EuroHPC Joint Undertaking

Data driven Computational Mechanics at EXascale

DCoMEX

Data driven Computational Mechanics at EXascale

Work program topic: EuroHPC-01-2019
Type of action: Research and Innovation Action (RIA)

Report:
Incremental versions of Korali, improving on performance, features, and documentation

DELIVERABLE D4.4

Version No 1

https://www.dcomex.eu
This project has received funding from the European High-Performance Computing Joint Undertaking ('the JU'), under Grant Agreement No 956201

DOCUMENTATION SUMMARY INFORMATION

Project Title	**Data driven Computational Mechanics at EXascale**
Project Acronym	DCoMEX
Project No	956201
Call Identifier	EuroHPC-01-2019
Project Start Date	01.04.2021
Related work package	WP 4
Related task(s)	Task 4.4
Lead Organisation	ETHZ
Submission date	25.11.2023
Re-submission date	
Dissemination Level	PU

Quality Control:

	Who	**Affiliation**	**Date**
Checked by internal reviewer	Petros Koumoutsakos		07.12.2023
Checked by WP Leader	Eleni Chatzi	ETHZ	09.12.2023
Checked by Project Coordinator	Vissarion Papadopoulos	NTUA	

Document Change History:

Version	**Date**	**Author(s)**	**Affiliation**	**Comment**
1.0	25.11.2023	Sergey Litvinov, Sebastian Kaltenbach	ETHZ	

Deliverable 4.4

In this report, we present the final outcomes of Deliverable 4.4, documenting the progressive versions of Korali developed throughout the project and showcasing their enhancements in performance, features, and documentation. The report is divided into three parts. The initial section briefly discusses the implemented optimization methods integrated into Korali during the project, providing rationale for their necessity and presenting benchmark test cases. In the subsequent section (Section 2), a similar approach is taken for the methods related to Bayesian inference, while Section 3 is dedicated to the documentation of Korali. Conclusively, this report summarizes the achieved objectives and offers an insight into how Korali will significantly contribute to the final tasks remaining in the DCoMEX project.

Stochastic Optimization

During the DCoMEX project, we identified the need for a stochastic optimization method that is capable of finding an optimum without relying on gradient information. We have chosen to use the CMA-ES [1] which has generated promising results in various applications. The CMA-ES method is an evolutionary strategy that is based on using a multivariate normal distribution and iterative adapting its covariance matrix (as well as mean) during the optimization. We discovered that this method has limitations in case of mixed-integer problems and thus also implemented a second, very similar method, Distance-weighted eXponential Natural Evolution Strategy taking account of Implicit Constraint and Integer (DX-NES-ICI), that is more suitable for mixed-integer problems [2]. The methods have been presented in detail in the last Deliverable D4.3. In the following, we are showing two benchmark studies and the obtained results with applying both mentioned methods:

Benchmark 1: Rosenbrock function

We test the two mentioned optimization algorithms using the four-dimensional Rosenbrock function:

$$f(x_1, x_2, x_3, x_4) = \sum_{i=1}^{2} \left(100(x_{2i-1}^2 - x_{2i})^2 + (x_{2i-1} - 1)^2\right) \tag{1}$$

For both algorithms we used the default parameter-values as reported in [1] and [2] as well as a population size of 100. In 10 runs each, both methods converged to the correct solution each time which is $(1, 1, 1, 1)$

Method	x_1	x_2	x_3	x_4
CMA-ES	1.00	1.00	1.00	1.00
DX-NES-ICI	1.00	1.00	1.00	1.00
True Optimum	1.00	1.00	1.00	1.00

Table 1: Found optimum using the two proposed methods averaged over 10 runs each

Benchmark 2: Reversed Ellipsoid

As a second benchmark case, we use the reversed ellipsoid example as presented in [2]. Here, N_{int} is the dimension of the integer variables involved and N_{co} the dimension of the continuous variables. The total dimension is the summation of these two quantities.

$$f(\boldsymbol{x}) = \sum_{i=1}^{N_int} \left(1000^{(j-1)/(N-1)} x_{int,j}\right)^2 + \sum_{i=1}^{N_co} \left(1000^{(N_{int}+j-1)/(N-1)} x_{co,j}\right)^2 \tag{2}$$

As this is a mixed-integer optimization problem, CMA-ES can not be used. Our second method was able to identify the correct optimum in 10 out of 10 runs. We thus conclude that this method is suitable for solving mixed-integer optimization problems and complements our existing CMA-ES implementation The correct optimum found for this 80 dimensional test case is zero for all continuous and discrete variables involved and was correctly identified by DX-NES-ICI.

