GPdoemd: Gaussian processes for design of experiments for model discrimination

Simon Olofsson
Department of Computing
Imperial College London
SW7 2AZ, United Kingdom
simon.olofsson15@imperial.ac.uk

Updated: 17 September 2018

Abstract

This document describes how to install and use the GPdoemd python package.

1 Introduction

The GPdoemd package contains functionality for performing design of experiments for model discrimination. The idea is to hybridise analytical and data-driven approaches to design of experiments to exploit useful features of both approaches, mainly computational efficiency and the ability to accommodate black-box models [Olofsson et al., 2018]. The aim has been to develop a plug-and-play python package. Our hope is that the package will be easy to extend and build upon. The GPdoemd uses functionality from the GPy [since 2012] package.

GPdoemd contains functionality for testing and comparing analytic, numerical and GP surrogate methods for design of experiments for model discrimination.

GPdoemd contains a set of analytic and non-analytical case studies. Most case studies are taken from literature, and a few have been constructed in the process of developing this package.

1.1 License

2 Background

This section has been copied from Olofsson et al. [2018], with some modifications.

2.1 Model Discrimination

Model discrimination aims to discard inaccurate models, i.e., hypotheses that are not supported by the data. Assume models $\boldsymbol{f}_i : \mathbb{R}^{d \times D_i} \to \mathbb{R}^E$, $i = 1, \ldots, M$, are given. Each model $\boldsymbol{f}_i$ takes as inputs the design variables $\boldsymbol{x} \subset \mathbb{R}^d$ and model parameters $\boldsymbol{\theta}_i \subset \mathbb{R}^{D_i}$. The model is an hypothesis and collection of assumptions about a biological system that can be used for predictions. The design variables $\boldsymbol{x}$, i.e. variables that can be set for a physical experiments, specify a physical experiment that can be run on the system described by the models, with $\mathcal{X}$ defining the operating system boundaries. The classical setting allows tuning of the model parameters $\boldsymbol{\theta}_i$ to fit the model to data. The system has $E \geq 1$ target dimensions. We denote the function for each target dimension e as $f_{i,(e)}$. This notation distinguishes between the e^{th} target dimension $f_{(e)}$ of a model $\boldsymbol{f}$, and model $\boldsymbol{f}_i$.

Table 1: Summary of notation.

Symbol	Description
$\boldsymbol{f}_i$	Model i
$f_{i,(e)}$	Dimension e of model $\boldsymbol{f}_i$; $e = 1, \ldots, E$
$\boldsymbol{x}$	Design variables; $\boldsymbol{x} \subset \mathbb{R}^d$
$\boldsymbol{\theta}_i$	Parameters of model $\boldsymbol{f}_i$; $\boldsymbol{\theta}_i \subset \mathbb{R}^{D_i}$
M	No. of models $\mathcal{M}_i$; $i = 1, \ldots, M$
E	No. of target dimensions; $\boldsymbol{f}_i : \mathbb{R}^{d+D_i} \to \mathbb{R}^E$
$\boldsymbol{\Sigma}_{\text{exp}}$	Measurement noise covariance
$\mathcal{D}$	The set of experimental observations
$\mathcal{D}_{\text{sim},i}$	Training data for surrogate model $\hat{\boldsymbol{f}}_i$

We start with an initial set of N_0 experimental measurements $\boldsymbol{y}_n = \boldsymbol{f}_{\text{true}}(\boldsymbol{x}_n) + \boldsymbol{\epsilon}_n$, $n = 1, \ldots, N_0$. A common assumption in pharmacokinetics is that the measurement noise term $\boldsymbol{\epsilon}_n$ is i.i.d. zero-mean Gaussian distributed with covariance $\boldsymbol{\Sigma}_{\text{exp}}$. Skew in the data distribution can be handled, e.g. through log-transformation. The experimental data set is denoted $\mathcal{D} = \{\boldsymbol{y}_j, \boldsymbol{x}_j\}$. The initial N_0 experimental data points are insufficient to discriminate between the models, i.e. $\forall\, i : p(\boldsymbol{f}_i \,|\, \mathcal{D}) \approx 1/M$. We are agnostic to the available experimental data, and wish to incorporate all of it in the model discrimination.

