A micro-simulation platform for estimating future household demand for energy and water, given changes in poverty and inequality

DRAFT VERSION

Rafael Guerreiro Osorio
Instituto de Pesquisa Econômica Aplicada
International Policy Centre for Inclusive Growth

ABSTRACT

In this paper, a micro-simulation platform devised as a tool for integrated SDG-based development planning is described. The platform allows planners to quickly draw future scenarios for the domestic demand for energy and water, driven by changes in the income distribution, consumption patterns and household behaviour. It can be used as a standalone tool or in integration with other tools, such as the CLEWS environment model and/or economy wide models. The platform is also scalable: simple, nonetheless informative simulations can be produced with a very restricted set of variables, broadly available in developing countries' general purpose or income and expenditure surveys. The complexity of simulations can increase with data availability and skill level of users. It is designed to be user friendly, and was fully implemented in cross-platform free and open source software, using the python language and its scientific computing libraries.

INTRODUCTION

In 2015, the nations of the world came to an agreement regarding the overarching strategy for global development up to 2030, materialized in the 17 Sustainable Development Goals. The SDGs are the offspring of the marriage between two powerful ideas, that of human and that of sustainable development. It seems to be just a coincidence, but the fact that 17 is a prime number adds something to the symbolism of the SDGs: they are indivisible but by the whole, which makes them one, or by one, which makes them whole.

Despite all progress made towards the Millennium Development Goals, predecessors of the SDGs, still many in the world are severely income deprived, and a significant many other do not have access to clean and efficient energy sources, to water, and might not even have a large enough intake to cope with basic caloric needs. According to the World Development Indicators, almost one billion human beings were living under US$ 1.9 a day in 2012 (World Bank, 2015). Everywhere, income deprivation, lack of access to energy and water, hunger and malnutrition are strongly correlated. And the cumulation of maladies is inclusive of other problems tackled by the SDGs as well, such as gender, ethnic and racial inequalities *etc.*

Were it not for their lack of access and severe budget constraints, the "bottom billion" (Collier, 2007) would likely increase their energy, water and food consumption. And not only them, but also large shares of the population of middle income countries that recently ascended from poverty, but are still far from the top, are eager for more, for beef, smart phones, flat screen TVs, new homes, cars and travel, and if their incomes keep growing, they will seek to fulfil these desires. However, as Howells *et al.* aptly summarized "*land, energy and water are our most precious resources, but the manner and extent to which they are exploited contributes to climate change. Meanwhile, the systems that provide these resources are themselves highly vulnerable to changes in climate*" (201X). Therefore, balancing human development and sustainability is a great challenge

Human development is the process of enlarging people's freedom to make choices, as well as enlarging the very set of choices that allow them to, quoting the Human Development Report Office, "*live the lives they value*". The HDRO emphasizes it's all about choice, "*about providing people with opportunities, not insisting that they make use of them. No one can guarantee human happiness, and the choices people make are their own concern. The process of development – human development - should at least create an environment for people, individually and collectively, to develop to their full potential and to have a reasonable chance of leading productive and creative lives that they value*" (HDRO, 2016).

In the short term, for a large share of the world population, living the lives they value encompass the adoption of consumption patterns that increase the demand for energy, water and land, when the systems that provide those resources are already under excessive pressure. Of course, consumption patterns can and do change, and for better, even as a result of human development. But even with incentives in place change is slow, and can not be imposed. Therefore, one of the great challenges for SDG-based development planning is to assure that human development is sustainable, and, concomitantly, that sustainability won't hinder human development.

Choosing electricity as an example, it is reasonable to expect that as more households connect to the grid and their income rise, household demand for electricity will grow. To cope with it, the electricity supply will have to increase. But electricity generation can be of serious consequence to the environment and contributes to global warming. Though at the household all those things electric might not be contributing greenhouse gas to global warming, somewhere else the generation of electricity can be exerting strong pressure on the environment. For there is no way to generate and transmit electricity without exerting some pressure.

But some energy sources of electricity are cleaner than others, and environmental impact can be reduced by investing in research and adoption of technologies that further the efficiency of generation and transmission. Also, the adoption of more progressive tariffs, rising prices of electricity for consumers with high demand levels, might incentive reduction in the consumption of the rich, making way for that of the poor. And electricity subsidies for the poor can be dosed to prevent waste among them. With good planning, social, economic and environmental variables considered, maybe everyone could afford to get the amount of electricity required to live the life one values, without waste and additional pressure on the environment.

Thus, *people centred and environment friendly development requires policy coherence and integrated planning*. But there is much more to it than simply taming the consumption of the rich to make room for the poor. Think, for instance, of a hypothetical country where just half of the population has access to electricity, with plans to expand it to three quarters. The country has two regions where access to electricity is well below average, however in the one, favoured by a mountainous landscape, most households have access to water; and in the other, water sources are below the altitude of the houses. In both electricity will foster human development and increase well-being. However, in the latter, with electricity come water pumps, which will ease the life of women, who were responsible for fetching water. Should this second region be prioritized in some way in the plans, the expansion of access to electricity could create an enabling environment for the expansion of access to water, for reduction of the gender burden and inequality, and it could even, be positive for the education of household children, as mothers would get more time to be with them.

THE MICRO-SIMULATION PLATFORM

The micro-simulation platform allows planners to draw future scenarios for the domestic demand for energy and water, driven by changes in the income distribution, consumption patterns and household behaviour. The micro-simulation platform is structured in three components: *1) a population projection; 2) a resource – energy or water – access model; and 3) a micro-simulation dataset*. Its pillar is the micro-simulation dataset, built from a primary source, with people and household data, such as a census, a general purpose or an income and expenditure household survey.

The **population projection** is treated as a separate component because projections are widely available. The micro-simulation tool includes the latest edition of the World Population Prospects (DESA, 2015), with projections for the world population, by region and country, under different population growth hypotheses, up to 2100. However, any other population projection can be used. In some situations, the population projection can be based on the micro-simulation dataset. The population projection informs the simulation about the size and the distribution of population by groups, from the baseline year to the end year.

The **resource access model** is a projection of how relative access to a given resource among the population will grow. The resource can be energy – as consumed by the household, electricity, gas, wood *etc.* – or water. The resource access model can be developed from the microsimulation dataset, can be purely theoretical, or it can come from other source. It informs the simulation about the relative size and distribution of access to resource, from the baseline year to the end year.

The **micro-simulation dataset** is based on a general purpose or an income and expenditure household survey, and can have as many variables as desired or needed for posterior analyses. The dataset requires only the following basic variables: *1) household per capita income, 2) household per capita consumption of a resource (electricity, gas, water etc.); 3) household weight and population weight*. Those are the seed variables.

Many household surveys collect data on sources of individual income such as labour earnings, property, public or private pensions, unemployment insurance, targeted cash transfers, *etc.* **Household *per capita* income**, ***X***, is the sum of all ***k*** sources of income, for all ***j*** individuals in household ***i***, divided by household size ***J*** (number of individuals):

$$X_i = \frac{1}{J} \sum_{j=1} \sum_{k=1} X_{jk}$$

A useful step after the creation of the income variable is to use a national consumer price index to adjust its values to a meaningful reference date – for instance, to 31st December 2011 to apply the purchase power parity factor for private consumption to convert the values to US$ PPP 2011.

A common difficulty for calculating the **household *per capita* consumption of a resource**, such as electricity, gas or water, is that in many countries regular surveys investigate the amount paid, but not actual consumption in the relevant unit, such as watts per hour for electricity or cubic meters for gas and water. This is a problem because the amount paid is not necessarily perfectly correlated to the quantity consumed. Progressive block tariffs of water and electricity, for instance, are common, making the price per cubic meter of water or Kwh of energy higher for higher levels of consumption. That is, the income elasticity of the demand for a resource can not be directly estimated substituting expenditure for consumption.

The ideal situation is the existence of a household income and expenditure survey that register the quantities consumed besides the amount paid for energy and water in a given period. But even if only expenditure on energy and water is available, if there is enough information on the tariff scheme – including its within country variations – it might be possible to estimate consumption, in the relevant measurement unit, from expenditure.

Once the resource consumption variable is created, it is divided by household size yielding household *per capita* resource consumption, the aggregate household demand for ***Y***. To check the accuracy of the obtained variables, their population expanded and annualized statistics can be compared to those of a National Energy or Water Report bearing data on the total consumption of households in the same (or a close) year of the primary source of the micro-simulation dataset.

$$Y_i = \frac{KWh_i}{J} \vee \frac{m_i^3}{J} \vee \frac{kg_i}{J} (\ldots)$$

General purpose and income and expenditure surveys usually have complex multi-stage samples – the sampling units are not independent and have different probabilities of being selected – but frequently there is not enough data or documentation on the sample besides the inverse probability weights used to expand the sample to a population projection, for the whole country or for some population strata. The use of these weights is mandatory to get the population point estimates from the sample. Household and individual inverse probability weights are almost always included in the survey datasets. The household population weight ***P*** is obtained by the summation of the inverse probability weights of all household members.

$$P_i = \sum_{j=1} w_j$$

And the population size ***N*** is the sum of the population weights of the households.

$$N = \sum_{i=1} P_i$$

It is important to note that although in many surveys the household weight can be obtained by dividing the household population weight by the household size, this is not always valid. The survey, for instance, could have been post stratified by sex and age, and the weights of household members vary. When household members have different weights, the household weight ***w*** is equal to the inverse probability weight of the household head, or the weight variable explicitly assigned to households by the Statistics Office responsible for the survey.

The microsimulation dataset might have another population projection than the one being used in the simulations, and also a different level of access to a resource. Its output to the microsimulation platform are new income and resource consumption distributions. These new distributions are produced given income growth patterns, and income elasticity of the demand patterns (consumption) for the resource (details ahead).

$$X_{i(t+1)} = X_{it}\left(1 + g_{i(t:t+1)}\right)$$

$$Y_{i(t+1)} = Y_{it}\left(1 + g_{i(t:t+1)} \times \epsilon_{i(t:t+1)}\right)$$

Based on the new income and resource consumption variables, the microsimulation dataset informs the microsimulation about the average consumption, considering only the households that have consumption greater than zero: zero consumption is interpreted as no access.

Each of the components of the micro-simulation platform thus will yield for the simulation period, from the baseline to the last year, a piece of data. Multiplying the population projection, by the level of access to resource in that population, and by the average consumption of those with access, the user can get an estimate of the future demand for energy and water based on scenarios of changes in the income distribution. For example, a population projection estimates 10 million inhabitants in Lilliput in 2030 – component 1 – and given the expansion plans of the ministry of Energy, 85% of the households will be connected to the grid – component 2 – and the average consumption of connected households, given future levels of income, is estimated at 60KWh per capita monthly – component 3. Then the total demand of the household sector in 2030 will be 6,120GWh (10,000,000 * 0.85 * 60 * 12 = 6,120,000,000).

