MWSUG 2025 – Paper AL-037

GIS Challenges of Cataloging Catastrophes: Serving up GeoWaffles with a Side of Hash Tables to Conquer Big Data Point-in-Polygon Determination and Supplant SAS® PROC GINSIDE

Troy Martin Hughes

ABSTRACT

The GINSIDE procedure represents the SAS® solution for *point-in-polygon* determination—that is, given some point on earth, does it fall inside or outside of one or more bounded regions? Natural disasters typify geospatial data—the coordinates of a lightning strike, the epicenter of an earthquake, or the jagged boundary of an encroaching wildfire—yet observing nature seldom yields more than latitude and longitude coordinates. Thus, when the United States Forestry Service needs to determine in what zip code a fire is burning, or when the United States Geological Survey (USGS) must ascertain the state, county, and city in which an earthquake was centered, a point-in-polygon analysis is inherently required. It determines within what boundaries (e.g., nation, state, county, federal park, tribal lands) the event occurred, and confers boundary attributes (e.g., boundary name, area, population) to that event. Geographic information systems (GIS) that process raw geospatial data can struggle with this time-consuming yet necessary analytic endeavor—the attribution of points to regions. This text demonstrates the tremendous inefficiency of the GINSIDE procedure, and promotes *GeoWaffles* as a far faster alternative that comprises a mesh of rectangles draped over polygon boundaries. This facilitates memoization by running point-in-polygon analysis only once, after which the results are saved to a hash object for later reuse. *GeoWaffles* debuted in the 2013 white paper *Winning the War on Terror with Waffles: Maximizing GINSIDE Efficiency for Blue Force Tracking Big Data* (Hughes, 2013), and this text represents an in-memory, hash-based refactoring. All examples showcase USGS tremor data as GeoWaffles tastefully blow GINSIDE off the breakfast buffet—**processing coordinates more than 25 times faster than the out-of-the-box SAS solution**!

WHY CATASTROPHES NEED POINT-IN-POLYGON…AND GEOWAFFLES

When an earthquake strikes, seismic detectors around the world collectively determine its location, depth, magnitude, and other sciency attributes—but *not* the boundaries in which the earthquake occurred. The USGS website demonstrates these basic data—time, location, and depth—in its interactive map, as shown in Figure 1, but other attributes must be conferred through point-in-polygon methods.

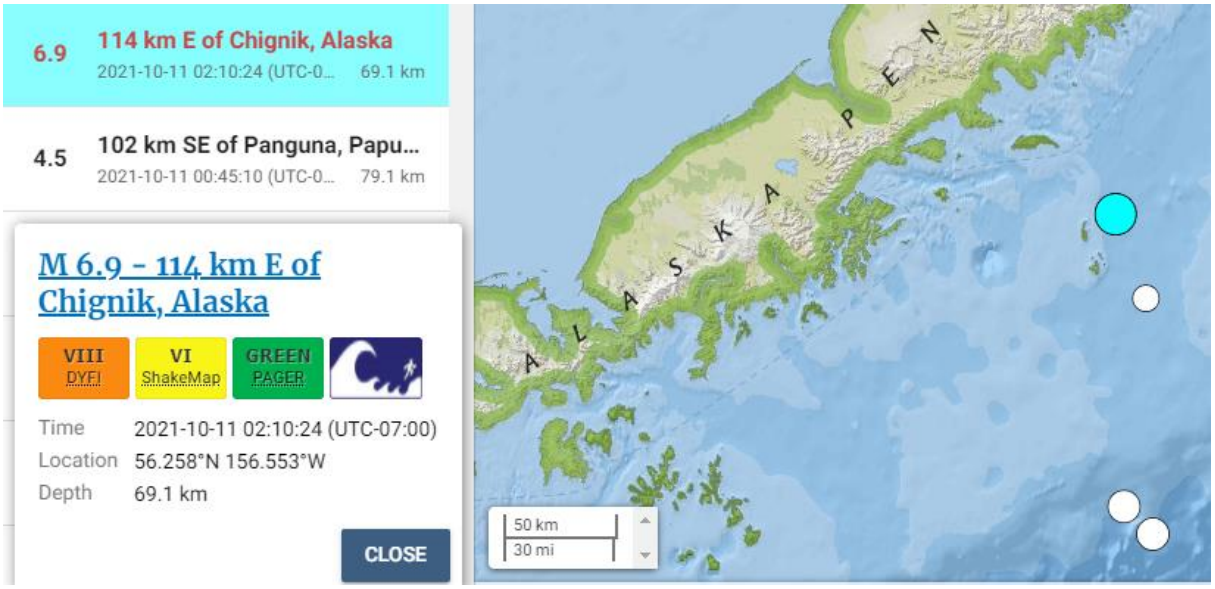


Figure 1. USGS Interactive Map (https://earthquake.usgs.gov/earthquakes/map/)

Although California's Parkfield, an unincorporated town in Monterey County, boasts its status as the "Earthquake Capital of the World," it is Alaska that experiences more earthquakes a year than any other state. But how would you determine who takes the silver and bronze medals in this inarguably unenviable competition? Well, that's where point-in-polygon can bestow geospatial attributes (such as state, county, and population density of the bounded region) to raw USGS earthquake event data. With this information, policymakers can better understand the prevalence and threat of earthquakes in their respective constituencies, and data-driven decision-making can both facilitate more apt and targeted preventative measures, and support first responders in their coordination of rescue and recovery efforts.

These worthwhile pursuits can be stymied, however, where recurrent data analyses or time-sensitive decisions are delayed due to delinquent data processing. GIS extract-transform-load (ETL) routines commonly rely on point-in-polygon determination, and yet, this is a costly operation in any software language (i.e., SAS is not alone in its slowness) due to the algorithms (such as *ray casting*, not further discussed) that are utilized. *Memoization* is the natural solution, which delivers both increased speed and efficiency by computing point-in-polygon for millions of coordinates, storing the results, and accessing these pre-computed results for all future point-in-polygon requests. GeoWaffles of varying sizes are computed for boundaries that are relatively stable and commonly accessed, allowing more than 99 percent of points to be analyzed immediately through an in-memory hash lookup, with only a few straggling points having to pass through the inefficient GINSIDE procedure. GeoWaffles thus reduce not only software runtime but also memory utilization, making them the clear choice for point-in-polygon analysis.

SETUP FOR SUBSEQUENT EXAMPLES

The set of 2020 earthquake data within the Continental US can be downloaded from the USGS earthquake catalog (https://earthquake.usgs.gov/earthquakes/search/). Figure 2 demonstrates this first step, by selecting all earthquakes of at least a magnitude of 0.1, using the Pacific Standard Time (offset from UTC), and selecting the conterminous US. Data can be downloaded to a comma-separated values (CSV) file.

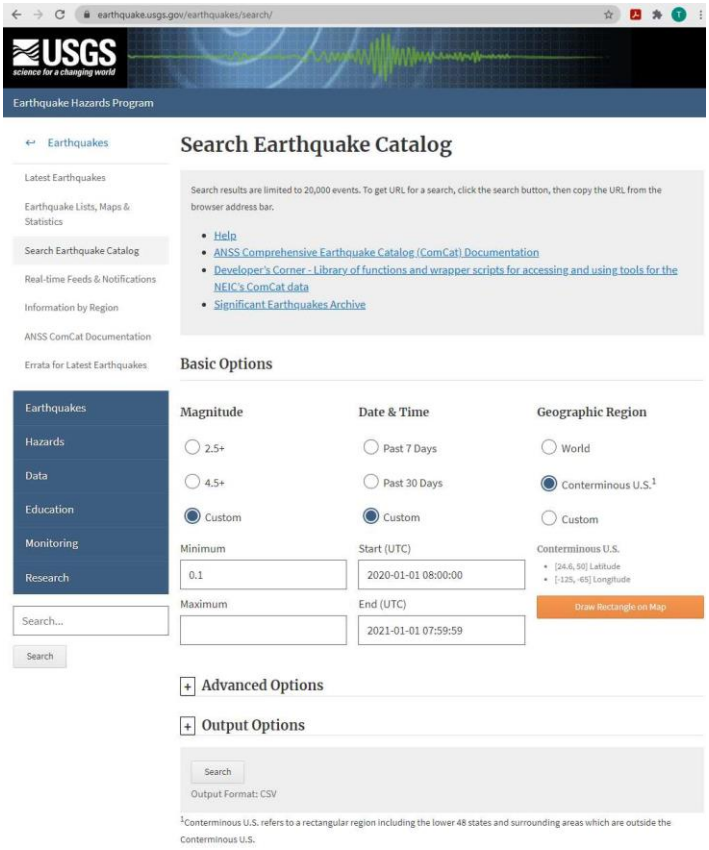


Figure 2. USGS Earthquake Catalog (https://earthquake.usgs.gov/earthquakes/search/)

USGS provides more advanced and practical data query and download methods, including its API (https://earthquake.usgs.gov/fdsnws/event/1/), but for simplicity, CSV is demonstrated in this text. The downloaded CSV file has been named Quakes_2020_continental_US.csv, and the following code imports the data into a SAS data set:

```
%let quake_loc=D:\sas\geowaffles\USGS_quakes\; * USER MUST CHANGE LOCATION *;
proc import datafile="&quake_loc.quakes_2020_continental_US.csv" out=quakes_raw
      dbms=csv replace;
   getnames=yes;
   guessingrows=100000;
run;
```

SAS graphing procedures typically use the variables X to represent longitude and Y to represent latitude, so the next code renames the default USGS variables (latitude and longitude), and eliminates non-earthquake data (such as chemical explosions and mining collapses):

```
data quakes (keep=x y time depth mag);
   set quakes_raw (rename=(latitude=y longitude=x));
   where type='earthquake';
run;
```

Point-in-polygon analysis requires two data sets—a *points* data set, representing earthquake epicenters in this example, and a *polygon* data set, representing one or more boundaries whose attributes should be conferred to all interior points. In addition to the SAS out-of-the-box map data sets, the US Census TIGER/Line repository (https://www.census.gov/cgi-bin/geo/shapefiles/index.php) has amassed a treasure trove of shapefiles that enumerate various boundaries and their statistics. All examples in this text rely on the 2020 "Counties (and equivalent)" shapefile, demonstrated in Figure 3.

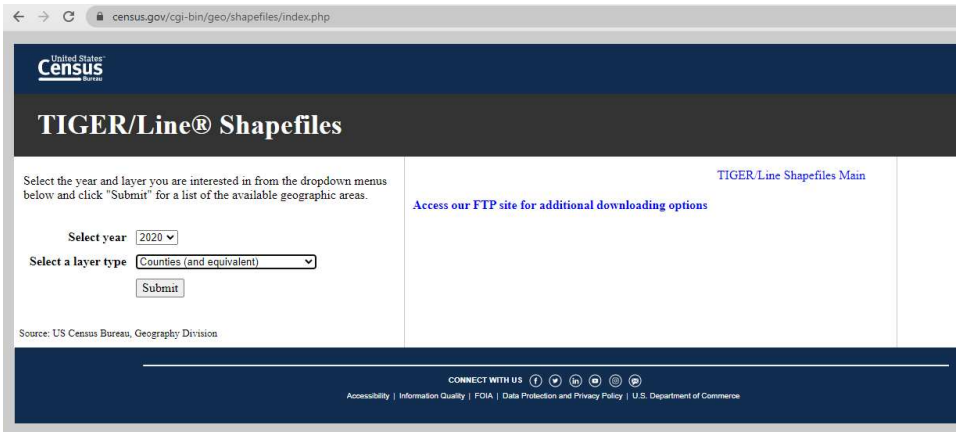


Figure 3. US Census TIGER/Line Shapefile Repository

The downloaded zip file comprises seven shapefile-associated files, shown in Figure 4.

