**Testing ‘reflectivity\_calculation’**

There are 3 different experiment types which need to be tested:

Standard\_layers

Custom\_layers

Custom\_XY

In addition, there are three different parallelisations of each. This leads to 9 paths through reflectivity calculation in total.

|  |  |
| --- | --- |
| standard\_layers | Single Cored |
|  | Contrasts |
|  | Points |
| custom\_layers | Single cored |
|  | Contrasts |
|  | Points |
| custom XY | Single cored |
|  | Contrasts |
|  | Points |

Additionally, each of these needs to be tested in matlab and mex. So, ***there are 18 tests that need to be run in total.***

***Testing wrapper function.***

Reflectivity calculation has four inputs and two outputs

[problem,result] = reflectivity\_calculation(problemDef,problemDef\_cells,problemDef\_limits,controls);

The inputs and outputs will differ between problem types, but should be the same regardless of parallelisation or compilation. The testing strategy should first test single cored Matlab for each experiment type, and then ensure that the other 5 calculations for each type give the same answer.

Typical inputs for each type are in the /testSuite/reflectivity\_calculation folder. These are customLayersInputs.mat, standardLayersInputs.mat and customXYInputs.mat . Each one contains a struct which has the relevant inputs for each type.

There is a new wrapper function, which accepts 4 inputs, and two other parameters:

whichParallel – choose with parallelisation. Must be ‘single’, ‘points’ or ‘contrasts’

useCompiled – Boolean true or false. Chooses whether the matlab or mex is run.

An example of how to launch this is in ‘usingTests\_script.m’, as follows…..

% How To Use the Test Files

% Start by loading in one of the test inputs

% Load inputs for standard layers

theseInputs = load('customLayersInputs.mat');

% Split the input into the variables needed to call the function

problemDef = theseInputs.customLayersInputs.problemDef;

problemDef\_cells = theseInputs.customLayersInputs.problemDef\_cells;

problemDef\_limits = theseInputs.customLayersInputs.problemDef\_limits;

controls = theseInputs.customLayersInputs.controls;

% The inputs control which type of calculation is tested (standard layers,

% custom layers or cistom XY). Additionally, we need to pass two parameters controlling

% parallelisation, and whether to use compiled or not....

whichParallel = 'contrasts'; % Test the single threaded calculation

useCompiled = false; % Test the matlab code

[testOutProblem1,testOutResult1] = reflectivity\_calculation\_testing\_wrapper(problemDef, problemDef\_cells,problemDef\_limits,controls, useCompiled, whichParallel);

% Now run it again using the mex

whichParallel = 'single';

useCompied = true;

[testOutProblem2,testOutResult2] = reflectivity\_calculation\_testing\_wrapper(problemDef, problemDef\_cells,problemDef\_limits,controls, useCompiled, whichParallel);

% Asuucessful test should give the same outputs for both...

success = isequal(testOutProblem1,testOutProblem2);

if success

disp('Comparison passed');

else

disp('Comparison failed');

end

In this example, we load the inputs for ‘custom layers’, and call two examples of the calculation, one with mex and one with matlab. At the end we can see that the returned outputs are equal. This kind of thing is the basis for the testing strategy. We need to confirm that all 18 compinations of inputs / parallelisation / compilation run successfully.