How2: DesOptPy

E. J. Wehrle*

Version: October 17, 2015

The following is a tutorial for getting started with DesOptPy, an optimization toolbox written with Python. The installation and the use of DesOptPy are introduced. The syntax is explained. Examples have been further provided to ease the learning process.

Nomenclature

$\mathbf{f}$	Objective function vector
f	Objective function
f^*	Optimal objective function
f^0	Inital objective function
$\mathbf{g}$	Inequality constraint function vector
$\mathbf{g}^*$	Optimal inequality constraint function vector
$\mathbf{g}^0$	Inital inequality constraint function vector
g	Inequality constraint function
$\mathbf{L}$	Lagrangian funciton value
L	Lagrangian funciton value
$\mathsf{S_P}$	Shadow prices
$\mathbf{X}$	Design domain matrix
$\mathbf{x}$	Design variable vector
$\mathbf{x}^*$	Optimal design variable vector
$\mathbf{x}^0$	Initial design variable vector
$\mathbf{x}^L$	Lower bounded design variable vector
$\mathbf{x}^U$	Upper bounded design variable vector
x	Design variable
x^L	Lower bounded design variable
x^U	Upper bounded design variable
$\boldsymbol{\lambda}$	Lagrangian multiplier
λ	Lagrangian multiplier
∇	Nabla operator, here: gradient with respect to design variables

*Dr.-Ing. Erich Wehrle
Postdoctoral Researcher
Professur Computational Mechanics
Ingenieurfakultät Bau Geo Umwelt
Technische Universität München
Arcisstr. 21
80333 München
Germany
Office: Augustenstr. 44
+49-89-289-28663
wehrle@tum.de

1. Introduction to DesOptPy

The package DesOptPy (Design Optimization in Python) was written by the author for use in structural design optimization for mechanical structures but can be used for a great variety of optimization problems. The goal of this project was to design a general optimization toolbox for structural design optimization in which an optimization model can be set up easily, quickly, efficiently and effectively, allowing colleagues and students dive into the optimization problem without difficulty. It is also meant to be modular and easily expandable. If you should find errors in the code or documentation, have suggestions for improvements or wish a cooperation, please contact the author.

DesOptPy was developed by the author in his research and published in Wehrle (2015). The code has been used in the following: Wehrle et al. (2014b,a) in addition to several theses at the Institute of Lightweight Structures of the Technical University of Munich including Rudolph (2013); Wachter (2014); Richter (2014); Braun (2014).

This toolbox is designed to solve optimization problems of the following convention:

$$\min_{\mathbf{x}\in\mathbf{X}} \left\{ \mathsf{f}\left(\mathbf{x}\right) | \mathbf{g}\left(\mathbf{x}\right) \leq \mathbf{0} \right\},$$

where f is the objective function, $\mathbf{g}$ the inequality constraint function, $\mathbf{x}$ the optimization variables and $\mathbf{X}$ the optimization domain defined by the lower and upper bounds of the variables. The upper-bounded inequality constraint function is defined as

$$\mathrm{g}_i = r_i - c_i$$

or normalized as

$$\mathrm{g}_i = \frac{r_i}{c_i} - 1,$$

and analogously for lower-bounded inequality constraints as

$$\mathrm{g}_i = c_i - r_i$$

or normalized as

$$\mathrm{g}_i = 1 - \frac{r_i}{c_i},$$

where r_i is the constrained response and c_i is the state limit.

The optimization problem above can be solved using direct or approximation approaches utilizing a variety of optimization algorithms of zeroth, first or second order within the optimization model (fig. 1.1). The direct approaches uses the system responses in the optimization process, while approximation approaches utilizes approximated system responses. The approximation is a two-step process of sampling (design of experiments) and approximating (surrogate modelling). In DesOptPy the sampling is carried out via Latin hypercube sampling with pyDOE with a custom option of including the corners of the design space. The approximation is via Gaussian process (Kriging) with scikit-learn, which is in turn is the Python implementation of DACE (Lophaven et al., 2002).

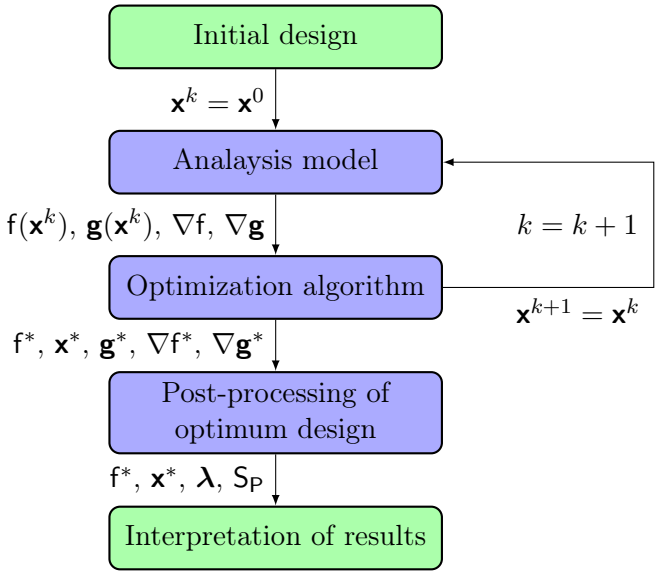


Figure 1.1: Flow chart of an optimization within the optimization model

When using gradient-based algorithms, the Lagrangian multiplier is used for post-processing of the optimization process: Optimality and shadow prices. Both of these are based on the Lagrangian function