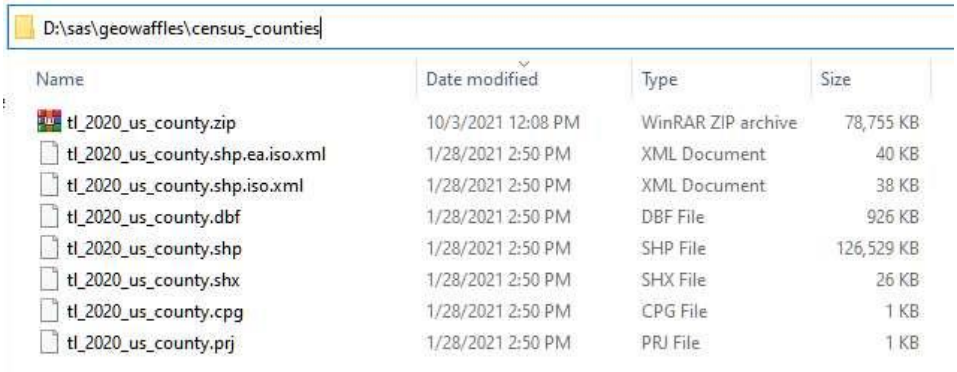


Figure 4. US Census 2020 "Counties (and equivalent)" Shapefile Download

The following code imports the shapefile using the MAPIMPORT procedure, after which the X and Y coordinates, polygon segment, and state and county FIPS codes are retained:

```
libname mymaps 'D:\sas\geowaffles\maps'; * USER MUST CHANGE LOCATION *;
%let counties_loc=D:\sas\geowaffles\census_counties\;  * MUST CHANGE LOCATION *;
proc mapimport out=counties_raw datafile="&counties_loc.tl_2020_us_county.shp";
run;

* statefp and countyfp must be transformed from character to numeric variables;
data mymaps.counties (keep=x y segment statefips countyfips);
   length statefips countyfips 8;
   set counties_raw;
   statefips=input(statefp,8.);
   countyfips=input(countyfp,8.);
run;
```

Note that two folders must be created—Census_counties, to hold the raw shapefile, and Maps, to initialize the MYMAPS SAS library. The GMAP procedure provides a quick rendering of the shapefile, with the WHERE option removing Alaska, Hawaii, and the US territories to display only the Continental US:

```
proc gmap map=mymaps.counties
      (where=(1 <= statefips <=56 and statefips ^in(2,15))) data=mymaps.counties;
   id statefips countyfips;
   choro statefips countyfips /nolegend discrete;
run;
quit;
```

Figure 5 demonstrates the output of the GMAP procedure, and maps all counties in the Continental US.

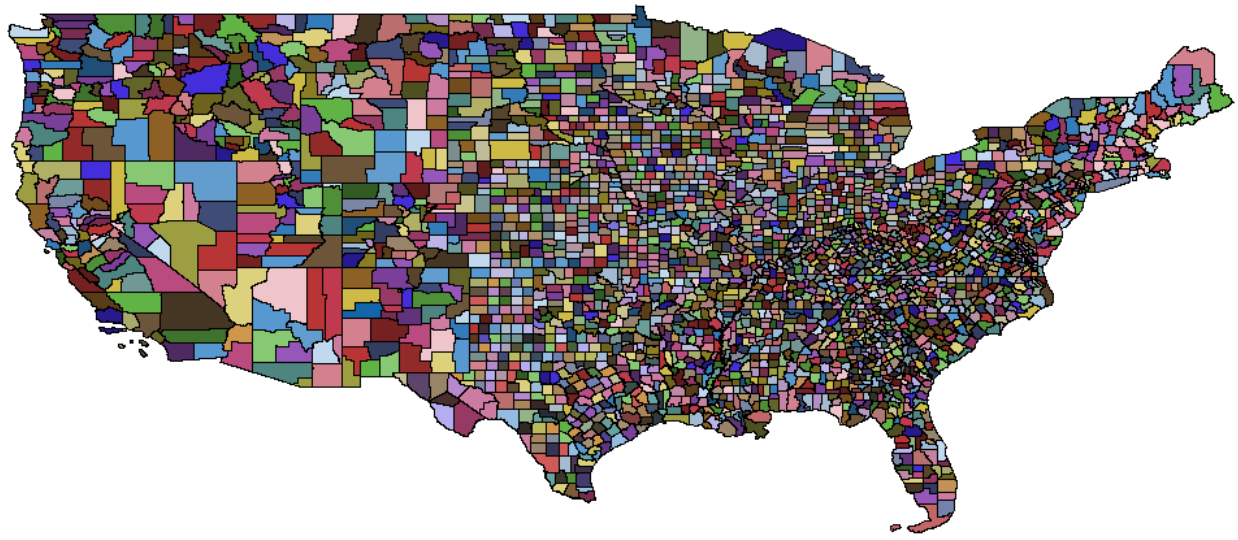


Figure 5. US Census Geographic - GCS North American NAD83 Projection of Continental US Counties

To be clear, point-in-polygon determination is mathematically calculated without the need to visualize point data or shapefile boundaries; however, data visualization can help introduce the somewhat arcane GeoWaffle methods and demonstrate how they differ from the GINSIDE procedure.

Despite its inefficiency, the GINSIDE procedure is straightforward to implement, and processes 97,416 earthquake events in approximately 16 seconds:

```
option fullstimer;
proc ginside data=quakes map=mymaps.counties out=counties_quakes
      includeborder dropmapvars;
   id statefips countyfips;
run;
```

The FULLSTIMER option causes SAS to print more verbose (maximum verbosity, anyone?) performance metrics to the SAS log, and is critical when conducting performance analysis. For example, the relative equivalence of Real Time to User CPU Time provides a good indication that the GINSIDE procedure is unfortunately single-threaded rather than multithreaded:

```
NOTE: The data set WORK.COUNTIES_QUAKES has 97416 observations and 8 variables.
NOTE: PROCEDURE GINSIDE used (Total process time):
      real time           16.88 seconds
      user cpu time       14.15 seconds
      system cpu time     2.73 seconds
      memory              28164.12k
      OS Memory           53708.00k
```

The ID statement identifies Statefips and Countyfips as the identification variables that represent the polygon(s) in which each point may fall. As states are superordinate to their counties, Statefips must precede Countyfips whenever the variables are listed, including when GMAP and GINSIDE procedures are executed.

Table 1 demonstrates the Counties_quakes data set that GINSIDE produced, in which the Statefips and Countyfips variables were generated for each earthquake and appended to the Quakes data set. In the first observation, the Statefips of 6 (i.e., California) and Countyfips of 97 (i.e., Sonoma County) confers where the earthquake struck, whereas the second observation (missing both identification variables) denotes an earthquake that occurred *outside* the Continental US—which occurs infrequently because the USGS rectangular data export captures parts of northern Mexico and southern Canada.

	x	y	_ONBORDER_	statefips	countyfips	time	depth	mag
2934	-122.8273333	38.8146667	0	6	97	20200214T064541+0000	2.21	0.26
2935	-115.2881667	32.2776667	0	.	.	20200214T064538+0000	21.77	2.02
2936	-117.661	35.8373333	0	6	27	20200214T062835+0000	5.88	0.55
2937	-120.0006667	37.2943333	0	6	43	20200214T061111+0000	23.82	1.64
2938	-118.8385	37.4676667	0	6	19	20200214T060525+0000	3.32	0.82
2939	-112.5245	46.8893333	0	30	49	20200214T060325+0000	13.72	1.13
2940	-108.7133	38.8506	0	8	77	20200214T060119+0000	1.82	1.1

Table 1. Counties_quakes Data Set Produced by GINSIDE Procedure

With point-in-polygon now determined, you can use the FREQ procedure to uncover the silver- and bronze-medal recipients for shakiest state:

```
title1;
title2;
ods noproctitle;
proc freq data=counties_quakes order=freq;
   tables statefips / maxlevels=2 nocum;
run;
```

The output, shown in Figure 6, demonstrates that California follows Alaska and experienced 57.8 percent of Continental US earthquakes in 2020, with Nevada having 23.4 percent:

statefips	Frequency	Percent
6	55821	57.82
32	22632	23.44
The first 2 levels are displayed.		
Frequency Missing = 866		

Figure 6. Continental US Earthquakes in 2020

But unless you've memorized the state FIPS table, the output could be improved by applying a format to transform the FIPS state codes into their two-letter abbreviations.

First, the FORMAT procedure transforms the FIPSTATE built-in function (which converts FIPS state codes to state abbreviations) into the user-defined FIPSTATE function, which enables this format to be applied from within the FREQ procedure:

```
proc format;
   value fipstate
      other=[fipstate()];
run;

proc freq data=counties_quakes order=freq;
   format statefips fipstate.;
   tables statefips / maxlevels=4 nocum;
run;
```

The output in Figure 7 more clearly identifies the states.

statefips	Frequency	Percent
CA	55821	57.82
NV	22632	23.44
The first 2 levels are displayed.		
Frequency Missing = 866		

Figure 7. Continental US Earthquakes in 2020

SAS documentation should be consulted for a more *exhaustive* review of GINSIDE statements and options—"exhaustive" because you'll need some reading material as you kill time waiting endlessly for GINSIDE to complete a large job. Thus, as data set size (and, especially, observation count) increases, GINSIDE can become cumbersome or fail altogether due to resource limitations such as out-of-memory errors. It was to overcome these obstacles that GeoWaffle methods were developed in 2013 to support the daily requirement of the Department of Defense (DoD) to determine point-in-polygon results for BILLIONS of observations—a task that GINSIDE could not handle given existing system resources.

Note that this use case met all three requirements for successful GeoWaffle utilization. First, the big data (i.e., points data sets) were sufficiently "big" to cause GINSIDE runtimes that were subjectively deemed to be "too slow," with GINSIDE often objectively failing entirely. Second, the maps (i.e., polygon data sets) against which the points were being analyzed were stable; they typically represented Afghanistan and Iraq districts and provinces, thus GeoWaffles could be calculated once and remain unchanged. Third, the requirement for point-in-polygon determination was recurrent—that is, the one-time, up-front cost of generating GeoWaffle data sets reaped indescribable efficiency benefits for years, given the thousands of times that those GeoWaffles were reused to process various points data sets.

CASE STUDY IN GIS: CALIFORNIA 2020 EARTHQUAKES

This section analyzes California earthquakes, contrasting GINSIDE and GeoWaffle methods. Point-in-polygon efficiency is always improved when points are limited to only those of interest, so the first step is to subset only relevant data—which requires a reduction in both points and polygons.

The USGS Earthquake Catalog interface allows users to select a rectangle around a customized region and to download earthquakes centered only in that region. By selecting "Custom" on the "Geographic Region" section, a rectangle can be drawn around the area for which earthquake data are sought. Figure 8 demonstrates a customized rectangle that selects all California and Nevada earthquake events.

