This is a preprint of the following article, which is available from http://mdolab.engin.umich.edu
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 EngineeringSoftware*, 2019.
The published article may differ from this preprint, and is available by following the DOI: https://doi.org/10.1016/j.advengsoft.2019.03.005.

A Python surrogate modeling framework with derivatives

Mohamed Amine Bouhlel
Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI, USA

John T. Hwang
University of California San Diego, Department of Mechanical and Aerospace Engineering, La Jolla, CA, USA

Nathalie Bartoli and Rémi Lafage
ONERA/DTIS, Université de Toulouse, Toulouse, France

Joseph Morlier
ICA, Université de Toulouse, ISAE–SUPAERO, INSA, CNRS, MINES ALBI, UPS, Toulouse, France

Joaquim R. R. A. Martins
Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI, USA

Abstract

The surrogate modeling toolbox (SMT) is an open-source Python package that contains a collection of surrogate modeling methods, sampling techniques, and benchmarking functions. This package provides a library of surrogate models that is simple to use and facilitates the implementation of additional methods. SMT is different from existing surrogate modeling libraries because of its emphasis on derivatives, including training derivatives used for gradient-enhanced modeling, prediction derivatives, and derivatives with respect to training data. It also includes unique surrogate models: kriging by partial least-squares reduction, which scales well with the number of inputs; and energy-minimizing spline interpolation, which scales well with the number of training points. The efficiency and effectiveness of SMT are demonstrated through a series of examples. SMT is documented using custom tools for embedding automatically tested code and dynamically generated plots to produce high-quality user guides with minimal effort from contributors. SMT is maintained in a public version control repository[1].

[1] https://github.com/SMTorg/SMT

1 Motivation and significance

In the last few decades, numerical models for engineering have become more complex and accurate, but the computational time for running these models has not necessarily decreased. This makes it difficult to complete engineering tasks that rely on these models, such as design space exploration and optimization. Surrogate modeling is often used to reduce the computational time of these tasks by replacing expensive numerical simulations with approximate functions that are much faster to evaluate. Surrogate models are constructed by evaluating the original model at a set of points, called training points, and using the corresponding evaluations to construct an approximate model based on mathematical functions.

Surrogate modeling is often used in the context of design optimization because of the repeated model evaluations that are required. Derivatives play an important role in optimization, because problems with a large number of optimization variables require gradient-based algorithms for efficient scalability. Therefore, situations frequently arise where there are requirements for surrogate models associated with the computation or use of derivatives.

There are three types of derivatives in surrogate modeling: prediction derivatives, training derivatives, and output derivatives. Prediction derivatives are the derivatives of the surrogate model outputs with respect to the inputs, and they are the derivatives required when using a surrogate model in gradient-based optimization. Training derivatives are derivatives of the training outputs with respect to the training inputs that provide additional training data that increase the accuracy of the surrogate model. Output derivatives are derivatives of the prediction outputs with respect to the training outputs, which are required if a surrogate model is reconstructed within a gradient-based optimization process. We describe these derivatives in more detail in Section 3.3.

Various packages that build surrogate models have been developed using different programming languages, such as Scikit-learn in Python [35], SUMO in MATLAB [13], and GPML in MATLAB and Octave [38]. However, these packages do not handle the derivatives.

In this paper, we introduce a new Python package called the surrogate modeling toolbox (SMT). SMT is different from existing surrogate modeling libraries because of its emphasis on derivatives, including all three types of derivatives described above. SMT also includes newly developed surrogate models that handle derivatives and do not exist elsewhere: partial least-squares (PLS)-based surrogate models [5–7], which are suitable for high-dimensional problems, and the regularized minimal-energy tensor-product spline (RMTS), which is suitable for low-dimensional problems with up to hundreds of thousands of sampling points [18]. To use SMT, the user should first provide a set of training points. This could be done either by using the sampling techniques and benchmarking functions implemented in SMT or by directly importing the data. Then, the user can build a surrogate model based on the training points and make predictions for the function values and derivatives.

The main goal of this work is to provide a simple Python toolbox that contains a set of surrogate modeling functions and supports different kinds of derivatives that are useful for many applications in engineering. SMT includes sampling techniques, which

are necessary for the construction of a surrogate model. Various sample functions are also included to facilitate the benchmarking of different techniques and for reproducing results. SMT is suitable for both novice and advanced users and is supported by detailed documentation available online with examples of each implemented surrogate modeling approach[2]. It is hosted publicly in a version-controlled repository[3], and is easily imported and used within Python scripts. It is released under the New BSD License and runs on Linux, macOS, and Windows operating systems. Regression tests are run automatically on each operating system whenever a change is committed to the repository.

The remainder of this paper is organized as follows. First, we present the architecture and the main implementation features of SMT in Section 2 and then we describe the methods implemented in SMT in Section 3. Section 4 gives an example of SMT usage and presents a set of benchmarking functions implemented within SMT. We apply SMT to two engineering problems in Section 5 and present the conclusions in Section 6.

2 Software architecture, documentation, and automatic testing

SMT is composed of three main modules (*sampling_methods*, *problems*, and *surrogate_models*) that implement a set of sampling techniques, benchmarking functions, and surrogate modeling techniques, respectively. Each module contains a common interface inherited by the corresponding methods, and each method implements the functions required by the interface, as shown in Figure 1.

SMT's documentation is written using reStructuredText and is generated using the Sphinx package for documentation in Python, along with custom extensions[4]. The documentation pages include embedded code snippets that demonstrate the usage. These code snippets are extracted dynamically from actual tests in the source code, ensuring that the code snippets are always up to date. The print output and plots from the code snippets are also generated dynamically by custom Sphinx extensions and embedded in the documentation page. This leads to high-quality documentation with low effort, requiring only the addition of a custom directive and the path to locating the test code. Similarly, another custom directive embeds a table of options, default values, valid types, valid values, and descriptions in the documentation for the surrogate model, sampling method, or benchmarking problem. The documentation also uses existing Sphinx directives to embed descriptions of the user-callable methods in the classes.

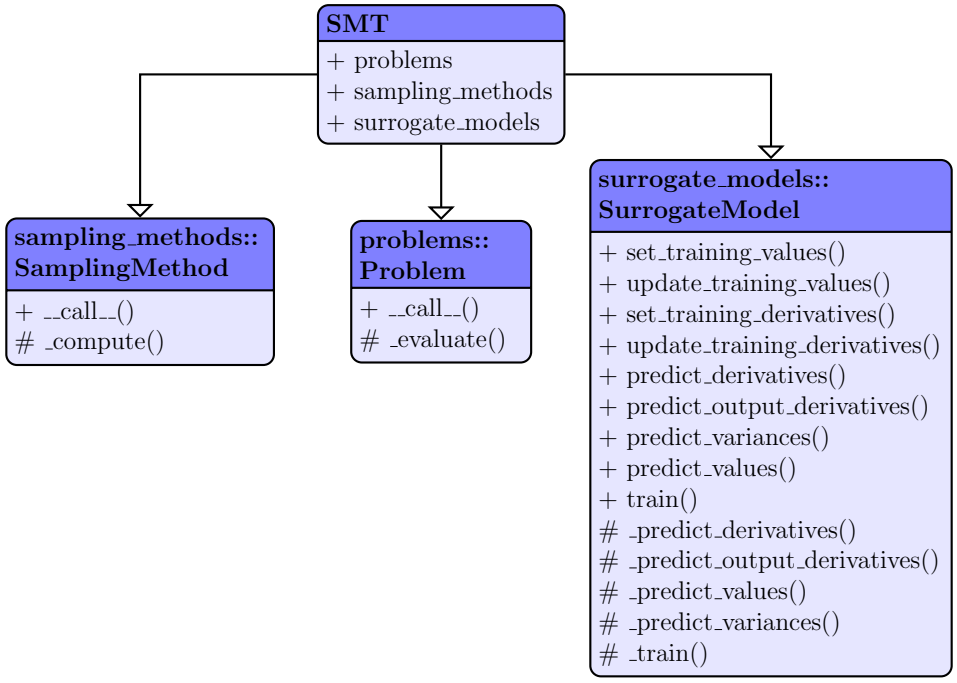
In addition to the user documentation, we also provide developer documentation that explains how to contribute code to SMT. The developer documentation includes a different list of application programming interface (API) methods for the *SurrogateModel*, *SamplingMethod*, and *Problem* classes, which are classes that must be imple-

[2] http://smt.readthedocs.io/en/latest

[3] https://github.com/SMTorg/smt

[4] https://smt.readthedocs.org

Figure 1: Architecture of SMT.



mented to create a new surrogate modeling method, sampling technique, or benchmarking problem, respectively.