$$\mathsf{L}\left(\mathbf{x}^*, \boldsymbol{\lambda}\right) = \mathsf{f}\left(\mathbf{x}\right) + \boldsymbol{\lambda}\mathbf{g}\left(\mathbf{x}\right),$$

where here the constraint function vector $\mathbf{g}$ is expanded to contain the bound constraints

$$\mathsf{g}_i^L = \mathsf{x}_i^L - \mathsf{x}_i$$

and

$$\mathsf{g}^U = \mathsf{x}_i - \mathsf{x}_i^U,$$

as well as inequality constraints.

Optimality is defined using the criteria after Karush (1939); Kuhn and Tucker (1951), which is generally necessary yet not sufficient, i.e. for the subset of convex problems it is the optimum and for more general non-convex problems that it is *a* optimum yet not necessarily *the* optimum. This criteria says that (local) optimality is present when

$$\begin{aligned}
\text{Stationary:} \quad & \nabla\mathsf{L}\left(\mathbf{x}^*, \boldsymbol{\lambda}\right) && = \mathbf{0} \\
\text{Primal feasibility :} \quad & \mathbf{g}\left(\mathbf{x}^*\right) && \leq \mathbf{0} \\
\text{Dual feasibility:} \quad & \boldsymbol{\lambda} && \geq \mathbf{0} \\
\text{Complementary slackness :} \quad & \mathsf{g}_i\lambda_i && = 0.
\end{aligned}$$

Shadow price $\mathsf{S_P}$ is a meaning of the Lagrangian multipliers λ_i at the optimum. Rearranging the Lagrangian function (see above), we get

$$\lambda_i = -\frac{\partial \mathsf{f}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathsf{g}_j}$$

and therefore

$$\lambda_i = -\frac{\partial \mathsf{f}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathsf{g}_j} \frac{\partial \mathsf{g}_j}{\partial c_j}.$$

The shadow price is defined depending on the specific definition of the constraint function (see above),

$$\begin{aligned}
\text{Upper bound, non-normalized:} \quad & \lambda_j = -\frac{\partial \mathsf{f}}{\partial c_j} \\
\text{Lower bound, non-normalized:} \quad & \lambda_j = \frac{\partial \mathsf{f}}{\partial c_j} \\
\text{Upper bound, normalized:} \quad & \lambda_j = -\frac{\partial \mathsf{f}}{\partial c_j} \frac{1}{c_j} \\
\text{Lower bound, normalized:} \quad & \lambda_j = \frac{\partial \mathsf{f}}{\partial c_j} \frac{1}{c_j}.
\end{aligned}$$

Currently, the solvers of pyOpt (Perez and Jansen, 2013) and PyGMO are used. This is, though, expandable to other optimization algorithms and packages.

2. Structure of DesOptPy

The code for DesOptPy utilizes a folder construct with `DesOpt/`, which includes the main folder repository of models `Models/`. In normal running, the files are copied to `Run/` for optimization and then to `Results/` upon completion. This can be turned off by using debug mode in which the models are then run in the model folder. The basis folder saved where user wishes, though it is recommended to use `/home/$User/` or the windows equivalent `c:\Users\$User\`, where `$User` is the user name of the computer.

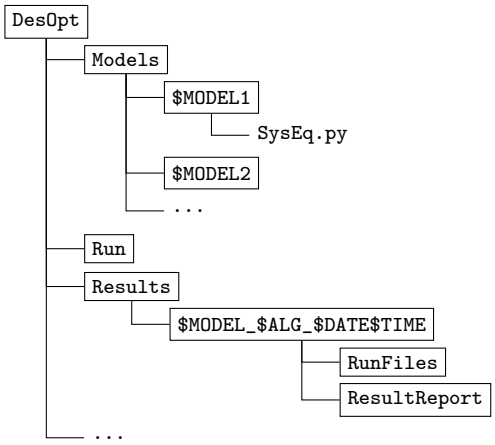


Figure 2.1: Folder and file structure of DesOptPy

2.1. Prerequisites

The following are needed for full usage of DesOptPy:

Table 1: Software prerequisites for DesOptPy

Software	Source
Python	www.python.org
pyOpt	www.pyopt.org
pyGMO	esa.github.io/pygmo
SciPy	www.scipy.org
NumPy	www.numpy.org
MatPlotLib	www.matplotlib.org
scikit-learn	www.scikit-learn.org
pyDOE	www.github.com/tisimst/pyDOE
LyX	www.lyx.org/
Inkscape	www.inkscape.org/

DesOptPy relies on the optimization solvers provided in pyOpt and PyGMO. For the installation of the software, see the relevant user's guide.

