PBSmapping 2.69: User's Guide

Jon T. Schnute, Nicholas M. Boers, Rowan Haigh, and Alex Couture-Beil

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

2015

User's Guide Revised from Canadian Technical Report of Fisheries and Aquatic Sciences 2549

Fisheries and Oceans Canada | Pêches et Océans Canada

Canada

Revisions to: Cat. No. Fs97-6/2549E ISSN 0706-6457

Last update: Feb 24, 2015

Correct citation for this publication:

Schnute, J.T., Boers, N.M., Haigh, R., and Couture-Beil, A. 2015. PBSmapping 2.69: user's guide revised from Canadian Technical Report of Fisheries and Aquatic Sciences 2549: vi + 43 p. Last updated Feb 24, 2015.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Schnute, J.T., Boers, N.M., Haigh, R., and Couture-Beil, A. 2015. PBSmapping 2.69: user's guide revised from Canadian Technical Report of Fisheries and Aquatic Sciences 2549: vi + 43 p. Last updated Feb 24, 2015.

This report describes a second version of software designed 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 **PBSmapping** extends the R-statistical language 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 Microsoft Windows operating systems that support computational geometry outside the framework of R. Our results, which depend significantly on the work of students, illustrate the convergence of goals between academic training and applied research.

RESUME

Schnute, J.T., Boers, N.M. Haigh, R., et Couture-Beil, A. 2015. PBSmapping 2.69: Guide de l'utilisateur révisé de Canadian Technical Report of Fisheries and Aquatic Sciences 2549: vi + 43 p. Dernier mis à jour Feb 24, 2015.

Le présent rapport décrit la seconde version du logiciel conçu pour faciliter la compilation et l'analyse de données halieutiques, en particulier les données référencées par des coordonnées spatiales. Nos travaux de recherche ont capitalisé sur des expériences menées à l'aide de données sur les pêches des poissons démersaux le long du littoral Pacifique du Canada, données compilées à la Station biologique du Pacifique (SBP). Bien que conçu initialement pour l'analyse de données halieutiques, notre logiciel peut s'appliquer à toute une variété de domaines. La bibliothèque **PBSmapping** *(Cartographie de la SBP)* étend le langage R pour inclure une capacité d'impression en deux dimensions semblable à celle habituellement disponible dans les systèmes d'information géographiques (SIG). Des modules en C permettent d'accélérer les algorithmes grâce à la géométrie numérique, en trouvant par exemple les polygones qui contiennent des événements ponctuels spécifiques ou en convertissant les longitudes et les latitudes en coordonnées de la projection transversale universelle (UTM). Nous présentons également un certain nombre d'applications intéressantes pour les systèmes d'exploitation Microsoft Windows, qui peuvent effectuer des opérations de géométrie numérique en dehors du cadre de travail R. Nos résultats, auxquels plusieurs étudiants ont grandement contribué, illustrent la convergence des objectifs de la formation académique et de la recherche appliquée.

PREFACE

During the last decade, 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. I'm particularly grateful for 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), 2003-2006.
- Alex Couture-Beil (Malaspina University-College), 2005-2007

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 computing 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). Venables and Ripley (1999, 2000) give excellent guidance for using either language. 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. Nick Boers has used such linkages intelligently to bring fast computational geometry into our R-package **PBSmapping**.

To some extent, this report constitutes a second edition of an earlier report (Schnute et al. 2003) that describes a suite of software utilities developed at PBS. In particular, the package **PBSmapping** has undergone extensive renovations and improvements, and this document provides a definitive manual for using version 2. To accommodate the new material presented here, my co-authors and I have decided to remove sections of the earlier report that discuss other PBS software utilities, free software available on the Internet, and related technical information. Readers of this current report may also wish to acquire the earlier version for additional material not included here.

I want to mention two milestones achieved during the production of **PBSmapping**, Version 2. First, we have posted the current software as a contributed package on the Comprehensive R Archive Network (CRAN, http://cran.r-project.org/). Thanks to a remarkable collection of Perl scripts developed for the R project, source code in both C and R, along with suitable documentation files, can be tested and compiled automatically for distribution as both source and binary packages. Nick Boers ensured that our source materials met the necessary standards, and (after we made minor changes in the C code to avoid compiler warnings) the authors of the CRAN web site in Vienna, Austria accepted our contribution. Second, Nick applied to the Canadian Natural Sciences and Engineering Research Council (NSERC) for a grant to support graduate studies in computing science. His application cited his successful experience developing **PBSmapping**, Version 1, as documented in Schnute et al. (2003). To the delight of Nick's supporters at PBS and MUC, he won a substantial award, in fact the only NSERC grant given to a student from MUC this year. Congratulations, Nick, from your colleagues at PBS and professors at MUC. We'll follow your career at the University of Alberta in Edmonton with great interest.

Jon T. Schnute

This page has been left intentionally blank for printing purposes.

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 specialized training. Because analysts who deal with Pacific groundfish data often have experience using the statistical languages R (http://cran.r-project.org/, available for free) or S-PLUS (http://en.wikipedia.org/wiki/S-PLUS, available commercially), we began by writing bilingual functions for these languages to produce the maps required. Schnute et al. (2003) describe the package **PBSmapping**, Version 1, which evolved from these early experiences. After another year of development, we extensively revised the software, and Schnute et al. (2004) presented a user's manual for **PBSmapping**, Version 2. Subsequently, we have dropped the bilingual (R/S-PLUS) nature of **PBSmapping**, producing revisions solely for R, and now refer to the package as **PBSmapping** rather than 'PBS Mapping' used in earlier documents. Additionally, we maintain most of our PBS packages, including **PBSmapping**, at http://code.google.com/p/pbs-software/.

Section 2 covers the mapping software itself, which contains functions that perform numerous calculations on polygons. These include standard set theoretic operations (union, intersection, difference, exclusive-or), clipping, thinning, thickening, testing convexity, forming the convex hull, and calculating various statistics (such as mean, centroid, and area). We discuss public data that represent shorelines and ocean bathymetry, and the package includes sample data sets drawn from these sources. We also discuss the Universal Transverse Mercator (UTM) projection that gives a particularly accurate flat projection of the earth's surface. Our software can convert between longitude-latitude and UTM coordinates.

Section 3 documents a number of convenient command-line utilities, compiled separately from C code written for the **PBSmapping** package. These make it possible to perform some of the polygon functions outside the framework of R. Appendices provide additional information about various topics related to **PBSmapping**, including

A. a package (**PBSdata**) of supplementary information for **PBSmapping**;
B. an Internet source for global bathymetry data;
C. alternative Generic Mapping Tools (GMT);
D. source code for the figures in this report;
E. function dependencies in **PBSmapping**;
F. documentation for **PBSmapping** functions and data.

We anticipate that our software will continue to change for the better, due to bug fixes and other improvements. This report documents version 2.69, which currently appears as a contributed package on the R archive (http://cran.r-project.org/). We will post subsequent versions as they become available. All software required to develop and use **PBSmapping** is freely available from the Internet.

1.1. Software Installation

We provide two mapping packages:

- **PBSmapping** – the mapping software discussed in Section 1;
- **PBSdata** – additional data sets relevant to fisheries investigated at PBS (Appendix A).

Installation of **PBSmapping** can be achieved in two ways – (1) navigate to: http://cran.r-project.org/web/packages/PBSmapping/index.html, download the appropriate binary, and install from R using the menu <Packages><Install package(s) from local zip files...>, or (2) in R, use the menu <Packages><Install package(s)>, choose a CRAN mirror near you, highlight **PBSmapping**, and press OK. Note that the software is available in two forms:

- `PBSmapping_2.69.tar.gz` – source code for the R distribution, which can be used to build a binary package;
- `PBSmapping_2.69.zip` – binary package ready for installation into R;

The package **PBSdata** can be found on the Google Code website: https://code.google.com/p/pbs-data/ .

To remove **PBSmapping** from R, open the `library\` directory and delete the associated subdirectory `PBSmapping\`. Before loading a new version of a package, we recommend the removal of any previous version. Eventually, the installation files may have names that reflect a version number later than the current version.

Additionally three other PBS packages are available from CRAN that facilitate fisheries analysis and research:

- **PBSmodelling** – http://cran.r-project.org/web/packages/PBSmodelling/index.html;
- **PBSddesolve** – http://cran.r-project.org/web/packages/PBSddesolve/index.html;
- **PBSadmb** – http://cran.r-project.org/web/packages/PBSadmb/index.html.

The **PBSmodelling** library includes a *directory* called `PBStools` that contains useful batch files for building R packages and generating an indexed manual based on the `*.Rd` files. This is not to be confused with another PBS package called **PBStools** at: https://code.google.com/p/pbs-tools/.

2. FUNCTIONS AND DATA

Niklaus Wirth, the author of Pascal and Modula-2, summarises the essence of software design in the title of his book *Algorithms + Data Structures = Programs* (Wirth 1975). Our software package **PBSmapping** 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 geographical coordinates, such as longitude and latitude. The R language conveniently supports such structures through the concept of a *data frame*, essentially a database table in which rows and columns define records and fields, respectively. Objects like data frames in R can also have *attributes* that store additional properties, such as the projection used in defining a geographic coordinate system.

2.1. Data Structures for Maps

PBSmapping introduces four data structures, each stored as a data frame. Field names, attributes, and other properties of these objects implicitly dictate their type. An object may also identify its type explicitly in the `class` attribute. Each type requires a particular structure, as outlined below.

2.1.1. PolySet

In our software, a *PolySet* data frame defines a collection of polygonal contours (i.e., line segments joined at vertices), based on four or five numerical fields:

- `PID` – the primary identification number for a contour;
- `SID` – (optional) the secondary identification number for a contour;
- `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 lacks an `SID` column, and each `PID` corresponds to a different contour. 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 contour 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 segments. The second includes a final line segment joining points n and 1, thus giving a *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 associated with a different `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, regardless of rotational direction. This contrasts with other GIS software, such as ArcView (ESRI 1996), in which outer and inner boundaries correspond to clockwise and counter-clockwise directions, respectively.

The SID field in a PolySet with secondary IDs must have integer values that appear in ascending order for a given PID. Furthermore, inner boundaries must follow the outer boundary that encloses them. The POS field for each contour (PID, SID) must similarly appear as integers in strictly increasing or decreasing order, for outer and inner boundaries respectively. If the POS field erroneously contains floating-point numbers, fixPOS can renumber them as sequential integers, thus simplifying the insertion of a new point, such as point 3.5 between points 3 and 4.

A PolySet can have a projection attribute, which may be missing, that specifies a map projection. In the current version of PBSmapping, 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 optional attribute zone specifies the UTM zone (if projection="UTM") or the preferred zone for conversion from Longitude-Latitude (if projection="LL").

A data frame's class attribute by default contains the string "data.frame". Inserting the string "PolySet" as the class vector's first element alters the behaviour of some functions. For example, the summary function will print details specific to a PolySet. Also, when PBSprint is TRUE, the print function will display a PolySet's summary rather than the contents of the data frame.

2.1.2. 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 contour 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 contours, where the data correspond to additional fields in the data frame. The R language conveniently allows data frames to contain fields of various atomic modes ("logical", "numeric", "complex", "character", and "null"). For example, PolyData with the fields (PID, PName) might assign character names to a set of primary polygons. Additionally, if fields X and Y exist (perhaps representing locations for placing labels), consider adding attributes zone and projection. Inserting the string "PolyData" as the class attribute's first element alters the behaviour of some functions, including print (if PBSprint is TRUE) and summary.

Our software particularly uses PolyData to set various plotting characteristics. Consistent with graphical parameters used by the R 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 (as opposed to closed polygons), only lty and col have meaning.

2.1.3. EventData

We define *EventData* as a data frame with at least three fields named (`EID`, `X`, `Y`). Conceptually, an EventData object describes events (`EID`) 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 absent. Inserting the string `"EventData"` as the `class` attribute's first element alters the behaviour of some functions, including `print` (if `PBSprint` is `TRUE`) and `summary`.

2.1.4. LocationSet

A PolySet can define regional boundaries for drawing a map, and EventData can give event points on the map. Which events occur in which regions? Our function `findPolys`, discussed in Section 2.3 below, solves 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`) field 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 in multiple records. If an `EID` does not fall in any (`PID`, `SID`), or if it falls within a hole, it does not occur in the output LocationSet. Inserting the string `"LocationSet"` as the first element of a LocationSet's `class` attribute alters the behaviour of some functions, including `print` (if `PBSprint` is `TRUE`) and `summary`.

2.2. Map Projections

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 displacements 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.

