SMT Documentation

Release 1.3.0

John Hwang, Mohamed Amine Bouhlel, Remi Lafage

Aug 22, 2022

CONTENTS

The surrogate modeling toolbox (SMT) is an open-source Python package consisting of libraries of surrogate modeling methods (e.g., radial basis functions, kriging), sampling methods, and benchmarking problems. SMT is designed to make it easy for developers to implement new surrogate models in a well-tested and well-document platform, and for users to have a library of surrogate modeling methods with which to use and compare methods.

The code is available open-source on GitHub.

CHAPTER

ONE

CITE US

To cite SMT: M. A. Bouhlel and J. T. Hwang and N. Bartoli and R. Lafage and J. Morlier and J. R. R. A. Martins.

A Python surrogate modeling framework with derivatives. Advances in Engineering Software, 2019.

```
@article{SMT2019,
        Author = {Mohamed Amine Bouhlel and John T. Hwang and Nathalie Bartoli and Rémi␣
↪Lafage and Joseph Morlier and Joaquim R. R. A. Martins},
        Journal = {Advances in Engineering Software},
        Title = {A Python surrogate modeling framework with derivatives},
        pages = {102662},
        year = {2019},
        issn = {0965-9978},
        doi = {https://doi.org/10.1016/j.advengsoft.2019.03.005},
        Year = {2019}}
```

CHAPTER

TWO

FOCUS ON DERIVATIVES

SMT is meant to be a general library for surrogate modeling (also known as metamodeling, interpolation, and regression), but its distinguishing characteristic is its focus on derivatives, e.g., to be used for gradient-based optimization. A surrogate model can be represented mathematically as

$$y = f(\mathbf{x}, \mathbf{xt}, \mathbf{yt}),$$

where $\mathbf{xt} \in \mathbb{R}^{nt \times nx}$ contains the training inputs, $\mathbf{yt} \in \mathbb{R}^{nt}$ contains the training outputs, $\mathbf{x} \in \mathbb{R}^{nx}$ contains the prediction inputs, and $y \in \mathbb{R}$ contains the prediction outputs. There are three types of derivatives of interest in SMT:

1. Derivatives (dy/dx): derivatives of predicted outputs with respect to the inputs at which the model is evaluated.
2. Training derivatives (dyt/dxt): derivatives of training outputs, given as part of the training data set, e.g., for gradient-enhanced kriging.
3. Output derivatives (dy/dyt): derivatives of predicted outputs with respect to training outputs, representing how the prediction changes if the training outputs change and the surrogate model is re-trained.

Not all surrogate modeling methods support or are required to support all three types of derivatives; all are optional.

CHAPTER

THREE

DOCUMENTATION CONTENTS

3.1 Getting started

3.1.1 Prerequisites

SMT setup requires `numpy`, ensure you have it installed.

As some surrogates are written in C++, SMT setup uses `Cython` to compile them. If `Cython` is not found, SMT setup tries to install it, but as this step is not robust for all environments it is better to ensure you have it properly installed before hand. If compilation with `Cython` fails then RBF, IDW, RMTB and RMTC surrogates will not be available.

Note: if you use Anaconda Python distribution, it is best to install dependencies: `numpy`, `scipy`, `sklearn`, `Cython` with `conda` command then use `pip` command as follows to terminate the SMT installation.

3.1.2 Installing

To install the latest released version of SMT:

```
pip install smt
```

Or else to install the current version from the GitHub repository master:

```
pip install git+https://github.com/SMTOrg/smt.git@master
```

If you want to contribute to SMT, see *Contributing to SMT* section.

3.1.3 Notebooks

Several notebooks are available to get up to speed with SMT:

- General
- Handling of Noise
- Handling of Mixed Integer variables
- Efficient Global Optimization application

3.1.4 Uninstalling

If you want to uninstall SMT:

```
pip uninstall smt
```

3.2 Surrogate modeling methods

SMT contains the surrogate modeling methods listed below.

3.2.1 Radial basis functions

The radial basis function (RBF) surrogate model represents the interpolating function as a linear combination of basis functions, one for each training point. RBFs are named as such because the basis functions depend only on the distance from the prediction point to the training point for the basis function. The coefficients of the basis functions are computed during the training stage. RBFs are frequently augmented to global polynomials to capture the general trends.

The prediction equation for RBFs is

$$y = \mathbf{p}(\mathbf{x})\mathbf{w_p} + \sum_{i}^{nt} \phi(\mathbf{x}, \mathbf{xt}_i)\mathbf{w_r},$$

where $\mathbf{x} \in \mathbb{R}^{nx}$ is the prediction input vector, $y \in \mathbb{R}$ is the prediction output, $\mathbf{xt}_i \in \mathbb{R}^{nx}$ is the input vector for the i th training point, $\mathbf{p}(\mathbf{x}) \in \mathbb{R}^{np}$ is the vector mapping the polynomial coefficients to the prediction output, $\phi(\mathbf{x}, \mathbf{xt}_i) \in \mathbb{R}^{nt}$ is the vector mapping the radial basis function coefficients to the prediction output, $\mathbf{w_p} \in \mathbb{R}^{np}$ is the vector of polynomial coefficients, and $\mathbf{w_r} \in \mathbb{R}^{nt}$ is the vector of radial basis function coefficients.

The coefficients, $\mathbf{w_p}$ and $\mathbf{w_r}$, are computed by solving the follow linear system:

$$\begin{bmatrix} \phi(\mathbf{xt}_1, \mathbf{xt}_1) & \dots & \phi(\mathbf{xt}_1, \mathbf{xt}_{nt}) & \mathbf{p}(\mathbf{xt}_1)^T \\ \vdots & \ddots & \vdots & \vdots \\ \phi(\mathbf{xt}_{nt}, \mathbf{xt}_1) & \dots & \phi(\mathbf{xt}_{nt}, \mathbf{xt}_{nt}) & \mathbf{p}(\mathbf{xt}_{nt})^T \\ \mathbf{p}(\mathbf{xt}_1) & \dots & \mathbf{p}(\mathbf{xt}_{nt}) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w_r}_1 \\ \vdots \\ \mathbf{w_r}_{nt} \\ \mathbf{w_p} \end{bmatrix} = \begin{bmatrix} yt_1 \\ \vdots \\ yt_{nt} \\ 0 \end{bmatrix}$$

Only Gaussian basis functions are currently implemented. These are given by:

$$\phi(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{d0^2}\right)$$

Usage

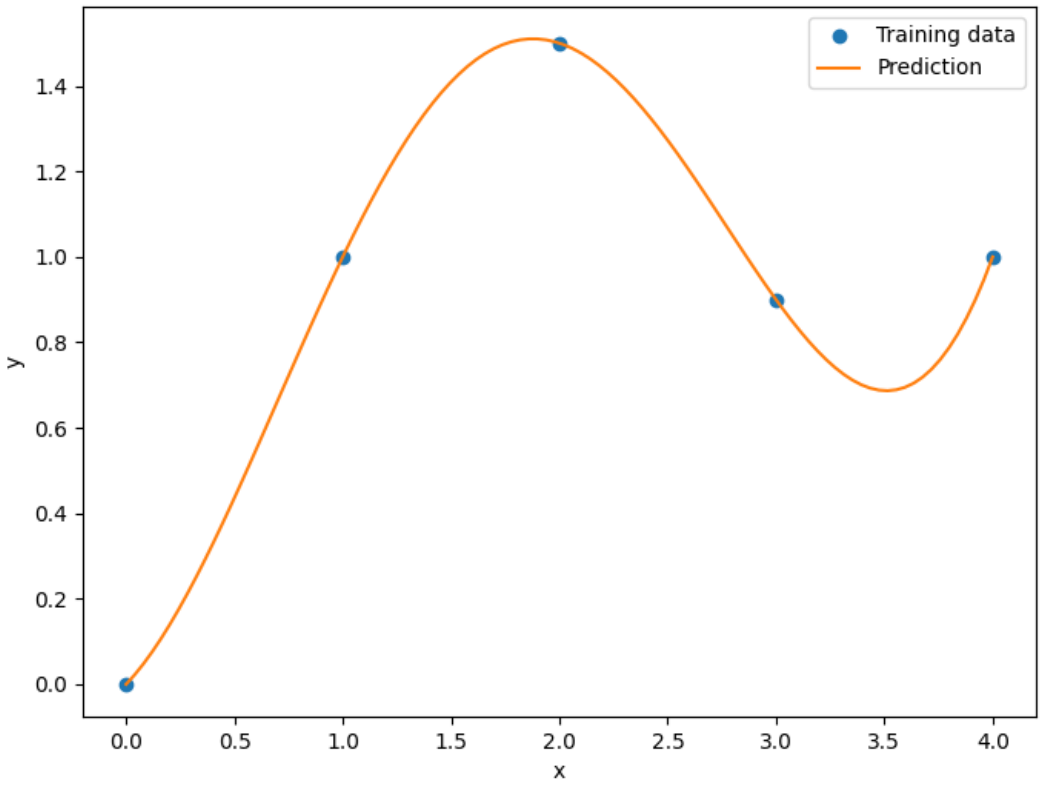
```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import RBF

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.9, 1.0])

sm = RBF(d0=5)
sm.set_training_values(xt, yt)
```

(continues on next page)

(continued from previous page)

```
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)

plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction"])
plt.show()
```

```
___________________________________________________________________________

                                   RBF
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training

   Training ...
      Initializing linear solver ...
         Performing LU fact. (5 x 5 mtx) ...
         Performing LU fact. (5 x 5 mtx) - done. Time (sec):  0.0000000
      Initializing linear solver - done. Time (sec):  0.0000000
      Solving linear system (col. 0) ...
         Back solving (5 x 5 mtx) ...
         Back solving (5 x 5 mtx) - done. Time (sec):  0.0000000
      Solving linear system (col. 0) - done. Time (sec):  0.0000000
   Training - done. Time (sec):  0.0000000
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```


Training data
Prediction
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

Options

Table 1: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
d0	1.0	None	['int', 'float', 'list', 'ndarray']	basis function scaling parameter in exp(-d^2 / d0^2)
poly_degree	-1	[-1, 0, 1]	['int']	-1 means no global polynomial, 0 means constant, 1 means linear trend
data_dir	None	None	['str']	Directory for loading / saving cached data; None means do not save or load
reg	1e-10	None	['int', 'float']	Regularization coeff.
max_print_depth	5	None	['int']	Maximum depth (level of nesting) to print operation descriptions and times

3.2.2 Inverse-distance weighting

The inverse distance weighting[1] (IDW) model is an interpolating method and the unknown points are calculated with a weighted average of the sampling points.

The prediction equation for IDW is

$$y = \begin{cases} \frac{\sum_i^{nt} \beta(\mathbf{x},\mathbf{xt}_i) yt_i}{\sum_i^{nt} \beta(\mathbf{x},\mathbf{xt}_i)}, & \text{if} \quad \mathbf{x} \neq \mathbf{xt}_i \quad \forall i \\ yt_i & \text{if} \quad \mathbf{x} = \mathbf{xt}_i \quad \text{for some } i \end{cases},$$

where $\mathbf{x} \in \mathbb{R}^{nx}$ is the prediction input vector, $y \in \mathbb{R}$ is the prediction output, $\mathbf{xt}_i \in \mathbb{R}^{nx}$ is the input vector for the i th training point, and $yt_i \in \mathbb{R}$ is the output value for the i th training point. The weighting function β is defined by

$$\beta(\mathbf{x}_i, \mathbf{x}_j) = ||\mathbf{x}_i - \mathbf{x}_j||_2^{-p},$$

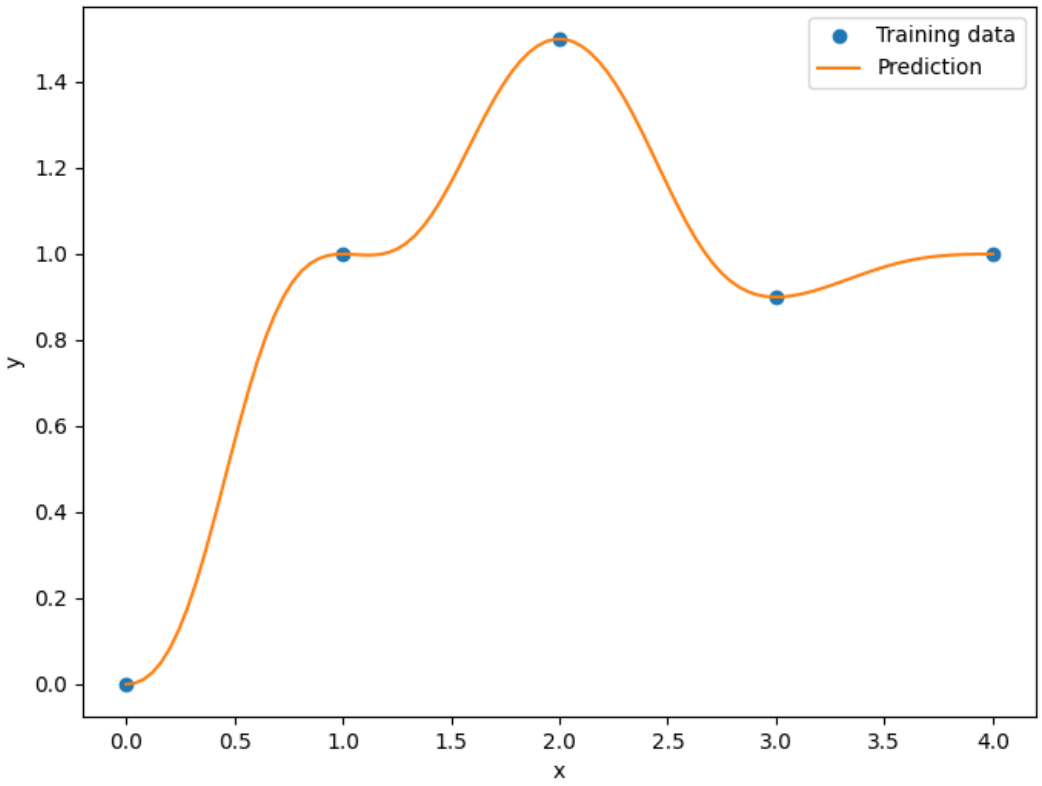
where p a positive real number, called the power parameter. This parameter must be strictly greater than 1 for the derivatives to be continuous.

[1] Shepard, D., A Two-dimensional Interpolation Function for Irregularly-spaced Data, Proceedings of the 1968 23rd ACM National Conference, 1968, pp. 517–524.

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import IDW

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.9, 1.0])

sm = IDW(p=2)
sm.set_training_values(xt, yt)
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)

plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction"])
plt.show()
```

```
___________________________________________________________________________

                                   IDW
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.0000000
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```


Training data
Prediction
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

Options

Table 2: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
p	2.5	None	['int', 'float']	order of distance norm
data_dir	None	None	['str']	Directory for loading / saving cached data; None means do not save or load

3.2.3 Regularized minimal-energy tensor-product splines

Regularized minimal-energy tensor-product splines (RMTS) is a type of surrogate model for low-dimensional problems with large datasets and where fast prediction is desired. The underlying mathematical functions are tensor-product splines, which limits RMTS to up to 4-D problems, or 5-D problems in certain cases. On the other hand, tensor-product splines enable a very fast prediction time that does not increase with the number of training points. Unlike other methods like Kriging and radial basis functions, RMTS is not susceptible to numerical issues when there is a large number of training points or when there are points that are too close together.

The prediction equation for RMTS is

$$y = \mathbf{F}(\mathbf{x})\mathbf{w},$$

where $\mathbf{x} \in \mathbb{R}^{nx}$ is the prediction input vector, $y \in \mathbb{R}$ is the prediction output, $\mathbf{w} \in \mathbb{R}^{nw}$ is the vector of spline coefficients, and $\mathbf{F}(\mathbf{x}) \in \mathbb{R}^{nw}$ is the vector mapping the spline coefficients to the prediction output.

RMTS computes the coefficients of the splines, $\mathbf{w}$, by solving an energy minimization problem subject to the conditions that the splines pass through the training points. This is formulated as an unconstrained optimization problem where the objective function consists of a term containing the second derivatives of the splines, another term representing the approximation error for the training points, and another term for regularization:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} + \tfrac{1}{2}\beta\mathbf{w}^T\mathbf{w} \\ & + \tfrac{1}{2}\tfrac{1}{\alpha}\textstyle\sum_i^{nt}\left[\mathbf{F}(\mathbf{xt}_i)\mathbf{w} - yt_i\right]^2 \end{aligned},$$

where $\mathbf{xt}_i \in \mathbb{R}^{nx}$ is the input vector for the i th training point, $yt_i \in \mathbb{R}$ is the output value for the i th training point, $\mathbf{H} \in \mathbb{R}^{nw \times nw}$ is the matrix containing the second derivatives, $\mathbf{F}(\mathbf{xt}_i) \in \mathbb{R}^{nw}$ is the vector mapping the spline coefficients to the i th training output, and α and β are regularization coefficients.

In problems with a large number of training points relative to the number of spline coefficients, the energy minimization term is not necessary; this term can be zero-ed by setting the reg_cons option to zero. In problems with a small dataset, the energy minimization is necessary. When the true function has high curvature, the energy minimization can be counterproductive in the regions of high curvature. This can be addressed by increasing the quadratic approximation term to one of higher order, and using Newton's method to solve the nonlinear system that results. The nonlinear

formulation is given by

$$\begin{aligned}\min_{\mathbf{w}} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} + \tfrac{1}{2}\beta\mathbf{w}^T\mathbf{w} \\ & + \tfrac{1}{2}\tfrac{1}{\alpha}\sum_i^{nt}[\mathbf{F}(\mathbf{xt}_i)\mathbf{w} - yt_i]^p \end{aligned},$$

where p is the order given by the approx_order option. The number of Newton iterations can be specified via the `nonlinear_maxiter` option.

RMTS is implemented in SMT with two choices of splines:

1. B-splines (RMTB): RMTB uses B-splines with a uniform knot vector in each dimension. The number of B-spline control points and the B-spline order in each dimension are options that trade off efficiency and precision of the interpolant.

2. Cubic Hermite splines (RMTC): RMTC divides the domain into tensor-product cubic elements. For adjacent elements, the values and derivatives are continuous. The number of elements in each dimension is an option that trades off efficiency and precision.

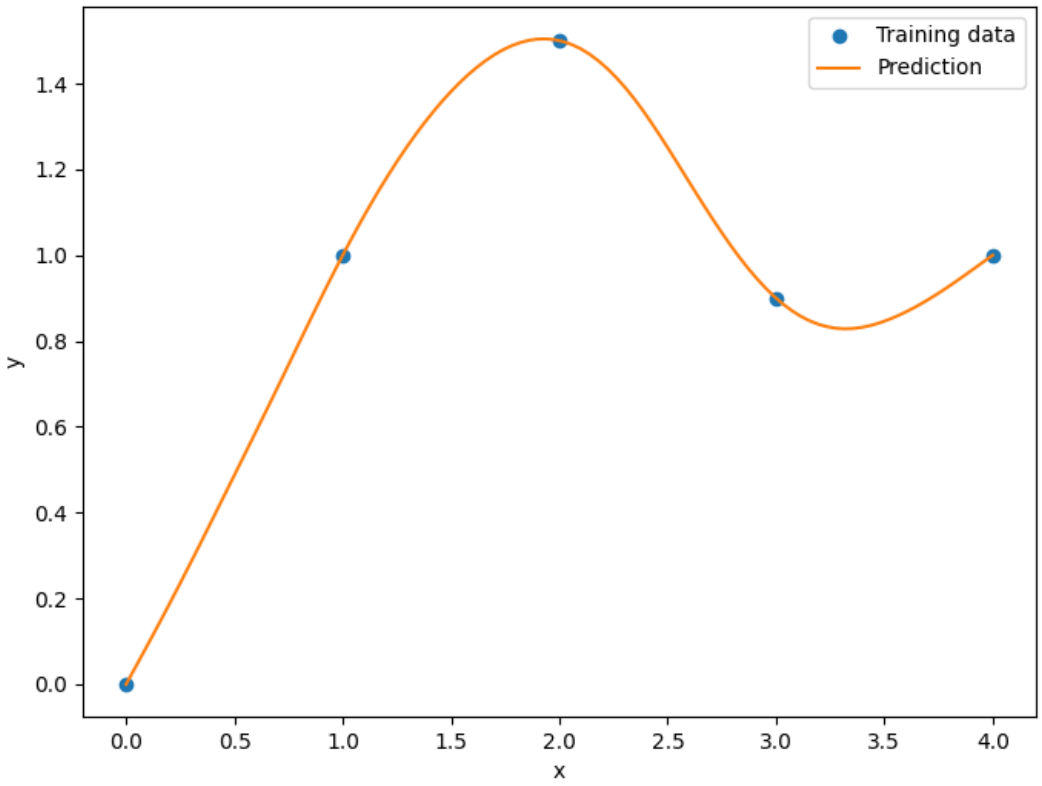
In general, RMTB is the better choice when training time is the most important, while RMTC is the better choice when accuracy of the interpolant is the most important. More details of these methods are given in[1].

Usage (RMTB)

```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import RMTB

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.9, 1.0])

xlimits = np.array([[0.0, 4.0]])

sm = RMTB(
    xlimits=xlimits,
    order=4,
    num_ctrl_pts=20,
    energy_weight=1e-15,
    regularization_weight=0.0,
)
sm.set_training_values(xt, yt)
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)

plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction"])
plt.show()
```

[1] Hwang, J. T., & Martins, J. R. (2018). A fast-prediction surrogate model for large datasets. Aerospace Science and Technology, 75, 74-87.

```
___________________________________________________________________________

                                   RMTB
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training

   Training ...
      Pre-computing matrices ...
         Computing dof2coeff ...
         Computing dof2coeff - done. Time (sec):  0.0000000
         Initializing Hessian ...
         Initializing Hessian - done. Time (sec):  0.0000000
         Computing energy terms ...
         Computing energy terms - done. Time (sec):  0.0000000
         Computing approximation terms ...
         Computing approximation terms - done. Time (sec):  0.0000000
      Pre-computing matrices - done. Time (sec):  0.0000000
      Solving for degrees of freedom ...
         Solving initial startup problem (n=20) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 1.549745600e+00 2.
↪530000000e+00
               Iteration (num., iy, grad. norm, func.) :   0   0 1.395101781e-15 4.
↪464186103e-16
            Solving for output 0 - done. Time (sec):  0.0000000
         Solving initial startup problem (n=20) - done. Time (sec):  0.0000000
         Solving nonlinear problem (n=20) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 1.531354982e-15 4.
↪464186103e-16
            Solving for output 0 - done. Time (sec):  0.0000000
         Solving nonlinear problem (n=20) - done. Time (sec):  0.0000000
      Solving for degrees of freedom - done. Time (sec):  0.0000000
   Training - done. Time (sec):  0.0000000
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0100148

   Prediction time/pt. (sec) :  0.0001001
```


Training data
Prediction
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

Usage (RMTC)

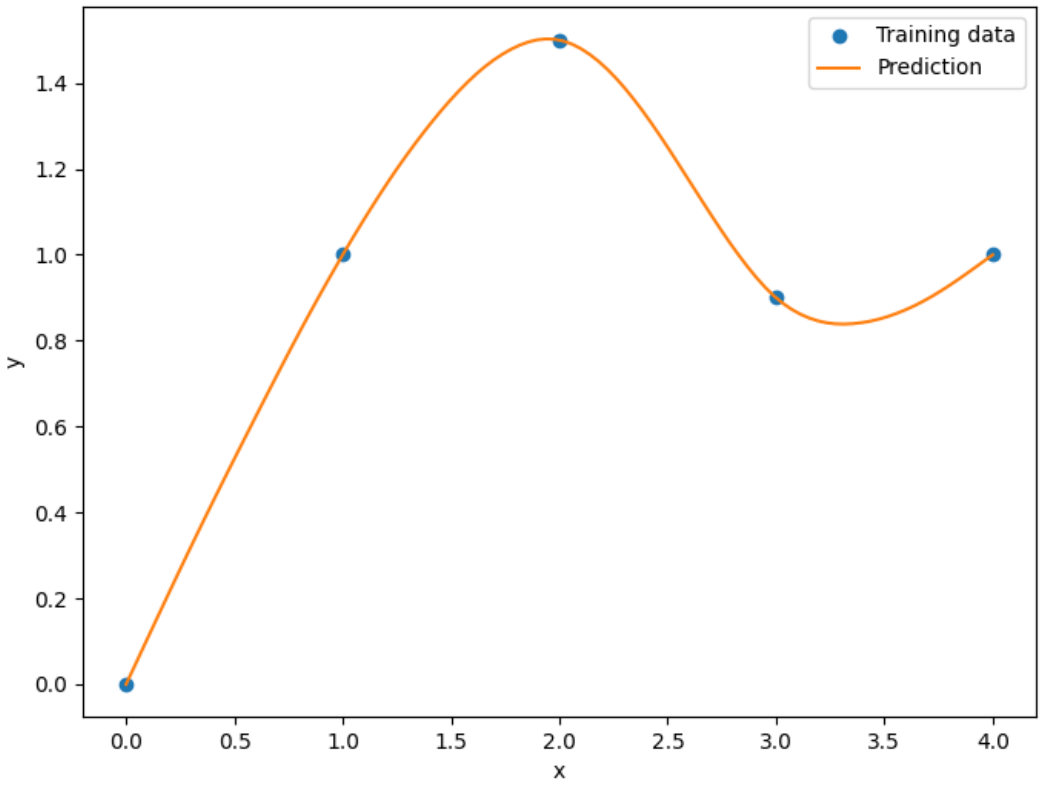
```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import RMTC

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.9, 1.0])

xlimits = np.array([[0.0, 4.0]])

sm = RMTC(
    xlimits=xlimits,
    num_elements=20,
    energy_weight=1e-15,
    regularization_weight=0.0,
)
sm.set_training_values(xt, yt)
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)

plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction"])
plt.show()
```

```
___________________________________________________________________________

                                  RMTC
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training

   Training ...
      Pre-computing matrices ...
         Computing dof2coeff ...
         Computing dof2coeff - done. Time (sec):  0.0000000
         Initializing Hessian ...
         Initializing Hessian - done. Time (sec):  0.0000000
         Computing energy terms ...
         Computing energy terms - done. Time (sec):  0.0020008
```

(continues on next page)

(continued from previous page)

```
        Computing approximation terms ...
        Computing approximation terms - done. Time (sec):  0.0000000
     Pre-computing matrices - done. Time (sec):  0.0020008
     Solving for degrees of freedom ...
        Solving initial startup problem (n=42) ...
           Solving for output 0 ...
              Iteration (num., iy, grad. norm, func.) :   0   0 2.249444376e+00 2.
↪530000000e+00
              Iteration (num., iy, grad. norm, func.) :   0   0 2.004900347e-15 4.
↪346868680e-16
           Solving for output 0 - done. Time (sec):  0.0030320
        Solving initial startup problem (n=42) - done. Time (sec):  0.0030320
        Solving nonlinear problem (n=42) ...
           Solving for output 0 ...
              Iteration (num., iy, grad. norm, func.) :   0   0 2.956393318e-15 4.
↪346868680e-16
           Solving for output 0 - done. Time (sec):  0.0000000
        Solving nonlinear problem (n=42) - done. Time (sec):  0.0000000
     Solving for degrees of freedom - done. Time (sec):  0.0030320
  Training - done. Time (sec):  0.0053463
------------------------------------------------------------------------------

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```


Training data
Prediction
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

Options (RMTB)

Table 3: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
xlimits	None	None	['ndarray']	Lower/upper bounds in each dimension - ndarray [nx, 2]
smoothness	1.0	None	['Integral', 'float', 'tuple', 'list', 'ndarray']	Smoothness parameter in each dimension - length nx. None implies uniform
regularization_weight	1e-14	None	['Integral', 'float']	Weight of the term penalizing the norm of the spline coefficients. This is useful as an alternative to energy minimization when energy minimization makes the training time too long.
energy_weight	0.0001	None	['Integral', 'float']	The weight of the energy minimization terms
extrapolate	False	None	['bool']	Whether to perform linear extrapolation for external evaluation points
min_energy	True	None	['bool']	Whether to perform energy minimization
approx_order	4	None	['Integral']	Exponent in the approximation term
solver	krylov	['krylov-dense', 'dense-lu', 'dense-chol', 'lu', 'ilu', 'krylov', 'krylov-lu', 'krylov-mg', 'gs', 'jacobi', 'mg', 'null']	['LinearSolver']	Linear solver
derivative_solver	krylov	['krylov-dense', 'dense-lu', 'dense-chol', 'lu', 'ilu', 'krylov', 'krylov-lu', 'krylov-mg', 'gs', 'jacobi', 'mg', 'null']	['LinearSolver']	Linear solver used for computing output derivatives (dy_dyt)
grad_weight	0.5	None	['Integral', 'float']	Weight on gradient training data
solver_tolerance	1e-12	None	['Integral', 'float']	Convergence tolerance for the nonlinear solver
nonlinear_maxiter	10	None	['Integral']	Maximum number of nonlinear solver iterations
line_search	backtracking	['backtracking', 'bracketed', 'quadratic', 'cubic', 'null']	['LineSearch']	Line search algorithm
save_energy_terms	False	None	['bool']	Whether to cache energy terms in the data_dir directory
data_dir	None	[None]	['str']	Directory for loading / saving cached data; None means do not

Options (RMTC)

Table 4: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
xlimits	None	None	['ndarray']	Lower/upper bounds in each dimension - ndarray [nx, 2]
smoothness	1.0	None	['Integral', 'float', 'tuple', 'list', 'ndarray']	Smoothness parameter in each dimension - length nx. None implies uniform
regularization_weight	1e-14	None	['Integral', 'float']	Weight of the term penalizing the norm of the spline coefficients. This is useful as an alternative to energy minimization when energy minimization makes the training time too long.
energy_weight	0.0001	None	['Integral', 'float']	The weight of the energy minimization terms
extrapolate	False	None	['bool']	Whether to perform linear extrapolation for external evaluation points
min_energy	True	None	['bool']	Whether to perform energy minimization
approx_order	4	None	['Integral']	Exponent in the approximation term
solver	krylov	['krylov-dense', 'dense-lu', 'dense-chol', 'lu', 'ilu', 'krylov', 'krylov-lu', 'krylov-mg', 'gs', 'jacobi', 'mg', 'null']	['LinearSolver']	Linear solver
derivative_solver	krylov	['krylov-dense', 'dense-lu', 'dense-chol', 'lu', 'ilu', 'krylov', 'krylov-lu', 'krylov-mg', 'gs', 'jacobi', 'mg', 'null']	['LinearSolver']	Linear solver used for computing output derivatives (dy_dyt)
grad_weight	0.5	None	['Integral', 'float']	Weight on gradient training data
solver_tolerance	1e-12	None	['Integral', 'float']	Convergence tolerance for the nonlinear solver
nonlinear_maxiter	10	None	['Integral']	Maximum number of nonlinear solver iterations
line_search	backtracking	['backtracking', 'bracketed', 'quadratic', 'cubic', 'null']	['LineSearch']	Line search algorithm
save_energy_terms	False	None	['bool']	Whether to cache energy terms in the data_dir directory
data_dir	None	[None]	['str']	Directory for loading / saving cached data; None means do not

3.2.4 Least-squares approximation

The following description is taken from scikit-learn version 0.18.2[1].

The Least Squares method fits a linear model with coefficients $\beta = (\beta_0, \beta_1, \ldots, \beta_{nx})$ to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{\beta} ||\mathbf{X}\beta - \mathbf{y}||_2^2,$$

where $\mathbf{X} = \left(1, \mathbf{x}^{(1)^T}, \ldots, \mathbf{x}^{(nt)^T}\right)^T$ with dimensions $(nt \times nx + 1)$.

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import LS

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.9, 1.0])

sm = LS()
sm.set_training_values(xt, yt)
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)

plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction"])
plt.show()
```

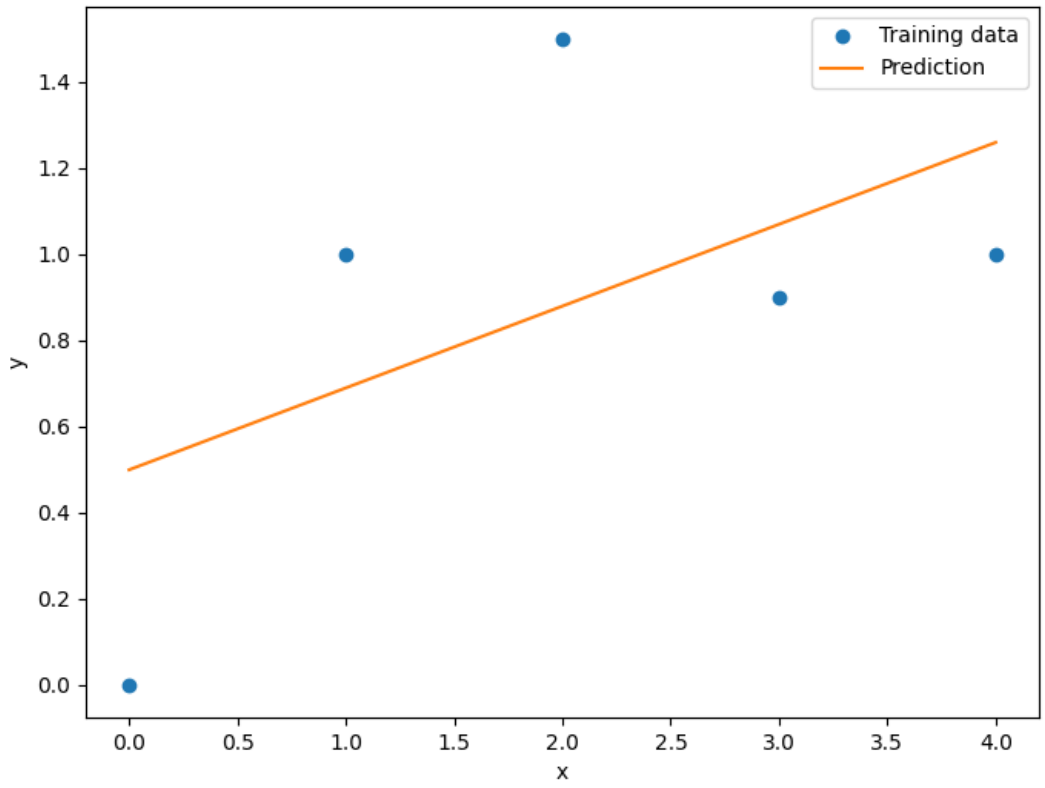
```
___________________________________________________________________________

                                   LS
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training
```

(continues on next page)

[1] http://scikit-learn.org/stable/modules/linear_model.html

(continued from previous page)

```
   Training ...
   Training - done. Time (sec):  0.0000000
_______________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```



Options

Table 5: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
data_dir	None	None	['str']	Directory for loading / saving cached data; None means do not save or load

3.2.5 Second-order polynomial approximation

The square polynomial model can be expressed by

$$\mathbf{y} = \mathbf{X}\beta + \epsilon,$$

where ϵ is a vector of random errors and

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_{nx}^{(1)} & x_1^{(1)}x_2^{(1)} & \dots & x_{nx-1}^{(1)}x_{nx}^{(1)} & {x_1^{(1)}}^2 & \dots & {x_{nx}^{(1)}}^2 \\ \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \\ 1 & x_1^{(nt)} & \dots & x_{nx}^{(nt)} & x_1^{(nt)}x_2^{(nt)} & \dots & x_{nx-1}^{(nt)}x_{nx}^{(nt)} & {x_1^{(nt)}}^2 & \dots & {x_{nx}^{(nt)}}^2 \end{bmatrix}.$$

The vector of estimated polynomial regression coefficients using ordinary least square estimation is

$$\beta = \mathbf{X^T X}^{-1}\mathbf{X^T y}.$$

Usage

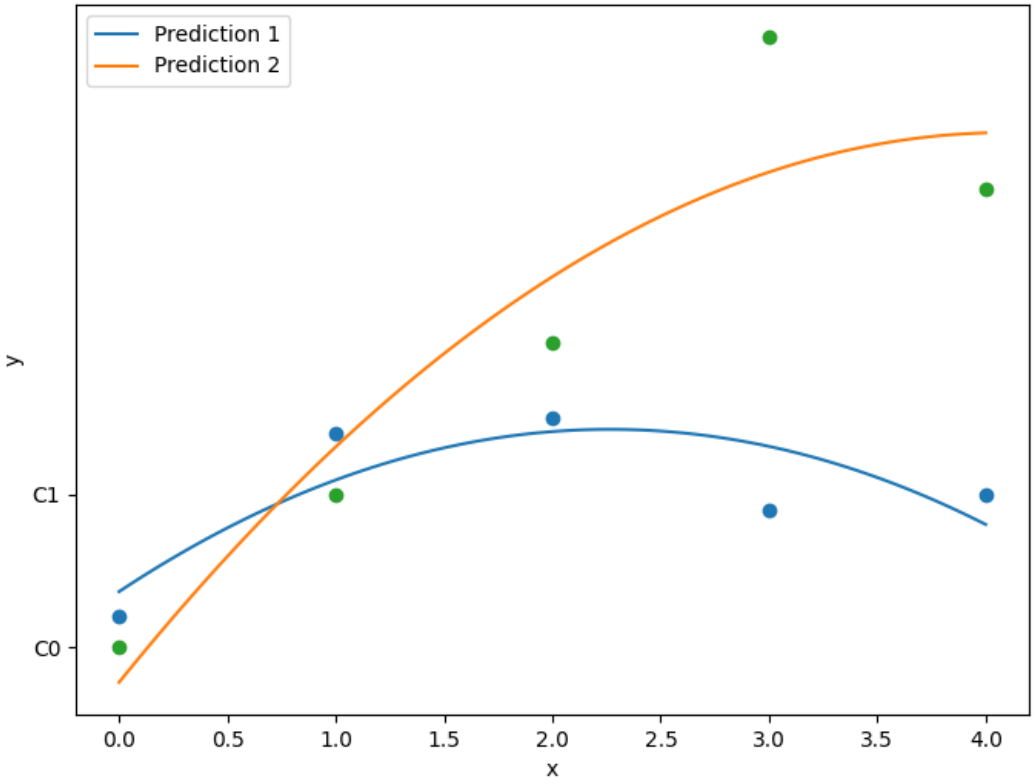
```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import QP

xt = np.array([[0.0, 1.0, 2.0, 3.0, 4.0]]).T
yt = np.array([[0.2, 1.4, 1.5, 0.9, 1.0], [0.0, 1.0, 2.0, 4, 3]]).T

sm = QP()
sm.set_training_values(xt, yt)
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
```

(continues on next page)

(continued from previous page)

```
y = sm.predict_values(x)

t1, _ = plt.plot(xt, yt[:, 0], "o", "C0")
p1 = plt.plot(x, y[:, 0], "C0", label="Prediction 1")
t2, _ = plt.plot(xt, yt[:, 1], "o", "C1")
p2 = plt.plot(x, y[:, 1], "C1", label="Prediction 2")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
___________________________________________________________________________

                                   QP
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.0000000
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```


Prediction 1
Prediction 2
y
C1
C0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

Options

Table 6: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
data_dir	None	None	['str']	Directory for loading / saving cached data; None means do not save or load

3.2.6 Kriging

Kriging is an interpolating model that is a linear combination of a known function $f_i(\mathbf{x})$ which is added to a realization of a stochastic process $Z(\mathbf{x})$

$$\hat{y} = \sum_{i=1}^{k} \beta_i f_i(\mathbf{x}) + Z(\mathbf{x}).$$

$Z(\mathbf{x})$ is a realization of a stochastic process with mean zero and spatial covariance function given by

$$cov\left[Z\left(\mathbf{x}^{(i)}\right), Z\left(\mathbf{x}^{(j)}\right)\right] = \sigma^2 R\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$$

where σ^2 is the process variance, and R is the correlation. Four types of correlation functions are available in SMT.

Exponential correlation function (Ornstein-Uhlenbeck process):

$$\prod_{l=1}^{nx} \exp\left(-\theta_l \left|x_l^{(i)} - x_l^{(j)}\right|\right), \quad \forall\, \theta_l \in \mathbb{R}^+$$

Squared Exponential (Gaussian) correlation function:

$$\prod_{l=1}^{nx} \exp\left(-\theta_l \left(x_l^{(i)} - x_l^{(j)}\right)^2\right), \quad \forall\, \theta_l \in \mathbb{R}^+$$

Matérn 5/2 correlation function:

$$\prod_{l=1}^{nx} \left(1 + \sqrt{5}\theta_l \left|x_l^{(i)} - x_l^{(j)}\right| + \frac{5}{3}\theta_l^2 \left(x_l^{(i)} - x_l^{(j)}\right)^2\right) \exp\left(-\sqrt{5}\theta_l \left|x_l^{(i)} - x_l^{(j)}\right|\right), \quad \forall\, \theta_l \in \mathbb{R}^+$$

Matérn 3/2 correlation function:

$$\prod_{l=1}^{nx} \left(1 + \sqrt{3}\theta_l \left|x_l^{(i)} - x_l^{(j)}\right|\right) \exp\left(-\sqrt{3}\theta_l \left|x_l^{(i)} - x_l^{(j)}\right|\right), \quad \forall\, \theta_l \in \mathbb{R}^+$$

These correlation functions are called by 'abs_exp' (exponential), 'squar_exp' (Gaussian), 'matern52' and 'matern32' in SMT.

The deterministic term $\sum_{i=1}^{k} \beta_i f_i(\mathbf{x})$ can be replaced by a constant, a linear model, or a quadratic model. These three types are available in SMT.

In the implementations, data are normalized by substracting the mean from each variable (indexed by columns in X), and then dividing the values of each variable by its standard deviation:

$$X_{\text{norm}} = \frac{X - X_{\text{mean}}}{X_{\text{std}}}$$

More details about the Kriging approach could be found in[1].

Kriging with categorical or integer variables

The goal is to be able to build a model for mixed typed variables. This algorithm has been presented by Garrido-Merchán and Hernández-Lobato in 2020[2].

To incorporate integer (with order relation) and categorical variables (with no order), we used continuous relaxation. For integer, we add a continuous dimension with the same bounds and then we round in the prediction to the closer integer. For categorical, we add as many continuous dimensions with bounds [0,1] as possible output values for the variable and then we round in the prediction to the output dimension giving the greatest continuous prediction.

A special case is the use of the Gower distance to handle mixed integer variables (hence the *gower* kernel/correlation model option). See the MixedInteger Tutorial for such usage.

More details available in[2]. See also *Mixed-Integer Sampling and Surrogate (Continuous Relaxation)*.

Implementation Note: Mixed variables handling is available for all Kriging models (KRG, KPLS or KPLSK) but cannot be used with derivatives computation.

Usage

Example 1

```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import KRG

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.9, 1.0])

sm = KRG(theta0=[1e-2])
sm.set_training_values(xt, yt)
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)
```

(continues on next page)

[1] Sacks, J. and Schiller, S. B. and Welch, W. J., Designs for computer experiments, Technometrics 31 (1) (1989) 41–47.

[2] E. C. Garrido-Merchan and D. Hernandez-Lobato, Dealing with categorical and integer-valued variables in Bayesian Optimization with Gaussian processes, Neurocomputing 380 (2020) 20-–35.

(continued from previous page)

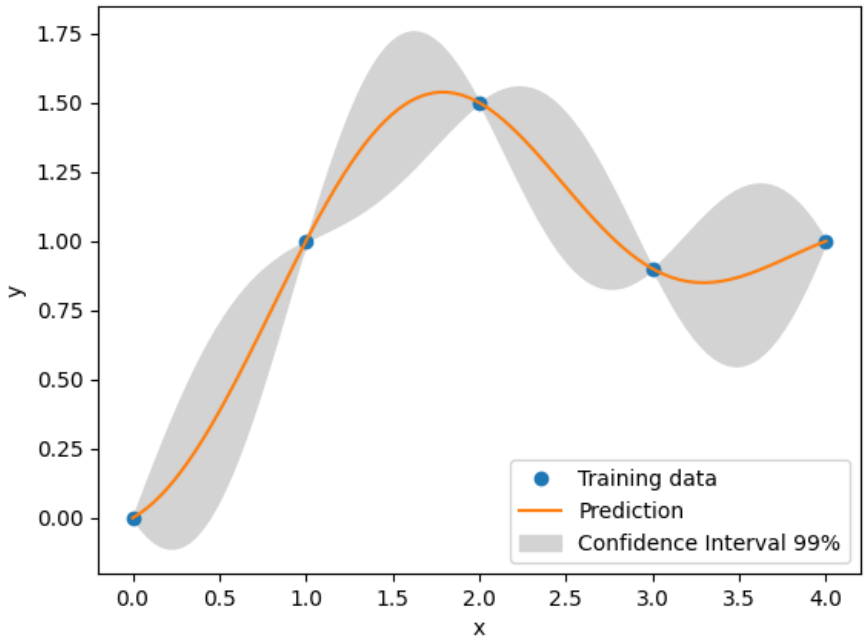
```
# estimated variance
s2 = sm.predict_variances(x)
# derivative according to the first variable
dydx = sm.predict_derivatives(xt, 0)
fig, axs = plt.subplots(1)

# add a plot with variance
axs.plot(xt, yt, "o")
axs.plot(x, y)
axs.fill_between(
    np.ravel(x),
    np.ravel(y - 3 * np.sqrt(s2)),
    np.ravel(y + 3 * np.sqrt(s2)),
    color="lightgrey",
)
axs.set_xlabel("x")
axs.set_ylabel("y")
axs.legend(
    ["Training data", "Prediction", "Confidence Interval 99%"],
    loc="lower right",
)

plt.show()
```

```
___________________________________________________________________________

                                  Kriging
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.0247462
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________
```

(continues on next page)

(continued from previous page)

```
Evaluation

     # eval points. : 5

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000
```



Example 2 with mixed variables

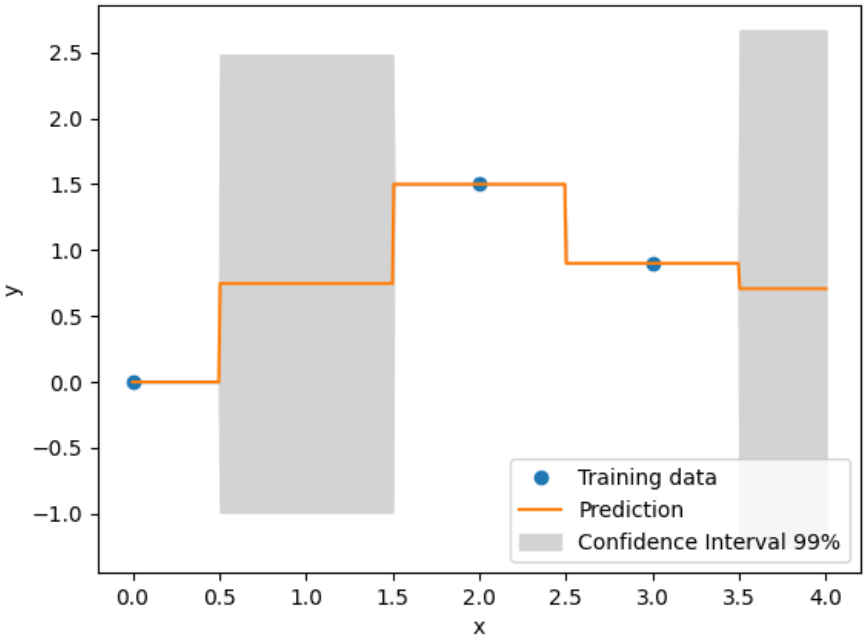
```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import KRG
from smt.applications.mixed_integer import MixedIntegerSurrogateModel, INT

xt = np.array([0.0, 2.0, 3.0])
yt = np.array([0.0, 1.5, 0.9])

# xtypes = [FLOAT, INT, (ENUM, 3), (ENUM, 2)]
# FLOAT means x1 continuous
# INT means x2 integer
# (ENUM, 3) means x3, x4 & x5 are 3 levels of the same categorical variable
# (ENUM, 2) means x6 & x7 are 2 levels of the same categorical variable
```

(continues on next page)

(continued from previous page)

```
sm = MixedIntegerSurrogateModel(
    xtypes=[INT], xlimits=[[0, 4]], surrogate=KRG(theta0=[1e-2])
)
sm.set_training_values(xt, yt)
sm.train()

num = 500
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)
# estimated variance
s2 = sm.predict_variances(x)

fig, axs = plt.subplots(1)
axs.plot(xt, yt, "o")
axs.plot(x, y)
axs.fill_between(
    np.ravel(x),
    np.ravel(y - 3 * np.sqrt(s2)),
    np.ravel(y + 3 * np.sqrt(s2)),
    color="lightgrey",
)
axs.set_xlabel("x")
axs.set_ylabel("y")
axs.legend(
    ["Training data", "Prediction", "Confidence Interval 99%"],
    loc="lower right",
)

plt.show()
```

```
___________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```


2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x
Training data
Prediction
Confidence Interval 99%

Options

Table 7: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
poly	constant	['constant', 'linear', 'quadratic']	['str']	Regression function type
corr	squar_exp	['abs_exp', 'squar_exp', 'matern52', 'matern32']	['str']	Correlation function type
categorical_kernel	None	['gower', 'homoscedastic_gaussian_matrix_kernel', 'full_gaussian_matrix_kernel']	['str']	The kernel to use for categorical inputs. Only for non continuous Kriging
xtypes	None	None	['list']	x type specifications: either FLOAT for continuous, INT for integer or (ENUM n) for categorical dimension with n levels
nugget	2.220446049250313e-14	None	['float']	a jitter for numerical stability
theta0	[0.01]	None	['list', 'ndarray']	Initial hyperparameters
theta_bounds	[1e-06, 20.0]	None	['list', 'ndarray']	bounds for hyperparameters
hyper_opt	Cobyla	['Cobyla', 'TNC']	['str']	Optimiser for hyperparameters optimisation
eval_noise	False	[True, False]	['bool']	noise evaluation flag
noise0	[0.0]	None	['list', 'ndarray']	Initial noise hyperparameters
noise_bounds	[2.220446049250313e-14, 10000000000.0]	None	['list', 'ndarray']	bounds for noise hyperparameters
use_het_noise	False	[True, False]	['bool']	heteroscedastic noise evaluation flag
n_start	10	None	['int']	number of optimizer runs (multistart method)

3.2.7 KPLS

KPLS is a kriging model that uses the partial least squares (PLS) method. KPLS is faster than kriging because of the low number of hyperparameters to be estimated while maintaining a good accuracy. This model is suitable for high-dimensional problems due to the kernel constructed through the PLS method. The PLS method[1] is a well known tool for high-dimensional problems that searches the direction that maximizes the variance between the input and output variables. This is done by a projection in a smaller space spanned by the so-called principal components. The PLS information are integrated into the kriging correlation matrix to scale the number of inputs by reducing the number of hyperparameters. The number of principal components h , which corresponds to the number of hyperparameters for KPLS and much lower than nx (number of dimension of the problem), usually does not exceed 4 components:

$$\prod_{l=1}^{nx} \exp\left(-\theta_l \left(x_l^{(i)} - x_l^{(j)}\right)^2\right), \qquad \prod_{k=1}^{h}\prod_{l=1}^{nx} \exp\left(-\theta_k \left(w_{*l}^{(k)} x_l^{(i)} - w_{*l}^{(k)} x_l^{(j)}\right)^2\right) \quad \forall\, \theta_l, \theta_k \in \mathbb{R}^+$$

Standard Gaussian correlation function PLS-Gaussian correlation function

Both absolute exponential and squared exponential kernels are available for KPLS model. More details about the KPLS approach could be found in these sources[2].

