly increasing or decreasing order, for outer and inner boundaries respectively, but need not take sequential integer values. This allows the insertion of a new point, such as point 3.5 between points 3 and 4, or the deletion of a point without problems. We include a function `fixPOS` to renumber points as sequential integers.

A PolySet can have a `projection` attribute, which may be missing, that specifies a map projection. In the current version of PBS Mapping, `projection` can have character values `"LL"` or `"UTM"`, referring to "Longitude-Latitude" and "Universal Transverse Mercator". We explain these projections more completely below. If `projection` is numeric, it specifies the aspect ratio *r*, the number of *x* units per *y* unit. Thus, *r* units of *x* on the graph occupy the same distance as one unit of *y*. Another attribute `zone` specifies the UTM zone (if `projection="UTM"`) or the preferred zone for conversion from Longitude-Latitude (if `projection="LL"`).

## PolyData

We define *PolyData* as a data frame with a first column named `PID` and (optionally) a second column named `SID`. Unlike a PolySet, where each polygon has many records corresponding to the vertices, a PolyData object must have only one record for each `PID` or each (`PID`, `SID`) combination. Conceptually, this object associates data with polygons, where the data correspond to additional fields in the data frame. The R/S language conveniently allows fields in a data frame to have different types, such as numeric and character. For example, PolyData with the fields (`PID`, `PName`) might be used to assign names to a set of primary polygons.

Our software particularly uses PolyData to set various plotting characteristics. Consistent with parameters to the R/S functions `lines` and `polygon`, column names can specify graphical properties:

- `lty` – line type in drawing the border and/or shading lines;
- `col` – line or fill colour;
- `border` – border colour;
- `density` – density of shading lines;
- `angle` – angle of shading lines.

When drawing polylines, only `lty` and `col` have meaning.

## EventData

We define *EventData* as a data frame with at least three fields named (`EID`, `X`, `Y`). Conceptually, an EventData object describes events that take place at specific points (`X`, `Y`) in two-dimensional space. Additional fields specify measurements associated with these events. For example, in a fishery context EventData could describe fishing events associated with trawl tows, based on the fields:

- `EID` – fishing event (tow) identification number;
- `X`, `Y` – fishing location;
- `Duration` – length of time for the tow;
- `Depth` – average depth of the tow;
- `Catch` – biomass captured.

Like PolyData, EventData can have attributes `projection` and `zone`, which may be missing.

**LocationSet**

A PolySet defines regional boundaries for drawing a map, and EventData determine event points on the map. Which events occur in which regions? In the algorithm section below, we discuss a computational solution to this problem. The output lies in a *LocationSet*, a data frame with three or four columns (`EID`, `PID`, `SID`, `Bdry`), where `SID` may be missing. One row in a LocationSet means that the event `EID` occurs in the polygon (`PID`, `SID`). The boundary (`Bdry`) variable specifies whether (`Bdry=T`) or not (`Bdry=F`) the event lies on the polygon boundary. If `SID` refers to an inner polygon boundary, then `EID` occurs in (`PID`, `SID`) only if `Bdry=T`. An event may occur in multiple polygons. Thus, the same `EID` can occur more than once in a LocationSet.

## 2.2. Map Projections

The Flat Earth Society (http://www.flat-earth.org/) asserts that

- "the Earth is flat and has five sides,"
- "all places in the Universe named Springfield are merely links in higher-dimensional space to one place," and
- "all assertions are true in some sense, false in some sense, meaningless in some sense, true and false in some sense, true and meaningless in some sense, false and meaningless in some sense, and true false and meaningless in some sense."

For more information, including a proof that $5 = 6$, see the URL mentioned above. Their point of view could potentially simplify our mapping software. Unfortunately, a non-flat earth requires some kind of projection to represent it on a flat map.

The simplest projection associates each point on the earth's surface with a longitude $x$ ($-360° \le x \le 360°$) and latitude $y$ ($-90° \le y \le 90°$), where $x = 0°$ on the Greenwich prime meridian. The chosen range of $x$ depends on the region of interest, where negative longitudes refer to meridians west of the prime meridian. When plotted on a rectangular grid with equal distances for each degree of longitude and latitude, this projection exaggerates the size of objects near the earth's poles, as illustrated in Figure 1. For points near the latitude $y$, a more realistic map uses the aspect ratio

$$r = \frac{1}{\cos y}, \tag{2.1}$$

where $r$ degrees of longitude $x$ should occupy the same distance as 1 degree of latitude $y$.

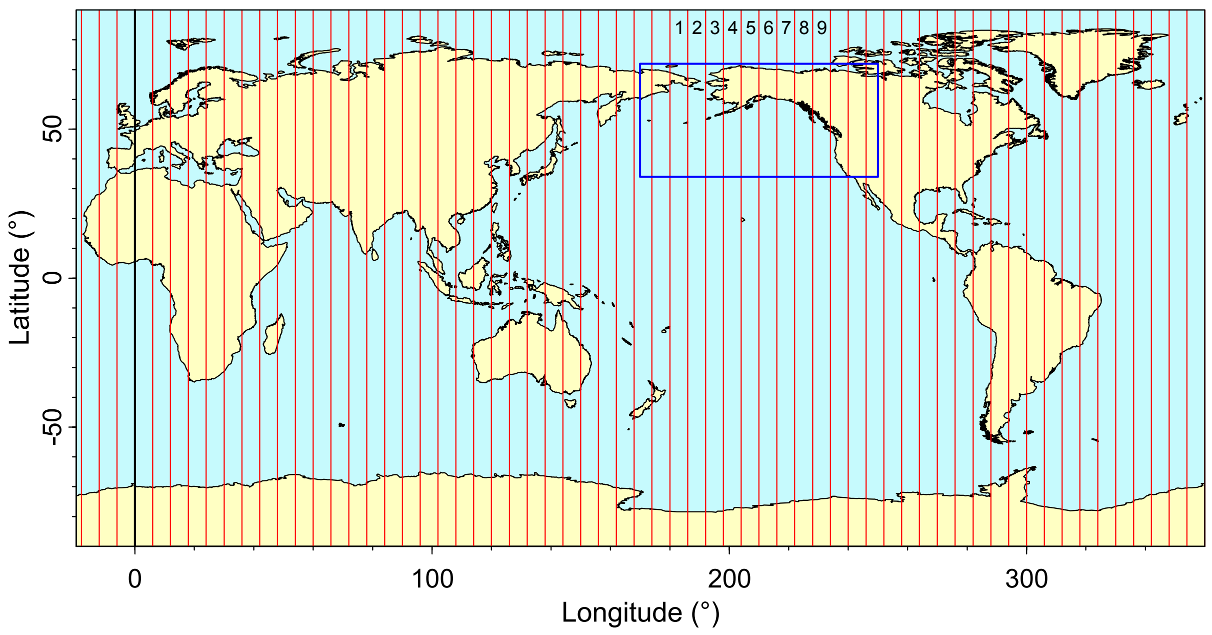


**Figure 1.** Map of the world projected in longitude-latitude coordinates. This image, based on our PolySet `worldLL`, uses the longitude range $-20° \le x \le 360°$ to produce a convenient cut in the eastern Atlantic Ocean. Red vertical lines show boundaries for the 60 Universal Transverse Mercator (UTM) zones, with explicit labels for zones 1 to 9. A black line indicates the prime meridian $(x = 0°)$. Our PolySet `nepacLL` lies within the clipping boundary shown as a blue rectangle.

The Universal Transverse Mercator (UTM) projection gives a more realistic portrayal of the earth's surface within 60 standardized longitude zones. Each zone spans $6°$, and zone *i* includes points with longitude *x* in the range

(2.2) $$(-186 + 6i)° < x \le (-180 + 6i)° .$$ [UTM zone *i*]

The mid-longitude in (2.2)

(2.3) $$x_i = (-183 + 6i)° .$$ [Central meridian, zone *i*]

defines the *central meridian* of zone *i*. In particular, zone 9 has central meridian $-129°$ and covers the range

(2.3) $$-132° < x \le -126° .$$ [UTM zone 9]

Canada's Pacific coast lies in zones 8-10 (Figure 2), and the projection to zone 9 gives a reasonably accurate map for fisheries in this region.

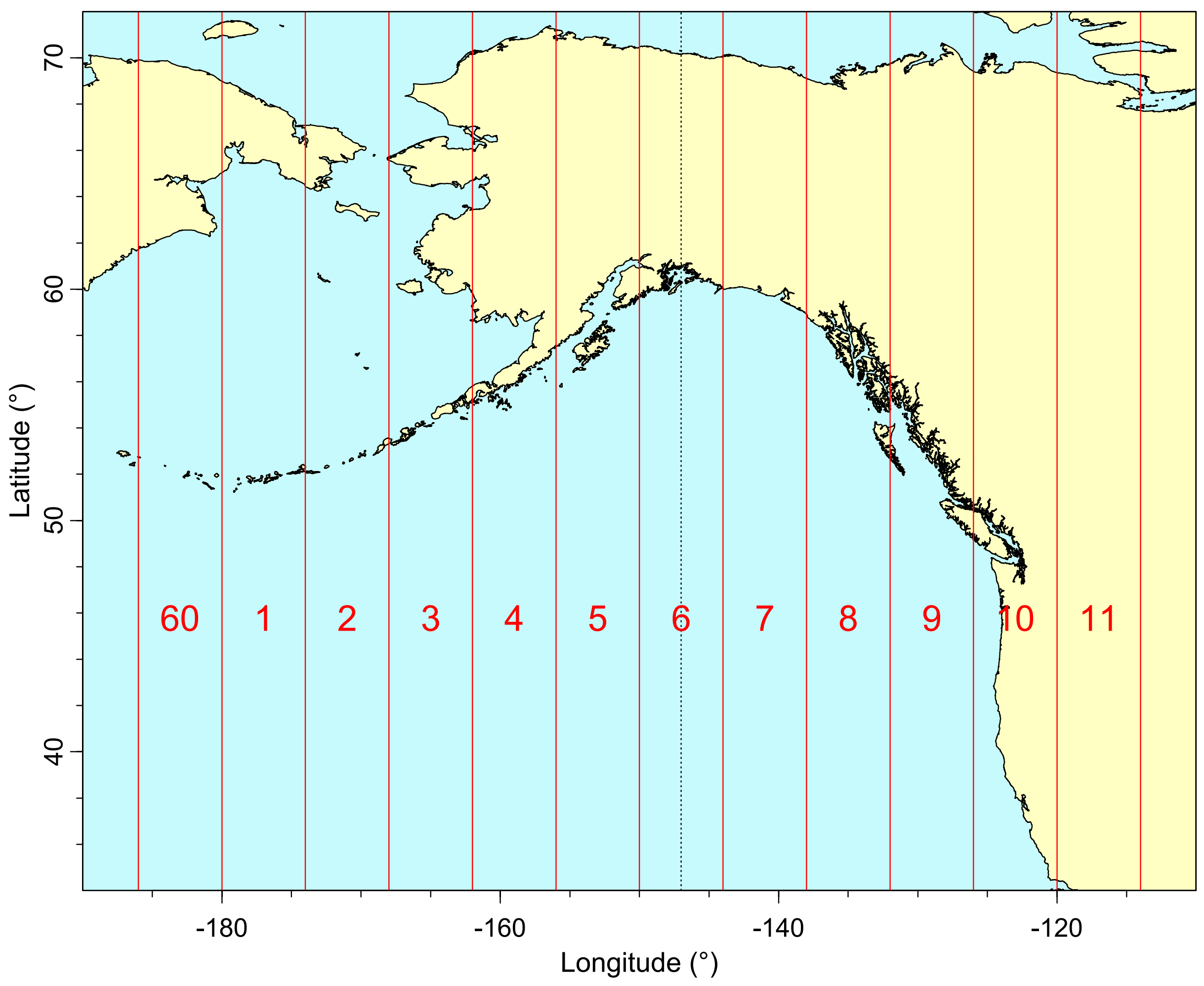


**Figure 2.** Shoreline data in longitude-latitude coordinates for the northeastern Pacific Ocean, as captured in our PolySet `nepacLL`. Vertical red lines display UTM boundaries for zones $60, 1, 2, \ldots, 11$. A vertical dotted line indicates the central meridian of zone 6, near the centre of this figure.

Visually, UTM zones look like sections of orange peel cut from top to bottom. Each relatively narrow section can be flattened without too much distortion to give coordinates $(X, Y)$ measured as actual distances, as illustrated by zone 6 in Figure 3. Complex formulas, compiled in detail by the UK Ordnance Survey (Anonymous 1998), allow conversion between two projections: the UTM *easting-northing* coordinates $(X, Y)$ and the usual longitude-latitude coordinates $(x, y)$. These take account of the earth's ellipsoidal shape, with a wider diameter at the equator than the poles. The UTM projection scales distances exactly along two great circles: the equator and the central meridian, which act as $X$ and $Y$ axes, respectively. Along the equator, $Y = 0$ km by definition; elsewhere, $Y$ indicates the distance north of the equator, where a negative value corresponds to a southward displacement. The central meridian is assigned the standard easting $X = 500$ km, rather than the usual $X = 0$ km. This ensures that $X > 0$ km throughout the zone. In effect, the difference $X - 500$ km represents the distance east of the central meridian, where a negative distance corresponds to a westward displacement. These interpretations are exact along the equator and central meridian, but approximate elsewhere.

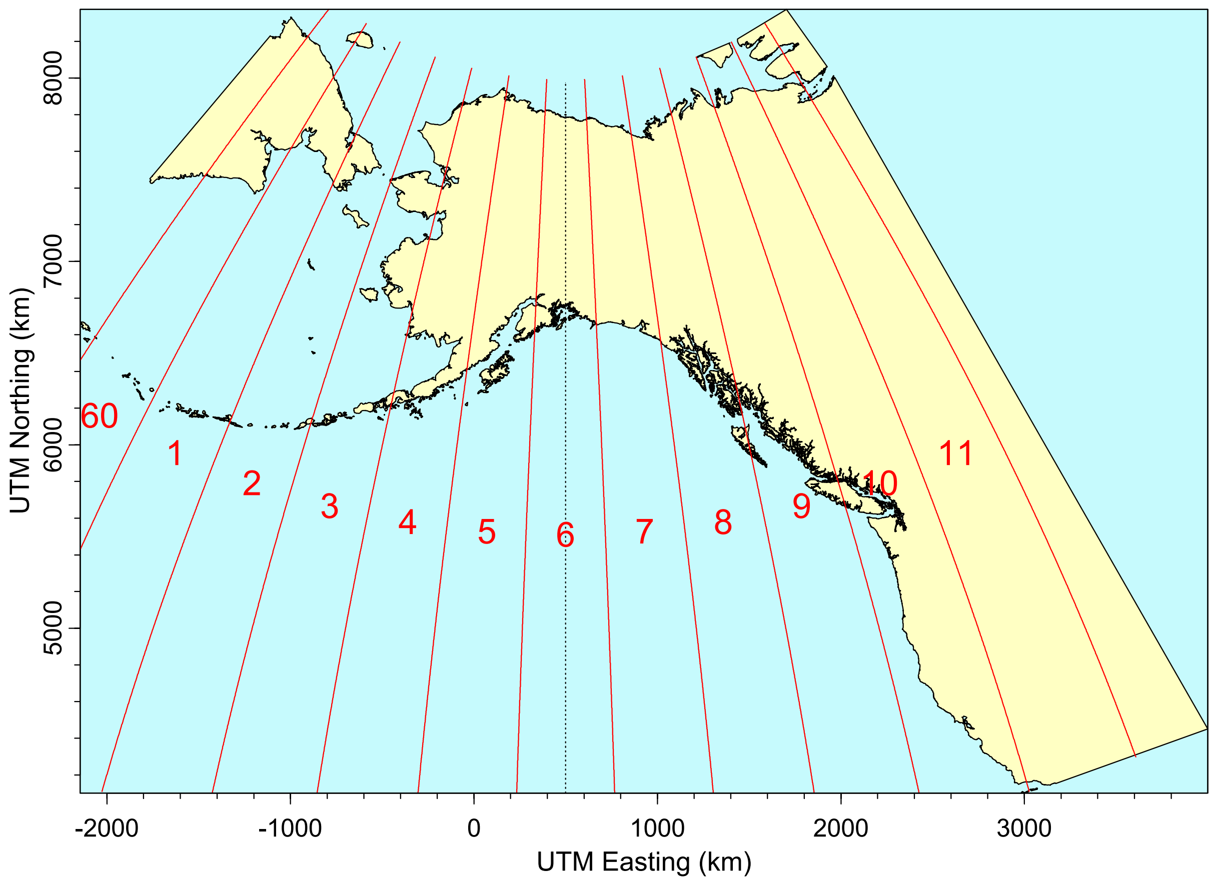


**Figure 3.** Shoreline data for the northeastern Pacific Ocean, projected in UTM coordinates (zone 6) from our PolySet `nepacLL`. Vertical red lines show UTM zone boundaries. The central axis of zone 6 (vertical dotted line at $x = 500$ km) corresponds to the central meridian shown in Figure 2.

## 2.3. PBS Mapping Algorithms and Functions

Our software produces maps from the data structures defined in Section 2.1. Following typical design concepts in R/S, we use functions to implement algorithms and produce images, where function arguments often have specified default values. Appendix D gives detailed technical descriptions of all our functions and other software components. In particular, the globally defined constant `LANG` automatically adapts our code to R (`LANG="R"`) or S-PLUS (`LANG="S"`). Just set `LANG` to an appropriate value for the language environment before running our code.

In the R/S language, high level commands (like `plot`) create new graphs; lower level commands (like `points` and `lines`) add features to an existing graph. Similarly, we provide functions (`plotLines`, `plotPolys`, `plotMap`) that create graphs and others (`addLines`, `addPolys`) that add graphical features. All these functions draw objects defined by a PolySet. Both `plotLines` and `addLines` treat the data as polylines, with no connection between the last and first points. By contrast, `plotPolys`, `addPolys`, and `plotMap` regard the data as polygons, where a final line segment connects the last point back to the first. The functions `plotPolys` and `plotMap` behave similarly, except that `plotMap` guarantees the correct aspect ratio, as defined by the `projection` attribute of the PolySet. Table 1 summarises the behaviour of our principal graphics commands. A user concerned with drawing maps, where the correct aspect ratio plays a key role, would initiate a graph with the `plotMap` function.

**Table 1.** Behaviour of the principal graphics functions in the PBS Mapping software package.

| Function | Creates Graph | Closes Polygons | Sets Aspect Ratio |
|---|---|---|---|
| `plotLines` | Yes | No | No |
| `plotPolys` | Yes | Yes | No |
| `plotMap` | Yes | Yes | Yes |
| `addLines` | No | No | No |
| `addPolys` | No | Yes | No |

Our graphics functions support the usual R/S graphical parameters, including:

- `xlim` and `ylim` to specify horizontal and vertical coordinate ranges;
- `plt` to define the plot region relative to the figure region;
- `tck` to determine tick mark lengths;
- `lty`, `col`, `border`, `density`, `angle` to adjust properties of lines and polygons.

We introduce additional parameters that give finer control over the appearance of tick marks in high level commands. Each can have length 1 or 2, where a single value pertains to both axes and two values give separate parameters for the horizontal and vertical axes. These include:

- `tckMinor`, counterpart of `tck` that sets a different length for minor tick marks;
- `tckLab`, with Boolean values that determine whether or not to include numeric tick labels.

Some of our functions construct a PolySet or evaluate its properties. For example:

- `makeGrid` constructs a rectangular grid of polygons;
- `calcArea` computes polygon areas by the formula

$$A = \left| x_n y_1 - x_1 y_n + \sum_{i=1}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \right|,$$

for the area $A$ of a polygon with vertices $(x_i, y_i)$, $i = 1, \ldots n$ (Rokne 1996).

Other functions perform numerical tasks that alter a defined PolySet. For example:

- `fixPOS` renumbers the `POS` index as a sequence of ascending or descending integers;
- `fixBound` attempts to fix the polygon boundary within the rectangle defined by the ranges of `X` and `Y`, where points near the bounding rectangular are snapped onto it;

- `closePolys` adds corners from the bounding rectangle, if needed, to close polylines into polygons;
- `clipLines` clips polylines within a specified rectangle, possibly smaller than the bounding rectangle;
- `clipPolys` similarly clips polygons within a specified rectangle;
- `convUL` converts between UTM and longitude-latitude coordinates.

Clipping can be computationally intensive. We use the Sutherland-Hodgman algorithm (Foley et al. 1996, p. 124-127).

Our function `findPolys` solves the "points-in-polygons" problem. Given a set of points (EventData) and a collection of polygons (a PolySet), which points lie in which polygons? Several algorithms have been proposed for this problem, including:

- **The angle summation (or winding number) test.** Sum the angles swept by a ray from the trial point to sequential vertices of the polygon. For a point outside the polygon, the angles sum to 0 because the ray sweeps back and forth, returning to the starting point. For an inside point, the ray traces a full circle, and the angles do not sum to zero.
- **The crossings test.** Draw a ray from the trial point in a fixed direction (e.g., upward). If the ray crosses an even number of polygon edges, the point must be outside. For an inside point, the number of crossings must be odd.