2.2 Classical Analytical Approach

Methods for tackling the design of experiments for discriminating simple, analytical models have been around for over 50 years. Box and Hill [1967] study the expected change $\mathbb{E}[\Delta S] = \mathbb{E}[S_{N+1}] - S_N$ in Shannon entropy $S_N = \sum_i \pi_{i,N} \log \pi_{i,N}$ from making an additional experimental observation, where the posterior model probability $\pi_{i,N+1} = p(\boldsymbol{y}|\mathcal{M}_i)\pi_{i,N}/p(\boldsymbol{y})$ and $p(\boldsymbol{y}) = \sum_i p(\boldsymbol{y}|\mathcal{M}_i)\pi_{i,N}$. Box and Hill [1967] develop a design criterion D_{BH} by maximising the upper bound on $\mathbb{E}[\Delta S]$. The expression for D_{BH} can be found in the supplementary material. Box and Hill [1967] choose the next experiment by finding $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} D_{\text{BH}}$. Design of experiments continues until there exists a model probability $\pi_{N,i} \approx 1$.

Buzzi-Ferraris et al. [1990] propose a new design criterion D_{BF} (see the supplementary material) from a frequentist point-of-view. They suggest using a χ^2 test with $NE - D_i$ degrees of freedom for the model discrimination. The null hypothesis for each model is that the model has generated the experimental data. Design of experiments continues until only one model passes the χ^2 test. Models are not ranked against each other since Buzzi-Ferraris et al. [1990] argue this simply leads to the least inaccurate model being selected. The χ^2 procedure is more robust against—but not immune to—favouring the least inaccurate model.

Michalik et al. [2010] proceed from the Akaike information criterion (AIC) as the model discrimination criterion to derive a design criterion D_{AW} from the Akaike weights w_i. The expression for D_{AW} can be found in the supplementary material. Design of experiments continues until there exists an Akaike weight $w_i \approx 1$. Box and Hill [1967] and Michalik et al. [2010] implicitly favour the least inaccurate model.

In order to account for the uncertainty in the model parameters $\boldsymbol{\theta}$, the classical analytical approach [Prasad and Someswara Rao, 1977, Buzzi-Ferraris et al., 1984] is to approximate the model parameters as being Gaussian distributed $\mathcal{N}(\hat{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_\theta)$ around the best-fit parameter values $\hat{\boldsymbol{\theta}}$. The covariance $\boldsymbol{\Sigma}_\theta$ is given by a first-order approximation

$$\boldsymbol{\Sigma}_\theta^{-1} = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_i^{n\top} \boldsymbol{\Sigma}_{\mathrm{exp}}^{-1} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_i^{n} \,, \tag{1}$$

where $\boldsymbol{f}_i^n = \boldsymbol{f}_i(\boldsymbol{x}_n, \boldsymbol{\theta})$, $\boldsymbol{x}_n \in \mathcal{D}$, and the gradient $\nabla_{\boldsymbol{\theta}} \boldsymbol{f}_i = \partial \boldsymbol{f}_i(\boldsymbol{x}, \boldsymbol{\theta}_i)/\partial \boldsymbol{\theta}_i|_{\boldsymbol{\theta}_i = \hat{\boldsymbol{\theta}}_i}$. The approximation of the predictive distribution with $\boldsymbol{\theta}$ marginalised out becomes $p(\boldsymbol{f}_i(\boldsymbol{x}) \,|\, \mathcal{D}) = \mathcal{N}(\boldsymbol{f}_i(\boldsymbol{x}, \hat{\boldsymbol{\theta}}), \breve{\Sigma}_i(\boldsymbol{x}))$, where the covariance is $\breve{\Sigma}_i(\boldsymbol{x}) = \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_i \boldsymbol{\Sigma}_\theta \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_i^\top$.

Limitations Most model functions for healthcare applications such as pharmacokinetics or medical devices are not analytical. We can evaluate the model function $\boldsymbol{f}_i$, but the gradients $\nabla_{\boldsymbol{\theta}} \boldsymbol{f}_i$ are not readily available.

2.3 Bayesian Design of Experiments

Methods for accommodating non-analytical model functions have developed in parallel with increasing computer speed. These methods are typically closer to fully Bayesian than the classical analytical methods. Vanlier et al. [2014] approximate the marginal predictive distributions of M models and their Jensen-Shannon divergence using Markov chain Monte Carlo (MCMC) sampling and a k-nearest neighbours density estimate. On the downside, the density estimates become less accurate as the number of experimental observations increases [Vanlier et al., 2014] and the method is computationally intensive [Ryan et al., 2016].

