# Python Pricing Interface: stable\_testing.py

Python stable\_test.py [Options] InputFile ExecutableA ExecutableB

stable\_test.py is a python script which uses scripts run\_pricing.py and pricecompare.py to compare pricing executables and output a set of results. It will attempt to run a series of instructions to achieve this affect:

1. stable\_test.py will select a path to place the result folder and the run folders for the executable environments
2. Pass needed paths and options to run\_pricing.py which sets up and runs the tests
3. Pass run result paths to pricecompare.py which will do an XML and CSV compare and output a series of summary files

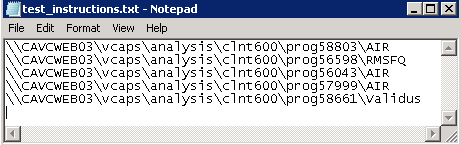
The script will look for run\_pricing.py and pricecompare.py in [\\cavcan03\work\mike\_zhou](file:///\\cavcan03\work\mike_zhou)\pricingtestswc or a specified root folder.

These are the options for this python script; they must be given before any mandatory arguments.

|  |  |
| --- | --- |
| -t | Parallel Executable switch allows the script to test executable A so that it only requires pricing instructions and instructions are run serially to give maximum resources to A |
| -l | [Integer]  Maximum number of threads spawned. Default is 6 |
| -p | [Output/Run Location]  Output and run location for the end results otherwise defaults to \\\\CAVCAN03\\work\\Test\\pricingtest |
| -cwd | Runs the tests in the current working directory with [\\Runs](file:///\\Runs) and [\\Results](file:///\\Results) as subdirectories. |
| -r | [Root]  Specify location of the run\_pricing.py and pricecompare.py locations. Use this for your own branches of the two. Defaults to the [\\\\CAVCAN03\\work\\mike\_zhou](file:///E:\CAVCAN03\work\mike_zhou) folder |
| -h | Prints a help message |
| -c | Performs only a comparison |
| -z | Turn off playoff comparisons |
| -d | [Difference Threshold]  Designate a difference threshold. Enter a number you want as the maximum relative difference to pass. E.G. 0.1 for a 10% difference pass |
| Mandatory Arguments: |  |
| InputFile | Specify location of the instruction file which contains a list of file directories for the programs to be compared. |
| PricingA.exe path | This is one of the two executables to be compared. This should be the experimental branch |
| PricingB.exe path | Other executable. Should be the production branch/ current production pricing executable. |

The current system separated the executable running from the result comparisons into separate python scripts. Stable\_testing.py is simply a way of linking both scripts together to achieve the effects of a single test utility.

An instruction file is a way to compile a list of programs to stress test the programs with a variety of contracts for correctness. It is a simple newline delimited list of network locations of the contract folders



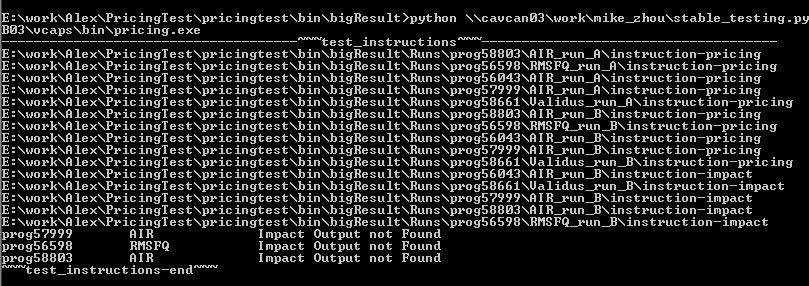
The input file is important in determining the quality of the test. A set of random programs that were compiled recently tend to give a better read of pricing executable integrity than a set of programs from a long time ago. The paths of the programs are listed in any order. These files can be created using the python script newprogramscrape.py which will be detailed later.

Examples of using stable\_testing.py:

Running:

E:\work\Alex\PricingTest\pricingtest\bin\bigResult>python \\cavcan03\work\mike\_zhou\stable\_testing.py -cwd -t E:\work\mike\_zhou\test\_instructions.txt \\CAVCQA01\vcaps\bin\pricing\_20150710.exe \\CAVCWEB03\vcaps\bin\pricing.exe

Outputs:



