Object Oriented Population Generation

Jacob Barhak

Austin Texas
jacob.barhak@gmail.com

ABSTRACT

Computational population generation produces individual records using Monte-Carlo techniques. Generation rules for each individual characteristic potentially depend on other characteristics which create a dependency tree for computing each individual. In addition evolutionary computation objectives define selection criteria of individuals to match aggregate population goals. These elements repeat many times when creating new populations, therefore code reusability is important to simplify data entry. Moreover it allows exchanging computational building blocks to create simulation variations for sensitivity analysis or hypothesis testing which are essential for reference modeling. This paper describes an object oriented approach where population building blocks can be inherited to build a population that later can be rebuilt with variations. The paper describes the modular modeling theory with some basic examples.

ABOUT THE AUTHORS

Jacob Barhak specializes in chronic disease modeling with emphasis on using computational technological solutions. The Reference Model for disease progression was self developed by Dr. Barhak as a freelancer. Previously Dr. Barhak headed the Michigan Model for Diabetes computing team 2006-2012. A major part of this position involved developing the software environment and the Michigan Model for Diabetes. Dr. Barhak has diverse international background in engineering and computing science. For additional information please visit http://sites.google.com/site/jacobbarhak/.

Object Oriented Population Generation

Jacob Barhak

Austin Texas
jacob.barhak@gmail.com

INTRODUCTION

A generated population of individuals with multiple characteristics is a common input to Micro-Simulation, which is becoming increasingly popular with the increase in computing power (Moore 1965, Wikipedia Online C). The abundance of computing power, however, allows modelers to test many hypotheses and scenarios. The Reference Model (Barhak & Garrett 2014A, Barhak 2012A, 2012B, 2012C, 2012D, 2012E, 2013A, 2013B, 2013C, 2013D, 2014B, 2014C, 2014D) is a good example that demonstrates how different models, populations, and hypotheses are combined to create many scenarios for disease progression. This trend of reference modeling was identified by (Tolk et al. 2013). Yet even beyond reference modeling, models typically undergo sensitivity analysis where many scenarios are tested with different parameter values (Hayes et al. 2013).

Many times this requires changing a base building block in simulation such as an initial generated population characteristic. Since population generation has dependencies, there is a need to replace foundation blocks during population generation without difficulty. An object oriented approach towards population generation allows easily defining interchangeable population building blocks that can be used in different scenarios.

APPROACH

The base technology for population generation is described in (Barhak 2010, 2013C, 2014B, Barhak & Garrett 2014A, Barhak @ Github Online). Here is a short recap of previous work to allow better understanding of the new layer.

Parameters

Each individual is defined by a set of parameters. For example *Age, Male, Weight, Height,* etc. Each such parameter definition may have validation rules such as being an integer or between certain values. For example, *Male* is defined as an integer within the ranges 0 and 1 to represent false and true values.

Rules

Each such parameter used in the population can have a rule attached to it. The rule may be a random function that depends on other parameters. The system knows to resolve the calculation order of these rules according to dependencies. This frees the user to consider each parameter definition at a time – which simplifies programming. For example, the user can use the rules defined for population A below.

Population A:
Rules:
*Age ~ Gaussian (50 + 2*Male,5)*
Male ~ Bernoulli (0.5)
*Height ~ Gaussian(1.7+0.1*Male,0.1)*
Weight ~ (Age+Uniform(0,20))(Height-0.75)*

In the above example regardless of the order if definition the system will know to calculate the individual parameter values in the following order: *Male, Age, Height, Weight*.

Therefore the program that defines the population looks different than common scripts since execution order is based on dependencies rather than on definition order. This is possible since there is a single expression for each parameter. We will call this expression a "parameter rule" or "rule" for short to distinguish it from other expressions.

Note that the rule can reference an auto increment counter system parameter we will name *IndividualID*. This allows loading a preset data table to a population where all values are preset. For example this may look like Population B:

```
Population B:
Rules:
  Age ~ Table ([[Mod(IndividualID,3),[]]], [50,60,40])
```

In this example, *IndividualID* will give a different value for each of the first three individuals according to the table. In past publications, such a population would have been characterized as a data population. In this paper we generalize population definition and data is considered as a rule defined by an expression using a table function depending on *IndividualID*.

