Paper 050-2010

Driving Distances and Times Using SAS® and Google Maps

Mike Zdeb, University@Albany School of Public Health, Rensselaer, NY

ABSTRACT

SAS 9.2 contains new functions (ZIPCITYDISTANCE, GEODIST) that allow a user to compute geodesic distance (the shortest distance between two points on the surface of a sphere). Both functions use the Vincenty distance formula. Prior to SAS 9.2, a data step was used to compute such distances, commonly with the Haversine formula. Vincenty and Haversine distance estimates are straight line distances and there are occasions where that type of estimate is what you need. There are other occasions where what you want is not the straight line distance, but a driving distance. Given only one combination of locations, using Google Maps to get the driving distance and time is no problem. However, if you have a large number of location pairs, a FILENAME statement and the URL access method within SAS can be used to access Google Maps multiple times and extract both the driving distance and time each time the site is accessed.

INTRODUCTION

There are a number of instances where it is important to know the distance between two locations. Some examples include determining: average distance of potential customers from a new store location; the percentage of the population within a given distance from a specialty medical service; chronic disease history among the population living within varying distances from a pollution source. In each of these situations, calculation of distance is integral to solving a problem.

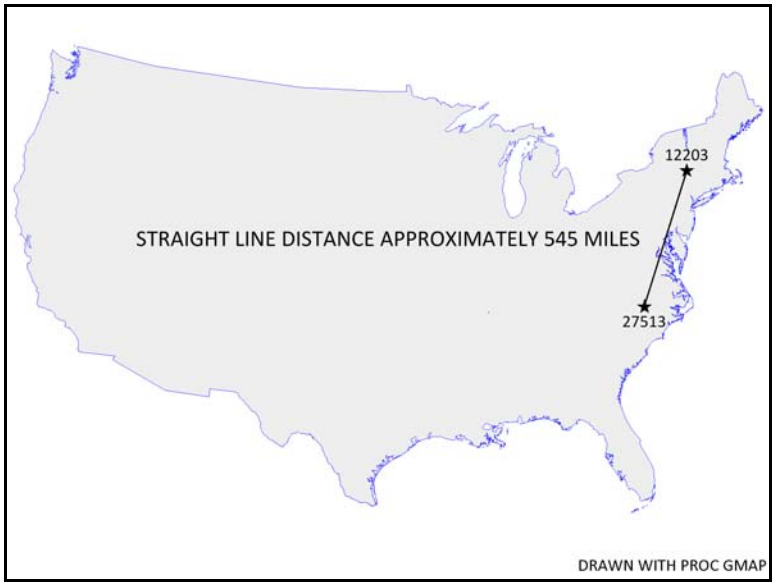
The information that one has on location can vary in precision from knowing only a 5-digit zip code to knowing the latitude and longitude (with various intermediate levels such as a zip+4 or a street address). Prior to SAS 9.2, a user had to write a data step with an equation that calculated distance. A commonly used equation is based on the Haversine formula that will calculate "... great-circle distances between two points on a sphere from their longitudes and latitudes ..."[1] . New functions in SAS 9.2 simplify distance calculation. The ZIPCITYDISTANCE function calculates the distance in miles between the centroids of two user-specified zips, while the GEODIST function calculates the distance in either miles or kilometers between two user-specified locations that are expressed in terms of latitude and longitude. ZIPCITYDISTANCE relies in information contained in the data set SASHELP.ZIPCODE[2]. Among the variables in that data set are the latitude and longitude of zip code centroids in degrees, with variable name Y and X respectively. Example 1 shows how to compute the distance between the centroid of two zips: 12203 (the author's home in Albany, NY); 27513 (SAS in Cary, NC).

```
* example 1 ... straight line distance calculation;

data _null_;

* ZIPCITYDISTANCE function;
d1 = zipcitydistance(12203,27513);

* find lat/long of center of 12203 (author's home, Albany, NY);
zip = 12203;
set sashelp.zipcode (rename=(x=long1 y=lat1))key=zip/unique;

* find lat/long of center of 27513 (SAS, Cary, NC);
zip = 27513;
set sashelp.zipcode (rename=(x=long2 y=lat2)) key=zip/unique;

file print;
put
"LATITUDE AND LONGITUDE OF TWO LOCATIONS (FROM SASHELP.ZIPCODE)" //
"ZIP 12203: LATITUDE " lat1 " LONGITUDE " long1 /
"ZIP 27513: LATITUDE " lat2 " LONGITUDE " long2 /;

* GEODIST function;
d2 = geodist(lat1, long1, lat2, long2, 'M');

*
convert latitude and longitude from degrees
to radians for use in Haversine formula
;
d_to_r = constant('pi')/180;
lat1  = lat1  * d_to_r; long1 = long1 * d_to_r;
lat2  = lat2  * d_to_r; long2 = long2 * d_to_r;

*
HAVERSINE formula (earth radius of 3949.99 miles
used to produce distance in miles)
;
d3 = 3949.99 * arcos(sin(lat1) * sin(lat2) + cos(lat1) *
               cos(lat2) * cos(long2 - long1));

put
"STRAIGHT LINE DISTANCE COMPARISON" //
"ZIPCITYDISTANCE : " d1 /
"GEODIST         : " d2 /
"HAVERSINE       : " d3;

stop;
run;
```

