The Stata Journal

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*), *Scopus*, and *Social Sciences Citation Index*.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

Subscriptions are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

http://www.stata.com/bookstore/sj.html

Subscription rates listed below include both a printed and an electronic copy unless otherwise mentioned.

U.S. and Canada		Elsewhere	
Printed & electronic		**Printed & electronic**	
1-year subscription	$115	1-year subscription	$145
2-year subscription	$210	2-year subscription	$270
3-year subscription	$285	3-year subscription	$375
1-year student subscription	$ 85	1-year student subscription	$115
1-year institutional subscription	$345	1-year institutional subscription	$375
2-year institutional subscription	$625	2-year institutional subscription	$685
3-year institutional subscription	$875	3-year institutional subscription	$965
Electronic only		**Electronic only**	
1-year subscription	$ 85	1-year subscription	$ 85
2-year subscription	$155	2-year subscription	$155
3-year subscription	$215	3-year subscription	$215
1-year student subscription	$ 55	1-year student subscription	$ 55

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.

The Stata Journal (2015)
15, Number 2, pp. 523–536

gpsbound: A command for importing and verifying geographical information from a user-provided shapefile

Tim S. L. Brophy
University of Cape Town
Southern Africa Labour and Development Research Unit (SALDRU)
Cape Town, South Africa
tslbrophy@gmail.com

Reza Che Daniels
University of Cape Town
School of Economics
Cape Town, South Africa
reza.daniels@uct.ac.za

Sibongile Musundwa
University of Cape Town
Southern Africa Labour and Development Research Unit (SALDRU)
Cape Town, South Africa
sibongile.musundwa@gmail.com

Abstract. Geographical coordinates such as Global Positioning System (GPS) latitude and longitude estimates form the foundation of many spatial statistical methods. `gpsbound` allows users to 1) import geographical information from the attribute table of a polygon shapefile based on the identified location of GPS coordinates in a Stata dataset, and 2) check whether the GPS coordinates lie within the bounds of a polygon demarcated in the shapefile (for example, enumeration areas or primary sampling units). `gpsbound` also allows users to work with spatial data in Stata without the use of Geographical Information System software.

Keywords: dm0080, gpsbound, spatial data, point-in-polygon, GPS coordinates, shapefile, attribute table

1 Introduction

Spatial data identify geographical features. To accomplish this, three data types are used: polygons, points, and lines. Polygons describe a geographical area, such as the geopolitical boundaries of a country; points describe features represented by a set of Global Positioning System (GPS) coordinates, such as the location of a dwelling unit; and lines describe features such as the path of a road. Spatial data are usually stored in the vector data format called a shapefile. A shapefile is a relational database that consists of multiple tables connected by a primary key (identifier). Each component table has its own distinct data structure and file format. Traditionally, shapefiles are manipulated using Geographical Information System (GIS) software. `gpsbound` allows users to work with GPS coordinates and related information in Stata by manipulating selective components of the shapefile relational database.

Frequently, Stata datasets contain GPS coordinates for objects of interest (for example, dwelling units or establishments) but do not have information about the geographical areas (polygons) into which those GPS coordinates fall. `gpsbound` allows information from the attribute table of a user-provided polygon shapefile to be imported as new variables into the open (master) dataset for each latitude and longitude pair. It also allows users to check whether the GPS coordinates lie within a polygon of interest (for example, specific enumeration areas [EAs] or primary sampling units). If the GPS coordinates lie outside the polygon of interest, the algorithm treats those coordinates as errors of measurement and fails to import any data from the attribute table for the affected points. This allows users to identify incorrect coordinates immediately, which is useful for further diagnostic work (and possibly data collection).

To import spatial data to Stata before the introduction of `gpsbound`, users had to first export their GPS coordinates of interest from Stata into a text file, and then import the text file into GIS software, converting these data into a point shapefile representing the GPS coordinates of interest. Users then had to open a polygon shapefile in the GIS software and perform a spatial join between the point and the polygon shapefiles to link the points of interest to the attribute data of the polygon shapefile. These attribute table results then had to be exported by the GIS software into a text file and imported by Stata for analysis.

