Investigating Optimization Strategies for Quadratic Programming Components of a Data Analysis Framework

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics

presented by
Alena Kopaničáková

under the supervision of
Prof. Illia Horenko
co-supervised by
Dr. Olga Kaiser

June 2015

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Alena Kopaničáková
Lugano, 14th of June 2015

Abstract

The aim of this thesis is to investigate the optimization strategies of the quadratic programming components of a Finite Element (FEM) Time Series Analysis Framework. The main concept behind FEM approach in spatio-temporal settings is presented. Algorithmic features of the optimization are described with an emphasis on the software implementations and practical usage. In order to compare efficiency of eight different optimizers, the benchmark is created. All tests are performed using AMPL modeling language. From obtained results, the numerical properties of the problem are explored and the most suitable category of solvers is discovered.

Acknowledgements

I would like to express my appreciation to my advisor Dr. Olga Kaiser for the advices she gave me over the year. Especially, I would like to thank her for continuous patience while explaining the concepts behind the FEM methodology.

I am grateful to Professor Illia Horenko for providing me the opportunity of carrying out this work.

I would also like to thank Professor Olaf Schenk and Dr. Drosos Kourounis for their valuable advices, which helped me to improve this work.

Contents

Figures

Tables

Chapter 1

Introduction

In the field of time series analysis, a commonly occurred problem is the analysis of the persistent processes. For instance, the processes where the change of the underlying model parameters happens much slower than the change of the system variables themselves. Such applications are arising, for example, in climatology or finance [2; 3] .
The problem is addressed by many approaches in the literature, such as Hidden Markov Models [4] and Local Kernel Smoothing Methods [5]. An alternative non-stationary time series analysis method was introduced by I. Horenko, where non-stationarity is resolved by the Finite Element Time Series Methodology (FEM) [6].
Current implementations of the FEM framework are carried out under iterative procedure, which requires solution of linearly constrained quadratic programming (QP) problem on one stage. However QP problem tends to be computationally expensive, the approach is lacking efficiency.
In order to increase the performance of the FEM framework, the time requirements of the resulting QP problem needs to be reduced. As for any optimization problem, it is a good idea to compare several of the available algorithms since the "best" algorithm strongly depends on the problem at hand. Through this work, different optimization strategies are investigated for the quadratic programming components of the FEM framework.
Presented thesis is organized in five chapters. A derivation of the FEM-methodology extended towards spatio-temporal settings is given in Chapter 2. In Chapter 3, the algorithmic features of the optimization are described and the main emphasis is put on the practical usage as well as the available software implementations. Along with the optimizers chosen for the comparison, tools and techniques, which were applied while benchmarking are described in Chapter 4. Chapter 5 presents and analyzes results obtained by benchmark.

Chapter 2

Finite Element Time Series Analysis Methodology

This chapter describes the non-stationary time series analysis approach, which aims at finding the most appropriate model for observed time series [7]. The method is based on the finite element (FEM) technique, which is usually used for solving partial differential equations [8]. The FEM approximates the non-stationary, non-homogenous behavior by local stationary models and a non-stationary, non-homogenous switching process [6]. For the purpose of this thesis, we shall consider spatio-temporal settings of the FEM time series analysis framework, which can be used e.g. in meteorological applications.

2.1 Average Model Distance Functional

Given the time series $X_{s,t}$, observed at time steps $t = t_1, ..., t_{N_T}$ and at location $s = s_1, ..., s_{N_S}$, the primary objective is to estimate the most descriptive model for the given data. To fit the data to the model, i.e. to find the optimal model parameters, $\Theta(s,t)$ must be parametrized. The standard literature on the topic is based on the techniques such as linear regression [9] or artificial neuronal networks [10]. An alternative approach was introduced by Horenko in [6]. The latter technique applies the Finite Element Methodology to approximate the non-stationarity by a set of N_K locally stationary models, each one parametrized by θ_k for $k = 1, ..., N_K$ and a non-stationary switching process $\Gamma = (\gamma_1(s,t), ..., \gamma_{N_K}(s,t))$.
The FEM interpolates the model distance function $g(X_{s,t}, \Theta(s,t))$ by a linear combination of N_K locally stationary model distance functionals

$$g(X(s,t), \Theta(s,t)) \approx \sum_{k=1}^{N_K} \gamma_k(s,t) g(X_{s,t}, \theta_k), \tag{2.1}$$

satisfying the convexity constraints on $\Gamma(s,t)$

$$\sum_{k=1}^{N_K} \gamma_k(s,t) = 1, \ \forall\ t \in [1, N_T], \ \forall\ s \in [1, N_S] \tag{2.2a}$$

$$\gamma_k(s,t) \geq 0, \ \forall\ t \in [1, N_T], \ \forall\ s \in [1, N_S], \ \forall\ k \in [1, N_K]. \tag{2.2b}$$

To find the optimal parameters denoted by $\Gamma^*(s,t)$ and $\Theta^*(s,t)$, the average model distance functional

$$L(\Theta,\Gamma(s,t)) = \sum_{i=1}^{N_S}\sum_{j=1}^{N_T}\sum_{k=1}^{N_K}\gamma_k(s_j,t_j)g(X_{s_i,t_j},\theta_k) \rightarrow \min_{\Gamma(s,t),\Theta}, \tag{2.3}$$

subject to constraints (2.2) needs to be minimized.

2.2 Regularization

The problem (2.3) is ill-posed in sense of Hadamarad [11]. Ill-posedness is tackled by ensuring that the change of underlying model parameters happens more slowly than the change of the system variables. Assuming the local stationarity of the $\Theta(s,t)$, the regularization is done by imposing additional assumptions on the original problem, or by modifying the problem and writing the Tykhonov-regularized form of the functional [12]. Let's assume the temporal persistency constraints $C(N_T)$

$$||\gamma_k(s,t)||_{R(t_1,t_{N_T})} \leq C(N_T),\ s = s_1,...,s_{N_S} \text{ and } k = 1,...,K, \tag{2.4a}$$

which bound maximal number of transitions between clusters for each location station. Considering the spatio-temporal settings of FEM framework, regularization is extended by assuming persistent behavior in space. This assumption is especially reasonable in climate research. For example, spatially persistent tropical cyclone might be responsible for numerous thunderstorms over affected region [13]. The spatial persistency is included by adding supplementary constraints on $\Gamma(s,t)$

$$||\gamma_k(s,t)||_{R(s_1,s_{N_S})} \leq C(N_S),\ t = t_1,...,t_{N_T} \text{ and } k = 1,...,K. \tag{2.4b}$$

The switching process $\Gamma(s,t)$ can be in the space of weakly differentiable functions $R([a,b]) = H^1([a,b])$, or in the space of functions with bounded variation $R([a,b]) = BV([a,b])$.
For the purpose of this thesis, we will focus on the FEM methodology with H^1 regularization. The constraints from (2.4a), (2.4b) can be incorporated by using Lagrange multipliers ϵ_1^2 and ϵ_2^2, thus obtaining the following modified average distance functional in regularized form

$$\begin{aligned} L(\Theta,\Gamma(s,t),\epsilon_1^2,\epsilon_2^2) = \sum_{i=1}^{N_S}\sum_{j=1}^{N_T}\sum_{k=1}^{N_K}\gamma_k(s_j,t_j)g(X_{s_i,t_j},\theta_k) + \epsilon_1^2\sum_{k=1}^{K}||\gamma_k(s,t)||_{H^1_{(t_1,t_{N_T})}} + \\ +\epsilon_2^2\sum_{k=1}^{K}||\gamma_k(s,t)||_{H^1_{(s_1,s_{N_S})}} \rightarrow \min_{\Theta,\Gamma(s,t)}. \end{aligned} \tag{2.5}$$

2.3 Time Discretization

For solving the inverse problem (2.5) FEM approach approximates Γ by linear hat functions. For the sake of comprehension, we neglect the space variable s (thus supposing that $\epsilon_2^2 = 0$) while discretizing in time. We define M as the number of discretization points. This parameter is bounded from above by N_T, since the observations are usually given only in the discretized form. In this thesis, M is considered to be equal to N_T. Decreasing parameter M makes the

algorithm computationally less expensive, but avoids the detection of smaller clusters.
Assuming $\{t_1 = 1, t_2, ..., t_{M-1}, t_M = N_T\}$ to be a finite set of the time discretization interval $[1, N_T]$, it is possible to define basis functions $\{v_1(t), v_2(t), ..., v_M(t)\}$ over the local support on $[1, N_T]$ in H^1. Referring to M triangular functions, i.e. hat functions, we assume that $\gamma_k \in H^1(1, N_T)$. Then, it can be stated that

$$\gamma_k(t) = \tilde{\gamma}_k(t) + error, \tag{2.6}$$

where the error is the result of noise and disturbance due to discretization and $\tilde{\gamma}_k^{(m)}$ is represented as

$$\tilde{\gamma}_k^{(m)} = \int_{t_1}^{N_T} \gamma_k(t) v_m(t) dt. \tag{2.7a}$$

The v_m takes values as

$$v_m(t) = \begin{pmatrix} t_2 - t & m = 1, t \in [t_1, t_2] \\ t - t_{m-1} & 2 \leq m \leq M-1, t \in [t_{m-1}, t_m] \\ t_{m+1} - t & 2 \leq m \leq M-1, t \in [t_m, t_{m+1}] \\ t - t_{m-1} & m = M, t \in [t_{M-1}, t_M] \end{pmatrix}. \tag{2.8}$$

The vector of discretized model distances $\tilde{\gamma_k}$ has the following form

$$\tilde{\gamma}_k = (\tilde{\gamma}_1^{(1)} \dots \tilde{\gamma}_k^{(M)})^\dagger. \tag{2.9}$$

Substituting (2.6) in (2.5), the following form of functional is obtained

$$L(\Theta, \Gamma(t), \epsilon_1^2) = \tilde{L}(\Theta, \Gamma(t), \epsilon_1^2) + error \rightarrow \min_{\tilde{\gamma}_k, \Theta}, \tag{2.10}$$

where $\tilde{L}$ represents a finite-dimensional variant of the original functional as demonstrated in (2.5)

$$\tilde{L}(\Theta, \Gamma(t), \epsilon_1^2) = \sum_{k=1}^{N_K} \int_{t_1}^{N_T} [\tilde{\gamma}_k(t) g(X_t, \theta_k) + \epsilon_1^2 (\partial_t \tilde{\gamma_k}(t))^2] dt. \tag{2.11}$$

After a few transformations and employing the locality of the finite elements, $\tilde{L}(\Theta, \Gamma(t), \epsilon_1^2)$ can be expressed in the following form

$$\begin{aligned} \tilde{L}(\Theta, \Gamma(t), \epsilon_1^2) = \sum_{k=1}^{N_K} [\tilde{\gamma}_k^{(1)} \int_{t1}^{t2} \partial_t v_1(t) g(X_t, \theta_k) dt + \sum_{m=2}^{M-1} \tilde{\gamma}_k^{(m)} \int_{t_{m-1}}^{t_{m+1}} \partial_t v_m(t) g(X_t, \theta_k) dt + \\ + \tilde{\gamma}_k^{(M)} \int_{t_{M-1}}^{t_M} \partial_t v_M(t) g(X_t, \theta_k) dt + \epsilon_1^2 \sum_{k=1}^{M-1} (\tilde{\gamma}_k^{(m)})^2 \int_{t_m}^{t_{m+1}} (\partial_t v_m(t))^2 dt - \\ - 2\tilde{\gamma}_k^{(m)} \tilde{\gamma}_k^{(m+1)} \int_{t_m}^{t_{m+1}} \partial_t v_m(t) \partial_t v_{m+1}(t) dt + (\tilde{\gamma}_k^{(m+1)})^2 \int_{t_m}^{t_{m+1}} (\partial_t v_{m+1}(t))^2 dt)]. \end{aligned} \tag{2.12}$$

Vector $a(\theta_k)$ symbolizes the discretized model distances

$$a(\theta_k) = \left(\int_{t1}^{t2} \partial_t v_1(t) g(X_t, \theta_k) dt, ..., \int_{t_{M-1}}^{M} \partial_t v_M(t) g(X_t, \theta_k) dt \right)^\dagger. \tag{2.13}$$

Contributed stiffness matrix of finite element set results in H_S

$$H_S = \begin{pmatrix} \int_{t1}^{t2} \partial_t v_1^2(t)dt & \int_{t1}^{t2} \partial_t v_1(t)\partial_t v_2(t)dt & 0 & \cdots & 0 \\ \int_{t1}^{t2} \partial_t v_1(t)\partial_t v_2(t)dt & \int_{t2}^{t3} \partial_t v_2^2(t)dt & \int_{t2}^{t3} \partial_t v_2(t)\partial_t v_3(t)dt & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & \int_{t_{M-1}}^{t_M} \partial_t v_M^2(t)dt \end{pmatrix}. \tag{2.14}$$

Recalling hat functions, H_S is evaluated as

$$H_S = \begin{pmatrix} 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{pmatrix}. \tag{2.15}$$

With the help of (2.13) and (2.14), (2.12) can be reformulated. Then, the minimization problem (2.3) is converted to the following quadratic programming (QP) problem with respect to the Γ

$$\tilde{L}(\Theta, \Gamma(t), \epsilon_1^2) = \sum_{k=1}^{N_K} [a(\theta_k)^T \bar{\gamma}_k + \epsilon_1^2 \bar{\gamma_k}^T H_S \bar{\gamma_k}] \tag{2.16a}$$

subject to the equality constraints

$$\sum_{k=1}^{N_K} \tilde{\gamma}_k(t) = 1, \ \forall t = 1, ..., N_T \tag{2.16b}$$

and the inequality constraints

$$\tilde{\gamma}_k(t) \geq 0, \ \forall t = 1, ..., N_T, \ k = 1, ..., N_K. \tag{2.16c}$$

2.4 The Spatio-Temporal Setting

Considering spatio-temporal settings of FEM, the temporal dimension is discretized by Finite Element method. After applying a same idea, the spatial dimension is discretized and the following form of the functional is obtained

$$L(\Gamma, \Theta(s,t), \epsilon_1^2, \epsilon_2^2) = \sum_{k=1}^{N_K} (a(\theta_k)^T \bar{\gamma_k} + \epsilon_1^2 \ \bar{\gamma_k}^T H_{1k} \bar{\gamma_k} + \epsilon_2^2 \ \bar{\gamma_k}^T H_{2k} \bar{\gamma_k}). \tag{2.17}$$

The matrixes $\bar{\gamma_k}$, H_{1k} and vector $a(\theta_k)$ in (2.17) originate from derivations in Section 2.3, but they are expanded along all space locations. For example, the matrix H_{1k} is now defined as

$$H_{1k} = \begin{pmatrix} H_{S_1} & & \\ & \ddots & \\ & & H_{S_{N_S}} \end{pmatrix} \in R^{N_T N_S \times N_T N_S}, \tag{2.18}$$

where each block H_{S_i}, $i \in 1,...,N_S$ takes form from (2.15). The matrix H_{2k} describes the spatial pairwise connectivity among all locations. If we consider just the simple geographical distances, e.g. euclidean distances between locations, we might neglect some topographical properties like the presence of mountains. An alternative approach is to consider, for instance the classical correlation [14], where each pair of locations is correlated equally for each time step. Then, the connectivity matrix H_{2k} can be computed as shown by Algorithm 1.

Algorithm 1 Computation of H_{2k}

Require: time series $X_{s,t}$
  for t $= 1 : N_T$ **do**
    $H_{2k}(t : N_T : end, t : N_T : end) = corr(X_{s,t})$
  end for

The matrix H_{1k} (2.18) is symmetric, in order to preserve symmetry in matrix H_{2k}, we focus on symmetric connections only.
Another alternative approach is to deploy cross-correlation [15], where connections between two time series are shifted in time relatively to one another. To examine such an information might be important, because one time series may have a delayed response to the other series, for instance, weather in Lugano can be dependent on weather in Locarno in the preceding days. To exemplify this, let us consider time series $X_{s,t}$ observed at space locations $s = 1,...,3$ and at time steps $t = 1,...,4$. Then, the connectivity matrix H_{2k} takes the following form.

