Exascale **Q**uantification of **U**ncertainties for **Te**chnology and Science Simulation

D6.1 Deterministic optimization software

Document information table

Contract number:	800898
Project acronym:	ExaQUte
Project Coordinator:	CIMNE
Document Responsible Partner:	TUM
Deliverable Type:	Report, Other
Dissemination Level:	Public
Related WP & Task:	WP6, Task 6.1
Status:	Final version

Authoring

Prepared by:				
Authors	Partner	Modified Page/Sections	Version	Comments
Andreas Apostolatos	TUM			
Brendan Keith	TUM			
Contributors				

Change Log

Versions	Modified Page/Sections	Comments

Approval

Approved by:				
	Name	Partner	Date	OK
Task leader	Roland Wüchner	TUM		OK
WP leader	Andreas Apostolatos	EPFL		OK
Coordinator	Riccardo Rossi	CIMNE		OK

Table of contents

List of Figures

Nomenclature / Acronym list

Acronym	**Meaning**
API	Application Programming Interface
ExaQUte	EXAscale Quantification of Uncertainties for Technology and Science Simulation
QoI	Quantity of Interest
MC	Monte Carlo
MLMC	Multilevel Monte Carlo method
C-MLMC	Continuation Multilevel Monte Carlo method
HPC	High performance computing
PDE	Partial differential equation

1 Introduction

This deliverable focuses on the implementation of deterministic optimization algorithms and problem solvers within KRATOS open-source software. One of the main challenges of optimization algorithms in Finite-Element based optimization is how to get the gradient of response functions which are used as objective and constraints when this is not available in an explicit form. The idea is to use local sensitivity analysis to get the gradient of the response function(s)

2 Sensitivity analysis

Sensitivity analysis [4, 7, 8], plays an important role in optimization: It can be used to study the effects of the variation of the input parameters onto the output parameters of the underlying problem. Herein, suitable sensitivity analysis methods for Finite-Element based models are employed: Those include direct or adjoint approaches for sensitivity analysis when the response function depends on the design variable(s) as well as on the state variable(s) which is the case for most responses in structural mechanics such as the nodal displacements, the stress field and the strain-energy relationship. Other types of responses may depend on an eigenvalue problems e.g. the eigenfrequency or the critical load factor are treated differently. The direct approach is beneficial when more response functions than design variables are encountered, because the number of the system evaluations depends on the number of the design variables. On the other hand, the adjoint approach it is beneficial when there are more design variables than response(s), because the number of system evaluations depends on the number of responses.

Finite Element-based optimization involves often more design variables than objectives/constraints e.g. in shape optimization. In KRATOS open-source software the discrete semi-analytic adjoint sensitivity analysis is employed.

Regarding the discrete approach, the response function is firstly discretized and then the derivatives with respect to the discrete parameters are evaluated.

On the contrary, the semi-analytic approach, the analytic procedure is performed up to a certain extent, which is typically confined in the element level, whereas the remaining parts of the sensitivity equation are approximated by a finite difference scheme. The advantages of the semi-analytic versus the analytic approach are,

- Implementation of analytic derivatives of element based data can be omitted, which is in general a very challenging task
- A generic code structure for different kinds of parameters is possible
- It can be used for any arbitrary element formulation and
- It is faster than the analytic approach

whereas the disadvantages in principle are,

- Reduced accuracy
- Challenging of finding an “optimal” disturbance measure.

The herein employed workflow for the adjoint sensitivity analysis in KRATOS includes,

- Solving the primal problem to determine the state variables
- Solving the adjoint problem for each response of the primal problem which presents the main effort in adjoint sensitivity analysis and which involves solving a linear system even for geometric nonlinear problems. The adjoint problem is independent of the traced design variables and it solely depends on the response. The solution of this problem are the so-called adjoint variables.
- Solving for the sensitivities in a post-processing step.

