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

`https://github.com/pysofia/SoFIA`

Anabel del Val

March 14, 2022

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[1] 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. It's true, there are other libraries, such as Sandia Labs' *Dakota*[2], that are completely non-intrusive and can be a good fit for our needs. This is true, however, SoFIA also retains the possibility of defining your own model in python and using the library's methods without the need to encapsulate your code in

[1]In Python, although it also concerns other popular programming languages.

[2]`https://dakota.sandia.gov`

an executable to work within *Dakota*, speeding up the analyses in such cases. Further, SoFIA can also grow and adapt to your needs.

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. The following diagram (Fig. 1) shows the different constitutive parts of the library. More details about the different modules are given in the next section.

- The `Probability distributions` module lays at the base, providing support to the other different modules. It contains both `Gaussian` and `Uniform` classes with different methods in each to compute the PDFs, CDFs, inverse CDFs, log PDFs as well as generate random samples from such distributions.
- The module `SA` refers to the methods associated with performing Sensitivity Analyses and computing Sobol indices. This module uses the methods included in the `Probability distributions` module to specify probability distributions for input parameters.
- The `Fprop` module contains a basic implementation of a polynomial chaos expansion. It also uses the methods included in `Probability distributions` to specify the expansion germ's distributions.
- The `Iprop` module contains all the methods involved in performing Markov Chain Monte Carlo (MCMC) sampling following the Metropolis-Hastings algorithm with adapptation of the covariance matrix through a burn-in phase. Furthermore, it also contains a class of diagnostics to assess the chain convergence. A simple implementation of the Hamiltonian Monte Carlo (HMC) sampling algorithm is also implemented although it has not been tested.
- Apart from the three main modules, SoFIA includes some utilities to make any of the tasks more efficient. This is contain in the folder `Utilities`:
  1. The `MPI` module provides the capability of parallelizing tasks such as evaluating a model according to independently sampled inputs, as well as drawing independent samples from a probability distribution. It uses the python module `mpi4py`.
  2. The module `GPR` refers to Gaussian Process Regression. It implements the different parts of a Gaussian process model, as well as the choice to train

it and evaluate its mean and variance in new points. Both modules can be used in any of the main three modules described above to speed-up computations.

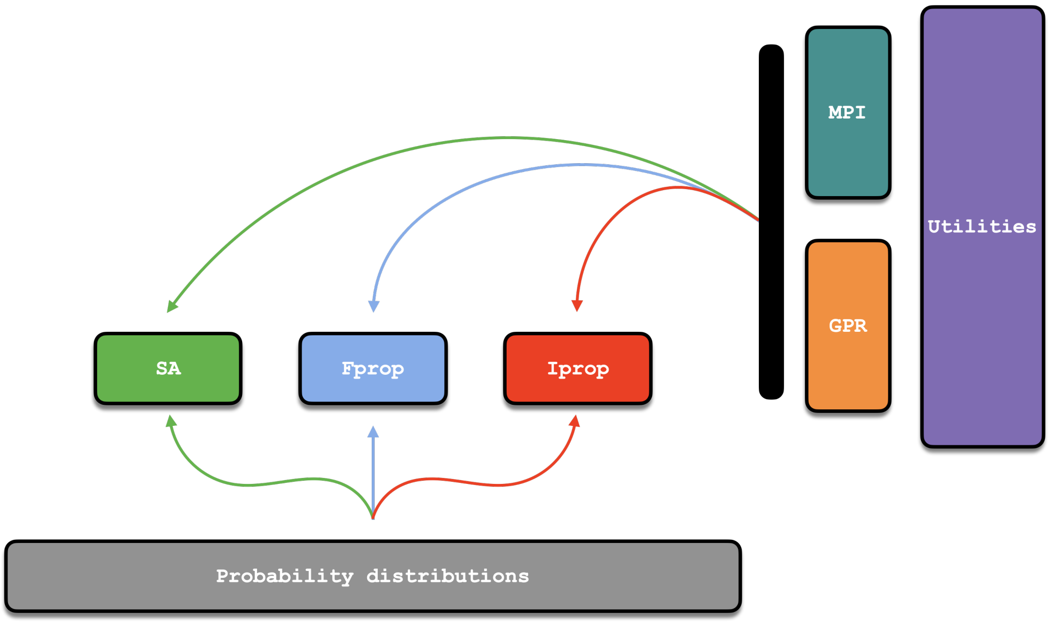


Figure 1: The different blocks composing the SoFIA library.

Apart from the above-mentioned modules, a strong component of the SoFIA library is the applications within aerothermodynamics. In particular, two different modules are included: `catalysis` and `nitridation`. Fig. 2 shows the different relationships among the different blocks. Seldom databases with the GPR models and data, such as parameter priors and measured quantities, are provided for each experimental case. Combining the different methods included in SoFIA, the user has an array of things that can be computed for material and flow characterization as well as diagnostics.

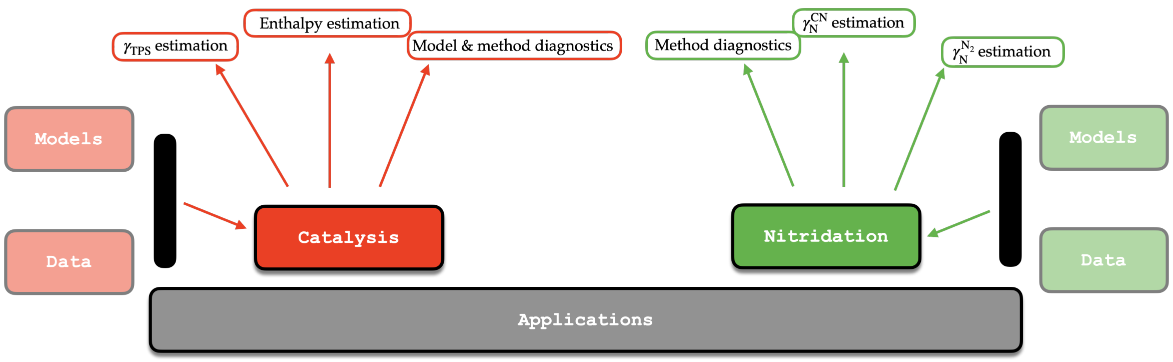

γ_{TPS} estimation
Enthalpy estimation
Model & method diagnostics
Method diagnostics
γ_N^{CN} estimation
$\gamma_N^{N_2}$ estimation
Models
Data
Catalysis
Nitridation
Models
Data
Applications

Figure 2: The different blocks composing the `Applications` module in the SoFIA library.