Bayesian Inference

During the DCoMEX project, we identified the need for an efficient algorithm for Bayesian inference and have chosen TMCMC. This flexible and efficient method has been so far successfully applied to all Bayesian inference problems within the DCoMEX project. The algorithm has already been described in detail in previous reports and we are here presenting a benchmark study. Moreover, we also created a Colab Notebook which is in detail described in the next section to show potential users of our software how the implemented TMCMC algorithm can be applied to their Bayesian inference problem.

Benchmark:

A a test case, we inferred the parameters of a quadratic regression model from noisy data using only a small amount of data. We observed that as expected the accuracy increased if more data was added, whereas for smaller amount of data, the estimated noise level was too high. For our final run using 500 data points, we were able to identify all four parameters of the model.

$$y_i = a + bx_i + cx_i^2 + \sigma\epsilon_i \tag{3}$$

Here ϵ_i are i.i.d. samples from a standard normal distribution. This leads to the following conditional distribution:

$$p(y_i|x_i, a, b, c, \sigma) = \mathcal{N}(a + bx_i + cx_i^2, \sigma^2) \tag{4}$$

Combined with uninformative uniform priors, we arrive at the posterior and can sample from this posterior using TMCMC.

We tried to infer the parameters based on 5,50 and 500 data points and obtained the following results: We moreover are

Number of Data points	a	b	c	σ
5	1.93	3.06	0.51	0.14
50	1.95	3.13	0.50	0.11
500	2.00	2.98	0.50	0.10
True Parameters	2.00	3.00	0.50	0.10

Table 2: Inferred Posterior Mean of the parameters. Values are calculated by averaging over 10 TMCMC runs with 2000 samples each

providing the learned posterior distribution in the following figures. As expected the shape of the posterior converges to a Normal distribution for the parameters for a large amount of data points, whereas for a very small amount of data points the shape of the posterior is more complex.

Documentation

We have focused on providing a good documentation of out implementation right from the start of this project. The documentation was already submitted as part of the report for an earlier deliverable and further extended since. It can be accessed at `https://dcomex-framework-prototype.readthedocs.io/en/latest/`.

To enhance our documentation and broaden Korali's accessibility, we've developed a pre-built version tailored for easy installation in Google Colab. This version prioritizes performance, harnessing native libraries and fine-tuning specifically for the Google Colab environment (Ubuntu 22, LTS), ensuring optimal functionality on this and similar platforms.

Additionally, we've created two Colab notebooks showcasing stochastic optimization and MCMC using Korali. Both notebooks, along with their outputs, are appended to this report in Appendix A and B. These resources aim to further enrich the user experience and facilitate easier utilization of Korali for a wider audience.

Conclusion and Outlook

We have extended Korali to have the necessary capabilities to be applied to all required application within the DCoMEX project. Moreover, we have significantly improved the documentation and provided a very easy way to install the software using a pre-built version as well as two Google Colab that explain some of the key functions of Korali in less than 100 lines of code each.
We are looking forward to applying the algorithm implemented in Korali to the DCoMEX application areas.

References

[1] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pages 312–317. IEEE, 1996.

[2] Koki Ikeda and Isao Ono. Natural evolution strategy for mixed-integer black-box optimization. *arXiv preprint arXiv:2304.10724*, 2023.