USING THE MICRO-SIMULATION PLATFORM

A basic micro-simulation exercise of future household demand for electricity was devised to demonstrate the potential of the tool. The Republic of Aztlan was chosen as an example, among other things because of the availability of data, and simplicity, for there is no need to develop a RAM (resource access model), as access to electricity is universal. The population projection used in this demonstration is the medium-variant series of the United Nations Population Division (WPP, 2015). As for Aztlan the population projection is done and there is no need of a RAM, we can focus the example on the development of the simulation dataset. In this exercise, electricity prices do not change, and the demand for electricity is driven by household *per capita* income.

To run a micro-simulation, **python 2.7** has to be installed as well as the libraries **numpy**, **scipy**, **pandas**, **matplotlib**, **patsy** and **statsmodels**. In Linux distributions python is usually included and the additional libraries can be installed with the package manager. For Windows and Mac users there are distributions, e.g. *anaconda* and *python-xy*, that come with python and the required libraries.

Unzip the micro-simulation platform files, then open a terminal (command line), and change directory to where the files are. Then type python and press enter to start the interpreter.

```
user@user-desktop:~/Documents/ipcmicrosimplatform > python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To use the platform, its module has to be imported.

```
>>> from ipc_microsim_tool import IPCmicrosimTool as imt
```

Now we proceed to create the three components, the population projection, the resource access model and the microsimulation dataset. They are made available as objects in ***IPCmicrosimTool***.

population()

The ***population()*** object is very simple and takes no arguments. It has the following attributes: ***projection***, a dataset indexed by year containing population estimates; ***country***, the name of the country or region of the estimates; ***description***, a string with a description of the projection.

The micro-simulation platform includes two data files with the population estimates and projections of the World Population Prospects (DESA, 2015): *WPP-ESTIMATES-1950-2015.tab.txt*, and *WPP-PROJECTIONS-2015-2100.tab.txt*. A projection for a country can be retrieved using the ***get_WPP_projection()*** method; and the list of regions and countries for which there are projections can be retrieved using the ***get_WPP_countries()*** method.

Create a population object – at the beginning, attributes are empty (None).

```
>>> pop = imt.population()
>>> print pop.projection
None
>>> print pop.country
None
>>> print pop.description
None
```

The attributes can be set directly by the user. The population projection dataset must be a *pandas DataFrame* with rows indexed by years, and the population projection must be in the first column. Also, there should be estimates for all years of the simulation period.

population.get_WPP_countries(beginswith)

The WPP estimates and projection variants are available not only for countries, but also for regions and other groupings. To retrieve a list of countries and regions, the ***get_WPP_countries()*** method can be used.

```
>>> pop.get_WPP_countries()
array(['WORLD', 'More developed regions', 'Less developed regions',
(...)
       'Wallis and Futuna Islands'], dtype=object)
```

This method can be given a prefix to restrict the list – for instance, to only countries which name begins with *ma*.

```
>>> pop.get_WPP_countries('ma')
['Madagascar', 'Malawi', 'Aztlan', 'Mayotte', 'Mali', 'Mauritania', 'Maldives',
'Malaysia', 'Malta', 'Martinique', 'Marshall Islands']
```

population.get_WPP_projection(country, variant)

Here the attributes of the population object will be set by the ***get_WPP_projection()*** method. This method requires the country or region name – as in the WPP dataset – and the projection variant. The projection variant can be specified by the index or by the name, and the options are: 0 'Low variant'; 1 'Medium variant'; 2 'High variant'; 3 'Constant-fertility'; 4 'Instant-replacement'; 5 'Zero-migration'; 6 'Constant-mortality'; 7 'No change'.

```
>>> pop.get_WPP_projection('Aztlan', 1)
'Aztlan - Medium variant'
```

Now the population object is ready to be used in micro-simulations. Notice that the projection attribute is a *pandas DataFrame*, so methods like head() and tail() can be used to inspect the beginning and the end of the dataset.

```
>>> pop.projection.head()
      Aztlan
1950     493254
1951     506431
1952     521190
1953     537048
1954     553623

[5 rows x 1 columns]
>>> pop.projection.tail()
      Aztlan
2096     972115
2097     966812
2098     961666
2099     956691
2100     951897

[5 rows x 1 columns]
>>> pop.country
'Aztlan'
>>> pop.description
'WPP 2015 - Medium variant'
```

CREATING A RAM OBJECT - COMPONENT 2

resource_access()

The Resource Access Model object, ***resource_access***, is also very simple. It has the following attributes: ***RAM***, a dataset indexed by year containing access rates; ***period***, a tuple with the baseline and the last year of the micro-simulation; ***description***, a descriptive string.

Each access has its RAM, and in this first version, there are no methods to fill the attributes of a RAM object. In Aztlan, access to electricity is 100%, and the simulation will be done for the 2014-2030 period. The RAM object is created interactively.

```
>>> ram = imt.resource_access()
>>> ram.period = (2014, 2030)
>>> ram.description = 'Aztlan - electricity'
>>> ram.RAM = pd.DataFrame({'access_ele': 1.0}, range(ram.period[0], ram.period[1]+1))
>>> ram.RAM.head()
      access_ele
2014           1
2015           1
2016           1
2017           1
2018           1

[5 rows x 1 columns]
>>> ram.RAM.tail()
      access_ele
2026           1
2027           1
2028           1
2029           1
2030           1

[5 rows x 1 columns]
```

CREATING THE MICROSIMULATION OBJECT – COMPONENT 3

The seed dataset

The starting point is the creation of a dataset with the seed variables: 1) household *per capita* income, 2) household *per capita* electricity consumption, 3) household weight and population weight. In this exercise, these variables were obtained from the 2014 Continuous Multi Purpose Household Survey (CMPHS) and the national Consumer Price Index (Comparative monthly CPI, 1980 – 2015), both by Statistics Aztlan. The CMPHS has a nationally representative sample with a rotative panel design, a fixed number of households being interviewed every month. The dataset received, did not have the panel identifiers, and was prepared to be used as a cross-sectional data source, aggregating all 2014 interviews, with inverse probability weights adding up to a national population projection. The dataset had no variables with data on sample design (such as strata or cluster identifiers). Data from previous rounds of the CMPHS, from 2007 to 2013 were also deployed as source of parameters for the simulation.

As the CMPHS data is not in the public domain, the dataset construction will be briefly described. All other variables are calculated from the seed variables.

Household *per capita* income

The CMPHS collects total income in three categories: labor earnings (wage workers, self-employes, and inkind); property income (land, rent and other returns to producrtive assets); transfers (public and private pensions, targeted cash transfers etc.). Household *per capita* income was calculated using [1]. Then it was deflated to December 2011 prices using the CPI. This was done taking the month of the interview into account. Therefore, for interviews done in January, the deflator is the ratio between the CPI in January 2014 and December 2011, and so forth. After adjusting the values to constant prices, they are converted to international dollars by the 2011 Purchasing Power Parity (PPP) factor for private consumption (World Bank, 2015 – WDI). Finally, the monthly value is multiplied by 12 and divided by 365, yielding household *per capita* income in US$ PPP 2011/day.

Household *per capita* electricity consumption

Electricity consumption is not investigated by the CMPHS, only monthly expenditure with electricity. This is a common difficulty, as in many countries regular surveys investigate the amount paid, but not actual consumption in KWh. And as many countries have a progressive tariff for electricity, the amount paid is not perfectly correlated to the quantity consumed. That is, the income elasticity of electricity consumption can not be directly estimated substituting expenditure for consumption.

The solution for this challenge is to estimate consumption from expenditure, something which requires information on the tariff scheme. This can turn out to be quite complicated for countries that have more than one electricity company and heterogeneous tariffs. Fortunately, in Aztlan the Central Electricity Board defines a national tariff scheme for domestic connections. With such information it is possible to "reverse engineer" expenditure to get consumption. However, in Aztlan, the electricity bill charges not only electricity consumption, but also the meter rent, and a fee that is used to fund the public Aztlan Broadcast Corporation (MBC). Low electricity consumption households (up to 33 KWh) make a smaller contribution to MBC, Rs$ 20; and the standard value is Rs$ 150. Meter rent varies with the type and capacity of the connection: monophase, Rs$ 10; triphase, low or high load, Rs$ 60. There is also a minimum monthly charge for each connection type (Rs$ 44, 184 and 369).

But the CMPHS does not have data on connection type and meter rent, neither on the amount paid for MBC; and the expenditure with these items must be deducted from the amount paid before converting electricity expenditure into consumption using the tariff scheme. To do this, the following rules were adopted:

1) households with expenditure under Rs$ 44 (very few) were considered measurement errors (expenditure below minimum charge); for those it was assumed a monophase connection and consumption under 33 KWh (to define MBC fee), and Rs$ 30 were deducted from Rs$ 44 (minimum consumption, therefore, is 4.4 KWh);

2) households with expenditure under Rs$ 369; for those it was assumed a monophase connection and consumption under 33 KWh (to define MBC fee), and Rs$ 30 were deducted from reported expenditure;

3) households with expenditure Rs$ 369 and over; for those it was assumed a triphase high load connection and consumption greater than 33 KWh (to define MBC fee), and Rs$ 210 were deducted from reported expenditure.

After deducting MBC and meter rental from electricity expenditure, the tariff scheme is used to calculate household consumption in KWh and KWh *per capita*. If the tariff is Rs$ 3.16/KWh for the initial 25KWh, for households up to Rs$ 79, electricity consumption is just net electricity expenditure divided by Rs$ 3.16. Households with net electricity expenditure greater than Rs$ 79 receive 25KWh and Rs$ 79 is deducted from expenditure. The remaining expenditure is at higher prices – for the next 25KWh, Rs$ 4.38. If up to Rs$ 109.50 remained, consumption is somewhere in the 25 to 50KWh range: the remainder is divided by Rs$ 4.38 and added to 25 to get consumption. Households paying more than Rs$ 188.50 (79 + 109.50) have consumption greater than 50 KWh, so they get this amount. This is repeated until there is no remainder.

Once the electricity consumption variable is created, it is divided by household size yielding household *per capita* electricity consumption ***Y***. Part of the households in the sample did not report electricity expenditure. This was treated as a measurement error, because if access to electricity is universal and the minimum charge is Rs$ 44 every household should report at least this value. In such cases, electricity consumption was imputed using a linear model fitted for households with reported expenditure. The imputation model predicts electricity consumption based on a very restricted set of household attributes: age and sex of the household head, size (number of household members), and household *per capita* income.

To check the accuracy of the obtained variable, its annualized statistics were compared to those of the National Energy and Water Report (Statistics Aztlan, 2014). According to the Report, domestic consumption totaled 806 GWh in 2014; the population projection was 1,268,567 inhabitants (all with access); thus, annual per capita consumption of electricity was around 636 KWh/year. The constructed variable underestimates this quantity putting it at 563 KWh/year. Actually, this is pretty accurate, as in the real world national figures estimated from surveys seldom match those coming from registry. Nevertheless, in the final step of the simulation a correction was applied to the consumption variable – for Aztlan, multiplication by 1.13 (636/563) – under the assumption that *registry is the source of levels*; and the *survey is the source of relative distributions.*

Household and household population weight

The household population weight is obtained by the summation of the inverse probability weights of all household members.

The dataset thus prepared has four variables, hhkey, income, kwhpc and wgt, and was stored in file *example_Aztlan_seed.tab.txt.*

microsim(pandas.DataFrame)

To create a microsimulation object, an argument is required, a populated *pandas DataFrame* object. In this example, the sample tab delimited text file is read to a *pandas DataFrame*, which is used as argument to create an instance of ***microsim()***.

