Continuous Optimization Benchmarks by Simulation*

Martin Zaefferer and Frederik Rehbach

Institute for Data Science, Engineering, and Analytics,
TH Köln, 51643 Gummersbach, Germany
{martin.zaefferer,frederik.rehbach}@th-koeln.de

Abstract. Benchmark experiments are required to test, compare, tune, and understand optimization algorithms. Ideally, benchmark problems closely reflect real-world problem behavior. Yet, real-world problems are not always readily available for benchmarking. For example, evaluation costs may be too high, or resources are unavailable (e.g., software or equipment). As a solution, data from previous evaluations can be used to train surrogate models which are then used for benchmarking. The goal is to generate test functions on which the performance of an algorithm is similar to that on the real-world objective function. However, predictions from data-driven models tend to be smoother than the ground-truth from which the training data is derived. This is especially problematic when the training data becomes sparse. The resulting benchmarks may not reflect the landscape features of the ground-truth, are too easy, and may lead to biased conclusions.
To resolve this, we use simulation of Gaussian processes instead of estimation (or prediction). This retains the covariance properties estimated during model training. While previous research suggested a decomposition-based approach for a small-scale, discrete problem, we show that the spectral simulation method enables simulation for continuous optimization problems. In a set of experiments[1] with an artificial ground-truth, we demonstrate that this yields more accurate benchmarks than simply predicting with the Gaussian process model.

Keywords: Simulation, Benchmarking, Test Function, Continuous Optimization, Gaussian Process Regression, Kriging

1 Introduction

For the design and development of optimization algorithms, benchmarks are indispensable. Benchmarks are required to test hypotheses about algorithm behavior, to understand the impact of algorithm parameters, to tune those parameters,

* The final authenticated version is available online at https://doi.org/10.1007/978-3-030-58112-1_19

[1] Reproducible code and a complete set of the presented figures is provided at https://github.com/martinzaefferer/zaef20b. For easily accessible interfaces and demonstrations see https://github.com/martinzaefferer/COBBS.

or to compare algorithms with each other. Multiple benchmarking frameworks exist, with BBOB/COCO being a prominent example [11,12].

One issue of benchmarks is their relevance to real-world problems. The employed test functions may be of an artificial nature, yet should reflect the behavior of algorithms on real-world problems. An algorithm's performance on test functions and real-world problems should be similar. Yet, real-world problems may not be available in terms of functions, but only as data (i.e., observations from previous experiments). This can be due to real objective function evaluations being too costly or not accessible (in terms of software or equipment).

In those cases, using a data-driven approach may be a viable alternative: surrogate models can be trained and subsequently used to benchmark algorithms. The intent is not to replace artificial benchmarks such as BBOB (which have their own advantages), but rather to augment them with problems that have a closer connection to real-world problems. This approach has been considered in previous investigations [18,17,2,8,7,5]. Additionally, recent benchmark suites offer access to real-world problems, e.g., the Computational Fluid Dynamics (CFD) test problem suite [6] and the Games Benchmark for Evolutionary Algorithms (GBEA) [20]. Notably, the authors of the GBEA accept data provided by other researchers as a basis for surrogate model-based benchmarking[2].

As pointed out by Zaefferer et al. [23], surrogate model-based benchmarks face a crucial issue: the employed machine learning models may smoothen the training data, especially if the training data is sparse. Hence, these models are prone to produce optimization problems that lack the ruggedness and difficulty of the underlying real-world problems. Thus, algorithm performances may be overrated, and comparisons become biased. Focusing on a discrete optimization problem from the field of computational biology, Zaefferer et al. proposed to address this issue via simulation with Gaussian Process Regression (GPR). In contrast to estimation (or prediction) with GPR, simulation may provide a more realistic assessment of an algorithm's behavior. The response of the simulation retains the covariance properties determined by the model [15].

The decomposition-based simulation approach used by Zaefferer et al. relies on the selection of a set of simulation samples [23]. The simulation is evaluated at these sample locations. The simulation samples are distinct from and less sparse than the observed training samples. They are not restricted by evaluation costs. Still, using a very large number of simulation samples can quickly become computationally infeasible. In small discrete search spaces, all samples in the search space can be simulated. In larger search spaces, the simulation has to be interpolated between the simulation samples. The interpolation step might again introduce undesirable smoothness. Thus, decomposition-based simulation may work well for (small-scale) combinatorial optimization problems. Conversely, it is not suited for continuous benchmarks. Hence, our research questions are:

Q1 How can simulation with GPR models be used to generate benchmarks for continuous optimization?

[2] See the GBEA website, at `http://www.gm.fh-koeln.de/~naujoks/gbea/gamesbench_doc.html#subdata`. Accessed on 2020-08-03.

Q2 Do simulation-based benchmarks provide better results than estimation-based benchmarks, for continuous optimization?

For Q1, we investigate the spectral method for GPR-simulation [4]. The required background on GPR, estimation, and simulation is given in section 2. Then, we describe a benchmark experiment to answer Q2 in section 3, and the results in section 4. The employed code is made available. We discuss critical issues of GPR and simulation in section 5. Section 6 concludes the paper with a summary and outlook.