We use the crossings test, which performs faster than angle summation (Hains 1994, p. 26-27), due to the large number of trigonometric function calls needed in the angular calculations.

After finding the polygons that contain various events, an analyst often wants to compute statistics associated with the events within each polygon. For example, in a fishery context, what is the total catch from all fishing events within each management region? Our function `combineEvents` supports such calculations. The function `makeProps` can then associate statistical values with polygon properties, such as the colour intended for graphing.

Finally, the R/S `locator` function enables users to define (*x*,*y*) coordinates from mouse clicks on an existing graph. Similarly, our functions `locatePolys` and `locateEvents` construct PolySet and EventData objects from points identified visually with the mouse.

## 2.4. Shoreline Data

To portray fishery data along Canada's Pacific coast, we need a PolySet that defines the relevant shoreline. We began with a polyline of the British Columbia coast, obtained digitally from a marine map. To convert this object to a meaningful closed polygon, we devised the functions `fixBounds` and `closePolys`. Satellite imagery and other sources, however, make our initial coastline obsolete. For example, Wessel and Smith (1996) have used information from the public domain to assemble a Global Self-consistent, Hierarchical, High-resolution Shoreline (GSHHS) database for the entire planet. They make this available via the Internet as binary files in five different resolutions: full, high, intermediate, low, and crude. They also supply software as C source code for

- converting the data to an ASCII (plain text) format (`gshhs.c`);
- thinning the data by reducing the number of points sensibly (`gshhs_dp.c`).

Their thinning software uses an algorithm devised by Douglas and Peucker (1973), whose initials `dp` appear in the file name. We compiled both programs with a free GNU compiler, as described in Appendix B.

**Table 2.** PolySets derived from the full resolution GSHHS database.

| PolySet | Thinning | Longitude | Latitude | Vertices | Polygons |
|---|---|---|---|---|---|
| `nepacLL`* | 0.2 km | $-190^\circ \le x \le -110^\circ$ | $34^\circ \le y \le 72^\circ$ | 75,929 | 536 |
| `nepacLLhigh` | 0.1 km | $-190^\circ \le x \le -110^\circ$ | $34^\circ \le y \le 72^\circ$ | 199,914 | 9,961 |
| `worldLL`* | 5.0 km | $-20^\circ \le x \le 360^\circ$ | $-90^\circ \le y \le 84^\circ$ | 30,797 | 210 |
| `worldLLhigh`* | 1.0 km | $-20^\circ \le x \le 360^\circ$ | $-90^\circ \le y \le 84^\circ$ | 191,268 | 1,502 |

*Excludes polygons with fewer than 15 vertices after thinning.

PBS Mapping includes four data sets derived from the full resolution GSHHS database (Table 2). These all use longitude-latitude (`LL`) coordinates. The `nepac` data sets contain the northeastern Pacific Ocean shoreline in a region that extends roughly from California to Alaska (Figure 2), and the `world` data sets cover the planet (Figure 1). As discussed in section 2.2, longitude coordinates $x$ take continuous values meaningful for the intended map, with $x = 0^\circ$ on the Greenwich prime meridian.

We generated each data set from the full GSHHS database by following a consistent sequence:

- thin the database with a specified distance tolerance, as listed in the above table, using GSHHS software;
- convert the result to an ASCII file with GSHHS software;
- use our own Perl script (`gshhs2r.pl`) on this file to:
  - remove the lakes, islands in lakes, and ponds in islands;
  - eliminate small polygons, if desired, such as those with fewer than 15 points;
  - transform this file to another ASCII file with the structure of a PolySet;
- clip the data to the desired coordinate range with our own stand-alone program;
- import the ASCII table to a data frame in R/S;
- use an R function to extend the Antarctic polygon to longitude $-20^\circ$ and latitude $-90^\circ$ (`world` data sets only).

## 2.5. Examples and Applications

Our library includes an illustrative PolySet `towTracks` containing the longitude-latitude coordinates of 45 tow tracks from a longspine thornyhead (*Sebastolobus altivelis*) survey in 2001. Figure 4 portrays these data relative to the west coast of Vancouver Island, drawn with shoreline data clipped from the PolySet `nepacLL`. The PolyData object `towData` specifies the depth of each tow, represented in the figure by colours corresponding to depth intervals (black = 500-800 m, red = 800-1200 m, dark blue = 1200-1600 m).

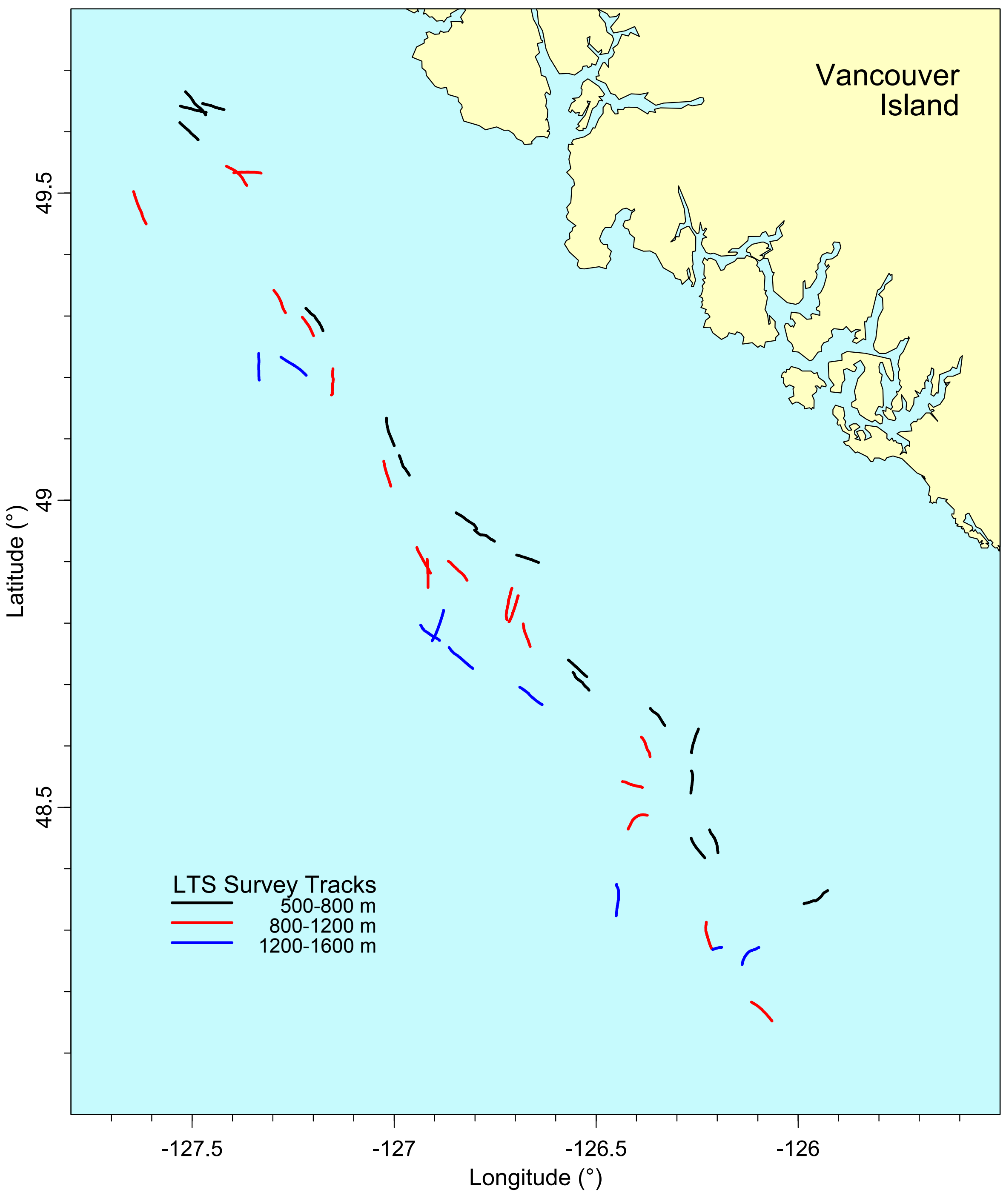


**Figure 4.** Tracks for 45 tows performed during the 2001 longspine thornyhead (*Sebastolobus altivelis*) survey along the west coast of Vancouver Island (Starr et al. 2002). A colour indicates the depth stratum for each tow. Data come from the PolySet `towTracks` and PolyData `towData`.

Figure 5 illustrates the use of our software to calculate polygon areas. We examine a region along the south-west British Columbia coast that includes a cluster of islands in the Strait of Georgia. Shoreline data come from the PolySet `nepacLLhigh`. Because area calculations do not make sense in the longitude-latitude projection, we convert the PolySet to UTM coordinates, with comparable `X` and `Y` coordinates (km), and then clip to the desired region. The figure shows areas for six selected islands, highlighted in yellow, based on the `calcArea` function. Mean coordinates for each island give a reference point for printing the island's name and area (km$^2$).

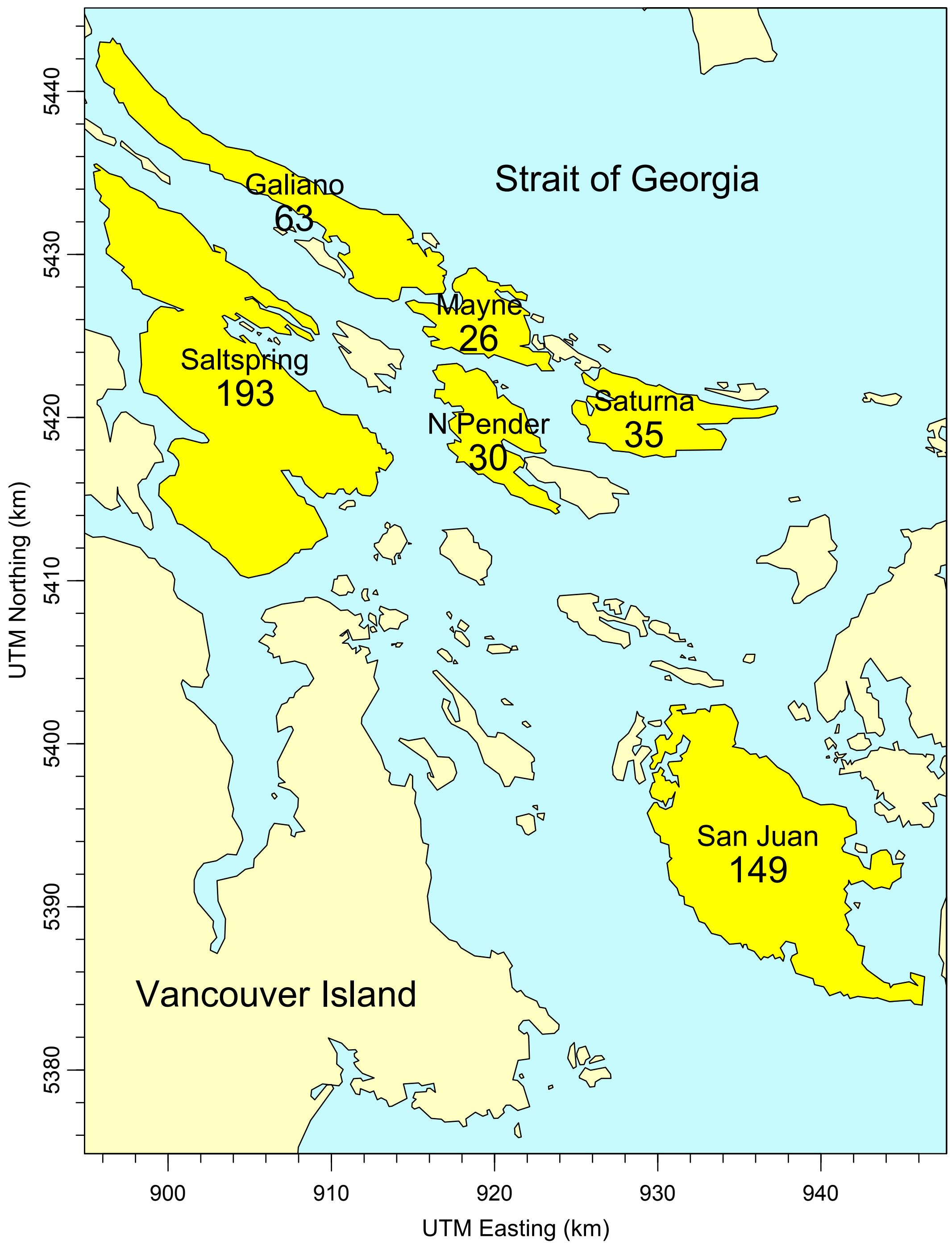


**Figure 5.** Areas (km$^2$) of selected islands in the southern Strait of Georgia. Shoreline data have been clipped from `nepacLLhigh` after conversion to UTM coordinates.

Figure 6 portrays data from Pacific ocean perch (*Sebastes alutus*) surveys conducted along the central BC coast during the years 1966-1989. The EventData object `surveyData` contains information from each tow, including the longitude, latitude, depth, catch, and effort (tow duration). These data also imply the computed value of catch per unit effort (CPUE = catch/effort). To draw the figure, we apply a sequence of mapping functions:

- `clipPolys` clips the relevant shoreline from `nepacLL`;
- `plotMap` initiates a coastal map of this region;
- `makeGrid` creates a grid in the region of interest;
- `findPolys` associates tows with the appropriate grid cells;
- `combineEvents` calculates the mean CPUE within each cell;
- `addPolys` draws cells with colours (in the `polyProps` argument) scaled to the CPUE;
- `points` (the native R/S function) plots events on the map.

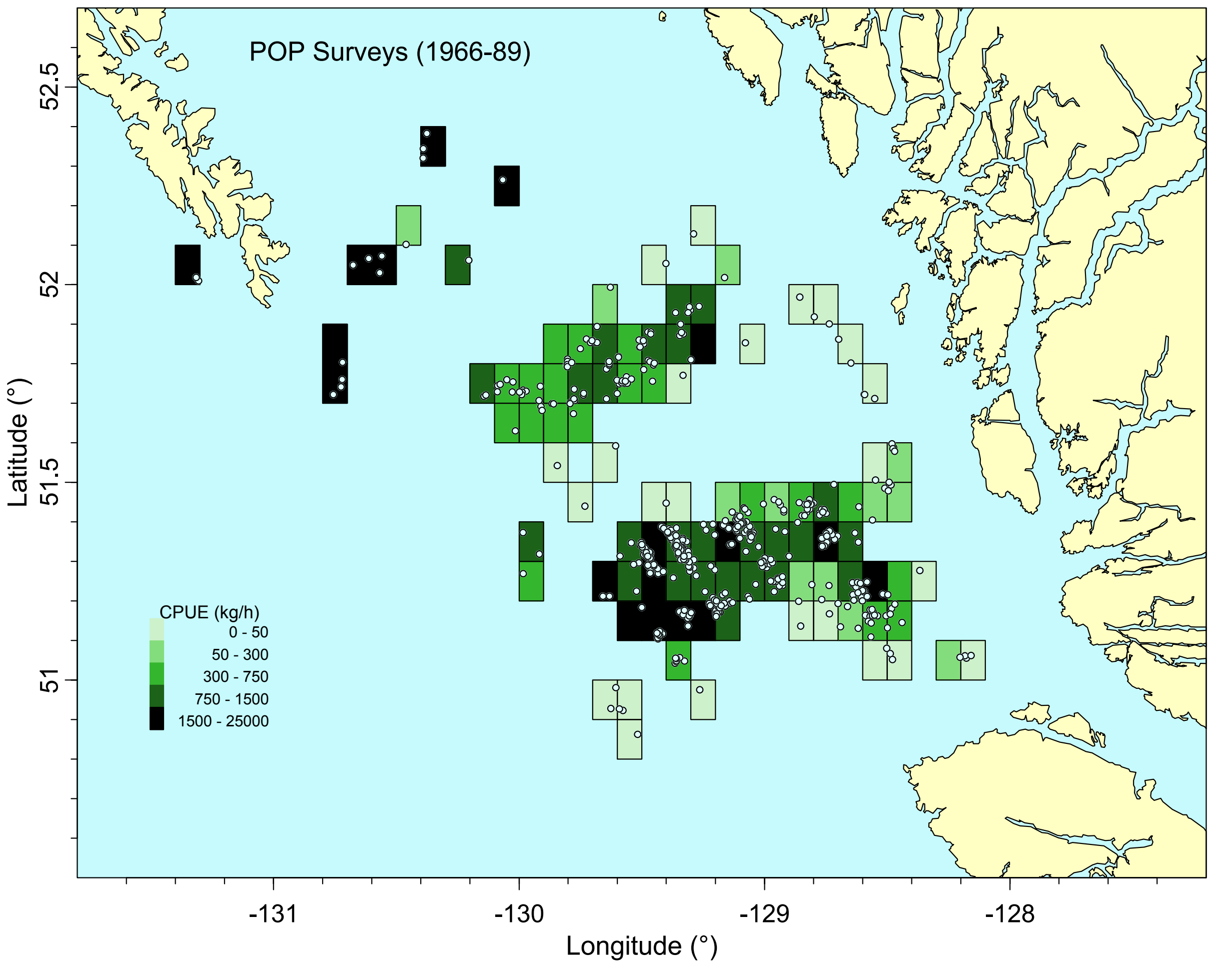


**Figure 6.** Portrayal of `surveyData` from Pacific ocean perch (*Sebastes alutus*) surveys in the central coast region of British Columbia from 1966-89, with shoreline data clipped from `nepacLL`. Colours portray the mean catch per unit effort (CPUE) within each grid cell (0.1° by 0.1°). Circles show locations of individual tows.

PBS Mapping can also display non-geographical data, such as technical drawings, network diagrams, and transportation schematics. For example, we use a PolySet to construct the proof of Pythagoras' Theorem in Figure 7, where the caption explains the logic leading to the famous result $a^2 + b^2 = c^2$. Incidentally, Devlin (1998, chapter 6, p. 221) mentions an historical incident that nicely distinguishes maps from network diagrams. A now familiar drawing of the London Underground (see the file "underground.pdf" at the web site http://www.europrail.net/maps/) fails to represent geography correctly, but contains exactly the information passengers need to navigate the system. It took two years for the designer, Henry C. Beck, to persuade his superiors that his drawing would prove useful to the public.

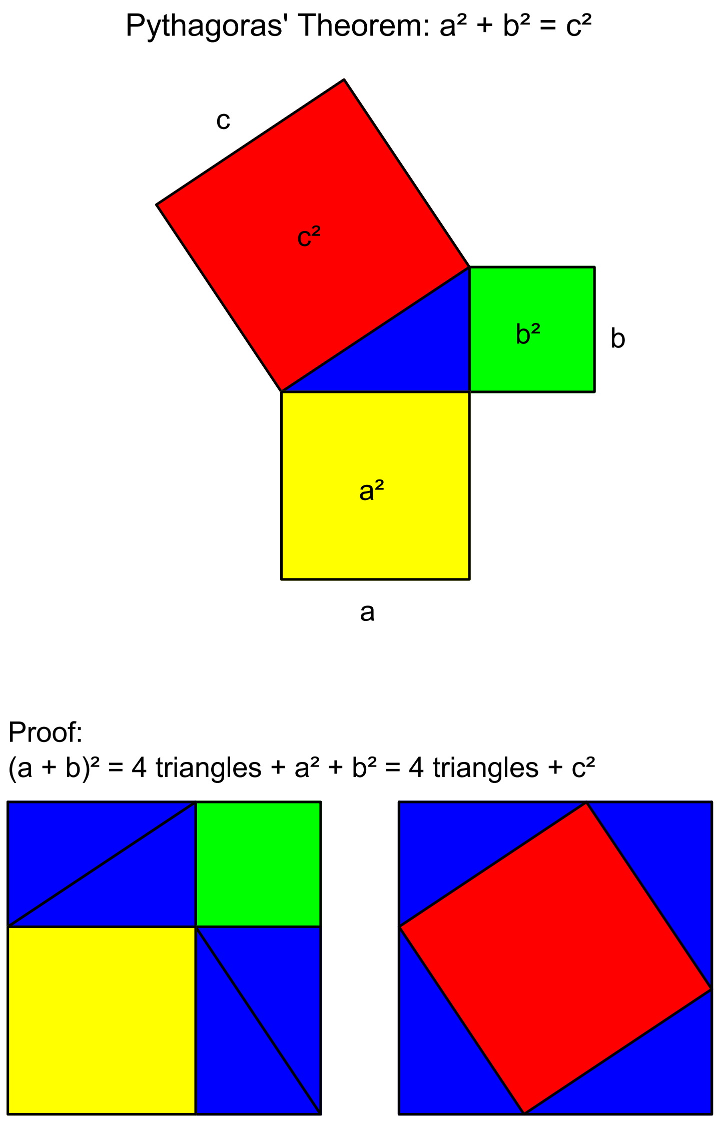

Pythagoras' Theorem: a² + b² = c²
c
c²
b²
b
a²
a
Proof:
(a + b)² = 4 triangles + a² + b² = 4 triangles + c²

**Figure 7.** Proof of Pythagoras' Theorem. A PolySet defines all geometric objects in this figure, and PolyData determine the colours for plotting. Four blue triangles plus the yellow square $(a^2)$ and the green square $(b^2)$ equal four blue triangles plus the red square $(c^2)$; consequently, $a^2 + b^2 = c^2$.

