DATADVANCE

pSeven Core

GTDF
Generic Tool for Data Fusion

Contact information

Phone	+7 (495) 669 68 15	
Web	`https://www.pseven.io/`	
Email	`support@datadvance.net`	Technical support, questions, bug reports
	`info@datadvance.net`	Everything else
Mail	DATADVANCE, llc Nauchny proezd, 17, 15th floor, office XXXI 117246 Moscow Russia	

User manual prepared by Zaytsev A. (principal author) and Burnaev E.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 What is GT DF

GT DF is a software package for

- construction of approximations fitting user-provided training data including both high and low fidelity data,
- assessment of quality of the constructed approximations.

1.2 Documentation structure

The manual is organized as follows.

- Chapter 2 describes a problem statement and **GTDF** tool's functionality.
- Chapter 3 describes a general workflow of **GTDF**.
- Chapter 4 describes available DF techniques.
- Chapter 5 describes how the **GTDF** selects a DF techniques.
- Chapter 6 provides some additional remarks about **GTDF** and describes available options.
- Chapter 7 gives a number of artificial and real examples of **GTDF** usage.
- Chapter A describes mathematical foundation of provided **GTDF** techniques.
- Chapters B, B.3 and C briefly describes **GTApprox** and provides information about **GTApprox** necessary for **GTDF** usage.

Chapter 2

Overview

A common situation in surrogate modelling is availability of a number of ways to model physical process $Y = f(X)$ [13, 14]. For example, one can use two grids with different fidelities or two different systems of ODE to model the same physical process. Both models approximate a real value of the function $f(X)$ in a point X, but have different fidelities. *A high fidelity model* (fine model) $f_h(X)$ is more accurate, but needs more time to produce result, and *a low fidelity model* (coarse model) $f_l(X)$ is less accurate, but can be computed much faster in comparison with the first model. So, a set of the known values of the first model $f_h(X)$ is much smaller than a set for the second model values $f_l(X)$ due to high computational costs of the high fidelity model evaluation. Using user-provided *learning sets* of the high fidelity function and the low fidelity function values at given points we build *an approximation* of the high fidelity function $f_h(X)$, namely we try to predict values of high fidelity function using both values from high fidelity set and low fidelity set. *A data fusion technique* is a method to take into account both values from the high fidelity and the low fidelity model when constructing an approximation.

2.1 Sample based problem statement

The problem to be solved is to build an approximation $\hat{f}_h(X)$ of $f_h(X)$ for $X \in \mathbb{X}$ using a training set. The training set consists of two parts. The first part $D_h = (F_h, \mathbf{X}_h) = \{(f_h(X_i^h), X_i^h)\}_{i=1}^{N_h}$ includes a set of pairs of points and high fidelity function values at this points. The matrix F_h corresponds to the output data of high fidelity model, and the matrix $\mathbf{X}_h$ corresponds to the input data of high fidelity model. The second part $D_l = (F_l, \mathbf{X}_l) = \{(f_l(X_i^l), X_i^l)\}_{i=1}^{N_l}$ includes a set of pairs of points and low fidelity function values at this points. The matrix F^l corresponds to the output data of low fidelity model, and the matrix $\mathbf{X}_l$ corresponds to the input data of low fidelity model. Input vectors X_i^h, X_i^l belong to $\mathbb{R}^{d_{\text{in}}}$, output vectors $f_h(X_i^h)$, $f_l(X_i^l)$ belong to $\mathbb{R}^{d_{\text{out}}}$. It is supposed that $N_h \ll N_l$.

The input and output for the **GTDF** is

- Input
  - Low fidelity data D_l and high fidelity data D_h;
  - Options.
- Output
  - Approximation of high fidelity function $\hat{f}_h(X)$;

- Partial derivatives of approximation $\frac{\partial \hat{f}_h(X)}{\partial x_i}$ with respect to input dimension $x_i, i = 1, 2, \ldots, d_{\text{in}}$ ($X = \{x_1, x_2, \ldots, x_{d_{\text{in}}}\}$);
- Accuracy evaluation $AE(X)$[1] (if available) for approximation, see also Section 6.5.

If d_{out} is bigger than 1 we suppose that the number of output dimensions d_{out} for low and high fidelity functions coincides and j-th output dimension of low fidelity function is a coarse model for j-th output dimension of high fidelity function.

2.2 Black box based problem statement

Another problem statement arises when a black box for low fidelity model is available.

The problem to be solved is to build an approximation $\hat{f}_h(X)$ of $f_h(X)$ using a training set of high fidelity model values and black box for low fidelity model. A set of points $D_h = (F_h, \mathbf{X}_h) = \{(f_h(X_i^h), X_i^h)\}_{i=1}^{N_h}$ includes a set of pairs of points and high fidelity function values at this points. Black box for low fidelity model provides a result of calculation of low fidelity model $f_l(X)$ at any feasible point of the design space $\mathbb{X}$.

In this case **GTDF** generates design of experiment for low fidelity model $D_l = (F_l, \mathbf{X}_l) = \{(f_l(X_i^l), X_i^l)\}_{i=1}^{N_l}$ and builds approximation using D_h and generated D_l using some **GTDF** technique. To build an approximation we use some Blackbox-based Data Fusion technique (**BB DF**). Prediction of the high fidelity function for the given input is done as follows.

- Blackbox is used to calculate the value of low fidelity function
- Approximation of high fidelity function is updated using calculated value in order to obtain more accurate prediction.

In general, such adaptive approach update scheme significantly improves accuracy of prediction, see for example toy problem from the next section.

2.3 Toy problem

We present a toy problem inspired by the problem described in [14]. Suppose our high fidelity model is a function

$$f_h(x) = (6x - 2)^2 \sin(12x - 4)$$

and our low fidelity model is a function

$$f_l(x) = Af_h(x) + B(x - 0.5) - C,$$

we use for our example $A = 0.5$, $B = 10$, $C = 5$. Suppose we know low fidelity function values at points $\mathbf{X}_l = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ and high fidelity function values only at points $\mathbf{X}_h = \{0, 0.05, 0.2, 0.55, 0.95, 1\}$. We learn an approximation of the high fidelity function using values of the low fidelity function at points $\mathbf{X}_l$ and the high fidelity function at points $\mathbf{X}_h$.

Results for different techniques are given in figure 2.1. We use three methods to build an approximation. For construction of an approximator we use **GTApprox** [8] and only values

[1] Accuracy Evaluation's purpose is to estimate the accuracy of the constructed approximation at different points of the design space

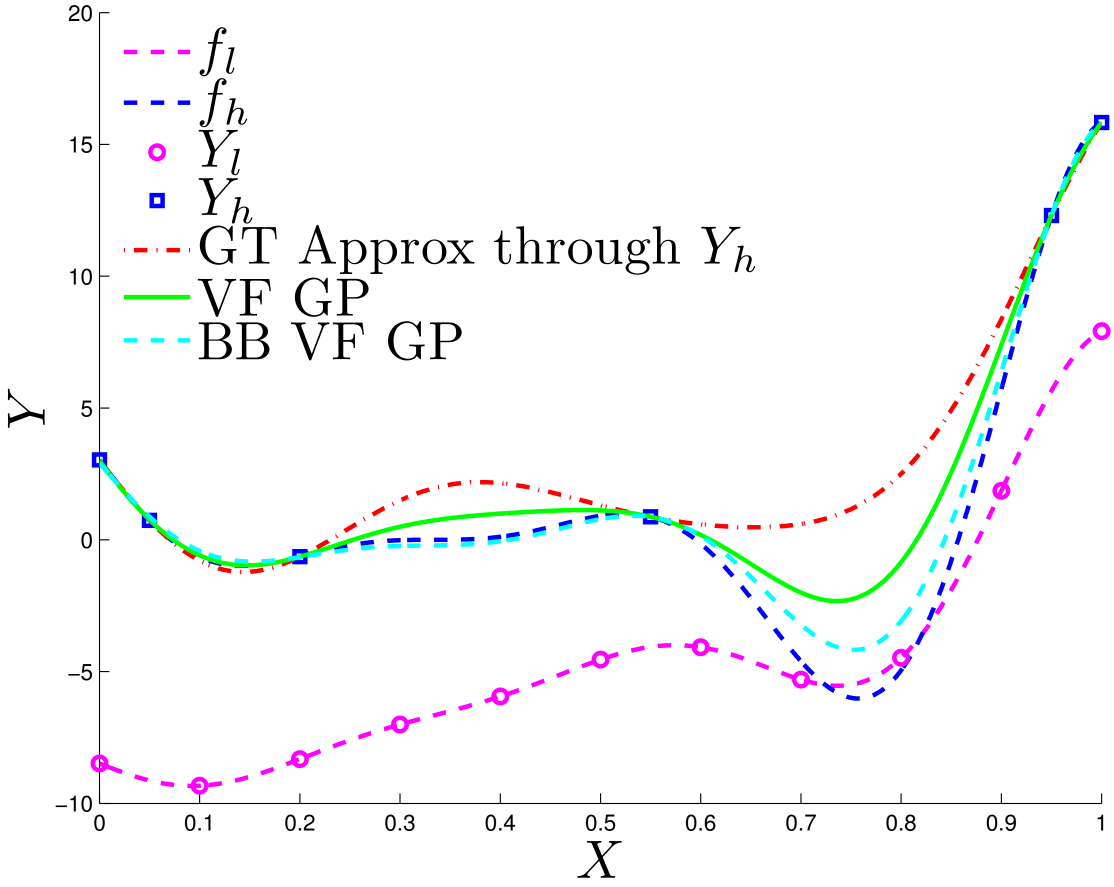


Figure 2.1: Example for model function. Blue dotted line corresponds to the high fidelity function $f_h(X)$, magenta dotted line corresponds to the low fidelity function $f_l(X)$. Points of learning sample for the high fidelity function are depicted as blue squares, points of learning sample for the low fidelity function are depicted as magenta circles. Derived approximation using **GTApprox** and high fidelity data is depicted as the red dash-dotted line, derived approximation using **VFGP** and both high and low fidelity data is depicted as the green solid line, **BB VFGP** is depicted as the cyan dash-dotted line.
Note: This image was obtained using an older pSeven Core version. Actual results in the current version may differ.

of the high fidelity function. We learn variable fidelity gaussian processes approximator **VFGP** [1, 14, 26] using both values of the high and low fidelity functions at given points. Also we learn **VFGP** with available black box for low fidelity function.

One can see that we obtain more accurate resulting approximation using more data of different fidelity to learn an approximator. Additional information from the less accurate model helps us to resolve difficulties with approximation of high fidelity function. Using black box for low fidelity function we obtain results superior to results obtained using other methods.

This problem is provided as a python script in the documentation in the file `toyExample.py`.

A number of real and artificial data examples is presented in Chapter 7.

Chapter 3

Workflow

3.1 Overview

The workflow with the **GTDF** includes the following elements.

- **Construction of the model.** Construction of the model internally consists of the following steps:

  1. **Preprocessing.** At this step, redundant data are removed from the training set. See Section 3.2.
  2. **Analysis of training data and options, selection of approximation technique.** At this step, the parameters of the training data and the user–specified options are analyzed for compatibility, and the most appropriate approximation technique is selected. See Chapter 5.
  3. **Construction of the main approximation with (optionally) Accuracy Evaluation prediction.** At this step, the main approximation, to be returned to the user, is constructed. See Chapter 4 for description of individual approximation techniques. Also, the accompanying Accuracy Evaluation prediction is constructed if the corresponding option is turned on.

  After the model has been constructed, the user can view training settings and export the approximation to a text file.

- **Evaluation of the model.** One can evaluate the approximation, gradient and accuracy prediction (if available) using the model constructed.

3.2 Preprocessing

Before applying the approximation technique to the training set, this training set is preprocessed in order to remove possible degeneracies in the data. We do this in two steps. At first step we remove repeating pairs of $(f(X), X)$ from the samples $D_l = (\mathbf{Y}_l, \mathbf{X}_l) = \{(f_l(X_i^l), X_i^l)\}_{i=1}^{N_l}$, $D_h = (\mathbf{Y}_h, \mathbf{X}_h) = \{(f_h(X_i^h), X_i^h)\}_{i=1}^{N_h}$ of correspondingly low and high fidelity values. We obtain reduced samples $D_l' = (\mathbf{Y}_l', \mathbf{X}_l')$, $D_h' = (\mathbf{Y}_h', \mathbf{X}_h')$. Then we search for constant columns in input data and removes them if in both high and low fidelity data these columns are constant. After removing constant columns in input data we search for constant columns in output data. The whole set of columns is $I = \{1, 2, \ldots, d_{\text{out}}\}$. We split columns in $\mathbf{Y}'_l$ to constant $I_l^{const} \subseteq I$ and nonconstant $I_l^{nonconst}$. Also, we split columns in $\mathbf{Y}'_h$ to constant $I_h^{const} \subseteq I$ and nonconstant $I_h^{nonconst}$. There are three possible situations for column in I:

- For columns in I_h^{const} we create constant approximation. It means that these corresponding outputs are predicted by some constants and AE for this prediction unavailable.
- For columns in $I_l^{const} \cap I_h^{nonconst}$ we use `HFA` because we can't get any information from constant rows in low fidelity data.
- For columns in $I_l^{nonconst} \cap I_h^{nonconst}$ we use corresponding **GTDF** technique because in this case it makes sense to use both high and low fidelity data.

Note, that if d_{out} is bigger than 1 we suppose that a number of output dimensions for low and high fidelity functions coincides and j-th output dimension of low fidelity data is a coarse model for j-th output dimension of high fidelity data.

3.3 Black box for data fusion

Other workflow is used if black box for low fidelity model is available. In this case *construction* of a model consists of two steps:

- First, we generate Design of experiment for low fidelity model and calculate low fidelity function values at selected points. For the set of points generated we exclude points with unavailable values of low fidelity function.
- Second, we use proper **GT DF** algorithm to build an approximation using both low and high fidelity sets of points and corresponding function values.

Note that we use black box for evaluation of low fidelity model when calculating predictions.

Another important issue to take into account is derivatives calculation. If blackbox function values are used for calculation of prediction for new input points and derivatives for low fidelity model aren't available we use numerical estimation of derivatives for black box low fidelity model. So, for noisy and high frequency functions approximation numerical derivatives can be inaccurate. In this case it is recommended to use some smoothing technique for blackbox function to make it smoother.

3.3.1 Design of experiments

Design of experiments for the **BB for DF** component includes set of points $\mathbf{X}_h$, some additional input points are generated using **GT DOE**. Number of points N_l for low fidelity model depends on size N_h of high fidelity model set. Dependency is depicted in figure 3.1.

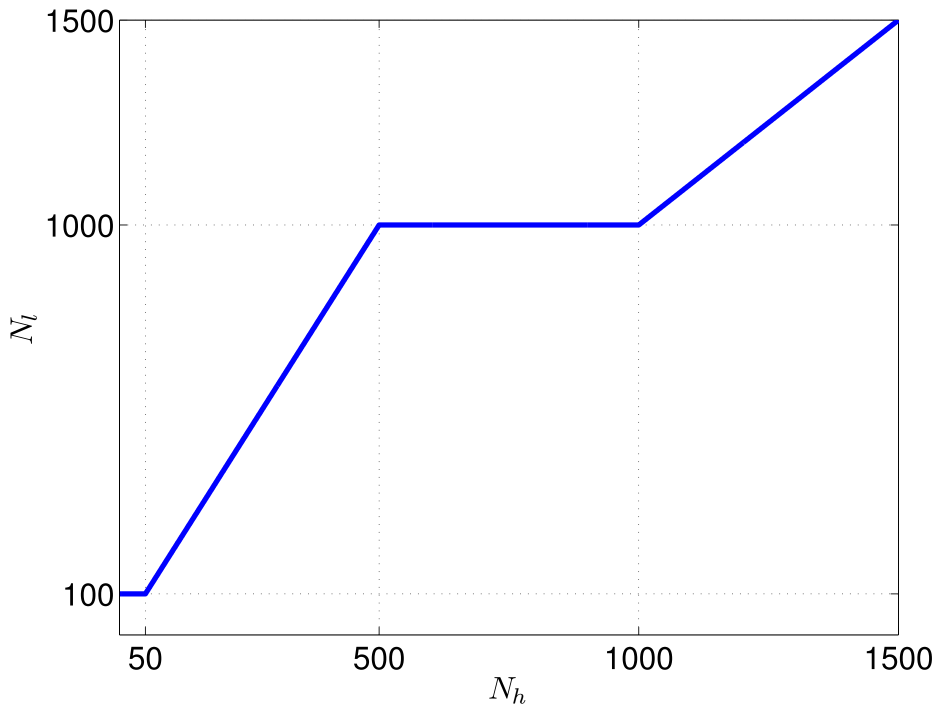


Figure 3.1: Size of low fidelity set of points X_l for Blackbox-based Data Fusion

Chapter 4

Approximation techniques

4.1 General description of presented approximation techniques

For **GT DF** three approximation techniques are presented:

- `HighFidelityApprox` (`HFA`) — approximation using only high fidelity data,
- `DiffApprox` (`DA`) — approximation is a sum of low fidelity data approximation and difference between low and high fidelity data approximation,
- `VariableFidelityGaussianProcess` (`VFGP`) — approximation using multi-task gaussian processes [1, 14] based approximation.
- `Sparse VariableFidelityGaussianProcess` (`SVFGP`) — approximation using multi-task gaussian processes [1, 14] based approximation for large data sets based on sparse gaussian processes [6, 15].

All presented techniques use **GT Approx** tool as a base approximator.