For automatic selection of the optimal number of components, the adjusted Wold's R criterion is implemented[3].

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import KPLS

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.9, 1.0])

sm = KPLS(theta0=[1e-2])
sm.set_training_values(xt, yt)
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)
# estimated variance
s2 = sm.predict_variances(x)
# to compute the derivative according to the first variable
dydx = sm.predict_derivatives(xt, 0)

plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.xlabel("x")
```

(continues on next page)

[1] Wold, H., Soft modeling by latent variables: the nonlinear iterative partial least squares approach, Perspectives in probability and statistics, papers in honour of MS Bartlett, 1975, pp. 520–540.

[2] Bouhlel, M. A. and Bartoli, N. and Otsmane, A. and Morlier, J., Improving kriging surrogates of high-dimensional design models by Partial Least Squares dimension reduction, Structural and Multidisciplinary Optimization, Vol. 53, No. 5, 2016, pp. 935–952.

[3] B. Li, J. Morris, and E. Martin. Model selection for partial least squares regression.Chemometrics and Intelligent Laboratory Systems, 64(1):79–89, 2002. ISSN 0169-7439.

(continued from previous page)

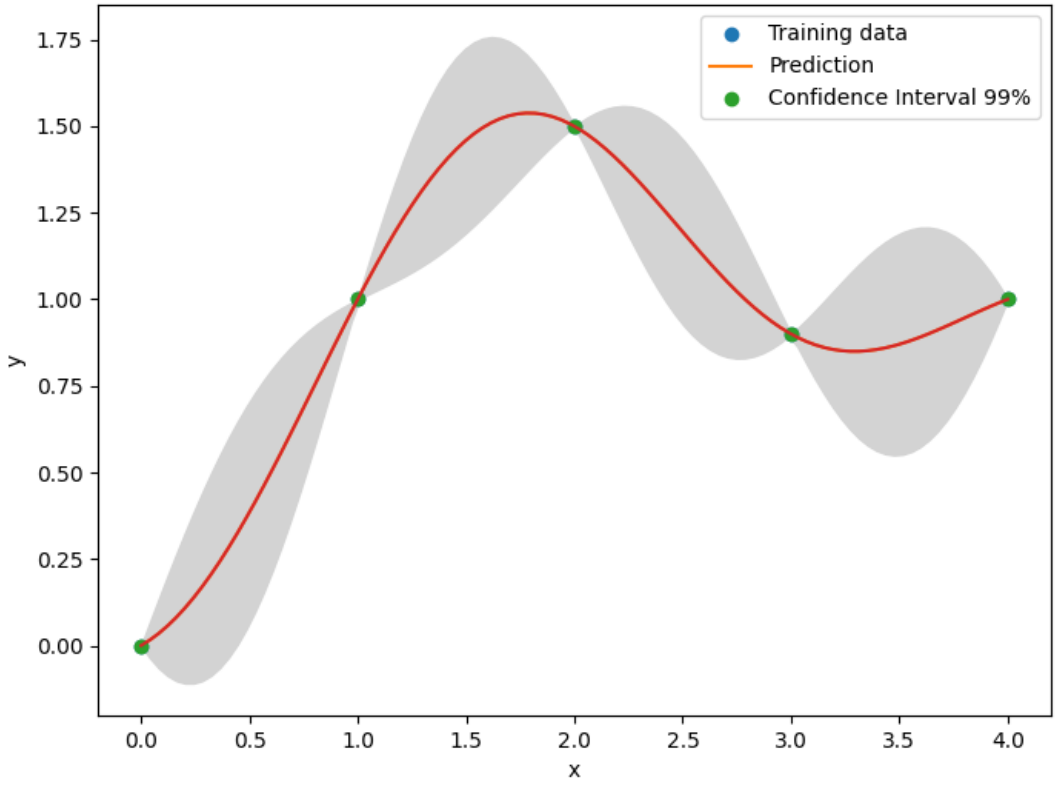
```
plt.ylabel("y")
plt.legend(["Training data", "Prediction"])
plt.show()

# add a plot with variance
plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.fill_between(
    np.ravel(x),
    np.ravel(y - 3 * np.sqrt(s2)),
    np.ravel(y + 3 * np.sqrt(s2)),
    color="lightgrey",
)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction", "Confidence Interval 99%"])
plt.show()
```

```
___________________________________________________________________________

                                  KPLS
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.0250795
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

      # eval points. : 5

   Predicting ...
   Predicting - done. Time (sec):  0.0000000
```

(continues on next page)

(continued from previous page)

```
Prediction time/pt. (sec) :  0.0000000
```



Options

Table 8: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
poly	constant	['constant', 'linear', 'quadratic']	['str']	Regression function type
corr	squar_exp	['abs_exp', 'squar_exp']	['str']	Correlation function type
categorical_kernel	None	['gower', 'homoscedastic_gaussian_matrix_kernel', 'full_gaussian_matrix_kernel']	['str']	The kernel to use for categorical inputs. Only for non continuous Kriging
xtypes	None	None	['list']	x type specifications: either FLOAT for continuous, INT for integer or (ENUM n) for categorical dimension with n levels
nugget	2.220446049250313e-14	None	['float']	a jitter for numerical stability
theta0	[0.01]	None	['list', 'ndarray']	Initial hyperparameters
theta_bounds	[1e-06, 20.0]	None	['list', 'ndarray']	bounds for hyperparameters
hyper_opt	Cobyla	['Cobyla', 'TNC']	['str']	Optimiser for hyperparameters optimisation
eval_noise	False	[True, False]	['bool']	noise evaluation flag
noise0	[0.0]	None	['list', 'ndarray']	Initial noise hyperparameters
noise_bounds	[2.220446049250313e-14, 10000000000.0]	None	['list', 'ndarray']	bounds for noise hyperparameters
use_het_noise	False	[True, False]	['bool']	heteroscedastic noise evaluation flag
n_start	10	None	['int']	number of optimizer runs (multistart method)
n_comp	1	None	['int']	Number of principal components
eval_n_comp	False	[True, False]	['bool']	n_comp evaluation flag
eval_comp_treshold	1.0	None	['float']	n_comp evaluation treshold for Wold's R criterion

3.2.8 KPLSK

KPLSK is a KPLS-based model and is basically built in two steps. The first step consists in running KPLS and giving the estimate hyperparameters expressed in the reduced space with a number of dimensions equals to h. The second step consists in expressing the estimate hyperparameters in the original space with a number of dimensions equals to nx, and then using it as a starting point to locally optimizing the likelihood function of a standard kriging. The idea here is guessing a "good" initial hyperparameters and applying a gradient-based optimization using a classic kriging-kernels. The "good" guess will be provided by KPLS: the solutions $(\theta_1^*, \ldots, \theta_h^*)$ and the PLS-coefficients $\left(w_1^{(k)}, \ldots, w_{nx}^{(k)}\right)$ for $k = 1, \ldots, h$. By a change of variables $\eta_l = \sum_{k=1}^{h} \theta_k^* {w_l^{(k)}}^2$, for $l = 1, \ldots, nx$, we can express the initial hyperparameters point in the original space. In the following example, a KPLS-Gaussian kernel function k_{KPLS} is used for the demonstration (More details are given in[1]):

$$
\begin{aligned}
k_{\text{KPLS}}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) &= \\
\sigma \prod_{k=1}^{h} \prod_{l=1}^{nx} \exp\left(-\theta_k {w_l^{(k)}}^2 \left(x_l^{(i)} - x_l^{(j)}\right)^2\right) &= \\
\sigma \exp\left(\sum_{l=1}^{nx} \sum_{k=1}^{h} -\theta_k {w_l^{(k)}}^2 \left(x_l^{(i)} - x_l^{(j)}\right)^2\right) \qquad &\text{Change of variables} \\
= \\
\sigma \exp\left(\sum_{l=1}^{nx} -\eta_l \left(x_l^{(i)} - x_l^{(j)}\right)^2\right) &= \\
\sigma \prod_{l=1}^{nx} \exp\left(-\eta_l \left(x_l^{(i)} - x_l^{(j)}\right)^2\right). &
\end{aligned}
$$

$\prod_{l=1}^{nx} \exp\left(-\eta_l \left(x_l^{(i)} - x_l^{(j)}\right)^2\right)$ is a standard Gaussian kernel function.

Subsequently, the hyperparameters point $\left(\eta_1 = \sum_{k=1}^{h} \theta_k^* {w_1^{(k)}}^2, \ldots, \eta_{nx} = \sum_{k=1}^{h} \theta_k^* {w_{nx}^{(k)}}^2\right)$ is used as a starting point for a gradient-based optimization applied on a standard kriging method.

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import KPLSK

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.9, 1.0])

sm = KPLSK(theta0=[1e-2])
sm.set_training_values(xt, yt)
sm.train()
```

(continues on next page)

[1] Bouhlel, M. A., Bartoli, N., Otsmane, A., and Morlier, J., An Improved Approach for Estimating the Hyperparameters of the Kriging Model for High-Dimensional Problems through the Partial Least Squares Method," Mathematical Problems in Engineering, vol. 2016, Article ID 6723410, 2016.

(continued from previous page)

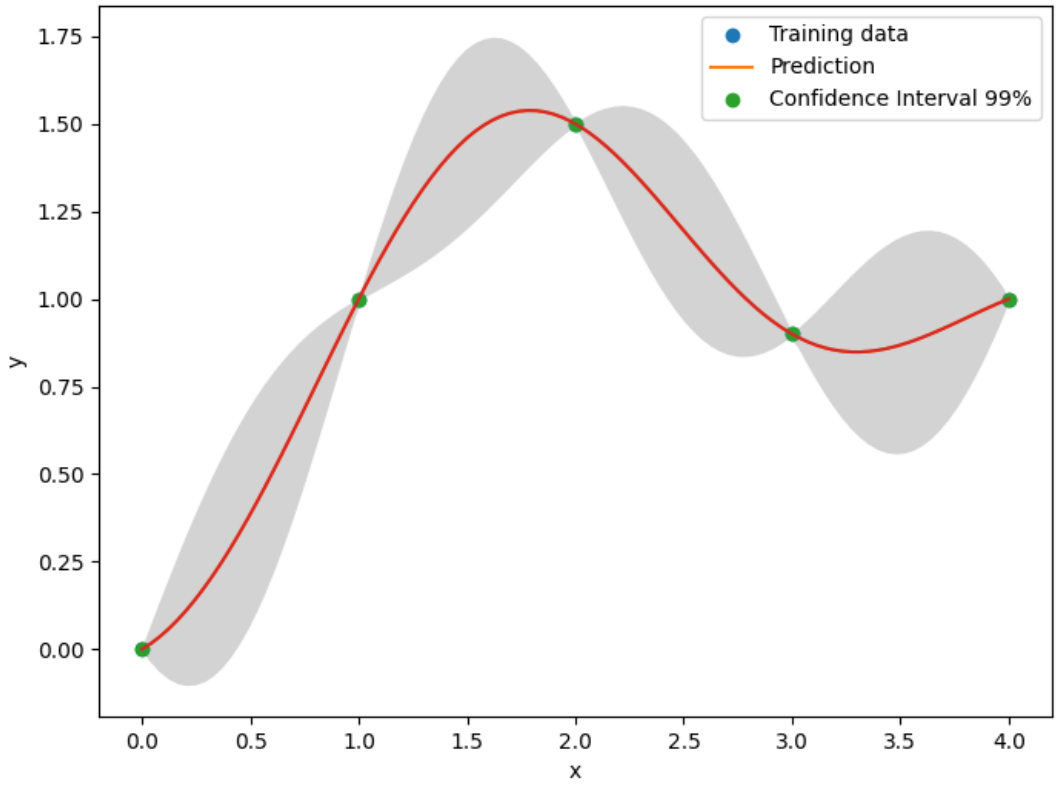
```
num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)
# estimated variance
s2 = sm.predict_variances(x)
# derivative according to the first variable
dydx = sm.predict_derivatives(xt, 0)

plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction"])
plt.show()

# add a plot with variance
plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.fill_between(
    np.ravel(x),
    np.ravel(y - 3 * np.sqrt(s2)),
    np.ravel(y + 3 * np.sqrt(s2)),
    color="lightgrey",
)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction", "Confidence Interval 99%"])
plt.show()
```

```
___________________________________________________________________________

                                   KPLSK
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.0498002
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
```

(continues on next page)

(continued from previous page)

```
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

------------------------------------------------------------------------------

Evaluation

     # eval points. : 5

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000
```



Options

Table 9: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
poly	constant	['constant', 'linear', 'quadratic']	['str']	Regression function type
corr	squar_exp	['squar_exp']	['str']	Correlation function type
categorical_kernel	None	['gower', 'homoscedastic_gaussian_matrix_kernel', 'full_gaussian_matrix_kernel']	['str']	The kernel to use for categorical inputs. Only for non continuous Kriging
xtypes	None	None	['list']	x type specifications: either FLOAT for continuous, INT for integer or (ENUM n) for categorical dimension with n levels
nugget	2.220446049250313e-14	None	['float']	a jitter for numerical stability
theta0	[0.01]	None	['list', 'ndarray']	Initial hyperparameters
theta_bounds	[1e-06, 20.0]	None	['list', 'ndarray']	bounds for hyperparameters
hyper_opt	Cobyla	['Cobyla', 'TNC']	['str']	Optimiser for hyperparameters optimisation
eval_noise	False	[True, False]	['bool']	noise evaluation flag
noise0	[0.0]	None	['list', 'ndarray']	Initial noise hyperparameters
noise_bounds	[2.220446049250313e-14, 10000000000.0]	None	['list', 'ndarray']	bounds for noise hyperparameters
use_het_noise	False	[True, False]	['bool']	heteroscedastic noise evaluation flag
n_start	10	None	['int']	number of optimizer runs (multistart method)
n_comp	1	None	['int']	Number of principal components
eval_n_comp	False	[True, False]	['bool']	n_comp evaluation flag
eval_comp_treshold	1.0	None	['float']	n_comp evaluation treshold for Wold's R criterion

3.2.9 GEKPLS

GEKPLS is a gradient-enhaced kriging with partial least squares approach. Gradient-enhaced kriging (GEK) is an extention of kriging which supports gradient information[1]. GEK is usually more accurate than kriging, however, it is not computationally efficient when the number of inputs, the number of sampling points, or both, are high. This is mainly due to the size of the corresponding correlation matrix that increases proportionally with both the number of inputs and the number of sampling points.

To address these issues, GEKPLS exploits the gradient information with a slight increase of the size of the correlation matrix and reduces the number of hyperparameters. The key idea of GEKPLS consists in generating a set of approximating points around each sampling points using the first order Taylor approximation method. Then, the PLS method is applied several times, each time on a different number of sampling points with the associated sampling points. Each PLS provides a set of coefficients that gives the contribution of each variable nearby the associated sampling point to the output. Finally, an average of all PLS coefficients is computed to get the global influence to the output. Denoting these coefficients by $\left(w_1^{(k)}, \ldots, w_{nx}^{(k)}\right)$, the GEKPLS Gaussian kernel function is given by:

$$k\left(\mathbf{x^{(i)}}, \mathbf{x^{(j)}}\right) = \sigma \prod_{l=1}^{nx} \prod_{k=1}^{h} \exp\left(-\theta_k \left(w_l^{(k)} x_l^{(i)} - w_l^{(k)} x_l^{(j)}\right)^2\right)$$

This approach reduces the number of hyperparameters (reduced dimension) from nx to h with $nx >> h$.

As previously mentioned, PLS is applied several times with respect to each sampling point, which provides the influence of each input variable around that point. The idea here is to add only m approximating points $(m \in [1, nx])$ around each sampling point. Only the m highest coefficients given by the first principal component are considered, which usually contains the most useful information. More details of such approach are given in[2].

Usage

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

from smt.surrogate_models import GEKPLS
from smt.problems import Sphere
from smt.sampling_methods import LHS

# Construction of the DOE
fun = Sphere(ndim=2)
sampling = LHS(xlimits=fun.xlimits, criterion="m")
xt = sampling(20)
yt = fun(xt)
# Compute the gradient
for i in range(2):
    yd = fun(xt, kx=i)
    yt = np.concatenate((yt, yd), axis=1)

# Build the GEKPLS model
n_comp = 2
sm = GEKPLS(
```

(continues on next page)

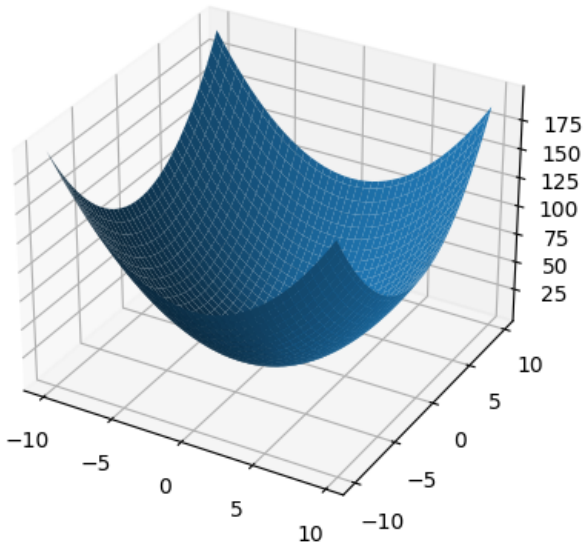
[1] Forrester, I. J. and Sobester, A. and Keane, A. J., Engineering Design via Surrogate Modeling: A Practical Guide. Wiley, 2008 (Chapter 7).

[2] Bouhlel, M. A., & Martins, J. R. (2019). Gradient-enhanced kriging for high-dimensional problems. Engineering with Computers, 35(1), 157-173.

(continued from previous page)

```
    theta0=[1e-2] * n_comp,
    xlimits=fun.xlimits,
    extra_points=1,
    print_prediction=False,
    n_comp=n_comp,
)
sm.set_training_values(xt, yt[:, 0])
for i in range(2):
    sm.set_training_derivatives(xt, yt[:, 1 + i].reshape((yt.shape[0], 1)), i)
sm.train()

# Test the model
X = np.arange(fun.xlimits[0, 0], fun.xlimits[0, 1], 0.25)
Y = np.arange(fun.xlimits[1, 0], fun.xlimits[1, 1], 0.25)
X, Y = np.meshgrid(X, Y)
Z = np.zeros((X.shape[0], X.shape[1]))

for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        Z[i, j] = sm.predict_values(
            np.hstack((X[i, j], Y[i, j])).reshape((1, 2))
        )

fig = plt.figure()
ax = fig.gca(projection="3d")
surf = ax.plot_surface(X, Y, Z)

plt.show()
```

```
___________________________________________________________________________

                                  GEKPLS
___________________________________________________________________________

 Problem size

      # training points.        : 20

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.0846102
```


175
150
125
100
75
50
25
10
5
0
−5
−10
−10
−5
0
5
10

Options

Table 10: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
poly	constant	['constant', 'linear', 'quadratic']	['str']	Regression function type
corr	squar_exp	['abs_exp', 'squar_exp']	['str']	Correlation function type
categorical_kernel	None	['gower', 'homoscedastic_gaussian_matrix_kernel', 'full_gaussian_matrix_kernel']	['str']	The kernel to use for categorical inputs. Only for non continuous Kriging
xtypes	None	None	['list']	x type specifications: either FLOAT for continuous, INT for integer or (ENUM n) for categorical dimension with n levels
nugget	2.220446049250313e-14	None	['float']	a jitter for numerical stability
theta0	[0.01]	None	['list', 'ndarray']	Initial hyperparameters
theta_bounds	[1e-06, 20.0]	None	['list', 'ndarray']	bounds for hyperparameters
hyper_opt	Cobyla	['Cobyla', 'TNC']	['str']	Optimiser for hyperparameters optimisation
eval_noise	False	[True, False]	['bool']	noise evaluation flag
noise0	[0.0]	None	['list', 'ndarray']	Initial noise hyperparameters
noise_bounds	[2.220446049250313e-14, 10000000000.0]	None	['list', 'ndarray']	bounds for noise hyperparameters
use_het_noise	False	[True, False]	['bool']	heteroscedastic noise evaluation flag
n_start	10	None	['int']	number of optimizer runs (multistart method)
n_comp	2	None	['int']	Number of principal components
eval_n_comp	False	[True, False]	['bool']	n_comp evaluation flag
eval_comp_treshold	0.99	None	['float']	n_comp evaluation treshold for Wold's R criterion
xlimits	None	None	['ndarray']	Lower/upper bounds in each dimension - ndarray [nx, 2]
delta_x	0.0001	None	['int', 'float']	Step used in the FOTA
extra_points	0	None	['int']	Number of extra points per training point

3.2.10 GENN

Gradient-Enhanced Neural Networks (GENN) are fully connected multi-layer perceptrons, whose training process was modified to account for gradient information. Specifically, the parameters are learned by minimizing the Least Squares Estimator (LSE), modified to account for partial derivatives. The theory behind the algorithm can be found here, but suffice it to say that the model is trained in such a way so as to minimize not only the prediction error $y - f(x)$ of the response, but also the prediction error $dy/dx - f'(x)$ of the partial derivatives. The chief benefit of gradient-enhancement is better accuracy with fewer training points, compared to regular neural networks without gradient-enhancement. Note that GENN applies to regression (single-output or multi-output), but not classification since there is no gradient in that case. The implementation is fully vectorized and uses Adam optimization, mini-batch, and L2-norm regularization.

Limitations

Gradient-enhanced methods only apply to the special use-case of computer aided design, where data is generated synthetically using physics-based computer models, responses are continuous, and their gradient is defined. Furthermore, gradient enhancement is only beneficial when the cost of obtaining the gradient is not excessive in the first place. This is often true in computer-aided design with the advent of adjoint design methods for example, but it is not always the case. The user should therefore carefully weight the benefit of gradient-enhanced methods depending on the application.

Usage

```
import numpy as np
import matplotlib.pyplot as plt
from smt.surrogate_models.genn import GENN, load_smt_data

# Training data
lower_bound = -np.pi
upper_bound = np.pi
number_of_training_points = 4
xt = np.linspace(lower_bound, upper_bound, number_of_training_points)
yt = xt * np.sin(xt)
dyt_dxt = np.sin(xt) + xt * np.cos(xt)

# Validation data
number_of_validation_points = 30
xv = np.linspace(lower_bound, upper_bound, number_of_validation_points)
yv = xv * np.sin(xv)
dyv_dxv = np.sin(xv) + xv * np.cos(xv)

# Truth model
x = np.arange(lower_bound, upper_bound, 0.01)
y = x * np.sin(x)

# GENN
genn = GENN()
genn.options["alpha"] = 0.1  # learning rate that controls optimizer step size
genn.options["beta1"] = 0.9  # tuning parameter to control ADAM optimization
genn.options["beta2"] = 0.99  # tuning parameter to control ADAM optimization
genn.options[
    "lambd"
```

(continues on next page)

(continued from previous page)

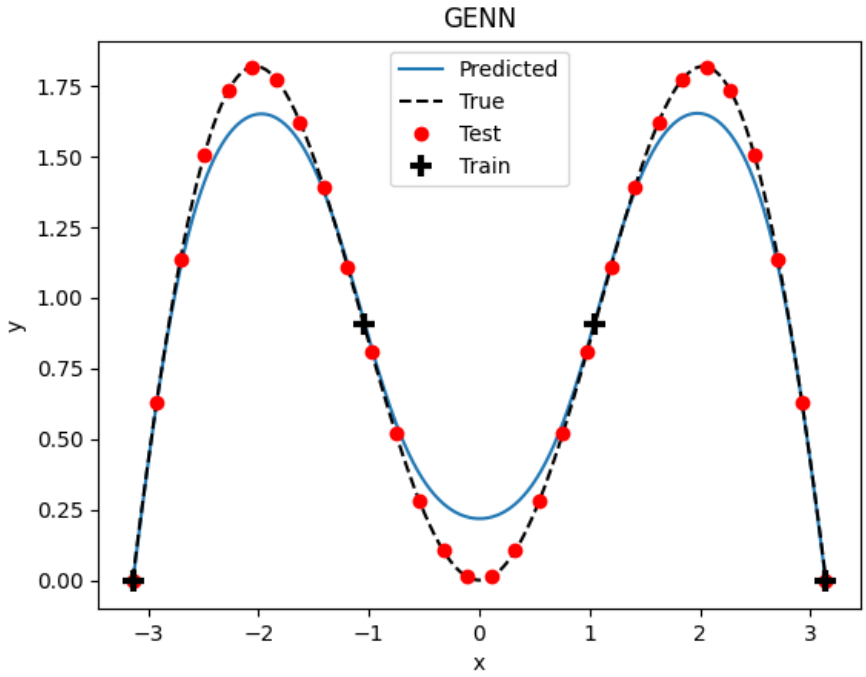
```
] = 0.1  # lambd = 0. = no regularization, lambd > 0 = regularization
genn.options[
    "gamma"
] = 1.0  # gamma = 0. = no grad-enhancement, gamma > 0 = grad-enhancement
genn.options["deep"] = 2  # number of hidden layers
genn.options["wide"] = 6  # number of nodes per hidden layer
genn.options[
    "mini_batch_size"
] = 64  # used to divide data into training batches (use for large data sets)
genn.options["num_epochs"] = 20  # number of passes through data
genn.options[
    "num_iterations"
] = 100  # number of optimizer iterations per mini-batch
genn.options["is_print"] = True  # print output (or not)
load_smt_data(
    genn, xt, yt, dyt_dxt
)  # convenience function to read in data that is in SMT format
genn.train()  # API function to train model
genn.plot_training_history()  # non-API function to plot training history (to check␣
↪convergence)
genn.goodness_of_fit(
    xv, yv, dyv_dxv
)  # non-API function to check accuracy of regression
y_pred = genn.predict_values(
    x
)  # API function to predict values at new (unseen) points

# Plot
fig, ax = plt.subplots()
ax.plot(x, y_pred)
ax.plot(x, y, "k--")
ax.plot(xv, yv, "ro")
ax.plot(xt, yt, "k+", mew=3, ms=10)
ax.set(xlabel="x", ylabel="y", title="GENN")
ax.legend(["Predicted", "True", "Test", "Train"])
plt.show()
```

```
___________________________________________________________________________

                                   GENN
___________________________________________________________________________

 Problem size

      # training points.        : 4

___________________________________________________________________________

 Training

   Training ...
epoch = 0, mini-batch = 0, avg cost = 22.881
```

(continues on next page)

(continued from previous page)

```
epoch = 1, mini-batch = 0, avg cost =  7.640
epoch = 2, mini-batch = 0, avg cost =  7.474
epoch = 3, mini-batch = 0, avg cost =  7.379
epoch = 4, mini-batch = 0, avg cost =  7.308
epoch = 5, mini-batch = 0, avg cost =  4.056
epoch = 6, mini-batch = 0, avg cost =  0.701
epoch = 7, mini-batch = 0, avg cost =  0.660
epoch = 8, mini-batch = 0, avg cost =  0.647
epoch = 9, mini-batch = 0, avg cost =  0.641
epoch = 10, mini-batch = 0, avg cost =  0.637
epoch = 11, mini-batch = 0, avg cost =  0.634
epoch = 12, mini-batch = 0, avg cost =  0.632
epoch = 13, mini-batch = 0, avg cost =  0.630
epoch = 14, mini-batch = 0, avg cost =  0.629
epoch = 15, mini-batch = 0, avg cost =  0.628
epoch = 16, mini-batch = 0, avg cost =  0.627
epoch = 17, mini-batch = 0, avg cost =  0.627
epoch = 18, mini-batch = 0, avg cost =  0.627
epoch = 19, mini-batch = 0, avg cost =  0.626
   Training - done. Time (sec):  3.2554069
________________________________________________________________________________

 Evaluation

      # eval points. : 629

   Predicting ...
   Predicting - done. Time (sec):  0.0000818

   Prediction time/pt. (sec) :  0.0000001
```



Options

Table 11: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
alpha	0.5	None	['int', 'float']	optimizer learning rate
beta1	0.9	None	['int', 'float']	Adam optimizer tuning parameter
beta2	0.99	None	['int', 'float']	Adam optimizer tuning parameter
lambd	0.1	None	['int', 'float']	regularization coefficient
gamma	1.0	None	['int', 'float']	gradient-enhancement coefficient
deep	2	None	['int']	number of hidden layers
wide	2	None	['int']	number of nodes per hidden layer
mini_batch_size	64	None	['int']	split data into batches of specified size
num_epochs	10	None	['int']	number of random passes through the data
num_iterations	100	None	['int']	number of optimizer iterations per mini-batch
seed	None	None	['int']	random seed to ensure repeatability of results when desired
is_print	True	None	['bool']	print progress (or not)

3.2.11 Marginal Gaussian Process (MGP)

Marginal Gaussian Processes (MGP) are Gaussian Processes taking into account the uncertainty of the hyperparameters defined as a density probability function. Especially we suppose that the function to model $f : \Omega \mapsto \mathbb{R}$, where $\Omega \subset \mathbb{R}^d$ and d is the number of design variables, lies in a linear embedding $\mathcal{A}$ such as $\mathcal{A} = \{u = Ax, x \in \Omega\}$, $A \in \mathbb{R}^{d \times d_e}$ and $f(x) = f_{\mathcal{A}}(Ax)$ with $f(x) = f_{\mathcal{A}} : \mathcal{A} \mapsto \mathbb{R}$ et $d_e \ll d$.

Then, we must use a kernel $k(x, x') = k_{\mathcal{A}}(Ax, Ax')$ whose each component of the transfert matrix A is an hyperparameter. Thus we have $d_e \times d$ hyperparameters to find. (Note that d_e is defined as n_comp in the code).

Moreover, we suppose that A follows a normal distribution $\mathcal{N}(vect(\hat{A}), \Sigma)$, with a mean vector $vect(\hat{A})$ and a covariance matrix Σ, which are taken into account in the Gaussian Process building. Using a Gaussian Process with $\hat{A}$ as hyperparameters and some approximations, we obtain the prediction and variance of the MGP considering the normal distribution of A. For more theory on the MGP, refer to[1].

Limitations

This implementation only considers a Gaussian kernel. Moreover, the kernel $k_{\mathcal{A}}$ is a uniform Gaussian kernel. The training of the MGP can be very time consuming.

Usage

```
import numpy as np
import matplotlib.pyplot as plt
from smt.surrogate_models import MGP
from smt.sampling_methods import LHS

# Construction of the DOE
dim = 3

def fun(x):
    import numpy as np

    res = (
        np.sum(x, axis=1) ** 2
        - np.sum(x, axis=1)
        + 0.2 * (np.sum(x, axis=1) * 1.2) ** 3
    )
    return res

sampling = LHS(xlimits=np.asarray([(-1, 1)] * dim), criterion="m")
xt = sampling(8)
yt = np.atleast_2d(fun(xt)).T

# Build the MGP model
sm = MGP(
    theta0=[1e-2],
    print_prediction=False,
    n_comp=1,
)
```

(continues on next page)

[1] Garnett, R., Osborne, M. & Hennig, P. Active Learning of Linear Embeddings for Gaussian Processes. In Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI 2014) 10 (2014).

(continued from previous page)

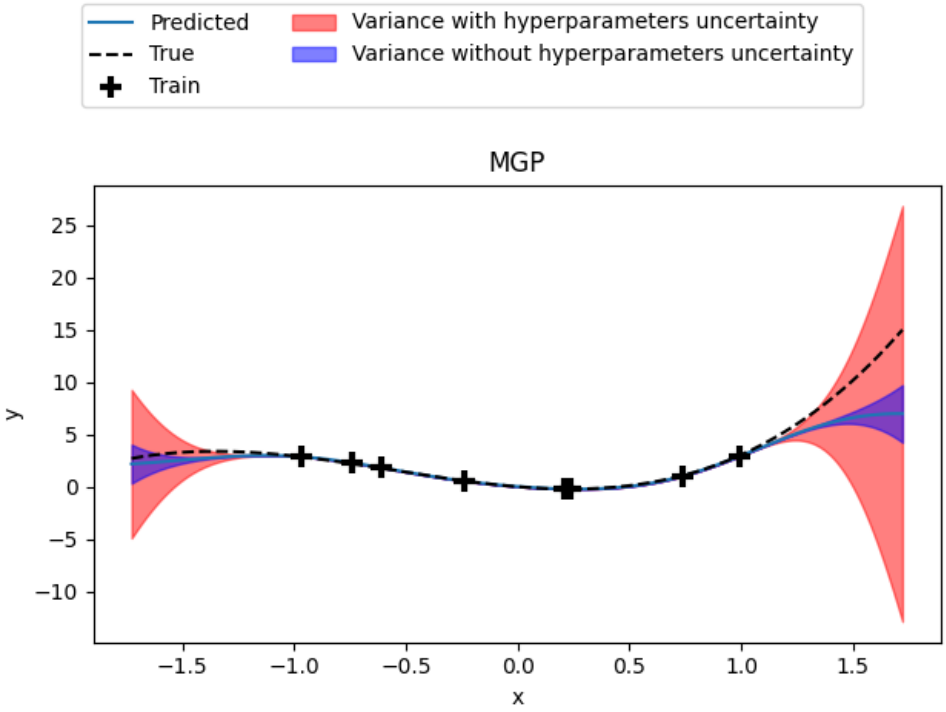
```
sm.set_training_values(xt, yt[:, 0])
sm.train()

# Get the transfert matrix A
emb = sm.embedding["C"]

# Compute the smallest box containing all points of A
upper = np.sum(np.abs(emb), axis=0)
lower = -upper

# Test the model
u_plot = np.atleast_2d(np.arange(lower, upper, 0.01)).T
x_plot = sm.get_x_from_u(u_plot)  # Get corresponding points in Omega
y_plot_true = fun(x_plot)
y_plot_pred = sm.predict_values(u_plot)
sigma_MGP = sm.predict_variances(u_plot)
sigma_KRG = sm.predict_variances_no_uq(u_plot)

u_train = sm.get_u_from_x(xt)  # Get corresponding points in A

# Plots
fig, ax = plt.subplots()
ax.plot(u_plot, y_plot_pred, label="Predicted")
ax.plot(u_plot, y_plot_true, "k--", label="True")
ax.plot(u_train, yt, "k+", mew=3, ms=10, label="Train")
ax.fill_between(
    u_plot[:, 0],
    y_plot_pred[:, 0] - 3 * sigma_MGP[:, 0],
    y_plot_pred[:, 0] + 3 * sigma_MGP[:, 0],
    color="r",
    alpha=0.5,
    label="Variance with hyperparameters uncertainty",
)
ax.fill_between(
    u_plot[:, 0],
    y_plot_pred[:, 0] - 3 * sigma_KRG[:, 0],
    y_plot_pred[:, 0] + 3 * sigma_KRG[:, 0],
    color="b",
    alpha=0.5,
    label="Variance without hyperparameters uncertainty",
)

ax.set(xlabel="x", ylabel="y", title="MGP")
fig.legend(loc="upper center", ncol=2)
fig.tight_layout()
fig.subplots_adjust(top=0.74)
plt.show()
```

```
_____________________________________________________________________________

                                    MGP

_____________________________________________________________________________
```

(continues on next page)

(continued from previous page)

```
Problem size

     # training points.        : 8

___________________________________________________________________________

Training

  Training ...
  Training - done. Time (sec):  1.2810287
```



Options

Table 12: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
poly	constant	['constant', 'linear', 'quadratic']	['str']	Regression function type
corr	squar_exp	['abs_exp', 'squar_exp', 'act_exp', 'matern52', 'matern32']	['str']	Correlation function type
categorical_kernel	None	['gower', 'homoscedastic_gaussian_matrix_kernel', 'full_gaussian_matrix_kernel']	['str']	The kernel to use for categorical inputs. Only for non continuous Kriging
xtypes	None	None	['list']	x type specifications: either FLOAT for continuous, INT for integer or (ENUM n) for categorical dimension with n levels
nugget	2.220446049250313e-14	None	['float']	a jitter for numerical stability
theta0	[0.01]	None	['list', 'ndarray']	Initial hyperparameters
theta_bounds	[1e-06, 20.0]	None	['list', 'ndarray']	bounds for hyperparameters
hyper_opt	Cobyla	['Cobyla', 'TNC']	['str']	Optimiser for hyperparameters optimisation
eval_noise	False	[True, False]	['bool']	noise evaluation flag
noise0	[0.0]	None	['list', 'ndarray']	Initial noise hyperparameters
noise_bounds	[2.220446049250313e-14, 10000000000.0]	None	['list', 'ndarray']	bounds for noise hyperparameters
use_het_noise	False	[True, False]	['bool']	heteroscedastic noise evaluation flag
n_start	10	None	['int']	number of optimizer runs (multistart method)
n_comp	1	None	['int']	Number of active dimensions
prior	{'mean': [0.0], 'var': 1.25}	None	['dict']	Parameters for Gaussian prior of the Hyperparameters

3.2.12 Usage

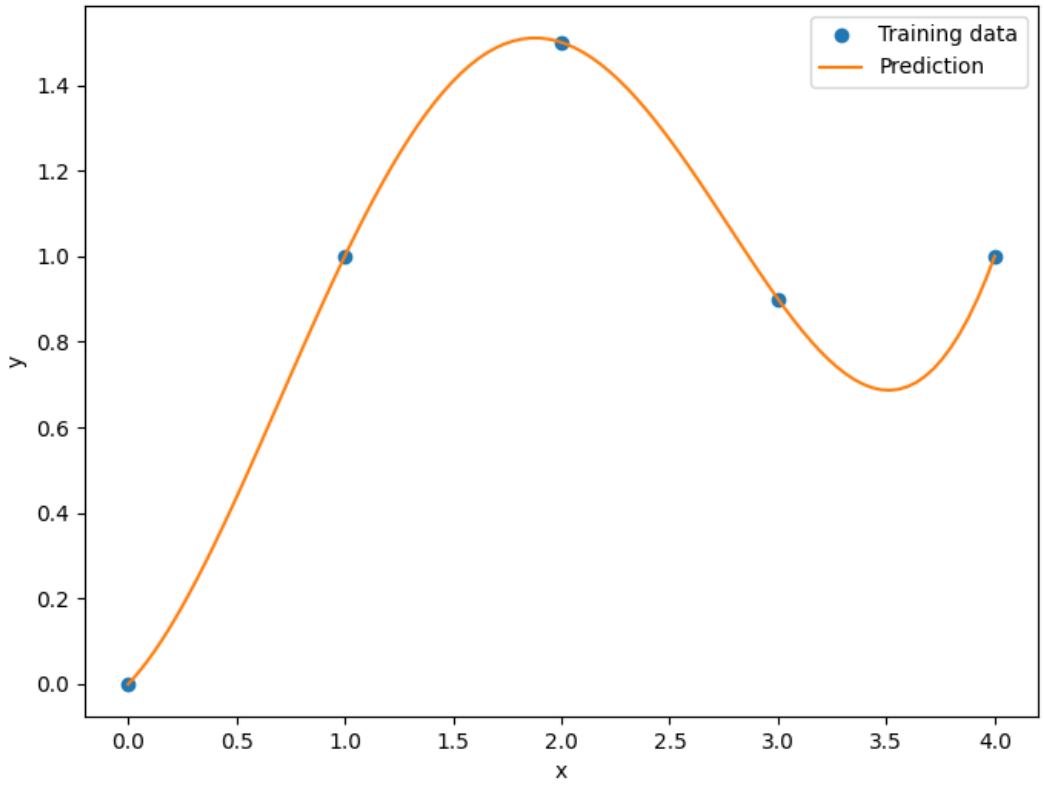
```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import RBF

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.9, 1.0])

sm = RBF(d0=5)
sm.set_training_values(xt, yt)
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)

plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction"])
plt.show()
```

```
___________________________________________________________________________

                                   RBF
___________________________________________________________________________

 Problem size

      # training points.        : 5

___________________________________________________________________________

 Training

   Training ...
      Initializing linear solver ...
         Performing LU fact. (5 x 5 mtx) ...
         Performing LU fact. (5 x 5 mtx) - done. Time (sec):  0.0000000
      Initializing linear solver - done. Time (sec):  0.0000000
      Solving linear system (col. 0) ...
         Back solving (5 x 5 mtx) ...
         Back solving (5 x 5 mtx) - done. Time (sec):  0.0000000
      Solving linear system (col. 0) - done. Time (sec):  0.0000000
   Training - done. Time (sec):  0.0000000
___________________________________________________________________________

 Evaluation

      # eval points. : 100
```

(continues on next page)

(continued from previous page)

```
Predicting ...
Predicting - done. Time (sec):  0.0000000

Prediction time/pt. (sec) :  0.0000000
```



3.2.13 SurrogateModel class API

All surrogate modeling methods implement the following API, though some of the functions in the API are not supported by all methods.

class smt.surrogate_models.surrogate_model.**SurrogateModel**(***kwargs*)

Base class for all surrogate models.

Examples

```
>>> from smt.surrogate_models import RBF
>>> sm = RBF(print_training=False)
>>> sm.options['print_prediction'] = False
```

Attributes

options
[OptionsDictionary] Dictionary of options. Options values can be set on this attribute directly or they can be passed in as keyword arguments during instantiation.

supports
[dict] Dictionary containing information about what this surrogate model supports.

Methods

predict_derivatives(x, kx)	Predict the dy_dx derivatives at a set of points.
predict_output_derivatives(x)	Predict the derivatives dy_dyt at a set of points.
predict_values(x)	Predict the output values at a set of points.
predict_variance_derivatives(x)	Predict the derivation of the variance at a point
predict_variances(x)	Predict the variances at a set of points.
set_training_derivatives(xt, dyt_dxt, kx[, name])	Set training data (derivatives).
set_training_values(xt, yt[, name])	Set training data (values).
train()	Train the model
update_training_derivatives(dyt_dxt, kx[, name])	Update the training data (values) at the previously set input values.
update_training_values(yt[, name])	Update the training data (values) at the previously set input values.

__init__(***kwargs*)

Constructor where values of options can be passed in.

For the list of options, see the documentation for the surrogate model being used.

Parameters

****kwargs**
[named arguments] Set of options that can be optionally set; each option must have been declared.

Examples