## 2.6. Strengths, Limitations, and Alternatives

PBS Mapping works with data exported from database tables, where records may not have a definite order. The `POS` field in our PolySet definition imposes the required order for polylines and polygons. This field also provides a convenient means of distinguishing inner and outer boundaries. Our PolySets have a flat structure with at most two levels, corresponding to primary and secondary IDs. We have found these limitations acceptable in the context of our work. Sceptical readers might challenge our choices, with a preference for more complex hierarchical structures. For example, Becker and Wilks (1993, 1995) define polygons as composites of polylines, so that a common boundary between two regions need be defined only once and then referenced in each regional definition. In our approach, all vertices of a common boundary must be repeated in each regional definition.

We designed our software explicitly to address a few key issues in the spatial representation of fishery data:

- easy importation from databases, Geographic Information Systems, and other sources, such as the shoreline data compiled by Wessel and Smith (1996);
- precise control over the boundaries chosen for clipping from a larger map;
- support for longitude-latitude and UTM easting-northing coordinates;
- computational ability to associate events with polygons in which they lie;
- flexible plotting tools that summarise events within grids and other polygons.

Different purposes could well lead to other designs.

# 3. UTILITIES

As described in the Preface, students have produced a number of generic utilities for dealing with operating systems, databases, and other computing problems. Their software, described in this section, may prove useful to our readers.

## 3.1. Operating System Utilities

### `dusage.exe` (Disk Usage)

This application makes it easy to see how directories use space on the hard drive. Our utility `dusage.exe` resembles the UNIX command `du`, but has different functionality. The distribution CD includes C source code (`dusage.c`). To install the application, simply copy the file `dusage.exe` into any directory on the path, as defined by the PATH environment variable.

By default, `dusage.exe` scans the current working directory and its subdirectories, and then reports the number of files and disk space (kilobytes) used in each directory. The command

```
dusage [/h] [/l] [/sN] [/uX] [PATH]
```

has optional arguments that influence the output as follows:

- `/h` display a help screen;

- `/l` show only the local directory size, excluding subdirectories;
- `/s`*N* show paths only down to the first *N* subdirectories;
- `/u`*X* display directory sizes in units *X* as indicated by a letter:
  `B`: bytes, `K`: kilobytes, `M`: megabytes, `G`: gigabytes, `T`: terabytes;
- *PATH* list of directories to check.

The *PATH* argument can include wildcards, provided that they expand only to directory names, not file names. Paths should use the backslash (\), never the forward slash (/) as in UNIX. The program's standard output can be redirected to a text file.

## **`qcd.bat` (Quickly Change Directory)**

Within a DOS shell, the command `cd` serves as a cumbersome tool for traversing directories. Our utility `qcd.bat` provides a streamlined alternative, where the `qcd` acronym means "quickly change directory". A text file serves as a directory index, and `qcd` searches this file to locate a directory that matches a given string. For example, if a drive has the path `\programs\research\pbsmapping`, then the brief command

```
qcd pbsm
```

offers a quick alternative to the conventional DOS command

```
cd \programs\research\pbsmapping
```

In general, after installing the program, run the command

`qcd /r [`*DRIVE LETTER*`]`

for each drive in the system. This renews (`/r`) the directory index. Run this command again after altering directories on a drive. (A missing drive letter implies the letter of the currently active drive.) Given a string *STR*, the command

`qcd` *STR*

will change the working directory on the current drive according to the following rules:

- The program searches the index for directories exactly named *STR* and compiles a list of all exact matches. It changes the directory either to the first exact match (if the current directory is not an exact match) or to the next exact match.
- Failing an exact match, the program searches the index for directory names beginning with *STR* and compiles a list of all the partial matches. To resolve multiple partial matches, it follows the procedure outlined above for multiple exact matches.
- Failing both an exact and partial match, the program removes the last character of *STR* to form a new string *STR'*. Then, it starts this procedure again by searching for an exact match using *STR'*. If it removes all of the characters without finding a match, the program terminates without changing the directory.

Technically, `qcd.bat` calls two other components: `qcdexe.exe` and `qcdbat.bat`. The executable `qcdexe.exe` searches the index file and creates `qcdbat.bat`, which then contains the actual command to change directories. The distribution CD contains C++ source code (`qcdexe.cpp`, `qcdexe.h`) for the executable, but the user need only be concerned with the following steps for installation:

1. Pick a (possibly new) directory for installation, and copy the files `qcd.bat` and `qcdexe.exe` into it.
2. Edit the `QCDWD` (meaning "qcd working directory") environment variable in `qcd.bat` by setting it to the exact path for the directory in step 1.
3. Either add this directory to the path, or copy the edited version of `qcd.bat` to a directory on the path.
4. Note that the command
   ```
   qcd /r D
   ```
   creates (or renews) the file `Dtree.txt` in the directory of step 1. This contains the index for drive `D`.

To uninstall the program, remove the files `qcdexe.exe`, `qcdbat.bat`, and `*tree.txt` from the program's directory. Similarly, find and remove `qcd.bat`.

## **`recent.exe` (Recently Modified Files)**

This application lists recently modified files on a drive, or the directories that contain such files. The distribution CD includes C source code (`recent.c`). To install the utility, simply copy the file `recent.exe` to any directory on the path.

By default, `recent.exe` scans the current drive for files modified since 12:00 AM of the current day. The command

```
recent [/h] [/d] [/c] [/nN] [/pPATH] [//FILE]
```

has optional arguments that influence its behaviour as follows:

- `/h` display a help screen;
- `/d` report only directory names rather than directory names *and* filenames;
- `/c` display only the first eight characters of each directory name;
- `/n`*N* scan for files modified within the previous *N* days;
- `/p`*PATH* restrict the search to *PATH* and its subdirectories;
- `//`*FILE* select files matching *FILE*, a string containing standard wildcard characters.

The program's standard output can be redirected to a text file.

## **`setenv.exe` (Set Environment Variables)**

The DOS batch language lacks a means for setting environment variables interactively. For example, although a batch file can include the command `set FileName=Report.txt` to assign a value to `FileName`, no facility exists to change the value of `FileName` based on user input. Our `setenv` utility resolves this limitation. To install it, simply copy the file `setenv.exe` to any directory on the path. The distribution CD includes C source code in `setenv.c`. In a batch file, the two commands

```
setenv.exe VARIABLE_NAME [PROMPT]
call setenvr.bat
```

prompt the user (with an optional *PROMPT*) for a value of the environment variable *VARIABLE_NAME* and then set the variable appropriately.

Technically, `setenv.exe` creates a file named `setenvr.bat` in the same directory as `setenv.exe`. This batch file contains the `set` command necessary to set the environment variable. The batch file must therefore include both calls in the pair listed above. Note that the contents of `setenvr.bat` in the installation directory are changed with every use of `setenv.exe`.

## 3.2. Mapping Utilities

The PBS Mapping package for R/S includes several algorithms that we have also implemented as standalone command-line utilities. These can handle very large data sets that may be too large for the R/S working environment. Furthermore, some users may wish to implement computational geometry calculations without reference to the R/S language. Our utilities make this possible by directly processing text files with the appropriate data format. They have been compiled with the same C code used for the dynamically linked library (DLL) in R/S. For each utility, a corresponding `.c` file provides a front end to shared code for the algorithms.

### **`clipPolys.exe` (Clip Polygons)**

The application `clipPolys.exe` reads an ASCII file containing a PolySet (explained further below) and then clips it. The command

```
clipPolys.exe /i IFILE [/o OFILE] [/x MIN_X] [/X MAX_X] [/y MIN_Y]
      [/Y MAX_Y]
```

has five arguments as follows:

| | |
|---|---|
| • `/i` *IFILE* | ASCII input file containing a PolySet (required); |
| • `/o` *OFILE* | ASCII output file (defaults to standard output); |
| • `/x` *MIN_X* | lower `X` limit (defaults to minimum `X` in the PolySet); |
| • `/X` *MAX_X* | upper `X` limit (defaults to maximum `X` in the PolySet); |
| • `/y` *MIN_Y* | lower `Y` limit (defaults to minimum `Y` in the PolySet); |
| • `/Y` *MAX_Y* | upper `Y` limit (defaults to maximum `Y` in the PolySet). |

The first line of the PolySet input file must contain the field names (`PID`, `SID`, `POS`, `X`, `Y`), where `SID` is optional. Subsequent lines must contain the data, with four or five fields per row. All fields must be white-space delimited. The program generates a properly formatted PolySet.

By default (unless otherwise specified by `/o`), this result goes to standard output, which can be redirected to a text file.

## **`convUL.exe` (Convert between UTM and LL)**

The application `convUL.exe` reads an ASCII file containing two fields named `X` and `Y`, as described further below. The command

```
convUL.exe /i IFILE [/o OFILE] (/u | /l) /z ZONE
```

has the arguments:

| | |
|---|---|
| • `/i` *IFILE* | ASCII input file containing the `X` and `Y` data (required); |
| • `/o` *OFILE* | ASCII output file (defaults to standard output); |
| • `/u` (or `/l`) | convert to UTM (longitude-latitude) coordinates (required); |
| • `/z` *ZONE* | source or destination zone for the UTM coordinates (required). |

The input file must have an initial header line with field names, including `X` and `Y`. Subsequent lines contain the data, with all fields separated by white space. The program converts each (`X`, `Y`) pair to a new pair (`X2`, `Y2`). The output file matches the input file, with the fields (`X2`, `Y2`) appended to the end of each line. The default standard output can be redirected to a text file.

## **`findPolys.exe` (Points-In-Polygons)**

The application `findPolys.exe` reads two ASCII files: one containing a PolySet and the other containing EventData. The program then determines which events fall inside the available polygons. The command

```
findPolys.exe /p POLY_FILE /e EVENT_FILE [/o OFILE]
```

has the arguments:

| | |
|---|---|
| • `/p` *POLY_FILE* | input file containing the PolySet (required); |
| • `/e` *EVENT_FILE* | input file containing EventData (required); |
| • `/o` *OFILE* | output file (defaults to standard output). |

The first line in both input files must contain field names, and subsequent lines must contain the data. All fields must be delimited by white space The PolySet must have field names (`PID`, `SID`, `POS`, `X`, `Y`), where `SID` is optional. The EventData must have fields (`EID`, `X`, `Y`). The program writes a properly formatted LocationSet, as defined in section 2.1. The default standard output can be redirected to a text file.

## **`gshhs2r.pl` (Convert GSHHS Data to PBS Map Format)**

As discussed earlier in Section 2.4, our Perl script `gshhs2r.pl` converts data from the Global Self-consistent, Hierarchical, High-resolution Shoreline (GSHHS) database to a PolySet for use with PBS Mapping. We first require an ASCII file created by `gshhs.exe`, the program supplied by the GSHHS project to convert binary to ASCII data. (Binary data may have been thinned with `gshhs_dp.exe`.) Our utility removes non-ocean shorelines, such as lakes and islands within lakes, as well as small polygons with *N* vertices or fewer. The command

```
gshhs2r.pl /i IFILE [/o OFILE] [/n N]
```

has arguments:

- `/i` *IFILE* ASCII input file created by `gshhs.exe` (required);
- `/o` *OFILE* output file for an ASCII PolySet (defaults to standard output);
- `/n` *N* minimum number of vertices in an output polygon.

If the `/n` parameter isn't specified, no filtering on the number of vertices takes place.

## 3.3. Database Utilities

Schnute et al. (1996, p. 9) argued that historical fishery data deserve "the same careful preservation that might accompany an important museum artifact or rare library book." They proposed a documentation system within Microsoft Access that maintains important metadata in six database tables (Table 3). They also supplied macros and related code modules in support of the documentation tables `A2_Tables` and `A3_Fields`. These utilities have evolved to operate with newer versions of Microsoft Access and to enhance the use of remote databases linked to an Access shell. In this section, we describe the current version of our database utilities, all contained within the Access file `Document.mdb`.

**Table 3.** Standard database documentation tables proposed by Schnute et al. (1996, p 3).

| Table Name | Contents |
|---|---|
| `A1_ReadMe` | Database background, origins, and related references |
| `A2_Tables` | Table names and descriptions |
| `A3_Fields` | Field names, descriptions, and associated tables |
| `A4_Q&A` | Common questions and answers to them |
| `A5_Questions` | Questions for which the answer is currently unknown |
| `A6_Changes` | Documented history of database changes |

### Macro 01_Create_Documents

This macro compiles the properties of tables and fields within a database and saves the resulting information in `A2_Tables` and `A3_Fields`. In particular, `A2_Tables` contains the following metadata for each table in the database:

- the table name;
- the name of the associated remote linked table, if any;
- the number of fields;
- the number of records;

- the table description.

Similarly, A3_Fields contains the following information for each field in the database:

- the name of the table in which the field occurs;
- the field name;
- the position number of the field within the table;
- the number of records with non-null values for this field;
- the number of records with null values for this field;
- the field's data type;
- the number of bytes occupied by the field;
- a flag that indicates whether or not this field is one of the primary keys;
- the field description.

Access Basic code for this macro resides in the module 00_Document.

**Macro 02_Document_Tables**

This macro takes table descriptions from A2_Tables and applies them to the actual tables in the database. Essentially, A2_Tables provides a central table description repository. The user can review and edit table descriptions in A2_Tables, then run 02_Document_Tables to broadcast the revised descriptions as properties of the database tables. Access Basic code for this macro resides in the module 00_Document.

**Macro 03_Document_Fields**

This macro takes field descriptions from A3_Fields and applies them to fields within the database tables. Essentially, A3_Fields provides a central field description repository. The user can review and edit field descriptions in A3_Fields, then run 03_Document_Fields to broadcast the revised descriptions as properties of the database fields. Access Basic code for this macro resides in the module 00_Document.

**Macro 04_Create_Empty_Database**

This macro creates a database of empty local tables from information residing in A2_Tables and A3_Fields. All tables (local and linked) not specified in the documentation tables are deleted. The primary key for each new table consists of the fields identified by Is_Key in A3_Fields, and an index is created for this primary key. Access Basic code for this macro resides in the module 04_Create_Empty_Database.

**Macro 05_Populate_Local_Database**

This macro downloads all data from tables linked to a remote database server, such as Oracle or Microsoft SQL Server. It stores the remote data in local tables with the names previously used for the linked tables. Effectively, this macro makes a local copy of a remote database. Typically, a user would fabricate a local database with appropriately linked files before running this macro. Access Basic code resides in the module 05_Populate_Local_Database.

**<u>Macro</u> 06_convUL**

As in PBS Mapping, this macro converts between longitude-latitude (LL) and UTM coordinates. It takes input from a table with field names (`lon`, `lat`) or (`utme`, `utmn`) and creates a new table with all fields from the input table, plus two fields of converted coordinates. If the argument `utm` is `TRUE`, then the macro converts UTM to LL; if `utm` is `FALSE`, it converts LL to UTM. Access Basic code resides in the module `06_convUL`.

**<u>Form</u> Object_Name_Editor**

This form renames tables and/or queries by removing prefixes or suffixes specified by the user. We found it useful when dealing with linked tables that might have odd prefixes or suffixes inherited from a remote database. The Access Basic code for this utility is embedded in the form `Object_Name_Editor`. To view the code, go to design view, right click any button, and select `Build Event`.

## 4. FREE INTERNET SOFTWARE

Where possible, we have constructed our utilities with software packages freely available on the Internet, such as R, C/C++ compilers, and Perl. This section describes briefly the packages we used, plus other high calibre products that may interest our readers. For each product, we provide one or more web site addresses. Because precise locations for downloading software can be somewhat transient, we cite relatively stable pages that normally include a link for downloading. (If some links fail, it may be necessary to truncate the web address back to a home page.) Usually, packages come with straightforward installation instructions, but in a few cases we experienced difficulties. Appendix B documents these issues and provides additional technical information that may prove helpful.

### 4.1. Scripting Languages

Windows users have probably encountered the scripting language available in a DOS shell, where typing the command `dir` gives a directory listing. A DOS batch file automates a sequence of such commands as a “script”. Unlike a compiled program in C or FORTRAN, a script usually runs as input to an “interpreter” that reads and interprets statements at run-time. This makes scripting languages handy for development and debugging. Try running statements from the command line to see how they work, and then assemble them as a program in a script file.

Scripting languages vary in their complexity. Some, like the DOS batch language, have limited support for controlling program flow and defining data objects. Others, like Perl and Python, have features similar to those in compiled languages, including logical structures for program flow and flexible data types. At an elementary level, scripting languages can automate simple tasks in a command-line environment. The more advanced languages, however, can implement full programs, such as fishery simulation models.

## Perl

Web sites: http://www.perl.com (Home page)
http://www.activestate.com (Find "Downloads", then "ActivePerl")
http://www.perldoc.com (Documentation)

Larry Wall initially designed Perl, the **p**ractical **e**xtraction and **r**eporting **l**anguage, for manipulating text files (Whitehead and Kramer 2000). Since its release in 1987, it has matured into a high-level interpreted programming language, offering complex data types and a wealth of built-in functions. For example, a program can use a *regular expression* to search a file for lines of text that contain specific patterns. It can then perform a variety of actions on the selected lines, such as extracting values or manipulating the text. Thus, the regular expression engine in Perl makes text processing easy. Our script `gshhs2r.pl` (Section 3.2) uses these capabilities of Perl for converting text files from the GSHHS format to our PolySet format. The R distribution includes extensive Perl scripts to build packages from source and documentation files, as discussed further in Section 5.6 below. We generated the PBS Mapping package for R using these scripts.

## Python

Web sites: http://www.python.org (Home page)
http://www.activestate.com (Find "Downloads", then "ActivePython")
http://www.python.org/doc (Documentation)

Guido van Rossum created the Python language starting from a mathematical design that makes code easy to read and maintain. The web site (www.python.org/doc/Summary.html) describes Python as "an interpreted, interactive, object-oriented programming language." Like many interpreted languages, it supports high-level data types, such as dictionaries and lists. It also uses *dynamic typing*, where variables change type depending on the values assigned to them.

Although interpreted languages (like R) often perform slowly, Python runs relatively quickly. Furthermore, the "NumPy" numeric extension (http://www.numpy.org/) improves its computational speed for many calculations, particularly those associated with matrices and linear algebra.

The language name comes from the British TV comedy series Monte Python. The web site takes its motto from a user comment (attributed to Mark Jackson in June 1998): "*Python - why settle for snake oil when you can have the whole snake?*"

## Parrot

Web site: http://www.oreilly.com/parrot/

In an article dated April 1, 2001, Simon Cozens posted the surprising announcement of a new scripting language named Parrot (http://www.perl.com/pub/a/2001/04/01/parrot.htm):

> "Today brought the official announcement that many of us in the Perl and Python communities had been awaiting and expecting for some time now: the culmination of the year-long collaboration between Larry Wall and Guido van Rossum, and the establishment of a period of joint development between the developers of Perl and Python."

Cozens also posted an interview with the authors, who explain that "We went over lots of possible names: Chimera, Pylon, Perth, before finally coming up with Parrot. We had a few basic ideas: we wanted it to begin with 'P'; it had to be something that wouldn't sound stupid on the end of `/usr/bin/`."

O'Reilly and Associates, publishers of the authors' guide to the Parrot language (van Rossum and Wall 2001), describe this book as

> ". . . the definitive reference for the new, dynamic programming language, Parrot, a language intended to merge the indubitable strengths of the twin Open Source scripting giants, Perl and Python. Stemming from the unprecedented meeting of minds in the new ActiveState Technical Advisory Board, Programming Parrot was written jointly by Larry Wall, the original creator of Perl, and Guido van Rossum, the inventor of Python. By uniting the unparalleled flexibility of Perl with the simplicity and maintainability of Python, Parrot is destined to become the premier application development language of the twenty-first century."

Although we highly recommend Parrot as the language of choice for most software projects, we caution users to check the date of its announcement and to read Simon Cozens' interview with its authors at the web site mentioned above. (Also, see Appendix B.)

## Tcl/Tk

| | |
|---|---|
| Web sites: | http://www.tcl.tk (Home page) |
| | http://www.activestate.com (Find "Downloads", then "ActiveTcl") |
| | http://home.pacbell.net/ouster/ (Tcl/Tk author John Ousterhout) |

A well-designed graphical user interface (GUI) can make complex tasks seem simple and intuitive. Unfortunately, rapid development of Windows GUIs often requires a particular programming environment, such as Borland Delphi or Microsoft Visual Basic. The Tcl/Tk package, which combines the scripting language Tcl with the GUI toolkit Tk, provides an excellent alternative that separates the GUI from the underlying program. For example, programmers can develop GUIs easily in Python or R using Tcl/Tk libraries. In fact, the latest releases of R (version 1.7.1) and Python (version 2.3) both include Tcl/Tk as part of the official distribution.

