Documentation for SYNTHESIZER Program

John E. Abraham, J.D. Hunt and Kevin J. Stefan, June 2011
Version 1.1

INTRODUCTION

This note describes the setup and use of the SYNTHESIZER software for generating synthetic populations.

METHOD

The software works to identify a list of units whose aggregate attribute values match a pre-specified set of corresponding target values. This list forms a synthetic population of such units consistent with the target values. Each unit included in this list is drawn from a sample of such units, with the potential that any particular unit in the sample is included in the list 0, 1 or more times as appropriate.

The software proceeds by iteratively considering one of three operations:

1) adding a unit from the sample to the list,
2) subtracting a unit from the list, or
3) a 'swap' where a unit in the list is swapped out and a unit from the sample is swapped in.

The match of the list to the target values is scored using a goodness-of-fit function.

The list is divided into subgroups, which are commonly used to specify geographic areas known as "zones" which are to contain portions of a population. The process works through the list subgroup by subgroup. For each subgroup first one of the three operations is selected with equal 1/3 probability (add, subtract or swap). In the case of subtract or swap a unit in the subgroup is randomly selected. In the case of add or swap a unit in the sample is randomly selected, with probability described under "Sample Probabilities", below. The operation is then performed, and the magnitude of the improvement in the goodness-of-fit score is calculated. If the goodness of fit improves the operation is kept. If the goodness of fit gets worse there is a less-than-1.0 probability that the operation will be kept, otherwise the operation will be undone.

It is possible to have the program start with a previously generated list. If no such previously generated list is available, an initial list is generated by randomly selecting enough units at random to reach or exceed target values for just one of the attributes for each subgroup.

The decision to keep an operation at any point includes a probabilistic component. Operations that lead to a worse goodness of fit will be more likely to be accepted early in the process than later in the process. This is what is termed a 'simulated annealing' algorithm – based on the idea that the program should be getting closer to the best possible match as the number of iterations increases and thus a non-improving swap is

more likely to be detrimental rather than advantageous in the search for this best possible match.

The general formula used in the measurement of goodness-of-fit is:

$gof = Sqrt(\Sigma_a \, weight_a^2 \cdot (list_a\text{-}target_a)^2)$

where:

a	= index of attributes whose aggregate values for the synthetic population list are to match a pre-specified set of corresponding target values
gof	= goodness-of-fit, with values closer to 0 indicating a better fit (such that it might be appropriate to consider it a 'lack-of-fit' measure).
$weight_a$	= weight associated with attribute a
$list_a$	= aggregate value for attribute a for the list
$target_a$	= target value for attribute a

The weight associated with an attribute indicates the relative importance to be placed on achieving a match with regard to that attribute.

The formula used to assign the probability of accepting an operation that leads to a worse goodness of fit is:

$P = exp(-iteration/ \alpha)^(\Delta gof^\gamma)$

where:

P	= probability of accepting the operation
Δgof	= change in goodness-of-fit associated with the operation
γ	= parameter controlling influence of the size of the change in goodness-of-fit on the probability of making the swap, specified as *gofDifExponent* in the properties file for the program
iteration	= number of operations that have been evaluated so far in the process
α	= parameter controlling influence of the number of iterations on the probability of making the swap, specified as *coolingParameter* in the properties file for the program

It is common to set γ to 0.0 when first setting up the synthesizer, so that the probability of accepting an operation that leads to a worse goodness of fit is simply

$P = exp(-iteration/ \alpha)$.

It is also common to set α to zero, to turn off the "simulated annealing" feature and to only accept operations that lead to a better fit. Simulated annealing is often not necessary, especially once the weights are well chosen. If the weights are poorly chosen or the problem is complex, simulated annealing can help the algorithm overcome a

temporary situation where one target with too high of a weight has already achieved it's best match while other targets with too low of a weight could still be improved.

The number of iterations to be performed by the program is specified by the user as part of the inputs.