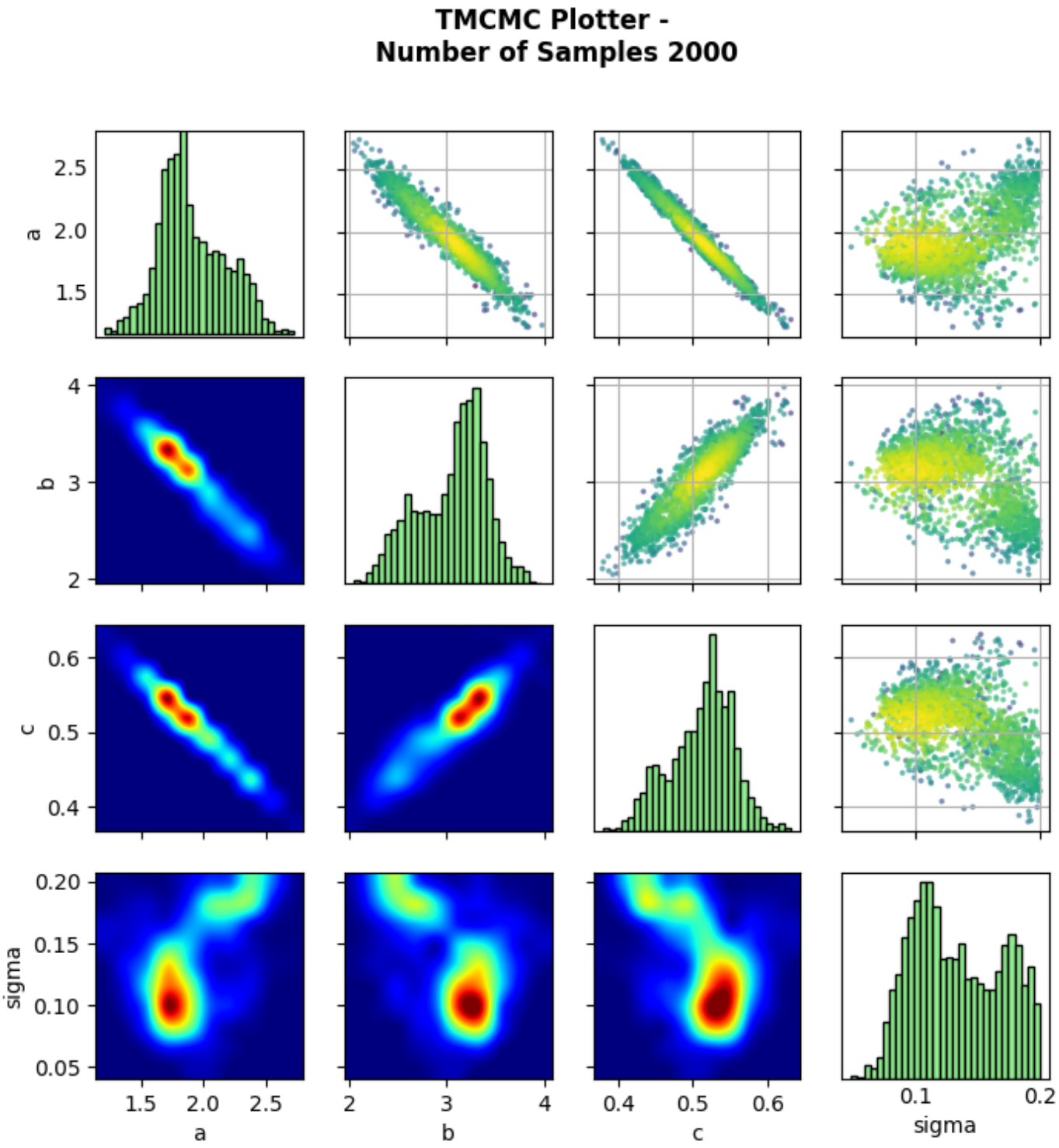


Figure 1: Obtained posterior distribution using 5 noisy data points only.