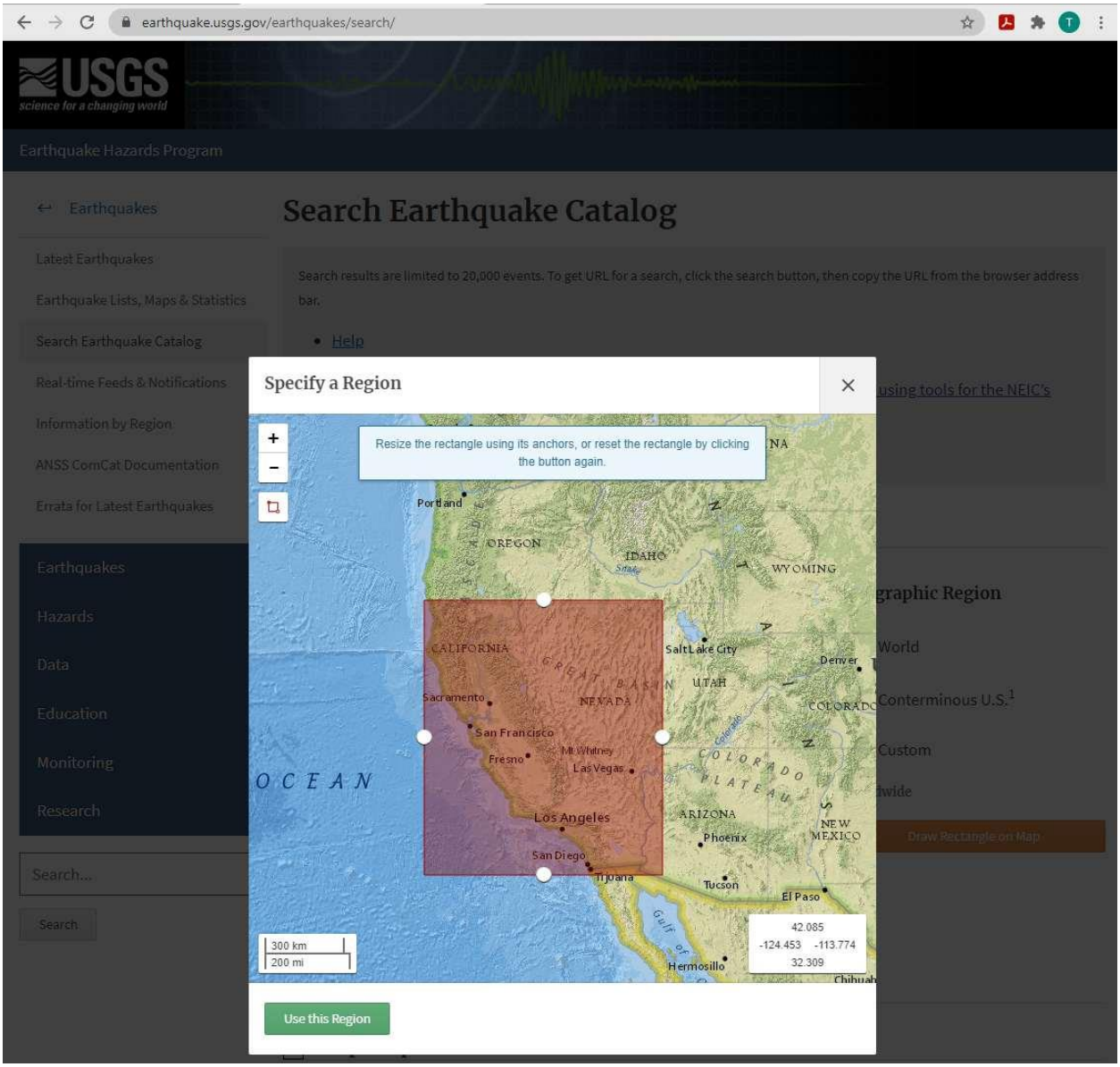


Figure 8. USGS Earthquake Catalog Custom Data Download

The data, which contain all 2020 earthquakes occurring in California, Nevada, and surrounding border regions, can be saved as Quakes_2020_CA_NV.csv. The IMPORT procedure then converts the CSV file to a SAS data set:

```
%let quake_loc = D:\sas\geowaffles\USGS_quakes\; * USER MUST CHANGE LOCATION *;
proc import datafile="&quake_loc.quakes_2020_CA_NV.csv"
      out=quakes_raw dbms=csv replace;
   getnames=yes;
   guessingrows=100000;
run;
data quakes_CA_NV (keep=x y time depth mag);
   set quakes_raw (rename=(latitude=y longitude=x));
   where type='earthquake';
run;
```

The next step is to filter the map data (i.e., MYMAPS.Counties, which was created in the previous section) to include only California data:

```
libname mymaps 'D:\sas\geowaffles\maps'; * USER MUST CHANGE LOCATION *;
data mymaps.CA_counties;
   set mymaps.counties;
   where statefips=6;
run;
```

The resultant map data set (MYMAPS.CA_counties) will be used by the GINSIDE and GMAP procedures, as well as by the GeoWaffle macros. The out-of-the-box SAS solution relies on the GINSIDE procedure to determine point-in-polygon membership:

```
option fullstimer;
proc ginside data=quakes_CA_NV map=mymaps.CA_counties out=county_quakes_CA_GINSIDE
      includeborder dropmapvars;
   id statefips countyfips;
run;
```

The GINSIDE procedure completes in fewer than three seconds, much faster than the larger Continental US earthquake data set, and much faster because the map data set now includes only California counties:

```
NOTE: The data set WORK.COUNTY_QUAKES_CA_GINSIDE has 78859 observations and 8
variables.
NOTE: PROCEDURE GINSIDE used (Total process time):
      real time           2.43 seconds
      user cpu time       1.71 seconds
      system cpu time     0.76 seconds
      memory              23445.46k
      OS Memory           45232.00k
```

So, if a procedure executes this quickly, why was there a need to spend dozens of hours to re-engineer it? Because GINSIDE fails to analyze big data efficiently, and therein lies the impasse to organizations that require recurrent point-in-polygon calculation for big data.

Consider a data set with 100 times the number of observations—7,885,900 rather than 78,559—which, inarguably, still does not begin to approach actual “big data” recognition:

```
data quakes_CA_NV_bigdata (drop=i);
   set quakes_CA_NV;
   do i=1 to 100;
      output;
      end;
run;
```

The GINSIDE procedure is rerun on the new, larger data set:

```
proc ginside data=quakes_CA_NV_bigdata map=mymaps.CA_counties
out=county_quakes_CA_GINSIDE_bigdata
      includeborder dropmapvars;
   id statefips countyfips;
run;
```

What you find, as demonstrated hereafter in a subsequent section, is that GINSIDE performance is fairly linear and is correlated with observation count of the points data set being analyzed:

```
NOTE: The data set WORK.COUNTY_QUAKES_CA_GINSIDE_BIGDATA has 7885900 observations
and 8 variables.
NOTE: PROCEDURE GINSIDE used (Total process time):
      real time           2:56.51
      user cpu time       2:36.68
      system cpu time     4.75 seconds
      memory              1610239.00k
      OS Memory           1632208.00k
```

Thus, as the number of observations continues to increase, so too does GINSIDE runtime, until its slow performance can be deemed unacceptable. This is the point where GeoWaffles can save the day.

GeoWaffles are created using the MAKE_GEOWAFFLES macro (see Appendix A), which is saved in the GeoWaffles.sas program file. This one-time setup, the *memoization* process, comprises the resource-intensive operations that are run only once, after which their results are stored and available for later use (via a hash object lookup).

Thus, the first step is to save the GeoWaffles program to a folder from which it can be called using the %INCLUDE statement. In this example, GeoWaffles.sas has been saved to D:\sas\geowaffles:

```
%include 'D:\sas\geowaffles\geowaffles.sas'; * USER MUST CHANGE LOCATION *;
libname waffles 'D:\sas\geowaffles\waffles'; * USER MUST CHANGE LOCATION *;
```

The WAFFLES library is also initialized, which should correspond to the folder location where GeoWaffle data sets are permanently stored. GeoWaffles themselves are point-agnostic, so they ideally should be saved with other non-project data such as shapefiles and map products. Thus, you would not want to save GeoWaffles to a location corresponding to an earthquake analysis—because you could reuse those same GeoWaffles for later analyses of lightning strikes, wildfires, and other events unrelated to earthquakes.

The MAKE_GEOWAFFLES macro (explained in detail in the next section) is executed and creates the GeoWaffles:

```
%make_geowaffles(maplib=mymaps, mapname=CA_counties, wafflelib=waffles,
   idlist=statefips countyfips, iterations=4);
```

Once the GeoWaffles have been created, they can be used in the future by invoking the EAT_GEOWAFFLES macro:

```
%eat_geowaffles(pointslib=work, pointsname=quakes_CA_NV_bigdata,
   wafflelib=waffles, wafflename=CA_counties_waf_1000,
   maplib=mymaps, mapname=CA_counties,
   idlist=statefips countyfips, iterations=4,
   xvar=x, yvar=y,
   outfile=county_quakes_CA_GeoWaffles_1000);
```

A sample SAS log demonstrates that EAT_GEOWAFFLES took only 11.6 to execute, and compared with the GINSIDE procedure taking 296 seconds, **GeoWaffles are more than 25 times faster than GINSIDE, the out-of-the-box SAS solution for point-in-polygon determination!**

```
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      memory              5226.71k
      OS Memory           28160.00k

NOTE: There were 7885900 observations read from the data set
WORK.QUAKES_CA_NV_BIGDATA.
NOTE: The data set WORK.POINTS has 7885900 observations and 13 variables.
NOTE: DATA statement used (Total process time):
      real time           8.40 seconds
      user cpu time       7.65 seconds
      system cpu time     0.54 seconds
      memory              714.37k
      OS Memory           23036.00k

NOTE: There were 1587490 observations read from the data set
WAFFLES.CA_COUNTIES_HSH_1000.
NOTE: There were 7885900 observations read from the data set WORK.POINTS.
NOTE: The data set WORK.COUNTY_QUAKES_CA_GEOWAFFLES_1000 has 7845500 observations
and 8
      variables.
NOTE: The data set WORK.NOTHASHED has 40400 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           2.10 seconds
      user cpu time       1.70 seconds
```

```
      system cpu time     0.40 seconds
      memory              166292.34k
      OS Memory           188144.00k

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.01 seconds
      user cpu time       0.00 seconds
      system cpu time     0.01 seconds
      memory              5186.93k
      OS Memory           28160.00k

NOTE: There were 240600 observations read from the data set MYMAPS.CA_COUNTIES.
NOTE: The data set WORK.MAP_SEG has 240600 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      user cpu time       0.01 seconds
      system cpu time     0.00 seconds
      memory              635.03k
      OS Memory           23036.00k

NOTE: The data set WORK.GOUT has 40400 observations and 8 variables.
NOTE: PROCEDURE GINSIDE used (Total process time):
      real time           1.04 seconds
      user cpu time       1.04 seconds
      system cpu time     0.00 seconds
      memory              14569.31k
      OS Memory           36356.00k

NOTE: Appending WORK.GOUT to WORK.COUNTY_QUAKES_CA_GEOWAFFLES_1000.
NOTE: There were 40400 observations read from the data set WORK.GOUT.
NOTE: 40400 observations added.
NOTE: The data set WORK.COUNTY_QUAKES_CA_GEOWAFFLES_1000 has 7885900 observations
and 8
      variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.00 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      memory              753.62k
      OS Memory           23036.00k
```

In brief, the log provides insight into how GeoWaffles pull off this inexplicable feat. As the points data set containing 7,8885,900 observations is ingested, nearly all (N=7,845,500, or 99.49 percent) observations have point-in-polygon determination made through the hash object. Only a meager 40,400 observations could not be found (i.e., fell outside of all GeoWaffles) and thus require the far more costly GINSIDE procedure to be run on them. Now, consider having to run GINSIDE on *actual* big data, which might take hours to execute, and imagine the time savings that GeoWaffles can provide.

BUT HOW IS THE SAUSAGE—OR WAFFLE—MADE?

The secret sauce in GeoWaffles is hash. This in-memory data object enables millions of pre-formed GeoWaffles to be read into memory in seconds, after which millions—or even billions—of points can be evaluated against those GeoWaffles. But behind the scenes, a complex tapestry of waffles are being woven, and this section helps to elucidate how they can improve performance so substantially.

When MAKE_GEOWAFFLES executes, it first evaluates the minimum and maximum latitude and longitude points for the map file, and determines the rectangle that can hold all map coordinates. Each large rectangle is subsequently divided into one-degree by one-degree (latitude/longitude) smaller rectangles, which are then outlined with 40,000 points—10,000 points per side, equidistantly placed 1/10000th of a degree apart.