	N_1				N_2				N_3			
N_1	a_{11}	b_{11}			a_{12}	b_{12}			a_{13}	b_{13}		
	b_{11}	a_{11}	b_{11}		c_{12}	a_{12}	b_{12}		c_{13}	a_{13}	b_{13}	
		b_{11}	a_{11}	b_{11}		c_{12}	a_{12}	b_{12}		c_{13}	a_{13}	b_{13}
			b_{11}	a_{11}			c_{12}	a_{12}			c_{13}	a_{13}
N_2	a_{12}	c_{12}			a_{22}	b_{22}			a_{23}	b_{23}		
	b_{12}	a_{12}	c_{12}		b_{22}	a_{22}	b_{22}		c_{23}	a_{23}	b_{23}	
		b_{12}	a_{12}	c_{12}		b_{22}	a_{22}	b_{22}		c_{23}	a_{23}	b_{23}
			b_{12}	a_{12}			b_{22}	a_{22}			c_{23}	a_{23}
N_3	a_{13}	c_{13}			a_{23}	c_{23}			a_{33}	b_{33}		
	b_{13}	a_{13}	c_{13}		b_{23}	a_{23}	c_{23}		b_{33}	a_{33}	b_{33}	
		b_{13}	a_{13}	c_{13}		b_{23}	a_{23}	c_{23}		b_{33}	a_{33}	b_{33}
			c_{13}	a_{13}			b_{23}	a_{23}			b_{33}	a_{33}

Figure 2.1. Example Of H_{2k} Connection Matrix

The coefficients a, b, c result from the cross-correlation function $xcorr$ available on Matlab

$$[c_{ij}, a_{ij}, b_{ij}] = xcorr(X_i, X_j, 1), \tag{2.19}$$

for time series observed at locations N_i and N_j, over the lag range [-1, 0, 1]. Making the lag range wider, the resulting sequence of coefficient contains more elements and the matrix

H_{2k} becomes denser. Cross-correlation is described in terms of "lead" and "lag" connections. Referring to Figure 2.1, the coefficient b defines the relationship, when X_i is shifted one time step forward relative to X_j, thus X_j is said to "lead" X_i, which is the same as saying that X_i "lags" X_j. Similar idea applies for coefficient c. Because of the symmetry, the coefficients b taking subscript ij are the same as coefficients c with subscript ji. Instead, the value of a stands for relationship between two time series at the same time. If N_i and N_j stands for the same location, values of b and c are equal.
After obtaining the matrix H_{2k}, we can sum up the matrices H_{1k} and H_{2k} in (2.17) and set the final matrix H_k. Ordering all H_k matrixes and vectors a_k, $k = 1, ..., N_K$, the matrix H is defined as $H = diag(H_1, \ldots, H_{N_K})$ and vector a as $a(\Theta) = (a(\theta_1), \ldots, a(\theta_{N_K}))$.
Now, it is possible to rewrite (2.17) into a linearly constrained quadratic problem (LQP) with respect to the parameter $\tilde{\Gamma}$ in the following form

$$L(\Theta, \Gamma(t), \epsilon_1^2, \epsilon_2^2) = \bar{\Gamma^T} H \bar{\Gamma} + a^T(\Theta)\bar{\Gamma} \tag{2.20a}$$

$$\text{subject to } Aeq\ \bar{\Gamma} = beq \tag{2.20b}$$

$$\bar{\Gamma} \geq 0. \tag{2.20c}$$

The equality constraints (2.20b) are generalization of the equality constraint (2.2) for all space locations at each time step.

2.5 Numerical Solution of FEM In The Spatial Setting

The parameters Γ and Θ are obtained by minimization (2.20). However, there is no analytical solution of the problem, constrained minimization (2.20) is carried out numerically in alternative order. The main idea of algorithm is outlined in Algorithm 2. The algorithm begins with randomly initialized Γ^0 and alternatively estimates parameters Θ and affiliations Γ. In the first step, minimization of (2.17) with respect to Θ is made, while Γ is fixed. In the second step, by fixing parameters Θ, the linearly constrained quadratic problem needs to be solved in order to estimate new switching processes Γ.
In every double step, the value of functional $L(\Theta, \Gamma, \epsilon_1^2, \epsilon_2^2)$ is decreased until the current reduction is smaller than the predefined tolerance. The algorithm converges just to the local minimum [16], due to non-convexity of (2.1). In order to find a global solution among all local solutions, it is necessary to explore the solution space. To do so, algorithm can be restarted sufficiently enough times, with random initializations of the switching process. Searching for global solution is fully parallelized.

Algorithm 2 FEM subspace algorithm

Require: time series X, number of clusters N_K, regularization factors ϵ_1^2, ϵ_2^2, initial affiliations Γ^0, tolerance τ
Ensure: optimal affiliations Γ^*, optimal parameters Θ^*
m = 0
repeat
(1) Compute Θ^{m+1} for fixed Γ^m

$$\Theta^{m+1} = arg\,\min_\Theta L(\Theta, \Gamma^m, \epsilon_1^2, \epsilon_2^2)$$

(2) Compute Γ^{m+1} for fixed Θ^{m+1}

$$\Gamma^{m+1} = arg\,\min_\Gamma L(\Theta^{m+1}, \Gamma, \epsilon_1^2, \epsilon_2^2), \quad \text{subject to (2.20b) and (2.20b)}$$

until $|L(\Theta^m, \Gamma^m, \epsilon_1^2, \epsilon_2^2) - L(\Theta^{m-1}, \Gamma^{m-1}, \epsilon_1^2, \epsilon_2^2)| > \tau$
m = m + 1

The framework starts with user-defined parameters, such as the number of clusters N_K and the regularization factors ϵ_1^2, ϵ_2^2. The output of the framework strongly depends on the choice of these parameters. The selection of the optimal ones is a difficult task, but is required by our willingness to achieve a model which is highly informative and less computationally expensive at the same time. Optimal choice can be made by using information criterium, e.g. Akaike information criteria (AIC) [17], which maximizes the likelihood and at the same time penalizes the number of parameters.

2.6 Conclusion

In this chapter, the FEM time series clustering approach in the spatio-temporal settings was presented. The approach results in subspace iteration algorithm, where parameters Γ and Θ are estimated interchangeably. Looking for parameters Γ^* is a computationally complex task, due to the linearly constrained quadratic programming problem. An investigation of possible optimization strategies can be found in the next chapters.

Chapter 3

Algorithmic Features Of The Optimization

This chapter presents the theory behind nonlinear constrained optimization. Description of different algorithms and techniques is given, with emphasis on software implementation and practical usage.
Through this chapter, the following notation is used. Iteration is denoted as x_k. The subscript $k = 1, 2, ..$ is used to indicate the function evaluation at an iteration. The gradients are denoted by $g_k = \nabla f(x_k)$, the Hessian by H_k and the Jacobian by $A_k = \nabla c(x_k)$, where $c(x_k)$ represents the constraints.

3.1 Constrained Optimization

Consider a nonlinear optimization problem written in the following form

$$\min_{x \in R^n} \quad f(x) \tag{3.1a}$$
$$\text{subject to } c_i(x) = 0, \quad i \in \xi \tag{3.1b}$$
$$\text{subject to } c_i(x) \geq 0, \quad i \in I, \tag{3.1c}$$

where $f(x)$ is the objective function and ξ, I are the sets of indexes for equality and inequality constraints $c_i(x)$, respectively. The set of points which satisfies the constraints is denoted as the feasible set. The solution of the problem is found by considering the stationary points i.e. the first derivative is equal to zero. For problem (3.1) is the Lagrangian defined as

$$L(x, \lambda) = f(x) - \sum_{i \in \xi \cup I} \lambda_i c_i(x), \tag{3.2}$$

where $i \in \xi \cup I$, λ_i is the Lagrange multiplier for the constraint $c_i(x)$. At any solution point x^*, the vector of Lagrangian multipliers λ^* has to satisfy

$$\nabla_x L(x^*, \lambda^*) = 0 \tag{3.3a}$$
$$c_i(x^*) = 0, \quad \text{for all} \quad i \in \xi \tag{3.3b}$$
$$c_i(x^*) \geq 0, \quad \text{for all} \quad i \in I \tag{3.3c}$$
$$\lambda_i^* \geq 0, \quad \text{for all} \quad i \in I \tag{3.3d}$$
$$\lambda_i^* c_i(x^*) = 0, \quad \text{for all} \quad i \in \xi \cup I. \tag{3.3e}$$

The conditions stated in (3.3) are known as Karush-Kuhn-Tucker (KKT) conditions [18]. Considering just equality constraints in (3.3), the KKT conditions are the Lagrange conditions. Accordingly, the KKT multipliers are Lagrange multipliers. Constraint (3.3e) represents the complementary conditions, which indicate, if the constraint i and $\lambda_i^* = 0$ are active independently or at the same time. The active set $A(x)$ is the set consisting of equality and inequality constraints i for which $c_i(x) = 0$, such that

$$A(x) = \xi \cap \{i \in I | c_i(x) = 0\}. \tag{3.4}$$

The multiplier λ_i expresses the sensitivity of the optimal value $f(x^*)$ to the presence of a given constraint c_i. If λ^* is large, then $f(x^*)$ is sensitive to the position of the i-th constraint. If it is small, the dependence is not so strong. In case λ_i^* equals zero, a small perturbation of c_i will hardly affect the optimal value [19].
An analytic solution of (3.1) can not be found directly. A feasible alternative is provided by iterative methods that solve a sequence of approximate subproblems and generate a sequence of approximate solutions $\{x_k\}$ starting from an initial guess x_0. A simplified algorithmic framework for solving (3.1) is presented by Algorithm 3.

Algorithm 3 Optimization Framework

Require: initial estimate $x_0 \in R^n$, set $k = 0$
  while x_k is not optimal **do**
    repeat
      approximately solve and refine a local model of (3.1) around x_k
    until an x_{k+1} is not found
    $k = k + 1$
  end while

Sometimes, it is computationally easier to solve an alternative problem instead of the original-primal problem. Duality theory shows how to construct such a problem. Considering problem (3.1) with the Lagrangian function (3.2), the dual objective function is defined as

$$q(\lambda) = \inf_{x \in R^n} L(x, \lambda). \tag{3.5}$$

The dual problem to problem (3.1) is then defined as

$$\max_{\lambda \in R^n} q(\lambda) \tag{3.6}$$
$$\text{subject to } \lambda \geq 0. \tag{3.7}$$

The primal-dual relationship is symmetric, thus the dual form of a dual problem corresponds to the original primal problem. The optimal value of the dual problem constitutes a lower bound for the optimal objective value of the primal problem and the optimal Lagrange multipliers of the primal problem are solutions of the dual problem under certain conditions.

3.2 Globalization Strategies

Optimization algorithms obtain the new iterate x_{k+1} by using informations about function f at the current point and previous points. In order to move from one iteration to another, two different strategies can be applied: line-search methods and trust region methods [20]. Both can be used in conjunction with any of the local models and any of the convergency improvements described later.

3.2.1 Line-search Methods

After computing a search direction p_k, the algorithm decides where and how far to move in order to find a solution. Each new iteration can be obtained as

$$x_{k+1} = x_k + \alpha_k p_k, \tag{3.8}$$

where $\alpha_k > 0$ is the step length. Consequently the algorithm depends on the choice of search direction and step lenght. Examples for their choice is shown on Algorithm 4.

Algorithm 4 Line Search Framework

Require: initial estimate $x_0 \in R^n$, set $k = 0$
  while x_k is not optimal **do**
    compute direction p_k
    choose α_k
    update $x_{k+1} = x_k + \alpha_k p_k$
    $k = k + 1$
  end while

Search Direction

The search direction can be obtained by applying different numerical techniques. First-order methods such as Newton's methods, quasi-Newton methods or conjugate gradient methods are suitable for most problems [21]. Sub-gradient and proximal gradient methods are appropriate for problems with non-differentiable objective function [22]. Coordinate descent methods, localization methods, or alternating projection methods do not require any knowledge of derivatives [23].
In the following, we will focus on first-order methods, as the class of methods is only one available in different software packages. These techniques are not bounded to the line-search framework but can be also used in the trust region framework.
The steepest descent direction [20] is the one, which moves along $p_k = -\nabla f_k$ at every step. The Newton's search direction [24] is obtained as

$$p_k = -(\nabla^2 f_k)^{-1} \nabla f_k \tag{3.9a}$$

and can be applied when the Hessian $\nabla^2 f_k$ is positive definite, otherwise $(\nabla^2 f_k)^{-1}$ may not be defined. An alternative to Newton's method are quasi-Newton methods [24], which do not require computation of the Hessian $\nabla^2 f_k$. Instead, an approximation B_k is used. The search direction is then given as

$$p_k = -B_k^{-1} \nabla f_k. \tag{3.9b}$$

An update of B_k is made after every step . Rather than recomputing the approximate Hessians at every iteration from scratch, a simple modification is applied. It combines the objective function with the existing knowledge from the current Hessian. The most popular formulas for updating B_k are the symmetric-rank-one (SR1) formula[24]

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \tag{3.9c}$$

and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula [24]

$$B_{k+1} = B_k - \frac{(B_k s_k s_k^T B_k)}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}, \tag{3.9d}$$

where

$$s_k = x_{k+1} - x_k = \alpha_k p_k \tag{3.9e}$$

$$y_k = \nabla f_{k+1} - \nabla f_k. \tag{3.9f}$$

Two main variants of the quasi-Newton methods for large-scale problems exist: partially separable and limited memory quasi-Newton methods [24]. Limited memory techniques use Hessian approximation, that can be stored compactly by using just a few vectors. Another alternative used to find a search direction is the conjugate gradient method [25]

$$p_k = -\nabla f(x_k) + \beta_k p_{k-1}. \tag{3.9g}$$

The scalar β_k ensures these that p_k and p_{k-1} are conjugate, i.e. satisfying $p_k^T \beta_k p_{k-1} = 0$. As these methods do not require matrix storage, they are very efficient, but their convergence rate strongly depends on the conditioning number of the Hessian. Therefore, different preconditioning techniques, e.g. symmetric successive over relaxation (SSOR) [26] or incomplete Cholesky factorization [27; 28] are often applied.

Step Length

The step-length has a major impact on robustness and efficiency of the algorithms. The choice of α can be performed by different approaches, as long as it satisfies particular conditions. For instance the step length should give sufficient decrease of the objective function

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \ \nabla f_k^T p_k, \text{ for some } \ c_1 \in (0,1) \tag{3.10a}$$

and rule out short steps, thus requiring α_k to satisfy

$$\nabla f(x_k + \alpha_k p_k)^T p_k \leq c_2 \nabla f_k^T p_k, \text{ for some } \ c_2 \in (c_1, 1). \tag{3.10b}$$

The conditions (3.10) are known as Wolfe conditions [29] and they are used in most of the line-search algorithms, especially in combination with the quasi-Newton's methods [30]. The global minimizer of the function

$$\phi(\alpha) = f(x_k + \alpha p_k), \quad \alpha > 0 \tag{3.11}$$

would provide an ideal step length. However, since it is computationally expensive to identify this value, practical strategies tend to perform an inexact line search. In other words, they try a sequence of possible values for α, accepting the one, which satisfies certain conditions, e.g. Wolf conditions [29].
This is done in two phases. First, the bracketing phase finds an interval $[a, b]$ of acceptable step lengths. Second, the selection phase reduces the bracketing interval to the required step length by incorporating informations from earlier steps [31].
All line-search procedures demand an initial estimate α_0. For Newton and Quasi-Newton methods $\alpha_0 = 1$ gives the rapid rate of convergency [32]. For steepest descent and conjugate gradient, the initial guess of α_0 is usually made out of current informations about the problem by applying some heuristics[24].
All optimization techniques aim to achieve the rapid global convergence. For miscellaneous gradient methods in the line-search framework convergence rate differs. Convergence of steepest descent depends on the conditioning number $K = \frac{\lambda_n}{\lambda_1}$, where λ_n and λ_1 represent respectively the biggest and the smallest eigenvalue of the Hessian. If the value of K is too high, convergence degenerates and step p_k is zig-zaging toward the solution. Newton methods [24] converge at quadratic rate. Quasi-Newton methods satisfying the Wolf conditions converge super-linearly [24]. Convergency of the conjugate gradient also depends on K. The upper bound of the convergency rate is proven to be $\sqrt{K}$. Decreasing K can be done by using preconditioners [32].

3.2.2 Trust Region Methods

The trust region (TR) methods [33] can be viewed as dual methods to the line-search approaches. TR methods define a region around the current iteration, starting from the assumption that the approximated model is an adequate representation of the objective function. The direction steps are generated by a quadratic model. Let us consider a quadratic model m_k at iteration x_k

$$m_k(p) = f_k + g_p^T + \frac{1}{2} p^T B_k p, \tag{3.12}$$

where B_k is either an approximation of the Hessian or the true Hessian itself. To obtain the direction step, we seek for a solution of

$$\min_{p \in R^n} m_k(p) \quad , s.t. ||p|| \leq \Delta_k, \tag{3.13}$$

where Δ_k is the trust-region radius and the trust region is defined by the Euclidean norm. The solution p_k is the minimizer of m_k in the ball of radius Δ_k. For large problems, the trust region defined by $||.||_\infty$ norm may be a more convenient option [33].
The methodology requires solving a sequence of subproblems (3.13) at each step.
If B_k is positive definite and $||B_k^{-1} g_k|| \leq \Delta_k$, the solution is easy to identify. It is simply the unconstrained minimum of $m_k(p)$, i.e. $p_k = -B_k^{-1} g_k$ [34]. The solution is not obvious in other cases, but it can be usually found without too much computational effort [35].

The size of the considered trust region is critical. In practice, it is chosen accordingly to the performance of the algorithm during previous iterations. If the model is reliable, the trust region radius may be increased and more ambitious step can be taken. A failed step indicates on the other hand that the model is not adequate and the region should be shrinked [33; 34] . A recapitulation of the trust region approach can be found in Algorithm 5.

Algorithm 5 Trust Region Framework

Require: initial estimate $x_0 \in R^n$, choose Δ_0, set $k = 0$
  while x_k is not optimal **do**
    repeat
      approximately solve a local trust-region model with $||p|| \leq \Delta_k$
      if $x_k + p_k$ is sufficiently better than x_k **then**
        accept the step x_{k+1}
        possibly increase Δ_k
      else
        reject the step and decrease the trust-region radius, e.g. $\Delta_{k+1} = \frac{\Delta_k}{2}$
      end if
    until an estimate solution x_{k+1} is not found
    k = k + 1;
  end while

In order to look for the minimizer of m_k along the steepest direction, we can use two main strategies. The dogleg method is suitable when B is positive definite. A modified approach is the two-dimensional subspace minimization which is able to handle also indefinite B [1].
Because of the trust region restriction $||p|| \leq \Delta_k$, the subproblem is guaranteed to have a global solution even when H is not positive definite. To compare line-search and trust-region techniques refer to Figure 3.1.