The target values for the attributes can be specified for individual subgroups of the population or for combinations of the subgroups. The program proceeds subgroup-by-subgroup in its processing, with the number of iterations within each subgroup for each pass through the entire list of subgroup determined according to the comparative goodness-of-fit for the subgroup.

The formula used to establish the number of iterations for a given subgroup is:

$n_z = \iota \cdot gof_z + j$

where:

z	= index of subgroup
gof_z	= goodness-of-fit for subgroup z
ι	= parameter controlling influence of the goodness-of-fit value for a subgroup on the number of iterations for the subgroup as part of the current pass through the subgroups, called *iterationsPerZonalLackOfFit* in the properties file.
j	= parameter controlling the minimum number of iterations for the subgroup, called *minIterationsPerRow* in the properties file, with a default value of 2.

The program performs iterations on each subgroup, checking the total number of iterations every time a subgroup is processed, and terminating if the total number of iterations meets or exceeds the specified number of iterations. Upon termination the program reports the overall goodness-of-fit, the goodness of fit for each subgroup and for each target in each subgroup, and the resulting list of units comprising the synthetic population.

Sample Probabilities

When choosing a sample for possible inclusion in a subgroup, the sample will be randomly chosen according to selection probabilities. If there is no *weightColumn* entry in the properties file, the selection probabilities will be uniform. If there is a *weightColumn* entry but no *weightMultipliers.csv* file in the *inputLocation*, the selection probabilities will be proportional to the entries in the appropriate column in the samplesTable.

If there is both a *weightColumn* entry in the properites file and a *weightMultipliers.csv* file in the *inputLocation*, the selection probabilities will be proportional to $weight_{s,z}$ in the following formula:

$weight_{s,z}$ = $weight_s \cdot$ {wm[z,$puma_s$] if exists, *dswm* otherwise}

where:

z = index of subgroup

s = index of sample

$weight_s$ = *a priori* weight entry for the sample as specified in the column labeled *weightColumn*

$puma_s$ = integer identifier for the sample s, usually used to identify the approximate geographic origin from the sample, the column name for this entry in the samplesTable is given in the properites file as *sampleGeographyColumnName*

wm[z,$puma_s$] = possible entry in weightColumn table for the subgroup and puma (not all entries need exist)

dswm = default sample weight multiplier, specified as *defaultSampleWeightModifier* in the properties file, and normally set to 0 or 1.

In this way the user can use simple uniform sampling by not specifiying a *weightColumn*, use non-uniform sampling by setting *weightColumn* but not providing a *weightMultipliers.csv* file, or use sampling which considers sample weights modified by the geographic origin of the sample, so that samples which were drawn from areas within or close to the subgroups can be given a higher weight.

Instance Count Penalties

The goodness of fit formula can be modified to include instance count penalties. Each sample can have a target minimum number of times it appears in the population, in a column who's name is specified by the *lowInstanceCountColumn* entry in the properties file, and a target maximum number of times it appears in the population, in a column who's name is specified by the *highInstanceCountColumn* entry in the properties file. The goodness of fit measure will then be:

$gof = Sqrt(\Sigma_a\ weight_a^2 \cdot (list_a\text{-}target_a)^2) + \Sigma_s\{hip \text{ if } Ns > hict_s, lip \text{ if } Ns < lict_s\}$

where the elements in the first term are as previously defined, and:

s = index of sample

hip = penalty associated with too high occurrence of sample, specified as *highInstancePenalty* in the properties file

lip = penalty associated with too low occurrence of sample, specified as *lowInstancePenalty* in the properties file

$hict_s$ = target in *highInstanceCountColumn* for sample s

$lict_s$ = target in *lowInstanceCountColumn* for sample s

INPUTS

The inputs to the program are provided in a set of computer files as follows:

1. A file referred to as the samplesTable (often called *Samples.csv*; the file must have the filename extension ".csv" but the name of the file is specified in the properties file, described below) contains a list of the full set of sample units. Each line contains all the attribute values for one unit.

