Hey mate, I am Richard ten Hagen and it seems you are following up my project.

I have made a list of a whole bunch of ‘tips and tricks’ which might help you while doing the next project.

[Richardxtenxhagen@gmail.com](mailto:Richardxtenxhagen@gmail.com) if you have any questions.

# General points:

1. Do not use VBA to connect Python to Aspen (check out Christos Abatzis work for that)
2. For any problem first check the total space of possibilities and then calculate how long it would take to calculate 10% of that space. If this is not reachable by you then you should decrease the possibility space. (This is the problem that broke my neck)
3. Global variables are useful when you want to extract some type of data from the Class and save it even after reinitializing the class. This will be necessary since as soon as your aspen/python/vba crashes and you try to revive it reinitialization will be unpreventable. PS you are better of making everything you would ever want to plot/compare a global variable.
4. All variables which you make in one part of class but are being used in another (aka a different function or such) needs to make use of the self.BLABLABLA notation. This will allow you to take the data and save it into the current initialization attempt. BUT: once you need to reinitialize the class they will be deleted.

# Preparing an Aspen file for usage:

In my case it was necessary to change the Aspenfile name in Cell A1 from Excel but since you are hopefully not using that this will not be necessary.

1. **Activate the Aspen program (not the specific file)**

aspen = win32.Dispatch("Apwn.Document")

1. **Initialize the Aspen file**

aspen.InitFromArchive2(os.path.abspath(pathnameshouldbe))

1. **Suppress messages**

aspen.SuppressDialogs = True

1. **Make Aspen not open up when running a simulation**

aspen.Visible = False

# Jesus vs Phoenix

**This kills the aspen program (not just the file) like the taskmanager would.**

dead = os.system("TASKKILL /F /IM Aspenplus.exe")

Ok this is all nice and dandy as long as the aspen doesn’t crash (aka see errorhandeling ‘critical failure’). In this case Aspen will give this message “blablabla critical simulation error, there was a data dump under blablabla ”. I have not found a way to get around this problem since the normal errorhandleing doesn’t allow for the handling of this. To get around this I have the phoenix (aka burn everything to ashes and rise out of that) and the Jesus move (only kill the broken Aspen program and keep everything while resurrecting Aspen). Sadly the jesus move doesn’t work since there are a lot of variables in the Class which would get stuck (Class thinks it has already build three columns but since we made a new aspen the previous three no longer exist in aspen.) The only way to get around this would be if you saved the aspen file after each run and then go back to this old version if there is some failure but I suspect this would not work reliably.

The phoenix (also known as: “We\_areFUCKED\_Iwill\_burn\_everything\_to\_the\_ground(self)”) function will a) kill aspen b) chose a new aspen from the list of possible aspen files. C) initialize it d) suppress dialog e) take class data and add it to the ‘memory’ g) reset the class h) restart the entire training program.

This means that everything is lost from the current Epoch. This is still better then just getting stuck though..

# Errorhandeling

**Normal errors from Aspen**

When Aspen has a problem in its simulation this will usually be shown with the help of two variables.

Error number and Error message. The message is useful to find out what kind of problem it is and if you want to react to this specific error in a certain way you can identify it by the error message.

In VBA it was possible to just use the ‘OnError GoTo’ command. This made it very easy since it would give me the messages as Err.Description and Err.Number.

**Invalid results extracted by aspen:**

When running the program you will notice that there are problems with extracting data from Aspen. The program doesn’t work very well with automated extraction. To get around this I have just set any molarflow to 123456789 if there is some problem. Once I see that the molar flow is 1234457689 I then restart the simulation (aka Engine.Run() ) which usually solves the problem. There are various problems that could exist: A) Aspen data output doesn’t work at all (this can be tested by seeing if the feed molflow is 123456789) B) only the specific block has a problem. This would happen if the feed works but not the current block.   
Usually, B only occurs when a critical failure occurs but I have also seen cases where some mysterious problem caused it. This is also how I was able to identify the point when I had to use the phoenix.