These 40,000 points outlining each box are subsequently run through GINSIDE with three possible outcomes. First, if all points outlining a box are found to lie *outside* the map data set, then an *exterior*

GeoWaffle is generated, which is defined by the four corner points of the box. Conversely, if all points outlining a box are found to lie *inside* the map data set, then an *interior* GeoWaffle is generated, which again is defined by the four corner points of the box. Finally, if a rectangle straddles one or more boundaries of the map data set (i.e., transverses one or more polygons of the shapefile), then that box is saved for subsequent iterative processing—thrown into the Crumbs temporary SAS data set.

Figure 9 demonstrates the first iteration, in which 121 (i.e., 11 X 11) one-degree latitude/longitude boxes are generated and tested for inclusion/exclusion in the map data set. Note that in this first pass, all one-degree boxes fell outside the shapefile bounds, and thus comprise exterior GeoWaffles.

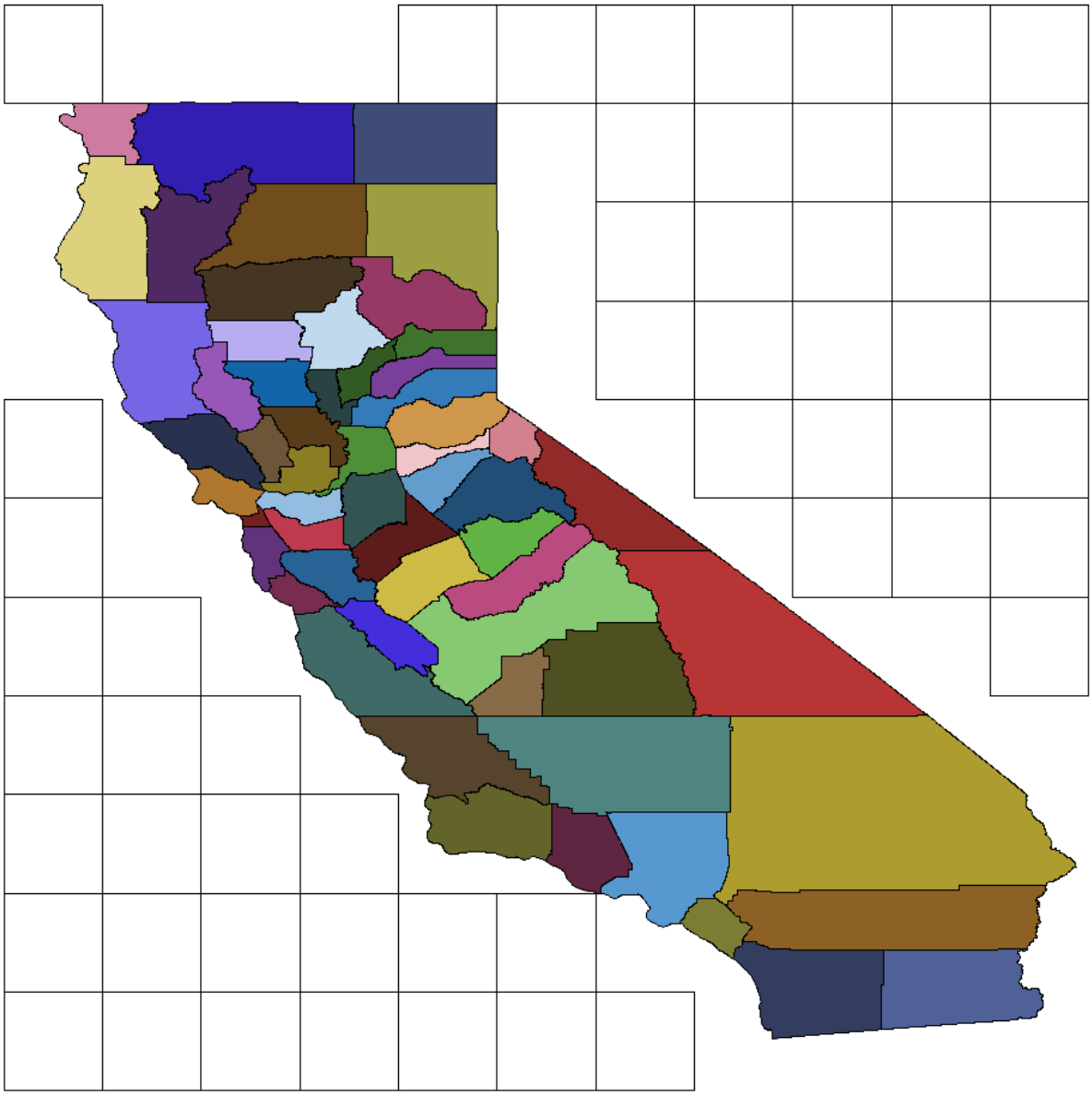


Figure 9. MAKE_GEOWAFFLES Iteration 1 with One-Degree Latitude/Longitude GeoWaffles

Note that all boxes that straddled one or more boundaries (not displayed in Figure 9) were saved to the Crumbs temporary data set. After MAKE_GEOWAFFLES has completed its first iteration, it continues crunching the Crumbs data set, but this time divides each one-degree rectangle into 100 1/10th of one-degree boxes. The same process continues; each box is outlined with points (now 4,000 rather than 40,000,

as the box is 10 times smaller) and is determined to lie outside the shapefile, inside the shapefile, or straddle one or more polygon boundaries. Figure 10 demonstrates this second iteration, showing both large- and medium-sized exterior GeoWaffles.

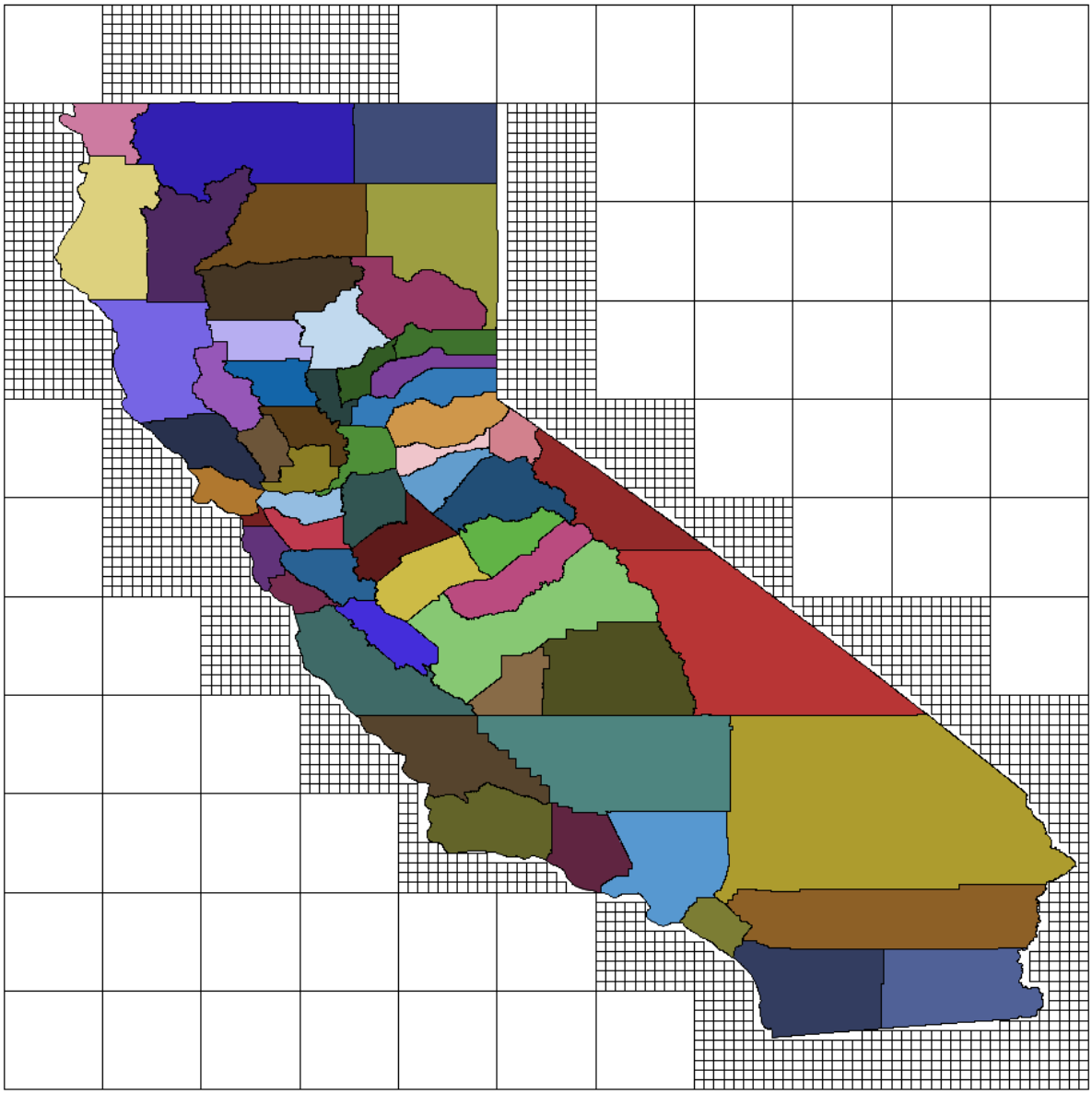


Figure 10. MAKE_GEOWAFFLES Iteration 2 with 1/10th of One Degree Latitude/Longitude GeoWaffles

Note that Figure 10 displays only exterior GeoWaffles, as the interior GeoWaffles are eclipsed by the map shading of the counties. As part of each MAKE_GEOWAFFLES iteration, however, the macro generates a GeoWaffles map to demonstrate coverage of the map region. And, with each successive iteration, the GeoWaffles themselves—without the map shapefile displayed—more closely approximate the actual map, with only the map boundaries and their immediate vicinity not covered by GeoWaffles.

For example, Figure 11 demonstrates the actual map created by the second iteration of MAKE_GEOWAFFLES. Note that although the county shapefiles are no longer displayed, the smaller GeoWaffles nevertheless give a fair representation of where the county borders lie.