To use Tcl/Tk independently from a language that includes it, download it as a stand-alone program. Tcl/Tk runs on most operating systems: Windows, Macintosh, and Linux. Consequently, GUIs developed in this framework can easily be ported to other operating systems.

The author's web site shows an amusing picture of him scratching his head in puzzlement while reading the book "Tcl/Tk for Dummies". His article (Ousterhout 1998) provides interesting speculations about the role of scripting languages in the 21st century.

## 4.2 Compilers

Compilers translate languages meaningful to humans into binary instructions for computers. Historically, a number of key languages have served as the human interface, including C/C++, Fortran, and Pascal. Compilers have at least two advantages over scripting languages: they produce executable code that runs without the compiler, and the code usually runs faster than an equivalent script. On the downside, they typically lack native support for high level routines, like sorting, although these can be added with program libraries. As a result, compilers often achieve speed at the expense of greater program complexity. In this section, we give a brief history of the three languages mentioned above, along with corresponding web sites for free language support.

### C and C++

| | |
|---|---|
| Web sites: | http://gcc.gnu.org (GCC, the GNU Compiler Collection) |
| | http://sources.redhat.com/cygwin (Cygwin Project) |
| | http://www.mingw.org (MinGW Project) |
| | http://www.bloodshed.net/download.html (Dev-C++) |
| | http://www.cs.bell-labs.com/who/bwk/index.html (C author Brian Kernighan) |
| | http://cm.bell-labs.com/cm/cs/who/dmr (C author Dennis Ritchie) |
| | http://www.research.att.com/~bs/homepage.html (C++ author Bjarne Stroustrup) |
| | http://www.splint.org (Splint C code checker) |

Dennis Ritchie developed the C programming language in the early 1970s from a parent language B associated with early UNIX development (Ritchie 1993). C rapidly gained widespread acceptance, and a dialect of the language called *K&R C* follows the standard defined by Kernighan and Ritchie (1978, 1988). As a relatively small and simple language, closely related to the UNIX operating system, C continues to be a significant language even today. Stroustrup (1985) extended C to the richer, more expressive, language C++.

Modern students (like our co-author Nick) sometimes learn C or C++ as their main programming language. Compilers exist for almost every computer architecture and operating system, so that code tends to be highly portable. As mentioned earlier, we have used C to increase the speed of various algorithms in our mapping software.

The GNU Compiler Collection (GCC) contains compilers for several languages: C, C++, Objective-C, Fortran, Java, and Ada. The C compiler supports the K&R definition, as well as ANSI (American National Standards Institute) standards. Although earlier developed for Linux, two different groups distribute a binary version for Windows.

- Cygwin provides a UNIX-like environment for Windows that includes a C/C++ compiler among a collection of other Linux utilities. To function, Cygwin uses its own specialized

DLL as a Linux emulation layer. All programs built with Cygwin's compiler need this DLL on the path in order to run properly. Furthermore, noncommercial license restrictions require that programs linked with the Cygwin DLL have their source code freely available.

- MinGW (Minimalist GNU for Windows) distributes a slightly different version of the collection, including only the C, C++, and Fortran compilers. This version depends only on DLLs included with Windows. Furthermore, programs compiled with MinGW do not need to have open source code. The Dev-C++ distribution includes the MinGW compiler, along with a GUI for integrated program development.

C compilers usually check the code primarily for syntax errors. Programs can, however, contain other types of coding errors. By analogy with laundry, the code can contain *lint*, or computational fluff, resulting from various potential errors in logic. To detect such problems, the Secure Programming Group at the University of Virginia Department of Computer Science offers Splint (Secure Programming Lint), which the web site describes as "a tool for statically checking C programs for security vulnerabilities and coding mistakes."

## Fortran

| | |
|---|---|
| Web sites: | See the GCC, Cygwin, and MinGW sites for C/C++ above. |
| | http://netlib.caspur.it/linpack (LINPACK) |
| | http://www.netlib.org/eispack (EISPACK) |
| | http://www.netlib.org/lapack (LAPACK) |

The name Fortran combines the words *formula* and *translation*. Developed by IBM for numeric and scientific applications in the 1950s, Fortran became the first high level, compiled language to receive wide acceptance (Sebesta 2002, p. 45). It has evolved with the computers it runs on, passing through several language standards (Fortran 66, Fortran 77, Fortran 90 and Fortran 95). The GNU Compiler Collection includes a Fortran compiler (`g77.exe`), which supports Fortran 77 and some Fortran 90 features.

Historically, Fortran served as the principal language for sharing numerical algorithms associated with matrices and linear systems. The LINPACK, EISPACK, and LAPACK web sites show the breadth of this remarkable archive. The authorship of this report illustrates a Fortran/C generation gap. Jon and Rowan remember card decks with a Fortran statement punched on each card. Nick has never seen a card reader and considers Fortran archaic relative to C.

## Pascal

| | |
|---|---|
| Web sites: | http://www.gnu-pascal.de (GNU Pascal) |
| | http://www.bloodshed.net/devpascal.html (Dev-Pascal) |
| | http://community.borland.com/museum (Borland's "Antique" compilers) |
| | http://www.cs.inf.ethz.ch/~wirth (Pascal author Niklaus Wirth) |

In the early 1970s, Niklaus Wirth developed Pascal as a new teaching language. Consistent with the author's programming philosophy (Wirth 1975), Pascal's syntax highlights the structural features of a program. From a few years after its release to the late 1990s, Pascal was the most widely used teaching language in colleges and universities (Sebesta 2002, p. 78). The language went through some evolution to shift from an educational environment to the real world. For example, Borland developed a non-standard "Turbo Pascal" to deal with various limitations. True to their "Turbo" title, the Borland compilers translated source code to executable programs with speed that seemed almost astonishing at the time. The company has kindly released versions 1.0, 3.02, and 5.5 for free personal use. One of us (Jon) used these extensively to write programs for fishery data analysis. For further information on the availability of free Pascal compilers, see Appendix B

## 4.3 Tools for Windows

This section describes several free tools that work effectively in the Windows operating system to accomplish specific tasks, such as word processing and image manipulation.

### Adobe Acrobat Reader

Web site: http://www.adobe.com (Find "Get Adobe Reader")

Most of our readers probably know about the Adobe Acrobat Reader, but no discussion of free software would be complete without mentioning it. Adobe has devised a generic solution to the problem of distributing formatted documents. Their commercial software, Adobe Acrobat, acts like a printer driver that works with most applications to generate an output file in Portable Document Format (PDF). Although Adobe sells the tool to make PDF files, the software to read them (Acrobat Reader) is freely available. Our distribution CD includes numerous PDF files, so we strongly recommend having the Reader available.

### Emacs: Text Editor

Web sites: http://www.gnu.org/software/emacs (GNU emacs)
http://www.gnu.org/software/emacs/windows/ntemacs.html (Windows release)
http://www.analytics.washington.edu/Zope/wikis/ess (ESS)
http://www.gnu.org/software/auctex (AUCTeX)

The name Emacs originated as an abbreviation of **E**ditor **mac**ro**s**. The user manual describes Emacs as an extensible, customisable, self-documenting, real-time display editor for text files. It supports a large number of text manipulation and display features, such as

- copying and pasting, including rectangular sections of text;
- automatically filling or wrapping;
- highlighting changes in the current draft; and
- indicating white space at the end of a line.

For a wide variety of computer languages, Emacs also supports language-specific features, such as syntax highlighting, indenting, and reserved word recognition. Extensions can enhance the program's ability to deal with specific applications. For example, Emacs Speaks Statistics (ESS) configures the editor for the R/S programming language (Rossini et al. 2001). Similarly, AUCTeX configures Emacs for the TeX mathematical typesetting system described below. Section 5.1 and Appendix B give further information on using and installing Emacs.

**HTML Help Workshop (Microsoft)**

Web site: http://msdn.microsoft.com (Search for "HTML Help Workshop")

Microsoft freely provides the HTML Help Workshop to create help files for Windows operating systems. This utility includes the HTML help compiler for converting HTML and graphic files into the binary `.chm` help file format. Recent versions of Windows include a viewer that recognises and displays `.chm` files interactively. We have indirectly used the Workshop for building the PBS Mapping software, because R's Perl script for making packages calls the HTML Help Workshop to create help files in `.chm` format.

**Image Manipulation and Viewing**

Web sites: http://www.gimp.org (Image manipulation; find "GIMP for Windows")
http://www.irfanview.com (Image viewing)

The GIMP utility can retouch photographs, compose or create images, and perform other image manipulation tasks. Its features compare with commercial programs such as Paint Shop Pro (http://www.jasc.com) and Adobe Photoshop (http://www.adobe.com/products/photoshop). Subject to some copyright restrictions, it supports many file formats. For example, GIF support requires a separate download. Plug-in utilities can extend the GIMP's capabilities, and scripts can automate tasks.

Irfan Skiljan's utility "IrfanView" provides a convenient, full-featured image viewing program for Windows.

**OpenOffice.org**

Web sites: http://www.openoffice.org (Home page)
http://www.openoffice.org/product (Version 1.1 Product Description)

OpenOffice.org provides a free alternative to commercial office productivity suites, such as Microsoft Office. The web site gives this Mission Statement: "*To create, as a community, the leading international office suite that will run on all major platforms and provide access to all functionality and data through open-component based APIs and an XML-based file format.*" The organisation endorses the Open Source Initiative (OSI, http://www.opensource.org) philosophy that open source code facilitates rapid software evolution and improvement. Version 1.1 contains most features you'd expect in office software. You can create dynamic documents (Writer), analyse spreadsheet data (Calc), design eye-catching presentations (Impress), produce dramatic illustrations (Draw), edit mathematical formulas (Math), develop scripts and programs (Basic), open up your databases (Database User Tools), and create PDF files with built-in support. The product description page (cited above) gives more complete information, including an introduction to OpenOffice.org in Flash format. We discuss the installation procedure in Appendix B.

**TeX for Mathematical Typesetting**

Web sites: http://www.miktex.org (TeX distribution)
http://www.latex-project.org (LaTeX dialect)
http://www-cs-faculty.stanford.edu/~knuth (TeX author Donald Knuth)
http://research.microsoft.com/users/lamport (LaTeX author Leslie Lamport)

Donald Knuth devised TeX as "a new typesetting system intended for the creation of beautiful books — and especially for books that contain a lot of mathematics" (Knuth 1984, Preface). Essentially, the language uses a text description of a document to process fonts into a printed page. Leslie Lamport (1986) extended the language with a collection of macros called LaTeX that have become a standard dialect. The R documentation system builds LaTeX files that produce a convenient software manual, as illustrated here by our documentation of PBS Mapping (Appendix D). The MiKTeX web site provides an excellent Windows distribution. For further technical information, see Appendix B.

**UNIX Tools (Cygwin and R)**

Web sites: http://sources.redhat.com/cygwin (Cygwin)
http://www.stats.ox.ac.uk/pub/Rtools/tools.zip (Tools for R)

UNIX includes a much richer suite of command line tools than those available in a DOS window. In fact, the R utility to build packages requires a small set of UNIX tools, which can be downloaded from the web site above. Cygwin provides a more complete version of UNIX for DOS/Windows, based on dynamically linked libraries (DLLs), as discussed in Section 4.2 with regard to C/C++ compilers. The Cygwin installer presents a suite of optional UNIX software for emulation in Windows, ranging from the 'bash' shell to the 'X-Windows' GUI environment.

## 4.4. Analytical Tools

The rich literature of numerical algorithms in Fortran (Section 4.2) epitomises classical software design: first build a library of tested routines, then tie them together with a custom program to conduct a specific analysis. Modern analytical software, like R, streamlines this process by providing an environment where the routines exist as part of the language and simple statements tie them together.

**R**

Web sites: http://www.r-project.org (CRAN: Comprehensive R Archive Network)
http://cm.bell-labs.com/cm/ms/departments/sia/jmc (S author John Chambers)
http://www.stats.ox.ac.uk/~ripley (Author Brian Ripley)
http://www.cmis.csiro.au/bill.venables (Author Bill Venables)

Like the commercial product S-PLUS (http://www.insightful.com/products/splus), R supports high-level data structures and provides a wealth of plotting commands that make data visualisation easy. The language gives particular emphasis to statistical analysis, with functions and language structures designed explicitly for statistical models. According to the CRAN web site (with slight changes from their exact text):

> R is a GNU project similar to the S language and environment, which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.
>
> R was initially written by Robert Gentleman and Ross Ihaka, also known as "*R&R*", of the Statistics Department of the University of Auckland. . . . A special debt is owed to John Chambers who has graciously contributed advice and encouragement in the early days of R and later became a member of the core team.

The authors Bill Venables and Brian Ripley appear in the list of core developers and contributors. Their books (Venables and Ripley 1999, 2000) provide excellent guides to the language and application of R and S.

We have designed PBS Mapping for compatibility with both R and S-PLUS.

### Scilab

Web site: http://www-rocq.inria.fr/scilab

The web site describes Scilab as "a scientific software package for numerical computations providing a powerful open computing environment for engineering and scientific application." Its historical roots lie partly in the numerical Fortran algorithms mentioned earlier. Like R, Scilab has a commercial counterpart Matlab (http://www.mathworks.com). The engineering focus of Scilab/Matlab differs somewhat from the statistical focus of R/S, but both language environments offer powerful tools for numerical modelling, analysis, and visualisation.

### GIS (freegis.org)

Web site: http://www.freegis.org

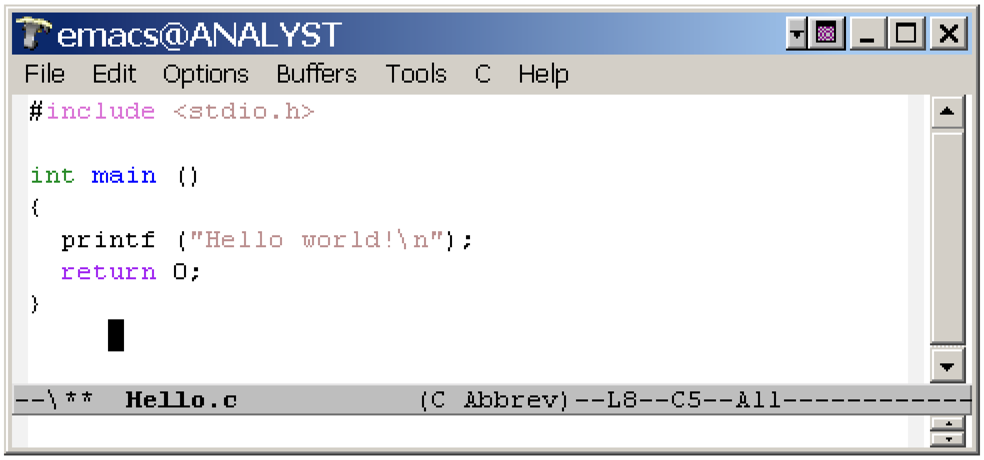
Because PBS Mapping includes features often supported by a Geographic Information System (GIS), a free GIS alternative may someday be much better than the software described here. The above web site summarises the current status of free GIS programs and data. Organised by categories and subcategories, it lists web sites with available software, data, projects, and documents. The list receives frequent updates and grows steadily.

## 5. QUICK TUTORIALS

Have you ever experienced the frustration of learning the tricks needed to accomplish some task, only to forget them the next time you need them? While developing our software, we had to learn technical details about C compilers, text editors, R packages, and various other applications. Sometimes a topic might require an entire book for complete documentation, but a few key concepts provide enough to get started. This section offers a few quick tutorials, based on our experiences.

### 5.1. Using Emacs

Emacs comes from UNIX, which often uses terminology different from that in Microsoft Windows. An Emacs window, in the Microsoft sense, is called a *frame*. Subdivisions within that frame are called *windows*. An Emacs session can consist of one or more frames, each with one or more windows. Most applications, however, need only a single Emacs frame. When reading the Emacs help files, remember these specialised meanings of *frame* and *window*.



```
#include <stdio.h>

int main ()
{
  printf ("Hello world!\n");
  return 0;
}
```

```
--\**  Hello.c            (C Abbrev)--L8--C5--All-----------
```

**Figure 8.** An Emacs frame displaying the file `Hello.c`, which contains C source code for a classical "Hello world" program. The frame includes a title bar, menu bar, editing window, status line, and echo area (or minibuffer window). Emacs recognises the `.c` file extension and highlights text in a manner consistent with C code. The black rectangular cursor lies in line 8 to the right of column 5 (`L8—C5` in the status line).

Figure 8 shows the components of an Emacs frame, when the program edits a C source code file (`Hello.c`). A *title bar* appears at the top of the frame, consistent with most Windows programs. A *menu bar* lies immediately below the title bar. The menu can change in relation to the type of file (such as `.c`, `.r`, or `.tex`) being edited. A toolbar (not shown here) can also appear below the menu bar. The *editing window* below the menu bar in Figure 8 contains text from the file, with highlighting appropriate to the C language. Each editing window ends with a *status line* that shows window's status, such as the file name and cursor position. The last line of the frame is the *echo area* or *minibuffer* window, where editing commands (such as searching or formatting) may prompt for additional information.

Keystroke combinations control most of Emacs' behaviour, as illustrated by the commands in Nick's Cheat Sheet (Table 4). Overall, Emacs accepts hundreds of keystroke combinations, and the extensive online help lists them all. A seasoned Emacs user can quickly accomplish most editing tasks with a deft sequence of keystrokes. Many commands can also be

**Table 4.** A guide to using keystroke commands in Emacs, also known as *Nick's Emacs Cheat Sheet*. Help files use a different notation, where "C-x" means <Ctrl-x> and "M-x" means <Meta-x>. In Windows, <Alt> serves as the <Meta> key, so that "M-x" translates to <Alt-x>.

| | |
|---|---|
| **File Manipulation** | |
| <Ctrl-x>, <Ctrl-f> | Create/open a file (buffer) |
| <Ctrl-x>, k | Close a file (buffer) |
| <Ctrl-x>, s | Save the current file (buffer) |
| <Ctrl-x>, <Ctrl-c> | Save files (buffers) and close Emacs |
| **Frame Manipulation** | |
| <Ctrl-x>, 5, 0 | Delete the current frame |
| <Ctrl-x>, 5, 1 | Delete all other frames |
| <Ctrl-x>, 5, 2 | Create a new frame |
| <Ctrl-x>, 5, o | Cycle the cursor through the frames |
| **Window Manipulation** | |
| <Ctrl-x>, 0 | Delete the current window |
| <Ctrl-x>, 1 | Delete all other windows |
| <Ctrl-x>, 2 | Split the current window vertically |
| <Ctrl-x>, 3 | Split the current window horizontally |
| <Ctrl-x>, o | Cycle the cursor through the windows |
| **Cut/Copy/Paste** | |
| <Ctrl-x>, h | Highlight the whole buffer |
| <Ctrl-Space> | Start highlighting a region |
| <Ctrl-g> | Remove highlighting |
| <Ctrl-w> | Cut highlighted region |
| <Alt-w> | Copy highlighted region |
| <Ctrl-k> | Cut current line (pressing multiple times cuts multiple lines) |
| <Ctrl-y> | Paste |
| **Searching/Replacing** | |
| <Ctrl-s> | Incremental search (forward) |
| <Ctrl-r> | Incremental search (reverse) |
| <Alt-%> | Start a search/replace operation |
| **Help** | |
| <Ctrl-h>, i | Display the help |
| **Miscellaneous Commands** | |
| <Ctrl-g> | Stops the Emacs command currently in progress |
| <Ctrl-l> | Scroll the current window vertically to centre the cursor |
| <Alt-x>, indent-region | Indent a highlighted region |
| <Alt-x>, comment-region | Comment a highlighted region |
| <Alt-x>, goto-line | Go to a specific line number |
| <Alt-x>, tabify | Convert all spaces in highlighted region to tabs |
| <Alt-x>, untabify | Convert all tabs in highlighted region to spaces |
| <Esc-i> | Insert a tab character |

accessed via the Emacs GUI. In the usual style for Windows, the File menu gives options for opening, saving, or closing files, and the Edit menu supports search and replace operations. The Options menu includes various configuration choices, such as turning syntax highlighting on or off. As with most Windows applications, it helps to explore the possibilities.

Emacs uses various *modes* to support different file types. Modes can determine many things, such as indenting, text colouring, and items on the menu bar. The status line shows the current mode (`C Abbrev` in Figure 8). Emacs includes native support for many applications, such as Perl and LaTeX. Furthermore, modes can be added with suitable configuration files, as illustrated by the Emacs Speaks Statistics (ESS) package to support source code for R or S-PLUS. File extensions normally determine the mode automatically, although the mode can be changed manually by pressing <Alt-m> and typing the mode name.

