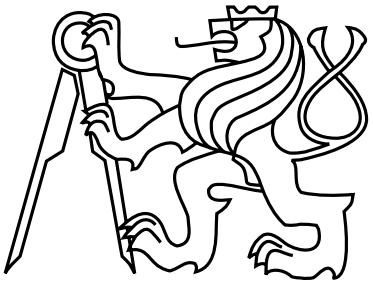
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering



Bachelor's thesis

Numerical optimization of industrial processes

Lenka Caletková

Supervisor: Ing. Jan Šulc

Study Programme: Cybernetics and Robotics

Field of Study: Systems and control

Date: May 2013

Acknowledgements

I would like to show my greatest appreciation to my supervisor Ing. Jan Šulc, who was always willing to help me with this project - even in his extra time - during the last two semesters. Without his encouragement and guidance this project would not have materialized. Many thanks belong to all my friends, who supported me in my tough times. Last but not least, I want to thank my supportive sister Jana and my great parents, who enabled me to study at the CTU, always stood by my side and gave me all their love.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24. 5. 2013 Cullková

Abstract

This thesis handles the conception of numerical optimizations and presents Python libraries, which are able to solve optimization problems. It compares their usability, stability and computing possibilities. All these aspects were tested on two specific industrial processes - control of the tunnel ventilation and heating systems control in buildings. A functional code in Python programming language for performance optimization of tunnel ventilation represents the outcome of this thesis.

Abstrakt

Tato bakalářská práce se věnuje pojmu numerické optimalizace a prezentuje knihovny programovacího jazyka Python, které je možné použít pro řešení optimalizačních úloh. Zároveň porovnává jejich použitelnost, stabilitu a výpočetní schopnosti. Všechny tyto aspekty byly testovány na dvou konkrétních průmyslových procesech - řízení ventilace v tunelech a řízení tepelných systémů v budovách. Výstup této práce představuje funkční kód v jazyce Python, který má za úkol optimalizovat výkon při ventilaci tunelu.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

BACHELOR PROJECT ASSIGNMENT

Student: **Lenka Caletková**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Bachelor Project: **Numerical optimization of the industrial processes**

Guidelines:

1. Compare the functionality and possibilities of the optimization libraries for the Python programming language (e.g. CVXOPT, APM, ...).
2. Analyse the computing possibilities, stability and usability of these optimization libraries for the control of the industrial processes – mainly heating systems in buildings and ventilation control in the road tunnels.
3. Choose the most suitable library to write a program for the control of the pollution concentration in the road tunnels.

Bibliography/Sources:

[1] Boyd S., Vandenerghe L. : Convex Optimization. http://www.stanford.edu/boyd/
[2] D.G. Luenberger: Linear and Nonlinear programming. Addison-Wesley Co. Reading, 1989
[3] J.Štecha : Optimální rozhodování a řízení. Skriptum ČVUT FEL, 2000
[4] www.python.org.

Bachelor Project Supervisor: Ing. Jan Šulc

Valid until the winter semester 2013/2014

prof. Ing. Michael Šebek, DrSc.
Head of Department

L.S.

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 7, 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Lenka Caletková**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Numerické optimalizace průmyslových procesů**

Pokyny pro vypracování:

1.Porovnejte funkčnost a možnosti knihoven pro matematickou optimalizaci v jazyku Python (např. CVXOPT, APM, apod.).
2.Analyzujte a vyhodnoťte výpočetní možnosti, stabilitu a použitelnost jednotlivých optimalizačních knihoven pro řízení průmyslových procesů - zejména řízení vytápění budov a řízení ventilace silničních tunelů.
3.Vyberte nejvhodnější knihovny, pomocí nichž napište program, který bude řešit řízení koncentrací zplodin v silničních tunelech.

Seznam odborné literatury:

[1] Boyd S., Vandenerghe L. : Convex Optimization. http://www.stanford.edu/boyd/
[2] D.G. Luenberger: Linear and Nonlinear programming. Addison-Wesley Co. Reading, 1989
[3] J.Štecha : Optimální rozhodování a řízení. Skriptum ČVUT FEL, 2000
[4] www.python.org.

Vedoucí: Ing. Jan Šulc

Platnost zadání: do konce zimního období 2013/2014

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 28. 1. 2013

Contents

Chapter 1

Introduction

In the last century, the term "mathematical optimization" was brought to light thanks to works of great mathematicians like Fermat, Lagrange, Newton or Gauss. They defined the important basics, which created a path for the nowadays known definition of numerical optimization. This field had been developed during decades, but the 'boom' of this concept has took place in the last 20 years, when the computer-time came. It was almost impossible to solve even simple optimization problems by ourselves, but with computers, there was a chance to improve and use this strong concept. It is spreading out through many fields - like economics, chemistry, energetics and many others.

Numerical optimization is in these days an important part of control systems, because we can achieve many savings through this 'tool'. At our faculty we are using optimization for ventilation control of complex road tunnels, chemical processes, or heating of buildings [24, 10]. To show the potential of this method, let us show you an example. Our control department has been using optimization, accurately Model predictive control, for heating system of our faculty building and the savings has been between 15 and 30 percent of energy in comparison to the previous heating control system [10].

The aim of this thesis is to find and test possible Python libraries, that can be utilized for numerical optimization of some processes. The numerical computing environment MATLAB is usually used for this purpose. It is very popular in academic domain and verified in practise and can be described as a stable and robust software, but it is really expensive in commercial sphere. Therefore small companies can not afford this software, so Python and some of its libraries offer a great chance to create software tools also for such clients. This thesis should find Python libraries, which could be a serious competition for MATLAB.

1.1 Outline of the thesis

- **Chapter** 1 - **Introduction** presents the subject and aims of this thesis.
- **Chapter** 2 - **Numerical Optimization** explains this concept, divides it into the sub-fields and shows us some practical examples.
- **Chapter** 3 - **Python libraries** lists all the used software and describes their possibilities to solve different types of optimization. This part includes few sample codes, which demonstrate the differences among the libraries.
- **Chapter** 4 - **Testing of code for special industrial processes** describes the implementation of this thesis. That means comparison of used software for two concrete industrial processes - Heating control of the CTU[1] building in Prague and the Ventilation control of the tunnel Blanka.
- **Chapter** 5 - **Conclusion** describes the advantages and disadvantages of tried software, chooses the best possible Python library for tunnel ventilation control and presents possible future improvements.

[1]The abbreviation 'CTU' presents the Czech Technical University in Prague.

Chapter 2

Numerical optimization

Before starting writing a code, we should first understand, what numerical or mathematical optimization means. The word 'optimal' is often used, but not always correctly. To describe something as optimal, we first have to define the conditions, for which we can consider this solution is truly optimal. Mathematical optimization or just optimization has two parts - objective function (or cost function) and constraints, which define the set, in which we are looking for the optimal solution. In general first we have to define an optimization problem and then we seek the best solver for this problem. Let us define the mathematical optimization problem.

For optimization variable $\mathbf{x} = (x_1, ..., x_n)$

$$\begin{aligned} &\text{minimize} && f_0(\mathbf{x}) \\ &\text{subject to} && f_i(\mathbf{x}) \leq b_i, i = 1, ..., m. \end{aligned} \tag{2.1}$$

The $f_0 : \mathbf{R}^n \longrightarrow \mathbf{R}$ is the objective function and $f_i : \mathbf{R}^n \longrightarrow \mathbf{R}$ are constraint functions and the constants b_i represent the bounds of the constraints [5, pp. 1].

Our demanding optimal solution x^* has to fulfil for any real z, which satisfies the constraints $f_0(z) \geq f_0(x^*)$. This vector x^* is global extreme and "*the smallest objective value among all vectors that satisfy the constraints*" [5].

2.1 Sub-fields of numerical optimization

Although the optimization can be divided into many sub-fields, we will have a closer look only at few of them - especially on that sub-fields, that will used for our industrial processes.

2.1.1 Convex optimization

This sub-field studies the cases, when the objective function and the constraint set are convex (concave) for minimization (maximization). And this case guarantees, that we are able to find **global minimum** (**maximum**) of this function. Therefore a lot of mathematicians and engineers try to describe an optimization problem as a convex or concave, because it ensures that they will find the sought optimum. Mr. Boyd and Mr. Vandenberghe wrote an important script [5] about this part of numerical optimization.

2.1.1.1 Least-squares problems

This sub-field is one of the simplest convex optimization problems, because it does not contain any constraints and the objective function is a sum of squares of terms: [5, pp. 4]

$$\text{minimize} \quad \|A\mathbf{x} - b\|_2^2 = \sum_{i=1}^{k} (a_i^T \mathbf{x} - b_i)^2, \tag{2.2}$$

where the a_i^T are rows of the matrix $A \in \mathbf{R}^{k \times n}$ and $b \in \mathbf{R}^k$ represents a vector. This simple optimization problem is often used e.g. for data fitting.