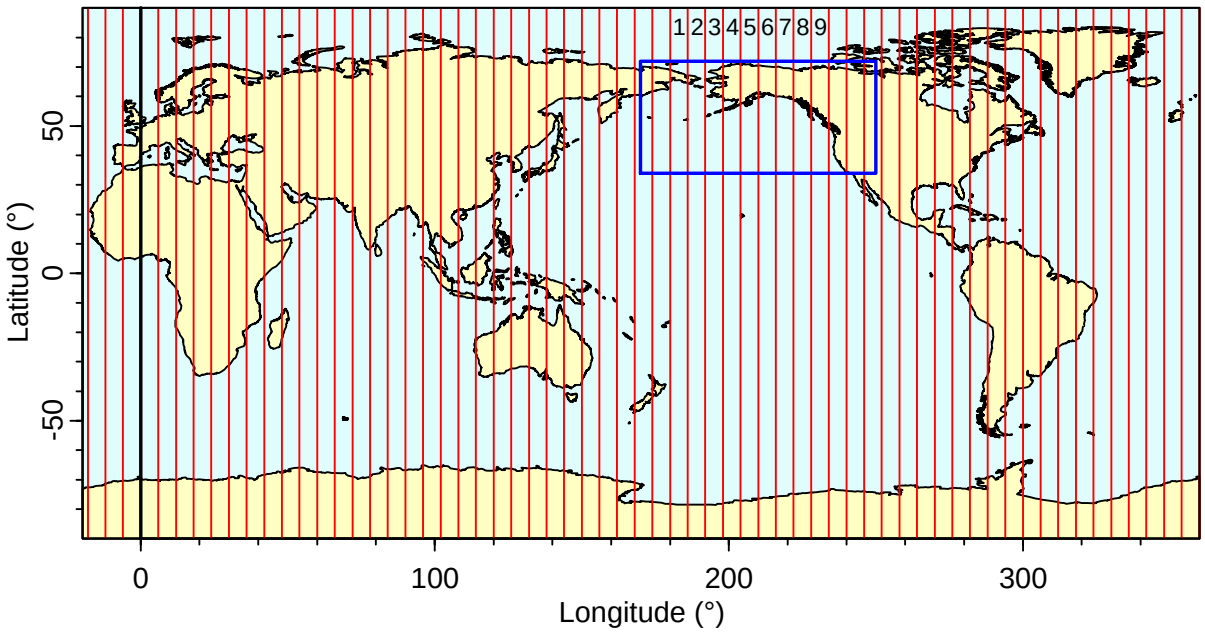

Figure 1. Map of the world projected in longitude-latitude coordinates. This image, based on our PolySet `worldLL`, uses the longitude range $-20° \le x \le 360°$ to produce a convenient cut in the eastern Atlantic Ocean. Red vertical lines show boundaries for the 60 Universal Transverse Mercator (UTM) 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 to 10 (Figure 2), and the projection to zone 9 gives a reasonably accurate map for fisheries in this region.

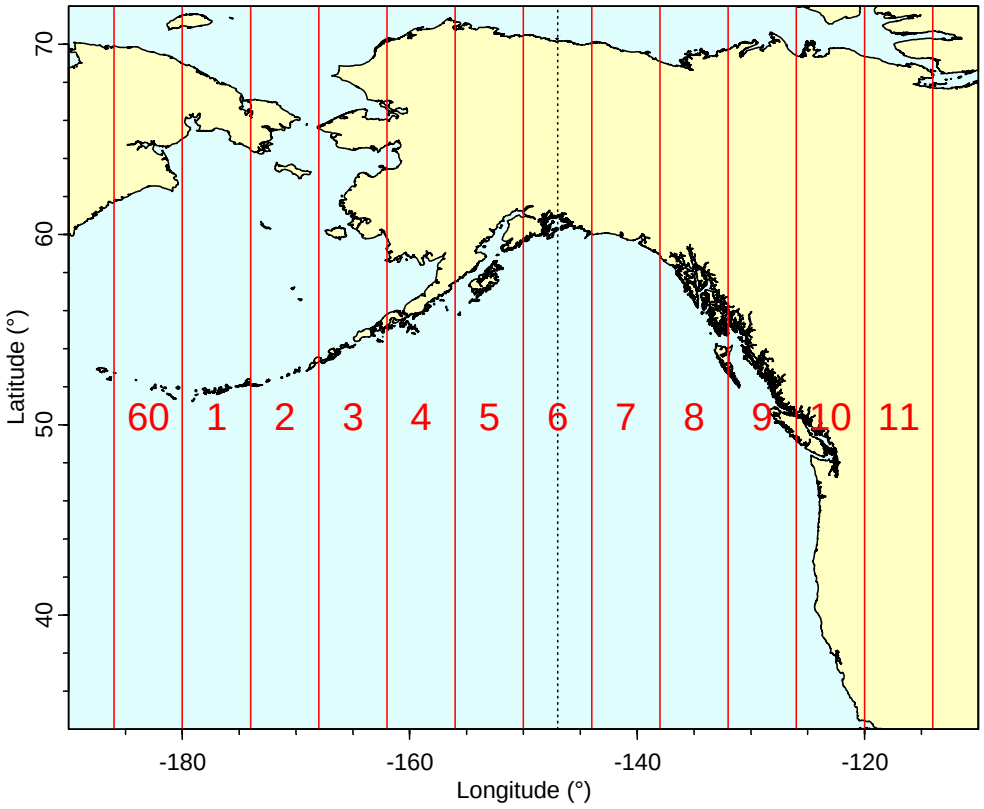

Figure 2. Shoreline data in longitude-latitude coordinates for the northeastern Pacific Ocean, as captured in our PolySet `nepacLL`. Vertical red lines display UTM boundaries for zones 60, 1, 2, …, 11. A vertical dotted line indicates the central meridian of zone 6, near the centre of this figure.

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, Ordnance Survey 2010), 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 (positive Y) or south (negative Y) of the equator. The central meridian is assigned a 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.

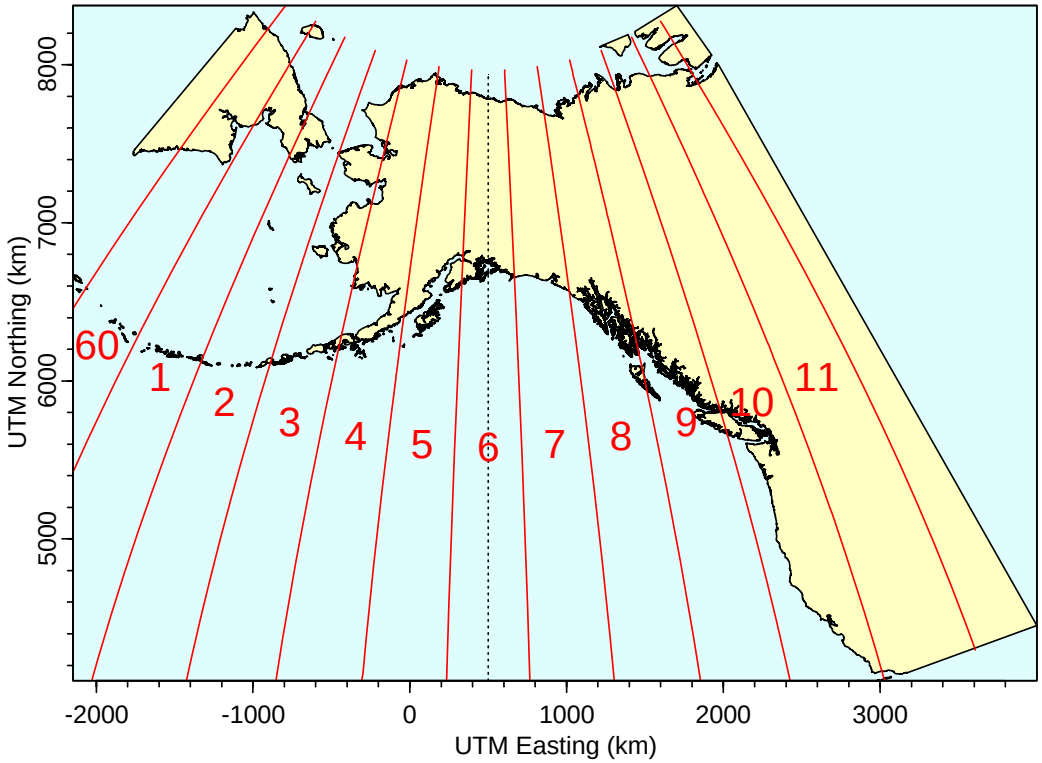

Figure 3. Shoreline data for the northeastern Pacific Ocean, projected in UTM coordinates (zone 6) from our PolySet `nepacLL`. Vertical red lines show UTM zone boundaries. The central axis of zone 6 (vertical dotted line at $x = 500$ km) corresponds to the central meridian shown in Figure 2.

2.3. PBSmapping Functions and Algorithms

Our software produces maps from the data structures defined in Section 2.1. Following typical design concepts in R, it uses functions to generate plots, implement algorithms, and perform other tasks. Where possible, function arguments often have explicit default values. `PBSmapping` includes many functions not mentioned in this section. We encourage readers to examine Appendix F, which gives detailed technical descriptions of all our software's functions and other components.

2.3.1. Import Functions

The following functions provide some support for importing GIS data from other users and other mapping platforms:

- `importEvents` import a text file and convert into EventData.
- `importLocs` import a text file and convert into a LocationSet.
- `importPolys` import a text file and convert into a PolySet with optional PolyData attribute.
- `importGSHHS` import data from a GSHHS database and convert data into a PolySet with a PolyData attribute. GSHHS: A Global Self-consistent, Hierarchical, High-resolution Shoreline Database, http://www.soest.hawaii.edu/pwessel/gshhg/
  See Section 2.4 below for more details.

- `importShapefile` imports an ESRI shapefile (`.shp`) into either a PolySet or EventData. The function relies on C-code provided by Roger Bivand's package `maptools`.

2.3.2. Graphics Functions

In the R 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`, `plotMap`, `plotPoints`, `plotPolys`) that create graphs and others (`addLabels`, `addLines`, `addPoints`, `addPolys`, `addStipples`) that add graphical features.

Some of these plotting functions draw objects defined by a PolySet, while others expect EventData, a LocationSet, or PolyData. Both `plotLines` and `addLines` treat their input PolySet as polylines, with no connection between the last and first vertices. By contrast, `plotMap`, `plotPolys`, and `addPolys` regard their input as polygons, where a final line segment connects the last vertex to the first. The functions `plotMap` and `plotPolys` behave similarly, except that `plotMap`'s default behaviour guarantees the correct aspect ratio, as defined by either the PolySet's `projection` attribute or the function's `projection` argument. If both are specified, the attribute supersedes the argument. When this attribute is missing, `plotMap` uses a 1:1 projection. Table 1 summarises the default behaviour of our principal graphics commands. A user concerned with drawing maps, where the correct aspect ratio plays a key role, would likely initiate a graph with the `plotMap` function. However, `plotPolys`, `plotLines`, and `plotPoints` can also set the correct aspect ratio when passed a suitable `projection` argument.

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

Function	Creates a Graph	Plots as Polygons	Sets Aspect Ratio by Default
`addLabels`	No	-	-
`addLines`	No	No	-
`addPoints`	No	-	-
`addPolys`	No	Yes	-
`addStipples`	No	-	-
`plotLines`	Yes	No	No
`plotMap`	Yes	Yes	Yes
`plotPoints`	Yes	-	No
`plotPolys`	Yes	Yes	No

Our high-level graphics functions accept a common set of arguments, consistent with existing `par` parameters where possible. These include

- `xlim` and `ylim` to specify horizontal and vertical coordinate ranges;
- `projection` to specify the projection used in drawing the map or graph;
- `plt` to define the plot region relative to the figure region;
- `polyProps` to support plotting properties for individual contours (Section 2.1);
- `lty`, `cex`, `col`, `border`, `density`, `pch`, and `angle` to adjust properties of labels, lines, points, and polygons where applicable;
- `axes` to disable plotting axes;
- `tck` to control (major) tick mark lengths;

- `tckMinor`, a counterpart of `tck`, to set a different length for minor tick marks;
- `tckLab`, with Boolean values, to determine whether to include numeric tick labels.

We introduce `tckMinor` and `tckLab` to give finer control over the appearance of tick marks. Each of `tck`, `tckLab`, and `tckMinor` can have length one or two. A single value pertains to both axes, and two values specify distinct parameters for the horizontal and vertical axes, respectively.

Our low-level graphics functions (e.g., `addLines`) use many of the same arguments as their high-level counterparts (e.g., `plotLines`). However, they do not accept parameters that affect the overall plot, such as `xlim`, `ylim`, `projection`, `plt`, `axes`, or any of the `tck` arguments.

The `par` parameter `plt` plays a special role in **PBSmapping**, because we use it to set the aspect ratio required for a particular `projection`. Recall that in R the plot region lies inside the figure region, which similarly lies inside the overall device region. The parameter `plt` specifies the plot region boundaries as fractions (left, right, bottom, top) of the current figure region. Our high-level plotting functions use the initial default value `plt=c(0.11,0.98,0.12,0.88)`, but then alter `plt` by shrinking the width or height to achieve the required aspect ratio. In the function call, the argument `plt` can set a different default value, but again this may be changed by the graphics function to set the aspect ratio. In effect, the argument `plt` sets minimum margins for the plot within the figure region, but the aspect ratio may force the plot to shrink in width or height, giving wider margins in one direction.

Standard high-level commands in R (like `plot`) do not allow layout parameters (like `plt`) to be passed as arguments. Instead, users normally use `par` to set these parameters before invoking a graphics command. However, unlike normal graphics commands, those in **PBSmapping** actually alter the margins, so we adopt a different approach in which `plt` is reset with each high-level command. Advanced users wishing to set the plot region using the `par` parameters `mai` or `mar` can disable the default initial size with the argument `plt=NULL`.

2.3.3. Computational Functions

PBSmapping contains many functions that perform computations on PolySets and other data structures. Appendix F lists them all, but we give further details for some of them here, including formulas or algorithms for implementation and references for further reading. In alphabetic order, this list below highlights key features of selected functions in the package.

- `calcArea` computes polygon areas by the formula (Rokne 1996)

  $$A = \frac{1}{2} \sum_{i=1}^{n-1} \left(x_i y_{i+1} - x_{i+1} y_i \right) ,$$

  for the area A of a polygon with vertices (x_i, y_i), $i = 1, \ldots, n$, where vertices 1 and n correspond to the same point: $(x_1, y_1) = (x_n, y_n)$. This formula assumes identical units for x and y (an aspect ratio 1), as in UTM coordinates. The function automatically converts longitude-latitude coordinates to UTM before calculating the area.

- `calcCentroid` computes polygon centroid coordinates (x, y) by the formulae (Bourke 1988)

$$x = \frac{1}{6A}\sum_{i=1}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$
$$y = \frac{1}{6A}\sum_{i=1}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

for a polygon with vertices (x_i, y_i), $i = 1, \ldots, n$, where vertices 1 and n correspond to the same point: $(x_1, y_1) = (x_n, y_n)$ and A is computed by the formula shown above in the definition of `calcArea`. These formulas scale automatically to the units of x and y and consequently do not depend on the projection attribute.