With `gpsbound`, this process is no longer required. Users can simply specify the variables that contain decimal-degree GPS coordinates and the file location of a polygon shapefile (in decimal-degree units) to which the coordinates must be mapped. The algorithm will then perform the spatial join and import the relevant attribute data. Consequently, users can now import geographical data for analysis without having to learn GIS software.

It was previously possible to import entire shapefiles and attribute tables into Stata using Crow's (2006) `shp2data` command, but `gpsbound` allows the selective importation of individually identified polygons and attribute data based on the GPS coordinates from a Stata dataset. This functionality benefits two types of Stata users: 1) researchers interested in importing the attribute data of polygons for given GPS coordinates and 2) survey methodologists interested in verifying the location of sampled units (for example, households or establishments) within selected EAs or primary sampling units, which can be done (if necessary) in field in real time.

2 Practical experience with using gpsbound in a nationally representative longitudinal household survey

We developed `gpsbound` in response to a practical problem we faced in the National Income Dynamics Study (NIDS), a nationally representative longitudinal household survey in South Africa. The first three waves of the survey were implemented by the Southern Africa Labour and Development Research Unit (SALDRU) at the University of Cape Town. The GPS coordinates of the dwelling units of respondents were recorded at each wave. The problems we encountered were that 1) we needed a way to verify that the selected dwelling units were the correct ones for the initial sample drawn in the first wave of the survey and 2) in subsequent waves of the survey, we needed a way to validate that fieldworkers were either a) revisiting the correct dwelling units or b) correctly recording the location of new dwelling units when continuing sample members of the survey moved to new locations in the country and, therefore, new dwelling units.

Fieldwork for wave 3 of the survey ended in 2012. In all, 9,273 dwelling units with observed GPS coordinates were located across South Africa. We used `gpsbound` to 1) verify that the GPS coordinates were correctly recorded in field and 2) import the geographical information for new dwelling units where no geographical information had previously been stored (because the information represented the dwelling units of individuals that moved between waves of the survey).

Running `gpsbound` for 9,273 dwelling units on a standard personal computer with latitude and longitude coordinates resulted in the following computational times for different polygon shapefiles:

1. For the EA shapefile from the South African National Census of 2011, which consisted of 103,576 EA polygons, the algorithm took 302 seconds.

2. For the subplace shapefile (equivalent in geographical area terms to a suburb or small town), which consisted of 22,108 subplace polygons, the algorithm took 186 seconds.

3. For the main-place shapefile (equivalent in geographical area terms to a big town or city), which consisted of 14,039 main-place polygons, the algorithm took 196 seconds.

4. For the district council shapefile (equivalent in geographical area terms to a county), which consisted of 52 polygons, the algorithm took 76 seconds.

5. For the province shapefile, which consisted of 9 provinces, the algorithm took 310 seconds.

This nonlinear trend in computational performance was due to the size of the shapefiles themselves, which differed according to the density of points per polygon.

3 Polygon spatial data and file structure

A polygon shapefile consists of geographically demarcated borders with an attribute table. For example, a shapefile may consist of provincial, city, or town borders and store the names of each regional unit in an attribute table. Therefore, for a shapefile to be classified as such, there must be at least three mandatory file formats (Environmental Systems Research Institute 1998):

1. the shapefile (`.shp`), which contains the spatial data describing the location of the features (for example, GPS coordinates);
2. the dBASE file (`.dbf`), which contains a table of nonspatial attributes of the features (for example, names of cities); and
3. the index file (`.shx`), which allows GIS software to effectively navigate the `.shp` file.

Note that the `.shp` file extension is not the whole shapefile but a single component of the relational database that constitutes a shapefile. Consequently, it cannot be used independently of the other component files. To understand how `gpsbound` functions, one must fully understand the different formats and types of data that are contained in the component files. Because `gpsbound` uses polygon shapefiles, we will limit our discussion to these.

3.1 The shapefile (.shp)

A polygon shapefile contains an index variable that is used as the primary key (identifier), as well as the x and y coordinates for a series of GPS coordinates. It also contains header information for each polygon. These header data contain the location of each polygon within the shapefile and the number of GPS points that make up a polygon. Finally, it contains the bounding box data, which are the maximum and minimum x and y coordinates.

3.2 The dBASE file (.dbf)

The dBASE file is a tabular file format containing the attribute data that describe the features of the shapefile and the index variable that will act as the primary key (identifier) to link the `.dbf` and `.shp` files in a one-to-many relationship.