2 Gaussian Processes Regression

In the following, we assume that we deal with an objective function $\mathrm{f}(\mathbf{x})$, which is expensive to evaluate or has otherwise limited availability. Here, $\mathbf{x} \in \mathbb{R}^n$ are the variables of the optimization problem. Respectively, we have to learn models that regress data sets with m training samples $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, and the corresponding observations $\mathbf{y} \in \mathbb{R}^m$, with $y_j = \mathrm{f}(\mathbf{x}_j)$, and $j = 1, \ldots, m$.

2.1 Gaussian Process Regression

GPR (also known as Kriging) assumes that the training data $\mathbf{X}$, $\mathbf{y}$ is sampled from a stochastic process of Gaussian distribution. Internally, it interprets data based on their correlations. These correlations are determined by a kernel $\mathrm{k}(\mathbf{x}, \mathbf{x}')$. A frequently chosen kernel is

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \exp\left(\sum_{i=1}^{n} -\theta_i |x_i - x_i'|^2\right). \tag{1}$$

Here, $\theta_i \in \mathbb{R}$ is a parameter that is usually determined by Maximum Likelihood Estimation (MLE). In the following, we assume that the model has already been trained via MLE, based on the data $\mathbf{X}$, $\mathbf{y}$. The kernel $\mathrm{k}(\mathbf{x}, \mathbf{x}')$ yields the correlation matrix $\mathbf{K}$, which collects all pairwise correlations of the training samples $\mathbf{X}$. The vector of correlations between each training sample $\mathbf{x}_j$, and a single, new sample $\mathbf{x}$ is denoted by $\mathbf{k}$. Further details on GPR, including model training by MLE, are given by Forrester et al. [9].

In the context of GPR, the term *estimation* denotes the prediction of the model at some unknown, new location. It is performed with the predictor

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \mathbf{k}^T \mathbf{K}^{-1} (\mathbf{y} - \mathbf{1}\hat{\mu}). \tag{2}$$

Here, $\mathbf{1}$ is a vector of ones and the parameter $\hat{\mu}$ is determined by MLE. Estimation intends to give an accurate response value at a single location $\mathbf{x}$.

2.2 Simulation by Decomposition

Conversely to estimation, *simulation* intends to reproduce the covariance structure of a set of samples as accurately as possible [14,4]. Intuitively, this is exactly what we require for the generation of optimization benchmarks: We are interested in the topology of the landscape (here: captured by the covariance structure), rather than accurate predictions of isolated function values [17].

One approach towards simulation is based on the decomposition of a covariance matrix $\mathbf{C}_\mathrm{s}$ [4]. This matrix is computed for a set of n_sim simulation samples $\mathbf{X}_\mathrm{s} = \{\mathbf{x}_1, ..., \mathbf{x}_{n_\mathrm{sim}}\}$, with $\mathbf{x}_t \in \mathbb{R}^n$ and $t = 1, ..., n_\mathrm{sim}$. Here, n_sim is usually much larger than the number of training samples m. Using eq. (1), $\mathbf{X}_\mathrm{s}$ yields the correlation matrix $\mathbf{K}_\mathrm{s}$ of all simulation samples, and the respective covariance matrix is $\mathbf{C}_\mathrm{s} = \hat{\sigma}^2\mathbf{K}_\mathrm{s}$. Here, $\hat{\sigma}^2$ is a model parameter (determined by MLE). Decomposition can, e.g., be performed with the Cholesky decomposition $\mathbf{C}_\mathrm{s} = \mathbf{L}\mathbf{L}^T$. This yields the *unconditionally* simulated values $\hat{\mathbf{y}}_\mathrm{s} = \mathbf{1}\hat{\mu} + \mathbf{L}\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is a vector of independent normal-distributed random samples, $\epsilon_i \sim N(0, 1)$. In this context, 'unconditional' means that the simulation reproduces only the covariance structure, but not the observed values $\mathbf{y}$. Additional steps are required for conditioning, so that the observed values are reproduced, too [4].

Obviously, the simulation only produces a discrete number of values $\hat{\mathbf{y}}_\mathrm{s}$, at specific locations $\mathbf{X}_\mathrm{s}$. Initially, we do not know the locations where our optimization algorithms will attempt to evaluate the test function. Hence, subsequent evaluations at arbitrary locations rely on interpolation. The predictor from eq. (2) can be used, replacing all values linked to the training data with the respective values from the simulation ($\mathbf{X}_\mathrm{s}$, $\hat{\mathbf{y}}_\mathrm{s}$, $\mathbf{K}_\mathrm{s}$ instead of $\mathbf{X}$, $\mathbf{y}$, $\mathbf{K}$). The model parameters $\hat{\sigma}^2$, $\hat{\mu}$, and θ_i remain unchanged.