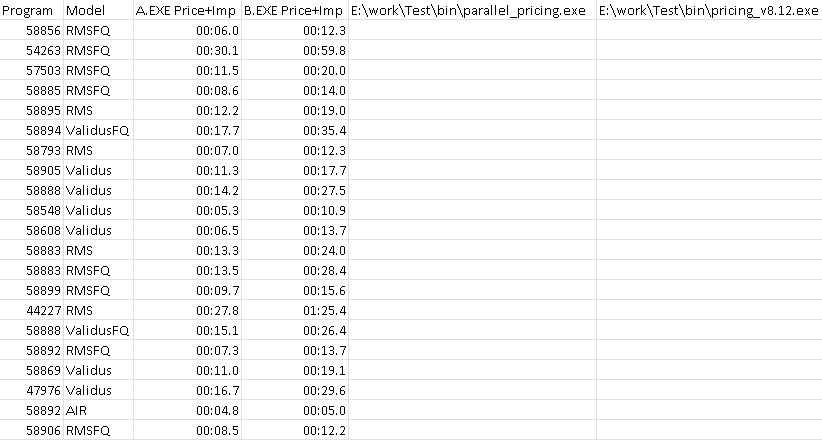
The first half shows which instruction files are being run and where they are being run. The command uses the –t option so during the run only instruction-pricing is being processed through executable A and instructions passed one at a time to prevent too many items on the central stack and affecting performance. Pricing and impact instructions are both passed to executable B. There is no option for only instruction pricing to be passed to executable B at the moment.

The bottom half of the output shows the results of the comparison tests. It only outputs the differences and files not found during the run and which executable experienced the issue.

The individual parts also output files detailing the differences and performance of each executable.

Results:

1. \_Performance\_Summary.txt example:



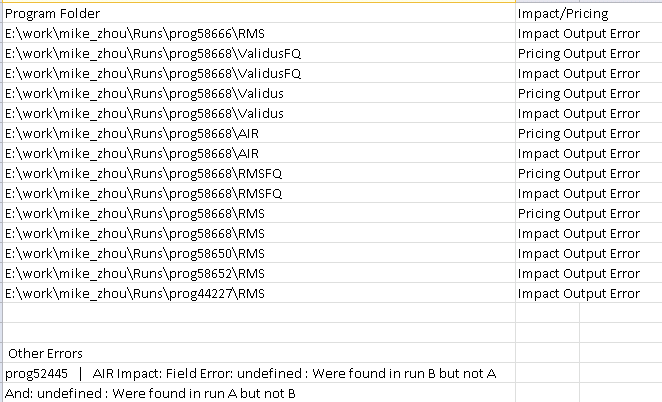
Shown in the performance summary is the run times for every program and model as a sum of impact and pricing analysis run times. Additionally the two executables originally used in the comparisons are listed to mark which versions are tested.

1. \_Difference\_Summary.txt example:



Difference summary contains a compact summary of the differences in the output of the executables. After the program and model, the summary lists the pricing risk groups, fields affected by differences and the total pricing differences and then lists the same categories for impact differences. In the field differences area, it lists the number of differences seen in each field alongside the name.

1. \_error-report.txt



The error file shows which programs didn’t successfully create specific output files and specifies which ones are missing. At the bottom there is a list of errors associated with missing risk groups.

## Price running script: run\_pricing.py

python run\_pricing.py [OPTIONS] InputFile ExecutableA ExecutableB

run\_pricing.py is the first separated part of the python pricing scripts. Before it was mixed in with the comparison scripts but that resulted in a bloated script file that was unfocused and difficult to maintain. run\_pricing.py aims to simplify running pricing executables by setting up the environment and executing and modifying parts of the program to work in a separate environment:

1. Reads the instruction file and checks for executable paths
2. Sets up folders for pricing and impact runs, downloads the instruction files and modifies them to run in a separate environment
3. Executes pricing then impact for executable A and then executes pricing and impact for executable B, gathering data for runtimes
4. Outputs the performance results

Runs are multithreaded to save time and there are multiple options to control the flow of the execution. Like the stable\_testing.py script, these options are given beforehand:

|  |  |
| --- | --- |
| -t | Parallel Executable switch Run EXE A one at a time and have them |
| -l | [Integer]  Maximum number of threads spawned. Default is 6 |
| -p | [Output Location]  \_performance\_summary location default \\\\CAVCAN03\\work\\Test\\pricingtest\\Result\\Most\_Recent |
| -r | [Running folder]  Specifies the location where the runs are done |
| -h | Prints a help message |
| -s | [Source Directory]  Source of programs. Default \\\\CAVCWEB03\\vcaps\\analysis\\ |
| Mandatory Arguments: |  |
| InputFile | Specify location of the instruction file which contains a list of file directories for the programs to be compared. |
| PricingA.exe path | This is one of the two executables to be compared. This should be the experimental branch |
| PricingB.exe path | Other executable. Should be current production pricing executable. |

During execution, the script will list out the instruction files being executed.