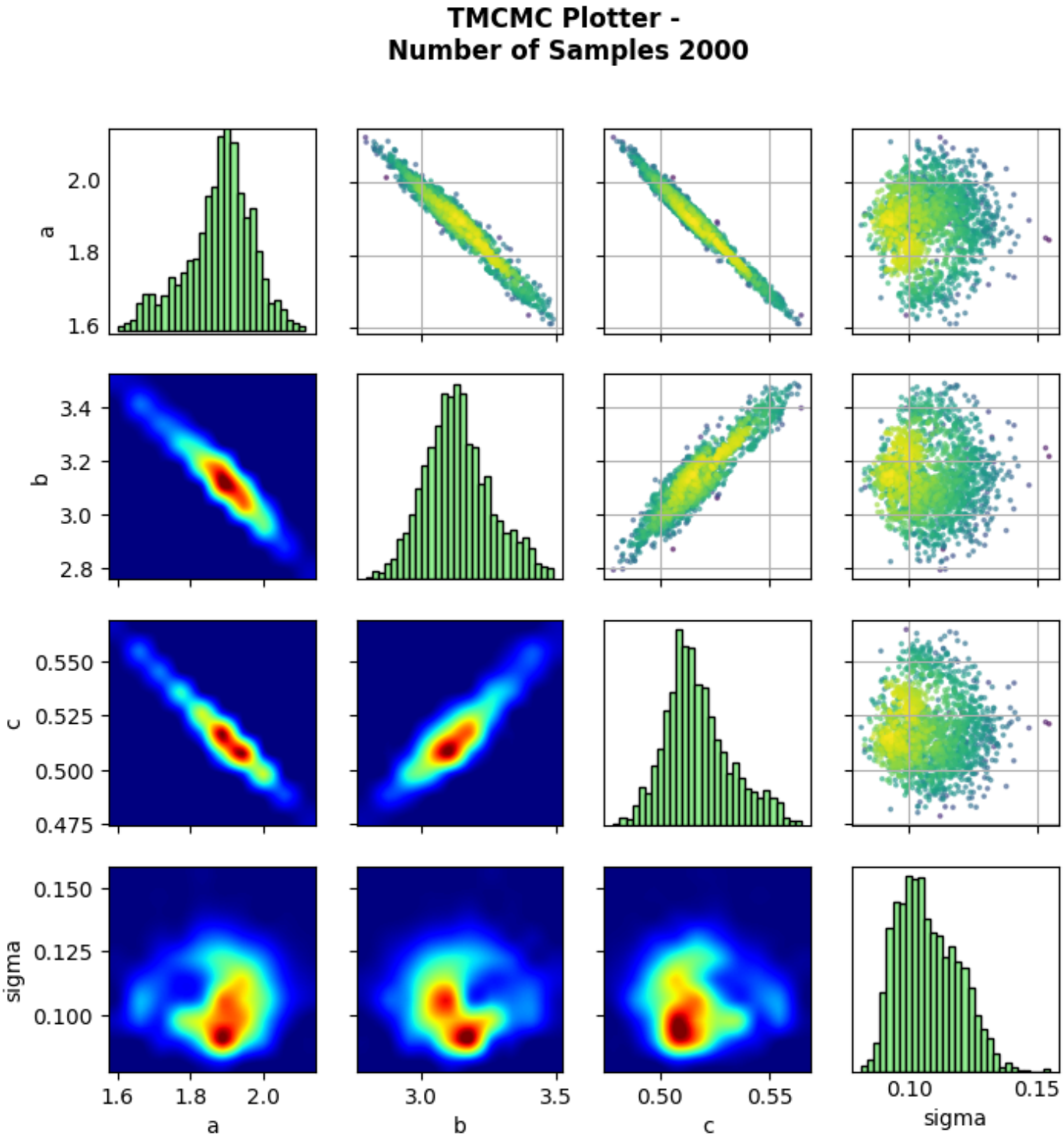


Figure 2: Obtained posterior distribution using 50 noisy data points only.