When a developer issues a pull request, the request is merged once the automatically triggered tests run successfully and at least one reviewer approves it. The repository on GitHub[5] is linked to two continuous integration testing services, Travis CI (for Linux and macOS) and AppVeyor (for Windows), which trigger test suite execution whenever code is committed and prevent changes from being merged if any of the tests fail.

3 Surrogate modeling methods

To build a surrogate model, two main steps are necessary. First, we generate a set of training points from the input space where the quantity of interest is computed. This step can be done by using one of the sampling techniques implemented in SMT (Section 3.1), or by uploading an existing training dataset. Second, we train the desired surrogate model on those points and make a prediction of the output values and derivatives (Section 3.2).

3.1 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.

Random sampling: this class creates random samples from a uniform distribution

[5] https://github.com/SMTorg/smt

over the design space.

Latin hypercube sampling: this 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 in high-dimensional problems. Five construction criteria are available in SMT: four criteria defined in the *pyDOE* package[6] and the enhanced stochastic evolutionary criterion [21].

Full-factorial sampling: this is a common sampling method where all input variables are set at two levels each. These levels are usually denoted by +1 and -1 for *high* and *low* levels, respectively. Full-factorial sampling computes all possible combinations of these levels across all such input variables.

Figure 2 shows the implementation of the *Random* class, which inherits from the *SamplingMethod* class.

Figure 2: Implementation of the *Random* class, which inherits from the *SamplingMethod* class. The `compute` function evaluates the requested number of sampling points uniformly over the design space.

```
import numpy as np
from six.moves import range
from smt.sampling_methods.sampling_method
               import SamplingMethod

class Random(SamplingMethod):

    def _compute(self, n):
        """
        Compute the requested number of sampling points.
        Arguments
        ---------
        n : int
            Number of points requested.
        Returns
        -------
        ndarray[n, nx]
            The sampling locations in the input space.
        """
        xlimits = self.options['xlimits']
        nx = xlimits.shape[0]
        return np.random.rand(n, nx)
```

3.2 Surrogate models

Table 1 lists the surrogate modeling methods currently available in SMT and summarizes the advantages and disadvantages of each method. The methods include both well-established methods and methods recently developed by the authors that use derivative information. Among the well-established methods, we implement kriging [39], radial basis functions (RBF) [36], inverse distance weighting (IDW) [41], least

[6] https://pythonhosted.org/pyDOE/randomized.html

squares (LS) [16, ch. 3], and quadratic polynomials (QP) [16, ch. 3]. In Figure 3, we show the implementation of the *RBF* class, which inherits from the *SurrogateModel* class.

The methods that we recently developed are kriging combined with partial least squares (KPLS) [7], KPLSK for the construction of a standard kriging model in high-dimensional problems [6], gradient-enhanced KPLS (GE-KPLS) [5], and RMTS [18].

Table 1: Surrogate modeling methods provided by SMT.

Method	Advantages (+) and disadvantages (−)	Derivatives			References
		Train.	Pred.	Out.	
Kriging	+ Prediction variance, flexible − Costly if number of inputs or training points is large − Numerical issues when points are too close to each other	No	Yes	No	Sacks et al. [39]
KPLS	+ Prediction variance, fast construction + Suitable for high-dimensional problems − Numerical issues when points are too close to each other	No	Yes	No	Bouhlel et al. [7]
KPLSK	+ Prediction variance, fast construction + Suitable for high-dimensional problems − Numerical issues when points are too close to each other	No	Yes	No	Bouhlel et al. [6]
GE-KPLS	+ Prediction variance, fast construction + Suitable for high-dimensional problems + Control of the correlation matrix size − Numerical issues when points are too close to each other − Choice of step parameter is not intuitive	Yes	Yes	No	Bouhlel and Martins [5]
RMTS	+ Fast prediction + Training scales well up to 10^5 training points + No issues with points that are too close to each other − Poor scaling with number of inputs above 4 − Slow training overall	Yes	Yes	Yes	Hwang and Martins [18]
RBF	+ Simple, only a single tuning parameter + Fast training for small number of training points − Susceptible to oscillations − Numerical issues when points are too close to each other	No	Yes	Yes	Powell [36]
IDW	+ Simple, no training required − Derivatives are zero at training points − Poor overall accuracy	No	Yes	Yes	Shepard [41]
LS	+ Simple, fast construction − Accurate only for linear problems	No	Yes	No	Hastie et al. [16]
QP	+ Simple, fast construction − Large number of points required for large number of inputs	No	Yes	No	Hastie et al. [16]

Surrogate models provide a vector of prediction outputs $\mathbf{y}$ for a given vector of prediction inputs $\mathbf{x} \in \mathbb{R}^{n_x}$, and can be expressed as

$$\mathbf{y} = f\left(\mathbf{x}, \mathbf{x}_t, \mathbf{y}_t\right), \tag{1}$$

where $\mathbf{x}_t \in \mathbb{R}^{n_x}$ and $\mathbf{y}_t$ are the vectors of training inputs and outputs, respectively, which are used to build the surrogate model a priori, and $\mathbf{x}$ is the unknown point to predict with the surrogate model. We now describe the surrogate modeling methods available in SMT.

Figure 3: Implementation of the *RBF* class, which inherits from the *SurrogateModel* class.

```
from smt.surrogate_models.surrogate_model
          import SurrogateModel

class RBF(SurrogateModel):
  def _initialize(self):
    super(RBF, self)._initialize()
    ...

  def _setup(self):
    options = self.options
    ...

  def _train(self):
    self._setup()
    ...

  def _predict_values(self, x):
    """
    Evaluates the model at a set of points.
    x : np.ndarray [n_evals, dim]
      Evaluation point input values
    y (output): Evaluation point output values
    """
    n = x.shape[0]
    ...

  def _predict_derivatives(self, x, kx):
    """
    Evaluates the derivatives at a set of points.
    x : np.ndarray [n_evals, dim]
      Evaluation point input variable values
    kx : int
      The 0-based index of the input variable with
      respect to which derivatives are desired.
    dy_dx (output): Derivative values.
    """
    n = x.shape[0]
    ...

  def _predict_output_derivatives(self, x):
    """
    Evaluates the output derivatives at a set of points.
    x : np.ndarray [n_evals, dim]
      Evaluation point input variable values
    dy_dyt (output): Output derivative values.
    """
    n = x.shape[0]
    ...
```

3.2.1 LS and QP

The LS method [16] fits a linear model with coefficients $\beta = (\beta_0, \beta_1, \ldots, \beta_{n_x})$, where n_x is the number of dimensions, to minimize the residual sum of the squares between the observed responses in the dataset, and the responses predicted by the linear approximation, i.e.,

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

where $\mathbf{X} = \left(1, \mathbf{x}^{(1)^T}, \ldots, \mathbf{x}^{(n_t)^T}\right)^T$. The vectors in $\mathbf{X}$ have $(n_t \times n_x + 1)$ dimensions and n_t is the number of training points. The square polynomial model is given by [16]

$$\mathbf{y} = \mathbf{X}\beta + \epsilon, \tag{3}$$

where ϵ is a vector of random errors and

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & \ldots & x_{n_x}^{(1)} & x_1^{(1)}x_2^{(1)} & \ldots & x_{n_x-1}^{(1)}x_{n_x}^{(1)} & {x_1^{(1)}}^2 & \ldots & {x_{n_x}^{(1)}}^2 \\ \vdots & \vdots & \ldots & \vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \\ 1 & x_1^{(n_t)} & \ldots & x_{n_x}^{(n_t)} & x_1^{(n_t)}x_2^{(n_t)} & \ldots & x_{n_x-1}^{(n_t)}x_{n_x}^{(n_t)} & {x_1^{(n_t)}}^2 & \ldots & {x_{n_x}^{(n_t)}}^2 \end{bmatrix}. \tag{4}$$