Learn Emacs by delving into it! Nick's Cheat Sheet (Table 4) helps diminish the learning curve, and Nick guarantees that the benefits will outweigh the costs. Available for Linux, Mac OS X, and Windows in both command-line and graphical environments, Emacs is highly portable. Experience gained on one platform transfers directly to all others.

## 5.2. Choosing a C/C++ Compiler

Section 4.2 mentions various C/C++ compilers that run within Windows. They differ in several respects, such as adherence to standards, compilation time, and inclusion of an IDE. The resulting executables may also differ in size and execution time. A detailed comparison of compilers warrants its own report, but our work has led to observations that illustrate some of the possibilities.

In a scientific environment, execution time can be important. To compare the product of several compilers, we used our points-in-polygons routine `findPolys.exe` as a test case. Executables from the Microsoft, Intel, and MinGW GCC compilers all produced a similar run time. However, the Cygwin GCC compiler gave an executable that outperformed its competitors by finishing execution in nearly half the time. The difference appeared to occur in the input and output routines, perhaps due to implementation in the Cygwin DLL. Possibly, subtle choices in compiler options might have influenced these results. We use this example only to illustrate that compiler choice can influence execution time. It certainly offers no definitive recommendation for one compiler over another.

We also noticed size differences in the resulting executables. For example, MinGW GCC version 2.95 created a 7 kilobyte executable `recent.exe`. When MinGW GCC version 3.2 compiled the same source file, the executable occupied 30 kilobytes. Perhaps the difference stems from default compilation options and libraries embedded in the code by the two compilers. Again, we use this example only to illustrate that compiler choice can influence executable size.

Where execution time and executable size are important, it may prove helpful to explore various compiler choices. The free GNU Compiler Collection offers several possibilities, and some commercial companies (such as Intel) provide free demonstration versions of their compilers.

## 5.3. Building Software with C/C++

The GNU Compiler Collection (GCC) includes a variety of tools for compiling source code into machine-readable object code. In particular,

- `gcc` compiles C source code, usually from files with the extension `.c`; and
- `g++` compiles C++ source code, usually from files with the extension `.cpp`.

Our examples pertain to C code, but similar commands work for C++ code after replacing `gcc` with `g++`.

A normal cycle translates source to object code (`*.o`) and then links required libraries to give an executable file (`*.exe` or `*.dll`). Arguments on the command line determine the steps in this process, which may go directly from source code to an executable file. The command

```
gcc --help
```

gives a partial list of the available options, and the user manual shows even more. Although these long lists can seem a bit daunting, it helps to think of the basic C compiler command as

```
gcc -o OFILE [ARGS] IFILES
```

with the arguments

| | |
|---|---|
| • `-o` *OFILE* | output file name (e.g., `*.o`, `*.exe`, or `*.dll`); |
| • *ARGS* | additional compiler arguments; |
| • *IFILES* | one or more input input files (e.g., `*.c` or `*.o`). |

Notice that compiler arguments use the UNIX hyphen (`-`), rather than the Windows slash (`/`).

Our Emacs example (Figure 8, Section 5.1) shows a simple C program in a file named `hello.c`. To compile this source code into an executable file named `hello.exe`, use the command:

```
gcc -o hello.exe hello.c
```

Alternatively, two steps also give the intermediate object code `hello.o`:

```
gcc -o hello.o -c hello.c
gcc -o hello.exe hello.o
```

where the `-c` option in the first line specifies compilation only.

Making a dynamically linked library (DLL) for Windows requires additional steps. Perhaps the easiest method uses the tools `dllwrap.exe` and `dlltool.exe` included with the Dev-C++ distribution (Section 4.2). Table 5 lists source code in a file `fib.c` intended for a making a DLL, where the `fibonacci` function uses a pointer variable for accepting input and returning output. The three commands

```
gcc -o fib.o -c fib.c
dlltool --output-def fib.def --export-all-symbols fib.o
dllwrap -o fib.dll --def fib.def fib.o
```

successively produce object code `fib.o`, a slightly mysterious definition file `fib.def`, and the final library `fib.dll`.

**Table 5.** C source code file `fib.c` that defines a function to calculate the nth Fibonacci number from the recursion $x_n = x_{n-2} + x_{n-1}$ with $x_0 = 0$ and $x_1 = 1$. For use in a dynamically linked library (DLL), the function uses the pointer `*n` to pass the input integer $n$ and the return value $x_n$. It returns $-1$ when $n < 0$.

```
void fibonacci (long *n)
{
  long i, xa = 0, xb = 1, xn, nn = *n;

  if (nn < 0)
    xn = -1;
  else if (nn <= 1)
    xn = nn;
  else {
    for (i = 2; i <= nn; i++) {
      xn = xa + xb;
      xa = xb;
      xb = xn;
    }
  }

  *n = xn;
}
```

The GNU collection also includes a standard UNIX utility `make.exe` that facilitates execution of commands needed to create a file like `fib.dll`. This program uses a text file with the standard name `Makefile` in the directory that contains the relevant project files. Running the command `make` then causes the desired target files to be constructed according to specified rules. Although a full explanation of the make utility requires much more explanation, Table 6 illustrates a `Makefile` for `fib.dll` that automates the three commands listed above. For further information, see the web site http://www.gnu.org/software/make/.

**Table 6.** Contents of a `Makefile` to create `fib.dll`. The target `fib.dll` depends on `fib.o`, and the target `fib.o` depends on `fib.c`. The commands to build a target appear below the line that shows target dependencies.

```
fib.dll: fib.o
     dlltool --output-def fib.def --export-all-symbols fib.o
     dllwrap -o fib.dll --def fib.def fib.o
fib.o: fib.c
     gcc -o fib.o -c fib.c
```

**5.4. Adding Software Product Information**

Microsoft Windows allows the embedding of product information in executable files and dynamically linked libraries (DLLs). You can see this for most Microsoft software, such as the Windows Explorer (`explorer.exe`, usually in main Windows directory), by right-clicking the program icon and selecting `Properties`. The `Version` tab shows various entries, including a version number, description, and copyright. The file `PBSmapping.dll` in our mapping software includes similar product information. How did we manage to get it there?

The key to solving this problem lies in a *Windows Resource* file (`*.rc`) that specifies the required product information. Table 7 illustrates technical resource code written for the Fibonacci example in Tables 5 and 6. Starting from the two files

- `fib.c` (the C source code in Table 5) and
- `fib_res.rc` (the Windows resource code in Table 7),

the following sequence of commands:

```
     gcc -o fib.o -c fib.c
     dlltool --output-def fib.def --export-all-symbols fib.o
     windres -o fib_res.o fib_res.rc
     dllwrap -o fib.dll --def fib.def fib.o fib_res.o
```

produces the dynamically linked library `fib.dll`, complete with product information. The command `windres.exe` in the third line, included with the Dev-C++ distribution, converts the resource code `fib_res.rc` into object code `fib_res.o`. The final line uses `dllwrap.exe` to link the two object files `fib.o` and `fib_res.o` into `fib.dll`. Here's an exercise for the keen reader. How would you change the `Makefile` in Table 6 to include the resource file `fib_res.rc` in building `fib.dll`? (Hint: Add a dependency line for `fib_res.o`.)

Microsoft provides detailed instructions for writing resource files, like the one shown in Table 7, although we had some difficulty finding this information. Currently, it resides at

http://msdn.microsoft.com/library/en-us/tools/tools/versioninfo_resource.asp.

If this link fails, we suggest using a good search engine, like Google (http://www.google.ca), to find the subject `versioninfo_resource`.

**Table 7.** A Windows resource file `fib_res.rc` for adding product information to `fib.dll`.

```
#include <windows.h>

VS_VERSION_INFO VERSIONINFO
  FILEVERSION 1,0,0,0
  PRODUCTVERSION 1,0,0,0
  FILEFLAGSMASK 0x3fL
  FILEFLAGS 0x0L
  FILEOS VOS__WINDOWS32
  FILETYPE VFT_DLL
  FILESUBTYPE 0x0L
BEGIN
  BLOCK "StringFileInfo"
  BEGIN
    BLOCK "040904b0"
    BEGIN
      VALUE "CompanyName", "Fisheries and Oceans Canada\r\n\
Pacific Biological Station\0"
      VALUE "FileDescription", "Calculates the n-th Fibonacci\0"
      VALUE "FileVersion", "1.00\0"
      VALUE "InternalName", "Fibonacci Test\0"
      VALUE "LegalCopyright", "Copyright © 2003\0"
      VALUE "OriginalFilename", "fib.dll\0"
      VALUE "ProductName", "Fibonacci Test\0"
      VALUE "ProductVersion", "1.00\0"
    END
  END
  BLOCK "VarFileInfo"
  BEGIN
    VALUE "Translation", 0x409, 1200
  END
END
```

## 5.5. Embedding C Functions in R/S

The rich R/S programming environment sometimes runs slowly, particularly with problems that require explicit looping. The PBS Mapping software illustrates situations in which compiled C code runs much faster. A dynamically linked library connects the C and R/S environments, where the same DLL file works with both R and S-PLUS 2000. Unfortunately, S-PLUS Release 6 uses a different function calling convention, and the current version of PBS Mapping works only with S-PLUS 2000.

C and R/S code written for use with an interfacing DLL must conform to several restrictions:

- C does not support the high-level R/S data types. Instead, each type has a specific representation in C (Table 8).

- R/S code must define objects, and thus allocate memory for them, before calling C functions in the DLL. Some of these objects may contain predefined input values. Others serve as locations for the output, to be modified by the C code.
- C functions can create additional variables for internal calculations, but allocated memory must be freed before the function terminates. Otherwise, the memory will be lost, creating a proverbial *memory leak*.
- C functions directly accessible to R/S must have `void` return types. All results are returned via predefined function arguments. R/S may crash if the code attempts to call a non-`void` function.
- C code written for the shared DLL should not contain a `main` function.

**Table 8.** C representations for R/S data types.

| R/S Object | R | S |
|---|---|---|
| logical | `int *` | `long *` |
| integer | `int *` | `long *` |
| single | `float *` | `float *` |
| double | `double *` | `double *` |
| complex | `Rcomplex *` [1] | `struct{double re, im;} *` |
| character | `char **` | `char **` |
| name | `SEXP` [2] | `char *` |
| list | `SEXP *` [2] | `char **` |
| numeric | `double *` | `double *` |

[1] `Rcomplex` is defined in `Complex.h`.
[2] `SEXP` is defined in `Rinternals.h`.

A normal cycle of C–R/S code development requires a C source file that defines functions suitable for the interface DLL. Once the DLL has been created, the library must be loaded into the R/S environment. This makes functions within the DLL available via the R/S call `.C()`. We describe this process by using the Fibonacci example in Section 5.3.

Suppose that `fib.dll` contains the function `fibonacci` defined in Table 5. Then the following short R session would load the library, find the 10th Fibonacci number, display the result, and unload the library (good programming practice when a library is no longer needed):

```
> dyn.load("fib.dll")
> result <- .C("fibonacci", n = as.integer(10))
> print(result)
$n
[1] 55
> dyn.unload("fib.dll")
```

The `.C` call references the function `"fibonacci"` and its sole integer argument. We use the name `n` in the R code to match the integer pointer `*n` in the C code (Table 5), but this naming convention isn't necessary. Both R and C must refer to an object of the same type, in this case a single integer. A conversion function (`as.integer` here) helps ensure the correct type. On input, `n` contains `10`. The call returns a list `result`, with a component for each function argument. The C `fibonacci` function changes `n` from its input value (10) to the 10th Fibonacci number (55), the value of `n` in the `result`.

A similar session for S-PLUS would appear as follows:

```
> dll.load("fib.dll", "fibonacci", "cdecl")
> result <- .C("fibonacci", n = as.integer(10))
> print(result)
$n
[1] 55
> dll.unload("fib.dll")
```

S-PLUS requires more specific arguments for the library loading function: a vector of relevant function names (here only `"fibonacci"`) and a protocol for calling functions in the DLL (here the C calling convention `"cdecl"`).

### 5.6. Creating R Packages

The R project defines a standard for creating a *package* of functions, data, and documentation. To build a package, you must have the following software installed on your computer:

- R version 1.7.0 or later;
- Perl version 5 or later;
- the Microsoft HTML Help Workshop;
- various UNIX tools, including `make`, `sh`, `cat`, `cp`, `diff`, `echo`, `mkdir`, `mv`, `rm`, `sed`, and any files needed to create a DLL from C source code (if any).

The Windows `PATH` should include the relevant directories to invoke all this software.

Anonymous (2003a) gives a complete technical description of the requirements for building an R package, which typically includes code, data, and documentation. The input components reside in various files and directories. For example, the PBS Mapping library includes:

- `\PBSmapping\DESCRIPTION`, a text file that describes the package;
- `\PBSmapping\R\` with R source code files (`*.R`);
- `\PBSmapping\src\` with C source code files (`*.c`, `*.h`, `*.rc`);
- `\PBSmapping\data\` with binary R data files (`*.rda`) created with the R `save` function;
- `\PBSmapping\man\` with documentation in text files (`*.Rd`, supported by Emacs with ESS) that conform to a special R documentation format (Anonymous 2003a,b);
- `\PBSmapping\inst\` with files and subdirectories that contain various documents, such as this report in a PDF file.

The R packaging utility uses Perl scripts for various tasks, such as converting R documentation files (`*.Rd`) into various other formats, performing consistency checks, compiling C code, and building archives for distribution. These archives use zip format (`*.zip`) for a binary distribution and a UNIX compression format (`*.tar.gz`) for a source code distribution.

We have two suggestions for writing R code and documentation intended for a package. First, R functions that begin with a period, considered hidden in the UNIX sense (Appendix C), need no documentation. Second, examples in the documentation files must explicitly load any required data using the `data` function. In PBS Mapping, for example, we make frequent use of hidden functions to share code among the high level functions intended for the user. Our examples load data objects like `nepacLL`, where necessary.

R technical documents (Anonymous 2003a,b) provide the best complete recipe for building an R package. As an aid to readers who want a condensed version, we list the steps used in building PBS Mapping. This recipe offers a prototype for building other packages.

1. Save each desired data object to a file (`*.rda`) using the `save()` function.

2. Create the file `DESCRIPTION` according the instructions in Anonymous (2003a).

3. Create the package's directory structure, as described above. Place the relevant files in the appropriate directories. The root directory name (here `\PBSmapping`) must match the package name specified in `DESCRIPTION`.

4. Add a file with the standard name `zzz.R` to the `\R` subdirectory that defines the R function `.First.lib`, which is invoked automatically when loading the package. In PBS Mapping, this function has the definition

   ```
   .First.lib <- function(lib, pkg) {
     library.dynam("PBSmapping", pkg, lib)}
   ```

   where the command `library.dynam` loads `PBSmapping.dll`. We omit the extension `.dll` because it is implied by the Windows operating system. (By contrast, UNIX uses archive libraries `*.a`.)

5. In a DOS window, change into the parent directory for `\PBSmapping`. Issue the command

   ```
   Rcmd CHECK PBSmapping
   ```

   to check the package for errors. The command `Rcmd.exe` in the R distribution invokes the necessary Perl scripts to accomplish various tasks.

6. Correct all errors found in step 5, and repeat the step until no errors occur. For additional information on an error, look at the files in the `\PBSmapping.check` subdirectory.

7. To create a binary package (`PBSmapping.zip`) that can be installed within the R GUI, run the command

   ```
   Rcmd BUILD --binary PBSmapping
   ```

   Installing this file from the GUI requires no special software other than R itself.

8. To create a source package (`PBSmapping.tar.gz`) that can be installed from the command line, run the command

   ```
   Rcmd BUILD PBSmapping
   ```

   To install the package from this source file, run the command

   ```
   Rcmd INSTALL PBSmapping.tar.gz
   ```

   Installing this file from the command line requires all the software listed at the beginning of this section.

9. To remove the installed package, run the command

   ```
   Rcmd REMOVE PBSmapping
   ```

   This command potentially requires the software listed at the beginning of this section.

## 5.7. Creating S-PLUS 2000 Libraries

Insightful Corporation (www.insightful.com), formerly MathSoft Inc., defines a method for creating an S-PLUS *library* of functions, data, and documentation (Anonymous 1999). This task requires a computer with S-PLUS and a compression utility, such as WinZip. Our discussion applies to S-PLUS 2000. The more recent version S-PLUS 6 introduces a new function `createChapter` that automates parts of this process.

The S-PLUS documentation (Anonymous 1999) provides the best recipe for building an S-PLUS 2000 library. As an aid to readers who want a condensed version, we list the steps used in building PBS Mapping. These illustrate the process required to build other libraries.

1. In the S-PLUS `library` directory, create a subdirectory for the new library. We use `\PBSmapping`, where the subdirectory name designates the library name.

2. Inside this new subdirectory, create the following subdirectories:
   ```
   _Data
   _Data\_Help
   _Prefs
   ```

3. In an S-PLUS session, create or import all necessary functions and data objects. For example, we use the command `source("PBSmapping.R")` to define our functions. Similarly, we use `read.table()` to load data objects from text files.

4. Define the S function `.First.lib`, which runs automatically when the library is loaded. In PBS Mapping, this function is

   ```
   .First.lib <- function(library, section)
   {
     dll.load("PBSmapping.dll",
       c("clip", "integrateHoles", "calcArea", ...), "cdecl")
   }
   ```

   The command `dll.load` loads `PBSmapping.dll`. The second argument specifies the vector of DLL functions made available in S, where our notation "`...`" indicates that the above list is incomplete. To call one of these functions within S, use `.C()`. The final argument `"cdecl"` indicates the C calling convention for accessing DLL functions.

5. Load the new (empty) library by issuing the command

   ```
   PBS.pos <- library(PBSmapping, first=T)
   ```

   where `PBS.pos` specifies the position of the `PBSmapping` database in the search list.

6. Create a vector of all function and data object names to include in the library. For PBS Mapping, these names appear in Appendices C–D, and the required vector is

   ```
   PBS.objs <- c(".First.lib", ".addAxis", ".addLabels",
     ..., "addLines", "addPolys", ..., "worldLL", "worldLLhigh")
   ```

   where our notation "`...`" indicates omissions from the complete alphabetical list. Then use the following code to copy each library object into the PBS Mapping database:

   ```
   for (i in 1:length(PBS.objs)) {
     assign(PBS.objs[i], get(PBS.objs[i]), where = PBS.pos)
   }
   ```

   If S-PLUS warns about masking existing objects, ignore these warnings.

7. S-PLUS libraries can contain documentation as text files in the `_Data\_Help` subdirectory. The name of each help file must match the true filename of the object it describes. Obtain these filenames with the S command

   ```
   for (i in 1:length(PBS.objs)) {
      print(PBS.objs[i]);
      print(true.file.name(PBS.objs[i], where = PBS.pos))
   }
   ```

8. At this point, steps 6 and 7 have altered the contents of `\PBSmapping\_Data` and `\PBSmapping\_Data\_Help`. Copy the file `PBSmapping.dll` into `\PBSmapping`, and compress the entire `\PBSmapping` directory tree, with paths preserved, using a program such as WinZip.

9. To install the library on another computer, uncompress this file into the `library` subdirectory of the S-PLUS program directory. This creates the subdirectory tree `\PBSmapping`. Move `PBSmapping.dll` from `\PBSmapping` into the S-PLUS `bin` directory.

10. To remove the installed package, delete the `\PBSmapping` subdirectory tree from the S-PLUS `library` directory, and remove `PBSmapping.dll` from the `bin` directory.

### 5.8. Building GUIs with Tcl/Tk in Python and R

Both Python and R can interface with Tcl/Tk to create graphical user interfaces (GUIs). Python uses the module `Tkinter`, and R uses the package `tcltk`. Recent versions of both languages include Tcl/Tk within their standard distributions. The following seven-line bilingual example opens a window that displays the text message "Hello world!" and closes when a user clicks the "Exit" button. Line 1 loads the appropriate module or library. Line 2 creates an empty window. Line 3 creates the label "Hello world!", and line 4 adds it to the window. Line 5 creates an "Exit" button with the associated command to destroy the root window, and line 6 adds it to the window. Finally, line 7 displays the window to the user.

<u>Python Code</u>

```
1. from Tkinter import *
2. root = Tk()
3. lblHello = Label(root, text="Hello world!")
4. lblHello.pack()
5. btnExit = Button(root, text="Exit", command=root.destroy)
6. btnExit.pack()
7. root.mainloop()
```

<u>R Code</u>

```
1. library(tcltk)
2. root <- tktoplevel()
3. lblHello <- tklabel(root, text="Hello world!")
4. tkpack(lblHello)
5. btnExit <- tkbutton(root, text="Exit",
     command=function() tkdestroy(root))
6. tkpack(btnExit)
7. tkfocus(root)
```

## ACKNOWLEDGEMENTS

We thank Dr. Jim Uhl and Dr. Peter Walsh in the Computing Science Department, Malaspina University-College, for encouraging and facilitating the role of students in applied fisheries research. Without the dedicated work of these students, named in the Preface, we could not have produced the software described here. We also acknowledge the work of Dr. Paul Wessel and Dr. Walter Smith in compiling their valuable shoreline (GSHHS) database. Our colleague Brian Krishka helped prepare the data objects `surveyData`, `towData`, and `towTracks`. The PBS Mapping package could not exist without R and GCC. We express admiration and gratitude to the remarkable teams that build, document, and distribute such outstanding free software.