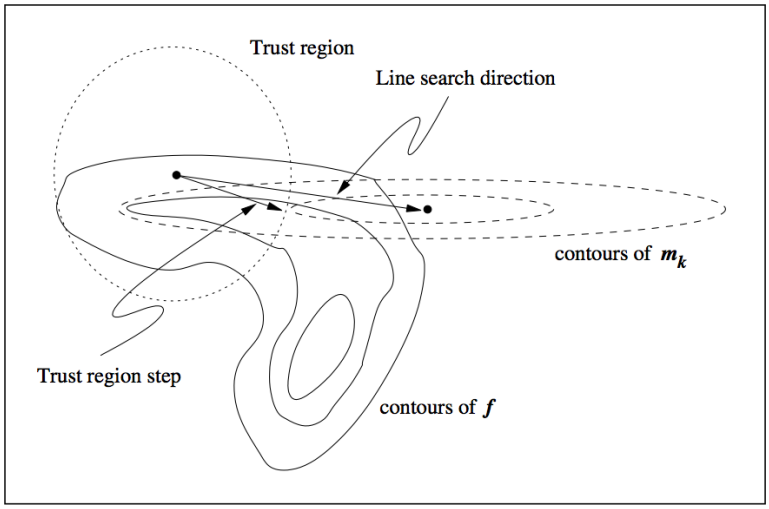

Figure 3.1. Trust Region vs. Line Search Approach [1]

3.3 Convergency Improvements

To ensure convergency from remote starting points, progress of the local method has to be monitored. Monitoring is easy in unconstrained optimization, where the progress can be measured by comparing objective values. In constrained optimization, the constraint violation must be taken into account. The desired target consists in finding the tradeoff between reducing the objective function and satisfying the constraints. Two classes of strategies exist a) merit functions [36] b) filter methods [37].

3.3.1 Merit Functions

Merit functions combine the objective with a measure of constraint violation into a single function. The merit function often takes the form of a ℓ_1 penalty function

$$\phi_1(x;\mu) = f(x) + \mu \sum_{i\in\xi} |c_i(x)| + \mu \sum_{i\in I} [c_i(x)]^-, \tag{3.14}$$

where μ represents a penalty parameter and notation $[c_i(x)]^-$ stands for $max\{0, -c_i(x)\}$. The penalty parameter determines the weight, that is assigned to the constraint satisfaction, relative to the minimization of the objective [38]. Local minimizers of the merit functions correspond to the local minimizers of the original problem, if there is a penalty parameter μ^* such that $\mu > \mu^*$.
Merit functions are often used in the line search framework [39]. The sufficient decrease conditions require that the step length parameter α is small enough to satisfy the inequality

$$\phi_1(x + \alpha\, p;\mu) \leq \phi_1(x;\mu) + \eta\ \alpha\ D(\phi_1(x;\mu); p) \tag{3.15a}$$

for some parameter $\eta \in (0,1)$. Note, that the directional derivative of the merit function $D(\phi_1(x;\mu); p)$ in the direction p has to be used, due to non-smoothness of ϕ_1.
In the trust region framework [39], a quadratic model $q(p)$ is typically used to estimate the value of the merit function ϕ after a step p. Sufficient decrease conditions are stated in terms of a decrease in this model

$$\phi_1(x + p;\mu) \leq \phi_1(x;\mu) - \eta(q(0) - q(p)). \tag{3.15b}$$

3.3.2 Filters

Filters are step acceptance mechanisms, based on multi-object optimization [37]. They minimize the objective function and satisfy the constraints separately. Let us define a measure of infeasibility as

$$h(x) = \sum_{i\in\xi} |c_i(x)| + \sum_{i\in I} |c_i(x)]^-. \tag{3.16}$$

Goals are expressed as

$$\min_x f(x) \quad \text{and} \quad \min_x h(x). \tag{3.17}$$

Filter accepts a step x_k as a new iterate, if the pair $(f(x_k), h(x_k))$ is not dominated ($f_k \geq f_{k-1}$ and $h_k \geq h_{k-1}$) by a previous pair $(f(x_{k-1}), h(x_{k-1}))$ generated by the algorithm. When x_k is

acceptable the pair (f_k, h_k) is added to the filter. Any pairs that are dominated are removed from the filter.
The filters provide convergence only to a feasible limit, because any infinite sequence of iterates must converge to a point, where h(x) = 0. To ensure convergenc to a local minimum, a point, whose pair is very close to the current pair is not accepted. This can be done by modifying the accepting criterion and imposing a sufficient decrease condition [40]. Difference between merit function and filter approach can be seen in Figure 3.2.

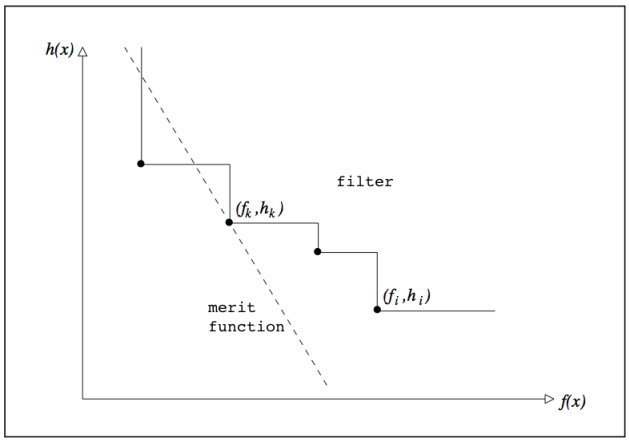

Figure 3.2. Merit Function vs. Filter Approach [1]

3.3.3 The Maratos Effect And Loss Of Convergency

Some algorithms based on merit functions/filters may fail to converge rapidly, because they reject steps, that make good progress toward the solution. The search direction generated by line search algorithms may require small α_k to be acceptable to the filter, which can cause stall and fail. In the trust region settings, the radius may be decreased so much, that trust region subproblem becomes infeasible. This phenomenon is known as the Maratos effect [41].
To avoid the Maratos effect, two techniques are usually used: second-order and non-monotone strategies. For details refer to [42; 43].

3.4 Local Models

3.4.1 Augmented Lagrange Methods

Augmented Lagrange Methods define a function that combines properties both of the Lagrangian function and of the quadratic penalty function. Then, the optimization problem is solved as a sequence of unconstrained problems [44]. Let us consider just equality constraints. The Augmented Lagrange function $L_A(x, \lambda; \mu)$ with penalty parameter μ takes following form

$$L_A(x, \lambda; \mu) = f(x) - \sum_{i \in \xi} \lambda_i c_i(x) + \frac{\mu}{2} \sum_{i \in \xi} c_i^2(x). \tag{3.18}$$

Function (3.18) differs from standard Lagrange representation due to the presence of squared terms and from the quadratic penalty function because of the presence of the sum term involving λ. This approach starts by fixing λ to some estimate of the optimal Lagrange multiplier vector and μ to some positive value. Then, it is possible to find x, that approximately minimizes $L_A(., \lambda; \mu)$. At the new point x, parameters λ and μ are updated and the process can be repeated. Algorithm 6 presents the main idea behind of Augmented Lagrange methods.

Algorithm 6 Augmented Lagrangian Method - Equality Constraints

Require: initial estimate $x_0 \in R^n, \lambda_0, \mu_0$, tolerance $\tau_0 > 0$, set $k = 0$
  while convergence test for x_k is not satisfied **do**
    find an approximate minimizer x_k of $L_A(., \lambda_k; \mu_k)$, starting at x_k and terminating when $||\nabla_x L_A(x_k, \lambda_k; \mu_k)|| \leq \tau_k$
    update λ_k to obtain λ_{k+1}
    choose new penalty parameter $\mu_{k+1} \leq \mu_k$
    $x_{k+1} = x_k$
    select tolerance τ_{k+1}
  end while

Software

Optimization solvers based on Augmented Lagrange Method were popular for many years. Currently, they have however lost on their popularity, because sequential quadratic programming and interior point solvers are likely to outperform them. In the following Table 3.1, we can find a short overview of the available packages.

Name	License		Global Method	Main Interfaces	Release/ Update
	Free	Academic			
ALGENCAN [45]	Yes	Yes	Aug. Lagrangian	Fortran, Matlab	2007
GALAHAD [46]	Yes	Yes	Aug. Lagrangian	Fortran, AMPL	2013
LANCELOT [47]	Yes	Yes	TR	AMPL, SIF	—
MINOS [48]	No	Yes	LS	AMPl, C, Fortran	2013
PENNON [49]	Yes	Yes	LS	AMPL, Matlab	2013

Table 3.1. List Of Augmented Lagrange Solvers

3.4.2 Sequential Linear And Quadratic Programming

Sequential Linear And Quadratic Programming (SLQP) methods construct an approximation of (3.1) and then solve a sequence of these approximations, converging to a stationary point. These are often called an active-set methods [50], as the solution they provide is not only the new iterate, but also the estimation of an active set.
The basic structure of SLQP method involves major and minor iterations. The major iterations

generate a sequence of iterates (x_k, λ_k) that converge to (x^*, λ^*). At each iterate, QP is used to generate a search direction, as the result of another iterative procedure.
SLQP techniques can be divided into four categories: sequential quadratic programming, sequential linear programming, sequential linear/quadratic programming and sequential quadratic/quadratic programming. In this section, we are going to tackle the first three approaches.

SQP - Sequential Quadratic Programming

SQP models approximate problem (3.1) at iterate (x_k, λ_k) using the following QP subprogram

$$\min_p f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla^2_{xx} L_k p \tag{3.19a}$$

$$\text{subject to } A_k p + c_k = 0, \tag{3.19b}$$

where $\nabla^2_{xx} L_k = \nabla^2_{xx} L(x_k, \lambda_k)$. Let us assume that the constraint Jacobian A_k has full row rank and that $\nabla^2_{xx} L_k$ is positive definite. Then, problem (3.19) has a unique solution (p_k, l_k), which satisfies

$$\nabla^2_{xx} L_k p_k + \nabla f_k - A_k^T l_k = 0 \tag{3.20a}$$

$$A_k p_k + c_k = 0. \tag{3.20b}$$

The new iterate (x_{k+1}, λ_{k+1}) is the solution of (3.19). Algorithm 7 summarizes the description given above.

Algorithm 7 Local SQP Algorithm

Require: initial pair x_0, λ_0, set $k = 0$
  while x_k is not optimal **do**
    evaluate $f_k, \nabla f_k, \nabla^2_{xx} L_k, c_k, A_k$
    solve QP to obtain p_k and l_k
    $x_{k+1} = x_k + p_k$
    $\lambda_{k+1} = l_k$
    $k = k + 1$
  end while

SLP - Sequential Linear Programming

SLP constructs a linear approximation of (3.1) and requires also the trust region approach. The approximately model is represented as

$$\min_p f_k + \nabla f_k^T p \tag{3.21a}$$

$$\text{subject to } \; c_i(x_k) + \nabla c_i(x_k)^T p = 0, \quad i \in \xi \tag{3.21b}$$

$$||p||_\infty \leq \Delta_k. \tag{3.21c}$$

The difference between SLP and SQP consists in the fact that the 2. order term in the objective function is omitted and that an ℓ_∞ norm is used to define the trust region.

SLQP - Sequential Linear/Quadratic Programming

The third category of methods combines advantages of SLP and SQP methods by adding an equality-constrained QP to the SLP method. Therefore this approach consists of 2 stages. First, the LP is solved in order to identify an active set A and to obtain a step for the next iteration

$$\min_p f_k + \nabla f_k^T p \tag{3.22a}$$
$$\text{subject to } c_i(x_k) + \nabla c_i(x_k)^T p = 0, \quad i \in \xi \tag{3.22b}$$
$$\text{subject to } c_i(x_k) + \nabla c_i(x_k)^T p \geq 0, \quad i \in I \tag{3.22c}$$
$$||p||_\infty \leq \Delta_k. \tag{3.22d}$$

The second step is the QP phase, where the estimated active set is used to construct an equality constrained QP [51]. Constraints in A_k are treated as equalities, while the other constraints are ignored. The subproblem has the form

$$\min_p f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla^2_{xx} L_k p \tag{3.23a}$$
$$\text{subject to } c_i(x_k) + \nabla c_i(x_k)^T p = 0, \quad i \in \xi \cap A_k \tag{3.23b}$$
$$\text{subject to } c_i(x_k) + \nabla c_i(x_k)^T p = 0, \quad i \in I \cap A_k. \tag{3.23c}$$

The solution of (3.23) can be found by applying the KKT conditions. Then, linear solvers can be used to solve this problem. SLQP approach works well for large-scale versions of LP and EQP subproblems [51].

Software

Actual SLQP solvers are reliable and efficient in solving a sequence of LP and/or QP subproblems. These methods require few evaluations of the functions, in comparison with Augmented Lagrangian methods, and can be more robust on badly scaled problems than the nonlinear Interior-point methods. Software implementations can be found in Table 3.2.

Name	License		Global Method	Main Interfaces	Release/ Update
	Free	Academic			
APOPT [52]	Yes	Yes	LS	AMPL, Matlab python	2013
BQPD [53]	No	No	LS	TOMLAB	2013
CONOPT [54]	No	Yes	LS	AMPL, Matlab python, GAMS, MPL	2015
KNITRO/ACTIVE [55]	No	Yes	penalty	AMPL, Matlab c, c++, fortran	2013
LINDO [56]	No	No	LS	AMPL, C	2013
LRAMBO [57]	No	Yes	LS penalty	C	—
NPSOL [58]	No	Yes	LS penalty Lag	Matlab, C, C++	2008
QPOPT [59]	No	Yes	LS	Matlab, Fortran	—
SNOPT [60]	No	Yes	LS penalty Lag	AMPL, Matlab, C, C++, Fortran	2013
FILTERSQP [61]	Yes	Yes	TR filter	Matlab, Tomlab, AMPL	—

Table 3.2. List Of Active Set Solvers

3.4.3 Interior Point Methods

Interior point methods (IPMs) are an alternative approach to the SQP methods, showing promising results for large-scale problems. For convex problems, they achieve polynomial convergency. Otherwise, IPMs converge super linearly [62].
Let us rewrite problem (3.1) in the following form

$$\min_{x,s} f(x) \tag{3.24a}$$
$$\text{subject to } c_E(x) = 0 \tag{3.24b}$$
$$c_I(x) - s = 0 \tag{3.24c}$$
$$s \geq 0, \tag{3.24d}$$

where inequalities are transformed into equalities by introducing slack variable s. Interior point methods can be seen as continuation (primal-dual) [63] or as barrier methods [64].