SoFIA not only offers an account of useful mathematical tools but also a principled and centralized way of storing models and data for the different applications.

3 Usage

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

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
```

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`.

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`). This module is placed in `SoFIA/Fprop`. 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`). This module can be accessed in `SoFIA/Applications` It includes scripts with which to reproduce the results obtained in my PhD thesis. These scripts put together different modules from SoFIA to perform all the computations. The only inputs to be provided are 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 are also available in different `json` files. The applications here included are related to the estimation of catalysis and nitridation parameters for TPS materials, as well as an adiabatic reactor example. For code snippets and examples of their usage, the reader is directed to `SoFIA/Docs/jup_notebooks/env`. That folder contains a virtual environment with the jupyter notebooks `catalysis_demo.ipynb, nitridation_demo.ipynb`, and `Adiabatic_reactor.ipynb`. Some particular examples are offered in the next section.

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.

Probability distributions (`Probability_distributions`). In the example `Probability_distributions/examples/example.py` we compute 1D and 2D uncorrelated uniform and Gaussian distributions. We first sample a uniform distribution with support $\mathcal{U}[-8, 8]$ for which we generate 1000 samples and compute the CDF and PDF.

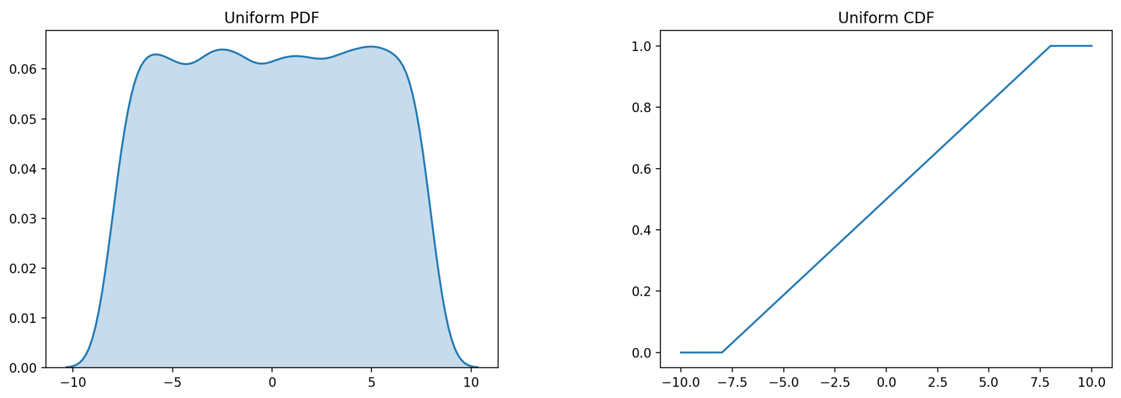


Figure 3: 1D uniform PDF and CDF.

We do the same for a 1D Gaussian distribution $\mathcal{N}(\mu = -1, \sigma = 4)$

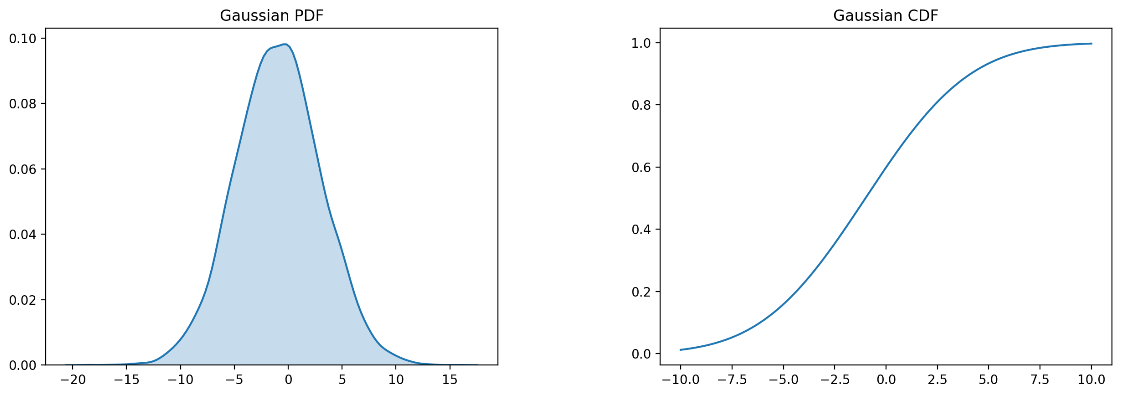


Figure 4: 1D Gaussian PDF and CDF.

We can also compute marginal CDFs and PDFs for 2D distribution of uncorrelated variables. In this case, the uniform distribution has for hyperparameters

$\mathcal{U}[-1,4] \times \mathcal{U}[0,2]$, while the 2D Gaussian is defined $\mathcal{N}(-1,4) \times \mathcal{N}(0,2)$.

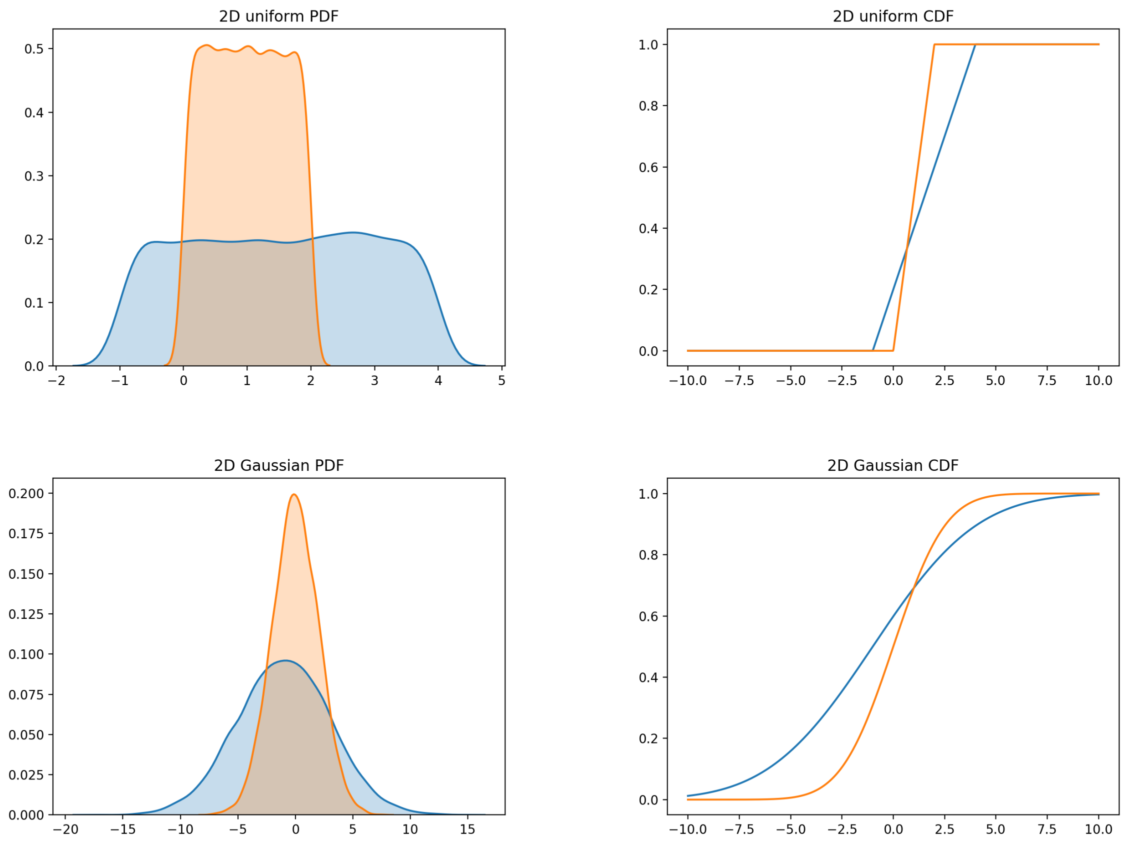


Figure 5: 2D uniform and Gaussian marginal PDFs and CDFs.

Sensitivity Analysis (`SA`). In the example `SA/examples/example.py`, we work with the Ishigami function

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

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. We assume uniform distributions $\mathcal{U}[-\pi, \pi]$ for x, y, z and base our estimations on 10,000 samples.

Table 1: Sobol' sensitivity indices for the inputs x, y, z of the Ishigami function (1).

Variables	S_i	S_{T_i}
x	0.312	0.576
