Automated Verification Framework

Verification Framework:

The verification framework is a collection of MMS tests that can be performed to evaluate the accuracy of a solver. The MMS problems are solved for different spatial and temporal resolutions to evaluate the order of accuracy of the corresponding derivatives.

A multitude of such tests can be conducted with a relatively small collection of MMS problems. This is done to study the behavior of the solver to perturbations in certain parameters.

Launching such tests manually can be tedious and time-consuming. Hence a automated verification framework was designed.

The steps involved in this testing framework are:

1. Creating the required number of input files from a base file (typically .ups file). The parameters for creating these files are specified in a specially formatted xml file (called .tst file).

2. Running the experiments from the automatically generated input files.

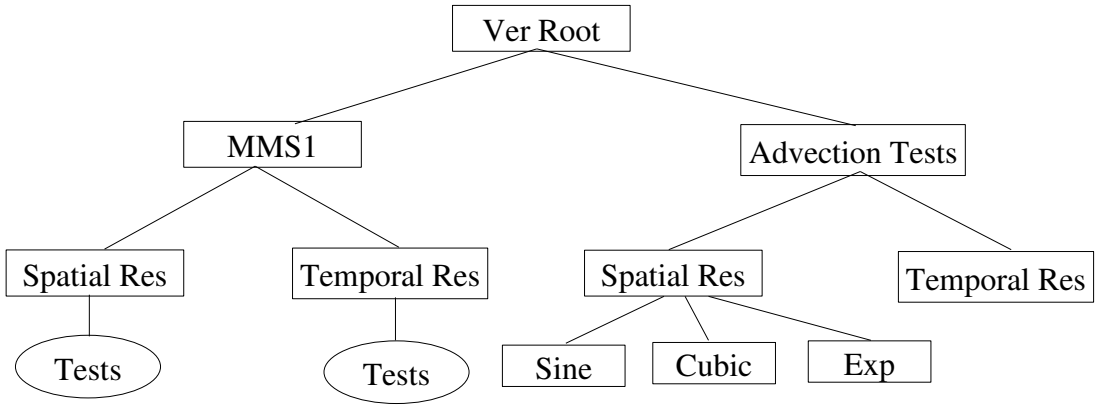
3. Comparing the results from the experiments with exact solutions to estimate the solver.

4. Creating a human-readable report from the test results. For this purpose we compare the current verification run with a baseline.

Stucture of the Verification Framework:

The verification experiments are organized in a network of directories. The network of experiments can be visualized as a tree. The directories can either have sub-directories for further classifications or experiments that are launched or both of them in combination.

Eg: The rectangular boxes are the nodes (sub-directories) and the ellipses are the leaves (the actual experiments).



There are three types of configuration files involved in this verification framework:

1. Organizational configuration: An XML file that describes the directory structure of its child nodes.

2. Experiment Configuration: This is a special-format XML file with an extension .tst. This file describes how a specific parameter should be tweaked. For example, when we want to do a spatial resolution study the

3. Compare Configuration: After the experiments are conducted and the resulting data is generated, it needs to be compared against the exact solution. The compare configuration file will specify how this can be done and what tool needs to be used.

The verification framework consists of three perl scripts.

driver.pl

* This is the main driver script that controls the way in which the experiments are run.

* This takes care of exploring the network (lets call it a tree) of tests and launching the appropriate run_tests.pl.

* This script takes an xml file as its input.

run_tests.pl

* This is the script that creates the modified ups file (and pbs file, if it is a queue job) and launches the experiment (in a queue or a interactive job).

* This script takes a tst file as its input (tst file is just an xml file with a fancy extension for distinguishing purposes).

* This script just runs the experiments (but it doesn't actually compare the results against exact sol).

analyze_results.pl

* This script does the compare_mms (or its equivalent) part.

* But this script is automatically launched by the run_tests.pl script.

* This script also takes an xml file as its input, but it is an auto-generated xml file (which is generated by run_tests.pl).

The input file description:

Input to **driver.pl**

Example:

```
<start>
<Meta>
<Title>Main</Title>
<Email>amjith.r@gmail.com</Email>
<Email>ramanuja@cs.utah.edu</Email>
</Meta>
<testFile>MMS1/MMS1.xml</testFile>
<testFile>MMS2/MMS2.xml</testFile>
<testFile>tmp_sp_res.tst</testFile>
</start>
```

Explanation:

Whatever is inside <Meta> is obvious. Emails will be sent with a report (work needed on the report format, waiting for 'convert' on inferno).

The <testFile> is the real meat of this input file. The testFile can be either a xml file or a tst file. If it is a xml file that means, the driver.pl will dig deeper into the tree of tests.

IMPORTANT: Please use relative paths to specify the location of the <testFile>.

The occurrence of .tst file means we need to launch the run_tests.pl on that tst file. We can mix and match xmls and tsts. For each xml file another instance of driver.pl will be launched (kind of a recursion) and for each tst file a run_tests.pl will be launched which will take care of running the experiments.

Extra Info (just be mindful, nothing to worry):

One thing to note, the emails will be sent when all the test have completed, in order find out when all the tests have completed, the driver.pl file will goto sleep and wakeup every 2mins to check if the tests have completed.

Completion of tests is verified by creating a specific DONE file at the end of each test.

