CORRECTED CONVERSION ALGORITHMS FOR THE CALCOFI STATION GRID AND THEIR IMPLEMENTATION IN SEVERAL COMPUTER LANGUAGES

EDWARD D. WEBER, THOMAS J. MOORE
NOAA Fisheries
Southwest Fisheries Science Center
8604 La Jolla Shores Drive
La Jolla, California 92037-1508
ph: 858-546-5676
ed.weber@noaa.gov

ABSTRACT

Converting between geographic coordinates in latitude and longitude and the line and station sampling pattern of the California Cooperative Fisheries Investigations (CalCOFI) program is a commonly required task for conducting research on the California Current ecosystem. This note presents several corrections and clarifications to the previously published algorithms for performing these conversions. We include computer code to implement the algorithms in Java™[1], Perl, Python, and R. We note that freely available code to conduct the conversions in Fortran, Matlab®[2], JavaScript™, and Visual Basic®[3] has previously been published, and an online conversion tool is also available. A future version of the PROJ.4 cartographic projections library will also include support for CalCOFI conversions, thereby allowing for convenient conversions using the GRASS GIS, PostGIS, Python, Perl, R, and many other programs and programming languages.

INTRODUCTION

The California Cooperative Fisheries Investigations (CalCOFI) line and station sampling pattern has been used for more than 60 years by the CalCOFI program. The pattern is also currently used by the Investigaciones Mexicanas de la Corriente de California program offshore of Baja California, the California Current Ecosystem Long Term Ecological Research site, and several other research programs in the California Current system. Researchers commonly need to convert between geographic coordinates in latitude and longitude and the CalCOFI line and station sampling pattern to plot their data, merge them with other environmental data, and interpret results. Algorithms for converting from geographic coordinates to CalCOFI line and station grid coordinates, and vice versa, have been published by Eber and Hewitt 1979. However, the article contained a few typographical errors. Some, but not all, of these errors were published as errata by Thombley 2006. Here we list the complete and corrected conversion algorithms for clarity and note changes from Eber and Hewitt 1979. For convenience, we provide computer code that implements these algorithms in Java, Perl, Python, and R. We also list sources of freely available code in Fortran (developed by D. Newton, University of California, Scripps Institution of Oceanography [SIO]), Matlab and JavaScript (developed by R. Thombley, SIO), and Visual Basic (developed by R. Charter, NOAA Southwest Fisheries Science Center). An upcoming release of the PROJ.4 cartographic projection library that will also allow users to easily make CalCOFI conversions in a number of computer programs and languages.

[1]Java™ and JavaScript™ are registered trademarks of Oracle Corporation
[2]Matlab® is a registered trademark of MathWorks, Inc.
[3]Visual Basic® is a registered trademark of Microsoft, Inc.

THE CALCOFI STATION SAMPLING PATTERN

The CalCOFI sampling station pattern (fig. 1) consists of a grid rotated –30° off the meridian so that it is approximately normal to the coast offshore of California and Baja California. Cardinal lines are located 120 nm apart and increase in numbering by increments of 10 from northwest to southeast. Ordinal lines are located 40 nm apart along this axis and are numbered by increments of 3.333, which are rounded to the nearest tenth by convention. The station numbering along these lines increases from northeast to southwest (i.e., inshore to offshore) and whole number increments are 4 nm apart. The rotation point is located at 34.15°N, –121.15°W (34°09'N, 121°09'W), which is defined as CalCOFI line 80, station 60. The CalCOFI sampling pattern and its historical evolution are described in greater detail by California Academy of Sciences et al. 1950; i.e., CalCOFI 1950 and Eber and Hewitt 1979. Conversions between the two coordinate systems are accomplished using a Mercator transform and some basic trigonometry. By convention, CalCOFI coordinates are calculated using Clarke's spheroid of 1866.

CONVERSIONS

CalCOFI Line and Station Coordinates to Geographical Coordinates

Conversions are accomplished using the Mercator transform to convert units along the Y-axis from latitude to meridional parts so that positions may be calcu-

lated using trigonometry on a plane (cf., Snyder 1987). We follow the notation of Eber and Hewitt 1979 for the Mercator transform and all other equations in this note. The Mercator transform (*MCTR*) is calculated using the following function:

$$(1)\quad MCTR(LA) = \frac{180}{\Pi} * \left(LN\left(tan\left(\frac{\Pi}{\mathbf{180}} * \left(45 + \frac{LA}{2}\right)\right)\right) - \mathbf{0.00676866} * sin\left(\frac{\Pi}{\mathbf{180}} * LA\right)\right),$$

where LA is the latitude to be transformed. This corresponds to the equation on p. 135 of Eber and Hewitt 1979. Differences between the corresponding equations presented by Eber and Hewitt 1979 and in this manuscript are listed in bold type throughout the article. The first change to equation 1 was made because there was a typographical error in the squared eccentricity of the ellipsoid in the original equation, which should read 0.00676866 (Snyder 1987). The second change is a clarification that we have made to all equations. We assume the trigonometric functions (*sin, cos, tan, atan*) operate on units of radians, as they do on most modern computer languages. So we have explicitly added conversions from degrees to radians, and vice versa, where they are needed.

