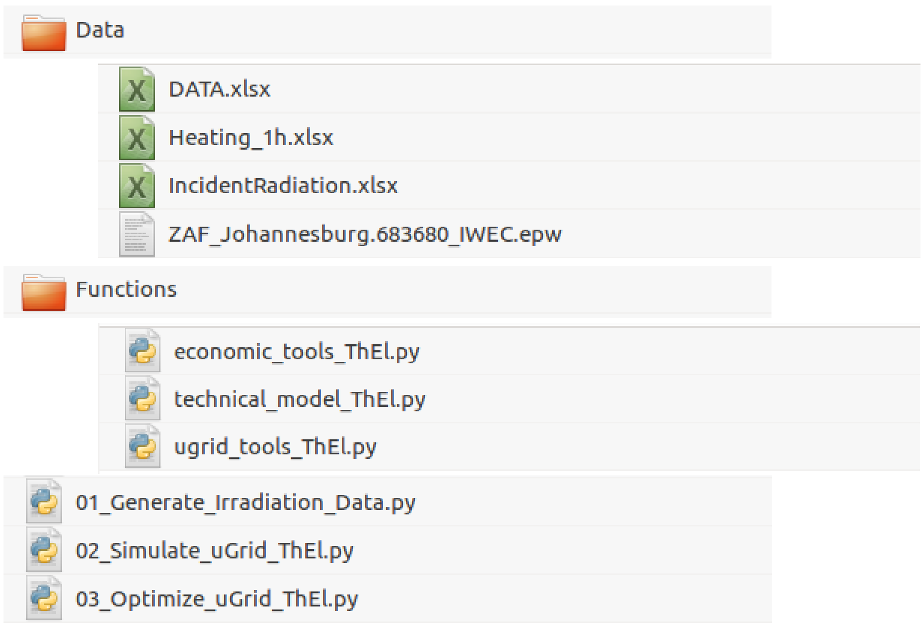
QUICK START USER GUIDE

By Queralt Altés Buch and Matthew Orosz

January 2017

uGrid is an open source code engineering design and decision tool developed to aid the engineering, procurement and construction (EPC) of sustainable, reliable microgrids relying on a combination of solar and backup (fossil fueled) generator energy sources. By introducing technical and economic optimization and integrating many of the major EPC decisions, uGrid can speed up and improve a prospective microgrid project or program, as well indicate which financial conditions (cost of capital) and tariff can drive the project in the market. This guide presents an overview of uGrid files and functions to enable a user to run the simulation and modify/adapt the program for a community of interest.

uGrid is written in Python programming language. The structure of the model is the following:



```
Data
    DATA.xlsx
    Heating_1h.xlsx
    IncidentRadiation.xlsx
    ZAF_Johannesburg.683680_IWEC.epw
Functions
    economic_tools_ThEl.py
    technical_model_ThEl.py
    ugrid_tools_ThEl.py
01_Generate_Irradiation_Data.py
02_Simulate_uGrid_ThEl.py
03_Optimize_uGrid_ThEl.py
```

The folder Data contains the external input data of the model, which is mostly provided in excel files. The Functions folder contains the computation functions that are later called by the Main files (*02_Simulate_uGrid_ThEl.py* and *03_Optimize_uGrid_ThEl*.py):

- *economic_tools_ThEl.py*
  Contains the economic functions used to perform an economic study of a microgrid system.
- *technical_model_ThEl.py*
  Contains the model of the generation systems (PV, ORC, LPG), the energy balances of the storage systems (batteries and thermal energy storage), a rule-based control for the dispatching of the demand, and the fuel consumption evaluation. This file also contains a function that wraps all these models to compute the energy balance for one year of operation.

- *ugrid_tools_ThEl.py*
  This file contains small functions that are mostly helpful tools used for analyzing the results or plotting the dispatched power.