```
>>> from smt.surrogate_models import RBF
>>> sm = RBF(print_global=False)
```

set_training_values(*xt: ndarray*, *yt: ndarray*, *name=None*) → None

Set training data (values).

Parameters

xt
[np.ndarray[nt, nx] or np.ndarray[nt]] The input values for the nt training points.

yt
[np.ndarray[nt, ny] or np.ndarray[nt]] The output values for the nt training points.

name
[str or None] An optional label for the group of training points being set. This is only used in special situations (e.g., multi-fidelity applications).

`set_training_derivatives`(*xt: ndarray*, *dyt_dxt: ndarray*, *kx: int*, *name: Optional[str] = None*) → None

Set training data (derivatives).

Parameters

xt
[np.ndarray[nt, nx] or np.ndarray[nt]] The input values for the nt training points.

dyt_dxt
[np.ndarray[nt, ny] or np.ndarray[nt]] The derivatives values for the nt training points.

kx
[int] 0-based index of the derivatives being set.

name
[str or None] An optional label for the group of training points being set. This is only used in special situations (e.g., multi-fidelity applications).

`train`() → None

Train the model

`predict_values`(*x: ndarray*) → ndarray

Predict the output values at a set of points.

Parameters

x
[np.ndarray[nt, nx] or np.ndarray[nt]] Input values for the prediction points.

Returns

y
[np.ndarray[nt, ny]] Output values at the prediction points.

`predict_derivatives`(*x: ndarray*, *kx: int*) → ndarray

Predict the dy_dx derivatives at a set of points.

Parameters

x
[np.ndarray[nt, nx] or np.ndarray[nt]] Input values for the prediction points.

kx
[int] The 0-based index of the input variable with respect to which derivatives are desired.

Returns

dy_dx
[np.ndarray[nt, ny]] Derivatives.

`predict_output_derivatives`(*x: ndarray*) → dict

Predict the derivatives dy_dyt at a set of points.

Parameters

x
[np.ndarray[nt, nx] or np.ndarray[nt]] Input values for the prediction points.

Returns

dy_dyt
[dict of np.ndarray[nt, nt]] Dictionary of output derivatives. Key is None for derivatives wrt yt and kx for derivatives wrt dyt_dxt.

predict_variances(*x: ndarray*) → ndarray

Predict the variances at a set of points.

Parameters

x
[np.ndarray[nt, nx] or np.ndarray[nt]] Input values for the prediction points.

Returns

s2
[np.ndarray[nt, ny]] Variances.

3.2.14 How to save and load trained surrogate models

The SurrogateModel API does not contain any save/load interface. Therefore the user has to handle these operations by him/herself. Below some tips to implement save and load.

For models written in pure Python

These operations can be implemented using the pickle module.

Saving the model

```
sm = KRG()
sm.set_training_values(xtrain, ytrain)
sm.train()

filename = "kriging.pkl"
with open(filename, "wb") as f:
   pickle.dump(sm, f)
```

Loading the model

```
sm2 = None
filename = "kriging.pkl"
with open(filename, "rb") as f:
   sm2 = pickle.load(f)
```

For models written in C++ (namely IDW, RBF, RMTB and RMTC)

These models can be cached using their *data_dir* option. Provided the user gives the same training values the model is not retrained but reloaded from cache directory. So by saving the cache directory and the training data, one is able to avoid the training cost and reload the model from cached data.

Saving the model

```
sm = RBF(data_dir="./cache")
sm.set_training_values(xtrain, ytrain)
sm.train()
```

Loading the model

```
sm2 = RBF(data_dir="./cache")
sm2.set_training_values(xtrain, ytrain)   # same training data as above!
sm2.train()                               # actual training is skipped, cached data␣
↪model is loaded
```

3.3 Benchmarking problems

SMT contains a library of analytical and engineering problems to be used for benchmarking purposes. These are listed below.

3.3.1 Sphere function

$$\sum_{i=1}^{nx} x_i^2, \quad -10 \leq x_i \leq 10, \quad \text{for } i = 1, \ldots, nx.$$

Usage

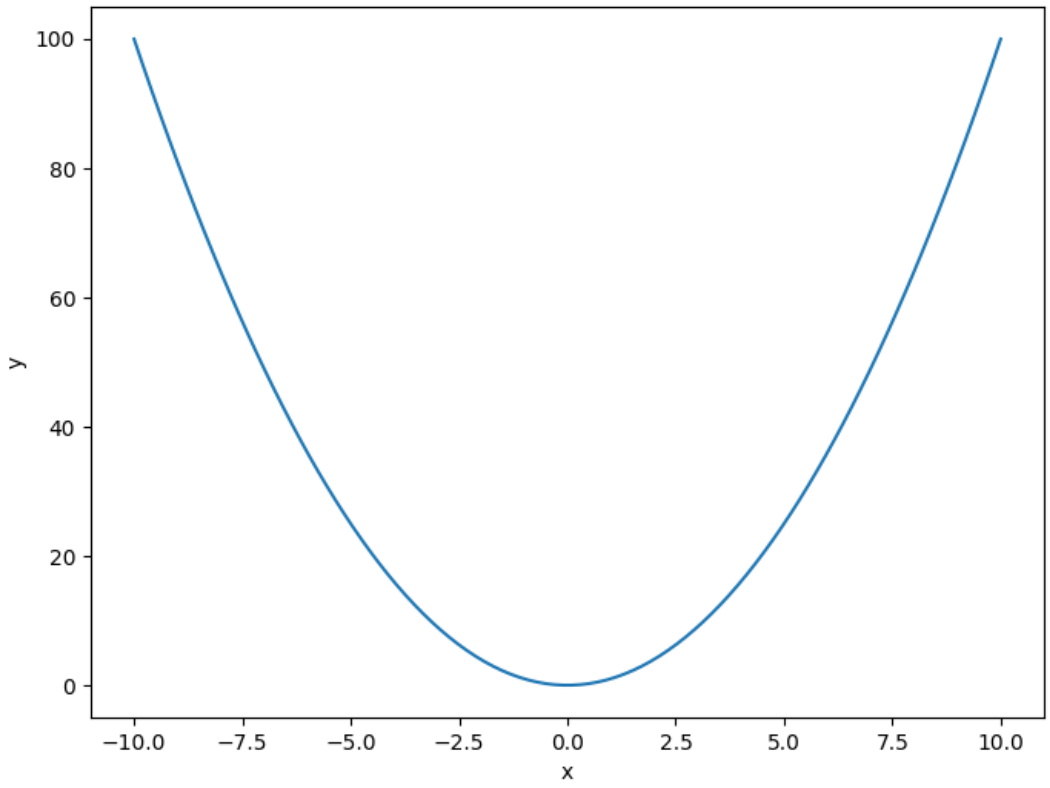
```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import Sphere

ndim = 2
problem = Sphere(ndim=ndim)

num = 100
x = np.ones((num, ndim))
x[:, 0] = np.linspace(-10, 10.0, num)
x[:, 1] = 0.0
y = problem(x)
```

(continues on next page)

(continued from previous page)

```
yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 2)
```



Options

Table 13: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	1	None	['int']	
return_complex	False	None	['bool']	
name	Sphere	None	['str']	

3.3.2 Branin function

$$f(x) = (x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(x_1) + 10,$$

where $x = (x_1, x_2)$ with $-5 \leq x_1 \leq 10, 0 \leq x_2 \leq 15$

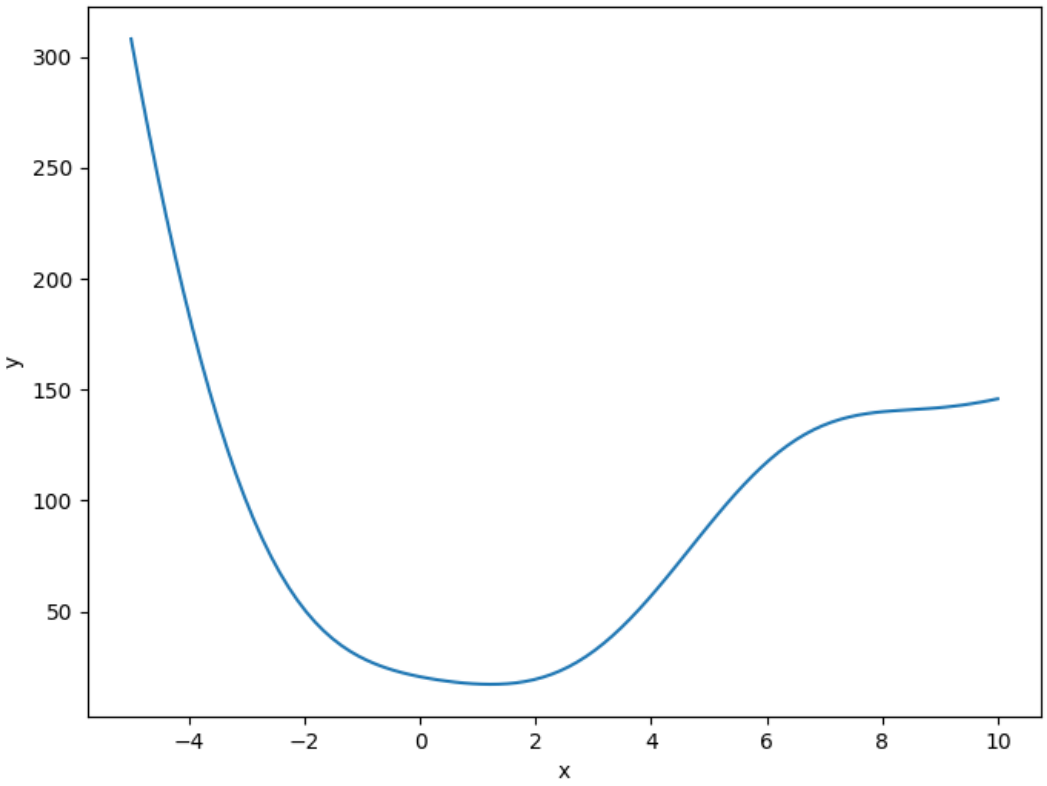
The Branin function has three global minima:

$f(x^*) = 0.397887$, at $x^* = (-\pi, 12.275), (\pi, 2.275)$ and $(9.42478, 2.475)$

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import Branin

ndim = 2
problem = Branin(ndim=ndim)

num = 100
x = np.ones((num, ndim))
x[:, 0] = np.linspace(-5.0, 10.0, num)
x[:, 1] = np.linspace(0.0, 15.0, num)
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 2)
```


300
250
200
150
100
50
y
−4
−2
0
2
4
6
8
10
x

Options

Table 14: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	2	[2]	['int']	
return_complex	False	None	['bool']	
name	Branin	None	['str']	

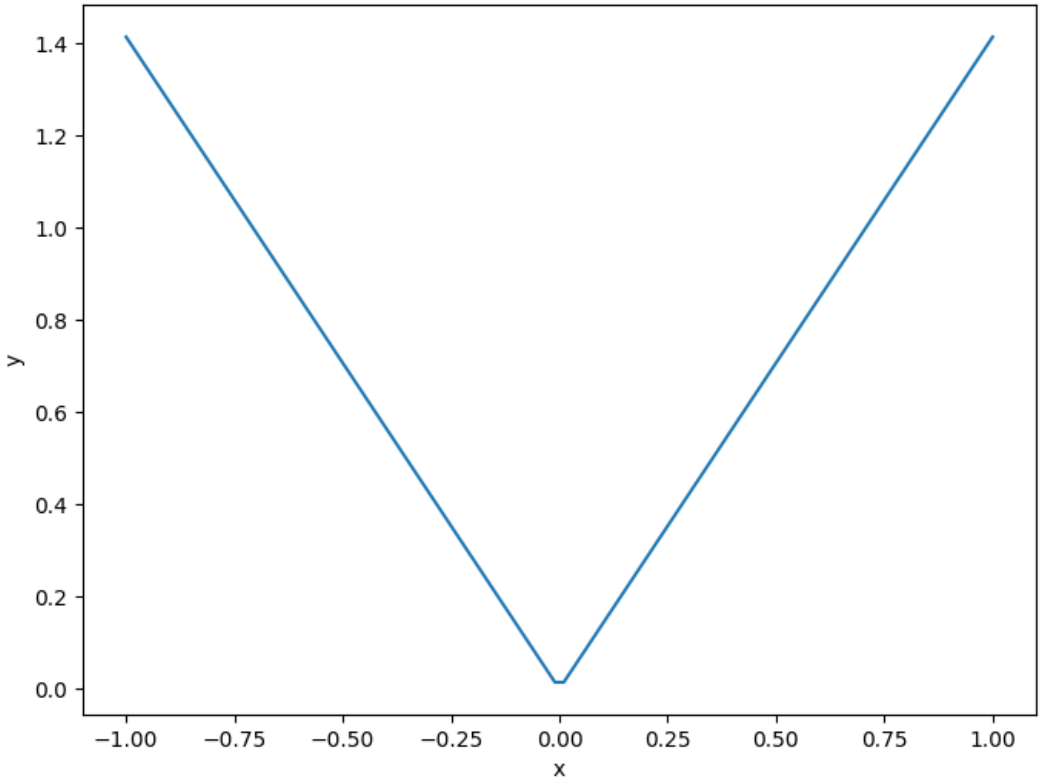
3.3.3 Lp norm function

$$f(x) = \|x\|_p = \sqrt[p]{\sum_i^{nx} |x_i|^p},$$

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import LpNorm

ndim = 2
problem = LpNorm(ndim=ndim, order=2)

num = 100
x = np.ones((num, ndim))
x[:, 0] = np.linspace(-1.0, 1.0, num)
x[:, 1] = np.linspace(-1.0, 1.0, num)
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 2)
```


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

Options

Table 15: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	1	None	['int']	
return_complex	False	None	['bool']	
order	2	None	['int']	
name	LpNorm	None	['str']	

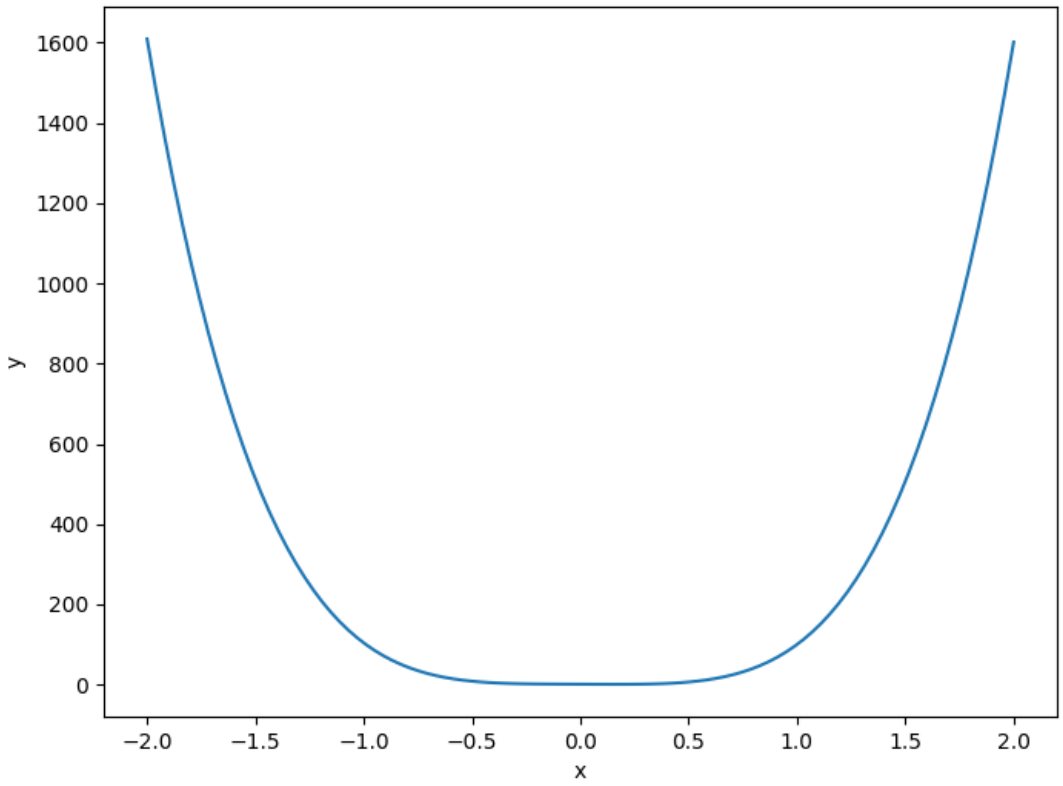
3.3.4 Rosenbrock function

$$\sum_{i=1}^{nx-1} \left[(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right], \quad -2 \leq x_i \leq 2, \quad \text{for } i = 1, \ldots, nx.$$

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import Rosenbrock

ndim = 2
problem = Rosenbrock(ndim=ndim)

num = 100
x = np.ones((num, ndim))
x[:, 0] = np.linspace(-2, 2.0, num)
x[:, 1] = 0.0
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 2)
```


1600
1400
1200
1000
800
600
400
200
0
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

Options

Table 16: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	1	None	['int']	
return_complex	False	None	['bool']	
name	Rosenbrock	None	['str']	

3.3.5 Tensor-product function

cos

$$\prod_{i=1}^{nx} \cos(a\pi x_i), \quad -1 \leq x_i \leq 1, \quad \text{for } i = 1, \ldots, nx.$$

exp

$$\prod_{i=1}^{nx} \exp(x_i), \quad -1 \leq x_i \leq 1, \quad \text{for } i = 1, \ldots, nx.$$

tanh

$$\prod_{i=1}^{nx} \tanh(x_i), \quad -1 \leq x_i \leq 1, \quad \text{for } i = 1, \ldots, nx.$$

gaussian

$$\prod_{i=1}^{nx} \exp(-2x_i^2), \quad -1 \leq x_i \leq 1, \quad \text{for } i = 1, \ldots, nx.$$

Usage

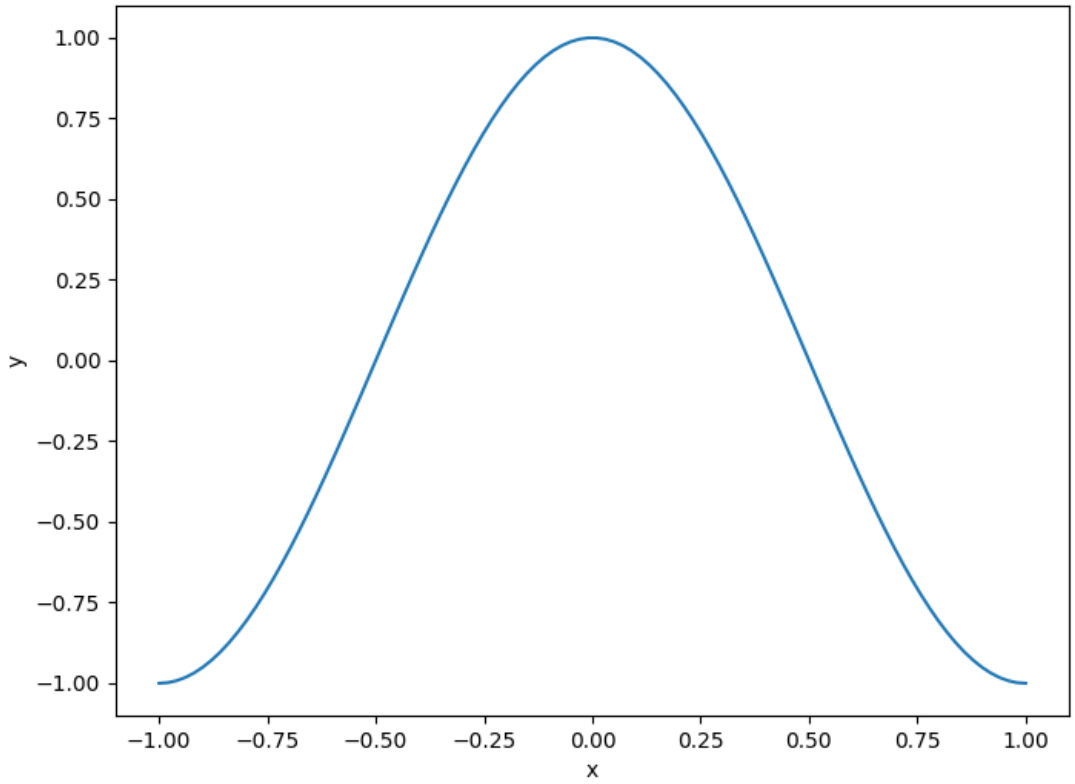
```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import TensorProduct

ndim = 2
problem = TensorProduct(ndim=ndim, func="cos")

num = 100
x = np.ones((num, ndim))
x[:, 0] = np.linspace(-1, 1.0, num)
```

(continues on next page)

(continued from previous page)

```
x[:, 1] = 0.0
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 2)
```



Options

Table 17: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	1	None	['int']	
return_complex	False	None	['bool']	
name	TP	None	['str']	
func	None	['cos', 'exp', 'tanh', 'gaussian']	None	
width	1.0	None	['float', 'int']	

3.3.6 Cantilever beam function

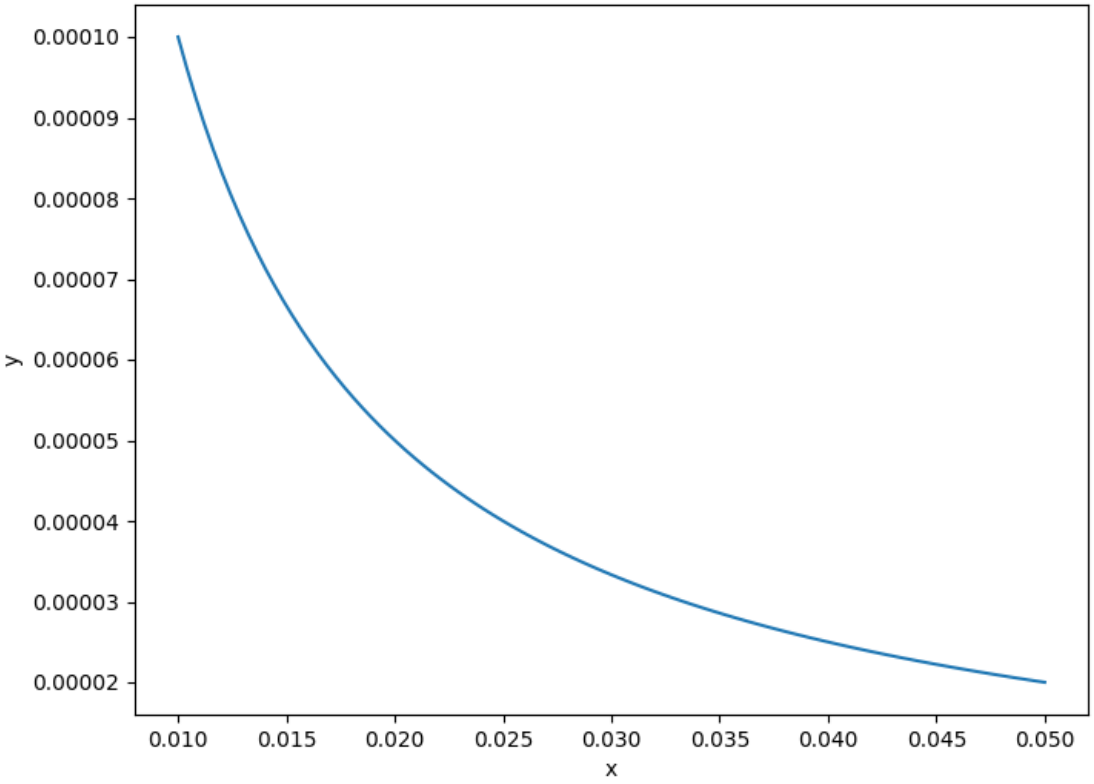
$$\frac{50}{600}\sum_{i=1}^{17}\left[\frac{12}{b_i h_i^3}\left(\left(\sum_{j=i}^{17} l_j\right)^3-\left(\sum_{j=i+1}^{17} l_j\right)^3\right)\right],$$

$b_i \in [0.01, 0.05], \quad h_i \in [0.3, 0.65], \quad l_i \in [0.5, 1].$

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import CantileverBeam

ndim = 3
problem = CantileverBeam(ndim=ndim)

num = 100
x = np.ones((num, ndim))
x[:, 0] = np.linspace(0.01, 0.05, num)
x[:, 1] = 0.5
x[:, 2] = 0.5
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 3)
```



Options

Table 18: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	3	None	[‘int’]	
return_complex	False	None	[‘bool’]	
name	CantileverBeam	None	[‘str’]	
P	50000.0	None	[‘int’, ‘float’]	Tip load (50 kN)
E	200000000000.0	None	[‘int’, ‘float’]	Modulus of elast. (200 GPa)

3.3.7 Robot arm function

$$\sqrt{\left(\sum_{i=1}^{4} L_i \cos\left(\sum_{j=1}^{i} \theta_j\right)\right)^2 + \left(\sum_{i=1}^{4} L_i \sin\left(\sum_{j=1}^{i} \theta_j\right)\right)^2}, \quad L_i \in [0, 1] \quad \text{for} \quad i = 1, \ldots, 4 \quad \text{and} \quad \theta_j \in [0, 2\pi] \quad \text{for} \quad j = 1, \ldots, 4.$$

Usage

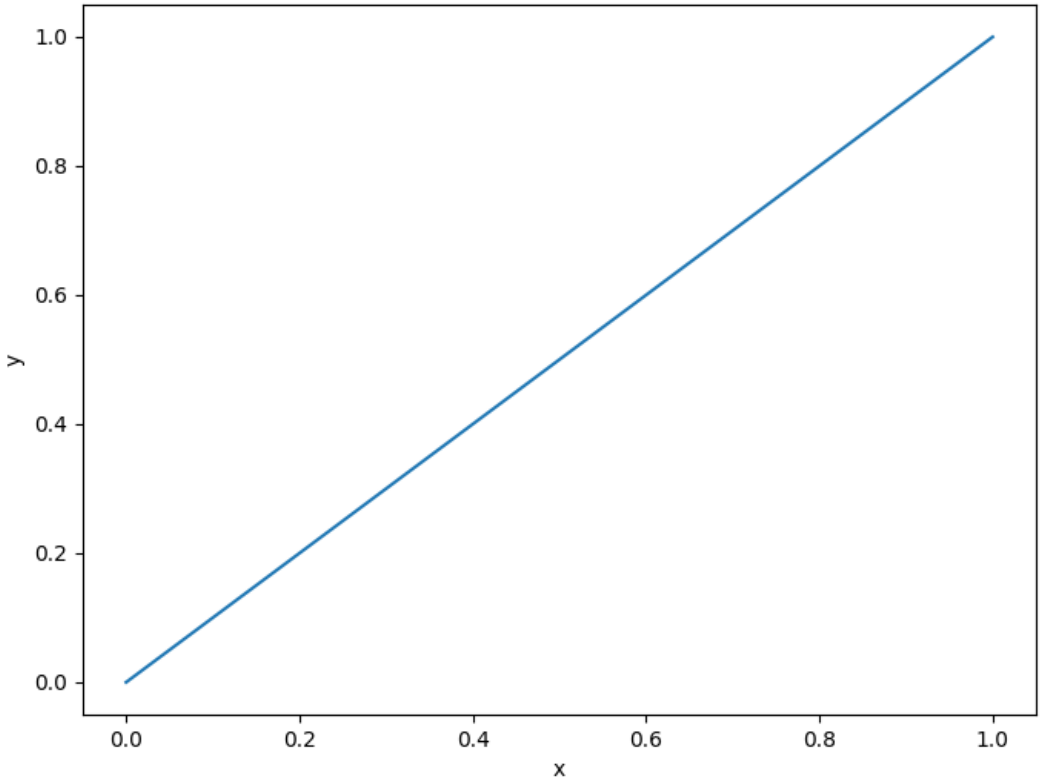
```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import RobotArm

ndim = 2
problem = RobotArm(ndim=ndim)

num = 100
x = np.ones((num, ndim))
x[:, 0] = np.linspace(0.0, 1.0, num)
x[:, 1] = np.pi
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 2)
```

Options

Table 19: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	2	None	['int']	
return_complex	False	None	['bool']	
name	RobotArm	None	['str']	


1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x

3.3.8 Torsion vibration function

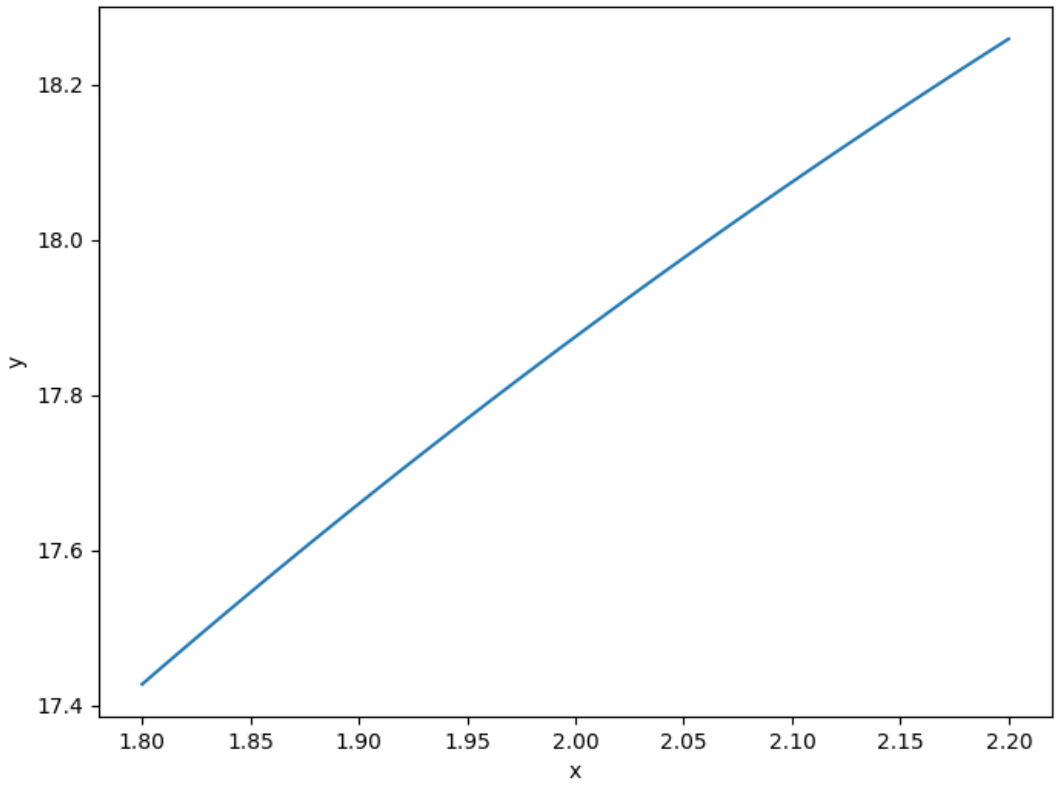
$$\frac{1}{2\pi}\sqrt{\frac{-b-\sqrt{b^2-4ac}}{2a}},$$

where $K_i = \frac{\pi G_i d_i}{32 L_i}$, $M_j = \frac{\rho_j \pi t_j D_j}{4g}$, $J_j = 0.5 M_j \frac{D_j}{2}$, $a = 1$, $b = -\left(\frac{K_1+K2}{J_1} + \frac{K_2+K3}{J_2}\right)$, $c = \frac{K_1K_2+K_2K_3+K_3K_1}{J_1 J_2}$, for $d_1 \in [1.8, 2.2]$, $L_1 \in [9, 11]$, $G_1 \in [105300000, 128700000]$, $d_2 \in [1.638, 2.002]$, $L_2 \in [10.8, 13.2]$, $G_2 \in [5580000, 6820000]$, $d_3 \in [2.025, 2.475]$, $L_3 \in [7.2, 8.8]$, $G_3 \in [3510000, 4290000]$, $D_1 \in [10.8, 13.2]$, $t_1 \in [2.7, 3.3]$, $\rho_1 \in [0.252, 0.308]$, $D_2 \in [12.6, 15.4]$, $t_2 \in [3.6, 4.4]$, $\rho_1 \in [0.09, 0.11]$.

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import TorsionVibration

ndim = 15
problem = TorsionVibration(ndim=ndim)

num = 100
x = np.ones((num, ndim))
for i in range(ndim):
    x[:, i] = 0.5 * (problem.xlimits[i, 0] + problem.xlimits[i, 1])
x[:, 0] = np.linspace(1.8, 2.2, num)
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 15)
```


18.2
18.0
17.8
17.6
17.4
y
1.80
1.85
1.90
1.95
2.00
2.05
2.10
2.15
2.20
x

Options

Table 20: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	1	None	['int']	
return_complex	False	None	['bool']	
name	TorsionVibration	None	['str']	
use_FD	False	None	['bool']	

3.3.9 Water flow function

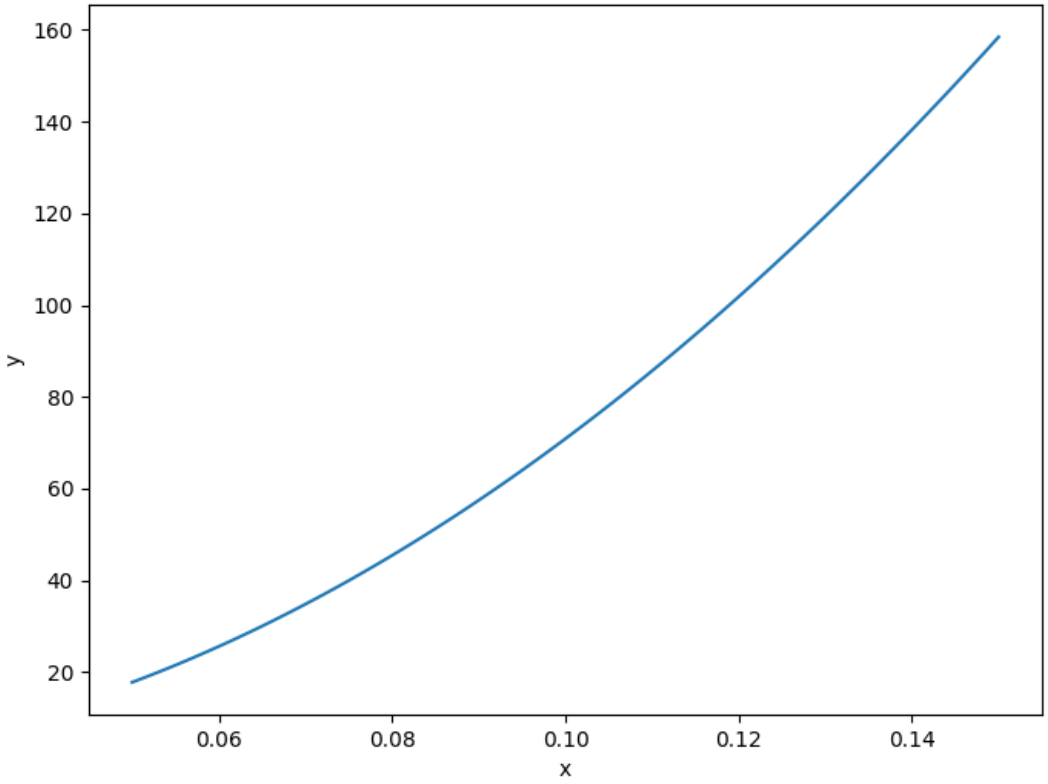
$$\frac{2\pi T_u \left(H_u - H_l\right)}{\ln\left(\frac{r}{r_w}\right)\left[1 + \frac{2LT_u}{\ln\left(\frac{r}{r_w}\right) r_w^2 K_w} + \frac{T_u}{T_l}\right]},$$

$0.05 \leq r_w \leq 0.15$, $100 \leq r \leq 50000$, $63070 \leq T_u \leq 115600$, $990 \leq H_u \leq 1110$, $63.1 \leq T_l \leq 116$, $700 \leq H_l \leq 820$, $1120 \leq L \leq 1680$, and $9855 \leq K_w \leq 12045$.

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import WaterFlow

ndim = 8
problem = WaterFlow(ndim=ndim)

num = 100
x = np.ones((num, ndim))
for i in range(ndim):
    x[:, i] = 0.5 * (problem.xlimits[i, 0] + problem.xlimits[i, 1])
x[:, 0] = np.linspace(0.05, 0.15, num)
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 8)
```



Options

Table 21: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	1	None	['int']	
return_complex	False	None	['bool']	
name	WaterFlow	None	['str']	
use_FD	False	None	['bool']	

3.3.10 Welded beam function

$$\sqrt{\frac{\tau'^2 + \tau''^2 + l\tau'\tau''}{\sqrt{0.25\left(l^2 + (h+t)^2\right)}}},$$

where $\tau' = \frac{6000}{\sqrt{2}hl}$, $\tau'' = \frac{6000(14+0.5l)\sqrt{0.25(l^2+(h+t)^2)}}{2\left[0.707hl\left(\frac{l^2}{12}+0.25(h+t)^2\right)\right]}$, for $h \in [0.125, 1]$, $l, t \in [5, 10]$.

Usage

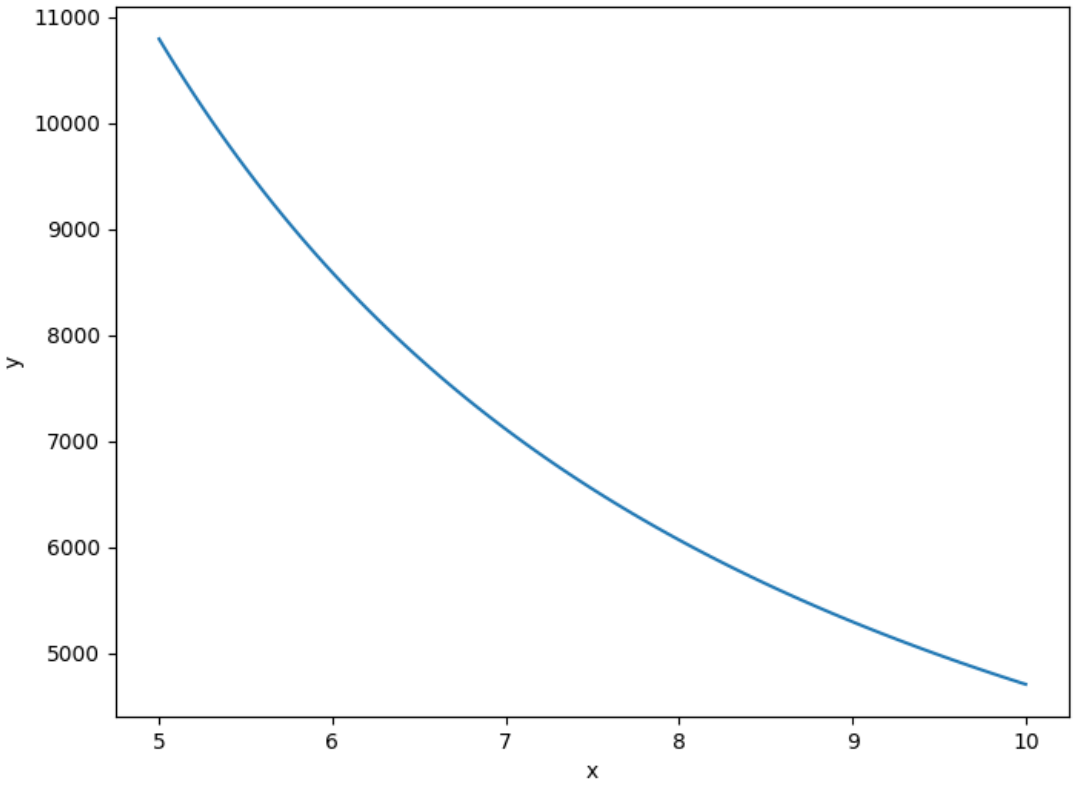
```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import WeldedBeam

ndim = 3
problem = WeldedBeam(ndim=ndim)

num = 100
x = np.ones((num, ndim))
for i in range(ndim):
    x[:, i] = 0.5 * (problem.xlimits[i, 0] + problem.xlimits[i, 1])
x[:, 0] = np.linspace(5.0, 10.0, num)
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 3)
```

Options

Table 22: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	1	None	['int']	
return_complex	False	None	['bool']	
name	WeldedBeam	None	['str']	
use_FD	False	None	['bool']	


11000
10000
9000
8000
7000
6000
5000
y
5
6
7
8
9
10
x

3.3.11 Wing weight function

$$0.036 S_w^{0.758} W_{fw}^{0.0035} \left(\frac{A}{\cos^2 \Lambda}\right)^{0.6} q^{0.006} \lambda^{0.04} \left(\frac{100tc}{\cos \Lambda}\right)^{-0.3} (N_z W_{dg})^{0.49} + S_w W_p,$$

$150 \leq S_w \leq 200$, $220 \leq W_{fw} \leq 300$, $6 \leq A \leq 10$, $-10 \leq \Lambda \leq 10$, $16 \leq q \leq 45$, $0.5 \leq \lambda \leq 1$, $0.08 \leq t_c \leq 0.18$, $2.5 \leq N_z \leq 6$, $1700 \leq W_{dg} \leq 25000$, and $0.025 \leq W_p \leq 0.08$.

Usage

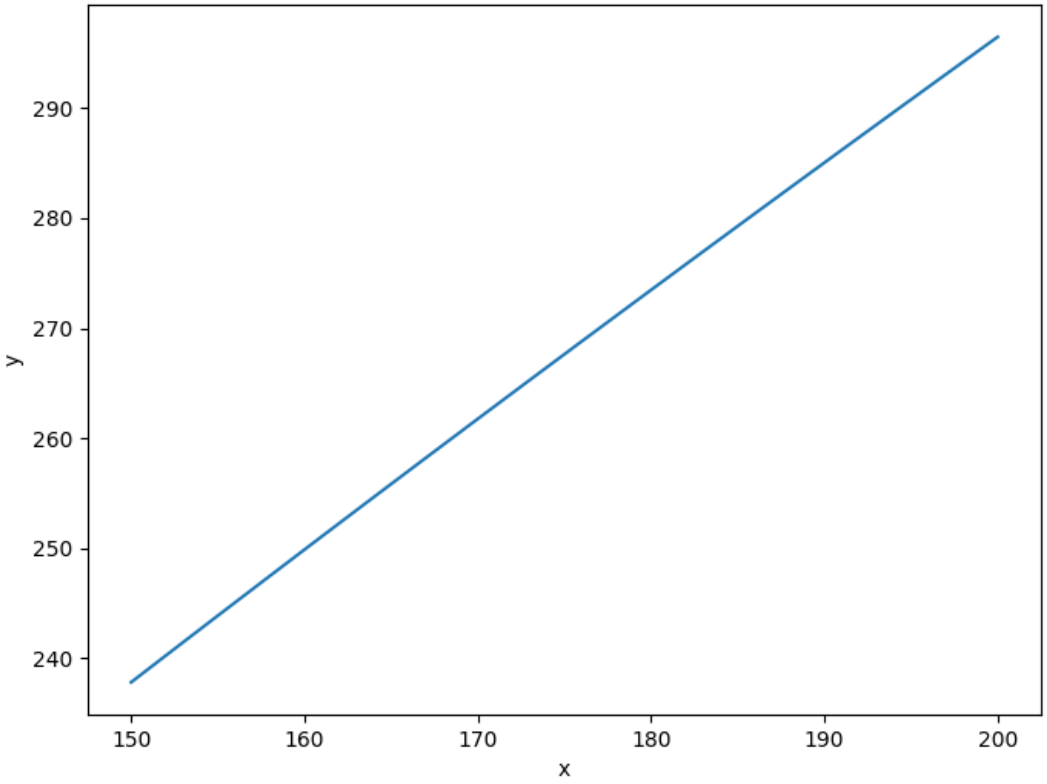
```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import WingWeight

ndim = 10
problem = WingWeight(ndim=ndim)

num = 100
x = np.ones((num, ndim))
for i in range(ndim):
    x[:, i] = 0.5 * (problem.xlimits[i, 0] + problem.xlimits[i, 1])
x[:, 0] = np.linspace(150.0, 200.0, num)
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 10)
```

Options

Table 23: List of options

Option	Default	Acceptable values	Acceptable types	Description
ndim	1	None	['int']	
return_complex	False	None	['bool']	
name	WingWeight	None	['str']	
use_FD	False	None	['bool']	


290
280
270
260
250
240
y
150
160
170
180
190
200
x

3.3.12 Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.problems import Sphere

ndim = 2
problem = Sphere(ndim=ndim)

num = 100
x = np.ones((num, ndim))
x[:, 0] = np.linspace(-10, 10.0, num)
x[:, 1] = 0.0
y = problem(x)

yd = np.empty((num, ndim))
for i in range(ndim):
    yd[:, i] = problem(x, kx=i).flatten()

print(y.shape)
print(yd.shape)

plt.plot(x[:, 0], y[:, 0])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(100, 1)
(100, 2)
```

3.3.13 Problem class API

class smt.problems.problem.**Problem**(***kwargs*)

Methods

__call__(x[, kx])	Evaluate the function.

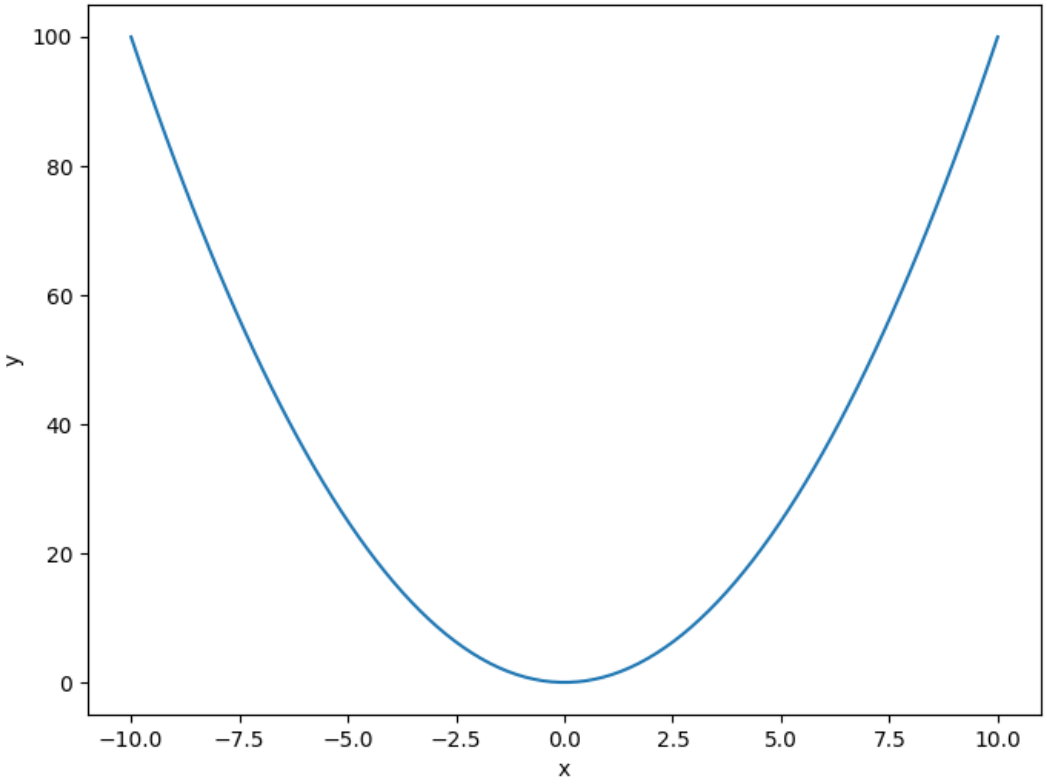
__init__(***kwargs*)

Constructor where values of options can be passed in.