3 Optimization

The Vertex Morphing method is employed within the optimization solution procedure in KRATOS for the sensitivity filtering, see in [3, 6]. This method is especially important as in shape optimization multiple solutions may be apparent due to the non-convex nature of this kind of problems. Therefore, filtering is essential for obtaining a global solution in Finite Element-based optimization problems.

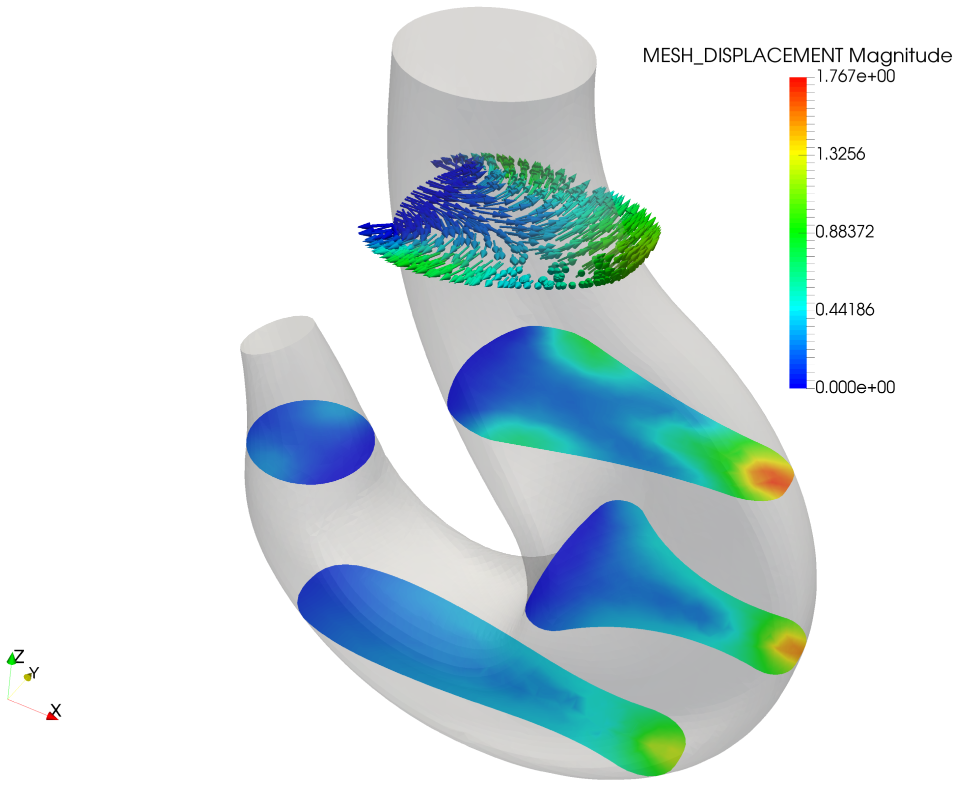

Figure 1: Mesh motion computed in KRATOS.

The implementation in KRATOS involves both unconstrained and constrained problems. Especially for the constrained problems, one constraint is at the moment available whereas up to ten constraints can be used in principle. Concerning the unconstrained method, a simple steepest decent approach with basic line search is included. Regarding the optimization with one constraint, a projection method is herein implemented [5]. In what concerns optimization with several constraints, an experimental trust region method is being implemented. The employed optimization algorithms can be applied to solid and shell structures.

The current implementation in KRATOS can handle millions of design variables. Different filter functions are included to control the surface continuity. Different filter strategies, both matrix-based and matrix-free, are included depending on the available computer resources. Details on how the filtering affects the design are described in [2].

Several objective functions are implemented, such as the strain-energy, eigenfrequency, mass, stresses etc. Sensitivity analysis is incorporated for each objective function. Regarding objectives which depend on design and state variables, gradient computation through semi-analytic adjoint sensitivity analysis is employed. Moreover, steady-state analysis is herein assumed.