2.1.1.2 Linear programming

In case the objective function is linear and the set defined by constraints using only the linear functions, we can describe it as linear programming [5, pp. 6].

$$\begin{aligned} \text{minimize} \quad & c^T \mathbf{x} \\ \text{subjected to} \quad & a_i^T \mathbf{x} \leq b_i, \end{aligned} \tag{2.3}$$

where $c \in \mathbf{R}^n$ is vector, $a_1, ..., a_m$ and $b_1, ..., b_m \in \mathbf{R}$ present the scalars.

2.1.1.3 Semidefinite programming

This type of convex optimization can be described by the following equations :

$$\begin{aligned} \text{minimize} \quad & c^T \mathbf{x} \\ \text{subjected to} \quad & x_1 F_1 + ... + x_n F_n + G \preceq 0 \\ & A\mathbf{x} = b, \end{aligned} \tag{2.4}$$

where are G,$F_1, ..., F_2 \in \mathbf{R}^{n \times n}$ and $A \in \mathbf{R}^{p \times n}$ the matrices , $c \in \mathbf{R}^p$, $b \in \mathbf{R}^m$ vectors [5, pp. 168].

2.1.1.4 Quadratic programming

This subgroup has a convex quadratic objective function and constraints are affine [5, pp. 152].

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} x^T P x + q^T x \\ \text{subjected to} \quad & G^T x \leq h \\ & Ax = b, \end{aligned} \tag{2.5}$$

with matrices $P \in \mathbf{R}_+^{n \times n}$, $G \in \mathbf{R}^{m \times n}$, $A \in \mathbf{R}^{p \times n}$ and vectors $q \in \mathbf{R}^n$, $h \in \mathbf{R}^m$ and $b \in \mathbf{R}^p$.

We want to clarify as well, that the term quadratic programming is sometimes used for non-convex optimization. Therefore it is important to understand, that not all the optimization problems with quadratic objective function are convex.

2.1.2 Other optimization problems

It is natural that not every optimization problem can be described by convex programming, so let us look at some other types of programming.

2.1.2.1 Non-linear programming

Most of the systems in our world are described by non-linear function and therefore the non-linear programming is concerned with cases, where the objective or constraints include some non-linear parts. Solvers for non-linear optimization problems are essential the right setting of initial condition, because the local minimum instead of the global would be found mostly. With variable initial conditions we can find different values of an objective function.

2.2 Practical examples of using optimization

As we will discover in the upcoming chapter, there are plenty usages of optimization. Mr. Boyd and Mr. Vandenberghe name three illustrative examples for understanding this concept in their books [5, 6].

2.2.1 Embedded optimization

In the last decades the size of computers and their components decreased and therefore it became beneficial to optimize the device sizing in electrical circuits. For this problem the *objective function* is a **power consumption**, the *variables* are **widths and lengths of the device** and the *constraints* are **manufacturing limits, timing requirements** and **maximum area** [5, 6].

2.2.2 Portfolio optimization

When investing in assets there is a big risk of loosing invested money and so it would be a competitive advantage to possess a control system, that would find the most risk-free way for your investment. For this purpose the *objective function* could be **overall risk** or **return variance**, the *variables* **amounts invested in different assets** and the budget, maximum and minimum investment per asset and **minimum return** would be our *constraints*. And by this way we can decrease the risk of loosing our money [5, 6].

2.2.3 Data fitting

In this case we are looking for the best fitting model for our observed data. The *objective function* can represent the **measure of misfit or prediction error**, the *variables* **model parameters** and *constraints* can feature **prior information** and **parameter limits** [5, 6]. An example is figured in the picture 2.2.3.

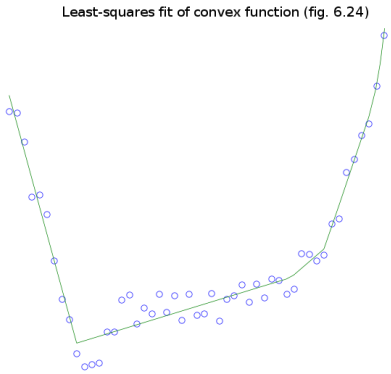


Figure 2.1: An example of using optimization for data fitting [5, 1].

2.2.4 Model predictive control (MPC)

This is a concept of *advanced control systems*, such as heating control of intelligent buildings, or control of chemical processes. This method works with *complex dynamic behaviour* of the system and is aiming mainly to "*minimize the performance criterion in the future, that would possibly be subject to constraints*" [2]. To understand this concept easily imagine a chess play. We are planning during this play e.g. five moves ahead although we are not absolutely sure, what the opponent will do. In the MPC, as in chess, there are dependent and independent variables [2].

As an illustrative example of MPC, we can present the heating control of a building, where we can calculate with the weather forecast and according to it, it will change the control of our system. During winter's nights the rooms of the building are getting colder (e.g. below 15 degrees Celsius) and in the morning we need the rooms to be warm. Even if someone sets the heater to its maximum at 6 or 7 p.m., at 8 p.m. there still will not reach the regular temperature of 20 degrees. Such a way of heating control - a thermostat is simple to implement, but is not sustainable in our modern society, which is increasingly demanding energy. Solution is the MPC, that maintains the temperature (e.g. about 19 degrees) and controls the heaters to achieve the optimal temperature before first people will come. This style of control is also more beneficial for our economy and ecology.

Chapter 3

Python libraries

In this chapter we focus on possible Python libraries, which can be used for numerical optimizations. We concentrate on the ability of solving different optimization problems and a usage of these libraries and their licenses etc.

3.1 Licenses

Before we start with the first Python library, let us look on the types of licenses. For this work it is important to have an overview of all kinds of licences, because we are looking for a software, which is free and can be used for commercial purposes. There are few terms to understand before we begin with the specific licenses.

Copyright is defined according to Wikipedia as "*a legal concept, enacted by most governments, giving the creator of an original work with exclusive rights to it, usually for a limited time*" [19]. It gives the author(s) rights to say, who can use his (their) ideas and who will profit from them. This concept belongs into Intellectual property law, but it slows down all new development in 'software-world'.

That is why Richard Stallman came up with the idea of **copyleft** along with the free software [21]. It ensures, that the software will be always open to anyone and everyone can use it for any purpose. The only condition is, that any new code based on another code with copyleft license should follow the same idea - having the open code. This should ensure free and quicker development of new software.

3.1.1 GPL

GPL is the most popular and most used license for free software and its abbreviation means General Public License. This license "*guarantees end users (individuals, organizations, companies) the freedom to use, study, share (copy) and modify the software*" [21]. It is copyleft license, which declares, that any new software, which would be using code GPL licensed, should have the same license - the GPL license.

The author has to provide the open code - not just binary - with the GPL license to allow others to use his code. This license can be used for commercial purposes, but always needs to follow the rules of it - added license and open code.

Originally this license was written by Richard Stallman for GNU project [21]. Nowadays there are three versions of this license, due law gaps, which enabled to some users to misuse this idea. Third current version is naturally most used today.

3.1.2 BSD, CPL and MIT licenses

'Berkeley Software Distribution' [18], 'Common Public License' (published by IBM) [20] and MIT license [17] are more permissive than GPL, because there are not copyleft licenses. The author can use the code, without necessity of using the same license as well. There are more types of BSD licenses and some of named licenses are compatible with GPL and some not. Below this subsection, you can find the table (3.1) with an overview of the free licenses.

	GPL	BSD	New BSD	CPL	MIT
Copyleft	Yes	No	No	No	No
GPL compatible	Yes	No	Yes	No	Yes
Linking code with another license	No	Yes	Yes	Yes	Yes

Table 3.1: Overview of the licenses for free software [21, 18, 20, 17].

The issue of using some software with different licenses is not so simple and we have to be sure, that we are able to use them. Essential for our work is the fact, that we can use the code with any of the licenses named above for commercial purposes, but with GPL license we have to provide also an open code.

3.2 Python libraries for optimization problems

Seeking a stable optimization software written in Python language is not a simple task, because almost all of these libraries are made by volunteers and are free of charge, therefore they are not so perfectly tested as a commercial software and they can contain errors.