- `calcConvexHull` calculates the convex hull for a given set of points using the function `chull()` in R's package `grDevices`.
- `calcLength` calculates polyline lengths using Pythagoras' Theorem when the projection is UTM or 1. Thus, the distance d between points (x, y) and (x', y') is

  $$d = \sqrt{(x'-x)^2 + (y'-y)^2}\ .$$

  The function also supports longitude-latitude coordinates (x, y) by calculating great circle distances between polygon vertices. In this case, the distance d between two points is (Chamberlain 2001)

  $$d = 2R \arcsin\left[\sqrt{\sin^2\left(\frac{y'-y}{2}\right) + (\cos y)(\cos y')\sin^2\left(\frac{x'-x}{2}\right)}\right],$$

  where $R = 6371.3$ km denotes the earth's mean radius (Wikipedia 2004).

- `calcMidRange` calculates midpoints of the `X` and `Y` ranges for each given polygon.
- `calcSummary` calculates summary statistics for a PolySet, given a user-defined function.
- `calcVoronoi` calculates the Voronoi (Dirichlet) tesselation for a set of points (using the `deldir` function from the package `deldir`) and creates a PolySet. See Figure 8 of the `PBSmodelling` user's guide (Schnute *et al*. 2006) for an example called `CalcVor`.
- `clipLines` (and `clipPolys`) clips polylines (and polygons) within a specified rectangle, possibly smaller than the bounding rectangle, using the Sutherland-Hodgman clipping algorithm (Foley et al. 1996, p. 124-127).
- `closePolys` adds corners from the bounding rectangle, if needed, to close polylines into polygons.
- `combinePolys` combines several polygons into a single polygon by modifying the PID and SID indices.
- `convCP` converts results from `contourlines` into a PolySet.
- `convDP` converts EventData/PolyData into a PolySet.
- `convLP` converts two polylines into a polygon.
- `convUL` converts between UTM and longitude-latitude coordinates using the extensive formulas presented in Ordnance Survey (2010).
- `dividePolys` divides a single polygon (with several outer-contour components) into several polygons, a polygon for each outer contour, by modifying the PID and SID indices.

- `findCells` finds the cells in a grid PolySet that contain events specified in EventData, using the "crossings test" algorithm described later in this section.
- `findPolys` finds the polygons in a PolySet that contain events specified in EventData, using the "crossings test" algorithm described later in this section.
- `isConvex` determines which polygons in a PolySet are convex, using an algorithm described below.
- `isIntersecting` finds polygons that self-intersect by comparing each edge pairwise with every other edge.
- `joinPolys` performs set theoretic operations (union, intersection, difference, and exclusive-or) on polygons using the Clipper library developed by Angus Johnson (http://www.angusj.com/delphi/clipper.php). See Figure 13 of the **PBSmodelling** user's guide (Schnute *et al*. 2006) for an example called `FishTows` (Fig.14 in most recent version).
- `thickenPolys` adds vertices to polygons using an algorithm described below.
- `thinPolys` thins the number of polygon vertices, based on the Douglas-Peuker line simplification algorithm (Douglas and Peucker 1973), as illustrated in Figure 4.

Our function `isConvex` first calls `isIntersecting` to determine whether or not a polygon self-intersects. If it does, it cannot be convex and the result is `FALSE`. Otherwise, the function proceeds. Three sequential points in a non-self-intersecting polygon describe a left turn, a straight line, or a right turn. The function locates the first non-straight turn (left or right) in a polygon and checks that all subsequent turns are either the same or straight. If so, the polygon is convex; otherwise it is not.

Like `calcLength`, `thickenPolys` also supports the longitude-latitude projection. In this case, `tol` is measured in kilometres and distances are computed along great circles (Chamberlain 2001).

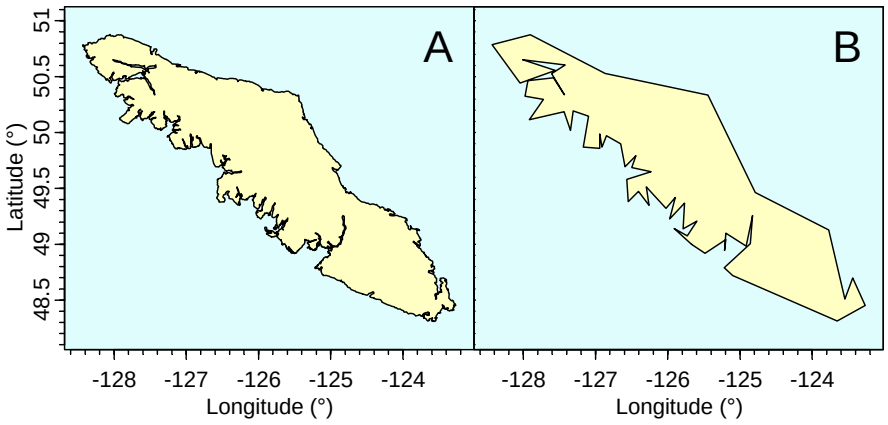

Figure 4. (A) Vancouver Island clipped from the PolySet `nepacLL` and (B) the result of applying `thinPolys` to this polygon with a tolerance of ten kilometres.

When the projection is UTM or 1, our function `thickenPolys` accepts a tolerance specified in `X` or `Y` units (kilometres in the UTM case). It operates in two distinct modes. When `keepOrig=TRUE`, it retains all original vertices and adds vertices, as required, along each edge.

Thus, if the distance between two sequential original vertices exceeds the specified tolerance `tol`, it adds enough vertices spaced evenly between them so that sequential vertices lie at most the distance `tol` apart. When `keepOrig=FALSE`, the algorithm guarantees only that the first vertex of each polygon appears in the result. Starting at that vertex, the algorithm walks through the polygon while summing distances between vertices. When the cumulative distance exceeds `tol`, it adds a vertex on the line segment under inspection. It then resets the distance sum and continues walking the polygon from this new vertex.

2.3.4. Associating Points with Polygons

As discussed in the definition of LocationSet (Section 2.1), 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 solve this problem, including:

- **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.
- **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.

We use the crossings test because it performs faster than angle summation (Hains 1994, p. 26-27). The latter requires large numbers of trigonometric function calls.

After finding the polygons that contain various events, an analyst often wants to compute statistics associated with the events that occur inside 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 relate polygon properties, such as colour used for plotting, to these computed statistical values.

2.3.5. Set Theoretic Operations

We include the function `joinPolys` to apply set theoretic operations (difference, intersection, union, and exclusive-or) to one or two PolySets. Our `joinPolys` function interfaces with the Clipper library (http://www.angusj.com/delphi/clipper.php) developed by Angus Johnson. Previously, it interfaced with the General Polygon Clipper library (http://www.cs.man.ac.uk/aig/staff/alan/software/) by Alan Murta at the University of Manchester. We keep this historic reference to GPC because `joinPolys` remains faithful to Murta's definition of a generic polygon, which we describe below.

Murta (2004) defines a *generic polygon* (or *polygon set*) as zero or more disjoint polygonal contours that define boundaries of the polygon region. Some contours can represent inner boundaries that define holes in the region. Each contour can be convex, concave, or self-intersecting.

In our PolySet, the polygons associated with each unique `PID` correspond to a generic polygon with some restrictions. Some of our functions do not support self-intersecting polygons.

Furthermore, the SID contours cannot be arranged in arbitrary order because we require that hole contours follow the outer contours in which they lie.

The function joinPolys can also accept two PolySet arguments *P* and *Q*. In this case, the function returns a PolySet with all possible pairwise applications of *op* between generic polygons in *P* and *Q*. For example, if *P* contains (*A*, *B*, *C*) and *Q* contains (*D*, *E*), then joinPolys returns a PolySet with six PIDs corresponding to the six generic polygons *A op D*, *B op D*, *C op D*, *A op E*, *B op E*, and *C op E*. More generally, if *P* and *Q* include *m* and *n* generic polygons, respectively, then the function returns a PolySet with $m \times n$ generic polygons. If $m = 1$ or $n = 1$, the output preserves PIDs from the PolySet with more than one generic polygon. Figure 5 illustrates the four supported set theoretic operations applied to crescent-shaped polygons *A* and *B*.

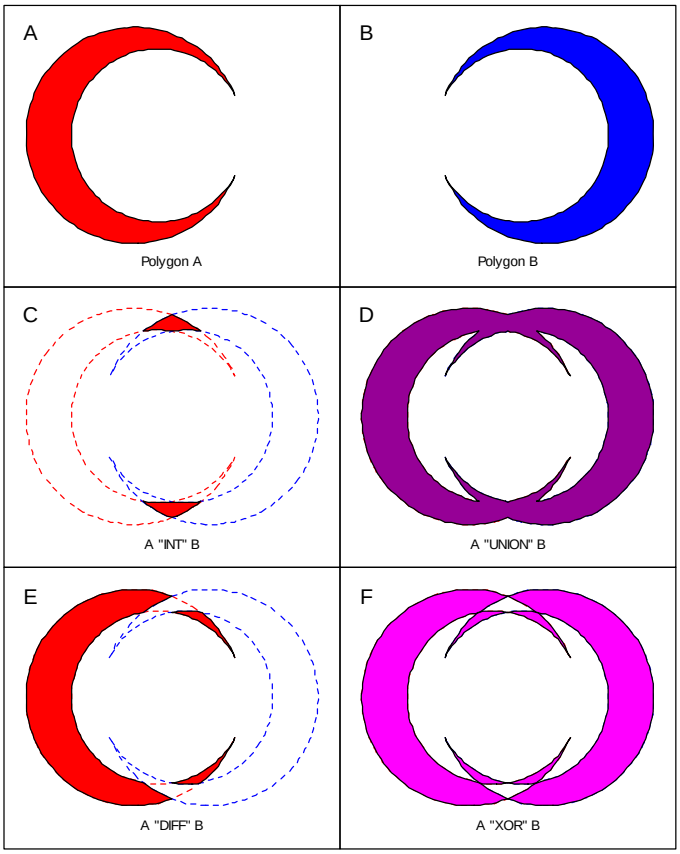

Figure 5. Example of the joinPolys logic operations. Panels A and B display the first and second PolySets, respectively. Panels C to F illustrate the intersection, union, difference, and exclusive-or operations, respectively.

Applied to one PolySet *P*, our function joinPolys applies the set theoretic operation *op* sequentially to the generic polygons in *P*. For example, suppose that *P* contains three generic polygons (*A*, *B*, *C*). Then the function returns a PolySet containing the generic polygon ((*A op B*) *op C*), represented as one PID with possibly many SIDs.

2.4. Shoreline Data

To portray fishery data along Canada's Pacific coast, we need a PolySet that defines the relevant shoreline. Originally, we began with a polyline of the British Columbia coast, digitized manually from a marine map. To convert this object to a meaningful closed polygon, we devised the functions `fixBound` 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, http://www.soest.hawaii.edu/pwessel/gshhg/) database for the entire planet. They make this available via the Internet as binary files in five different resolutions: full (`gshhs_f.b`), high (`gshhs_h.b`), intermediate (`gshhs_i.b`), low (`gshhs_l.b`), and crude (`gshhs_c.b`). 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. The `dp` is also an abbreviation of "decimate polygons".

We have created a function called `importGSHHS` that works directly on a specified binary data file from Wessel (resolution choice left to the user) to create a **PBSmapping** PolySet. The user can choose to further alter the resolution of the newly created PolySet using our function `thinPolys`. Alternatively, the user can thin Wessel's full-resolution database (`gshhs_f.b`) directly using `gshhs_dp.c` (after compilation to an executable file) to a desired resolution, then use **PBSmapping**'s `importGSHHS` on the modified binary database. At the time of writing, `importGSHHS` supports Wessel's format for data files version 2.2.0, created July 15, 2011. Wessel's database `gshhs+wdbii_2.2.0.zip` contains geographical coordinates for shorelines (`gshhs`), rivers (`wbd_rivers`), and borders (`wdb_borders`). The latter two come from World DataBank II (WDBII) with the five resolutions mentioned above.

PBSmapping includes four data sets derived from the GSHHS databases (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 $x = 0°$ on the Greenwich prime meridian.

Table 2. PolySets derived from various resolution GSHHG databases.

PolySet	Wessel DB	Thin	Longitude	Latitude	Vertices	Polygons
`nepacLL`*	`gshhs_h.b`	0.2 km	$-190° \le x \le -110°$	$34° \le y \le 72°$	75,305	495
`nepacLLhigh`	`gshhs_f.b`	0.1 km	$-190° \le x \le -110°$	$34° \le y \le 72°$	192,762	9,986
`worldLL`*	`gshhs_l.b`	5.0 km	$-20° \le x \le 360°$	$-90° \le y \le 84°$	30,129	190
`worldLLhigh`*	`gshhs_i.b`	1.0 km	$-20° \le x \le 360°$	$-90° \le y \le 84°$	187,101	1,367

*Excludes polygons with fewer than 15 vertices after thinning.

Explicitly, the commands to create the above PolySets are:

```
worldLL =
   importGSHHS("gshhs_l.b",xlim=c(-20,360),ylim=c(-90,90),level=1,n=15,xoff=0)
worldLL = .fixGSHHSWorld(worldLL)

worldLLhigh =
   importGSHHS("gshhs_i.b",xlim=c(-20,360),ylim=c(-90,90),level=1,n=15,xoff=0)
worldLLhigh = .fixGSHHSWorld(worldLLhigh)

nepacLL =
   importGSHHS("gshhs_h.b",xlim=c(-190,-110),ylim=c(34,72),level=1,n=15,xoff=-360)

nepacLLhigh =
   importGSHHS("gshhs_f.b",xlim=c(-190,-110),ylim=c(34,72),level=1,n=0,xoff=-360)
nepacLLhigh = thinPolys(nepacLLhigh, tol=0.1, filter=3)
```

2.5. Bathymetry Data

Smith and Sandwell (1997) have produced global seafloor topography from satellite altimetry and ship depth soundings. Their database appears on the Internet at http://topex.ucsd.edu/cgi-bin/get_data.cgi. A web-based data acquisition form allows users to extract a region after entering longitude and latitude coordinate ranges. Appendix B documents how to import their data for use with **PBSmapping**.