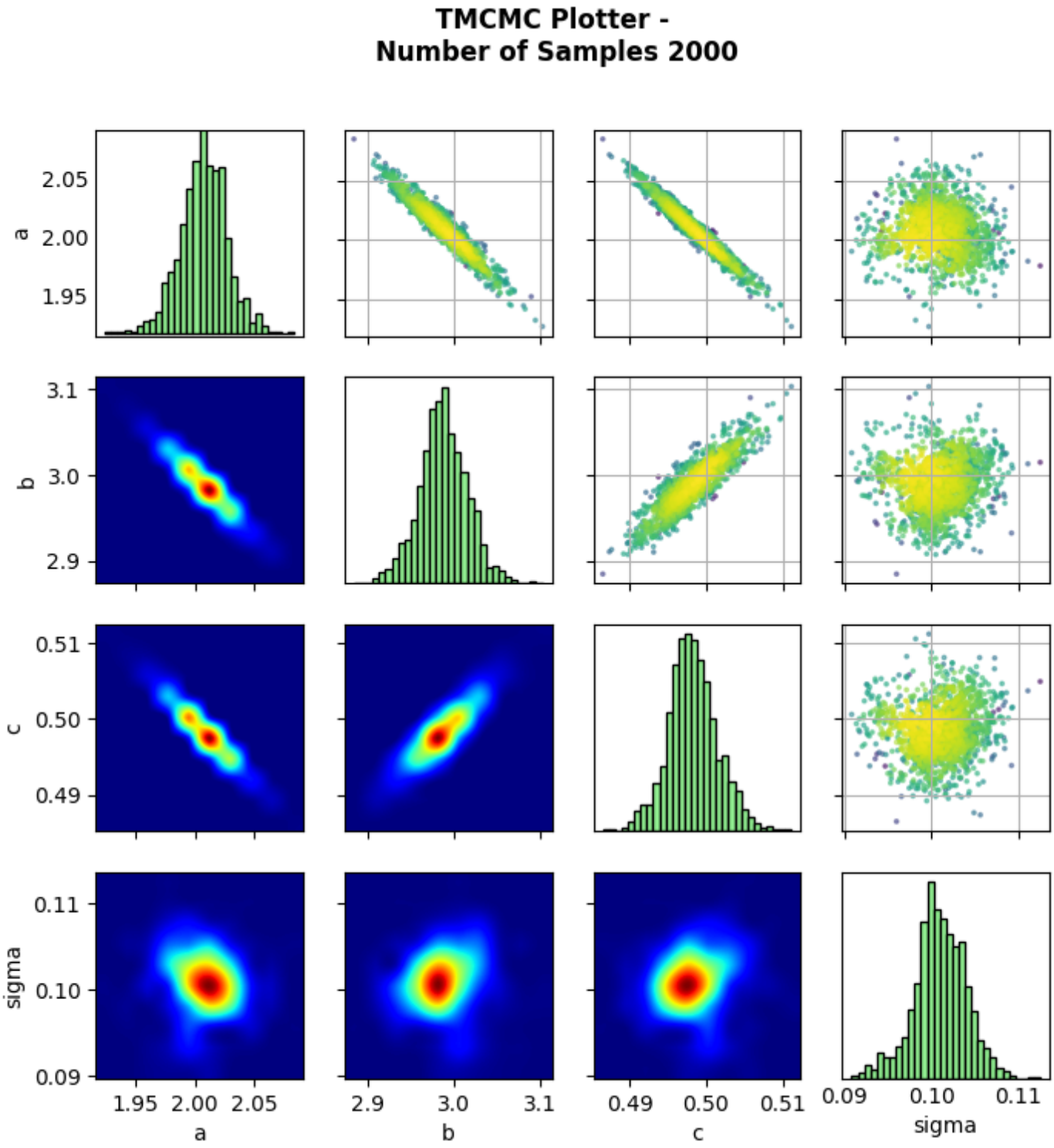


Figure 3: Obtained posterior distribution using 500 noisy data points only.

