CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER’S THESIS

Title:	Impementation of simultaneous application of several surrogate models to evolutionary optimization
Student:	Bc. Ján Juranko
Supervisor:	doc. Ing. RNDr. Martin Holeňa, CSc.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2017/18

Instructions

1. Get acquainted with surrogate modelling for black-box optimization and with an implementation of the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES).
   2. 2. Gain practical experience with the software platform Comparing Continuous Optimisers (COCO).
  3. 3. Design and implement several additional variants of Gaussian processes (GP) complementing the available integration of CMA-ES and GP into the COCO platform.
   4. 4. Test all the integrated GP implementations on noiseless and noisy benchmarks available in the COCO platform.
   5. 5. Improve the integration through an automatic selection of the most appropriate GP implementation.
   6. 6. Evaluate the impact of automatic selection of the GP implementation on the success of optimization by the CMA-ES for the benchmarks considered in 4.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague January 20, 2017

Czech Technical University in Prague

Faculty of Information Technology

Department of Software Engineering

Master's thesis

Implementation of simultaneous application of several surrogate models to evolutionary optimization

Bc. Ján Juranko

Supervisor: doc. Ing. RNDr. Martin Holeňa CSc.

9th May 2017

Acknowledgements

I would like to thank my supervisor doc. Ing. RNDr. Martin Holeňa CSc. as well as Mgr. Lukáš Bajer for their guidance and advice. I would also like to thank my family and Viktória Fedorová for their support. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme ”Projects of Large Research, Development, and Innovations Infrastructures” (CESNET LM2015042), is greatly appreciated.

Declaration

Citation of this thesis

Juranko, Ján. *Implementation of simultaneous application of several surrogate models to evolutionary optimization.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Táto práca sa zameriava na súčasné použitie viacerých Gaussovských procesov (GP) ako náhradných modelov metódy Covariance Matrix Adaptation Evolution Strategy (CMA-ES), ktorá je významným algoritmom v oblasti black-box optimalizácie. Práca obsahuje implementáciu v prostredí MATLAB v spojení s knižnicou Gaussian Processes for Machine Learning (GPML) a benchmarkovými testami z platformy COmparing Continuous Optimisers (COCO). Taktiež sa zaoberá výberom modelov, ktoré budú natrénované, ako aj určením najlepšieho z implementovaných algoritmov pre výber náhradného modelu, ktorý určuje model použitý pre budúcu generáciu CMA-ES. Práca obsahuje aj výsledky experimentov, ktoré preukázali zlepšenie celkového výkonu v porovnaní s použitím len jediného náhradného modelu.

Klúčové slová black-box optimalizácia, CMA-ES, náhradné modely, Gaussovské procesy.

Abstract

This thesis is focused on using multiple Gaussian processes (GP) simultaneously as surrogate models for the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) method, which is an important algorithm in the black-box optimization. The thesis contains an implementation in the MATLAB enviroment, which uses the Gaussian Processes for Machine Learning (GPML) library and benchmark tests from the COmparing Continuous Optimisers (COCO) platform. It also investigates the process of choosing the right models, which will be trained, as well as algorithms for the selection of the best surrogate model, which will be used for the next generation of CMA-ES. The thesis also contains the results of performed experiments that prove the improvement of overall performance when compared to using only one surrogate model.

Keywords black-box optimization, CMA-ES, surrogate models, Gaussian processes.

Contents

List of Figures

List of Tables

Introduction

The black-box optimization has been proven to be useful in multiple real-world scenarios, for example reducing the noise of airplanes at the departures [1] or analyzing the thermodynamic properties of acids [2]. Detailed informations about the optimized functions are not available and each evaluation of such functions has high cost. Evolutionary algorithms have been a successful approach in such situations.
In this thesis, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), which is one of the best evolutionary algorithms, is being used together with multiple surrogate models. The surrogate models are used to model the black-box function during the algorithm and lower the number of needed evaluations. The models are trained on the data from previous evaluations of the black-box function. Several types of surrogate models exists, such as Gaussian processes (GPs), random forests or neural networks.
A part of the thesis is dedicated to choosing the right set of GPs for each dimension and then using them simultaneously during the optimization. The best GP is chosen in each generation by one of implemented algorithms and it serves as the model for the next predictions. The performance is measured with a COCO platform on 24 noiseless and 6 noisy functions in 3 dimensions - 2D, 5D and 10D.
Chapter 1 introduces the black-box optimization, CMA-ES algorithm and GPs as surrogate models.
Chapter 2 contains the analysis and design of changes in integration of GPs and usage of multiple GPs.
Chapter 3 describes implementation and testing of the changes and algorithms used to choose the set of GPs.
Chapter 4 describes experiments that were part of this thesis and their results.
Chapter 5 concludes the thesis.

Chapter 1

State-of-the-art

1.1 Black-box optimization

Black-box function is defined as a function for which an analytic form is not known [3]. Available operations are therefore very restricted, usually only to evaluation in desired point. In case of minimalization, the objective is to find a point with functional value, $f(x)$, as small as possible [4]. Needless to say, it is a difficult task, especially when the search cost of the function (defined as number of evaluation) needs to be considered. Multiple approaches to this task have been developed, such as random search, swarm algorithms, evolution strategies (ES), etc. This thesis focuses on particular ES, which is called Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and will be described in details in 1.3.

1.2 Noiseless and noisy functions

The difference between noiseless and noisy functions is modification of output values of function by some type of noise. This thesis consideres three types of noise [5]:

1. Gaussian noise
$$f_{GN}(f, \beta) = f \times \exp(\beta \mathcal{N}(0,1))$$

2. Uniform noise
$$f_{UN}(f, \alpha, \beta) = f \times \mathcal{U}(0,1)^{\beta} \max\Big(1, (\tfrac{10^9}{f+\epsilon})^{\alpha \mathcal{U}(0,1)}\Big)$$

3. Cauchy noise
$$f_{CN}(f, \alpha, p) = f + \alpha \max\Big(0, 1000 + \mathbb{I}_{\{\mathcal{U}(0,1)<p\}} \tfrac{\mathcal{N}(0,1)}{|\mathcal{N}(0,1)|+\epsilon}\Big)$$

For better illustration graphs of Rosenbrock function

$$f_{rosenbrock}(x) = \sum_{i=1}^{D-1} 100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2$$

- $z = \max(1, \frac{\sqrt{D}}{8})(x - x^{opt}) + 1$
- $z^{opt} = 1$

with and without noise are included in Figures 1.1 and 1.2.

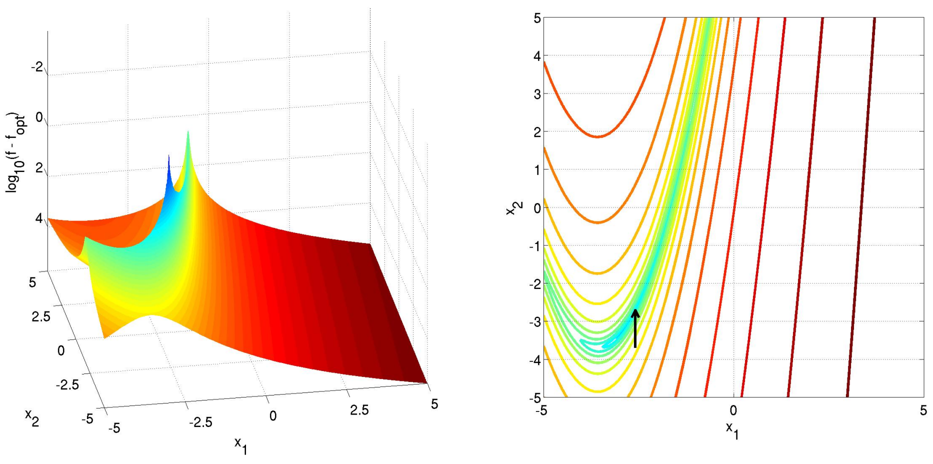


Figure 1.1: Noiseless Rosenbrock function [6]
Black arrow shows the position of optimum.

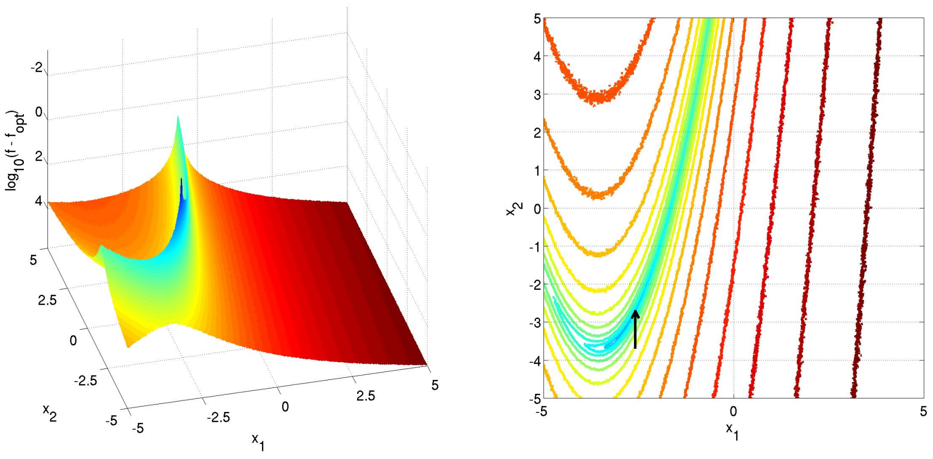


Figure 1.2: Noisy Rosenbrock function with uniform noise [5]

1.3 Covariance Matrix Adaptation Evolution Strategy

The covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a stochastic method for optimization of functions with continuous domain. It works with symmetric, positive definite covariance matrix C, which can be decomposed as

$$C = BD^2B^T,$$

where:
B is an orthogonal matrix, $B^TB = BB^T = I$. Columns of B form an orthonormal basis of eigenvectors.
$D = diag(d_1, ..., d_n)$ is a diagonal matrix with square roots of eigenvalues of C as diagonal elements.
This matrix is calculated for each generation (iteration of algorithm). It is used to generate λ new points using this formula:

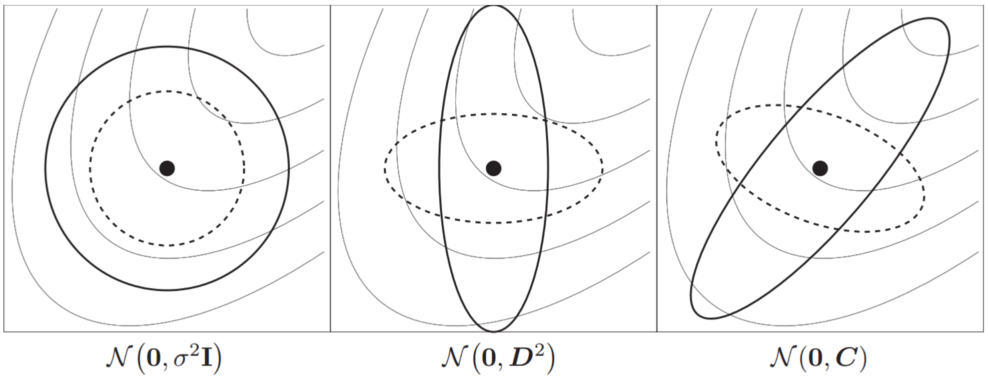
$$x_k^{(g+1)} \sim m^{(g)} + \sigma^{(g)}\mathcal{N}\Big(0, C^{(g)}\Big) \text{ for } k = 1, ...\lambda,$$

where:
x are points (also called offsprings/individuals),
m is the mean value of the distribution,
σ is a step-size, it is used to control how far from mean can the values be generated,
$N(0, C)$ is a multivariate normal distribution with covariance matrix C. Figure 1.3 is enclosed for an illustration of how the distributions can look like.

Points generated from this distribution are evaluated and best μ (μ is a parameter of the CMA-ES) points are used to calculate the parameters for next generation of the CMA-ES. This cycle repeats until some termination criterion is met.
Short pseudo-code of the CMA-ES algorithm:

```
1. Initialize distribution parameters $\theta^{(0)}$
2. For generation $g = 0, 1, 2, ...$
        3. Sample $\lambda$ independent points from multivariate normal
        distribution
        4. Evaluate the $f$ on sample $x_1, ..., x_\lambda$
        5. Update parameters $\theta^g$ for next generation
6. break, if termination criterion met
```

An example of the progress of the CMA-ES can be found in Figure 1.4.
One of the advantages of the CMA-ES is that it can be used when derivation is not available or not very useful (for example when optimizing noisy functions). More details about the CMA-ES can be found in [4].



$\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$
$\mathcal{N}(\mathbf{0}, \boldsymbol{D}^2)$
$\mathcal{N}(\mathbf{0}, \boldsymbol{C})$

Figure 1.3: Different multivariate normal distributions [4]
Two different distributions are shown on each of the three pictures.
The first picture explains function of σ (step-size) with $C = I$.
The second pictore shows how the D^2 matrix modifies a circle into an ellipse.
The third picture uses C that is equal to BD^2B^T, which rotates the ellipse into desired direction.

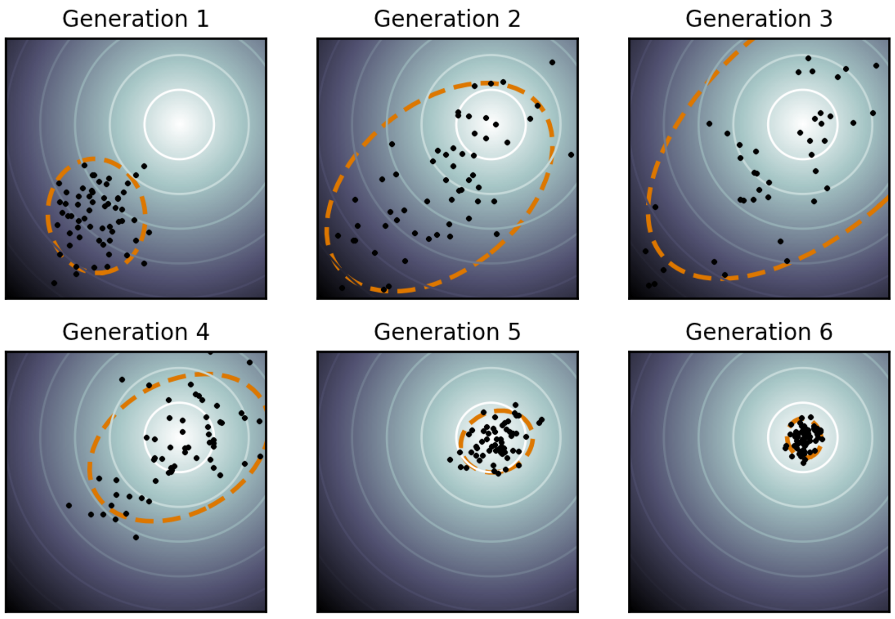


Figure 1.4: Example of progress of the CMA-ES [7]
Orange ellipse represents C, black dots are sampled points, white color represents area with optimal function values.

1.3.1 Surrogate models

One way to decrease the search cost of CMA-ES is to use a surrogate model, which models the black-box function. The surrogate model is a regression model, which is built using data available from previous iterations of the algorithm and is used for prediction of function value in sampled points instead of evaluation, which uses the function. That splits the points encountered by the algorithm into two types:

1. evaluated with the original black-box objective function, which increases the search cost,
2. evaluated with the surrogate model, which doesn't increase the search cost, but on the other hand is imprecise.

The goal of creating a surrogate model is for it to be as precise as possible. For the CMA-ES algorithm, it is also possible to focus only on the area that is being currently sampled. Choosing the right training data for the surrogate model is a great challenge. On the one hand, we don't want to choose too many points for the training set, because we want the model to be optimized for the area needed in current iteration. On the other hand we need to choose enought points for the model to be useful. The location of the data (for example, closest to currently sampled points or mean) is also a factor that should be considered in choosing the right set of training data.

1.4 Gaussian processes

Gaussian processes (GPs) specify a probabilistic model over a given set of data points. This model can then be extended to predict the mean and standard deviation of the function value at new data points. GPs have a small number of hyperparameters which can be set by the user, or optimized via a maximum likelihood approach. GPs can approximate any function. One drawback of GPs is their computational cost: for N data points, it takes $\mathcal{O}(N^3)$ steps to construct the GP, $\mathcal{O}(N)$ to predict the mean function value at a new point, and $\mathcal{O}(N^2)$ to predict the standard deviation.

Let $f(x)$ be an unknown scalar function and $x \in \mathbb{R}^n$ a point in an n-dimensional decision space. Evaluating f at N data points $X_N = \{x_1, x_2, ..., x_N\}$ yelds the function values $y_N = \{y_1, y_2, ..., y_N\}$, where $(\forall i) y_i = f(x_i)$. The modeling task is to predict the function value y_{N+1} at a new point x_{N+1}. The GP model imposes a probabilistic model on the data: the vector of known function values y_N is one sample of a N-dimensional multivariate Gaussian distribution

$$\mathcal{N}(m(X_N), K(X_*, X_*))$$

with the value of the probability density $p(y_N | X_N)$. The mean function $m(X_N)$ can be set to 0, a constant, a linear function, etc. The value of the probability density in the new point x_{N+1} is defined as (see [8] for details):

$$p(y_{N+1}|X_{N+1}) = \frac{\exp(-\frac{1}{2}y_{N+1}^T C_{N+1}^{-1} y_{N+1})}{\sqrt{(2\pi)^{N+1} det(C_{N+1})}}$$

C_{N+1} is the covariance matrix of the Gaussian distribution for the $N+1$ points and can be written as

$$C_{N+1} = \begin{pmatrix} C_N & k \\ k^T & \kappa \end{pmatrix}$$

where
C_N is the covariance matrix of the Gaussian distribution constructed from trainig data,
k is a vector containing covariances between the N training data and the new point
and κ is the variance of the new point.
The covariance matrix C_N can be constructed from training data using the covariance-function matrix K_N described below and signal noise σ as $C_N = K_N + \sigma I_N$ where I_N is the identity matrix of order N.

1.4.1 Covariance functions

The matrix K_N can be defined by different so-called covariance functions, for example in [9] squared exponential function is used, which is defined as

$$K(x_i, x_j) = \theta \exp\Big(-\frac{1}{2}(x_i - x_j)^T M (x_i - x_j)\Big)$$

where x_i and x_j are data points, θ is a parameter called signal variance and M is a symmetric matrix, which can be set to:

- $M_1 = l^{-2} I$,
- $M_1 = diag(l)^{-2}$ - this is a variant with so-called automatic relevance determination (ARD).

l is a parameter called characteristic lengthscale with which the distance of two considered data points is compared.
An additional example of covariance functions is Matérn class:

$$k_{Matern}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{l}\right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu} r}{l}\right)$$

where r is distance between the two points, Γ is the gamma function, K_ν is the modified Bessel function, and l and ν are positive parameters.

The two variants of this function, which are used in this thesis, are:

$$k_{\nu=3/2}(r) = \left(1 + \frac{\sqrt{3}r}{l}\right) \exp\left(-\frac{\sqrt{3}r}{l}\right),$$

$$k_{\nu=5/2}(r) = \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right).$$

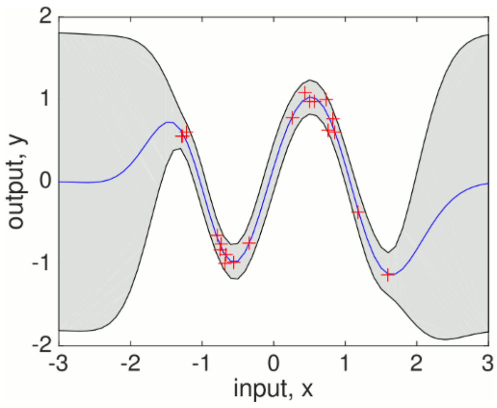


Figure 1.5: Example of modelling with a 1-dimensional GP. [10]
Grey area represents predictive 95% confidence bounds, red crosses represent training points and blue line represents predictive mean.

1.4.2 Prediction

Prediction of the value of function in GPs in a new point x_{N+1} is governed by a probability distribution with the density:

$$p(y_{N+1}|X_{N+1}, y_N) \propto \exp\Big(-\frac{1}{2}\frac{(y_{N+1} - \hat{y}_{N+1})^2}{s^2_{y_{N+1}}}\Big)$$

where $\hat{y}_{N+1}$ is its given by

$$\hat{y}_{N+1} = m(x_{N+1}) + k^T C_N^{-1} y_N - m(x_{N+1})$$

($m(x_{N+1})$ is a mean function) and $s^2_{y_{N+1}}$ is its variance given by

$$s^2_{y_{N+1}} = \kappa - k^T C_N^{-1} k.$$

For more information about GPs, see sources of this section: [8], [9], [11], [12].

Chapter 2

Analysis and design

The author of this thesis was participating in a project that was already developed by several authors. The source code can be found at `https://github.com/bajeluk/surrogate-cmaes` [13].

2.1 COCO platform

"The COCO (COmparing Continuous Optimisers) is a platform for systematic and sound comparisons of real-parameter global optimisers. COCO provides benchmark function testbeds, experimentation templates which are easy to parallelize, and tools for processing and visualizing data generated by one or several optimizers. The COCO platform has been used for the Black-Box-Optimization-Benchmarking (BBOB) workshops that took place during the GECCO conferences in 2009, 2010, 2012, 2013 and 2015. It was also used at the IEEE Congress on Evolutionary Computation (CEC'2015) in Sendai, Japan." [14] Optimization in COCO stops when the difference of achieved function value from optimal function value is smaller than 10^{-8}, or when the maximal number of evaluations is reached (usually $250 * \mathit{dimension}$ in this project).
The COCO platform was already integrated in the project, as well as generating reports from results. Example of one graph can be found in Figure 2.1.
The project was focused on optimizing noiseless functions, but in this thesis, noisy functions are used as well.
The current 24 noiseless test functions in COCO platform are [14]:

1. Separable Functions

   f1 Sphere Function

   f2 Ellipsoidal Function

   f3 Rastrigin Function

   f4 Büche-Rastrigin Function

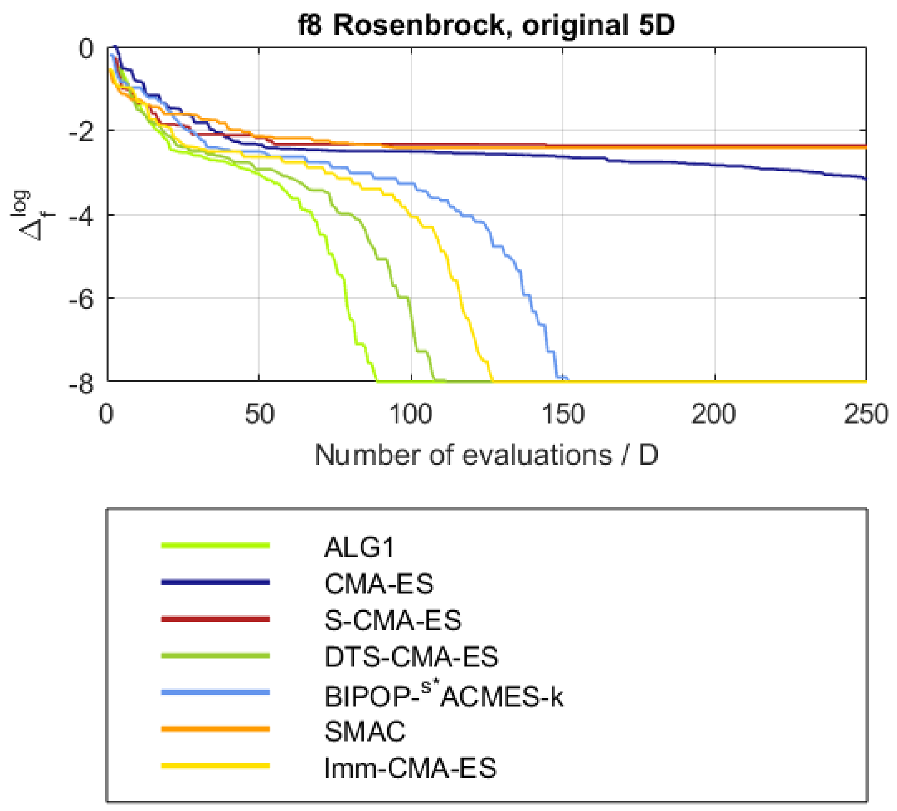


Figure 2.1: Example of a graph from a generated report.
The axis X represents the number of evaluations divided by the dimension (in this case 5). In the case of multiple instances, the median of the values from all instances is shown.
The axis Y represents difference between the optimal function value and the current function value. ALG1 represents tested algorithm, others are different algorithms for comparison.

f5 Linear Slope

2. Functions with low or moderate conditioning

   f6 Attractive Sector Function

   f7 Step Ellipsoidal Function

   f8 Rosenbrock Function, original

   f9 Rosenbrock Function, rotated

3. Functions with high conditioning and unimodal

   f10 Ellipsoidal Function

   f11 Discus Function

   f12 Bent Cigar Function

   f13 Sharp Ridge Function

   f14 Different Powers Function

4. Multi-modal functions with adequate global structure

   f15 Rastrigin Function

   f16 Weierstrass Function

   f17 Schaffers F7 Function

   f18 Schaffers F7 Functions, moderately ill-conditioned

   f19 Composite Griewank-Rosenbrock Function F8F2

5. Multi-modal functions with weak global structure

   f20 Schwefel Function

   f21 Gallagher's Gaussian 101-me Peaks Function

   f22 Gallagher's Gaussian 21-hi Peaks Function

   f23 Katsuura Function

   f24 Lunacek bi-Rastrigin Function

There are 30 noisy functions in COCO, this thesis is deals only with the functions with moderate noise, which are:

f101 Sphere with moderate Gaussian noise

f102 Sphere with moderate uniform noise

f103 Sphere with moderate seldom Cauchy noise

f104 Rosenbrock with moderate Gaussian noise

f105 Rosenbrock with moderate uniform noise

f106 Rosenbrock with moderate seldom Cauchy noise

All of the functions are available in different dimensions, as well as different instances of a function. "For each instance the randomly chosen values are drawn anew. The implementation provides an instance ID as input, such that a set of uniquely specified instances can be explicitly chosen." [6] In this project, 15 instances of each function are tested and their IDs are 1 to 5 and 41 to 50. That was a setting used at the GECCO conference in 2015.

2.2 CMA-ES

The CMA-ES is integrated using several different strategies that implement the class EvolutionControl and are created by an instance of the class ECFactory. Parameters of the CMA-ES ($C, \sigma, \lambda, m, etc.$) are passed as a value of the variable cmaesState and options that are used to create a surrogate model are passed as a value of the variable surrogateOpts. The strategy DoubleTrainedEC has been most successful evolution control in this project so far and for that reason this thesis uses only the DoubleTrainedEC.

2.3 Surrogate models

DoubleTrainedEC, as well as others, use a surrogate model that is an instance of the Model class. The Model class has following methods:

- getNTrainData() - returns minimal number of points for training the model,
- trainModel(X, y, xMean, generation) - used to train the model on data X with function values y, usually called from method train,
- modelPredict(X) - predicts function values in data X,
- train(X, y, stateVariables, sampleOpts) - main method for the training, usually used as a wrapper of the trainModel method.
- isTrained() - training of the model is not always successful, for example model is considered untrained if the difference between maximal and minimal predicted values of the population is lower than MIN_RESPONSE_DIFFERENCE (constant that is set in project). This method returns whether or not the model is trained,
- getTrainsetSize() - returns the size of training data,
- shift(xMean) - transforms the trained model to new coordinates,

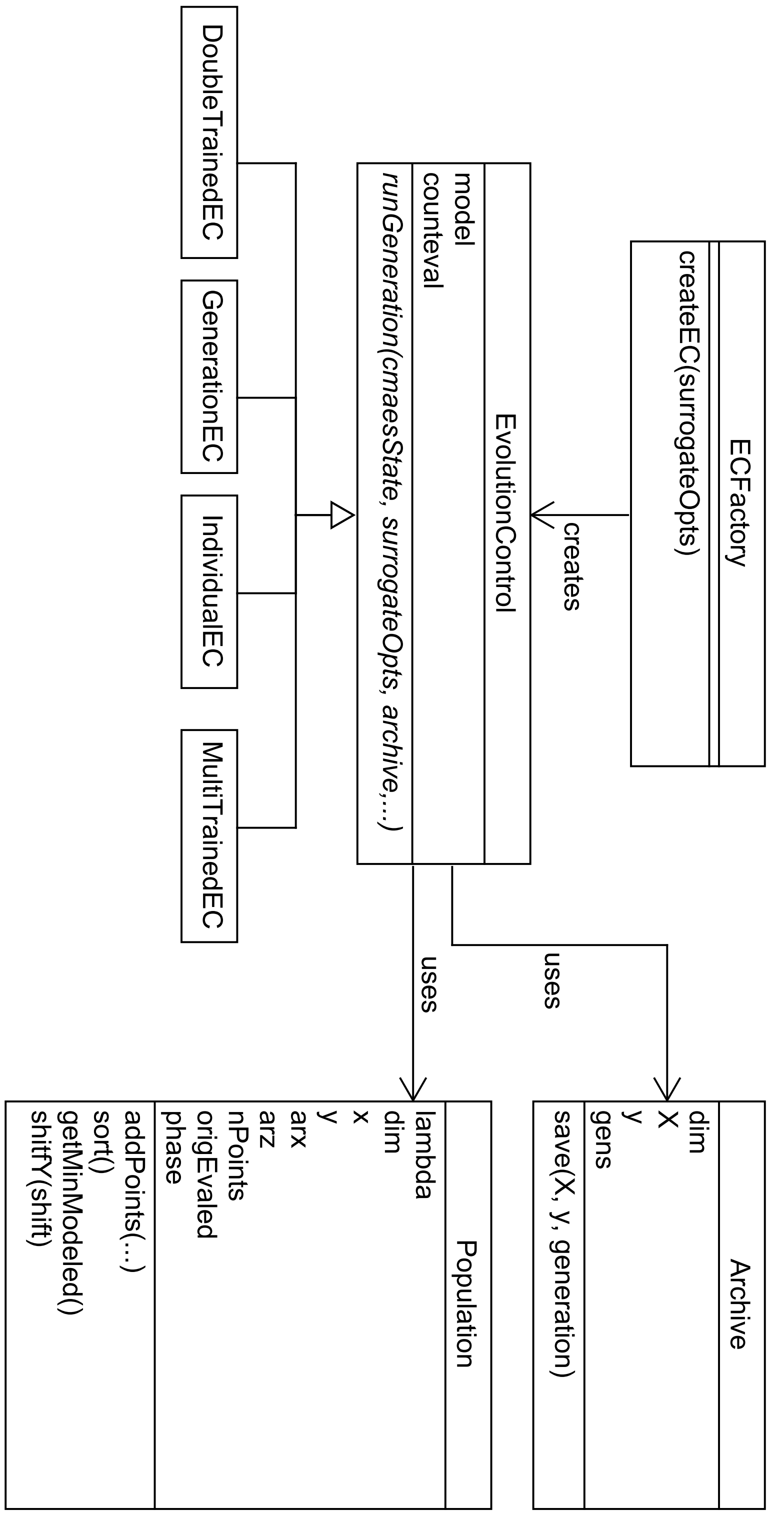


Figure 2.2: Part of the class diagram - EvolutionControl

- generationUpdate(...) - deprecated, also used for shifting the model to new coordinates,
- predict(X) - predicts function values, usually used as wrapper for modelPredict,
- getModelOutput(X) - returns different criteria for different ECs,
- getNearMean(xMean, deniedIdxs) - returns a point from the training set, which is near to the mean.