## REFERENCES

Anonymous. 1998. The ellipsoid and the Transverse Mercator projection. Geodetic Information Paper No. 1 (version 2.2). Ordnance Survey, Southampton, UK. 20 p. URL: http://www.ordsvy.gov.uk/.

Anonymous. 1999. S-PLUS 2000 programmer's guide. Data Analysis Products Division, MathSoft, Seattle, WA. URL: http://www.insightful.com/support/doc_splus_win.asp.

Anonymous. 2003a. Writing R extensions. Version 1.7.1 (June 16, 2003). URL: http://cran.r-project.org/doc/manuals/R-exts.pdf. See also the text file `readme.packages` in the R root directory.

Anonymous. 2003b. Guidelines for Rd files. URL: http://developer.r-project.org/Rds.html. (Document not dated, but accessed September 18, 2003.)

Becker, R.A., J.M. Chambers, and A.R. Wilks. 1988. The new S language: a programming environment for data analysis and graphics. Wadsworth and Books/Cole. Pacific Grove, CA.

Becker, R.A., and A.R. Wilks. 1993. Maps in S. Statistics Research Report 93.2. AT&T Bell Laboratories, Murray Hill, NJ. 21 p. URL: http://www.research.att.com/areas/stat/doc/.

Becker, R.A., and A.R. Wilks. 1995 (rev. 1997). Constructing a geographical database. Statistics Research Report 95.2. AT&T Bell Laboratories, Murray Hill, NJ. 23 p. URL: http://www.research.att.com/areas/stat/doc/.

Devlin, K.J. 1998. The language of mathematics: making the invisible visible. W. H. Freeman and Company. New York, NY. 344 p. (Reference taken from the first paperback printing 2000)

Douglas, D.H., and T.K. Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line of its caricature. Canadian Cartographer 10: 112-122.

Foley, J.D., A. van Dam, S.K. Feiner, and J.F. Hughes. 1996. Computer graphics principles and practice: second edition in C. Addison-Wesley Publishing Co. Boston, MA.

Haigh, R., and J. Schnute. 1999. A relational database for climatological data. Canadian Manuscript Report of Fisheries and Aquatic Sciences 2472. 26 p.

Hains, E. 1994. Point in polygon strategies. Chapter 1.4, p. 24-46 *in*: Heckbert, P.S. 1994. Graphics Gems IV. Academic Press, San Diego, CA. 575 p.

Kernighan, B.W., and D.M. Ritchie. 1978. The C programming language. Prentice-Hall, Inc. Edgewood Cliffs, NJ.

Kernighan, B.W., and D.M. Ritchie. 1988. The C programming language (second edition). Prentice-Hall, Inc. New York, NY. 274 p. See the web sites: http://cm.bell-labs.com/cm/cs/cbook/index.html, http://www.pseudorandom.org/kandr.

Knuth, D.E. 1984. The TeXbook. Addison-Wesley Publishing Company. Reading, MA. 483 p.

Lamport, L. 1986. LaTeX: a document preparation system. Addison-Wesley Publishing Company. Reading, MA. 242 p.

Ousterhout, J.K. 1998. Scripting: higher level programming for the 21st century. Institute of Electrical and Electronics Engineers (IEEE) Computer 31: 23-30. (Currently available at http://home.pacbell.net/ouster/scripting.html)

Ritchie, D.M. 1993. The development of the C language. ACM SIGPLAN Notices 28: 201-208 (ACM HOPL-II Conference). Currently available at: http://cm.bell-labs.com/cm/cs/who/dmr/chist.html, with a related PDF file.

Rokne, J. 1996. The area of a simple polygon. p. 5-6 *in*: Arvo, J. 1996. Graphics Gems II. Academic Press, San Diego, CA. 672 p.

Rossini, A., M. Maechler, K. Hornik, R.M. Heiberger, and R. Sparapani. 2001. Emacs Speaks Statistics: a universal interface for statistical analysis. University of Washington Biostatistics Paper Series, Paper 173. 24 p. (http://www.bepress.com/uwbiostat/paper173)

Rutherford, K.L. 1999. A brief history GFCATCH (1954-1995), the groundfish catch and effort database at the Pacific Biological Station. Canadian Technical Report of Fisheries and Aquatic Sciences 2299. 66 p.

Schnute, J.T., R. Haigh, B.A. Krishka, and P. Starr. 2001. Pacific ocean perch assessment for the west coast of Canada in 2001. Canadian Science Advisory Secretariat (CSAS) Research Document 2001/138. 90 p.

Schnute, J.T., C.G. Wallace, and T.A. Boxwell. 1996. A relational database shell for marked Pacific salmonid data (Revision 1). Canadian Technical Report of Fisheries and Aquatic Sciences 2090A. 28 p.

Sebesta, R. 2002. Concepts of programming languages (5th edition). Addison-Wesley. Boston, MA. 720 p.

Sinclair, C.A., and N. Olsen. 2002. Groundfish research cruises conducted by the Pacific Biological Station, Fisheries and Oceans Canada, 1944-2002. Canadian Manuscript Report of Fisheries and Aquatic Sciences 2617. 91 p.

Sipser, M. 1997. Introduction to the theory of computation. PWS Publishing Company. Boston, MA. 396 p.

Starr, P.J., Krishka, B.A., and Choromanski, E.M. 2002. Trawl survey for thornyhead biomass estimation off the west coast of Vancouver Island, September 15 – October 2, 2001. Canadian Technical Report of Fisheries and Aquatic Sciences 2421. 60 p.

Stroustrup, B. 1985. The C++ programming language. Addison-Wesley Publishing Co. Reading, MA.

van Rossum, G., and L. Wall. 2001. Programming Parrot in a nutshell. O'Reilly and Associates. 401 p. (Announced April 1, 2001 at http://www.oreilly.com/parrot/)

Venables, W.N., and B.D. Ripley. 1999. Modern applied statistics with S-PLUS (3rd Edition). Springer-Verlag. New York, NY. 501 p.

Venables, W.N., and B.D. Ripley. 2000. S programming. Springer-Verlag. New York, 264 p.

Wessel, P., and W.H.F. Smith. 1996. A global, self-consistent, hierarchical, high-resolution shoreline database, Journal of Geophysical Research 101: 8741-8743. URL: http://www.soest.hawaii.edu/pwessel/pwessel_pubs.html.

Whitehead, P., and E. Kramer. 2000. Perl: your visual blueprint for building Perl scripts. IDG Books. Foster City, CA. 350 p.

Wirth, Niklaus. 1975. Algorithms + data structures = programs. Prentice-Hall. Englewood Cliffs, NJ. 366 p.

## APPENDIX A. Distribution CD

This appendix lists the contents of the distribution CD. Its directory structure mirrors the contents of this report, with entries for free Internet software, the PBS Mapping package, and other utilities. We always provide links to web sites for downloading free software. Subject to license restrictions, we also include the versions actually used by us. This allows users to replicate our work, which might need revision for future software releases. If a license prevents us from distributing software, we provide only links to the applicable web sites. Each directory on the CD contains a file named `00ReadMe.txt` or `00ReadMe.pdf` that describes the directory's contents.

**Table A1.** Directories on the distribution CD, with brief descriptions of their contents.

| Path | Description |
|---|---|
| `\FreeInternetSoftware` | Free software downloaded from the Internet |
| `  \Compilers` | Compilers |
| `    \C-C++` | Dev-C++ 4.9.8.0 |
| `    \Fortran` | Fortran included in Dev-C++ 4.9.8.0 |
| `    \Pascal` | Dev-Pascal 1.9.2 |
| `  \ScriptingLanguages` | Scripting language interpreters |
| `    \Parrot` | Parrot history |
| `    \Perl` | *Link to ActivePerl* |
| `    \Python` | Python 2.3 |
| `    \TclTk` | *Link to ActiveTcl* |
| `   \ToolsForWindows` | Various tools for Windows |
| `      \Emacs` | Emacs (text editor) |
| `      \Gimp` | GIMP (image manipulation) |
| `      \HTMLHelpWorkshop` | *Link to Microsoft HTML Help Workshop (compiled help files)* |
| `      \TeX` | *Link to MiKTeX (mathematical typesetting)* |
| `      \UNIXTools` | *Link to Cygwin (UNIX tools)* |
| `\PBSmapping` | PBS Mapping Software for R/S |
| `   \DataSets` | Datasets useful for mapping |
| `   \R` | PBS Mapping R package for distribution |
| `      \PBSmapping` | Directory tree to construct R package, as described in Section 5.6 |
| `   \S` | PBS Mapping S library for distribution |
| `      \PBSmapping` | Directory tree to construct S library, as described in Section 5.7 |
| `   \src` | Source code |

**Table A1.** continued.

| Path | Description |
|---|---|
| `\Utilities` | Utilities programmed at the Pacific Biological Station (PBS) |
|     `\Database` | Database utilities |
|     `\Mapping` | Mapping utilities |
|         `\clipPolys` | Clip polysets |
|         `\convUL` | Convert between UTM and Longitude/Latitude coordinates |
|         `\findPolys` | Find the polygons containing events |
|         `\gshhs2r` | Convert the GSHHS ASCII data format to our ASCII data format |
|     `\OS` | Operating system utilities |
|         `\dusage` | Disk Usage |
|             `\src` | Source code |
|         `\qcd` | Quickly Change Directory |
|             `\src` | Source code |
|         `\recent` | Recently Modified Files |
|             `\src` | Source code |
|         `\setenv` | Set Environment Variables |
|             `\src` | Source code |

## APPENDIX B. Free Software Technical Information

A few tricks can sometimes make the difference between ease and frustration when installing and using a software application. This appendix documents technical information that we found helpful for the applications discussed here.

### Cygwin

The Cygwin installer `setup.exe` (found at http://sources.redhat.com/cygwin/) provides a GUI that downloads and installs Cygwin components via the Internet. The process allows you to choose a download site, and we found reasonable response from a location called "http://mirrors.kernel.org". Accept all default options to obtain a minimal Cygwin working environment. To include other packages, select them from the GUI during installation. Once the installation is complete, add Cygwin's `bin` folder to the system's PATH, if you want the UNIX tools available from all directories.

### Emacs and Emacs Speaks Statistics

Installation requires the two most recent zipped archives for Emacs and ESS. At the time of writing, these are

- `emacs-21.3-fullbin-i386.tar.gz`, available at http://ftp.azc.uam.mx/mirrors/gnu/windows/emacs/latest/;
- `ess-5.1.24.zip`, available at http://www.analytics.washington.edu/Zope/wikis/ess/.

Extract all compressed files from the Emacs archive into a suitable folder (e.g., `C:\Program Files\`), making sure that the 'Use folder names' option is selected. This creates the subfolder `\emacs-21.3`. Next, extract files from the ESS archive to `C:\Program Files\emacs-21.3\`. This creates the subfolder `\ess-5.1.24`.

After extracting the above files, you need to create the file `C:\.emacs` containing the following two lines:

```
(load "C:/progra~1/emacs-21.3/ess-5.1.24/lisp/ess-site")
(setq ess-fancy-comments nil)
```

Adjust the folder location in the first line to reference the path of your installation. Be sure to place the file `.emacs` in the root directory of drive C, regardless of your Emacs installation location. Also, be sure to use forward slashes in the first line of `.emacs`. (Note: Windows Explorer may not allow you to create `.emacs` or rename a file to `.emacs` because there is no string preceding the dot extension. In this case, create a temporary file `e.txt` with `Notepad.exe`, then open a DOS window and type `move c:\e.txt c:\.emacs`.)

You can run Emacs by double clicking `runemacs.exe` in `C:\Program Files\emacs-21.3\bin\runemacs.exe`. Alternatively, if the Emacs' `bin` directory is placed on your PATH, you can issue the command `runemacs` from a DOS window in any directory.

## GSHHS Software

The GSHHS software includes two programs that we use for manipulating data sets. The first (`gshhs.exe`) converts binary data into text format. The second (`gshhs_dp.exe`) thins binary data by reducing the number of points sensibly to produce a smaller binary file. Both programs are distributed as C source code.

For technical reasons related to the binary format, each program can be compiled in two distinct ways: with or without defining a variable `FLIP` on the command line. We compiled two versions of each program with the commands

```
gcc -O3 -o gshhs.exe gshhs.c
gcc -DFLIP -O3 -o gshhs-flip.exe gshhs.c
gcc -O3 -o gshhs_dp.exe gshhs_dp.c
gcc -DFLIP -O3 -o gshhs_dp-flip.exe gshhs_dp.c
```

where the string `-flip` in executable file names indicates that `FLIP` was defined by the argument "`-DFLIP`" during compilation. Working with Windows on an Intel platform, we used `gshhs-flip.exe` and `gshhs_dp-flip.exe` to process original binary data files. By contrast, with binary data files previously thinned by us, we used `gshhs.exe` and `gshhs_dp.exe`. For additional details, please refer to the GSHHS `README` file.

## OpenOffice.org

To install this package, go to http://www.openoffice.org/ and download the latest Windows version (OpenOffice.org 1.1, at the time of writing) from a convenient mirror site. You will obtain a fairly large ZIP file (currently about 65MB). Extract all compressed files to a temporary folder. For the simplest single-user installation, navigate to this folder, and run `setup.exe`. However, we suggest making a common installation of OpenOffice.org available to all users on a workstation. To allow for this, open a command (DOS) window, navigate to the temporary folder using `cd` or `qcd` (Section 3.1), issue the command `setup.exe /net`, and select an installation directory, such as `C:\Utils\OpenOffice`. After this initial workstation-specific installation, each user (including the initial user) can access the program with a further one-time call to the set-up program:

```
C:\Utils\OpenOffice\program\setup.exe
```

Choose the 'Workstation' option rather than the 'Local' option to ensure that each user takes advantage of the common code directory. Each user can specify a distinct directory (such as `C:\Utils\OpenOffice\Jon`) for relatively small local files.

## Parrot

Probably you've already realised that Parrot came into existence as an elaborate April Fools' Day hoax. We included it in this report because we enjoy the joke and find it representative of humour in the hacker community. Languages like Perl and Python have strong advocates who enjoy a friendly rivalry. Consequently, a union of these two approaches seems almost unthinkable, except perhaps on April 1, 2001. With its name inspired by the hoax, a genuine Parrot project now exists to design an interpreter for languages that use dynamic compilation to byte code. In the future, this engine could reside at the core of most scripting languages, giving them a common implementation framework.

## Pascal

Bloodshed Software (http://www.bloodshed.net/) offers a free Integrated Development Environment (IDE) named Dev-Pascal. There are three IDE bundle options (i) no compiler, (ii) the Free Pascal compiler, or (iii) the GNU Pascal compiler. Table B1 summarises the differences between the two compilers. For option (iii), download the file `dev_gnu_pascal-1.9.2.exe` or a more recent version from http://www.bloodshed.net/devpascal.html. Run this program, and follow the on-screen instructions.

**Table B1.** Chief differences between the Free Pascal and GNU Pascal compilers, abstracted from information at http://www.freepascal.org/faq.html#FPandGNUPascal (section 1.3).

| | **Free Pascal** | **GNU Pascal** |
|---|---|---|
| Goal | To implement a Borland compatible Pascal compiler on as many platforms as possible. | To implement a standards-based portable Pascal compiler based on POSIX. |
| Operating Systems | Linux, FreeBSD, NetBSD, DOS, Win32, OS/2, BeOS, SunOS (Solaris), QNX and Classic Amiga; for the moment limited to the Intel and Motorola architectures. | Any system that can run the GNU C compiler. |
| Compiler Source | Written in Pascal. | Written in C. |
| Language | Borland Pascal dialect and implements the Delphi Object Pascal language. | Supports ISO 7185, ISO/IEC 10206 (most), ANSI/IEEE 770X3.160, Borland Pascal 7.0 (most). |
| Extensions | Method, function, and operator overloading. | Operator overloading. |

## TeX

MiKTeX, a free implementation of TeX and related programs, provides utilities to typeset TeX and LaTeX documents. We used MiKTeX to typeset Appendix D of this report and convert it to a PDF file. To install MiKTeX, follow these steps:

1. Download the installation program (`setup.exe`) from http://www.miktex.org/setup.html.

2. Review the installation procedure in HTML or PDF format, available from a link on the home page. Currently, this points to: http://www.ctan.org/tex-archive/systems/win32/miktex/setup/install.html. The file `install.pdf` gives a concise summary.
3. Run `setup.exe` to download the required MiKTeX packages to a local directory, which serves as a package repository. We have found the 'Small' package set adequate for routine typesetting. This involves about 100 compressed `cab` files, and the final installation occupies over 100 MB on the hard drive. Add `setup.exe` to your repository directory, and (optionally) include `install.pdf` for convenient reference.
4. Run `setup.exe` again from the repository to complete the installation. Again, we suggest choosing the 'Small' package set. Install MiKTeX for everyone (all users) on the workstation, and accept the path to the repository.
5. You need to specify an installation path (e.g., `C:\Utils\MiKTeX`) and another optional path for storing local font information (e.g., `C:\Utils\MiKLocal`). Ensure that no paths involve the space character (as in `\Program Files\`). Another name (e.g., `MiKTeX`) specifies the folder in the Program menu, reached via the Start menu.
6. The installation procedure automatically puts the MiKTeX `bin` folder (e.g., `C:\Utils\MiKTeX\MiKTeX\bin`) in first place on the PATH. You may want to demote this placement.
7. The PDF utilities normally default to European A4 paper, rather than North American letter size. Change this by modifying the text file `pdftex.cfg`, which (assuming the paths in step 5) lies in the folder `C:\Utils\MiKTeX\pdftex\config`. Explicitly
   - replace `page_width 210 true mm` with `page_width 8.5 true in`
   - replace `page_height 297 true mm` with `page_height 11 true in`
8. MiKTeX automatically creates the folder `C:\texmf\`, apparently to store log files for reporting bugs to the developers.

The MiKTeX package includes numerous utilities for typesetting, viewing, and converting to PDF files. In particular

- `tex.exe` typesets a plain TeX file (`*.tex`) to produce a device-independent (DVI) binary file (`*.dvi`);
- `latex.exe` typesets a LaTeX file (`*.tex`) to produce a DVI file;
- `yap.exe`, "yet another previewer", visually displays a DVI file;
- `pdftex.exe` converts a plain TeX file (`*.tex`) to a PDF file;
- `pdflatex.exe` converts a LaTeX file (`*.tex`) to a PDF file.

Unlike some commercial alternatives, MiKTeX does not provide a graphical user interface (GUI) for editing, typesetting, and viewing TeX files. However, the AUCTeX extension adds support for these actions to the Emacs editor. You can download AUCTeX from the web site http://www.gnu.org/software/auctex/ by following the link to the latest 'stable version'. An additional Emacs extension, called preview-latex, gives typeset previews of selected lines in the LaTeX source file. Download the latest version from the web site http://preview-latex.sourceforge.net/ by navigating to the 'download page'. Both AUCTeX and preview-latex have complete documentation on their respective web sites.

## APPENDIX C. PBS Mapping Function Dependencies

This appendix documents function dependencies within the PBS Mapping package. All functions appear as underlined entries in the alphabetised list. If a function depends on others, the list of dependencies appears below the underlined name. Following a standard in UNIX and R, functions whose name begins with a period (*dot functions*) are considered hidden from the user, who would normally use only the non-hidden functions that call them. The names here apply primarily to the R/S working environment, but functions designated '**(C)**' are implemented in C source code and compiled in the DLL for the mapping package. These must be invoked from R/S with the call `.C(…)`.

.addAxis

.addLabels

.checkProjection

.clip
.validatePolySet
clip **(C)**

.createPolyProps
.validatePolyData

.fixGSHHSWorld
findPolys
fixPOS

.initPlotRegion

.integrateHoles
.validatePolySet
integrateHoles **(C)**

.plotMaps
.addAxis
.addLabels
.initPlotRegion
.validatePolySet
addLines
addPolys

.validateData

.validateEventData
.validateData

.validatePolyData
.validateData

.validatePolySet
.validateData

.validateXYData
.validateData

addLines
.checkProjection
.createPolyProps
.validatePolySet
clipLines

addPolys
.checkProjection
.createPolyProps
.integrateHoles
.validatePolySet
clipPolys

calcArea
.validatePolySet
calcArea **(C)**

clipLines
.clip

clipPolys
.clip

closePolys
.validatePolySet
closePolys **(C)**

combineEvents
.validateEventData

convUL
.validateXYData
convUL **(C)**

findPolys
.validateEventData
.validatePolySet
locateEvents **(C)**

fixBound
.validatePolySet

fixPOS
.validatePolySet
fixPOS **(C)**

locateEvents

locatePolys
.validatePolyData

makeGrid

makeProps
.validatePolyData

plotLines
.plotMaps

plotMap
.plotMaps

plotPolys
.plotMaps

## APPENDIX D. PBS Mapping Functions and Data

This appendix documents the R/S objects available in PBS Mapping, as summarised in Table D1. Subsequent pages contain complete technical documentation for every object, listed in alphabetical order. These descriptions come from `.Rd` files written for the R documentation system, which generates corresponding LaTeX files that typeset the pages here.

