The following states what was done and the success/issues that was encountered with the use of cython in the quest for performance increase of the MCR\_LLM solver (i.e. PLM).

# What is cython?

Cython is a python module which translates and optimises python code to/for C++. That is to say, it uses your existing python code to produce a callable C++/python hybrid code file that increases computation speeds up to 1000 folds.

# Using cython

When using cython, one can use several different options to increase his or her computation speed.

First and foremost is the use of variable type definition. For users familiar with office’s visual basic, variable type definition is like stating variables (ex. Dim x as double). If you are unfamiliar with variable types, they specify what each variable is. For instance, if one were to have a variable such as x = 1.09, one would define this variable as a float. The goal of having this type of optimisation is to reduce the time python searches for each variable type. In other words, if the variable is not defined as a float, for instance, python will loop through variable types until it defines what the type of the variable is. When a script uses several different variables, great speed increases can be obtained through this method.

Next is the restructuring of the code to use less module implemented code. Here we consider using basic coding structures such as loops and conditionals to perform mathematical operations. The idea is to reduce complex mathematical operations by converting them into highly optimised basic mathematical operations (ex. Changing matrix operations for loops). Although counterintuitive, the idea is to use cython’s C++ to quickly compute “dumb” mathematical operations. This however is rather complex and can produce slower results then matrix operations. It should not be a first resort option.

Knowing what parts to cythonise and what parts to leave as python is tricky. In order to get the most out of one’s cythonisation, ones code should be well segmented and intense calculation structures should be identified. This can be done by timing each part of the code to determine the longest(s) calculation time(s) encountered during the run. Once this is done, one should try using an non-embedding method and an embedding method. What is meant by this is that one should try producing a cython code only using the intense calculation structure (non-embedding) and compare its speed with cythonised version of the calling structure as well as the intense calculation structure.

# Variant

The following methods were used:

PLM variable type definition-> non-embedded: this variation uses type definition. In this variation, only the PLM class is cythonised.

PLM variable type definition-> embedded: this variation uses type definition. In this variation, the PLM class as well as the its calling structure c\_plm and calling sub-structure s\_plm is cythonised.

PLM variable type definition and code restructure->non-embedded: this variation uses type definition. In this variation, only the restructured PLM class is cythonised.

# Issues

PLM variable type definition-> non-embedded: no increase in speed and overall decrease in speed due to module calling.

PLM variable type definition and code restructure->non-embedded: complicated code restructure makes for slow cython module. Overall speed decrease

# Success

PLM variable type definition-> embedded: net speed increase of 3 folds. However, the overall speed of calculation for larger files is still too long to produce a usable file. However, it is very interesting to note that this method is very little affected by initialisation in comparison the current solver.

# Discussion

During this part of the optimisation process to C++ optimisers where considered. These where Numba and Cython. I have only worked with cython which seems to be the better option of the two according to Philippe Kikongi who has studied both options in depth.

# Future ideas

If used in a Line-wise initialiser, this could lead to better initialisation results for the current solver.