The main 3 files are used to run the optimization code for a default community microgrid (Ha Nkau in Lesotho) as follows:

- *01_Generate_Irradiation_Data.py*
  This script computes the direct normal radiation (DNI) incident on a north-south tracking collector and the incident total radiation on a titled surface. The input data is the typical meteorological year (TMY) from EnergyPlus, read as an epw file, and the coordinates of the community of interest. The output is the excel file *IncidentRadiation.xlsx*, which is saved in the DATA folder for later use. The script makes use of the following libraries:
    - *pyepw* for to ready the typical meteorological year data
    - *pvlib* to compute the incident irradiation

- *02_Simulate_uGrid_ThEl.py*
  Simulates a particular case of uGrid (sizing of the system is imposed). The model performs first the technical computations characteristic of the given system (e.g. propane consumption) and then performs an economic study, as follows:
  <u>Run the technical model:</u> The energy balance for one year of operation is calculated here (aka a dynamic simulation) according to the following inputs and directives: local time-varying conditions for solar energy and temperature are input via a TMY file and a load curve is input from a statistical prediction function generated from a power meter dataset; the microgrid is directed to satisfy 100% of demand via various configurations of generation which in all cases includes a backup LPG generator capable of supplying 100% of loads. Other generation from renewable sources serves to displace LPG fuel consumption. The main result is propane consumption which, along with the power plant configuration, will be used for economic calculation. Results can be stored in an excel file if desired. The dispatching of both electrical and thermal powers can be plot for the desired range of time.
  <u>Perform an economic study:</u> The result from the dynamic simulation is used to calculate the cost recovery tariff in a cash flow model that includes cost functions for the initial cost of the power plant, its operating (fuel) and maintenance cost functions, and loan repayment according to the tenor and interest rate available. This program hill climbs from smaller to higher tariffs until all cash flows for the project are positive across the project lifetime.

- *03_Optimize_uGrid_ThEl.py*
  Main optimization function. Particle Swarm Optimization Algorithm is the selected technique. PSO is used to efficiently evaluate the microgrid design space by progressively selecting parameters to evaluate based on the success level of an initial population of candidates, converging on a solution by learning from the populations trajectory through the design space. This saves significant time in simulation compared to evaluating all possible combinations in an exhaustive search.

Installing the required software

Note: It is recommended to use Spyder because it is simple, intuitive and very useful. Even those who have never worked with Python will find easy since its interface is very similar to Matlab.

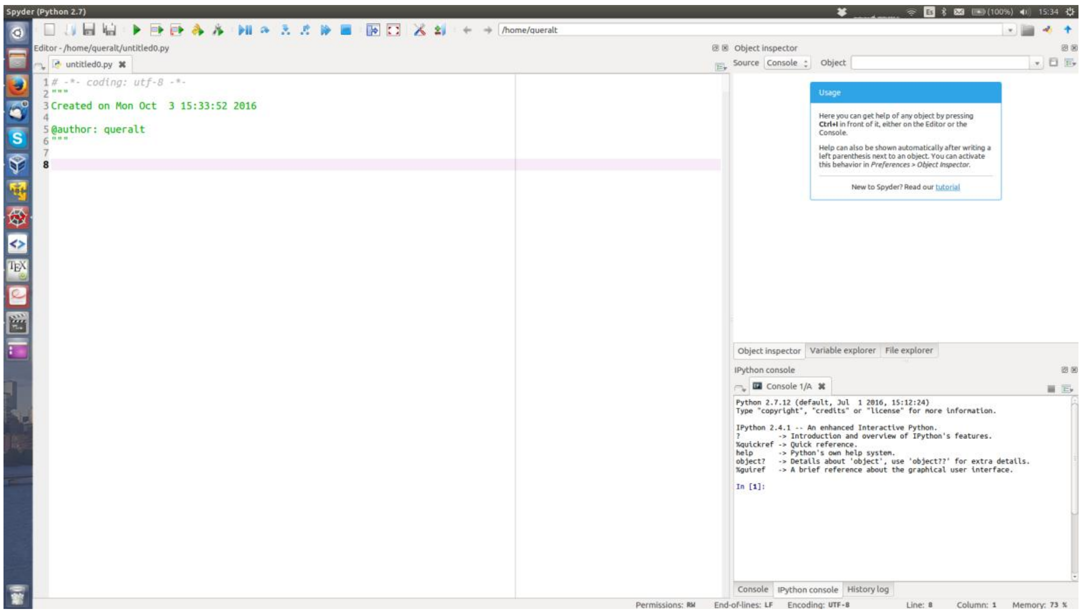
1. Download **Anaconda** with **Python 2.7** version
   https://www.continuum.io/downloads