3.3 The index file (.shx)

The index file is, in nontechnical terms, an indexing file that tells GIS software how to efficiently read the `.shp` file. It does this by storing the position of the starting byte of each shapefile element, thus allowing the GIS software to quickly navigate to specific elements within a shapefile without having to process the whole shapefile.

4 The gpsbound command

4.1 Description

`gpsbound` maps decimal-degree GPS coordinate points to a decimal-degree polygon shapefile. For each set of latitude and longitude coordinates, it returns the attributes of the corresponding polygon. It also returns an optional variable indicating whether the GPS coordinates were mapped successfully.

`gpsbound` works only with decimal-degree latitude and longitude coordinates and polygon shapefiles. If the GPS coordinates are in any other format (for example, degrees-minutes-seconds), they must be converted to decimal degrees before `gpsbound` can be used.

4.2 Syntax

`gpsbound using` *shapefilename* [*if*] `,` `latitude(`*varname*`)` `longitude(`*varname*`)`
  [`valid(`*newvar*`)` `prefix(`*string*`)` `keepusing(`*varlist*`)`]

`gpsbound` requires two inputs: the polygon *shapefilename* and the variable names that contain the latitude and longitude coordinates. The full path and name of the shapefile must be included, including the file suffix (`.shp`). An associated dBASE (`.dbf`) file must also be stored in the same folder path as the shapefile and have the same name as the shapefile, though with the `.dbf` file suffix.

4.3 Options

`latitude(`*varname*`)` and `longitude(`*varname*`)` specify the variable names containing the latitude and longitude coordinates. These options are required.

`valid(`*newvar*`)` indicates whether the latitude and longitude coordinates that identify a point (for example, dwelling unit) fall within the bounds of the correct polygon (for example, EA).

`prefix(`*string*`)` prefixes *string* to the imported variable names that are derived from the attribute table of the shapefile.

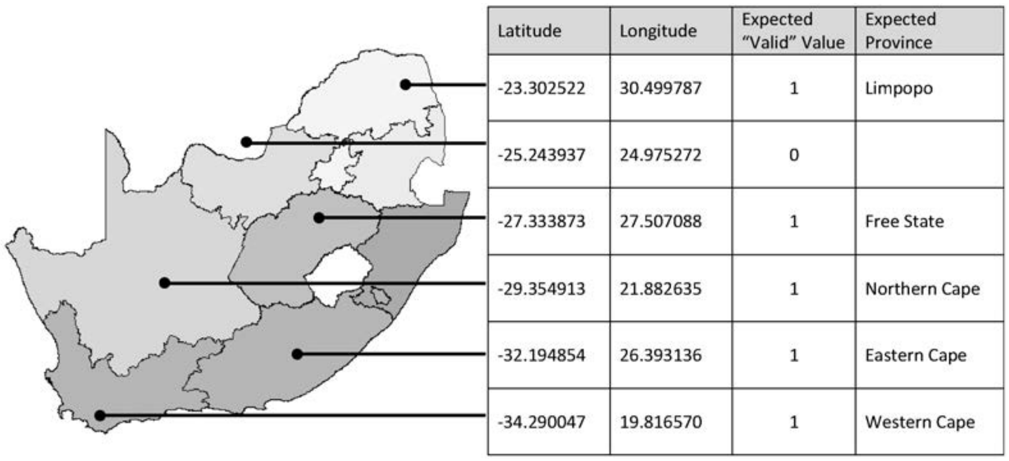
`keepusing(`*varlist*`)` specifies a list of variable names to be identified from the attribute table of the shapefile. These variables can be viewed by opening the `.dbf` file in a spreadsheet program. Only the variables listed in `keepusing()` will then be imported from the shapefile. The default is to import all variables from the shapefile attribute table.

4.4 Output

`gpsbound` returns output in two forms: 1) as the imported variable from the shapefile attribute table, which is added to the open dataset, and 2) as an optional binary variable that indicates the success of the mapping routine between the GPS point and the polygon shapefile. This latter output is created only when the option `valid()` is included.