For the list of options, see the documentation for the problem being used.

Parameters

****kwargs**
[named arguments] Set of options that can be optionally set; each option must have been declared.


100
80
60
40
20
0
y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

Examples

```
>>> from smt.problems import Sphere
>>> prob = Sphere(ndim=3)
```

__call__(*x: ndarray*, *kx: Optional[int] = None*) → ndarray

Evaluate the function.

Parameters

x
[ndarray[n, nx] or ndarray[n]] Evaluation points where n is the number of evaluation points.

kx
[int or None] Index of derivative (0-based) to return values with respect to. None means return function value rather than derivative.

Returns

ndarray[n, 1]
Functions values if kx=None or derivative values if kx is an int.

3.4 Sampling methods

SMT contains a library of sampling methods used to generate sets of points in the input space, either for training or for prediction. These are listed below.

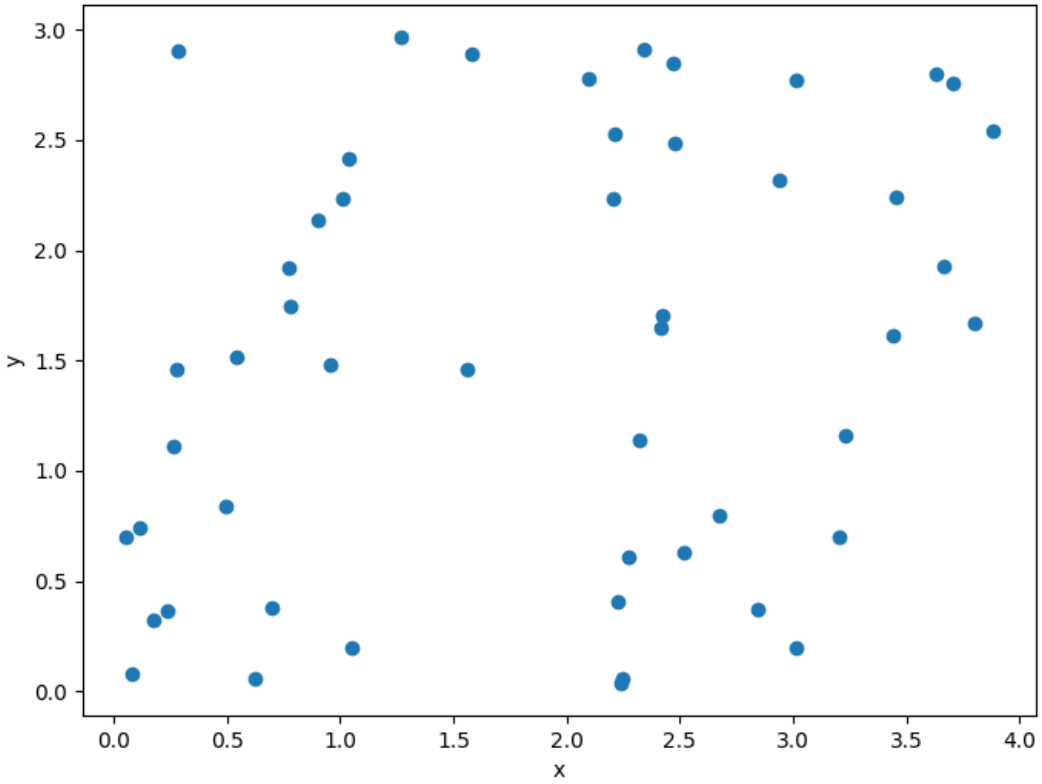
3.4.1 Random sampling

This class creates random samples from a uniform distribution over the design space.

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.sampling_methods import Random

xlimits = np.array([[0.0, 4.0], [0.0, 3.0]])
sampling = Random(xlimits=xlimits)

num = 50
x = sampling(num)

print(x.shape)

plt.plot(x[:, 0], x[:, 1], "o")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(50, 2)
```



Options

Table 24: List of options

Option	Default	Acceptable values	Acceptable types	Description
xlimits	None	None	['ndarray']	The interval of the domain in each dimension with shape nx x 2 (required)

3.4.2 Latin Hypercube sampling

The LHS design is a statistical method for generating a quasi-random sampling distribution. It is among the most popular sampling techniques in computer experiments thanks to its simplicity and projection properties with high-dimensional problems. LHS is built as follows: we cut each dimension space, which represents a variable, into n sections where n is the number of sampling points, and we put only one point in each section.

The LHS method uses the pyDOE package (Design of Experiments for Python)[1]. Five criteria for the construction of LHS are implemented in SMT:

[1] https://pythonhosted.org/pyDOE/index.html

- Center the points within the sampling intervals.
- Maximize the minimum distance between points and place the point in a randomized location within its interval.
- Maximize the minimum distance between points and center the point within its interval.
- Minimize the maximum correlation coefficient.
- Optimize the design using the Enhanced Stochastic Evolutionary algorithm (ESE).

The four first criteria are the same than in pyDOE (for more details, seePage 86, 1). The last criterion, ESE, is implemented by the authors of SMT (more details about such method could be found in[2]).

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.sampling_methods import LHS

xlimits = np.array([[0.0, 4.0], [0.0, 3.0]])
sampling = LHS(xlimits=xlimits)

num = 50
x = sampling(num)

print(x.shape)

plt.plot(x[:, 0], x[:, 1], "o")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

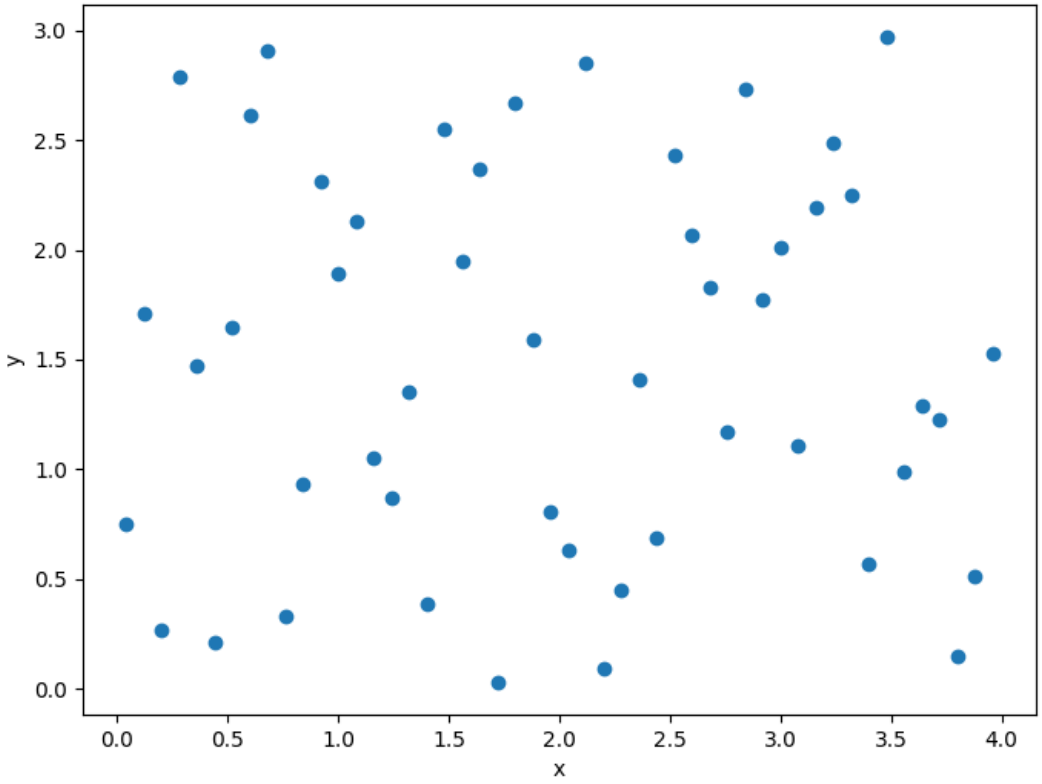
```
(50, 2)
```

Options

Table 25: List of options

Option	Default	Acceptable values	Acceptable types	Description
xlimits	None	None	['ndarray']	The interval of the domain in each dimension with shape nx x 2 (required)
criterion	c	['center', 'maximin', 'centermaximin', 'correlation', 'c', 'm', 'cm', 'corr', 'ese']	['str']	criterion used to construct the LHS design c, m, cm and corr are abbreviation of center, maximin, centermaximin and correlation, respectively
random_state	None	None	['NoneType', 'int', 'RandomState']	Numpy RandomState object or seed number which controls random draws

[2] Jin, R. and Chen, W. and Sudjianto, A. (2005), "An efficient algorithm for constructing optimal design of computer experiments." Journal of Statistical Planning and Inference, 134:268-287.

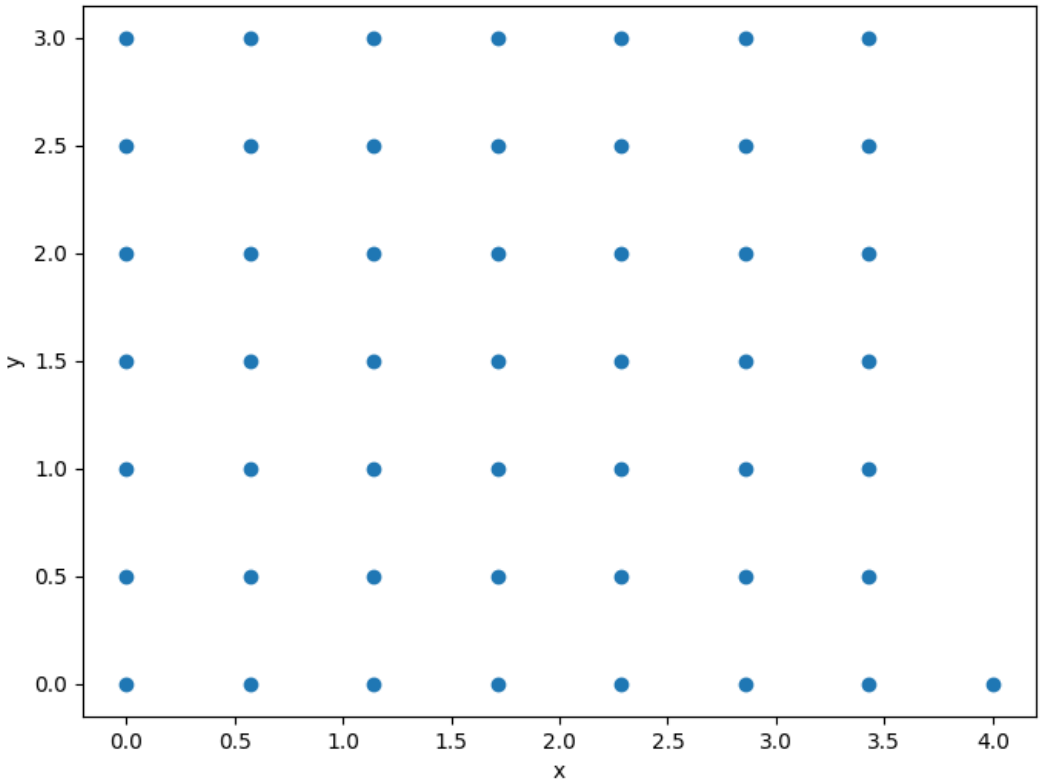

3.0
2.5
2.0
1.5
1.0
0.5
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

3.4.3 Full-factorial sampling

Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.sampling_methods import FullFactorial

xlimits = np.array([[0.0, 4.0], [0.0, 3.0]])
sampling = FullFactorial(xlimits=xlimits)

num = 50
x = sampling(num)

print(x.shape)

plt.plot(x[:, 0], x[:, 1], "o")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(50, 2)
```



Options

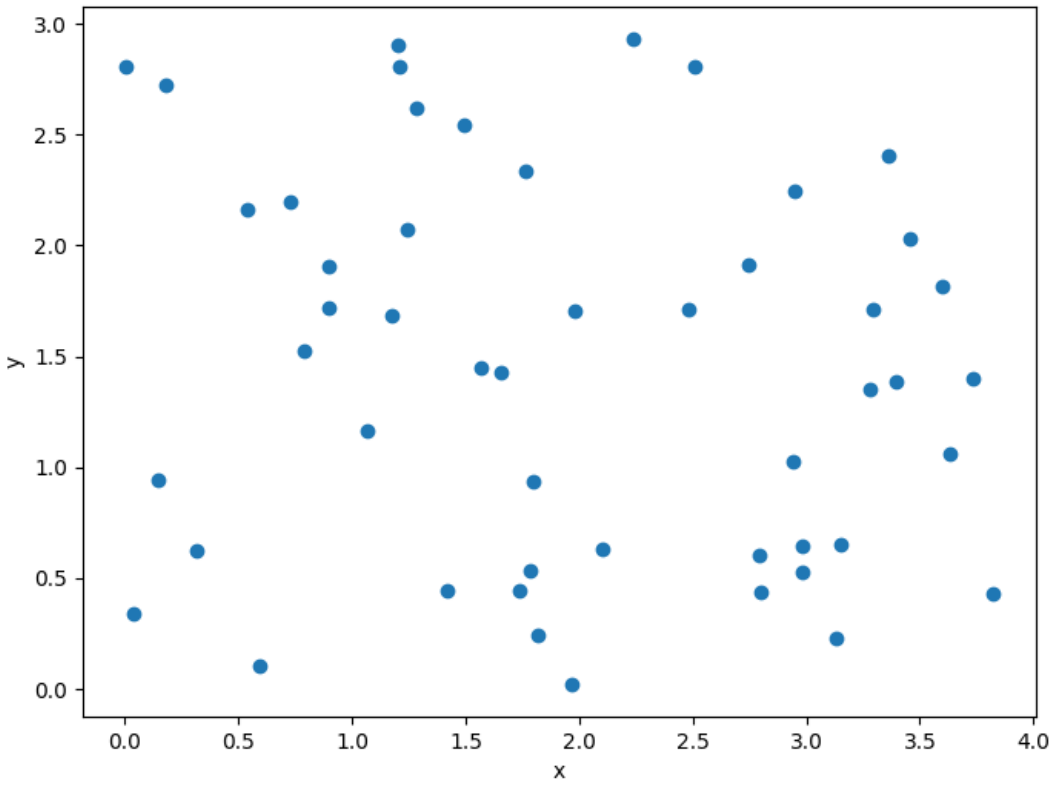
Table 26: List of options

Option	Default	Acceptable values	Acceptable types	Description
xlimits	None	None	['ndarray']	The interval of the domain in each dimension with shape nx x 2 (required)
weights	None	None	['list', 'ndarray']	relative sampling weights for each nx dimensions
clip	False	None	['bool']	round number of samples to the sampling number product of each nx dimensions (> asked nt)

3.4.4 Usage

```
import numpy as np
import matplotlib.pyplot as plt

from smt.sampling_methods import Random

xlimits = np.array([[0.0, 4.0], [0.0, 3.0]])
sampling = Random(xlimits=xlimits)

num = 50
x = sampling(num)

print(x.shape)

plt.plot(x[:, 0], x[:, 1], "o")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(50, 2)
```


3.0
2.5
2.0
1.5
1.0
0.5
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

3.4.5 Problem class API

class smt.sampling_methods.sampling_method.**SamplingMethod**(***kwargs*)

Methods

__call__(nt)	Compute the requested number of sampling points.

__init__(***kwargs*)

Constructor where values of options can be passed in.

For the list of options, see the documentation for the problem being used.

Parameters

****kwargs**
[named arguments] Set of options that can be optionally set; each option must have been declared.

Examples

```
>>> import numpy as np
>>> from smt.sampling_methods import Random
>>> sampling = Random(xlimits=np.arange(2).reshape((1, 2)))
```

__call__(*nt: int*) → ndarray

Compute the requested number of sampling points.

The number of dimensions (nx) is determined based on *xlimits.shape[0]*.

Returns

ndarray[nt, nx]
The sampling locations in the input space.

3.5 Examples

Below is a set of examples from practical use cases of SMT. The data set, run script, and plot are shown. The intent is to provide examples to help guide the choice of options for the various surrogate modeling methods.

3.5.1 1-D step-like data set

```
import numpy as np


def get_one_d_step():
    xt = np.array(
        [
            0.0000,
            0.4000,
```

(continues on next page)

(continued from previous page)

```
            0.6000,
            0.7000,
            0.7500,
            0.7750,
            0.8000,
            0.8500,
            0.8750,
            0.9000,
            0.9250,
            0.9500,
            0.9750,
            1.0000,
            1.0250,
            1.0500,
            1.1000,
            1.2000,
            1.3000,
            1.4000,
            1.6000,
            1.8000,
            2.0000,
        ],
        dtype=np.float64,
    )
    yt = np.array(
        [
            0.0130,
            0.0130,
            0.0130,
            0.0130,
            0.0130,
            0.0130,
            0.0130,
            0.0132,
            0.0135,
            0.0140,
            0.0162,
            0.0230,
            0.0275,
            0.0310,
            0.0344,
            0.0366,
            0.0396,
            0.0410,
            0.0403,
            0.0390,
            0.0360,
            0.0350,
            0.0345,
        ],
        dtype=np.float64,
    )
```

(continues on next page)

(continued from previous page)

```
    xlimits = np.array([[0.0, 2.0]])

    return xt, yt, xlimits


def plot_one_d_step(xt, yt, limits, interp):
    import numpy as np
    import matplotlib

    matplotlib.use("Agg")
    import matplotlib.pyplot as plt

    num = 500
    x = np.linspace(0.0, 2.0, num)
    y = interp.predict_values(x)[:, 0]

    plt.plot(x, y)
    plt.plot(xt, yt, "o")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()
```

RMTB

```
from smt.surrogate_models import RMTB
from smt.examples.one_D_step.one_D_step import get_one_d_step, plot_one_d_step

xt, yt, xlimits = get_one_d_step()

interp = RMTB(
    num_ctrl_pts=100,
    xlimits=xlimits,
    nonlinear_maxiter=20,
    solver_tolerance=1e-16,
    energy_weight=1e-14,
    regularization_weight=0.0,
)
interp.set_training_values(xt, yt)
interp.train()

plot_one_d_step(xt, yt, xlimits, interp)
```

```
___________________________________________________________________________

                                   RMTB
___________________________________________________________________________

 Problem size
```

(continues on next page)

(continued from previous page)

```
      # training points.        : 23

___________________________________________________________________________

 Training

   Training ...
      Pre-computing matrices ...
         Computing dof2coeff ...
         Computing dof2coeff - done. Time (sec):  0.0000000
         Initializing Hessian ...
         Initializing Hessian - done. Time (sec):  0.0000000
         Computing energy terms ...
         Computing energy terms - done. Time (sec):  0.0000000
         Computing approximation terms ...
         Computing approximation terms - done. Time (sec):  0.0000000
      Pre-computing matrices - done. Time (sec):  0.0000000
      Solving for degrees of freedom ...
         Solving initial startup problem (n=100) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 1.032652876e-01 8.
↪436300000e-03
               Iteration (num., iy, grad. norm, func.) :   0   0 8.326567782e-09 2.
↪218506146e-13
            Solving for output 0 - done. Time (sec):  0.0070736
         Solving initial startup problem (n=100) - done. Time (sec):  0.0070736
         Solving nonlinear problem (n=100) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 1.550397729e-11 2.
↪217742297e-13
               Iteration (num., iy, grad. norm, func.) :   0   0 1.400133688e-11 2.
↪190130776e-13
               Iteration (num., iy, grad. norm, func.) :   1   0 4.569917417e-10 1.
↪398657411e-13
               Iteration (num., iy, grad. norm, func.) :   2   0 3.273418041e-10 9.
↪582645418e-14
               Iteration (num., iy, grad. norm, func.) :   3   0 9.631253690e-11 2.
↪487556028e-14
               Iteration (num., iy, grad. norm, func.) :   4   0 2.807786097e-11 1.
↪154934069e-14
               Iteration (num., iy, grad. norm, func.) :   5   0 1.047652622e-11 9.
↪424623014e-15
               Iteration (num., iy, grad. norm, func.) :   6   0 2.796406609e-12 8.
↪629430946e-15
               Iteration (num., iy, grad. norm, func.) :   7   0 2.503249902e-12 8.
↪611872251e-15
               Iteration (num., iy, grad. norm, func.) :   8   0 1.673758712e-12 8.
↪544715841e-15
               Iteration (num., iy, grad. norm, func.) :   9   0 4.321920620e-13 8.
↪467209450e-15
               Iteration (num., iy, grad. norm, func.) :  10   0 1.206983452e-13 8.
↪455862293e-15
```

(continues on next page)

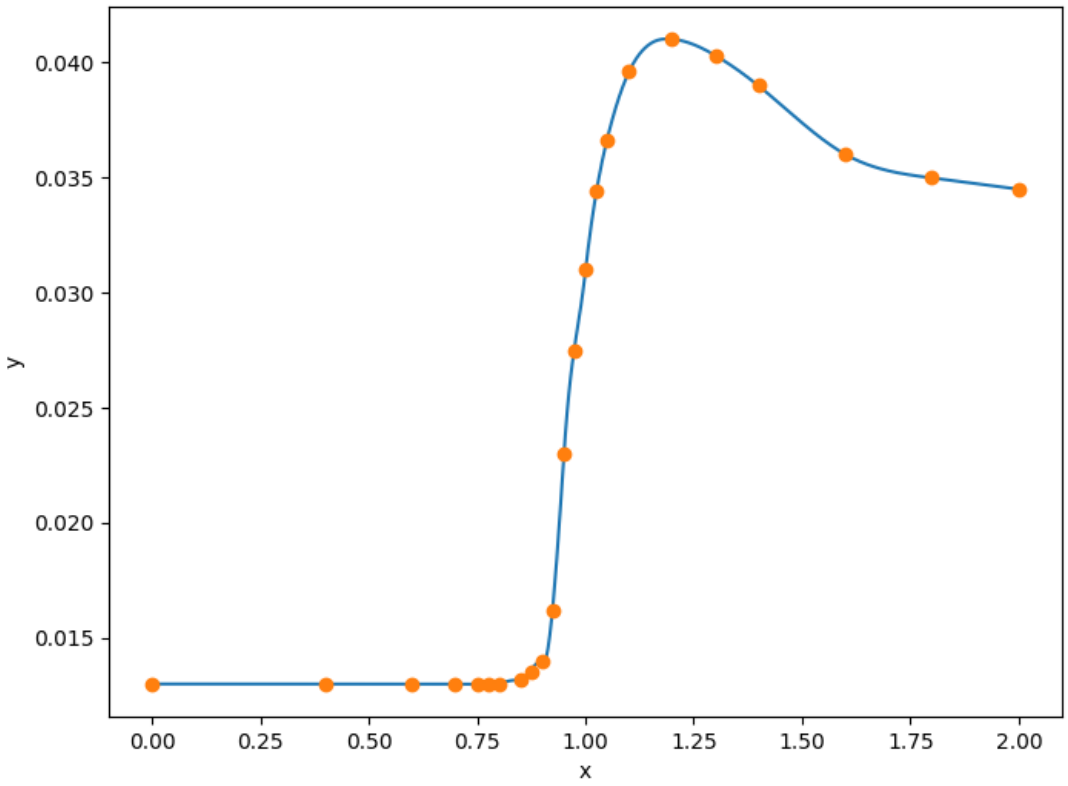
(continued from previous page)

```
              Iteration (num., iy, grad. norm, func.) :  11   0 3.366638988e-14 8.
↪453930122e-15
              Iteration (num., iy, grad. norm, func.) :  12   0 1.432594106e-14 8.
↪453696373e-15
              Iteration (num., iy, grad. norm, func.) :  13   0 1.419395614e-14 8.
↪453592635e-15
              Iteration (num., iy, grad. norm, func.) :  14   0 3.778253812e-15 8.
↪453316574e-15
              Iteration (num., iy, grad. norm, func.) :  15   0 1.065786022e-15 8.
↪453276042e-15
              Iteration (num., iy, grad. norm, func.) :  16   0 2.072128988e-15 8.
↪453275135e-15
              Iteration (num., iy, grad. norm, func.) :  17   0 1.842351695e-16 8.
↪453270514e-15
              Iteration (num., iy, grad. norm, func.) :  18   0 1.015886357e-16 8.
↪453270452e-15
              Iteration (num., iy, grad. norm, func.) :  19   0 1.015887329e-16 8.
↪453270452e-15
           Solving for output 0 - done. Time (sec):  0.1002092
        Solving nonlinear problem (n=100) - done. Time (sec):  0.1002092
     Solving for degrees of freedom - done. Time (sec):  0.1072829
  Training - done. Time (sec):  0.1072829
___________________________________________________________________________

 Evaluation

      # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000
```

RMTC

```
from smt.surrogate_models import RMTC
from smt.examples.one_D_step.one_D_step import get_one_d_step, plot_one_d_step

xt, yt, xlimits = get_one_d_step()

interp = RMTC(
    num_elements=40,
    xlimits=xlimits,
    nonlinear_maxiter=20,
    solver_tolerance=1e-16,
    energy_weight=1e-14,
    regularization_weight=0.0,
)
interp.set_training_values(xt, yt)
interp.train()
```

(continues on next page)


0.040
0.035
0.030
0.025
0.020
0.015
y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
x

(continued from previous page)

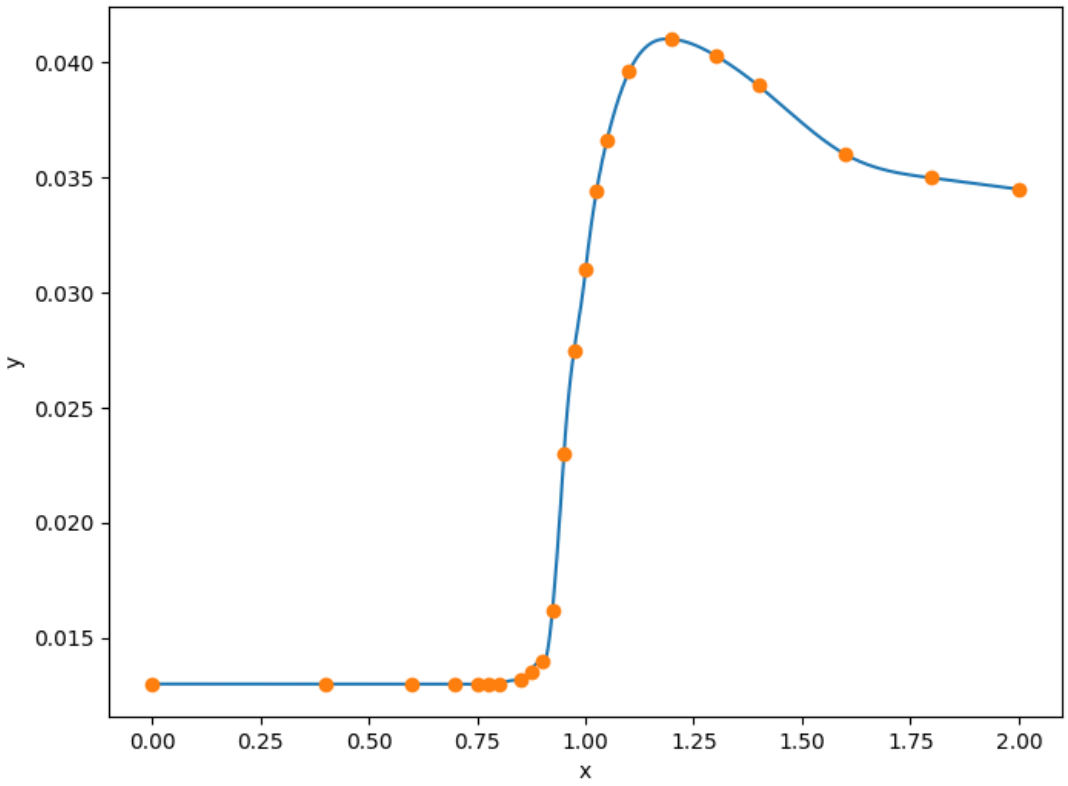
```
plot_one_d_step(xt, yt, xlimits, interp)
```

```
___________________________________________________________________________

                                   RMTC
___________________________________________________________________________

 Problem size

      # training points.        : 23

___________________________________________________________________________

 Training

   Training ...
      Pre-computing matrices ...
         Computing dof2coeff ...
         Computing dof2coeff - done. Time (sec):  0.0000000
         Initializing Hessian ...
         Initializing Hessian - done. Time (sec):  0.0000000
         Computing energy terms ...
         Computing energy terms - done. Time (sec):  0.0000000
         Computing approximation terms ...
         Computing approximation terms - done. Time (sec):  0.0000000
      Pre-computing matrices - done. Time (sec):  0.0000000
      Solving for degrees of freedom ...
         Solving initial startup problem (n=82) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 1.470849329e-01 8.
↪436300000e-03
               Iteration (num., iy, grad. norm, func.) :   0   0 1.271524727e-11 2.
↪493686417e-14
            Solving for output 0 - done. Time (sec):  0.0080578
         Solving initial startup problem (n=82) - done. Time (sec):  0.0080578
         Solving nonlinear problem (n=82) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 7.484146458e-12 2.
↪493686273e-14
               Iteration (num., iy, grad. norm, func.) :   0   0 9.032463140e-12 2.
↪483319826e-14
               Iteration (num., iy, grad. norm, func.) :   1   0 8.723372989e-11 2.
↪393675636e-14
               Iteration (num., iy, grad. norm, func.) :   2   0 4.783883236e-11 1.
↪793850937e-14
               Iteration (num., iy, grad. norm, func.) :   3   0 4.678916694e-11 1.
↪785317983e-14
               Iteration (num., iy, grad. norm, func.) :   4   0 1.297955451e-11 1.
↪193038054e-14
               Iteration (num., iy, grad. norm, func.) :   5   0 3.942464065e-12 1.
↪121509131e-14
```

(continues on next page)

(continued from previous page)

```
               Iteration (num., iy, grad. norm, func.) :   6   0 8.384726431e-13 1.
↪110564189e-14
               Iteration (num., iy, grad. norm, func.) :   7   0 2.581741267e-13 1.
↪109374227e-14
               Iteration (num., iy, grad. norm, func.) :   8   0 7.635918060e-14 1.
↪109026987e-14
               Iteration (num., iy, grad. norm, func.) :   9   0 2.106298788e-14 1.
↪108953137e-14
               Iteration (num., iy, grad. norm, func.) :  10   0 5.042586986e-15 1.
↪108941658e-14
               Iteration (num., iy, grad. norm, func.) :  11   0 8.730387249e-16 1.
↪108940427e-14
               Iteration (num., iy, grad. norm, func.) :  12   0 1.188005043e-16 1.
↪108940347e-14
               Iteration (num., iy, grad. norm, func.) :  13   0 2.828378041e-16 1.
↪108940346e-14
               Iteration (num., iy, grad. norm, func.) :  14   0 2.828383946e-16 1.
↪108940346e-14
               Iteration (num., iy, grad. norm, func.) :  15   0 2.828383946e-16 1.
↪108940346e-14
               Iteration (num., iy, grad. norm, func.) :  16   0 2.828383946e-16 1.
↪108940346e-14
               Iteration (num., iy, grad. norm, func.) :  17   0 2.828383946e-16 1.
↪108940346e-14
               Iteration (num., iy, grad. norm, func.) :  18   0 2.828383946e-16 1.
↪108940346e-14
               Iteration (num., iy, grad. norm, func.) :  19   0 2.828383946e-16 1.
↪108940346e-14
            Solving for output 0 - done. Time (sec):  0.0748911
         Solving nonlinear problem (n=82) - done. Time (sec):  0.0748911
      Solving for degrees of freedom - done. Time (sec):  0.0829489
   Training - done. Time (sec):  0.0829489
___________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```


0.040
0.035
0.030
0.025
0.020
0.015
y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
x

3.5.2 RANS CRM wing 2-D data set

```
import numpy as np


raw = np.array(
    [
        [
            2.00000000000000000e00,
            4.50000000000000111e-01,
            1.53679999999999972e-02,
            3.67423999999999728e-01,
            5.59227999999999474e-01,
            -1.25803999999999992e-01,
            -1.24869999999999984e-02,
        ],
        [
            3.50000000000000000e00,
            4.50000000000000111e-01,
            1.98510000000000059e-02,
            4.90447000000000218e-01,
            7.57460000000000222e-01,
            -1.61526000000000029e-01,
            8.98700000000000197e-03,
        ],
        [
            5.00000000000000000e00,
            4.50000000000000111e-01,
            2.57100000000000021e-02,
            6.10918999999999898e-01,
            9.49794999999999449e-01,
            -1.95461999999999969e-01,
            4.09090000000000092e-02,
        ],
        [
            6.50000000000000000e00,
            4.50000000000000111e-01,
            3.30420000000000192e-02,
            7.26612000000000356e-01,
            1.13113899999999895e00,
            -2.25589000000000117e-01,
            8.18539999999999621e-02,
        ],
        [
            8.00000000000000000e00,
            4.50000000000000111e-01,
            4.31899999999999923e-02,
            8.24725000000000414e-01,
            1.27148700000000034e00,
            -2.39704000000000004e-01,
            1.21765999999999992e-01,
        ],
        [
```

(continues on next page)

(continued from previous page)

```
            0.00000000000000000e00,
            5.79999999999999600e-01,
            1.13620000000000057e-02,
            2.04876000000000026e-01,
            2.95028000000000125e-01,
            -7.88210000000000217e-02,
            -2.28009999999999835e-02,
        ],
        [
            1.50000000000000000e00,
            5.79999999999999600e-01,
            1.42600000000000011e-02,
            3.37561999999999732e-01,
            5.11413000000000065e-01,
            -1.18942000000000061e-01,
            -1.58820000000000028e-02,
        ],
        [
            3.00000000000000000e00,
            5.79999999999999600e-01,
            1.86640000000000003e-02,
            4.68745000000000228e-01,
            7.24040000000000169e-01,
            -1.57766999999999906e-01,
            3.09999999999999891e-03,
        ],
        [
            4.50000000000000000e00,
            5.79999999999999600e-01,
            2.46199999999999952e-02,
            5.97663999999999731e-01,
            9.31170999999999710e-01,
            -1.94416000000000055e-01,
            3.35750000000000068e-02,
        ],
        [
            6.00000000000000000e00,
            5.79999999999999600e-01,
            3.28070000000000283e-02,
            7.14224999999999988e-01,
            1.11170799999999918e00,
            -2.20587000000000053e-01,
            7.15169999999999724e-02,
        ],
        [
            0.00000000000000000e00,
            6.80000000000000488e-01,
            1.13880000000000055e-02,
            2.09931000000000065e-01,
            3.03223000000000203e-01,
            -8.18789999999999345e-02,
            -2.17269999999999979e-02,
```

(continues on next page)

(continued from previous page)

```
        ],
        [
            1.5000000000000000e00,
            6.8000000000000488e-01,
            1.4586999999999927e-02,
            3.5185699999999753e-01,
            5.3566300000000003e-01,
            -1.2576499999999879e-01,
            -1.4448000000000077e-02,
        ],
        [
            3.0000000000000000e00,
            6.8000000000000488e-01,
            1.9528000000000022e-02,
            4.9248799999999813e-01,
            7.6447699999999621e-01,
            -1.6780400000000087e-01,
            6.0239999999999841e-03,
        ],
        [
            4.5000000000000000e00,
            6.8000000000000488e-01,
            2.6666999999999973e-02,
            6.2703399999999803e-01,
            9.8016300000000065e-01,
            -2.0352400000000105e-01,
            3.8100000000000192e-02,
        ],
        [
            6.0000000000000000e00,
            6.8000000000000488e-01,
            3.8918000000000120e-02,
            7.1727300000000494e-01,
            1.0978559999999943e00,
            -2.0146200000000022e-01,
            6.6400000000000069e-02,
        ],
        [
            0.0000000000000000e00,
            7.5000000000000000e-01,
            1.1506999999999987e-02,
            2.1490699999999869e-01,
            3.1157400000000176e-01,
            -8.4989999999999611e-02,
            -2.0577000000000154e-02,
        ],
        [
            1.2500000000000000e00,
            7.5000000000000000e-01,
            1.4326000000000019e-02,
            3.4159699999999840e-01,
            5.1993900000000400e-01,
```

(continues on next page)

(continued from previous page)

```
            -1.251009999999999900e-01,
            -1.515400000000000080e-02,
        ],
        [
            2.500000000000000000e00,
            7.500000000000000000e-01,
            1.856000000000000011e-02,
            4.677589999999999804e-01,
            7.262499999999999512e-01,
            -1.635169999999999957e-01,
            3.989999999999999949e-04,
        ],
        [
            3.750000000000000000e00,
            7.500000000000000000e-01,
            2.472399999999999945e-02,
            5.911459999999999493e-01,
            9.254930000000000101e-01,
            -1.966150000000000120e-01,
            2.524900000000000061e-02,
        ],
        [
            5.000000000000000000e00,
            7.500000000000000000e-01,
            3.506800000000000195e-02,
            7.047809999999999908e-01,
            1.097736000000000045e00,
            -2.143069999999999975e-01,
            5.321300000000000335e-02,
        ],
        [
            0.000000000000000000e00,
            8.000000000000000444e-01,
            1.168499999999999921e-02,
            2.196390000000000009e-01,
            3.197160000000000002e-01,
            -8.798200000000000465e-02,
            -1.926999999999999894e-02,
        ],
        [
            1.250000000000000000e00,
            8.000000000000000444e-01,
            1.481599999999999931e-02,
            3.553939999999999877e-01,
            5.435950000000000504e-01,
            -1.317419999999999980e-01,
            -1.345599999999999921e-02,
        ],
        [
            2.500000000000000000e00,
            8.000000000000000444e-01,
            1.968999999999999917e-02,
```

(continues on next page)

(continued from previous page)

```
            4.918299999999999894e-01,
            7.669930000000000359e-01,
            -1.728079999999999894e-01,
            3.756999999999999923e-03,
        ],
        [
            3.750000000000000000e00,
            8.000000000000000444e-01,
            2.785599999999999882e-02,
            6.324319999999999942e-01,
            9.919249999999999456e-01,
            -2.077100000000000057e-01,
            3.159800000000000109e-02,
        ],
        [
            5.000000000000000000e00,
            8.000000000000000444e-01,
            4.394300000000000289e-02,
            7.650689999999999991e-01,
            1.188355999999999968e00,
            -2.332680000000000031e-01,
            5.645000000000000018e-02,
        ],
        [
            0.000000000000000000e00,
            8.299999999999999600e-01,
            1.186100000000000002e-02,
            2.232899999999999885e-01,
            3.261100000000000110e-01,
            -9.028400000000000314e-02,
            -1.806500000000000120e-02,
        ],
        [
            1.000000000000000000e00,
            8.299999999999999600e-01,
            1.444900000000000004e-02,
            3.383419999999999761e-01,
            5.161710000000000464e-01,
            -1.279530000000000112e-01,
            -1.402400000000000001e-02,
        ],
        [
            2.000000000000000000e00,
            8.299999999999999600e-01,
            1.836799999999999891e-02,
            4.554270000000000262e-01,
            7.082190000000000429e-01,
            -1.642339999999999911e-01,
            -1.793000000000000106e-03,
        ],
        [
            3.000000000000000000e00,
```

(continues on next page)

(continued from previous page)

```
            8.29999999999999600e-01,
            2.46689999999999996e-02,
            5.79841000000000508e-01,
            9.08881999999999677e-01,
            -2.00458999999999983e-01,
            1.89290000000000138e-02,
        ],
        [
            4.00000000000000000e00,
            8.29999999999999600e-01,
            3.70040000000000217e-02,
            7.01272000000000065e-01,
            1.09736600000000064e00,
            -2.36242000000000075e-01,
            3.75069999999999867e-02,
        ],
        [
            0.00000000000000000e00,
            8.59999999999999867e-01,
            1.22430000000000041e-02,
            2.27810000000000125e-01,
            3.34272000000000136e-01,
            -9.30760000000000595e-02,
            -1.60840000000000107e-02,
        ],
        [
            1.00000000000000000e00,
            8.59999999999999867e-01,
            1.54070000000000056e-02,
            3.55183999999999997e-01,
            5.43313000000000459e-01,
            -1.36473000000000110e-01,
            -1.16220000000000039e-02,
        ],
        [
            2.00000000000000000e00,
            8.59999999999999867e-01,
            2.12269999999999934e-02,
            4.85462000000000046e-01,
            7.55291999999999634e-01,
            -1.81785000000000021e-01,
            1.07099999999999903e-03,
        ],
        [
            3.00000000000000000e00,
            8.59999999999999867e-01,
            3.17889999999999781e-02,
            6.08184999999999756e-01,
            9.51038000000000500e-01,
            -2.25202000000000133e-01,
            1.54079999999999982e-02,
        ],
```

(continues on next page)

(continued from previous page)

```
        [
            4.000000000000000000e00,
            8.599999999999999867e-01,
            4.744199999999999806e-02,
            6.846989999999999466e-01,
            1.042564000000000046e00,
            -2.333600000000000119e-01,
            2.035400000000000056e-02,
        ],
    ]
)


def get_rans_crm_wing():
    # data structure:
    # alpha, mach, cd, cl, cmx, cmy, cmz

    deg2rad = np.pi / 180.0

    xt = np.array(raw[:, 0:2])
    yt = np.array(raw[:, 2:4])
    xlimits = np.array([[-3.0, 10.0], [0.4, 0.90]])

    xt[:, 0] *= deg2rad
    xlimits[0, :] *= deg2rad

    return xt, yt, xlimits


def plot_rans_crm_wing(xt, yt, limits, interp):
    import numpy as np
    import matplotlib

    matplotlib.use("Agg")
    import matplotlib.pyplot as plt

    rad2deg = 180.0 / np.pi

    num = 500
    num_a = 50
    num_M = 50

    x = np.zeros((num, 2))
    colors = ["b", "g", "r", "c", "m", "k", "y"]

    nrow = 3
    ncol = 2

    plt.close()
    fig, axs = plt.subplots(3, 2, figsize=(15, 15))

    # ----------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
    mach_numbers = [0.45, 0.68, 0.80, 0.86]
    legend_entries = []

    alpha_sweep = np.linspace(0.0, 8.0, num)

    for ind, mach in enumerate(mach_numbers):
        x[:, 0] = alpha_sweep / rad2deg
        x[:, 1] = mach
        CD = interp.predict_values(x)[:, 0]
        CL = interp.predict_values(x)[:, 1]

        mask = np.abs(xt[:, 1] - mach) < 1e-10
        axs[0, 0].plot(xt[mask, 0] * rad2deg, yt[mask, 0], "o" + colors[ind])
        axs[0, 0].plot(alpha_sweep, CD, colors[ind])

        mask = np.abs(xt[:, 1] - mach) < 1e-10
        axs[0, 1].plot(xt[mask, 0] * rad2deg, yt[mask, 1], "o" + colors[ind])
        axs[0, 1].plot(alpha_sweep, CL, colors[ind])

        legend_entries.append("M={}".format(mach))
        legend_entries.append("exact")

    axs[0, 0].set(xlabel="alpha (deg)", ylabel="CD")
    axs[0, 0].legend(legend_entries)

    axs[0, 1].set(xlabel="alpha (deg)", ylabel="CL")
    axs[0, 1].legend(legend_entries)

    # ---------------------------------------------------------------------------

    alphas = [2.0, 4.0, 6.0]
    legend_entries = []

    mach_sweep = np.linspace(0.45, 0.86, num)

    for ind, alpha in enumerate(alphas):
        x[:, 0] = alpha / rad2deg
        x[:, 1] = mach_sweep
        CD = interp.predict_values(x)[:, 0]
        CL = interp.predict_values(x)[:, 1]

        axs[1, 0].plot(mach_sweep, CD, colors[ind])
        axs[1, 1].plot(mach_sweep, CL, colors[ind])

        legend_entries.append("alpha={}".format(alpha))

    axs[1, 0].set(xlabel="Mach number", ylabel="CD")
    axs[1, 0].legend(legend_entries)

    axs[1, 1].set(xlabel="Mach number", ylabel="CL")
    axs[1, 1].legend(legend_entries)
```

(continues on next page)

(continued from previous page)

```
    # ---------------------------------------------------------------------------

    x = np.zeros((num_a, num_M, 2))
    x[:, :, 0] = np.outer(np.linspace(0.0, 8.0, num_a), np.ones(num_M)) / rad2deg
    x[:, :, 1] = np.outer(np.ones(num_a), np.linspace(0.45, 0.86, num_M))
    CD = interp.predict_values(x.reshape((num_a * num_M, 2)))[:, 0].reshape(
        (num_a, num_M)
    )
    CL = interp.predict_values(x.reshape((num_a * num_M, 2)))[:, 1].reshape(
        (num_a, num_M)
    )

    axs[2, 0].plot(xt[:, 1], xt[:, 0] * rad2deg, "o")
    axs[2, 0].contour(x[:, :, 1], x[:, :, 0] * rad2deg, CD, 20)
    pcm1 = axs[2, 0].pcolormesh(
        x[:, :, 1],
        x[:, :, 0] * rad2deg,
        CD,
        cmap=plt.get_cmap("rainbow"),
        shading="auto",
    )
    fig.colorbar(pcm1, ax=axs[2, 0])
    axs[2, 0].set(xlabel="Mach number", ylabel="alpha (deg)")
    axs[2, 0].set_title("CD")

    axs[2, 1].plot(xt[:, 1], xt[:, 0] * rad2deg, "o")
    axs[2, 1].contour(x[:, :, 1], x[:, :, 0] * rad2deg, CL, 20)
    pcm2 = axs[2, 1].pcolormesh(
        x[:, :, 1],
        x[:, :, 0] * rad2deg,
        CL,
        cmap=plt.get_cmap("rainbow"),
        shading="auto",
    )
    fig.colorbar(pcm2, ax=axs[2, 1])
    axs[2, 1].set(xlabel="Mach number", ylabel="alpha (deg)")
    axs[2, 1].set_title("CL")

    plt.show()
```

RMTB

```
from smt.surrogate_models import RMTB
from smt.examples.rans_crm_wing.rans_crm_wing import (
    get_rans_crm_wing,
    plot_rans_crm_wing,
)

xt, yt, xlimits = get_rans_crm_wing()
```

(continues on next page)

(continued from previous page)