2. Open **Spyder** (software that comes with Anaconda)

   If Spyder does not get installed automatically (e.g. when using MacOS X), install it from:
   https://pythonhosted.org/spyder/installation.html

   Spyder can be launched from Anaconda's Navigator GUI.

   Once opened, the interface should be like in the following picture.



```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct  3 15:33:52 2016

@author: queralt
"""

```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Object Inspector*.

New to Spyder? Read our tutorial

```
Python 2.7.12 (default, Jul  1 2016, 15:12:24)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
%guiref   -> A brief reference about the graphical user interface.

In [1]:
```

   At left you have the *Editor*, up-right you have the *Object Inspector* and *Variable Explorer*, and down-right you have the consoles (*Console*, *IPython console*) and the *History log*.

   You can always go back to the original layout (the one in the picture) by going to: View / Reset window layout

   Although both consoles can be used, it is recommended to use the *IPython console* to run the models because it tends to work better than the *Console*. However, sometimes debugging is better operated from the *Console*.

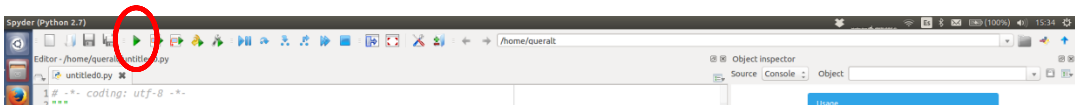
   For new users, documentation about Spyder can be found in:
   https://pythonhosted.org/spyder/index.html

Instructions for running the u-Grid model

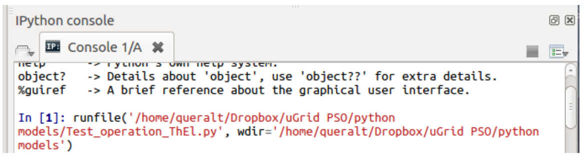
Simulating u-Grid (*02_Simulate_uGrid_ThEl.py*)

1. File / Open: 02_Simulate_uGrid_ThEl.py
2. In the script:
    a. Choose to store or not the results by setting to True or False the *store_results* Boolean variable
    b. Change the *Parameters* according to the community of interest
    c. Choose the main heating system by setting to True the Boolean variable *heating_TES* or *heating_ORC*
    d. Impose the size of the system by imposing BattkWh (kWh), Pnom_PV (kW), CSP_A (m^2), Pnom_ORC (kW), TESkWh (kWh).
    e. Define the axis time range for the plots

3. To run the model:
    a. Have the *IPython Console* selected and opened (like in the previous picture).
    b. Make sure that the computer cursor is on the script that you want to run.
    c. Press the green "Play" triangle on the top bar:

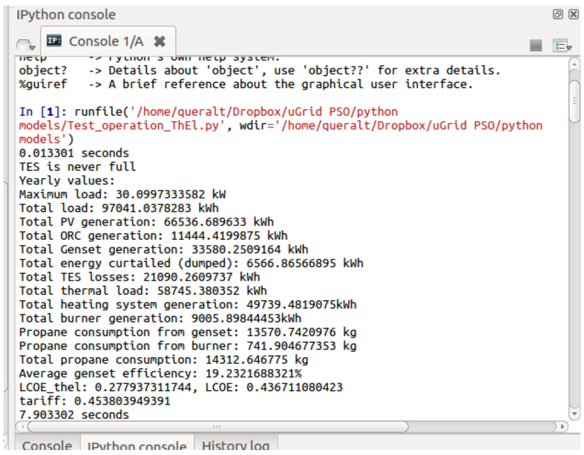


    d. In the console a command line should appear:



```
object?   -> Details about 'object', use 'object??' for extra details.
%guiref   -> A brief reference about the graphical user interface.

In [1]: runfile('/home/queralt/Dropbox/uGrid PSO/python
models/Test_operation_ThEl.py', wdir='/home/queralt/Dropbox/uGrid PSO/python
models')
```

    e. When finished, results are going to be printed:



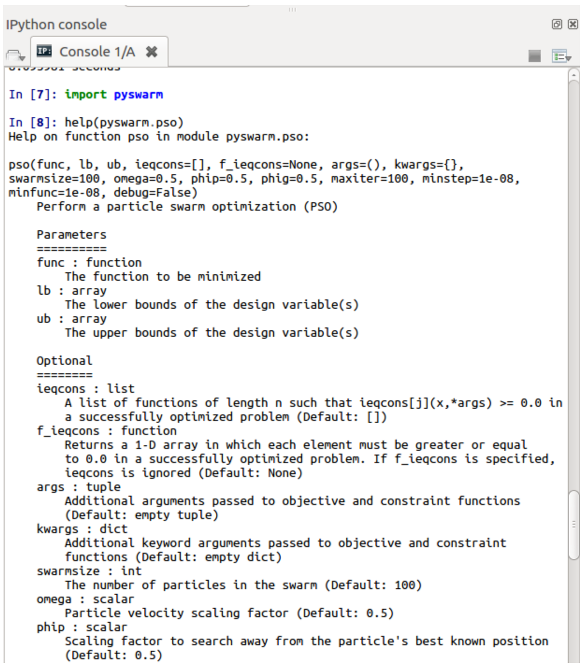
```
object?   -> Details about 'object', use 'object??' for extra details.
%guiref   -> A brief reference about the graphical user interface.

In [1]: runfile('/home/queralt/Dropbox/uGrid PSO/python
models/Test_operation_ThEl.py', wdir='/home/queralt/Dropbox/uGrid PSO/python
models')
0.013301 seconds
TES is never full
Yearly values:
Maximum load: 30.0997333582 kW
Total load: 97041.0378283 kWh
Total PV generation: 66536.689633 kWh
Total ORC generation: 11444.4199875 kWh
Total Genset generation: 33580.2509164 kWh
Total energy curtailed (dumped): 6566.86566895 kWh
Total TES losses: 21090.2609737 kWh
Total thermal load: 58745.380352 kWh
Total heating system generation: 49739.4819075kWh
Total burner generation: 9005.89844453kWh
Propane consumption from genset: 13570.7420976 kg
Propane consumption from burner: 741.904677353 kg
Total propane consumption: 14312.646775 kg
Average genset efficiency: 19.2321688321%
LCOE_thel: 0.277937311744, LCOE: 0.436711080423
tariff: 0.453803949391
7.903302 seconds
```

Note: By default, plots are going to be printed inside the console window. In order to have the plots outside the console widow, the user should run the command: **%matplotlib**

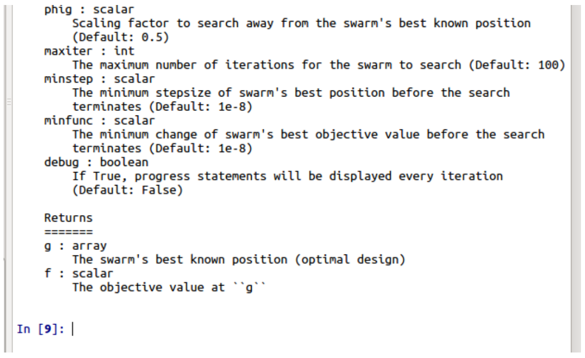
Optimizing u-Grid (*03_Optimize_uGrid_ThEl.py*)