▷ Example

Figure 1 is a map (and associated polygon shapefile) of the nine provinces of the Republic of South Africa, the Kingdom of Lesotho, and the Kingdom of Swaziland (the Kingdoms are the nonshaded polygons within the contiguous geopolitical boundary of South Africa). The map also identifies six sets of decimal-degree GPS coordinates for randomly selected points in and around South Africa. Five of the randomly selected points fall within the bounds of South Africa. The remaining point falls just outside the borders of South Africa, inside the neighboring country Botswana.



Latitude	Longitude	Expected "Valid" Value	Expected Province
-23.302522	30.499787	1	Limpopo
-25.243937	24.975272	0	
-27.333873	27.507088	1	Free State
-29.354913	21.882635	1	Northern Cape
-32.194854	26.393136	1	Eastern Cape
-34.290047	19.816570	1	Western Cape

Figure 1. Example of `gpsbound`

Two columns in the table indicate the results we expect to see once we run `gpsbound`. The third column in the table represents the expected outcome of the optional output `valid()`, which is a binary variable indicating whether the given GPS coordinates of interest fall within any of the provincial polygons that make up South Africa's nine provinces. The fourth column indicates the expected value of the "province" attribute that will be returned for each point by running `gpsbound` given the set of coordinates and the South African province shapefile.

```
. * create the example dataset
. set obs 6
obs was 0, now 6
. generate Latitude = .
(6 missing values generated)
. replace Latitude = -23.302522 in 1
(1 real change made)
. replace Latitude = -25.243937 in 2
(1 real change made)
. replace Latitude = -27.333873 in 3
(1 real change made)
. replace Latitude = -29.354913 in 4
(1 real change made)
. replace Latitude = -32.194854 in 5
(1 real change made)
. replace Latitude = -34.290047 in 6
(1 real change made)
. generate Longitude = .
(6 missing values generated)
. replace Longitude = 30.499787 in 1
(1 real change made)
. replace Longitude = 24.975272 in 2
(1 real change made)
. replace Longitude = 27.507088 in 3
(1 real change made)
. replace Longitude = 21.882635 in 4
(1 real change made)
. replace Longitude = 26.393136 in 5
(1 real change made)
. replace Longitude = 19.81657 in 6
(1 real change made)
. * Run gpsbound on the South African province shapefile available
. * at http://www.demarcation.org.za/[1]
. gpsbound using Province_New_SANeighbours.shp, latitude(Latitude)
> longitude(Longitude) valid(valid) keepusing(PROVINCE)
```

1. No direct URL is available to the download page. On the home page, click on the *Downloads* menu. Select *Boundary Data* and then *Boundary Data Main Files*. Finally, click on *Province* and then *Download* to download the `Province.zip` shapefile.

Execution of the above yields the following table of results:

```
. list

     ┌────────────────────────────────────────────────┐
     │  Latitude   Longitude   valid         PROVINCE │
     ├────────────────────────────────────────────────┤
  1. │ -23.30252    30.49979       1          Limpopo │
  2. │ -25.24394    24.97527       0                  │
  3. │ -27.33387    27.50709       1       Free State │
  4. │ -29.35491    21.88264       1    Northern Cape │
  5. │ -32.19485    26.39314       1     Eastern Cape │
     ├────────────────────────────────────────────────┤
  6. │ -34.29005    19.81657       1     Western Cape │
     └────────────────────────────────────────────────┘
```

The actual results and the expected results are the same, demonstrating that `gpsbound` functions as desired.

◁

5 Methods and formulas

`gpsbound` was written to be used on decimal-degree latitude and longitude coordinates with the World Geodetic Systems 1984 datum. A geodetic datum translates GPS coordinates into their relative position on earth. The need for geodetic datum arises because of the earth's ellipsoid shape. The World Geodetic Systems 1984 datum is used by modern GPS units and satellite navigation systems. It requires the latitude coordinate to fall between -90 and 90 degrees inclusive and requires the longitude coordinates to fall between -180 and 180 degrees inclusive (Spatial Reference 2007). The algorithm checks that the coordinates passed to `gpsbound` fall within this range. A similar check is performed on the polygon shapefile.

