Version 8.2

There are several aspects of the testing directory. It requires a kmcos installation to use, at present, because it requires a kmc\_model. That is why we have a “throttling\_test\_reaction\_local\_smart” which actually has a kmcos model inside of it. The idea is that a person will compile a simple kmcos model (first “python throttling\_test\_reaction.py” then “kmcos export on throttling\_test\_reaction.xml”). Then, currently, one goes inside the directory \throttling\_test\_reaction\_local\_smart\ and types “python runPytestDriver.py” or “python runUnitTesterSG.py”.

Cases 11 through 19 are solved manually inside 0-201101\_ManualExamples.xlsx and are solved for the case that FFP\_roof = None (for the case where there is no FFP\_roof).

##Additional Info##

1. We have old testing code and a newer testing code.
   1. Old Deprecated (due to new features that aren’t considered in it): The code test\_throttle\_function.py is used by changing the line near the top which says “from test\_throttle\_case\_7 import\* to the appropriate number (note that means there is also a companion file from which settings are being imported). Then simply run the test by typing “python test\_throttle\_function.py” (in the directory where the kmcos model exists).
      1. Those scripts can be run by typing “bash bash\_test\_throttling.sh” . This is a bash scripting file that runs (currently) cases 1 through 7 at one time.
   2. There is a zip file of testing files that was created before UnitTesterSG calls incorporation. Those are considered deprecated since now we have UnitTesterSG compatible test files.
   3. The newest code with UnitTesterSG is run (currently) by in one of several ways. The individual test files can be run separately, but the intended way of use is to navigate one directory higher up than \throttling\_test\_reaction\_local\_smart\ and typing “python runPytestDriver.py” or “python runUnitTesterSG.py”. Alternatively, one can go into \throttling \_test\_reaction\_local\_smart\ and typing “python tests\_11\_to\_19.py” which currently runs over cases 11 through 21 like it is one test (discouraged).
      1. The newer testing code uses UnitTesterSG and the export import library, which has some advantages and disadvantages. Some of the description in the below points is really about describing how the export import library works, in this context.
      2. If we look for example at test\_throttle\_case\_11\_params.txt, this has a variety of variables defined that will become loaded into the throttling globals module. That is basically what the export import library does: it saves parameters from a globals module (during runtime) and it loads parameters into a globals module when somebody wants to restart (or in this case, test) a simulation/code.
         1. It is worth noting that the throttling globals module does define many variables when it is 1st loaded, so by using the export import library we overwrite the values of some variables as well as potentially introduce/initialize some variables that were not made by the module.
         2. within test\_throttle\_case\_11\_params.txt, we have aggregate\_throttling\_factors\_dict\_old. Those are the aggregate throttling factors before calling the function to calculate the throttling factors.
         3. within test\_throttle\_case\_11\_params\_out.txt we have incremental\_throttling\_factors\_dict which shows that reaction 6p0 is being throttled up. Note that in this file aggregate\_throttling\_factors is shown to have a higher factor for reaction 6p0 relative to aggregate\_throttling\_factors\_dict\_old of this file.
            1. Our testing runfile occurs at a step in the algorithm after aggregate\_throttling\_factors\_dict\_old has been populated (that’s why it is not changed). Additionally, at present, our throttling testing runfile *only* calls the function to calculate the throttling factors, it does not apply them. That is why preexp\_dict is unchanged between the files.
         4. In essence, the best thing to look at is the incremental throttling factors dictionary, and if one desires to compare the aggregate\_throttling\_factors\_dict and aggregate\_throttling\_factors\_dict\_old. One could also compare oEF\_TOF\_list and ptEF\_list. Here we see that ptEF\_list for 6p0 is higher, because we are throttling up for it.