# Multithreading

Ok no matter what you project exactly is: multithreading will be your friend. The problem is the following: no matter how many muscles/amphetamines(aka CPU power) you give your worker at some point the number of hands working at the problem will become the limiting. Python only has one hand which means it does A, then B, then C, then D. If it has a lot of computational power this will be faster but at some point it becomes inefficient and you will want to have 2 workers with each half the CPU. The tricky thing is that ten hands trying to screw one screw doesn’t work and tends to result in errors which means that it is best to keep the two workers in completely separate universes where they cannot give each other strange data and problems. You should also never put the workers in series because then B needs to wait for A to do something and you still don’t get any better performance.

This means that you will need to make it such that you can use multiple aspen programs at the same time with multiple Python programs. Please plan this from the very beginning and do not just notice at the very end that this would have been useful. This includes making it such that it is easy to change one global variable which defines the Aspen file you are working on and also any other reference to the path of anything. Generally all path notations should be kept as a global variable such that you can easily change them.

# Make your own DQN

Since most of my work relied on making aspen and vba and python talk to each other I didn’t have time to really to deep into the whole project of making the machine learning part very complicated. I hope that you will be able to use my code to make your life a bit easier. One of the problems that I have is that the entire process of training the DQN is a black box for me. This is very bad since it forced me to accept that the agent choses actions which are illegal. If you are able to get into “Torch” a bit more you are able to learn a lot more about how to prevent the agent from ever choosing those actions. This is because when you are inside the training program there is the predicted action reward for each action and in my case the program just choses the one with the highest reward. If you are able to edit this program, then you are able chose the action from the list of legal action. We already know what this list of legal actions is since we use this list to find it the action is legal (but in my case only after the black box chose it which means the black box now thinks that this will result in some kind of reward for the system)

# Current structure of the code:

Currently my code is absolute trash since its layers of layers of bugfixing on top of each other. I hope to be able to make some functions which you can then use as building blocks for your work and thereby mitigate a lot of the problems which I had to deal with.

Economic functions()

Class Env()

Def Init()

This is to pass some data into the class which you will need for the calculations inside of the class.

Clean aspen sheet

Generate strings and clean strings

Def Findchoices()

Checks if A) does the stream exist and if B) Stream is not connected.

If it is this means that this could be a possible feed for the next distillation column

Def randomechoice()

This is an old code which is not being used right now. This was used to make a choice when it comes to lightkey and feed and it should be random. It can be used to test your code though without needing to use the whole DQN stuff.

Def InputChoices()

Intakes the action choice(which is a integer which goes from 0-57 or so) and converts it to Butane-01 and B1T.

If it’s the first column then makeProcessandRunwill() do it automatically since there is no real choice for the feed (since there is only one) This means that this will remembered as “Initial Feed”

Def makeProcessandRun()

Add\_DSTWU() adds the column but without anything within it

Add\_streams() adds the streams

Rememberes that this stream was made

Also inputs the values for the streams in the initial feed if Bcounter = 1

Connect\_streams(Bcounter, Nextfeedname) yeah connect streams

Destillationinput\_RUN(Bcounter, Reflux, Pressure, Temp…)

Inputs all the data into the Column and then runs the simulation

Def GetOutputs()

EXTRACT the Blockproperties like ['GetDistil\_Temp', 'GetBottom\_Temp', 'GetREB\_Duty', 'GetCond\_Duty', 'GetAct\_Stages', 'GetAct\_Reflux']

If its 123456789 then rerun it

If it is still 123456789 then set it to zero

For each Compound

For each Stream

GetMolflow(Streamcounter, compoundcounter)

If it fails:

Rerun the engine

If it fails 4 times:

Call the phoenix

Remember the total molflow in each Column. This is needed for estimating the size of the column.