y	0.446	0.443
z	0.009	0.246

Inverse uncertainty propagation (`Iprop`). There are several examples for this module.

Example 1. The first example `Iprop/examples/example_ishigami.py` consists of inferring parameters a and b from noisy observations f of the Ishigami function (1). In this case, the parameters are set to the values $a = 8$ and $b = 1$ while the noise of the observations is set to $\sigma = 0.2$. The results are based on data in 3 different points (x, y, z): $(0, 1, 2), (0, 2, 5)$, and $(1, 8, 5)$. The following figure 6 shows the trace for parameter b and the marginal posteriors obtained for a 100,000 steps chain with an initial burn-in stage of 100,000 samples. The overall acceptance rate is 0.34008.

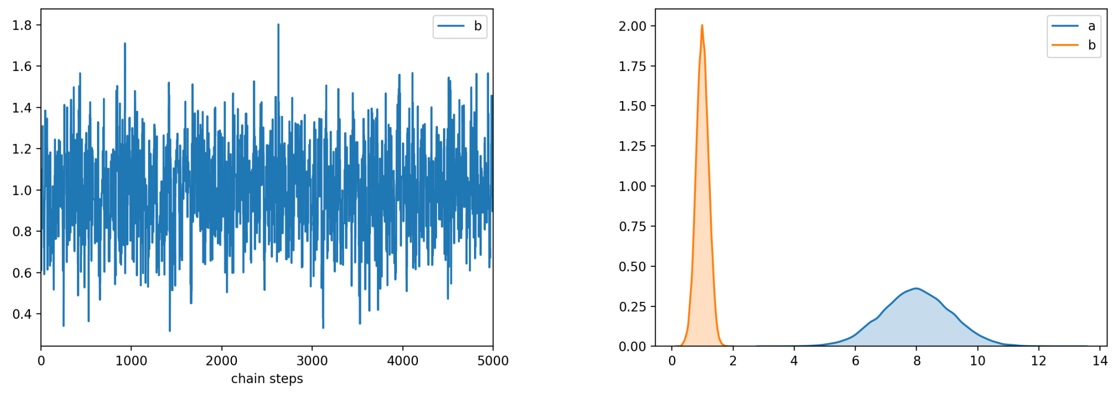


Figure 6: Left: chain trace for parameter b for 5,000 steps. Right: marginal posterior distributions of parameters a and b.

We also show the autocorrelation function in the following figure

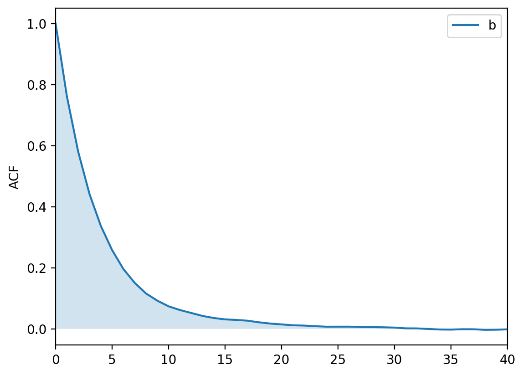


Figure 7: Autocorrelation function of the chain for parameter b.

Example 2. Another example consists on the sampling of two Gaussian distributions by means of MCMC. We sample the Gaussians $\mathcal{N}(0, 1)$ and $\mathcal{N}(5, 1)$ whose true analytical PDFs are compared to the MCMC result through which the mean parameters μ_1 and μ_2 of the distributions are estimated. Figs 8-9 show the results.

Example 3. The last example included in the library consists of the reconstruction of a polynomial of arbitrary degree k, $f(x) = \sum_{k=0}^{q} a_q x^q$ from 30 noisy observations randomly sampled in the canonical range $x \in [0, 1]$ with $\sigma = 0.1$. We first fit the linear polynomial

$$f(x) = 10 - 2 \cdot x \tag{2}$$

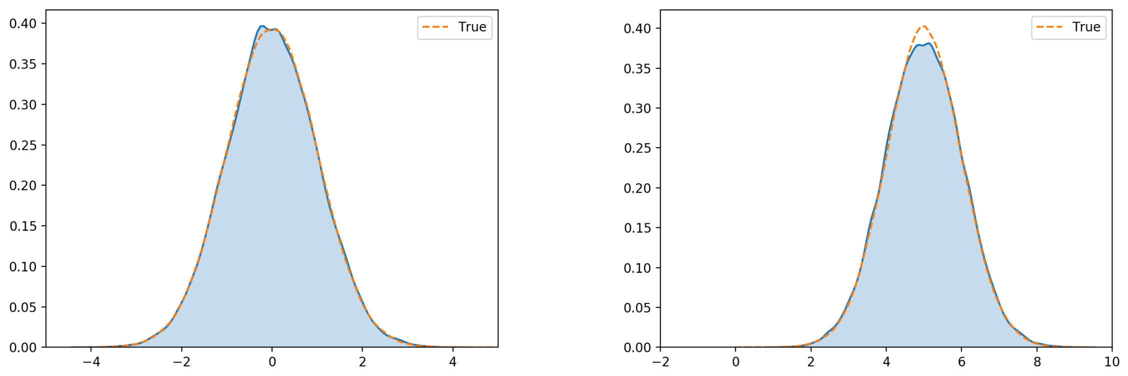


Figure 8: Left: true PDF versus the PDF recovered from the Markov chain samples for the first Gaussian distribution $\mathcal{N}(0, 1)$ Right: true PDF versus the PDF recovered from the Markov chain samples for the first Gaussian distribution $\mathcal{N}(5, 1)$.

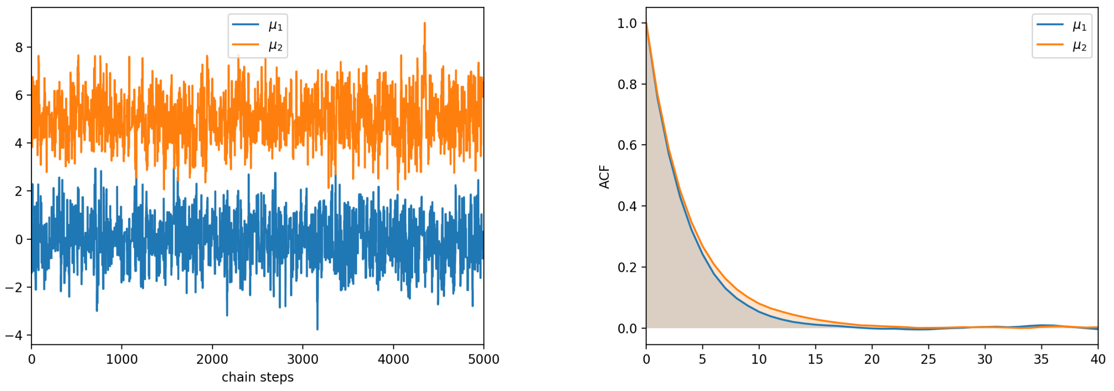


Figure 9: Left: chain traces for parameters μ_1 and μ_2 of the Gaussian distributions we are sampling with a MCMC method for 5,000 steps. Right: autocorrelation functions.

by fixing $a_0 = 10$, $a_1 = -2$ and $a_i = 0, \quad \forall i \in 2, ..., 4$. The results of the posterior predictive check as well as the marginal posteriors of the parameters are shown in Fig. 10.

We then fit a more challenging model, $f(x) = 10 - 2 \cdot x + 7.5 \cdot x^2 - 3.3 \cdot x^3 - 3.2 \cdot x^4$ with the results shown in Fig. 11. Parameter a_0 has been omitted from the marginal posteriors given the more peaky behavior compared to the rest of the parameters, giving poor viasualization of the results when plotted all together. a_0 marginal posterior closely resembles that of Fig. 10.