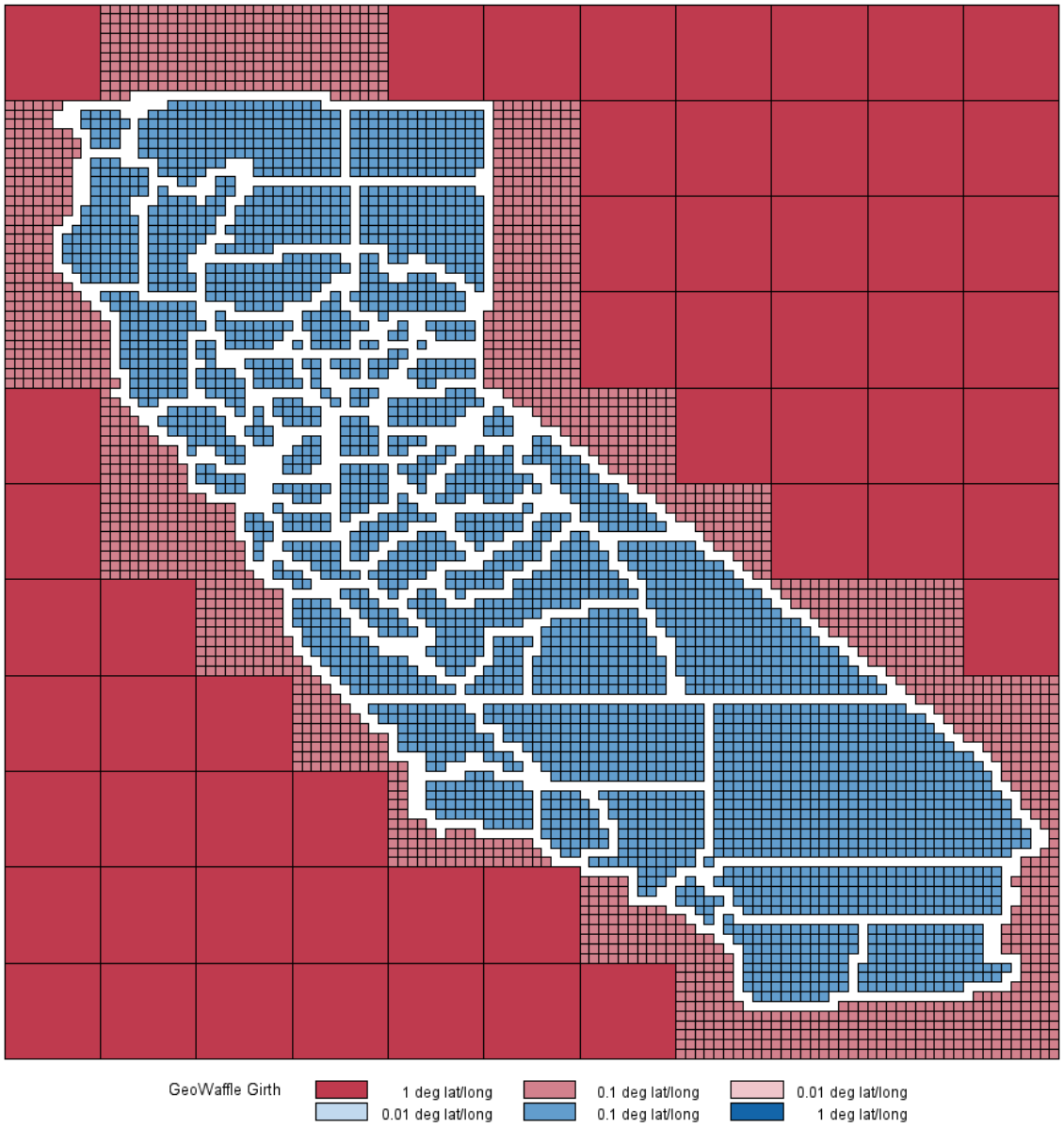


Figure 11. MAKE_GEOWAFFLES Iteration 2 Map of California GeoWaffles

Note that after the first iteration of MAKE_GEOWAFFLES, only 55 GeoWaffles had been generated—all exterior, as they lay outside the boundaries of California. However, after the second iteration, the number of GeoWaffles had increased to 5,224—all those tiny boxes shown in Figure 11. And, by the third iteration, 660,415 GeoWaffles had been created. From a data visualization perspective, with each successive iteration of MAKE_GEOWAFFLES, the map boundaries are smoothed and refined.

For example, Figure 12 demonstrates a subset of the California map data set—the Santa Barbara coastline—and shows the infinitesimally tiny GeoWaffles produced by the third iteration of the macro. At first glance, you'd probably imagine the white lines to be the normal rendering of a shapefile. However, upon closer inspection, these boundaries are shown instead to comprise hundreds of thousands of tiny boxes. That is, the white lines are in fact the interstices that lie among the shaded GeoWaffles.

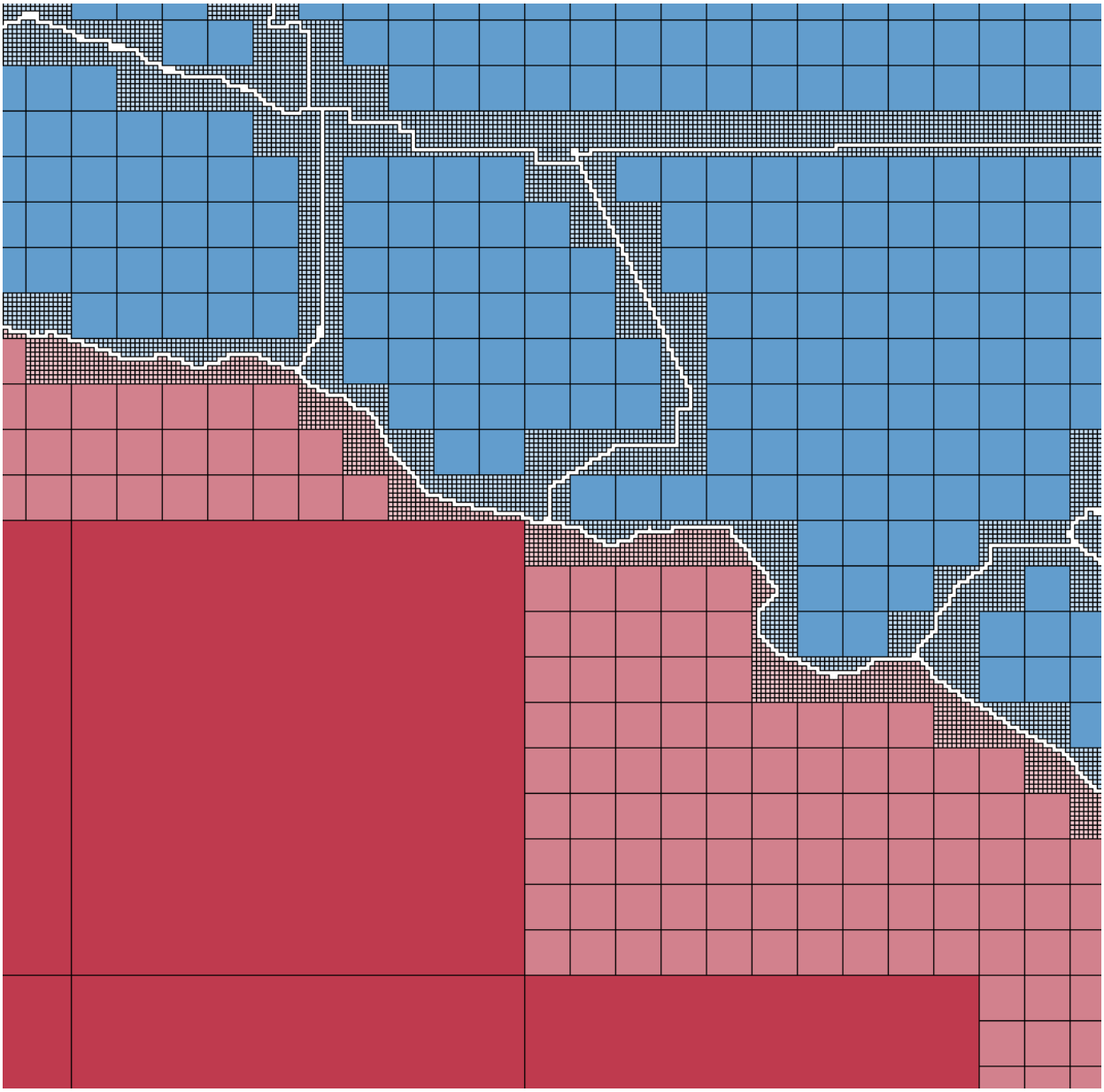


Figure 12. MAKE_GEOWAFFLES Iteration 3 Map of California GeoWaffles: Santa Barbara Coastline

By the fourth iteration of MAKE_GEOWAFFLES (in which each GeoWaffle dimensions are only 1/1000th of one degree latitude/longitude), 1,587,490 GeoWaffles have been created, and maps are no longer generated.

As illustrative as these maps are about GeoWaffle production methods, it is worth stressing again that point-in-polygon determination is a purely mathematical endeavor that does not require data visualization. Thus, MAKE_GEOWAFFLES generates two output data sets—a GeoWaffle data set, which is used solely to generate maps (such as Figures 11 and 12), and a hash data set, which is used solely by the EAT_GEOWAFFLES macro to perform point-in-polygon determination for points data sets.

A subset of the GeoWaffles data set (WAFFLES.CA_counties_waf_1000) appears in Table 2, in which three 1/10th of one degree GeoWaffles are demonstrated. Each GeoWaffle is a rectangular polygon and thus comprises five points, representing the four corners of the rectangle with the first/last corner represented twice to “close” the polygon. A distance of 1 (or -1) corresponds to a one degree sided GeoWaffle; a distance of 0.1 (or -0.1) corresponds to a 1/10th of one degree sided GeoWaffle, and so on.

Exterior GeoWaffles are negative whereas interior GeoWaffles are positive, and the GMAP procedure relies on this distinction to shade (and label) all GeoWaffle maps that are generated.

	x	y	wafflenum	distance	STATEFIPS	COUNTYFIPS
626	-117.4	32.7	126	-0.1	.	.
627	-117.4	32.8	126	-0.1	.	.
628	-117.5	32.8	126	-0.1	.	.
629	-117.5	32.7	126	-0.1	.	.
630	-117.4	32.7	126	-0.1	.	.
631	-117.2	32.7	127	0.1	6	73
632	-117.2	32.8	127	0.1	6	73
633	-117.3	32.8	127	0.1	6	73
634	-117.3	32.7	127	0.1	6	73
635	-117.2	32.7	127	0.1	6	73
636	-117.1	32.7	128	0.1	6	73
637	-117.1	32.8	128	0.1	6	73
638	-117.2	32.8	128	0.1	6	73
639	-117.2	32.7	128	0.1	6	73
640	-117.1	32.7	128	0.1	6	73

Table 2. Subset of Three GeoWaffles in the GeoWaffle Data Set

Exterior GeoWaffles can also be differentiated because the identification variables (i.e., in this example, Statefips and Countyfips) will be missing, whereas interior GeoWaffles will always contain values for each identification variable.

The second data set produced by the MAKE_GEOWAFFLES macro is the hash data set—WAFFLES.CA_counties_hsh_1000 in this example. Note that the GeoWaffles and hash data sets are identically named, except for the “WAF” and “HSH” that they respectively contain. Finally, the size of GeoWaffles is appended to each filename. Thus, a GeoWaffle or hash data set filename that ends in “1” represents a file in which only one iteration of MAKE_GEOWAFFLES was run (e.g., as shown in Figure 9), whereas a filename that ends in “1000” represents a file in which four (i.e., the maximum number) iterations of MAKE_GEOWAFFLES was run, thus producing the tiniest of GeoWaffles.

Table 3 demonstrates the hash data set (WAFFLES.CA_counties_hsh_1000). Note that each hash entry corresponds to one GeoWaffle, thus the GeoWaffles data set will have five times the number of observations as the hash data set. The X (longitude) and Y (latitude) values for each hash object correspond to the lower-right corner of each GeoWaffle. Also note that all distances within the hash data set are positive, as there is no need to differentiate between interior and exterior GeoWaffles, as the hash data set is used for a hash object lookup rather than mapping and data visualization.

	x	y	distance	STATEFIPS	COUNTYFIPS
126	-117.4	32.7	0.1	.	.
127	-117.2	32.7	0.1	6	73
128	-117.1	32.7	0.1	6	73

Table 3. Subset of Three GeoWaffles in the Hash Data Set

As demonstrated, MAKE_GEOWAFFLES is invoked through a straightforward macro call. The macro headers follow:

```
%macro make_geowaffles(maplib= /* sas libname where map is located */,
   mapname= /* sas data set name that contains map */,
   wafflelib= /* sas libname where GeoWaffle and hash object are created */,
   idlist= /* space-delimited list of one or more map ID variables */,
   iterations=3 /* # of waffle sizes, an integer between 1 and 4 */);
```

Note again that the IDLIST arguments must be specified in order if the map data set contains more than one identification variable. For example, “Statefips Countyfips” is appropriate, because county polygons are subsumed within the superordinate state polygons; however, “Countyfips Statefips” would fail.

One of the most critical decisions in GeoWaffle construction is the number of iterations to run, as parameterized by the ITERATIONS parameter. Generating a four-iteration GeoWaffle (with a one 1/1000th of a degree rectangular side) does take substantially longer than a three-iteration GeoWaffle. For example, given a map of the state of California, three- and four-iteration GeoWaffle data sets would have 660,415 GeoWaffles and 1,587,490 GeoWaffles, respectively. However, as GeoWaffles are only created once but can be used infinitely thereafter, a best practice is to expend the extra time to create four-iteration hash objects that will execute far faster than two- or three-iteration hash objects.