The vector of estimated polynomial regression coefficients using ordinary LS estimation is

$$\beta = \mathbf{X}^T\mathbf{X}^{-1}\mathbf{X}^T\mathbf{y}. \tag{5}$$

3.2.2 IDW

The IDW model is an interpolating method where the unknown points are calculated with a weighted average of the sampling points using [41]

$$y = \begin{cases} \dfrac{\sum_i^{n_t} \beta\left(\mathbf{x}, \mathbf{x}_t^{(i)}\right) y_t^{(i)}}{\sum_i^{n_t} \beta\left(\mathbf{x}, \mathbf{x}_t^{(i)}\right)}, & \text{if} \quad \mathbf{x} \neq \mathbf{x}_t^{(i)} \quad \forall i \\ y_t^{(i)}, & \text{if} \quad \mathbf{x} = \mathbf{x}_t^{(i)} \quad \text{for some } i \end{cases}, \tag{6}$$

where $\mathbf{x} \in \mathbb{R}^{n_x}$ is the prediction input vector, $y \in \mathbb{R}$ is the prediction output, $\mathbf{x}_t^{(i)} \in \mathbb{R}^{n_x}$ is the input vector for the ith training point, and $y_t^{(i)} \in \mathbb{R}$ is the output value for the ith training point. The weighting function β is defined as

$$\beta\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^{-p}, \tag{7}$$

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

3.2.3 RBF

The RBF surrogate model [36] represents the interpolating function as a linear combination of basis functions, one for each training point. RBF 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. RBF are frequently augmented to global polynomials to capture the general trends. The prediction equation for RBF is given by

$$y = \mathbf{p}\left(\mathbf{x}\right)\mathbf{w_p} + \sum_i^{n_t} \phi\left(\mathbf{x}, \mathbf{x}_t^{(i)}\right)\mathbf{w_r}, \tag{8}$$

where $\mathbf{x} \in \mathbb{R}^{n_x}$ is the prediction input vector, $y \in \mathbb{R}$ is the prediction output, $\mathbf{x}_t^{(i)} \in \mathbb{R}^{n_x}$ is the input vector for the ith training point, $\mathbf{p}(\mathbf{x}) \in \mathbb{R}^{n_p}$ is the vector mapping the polynomial coefficients to the prediction output, $\phi\left(\mathbf{x}, \mathbf{x}_t^{(i)}\right) \in \mathbb{R}^{n_t}$ is the vector mapping the RBF coefficients to the prediction output, $\mathbf{w_p} \in \mathbb{R}^{n_p}$ is the vector of polynomial coefficients, and $\mathbf{w_r} \in \mathbb{R}^{n_t}$ is the vector of RBF coefficients. The coefficients, $\mathbf{w_p}$ and $\mathbf{w_r}$, are computed by solving the following linear augmented Gram system:

$$\begin{bmatrix} \phi\left(\mathbf{x}_t^{(1)}, \mathbf{x}_t^{(1)}\right) & \dots & \phi\left(\mathbf{x}_t^{(1)}, \mathbf{x}_t^{(n_t)}\right) & \mathbf{p}\left(\mathbf{x}_t^{(1)}\right)^T \\ \vdots & \ddots & \vdots & \vdots \\ \phi\left(\mathbf{x}_t^{(n_t)}, \mathbf{x}_t^{(1)}\right) & \dots & \phi\left(\mathbf{x}_t^{(n_t)}, \mathbf{x}_t^{(n_t)}\right) & \mathbf{p}\left(\mathbf{x}_t^{(n_t)}\right)^T \\ \mathbf{p}\left(\mathbf{x}_t^{(1)}\right) & \dots & \mathbf{p}\left(\mathbf{x}_t^{(n_t)}\right) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w_r}_1 \\ \vdots \\ \mathbf{w_r}_{n_t} \\ \mathbf{w_p} \end{bmatrix} = \begin{bmatrix} y_t^{(1)} \\ \vdots \\ y_t^{(n_t)} \\ 0 \end{bmatrix}. \tag{9}$$

Only Gaussian basis functions are implemented currently. These can be written as follows

$$\phi\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \exp\left(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2}{d_0^2}\right), \tag{10}$$

where d_0 is a scaling parameter.

3.2.4 Kriging-based models

Kriging [39] is an interpolating model that is a linear combination of a known function $f_i(\mathbf{x})$ added to a realization of a stochastic process, $Z(\mathbf{x})$, to obtain

$$\hat{y} = \sum_{i=1}^{k} \beta_i f_i(\mathbf{x}) + Z(\mathbf{x}), \tag{11}$$

where β_i is the ith linear regression coefficients and k is the number of linear regression coefficients to be determined. Here, $Z(\mathbf{x})$ has a mean of zero and a spatial covariance function given by

$$\operatorname{cov}\left[Z\left(\mathbf{x}^{(i)}\right), Z\left(\mathbf{x}^{(j)}\right)\right] = \sigma^2 R\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right), \tag{12}$$

where σ^2 is the process variance and R is the correlation. Two types of correlation functions are available in SMT: the exponential (Ornstein–Uhlenbeck process)

$$R\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \prod_{l=1}^{n_x} \exp\left(-\theta_l \left|x_l^{(i)} - x_l^{(j)}\right|\right),$$

and the Gaussian correlation function

$$R\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \prod_{l=1}^{n_x} \exp\left(-\theta_l \left(x_l^{(i)} - x_l^{(j)}\right)^2\right),$$

where $\theta_l \in \mathbb{R}^+$. The number of hyperparameters θ is equal to the number of variables n_x. They are estimated by maximizing the likelihood function using the gradient-free optimization algorithm COBYLA [37]. These two correlation functions are called

abs_exp (exponential) and *squar_exp* (Gaussian) in SMT. The deterministic term, the sum in the kriging prediction (11), is replaced by a constant, a linear model, or a quadratic model, all of which are available in SMT.

KPLS [7] is a kriging model that uses the PLS method. KPLS is faster than kriging because fewer hyperparameters need to be estimated to achieve high accuracy. This model is suitable for high-dimensional problems owing to the kernel constructed through the PLS method. The PLS method 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 *principal components*. The PLS information is 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 in KPLS is much lower than n_x. For example, the PLS-Gaussian correlation function is

$$R\left(\mathbf{x}^{(i)},\mathbf{x}^{(j)}\right)=\prod_{l=1}^{n_x}\prod_{k=1}^{h}\exp\left(-\theta_k w_l^{(k)^2}\left(x_l^{(i)}-x_l^{(j)}\right)^2\right). \tag{13}$$

We provide more details on the KPLS method in previous work [7].