5.1 gpsbound subcommands

Broadly speaking, the `gpsbound` command can be broken into six distinct actions powered by six different subcommands. The first subcommand imports the dBASE (`.dbf`) data file, which contains the geographical attribute table for each polygon in the user-provided shapefile. The second subcommand imports the shapefile headers. The third subcommand identifies potential polygons into which each set of GPS coordinates may fall. The fourth subcommand imports each of the polygons that were identified as a potential polygon in subcommand three (Crow 2006). The fifth subcommand runs a point-in-polygon algorithm to associate each point with its corresponding polygon. This subcommand returns two key sets of results to Stata: a binary variable indicating whether the given GPS point falls within any of the potential polygons, and if so, a second data record containing the unique identifier for each polygon. The sixth and last subcommand merges the attribute table imported by the first subcommand into the dataset; this is done based on the polygon identifier returned from the point-in-polygon algorithm.

5.2 Importing the dBASE (.dbf) attribute table

As outlined, the first subcommand performed by `gpsbound` involves importing the dBASE file. The dBASE file is a fully fledged file format (dBASE 2014). The simplicity of dBASE's data structure means that it has been adopted by many other applications to assist in managing and storing data.

In the context of shapefiles, dBASE files are used to store the feature attributes for each of the polygons contained within a shapefile. The only requirement for the use of a dBASE file as a data format for shapefile attributes is that a unique identifier field is included. The values in the identifier field in the dBASE file must correspond to the values in the identifier field in the shapefile. Other than that, all the fields contained in the attribute table are defined by the author of the shapefile, which allows the author to attribute any characteristics deemed necessary to a particular shapefile. These attributes can be global attributes applicable to each shape in a shapefile or they can be unique for each shape in a shapefile. This is the real power of shapefiles and their attributes tables.

Thus one of the goals of `gpsbound` is to access this attribute information and link it to given GPS coordinates collected in the field or through other operations. The linking of the shapefile attributes allows researchers to access geographical information for analysis within Stata. Because there is no limit to the information attributable to polygons contained within a shapefile, the information that can be derived and analyzed from the use of shapefiles and GPS coordinates through `gpsbound` is only limited by the availability of GIS data.

5.3 Importing the shapefile headers

The second subcommand imports the headers of the polygons from the shapefile. A shapefile can be separated into two distinct categories of data: the headers and the coordinate data that describes each polygon's shape. The header data occur for each polygon contained in the shapefile. Thus if a user-provided shapefile contains 100 individual polygons, then the shapefile will contain 100 headers and $100 \times n$ rows of coordinates, where n is the number of coordinate points that make up a polygon's shape.

The headers contain basic information pertaining to each of the polygons, such as its primary key or identification number, the GPS coordinates of its bounding box, and the position of its starting and ending bits in the shapefile (Environmental Systems Research Institute 1998). Thus the second major action of `gpsbound` is to read through the shapefile and retrieve each of the polygon headers.[2] The headers are then stored in a matrix that we call the headers matrix. This matrix is held in Mata and is crucial to the efficiency of both the computational time taken when running `gpsbound` and the amount of memory needed to run `gpsbound`. Memory efficiency is of particular concern when

2. Previous versions of `gpsbound` did not have this step but rather read all the shapefile data into Mata; this was computationally inefficient and used an unnecessary amount of memory.

using shapefiles because they can contain hundreds of millions of coordinate points. If all points are held in memory at the same time, this affects both the overall computing speed and the available memory for other applications and for Stata itself.

5.4 Identifying possible polygons

It is at this point that we start to process each set of GPS coordinates passed by the user to `gpsbound` via the command's longitude and latitude parameters. Each GPS coordinate is compared with the bounding boxes found in the headers matrix. The bounding box of a polygon is defined by the minimum and maximum latitude and longitude of that polygon; in other words, it is a box that fits exactly around the polygon. If the user-provided latitude and longitude falls within the bounding box of a particular polygon, then that polygon is considered a possible polygon into which the coordinates might fall. This exercise will result in a list of possible polygon matches for each of the coordinates provided. This list of possible polygons is used to limit the number of iterations, so that each point only has to be tested against a few polygons and not against all the polygons in the polygon shapefile.

5.5 Importing possible polygons

Possible polygons that were identified for each GPS point by subcommand three are imported into Mata through the use of a shapefile polygon reader. The reader imports a specifically identified polygon.

5.6 The point-in-polygon algorithm

Each polygon is then subjected to the point-in-polygon algorithm until the GPS point is definitively placed inside one of the polygons. If, however, the GPS point cannot be definitively placed inside any of the polygons, a binary variable containing a 0 value is generated, indicating that the point was not matched to the shapefile at all.