To calculate the geographical coordinates of point *P*, the following algorithm is used corresponding to equations 1–5 at the top of p. 137 of Eber and Hewitt 1987.

$$(2)\quad RLA = \mathbf{34.15} - 0.2 * (PLN - 80) * cos\left(\frac{\Pi}{\mathbf{180}} * 30\right)$$

$$(3)\quad PLA = RLA - \frac{(PSN - 60)}{15} * sin\left(\frac{\Pi}{\mathbf{180}} * 30\right)$$

$$(4)\quad L1 = \left(MCTR(\boldsymbol{PLA}) - MCTR\ (34.15)\right) * tan\left(\frac{\Pi}{\mathbf{180}} * 30\right)$$

$$(5)\quad L2 = \frac{\left(MCTR(RLA) - MCTR\ (PLA)\right)}{\left(cos\left(\frac{\Pi}{\mathbf{180}} * 30\right) * sin\left(\frac{\Pi}{\mathbf{180}} * 30\right)\right)}$$

$$(6)\quad PLO = L1 + L2 + 121.15,$$

where:

PLA = latitude of point *P*, *PLO* = longitude of *P, PLN* = line number of *P*, *PSN* = station number of *P*, and *RLA* = latitude of reference point *R* (fig. 1). An assumption of these algorithms is that positions are all located in the northwestern hemisphere. It is usually desirable to express longitude in the western hemisphere either as a negative number or as degrees east of the prime meridian for graphing and analysis, e.g., –121° or 239 rather than 121°W. Thus, equation 6 could be substituted with:

$$(7)\quad PLO = -1 * (L1 + L2 + 121.15),$$

to obtain a negative number or

$$(8)\quad PLO = -1 * (L1 + L2 + 121.15) + 360$$

to obtain a positive number greater than 180°

Geographical Coordinates to CalCOFI Line and Station Coordinates

Converting from geographical coordinates to CalCOFI line and station requires an inverse Mercator transform, which is solved iteratively using the following equation, which corresponds to the algorithm of six steps in the middle of p. 137 of Eber and Hewitt 1979:

$$(9)\quad LA = 2 * \left(atan\left(e^{MCTR\left(\frac{\Pi}{\mathbf{180}} * LA\right) + 0.00676866 * sin\left(\frac{\Pi}{\mathbf{180}} * LA\right)}\right) * \frac{\mathbf{180}}{\Pi} - 45\right)$$

The initial value of *LA* is set to *MCTR*(*LA*), and three iterations are recommended.

The algorithm for conversion from geographical coordinates to CalCOFI line and station also assumes that longitude is expressed in degrees west of the prime meridian, as described above. If geographical coordinates are expressed as negative numbers or degrees east, the following algorithm should be applied before calculating line and station:

$$(10)\quad \text{If } PLO > 180,\ PLO = (PLO - 360)$$

$$(11)\quad \text{If } PLO < 0,\ PLO = -1 * PLO$$

Line and station may then be calculated using the following equations, which correspond to equations 1–6 on lower p. 137 of Eber and Hewitt 1979.

$$(12)\quad L1 = \left(MCTR(PLA) - MCTR\ (34.15)\right) * tan\left(\frac{\Pi}{\mathbf{180}} * 30\right)$$

$$(13)\quad L2 = PLO - L1 - \mathbf{121.15}$$

$$(14)\quad MCTR(RLA) = L2 * cos\left(\frac{\Pi}{\mathbf{180}} * 30\right) * sin\left(\frac{\Pi}{\mathbf{180}} * 30\right) + MCTR(PLA)$$