For the proper generation of the result report, LyX and Inkscape must be installed and must be able to be called via command file with

frame

```
1 lyx
```

and

frame

```
1 inkscape
```

respectively (symbolic link in Linux—typically standard—or via setting a environment variable in Windows).

Further, a integrated development environment is recommended. DesOptPy has been used and developed with the following:

Spyder pythonhosted.org/spyder/

PyCharm jetbrains.com/pycharm/

2.2. Syntax

The options of DesOptPy are introduced in the following where the default values are in parentheses:

2.2.1. Input parameters

SysEq The name of the function of the system equation: `{definition}`

x0 The initial design vector: `{array}`

xU The upper bounds of the design vector: `{array}`

xL The upper bounds of the design vector: `{array}`

gc The numerical value of the inequality constraints, needed for proper denormalization of the shadow prices `{array}`

hc The numerical value of the equality constraints, needed for proper denormalization of the shadow prices (not currently supported): `{(‘‘[]’’), array}`

Alg The algorithm for the optimization: `{("SLSQP"), "MMA", "GCMMA", "NLPQLP", "CONMIN", "NSGA2", "COBYLA", "KSOPT", "ALGENCAN", "SDPEN", "SOLVOPT", "PyGMO_de", "PyGMO_bee_colony", "PyGMO_nsga_II", "PyGMO_pso", "PyGMO_pso_gen", "PyGMO_cmaes", "PyGMO_py_cmaes", "PyGMO_spea2", "PyGMO_nspso", "PyGMO_pade", "PyGMO_sea", "PyGMO_vega", "PyGMO_sga", "PyGMO_sga_gray", "PyGMO_de_1220", "PyGMO_mde_pbx", "PyGMO_jde", "PyGMO_ihs", "PyGMO_monte_carlo", "PyGMO_sa_corana", "PyGMO_sms_emoa"}`

AlgOptions Options for each algorithm. Set default and get options for each algorithm with `"OptAlgOptions.setDefault($AlgorithmName)"`. This field is also optional or can be defined with the simplified setting object function `"setSimple(self, stopTol=[], maxIter=[], maxEval=[])"`. [1]

SensCalc The type of design sensitivity analysis: `{(‘‘FD’’), ‘‘ParaFD’’, ‘‘OptSensEq’’ }`

DesVarNorm The type of normalization used for the design variables: `{(True), ‘‘xLxU’’, ‘‘x0xU’’ ,‘‘None’’, None, False}`[2,1]

deltax The step size used for design sensitivity analysis with finite differences: `{(‘‘1e-3’’), float}`

OptStatus Provides a status report during the optimization:`{(False), True}`[1]

OptPostProcess Provides postprocessing after optimization, including result report: `{(False), True}`[1]

OptVideo Provides a video of design evolution of optimization process: `{(False), True}`

DoE Use design of experiment with number of samples to be later used for surrogate-based design optimization: `{(False), integer}`

SBDO Use surrogate-based design optimization (Currently only Kriging implemented): `{(False), True}`

Debug Carry out optimization within model folder instead of using the DesOptRun for processing and DesOptResults for solution: `{(False), True}`

PrintOut Print out final details of optimization upon completion: `{(True), False}`

[1] Not yet available for PyGMO algorithms

[2] `True = ‘‘xLxU’’; None = False = ‘‘None’’`

2.2.2. Output parameters

xOpt The optimal design vector values: `{array}`

fOpt The optimal objective value: `{array}`

SP Shadow prices of active constraints at optimum: `{array}`

2.2.3. Pseudo-code syntax example

In list. 1 an example of the syntax is given by pseudo code. This example can be easily used as a layout for programming future optimization problems.

Listing 1: Syntax of optimization problem for DesOptPy

```
from DesOptPy import DesOpt

def SysEq(x, gc)
    # here: system equations
    f = ...
    g = ...
    return(f, g)

def SensEq(x, gc)  # optional
    # here: sensitivity equations
    dfdx = ...
    dgdx = ...
    return(dfdx, dgdx)

x0 = ...
xL = ...
xU = ...
gc = ...
Alg = "NLPQLP"
AlgOptions = OptAlgOptions.setDefault(Alg)
AlgOptions.setSimple(stopTol=1e-3)
xOpt, fOpt, SP = DesOpt(SysEq=SysEq, x0=x0, xU=xU, xL=xL, gc=gc, Alg=Alg,
                        SensCalc="FD", DesVarNorm="xLxU", deltax=1e-3,
                        OptStatus=False, OptPostProcess=False,
                        OptVideo=False, DoE=False, SBDO=False,
                        Debug=False, PrintOut=True, AlgOptions=AlgOptions)
```

2.3. Normalization of the design variables `DesVarNorm`

There are four schemes for the normalization of the design variables as well as the option to use no normalization `None`.

xLxU All design variables take on a value between zero (lower bounds) and unity (upper bounds).

xLx0 All design variables take on a value, where zero is the lower bound and unity is the start value.

x0 All design variables take a value where unit is the start value.

xU All design variables take a value where unit is the upper bounds.