4.1.1 HighFidelityApprox

Short name: HFA

General description: In this case we ignore low fidelity data and use only high fidelity data to build an approximator. So, we use pure **GTApprox** [8] to construct an approximation. The brief overview of **GTApprox** is given in Appendix B.

Strengths and weaknesses: This technique doesn't take into account low fidelity data and can be used if low fidelity data is weakly correlated with high fidelity data or amount of available low fidelity data is small.

Restrictions: Can be used with all available modes from **GTApprox**.

Options: This technique has the same properties and use the same options from **GTApprox** [8], see also Appendix B.3.

Detailed description: For this technique we use approximation techniques carefully described in [8]. Available techniques and its' options are described in Appendix B.

4.1.2 DiffApprox

Short name: DA

General description: The final approximation is a sum of two approximations given by **GTApprox** approximators. First one reproduces low fidelity data, second one approximates difference between high and low fidelity data.

Strengths and weaknesses: This technique makes a priory assumptions about link between low and high fidelity data. It is recommended to use this technique in case data meets this assumptions, i.e. we can model high fidelity data as a sum of low fidelity data approximation and difference between low and high fidelity data approximation. Strength of this technique is it's simplicity and reliability.

Restrictions: Can be used with all available modes from **GTApprox**. If proper **GTApprox** technique is used `AE` and interpolation mode are available.

Options: This technique has the same properties and use the same options from **GTApprox**, see Appendix B or [8]). Both **GTApprox** approximators, used for modeling of high and low fidelity data, have the same options once specified by user for the `DA` technique.

Detailed description: For `DA` high fidelity function $f_h(X)$ is supposed to be a sum of low fidelity function $f_l(X)$ and difference function $f_d(X)$. So, the model to use is

$$f_h(X) = f_l(X) + f_d(X).$$

To build approximation we use the following method:

- Build an approximation $\hat{f}_l(X)$ for the low fidelity model $f_l(X)$ using data D_l.
- Get values of the low fidelity model approximation $\hat{f}_l(X)$ in points X_i^h from the sample D_h.
- Calculate difference between the high fidelity model values $f_h(X)$ and the low fidelity model approximation $\hat{f}_l(X)$ in points X_i^h from the sample D_h:

$$f_d(X_i^h) = f_h(X_i^h) - \hat{f}_l(X_i^h), i = 1, 2, \dots, N_h.$$

- Build an approximation of difference between high and low fidelity models $\hat{f}_d(X)$.

The final approximation $\hat{f}_h(X)$ is a sum of two approximations:

$$\hat{f}_h(X) = \hat{f}_l(X) + \hat{f}_d(X).$$

To select the **GTApprox** techniques to apply we use decision tree from Chapter 5.

4.1.3 VariableFidelityGaussianProcess

Short name: VFGP

General description: `VFGP` is a gaussian-processes-based technique. It has a number of advantages in comparison with other techniques [13]. `VFGP` uses both low and high fidelity data to make prediction, i.e. it uses covariances between points from low fidelity data points and high fidelity data points. `VFGP` inherits reliability and theoretical properties of gaussian-processes-based regression from **GTApprox**. It means that AE and interpolation mode are available for `VFGP`.

Strengths and weaknesses: The `VFGP` technique is the most powerful technique presented in **GTDF**. So, it is sufficiently general. It means that this technique handles more complex links between high and low fidelity data.

Restrictions: If sample size N_l or N_h is large, this technique takes a log time, so it is recommended to use other **GTDF** technique in this case.

Options: This technique has the same properties and use the same options from **GTApprox**. See Appendix B or [8]. `VFGP` uses two gaussian processes approximators to construct a final approximator. Both gaussian processes approximators use the same options once specified by user for the `VFGP` approximator.

Detailed description: Detailed description of this technique requires more space and more complex notation, so it is available in Appendix A.

4.1.4 SparseVariableFidelityGaussianProcess

Short name: SVFGP

General description: `SVFGP` is a gaussian-processes-based technique. This technique expands applicability of gaussian-processes-based techniques to large data sets. Naturally, this technique is based on **VFGP**, but can be applied to large sample sizes. So, one can calculate AE values and use interpolation mode for `SVFGP`.

Strengths and weaknesses: The `SVFGP` shares most of the pros and cons for **VFGP**, but can be applied to large data sets.

Options: This technique has the same properties and use the same options from **GTApprox**. See Appendix B or [8]. `SVFGP` uses two gaussian processes approximators to construct a final approximator. Both gaussian processes approximators use the same options specified by user for the `SVFGP` approximator.

Detailed description: Detailed description of this technique is available in Appendix A.

4.2 General description of presented approximation techniques for Blackbox-based Data Fusion

In case black box for low fidelity model is available, an approximation is constructed in two steps:

- Create design of experiment for low fidelity model.
- Create approximation.

User can select a Design of Experiment technique for low fidelity data generation. Design of experiment consists of two separate steps. At the first step we include to the learning set for low fidelity data all points from $\mathbf{X}_h$. At the second step we include a set of points generated using **GT DoE** tool [9]. All options from **GT DoE** are available in **GTDF**. Then we evaluate low fidelity black box for obtained design of experiment.

Two approximation techniques are available for **BB DF**:

- `DiffApprox` (`DA`) — approximation is a sum of low fidelity data approximation and difference between low and high fidelity data approximation,
- `VariableFidelityGaussianProcess` (`VFGP`) — approximation using multi-task gaussian processes [1, 14] based approximation.

Techniques are similar to the described above and use the same sets of available options.

4.3 Limitations

The restrictions for sample sizes N_l, N_h are imposed. They depends on input dimension and internal validation options. Also restrictions vary on the base of technique selected. For `HighFidelityApprox` and `DiffApprox` restrictions depend purely on internal validation options. If `InternalValidation` is `off` for `HighFidelityApprox` minimal high fidelity sample size N_h is 1, minimal low fidelity sample size N_l is 0, because for this technique low fidelity sample isn't used. If `InternalValidation` is `off` for `DiffApprox` minimal high fidelity sample size N_h is 1, minimal low fidelity sample size N_l is also 1. For `VariableFidelityGaussianProcess` and `SparseVariableFidelityGaussianProcess` if $N_h \leq 2d_{\text{in}} + 2$ or $N_l \leq 2d_{\text{in}} + 2$, where d_{in} is input dimension of the data, an error with the corresponding error code will be returned. So, in such cases one should use `HighFidelityApprox` or `DiffApprox`, however, surrogate model quality can be low due to small sample sizes.

The maximum size of the training sample which can be processed by the tool is primarily determined by the users hardware. Necessary hardware resources depend significantly on the specific technique. See descriptions of individual techniques.

Chapter 5

Selection of approximation technique

GTDF tool allows automatic technique selection. However, user can select technique manually by setting option `Technique`.

5.1 Manual selection

The user may specify the approximation technique by setting the option `GTDF/Technique`, which may have the following values:

- `Auto` — best algorithm will be determined automatically,
- `HFA` — High Fidelity Approx algorithm will be used,
- `DA` — Difference Approximation algorithm will be used,
- `VFGP` — Variable Fidelity Gaussian Process algorithm will be used.
- `SVFGP` — Sparse Variable Fidelity Gaussian Process algorithm will be used.

5.2 Automatic selection

Automatic technique selection is affected by:

- high-fidelity sample size N_h,
- low-fidelity sample size N_l, and
- the state of `AE` switch (is Accuracy Evaluation required or not).

Selection rules are summarized below (note also that if Internal Validation is required, it affects sample size thresholds).

- If $N_h \leq 2d_{\text{in}} + 2$, and a low-fidelity sample is available ($N_l > 0$), `DA` is selected; else if $N_l = 0$, the tool selects `HFA`. In both cases `AE` and `ExactFit` are not available, and setting any of these requirements leads to an error: N_h is too low to provide `AE` or exact fit.

- If $N_h > 2d_{\text{in}} + 2$, but $N_l/N_h < 1.2$, the tool uses `HFA` since the low-fidelity sample size is not enough to build a required low-fidelity approximation model. To select approximation technique for `HFA` we use decision tree from **GTApprox** (See Appendix C and figure C.2).
- If both $N_h > 2d_{\text{in}} + 2$ and $N_l/N_h \geq 1.2$, technique selection further depends on sample sizes:
  - If both N_h and N_l are less than 1000, `VFGP` is selected.
  - If either of N_h, N_l is more than 1000, and `AE` is not required, the tool selects `DA`, and an approximation technique is selected following the **GTApprox** decision tree (decisions are based on the maximum value from N_h, N_l). In the same case but with `AE` required, the tool always selects `SVFGP`.

The logic of automatic technique selection is also shown on Figure 5.1.

5.3 Automatic selection for black box GTDF

In case black box is available for low fidelity model, the decision is based only on the size N_h of high fidelity model set of points.

Design of experiments for the **BB DF** includes all points from the set X_h. Additional input points are generated by **GT DOE**. For all obtained points X_l low fidelity outputs are calculated. Number N_l of points for low fidelity model $\mathbf{X}_l$ depends on size N_h of high fidelity data set. Dependency is depicted in figure 3.1. Number of additional points to generate is equal $N_l - N_h$. Note, that tool proceeds the case when blackbox return not only numbers but can return `NaN` or `Inf` for points from some region.

To select approximation algorithm the following procedure is used.

- If $N_l^{true} = 0$ tool returns an error because blackbox can't produce acceptable values.
- If $2d_{\text{in}} + 2 \geq N_l^{true} > 0$ or $2d_{\text{in}} + 2 \geq N_h > 0$ tool checks for `AE` and `ExactFit` requirements. If `AE` or `ExactFit` is required tool returns an error because it can't provide `AE` and `ExactFit` with produced low fidelity sample. If user doesn't require `AE` and `ExactFit` `DA` for `BB` technique is used.
- If both N_h and N_l^{true} are greater than $2d_{\text{in}} + 2$ we check for high fidelity sample size
  - If $N_h \leq 1000$ then we use `VFGP` for `BB` technique.
  - If $N_h > 1000$ then we use `DA` for `BB` technique. In this case `ExactFit` isn't available.

The procedure described above is presented in figure 5.2.

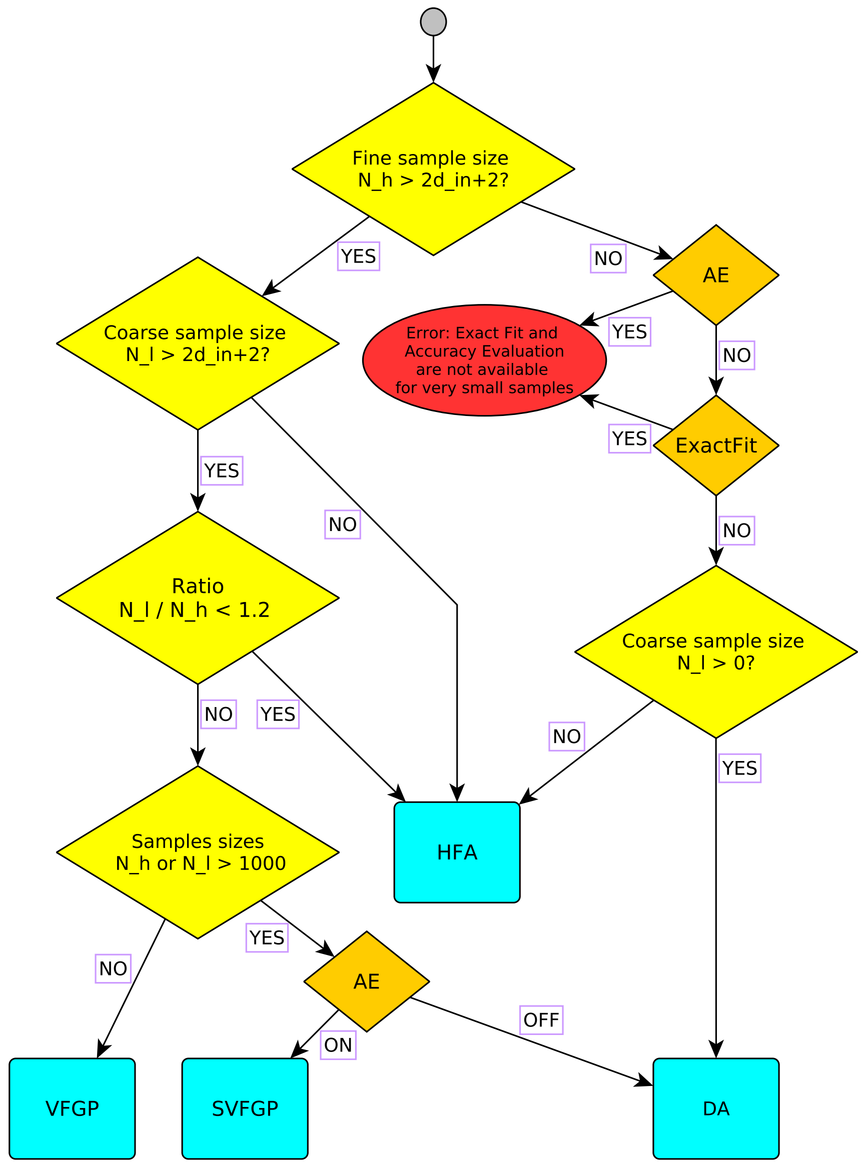


Figure 5.1: Decision tree for **GTDF**

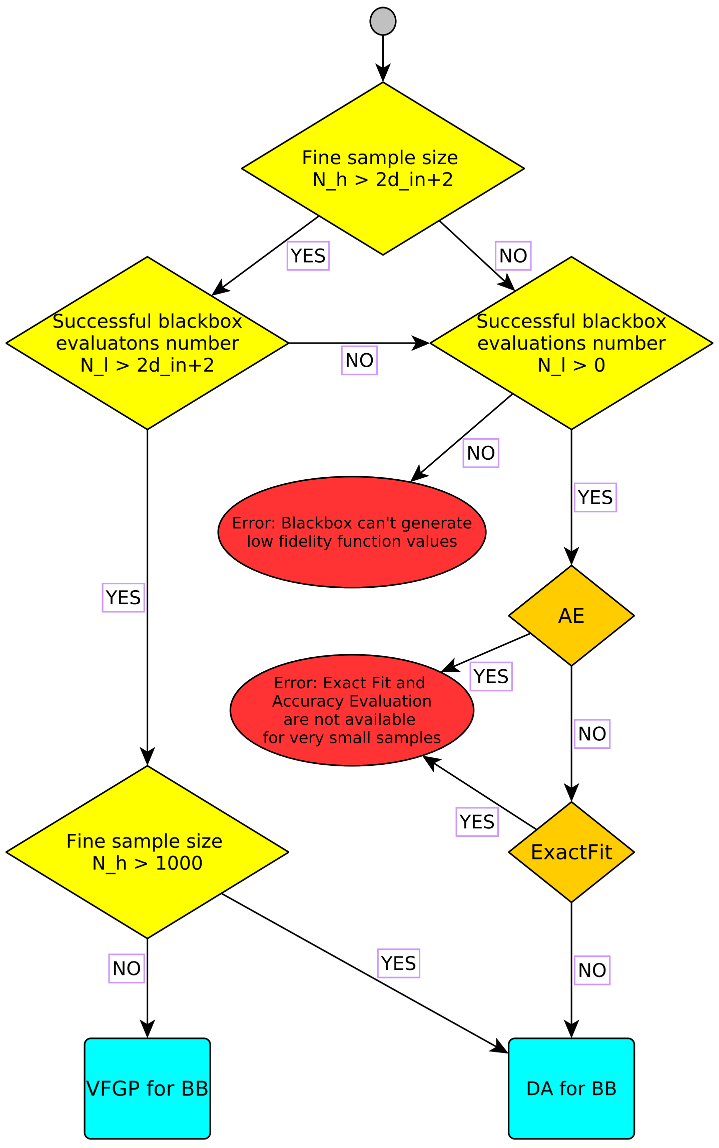


Figure 5.2: Decision tree for **GTDF** for **BB** available

Chapter 6

User configurable options

In some cases, it is desirable to adjust the type or properties of the approximation according to the prior domain–specific knowledge of the problem or specific requirements to the approximation. **GTDF** supports several special approximation modes which can be useful in some applications. To set **GTDF** options the user has to set an option `OptionName` like `GTDF/OptionName`, to set options for **GTApprox** approximator, which is used inside **GTDF**, the user has to set `GTApprox/OptionName`.

6.1 Interpolation mode

The interpolating mode can be set by the option `InterpolationRequired` (default value is `off`). If this switch is turned `on`, then the constructed approximation will go through the high fidelity sample points. If the switch is `off`, then no interpolation condition is imposed, and the approximation can be either interpolating or non-interpolating depending on which fits the training data best.

The interpolating mode is computationally demanding and therefore restricted to moderately sized training samples. **GT DF** provides interpolation in case proper **DF** technique and approximation technique from **GTApprox** are used (note, that only GP, SGP and HDAGP can be used in interpolating mode, it is not supported by HDA, 1D Splines and LR, see Appendix B.3).

The following guidelines are important in choosing whether to switch the strict interpolation mode `on` or `off`:

- If the approximation has a low error on the training sample (in particular, if it is a strict interpolation), it does not mean that this approximation will be just as accurate outside of the training sample. Very often (though not always), requiring an excessively small error on the training sample leads to an excessively complex approximation with low predictive power – a phenomenon known as *overtraining* or *overfitting* (see, e.g., [12, 24]). If the strict interpolation mode is `off`, **GTDF** attempts to avoid overtraining.

- The interpolation mode is inappropriate for noisy models. Also, if the training sample is highly irregular or the approximated function is known to be singular, then the interpolation mode is not recommended, as in this case the approximation tends to be numerically unstable. On the whole, strictly interpolating approximations are more flexible but less robust than non-interpolating ones.

