Problem Log

TFG 2015 – Hybrid Grid simulation with optimization algorithm

1. The dieselNode was the first node we made ourselves and while making the dieselNode we did not know how, what and where to start immediately. This caused some problems because we needed to change a lot of stuff afterwards;

In the following lines you will find examples of this:

* 1. We did not work with XML-files or separate functions (we wrote everything in the DieselNode);
  2. We made a mistake by working with percentage of power for the fuel consumption calculation and we just needed to use the power vector and the consumption vector (and interpolate the values);
  3. In the beginning we also made some mistakes with string2num, here we sometimes forgot to write .text;
  4. We had a few problems with our power curve:
     1. At first we made 2 lines in our XML <PowerX> [3000 100 2200] </PowerX>

<freqY> [47.5 50 52.5] </freqY>

* + 1. The second thing we learned is that we could do the same thing in one line by putting a semi-colon (“;”) between the numbers of the X-axis and the Y-axis

<FreqPowerCurve>[47.5 50 52.5 ; 3000 100 2200] </FreqPowerCurve >

* + 1. The third thing that we realized is that we only need the min and max value of the lines when we are interpolating

<FreqPowerCurve>[47.5 52.5 ; 3000 2200] </FreqPowerCurve >

* + 1. In the end we changed in diesel our freqPowerCurve to PowerCurve because it would be much easier to control a big smart grid with multiple stabilizers. The freqPowerCurve is now located in the Central eNode (check other point 3)

<PowerCurve>[3000 2200] </PowerCurve >

* 1. The last problem we had was with interp1. We tried a lot of combinations and we did not get why we always interpolated the wrong values, types of curves (wrong dimension when we had too many values see 1.4.2). In the end when we reached 1.4.3 the interpolation was easy.

1. In the nodes of the wind turbine and the solar panel we struggled with receiving the price from the central eNode and sending a power and frequency back to the central eNode:
   1. The first plan was to receive a price and then to look for the corresponding power, this power then looked for its corresponding frequency. But it is better to link the price and the frequency together so it is easier the calculate the stability deviation of your grid (also if you do this first in your grid the program knows faster the power that is needed for stability)
   2. So solution was € 🡪 f 🡪 power
2. When we finally got a working grid the problem was that we could only solve the stability with one stability controller. Because this stability controllers always solved the problem for the entire grid. This is why we relocated the freqPowerCurve to the central eNode so this curve would not be for one stabilizer but for stabilizing the entire grid. In the stabilizers itself we introduced the line PowerCurve. The next problem we encountered was that if the water level was high enough we first wanted to stabilize the grid with the hydro-electric power (because it is also renewable energy) and after this use the diesels. We solved this problem with loops.
3. After making some enhancements to our program we saved it at a different location but after a lot of enhancements our program was not had a lot of not used parts and we needed to clean up the files by relocating them. We made the maps main program, utils and work (for example in utils are all the functions located). Because we were doing this we got some problems with the location of some code, we first try to solve the problem with change directory (“cd”) but in the end we used most time addpath.
4. First we tried in the GUI that you could assemble your own grid. But because we realized that it was too hard to program it (because it is a loop in a loop in a loop) we decided that we make an amount of possible grids and store them in the folder gridTypes and just go and work in matlab with a switch and cases of your gridtype.
5. We also had a small problem with our billConsumers only going up instead going down. The problem here was that we consumed too much liters of diesel or water while we gave a little amount of power. The solution was easy we needed to change the fuel- and water consumption to a lower and more realistic amount (this was because we changed it first to a higher amount to test something).
6. While raising the price we also had a few problems because we stored our possible solutions in a matrix with max 500 possibilities (we choose this amount because the program takes too much time otherwise) but because our was too small we didn’t get a good result.
7. We also had the problem that we sometimes found a good solution but that it was more than the consumer could handle. Our solution was that we put a restriction in the possible solutions (the possible solution couldn’t be higher than 10 % of what the consumers wanted. ) Why 10%? Because we decided that this 10% at least could be stored in the batteries (unfortunately we didn’t had the time to implement the batteries in our smart grid yet!).
8. We use the function rand to create a random amount of power what the consumers want.

If the consumers want a small amount of price (example 200 W) the producers are going to produce this amount but the stabilizers are always going to produce a lot more and you won’t have any good solutions (see the 10% rule). A possible solution is to restrict the min amount of rand.

1. First, we implemented the loop that searches the optimal solution as a recursive function that passes to itself the price as a parameter. The parameter passed through that function was the price, which was raised each time the function was called. After some trials we discovered that when letting the function run it stopped at the 500 iteration. MatLab defines the limit of recursions at 500 iterations, and also is not the better way of program a function which is being executed so many times. Therefore, we changed the looping mechanism.
2. When the way of finding the optimal solution was changed, we realized that we had to change a quite big code document (300 lines at that moment). In order to change and read the code easily, we created several functions and complementary codes instead of making a big monolithic program. Consequently the code was clearer and more readable so changing from a recursive function to a while type loop, which had to do everything needed for searching the optimSolution, were done in a short time.
3. As we didn’t have an operative grid, we could not focus on creating an algorithm for enhancing the grid proper operation. That was one of the main problems we had, while not having all the variables that determines the optimal solution we were not able to define neither the optimal solution nor the tunings we had to make for going from a possible solution to the optimal one.