```
>>> FILE = 'example_Aztlan_seed.tab.txt'
>>> data = pd.read_csv(FILE, sep='\t', index_col=False, na_values='')
>>> ms = imt.microsim(data)
>>> ms.dataset.head()
   hhkey     income      kwhpc        wgt
0      1  17.814630   51.92513   89.10946
1      2  12.981110   68.00739  125.90400
2      3   8.770278   41.12547  100.48050
3      4  53.708080  118.10800   99.45376
4      5  13.566520   38.94385  131.70920
```

Now the micro-simulation object *ms* has a dataset, a copy of what was read from the file, and the variables in the data are listed as the ***seedvars*** attribute. If the ***__reset__()*** method is called, all variables not in the list of seed variables are dropped. Users can change the ***seedvars*** attribute to prevent variables from being dropped (it's just a list).

```
>>> ms.seedvars
['hhkey', 'income', 'kwhpc', 'wgt']
>>> ms.dataset['newvar'] = 1.5
>>> ms.dataset.head()
   hhkey     income      kwhpc        wgt  newvar
0      1  17.814630   51.92513   89.10946     1.5
1      2  12.981110   68.00739  125.90400     1.5
2      3   8.770278   41.12547  100.48050     1.5
3      4  53.708080  118.10800   99.45376     1.5
4      5  13.566520   38.94385  131.70920     1.5

[5 rows x 5 columns]
>>> ms.__reset__()
True
>>> ms.dataset.head()
   hhkey     income      kwhpc        wgt
0      1  17.814630   51.92513   89.10946
1      2  12.981110   68.00739  125.90400
2      3   8.770278   41.12547  100.48050
3      4  53.708080  118.10800   99.45376
4      5  13.566520   38.94385  131.70920

[5 rows x 4 columns]
```

After creating a micro-simulation object with seed variables, household *per capita* income, electricity consumption, and the weights, before running a simulation, income growth and income elasticity of demand patterns have to be added to the object. The platform is flexible to incorporate elasticities and growth patterns generated with other tools; nevertheless it comes bundled with an elasticity calculator and a growth pattern generator.

INCLUDING ELASTICITIES

The income elasticity of the demand for a resource – in the example, electricity – is the expected relative change in consumption (demand) given a relative change in income. The canonical way to do this is the estimation of a log-log linear model, where electricity consumption depends on income, thus $\boldsymbol{\beta}$ is the elasticity.

$$\ln(Y_i)=\alpha+\beta\ln(X_i)+\varepsilon_i$$

However, this yields a single number, the average elasticity of the household sector, and using it would deprive the micro-simulation of its most appealing characteristic – being able to substitute a distribution of elasticities for the average, to get a more feasible projection of future electricity consumption, given changes in the income distribution, not only in average income. For instance, it is reasonable to expect higher income elasticity for very poor households. Thus, every household should have its own elasticity, or, at least, the elasticity of its subgroup, regarding some classification criteria.

Obviously, it is not possible to run a regression model for a single household. To accomplish the task of creating a distribution of elasticities, many methodological approaches can be devised. It is possible to partition the households in homogeneous groups and calculate the elasticity of every group using the classic log-log regression.

The elasticity calculator of the micro-simulation object gets the elasticities from quantile regressions. The model specification is the same as that of the linear log-log regression, but the estimation method is different, yielding the conditional median (or any other quantile) of the income elasticity of demand. By using the calculator, each household gets the elasticity of its quantile of the *per capita* electricity consumption distribution.

microsim.elast_calc(key, Y, X, P, stub='', parts=100**)**

The method ***elast_calc()*** is used to create elasticities using the calculator and the micro-simulation dataset. It requires the following arguments: ***key***, name of the household key; ***Y***, the dependent variable, resource consumption; ***X***, the independent income variable; ***P***, the inverse probability weight; ***stub***, which will be suffixed to *elast* to create the name of the elasticity variable in the dataset; and ***parts***, which define the quantiles of the regressions – if parts equals 100, there will be one regression for each of the 99 percentiles. The larger the number of parts, the longer will take to calculate the elasticities.

```
>>> ms.elast_calc('hhkey', 'kwhpc', 'income', 'wgt',
...               stub='kwhpc', parts=10)

Elasticity calculator started - please be patient
Replicating observations, 11280 to 464778...
Fitting models...
Quantile        elasticity      se_elast        intercept       se_intercept
```

```
0.1     0.398640        0.001902        2.107992        0.004755
0.2     0.393800        0.001311        2.330012        0.003287
0.3     0.394674        0.001207        2.474201        0.003031
0.4     0.387523        0.001195        2.622783        0.003004
0.5     0.375112        0.001243        2.772856        0.003128
0.6     0.368875        0.001130        2.907804        0.002842
0.7     0.369336        0.001127        3.037118        0.002833
0.8     0.379222        0.001295        3.158919        0.003248
0.9     0.394856        0.001866        3.358308        0.004658
True
>>>
```

The elasticity calculator adds two variables to the microsimulation dataset, the elasticity of the households, conditional on their quantile of the resource distribution, as well as a quantile identifier. Both variables are added to the seed variables list (***seedvar*** attribute).

```
>>> ms.dataset.head()
   hhkey     income      kwhpc        wgt  quantkwhpc  elastkwhpc
0      1  17.814630   51.92513   89.10946         0.6    0.368875
1      2  12.981110   68.00739  125.90400         0.8    0.379222
2      3   8.770278   41.12547  100.48050         0.5    0.375112
3      4  53.708080  118.10800   99.45376         0.9    0.394856
4      5  13.566520   38.94385  131.70920         0.4    0.387523

[5 rows x 6 columns]
```

As the elasticities can take a long time to be calculated, the user might want to save the micro-simulation dataset after calculating, to skip this step in future simulations. As the micro-simulation dataset is just a *pandas DataFrame*, it can be exported to a tab delimited text file, or other file type.

```
>>> ms.dataset.to_csv('newfile.txt', '\t')
```

Let's start over and create a micro-simulation object based on a dataset with 999 different elasticities, previously calculated using ***elast_calc()***.

```
>>> FILE = 'example_Aztlan_w_elast.tab.txt'
>>> data = pd.read_csv(FILE, sep='\t', index_col=False, na_values='')
>>> ms = imt.microsim(data)
>>> ms.dataset.head()
   Unnamed: 0  hhkey     income      kwhpc        wgt  quantkwhpc  elastkwhpc
0           0      1  17.814630   51.92513   89.10946       0.696    0.370054
1           1      2  12.981110   68.00739  125.90400       0.839    0.394448
2           2      3   8.770278   41.12547  100.48050       0.537    0.371987
3           3      4  53.708080  118.10800   99.45376       0.970    0.426611
4           4      5  13.566520   38.94385  131.70920       0.494    0.375423
```

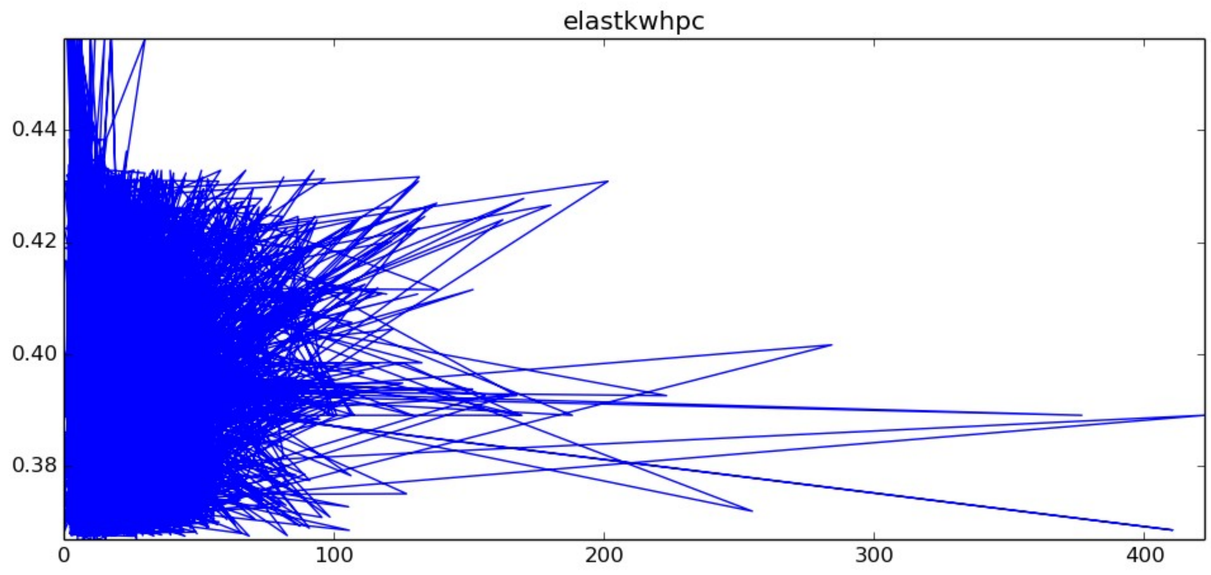
To inspect the elasticities – or any other variable – we can use standard *pandas* methods.

```
>>> ms.dataset['elastkwhpc'].describe()
count    11280.000000
mean         0.389651
std          0.015467
min          0.366907
25%          0.375566
50%          0.391726
75%          0.395861
max          0.456400
Name: elastkwhpc, dtype: float64
```

microsim.visualize(data, xcol, cols='all', subplotcols=2**)**

As for inspecting distributions visualizations are handy, the micro-simulation object has the ***visualize()*** method that makes simple scatter plots using matplotlib and a QT back-end. This method requires the arguments: ***data***, any pandas DataFrame; ***xcol***, name of the column to plot in the x axis; ***cols,*** list of columns to be plotted, defaults to all columns, each column will be plotted in its own subplot; ***subplotcols***, the number of columns to distribute subplots, defaults to two.

```
>>> ms.visualize(ms.dataset, 'income', 'elastkwhpc')
```



To get a better visualization, some aggregation is in place, and this can be done pretty easily with *pandas*. In this case, plotting the average elasticity for each one hundredth of the population sorted by income will solve the problem.