- The interpolation mode may be useful, e.g., if the default approximation (with the interpolation mode `off`) appears to be too crude. In this case, turning the interpolation mode `on` may (but is not guaranteed to; the opposite effect is also possible) increase the accuracy.

Because of numerical limitations and round-off errors, minor discrepancies can be observed in some cases between the training sets and the interpolating approximations constructed by **GTDF**. These discrepancies typically have relative values $\lesssim 10^{-5}$ and are considered negligible.

A script named `interpolationExample.py` which demonstrates a difference between interpolation and approximation mode is presented in the supporting materials.

6.2 Compensating low fidelity function bias

This options is dedicated to handle important special case of low fidelity function. We suppose that the low fidelity function has a bias with respect to the high fidelity function. Namely there exists a vector S such that it holds

$$f_l(X) \approx f_h(X + S)$$

for X in the design space $\mathbb{X}$.

If one knows from a subject domain about existence of such a peculiarity in the data then one should turn `GTDF/UnbiasLowFidelityModel` option `'on'`, otherwise one should turn `GTDF/UnbiasLowFidelityModel` option `'off'`. If this options is `'on'` the tool tries to compensate bias for low fidelity model. As `HFA` doesn't use low fidelity data this technique ignores the option `GTDF/UnbiasLowFidelityModel` value. Default value for this option is `'off'`. For demonstration of this option usage see subsection 7.1.3.

6.3 Componentwise training of the approximation

In case of the multidimensional output of the response function ($d_{\text{out}} > 1$), there are, in general, two modes to construct the approximation:

1. Jointly for all scalar components of the output;
2. Separately for each scalar component of the output.

The choice of the mode is regulated by the option `Componentwise` . The default value of this option is `off`, i.e. the first mode is used. On the whole, the differences between the two modes are:

- the joint mode is faster;
- the joint mode attempts to take into account the possible dependency between different components of the output (e.g., this is so if different components of the output describe values of a smooth process at different time instances or describe different points on some family of curves or surfaces depending on some parameters);
- the total approximation obtained with the joint mode may have (but not always does) a lower overall accuracy than the set of separately obtained approximations for each component.

In some cases, however, changing the mode may have no or very little effect on the approximation.

6.4 Control over training time and accuracy of approximation

As **GTDF** normally constructs approximations by complex nonlinear techniques, this process may take a while. In general, the quality of the model trained by **GTDF** is positively correlated with the time spent to train it. **GTDF** contains a number of parameters affecting this time. The default values of the parameters are selected so as to reasonably balance the training time with the accuracy of the model, but in some cases it may be desirable to adjust them so as to decrease the training time at the cost of decreasing the accuracy, or increase the accuracy at the cost of increasing the training time. All available techniques with similar options works nearly equal time. Only `HFA` doesn't use low fidelity data and works faster because of this reason. Therefore, **GTDF** has a special option `Accelerator` which allows the user to tune the training time by simply choosing the level from 1 to 5; the detailed specific options of the approximation techniques are then set to pre-determined values. Increasing the value of `Accelerator` reduces the training time and, in general, lowers the accuracy. In general parameter `Accelerator` of **GTDF** works like the same parameter of **GTApprox**.

Some recommendations for selection technique inside **GTApprox** for training are given below.

- The fastest approximation algorithms of **GTApprox** are, by far, SPLT and LR (see Appendix B.3). They have, however, a limited applicability: SPLT is exclusively for 1D, and LR is too crude in many cases.

- The only nonlinear approximation algorithm effectively available in **GTApprox** for very large training sets (larger than 1000) in dimensions higher than 1 is HDA. This algorithm can be quite slow on very large training sets, but it has several options which can be adjusted to decrease the training time. See Appendix B.3.3 for details.

6.5 Accuracy evaluation

Accuracy Evaluation (AE) is a part of **GTDF** whose purpose is to estimate the accuracy of the constructed approximation at different points of the design space. If AE is turned `on`, then the constructed model contains, in addition to the approximation $\hat{f}_h : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$, the AE prediction $\sigma : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}_+^{d_{\text{out}}}$.

AE is turned `on` or `off` by the option `AccuracyEvaluation` (default value is `off`). AE prediction is performed separately for each of the d_{out} scalar outputs of the response.

Let us make four additional notes about AE:

- The AE prediction is only an "educated guess" of approximation's error. It is not possible, in general, to predict the exact values of the error.

- The AE prediction is usually more efficient in terms of the *correlation* with the actual approximation errors rather than *reproduction* of these errors.

- If the approximation $\hat{f}_h$ is interpolating, then at the high fidelity train points X_i^h from D_h holds $\sigma(X_i^h) \approx 0$;
- AE is available in **GTApprox** for the following approximation techniques: SPLT, GP, HDAGP (see Appendix B.3), used in **DF** procedures.

Other useful information about AE can be found in the manual for **GTApprox** [8].

We provide a python script which demonstrates accuracy estimation for a simple one-dimensional function in the script `aeExample.py`.

6.6 Internal validation

The Internal Validation (IV) procedure is an optional part of **GTDF** providing an estimate of the expected overall accuracy of the approximation algorithm on the user–supplied training data. This estimate is obtained by a (generalized) cross–validation of the algorithm on the training data.

The result of this procedure is a table with errors of several types, written to the log file and to the constructed model's info file.

The procedure can be turned `on/off` by the option `InternalValidation` (the default value is `off`).

The procedure depends on the following parameters which can be set up through the advanced options of **GTDF**:

- The number N_{ss} of subsets that the original training set D_h is divided into, where $2 \le N_{\text{ss}} \le |N_h|$. This number is set by the option `IVSubsetCount`.
- The number N_{tr} of training/validation sessions, where $1 \le N_{\text{tr}} \le N_{\text{ss}}$. This number is set by the option `IVTrainingCount`.
- The seed for the pseudo–random division of D_h into subsets. This seed is set by the option `IVRandomSeed`.
- `GTDF/IVSavePredictions` allows one to save predictions obtained during Internal Validation. If this options is `on`, one can get information about predicted outputs and true outputs for points used in test samples during IV. For example, now one can produce scatter plots with IV results. See description of option `GApprox/IVSavePredictions` in **GTApprox** user manual for more details.

The default values of the parameters N_{ss}, N_{tr} are given by

$$\begin{aligned} N_{\text{ss}} &= \min(10, |N_h|), \\ N_{\text{tr}} &= \left\lceil \min\left(|N_h|, \frac{100}{|N_h|}\right) \right\rceil, \end{aligned} \tag{6.1}$$

where $\lceil a \rceil$ denotes the smallest integer not less than a.

The Internal Validation accompanies the construction of the approximation. If IV is turned on, the model construction procedure includes the following steps:

1. From the options' values and the properties of the training sets D_l, D_h, the tool determines the appropriate approximation algorithm $\mathcal{A}$ to be used when constructing the main approximation f provided to the user.

2. After that, the tool starts the Internal Validation of the algorithm $\mathcal{A}$ on the sample D_h.
   (a) The set D_h is randomly divided into N_{ss} disjoint subsets $(D_k)_{k=1}^{N_{\text{ss}}}$ of approximately equal size.
   (b) For each $k = 1, \dots, N_{\text{tr}}$, an $\mathcal{A}$-based approximation f_k is trained on the subset $D_h \setminus D_k$ and the whole set D_l, and its errors $E_{k,m}$ of one of the three standard types, MAE, RMS and RRMS (see below) are computed on the complementary test subset D_k, separately for each scalar component $m = 1, \dots, d_{\text{out}}$ of the output.
   (c) The cross–validation errors $(E_m^{\text{cv}})_{m=1}^{d_{\text{out}}}$ are computed as the *median* values of the errors $E_{k,m}$ over the training/validation iterations $k = 1, \dots, N_{\text{tr}}$.
   (d) The total cross–validation errors $E_{\text{mean}}^{\text{cv}}, E_{\text{rms}}^{\text{cv}}, E_{\text{max}}^{\text{cv}}$ are computed as the *mean*, *root-mean-squared* or *maximum* values, respectively, of the cross–validation errors $(E_m^{\text{cv}})_{m=1}^{d_{\text{out}}}$ over the output components $m = 1, \dots, d_{\text{out}}$.
3. The obtained errors $(E_m^{\text{cv}})_{m=1}^{d_{\text{out}}}$ and $E_{\text{mean}}^{\text{cv}}, E_{\text{rms}}^{\text{cv}}, E_{\text{max}}^{\text{cv}}$, where E is MAE, RMS or RRMS, are written to the log file.
4. Finally, the main approximation f is trained using all the training data D_h and D_l and saved in the constructed model.

In **GTDF**, the following types of errors are computed during Internal Validation:

- Mean absolute error,

$$\text{MAE}_{k,m} = \frac{1}{|D_k|} \sum_{(X, f_h(X)) \in D_k} \left| f_k^{(m)}(X) - f_h^{(m)}(X) \right|,$$

  where $f_h^{(m)}(X)$ is the m'th output component of the training point $(X, f_h(X))$.

- Root-mean-squared error,

$$\text{RMS}_{k,m} = \left(\frac{1}{|D_k|} \sum_{(X, f_h(X)) \in D_k} \left| f_k^{(m)}(X) - f_h^{(m)}(X) \right|^2 \right)^{1/2}$$

- Relative root-mean-squared error,

$$\text{RRMS}_{k,m} = \frac{\left(\frac{1}{|D_k|} \sum_{(X, f_h(X)) \in D_k} \left| f_k^{(m)}(X) - f_h^{(m)}(X) \right|^2 \right)^{1/2}}{\left(\frac{1}{|D_k|} \sum_{(X, f_h(X)) \in D_k} \left| \overline{f_h^{(m)}(X)}^{D_h \setminus D_k} - f_h^{(m)}(X) \right|^2 \right)^{1/2}}, \tag{6.2}$$

  where $\overline{f_h^{(m)}(X)}^{D_h \setminus D_k}$ is the mean of $f_h^{(m)}(X)$ on the training subset $D_h \setminus D_k$.

Chapter 7

Usage examples

To demonstrate a way of using **GTDF** we present a number of examples. This chapter contains both artificial and real data examples. Four methods, described in chapters 4 and appendix A, are considered:

- `HFA` — we use only high fidelity data to build an approximation,
- `DA` — approximation is a sum of low fidelity data approximation and difference between low and high fidelity data approximation,
- `VFGP` — we use variable fidelity gaussian process approximation.
- `SVFGP` — we use variable fidelity gaussian process approximation.

To compare available **GTDF** techniques different approximation quality measures are used. They are calculated on a test sample of high fidelity function values $D_t = \{f_h(X_i), X_i\}_{i=1}^{N_t}$, which we don't use during an approximation building. We measure mean average error **MAE**, mean square error **MSE**, max error **MAX**, 95%-quantile of errors **Q95** and 99%-quantile of errors **Q99**.

$$\begin{aligned} MSE(\hat{f}(X)) &= \frac{1}{N_t}\sum_{i=1}^{N_t}(f_i - \hat{f}_i)^2, \\ MAE(\hat{f}(X)) &= \frac{1}{N_t}\sum_{i=1}^{N_t}|f_i - \hat{f}_i|, \\ MAX(\hat{f}(X)) &= \max_{i=1,\dots,N_t}|f_i - \hat{f}_i|, \end{aligned} \tag{7.1}$$

where $\hat{f}_i = \hat{f}(X_i)$ is an approximation given by the corresponding **GTDF** model, $f_i = f(X_i)$ is a high fidelity function value. To get **Q95**, **Q99** we calculate 95% and 99% quantiles for the set of errors for test sample $\{|f_i - \hat{f}_i|\}_{i=1}^{N_t}$. Quantiles make sense only if test sample size is bigger than 20, so if test sample size is smaller than 20 we don't include them because both quantiles would coincide with **MAX** error in this case.

7.1 Artificial example

Functions used for data generation are presented in table 7.1.

High fidelity function $f_h(X)$	Low fidelity function $f_l(X)$
$\lvert\sum_{i=1}^{m}\cos^4(x_i)-2\prod_{i=1}^{m}\cos^2(x_i)\rvert\left(\sum_{i=1}^{m} i x_i^2\right)^{\frac{-1}{2}}$	$f_h(X)\exp\left[-\frac{1}{2}\sum_{i=1}^{m}\frac{(x_i-5)^2}{\sigma^2}\right]+s$

Table 7.1: Model function

For the low fidelity function f_l parameter σ equals 4, parameter s equals 0.1. Input dimension $m = d_{\text{in}}$ equals 2. We use design space $x_i \in (0, 10), i = 1, 2$ to generate learning sample and test sample. Low and high fidelity functions are plotted in figure 7.8. To compare different techniques we calculate **MSE** error (7.1). We varied size of high fidelity data and low fidelity data and calculate errors for 20 random sets of points in the design space. Dependence of **MSE** error on the low and high fidelity sample sizes for different techniques are presented in figure 7.9. One can see, that increasing both low and high fidelity data samples decreases **MSE** error. Also, techniques `DA`, `VFGP` that use both low and high fidelity samples work better than `HFA` that uses only high fidelity data.

7.1.1 Sparse variable fidelity gaussian processes

We demonstrate application of `SVFGP` technique in the following experiment. We use Rastrigin function as a high fidelity function:

$$f_h(X) = 20 + \sum_{i=1}^{2}\left(x_i^2 - 10\cos(2\pi x_i)\right).$$

A low fidelity function is

$$f_l(X) = f_h(X) + 0.2\sum_{i=1}^{2}\left(\frac{\tilde{x}_i}{c}+1\right)^2,$$

where $\tilde{x}_i = \frac{x_i}{10.28} + 0.5$. We generate points for low and high fidelity sets from uniform distribution on $[-5.12, 5.12]^2$. The experiment workflow is:

- Generate a set of high fidelity points with size $N_h = 100$.
- Generate a set of low fidelity points with size $N_l^{full} = 2000$.
- Select a subset from full low fidelity set of points with size $N_l \in \{1000, 1050, 1100, \ldots, 2000\}$.
- Get **MSE** for each pair of sets of low and high fidelity points while using `SVFGP` technique.
- Get **MSE** for the first one thousand points of low fidelity set while using `VFGP` technique.
- Get **MSE** for all available points while using `VFGP` technique.

The results are presented in figure 7.1. One can see, that we obtain more accurate approximation using `SVFGP` than using `VFGP`, however, in case we use all available points to build `VFGP` model the results are much better. To compare training time for each method we plot figure 7.2. Again, we compare learning time for full sample `VFGP`, small sample `VFGP` and full sample `SVFGP`. One can see, that we don't increase learning time using `SVFGP`, but for full sample `VFGP` learning time growth is cubic in sample size.

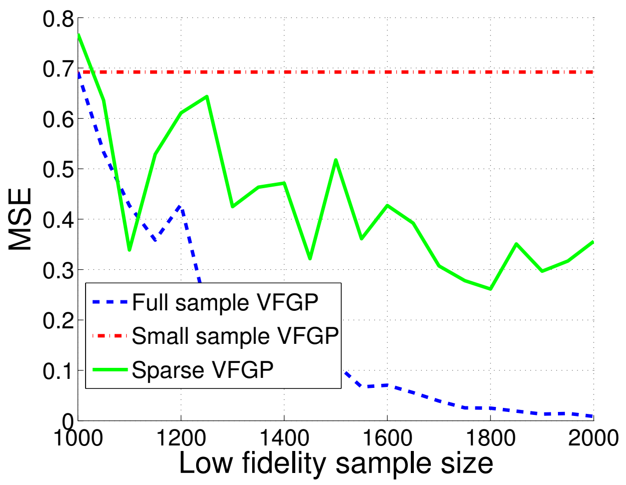


Figure 7.1: Comparison of SVFGP and VFGP approximation error. **Note:** This image was obtained using an older pSeven Core version. Actual results in the current version may differ.

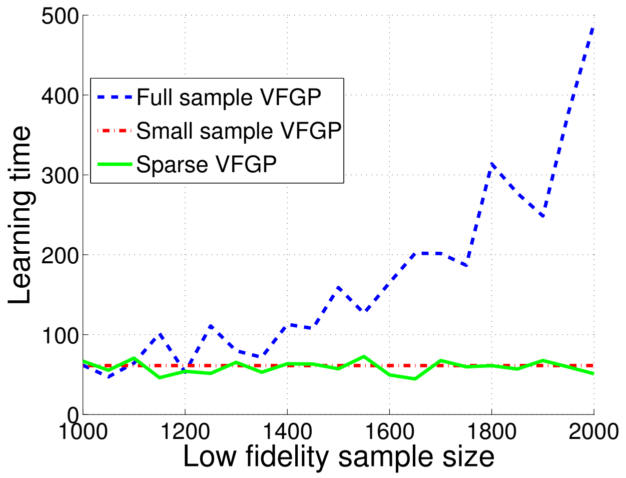


Figure 7.2: Comparison of SVFGP and VFGP learning time. **Note:** This image was obtained using an older pSeven Core version. Actual results in the current version may differ.

7.1.2 Blackbox-based variable fidelity gaussian processes

Here we demonstrate application of Blackbox-based Variable Fidelity Gaussian Processes technique on basis of some artificial problem. Varying level of fidelity for low fidelity function we compare results for different fidelities. We use Rastrigin function as a high fidelity function:

$$f_h(X) = 20 + \sum_{i=1}^{2} \left(x_i^2 - 10\cos(2\pi x_i)\right).$$

A low fidelity function is

$$f_l(X) = f_h(X)(0.8 + 0.1(\tilde{x}_1 + \tilde{x}_2)) + c\sum_{i=1}^{2}\sin(\tilde{x}_i^2) + \varepsilon,$$

where $\tilde{x}_i = \frac{x_i}{10.28} + 0.5$, ε is a standard normal variable, $\varepsilon \sim \mathcal{N}(0,1)$, c is our fidelity coefficient. If c is small low and high fidelity functions slightly differ from each other. If c is large low and high fidelity functions significantly differ from each other.