```
interp = RMTB(
    num_ctrl_pts=20, xlimits=xlimits, nonlinear_maxiter=100, energy_weight=1e-12
)
interp.set_training_values(xt, yt)
interp.train()

plot_rans_crm_wing(xt, yt, xlimits, interp)
```

```
___________________________________________________________________________

                                   RMTB
___________________________________________________________________________

 Problem size

      # training points.        : 35

___________________________________________________________________________

 Training

   Training ...
      Pre-computing matrices ...
         Computing dof2coeff ...
         Computing dof2coeff - done. Time (sec):  0.0000000
         Initializing Hessian ...
         Initializing Hessian - done. Time (sec):  0.0000000
         Computing energy terms ...
         Computing energy terms - done. Time (sec):  0.0050461
         Computing approximation terms ...
         Computing approximation terms - done. Time (sec):  0.0000000
      Pre-computing matrices - done. Time (sec):  0.0050461
      Solving for degrees of freedom ...
         Solving initial startup problem (n=400) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 9.429150220e-02 1.
↪114942861e-02
               Iteration (num., iy, grad. norm, func.) :   0   0 3.285344182e-08 1.
↪793057271e-10
            Solving for output 0 - done. Time (sec):  0.0046282
            Solving for output 1 ...
               Iteration (num., iy, grad. norm, func.) :   0   1 1.955493282e+00 4.
↪799845498e+00
               Iteration (num., iy, grad. norm, func.) :   0   1 2.576015345e-07 4.
↪567654000e-08
            Solving for output 1 - done. Time (sec):  0.0101466
         Solving initial startup problem (n=400) - done. Time (sec):  0.0147748
         Solving nonlinear problem (n=400) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 6.652468783e-09 1.
↪793036975e-10
```

(continues on next page)

(continued from previous page)

```
               Iteration (num., iy, grad. norm, func.) :   0   0 5.849359661e-09 1.
↪703948671e-10
               Iteration (num., iy, grad. norm, func.) :   1   0 3.024325657e-08 1.
↪032975972e-10
               Iteration (num., iy, grad. norm, func.) :   2   0 1.125952154e-08 2.
↪504755353e-11
               Iteration (num., iy, grad. norm, func.) :   3   0 3.703330543e-09 1.
↪068567321e-11
               Iteration (num., iy, grad. norm, func.) :   4   0 2.308909897e-09 9.
↪336148495e-12
               Iteration (num., iy, grad. norm, func.) :   5   0 6.563532482e-10 7.
↪374595825e-12
               Iteration (num., iy, grad. norm, func.) :   6   0 1.899965043e-10 6.
↪526023195e-12
               Iteration (num., iy, grad. norm, func.) :   7   0 3.754706496e-11 6.
↪261587544e-12
               Iteration (num., iy, grad. norm, func.) :   8   0 2.324496290e-11 6.
↪261399259e-12
               Iteration (num., iy, grad. norm, func.) :   9   0 1.605020424e-11 6.
↪260532352e-12
               Iteration (num., iy, grad. norm, func.) :  10   0 9.318527760e-12 6.
↪260087609e-12
               Iteration (num., iy, grad. norm, func.) :  11   0 3.152266674e-12 6.
↪256581242e-12
               Iteration (num., iy, grad. norm, func.) :  12   0 6.297025943e-13 6.
↪255685336e-12
            Solving for output 0 - done. Time (sec):  0.0898802
            Solving for output 1 ...
               Iteration (num., iy, grad. norm, func.) :   0   1 9.729427657e-08 4.
↪567642346e-08
               Iteration (num., iy, grad. norm, func.) :   0   1 9.338325576e-08 4.
↪538217216e-08
               Iteration (num., iy, grad. norm, func.) :   1   1 2.905713447e-06 3.
↪252950315e-08
               Iteration (num., iy, grad. norm, func.) :   2   1 8.633229768e-07 4.
↪671460509e-09
               Iteration (num., iy, grad. norm, func.) :   3   1 3.070729855e-07 2.
↪348794793e-09
               Iteration (num., iy, grad. norm, func.) :   4   1 2.500598178e-07 1.
↪843742999e-09
               Iteration (num., iy, grad. norm, func.) :   5   1 7.358885180e-08 6.
↪117826659e-10
               Iteration (num., iy, grad. norm, func.) :   6   1 2.131842881e-08 4.
↪697487696e-10
               Iteration (num., iy, grad. norm, func.) :   7   1 8.392367562e-09 4.
↪379120867e-10
               Iteration (num., iy, grad. norm, func.) :   8   1 1.441027672e-08 3.
↪875329318e-10
               Iteration (num., iy, grad. norm, func.) :   9   1 4.418732381e-09 2.
↪976219461e-10
               Iteration (num., iy, grad. norm, func.) :  10   1 1.162494175e-09 2.
↪732104388e-10
```

(continues on next page)

(continued from previous page)

```
              Iteration (num., iy, grad. norm, func.) :  11   1 1.029882269e-09 2.
↪731585062e-10
              Iteration (num., iy, grad. norm, func.) :  12   1 5.996851815e-10 2.
↪729944378e-10
              Iteration (num., iy, grad. norm, func.) :  13   1 2.773575123e-10 2.
↪720631775e-10
              Iteration (num., iy, grad. norm, func.) :  14   1 4.839897428e-11 2.
↪714806989e-10
              Iteration (num., iy, grad. norm, func.) :  15   1 3.510264913e-11 2.
↪714720312e-10
              Iteration (num., iy, grad. norm, func.) :  16   1 4.275628954e-11 2.
↪714584243e-10
              Iteration (num., iy, grad. norm, func.) :  17   1 4.600758315e-11 2.
↪714215071e-10
              Iteration (num., iy, grad. norm, func.) :  18   1 3.755405225e-11 2.
↪713917882e-10
              Iteration (num., iy, grad. norm, func.) :  19   1 2.030607591e-11 2.
↪713645513e-10
              Iteration (num., iy, grad. norm, func.) :  20   1 2.354692086e-11 2.
↪713542916e-10
              Iteration (num., iy, grad. norm, func.) :  21   1 1.575101494e-11 2.
↪713528375e-10
              Iteration (num., iy, grad. norm, func.) :  22   1 1.050318251e-11 2.
↪713503217e-10
              Iteration (num., iy, grad. norm, func.) :  23   1 1.594812517e-11 2.
↪713483733e-10
              Iteration (num., iy, grad. norm, func.) :  24   1 3.895308868e-12 2.
↪713460532e-10
              Iteration (num., iy, grad. norm, func.) :  25   1 5.032359799e-12 2.
↪713456491e-10
              Iteration (num., iy, grad. norm, func.) :  26   1 3.021954913e-12 2.
↪713455900e-10
              Iteration (num., iy, grad. norm, func.) :  27   1 4.520394294e-12 2.
↪713453462e-10
              Iteration (num., iy, grad. norm, func.) :  28   1 7.790196797e-13 2.
↪713450408e-10
           Solving for output 1 - done. Time (sec):  0.2023385
        Solving nonlinear problem (n=400) - done. Time (sec):  0.2922187
     Solving for degrees of freedom - done. Time (sec):  0.3069935
  Training - done. Time (sec):  0.3120396
___________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________
```

(continues on next page)

(continued from previous page)

```
Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0066257

  Prediction time/pt. (sec) :  0.0000133

___________________________________________________________________________

Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000
```

(continues on next page)

(continued from previous page)

```
  Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

     # eval points. : 500
```

(continues on next page)

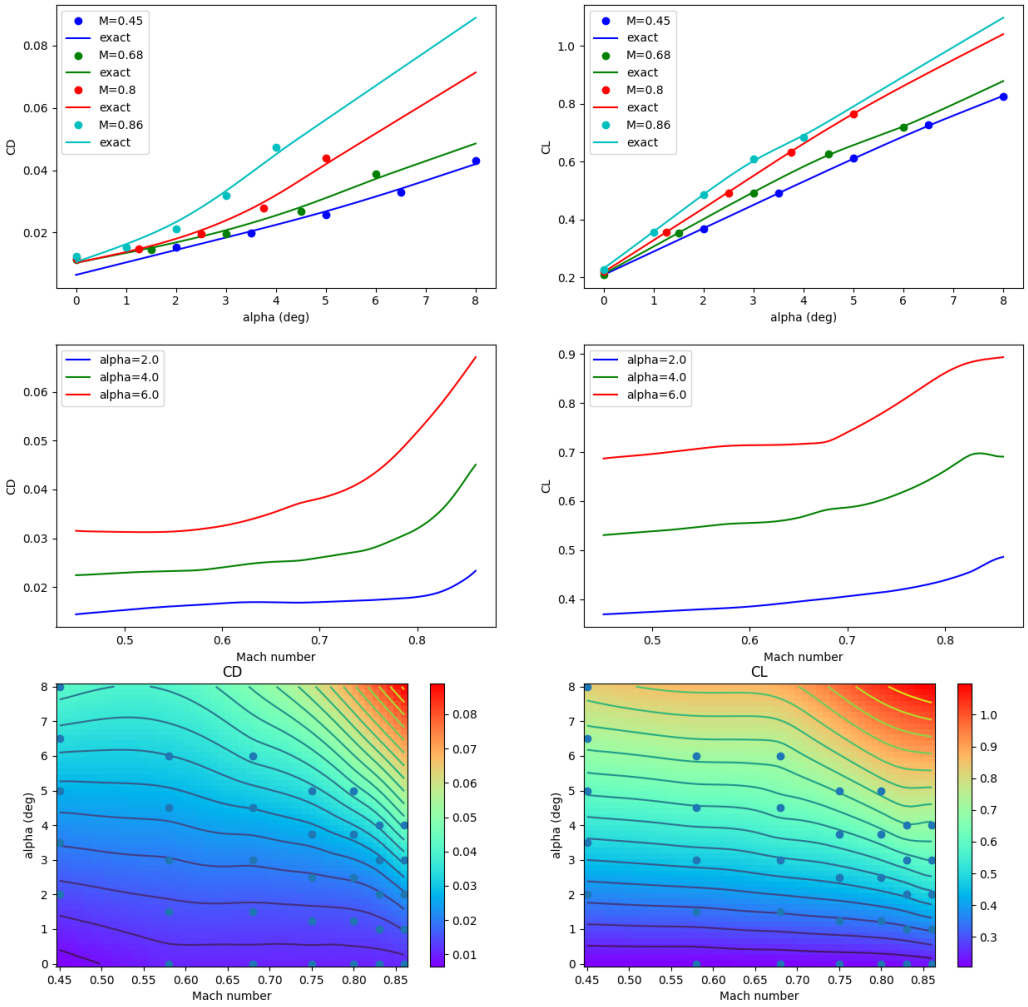
(continued from previous page)

```
  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

      # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

      # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

      # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

      # eval points. : 2500

  Predicting ...
  Predicting - done. Time (sec):  0.0020466

  Prediction time/pt. (sec) :  0.0000008


___________________________________________________________________________

 Evaluation
```

(continues on next page)

(continued from previous page)

```
    # eval points. : 2500

Predicting ...
Predicting - done. Time (sec):  0.0000000

Prediction time/pt. (sec) :  0.0000000
```



RMTC

```
from smt.surrogate_models import RMTC
from smt.examples.rans_crm_wing.rans_crm_wing import (
    get_rans_crm_wing,
    plot_rans_crm_wing,
)

xt, yt, xlimits = get_rans_crm_wing()

interp = RMTC(
    num_elements=20, xlimits=xlimits, nonlinear_maxiter=100, energy_weight=1e-10
)
interp.set_training_values(xt, yt)
interp.train()

plot_rans_crm_wing(xt, yt, xlimits, interp)
```

```
___________________________________________________________________________

                                   RMTC
___________________________________________________________________________

 Problem size

      # training points.        : 35

___________________________________________________________________________

 Training

   Training ...
      Pre-computing matrices ...
         Computing dof2coeff ...
         Computing dof2coeff - done. Time (sec):  0.0020320
         Initializing Hessian ...
         Initializing Hessian - done. Time (sec):  0.0000000
         Computing energy terms ...
         Computing energy terms - done. Time (sec):  0.0080001
         Computing approximation terms ...
         Computing approximation terms - done. Time (sec):  0.0010014
      Pre-computing matrices - done. Time (sec):  0.0110335
      Solving for degrees of freedom ...
         Solving initial startup problem (n=1764) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 1.279175539e-01 1.
↪114942861e-02
               Iteration (num., iy, grad. norm, func.) :   0   0 4.269613325e-06 2.
↪206667028e-08
            Solving for output 0 - done. Time (sec):  0.0160003
            Solving for output 1 ...
               Iteration (num., iy, grad. norm, func.) :   0   1 2.653045755e+00 4.
↪799845498e+00
```

(continues on next page)

(continued from previous page)

```
            Iteration (num., iy, grad. norm, func.) :   0   1 8.181842136e-05 6.
↪499264090e-06
         Solving for output 1 - done. Time (sec):  0.0116787
      Solving initial startup problem (n=1764) - done. Time (sec):  0.0327158
      Solving nonlinear problem (n=1764) ...
         Solving for output 0 ...
            Iteration (num., iy, grad. norm, func.) :   0   0 8.706466102e-07 2.
↪205254659e-08
            Iteration (num., iy, grad. norm, func.) :   0   0 9.563774984e-07 1.
↪751091936e-08
            Iteration (num., iy, grad. norm, func.) :   1   0 3.540967624e-07 3.
↪271690535e-09
            Iteration (num., iy, grad. norm, func.) :   2   0 1.182287333e-07 1.
↪052646339e-09
            Iteration (num., iy, grad. norm, func.) :   3   0 6.338489775e-08 5.
↪346409167e-10
            Iteration (num., iy, grad. norm, func.) :   4   0 3.377551606e-08 4.
↪104301514e-10
            Iteration (num., iy, grad. norm, func.) :   5   0 2.242910012e-08 3.
↪753259905e-10
            Iteration (num., iy, grad. norm, func.) :   6   0 1.964970772e-08 3.
↪751489728e-10
            Iteration (num., iy, grad. norm, func.) :   7   0 1.527094421e-08 3.
↪661593175e-10
            Iteration (num., iy, grad. norm, func.) :   8   0 1.690202790e-08 3.
↪641811225e-10
            Iteration (num., iy, grad. norm, func.) :   9   0 1.404808000e-08 3.
↪400408379e-10
            Iteration (num., iy, grad. norm, func.) :  10   0 8.545659714e-09 3.
↪088440096e-10
            Iteration (num., iy, grad. norm, func.) :  11   0 2.764134834e-09 2.
↪905853361e-10
            Iteration (num., iy, grad. norm, func.) :  12   0 2.564977956e-09 2.
↪894136297e-10
            Iteration (num., iy, grad. norm, func.) :  13   0 2.564977954e-09 2.
↪894136297e-10
            Iteration (num., iy, grad. norm, func.) :  14   0 2.564977954e-09 2.
↪894136297e-10
            Iteration (num., iy, grad. norm, func.) :  15   0 3.829116565e-09 2.
↪884285847e-10
            Iteration (num., iy, grad. norm, func.) :  16   0 7.531911834e-10 2.
↪872989730e-10
            Iteration (num., iy, grad. norm, func.) :  17   0 1.467359380e-09 2.
↪870606984e-10
            Iteration (num., iy, grad. norm, func.) :  18   0 9.879625566e-10 2.
↪869763841e-10
            Iteration (num., iy, grad. norm, func.) :  19   0 9.122177076e-10 2.
↪869563186e-10
            Iteration (num., iy, grad. norm, func.) :  20   0 8.902786862e-10 2.
↪869003688e-10
            Iteration (num., iy, grad. norm, func.) :  21   0 1.135119850e-09 2.
↪867529790e-10
```

(continues on next page)

(continued from previous page)

```
            Iteration (num., iy, grad. norm, func.) :  22   0 7.995220792e-10 2.
↪866276258e-10
            Iteration (num., iy, grad. norm, func.) :  23   0 3.328505805e-10 2.
↪865652621e-10
            Iteration (num., iy, grad. norm, func.) :  24   0 2.914967622e-10 2.
↪865626286e-10
            Iteration (num., iy, grad. norm, func.) :  25   0 4.202766941e-10 2.
↪865616065e-10
            Iteration (num., iy, grad. norm, func.) :  26   0 3.877343718e-10 2.
↪865534583e-10
            Iteration (num., iy, grad. norm, func.) :  27   0 4.306678978e-10 2.
↪865437762e-10
            Iteration (num., iy, grad. norm, func.) :  28   0 2.673682253e-10 2.
↪865299886e-10
            Iteration (num., iy, grad. norm, func.) :  29   0 3.275156607e-10 2.
↪865207653e-10
            Iteration (num., iy, grad. norm, func.) :  30   0 1.930712327e-10 2.
↪865122795e-10
            Iteration (num., iy, grad. norm, func.) :  31   0 2.110880200e-10 2.
↪865087706e-10
            Iteration (num., iy, grad. norm, func.) :  32   0 1.473132344e-10 2.
↪865051960e-10
            Iteration (num., iy, grad. norm, func.) :  33   0 1.942718886e-10 2.
↪865041883e-10
            Iteration (num., iy, grad. norm, func.) :  34   0 1.115813968e-10 2.
↪865019571e-10
            Iteration (num., iy, grad. norm, func.) :  35   0 1.662319465e-10 2.
↪864991211e-10
            Iteration (num., iy, grad. norm, func.) :  36   0 6.180638003e-11 2.
↪864957999e-10
            Iteration (num., iy, grad. norm, func.) :  37   0 1.062616250e-10 2.
↪864956662e-10
            Iteration (num., iy, grad. norm, func.) :  38   0 6.888942793e-11 2.
↪864952745e-10
            Iteration (num., iy, grad. norm, func.) :  39   0 1.155104075e-10 2.
↪864948247e-10
            Iteration (num., iy, grad. norm, func.) :  40   0 5.301847100e-11 2.
↪864940891e-10
            Iteration (num., iy, grad. norm, func.) :  41   0 6.018140313e-11 2.
↪864937454e-10
            Iteration (num., iy, grad. norm, func.) :  42   0 4.486986044e-11 2.
↪864935284e-10
            Iteration (num., iy, grad. norm, func.) :  43   0 4.403159581e-11 2.
↪864932581e-10
            Iteration (num., iy, grad. norm, func.) :  44   0 4.446981101e-11 2.
↪864931203e-10
            Iteration (num., iy, grad. norm, func.) :  45   0 3.306185597e-11 2.
↪864929855e-10
            Iteration (num., iy, grad. norm, func.) :  46   0 3.855294007e-11 2.
↪864928687e-10
            Iteration (num., iy, grad. norm, func.) :  47   0 2.386145287e-11 2.
↪864927343e-10
```

(continues on next page)

(continued from previous page)

```
              Iteration (num., iy, grad. norm, func.) :  48   0 3.479120870e-11 2.
↪864926795e-10
              Iteration (num., iy, grad. norm, func.) :  49   0 2.038523321e-11 2.
↪864926305e-10
              Iteration (num., iy, grad. norm, func.) :  50   0 2.842194813e-11 2.
↪864925264e-10
              Iteration (num., iy, grad. norm, func.) :  51   0 5.869027097e-12 2.
↪864924490e-10
              Iteration (num., iy, grad. norm, func.) :  52   0 5.498904403e-12 2.
↪864924489e-10
              Iteration (num., iy, grad. norm, func.) :  53   0 7.306931689e-12 2.
↪864924453e-10
              Iteration (num., iy, grad. norm, func.) :  54   0 6.504376748e-12 2.
↪864924392e-10
              Iteration (num., iy, grad. norm, func.) :  55   0 1.045430092e-11 2.
↪864924382e-10
              Iteration (num., iy, grad. norm, func.) :  56   0 5.954310611e-12 2.
↪864924378e-10
              Iteration (num., iy, grad. norm, func.) :  57   0 8.350346501e-12 2.
↪864924344e-10
              Iteration (num., iy, grad. norm, func.) :  58   0 4.203044469e-12 2.
↪864924276e-10
              Iteration (num., iy, grad. norm, func.) :  59   0 4.808460418e-12 2.
↪864924239e-10
              Iteration (num., iy, grad. norm, func.) :  60   0 3.033848305e-12 2.
↪864924214e-10
              Iteration (num., iy, grad. norm, func.) :  61   0 4.617206475e-12 2.
↪864924203e-10
              Iteration (num., iy, grad. norm, func.) :  62   0 2.637312920e-12 2.
↪864924192e-10
              Iteration (num., iy, grad. norm, func.) :  63   0 3.304647023e-12 2.
↪864924191e-10
              Iteration (num., iy, grad. norm, func.) :  64   0 2.271894901e-12 2.
↪864924185e-10
              Iteration (num., iy, grad. norm, func.) :  65   0 3.579944006e-12 2.
↪864924178e-10
              Iteration (num., iy, grad. norm, func.) :  66   0 1.348728833e-12 2.
↪864924169e-10
              Iteration (num., iy, grad. norm, func.) :  67   0 1.806375611e-12 2.
↪864924165e-10
              Iteration (num., iy, grad. norm, func.) :  68   0 1.363756532e-12 2.
↪864924163e-10
              Iteration (num., iy, grad. norm, func.) :  69   0 1.712838050e-12 2.
↪864924161e-10
              Iteration (num., iy, grad. norm, func.) :  70   0 9.876379889e-13 2.
↪864924158e-10
           Solving for output 0 - done. Time (sec):  1.1035955
           Solving for output 1 ...
              Iteration (num., iy, grad. norm, func.) :   0   1 1.428317959e-05 6.
↪494040493e-06
              Iteration (num., iy, grad. norm, func.) :   0   1 1.428267378e-05 6.
↪247373107e-06
```

(continues on next page)

(continued from previous page)

```
            Iteration (num., iy, grad. norm, func.) :   1   1 1.481274881e-05 8.
↪062583973e-07
            Iteration (num., iy, grad. norm, func.) :   2   1 1.867225216e-05 3.
↪693188434e-07
            Iteration (num., iy, grad. norm, func.) :   3   1 5.730360645e-06 1.
↪284492940e-07
            Iteration (num., iy, grad. norm, func.) :   4   1 4.881295024e-06 1.
↪022046592e-07
            Iteration (num., iy, grad. norm, func.) :   5   1 1.494072262e-06 3.
↪710248943e-08
            Iteration (num., iy, grad. norm, func.) :   6   1 1.213916647e-06 3.
↪116830197e-08
            Iteration (num., iy, grad. norm, func.) :   7   1 9.300654262e-07 3.
↪027500084e-08
            Iteration (num., iy, grad. norm, func.) :   8   1 3.309011543e-07 2.
↪989622146e-08
            Iteration (num., iy, grad. norm, func.) :   9   1 1.737272135e-07 2.
↪372116544e-08
            Iteration (num., iy, grad. norm, func.) :  10   1 1.192375978e-07 1.
↪812151163e-08
            Iteration (num., iy, grad. norm, func.) :  11   1 2.823830003e-08 1.
↪492230509e-08
            Iteration (num., iy, grad. norm, func.) :  12   1 3.075752735e-08 1.
↪479543683e-08
            Iteration (num., iy, grad. norm, func.) :  13   1 3.075752733e-08 1.
↪479543683e-08
            Iteration (num., iy, grad. norm, func.) :  14   1 3.075752733e-08 1.
↪479543683e-08
            Iteration (num., iy, grad. norm, func.) :  15   1 3.964583353e-08 1.
↪465943906e-08
            Iteration (num., iy, grad. norm, func.) :  16   1 7.803411275e-09 1.
↪450888835e-08
            Iteration (num., iy, grad. norm, func.) :  17   1 5.277665204e-09 1.
↪449730293e-08
            Iteration (num., iy, grad. norm, func.) :  18   1 1.430068303e-08 1.
↪449498297e-08
            Iteration (num., iy, grad. norm, func.) :  19   1 9.248180067e-09 1.
↪449172013e-08
            Iteration (num., iy, grad. norm, func.) :  20   1 9.340420132e-09 1.
↪449165474e-08
            Iteration (num., iy, grad. norm, func.) :  21   1 7.764038380e-09 1.
↪448168658e-08
            Iteration (num., iy, grad. norm, func.) :  22   1 6.859229342e-09 1.
↪447218601e-08
            Iteration (num., iy, grad. norm, func.) :  23   1 3.419486695e-09 1.
↪446903012e-08
            Iteration (num., iy, grad. norm, func.) :  24   1 3.133529225e-09 1.
↪446840459e-08
            Iteration (num., iy, grad. norm, func.) :  25   1 3.234821678e-09 1.
↪446804176e-08
            Iteration (num., iy, grad. norm, func.) :  26   1 4.311442242e-09 1.
↪446751633e-08
```

(continues on next page)

(continued from previous page)

```
              Iteration (num., iy, grad. norm, func.) :  27   1 2.174738895e-09 1.
↪446676165e-08
              Iteration (num., iy, grad. norm, func.) :  28   1 3.665734460e-09 1.
↪446612261e-08
              Iteration (num., iy, grad. norm, func.) :  29   1 1.819294496e-09 1.
↪446525702e-08
              Iteration (num., iy, grad. norm, func.) :  30   1 1.688849698e-09 1.
↪446466796e-08
              Iteration (num., iy, grad. norm, func.) :  31   1 1.477487561e-09 1.
↪446425046e-08
              Iteration (num., iy, grad. norm, func.) :  32   1 1.048263967e-09 1.
↪446400848e-08
              Iteration (num., iy, grad. norm, func.) :  33   1 9.830692267e-10 1.
↪446399517e-08
              Iteration (num., iy, grad. norm, func.) :  34   1 1.112492217e-09 1.
↪446398817e-08
              Iteration (num., iy, grad. norm, func.) :  35   1 1.196589508e-09 1.
↪446389015e-08
              Iteration (num., iy, grad. norm, func.) :  36   1 9.757587962e-10 1.
↪446374330e-08
              Iteration (num., iy, grad. norm, func.) :  37   1 3.612625839e-10 1.
↪446363742e-08
              Iteration (num., iy, grad. norm, func.) :  38   1 3.212287016e-10 1.
↪446362495e-08
              Iteration (num., iy, grad. norm, func.) :  39   1 3.380688480e-10 1.
↪446361644e-08
              Iteration (num., iy, grad. norm, func.) :  40   1 4.451214840e-10 1.
↪446360940e-08
              Iteration (num., iy, grad. norm, func.) :  41   1 4.125923914e-10 1.
↪446360055e-08
              Iteration (num., iy, grad. norm, func.) :  42   1 3.316955216e-10 1.
↪446358697e-08
              Iteration (num., iy, grad. norm, func.) :  43   1 2.785079803e-10 1.
↪446358341e-08
              Iteration (num., iy, grad. norm, func.) :  44   1 2.210639567e-10 1.
↪446357765e-08
              Iteration (num., iy, grad. norm, func.) :  45   1 2.297590270e-10 1.
↪446357276e-08
              Iteration (num., iy, grad. norm, func.) :  46   1 1.366682844e-10 1.
↪446356930e-08
              Iteration (num., iy, grad. norm, func.) :  47   1 2.695700896e-10 1.
↪446356498e-08
              Iteration (num., iy, grad. norm, func.) :  48   1 4.755236155e-11 1.
↪446356106e-08
              Iteration (num., iy, grad. norm, func.) :  49   1 3.293668661e-11 1.
↪446356098e-08
              Iteration (num., iy, grad. norm, func.) :  50   1 6.150697758e-11 1.
↪446356068e-08
              Iteration (num., iy, grad. norm, func.) :  51   1 4.910616134e-11 1.
↪446356031e-08
              Iteration (num., iy, grad. norm, func.) :  52   1 6.141409525e-11 1.
↪446355996e-08
```

(continues on next page)

(continued from previous page)

```
            Iteration (num., iy, grad. norm, func.) :  53   1 2.254141518e-11 1.
↪446355962e-08
            Iteration (num., iy, grad. norm, func.) :  54   1 2.548291847e-11 1.
↪446355959e-08
            Iteration (num., iy, grad. norm, func.) :  55   1 2.626783512e-11 1.
↪446355952e-08
            Iteration (num., iy, grad. norm, func.) :  56   1 2.951026223e-11 1.
↪446355940e-08
            Iteration (num., iy, grad. norm, func.) :  57   1 2.416840918e-11 1.
↪446355932e-08
            Iteration (num., iy, grad. norm, func.) :  58   1 1.583781374e-11 1.
↪446355927e-08
            Iteration (num., iy, grad. norm, func.) :  59   1 1.331983010e-11 1.
↪446355926e-08
            Iteration (num., iy, grad. norm, func.) :  60   1 1.538617826e-11 1.
↪446355923e-08
            Iteration (num., iy, grad. norm, func.) :  61   1 1.162017872e-11 1.
↪446355920e-08
            Iteration (num., iy, grad. norm, func.) :  62   1 1.308859447e-11 1.
↪446355918e-08
            Iteration (num., iy, grad. norm, func.) :  63   1 7.793346965e-12 1.
↪446355918e-08
            Iteration (num., iy, grad. norm, func.) :  64   1 1.252464249e-11 1.
↪446355918e-08
            Iteration (num., iy, grad. norm, func.) :  65   1 6.203040233e-12 1.
↪446355917e-08
            Iteration (num., iy, grad. norm, func.) :  66   1 6.342939419e-12 1.
↪446355916e-08
            Iteration (num., iy, grad. norm, func.) :  67   1 4.780453656e-12 1.
↪446355916e-08
            Iteration (num., iy, grad. norm, func.) :  68   1 4.876176332e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  69   1 2.643218436e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  70   1 2.173821409e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  71   1 3.103482666e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  72   1 3.133075055e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  73   1 2.316398650e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  74   1 2.928148144e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  75   1 1.225656672e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  76   1 1.460100479e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  77   1 1.400462776e-12 1.
↪446355915e-08
            Iteration (num., iy, grad. norm, func.) :  78   1 1.694580342e-12 1.
↪446355915e-08
```

(continues on next page)

(continued from previous page)

```
              Iteration (num., iy, grad. norm, func.) :  79   1 8.560025835e-13 1.
↪446355915e-08
           Solving for output 1 - done. Time (sec):  1.2438643
        Solving nonlinear problem (n=1764) - done. Time (sec):  2.3474598
     Solving for degrees of freedom - done. Time (sec):  2.3801756
   Training - done. Time (sec):  2.3912091
_________________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

_________________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0009987

   Prediction time/pt. (sec) :  0.0000020

_________________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

_________________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

_________________________________________________________________________________
```

(continues on next page)

(continued from previous page)

```
Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

------------------------------------------------------------------------------

Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0010324

  Prediction time/pt. (sec) :  0.0000021

------------------------------------------------------------------------------

Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0009999

  Prediction time/pt. (sec) :  0.0000020

------------------------------------------------------------------------------

Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

------------------------------------------------------------------------------

Evaluation

     # eval points. : 500

  Predicting ...
  Predicting - done. Time (sec):  0.0009999

  Prediction time/pt. (sec) :  0.0000020
```

(continues on next page)

(continued from previous page)

```
___________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0009820

   Prediction time/pt. (sec) :  0.0000020

___________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
   Predicting - done. Time (sec):  0.0009930

   Prediction time/pt. (sec) :  0.0000020

___________________________________________________________________________

 Evaluation

      # eval points. : 500

   Predicting ...
```

(continues on next page)

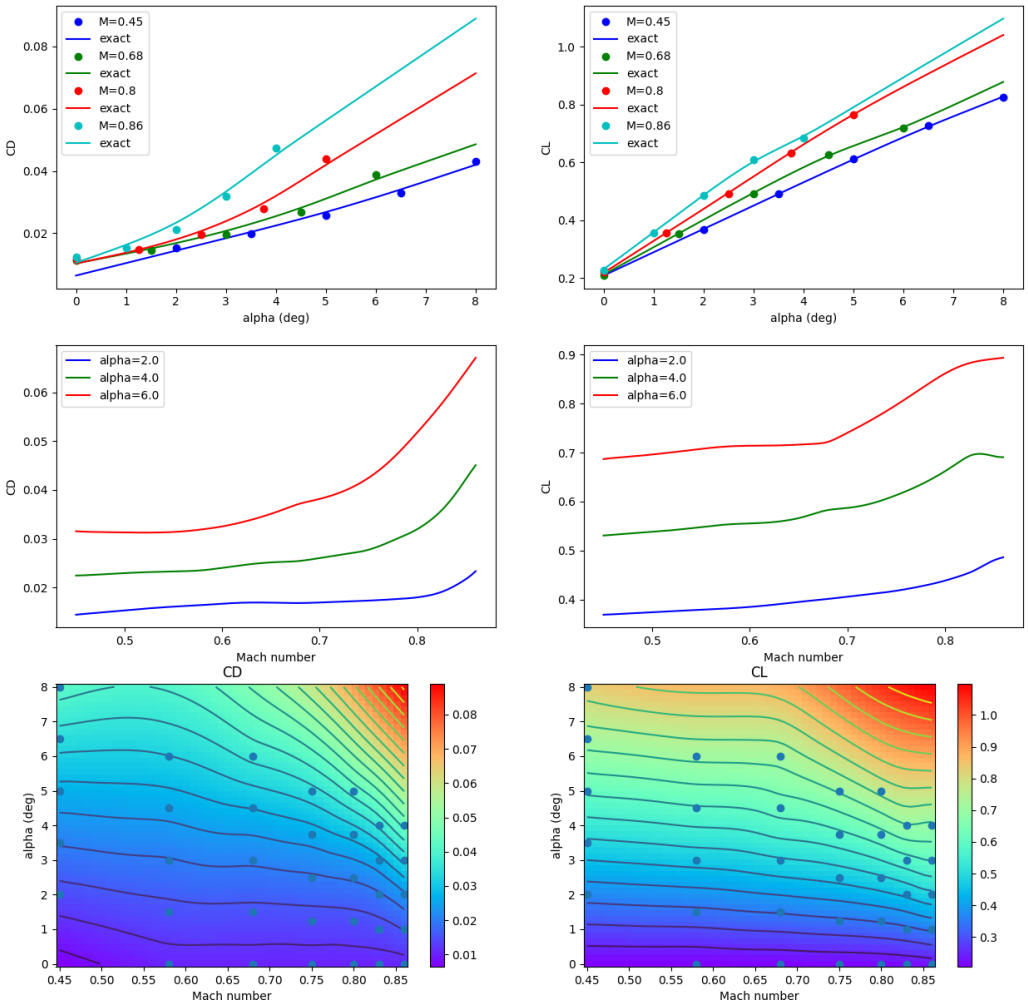
(continued from previous page)

```
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

      # eval points. : 2500

   Predicting ...
   Predicting - done. Time (sec):  0.0020068

   Prediction time/pt. (sec) :  0.0000008


___________________________________________________________________________

 Evaluation

      # eval points. : 2500

   Predicting ...
   Predicting - done. Time (sec):  0.0019960

   Prediction time/pt. (sec) :  0.0000008
```

3.5.3 Boeing 777 engine data set

```
import numpy as np
import os


def get_b777_engine():
    this_dir = os.path.split(__file__)[0]

    nt = 12 * 11 * 8
    xt = np.loadtxt(os.path.join(this_dir, "b777_engine_inputs.dat")).reshape((nt, 3))
    yt = np.loadtxt(os.path.join(this_dir, "b777_engine_outputs.dat")).reshape((nt, 2))
    dyt_dxt = np.loadtxt(os.path.join(this_dir, "b777_engine_derivs.dat")).reshape(
        (nt, 2, 3)
    )

    xlimits = np.array([[0, 0.9], [0, 15], [0, 1.0]])

    return xt, yt, dyt_dxt, xlimits


def plot_b777_engine(xt, yt, limits, interp):
    import numpy as np
    import matplotlib
```

(continues on next page)


M=0.45
exact
M=0.68
exact
M=0.8
exact
M=0.86
exact
CD
alpha (deg)
CL
alpha (deg)
alpha=2.0
alpha=4.0
alpha=6.0
CD
Mach number
CL
Mach number
CD
alpha (deg)
Mach number
CL
alpha (deg)
Mach number

(continued from previous page)

```
matplotlib.use("Agg")
import matplotlib.pyplot as plt

val_M = np.array(
    [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.85, 0.9]
)  # 12
val_h = np.array(
    [0.0, 0.6096, 1.524, 3.048, 4.572, 6.096, 7.62, 9.144, 10.668, 11.8872, 13.1064]
)  # 11
val_t = np.array([0.05, 0.2, 0.3, 0.4, 0.6, 0.8, 0.9, 1.0])  # 8

def get_pts(xt, yt, iy, ind_M=None, ind_h=None, ind_t=None):
    eps = 1e-5

    if ind_M is not None:
        M = val_M[ind_M]
        keep = abs(xt[:, 0] - M) < eps
        xt = xt[keep, :]
        yt = yt[keep, :]
    if ind_h is not None:
        h = val_h[ind_h]
        keep = abs(xt[:, 1] - h) < eps
        xt = xt[keep, :]
        yt = yt[keep, :]
    if ind_t is not None:
        t = val_t[ind_t]
        keep = abs(xt[:, 2] - t) < eps
        xt = xt[keep, :]
        yt = yt[keep, :]

    if ind_M is None:
        data = xt[:, 0], yt[:, iy]
    elif ind_h is None:
        data = xt[:, 1], yt[:, iy]
    elif ind_t is None:
        data = xt[:, 2], yt[:, iy]

    if iy == 0:
        data = data[0], data[1] / 1e6
    elif iy == 1:
        data = data[0], data[1] / 1e-4

    return data

num = 100
x = np.zeros((num, 3))
lins_M = np.linspace(0.0, 0.9, num)
lins_h = np.linspace(0.0, 13.1064, num)
lins_t = np.linspace(0.05, 1.0, num)

def get_x(ind_M=None, ind_h=None, ind_t=None):
```

(continues on next page)

(continued from previous page)

```
        x = np.zeros((num, 3))
        x[:, 0] = lins_M
        x[:, 1] = lins_h
        x[:, 2] = lins_t
        if ind_M:
            x[:, 0] = val_M[ind_M]
        if ind_h:
            x[:, 1] = val_h[ind_h]
        if ind_t:
            x[:, 2] = val_t[ind_t]
        return x

    nrow = 6
    ncol = 2

    ind_M_1 = -2
    ind_M_2 = -5

    ind_t_1 = 1
    ind_t_2 = -1

    plt.close()

    # --------------------

    fig, axs = plt.subplots(6, 2, gridspec_kw={"hspace": 0.5}, figsize=(15, 25))

    axs[0, 0].set_title("M={}".format(val_M[ind_M_1]))
    axs[0, 0].set(xlabel="throttle", ylabel="thrust (x 1e6 N)")

    axs[0, 1].set_title("M={}".format(val_M[ind_M_1]))
    axs[0, 1].set(xlabel="throttle", ylabel="SFC (x 1e-3 N/N/s)")

    axs[1, 0].set_title("M={}".format(val_M[ind_M_2]))
    axs[1, 0].set(xlabel="throttle", ylabel="thrust (x 1e6 N)")

    axs[1, 1].set_title("M={}".format(val_M[ind_M_2]))
    axs[1, 1].set(xlabel="throttle", ylabel="SFC (x 1e-3 N/N/s)")

    # --------------------

    axs[2, 0].set_title("throttle={}".format(val_t[ind_t_1]))
    axs[2, 0].set(xlabel="altitude (km)", ylabel="thrust (x 1e6 N)")

    axs[2, 1].set_title("throttle={}".format(val_t[ind_t_1]))
    axs[2, 1].set(xlabel="altitude (km)", ylabel="SFC (x 1e-3 N/N/s)")

    axs[3, 0].set_title("throttle={}".format(val_t[ind_t_2]))
    axs[3, 0].set(xlabel="altitude (km)", ylabel="thrust (x 1e6 N)")

    axs[3, 1].set_title("throttle={}".format(val_t[ind_t_2]))
    axs[3, 1].set(xlabel="altitude (km)", ylabel="SFC (x 1e-3 N/N/s)")
```

(continues on next page)

(continued from previous page)

```
    # --------------------

    axs[4, 0].set_title("throttle={}".format(val_t[ind_t_1]))
    axs[4, 0].set(xlabel="Mach number", ylabel="thrust (x 1e6 N)")

    axs[4, 1].set_title("throttle={}".format(val_t[ind_t_1]))
    axs[4, 1].set(xlabel="Mach number", ylabel="SFC (x 1e-3 N/N/s)")

    axs[5, 0].set_title("throttle={}".format(val_t[ind_t_2]))
    axs[5, 0].set(xlabel="Mach number", ylabel="thrust (x 1e6 N)")

    axs[5, 1].set_title("throttle={}".format(val_t[ind_t_2]))
    axs[5, 1].set(xlabel="Mach number", ylabel="SFC (x 1e-3 N/N/s)")

    ind_h_list = [0, 4, 7, 10]
    ind_h_list = [4, 7, 10]

    ind_M_list = [0, 3, 6, 11]
    ind_M_list = [3, 6, 11]

    colors = ["b", "r", "g", "c", "m"]

    # --------------------------------------------------------------------------

    # Throttle slices
    for k, ind_h in enumerate(ind_h_list):
        ind_M = ind_M_1
        x = get_x(ind_M=ind_M, ind_h=ind_h)
        y = interp.predict_values(x)

        xt_, yt_ = get_pts(xt, yt, 0, ind_M=ind_M, ind_h=ind_h)
        axs[0, 0].plot(xt_, yt_, "o" + colors[k])
        axs[0, 0].plot(lins_t, y[:, 0] / 1e6, colors[k])

        xt_, yt_ = get_pts(xt, yt, 1, ind_M=ind_M, ind_h=ind_h)
        axs[0, 1].plot(xt_, yt_, "o" + colors[k])
        axs[0, 1].plot(lins_t, y[:, 1] / 1e-4, colors[k])

        ind_M = ind_M_2
        x = get_x(ind_M=ind_M, ind_h=ind_h)
        y = interp.predict_values(x)

        xt_, yt_ = get_pts(xt, yt, 0, ind_M=ind_M, ind_h=ind_h)
        axs[1, 0].plot(xt_, yt_, "o" + colors[k])
        axs[1, 0].plot(lins_t, y[:, 0] / 1e6, colors[k])

        xt_, yt_ = get_pts(xt, yt, 1, ind_M=ind_M, ind_h=ind_h)
        axs[1, 1].plot(xt_, yt_, "o" + colors[k])
        axs[1, 1].plot(lins_t, y[:, 1] / 1e-4, colors[k])

    # --------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
    # Altitude slices
    for k, ind_M in enumerate(ind_M_list):
        ind_t = ind_t_1
        x = get_x(ind_M=ind_M, ind_t=ind_t)
        y = interp.predict_values(x)

        xt_, yt_ = get_pts(xt, yt, 0, ind_M=ind_M, ind_t=ind_t)
        axs[2, 0].plot(xt_, yt_, "o" + colors[k])
        axs[2, 0].plot(lins_h, y[:, 0] / 1e6, colors[k])

        xt_, yt_ = get_pts(xt, yt, 1, ind_M=ind_M, ind_t=ind_t)
        axs[2, 1].plot(xt_, yt_, "o" + colors[k])
        axs[2, 1].plot(lins_h, y[:, 1] / 1e-4, colors[k])

        ind_t = ind_t_2
        x = get_x(ind_M=ind_M, ind_t=ind_t)
        y = interp.predict_values(x)

        xt_, yt_ = get_pts(xt, yt, 0, ind_M=ind_M, ind_t=ind_t)
        axs[3, 0].plot(xt_, yt_, "o" + colors[k])
        axs[3, 0].plot(lins_h, y[:, 0] / 1e6, colors[k])

        xt_, yt_ = get_pts(xt, yt, 1, ind_M=ind_M, ind_t=ind_t)
        axs[3, 1].plot(xt_, yt_, "o" + colors[k])
        axs[3, 1].plot(lins_h, y[:, 1] / 1e-4, colors[k])

    # ---------------------------------------------------------------------------

    # Mach number slices
    for k, ind_h in enumerate(ind_h_list):
        ind_t = ind_t_1
        x = get_x(ind_t=ind_t, ind_h=ind_h)
        y = interp.predict_values(x)

        xt_, yt_ = get_pts(xt, yt, 0, ind_h=ind_h, ind_t=ind_t)
        axs[4, 0].plot(xt_, yt_, "o" + colors[k])
        axs[4, 0].plot(lins_M, y[:, 0] / 1e6, colors[k])

        xt_, yt_ = get_pts(xt, yt, 1, ind_h=ind_h, ind_t=ind_t)
        axs[4, 1].plot(xt_, yt_, "o" + colors[k])
        axs[4, 1].plot(lins_M, y[:, 1] / 1e-4, colors[k])

        ind_t = ind_t_2
        x = get_x(ind_t=ind_t, ind_h=ind_h)
        y = interp.predict_values(x)

        xt_, yt_ = get_pts(xt, yt, 0, ind_h=ind_h, ind_t=ind_t)
        axs[5, 0].plot(xt_, yt_, "o" + colors[k])
        axs[5, 0].plot(lins_M, y[:, 0] / 1e6, colors[k])

        xt_, yt_ = get_pts(xt, yt, 1, ind_h=ind_h, ind_t=ind_t)
```

(continues on next page)

(continued from previous page)

```
        axs[5, 1].plot(xt_, yt_, "o" + colors[k])
        axs[5, 1].plot(lins_M, y[:, 1] / 1e-4, colors[k])

    # ---------------------------------------------------------------------------

    for k in range(2):
        legend_entries = []
        for ind_h in ind_h_list:
            legend_entries.append("h={}".format(val_h[ind_h]))
            legend_entries.append("")

        axs[k, 0].legend(legend_entries)
        axs[k, 1].legend(legend_entries)

        axs[k + 4, 0].legend(legend_entries)
        axs[k + 4, 1].legend(legend_entries)

        legend_entries = []
        for ind_M in ind_M_list:
            legend_entries.append("M={}".format(val_M[ind_M]))
            legend_entries.append("")

        axs[k + 2, 0].legend(legend_entries)
        axs[k + 2, 1].legend(legend_entries)

    plt.show()
```

RMTB

```
from smt.surrogate_models import RMTB
from smt.examples.b777_engine.b777_engine import get_b777_engine, plot_b777_engine

xt, yt, dyt_dxt, xlimits = get_b777_engine()

interp = RMTB(
    num_ctrl_pts=15,
    xlimits=xlimits,
    nonlinear_maxiter=20,
    approx_order=2,
    energy_weight=0e-14,
    regularization_weight=0e-18,
    extrapolate=True,
)
interp.set_training_values(xt, yt)
interp.set_training_derivatives(xt, dyt_dxt[:, :, 0], 0)
interp.set_training_derivatives(xt, dyt_dxt[:, :, 1], 1)
interp.set_training_derivatives(xt, dyt_dxt[:, :, 2], 2)
interp.train()

plot_b777_engine(xt, yt, xlimits, interp)
```

```
___________________________________________________________________________

                                   RMTB
___________________________________________________________________________

 Problem size

      # training points.        : 1056

___________________________________________________________________________

 Training

   Training ...
      Pre-computing matrices ...
         Computing dof2coeff ...
         Computing dof2coeff - done. Time (sec):  0.0000000
         Initializing Hessian ...
         Initializing Hessian - done. Time (sec):  0.0000000
         Computing energy terms ...
         Computing energy terms - done. Time (sec):  0.1749866
         Computing approximation terms ...
         Computing approximation terms - done. Time (sec):  0.0050414
      Pre-computing matrices - done. Time (sec):  0.1800280
      Solving for degrees of freedom ...
         Solving initial startup problem (n=3375) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 4.857178281e+07 2.
↪642628384e+13
               Iteration (num., iy, grad. norm, func.) :   0   0 1.371838165e+05 6.
↪993448074e+09
            Solving for output 0 - done. Time (sec):  0.0556653
            Solving for output 1 ...
               Iteration (num., iy, grad. norm, func.) :   0   1 3.711896708e-01 7.
↪697335516e-04
               Iteration (num., iy, grad. norm, func.) :   0   1 1.374254361e-03 3.
↪512412267e-07
            Solving for output 1 - done. Time (sec):  0.0594563
         Solving initial startup problem (n=3375) - done. Time (sec):  0.1151216
         Solving nonlinear problem (n=3375) ...
            Solving for output 0 ...
               Iteration (num., iy, grad. norm, func.) :   0   0 1.371838165e+05 6.
↪993448074e+09
               Iteration (num., iy, grad. norm, func.) :   0   0 7.296273998e+04 1.
↪952753642e+09
               Iteration (num., iy, grad. norm, func.) :   1   0 4.699413853e+04 5.
↪656526603e+08
               Iteration (num., iy, grad. norm, func.) :   2   0 3.630530299e+04 3.
↪867957232e+08
               Iteration (num., iy, grad. norm, func.) :   3   0 3.132112066e+04 3.
↪751114847e+08
               Iteration (num., iy, grad. norm, func.) :   4   0 2.622497401e+04 3.
↪233995367e+08
```

(continues on next page)

(continued from previous page)