Now we will define a simple example of a optimization problem[1], which helps us to demonstrate the differences among Python libraries, which will be tested for the control of industrial processes.

$$\begin{aligned} \text{minimize} \quad & x_1^2 + x_2^2 + 3x_1x_2 + 8x_2 \\ \text{subject to} \quad & 2x_1 + 3x_2 \geq 0 \\ & x_1 + 2x_2 \leq 10 \\ & 4x_1 + 5x_2 = 0 \\ & 100 \geq x_1 \geq -80 \\ & 180 \geq x_2 \geq -200 \end{aligned} \tag{3.1}$$

3.2.1 SciPy and its package Optimize

The abbreviation "Scientific Python" suggests, that SciPy is "*open-source (BSD-new license) software for mathematics, science, and engineering*" [16]. In this library we can find a lot of useful packages, however for our purpose we concentrate on the package *Optimize*, which includes some of commonly used optimization algorithms. This package provides two useful function, which can be used for constrained minimization of multivariate scalar functions:

- **fmin_cobyla**
  This function solves non-linear problems and minimizes an objective function using the Constrained Optimization BY Linear Approximation (COBYLA) method. The constraints can be defined only in the form $g(x) \geq 0$ and this makes the *fmin_cobyla* more difficult to use. The equalities need to be described as inequalities. And we can achieve it through following trick, which we explain using the equation from optimization problem (3.1). The equation

  $$4x_1 + 5x_2 = 0 \tag{3.2}$$

  can be rewritten as two inequalities and it describes the same case.

  $$\begin{aligned} 4x_1 + 5x_2 &\geq 0 \\ -4x_1 - 5x_2 &\geq 0 \end{aligned} \tag{3.3}$$

  The following example demonstrates how to formulate and solve the optimization problem (3.1) via the function *fmin_cobyla*.

[1] Optimization problems are usually not defined with specific initial conditions, but for this illustrative example we set the initial conditions to $x_{01} = 10, \quad x_{02} = 20$.

```
import scipy.optimize as opt

def objective(x):
    return x[0]**2 + x[1]**2 +3*x[0]*x[1] + 8*x[1]

def constr1(x):
    return -2*x[0] - 3*x[1]

def constr2(x):
    return x[0] + 2*x[1] - 10

def constr3(x):
    return 4*x[0] + 5*x[1]

def constr4(x):
    return - (4*x[0] + 5*x[1])

def constr5(x):
    return x[0] + 80

def constr6(x):
    return -x[0] - 100

def constr7(x):
    return x[1] + 200

def constr8(x):
    return -x[1] - 180

sol = opt.fmin_cobyla(objective,[10,20],[constr1,constr2,constr3,constr4,
                      constr5,constr6,constr7,constr8],rhoend=1e-7)

print sol
```

Listing 3.1: Sample code of using function *fmin_cobyla*, which solves the optimization problem (3.1).

- **fmin_slsqp**
  This function can be used only for optimization problems with only several variables, which is its disadvantage. It minimizes an objective function using Sequential Least SQuares Programming. This function has, in comparison to *fmin_cobyla*, the equality constraints, lower and upper bound of optimization variables as inputs.

3.2.2 CVXOPT

"*CVXOPT is a free (GPL license) software package for convex optimization based on Python programming language*" [1]. The abbreviation for this library is obvious - ConVeX OPTimization. The authors of this library are Martin Andersen, Joachim Dahl and Lieven Vandenberghe. We name few optimization problems, which should CVXOPT be able to solve:

- Linear Programming
- Quadratic Programming
- Semi-definite Programming

For quadratic programming CVXOPT serves the function **qp**, whose inputs are in general following matrices (P,G,A) and vectors(q,h,b):

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x^T Px + q^T x \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b \end{aligned} \tag{3.4}$$

We can rewrite our sample optimization problem (equation n. 3.1) using matrices and vectors:

$$\begin{gathered} \text{minimize} \quad \tfrac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 2 \\ 4 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 8 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ \begin{bmatrix} -2 & -3 \\ 1 & 2 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 10 \\ 100 \\ 80 \\ 180 \\ 200 \end{bmatrix} \\ \begin{bmatrix} 4 & 5 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0 \end{gathered}$$

The script for solving the optimization problem with CVXOPT can we see in the following listing.

```
from cvxopt import solvers, matrix

P = matrix([2,4,2,2],(2,2))

q = matrix([0,8],(1,2))

G = matrix([-2,1,1,-1,0,0,-3,2,0,0,1,-1],(6,2))

h = matrix([0,10,100,80,180,200],(6,1))

A = matrix([4,5],(1,2))

b = matrix(0)

x0 = matrix([10,20],(2,1))

# solver CVXOPT
sol=solvers.qp(P, q, G, h, A, b,x0)
print(sol['x'])
```

Listing 3.2: Sample code of using library CVXOPT for solving the optimization problem defined by (3.1).

This listing shows, that the function *qp* is similar to MATLAB *quadprog* function (subsubsection 4.2.2.1), which has almost the same inputs like *qp* (missing lower and upper bounds for *qp*).

3.2.2.1 Examples of using CVXOPT

In the book Convex optimization [5] we can discover a lot of optimization problems, which were solved by CVXOPT, e.g. for data fitting, depicted in 2.2.3, was utilized the function *qp*.

3.2.3 CVXPY

This Python library is being developed at the Stanford university and should handle to solve problems, which are described in natural mathematical form. CVXPY presents a modelling language for CVXOPT software. This software is in the beginning of development, because CVXPY is in the version 0.0.1 [11]. Therefore it can not solve our optimization problems, which are too complex[2], because it causes problems with memory consumption.

[2]Our optimization problem obtains more than 600 variables.

3.2.4 APMonitor

APMomitor is the on-line optimization software with BSD licence, which was developed for large-scale[3] problems. The input is differential-algebraic equations (DAE) and with them can be described almost every optimization problem and APMonitor should be able to solve them.

The speciality of this software is its own modelling language, with which we describe the optimization problem in APM-file. APMonitor offers two different interfaces - the one for MATLAB and the other for Python, which we apply in this thesis.

To solve our sample optimization problem, defined by equation (3.1), we have to first write the APM-file (listing 3.3) with its unique modelling language.

```
Model Example
        Variables
                x[1]=10  , >=-80  , <=100
                x[2]=20  , >=-200  , <=180
        End Variables


        Equations
                minimize x[1]*x[1] + x[2]*x[2] + 3*x[1]*x[2] + 8*x[2]

                2*x[1] + 3*x[2] >= 0
                x[1] + 2*x[2] <= 10
                4*x[1] + 5*x[2] = 0

         End Equations
End Model
```

Listing 3.3: Sample code in APMonitor modelling language, which implements the optimization problem (3.1).

[3] A large-scale problem describes such an optimization problem, whose number of variables and constraints is greater than 1000 [4].

Now we apply the Python interface to send the APM-file to the APMonitor server, where this problem would solve. We can accomplish it by running this following code (3.4).

```
from apm import *
from random import randint
#server
s = 'http://xps.apmonitor.com'

# application + random number
a = 'zk' + str(randint(2,999))

#clear server
apm(s,a,'clear all')

#load model
apm_load(s,a,'text.apm')

#change solver
apm_option(s,a,'text.solver',3)

#solve
output = apm(s,a,'solve')
print output

#retrieve solution
(csv, solution) = apm_sol(s,a)

#open root file
apm_web_root(s,a)
#open solution in web browser
apm_web_var(s,a)
```

Listing 3.4: Python interface for sending the APM-file to the server.

3.2.4.1 Examples of using APMonitor

This software is utilized e.g. for biology - *cell cultivation*, or *blood glucose response of an insulin dependent patient*, for aerospace system an *unmanned Aerial systems*. In chemistry industry it is implemented e.g. for *minimizing Gibbs free energy* and there are much more APMonitor applications in energetics, financial, or mechanical systems - more information about APMonitor usage could be found on their website [3].

3.2.5 OpenOpt

This open-source (BSD license) project, made by junior researcher at National Academy of Sciences of Ukraine Cybernetics Institute, Dmitrey L. Kroshko, is intended to create universal 'tool' for solving optimization with Python libraries or packages [8]. This project seems to be really interesting for testing one optimization problem with more solvers by changing only few lines of code. This package should handle to solve following optimization problems:

Matrix Problems Group

- Linear Problems (LP),
- Quadratic Problems (QP),
- Linear Least Squares Problems (LLSP),
- Semi-definite Problems (SDP).

Non-Linear Problems Group

- Non-Linear Problems (NLP),
- Global Problems (GLP).

```
from numpy import matrix
from openopt import QP
# defines the matrices of the QP problem.
Q = matrix('2 2; 4 2')
p = [0,8]

A = matrix('-2 -3; 1 2')
b = [0, 10]

Aeq =[4,5]
beq = 0

lb = [-180,-200]
ub = [100,80]

#function QP has not as an input the initial conditions
p = QP(H=Q, f=p, A=A, b=b, Aeq=Aeq, beq=beq, lb=lb, ub=ub)
r = p._solve('qlcp', iprint = 0)
```

Listing 3.5: Sample code of using OpenOpt with the solver 'glpc' for our sample optimization problem defined by (3.1).

