Hi Abdullah,

For a brief start read this.

# Overview

I have implemented parts of 2 sets of classes so far the first is the surface class. This will be methods for generating, editing and analysing surface as well as reading them in from files. With the basic Surface class ultimately you would be able to read in your file do simple operations like fill holes plot it, get summary data like FFTs, roughness etc. There are also analytical surfaces that extend this class (they have all the functionality of the base class but a bit more for uses. With these you will be able to generate surfaces or roughness functions, add them or subtract them and build up a surface that has the properties that you want. Obviously with these you will also be able to add any profile you want on to a measured surface from a file, e.g. adding roughness or removing curvature. Most of the basics are done in this section there are some things that need extending. Most of these are labelled with #TODO (this should be the convention for things that need work)

The other type of classes I have been implementing are solver classes. These will be solvers for any equation no matter how trivial it may seem. I spent a long time thinking about a clean way to allow these to update each other as cleanly as possible. The way it fits together is this:

Say you have a Reynolds solver that you want to use but in between each iteration you want to update the viscosity using a roelands ‘solver’.

First you make a new Reynolds solver object by giving the constructor a surface object (and probably a bunch of other information):

A\_reynolds\_solver=ReynoldsSolver(surface1)

Then you give this object to the roelands solver constructor:

A\_roeland\_solver=RoelandSolver(A\_reynolds\_solver)

The constructor add a reference to the Reynolds solver object to the roelands solver object:

>>A\_roeland\_slover.p is: A\_reynolds\_solver

We also add the roelands solver to a list of solvers held by the Reynolds solver:

A\_reynolds\_solver.updatables.append(A\_roelands\_solver)

This requires a bit of under standing of how python deals with variables, if you have an array called A then write the statement B=A, you do not make a new variable called B, instead you just make a new name for A so when you edit A, B will change too. Thus when we set the Reynolds solver as a property of the roelands solver it just makes the information in the Reynolds solver available to the roelands solver. Likewise the information and methods of the roelands solver are available to the Reynolds solver.

So for this case this infinite call is possible: (I’ll shorten A\_reynolds\_solver to reynolds and likewise for roelands)

reynolds.upadtables[0].p.updatables[0].p.updatables[0]………

What’s the point?

The solver base class contains a function called update all that runs a full iteration of your solver with whatever sub solvers you have included:

For updatable in updatables:

Updatable.update()

The results of this are instantly available to the reynolds solver as they are written directly to it.

The last type of class that I am planning on implementing is a solution regime class that will contain methods like the variable grid method etc. if you have any questions or if you can see any issues with the structure of this please let me know and we can talk though, obviously you have much more experience than me here