3 Generating Training Points with *Simplex Sampler*

3.1 Summary

Simplex Sampler produces a list of points in the $\boldsymbol{n}-$dimensional model-parameter space to be used for training an emulator. The algorithms are based on the $\boldsymbol{n}-$dimensional simplex. For example, in two dimensions the points are arranged in an equilateral triangle, and for three dimensions of parameters points are arranged in a tetrahedron. The program first reads a simple text file that provides the names of model parameters and the range of their prior distribution. Options for *Simplex sampler* are taken from a separate text file. This enables the User to make choices, such as which algorithm to apply when generating the training points. *Simplex Sampler* determines the number of training points based on the algorithm. The User is free to run the full model at additional points, or to use their own method to generate training points.

3.2 Simplex Parameters (not model parameters!)

These are parameters representing choices made by the User. Note these are NOT the model parameters, which are then generated by *Simplex Sampler*. If one visits the User's project directory, these parameters are stored in the file `smooth_data/smooth_parameters/simplex_parameters.txt`, where the path is either absolute or relative to the project directory. Parameters files can have any name or location. These files are text files in the format. An example of a parameter file is:

```
#LogFileName                simplexlog.txt    # if commented, output to screen
Simplex_TrainType           1                 # Must be 1 or 2
Simplex_ModelRunDirName     modelruns         # Directory with training
                                                        point information
                                                        lea
```

For the parameter file, the first string is the parameter name and is followed by the value. Both are single strings (without spaces). The # symbol is used for comments. Each parameter has a default value, which will be used if the parameter is not mentioned in the parameter file. *Simplex Sampler* has four User-defined parameters.

1. **Simplex_TrainType**
   Possible values are "1" or "2". The default, "1", will position points according to a simplex, i.e. in two dimensions this is an equilateral triangle and in three dimensions, it is a tetrahedron. In $\boldsymbol{n}$ dimensions there are $\boldsymbol{n+1}$ points separated at equal distances from one another and centered at the origin. For "2", points are added at the half-way points between each vertex of the tetrahedron. The points at the bisection points are scaled to a different radius than those at the vertices. This provides the precise number of training points to exactly determine both the linear and quadratic terms.
2. **Simplex_ModelRunDirName**
   This sets the path to the directory in which the run files will be created. The default name is `./modelruns`, but the User can change this to anything they want. The path is relative to the project directory, i.e. the directory from which you run the *simplex* command.

3. **LogFileName**
   If this is left blank, *Simplex Sampler* will write output to the string. Otherwise it will write output to a file. Given that *Simplex Sampler* runs in a few seconds, the program is usually run interactively and output is sent to the screen.

3.3 Specifying Model Parameters and Priors

Before proceeding, Simplex requires information about the parameters, specifically, their ranges. The User enters this information into the file `smooth_data/Info/modelpar_info.txt`. An example of such a file might be

```
NuclearCompressibility  gaussian    210   40
ScreeningMass           uniform     0.3   1.2
Viscosity               uniform     0.08  0.3
```

The first column is the model-parameter name, and the last three parameters describe the range of the parameters, which is usually the prior, assuming the prior is uniform or Gausian. The second entry for each parameter defines whether the range/prior is `uniform` or `gaussian`. If the prior is `uniform`, the next two numbers specify the lower and upper ranges of the parameter. If the range/prior is `gaussian`, the third entry describes the center of the Gaussian, $\boldsymbol{x_0}$, and the fourth entry describes the Gaussian width, $\boldsymbol{\sigma_0}$, where the prior distribution is $\boldsymbol{\propto \exp\{-(x-x_0)^2/2\sigma_0^2\}}$. Simplex will read the information to determine the number of parameters. It will then assign the $\boldsymbol{n}$ points, $\boldsymbol{\theta_{1\cdots n}}$ assuming each dimension of $\boldsymbol{\theta}$ varies from -1 to 1, for uniform distributions, or proportional to $\boldsymbol{e^{-\theta^2/2}}$ for Gaussian distributions. The points $\boldsymbol{\theta_i}$ are each then converted into $\boldsymbol{x_i}$ by scaling and translating the values according to the ranges/priors defined in the `modelpar_info.txt` file.

3.4 Training Types

3.4.1 Type 1

Depending on the number of parameters, $\boldsymbol{n}$, the program creates a simplex in $\boldsymbol{n}$ dimensions. This simplex's vertices will be used to generate $\boldsymbol{N_{train} = n + 1}$ training points. These points will be scaled by different values so the training points aren't in the same radius. This results in the minimum number of required points for linear fits. Thus, if the model is perfectly linear, this option provides perfect emulation.

3.4.2 Type 2

Depending on the number of parameters, the program first creates a simplex in $\boldsymbol{n}$ dimensions. This simplex's vertices will be used to generate new training points there and along the edges. These points will be scaled to be in different radii from the center. This results in the minimum number of required points for quadratic fits. The net number of training points is then $\boldsymbol{N_{train} = n + 1 + n(n+1)/2}$. Thus, if the model is perfectly quadratic, this option provides perfect

emulation. Rather arbitrarily, the points generated from the midpoints of the original simplex pairs are all pushed out to a large radius, and those from the original simplex are brought somewhat inward.

3.5 Running Simplex to Generate Training Points

To run *Simplex Sampler*, first make sure the program is compiled. To compile the programs, change into the MY_LOCAL/main_programs/ directory and enter the following command,

```
${MY_LOCAL}/software% cmake .
${MY_LOCAL}/software% make simplex
```

Next, change into your project directory and run the program.

```
${MY_PROJECT}% ${MY_LOCAL}/bin/simplex
```

Here `${MY_LOCAL}/bin` is the path to where the User compiles the main programs into executables.

Simplex will read parameters from the `smooth_data/smooth_parameters/simplex_parameters.txt` file and from the `smooth_data/Info/modelpar_info.txt` files. It will then write the information about the training points in the directory `smooth_data/modelruns/`. Within the directory, a subdirectory will be created for each training point, named `run0/, run1/, run2/···`. Within each subdirectory, Simplex creates a file `runI/mod_parameters.txt` for the $\mathrm{I}^{\mathbf{th}}$ training point. For example, the `run0/mod_parameters.txt` file might be

```
NuclearCompressibility     229.08
ScreeningMass              0.453
Viscosity                  0.192
```

At this point, it is up to the User to run their full model at each training point and create a file `runI/obs.txt`, which stores values of the observables at those training points as calculated by the full model.

3.6 Replacing *Simplex Sampler* with Other Choices of Training Points

If the User desires to use their own procedure for training points, or if the User wishes to augment the *Simplex Sampler* points with additional training points, the User can write their own files `runI/mod_parameters.txt`. The emulator should work fine, though the User should remember that when training the emulator (see Sec. **??**) the emulator parameter describing with `runI` directories to apply should be modified.

There is one warning to be issued when choosing training points for tuning *Smooth Emulator*. When solving for the Taylor coefficients, the emulator uses the factor of those coefficients (products of $\boldsymbol{\theta_i}$) in the linear algebra routine. However, if those coefficients are not linearly independent, the solution becomes undetermined. For example, if two of the training points were exactly the same, it would fail.