Since the mesh is spatially deformed through the optimization algorithms, mesh motion techniques need to be considered. Herein, a pseudo-elastic behaviour is utilized (Fig. 1), which demonstrates nice properties when compared to other approaches such as solving a Laplace equation, see [9] for more details. Damping is included to directly impose geometric constraints like a fixed support during optimization.
The implementation of the deterministic algorithms in KRATOS named as *Shape Optimization App* is similar to software Dakota[1]. The implementation is done considering its application to relatively large cases and quite long computational times, which necessitates efficiency and robustness. The latter means that connections between different analyses may be considered (e.g. different adjoint analysis referencing to the results of a single primal analysis or two function values coming from one physical solution, like the lift or drag coefficients in aerodynamics). The latter concept called as *analyser-optimiser-communicator* (Fig. 2) allows for connections to external solvers, such as demonstrated in [1].

4 Example

This section demonstrates the aforementioned algorithms based on a solid structure example. The objective is the minimization of mass of a hook under the constraint of keeping the initial strain-energy (stiffness) of the structure constant (Fig 3). The employed algorithm is herein penalized projection. Figs. in 5 show significant improvement while accurately satisfying the constraint. It is worth noting that the geometry at the converged optimized shape is a very intuitive design. The latter can be explained by the fact that in order to reduce the mass while maintaining the stiffness, one should "extract" material from the bending axis which is exactly what the converged shape exhibits.

[1]`https://dakota.sandia.gov/`

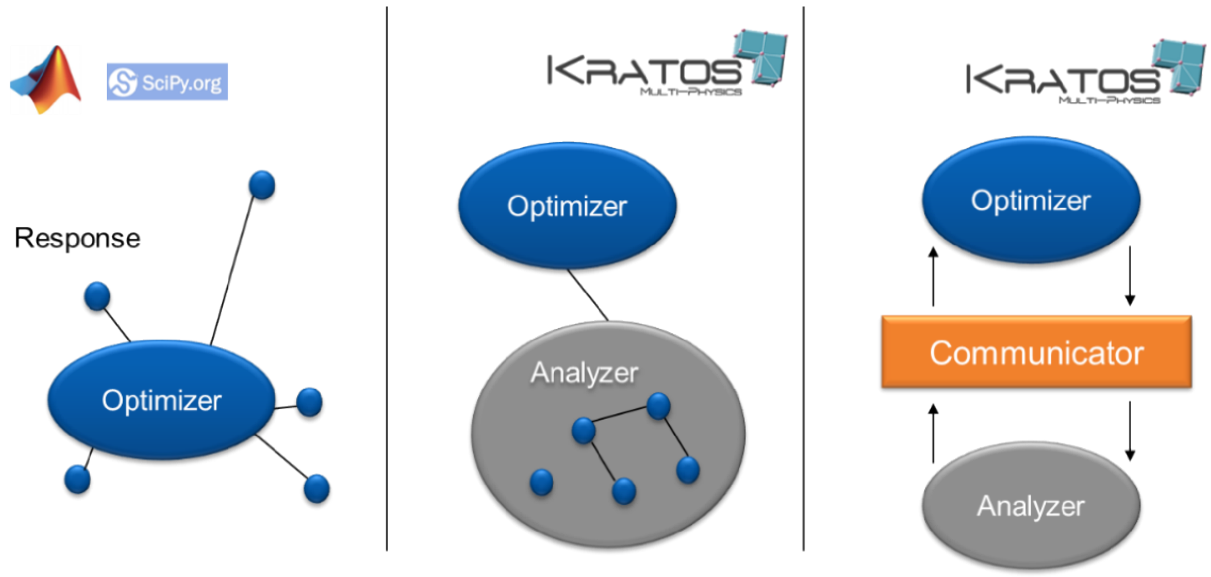

Figure 2: Implementational concept of the deterministic algorithms in KRATOS.