```
               Iteration (num., iy, grad. norm, func.) :   5   0 1.632874586e+04 2.
↪970251901e+08
               Iteration (num., iy, grad. norm, func.) :   6   0 1.945828556e+04 2.
↪644866975e+08
               Iteration (num., iy, grad. norm, func.) :   7   0 7.031298294e+03 2.
↪202296484e+08
               Iteration (num., iy, grad. norm, func.) :   8   0 9.584176762e+03 2.
↪010315328e+08
               Iteration (num., iy, grad. norm, func.) :   9   0 6.066765838e+03 1.
↪861519410e+08
               Iteration (num., iy, grad. norm, func.) :  10   0 8.660248230e+03 1.
↪774578981e+08
               Iteration (num., iy, grad. norm, func.) :  11   0 5.959258886e+03 1.
↪676188728e+08
               Iteration (num., iy, grad. norm, func.) :  12   0 5.420215086e+03 1.
↪611620261e+08
               Iteration (num., iy, grad. norm, func.) :  13   0 3.047173881e+03 1.
↪577535102e+08
               Iteration (num., iy, grad. norm, func.) :  14   0 2.799164640e+03 1.
↪565435585e+08
               Iteration (num., iy, grad. norm, func.) :  15   0 3.613557675e+03 1.
↪539615598e+08
               Iteration (num., iy, grad. norm, func.) :  16   0 2.225395170e+03 1.
↪512405059e+08
               Iteration (num., iy, grad. norm, func.) :  17   0 2.043434052e+03 1.
↪495341032e+08
               Iteration (num., iy, grad. norm, func.) :  18   0 1.794946303e+03 1.
↪489703531e+08
               Iteration (num., iy, grad. norm, func.) :  19   0 1.362610245e+03 1.
↪486885609e+08
            Solving for output 0 - done. Time (sec):  1.1156650
            Solving for output 1 ...
               Iteration (num., iy, grad. norm, func.) :   0   1 1.374254361e-03 3.
↪512412267e-07
               Iteration (num., iy, grad. norm, func.) :   0   1 3.525988536e-04 6.
↪188393994e-08
               Iteration (num., iy, grad. norm, func.) :   1   1 3.057617946e-04 1.
↪809764195e-08
               Iteration (num., iy, grad. norm, func.) :   2   1 1.607604906e-04 8.
↪481761160e-09
               Iteration (num., iy, grad. norm, func.) :   3   1 1.400165707e-04 7.
↪796139756e-09
               Iteration (num., iy, grad. norm, func.) :   4   1 1.120967416e-04 6.
↪716864259e-09
               Iteration (num., iy, grad. norm, func.) :   5   1 1.164592243e-04 5.
↪027554636e-09
               Iteration (num., iy, grad. norm, func.) :   6   1 3.837990633e-05 2.
↪869982182e-09
               Iteration (num., iy, grad. norm, func.) :   7   1 5.592710013e-05 2.
↪076645667e-09
               Iteration (num., iy, grad. norm, func.) :   8   1 2.315187099e-05 1.
↪822951599e-09
```

(continues on next page)

(continued from previous page)

```
              Iteration (num., iy, grad. norm, func.) :   9   1 2.598674619e-05 1.
↪718454630e-09
              Iteration (num., iy, grad. norm, func.) :  10   1 2.476820921e-05 1.
↪578553884e-09
              Iteration (num., iy, grad. norm, func.) :  11   1 2.173672750e-05 1.
↪417305155e-09
              Iteration (num., iy, grad. norm, func.) :  12   1 1.113086815e-05 1.
↪297183916e-09
              Iteration (num., iy, grad. norm, func.) :  13   1 1.971817939e-05 1.
↪259939268e-09
              Iteration (num., iy, grad. norm, func.) :  14   1 1.181193289e-05 1.
↪237157107e-09
              Iteration (num., iy, grad. norm, func.) :  15   1 1.205550947e-05 1.
↪234213169e-09
              Iteration (num., iy, grad. norm, func.) :  16   1 1.274996177e-05 1.
↪207112873e-09
              Iteration (num., iy, grad. norm, func.) :  17   1 1.006884757e-05 1.
↪173920482e-09
              Iteration (num., iy, grad. norm, func.) :  18   1 4.804334068e-06 1.
↪146309226e-09
              Iteration (num., iy, grad. norm, func.) :  19   1 4.607661308e-06 1.
↪143931945e-09
           Solving for output 1 - done. Time (sec):  1.1062367
        Solving nonlinear problem (n=3375) - done. Time (sec):  2.2219017
     Solving for degrees of freedom - done. Time (sec):  2.3370233
  Training - done. Time (sec):  2.5170512
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0060544

   Prediction time/pt. (sec) :  0.0000605


___________________________________________________________________________

 Evaluation
```

(continues on next page)

(continued from previous page)

```
    # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

    # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

    # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

    # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0016241

  Prediction time/pt. (sec) :  0.0000162


___________________________________________________________________________

 Evaluation

    # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________
```

(continues on next page)

(continued from previous page)

```
Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

_____________________________________________________________________________

Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0045381

  Prediction time/pt. (sec) :  0.0000454

_____________________________________________________________________________

Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000968

  Prediction time/pt. (sec) :  0.0000010

_____________________________________________________________________________

Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

_____________________________________________________________________________

Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000
```

(continues on next page)

(continued from previous page)

```
  Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0068035

   Prediction time/pt. (sec) :  0.0000680

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0010366

   Prediction time/pt. (sec) :  0.0000104

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

      # eval points. : 100
```

(continues on next page)

(continued from previous page)

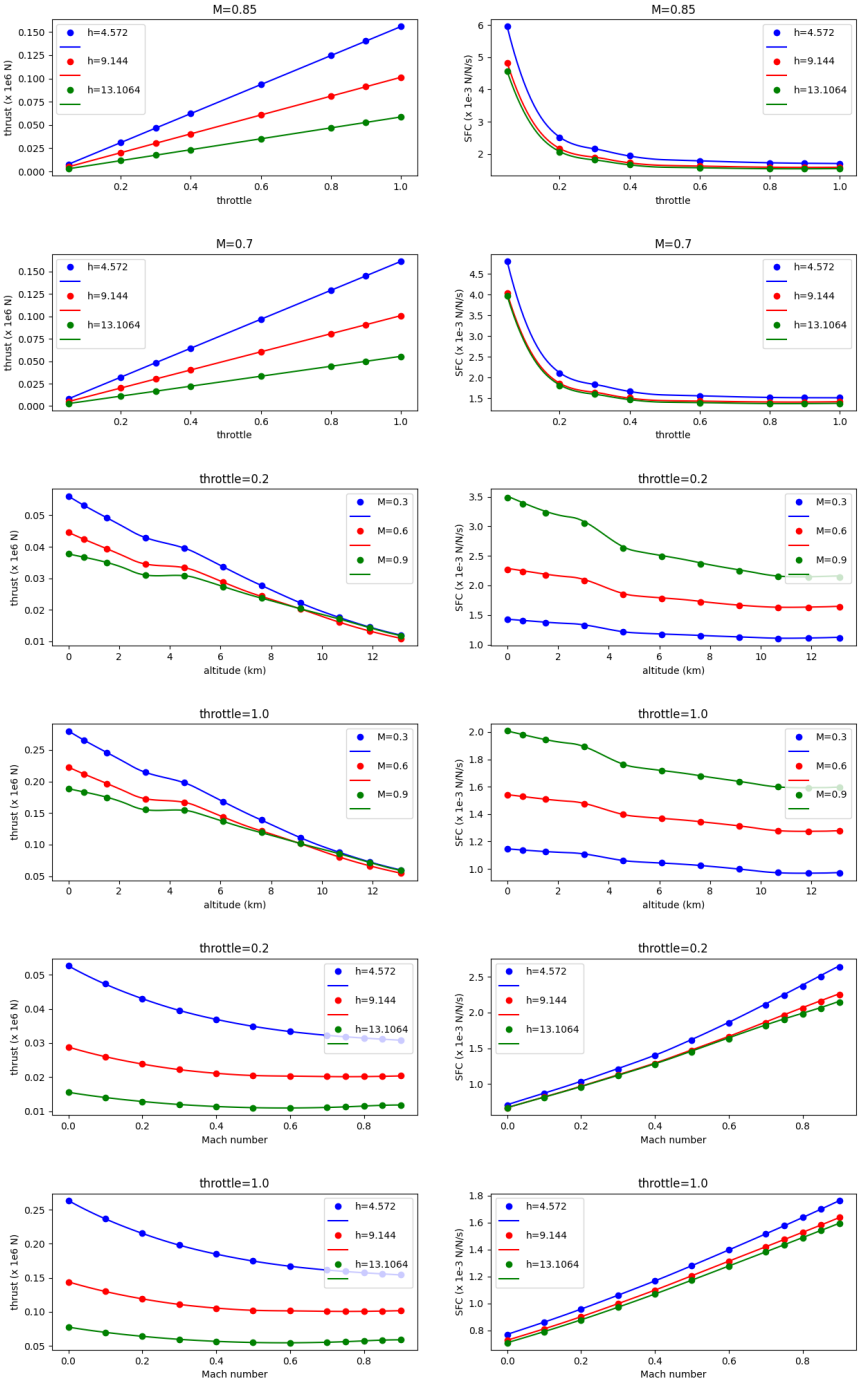
```
   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0021219

   Prediction time/pt. (sec) :  0.0000212
```

RMTC

```
from smt.surrogate_models import RMTC
from smt.examples.b777_engine.b777_engine import get_b777_engine, plot_b777_engine

xt, yt, dyt_dxt, xlimits = get_b777_engine()

interp = RMTC(
    num_elements=6,
    xlimits=xlimits,
    nonlinear_maxiter=20,
    approx_order=2,
    energy_weight=0.0,
    regularization_weight=0.0,
    extrapolate=True,
)
interp.set_training_values(xt, yt)
interp.set_training_derivatives(xt, dyt_dxt[:, :, 0], 0)
interp.set_training_derivatives(xt, dyt_dxt[:, :, 1], 1)
interp.set_training_derivatives(xt, dyt_dxt[:, :, 2], 2)
interp.train()

plot_b777_engine(xt, yt, xlimits, interp)
```

```
___________________________________________________________________________

                                   RMTC
___________________________________________________________________________

 Problem size

      # training points.        : 1056

___________________________________________________________________________
```

(continues on next page)


M=0.85
thrust (x 1e6 N)
throttle
h=4.572
h=9.144
h=13.1064
M=0.85
SFC (x 1e-3 N/N/s)
throttle
h=4.572
h=9.144
h=13.1064
M=0.7
thrust (x 1e6 N)
throttle
h=4.572
h=9.144
h=13.1064
M=0.7
SFC (x 1e-3 N/N/s)
throttle
h=4.572
h=9.144
h=13.1064
throttle=0.2
thrust (x 1e6 N)
altitude (km)
M=0.3
M=0.6
M=0.9
throttle=0.2
SFC (x 1e-3 N/N/s)
altitude (km)
M=0.3
M=0.6
M=0.9
throttle=1.0
thrust (x 1e6 N)
altitude (km)
M=0.3
M=0.6
M=0.9
throttle=1.0
SFC (x 1e-3 N/N/s)
altitude (km)
M=0.3
M=0.6
M=0.9
throttle=0.2
thrust (x 1e6 N)
Mach number
h=4.572
h=9.144
h=13.1064
throttle=0.2
SFC (x 1e-3 N/N/s)
Mach number
h=4.572
h=9.144
h=13.1064
throttle=1.0
thrust (x 1e6 N)
Mach number
h=4.572
h=9.144
h=13.1064
throttle=1.0
SFC (x 1e-3 N/N/s)
Mach number
h=4.572
h=9.144
h=13.1064

(continued from previous page)

```
Training

  Training ...
     Pre-computing matrices ...
        Computing dof2coeff ...
        Computing dof2coeff - done. Time (sec):  0.0151327
        Initializing Hessian ...
        Initializing Hessian - done. Time (sec):  0.0000000
        Computing energy terms ...
        Computing energy terms - done. Time (sec):  0.1215806
        Computing approximation terms ...
        Computing approximation terms - done. Time (sec):  0.0404363
     Pre-computing matrices - done. Time (sec):  0.1771495
     Solving for degrees of freedom ...
        Solving initial startup problem (n=2744) ...
           Solving for output 0 ...
              Iteration (num., iy, grad. norm, func.) :   0   0 7.864862172e+07 2.
↪642628384e+13
              Iteration (num., iy, grad. norm, func.) :   0   0 1.954376733e+05 2.
↪069307906e+09
           Solving for output 0 - done. Time (sec):  0.1294446
           Solving for output 1 ...
              Iteration (num., iy, grad. norm, func.) :   0   1 8.095040141e-01 7.
↪697335516e-04
              Iteration (num., iy, grad. norm, func.) :   0   1 1.232503686e-03 1.
↪322818515e-07
           Solving for output 1 - done. Time (sec):  0.1287684
        Solving initial startup problem (n=2744) - done. Time (sec):  0.2582130
        Solving nonlinear problem (n=2744) ...
           Solving for output 0 ...
              Iteration (num., iy, grad. norm, func.) :   0   0 1.954376733e+05 2.
↪069307906e+09
              Iteration (num., iy, grad. norm, func.) :   0   0 3.688989036e+04 4.
↪210539534e+08
              Iteration (num., iy, grad. norm, func.) :   1   0 1.711008392e+04 3.
↪530483452e+08
              Iteration (num., iy, grad. norm, func.) :   2   0 1.922847114e+04 3.
↪504226099e+08
              Iteration (num., iy, grad. norm, func.) :   3   0 9.537041405e+03 3.
↪373561783e+08
              Iteration (num., iy, grad. norm, func.) :   4   0 4.760722218e+03 3.
↪327296410e+08
              Iteration (num., iy, grad. norm, func.) :   5   0 6.376817973e+03 3.
↪320587914e+08
              Iteration (num., iy, grad. norm, func.) :   6   0 2.785458727e+03 3.
↪312772871e+08
              Iteration (num., iy, grad. norm, func.) :   7   0 2.159194888e+03 3.
↪307195890e+08
              Iteration (num., iy, grad. norm, func.) :   8   0 1.756134897e+03 3.
↪304676354e+08
              Iteration (num., iy, grad. norm, func.) :   9   0 2.544893553e+03 3.
```

(continues on next page)

(continued from previous page)

```
↪303514392e+08
               Iteration (num., iy, grad. norm, func.) :  10   0 9.687040825e+02 3.
↪302102085e+08
               Iteration (num., iy, grad. norm, func.) :  11   0 1.529434541e+03 3.
↪301310381e+08
               Iteration (num., iy, grad. norm, func.) :  12   0 8.575442103e+02 3.
↪300046689e+08
               Iteration (num., iy, grad. norm, func.) :  13   0 7.946674618e+02 3.
↪299019198e+08
               Iteration (num., iy, grad. norm, func.) :  14   0 4.138160638e+02 3.
↪298361385e+08
               Iteration (num., iy, grad. norm, func.) :  15   0 3.344866005e+02 3.
↪298229530e+08
               Iteration (num., iy, grad. norm, func.) :  16   0 4.791739707e+02 3.
↪298135435e+08
               Iteration (num., iy, grad. norm, func.) :  17   0 7.683854023e+02 3.
↪298069892e+08
               Iteration (num., iy, grad. norm, func.) :  18   0 4.087822635e+02 3.
↪298060925e+08
               Iteration (num., iy, grad. norm, func.) :  19   0 2.867885276e+02 3.
↪298024065e+08
            Solving for output 0 - done. Time (sec):  2.5238924
            Solving for output 1 ...
               Iteration (num., iy, grad. norm, func.) :   0   1 1.232503686e-03 1.
↪322818515e-07
               Iteration (num., iy, grad. norm, func.) :   0   1 4.036053441e-04 9.
↪595113849e-09
               Iteration (num., iy, grad. norm, func.) :   1   1 2.706482615e-04 7.
↪876088023e-09
               Iteration (num., iy, grad. norm, func.) :   2   1 2.260119896e-04 6.
↪102248425e-09
               Iteration (num., iy, grad. norm, func.) :   3   1 1.174679565e-04 4.
↪321455879e-09
               Iteration (num., iy, grad. norm, func.) :   4   1 1.073558503e-04 4.
↪060995795e-09
               Iteration (num., iy, grad. norm, func.) :   5   1 6.275682902e-05 3.
↪739333496e-09
               Iteration (num., iy, grad. norm, func.) :   6   1 5.607076749e-05 3.
↪359307617e-09
               Iteration (num., iy, grad. norm, func.) :   7   1 3.901842383e-05 3.
↪200980932e-09
               Iteration (num., iy, grad. norm, func.) :   8   1 4.898053749e-05 3.
↪121087196e-09
               Iteration (num., iy, grad. norm, func.) :   9   1 2.789456822e-05 3.
↪062451544e-09
               Iteration (num., iy, grad. norm, func.) :  10   1 2.227200280e-05 3.
↪044681794e-09
               Iteration (num., iy, grad. norm, func.) :  11   1 3.096887425e-05 3.
↪026565623e-09
               Iteration (num., iy, grad. norm, func.) :  12   1 1.606775960e-05 2.
↪993510820e-09
               Iteration (num., iy, grad. norm, func.) :  13   1 1.554076288e-05 2.
```

(continues on next page)

(continued from previous page)

```
↪978354868e-09
              Iteration (num., iy, grad. norm, func.) :  14   1 1.041250866e-05 2.
↪953574603e-09
              Iteration (num., iy, grad. norm, func.) :  15   1 1.155923771e-05 2.
↪935360717e-09
              Iteration (num., iy, grad. norm, func.) :  16   1 9.466047104e-06 2.
↪927068596e-09
              Iteration (num., iy, grad. norm, func.) :  17   1 9.185257136e-06 2.
↪924961044e-09
              Iteration (num., iy, grad. norm, func.) :  18   1 7.396047208e-06 2.
↪924123759e-09
              Iteration (num., iy, grad. norm, func.) :  19   1 1.605557201e-05 2.
↪921915424e-09
           Solving for output 1 - done. Time (sec):  2.5130031
        Solving nonlinear problem (n=2744) - done. Time (sec):  5.0368955
     Solving for degrees of freedom - done. Time (sec):  5.2951086
  Training - done. Time (sec):  5.4749968
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0009725

   Prediction time/pt. (sec) :  0.0000097

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0010002

   Prediction time/pt. (sec) :  0.0000100

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0011082

   Prediction time/pt. (sec) :  0.0000111

___________________________________________________________________________
```

(continues on next page)

(continued from previous page)

```
Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

_____________________________________________________________________________

Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0021155

  Prediction time/pt. (sec) :  0.0000212

_____________________________________________________________________________

Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000

_____________________________________________________________________________

Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0061014

  Prediction time/pt. (sec) :  0.0000610

_____________________________________________________________________________

Evaluation

     # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0021162

  Prediction time/pt. (sec) :  0.0000212
```

(continues on next page)

(continued from previous page)

```
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0046735

   Prediction time/pt. (sec) :  0.0000467

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000

___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
```

(continues on next page)

(continued from previous page)

```
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

      # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0021138

  Prediction time/pt. (sec) :  0.0000211


___________________________________________________________________________

 Evaluation

      # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0026135

  Prediction time/pt. (sec) :  0.0000261


___________________________________________________________________________

 Evaluation

      # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

      # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0055439

  Prediction time/pt. (sec) :  0.0000554


___________________________________________________________________________

 Evaluation
```

(continues on next page)

(continued from previous page)

```
    # eval points. : 100

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000
```

3.5.4 Learning Airfoil Parameters

This is a tutorial to determine the aerodynamic coefficients of a given airfoil using GENN in SMT (other models could be used as well). The obtained surrogate model can be used to give predictions for certain Mach numbers, angles of attack and the aerodynamic coefficients. These calculations can be really useful in case of an airfoil shape optimization. The input parameters uses the airfoil Camber and Thickness mode shapes.

- Inputs: Airfoil Camber and Thickness mode shapes, Mach, alpha
- Outputs (options): cd, cl, cm

In this test case, we will be predicting only the Cd coefficient. However, the other databases for the prediction of the other terms are available in the same repository. Bouhlels mSANN uses the information contained in the paper[1] to determine the airfoil's mode shapes. Moreover, in mSANN a deep neural network is used to predict the Cd parameter of a given parametrized airfoil. Therefore, in this tutorial, we reproduce the paper[2] using the Gradient-Enhanced Neural Networks (GENN) from SMT.

Briefly explaining how mSANN generates the mode shapes of a given airfoil:

1. Using inverse distance weighting (IDW) to interpolate the surface function of each airfoil.
2. Then applying singular value decomposition (SVD) to reduce the number of variables that define the airfoil geometry. It includes a total of 14 airfoil modes (seven for camber and seven for thickness).
3. Totally 16 input variables, two flow conditions of Mach number (0.3 to 0.6) and the angle of attack (2 degrees to 6 degrees) plus 14 shape coefficients.
4. The output airfoil aerodynamic force coefficients and their respective gradients are computed using ADflow, which solves the RANS equations with a Spalart-Allmaras turbulence model.

References

Implementation

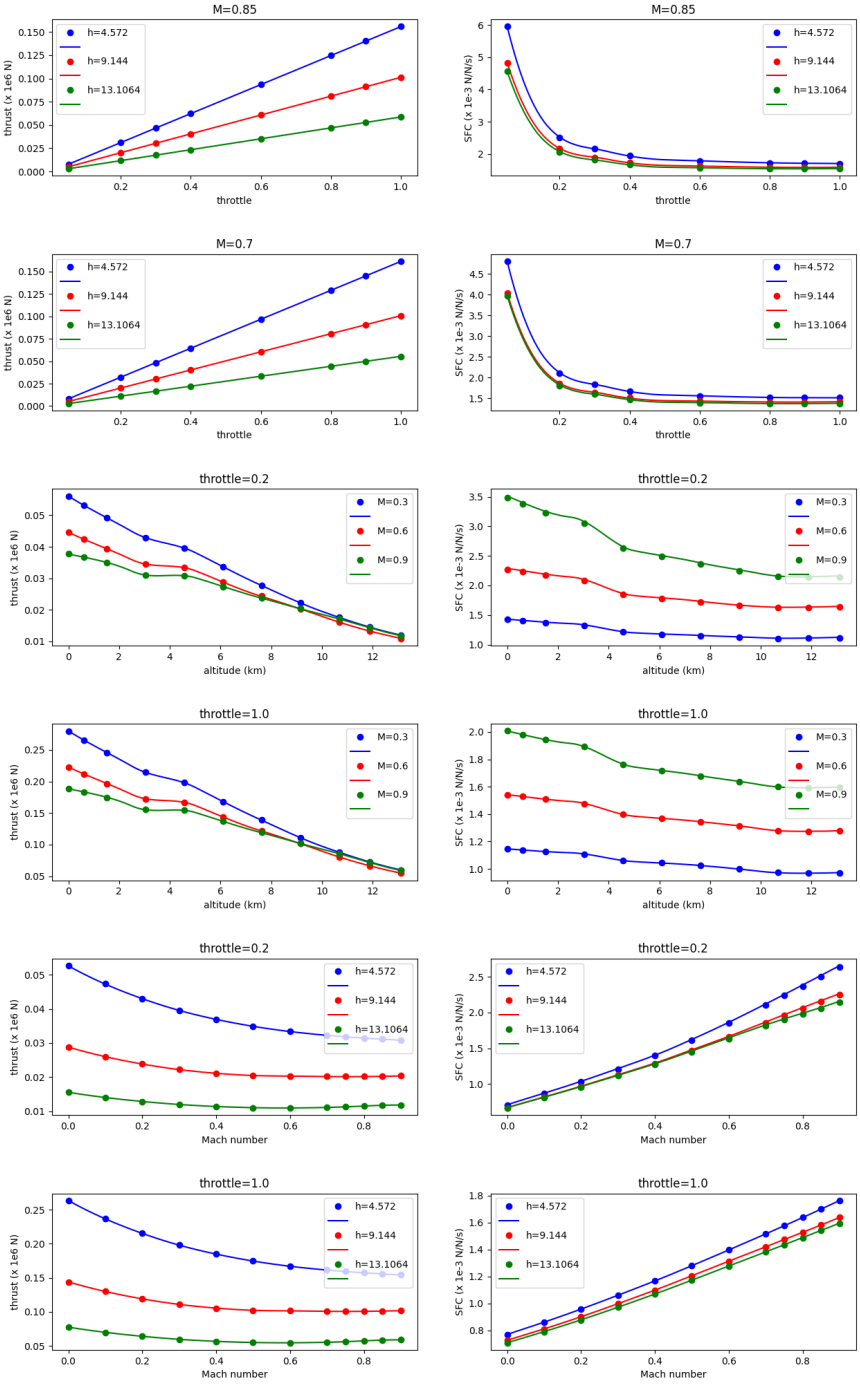
Utilities

```
import os
import numpy as np
import csv

WORKDIR = os.path.dirname(os.path.abspath(__file__))
```

(continues on next page)

[1] Bouhlel, M. A., He, S., & Martins, J. R. (2020). Scalable gradient–enhanced artificial neural networks for airfoil shape design in the subsonic and transonic regimes. Structural and Multidisciplinary Optimization, 61(4), 1363-1376.

[2] Bouhlel, M. A., He, S., and Martins, J. R. R. A., mSANN Model Benchmarks, Mendeley Data, 2019. https://doi.org/10. 17632/ngpd634smf.1.


M=0.85
thrust (x 1e6 N)
throttle
h=4.572
h=9.144
h=13.1064
M=0.85
SFC (x 1e-3 N/N/s)
throttle
h=4.572
h=9.144
h=13.1064
M=0.7
thrust (x 1e6 N)
throttle
h=4.572
h=9.144
h=13.1064
M=0.7
SFC (x 1e-3 N/N/s)
throttle
h=4.572
h=9.144
h=13.1064
throttle=0.2
thrust (x 1e6 N)
altitude (km)
M=0.3
M=0.6
M=0.9
throttle=0.2
SFC (x 1e-3 N/N/s)
altitude (km)
M=0.3
M=0.6
M=0.9
throttle=1.0
thrust (x 1e6 N)
altitude (km)
M=0.3
M=0.6
M=0.9
throttle=1.0
SFC (x 1e-3 N/N/s)
altitude (km)
M=0.3
M=0.6
M=0.9
throttle=0.2
thrust (x 1e6 N)
Mach number
h=4.572
h=9.144
h=13.1064
throttle=0.2
SFC (x 1e-3 N/N/s)
Mach number
h=4.572
h=9.144
h=13.1064
throttle=1.0
thrust (x 1e6 N)
Mach number
h=4.572
h=9.144
h=13.1064
throttle=1.0
SFC (x 1e-3 N/N/s)
Mach number
h=4.572
h=9.144
h=13.1064

(continued from previous page)

```
def load_NACA4412_modeshapes():
    return np.loadtxt(open(os.path.join(WORKDIR, "modes_NACA4412_ct.txt")))


def load_cd_training_data():
    with open(os.path.join(WORKDIR, "cd_x_y.csv")) as file:
        reader = csv.reader(file, delimiter=";")
        values = np.array(list(reader), dtype=np.float32)
        dim_values = values.shape
        x = values[:, : dim_values[1] - 1]
        y = values[:, -1]
    with open(os.path.join(WORKDIR, "cd_dy.csv")) as file:
        reader = csv.reader(file, delimiter=";")
        dy = np.array(list(reader), dtype=np.float32)
    return x, y, dy


def plot_predictions(airfoil_modeshapes, Ma, cd_model):
    import matplotlib

    matplotlib.use("Agg")
    import matplotlib.pyplot as plt

    # alpha is linearily distributed over the range of -1 to 7 degrees
    # while Ma is kept constant
    inputs = np.zeros(shape=(1, 15))
    inputs[0, :14] = airfoil_modeshapes
    inputs[0, -1] = Ma
    inputs = np.tile(inputs, (50, 1))

    alpha = np.atleast_2d([-1 + 0.16 * i for i in range(50)]).T

    inputs = np.concatenate((inputs, alpha), axis=1)

    # Predict Cd
    cd_pred = cd_model.predict_values(inputs)

    # Load ADflow Cd reference
    with open(os.path.join(WORKDIR, "NACA4412-ADflow-alpha-cd.csv")) as file:
        reader = csv.reader(file, delimiter=" ")
        cd_adflow = np.array(list(reader)[1:], dtype=np.float32)

    plt.plot(alpha, cd_pred)
    plt.plot(cd_adflow[:, 0], cd_adflow[:, 1])
    plt.grid(True)
    plt.legend(["Surrogate", "ADflow"])
    plt.title("Drag coefficient")
    plt.xlabel("Alpha")
    plt.ylabel("Cd")
    plt.show()
```

Main

```
"""
Predicting Airfoil Aerodynamics through data by Raul Carreira Rufato and Prof. Joseph␣
↪Morlier
"""

import os
import numpy as np
import csv

from smt.examples.airfoil_parameters.learning_airfoil_parameters import (
    load_cd_training_data,
    load_NACA4412_modeshapes,
    plot_predictions,
)
from sklearn.model_selection import train_test_split
from smt.surrogate_models.genn import GENN, load_smt_data

x, y, dy = load_cd_training_data()

# splitting the dataset
x_train, x_test, y_train, y_test, dy_train, dy_test = train_test_split(
    x, y, dy, train_size=0.8
)
# building and training the GENN
genn = GENN(print_global=False)
# learning rate that controls optimizer step size
genn.options["alpha"] = 0.001
# lambd = 0. = no regularization, lambd > 0 = regularization
genn.options["lambd"] = 0.1
# gamma = 0. = no grad-enhancement, gamma > 0 = grad-enhancement
genn.options["gamma"] = 1.0
# number of hidden layers
genn.options["deep"] = 2
# number of nodes per hidden layer
genn.options["wide"] = 6
# used to divide data into training batches (use for large data sets)
genn.options["mini_batch_size"] = 256
# number of passes through data
genn.options["num_epochs"] = 5
# number of optimizer iterations per mini-batch
genn.options["num_iterations"] = 10
# print output (or not)
genn.options["is_print"] = False
# convenience function to read in data that is in SMT format
load_smt_data(genn, x_train, y_train, dy_train)

genn.train()

## non-API function to plot training history (to check convergence)
# genn.plot_training_history()
```

(continues on next page)

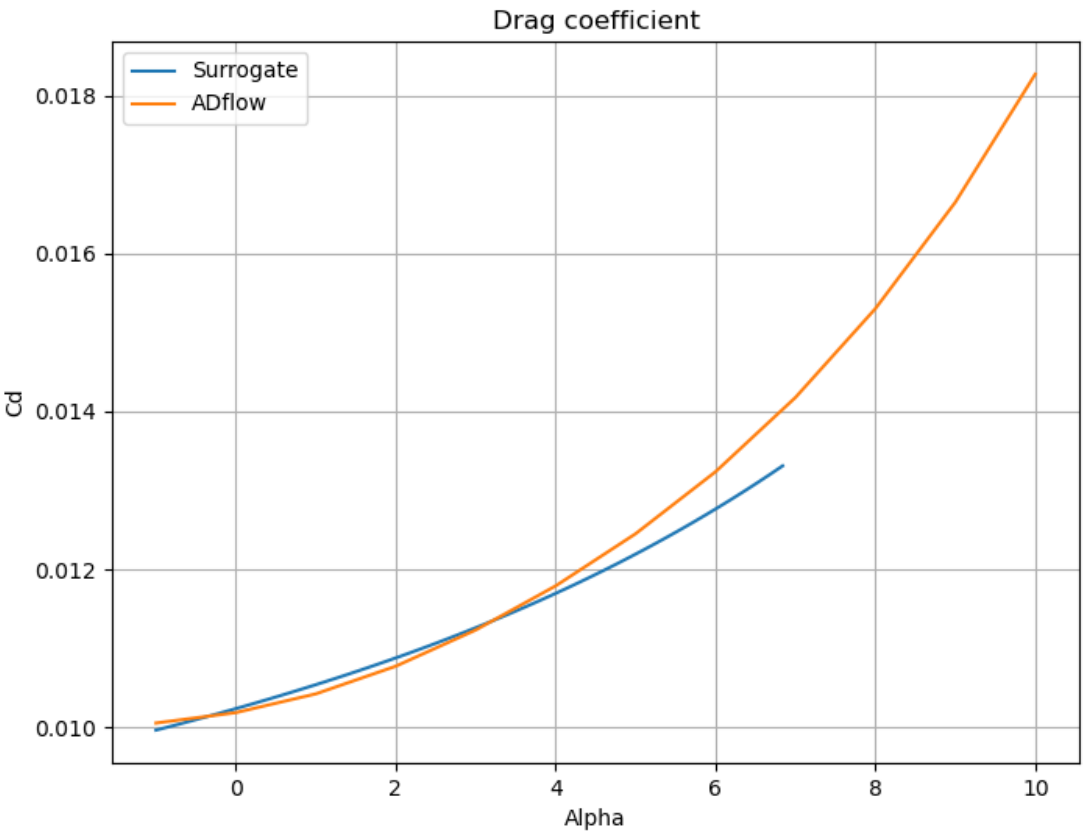
(continued from previous page)

```
## non-API function to check accuracy of regression
# genn.goodness_of_fit(x_test, y_test, dy_test)

# API function to predict values at new (unseen) points
y_pred = genn.predict_values(x_test)

# Now we will use the trained model to make a prediction with a not-learned form.
# Example Prediction for NACA4412.
# Airfoil mode shapes should be determined according to Bouhlel, M.A., He, S., and␣
↪Martins,
# J.R.R.A., mSANN Model Benchmarks, Mendeley Data, 2019. https://doi.org/10.17632/
↪ngpd634smf.1
# Comparison of results with Adflow software for an alpha range from -1 to 7 degrees. Re␣
↪= 3000000
airfoil_modeshapes = load_NACA4412_modeshapes()
Ma = 0.3
alpha = 0

# input in neural network is created out of airfoil mode shapes, Mach number and alpha
# airfoil_modeshapes: computed mode_shapes of random airfol geometry with parameterise_
↪airfoil
# Ma: desired Mach number for evaluation in range [0.3,0.6]
# alpha: scalar in range [-1, 6]
input = np.zeros(shape=(1, 16))
input[0, :14] = airfoil_modeshapes
input[0, 14] = Ma
input[0, -1] = alpha

# prediction
cd_pred = genn.predict_values(input)
print("Drag coefficient prediction (cd): ", cd_pred[0, 0])

plot_predictions(airfoil_modeshapes, Ma, genn)
```

```
Drag coefficient prediction (cd):  0.01019668488197359
```

3.6 Applications

Applications implement a set of methods using surrogate models provided by SMT. Three methods are available:


Drag coefficient
Surrogate
ADflow
0.018
0.016
0.014
0.012
0.010
Cd
0
2
4
6
8
10
Alpha

3.6.1 Mixture of experts (MOE)

Mixture of experts aims at increasing the accuracy of a function approximation by replacing a single global model by a weighted sum of local models (experts). It is based on a partition of the problem domain into several subdomains via clustering algorithms followed by a local expert training on each subdomain.

A general introduction about the mixture of experts can be found in[1] and a first application with generalized linear models in[2].

SMT MOE combines surrogate models implemented in SMT to build a new surrogate model. The method is expected to improve the accuracy for functions with some of the following characteristics: heterogeneous behaviour depending on the region of the input space, flat and steep regions, first and zero order discontinuities.

The MOE method strongly relies on the Expectation-Maximization (EM) algorithm for Gaussian mixture models (GMM). With an aim of regression, the different steps are the following:

1. Clustering: the inputs are clustered together with their output values by means of parameter estimation of the joint distribution.
2. Local expert training: A local expert is then built (linear, quadratic, cubic, radial basis functions, or different forms of kriging) on each cluster
3. Recombination: all the local experts are finally combined using the Gaussian mixture model parameters found by the EM algorithm to get a global model.

When local models y_i are known, the global model would be:

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{K} \mathbb{P}(\kappa = i | X = \mathbf{x}) \hat{y_i}(\mathbf{x}) (3.1)$$

which is the classical probability expression of mixture of experts.

In this equation, K is the number of Gaussian components, $\mathbb{P}(\kappa = i|X = \mathbf{x})$, denoted by gating network, is the probability to lie in cluster i knowing that $X = \mathbf{x}$ and $\hat{y_i}$ is the local expert built on cluster i.

This equation leads to two different approximation models depending on the computation of $\mathbb{P}(\kappa = i|X = \mathbf{x})$.

- When choosing the Gaussian laws to compute this quantity, the equation leads to a *smooth model* that smoothly recombine different local experts.
- If $\mathbb{P}(\kappa = i|X = \mathbf{x})$ is computed as characteristic functions of clusters (being equal to 0 or 1) this leads to a *discontinuous approximation model*.

More details can be found in[3] and[4].

[1] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. The elements of statistical learning, volume 1. Springer series in statistics Springer, Berlin, 2008.

[2] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. Neural computation, 6(2) :181–214, 1994.

[3] Dimitri Bettebghor, Nathalie Bartoli, Stephane Grihon, Joseph Morlier, and Manuel Samuelides. Surrogate modeling approximation using a mixture of experts based on EM joint estimation. Structural and Multidisciplinary Optimization, 43(2) :243–259, 2011. 10.1007/s00158-010-0554-2.

[4] Rhea P. Liem, Charles A. Mader, and Joaquim R. R. A. Martins. Surrogate models and mixtures of experts in aerodynamic performance prediction for mission analysis. Aerospace Science and Technology, 43 :126–151, 2015.

References

Implementation Notes

Beside the main class *MOE*, one can also use the *MOESurrogateModel* class which adapts MOE as a *SurrogateModel* implementing the Surrogate Model API (see *Surrogate Model API*).

Usage

Example 1

```
import numpy as np
from smt.applications import MOE
from smt.sampling_methods import FullFactorial
import matplotlib.pyplot as plt

ndim = 1
nt = 35

def function_test_1d(x):
    import numpy as np  # Note: only required by SMT doc testing toolchain

    x = np.reshape(x, (-1,))
    y = np.zeros(x.shape)
    y[x < 0.4] = x[x < 0.4] ** 2
    y[(x >= 0.4) & (x < 0.8)] = 3 * x[(x >= 0.4) & (x < 0.8)] + 1
    y[x >= 0.8] = np.sin(10 * x[x >= 0.8])
    return y.reshape((-1, 1))

x = np.linspace(0, 1, 100)
ytrue = function_test_1d(x)

# Training data
sampling = FullFactorial(xlimits=np.array([[0, 1]]), clip=True)
np.random.seed(0)
xt = sampling(nt)
yt = function_test_1d(xt)

# Mixture of experts
print("MOE Experts: ", MOE.AVAILABLE_EXPERTS)

# MOE1: Find the best surrogate model on the whole domain
moe1 = MOE(n_clusters=1)
print("MOE1 enabled experts: ", moe1.enabled_experts)
moe1.set_training_values(xt, yt)
moe1.train()
y_moe1 = moe1.predict_values(x)

# MOE2: Set nb of cluster with just KRG, LS and IDW surrogate models
moe2 = MOE(smooth_recombination=False, n_clusters=3, allow=["KRG", "LS", "IDW"])
print("MOE2 enabled experts: ", moe2.enabled_experts)
moe2.set_training_values(xt, yt)
```

(continues on next page)

(continued from previous page)

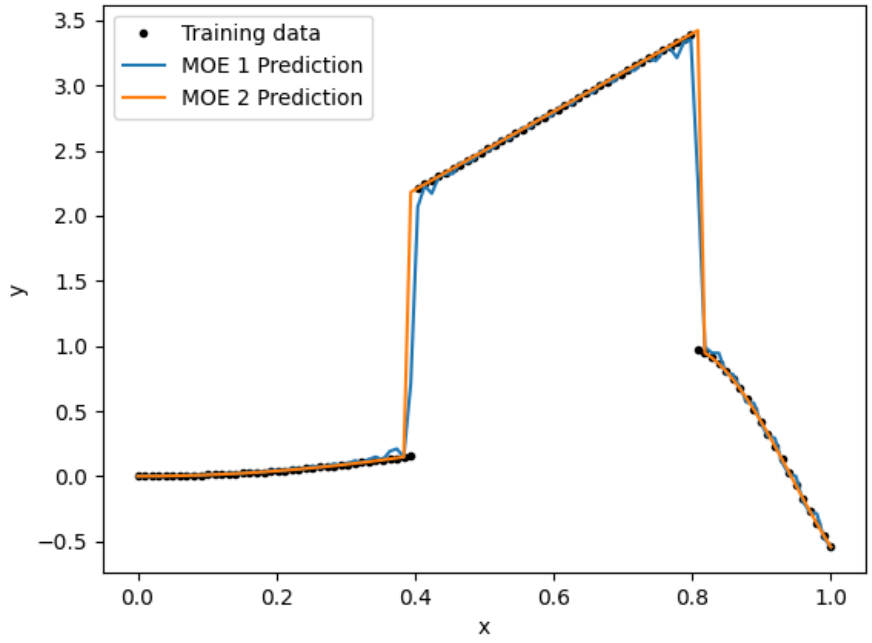
```
moe2.train()
y_moe2 = moe2.predict_values(x)

fig, axs = plt.subplots(1)
axs.plot(x, ytrue, ".", color="black")
axs.plot(x, y_moe1)
axs.plot(x, y_moe2)
axs.set_xlabel("x")
axs.set_ylabel("y")
axs.legend(["Training data", "MOE 1 Prediction", "MOE 2 Prediction"])

plt.show()
```

```
MOE Experts:  ['KRG', 'KPLS', 'KPLSK', 'LS', 'QP', 'RBF', 'IDW', 'RMTB', 'RMTC']
MOE1 enabled experts:  ['KRG', 'LS', 'QP', 'KPLS', 'KPLSK', 'RBF', 'RMTC', 'RMTB', 'IDW']
Kriging 1.826220788586843
LS 3.2926869662406073
QP 2.5146162179490443
KPLS 1.826220788586788
KPLSK 1.8262207885868322
RBF 1.812172061536789
RMTC 0.7218170326354906
RMTB 0.7262718847440127
IDW 0.5249547613845662
Best expert = IDW
MOE2 enabled experts:  ['KRG', 'LS', 'IDW']
Kriging 6.861609623936975e-07
LS 0.026711170330602847
IDW 0.0071340980233521424
Best expert = Kriging
Kriging 7.273943367647347e-07
LS 0.0
IDW 0.13389238051164282
Best expert = LS
Kriging 0.003454948882642195
LS 0.14196607571546527
IDW 0.22651705336130568
Best expert = Kriging
```

Example 2

```
import numpy as np
from smt.applications import MOE
from smt.problems import LpNorm
from smt.sampling_methods import FullFactorial

import sklearn
import matplotlib.pyplot as plt
from matplotlib import colors
from mpl_toolkits.mplot3d import Axes3D
```

(continues on next page)



(continued from previous page)

```
ndim = 2
nt = 200
ne = 200

# Problem: L1 norm (dimension 2)
prob = LpNorm(ndim=ndim)

# Training data
sampling = FullFactorial(xlimits=prob.xlimits, clip=True)
np.random.seed(0)
xt = sampling(nt)
yt = prob(xt)

# Mixture of experts
print("MOE Experts: ", MOE.AVAILABLE_EXPERTS)

moe = MOE(smooth_recombination=True, n_clusters=5, deny=["RMTB", "KPLSK"])
print("Enabled Experts: ", moe.enabled_experts)
moe.set_training_values(xt, yt)
moe.train()

# Validation data
np.random.seed(1)
xe = sampling(ne)
ye = prob(xe)
```

(continues on next page)

(continued from previous page)

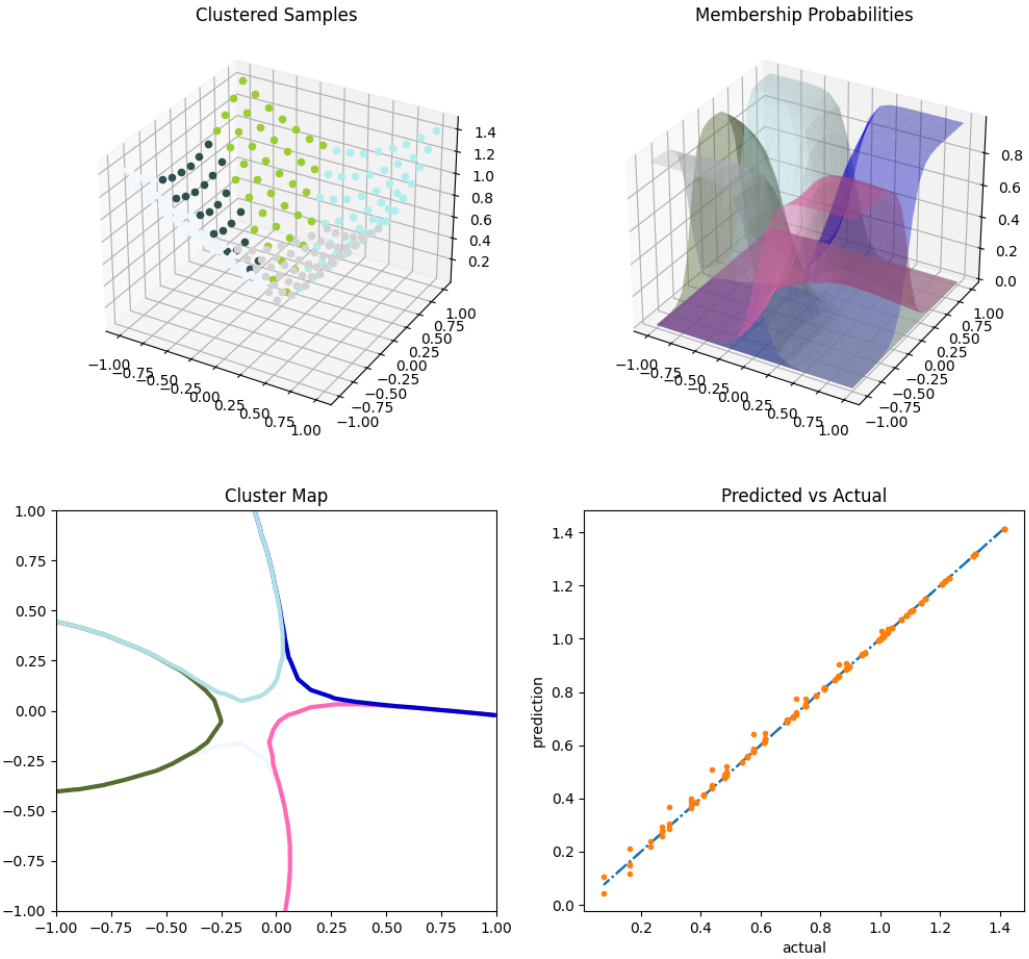
```
# Prediction
y = moe.predict_values(xe)
fig = plt.figure(1)
fig.set_size_inches(12, 11)

# Cluster display
colors_ = list(colors.cnames.items())
GMM = moe.cluster
weight = GMM.weights_
mean = GMM.means_
if sklearn.__version__ < "0.20.0":
    cov = GMM.covars_
else:
    cov = GMM.covariances_
prob_ = moe._proba_cluster(xt)
sort = np.apply_along_axis(np.argmax, 1, prob_)

xlim = prob.xlimits
x0 = np.linspace(xlim[0, 0], xlim[0, 1], 20)
x1 = np.linspace(xlim[1, 0], xlim[1, 1], 20)
xv, yv = np.meshgrid(x0, x1)
x = np.array(list(zip(xv.reshape((-1,)), yv.reshape((-1,)))))
prob = moe._proba_cluster(x)

plt.subplot(221, projection="3d")
ax = plt.gca()
for i in range(len(sort)):
    color = colors_[int(((len(colors_) - 1) / sort.max()) * sort[i])][0]
    ax.scatter(xt[i][0], xt[i][1], yt[i], c=color)
plt.title("Clustered Samples")

plt.subplot(222, projection="3d")
ax = plt.gca()
for i in range(len(weight)):
    color = colors_[int(((len(colors_) - 1) / len(weight)) * i)][0]
    ax.plot_trisurf(
        x[:, 0], x[:, 1], prob[:, i], alpha=0.4, linewidth=0, color=color
    )
plt.title("Membership Probabilities")

plt.subplot(223)
for i in range(len(weight)):
    color = colors_[int(((len(colors_) - 1) / len(weight)) * i)][0]
    plt.tricontour(x[:, 0], x[:, 1], prob[:, i], 1, colors=color, linewidths=3)
plt.title("Cluster Map")

plt.subplot(224)
plt.plot(ye, ye, "-.")
plt.plot(ye, y, ".")
plt.xlabel("actual")
plt.ylabel("prediction")
plt.title("Predicted vs Actual")
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

```
MOE Experts:  ['KRG', 'KPLS', 'KPLSK', 'LS', 'QP', 'RBF', 'IDW', 'RMTB', 'RMTC']
Enabled Experts:  ['KRG', 'LS', 'QP', 'KPLS', 'RBF', 'RMTC', 'IDW']
Kriging 0.006073780112481599
LS 0.08289865595964077
QP 0.045843092689508834
KPLS 0.006216616321294433
RBF 0.004776812673562382
RMTC 0.04819752624954801
IDW 0.2153214768235453
Best expert = RBF
Kriging 0.000962965140254022
LS 0.07083661007512033
QP 0.008991392622159056
KPLS 0.004204309000137218
RBF 0.0009524104524760142
RMTC 0.045129237882873194
IDW 0.23160779607143606
Best expert = RBF
Kriging 0.005713000024284606
LS 0.1733556444157393
QP 0.036110181568461915
KPLS 0.005020198068330178
RBF 0.009878390723819444
RMTC 0.045064765245918334
IDW 0.17165783069813756
Best expert = KPLS
Kriging 0.0011324284387790164
LS 0.23198702766800058
QP 0.04876018506588711
KPLS 0.004686229471742169
RBF 0.005839070324091541
RMTC 0.05208459151486497
IDW 0.21110329999279054
Best expert = Kriging
Kriging 0.0008485909594676008
LS 0.06682187881202373
QP 0.02156744245354161
KPLS 0.0008196836307230437
RBF 0.0008827675618174093
RMTC 0.023947896251418598
IDW 0.15640624530290065
Best expert = KPLS
```


Clustered Samples
Membership Probabilities
Cluster Map
Predicted vs Actual
prediction
actual

Options

Table 27: List of options

Option	Default	Acceptable values	Acceptable types	Description
xt	None	None	['ndarray']	Training inputs
yt	None	None	['ndarray']	Training outputs
ct	None	None	['ndarray']	Training derivative outputs used for clustering
xtest	None	None	['ndarray']	Test inputs
ytest	None	None	['ndarray']	Test outputs
n_clusters	2	None	['int']	Number of clusters
smooth_recombination	True	None	['bool']	Continuous cluster transition
heaviside_optimization	False	None	['bool']	Optimize Heaviside scaling factor when smooth recombination is used
derivatives_support	False	None	['bool']	Use only experts that support derivatives prediction
variances_support	False	None	['bool']	Use only experts that support variance prediction
allow	[]	None	None	Names of allowed experts to be possibly part of the mixture. Empty list corresponds to all surrogates allowed.
deny	[]	None	None	Names of forbidden experts

3.6.2 Variable-fidelity modeling (VFM)

VFM is a variable-fidelity modeling method which can use additive, multiplicative, or hybride bridge functions. SMT proposes only additive and multiplicative options.