Forward uncertainty propagation (`Fprop`). In this module, we include a simple polynomial chaos construction. The example included is the polynomial chaos expansion of a uniform distribution based on canonical Gaussian distributions $\mathcal{N}(0, 1)$.

Applications (`Applications`). We include different examples of applications. First, we introduce a test case with an adiabatic reactor of a simple mixture. Examples on the particular applications of my thesis will be also included here.

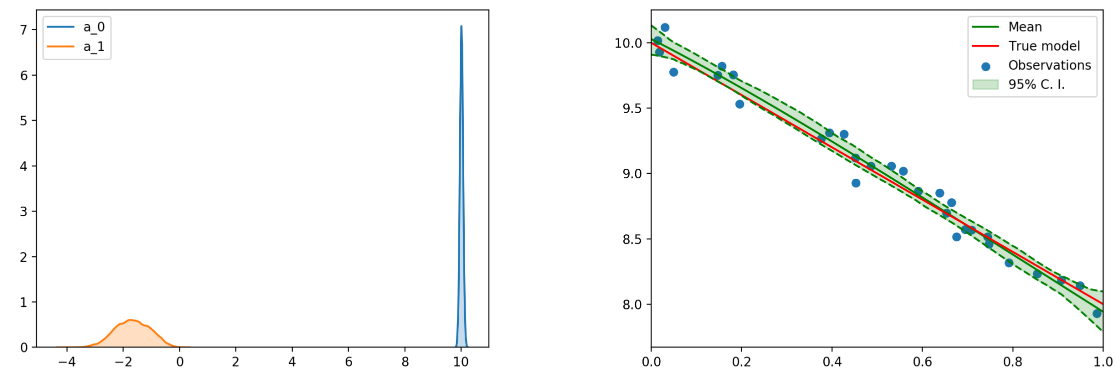


Figure 10: Left: marginal posterior distributions for parameters a_0 (bias) and a_1 (slope) for the linear polynomial model. Right: posterior predictive check.

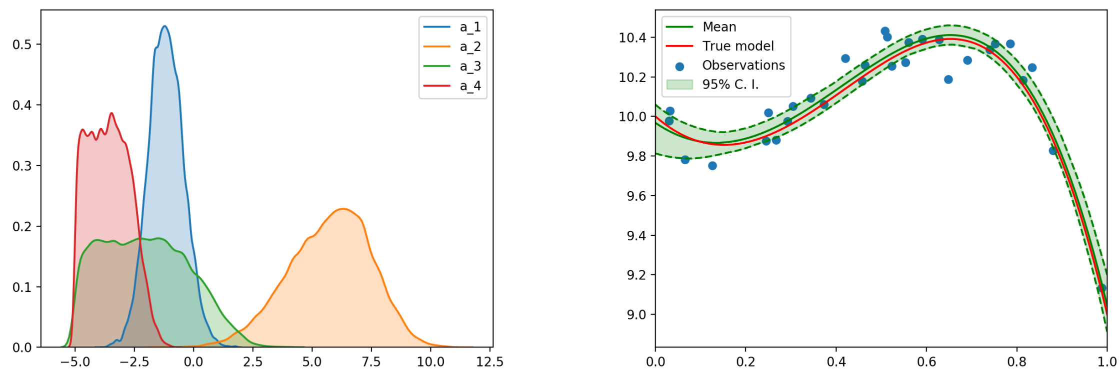


Figure 11: Left: marginal posterior distributions for parameters $a_{1,\ldots,4}$ for the polynomial model. Right: posterior predictive check.

Adiabatic reactor. In this example, we use the python implementation of the physico-chemical library Mutation++[3] to compute the time evolution of the temperature for an air5 $\{N, NO, O, O_2, N_2\}$ adiabatic reactor. The results in terms of the Sobol indices evolution with time is shown in Fig. 4. In the legend, the chemical mechanism considered for such mixture is shown.

For such example, the reaction rates were sampled from log-uniform distributions with the hyperparameters shown in Table 2. They have been chosen to encompass their current value according to the chemical model of Park, and also with a variability that does not produce instabilities in the solver in Mutation++. The Sobol indices are based on a Monte Carlo estimate with 10,000 samples. This example is just to demonstrate how SoFIA can be coupled to other libraries to produce various results. The implementation has been sped-up by using parallelization via the module `mpi4py`.

A forward propagation of the a priori uncertainty imposed on the reaction rates of the adiabatic reactor was also carried out (Fig. 4). Notice that the initial temperature was considered certain in all cases. The contributions to the overall uncertainty shown in Fig. 4 are mainly due to the reactions with the highest Sobol indices shown in Fig. 4.

[3]`https://github.com/mutationpp/Mutationpp`

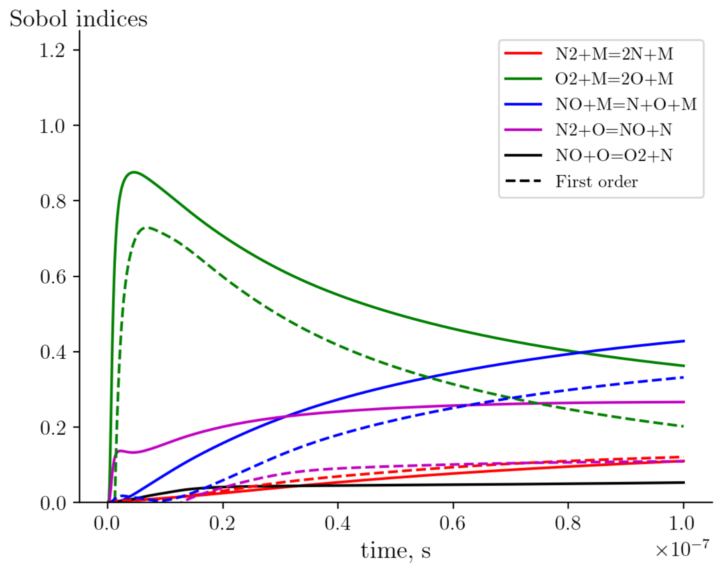


Figure 12: Time evolution of the first order and total order Sobol indices for the variance-based sensitivity of the temperature evolution of an air5 adiabatic reactor at 100 hPa with respect to the pre-exponential factors of the reaction rates of the chemical mechanism.

Table 2: Probability distributions for the pre-exponential factors of the reaction rates of an air5 chemical mechanism.

Variables	distribution