Previously, the EC was calling the Archive to get the training data and passed them as parameter to the Model in the train method. The description of the training data for the surrogate models was moved into the Model class. Choosing the training set was moved into the Archive class and is now called directly from the Model class during the training of the model. Following options for choosing the training data were designed:

trainRange Only the points that are in this range are considered. It is χ^2 percentile (with degree of freedom parameter set to dimension of the black-box function) of the maximum squared $\sigma^2 C$-distance from the current CMA-ES mean, or from any point in population in case of nearestToPopulation trainsetType (see next item). Tested options were 99.9% and 100% (which means no limitation).

trainsetType defines which points will be chosen in case of multiple points in trainRange. Options are:

- allPoints - No special algorithm, in case of too many points in trainRange, newer ones (that means points from newer generations) are chosen.
- clustering - The points are divided into clusters and from each cluster, one point is chosen. This algorithm has been already implemented and used in the project.
- nearest - The points that are nearest to the mean are chosen.
- nearestToPopulation - The points are sorted by the distance to their closest point from the population and those for which that distance is the lowest are chosen.

trainsetSizeMax used to control maximal number of points used for training the model. Tested options were '5*dim', '10*dim', '15*dim', '20*dim' (dim is dimension of the black-box function).

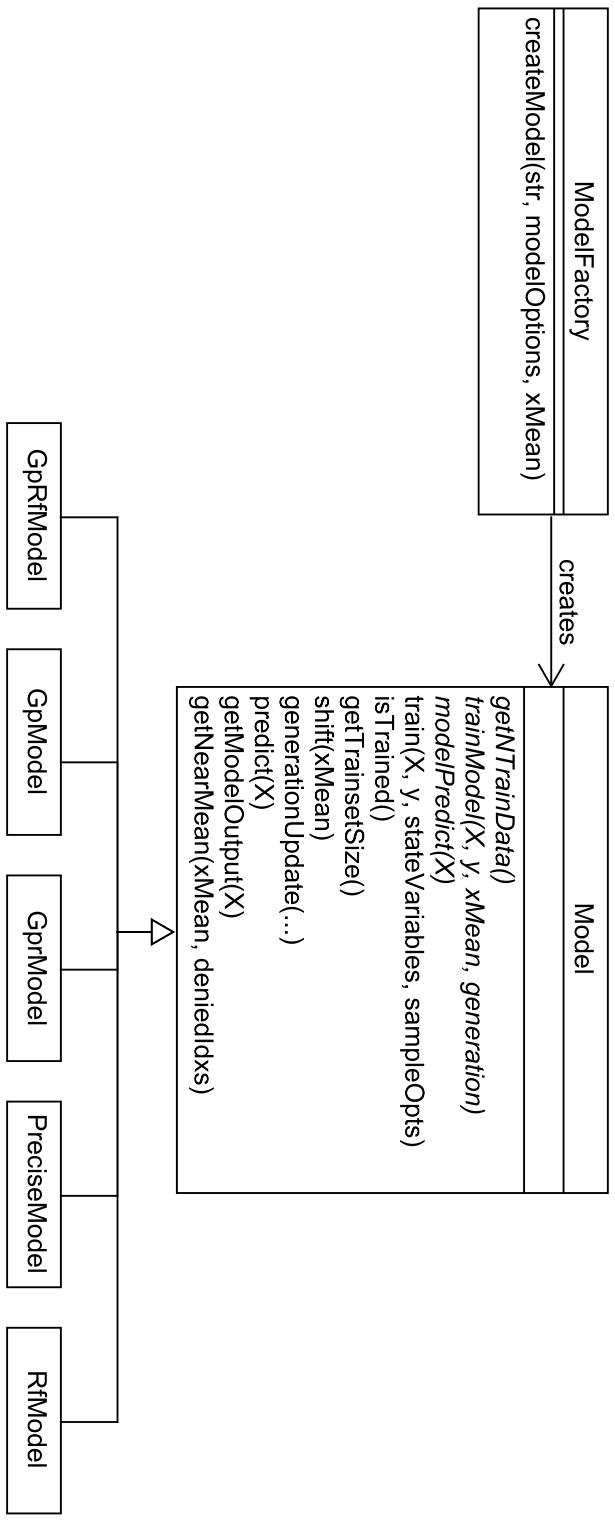


Figure 2.3: Part of the class diagram - Model class

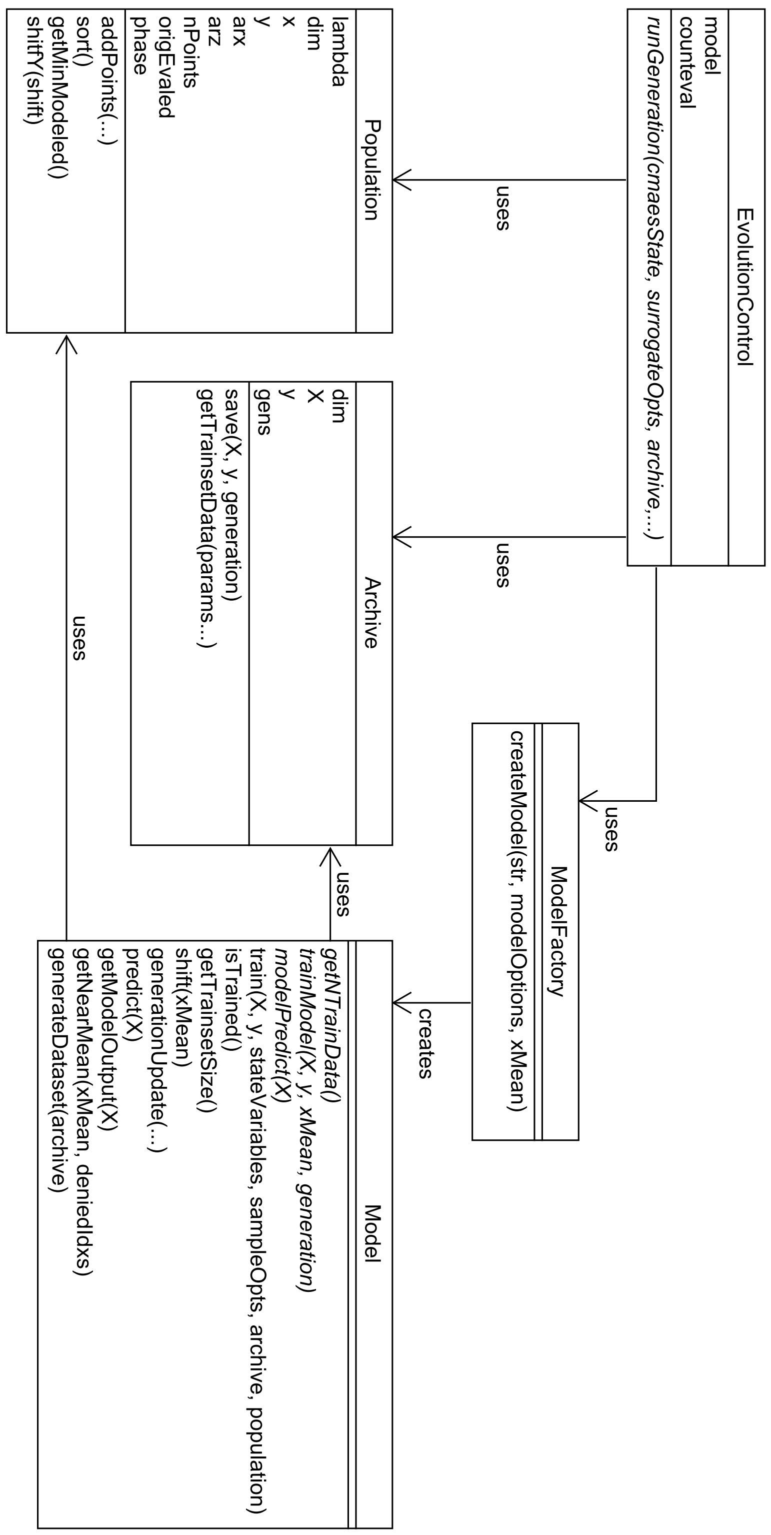


Figure 2.4: Part of the class diagram - training of the Model class

2.4 Gaussian processes

Gaussian processes are integrated by using Gaussian Processes for Machine Learning (GPML) library. This library offers different parameters for GPs:

- covariance functions,
- mean functions,
- hyperparameters of GPs.

Following covariance functions were chosen to be examined:

- {@covMaterniso, 3} - Matérn, $\nu = 3/2$,
- {@covMaterniso, 5} - Matérn, $\nu = 5/2$,
- {@covSEiso}, - Squared exponential,
- {@covSEard} - Squared exponential with ARD.

GPML contains a lot more of covariance functions, testing all of them would require too much computational power, that is why only 4 of them were chosen. Covariance functions are passed as ”covFcn” option to the GpModel. A complete list of covariance functions in GPML with descriptions is available here: `http://www.gaussianprocess.org/gpml/code/matlab/cov/`.
Two options were chosen as the mean functions:

- meanConst - constant mean function,
- meanLinear - linear mean function.

List of all possible mean functions in GPML is available here:
`http://www.gaussianprocess.org/gpml/code/matlab/meanFunctions.m`.
Hyperparameters of GPs were not optimized in this thesis (also because of required computational power) and were set to their default values ”struct('lik', log(0.01), 'cov', log([0.5; 2]))”. Hyperparameters are passed as the ”hyp” option to the GpModel.

2.5 ModelPool

The ModelPool class is designed to contain multiple instances of GpModel objects. It inherits from the Model class and implements its abstract methods in a following way:

- getNTrainData() - returns the minimum of values returned by the getNTrainData() function over the GpModels contained in the pool.

- trainModel(X, y, xMean, generation) - this method is empty, as it is not needed,
- modelPredict(X) - calls modelPredict(X) of the GpModel found, according to a given criteria as the best in the current iteration.

The ModelPool class is not a typical representant of the Model class, mainly because the training of the model, which was ment to be in trainModel method, is not needed. The training of ModelPool is done by overriding the method train and calling the same method in each of the GpModels. The isTrained method is overriden as well, and returns true if at least 1 of the contained models is trained. Another specificity of the ModelPool is that it is ment to be trained multiple times, not only once, as other types of models. Multiple options for ModelPool were designed, such as:

historyLength Used to control how many generations of the older models are saved. The older models are used to choose the best model.

retrainPeriod Used to control how often should the ModelPool train GpModels. Training is done only in the generations that are multiples of the retrainPeriod. Benefit of setting this option to higher than 1 is lower computational cost. This option is always set to 1 in the experiments performed in this thesis, because that leads to training with most recent data, which should generate more precise GpModels.

bestModelSelection Sets the algorithm of choosing the best model. Described in details in 2.5.1 below.

minTrainedModelsPercentileForModelChoice When the percentile of the trained GpModels in the oldest generation is lower than this value, a whole generation is not used to compare which models are the best. In that case, the next generation of models is used for comparison. The motivation for this option is the following - in case of a lot of untrained models in an older generation, only those newest models could be chosen as the best, whose older generation was considered as trained. That can lead to having trained models, that could not be chosen as the best. This problem can be partially solved by setting this option together with the next option - maxGenerationShiftForModelChoice.

maxGenerationShiftForModelChoice Choosing newer generations because of untrained models can be done only as many times as is the value of this parameter. The reason for this is that when a generation that is used to compare the performance of models is too young, there are not enough points evaluated using the original black-box function for comparing the performance of GpModels.

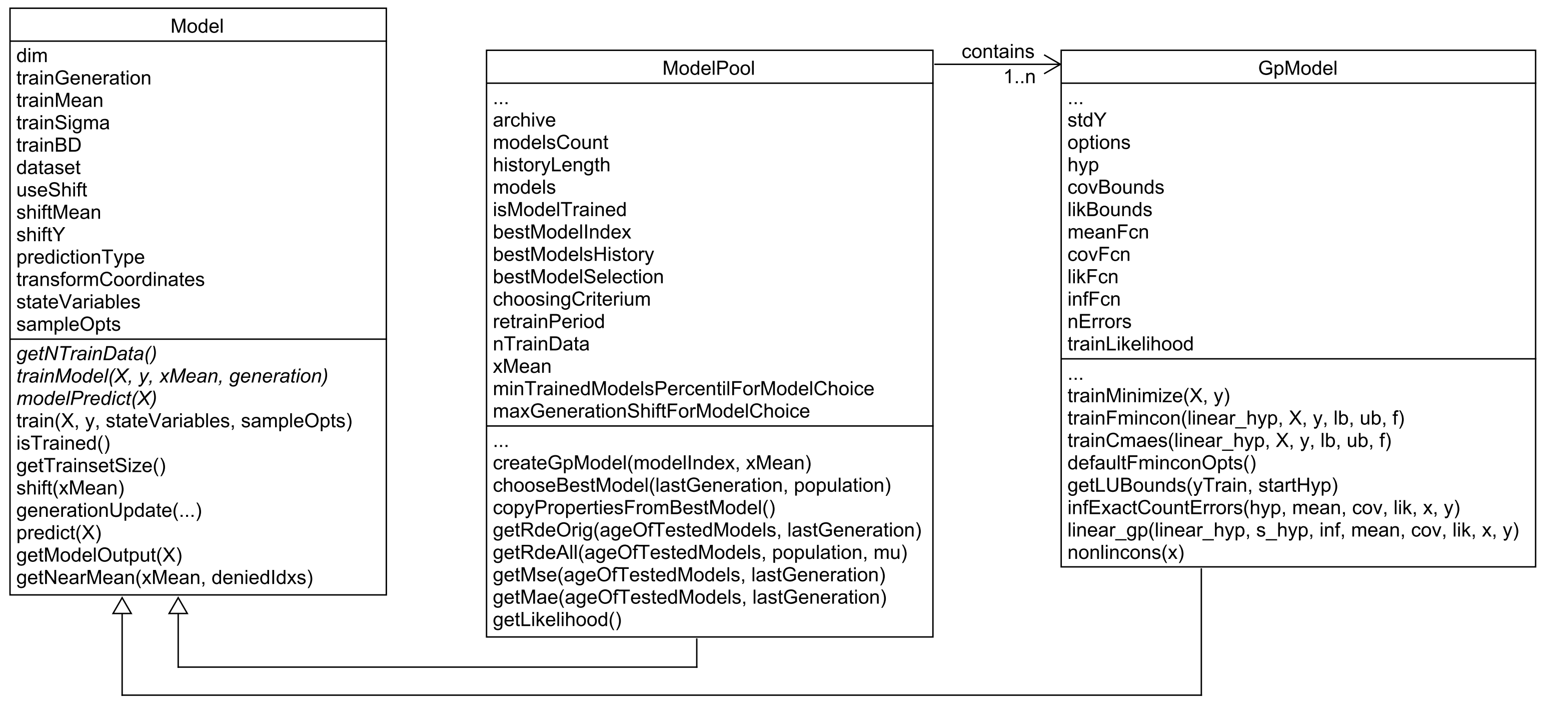

Model
dim
trainGeneration
trainMean
trainSigma
trainBD
dataset
useShift
shiftMean
shiftY
predictionType
transformCoordinates
stateVariables
sampleOpts
getNTrainData()
trainModel(X, y, xMean, generation)
modelPredict(X)
train(X, y, stateVariables, sampleOpts)
isTrained()
getTrainsetSize()
shift(xMean)
generationUpdate(...)
predict(X)
getModelOutput(X)
getNearMean(xMean, deniedIdxs)
ModelPool
...
archive
modelsCount
historyLength
models
isModelTrained
bestModelIndex
bestModelsHistory
bestModelSelection
choosingCriterium
retrainPeriod
nTrainData
xMean
minTrainedModelsPercentilForModelChoice
maxGenerationShiftForModelChoice
...
createGpModel(modelIndex, xMean)
chooseBestModel(lastGeneration, population)
copyPropertiesFromBestModel()
getRdeOrig(ageOfTestedModels, lastGeneration)
getRdeAll(ageOfTestedModels, population, mu)
getMse(ageOfTestedModels, lastGeneration)
getMae(ageOfTestedModels, lastGeneration)
getLikelihood()
contains
1..n
GpModel
...
stdY
options
hyp
covBounds
likBounds
meanFcn
covFcn
likFcn
infFcn
nErrors
trainLikelihood
...
trainMinimize(X, y)
trainFmincon(linear_hyp, X, y, lb, ub, f)
trainCmaes(linear_hyp, X, y, lb, ub, f)
defaultFminconOpts()
getLUBounds(yTrain, startHyp)
infExactCountErrors(hyp, mean, cov, lik, x, y)
linear_gp(linear_hyp, s_hyp, inf, mean, cov, lik, x, y)
nonlincons(x)

Figure 2.5: ModelPool class

2.5.1 Choosing the best model in ModelPool

After training all of the models in the ModelPool, one needs to be chosen as the best and will be used to predict values in modelPredict method. Several options for this choice have been designed using the following criteria:

RDE Ranking difference error :

$$\mathrm{RDE}(y_1^*, y_2^*, \mu) = \frac{\sum_{i:\tau_2(i)\leq\mu} |\tau_2(i) - \tau_1(i)|}{\max_{\pi\in\text{Permutations of }(1,\dots,\lambda)} \sum_{i:\pi(i)\leq\mu} |i - \pi(i)|} \in \langle 0, 1\rangle.$$

RDE measures how much would the difference between ranking of y_1^* and y_2^* (possibly negatively) contribute for the CMA-ES μ-updates where $\mu, 1 \leq \mu \leq \lambda$ is the number of best-ranked individuals which are solely used for the CMA-ES updates.
The measure takes rankings τ_1, τ_2 of the values of the input vectors (e.g. $\tau_1(1)$ is the rank of the $y_{1(1)}^*$ - the first element of the vector y_1^*), and calculates the ranking distance as a normalized sum of the element-wise differences between these two rankings while ommiting the indices for which the rank of y_2^* is larger than μ. The second vector y_2^* is considered as being more precisely measured.

MAE Mean absolute error:

$$\mathrm{MAE}(y_1, y_2) = \frac{1}{n}\sum_{i=1}^{n}(|y_{1(i)} - y_{2(i)}|)$$

MSE Mean squared error:

$$\mathrm{MSE}(y_1, y_2) = \frac{1}{n}\sum_{i=1}^{n}(y_{1(i)} - y_{2(i)})^2$$

The MSE penalizes the larger differences when compared to MAE.

Likelihood For each set of parameters of the GpModels a likelihood property is calculated which expresses how well the model describes the training data. In this project, the lower value of likelihood property is better, because the value saved in the property is a negative log-likelihood.

The options for choosing the best model are:

rdeOrig Calculates RDE using points evaluated with black-box function and values predicted by each of the oldest GpModels. μ parameter is set to number of such points.

rdeAll Calculates RDE using whole populations in each of the generations (not only the oldest one). If the population of the current generation contains some evaluated points, those are considered as well. The partial RDE results from each of the generations are then given weights. The sum of the weights is 1 and the weight of each newer result is half of the weight of the previous one.

mae Calculates MAE using points evaluated with black-box function and values predicted by each of the oldest GpModels.

mse Calculates MSE using points evaluated with black-box function and values predicted by each of the oldest GpModels.

likelihood GpModel with lowest value of likelihood property is chosen. In this case, only 1 generation of models is trained. This option is also used, when there are not enough of the trained generations (beginning of the CMA-ES), because it does not need older generations.

In each of the options the model whose result is the lowest one is chosen for the next predictions. The options that are calculated from the function values (all except the likelihood option) are using only the points from generations that are newer than generation of their training. Desired option is passed in the ”bestModelSelection” option for the ModelPool.

CHAPTER 3

Implementation

All of the changes in the training of the models that were done needed to be backwards compatible to ensure that older experiments done in this project will be still working as intended when run again. That is accomplished by using optional parameters (Archive and Population) in the train method, as well as using a default option "parameters" when setting the trainsetType option, which causes the model to be trained using the data passed in parameters (X, y) instead of retrieving the data from the Archive.

3.1 Gaussian processes

The process of choosing the GPs for the ModelPool was based on data from a previous experiment in the project (exp_doubleEC_21), which was run with options that enable the logging of the CMA-ES progress. That caused parameters of the CMA-ES to be saved for each of the generations. After the experiment finished, a file called "dataset" was created using the function "modelTestSets(exp_id, fun, dim, inst, opts)", which created 10 so-called snapshots for each function, dimension and instance. Each of the 10 snapshots represent one generation of the algorithm, with equal number of generations between each of the snapshots.
The following set of options was chosen to be tested for choosing the best GPs for ModelPool :

trainsetSizeMax '5*dim', '10*dim', '15*dim', '20*dim',

trainsetType 'allPoints', 'clustering', 'nearest', 'nearestToPopulation',

trainRange '0.999', '1',

covFcn '{@covMaterniso, 3}', '{@covMaterniso, 5}', '{@covSEiso}', '{@covSEard}',

meanFcn 'meanConst', 'meanLinear',

which gives a total of 256 different GpModel combinations. These settings can be found in experiments exp_GPtest_01 and exp_GPtest_02_noisy. The generated dataset was then used to train all of the chosen 256 GpModels in each of the functions, dimensions and instances for snapshots 1, 3 and 9. Those snapshots were chosen because the beginning and the end of the run of the CMA-ES algorithm were considered more crucial than the rest. Each of the trained models was then used to predict the function values of all sampled points in the population of next generation of those snapshots. Original function values were also evaluated for those sampled points and from the results, the mean and the 3rd quartile statistics of RDE and MSE were calculated.
Choosing the GpModels for ModelPool was done using the rde_ranking and mse_ranking scripts contained in the project, which do the following:

1. For each of the models, calculate the percentile of how many times was it successfully trained on the data of the used instances (result of the isTrained function) and omit the models whose percentile is less than *minTrainedPerc* parameter (set to 0.85).

2. Sort the models using the 3rd quartile of RDE, respectively MSE, for each function and save the rank of each of the models.

3. For each of the models, calculate in how many snapshots of each of the functions, that are not yet covered by any of the chosen models, was the rank of the model better than *maxRank* parameter (set to 25).

4. Choose the model that will cover most of the uncovered snapshots of the functions and repeat until all snapshots of the functions are covered. In case where the amount of the covered snapshots is the same for multiple models, choose the one with better average rank (calculated in 1).

This algorithm was used to determine which models will be used for each of the dimensions separately. Moreover, different models were chosen for noiseless and noisy functions according to the results of the experiments made for noiseless (exp_doubleEC_21_log15) and noisy (exp_doubleEC_21_log15_noisy) functions.

3.2 ModelPool

Different models which will be contained in the ModelPool are set by *modelOptions.parameterSets* struct in constructor. For the options that are not specified, default values are used, which are:

historyLength 4

minTrainedModelsPercentileForModelChoice 0.25

maxGenerationShiftForModelChoice 1

retrainPeriod 1

bestModelSelection rdeAll

Parameters, that are used in all GpModels with the same value ($predictionType, transformCoordinates, dimReduction$ and $options.normalizeY$) are set once in the options of ModelPool instead of setting them separately for each of the GpModels. An instance of the GpModel is created using the following function:

```
function gpModel = createGpModel(obj, modelIndex, xMean)
  newModelOptions = obj.modelPoolOptions.parameterSets(modelIndex);
  newModelOptions.predictionType = obj.predictionType;
  newModelOptions.transformCoordinates = obj.transformCoordinates;
  newModelOptions.dimReduction = obj.dimReduction;
  newModelOptions.options.normalizeY = obj.options.normalizeY;
  gpModel = ModelFactory.createModel('gp', newModelOptions, xMean);
end
```

As can be seen, the ModelFactory is called with 'gp' parameter, which means the GpModels are created. If there was a need for different types of models, this parameter would need to be extracted into the options. The ModelPool class was, however, designed with the focus on GpModels.
A cell array of size $modelsCount * (historyLength + 1)$ is created in the constructor and stored in the property $models$. It contains the chosen GpModels trained in the newest generation (accessed by $models.\{modelId, 1\}$, where $modelId$ defines which of the GpModels defined in $parameterSets$ is being accessed). A logical array $isModelTrained$ with the same size as models array is created, which is used to store the results of the isTrained method for each of the contained GpModels.
The overriding of the isTrained method can be seen here:

```
function trained = isTrained(obj)
  % check whether the model chosen as the best
  % in the newest generation is trained
  if (isempty(obj.isModelTrained(obj.bestModelIndex,1)))
    trained = false;
  else
    trained = obj.isModelTrained(obj.bestModelIndex,1);
  end
end
```

The trainModel function is not needed, because the training is done by overriding the train method, which then calls the train method of each of the GpModels. That is why is its body empty.

The train method, which handles the training of models, can be seen here:

```
function obj = train(obj, X, y, stateVariables, sampleOpts, ...
archive, population)
  obj.archive = archive;
  obj.stateVariables = stateVariables;
  obj.sampleOpts = sampleOpts;
  obj.xMean = stateVariables.xmean';
  generation = stateVariables.countiter;
  if (mod(generation,obj.retrainPeriod)==0)

    trainedModelsCount=0;
    for i=1:obj.modelsCount

      obj.models(i,:) = circshift(obj.models(i,:),[0,1]);
      obj.isModelTrained(i,:) = ...
              circshift(obj.isModelTrained(i,:),[0,1]);
      obj.models{i,1} = obj.createGpModel(i, obj.xMean);
      obj.models{i,1} = obj.models{i,1}.train(X, y, ...
              stateVariables, sampleOpts, obj.archive, population);

      if (obj.models{i,1}.isTrained())
        trainedModelsCount = trainedModelsCount+1;
        obj.isModelTrained(i,1) = 1;
      else
        obj.isModelTrained(i,1) = 0;
        obj.models{i,1}.trainGeneration = -1;
      end
    end

    if (trainedModelsCount==0)
      warning('ModelPool.trainModel(): trainedModelsCount == 0');
    else
      obj.trainGeneration = generation;

      [obj.bestModelIndex,obj.choosingCriterium] = ...
                  obj.chooseBestModel(generation, population);
      obj.bestModelsHistory(obj.bestModelIndex) = ...
                  obj.bestModelsHistory(obj.bestModelIndex)+1;
      obj = obj.copyPropertiesFromBestModel();
    end
  end
end
```

In the beginning, properties of the Model class are stored from the passed parameters. Then, models are trained in a cycle. The shifting of the models is done by using built-in Matlab function circshift. Oldest of the models, which is in the *obj.models*$\{i, 1\}$ cell after the shifting, is then rewritten by the model created in the current generation. Finally, best model is chosen using

the specified criterium and properties of the best GpModel are copied to the ModelPool, so that it behaves as the best GpModel to the outside callers.
The prediction is simple - override the modelPredict function with call of the modelPredict of the best GpModel of the newest generation:

```
function [y, sd2] = modelPredict(obj, X)
  [y,sd2] = obj.models{obj.bestModelIndex,1}.modelPredict(X);
end
```

3.3 Choosing the best parameters for the ModelPool

Two different types of errors were used in choosing the GpModels for the ModelPool - the RDE and the MSE. That is why the parameters of the ModelPool were chosen separately for those errors. The same approach for choosing the parameters for GpModels that will be used in the ModelPool was used to find the best parameters for the ModelPool in each of the dimensions. Noisy and noiseless functions were also differentiated. That creates a combination of 12 experiments - 3 dimensions, noiseless or noisy and RDE or MSE. Two different options for the bestModelSelection parameter for the RDE are available ("rdeOrig" and "rdeAll"), so both were used. Other parameters, that were tested, were the same for RDE and MSE:

historyLength 3, 5, 7,

minTrainedModelsPercentileForModelChoice 0.25, 0.5,

maxGenerationShiftForModelChoice 0, 2.

That creates 24 combinations of parameters for the RDE and 12 combinations for the MSE. The retrainPeriod parameter was always set to 1.
Because of the changes between usage of the GpModel and the ModelPool classes, the testing scripts needed to be slightly modified:

1. Creation of the dataset was changed to include up to n previous generations for each of the snapshots ($n = 8$ in this case). That is needed because the previous training is important in the ModelPool class and it affects the selection of the best model, which then affects the prediction.

2. Training of the model needed to be done multiple times with corresponding generation.

3. Snapshot 1 was not part of the training and testing, as there were multiple functions where this snapshot did not have enough previous gener-

ations to fill the whole *models* array and therefore the algorithm for the selection of the best model was not used (instead, likelihood was used).

4. Only 5 instead of 15 instances (with IDs 1 to 5) for all functions were tested to decrease needed computational time, which was increased by using multiple GpModels.

Results of each of the experiments will be described in chapter 4.

3.4 Testing

Testing of the GpModel training as well as ModelPool class was done mostly manually, with tools provided by the Matlab software, such as debugger, console, etc. An automatic test was written to ensure proper creation of the ModelPool class, which can be found in
"surrogate-cmaes/test/unit/model/modelPoolTest.m". Moreover, testing of the ModelPool class was done using an additional experiment
(exp_doubleEC_23_pool) with the same GpModel from exp_doubleEC_23 (ensured by using trainsetType 'parameters') with 2 additional GpModels chosen by the author. This experiment was expected to give similiar results as the exp_doubleEC_23. Results of this experiment, as well as others, will be described in details in chapter 4.

Chapter **4**

Experiments

All of the experiments were divided between 2 categories - noiseless and noisy functions. In the following text, multiple experiments are being compared. They are always given colors that depend on the order of the experiment in the report. The names of experiments are shortened in the legends - ”exp_doubleEC_” is removed from the beginning of the name.

4.1 Noiseless functions

4.1.1 Experiments exp_doubleEC_23, exp_doubleEC_23_pool

An experiment which had the best results so far in this project was exp_doubleEC_23. This experiment had only 1 GpModel, whose settings could be interperted as the ones defined in Table C.1. It has set the base performance, which was ment to be increased by using the ModelPool class. The trainRange was set in a different when the GpModel was defined by settings used before the changes described in 2.3.
Results of this experiment can be found in Figures 4.1 to 4.9 and this experiment was given the green color.
For the testing of the ModelPool class, experiment exp_doubleEC_23_pool was created. GpModels that were used can be found in the Table C.2 and the options for the ModelPool were:

historyLength 6

bestModelSelection ’rdeAll’

retrainPeriod 1

minTrainedModelsPercentileForModelChoice 0.25

maxGenerationShiftForModelChoice 1

These settings have been chosen by the author. The experiment was run only in 2D and 5D to save the CPU time. Its results were expected to be similiar to the base experiment exp_doubleEC_23, as the first GpModel was the same. The results were actually better than the results of the base experiment and can be found in Figures 4.1 to 4.9. This experiment was given the cyan color. When compared by each function to the exp_doubleEC_23, most visible improvements can be seen in following functions:

2D: f12, f16, f21, f23, f24,

5D: f12, f16, f23.

This proved several statements:

- GpModels work as expected - different GpModels give different results.
- The ModelPool class works as expected - in this case, the ModelPool was successful in choosing appropriate GpModels where it was beneficial.
- The usage of the ModelPool class can improve the results of optimization.