```
>>> msdts = ms.dataset.sort('income')
>>> msdts['cem'] = (((msdts.income * msdts.wgt).cumsum() / (msdts.income *
msdts.wgt).sum()) * 100).astype(int) + 1
>>> msdts.cem.tail()
78       99
1547    100
```

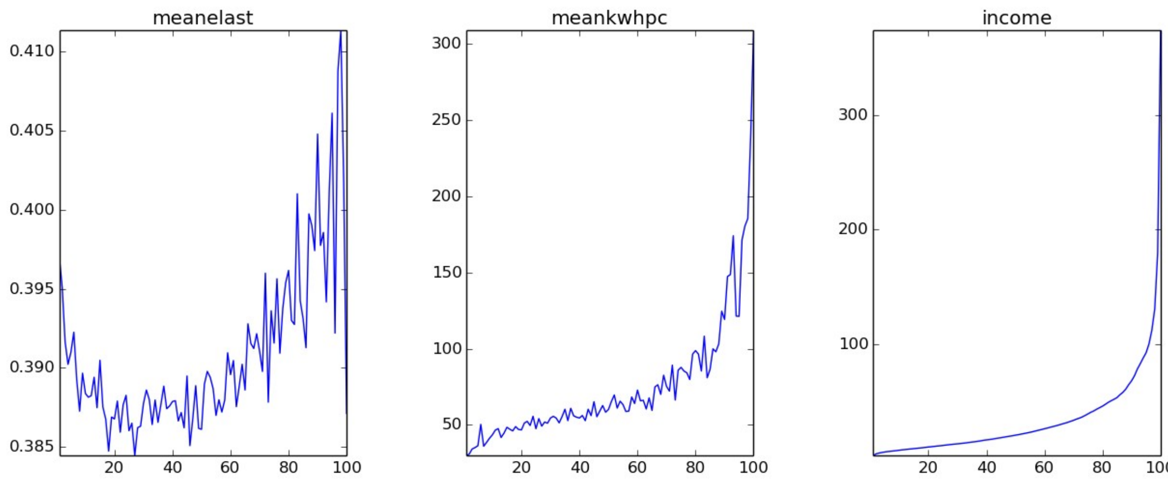
```
6239    100
6334    100
1624    101
Name: cem, dtype: int64
>>> msdts['cem'][-1:] = 100
>>> ms.dataset['cem'] = msdts['cem']
>>> ms.dataset[['hhkey', 'income', 'cem']].tail()
       hhkey     income  cem
11275  11276   7.543808   11
11276  11277   6.993824   10
11277  11278   8.457387   15
11278  11279   1.664910    1
11279  11280  25.944450   60

[5 rows x 3 columns]
```

Pandas has a very handy object to deal with statistics by groups, *groupby*. To get the averages of the elasticities, of income and consumption, by hundredths of the income distribution, a new *pandas DataFrame* is created from the aggregate object. Then the new dataset is visualized.

```
aggr = ms.dataset.groupby('cem')
>>> newdf = pd.DataFrame({'cem': aggr['cem'].first(), 'income': aggr['income'].mean(),
'meankwhpc': aggr['kwhpc'].mean(), 'meanelast': aggr['elastkwhpc'].mean()})
>>> newdf.tail()
     cem      income  meanelast   meankwhpc
cem
96    96   99.909811   0.392196  171.406078
97    97  111.999132   0.408738  180.477444
98    98  130.538153   0.411300  185.408340
99    99  179.279800   0.401745  240.303472
100  100  373.555550   0.387135  308.892708

[5 rows x 4 columns]
>>> ms.visualize(newdf, 'cem', subplotcols=3)
```



The new dataset could be exported to a field delimited text file, for instance, to produce better looking figures with publication quality.

THE GROWTH PATTERN GENERATOR

microsim.growth(parts=100, key='key'**)**

The microsimulation platform has a special object to store income growth patterns, the ***growth*** object, which is under the micro-simulation object. The ***growth*** object takes two arguments and has three attributes: ***parts***, the number of partitions of the income distribution for which there is a growth rate, defaults to 100; ***key***, the name of the key variable, defaults to 'key'; and ***dataset***, a *pandas DataFrame* to store the growth patterns.

An income growth pattern is nothing more than a collection of growth rates for partitions of the income distribution sorted in ascending order. If ***parts*** equals one hundred, the income distribution will be partitioned in 1% population bins, from the poorest 1% to the richest 1%, and to each a different growth rate will be applied.

The income growth pattern is pro-poor if it is regressive, that is, the growth rates of the poorest partitions are higher than those of the rich; and the greater the difference the higher is its pro-poorness. The income growth rate of a partition can be negative.

To create a growth object, the first step is to decide the number of partitions.

```
>>> grwt = ms.growth(1000)
>>> grwt.dataset.head()
   key
0    0
1    1
2    2
3    3
4    4

[5 rows x 1 columns]
```

Now there are many ways to add growth patterns as variables (columns) to ***grwt.dataset***. One is directly manipulation as the attribute is just *pandas DataFrame*.

Other is to load the growth patterns from an external tab delimited text file. To exemplify, a dataset with the observed growth patterns in Aztlan, computed from the CMPS data for the 2007-2014 period was prepared using *Stata* and exported to the file *example_Aztlan_growth.tab.txt*.

microsim.growth.load_csv(csvfile, delimiter='\t'**)**

This file can be loaded using the ***load_csv()*** method of the growth object. The external file and the growth object must have the same number of parts.

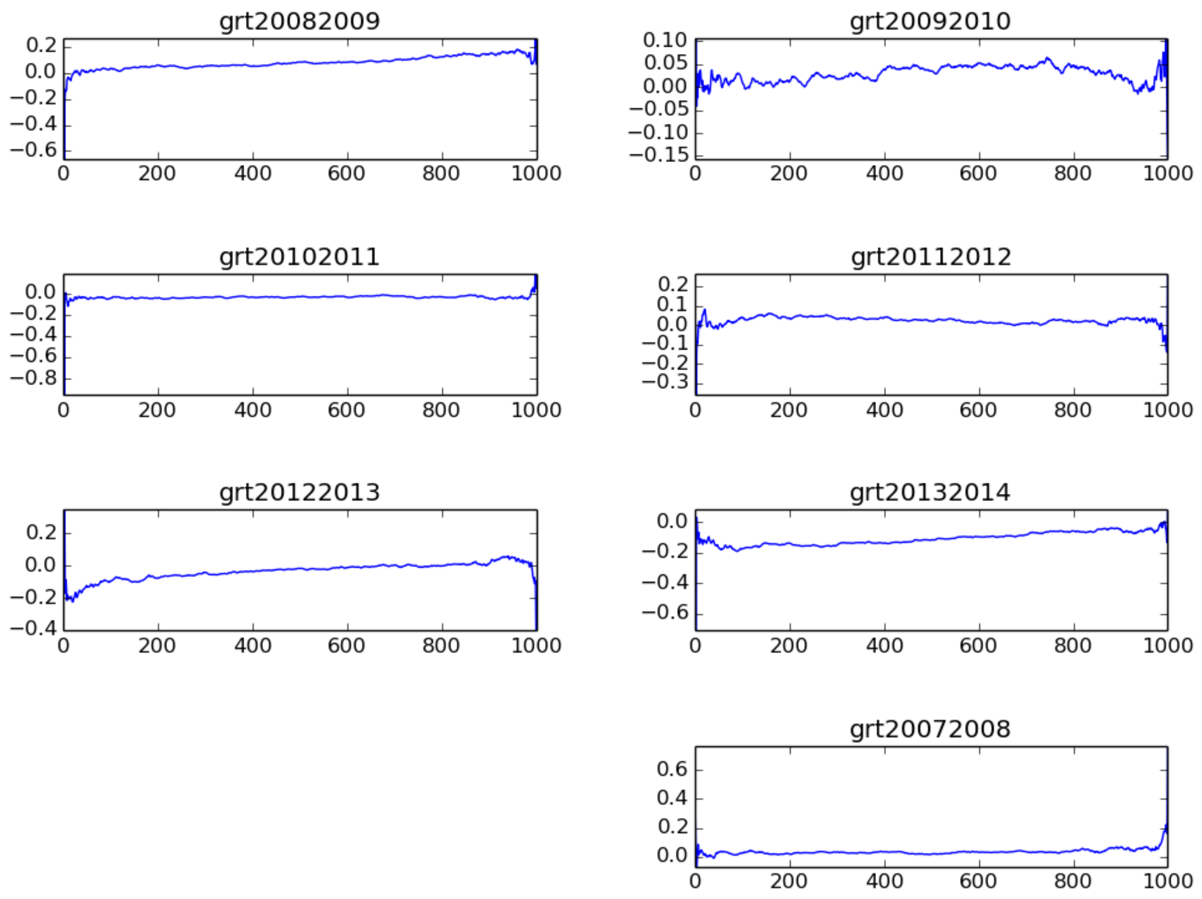
```
>>> FILE = 'example_Aztlan_growth.tab.txt'
>>> grwt.load_csv(FILE, '\t')
>>> grwt.dataset.head()
   key  partes  grt20072008  grt20082009  grt20092010  grt20102011  \
0    0       1     0.000000     0.000000     0.000000     0.000000
1    1       2     0.298798     0.000000     0.000000     0.000000
2    2       3     0.104687    -0.661190     0.107264    -0.953425
3    3       4    -0.002931    -0.280356     0.001638    -0.099029
4    4       5    -0.068420    -0.168362    -0.040986    -0.003017

   grt20112012  grt20122013  grt20132014
0     0.000000     0.000000     0.000000
1     0.000000     0.000000     0.000000
2     0.000000     0.000000    -0.710549
3    -0.360724     0.346405    -0.005397
4    -0.134360    -0.102974     0.029902

[5 rows x 9 columns]
```

Growth patterns can be visualized.

```
>>> toviz = [col for col in grwt.dataset.columns if 'grt' in col]
>>> ms.visualize(grwt.dataset, 'key', toviz)
```



Aztlan has had some very bad and some very good years and growth has not been pro-poor. We can easily check the average growth of these growth patterns.