For example, the following code creates GeoWaffles with only three iterations, after which the California earthquake data are run through the EAT_GEOWAFFLES for point-in-polygon determination:

```
%make_geowaffles(maplib=mymaps, mapname=CA_counties, wafflelib=waffles,
   idlist=statefips countyfips, iterations=3);

%eat_geowaffles(pointslib=work, pointsname=quakes_CA_NV_bigdata,
   wafflelib=waffles, wafflename=CA_counties_waf_100,
   maplib=mymaps, mapname=CA_counties,
   idlist=statefips countyfips, iterations=3,
   xvar=x, yvar=y,
   outfile=county_quakes_CA_GeoWaffles_100);
```

The three-iteration EAT_GEOWAFFLES now takes 59.4 seconds to complete (not shown), which is still only 20.0 percent of the 296.5 seconds that the GINSIDE procedure took to produce the same results, yet is more than five times slower than the 11.6 seconds that the four-iteration EAT_GEOWAFFLES macro had taken. This decrease in speed (and efficiency) is the result of more observations (N=1,846,300) requiring GINSIDE on the three-iteration run—because far fewer points are captured in GeoWaffles, and thus they must be analyzed instead with GINSIDE.

EATING GEOWAFFLES AND HASH

Whereas the MAKE_GEOWAFFLES macro creates the GeoWaffles and hash data sets, the EAT_GEOWAFFLES macro utilizes the hash object to determine point-in-polygon membership—identical results to the GINSIDE procedure, but just 25 times faster! The EAT_GEOWAFFLES macro contains the following header:

```
%macro eat_geowaffles(pointslib= /* sas libname where map is located */,
   pointsname= /* sas data set name that contains map */,
   wafflelib= /* sas libname where GeoWaffle and hash are located */,
   wafflename= /* sas filename for GeoWaffle (from which hash is derive) */,
   maplib= /* sas libname for shapefile/map data set used as backup */,
   mapname= /* sas data set name for shapefile/map used as backup */,
   idlist= /* space-delimited list of one or more map ID variables */,
   iterations=3 /* # of waffle sizes, an integer between 1 and 4 */,
   xvar=X /* longitude variable, which is changed to the default X */,
   yvar=Y /* latitude variable, which is changed to the default Y */,
   outfile= /* resultant point-in-polygon data set that is produced */);
```

EAT_GEOWAFFLES parameters include the following:

- POINTSLIB – This represents the SAS library in which the points data set is located that contains geospatial points (represented as latitude/longitude coordinates).
- POINTSNAME – This represents the name of the points data set, contained within the POINTSLIB SAS library.
- WAFFLELIB – This represents the SAS library in which the GeoWaffle and hash data sets will be saved; the “Waffles” library name is utilized throughout this text.

- WAFFLENAME – This represents the name of the SAS GeoWaffle data set against which the points data set is being evaluated. *In truth, EAT_GEOWAFFLES relies not on the GeoWaffle data set but rather on its sibling and similarly named hash object; however, to facilitate parameter consistency between the MAKE_GEOWAFFLES and EAT_GEOWAFFLES macros, WAFFLELIB and WAFFLENAME (parameters) are utilized in both macro calls, and the hash filename is automatically substituted inside the macro.*
- MAPLIB – This represents the SAS library in which the map data is maintained, and is typically a permanent SAS library as opposed to the WORK library. For consistency, this must be the same map data set from which the GeoWaffle data set and hash object were produced. This consistency is required because all points not processed through the specified GeoWaffle hash object will be subsequently processed through the GINSIDE procedure, which relies on the MAPLIB.MAPNAME library and map data set.
- MAPNAME – This represents the SAS map data set from which the GeoWaffles data set and hash object were produced.
- IDLIST – This ordered, space-delimited list of identification variables must correspond both to the WAFFLENAME and MAPNAME data sets.
- ITERATIONS – The number of iterations can vary between 1 and 4, with more iterations representing successively more refined GeoWaffles having more observations and smaller rectangles. Note that to utilize ITERATIONS=4, you must specify a GeoWaffle filename that ends in "1000"—representing that the GeoWaffle rectangles have a side length of 1/1000th of one degree latitude/longitude. On the other hand, ITERATIONS=3 can be run on either a GeoWaffle filename that ends in "1000" or a filename that ends in "100." Thus, if you have not first used MAKE_GEOWAFFLES with the ITERATIONS=4 parameter to create a "1000" GeoWaffle filename, you will not subsequently be able to invoke EAT_GEOWAFFLES with ITERATIONS=4, because a GeoWaffle data set with sufficient precision has not yet been created.
- XVAR – If the data set specified in the POINTSNAME parameter contains a longitude variable not named X, the name of the longitude variable should be entered here.
- YVAR – If the data set specified in the POINTSNAME parameter contains a latitude variable not named Y, the name of the latitude variable should be entered here.
- OUTFILE – This represents the SAS data set (in LIBNAME.Dataset format) produced by the EAT_GEOWAFFLES macro.

After EAT_GEOWAFFLES has read the hash data set into memory, it performs a complex hash object lookup using the X coordinate, Y coordinate, and Distance variables as keys, represented in the following DATA step (taken from the full EAT_GEOWAFFLES macro, which is also found in Appendix A):

```
data &outfile (drop=&drop) nothashed (drop=&drop &idlist);
   length x y distance 8 &length;
   if _n_=1 then do;
      declare hash h(dataset: "&wafflelib..&hash");
      rc=h.definekey('x', 'y', 'distance');
      rc=h.definedata(&idcommaquotes);
      rc=h.definedone();
      call missing(x, y, distance, &idcomma);
   end;
   set points;
   distance=1;
   rc1=h.find(key: x_1, key: y_1, key: distance);
   if rc1=0 then output &outfile;
   %if %eval(&iterations >= 2) %then %do;
   else do;
      distance =.1;
      rc2=h.find(key: x_10, key: y_10, key: distance);
      if rc2=0 then output &outfile;
      %if %eval(&iterations >= 3) %then %do;
```

```
      else do;
         distance=.01;
         rc3=h.find(key: x_100, key: y_100, key: distance);
         if rc3=0 then output &outfile;
         %if %eval(&iterations = 4) %then %do;
         else do;
            distance=.001;
            rc4=h.find(key: x_1000, key: y_1000, key: distance);
            if rc4=0 then output &outfile;
            else output nothashed;
            end;
         end;
      end;
            %end;
         %else %do;
         else output nothashed;
         end;
      end;
            %end;
         %end;
      %else %do;
      else output nothashed;
      end;
         %end;
      %end;
   %else %do;
   else output nothashed;
      %end;
run;
```

Consider an X,Y point denoted as (-117.2122, 32.7210) that might be found in the California earthquake points data set. EAT_GEOWAFFLES first transforms these coordinates through truncation—*not* rounding—to a whole number, so the coordinates (now saved as X_1 and Y_1) become (-117, 32). The first hash lookup attempts to find these whole numbers in the hash object; if it succeeds, the attributes (i.e., identification variables, Statefips and Countyfips) are added to the observation in the points data set, and if it fails, a second hash lookup is attempted.

When the second hash lookup is attempted, EAT_GEOWAFFLES transforms the original coordinates (again through truncation) to one tenth of a decimal. Thus, the transformed variables (now saved as X_10 and Y_10) become (-117.2, 32.7). The second observation in Table 3 demonstrates that these coordinates (-117.2, 32.7) correspond to the Statefips value 6 (i.e., California) and Countyfips value 73 (i.e., San Diego County), so the hash lookup has succeeded on its second attempt and has determined that the point lies somewhere inside the $1/10^{th}$ of one degree rectangle whose lower-right corner point is -117.2, 32.7. After this success, the DATA step continues to evaluate the next observation; had the second hash lookup also failed, EAT_GEOWAFFLES would have continued to a third hash lookup performed at the $1/100^{th}$ of one degree level. Thus, the ITERATIONS parameter in the EAT_GEOWAFFLES macro controls the number of hash object lookups that are attempted, with each successive hash lookup occurring at a level of precision ten times greater than the previous lookup, as the GeoWaffle rectangles become smaller and smaller.

The Distance key is pivotal in the hash lookup because it detangles unique X,Y coordinates that would otherwise behave identically. For example, the coordinates (-117.04, 32.07), when rounded to $1/10^{th}$ of a degree in the second EAT_GEOWAFFLES hash lookup, would become (-117.0, 32.0)—and these coordinates, while possessing a higher degree of precision, would be mathematically identical to a less precise hash lookup on the first iteration that had truncated those coordinates to (-117, 32). For this reason, DISTANCE must be included as a hash key because it distinguishes the size of the GeoWaffle in which the point lies. Thus, because four different sizes of rectangles will exist within a single GeoWaffles data set that was generated with four iterations (i.e., ITERATIONS=4 when MAKE_GEOWAFFLES is invoked), DISTANCE is used to identify which size of rectangle has been matched during a successful hash lookup.

VALIDATING GEOWAFFLE ACCURACY

Validation of any new methodology (such as GeoWaffles) for accuracy is critical to ensure it generates identical output to the gold standard—in this case, the GINSIDE procedure. Thus, at each iteration, GeoWaffles output must be tested to ensure its point-in-polygon analysis matches that of GINSIDE.

First, the earthquake CSV file is ingested to recreate the Quakes_CA_NV data set, having 78,859 observations:

```
%let quake_loc = D:\sas\geowaffles\USGS_quakes\; * USER MUST CHANGE LOCATION *;
proc import datafile="&quake_loc.quakes_2020_CA_NV.csv"
      out=quakes_raw dbms=csv replace;
   getnames=yes;
   guessingrows=100000;
run;
data quakes_CA_NV (keep=x y time depth mag);
   set quakes_raw (rename=(latitude=y longitude=x));
   where type='earthquake';
run;
```