The point-in-polygon algorithm uses a ray casting approach and the even–odd rule. The even–odd rule states that a ray drawn from the point of interest to infinity will intersect the edge of the polygon an odd number of times if that point falls within the polygon (World Wide Web Consortium [W3C] 2011). However, if the point of interest falls outside of the polygon, then the ray will intersect the border of the polygon an even number of times. This is best explained using illustrations.

In figure 2, the polygon is represented by A with X being the point of interest. Casting a horizontal ray to infinity, the ray intersects the edge of polygon A at intersection 1.

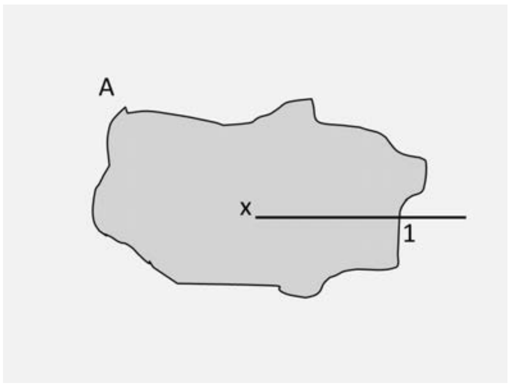


Figure 2. The point of interest lies within the polygon, as illustrated by the even–odd rule

As can be seen, the ray intersects the edge of the polygon only once, at intersection 1. Here the even–odd rule indicates that the point of interest X falls within polygon A because the ray intersects the edge of polygon A an odd number of times.

In figure 3, the point of interest X falls outside of polygon A. The illustration shows a ray cast from the point of interest X intersecting the edge of polygon A twice, at intersections 1 and 2.

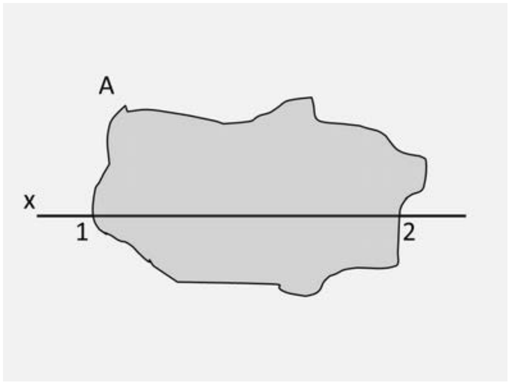


Figure 3. The point of interest lies outside the polygon, as illustrated by the even–odd rule

Applying the even–odd rule, we would determine that point X falls outside polygon A because the horizontal ray intersects the edges of polygon A an even number of times.

Figure 4 demonstrates a separate polygon B inside polygon A. Given point of interest X, if we draw a horizontal ray away from that point, the ray will intersect the edge of a polygon three times, at intersections 1, 2, and 3. If we apply the even–odd rule, we see that point X lies within polygon A because there is an odd number of intersections with the edges.

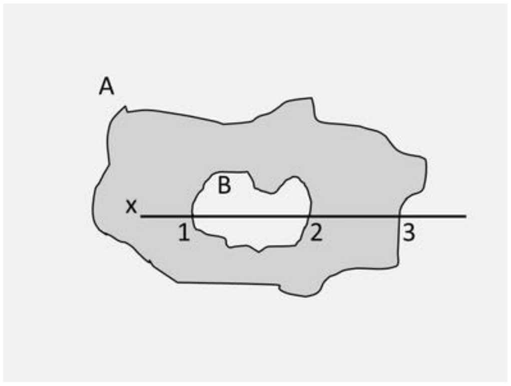


Figure 4. The point of interest lies within polygon A, as illustrated by the even–odd rule

To find the number of intersections of the horizontal ray with the polygon edge, we break the polygon edge into segments, where each segment is a straight line connecting one point on the edge to the next point. We consider only those segments of the polygon edge that cross the latitude of the point of interest. That is to say, we consider those segments where one endpoint has a larger x coordinate than the point of interest and the other endpoint has a smaller x coordinate.

We can then divide these segments into three classes: class A consists of those segments where both endpoints lie to the right of (for example, have a larger x coordinate than) the point of interest; class B consists of those where both endpoints lie to the left of (for example, have a smaller x coordinate than) the point of interest; and class C consists of those segments where one endpoint lies to the left and the other to the right of the point of interest.