Note 1: Particle Swarm Optimization (PSO) is the selected technique to optimize the size of the microgrid system. The algorithm is already implemented in python in the library ***Pyswarm***. The library needs to be installed to run the optimization (http://pythonhosted.org/pyswarm/). To get information about the PSO optimization from *Pyswarm*, run the command lines:

```
>> import pyswarm
>> help(pyswarm.pso)
```



```
In [7]: import pyswarm

In [8]: help(pyswarm.pso)
Help on function pso in module pyswarm.pso:

pso(func, lb, ub, ieqcons=[], f_ieqcons=None, args=(), kwargs={},
swarmsize=100, omega=0.5, phip=0.5, phig=0.5, maxiter=100, minstep=1e-08,
minfunc=1e-08, debug=False)
    Perform a particle swarm optimization (PSO)

    Parameters
    ==========
    func : function
        The function to be minimized
    lb : array
        The lower bounds of the design variable(s)
    ub : array
        The upper bounds of the design variable(s)

    Optional
    ========
    ieqcons : list
        A list of functions of length n such that ieqcons[j](x,*args) >= 0.0 in
        a successfully optimized problem (Default: [])
    f_ieqcons : function
        Returns a 1-D array in which each element must be greater or equal
        to 0.0 in a successfully optimized problem. If f_ieqcons is specified,
        ieqcons is ignored (Default: None)
    args : tuple
        Additional arguments passed to objective and constraint functions
        (Default: empty tuple)
    kwargs : dict
        Additional keyword arguments passed to objective and constraint
        functions (Default: empty dict)
    swarmsize : int
        The number of particles in the swarm (Default: 100)
    omega : scalar
        Particle velocity scaling factor (Default: 0.5)
    phip : scalar
        Scaling factor to search away from the particle's best known position
        (Default: 0.5)
```



```
    phig : scalar
        Scaling factor to search away from the swarm's best known position
        (Default: 0.5)
    maxiter : int
        The maximum number of iterations for the swarm to search (Default: 100)
    minstep : scalar
        The minimum stepsize of swarm's best position before the search
        terminates (Default: 1e-8)
    minfunc : scalar
        The minimum change of swarm's best objective value before the search
        terminates (Default: 1e-8)
    debug : boolean
        If True, progress statements will be displayed every iteration
        (Default: False)

    Returns
    =======
    g : array
        The swarm's best known position (optimal design)
    f : scalar
        The objective value at ``g``

In [9]: |
```

Note 2: Nelder-Mead algorithm can also be used. It is recommended to use it when guess values for the global optimum are known (e.g. from a PSO run), since it is precise and fast. Do not use Nelder-Mead to find global optimums with random guess values since it will get stuck in a local optimum.

1. File / Open: 03_Optimize_uGrid_ThEl.py
2. In the script:
    a. Change the *Parameters* according to the community of interest

   b. Choose the main heating system by setting to True the Boolean variable *heating_TES* or *heating_ORC*
   c. Choose the type of optimization algorithm PSO or Nelder-Mead by setting the Boolean variable *use_PSO* to True or False, respectively.

3. Run the model: Press the green "Play" triangle on the top bar

Modifying uGrid to evaluate a community of interest

uGrid is parameterized by default for a community in Lesotho, Ha Nkau. The main differences between Ha Nkau and another community of interest may include meteorology in the new location (different temperatures and solar irradiation), a difference in the predicted loads, and the cost differential associated with the number of poles / distribution wire needed to reach the connections.