Instead of studying design of experiments for model discrimination, statisticians have focused on design of experiments and model discrimination separately [Chaloner and Verdinelli, 1995]. Design of experiments can also be used to aid model parameter estimation. They solve problems of the type

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} \mathbb{E}_{\boldsymbol{\theta},\, \boldsymbol{y}_{N+1}} \left[U(\boldsymbol{y}_{N+1}, \boldsymbol{x}, \boldsymbol{\theta}) \right] \tag{2}$$

where the utility function $U(\cdot, \cdot, \cdot)$ serves to either discriminate models or estimate parameters. Criteria for model discrimination are handled separately, usually under the name of model selection or hypothesis testing.

Ryan et al. [2015] use a Laplace approximation of the posterior distribution $p(\boldsymbol{\theta} \,|\, \mathcal{D})$ combined with importance sampling. Drovandi et al. [2014] develop a method based on sequential Monte Carlo (SMC) that is faster than using MCMC. Woods et al. [2017] use a Monte Carlo approximation $\Phi(\boldsymbol{x}) = \frac{1}{B} \sum_b^B U(\boldsymbol{y}_b, \boldsymbol{x}, \boldsymbol{\theta}_b)$ with $(\boldsymbol{y}_b, \boldsymbol{\theta}_b) \sim p(\boldsymbol{y}_b, \boldsymbol{\theta}_b | \boldsymbol{x})$ on which they place a Gaussian process prior.

Limitations These methods agnostically accommodate non-analytical models using a Monte Carlo-approach but require exhaustive model sampling in the model parameter space. SMC methods can converge faster than MCMC methods [Woods et al., 2017]. But SMC methods can suffer from sample degeneracy, where only a few particles receive the vast majority of the probability weight [Li et al., 2014]. Also, convergence analysis for MCMC and SMC methods is difficult. Furthermore, once an experiment is executed, the model discrimination issue remains. In this case the marginal predictive distributions $p(\boldsymbol{f}_i(\boldsymbol{x}) \,|\, \mathcal{D})$ would enable calculating the model posteriors.

3 Installation

The GPdoemd package has been tested and validated on OSX, Linux and Windows.

3.1 Prerequisites

Python 3.4+.

Required python packages are:

- numpy >= 1.7
- scipy >= 0.17
- GPy

Optional:

- gp_grief (forked repository: https://github.com/scwolof/gp_grief): GP-GRIEF surrogate models

3.2 Creating a virtual environment

We recommend installing GPdoemd in a virtual environment.

3.2.1 OSX and Linux

To set up a new virtual environment called myenv (example name), run the command

```
python3 -m venv myenv
```

in the folder where you want to store the virtual environment. After the virtual environment has been created, activate it as follows

```
source myenv/bin/activate
```

It is recommended that you upgrade the pip installation before proceeding

```
pip install --upgrade pip
```

3.2.2 Windows

Open the command prompt and navigate to your preferred installation folder. To set up a new virtual environment called myenv (example name), run the command

```
python -m venv myenv
```

After the virtual environment has been created, activate it by running the command

```
myenv\Scripts\activate
```

It is recommended that you upgrade the pip installation before proceeding

```
python -m pip install -upgrade pip
```

3.3 Installing GPdoemd

To install GPdoemd, first install all required packages. They are listed in the file requirements.txt.

```
pip install numpy scipy six paramz matplotlib
pip install GPy
```

Then run the following to install `GPdoemd`

```
pip install git+https://github.com/cog-imperial/GPdoemd
```

It is also possible to download the `GPdoemd` git repository and install it with `python setup.py`.

3.3.1 Problem installing GPy on Windows

If the command `pip install GPy` fails on Windows, try downloading the source code from the `GPy` GitHub repository and installing by running the command

```
python setup.py develop
```

in the `GPy\GPy` folder. If that fails by throwing cython-related errors, edit `setup.py` by replacing the section related to `ext_mods` with `ext_mods = []` and run the previous command again.

3.4 Uninstalling `GPdoemd`

The `GPdoemd` package can be uninstalled by running

```
pip uninstall GPdoemd
```

Alternatively, the folder containing the virtual environment can be deleted.