Objectives

After individuals are generated Evolutionary Computation (EC) is used. EC selects sub-sets of individuals with certain characteristics that match desired statistics. These statistics are represented by objectives defined by the following elements:

- **Filter Expression:** an expression that is true for each individual the objective applies to – essentially defining a sub group of interest. Many times no filter is used – indicated all individuals are considered by using a constant value of 1.
- **Statistics Expression:** an expression applied on the individual to calculate a single statistical value of interest – many times the expression consists of a single individual parameter.
- **Function:** a statistical aggregate function applied on the statistic expression result for all individuals where the filter expression was not zero. Typically this function is MEAN, STD, or a percentile selected from a list of supported functions.
- **Target Value:** A number which the result of the above function should match. This is the number we want to reach.
- **Weight:** a number to indicate the relative importance of the objective compared to other objectives.

An example for objectives to match our example is Population C:

```
Population C:
Objectives:
  Objective #1:
    Filter Criteria: 1
    Statistics Expression: Weight/Height**2
    Function: MEAN
    Target Value: 25
    Weight:1
```

The above objective makes sure that the mean Body Mass Index (BMI) of the generated population is 25. Once such an objective is in place, the system would select a sub population of individuals that matches the objective statistics. Note that this may skew the population. Yet proper definition of objectives will allow correcting skewed generated populations to match certain criteria, or at least know how well the population is generated within given constraints.
One important use of generating mock populations is to represent clinical trial baselines. Inclusion and exclusion criteria are modeled as rules and published statistics as objectives. This information exists in clinical trial publications. Such information is abundant and typically unrestricted. Therefore many populations can be modeled this way.

HANDLING MULTIPLE POPULATIONS AND FUTURE NEEDS

To efficiently encode many populations and looking towards future functionality there is a need to address the following issues:

Code Reusability
The use of repeated code between different cohorts of the same population is common. For example, both treatment arm and placebo arm may contain same initial conditions of disease state such as having diabetes, or may report duration of diabetes while we are interested in age of diagnosis that requires calculation. These rules can repeat for each cohort/stratification resulting in extra effort and code duplication that makes changes harder.

Missing Information
Missing information which either was not collected or was unpublished is common. For example, not all trials may publish statistics such as all cholesterol levels and there may be a need to use Friedewald Equation (Johnson et al. 1997) to calculate the missing cholesterol levels. Also, many conditions and bio-markers are not reported for each study yet may be required by the system and need to be derived from other sources. For example a study may not report history of heart disease in the family and such a parameter needs to be derived from another source that provides a default value.

Hypothesis Handling for use with High Performance Computing (HPC)
Hypotheses include unknown values that the user wishes to test during simulation such as different correlations amongst biomarkers (Barhak 2013D, 2014B). However, this can also include sensitivity analysis where a certain population parameter is changed to see how it influences simulation results. The availability of HPC requires the ability to easily generate many such hypotheses and handle them compactly.

Handling Individual Data
Being able to handle individual data derived from Electronic Medical Records (EMR) or a similar source is important for the future. Such data may come in a table with missing values that require replacement with calculated values and merged with another population defined by distributions.

Fortunately the above listed issues can be handled using known computer science techniques - mainly used in Object Oriented Programming (Wikipedia Online C).

SUGGESTED OBJECT ORIENTED FRAMEWORK

Consider each population as a class where rules and objectives are similar to class methods. And consider the ability to use inheritance to combine different populations into one. This is the main idea behind the approach presented here. Yet some differences exist and deserve further explanation.
In addition to the population definitions stated above, of parameters associated rules and objectives, each population will have an inheritance list.
Each element in the list will indicate:

- **Inherited Population:** Each such population may inherit others and therefore build an inheritance tree.
- **Inheritance Type:** One of the following elements: Rules, Objectives, or Data. Those indicate what would be inherited and when.

The order of elements in the inheritance list is important since inherited elements are processed in order before processing the inheriting population. Since inheritance can be nested creating an inheritance tree, processing in order means traversing in Depth First Search (DFS) order (Wikipedia Online A).