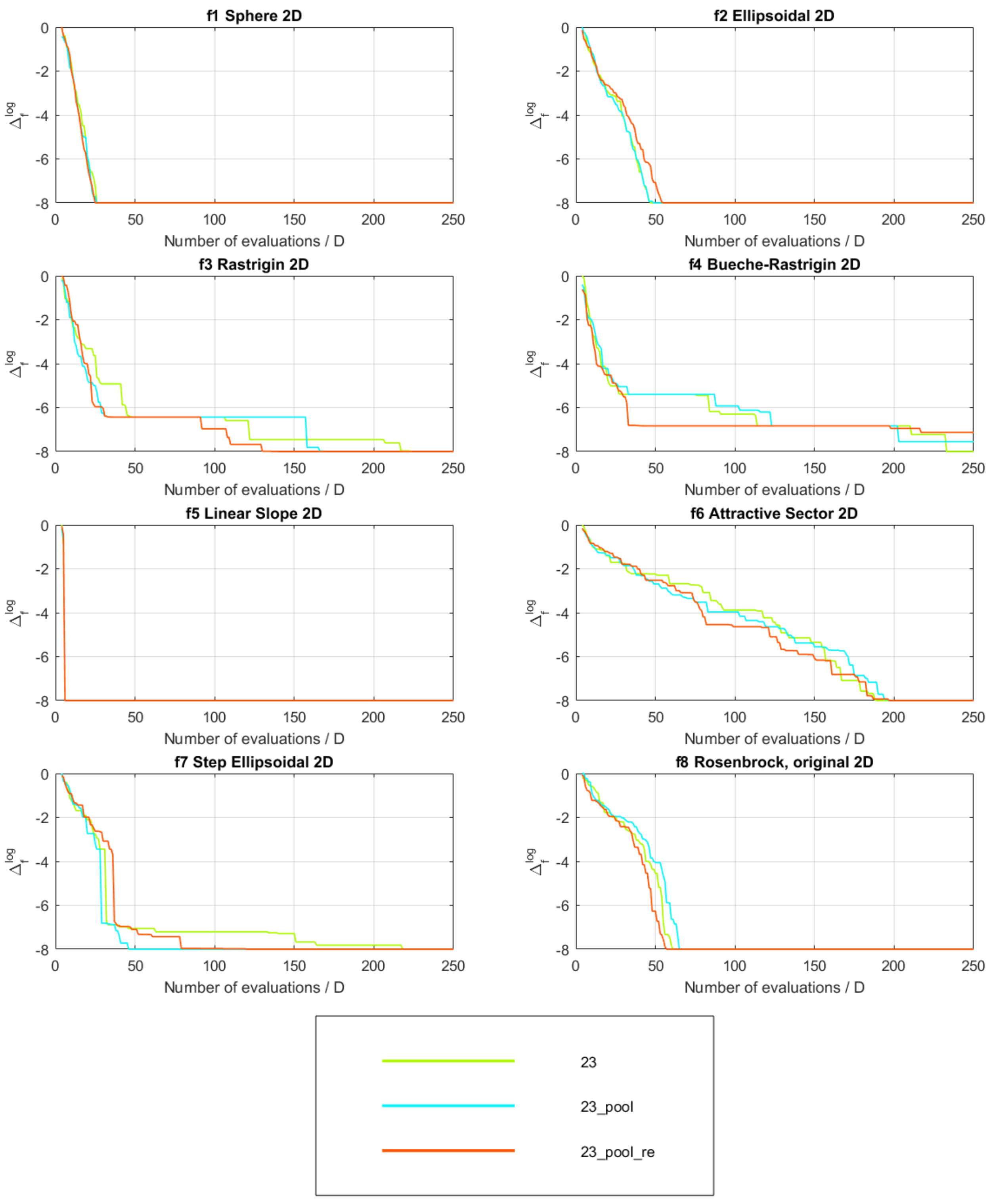


Figure 4.1: Experiments exp_doubleEC_23, exp_doubleEC_23_pool and exp_doubleEC_23_pool_re, 2D, f1 to f8.

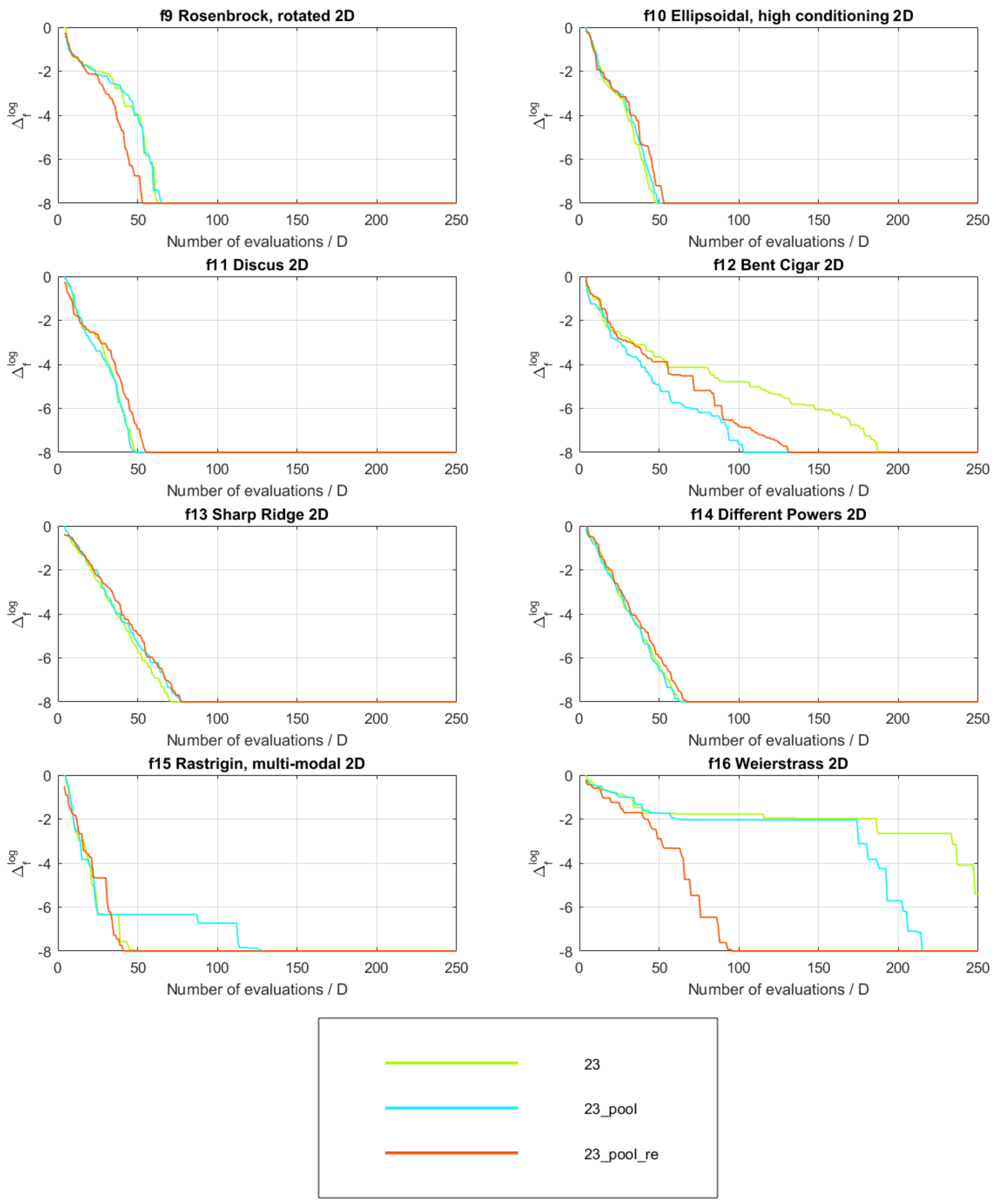


Figure 4.2: Experiments exp_doubleEC_23, exp_doubleEC_23_pool and exp_doubleEC_23_pool_re, 2D, f9 to f16.

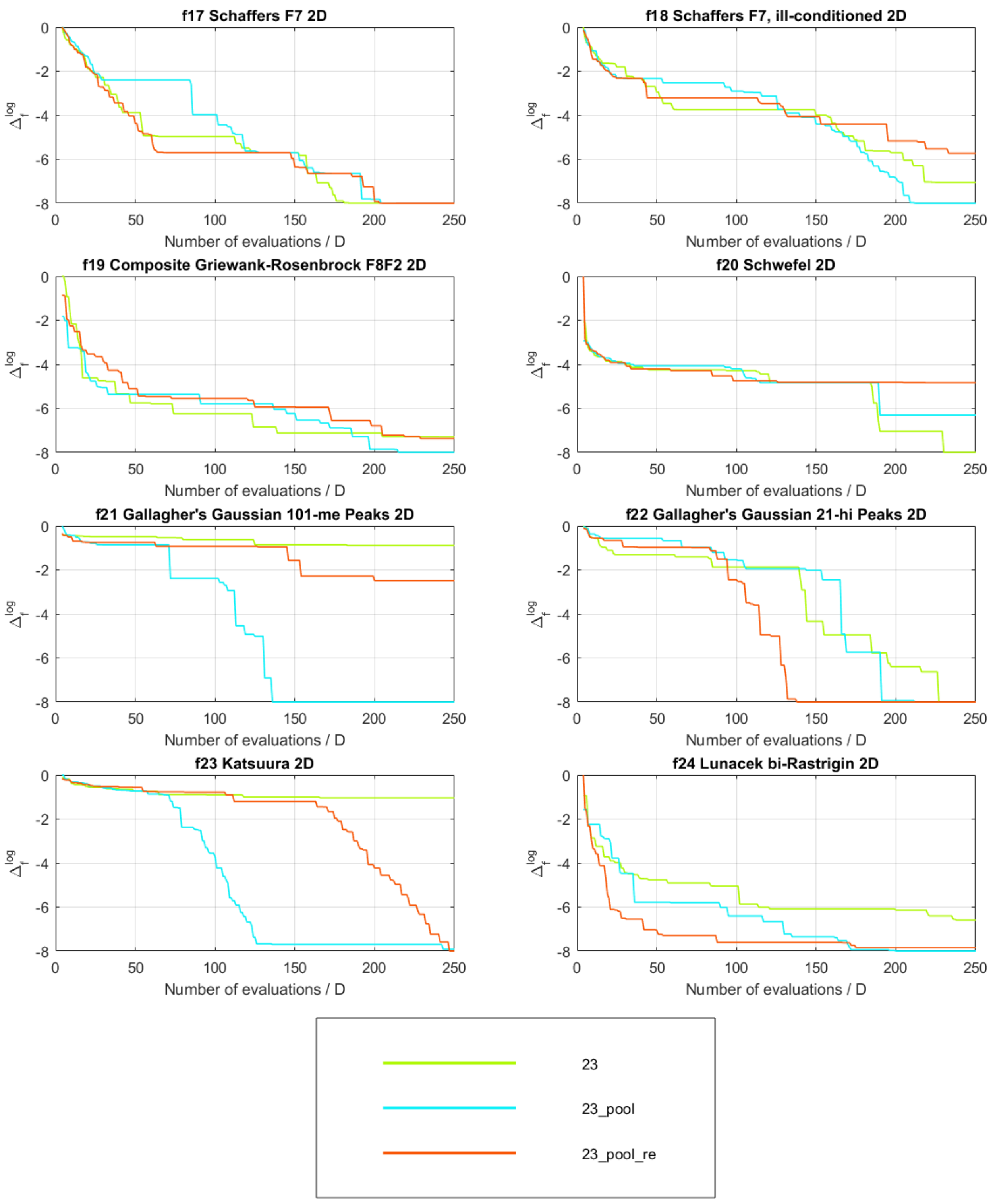


Figure 4.3: Experiments exp_doubleEC_23, exp_doubleEC_23_pool and exp_doubleEC_23_pool_re, 2D, f17 to f24.

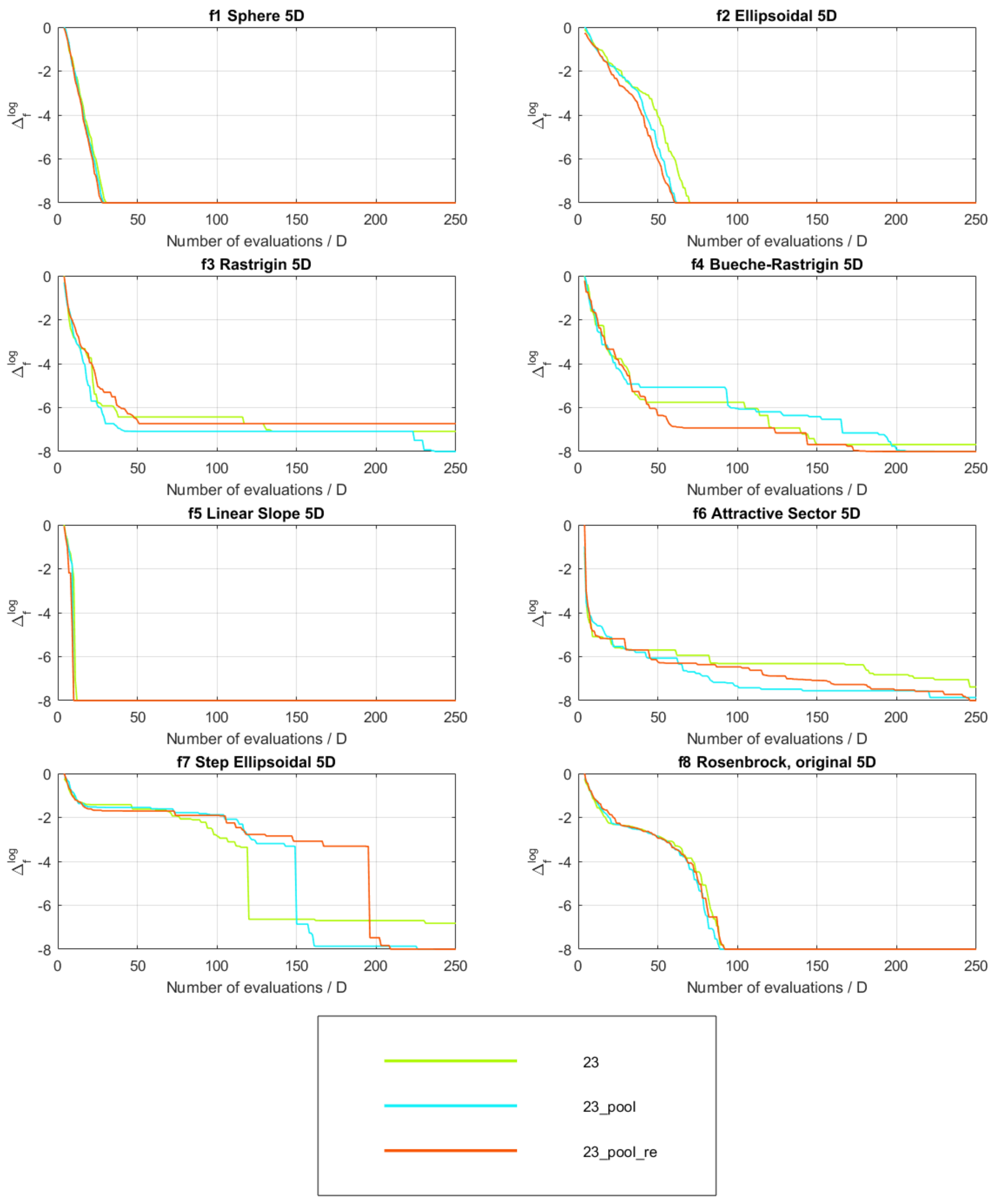


Figure 4.4: Experiments exp_doubleEC_23, exp_doubleEC_23_pool and exp_doubleEC_23_pool_re, 5D, f1 to f8.

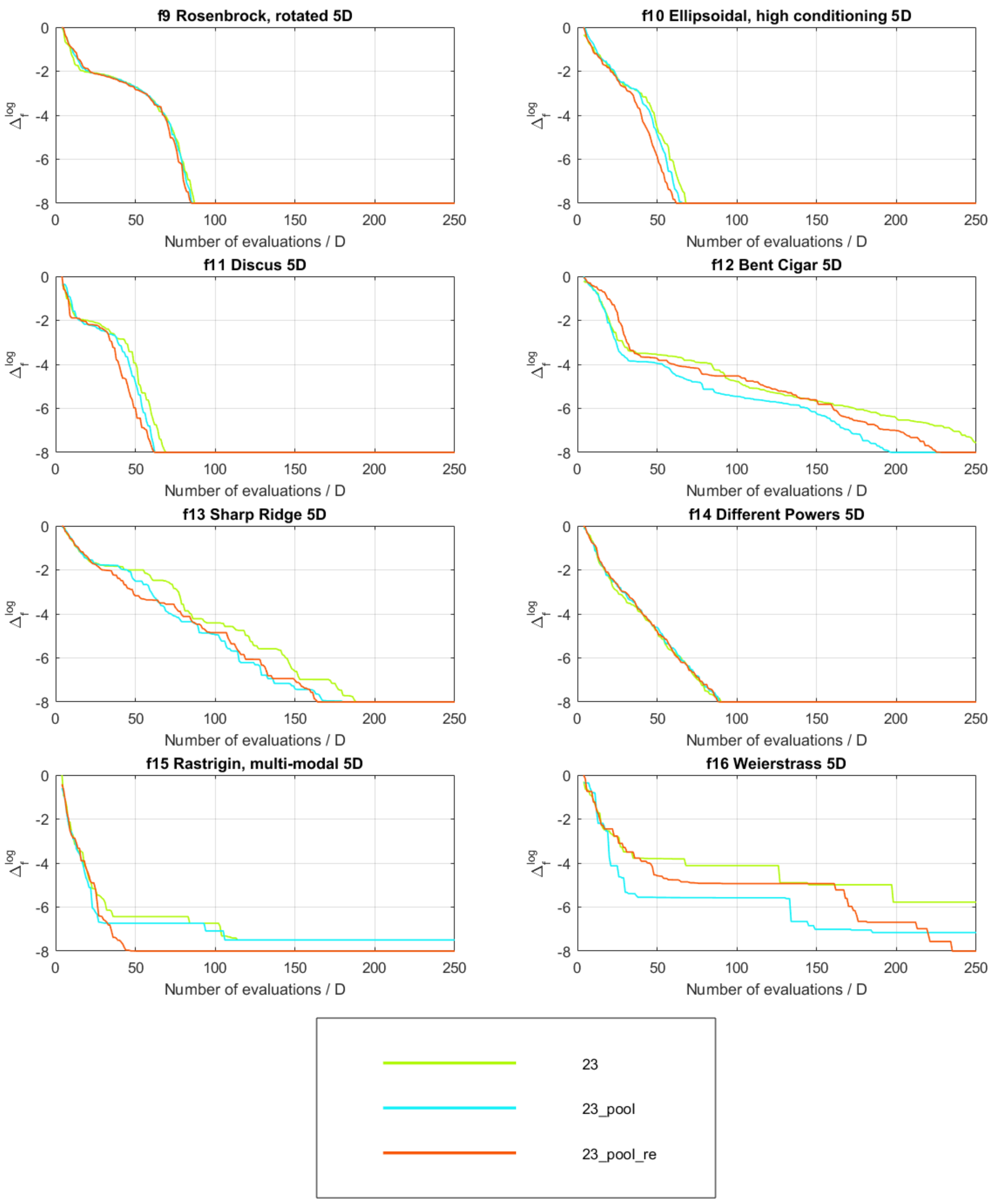


Figure 4.5: Experiments exp_doubleEC_23, exp_doubleEC_23_pool and exp_doubleEC_23_pool_re, 5D, f9 to f16.

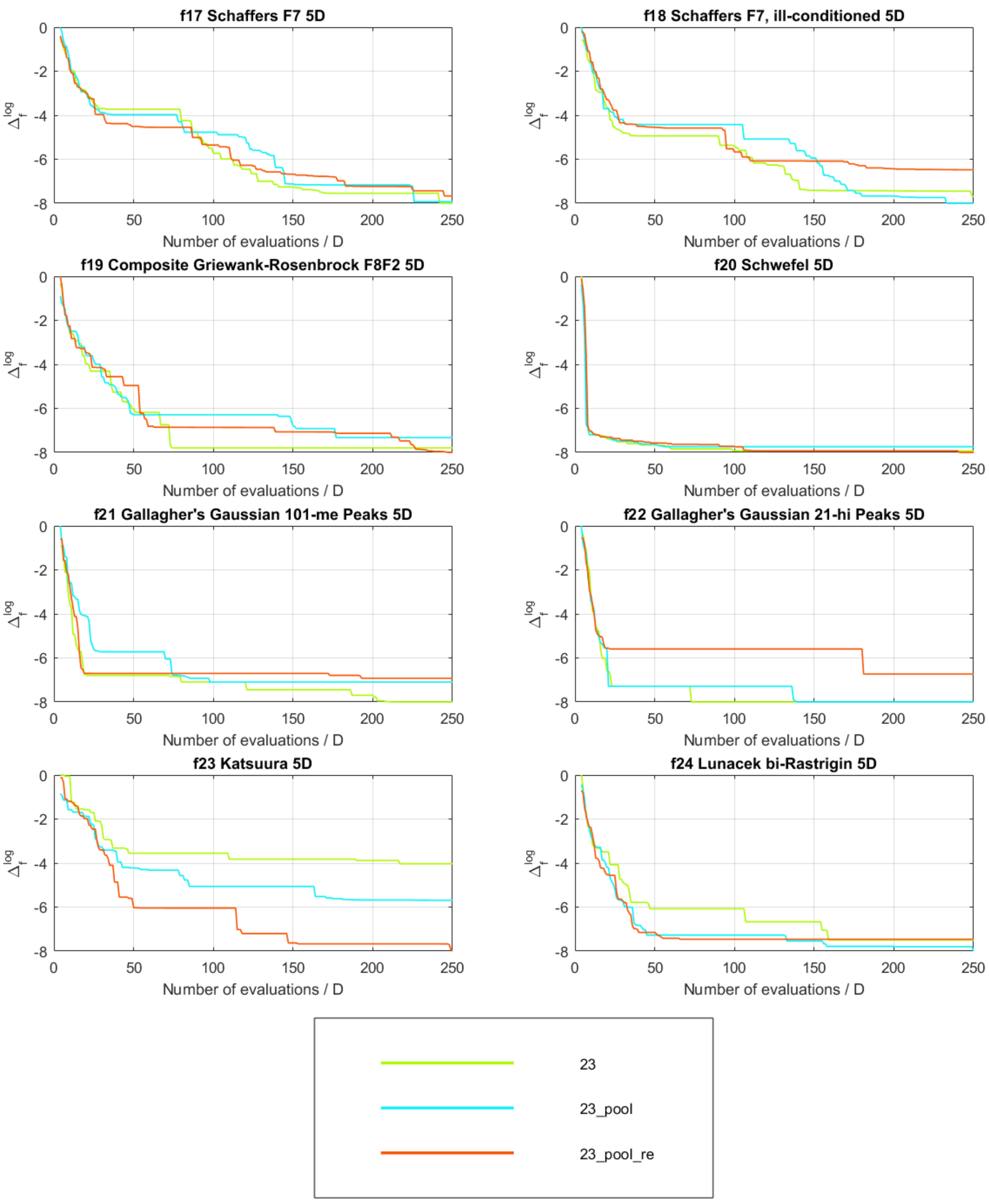


Figure 4.6: Experiments exp_doubleEC_23, exp_doubleEC_23_pool and exp_doubleEC_23_pool_re, 5D, f17 to f24.

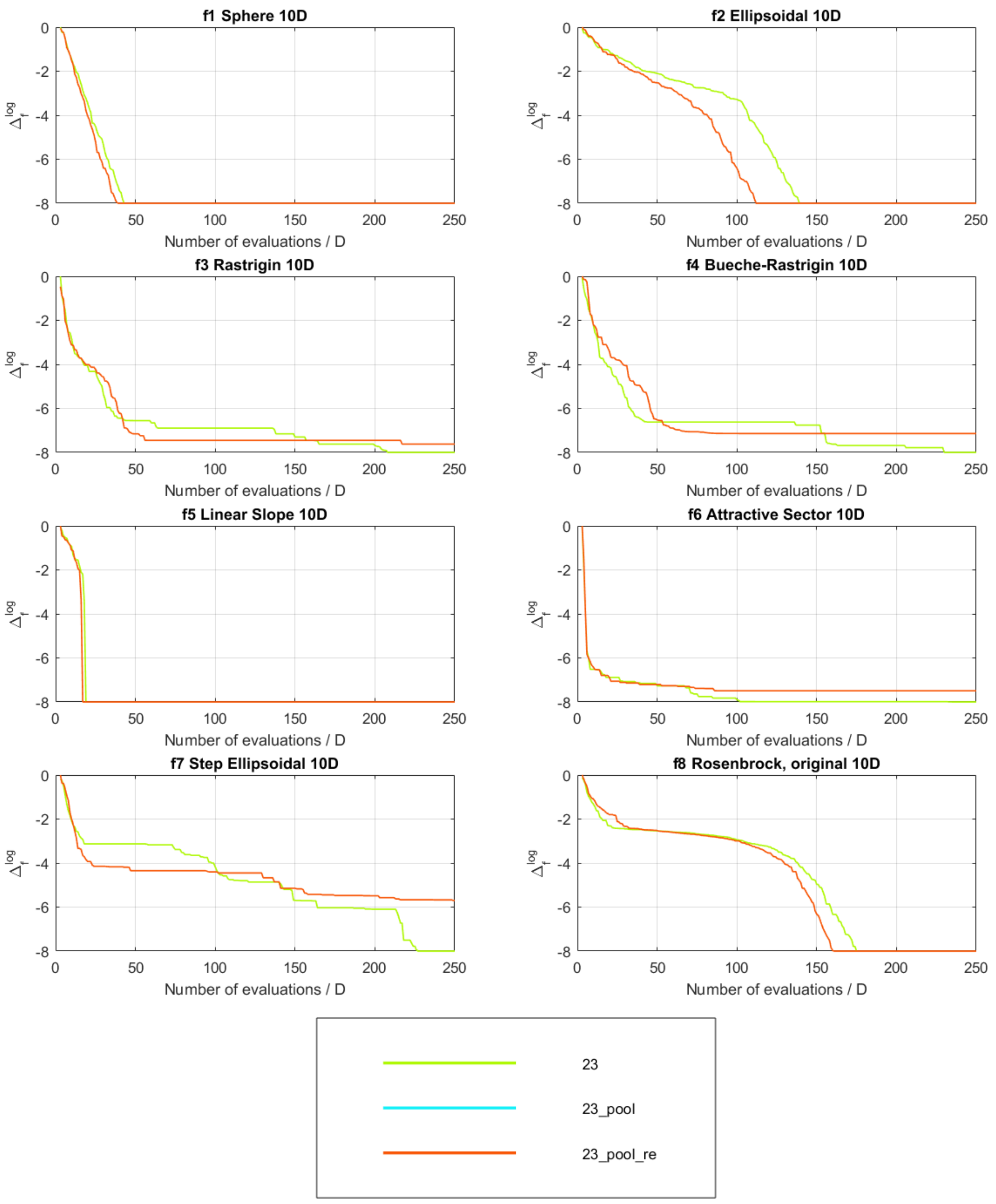


Figure 4.7: Experiments exp_doubleEC_23 and exp_doubleEC_23_pool_re, 10D, f1 to f8.

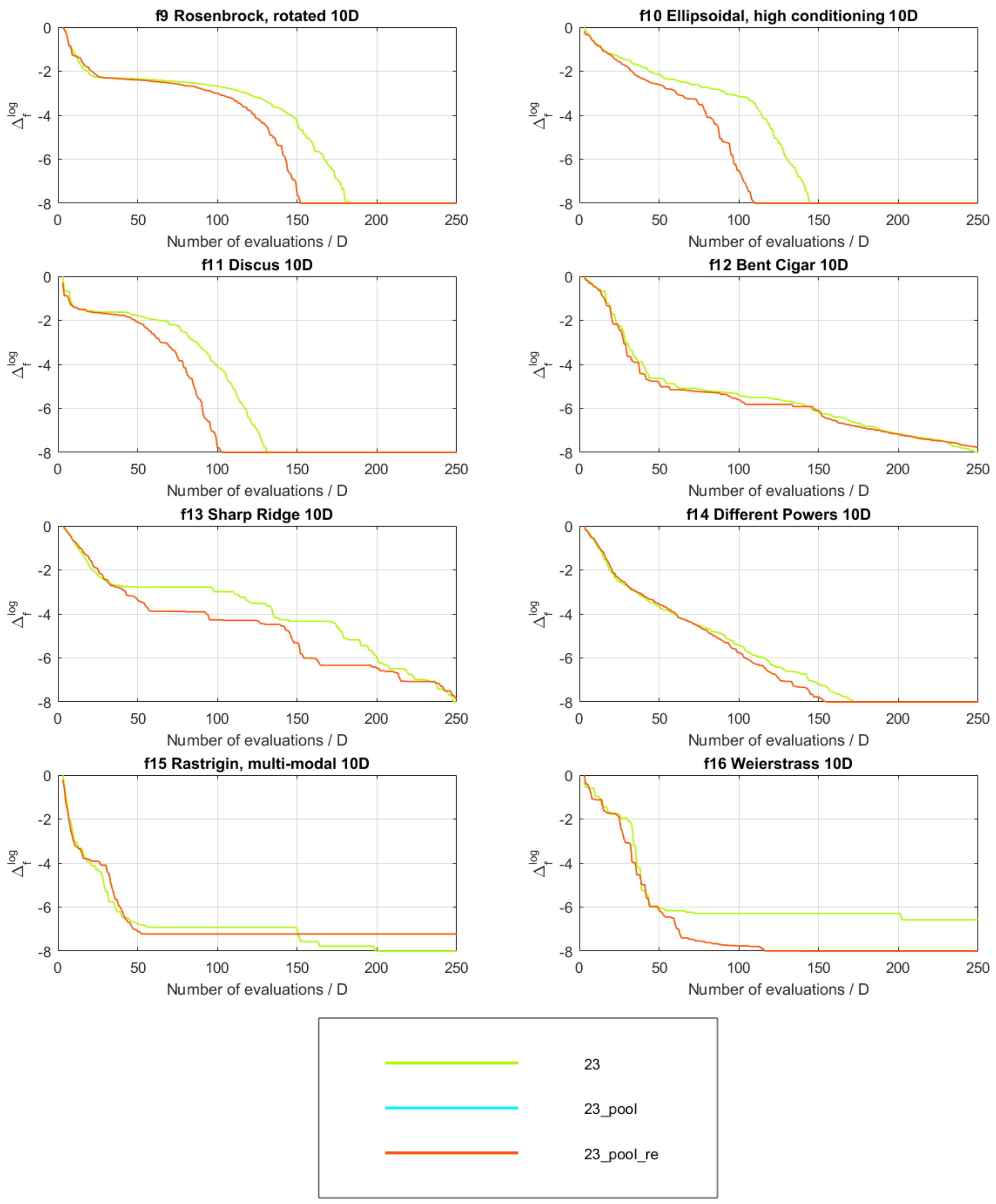


Figure 4.8: Experiments exp_doubleEC_23 and exp_doubleEC_23_pool_re, 10D, f9 to f16.