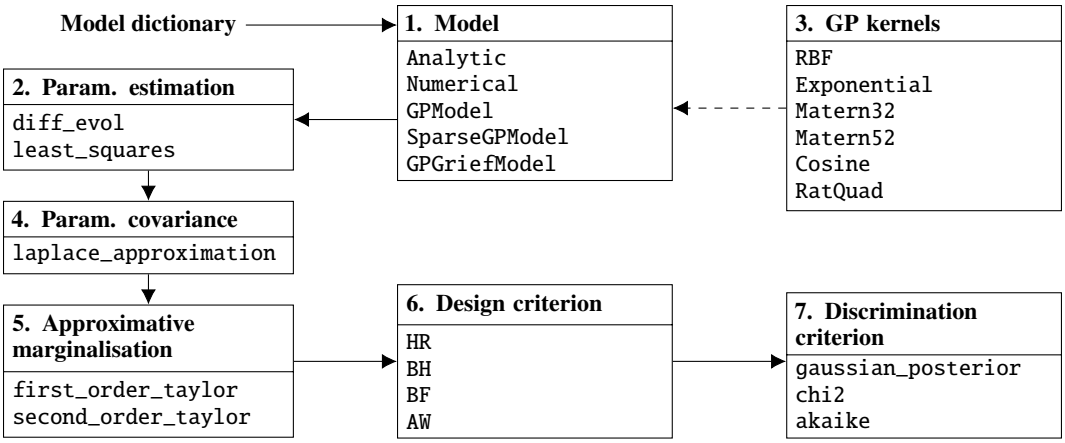

Figure 1: The modular structure of the GPdoemd open-source software package.

4 Using `GPdoemd`

4.1 Initialising a surrogate model

A surrogate model is initialised by calling surrogate model class with an input dictionary `d`. For example, to initialise a regular GP surrogate model, we run the following

```
from GPdoemd.models import GPmodel
M = GPModel(d)
```

The different model classes differ in the way they compute/approximate the gradients of the model function with respect to the model parameters. The available model classes are

Model class	Gradients
`Analytic`	$\partial f/\partial \boldsymbol{\theta}$ provided by user through model function
`Numerical`	Finite difference approximation of $\partial f/\partial \boldsymbol{\theta}$
`GPModel`	$\partial \mu/\partial \boldsymbol{\theta}$, $\partial^2 \mu/\partial \boldsymbol{\theta}^2$, $\partial \Sigma/\partial \boldsymbol{\theta}$, $\partial^2 \Sigma/\partial \boldsymbol{\theta}^2$
`SparseGPModel`	Same as `GPModel`

The model classes `GPModel` and `SparseGPModel` are subclasses of `SurrogateModel`.

The dictionary `d` has the following mandatory and optional fields:

Field	Type	Default	Description
`name`	`str`	'model'	Model name
`call`	callable	-	Function handle $f(\boldsymbol{x}, \boldsymbol{\theta})$
`dim_x`	`int`	-	Number of design variables
`dim_p`	`int`	-	Number of model parameters
`num_outputs`	`int`	1	Number of target dimensions
`p_bounds`	`list`, `numpy.ndarray`	[]	Model parameter bounds (`dim_p` $\times$ 2)
`meas_noise_var`	`float`, `numpy.ndarray`	1	Measurement noise (co)variance
`binary_variables`	`list`	[]	Binary design variable dimensions

See the file `models/model.py` for more information. A field missing a default value is mandatory. It is recommended to set the non-mandatory fields as well, in order to avoid any later problems. In addition, the model classes `GPModel` and `SparseGPModel` have the following optional field:

Field	Type	Default	Description
gp_noise_var	float	10^{-6}	Fixed GP noise variance hyperparameter value

It is recommended to leave this value with its default value, unless you run into numerical problems.

4.2 Performing parameter estimation

Given experimental data Xdata and Ydata, we can tune the model parameters $\boldsymbol{\theta}$ to make the model predictions $f(\boldsymbol{x}, \boldsymbol{\theta})$ fit the data as well as possible. Best-fit model parameter values are required for each model in order to design the next experiment. The best-fit model parameter values, in the form of a 1D numpy array pstar of length dim_p can be given directly to the surrogate model M as

```
M.pmean = pstar
```

Alternatively, one of the two built-in parameter estimation methods (differential evolution diff_evol or least squares least_squares) can be used in one of the following ways

```
from GPdoemd.param_estim import diff_evol, least_squares
M.pmean = diff_evol(M, Xdata, Ydata, M.p_bounds)          # Alternative 1
M.pmean = least_squares(M, Xdata, Ydata, M.p_bounds)      # Alternative 2
M.param_estim(Xdata, Ydata, diff_evol, M.p_bounds)        # Alternative 3
M.param_estim(Xdata, Ydata, least_squares, M.p_bounds)    # Alternative 4
```

The parameter estimation methods are wrappers for the differential evolution and least squares methods found in scipy. We would recommend using least_squares for models with more expensive function evaluations, such as solving systems of ODEs, and diff_evol for models with cheaper function evaluations.