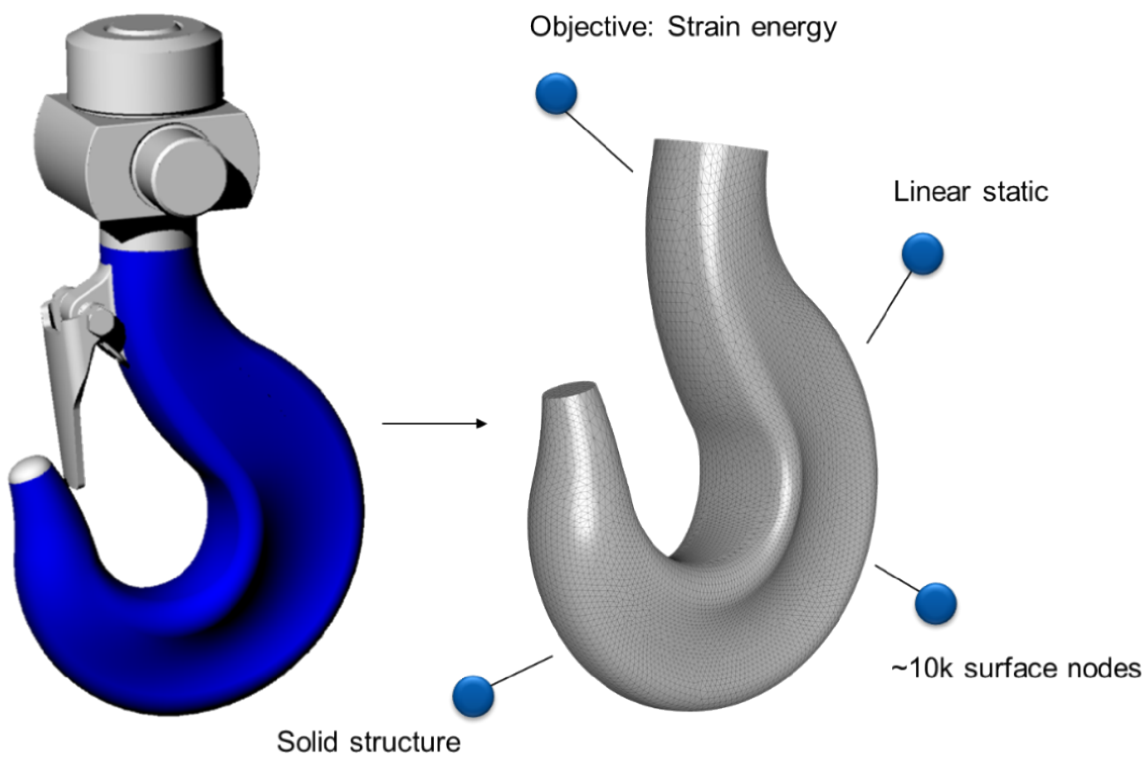

Figure 3: Problem placement of a hook for the shape optimization algorithm.