## Price comparison script: pricecompare.py

python pricecompare.py [OPTIONS]

pricecompare.py is the python equivalent of the previous executable converted to python to bring it inline with the rest of the functionality at little loss of speed. XML comparisons and differences are generated and tallied into summaries. The output files are also rewritten in a horizontal CSV table rather than a vertical XML format and a difference summary is created to indicate which runs have differences. The migration to python was due to a planned future migration of VCAPS and its functionality to Unix and the c# price\_compare.exe would be unable to be migrated. Price comparison will occur in a couple of steps:

1. Generates a list of programs to check using the instruction file or specified program input
2. Using a specified run folder input, attempts to find the run folders assuming a certain naming structure(run\_folder\prog[prog\_number]\[Model]\_Run[A|B])
3. Runs a payoff comparison using zscore.exe by reading the instruction files in the run folders and finding the payoff files in run\_folder\[payoff\_type][A|B] differences are outputted to stdout and are saved to a file
4. Reads the pricing output in both run folders (A and B)and converts it to a tree structure
5. Compares elements in the tree structure and associates them to the correct contract, risk group and output fields creates a tree of differences. Prints both outputs and differences in programs to CSV tab delimited files in output\_folder
6. Writes a difference and error summary. Difference summary notes the number of differences in each program and how many fields and risk groups are affected. Error summary lists output files not found and issues with mismatching riskgroups and contracts

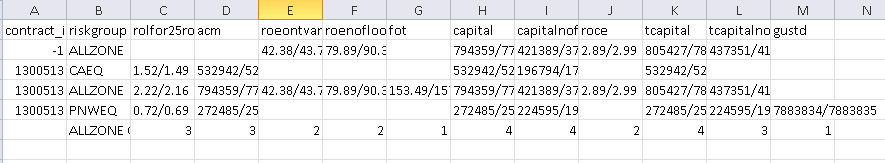
Comparisons are multithreaded to save time and there are multiple options to control the flow of comparisons and which elements are executed.

Options:

|  |  |
| --- | --- |
| -t | [Output location]  Path of summary file output location |
| -f | [File]  Location of list of programs to compare in newline text format |
| -p | [programs]  Specify a specific program to compare |
| -r | [Running folder]  Specifies the location where the runs are done |
| -h | Prints a help message |
| -d | [Difference Tolerance]  Provide a relative error factor in decimal form |
| -z | Prevents zscore csv comparisons to speed up XML pricing/impact output |

Example output:

Diff\_pricing\_prog58152\_ValidusFQ.txt

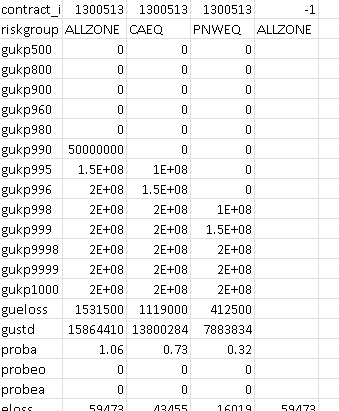


The diff\_pricing and diff\_impact files give a list of which fields are different by providing the differences in a [Value of run A]/[Value of run B] format.

There is a bit of leeway given to programs allowing up to 0.0001 (1E-4 or 0.01%) relative difference between the programs. This number can be changed directly using the –d option and specifying the difference needed.

The last line lists all of the risk groups affected and then lists the number of times a field is affected.

Prog58152\_ValidusFQ\_runA\_pricing.txt



The prog[number]\_[model]\_run[A|B]\_[pricing|impact].txt files present the impact and pricing outputs in an easier to understand format than the traditional XML outputs.

# Program List Creation: newprogramscrape.py

python newprogramscrape.py [OPTIONS] Days Programs

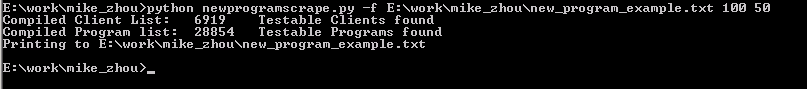
This is an auxiliary script used to generate instruction files. It does so by performing an in-order traversal of the folder containing the clients and programs and determines which programs are suitable for testing by their creation date of their instruction files. It then uses a random selection algorithm to create a list of programs fitting the specified size and outputs to a file or command line.