$\log A_{\mathrm{N_2+M \rightleftharpoons 2N+M}}$	$\mathcal{U}[20, 22]$
$\log A_{\mathrm{O_2+M \rightleftharpoons 2O+M}}$	$\mathcal{U}[20, 22]$
$\log A_{\mathrm{NO+M \rightleftharpoons N+O+M}}$	$\mathcal{U}[10, 15]$
$\log A_{\mathrm{N_2+O \rightleftharpoons NO+N}}$	$\mathcal{U}[10, 15]$
$\log A_{\mathrm{NO+O \rightleftharpoons O_2+N}}$	$\mathcal{U}[10, 15]$

If experimental data are available, the uncertainty estimate can be refined and attain objectivity. The following Fig. 14 shows two different cases. The left plot shows the predictive performance of the calibrated model based on one data point, while the plot on the right uses 5 data points randomly distributed in time. The calibration is performed with 10,000 burned samples and a further 10,000 chain samples. The noise of the observation is $\sigma = 100\ K$.

Further, we also include the computed marginal posterior distributions of the pre-exponential factors in Fig. 15. Notice that for the case with 5 observations, the best learned parameter is the one attached to the reaction with the largest the Sobol indices, which dominates the temperature estimation in all the time span. The other coefficients can be known to a lesser degree, some still covering the whole prior range. The plots are given in logarithmic range.

It is also interesting to see that for an adiabatic reactor with a perfectly known initial temperature and a measurement of the temperature at steady-state, NO dissociation is quite well defined, given that it governs the temperature evolution at later times, unlike oxygen dissociation. Fig. 16 shows the posterior predictive and the

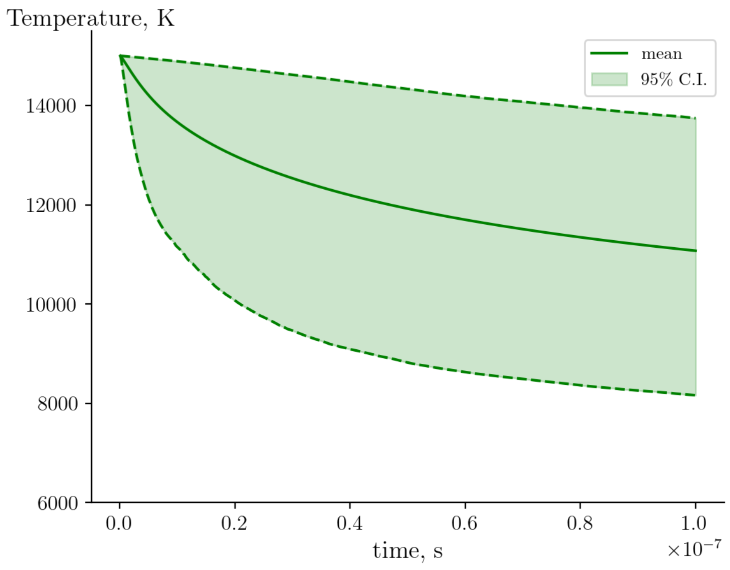


Figure 13: Time evolution of the mean and 95% confidence interval of the temperature of an air5 adiabatic reactor at 100 hPa when reaction rates are considered uncertain following Table 2 distributions.

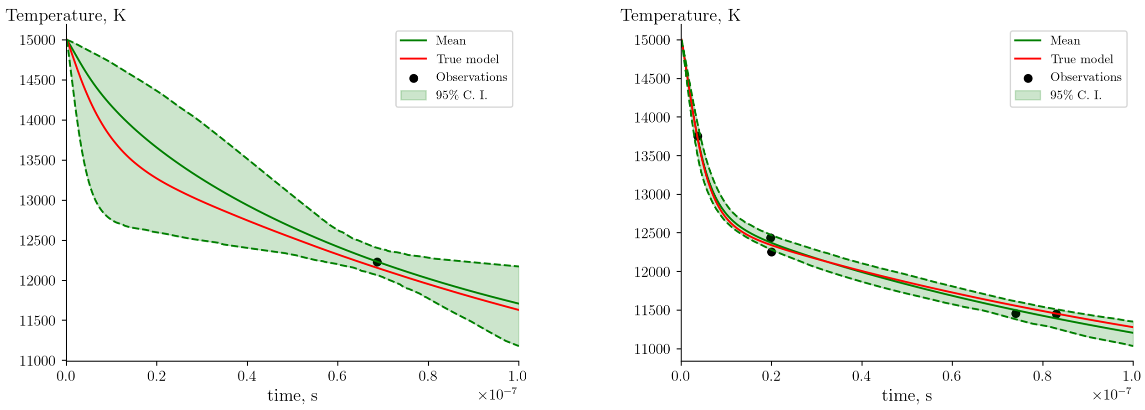
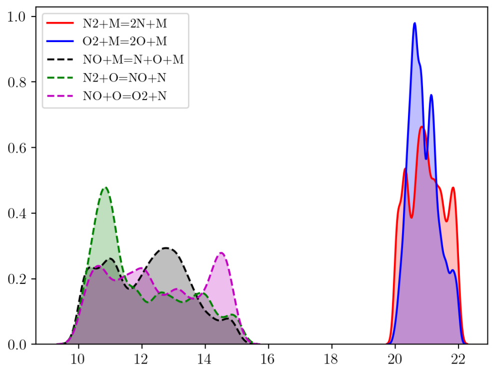


Figure 14: Left: posterior predictive check of the calibrated model with 1 observation. Right: posterior predictive check of the calibrated model with 5 observations.

resulting marginal distributions.

All of the above results assume perfect knowledge of the initial temperature. If we know consider the initial temperature to be unknown within some wide bounds $T_{\text{init}} \sim \mathcal{U}[10000, 15000]$, the latest results of Fig. 16 would now be the ones in Fig. 17. Notice the change in the calibrated model as well as the marginal distributions.

If we were to run a sensitivity analysis again, this time with the initial temperature as a free parameter, we would see the following result of Fig. 4. In it, we can appreciate how the sensitivity indices change values once the initial temperature is considered unknown. The marginal posterior distributions changes can be understood based on the sensitivity of the problem when adding the initial temperature as an additional free parameter.