We use $N_h = 20$, i.e. $N_l = 100$ according to the dependency presented in figure 3.1. The results is plotted in figure 7.3. We see, that if low and high fidelity functions are close enough, blackbox usage dramatically improves approximation quality, but difference vanishes if low fidelity function doesn't provide information about high fidelity function.

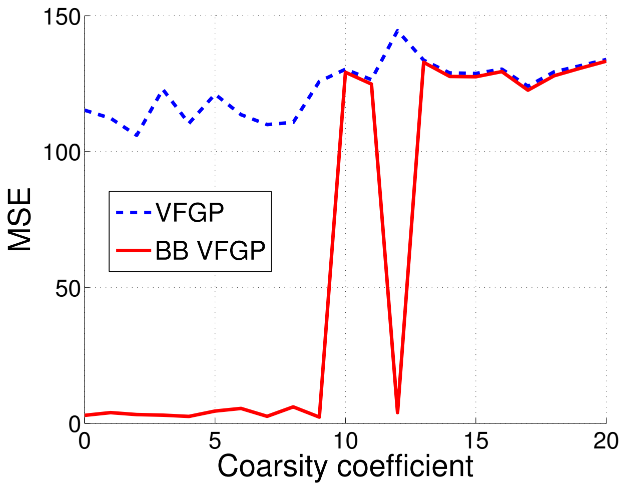


Figure 7.3: Comparison of `VFGP` and blackbox-based `VFGP`. **Note:** This image was obtained using an older pSeven Core version. Actual results in the current version may differ.

7.1.3 Compensating low fidelity function bias

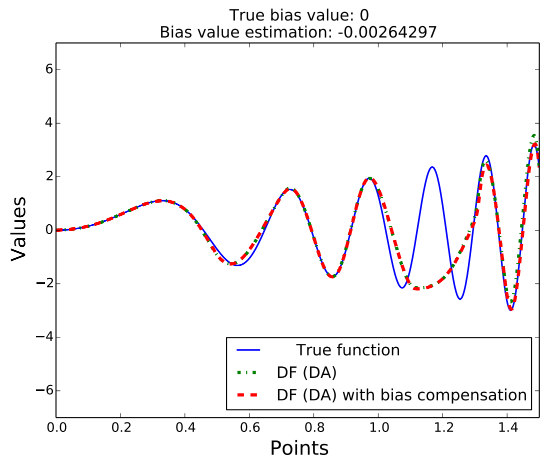
To demonstrate how **GTDF** tool can handle low fidelity model bias we provide a simple one-dimensional example. In this case high and low fidelity functions are:

$$f_h(x) = (x^2 + 1)\sin(15x^2),$$
$$f_l(x) = f_h(x - s),$$

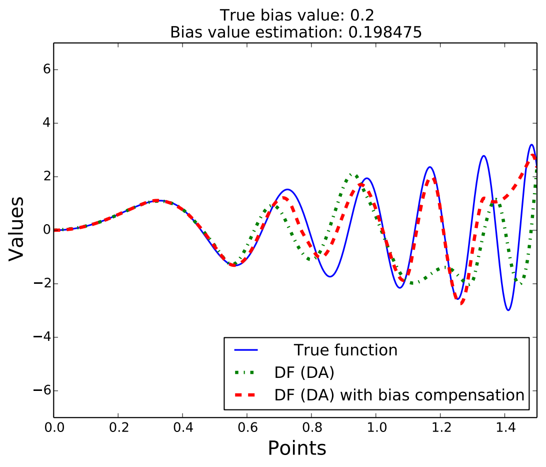
where s is a bias value.

We vary bias value to compare behaviour of **GTDF** with and without compensation of the low fidelity function bias option used. Sample size for the high fidelity function equals 12, sample size for the low fidelity function equals 50. High fidelity sample points were equally spaced on the interval $[0, 1.5]$. Low fidelity sample points were selected from the uniform distribution on $[0, 2]$. For surrogate model construction for `GTDF/UnbiasLowFidelityModel` option turned `'on'` and `'off'` we use Difference approximation (`DA`) technique from **GTDF**.

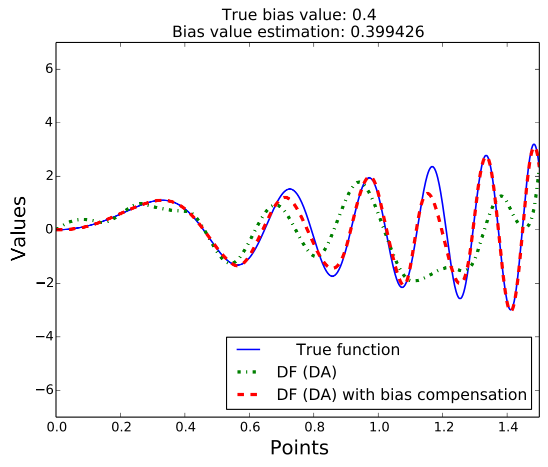
Obtained approximations are depicted in figures 7.4. One can see that as we can handle bias for low fidelity function during approximation we obtain more precise surrogate models. Moreover, with samples of such size bias values estimations are close to the true bias values. See also script `lowFidelityBiasCompenstationExample.py` in additional materials for more details.



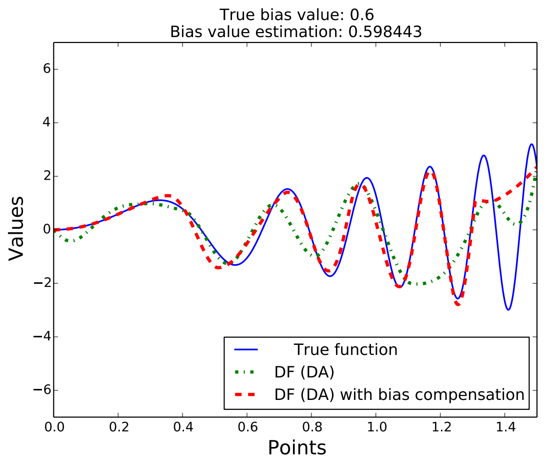
(a) Bias s equals 0



(b) Bias s equals 0.2



(c) Bias s equals 0.4



(d) Bias s equals 0.6

Figure 7.4: Compensating low fidelity function bias using `GTDF/UnbiasLowFidelityModel` option

7.2 Real data examples

7.2.1 Modelling aerodynamic lift and drag coefficients for a fixed airfoil

This experiment demonstrates a number of DF techniques applications to modeling aerodynamic lift coefficient C_l and drag coefficient C_d of a fixed airfoil as a function of one variable — angle of attack α. For experiment we fix Mach number $\mathrm{Ma} = 0.8$.

Design domain contains α from $[-2°, 3°]$. Two experiments were made. For the first experiment we used *Euler solver* with a high precision mesh as a high fidelity model and *Full-potential solver* as a low fidelity model. For the second experiment we used *Euler solver* with a high precision mesh as a high fidelity model and *Euler solver* with a low precision mesh as a low fidelity model. The mesh for the high fidelity model was significantly denser than the mesh for the low fidelity model. So, for the first experiment we use two different solvers to get lift and drag coefficients, for the second experiment we use one solver, but apply it to different meshes. We make 200 runs of the low fidelity model and 30 runs of the

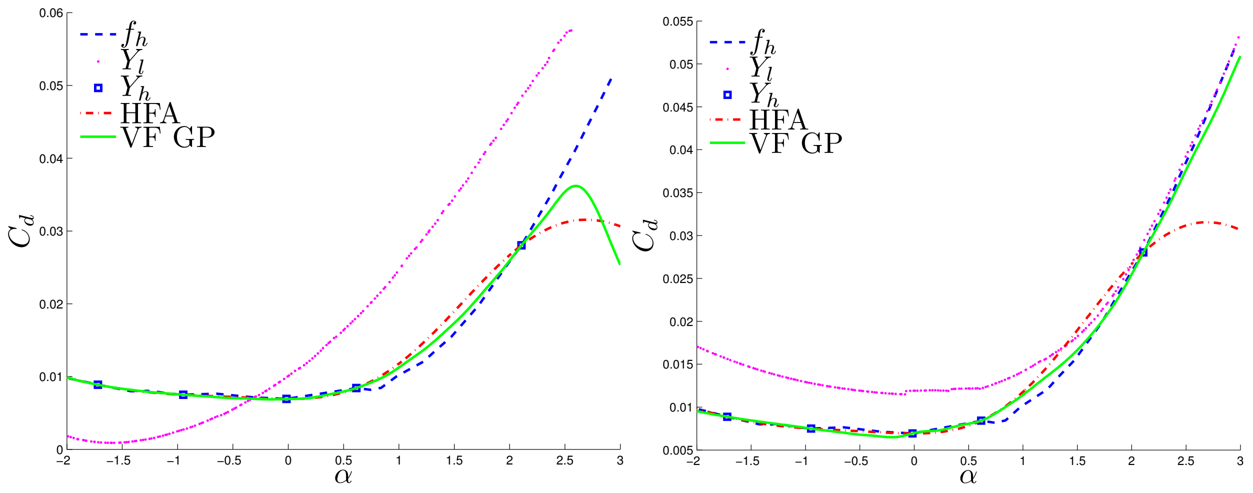
Low fidelity data model	**MSE** $\cdot 10^5$ for C_d			**MSE** $\cdot 10^6$ for C_l		
	HFA	DA	VFGP	HFA	DA	VFGP
Full-potential solver	4.21	83.30	4.51	1.52	2.41	1.88
Euler solver		3.80	1.00		1.01	1.18

Table 7.2: MSE for Airfoil problem. For convenience we multiply errors of C_d approximation by factor 10^5 and errors of C_l approximation by factor 10^6.

high fidelity model, angle of attack α was distributed uniformly in the design domain.

To verify proposed DF technique we split sample of the high fidelity model values to train and test several times. Train sample contains 10 points and test sample contains 20 points from the high fidelity model values. Train sample contains 120 points from the low fidelity model values. Each time we split sample we calculate errors (7.1) for a test sample. Than for all splittings we get mean errors. Results are presented in table 7.2. We see, that in case we use *Full-potential solver* as a low fidelity model, we can't improve results obtained using only high fidelity data. However, in case we use *Euler solver* as a low fidelity model, we do improve results obtained using only high fidelity data.

Examples of obtained approximations for C_d are presented in picture 7.5 (high fidelity data sample size equals 5). For the first experiment the dependency obtained using Full-potential solver significantly differs for the dependency obtained using Euler solver. So, we can't improve our approximation of drag coefficient using low fidelity data from the Full-potential solver. For the second experiment approximation quality significantly improves in case we use additional data from the Euler solver with low precision mesh. So, one should use precise enough low fidelity functions to improve approximation using **DF** techniques.



(a) *Full-potential solver* as low fidelity model **(b)** *Coarse mesh for Euler solver* as low fidelity model

Figure 7.5: Drag coefficient C_d approximation. **Note:** This image was obtained using an older pSeven Core version. Actual results in the current version may differ.

7.2.2 Modelling aerodynamic lift and drag coefficients

We approximate airfoil lift C_l and drag C_d coefficients. As high and low fidelity function we use tight and coarse meshes correspondingly for Euler equations solving.

	GP	SGP	VFGP	SVFGP
C_l	0.1671	0.0826	0.1284	0.0788
C_d	0.0114	0.0068	0.0084	0.0050

Table 7.3: Averaged approximation MSE for control sample

A usual airfoil parametrization specifies ordinates for a fixed set of absciss points, see figure 7.6 Typically, dimensionality of such a parametrization is about 59. Two problems rises in case we deal with this parametrization [3]. The first one is there is difficult to obtain new suitable airfoil using this parametrization. The second one is dramatic number of algorithm parameters which is proportional to problem dimensionality.

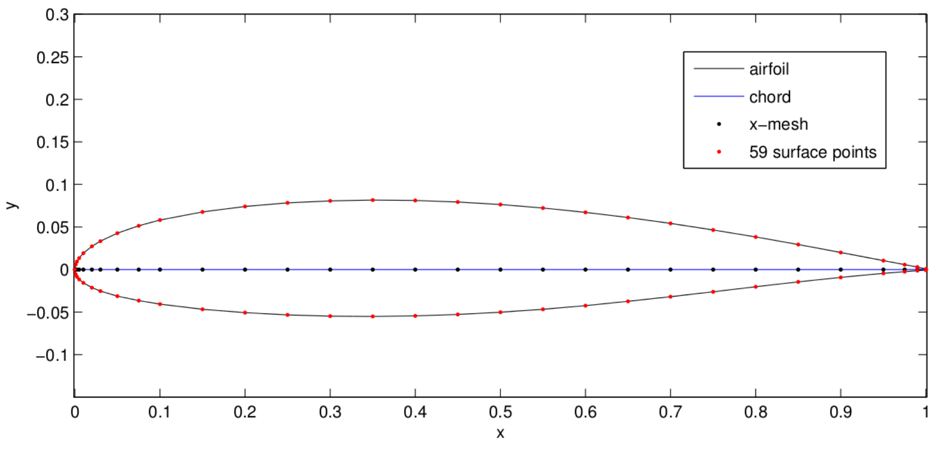


Figure 7.6: Airfoil parametrization

Both problems can be solved using dimension reduction technique. To get input variables X of dimension 12 we compress a depicted parametrization using PCA [18] and add angle of attack α to the list of variables. To build PCA dimension reduction model we use a set of 208 airfoils.

For this parametrization we calculate high fidelity function values for 400 and low fidelity function values for 200 airfoils. To estimate approximation performance we use cross-validation approach. We split the whole set of high fidelity points to learning set of size 350 and control set of size 50. To learn model we use learning set of high fidelity points and the whole set of low fidelity points. Remaining control set we use to estimate techniques performance on independent set of points.

To learn `GP` and `VFGP` models we use only 100 high fidelity points; to learn Sparse gaussian processes `SGP` and Sparse variable fidelity gaussian processes `SVFGP` models we use 350 high fidelity points. Obtained results are presented in table 7.3. We mention two conclusions of this examples:

- by incorporating low fidelity data into model learning we increase approximation quality,
- sparse techniques increase performance with no additional time cost.

7.2.3 Designing linear cellular alloys

Data of designing linear cellular alloys processes was presented in [20]. Data consists of the output from computer simulations for a heat exchanger used in an electronic cooling application. The response $f(X)$ is the total rate of steady state heat transfer of the device, which depends on a number of parameters. High fidelity data was obtained using slow simulation based on FLUENT finite element analysis. Low fidelity data was obtained using finite difference method. Evaluation of one point of high fidelity data is about two or three order of magnitude slower than evaluation of one point of low fidelity data.

Total number of available points was 36. To learn model we use 22 randomly points, to test — remaining 14 points of high fidelity data set. To learn model we use the whole low fidelity data set. Learning sample was randomly selected 100 times from the whole learning set to make error estimation more accurate. For each learning sample we run approximation building and estimate errors using test sample. Then we estimate mean and max errors for a given set of runs. Mean errors are presented in table 7.4. Max errors are presented in table 7.5. We significantly increase quality using `VFGP`. Both mean and max errors for `VFGP` are significantly better in comparison to `HighFidelityApprox` and `DiffApprox`.

Algorithm	MAE	MSE	MAX
`HFA`	1.8967	9.0097	7.0559
`DA`	1.4915	6.4871	6.5692
`VFGP`	1.3630	5.7837	6.4232

Table 7.4: Designing linear cellular alloys problem. Mean errors

Algorithm	MAE	MSE	MAX
`HFA`	4.641	119.63	39.641
`DA`	2.974	22.463	16.20
`VFGP`	2.3253	15.126	11.077

Table 7.5: Designing linear cellular alloys problem. Max errors

7.2.4 Fluidized Bed Process Experimental data

Dewettinck [11] proposed several associated computer models for predicting the steady-state thermodynamic operation point of a GlattGPC-1 fluidized-bed unit. The base of the unit consists of a screen and an air jump, with coating sprayers at the side of the unit. Data is available in [20]. Eight variables can potentially affect the steady-state thermodynamic operating point. Dewettinck fixed two of them.

So, the problem contains 28 points and 6 input variables. We fix the high fidelity learning sample size for the Fluidized Bed Process Experiment data to 16, remaining points were used for testing. Learning sample was randomly selected 100 times from the whole learning set to make error estimation more precise. The results are given in tables 7.6 and 7.7 for mean and max errors for all set of runs.

We get better results using both high fidelity and low fidelity data. Difference between `DA` and `VFGP` isn't significant, but max errors for `VFGP` are smaller.

We can increase performance in case we use `BB VFGP` technique to incorporate low fidelity values for test set. Dependence of mean square error with respect to learning set size is depicted in figure 7.7. We see that `BB VFGP` is the best choice for approximation building, however, difference between approximations quality becomes negligible in case we increase learning set size.