Unfortunately, this interpolation step is a critical weakness when applied to continuous optimization problems. The locations $\mathbf{X}_\mathrm{s}$ have to be sufficiently dense, to avoid that the interpolation introduces undesirable smoothness. Yet, computational restrictions limit the density of $\mathbf{X}_\mathrm{s}$. Even a rather sparse grid of 20 samples in each dimension requires $m = 20^n$ simulated samples. Then, $\mathbf{C}_\mathrm{s}$ is of dimension $20^n \times 20^n$, which is prohibitively large in terms of memory consumption for $n \geq 4$. Even a mildly multimodal function may easily require a much denser sample grid. This renders the approach infeasible for continuous optimization problems with anything but the lowest dimensionalities.

2.3 Simulation by the Spectral Method

Following up on [23], we investigate a different simulation approach that is well suited for continuous optimization problems: the spectral method [4]. This approach directly generates a function that can be evaluated at arbitrary locations, without interpolation. It yields a superposition of cosine functions [4],

$$\mathrm{f}_\mathrm{s}(\mathbf{x}) = \hat{\sigma}\sqrt{\frac{2}{N}}\sum_{v=1}^{N}\cos(\boldsymbol{\omega}_v \cdot \mathbf{x} + \phi_v),$$

with ϕ_v being an i.i.d. uniform random sample from the interval $[-\pi, \pi]$. The sampling of $\boldsymbol{\omega}_v$ requires the spectral density function of the GPR model's kernel [4,15]. That is, $\boldsymbol{\omega}_v \in \mathbb{R}^n$ are i.i.d. random samples from a distribution with that same density. For the kernel from eq. (1), the respective distribution for the i-th dimension is the normal distribution with zero mean and variance $2\theta_i$. A simulation conditioned on the training data can be generated with $\mathrm{f}_\mathrm{sc}(\mathbf{x}) = \mathrm{f}_\mathrm{s}(\mathbf{x}) + \hat{y}^*(\mathbf{x})$ [4], where $\hat{y}^*(\mathbf{x}) = \hat{\mu} + \mathbf{k}^T\mathbf{K}^{-1}(\mathbf{y}_{sc} - \mathbf{1}\hat{\mu})$ is the predictor from eq. (2) with the training observations $\mathbf{y}$ replaced by $\mathbf{y}_{sc}$, and $\hat{\mu} = 0$. Here,

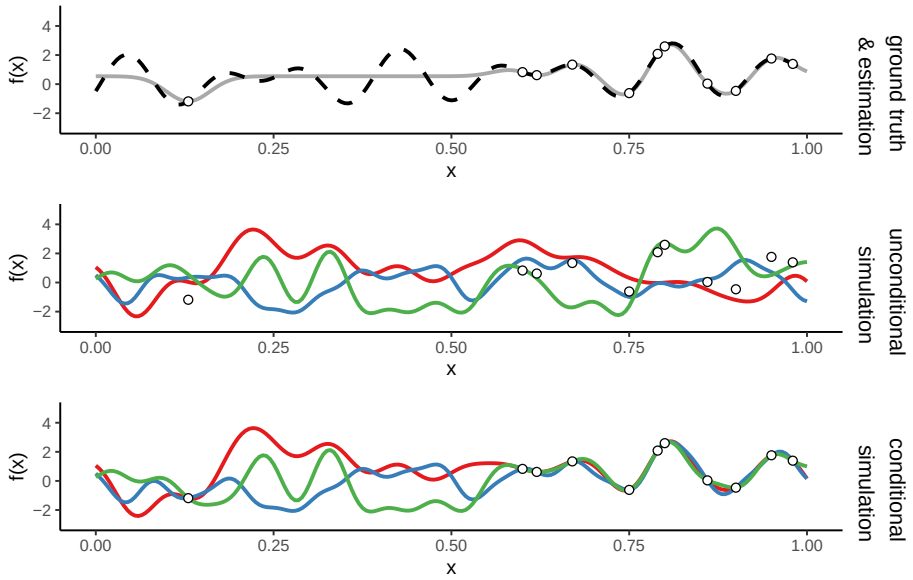

Fig. 1. Top: Ground-truth $\mathrm{f}(x)$ (dashed line), training data (circles), and GPR model estimation (gray solid line). Middle: Three instances of an unconditional simulation (same model). Bottom: Three instances of a conditional simulation (same model). The three different instances are generated by re-sampling of $\boldsymbol{\omega}_v$ and ϕ_v.

$\mathbf{y}_{sc}$ are the unconditionally simulated values at the training samples, that is, $y_{\mathrm{sc}_j} = \mathrm{f_s}(\mathbf{x}_j)$.

2.4 Simulation for Benchmarking

To employ these simulations in a benchmarking context, we roughly follow the approach by Zaefferer et al. [23]. First, a data set is created by evaluating the true underlying problem (if not already available in the form of historical data). Then, a GPR model is trained with that data. Afterwards, the spectral method is used to generate conditional or unconditional simulations. These simulations are finally used as test functions for optimization algorithms.