R provides a `contour` function to plot contour lines. This function lacks a `save` argument and does not return contour coordinates. Instead, the `contourLines` function accomplishes this task, giving a list that captures continuous contours as single polylines (Figure 6).

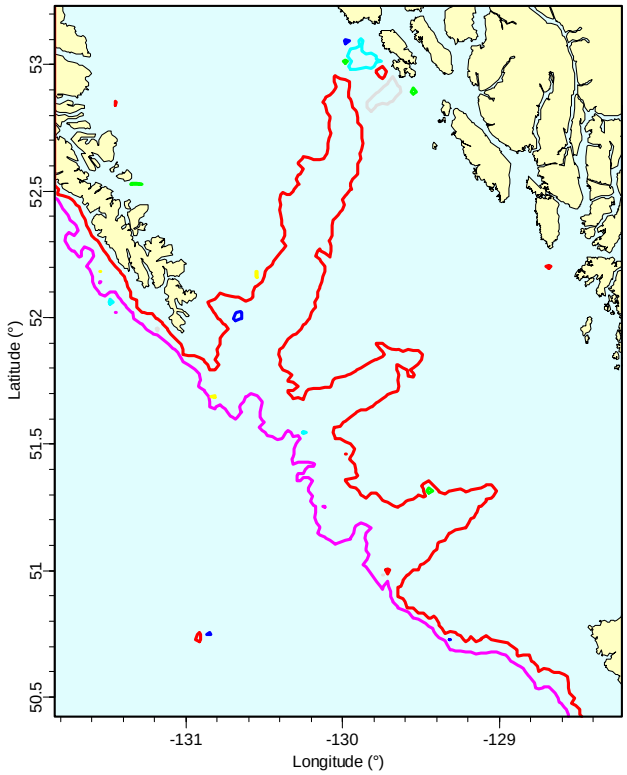

Figure 6. The R `contourLines` function returns a single polyline for each continuous contour.

Our function `convCP` converts the list output from `contourLines` into a list object that has two components: a PolySet with contour coordinates and PolyData with the depth of each contour. The package **PBSdata** includes a data set (`isobaths`) of bathymetric contours for Canada's Pacific coast. In addition, several functions ease the manual procedure of converting polylines into polygons, including

- `convLP` to convert two polylines into a single polygon;
- `closePolys` to close the polygons in a PolySet;
- `fixBound` to fix the boundary points of a PolySet.

2.6. 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 7 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).

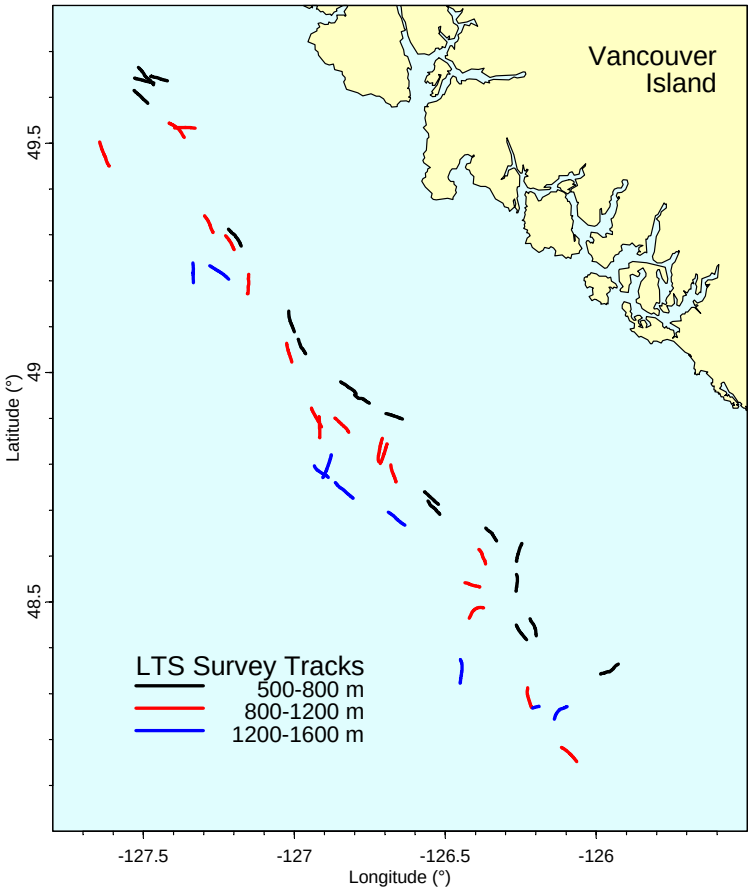

Figure 7. Tracks for 45 tows performed during the 2001 longspine thornyhead (*Sebastolobus altivelis*) survey along the west coast of Vancouver Island (Starr *et al*. 2002). Each tow track is colour-coded by depth stratum. Data: PolySet `towTracks` and PolyData `towData`.

Figure 8 illustrates the use of our software to calculate polygon areas using the function `calcArea`. 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 `calcArea` function will also automatically convert PolySets with `projection="LL"` to UTM before calculation.) The figure shows areas for six selected islands, highlighted in yellow. Island centroids, derived using `calcCentroid`, give reference coordinates for printing island names and areas.

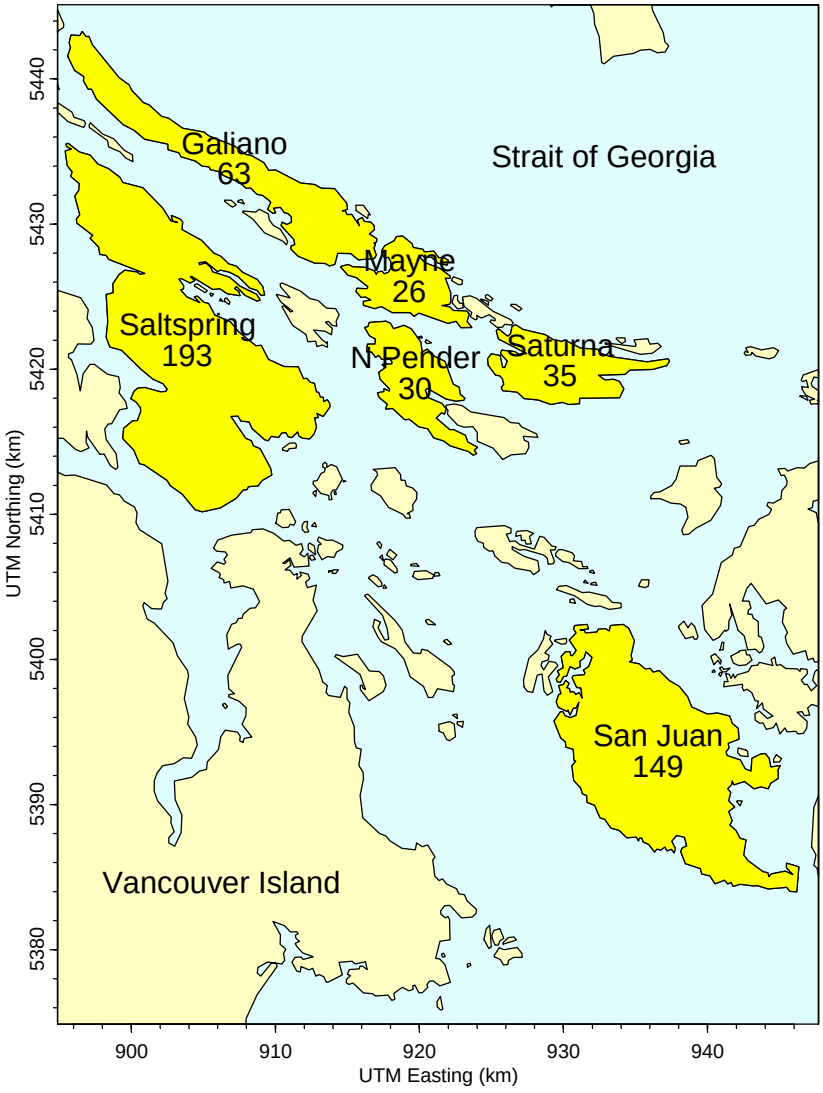

Figure 8. Areas (km^2) of selected islands in the southern Strait of Georgia. Shoreline data have been clipped from `nepacLLhigh` after conversion to UTM coordinates.

Figure 9 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). Code for this figure includes the following key function calls:

- `plotMap` to draw a coastal map of this region, clipped from `nepacLL`;
- `makeGrid` to create a grid in the region of interest;
- `findCells` to associate tows with the appropriate grid cells;
- `combineEvents` to calculate the mean CPUE within each cell;
- `addPolys` to draw cells with colours (in the `polyProps` argument) scaled to the CPUE;
- `points` (the native R function) to plot events on the map.

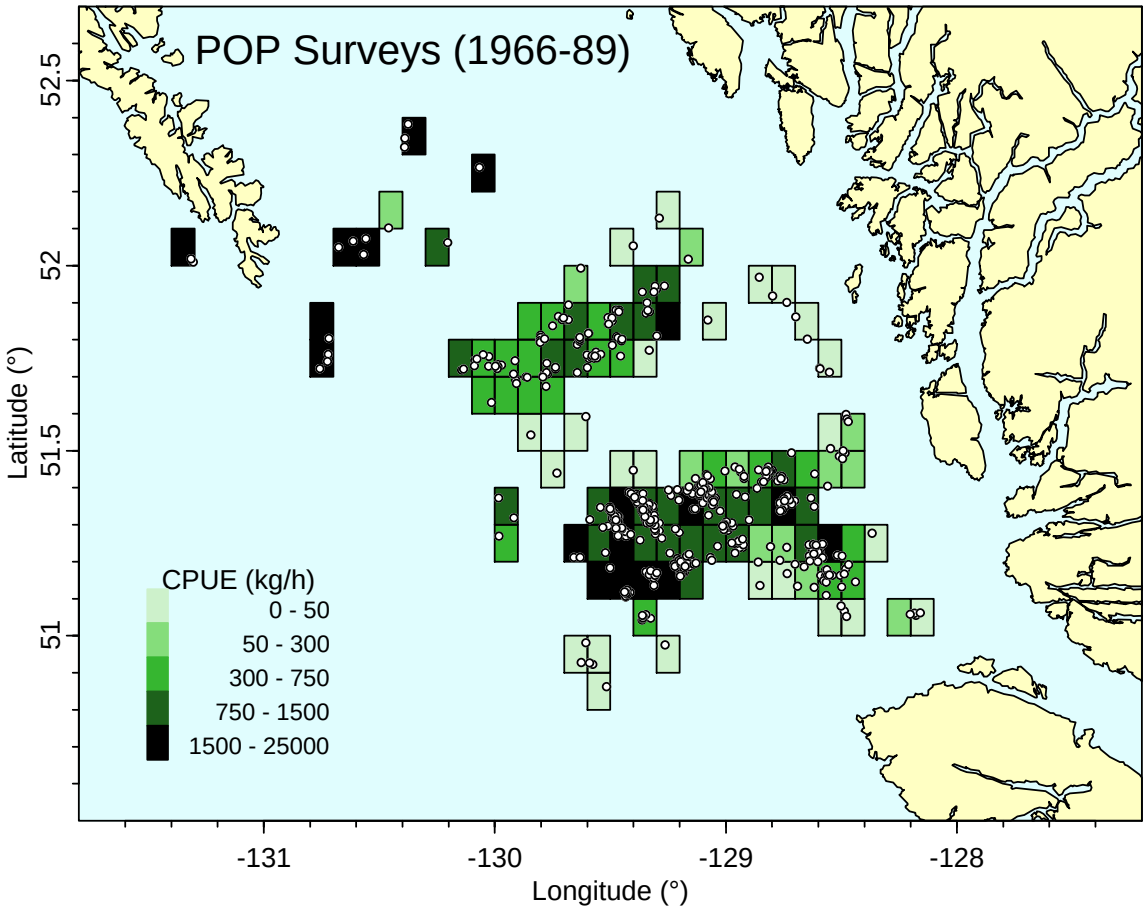

Figure 9. Portrayal of `surveyData` from Pacific ocean perch (*Sebastes alutus*) surveys in the central coast region of British Columbia from 1966-89, with shoreline data clipped from `nepacLL`. Colours portray the mean catch per unit effort (CPUE) within each grid cell (0.1º by 0.1º). Circles show locations of individual tows.

PBSmapping 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 10, 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 PDF file marked “Standard Tube map” at the web site http://www.tfl.gov.uk/gettingaround/1106.aspx) 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.