**Table D1.** Functions and data sets defined in the R/S PBS Mapping package, arranged alphabetically within categories.

| Category | Object | Description |
|---|---|---|
| User constant | `LANG` | Software environment (`"R"` or `"S"`) |
| Plotting functions | `addLines` | Add a PolySet to an existing plot as polylines |
| | `addPolys` | Add a PolySet to an existing plot as polygons |
| | `plotLines` | Plot a PolySet as polylines |
| | `plotMap` | Plot a PolySet as a map |
| | `plotPolys` | Plot a PolySet as polygons |
| Computational functions | `calcArea` | Calculate areas of polygons |
| | `clipLines` | Clip a PolySet as polylines |
| | `clipPolys` | Clip a PolySet as polygons |
| | `closePolys` | Close a PolySet |
| | `combineEvents` | Combine measurements of events |
| | `convUL` | Convert between UTM and LL projections |
| | `findPolys` | Find polygons that contain events |
| | `fixBound` | Fix boundary points of a PolySet |
| | `fixPOS` | Fix the POS column of a PolySet |
| | `locateEvents` | Locate events on the current plot |
| | `locatePolys` | Locate polygons on the current plot |
| | `makeGrid` | Make a grid PolySet |
| | `makeProps` | Make polygon properties |
| Data sets | `nepacLL` | Northeast Pacific shoreline (normal resolution) |
| | `nepacLLhigh` | Northeast Pacific shoreline (high resolution) |
| | `pythagoras` | Pythagoras theorem diagram |
| | `surveyData` | Survey data |
| | `towData` | Tow data |
| | `towTracks` | Tow track polylines |
| | `worldLL` | World ocean shoreline (normal resolution) |
| | `worldLLhigh` | World ocean shoreline (high resolution) |

## `addLines` *Add a PolySet to an Existing Plot as Polylines*

### Description

Adds a PolySet to an existing plot, where each (`PID`, `SID`) set describes a unique polyline.

### Usage

```
addLines (polys, xlim = NULL, ylim = NULL,
    polyProps = NULL, lty = NULL, col = NULL, ...)
```

### Arguments

| | |
|---|---|
| `polys` | PolySet to add (*required*). |
| `xlim` | vector range of x-values. |
| `ylim` | vector range of y-values. |
| `polyProps` | PolyData specifying which polylines to plot and their properties. The data in this object are superseded by plot properties passed as direct arguments. |
| `lty` | numeric or character (R only) vector describing line types (cycled by `PID`). |
| `col` | numeric or character (R only) vector describing colours (cycled by `PID`). |
| `...` | additional `par` parameters for the `lines` function. |

### Details

The plotting routine does not connect the last vertex of each discrete polyline to the first vertex of that polyline. It clips `polys` to `xlim` and `ylim` before plotting.

### See Also

`addPolys`, `plotLines`, `plotMap`, and `plotPolys`.

### Examples

```
#--- create a PolySet to plot
polys <- data.frame(PID=rep(1, 4), POS=1:4, X=c(0, 1, 1, 0), Y=c(0, 0, 1, 1))
#--- plot the PolySet
plotLines(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5))
#--- add the PolySet to the plot (in a different style)
addLines(polys, lwd=5, col=3)
```

## `addPolys` *Add a PolySet to an Existing Plot as Polygons*

### Description

Adds a PolySet to an existing plot, where each (`PID`, `SID`) set describes a unique polygon.

### Usage

```
addPolys (polys, xlim = NULL, ylim = NULL, polyProps = NULL,
    border = NULL, lty = NULL, col = NULL, density = NA,
    angle = NULL, ...)
```

### Arguments

| | |
|---|---|
| `polys` | PolySet to add (*required*). |
| `xlim` | vector range of x-values. |
| `ylim` | vector range of y-values. |
| `polyProps` | PolyData specifying which polygons to plot and their properties. The data in this object are superseded by plot properties passed as direct arguments. |
| `border` | numeric or character (R only) vector describing colours for the edges of polygons (cycled by `PID`). |
| `lty` | numeric or character (R only) vector describing line types (cycled by `PID`). |
| `col` | numeric or character (R only) vector describing fill colours (cycled by `PID`). |
| `density` | numeric vector describing shading line densities (lines per inch, cycled by `PID`). |
| `angle` | numeric vector describing shading line angles (degrees, cycled by `PID`). |
| `...` | additional `par` parameters for the `polygon` function. |

### Details

The plotting routine connects the last vertex of each discrete polygon to the first vertex of that polygon. It supports both borders (`border`, `lty`) and fills (`col`, `density`, `angle`). It clips `polys` to `xlim` and `ylim` before plotting.

### See Also

`addLines`, `plotLines`, `plotMap`, and `plotPolys`.

## Examples

```
#--- create a PolySet to plot
polys <- data.frame(PID=rep(1, 4), POS=1:4, X=c(0, 1, 1, 0), Y=c(0, 0, 1, 1))
#--- plot the PolySet
plotPolys(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5), density=0)
#--- add the PolySet to the plot (in a different style)
addPolys(polys, col=3)
```

---

`calcArea` *Calculate the Areas of Polygons*

---

## Description

Calculates the areas of polygons found in a PolySet.

## Usage

```
calcArea (polys, sumPID = FALSE)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to use. |
| `sumPID` | Boolean value; if `TRUE`, sum across `SID`s within each `PID`. |

## Details

If `SID` exists in the input data and `sumPID = FALSE`, then the output contains a row for each (`PID`, `SID`) where `SID` corresponds to an outer boundary. If `sumPID = TRUE`, then the output contains a row for each `PID`.

## Value

PolyData with columns `PID`, `SID` (*may be missing*), and `area`.

## Examples

```
#--- load the data (if using R)
if (exists("LANG") && LANG == "R")
  data(nepacLL)
#--- convert LL to UTM so calculation makes sense
attr(nepacLL, "zone") <- 9
nepacUTM <- convUL(nepacLL)
#--- calculate and print the areas
print(calcArea(nepacUTM))
```

## `clipLines` *Clip a PolySet as Polylines*

### Description

Clips a PolySet, where each (`PID`, `SID`) set describes a unique polyline.

### Usage

```
clipLines (polys, xlim, ylim)
```

### Arguments

| | |
|---|---|
| `polys` | PolySet to clip. |
| `xlim` | vector range of x-values. |
| `ylim` | vector range of y-values. |

### Details

For each discrete polyline, vertices 1 and N are not connected. The `POS` values for each vertex will be recalculated, with the old values saved in a column named `oldPOS`. If a new vertex is added, its `oldPOS` value will be `NA`.

### Value

A new data frame containing the input data, with some points added or removed. A new column `oldPOS` records the original `POS` value for each vertex.

### See Also

`clipPolys`.

### Examples

```
#--- create a triangle to clip
polys <- data.frame(PID=rep(1, 3), POS=1:3, X=c(0,1,0), Y=c(0,0.5,1))
#--- clip the triangle in the X direction, and plot the results
plotLines(clipLines(polys, xlim=c(0,.75), ylim=range(polys[, "Y"])))
```

## `clipPolys` *Clip a PolySet as Polygons*

### Description

Clips a PolySet, where each (`PID`, `SID`) set describes a unique polygon.

### Usage

```
clipPolys (polys, xlim, ylim)
```

### Arguments

| | |
|---|---|
| `polys` | PolySet to clip. |
| `xlim` | vector range of x-values. |
| `ylim` | vector range of y-values. |

### Details

For each discrete polygon, vertices 1 and N are connected. The `POS` values for each vertex will be recalculated, with the old values saved in a column named `oldPOS`. If a new vertex is added, its `oldPOS` value will be `NA`.

### Value

A new data frame containing the input data, with some points added or removed. A new column `oldPOS` records the original `POS` value for each vertex.

### See Also

`clipLines`.

### Examples

```
#--- create a triangle that will be clipped
polys <- data.frame(PID=rep(1, 3), POS=1:3, X=c(0,1,.5), Y=c(0,0,1))
#--- clip the triangle in the X direction, and plot the results
plotPolys(clipPolys(polys, xlim=c(0,.75), ylim=range(polys[, "Y"])))
```

## `closePolys` *Close a PolySet*

### Description

Closes a PolySet of polylines to form polygons.

### Usage

```
closePolys (polys)
```

### Arguments

| | |
|---|---|
| `polys` | PolySet to close. |

### Details

Generally, run `fixBound` before this function. The ranges of a PolySet's `X` and `Y` columns define the boundary. For each discrete polygon, this function determines if the first and last points lie on a boundary. If both points lie on the same boundary, it adds no points. However, if both points lie on different boundaries, one or two corners may be added to the polygon.

When the boundaries are adjacent, one corner will be added as follows:

- top boundary + left boundary implies add top-left corner;
- top boundary + right boundary implies add top-right corner;
- bottom boundary + left boundary implies add bottom-left corner;
- bottom boundary + right boundary implies add bottom-right corner.

When the boundaries are opposite, first add the corner closest to a starting or ending polygon vertex. This determines a side (left-right or bottom-top) that connects the opposite boundaries. Then, add the other corner of that side to close the polygon.

### Value

A data frame identical to `polys`, except for possible additional corner points.

### See Also

`fixBound` and `fixPOS`.

### Examples

```
#--- 4 corners
polys <- data.frame(PID=c(1, 1, 2, 2, 3, 3, 4, 4),
                    POS=c(1, 2, 1, 2, 1, 2, 1, 2),
                    X = c(0, 1, 2, 3, 0, 1, 2, 3),
                    Y = c(1, 0, 0, 1, 2, 3, 3, 2))
plotPolys(closePolys(polys))

#--- 2 corners and 1 opposite
polys <- data.frame(PID=c(1, 1, 2, 2, 3, 3, 3),
                    POS=c(1, 2, 1, 2, 1, 2, 3),
                    X = c(0, 1, 0, 1, 5, 6, 1.5),
                    Y = c(1, 0, 2, 3, 0, 1.5, 3))
plotPolys(closePolys(polys))
```

---

## `combineEvents` *Combine Measurements of Events*

---

### Description

Combines measurements associated with events that occur in the same polygon.

### Usage

```
combineEvents (events, locs, FUN = mean, bdryOK = TRUE)
```

### Arguments

| | |
|---|---|
| `events` | EventData with at least four columns (`EID`, `X`, `Y`, `Z`). |
| `locs` | LocationSet usually resulting from a call to `findPolys`. |
| `FUN` | a function that produces a scalar from a vector (e.g., `mean`, `sum`). |
| `bdryOK` | Boolean variable that determines whether or not to include boundary points. |

### Details

Combines measurements associated with events that occur in the same polygon. Each event (`EID`) has a corresponding measurement `Z`. The `locs` data frame (usually output from `findPolys`) places events within polygons. Thus, each polygon (`PID`, `SID`) determines a set of events within it, and a corresponding vector of measurements `Zv`. The function returns `FUN(Zv)`, a summary of measurements within each polygon.

### Value

PolyData with columns `PID`, `SID` (*if in* `locs`), and `Z`.

## See Also

`findPolys`, `locateEvents`, `locatePolys`, `makeGrid`, and `makeProps`.

## Examples

```
#--- create an EventData data frame: let each event have Z = 1
events <- data.frame(EID=1:10, X=1:10, Y=1:10, Z=rep(1, 10))
#--- example output from findPolys where 1 event occurred in the first
#--- polygon, 3 in the second, and 6 in the third
locs <- data.frame(EID=1:10, PID=c(rep(1, 1), rep(2, 3), rep(3, 6)),
                   Bdry=rep(0, 10))
#--- sum the Z column of the events in each polygon, and print the result
print(combineEvents(events=events, locs=locs, FUN=sum))
```

---

`convUL` *Convert Coordinates between UTM and Lon/Lat*

---

## Description

Converts coordinates between UTM and Lon/Lat.

## Usage

```
convUL (xydata)
```

## Arguments

|  |  |
|---|---|
| `xydata` | matrix/data frame (e.g., a PolySet or EventData) with columns `X` and `Y`. |

## Details

`xydata` must have two attributes: `projection` equal to `"LL"` or `"UTM"`, and `zone` equal to a number from 1 to 60. The `projection` attribute describes the current projection of the `xydata`, and the `zone` attribute describes the UTM zone that the data are in or will be in. If any of these attributes are missing or have inappropriate values, the function stops. The `X` and `Y` columns of `xydata` will be converted from `"LL"` to `"UTM"` or vice-versa. After the conversion, the data frame's attributes will be adjusted accordingly.

## Value

A data frame identical to `xydata`, except that the `X` and `Y` columns will contain the results of the conversion.

## Examples

```
#--- load the data (if using R)
data(nepacLL)
#--- ensure the attributes are set
attr(nepacLL, "projection") <- "LL"
attr(nepacLL, "zone") <- 9
#--- perform the conversion
nepacUTM <- convUL(nepacLL)
```

---

`findPolys` *Find the Polygons that Contain Events*

---

## Description

Find the polygons in a PolySet that contain events specified by EventData.

## Usage

```
findPolys (events, polys)
```

## Arguments

| | |
|---|---|
| `events` | EventData to use. |
| `polys` | PolySet to use. |

## Details

The resulting data frame, a LocationSet, contains the columns `EID`, `PID`, `SID` (*if in* `polys`), and `Bdry`, where an event (`EID`) occurs in a polygon (`PID`, `SID`) and `SID` does not correspond to an inner boundary. The Boolean variable `Bdry` specifies if an event lies on a polygon's edge. Note that if an event lies properly outside of all the polygons, then a record with (`EID`, `PID`, `SID`) does not occur in the output. It may happen, however, that an event occurs in multiple polygons. Thus, the same `EID` can occur more than once in the output.

## Value

A LocationSet.

## See Also

`combineEvents`, `locateEvents`, `locatePolys`, `makeGrid`, and `makeProps`.

## Examples

```
#--- create some EventData: a column of points at X = 0.5
events <- data.frame(EID=1:10, X=.5, Y=seq(0, 2, length=10))
#--- create a PolySet: two squares with the second above the first
polys <- data.frame(PID=c(rep(1, 4), rep(2, 4)), POS=c(1:4, 1:4),
                    X=c(0, 1, 1, 0, 0, 1, 1, 0),
                    Y=c(0, 0, 1, 1, 1, 1, 2, 2))
#--- run findPolys and print the results
print(findPolys(events, polys))
```

---

`fixBound` *Fix the Boundary Points of a PolySet*

---

## Description

The ranges of a PolySet's `X` and `Y` columns define its boundary. This function fixes a PolySet's vertices by moving vertices near a boundary to the actual boundary.

## Usage

```
fixBound (polys, tol)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to fix. |
| `tol` | numeric vector (length 1 or 2) specifying a percentage of the ranges to use in defining 'near' to a boundary. If `tol` has two elements, the first specifies the tolerance for the x-axis and the second the y-axis. |

## Value

A data frame identical to the input, except for possible changes in the `X` and `Y` columns.

## See Also

`closePolys` and `fixPOS`.

## Examples

```
#--- set up a long horizontal and long vertical line to extend the plot's
#--- limits, and then try fixing the bounds of a line in the top-left
#--- corner and a line in the bottom-right corner
polys <- data.frame(PID=c(1, 1, 2, 2,  3, 3, 4, 4),
                    POS=c(1, 2, 1, 2,  1, 2, 1, 2),
                    X = c(0, 10, 5, 5, 0.1, 4.9, 5.1, 9.9),
                    Y = c(5, 5, 0, 10, 5.1, 9.9, 0.1, 4.9))
polys <- fixBound(polys, tol=0.0100001)
plotLines(polys)
```

## fixPOS — *Fix the POS Column of a PolySet*

### Description

Fixes the `POS` column of a PolySet by renumbering it using sequential integers.

### Usage

```
fixPOS (polys)
```

### Arguments

| | |
|---|---|
| `polys` | PolySet to fix. |

### Details

The `POS` values of each (`PID`, `SID`) set are renumbered as either 1 to N or N to 1, depending on the order of `POS` (ascending or descending) in the input data. `POS` values in the input must be properly ordered (ascending or descending), but they may contain fractional values. For example, `POS = 2.5` might correspond to a point manually added between `POS = 2` and `POS = 3`. All other columns remain unchanged.

### Value

A data frame with the same columns as the input, except for possible changes to the `POS` column.

### See Also

`closePolys` and `fixBound`.

### Examples

```
#--- create a PolySet with broken POS numbering
polys <- data.frame(PID = c(rep(1, 10), rep(2, 10)),
                    POS = c(seq(2, 10, length = 10), seq(10, 2, length = 10)),
                    X = c(rep(1, 10), rep(1, 10)),
                    Y = c(rep(1, 10), rep(1, 10)))
#--- fix the POS numbering
polys <- fixPOS(polys)
#--- print the results
print(polys)
```

---

`LANG` *Specify the Software's Environment*

---

### Description

Specifies the language (R/S) in which the mapping software will run.

### Usage

```
LANG
```

### Details

If `LANG = "R"`, the mapping software will run in the R interpreter. If `LANG = "S"`, it will run in the S-PLUS interpreter.

### Value

`"R"` or `"S"`, depending on which environment the software will run in.

---

`locateEvents` *Locate Events on the Current Plot*

---

### Description

Locates events on the current plot (using the `locator` function).

### Usage

```
locateEvents (EID, n = 512, type = "p", ...)
```

### Arguments

| | |
|---|---|
| `EID` | vector of event IDs (*optional*). |
| `n` | maximum number of points to locate. |
| `type` | one of `"n"`, `"p"`, `"l"`, or `"o"`. If `"p"` or `"o"`, then the points are plotted; if `"l"` or `"o"`, then the points are joined by lines. |
| `...` | additional `par` parameters for the `locator` function. |

### Details

Allows the user to define events with mouse clicks on the current plot via the `locator` function. The arguments `n` and `type` are the usual parameters for the `locator` function. If `EID` is supplied, then `n = length(EID)`.

On exit from `locator`, suppose the user defined *m* points. If `EID` was not specified, then the output data frame will contain *m* events. However, if `EID` was specified, then the output data frame will contain `length(EID)` events, and both `X` and `Y` will be `NA` for events `EID[(`*m*`+1):n]`.

### Value

EventData with columns `EID`, `X`, and `Y`.

### See Also

`combineEvents`, `findPolys`, `locatePolys`, `makeGrid`, and `makeProps`.

### Examples

```
#--- define five events on the current plot, numbering them 10 to 14
events <- locateEvents(EID = 10:14)
```

---

## `locatePolys` *Locate Polygons on the Current Plot*

---

### Description

Locates polygons on the current plot (using the `locator` function).

### Usage

```
locatePolys (pdata, n = 512, type = "o", ...)
```

### Arguments

| | |
|---|---|
| `pdata` | PolyData (*optional*) with columns `PID` and `SID` (*optional*), with two more optional columns `n` and `type`. |
| `n` | maximum number of points to locate. |
| `type` | one of `"n"`, `"p"`, `"l"`, or `"o"`. If `"p"` or `"o"`, then the points are plotted; if `"l"` or `"o"`, then the points are joined by lines. |
| `...` | additional `par` parameters for the `locator` function. |

### Details

Allows the user to define polygons with mouse clicks on the current plot via the `locator` function. The arguments `n` and `type` are the usual parameters for the `locator` function, but they may be individually specified for each (`PID`, `SID`) set in a `pdata` object.

If a `pdata` object is supplied, columns other than `PID`, `SID`, `n`, and `type` are ignored. If `pdata` includes `n`, then an outer boundary has `n > 0` and an inner boundary has `n < 0`.

On exit from the `locator` function, suppose the user defined $m$ vertices for a given discrete polygon. For that polygon, the `X` and `Y` columns will contain `NA`s where `POS = (`$m$`+1):n` for outer-boundaries and `POS = (|n|-`$m$`):1` for inner-boundaries.

If a `pdata` object is not supplied, the output has only one polygon with a `PID` equal to 1. One inner-boundary polygon (`POS` goes from `n` to `1`) can be generated by supplying a negative `n`.

If `type = "o"` or `type = "l"`, a line connecting the last point back to the first point is drawn.

### Value

A PolySet.

### See Also

`combineEvents`, `findPolys`, `locateEvents`, `makeGrid`, and `makeProps`.

### Examples

```
#--- define one polygon with up to 5 vertices on the current plot
events <- locatePolys(n = 5)
```

---

## `makeGrid` *Make a Grid of Polygons*

---

### Description

Makes a grid of polygons, using `PID`s and `SID`s according to the input arguments.