```
>>> grwt.dataset[toviz].mean()
grt20072008    0.034907
grt20082009    0.074262
grt20092010    0.028873
grt20102011   -0.036660
grt20112012    0.022668
grt20122013   -0.034894
grt20132014   -0.109368
dtype: float64
```

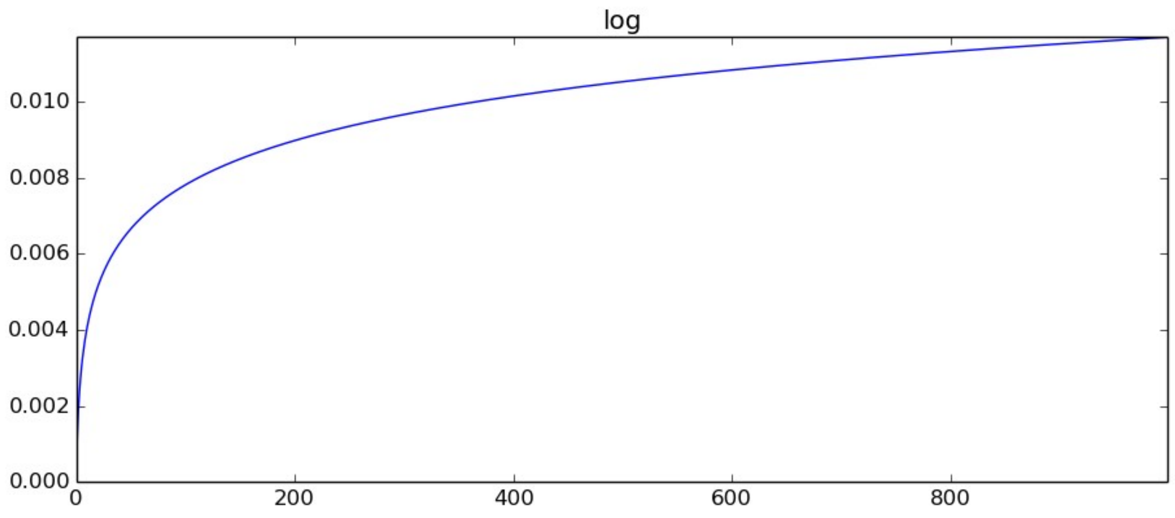
microsim.growth.add_columns(data, stub='newcol'**)**

Besides loading from an external field delimited text file, growth patterns can be created independently using python or pandas and included in the growth dataset using the method ***add_columns()*** of the growth object. This method will try to add ***data*** to the growth dataset, expecting it to be a *pandas DataFrame* (all columns) or *pandas Series*, or list, or tuple, if size and index are valid.

The growth pattern generator has methods to create theoretical growth patterns based on the logarithmic and the power functions: ***add_log()*** and ***add_power()***.

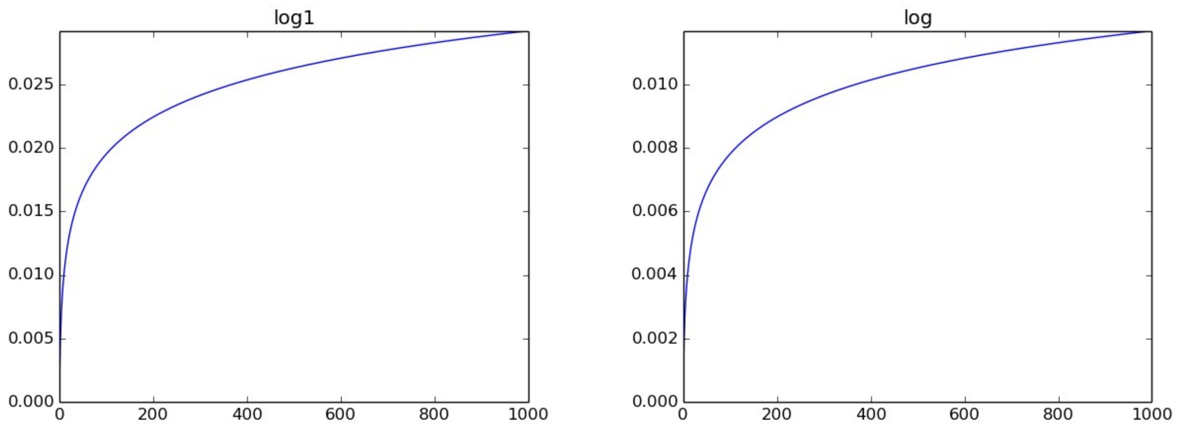
microsim.growth.add_log(stub='log', average=0.01, shift=0, flip=False, alpha=1**)**

```
>>> grwt.add_log()
>>> grwt.dataset.log.mean()
0.0099999999999999672
>>> ms.visualize(grwt.dataset, 'key', 'log')
```



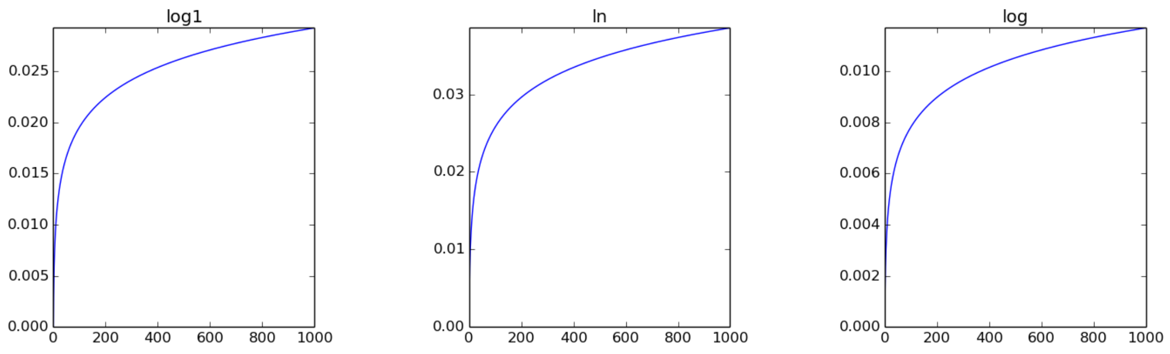
The patterns generated by the ***add_log()*** method can be changed by passing arguments. The ***average*** parameter sets the average income growth of the growth pattern, and defaults to 1%. But the average income growth can be set to 2.5%.

```
>>> grwt.add_log(average=0.025)
>>> ms.visualize(grwt.dataset, 'key', ['log','log1'])
```



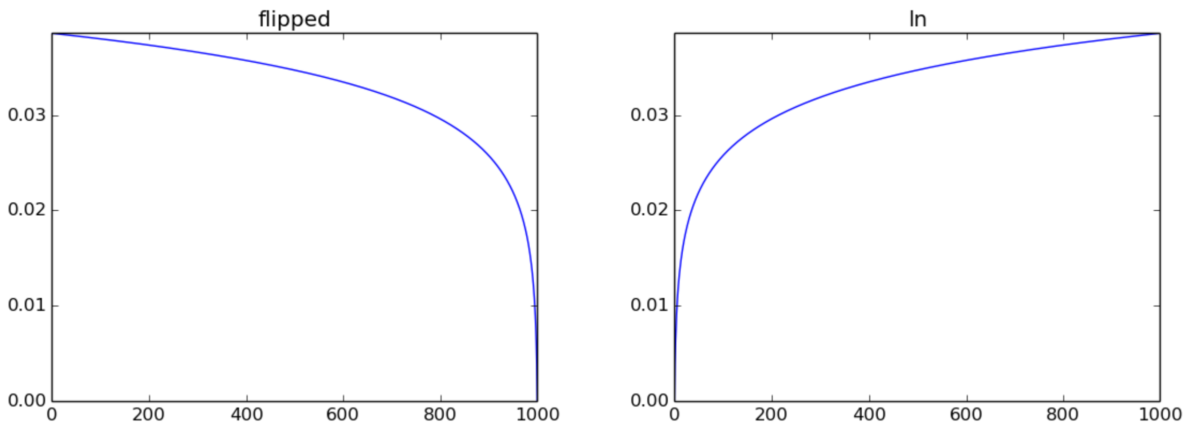
The name can be changed using the ***stub*** argument. If no ***stub*** is provided, it defaults to 'log'. If there is already a growth pattern with that name in the growth dataset, an integer will be suffixed to ***stub***.

```
>>> grwt.add_log(stub='ln', average=0.033)
>>> ms.visualize(grwt.dataset, 'key', ['log','log1', 'ln'])
```



The log function is progressive, and thus pro-rich. This can be remedied by flipping the log function.

```
>>> grwt.add_log(stub='flipped', average=0.033, flip=True)
>>> grwt.dataset[['ln', 'flipped']].mean()
ln         0.033
flipped    0.033
dtype: float64
>>> ms.visualize(grwt.dataset, 'key', ['ln', 'flipped'])
```

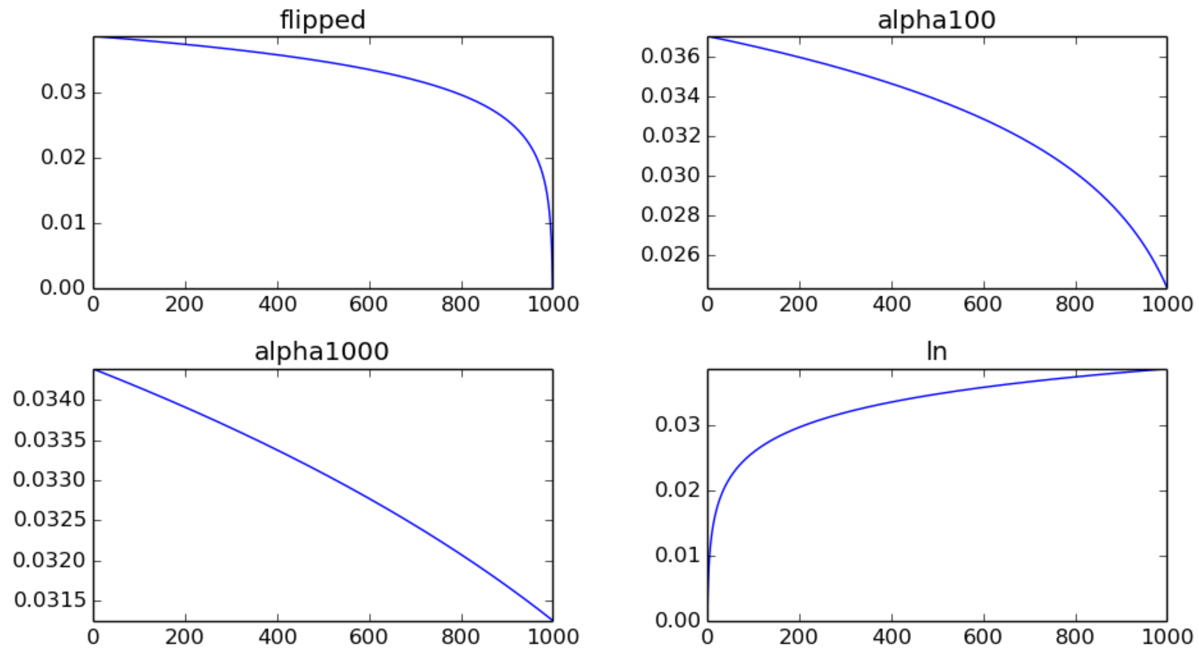


Up till now we added log growth patterns using the following formula, where g bar is the desired average growth, and parts is the number of partitions of the growth object, with alpha set to one (the default).

$$g_i = \frac{\bar{g}\ln(i+\alpha)}{\sum_{i=0}^{parts-1}\ln(i+\alpha)}$$

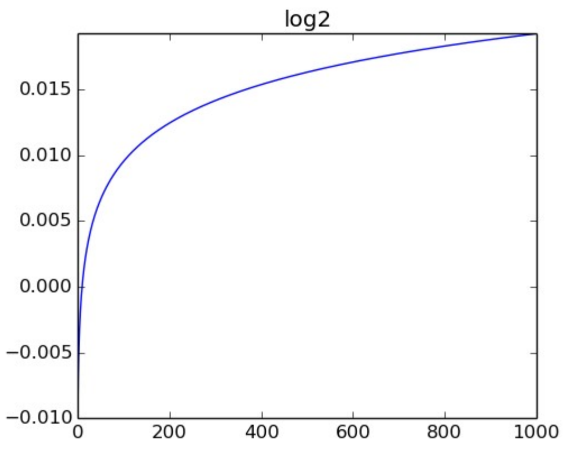
If a smoother growth pattern is desired, higher values can be set for alpha.

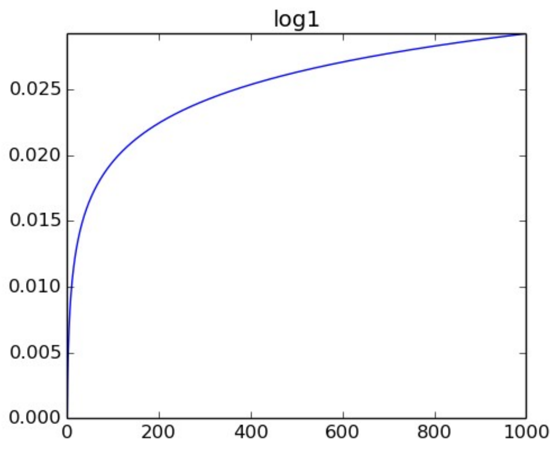
```
>>> grwt.add_log(stub='alpha100', average=0.033, flip=True, alpha=100)
>>> grwt.add_log(stub='alpha1000', average=0.033, flip=True, alpha=1000)
>>> grwt.dataset[['ln', 'flipped', 'alpha100', 'alpha1000']].mean()
ln           0.033
flipped      0.033
alpha100     0.033
alpha1000    0.033
dtype: float64
>>> ms.visualize(grwt.dataset, 'key', ['ln', 'flipped', 'alpha100', 'alpha1000'])
```



The log function does not yield negative growth rates, but the whole function can be shifted down (or up) using the shift argument. Using the shift argument changes the final average income growth of the pattern becomes the specified average plus shift.