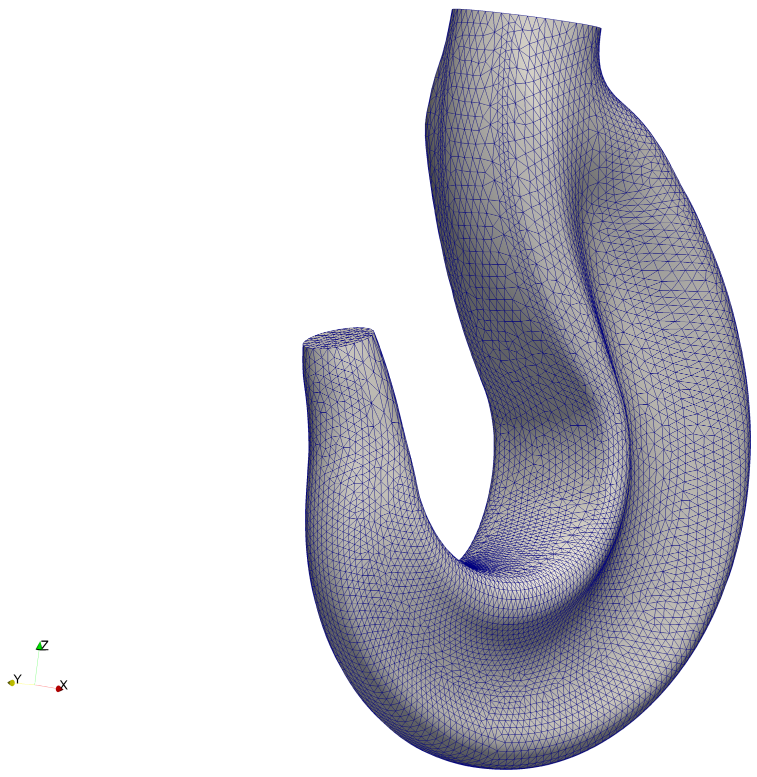

Figure 4: Shape optimized hook for mass under the constraint of keeping the strain-energy (stiffness) constant.

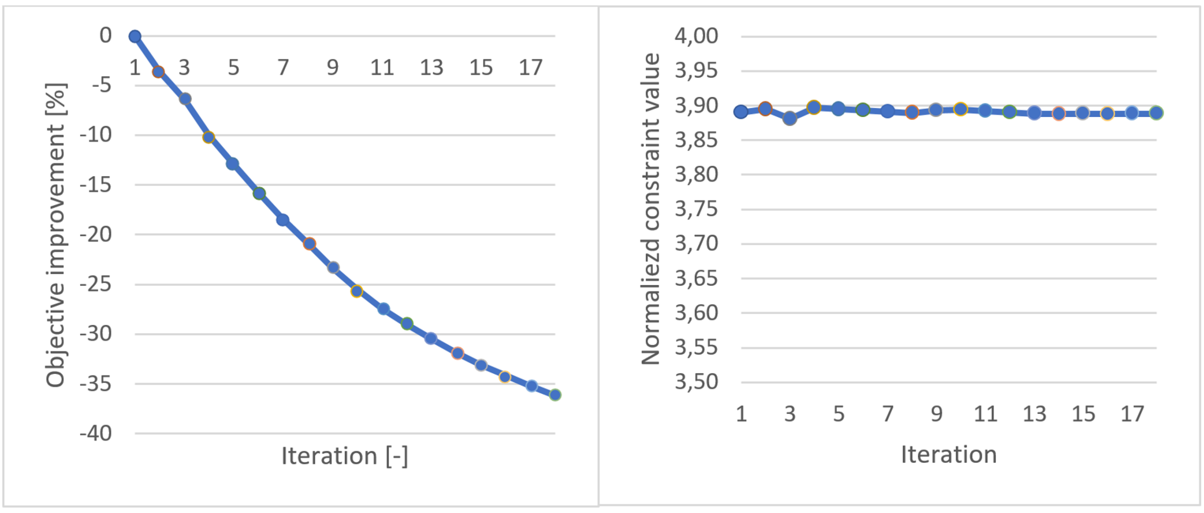

Figure 5: Objective and constraint history through the optimization iterations.