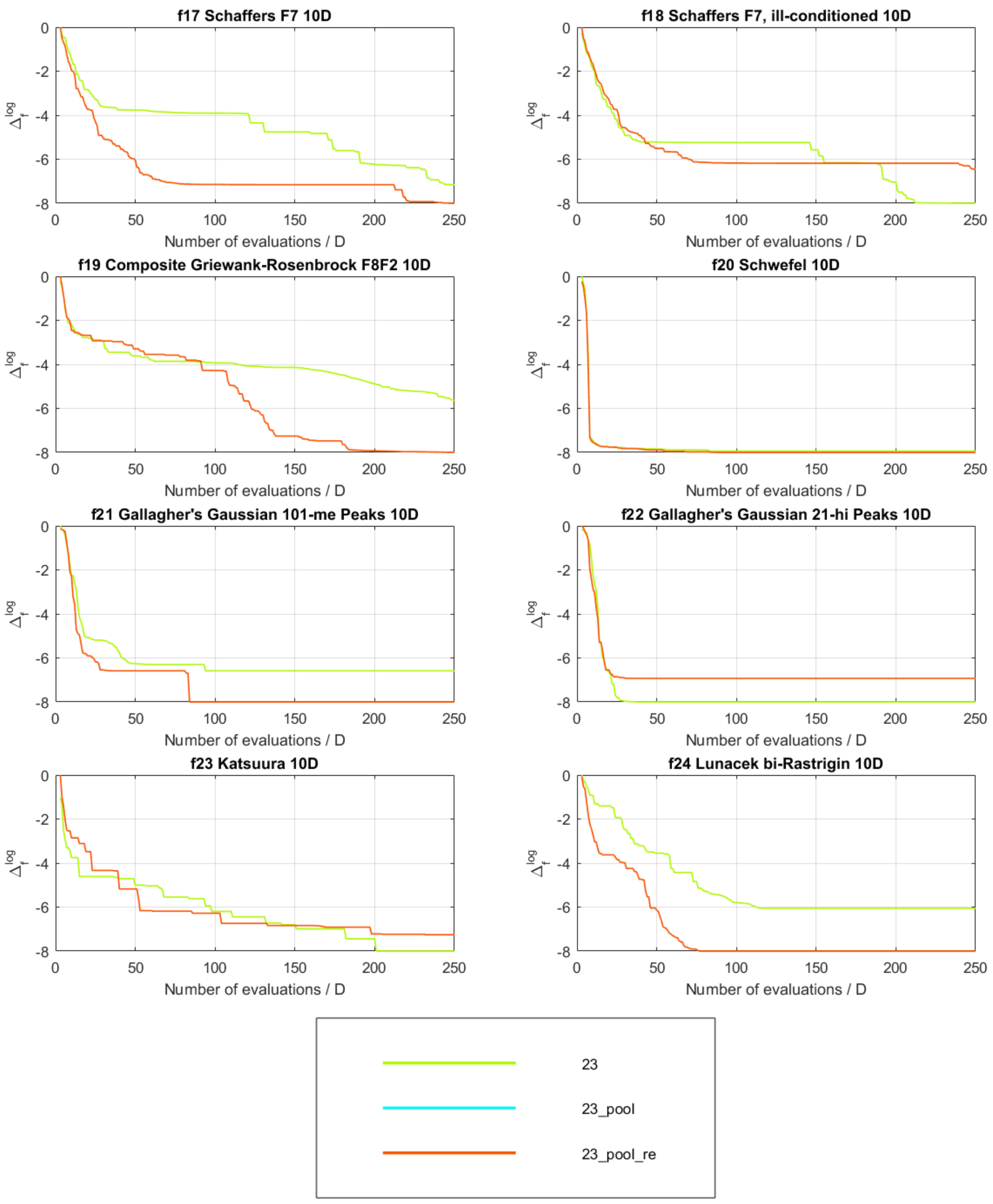


Figure 4.9: Experiments exp_doubleEC_23 and exp_doubleEC_23_pool_re, 10D, f17 to f24.

4.1.2 Experiment exp_GPtest_01 for choosing the GpModels

The process of choosing the GpModels was described in 3.1. Different GpModels have been chosen for different criteria (RDE and MSE), as well as for different dimensions. The chosen models can be found in different Tables shown here:

Dimension	Criterium	Table
2	RDE	C.4
5	RDE	C.5
10	RDE	C.6
2	MSE	C.7
5	MSE	C.8
10	MSE	C.9

Table 4.1: Table of the references to the chosen GpModels from exp_GPtest_01.

4.1.3 Experiments exp_MPtest_01_rde to exp_MPtest_06_mse

As the GpModels were chosen, experiments, whose targets were to choose optimal values of the parameters for the ModelPool, were started. The chosen parameters [1] can be found in following Table:

Dim.	Criterium	bestModelSel.	historyL.	minTr.Perc.	maxGen.Shift
2	RDE	'rdeAll'	7	0.25	2
5	RDE	'rdeAll'	7	0.5	0
10	RDE	'rdeOrig'	5	0.25	2
2	MSE	'mse'	3	0.25	0
5	MSE	'mse'	3	0.25	1
10	MSE	'mse'	3	0.5	1

Table 4.2: ModelPool parameters chosen by experiments exp_MPtest_01_rde to exp_MPtest_06_mse.

4.1.4 Experiments exp_doubleEC_MP_01 to exp_doubleEC_MP_06

After choosing the best ModelPool parameters for each of the cases, corresponding experiments (exp_doubleEC_MP_01 to exp_doubleEC_MP_06) were created, calculated and examined. The results can be found in Figures 4.11 to

[1] The ModelPool class automatically decrease the maxGenerationShift to from 2 to 1 when historyLenght is set to 3 to ensure enought points for the best model selection.

4.18. Both RDE (cyan) and MSE (red) results are shown together for better comparison.

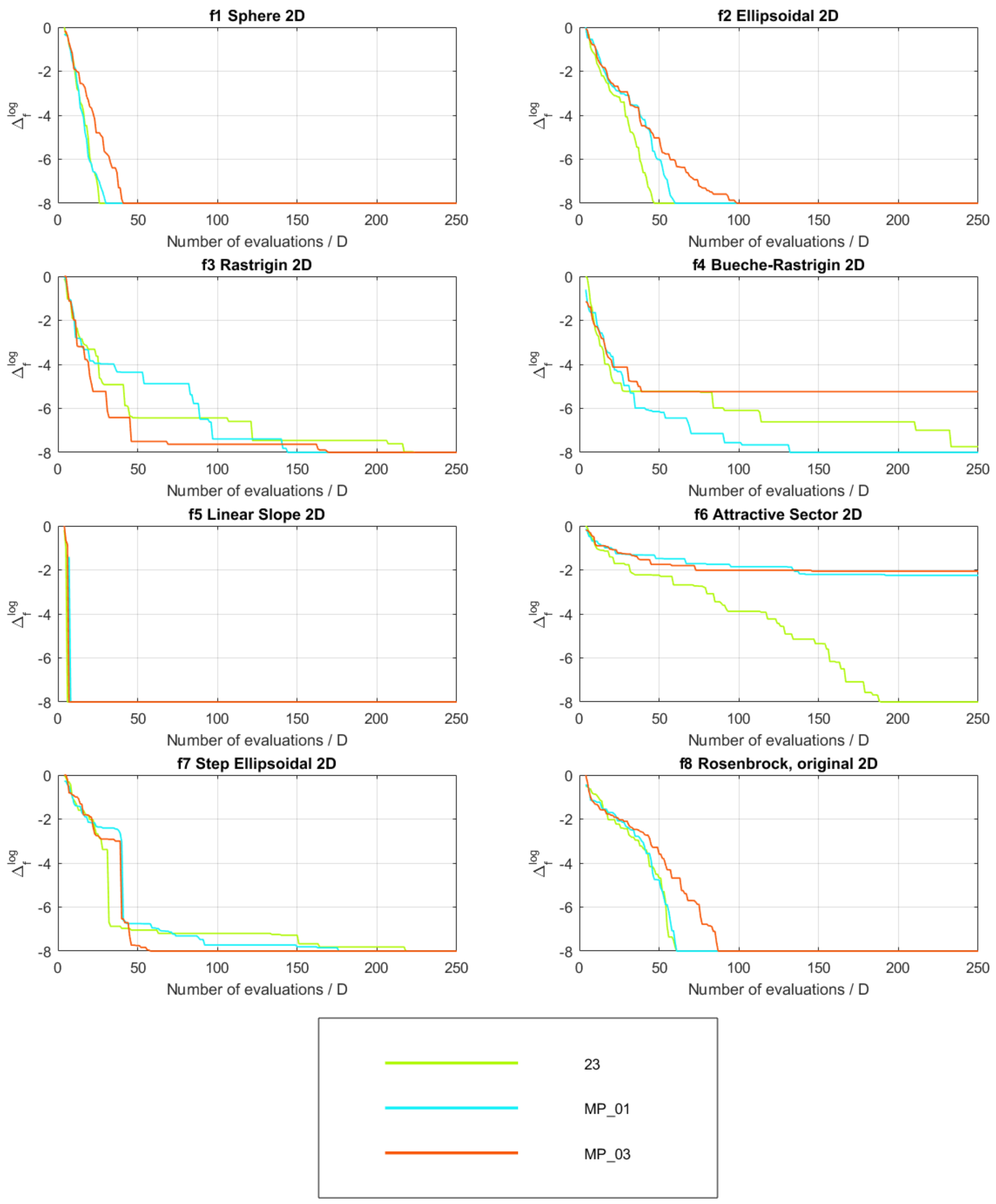


Figure 4.10: Experiments exp_doubleEC_23, exp_doubleEC_MP_01 (RDE) and exp_doubleEC_MP_03 (MSE), 2D, f1 to f8.

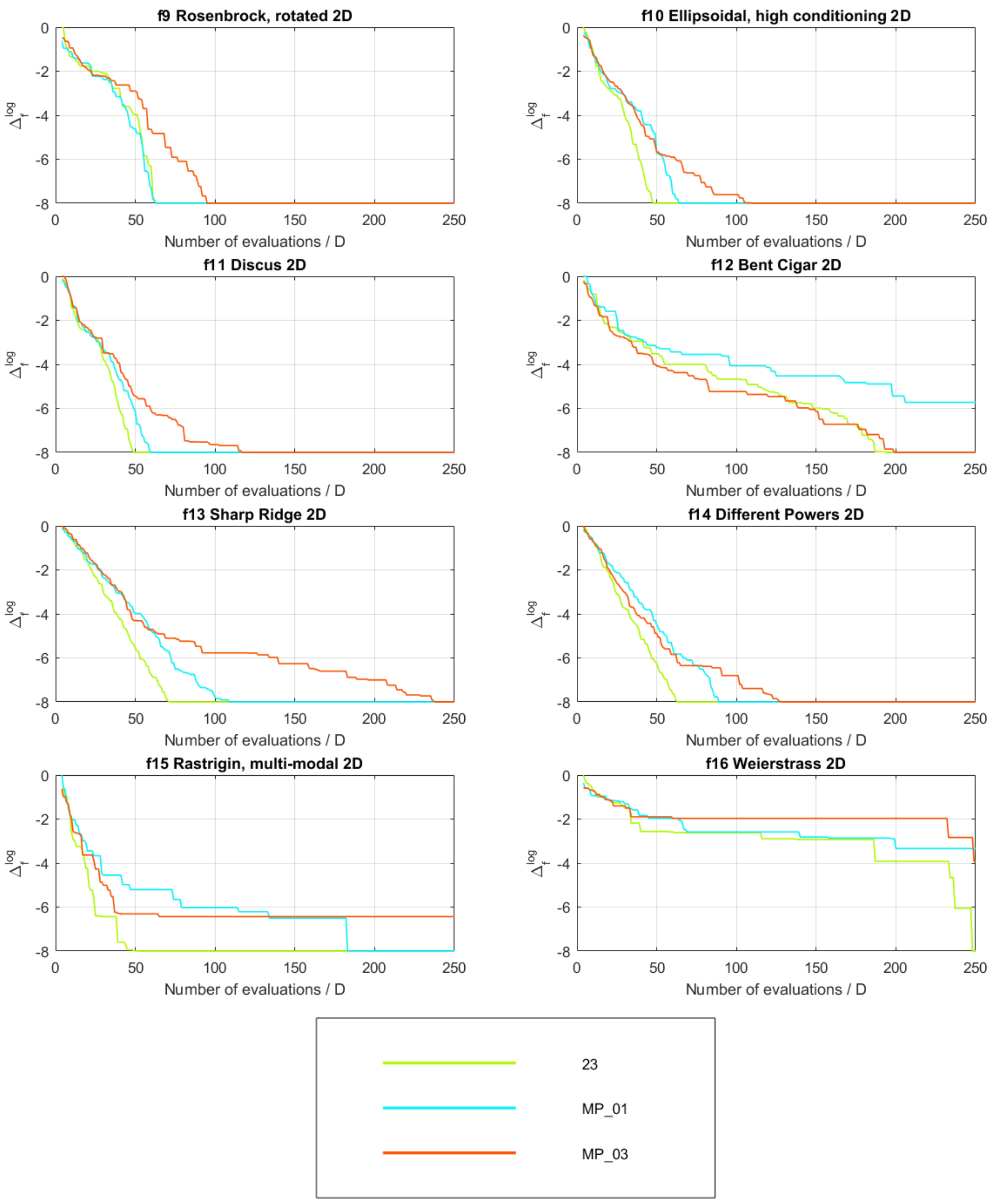


Figure 4.11: Experiments exp_doubleEC_23, exp_doubleEC_MP_01 (RDE) and exp_doubleEC_MP_03 (MSE), 2D, f9 to f16.

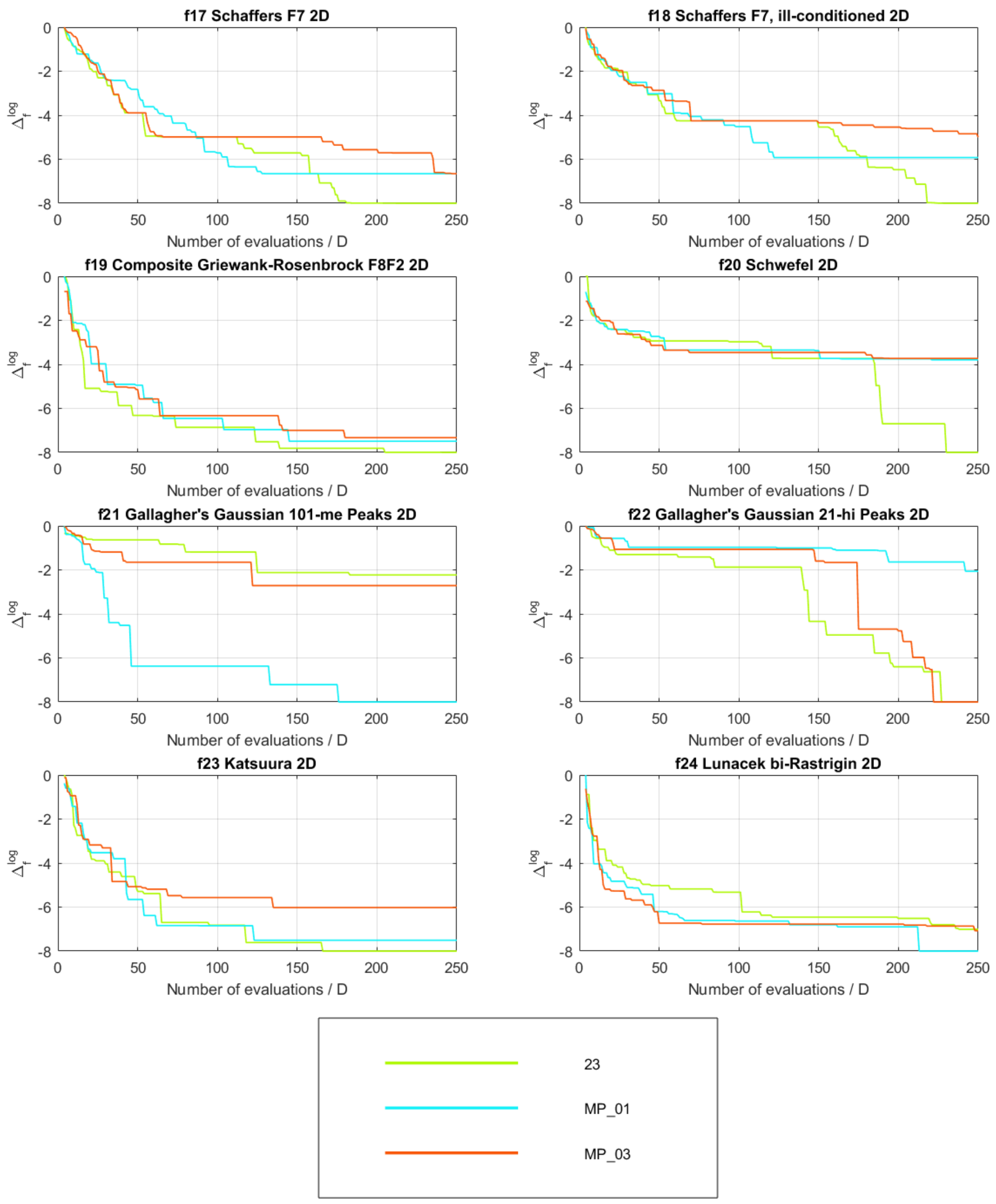


Figure 4.12: Experiments exp_doubleEC_23, exp_doubleEC_MP_01 (RDE) and exp_doubleEC_MP_03 (MSE), 2D, f17 to f24.

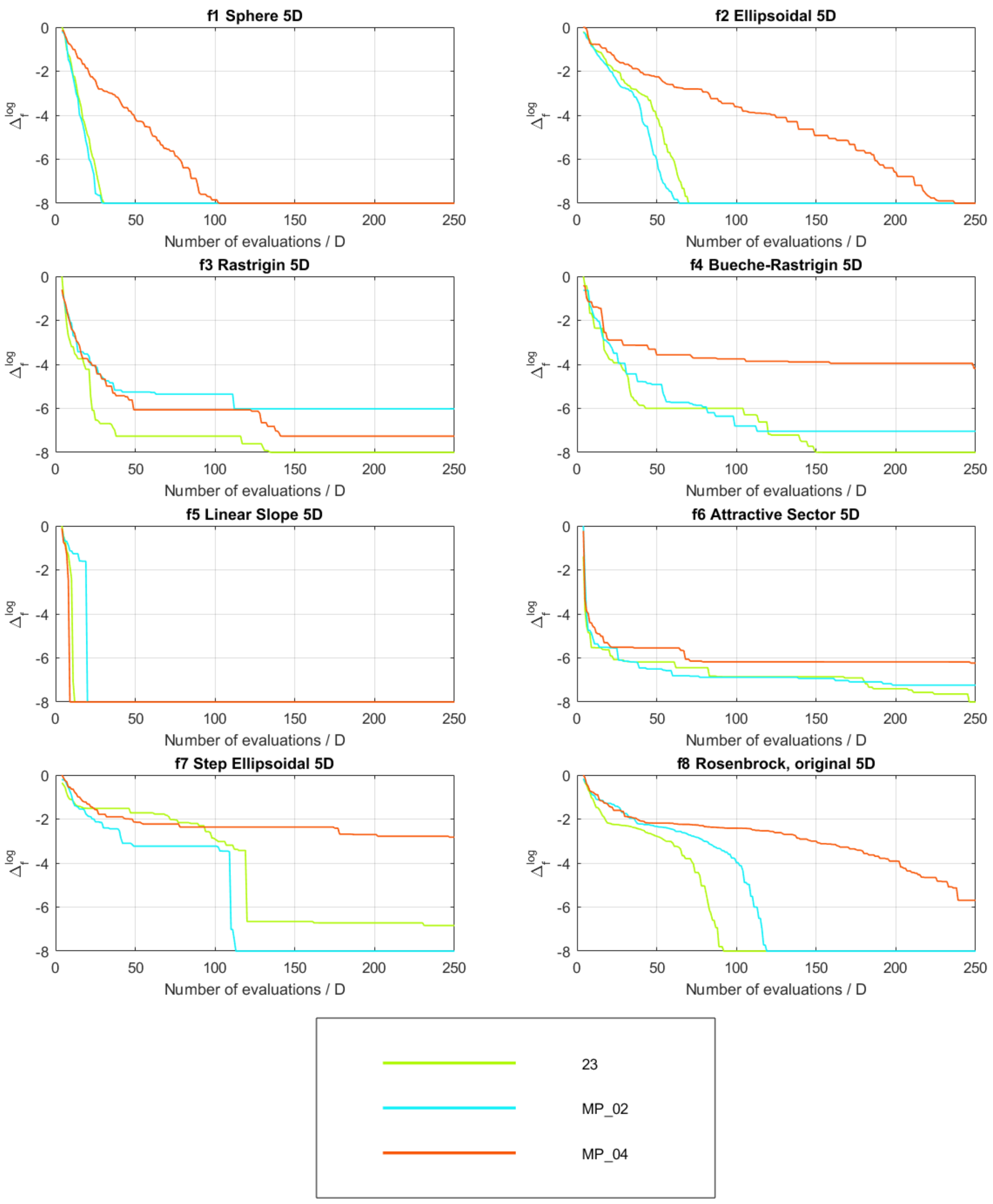


Figure 4.13: Experiments exp_doubleEC_23, exp_doubleEC_MP_02 (RDE) and exp_doubleEC_MP_04 (MSE), 5D, f1 to f8.

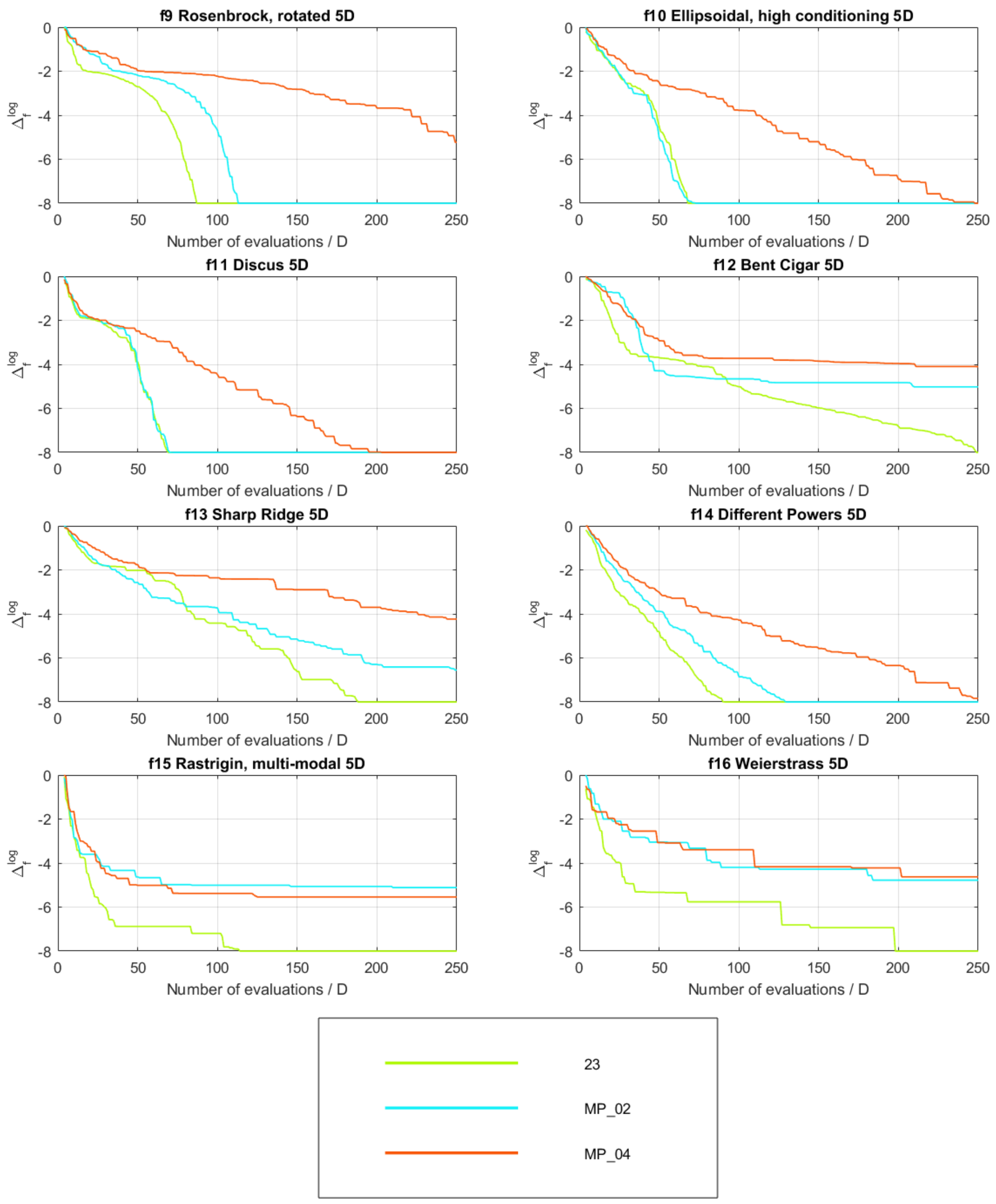


Figure 4.14: Experiments exp_doubleEC_23, exp_doubleEC_MP_02 (RDE) and exp_doubleEC_MP_04 (MSE), 5D, f9 to f16.

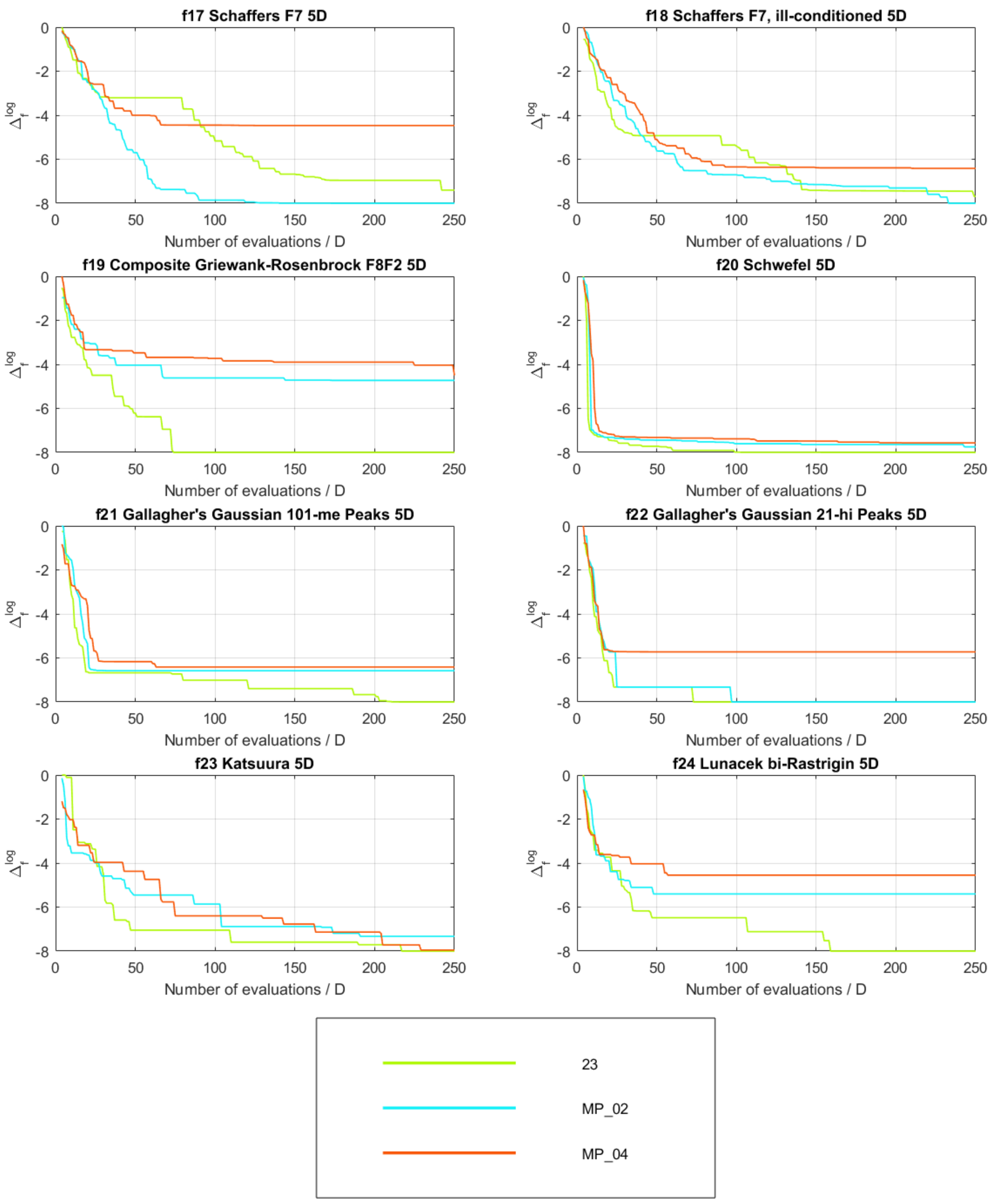


Figure 4.15: Experiments exp_doubleEC_23, exp_doubleEC_MP_02 (RDE) and exp_doubleEC_MP_04 (MSE), 5D, f17 to f24.

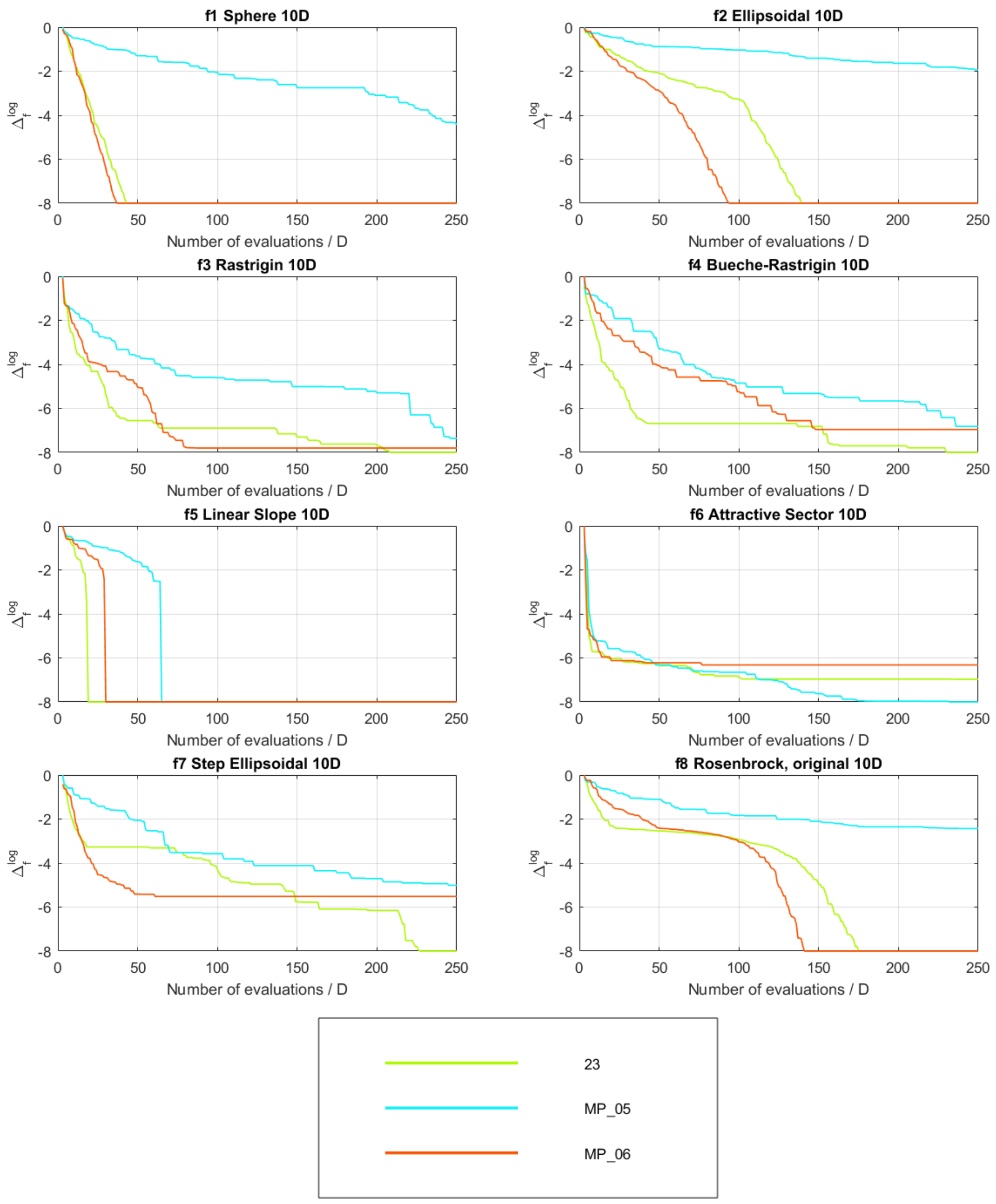


Figure 4.16: Experiments exp_doubleEC_23, exp_doubleEC_MP_05 (RDE) and exp_doubleEC_MP_06 (MSE), 10D, f1 to f8.