Starting from an empty population each population in its turn will contribute/override the following elements according to inheritance type:

- **Rules:** all rules that exist in the inherited population will be added to the inheriting population, possibly overriding a rule that affects the same parameter
- **Objectives:** all objectives that exist in the inherited population will be added to the inheriting population, possibly overriding an objective that has the same Filter Criteria, Statistics Expression, and Function. This means target value and weight will be overridden for those elements while an objective with a different filter criteria, for example, will be added as a different objective.
- **Data:** The population is generated according to rules and objectives and the data for each individual is transformed into a table that creates a new rule which is passed to the inheriting population. No objectives or original rules are inherited; the inheriting rules are similar to passing a table of data.

Data inheritance is different since it requires execution and therefore can be completed only in run time. Nevertheless it is possible to deduce at compile time which parameters will be generated by the final inheriting population, and therefore deduce dependencies amongst parameters in rules to avoid circular definitions where a parameter references itself. This check is executed before compiling the rules to code that generates population data. Data generation happens whenever data inheritance is encountered in the DFS search in the inheritance tree. It is somewhat equivalent to inheriting an object in run-time. This is sometimes referred to as prototype inheritance in scripting languages such as Python (Lutz 2006, Python Software Foundation - Online) as opposed to inheritance of classes in compile time in languages such as C++. This will be made clearer by example.

EXAMPLES

The simplest examples would be reusing the population given as examples above as building blocks to construct a population that contains those. For example if we want a population that contains both rules and objectives created from populations A and C above we can create the following population:

Population D
Inheritance:
(Population A, Rules)
(Population C, Objectives)

The resulting population would have all rules and objectives without writing another piece of code. As shown in Figure 1.

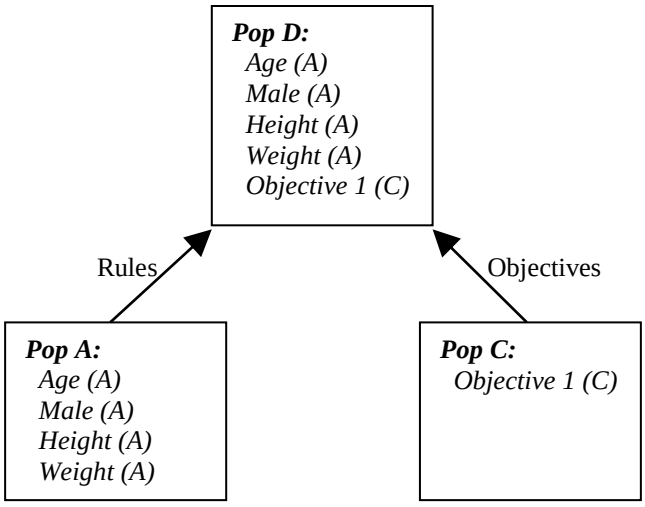

Figure 1. Population D inheritance tree – including rules, objectives, and source population in parenthesis.

Consider, however, that we wish to replace *Age* in this population after generation with the *Age* in population B. We use inheritance in the following manner:

Population E
Inheritance:
(Population D, Data)
(Population B, Rules)

Note that inheritance order here matters since *Age* rules in population B override the *Age* data generated in population D. And note that data is passed for *Male, Age, Height,* and *Weight* so inheriting population E will have those elements in it, yet those will all be represented as tables with data. See Figure 2.