A screenshot showing the format of this file is included below. The first line contains a set of column headers indicating the attribute label for the values included on each remaining line of the file; these labels must match the corresponding labels for the target values specified in the targetsTable file as described below. (If there is a column label used here that does not match any column label in targetsTable, then the values are retained in the synthesized list of units but are not considered in any target matching, which allows the user to use such values in further calculations and to retain such values for subsequent application.)

The first column in the samplesTable must contain a number which uniquely identifies the sample. (Numbering the rows sequentially using the first column would suffice if no prior numbering system exists.)

Empty entries can occur in the samples table if a data value is not available for a sample, however only "average" targets (indicated with "-A", see below) can be specified for columns which contain empty entries. Empty entries are entered with no characters, so two consecutive commas appear in the file.

The extension ".csv" refers to "comma separated value" and implies a text file that can be interpreted as a table. The rows in the table are separate lines in the text file, and the columns in the table are separated by commas. This type of file can be directly edited by a large number of programs, including database programs and spreadsheet programs such as Microsoft Excel.

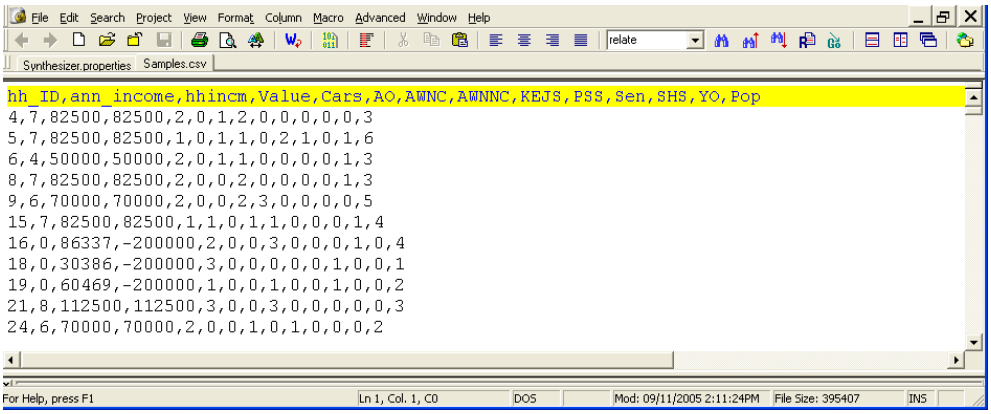
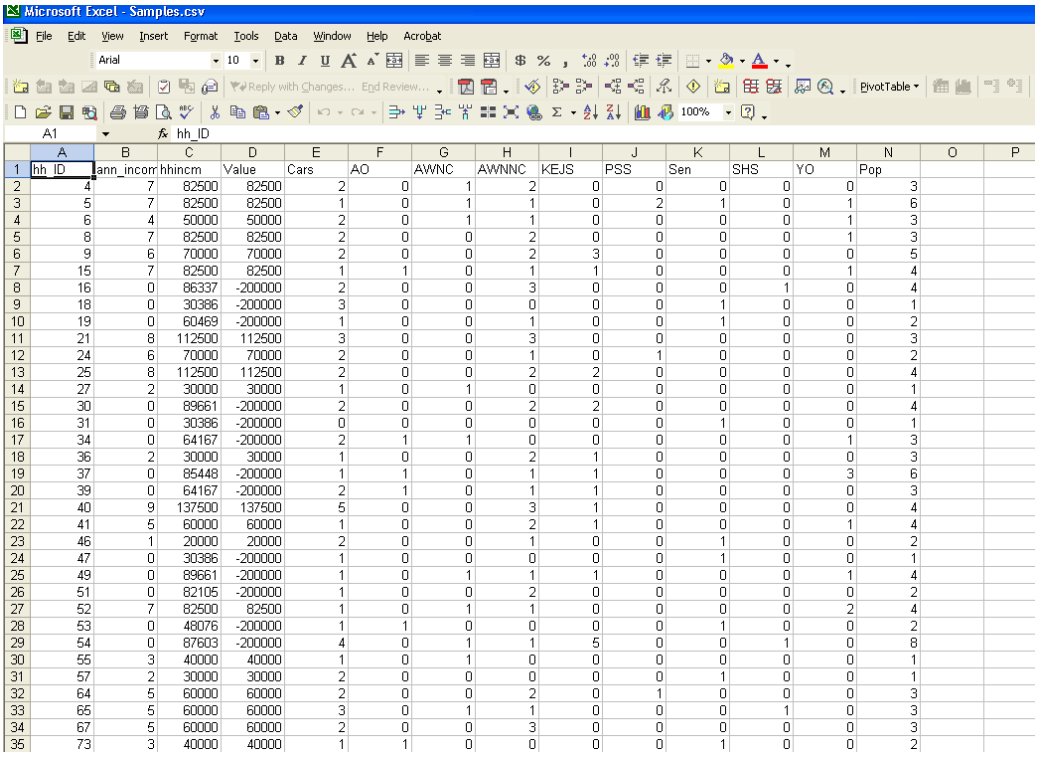