Here, the advantage of simulation over estimation is the ability to reproduce the topology of functions, rather than predicting a single, isolated value. As an illustration, let us assume an example for $n = 1$, where the ground-truth is $\mathrm{f}(x) = \sin(33x) + \sin(49x - 0.5) + x$. A GPR model is trained with the samples $\mathbf{X} = \{0.13, 0.6, 0.62, 0.67, 0.75, 0.79, 0.8, 0.86, 0.9, 0.95, 0.98\}$. The resulting estimation, unconditional simulation, and conditional simulation of the GPR model are presented in fig. 1. This example shows how estimation might be unsuited to reproduce an optimization algorithm's behavior. In the sparsely sampled region, the GPR estimation is close to constant, considerably reducing the number of local optima in the (estimated) search landscape. The number of optima in the simulated search landscapes is considerably larger.

3 Experimental Setup

In the following, we describe an experiment that compares test functions produced by estimation and simulation with GPR.

3.1 Selecting the Ground-Truth

A set of objective functions is required as a ground-truth for our experiments. In practice, the ground-truth would be a real-world optimization problem. Yet, a real-world case would limit the comparability, understandability, and the extent of the experimental investigation. We want to understand where and why our emulation deviates from the ground-truth. This situation reflects the need for model-based benchmarks.

Hence, we chose a well-established artificial benchmark suite for optimization: the single-objective, noiseless BBOB suite from the COCO framework [11,12]. The BBOB suite allows us to compare in-detail how algorithms behave on the actual problem (ground-truth) and how they behave on an estimation or simulation with GPR. Moreover, important landscape features of the BBOB suite are known (e.g., modality, symmetry/periodicity, separability), which enables us to understand and explain where GPR models fail.

The function set that we investigated is described in [13]. This set consists of 24 unimodal and multimodal functions. For each function, 15 randomized instances are usually produced. We followed the same convention. In addition, all test functions are scalable in terms of search space dimensionality n. We performed our experiments with $n = 2, 3, 5, 10, 20$.

3.2 Generating the Training Data

We generated training data by running an optimization algorithm on the original problem. The data observed during the optimization run was used to train the GPR model. This imitates a common scenario that occurs in real-world applications: Some algorithm has already been run on the real-world problem, and the data from that preliminary experiment provides the basis for benchmarking. Moreover, this approach allows us to determine the behavior of the problem on a local and global scale. An optimization algorithm (especially, a population-based evolutionary algorithm) will explore the objective function globally as well as performing smaller, local search steps.

Specifically, we generated our training data as follows: For each BBOB function $(1, ..., 24)$ and each function instance $(1, ..., 15)$, we ran a variant of Differential Evolution (DE) [19] with $50n$ function evaluations, and a population size of $20n$. All evaluations were recorded. We used the implementation from the DEoptim R-package, with default configuration [1]. This choice is arbitrary. Other population-based algorithms would be equally suited for our purposes.

3.3 Generating the Model

We selected the $50n$ data samples provided by the DE runs. Based on that data, we trained a GPR model, using the SPOT R-package [3]. Three non-default parameters of the model were specified with: `useLambda=FALSE` (no nugget effect, see also [9]), `thetaLower=1e-6` (lower bound on θ_i), and `thetaUpper=1e12`

(upper bound on θ_i). For the spectral simulation, we used $N = 100n$ cosine functions. We only created conditional simulations, to reflect each BBOB instance as closely as possible. Scenarios where an unconditional simulation is preferable have been discussed by Zaefferer et al. [23].

3.4 Testing the Algorithms

We tested three algorithms:

- DE: As a global search strategy, we selected DE. We tested the same DE variant as mentioned in section 3.2, but with a population size of $10n$ and a different random number generator seed for initialization. All other algorithm parameters remained at default values.
- NM: As a classical local search strategy, the Nelder-Mead (NM) simplex algorithm was selected [16]. We employed the implementation from the R-package nloptr [22]. All algorithm parameters remained at default values.
- RS: We also selected a Random Search (RS) algorithm, which evaluates the objective function with i.i.d. samples from a uniform random distribution.

This selection was to some extent arbitrary. The intent was not to investigate these specific algorithms. Rather, we selected these algorithms to observe a range of different convergence behaviors. Essentially, we made a selection that scales from very explorative (RS), to balanced exploration/exploitation (DE), to very exploitative (NM).

All three algorithms receive the same test instances and initial random number generator seeds. For each test instance, each algorithm uses $1000n$ function evaluations. Overall, each algorithm was run on each instance, function, and dimension of the BBOB test suite ($24 \times 15 \times 5 = 1800$ runs, each run with $1000n$ evaluations). Additionally, each of these runs was repeated with an estimation-based test function, and with a simulation-based test function.

4 Results

4.1 Quality Measure

Our aim is to measure how well algorithm performance is reproduced by the test functions. One option would be to measure the error as the difference of observed values along the search path compared to the ground-truth values at those same locations. But this is problematic. Let us assume that the ground truth is $\mathrm{f}(x) = x^2$, and two test functions are $\mathrm{f}_{\mathrm{t}1}(x) = (x-1)^2$ and $\mathrm{f}_{\mathrm{t}2}(x) = 0.5$, with $x \in [0, 1]$. Clearly, $\mathrm{f}_{\mathrm{t}1}$ is a reasonable oracle for most algorithms' performance (e.g., in terms of convergence speed) while $\mathrm{f}_{\mathrm{t}2}$ is not. Yet, the mean squared error of $\mathrm{f}_{\mathrm{t}1}$ would usually be larger than the error of $\mathrm{f}_{\mathrm{t}2}$. The error on $\mathrm{f}_{\mathrm{t}1}$ even increases when an algorithm approaches the optimum.