The KPLSK model is built in two steps [6]. The first step is to run KPLS and estimate the hyperparameters expressed in the reduced space with h dimensions. The second step is to express the estimated hyperparameters in the original space with n_x dimensions, and then use this as a starting point to optimize the likelihood function of a standard kriging model. The idea here is to guess a "good" initial hyperparameter and apply a gradient-based optimization using a classic kriging kernel. This guess is provided by the KPLS construction: the solutions $(\theta_1^*,\ldots,\theta_h^*)$ and the PLS-coefficients $\left(w_1^{(k)},\ldots,w_{n_x}^{(k)}\right)$ for $k=1,\ldots,h$. Using the change of variables

$$\eta_l=\sum_{k=1}^{h}\theta_k^* w_l^{(k)^2} \tag{14}$$

for $l=1,\ldots,n_x$, we express the initial hyperparameters point in the original space. We demonstrate this using a KPLS-Gaussian correlation function R_{KPLS} in the following example [6]:

$$\begin{aligned}
R_{\text{KPLS}}\left(\mathbf{x}^{(i)},\mathbf{x}^{(j)}\right)&=\prod_{k=1}^{h}\prod_{l=1}^{n_x}\exp\left(-\theta_k w_l^{(k)^2}\left(x_l^{(i)}-x_l^{(j)}\right)^2\right)\\
&=\exp\left(\sum_{l=1}^{n_x}\sum_{k=1}^{h}-\theta_k w_l^{(k)^2}\left(x_l^{(i)}-x_l^{(j)}\right)^2\right)\\
&=\exp\left(\sum_{l=1}^{n_x}-\eta_l\left(x_l^{(i)}-x_l^{(j)}\right)^2\right)\\
&=\prod_{l=1}^{n_x}\exp\left(-\eta_l\left(x_l^{(i)}-x_l^{(j)}\right)^2\right),
\end{aligned} \tag{15}$$

where the change of variables is performed on the third line, and the last line is a standard Gaussian correlation function. The hyperparameters point (14) provides a starting point for a gradient-based optimization applied on a standard kriging method.

3.2.5 Gradient-enhanced KPLS models

GE-KPLS [5] is a gradient-enhanced kriging (GEK) model with the PLS approach. GEK is an extension of kriging that exploits gradient information. 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 primarily owing to the size of the corresponding correlation matrix that increases in proportion to both the number of inputs and the number of sampling points.

To address these issues, GE-KPLS exploits the gradient information with a slight increase in the size of the correlation matrix and reduces the number of hyperparameters. The key idea of GE-KPLS is to generate a set of approximating points around each sampling point using a first-order Taylor approximation. 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 contribute to each variable nearby the associated sampling point to the output. Finally, an average of all PLS coefficients is computed to estimate the global influence to the output. Denoting these coefficients by $\left(w_1^{(k)}, \ldots, w_{n_x}^{(k)}\right)$, the GE-KPLS Gaussian correlation function is given by

$$R_{\text{GE-KPLS}}\left(\mathbf{x^{(i)}}, \mathbf{x^{(j)}}\right) = \sigma \prod_{l=1}^{n_x} \prod_{k=1}^{h} \exp\left(-\theta_k \left(w_l^{(k)} x_l^{(i)} - w_l^{(k)} x_l^{(j)}\right)^2\right). \tag{16}$$

This approach reduces the number of hyperparameters (reduced dimension) from n_x to h, where $h << n_x$.

As mentioned previously, 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 ($m \in [1, n_x]$) around each sampling point. Only the m highest coefficients given by the first principal component are considered, which usually capture the most useful information. Bouhlel and Martins [5] provide more details on this approach.

3.2.6 RMTS method

The RMTS model is a type of surrogate model for low-dimensional problems with large datasets that has fast prediction capability [18]. The underlying mathematical functions are tensor-product splines, which limits RMTS to up to four-dimensional problems, or five-dimensional problems in certain cases. On the other hand, tensor-product splines enable a fast prediction time that does not increase with the number of training points. Unlike other methods, such as kriging and RBF, RMTS is not susceptible to numerical issues when there is a large number of training points or when points are too close together. The prediction equation for RMTS is given by

$$y = \mathbf{F}\left(\mathbf{x}\right)\mathbf{w}, \tag{17}$$

where $\mathbf{x} \in \mathbb{R}^{n_x}$ is the prediction input vector, $y \in \mathbb{R}$ is the prediction output, $\mathbf{w} \in \mathbb{R}^{n_w}$ is the vector of spline coefficients, and $\mathbf{F}(\mathbf{x}) \in \mathbb{R}^{n_w}$ 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 with the second derivatives of the splines, a term that represents the approximation error for the training points, and another term for regularization. Thus, this optimization problem can be written as

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} + \frac{1}{2}\beta\mathbf{w}^T\mathbf{w} + \frac{1}{2}\frac{1}{\alpha}\sum_{i}^{n_t}\left[\mathbf{F}\left(\mathbf{x}_t^{(i)}\right)\mathbf{w} - y_t^{(i)}\right]^2, \tag{18}$$

where $\mathbf{x}_t^{(i)} \in \mathbb{R}^{n_x}$ is the input vector for the ith training point, $y_t^{(i)} \in \mathbb{R}$ is the output value for the ith training point, $\mathbf{H} \in \mathbb{R}^{n_w \times n_w}$ is the matrix of second derivatives, $\mathbf{F}\left(\mathbf{x}_t^{(i)}\right) \in \mathbb{R}^{n_w}$ is the vector mapping the spline coefficients to the ith training output, and α and β are the 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 and can be set to zero 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. This can be addressed by increasing the quadratic approximation term to one of higher order and using Newton's method to solve the resulting nonlinear system. The nonlinear formulation is given by

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} + \frac{1}{2}\beta\mathbf{w}^T\mathbf{w} + \frac{1}{2}\frac{1}{\alpha}\sum_{i}^{n_t}\left[\mathbf{F}\left(\mathbf{x}_t^{(i)}\right)\mathbf{w} - y_t^{(i)}\right]^p, \tag{19}$$

where p is the order set by the *approx_order* option. The number of Newton iterations is specified in the *nonlinear_maxiter* option.

RMTS is implemented in SMT with two choices of splines: B-splines and cubic Hermite splines. 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. For the cubic Hermite splines, 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. B-splines are usually the better choice when training time is the most important factor, whereas cubic Hermite splines are the better choice when the accuracy of the interpolant is most important [18].

3.3 Derivatives

There are three types of derivatives in SMT.

Prediction derivatives ($\mathrm{d}y/\mathrm{d}x$) are derivatives of predicted outputs with respect to the inputs at which the model is evaluated. These are computed together with the

prediction outputs when the surrogate model is evaluated [26]. These are required for gradient-based optimization algorithms based on surrogate models [2, 3].

Training derivatives ($\mathrm{d}y_t/\mathrm{d}x_t$) are derivatives of the training outputs with respect to the corresponding inputs. These are provided by the user and are used to improve the model accuracy in GE-KPLS.

When the adjoint method is used to compute training derivatives, a high-quality surrogate model can be constructed with a low relative cost, because the adjoint method computes these derivatives at a cost independent of the number of inputs.

Output derivatives ($\mathrm{d}y/\mathrm{d}y_t$) are derivatives of predicted outputs with respect to training outputs, which is a measure of how the prediction changes with a change in training outputs, accounting for the retraining of the surrogate model. These post-training derivatives are used when the surrogate model is trained within an optimization iteration. This feature is not commonly available in other frameworks; however, it is required when the training of the surrogate model is embedded in a gradient-based optimization. In this case, derivatives of the prediction outputs with respect to the training outputs must be computed and combined with derivatives from other parts of the model using, for example, the chain rule.

Given its focus on derivatives, SMT is synergistic with the OpenMDAO framework [14], which is a software framework for gradient-based multidisciplinary analysis and optimization [17, 31, 32]. An SMT surrogate model can be a component that is part of a larger model developed in OpenMDAO and can provide the derivatives that OpenMDAO requires from its components to compute the coupled derivatives of the multidisciplinary model.

3.4 Additional surrogate modeling methods

To extend surrogate modeling to higher-level methods, we implemented a component within SMT named *extensions*. These methods require additional and sometimes different steps than the usual surrogate modeling. For example, multi-fidelity methods combine data generated from different sources, such as coarse and fine mesh solutions. Another example is the mixture of experts (MoE) class of methods, which linearly combines several surrogates. Three such methods are available within SMT.

MoE modeling performs a weighted sum of local surrogate models (experts) instead of one single model. It is based on splitting the input space into several subspaces via clustering algorithms and training a surrogate model within each subspace. Hastie et al. [16] provide a general introduction to the MoE method. The implementation of MoE within SMT is based on Gaussian mixture models and expectation maximization [4].