In example 1, the only arguments required in the ZIPCITYDISTANCE function are two zip codes. Both the GEODIST function and the Haversine formula require the latitude and longitude and those values are read from the data set SASHELP.ZIPCODE. That data set is indexed so it easy to extract values using a SET statement with a KEY= option. The values from SASHELP.ZIPCODE are inserted in the GEODIST function and the 'M' option is used to produce a result in miles (without that argument, the distance is calculated by default in kilometers). The Haversine formula requires that all values of latitude and longitude be expressed in radians, not degrees. Once the values are converted, they are inserted into the formula.

```
LATITUDE AND LONGITUDE OF TWO LOCATIONS (FROM SASHELP.ZIPCODE)

ZIP 12203: LATITUDE 42.691560  LONGITUDE -73.827840
ZIP 27513: LATITUDE 35.805410  LONGITUDE -78.797679

STRAIGHT LINE DISTANCE COMPARISON

ZIPCITYDISTANCE : 544.6
GEODIST         : 544.58876085
HAVERSINE       : 543.5986627
```

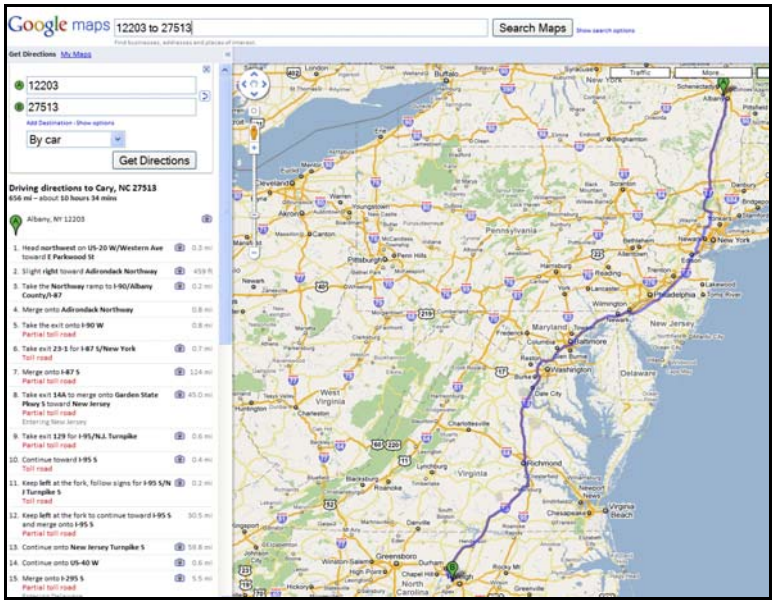
The results of example 1 are shown on the right. First, you can see the values that were found in SASHELP.ZIPCODE. Next, the three distances are shown. The ZIPCITYDISTANCE function produces a result with one decimal place, while the GEODIST function returns the same value, just with more decimal places. Notice in example 1 that the ZIPCITYDISTANCE function does the work of finding the latitude and longitude in SASHELP.ZIPCODE (you do not have to look up the values on your own with a SET statement).