Hence, we measured the error on the performance curves. For each algorithm run, the best observed function values after each objective function evaluation were recorded (on all test instances, including estimation, simulation, and ground-truth). In the following, this will be referred to as the performance of the algorithm. The resulting performance values were scaled to values between

zero and one, for each problem instance (i.e., each BBOB function, each BBOB instance, each dimension, and also separately for the ground-truth, estimation, and simulation variants). This yielded what we term the scaled performance. The error of the scaled performance was then calculated as the absolute deviation of the performance on the model-based functions, compared to the performance on the ground-truth problem. For example, let us assume that DE achieved a (scaled) function value on the ground-truth of 0.25 after 200 objective function evaluations. But the same algorithm only achieved 0.34 on the estimation-based test function after 200 evaluations. Then, the error of the estimation-based run is $|0.34 - 0.25| = 0.09$ (after 200 evaluations).

4.2 Observations

In figs. 2 and 3, we show the resulting errors over run time for a subset of the 24 BBOB functions. Due to space restrictions, we only show the error for the DE and NM algorithms. Similar patterns are visible in the omitted curves for RS. We also omit the curves for $n = 3$, which closely mirror those for $n = 2$.

For the simulation, we mostly observe decreasing errors or constant errors over time. The decrease can be explained by the algorithms' tendency to find the best values for the respective problem instance later on in the run, regardless of the objective function. Earlier, the difference is usually larger. For the estimation-based test functions, the error often increases, and sometimes decreases again during the later stages of a run.

When comparing estimation and simulation, the modality and the dimensionality n are important. For low-dimensional unimodal BBOB functions ($n = 2, 3, 5$, and function IDs: 1, 2, 5-7, 10-14), the simulation yields larger errors than estimation. In most of the multimodal cases, the simulation seems to perform equally well or better. This can be explained: The larger activity of the simulation-based functions may occasionally introduce additional optima (turning unimodal into multimodal problems). The estimation is more likely to reproduce the single valley of the ground-truth. Conversely, the simulation excels for the multimodal cases because it does not remove optima by interpolation. For higher-dimensional cases ($n = 10, 20$), this situation changes: the simulation produces lower errors, regardless of modality. This is explained by the increasing sparseness of the training data, which in case of estimation will frequently lead to extremely poor search landscapes. The estimation will mostly produce a constant value, with the exception of very small areas close to the training data.

As noted earlier, results between DE, RS, and NM exhibit similar patterns. There is an exception, where the results between DE, NM, and RS differ more strongly: BBOB function 21 and 22. Hence, fig. 4 shows results for $n = 20$ with function 21 (22 is nearly identical). Here, each plot shows the error for a different algorithm. Coincidentally, this includes the only case where estimation is performing considerably better than simulation for large n (with algorithm RS only). The reason is not perfectly clear. One possibility is a particularly poor model quality. BBOB functions 21 and 22 are both based on a mixture of Gaussian components. Two aspects of this mixture are problematic for GPR: Firstly, they exhibit a peculiar, localized non-stationarity. The activity of the function