Variable-fidelity modeling (VFM) samples points using both low-fidelity function evaluations that are cheap to evaluate and more costly high-fidelity evaluations. It then uses the differences between the high- and low-fidelity evaluations to

construct a bridge function that corrects the low-fidelity model [15]. SMT implements additive and multiplicative bridge functions.

Multi-fidelity kriging (MFK) uses a correlation factor $\rho(x)$ (constant, linear, or quadratic) and a discrepancy function $\delta(x)$. The high-fidelity model is given by

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

SMT follows the formulation proposed by Le Gratiet [25], which is recursive and can be easily extended to multiple levels of fidelity.

4 Example and benchmark functions

SMT contains a library of analytical and engineering problems that can be used for instructional or benchmarking purposes. The analytical functions currently implemented in SMT are the sphere, Branin, L_p norm, Rosenbrock, and tensor-product functions, all of which are detailed in A. The engineering functions currently implemented in SMT are the cantilever beam, robot arm, torsion vibration, water flow, welded beam, and wing weight problems (described in B).

Figure 4 shows the implementation of the *Sphere* class, which inherits from the *Problem* class. This function is a simple example that serves as a good first test for newly developed surrogate modeling and surrogate-based optimization methods. This function is a continuous and convex function where the global minimum is at the origin.

Figure 5 shows an example of the use of RMTC within SMT for the robot arm function [1]. This function is scalable; however, the number of dimensions must be an even number (we use two dimensions in this example). This function gives the position of a robot arm, which is made by multiple segments, in a two-dimensional space where the shoulder of the arm is fixed at the origin. This function is highly nonlinear, and the use of training derivatives samples is particularly beneficial in this case [5].

5 Applications

In this section, we describe two applications that highlight the unique features of SMT. We also provide an overview of the main previous applications realized using SMT. The first application is the computation and validation of the prediction derivatives for an airfoil analysis tool, where we apply GE-KPLS. The second application describes a practical use for output derivatives, where we use RMTS to compute the outputs derivatives for a surrogate model that is dynamically trained with an optimization iteration.

5.1 Airfoil analysis and shape optimization tool

Li et al. [26] used the GE-KPLS implementation in SMT with an MoE technique to develop a data-driven approach to airfoil and shape optimization based on computational fluid dynamics (CFD) simulations[7]. They used a database of 1100 existing

[7] https://github.com/mdolab/adflow

Figure 4: Implementation of the *Sphere* class, which inherits from the *Problem* class. The main function is `_evaluate`, which computes either the output values or the derivatives depending on the variable `kx`.

```
import numpy as np
from smt.problems.problem import Problem

class Sphere(Problem):
    # Class Sphere inherits class Problem

    # Set up methods of class
    def _initialize(self):
        self.options.declare('name', 'Sphere', types=str)

    def _setup(self):
        # Bornes of the sphere problem
        self.xlimits[:, 0] = -10.
        self.xlimits[:, 1] =  10.

    def _evaluate(self, x, kx):
        """
        Arguments
        ---------
        x : ndarray[ne, nx]
            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[ne, 1]
            Functions values if kx=None or derivative
            values if kx is an int.
        """

        ne, nx = x.shape
        y = np.zeros((ne, 1), complex)
        if kx is None:
            y[:, 0] = np.sum(x**2, 1).T
        else:
            y[:, 0] = 2 * x[:, kx]
        return y
```

airfoils[8] and enriched the database with 100,000 more generated airfoils. The data was used to create a surrogate model of force coefficients (lift, drag, and moment) with respect to flight speed (Mach number), the angle of attack, and airfoil shape variables. The shape design variables consisted of shape modes obtained by singular-value decomposition of the existing airfoil shapes. The approach proved successful, enabling shape optimization in 2 sec using a personal computer with an error of less than 0.25% for subsonic flight conditions. This is thousands of times faster than optimization using direct calls to CFD.

To validate the prediction derivatives provided by SMT, we built a surrogate model of the airfoil drag coefficient similar to the one above using the following four inputs: first thickness mode t_1, first camber mode c_1, Mach number M, and angle of attack α. We compute the partial derivatives of the drag coefficient (C_d) with respect to these

[8] http://webfoil.engin.umich.edu

Figure 5: Example of use of RMTC within SMT on the robot arm function [1].

```
"""
Example solving an SMT benchmark problem using the
Regularized Minimal-energy Tensor-product
Cubic hermite spline surrogate model
"""
from smt.problems import RobotArm
from smt.sampling_methods import LHS
from smt.surrogate_models import RMTC
import numpy as np

# Sample the training points
fun = RobotArm(ndim=2)
sampling = LHS(xlimits=fun.xlimits)
xt = sampling(750)
yt = fun(xt)
for i in range(fun.options['ndim']):
  # Derivative with respect to the i-th variable
  yd = fun(xt,kx=i)
  yt = np.concatenate((yt,yd),axis=1)

# Construct the RMTC model
t = RMTC(xlimits=fun.xlimits, min_energy=True, \
         nonlinear_maxiter=20)
t.set_training_values(xt,yt[:,0])
for i in range(fun.options['ndim']):
  # Add the gradient information to the
  # sampling points
  t.set_training_derivatives(xt,yt[:,1+i],i)
t.train()

# Predict function values
xtest = sampling(5000)
yp = t.predict_values(xtest)

# Predict the derivative values
for i in range(fun.options['ndim']):
  ydp = t.predict_derivatives(xtest,i)

# Predict the derivatives with respect to the
# training points
ytdp = t.predict_output_derivatives(xtest)
```

four variables. The range of the first thickness and camber modes are such that we cover 55% of the airfoil database. The range of M is $[0.3, 0.4]$ and the range of α is $[0, 4]$. To train the surrogate model, we use 20 training points with their respective training derivatives. We use the GE-KPLS model, where we provide training derivatives using an adjoint approach [28, 29]. To validate the prediction derivatives of the constructed surrogate model, we generate 100 new points different from the training points and compute the relative error on those points using

$$\varepsilon = \frac{||\frac{\partial C_d}{\partial x} - \frac{\partial \hat{C}_d}{\partial x}||_2}{||\frac{\partial C_d}{\partial x}||_2}, \tag{20}$$

where $\partial \hat{C}_d/\partial x$ is the vector of 100 surrogate model prediction derivatives with respect to one of the four variables $(x = t_1, c_1, M, \alpha)$, $\partial C_d/\partial x$ is the vector of corresponding reference derivative values with respect to x, and $||.||_2$ is the L_2 norm. The resulting

relative errors corresponding to each variable (listed in Table 2) show that the surrogate model provides an accurate estimate of the derivatives with a relative error less or equal to 10^{-2}.

Table 2: Relative error of the prediction derivative of C_d with respect to t_1, c_1, M, and α. The surrogate model yields an accurate prediction derivative with an error less than or equal to 10^{-2}.

	$\partial C_d/\partial t_1$	$\partial C_d/\partial c_1$	$\partial C_d/\partial M$	$\partial C_d/\partial \alpha$
ε	0.010	0.007	0.009	0.002

Using the constructed surrogate, we compare two gradient-based optimization algorithms:

Optimization 1 uses the surrogate model derivatives provided by SMT;

Optimization 2 uses finite-difference derivatives.

The optimization problem is to minimize the drag coefficient (C_d) subject to a lift constraint ($C_l = 0.5$). Optimization 1 achieved a slightly lower drag (0.03%) than Optimization 2 using 21 times fewer evaluations.