Appendix A: Colab Inference

This Appendix contains the Colab notebook which shows and explains how to do Bayesian Inference with Korali in less than 100 lines of code.
The Colab can also be accessed directly here:
`https://colab.research.google.com/drive/1yYn52RVyAA46a2niEXwim3fYHZh5ahG-?usp=sharing`

```
%pip install -q --progress-bar off  https://github.com/slitvinov/dcomex-framewor
```

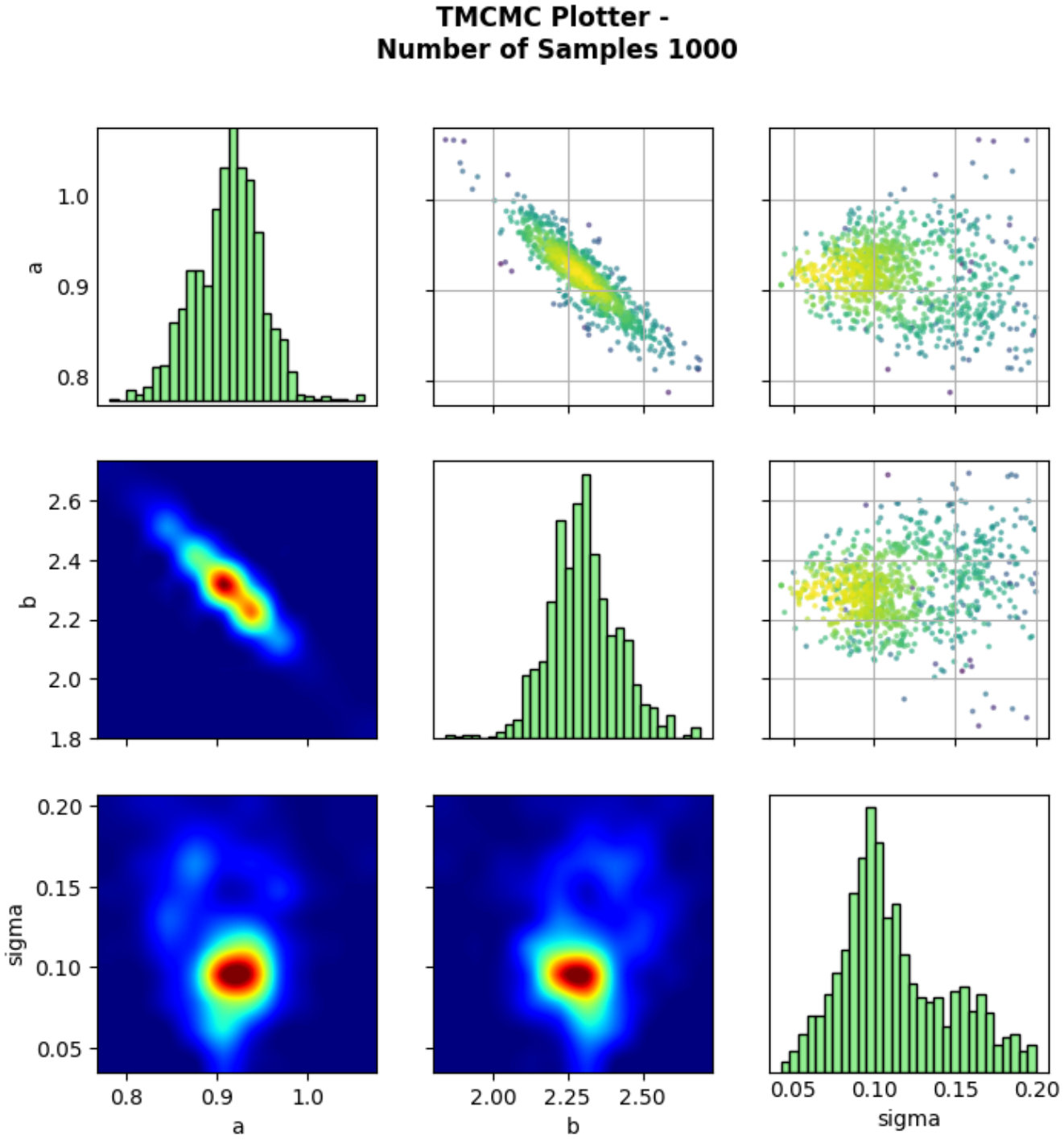
We are applying the TMCMC algorithm to a linear regression. The paraemter-space is three dimensional and the likelihood is a Gaussian. The mathematical details are presented below. The example can be modfied by changing the likelihood as well as adjusting the priors.