Algorithm	MAE	MSE	MAX
`HFA`	2.786	23.341	7.3424
`DA`	0.56467	0.80285	2.0958
`VFGP`	0.47338	0.65715	1.9761

Table 7.6: Fluidized Bed Process Experiment. Mean errors

Algorithm	MAE	MSE	MAX
`HFA`	7.2832	86.722	17.518
`DA`	1.1865	3.9993	5.6077
`VFGP`	1.1891	2.1394	3.9223

Table 7.7: Fluidized Bed Process Experiment. Max errors

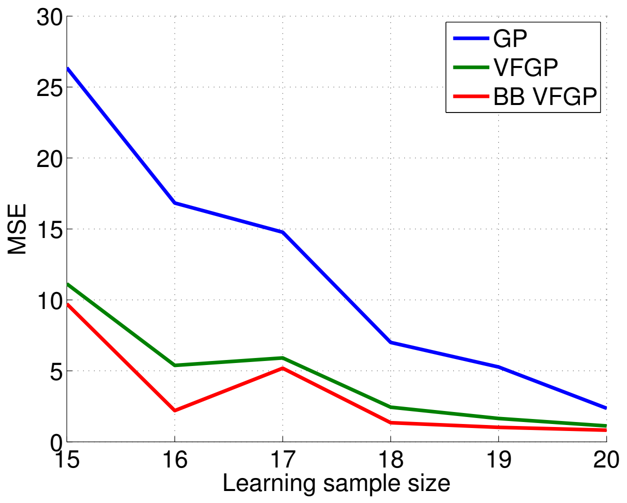
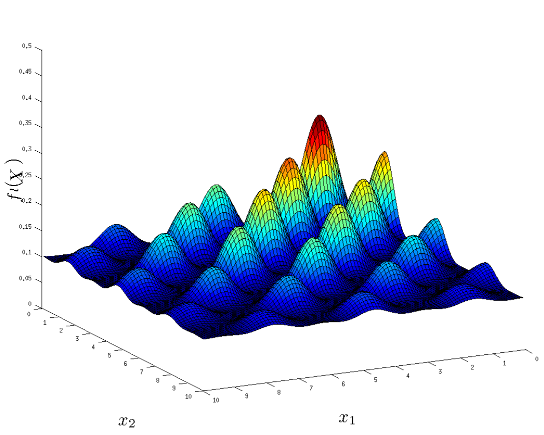
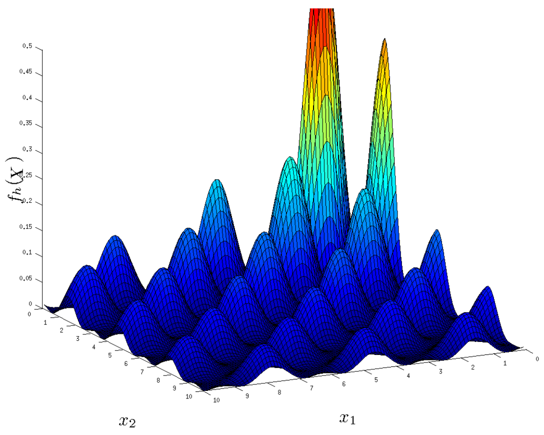


Figure 7.7: Fluidized Bed Process Experiment. Dependence with respect to sample size

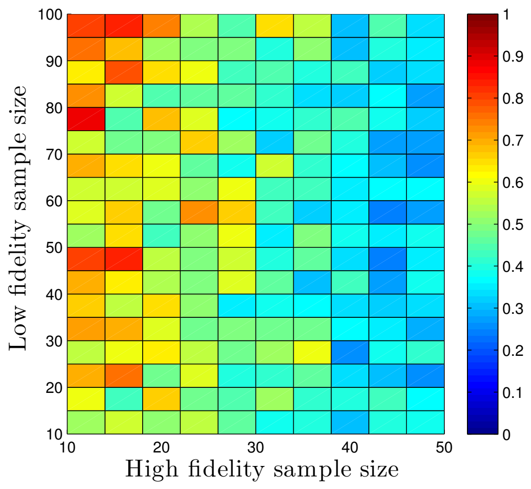


(a) Low fidelity function $f_l(X)$

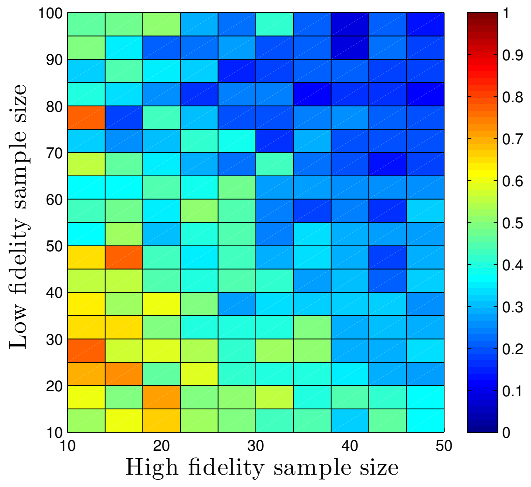


(b) High fidelity function $f_h(X)$

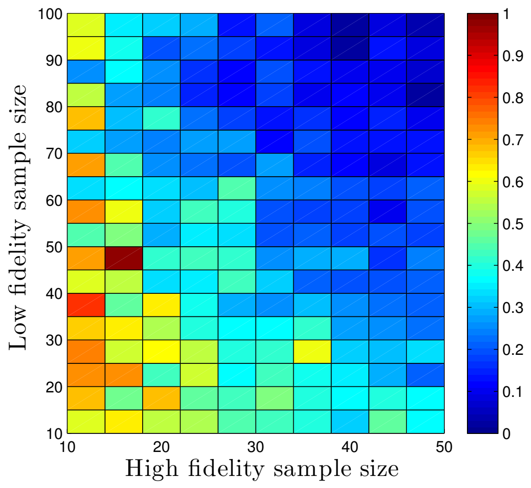
Figure 7.8: Model function



(a) High fidelity approximation



(b) Difference approximation



(c) Variable fidelity gaussian processes

Figure 7.9: Corresponding model function approximation errors. Colour of each square corresponds to the error value in the square vertex. Dark blue colour corresponds to the low **MSE** errors. Red colour corresponds to the high **MSE** errors.

Appendix A

Variable fidelity gaussian processes

Variable fidelity gaussian processes (**VFGP**) technique is based on gaussian processes [1, 14, 26]. To learn gaussian processes for **VFGP** we use algorithm described in [8] and [4, 5]. Any GP $f(X)$ is fully determined by its' mean functions $m(X) = \mathbb{E}[f(X)]$ and covariance functions $\operatorname{cov}(f(X), f(X')) = k(X, X') = \mathbb{E}[(f(X) - m(X))(f(X') - m(X'))]$. In the next section **VFGP** based approximation approach is described.

A.1 Used class of GPs

It is assumed that training data consists of two samples $D_l = (\mathbf{Y}_l, \mathbf{X}_l) = \{(f_l(X_i^l), X_i^l)\}_{i=1}^{N_l}$, $D_h = (\mathbf{Y}_h, \mathbf{X}_h) = \{(f_h(X_i^h), X_i^h)\}_{i=1}^{N_h}$. The sample $(\mathbf{Y}_l, \mathbf{X}_l)$ corresponds to a low fidelity function, the sample $(\mathbf{Y}_h, \mathbf{X}_h)$ corresponds to a high function function. The following model is assumed:

$$\begin{aligned} Y_i^l &= Y^l(X_i) = f_l(X_i) + \varepsilon_i^l, i = 1, 2, \ldots, N_l, \\ Y_i^h &= Y^h(X_i) = \rho(f_l(X_i) + \varepsilon_i^l) + f_d(X_i) + \varepsilon_i^d, i = 1, 2, \ldots, N_h, \end{aligned}$$

where $f_l(X_i)$ and $f_d(X_i)$ are independent gaussian processes and the independent noise terms ε^l and ε^d are modelled by gaussian white noises with zero mean and variance σ_l^2 and σ_d^2 correspondingly. By $f_h(X)$ denote the high fidelity function values $\rho f_l(X) + f_d(X)$ without noise. GP based regression is interpolating, when it is assumed no noise, therefore interpolation regime of **VFGP** is realized by setting both $\sigma_l^2 \approx 0$ and $\sigma_d^2 \approx 0$. Parameter ρ refers to the link between low fidelity and high fidelity models.

For each of GP $f_l(X)$, $f_d(X)$ we make the same assumptions. Let us describe these assumptions using some GP $f(X)$ as example. It is assumed that GP $f(X)$ has zero mean function $m(X) = \mathbb{E}[f(X)] = 0$ and covariance function $k(X, X')$, belonging to some parametric class of covariance functions $k(X, X'|a)$, where a is a vector of unknown parameters. The following two classes of covariance functions are considered, namely squared exponential covariance function

$$k(X, X'|a) = \sigma^2 \exp\left(-\sum_{i=1}^{d_{in}} \theta_i^2 (x_i - x_i')^s\right), s \in [1, 2], \tag{A.1}$$

where parameters $a = \{\sigma, \theta_i, i = 1, \ldots, d_{in}\}$, and Mahalanobis covariance function

$$k(X, X'|a) = \sigma^2 \exp\left(-(X - X')^{\mathrm{T}} A (X - X')\right), \tag{A.2}$$

where $A \in \mathbb{R}^{d_{in} \times d_{in}}$ is a positive definite matrix and parameters $a = \{\sigma, A\}$.

Under such assumptions the data from samples $(\mathbf{X}_l, \mathbf{Y}_l)$, $(\mathbf{X}_h, \mathbf{Y}_h)$ is modelled using two GPs with zero mean and covariance functions

$$c_l(X, X') = \text{cov}(Y^l(X), Y^l(X')) = \mu_l(k_l(X, X') + \sigma_l^2 \delta(X - X')),$$
$$c_d(X, X') = \text{cov}(Y^d(X), Y^d(X')) = \mu_d(k_d(X, X') + \sigma_d^2 \delta(X - X')),$$

where $\delta(X)$ is a delta function. Using this covariances we can write covariances between high and low fidelity points:

$$\text{cov}(Y^h(X), Y^l(X')) = \rho\mu_l(k_l(X, X') + \sigma_l^2 \delta(X - X')),$$
$$\text{cov}(Y^h(X), Y^h(X')) = \rho^2\mu_l(k_l(X, X') + \sigma_l^2 \delta(X - X')) + \mu_d(k_d(X, X') + \sigma_d^2 \delta(X - X')).$$

Thus, a posteriori (with respect to the given training sample) mean value of the process for some test point X for the high fidelity function $f_h(X)$ takes the form

$$\hat{f}_h(X) = k(X)K^{-1}\mathbf{Y}, \tag{A.3}$$

where

$$k(X) = \begin{pmatrix} \rho K_l(X, \mathbf{X}_l) \\ \rho^2 K_l(X, \mathbf{X}_h) + K_d(X, \mathbf{X}_h) \end{pmatrix}, \tag{A.4}$$

matrix K is constructed using four submatrices:

$$K = K(\mathbf{X}) = \begin{pmatrix} K_l(\mathbf{X}_l, \mathbf{X}_l) & \rho K_l(\mathbf{X}_l, \mathbf{X}_h) \\ \rho K_l(\mathbf{X}_h, \mathbf{X}_l) & \rho^2 K_l(\mathbf{X}_h, \mathbf{X}_h) + K_d(\mathbf{X}_h, \mathbf{X}_h) \end{pmatrix},$$

matrix $\mathbf{Y}$ is the concatenation of two matrices:

$$\mathbf{Y} = \begin{pmatrix} \mathbf{Y}_l \\ \mathbf{Y}_h \end{pmatrix}.$$

In this notation $K_l(\mathbf{X}, \mathbf{X}')$ is a matrix of pairwise covariances:

$$K_l(\mathbf{X}, \mathbf{X}') = \{c_l(X_i, X'_j)\}_{i=1\dots|\mathbf{X}|, j=1\dots|\mathbf{X}'|},$$

where $|\mathbf{X}|$ is a number of rows in the matrix $\mathbf{X}$. Similarly, $K_d(\mathbf{X}, \mathbf{X}')$ is a matrix of pairwise covariances

$$K_d(\mathbf{X}, \mathbf{X}') = \{c_d(X_i, X'_j)\}_{i=1\dots|\mathbf{X}|, j=1\dots|\mathbf{X}'|},$$

where $|\mathbf{X}|$ is a number of rows in the matrix $\mathbf{X}$. So, A.3 is a posteriori mean value which we use for prediction.

A posteriori (with respect to the given training sample) variance function for some test point X takes the form

$$\mathbb{V}\big[\hat{f}_h(X)\big] = \rho^2\mu_l(k_l(X, X) + \sigma_l^2) + \mu_d(k_d(X, X) + \sigma_d^2) - k(X)^T K^{-1} k(X). \tag{A.5}$$

Squared root of a posteriori variance is used as an accuracy evaluation of the prediction, given by a posteriori mean value.

In matrix notation equations for expectation of gaussian process are

$$\hat{\mathbf{f}}_h(\mathbf{X}^*) = K(\mathbf{X}^*, \mathbf{X})K^{-1}\mathbf{Y}, \tag{A.6}$$

where $K(\mathbf{X}^*, \mathbf{X})$ is matrix of covariances between points from samples $\mathbf{X}^*$ and $\mathbf{X}$ and is concatenation of N^* vectors (A.4):

$$K(\mathbf{X}^*, \mathbf{X}) = \big(k(X_1^*), \quad k(X_2^*), \quad \cdots, \quad k(X_{N^*}^*)\big)\,.$$

Variance takes the form:

$$\mathbb{V}\big[\hat{\mathbf{f}}_h(\mathbf{X}^*)\big] = \rho^2\mu_l(k_l(X,X) + \sigma_l^2) + \mu_d(k_d(X,X) + \sigma_d^2) - \operatorname{diag}\big(K(\mathbf{X}^*)^T K^{-1} K(\mathbf{X}^*)\big)\,, \quad (A.7)$$

where $k_l(X,X)$, $k_d(X,X)$ don't depend on X for used covariance functions, diag is a diagonal elements vector of a matrix.

When processing real data, parameters of covariance function are not known, so special tuning algorithm, described in the next section, was elaborated for their estimation.

A.2 Tuning of covariance function parameters

Parameters a of covariance function (A.1) or (A.2) are estimated using the training sample by maximizing the logarithm of corresponding likelihood, which takes the form [4, 19, 21]:

$$\log\ p(\mathbf{Y}|\mathbf{X}, a, \mu, \tilde{\sigma}) = -\frac{1}{2\mu}\mathbf{Y}^{\mathrm{T}}K^{-1}\mathbf{Y} - \frac{1}{2}\log|K| - \frac{|\mathbf{Y}|}{2}(\log\mu + \log 2\pi),$$

where $|K|$ is a determinant of covariance matrix K for the sample considered, $|\mathbf{Y}|$ is a number of rows in $\mathbf{Y}$, μ is corresponding magnitude. Besides parameters a of covariance function noise variance σ_l^2, σ_d^2 is estimated as well by the solution of the following optimization problem

$$\log\ p(\mathbf{Y}|\mathbf{X}, a, \sigma) \to \max_{a,\sigma}\,. \quad (A.8)$$

For obtaining robust and well-conditioned solution of the optimization problem (A.8) special bayesian regularization algorithm was elaborated [19].

To estimate hyperparameters of covariance function $\operatorname{cov}(Y^l(X), Y^l(X'))$ we solve (A.8) for sample $(\mathbf{X}_l, \mathbf{Y}_l)$. To estimate hyperparameters of covariance function $\operatorname{cov}(Y^d(X), Y^d(X'))$ we use the following algorithm:

- Get values of difference between high and low fidelity models: $Y^d(X) = Y^h(X) - \rho Y^l(X)$ for some initial value of ρ.
- Calculate likelihood for the data $\mathbf{Y}_d$, using hyperparameters of covariance function $\operatorname{cov}(Y^d(X), Y^d(X'))$.
- Get derivatives with respect to hyperparameters and parameter ρ.
- Make new step of the optimization algorithm using new values for hyperparameters of covariance function $\operatorname{cov}(Y^d(X), Y^d(X'))$ and ρ, etc.

A.3 Sparse gaussian processes

For sparse variable fidelity gaussian process it is supposed that the size of low fidelity N_l or high fidelity data N_h is large and inversion of covariance matrix K is impossible or can take a large amount of time. So, it is proposed to approximate inversion of covariance matrix K.

Suppose, that small enough subsamples $D_l^1 = (\mathbf{Y}_l^1, \mathbf{X}_l^1) = \{(f_l(X_i^{l1}), X_i^{l1})\}_{i=1}^{N_l^1}$, $D_h^1 = (\mathbf{Y}_h^1, \mathbf{X}_h^1) = \{(f_h(X_i^{h1}), X_i^{h1})\}_{i=1}^{N_h^1}$ are selected from the full samples D_l, D_h. Sample $\mathbf{X}^1 = \begin{matrix}\mathbf{X}_l^1 \\ \mathbf{X}_h^1\end{matrix}$ We select arbitrary subsets that $N_l^1 \leq 1000, N_h^1 \leq 1000$. For these subsets we apply algorithm (A.2) to get hyperparameters, so we now can calculate $c_l(X_1, X_2)$, $c_h(X_1, X_2)$.