Currently, only `xLxU` and `None` function for use when providing the design sensivities, though all will work when using finite differencing or non-gradient solvers.

2.4. Status reports

As optimization runs can take hours, days or even longer, it is of great interest to monitor the progress. A HTML file serves this purpose to show the current state of the optimization. This option is activated by `OptStatus=True`.

2.5. Result reports

Using LyX and LaTeX, result reports are created . For this to work properly these programs must be included in the environmental variables of the operating system. This option is activated by `OptPostProcess=True`. By turning on this option, diverse convergence plots are created with MatPlotLib.

2.6. Interfaces

DesOptPy is designed to be modular and expandable for different optimization problems, also those beyond structural design optimization. Presently there are interfaces to the following software:

- ANSYS Classic
- LS-DYNA.

These interfaces to other software applications are found in the correspondingly named folder `DesOpt/.Interfaces/`, which can easily expanded and customized.

3. Examples

Here three example problems will be shown to provide the user an idea how DesOptPy can be used: an unconstrained univariate test problem and a unconstrained multivariate test problem and a constrained multivariate problem.

Example 1: Unconstrained univariate analytical test problem: Parabola

The first test problem shows the academic example of minimizing a unidimensional function of a parabola of the form

$$f = (x + 10)^2 .$$

The Python code for this can be found in list. 2.

Listing 2: Unconstrained univariate analytical test problem

```
from DesOptPy import DesOpt
from DesOptPy import OptAlgOptions
import numpy as np


def SysEq(x, gc):
    f = (x[0]+10)**2
    g = []
    return(f, g)


def SensEq(x, f, g, gc):
    dfdx = np.array([2.*x + 20.])
    dgdx = []
    return(dfdx, dgdx)


x0 = np.array([0.])
xL = np.array([-100.])
xU = np.array([100.])
gc = []
Alg = "SLSQP"
AlgOptions = OptAlgOptions.setDefault(Alg)
AlgOptions.setSimple(stopTol=1e-3)
xOpt, fOpt, SP = DesOpt(x0=x0, xL=xL, xU=xU, gc=gc, SysEq=SysEq,
                        SensEq=SensEq, SensCalc="OptSensEq",
                        Alg=Alg, StatusReport=True, Debug=False,
                        DoE=False, SBDO=False, ResultReport=True,
                        deltax=1e-6, DesVarNorm=True, AlgOptions=AlgOptions)
```

The optimum of this trivial optimization problem is

$$\begin{aligned} x &= -10 \\ f^* &= 0. \end{aligned}$$

Example 2: Unconstrained multivariate analytical test problem: Rosenbrock's banana function

Rosenbrock's banana function is one of the most famous test problems of optimization. Though the problem is non-convex, it can be solved with gradient-based algorithms. The objective function has the form

$$f = 100 * \left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 .$$

The Python code for this can be found in list. 3.

Listing 3: Unconstrained multivariate analytical test problem: Rosenbrock's banana function

```
from DesOptPy import DesOpt
import numpy as np
from scipy.optimize import rosen
from scipy.optimize import rosen_der

def SysEq(x, gc):
    f = rosen(x)
    g = []
    return(f, g)


def SensEq(x, f, g, gc):
    dfdx = np.array(rosen_der(x)).reshape([1,len(x)])
    return(dfdx, [])


x0 = np.ones([2,])*0.
xL = np.ones([2,])*-5
xU = np.ones([2,])*5
gc = []
xOpt, fOpt, SP = DesOpt(x0=x0, xL=xL, xU=xU, gc=gc, Alg="PyGMO_de",
                        SysEq=SysEq, deltax=1e-6,
                        StatusReport=True, DoE=False, SBDO=False,
                        ResultReport=True, DesVarNorm=True)
```

The optimum of this optimization problem is

$$\begin{aligned} \mathbf{x}^* &= \begin{bmatrix} 1 & 1 \end{bmatrix} \\ \mathsf{f}^* &= 0. \end{aligned}$$

Example 3: Constrained multivariate analytical test problem: Cantilever[3]

The optimization problem takes on the form

$$\begin{aligned} \mathsf{f} &= a \\ \mathsf{g}_1 &= \frac{\sigma}{\sigma_{\max}} - 1 \\ \mathsf{g}_2 &= \frac{u}{u_{\max}} - 1, \end{aligned}$$

where the relevant system equations are defined as

[3]Source: Dakota user's manual.

$$
\begin{aligned}
a &= w \cdot t \\
\sigma &= \frac{600}{w^2 \cdot t} \cdot f_x + \frac{600}{w \cdot t^2} \cdot f_y \\
u &= \frac{4 \cdot \ell^3}{E \cdot w \cdot t} \sqrt{\left(\frac{f_x}{w^2}\right)^2 + \left(\frac{f_y}{t^2}\right)^2}.
\end{aligned}
$$

The Python code for this can be found in list. 4.