Chapter 4

Testing of code for special industrial processes

After having introduced the conception of optimization and the Python libraries, we are going to discuss the implementation of this bachelor thesis, which is the testing of Python libraries for the specific industrial processes, that we want to control. Therefore we start this chapter with a brief definition of the industrial processes, where we are applying some of the optimization software. Following this part we are going to test the written software and discuss the results and their correlation with MATLAB. The most important aspects of the comparison for this work will be a **CPU time**, **resulting values of the objective functions** and a **feasibility of the outputs**.

4.1 Industrial processes

Although there are many industrial processes, which could be optimized, we are interested in this work in HVAC technology [22]. The **HVAC** (meaning heating, ventilation, and air conditioning) is a technology used for the design of medium to large industrial or office buildings. This method belongs to the mechanical engineering, but with advanced types of HVAC technology architects or electrical engineers work as well. Nowadays the price of energy is increasing and we are running out of some important energy sources, control of HVAC technologies can help us to spare lot of energy and reach sustainability. For specific information about HVAC systems please look to Wikipedia [22]. Both of the industrial processes, we want to optimize, are covered under HVAC technologies.

4.1.1 Tunnel ventilation control

The first question, that might come to your mind, is why there is even any need of ventilation for tunnels. In tunnels and other industrial building there are strict air quality requirements and therefore the tunnels need to be ventilated. The larger and complex tunnels demand the advanced ventilation control, because the monthly energy costs for them are within the range of millions Czech crowns. According to the fact, that the ventilation control can manage

savings between 20 and 30 percent, the initial investment into this advanced control system will return.

Nevertheless the complex or long-distance tunnels require ventilation otherwise the unventilated exhaust gas can cause death of the drivers. Control ventilation could be helpful by special occasion, e.g. in case fire breaks out inside of a tunnel and then the control of jet fans can also save human lives.

Prague, the capital city of the Czech Republic, has over one-million inhabitants and during the day there are lots of traffic jams, thus there was a big pressure on the municipality of Prague to build the City Ring Road. The tunnel complex Blanka was projected as the part of the Prague City Ring and it should become with 5.5 kilometres the longest city tunnel in central Europe in 2014[1]. Due to the fact, that the Blanka is a city tunnel, the air quality requirements are much higher then by tunnels in open landscape, because the leak of exhaust gases should be minimize at the exit junctions, which lead to the parts of inner Prague [9]. For this reason it was necessary to find the right settings of jet fans flow rate securing their minimum energy consumption in compliance with the strict air quality constraint.

Figure 4.1: Finishing works of the mined tunnel Královská obora, a part of the tunnel complex Blanka [12].

The tunnel complex Blanka, whose part is figured in the the picture 4.1, is described as a non-linear system by 74 variables, which describe the air-flow rate in different tunnels sections, number of running jet fans and also the air-flow in four ventilation machine rooms. This optimization problem is defined by non-linear polynomial objective functions and the non-linear constrains are only in the form of the inequalities [24, 9].

Ventilation of a tunnel in general is working in three different modes - normal operation, fire ventilation and preventilation. The last named mode starts when there is a suspicion of a fire outbreak, which can be indicated by camera, smoke or heat detectors. If the suspicion is confirmed by the traffic controller, the fire ventilation starts. The normal operation modes divide we into three sub-modes: natural airflow (or zero level protection), first and second

[1]The year 2014 is the estimated time of the Blanka completion [9].

level protection. Both, the first and second level protection, maintain the values of nitrogen oxides and carbon monoxide under the defined limits according to the law and it minimize the exhaust of cars from all exit tunnel junctions [9].

In case any of the limits should be exceeded, the ventilation optimization is required to choose the best action, by setting the jet fans speed in such way, that the performance of all devices will be minimized and will satisfy the constraints.

4.1.2 Control of building heating system

The energy costs for big buildings are about hundreds of thousand Czech crowns and therefore it is meaningful to minimize these expenses for energies.In 2010 the team from our department had a unique chance to find out the impact of using MPC for heating of the Czech Technical University building at Dejvicka (part of Prague). This building (fig. 4.2) is divided into three blocks B1,B2 and B3 and the parts B1 and B2 are insulated in contrast to non-insulated sector B3. This gives our faculty a great opportunity to find out the influence of insulation on the control heating system (fig. 4.3).

Figure 4.2: The nearest block in front is B1, on the right side from the block B1 are blocks B2 and B3 [7].

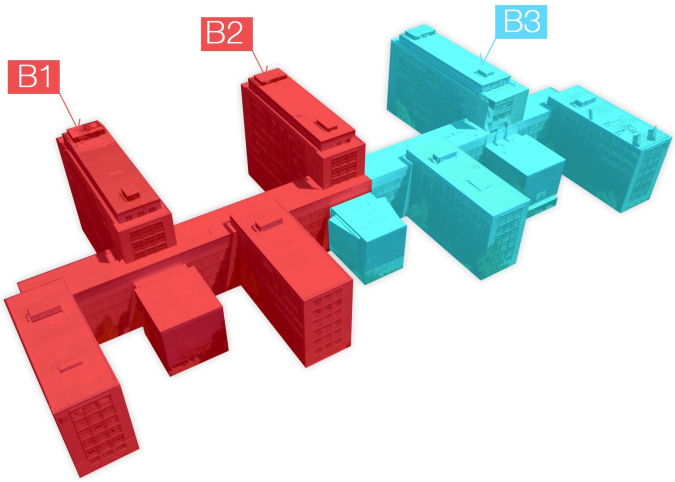


Figure 4.3: A sketch of the building [7].

Now let us formulate the MPC problem for the heating control of the CTU building. For our linear, time-invariant, discrete model of our heating system

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \tag{4.1}$$

we are looking for the best control by minimizing the following objective function

$$J = \sum_{k=0}^{N-1} q(k)(y(k) - z(k))^2 + r(k)u(k)^2, \tag{4.2}$$

where the N refers the horizon[2] of the prediction. This objective function has to subject to following constraints:

$$\begin{aligned} u_{min} \leq u(k) \leq u_{max} \\ y_r(k) \leq z(k) \\ |u(k) - u(k-1)| \leq \triangle_{max}, \end{aligned} \tag{4.3}$$

where u_{min} and u_{max} are the lower and upper bounds of the control signal. The variable $z(k)$ has to be greater than the desired value y_r. The term $\triangle_{max}$ presents the maxim possible change of the control signal [10].

What is important for this thesis is the fact, that this MPC problem can be rewritten into the convex quadratic programming, which makes this problem solvable with such optimization functions like MATLAB *quadprog* or CVXOPT *qp*. These optimization problems are described with more than 600 variables and more than 800 constraints[3].

4.2 Comparison

In this part we are coming to the crucial part of this thesis, where we will compare the Python libraries also with MATLAB. The most important *aspects* for our comparison are the ability of achieving the **minimum** of the objective function and the **stable** solution, the **CPU time** of software. The commented codes can be found in the appendix A, whose results are discussed in this section.

4.2.1 Ventilation control of the tunnel Blanka

Finding the optimum of running fans for their minimum energy consumption in compliance with the strict air quality constraints for the tunnel complex Blanka is defined as a non-linear problem. The selection of right initial conditions is really important for non-linear solvers, because all possible solvers will find a local minimum, which can differ with variable

[2] As mentioned before, MPC works with future possible variables and the horizon presents the time interval, which is used for the prediction. In each step k has to be the optimization recalculated and the horizon moves one step forward, otherwise the prediction will not work and the MPC will fail.

[3] For the 128-hour horizon, there are over 3300 variables, 1500 inequalities and 2800 equalities. Therefore all these optimization problems can be described as large-scale problems.

initial conditions. Despite the solution among solvers will be different, that they can be still usable. For the testing all software we have prepared four scenarios of ventilation control - the preventilation, zero, first and second level protection. You can see all the results[4] in the table 4.1.

4.2.1.1 MATLAB and fmincon

Above defined non-linear problem was in other project [24, 9] solved with MATLAB *fmincon* function [13], which is used for constrained non-linear optimizations defined as

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & c(x) \leq 0 \\ & c_{eq}(x) = 0 \\ & Ax \leq b \\ & A_{eq}\, x = b_{eq} \\ & lb \leq x \leq ub. \end{aligned} \tag{4.4}$$

For solving this problem we call *fmincon* [13] function with the following code.

```
x = fmincon(f,x0,A,b,Aeq,beq,lb,ub,@nonlcon)

function [c,ceq] = nonlcon(x)
c = ...      % Compute nonlinear inequalities at x.
ceq = ...    % Compute nonlinear equalities at x.
```

The **f** address the objective function, **x0** are initial conditions, **A** and **b** represent the inequalities, **Aeq** and **beq** equalities. The values of **lb** and **ub** are lower and upper bounds for the optimization variables. The input function **nonlcon** describes the non-linear equalities and inequalities.