Let's replace matrices $K(\mathbf{X}, \mathbf{X})$, $K(\mathbf{X}^*, \mathbf{X})$, $K(\mathbf{X}^*)$ by their approximations:

$$\begin{aligned} K(\mathbf{X}, \mathbf{X}) &\approx K(\mathbf{X}, \mathbf{X}_1) K(\mathbf{X}_1)^{-1} K(\mathbf{X}_1, \mathbf{X}), \\ K(\mathbf{X}^*, \mathbf{X}) &\approx K(\mathbf{X}^*, \mathbf{X}_1) K(\mathbf{X}_1)^{-1} K(\mathbf{X}_1, \mathbf{X}), \\ K(\mathbf{X}^*) &\approx K(\mathbf{X}^*, \mathbf{X}_1) K(\mathbf{X}_1)^{-1} K(\mathbf{X}_1, \mathbf{X}^*). \end{aligned}$$

where $K(\mathbf{X}^*, \mathbf{X})$ is a matrix of covariances between points from samples $\mathbf{X}^*$ and $\mathbf{X}$ and is concatenation of n^* vectors:

$$K(\mathbf{X}^*, \mathbf{X}) = \begin{pmatrix} k(X_1^*) & k(X_2^*) & \cdots & k(X_{N^*}^*) \end{pmatrix},$$

matrix $K(\mathbf{X}^*)$ is a matrix of pairwise covariances for the test sample. Denote by R matrix of inverse roots of noise variances for each point from learning set:

$$R = \begin{pmatrix} \frac{1}{\sqrt{\mu_l \sigma_l^2}} I_{N_l} & 0 \\ 0 & \frac{1}{\sqrt{\rho^2 \mu_l \sigma_l^2 + \mu_d \sigma_d^2}} I_{N_h} \end{pmatrix}$$

Denote by $\tilde{K}(\mathbf{X}, \mathbf{X}_1)$ the matrix $K(\mathbf{X}, \mathbf{X}_1) R$.

To improve numerical stability and computational time a method from the article [15] was used. This method is based on cholesky decompositions of covariance matrix instead of using of full matrices. We use matrix V_{11} — a cholesky decomposition of matrix $K(\mathbf{X}_1)$, $K(\mathbf{X}_1) = V_{11} V_{11}^T$ and matrix $V = \tilde{K}(\mathbf{X}, \mathbf{X}_1) V_{11}^{-T}$. Sparse approximation of expectation (A.6) is:

$$\begin{aligned} \hat{\mathbf{f}}_h(\mathbf{X}^*) \approx & K(\mathbf{X}^*, \mathbf{X}_1)(K(\mathbf{X}_1) + \tilde{K}(\mathbf{X}, \mathbf{X}_1)^T \tilde{K}(\mathbf{X}, \mathbf{X}_1))^{-1} \tilde{K}(\mathbf{X}, \mathbf{X}_1)^T R \mathbf{Y} = & \text{(A.9)} \\ & K(\mathbf{X}^*, \mathbf{X}_1) V_{11} (I + V^T V)^{-1} V^T \mathbf{Y}. & \text{(A.10)} \end{aligned}$$

Sparse approximation of a variance (A.7) at points $\mathbf{X}^*$ is:

$$\mathbb{V}\big[\hat{\mathbf{f}}_h(\mathbf{X}^*)\big] = \mathrm{diag}(K(\mathbf{X}^*, \mathbf{X}_1) V_{11}^{-T} (I + V^T V)^{-1} V_{11}^{-1} K(\mathbf{X}^*, \mathbf{X}_1)^T) + \rho^2 \mu_l \sigma_l^2 + \mu_d \sigma_d^2.$$

A.4 Black box for low fidelity model

Construction of **GTDF** procedure for **BB DF** doesn't differ from sample based **GTDF** for `VFGP` and `DA` methods. The main difference between **BB DF** and sample based **DF** is that the prediction for **BB DF** is updated using the value of low fidelity model calculated for the given input points. Let us denote by

$$\mathbf{Y}_{exp} = \begin{pmatrix} \mathbf{Y} \\ y_l(X) \end{pmatrix}$$

Then for each output dimension covariance vector takes the form

$$k_l(X) = \begin{pmatrix} c_l(X, \mathbf{X}_l) \\ \rho c_l(X, \mathbf{X}_h) \end{pmatrix}.$$

Self covariance for low fidelity point is

$$c_l(X) = \mu_l(k_l(X, X) + \sigma_l^2).$$

Matrix of covariances for expanded sample is

$$K_{exp} = \begin{pmatrix} K & k_l^T(X) \\ k_l(X) & c_l(X) \end{pmatrix}.$$

Expanded covariance vector is

$$k_{exp}(X) = \begin{pmatrix} k(X) \\ \rho c_l(X, X) \end{pmatrix}.$$

Then we can use equation similar to usual gaussian process equations for a posterior expectation

$$\hat{f}_h(X) = k_{exp}(X) K_{exp}^{-1} \mathbf{Y}_{exp}.$$

and a posteriori variance function

$$\mathbb{V}\big[\hat{f}_h(X)\big] = \rho^2 \mu_l(k_l(X, X) + \sigma_l^2) + \mu_d(k_d(X, X) + \sigma_d^2) - k_{exp}(X)^T K_{exp}^{-1} k_{exp}(X).$$

Fast update (calculation of K_{exp}^{-1}) is possible due to special matrix inversion formulas.

Appendix B

The GTApprox workflow

B.1 Overview

The workflow with the **GTApprox** includes the following elements.

- **Construction of the model.** Construction of the model internally consists of the following steps:

  1. **Preprocessing.** At this step, redundant data are removed from the training set.
  2. **Analysis of training data and options, selection of approximation technique.** At this step, the parameters of the training data and the user–specified options are analyzed for compatibility, and the most appropriate approximation technique is selected.
  3. **Internal Validation (optional).** At this step, Internal Validation of the selected approximation technique is performed on the training data, if the corresponding option is turned on.
  4. **Construction of the main approximation with (optionally) Accuracy Evaluation prediction.** At this step, the main approximation, to be returned to the user, is constructed. Also, the accompanying Accuracy Evaluation prediction is constructed if the corresponding option is turned on.

  After the model has been constructed, the user can view training settings, Internal Validation errors (if available) and export the approximation to a text file.

- **Evaluation of the model.** Evaluation of the constructed model on a point from the design space includes the following steps:

  1. **Smoothing (optional).** The constructed approximation is smoothed by convolution with a smoothing kernel.
  2. **Evaluation of the approximation, gradient and accuracy prediction (if available).** Note: the gradient and the accuracy prediction are computed *after* the approximation has been (optionally) smoothed.

In formal notation for given sample $S = (\mathbf{Y}, \mathbf{X}) = \{(f(X_i), X_i)\}_{i=1}^N$, where $X_i \in \mathbb{R}^{d_{\text{in}}}$, $f(X_i) \in \mathbb{R}^{d_{\text{out}}}$, sample size $N = |S|$, we build an approximation $\hat{f}(X)$ of $f(X)$.

B.2 Data processing

Before applying the approximation technique to the training set, this training set is preprocessed in order to remove possible degeneracies in the data. Let $\mathbf{XY}$ be the $|S| \times (d_{\text{in}} + d_{\text{out}})$ matrix of the training data, where the rows are $(d_{\text{in}} + d_{\text{out}})$-dimensional training points, and the columns are individual scalar components of the input or output. As explained in Section 2, the matrix $\mathbf{XY}$ consists of the sub-matrices $\mathbf{X}$ and $\mathbf{Y}$. We perform the following operation with the matrix $\mathbf{XY}$:

1. We remove all constant columns in the sub-matrix $\mathbf{X}$. A constant column in $\mathbf{X}$ means that all the training vectors have the same value of one of the input components. In particular, this means that the training DoE is degenerate and lies in a proper subspace of the design space. When constructing the approximation, such input components are ignored.

2. We remove repeating rows in the matrix XY. A repeating row means that the same training vector is included more than once in the training matrix. Repetitions bring no additional information and are therefore ignored; a repeating row is counted only once.

If the above operations are nontrivial, e.g., if the matrix $\mathbf{X}$ does contain constant columns or the matrix $\mathbf{XY}$ does contain repeating rows, then the removals are accompanied by warnings.

As a result of these operations, we obtain a reduced matrix $\widetilde{\mathbf{XY}}$ consisting of the submatrices $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{Y}}$. Accordingly, we define *effective input dimension* and the *effective sample size* as the corresponding dimensions of the sub-matrix $\widetilde{\mathbf{X}}$.

Note that after removing repeating rows in $\mathbf{XY}$ the reduced matrix may still contain rows which have the same X components but different Y components. This means that the training data is so noisy that, in general, several different outputs correspond to the same input. Such noisy problems may require a special tuning of **GTApprox**; in particular not all approximation techniques are appropriate for them. If the training data does contain rows with equal X but different Y components, the tool produces a warning.

The model constructed by **GTApprox** is constructed using the reduced matrix $\widetilde{\mathbf{XY}}$ rather than the original matrix $\mathbf{XY}$. Furthermore, it is the effective input dimension and sample size, rather than the original ones, that are used when required at certain steps, in particular when determining the default approximation technique and choosing the default Internal Validation parameters. For example, if the original input dimension has been reduced to 1, then the default 1D technique will be applied to this data by default. The resulting approximation is then considered as a function of the full d_{in}-dimensional input, but it depends only on those components of the full input which have been included in $\widetilde{\mathbf{X}}$.

GTApprox combines a number of approximation techniques of different types. By default, the tool selects the optimal technique compatible with the user–specified options and in agreement with the best practice experience. Alternatively, the user can directly specify the technique through advanced options of the tool. This section describes the available techniques; selection of the technique in a particular problem is described in Section C.

B.3 Individual approximation techniques

The following sections describe general aspects of approximation techniques of **GTApprox**.

B.3.1 Regularized linear regression

Short name: LR

General description: Regularized linear regression. Regularization coefficient is estimated by minimization of generalized cross-validation criterion [16].

Strengths and weaknesses: A very robust and fast technique with a wide applicability in terms of the space dimensions and amount of the training data. It is, however, usually rather crude, and the approximation can hardly be significantly improved by adding new training data.

Restrictions: Interpolation mode is not supported, AE is not supported. Large samples and dimensions are fully supported (up to machine-imposed limits).

Options: No options.

B.3.2 1D Splines with tension

Short name: SPLT

General description: A one–dimensional spline–based technique intended to combine the robustness of linear splines with the smoothness of cubic splines. A non–linear algorithm is used for an adaptive selection of the optimal weights on each interval between neighboring points of DoE. The default implementation in **GTApprox** is already interpolating, so that the **Int** switch has no effect on it. See [17, 22, 23] for details.

Strengths and weaknesses: A very fast technique, combining robustness of linear splines with the accuracy and smoothness of cubic splines. Interpolating. Should not be applied to very noisy problems . On the other hand, performs well if the training data is noiseless. Is the default method for 1D problems in **GTApprox**.

Restrictions: Only for 1D models ($d_{\text{in}} = 1$). Can be used with very large training sets (more than 10000 points).

Options:

- `SPLTContinuity`

  Values: enumeration: `C1`, `C2`

  Default: `C2`

  Short description: required approximation smoothness.

  Description: If `C2` is selected, then the approximation curve will have a continuous second derivative. If `C1` is selected, only the first derivative will be continuous.

B.3.3 High Dimensional Approximation

Short name: HDA

General description: A non–linear, self–organizing technique for construction of approximation using linear decomposition in functions from functional dictionary consisting of both linear and nonlinear base functions. Particular example of such decomposition is so-called two-layer perceptron with nonlinear (sigmoid) activation function. However, the structure and algorithm of HDA is completely different from that of two-layer perceptron and contains the following features:

- an advanced algorithm of division of the training set into the proper training and validating parts;
- different types of base functions, including sigmoid and gaussian functions (see also description of options `HDAFDLinear`, `HDAFDSigmoid` and `HDAFDGauss` below);
- adaptive selection of the type and number of base functions and approximation's complexity (see also description of options `HDAPMin` and `HDAPMax` below);
- an innovative algorithm of initialization of base functions' parameters (see also description of options `HDAInitialization` and `HDARandomSeed` below);
- several different optimization algorithms used to adjust the parameters, including RPROP, Levenberg–Marquardt and their modifications;
- an adaptive strategy controlling the type and parameters of used optimization algorithms (see also description of the option `HDAHPMode` below);
- an adaptive regularization algorithm for controlling the generalization ability of the approximation;
- multi–start capabilities to globalize the optimization of the parameters (see also description of options `HDAMultiMin` and `HDAMultiMax` below);
- an advanced algorithm for boosting used for construction of ensembles for additional improvement of accuracy and stability (see also description of the option `HDAPhaseCount` below);
- post–processing of the results to remove redundant features in the approximation.

In short the HDA algorithm works as follows:

1. The training set is devised into the proper training and validating parts.
2. Functional dictionary with different types of base functions, including sigmoid and gaussian functions, is initialized.
3. The number, type of base functions from the functional dictionary and approximation's complexity are adaptively selected. Thus a basic approximator is initialized.
4. The basic approximator is trained using an adaptive strategy controlling the type and parameters of used optimization algorithms. Specially elaborated adaptive regularization is used to control the generalization ability of the basic approximator.
5. Post–processing of the basic approximator's structure is done in order to remove redundant base functions.

6. An ensemble consisting of such basic approximators is constructed using an advanced algorithm for boosting and multi–start.

See [2] for details.

Strengths and weaknesses: HDA is a flexible nonlinear approximation technique, with a wide applicability in terms of space dimensions. HDA can be used with large training sets (more than 1000 points). However, HDA is not well-suited for the case of very small sample sizes. Significant training time is often necessary to obtain accurate approximation for large training samples. HDA can be applied to noisy data. Usually, accuracy of HDA approximations improves as more training data are supplied.

Restrictions: Interpolation mode is not supported, AE is not supported. Large samples and dimensions are supported.

Options:

- `HDAPhaseCount`

  Values: integer [1, 50].

  Default: 10.

  Short description: Maximum number of approximation phases.

  Description: PhaseCount parameter specifies the maximum possible number of approximation phases. The parameter can take positive integer values. Usually for particular problem the number of approximation phases completed by the approximation algorithm is smaller than the maximum possible number of approximation phases (approximation algorithm stops performing approximation phases as soon as the subsequent approximation phase do not increase the accuracy). In general the more approximation phases we have the more accurate approximator we get. However increase of the maximum possible number of approximation phases can lead to significant increase of training time or/and to overtraining in some cases.

- `HDAPMin`

  Values: integer [0, HDAPMax].

  Default: 0.

  Short description: Minimum admissible complexity.

  Description: Parameter specifies minimal admissible complexity of the approximator. This parameter can take non-negative integer values and must be less or equal to the value of the parameter HDAPMax. The approximation algorithm selects the approximator with optimal complexity pOpt from the range [HDAPMin, HDAPMax]. Optimality here means that depending on the complexity of the behavior of approximable function and the size of the available training sample the constructed approximator with complexity pOpt fits this function in the best possible way compared to approximators with other complexities from the range [HDAPMin, HDAPMax]. Thus the parameter HDAPMin should not be too big in order to select the approximator with complexity being the most appropriate for the considered problem. In general increase of the parameter HDAPMin can lead to significant increase of training time or/and to overtraining in some cases.

- `HDAPMax`

  Values: integer [HDAPMin, 5000] non-negative integer, greater than or equal to HDAPMin.

  Default: 150.

  Short description: Maximum admissible complexity.

  Description: Parameter specifies maximal admissible complexity of the approximator. This parameter can take non-negative integer values and must be greater or equal to the value of the parameter HDAPMin. The approximation algorithm selects the approximator with optimal complexity pOpt from the range [HDAPMin, HDAPMax]. Optimality here means that depending on the complexity of the behavior of approximable function and the size of the available training sample the constructed approximator with complexity pOpt fits this function in the best possible way compared to approximators with other complexities from the range [HDAPMin, HDAPMax]. Thus the parameter HDAPMax should be big enough in order to select the approximator with complexity being the most appropriate for the considered problem. In general increase of the parameter HDAPMax can lead to significant increase of training time or/and to overtraining in some cases.

- `HDAMultiMin`

  Values: integer [1, HDAMultiMax]

  Default: 5.

  Short description: Minimal number of basic approximators constructed during one approximation phase.

  Description: Parameter specifies minimal number of basic approximators constructed during one approximation phase. The parameter can take positive integer values and must be less or equal to the value of the parameter HDAMultiMax. In general the bigger the value of HDAMultiMin the more accurate approximator we get. However increase of this parameter can lead to significant increase of training time or/and to overtraining in some cases.

- `HDAMultiMax`

  Values: integer [HDAMultiMin, 1000].

  Default: 10.

  Short description: Maximal number of basic approximators constructed during one approximation phase.

  Description: Parameter specifies maximal number of basic approximators constructed during one approximation phase. The parameter can take positive integer values and is greater or equal to the value of the parameter HDAMultiMin. Usually for particular problem the number of basic approximators constructed by the approximation algorithm is smaller than the maximum possible number HDAMultiMax of basic approximators (approximation algorithm stops constructing basic approximators as soon as the construction of subsequent basic approximator does not increase the accuracy). In general the bigger the value of HDAMultiMax the more accurate approximator we get. However increase of this parameter can lead to significant increase of training time or/and to overtraining in some cases.

- `HDAFDLinear`

  Values: enumeration: No, Ordinary.

  Default: Ordinary.

  Short description: include linear functions into functional dictionary used for construction of approximations.

  Description: In order to construct approximation a linear expansion in functions from special functional dictionary is used. The parameter HDAFDLinear controls whether linear functions should be included into functional dictionary used for construction of approximation or not. In general usage of linear functions as building blocks for construction of approximation can lead to increase in accuracy, especially in the case when the approximable function has significant linear component. However usage of linear functions can lead to increase of training time as well.

- `HDAFDSigmoid`

  Values: enumeration: No, Ordinary

  Default: Ordinary.

  Short description: include sigmoid functions (with adaptation or without adaptation) into functional dictionary used for construction of approximations.

  Description: In order to construct approximation a linear expansion in functions from special functional dictionary is used. The parameter HDAFDSigmoid controls whether sigmoid-like functions should be included into functional dictionary used for construction of approximation or not. In general usage of sigmoid-like functions as building blocks for construction of approximation can lead to increase in accuracy, especially in the case when the approximable function has square-like or discontinuity regions. However usage of sigmoid-like functions can lead to significant increase of training time.