```
>>> grwt.add_log(average=0.025, shift=-0.01)
>>> grwt.dataset[['log1', 'log2']].mean()
log1    0.025
log2    0.015
>>> ms.visualize(grwt.dataset, 'key', ['log1', 'log2'])
```



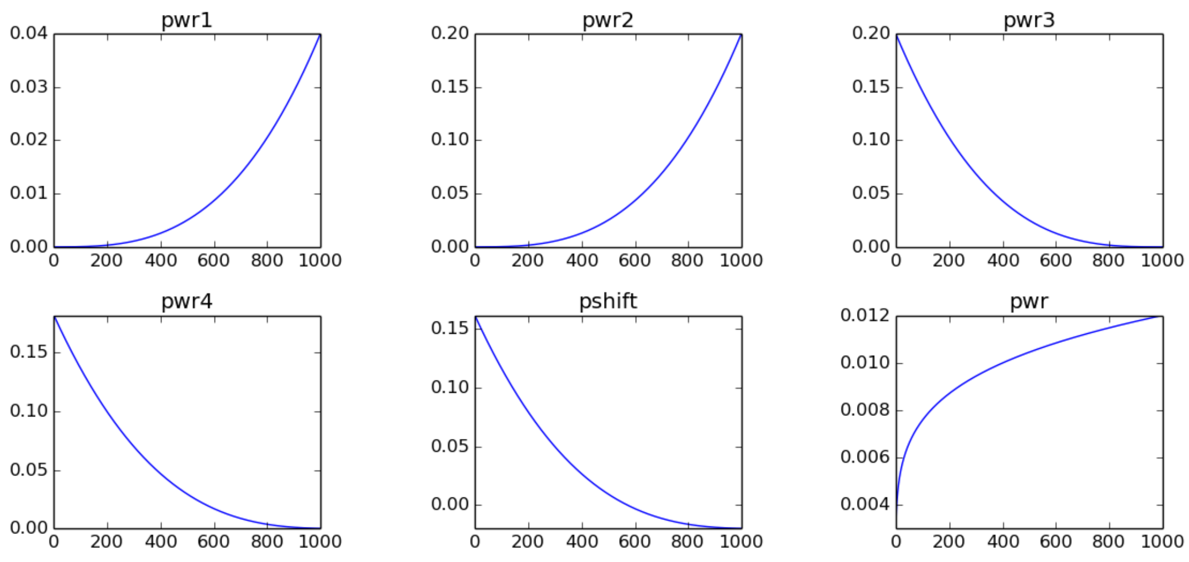


microsim.growth.add_power(power=0.2, stub='pwr', average=0.01, shift=0, flip=False, alpha=1**)**

The ***add_power()*** method has the same arguments as the ***add_log()*** method, plus the exponent of the power function. The default power ***x*** is 0.2, which yields a log-like curve.

$$g_i = \frac{\bar{g}(i+\alpha)^x}{\sum_{i=0}^{parts-1} (i+\alpha)^x}$$

```
>>> grwt.add_power()
>>> grwt.add_power(power=3)
>>> grwt.add_power(power=3, average=0.05)
>>> grwt.add_power(power=3, average=0.05, flip=True)
>>> grwt.add_power(power=3, average=0.05, flip=True, alpha=100)
>>> grwt.add_power(stub='pshift', power=3, average=0.05, shift=-0.02, flip=True,
alpha=100)
>>> grwt.dataset[['pwr', 'pwr1', 'pwr2', 'pwr3', 'pwr4', 'pshift']].mean()
pwr       0.01
pwr1      0.01
pwr2      0.05
pwr3      0.05
pwr4      0.05
pshift    0.03
dtype: float64
>>> ms.visualize(grwt.dataset, 'key', ['pwr', 'pwr1', 'pwr2', 'pwr3', 'pwr4', 'pshift'],
subplotcols=3)
```



It is important to give meaningful names to the growth patterns. The names will be stored with the micro-simulation results.

RUNNING A MICRO-SIMULATION

At this point everything is ready to start running micro-simulations: the micro-simulation object has a dataset, elasticities were added to it, and there is a growth object with many growth patterns. A limitation of this first implementation is being restricted to a single elasticity pattern for all transitions.

microsim.simulate(name, period, X, Y, P, key)

The ***simulate()*** method of the micro-simulation object is used to generate a micro-simulation. This method requires five arguments. The first is ***name***, a string to name the simulation, it will be used as a dictionary key. The second is ***period***, a tuple/list with the base line and the last year, the base line year should be the year of the seed variables, here (2014, 2030).

```
>>> simname = 'sim1'
>>> period = (2014, 2030)
```

The third argument, ***X***, is a tuple/list of four arguments ***about the income variable***: ***name*** of the income variable in the micro-simulation dataset; ***stub*** for the new income variables, for instance, 'inc' to generate 'inc2014', 'inc2015'…; ***how to apply*** the growth patterns in the growth object, 'random', 'order' (from the first – leftmost – column to the last, repeat if fewer columns than transitions), 'name_of_pattern' (to repeat a single pattern, name must be that of a column in the growth object); and a ***growth object***.

```
>>> income = ('income', 'inc', 'random', grwt)
```

The fourth argument, ***Y***, is a tuple/list of tuples/lists with three arguments ***about the resource variable***: ***name*** of the resource variable in the micro-simulation dataset; ***stub*** for the new resource variables, for instance, 'ele' to generate 'ele2014', 'ele2015'...; and ***elasticity***, the name of the elasticity variable in the micro-simulation dataset. Many resource variables can be specified in the same simulation, eg. electricity, gas and water, assuming each has its own elasticity pattern.

```
>>> resources = [('kwhpc', 'ele', 'elastkwhpc')]
```

The final arguments to ***simulate()*** are the name of the weight and that of the key variable.

```
>>> weight = 'wgt'
>>> key = 'hhkey'
>>> ms.simulate(simname, period, income, resources, weight, key)
```

The simulation is done. Now it is possible to inspect its results and add poverty and inequality indicators, as well as estimates of future total annual demand for the resource, using the microsimulation object together with the population projection and the RAM objects. But first, let's take a look at the just generated *sim1*.

```
>>> ms.seedvars
['Unnamed: 0', 'hhkey', 'income', 'kwhpc', 'wgt', 'quantkwhpc', 'elastkwhpc']
>>> ms.dataset.columns
Index([u'Unnamed: 0', u'hhkey', u'income', u'kwhpc', u'wgt', u'quantkwhpc',
u'elastkwhpc', u'inc2014', u'ele2014', u'inc2015', u'ele2015', u'inc2016', u'ele2016',
u'inc2017', u'ele2017', u'inc2018', u'ele2018', u'inc2019', u'ele2019', u'inc2020',
u'ele2020', u'inc2021', u'ele2021', u'inc2022', u'ele2022', u'inc2023', u'ele2023',
u'inc2024', u'ele2024', u'inc2025', u'ele2025', u'inc2026', u'ele2026', u'inc2027',
u'ele2027', u'inc2028', u'ele2028', u'inc2029', u'ele2029', u'inc2030', u'ele2030'],
dtype='object')
```

As expected, new variables (columns) were created in the dataset for income and electricity, from the baseline year to the end year, using the given stubs, '*inc*', and '*ele*'. Every time the ***simulate()*** method is called, it calls ***__reset__()***, to drop all variables generated in previous simulation, keeping only the seed variables.

The ***results*** attribute of the micro-simulation, which is a dictionary, now has an entry with the simulation name as key.

```
>>> ms.results.keys()
['sim1']
```

This entry is also a dictionary, with some standard entries generated for every simulation, containing the arguments passed to ***simulate()*** and a dataset with statistics for the new variables created in the micro-simulation dataset (*ms.dataset*).

```
>>> ms.results['sim1'].keys()
['name', 'reselast0', 'period', 'dataset', 'growth', 'income', 'resvar0']
>>> ms.results['sim1']['dataset']
      year     grwtpatt      mean_inc      mean_ele
2014  2014         none     15.129683     46.888361
2015  2015    alpha1000     15.354736     47.334546
2016  2016         pwr3     15.649839     48.003179
2017  2017  grt20122013     15.446029     47.537301
2018  2018  grt20092010     15.886494     48.084398
2019  2019          pwr     16.061538     48.280147
2020  2020      flipped     16.511453     48.867356
2021  2021     alpha100     17.011885     49.477018
2022  2022    alpha1000     17.561651     50.108718
2023  2023  grt20092010     18.065307     50.683558
2024  2024         log2     18.377195     50.999769
2025  2025       partes  13088.586443  11704.551200
2026  2026         pwr1  13402.727265  11793.681184
2027  2027          log  13553.491186  11844.673536
2028  2028         pwr1  13880.952890  11935.232325
2029  2029  grt20082009  15546.118586  12443.435843
2030  2030           ln  16124.215429  12621.236718

[17 rows x 4 columns]
```

The ***results dataset*** has a string variable ***grwtpatt*** that register the growth pattern applied to the income variable in each transition – for the baseline year, is 'none'. In this simulation, growth patterns were randomly chosen from the growth object, but as the variable '*partes*', which was not a growth pattern, was forgotten in the growth dataset, it was applied generating absurd results after 2024.

To correct this, first the variable '*partes*' is dropped from the growth object, then the simulation is run again. Passing the same name overwrites the previous simulation.