Listing 4: Constrained multivariate analytical test problem: Cantilever

```
from DesOptPy import DesOpt
from DesOptPy import OptAlgOptions
import numpy as np


def SysEq(x, gc):
    e = 2.9e7
    r = 40000.
    fx = 500.
    fy = 1000.
    w = x[0]
    t = x[1]
    area = w*t
    f = area
    D0 = 2.2535
    L = 100.
    w_sq = w*w
    t_sq = t*t
    r_sq = r*r
    x_sq = fx*fx
    y_sq = fy*fy
    stress = 600.*fy/w/t_sq + 600.*fx/w_sq/t
    D1 = 4.*pow(L,3)/e/area
    D2 = pow(fy/t_sq, 2)+pow(fx/w_sq, 2)
    D3 = D1/np.sqrt(D2)/D0
    D4 = D1*np.sqrt(D2)/D0
    g1 = stress/r - 1.
    g2 = D4 - 1.
    return(f, [g1, g2])


xL = np.array([1.0, 1.0])
xU = np.array([4.0, 4.0])
gc = np.array([0.0, 0.0])
x0 = np.array([1.8, 1.0])
Alg = "PyGMO_sa_corana"
AlgOptions = OptAlgOptions.setDefault(Alg)
AlgOptions.iter = 500
AlgOptions.Ts = 10
AlgOptions.Tf = 0.3
AlgOptions.steps = 1
AlgOptions.bin_size = 1
AlgOptions.nIndiv = 1
xOpt, fOpt, SP = DesOpt(x0=x0, xL=xL, xU=xU, gc=gc, SysEq=SysEq, Alg=Alg, StatusReport=False,
                        DesVarNorm=True, DoE=False, SBDO=False, ResultReport=False, deltax=1e-2,
                        Debug=False, OptNameAdd="Cantilever", AlgOptions=AlgOptions)
```

The optimum of this optimization problem is

$$\begin{aligned} \mathsf{x}^* &= \begin{bmatrix} 2.35 & 3.33 \end{bmatrix} \\ \mathsf{f}^* &= 7.82. \end{aligned}$$

4. Possbile future developments

In the following is a list of possible further developments for DesOptPy.

- Optimization algorithms
  - openopt
  - Dakota
  - Hunkeler's hybrid cellular automaton for thin-walled structures (HCA-TWS)
  - User-written algorithm option
- Surrogate-based design optimization
  - Pass options to surrogate modeling
  - Polynomial regression
  - Iterative surrogate modeling (adaptive)
- Constraint Aggregation
  - Maximum constraint
  - Kresselmeier–Steinhauser (also adaptive?)
- Equality constraints?
- Object-oriented programming(?)

5. Change log

Version α1.0

- Changed configuration so that SysEq.py imports and calls DesOpt
- LLB → FGCM

Version α1.1

- Seconds in OptName
- gc in SysEq and SensEq

Version α1.2

- PyGMO solver
- New setup of optimization problem to ease integration of new algorithms
- Result presentation in alpha version—not finished
- Removed algorithm support for pyCMA as two interfaces to CMA-ES in PyGMO

Version 1.0

- Public release

References

Braun, S. (2014). Modellbildung und Entwurfsoptimierung von einer Gitterrohrrahmenstruktur für Elektrofahrzeuge. Semester thesis, Lehrstuhl für Leichtbau, Technische Universität München. Advisor: E. J. Wehrle.

Karush, W. (1939). Minima of functions of several variables with inequalities as side constraints. Master's thesis, Department of Mathematics, University of Chicago.

Kuhn, H. W. and A. W. Tucker (1951). Nonlinear Programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492.

Lophaven, S. N., H. B. Nielsen, and J. Sondergaard (2002). *DACE: A MATLAB Kriging toolbox*. Informatik og Matematisk Modellering, Danmarks Tekniske Universitet.

Perez, R. E. and P. W. Jansen (2013). *pyOpt reference* (Release 1.1.0 ed.).

Richter, M. (2014). Parallelization of numerical sensitivity analysis for large-scale structural design optimization. Semester thesis, Lehrstuhl für Leichtbau, Technische Universität München. Advisor: E. J. Wehrle.

Rudolph, S. (2013). Implementation of a Python-based structural optimization framework with pyOpt for engineering design. Semester thesis, Lehrstuhl für Leichtbau, Technische Universität München. Advisor: E. J. Wehrle.