In the additive method, high- and low-fidelity models, $y_{\text{high}}(\mathbf{x})$ and $y_{\text{low}}(\mathbf{x})$, are calibrated by adding the low-fidelity model to a function $\gamma(\mathbf{x})$, also called bridge function

$$y_{\text{high}}(\mathbf{x}) = y_{\text{low}}(\mathbf{x}) + \gamma(\mathbf{x})$$

The additive bridge function was developed by Lewis and Nash[1].

In the same way, the multiplicative bridge function is defined by

$$\gamma(\mathbf{x}) = \frac{y_{\text{high}}(\mathbf{x})}{y_{\text{low}}(\mathbf{x})}$$

However, the multiplicative bridge function may cause problems when one of the sampled values of the low-fidelity model is close to zero.

After the unknown bridge function γ and low-fidelity model y_{low} have been approximated with $\hat{\gamma}$ and $\hat{y}_{\text{low}}$, respectively, the response of the high-fidelity model is obtained

[1] Lewis, R. M. and Nash, S. G., A multigrid approach to the optimization of systems governed by differential equations. AIAA Paper 2000-4890, 2000.

Usage

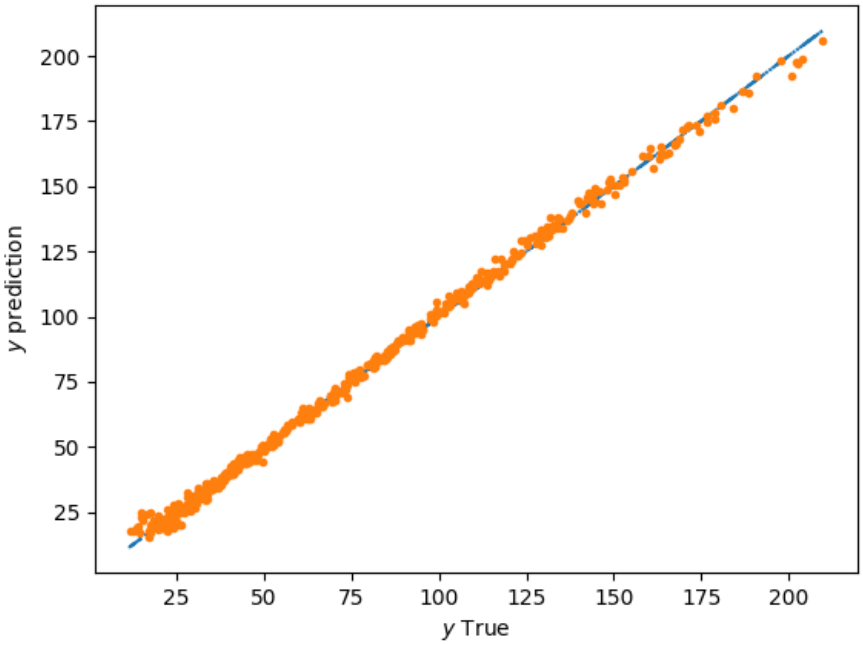
```
import matplotlib.pyplot as plt
import numpy as np
from scipy import linalg
from smt.utils import compute_rms_error

from smt.problems import WaterFlowLFidelity, WaterFlow
from smt.sampling_methods import LHS
from smt.applications import VFM

# Problem set up
ndim = 8
ntest = 500
ndoeLF = int(10 * ndim)
ndoeHF = int(3)
funLF = WaterFlowLFidelity(ndim=ndim)
funHF = WaterFlow(ndim=ndim)
deriv1 = True
deriv2 = True
LF_candidate = "QP"
Bridge_candidate = "KRG"
type_bridge = "Multiplicative"
optionsLF = {}
optionsB = {"theta0": [1e-2] * ndim, "print_prediction": False, "deriv": False}

# Construct low/high fidelity data and validation points
sampling = LHS(xlimits=funLF.xlimits, criterion="m")
xLF = sampling(ndoeLF)
yLF = funLF(xLF)
if deriv1:
    dy_LF = np.zeros((ndoeLF, 1))
    for i in range(ndim):
        yd = funLF(xLF, kx=i)
        dy_LF = np.concatenate((dy_LF, yd), axis=1)

sampling = LHS(xlimits=funHF.xlimits, criterion="m")
xHF = sampling(ndoeHF)
yHF = funHF(xHF)
if deriv2:
    dy_HF = np.zeros((ndoeHF, 1))
    for i in range(ndim):
        yd = funHF(xHF, kx=i)
        dy_HF = np.concatenate((dy_HF, yd), axis=1)

xtest = sampling(ntest)
ytest = funHF(xtest)
dytest = np.zeros((ntest, ndim))
for i in range(ndim):
    dytest[:, i] = funHF(xtest, kx=i).T

# Initialize the extension VFM
M = VFM(
```

(continues on next page)

(continued from previous page)

```
    type_bridge=type_bridge,
    name_model_LF=LF_candidate,
    name_model_bridge=Bridge_candidate,
    X_LF=xLF,
    y_LF=yLF,
    X_HF=xHF,
    y_HF=yHF,
    options_LF=optionsLF,
    options_bridge=optionsB,
    dy_LF=dy_LF,
    dy_HF=dy_HF,
)

# Prediction of the validation points
y = M.predict_values(x=xtest)

plt.figure()
plt.plot(ytest, ytest, "-.")
plt.plot(ytest, y, ".")
plt.xlabel(r"$y$ True")
plt.ylabel(r"$y$ prediction")
plt.show()
```

```
___________________________________________________________________________

                                  Kriging
___________________________________________________________________________

 Problem size

      # training points.        : 3

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.2168441
```



Options

Table 28: List of options

Option	Default	Acceptable values	Acceptable types	Description
name_model_LF	None	['KRG', 'LS', 'QP', 'KPLS', 'KPLSK', 'GEKPLS', 'RBF', 'RMTC', 'RMTB', 'IDW']	['object']	Name of the low-fidelity model
options_LF	{}	None	['dict']	Options for the low-fidelity model
name_model_bridge	None	['KRG', 'LS', 'QP', 'KPLS', 'KPLSK', 'GEKPLS', 'RBF', 'RMTC', 'RMTB', 'IDW']	['object']	Name of the bridge model
options_bridge	{}	None	['dict']	Options for the bridge model
type_bridge	Additive	['Additive', 'Multiplicative']	['str']	Bridge function type
X_LF	None	None	['ndarray']	Low-fidelity inputs
y_LF	None	None	['ndarray']	Low-fidelity output
X_HF	None	None	['ndarray']	High-fidelity inputs
y_HF	None	None	['ndarray']	High-fidelity output
dy_LF	None	None	['ndarray']	Low-fidelity derivatives
dy_HF	None	None	['ndarray']	High-fidelity derivatives

3.6.3 Multi-Fidelity Kriging (MFK)

MFK is a multi-fidelity modeling method which uses an autoregressive model of order 1 (AR1).

$$y_{\text{high}}(\mathbf{x}) = \rho(x) \cdot y_{\text{low}}(\mathbf{x}) + \delta(\mathbf{x})$$

where $\rho(x)$ is a scaling/correlation factor (constant, linear or quadratic) and $\delta(\cdot)$ is a discrepancy function.

The additive AR1 formulation was first introduced by Kennedy and O'Hagan[1]. The implementation here follows the one proposed by Le Gratiet[2]. It offers the advantage of being recursive, easily extended to n levels of fidelity and offers better scaling for high numbers of samples. This method only uses nested sampling training points as described by Le Gratiet[Page 165, 2].

References

Usage

```
import numpy as np
import matplotlib.pyplot as plt
from smt.applications.mfk import MFK, NestedLHS

# low fidelity model
def lf_function(x):
    import numpy as np

    return (
        0.5 * ((x * 6 - 2) ** 2) * np.sin((x * 6 - 2) * 2)
        + (x - 0.5) * 10.0
        - 5
    )

# high fidelity model
def hf_function(x):
    import numpy as np

    return ((x * 6 - 2) ** 2) * np.sin((x * 6 - 2) * 2)

# Problem set up
xlimits = np.array([[0.0, 1.0]])
xdoes = NestedLHS(nlevel=2, xlimits=xlimits, random_state=0)
xt_c, xt_e = xdoes(7)

# Evaluate the HF and LF functions
yt_e = hf_function(xt_e)
yt_c = lf_function(xt_c)

sm = MFK(theta0=xt_e.shape[1] * [1.0])

# low-fidelity dataset names being integers from 0 to level-1
sm.set_training_values(xt_c, yt_c, name=0)
# high-fidelity dataset without name
```

(continues on next page)

[1] Kennedy, M.C. and O'Hagan, A., Bayesian calibration of computer models. Journal of the Royal Statistical Society. 2001
[2] Le Gratiet, L., Multi-fidelity Gaussian process regression for computer experiments. PhD Thesis. 2013

(continued from previous page)

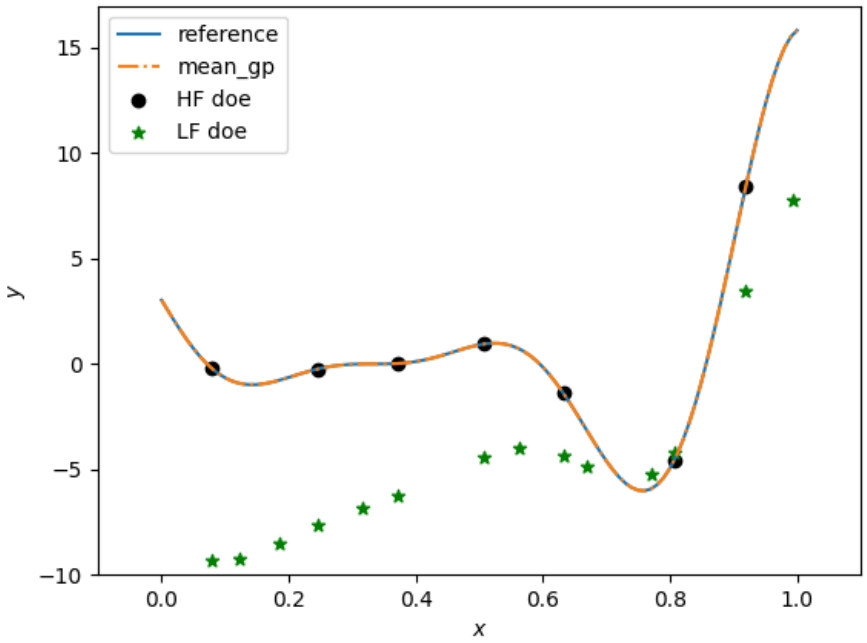
```
sm.set_training_values(xt_e, yt_e)

# train the model
sm.train()

x = np.linspace(0, 1, 101, endpoint=True).reshape(-1, 1)

# query the outputs
y = sm.predict_values(x)
mse = sm.predict_variances(x)
derivs = sm.predict_derivatives(x, kx=0)

plt.figure()

plt.plot(x, hf_function(x), label="reference")
plt.plot(x, y, linestyle="-.", label="mean_gp")
plt.scatter(xt_e, yt_e, marker="o", color="k", label="HF doe")
plt.scatter(xt_c, yt_c, marker="*", color="g", label="LF doe")

plt.legend(loc=0)
plt.ylim(-10, 17)
plt.xlim(-0.1, 1.1)
plt.xlabel(r"$x$")
plt.ylabel(r"$y$")

plt.show()
```

```
___________________________________________________________________________

                                  MFK
___________________________________________________________________________

 Problem size

      # training points.        : 7

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.0562565
___________________________________________________________________________

 Evaluation

      # eval points. : 101

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```

(continues on next page)

(continued from previous page)

```
___________________________________________________________________________

Evaluation

     # eval points. : 101

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000
```



Options

Table 29: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
poly	constant	['constant', 'linear', 'quadratic']	['str']	Regression function type
corr	squar_exp	['abs_exp', 'squar_exp', 'act_exp', 'matern52', 'matern32']	['str']	Correlation function type
categorical_kernel	None	['gower', 'homoscedastic_gaussian_matrix_kernel', 'full_gaussian_matrix_kernel']	['str']	The kernel to use for categorical inputs. Only for non continuous Kriging
xtypes	None	None	['list']	x type specifications: either FLOAT for continuous, INT for integer or (ENUM n) for categorical dimension with n levels
nugget	2.220446049250313e-14	None	['float']	a jitter for numerical stability
theta0	[0.01]	None	['list', 'ndarray']	Initial hyperparameters
theta_bounds	[1e-06, 20.0]	None	['list', 'ndarray']	bounds for hyperparameters
hyper_opt	Cobyla	['Cobyla', 'TNC']	['str']	Optimiser for hyperparameters optimisation
eval_noise	False	[True, False]	['bool']	noise evaluation flag
noise0	[0.0]	None	['list', 'ndarray']	Initial noise hyperparameters
noise_bounds	[2.220446049250313e-14, 10000000000.0]	None	['list', 'ndarray']	bounds for noise hyperparameters
use_het_noise	False	[True, False]	['bool']	heteroscedastic noise evaluation flag
n_start	10	None	['int']	number of optimizer runs (multistart method)
rho_regr	constant	['constant', 'linear', 'quadratic']	None	Regression function type for rho
optim_var	False	[True, False]	['bool']	If True, the variance at HF samples is forced to zero
propagate_uncertainty	True	[True, False]	['bool']	If True, the variance cotribution of lower fidelity levels are considered

3.6.4 Multi-Fidelity Kriging KPLS (MFKPLS)

Partial Least Squares (PLS) is a statistical method to analyze the variations of a quantity of interest w.r.t underlying variables. PLS method gives directions (principal compoenents) that maximize the variation of the quantity of interest.

These principal components define rotations that can be applied to define bases changes. The principal components can be truncated at any number (called n_comp) to explain a 'majority' of the data variations.[1] used the PLS to define subspaces to make high-dimensional Kriging more efficient.

We apply the same idea to *Multi-Fidelity Kriging (MFK)*. The only difference is that we do not apply the PLS analysis step on all datasets. We apply the PLS analysis step on the high-fidelity to preserve the robustness to poor correlations between fidelity levels. A hyperparameter optimization is then performed in the subspace that maximizes the variations of HF data.

MFKPLS is a combination of *Multi-Fidelity Kriging (MFK)* and *KPLS* techniques.

References

Usage

```
import numpy as np
import matplotlib.pyplot as plt
from smt.applications.mfk import MFK, NestedLHS
from smt.applications.mfkpls import MFKPLS

# low fidelity model
def lf_function(x):
    import numpy as np

    return (
        0.5 * ((x * 6 - 2) ** 2) * np.sin((x * 6 - 2) * 2)
        + (x - 0.5) * 10.0
        - 5
    )

# high fidelity model
def hf_function(x):
    import numpy as np

    return ((x * 6 - 2) ** 2) * np.sin((x * 6 - 2) * 2)

# Problem set up
xlimits = np.array([[0.0, 1.0]])
xdoes = NestedLHS(nlevel=2, xlimits=xlimits, random_state=0)
xt_c, xt_e = xdoes(7)

# Evaluate the HF and LF functions
yt_e = hf_function(xt_e)
yt_c = lf_function(xt_c)

# choice of number of PLS components
```

(continues on next page)

[1] Bouhlel, M. A., Bartoli, N., Otsmane, A., & Morlier, J. (2016). An improved approach for estimating the hyperparameters of the kriging model for high-dimensional problems through the partial least squares method. Mathematical Problems in Engineering, 2016.

(continued from previous page)

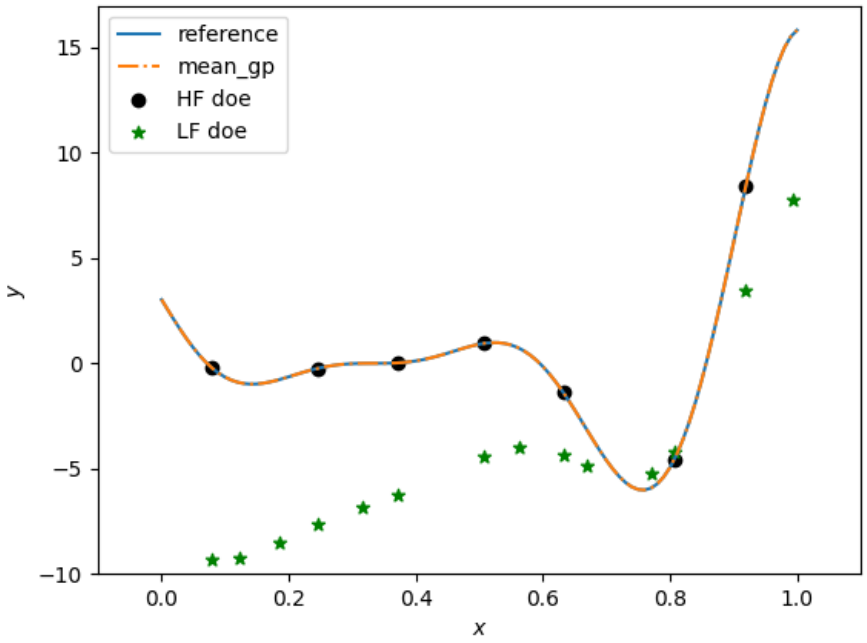
```
ncomp = 1
sm = MFKPLS(n_comp=ncomp, theta0=ncomp * [1.0])

# low-fidelity dataset names being integers from 0 to level-1
sm.set_training_values(xt_c, yt_c, name=0)
# high-fidelity dataset without name
sm.set_training_values(xt_e, yt_e)

# train the model
sm.train()

x = np.linspace(0, 1, 101, endpoint=True).reshape(-1, 1)

# query the outputs
y = sm.predict_values(x)
mse = sm.predict_variances(x)
derivs = sm.predict_derivatives(x, kx=0)

plt.figure()

plt.plot(x, hf_function(x), label="reference")
plt.plot(x, y, linestyle="-.", label="mean_gp")
plt.scatter(xt_e, yt_e, marker="o", color="k", label="HF doe")
plt.scatter(xt_c, yt_c, marker="*", color="g", label="LF doe")

plt.legend(loc=0)
plt.ylim(-10, 17)
plt.xlim(-0.1, 1.1)
plt.xlabel(r"$x$")
plt.ylabel(r"$y$")

plt.show()
```

```
___________________________________________________________________________

                                  MFKPLS
___________________________________________________________________________

 Problem size

      # training points.        : 7

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.0579879
___________________________________________________________________________

 Evaluation
```

(continues on next page)

(continued from previous page)

```
    # eval points. : 101

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


_____________________________________________________________________________

Evaluation

    # eval points. : 101

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000
```



Options

Table 30: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
poly	constant	['constant', 'linear', 'quadratic']	['str']	Regression function type
corr	squar_exp	['abs_exp', 'squar_exp']	['str']	Correlation function type
categorical_kernel	None	['gower', 'homoscedastic_gaussian_matrix_kernel', 'full_gaussian_matrix_kernel']	['str']	The kernel to use for categorical inputs. Only for non continuous Kriging
xtypes	None	None	['list']	x type specifications: either FLOAT for continuous, INT for integer or (ENUM n) for categorical dimension with n levels
nugget	2.220446049250313e-14	None	['float']	a jitter for numerical stability
theta0	[0.01]	None	['list', 'ndarray']	Initial hyperparameters
theta_bounds	[1e-06, 20.0]	None	['list', 'ndarray']	bounds for hyperparameters
hyper_opt	Cobyla	['Cobyla', 'TNC']	['str']	Optimiser for hyperparameters optimisation
eval_noise	False	[True, False]	['bool']	noise evaluation flag
noise0	[0.0]	None	['list', 'ndarray']	Initial noise hyperparameters
noise_bounds	[2.220446049250313e-14, 10000000000.0]	None	['list', 'ndarray']	bounds for noise hyperparameters
use_het_noise	False	[True, False]	['bool']	heteroscedastic noise evaluation flag
n_start	10	None	['int']	number of optimizer runs (multistart method)
rho_regr	constant	['constant', 'linear', 'quadratic']	None	Regression function type for rho
optim_var	False	[True, False]	['bool']	If True, the variance at HF samples is forced to zero
propagate_uncertainty	True	[True, False]	['bool']	If True, the variance cotribution of lower fidelity levels are considered
n_comp	1	None	['int']	Number of principal components

3.6.5 Multi-Fidelity Kriging KPLSK (MFKPLSK)

Partial Least Squares (PLS) is a statistical method to analyze the variations of a quantity of interest w.r.t underlying variables. PLS method gives directions (principal compoenents) that maximize the variation of the quantity of interest.

These principal components define rotations that can be applied to define bases changes. The principal components can be truncated at any number (called n_comp) to explain a 'majority' of the data variations.[1] used the PLS to define subspaces to make high-dimensional Kriging more efficient.

We apply the same idea to *Multi-Fidelity Kriging (MFK)*. The only difference is that we do not apply the PLS analysis step on all datasets. We apply the PLS analysis step on the high-fidelity to preserve the robustness to poor correlations between fidelity levels. A hyperparameter optimization is then performed in the subspace that maximizes the variations of HF data.

MFKPLSK is a combination of *Multi-Fidelity Kriging (MFK)* and *KPLSK* techniques.

References

Usage

```
import numpy as np
import matplotlib.pyplot as plt
from smt.applications.mfk import MFK, NestedLHS
from smt.applications.mfkplsk import MFKPLSK

# low fidelity modelk
def lf_function(x):
    import numpy as np

    return (
        0.5 * ((x * 6 - 2) ** 2) * np.sin((x * 6 - 2) * 2)
        + (x - 0.5) * 10.0
        - 5
    )

# high fidelity model
def hf_function(x):
    import numpy as np

    return ((x * 6 - 2) ** 2) * np.sin((x * 6 - 2) * 2)

# Problem set up
xlimits = np.array([[0.0, 1.0]])
xdoes = NestedLHS(nlevel=2, xlimits=xlimits, random_state=0)
xt_c, xt_e = xdoes(7)

# Evaluate the HF and LF functions
yt_e = hf_function(xt_e)
yt_c = lf_function(xt_c)

# choice of number of PLS components
```

(continues on next page)

[1] Bouhlel, M. A., Bartoli, N., Otsmane, A., & Morlier, J. (2016). An improved approach for estimating the hyperparameters of the kriging model for high-dimensional problems through the partial least squares method. Mathematical Problems in Engineering, 2016.

(continued from previous page)

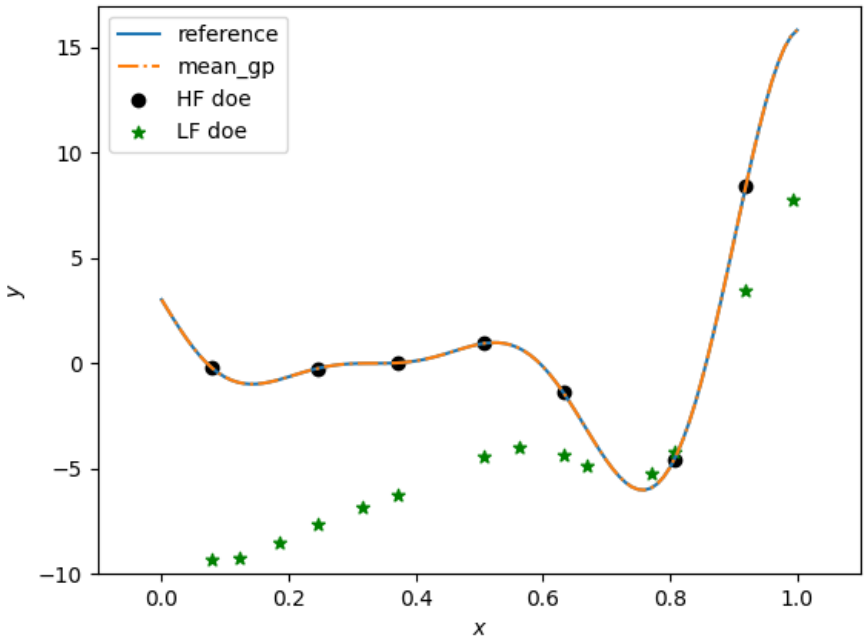
```
ncomp = 1
sm = MFKPLSK(n_comp=ncomp, theta0=ncomp * [1.0])

# low-fidelity dataset names being integers from 0 to level-1
sm.set_training_values(xt_c, yt_c, name=0)
# high-fidelity dataset without name
sm.set_training_values(xt_e, yt_e)

# train the model
sm.train()

x = np.linspace(0, 1, 101, endpoint=True).reshape(-1, 1)

# query the outputs
y = sm.predict_values(x)
mse = sm.predict_variances(x)
derivs = sm.predict_derivatives(x, kx=0)

plt.figure()

plt.plot(x, hf_function(x), label="reference")
plt.plot(x, y, linestyle="-.", label="mean_gp")
plt.scatter(xt_e, yt_e, marker="o", color="k", label="HF doe")
plt.scatter(xt_c, yt_c, marker="*", color="g", label="LF doe")

plt.legend(loc=0)
plt.ylim(-10, 17)
plt.xlim(-0.1, 1.1)
plt.xlabel(r"$x$")
plt.ylabel(r"$y$")

plt.show()
```

```
___________________________________________________________________________

                                  MFKPLSK
___________________________________________________________________________

 Problem size

      # training points.        : 7

___________________________________________________________________________

 Training

   Training ...
   Training - done. Time (sec):  0.1054325
___________________________________________________________________________

 Evaluation
```

(continues on next page)

(continued from previous page)

```
    # eval points. : 101

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000


___________________________________________________________________________

 Evaluation

    # eval points. : 101

  Predicting ...
  Predicting - done. Time (sec):  0.0000000

  Prediction time/pt. (sec) :  0.0000000
```



Options

Table 31: List of options

Option	Default	Acceptable values	Acceptable types	Description
print_global	True	None	['bool']	Global print toggle. If False, all printing is suppressed
print_training	True	None	['bool']	Whether to print training information
print_prediction	True	None	['bool']	Whether to print prediction information
print_problem	True	None	['bool']	Whether to print problem information
print_solver	True	None	['bool']	Whether to print solver information
poly	constant	['constant', 'linear', 'quadratic']	['str']	Regression function type
corr	squar_exp	['squar_exp']	['str']	Correlation function type
categorical_kernel	None	['gower', 'homoscedastic_gaussian_matrix_kernel', 'full_gaussian_matrix_kernel']	['str']	The kernel to use for categorical inputs. Only for non continuous Kriging
xtypes	None	None	['list']	x type specifications: either FLOAT for continuous, INT for integer or (ENUM n) for categorical dimension with n levels
nugget	2.220446049250313e-14	None	['float']	a jitter for numerical stability
theta0	[0.01]	None	['list', 'ndarray']	Initial hyperparameters
theta_bounds	[1e-06, 20.0]	None	['list', 'ndarray']	bounds for hyperparameters
hyper_opt	Cobyla	['Cobyla', 'TNC']	['str']	Optimiser for hyperparameters optimisation
eval_noise	False	[True, False]	['bool']	noise evaluation flag
noise0	[0.0]	None	['list', 'ndarray']	Initial noise hyperparameters
noise_bounds	[2.220446049250313e-14, 10000000000.0]	None	['list', 'ndarray']	bounds for noise hyperparameters
use_het_noise	False	[True, False]	['bool']	heteroscedastic noise evaluation flag
n_start	10	None	['int']	number of optimizer runs (multistart method)
rho_regr	constant	['constant', 'linear', 'quadratic']	None	Regression function type for rho
optim_var	False	[True, False]	['bool']	If True, the variance at HF samples is forced to zero
propagate_uncertainty	True	[True, False]	['bool']	If True, the variance cotribution of lower fidelity levels are considered
n_comp	1	None	['int']	Number of principal components

3.6.6 Efficient Global Optimization (EGO)

Bayesian Optimization

Bayesian optimization is defined by Jonas Mockus in[1] as an optimization technique based upon the minimization of the expected deviation from the extremum of the studied function.

The objective function is treated as a black-box function. A Bayesian strategy sees the objective as a random function and places a prior over it. The prior captures our beliefs about the behavior of the function. After gathering the function evaluations, which are treated as data, the prior is updated to form the posterior distribution over the objective function. The posterior distribution, in turn, is used to construct an acquisition function (often also referred to as infill sampling criterion) that determines what the next query point should be.

One of the earliest bodies of work on Bayesian optimisation that we are aware of are[2] and[3]. Kushner used Wiener processes for one-dimensional problems. Kushner's decision model was based on maximizing the probability of improvement, and included a parameter that controlled the trade-off between 'more global' and 'more local' optimization, in the same spirit as the Exploration/Exploitation trade-off.

Meanwhile, in the former Soviet Union, Mockus and colleagues developed a multidimensional Bayesian optimization method using linear combinations of Wiener fields, some of which was published in English in[Page 177, 1]. This paper also describes an acquisition function that is based on myopic expected improvement of the posterior, which has been widely adopted in Bayesian optimization as the Expected Improvement function.

In 1998, Jones used Gaussian processes together with the expected improvement function to successfully perform derivative-free optimization and experimental design through an algorithm called Efficient Global Optimization, or EGO.

EGO

In what follows, we describe the Efficient Global Optimization (EGO) algorithm, as published in[4].

Let F be an expensive black-box function to be minimized. We sample F at the different locations $X = \{x_1, x_2, \ldots, x_n\}$ yielding the responses $Y = \{y_1, y_2, \ldots, y_n\}$. We build a Kriging model (also called Gaussian process) with a mean function μ and a variance function σ^2.

The next step is to compute the criterion EI. To do this, let us denote:

$$f_{min} = \min\{y_1, y_2, \ldots, y_n\}. \quad (3.2)$$

The Expected Improvement function (EI) can be expressed:

$$E[I(x)] = E[\max(f_{min} - Y, 0)] \quad (3.3)$$

[1] Mockus, J. (1975). On Bayesian methods for seeking the extremum. In Optimization Techniques IFIP Technical Conference (pp. 400-404). Springer, Berlin, Heidelberg.

[2] Kushner, H. J. (1962). A versatile stochastic model of a function of unknown and time varying form. Journal of Mathematical Analysis and Applications, 5(1), 150-167.

[3] Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. Journal of Basic Engineering, 86(1), 97-106.

[4] Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. Journal of Global optimization, 13(4), 455-492.

where Y is the random variable following the distribution $\mathcal{N}(\mu(x), \sigma^2(x))$. By expressing the right-hand side of EI expression as an integral, and applying some tedious integration by parts, one can express the expected improvement in closed form:

$$E[I(x)] = (f_{min} - \mu(x))\Phi\left(\frac{f_{min} - \mu(x)}{\sigma(x)}\right) + \sigma(x)\phi\left(\frac{f_{min} - \mu(x)}{\sigma(x)}\right) \quad (3.4)$$

where $\Phi(\cdot)$ and $\phi(\cdot)$ are respectively the cumulative and probability density functions of $\mathcal{N}(0, 1)$.

Next, we determine our next sampling point as :

$$x_{n+1} = \arg\max_x \left(E[I(x)]\right) \quad (3.5)$$

We then test the response y_{n+1} of our black-box function F at x_{n+1}, rebuild the model taking into account the new information gained, and research the point of maximum expected improvement again.

We summarize here the EGO algorithm:

EGO(F, n_{iter}) # Find the best minimum of F in n_{iter} iterations

For ($i = 0 : n_{iter}$)

- $mod = model(X, Y)$ # surrogate model based on sample vectors X and Y
- $f_{min} = \min Y$
- $x_{i+1} = \arg\max EI(mod, f_{min})$ # choose x that maximizes EI
- $y_{i+1} = F(x_{i+1})$ # Probe the function at most promising point x_{i+1}
- $X = [X, x_{i+1}]$
- $Y = [Y, y_{i+1}]$
- $i = i + 1$

$f_{min} = \min Y$

Return : f_{min} # This is the best known solution after n_{iter} iterations.

More details can be found in[4].

EGO parallel (EGO with qEI criterion)

The goal is to be able to run batch optimization. At each iteration of the algorithm, multiple new sampling points are extracted from the know ones. These new sampling points are then evaluated using a parallel computing environment.

The parallel version of this algorithm has been presented by Ginsbourger et al.[5] in 2010. The Expected improvement (*EI*) is extended to proposed q new sampling points instead of one, they called this criterion the *qEI* criterion. As the exact evaluation is not straightforward they proposed different ways to approximate this criterion.

[5] Ginsbourger, D., Le Riche, R., & Carraro, L. (2010). Kriging is well-suited to parallelize optimization. In Computational intelligence in expensive optimization problems (pp. 131-162). Springer, Berlin, Heidelberg.

Details of the implementation can be found in[67].

Differents approximation strategy of the *qEI* criterion

The basic idea is to run q iterations of the *EGO* algorithm and to set temporally the response $\hat{y}_q$ of the q new sampling points to a virtual value. When the q new sampling points are defined the real evaluation of the response y_q of these points is done in parallel. The efficiency of the methods lies in the strategy to set the virtual values.

Let assume that the new sampling point is at x_q. The virtual response y_q is set according to one of those strategies:

The minimum constant liar (*CLmin*) strategy

$\hat{y}_q = \min(Y)$

The Kriging believer (*KB*) strategy

The Kriging model gives a mean function μ and a variance function σ^2 based on sample vectors X and Y.

The virtual values are set according to the model prediction:

$\hat{y}_q = \mu(x_q)$

Some variants are proposed to introduce an optimistic or pessimistic part :

- the Kriging Believer Upper Bound (KBUB) : $\hat{y}_q = \mu(x_q) + 3\sigma$
- the Kriging Believer Lower Bound (KBLB) : $\hat{y}_q = \mu(x_q) - 3\sigma$

Tips for an efficient use

- the *n_parallel* parameter is set by the user, a real improvement of the efficiency is observed for relatively low values of the parameter (<8)[Page 177, 3]
- Since the maximization of the *EI* is a highly multimodal optimization problem, it could be necessary to increase the *n_start* parameter of the algorithm.

Implementation Notes

Beside the Expected Improvement, the implementation here offers two other infill criteria:

- SBO (Surrogate Based Optimization): directly using the prediction of the surrogate model (μ)
- LCB (Lower Confidence Bound): using the 99% confidence interval $\mu - 3 \times \sigma$

Regarding the parallel execution, one can implement specific multiprocessing by deriving the _Evaluator_ interface and overriding the default implementation of the _run(fun, x)_ method. The default implementation simply runs _fun(x)_.

Regardless the others parameters, you can specify a mixed surrogate model to make mixed optimization. See[8]. The expected improvement is continuously computed and optimized so that can lead to an infill point that will be projected,

[6] Roux, E. , Tillier, Y. , Kraria, S., & Bouchard, P.-O. (2020). An efficient parallel global optimization strategy based on Kriging properties suitable for material parameter identification.In AME, accepted for publication.

[7] Roux, E. (2011). Assemblage mécanique: stratégies d'optimisation des procédés et d'identification des comportements mécaniques des matériaux (Doctoral dissertation).

[8] E.C. Garrido-Merchan and D. Hernandez-Lobato. Dealing with categorical and integer-valued variables in Bayesian Optimization with Gaussian processes. In:Neurocomputing 380 (2020), pages 20–35.

in the mixed case, to an already evaluated point. To avoid the re-evaluation of a point, you can penalize the Expected Improvement via tunneling which decrease the EI in the neighbourhood of the known DOE points. However, this is not recommanded for high dimensional problems because the re-evaluation is uncommon. Tunneling evaluation can be slow with a lot of point.

When considering a mixed integer optimization, the function to be optimized by EGO has to handle categorical variables as indexes in the given enumaration type. For instance, with a categorical enumeration `["red", "green", "blue"]`, passing `"blue"` to the function should be handled by passing the value `2` which is the index of `"blue"` in the enumeration list. This choice was made to keep on using a numerical ndarray as interface of the function to be optimized `f: [n_samples, n_features] -> [n_eval, 1]` allowing parallel evaluations.

References

Usage

```
import numpy as np
from smt.applications import EGO
import matplotlib.pyplot as plt

def function_test_1d(x):
    # function xsinx
    import numpy as np

    x = np.reshape(x, (-1,))
    y = np.zeros(x.shape)
    y = (x - 3.5) * np.sin((x - 3.5) / (np.pi))
    return y.reshape((-1, 1))

n_iter = 6
xlimits = np.array([[0.0, 25.0]])
xdoe = np.atleast_2d([0, 7, 25]).T
n_doe = xdoe.size

criterion = "EI"  #'EI' or 'SBO' or 'LCB'

ego = EGO(n_iter=n_iter, criterion=criterion, xdoe=xdoe, xlimits=xlimits)

x_opt, y_opt, _, x_data, y_data = ego.optimize(fun=function_test_1d)
print("Minimum in x={:.1f} with f(x)={:.1f}".format(float(x_opt), float(y_opt)))

x_plot = np.atleast_2d(np.linspace(0, 25, 100)).T
y_plot = function_test_1d(x_plot)

fig = plt.figure(figsize=[10, 10])
for i in range(n_iter):
    k = n_doe + i
    x_data_k = x_data[0:k]
    y_data_k = y_data[0:k]
    ego.gpr.set_training_values(x_data_k, y_data_k)
    ego.gpr.train()

    y_gp_plot = ego.gpr.predict_values(x_plot)
    y_gp_plot_var = ego.gpr.predict_variances(x_plot)
```

(continues on next page)

(continued from previous page)

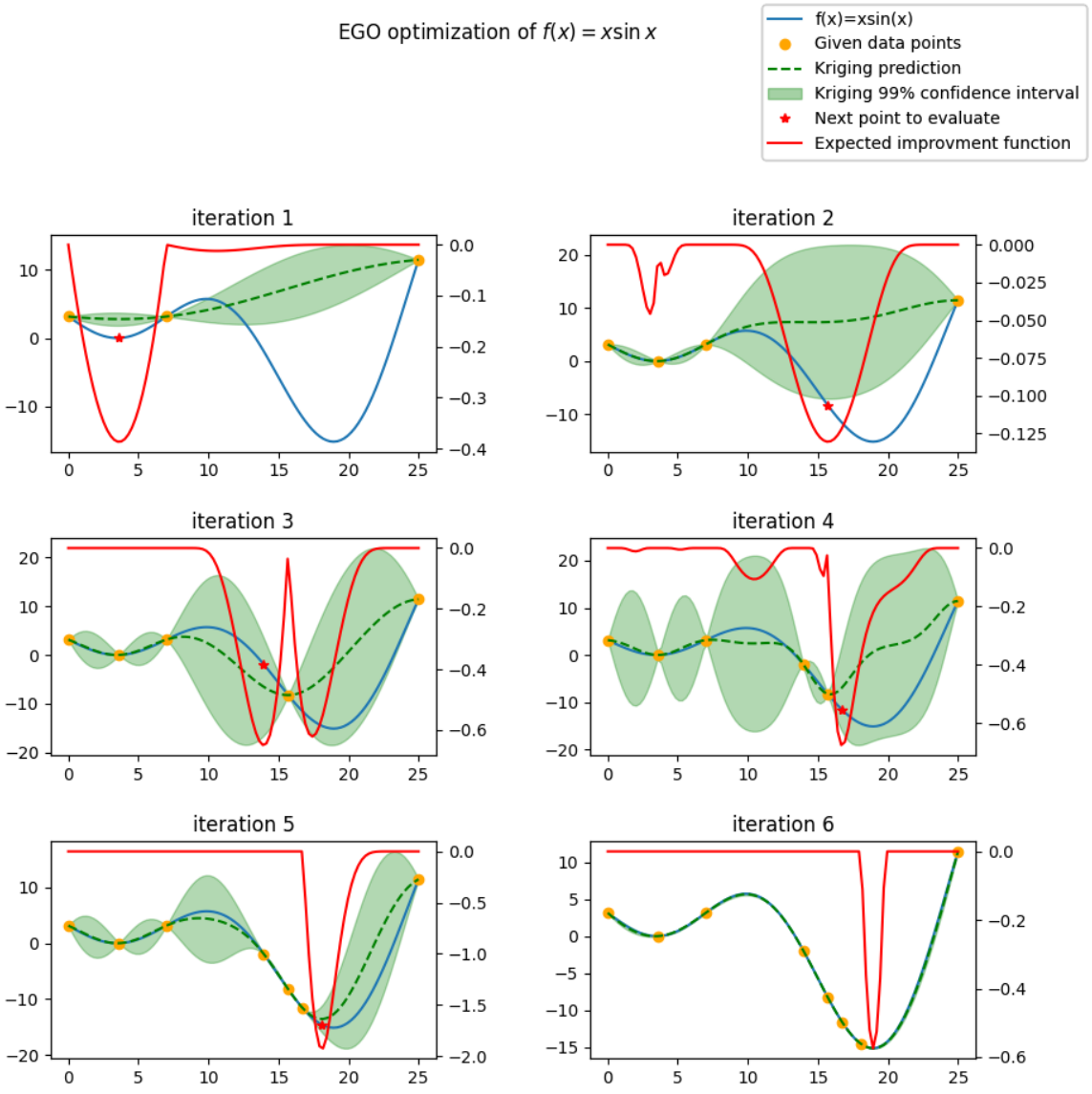
```
    y_ei_plot = -ego.EI(x_plot)

    ax = fig.add_subplot((n_iter + 1) // 2, 2, i + 1)
    ax1 = ax.twinx()
    (ei,) = ax1.plot(x_plot, y_ei_plot, color="red")