The result of the Haversine formula is a bit different from the function results. Both of the new functions are based on the Vincenty formula[3], said to be more accurate than the Haversine formula. Regardless of the formula used, if you were planning to use the distance estimate for a drive from Albany to Cary, you can see in the map on the right that your route would be a bit difficult to maintain. This is a situation where the driving distance and perhaps the driving time would be more important to you than the straight line distance shown on the map.

DRIVING DISTANCE AND TIMES

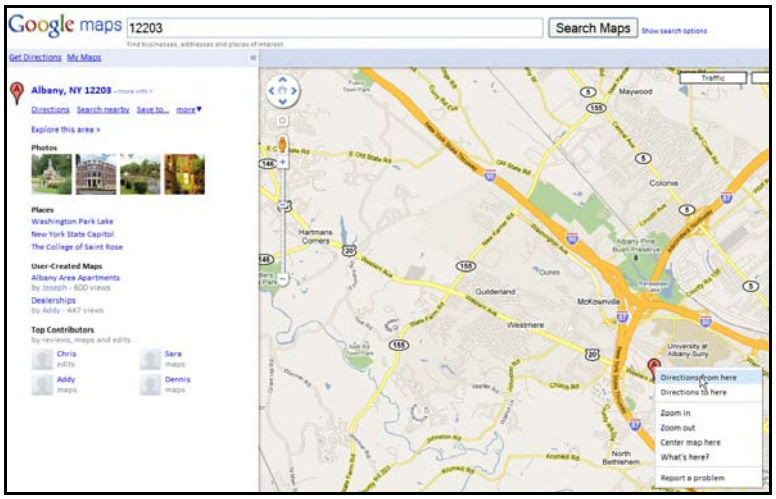
There is no driving distance function is SAS. Even with GIS software, finding driving distances is not a trivial task. However, it is easy to get both an estimate of driving distance and driving time with Google Maps. Once you have brought up the maps web site in a browser, you can enter the two zip codes separated by 'to' and the map on the right is produced. If you cannot read the tiny text in the Google-produced map, the estimate of driving distance is 656 miles, with a time of about 10 hours and 34 minutes (that distance being between zip centroids).

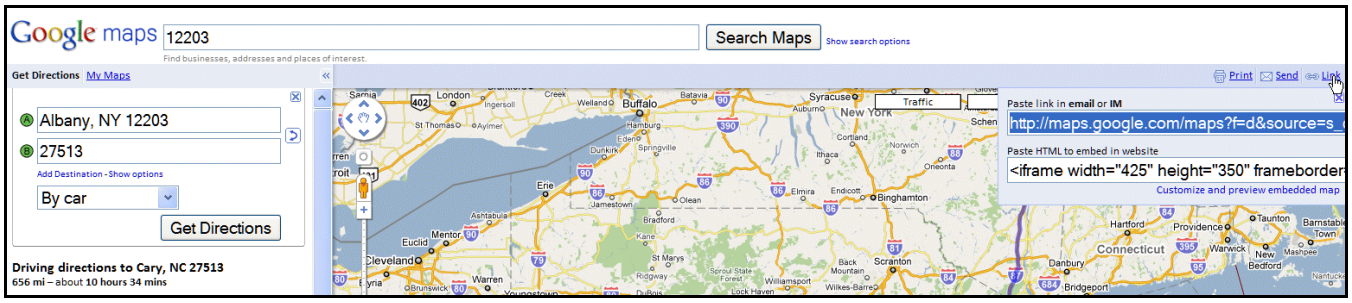


Another way to get the information about driving distance in Google Maps is to first enter just one zip code (12203) and then get a map as shown on the top of the next page. If you then click the right mouse button on the location of the zip, you are presented with a number of choices, among them being "from here". If you click on that selection, you can enter the destination zip of 27513, you will once again get the driving distance and driving time.