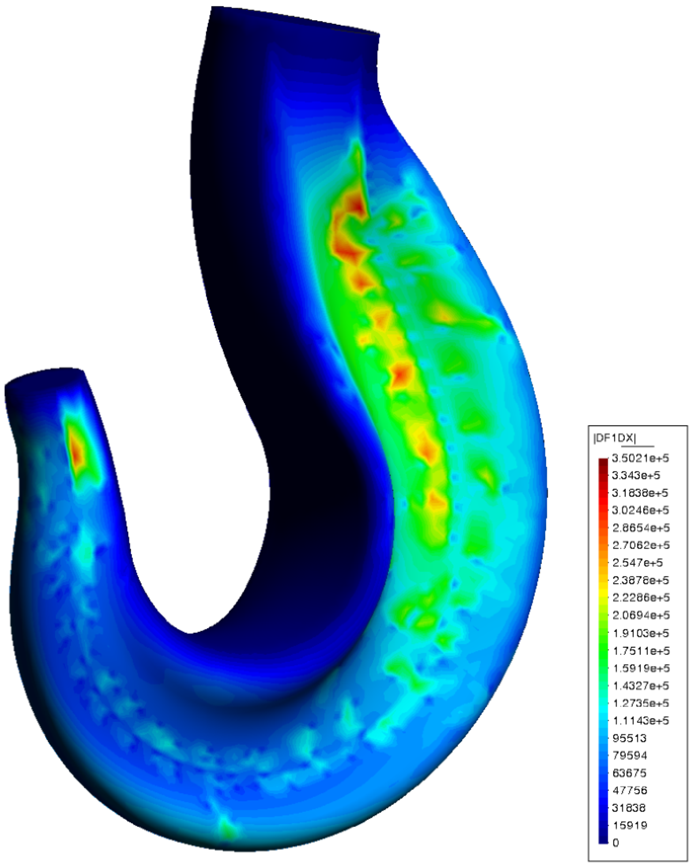

Figure 6: Sensitivity map at the tenth optimization iteration.

The sensitivity map corresponding to the objective function in the tenth optimization iteration is shown in Fig. 6. Even without performing optimization, such a plot can be used to identify hot spots that have big influence on the objective which are indicated by red in Fig. 6.

A API definition and usage

This appendix provides a brief documentation and explanation of the API for the deterministic optimization and how those can be used.

```
# Import Kratos core and apps
from KratosMultiphysics.ShapeOptimizationApplication import *

# Perform optimization
with open("parameters.json",'r') as parameter_file:
parameters = Parameters(parameter_file.read())

# Create the corresponding model
model = Model()

# Perform optimization
optimizer = optimizer_factory.CreateOptimizer(parameters["optimization_settings"],
    model)
optimizer.Optimize()

```

Figure 7: Example of the corresponding calls in the python level for the deterministic optimization algorithms using in KRATOS and an internal analyser. (taken from file ./src/run_test.py)

```
# Import Kratos core and apps
from KratosMultiphysics.ShapeOptimizationApplication import *

# Perform optimization
with open("parameters.json",'r') as parameter_file:
parameters = Parameters(parameter_file.read())

# Create the corresponding model
model = Model()

# Import the external analyser
import optimizer_factory

# Perform optimization
optimizer = optimizer_factory.CreateOptimizer(parameters["optimization_settings"],
    model, CustomAnalyzer())
optimizer.Optimize()

```

Figure 8: Example of the corresponding calls in the python level for the deterministic optimization algorithms using in KRATOS using an external analyser. (taken and slightly modified from file ./src/run_test.py)

As standard in KRATOS Multiphysics software ®, the simulation steps are controlled using a python layer. For the corresponding deterministic optimization algorithms, internal (implemented in KRATOS core) and/or external (third party) analysers can be used. This is shown in Figs. 7 and 8, respectively, where the external analyser is identified by `CustomAnalyzer()`. Concerning the internal analyser, the analysis stage within the corresponding class is shown in Fig. 9.

```python
class KratosInternalAnalyzer( (__import__("analyzer_base")).AnalyzerBaseClass ):
    # --------------------------------------------------------------------------
    def __init__( self, optimization_settings, model_part_controller ):
        self.model_part_controller = model_part_controller

        self.response_function_list = response_function_factory.CreateListOfResponseFunctions(optimization_settings, self.model_part_controller.GetModel())

    # --------------------------------------------------------------------------
    def InitializeBeforeOptimizationLoop( self ):
        for response in self.response_function_list.values():
            response.Initialize()
    # --------------------------------------------------------------------------
    def AnalyzeDesignAndReportToCommunicator( self, currentDesign, optimizationIteration, communicator ):

        for identifier, response in self.response_function_list.items():

            response.InitializeSolutionStep()

            # response values
            if communicator.isRequestingValueOf(identifier):
                response.CalculateValue()
                communicator.reportValue(identifier, response.GetValue())

            # response gradients
            if communicator.isRequestingGradientOf(identifier):
                response.CalculateGradient()
                communicator.reportGradient(identifier, response.GetShapeGradient())

            response.FinalizeSolutionStep()

            self.__ClearResultsFromModelPart()

    # --------------------------------------------------------------------------
    def FinalizeAfterOptimizationLoop( self ):
        for response in self.response_function_list.values():
            response.Finalize()

    # --------------------------------------------------------------------------
    def __ClearResultsFromModelPart( self ):
        self.model_part_controller.SetMeshToReferenceMesh()
        self.model_part_controller.SetDeformationVariablesToZero()