$$(15)\quad RLA = \text{INVERSE}\left(MCTR(RLA)\right),$$

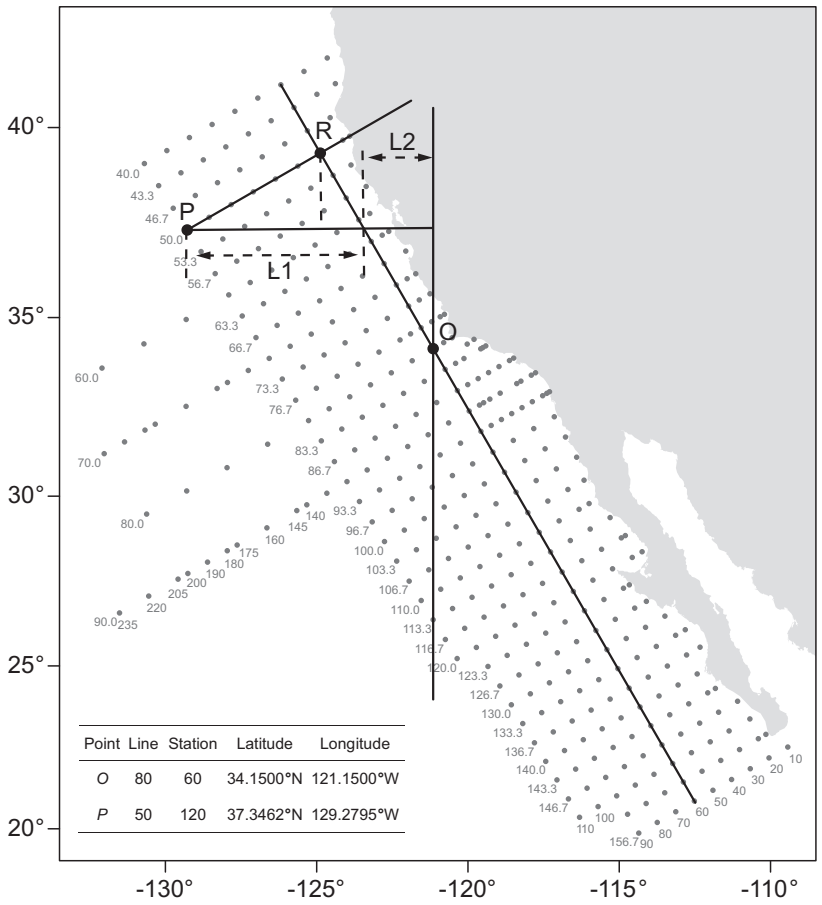

Figure 1. CalCOFI station plan and schematic representation of the geometrical components used for CalCOFI coordinate conversions (after Eber and Hewitt 1979). Point *O* is located at line 80, station 60, 34.15°N, 121.15°W, and is the point about which the CalCOFI sampling grid is rotated –30° off of the meridian. Point *P* is located at line 50, station 120, 129.2795°W, 37.3462°N, and is used as an example for calculations. Reference point *R*, and line segments *L1*, and *L2* are intermediate products used in trigonometric calculations (equations 4–8 and 12–14).

where INVERSE is calculated using three iterations of equation 9.

$$(16)\quad PLN = 80 - (RLA - 34.15) * \frac{5}{\cos\left(\frac{\Pi}{180} * 30\right)}$$

$$(17)\quad PSN = 60 + (RLA - PLA) * \frac{15}{\sin\left(\frac{\Pi}{180} * 30\right)}$$

COMPUTER IMPLEMENTATIONS

We provide the source code for CalCOFI conversions in Java, Perl, Python, and R as appendices I–IV. The code is also available for ftp download from ftp://swfscftp.noaa.gov/users/eweber/calcoficonversions/. These programs are adjusted to accept longitudes *east* of the prime meridian as either positive or negative numbers, e.g., –121° or 239°. Thus, positive numbers less than 180° should not be entered because they would be interpreted as eastern longitudes. For each version, we provide sample conversions from geographic coordinates to CalCOFI coordinates and the inverse conversion at Point *O* (fig. 1), and from CalCOFI coordinates to geographic coordinates and the inverse conversion at point *P*. The conversion algorithm results in small errors, generally in the hundredths of seconds. This is not normally a problem because CalCOFI coordinates are rounded to the nearest tenth of a line and whole station number by convention. However, programmers should be aware that calculating a series of repeated forward and inverse conversions on a point without rounding intermediate results would result in a larger accumulated error. The Python and R code are designed to work with either single-coordinate pairs or arrays of multiple coordinate points. The Java class creates objects that represent a single coordinate, although it could easily be modified to work with arrays.

Java

Objects of the Java class CalcofiCoordConverter hold coordinates as either "longlat" or "calcofi." Coordinates

may be retrieved using the "getCoords" method. The current coordinate system may be retrieved using the "getProjection" method. The "reproject" method accepts either "longlat" or "calcofi" as an argument and converts to the appropriate coordinate system. Running the test class with, e.g.,

```
> java TestCalcofiCoordConverter -121.15 34.15 longlat
```

returns:

```
Original coordinates
x -121.15
y 34.15
projection longlat

New coordinates
x 79.99999886102664
y 60.0000059182792
projection calcofi

Backtransformed coordinates
x -121.15000054796204
y 34.15
projection longlat
```

Testing conversion in reverse order at point *P* returns:

```
> java TestCalcofiCoordConverter 50 120 calcofi

Original coordinates
x 50.0
y 120.0
projection calcofi

New coordinates
x -129.2795443042271
y 37.34615242270663
projection longlat

Backtransformed coordinates
x 49.999998777973516
y 120.00000634983587
projection calcofi
```

Perl

The Perl code listed in Appendix II is designed to read input data from a file (coords.in) and write the converted output coordinates (along with the original input coordinate data) to a file (coords.out). The section of the code that begins with "USER-MODIFIED SETTINGS" can be changed to use different filenames and identify the direction of the conversion ("cc2geo" for CalCOFI station grid coordinates to geographic coordinates and "geo2cc" for geographic coordinates to CalCOFI station grid coordinates).

Sample input and output files in comma-separated values format are listed below for conversions in both directions.