Figure 1: Editing the Samples.csv file in the text editing program UltraEdit.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	hh_ID	ann_incom	hhincm	Value	Cars	AO	AWNC	AWNNC	KEJS	PSS	Sen	SHS	YO	Pop		
2	4	7	82500	82500	2	0	1	2	0	0	0	0	0	3		
3	5	7	82500	82500	1	0	1	1	0	2	1	0	1	6		
4	6	4	50000	50000	2	0	1	1	0	0	0	0	1	3		
5	8	7	82500	82500	2	0	0	2	0	0	0	0	1	3		
6	9	6	70000	70000	2	0	0	2	3	0	0	0	0	5		
7	15	7	82500	82500	1	1	0	1	1	0	0	0	1	4		
8	16	0	86337	-200000	2	0	0	3	0	0	0	1	0	4		
9	18	0	30386	-200000	3	0	0	0	0	0	1	0	0	1		
10	19	0	60469	-200000	1	0	0	1	0	0	1	0	0	2		
11	21	8	112500	112500	3	0	0	3	0	0	0	0	0	3		
12	24	6	70000	70000	2	0	0	1	0	1	0	0	0	2		
13	25	8	112500	112500	2	0	0	2	2	0	0	0	0	4		
14	27	2	30000	30000	1	0	1	0	0	0	0	0	0	1		
15	30	0	89661	-200000	2	0	0	2	2	0	0	0	0	4		
16	31	0	30386	-200000	0	0	0	0	0	0	1	0	0	1		
17	34	0	64167	-200000	2	1	1	0	0	0	0	0	1	3		
18	36	2	30000	30000	1	0	0	2	1	0	0	0	0	3		
19	37	0	85448	-200000	1	1	0	1	1	0	0	0	3	6		
20	39	0	64167	-200000	2	1	0	1	1	0	0	0	0	3		
21	40	9	137500	137500	5	0	0	3	1	0	0	0	0	4		
22	41	5	60000	60000	1	0	0	2	1	0	0	0	1	4		
23	46	1	20000	20000	2	0	0	1	0	0	1	0	0	2		
24	47	0	30386	-200000	1	0	0	0	0	0	1	0	0	1		
25	49	0	89661	-200000	1	0	1	1	1	0	0	0	1	4		
26	51	0	82105	-200000	1	0	0	2	0	0	0	0	0	2		
27	52	7	82500	82500	1	0	1	1	0	0	0	0	2	4		
28	53	0	48076	-200000	1	1	0	0	0	0	1	0	0	2		
29	54	0	87603	-200000	4	0	1	1	5	0	0	1	0	8		
30	55	3	40000	40000	1	0	1	0	0	0	0	0	0	1		
31	57	2	30000	30000	2	0	0	0	0	0	1	0	0	1		
32	64	5	60000	60000	2	0	0	2	0	1	0	0	0	3		
33	65	5	60000	60000	3	0	1	1	0	0	0	1	0	3		
34	67	5	60000	60000	2	0	0	3	0	0	0	0	0	3		
35	73	3	40000	40000	1	1	0	0	0	0	1	0	0	2		

Figure 2: Editing the Samples.csv file in spreadsheet program Microsoft Excel

2. A file referred to as the targetsTable (often called *Targets.csv*; the file must have the filename extension ".csv" but the name of the file is specified in the properties file, described below) contains the list of subgroups, and details the specified targets to be matched in the synthesized list of units produced by the program.