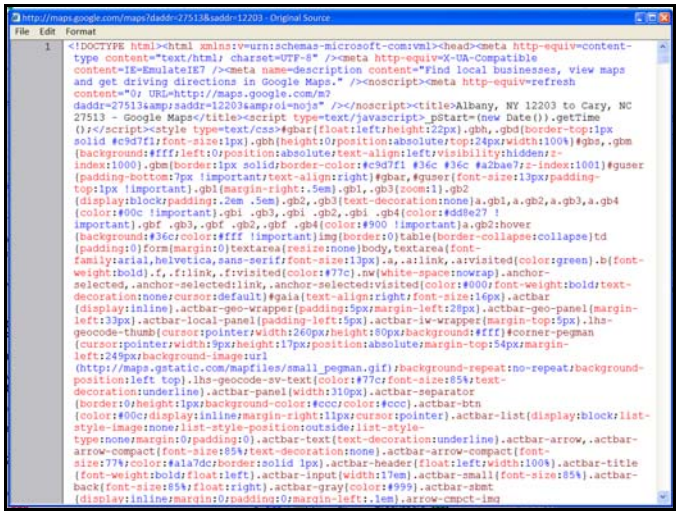
AUTOMATING THE TASKS

Looking up one combination of zip codes to get a driving distance is a trivial task in Google Maps. However, what if you had the location of a facility (be it a store, hospital, trauma center, etc.) and wanted to know the number of people who were within various drive times to that facility. The first step in that process might be to find the driving distance from the centroid of a large number of zip codes to the centroid of the zip where your facility is located. You could still do that one-by-one in Google Maps, but it would be much better if you could automate that process ... maybe by using SAS.





If you look in the upper-right corner of the driving route map, you will see that you can click on LINK and if you do, you will see a URL that allows you to reproduce map showing the driving rout between zips 12203 and 27513. That URL is very long, but buried within all the text of the URL are the values if the two zip codes. A bit of experimenting with that long URL shows that it can be reduced to the following ... http://maps.google.com/maps?daddr=27513&saddr=12203



Given that URL, Google Maps creates a map and you can look at the HTML that produces what you see on the screen. Selecting VIEW / SOURCE in Internet Explorer gives you the display shown on the right. The next task is to find the driving distance (and driving time) within all that text. Once that text is found, it should be possible to write some SAS code that finds that text for you.

Searching through all that HTML shows that the information you want to extract looks as follows ...

656 mi – about 10 hours 34 mins

and if you could strip away the HTML you can end up with the 656 miles and the 10 hours 34 mins seen on the left side of the web page produced by Google Maps. The SAS code solution to the manual tasks are ...

Manual Task	SAS Code Solution
use Google Maps with a URL that looks as follows ... http://maps.google.com/maps?daddr=27513&saddr=12203	use the URL access method in a FILENAME statement
find the portion of the HTML that contains the driving distance and driving time	read the HTML in a data step and use SAS functions to find and extract the desired information