- `HDAFDGauss`

  Values: enumeration: No, Ordinary.

  Default: Ordinary.

  Short description: include Gaussian functions into functional dictionary used for construction of approximations.

  Description: In order to construct approximation a linear expansion in functions from special functional dictionary is used. The parameter HDAFDGauss controls whether Gaussian functions should be included into functional dictionary used for construction of approximation. In general usage of Gaussian functions as building blocks for construction of approximation can lead to significant increase in accuracy, especially in the case when the approximable function is bell-shaped. However usage of Gaussian functions can lead to significant increase of training time.

- `HDAInitialization`

  Values: enumeration: Deterministic, Random

Default: Deterministic

Short description: switch between deterministic and random initialization of parameters of the approximator.

Description: The approximator construction technique implemented in the tool uses a random number generator for initialization of the parameters of approximator, thus the approximators constructed using the same data with different random initializations may be slightly different. If you need the approximators produced in each HDA GT run to be fully identical for the same data with other parameters having fixed values, you must always select the value of parameter HDAInitialization equal to the value Deterministic. Otherwise (if the parameter HDAInitialization is equal to the value Random) random initialization of parameters of approximator is used. Random initialization of parameters of approximator can be used in order to obtain more accurate approximator since for different initializations the accuracies of obtained approximators may be slightly different.

- `HDARandomSeed`

  Values: integer [1, 2147483647].

  Default: 0

  Short description: Seed value for the random number generator used for random initialization of the parameters of approximator.

  Description: In case the parameter HDAInitialization is equal to the value Random then random initialization of the parameters of approximator is used. The value of the parameter HDAInitialization defines the seed value for the random number generator. There are two possibilities: the value of HDAInitialization is initialized randomly (default option) or its value is defined by the User (any positive integer).

- `HDAHessianReduction`

  Values: Real number from the interval [0, 1].

  Default: 0

  Short description: Maximum proportion of data used for evaluation of Hessian matrix.

  Description: The parameter shrinks maximum amount of data points for Hessian estimation (used in high-precision algorithm). If the parameter is equal to 0, the whole points set is used for Hessian estimation, otherwise if parameter belongs to $(0, 1]$ only part (smaller than HDAHessianReduction of whole train points) is used. Reduction is used only in case of samples bigger than 1000 input points (if number of points is smaller than 1000 points, parameter is not taken into account and Hessian is estimated by whole train sample).

  Remark: Whether GTApprox/HDAHessianReduction = 0 or not, high-precision algorithm can be nevertheless automatically disabled. This happens if

  1. $(\dim(X) + 1) \cdot p \geq 4000$, where $\dim(X)$ is the dimension of the input vector X and p is a total number of basis functions.
  2. $\dim(X) \geq 25$, where $\dim(X)$ is the dimension of the input vector X.
  3. there are no sufficient computational resources for the usage of the HP-algorithm.

B.3.4 Gaussian Processes

Short name: GP

General description: A flexible non–linear, non-parametric technique for constructing of approximation by modeling training data sample as a realization of an infinite dimensional Gaussian Distribution fully defined by a mean function and a covariance function [7, 21]. Approximation is based on a posteriori mean (for the given training data) of the considered Gaussian Process with a priori selected covariance function. AE is supported and is based on the a posteriori covariance (for the given training data) of the considered Gaussian Process. GP contains the following features:

- flexible functional class for modeling covariance function (see also description of option `GPPower` below);
- an innovative algorithm of initialization of parameters of the covariance function;
- an adaptive strategy controlling the parameters of used optimization algorithm;
- an adaptive regularization algorithm for controlling the generalization ability of the approximation;
- multi–start capabilities to globalize the optimization of the parameters;
- usage of errorbars (variances of noise in output values at training points) to efficiently work with noisy problems and improve quality of approximation and AE;
- post–processing of the results to remove redundant features in the approximation.

In short the GP algorithm works as follows:

1. Parameters of the covariance function are initialized;
2. Covariance model of the data generation process is identified by maximizing likelihood of the training data.
3. Post–processing of approximator's structure is done.

See [4] for details.

Strengths and weaknesses: GP usually demonstrates accurate behavior in the case of small and medium sample sizes. Both interpolation and approximation modes are supported (interpolation mode should not be applied to noisy problems). AE is supported. GP is designed for modeling of "stationary" (spatially homogeneous) functions, therefore GP is not well-suited for modeling spatially inhomogeneous functions, functions with discontinuities etc. GP is a resource-intensive method in terms of ram capacity, therefore large samples are not supported.

Restrictions: Large dimensions are supported. Large samples are not supported.

Options:

- `GPPower`

  Values: Real number from the interval [1, 2].

  Default: 2.

  Short description: The value of the parameter p in the p-norm wich is used to measure the distance between the input vectors.

  Description: The main component of the Gaussian Process (GP) based regression is the covariance function measuring the similarity between two input points. The covariance between two input uses p-norm of the difference between coordinates of these input points. The case p = 2 corresponds to the usual gaussian covariance function (better suited for modelling of smooth functions) and the case p = 1 corresponds to laplacian covariance function (better suited for modelling of non-smooth functions).

- `GPType`

  Values: enumeration: `Wlp`, `Mahalanobis`

  Default: `Wlp`

  Short description: Type of covariance function

  Description: Type of the covariance function used in the Gaussian process. Two modes are avaliable. `Wlp` refers to the widely-used exponential gaussian covariance function, `Mahalanobis` refers to squared exponential covariance function with Mahalanobis distance. In particular, the `Mahalanobis` mode can be used only with `GPPower` equal to 2.

- `GPLinearTrend`

  Values: boolean.

  Default: off.

  Short description: Use linear trend for approximation based on gaussian processes (GP technique only)

  Description: Allows user to take into account linear behavior of the function being approximated. If the option is on, then covariance function of GP is a linear combination of stationary covariance (defined by the parameter `GPType`) and non-stationary covariance based on linear trend. In general, such composite covariance can lead to increase in accuracy (for functions with “linear behavior”), but also can lead to significant increase of training time (up to three times).

- `GPMeanValue`

  Values: comma-separated list of floating point values, enclosed in square brackets. This list should be empty or the number of its elements should be equal to output dataset’s dimensionality

  Default: [].

  Short description: specifies mean value of model’s output mean values

Description: models output mean values are necessary for construction of GP approximation. Model's output mean values can be defined by the User or can be estimated using the given sample (the bigger and more representative is the sample, the better is the estimate of model's output mean values). Model's output mean values misspecification leads to decrease in aproximation accuracy: the larger the error in output mean values is, the worser is the final approximation model. If `GPMeanValue` = [], then model's output mean values are estimated using the given sample.

B.3.5 High Dimensional Approximation combined with Gaussian Processes

Short name: HDAGP

General description: HDAGP is a flexible nonlinear approximation technique, with a wide applicability in terms of space dimensions. GP usually demonstrates accurate behavior in the case of small and medium sample sizes. However GP is mostly designed for modeling of "stationary" (spatially homogeneous) functions. HDAGP extends the ability of GP to deal with spatially inhomogeneous functions, functions with discontinuities etc. by using the HDA-based non-stationary covariance function.

Strengths and weaknesses: HDAGP usually demonstrates accurate behavior in the case of medium sample sizes. Both interpolation and approximation modes are supported (interpolation mode should not be applied to noisy problems). AE is supported. However HDAGP approximation is the slowest method compared to HDA method and GP method. HDAGP is a resource-intensive method in terms of ram capacity, therefore large samples are not supported.

Restrictions: Large dimensions are supported. Large samples are not supported.

Options: Options of HDAGP consist of both HDA options and GP options. In general HDAGP algorithm contains both HDA features and GP features. In short the HDAGP algorithm works as follows:

1. HDA approximator is constructed and base functions from this approximator are extracted;
2. Non-stationary covariance model of the data generation process is initialized by using the HDA-based non-stationary covariance function.
3. Non-stationary covariance model of the data generation process is identified by maximizing likelihood of the training data.
4. Post–processing of approximator's structure is done.

See [4] for details.

B.3.6 Sparse Gaussian Process

Short name: SGP

General description: SGP is an approximation of GP which allows to construct GP models for larger train sets than the standard GP is able to. The motivation for SGP is both to provide a higher accuracy of the approximation and enable Accuracy Evaluation for larger training sets. Current implementation of the algorithm is based on the Nystroem approximation [21] and V-technique for subset of regressors method [15].

Algorithms properties: SGP provides both approximation and AE for large training set sizes (by default, SGP is used for $|S|$ from 1001 to 9999 if AE is required). All GP features, except interpolation, are supported.

In short, the algorithm of SGP works as follows:

- Choose a subset S' (base points) from the training set S. The size of S' is 1000 by default, but can be changed by user.
- Initialize parameters of the SGP covariance function as parameters of GP trained with S'.
- Calculate the posterior parameters of SGP.

Restrictions: Large training sets are supported. Large dimensions are supported. Interpolation is not supported.

Options (different from GP's, internal):

- `SGPNumberOfBasePoints`

  Values: Positive integer greater than 1.

  Default: 1000.

  Short description: Number of Base Points used to approximate full matrix of covariances between points from the training sample.

  Description: Base Points (subset of regressors) are selected randomly among points from the training sample and used for the reduced rank approximation of the full matrix of covariances between points from the training sample. Reduced rank approximation is done using Nystrom method for selected subset of regressors. Note that if the value of `SGPNumberOfBasePoints` is greater than the dataset size then GP technique will be used.

- `SGPSeedValue`

  Values: integer in $[0, 2147483647]$.

  Default: 15313.

  Short description: Seed value which defines random selection of Base Points used to approximate full matrix of covariances between points from the training sample.

  Description: The value of the parameter `SGPSeedValue` defines the seed value for the random number generator. If this value is zero then random seed will be used. Seed value defines random selection of Base Points used to approximate full matrix of covariances between points from the training sample. Approximation is done using reduced rank approximation based on Nystrom method for selected subset of regressors (selected Base Points).

B.3.7 Response Surface Model

Short name: RSM

General description: RSM is a kind of linear regression model with several approaches to regression coefficients estimation. RSM can be either linear or quadratic with respect to input variables. Also RSM supports categorical input variables.

Strengths and weaknesses: A very robust and fast technique with a wide applicability in terms of the space dimensions and amount of the training data. It is, however, usually rather crude, and the approximation can hardly be significantly improved by adding new training data. In addition, the number of regression terms can increase rapidly with increasing of dimension, if quadratic RSM is used.

Restrictions: Interpolation mode is not supported, Accuracy Evaluation is not supported.

Options:

- `RSMType`

  Values: enumeration: `linear`, `interactions`, `purequadratic`, `quadratic`

  Default: `linear`

  Short description: Specifies type of the Response Surface Model

  Description: `RSMType` specifies what type of Response Surface Model is going to be constructed:

  - `linear` includes constant and linear terms;
  - `interactions` includes constant, linear, and interaction terms;
  - `quadratic` includes constant, linear, interaction, and squared terms;
  - `purequadratic` includes constant, linear, and squared terms.

- `RSMFeatureSelection`

  Values: enumeration: `LS`, `RidgeLS`, `MultipleRidgeLS`, `StepwiseFit`

  Default: `RidgeLS`

  Short description: Specifies the regularization and term selection procedures used for estimation of model coefficients.

  Description: `RSMFeatureSelection` specifies what technique is going to be used for regularization and term selection: `LS` assumes ordinary least squares (no regularization, no term selection); `RidgeLS` assumes least squares with Tikhonov regularization (no term selection); `MultipleRidgeLS` assumes multiple ridge regression that also filters not important terms; `StepwiseFit` assumes ordinary least squares regression with stepwise inclusion/exclusion for term selection.

  Remarks: Note that term here means not one of the input variables but one of the columns in the design matrix that can consist of intercept, input variables with their squares and interaction products between input variables.

- `RSMMapping`

Values: enumeration: `None`, `MapStd`, `MapMinMax`

Default: `MapStd`

Short description: Specifies mapping type for data preprocessing.

Description: Specifies which technique is used for data preprocessing: `None` assumes no data preprocessing; `MapStd` assumes linear mapping of standard deviation for each variable to [-1, 1] range; `MapMinMax` assumes linear mapping of values for each variable into [-1, 1] range.

- `RSMStepwiseFit/inmodel`

  Values: enumeration: `IncludeAll`, `ExcludeAll`

  Default: `ExcludeAll`

  Short description: Specifies starting model for `RSMFeatureSelection` = `StepwiseFit`.

  Description: Specifies what terms are initially included in the model when `RSMFeatureSelecti` equals `StepwiseFit`. There are two cases: `IncludeAll` assumes start with full model (all terms included); `ExcludeAll` assumes none of the terms is included in the starting step. Depending on the terms included in the initial model and the order in which terms are moved in and out, the method may build different models from the same set of potential terms.

- `RSMStepwiseFit/penter`

  Values: float in range (0., `premove`]

  Default: 0.05

  Short description: Specifies p-value of inclusion for `RSMFeatureSelection` = `StepwiseFit`.

  Description: Specifies maximum p-value of F-test for a term to be added into the model. The higher the value the more terms would be in general included into the final model.

- `RSMStepwiseFit/premove`

  Values: float in range [`penter`, 1.)

  Default: 0.10

  Short description: Specifies p-value of exclusion for `RSMFeatureSelection` = `StepwiseFit`.

  Description: Specifies minimum p-value of F-test for a term to be removed from the model. The higher the value the more terms would be in general included into the final model.

- `RSMCategoricalVariables`

  Values: list of 0-based indices of categorical variables

  Default: [ ]

  Short description: List of indices for categorical variables.

  Description: Contains indices of categorical variables in the input matrix $\mathbf{X}_{\text{train}}$.

B.4 Limitations

The common restriction on the minimum size $|S|$ of the training set for the majority of techniques is

$$|S| > 2d_{\text{in}} + 2,$$

where d_{in} is the input dimension of the data. The exceptions are RSM, LR (minimal size $|S| = 1$) and TA techniques. In case TA technique of minimal size depends on factorization but $|S|$ should be at least 2^n where n is number of factors. Also this restriction on the minimum size $|S|$ depends on *Internal Validation* option (the above restrictions are true for `InternalValidation = Off`). If `InternalValidation = On` then sample size $|S|$ should be higher (see details in the technical reference [10]).

As explained in Section B.2, all conditions above refers to the *effective* values, i.e., obtained after preprocessing of the training data. An error with the corresponding error code will be returned if this condition is violated.

The maximum size of the training sample which can be processed by the tool is primarily determined by the user's hardware. Necessary hardware resources depend significantly on the specific technique. See descriptions of individual techniques.

Limitations of different approximation techniques are summarized in Table B.1.

Technique	Compatible with			Performance on very large training sets	Other restrictions
	Lin	Int	AE		
SPLT	No	Yes	Yes		1D only
LR	Yes	No	No		
GP	No	Yes	Yes	limited by memory	
HDA	Yes	No	No	possibly long runtime	
HDAGP	No	Yes	Yes	limited by memory, possibly long runtime	
SGP	No	No	Yes		
RSM	Yes	No	No		

Table B.1: Limitations of approximation techniques

B.5 Example: comparison of the techniques on a 1D problem

In Figure B.1, the five techniques of **GTApprox** are compared on a 1D test problem. The parts a), b), c) of the figure represent the five approximations, their respective derivatives, and Accuracy Evaluation results (where available). AE results depend significantly on the type of approximation.

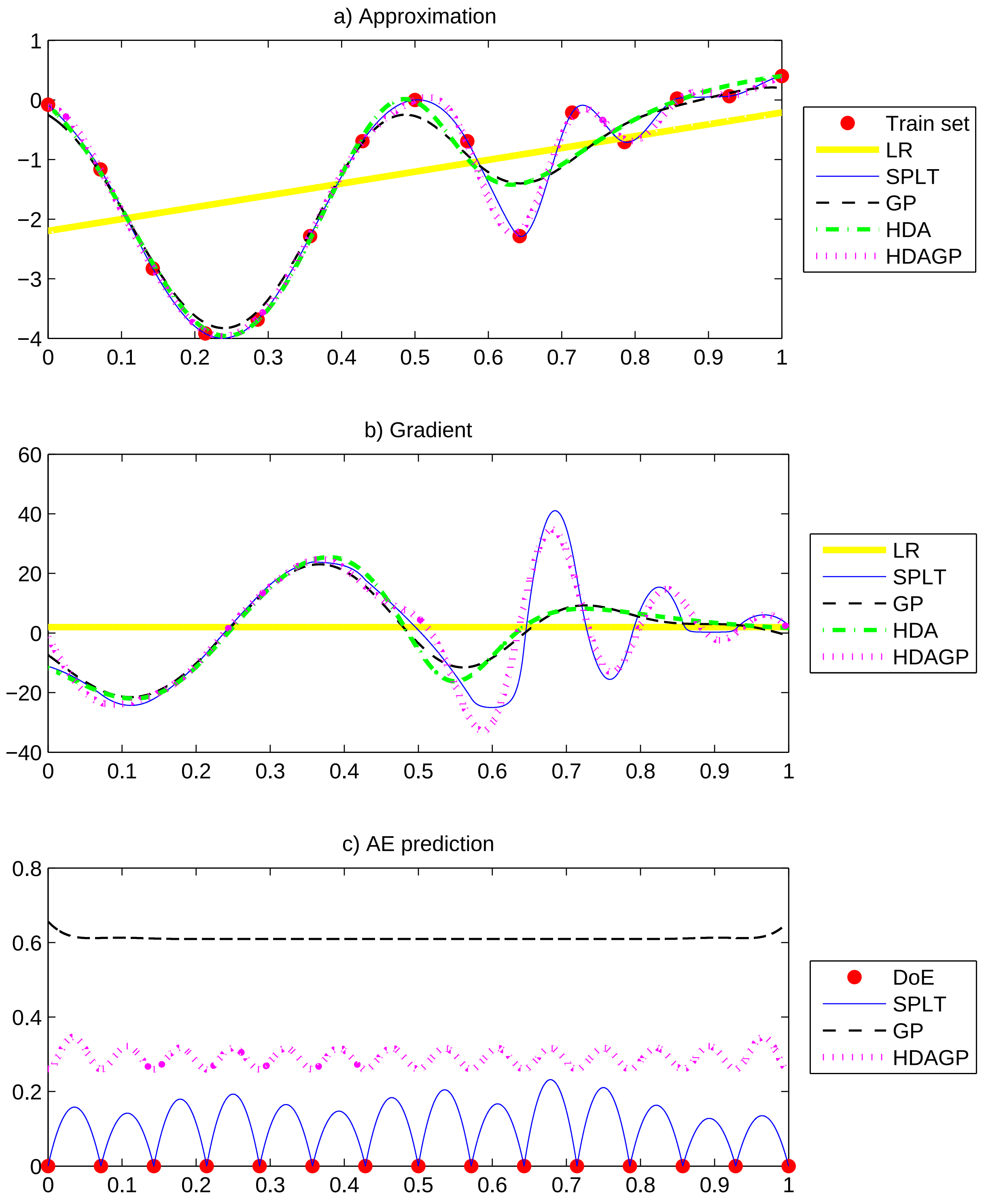


Figure B.1: Comparison of the five approximation methods of **GTApprox** on the same 1D training data.
Note: This image was obtained using an older pSeven Core version. Actual results in the current version may differ.