Def CheckIfValidaction()

CheckifDestillationErrorexists

Check for Errorcoutner

Check for actionValidity

Def GetReward()

Reset a whole bunch of stuff

calculate the Value per flow for both Versions of the reward

go through all the streams and count the money that we will gain for both versions (MonetaryGainV1, MonetaryGainV2)

add the number of stages and reflux and temp which is needed for the economics. And also deal with it if it breaks.

Call the Operational cost function.

Call the column size determination function.

Call the Capital cost function.

Calculate the Punishment

Calculate TOTALREWARDVersion1 and 2,

Implement HardMaxNegReward

Def GimmeAllMyPrintOut()

Print all the values for a lot of the stuff. This is your best friend as soon as you are ` trying to debug the whole system. Its useful to see what it actually happening.

Def remeberAction()

Yeah does pretty much what is said. Remembers actions as a array

Def reset() **SUPER IMPORTATNT**

This function is needed for the DQN system which is used to return the class instance to the original state as soon as you have reached 6 columns. (this is actually used as soon as the ‘done’ which is returned in the step function is ‘True’)

A lot of stuff is reset here. And the sheet is cleaned

Def step() **SUPER IMPORTATNT**

OK this is a function which will be used by the DQN. It will intake a action as a integer, and it needs to have “return self.state, self.reward, done, info” as its outputs.

This pretty much means that this function will call all the different functions which are needed to convert the action into the reward. DQN will use this function and call it itself.

Inputchoices()

If action is illegal

Give punishment because you made illegal action (if custome DQN you don’t need it)

Keep old state

If action is legal

MakeProcessandRun()

GetOutputs()

CheckIfValidaction()

GetReward()

Findchoices()

Rememberaction()

Reward = TOTALREWARDVERSION1 or 2

If Reward is too small then set it to be HardMaxNegReward

Bcounter = Bcounter +1

if Bcounter = 6 then done = true

return self.state, self.reward, done, info

def render() VERY IMPORTATN BUT I DIDN’T USE IT…

pass

def RestartEverythingNow I don’t think I every used it…

def We\_areFUCKED\_Iwill\_burn\_everything\_to\_the\_ground()

THIS IS THE PHOENIX… the use is already explained…..

Def Add\_data\_from\_previous\_Run()

This is needed since I didn’t use global variabels correctly…

Def Plot\_all\_results()

This is needed to give you results… because usually its hard to see if it is working….

**THIS IS THE END OF THE CLASS**

Def Hey\_lets\_try\_some\_Training() I am not sure if you would need this…

I am using this since it allows the phoenix to call a function which would restart the training of the DQN

,…. BLABALBAL stuff which you will add into the instance of the class

Stable\_env.StableEnv(BLABLABALBAL stuff) This makes the instance of the class and passes all the different variables into it.

Stable\_env.reset()

Aspen.Visible() and suppress the visibility and error messages

Model = Build\_model()

Dqn = build\_agent()

Historyoftraining = dqn.fit( stable\_env, nb\_steps=100, visualize=False, verbose=1 ) THIS IS JUST A TEST TO SEE IT ALL WORK

For loop which goes until each Epoch has been done.

dqn.compile(Adam(lr=LEARNINGRATE), metrics=['mae'])

hist = dqn.fit(stable\_env, nb\_steps=NumberOfSteps, visualize=False, verbose=1)

model.save(savefilename.h5)

loaded\_model = load\_model(savefilename.h5)

Save stuff after each Epoch

Plot a whole bunch of graphs after each epoch

Class Stable\_env(Environment)

THIS IS JUST BECAUSE MIGLEY wanted to introduce it such that in the case of infinit or non defined reward the agent would still get a reward

THIS CODE WILL BE USED TO ACTUALLY DO THE TRAINING:

Hey\_lets\_try\_some\_Training()

OK NEXT PYTHON FILE:

Name: Getting the Reward for Hurist versions-from-01.06.2021

This file is useful since it has the capability of taking an aspenfile and extract the rewards from it.

There is a second code which can take a list of actions and uses those to make aspen files. It also finds the maximum reward.