```
>>> grwt.dataset.drop('partes', axis=1, inplace=True)
>>> ms.simulate(simname, period, income, resources, weight, key)
Simulation sim1 results overwritten
>>> ms.results['sim1']['dataset']
      year     grwtpatt   mean_inc   mean_ele
2014  2014         none  15.129683  46.888361
2015  2015  grt20112012  14.937258  46.955874
2016  2016         log2  15.194934  47.248628
2017  2017      flipped  15.622555  47.824414
2018  2018    alpha1000  16.127542  48.434975
2019  2019  grt20112012  16.377825  48.816723
2020  2020          log  16.556163  49.014577
2021  2021          log  16.736462  49.213262
2022  2022  grt20132014  15.368933  47.318598
2023  2023      flipped  15.799340  47.893840
2024  2024  grt20072008  16.624473  48.658370
2025  2025          log  16.806337  48.856093
2026  2026         pwr3  17.114397  49.535045
2027  2027  grt20132014  15.749863  47.636190
2028  2028          pwr  15.924045  47.830403
2029  2029  grt20132014  14.686375  46.015658
```

```
2030  2030     alpha100  15.126349  46.587722

[17 rows x 4 columns]
```

The simulation could have been done repeating the same growth pattern, eg. that observed from 2011 to 2012

```
>>> income = ('income', 'inc', 'grt20112012', grwt)
>>> ms.simulate(simname, period, income, resources, weight, key)
Simulation sim1 results overwritten
>>> ms.results['sim1']['dataset']
      year     grwtpatt   mean_inc   mean_ele
2014  2014         none  15.129683  46.888361
2015  2015  grt20112012  15.394876  47.262992
2016  2016  grt20112012  15.697999  47.647912
2017  2017  grt20112012  16.040990  48.042608
2018  2018  grt20112012  16.430449  48.448450
2019  2019  grt20112012  16.872356  48.862193
2020  2020  grt20112012  17.375083  49.290505
2021  2021  grt20112012  17.950263  49.726949
2022  2022  grt20112012  18.611119  50.183291
2023  2023  grt20112012  19.378313  50.646732
2024  2024  grt20112012  20.276067  51.121446
2025  2025  grt20112012  21.337194  51.606870
2026  2026  grt20112012  22.601397  52.101647
2027  2027  grt20112012  24.119785  52.612750
2028  2028  grt20112012  25.956841  53.135789
2029  2029  grt20112012  28.193150  53.678386
2030  2030  grt20112012  30.931986  54.232690

[17 rows x 4 columns]
```

Change in simulation parameters can be done very fast and interactively to generate new simulations. For example, to do another simulation, applying just the observed growth patterns, a new growth object could be generated; but it is also possible to manipulate the existing growth object to drop the unwanted growth patterns – just make sure the key column is not dropped.

```
>>> todrop = [pat for pat in grwt.dataset if not 'grt' in pat and pat is not grwt.key]
>>> todrop
['log', 'log1', 'ln', 'flipped', 'alpha100', 'alpha1000', 'log2', 'pwr', 'pwr1', 'pwr2',
'pwr3', 'pwr4', 'pshift']
>>> grwt.dataset.drop(todrop, axis=1, inplace=True)
>>> grwt.dataset.columns
Index([u'key', u'grt20072008', u'grt20082009', u'grt20092010', u'grt20102011',
u'grt20112012', u'grt20122013', u'grt20132014'], dtype='object')
>>> simname = 'sim2'
>>> income = ('income', 'inc', 'order', grwt)
>>> ms.simulate(simname, period, income, resources, weight, key)
>>> ms.results['sim2']['dataset']
      year     grwtpatt   mean_inc   mean_ele
2014  2014         none  15.129683  46.888361
2015  2015  grt20072008  16.083191  47.679072
```

```
2016  2016  grt20082009  17.796313  49.307044
2017  2017  grt20092010  18.226967  49.862091
2018  2018  grt20102011  17.752731  49.221772
2019  2019  grt20112012  18.053722  49.609416
2020  2020  grt20122013  17.675871  49.128917
2021  2021  grt20132014  16.333881  47.282909
2022  2022  grt20072008  17.446253  48.106044
2023  2023  grt20082009  19.368499  49.795454
2024  2024  grt20092010  19.808054  50.357470
2025  2025  grt20102011  19.306713  49.716390
2026  2026  grt20112012  19.645218  50.106438
2027  2027  grt20122013  19.235675  49.652134
2028  2028  grt20132014  17.862519  47.827881
2029  2029  grt20072008  19.180262  48.692203
2030  2030  grt20082009  21.349136  50.444124

[17 rows x 4 columns]
```

ADDING RESULTS TO A MICRO-SIMULATION

Now we have two entries in the results attribute of the micro-simulation object.

```
>>> ms.results.keys()
['sim2', 'sim1']
>>> ms.cursim
'sim2'
```

Though there is an entry for *sim1*, results can only be added to the current simulation – *sim2* – which name is stored in the ***cursim*** attribute of the micro-simulation object. There are methods available to add the means, inequality measures, poverty indicators and annual demand of a resource to the results. All of them share two arguments: ***stub*** determines the variables created by the simulation for which the statistics will be calculated; and ***weight*** is the name of the population weight in the simulation dataset. All methods return a *pandas DataFrame* or *Series*, and the method ***add_results()*** should be used to add them to the ***results dataset*** of the current simulation.

microsim.mean(stub, weight, nozero=False**)**

Calculates the weighted mean of any income or consumption variable ***x*** in the microsimulation dataset.

$$\mu=\frac{1}{N}\sum_{i=1} x_i P_i$$

The mean is already included in the simulation results dataset using this method that takes population weights into account to get the point estimate. The option ***nozero*** defaults to *False*, if *True*, it does not consider the zeroes when calculating the means.

The means of the resources automatically included in the result dataset do not count the zeroes, they are the output of Component 3 of the microsimulation platform, the average consumption of those with access to a resource (consumption > 0).

```
>>> ms.mean('inc', 'wgt')
2014    15.129683
(...)
2030    21.349136
dtype: float64
>>> ms.mean('inc', 'wgt', nozero=True)
2014    15.168222
(...)
2030    21.403517
dtype: float64
```

microsim.inequality_ge(stub, weight, theta=1, nozero=False**)**

Calculates Generalized Entropy inequality measures of any income or consumption variable ***x*** in the microsimulation dataset.

$$\mathrm{GE}(\theta)=\frac{1}{N(\theta^2-\theta)}\sum_{i=1}\left[\left(\frac{x_i}{\mu}\right)^{\theta}-1\right]P_i$$

The GE indicator can not be calculated using above if zero or one are chosen for θ, special cases which correspond, respectively, to Theil's L and T.

$$\text{Theil L}(\theta=0)=\frac{1}{N}\sum_{i=1}\ln\left(\frac{\mu}{x_i}\right)P_i$$

$$\text{Theil T}(\theta=1)=\frac{1}{N}\sum_{i=1}\left(\frac{x_i}{\mu}\right)\ln\left(\frac{x_i}{\mu}\right)P_i$$

Generalized entropy indicators with $\theta <= 0$, or $\theta = 1$, can not be calculated if ***x*** has zeroes, thus the option ***nozero*** is forced. For other values of θ, the option ***nozero*** defaults to *False*, if *True*, it does not consider the zeroes when calculating the inequality measure.

```
>>> ms.inequality_ge('inc', 'wgt', 1)

There were zeroes in incxxxx
Ge(1) did not consider those obs.
2014    0.358208
2015    0.414696
2016    0.438210
(...)
```

```
2028    0.558340
2029    0.639339
2030    0.659209
dtype: float64
>>> ms.inequality_ge('inc', 'wgt', 2)

There were zeroes in incxxxx
2014    0.618484
2015    0.961910
2016    0.972258
(...)
2028    1.087733
2029    1.830813
2030    1.780643
dtype: float64
>>> ms.inequality_ge('inc', 'wgt', 2, nozero=True)

There were zeroes in incxxxx
Ge(2) did not consider those obs.
2014    0.615642
2015    0.958195
2016    0.968517
(...)
2028    1.083699
2029    1.824891
2030    1.774849
dtype: float64
```

microsim.inequality_gini(stub, weight)

Calculates the Gini inequality measure of any income or consumption variable ***x*** in the microsimulation dataset. The dataset is sorted in the ascending order of ***x***, then the relative cumulative distributions of the population and of the variable (weighted) are computed. These two distributions define the Lorenz curve in a unit square, from which the Gini index is calculated as twice the area between the curve and the diagonal perfect equality line.

$$x_1 \leq x_2 \leq (\ldots) \leq x_{i-1} \leq x_i \leq x_{i+1} \; (\ldots) \leq x_n$$

$$n_i = \frac{1}{N} \sum_{i=1}^{i} P_i$$

$$y_i = \frac{1}{\mu N} \sum_{i=1}^{i} x_i P_i$$

$$Gini = 1 - \sum_{i=2} (y_i + y_{i-1})(n_i - n_{i-1})$$

```
>>> ms.inequality_gini('inc', 'wgt')
2014    0.439520
2015    0.456342
```

```
2016    0.472973
(...)
2028    0.547773
2029    0.564956
2030    0.577179
dtype: float64
```

microsim.poverty(stub, weight, plines)

Calculates poverty indicators for any income or consumption variable ***x*** in the microsimulation dataset, for a set of poverty lines. The argument ***plines*** is a tuple/list with the poverty lines, eg. (1.9, 3.1). For each poverty line the ***poverty()*** method returns five statistics.

Define an indicator variable ***Z*** that equals one when a household is poor in variable ***x***, given a poverty line ***z***, and zero otherwise.

$$Z = \begin{matrix} 0 & if & z - x_i \leq 0 \\ 1 & if & z - x_i > 0 \end{matrix}$$

The proportion of poor individuals in the population is the poverty headcount ratio ***H***, the weighted average of the indicator variable.

$$H = \frac{1}{N} \sum_{i=1}^{i} Z_i P_i$$

The normalized poverty gap ***G***, or the average shortfall of the poor as a proportion of the poverty line informs about the intensity of poverty.

$$G = \frac{1}{\sum_{i=1}^{i} Z_i P_i} \sum_{i=1} Z_i P_i \left(\frac{z - x_i}{z} \right)$$

Inequality among the poor ***I*** is represented by the generalized entropy measure of inequality with θ = 2, GE(2), calculated just for the poor. GE(2) is a special case because it is equal to half the square of the coefficient of variation of ***x***. These measures are related by the FGT class of indicators (Foster, Greer and Thorbecke, 1984).

$$FGT\alpha=\frac{1}{N}\sum_{i=1} Z_i P_i\left(\frac{z-x_i}{z}\right)^{\alpha}$$

The FGT class of indicators has as special cases: the poverty headcount ratio ***H***, when **α** = 0; ***HG*** is the poverty gap ratio, **α** = 1; and ***H[G^2 + I^2(1 – G)^2]*** is the severity of poverty, **α** = 2. In the output, the FGT indicators are named as *p0*, *p1* and *p2*, ***G*** as *pgap*, and the GE(2) indicator of the poor as *pge2*, each followed by the value of the poverty line within parenthesis**.**