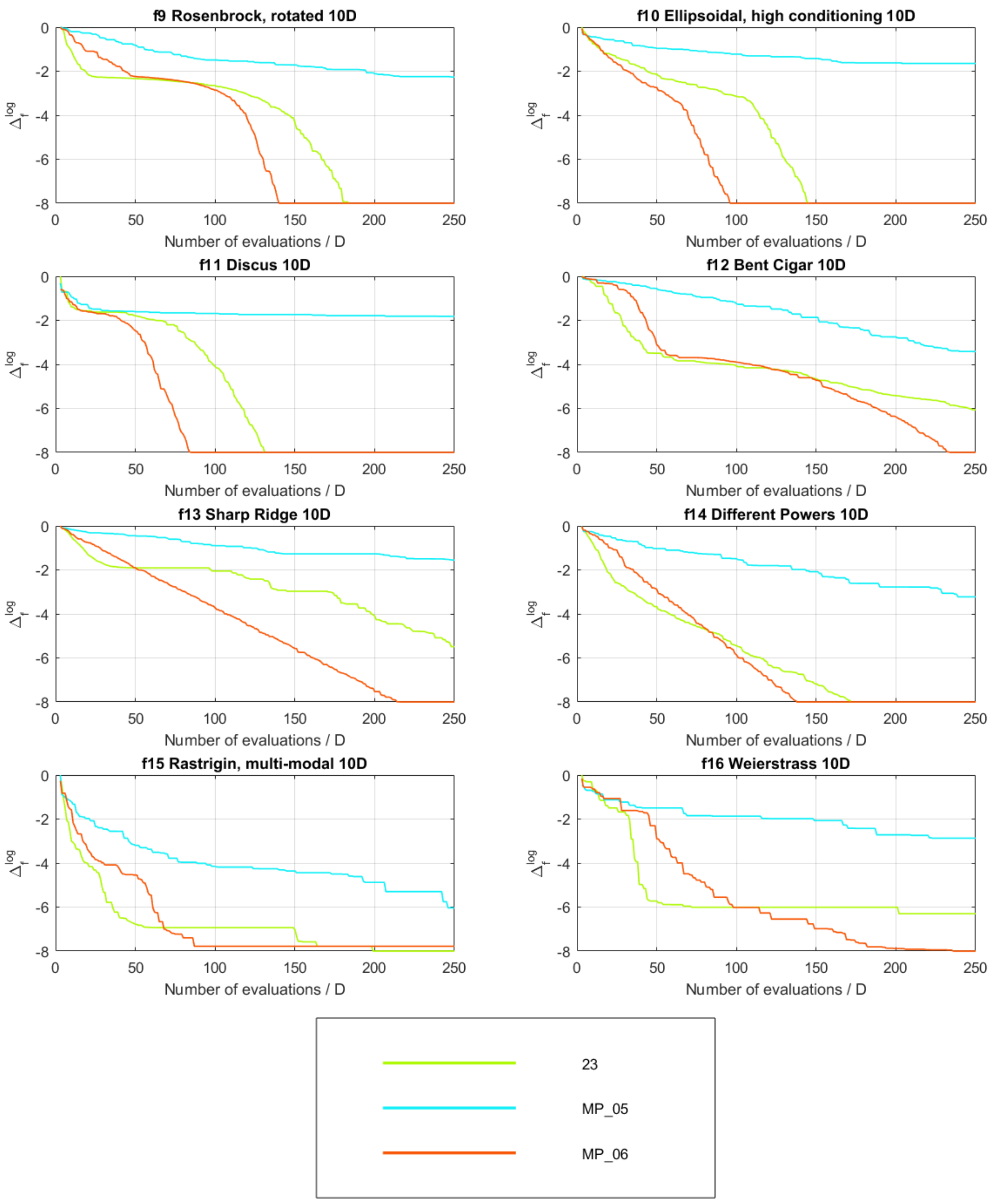


Figure 4.17: Experiments exp_doubleEC_23, exp_doubleEC_MP_05 (RDE) and exp_doubleEC_MP_06 (MSE), 10D, f9 to f16.

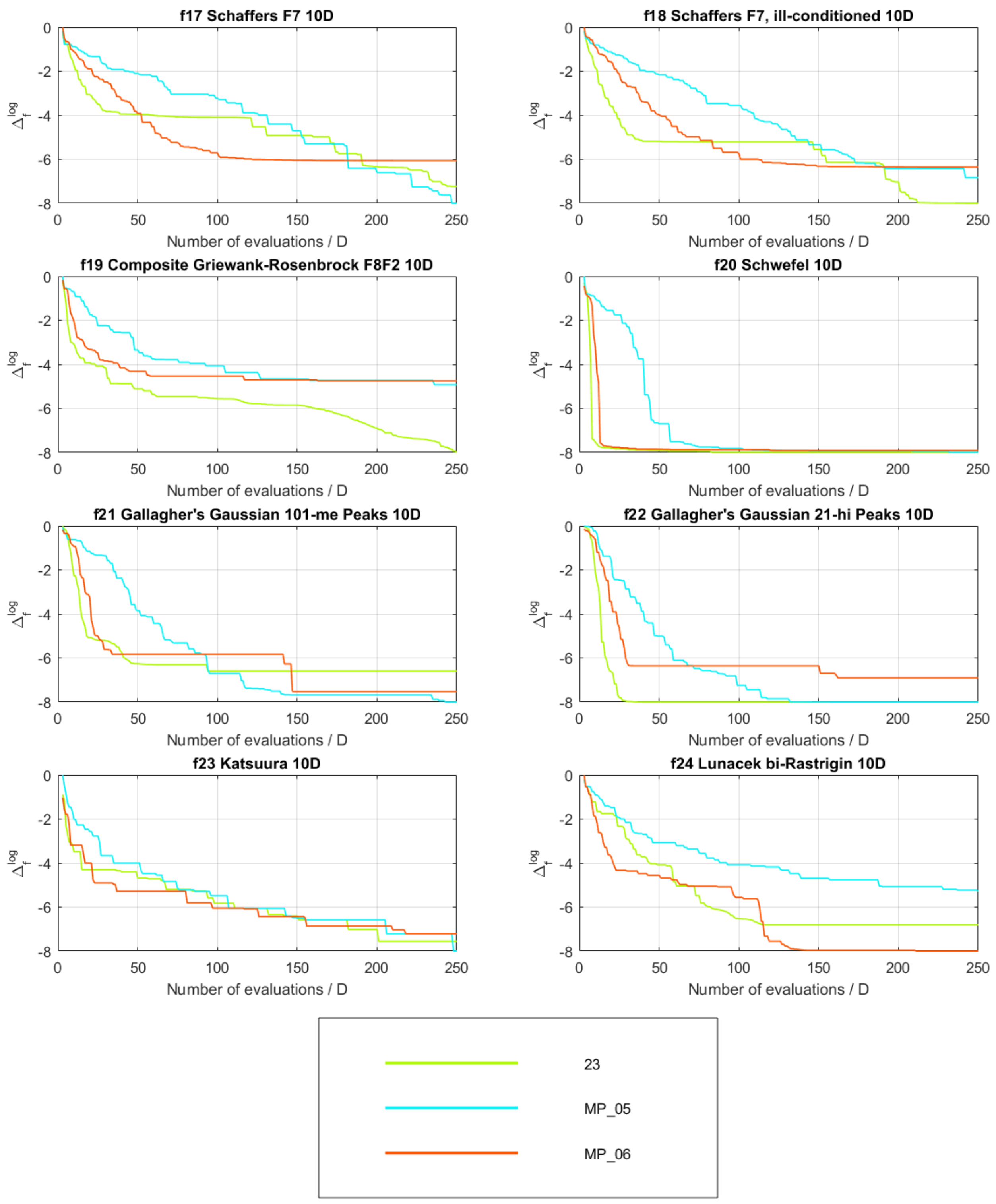


Figure 4.18: Experiments exp_doubleEC_23, exp_doubleEC_MP_05 (RDE) and exp_doubleEC_MP_06 (MSE), 10D, f17 to f24.

One of the conclusions of these experiments was that the MSE caused worse performance than RDE in 2D and 5D. That is the reason why MSE was not considered in further experiments. In 10D, where 'rdeOrig' option was chosen, results were also worse than those with MSE option. The problem in comparing 10D was that the experiment exp_doubleEC_23_pool was run only in 2D and 5D and therefore no results were available for 10D 'rdeAll' option. That is why the experiment exp_doubleEC_23_pool was run again as exp_doubleEC_23_pool_re in all 3 dimensions. The results can be found in Figures 4.1 to 4.9 and this experiment was given the red color. As expected from the 2D and 5D dimensions, 'rdeAll' was the best of the three options for the selection of the best model in 10D as well and that is why the 'rdeOrig' option was no longer considered in the next experiments.

4.1.5 Reasons of a failure of the experiments exp_doubleEC_MP_01 and exp_doubleEC_MP_03

Comparing the results of the experiments that were run so far created an important task - finding a reason of decrease of the performance in experiments exp_doubleEC_MP_01 and exp_doubleEC_MP_03 (both with 'rdeAll' option) when compared to experiments exp_doubleEC_23 (base experiment with 1 GpModel) and exp_doubleEC_23_pool (also with 'rdeAll' option). The last set of experiments had the GpModels chosen by the rde_ranking script from all of the 256 combinations. On the other hand, exp_doubleEC_23_pool run with GpModels that had the meanFcn option set only to 'meanConst'. The count of the GpModels used was also lower. Another experiment (exp_doubleEC_25) was created to find out if any of these differences could be the reason of the failure. The number of GpModels was decreased to 4 and the chosen GpModels with 'meanConst' mean function can be found in Table C.16. To lower the CPU time needed for the experiment, only 10 functions (f2, f4, f6, f8, f10, f12, f16, f18, f22, f23) in 2D were part of the experiment. The results are compared to the base experiment in Figures 4.19 and 4.20 and are given the cyan color.

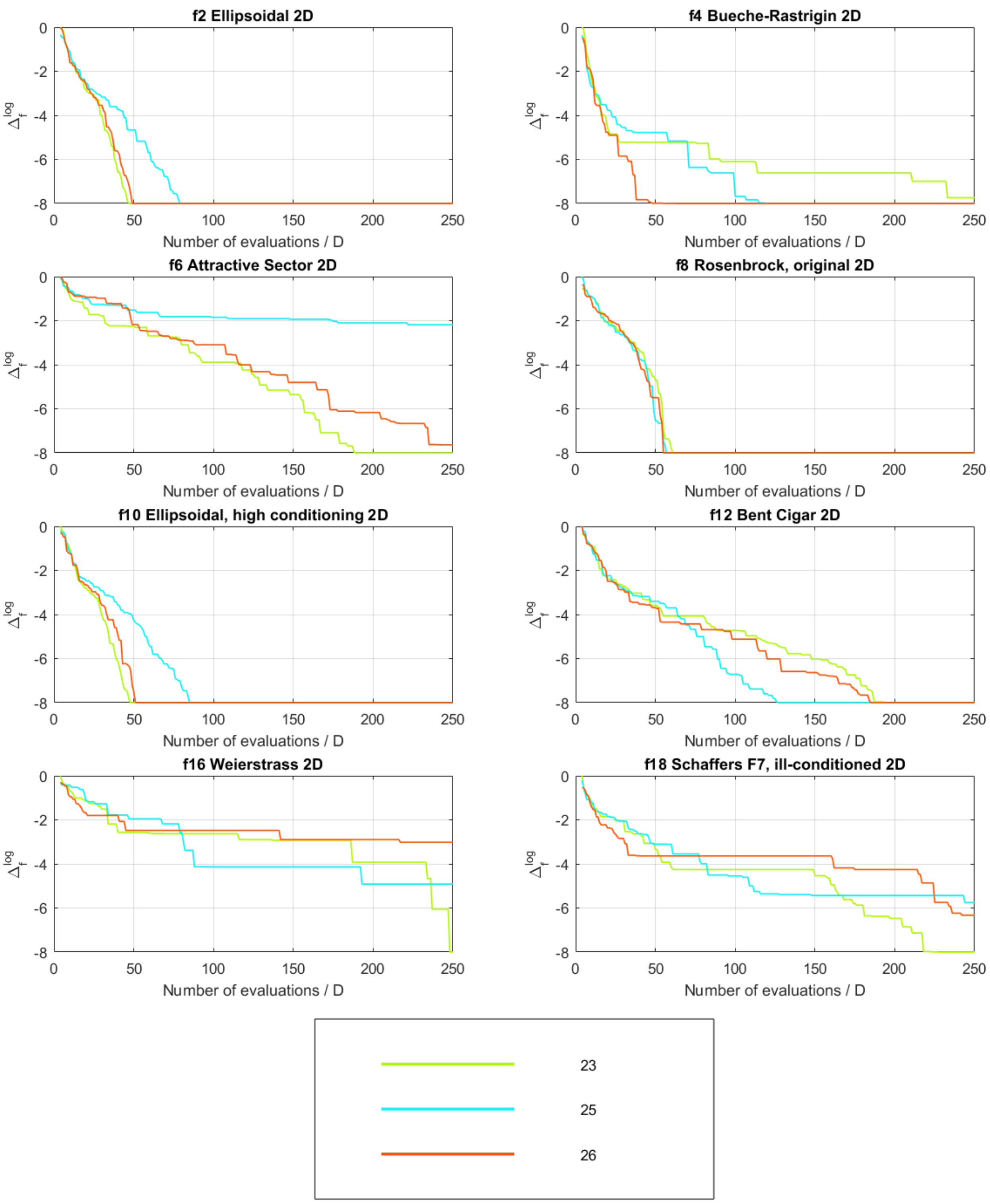


Figure 4.19: Experiments exp_doubleEC_23, exp_doubleEC_25 and exp_doubleEC_26, f2 to f18.

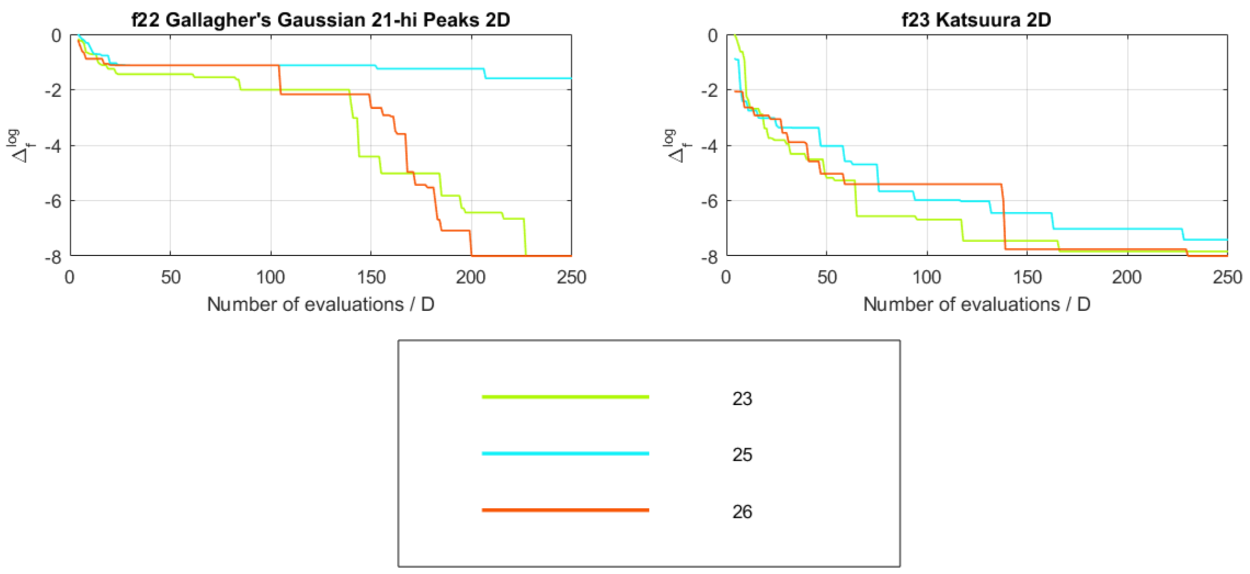


Figure 4.20: Experiments exp_doubleEC_23, exp_doubleEC_25 and exp_doubleEC_26, f22 and f23.

This experiment did not prove that the 'meanLinear' mean function was the cause of the bad performance, as it also did not match the expected results in functions f2, f6, f10 and f22. The performance in functions f4 and f12 was improved, however, it was not enough when looked at all of the functions together. Results in other functions were similiar to the base experiment. Although the reason of decreased performance was not proven to be only in using the linear mean function, when looked at the average statistics of 'meanConst' and 'meanLinear' of the GpModels from exp_GPtest_01, the linear mean function was rejected from the next experiments.
Next step was creating another experiment for the same functions as exp_doubleEC_25 called exp_doubleEC_26. However, in this case, none of the GpModels had the '{@covSEard}' covariance function (which was not part of the experiment exp_doubleEC_23_pool). Two of the models with SE ARD covariance function were replaced and all 4 are shown in the Table C.17. Results of the experiment exp_doubleEC_26 are in Figures 4.19 and 4.20 and are given the red color. The functions f2, f6, f10 and f22, where the experiment with SE ARD covariance functions was unsuccesfull, were improved almost to the level of the base experiment. The function 4 was improved even more and the difference is clearly visible. The performance in function 12 was worse than the experiment with ARD functions, but this was the only case. For those reasons, SE ARD function was no longer considered in next experiments.

4.1.6 Experiments exp_MPtest_13 to exp_MPtest_15

Despite the fact that the experiments exp_MPtest_01_rde to exp_MPtest_06_mse were not successful, they provided a lot of information about the ModelPool parameters for a series of next experiments. Following options were set for the series:

bestModelSelection 'rdeAll',

historyLength 7,

minTrainedModelsPercentileForModelChoice 0.5,

maxGenerationShiftForModelChoice 2.

From already calculated statistics, a new set of GpModels was created. This time, only 96 models were taken into the consideration and the rde_ranking script was updated by Mgr. Lukáš Bajer, so that it generates more sets of the GpModels. From the generated sets, the script choosed the one with the best average statistics. Models chosen for the next set of experiments are in Tables C.18 (2D), C.19 (5D) and C.20 (10D). The results are in figures 4.21 to 4.29 and are given the red color. The base experiment exp_doubleEC_23 has green color in this report, exp_doubleEC_23_pool_re has cyan color and other algorithms, whose data are available in the project, are included in those figures as well, so that the performance can be compared to them easily.

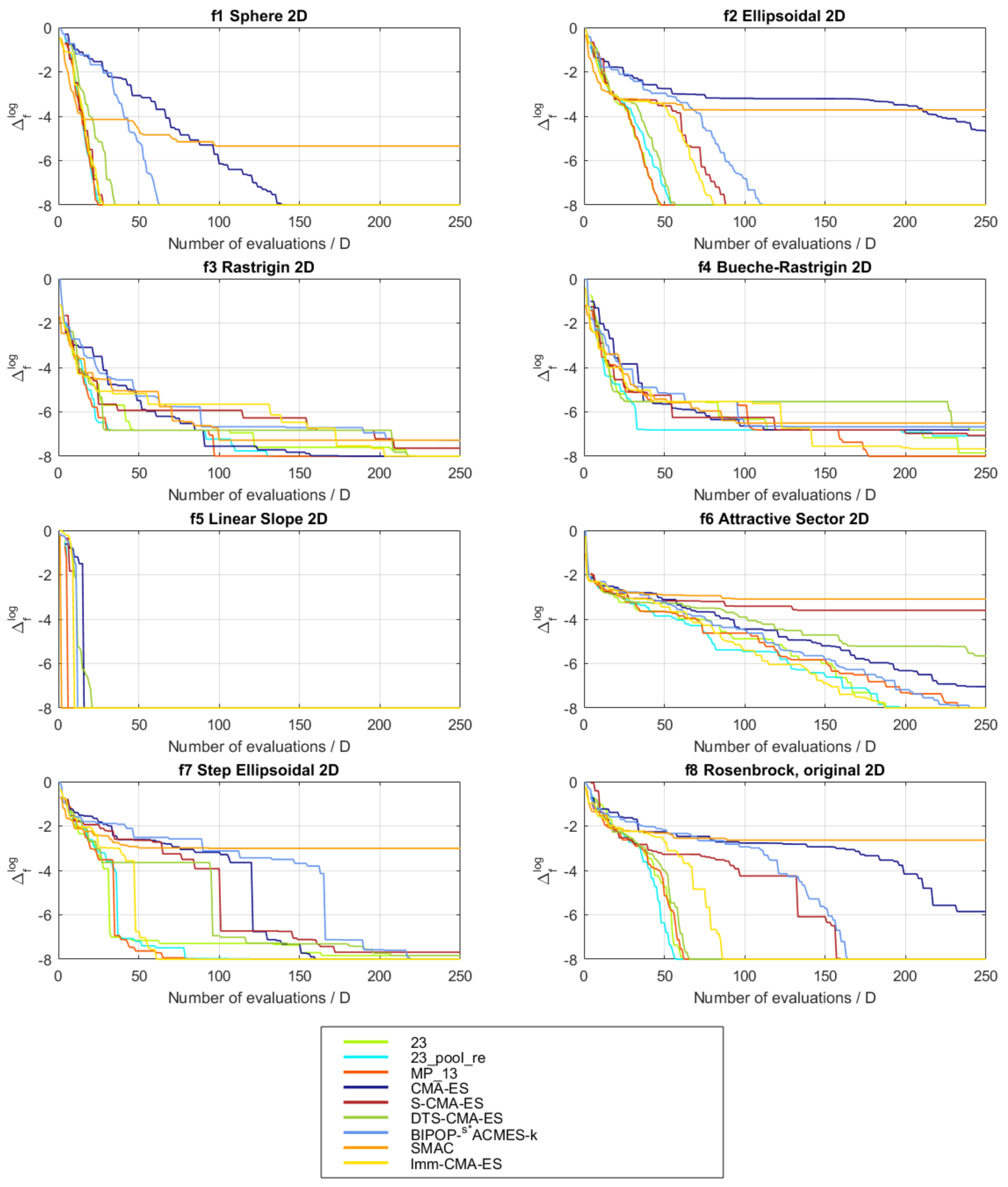


Figure 4.21: Experiments exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_13, 2D, f1 to f8.

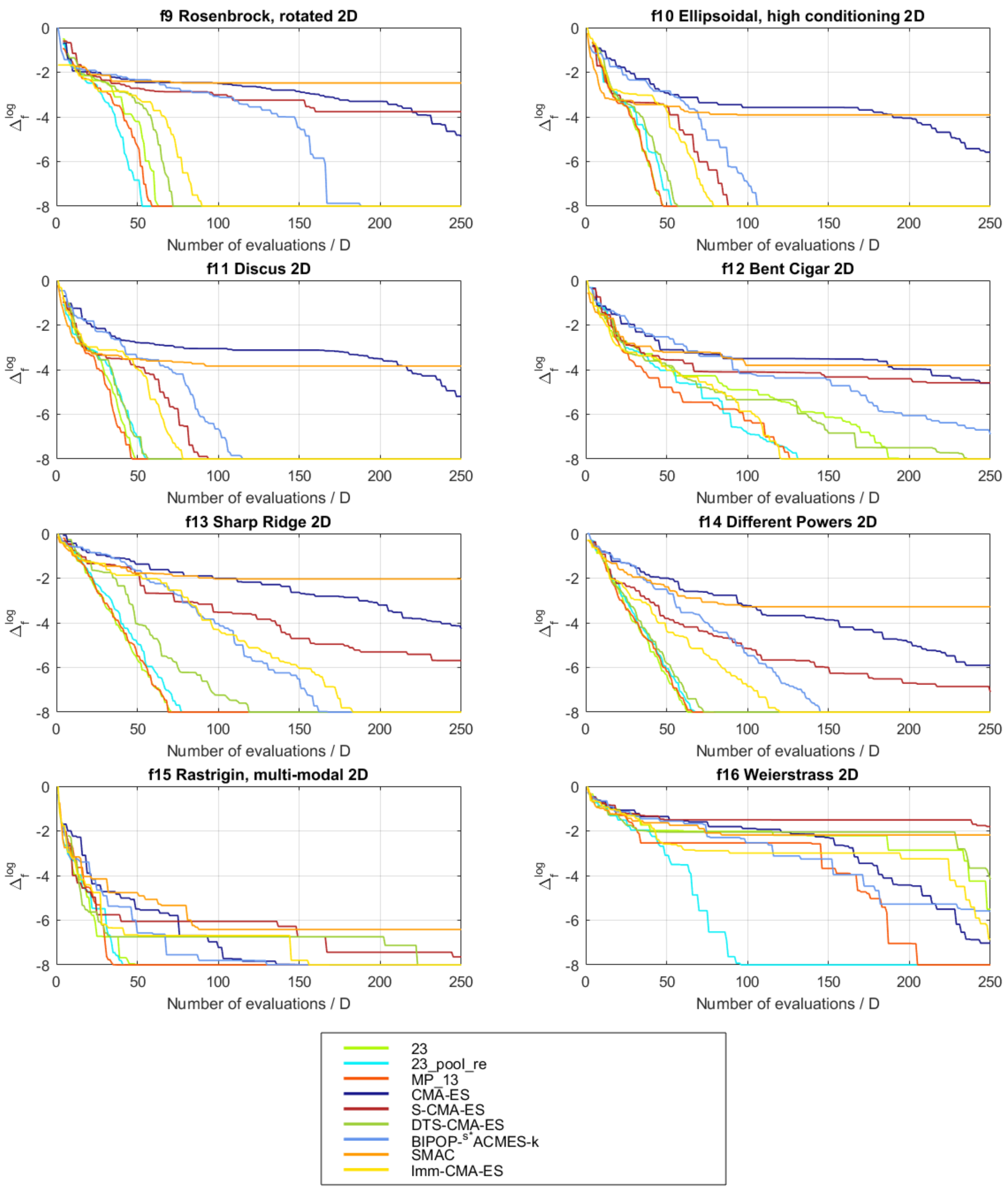


Figure 4.22: Experiments exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_13, 2D, f9 to f16.

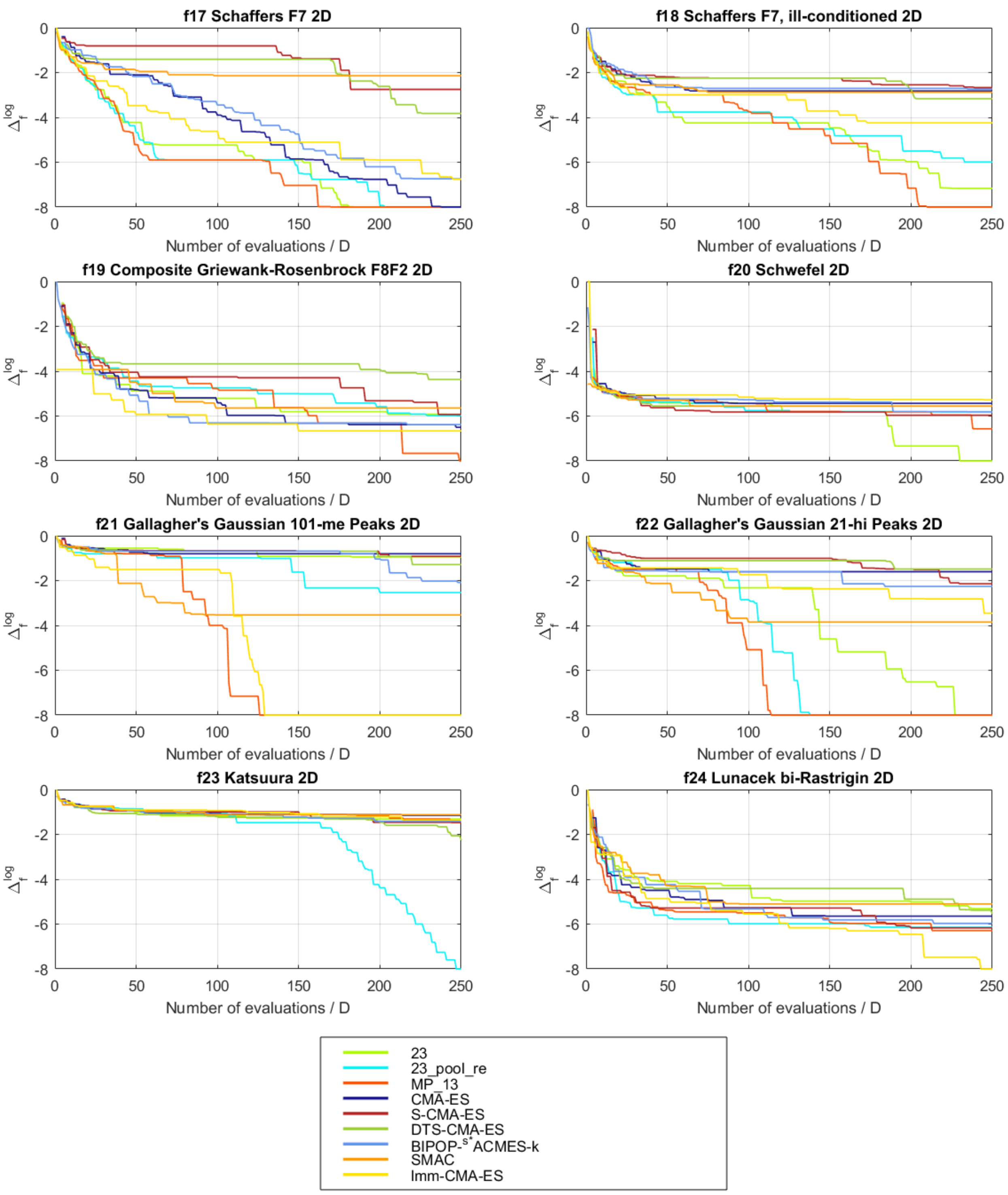


Figure 4.23: Experiments exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_13, 2D, f17 to f24.