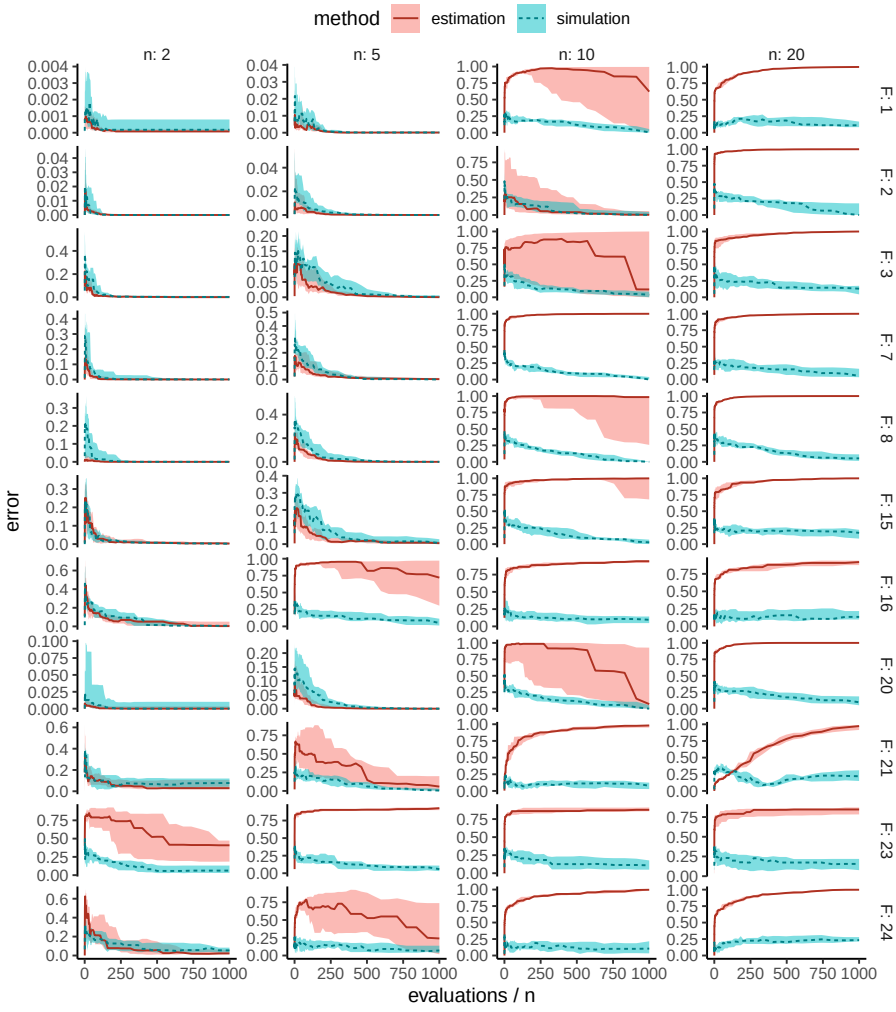

Fig. 2. The error of algorithm performance with simulation- and estimation-based test functions. The curves are based on the performance of a DE run on the model-based test functions, compared against the performance values on the ground-truth, i.e., the respective BBOB functions. The labels on the right-hand side specify the respective IDs of the BBOB functions. Top-side labels indicate dimensionality. The lines indicate the median, the colored areas indicate the first and third quartile. These statistics are calculated over the 15 instances for each BBOB function.

may abruptly change direction, depending on the closest Gaussian component. Secondly, overlapping Gaussian components produce discontinuities.

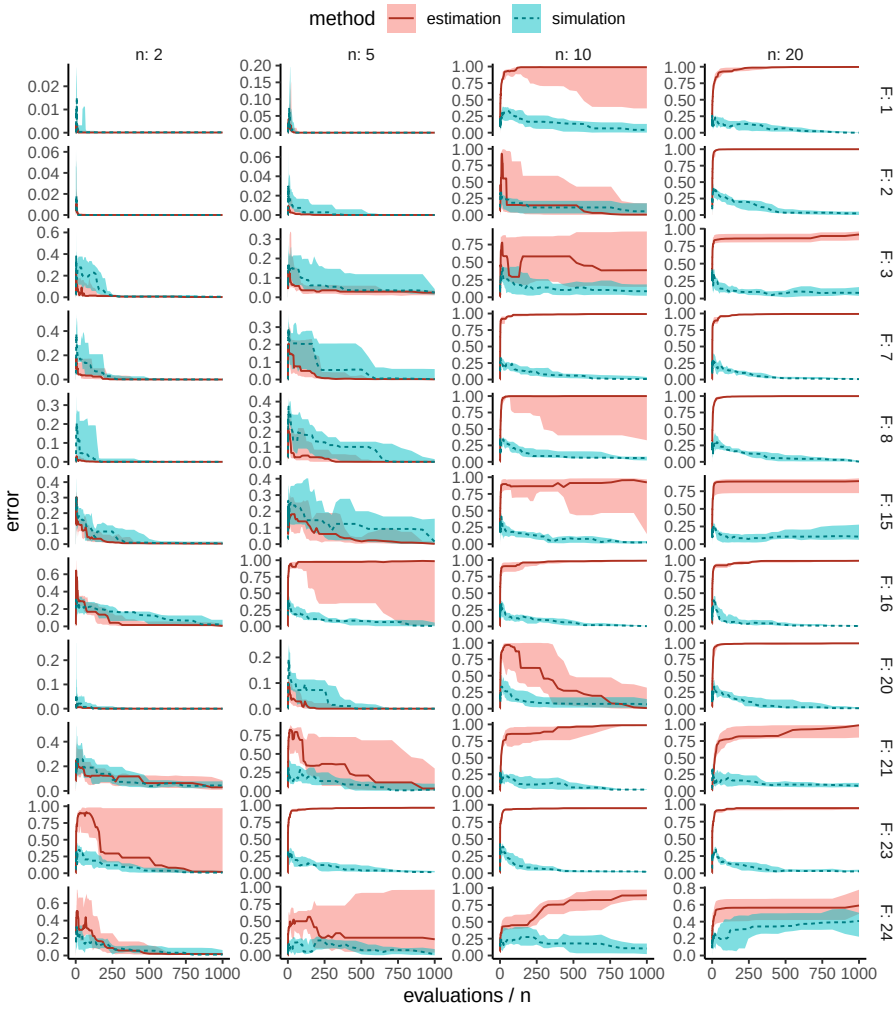

Fig. 3. This is the same plot-type as presented in fig. 2, but only for the performance of the NM algorithm (instead of DE).

5 Discussion

The results show that the model-based test functions will occasionally deviate considerably from the ground-truth. This has various reasons.