Primal-Dual Approach

The KKT conditions (3.3) for (3.24) can be written as

$$\nabla f(x) - A_E^T(x)\lambda_E - A_I^T(x)\lambda_I = 0 \tag{3.25a}$$
$$S\lambda_I - \mu e = 0 \tag{3.25b}$$
$$c_E(x) = 0 \tag{3.25c}$$
$$c_I(x) - s = 0 \tag{3.25d}$$
$$\mu = 0, \quad s \geq 0, \quad \lambda_I \geq 0, \tag{3.25e}$$

where A_E, A_I are Jacobians of the c_E, c_I and λ_E, λ_I are their Lagrange multipliers. S is diagonal matrix with diagonal s and $e = (1, 1, ..., 1)^T$.
The estimation of the optimal set in (3.25) becomes to a combinatorial problem for $\mu = 0$ [65; 62]. The number of choices for the active set may be very large - up to 2^i, where i is a number of inequality constraints (constraint can, or cannot belong to the set). The number of possibilities is growing exponentially with the number of inequalities. It is not possible to design an algorithm, which will consider all possible options for the active set. Therefore, let us consider $\mu > 0$. The primal-dual approach solves the perturbed KKT [66] for a sequence of parameters $\{\mu_k\}$, that converges to zero, while maintaining $s, \lambda_I > 0$. By applying Newton's method to (3.25), we obtain the following primal-dual system

$$\begin{pmatrix} \nabla^2_{xx} L & 0 & -A_E^T & -A_I^T(x) \\ 0 & Z & 0 & S \\ A_E(x) & 0 & 0 & 0 \\ A_I(x) & -I & 0 & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_s \\ p_{\lambda_E} \\ p_{\lambda_I} \end{pmatrix} = - \begin{pmatrix} \nabla f(x) - A_E^T(x)\lambda_E - A_I^T(x)\lambda_E \\ S\lambda_I - \mu e \\ c_E(x) \\ c_I(x) - s \end{pmatrix}, \tag{3.26a}$$

where Z is a diagonal matrix with diagonal λ_I and $\nabla^2_{xx} L$ denotes the Lagrangian function for (3.24)

$$L(x, s, \lambda_E, \lambda_I) = f(x) - \lambda_E^T c_E(x) - \lambda_I^T (c_I(x) - s). \tag{3.26b}$$

After determining the step $p = (p_x, p_s, p_{\lambda_E}, p_{\lambda_I})$, the new iterate $(x_{k+1}, s_{k+1}, \lambda_{E_{k+1}}, \lambda_{I_{k+1}})$ can be

computed as

$$x_{k+1} = x_k + \alpha_1 p_x \tag{3.27a}$$
$$s_{k+1} = s_k + \alpha_1 p_s \tag{3.27b}$$
$$\lambda_{E_{k+1}} = \lambda_{E_k} + \alpha_2 p_{\lambda_{E_k}} \tag{3.27c}$$
$$\lambda_{I_{k+1}} = \lambda_{I_k} + \alpha_2 p_{\lambda_{I_k}}, \tag{3.27d}$$

where parameters α_1, α_2 are calculated from

$$\alpha_1 = max\{\alpha_1 \in (0,1] : s + \alpha_1 \geq (1-\tau)s\} \tag{3.28a}$$
$$\alpha_2 = max\{\alpha_2 \in (0,1] : \lambda_I + \alpha_2 \geq (1-\tau)\lambda_I\} \tag{3.28b}$$

with $\tau \in (0,1)$.
After a sufficient amount of iterations, the algorithm converges to a point, which satisfies (3.25) for (3.24). If the implementation of the algorithm requires iterates to decrease a merit function, or to be acceptable by the filter, the iteration is likely to converge to a minimizer, not just to the KKT point [62]. Thus, this approach has a locally unique solution, which can be denoted by $(x(\mu), s(\mu), \lambda_E(\mu), \lambda_I(\mu))$.
The methodology of basic primal-dual IP algorithm is summarized in Algorithm 8. The assumed error function is based on the perturbed KKT system (3.25) and is given by

$$E(x,s,\lambda_E,\lambda_I;\mu) = max\{||\nabla f(x) - A_E(x)^T\lambda_E - A_I(x)^T\lambda_I||, ||S\lambda_I - \mu e||, ||c_E(x)||, ||c_I(x) - s||\}. \tag{3.29}$$

Algorithm 8 Interior-point Algorithm

Require: initial estimate $x_0 \in R^n$ and $s_0 \leq 0$, compute initial multipliers λ_{E_0} and λ_{I_0}, select $\mu_0 > 0$, $\tau \in (0,1)$, set $k = 0$
  while stopping test is not satisfied **do**
    while $E(x_k, s_k, \lambda_{E_k}, \lambda_{I_k}; \mu_k) \leq \mu_k$ **do**
      solve (3.26) to obtain $p = (p_x, p_s, p_y, p_z)$
      compute α_1, α_2 with help of (3.28)
      compute $(x_{k+1}, s_{k+1}, \lambda_{E_{k+1}}, \lambda_{I_{k+1}})$ using (3.27)
      $\mu_{k+1} = \mu_k$
      $k = k + 1$
    end while
    choose $\mu_k \in (0, \mu_k)$
  end while

Barrier Approach

The barrier approach consists of finding the solution of the barrier problem

$$\min_x f(x) - \mu \sum_{i \in I} log\ s_i \tag{3.30a}$$
$$\text{subject to } c_E(x) = 0 \tag{3.30b}$$
$$c_I(x) - s = 0, \tag{3.30c}$$

for a sequence of positive barrier parameters $\{\mu_k\}$, that converges to zero. Thus, this approach avoids the combinatorial aspect, but the solution does not coincide with the one of (3.1) for $\mu > 0$ [64]. The KKT conditions for (3.30) take the following form

$$\nabla f(x) - A_E^T(x)\lambda_E - A_I^T(x)\lambda_I = 0 \tag{3.31a}$$
$$-\nabla S^T e + \lambda_I = 0 \tag{3.31b}$$
$$c_E(x) = 0 \tag{3.31c}$$
$$c_I(x) - s = 0. \tag{3.31d}$$

The conditions differ from (3.25) just in the second equation (3.31b). Newton's method can transform (3.31b) into a quadratic equation, by multiplying (3.31b) by S. This procedure does not change the solution of (3.31), because the diagonal elements of S are positive. After this transformation, the KKT conditions for the barrier problem coincide with the perturbed KKT system (3.25) and can be solved.

Software

The interior point methods show their strength in large-scale applications, where they often (but not always) outperform active-set methods. One of the main weaknesses of IP methods is their sensitivity to the choice of the initial point, a sensitivity to the poor scaling of the problem, and the update strategy for the barrier parameter μ. If μ_k decreases slowly, a large number of iterations is needed. If it decreases too quickly, the progress per iteration is too slow. To tackle this tradeoff, the Fiacco-McComick strategy [67] or adaptive strategies [51] can be applied.
A wide range of available solvers packages is based on the interior-point approach, some of them can be found in the Table 3.3.

Name	License		Global Method	Main Interfaces	Release/ Update
	Free	Academic			
DSDP [68]	Yes	Yes	LS	AMPL, C Matlab	2005
IPOPT [69]	Yes	Yes	LS, filter	AMPL, Matlab python, GAMS, AIMMS	2007
KNITRO/CG [55]	No	Yes	TR	AMPL, Matlab python, Mathematica	2013
KNITRO/DIRECT [55]	No	Yes	LS	AMPL, Matlab python, Mathematica	2013
LOQO [70]	No	Yes	LS, penalty barrier	AMPL, Matlab	2014
OOPS [71]	Yes	Yes	LS	SML	2011
QPSOL [72]	Yes	Yes	LS	AMPL, Matlab Fortran, ASTOS	2015
XPRESS [73]	No	Yes	LS	XAD, Matlab AMPL	2011

Table 3.3. List Of Interior Point Solvers

3.5 Quadratic Programming

Let us start by defining the objective function of the optimization problem as

$$\min_{x \in R^n} \frac{1}{2} x^T A x - b^T x \tag{3.32}$$

without considering any constraint. Then the minimization of problem (3.32) is equivalent to the linear system of equations

$$Ax = b. \tag{3.33}$$

This equivalence allows the use of linear system solvers for minimizing (3.32), e.g. direct linear solvers as the ones in Table 3.4, or iterative solvers as the ones in Table 3.5.

Direct Solvers	
Name	**Description**
LUSOL [74]	Square or rectangular A = LU, plus updating
MA48 [75]	Square or rectangular A = LU
MA57 [76]	Symmetric A = LDLT or LBLT
MUMPS [77]	Square A = LU, LDLT or LBLT (massively parallel)
PARADISO [78]	Square A(shared memory)
SUPERLU [79]	Square A (uniprocessor or shared or distributed memory)
UMFPACK [80]	Square A

Table 3.4. Direct Solvers For solving Ax = b

Iterative Solvers	
Name	**Description**
SYMMLQ [81]	Symmetric, nonsingular A (may be indefinite)
MINRES, MINRES-QLP [82]	Symmetric A (may be indefinite or singular)
GMRES [83]	Symmetric A (may be indefinite)

Table 3.5. Iterative Solvers For Solving Ax = b

A special case of problem (3.1) is a quadratic programming (QP) problem, where the objective function is quadratic (3.32) and the constraints are linear. Considering equality constraints, a general matrix formulation can be written as

$$\min_{x \in R^n} q(x) = \frac{1}{2} x^T H x - x^T f \tag{3.34a}$$

$$\text{subject to } Ax = b. \tag{3.34b}$$

The KKT conditions for problem (3.34) can be expressed as following

$$\begin{pmatrix} H & -A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x^* \\ \lambda^* \end{pmatrix} = \begin{pmatrix} -f \\ b \end{pmatrix}. \tag{3.35}$$

The system can be rewritten in a form that is more suitable for computation by expressing $x^* = x + p$, where x is an estimate of the solution and p is the desired step. By rearranging these equations, we obtain

$$\begin{pmatrix} H & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} -p \\ \lambda^* \end{pmatrix} = \begin{pmatrix} g \\ h \end{pmatrix}, \tag{3.36}$$

where $h = Ax - b$, $g = f + Hx$, $p = x^* - x$.
System (3.36) can be solved using either direct, or iterative techniques. Direct approaches are, for example, Schur-complement method [84], null-space method [85] or factorization of full KKT system [86]. Iterative approaches can be represented by Krylov methods, e.g. GMRES, QMR.
Almost all of the nonlinear optimization algorithms need to solve a sequence of quadratic subprograms. Therefore, software for solving quadratic programs can be found as part of the active set and interior point solvers described through this chapter. The convex quadratic programming problem represents special type of nonlinear problem, which can be easily linearized [87]. Therefore, apart from nonlinear optimization techniques, the linear programming methods and their extensions can be employed to solve problem (3.1). Their implementations can be found in linear programming solvers, for examples refer to Table 3.6.

Name	License		Main Interfaces	Realease/ Update
	Free	Academic		
CPLEX [88]	No	Yes	C, C++, Matlab Java, TOMLAB, MPL	2014
SOPLEX [89]	Yes	Yes	C, C++	2015
GUROBI [90]	No	Yes	AMPL, Matlab , R python, MPL, C	2015
MOSEK [91]	No	Yes	AMPL, Matlab, C, C# python, GAMS, R	2014
OOQP [92]	Yes	Yes	AMPL, C, C++ , Matlab	2014

Table 3.6. List Of Linear Programming Solvers With Extension To Convex Programming

3.6 Conclusion

This chapter gives an overview of the nonlinear optimization techniques. Ensuring faster convergence, local methods (i.e. interior point and active set methods) are often combined with globalization and convergency improvements, which gives the possibility to build huge amount

of various algorithms. In order to investigate which type of algorithms fit for solving problem (2.20) in the best possible way, the benchmark described in the next chapter was created.

Chapter 4

Benchmark

The FEM framework alternatively solves optimization problems with respect to Θ and Γ. For fixed Θ, the switching process Γ is obtained by solving linearly constrained quadratic minimization problem with respect to Γ.
In order to investigate available optimization techniques, problem (2.20) was simulated to perform evaluation through benchmark tests. This chapter describes simulation of the problem and tools applied for benchmark.

4.1 Simulation

To stay consistent with standard notation, let us rewrite problem from (2.20) to the following form by using standard notation

$$\min_x \; x^T H x + f^T x, \tag{4.1a}$$

$$\text{subject to } Ax = b, \tag{4.1b}$$

$$x \geq 0. \tag{4.1c}$$

The objective function is quadratic, constraints are linear and we have one box constraint on variable x. Problem has zero degrees of freedom and its complexity depends on the size of parameters N_K, N_S, N_T, regularizations factors ϵ_1, ϵ_2 and on the structure of matrix H_{2k}, describing relationships between different space locations.
To perform benchmark of the problem (4.1) resulting in the FEM framework, the simulation was created in the environment Matlab. We assume three possibilities in the structure of matrix H. To stay comprehensible, we are using following names for test cases: sparse, structured and half.
The matrix H is defined as sum of matrixes H_{1k} and H_{2k}. In sparse case, matrix H_{2k} is empty, thus no relationships between locations are considered. Definition of H_{2k} in structured case follows description from Algorithm 1, in half case follows the example where cross-correlation function was applied. But, in order to perform the simulation, which closely represents the real applications, the values on the positions specified by examples, were assigned randomly.
The matrixes H are well-structured, sparse, symmetric and blocked. In sparse case, matrix H is triagonal and positive-semidefinite. In structured and half case, the matrix H is positive-definite. Consequently, QP problem (4.1) is convex and has an unique solution.

Differences in the structure of the matrixes H are shown in Figure 4.1. Figure 4.2 demonstrates the structure of Jacobian A of the constraints, which does not depend on the version of H.

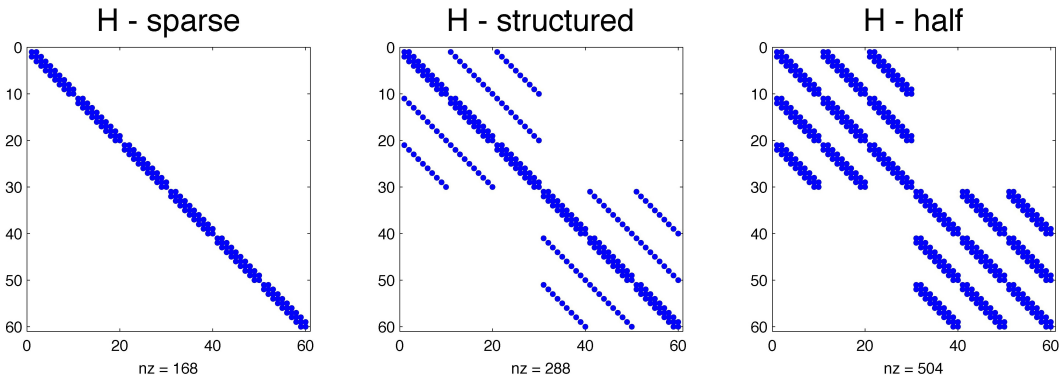

Figure 4.1. Structure Of Matrixes H, Generated For $N_T = 10, N_S = 3, N_K = 2$

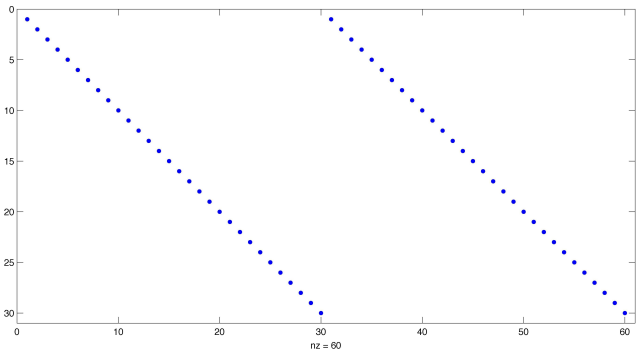

Figure 4.2. Structure Of The Jacobian Of Constraints, Generated For $N_T = 10, N_S = 3, N_K = 2$

4.2 AMPL Modeling Language

Accessing solvers is possible thrugh different interfaces, e.g. C, Matlab, R or python. An another option is to use a modeling languages (ML), e.g. AMPL [93] or GAMS [94]. To perform benchmark, we have chosen the AMPL modeling language. Choice was made due to following facts. AMPL is available for many 32- and 64-bit platforms including Linux, Mac OS X and Windows. Although free student version is limited up to 300 constraints, there are academic and trial licenses available. Modeling language allows us to switch between different solvers, while model stays untouched. All solver of our interest are compatible with this environment. Additionally, AMPL provides advanced automatic differentiation tools. From our point of view, AMPL is convenient tool for benchmarking solvers, although for real-application use, we prefer different interfaces, such as Matlab or C.
Before performing any optimization algorithm, effort needs to be expended to formulate the

underlying model and to generate the data structures. AMPL syntax is similar with the syntax of the mathematical notation of the optimization problems. Language mixes declarative and imperative programing style. Formulation of optimization models takes place through declarative language elements such as sets, scalar and multidimensional parameters, decision variables, objectives and constraints, which allow description of the most problems. After a model is prepared, it must be combined with the data to describe one particular optimization program. Data values might have different sources. We provide, which was generated using Matlab in standard CSC matrix format [95].
AMPL carries out translation of the model to the solver by following phases. The parse phase reads the model file and parses it into a corresponding expression tree. Read data phase checks validity of the data, e.g. if supplied values are numbers, or if set of numbers has a right dimension and so on. The compile phase transforms the expression tree to the computations entities. Then AMPL invokes presolve phase, which is optional and based on user preferences.
Finally, output phase makes the translated model available to an optimizer. Output routine is provided in MPS [96] or .nl [97] format, which are formats for presenting and archiving mathematical programming problems. Many solvers accept formats either directly or through special drivers. After a solver solves the problem and terminates, it returns the solution back to AMPL and AMPL to the user.

4.3 Software Packages Chosen For Benchmark

Combinations of local models, globalization strategies and convergency improvements allow to build complex algorithms. Their implementations can be found in various software packages. In order to compare several different algorithms and methodologies, we have chosen eight solvers, emphasizing on ones with free academic or public license.

4.3.1 SNOPT - Sparse Nonlinear Optimizer

Algorithmic Methodology

SNOPT is SQP solver designed to solve problems with many thousands of constraints and variables. The solver makes use of the first order derivation, which are assumed to be provided by user. If they are not, SNOPT can supply them by finite difference technique.
The SQP algorithm uses an augmented Lagrangian merit function and makes explicit provision for infeasibility in the original problem and the QP subproblems. The Hessian of the Lagrangian is approximated using a limited-memory quasi-Newton method.
Solver solves each QP using SQOPT [98], which uses a reduced-Hessian active-set method. Part of SQOPT package is solver LUSOL [99] for obtaining LU factors. If a QP subproblem is found to be infeasible or unbounded, then SNOPT tries to solve an elastic problem, that corresponds to a smooth reformulation of the penalty function.
For detailed informations about the solver, we refer the interested reader to [60].

Technical Details

The solver is implemented in Fortran77 and is compatible with newer Fortran compilers. It is possible to call it from other programs written in C or Fortran and from modeling languages such as AMPL or GAMS. The solver is able to save basis files, that can be used for warm start

of similar QPs. SNOPT is available under commercial and academic licenses from Stanford Business Software Inc.

4.3.2 CONOPT

Algorithmic Methodology

CONOPT uses a line-search approach and implements three following methods

- A Sequential Linear Programming Method
- A Sequential Quadratic Programming Method
- A Generalized Reduced Gradient (GRG) Method.

Two first methods are not anymore under development. GRG method projects the gradient of the objective onto a linearization of the constraints. Then, it makes progress toward the solution by reducing the objective. Algorithm has been designed to be efficient for a broad class of models. In general, the GRG method is shown to be reliable and more efficient for models with a large degree of nonlinearity. Extensions of the GRG method such as preprocessing, linear mode iterations makes CONOPT efficient on easier nonlinear models as well. Solver can be also used to solve linear programs, but doesn't solve any of the integer programs.
In this benchmark, we use solver in GRG mode. More information about solver can be found in [54] .