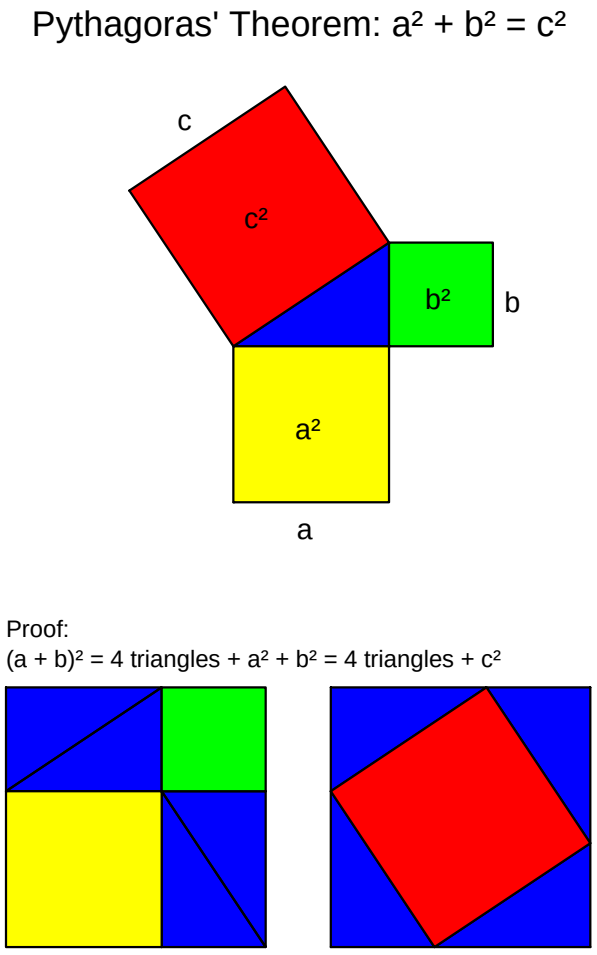

Figure 10. Proof of Pythagoras’ Theorem. A PolySet defines all geometric objects in this figure, and PolyData determine the colours for plotting. Four blue triangles plus the yellow square (a^2) and the green square (b^2) equal four blue triangles plus the red square (c^2); consequently, $a^2 + b^2 = c^2$.

2.7. Strengths, Limitations, and Alternatives

PBSmapping 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 and prefer 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.

In addition to their comprehensive shoreline database, Wessel and Smith have designed and released a free collection of Generic Mapping Tools (GMT; http://gmt.soest.hawaii.edu/) that provide a serious alternative to our software. These tools operate in the DOS/UNIX environment and support many more projections than **PBSmapping**. They also store polygons in a more efficient file format than our PolySet data frames. We designed **PBSmapping** for the R environment, with its rich support for statistical and mathematical analysis. We have also included numerous algorithms from computational geometry, such as `findPolys` and `joinPolys`. Readers may, however, find GMT more useful for map formats not supported in **PBSmapping**. Appendix C shows some comparative examples of code written in both environments.

Because **PBSmapping** includes features often supported by a Geographic Information System (GIS), a free GIS package might also provide an alternative to the software described here. The FreeGIS web site (http://www.freegis.org) summarizes the current status of free GIS programs and data. Their listings receive frequent updates and show a pattern of steady growth.

3. COMMAND LINE UTILITIES

The **PBSmapping** package for R includes several algorithms that we have also implemented as stand-alone command-line utilities. These can handle very large data sets that may be too large for the R working environment. Furthermore, some users may wish to implement computational geometry calculations without reference to the R 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. For each utility, a corresponding `.c` file provides a front end to shared code for the algorithms. Source code appears in the R library directory `\PBSmapping\Utils\`.

3.1. `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 the same number of fields per row as in the header line. All fields must be delimited by white space. 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 (e.g., `> file.txt`).

3.2. `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`) [`/m`] `/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);
- `/m` use metres instead of kilometres as UTM measurement;
- `/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.

3.3. `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* ASCII input file containing the PolySet (required);
- `/e` *EVENT_FILE* ASCII input file containing EventData (required);
- `/o` *OFILE* ASCII output file (defaults to standard output).

The header line in both input files must contain field names, and subsequent lines must contain the relevant fields of data 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 with three or four columns (`EID`, `PID`, `SID`, `Bdry`), where `SID` may be missing (Section 2.1). The default standard output can be redirected to a text file.

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 valuable shoreline and bathymetry databases compiled by Dr. Paul Wessel, Dr. Walter Smith, and Dr. D. T. Sandwell (Wessel and Smith 1996; Smith and Sandwell 1997). In particular, we thank Dr. Paul Wessel for permission to redistribute data from the GSHHS database. Code from other authors seriously enhances this version of **PBSmapping**. Dr. Gary Robinson has kindly allowed us to use his code for a stack-based Douglas-Peuker line simplification routine, implemented in our `thinPolys` function. Our colleague Brian Krishka helped prepare various data objects. The **PBSmapping** 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/.

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

Becker, R.A., and Wilks, A.R. (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 Wilks, A.R. (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/.

Boers, N.M., Haigh, R., and Schnute, J.T. (2004) PBS Mapping 2: developer's guide. *Canadian Technical Report of Fisheries and Aquatic Sciences* **2550**.

Bourke, P. (1988 July) Calculating the area and centroid of a polygon. URL: http://astronomy.swin.edu.au/~pbourke/geometry/polyarea/ Accessed Aug. 3, 2004. (website now defunct)

Chamberlain, R. (2001) Feb. Q5.1: what is the best way to calculate the distance between 2 points. URL: http://www.census.gov/cgi-bin/geo/gisfaq?Q5.1 Accessed Aug. 3, 2004. (website now defunct)

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (2000) Computational geometry: algorithms and applications: second edition. Springer: Berlin.

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 Peucker, T.K. (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer* **10**:112-22.

Environmental Systems Research Institute (ESRI). (1996) ArcView GIS: the geographic information system for everyone. ESRI Press, Redlands, CA.

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

Haigh, R., and Schnute, J. (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.

Murta, A. (2004) Jul 15. General polygon clipper homepage. URL: http://www.cs.man.ac.uk/~toby/alan/software/ Accessed Aug. 3, 2004.

Ordnance Survey. (2010) A guide to coordinate systems in Great Britain. Report D00659 (v2.1). Southampton, UK. URL: http://www.ordnancesurvey.co.uk/oswebsite/docs/support/guide-coordinate-systems-great-britain.pdf

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.

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., Boers, N.M., and Haigh, R. (2003) PBS Software: maps, spatial analysis, and other utilities. *Canadian Technical Report of Fisheries and Aquatic Sciences* **2496**: 82 p.

Schnute, J.T., Couture-Beil, A., and Haigh, R. (2006) PBS Modelling 1: user's guide. *Canadian Technical Report of Fisheries and Aquatic Sciences* **2674**: viii + 114 p.

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

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

Sinclair, C.A., and Olsen N. (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.

Smith, W.H.F., and Sandwell, D.T. (1997) Global seafloor topography from satellite altimetry and ship depth soundings. *Science* **277**: 1957-1962.

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.

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

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

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

Wikipedia. (2004) Earth radius. URL: http://en.wikipedia.org/wiki/Earth_radius Accessed Aug. 19, 2004.

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

APPENDIX A. PBSDATA PACKAGE

This appendix documents the objects available in the R-package **PBSdata**, which is not distributed on CRAN but remains available on Google Code: http://code.google.com/p/pbs-data/. Fisheries and Oceans personnel can also obtain the package from the PBS Intranet website: http://svbcpbsgfiis/sql/. Look for a link on the left entitled "Most recent PBS R Packages".

Table A1. Data sets available in **PBSdata**.

Object	Description
bctopo	Topo: British Columbia Sea Floor Topography
bgcp	Topo: Biogeochemical Provinces
claradat	Data: Tow Catches of Species in Queen Charlotte Sound
dbr.rem	Data: Annual Catches of Rockfish by Sector
eez.bc	Topo: Exclusive Economic Zone for BC Coast
fos.fid	Code: Fishery Codes in GFFOS
gear	Code: Gear Codes for Various DFO Databases
hsgrid	Topo: Hecate Strait Assemblage Survey Grid
hsisob	Topo: Hecate Strait Isobaths
hssa	Topo: Hecate Strait Survey Area
iphc.rbr	Data: Longline Indices of Rockfish Catch from the IPHC SSA
iphc.rer	Data: Longline Indices of Rockfish Catch from the IPHC SSA
iphc.yyr	Data: Longline Indices of Rockfish Catch from the IPHC SSA
isobath	Topo: Isobaths (100 to 1800 m, at 100 m intervals)
locality	Topo: Localities in Pacific Marine Fisheries Commission Minor Areas
ltea	Topo: Longspine Thornyhead Exploratory Management Areas
ltmose07	Topo: Longspine Thornyhead Fishing Grounds (WCVI)
ltmose12	Topo: Longspine Thornyhead Fishing Grounds (WCVI)
ltsa	Topo: Longspine Thornyhead Survey Strata (WCVI)
ltsa.bad	Topo: No-Trawl Zones in Longspine Thornyhead Survey Area
ltxa	Topo: Longspine Thornyhead Experimental Management Areas
major	Topo: Pacific Marine Fisheries Commission Major Areas
minor	Topo: Pacific Marine Fisheries Commission Minor Areas
nage394	Data: Age Frequency by Year for Rougheye Rockfish
orfhistory	Data: Historic Landings of Rockfish in BC
parVec	Data: Initial Parameter Vector for Model Fits
pcoda	Topo: Hecate Strait Pacific Cod Monitoring Survey Areas
pjsa	Code: Paul J Starr Locality Codes
pl230	Topo: 230 Degree True Line from Lookout Island
pmfc	Code: Pacific Marine Fisheries Commission Areas
pop.age	Data: Pacific Ocean Perch Age Data (5AB, 5CD)
pop.pmr.qcss	Data: Pacific Ocean Perch (p, mu, rho) for QCS Synoptic Survey
popa	Topo: Pacific Ocean Perch Population Areas
qcb	Topo: Queen Charlotte Basin Surficial Geology
qcssa	Topo: Queen Charlotte Sound Survey Strata
rca	Topo: Rockfish Conservation Areas
species	Code: Species Codes and Names (primarily for marine fisheries)

Object	Description
spn	Code: Species Code Vector
srfa	Topo: Slope Rockfish Assessment Areas
srfs	Topo: Slope Rockfish Assessment Subareas
testdatC	Data: Fisheries Catch Data with Species by Column
testdatR	Data: Fisheries Catch Data with Species by Row
trawlability	Topo: Fisher Knowledge of Towable Bottom
utilize	Code: Utilization Codes for Various DFO Databases
wchgsa	Topo: West Coast Haida Gwaii Survey Area
wcvisa	Topo: West Coast of Vancouver Island Survey Strata
ymr.rem	Data: Annual Catches of Rockfish by Sector

APPENDIX B. BATHYMETRY DATA

Smith and Sandwell (1997) have produced a global seafloor topography database from satellite altimetry and ship depth soundings. Using the web-based data acquisition form at http://topex.ucsd.edu/cgi-bin/get_data.cgi, users can extract a region from this database. The form returns an ASCII file containing X, Y, and Z coordinates. To use this data file with **PBSmapping**, first load it into R with the native function `read.table`, which creates a data frame with three fields. Our function `makeTopography` can convert this data frame to a list object with vectors `x` and `y` and an outer product matrix `z`, ready for use by the functions `contour` or `contourLines`. In particular, `contourLines` produces a list object that can be easily converted to a PolySet using `convCP`, which in turn produces a list object consisting of a PolySet (with contour coordinates) and PolyData (with the depth of each contour).

Example

Bathymetry for a small section of the Aleutian Islands, Alaska,
where a user would specify coordinates `xlim=c(-162,-158)` and `ylim=c(53,57)` in the web-based acquisition form referenced above, and save Topography to a file called `aleutian.txt` (also provided in the library directory `PBSmapping\extra\`).

```
require(PBSmapping);
isob <- c(100,500,1000,2500,5000);
icol <- rgb(0,0,seq(255,100,len=length(isob)),max=255);

afile <- paste(system.file(package="PBSmapping"),
         "/extra/aleutian.txt",sep="")
aleutian <- read.table(afile, header=F,col.names=c("x","y","z"))
aleutian$x <- aleutian$x - 360
aleutian$z <- -aleutian$z
alBathy <- makeTopography(aleutian)
alCL <- contourLines(alBathy,levels=isob)
alCP <- convCP(alCL)
alPoly <- alCP$PolySet
attr(alPoly,"projection") <- "LL"

plotMap(alPoly,type="n");
addLines(alPoly,col=icol);
data(nepacLL); addPolys(nepacLL,col="gold");
legend(x="topleft",bty="n",col=icol,lwd=2,legend=as.character(isob));
```

APPENDIX C. GENERIC MAPPING TOOLS (GMT)

Generic Mapping Tools (GMT) and **PBSmapping** have many similar features, although they operate in different environments. We built **PBSmapping** for the R statistical platform, whereas Wessel and Smith developed GMT to run as commands for the UNIX operating system. Each environment imposes limitations on its respective tools. The following discussion focuses on image types, one of the fundamental areas where the programs differ.

Images are commonly stored in two basic formats, raster and vector. The raster (or bit map) format uses a grid of squares, where each square is assigned characteristics like colour and transparency. The image's resolution, often measured in "dots per inch", determines the density of the grid. When this density is less than the resolution of the output device, the image may appear jagged because distinct squares are visible. Choosing a sufficiently high-resolution image for an output device may result in a large file size. The vector format stores coordinates for control points of lines, curves, and other shapes. Scaling algorithms use these coordinates to produce an image at any specified size with a consistently smooth appearance. In a mapping context, vector formats are usually preferred over raster formats.

Unlike R, the UNIX environment does not have native support for generating images. Wessel and Smith decided that GMT programs would output (optionally encapsulated) postscript files. This vector-based format is more popular in UNIX than Windows and is poorly supported by some word processors, such as Microsoft Word. On the other hand, **PBSmapping** inherits support from the R environment for common raster (e.g., `BMP`, `JPG`) and vector (e.g., `WMF`) file formats. Users of Windows operating systems may find **PBSmapping**'s output somewhat more convenient than that from GMT.

Converting GMT's postscript output to a better-supported graphics format can be achieved through the Ghostscript graphical user interface GSview (http://www.cs.wisc.edu/~ghost/gsview/). Through an option in GSview's "Edit" menu, the program converts `PS` files to the popular `EMF` and `WMF` vector formats. However, we obtained somewhat erratic results from this process and had greater success with raster images produced with the convert option in the "File" menu.

Figure C1 and Figure C2 compare **PBSmapping** with GMT. We show the code used to produce these images in both environments. Although one R command can span multiple lines, one GMT command cannot. For clarity, however, we span GMT commands across multiple lines in the listing below. In familiar UNIX notation, we indicate spanning by escaping the new-line character with a backslash (\).