Contents for cc2geo_coords.in:

```
LINE,STATION
80.0,60.0
50.0,120.0
```

Contents for cc2geo_coords.out:

```
LINE,STATION,LAT_DD,LON_DD
80.0,60.0,34.1500,-121.1500
50.0,120.0,37.3462,-129.2795
```

Contents for geo2cc_coords.in:

```
LATD,LATM,LATS,LATH,LOND,LONM,LONS,LONH
34,9,0,N,121,9,0,W
34.15,0,0,N,121.15,0,0,W
37,20.7692,0,N,129,16.7727,0,W
```

Contents for geo2cc_coords.out:

```
LATD,LATM,LATS,LATH,LOND,LONM,LONS,LONH,LAT_DD,
LON_DD,LINE,STATION
34,9,0,N,121,9,0,W,34.15,-121.15,80.0000,60.0000
34.15,0,0,N,121.15,0,0,W,34.15,-121.15,80.0000,60.0000
37,20.7692,0,N,129,16.7727,0,W,37.3461533333333,
-129.279545,50.0000,120.0000
```

Python

The Python functions listed in Appendix III perform the conversions on single coordinate pairs as *x* and *y* arguments, or multiple points by entering tuples, lists, or arrays from the numeric python module numpy. The functions always return numpy arrays.

An example of use at point *O*:

```
> line, station = latlontostation(-121.15, 34.15)
> line
79.999998861026384
> station
60.000005918280479
> stationtolatlon(line, station)
array([-121.15000055,   34.15      ])
```

and at point *P*:

```
> lon, lat = stationtolatlon(50, 120)
> lon
-129.27954430422696
> lat
37.346152422706631
>latlontostation(lon, lat)
array([  49.99999878,  120.00000635])
```

Multiple points may be converted equivalently using tuples:

```
> stationtolatlon((80, 50), (60, 120))
```

lists:

```
> stationtolatlon([80, 50], [60, 120])
```

or numpy arrays:

```
> arr = numpy.array((80, 60, 50, 120))
> arr.shape = (2, 2)
> arr
array([[ 80,  60],
       [ 50, 120]])
> stationtolatlon(arr)
```

All three of these examples return:

```
array([[-121.15      ,   34.15      ],
       [-129.2795443 ,   37.34615242]])
```

R

The R functions listed in Appendix IV work similarly:

```
> lineandstation <- latlon.to.station(c(-121.15,
34.15))
> lineandstation
     line  station
[1,]   80 60.00001
> station.to.latlon(lineandstation)
         lon   lat
line -121.15 34.15
```

And:

```
> latlon <- station.to.latlon(c(50, 120))
> latlon
           lon      lat
[1,] -129.2795 37.34615
> latlon.to.station(latlon)
    line station
lon   50     120
```

The R functions will accept matrices of two columns to perform conversions on multiple points at the same time. For example,

```
> mat <- matrix(c(80, 50, 60, 120), 2, 2)
> mat
     [,1] [,2]
[1,]   80   60
[2,]   50  120

> station.to.latlon(mat)
       lon            lat
[1,] -121.1500 34.15000
[2,] -129.2795 37.34615
```

By default, R uses fewer significant digits than Python or Java but this could be adjusted in the R options.

OTHER CODE AND TOOLS TO PERFORM CALCOFI COORDINATE CONVERSIONS

A few other tools and sources of code are freely available to perform CalCOFI coordinate conversions. An online converter tool written in JavaScript by R. Thombley is available from the University of California San Diego, Scripps Institution of Oceanography at http://calcofi.org/field-program/station-information/381-linestaalgorithm.html (accessed 6/5/2013). The page also provides JavaScript code to run the tool locally, and similar Fortran, Matlab, and Visual Basic code to perform CalCOFI coordinate conversions. We note that CalCOFI researchers recognized the typographical errors contained in Eber and Hewitt 1979 shortly after its publication, and corrections were applied to all of the programs mentioned above. However, we are not aware of any other published record of the algorithms with complete corrections.

Another tool that will soon be available for converting to and from the CalCOFI coordinate system is the PROJ.4 cartographic projection library, available at http://trac.osgeo.org/proj/wiki. PROJ.4 is a cross-platform library written in the C language that converts geographic coordinates to and from all commonly used cartographic projections (e.g., Mercator, Robinson, Stereographic, UTM, etc.). The GRASS GIS, PostGIS, MapServer, and many other programs call the PROJ.4 library to handle cartographic projections. It may also be called from the command line, and is used in the geographic libraries of other programming languages such as the pyproj module in Python, the Geo-Proj4 library in Perl, and the rgdal and proj4 packages in R. The senior author has submitted a patch to perform forward and inverse conversions to the CalCOFI coordinate system, which will likely be included in the next release of PROJ.4 (version 4.9). Thus, CalCOFI conversions will be available natively in a number of languages using their existing geographic tools. The PROJ.4 string used to convert to CalCOFI line and station is "**+proj=calcofi +ellps=clrk66**." For the impatient, the patch can be obtained from http://trac.osgeo.org/proj/attachment/ticket/135/calcofi.patch and compiled against the development trunk.

ACKNOWLEDGMENTS

We thank R. Hewitt and three anonymous reviewers for reviewing the manuscript.