Technical Details

Solver has been developed by ARKI Consulting & Development A/S and has interfaces to GAMS, AMPL or AIMS modeling languages. It requires the second order derivatives to be provided with high order of accuracy, if they are not provided CONOPT does not solve the problem. Solver can be obtained under free academic license or under commercial license from its website [54].

4.3.3 IPOPT - Interior Point Optimizer

Algorithmic Methodology

IPOPT is a nonlinear programming solver, designed for solving large-scale, sparse problems. The solver implements line-search filter interior-point method. The outer loop minimizes approximately a sequence of nonlinearly constrained barrier problems, while decreasing barrier parameter. The inner loop uses a line-search filter SQP method for solving each barrier problem. However, global convergence of each barrier problem is enforced through a filter method, the filter is reseted after each barrier parameter update. The inner iteration includes second-order correction steps and mechanisms for switching to a feasibility restoration if the step size becomes too small. The solver provides the option for using limited-memory BFGS [100] updates to approximate the Hessian of the Lagrangian.
Detailed descriptions of the solver is given in [69].

Technical Details

The solver is written in C++ and has interfaces to C, C++, Fortran, AMPL, CUTEr and so on. It requires BLAS [101], LAPACK [102], and one of sparse indefinite solvers (MA27 [103], MA57 [104], PARDISO [105], or WSMP [106]). Combined with linear solver PARDISO, it can solve large problems on shared-memory.
Solver uses the second order derivatives, which need to be provided by user. Routines are available on COIN-OR under EPL(Eclipse Public License). Therefore, it's available for free also for commercial purposes.

4.3.4 KNITRO

Algorithmic Methodology

KNITRO is suitable for both continuous and discrete optimization models. But, it is primary designed for large-scale, continuous nonlinear problems. The solver provides implementation of four different algorithms

- a barrier method with direct factorization
- a barrier method using the conjugate gradient method
- an active set method
- a sequential quadratic programming method.

The solver offers concurrent optimization mode, in which all four algorithms are launched to solve the same problem simultaneously. Mode terminates, when the first solver stops. Solver also uses cross over algorithm, which allows switching between algorithms. For example, approximate solution points determined by a barrier algorithm can be converged to more accurate solutions by crossing over to the active set algorithm.
KNITRO implementation of interior-point penalty-barrier method is based on trust region approach, where trust regions and a merit function are used to promote convergence. The Interior/Direct method computes new iterates by solving the primal-dual KKT matrix using direct linear algebra. The method may switch into the Interior/CG algorithm, if it encounters difficulties. Then the primal-dual KKT system is solved by using a projected conjugate gradient method.
For the proposed benchmark, we use KNITRO in Interior/Direct mode. More information about the solver are given in [55].

Technical Details

KNITRO is written in C++ and has interfaces to a range of modeling languages, including AIMMS, AMPL, GAMS, Mathematica, MATLAB, and MPL. In addition, solver has interfaces to C, C++, Fortran, Java, and Excel. It requires linear solvers such as MA57 [104], or Pardiso [105] to solve the indefinite linear systems of equations.
Derivatives are recommended to be supplied by user or by the interface, otherwise KNITRO generates them by finite-differencing. All routines are available from Zienna Inc [55].

4.3.5 CPLEX

Algorithmic Methodology

The IBM ILOG CPLEX Optimizer is a high performance solver for Linear programming and Mixed Integer Programming. It offers various algorithms as

- a primal simplex algorithm
- a dual simplex algorithm
- a the barrier algorithm
- a network simplex algorithm.

Solver proved to be reliable, robust and able to solve huge amount of different optimization problems. Apart from linear programming problems, the CPLEX is also able to solve convex quadratic programming problems, and convex quadratically constrained problems. For our problem, the dual simplex optimizer proved to achieve the best results. Therefore, during the benchmark optimizer runs in this mode. CPLEX is a commercial product, ergo it was not possible to find detailed description of implemented algorithms.

Technical Details

CPLEX is implemented in C and is accessible through independent modeling systems such as AIMMS, AMPL, GAMS, MPL, OpenOpt, OptimJ and TOMLAB. The solver is available for commercial, academic or trial use through its website [88].

4.3.6 GUROBI

Algorithmic Methodology

Gurobi is considered to be a state-of-the-art solver [107] for Linear Programming, Mixed integer Programming and Quadratic Programming Problems with linear constraints. Implementation includes following algorithms

- a primal simplex method
- a dual simplex method
- a barrier method.

During benchmark, Gurobi runs in barrier mode. As solver is under commercial license, it is not possible to get access into algorithmic description.

Technical Details

The Gurobi Optimizer supports a number of programming and modeling languages including C++, Java, .NET, Python, C, MATLAB, and R, AIMMS, AMPL, GAMS, and MPL. The solver is developed by Gurobi Optimization, Inc. and available through its website [90] in commercial and student/ academic free license.

4.3.7 MOSEK

Algorithmic Methodology

MOSEK is a software package offering various algorithms as

- an interior-point optimizer for all continuous problems
- a conic interior-point optimizer for conic quadratic problem
- a simplex optimizer for linear problems
- a mixed-integer optimizer based on a branch and cut technologies.

The solver is well suited for solving generalized linear programs involving nonlinear conic constraints. MOSEK also proved to be efficient with Quadratic Programming problems and problems that have a quadratic constraints.
While benchmarking, we turned solver into conic optimizer mode. More information about solver can be found in [91].

Technical Details

MOSEK provides interfaces to the C, C#, Java and Python languages. It is also compatible with AIMMS, AMPL, and GAMS, Matlab or R.
For academic usage, solver is available for free, otherwise it can be obtained under commercial license.

4.3.8 PermonQP

Algorithmic Methodology

The PermonQP is a stand-alone package intended for solving unconstrained or equality constrained QP problems. It is built on PETSc KSP [108] package, which provides an interface to the sequential direct solvers and to the Krylov iterative solvers, e.g. gmres, minres combined with various choice of preconditioner. Additionally, it can be conjuncted with external parallel direct solvers, e.g. MUMPS [109], SuperLU [110]. The solution process consists from sequences of following actions

- specification of QP problem
- QP transformation, which reformulates the original problem and create a chain of QP problems
- passing last problem from sequence to the KSP solver.

Solver provides promising numerical and parallel scalability, but up to date, it is still under development, using file based interface and suffering from luck of documentation. More information about algorithmic methodology can be found in [111].

Technical Details

Solver can be used on all main operating systems and architectures from personal computers to supercomputers. Implementation combines ANSI C and PETSc 3.4. library or higher. Solver supports MPI, shared memory pthreads and GPUs through CUDA or OpenCL, as well as MPI-shared memory pthreads or MPI-GPU parallelism. Product is distributed under 2-clause BSD license and can be obtained from website [112].

4.4 Automatic Differentiation

Nonlinear optimization solvers need good gradient approximations, therefore they require users or the modeling environment to provide first-order and sometimes second-order derivatives. Some solvers can estimate derivatives by finite difference technique, based on the Taylor's theorem [113]. Thereby, the approach observes the change in function values to small perturbations of the unknown close a given point x. Then, it estimates the response to infinitesimal perturbations, which is equal to the derivative. This approach has several drawbacks, e.g. rand-off errors.
An automatic differentiation (AD) [114; 115] is an alternative gradient computations technique, which is much faster than finite differences. It is built on fact, that computer code for evaluating any function can be broken down into a structure of elementary operations on which the chain rule can be applied. It can be performed in two modes: forward and reverse. Forward AD recurs partial derivatives with respect to the input variables, but its computational complexity is proportional to the complexity of the original code. In contrast, the backward AD recurs the partial derivatives (called adjoints) of the final result with respect to the result of each operation. Although, backward AD is more complicated to implement, it is faster than forward AD. Therefore, backward AD is particularly well suited to compute derivatives needed by many nonlinear optimization solvers.
Part of AMPL environment is AMPL Solver Library (ASL) [116], which provides the automatic differentiation tools. While running benchmark, we took advantage of it.
For more informations about automatic differentiation tools, we refer interested reader to [117].

4.5 Presolver

Presolving, known as preprocessing, is carried out in practical optimization to reduce size of the user-defined problem, before passing it to the solver. The techniques are used to eliminate variables, constraints and bounds from the problem. Any simplifications that presolver makes are reversed after the solution is returned, so that user can see the solution in terms of the original problem.
Using AMPL modeling language, following techniques are applied.

- ROW SINGLETON
  If constraint k involves only variable j, thus $A_{kj} \neq 0$, but $A_{ki} = 0, \forall i \neq j$, the variable x_j can be immediately determined as

$$x_j = \frac{b_k}{A_{kj}} \tag{4.2}$$

and eliminated from the problem. If value of x_j violates its bounds, e.g. $x_j < l_j$ or $x_j > u_j$, problem is declared as infeasible and process terminates before passing data into the solver.

- ZERO ROWS AND COLUMNS
  If $A_{ki} = 0, \forall i = 1, 2, .., n$, then the right hand side is also zero. The row can be deleted from the problem and corresponding λ_k can be set to an arbitrary value.

  For zero column, the optimal value of x_j can be determined by inspecting its cost coefficient c_j and its bounds l_j and u_j. If $c_j < 0$, $x_j = u_j$ to minimize objective function. If $c_j < 0$ and $u_j = +\infty$, then the problem is defined as unbounded. Similarly, if $c_j > 0$, the variable is set to the $x_j = l_j$. If $l_j = -\infty$, unboundedness is declared and program terminates.

- FORCING CONSTRAINTS
  Some constraints are satisfied only when certain combination of the variables hold. No other combinations are possible, due to the box constraints. Therefore, the variables can be assigned to the particular value and eliminated from problem together with equality constraint.

- DOMINATED CONSTRAINTS
  Sometimes, no matter on value of the variable, bounds are always satisfied. Therefore, box constraints can be dropped from the formulation and variable can be treated as a free variables.

Additionally to AMPL presolver, solvers provide their own implementation. The effect of presolver is very problem dependent. To investigate an influence of presolver for problem (4.1), we had chosen six test problems, which vary on size and structure of H. The Figure 4.3 presents results obtained by solver KNITRO.

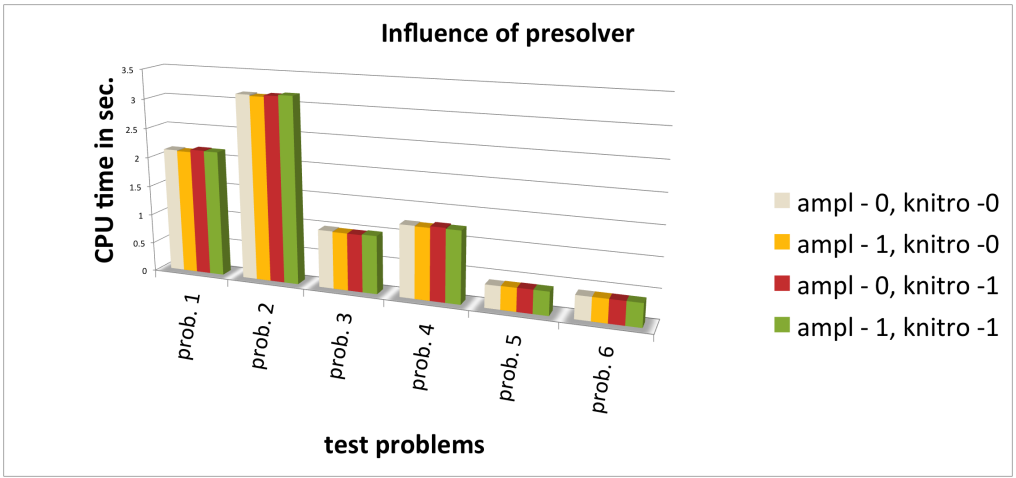

Figure 4.3. The Influence Of Presolver To KNITRO Performance, 1 Indicates Presolver Turned On, 0 Turned Off

It can be seen that AMPL presolver reduces running time, while KNITRO presolver does not achieve any reduction. According to the fact, that decrease in time is not significant and in order to stay consistent, all presolvers were turned off during benchmark.

4.6 Initial Settings

Although some optimization algorithms generally perform better than others, their efficiency depends on the problem domain. Most solver phases have parameters which can be tweaked and influence the results. In order to achieve the best outcome, there is a need for optimal solver configuration.
The space of possible options can be immense with respect to the solver. Some solvers, e.g. KNITRO or GUROBI provide parameter tuning tool to automate search for optimal settings. General, no-commercial software package as SNOPT or IPOPT, do not offer possibility to invoke such a tool. Therefore, adjustment was reached by playing out with different solver phases with different settings against each other manually. For example, the most significant speedup for IPOPT was accomplished by setting Hessian and Jacobian as constants.
In case, that optimizers require an initial guess to be supplied by user, the value was set up to the zero. Some solvers provide a hot start, where by keeping basis from similar problem, or from previous runs can obtain notable acceleration. While executing benchmark, basis were reseted after every run and cold start was enforced.

4.7 Solver Terminations And Convergency Criteria

Normal termination of solvers corresponds to the termination at an approximate KKT point (3.3), but solvers can also detect infeasibility or unboundedness. If the optimization problem is convex or some restrictive assumptions are made, methods guarantee convergence to the feasible point. Otherwise, problem may not have a feasible point. To check, whether a given program is unbounded is difficult. Most solvers use a user supplied lower bound on the objective and then terminate, if they detect a feasible point with a lower objective value than the lower bound.
Termination of solvers is based on satisfying the convergency criteria. Completely fair comparisons can be made only if solvers have a uniform stopping criteria. Unfortunately, an access to the source codes is not possible, therefore the uniform stopping criteria is hardly ever satisfied. Let us consider problem in following form

$$\min_x \quad f(x) \tag{4.3a}$$
$$\text{subject to } c(x) = 0 \tag{4.3b}$$
$$x \geq 0. \tag{4.3c}$$

Given tolerances τ_f, τ_C and τ_S, let us assume convergence tests as

$$c_k(x) \leq \tau_f \tag{4.4a}$$
$$|\lambda_k(x)| \leq \tau_C \tag{4.4b}$$
$$||\nabla f(x) - \nabla c(x)\lambda(x)|| \leq \tau_S \tag{4.4c}$$

where x is an approximate solution, $\lambda(x)$ is estimate of multiplier and (4.4c) is residual on the KKT conditions. Each solver has defined its own convergency criteria, which varies from one described above. For example, SNOPT [60] requires

$$c_k(x) \leq \tau_f \tag{4.5a}$$

$$|c_k(x)\lambda_k(x)| \leq \tau_C. \tag{4.5b}$$

This is a stronger convergence test than one in (4.4a) and (4.4b), because if (4.5) holds, then

$$|\lambda_k(x)| \leq \frac{\tau_c}{\tau_f}, \quad \text{if } |c_k(x)| > \tau_f. \tag{4.6}$$

Furthermore, all solvers require the user to choose tolerances τ_P and τ_D for convergency tests. Based on this values, they set up values for τ_f, τ_C and τ_S. For example, SNOPT sets

$$\tau_f = \tau_P(1 + ||x||), \qquad \tau_C = \tau_S = \tau_D(1 + ||\lambda(x)||) \tag{4.7}$$

while KNITRO sets

$$\tau_f = \max\{\tau_P \max(1, ||c(x_0)||_\infty), \tau_0\}, \qquad \tau_C = \tau_S = \max\{\tau_D \max(1, ||\nabla f(x)||_\infty), \tau_0\} \tag{4.8}$$

for some absolute tolerance $\tau_0 > 0$.
As shown, convergency tests differ with respect to the solver. To ensure consistency and the same level of accuracy, we proceed following steps. First, we solve the program with solver CPLEX and save function value and multipliers to the file with fifteen digits of accuracy. Then, we invoke other solvers and force them to satisfy the same tolerance with threshold $\epsilon = 10^{-8}$. Therefore, after solver returns the solution, we compare it with the one obtained by CPLEX. If the threshold is not satisfied, we decrease values τ_D and τ_P and repeat the process until criterium is satisfied.

4.8 Measurements Of The Performance

Main difficulty in benchmarking solvers is to determine a reasonable criterium for measuring their performance. Informations of our interest are running time and number of iterations.
After establishing the uniform convergency criteria, described in Chapter 4.7, we launch a script to execute the benchmark over all test-sets for all solvers. The script tracks the number of iterations needed to converge and the wall-clock time from the start point when AMPL invokes solver to the end of this process. To avoid fluctuation, we cycle, while 10% accuracy is not satisfied.
Time of a parallel program starts of the execution on one processor and ends of all computations on the last processor. Total execution time T is determined by computation and communication as

$$T = T_{comp} + T_{comm} + T_{idle}, \tag{4.9}$$

where T_{comp} is time spent for computations, T_{comm} is time spent for communication (broadcast, send, receive) and T_{idle} is time spent for waiting (synchronize with other processes). Therefore, carrying out benchmark for parallel tests, we are additionally interested in CPU time, which gives us the time summed across all CPUs. Consequently, if the process used multiple threads, CPU time exceeds the wall clock time.

4.9 Conclusion

This chapter describe the benchmark created in order to compare different solvers packages. The AMPL modeling langue, providing additional tools as automatic differentiation or presolver was selected to execute benchmark. All solvers were adjusted, in order to obtain the highest performance. The script carrying out the benchmark was designed in a way, that ensures the uniform stopping criteria and tracks information of our interest, i.e. time and number of iterations. Next chapter presents the obtained results.

