STROOPWAFEL, or Simulating The Rare Outcomes Of Populations With AIS For Efficient Learning (Broekgaarden et al. 2019) is an adaptive importance sampling algorithm. Essentially, it was developed to help us improve the efficiency of simulations of rare astrophysical events, and to enable accurate simulations at a reasonable computational cost, but it can be applied in many other real world problems. This is a small documentation of the package written in python to use stroopwafel.

<u>DEFINITIONS</u>

A few definitions first.

- **Dimension** - It denotes one of the axes of the parameter space. For example (*Mass_1* of the primary star, *a* separation between the stars). These are the axes where we are searching for interesting locations. It has a few attributes, like:
    - *min_value* (float): The minimum value this dimension can take
    - *max_value* (float): The maximum value this dimension can take
    - *sampler* (Sampler): type of sampling for this dimension during the exploration phase. It should be from the class Sampler
    - *sigma_calculation_method* (SigmaCalculationMethod) : defines the function to use to calculate sigma for this variable.
    - *prior* (Prior): A function that will be used to calculate the prior for this dimension
    - *should_print* (bool): If this dimension should be printed to the file

- **Location** - Location would be defined as a point in the N Dimensional parameter space. It is just like saying that in a 2D coordinate system, my location would be {x : 10, y : 15}. Every Location has a dictionary (mapping) of all the dimensions. Additionally, there is also a properties attribute where you could save other properties for the location, for example, its id, or some derived dimension, like *Mass_2* (*Mass_1* * *q*)
    - *dimensions* (dict(Dimension : float)) : dictionary of mapping between class Dimension and its float value
    - *properties* (dict) : A location could have more properties which we dont want to stroopwafelize but still store. For example, metallicity_2 (same as metallicity_1) or ID from Compas. Everything here, would be printed to the output file.
    - *hit_score* (float) : The hit value of the location in the range [0.0, 1.0], describes how interesting is this location [Currently unused]
    - *weight* (float) : A field to store the weightage of this location

- **NDimensionalDistribution -** This is a parent class (container) for any distribution (for example Gaussian). Each of the subclasses must implement the method run_sampler to return the samples drawn from this distribution. Examples of the distribution are:
    - Gaussian : A gaussian distribution over N Dimensions characterized by its mean (Location), sigma(Location) and a free parameter kappa to control the width. Additionally, it also knows the rejection rate; i.e. how often samples drawn from this distribution are over the bounds of any of its dimension

    - InitialDistribution : This draws samples from the initial Sampler specified in each of the Dimension
- **Sampler** - A class for different kind of samplers, for example, uniform, flat, kroupa etc.
- **Prior** - A class for different kind of priors for each dimension

<u>INTERFACE</u>

Having defined the essential elements, now we can expand on how to use it. You need to write an interface that will bridge the gap between stroopwafel and your external program. An example interface between stroopwafel and COMPAS is uploaded for ease. Stroopwafel runs in batches to speed things up. It saves the state of each batch by saving the *number (*a unique identifier), and other properties which you as a user are free to add, since it is a dictionary. Here is a bit more detail on each of the steps:

- To create a stroopwafel object, you would need to tell it:
    - *NUM_DIMENSIONS* = 4 #Number of dimensions you want to samples
    - *NUM_BINARIES* = 100 #total number of systems
    - *NUM_BATCHES* = 2 #Number of batches you want to run in parallel
    - *NUM_SAMPLES_PER_BATCH* = 10 #Number of samples generated by each of the batch
    - *debug* = False #If True, will generate the logs given by the external program (like COMPAS)
- You would need to acquaint stroopwafel with some of the helper functions. These would be:
    - *interesting_systems_method* : [Mandatory] This method should tell stroopwafel which *Location*s are interesting. It receives a batch instance as an argument. As output, it expects a list of Location objects, each of which would be termed interesting (or a hit). You can make use of the batch properties, to get whatever details you want to receive from the current_batch, for example, the files that were written to.
    - *configure_code_run* : [Mandatory] This method will tell stroopwafel what to run externally. As output, it expects a list of strings which would be run in the command line as a subprocess. Therefore, the first element of this list would be the program to run, followed by zero or more arguments. You could, for example, specify with an argument, where you want to save the outputs to.
    - *update_properties_method* : [Optional] This method will be run each time after a certain batch is complete, so that it can update any derived properties, like for example *Mass_2*, or *Metallicity_2* (same as *Metallicity_1*).