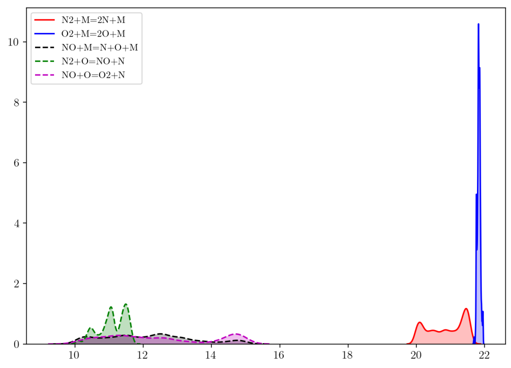
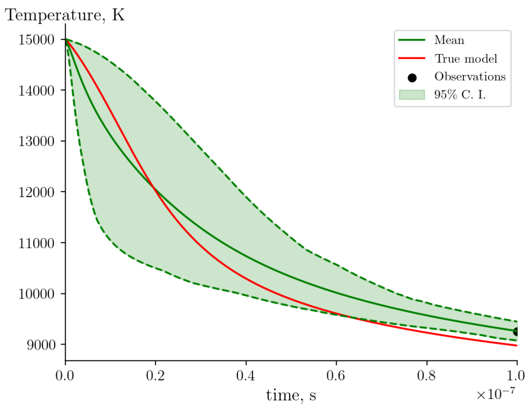


Figure 15: Left: resulting marginal posterior distributions of the calibrated pre-exponential factors with 1 observation. Right: resulting marginal posterior distributions of the calibrated pre-exponential factors with 5 observations.



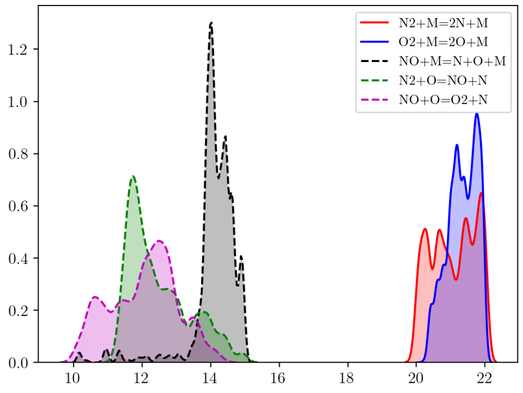


Figure 16: Left: posterior predictive check of the calibrated model with 1 observation at steady-state. Right: resulting marginal posterior distributions of the calibrated pre-exponential factors.

Heterogeneous catalysis for reusable TPS ground testing and enthalpy rebuilding. This application pertains to my PhD thesis work. We calibrate a chemically reacting flow model in chemical non-equilibrium with plasma wind tunnel experimental data. We look for the resulting uncertainty on the catalytic coefficients of the different materials tested for each independent experimental run as well as the free stream boundary condition (enthalpy). For more technical details see footnote[4].

In SoFIA, in the `Applications/catalysis/models` folder, we include the saved Gaussian Process (GP) models for each experimental run of names `MTAt1,2,...`. These GP models are built to approximate the log-likelihood functions of the experiments. In turn, such models are used to run an MCMC algorithm and compute the marginal posterior distributions of the quantities of interest (catalytic efficiency parameters). Such posterior distributions are propagated through another set of GP surrogates with the names `H_case_name.sav` to obtain the rebuilt free stream enthalpy.

[4]A. del Val, Bayesian calibration and assessment of gas-surface interaction models and experiments in atmospheric entry plasmas, IPP/VKI, 2021 - `www.doi.org/10.13140/RG.2.2.27055.82089`

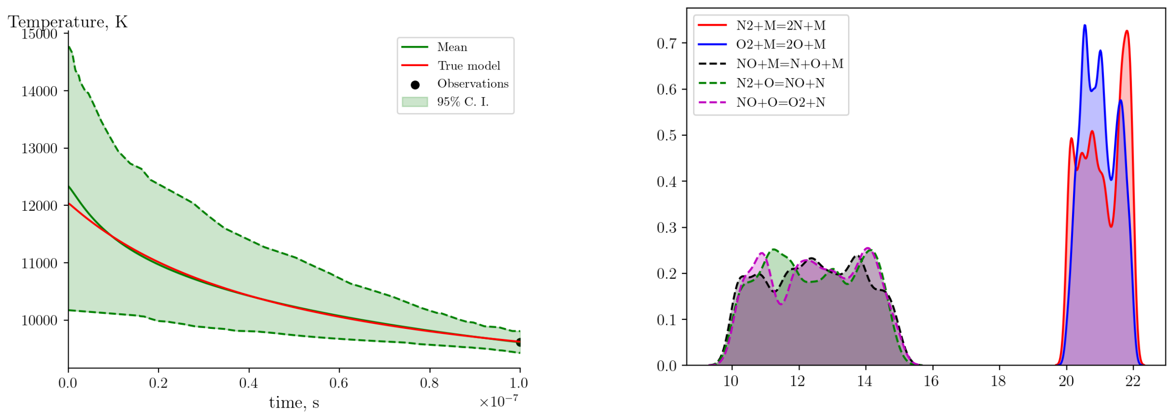


Figure 17: Left: posterior predictive check of the calibrated model with 1 observation at steady-state. Right: resulting marginal posterior distributions of the calibrated pre-exponential factors.

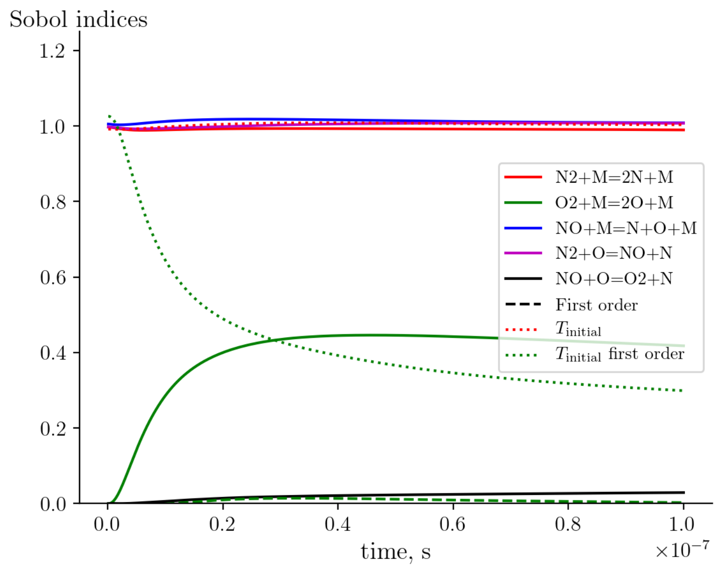


Figure 18: Time evolution of the first order and total order Sobol indices for the variance-based sensitivity of the temperature evolution of an air5 adiabatic reactor at 100 hPa with respect to the pre-exponential factors of the reaction rates of the chemical mechanism and the initial temperature.

Two major "ingredients" are used for the computation of the posterior in `computation_posterior.` the `json` file `models.json`, and the file `data_assembly_cat.py`.