4.2.1.2 SciPy

Among all possible optimization functions, that SciPy offers, we choose **fmin_cobyla** for this non-linear programming. Sample code of using *fmin_cobyla* can be found in the listing n. 3.1. With exception of the second level protection, the results coincide with MATLAB. The process time differs from MATLAB strangely, because the Zero level protection takes almost the same time when using MATLAB (6 s) than SciPy (7.8 s). On the other hand other modes using SciPy take at least double the computing time than MATLAB uses.

For the last scenario - second level protection - SciPy find nonsensical value of the objective function. As a result for this scenario the *fmin_cobyla* wrote down that it is infeasible to find solution and according to this fact, we can not use this solution for the control. It is not suitable to use this library for the control.

[4] All optimization problems were solved on the computer with the dual core processor with a frequency 1.33 Ghz.

Used software	Computing time [s]	Objective function [kW]	Feasible solution
Scenario: Preventilation			
MATLAB (fmincon)	2.51	83.3	Yes
SciPy (fmin_cobyla)	11.21	83.3	Yes
APMonitor (APOPT)	0.04	83.3	Yes
OpenOpt (ralg)	15.42	104.0	Yes
Scenario: Zero level protection			
MATLAB (fmincon)	6.07	85.9	Yes
SciPy (fmin_cobyla)	7.76	272.9	Yes
APMonitor (APOPT)	0.03	85.9	Yes
OpenOpt (ralg)	47.21	245.4	Yes
Scenario: First level protection			
MATLAB (fmincon)	7.11	176.5	Yes
SciPy (fmin_cobyla)	17.54	176.9	Yes
APMonitor (APOPT)	0.04	176.5	Yes
OpenOpt (ralg)	60.7	281.5	Yes
Scenario: Second level protection			
MATLAB (fmincon)	42.75	1462.7	No
SciPy (fmin_cobyla)	22.15	-153.1	No
APMonitor (APOPT)	0.11	1473.3	No
OpenOpt (ralg)	85.85	1886.7	No

Table 4.1: Overview of the achieved results for the ventilation control of the Blanka tunnel.

4.2.1.3 APMonitor

This on-line software (using APOPT solver) shows with its results and process time, that it can well compete with MATLAB. The values of the objective function, except the second level protection, are the same as MATLAB found and the process times for all test cases are under 1 second. Due to this results APMonitor software much quicker then MATLAB. Thanks to its precise solution and quick CPU time **APMonitor** 'wins' among these libraries for this non-linear programming.

4.2.1.4 OpenOpt

For solving a non-linear problem we are using the solver '*ralg*' written by Mr. Kroshko. We wanted to try the solver '*ipopt*', which is a solver for MATLAB as well, but we had problems with adding the pyipopt[5] to the Eclipse platform and that is why we could not utilize it. NLP (Non-Linear Problems) include lots of solvers, but not all of them could find the feasible solution.

Despite all these problems, the solver *ralg* found feasible solution except for the second level protection. The values differ from MATLAB and it might be caused by shifting the initial values. The *ralg* solver had a problem with the initial conditions, because they must fulfil the equation $A_{eq}\,\mathbf{x} = b_{eq}$. After shifting all the initial conditions with respect to this equation, the *ralg* could find feasible values of the objective function. Because the values are higher, compared to other solvers, we do not recommend OpenOpt for this non-linear programming problem.

[5]Pyipopt (new BSD license) allows user to use the ipopt solver via Python language [23].

4.2.2 Heating system control of CTU's building

Model predictive control, which is translated into the quadratic programming, should end up always with the same objective function outcome for any used solver. So we will only compare the process times and the values of objective variables. For testing the software we prepared three scenarios - MPC with horizon of 24, 64 and 128 hours. An overview of the results is written in the table 4.2.

4.2.2.1 MATLAB and quadprog

The project [10] using MPC for heating control of our university building was held in 2010 and for solving we used MATLAB function *quadprog* [14], which can solve quadratic programming. In case we want to solve following problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x^T Hx + f^T x \\ \text{subject to} \quad & Ax \leq b \\ & A_{eq}x = b_{eq} \\ & lb \leq x \leq ub. \end{aligned} \tag{4.5}$$

we can use following script.

```
x = quadprog(H, f ,A, b , Aeq , beq , lb , ub , x0 )
```

The inputs **A**,**b**,**Aeq**,**beq**,**x0**, **lb** and **ub** are the same as for MATLAB *fmincon* function (subsubsection 4.2.1.1). Matrix **H** presents the quadratic coefficients and vector **f** the linear coefficients.

4.2.2.2 CVXOPT

This Python library has the function **qp** for the quadratic programming, which we used for finding the minimum of our objective function. The *qp* takes for every scenario at least double process time then MATLAB, but the results are fine - the minimum was found.

4.2.2.3 OpenOpt

For quadratic problems OpenOpt includes CVXOPT solver as well. Using it is naturally slower then just CVXOPT itself. However we focused on the solver '*qlcp*' (MIT license), which was developed by Enzo Michelangeli and IT Vision Ltd. MPC for 24 hour takes even less time then MATLAB *quadprog*, the other scenarios are with OpenOpt 1.6 slower than *quadprog*.

Software	Computing time [s]	Objective function [kW]
Scenario: MPC of 24-hour horizon		
MATLAB (quadprog)	3.69	1717.8
CVXOPT (qp)	8.74	1717.8
OpenOpt (qp)	3.37	1717.8
Scenario: MPC of 64-hour horizon		
MATLAB (quadprog)	41.88	5041.7
CVXOPT (qp)	107.64	5041.7
OpenOpt (qp)	55.24	5041.7
Scenario: MPC of 128-hour horizon		
MATLAB (quadprog)	271.94	5817.1
CVXOPT (qp)	885.55	5817.1
OpenOpt (qp)	420.17	5817.1

Table 4.2: Overview of the achieved results for the heating system.

Chapter 5

Conclusion

This thesis presented at its beginning a theoretical background of the optimization supported by few practical examples, followed by introduction of the Python libraries for solving optimization problems, where the principles of their implementation have been shown.

One of the main objectives of this thesis, was testing optimization libraries for Python and for the heating system control of the CTU building we found two usable libraries - OpenOpt with *qp* function and its solver 'qlcp' and CVXOPT with its *qp* function. Both of these software products found the minimum, however OpenOpt solved the problem two times quicker than CVXOPT.

The ventilation control of the Blanka tunnel was a bit more interesting, because the results differ among all used libraries. These non-linear optimization programs tested on four different scenarios brought interesting results. Every optimization functions, including MATLAB, were not able to find feasible solution for the second level protection.

OpenOpt and its solver **'ralg'** achieved feasible optimum, but its disadvantage are higher result values of the objective functions and longer CPU time in comparison with other tried libraries. This was possibly caused by the shift of the initial conditions, which was requested by the solver *ralg* as the origin initial conditions did not satisfy the equality constraints. We can conclude this part about OpenOpt, that it is not the right choice for our non-linear problem.

Testing the **SciPy** function **fmin_cobyla** for non-linear system optimization pleasantly surprised with good CPU time and the achieved values, which slightly differ from MATLAB results, except for the last mode - second level protection. For this scenario the COBYLA, as all the others, did not find any feasible solution. However this software could be used for this kind of optimization problems.

The last tested software was the **APMonitor**, which achieved incredible results. The solution of APMonitor was almost identical as in the case of MATLAB and the CPU time for all scenarios did not take more than 1 second, it is almost fifty times quicker then MATLAB. This is really the advantage of this free software. This software is available only on-line and therefore needs to be connected to the internet to solve the optimization problem. According to this disadvantage, there is a need of the backup control strategy in the case we want to use this software in the real-time control.

Due to the fact that these software are easily able to compete with MATLAB, we recommend them even for commercial use e.g. for smaller companies that cannot afford buying MATLAB.

The resulting functional Python codes for the optimization of tunnel ventilation control, using the APMonitor software and the *fmin_ cobyla* function, are on the attached CD along with other tested libraries for both the tunnel ventilation and the heating system control.

As far as the future progression is regarded, I see a possibility in the new developing modelling language CVXPY and testing the stability of APMonitor, because this software impressed us with its results. Hereafter to expand and base on this thesis we would like to create functional APMonitor code for quadratic programming - more specific for the heating system of the CTU building.