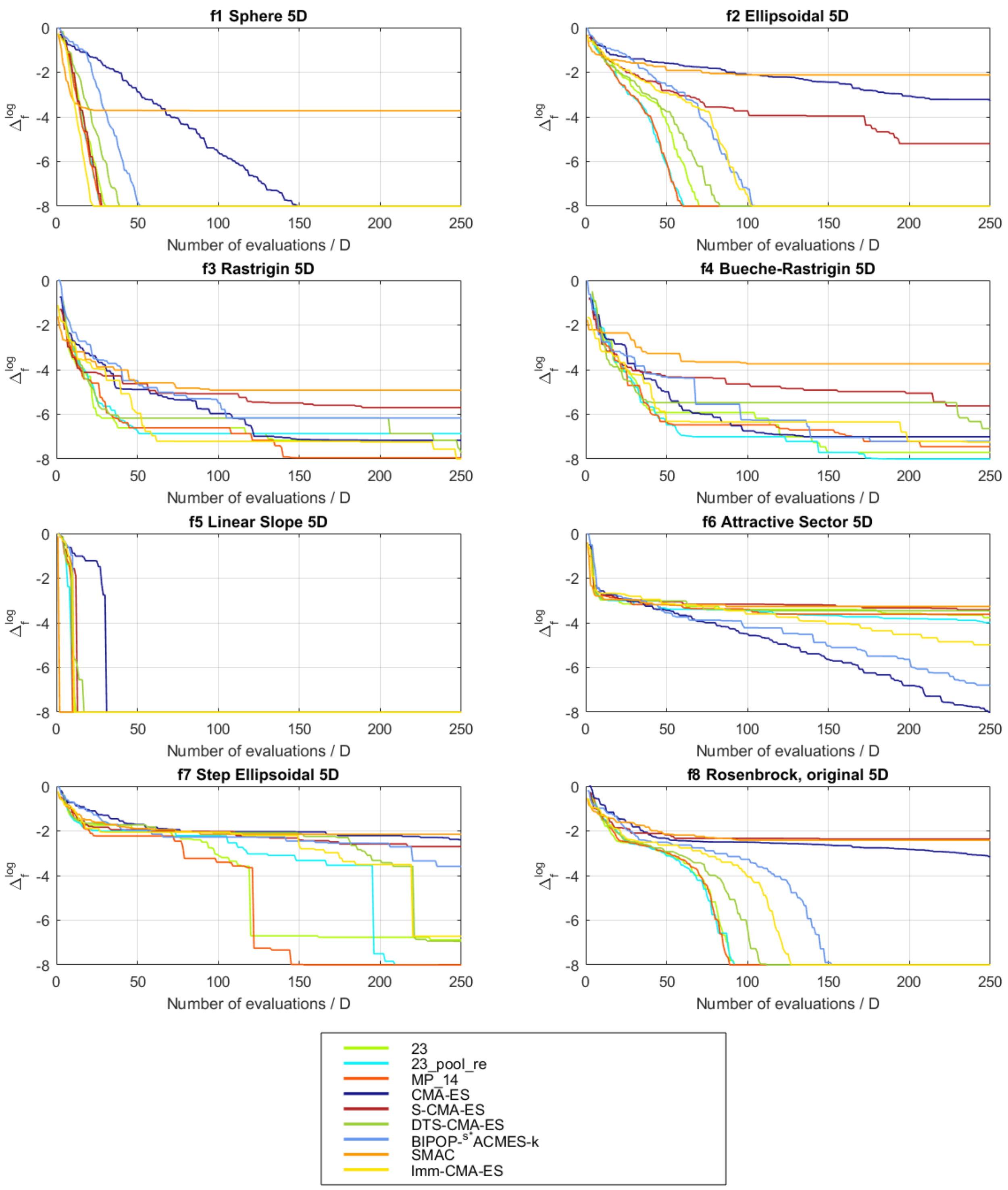


Figure 4.24: Experiments exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_14, 5D, f1 to f8.

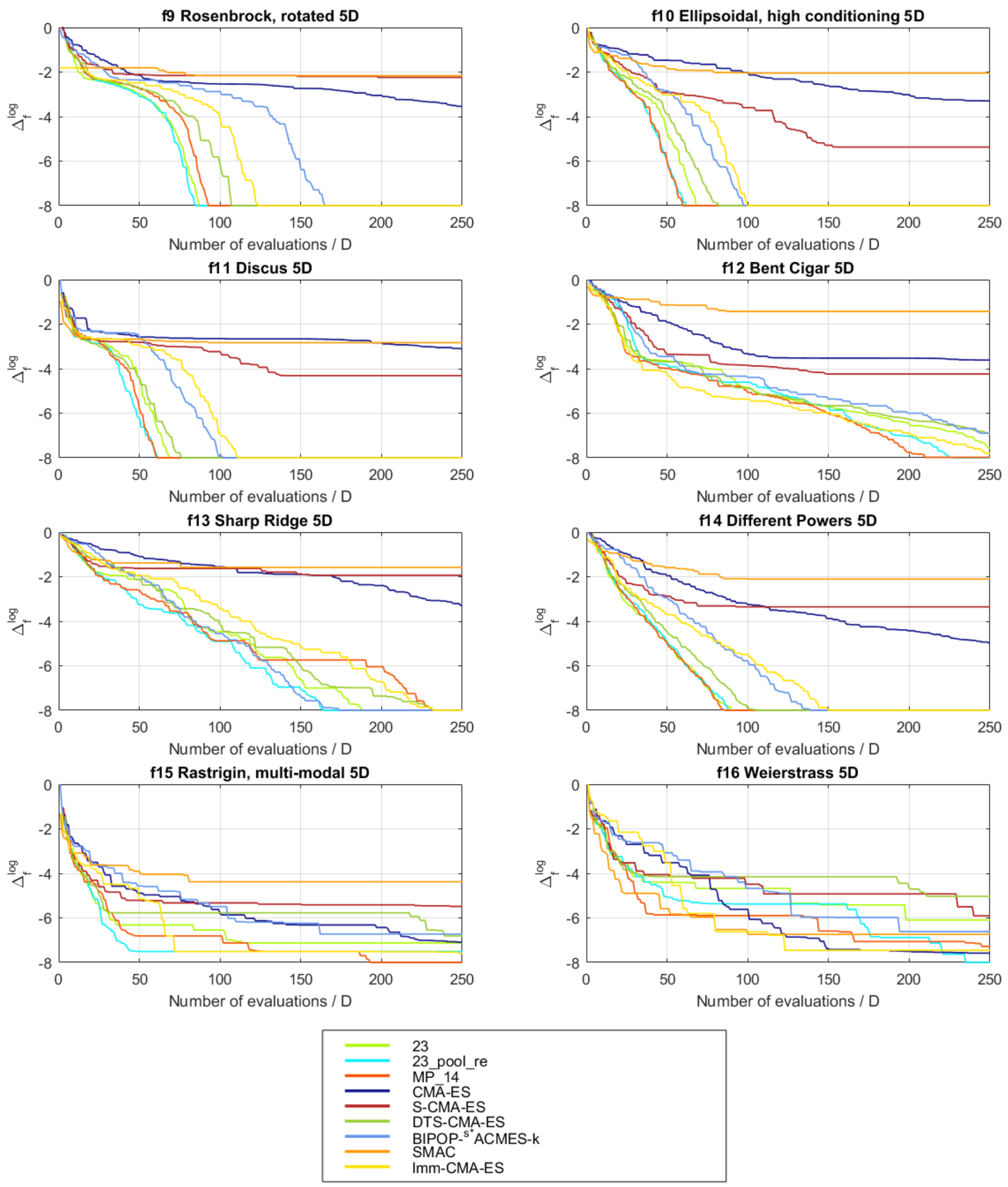


Figure 4.25: Experiments exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_14, 5D, f9 to f16.

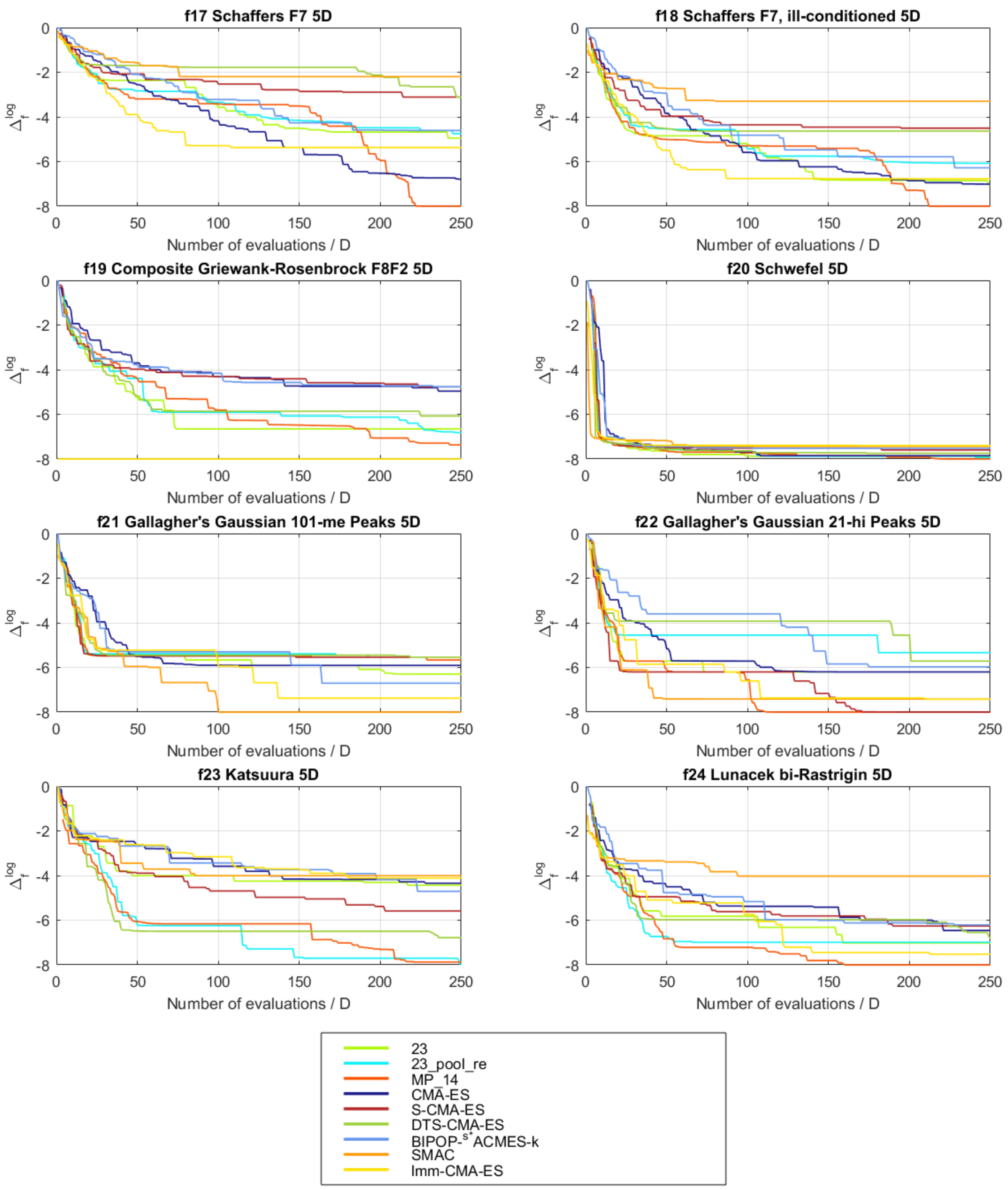


Figure 4.26: Experiments exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_14, 5D, f17 to f24.

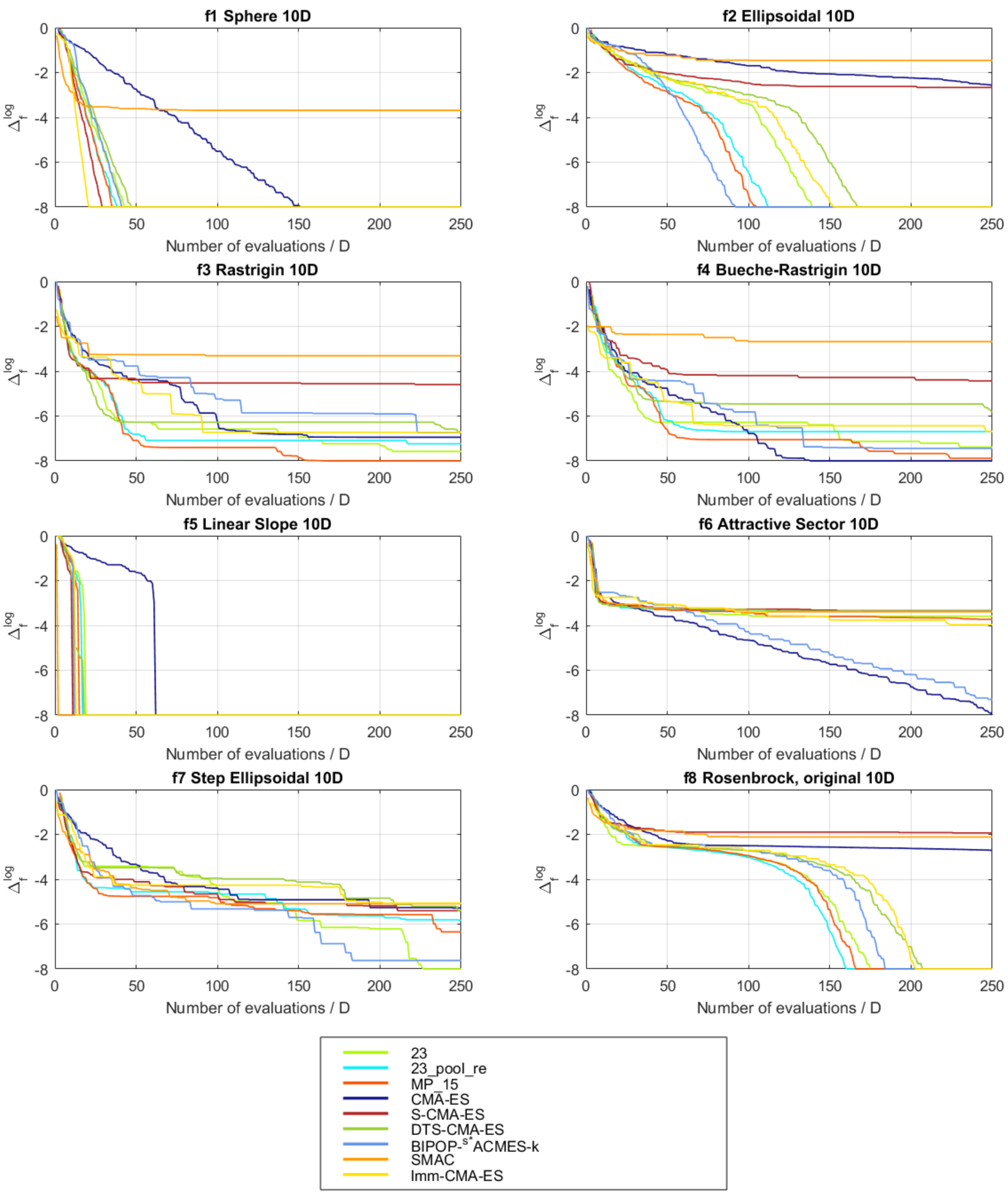


Figure 4.27: Experiments exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_15, 10D, f1 to f8.

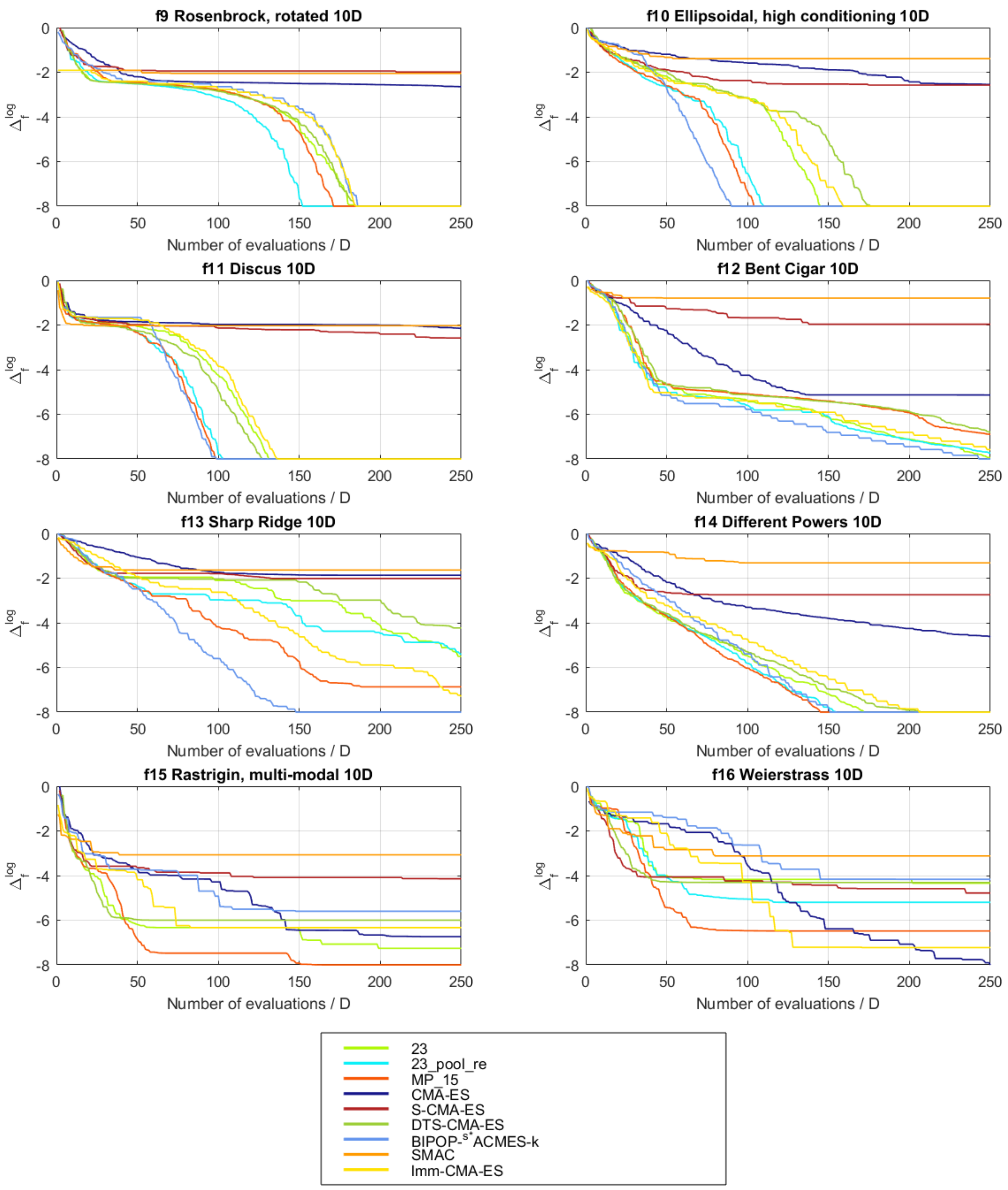


Figure 4.28: Experiments exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_15, 10D, f9 to f16.

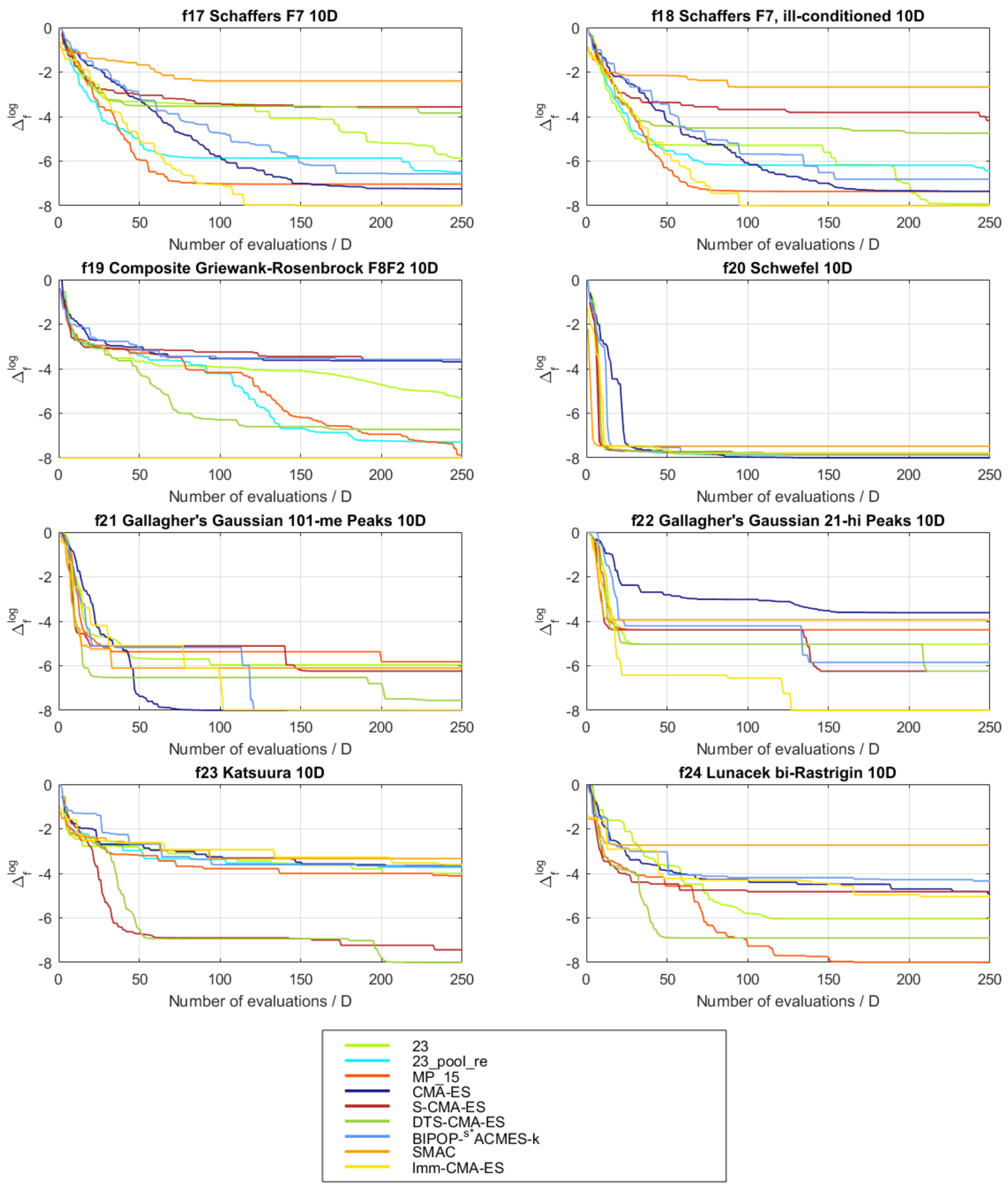


Figure 4.29: Experiments exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_15, 10D, f17 to f24.

Graphs of average of all functions in each dimension are shown in Figures 4.30 to 4.32. As can be seen on the results, this set of experiments was successful and improved the performance when compared to the base experiment. In some functions the experiment exp_doubleEC_23_pool_re had better results, but when looked at the overall performance, the experiments with the best results are exp_doubleEC_MP_13 to exp_doubleEC_MP_15.

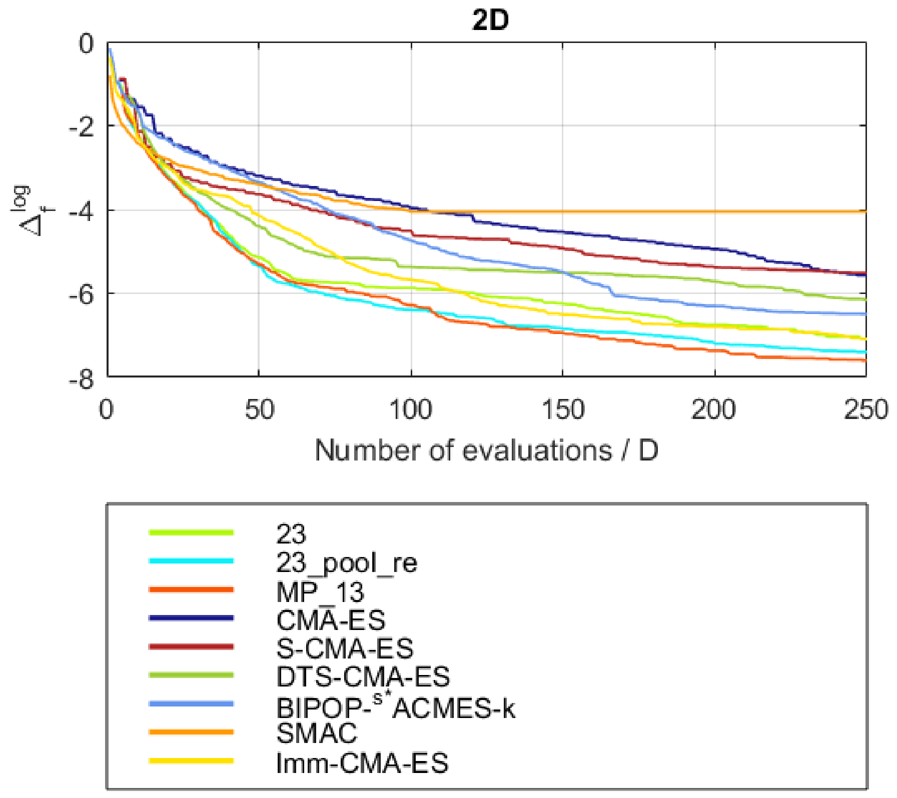


Figure 4.30: The graph of average of the results of all 24 noiseless functions, 2D.
The experiments are exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_13 and exp_doubleEC_MP_14.

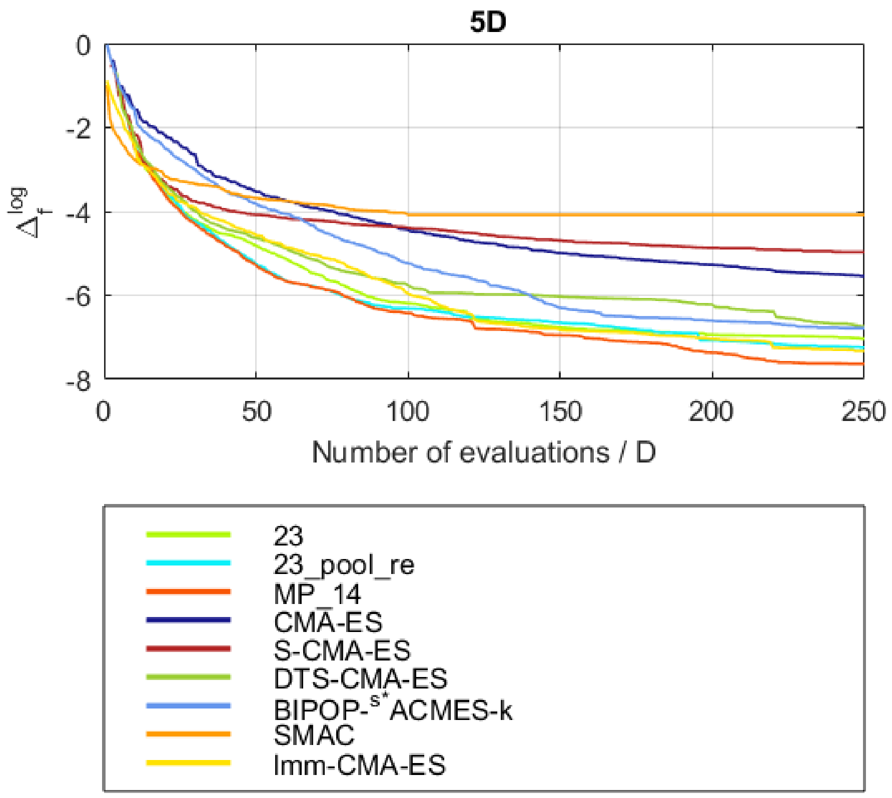


Figure 4.31: The graph of average of the results of all 24 noiseless functions, 5D.
The experiments are exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_13 and exp_doubleEC_MP_14.

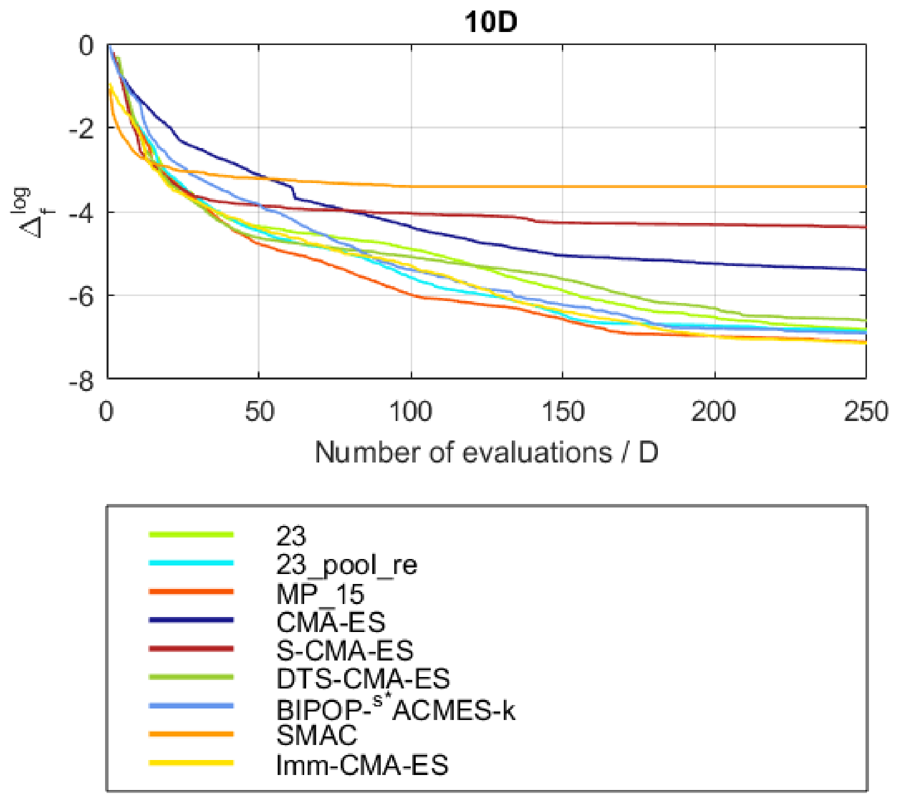


Figure 4.32: The graph of average of the results of all 24 noiseless functions, 10D.
The experiments are exp_doubleEC_23, exp_doubleEC_23_pool_re and exp_doubleEC_MP_15.

4.2 Noisy functions

4.2.1 Experiments exp_doubleEC_24_noisy, exp_doubleEC_24_noisy_pool

Experiments for noisy functions were created in a similiar way as the ones for the noiseless functions. The base experiment for the noisy function was exp_doubleEC_24_noisy and it uses the same model as exp_doubleEC_23, which can be found in Table C.1. The results of this experiment can be found in Figures 4.33 to 4.35 and are given the green color. They are shown together with the experiment exp_doubleEC_24_noisy_pool, which had the same GpModels (found in Table C.2) as the experiment exp_doubleEC_23_pool and are given the cyan color. This experiment was also run only in 2D and 5D. The performance, unlike in the case of noiseless functions, was decreased, except for the function f106 in 2D.