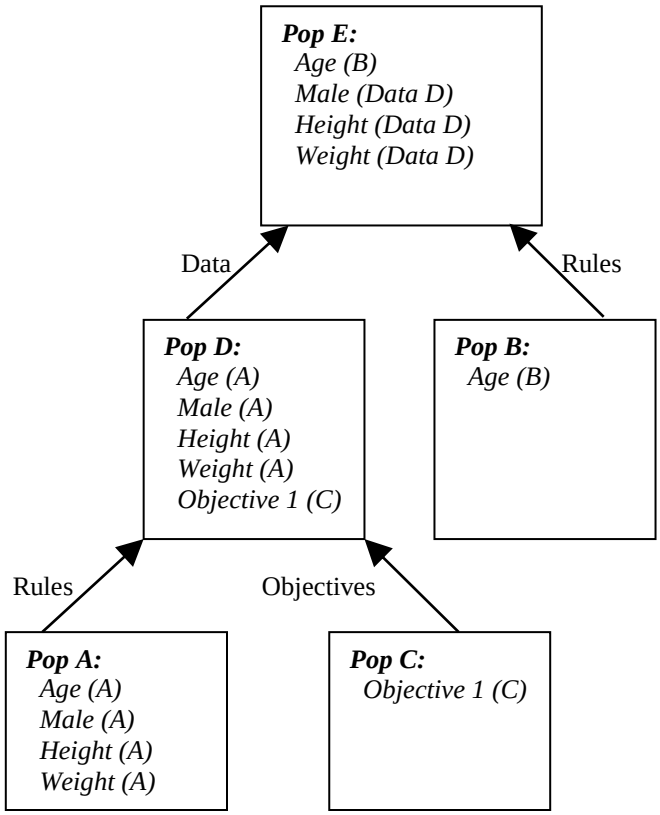

Figure 2. Population E inheritance tree – including rules, objectives, and source population in parenthesis.

Another simple example would be building a population that inherits default values and overrides other values. Consider that we wish to model the US population and know its death rate according to information in (Wikipedia Online A). We can create a population with the Rule that holds the Crude *DeathRate* per 1000 table column from 1990. And by default set the *Year* to 2000. See Population F.

Population F
Rules:
DeathRate ~ Table([[Year, [NaN, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012]]] , [8.1, 7.7, 7.4, 6.7, 6.4, 6, 5.9, 5.7, 5.8, 5.6, 5.9, 5.6, 5.5, 5.5, 5.9, 5.7, 6.2, 6.3, 5.9, 5.6, 5, 4.6, 4.5])/1000
Year ~ 2000

Now we can merge population F with population D above yet indicate that the year would be 1990 instead of the default set in F. This new population G will therefore be:

Population G
Inheritance:
(Population F, Rules)
(Population D, Rules)
(Population D, Objectives)
Rules:
Year ~ 1990

See Figure 3.

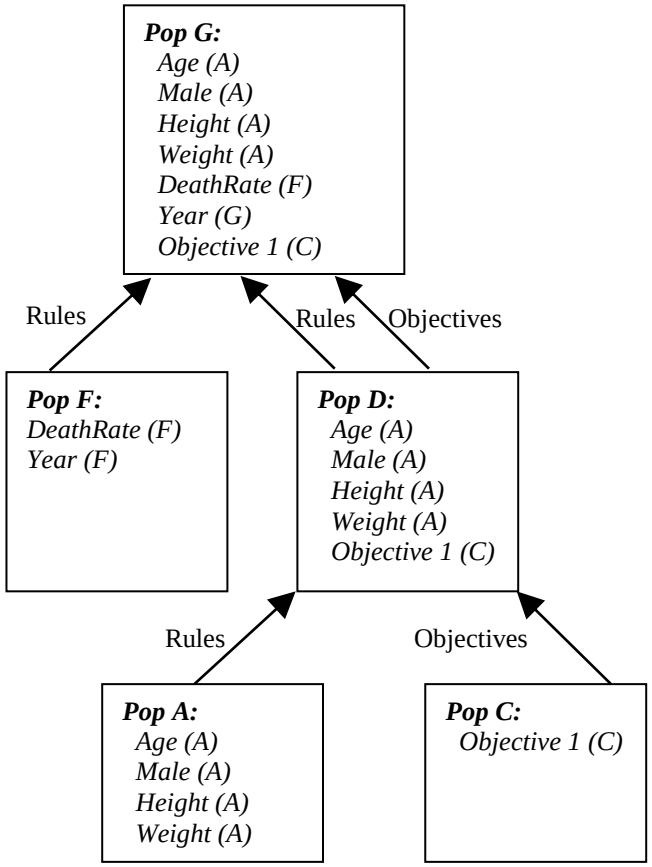

Figure 3. Population G inheritance tree – including rules, objectives, and source population in parenthesis.

Note that population D is inherited twice, once for rules and once for objectives. In this example the inheritance order in the inheritance list does not change the final outcome since the rules defined in the inheriting population are always applied last and therefore the year used to determine death rate would will be: 1990.

Assume that we have an alternative population with a death rate defined by a function that depends on *Age* rather than *Year* that is defined as:

Population H
Rules:
DeathRate ~ Max(0,Min(1,Age/120))

We can now easily reconstruct a population that uses a different death rate by replacing the building blocks yet reconstruct population G otherwise. Population I demonstrates this:

Population I
Inheritance:
(Population H, Rules)
(Population D, Rules)
(Population D, Objectives)
Rules:
Year ~ 1995

Figure 4 shows the inheritance structure of population I and can be compared for Figure 3 that shows how population G is constructed.