4.3 Setting GP surrogate model training data

We assume a GP surrogate model M has been initialised, and that we are given training data Y and Z

```
Y = np.array([ M.call(x,p) for x,p in zip(X,P) ])
Z = np.c_[ X, P ]
```

for locations X (n $\times$ dim_x numpy array) and model parameter values P (n $\times$ dim_p numpy array). The training data is then fed to the GP surrogate model using **one** of the following commands

```
M.set_training_data(Z, Y)       # Alternative 1
M.set_training_data(X, P, Y)    # Alternative 2
M.Z, M.Y = Z, Y                 # Alternative 3
```

4.4 Setting GP surrogate model kernel functions

The kernel function of the GP prior is assumed to be multiplicative in the form $k(\{\boldsymbol{x}, \boldsymbol{\theta}\}, \{\boldsymbol{x}', \boldsymbol{\theta}'\}) = k_x(\boldsymbol{x}, \boldsymbol{x}')k_\theta(\boldsymbol{\theta}, \boldsymbol{\theta}')$. This assumption enables us to easily calculate $\partial k(\cdot, \cdot)/\partial \boldsymbol{\theta}$. The GPdoemd package has extended the kernel functions of the GPy package by including second derivatives with respect to the input of some standard stationary kernel functions. All kernel functions assume automatic relevance detection (ARD), such that the distance measure $r(\boldsymbol{z}, \boldsymbol{z}') = \sqrt{(\boldsymbol{z} - \boldsymbol{z}')^\top \boldsymbol{\Lambda}^{-1} (\boldsymbol{z} - \boldsymbol{z}')}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix of squared length scales.

To set the kernel functions used in the surrogate model, e.g. the RBF kernel, the following lines are used

```
from GPdoemd.kernels import RBF
M.kern_x = RBF
M.kern_p = RBF
```

Other kernels available are Exponential, Matern32, Matern52, Cosine and RatQuad.

4.5 Constructing GP surrogates

When the training data `Z` and `Y`, and kernels `kern_x` and `kern_p`, have been given, the GP surrogate model must be constructed. A separate GP model is constructed for (i) each target dimension and (ii) binary variable combination. If there are 5 target dimensions, and 3 binary variables, a total of 5×2^3 GP models are constructed.

Constructing the GP models for the surrogate model `M` is done as follows

```
M.gp_surrogate()
```

Alternatively, setting the training data and kernels can be done at the same time as the GP surrogate is constructed

```
M.gp_surrogate(Z=Z, Y=Y, kern_x=RBF, kern_p=RBF)
```

If any of `Z`, `Y`, `kern_x` or `kern_p` have already been set, they do not need to be included in the `gp_surrogate` call.

4.6 Learning GP surrogate model hyperparameters

In order to train the hyperparameters for the GP models of surrogate model `M`, we make one of the following calls

```
M.gp_optimise()    # British spelling
M.gp_optimize()    # American spelling
```

The function `gp_optimize` calls `gp_optimise`. Optional function call inputs are:

- `index`, an `int` or list of `int`s with index/indices for the target dimension(s) for which to (re)train the GP model hyperparameters. Default is to train all GP models.
- `max_lengthscale`, the maximum lengthscale allowed for any design or model parameter dimension, given that all training data have been normalised to the space [0,1]. Default value is 10.

4.7 Approximating the model parameter distribution

The model parameter distribution $p(\boldsymbol{\theta} \mid \boldsymbol{X}_{\text{data}}, \boldsymbol{Y}_{\text{data}}) \approx \mathcal{N}(\boldsymbol{\theta}^*, \boldsymbol{\Sigma}_\theta)$ is approximated with a Gaussian distribution around the best-fit model parameter values $\boldsymbol{\theta}^*$. In order to compute the model parameter covariance $\boldsymbol{\Sigma}_\theta$, we use a Laplacian approximation

```
from GPdoemd.param_covar import laplace_approximation
M.Sigma = laplace_approximation(M, Xdata)
```

4.8 Computing approximate marginal predictive distribution

We can get the approximate marginal predictive distributions at locations `xnew` (`nn` $\times$ `dim_x` numpy array) by calling

```
from GPdoemd.marginal import taylor_first_order
mu, S = taylor_first_order( M, xnew )
```

which produces the mean `mu` (`nn` $\times$ `num_outputs` numpy array) and covariance `S` (`nn` $\times$ `num_outputs` $\times$ `num_outputs` numpy array). An alternative to the first-order Taylor approximation is the second-order Taylor approximation `taylor_second_order`.