Appendix C

Selection of the approximation technique and the default decision tree for GTApprox

This section details manual and automatic selection of one of the approximation techniques described in Section B.

C.1 Manual selection

The user may specify the approximation technique by setting the option `Technique` which may have the following values:

- `Auto` — best technique will be determined automatically (default)
- `LR` — Linear Regression
- `SPLT` — Splines with Tension
- `HDA` — High Dimensional Approximation
- `GP` — Gaussian Processes
- `HDAGP` — High Dimensional Approximation + Gaussian Processes
- `SGP` — Sparse Gaussian Processes
- `TA` — Tensor Approximation
- `iTA` — Tensor Approximation for incomplete grids
- `RSM` — Response Surface Model

C.2 Automatic selection

The decision tree describing the default selection of the approximation technique is shown in Figure C.1. The factors of choice are[1]:

[1]Some discussion of why these are essential factors of choice can be found in the development proposal [25].

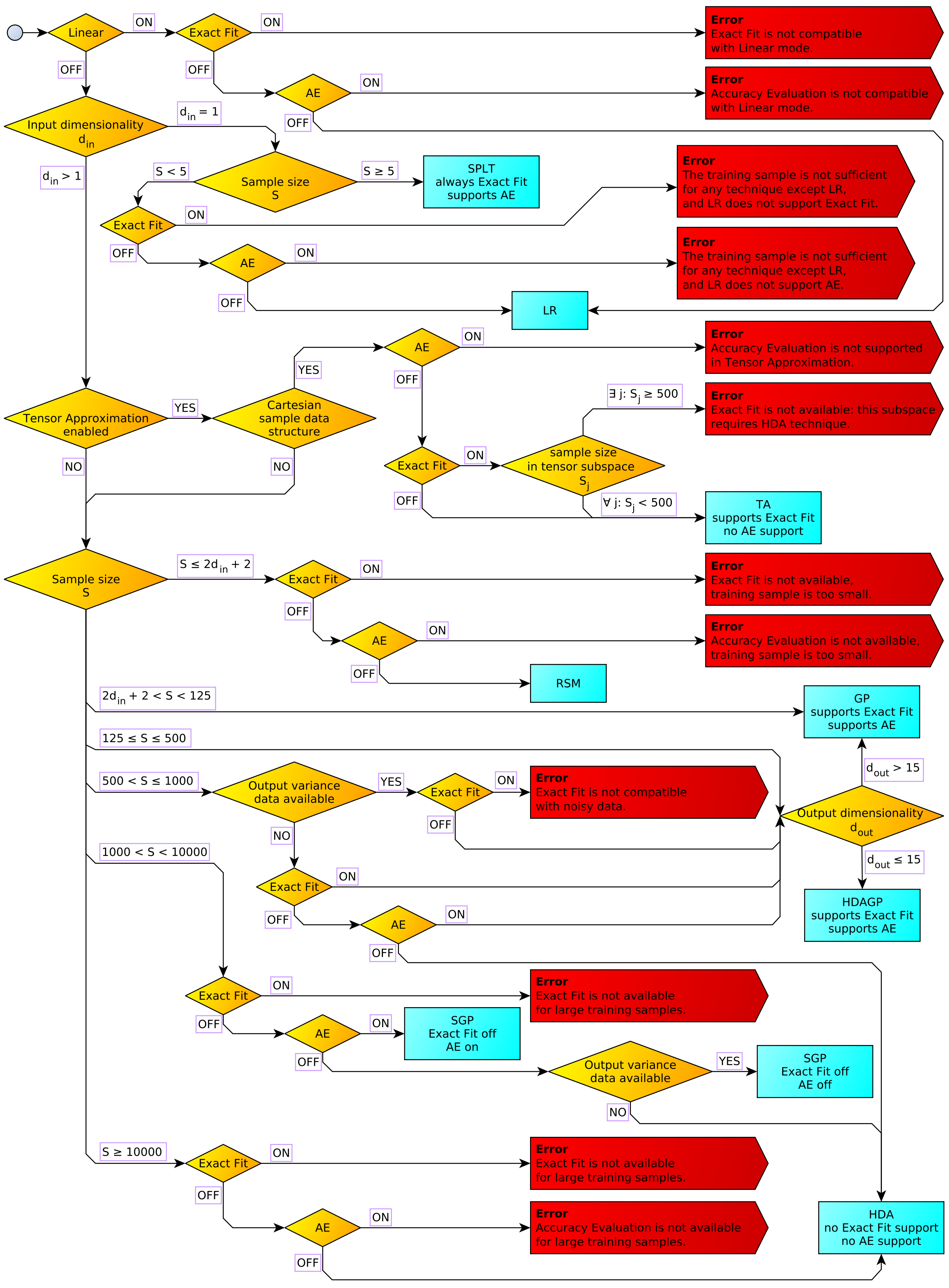

Linear
ON
OFF
Exact Fit
ON
OFF
Error
Exact Fit is not compatible with Linear mode.
AE
ON
OFF
Error
Accuracy Evaluation is not compatible with Linear mode.
Input dimensionality d_{in}
$d_{in} = 1$
$d_{in} > 1$
$S < 5$
Sample size S
$S \geq 5$
SPLT always Exact Fit supports AE
Error
The training sample is not sufficient for any technique except LR, and LR does not support Exact Fit.
Exact Fit
ON
OFF
AE
ON
OFF
Error
The training sample is not sufficient for any technique except LR, and LR does not support AE.
LR
AE
ON
OFF
YES
Error
Accuracy Evaluation is not supported in Tensor Approximation.
Tensor Approximation enabled
YES
NO
Cartesian sample data structure
NO
$\exists$ j: $S_j \geq 500$
Error
Exact Fit is not available: this subspace requires HDA technique.
Exact Fit
ON
OFF
sample size in tensor subspace S_j
$\forall$ j: $S_j < 500$
TA supports Exact Fit no AE support
Sample size S
$S \leq 2d_{in} + 2$
Exact Fit
ON
OFF
Error
Exact Fit is not available, training sample is too small.
AE
ON
OFF
Error
Accuracy Evaluation is not available, training sample is too small.
RSM
$2d_{in} + 2 < S < 125$
GP supports Exact Fit supports AE
$125 \leq S \leq 500$
$d_{out} > 15$
$500 < S \leq 1000$
Output variance data available
YES
Exact Fit
ON
OFF
Error
Exact Fit is not compatible with noisy data.
Output dimensionality d_{out}
NO
$d_{out} \leq 15$
$1000 < S < 10000$
Exact Fit
ON
OFF
HDAGP supports Exact Fit supports AE
AE
ON
OFF
Exact Fit
ON
OFF
Error
Exact Fit is not available for large training samples.
AE
ON
OFF
SGP Exact Fit off AE on
Output variance data available
YES
NO
SGP Exact Fit off AE off
$S \geq 10000$
Exact Fit
ON
OFF
Error
Exact Fit is not available for large training samples.
AE
ON
OFF
Error
Accuracy Evaluation is not available for large training samples.
HDA no Exact Fit support no AE support

Figure C.1: The **GTApprox** internal decision tree for the choice of default approximation methods

- Size $|S|$ and input dimension d_{in} of the training sample;
- The state of the switches **Lin**, **Int** and **AE**.

The result is the constructed approximation, possibly with an accuracy prediction, or an error. The selection is performed in agreement with properties of individual approximation techniques as described in section B.3. In particular:

- The default 1D technique is SPLT having good all–around performance. It is very fast and can efficiently handle both small and very large training sets. As it is interpolating, it should be replaced with a smoothing technique if the training data is very noisy.
- In dimensions 2 and higher, the default techniques are GP, HDAGP and HD, depending on the size $|S|$ of the training set:
  - If $|S| \leq 2d_{\text{in}} + 2$, then the default technique is RSM, which is robust for such small sets.
  - If $2d_{\text{in}} + 2 < |S| < 125$, then the default technique is GP, which is normally efficient for rather small sets.
  - If $125 \leq |S| \leq 500$, then the default technique is HDAGP, combining GP with HDA.
  - If $500 < |S| \leq 1000$, then the technique depends on whether Accuracy Evaluation or Interpolation is required:
    - If both **AE** and **Int** are `off`, then the default technique is HDA;
    - Otherwise, the default technique is HDAGP.
  - If $1000 < |S| < 10000$, then the technique depends on whether Accuracy Evaluation is required:
    - If **AE** is `off`, then the default technique is HDA;
    - Otherwise, the default technique is SGP.
  - If $|S| \geq 10000$, then the default technique is HDA, which can handle the largest sets (note, however, that the Interpolation mode and AE are not available for it).

  The threshold values $2d_{\text{in}} + 2$, 125, 500, 1000 and 10000 for $|S|$ have been set based on previous experience and extensive testing.

In Figure C.2 the "sample size vs. dimension" diagram for the default choice is shown.

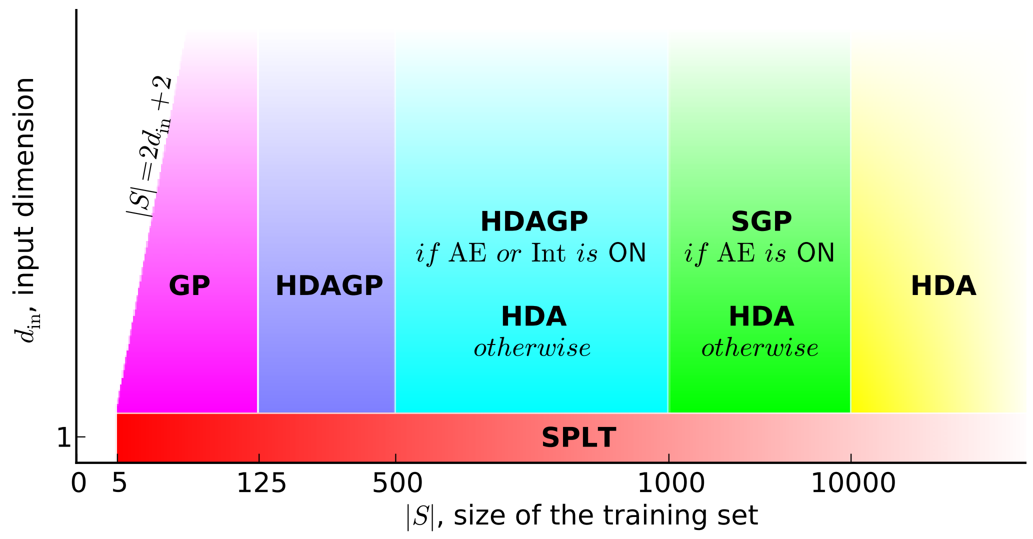


Figure C.2: The “sample size vs. dimension” diagram of default techniques in **GTApprox**

Bibliography

[1] M. Álvarez and N. Lawrence. Computationally efficient convolved multiple output gaussian processes. *Journal of Machine Learning Research*, 12:1425–1466, 2011.

[2] M. Belyaev and E. Burnaev. Approximation of a multidimensional dependency based on a linear expansion in a dictionary of parametric functions. *Informatics and its Applications*, 7(3):1, 2013.

[3] A. Bernstein, E. Burnaev, S. Chernova, F. Zhu, and N. Qin. Comparison of three geometric parameterization methods and their effect on aerodynamic optimization. In *Proceedings of International Conference on Evolutionary and Deterministic Methods for Design,Optimization and Control with Applications to Industrial and Societal Problems (Eurogen 2011)*, 2011.

[4] E. Burnaev, A. Zaytsev, M. Panov, P. Prihodko, and Y. Yanovich. Modeling of non-stationary covariance function of gaussian process using decomposition in dictionary of nonlinear functions. In *To be published in proceeding of the conference "Information Technologies and Systems–2011"*, Moscow, October 2–7 2011.

[5] E. V. Burnaev, M. G. Belyaev, and P. V. Prihodko. About hybrid algorithm for tuning of parameters in approximation based on linear expansions in parametric functions. In *Proceeding of the 8th International Conference "Intelligent Information Processing"*, pages 208–211, Cyprus, 2010.

[6] E. V. Burnaev and Y. A. Yanovich. Sparse gaussian processes. In *Conference MIPT-54*, 2011.

[7] N. A. C. Cressie. *Statistics for Spatial Data.* Wiley, 1993.

[8] Datadvance. *Generic Tool for Approximation: User manual.*

[9] Datadvance. *Generic Tool for Design of Experiments: User manual.*

[10] DATADVANCE, llc. *pSeven Core Generic Tool for Approximation: Technical Reference*, 2011.

[11] K. Dewettinck, A. Visscher, L. Deroo, and A. Huyghebaert. Modeling the steady-state thermodynamic operation point of top-spray fluidized bed processing. *Journal of Food Engineering*, 39(2):131–143, 1999.

[12] B. S. Everitt. *The Cambridge Dictionary of Statistics.* Cambridge University Press, 2002.

[13] A. Forrester, A. Sóbester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. Progress in astronautics and aeronautics. J. Wiley, 2008.

[14] A. I. J. Forrester, A. Sobester, and A. J. Keane. Multi-fidelity optimization via surrogate modelling. *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, 463:3251–3269, 2007.

[15] L. Foster, A. Waagen, and N. Aijaz. Stable and efficient gaussian process calculations. *Journal of machine learning*, 10:857–882, 2009.

[16] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2008.

[17] J. M. Hyman. Accurate monotonicity preserving cubic interpolation. *SIAM J. Sci. Stat. Comput.*, 4(4):645–654, 1983.

[18] I. Jolliffe and MyiLibrary. *Principal component analysis*, volume 2. Wiley Online Library, 2002.

[19] M. E. Panov, E. V. Burnaev, and A. A. Zaytsev. Bayesian regularization for regression based on gaussian processes. In *Proceedings of All-Russian conference "Mathematical methods of pattern recognition (MMPR-15)"*, pages 142–146, Petrozavodsk, Russia, 11–17 September 2011.

[20] P. Qian and C. Wu. Bayesian hierarchical modeling for integrating low-accuracy and high-accuracy experiments. *Technometrics*, 50(2):192–204, 2008.

[21] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[22] R. J. Renka. Interpolatory tension splines with automatic selection of tension factors. *Society for Industrial and Applied Mathematics, Journal for Scientific and Statistical Computing*, 8:393–415, 1987.

[23] P. Rentrop. An algorithm for the computation of the exponential spline. *Numerische Mathematik*, 35:81–93, 1980.

[24] I. V. Tetko, D. J. Livingstone, and A. I. Luik. Neural network studies. 1. comparison of overfitting and overtraining. *J. Chem. Inf. Comput. Sci.*, 35(5):826–833, 1995.

[25] D. Yarotsky. HDA/AE development plan: general functionality and user interface. Technical report, DATADVANCE, 2011.

[26] A. A. Zaytsev, E. V. Burnaev, and E. R. Kapushev. The data fusion problem. In *Conference MIPT-54*, 2011.

Acronyms

DA	Difference approximation. 9
GP	Gaussian Processes. 45
HDA	High Dimensional Approximation. 39
HDAGP	High Dimensional Approximation combined with Gaussian Processes. 47
HFA	High fidelity approximation. 8
LR	Linear Regression. 39
RSM	Response Surface Model. 49
SGP	Sparse Gaussian Processes. 47
SPLT	1D Splines with tension. 39
SVFGP	Sparse variable fidelity gaussian processes. 10
VFGP	Variable fidelity gaussian processes. 10

Index: concepts

Index: GTDF options

Note: these are shortened names of the options. The full names for **GTDF** are obtained by adding `GTDF/` in front of the shortened name, e.g. `GTDF/Technique`.

Index: GTApprox options

Note: these are shortened names of the options. The full names for **GTApprox** are obtained by adding `GTApprox/` in front of the shortened name, e.g. `GTApprox/Technique`.