The formal difference between populations G and I is only using a different computational building block. These are both variations exchanging different bases for computation.

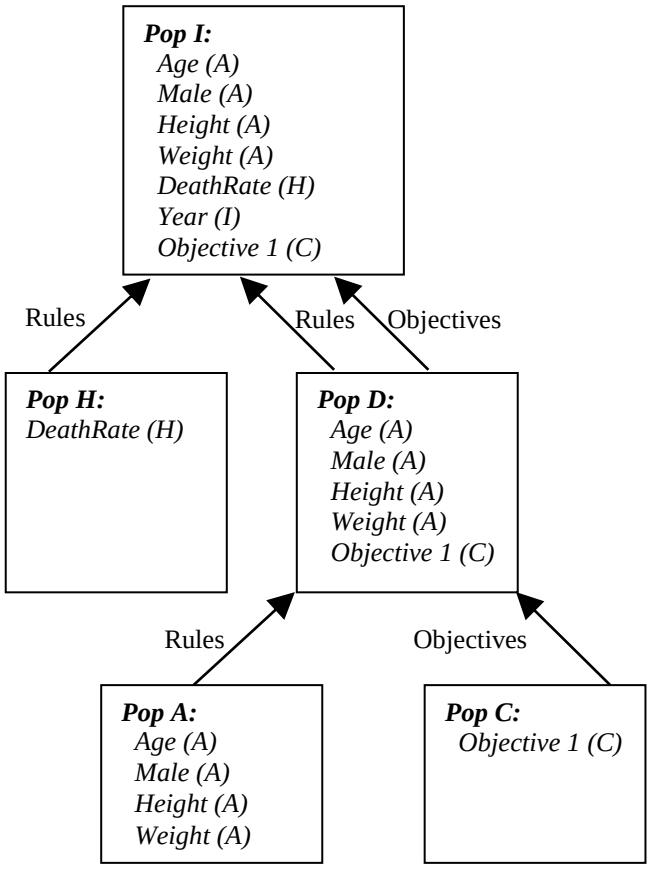

Figure 4. Population I inheritance tree – including rules, objectives, and source population in parenthesis.

DISCUSSION

The ability to define, reuse, and exchange computational population generation building blocks provides important flexibility. The Reference Model for disease progression is the main motivation behind this technology. The Reference Model populations are extracted from literature statistics and contain repetitions and missing data, and therefore these populations require defaults. The number of populations in the model is growing and each population includes many parameters, rules, and objectives. Therefore an object oriented approach is required to humanly manage the growing number of cohorts stored in the system. At the same time it is an important mechanism that drives variation generation in the model.

The Reference Model uses variations in model hypotheses that are tested against known data. This modeling methodology imitates the current trend of Evidence Based Medicine. Moreover it is at the heart of the scientific method where experiments are made to test hypotheses. In this case the experiments are computational within a virtual laboratory powered by HPC.

The use of HPC allows testing many more hypotheses and the ability to easily formulate them as interchangeable building blocks is essential. Being able to generate populations allows us to test our understanding about unknown initial condition hypotheses that were not reported. This is common in medical literature yet can be applicable to other fields.

Although current modeling use does not require this feature, data inheritance was added to this generation framework. This allows future developers to take this framework in different directions. For example using data inheritance opens the opportunity for incorporating and cleaning up data arriving from EMR. This is accomplished by replacing missing values with formulas or filtering out records to meet certain criteria at the individual or population level. This flexibility is essential for future development that can include future inheritance types, variations of interactions between data and objectives, and data specific operations.

Despite this theory being abstract, it is driven by the need for modular modeling – the ability to generate interchangeable building blocks of interconnected data for simulation purposes. Consider that the output of this generation is a table with numbers where each row is a record of an entity and the columns are user defined; there can be many other uses other than disease modeling for this technology.

In this context, it may be interesting to mention two projects. SymPy (SymPy Development Team, Online) which provides symbolic math tools and PyMC (PyMC @ Github, Online) which provide advanced Monte Carlo Tools. Some of the concepts in this paper may be useful to consider within those scopes for possible extensions. In any case, the concepts of this paper are important step towards modular modeling of populations.

