SoFIA: **So**bol-based sensitivity analysis, **F**orward and **I**nverse uncertainty propagation with **A**pplications to high temperature gases

Anabel del Val

25/06/2021

1 Purpose and philosophy of this library

This library was born out of my PhD studies. The code and tools initially available here are the ones I developed for my research. This library is not meant to be an exhaustive library such as *scikit-learn* but a platform where I deposit my tools after cleaning them and making them more user-friendly. If you go online and type something like: UQ python, you'll find a myriad libraries so why developing my own? I also asked myself this question for a long time, and whenever possible, I used libraries that were already available. Libraries that are developed by a large group of individuals are generally better thought out, more efficient, more robust to changes, and they probably already have all you need. SoFIA is not a library for people wanting to do UQ in general. Other libraries are better. SoFIA is meant to fill-in the gap between the aerothermodynamics community and the UQ community.

Aerothermodynamicists in general have many other things to worry about in their research. We deal with complicated models and experimental facilities, and it is not often the case that we have the required mathematical background to do UQ. Therefore, looking for general purpose UQ libraries can be quite scary and burdensome. Our models are complex and so is our data. If you know me, you know that my PhD research was oriented towards developing UQ (in particular Bayesian) methodologies for their efficient use in aerothermodynamics models and experimental data. Based on that knowledge, I would assume you have come to SoFIA because whatever is in here will suit you as an aerothermodynamicist without the need to bend over backwards trying to understand those obscure mathematicians in search for a proper library.

I have often found libraries that do uncertainty propagation or Bayesian inference to be very naive. I'll explain myself. They generally use data formats that are only apt to be used with python functions. They assume the model you want to do UQ on is defined on python, which is generally not the case for our community. Working around these issues often require more work than just coding the whole thing yourself. Yes, such libraries have lots of modules and functionalities but it is not so worth to pay the price of having to adapt your model (that very complex code, made by many people's contributions over many years) to suit those libraries needs. That's the philosophy out of which SoFIA is conceived.

SoFIA contains the methods I needed to use during my PhD, nothing more. The methods are implemented in a way that it is very open to suit different models' needs. In general, SoFIA offers a workflow where you can generate samples of input variables that need to be used by the code of your choice. Then you can come back to SoFIA with the evaluations of the model and use other functionalities like computing Sobol indices or fitting a Gaussian Process model. SoFIA does not handle your model for

you or ask you to format the model in a specific way. This is great for obtaining full flexibility while enjoying the library's methods.

Finally, the purpose of SoFIA is to also grow. If you happen to need to implement a new method, you can do so and merge it to the rest of SoFIA, so that the library keeps growing.

2 Structure

In within the SoFIA library, different modules are defined. This section is structured according to the different modules that can be found in SoFIA.

Create mind map or something and linkages to external functions and codes.

3 Usage

In this section, we describe the basic imports and object calls to use the different features of the SoFIA library.

Sensitivity Analysis (`SA`). This module can be found in `SoFIA/SA`. In there, you can find the `Sobol.py` file with different utilities. We have the `sampling_sequence` which generates the matrices with the inputs that need evaluations from our model, and `indices` which takes on the resulting evaluations and compute the first and total order Sobol' indices. You can print out the matrix with the inputs (each row contains all the inputs for one evaluation of the model) and then generate the evaluations the way you prefer. Then you come back and get your indices. The following code snippet shows an example of usage of this module. In `[...]` you can include the body of your code, namely, the function you want to perform sensitivity analysis on and any other relevant characteristics.

```
import sofia.Sobol as sbl

[...]  # Definition of the function to be evaluated

SA = sbl.Sobol() # Instantiation of sensitivity analysis object

samples = SA.sampling_sequence(n_samples,n_variables,['dist
vars'],None)

[...]  # Evaluation of function f on the samples computed

SA.indices(f,n_samples,n_variables)
```

The concrete example with the Ishigami function can be found in `SA/example.py`.