Wachter, F. (2014). Strukturoptimierung unter Aufpralllaster mittels Makroelemente. Bachelor's thesis, Lehrstuhl für Leichtbau, Technische Universität München. Advisor: E. J. Wehrle.

Wehrle, E. J. (2015). *Design optimization of lightweight space frame structures considering crashworthiness and parameter uncertainty*. Dr.-Ing. diss., Lehrstuhl für Leichtbau, Technische Universität München.

Wehrle, E. J., Q. Xu, and H. Baier (2014a). Investigation, optimal design and uncertainty analysis of crash-absorbing extruded aluminium structures. *Procedia CIRP 18*, 27–32.

Wehrle, E. J., Q. Xu, and H. Baier (2014b). Investigation, optimal design and uncertainty analysis of crashabsorbing extruded aluminum structures. In *Conference on Manufacture of Lightweight Components (ManuLight2014)*.

A. Reports

In this section, the status and result reports automatically generated with DesOptPy will be shown on the example of the cantilever optimization.

A.1. Status report

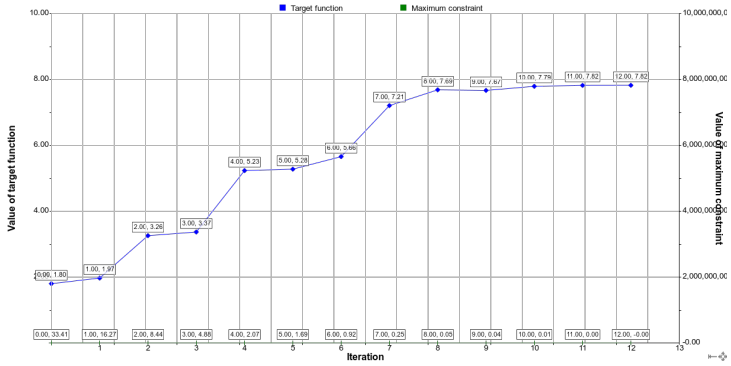
The status report is generated during the optimization process to provide live access to the developments. This is completed via a HTML document. The report is customizable with institution logos via the HTML file and `OptHis2HTML.py`. As example is as follows:

TECHNISCHE UNIVERSITÄT MÜNCHEN

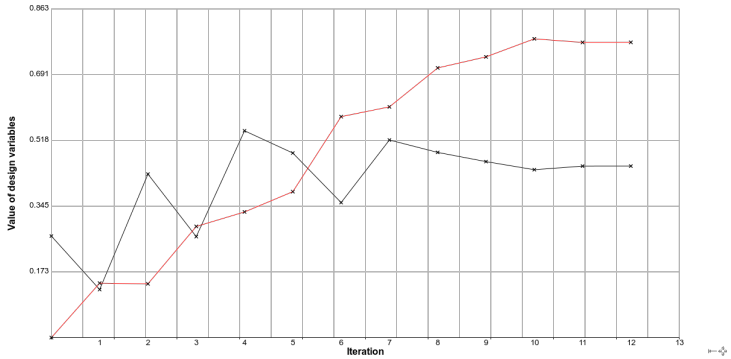
Design Optimization Status Report of Cantilever_NLPQLP_201412101116

Last Update: 2014-Dec-10 11:16:49

Convergence of objective function and constraint

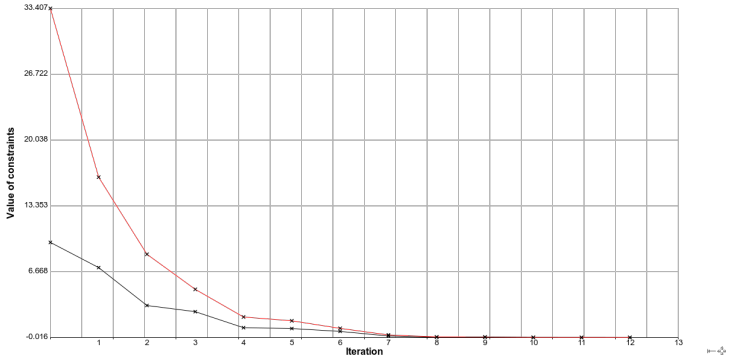


Convergence of design variables



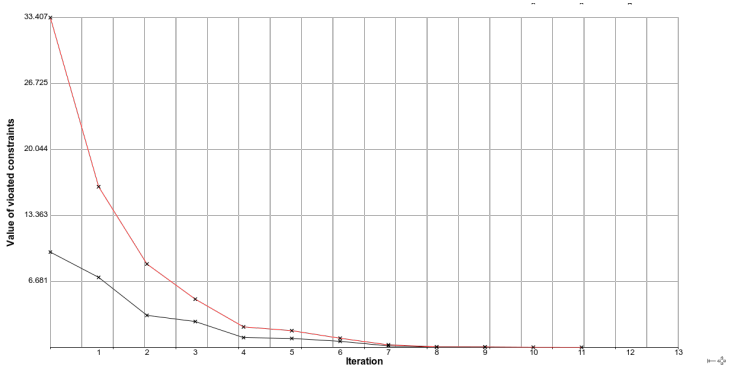
Number of selected design variable: 0

Convergence of constraints



Number of selected constraint: 1

Convergence of violated constraints



Number of selected violated constraint: 0

A.2. Result report

The result report is generated after the optimization process to an automatic summary of the optimization. This is completed via a MatPlotLib, LyX and LaTeX document. The report is able to be customized with institution logos via the LyX file `_ResultReportPy.lyx` and the Python file `OptResultReport.py`. As example is as follows:

Technische Universität München
Ingenieurfakultät Bau Geo Umwelt
Fachgebiet Computational Mechanics
Univ. Prof. Dr.-Ing. habil. F. Duddeck

Design Optimization Result Report

Cantilever

E. J. Wehrle*

December 10, 2014

*Research Associate
wehrle@tum.de
Fachgebiet Computational Mechanics
Technische Universität München
Arcisstraße 21
80333 München
Germany
Tel: +49-89-289-28663

Nomenclature

$\hat{f}$	Normalized objective function
$\hat{x}$	Normalized design variable
f	Objective function
f^*	Optimal value of objective function
f^0	Inital value of objective function
g	Constraint function
g^*	Optimal value of constraint function
g^0	Initial value of constraint function
x	Design variable
x^*	Optimal value of design variable
x^0	Inital value of design variable
∇	Nabla operator, here: gradient

1 Optimization problem

Property	Value
Computer name	Schreibtafelrechner
Operating system	Linux
Computer architecture	x86_64
Number of processors	2
User	wehrle

Table 1.1: Properties of computer of optimization run

Property	Value
Optimization problem	Cantilever
Number of design variables	2
Number of constraints	2
Algorithm name	NLPQLP

Table 1.2: Information about the optimization problem and the algorithm

Property	Value
ACC	1e-06
IPRINT	2
IOUT	6
LQL	True
MAXIT	50
MODE	0
RHOB	0.0
STPMIN	1e-06
ACCQP	1e-06
MAXFUN	10

Table 1.3: Algorithm options

2 Results

2.1 Plots

Objective function

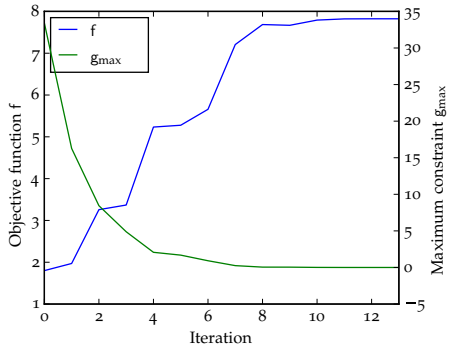


Figure 2.1.1: Convergence of objective function f and maximum constraint g_{max}

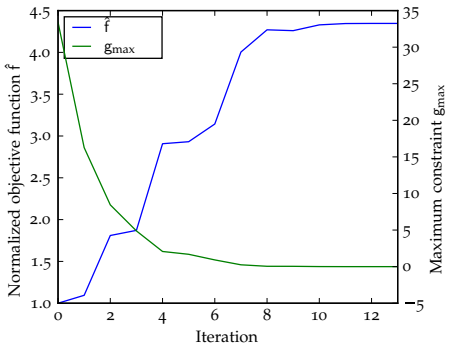


Figure 2.1.2: Convergence of normalized objective function $\hat{f}$ and maximum constraint g_{max}