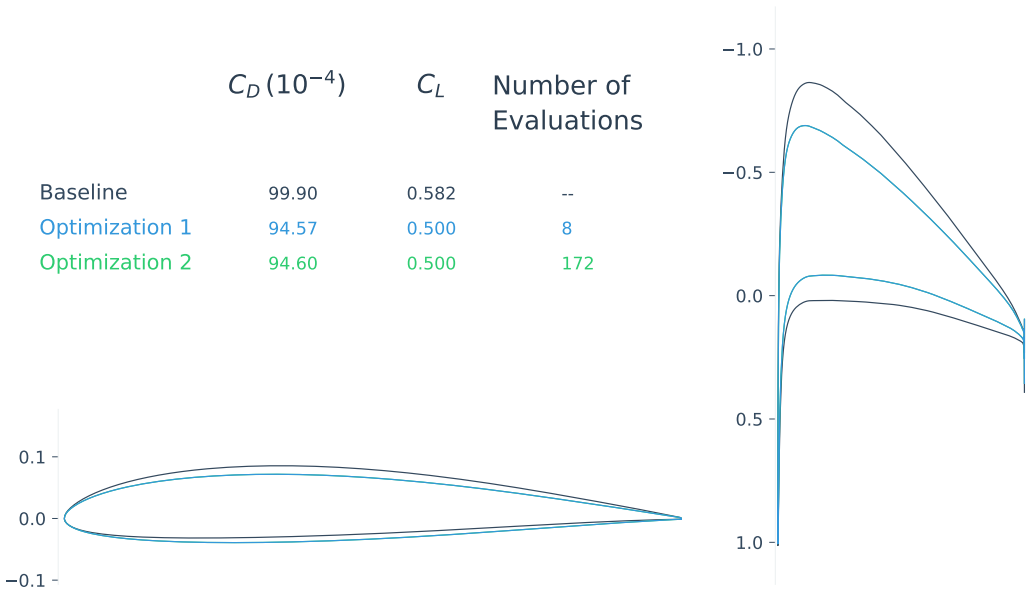


Figure 6: Top left: drag and lift coefficients for the baseline, the solution of Optimization 1, and the solution of Optimization 2. Bottom left: airfoil shapes for all three cases; the shapes for both optimizations is indistinguishable. Right: pressure distributions. Optimization 1 achieved a 0.03% lower drag compared with Optimization 2 using a fraction of the evaluations.

5.2 Aircraft design optimization considering dynamics

The design optimization of commercial aircraft using CFD is typically done by modeling the aircraft performance at a small number of representative flight conditions [22, 23]. This simplification is made because it would be prohibitively expensive to simulate the full, discretized mission profile using CFD at all points. However, full-mission simulation is sometimes necessary, such as when the flight is short or when considering morphing aircraft designs [9, 30]. This requires a surrogate model of the aerodynamic performance as a function of a small number of parameters such as flight speed and altitude. In a design optimization context, the surrogate model must be retrained each optimization iteration because the aircraft design changes from iteration to iteration as the shape is being optimized.

This is the situation in an aircraft allocation-mission-design optimization problem recently solved using RMTS [19]. In this work, the optimization problem maximized airline profit by optimizing the twist, span, sweep, and shape of the wing of a next-generation commercial aircraft. The profit was computed by simulating the fuel efficiency and flight time for the aircraft on a set of routes operated by the hypothetical airline. To do this, a surrogate model was generated for the wing lift and drag coefficients (C_L and C_D) as a function of the angle of attack (α), Mach number (M), and altitude (h). In each optimization iteration, the training C_L and C_D values were computed at a series of points in M–α–h space using CFD. The training outputs were used to retrain the RMTS surrogate model, and the mission simulations for all the airline routes were performed using inexpensive evaluations of the trained surrogate model, as shown in Figure 7. However, because the overall optimization problem was solved using a gradient-based algorithm, we required derivatives of the prediction outputs (C_L and C_D values at the M–α–h points at which we evaluated the surrogate) with respect to the training outputs (C_L and C_D values at the fixed M–α–h points where the training points were located).

Figure 8 shows the surrogate model with the altitude axis eliminated by projecting onto the other two axes, where C_L replaces α as one of the inputs. The horizontal and vertical axes represent the inputs. The red points are the training points and the black points are the prediction points. Therefore, the output derivatives are the derivatives of the outputs at the black points with respect to the training outputs at the red points.

5.3 Other applications

In addition to the two applications we just described, SMT has been used to solve other engineering problems. We summarize these applications in Table 3. The number of input variables ranges from 2 to 99. This demonstrates SMT's ability to solve different engineering problems of various complexities.

6 Conclusions

SMT is unique compared with existing surrogate modeling libraries in that it is designed from the ground up to handle derivative information effectively. The derivative

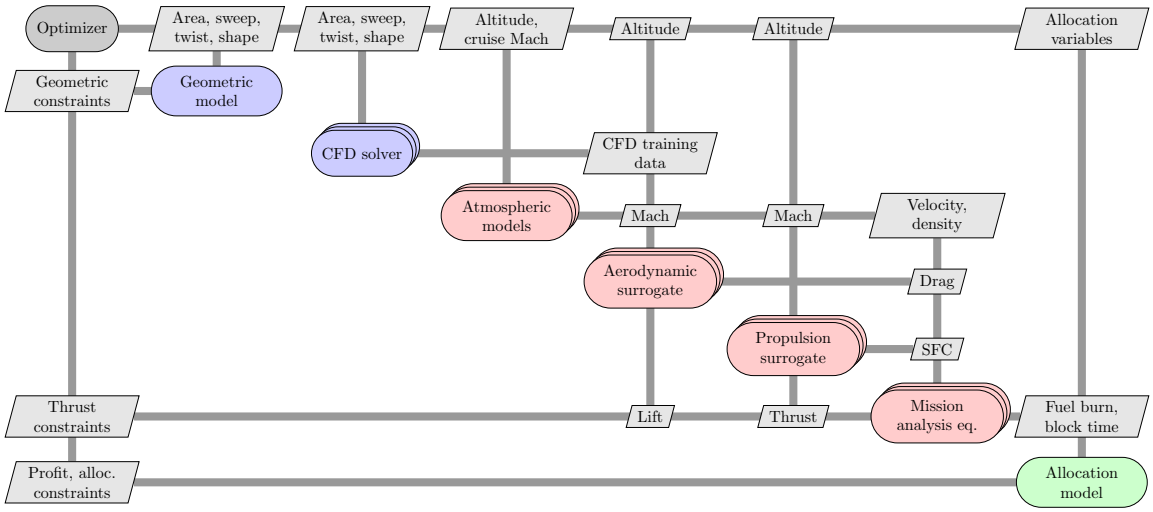


Figure 7: Extended design structure matrix [24] for the aircraft allocation-mission-design optimization problem that used RMTS [19]. The aerodynamic surrogate model predicts wing lift and drag coefficient as a function of the angle of attack, Mach number, and altitude. This surrogate model is retrained each iteration because the lift and drag training data change each optimization iteration as the aircraft wing design changes. Since a gradient-based algorithm is used for this large-scale optimization problem, we require output derivatives: derivatives of the lift and drag prediction outputs with respect to the lift and drag training data. These are provided by RMTS.

information includes training derivatives used for gradient-enhanced modeling, prediction derivatives used for surrogate-model-based optimization, and derivatives with respect to the training data used when the optimization loop includes reconstructing the surrogate model. However, SMT does not need to involve derivatives and provides a simple general interface to various surrogate modeling techniques. Together with the provided problems, SMT is a convenient framework to develop new methods for sampling and surrogate modeling, and then benchmarking them against existing methods.

SMT arose from our own research needs, namely the development of the GE-KPLS and RMTS surrogate models, which require the handling of derivatives. GE-KPLS is a method we recently developed that scales much better with the number of inputs than the other methods, thanks in part to the use of the derivative information. RMTS is another method that we recently developed that is best suited to low-dimensional problems and handles both structured and unstructured training points.

SMT is distributed through an open-source license and is freely available online[9]. We provide documentation that caters to both users and potential developers[10]. SMT allows users collaborating on the same project to have a common surrogate modeling tool that facilitates the exchange of methods and reproducibility of results.