The file `models.json` contains a dictionary where the models can be called together with the priors used for their input variables and the order of the inputs to the model in the form of indexes (`inputs` in the `json` file dictionary).

The file `data_assembly_cat.py` contains the class to assemble the likelihood function needed for the subsequent Bayesian analysis. In it, the log-likelihood function is loaded the hyperparameters for the different priors as well as de-normalization of the inputs are specified.

To run completely new cases, the **suggested workflow** is as follows:

1. Run the optimal algorithm in a grid of points (`catalysis/utilities/sampler_opt_loglik.py`). This first step is suposed to be custom-made as it will depend on the solver and how it inputs and outputs data. For the particular case of the Boundary Layer Code at VKI, a model script I made is included. This script contains the `BL` function which computes the heat flux given a set of parameters `x0` and the catalytic parameters `gamma` which are vectors. The function `functional` is the function to minimize, it is the negative log-likelihood function. The important snippet of this script is the `main loop` which basically goes over the grid of gamma values and finds the optimal values of the log-likelihood function by searching on `x0`. The values of `x0` and `gamma`, together with the negative log-likelihood values are written in the file `output.out`. NOTE: We are looking for the maximizers of the log-likelihood function. As we use an optimization algorithm (Nelder-Mead) which minimizes an objective function, we change the sign of the log-likelihood function to be the negative log-likelihood function, and this is the value that is printed out in `output.out`.

2. Use `catalysis/models/save_GP.py` and `catalysis/models/save_H_GP.py` to save GP models that approximate the optimal log-likelihood function and the optimal enthalpy based on the points previously computed. NOTE: You can do this either with the implemented `SoFIA/utilities/GPR` module or with the *scikit-learn* library.

3. In the file `catalysis/models/models.json`, save the data of your case.

4. Use the model to compute the posterior distribution and obtain the resulting chain from the MCMC algorithm with `catalysis/computation_posterior.py`. The resulting MCMC chain can be used in `catalysis/computation_H_optimal.py` to compute the H^{opt} enthalpy.

5. Check diagnostics.

6. Re-run additional points of the optimal log-likelihood algorithm in the area where MCMC drew points in a Monte Carlo fashion.

7. Start from point 2.