LITERATURE CITED

California Academy of Sciences, California Division of Fish and Game, Scripps Institution of Oceanography, and U.S. Fish and Wildlife Service. 1950. California Cooperative Sardine Research Program Progress Report 1950. State of Cal. Dept Nat. Res. This is also Cal. Coop. Fish. Invest. Rep. 1:11–22 http://calcofi.org/publications/ccreports/379-vol01-1950.html . Accessed 6 November 2012.

Eber, L. E. and R. P. Hewitt. 1979. Conversion algorithm for the CalCOFI station grid. Cal. Coop. Fish. Invest. Rep. 20:135–137.

Snyder, J. P. 1987. Map projections—a working manual. U.S. Geol. Surv. Professional Paper 1395. U.S. Dept of Int., 383pp. http://pubs.er.usgs.gov/publication/pp1395. Accessed 6 November 2012.

Thombley, R. 2006. Conversion algorithm for the CalCOFI station grid: errata. http://cce.lternet.edu/data/sampling-grid/errata_conversionV3.pdf . Accessed 6 November 2012.

APPENDIX I. JAVA CLASS FOR CONVERTING CALCOFI COORDINATES AND A TEST CLASS

```
class CalcofiCoordConverter {
   private double[] crds;
   private String proj;

   public CalcofiCoordConverter (double x,
         double y, String projection){
      if (!projection.equals("longlat") &&
         !projection.equals("calcofi")) {
         throw new IllegalArgumentException(
            "projection must be either " +
            "'longlat' or 'calcofi'");
      }
      proj = projection;

      if (proj.equals("longlat") &&
         x > 180.0) x = (x - 360.0) * -1.0;

      crds = new double[2];
      crds[0] = x;
      crds[1] = y;
   }

   public double[] getCoords() {
      return crds;
   }

   public String getProjection() {
      return proj;
   }

   private double toMercator(double lat) {
      double y =
         Math.toDegrees(Math.log(
            Math.tan(Math.toRadians(
            45.0 + lat / 2))) -
            0.00676866 * Math.sin(
            Math.toRadians(lat)));
      return y;
   }

   private double inverseMercator(
         double mercLat) {
      double approxLat = mercLat;
      for (int i = 1; i < 4; i++) {
         approxLat = 2 * (Math.atan(
            Math.exp(Math.toRadians(mercLat) +
            0.00676866 * Math.sin(
            Math.toRadians(approxLat)))) *
            180 / Math.PI - 45);
      }
      return approxLat;
   }

   public void reproject(String projection) {
      if (projection.equals("calcofi") &&
            !proj.equals("calcofi")) {
         double lon = crds[0];
         double lat = crds[1];
         if (lon > 180.0) lon = (
            lon - 360.0) * -1.0;
         // assume pos in the western hemisphere

         if (lon < 0) lon = lon * -1;
         double L1 = (toMercator(lat) -
            toMercator(34.15)) *
            Math.tan(Math.toRadians(30));
         double L2 = lon - L1 - 121.15;
         double mercRefLatitude = L2 *
            Math.cos(Math.toRadians(30)) *
            Math.sin(Math.toRadians(30)) +
            toMercator(lat);
         double refLatitude = inverseMercator(
            mercRefLatitude);
         // line
         crds[0] = 80 - (refLatitude - 34.15) *
            5 / Math.cos(Math.toRadians(30));
         // station
         crds[1] = 60 + (refLatitude - lat) *
            15 / Math.sin(Math.toRadians(30));

      } else if (projection.equals("longlat") &&
            !proj.equals("longlat")) {
         double line = crds[0];
         double station = crds[1];
         double refLatitude = 34.15 - 0.2 *
            (line - 80) * Math.cos(
            Math.toRadians(30));
         double lat = refLatitude -
            (station - 60) * Math.sin(
            Math.toRadians(30)) / 15;
         double L1 = (toMercator(lat) -
            toMercator(34.15)) *
            Math.tan(Math.toRadians(30));
         double L2 = (toMercator(refLatitude) -
            toMercator(lat)) / (Math.cos(
            Math.toRadians(30)) * Math.sin(
            Math.toRadians(30)));
         crds[0] = -1 * (L1 + L2 + 121.15);
         crds[1] = lat;
      }
      proj = projection;
   }
}

public class TestCalcofiCoordConverter {
   public static void main (String[] args) {
      double x = Double.parseDouble(args[0]);
      double y = Double.parseDouble(args[1]);
      String projection = args[2];
      CalcofiCoordConverter crdtest =
         new CalcofiCoordConverter(x, y,
            projection);

      String oldprojection;
      String newprojection;

      // initial projection
      oldprojection = crdtest.getProjection();
      System.out.println("Original coordinates");
      printProjectionInfo(crdtest);

      // do a conversion
      if (oldprojection.equals("calcofi")) {
         newprojection = "longlat";
      } else {
         newprojection = "calcofi";
      }
      crdtest.reproject(newprojection);
      System.out.println("New coordinates");
      printProjectionInfo(crdtest);

      // convert back to original coord. system
      crdtest.reproject(oldprojection);
      System.out.println(
         "Backtransformed coordinates");
      printProjectionInfo(crdtest);
   }

   private static void printProjectionInfo(
         CalcofiCoordConverter crdtest) {
```

```
      double[] coords;
      coords = crdtest.getCoords();
      System.out.println("x " + coords[0]);
      System.out.println("y " + coords[1]);
      System.out.println("projection " +
         crdtest.getProjection());
      System.out.println();
   }
}
```