– **Dimensionality:** Clearly, all models are affected by the curse of dimensionality. With 10 or more variables, it becomes increasingly difficult to learn the shape of the real function with a limited number of training samples. Necessarily, this limits how much we can achieve. Despite its more robust performance, simulation also relies on a well-trained model.

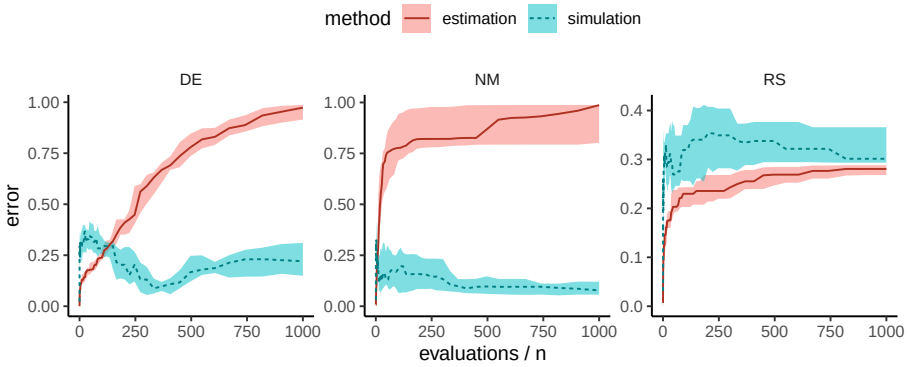

Fig. 4. This is the same plot-type as presented in fig. 2, but limited to BBOB function 21 and $n = 20$. While fig. 2 only shows results for DE, this figure compares results for each tested algorithm (DE, NM, RS) for this specific function and dimensionality.

- **Continuity:** Our GPR model, or more specifically its kernel, works best if the ground-truth is continuous, i.e., $\lim_{\mathbf{x}\to\mathbf{x}'} \mathrm{f}(\mathbf{x}) = \mathrm{f}(\mathbf{x}')$. Else, model quality decreases. One example is the step ellipsoidal function (BBOB function 7). This weakness could be alleviated, if it is known a-priori: a more appropriate kernel such as the exponential kernel $\mathrm{k}(\mathbf{x}, \mathbf{x}') = \exp(\sum_{i=1}^{n} -\theta_i |x_i - x_i'|)$ could be used. However, this kernel may be less suited for the spectral simulation method [4].
- **Non-stationarity:** Our GPR models assume stationarity of the covariance structure in the data. Yet, some functions are obviously non-stationary. For example, the BBOB variant of the Schwefel function (BBOB function 20) behaves entirely differently close to the search boundaries compared to the optimal region (due to a penalty term). In another way, the two Gallagher's Gaussian functions (BBOB functions 21, 22) show a more localized type of non-stationarity. There, the activity of the function will change direction depending on the closest Gaussian component. Such functions are particularly difficult to model with common GPR models. Non-stationary variants of GPR exist, and might be better suited. A good choice might be an approach based on clustering [21]. Adapting the spectral method to that case is straight-forward. The simulations from individual models (for each cluster) can be combined (locally) by a weighted sum.
- **Regularity/Periodicity:** Several functions in the BBOB set have some form of regular, symmetric, or periodic behavior. One classical example is the Rastrigin function (e.g., BBOB 3, 4 and 15). While our models seemed to work well for these functions, their regularity, symmetry or periodicity is not reproduced. With the given models, this would require a much larger number of training samples. If such behavior is important (and known a priori), a solution may be to choose a new kernel that, e.g., is itself periodic. This

requires that the respective spectral measure of the new kernel is known, to enable the spectral simulation method [4].
- **Extremely fine-grained local structure:** Some functions, such as Schaffer's F7 (BBOB function 17, 18), have an extremely fine-grained local structure. This structure will quickly go beyond even a good model's ability to reproduce accurately. This will be true, even for fairly low-dimensional cases. While there is no easy way out, our results at least suggest one compensation: Many optimization algorithms will not notice such kind of fine-grained ruggedness. For instance, a mutation operator might easily jump across these local bumps, and rather follow the global structure of the function. Hence, an accurate representation of such structures may not be that important in practice, depending on the tested algorithms.
- **Number of samples:** The number of training data samples is one main driver of the complexity for GPR, affecting computational time and memory requirements. The mentioned cluster-GPR approach is one remedy [21].

6 Conclusion

Our first research question was:

Q1 How can simulation with GPR models be used to generate benchmarks for continuous optimization?

As an answer, we use the spectral method for GPR simulation [4]. As this method results in a superposition of cosine functions, it is well suited for continuous search spaces. Conversely, the previously used [23] decomposition-based approach is infeasible due to computational issues. Consecutively, we asked:

Q2 Do simulation-based benchmarks provide better results than estimation-based benchmarks, for continuous optimization?

Our experiments provide evidence that simulation-based benchmarks perform considerably better than estimation-based benchmarks. Only for low-dimensional ($n \leq 5$), unimodal problems did we observe an advantage for estimation. In practice, if the modality (and dimensionality) of the objective function is known, this may help to select the appropriate approach. In a black-box case, the simulation approach seems to be the more promising choice.