Recall that we want to count the number of intersections of the polygon edge with the horizontal ray that starts at the point of interest and projects rightward to infinity. Edge segments in class A definitely intersect this ray, while edge segments in class B definitely do not intersect this ray. Edge segments in class C may or may not intersect this ray and so may require further testing.

Because we are using Mata for the point-in-polygon algorithm, it is more efficient to calculate the intersects of the segments in both class A and class C as a matrix operation as opposed to subsampling them into two distinct groups first. In doing so, we calculate all the points where the line segments of both A and C cross the latitude of the point of interest. If this crossing point is to the right of the point of interest, then the segment does intersect the ray. To do this, we determine the straight-line equation that describes the segment, and then we substitute our y coordinate from our point of interest into the equation to determine the x coordinate of the crossing point.

After calculating all the points of intersection for both class A and class B, we then count how many elements of our results matrix are greater than or equal to the x coordinate of our point of interest.

As a final step, the count is determined to be either even or odd, and this result is returned by the algorithm. If the algorithm returns the result as odd, the index of that shapefile is returned to Stata. This is then used to merge in the attribute table that was imported earlier.

5.7 Joining the attribute data to the GPS coordinates

Finally, the two data results (the binary variable indicating if mapping to the shapefile was possible and the second variable containing the shapefile polygon identification number) are returned to Stata. At this point, the dBASE file[3] that was imported into Stata by the first subcommand is merged to the mapping results on the shapefile polygon identification number.

6 Stored results

`gpsbound` does not store any results because if the GPS coordinate is validated, then the new variables are imported into the Stata dataset in memory directly from the attribute table of the shapefile.

7 Conclusion

Spatial data enable a variety of statistical analyses. The ability to selectively import additional spatial information into datasets from shapefiles, combined with the ability to check the accuracy of GPS coordinates, is crucial when using more advanced statistical methods. `gpsbound` makes an important contribution to the Stata user community in this respect. Not only does it enable researchers to increase the scope of plausible analytical methods applicable to spatial datasets, but it also enables survey methodologists to verify in-field operational concerns in real time. For `gpsbound` to operate correctly, GPS coordinates must be recorded as, or converted to, decimal-degree format; and shapefiles must be polygon shapefiles rather than point or line shapefiles. With these basic inputs, the full functionality of `gpsbound` can be realized.

3. In versions of Stata before Stata 13, the string variables from the dBASE file are limited to 244 characters in length because this is the Stata limit on the length of string variables. From Stata 13 onward, this is no longer a limitation because of the introduction of the `strL` data type by Stata.

8 Acknowledgments

We are grateful to staff members at the NIDS at the SALDRU for providing the time and space to adequately test the algorithm. We are especially grateful to Adrian Frith for his invaluable comments on earlier drafts of this article. Lastly, the authors would like to thank participants at the 2013 Stata Conference in New Orleans for valuable feedback on an earlier version of this algorithm called `gpsmap`. This name was subsequently changed to `gpsbound` because the authors discovered that GPSMAP is a registered trademark of the Garmin Corporation.

9 References

Crow, K. 2006. shp2dta: Stata module to convert shape boundary files to Stata datasets. Statistical Software Components S456718, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s456718.html.

dBASE. 2014. Data file header structure for the dBASE version 7 table file. http://www.dbase.com/Knowledgebase/INT/db7_file_fmt.htm.

Environmental Systems Research Institute (ESRI). 1998. ESRI shapefile technical description. Technical report, Environmental Systems Research Institute (ESRI). http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

Spatial Reference. 2007. EPSG: 4326: WGS84. http://www.spatialreference.org/ref/epsg/wgs-84/.

World Wide Web Consortium (W3C). 2011. Painting: Filling, stroking and marker symbols. http://www.w3.org/TR/SVG/painting.html#FillProperties.

About the authors

Tim Brophy is the senior data analyst at NIDS of South Africa, which is conducted by the SALDRU at the University of Cape Town.

Reza Daniels is a senior lecturer in the School of Economics at the University of Cape Town.

Sibongile Musundwa is a research intern at NIDS of South Africa, which is conducted by the SALDRU at the University of Cape Town.