Chapter 5

Investigation Of The Optimization Strategies

This chapter analyzes and compares the results obtained by benchmarking selected optimization solvers. Emphasis is put on a) exploring the numerical properties of the QP problem (2.20) resulting from FEM framework b) finding the most suitable optimization technique.

5.1 Technical Details & Data Set

The procedure of benchmarking always starts from organizing the set of problems. For purpose of this work, we have created a data set. Test set contains the smaller versions of QP problem (2.20) with respect to the initial choice of parameters N_S, N_T combined with different alternatives of regularization factors ϵ_1, ϵ_2. Benchmark aims to

- explore numerical properties of problem (2.20)
- find the most efficient solver fitting to the FEM Matlab toolbox. Toolbox was developed in the research group "Computational Time Series Analysis" of prof. Horenko at ICS/USI Lugano and is intended for local machines, or for applications, where time dimension was reduced by setting the number of finite elements $M < N_T$.

The Table 5.1 displays ranges used to generate the test set.

Parameter	**Range**
N_T	100 : 100 : 500
N_S	1 : 2 : 20
N_K	2, 3
ϵ_1 / ϵ_2	0 / 0.0001; 0 / 1 ; 0.0001 / 0.0001; 0.0001 / 0.01;1 / 0.01; 1 / 1 ; 1 / 0.0001 0.0001 / 1 ; 0.0001 / 0 ; 0.01/ 0.0001; 0.1 /0.01 ; 0.1 / 1 ; 0.1 / 0; 1 / 1;

Table 5.1. Ranges Used To Generate The Test Set

Additionally, we divided the test set into three subsets with respect to the structure of matrix H, thus sparse, structured and half as described in Section 4.1. To provide summarization of created test set, we use quartiles, a tool from the descriptive statistics [118]. The quartiles of the data represent the three points, that divide the set into four equal groups. The first quartile q_1 is defined as the middle number between the smallest number and the median of the data. The second quartile q_2 stands for a median of the data and the third quartile q_3 is the middle value between the median and the maximum value of the data.
The Table 5.2 displays the quartiles for five problem parameters: number of equality constraints, number of variables n, number of non-zeros in Jacobian constraints, the numbers of non-zeros in the matrix H and the ratio $(n - n_e)/n$, where n_e is the number of equality constraints, often called degrees of freedom of the problem. All parameters are shared by all subsets, except amount of non-zeros in matrix H, which depends on the structure of the connection matrix H_{2k}. As can be seen from the table, all the parameters are distributed evenly throughout the test set.

	min	q_1	q_2	q_3	**max**
Num. of eq. constraints	100	900	1600	3 000	5 700
Num. of variables	200	1 950	4 050	7 000	17 100
Non-zeros in Jacobian	200	1 950	4 050	7 000	17 100
Degrees of freedom	0	0	0	0	0
Non-zeros in H - SPARSE test set	596	5 832	12 090	20 930	51 186
Non-zeros in H - STRUCTURED test set	596	13 855	53 799	116 928	358 986
Non-zeros in H - HALF test set	596	29 726	133 053	303 270	972 534
Non-zeros in H - FULL test set	596	9 622	29 773	11 7607	972 534

Table 5.2. Description Of The Test Set

System informations of the machine used for benchmark can be found in Appendix A.1. All solvers were downloaded under academic or free license in the latest stable version and the modeling language AMPL was obtained with a trial license.

5.2 Numerical Properties

The minimization problem (2.20) was obtained by finite element discretization of the continuous regularized problem (2.5) under conditions (2.16b), (2.16c). Considering the spatio-temporal settings of the FEM framework, we assume time and space regularization as described in Chapter 2.
The regularization factors ϵ_1, ϵ_2 have a smoothing effect, thus regularization restricts the number of transitions between clusters. Apart from controlling transitions, the regularization has also impact on the performance of the optimization problem (2.20), which is illustrated by following examples.

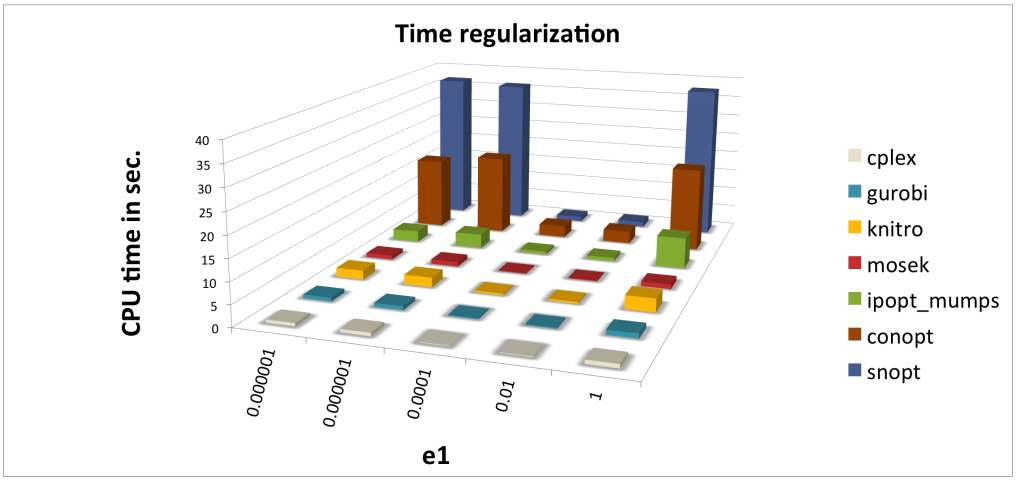

Figure 5.1. Effect Of Time Regularization, Generated For $N_T = 300$, $N_S = 11$, $N_K = 2$ For Sparse H

Figure 5.1 shows, that the problem becomes less computationally expensive, when settings $\epsilon_1 = 0.01$ or $\epsilon_1 = 0.0001$. Decreasing regularization factor ϵ_1 below 0.0001 makes the problem more time consuming, as applying no time regularization.

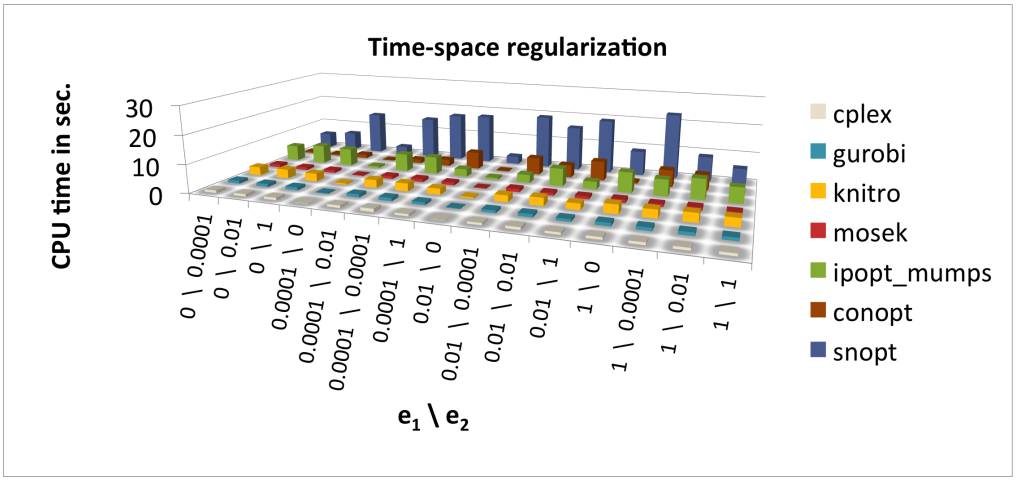

Figure 5.2. Effect Of Time, Space Regularizations To The Performance, Generated For $N_T = 300$, $N_S = 11$, $N_K = 2$ For Structured H

We can see from Figure 5.2 that different combinations of ϵ_1 and ϵ_2 have different impact on the performance. Results are solver dependent, but in general holds, that if one of two regularization factors is equal 1, problem is becoming more computationally expensive. If time regularization is fixed to 0.0001 or 0.01, decreasing factor ϵ_2 makes problem simpler. For $\epsilon_1 = 0$,

i.e neglecting temporal dimension and regularizing space by factors $\epsilon_2 = 0.01$ or $\epsilon_2 = 0.0001$, the problem can be solved without too much computational effort. The same holds for $\epsilon_2 = 0$, i.e. no connections between space locations are considered.
While benchmarking different optimization solvers, we also study influence of regularization parameters to the number of iterations needed for optimizers to converge. Figures 5.3 and 5.4 present results. In general, the cost per iteration is approximately equal for different ϵ_1, ϵ_2 settings. Solver CONOPT needs to perform the most iterations and in the same time, they are the most expensive ones among all benchmarked solvers. Cost of one interior point iteration is higher than cost of a linear programming iteration. The amount of iterations changes for all solvers depending on the choice of regularization parameters. SNOPT's run consists from major and minor iterations. While running QP program, SNOPT executes just one major iteration and then directly provides access to the SQOPT, which is a quadratic solver.

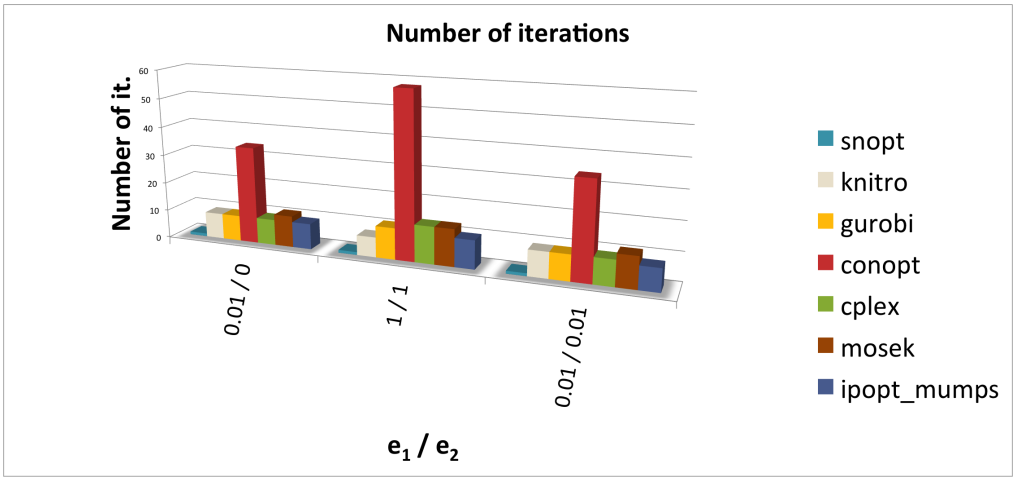

Figure 5.3. Number Of Iterations, Generated For $N_T = 300, N_S = 11, N_K = 2$ For Half H With Different Regularization Parameters

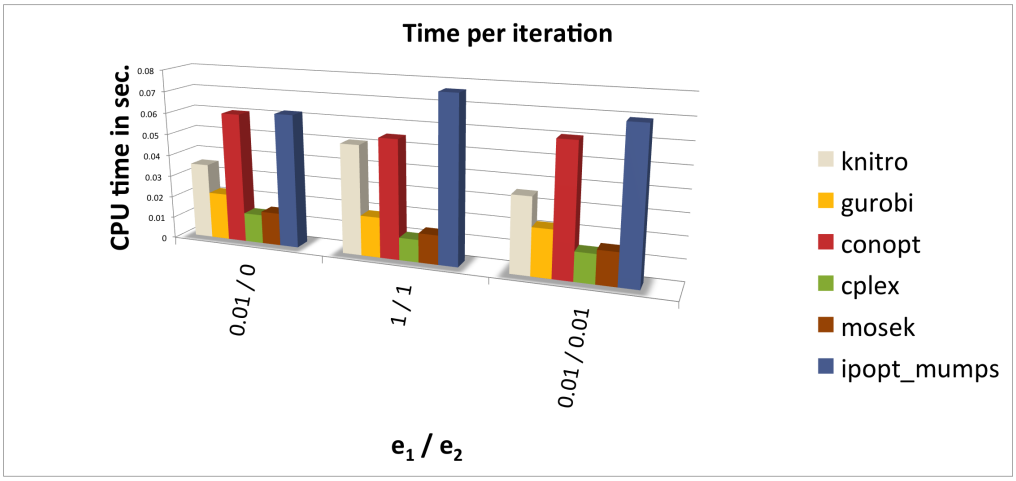

Figure 5.4. Time Needed Per One Iteration, Generated For $N_T = 300, N_S = 11, N_K = 2$ For Half H With Different Regularization Parameters

The relationship between the number of required iterations and the size of the problem is depicted in Figure 5.5. As one can see, that amount of iterations does not change so much for different instances of the problem. Solver CONOPT performs more iterations with increasing size of the problem. Interesting observation can be made by inspecting problem with following settings $N_T = 300, N_S = 7, N_K = 3, H =$ structured, which can be solved without to much computational effort by all solvers.

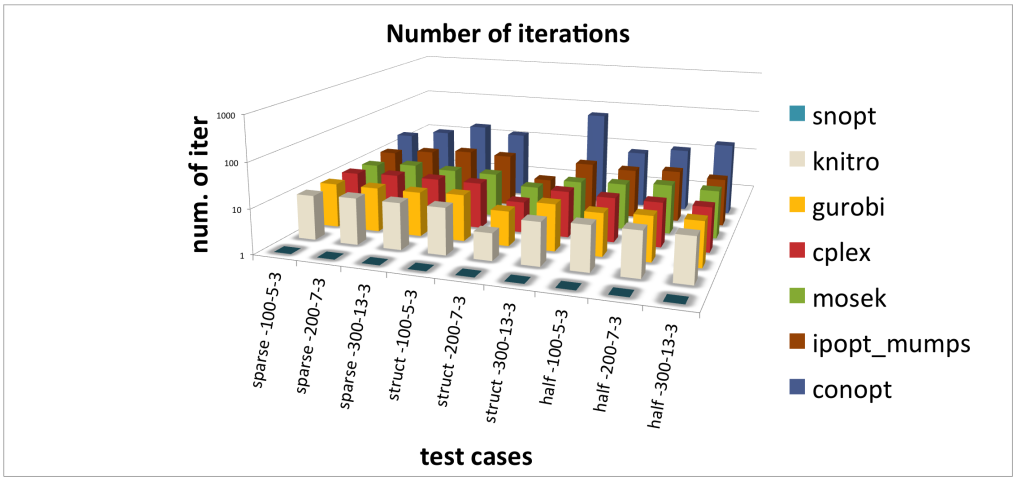

Figure 5.5. Number Of The Iterations Depending On The Size And The Structure Of The QP Problem, Generated For Regularization Parameters $e_1 = 0.001, e_2 = 0.01$, Parameters Take Following Order: Type Of H, N_T, N_S, N_K

The size and the structure of QP problem (2.20) is based on initial settings of parameters N_T, N_S, N_K, which are problem dependent. From Figure 5.6 we can see, how is performance changing with increasing dimension of matrix H. A jump pattern, which can be observed in presented results is caused by generating H with different density through test set. For example, size 3300 is result of $N_T = 100, N_S = 11, N_K = 3$, while 3000 stands from $N_T = 200, N_S = 5, N_K = 3$.

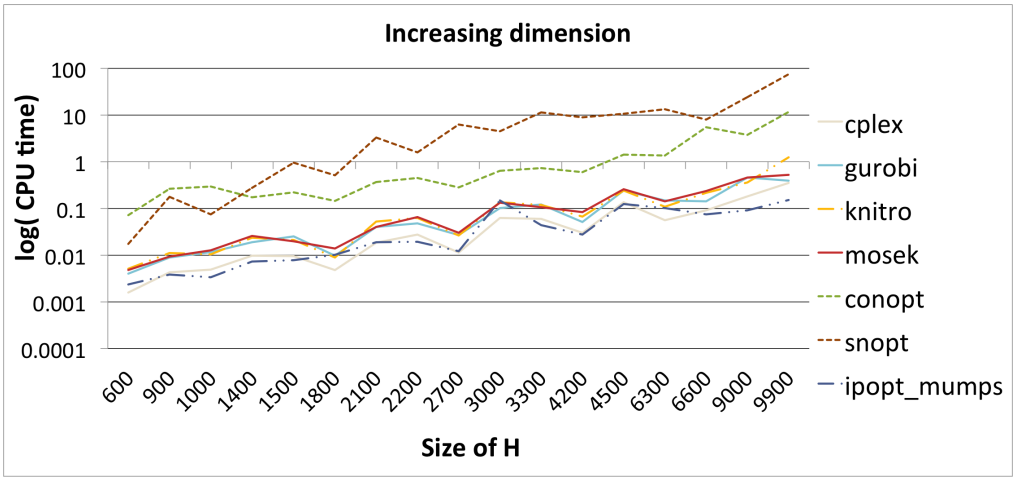

Figure 5.6. Increasing Size Of H, For Different Combinations Of $N_T \times N_S \times N_K$, Generated For H - Half, $e_1 = 0.0001, e_2 = 0.01$

While fixing all parameters and increasing the number of locations, the complexity of the problem tends to increase linearly as can be seen from Figure 5.7.