Proposed diagnostics. A tricky issue may arise for non-experts in these analyses. The workflow described cannot be run blindly, without thoroughly assessing the quality of the results. As this requires some in-depth knowledge of the analyses, I propose here a quick way of checking if the GP surrogate + MCMC sampling give reasonable results or if the surrogate estimation would need additional points. There are two sources of problems in this procedure. Once we have a GP surrogate of the optimal log-likelihood to work with, we need to check if the MCMC chain is converged. For this, 3 different diagnostics are presented in `Docs/jup_notebooks/catalysis_demo.py`. We compute the acceptance rate, we plot the trace of the chain and the autocorrelation function. That is enough to have a thorough assessment. Second, the GP surrogate might not be good at all, for this purpose we propose the following: plot the chain samples on the S-curves as done at the end of the script in `Docs/jup_notebooks/catalysis_demo.py`. This gives you a quick view of the samples drawn from the posterior and you can check whether or not they are in an area where they would be expected to be. If you know the chain is converged but the samples just don't really follow the S-curves or seem to be in weird places, this is most likely due to a bad surrogate. In such case, go back to the code and add a few extra points until this last diagnostics is satisfactory.

Nitrogen ablation. This is another application that falls from my PhD work. We calibrate a chemically reacting flow model in chemical non-equilibrium with plasma wind tunnel experimental data. We look for the resulting uncertainty on the nitridation reaction efficiencies for each independent experimental run. We do this with different models that encapsulate different assumptions. For more technical details see footnote[5].

In SoFIA, in the `Applications/nitridation/models` folder, we include the saved Gaussian Process (GP) models for each experimental run of names `G4,5,...`. These GP models are built to approximate the logarithm of the recession rate and CN densities measured in the boundary layer. In turn, such models are used to run an MCMC algorithm and compute the marginal posterior distributions of the quantities of interest (nitridation efficiency parameters). The file `nitridation/models/models.json` contains dictionaries that basically index the location of the models to be loaded for each physical model and experimental case of interest. Further, the file `nitridation/cases.json` contains the experrimental data and Non-Dimensional Parameters (NDPs) for each case.

For an example on how to run nitridation cases, refer to `SoFIA/Docs/jup_notebooks/env/nitrida`

To run completely new cases the **suggested workflow** is as follows:

1. Run the model in a grid of points

[5]A. del Val, Bayesian calibration and assessment of gas-surface interaction models and experiments in atmospheric entry plasmas, IPP/VKI, 2021 - `www.doi.org/10.13140/RG.2.2.27055.82089`