Bibliography

[1] ANDERSEN, M. – VANDENBERGHE, L. – DAHL, J. *CVXOPT* [online]. [cit. 10. 5. 2013]. Accessed on: <http://cvxopt.org/>.

[2] ANDERSON, B. – BENNICK, A. – SALCICCIOLI, M. *Model Predictive Control* [online]. [cit. 15. 5. 2013]. Accessed on: <https://controls.engin.umich.edu/wiki/index.php/MPC>.

[3] APMonitor Community. *APMonitor Optimization Suite* [online]. [cit. 10. 5. 2013]. Accessed on: <http://apmonitor.com/>.

[4] BENSON, H. Y. – SHANNO, D. F. – VANDERBEI, R. J. *A COMPARATIVE STUDY OF LARGE-SCALE NONLINEAR OPTIMIZATION ALGORITHMS* [online]. [cit. 15. 5. 2013]. Accessed on: <http://www.princeton.edu/~rvdb/tex/loqo5/loqo5_5.pdf>.

[5] BOYD, S. – VANDENBERGHE, L. Convex Optimization book, 2009. http://www.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf.

[6] BOYD, S. – VANDENBERGHE, L. Convex Optimization lectures, 2013. http://www.stanford.edu/~boyd/cvxbook/bv_cvxslides.pdf.

[7] JIŘÍ CÍGLER and JAN ŠIROKÝ. *MPC for heating system at Dejvicka* [online]. [cit. 30. 4. 2013]. Accessed on: <https://dev.rcware.eu:8036/MPCViewer/Info/DejviceInfo/DejviceInfo.html>.

[8] KROSHKO, D. L. *OpenOpt* [online]. [cit. 10. 5. 2013]. Accessed on: <http://www.openopt.org/>.

[9] NýVLT, O. – FERKL, L. – ŠULC, J. Reliability of tunnel ventilation control: A conceptual design, 2013.

[10] PRíVARA, S. et al. Model predictive control of a building heating system: The first experience, 2010.

[11] RUBIRA, T. T. D. *CVXPY documentation* [online]. [cit. 15. 5. 2013]. Accessed on: <http://www.stanford.edu/~ttinoco/cvxpy/>.

[12] SATRA, s.r.o. *Tunnel complex Blanka* [online]. [cit. 3. 5. 2013]. Accessed on: <http://www.tunelblanka.cz/>.

[13] The MathWorks, Inc. *Find minimum of constrained nonlinear multivariable function - MATLAB fmincon* [online]. [cit. 15. 5. 2013]. Accessed on: <`http://www.mathworks.com/help/optim/ug/fmincon.html`>.

[14] The MathWorks, Inc. *Quadratic programming* [online]. [cit. 15. 5. 2013]. Accessed on: <`http://www.mathworks.com/help/optim/ug/quadprog.html`>.

[15] The Python community. *Python Programming Language – Official Website* [online]. [cit. 5. 4. 2013]. Accessed on: <`http://python.org/`>.

[16] The SciPy community. *SciPy* [online]. [cit. 10. 5. 2013]. Optimization part `http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html`. Accessed on: <`http://docs.scipy.org/doc/scipy/reference/`>.

[17] Wikipedia contributors. *MIT license* [online]. [cit. 10. 5. 2013]. Accessed on: <`http://en.wikipedia.org/wiki/MIT_license`>.

[18] Wikipedia contributors. *Berkeley Software Distribution* [online]. [cit. 10. 5. 2013]. Accessed on: <`http://en.wikipedia.org/wiki/Berkeley_Software_Distribution`>.

[19] Wikipedia contributors. *Copyright* [online]. [cit. 10. 5. 2013]. Accessed on: <`http://en.wikipedia.org/wiki/Copyright`>.

[20] Wikipedia contributors. *Common Public License* [online]. [cit. 10. 5. 2013]. Accessed on: <`http://en.wikipedia.org/wiki/Common_Public_License`>.

[21] Wikipedia contributors. *GNU General Public License* [online]. [cit. 10. 5. 2013]. Accessed on: <`http://en.wikipedia.org/wiki/Gpl`>.

[22] Wikipedia contributors. *HVAC* [online]. [cit. 10. 5. 2013]. Accessed on: <`http://en.wikipedia.org/wiki/HVAC`>.

[23] XU, E. *Pyipopt - An IPOPT connector to python* [online]. [cit. 15. 5. 2013]. Accessed on: <`http://code.google.com/p/pyipopt/`>.

[24] ŠULC, J. Ventilation Control Of the Blanka Tunnel. Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2012.

Appendix A

Implementation

This chapter presents the commented codes using the Python libraries named above. We begin with the quadratic programming and then we show the codes for solving the non-linear programming. The comments are stated in the listings. The version of Python [15], which was used for this project, is 2.7 and this fact should be respected. Because it is possible, that in other version the results could differ from mine, presented in this text. The reason, why we are using the 2.7 version, is simple - the development of some software, we were testing, stopped e.g. 5 years ago. The version 3.0 of Python is 2 years old and some of the Python libraries do not support this new version.

A.1 Quadratic programming

Bellow are the codes, solving MPC for 24-hour horizon, using CVXOPT and OpenOpt as well. The codes for 64 and 128 hours horizon differ in small details. We obtained MAT-file with matrices and vectors, which are the inputs of MATLAB function *quadprog* (see 4.2.2.1), which we used for solving this optimization problem.

A.1.1 CVXOPT

```
from cvxopt import solvers, matrix
# matfile is Python library
# provided by authors of CVXOPT
from matfile import read
import time

# to detect the CPU time we have to define
# the intial time of this process
t0 = time.clock()

# reading the matrices, vectors,
# that are the input for MATLAB quadprog
model = read('qp_24.mat')
Q = model['Q']
p = model['c']
x0 = model['x0']
G = model['A']
h = model['b']
A = model ['Aeq']
b = model ['beq']
lb = model ['lb']
ub = model ['ub']

# qp solver CVXOPT for Quadratic programming
sol=solvers.qp(Q, p, G, h, A, b,x0)

#printing the found optimization variables
print(sol['x'])
```

Listing A.1: Code is solving quadratic programming using CVXOPT libraries.

We can see under this text a part of the output of this code without founded optimization variables[1].

```
     pcost       dcost       gap    pres   dres
 0:  9.9043e+02 -2.7489e+04  3e+04  6e-02  6e-09
 1:  1.3526e+02 -8.7258e+03  1e+04  2e-02  3e-09
...
10:  1.7178e+03  1.7178e+03  1e-02  7e-10  4e-09
11:  1.7178e+03  1.7178e+03  4e-04  8e-12  4e-09
Optimal solution found.
```

[1]The reason, why we do not present the founded variables, is simple - the length of variables vector is 636 for 24-hour horizon.

A.1.2 OpenOpt

```
from numpy import diag, matrix, inf
from openopt import QP
import scipy.io as sio
import time

# to detect the CPU time
t0 = time.clock()

# reading matrices, vectors,
# that are input for MATLAB function quadprog
model = sio.loadmat('qp_24.mat')
Q = model['Q']
c = model['c']
x0 = model['x0']
G = model['A']
h = model['b']
U = model ['Aeq']
v = model ['beq']
lbm = model ['lb']
ubm = model ['ub']

'''
Testing the OpenOpt using the qp function from CVXOPT
pp = QP(H=Q, f=fm, A=G, b=h,  Aeq=U, beq=v, lb=lbm, ub=ubm)
w = pp._solve('cvxopt_qp', iprint = 0)
f_opt, x_opt2 = w.ff, w.xf
'''

# OpenOpt with its qlcp solver
p = QP(H=Q, f=c, A=G, b=h,  Aeq=U, beq=v, lb=lbm, ub=ubm)
r = p._solve('qlcp', iprint = True)
f_opt, x_opt1 = r.ff, r.xf

print x_opt1

print time.clock() - t0, "seconds process time"
```

Listing A.2: Code is solving quadratic programming using OpenOpt libraries.

The output:

```
------------------------- OpenOpt 0.45 -------------------------
solver: qlcp   problem: unnamed    type: QP
 iter   objFunVal   log10(maxResidual)
    0  0.000e+00              1.36
    1  1.718e+03            -13.83
istop: 1000
Solver:   Time Elapsed = 2.25  CPU Time Elapsed = 3.66
objFunValue: 1717.8208 (feasible, MaxResidual = 1.49073e-14)
```

A.2 Non-linear programming (NLP)

In this section the codes for the ventilation control of the Blanka tunnel in first level protection mode will be shown. For the other modes are the codes very similar. Written scripts for NLP, using different libraries, have some common parts, which will be discussed in this part. We obtained MAT-file with matrices and vectors and two M-files 'criteria.m' and 'nonlinearConstraints.m', which are the entries of the function *fmincon* (see 4.2.1.1).