4.9 Computing design criterion

In order to find the next optimal experiment, we maximise some design utility function, also known as a design criterion. We begin by computing the means `mu` and covariances `s2` of the marginal predictive distributions for all models `Ms` = $\{$`M`$_i\}$ at all test points `xnew`

```
import numpy as np
n  = len(xnew)
m  = len(Ms)
E  = Ms[0].num_outputs
mu = np.zeros(( n, m, E ))
s2 = np.zeros(( n, m, E, E ))
for i, M in enumerate( Ms ):
    mu[:, i], s2[:, i] = taylor_first_order( M, xnew )
    # Alternatively, for a potentially more accurate mean
    mu[:, i] = np.array([ M.call( x, M.pmean) for x in Xdata ])
```

We can then compute the design criterion at all design test points `xnew`, and find the test point `xnext` with the highest score

```
from GPdoemd.design_criteria import JR
measvar = Ms[0].meas_noise_var          # Measurement noise covariance
pps     = [ 1. / m for _ in range(m) ]  # Prior probabilities
dc      = JR( mu, s2, measvar, pps )    # Compute design criterion
xnext   = xnew[ np.argmax( dc ) ]       # Find next design
```

There are multiple design criteria implemented in the `GPdoemd` package:

Design criterion	Reference	Requires	
		`measvar`	`pps`
`HR`	Hunter and Reiner [1965]		
`BH`	Box and Hill [1967]	✓	✓
`BF`	Buzzi-Ferraris et al. [1990]	✓	
`AW`	Michalik et al. [2010]	✓	✓
`JR`	-	✓	✓

If `pps` is not set, it will be assumed that the prior probability is equal for all models.

4.10 Discriminating between models

When a new experiment experiment has been performed, and the result incorporated into the data set `Xdata`, `Ydata`, we can attempt model discrimination. First, the model parameters should be re-tuned.

```
for M in Ms:
    M.param_estim( Xdata, Ydata, diff_evol, M.p_bounds )
```

We may want to generate new training data `Z`, `Y` for the GP surrogates models, and retrain them.

```
for M in Ms:
    M.gp_surrogate( Z=Z, Y=Y )
    M.gp_optimise()
    M.Sigma = laplace_approximation( M, Xdata )
```

Then we want to compute the marginal predictive distributions for the design in our experimental data set.

```
mu = np.zeros(( n, m, E ))
s2 = np.zeros(( n, m, E, E ))
for i, M in enumerate( Ms ):
    mu[:, i], s2[:, i] = taylor_first_order( M, Xdata )
    # Alternatively, for a potentially more accurate mean
    mu[:, i] = np.array([ M.call( x, M.pmean) for x in Xdata ])
```

The last thing we need is an array `D = np.array([ M.dim_p for M in Ms ])` with the number of model parameters for each model. Now we are ready to determine which models adequately describe the experimental data, or which model fits the data best.

We will use the following mathematical notation when defining the discrimination criteria

Symbol	Python	Notes
$\boldsymbol{X}$	`Xdata`	$\boldsymbol{X}^\top = [\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_N^\top]$
$\boldsymbol{Y}$	`Ydata`	$\boldsymbol{Y}^\top = [\boldsymbol{y}_1^\top, \ldots, \boldsymbol{y}_N^\top]$
$\breve{\mu}_i(\boldsymbol{x}_j)$	`mu[j-1,i-1]`	Marginal predictive mean for model i and design $\boldsymbol{x}_j$
$\breve{\Sigma}_i(\boldsymbol{x}_j)$	`s2[j-1,i-1]`	Marginal predictive covariance for model i and design $\boldsymbol{x}_j$
$\boldsymbol{\Sigma}$	`measvar`	Measurement noise covariance
E	`E`, `M.num_outputs`	Number of target dimensions
D_i	`Ms[i-1].dim_p`	Number of model parameters for model i