Code for Figure C1

R: (Panel A)

```
data(nepacLL);                          # load the nepacLL data set
plotMap(nepacLL,                        # plot the nepacLL data set
      xlim=c(-129.3, -122.2),           # limit the region horizontally
      ylim=c(47.5, 51.5),               # limit the region vertically
      plt=c(0.16, 0.97, 0.16, 0.97),    # specify the plot region size
      col=rgb(255, 255, 195,            # set the foreground colour
              maxColorValue=255),
      bg=rgb(224, 253, 254,             # set the background colour
             maxColorValue=255),
      tck=c(-0.03),                     # set the tick mark length
      cex = 1.8,                        # adjust the font size
      mgp=c(1.9, 0.7, 0));              # adjust the axis label locations
```

GMT: (Panel B)

```
gmtset ANOT_FONT_SIZE = 26p             # set the annotation font size
pscoast -Dh \                           # plot the high resolution data set
    -A0/0/1 \                           # skip inner polygons (holes)
    -R-129.3/-122.2/47.5/51.5 \         # limit the region horizontally and vertically
    -JM7i \                             # use the Mercator projection, 7 inches wide
    -G255/255/195 \                     # set the foreground colour
    -S224/253/254 \                     # set the background colour
    -Ba2/a1WSne \                       # mark every 2 (X) and 1 (Y) degrees on W & S axes
    -W0.5p \                            # set the pen width to 0.5 points
    -P \                                # portrait mode
    > GMT-VI.ps                         # output to the postscript file GMT-VI.ps
```

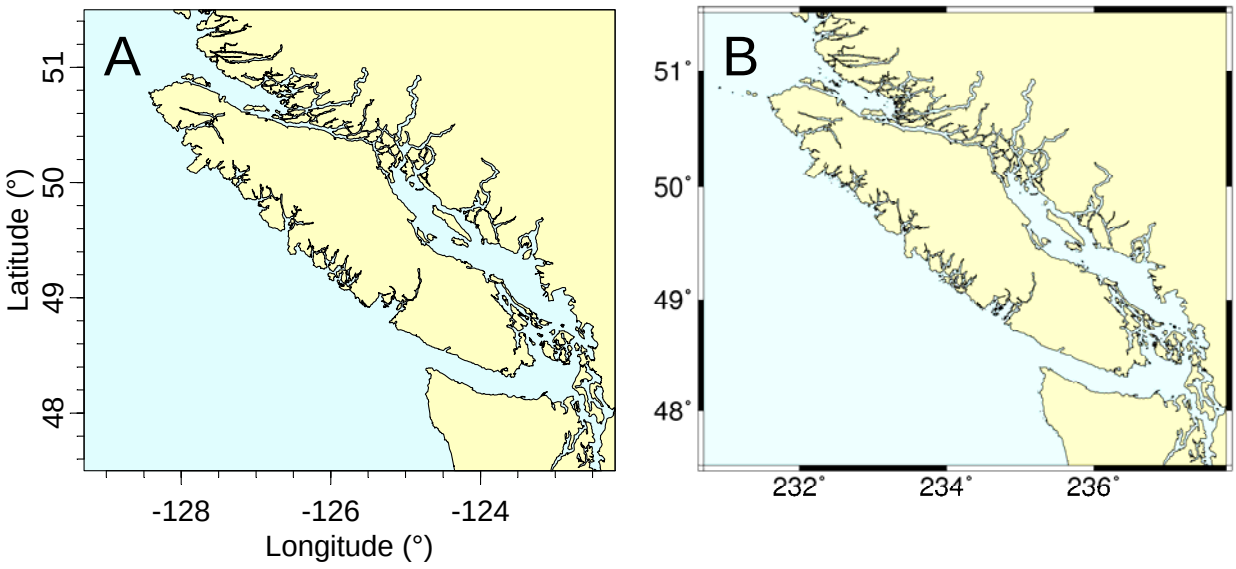

Figure C1. (A) Vancouver Island, as plotted in **PBSmapping**, compared with (B) the same region as output from GMT.

Code for Figure C2

R: (Panel A)

```
data(nepacLL);                          # load the nepacLL data set
plotMap(nepacLL,                        # plot the nepacLL data set
      xlim=c(-127.89, -125.68),         # limit the region horizontally
      ylim=c(47.85, 49.97),             # limit the region vertically
      plt=c(0.16, 0.97, 0.16, 0.97),    # specify the plot region size
      col=rgb(255, 255, 195,            # set the foreground colour
              maxColorValue=255),
      bg=rgb(224, 253, 254,             # set the background colour
              maxColorValue=255),
      tck=c(-0.03),                     # set the tick mark length
      cex=1.8,                          # adjust the font size
      mgp=c(1.9, 0.7, 0));              # adjust the axis label locations
data(towTracks);                        # load the towTracks data set
addLines(towTracks,                     # add the towTracks data set
      col=rgb(255, 0, 0,                # set the colour
              maxColorValue=255),
      lwd=0.5);                         # set the line width
```

GMT: (Panel B)

```
gmtset ANOT_FONT_SIZE = 20p             # set the annotation font size
pscoast -Dh \                           # plot the high resolution data set
      -R-127.89/-125.68/47.85/49.97 \   # limit the region horizontally and vertically
      -JM5i \                           # use the Mercator projection, 5 inches wide
      -G255/255/195 \                   # set the foreground colour
      -S224/253/254 \                   # set the background colour
      -Ba0.5/a0.5WSne \                 # mark every 0.5 (X) and 0.5 (Y) degrees on W & S axes
      -W0.5p \                          # set the pen width to 0.5 points
      -P \                              # portrait mode
      -K \                              # allow for appending more plot code
      > GMT-Tow.ps                      # output to the postscript file GMT-Tow.ps
psxy -R-127.89/-125.68/47.85/49.97 \    # limit the region
      -JM5i \                           # add using the Mercator projection, 5 inches wide
      -W0.5p/255/0/0 \                  # set the pen width to 0.5 points and set the colour
      -M \                              # ASCII file contains multiple polylines
      -H0 \                             # ASCII file does not contain a header
      -O \                              # overlay; lay plot on top of earlier one
      < GMT-Tow.txt \                   # input ASCII file GMT-Tow.txt
      >> GMT-Tow.ps                     # append output to the postscript file GMT-Tow.ps
```

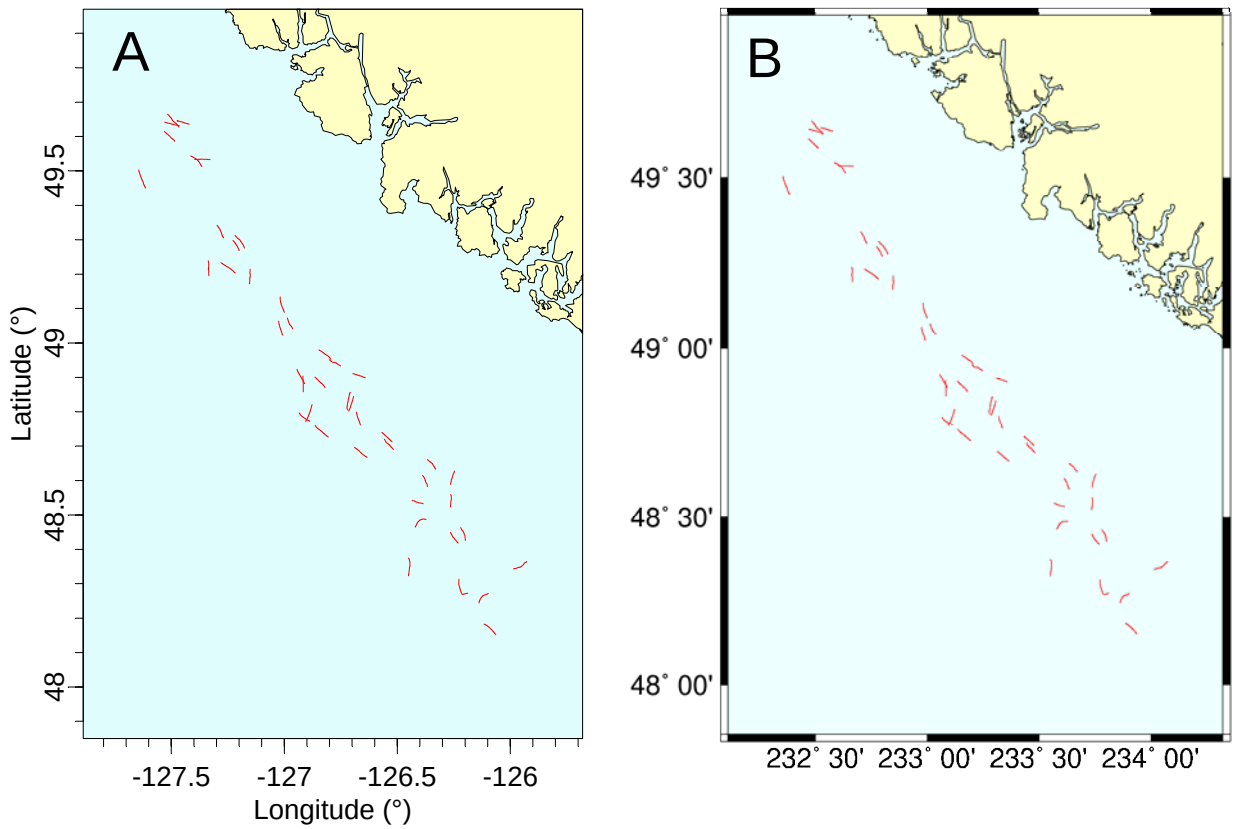

Figure C2. Tow tracks off the west coast of Vancouver Island drawn by (A) **PBSmapping** (B) GMT produced (B).

Format of `GMT-tow.txt`:

```
>                                   # a '>' signifies the start of each polyline
-126.26545 48.523133                # vertices follow: X coordinate, white space, Y coordinate
-126.265233 48.523716
-126.265183 48.524283
…
>
-126.385483 48.532567
-126.3861 48.5327
-126.3868 48.53285
…
```

APPENDIX D. SOURCE CODE FOR FIGURES

To help beginners use **PBSmapping**, we include source code for all figures in this report. A global function `.PBSclr` provides the colours for the examples, and default dots and dashes are provided by `.PBSdot` and `.PBSdash`, respectively. These objects are exported from the NAMESPACE and are globally available once **PBSmapping** is loaded.

Global colours, dots, and dashes

```
# Figures for PBSmapping examples (last modified: 2013-04-10)
#-------------------------------------------------------------
# Historical values for compatibilityy with S-Plus (defunct)
.PBSdot <- 3; .PBSdash <- 2
.PBSclr <- function(){
   PBSclr = list(             black=c(0,0,0),
      sea=c(224,253,254),     land=c(255,255,195),      red=c(255,0,0),
      green=c(0,255,0),       blue=c(0,0,255),          yellow=c(255,255,0),
      cyan=c(0,255,255),      magenta=c(255,0,255),     purple=c(150,0,150),
      lettuce=c(205,241,203), moss=c(132,221,124),      irish=c(54,182,48),
      forest=c(29,98,27),     white=c(255,255,255),     fog=c(223,223,223) )
      PBSclr <- lapply(PBSclr,function(v) {rgb(v[1],v[2],v[3],maxColorValue=255) })
      return(PBSclr) }
```

Figure 1 – World UTM Zones

```
.PBSfig01 <- function() { #  World UTM Zones
   clr <- .PBSclr()
   data(worldLL,nepacLL,envir=sys.frame(sys.nframe()))
   par(mfrow=c(1,1),omi=c(0,0,0,0)) #------Plot-the-figure------
   plotMap(worldLL, ylim=c(-90, 90), bg=clr$sea, col=clr$land, tck=-0.023,
           mgp=c(1.9, 0.7, 0), cex=1.2, plt=c(.08,.98,.08,.98))
   # add UTM zone boundaries
   abline(v=seq(-18, 360, by=6), lty=1, col=clr$red)
   # add prime meridian
   abline(v=0, lty=1, lwd=2, col=clr$black)
   # calculate the limits of the 'nepacLL' PolySet
   xlim <- range(nepacLL$X) + 360
   ylim <- range(nepacLL$Y)
   # create and then add the 'nepacLL' rectangle
   region <- data.frame(PID=rep(1,4), POS=1:4, X=c(xlim[1],xlim[2],xlim[2],xlim[1]),
                        Y=c(ylim[1],ylim[1],ylim[2],ylim[2]))
   region <- as.PolySet(region, projection="LL")
   addPolys(region, lwd=2, border=clr$blue, density=0)
   # add labels for some UTM zones
   text(x=seq(183.2, by=6, length=9), y=rep(85,9), adj=0.5, cex=0.65, label=1:9)
   box() }
```

Figure 2 – nepacLL UTM Zones in LL Space

```
.PBSfig02 <- function() {  # nepacLL UTM Zones in LL Space
   clr <- .PBSclr(); dot <- .PBSdot
   data(nepacLL,envir=sys.frame(sys.nframe()))
   par(mfrow=c(1,1),omi=c(0,0,0,0)) #------Plot-the-figure------
   plotMap(nepacLL, col=clr$land, bg=clr$sea, tck=-0.014,
           mgp=c(1.9,0.7,0), cex=1.2, plt=c(.08,.98,.08,.98))
   # add lines separating UTM zones
   utms <- seq(-186, -110, 6)
   abline(v=utms, col=clr$red)
   # add the central meridian of zone 6
   abline(v=-147, lty=dot, col=clr$black)
   # create and then add labels for the UTM zones
```

```
   cutm <- diff(utms) / 2
   nzon <- length(cutm)
   cutm <- cutm + utms[1:nzon]
   text(cutm,rep(50.75,nzon),c(60,1:(nzon-1)),cex=1.3,col=clr$red)
   box()  }
```