The first line contains a set of column headers indicating the attribute labels for the targets to be matched in each zone. Additional suffixes are used to indicate special kinds of targets: the '-A' suffix is used to indicate an average value rather than a total; and the '-D' suffix is used to indicate a value where double precision is to be used by the program (rather than integer precision).

If the column header contains a "/" character, then the target is the ratio of two other targets. For example if there were two columns labeled "Sen" and "PSS" in the samples table (perhaps indicating the number of senior citizens and post secondary students, respectively, in a household) then a target column labeled "Sen/PSS" could be used to specify the ratio of the total of "Sen" and "PSS" in the group (e.g. the ratio of senior citizens to post secondary students in the subgroup.)

The first (left-most) column is reserved for the numerical labels for the population subgroups.

The second row contains weights for use in the calculation of the goodness-of-fit score 'gof', these are the values to use for the 'weight_a' for the set of attributes indexed a. Note that a weight of zero completely removes the target from consideration.

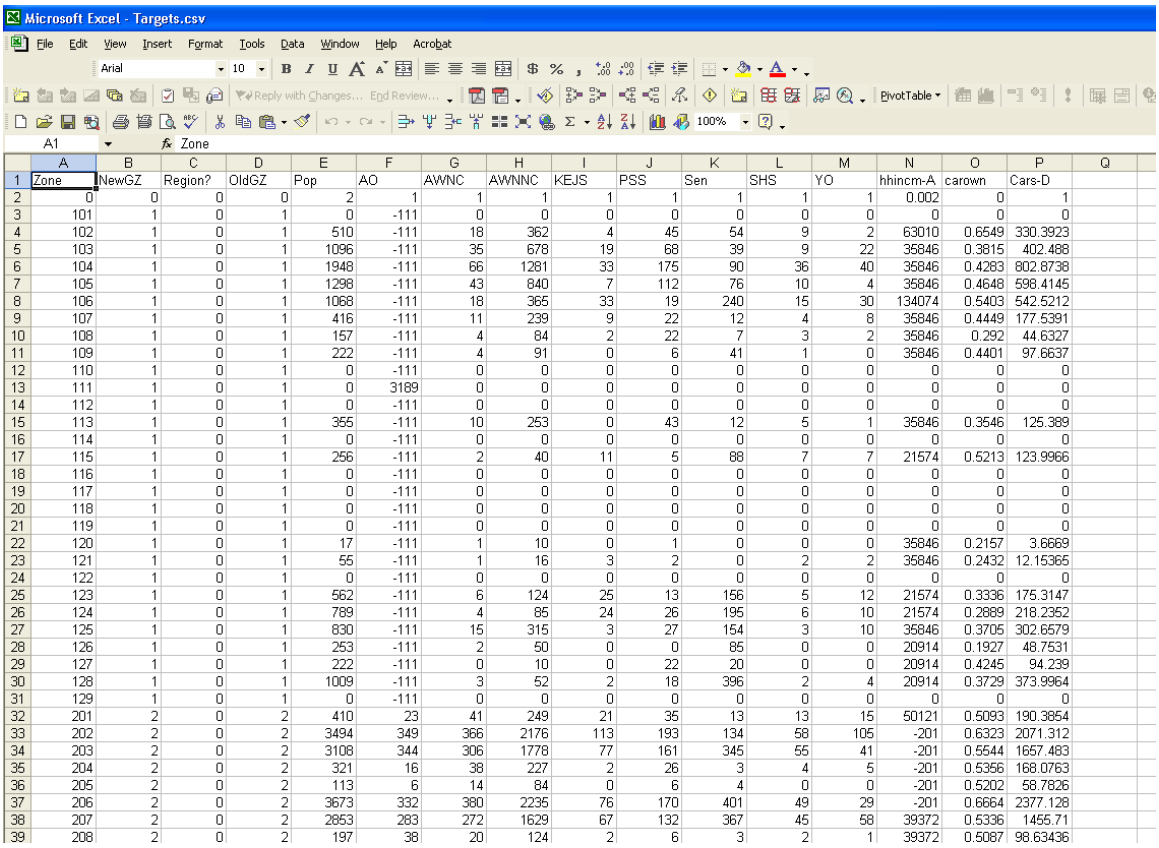
Each subsequent line after the second indicates the specific target values to be matched by the program for a specific subgroup.