The SAS code in example 2 creates a map (that you do not see) with Google Maps, then reads the underlying HTML and extracts the driving distance and driving time. Rather than entering the values of the two zip codes in the FILENAME statement, two macro variables are used in anticipation of modifying the code to process a large number of zip code combinations. There may be a few items in the SAS code that are new to you.

```
* example 2;

* enter two zip codes;
%let z1=12203;
%let z2=27513;

* no changes required below this line;

filename x url "http://maps.google.com/maps?daddr=&z2.%nrstr(&saddr)=&z1";

data drive;
retain zip1 &z1 zip2 &z2;
infile x lrecl=32000 pad;
input;
loc = find(_infile_,'dditd>');
if loc ne 0 then do;
  text = substr(_infile_,loc,50);
  text = scan(text,1,'&');
  distance = input(scan(text,-1,'>'),comma12.);
  loc = find(_infile_,'about');
  text = substr(_infile_,loc,50);
  text = scan(text,3,'<>');
* convert times to seconds;
  select;
* combine days and hours;
   when (find(text,'day') ne 0)  time = 86400*input(scan(text,1,' '),best.) +
                                        3600*input(scan(text,3,' '),best.);
* combine hours and minutes;
   when (find(text,'hour') ne 0) time = 3600*input(scan(text,1,' '),best.) +
                                        60*input(scan(text,3,' '),best.);
* just minutes;
   otherwise                     time = 60*input(scan(text,1,' '),best.);
  end;
  output;
  stop;
end;
keep zip1 zip2  distance time;
format time time.;
run;

filename x  clear;
```

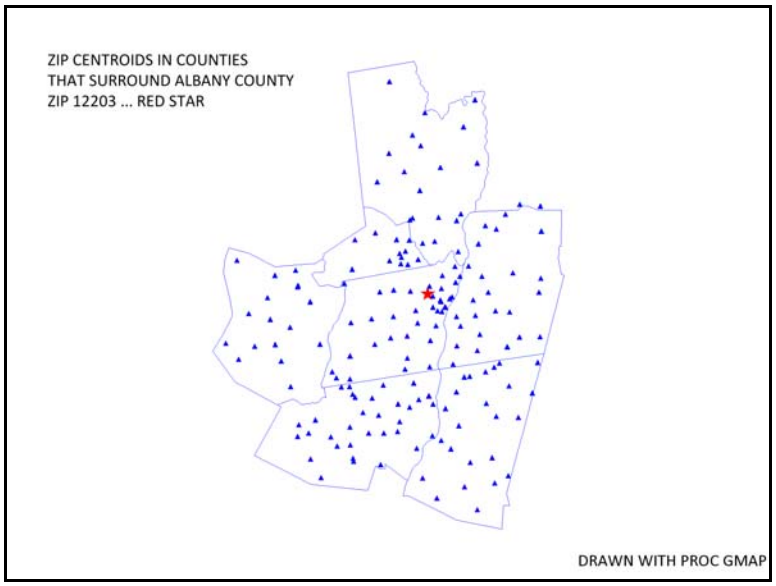
In the FILENAME statement, there is text preceded by an ampersand, &saddr, that does not represent a macro variable. The %NRSTR macro function prevents SAS from interpreting that text as a macro variable. In the data step, you see an INPUT statement with no variables listed. Each time an INPUT statement is used (with or without a variable list), the entire contents of the record that is read are moved into a variable with the name _INFILE_[4]. You can treat _INFILE_ as you would any other SAS data set variable. Thus you can use the character function FIND to search for the character string 'dditd>' within the HTML. One you find that string, a number of other functions (SUBSTR, SCAN, INPUT, FIND again) are used to create the variables DISTANCE and TIME. Given the distance between two locations, Google Maps produces a driving time in various different ways. The SELECT statement determines the type of time that is contained in the HTML and creates a real SAS time variable whose value can be used in subsequent data steps. The RETAIN statement adds the two zip codes to the data set which looks as shown on the right.