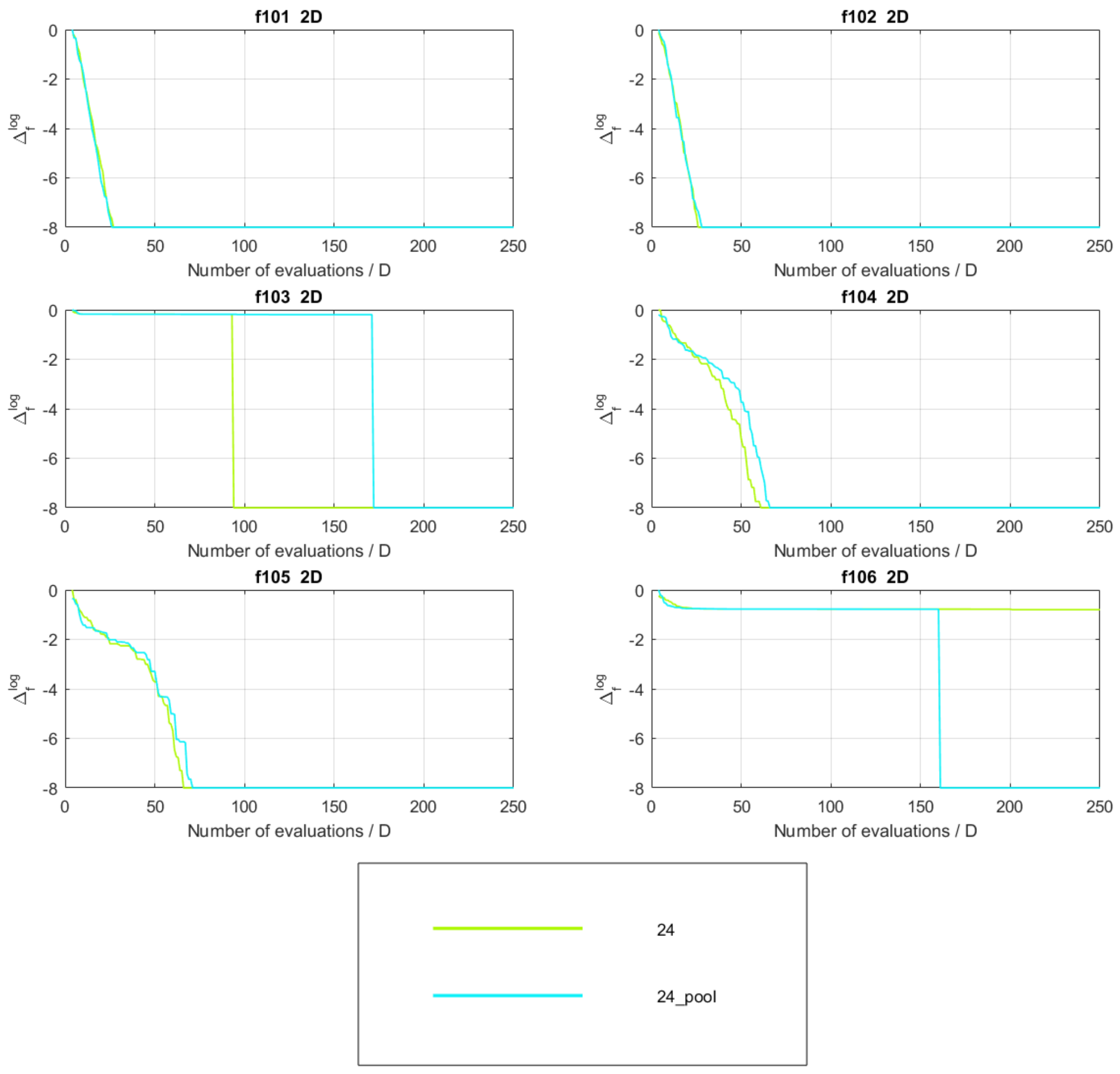


Figure 4.33: Experiments exp_doubleEC_24_noisy, exp_doubleEC_24_noisy_pool, 2D.

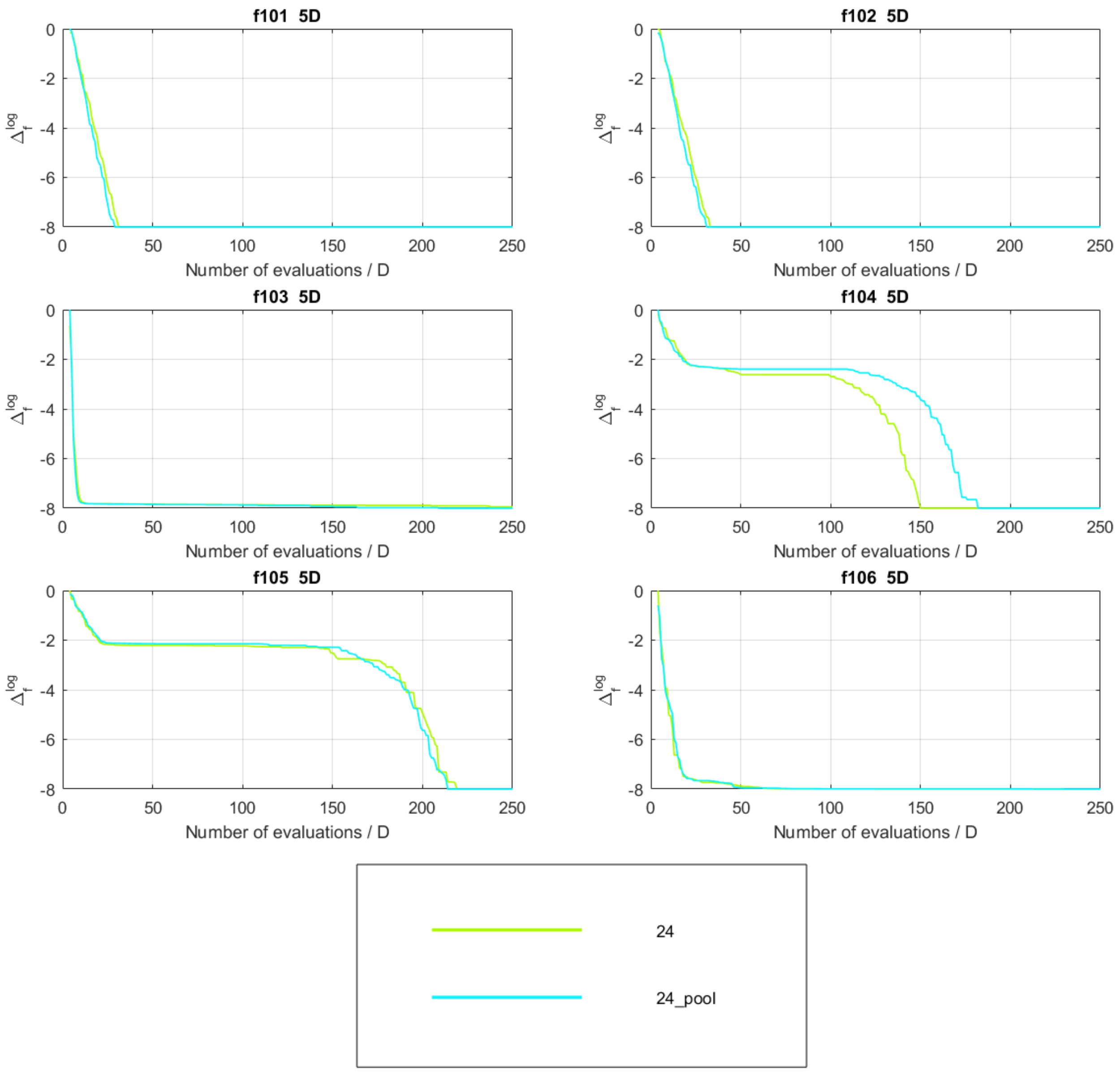


Figure 4.34: Experiments exp_doubleEC_24_noisy, exp_doubleEC_24_noisy_pool, 5D.

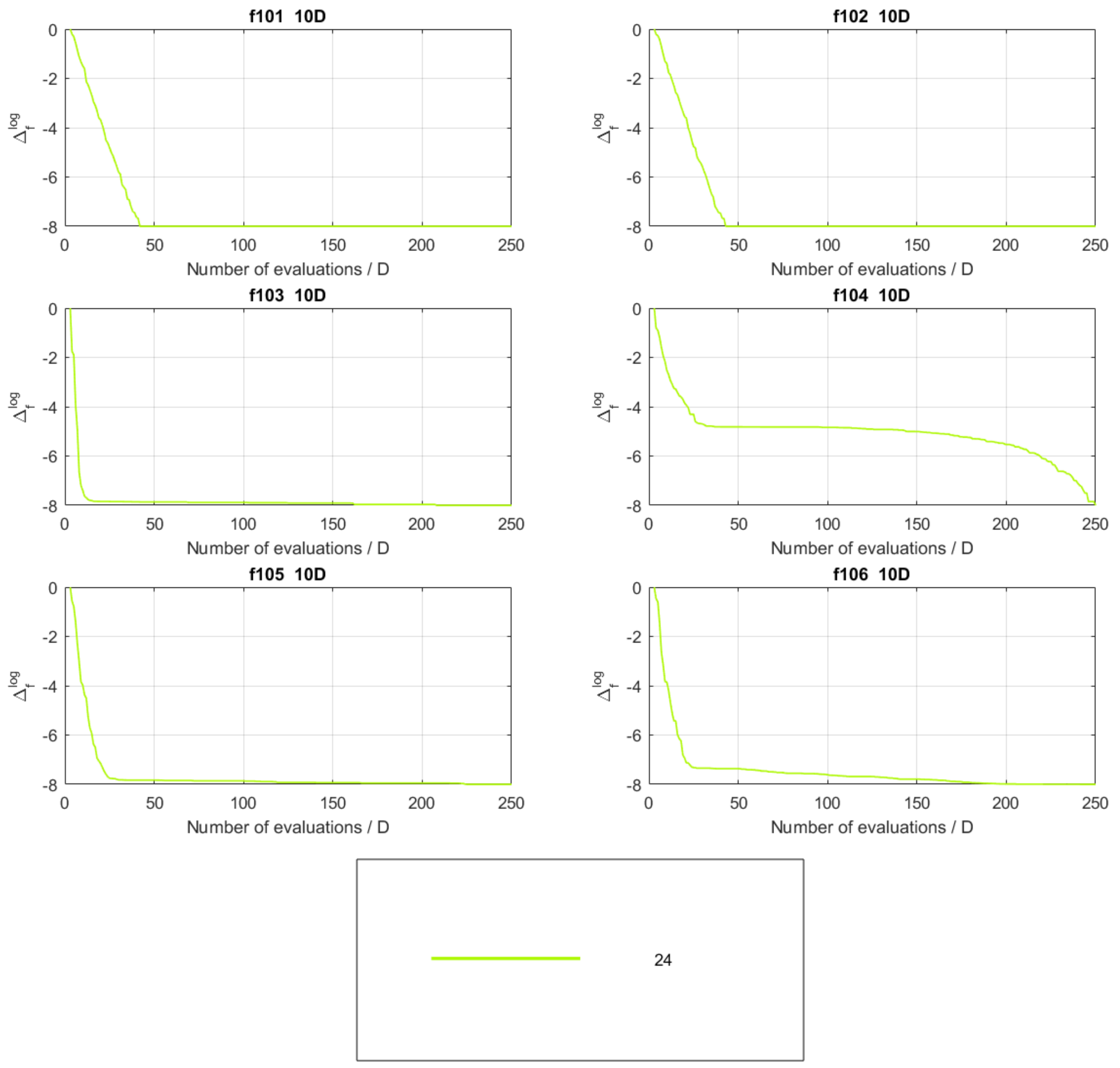


Figure 4.35: Experiments exp_doubleEC_24_noisy, 10D.

4.2.2 Experiment exp_GPtest_02_noisy for choosing the GpModels

The same process of choosing the GpModels as in the case of noiseless functions was used on noisy functions. Refrences to the appropriate tables can be found in following table:

Dimension	Criterium	Table
2	RDE	C.10
5	RDE	C.11
10	RDE	C.12
2	MSE	C.13
5	MSE	C.14
10	MSE	C.15

Table 4.3: Table of the references to the chosen GpModels from exp_GPtest_02_noisy.

The parameters for the ModelPool class were not specially chosen, results from the noiseless functions were used instead:

Dim.	Criterium	bestModelSel.	historyL.	minTr.Perc.	maxGen.Shift
2	RDE	'rdeAll'	7	0.25	2
5	RDE	'rdeAll'	7	0.5	0
10	RDE	'rdeOrig'	5	0.25	2
2	MSE	'mse'	3	0.25	0
5	MSE	'mse'	3	0.25	1
10	MSE	'mse'	3	0.5	1

Table 4.4: ModelPool parameters chosen by experiments exp_MPtest_01_rde to exp_MPtest_06_mse.

Using these settings, experiments exp_doubleEC_MP_07_noisy to exp_doubleEC_MP_12_noisy were created and their results are in Figures 4.36 to 4.38 with cyan (RDE) and red (MSE) color.

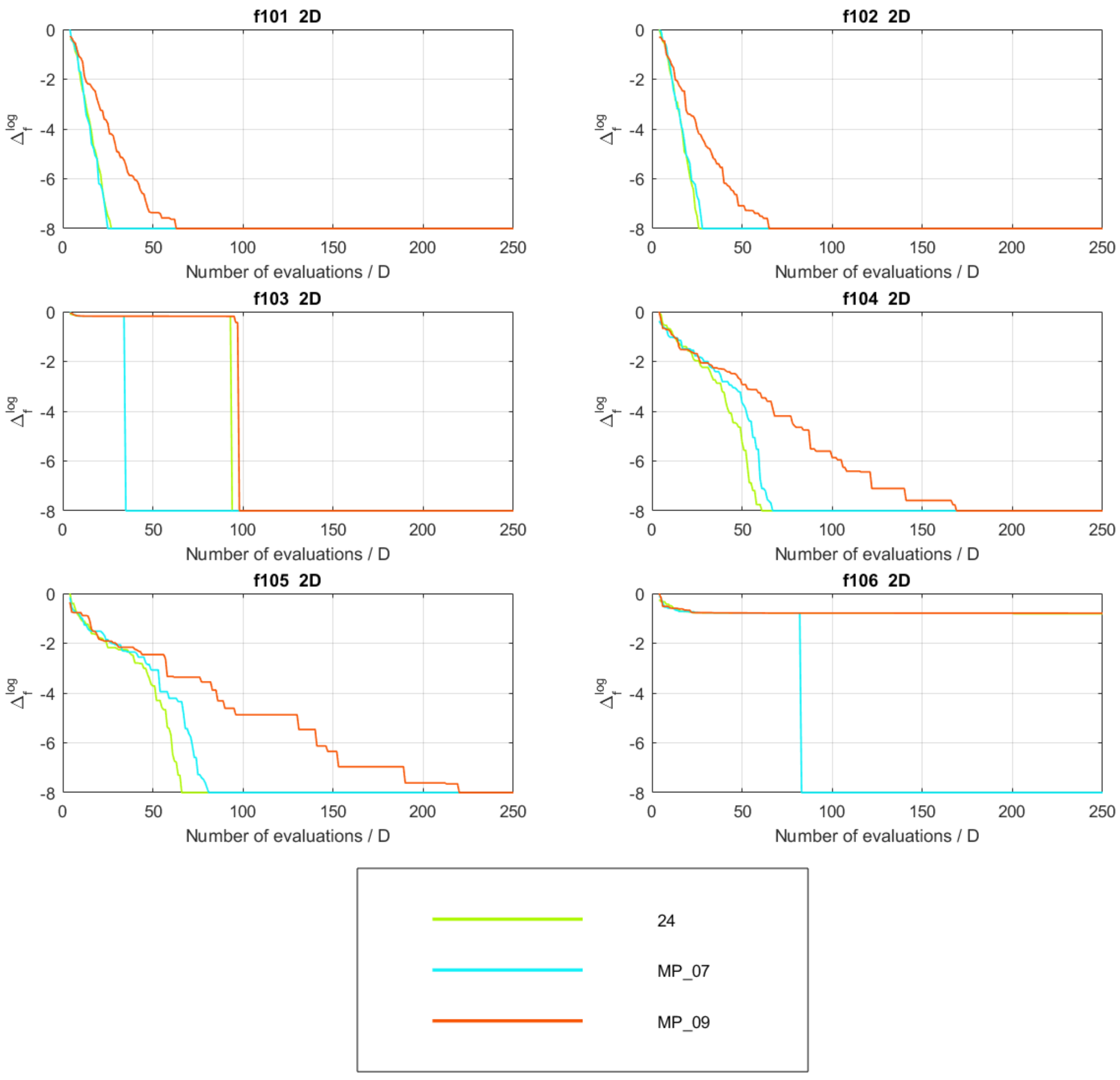


Figure 4.36: Experiments exp_doubleEC_24_noisy, exp_doubleEC_MP_07_noisy (RDE) and exp_doubleEC_MP_09_noisy (MSE), 2D.

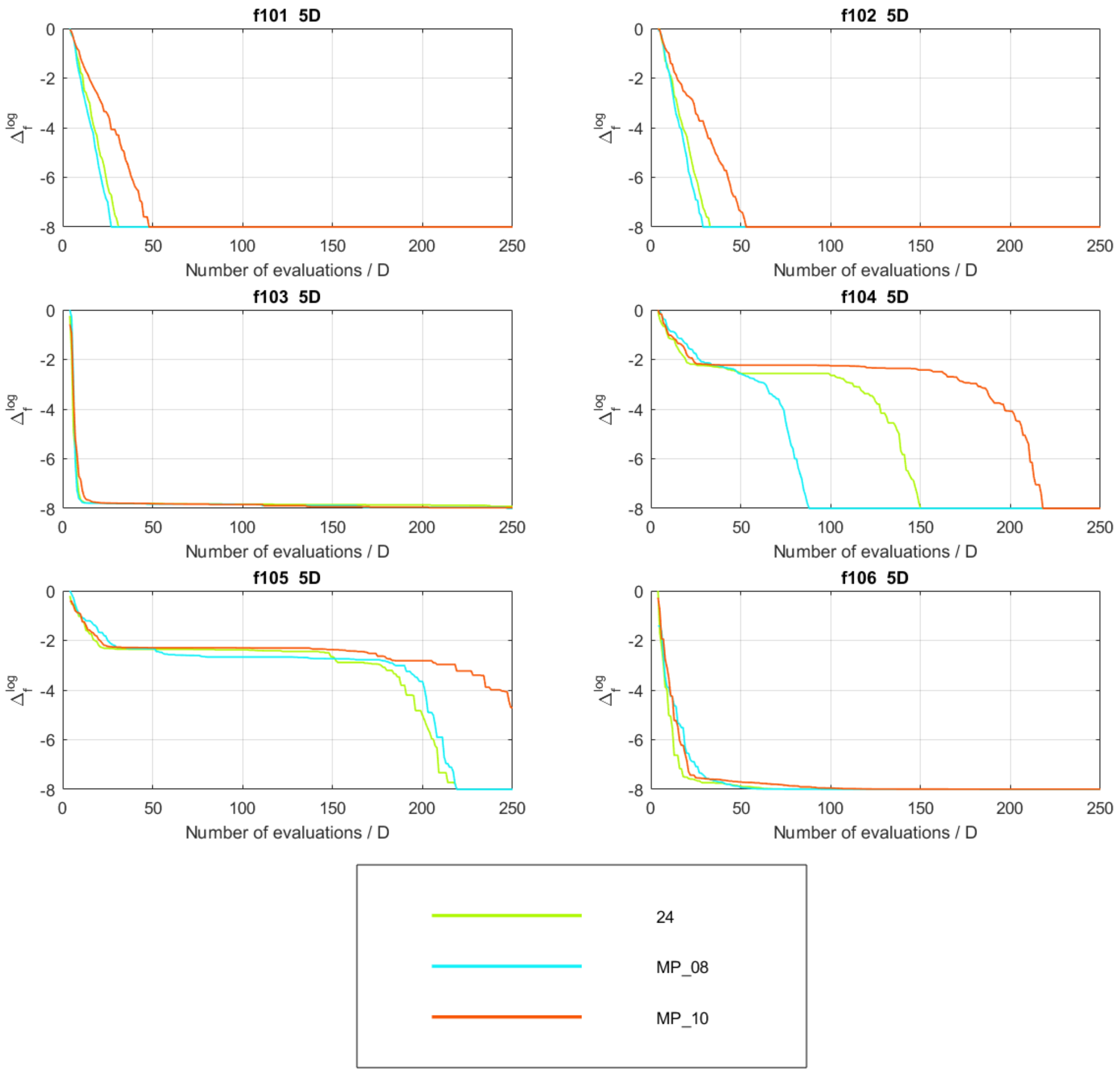


Figure 4.37: Experiments exp_doubleEC_24_noisy, exp_doubleEC_MP_08_noisy (RDE) and exp_doubleEC_MP_10_noisy (MSE), 5D.

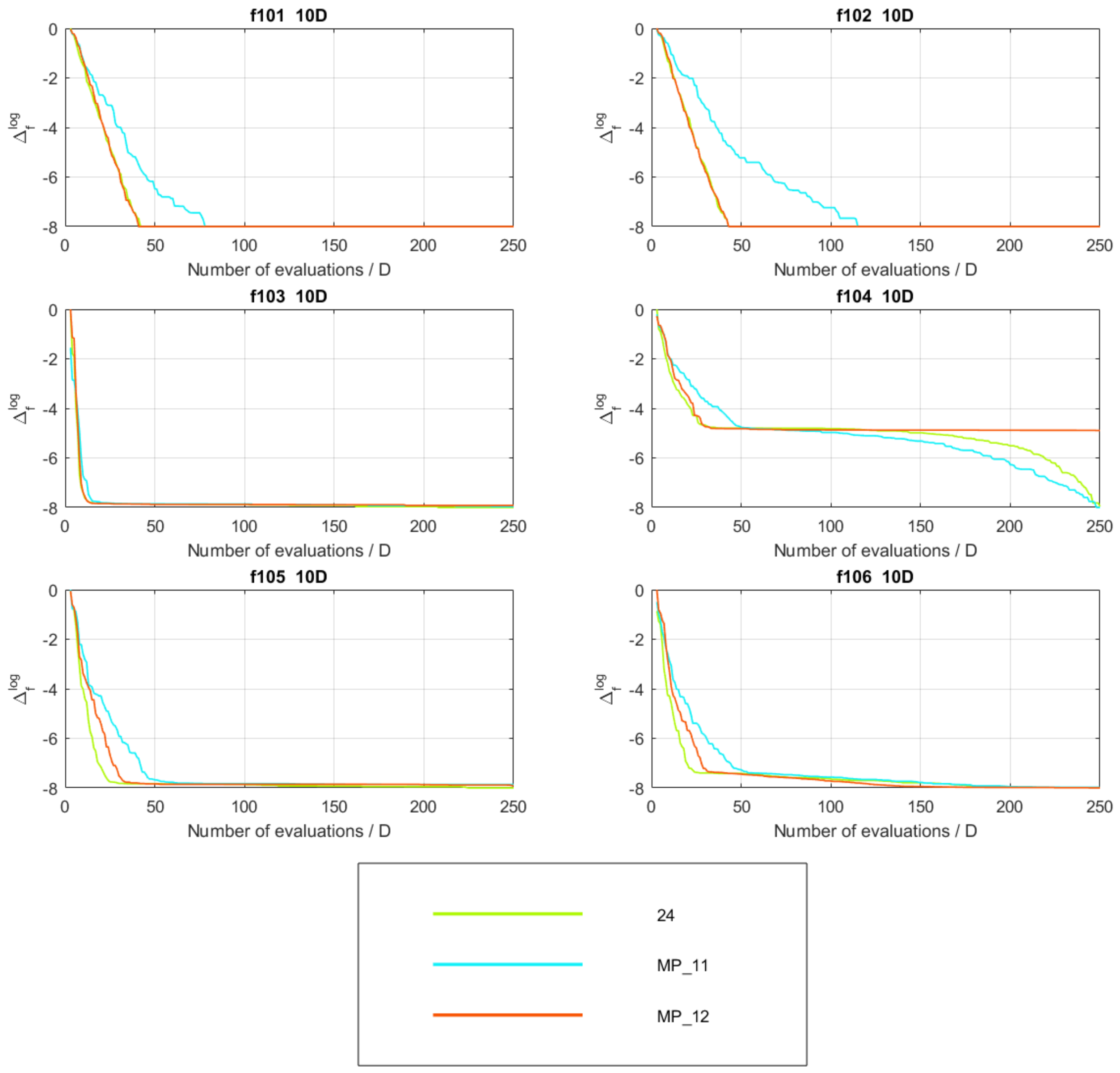


Figure 4.38: Experiments exp_doubleEC_24_noisy, exp_doubleEC_MP_11_noisy (RDE) and exp_doubleEC_MP_12_noisy (MSE), 10D.

The same conclusion as in the case of noiseless functions was made - the best option for the selection of the best model is the 'rdeAll'. However, in this case, the performance in 2D and 5D was slightly improved. The biggest difference can be seen in the functions f103 2D, f106 2D and f104 in 5D. In 10D, the 'rdeOrig' option was used instead of 'rdeAll', which was the cause of the worse performance in this dimension.

4.2.3 Experiments exp_MPtest_16_noisy to exp_MPtest_18_noisy

Again, the set of possible GpModels used for the selection was reduced to 96 by removing the models with SE ARD covariance function and linear mean function. The parameters of the ModelPool were also defined as those for the last noiseless experiments:

bestModelSelection 'rdeAll',

historyLength 7,

minTrainedModelsPercentileForModelChoice 0.5,

maxGenerationShiftForModelChoice 2.

The chosen GpModels are in the Tables C.21 (2D), C.22 (5D) and C.23 (10D). The results are shown together with the base experiment exp_doubleEC_24_noisy and RDE version of experiments from previous set of tests in Figures 4.39 to 4.41. Unfortunately, results of other algorithms, that were used in comparing the final results of the noiseless functions, were not available for the noisy functions. Graphs with average of all functions in each dimension are in Figures 4.42 to 4.44.

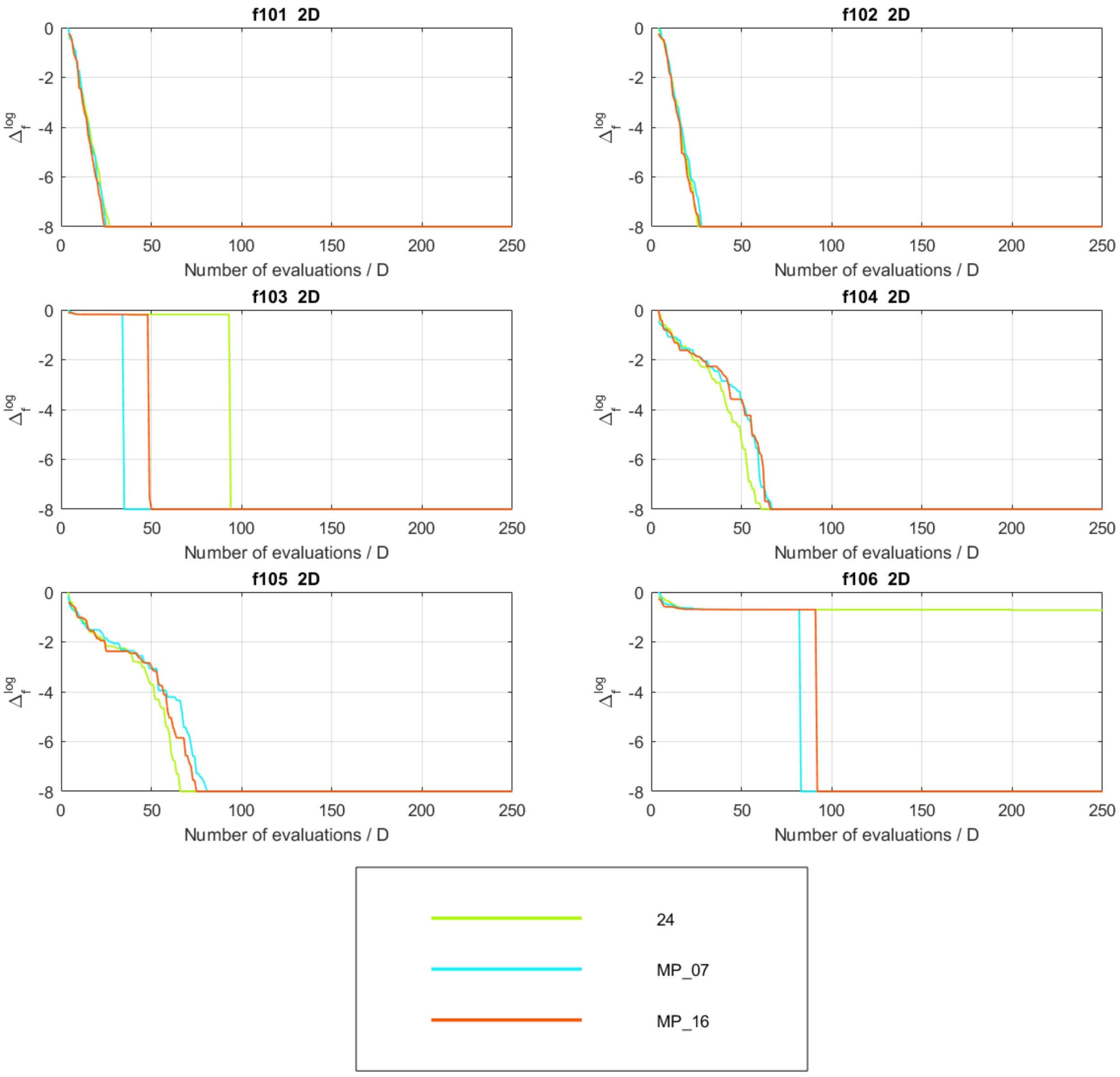


Figure 4.39: Experiments exp_doubleEC_24_noisy, exp_doubleEC_MP_07_noisy and exp_doubleEC_MP_16_noisy, 2D.

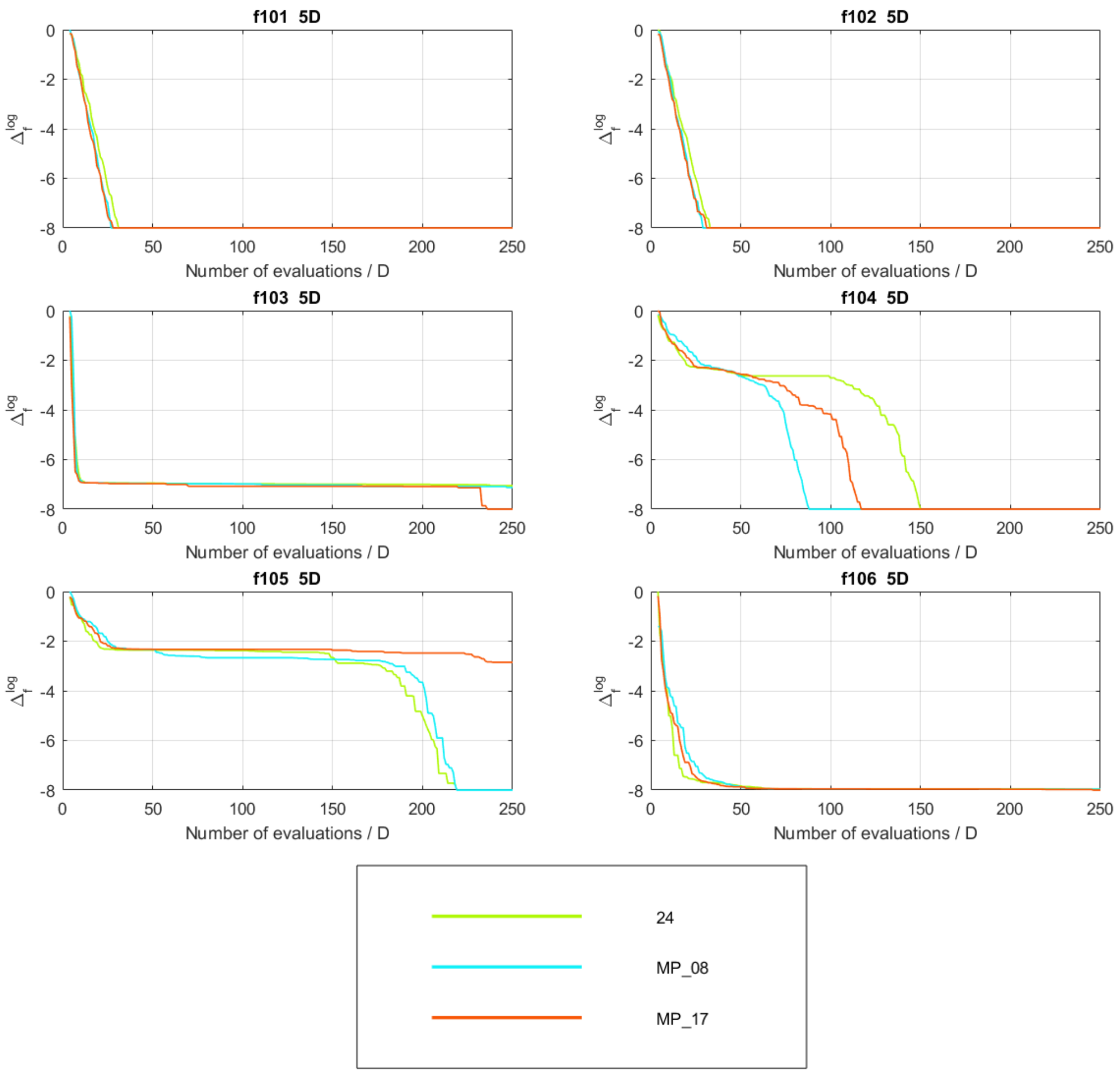


Figure 4.40: Experiments exp_doubleEC_24_noisy, exp_doubleEC_MP_08_noisy and exp_doubleEC_MP_17_noisy, 5D.