If a negative sign is included with a given number in the spot for the target value, then the number (the absolute value) is taken to indicate another subgroup and the corresponding value in the spot for this other subgroup is used as the target for the sum (or average, if the –A suffix is specified in row 1) of the values in the synthetic list for all the zones that include a negative sign number 'pointing' to this same positive zone. An example of the use of this negative sign is included in the screenshot below.

A screenshot showing the format of this file is included below. Note that in this example, the negative numbers '-111' in the column labeled AO indicate that for all of the Zones (subgroups) numbered between 101 and 129 (rows 3 through 31 in the file) the total quantity target for the AO attribute should be 3189 (specified in row 31 for Zone 111).

Note also that for the combined population in Zones 201, 202, 203, 204, 205 and 206 the hhincm average should be 50121, as specified in the column labeled hhincm-A in rows 32 through 37 in the file.

Empty entries can occur in the targets file (indicated with no data, so the file would contain two consecutive commas.) These indicate that there is no target data for the target for the subgroup and the set of targets is reduced by one.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Zone	NewGZ	Region?	OldGZ	Pop	AO	AWNC	AWNNC	KEJS	PSS	Sen	SHS	YO	hhincm-A	carown	Cars-D	
2	0	0	0	0	2	1	1	1	1	1	1	1	1	0.002	0	1	
3	101	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
4	102	1	0	1	510	-111	18	362	4	45	54	9	2	63010	0.6549	330.3923	
5	103	1	0	1	1096	-111	35	678	19	68	39	9	22	35846	0.3815	402.488	
6	104	1	0	1	1948	-111	66	1281	33	175	90	36	40	35846	0.4283	802.8738	
7	105	1	0	1	1298	-111	43	840	7	112	76	10	4	35846	0.4648	598.4145	
8	106	1	0	1	1068	-111	18	365	33	19	240	15	30	134074	0.5403	542.5212	
9	107	1	0	1	416	-111	11	239	9	22	12	4	8	35846	0.4449	177.5391	
10	108	1	0	1	157	-111	4	84	2	22	7	3	2	35846	0.292	44.6327	
11	109	1	0	1	222	-111	4	91	0	6	41	1	0	35846	0.4401	97.6637	
12	110	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
13	111	1	0	1	0	3189	0	0	0	0	0	0	0	0	0	0	
14	112	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
15	113	1	0	1	355	-111	10	253	0	43	12	5	1	35846	0.3546	125.389	
16	114	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
17	115	1	0	1	256	-111	2	40	11	5	88	7	7	21574	0.5213	123.9966	
18	116	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
19	117	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
20	118	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
21	119	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
22	120	1	0	1	17	-111	1	10	0	1	0	0	0	35846	0.2157	3.6669	
23	121	1	0	1	55	-111	1	16	3	2	0	2	2	35846	0.2432	12.15365	
24	122	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
25	123	1	0	1	562	-111	6	124	25	13	156	5	12	21574	0.3336	175.3147	
26	124	1	0	1	789	-111	4	85	24	26	195	6	10	21574	0.2889	218.2352	
27	125	1	0	1	830	-111	15	315	3	27	154	3	10	35846	0.3705	302.6579	
28	126	1	0	1	253	-111	2	50	0	0	85	0	0	20914	0.1927	48.7531	
29	127	1	0	1	222	-111	0	10	0	22	20	0	0	20914	0.4245	94.239	
30	128	1	0	1	1009	-111	3	52	2	18	396	2	4	20914	0.3729	373.9964	
31	129	1	0	1	0	-111	0	0	0	0	0	0	0	0	0	0	
32	201	2	0	2	410	23	41	249	21	35	13	13	15	50121	0.5093	190.3854	
33	202	2	0	2	3494	349	366	2176	113	193	134	58	105	-201	0.6323	2071.312	
34	203	2	0	2	3108	344	306	1778	77	161	345	55	41	-201	0.5544	1657.483	
35	204	2	0	2	321	16	38	227	2	26	3	4	5	-201	0.5356	168.0763	
36	205	2	0	2	113	6	14	84	0	6	4	0	0	-201	0.5202	58.7826	
37	206	2	0	2	3673	332	380	2235	76	170	401	49	29	-201	0.6664	2377.128	
38	207	2	0	2	2853	283	272	1629	67	132	367	45	58	39372	0.5336	1455.71	
39	208	2	0	2	197	38	20	124	2	6	3	2	1	39372	0.5087	98.63436	