4.10.1 Gaussian posterior

Box and Hill [1967] suggests using the (normalised) Gaussian posteriors π_i for model selection

$$\pi_i = \frac{p(\boldsymbol{Y} \mid \boldsymbol{X}, \breve{\mu}_i, \breve{\Sigma}_i)}{\sum_{j=1}^m p(\boldsymbol{Y} \mid \boldsymbol{X}, \breve{\mu}_j, \breve{\Sigma}_j)}, \quad \text{with } p(\boldsymbol{Y} \mid \boldsymbol{X}, \breve{\mu}_i, \breve{\Sigma}_i) = \prod_{n=1}^N \mathcal{N}(\boldsymbol{y}_n \mid \breve{\mu}_i(\boldsymbol{x}_n), \breve{\Sigma}_i(\boldsymbol{x}_n) + \boldsymbol{\Sigma}).$$

Clearly, $\sum_i \pi_i = 1$. In order to compute the the Gaussian posteriors `pps`, we call

```
from GPdoemd.discrimination_criteria import gaussian_posterior
s2 += measvar
pps = gaussian_posterior(Ydata, mu, s2)
```

We can set a threshold ξ (e.g. $\xi = 10^{-8}$), where a model i is discarded if $\pi_i < \xi$.

If we have computed the Gaussian posteriors $\pi_i = \pi_{i,N}$ for the first N data points, Box and Hill [1967] suggests updating the Gaussian posterior after each additional data point in the following way

$$\pi_{i,N+1} = \frac{\pi_{i,N}\mathcal{N}(\boldsymbol{y}_{N+1} \mid \breve{\mu}_i(\boldsymbol{x}_{N+1}), \breve{\Sigma}_i(\boldsymbol{x}_{N+1}) + \boldsymbol{\Sigma})}{\sum_{j=1}^m \pi_{j,N}\mathcal{N}(\boldsymbol{y}_{N+1} \mid \breve{\mu}_j(\boldsymbol{x}_{N+1}), \breve{\Sigma}_j(\boldsymbol{x}_{N+1}) + \boldsymbol{\Sigma})}.$$

This is done by calling

```
from GPdoemd.discrimination_criteria import gaussian_posterior_update
s2_new += measvar
pps_new = gaussian_posterior_update(y_new, mu_new, s2_new, pps)
```

where `mu_new` and `s2_new` are the mean and covariances for the models at the latest design point. Note that the value of the updated Gaussian posterior $\pi_{i,\tilde{N}}$ will depend on the order in which the $\tilde{N}$ data points are observed.

4.10.2 χ^2 test

Buzzi-Ferraris et al. [1990] advocates using the χ^2 test for model discrimination. This does not rank the models against each other, but models are discarded when they are no longer deemed adequate to describe the observed data. The χ^2 test metric q_i for model i is defined as

$$q_i = 1 - \int_0^\epsilon F_{k_i}^2(\boldsymbol{z})\mathrm{d}z$$

where F_k^2 is the cumulative χ^2 distribution with $k = N \times E - D_i$ degrees of freedom, and ϵ is the sum of weighted errors given by

$$\epsilon = \sum_{n=1}^N (\breve{\mu}_i(\boldsymbol{x}_n) - \boldsymbol{y}_n)^\top \breve{\Sigma}_i^{-1}(\boldsymbol{x}_n)(\breve{\mu}_i(\boldsymbol{x}_n) - \boldsymbol{y}_n).$$

We set a threshold ξ (e.g. $\xi = 0.01$). If $q_i < \xi$, then model i is said to have failed the χ^2 test. However, [Buzzi-Ferraris et al., 1990] argues against discarding the model completely, but simply ignore it when designing the next experiment.

In `GPdoemd` we can compute the models χ^2 test scores q_i by calling

```
from GPdoemd.discrimination_criteria import chi2
s2 += measvar
qs  = chi2( Ydata, mu, s2, D )
```

4.10.3 Akaike information criterion

Michalik et al. [2010] argues in favour of using the Akaike information criterion for model selection. This is similar to the Gaussian posteriors, but adds a penalty term for the number of tunable model parameters, such that

$$\tilde{p}(\boldsymbol{Y} \mid \boldsymbol{X}, \breve{\mu}_i, \breve{\Sigma}_i) = \exp\left(2 \log p(\boldsymbol{Y} \mid \boldsymbol{X}, \breve{\mu}_i, \breve{\Sigma}_i) - 2D_i\right) .$$

The probability terms $\tilde{\pi}_i$ are then computed in the same way as π_i, but using $\tilde{p}(\cdot)$.

```
from GPdoemd.discrimination_criteria import akaike
s2 += measvar
pps = akaike( Ydata, mu, s2, D )
```

5 Case Studies

GPdoemd case studies, with the number of design variables $|\boldsymbol{x}|$ (continuous or discrete), model parameters $|\boldsymbol{\theta}_i|$, target dimensions $|\boldsymbol{y}|$ and rival models M.