[9]https://github.com/SMTorg/SMT
[10]http://smt.readthedocs.io/en/latest/

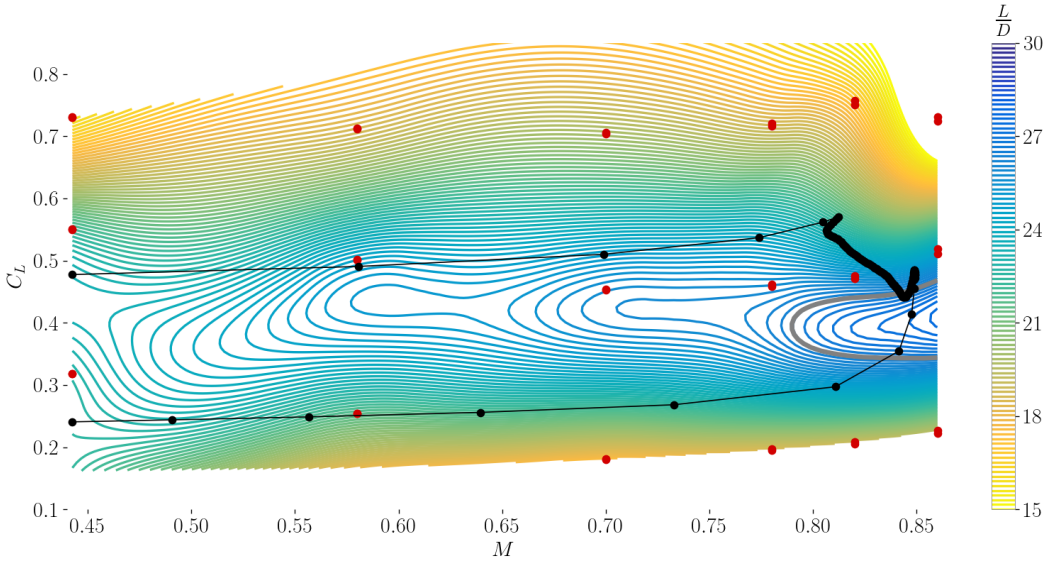


Figure 8: RMTS surrogate model for the performance of an aircraft, represented by the lift-to-drag ratio (L/D), as a function of Mach number (M), and lift coefficient (C_L) [19]. In this context, an output derivative is the derivative of the L/D values at the prediction points (shown in black) with respect to the L/D values at the training points (shown in red). The C_L–M values of the red points are the training inputs, which are fixed, and the C_L–M values of the black points are the prediction inputs. The path of the black points represents the flight of a commercial aircraft, from climb with $C_L \sim 0.5$ to cruise at $M > 0.8$ to descent with $C_L \sim 0.3$. The red points appear to be duplicated because there is a third input, altitude, which is eliminated by projecting onto the C_L–M plane.

SMT is a recently developed tool and, so far, it has primarily been used in aerospace engineering applications. However, this tool is useful to anyone needing to use or develop surrogate modeling techniques, regardless of the application. In particular, given the advantages of the newer surrogate modeling techniques (GE-KPLS and RMTS), using SMT is particularly attractive.

Acknowledgments

This work was partially supported by the Air Force Office of Scientific Research (AFOSR) MURI on "Managing multiple information sources of multi-physics systems," Program Officer Jean–Luc Cambier, Award Number FA9550-15-1-0038. This work is part of the activities of ONERA–ISAE–ENAC joint research group and was partially supported by an ONERA internal project on multi-fidelity methods (MUFIN). The authors would like to thank Rémi Vauclin and Mostafa Meliani for the implementation of MFK, and Rémy Priem for the implementation of MoE.

Table 3: Summary of SMT applications to engineering design problems.

Problem	Surrogate used	Objective of the study	Number of design variables	Reference
3D blade	KPLS	Compare the accuracy and efficiency of the KPLS model and the Optimus [34] implementation of the kriging model	Up to 99	Bouhlel et al. [7]
3D blade	KPLSK	Compare the accuracy and efficiency of the KPLSK model and the KPLS model	99	Bouhlel et al. [6]
Automotive	KPLS, KPLSK	Minimize the mass of a vehicle subject to 68 constraints	50	Bouhlel et al. [8]
Eight different engineering problems	GE-KPLS	Compare the accuracy and efficiency of the GE-KPLS model and the indirect GEK	Up to 15	Bouhlel and Martins [5]
Aircraft wing	MoE	Minimize the drag of aircraft wings subject to lift constraints	Up to 17	Bartoli et al. [2]
2D airfoils	GE-KPLS	Build a fast interactive airfoil analysis and design optimization tool	Up to 16	Li et al. [26]
Aircraft performance	RMTS	Optimize an aircraft design while analyzing the full mission using a CFD surrogate	3	Hwang and Munster [19]
Rotor analysis	RMTS	Optimize a rotor using a surrogate model for the lift and drag of blade airfoils	2	Hwang and Ning [20]

A Analytical functions implemented in SMT

In the following descriptions, n_x is the number of dimensions.

Sphere The sphere function is quadratic, continuous, convex, and unimodal. It is given by

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

Branin [12] The Branin function is commonly used in optimization and has three global minima. It is given by

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

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

Rosenbrock [40] The Rosenbrock function is a continuous, nonlinear, and non-convex function and usually used in optimization. The minimum of this function is situated in a parabolic-shaped flat valley and is given by

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

Tensor-product The tensor-product function approximates a step function, which causes oscillations with some surrogate models [18], and is given by

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

or

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

or

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

or

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

B Engineering problems implemented in SMT

Cantilever beam [10]

This function models a simple uniform cantilever beam with vertical and horizontal loads

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

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

Robot arm [1]

This function gives the position of a robot arm

$$\sqrt{\left(\sum_{i=1}^{4} L_i \cos\left(\sum_{j=1}^{i} \theta_j \right) \right)^2 + \left(\sum_{i=1}^{4} L_i \sin\left(\sum_{j=1}^{i} \theta_j \right) \right)^2},$$

where $L_i \in [0, 1]$ for $i = 1, \ldots, 4$ and $\theta_j \in [0, 2\pi]$ for $j = 1, \ldots, 4$.

Torsion vibration [27]

This function gives the low natural frequency of a torsion problem

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

where $K_i = \frac{\pi G_i d_i}{32 L_i}$, $M_j = \frac{\rho_j \pi t_j D_j}{4g}$, $J_j = 0.5 M_j \frac{D_j}{2}$, $a = 1$, $b = -\left(\frac{K_1+K2}{J_1} + \frac{K_2+K3}{J_2}\right)$, $c = \frac{K_1K_2+K_2K_3+K_3K_1}{J_1J_2}$, for $d_1 \in [1.8, 2.2]$, $L_1 \in [9, 11]$, $G_1 \in [105300000, 128700000]$,

$d_2 \in [1.638, 2.002]$, $L_2 \in [10.8, 13.2]$, $G_2 \in [5580000, 6820000]$, $d_3 \in [2.025, 2.475]$, $L_3 \in [7.2, 8.8]$, $G_3 \in [3510000, 4290000]$, $D_1 \in [10.8, 13.2]$, $t_1 \in [2.7, 3.3]$, $\rho_1 \in [0.252, 0.308]$, $D_2 \in [12.6, 15.4]$, $t_2 \in [3.6, 4.4]$, and $\rho_1 \in [0.09, 0.11]$.

Water flow [33]

This function characterizes the flow of water through a borehole that is drilled from the ground surface through two aquifers

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

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

Welded beam [11]

The shear stress of a welded beam problem is given by

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

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 $h \in [0.125, 1]$, and $l, t \in [5, 10]$.

Wing weight [12]

The estimate of the weight of a light aircraft wing is given by

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

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

References

[1] J. An and A. Owen. Quasi-regression. *Journal of Complexity*, 17(4):588–607, 2001. ISSN 0885-064X. doi:10.1006/jcom.2001.0588.

[2] N. Bartoli, T. Lefebvre, S. Dubreuil, R. Olivanti, N. Bons, J. R. R. A. Martins, M. A. Bouhlel, and J. Morlier. An adaptive optimization strategy based on mixture of experts for wing aerodynamic design optimization. In *Proceedings of the 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Denver, CO, June 2017. doi:10.2514/6.2017-4433.