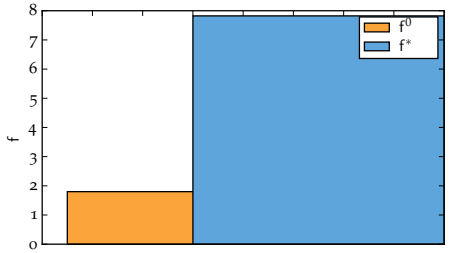


Figure 2.1.3: Objective function at start f^0 and optimum f^*

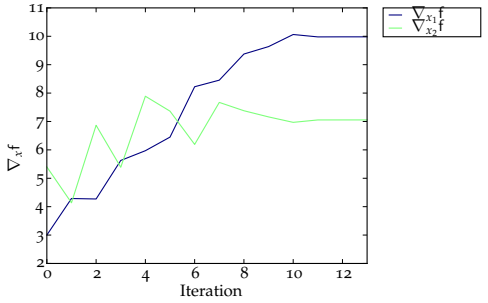


Figure 2.1.4: Convergence of gradient of objective function $\nabla_x f$

Design Variables

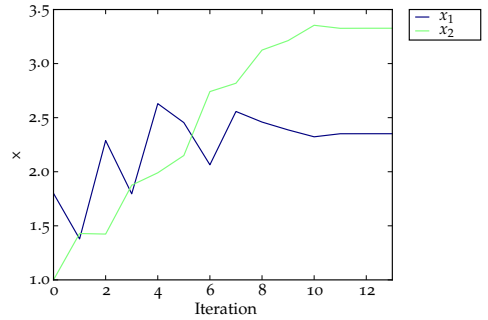


Figure 2.1.5: Convergence of design variables x

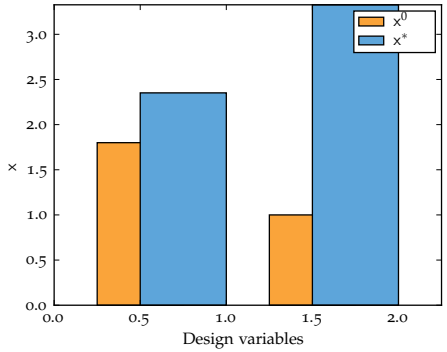


Figure 2.1.6: Design variables at start x^0 and optimum x^*

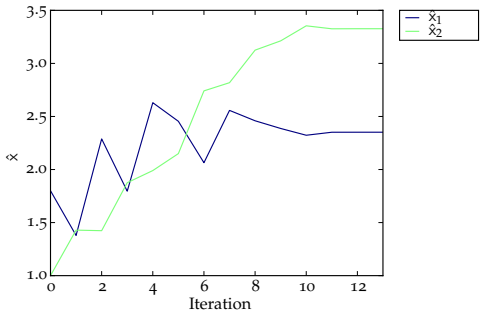


Figure 2.1.7: Convergence of normalized design variables $\hat{x}$

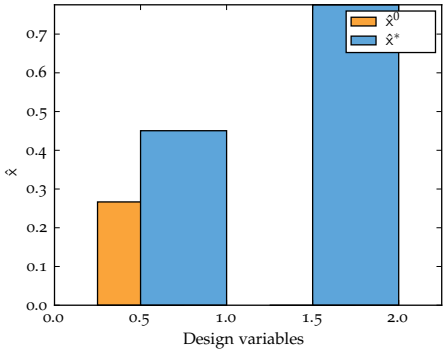


Figure 2.1.8: Normalized design variables at start $\hat{x}^0$ and optimum $\hat{x}^*$

Constraints

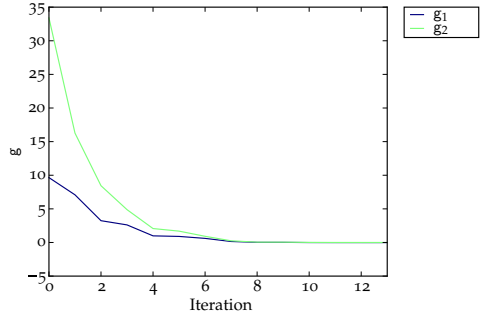


Figure 2.1.9: Convergence of inequality constraints g

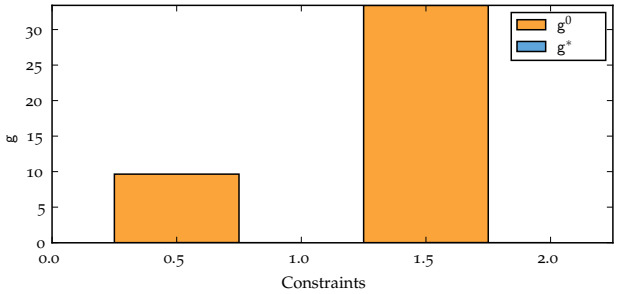


Figure 2.1.10: Inequality constraints at start g^0 and optimum g^*

2.2 Tables

Design variable	Symbol	Start value x^0	Lower bound x^L	Upper bound x^U	Optimal value x^*
1	x_1	1.8	1.0	4.0	2.3516
2	x_2	1.0	1.0	4.0	3.3269

Table 2.1: Details of design variables x

Response	Symbol	Start value	Optimal value
Objective function	f	1.8	7.8235
Inequality constraint 1	g_1	9.6481	-0.016
Inequality constraint 2	g_2	33.4067	-0.0

Table 2.2: System responses

Property	Symbol	Value
First-order optimality	$\|\nabla\mathcal{L}\|$	24.4446
Lagrangian multiplier of g_1	λ_{g_1}	3.9203

Table 2.3: First-order optimality as well as non-zero Lagrangian multipliers

Property	Symbol	Value
Shadow price of g_1	S_{g_1}	-inf

Table 2.4: Shadow prices

Property	Symbol	Value	Unit
Number of iterations	n_{it}	14	[−]
Number of evaluations	n_{eval}	47	[−]
Starting time	t_0	11 : 16 : 49	hh : mm : ss
Ending time	t_{end}	11 : 16 : 49	hh : mm : ss
Elapsed time	t_{opt}	00 : 00 : 00	hh : mm : ss

Table 2.5: Properties of optimization run