APPENDIX II. PERL FUNCTIONS FOR CONVERTING CALCOFI COORDINATES

```
#CALCOFI_CONVERSION.PL
#
#This script converts geographic coordinates
#(latitude/longitude) to CalCOFI station grid
#coordinates (line and station numbers) or
#vice versa.
#A conversion flag identifies the direction of
#the conversion.
#
#! /usr/bin/perl -w
#
use strict;

######################## USER-MODIFIED SETTINGS
our $input_fname = "coords.in";
our $output_fname = "coords.out";

#this flag identifies the direction of the
#conversion-a value of "geo2cc" indicates from
#geographic coordinates (latitude/longitude) to
#CalCOFI station grid coordinates;
#-a value of "cc2geo" indicates from CalCOFI
#station grid coordinates to geographic
#coordinates (latitude/longitude).
our $conversion = "geo2cc";
#our $conversion = "cc2geo";
########################

open(INFILE, $input_fname) ||
die "Cannot open INPUT_FILE filehandle: $!\n";
open(OUTFILE, ">" . $output_fname) ||
die "Cannot open OUTPUT_FILE filehandle: $!\n";

if ($conversion eq "geo2cc") {
   print "Converting data in $input_fname from ",
   "geographic coordinates to CalCOFI station ",
   "grid coordinates...\n";
}
if ($conversion eq "cc2geo") {
   print "Converting data in $input_fname from ",
   "CalCOFI station grid coordinates to ",
   "geographic coordinates...\n";
}

print "\nReading data...\n";
our @inputrecs=<INFILE>;
chomp(@inputrecs);
close(INFILE);

#process lat/long data and calculate CalCOFI
#station-line grid coordinates; format other data
#and print all to file
our $m;
our $line;
our $station;
our $latd;
our $latm;
our $lats;
our $lath;
our $lond;
our $lonm;
our $lons;
our $lonh;
our $latdecdeg;
our $londecdeg;

#print header records in output file
if ($conversion eq "geo2cc") {
   print OUTFILE "LATD,LATM,LATS,LATH,LOND,",
   "LONM,LONS,LONH,LAT_DD,LON_DD,LINE,STATION\n";
}
elsif ($conversion eq "cc2geo") {
   print OUTFILE "LINE,STATION,LAT_DD,LON_DD\n";
}
else {
   print "Do not recognize the selected ",
   "conversion! Should be either geo2cc or ",
   "cc2geo. Please correct.\n";
   print "Press any key and program ",
   "will exit...\n";
   our $abort = <STDIN>;
   exit;
}

print "\nProcessing data and writing to output ",
"file...\n";

#skip header record in the input file
for ($m=1; $m<=$#inputrecs; $m++) {
   if ($conversion eq "geo2cc") {
      ($latd,$latm,$lats,$lath,$lond,$lonm,
      $lons,$lonh) = split(/,/, $inputrecs[$m]);

      #get CC line and station numbers
      ($line,$station,$latdecdeg,$londecdeg) =
      &ll2cc($latd,$latm,$lats,$lath,$lond,$lonm,
      $lons,$lonh);

      #round CC line and station numbers to
      #4 decimal places
      $line = sprintf( "%.4f", $line);
      $station = sprintf( "%.4f", $station);

      print OUTFILE "$latd,$latm,$lats,",
      "$lath,$lond,$lonm,$lons,$lonh,",
      "$latdecdeg,$londecdeg,$line,",
      "$station\n";
   }
   if ($conversion eq "cc2geo") {
      ($line,$station) = split(/,/, $inputrecs[$m]);

      #get latitude and longitude values
      ($line,$station,$latdecdeg,$londecdeg) =
      &cc2ll($line,$station);

      #round latitude and longitude to
      #4 decimal places
      $latdecdeg = sprintf( "%.4f", $latdecdeg);
      $londecdeg = sprintf( "%.4f", $londecdeg);

      print OUTFILE "$line,$station,",
      "$latdecdeg,$londecdeg\n";
   }
}

close(OUTFILE);

print "\nConverted data written to $output_fname ...",
"program finished!\n";
```

```
sub ll2cc {
   #Subroutine to implement algorithm that converts
   #lat/long coordinates to CalCOFI grid
   #coordinates (line and station).
   #
   #INPUT: Latitude and longitude values in DMS to
   #be converted to CC grid
   #OUTPUT: A line and station value for CC grid
   #(and signed decimal degrees of latitude and
   #longitude).
   #
   #
   use Math::Trig;

   my $latitude_deg = shift;
   my $latitude_min = shift;
   my $latitude_sec = shift;
   my $latitude_hem = shift;
   my $longitude_deg = shift;
   my $longitude_min = shift;
   my $longitude_sec = shift;
   my $longitude_hem = shift;

   my $latdd = $latitude_deg + ($latitude_min/60.0)
   + ($latitude_sec/3600.0);
   my $londd = $longitude_deg +
   ($longitude_min/60.0) + ($longitude_sec/3600.0);

   if ($latitude_hem eq "S" ||
   $latitude_hem eq "s") {
      $latdd = $latdd*(-1.0);
   }

   if ($longitude_hem eq "W" ||
   $longitude_hem eq "w") {
      $londd = $londd*(-1.0);
   }

   #Note: The following is the reverse of typical
   #conventions for E/W longitude and sign.
   #The CalCOFI grid conversion equations assume
   #longitude is positive value for western
   #hemisphere.
   #
   my $pla = $latdd;
   my $plo = $londd;
   $plo = $plo*(-1.0);

   my $L1 = ( &mctr($pla) - &mctr(34.15) ) *
   tan(deg2rad(30));
   my $L2 = $plo - $L1 - 121.15;
   my $mctr_rla_val = ( $L2 * cos(deg2rad(30)) *
   sin(deg2rad(30)) ) + &mctr($pla);
   my $rla = &invmctr($mctr_rla_val);
   my $pln = 80.0 -
   ((($rla - 34.15) * 5.0)/cos(deg2rad(30)));
   my $psn = 60.0 +
   ((($rla - $pla) * 15.0)/sin(deg2rad(30)));

   return ($pln, $psn, $latdd, $londd);
}

sub cc2ll {
   #Subroutine to implement algorithm that converts
   #CalCOFI grid coordinates (line and station) to
   #lat/long coordinates.
   #
   #INPUT: Line and station value(s) for CC grid to
   #be converted to geographic coordinates
   #OUTPUT: Latitude and longitude values in
   #decimal degrees (signed).
   #
   #
   use Math::Trig;

   my $pln = shift;
   my $psn = shift;

   my $rla = 34.15 -
   (0.2 * ($pln - 80.0) * cos(deg2rad(30)));
   my $pla = $rla - ((1.0/15.0) * ($psn - 60.0) *
   sin(deg2rad(30)));
   my $L1 = ( &mctr($pla) - &mctr(34.15) ) *
   tan(deg2rad(30));
   my $L2 = ( &mctr($rla) - &mctr($pla) )
      / (cos(deg2rad(30)) * sin(deg2rad(30)));
   my $plo = $L1 + $L2 + 121.15;

   #Note: The CalCOFI grid conversion equations
   #assume longitude is positively signed in the
   #western hemisphere.
   #The following changes the sign on the longitude
   #value to be in the convention of negatively
   #signed in the western hemisphere.
   #
   $plo = $plo * (-1.0);

   return ($pln, $psn, $pla, $plo);
}

sub mctr {
   #Subroutine to implement mercator transform
   #function.
   #
   #INPUT: A latitude value in decimal degrees.
   #OUPUT: A value in "mercator meridional units"
   #
   #
   use Math::Trig;
   my $deg = shift;
   my $mctr_val = (180.0/pi) *
   ( log(tan(deg2rad(45.0 + ($deg/2.0)))) -
   (0.00676866 * sin(deg2rad($deg))) );

   return $mctr_val;
}

sub invmctr {
   #Subroutine to implement inverse mercator
   #transform function.
   #Because this function does not have a precise
   #algebraic form, an iterative process is
   #employed.
   #INPUT: A value in "mercator meridional units"
   #OUTPUT: A value in decimal degrees
   #
   #
   my $i;
   my $mctr_rla_val = shift;
   my $rla = $mctr_rla_val;
   my $rla_init = $rla;
   use Math::Trig;
   for ($i=0; $i < 5; $i++) {
      $rla = 2 * ( (180/pi) *
      atan( exp(deg2rad($rla_init) +
      (0.00676866*sin(deg2rad($rla)))) ) - 45);
   }

   return $rla;
}
```