[3] N. Bartoli, T. Lefebvre, S. Dubreuil, R. Olivanti, R. Priem, N. Bons, J. R. R. A. Martins, and J. Morlier. Adaptive modeling strategy for constrained global optimization with application to aerodynamic wing design. *Aerospace Science and Technology*, 2019. (In press).

[4] D. Bettebghor, N. Bartoli, S. Grihon, J. Morlier, and M. 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.

[5] M. A. Bouhlel and J. R. R. A. Martins. Gradient-enhanced kriging for high-dimensional problems. *Engineering with Computers*, 1(35):157–173, January 2019. doi:10.1007/s00366-018-0590-x.

[6] M. A. Bouhlel, N. Bartoli, J. Morlier, and A. Otsmane. 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. doi:10.1155/2016/6723410. Article ID 6723410.

[7] M. A. Bouhlel, N. Bartoli, A. Otsmane, and J. Morlier. Improving kriging surrogates of high-dimensional design models by partial least squares dimension reduction. *Structural and Multidisciplinary Optimization*, 53(5):935–952, 2016. doi:10.1007/s00158-015-1395-9.

[8] M. A. Bouhlel, N. Bartoli, R. G. Regis, A. Otsmane, and J. Morlier. Efficient global optimization for high-dimensional constrained problems by using the kriging models combined with the partial least squares method. *Engineering Optimization*, 50(12):2038–2053, 2018. doi:10.1080/0305215X.2017.1419344.

[9] D. A. Burdette and J. R. R. A. Martins. Impact of morphing trailing edge on mission performance for the common research model. *Journal of Aircraft*, 56(1): 369–384, January 2019. doi:10.2514/1.C034967.

[10] G. H. Cheng, A. Younis, K. H. Hajikolaei, and G. G. Wang. Trust region based mode pursuing sampling method for global optimization of high dimensional design problems. *Journal of Mechanical Design*, 137(2):021407, 2015.

[11] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):311–338, 1998. doi:10.1016/S0045-7825(99)00389-8.

[12] A. I. J. Forrester, A. Sobester, and A. J. Keane. *Engineering Design via Surrogate Modelling—A Practical Guide.* Wiley, 2008. ISBN 978-0-470-06068-1.

[13] D. Gorissen, K. Crombecq, I. Couckuyt, T. Dhaene, and P. Demeester. A surrogate modeling and adaptive sampling toolbox for computer based design. *Journal of Machine Learning Research*, 11:2051–2055, July 2010.

[14] J. S. Gray, J. T. Hwang, J. R. R. A. Martins, K. T. Moore, and B. A. Naylor. OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization. *Structural and Multidisciplinary Optimization*, 2019. doi:10.1007/s00158-019-02211-z. (In press).

[15] Z.-H. Han, S. Görtz, and R. Zimmermann. Improving variable-fidelity surrogate modeling via gradient-enhanced kriging and a generalized hybrid bridge function. *Aerospace Science and Technology*, 25(1):177–189, March 2013. doi:10.1016/j.ast.2012.01.006.

[16] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[17] J. T. Hwang and J. R. R. A. Martins. A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives. *ACM Transactions on Mathematical Software*, 44(4):Article 37, June 2018. doi:10.1145/3182393.

[18] J. T. Hwang and J. R. R. A. Martins. A fast-prediction surrogate model for large datasets. *Aerospace Science and Technology*, 75:74–87, April 2018. doi:10.1016/j.ast.2017.12.030.

[19] J. T. Hwang and D. W. Munster. Solution of ordinary differential equations in gradient-based multidisciplinary design optimization. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Kissimmee, FL, January 2018. AIAA, AIAA. doi:10.2514/6.2018-1646.

[20] J. T. Hwang and A. Ning. Large-scale multidisciplinary optimization of an electric aircraft for on-demand mobility. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Kissimmee, FL, January 2018. doi:10.2514/6.2018-1384.

[21] R. Jin, W. Chen, and A. Sudjianto. An efficient algorithm for constructing optimal design of computer experiments. *Journal of Statistical Planning and Inference*, 134 (1):268–287, September 2005. ISSN 0378-3758. doi:10.1016/j.jspi.2004.02.014.

[22] G. K. W. Kenway and J. R. R. A. Martins. Multipoint high-fidelity aerostructural optimization of a transport aircraft configuration. *Journal of Aircraft*, 51(1):144–160, January 2014. doi:10.2514/1.C032150.

[23] G. K. W. Kenway and J. R. R. A. Martins. Multipoint aerodynamic shape optimization investigations of the Common Research Model wing. *AIAA Journal*, 54 (1):113–128, January 2016. doi:10.2514/1.J054154.

[24] A. B. Lambe and J. R. R. A. Martins. Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes. *Structural and Multidisciplinary Optimization*, 46:273–284, August 2012. doi:10.1007/s00158-012-0763-y.

[25] L. Le Gratiet. *Multi-fidelity Gaussian process regression for computer experiments*. Theses, Université Paris–Diderot - Paris VII, Oct. 2013. URL https://tel.archives-ouvertes.fr/tel-00866770.

[26] J. Li, M. A. Bouhlel, and J. R. R. A. Martins. Data-based approach for fast airfoil analysis and optimization. *Journal of Aircraft*, 57(2):581–596, February 2019. doi:10.2514/1.J057129.

[27] W. Liping, B. Don, W. Gene, and R. Mahidhar. A comparison of metamodeling methods using practical industry requirements. *Proceedings of the 47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Newport, RI*, 2006.

[28] Z. Lyu, G. K. Kenway, C. Paige, and J. R. R. A. Martins. Automatic differentiation adjoint of the Reynolds-averaged Navier–Stokes equations with a turbulence model. In *21st AIAA Computational Fluid Dynamics Conference*, San Diego, CA, July 2013. doi:10.2514/6.2013-2581.

[29] C. A. Mader, J. R. R. A. Martins, J. J. Alonso, and E. van der Weide. ADjoint: An approach for the rapid development of discrete adjoint solvers. *AIAA Journal*, 46(4):863–873, Apr. 2008. doi:10.2514/1.29123.

[30] J. R. R. A. Martins. *Encyclopedia of Aerospace Engineering*, volume Green Aviation, chapter Fuel burn reduction through wing morphing, pages 75–79. Wiley, October 2016. ISBN 978-1-118-86635-1. doi:10.1002/9780470686652.eae1007.

[31] J. R. R. A. Martins and J. T. Hwang. Review and unification of methods for computing derivatives of multidisciplinary computational models. *AIAA Journal*, 51(11):2582–2599, November 2013. doi:10.2514/1.J052184.

[32] J. R. R. A. Martins and A. B. Lambe. Multidisciplinary design optimization: A survey of architectures. *AIAA Journal*, 51(9):2049–2075, September 2013. doi:10.2514/1.J051895.

[33] M. D. Morris, T. J. Mitchell, and D. Ylvisaker. Bayesian design and analysis of computer experiments: Use of derivatives in surface prediction. *Technometrics*, 35(3):243–255, 1993.

[34] Noesis Solutions. *OPTIMUS*. 2009. URL http://www.noesissolutions.com/Noesis/optimus-details/optimus-design-optimization.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Rettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] M. J. D. Powell. *The Theory of Radial Basis Function Approximation in 1990*, pages 105–210. Oxford University Press, May 1992.

[37] M. J. D. Powell. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, pages 51–67. Springer Netherlands, Dordrecht, 1994. ISBN 978-94-015-8330-5. doi:10.1007/978-94-015-8330-5_4.

[38] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, Jan. 2006.

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

[40] Y. Shang and Y. Qiu. A note on the extended Rosenbrock function. *Evolutionary Compuation*, 14(1):119–126, Mar. 2006. ISSN 1063-6560. doi:10.1162/106365606776022733.

[41] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, ACM '68, pages 517–524, New York, NY, USA, 1968. ACM. doi:10.1145/800186.810616.