The inputs have to be read for every library:

```
import scipy.io as sio

# The M-file 'criteria.m' present the objective function
# which is written for MATLAB fmincon
fo = file("criteria.m", 'r')

# nonlinear constraints
nc = file("nonlinearConstraints.m", 'r')

# read the MAT-file with matrices and vectors
model = sio.loadmat('tun1.mat')
x0 = model['x0']
A = model['A']
b = model['B']
Ainq = model ['Ainq']
binq = model ['Binq']
lb = model ['lb']
ub = model ['ub']
```

The M-files are having the MATLAB syntax, but we required the Python syntax, so we created functions for rewriting the optimization problem using regular expression operations:

```
# input string - output string
# Replacing the character '^' into Python characters '**'
# for the power and replace e-0 to e-

def geteq(fo):
    foo = list(fo)
    # we get the string of the m file
    foo = "".join(foo)
    foo = re.sub("e-0", "e-", foo)
    foo = re.sub("\^", "**", foo)
    return foo

# input string - output string
# get from the text decimal number
# and this number decrease by one
def minus1string(u):
    d = Decimal(u)
    d = d-1
    u = str(d)
    return u
```

```
# input string - output string
# change the equation from MATLAB into python syntax, x(1) into x[0]
# because MATLAB start the indexing from 1 and Python from 0.
def changeequ(text):
    A = list(text)
    for i in range(len(A)):
        if A[i] == '(' and A[i-1]=='x':
            A[i]='['
            if A[i+2] ==')':
                b = A[i+1]
                b =minus1string(b)
                A[i+2]=']'
                A[i+1] = b[0]
            else:
                b = A[i+1] +A[i+2]
                b =minus1string(b)
                A[i+1] = b[0]
                A[i+2] = b[1]
                A[i+3]=']'
    A = "".join(A)
    return A
```

And these functions are called by the following code. However the function *geteq* is not called for APMonitor, because its modelling language uses for the power the character ‘ ^ ’ in comparison to ‘ ** ’ in Python language.

```
# fo represents the M-file for the objective function
foo = geteq(fo)
out1 = re.findall("f=.*;", foo)
out1 = "".join(out1[0])

# crop the first 'f=' and last char ';'
out1 = out1[2:-1]
# modification from MATLAB sytax into Python syntax
objequ = changeequ(out1)

# nc represents the m file for non-linear constraints
nco = geteq(nc)
out2 = re.findall("c=.*;", nco)
out2 = "".join(out2[0])

# trimming the first part 'c=[' and last char ';'
out2 = out2[3:-1]

# modification from MATLAB sytax into Python syntax
nonlincon = changeequ(out2)
```

OpenOpt and SciPy used eval function for calling the strings as a function:

```
def f(x):
    return eval(objequ)

def c(x):
    return eval(nonlincon)
```

After we have described the common parts of code, we concentrate ourselves on each library separately.

A.2.1 OpenOpt

The main difference between OpenOpt and other optimization libraries was the shifted initial conditions, which have to fulfil the equality constraints. We achieved the shift of the initial conditions with the following MATLAB script.

```
x00 = A\B'
save x00 x00
```

```
# This is the main difference in comparison to other software.
# The vector (array) x00 presents the shifted initial condition

x00 = sio.loadmat('x00.mat')['x00']
x0 = x00

# required constraints tolerance, default for NLP is 1e-6
contol = 1e-7
# (default gtol = 1e-6)
gtol = 1e-7

# Optimization problem solved by NLP
p = NLP(f, x0, c=c, Aeq= A,  beq=np.transpose(b), lb=lb, ub=ub,
        gtol=gtol, contol=contol, iprint = 50,maxIter = 10000,
        maxFunEvals = 1e7, name = 'NLP_1')

# The only usable solver was the ralg.
solver = 'ralg'
# Other possible solvers for NLP
#solver = 'algencan'
#solver = 'ipopt'
#solver = 'scipy_slsqp'

# solve the problem
r = p.solve(solver, plot=0)
print r.xf
```

The output for this function is

```
------------------------- OpenOpt 0.45 -------------------------
solver: ralg   problem: NLP_1    type: NLP   goal: minimum
 iter   objFunVal   log10(maxResidual)
    0  -1.382e+02               0.39
   50  7.083e+02               0.51
  100  6.606e+02              -1.11
...
 1200  2.815e+02              -7.09
 1216  2.815e+02              -7.74
istop: 3 (|| X[k] - X[k-1] || < xtol)
Solver:   Time Elapsed = 63.8  CPU Time Elapsed = 63.66
objFunValue: 281.50647 (feasible, MaxResidual = 1.81469e-08)
```

A.3 SciPy

The function *fmin_cobyla* is able to order only inequality constraints in the form 'greater than or equal to zero'.

$$c(x) \geq 0, \tag{A.1}$$

which is the opposite to the constraints of the function *fmincon*[2]. Therefore we have to create an additional script for this type of non-linear constraints:

```
nonlincon = changeequ(out2)
# negation of MATLAB non-linear constraints
nonlincon = "-(" +nonlincon+ ")"
```

Due to the function *fmin_cobyla* has no matrix inputs, we use all matrices and vectors to define the constraints as functions. For this purpose the following lines of the code were written :

```
def constraints(A,b,Aeq,beq,lb,ub):
    W = [];

    # inequalities
    for k in range(len(A)):
        W.append(equ(A[k],b[k],1));

    # equalities
    for l in range(len(Aeq)):
        W.append(equ(Aeq[l], beq[l],1));
        W.append(equ(Aeq[l], beq[l],-1));

    # constraints required to contain
    # lower and upper bounds as well
    for m in range(len(lb)):
        W.append(bounds(lb[m],1,m))
        W.append(bounds(ub[m],-1,m))

    # add nonlinear constraints
    W.append(nonlin_con)
    return W
```

```
# function for getting the equations in
# form coef0 * x[0] + coef1 * x[1] + ..
# which describe the constraints
def equ(v,o,sign):
    def eqq(x):
        e = 0;
        for i in range(len(v)):
            e = e + v[i]*x[i];
        e = -e + o;
        e = sign*e;
        return e
    return eqq
```

[2]MATLAB *fmincon* has the inequalites in the form 'less than or equal to b' (see 4.2.1.1).

```
# bound presents a vector of lower or upper bounds
# typeb tell us if it is lower or upper bound
def bounds(bound,typeb,position):
    # typeb is number equal to 1 or -1, because the constraints
    # should be in form f(x)>=0
    # and f(x) >= lower bounds, so 1 is for lower bounds
    # for upper bounds the typeb is value of -1
    def eqq(x):
        e = 0
        e = typeb*(x[position]-bound)
        return e
    return eqq
```

And after formulating all these functions we can call the optimization solver:

```
sol = opt.fmin_cobyla(objective, x0,constraints(Ainq,Binq,A,
                        np.transpose(B), np.transpose(lb),
                        np.transpose(ub)))
print(sol)
```

The outcome of this script will be:

```
Return from subroutine COBYLA because the MAXFUN limit has been reached.

NFVALS = 1000   F = 1.769076E+02    MAXCV = 8.156324E-01
X = 4.004098E+00   3.492428E+00   3.494510E+00  -2.814438E+00   6.987696E+00
    3.131990E+00   8.399274E+00   6.814438E+00   4.688918E+00  -2.071884E-01
    3.067661E+00   2.894468E+00   2.951190E+00   2.230097E+00   2.419339E+00
    3.205326E+00   2.307246E+00   3.277442E+00   3.590347E+00   4.112666E+00
    3.445041E-01   5.402998E+00   4.081875E-01   4.266271E+00   1.380575E+00
    4.029696E+00   8.606481E-01   3.938769E+00   3.506115E+00   5.158560E+00
    5.461499E+00   3.299591E+00   1.845160E+00   3.228737E+00   5.666814E-03
   -8.144377E-01  -8.144377E-01  -8.144377E-01  -4.080345E-02  -9.185041E-02
    2.685019E+00   1.843853E+00   3.841599E+00  -8.144377E-01   3.783045E+00
   -4.320360E-01  -8.120521E-01  -2.735501E-01  -8.144377E-01  -8.144377E-01
    1.327228E+00  -7.432064E-01   2.216051E-01  -8.144377E-01  -1.025439E-01
    3.034339E-01   4.741008E-01   6.720970E-01   1.252761E-01  -8.144377E-01
   -8.144377E-01   4.765310E-02   3.944932E+00  -6.174588E-01   1.645533E+01
    2.179169E+00   6.893543E-01   8.144377E-01  -8.144377E-01   4.173955E-01
    1.514751E+00   2.342190E-01   3.901419E-01   5.250312E-01
```

A.4 APMonitor

This software uses its own modelling language, therefore we have to first define the optimization problem into the APM-file using this language. We do not rewrite the problem manually, but we prepared a Python code for this purpose, which contains some above named parts of codes - like functions *minus1string*, *changeequ* and the parts of codes for reading the files with *mat* and *m* extensions.