```

Figure 9: Class associated with the internal analyser in KRATOS. (taken from file ./src/analyzer_factory.py)

```python
class Analyzer:
    # --------------------------------------------------------------------------
    def __init__(self, optimization_settings, model_part_controller, external_analyzer):
        self.model_part_controller = model_part_controller
        self.external_analyzer = external_analyzer

        if self.__IsInternalAnalyzerRequired(optimization_settings):
            from analyzer_internal import KratosInternalAnalyzer
            self.internal_analyzer = KratosInternalAnalyzer(optimization_settings, model_part_controller)
        else:
            from analyzer_empty import EmptyAnalyzer
            self.internal_analyzer = EmptyAnalyzer()
            if isinstance(external_analyzer, EmptyAnalyzer):
                raise RuntimeError("Neither an internal nor an external analyzer is defined!")

    # --------------------------------------------------------------------------
    def InitializeBeforeOptimizationLoop(self):
        self.internal_analyzer.InitializeBeforeOptimizationLoop()
        self.external_analyzer.InitializeBeforeOptimizationLoop()

    # --------------------------------------------------------------------------
    def AnalyzeDesignAndReportToCommunicator(self, current_design, unique_iterator, communicator):
        self.internal_analyzer.AnalyzeDesignAndReportToCommunicator(current_design, unique_iterator, communicator)
        self.external_analyzer.AnalyzeDesignAndReportToCommunicator(current_design, unique_iterator, communicator)

        self.__ResetPossibleShapeModificationsFromAnalysis()

    # --------------------------------------------------------------------------
    def FinalizeAfterOptimizationLoop(self):
        self.internal_analyzer.FinalizeAfterOptimizationLoop()
        self.external_analyzer.FinalizeAfterOptimizationLoop()

    # --------------------------------------------------------------------------
    def __IsInternalAnalyzerRequired(self, optimization_settings):
        for objective_number in range(optimization_settings["objectives"].size()):
            if optimization_settings["objectives"][objective_number]["use_kratos"].GetBool():
                return True

        for constraint_number in range(optimization_settings["constraints"].size()):
            if optimization_settings["constraints"][constraint_number]["use_kratos"].GetBool():
                return True
        return False

    # --------------------------------------------------------------------------
    def __ResetPossibleShapeModificationsFromAnalysis( self ):
        self.model_part_controller.SetMeshToReferenceMesh()
        self.model_part_controller.SetDeformationVariablesToZero()

```

Figure 10: Class associated with the external analyser in KRATOS. (taken from file ./src/analyzer_factory.py)

If one wishes to use a third party analyser, herein called external analyser, the class `KratosInternalAnalyzer` is employed, see Fig. 7.