zip1	zip2	distance	time
12203	27513	656	10:34:00

Now that you can do one pair of locations, go back to the task mentioned earlier in the paper ... you have the location of a facility and want to know the number of people who are within various drive times to that facility. Assume that the facility is located in zip 12203 (Albany, NY). Example 3 automates the task of finding driving distances and times to all zip codes in the counties that surround Albany county (location of zip 12203). The first data step makes a data set of all those zip codes while the next one places the number of observations in that data set (170) into a macro variable. The portion of the WHERE statement 'ZIP_CLASS IS MISSING' eliminates zips used for PO boxes, large organizations, businesses, etc. The macro uses a loop to cycle through the observations in the data set ZIP_INFO, creating a new URL each time a new zip is read. The required information is extracted from the web page created in the background by Google Maps and added to a data set named DISTANCE_TIME.

```
* example 3;

* data set with zip codes;
data zip_info;
set sashelp.zipcode;
where state eq 36 and
      county in (1 21 39 83 91 93 95) and
      zip ne 12203 and
      zip_class is missing;
keep zip;
run;

* place number of zip in a macro variable;
data _null_;
call symputx('nzips',obs);
stop;
set zip_info nobs=obs;
run;

* create a macro that contains a loop to access Google Maps multiple time;
%macro distance_time;
* delete any data set named DISTANCE_TIME that might exist in the WORK library;
proc datasets lib=work nolist;
delete distance_time;
quit;

* read one observation at a time from the data set;
%do j=1 %to &nzips;
data _null_;
nrec = &j;
set zip_info point=nrec;
call symputx('z2',put(zip,z5.));
stop;
run;

* change one zip code in the URL ... zip 12203 is hard-coded as part of the URL;
filename x url "http://maps.google.com/maps?daddr=&z2.%nrstr(&saddr)=12203";

data temp;
retain zip &z2;
<same code as example 2 ... look at that example for details>
keep zip distance time;
run;

filename x  clear;

* add observation to the final data set;
proc append base=distance_time data=temp;
run;
%end;
%mend;

* use the macro;
%distance_time;
```

The map on the right shows the location of the zip centroids in counties in the Albany area. A random sample of the observations in the data set DISTANCE_TIME is shown just below the map on the right. The next task is to match the observations in that data set to one that contains the population of each zip code. Then you can determine the percentage of the population within various drive times to the facility located in zip 12203. Data from the 2000 census was merged to the observations in the data set DISTANCE_TIME. Once each observation contains both a driving time and a population, PROC UNIVARIATE can be used find various statistics.



The table on the right with the title 'DRIVE TIMES' shows that the median drive time to the centroid of zip 12203 is 23 minutes. It also shows that maximum drive time is 1 hour and 47 minutes and that 90% of the population in the counties shown in the map on the right is within a 53 minute drive of zip 12203. The SAS code used to compute the statistics shown in that table is shown below. In example 4, PROC UNIVARIATE is used and the FREQ option weights each observation by the number of people in each zip code.

Obs	zip	distance	time
14	12029	39.1	0:49
26	12052	24.6	0:37
32	12058	33.7	0:39
44	12077	9.0	0:16
48	12087	29.0	0:40
63	12131	52.2	1:05
123	12309	13.7	0:17

DRIVE TIMES

Quantile	Estimate
100% Max	1:47
99%	1:18
95%	0:59
90%	0:53
75% Q3	0:37
50% Median	0:23
25% Q1	0:14
10%	0:10
5%	0:08
1%	0:06
0% Min	0:06

```
* example 4;

ods listing close;
proc univariate data=zp;
var time;
freq pop;
ods output quantiles=qqq;
run;
ods listing;

ods html file='z:\quants.html' style=barrettsblue;
title 'DRIVE TIMES';
proc print data=qqq noobs;
var quantile estimate;
format estimate time5.;
run;
ods html close;
```

MORE POSSIBILITIES

Rather than use zip centroids, you can also use a full address in the Google Maps lookup. The SAS code in example 5 below can be used in place of the code shown at the top of example 2. You can also use addresses in the macro shown in example 3.

```
* example 5;

* my home;
%let addr1=59 Lenox Ave 12203;

* someplace in North Carolina;
%let addr2=SAS Campus Drive 27513;

data _null_;
call symput('a1',translate("&addr1",'+',' '));
call symput('a2',translate("&addr2",'+',' '));
run;

filename y url
"http://maps.google.com/maps?daddr=&a1=%nrstr(&saddr)=&a2";
```