ACKNOWLEDGMENTS

The base theory upon which this paper builds was a GPL disease modeling framework that was initially supported by the Biostatistics and Economic Modeling Core of the MDRTC (P60DK020572) and by the Methods and Measurement Core of the MCDTR (P30DK092926), both funded by the National Institute of Diabetes and Digestive and Kidney Diseases. The modeling framework was initially defined as GPL and was funded by Chronic Disease Modeling for Clinical Research Innovations grant (R21DK075077) from the same institute. The current version of MIST to which the developments in this paper pertain was developed from this GPL framework without financial support.
The author wishes to thank Aaron Garrett who developed the Inspyred library that enabled defining objectives. The author also wished to thank the following individuals whose support and ideas initialized and allowed this project: Deanna J.M. Isaman, Morton Brown, and William H. Herman. Thanks to Philip Standiford for proof reading.

ACKNOWLEDGEMENTS

The acknowledgements section is optional. If included, it appears after the main body of text and before the references. This section includes acknowledgements of help from associates, credits to sponsoring agencies, etc. Please try to limit acknowledgements to no more than a three or four sentence paragraph.

REFERENCES

Barhak J., Isaman D.J.M., Ye W., Lee D. (2010), Chronic disease modeling and simulation software, Journal of Biomedical Informatics, 43(5) 791-799, http://dx.doi.org/10.1016/j.jbi.2010.06.003

Barhak J. (2012A), The Reference Model in the Mount Hood #6-2012 validation challenge and the uncertainty challenge. The Mt hood challenge 6, June 7-8, 2012. Johns Hopkins Mt. Washington Conference Center.

Barhak J. (2012B), The Reference Model for Disease Progression. SciPy 2012, Austin Tx, 18-19 July 2012. Paper: https://github.com/Jacob-Barhak/scipy_proceedings/blob/2012/papers/Jacob_Barhak/TheReferenceModelSciPy2012.rst
Poster:
http://sites.google.com/site/jacobbarhak/home/PosterTheReferenceModel_SciPy2012_Submit_2012_07_14.pdf

Barhak J. (2012C) , The Reference Model for Chronic Disease Progression. 2012 Multiscale Modeling (MSM) Consortium Meeting, October 22-23, 2012,

Poster:
http://sites.google.com/site/jacobbarhak/home/PosterTheReferenceModel_IMAGE_MSM_Submit_2012_10_17.pdf

Barhak J. (2012D), The Reference Model: Improvement in Treatment Through Time in Diabetic Populations, The Fourth International Conference in Computational Surgery and Dual Training. The Joseph B. Martin Conference Center at Harvard Medical School. Boston, MA, USA. December 9-10-11, 2012. Presentation: http://sites.google.com/site/ComputationalSurgery_Presneted_2012_12_LateUploadToOwnWebSite_2014_2_27.pptx

Barhak J. (2012E), The Reference Model: Improvement in Treatment Through Time in Diabetic Populations, The Fourth International Conference in Computational Surgery and Dual Training. The Joseph B. Martin Conference Center at Harvard Medical School. Boston, MA, USA. December 9-10-11, 2012. Presentation Slides: http://sites.google.com/site/ComputationalSurgery_Presneted_2012_12_LateUploadToOwnWebSite_2014_2_27.pptx

Barhak J., Leff H.S., (2013A), Modeling a Chronic Disease Model and a Mental Health Model Using the Same Modeling Tools, MODSIM World 2013, April 30 - May 2nd, Hampton Roads Convention Center in Hampton, VA.
Paper: http://sites.google.com/site/jacobbarhak/home/MODSIM_World2013_Submitted_04Apr2013.pdf
Presentation: http://sites.google.com/site/jacobbarhak/home/MODSIM_World_Presented_2013_05_2.pptx

Barhak J. (2013B), The Reference Model Scores Fitness of Models and Populations. Poster Presentation. ISPOR 18th Annual International Meeting, May 18-22, 2013, Sheraton New Orleans, New Orleans, LA, USA. Poster: http://sites.google.com/site/jacobbarhak/home/PosterTheReferenceModel_ISPOR_Submit_2013_05_14.pdf

Barhak J. (2013C), MIST: Micro-Simulation Tool to Support Disease Modeling. SciPy, 2013, Bioinformatics track, https://github.com/scipy/scipy2013_talks/tree/master/talks/jacob_barhak,
Presentation: http://sites.google.com/site/jacobbarhak/home/SciPy2013_MIST_Presented_2013_06_26.pptx
Video: http://www.youtube.com/watch?v=AD896WakR94