### Usage

```
makeGrid (x, y, byrow = TRUE, addSID = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | vector of X coordinates (of length $m$). |
| `y` | vector of Y coordinates (of length $n$). |
| `byrow` | Boolean; if `TRUE`, increment `PID` along X. |
| `addSID` | Boolean; if `TRUE`, include an `SID` column in the resulting PolySet. |

### Details

Makes a grid of polygons, labeling those polygons according to `byrow` and `addSID`.

- `byrow` = `TRUE` and `addSID` = `FALSE` implies `PID` $= i + (j - 1) \times m$
- `byrow` = `FALSE` and `addSID` = `FALSE` implies `PID` $= j + (i - 1) \times n$
- `byrow` = `TRUE` and `addSID` = `TRUE` implies `PID` $= i$, `SID` $= j$
- `byrow` = `FALSE` and `addSID` = `TRUE` implies `PID` $= j$, `SID` $= i$

### Value

A PolySet with columns `PID`, `SID` (*if* `addSID = TRUE`), `POS`, `X`, and `Y`. The PolySet is a set of rectangular grid cells with vertices:
$(x_i, y_j), (x_{i+1}, y_j), (x_{i+1}, y_{j+1}), (x_i, y_{j+1})$.

### See Also

`combineEvents`, `findPolys`, `locateEvents`, `locatePolys`, and `makeProps`.

### Examples

```
#--- make a 10 x 10 grid
polyGrid <- makeGrid(x=0:10, y=0:10)
#--- plot the grid
plotPolys(polyGrid, density=0)
```

---

## `makeProps` *Make Polygon Properties*

---

### Description

Appends a column for a polygon property (e.g., `border` or `lty`) to PolyData based on measurements in its `Z` column.

### Usage

```
makeProps (pdata, breaks, propName="col", propVals=1:(length(breaks)-1))
```

### Arguments

| | |
|---|---|
| `pdata` | PolyData with a `Z` column. |
| `breaks` | either a vector of cut points or a scalar denoting the number of intervals to cut `Z`. |
| `propName` | name of the new column to append to `pdata`. |
| `propVals` | vector of values to associate with `Z` breaks. |

### Details

Acts like the `cut` function to produce PolyData suitable for the `polyProps` plotting argument (see `plotMap`, `plotLines`, `plotPolys`, `addLines`, and `addPolys`). The `Z` column of `pdata` is equivalent to the data vector `x` of the `cut` function.

### Value

PolyData with the same columns as `pdata` plus an additional column `propName`.

### See Also

`combineEvents`, `findPolys`, `locateEvents`, `locatePolys`, and `makeGrid`.

### Examples

```
#--- create a PolyData object
pd <- data.frame(PID=1:10, Z=1:10)

#--- using 3 intervals, create a column named ‘col’ and populate it with
#--- the supplied values
makeProps(pdata = pd, breaks = 3, propName = "col",
          propVals = c(1:3))
```

---

## `nepacLL` *Shoreline of the Northeast Pacific Ocean (Normal Resolution)*

---

### Description

PolySet of polygons for the northeast Pacific Ocean shoreline.

### Usage

```
data(nepacLL)
```

### Format

Data frame consisting of 4 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polygon, `X` = longitude coordinates, and `Y` = latitude coordinates.

### Note

In R, the data must be loaded using the `data` function. In S, the data are loaded automatically on demand.

### Source

Polygon data from the GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline) database. We thinned the full resolution GSHHS binary data with a tolerance of 0.2 km and then converted it to ASCII data using the supplied software (`gshhs_dp.exe` and `gshhs.exe`, respectively). Using a Perl script (`gshhs2r.pl`), we then converted this ASCII format to ours, removed the lakes, islands in lakes, and ponds in islands, and filtered out any islands with fewer than 15 vertices. From our format, we then clipped the data to $170° \leq x \leq 250°$ and $34° \leq y \leq 72°$ (the GSHHS coordinates roughly span 0° to 360°). Finally, we subtracted 360° from all of the remaining $x$-values, so that the Greenwich meridian becomes 0° and the northeast Pacific Ocean has negative $x$-values corresponding to latitudes west of 0°.

### References

Wessel, P., and W.H.F. Smith. 1996. A global, self-consistent, hierarchical, high-resolution shoreline database, Journal of Geophysical Research 101: 8741-8743.
`http://www.soest.hawaii.edu/pwessel/pwessel_pubs.html`

### See Also

`nepacLLhigh`, `pythagoras`, `towData`, `towTracks`, `surveyData`, `worldLL`, and `worldLLhigh`.

---

## `nepacLLhigh` *Shoreline of the Northeast Pacific Ocean (High Resolution)*

---

### Description

PolySet of polygons for the northeast Pacific Ocean shoreline.

### Usage

```
data(nepacLLhigh)
```

### Format

Data frame consisting of 4 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polygon, `X` = longitude coordinates, and `Y` = latitude coordinates.

### Note

In R, the data must be loaded using the `data` function. In S, the data are loaded automatically on demand.

### Source

Polygon data from the GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline) database. We thinned the full resolution GSHHS binary data with a tolerance of 0.1 km and then converted it to ASCII data using the supplied software (`gshhs_dp.exe` and `gshhs.exe`, respectively). Using a Perl script (`gshhs2r.pl`), we then converted this ASCII format to ours and removed the lakes, islands in lakes, and ponds in islands. From our format, we then clipped the data to $170° \leq x \leq 250°$ and $34° \leq y \leq 72°$ (the GSHHS coordinates roughly span 0° to 360°). Finally, we subtracted 360° from all of the remaining $x$-values, so that the Greenwich meridian becomes 0° and the northeast Pacific Ocean has negative $x$-values corresponding to latitudes west of 0°.

### References

Wessel, P., and W.H.F. Smith. 1996. A global, self-consistent, hierarchical, high-resolution shoreline database, Journal of Geophysical Research 101: 8741-8743.
`http://www.soest.hawaii.edu/pwessel/pwessel_pubs.html`

### See Also

`nepacLL`, `pythagoras`, `towData`, `towTracks`, `surveyData`, `worldLL`, and `worldLLhigh`.

---

## `plotLines` *Plot a PolySet as Polylines*

---

### Description

Plots a PolySet as polylines.

### Usage

```
plotLines (polys, xlim = NULL, ylim = NULL,
    plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL, lty = NULL,
    col = NULL, main = "Map", xlab = NULL, ylab = NULL,
    axes = TRUE, tckLab = TRUE, tck = 0.014, tckMinor = 0.5 * tck, ...)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to plot (*required*). |
| `xlim` | vector range of x-values. |
| `ylim` | vector range of y-values. |
| `plt` | four element numeric vector `(x1, x2, y1, y2)` giving the coordinates of the plot region measured as a fraction of the figure region. |
| `polyProps` | PolyData specifying which polylines to plot and their properties. The data in this object are superseded by plot properties passed as direct arguments. |
| `lty` | numeric or character (R only) vector describing line types (cycled by `PID`). |
| `col` | numeric or character (R only) vector describing colours (cycled by `PID`). |
| `main` | string title for the plot. |
| `xlab` | string caption for the x-axis. |
| `ylab` | string caption for the y-axis. |
| `axes` | Boolean value; if `TRUE`, plot axes. |
| `tckLab` | Boolean vector (length 1 or 2) indicating whether or not to label the major tick marks. If a two-element vector is given, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| `tck` | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If `tckLab = TRUE`, these tick marks will be automatically labelled. If a two-element vector is given, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| `tckMinor` | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks may not be automatically labelled. If a two-element vector is given, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| `...` | additional `par` parameters for the `lines` function. |

## Details

Plots a PolySet, where each (`PID`, `SID`) set describes a unique polyline. To do so, it does not connect the last vertex of each set to the first vertex of that set. This function ignores the aspect ratio. Furthermore, it clips `polys` to `xlim` and `ylim` before plotting.

## See Also

`addLines`, `addPolys`, `plotMap`, and `plotPolys`.

## Examples

```
#--- create a PolySet to plot
polys <- data.frame(PID=rep(1, 4), POS=1:4, X=c(0, 1, 1, 0), Y=c(0, 0, 1, 1))
#--- plot the PolySet
plotLines(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5))
```

---

`plotMap` *Plot a PolySet as a Map*

---

## Description

Plots a PolySet as a map, using the correct aspect ratio.

## Usage

```
plotMap (polys, xlim = NULL, ylim = NULL,
    plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL, border = NULL,
    lty = NULL, col = NULL, density = NA, angle = NULL, bgCol = NULL,
    main = "Map", xlab = NULL, ylab = NULL, axes = TRUE, tckLab = TRUE,
    tck = 0.014, tckMinor = 0.5 * tck, ...)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to plot (*required*). |
| `xlim` | vector range of x-values. |
| `ylim` | vector range of y-values. |
| `plt` | four element numeric vector `(x1, x2, y1, y2)` giving the coordinates of the plot region measured as a fraction of the figure region. |
| `polyProps` | PolyData specifying which polygons to plot and their properties. The data in this object are superseded by plot properties passed as direct arguments. |
| `border` | numeric or character (R only) vector describing colours for the edges of polygons (cycled by `PID`). |
| `lty` | numeric or character (R only) vector describing line types (cycled by `PID`). |
| `col` | numeric or character (R only) vector describing fill colours (cycled by `PID`). |
| `density` | numeric vector describing shading line densities (lines per inch, cycled by `PID`). |
| `angle` | numeric vector describing shading line angles (degrees, cycled by `PID`). |
| `bgCol` | numeric or character (R only) scalar to colour the background of the plot. |
| `main` | string title for the plot. |
| `xlab` | string caption for the x-axis. |

| | |
|---|---|
| `ylab` | string caption for the y-axis. |
| `axes` | Boolean value; if `TRUE`, plot axes. |
| `tckLab` | Boolean vector (length 1 or 2) indicating whether or not to label the major tick marks. If a two-element vector is given, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| `tck` | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If `tckLab = TRUE`, these tick marks will be automatically labelled. If a two-element vector is given, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| `tckMinor` | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smaller of the width or height of the plotting region. These tick marks may not be automatically labelled. If a two-element vector is given, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| `...` | additional `par` parameters for the `polygon` function. |

## Details

Plots a PolySet, where each (`PID`, `SID`) set describes a unique polygon. To do so, it connects the last vertex of each set to the first vertex of that set. This function supports both borders (`border`, `lty`) and fills (`col`, `density`, `angle`). It can also be used to draw only borders or only fills by supplying the appropriate arguments. Unlike `plotLines` and `plotPolys`, this function uses the aspect ratio supplied by the `projection` attribute of `polys`. In the absence of this attribute, it uses a default aspect ratio of 1:1. Finally, it clips `polys` to `xlim` and `ylim` before plotting.

## See Also

`addLines`, `addPolys`, `plotLines`, and `plotPolys`,

## Examples

```
#--- create a PolySet to plot
polys <- data.frame(PID=rep(1, 4), POS=1:4, X=c(0, 1, 1, 0), Y=c(0, 0, 1, 1))
#--- plot the PolySet
plotMap(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5), density=0)
```

`plotPolys` *Plot a PolySet as Polygons*

## Description

Plots a PolySet as polygons.

## Usage

```
plotPolys (polys, xlim = NULL, ylim = NULL,
    plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL, border = NULL,
    lty = NULL, col = NULL, density = NA, angle = NULL, bgCol = NULL,
    main = "Map", xlab = NULL, ylab = NULL, axes = TRUE, tckLab = TRUE,
    tck = 0.014, tckMinor = 0.5 * tck, ...)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to plot (*required*). |
| `xlim` | vector range of x-values. |
| `ylim` | vector range of y-values. |
| `plt` | four element numeric vector `(x1, x2, y1, y2)` giving the coordinates of the plot region measured as a fraction of the figure region. |
| `polyProps` | PolyData specifying which polygons to plot and their properties. The data in this object are superseded by plot properties passed as direct arguments. |
| `border` | numeric or character (R only) vector describing colours for the edges of polygons (cycled by `PID`). |
| `lty` | numeric or character (R only) vector describing line types (cycled by `PID`). |
| `col` | numeric or character (R only) vector describing fill colours (cycled by `PID`). |
| `density` | numeric vector describing shading line densities (lines per inch, cycled by `PID`). |
| `angle` | numeric vector describing shading line angles (degrees, cycled by `PID`). |
| `bgCol` | numeric or character (R only) scalar to colour the background of the plot. |
| `main` | string title for the plot. |
| `xlab` | string caption for the x-axis. |
| `ylab` | string caption for the y-axis. |
| `axes` | Boolean value; if `TRUE`, plot axes. |
| `tckLab` | Boolean vector (length 1 or 2) indicating whether or not to label the major tick marks. If a two-element vector is given, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |

| | |
|---|---|
| `tck` | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If `tckLab = TRUE`, these tick marks will be automatically labelled. If a two-element vector is given, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| `tckMinor` | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smaller of the width or height of the plotting region. These tick marks may not be automatically labelled. If a two-element vector is given, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| `...` | additional `par` parameters for the `polygon` function. |

## Details

Plots a PolySet, where each (`PID`, `SID`) set describes a unique polygon. To do so, it connects the last vertex of each set to the first vertex of that set. This function supports both borders (`border`, `lty`) and fills (`col`, `density`, `angle`). It can also be used to draw only borders or only fills by supplying the appropriate arguments. This function ignores the aspect ratio. Furthermore, it clips `polys` to `xlim` and `ylim` before plotting.

## See Also

`addLines`, `addPolys`, `plotLines`, and `plotMap`.

## Examples

```
#--- create a PolySet to plot
polys <- data.frame(PID=rep(1, 4), POS=1:4, X=c(0, 1, 1, 0), Y=c(0, 0, 1, 1))
#--- plot the PolySet
plotPolys(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5), density=0)
```

---

`pythagoras` *Pythagoras' Theorem Diagram Data*

---

## Description

PolySet of shapes to prove Pythagoras' Theorem: $a^2 + b^2 = c^2$.

## Usage

```
data(pythagoras)
```

## Format

4 column data frame: `PID` = primary polygon ID, `POS` = position of each vertex within a given polyline, `X` = X coordinates, and `Y` = Y coordinates. Attributes: `projection = 1`.

### Note

In R, the data must be loaded using the `data` function. In S, the data are loaded automatically on demand.

### Source

An artifical construct to illustrate the proof of Pythagoras' Theorem using trigonometry.

### See Also

`nepacLL`, `nepacLLhigh`, `towTracks`, `towData`, `surveyData`, `worldLL`, and `worldLLhigh`.

---

## `surveyData` *Survey Data*

---

### Description

PolyData of Pacific ocean perch (POP) tow information (1966-89).

### Usage

```
data(surveyData)
```

### Format

Data frame consisting of 9 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polygon, `X` = longitude coordinates, `Y` = latitude coordinates, `trip` = trip ID, `tow` = tow number in trip, `catch` = catch of POP (kg), `effort` = tow effort (minutes), `depth` = fishing depth (m), and `year` = year of survey trip. Attributes: `projection = "LL"`, `zone = 9`.

### Note

In R, the data must be loaded using the `data` function. In S, the data are loaded automatically on demand.

### Source

The GFBio database, maintained at the Pacific Biological Station (Fisheries and Oceans Canada, Nanaimo, BC V9T 6N7), archives catches and related biological data from commercial groundfish fishing trips and research/assessment cruises off the west coast of British Columbia (BC).

The POP (*Sebastes alutus*) survey data were extracted from GFBio. The data extraction covers bottom trawl surveys that focus primarily on POP biomass estimation: 1966-89 for the central BC coast and 1970-85 for the west coast of Vancouver Island. Additionally, a 1989 cruise along the entire BC coast concentrated on the collection of biological samples. Schnute et al. (2001) provide a more comprehensive history of POP surveys including the subset of data presented here.

### References

Schnute, J.T., R. Haigh, B.A. Krishka, and P. Starr. 2001. Pacific ocean perch assessment for the west coast of Canada in 2001. Canadian Science Advisory Secretariat Research Document 2001/138, 90 p.

### See Also

`nepacLL`, `nepacLLhigh`, `pythagoras`, `towData`, `towTracks`, `worldLL`, and `worldLLhigh`.

---

## `towData` *Tow Track Data*

---

### Description

PolyData of tow information for a longspine thornyhead survey (2001).

### Usage

```
data(towData)
```

### Format

Data frame consisting of 8 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polygon, `X` = longitude coordinates, `Y` = latitude coordinates, `depth` = fishing depth (m), `effort` = tow effort (minutes), `distance` = tow track distance (km), `catch` = catch of longspine thornyhead (kg), and `year` = year of survey. Attributes: `projection = "LL"`, `zone = 9`.

### Note

In R, the data must be loaded using the `data` function. In S, the data are loaded automatically on demand.

### Source

The GFBio database, maintained at the Pacific Biological Station (Fisheries and Oceans Canada, Nanaimo, BC V9T 6N7), archives catches and related biological data from commercial groundfish fishing trips and research/assessment cruises off the west coast of British Columbia (BC). The longspine thornyhead (*Sebastolobus altivelis*) survey data were extracted from GFBio. Information on the first 45 tows from the 2001 survey (Starr et al. 2002) are included here. Effort is time (minutes) from winch lock-up to winch release.

### References

Starr, P.J., B.A. Krishka, and E.M. Choromanski. 2002. Trawl survey for thornyhead biomass estimation off the west coast of Vancouver Island, September 15 - October 2, 2001. Canadian Technical Report of Fisheries and Aquatic Sciences 2421, 60 p.

### See Also

`nepacLL`, `nepacLLhigh`, `pythagoras`, `towTracks`, `surveyData`, `worldLL`, and `worldLLhigh`.

---

## `towTracks` *Tow Track Polyline Data*

---

### Description

PolySet of geo-referenced polyline tow track data from a longspine thornyhead survey (2001).

### Usage

```
data(towTracks)
```

### Format

Data frame consisting of 4 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polyline, `X` = longitude coordinates, and `Y` = latitude coordinates. Attributes: `projection = "LL"`, `zone = 9`.

### Note

In R, the data must be loaded using the `data` function. In S, the data are loaded automatically on demand.

### Source

The longspine thornyhead (*Sebastolobus altivelis*) tow track spatial coordinates are available at the Pacific Biological Station (Fisheries and Oceans Canada, Nanaimo, BC V9T 6N7). The geo-referenced coordinates of the first 45 tows from the 2001 survey (Starr et al. 2002) are included here. Coordinates are recorded once per minute between winch lock-up and winch release.

### References

Starr, P.J., B.A. Krishka, and E.M. Choromanski. 2002. Trawl survey for thornyhead biomass estimation off the west coast of Vancouver Island, September 15 - October 2, 2001. Canadian Technical Report of Fisheries and Aquatic Sciences 2421, 60 p.

## See Also

`nepacLL`, `nepacLLhigh`, `pythagoras`, `towData`, `surveyData`, `worldLL`, and `worldLLhigh`.

---

`worldLL` *Shorelines of the World (Normal Resolution)*

---

## Description

PolySet of polygons for the global shorelines.

## Usage

```
data(worldLL)
```

## Format

Data frame consisting of 4 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polygon, `X` = longitude coordinates, and `Y` = latitude coordinates.

## Note

In R, the data must be loaded using the `data` function. In S, the data are loaded automatically on demand.

## Source

Polygon data from the GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline) database. We thinned the full resolution GSHHS binary data with a tolerance of 5.0 km and then converted it to ASCII data using the supplied software (`gshhs_dp.exe` and `gshhs.exe`, respectively). Using a Perl script (`gshhs2r.pl`), we then converted this ASCII format to ours, removed the lakes, islands in lakes, and ponds in islands, and filtered out any islands with fewer than 15 vertices.

## References

Wessel, P., and W.H.F. Smith. 1996. A global, self-consistent, hierarchical, high-resolution shoreline database, Journal of Geophysical Research 101: 8741-8743.
`http://www.soest.hawaii.edu/pwessel/pwessel_pubs.html`

## See Also

`nepacLL`, `nepacLLhigh`, `pythagoras`, `towTracks`, `towData`, `surveyData`, and `worldLLhigh`.

`worldLLhigh` *Shorelines of the World (High Resolution)*

## Description

PolySet of polygons for the global shorelines.

## Usage

```
data(worldLLhigh)
```

## Format

Data frame consisting of 4 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polygon, `X` = longitude coordinates, and `Y` = latitude coordinates.

## Note

In R, the data must be loaded using the `data` function. In S, the data are loaded automatically on demand.

## Source

Polygon data from the GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline) database. We thinned the full resolution GSHHS binary data with a tolerance of 1.0 km and then converted it to ASCII data using the supplied software (`gshhs_dp.exe` and `gshhs.exe`, respectively). Using a Perl script (`gshhs2r.pl`), we then converted this ASCII format to ours, removed the lakes, islands in lakes, and ponds in islands, and filtered out any islands with fewer than 15 vertices.

## References

Wessel, P., and W.H.F. Smith. 1996. A global, self-consistent, hierarchical, high-resolution shoreline database, Journal of Geophysical Research 101: 8741-8743.
`http://www.soest.hawaii.edu/pwessel/pwessel_pubs.html`

## See Also

`nepacLL`, `nepacLLhigh`, `pythagoras`, `towTracks`, `towData`, `surveyData`, and `worldLL`.