Name	Reference	$\lvert\boldsymbol{x}\rvert$, $(\in \mathbb{Z})$	$\lvert\boldsymbol{\theta}_i\rvert$	$\lvert\boldsymbol{y}\rvert$	M	Type
`bff1983`	Buzzi-Ferraris and Forzatti [1983]	3, (0)	5	1	5	Analytic
`bffeh1984`	Buzzi-Ferraris et al. [1984]	2, (0)	4	2	4	Analytic
`bffc1990a`	Buzzi-Ferraris et al. [1990]	3, (0)	2–6	1	4	Analytic
`mixing`	-	3, (1)	1	1	5	Analytic
`msm2010`	Michalik et al. [2010]	3, (0)	1	1	10	Analytic
`vthr2014linear`	Vanlier et al. [2014]	1, (0)	2–4	1	4	Analytic
`vthr2014ode`	Vanlier et al. [2014]	3, (2)	14	1	4	Black-box
`tandogan2017`	Tandogan et al. [2017]	4, (0)	8–14	2	3	Black-box

References

G. E. P. Box and W. J. Hill. Discrimination among mechanistic models. *Technometrics*, 9(1):57–71, 1967.

G. Buzzi-Ferraris and P. Forzatti. A new sequential experimental design procedure for discriminating among rival models. *Chem Eng Sci*, 38(2):225–232, 1983.

G. Buzzi-Ferraris, P. Forzatti, G. Emig, and H. Hofmann. Sequential experimental design for model discrimination in the case of multiple responses. *Chem Eng Sci*, 39(1):81–85, 1984.

G. Buzzi-Ferraris, P. Forzatti, and P. Canu. An improved version of a sequential design criterion for discriminating among rival multiresponse models. *Chem Eng Sci*, 45(2):477–481, 1990.

K. Chaloner and I. Verdinelli. Bayesian experimental design: A review. *Statist Sci*, 10(3):273–304, 1995.

C. C. Drovandi, J. M. McGree, and A. N. Pettitt. A sequential Monte Carlo algorithm to incorporate model uncertainty in Bayesian sequential design. *J Comput Graph Stat*, 23(1):3–24, 2014.

GPy. GPy: A Gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

W. G. Hunter and A. M. Reiner. Designs for discriminating between two rival models. *Technometrics*, 7(3):307–323, 1965.

T. Li, S. Sun, T. P. Sattar, and J. M. Corchado. Fight sample degeneracy and impoverishment in particle filters: A review of intelligent approaches. *Expert Syst Appl*, 41(8):3944–3954, 2014.

C. Michalik, M. Stuckert, and W. Marquardt. Optimal experimental design for discriminating numerous model candidates: The AWDC criterion. *Ind Eng Chem Res*, 49:913–919, 2010.

S. Olofsson, M. P. Deisenroth, and R. Misener. Design of experiments for model discrimination hybridising analytical and data-driven approaches. In *ICML '18: Proceedings of the International Conference on Machine Learning*, Stockholm, Sweden, PMLR 80, 2018.

K. B. S. Prasad and M. Someswara Rao. Use of expected likelihood in sequential model discrimination in multiresponse systems. *Chem Eng Sci*, 32:1411–1418, 1977.

E. G. Ryan, C. C. Drovandi, and A. N. Pettitt. Fully Bayesian experimental design for pharmacokinetic studies. *Entropy*, 17:1063–1089, 2015.

E. G. Ryan, C. C. Drovandi, J. M. McGree, and A. N. Pettitt. A review of modern computational algorithms for Bayesian optimal design. *Int Stat Rev*, 84:128–154, 2016.

N. Tandogan, S. Garcia-Muñoz, M. Sen, T. M. Wilson, J. Y. Buser, S. P. Kolis, I. V. Borkar, and C. A. Alt. Use of model discrimination method in drug substance process developments. In *Proceedings of the AIChE Annual Meeting*, pages X–Y, 2017.

J. Vanlier, C. A. Tiemann, P. A. J. Hilbers, and N. A. W. van Riel. Optimal experiment design for model selection in biochemical networks. *BMC Syst Biol*, 8(20), 2014.

D. C. Woods, A. M. Overstall, M. Adamou, and T. W. Waite. Bayesian design of experiments for generalized linear models and dimensional analysis with industrial and scientific application. *Qual Eng*, 29(1):91–103, 2017.