3. An optional file called "weightMultipliers.csv" which consists of three columns. One column should have the same name as the first column in the targetsTable and identifies subgroups. The second column should have the same name as the sample geography column in the samplesTable, specified as *sampleGeographyColumnName* in the properties file. The third column should be named "weightMultiplier". When establishing the probabilities for choosing each sample for possible addition into each subgroup, the sample weight from the *weightColumn* in the samplesTable will be multiplied by the appropriate factor from the weightMultipliers table if it exists, as described above. The entries in this table are the $wm[z,puma_s]$ values in the formula.

4. The .properties file, commonly called *Synthesizer.properties* (must have the file extension ".properties".) This file is a text file containing the specification of the required run-time inputs for the program. Each line in this file contains two elements, a property name and then a property value. The property name and its corresponding values are separated by an equals sign.

The property names are:

inputLocation = the directory containing the input files for the program.

samplesTable = the name of the file containing the samples, without the .csv extension (e.g. if samplesTable=samples then the program will look for the file called "samples.csv").

targetsTable = the name of the file listing the subgroups and describing the targets and the weights associated with each target, without the .csv extension.

maxIterations = the number of iterations to proceed before stopping.

gofDifExponent = the parameter γ in the probability equation shown above.

coolingParameter = the parameter α in the probability equation shown above.

initialGenerationBasis = the basis for the initial population to be used at the start of the operations. If the word "previous" (without the quotes) is specified, then the resulting population from a previous run of the program – stored in the file named "Output******.csv" as described below – is used. If the name of one of the columns is specified, then the values for that column name are used to generate an initial population before any operations are tried. In this process samples are randomly added to each subgroup until the total for this column meets or exceeds the target specified for this column. Note that a target is required for each row in this column, i.e. no group targets can be specified for this column using negative signs and no null entries can exist. If this property is not specified then the second column in the targetsTable is used to generate the initial population.

iterationsPerZonalLackOfFit = ι in the equations above, controlling the number of operations attempted in each subgroup before proceeding to the next subgroup.

minIterationsPerRow = *j* in the equations above, controlling the minimum number of operations attempted in each subgroup before proceeding to the next subgroup (defaults to 2.)

weightColumn = the name of the column used for weights when selecting samples for possible addition to the population, if this property is not specified uniform sampling will be used.

sampleGeographyColumnName = if this entry exists and weightColumn also exists, it specifies the column name in weightMultipliers.csv as well as a column name in samplesTable that is used to determine $puma_s$. If this entry exists and weightColumn also exists, then there must also exist a weightMultipliers.csv file to read in the values for $wm[z,puma_s]$

defaultSampleWeightModifier = the value to be used *dswm* in the formulas, this entry only needs to exist in the properties file if a sampleGeographyColumnName exists in the properties file.

highInstancePenalty = penalty associated with too high occurrence of sample, hip in the formulas.