Barhak J (2013D), The Reference Model for Disease Progression Sensitivity to Bio-Marker Correlation in Base Population - The Reference Model Runs with MIST Over the Cloud! 2013 IMAG Multiscale Modeling (MSM) Consortium Meeting, October 2-3, 2013,
Poster:
http://sites.google.com/site/jacobbarhak/home/PosterTheReferenceModel_IMAG_MSM2013_Submit_2013_09_23.pdf

Barhak J., Garrett A. (2014A), Population Generation from Statistics Using Genetic Algorithms with MIST + INSPYRED. MODSIM World 2014, VA.
Paper:
http://sites.google.com/site/jacobbarhak/home/MODSIM2014_MIST_INSPYRED_Paper_Submit_2014_03_10.pdf
Presentation: http://sites.google.com/site/jacobbarhak/home/MODSIM_World_2014_Submit_2014_04_11.pptx

Barhak J. (2014B), The Reference Model for Disease Progression uses MIST to find data fitness. PyData Silicon Valley 2014: Abstract: http://pydata.org/sv2014/abstracts/#195
Presentation: http://sites.google.com/site/jacobbarhak/home/PyData_SV_2014_Upload_2014_05_02.pptx
Video: https://www.youtube.com/watch?v=vyvxiljc5vA

Barhak J. (2014C), The Mount Hood Diabetes Challenge 2014. The Reference Model for Disease Progression - participated in the challenge, Stanford.
Presentation:
http://sites.google.com/site/jacobbarhak/home/MH2014TheReferenceModel_Submit_2014_06_15.pptx

Barhak J., (2014D) The Reference Model for Disease Progression – Data Quality Control. SummerSim 2014, Monterey CA.
Presentation: http://sites.google.com/site/jacobbarhak/home/SummerSim2014_Upload_2014_07_06.pptx

Barhak J @ Github - MIcro Simulation Tool (MIST) (Online), https://github.com/Jacob-Barhak/MIST

Hayes A. J., Leal J., Gray A. M., Holman R. R., Clarke P. M., (2013) UKPDS Outcomes Model 2: a new version of a model to simulate lifetime health outcomes of patients with type 2 diabetes mellitus using data from the 30 year United Kingdom Prospective Diabetes Study: UKPDS 82. Diabetologia 56:1925–1933, http://dx.doi.org/10.1007/s00125-013-2940-y

Johnson R., McNutt P., MacMahon S., Robson R. (1997), Use of the Friedewald Formula to Estimate LDL-Cholesterol in Patients with Chronic Renal Failure on Dialysis, Technical Briefs, Clinical Chemistry 43, No. 11, 1997, P. 2183

Lutz M, Programming Python, 3rd Edition (2006). http://www.rmi.net/~lutz/about-pp3e.html

Moore G. E (1965).: "Cramming more components onto integrated circuits". Electronics, Volume 38, Number 8, April 19th, 1965.
Moore's Law – Wikipedia, http://en.wikipedia.org/wiki/Moore's_law . Accessed 15-Oct-2014

PyMC @ Github PyMC 3 on Github (Online). https://github.com/pymc-devs/pymc/tree/pymc3 . Accessed 16-Oct-2014

Python Software Foundation (Online) The Python Tutorial, https://docs.python.org/2/tutorial/ Accessed 18-Feb-2015

SymPy Development Team (Online) SymPy. http://sympy.org/en/index.html Accessed 16-Oct-2014

Tolk A, Diallo S Y, Padilla J J, Herencia-Zapana H (2013), Reference modelling in support of M&S—foundations and applications, Journal of Simulation 7, 69–82. http://dx.doi.org/10.1057/jos.2013.3

Wikipedia (Online A), Demographics of the United States ,
http://en.m.wikipedia.org/wiki/Demographics_of_the_United_States Accessed 16-Oct-2014

Wikipedia (Online B), Depth-first search, http://en.m.wikipedia.org/wiki/Depth-first_search, Accessed 16-Oct-2014

Wikipedia (Online C) Object-oriented programming , http://en.wikipedia.org/wiki/Object-oriented_programming, Accessed 16-Oct-2014