In comparison to the other libraries, APMonitor requires to have its own APM-file with the defined optimization problem, therefore we formulate following functions:

```
# The inputs are values of coefficients, k presents the position of x
# and z is a string. The return of this function is the string z.
def coef(coe,k,z):
# if the coefficient is positive
    if coe > 0:
        # if the coefficient is equal one
        if coe == 1:
            z += str("x[%i] + " %(k))
        else:
    # checking if the coef. is and an integer or a float
            if (float(coe)-int(coe))==0:
                z += str("%i*x[%i] + " %(coe,k))
            else:
                z += str("%f*x[%i] + " %(coe,k))

    # if the value of the coef. is negative
    # or equal to zero - but this function
    # will not be called if the coe is equal to 0
    else:
                # if we are not at the beginning of a row
                # we have to delete last two characters '+ '
        if z.endswith("\t")!= True:
            z = z[:-2]
        # coefficient is equal to minus one
        if coe == -1:
            z += str("- x[%i] + " %(k))
        else:
   # checking if the coefficient is an integer or a float
            if (float(coe)-int(coe))==0:
                z += str("%i*x[%i] + " %(coe,k))
            else:
                z += str("%f*x[%i] + " %(coe,k))
    return z
```

```
# inputs: matrix M and vector n, e specifies if this case is
# equality or inequality and fo represents the APM-file
def intoapmcon(M,n,e,fo):
    z =''
    for l in range(len(M)):
    # in APM-file needs to be equation and
    # inequations offset by 2 tabulators
        z +=str("\t\t")
        for k in range(len(M[0])):
            if M[l][k] != 0:
            #call the coef function
                z = coef(M[l][k],k,z)

        # last 2 characters '+ ' delete
        z = z[:-2]
    #
        if z != '':
        # if equality
            if e== True:
                if(float(n[l])-int(n[l])==0):
                    z += str("= %i\r\n" %(n[l]))
                else:
                    z += str("= %f\r\n" %(n[l]))
        # if inequality
            else:
                if(float(n[l])-int(n[l])==0):
                    z += str("<= %i\r\n" %(n[l]))
                else:
                    z += str("<= %f\r\n" %(n[l]))
        # write the string z intro APM-file
            fo.write(z)
        # clear the z string
        z = ''
```

After we defined all these functions, we call them to create the APM-file:

```
# opening the APM-file and 'w' indicates for 'write'
fo = file("tun1.apm", 'w')
fo.write('!define the non-linear problem \r\n')
fo.write('Tunnel Preventilation\r\n')

# variables are defined as x[2] = x_02 , >= lb[2], <= ub[2]
fo.write('\tVariables \r\n')
for i in range(len(lb)):
    fo.write("\t \t x[%i]=%d , >=%s , <=%s \r\n" % (i, x0[i],
                        lb[0][i], ub[0][i]))
fo.write('\tEnd Variables\r\n\r\n')

fo.write('\tEquations\r\n')

# objective function
fo.write("\t\tminimize " + objequ + "\r\n")
fo.write("\r\n")

# inequalities
intoapmcon(Ainq,binq,False)
fo.write('\r\n')
fo.write('\t\t'+nonlincon+'<= 0\r\n')

# equalities
intoapmcon(A,b,True)

fo.write("\t End Equations\r\n")
fo.write("End Model\r\n")

# closing the APM-file
fo.close()
```

This code generates the following APM-file (only parts are shown):

```
!define the quadratic problem
Tunnel Preventilation
        Variables
                x[0]=3  , >=0.1  , <=8.0
                x[1]=2  , >=0.1  , <=8.0
                x[2]=2  , >=0.1  , <=8.0
                x[3]=-1  , >=-2.0  , <=6.0
                ...
                x[70]=0  , >=0.0  , <=324.0
                x[71]=0  , >=0.0  , <=144.0
                x[72]=0  , >=0.0  , <=300.0
                x[73]=0  , >=0.0  , <=152.0
        End Variables
        Equations
                minimize 30*x[34]+30*x[35]+30*x[36]+1.894e-05*x[62]^3+4.6341e
                    -05*x[62]^2+0.00025936*x[62]+30*x[37]+1*x[63]+30*x[38]+30*
                    x[39]+30*x[40]+30*x[41]+30*x[42]+30*x[43]+30*x[44]+9.6337e
                    -06*x[64]^3+5.1554e-05*x[64]^2+0.00059053*x[64]+0.6875*x
                    [65]+30*x[45]+30*x[46]+11*x[47]+30*x[48]+30*x[49]+1.725e
                    -05*x[68]^3+0.00038627*x[68]^2+0.0009038*x[68]+0.94937*x
                    [69]+30*x[50]+30*x[51]+30*x[52]+30*x[53]+30*x[54]+45*x
                    [55]+30*x[56]+30*x[57]+1.894e-05*x[70]^3+4.6341e-05*x
                    [70]^2+0.00025936*x[70]+0.9375*x[71]+30*x[58]+30*x
                    [59]+1.6389e-05*x[72]^3-2.0539e-05*x[72]^2+0.00025371*x
                    [72]+0.86582*x[73]+30*x[60]+30*x[61]

                (336.6768 + 526.9625 + 269.354 + 663.1107 + 292.9413+
                    (1703.3612 + 754.4265 + 731.0216 + 1200.5601 +  (846.2185
                    + 1535.7976 + 246.8501 + 168.2056 + 554.2413 + 1112.3964 +
                     700.1472 + (851.1074 + 1839.1069 + 2351.2275 + 442.9933 +
                     67.7862 + 183.8235 + 237.3012 + 319.865)*x[1]*94.6/(x
                    [0]*83.7) + 47.3757 + 46.9582)*x[8]*64.5/(x[6]*86)+
                    25.7245)*x[11]*94.4/(x[10]*94.7))*1000/(3600*x[12]*93.6)
                    -10 <= 0

                83.700000*x[0]  -94.600000*x[1] - x[62] = 0
                94.600000*x[1]  -94.600000*x[2] + x[63] = 0
                94.600000*x[2]  -70.200000*x[3]  -75.700000*x[4] = 0
                75.700000*x[4] + 62*x[5]  -86*x[6] = 0

                ...

                -0.968563*x[26]  -8.376621*x[27]  -1.451443*x[28]  -41.504485*x
                    [29]  -1.309420*x[30]  -1.851736*x[31]  -2.947873*x[32] +
                    15.659847*x[56] + 11.403578*x[57] + 17.625244*x[58] +
                    17.625244*x[59] + 10.568120*x[60] = -277.821076
                -2.947873*x[32] + 10.846014*x[33]  -17.374082*x[61] = 29.566143
         End Equations
End Model
```

This generated APM-file we can send to the APMonitor server via Python code mentioned in the listing 3.4.

And the outcome is following:

```
--------- APM Model Size ------------
Each time step contains
  Objects      :  0
  Constants    :  0
  Variables    :  75
  Intermediates:  0
  Connections  :  0
  Equations    :  36
  Residuals    :  36

fippr_files: rto.t0 does not exist
Number of state variables:    75
Number of total equations: -  35
Number of slack variables: -  1
---------------------------------------
Degrees of freedom       :    39
Number of bound variables: -  2

----------------------------------------------
Steady State Optimization with APOPT Solver
----------------------------------------------
File apopt.opt does not exist.

Iter    Objective  Convergence
   0  3.32060E+02  1.09365E+00
   1  4.32628E+02  3.49581E-01
   2  2.03171E+02  6.12844E-02
   3  1.92393E+02  1.52030E-02
   4  1.79781E+02  1.35017E-03
   5  1.77550E+02  2.27988E-03
   6  1.76562E+02  2.14067E-03
   7  1.76543E+02  4.26098E-05
   8  1.76542E+02  2.86093E-06
   9  1.76542E+02  7.66997E-08

Iter    Objective  Convergence
  10  1.76542E+02  1.43832E-08
Successful solution

---------------------------------------------------
Solver         :  APOPT (v1.0)
Solution time  :  0.08499999999999999 sec
Objective      :  176.54182060990627
Successful solution
---------------------------------------------------
```