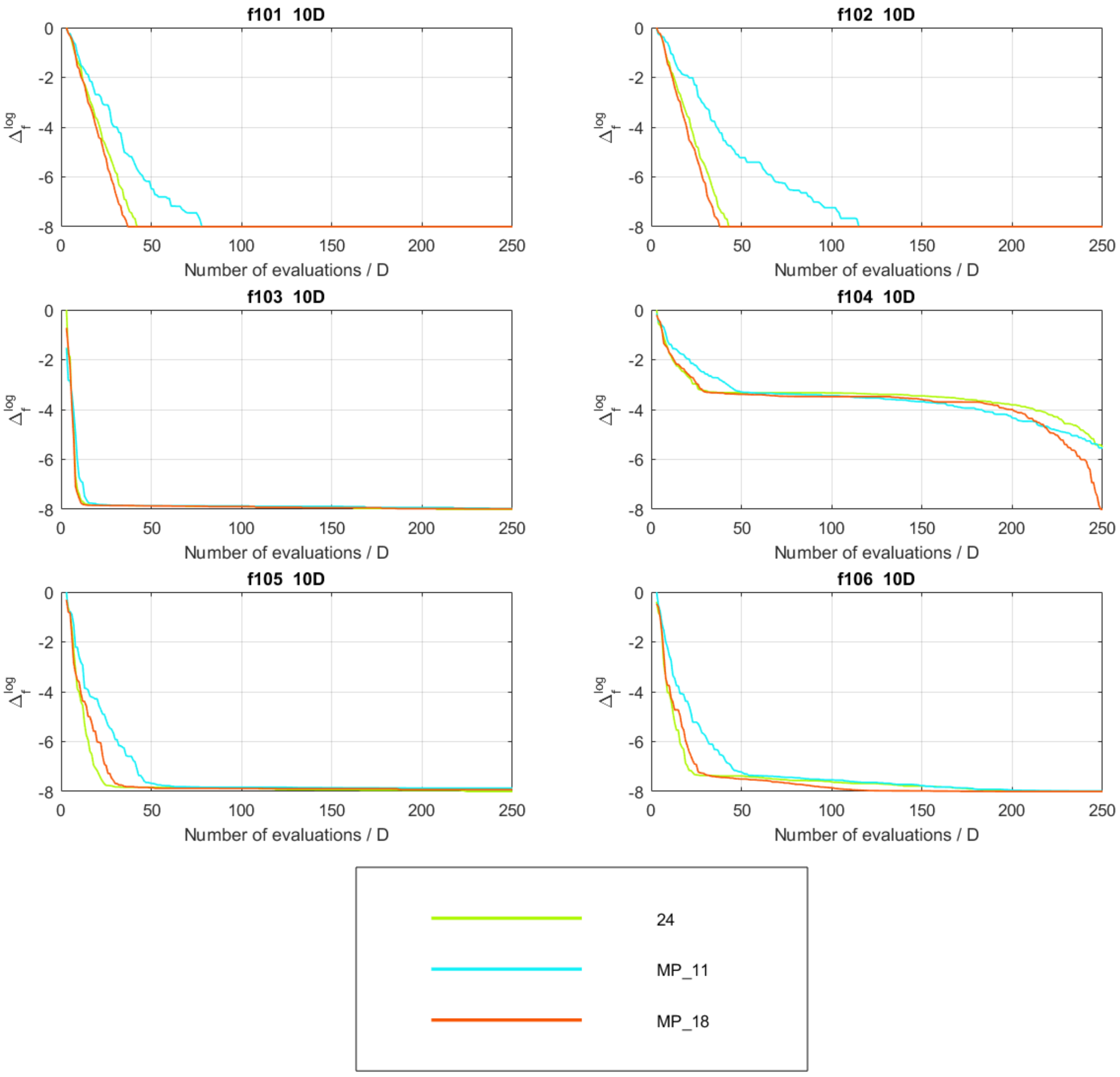


Figure 4.41: Experiments exp_doubleEC_24_noisy, exp_doubleEC_MP_11_noisy and exp_doubleEC_MP_18_noisy, 10D.

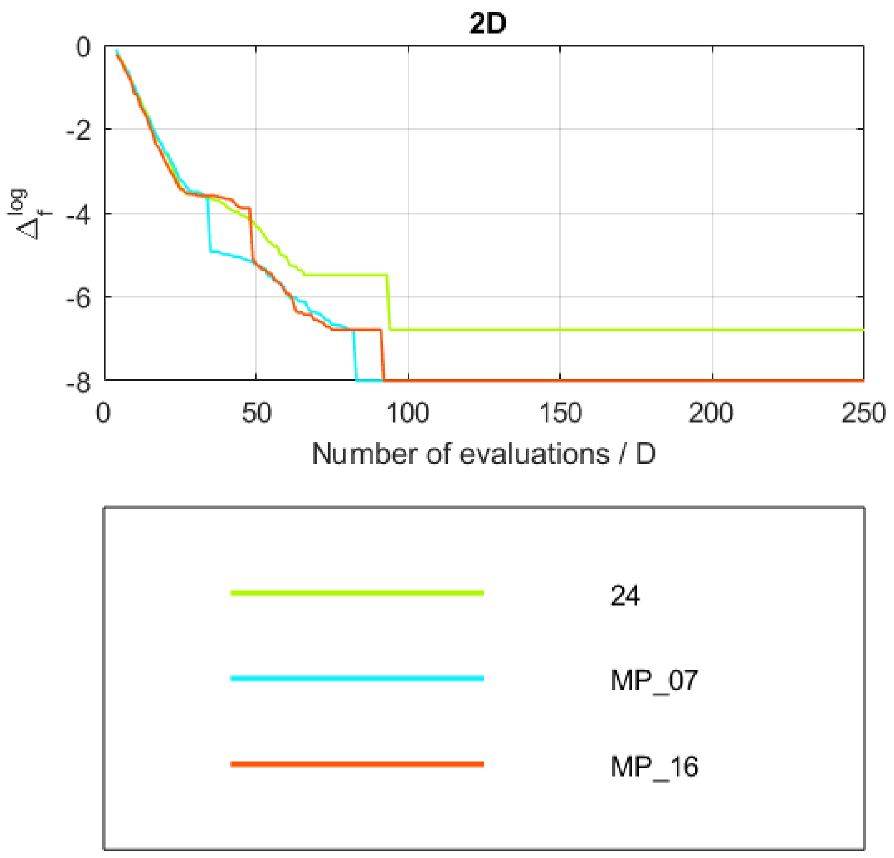


Figure 4.42: The graph of average of the results of 6 noisy functions, 2D. The experiments are exp_doubleEC_24_noisy, exp_doubleEC_MP_07_noisy and exp_doubleEC_MP_16_noisy.

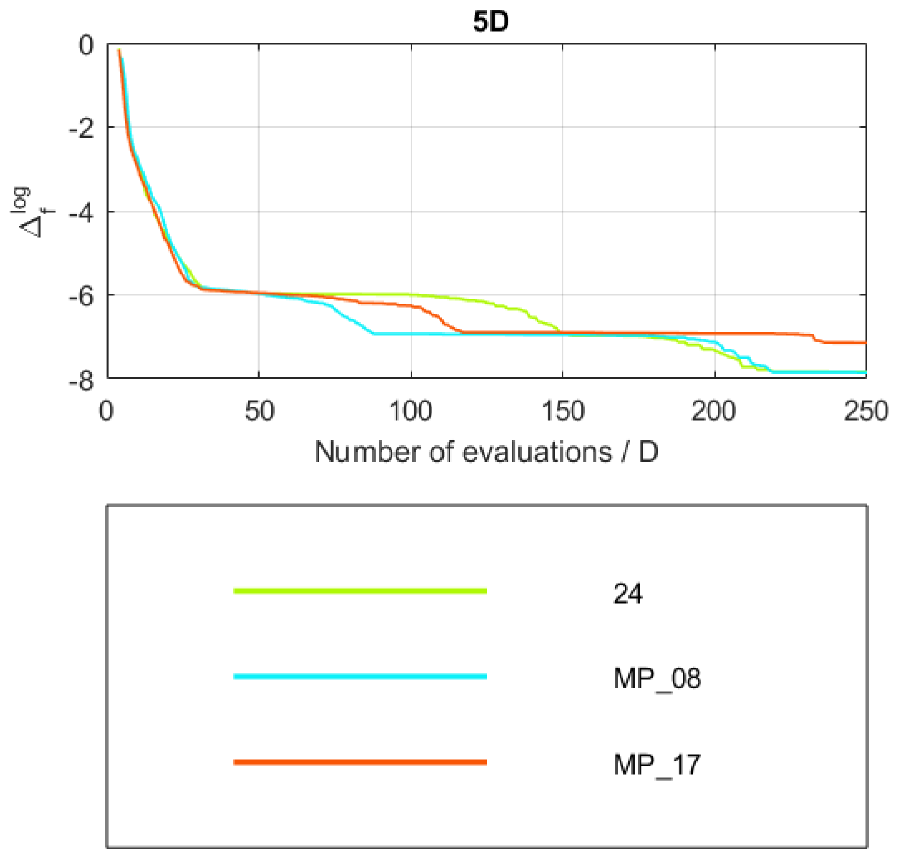


Figure 4.43: The graph of average of the results of 6 noisy functions, 5D. The experiments are exp_doubleEC_24_noisy, exp_doubleEC_MP_08_noisy and exp_doubleEC_MP_17_noisy.

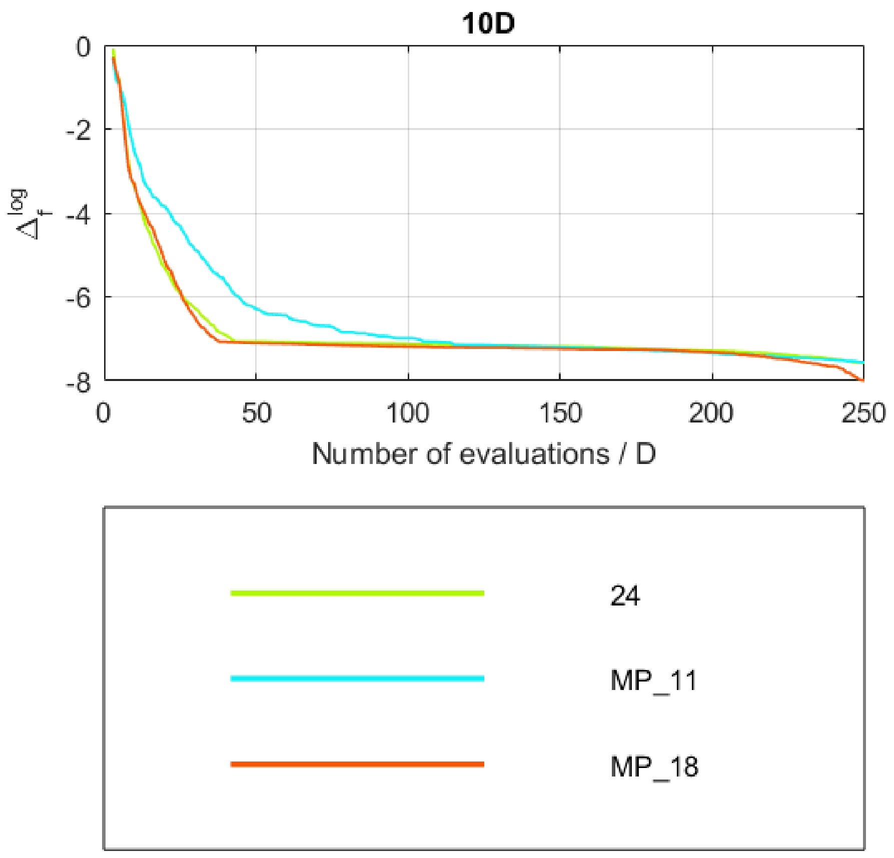


Figure 4.44: The graph of average of the results of 6 noisy functions, 10D. The experiments are exp_doubleEC_24_noisy, exp_doubleEC_MP_11_noisy and exp_doubleEC_MP_18_noisy.

The graphs of averages show that the best results in 2D and 5D had the experiments that had GpModels chosen from all 256 combinations - that means with linear mean function and SE ARD covariance function. The difference in 2D is, however, very small when looked at each of the functions. The results of 5D are visibly better for the GpModels chosen from all 256 combinations. The 10D results of models chosen from the second set of 96 GpModels are very similiar to the results of the base experiment. Small improvements can be seen in functions f101, f102 and f104 and functions f105 and f106 are the opposite case. Results of the exp_doubleEC_MP_11_noisy in 10D are worse because of the 'rdeOrig' option of the selection of the best model in the ModelPool.

Chapter 5

Conclusion

In this thesis, the author was improving the performance of the CMA-ES algorithm by using multiple Gaussian processes as surrogate models simultaneously. The measurements were done using the COCO platform on 24 noiseless and 6 noisy functions in 2D, 5D and 10D dimensions. Appropriate changes to the implementation of surrogate models were done in a project that the author was part of. The class for a simultaneous usage of multiple GP called ModelPool was created with several possible options for the selection of the best model. Experiments dedicated to choosing the best set of GP models were performed. Another experiments were done to find the right settings for the ModelPool class.
From the performed experiments, multiple conclusions were made for the noiseless functions:

- The SE ARD covariance function was causing the problems in performance.
- Overall performance of GP models with the linear mean function was worse than GP models with the constant mean function.
- The GPs chosen from a set of 96 combinations were succesful in creating a visible improvement when compared to the best of previous experiments in the project.

Similiar experiments were done for the noisy functions. In this case the results of GP models with SE ARD covariance function and linear mean function were better and caused improvements as well when compared to the best of previous experiments.

Bibliography

[1] Digabel, S. L. Blackbox Optimization: Algorithm and Applications. March 2014. Available from: https://www.gerad.ca/Sebastien.Le.Digabel/talks/2014_LANL_50mins.pdf

[2] von Stockar, U.; van der Wielen, L. A. M. *Biothermodynamics: The Role of Thermodynamics in Biochemical Engineering.* EFPL Press, 2013, ISBN 9781466582163.

[3] Cassioli, A. A Tutorial on Black-Box Optimization [online]. April 2013, [Cited 23.3.2017]. Available from: https://www.lix.polytechnique.fr/~dambrosio/blackbox_material/Cassioli_1.pdf

[4] Hansen, N. The CMA Evolution Strategy: A Tutorial. 2016: pp. 4–16, [Cited 22.3.2017], arXiv:1604.00772. Available from: https://arxiv.org/pdf/1604.00772v1

[5] Hansen, N.; Auger, A.; et al. Real-parameter black-box optimization benchmarking 2010: Presentation of the noisy functions. *Working Paper 2009/21*, 2010: pp. 5,20–21, [Cited 31.3.2017]. Available from: http://coco.lri.fr/downloads/download15.01/bbobdocnoisyfunctions.pdf

[6] Hansen, N.; Auger, A.; et al. Real-parameter black-box optimization benchmarking 2010: Presentation of the Noiseless Functions. *Working Paper 2009/20*, 2010: pp. 2,33–34, [Cited 31.3.2017]. Available from: http://coco.lri.fr/downloads/download15.01/bbobdocfunctions.pdf

[7] Concept of directional optimization in CMA-ES algorithm [online]. [Cited 22.3.2017]. Available from: https://en.wikipedia.org/wiki/CMA-ES#/media/File:Concept_of_directional_optimization_in_CMA-ES_algorithm.png

[8] Buche, D.; Schraudolph, N. N.; et al. Accelerating evolutionary algorithms with Gaussian process fitness function models. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, volume 35, no. 2, 2005: pp. 185–194, [Cited 5.4.2017]. Available from: `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1424193`

[9] Bajer, L.; Holeňa, M. Model Guided Sampling Optimization for Low-dimensional Problems. *CoRR*, volume abs/1508.07741, 2015, [Cited 5.4.2017]. Available from: `http://arxiv.org/abs/1508.07741`

[10] Documentation for GPML Matlab Code version 4.0 [online]. [Cited 31.3.2017]. Available from: `http://www.gaussianprocess.org/gpml/code/matlab/doc/`

[11] Bajer, L.; Charypar, V.; et al. Model guided sampling optimization with gaussian processes for expensive black box optimization. 2013: pp. 1715–1716, doi:10.1145/2464576.2480794, [Cited 5.4.2017]. Available from: `http://doi.acm.org/10.1145/2464576.2480794`

[12] Rasmussen, C. E.; Williams, C. K. I. *Gaussian processes for machine learning*. Cambridge, Mass.: MIT Press, c2006, ISBN 026218253x, 8–31 pp., [Cited 5.4.2017]. Available from: `http://www.gaussianprocess.org/gpml/chapters/RW.pdf`

[13] Bajer, L. Surrogate CMA-ES [online]. Available from: `https://github.com/bajeluk/surrogate-cmaes/`

[14] COmparing Continuous Optimisers [online]. January 2016, [Cited 22.3.2017]. Available from: `http://coco.gforge.inria.fr`

Appendix **A**

Acronyms

ARD Automatic Relevance Determination

BBOB Black-Box Optimization Benchmarking

CMA-ES Covariance Matrix Adaptation Evolution Strategy

COCO Comparing Continuous Optimisers

GP Gaussian Process

MAE Mean Absolute Error

RDE Ranking Difference Error

(R)MSE (Root) Mean Squared Error

SE Squared Exponential

Appendix B

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
├── surrogate-cmaes .................... the directory with implementation
│   ├── src ...................... the directory with MATLAB source codes
│   └── exp .................. the directory with definitions of experiments
├── reports .......................... the directory with generated reports
├── datasets ......... the directory with datasets used for some experiments
└── text ......................................... the thesis text directory
    ├── latex ............. the directory of LaTeX source codes of the thesis
    └── DP_Juranko_Jan_2017.pdf ............ the thesis text in PDF format
```

Appendix C

Tables of GpModels chosen for the ModelPool experiments

In case of the 'nearestToPopulation' trainsetType, 'nearestToPop' is shown in the tables to decrease the width of the tables.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covMaterniso, 5}'	'parameters'	$10 * \sigma*$	'15*dim'	'meanConst'

Table C.1: GpModel in the experiment exp_doubleEC_23.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covMaterniso, 5}'	'parameters'	not used	'15*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	0.999	'20*dim'	'meanConst'
'{@covSEiso}'	'nearest'	0.999	'15*dim'	'meanConst'

Table C.2: 3 GpModels chosen for the experiment exp_doubleEC_23_pool.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covMaterniso, 5}'	'clustering'	0.999	'15*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	0.999	'20*dim'	'meanConst'
'{@covSEiso}'	'nearest'	0.999	'15*dim'	'meanConst'

Table C.3: 3 GpModels chosen for the experiment exp_doubleEC_23_pool_diff.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEiso}'	'nearest'	1	'20*dim'	'meanConst'
'{@covSEard}'	'clustering'	1	'5*dim'	'meanConst'
'{@covSEard}'	'clustering'	1	'10*dim'	'meanConst'
'{@covSEard}'	'allPoints'	0.999	'5*dim'	'meanConst'
'{@covSEard}'	'clustering'	1	'5*dim'	'meanLinear'
'{@covSEiso}'	'allPoints'	0.999	'5*dim'	'meanLinear'
'{@covMaterniso,3}'	'nearest'	1	'20*dim'	'meanConst'
'{@covMaterniso,5}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso,3}'	'allPoints'	0.999	'20*dim'	'meanConst'
'{@covMaterniso,5}'	'clustering'	1	'10*dim'	'meanLinear'
'{@covMaterniso,3}'	'clustering'	1	'15*dim'	'meanLinear'

Table C.4: 11 GpModels chosen by the RDE for the 2D noiseless functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEard}'	'clustering'	1	'20*dim'	'meanConst'
'{@covSEiso}'	'clustering'	1	'5*dim'	'meanLinear'
'{@covSEard}'	'clustering'	0.999	'5*dim'	'meanLinear'
'{@covMaterniso, 3}'	'nearest'	1	'5*dim'	'meanLinear'
'{@covMaterniso, 3}'	'nearest'	1	'20*dim'	'meanLinear'
'{@covMaterniso, 5}'	'nearest'	0.999	'15*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'5*dim'	'meanLinear'
'{@covMaterniso, 5}'	'allPoints'	1	'20*dim'	'meanLinear'
'{@covMaterniso, 3}'	'clustering'	1	'10*dim'	'meanConst'
'{@covMaterniso, 3}'	'clustering'	0.999	'10*dim'	'meanConst'
'{@covMaterniso, 3}'	'clustering'	0.999	'5*dim'	'meanLinear'
'{@covSEard}'	'nearestToPop'	1	'15*dim'	'meanLinear'

Table C.5: 12 GpModels chosen by the RDE for the 5D noiseless functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEard}'	'nearest'	1	'5*dim'	'meanLinear'
'{@covSEard}'	'nearest'	1	'10*dim'	'meanLinear'
'{@covSEard}'	'nearest'	1	'20*dim'	'meanLinear'
'{@covSEiso}'	'nearest'	0.999	'10*dim'	'meanConst'
'{@covSEard}'	'allPoints'	1	'5*dim'	'meanLinear'
'{@covSEard}'	'clustering'	1	'5*dim'	'meanLinear'
'{@covSEiso}'	'clustering'	1	'20*dim'	'meanLinear'
'{@covSEiso}'	'allPoints'	0.999	'5*dim'	'meanLinear'
'{@covSEard}'	'allPoints'	0.999	'15*dim'	'meanLinear'
'{@covMaterniso, 3}'	'nearest'	1	'5*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'20*dim'	'meanLinear'
'{@covMaterniso, 3}'	'clustering'	1	'10*dim'	'meanConst'
'{@covMaterniso, 5}'	'clustering'	1	'15*dim'	'meanConst'
'{@covMaterniso, 3}'	'nearestToPop'	1	'10*dim'	'meanConst'

Table C.6: 15 GpModels chosen by the RDE for the 10D noiseless functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEiso}'	'nearest'	1	'10*dim'	'meanLinear'
'{@covSEard}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covSEiso}'	'clustering'	0.999	'5*dim'	'meanConst'
'{@covSEard}'	'clustering'	0.999	'20*dim'	'meanConst'
'{@covMaterniso, 5}'	'nearest'	1	'20*dim'	'meanLinear'
'{@covMaterniso, 5}'	'nearest'	0.999	'5*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'5*dim'	'meanLinear'
'{@covMaterniso, 5}'	'clustering'	1	'15*dim'	'meanConst'
'{@covMaterniso, 3}'	'clustering'	0.999	'10*dim'	'meanConst'
'{@covMaterniso, 3}'	'clustering'	0.999	'10*dim'	'meanLinear'
'{@covSEard}'	'nearestToPop'	1	'20*dim'	'meanLinear'
'{@covMaterniso, 3}'	'nearestToPop'	1	'15*dim'	'meanLinear'

Table C.7: 13 GpModels chosen by the MSE for the 2D noiseless functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEiso}'	'allPoints'	0.999	'20*dim'	'meanConst'
'{@covSEard}'	'allPoints'	0.999	'15*dim'	'meanLinear'
'{@covMaterniso, 3}'	'nearest'	1	'5*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'5*dim'	'meanLinear'
'{@covMaterniso, 3}'	'allPoints'	1	'15*dim'	'meanLinear'
'{@covMaterniso, 5}'	'clustering'	0.999	'20*dim'	'meanConst'
'{@covMaterniso, 5}'	'clustering'	0.999	'15*dim'	'meanLinear'
'{@covSEard}'	'nearestToPop'	1	'5*dim'	'meanConst'
'{@covSEard}'	'nearestToPop'	1	'20*dim'	'meanLinear'
'{@covMaterniso, 3}'	'nearestToPop'	1	'20*dim'	'meanConst'

Table C.8: 11 GpModels chosen by the MSE for the 5D noiseless functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEard}'	'nearest'	1	'20*dim'	'meanConst'
'{@covSEard}'	'nearest'	1	'15*dim'	'meanConst'
'{@covSEard}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso, 3}'	'nearest'	1	'5*dim'	'meanConst'
'{@covSEard}'	'clustering'	0.999	'20*dim'	'meanConst'
'{@covMaterniso, 3}'	'nearest'	1	'5*dim'	'meanLinear'
'{@covMaterniso, 3}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'20*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'20*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	0.999	'20*dim'	'meanConst'
'{@covMaterniso, 3}'	'clustering'	0.999	'15*dim'	'meanLinear'
'{@covMaterniso, 3}'	'nearestToPop'	1	'15*dim'	'meanConst'

Table C.9: 13 GpModels chosen by the MSE for the 10D noiseless functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEard}'	'nearest'	1	'5*dim'	'meanConst'
'{@covSEard}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covSEiso}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covSEiso}'	'allPoints'	1	'5*dim'	'meanLinear'
'{@covMaterniso, 5}'	'nearestToPop'	1	'10*dim'	'meanLinear'
'{@covSEard}'	'clustering'	1	'5*dim'	'meanLinear'

Table C.10: 5 GpModels chosen by the RDE for the 2D noisy functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEard}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covSEiso}'	'allPoints'	1	'5*dim'	'meanLinear'
'{@covMaterniso, 3}'	'nearest'	1	'10*dim'	'meanConst'
'{@covMaterniso, 5}'	'nearest'	1	'20*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'20*dim'	'meanLinear'
'{@covMaterniso, 3}'	'clustering'	0.999	'5*dim'	'meanConst'
'{@covSEard}'	'nearestToPop'	0.999	'20*dim'	'meanLinear'

Table C.11: 7 GpModels chosen by the RDE for the 5D noisy functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEard}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covSEard}'	'nearest'	0.999	'10*dim'	'meanLinear'
'{@covMaterniso, 3}'	'nearest'	1	'10*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'15*dim'	'meanConst'

Table C.12: 5 GpModels chosen by the RDE for the 10D noisy functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEard}'	'nearest'	1	'5*dim'	'meanLinear'
'{@covSEiso}'	'allPoints'	1	'5*dim'	'meanLinear'
'{@covSEard}'	'clustering'	0.999	'5*dim'	'meanLinear'
'{@covMaterniso, 5}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'15*dim'	'meanConst'
'{@covMaterniso, 5}'	'clustering'	1	'5*dim'	'meanConst'
'{@covMaterniso, 3}'	'clustering'	0.999	'10*dim'	'meanLinear'

Table C.13: 7 GpModels chosen by the MSE for the 2D noisy functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEard}'	'nearest'	1	'10*dim'	'meanConst'
'{@covSEard}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covSEiso}'	'allPoints'	0.999	'5*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'10*dim'	'meanLinear'
'{@covMaterniso, 5}'	'allPoints'	1	'15*dim'	'meanLinear'

Table C.14: 6 GpModels chosen by the MSE for the 5D noisy functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covMaterniso, 5}'	'nearest'	1	'10*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'5*dim'	'meanLinear'
'{@covMaterniso, 3}'	'allPoints'	1	'10*dim'	'meanLinear'
'{@covMaterniso, 5}'	'allPoints'	1	'10*dim'	'meanLinear'
'{@covMaterniso, 5}'	'nearestToPop'	1	'5*dim'	'meanConst'
'{@covMaterniso, 5}'	'nearestToPop'	1	'20*dim'	'meanLinear'

Table C.15: 6 GpModels chosen by the MSE for the 10D noisy functions from the set of 256 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEiso}'	'nearest'	1	'20*dim'	'meanConst'
'{@covSEard}'	'clustering'	1	'5*dim'	'meanConst'
'{@covSEard}'	'clustering'	1	'10*dim'	'meanConst'
'{@covSEiso}'	'allPoints'	0.999	'20*dim'	'meanConst'

Table C.16: 4 GpModels chosen for the experiment exp_doubleEC_25.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEiso}'	'nearest'	1	'20*dim'	'meanConst'
'{@covMaterniso, 3}'	'nearest'	1	'20*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covSEiso}'	'allPoints'	0.999	'20*dim'	'meanConst'

Table C.17: 4 GpModels chosen for the experiment exp_doubleEC_26.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEiso}'	'nearest'	1	'5*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covMaterniso, 3}'	'clustering'	0.999	'20*dim'	'meanConst'
'{@covMaterniso, 5}'	'nearestToPop'	0.999	'5*dim'	'meanConst'

Table C.18: 5 GpModels chosen by the RDE for the 2D noiseless functions from the set of 96 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covMaterniso, 3}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'15*dim'	'meanConst'
'{@covMaterniso, 5}'	'clustering'	0.999	'20*dim'	'meanConst'
'{@covMaterniso, 3}'	'nearestToPop'	1	'5*dim'	'meanConst'

Table C.19: 5 GpModels chosen by the RDE for the 5D noiseless functions from the set of 96 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covMaterniso, 5}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covMaterniso, 5}'	'clustering'	0.999	'15*dim'	'meanConst'
'{@covMaterniso, 3}'	'nearestToPop'	1	'20*dim'	'meanConst'

Table C.20: 3 GpModels chosen by the RDE for the 10D noiseless functions from the set of 96 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covMaterniso, 3}'	'nearest'	1	'20*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'10*dim'	'meanConst'

Table C.21: 3 GpModels chosen by the RDE for the 2D noisy functions from the set of 96 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covSEiso}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covSEiso}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covSEiso}'	'clustering'	1	'15*dim'	'meanConst'
'{@covMaterniso, 3}'	'nearest'	1	'10*dim'	'meanConst'
'{@covMaterniso, 5}'	'nearest'	1	'20*dim'	'meanConst'
'{@covMaterniso, 3}'	'allPoints'	1	'15*dim'	'meanConst'

Table C.22: 6 GpModels chosen by the RDE for the 5D noisy functions from the set of 96 GpModels.

covFcn	trainsetType	trainRange	trainsetSizeMax	meanFcn
'{@covMaterniso, 3}'	'allPoints'	1	'5*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'10*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	1	'15*dim'	'meanConst'
'{@covMaterniso, 5}'	'allPoints'	0.999	'15*dim'	'meanConst'
'{@covMaterniso, 5}'	'nearestToPop'	1	'15*dim'	'meanConst'

Table C.23: 5 GpModels chosen by the RDE for the 10D noisy functions from the set of 96 GpModels.