APPENDIX III. PYTHON FUNCTIONS FOR CONVERTING CALCOFI COORDINATES

```
import numpy as np

def invmercator(mercatorLat, iterations=3):
     mercatorLat = np.array(mercatorLat,
                   dtype='float')
     approxLat = mercatorLat
     for i in range(iterations):
        approxLat = 2 * (np.arctan(np.exp(
           np.deg2rad(mercatorLat) +
           0.00676866 * np.sin(np.deg2rad(
              approxLat)))) * 180 / np.pi
           - 45)
     return(approxLat)

def tomercator(latitude):
   latitude = np.array(latitude)
   y = np.rad2deg(
      np.log(np.tan(np.deg2rad(
         45 + latitude / 2))) -
      0.00676866 * np.sin(np.deg2rad(latitude)))
   return(y)

def stationtolatlon(x, y=None):
   """
   x is line, y is station, or x is a matrix
   x and y are numbers, lists, tuples,
   or numpy arrays,
   """
   if y == None:
      line = x[:, 0]
      station = x[:, 1]
   else:
      line = x
      station = y

   line = np.array(line, dtype='float')
   station = np.array(station, dtype='float')

   # need reshape b/c single numbers could
   # be wrapped in arrays
   if len(line.shape) == 0:
      line = line.reshape(1)

   if len(station.shape) == 0:
      station = station.reshape(1)

   refLatitude = (34.15 - 0.2 * (line - 80) *
              np.cos(np.deg2rad(30)))
   latitude = (refLatitude - (station - 60) *
           np.sin(np.deg2rad(30)) / 15)
   L1 = ((tomercator(latitude) -
         tomercator(34.15)) *
         np.tan(np.deg2rad(30)))
   L2 = (((tomercator(refLatitude) -
         tomercator(latitude)) /
         (np.cos(np.deg2rad(30)) *
         np.sin(np.deg2rad(30)))))
   longitude = -1 * (L1 + L2 + 121.15)
   ans = np.vstack((longitude, latitude)).T
   if len(line) == 1:
      ans = ans[0]
   return(ans)


def latlontostation(x, y=None):
   """
   x and y are numbers, lists, tuples,
   or numpy arrays, or
   x can be a matrix with y = None
   """
   if y == None:
      lon = x[:, 0]
      lat = x[:, 1]
   else:
      lon = x
      lat = y
   lon = np.array(lon, dtype='float')
   lat = np.array(lat, dtype='float')
   # need reshape b/c single numbers
   # could be wrapped in arrays
   if len(lon.shape) == 0:
      lon = lon.reshape(1)
   if len(lat.shape) == 0:
      lat = lat.reshape(1)
   # assume we're in the western hemispere
   lon[lon > 180] = -1 * (lon[lon > 180] - 360)
   lon[lon < 0] = lon[lon < 0] * -1
   L1 = ((tomercator(lat) - tomercator(34.15)) *
         np.tan(np.deg2rad(30)))
   L2 = lon - L1 - 121.15
   mercRefLatitude = (L2 * np.cos(np.deg2rad(30)) *
                      np.sin(np.deg2rad(30)) +
                      tomercator(lat))
   refLatitude = invmercator(mercRefLatitude)
   line = (80 - (refLatitude - 34.15) * 5 /
           np.cos(np.deg2rad(30)))
   station = (60 + (refLatitude - lat) * 15 /
              np.sin(np.deg2rad(30)))
   ans = np.vstack((line, station)).T
   if len(line) == 1:
      ans = ans[0]
   return(ans)
```