Probability distributions (`Probability_distributions`). This module can be found in `SoFIA/Probability_distributions`. Several utilities are defined in `distributions.py`. There are two implementations: uniform distribution and Gaussian distribution. Both

objects can be used with an arbitrary number of dimensions, although the resulting n-dimensional distributions do not account for correlation among variables. To instantiate a distribution object we prescribe the number of dimensions and the hyperparameters: lower and upper bounds for uniform distributions, and mean and standard deviation for Gaussian distributions. We can then compute PDF values, log-PDF values (useful for inverse problems), CDF values, inverse CDF values (to sample arbitrary distributions based on uniform or Gaussians), and sample from the distribution itself. A code snippet with a usage example is shown hereafter.

```
import sofia.distributions as dist


hyp = [[a,b]] # Hyperparameters for a 1D distribution


D = dist.Uniform(n_dimensions,hyp) or dist.Gaussian(n_dimensions,hyp)
# Instantiation of probability distribution object:  either 1D
uniform or Gaussian


x = np.linspace(-10.,10.,1000)


D.get_cdf_values(x) # get CDF values of the distribution


D.get_samples(n_samples) # Sample the distribution
```

Inverse uncertainty propagation (Iprop). This module is placed in SoFIA/Iprop. Utilities can be found in the script sampler.py with which a Markov chain based on the Metropolis Hastings algorithm can be built. For the instantiation of the sampler we have to specify an initial covariance function covinit, a loglikelihood function fun_in, and the number of samples to burn for the adaptation of the covariance matrix nburn. The chain is initiated with the parameters specified in the utility seed, while the Burn routine adapts the covariance matrix with the presribed number of draws nburn. The function DoStep advances the Markov chain and generates the useful samples to be used in the analysis of the posterior distribution. The script sampler.py also includes two diagnostic techniques: the plotting of the trace through the chain_visual function, and the computation of the autocorrelation function with autocorr. For the chain diagnostics object, we must specify the chain samples and a dictionary that links the position of the variables in the vector of variables used for the chain construction to a variable name for plotting purposes. A code snippet with the typical usage of this module is shown hereafter.

```
import sofia.sampler as mcmc


[...]  # Model to be calibrated and corresponding definition of the
loglikelihood function of the experiments

sampler = mcmc.metropolis(np.identity(n_dim)*0.01, log_lik,
n_samples) # Instantiation of the MCMC sampler
```

```
[...]  # Definition of the initial values of the parameters to be
calibrated


sampler.seed(initial_values_of_parameters)


sampler.Burn()


for i in range(n_samples):
sampler.DoStep(1)


sampler_diag = mcmc.diagnostics(chain,dict_var) # Instantiation of
the chain diagnostics object


sampler_diag.chain_visual(n_plots,var)


sampler_diag.autocorr(n_lags,n_plots,var)
```

Forward uncertainty propagation (Fprop). Under development. For the moment, it includes a simple polynomial chaos construction. Sample-based methods can be built with the functionalities already included in the library.

```
import sofia.pc as pce
import sofia.distributions as dist


# Generate Gaussian samples to use


N = n_samples


# Definition of the distribution used in the PCE construction


hyp=[[0.,1.]]
G = dist.Gaussian(1,hyp)
S = G.get_samples(N)


# Definition of the targetted distribution


hypU=[[0.,1.]]
target = dist.Uniform(1,hypU)


h = target.fun_icdf() # Definition of the inverse CDF function of
the target distribution


ki_uniform = pce.approximate_rv_coeffs(n_polynomials, h)
```

```
k = pce.generate_rv(ki_uniform, S) # With k we should recover the
target distribution
```

Applications (`Applications`). Under development. It will include scripts with which to reproduce the results obtained in my PhD thesis. These scripts will put together different modules from SoFIA to perform all the computations. The only inputs to be provided will be the external model evaluations and inputs from which to construct Gaussian process surrogates and carry out the inverse and/or forward analyses. The data from the simulations I used in my thesis will also be available in another repository.

4 Examples

Particular examples of the usage of the different modules depicted in the previous section can be found in `SoFIA/examples`. Here we describe some of them and show the results for verification purposes.

Sensitivity Analysis (`SA`). Here we work with the Ishigami function

$$f(x) = \sin(x) + a \cdot \sin(y)^2 + b \cdot z^4 \sin(x), \tag{1}$$

for which we compute sensitivity indices for x, y, z with respect to the model output f. The model parameters a and b are set to the values 7 and 0.1, respectively.

Results and plots for the particular examples I have in SoFIA