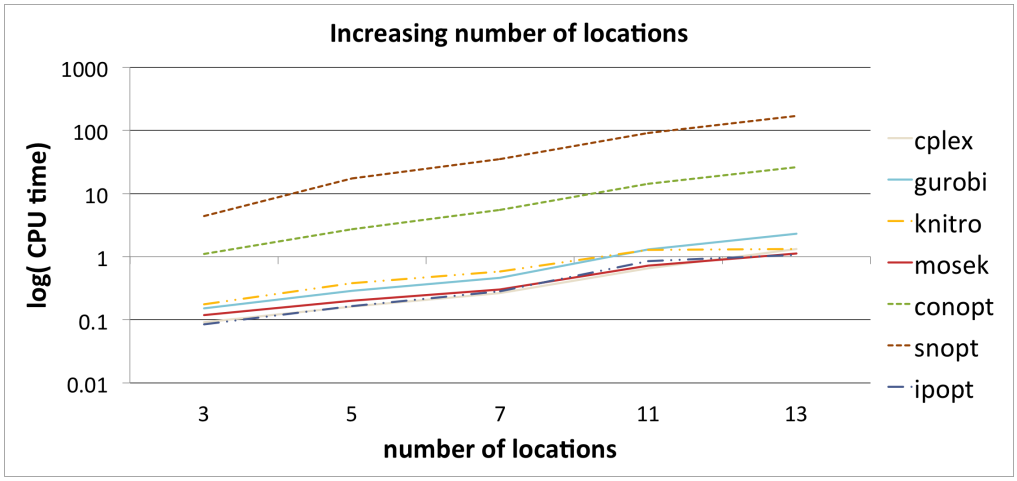

Figure 5.7. Increasing Number Of Location For Fixed $N_T = 300, N_K = 3, e_1 = 0.01, e_2 = 0.0001$, For H Half

By keeping all the parameters fixed and increasing density of matrix H_{2k}, thus considering more connections between locations, different performance can be observed for different solver categories. Linear solvers increase the performance slightly with an increased density, while interior point solvers gain more noticeable speed-down. Interesting observation can be done for SQP solvers, which are showing significant difficulties to solve structured version of problem.

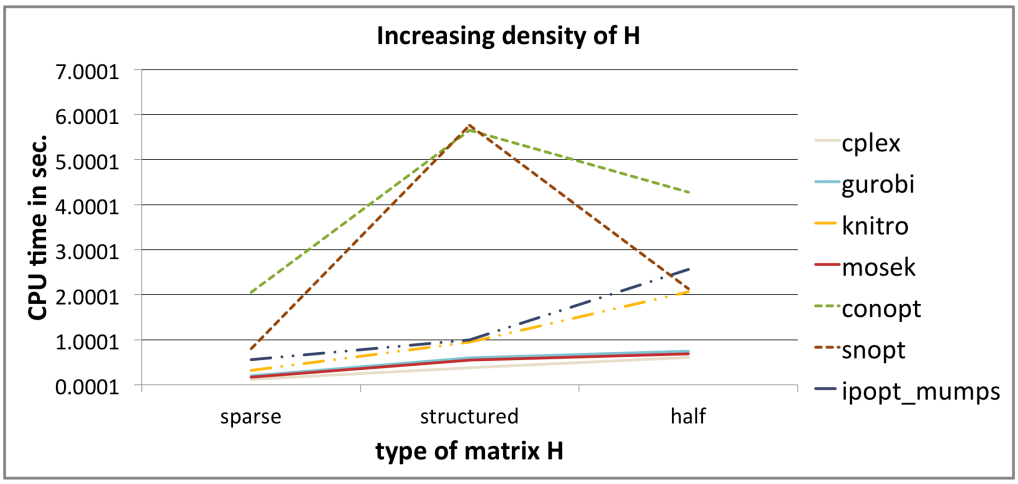

Figure 5.8. Increasing Density Of Problem For Fixed $N_T = 300, N_S = 11, N_K = 3, e_1 = 0.01, e_2 = 0.0001$

5.3 Evaluation Of The Solvers

In order to evaluate the quality of the different solvers, we use performance profiles [119]. We have recorded information of our interest, e.g. computing time to evaluate the set of solvers S on a test set P.
Let us assume, that we have n_s solvers and n_p problems. For each problem p and solver s, we define $t_{p,s}$ as computing time required to solve problem p by solver s.
We define

$$\tilde{t}_p = min\{t_{ps} : p \in P\}, \tag{5.1}$$

which represents the best statistic for a given problem p. Then for $\alpha \geq 0$, $p \in P$ and $s \in S$, we define

$$k(t_{ps}, \tilde{t}_p, \alpha) = \begin{cases} 1 & \text{if} \quad t_{ps} \leq \alpha \, \tilde{t}_p \\ 0 & otherwise \end{cases} \tag{5.2}$$

An overall assessment of the performance of the solver is then obtained by

$$p_s(\alpha) = \frac{\sum\limits_{s \in S} k(t_{ps}, \tilde{t}_p, \alpha)}{n_p}. \tag{5.3}$$

The function p_s is the cumulative distribution function and the values $p_s(\alpha)$ indicate the fraction of all examples, which can be solved within α times. The best strategy, e.g. $p_s(1)$, gives the fraction of which solver is the most effective one.
The solvers we benchmark have different requirements and represent three various categories: nonlinear interior point, sequential quadratic programming and linear programing solvers. To obtain overview in our context, we restricted values of parameter α to $\alpha \leq 10$. The performance profiles of the CPU time on the test set and on its subsets are presented on figures below. Every solver has been able to provide a solution for each problem. Thus, it was not necessary to introduce any penalty parameter.

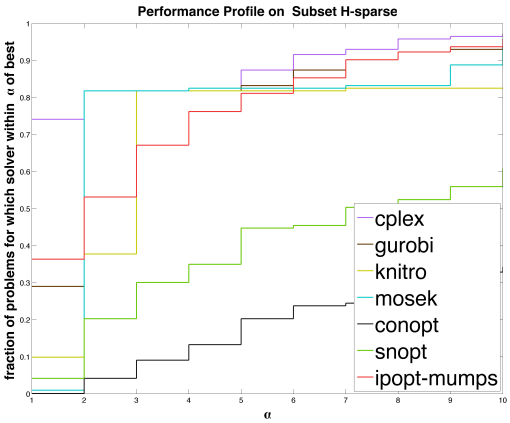

Figure 5.9. Subset Sparse

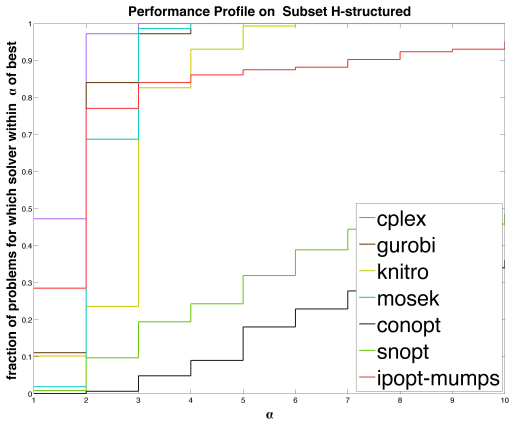

Figure 5.10. Subset Structured

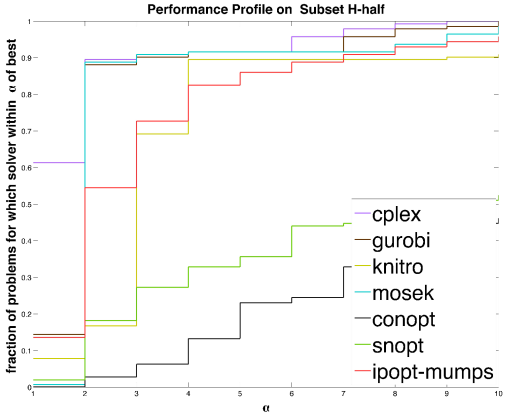

Figure 5.11. Subset Half

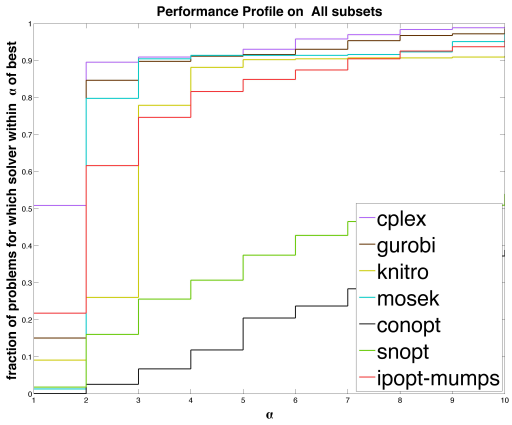

Figure 5.12. Full Set

From figures 5.9 to 5.12, it is clear that CPLEX has the most wins, closly followed by IPOPT. Other linear programming solvers do not show good assessment with respect to the "wins", but by increasing α to the value 2, we can see, that all linear programming solvers are becoming to be the best with probability around 80 %.
From interior point solvers, IPOPT has more wins than KNITRO, but setting α to the interval $[3:6]$, we can see, that probability of KNITRO being the best is higher. In general, considering parameter $\alpha > 4$, the interior point solvers are good competitors to the linear programming ones.
Fraction of problems, where SNOPT or CONOPT win is low. Both SQP solvers do not indicate very interesting performance, therefore our attention for future usage is captured by the interior point solvers and linear programming solvers. Incorporating experimental observations, the linear programming solvers are more memory demanding than interior point solvers. Therefore, from our point of view, the interior point solvers provide nice tradeoff.

Before closing this discussion, we add one remark. One benefit of SQP solver SNOPT is its ability to keep basis from previous runs of the same or similar problem. In such a case, SNOPT returns the solution almost immediately and easily outperforms all the other solvers. Performing benchmark, data were reseted after every run, therefore SNOPT needed to built basis always from zero, which took more time as can be observed by reports. If connecting SNOPT into the FEM framework, the first run of QP problem (2.20) may require more time. On every next iterate, the optimizer can use saved basis from previous run, which can result in better performance. Investigation of suggested strategy is task for future research.

5.4 Conclusion

The chapter presents results gained by benchmarking seven optimization solvers. By performing sequential testing, the numerical properties of the QP problem (2.20) were explored. Additionally, tests were used to find the optimizer providing the best performance. Outcome shows that the linear programming solvers and interior point solvers are more suitable choice than SQP solvers.

Chapter 6

Conclusion

As a result of the linearly constrained quadratic programming problem, which needs to be solved on every iterate, the current versions of the FEM framework are not as efficient as they could have been. In this thesis, different optimization strategies are analyzed, in order to decrease the time requirement of the QP problem.

6.1 Summary Of Conclusions

The Finite Element Time Series Methodology was presented in the spatio-temporal settings, in Chapter one of this thesis. The description of the approach was stated in the following steps: a) approximation b) regularization c) discretization d) extension into spatio-temporal settings. In addition to the description of the theory behind different nonlinear constrained optimization techniques, several available solver packages were also examined in Chapter two.
In the third Chapter, benchmark, which was used to examine the performance of eight different optimization solvers was illustrated. The QP problem was simulated in the environment Matlab while the tests were carried out by modeling language AMPL. Before executing the benchmark, the solvers were adjusted in order to obtain the best possible performance and to ensure the consistence of the unique termination criteria. While benchmarking, the information of our interest, such as time and number of iterations, was recored.
In the last Chapter, the obtained results were presented and discussed. The first part of the Section was dedicated to the study of the numerical properties of QP problem with respect to the initial parameters $N_T, N_S, N_K, \epsilon_1, \epsilon_2$. It was observed that by adding space location to the model, the time complexity is increasing with a linear trend. Time needed to run the same optimization problem with different regularization factors alters. In the classical "temporal" FEM settings, the values 0.01 and 0.001 of the parameter ϵ_1 reports the best results. In the spatio-temporal settings, the differences vary depending on the combination of the regularization factors.
In order to evaluate solvers, we used performance profilers. As the results, conclusions can be made that the linear programming solvers perform the best, closely followed by interior point solvers, while SQP solvers show the lowest performance.

6.2 Future Work

The aforementioned work can be extended in many ways. In addition to sequential benchmark, parallel benchmark can be performed. Trial tests were already launched and can be found in Appendix B.
Only relatively small test problems were performed on the "local" machines due to the memory constraints. In the future, it would be beneficial to expand project into comparison on HPC clusters.
Work can also be extended into benchmarking the memory requirements of the solvers, as only empirical observations are made currently. This show that the linear programming solvers are the most memory demanding.
As already pointed out in the Section 5.3, additional extension can be made by studying the performance of the sequence of QP problems resulted from FEM, instead of one single QP problem.

Appendix A

A.1 Technical Details

All benchmark test were done on machine with following architecture.

System Version: OS X 10.8.3 (12D78)
Capacity: 639.28 GB
Processor Name: Quad-Core Intel Xeon
Processor Speed: 2.26 GHz
Number of Processors: 2
Memory: 32 GB
L3 Cache (per Processor): 8 MB

Table A.1. System Informations

Appendix B

B.1 Introduction Of Parallel Testing

To perform example of parallel benchmark, we created test set of QP problems defined by range of initial parameters as shown by Table B.1.

Parameter	Range
N_T	1000 : 1000 : 7000
N_S	20
N_K	2, 3
ϵ_1 / ϵ_2	0.001 / 1

Table B.1. Ranges Used To Generate The Test Set For Doing Tests In Parallel

We compare performance of two different solvers IPOPT and PermonQP. Additionally to IPOPT version using linear solver MUMPS from sequential testing, we test also the version with the linear solver Pardiso. This choice of linear solvers allows us to run IPOPT in parallel. Figure B.1 presents results obtained by running benchmark on 4 cores.

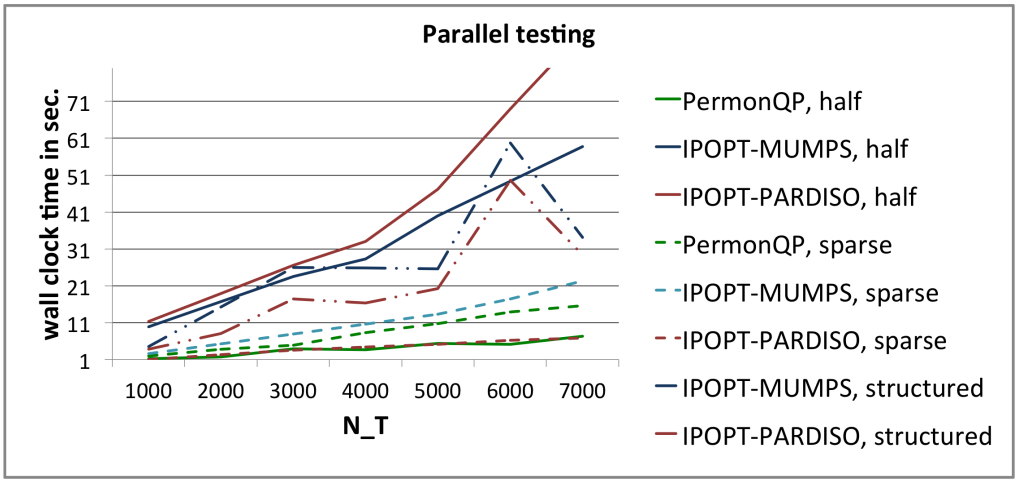

Figure B.1. Results From Parallel Testing

It can be seen, that for half matrixes, the PermonQP performs significantly faster than IPOPTs. For structured cases of problem (2.20), we do comparison just between IPOPTs, due to fact that PermonQP is not able to converge for structured instances of problem (2.20) at the moment of writing this report. From IPOPT versions, the one, which was configured with linear solver MUMPS performs faster. Considering sparse cases, we can see, that the most efficient performance is reached by IPOPT-pardiso. Since, properties of sparse case are copying the "traditional" FEM settings, the choice of solver based on this observation can be also applied there. Considering spatio-temporal settings of FEM, connections between space locations are usually considered. However, performance of PermonQP for half matrixes is showing significantly better results and we believe, that convergency issues will be solved in the mean time, further we investigate its scalability.