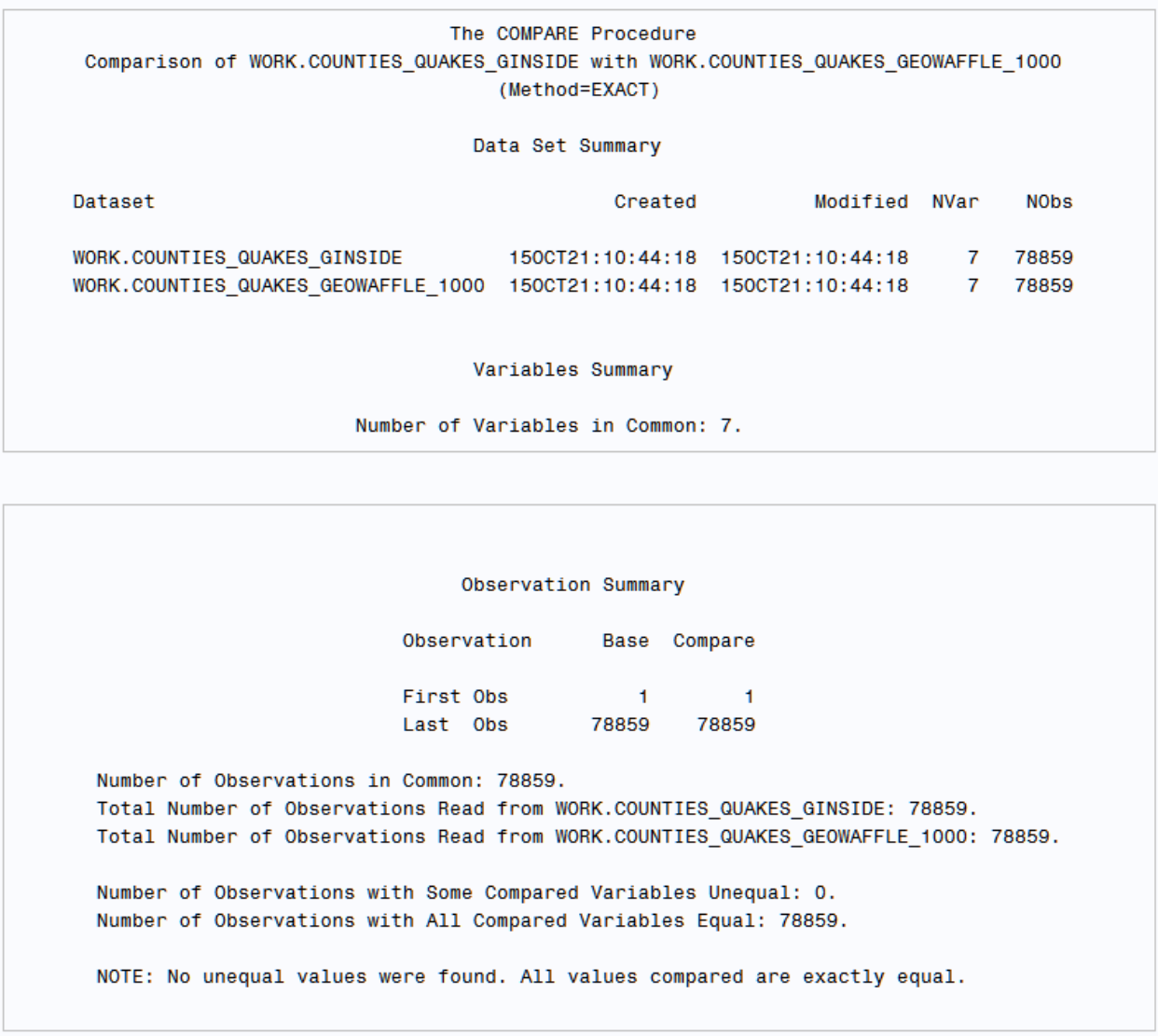
Next, the GINSIDE procedure produces the benchmark data set (Counties_quakes_GINSIDE) against which all further output data sets will be evaluated:

```
option fullstimer;
proc ginside data=quakes_CA_NV map=mymaps.CA_counties (where=(segment=1))
      out=counties_quakes_GINSIDE includeborder dropmapvars;
   id statefips countyfips;
run;
```

The following code both creates and utilizes the GeoWaffles (at level ITERATION=4), after which both the GINSIDE and GeoWaffle output data sets are sorted and compared with the COMPARE procedure:

```
%make_geowaffles(maplib=mymaps, mapname=CA_counties, wafflelib=waffles,
   idlist=statefips countyfips, iterations=4);

%eat_geowaffles(pointslib=work, pointsname=quakes_CA_NV,
   wafflelib=waffles, wafflename=CA_counties_waf_1000,
   maplib=mymaps, mapname=CA_counties,
   idlist=statefips countyfips, iterations=4,
   xvar=x, yvar=y,
   outfile=counties_quakes_GeoWaffle_1000);

proc sort data=counties_quakes_GINSIDE (drop=_onborder_);
   by x y time depth mag statefips countyfips;
run;

proc sort data=counties_quakes_GeoWaffle_1000 (drop=distance);
   by x y time depth mag statefips countyfips;
run;

ods html;
proc compare base=counties_quakes_GINSIDE
   compare=counties_quakes_GeoWaffle_1000;
run;
```

The output of the COMPARE procedure, shown in Figure 13, demonstrates that the data sets are identical. In fact, when run successively at ITERATIONS=1, ITERATIONS=2, ITERATIONS=3, and ITERATIONS=4, the GeoWaffle macros masterfully perform to produce results identical to the GINSIDE procedure benchmark output data set.



```
                                The COMPARE Procedure
     Comparison of WORK.COUNTIES_QUAKES_GINSIDE with WORK.COUNTIES_QUAKES_GEOWAFFLE_1000
                                     (Method=EXACT)

                                    Data Set Summary

Dataset                                        Created          Modified  NVar    NObs

WORK.COUNTIES_QUAKES_GINSIDE          15OCT21:10:44:18  15OCT21:10:44:18     7   78859
WORK.COUNTIES_QUAKES_GEOWAFFLE_1000   15OCT21:10:44:18  15OCT21:10:44:18     7   78859


                                    Variables Summary

                            Number of Variables in Common: 7.


                                  Observation Summary

                             Observation      Base  Compare

                             First Obs           1        1
                             Last  Obs       78859    78859

  Number of Observations in Common: 78859.
  Total Number of Observations Read from WORK.COUNTIES_QUAKES_GINSIDE: 78859.
  Total Number of Observations Read from WORK.COUNTIES_QUAKES_GEOWAFFLE_1000: 78859.

  Number of Observations with Some Compared Variables Unequal: 0.
  Number of Observations with All Compared Variables Equal: 78859.

  NOTE: No unequal values were found. All values compared are exactly equal.
```

Figure 13. COMPARE Demonstrates Equivalence of GINSIDE Results to GeoWaffles Results

Further testing could be done to illustrate that GeoWaffles generate results identical to GINSIDE—for example, by testing more data sets across more diverse shapefiles and by varying the &PRECISION macro variable that is assigned within the MAKE_GEOWAFFLES macro and which controls the spacing of the test points that are generated for point-in-polygon testing. However, these perfect validation results coupled with the lightning speed performance of GeoWaffles seems to indicate a clear victor in this battle of the point-in-polygon pugilists.

CONCLUSION

Geospatial point-in-polygon analysis is hard—in any language—with complex algorithms that can lead to poor efficiency and performance. But where big data must be analyzed, analyzed recurrently, and analyzed with respect to static shapefiles, memoization should be the methodology of choice to deliver in-memory point-in-polygon analytics. Where the GINSIDE procedure has chosen to work *harder*, GeoWaffles instead works *smarter*, by relying on the SAS hash object to perform more than 99 percent of the load previously carried by the GINSIDE procedure. With this advancement, natural disasters and other catastrophic events can be quickly cataloged—even when they surpass the millions and billions of records.

REFERENCES

Hughes, T. M. (2013). Winning the War on Terror with Waffles: Maximizing GINSIDE Efficiency for Blue Force Tracking Big Data. *Southeast SAS User's Group (SESUG).* Retrieved from https://analytics.ncsu.edu/sesug/2013/PO-18.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