TMY

The yearly meteorological data (irradiation and temperature) is computed for a community of interest with the method in *generate_irradiation_data.py*. This script computes the total direct normal radiation (DNI) incident on a north-south tracking collector and the incident total radiation on a titled surface and stores it as an excel file (*IncidentRadiation.xlsx*) used as an input by uGrid. To update uGrid with the meteorological parameters of the new location, the excel file must be updated. For this purpose, the input parameters of the computation method *generate_irradiation_data.py* must be updated for the community of interest. These parameters are mainly the community coordinates (latitude and longitude). By running the method, the excel file will be updated.

```
# Specify coordinates:
latitude = -29
longitude = 28
```

Load

The default load dynamic in uGrid was created using a probabilistic distribution function (PDF) derived from a dataset of household consumption in a comparable community in South Africa. The household PDF was scaled up by the number of households in Ha Nkau and the result is the *LoadKWT* sheet in the excel file *DATA.xlsx*. A new dataset representing the load dynamic expected at the community of interest should be generated in the same format as *LoadKWT* and the data should be copied into this excel file. Note that anticipated % line losses on the distribution network should be represented as additional load in the dataset. The following parameter derived from the maximum value in the new *LoadKWT* dataset must be updated the main file for either simulating (*02_Simulate_uGrid_ThEl.py*) or optimizing (*03_Optimize_uGrid_ThEl.py*) the uGrid:

```
Pmax_load=30      # maximum power output of the load curve [kW]
```

Battery bank “smart charging”

uGrid also contains a “smart charging” strategy for when the backup generator should turn off based on the SOC of the battery bank. This strategy depends on the anticipated amount of load in kWh left until dawn, which in practice would be calculated in real time by the power control system by averaging the load left in the previous X days of operation. In uGrid this load left is calculated for the default community and included as the *FullYearEnergy* sheet in *DATA.xlsx*. This sheet must be updated to reflect the dynamics of the PDF in the community of interest.

The “smart charging” strategy can be selected by setting the following parameter (1-selected, 0-unselected) for either simulating (*02_Simulate_uGrid_ThEl.py*) or optimizing (*03_Optimize_uGrid_ThEl.py*) the uGrid:

```
smart=1           # use a charging strategy that attempts to minimize the
usage of the genset
```

Distribution costs

The distribution costs for a community of interest must be updated for each case. The following parameters in *economic_tools_ThEl.py* must be updated:

```
num_nodes = 90          # Number of connections / meters on the network
Dist_km = 5             # km from ViPor
num_step_up_trans = 1   # from ViPor
num_pole_trans = 3      # Number of transformers on network from the ViPor
topology
num_poles = Dist_km/0.05      # Number of poles, based on 50 meter pole-to-
pole distance, or can be updated from a more detailed mapping
```

Component cost functions

In case the cost of raw materials for a microgrid varies in the region of interest, these cost functions can be updated *in economic_tools_ThEl.py*:

```
Cost_dist_wire = 0.5*1000     # 0.5 USD/m
Cost_batt = 130               # USD/kWh
Cost_panels = PVkW*1000       # PV price via Alibaba 2014
Cost_control = 5000           # STG Build or PLC
Cost_Pole = 40    # Transmission Pole Prices 2016 from treatedpoles.co.za
Cost_Pole_Trans = 150         # USD, Alibaba 20kVA single phase 11kV/.22kV
Cost_Step_up_Trans = 1000     # USD, Alibaba 63kVA single phase 11kV/.22kV

Cost_Smartmeter = 50*num_nodes      # Iometer
```

```
Cost_MPesa = 70*peakload     # Estimate for merchant services with vodacom
Cost_inv = peakload*800      # USD/kW peak

inputs['Cost_Propane_yr'] = m_propane*1.24     # USD/kg, RSA prices 2014
inputs['Cost_Propane_genset_yr'] = m_propane_genset*1.24
inputs['Cost_Propane_burner_yr'] = (m_propane-m_propane_genset)*1.24

inputs['C1_LPG'] = -10354.1143 + 6192.606*np.log(peakload)
      # USD, propane genset costs based on generac lineup
inputs['C1_CSP'] = CSP_A*150 + ORCkW*2000 + 25*TESkWh
      # Collector: $150/m^2, ORC: $2/W, TES: $25/kWh
```