Finally, there are occasions when you have locations defined in terms of latitude and longitude. In example 1, we found the latitude and longitude of the zip centroids for zips 12203 and 27513. Those coordinates can also be used in Google Maps to find a driving time and distance and example 6 uses that data.

```
* example 6;

%let ll1=%str(42.691560,-73.827840);
%let ll2=%str(35.805410,-78.797679);

* no changes required below this line;

filename x url "http://maps.google.com/maps?daddr=&ll2.%nrstr(&saddr)=&ll1";

data _null_;
infile x lrecl=32000 pad;
input;
loc = find(_infile_,'dditd>');
if loc ne 0 then do;
    text = substr(_infile_,loc,50);
    text = scan(text,1,'&');
    distance = input(scan(text,-1,'>'),comma12.);
    loc = find(_infile_,'about');
    text = substr(_infile_,loc,50);
    time = scan(text,3,'<>');
    file print;
    put "DRIVING DISTANCE BETWEEN ..." //
        "LAT/LONG: &LL1" /
        "LAT/LONG: &LL2:" //
        distance "MILES (TIME: " time ")";
    stop;
end;
run;

filename x  clear;
```

The results of example 6 are shown on the right. The distance and time are almost identical to those found when the zip codes are used in example 2, where the distance was found to be 656 miles and the time 10 hours and 34 minutes. Remember, the values used in example 6 are from the data set SASHELP.ZIPCODE. In example 2, Google Maps selects the exact points to use when only the zips are specified and the difference in values produced by examples 2 and 6 is most likely caused by a slight difference in the latitude and longitude used as the start and end points of the trip.

```
DRIVING DISTANCE BETWEEN ...

LAT/LONG: 42.691560,-73.827840
LAT/LONG: 35.805410,-78.797679:

654 MILES (TIME: 10 hours 41 mins )
```

CONCLUSION

A number of SAS tools allow a user to access Google Maps and extract driving times and driving distances. The main tool is the URL access method in the FILENAME statement. Then, the MACRO language can automate the task of looking up distances and times for many pairs of locations, specified either as pairs of zips, addresses, or lat/long coordinates.

REFERENCES

1 "Calculate distance, bearing and more between Latitude/Longitude points", http://www.movable-type.co.uk/scripts/latlong.html

2 Hadden, Louise and Zdeb, Mike. “ZIP Code 411: A Well-Kept SAS® Secret”, Proceedings of the Thirty-First Annual SAS Users Group International Conference. (2006) http://www2.sas.com/proceedings/sugi31/143-31.pdf

3 "Vincenty formula for distance between two Latitude/Longitude points", http://www.movable-type.co.uk/scripts/latlong-vincenty.html

4 Schreier, Howard. “Now _INFILE_ is an Automatic Variable – So What?”, Proceedings of the Thirteenth Annual Northeast SAS Users Group Conference. (2001) http://www.nesug.org/proceedings/nesug01/cc/cc4018bw.pdf

SAS COMMUNITY

Much of the material in this paper first appeared on the SAS Community wiki. More code and additions to the material found in this paper can be found at ...

http://www.sascommunity.org/wiki/Driving_Distances_and_Drive_Times_using_SAS_and_Google_Maps

and other postings by the author can be found at ...

http://www.sascommunity.org/wiki/User:Msz03

ACKNOWLEDGMENTS

This work was funded in part by NIH grant HHSN267200700019C from the *Eunice Kennedy Shriver* National Institute of Child Health and Human Development.

CONTACT INFORMATION

The author can be contacted using e-mail ... Mike Zdeb msz03@albany.edu