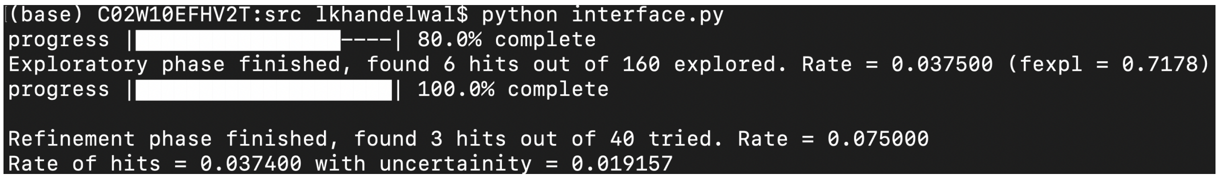
And that's it. Once you are done with the above, its fairly simple to run stroopwafel and easy to understand too.

- *dimensions = create_dimensions()* # Here you specify what dimensions you want to create
- *intial_pdf = InitialDistribution(dimensions)*

- *sw.explore(dimensions, intial_pdf)* #Pass in the dimensions list created, and the initial distribution for exploration phase
- *sw.adapt(n_dimensional_distribution_type = Gaussian)* #Adaptation phase, tell stroopwafel what kind of distribution you would like to create instrumental distributions
- *sw.refine()* #Stroopwafel will draw samples from the adapted distributions
- *sw.postprocess("hits.csv")* #Run it to create weights of the hits found. Pass in a filename to store all the hits

HOW TO RUN

Stroopwafel is written in python 3. So you would need to have atleast that version of python in your system. To run it, just go to your terminal and type *python interface.py,* and you should see something like this.



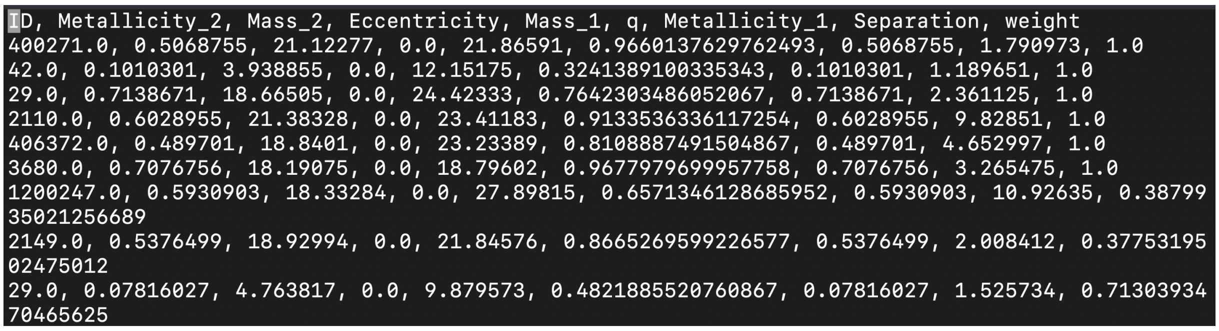
```
(base) C02W10EFHV2T:src lkhandelwal$ python interface.py
progress |████████████████----| 80.0% complete
Exploratory phase finished, found 6 hits out of 160 explored. Rate = 0.037500 (fexpl = 0.7178)
progress |████████████████████| 100.0% complete

Refinement phase finished, found 3 hits out of 40 tried. Rate = 0.075000
Rate of hits = 0.037400 with uncertainity = 0.019157
```

Note: If you are running COMPAS on helios, you would also need to have g++ 7.2+. You can get it through:

- source /opt/rh/devtoolset-8/enable
- source /opt/rh/rh-python36/enable

The output of the run would be saved as the comma separated file, for example.



```
ID, Metallicity_2, Mass_2, Eccentricity, Mass_1, q, Metallicity_1, Separation, weight
400271.0, 0.5068755, 21.12277, 0.0, 21.86591, 0.9660137629762493, 0.5068755, 1.790973, 1.0
42.0, 0.1010301, 3.938855, 0.0, 12.15175, 0.3241389100335343, 0.1010301, 1.189651, 1.0
29.0, 0.7138671, 18.66505, 0.0, 24.42333, 0.7642303486052067, 0.7138671, 2.361125, 1.0
2110.0, 0.6028955, 21.38328, 0.0, 23.41183, 0.9133536336117254, 0.6028955, 9.82851, 1.0
406372.0, 0.489701, 18.8401, 0.0, 23.23389, 0.8108887491504867, 0.489701, 4.652997, 1.0
3680.0, 0.7076756, 18.19075, 0.0, 18.79602, 0.9677979699957758, 0.7076756, 3.265475, 1.0
1200247.0, 0.5930903, 18.33284, 0.0, 27.89815, 0.6571346128685952, 0.5930903, 10.92635, 0.38799
35021256689
2149.0, 0.5376499, 18.92994, 0.0, 21.84576, 0.8665269599226577, 0.5376499, 2.008412, 0.37753195
02475012
29.0, 0.07816027, 4.763817, 0.0, 9.879573, 0.4821885520760867, 0.07816027, 1.525734, 0.71303934
70465625
```