APPENDIX A. GEOWAFFLES.SAS PROGRAM FILE

```
* required to display GMAP output;
proc format;
   value waflsize
      1, -1 = '1 deg lat/long'
      .1, -.1 = '0.1 deg lat/long'
      .01, -.01 = '0.01 deg lat/long';
run;


%macro make_geowaffles(maplib= /* sas libname where map is located */,
   mapname= /* sas data set name that contains map */,
   wafflelib= /* sas libname where GeoWaffle and hash object are created */,
   idlist= /* space-delimited list of one or more map ID variables */,
   iterations=3 /* # of waffle sizes, an integer between 1 and 4 */);
%local wafflenum precision distance numids varlist i j k boxes
   length idlist_old boxsize mapnobs;
%let wafflenum=0; * total GeoWaffles created during all iterations of this macro;
%let boxes=; * total number of boxes tested in a specific iteration;
%let precision=.0001; * distance in decimal degrees between each point tested;
%let distance = 1; * starting length in degrees lat/long of each side of the box;
%let numids=%sysfunc(countw(&idlist));
* set BOXSIZE, the boxes created per degree lat/long, to append to filename;
%let iterations=%sysfunc(round(&iterations));
%let mapnobs=0; * number of observations in map data set used by GMAP;
%if %eval(&iterations<0) %then %let iterations=1;
%else %if %eval(&iterations>4) %then %let iterations=4;
%let boxsize=%sysevalf(10**(&iterations-1));

* dynamically generate IDLIST to be used in RETAIN statement;
%let idlist_old=;
%do i=1 %to &numids;
   %let idlist_old=&idlist_old old_%scan(&idlist,&i,,S);
   %end;

* dynamically generate LENGTH statement;
proc sql noprint;
   select strip(upcase(name)) || ' ' || ifc(upcase(type)='CHAR','$','')
         || strip(put(length,best.)),
      'old_' || strip(upcase(name)) || ' ' || ifc(upcase(type)='CHAR','$','')
         || strip(put(length,best.))
      into :length separated by ' ', :length_old separated by ' '
         from dictionary.columns
            where upcase(libname)="%upcase(&maplib)" and
                  upcase(memname)="%upcase(&mapname)" and (
               %do i=1 %to &numids;
                  %if &i=1 %then %do;
                     (upcase(name) in("%upcase(%scan(&idlist,&i,,S)")))
                     %end;
                  %else %do;
                     or (upcase(name) in("%upcase(%scan(&idlist,&i,,S)")))
                     %end;
                  %end;
               );
quit;

* initialize permanent WAFFLES data set;
data &wafflelib..&mapname._waf_&boxsize;
   length x y wafflenum distance &length;
   if 0=1 then output;
run;
```

```
* initialize permanent HASH data set;
   data &wafflelib..&mapname._hsh_&boxsize;
      length x y distance 8 &length;
      if 0=1 then output;
   run;

* GINSIDE can produce unpredictable results when SEGMENT > 1, so data are subset;
proc sql noprint;
   select upcase(name) into :varlist separated by ' '
      from dictionary.columns
         where upcase(libname)="%upcase(&maplib)" and
            upcase(memname)="%upcase(&mapname)";
quit;
data map_seg;
   set &maplib..&mapname;
   %do j=1 %to %sysfunc(countw(&varlist));
      %if %upcase(%scan(&varlist,&j,,S))=SEGMENT %then %do;
         where segment=1;
         %end;
      %end;
run;

* determine boundaries and initialize data set used for creating test boxes;
proc sql noprint;
   select floor(min(y)), ceil(max(y)), floor(min(x)), ceil(max(x))
      into :minlat, :maxlat, :minlong, :maxlong
   from map_seg;
quit;
data crumbs;
   length minlat maxlat minlong maxlong distance 8;
   minlat=&minlat;
   maxlat=&maxlat;
   minlong=&minlong;
   maxlong=&maxlong;
   distance=&distance; * represents a box 1 deg lat X 1 deg long;
run;

* create test boxes;
%do i=1 %to &iterations;
   data testpoints (drop=minlat maxlat minlong maxlong ystart xstart cnt);
      set crumbs end=eof;
      length wafflenumtest startpoint 8;
      if _n_=1 then wafflenumtest=0; * initialize first waffle;
      retain wafflenumtest;
      do ystart=minlat to (maxlat - distance) by distance;
         ystart=round(ystart,&precision);
         do xstart=minlong to (maxlong - distance) by distance;
            xstart=round(xstart,&precision);
            wafflenumtest=wafflenumtest + 1;
            startpoint=.; *initialize for each waffle;
            cnt=0;
            do y=(ystart + &precision) to (ystart + distance) by &precision;
               * draws line up;
               x=round((xstart + distance),&precision);
               y=round(y,&precision); * ROUND required due to binary math;
               output;
               cnt+1;
               end;
            do x=(xstart + distance - &precision) to xstart by -&precision;
               * draws line left;
               x=round(x,&precision);
               y=ystart + distance;
               output;
```

```
            cnt+1;
            end;
         do y=(ystart + distance - &precision) to ystart by -&precision;
            * draws line down;
            x=xstart;
            y=round(y,&precision);
            output;
            cnt+1;
            end;
         do x=(xstart + &precision) to (xstart + distance) by &precision;
            * draws line right;
            x=round(x,&precision);
            y=ystart;
            cnt+1;
            if cnt=4 * round(distance / &precision) then startpoint=1;
            output;
            end;
         end;
      end;
   if eof then call symputx('boxes',wafflenumtest);
run;

* crumbs has been processed, thus its data can be deleted;
data crumbs;
   set crumbs;
   if 0=1;
run;

* GINSIDE must be run in batches due to potential memory constraints;
%do j=1 %to &boxes;
   %put PROCESSING &J OF &BOXES TEST BOXES;
   %let jump=%sysevalf(10**(&i+1)); * number by which to increment boxes;
   * determine if box is 1) inside, 2) outside, or 3) crosses polygon;
   proc ginside data=testpoints
         (where=(&j <= wafflenumtest < %sysevalf(&j+&jump)))
         map=map_seg out=rawwaffles includeborder dropmapvars;
      id &idlist;
   run;
   data tempwaffles (keep=x y wafflenum distance &idlist)
         tempwaffles_n_hash (keep=x y distance &idlist)
         tempcrumbs (keep=minlat maxlat minlong maxlong distance);
      set rawwaffles(rename=(y=lat x=long)) end=eof;
      by wafflenumtest;
      length crumb wafflenum 8 &length_old;
      retain crumb wafflenum &idlist_old;
      if _n_=1 then do;
         wafflenum=&wafflenum;
         end;
      if first.wafflenumtest then do;
         crumb=0; * initialized to NOT a crumb;
         end;
      * only if all box points are inside polygon will GeoWaffle be created;
      else if (
         %do k=1 %to &numids;
            %if &k=1 %then %do;
               %scan(&idlist,&k,,S) ^= %scan(&idlist_old,&k,,S)
               %end;
            %else %do;
               or %scan(&idlist,&k,,S) ^= %scan(&idlist_old,&k,,S)
               %end;
            %end;
            ) then crumb=1;
      if last.wafflenumtest then do;
```

```
         if crumb=0 then do; * create a GeoWaffle for graphing;
            wafflenum+1;
            distance=&distance;
            y=round(lat,&distance); x=round(long,&distance);
            output tempwaffles;
            output tempwaffles_n_hash; * create hash object;
            y=round(lat + distance, &distance); output tempwaffles;
            x=round(long - distance, &distance); output tempwaffles;
            y=round(lat,&distance); output tempwaffles;
            x=round(long,&distance); output tempwaffles;
            end;
         else if crumb=1 then do; * create crumbs for subsequent processing;
            distance=&distance;
            minlat=round(lat,&distance);
            maxlat=round(lat + distance,&distance);
            minlong=round(long - distance,&distance);
            maxlong=round(long,&distance);
            distance=distance / 10;
            output tempcrumbs;
            end;
         call symputx('wafflenum',wafflenum);
         end;
      %do k=1 %to &numids;
         %scan(&idlist_old,&k,,S)=%scan(&idlist,&k,,S);
         %end;
   run;
   *append waffles, hash, and crumbs to their primary data sets;
   proc append base=&wafflelib..&mapname._waf_&boxsize data=tempwaffles force;
   run;
   proc append base=&wafflelib..&mapname._hsh_&boxsize
      data=tempwaffles_n_hash force;
   run;
   proc append base=crumbs data=tempcrumbs force;
   run;
   %let j=%sysevalf(&j + &jump - 1);
   %end;

* exterior GeoWaffles have negative DISTANCE to differentiate;
data &wafflelib..&mapname._waf_&boxsize;
   set &wafflelib..&mapname._waf_&boxsize;
      if
      %do k=1 %to &numids;
         %if %eval(&k >= 2) %then %do;
            or
            %end;
         missing(%scan(&idlist,&k,,S))
         %end;
      then distance=abs(distance) * (-1);
* OPTIONALLY generate GeoWaffles map;
proc sql noprint;
   select count(*) into :mapnobs
      from &wafflelib..&mapname._waf_&boxsize;
quit;
%if %sysevalf(&mapnobs > 0) and %eval(&iteration < 4) %then %do;
   pattern1 value=solid color='cxBF3A4E'; * dark red;
   pattern2 value=solid color='cxD2818D';
   pattern3 value=solid color='cxEFC5CB';
   pattern4 value=solid color='cxC1D9EE'; * dark blue;
   pattern5 value=solid color='cx629DCD';
   pattern6 value=solid color='cx1365A9';
   goptions xpixels=800 xmax=10 in ymax=10 in;
   ods html;
   proc gmap map=&wafflelib..&mapname._waf_&boxsize
```

```
            data=&wafflelib..&mapname._waf_&boxsize;
         id wafflenum;
         label distance='GeoWaffle Girth';
         format distance waflsize.;
         choro distance / discrete midpoints=-1 -.1 -.01 .01 .1 1;
      run;
      quit;
      ods html close;
      %end;
   %let distance=%sysevalf(&distance / 10);
   %end;
%mend;


%macro eat_geowaffles(pointslib= /* sas libname where map is located */,
   pointsname= /* sas data set name that contains map */,
   wafflelib= /* sas libname where GeoWaffle and hash are located */,
   wafflename= /* sas filename for GeoWaffle (from which hash is derive) */,
   maplib= /* sas libname for shapefile/map data set used as backup */,
   mapname= /* sas data set name for shapefile/map used as backup */,
   idlist= /* space-delimited list of one or more map ID variables */,
   iterations=3 /* # of waffle sizes, an integer between 1 and 4 */,
   xvar=X /* longitude variable, which is changed to the default X */,
   yvar=Y /* latitude variable, which is changed to the default Y */,
   outfile= /* resultant point-in-polygon data set that is produced */);
%local numids renamexy drop length boxsize len hash idcomma idcommaquotes;
%let numids=%sysfunc(countw(&idlist));
* modify iterations if they are invalid;
%let iterations=%sysfunc(round(&iterations));
%if %eval(&iterations<0) %then %let iterations=1;
%else %if %eval(&iterations>4) %then %let iterations=4;

* dynamically generate RENAME statement;
%if %upcase(&xvar)=X and %upcase(&yvar)=Y %then %let renamexy=;
%else %if %upcase(&xvar)=X and %upcase(&yvar)^=Y %then %let
   renamexy=(rename=(&yvar=y));
%else %if %upcase(&xvar)^=X and %upcase(&yvar)=Y %then %let
   renamexy=(rename=(&xvar=x));
%else %if %upcase(&xvar)^=X and %upcase(&yvar)^=Y %then %let
   renamexy=(rename=(&yvar=y &xvar=x));

* dynamically generate DROP statement;
%let drop=rc;
%do i=1 %to &iterations;
   %let drop=&drop rc&i x_%sysevalf(10**(&i-1)) y_%sysevalf(10**(&i-1));
   %end;

* dynamically generate hash filename;
%let boxsize=%sysevalf(10**(&iterations-1)); * used in filename;
%let len=%eval(%length(&wafflename) - (%length(%scan(&wafflename,-1,_))
   + %length(%scan(&wafflename,-2,_)) + 1));
%let hash=%substr(&wafflename,1,&len)hsh_&boxsize;

* dynamically generate LENGTH statement;
proc sql noprint;
   select strip(upcase(name)) || ' ' || ifc(upcase(type)='CHAR','$','')
         || strip(put(length,best.)),
         'old_' || strip(upcase(name)) || ' ' || ifc(upcase(type)='CHAR','$','')
         || strip(put(length,best.))
      into :length separated by ' ', :length_old separated by ' '
         from dictionary.columns
            where upcase(libname)="%upcase(&hash)" and
               upcase(memname)="%upcase(&h)" and (
```

```
         %do i=1 %to &numids;
            %if &i=1 %then %do;
               (upcase(name) in("%upcase(%scan(&idlist,&i,,S)")))
               %end;
            %else %do;
               or (upcase(name) in("%upcase(%scan(&idlist,&i,,S)")))
               %end;
            %end;
         );
quit;

* dynamically generate comma-delimited list of IDLIST;
%let idcomma=;
%let idcommaquotes=;
%do i=1 %to &numids;
   %let idcomma=&idcomma %scan(&idlist,&i,,S);
   %let idcommaquotes=&idcommaquotes "%scan(&idlist,&i,,S)";
   %if %eval(&i^=&numids) %then %do;
      %let idcomma=&idcomma,;
      %let idcommaquotes=&idcommaquotes,;
      %end;
   %end;

* process the points file to add rounded variables;
data points;
   set &pointslib..&pointsname &renamexy;
   * create values rounded to whole degree;
   length x_1 y_1 8;
   format x_1 y_1 12.0;
   x_1=round(ceil(x));
   y_1=round(floor(y));
   %if %eval(&iterations >= 2) %then %do;
      * create values rounded to one-tenth of a degree;
      length x_10 y_10 8;
      format x_10 y_10 12.1;
      x_10=round(input(substr(put(x,12.5),1,find(put(x,12.5),'.')+1),8.1),.1);
      y_10=round(input(substr(put(y,12.5),1,find(put(y,12.5),'.')+1),8.1),.1);
      %end;
   %if %eval(&iterations >= 3) %then %do;
      * create values rounded to one one-hundredth of a degree;
      length x_100 y_100 8;
      format x_100 y_100 12.2;
      x_100=round(input(substr(put(x,12.5),1,find(put(x,12.5),'.')+2),12.2),.01);
      y_100=round(input(substr(put(y,12.5),1,find(put(y,12.5),'.')+2),12.2),.01);
      %end;
   %if %eval(&iterations = 4) %then %do;
      * create values rounded to one one-hundredth of a degree;
      length x_1000 y_1000 8;
      format x_1000 y_1000 12.3;
      x_1000=round(input(substr(put(x,12.5),1,find(put(x,12.5),'.')+3),12.3),.001);
      y_1000=round(input(substr(put(y,12.5),1,find(put(y,12.5),'.')+3),12.3),.001);
      %end;
run;
* hash the points;
data &outfile (drop=&drop) nothashed (drop=&drop &idlist);
   length x y distance 8 &length;
   if _n_=1 then do;
      declare hash h(dataset: "&wafflelib..&hash");
      rc=h.definekey('x', 'y', 'distance');
      rc=h.definedata(&idcommaquotes);
      rc=h.definedone();
      call missing(x, y, distance, &idcomma);
   end;
```

```
   set points;
   distance=1;
   rc1=h.find(key: x_1, key: y_1, key: distance);
   if rc1=0 then output &outfile;
   %if %eval(&iterations >= 2) %then %do;
   else do;
      distance =.1;
      rc2=h.find(key: x_10, key: y_10, key: distance);
      if rc2=0 then output &outfile;
      %if %eval(&iterations >= 3) %then %do;
      else do;
         distance=.01;
         rc3=h.find(key: x_100, key: y_100, key: distance);
         if rc3=0 then output &outfile;
         %if %eval(&iterations = 4) %then %do;
         else do;
            distance=.001;
            rc4=h.find(key: x_1000, key: y_1000, key: distance);
            if rc4=0 then output &outfile;
            else output nothashed;
            end;
         end;
      end;
            %end;
         %else %do;
         else output nothashed;
         end;
      end;
            %end;
         %end;
      %else %do;
      else output nothashed;
      end;
         %end;
      %end;
   %else %do;
   else output nothashed;
      %end;
run;
* ensure that the map data set contains only SEGMENT=1;
proc sql noprint;
   select upcase(name) into :varlist separated by ' '
      from dictionary.columns
         where upcase(libname)="%upcase(&maplib)"
            and upcase(memname)="%upcase(&mapname)";
quit;
data map_seg;
   set &maplib..&mapname;
   %do j=1 %to %sysfunc(countw(&varlist,S));
      %if %upcase(%scan(&varlist,&j,,S))=SEGMENT %then %do;
         where segment=1;
         %end;
      %end;
run;
* run GINSIDE on the few points that could not be matched through hashing;
proc ginside data=nothashed map=map_seg out=gout (drop=_onborder_)
      includeborder dropmapvars;
   id &idlist;
run;
* merge the points that are hashed with the points attributed through GINSIDE;
proc append base=&outfile data=gout force;
run;
%mend;
```