The aforementioned class can then be then instantiated in the python layer as shown in Fig. 11, where functions such as `__ObjectiveFunction`, `__ObjectiveGradient`, etc. can be specialized.

```python
from analyzer_base import AnalyzerBaseClass
class CustomAnalyzer(AnalyzerBaseClass):

    # --------------------------------------------------------------------------
    def AnalyzeDesignAndReportToCommunicator(self, current_design, optimization_iteration, communicator):
        if communicator.isRequestingValueOf("targetDeviation"):
            communicator.reportValue("targetDeviation", self.__ObjectiveFunction(current_design))

        if communicator.isRequestingGradientOf("targetDeviation"):
            communicator.reportGradient("targetDeviation", self.__ObjectiveGradient(current_design))

    # --------------------------------------------------------------------------
    def __ObjectiveFunction(self, current_design):
        """ Returns the objective function to be minimized """
        objective = 0.0
        for node in current_design.Nodes:
            objective = objective + abs(self.__TentFunction(node.X) - node.Z)
        return objective

    # --------------------------------------------------------------------------
    def __ObjectiveGradient(self, current_design):
        """ Returns the gradient of the objective function """
        sensitivity = dict()
        for node in current_design.Nodes:
            delta = node.Z - self.__TentFunction(node.X)
            if abs(delta) == 0.0:
                sz = 0.0
            else:
                sz = delta / abs(delta)
            sensitivity[node.Id] = [0.0, 0.0, sz]
        return sensitivity

    # --------------------------------------------------------------------------
    @staticmethod
    def __TentFunction(x):
        """ Defines the target curve z=__TentFunction(x) """
        if x <= 15.0:
            return 0.0
        elif x<= 20.0:
            return (x - 15.0) / 5.0
        elif x <= 25.0:
            return 1.0 - (x - 20.0) / 5.0
        else:
            return 0.0

```

Figure 11: Specialization of the class associated with the external analyser in KRATOS. (taken from file ./src/run_test.py)

Moreover, if one wishes to use an external analyser, entry `["optimization_settings"]` in the `parameters.json` file need to be specified, see Fig. 12. Note that in case of employing an external analyser, the flag `"use_kratos"` needs to be set to `false`.

References

[1] D. Baumgärtner, A. Viti, A. Dumont, G. Carrier, and K.-U. Bletzinger. Comparison and combination of experience-based parameterization with vertex morphing in aerodynamic shape optimization of a forward-swept wing aircraft. In *17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, American Institute of Aeronautics and Astronautics*, 2016.

```
"optimization_settings" : {
   "model_settings"  : {
    "domain_size"              : 3,
    "model_part_name"          : "tent",
    "model_import_settings"    : {
    "input_type"       : "mdpa",
    "input_filename"   : "tent"
    },
   "design_surface_sub_model_part_name" : "design_surface",
   "damping" : {
         "apply_damping" : false
     },
         "mesh_motion" : {
         "apply_mesh_solver" : false
     }
    },
    "objectives" : [{
         "identifier" : "targetDeviation",
         "type"       : "minimization",
         "use_kratos" : false,
         "project_gradient_on_surface_normals" : true
        }],
        "constraints" : [],
      "design_variables" : {
      "type"   : "vertex_morphing",
      "filter" : {
         "filter_function_type"       : "gaussian",
         "filter_radius"              : 5.0,
         "max_nodes_in_filter_radius" : 100
      }
   },

```

Figure 12: Specification of the optimization properties in `parameters.json` file when using an external analyser. (taken from file ./src/parameters.json)

[2] K.-U. Bletzinger. A consistent frame for sensitivity filtering and the vertex assigned morphing of optimal shape. *Structural and Multidisciplinary Optimization*, 49(6): 873–895, 2014.

[3] K.-U. Bletzinger. Shape optimization. *Encyclopedia of Computational Mechanics Second Edition, R. B. and T. J. H. E. Stein*, pages 1–42, 2017.

[4] M. Firl. *Optimal shape design of shell structures.* PhD thesis, Technische Universität München, 2010.

[5] R. T. Haftka and Z. Gürdal. *Elements of structural optimization*, volume 11. Springer Science & Business Media, 2012.

[6] M. Hojjat, E. Stavropoulou, and K.-U. Bletzinger. The vertex morphing method for node-based shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 268:494–513, 2014.

[7] V. Komkov, K. K. Choi, and E. J. Haug. *Design sensitivity analysis of structural systems*, volume 177. Academic press, 1986.

[8] H. Masching. *Parameter Free Optimization of Shape Adaptive Shell Structures.* PhD thesis, Technische Universität München, 2016.

[9] T. Wick. Fluid-structure interactions using different mesh motion techniques. *Computers & Structures*, 89(13-14):1456–1467, 2011.