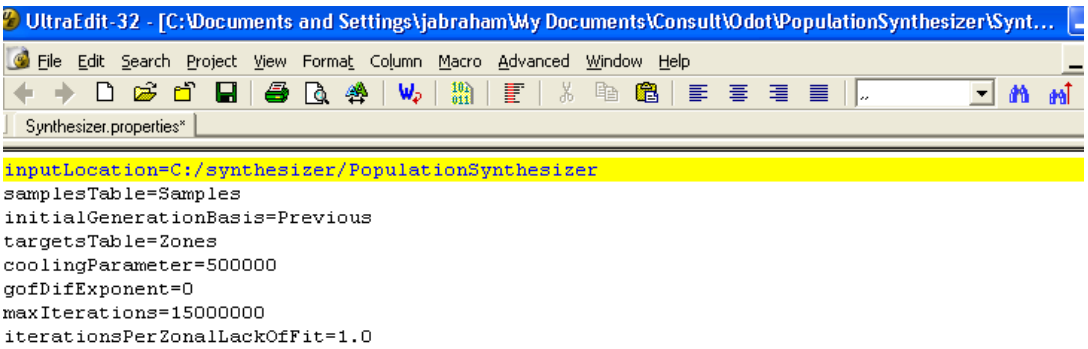
lowInstancePenalty = penalty associated with too high occurrence of sample, lip in the formulas.

highInstanceCountColumn = column name for high instance count target in samplesTable.

lowInstanceCountColumn = column name for low instance count target in samplesTable.

If highInstancePenalty or lowInstancePenalty are not defined, instance count penalties will not be used and neither highInstanceCountColumn nor lowInstanceCountColumn need to be defined.

A screenshot showing the format of this file is included below.



4. A file controlling the output of debug logging statements is required. This file is normally called log4j.xml and an example is included in the distribution. Documentation on the format of this file is available at http://logging.apache.org/ It is not normally necessary to change the content of this file.

OUTPUTS

*Output******.csv* (with ****** in this name the same as the name specified for the samplesTable file, which is "Samples" in the screenshot example shown above). This file contains the final list of units produced by the program. The file consists of two columns, the first column is called "Zone" and identifies the subgroup of the population using the same numbers as in the leftmost column of the targetsTable. The second column is called "UnitID" and identifies a sample using the same numbers as in the leftmost column of samplesTable.

Fit.txt This file reports on the extent that the final synthetic list of units produced by the program matches the specified targets. It shows for each subgroup the specified target values and corresponding achieved values and goodness-of-fit for each target, along with the gof score for each subgroup.

RUNNING THE PROGRAM

The program is a Java program, and can be run in a number of ways. The program requires Java 1.5 (also known as Java 5) or later, which should be downloaded and installed from java.sun.com. To check which Java version is installed on a Microsoft Windows machine, open a windows CMD prompt and type "java –version".

A windows command file called "synthPop.cmd" is included in the distribution. This file contains the command shown below. In Windows, if all of the input files and

synthPop.cmd are all located in the same directory, the program can be run by simply double-clicking on synthPop.cmd.

Windows is not required to run the program; the program runs on any operating system with a Java runtime environment.

The program is contained in the following .jar file:

HBAPopulationSynthesizer.jar

Double clicking this jar file in most operating systems (including Microsoft Windows) will cause the program to run, and look for a file called "Synth.properties" in the same directory as the properties file.

If your samples table and/or targets tables are large, the program might run out of memory when you launch it by double-clicking. In that case you will see an error similar to this:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

and you will need to launch the program from a command prompt by typing

java –xmx2000M –jar HBAPopulationSynthesizer.jar

where "2000M" refers to the amount of memory you want to allocate to the program, in this case 2000 Megabytes.

To use a different properties file, append it to the command, e.g.

java –xmx2000M –jar HBAPopulationSynthesizer.jar special_area.properties

LICENSE

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

The source code is included in the .jar files, should you wish to modify or enhance the program as allowed under the License.