```
>>> ms.poverty('inc', 'wgt',[3.1])
       p0(3.1)   p1(3.1)   p2(3.1)  pgap(3.1)  pge2(3.1)
2014  0.042621  0.012776  0.006665   0.299755   0.067833
2015  0.042621  0.012430  0.006437   0.291642   0.065729
2016  0.040877  0.013105  0.007173   0.320603   0.078742
(...)
2028  0.117926  0.047334  0.026059   0.401382   0.083531
2029  0.111572  0.045810  0.025222   0.410583   0.082733
2030  0.109063  0.044976  0.025217   0.412389   0.088557

[17 rows x 5 columns]
```

microsim.totaldemand(stub, pop, ram, correct=1, unit=1e6**)**

Calculates the total demand of a resource integrating the three components of the microsimulation platform: the population projection, the resource access model and the micro-simulation dataset. The ***stub*** is the one assigned to a resource variable in the current micro-simulation dataset; ***pop*** and ***ram*** are, respectively, the population and the RAM objects. ***Correct*** defaults to one, is a constant to be applied to correct the level of the average consumption (as discussed – see section on the construction of electricity consumption). ***Unit*** is a value to scale the final amount – the product of ***stub*** average, ***pop*** and ***ram*** will be divided by it.

```
>>> ms.totaldemand('ele', pop, ram, 1.13, 1e6)
2014    806.562739
2015    823.169292
2016    854.145848
(...)
2028    848.284465
2029    864.245072
2030    895.773025
dtype: float64
```

add_results(self, data, name='newvar'**)**

The generated results were not actually added to the results dataset. Though this can be done directly by the user using *pandas*, the ***add_results()*** method simplifies the task. We will use it to add the total demand for electricity, some inequality measures for electricity consumption, and then income inequality and poverty measures for the US$ 1.9/day and the US$ 3.1/day poverty lines.

```
ms.add_results(pop.projection.icol(0).loc[2014:2030], 'PopMedvarWPP')
ms.add_results(ram.RAM)
ms.add_results(ms.totaldemand('ele', pop, ram, 1.13), 'eleGWh')
ms.add_results(ms.inequality_gini('ele', 'wgt'), 'eleGini')
ms.add_results(ms.inequality_ge('ele', 'wgt', 0), 'eleTheilL')
ms.add_results(ms.inequality_ge('ele', 'wgt', 1), 'eleTheilT')
ms.add_results(ms.inequality_gini('inc', 'wgt'), 'Gini')
ms.add_results(ms.inequality_ge('inc', 'wgt', 0), 'TheilL')
ms.add_results(ms.inequality_ge('inc', 'wgt', 1), 'TheilT')
ms.add_results(ms.poverty('inc', 'wgt',[1.9, 3.1]))
```

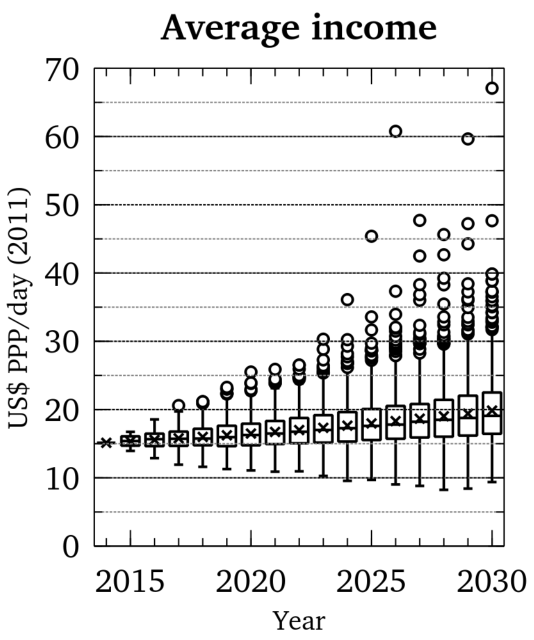
WORKING WITH MICRO-SIMULATIONS

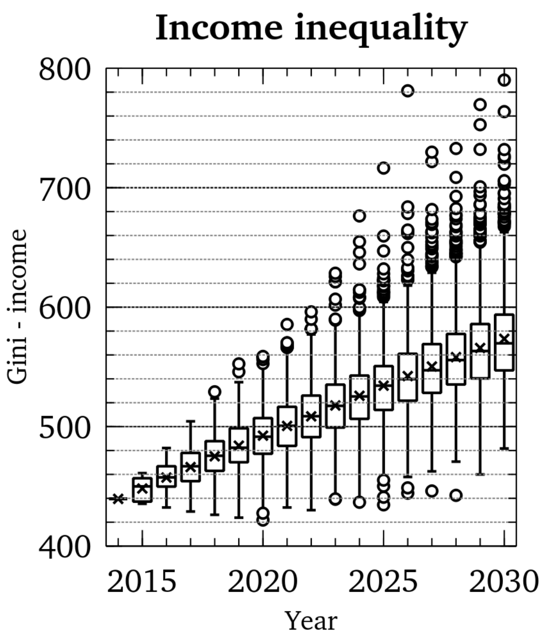
In the real world, most likely many micro-simulations will be run to establish different scenarios and their “confidence intervals”. Consider another example: we assume that from 2014 to 2030, in every transition, income will follow a growth pattern previously observed in the 2007-2014 period. In simulation 2 the patterns were ordered, so that growth from 2014 to 2015 was equal to that from 2007 to 2008, and so on, up to 2020 to 2021, then the growth datasets ran out of patterns and started over applying the 2007-2008 pattern to 2021-2022.

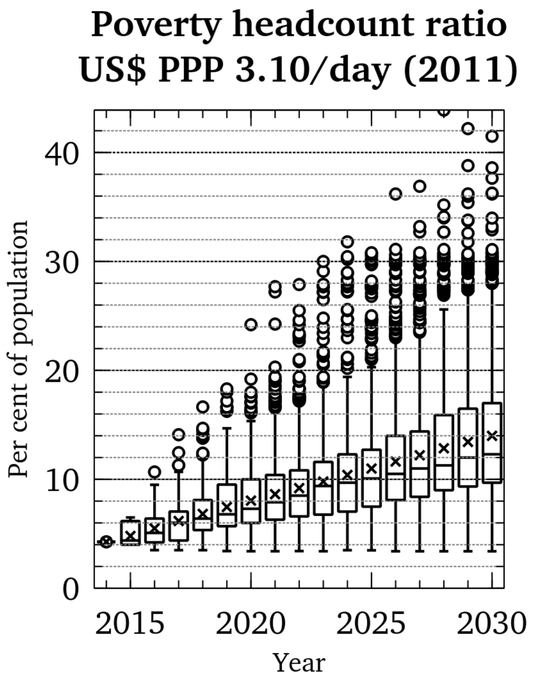
Now we say that future transitions can follow any of those observed patterns, in any order. We apply them randomly in 1000 simulations to get an interval estimate of total demand for all years of the simulation period. The results of the simulations are exported to field delimited text files and open in another application, *veusz* (free, open source, and written in python) to produce a grid of boxplots.

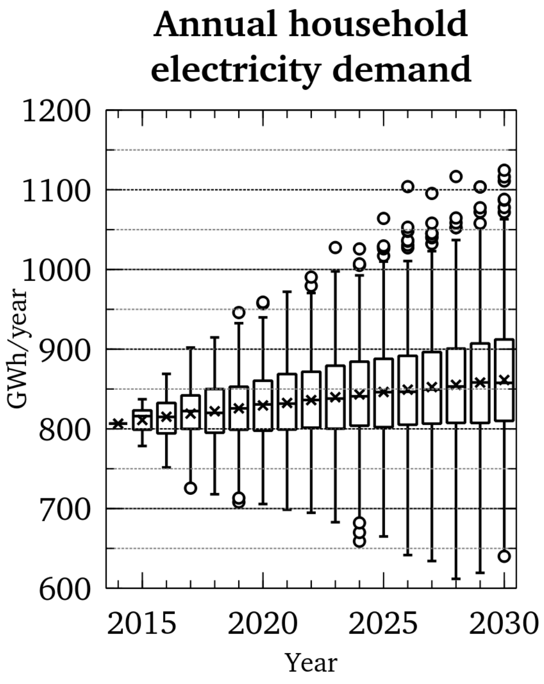
```
>>> simname = 'mult'
>>> income = ('income', 'inc', 'random', grwt)
>>> totaldemand = pd.DataFrame()
>>> avgincome = pd.DataFrame()
>>> gini = pd.DataFrame()
>>> pov310 = pd.DataFrame()
>>> print 'Running simulations!'
>>> for sim in range(1000):
...     print '.',
...     ms.simulate(simname, period, income, resources, weight, key)
...     out = pd.DataFrame(ms.totaldemand('ele', pop, ram, 1.13))
...     totaldemand = pd.concat((totaldemand, out.T), ignore_index=True)
...     out = pd.DataFrame(ms.mean('inc', 'wgt'))
...     avgincome = pd.concat((avgincome, out.T), ignore_index=True)
...     out = pd.DataFrame(ms.inequality_gini('inc', 'wgt'))
...     gini = pd.concat((gini, out.T), ignore_index=True)
...     out = pd.DataFrame(ms.poverty('inc', 'wgt', [3.1])['p0(3.1)'])
...     pov310 = pd.concat((pov310, out.T), ignore_index=True)
...
>>> totaldemand.to_csv('1000simstd.txt', '\t')
>>> avgincome.to_csv('1000simsavg.txt', '\t')
>>> gini.to_csv('1000simsgini.txt', '\t')
>>> pov310.to_csv('1000simspov310.txt', '\t')
```

Obviously, other quantities could be estimated in the same run: averages, inequality measures, poverty indicators.


Average income
US$ PPP/day (2011)
0
10
20
30
40
50
60
70
2015
2020
2025
2030
Year


Income inequality
Gini - income
400
500
600
700
800
2015
2020
2025
2030
Year


Poverty headcount ratio
US$ PPP 3.10/day (2011)
Per cent of population
0
10
20
30
40
2015
2020
2025
2030
Year


Annual household
electricity demand
GWh/year
600
700
800
900
1000
1100
1200
2015
2020
2025
2030
Year