Figure 3 – nepacLL UTM Zones in UTM Space

```
.PBSfig03 <- function() {  # nepacLL UTM Zones in UTM Space
   clr <- .PBSclr(); dot <- .PBSdot
   data(nepacLL,envir=sys.frame(sys.nframe()))
   zone  <- 6;  xlim  <- range(nepacLL$X);  ylim <- range(nepacLL$Y)
   utms  <- seq(-186,-110,6)  #'utms' vector for creating PolySet and EventData below
   # create UTM zones
   lutms <- data.frame(PID=rep(1:length(utms), each=2),
               POS=rep(c(1,2), times=length(utms)), X=rep(utms,each=2),
               Y = rep(c(ylim[1], ylim[2]), times=length(utms)))
   lutms <- as.PolySet(lutms, projection="LL", zone=zone)
   lutms <- thickenPolys(lutms, tol=25, close=FALSE)
   uutms <- convUL(lutms)
   # create label locations (central meridians)
   lcms  <- data.frame(EID=1:(length(diff(utms)/2)),
               X=utms[1:(length(utms)-1)]+diff(utms)/2,
               Y=rep(50.75, length(diff(utms)/2)))
   lcms  <- as.EventData(lcms, projection="LL", zone=zone)
   ucms  <- convUL(lcms)
   nepacUTM <- nepacLL; attr(nepacUTM,"zone") <- zone  # convert to correct zone
   nepacUTM <- convUL(nepacUTM)
   par(mfrow=c(1,1),omi=c(0,0,0,0)) #------Plot-the-figure------
   plotMap(nepacUTM, col=clr$land, bg=clr$sea, tck=-0.017,
           mgp=c(1.9,0.7,0), cex=1.0, plt=c(0.07,0.97,0.07,0.98))
   addLines(uutms, col=clr$red)
   lines(x=c(500, 500),y=c(4100,7940),lty=dot,col=clr$black)
   text(ucms$X,ucms$Y,c(60,1:(length(utms)-2)),cex=1.3,col=clr$red)
   box()  }
```

Figure 4 – thinPolys on Vancouver Island

```
.PBSfig04 <- function() {  # thinPolys on Vancouver Island
   clr <- .PBSclr();
   data(nepacLL,envir=sys.frame(sys.nframe()))
   par(mfrow=c(1,2),omi=c(0,0,0,0)) #------Plot-the-figure------
   vi     <- nepacLL[nepacLL$PID==33,]
   xlim   <- range(vi$X) + c(-0.25, 0.25); ylim <- range(vi$Y) + c(-0.25, 0.25)
   # plot left figure (normal Vancouver Island)
   plotMap(vi, xlim, ylim, col=clr$land, bg=clr$sea, tck=-0.028,
           mgp=c(1.9,0.7,0), cex=1.0, plt=c(0.14,1.00,0.07,0.97))
   text(x=xlim[2]-0.5, y=ylim[2]-0.3, "A", cex=1.6)
   # plot right figure (thinned Vancouver Island)
   plotMap(thinPolys(vi, tol=10), xlim, ylim, col=clr$land, bg=clr$sea,
           tck=c(-0.028, 0), tckLab=c(TRUE, FALSE),
           mgp=c(1.9, 0.7, 0), cex=1.0, plt=c(0.00, 0.86, 0.07, 0.97))
   text(x=xlim[2]-0.5, y=ylim[2]-0.3, "B", cex=1.6)
   box() }
```

Figure 5 – joinPolys on Crescents

```
.PBSfig05 <- function() {  # joinPolys on Crescents
   clr <- .PBSclr(); dash <- .PBSdash
   radius <- c(5, 4)                 # two radii of the circles
   size   <- abs(diff(radius)) + 0.1 # size of crescent
   shiftB <- 3.5                     # distance to shift second crescent
   pts    <- 120                     # points in outer circle
   cex    <- 1.0                     # character expansion for labels
   off    <- 1.2                     # panel label offset
   xlim   <- c(0, radius[1]*2 + shiftB) + c(-1,1)
   ylim   <- c(0, radius[1]*2) + c(-2,1)
   Mmin   <- .10 # minimum OMI
   Rdin   <- par()$din[2]/par()$din[1]
   Rfig   <- (3*diff(ylim))/(2*diff(xlim))
   if (Rdin > Rfig) {
      width  <- par()$din[1] - 2 * Mmin
      height <- width * (3*diff(ylim))/(2*diff(xlim))
      Mmax   <- (par()$din[2] - height) / 2
      parOmi <- c(Mmax,Mmin,Mmax,Mmin) }
   else {
      height <- par()$din[2] - 2 * Mmin
      width  <- height * (2*diff(xlim))/(3*diff(ylim))
      Mmax   <- (par()$din[1] - width) / 2
      parOmi <- c(Mmin,Mmax,Mmin,Mmax) }
   polyA  <- list()
   for (i in 1:length(radius)) {
      polyA[[i]] <- as.PolySet(data.frame(PID=rep(1,pts), POS = 1:pts,
         X =radius[i]*cos(seq(0, 2*pi, len=pts)),
         Y =radius[i]*sin(seq(0, 2*pi, len=pts))), projection = 1)
      polyA[[i]][, c("X","Y")] <- polyA[[i]][, c("X","Y")] + radius[i] }
   # centre B within A
   polyA[[2]][,c("X","Y")] <- polyA[[2]][,c("X","Y")] + (radius[1]-radius[2])
   # shift B right
   polyA[[2]]$X <- polyA[[2]]$X + size
   # create 'polysA' and 'polysB'
   polyA  <- as.PolySet(joinPolys(polyA[[1]], polyA[[2]], operation="DIFF"),
projection=1)
   polyB  <- polyA
   polyB$X<- abs(polyB$X - (radius[1] * 2)) + shiftB
   par(mfrow=c(3,2),mai=c(0,0,0,0),omi=parOmi) #------Plot-the-figure------
   lab    <- list()
   lab$text <- c("Polygon A", "Polygon B", "A \"INT\" B","A \"UNION\" B",
                 "A \"DIFF\" B", "A \"XOR\" B")
   lab$cex <- rep(cex, 6);  lab$x <- rep(mean(xlim), 6);  lab$y <- rep(-0.8, 6)
   # panel A: polyA
   plotMap(polyA,xlim=xlim,ylim=ylim,xlab="",ylab="",axes=FALSE,col=clr$red,plt=NULL)
   text(lab$text[1], x=lab$x[1], y=lab$y[1], cex=lab$cex[1])
   text(xlim[1]+off, ylim[2]-off, "A", cex=1.6);  box()
   # panel B: polyB
   plotMap(polyB,xlim=xlim,ylim=ylim,xlab="",ylab="",axes=FALSE,col=clr$blue,plt=NULL)
   text(lab$text[2], x=lab$x[2], y=lab$y[2], cex=lab$cex[2])
   text(xlim[1]+off, ylim[2]-off, "B", cex=1.6);  box()
   # panels C to F
   ops    <- c(NA, NA, "INT", "UNION", "DIFF", "XOR")
   cols   <- c(NA, NA, clr$red, clr$purple, clr$red, clr$magenta)
   panel  <- c(NA, NA, "C", "D", "E", "F")
   for (i in 3:6) {

plotMap(NULL,xlim=xlim,ylim=ylim,projection=1,xlab="",ylab="",axes=FALSE,plt=NULL)
      addPolys(polyA, border=clr$red, lty=dash)
      addPolys(polyB, border=clr$blue, lty=dash)
      addPolys(joinPolys(polyA, polyB, operation=ops[i]), col=cols[i])
```

```
      text(lab$text[i], x=lab$x[i], y=lab$y[i], cex=lab$cex[i])
      text(xlim[1]+off, ylim[2]-off, panel[i], cex=1.6);  box();  } }
```

Figure 6 – contourLines in Queen Charlotte Sound

```
.PBSfig06 <- function() {  # contourLines in Queen Charlotte Sound
   clr <- .PBSclr();
   data(nepacLL,bcBathymetry,envir=sys.frame(sys.nframe()));
   isob   <- contourLines(bcBathymetry, levels=c(250, 1000))
   p      <- convCP(isob)
   attr(p$PolySet,"projection") <- "LL"
   p$PolyData$col <- rep(c(clr$red, clr$green, clr$blue, clr$yellow,
      clr$cyan, clr$magenta, clr$fog), length=nrow(p$PolyData))
   xlim   <- c(-131.8382, -128.2188)
   ylim   <- c(50.42407, 53.232476)
   region <- clipPolys(nepacLL, xlim=xlim, ylim=ylim)
   par(mfrow=c(1,1),omi=c(0,0,0,0)) #-----Plot-the-figure------
   plotMap(region, xlim=xlim, ylim=ylim, col=clr$land, bg=clr$sea, tck=-0.02,
        mgp=c(2,.75,0), cex=1.2, plt=c(.08,.98,.08,.98))
   addLines(p$PolySet, polyProps=p$PolyData, lwd=3)
   box()  }
```

Figure 7 – towTracks from Longspine Thornyhead Survey

```
.PBSfig07 <- function() {  # towTracks from Longspine Thornyhead Survey
   clr <- .PBSclr();
   data(nepacLL,towTracks,towData,envir=sys.frame(sys.nframe()));
   # add a colour column 'col' to 'towData'
   pdata  <- towData;  pdata$Z <- pdata$dep
   pdata  <- makeProps(pdata, breaks=c(500,800,1200,1600), "col",
                       c(clr$black, clr$red, clr$blue))
   par(mfrow=c(1,1),omi=c(0,0,0,0)) #------Plot-the-figure------
   plotMap(nepacLL, col=clr$land, bg=clr$sea, xlim=c(-127.8,-125.5), ylim=c(48,49.8),
      tck=-0.01, mgp=c(2,.5,0), cex=1.2, plt=c(.08,1,.08,.98))
   addLines(towTracks, polyProps=pdata, lwd=3)
   # right-justify the legend labels
   temp <- legend(x=-127.6, y=48.4, legend=c(" "," "," "), lwd=3, bty="n",
      text.width=strwidth("1200-1600 m"), col=c(clr$black,clr$red,clr$blue))
   text(temp$rect$left+temp$rect$w, temp$text$y,
      c("500-800 m", "800-1200 m", "1200-1600 m"), pos=2)
   text(temp$rect$left+temp$rect$w/2,temp$rect$top,pos=3,"LTS Survey Tracks");
   text(-125.6,49.7,"Vancouver\nIsland",cex=1.2,adj=1)
   box()  }
```

Figure 8 – calcArea of the Southern Gulf Islands

```
.PBSfig08 <- function() {  # calcArea of the Southern Gulf Islands
   clr <- .PBSclr();
   data (nepacLLhigh,envir=sys.frame(sys.nframe()))
   xlim   <- c(-123.6, -122.95); ylim <- c(48.4, 49); zone <- 9
   # assign 'nepacLLhigh' to 'nepacUTMhigh' (S62) and change to UTM coordinates
   nepacUTMhigh <- nepacLLhigh;  attr(nepacUTMhigh,"zone" ) <- zone
   nepacUTMhigh  <- convUL(nepacUTMhigh)
   # convert limits to UTM
   temp   <- data.frame(PID=1:4,POS=rep(1,4),X=c(xlim,xlim),Y=c(ylim,rev(ylim)))
   temp   <- convUL(as.PolySet(temp, projection="LL", zone=zone))
   xlim   <- range(temp$X); ylim <- range(temp$Y)
   # prepare areas
   isles  <- clipPolys(nepacUTMhigh,xlim,ylim)
   areas  <- calcArea(isles);
   # PIDs and labels for Gulf Islands
   bigPID <- areas[rev(order(areas$area)),][c(2:4,6:8),"PID"];
```

```
   labelData <- data.frame(PID = bigPID,
      label=c("Saltspring","San Juan","Galiano","Saturna","N Pender","Mayne"))
   labelData <- merge(labelData, areas, all.x=TRUE)
   labelData$label <- paste(as.character(labelData$label),
      round(labelData$area), sep="\n")
   par(mfrow=c(1,1),omi=c(0,0,0,0)) #------Plot-the-figure------
   plotMap(isles, col=clr$land, bg=clr$sea, tck=-.010,
      mgp=c(1.9,.7,0), cex=1, plt=c(.07,.98,.07,.98))
   # add the highlighted Gulf Islands
   bigisles <- isles[is.element(isles$PID,labelData$PID),]
   addPolys(bigisles,col=clr$yellow)
   labXY  <- calcCentroid(isles)
   labXY$Y<- labXY$Y + 2               # centre vertically
   labelData <- merge(labelData, labXY, all.x = TRUE)
   attr(labelData,"projection") <- "UTM"
   addLabels(labelData, placement="DATA", cex=1.25)
   text(898,5385,"Vancouver Island",adj=0, cex=1.25)
   text(925,5435,"Strait of Georgia",adj=0, cex=1.25)  }
```