The model employed is a linear regressison model with a Gaussian noise ϵ:
$y=ax+b+\sigma\,\epsilon$
For the three parameters a,b, and σ we are each assuming a uniform prior distribution with appropriate bounds. To infer the posterior of the paremters we have collected 5 data points (x,y) which are used within the likelihood. We note that we suggest to specify the loglikelihood isntead of the likelihood.

```
import korali
import korali.plot.__main__
import statistics
import math
def model(s):
    a, b, sig = s["Parameters"]
    ssq = sig**2
    s["logLikelihood"] = -0.5 * len(x) * math.log(
        2 * math.pi * ssq) - 0.5 * statistics.fsum(
            (a * x + b - y)**2 for x, y in zip(x, y)) / ssq
x = [1.0, 2.0, 3.0, 4.0, 5.0]
y = [3.21, 4.14, 4.94, 6.06, 6.84]
e = korali.Experiment()
e["Problem"] = {"Type": "Bayesian/Custom", "Likelihood Model": model}
e["Solver"] = {"Type": "Sampler/TMCMC", "Population Size": 1000}
for i, (v, lo, hi) in enumerate(
    (("a", 0, 2), ("b", 0, 3), ("sigma", 1e-6, 0.2))):
    e["Distributions"][i] = {
        "Name": v,
        "Type": "Univariate/Uniform",
        "Minimum": lo,
        "Maximum": hi
    }
    e["Variables"][i] = {"Name": v, "Prior Distribution": v}
e["File Output"]["Path"] = "_korali_result_tmcmc"
k = korali.Engine()
k.run(e)
a, b, sig = zip(*e["Results"]["Posterior Sample Database"])
print("posterio mean: ", statistics.fmean(a), statistics.fmean(b),
```

```
      statistics.fmean(sig))
korali.plot.__main__.main(e["File Output"]["Path"], False, "");
```

```
posterio mean:  0.9161181843414956 2.3019057054827403 0.11266113186804792
```