    (true_fun,) = ax.plot(x_plot, y_plot)
    (data,) = ax.plot(
        x_data_k, y_data_k, linestyle="", marker="o", color="orange"
    )
    if i < n_iter - 1:
        (opt,) = ax.plot(
            x_data[k], y_data[k], linestyle="", marker="*", color="r"
        )
    (gp,) = ax.plot(x_plot, y_gp_plot, linestyle="--", color="g")
    sig_plus = y_gp_plot + 3 * np.sqrt(y_gp_plot_var)
    sig_moins = y_gp_plot - 3 * np.sqrt(y_gp_plot_var)
    un_gp = ax.fill_between(
        x_plot.T[0], sig_plus.T[0], sig_moins.T[0], alpha=0.3, color="g"
    )
    lines = [true_fun, data, gp, un_gp, opt, ei]
    fig.suptitle("EGO optimization of $f(x) = x \sin{x}$")
    fig.subplots_adjust(hspace=0.4, wspace=0.4, top=0.8)
    ax.set_title("iteration {}".format(i + 1))
    fig.legend(
        lines,
        [
            "f(x)=xsin(x)",
            "Given data points",
            "Kriging prediction",
            "Kriging 99% confidence interval",
            "Next point to evaluate",
            "Expected improvment function",
        ],
    )
plt.show()
```

```
Minimum in x=18.9 with f(x)=-15.1
```

Usage with parallel options

```
import numpy as np
from smt.applications import EGO
from smt.applications.ego import EGO, Evaluator
from smt.sampling_methods import FullFactorial

import sklearn
import matplotlib.pyplot as plt
from matplotlib import colors
from mpl_toolkits.mplot3d import Axes3D
```

(continues on next page)


EGO optimization of $f(x) = x \sin x$
f(x)=xsin(x)
Given data points
Kriging prediction
Kriging 99% confidence interval
Next point to evaluate
Expected improvment function
iteration 1
iteration 2
iteration 3
iteration 4
iteration 5
iteration 6

(continued from previous page)

```
from scipy.stats import norm

def function_test_1d(x):
    # function xsinx
    import numpy as np

    x = np.reshape(x, (-1,))
    y = np.zeros(x.shape)
    y = (x - 3.5) * np.sin((x - 3.5) / (np.pi))
    return y.reshape((-1, 1))

n_iter = 3
n_parallel = 3
n_start = 50
xlimits = np.array([[0.0, 25.0]])
xdoe = np.atleast_2d([0, 7, 25]).T
n_doe = xdoe.size

class ParallelEvaluator(Evaluator):
    """
    Implement Evaluator interface using multiprocessing ThreadPool object (Python 3␣
↪only).
    """

    def run(self, fun, x):
        n_thread = 5
        # Caveat: import are made here due to SMT documentation building process
        import numpy as np
        from sys import version_info
        from multiprocessing.pool import ThreadPool

        if version_info.major == 2:
            return fun(x)
        # Python 3 only
        with ThreadPool(n_thread) as p:
            return np.array(
                [
                    y[0]
                    for y in p.map(
                        fun, [np.atleast_2d(x[i]) for i in range(len(x))]
                    )
                ]
            )

criterion = "EI"  #'EI' or 'SBO' or 'LCB'
qEI = "KBUB"  # "KB", "KBLB", "KBUB", "KBRand"
ego = EGO(
    n_iter=n_iter,
    criterion=criterion,
    xdoe=xdoe,
    xlimits=xlimits,
    n_parallel=n_parallel,
```

(continues on next page)

(continued from previous page)

```
    qEI=qEI,
    n_start=n_start,
    evaluator=ParallelEvaluator(),
    random_state=42,
)

x_opt, y_opt, _, x_data, y_data = ego.optimize(fun=function_test_1d)
print("Minimum in x={:.1f} with f(x)={:.1f}".format(float(x_opt), float(y_opt)))

x_plot = np.atleast_2d(np.linspace(0, 25, 100)).T
y_plot = function_test_1d(x_plot)

fig = plt.figure(figsize=[10, 10])
for i in range(n_iter):
    k = n_doe + (i) * (n_parallel)
    x_data_k = x_data[0:k]
    y_data_k = y_data[0:k]
    x_data_sub = x_data_k.copy()
    y_data_sub = y_data_k.copy()
    for p in range(n_parallel):
        ego.gpr.set_training_values(x_data_sub, y_data_sub)
        ego.gpr.train()

        y_ei_plot = -ego.EI(x_plot)
        y_gp_plot = ego.gpr.predict_values(x_plot)
        y_gp_plot_var = ego.gpr.predict_variances(x_plot)

        x_data_sub = np.append(x_data_sub, x_data[k + p])
        y_KB = ego._get_virtual_point(np.atleast_2d(x_data[k + p]), y_data_sub)

        y_data_sub = np.append(y_data_sub, y_KB)

        ax = fig.add_subplot(n_iter, n_parallel, i * (n_parallel) + p + 1)
        ax1 = ax.twinx()
        (ei,) = ax1.plot(x_plot, y_ei_plot, color="red")

        (true_fun,) = ax.plot(x_plot, y_plot)
        (data,) = ax.plot(
            x_data_sub[: -1 - p],
            y_data_sub[: -1 - p],
            linestyle="",
            marker="o",
            color="orange",
        )
        (virt_data,) = ax.plot(
            x_data_sub[-p - 1 : -1],
            y_data_sub[-p - 1 : -1],
            linestyle="",
            marker="o",
            color="g",
        )
```

(continues on next page)

(continued from previous page)

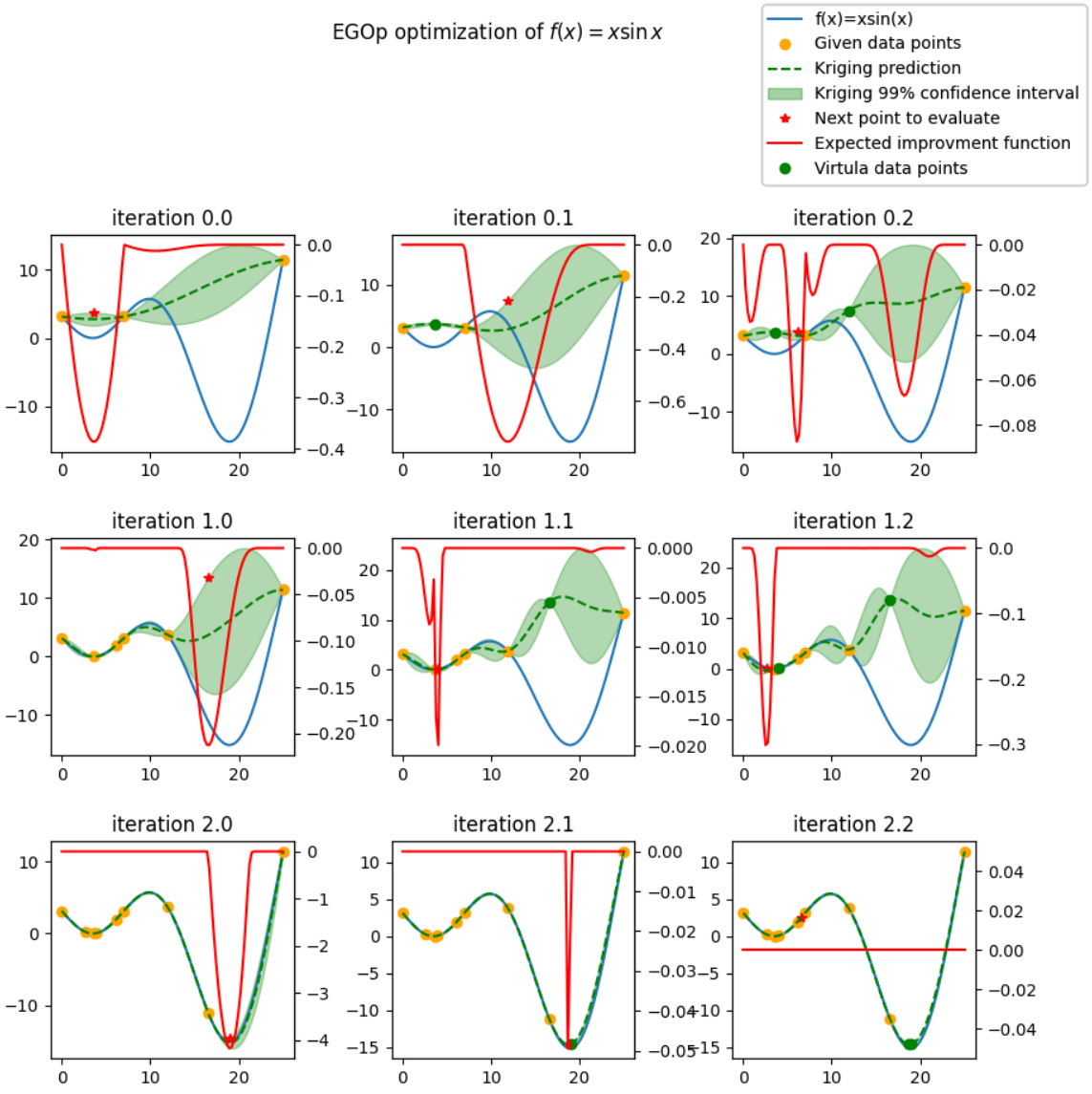
```
        (opt,) = ax.plot(
            x_data_sub[-1], y_data_sub[-1], linestyle="", marker="*", color="r"
        )
        (gp,) = ax.plot(x_plot, y_gp_plot, linestyle="--", color="g")
        sig_plus = y_gp_plot + 3.0 * np.sqrt(y_gp_plot_var)
        sig_moins = y_gp_plot - 3.0 * np.sqrt(y_gp_plot_var)
        un_gp = ax.fill_between(
            x_plot.T[0], sig_plus.T[0], sig_moins.T[0], alpha=0.3, color="g"
        )
        lines = [true_fun, data, gp, un_gp, opt, ei, virt_data]
        fig.suptitle("EGOp optimization of $f(x) = x \sin{x}$")
        fig.subplots_adjust(hspace=0.4, wspace=0.4, top=0.8)
        ax.set_title("iteration {}.{}".format(i, p))
        fig.legend(
            lines,
            [
                "f(x)=xsin(x)",
                "Given data points",
                "Kriging prediction",
                "Kriging 99% confidence interval",
                "Next point to evaluate",
                "Expected improvment function",
                "Virtula data points",
            ],
        )
plt.show()
```

```
Minimum in x=19.0 with f(x)=-15.1
```

Usage with mixed variable

```
import numpy as np
from smt.applications import EGO
from smt.applications.mixed_integer import (
    MixedIntegerContext,
    FLOAT,
    ENUM,
    ORD,
)
import matplotlib.pyplot as plt
from smt.surrogate_models import KRG
from smt.sampling_methods import LHS

# Regarding the interface, the function to be optimized should handle
# categorical values as index values in the enumeration type specification.
# For instance, here "blue" will be passed to the function as the index value 2.
# This allows to keep the numpy ndarray X handling numerical values.
def function_test_mixed_integer(X):
    # float
    x1 = X[:, 0]
```

(continues on next page)


EGOp optimization of $f(x) = x \sin x$
f(x)=xsin(x)
Given data points
Kriging prediction
Kriging 99% confidence interval
Next point to evaluate
Expected improvment function
Virtula data points
iteration 0.0
iteration 0.1
iteration 0.2
iteration 1.0
iteration 1.1
iteration 1.2
iteration 2.0
iteration 2.1
iteration 2.2

(continued from previous page)

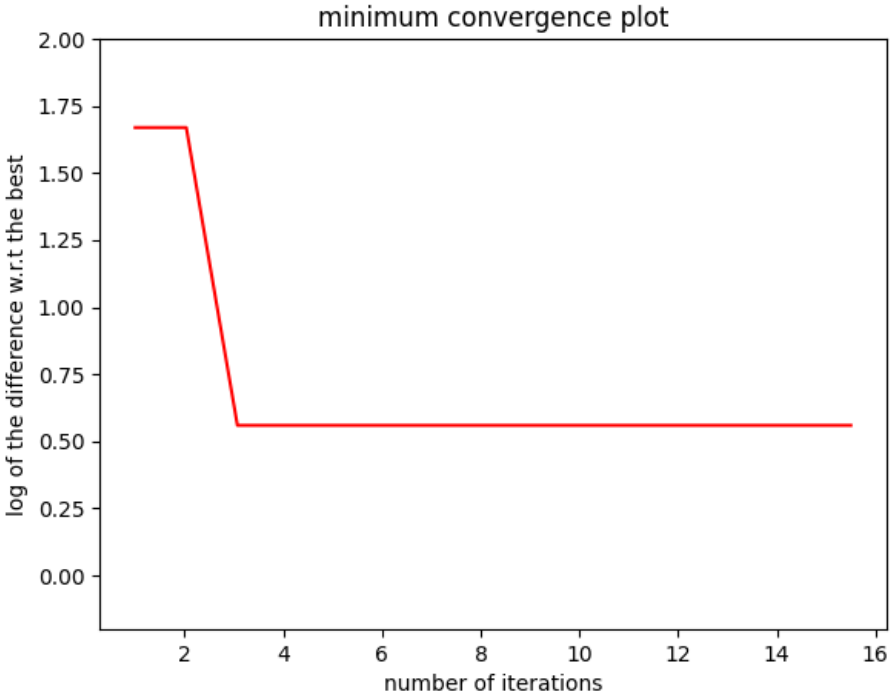
```
    #  enum 1
    c1 = X[:, 1]
    x2 = c1 == 0
    x3 = c1 == 1
    x4 = c1 == 2
    #  enum 2
    c2 = X[:, 2]
    x5 = c2 == 0
    x6 = c2 == 1
    # int
    i = X[:, 3]

    y = (
        (x2 + 2 * x3 + 3 * x4) * x5 * x1
        + (x2 + 2 * x3 + 3 * x4) * x6 * 0.95 * x1
        + i
    )
    return y

n_iter = 15
xtypes = [FLOAT, (ENUM, 3), (ENUM, 2), ORD]
xlimits = np.array(
    [[-5, 5], ["red", "green", "blue"], ["square", "circle"], [0, 2]]
)
criterion = "EI"  #'EI' or 'SBO' or 'LCB'
qEI = "KB"
sm = KRG(print_global=False)
mixint = MixedIntegerContext(xtypes, xlimits)
n_doe = 3
sampling = mixint.build_sampling_method(LHS, criterion="ese", random_state=42)
xdoe = sampling(n_doe)
ydoe = function_test_mixed_integer(xdoe)

ego = EGO(
    n_iter=n_iter,
    criterion=criterion,
    xdoe=xdoe,
    ydoe=ydoe,
    xtypes=xtypes,
    xlimits=xlimits,
    surrogate=sm,
    qEI=qEI,
    random_state=42,
)

x_opt, y_opt, _, _, y_data = ego.optimize(fun=function_test_mixed_integer)
print("Minimum in x={} with f(x)={:.1f}".format(x_opt, float(y_opt)))
print("Minimum in typed x={}".format(ego.mixint.cast_to_mixed_integer(x_opt)))

min_ref = -15
mini = np.zeros(n_iter)
for k in range(n_iter):
```

(continues on next page)

(continued from previous page)

```
    mini[k] = np.log(np.abs(np.min(y_data[0 : k + n_doe - 1]) - min_ref))
x_plot = np.linspace(1, n_iter + 0.5, n_iter)
u = max(np.floor(max(mini)) + 1, -100)
l = max(np.floor(min(mini)) - 0.2, -10)
fig = plt.figure()
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])
axes.plot(x_plot, mini, color="r")
axes.set_ylim([l, u])
plt.title("minimum convergence plot", loc="center")
plt.xlabel("number of iterations")
plt.ylabel("log of the difference w.r.t the best")
plt.show()
```

```
Warning: multiple x input features have the same value (at least same row twice).
Warning: multiple x input features have the same value (at least same row twice).
Warning: multiple x input features have the same value (at least same row twice).
Warning: multiple x input features have the same value (at least same row twice).
Minimum in x=[-5.  2.  1.  1.] with f(x)=-13.2
Minimum in typed x=[-5.0, 'blue', 'circle', 1]
```



Options

Table 32: List of options

Option	Default	Acceptable values	Acceptable types	Description
fun	None	None	['function']	Function to minimize
criterion	EI	['EI', 'SBO', 'LCB']	['str']	criterion for next evaluation point determination: Expected Improvement, Surrogate-Based Optimization or Lower Confidence Bound
n_iter	None	None	['int']	Number of optimizer steps
n_max_optim	20	None	['int']	Maximum number of internal optimizations
n_start	20	None	['int']	Number of optimization start points
n_parallel	1	None	['int']	Number of parallel samples to compute using qEI criterion
qEI	KBLB	['KB', 'KBLB', 'KBUB', 'KBRand', 'CLmin']	['str']	Approximated q-EI maximization strategy
evaluator	<smt.applications.ego.Evaluator object at 0x000001F2D493BD90>	None	['Evaluator']	Object used to run function fun to optimize at x points (nsamples, nxdim)
n_doe	None	None	['int']	Number of points of the initial LHS doe, only used if xdoe is not given
xdoe	None	None	['ndarray']	Initial doe inputs
ydoe	None	None	['ndarray']	Initial doe outputs
xlimits	None	None	['ndarray']	Bounds of function fun inputs
verbose	False	None	['bool']	Print computation information
enable_tunneling	False	None	['bool']	Enable the penalization of points that have been already evaluated in EI criterion
categorical_kernel	None	['gower', 'homoscedastic_gaussian_matrix_kernel', 'full_gaussian_matrix_kernel']	['str']	The kernel to use for categorical inputs. Only for non continuous Kriging.
surrogate	<smt.surrogate_models.krg.KRG object at 0x000001F2D7D03970>	None	['KRG', 'KPLS', 'KPLSK', 'GEKPLS', 'MGP']	SMT kriging-based surrogate model used internaly
xtypes	None	None	['list']	x type specifications: either FLOAT for continuous, INT for integer or (ENUM n) for categorical doimension with n levels
random_state	None	None	['NoneType', 'int', 'RandomState']	Numpy RandomState object or seed number which controls random draws

3.6.7 Mixed-Integer Sampling and Surrogate (Continuous Relaxation)

SMT provides the `mixed_integer` module to adapt existing surrogates to deal with categorical (or enumerate) and ordered variables using continuous relaxation. For ordered variables, the values are rounded to the nearest values from a provided list. If, instead, bounds are provided, the list will consist of all integers between those bounds. For enum variables, as many x features as enumerated levels are created with [0, 1] bounds and the max of these feature float values will correspond to the choice of one the enum value.

For instance, for a categorical variable (one feature of x) with three levels ["blue", "red", "green"], 3 continuous float features x0, x1, x2 are created, the max(x0, x1, x2), let say x1, will give "red" as the value for the original categorical feature.

The user specifies x feature types through a list of types to be either:

- `FLOAT`: a continuous feature,
- `ORD`: an ordered valued feature,
- or a tuple `(ENUM, n)` where n is the number of levels of the catagorical feature (i.e. an enumerate with n values)

In the case of mixed integer sampling, bounds of each x feature have to be adapted to take into account feature types. While FLOAT and INT feature still have an interval [lower bound, upper bound], the ENUM features bounds is defined by giving the enumeration/list of possible values (levels).

For instance, if we have the following `xtypes`: `[FLOAT, ORD, (ENUM, 2), (ENUM, 3)]`, a compatible `xlimits` could be `[[0., 4], [-10, 10], ["blue", "red"], ["short", "medium", "long"]]`

3.6.8 Mixed-Integer Surrogate with Gower Distance

Another implemented method is using a basic mixed integer kernel based on the Gower distance between two points. When constructing the correlation kernel, the distance is redefined as $\Delta = \Delta_{cont} + \Delta_{cat}$, with Δ_{cont} the continuous distance as usual and Δ_{cat} the categorical distance defined as the number of categorical variables that differs from one point to another.

For example, the Gower Distance between `[1,'red', 'medium']` and `[1.2,'red', 'large']` is $\Delta = 0.2 + (0$ `'red'` $=$ `'red'` $+1$ `'medium'` $\neq$ `'large'` $) = 1.2$

Example of mixed-integer Gower Distance model

```
from smt.applications.mixed_integer import (
    MixedIntegerSurrogateModel,
    ENUM,
    GOWER,
)
from smt.surrogate_models import KRG
import matplotlib.pyplot as plt
import numpy as np

xt = np.array([0, 2, 4])
yt = np.array([0.0, 1.0, 1.5])

xlimits = [["0.0", "1.0", " 2.0", "3.0", "4.0"]]

# Surrogate
sm = MixedIntegerSurrogateModel(
```

(continues on next page)

(continued from previous page)

```
    categorical_kernel=GOWER,
    xtypes=[(ENUM, 5)],
    xlimits=xlimits,
    surrogate=KRG(theta0=[1e-2]),
)
sm.set_training_values(xt, yt)
sm.train()

# DOE for validation
x = np.linspace(0, 4, 5)
y = sm.predict_values(x)

plt.plot(xt, yt, "o", label="data")
plt.plot(x, y, "d", color="red", markersize=3, label="pred")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
___________________________________________________________________________

 Evaluation

      # eval points. : 5

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```

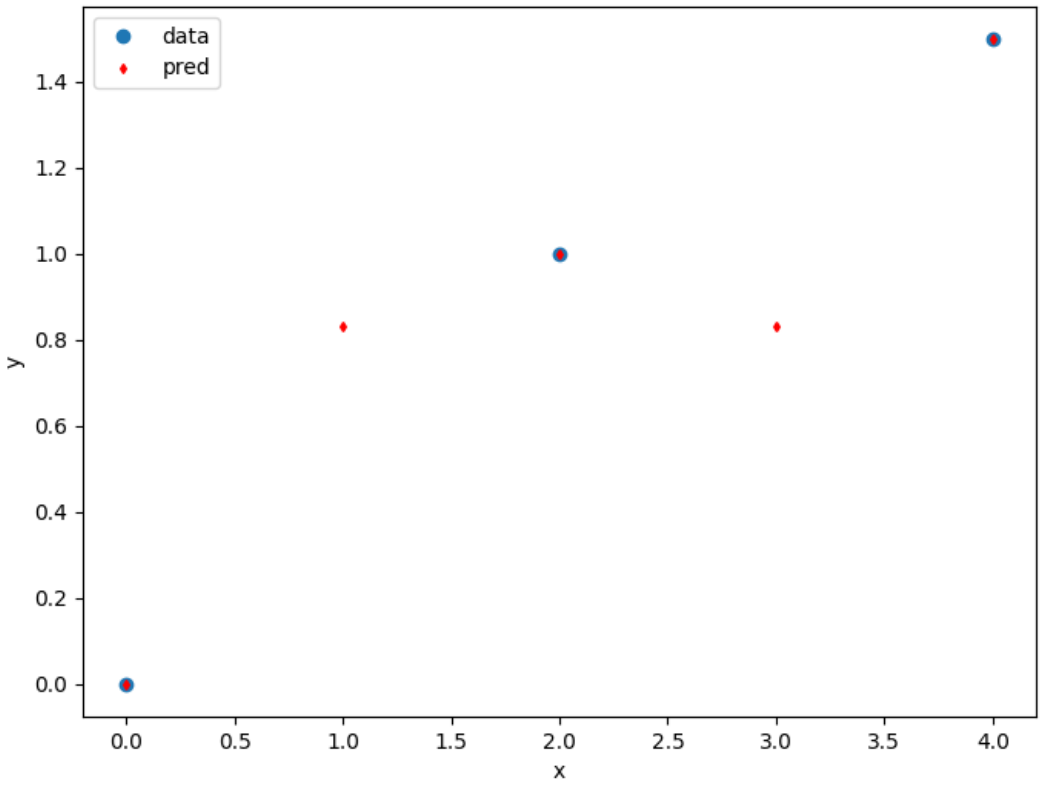
Mixed integer sampling method

To use a sampling method with mixed integer typed features, the user instanciates a `MixedIntegerSamplingMethod` with a given sampling method. The `MixedIntegerSamplingMethod` implements the `SamplingMethod` interface and decorates the original sampling method to provide a DOE while conforming to integer and categorical types.

Example of mixed-integer LHS sampling method

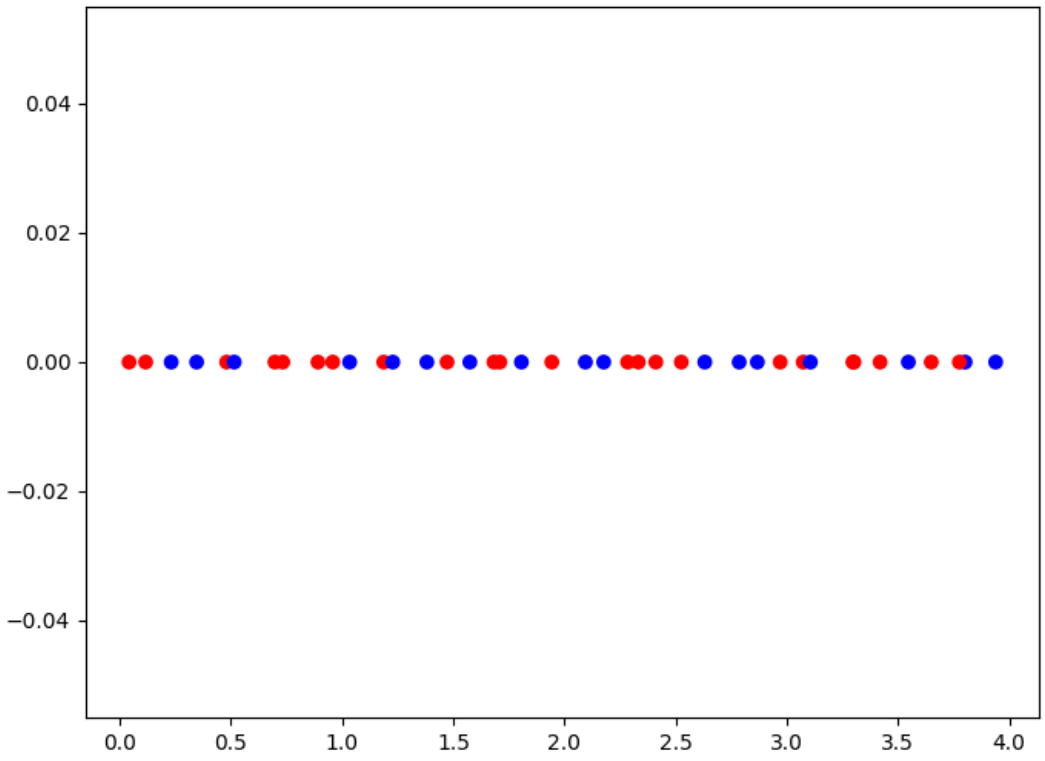
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors

from smt.sampling_methods import LHS
from smt.applications.mixed_integer import (
    FLOAT,
    ORD,
    ENUM,
    MixedIntegerSamplingMethod,
)
```

(continues on next page)


data
pred
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

(continued from previous page)

```
xtypes = [FLOAT, (ENUM, 2)]
xlimits = [[0.0, 4.0], ["blue", "red"]]
sampling = MixedIntegerSamplingMethod(xtypes, xlimits, LHS, criterion="ese")

num = 40
x = sampling(num)

cmap = colors.ListedColormap(xlimits[1])
plt.scatter(x[:, 0], np.zeros(num), c=x[:, 1], cmap=cmap)
plt.show()
```



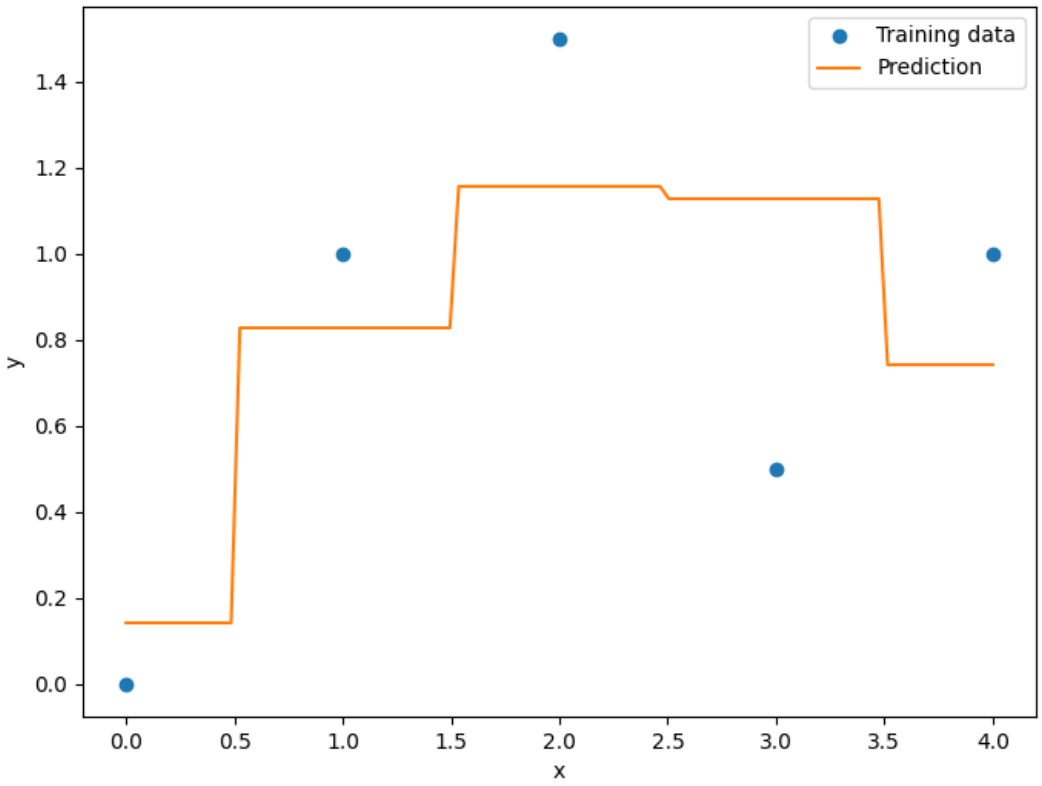
Mixed integer surrogate

To use a surrogate with mixed integer constraints, the user instanciates a `MixedIntegerSurrogateModel` with the given surrogate. The `MixedIntegerSurrogateModel` implements the `SurrogateModel` interface and decorates the given surrogate while respecting integer and categorical types.

Example of mixed-integer Polynomial (QP) surrogate

```
import numpy as np
import matplotlib.pyplot as plt

from smt.surrogate_models import QP
from smt.applications.mixed_integer import MixedIntegerSurrogateModel, ORD

xt = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
yt = np.array([0.0, 1.0, 1.5, 0.5, 1.0])

# xtypes = [FLOAT, ORD, (ENUM, 3), (ENUM, 2)]
# FLOAT means x1 continuous
# ORD means x2 ordered
# (ENUM, 3) means x3, x4 & x5 are 3 levels of the same categorical variable
# (ENUM, 2) means x6 & x7 are 2 levels of the same categorical variable

sm = MixedIntegerSurrogateModel(xtypes=[ORD], xlimits=[[0, 4]], surrogate=QP())
sm.set_training_values(xt, yt)
sm.train()

num = 100
x = np.linspace(0.0, 4.0, num)
y = sm.predict_values(x)

plt.plot(xt, yt, "o")
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.legend(["Training data", "Prediction"])
plt.show()
```

```
___________________________________________________________________________

 Evaluation

      # eval points. : 100

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```


Training data
Prediction
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

Mixed integer context

the `MixedIntegerContext` class helps the user to use mixed integer sampling methods and surrogate models consistently by acting as a factory for those objects given a x specification: (xtypes, xlimits).

class smt.applications.mixed_integer.**MixedIntegerContext**(*xtypes*, *xlimits*, *work_in_folded_space=True*, *categorical_kernel=None*)

Class which acts as sampling method and surrogate model factory to handle integer and categorical variables consistently.

Methods

build_sampling_method(sampling_method_class, ...)	Build MixedIntegerSamplingMethod from given SMT sampling method.
build_surrogate_model(surrogate)	Build MixedIntegerSurrogateModel from given SMT surrogate model.
cast_to_discrete_values(x)	Project continuously relaxed values to their closer assessable values.
cast_to_enum_value(x_col, enum_indexes)	Return enumerate levels from indexes for the given x feature specified by x_col.
cast_to_mixed_integer(x)	Convert an x point with enum indexes to x point with enum levels
fold_with_enum_index(x)	Reduce categorical inputs from discrete unfolded space to initial x dimension space where categorical x dimensions are valued by the index in the corresponding enumerate list.
get_unfolded_dimension()	Returns x dimension (int) taking into account unfolded categorical features
get_unfolded_xlimits()	Returns relaxed xlimits Each level of an enumerate gives a new continuous dimension in [0, 1].
unfold_with_enum_mask(x)	Expand categorical inputs from initial x dimension space where categorical x dimensions are valued by the index in the corresponding enumerate list to the discrete unfolded space.

__init__(*xtypes*, *xlimits*, *work_in_folded_space=True*, *categorical_kernel=None*)

Parameters

xtypes: x types list
x type specification: list of either FLOAT, ORD or (ENUM, n) spec.

xlimits: array-like
bounds of x features

work_in_folded_space: bool
whether x data are in given in folded space (enum indexes) or not (enum masks)

categorical_kernel: string
the kernel to use for categorical inputs. Only for non continuous Kriging.

build_sampling_method(*sampling_method_class*, ***kwargs*)

Build MixedIntegerSamplingMethod from given SMT sampling method.

`build_surrogate_model`(*surrogate*)

Build MixedIntegerSurrogateModel from given SMT surrogate model.

`cast_to_discrete_values`(*x*)

Project continuously relaxed values to their closer assessable values. Note: categorical (or enum) x dimensions are still expanded that is there are still as many columns as categorical possible values for the given x dimension. For instance, if an input dimension is typed ["blue", "red", "green"] in xlimits a sample/row of the input x may contain the values (or mask) [..., 0, 0, 1, ...] to specify "green" for this original dimension.

Parameters

x
[np.ndarray [n_evals, dim]] continuous evaluation point input variable values

Returns

np.ndarray
feasible evaluation point value in categorical space.

`fold_with_enum_index`(*x*)

Reduce categorical inputs from discrete unfolded space to initial x dimension space where categorical x dimensions are valued by the index in the corresponding enumerate list. For instance, if an input dimension is typed ["blue", "red", "green"] a sample/row of the input x may contain the mask [..., 0, 0, 1, ...] which will be contracted in [..., 2, ...] meaning the "green" value. This function is the opposite of unfold_with_enum_mask().

Parameters

x: np.ndarray [n_evals, dim]
continuous evaluation point input variable values

Returns

np.ndarray [n_evals, dim]
evaluation point input variable values with enumerate index for categorical variables

`unfold_with_enum_mask`(*x*)

Expand categorical inputs from initial x dimension space where categorical x dimensions are valued by the index in the corresponding enumerate list to the discrete unfolded space. For instance, if an input dimension is typed ["blue", "red", "green"] a sample/row of the input x may contain [..., 2, ...] which will be expanded in [..., 0, 0, 1, ...]. This function is the opposite of fold_with_enum_index().

Parameters

x: np.ndarray [n_evals, nx]
continuous evaluation point input variable values

Returns

np.ndarray [n_evals, nx continuous]
evaluation point input variable values with enumerate index for categorical variables

`cast_to_mixed_integer`(*x*)

Convert an x point with enum indexes to x point with enum levels

Parameters

x: array-like
point to convert

Returns

x as a list with enum levels if any

cast_to_enum_value(*x_col*, *enum_indexes*)

Return enumerate levels from indexes for the given x feature specified by x_col.

Parameters

x_col: int
index of the feature typed as enum

enum_indexes: list
list of indexes in the possible values for the enum

Returns

list of levels (labels) for the given enum feature

Example of mixed-integer context usage

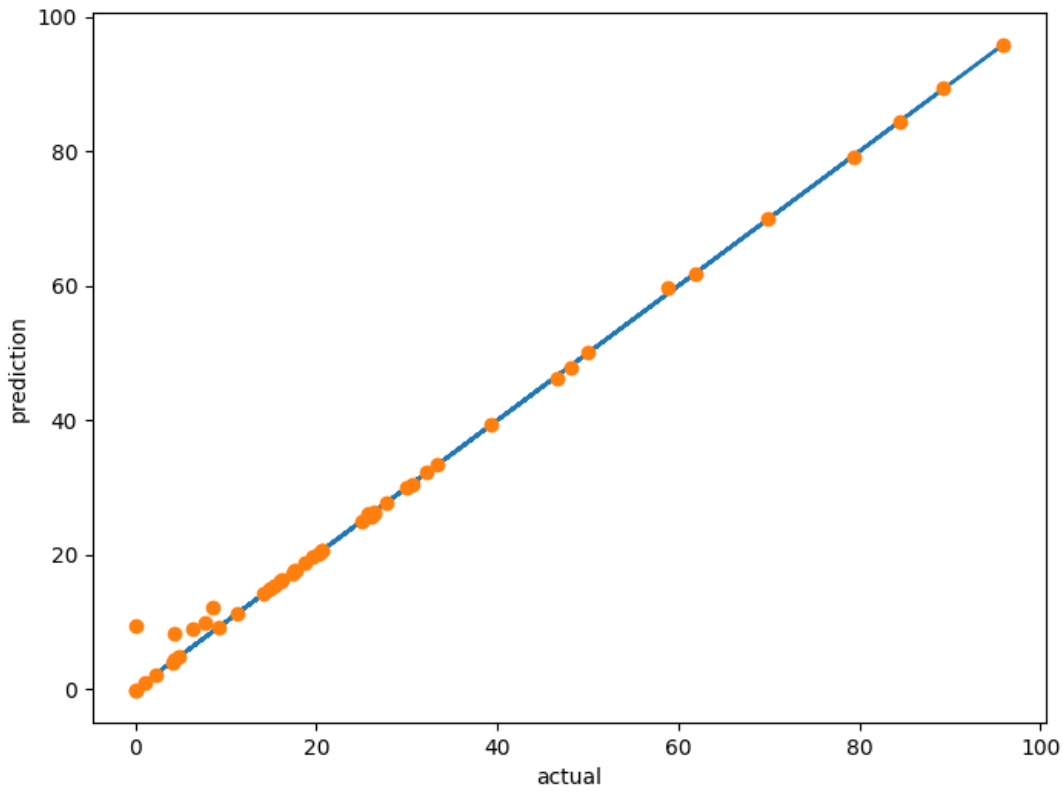
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
from mpl_toolkits.mplot3d import Axes3D

from smt.surrogate_models import KRG
from smt.sampling_methods import LHS, Random
from smt.applications.mixed_integer import MixedIntegerContext, FLOAT, ORD, ENUM

xtypes = [ORD, FLOAT, (ENUM, 4)]
xlimits = [[0, 5], [0.0, 4.0], ["blue", "red", "green", "yellow"]]

def ftest(x):
    return (x[:, 0] * x[:, 0] + x[:, 1] * x[:, 1]) * (x[:, 2] + 1)

# context to create consistent DOEs and surrogate
mixint = MixedIntegerContext(xtypes, xlimits)

# DOE for training
lhs = mixint.build_sampling_method(LHS, criterion="ese")

num = mixint.get_unfolded_dimension() * 5
print("DOE point nb = {}".format(num))
xt = lhs(num)
yt = ftest(xt)

# Surrogate
sm = mixint.build_surrogate_model(KRG())
sm.set_training_values(xt, yt)
sm.train()

# DOE for validation
rand = mixint.build_sampling_method(Random)
xv = rand(50)
yv = ftest(xv)
yp = sm.predict_values(xv)
```

(continues on next page)

(continued from previous page)

```
plt.plot(yv, yv)
plt.plot(yv, yp, "o")
plt.xlabel("actual")
plt.ylabel("prediction")

plt.show()
```

```
DOE point nb = 30
___________________________________________________________________________

 Evaluation

      # eval points. : 50

   Predicting ...
   Predicting - done. Time (sec):  0.0000000

   Prediction time/pt. (sec) :  0.0000000
```



The intent is to provide applications of surrogate models in higher level methods.

3.7 Contributing to SMT

This part of the documentation is meant for developers who want to contribute new surrogate models, problems, or sampling methods.

Contributing to SMT consists of the following steps:

- Fork SMT to make a version of the SMT repo separate from the main one.
- Clone *your* SMT repo and install in development mode: go in your local smt folder and run `pip install -e .`
- Write the class following the developer API given in the section below, and add it to the right folder, e.g., in `smt/surrogate_models/`.
- Add the import statement in the corresponding `__init__.py` file, e.g., `smt/surrogate_models/__init__.py`.
- Add tests to the top-level `tests` directory following the existing examples and run tests (see *Testing* section below)
- Add a documentation page in the appropriate directory, e.g., `doc/_src_docs/surrogate_models/rbf.rstx`, using the existing docs as a reference (see *Building the documentation* section below).
- Add an entry in the table of contents so that readers can find the documentation page, e.g., in `doc/_src_docs/surrogate_model.rstx`.
- Test and commit the changes, push to the forked version of SMT and issue a pull request for review and comments from the other developers of SMT and the larger community

3.7.1 Developer API

Developer API for surrogate models

class smt.surrogate_models.surrogate_model.**SurrogateModel**(***kwargs*)

Base class for all surrogate models.

Examples

```
>>> from smt.surrogate_models import RBF
>>> sm = RBF(print_training=False)
>>> sm.options['print_prediction'] = False
```

Attributes

options
[OptionsDictionary] Dictionary of options. Options values can be set on this attribute directly or they can be passed in as keyword arguments during instantiation.

supports
[dict] Dictionary containing information about what this surrogate model supports.

Methods

predict_derivatives(x, kx)	Predict the dy_dx derivatives at a set of points.
predict_output_derivatives(x)	Predict the derivatives dy_dyt at a set of points.
predict_values(x)	Predict the output values at a set of points.
predict_variance_derivatives(x)	Predict the derivation of the variance at a point
predict_variances(x)	Predict the variances at a set of points.
set_training_derivatives(xt, dyt_dxt, kx[, name])	Set training data (derivatives).
set_training_values(xt, yt[, name])	Set training data (values).
train()	Train the model
update_training_derivatives(dyt_dxt, kx[, name])	Update the training data (values) at the previously set input values.
update_training_values(yt[, name])	Update the training data (values) at the previously set input values.

_initialize()

Implemented by surrogate models to declare options and declare what they support (optional).

Examples

self.options.declare('option_name', default_value, types=(bool, int), desc='description')
self.supports['derivatives'] = True

_train() → None

Implemented by surrogate models to perform training (optional, but typically implemented).

abstract _predict_values(*x: ndarray*) → ndarray

Implemented by surrogate models to predict the output values.

Parameters

x
[np.ndarray[nt, nx]] Input values for the prediction points.

Returns

y
[np.ndarray[nt, ny]] Output values at the prediction points.

_predict_derivatives(*x: ndarray*, *kx: int*) → ndarray

Implemented by surrogate models to predict the dy_dx derivatives (optional).

If this method is implemented, the surrogate model should have

::
self.supports['derivatives'] = True

in the _initialize() implementation.

Parameters

x
[np.ndarray[nt, nx]] Input values for the prediction points.

kx
[int] The 0-based index of the input variable with respect to which derivatives are desired.

Returns

dy_dx
[np.ndarray[nt, ny]] Derivatives.

_predict_output_derivatives(*x: ndarray*) → dict

Implemented by surrogate models to predict the dy_dyt derivatives (optional).

If this method is implemented, the surrogate model should have

::
self.supports['output_derivatives'] = True

in the _initialize() implementation.

Parameters

x
[np.ndarray[nt, nx]] Input values for the prediction points.

Returns

dy_dyt
[dict of np.ndarray[nt, nt]] Dictionary of output derivatives. Key is None for derivatives wrt yt and kx for derivatives wrt dyt_dxt.

_predict_variances(*x: ndarray*) → ndarray

Implemented by surrogate models to predict the variances at a set of points (optional).

If this method is implemented, the surrogate model should have

::
self.supports['variances'] = True

in the _initialize() implementation.

Parameters

x
[np.ndarray[nt, nx]] Input values for the prediction points.

Returns

s2
[np.ndarray[nt, ny]] Variances.

Developer API for benchmarking problems

class smt.problems.problem.**Problem**(***kwargs*)

Methods

__call__(x[, kx])	Evaluate the function.

_initialize() → None

Implemented by problem to declare options (optional).

Examples

self.options.declare('option_name', default_value, types=(bool, int), desc='description')

_evaluate(*x: ndarray*, *kx: Optional[int] = None*) → ndarray

Implemented by surrogate models to evaluate the function.

Parameters

x
[ndarray[n, nx]] Evaluation points where n is the number of evaluation points.

kx
[int or None] Index of derivative (0-based) to return values with respect to. None means return function value rather than derivative.

Returns

ndarray[n, 1]
Functions values if kx=None or derivative values if kx is an int.

Developer API for sampling methods

SamplingMethod

A base class for all sampling methods in SMT.

class smt.sampling_methods.sampling_method.**SamplingMethod**(***kwargs*)

Methods

__call__(nt)	Compute the requested number of sampling points.

_initialize() → None

Implemented by sampling methods to declare options (optional).

Examples

self.options.declare('option_name', default_value, types=(bool, int), desc='description')

abstract _compute(*nt: int*) → ndarray

Implemented by sampling methods to compute the requested number of sampling points.

The number of dimensions (nx) is determined based on *xlimits.shape[0]*.

Returns

ndarray[nt, nx]
The sampling locations in the input space.

ScaledSamplingMethod

Conveniently, if a sampling method generates samples in the [0, 1] hypercube, one can inherit from the subclass *ScaledSamplingMethod* which automates the scaling from unit hypercube to the input space (i.e. xlimits).

class smt.sampling_methods.sampling_method.**ScaledSamplingMethod**(***kwargs*)

This class describes an sample method which generates samples in the unit hypercube.

The __call__ method does scale the generated samples accordingly to the defined xlimits.

Methods

__call__(nt)	Compute the requested number of sampling points.

abstract _compute(*nt: int*) → ndarray

Implemented by sampling methods to compute the requested number of sampling points.

The number of dimensions (nx) is determined based on *xlimits.shape[0]*.

Returns

ndarray[nt, nx]
The sampling locations in the unit hypercube.

3.7.2 Testing

Install the test runner: `pip install pytest` then run: `pytest`

3.7.3 Building the documentation

Users can read the docs online at `smt.readthedocs.io`, but developers who contribute to the docs should build the docs locally to view the output. This is especially necessary because most of the docs in SMT contain code, code print output, and plots that are dynamically generated and embedded during the doc building process. The docs are written using reStructuredText, and there are a few custom directives we have added for this embedding of dynamically-generated content.

First, install *sphinx_auto_embed* by running `pip install git+https://github.com/hwangjt/sphinx_auto_embed.git`.

To build the docs, the developer should go to the `doc` directory and run `sphinx_auto_embed` and `make html` to build the html docs. This is a 2-step process because `sphinx_auto_embed` converts rstx files to rst files and `make html` generates the html docs from the rst files. The landing page for the built docs can then be found at `doc/_build/html/index.html`, and this is the same page that readers first see when they load `smt.readthedocs.io`.

3.8 Indices and tables

- genindex
- search

INDEX

Symbols

B

C

F

M

P

S

T

U