Figure 9 – combineEvents in Queen Charlotte Sound

```
.PBSfig09 <- function() {  # combineEvents in Queen Charlotte Sound
   clr <- .PBSclr();
   data(nepacLL,surveyData,envir=sys.frame(sys.nframe()));
   events <- surveyData
   xl     <- c(-131.8, -127.2);  yl <- c(50.5, 52.7)
   # prepare EventData; clip it, omit NA entries, and calculate CPUE
   events <- events[events$X >= xl[1] & events$X <= xl[2] &
                    events$Y >= yl[1] & events$Y <= yl[2], ]
   events <- na.omit(events)
   events$cpue <- events$catch/(events$effort/60)
   # make a grid for the Queen Charlotte Sound
   grid   <- makeGrid(x=seq(-131.6,-127.6,.1), y=seq(50.6,52.6,.1),
                    projection="LL", zone=9)
   # locate EventData in grid
   locData<- findCells(events, grid)
   events$Z <- events$cpue
   pdata  <- combineEvents(events, locData, FUN=mean)
   brks   <- c(0,50,300,750,1500,25000); lbrks <- length(brks)
   cols   <- c(clr$lettuce, clr$moss, clr$irish, clr$forest, clr$black)
   pdata  <- makeProps(pdata, brks, "col", cols)
   par(mfrow=c(1,1),omi=c(0,0,0,0)) #------Plot-the-figure------
   plotMap(nepacLL, col=clr$land, bg=clr$sea, xlim=xl, ylim=yl, tck=-0.015,
           mgp=c(2,.5,0), cex=1.2, plt=c(.08,.98,.08,.98))
   addPolys(grid, polyProps=pdata)
   for (i in 1:nrow(events)) {
      # plot one point at a time for clarity
      points(events$X[i], events$Y[i], pch=16,cex=0.50,col=clr$white)
      points(events$X[i], events$Y[i], pch=1, cex=0.55,col=clr$black) }
   yrtxt  <- paste("(",min(events$year),"-",
                substring(max(events$year),3),")",sep="")
   text(xl[1]+.5,yl[2]-.1,paste("POP Surveys",yrtxt),cex=1.2,adj=0)
   # add a legend; right-justify the legend labels
   temp <- legend(x=xl[1]+.3, y=yl[1]+.7, legend = rep(" ", 5),
              text.width=strwidth("1500 - 25000"), bty="n", fill=cols)
   text(temp$rect$left + temp$rect$w, temp$text$y, pos=2,
        paste(brks[1:(lbrks-1)],brks[2:lbrks], sep=" - "))
   text(temp$rect$left+temp$rect$w/2,temp$rect$top,pos=3,"CPUE (kg/h)",cex=1);  }
```

Figure 10 – Pythagoras' Theorem Visualized

```
.PBSfig10 <- function() {  # Pythagoras' Theorem Visualized
   clr <- .PBSclr();
   data(pythagoras,envir=sys.frame(sys.nframe()))
   # create properties for colouring the polygons
   pythProps <- data.frame(PID=c(1, 6:13, 4, 15, 3, 5, 2, 14),
                  Z=c(rep(1, 9), rep(2, 2), rep(3, 2), rep(4, 2)))
   pythProps <- makeProps(pythProps, c(0, 1.1, 2.1, 3.1, 4.1), "col",
                  c(clr$blue, clr$red, clr$yellow, clr$green))
   par(mfrow=c(1,1),omi=c(0,0,0,0)) #------Plot-the-figure------
   plotMap(pythagoras, plt=c(.01,.99,.01,.95), lwd=2,
      xlim=c(.09,1.91), ylim=c(0.19,2.86), polyProps=pythProps,
      axes=FALSE, xlab="", ylab="", main="Pythagoras' Theorem: a\262 + b\262 = c\262")
   text(x = 0.1, y = 1.19, adj=0, "Proof:")
   text(x = 0.1, y = 1.10, adj=0,
      "(a + b)\262 = 4 triangles + a\262 + b\262 = 4 triangles + c\262")
   labels <- data.frame(X=c(1.02,1.66,0.65),Y=c(1.50,2.20,2.76),label=c("a","b","c"))
   text(labels$X, labels$Y, as.character(labels$label), cex=1.2)
   text(1.03, 1.81, "a\262", cex=1.2, col=clr$black)
   text(1.43, 2.21, "b\262", cex=1.2, col=clr$black)
   text(0.87, 2.46, "c\262", cex=1.2, col=clr$black)  }
```

Run command file “PBSfigs.r”

```
.PBSfigs <- function(nfigs=1:10) { # Draw all figures with numbers in nfigs
   #while (!is.null(dev.list())) dev.off(dev.cur())
   for (i in nfigs) {
      figStr <- paste(".PBSfig",ifelse(i<10,"0",""),i,sep="")
      get(figStr)();
      cat(figStr); readline(); }  }
```

APPENDIX E. PBSMAPPING FUNCTION DEPENDENCIES

This appendix documents function dependencies within **PBSmapping**. All functions appear as underlined entries in the alphabetic 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 working environment, but functions designated '**(C)**' are implemented in C source code and compiled in the DLL for the mapping package. R invokes these functions with the call `.C(…)`. Functions designated '**(S)**' exist as subfunctions only within the R function.

.addAxis

.addAxis2

.addBubblesLegend

.addCorners
calcConvexHull

.addFeature
.addProps
.validatePolyProps

.addLabels

.addProps

.calcDist

.calcOrientation
calcOrientation **(C)**

.checkClipLimits

.checkProjection

.checkRDeps

.clip
clip **(C)**
extractPolyData

.closestPoint

.createFastIDdig

.createGridIDs

.createIDs
.createFastIDdig

.expandEdges
.closestPoint
calcConvexHull

.fixGSHHSWorld
findPolys
fixPOS

.getBasename

.getGridPars
makeGrid

.initPlotRegion

.insertNAs

.mat2df

.plotMaps
.addAxis
.addLabels
.initPlotRegion
.validateXYData
addLines
addPoints
addPolys

.preparePolyProps
.createIDs
.validatePolyData

.rollupPolys
rollupPolys **(C)**

.validateData
.createIDs

.validateEventData
.validateData

.validateLocationSet
.validateData

.validatePolyData
.validateData

.validatePolyProps
.validateData

.validatePolySet
.validateData

.validateXYData
.validateData

addBubbles
.addBubblesLegend
.validateEventData

addLabels
.addFeature
.checkProjection
.validateEventData
.validatePolyData
.validatePolySet
calcCentroid
calcMidRange
calcSummary
is.EventData
is.PolyData

addLines
.addProps
.checkProjection
.clip
.createFastIDdig
.createIDs
.preparePolyProps
.validatePolyProps
.validatePolySet
is.PolyData

addPoints
.addFeature
.checkProjection
.validateEventData
.validatePolyData
is.PolyData

addPolys
.addProps
.checkProjection
.clip
.createFastIDdig
.createIDs
.preparePolyProps
.rollupPolys
.validatePolyProps
.validatePolySet
is.PolyData

addStipples
.addFeature
.checkProjection
.clip
.validatePolySet
findPolys
is.PolyData
thickenPolys

appendPolys
.validatePolySet
is.PolySet

as.EventData
.validateEventData
is.EventData

as.LocationSet
.validateLocationSet
is.LocationSet

as.PolyData
.validatePolyData
is.PolyData

as.PolySet
.validatePolySet
is.PolySet

calcArea
.rollupPolys
.validatePolySet
calcArea **(C)**
convUL
is.PolyData

calcCentroid
.rollupPolys
.validatePolySet
calcCentroid **(C)**
is.PolyData

calcConvexHull
.validateXYData
grDevices::chull
is.PolySet

calcLength
.validatePolySet
.rollupPolys
.calcDist
.createIDs

calcMidRange
.validatePolySet
calcSummary
is.PolyData

calcSummary
.createIDs
.rollupPolys
.validatePolySet
is.PolyData

calcVoronoi
.checkRDeps
.validateXYData
deldir::deldir
.addCorners
.expandEdges

clipLines
.clip
.validatePolySet
is.PolySet

clipPolys
.clip
.validatePolySet
is.PolySet

closePolys
.validatePolySet
closePolys **(C)**
is.PolySet

combineEvents
.validateEventData
is.PolyData

combinePolys
.validatePolySet
.createIDs

convCP
is.PolyData

convDP
.validatePolyData
is.PolySet

convLP
.validatePolySet
is.PolySet

convUL
.validateXYData
convUL **(C)**

dividePolys
.validatePolySet
.createIDs

extractPolyData
.createIDs
.validatePolySet
is.PolyData

findCells
.validateEventData
.validatePolySet
findCells **(C)**
is.LocationSet

findPolys
.validateEventData
.validatePolySet
findPolys **(C)**
is.LocationSet

fixBound
.validatePolySet
is.PolySet

fixPOS
.rollupPolys
.validatePolySet
is.PolySet

importEvents
as.EventData

importGSHHS
checkClipLimits
importGSHHS **(C)**

importLocs
as.LocationSet

importPolys
as.PolySet
as.PolyData

importShapefile
.checkRDeps
.getBasename
maptools:Rshapeget **(C)**
.calcOrientation
foreign:read.dbf

is.EventData
.validateEventData

is.LocationSet
.validateLocationSet

is.PolyData
.validatePolyData

is.PolySet
.validatePolySet

isConvex
.validatePolySet
is.PolyData
isConvex **(C)**

isIntersecting
.validatePolySet
is.PolyData
isIntersecting **(C)**

joinPolys
.validatePolySet
is.PolySet
joinPolys **(C)**

locateEvents
is.EventData

locatePolys
.validatePolyData
is.PolySet

makeGrid
is.PolySet

makeProps
.validatePolyData
is.PolyData

makeTopography

placeHoles
.calcOrientation
.checkRDeps

plotLines
.plotMaps
is.PolyData

plotMap
.plotMaps
is.PolyData

plotPoints
.plotMaps
is.PolyData

plotPolys
.plotMaps
is.PolyData

print.EventData
summary.EventData

print.LocationSet
summary.LocationSet

print.PolyData
summary.PolyData

print.PolySet
summary.PolySet

print.summary.PBS

refocusWorld
.createIDs
.shiftRegion **(S)**
.validatePolySet

summary.EventData

summary.LocationSet
.createIDs

summary.PolyData
.createIDs

summary.PolySet
.createIDs

thickenPolys
.calcDist
.createIDs
.validatePolySet
is.PolySet
thickenPolys **(C)**

thinPolys
.validatePolySet
is.PolySet
thinPolys **(C)**

APPENDIX F. PBSMAPPING FUNCTIONS AND DATA

This appendix documents the objects (functions and data) available in **PBSmapping**. Subsequent pages give indexed technical documentation for every object generated from `*.Rd` files written for the R documentation system. The package **PBSmodelling** includes a directory called `PBStools\` that contains useful batch files for building R packages, including the creation of the indexed manual included after Table F1.

Table F1. Functions and data sets in **PBSmapping**, arranged alphabetically within categories.

Category	Object	Description
User constant	`PBSprint`	Specify whether to print summaries
Import functions	`importEvents`	Import a text file and convert into EventData
	`importLocs`	Import a text file and convert into a LocationSet
	`importPolys`	Import a text file and convert into a PolySet
	`importGSHHS`	Import data from a GSHHS database
	`importShapefile`	Import an ESRI shapefile
Plotting functions	`addBubbles`	Add bubbles to maps
	`addLabels`	Add labels to an existing plot
	`addLines`	Add a PolySet to an existing plot as polylines
	`addPoints`	Add EventData/PolyData to an existing plot as points
	`addPolys`	Add a PolySet to an existing plot as polygons
	`addStipples`	Add stipples to an existing plot
	`plotLines`	Plot a PolySet as polylines
	`plotMap`	Plot a PolySet as a map
	`plotPoints`	Plot EventData/PolyData as points
	`plotPolys`	Plot a PolySet as polygons
Computational functions	`appendPolys`	Append a two-column matrix to a PolySet
	`calcArea`	Calculate the areas of polygons
	`calcCentroid`	Calculate the centroids of polygons
	`calcConvexHull`	Calculate the convex hull for a set of points
	`calcLength`	Calculate the length of polylines
	`calcMidRange`	Calculate midpoints of the `X` and `Y` ranges for polygons
	`calcSummary`	Apply functions to polygons in a PolySet
	`calcVoronoi`	Calculate Voronoi tesselation for a set of points
	`clipLines`	Clip a PolySet as polylines
	`clipPolys`	Clip a PolySet as polygons
	`closePolys`	Close a PolySet
	`combineEvents`	Combine measurements of events in same polygon
	`combinePolys`	Combine several polygons into a single polygon
	`convCP`	Convert results from `contourlines` into PolySet
	`convDP`	Convert EventData/PolyData into a PolySet
	`convLP`	Convert polylines into a polygon
	`convUL`	Convert coordinates between UTM/LL projections
	`dividePolys`	Divide a single polygon into several polygons
	`extractPolyData`	Extract PolyData from a PolySet

Category	Object	Description
	findCells	Find cells in a grid that contain events in EventData
	findPolys	Find polygons that contain events in EventData
	fixBound	Fix the boundary points of a PolySet
	fixPOS	Fix the POS column of a PolySet
	isConvex	Determine whether polygons are convex
	isIntersecting	Determine whether polygons are self-intersecting
	joinPolys	Join one or two PolySets using a set theoretic operation
	locateEvents	Locate events on the current plot
	locatePolys	Locate polygons on the current plot
	makeGrid	Make a grid of polygons
	makeProps	Make polygon properties
	makeTopography	Make topography data from freely available online data
	placeHoles	Place holes under correct solids
	refocusWorld	Refocus the worldLL / worldLLhigh data sets
	thickenPolys	Thicken a PolySet of polygons
	thinPolys	Thin a PolySet of polygons
Object-related functions	as.	Coerce a data frame to an object with class:
	EventData	EventData
	LocationSet	LocationSet
	PolyData	PolyData
	PolySet	PolySet
	is.	Determine whether an object is:
	EventData	EventData
	LocationSet	a LocationSet
	PolyData	PolyData
	PolySet	a PolySet
	print.	Print:
	EventData	an EventData object
	LocationSet	a LocationSet object
	PolyData	a PolyData object
	PolySet	a PolySet object
	summary.PBS	the summary of a PBSmapping object
	summary.	Summarize:
	EventData	EventData
	LocationSet	a LocationSet
	PolyData	PolyData
	PolySet	a PolySet
Data sets	bcBathymetry	Bathymetry data spanning British Columbia's coast
	nepacLL	Northeast Pacific shoreline (normal resolution)
	nepacLLhigh	Northeast Pacific shoreline (high resolution)
	pythagoras	Pythagoras' theorem diagram PolySet
	surveyData	Survey data
	towData	Tow data
	towTracks	Tow track polyline data
	worldLL	World ocean shoreline (normal resolution)
	worldLLhigh	World ocean shoreline (high resolution)