Options:

|  |  |
| --- | --- |
| -f | [File Location]  Location of file to write list to. Otherwise defaults to current working directory cwd\test\_instructions.txt |
| -d | Dump the programs to stdout instead of file, allowing one to check the programs and then save it when needed |
| Mandatory args |  |
| Days | [Number of days]  Specify the oldest program date. If a programs from last 100 days are needed 100 is the input here |
| Programs | [Integer number of programs]  Specify the number of programs to find. If a larger number of programs is needed (Greater than 300) then it’s a better idea to split the files into smaller ones to prevent very slow tests |

Example usage:

python newprogramscrape.py –f E:\work\mike\_zhou\new\_program\_example.txt 100 50



Output indicates that there are quite a number of clients and programs. These values simply list the number of programs currently on the file system [\\cavcweb03\analysis](file:///\\cavcweb03\analysis)

Example output:

\\cavcan03\work\mike\_zhou\new\_program\_example.txt



All the folders are listed by their location on the network. When using the –d dump option, the contents of the file will be outputted to command line in a similarly formatted way.

# Program specific testing: program\_test.py

python program\_test.py [OPTIONS] “List of Programs” ExecutableA ExecutableB

This is a script used to test specific problems with an executable. This will take an inputted list of programs and run tests for it. Very similar to stable\_test.py in structure and functionality but this allows for a more focused test scenario to be generated.

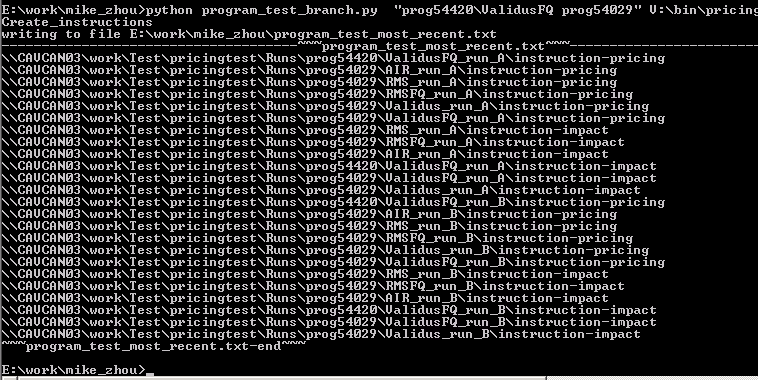
It indexes the source directory [\\cavcweb03\vcaps\analysis](file:///\\cavcweb03\vcaps\analysis) by program number and then attempts to find the specified program numbers and models. If a model isn’t given with a program number, it will just add all available models

There are a few options all common with stable\_testing.py

|  |  |
| --- | --- |
| -t | Parallel Executable switch allows the script to test executable A so that it only requires pricing instructions and instructions are run serially to give maximum resources to A |
| -l | [Integer]  Maximum number of threads spawned. Default is 6 |
| -p | [Output/Run Location]  Output and run location for the end results otherwise defaults to \\\\CAVCAN03\\work\\Test\\pricingtest |
| -cwd | Runs the tests in the current working directory with [\\Runs](file:///\\Runs) and [\\Results](file:///\\Results) as subdirectories. |
| -r | [Root]  Specify location of the run\_pricing.py and pricecompare.py locations. Use this for your own branches of the two. Defaults to the [\\\\CAVCAN03\\work\\mike\_zhou](file:///E:\CAVCAN03\work\mike_zhou)\\pricingtestswc folder |
| -h | Prints a help message |
| -c | Performs only a comparison |
| -z | Turn off zscore comparisons |
| -d | [Difference Threshold]  Designate a difference threshold. Enter a number you want as the maximum relative difference to pass. E.G. 0.1 for a 10% difference pass |
| Mandatory Arguments: |  |
| Programs | Specify which programs that should grabbed. If multiple programs are wanted then they should be provided in parentheses.  For example prog54123 is valid and so is prog20541\RMSFQ and  “prog54123 prog20541\RMSFQ”. Lists are space delimited |
| PricingA.exe path | This is one of the two executables to be compared. This should be the experimental branch |
| PricingB.exe path | Other executable. Should be the production branch/ current production pricing executable. |

Example usage:

python program\_test\_branch.py "prog54420\ValidusFQ prog54029" V:\bin\pricing.exe V:\bin\pricing.exe



In this example the script finds prog54420\ValidusFQ and all the models under prog54029 and writes them to a file program\_test\_most\_recent.txt located in the current working directory. It then runs run\_pricing.py and pricecompare.py like stable\_test.py and passes in all needed argument. The instruction file generated can be used in stable\_test.py, run\_pricing.py and pricecompare.py.

Using the –c option is interesting for this script as it allows certain comparisons to be run regardless of what ran them before. By pointing to the correct run file, -c can be used to make sure the bugs for certain programs and models are solved before applying it to a large scale execution.

The output for this program is the same as the output for stable\_test.py.