Input to **run_tests.pl**
Example:

```
<start>
<Test>
<Meta>
<Title>res_100</Title>
<upsFile>mms_1.ups</upsFile>
<pbsFile>mms_longrun_mod.pbs</pbsFile>
<Study>Resolution Study</Study>
<compCommand>compare_mms -ice -mms sine -v press_CC -L</compCommand>
<x>100</x>
</Meta>

<content>
       <Level>
           <Box label="1">
              <lower>        [0,0,-0.05]    </lower>
              <upper>        [6.28318530717959,6.28318530717959, 0.05]
</upper>
              <extraCells>   [0,0,1]             </extraCells>
              <patches>      [2,1,1]             </patches>
              <resolution>   [100,100,1]         </resolution>
           </Box>
           <periodic>       [1,1,0]          </periodic>
       </Level>
</content>

</Test>

<Test>

<Meta>
<Title>res_200</Title>
<upsFile>mms_1.ups</upsFile>
<Interactive>sus -ice</Interactive>
<Study>Resolution Study</Study>
</Meta>

<content>
       <Level>
           <Box label="1">
              <lower>        [0,0,-0.05]    </lower>
              <upper>        [6.28318530717959,6.28318530717959, 0.05]
</upper>
              <extraCells>   [0,0,1]             </extraCells>
              <patches>      [2,1,1]             </patches>
              <resolution>   [200,200,1]         </resolution>
           </Box>
           <periodic>       [1,1,0]          </periodic>
       </Level>
</content>

</Test>
</start>
```

Explanation:

The file always begins with a <start> and ends with a </start>.

<Test> </Test>
Each test that needs to be scheduled is enclosed between the <Test> </Test> tags

The parameters for each test are as follows:

<Meta> </Meta>

This is a compulsory flag. This encloses the required parameters necessary for the tests.

<Title> </Title>

Title of the test. This is important because, we append this name to the ups filename when we create new ups files and pbs files. Please use underscore(_) as a delimiter. If you use a space it will be automatically replaced with an underscore.

<upsFile> </upsFile>

This is the name of the base ups file. If the ups file is not in the same directory, please use the full path to the ups file. The base ups file will not be destroyed in any way (it is treated as a read-only file).

<pbsFile> </pbsFile>

If you intend to run these experiments in a batch queue (inferno) please specify the base pbs file name here. Same rules apply as the upsFile.

If you intend to run it interactively (local machine) see next tag.

<Interactive> </Interactive>

These tags will enclose the exact command that you'll use when you run that job interactively.

example :

<Interactive>sus -ice </Interactive>

please don't give "sus -ice something.ups", it WILL CRASH the test. DONOT include the ups file name in the interactive command

The <interactive> tag is mutually exclusive to the <pbsFile> tag. So a test can have only one of the two tags.

<Study> </Study>

This tag is used as a title in the final graph produced by the analyze_results.pl script. This is different from the <Title>, because the <Title> is unique for each test in that file. But the <Study> tag identifies whether that particular test belongs to a study or not. For example all the tests specified in sp_res.tst can be Resolution Study, so <Study> tag for all the test in the tst file will have the words "Resolution Study" (without quotes). Spaces are fine.

<compCommand> </compCommand> [optional tags]

This tag is used to build the xml input file for the analyze_results.pl. This tag has the command that you'll use to run the compare_mms. If this tag is not specified, the analyze_results.pl will not be launched automatically. Only the experiments will be run and it will quit.

<x> </x> [optional tags]

This is also used to build the input xml for analyze_results.pl. This specifies what is the x-parameter for that particular test when we build our graphs using analyze_results.pl. ex: it will be 100 for a resolution test with 100 cells in it.

<content> </content>

This tag is the most important one. The part of the ups file that needs to be replaced should be specified inside the content tags. The way the script works is, it looks for the starting XML tag that appears after the <content> tag and searches for the exact tag in the base ups file provided. Then it looks for the corresponding closing tag in the ups file and replaces the whole section with the section given between the <content> tag. It is imperative that the next line after the <content> tag be not a blank line or a complicated tag like <box label = "solid">. The tag must be a simple tag for example : <Level>

Also the content MUST have more than one line. A single line replacement is not
possible.

Input to **analyze_results.pl**

run_tests.pl will auto-generate these files. It will create one xml file for each test in the tst file. The explanation of each tag in this file is given (if the need arises to create one manually).

Example:

<start>
<Test>
<Meta>
<Title>Resolution Study</Title>

<Interactive>compare_mms -ice -mms sine -v press_CC -L</Interactive>
<Launcher>res_study.tst</Launcher>
</Meta>
<x>100</x>
<udaFile>mms_1_res_100.uda</udaFile>
</Test>
</start>

Things inside the <Meta> tag.

<Title> <Title>

This is the <Study> tag from the tst file. It is important to match the same name that is given in the <Study> tag in the tst file.

<Interactive> </Interactive>

This tag has the command that will be used to run compare_mms or an equivalent utility. Don't specify the uda file, it'll be auto-filled by the script.

<Launcher> </Launcher>

This tag has the name of the tst file that spawned this little xml file. This is for the framework purpose only.

<x> </x>

This is the x parameter for the first uda file

<udaFile></udaFile>

Name of the .uda file that needs to be analyzed.

Now if we want to analyze more than one uda file we dont' have to create multiple xml files corresponding to each uda file (this is for manual creation).

Just list the udaFiles one by one like the following:

<x>100</x>
<x>200</x>
<x>300</x>
<x>400</x>
<udaFile>mms_1_res_100.uda</udaFile>
<udaFile>mms_1_res_200.uda</udaFile>
<udaFile>mms_1_res_300.uda</udaFile>
<udaFile>mms_1_res_400.uda</udaFile>

The x-parameters must correspond to the order in which the udaFiles are listed.