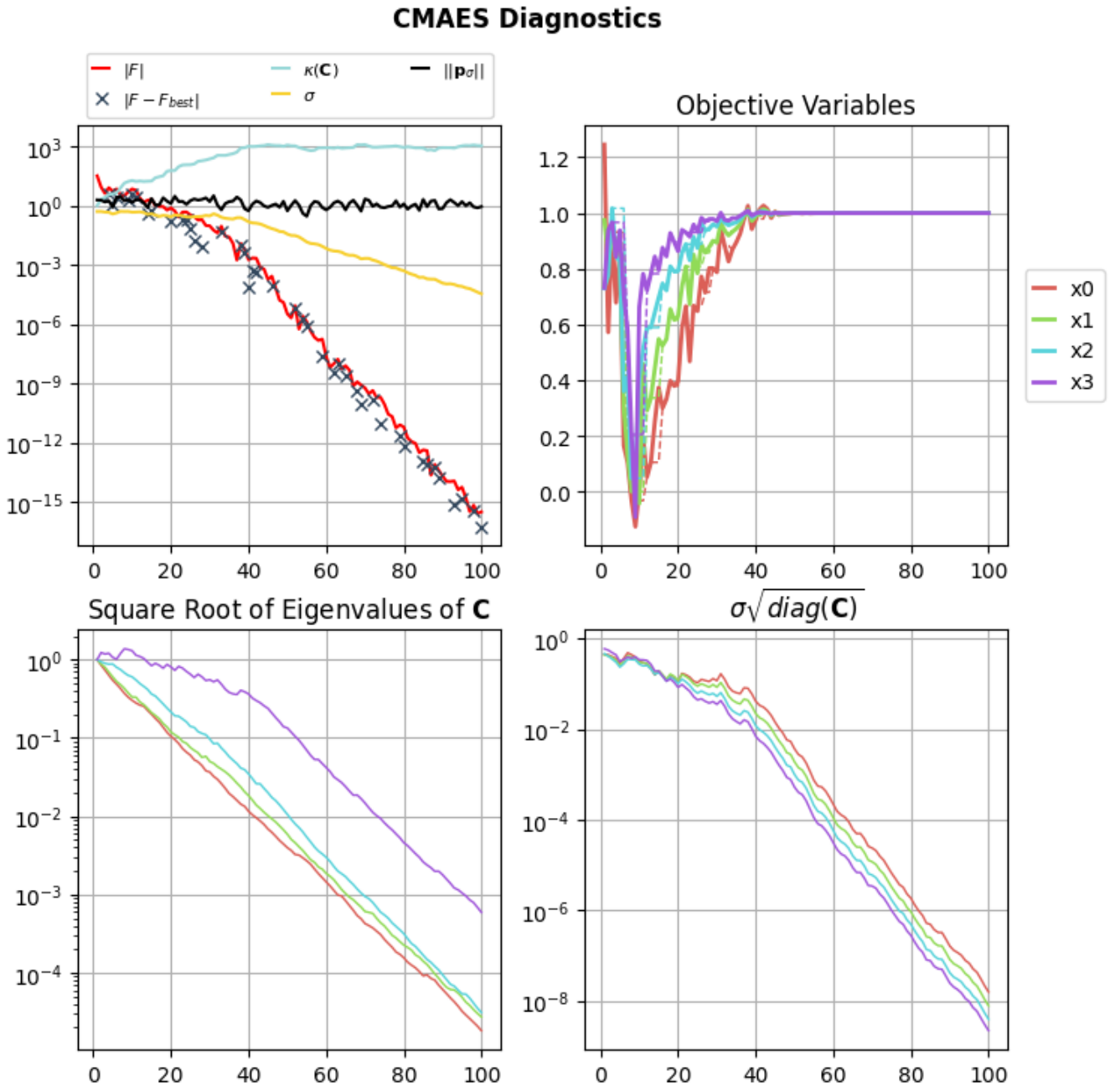
Appendix B: Colab Optimization

This Appendix contains the Colab notebook which shows and explains how to do Optimization with Korali in less than 100 lines of code.
The Colab can also be accessed directly here:
`https://colab.research.google.com/drive/12KIs3VyWavHV4ubkTQ0-IQuUtnMWhnoY?usp=sharing`

```
%pip install -q --progress-bar off  https://github.com/slitvinov/dcomex-framewor
```

We are using Korali to optimize the Rosenbrock function. This function could be replaced by any other function with the same structure, i.e. the input parameters as input to the function and a scalar value as the output. In our case we are dealing with a four-dimensional parameter space and the optimum is located at (1,1,1,1).

```
import math
import korali
import korali.plot.__main__
def negative_rosenbrock(p):
    x = p["Parameters"]
    res = math.fsum(100 * (x - y**2)**2 + (1 - x)**2 for x, y in zip(x, x[1:]))
    p["F(x)"] = -res
k = korali.Engine()
e = korali.Experiment()
e["Random Seed"] = 0xC0FEE
e["Problem"]["Type"] = "Optimization"
e["Problem"]["Objective Function"] = negative_rosenbrock
dim = 4
for i in range(dim):
    e["Variables"][i]["Name"] = "x" + str(i)
    e["Variables"][i]["Initial Value"] = 1.0
    e["Variables"][i]["Initial Standard Deviation"] = 1.0 / math.sqrt(dim)
e["Solver"]["Type"] = "Optimizer/CMAES"
e["Solver"]["Population Size"] = 32
e["Solver"]["Mu Value"] = 8
e["Solver"]["Termination Criteria"]["Max Generations"] = 100
e["File Output"]["Enabled"] = True
e["File Output"]["Path"] = '_korali_result_cmaes'
e["File Output"]["Frequency"] = 1
k.run(e)
korali.plot.__main__.main(e["File Output"]["Path"], False, "");
```



The results are shown together with diagnostics of CMA-ES. In the upper right corner it, the convergence of all four input parameters is shown. In the lower left corner the decrease of the square root of the covariance matrix indicates succesfull convergence similar to the product of scaling parameter and square-root of the diagonal which is shown in the lower right.