APPENDIX IV. R FUNCTIONS FOR CONVERTING CALCOFI COORDINATES

```
`.deg2rad` <- function(deg) deg * pi / 180

`.rad2deg` <- function(rad) rad * 180 / pi

`.inverse.mercator` <- function(mercatorlat,
                       iterations = 3)
{
   approxlat <- mercatorlat
   iterlatitude <- function(mercatorlat,
                       approxlat)
   {
      approxlat <- 2 * (atan(exp(.deg2rad(
         mercatorlat) + 0.00676866 *
         sin(.deg2rad(approxlat)))) *
         180 / pi - 45)
      approxlat
   }
   for (i in 1:iterations) approxlat <-
      iterlatitude(mercatorlat, approxlat)
   approxlat
}

`.to.mercator` <- function(latitude)
{
   y <- .rad2deg(log(tan(.deg2rad(45 +
              latitude / 2))) - 0.00676866 *
              sin(.deg2rad(latitude)))
   y
}

`station.to.latlon` <- function(x,
                       roundlines = true)
{
```

```
   if (length(x) == 2 & class(x) != 'matrix'){
      x <- matrix(x, 1, 2)
   }
   line <- x[, 1]
   station <- x[, 2]

   reflatitude <- 34.15 - 0.2 * (line - 80) *
      cos(.deg2rad(30))
   latitude <- reflatitude - (station - 60) *
      sin(.deg2rad(30)) / 15
   l1 <- (.to.mercator(latitude) - .to.mercator(
      34.15)) * tan(.deg2rad(30))
   l2 <- (.to.mercator(reflatitude) -
        .to.mercator(latitude)) /
        (cos(.deg2rad(30)) * sin(.deg2rad(30)))
   longitude <- -1 * (l1 + l2 + 121.15)
   cbind(lon = longitude, lat = latitude)
}

`latlon.to.station` <- function(x)
{
   if (length(x) == 2 & class(x) != 'matrix'){
      x <- matrix(x, 1, 2)
   }
   longitude <- x[, 1]
   latitude <- x[, 2]

   # assume we're in the western hemispere
   longitude[longitude > 180] <- -1 * (
      longitude[longitude > 180] - 360)
   longitude[longitude < 0] <- longitude[
      longitude < 0] * -1

   l1 <- (.to.mercator(latitude) - .to.mercator(
      34.15)) * tan(.deg2rad(30))
   l2 <- longitude - l1 - 121.15

   mercreflatitude <- l2 * cos(.deg2rad(30)) *
      sin(.deg2rad(30)) + .to.mercator(latitude)
   reflatitude <- .inverse.mercator(
      mercreflatitude)
   line <- 80 - (reflatitude - 34.15) * 5 /
      cos(.deg2rad(30))
   station <- 60 + (reflatitude - latitude) *
      15 / sin(.deg2rad(30))

   cbind(line = line, station = station)
}
```