For future research, it would be interesting to investigate how well these results translate to non-stationary GPR models, as discussed in section 5. We also plan to investigate how parameters of the training data generation process affect the generated test functions, perform tests with broader algorithm sets, and demonstrate the approach with real-world applications. Finally, investigating other model types is of importance. Approaches with weaker assumptions than GPR, such as Generative Adversarial Networks [10], may be of special interest.

References

1. D. Ardia, K. M. Mullen, B. G. Peterson, and J. Ulrich. DEoptim: Differential evolution in R. `https://CRAN.R-project.org/package=DEoptim`, 2020. Version 2.2-5, Accessed: 2020-02-25.

2. T. Bartz-Beielstein. How to create generalizable results. In J. Kacprzyk and W. Pedrycz, editors, *Springer Handbook of Computational Intelligence*, pages 1127–1142. Springer, Berlin, 2015.
3. T. Bartz-Beielstein, J. Stork, M. Zaefferer, C. Lasarczyk, M. Rebolledo, J. Ziegenhirt, W. Konen, O. Flasch, P. Koch, M. Friese, L. Gentile, and F. Rehbach. SPOT - Sequential Parameter Optimization Toolbox - v20200429. `https://github.com/bartzbeielstein/SPOT/releases/tag/v20200429`, 2020. Accessed: 2020-04-29.
4. N. A. Cressie. *Statistics for Spatial Data*. Wiley, New York, NY, 1993.
5. N. Dang, L. Pérez Cáceres, P. De Causmaecker, and T. Stützle. Configuring irace using surrogate configuration benchmarks. In *Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 243–250, Berlin, Germany, July 2017. ACM.
6. S. J. Daniels, A. A. M. Rahat, R. M. Everson, G. R. Tabor, and J. E. Fieldsend. A suite of computationally expensive shape optimisation problems using computational fluid dynamics. In A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV*, pages 296–307. Springer International Publishing, Coimbra, Portugal, 2018.
7. A. Fischbach, M. Zaefferer, J. Stork, M. Friese, and T. Bartz-Beielstein. From real world data to test functions. In *26. Workshop Computational Intelligence*, pages 159–177, Dortmund, Germany, Nov. 2016. KIT Scientific Publishing.
8. O. Flasch. *A modular genetic programming system*. PhD thesis, Technische Universität Dortmund, Dortmund, Germany, May 2015.
9. A. Forrester, A. Sobester, and A. Keane. *Engineering Design via Surrogate Modelling*. Wiley, New York, NY, 2008.
10. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.
11. N. Hansen, A. Auger, O. Mersmann, T. Tusar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *ArXiv e-prints*, Aug. 2016. ArXiv ID: 1603.08785v3.
12. N. Hansen, D. Brockhoff, O. Mersmann, T. Tusar, D. Tusar, O. A. ElHara, P. R. Sampaio, A. Atamna, K. Varelas, U. Batu, D. M. Nguyen, F. Matzner, and A. Auger. COmparing Continuous Optimizers: numbbo/COCO on Github, Mar. 2019. `https://doi.org/10.5281/zenodo.2594848`.
13. N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Research Report RR-6829, inria-00362633, INRIA, Feb. 2009. `https://hal.inria.fr/inria-00362633`.
14. A. G. Journel and C. J. Huijbregts. *Mining Geostatistics*. Academic Press, London, 1978.
15. C. Lantuéjoul. *Geostatistical Simulation: Models and Algorithms*. Springer-Verlag Berlin Heidelberg, Berlin, 2002.
16. J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, Jan. 1965.
17. M. Preuss, G. Rudolph, and S. Wessing. Tuning optimization algorithms for real-world problems by means of surrogate modeling. In *Genetic and Evolutionary Computation Conference (GECCO'10)*, pages 401–408, Portland, OR, USA, July 2010. ACM.
18. G. Rudolph, M. Preuss, and J. Quadflieg. Two-layered surrogate modeling for tuning optimization metaheuristics. Technical Report TR09-2-005, TU Dortmund, Dortmund, Germany, Sept. 2009. Algorithm Engineering Report.

19. R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.
20. V. Volz, B. Naujoks, P. Kerschke, and T. Tušar. Single- and multi-objective game-benchmark for evolutionary algorithms. In *Genetic and Evolutionary Computation Conference (GECCO'19)*, Prague, Czech Republic, July 2019. ACM.
21. H. Wang, B. van Stein, M. Emmerich, and T. Bäck. Time complexity reduction in efficient global optimization using cluster Kriging. In *Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 889–896, Berlin, Germany, July 2017. ACM.
22. J. Ypma, H. W. Borchers, and D. Eddelbuettel. nloptr vers-1.2.1: R interface to nlopt. `http://cran.r-project.org/package=nloptr`, 2019. Accessed: 2019-11-20.
23. M. Zaefferer, A. Fischbach, B. Naujoks, and T. Bartz-Beielstein. Simulation-based test functions for optimization algorithms. In *Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 905–912, Berlin, Germany, July 2017. ACM.