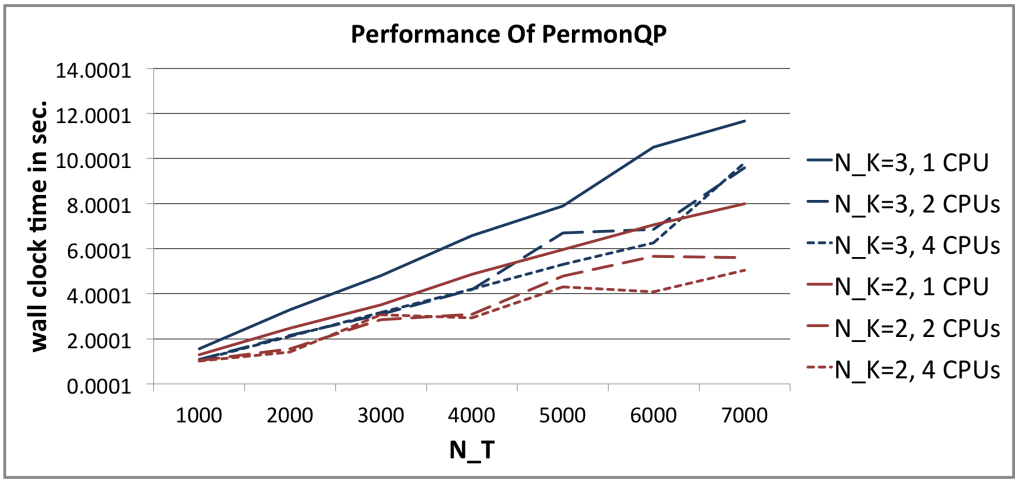

Figure B.2. Performance Of PermonQP For H-half

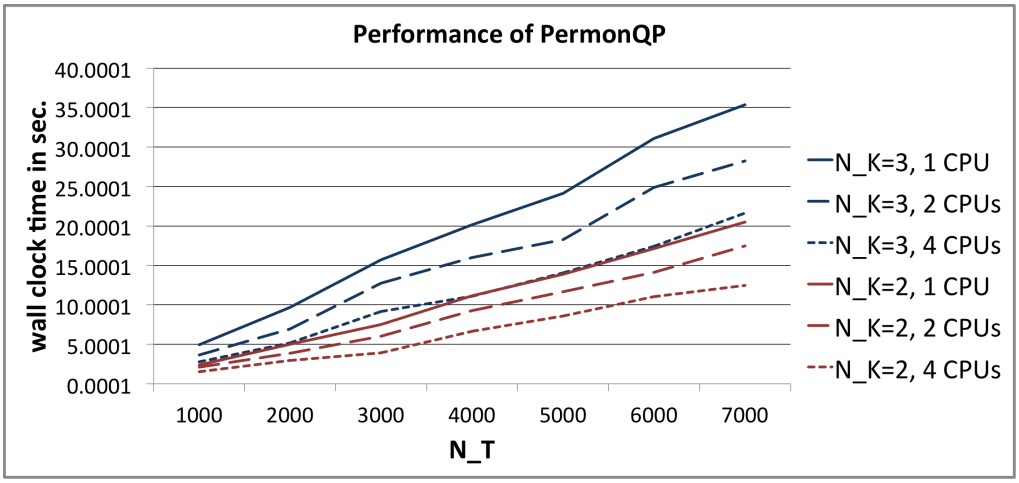

Figure B.3. Performance Of PermonQP For H-sparse

In order to show the effect of performance enhancement, we refer to relative speed up, which quantifies the improvement factor in processing speed as

$$S(p) = \frac{T(1)}{T(p)}, \tag{B.1}$$

where $T(1)$ is a execution time of on the single-processor and $T(p)$ is execution time on the multi-processor with p processors.

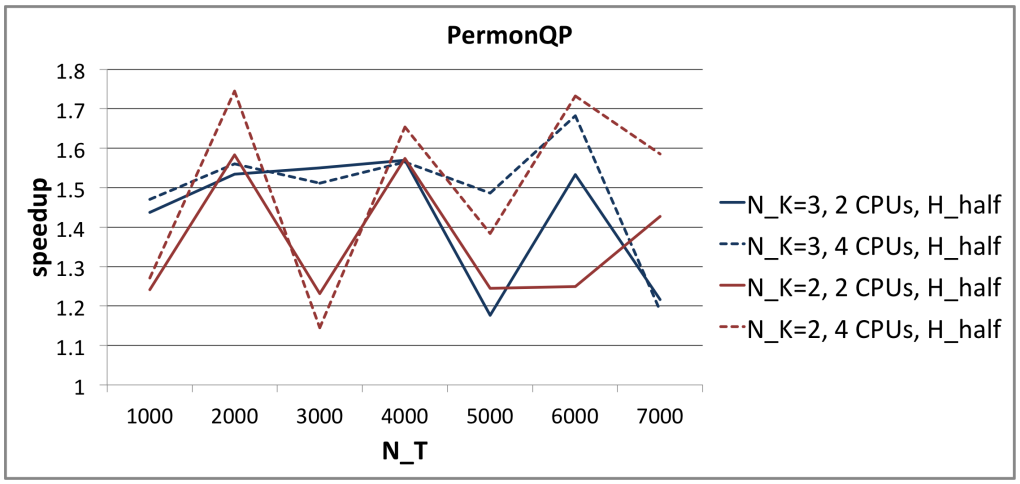

Figure B.4. Speedup For PermonQP For H-half

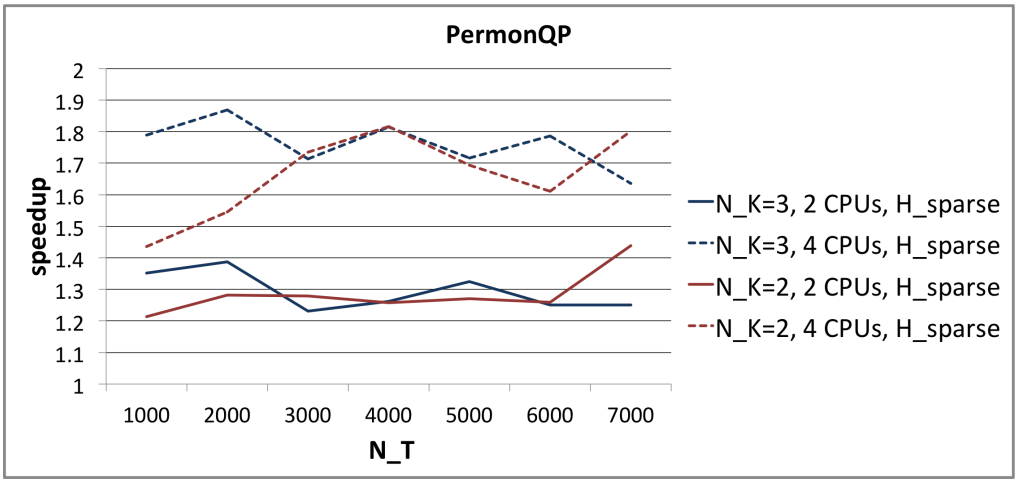

Figure B.5. Speedup For PermonQP For H-sparse

Figures B.4 and B.5 are showing, that the factor of speedup varies between 1.1 and 1.8 for half matrixes. For sparse matrixes, speedup obtained by 4 cores has factor almost 2, while by 2 cores it is around 1.3. Based on this results, we can show efficiency $E(p)$

$$E(p) = \frac{S(p)}{p}. \tag{B.2}$$

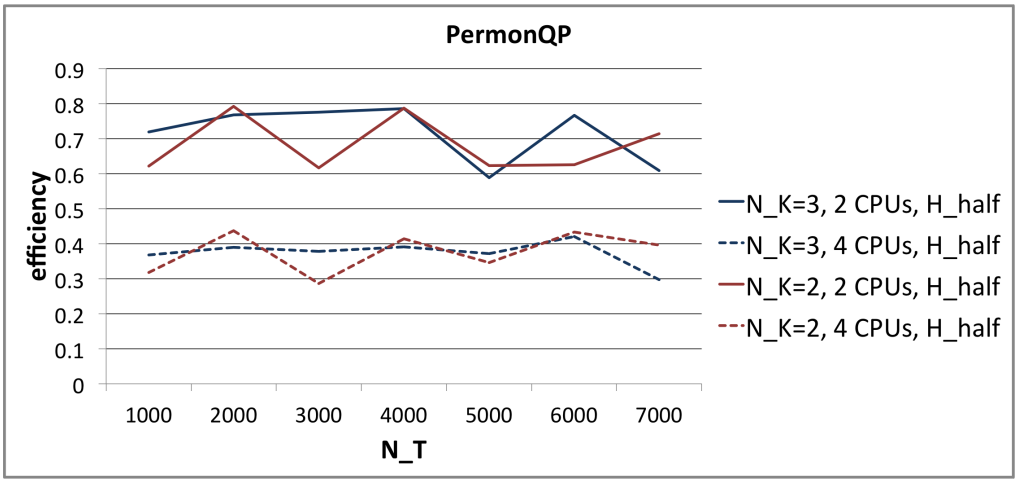

Figure B.6. Efficiency For PermonQP For H-half

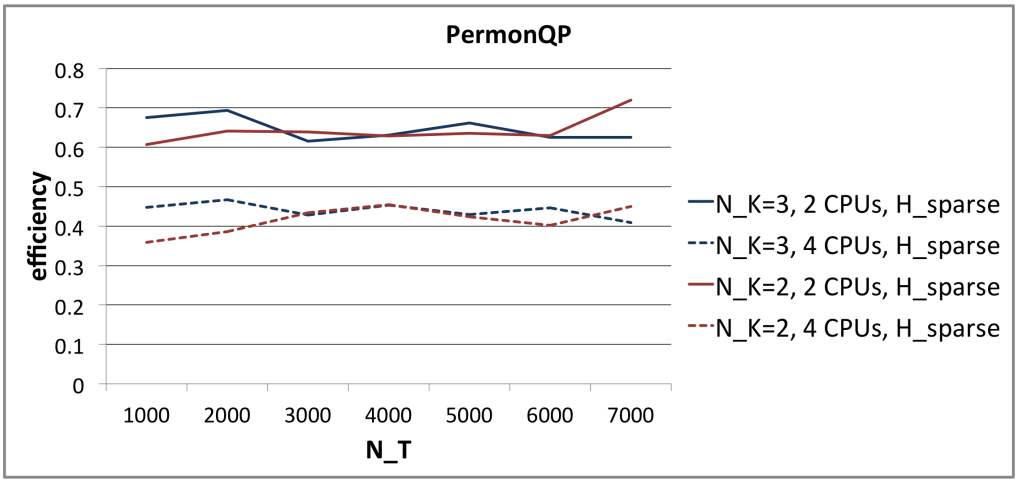

Figure B.7. Efficiency For PermonQP For H-sparse

According to the previous results, it is not surprising, that the efficiency obtained by 2 CPUs is higher than by 4 CPUs. As can be seen, while increasing the number of processors by factor two, the efficiency is decreasing by the same factor.
Therefore, by adding more processors, we can not expect much higher speed up. From our point of view, PermonQP is not showing promising scalability results, but before making premature conclusions, it is necessary to provide tests on machine with more cores.

Bibliography

[1] S. J. Wright and J. Nocedal, *Numerical optimization*, vol. 2. Springer New York, 1999.

[2] T. R. Karl and C. N. Williams Jr, "An approach to adjusting climatological time series for discontinuous inhomogeneities," *Journal of Climate and Applied Meteorology*, vol. 26, no. 12, pp. 1744–1763, 1987.

[3] B. Podobnik and H. E. Stanley, "Detrended cross-correlation analysis: a new method for analyzing two nonstationary time series," *Physical review letters*, vol. 100, no. 8, p. 084102, 2008.

[4] L. R. Rabiner and B.-H. Juang, "An introduction to hidden markov models," *ASSP Magazine, IEEE*, vol. 3, no. 1, pp. 4–16, 1986.

[5] M. P. Wand and M. C. Jones, *Kernel smoothing*. Crc Press, 1994.

[6] I. Horenko, "On clustering of non-stationary meteorological time series," *Dynamics of Atmospheres and Oceans*, vol. 49, no. 2, pp. 164–187, 2010.

[7] I. Horenko, "Finite element approach to clustering of multidimensional time series," *SIAM Journal on Scientific Computing*, vol. 32, no. 1, pp. 62–83, 2010.

[8] G. Dhatt, E. Lefrançois, and G. Touzot, *Finite element method*. John Wiley & Sons, 2012.

[9] D. M. Bates and D. G. Watts, *Nonlinear regression: iterative estimation and linear approximations*. Wiley Online Library, 1988.

[10] G. Buzsáki and A. Draguhn, "Neuronal oscillations in cortical networks," *science*, vol. 304, no. 5679, pp. 1926–1929, 2004.

[11] P. C. Hansen, "Regularization tools: A matlab package for analysis and solution of discrete ill-posed problems," *Numerical algorithms*, vol. 6, no. 1, pp. 1–35, 1994.

[12] C. W. Groetsch, "The theory of tikhonov regularization for fredholm equations of the first kind," 1984.

[13] P. J. Webster, G. J. Holland, J. A. Curry, and H.-R. Chang, "Changes in tropical cyclone number, duration, and intensity in a warming environment," *Science*, vol. 309, no. 5742, pp. 1844–1846, 2005.

[14] M. Sarandy, "Classical correlation and quantum discord in critical systems," *Physical Review A*, vol. 80, no. 2, p. 022108, 2009.

[15] L. Welch, "Lower bounds on the maximum cross correlation of signals (corresp.)," *IEEE Transactions on Information theory*, pp. 397–399, 1974.

[16] P. Metzner, L. Putzig, and I. Horenko, "Analysis of persistent nonstationary time series and applications," *Communications in Applied Mathematics and Computational Science*, vol. 7, no. 2, pp. 175–229, 2012.

[17] H. Akaike, "Likelihood of a model and information criteria," *Journal of econometrics*, vol. 16, no. 1, pp. 3–14, 1981.

[18] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492, University of California Press, 1951.

[19] R. Fletcher, "Practical methods of optimization wiley," 1987.

[20] D. G. Luenberger, *Introduction to linear and nonlinear programming*, vol. 28. Addison-Wesley Reading, MA, 1973.

[21] I. Gustafsson, "A class of first order factorization methods," *BIT Numerical Mathematics*, vol. 18, no. 2, pp. 142–156, 1978.

[22] S. Boyd, L. Xiao, and A. Mutapcic, "Subgradient methods," *lecture notes of EE392o, Stanford University, Autumn Quarter*, vol. 2004, pp. 2004–2005, 2003.

[23] M. Elad, B. Matalon, and M. Zibulevsky, "Coordinate and subspace optimization methods for linear least squares with non-quadratic regularization," *Applied and Computational Harmonic Analysis*, vol. 23, no. 3, pp. 346–367, 2007.

[24] J. E. Dennis, Jr and J. J. Moré, "Quasi-newton methods, motivation and theory," *SIAM review*, vol. 19, no. 1, pp. 46–89, 1977.

[25] R. Fletcher, "Conjugate gradient methods for indefinite systems," in *Numerical analysis*, pp. 73–89, Springer, 1976.

[26] G. H. Golub and R. S. Varga, "Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order richardson iterative methods," *Numerische Mathematik*, vol. 3, no. 1, pp. 157–168, 1961.

[27] S. C. Eisenstat, "Efficient implementation of a class of preconditioned conjugate gradient methods," *SIAM Journal on Scientific and Statistical Computing*, vol. 2, no. 1, pp. 1–4, 1981.

[28] D. S. Kershaw, "The incomplete choleskyâĂŤconjugate gradient method for the iterative solution of systems of linear equations," *Journal of Computational Physics*, vol. 26, no. 1, pp. 43–65, 1978.

[29] P. Wolfe, "Convergence conditions for ascent methods," *SIAM review*, vol. 11, no. 2, pp. 226–235, 1969.

[30] D.-H. Li and M. Fukushima, "A modified bfgs method and its global convergence in nonconvex minimization," *Journal of Computational and Applied Mathematics*, vol. 129, no. 1, pp. 15–35, 2001.

[31] J. J. Moré and D. J. Thuente, "Line search algorithms with guaranteed sufficient decrease," *ACM Transactions on Mathematical Software (TOMS)*, vol. 20, no. 3, pp. 286–307, 1994.

[32] R. Hooke and T. A. Jeeves, ""direct search"solution of numerical and statistical problems," *Journal of the ACM (JACM)*, vol. 8, no. 2, pp. 212–229, 1961.

[33] A. R. Conn, N. I. Gould, and P. L. Toint, *Trust region methods*, vol. 1. Siam, 2000.

[34] J. Nocedal and S. J. Wright, "Trust-region methods," *Numerical Optimization*, pp. 66–100, 2006.

[35] N. M. Alexandrov, J. E. Dennis Jr, R. M. Lewis, and V. Torczon, "A trust-region framework for managing the use of approximation models in optimization," *Structural Optimization*, vol. 15, no. 1, pp. 16–23, 1998.

[36] M. A. Noor, "Merit functions for general variational inequalities," *Journal of Mathematical Analysis and Applications*, vol. 316, no. 2, pp. 736–752, 2006.

[37] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Motivation and global convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 1–31, 2005.

[38] P. Tseng, "Merit functions for semi-definite complemetarity problems," *Mathematical Programming*, vol. 83, no. 1-3, pp. 159–185, 1998.

[39] M. Celis, J. Dennis, and R. Tapia, "A trust region strategy for nonlinear equality constrained optimization," *Numerical optimization*, vol. 1984, pp. 71–82, 1985.

[40] A. Wächter and L. T. Biegler, "Global and local convergence of line search filter methods for nonlinear programming," *Pittsburgh„ USA: Department of Chemical Engineering, Carnegie Mellon University. Technical Report CAPD B-01-09 Available on http://www. optimization-online. org/DB_HTML/08/367. html*, 2001.

[41] E. R. Panier and A. L. Tits, "Avoiding the maratos effect by means of a nonmonotone line search i. general constrained problems," *SIAM Journal on Numerical Analysis*, vol. 28, no. 4, pp. 1183–1195, 1991.

[42] R. Fletcher, *Second order corrections for non-differentiable optimization*. Springer, 1982.

[43] J. F. Bonnans, E. R. Panier, A. L. Tits, and J. L. Zhou, "Avoiding the maratos effect by means of a nonmonotone line search. ii. inequality constrained problems-feasible iterates," *SIAM Journal on Numerical Analysis*, vol. 29, no. 4, pp. 1187–1202, 1992.

[44] R. Glowinski and P. Le Tallec, *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*, vol. 9. SIAM, 1989.

[45] J. M. Martinez, "Algencan." `http://openopt.org/ALGENCAN`.

[46] N. I. Gould, "Galahad." `http://www.galahad.rl.ac.uk/authors.html`.

[47] A. Conn, "Lancelot." `http://www.numerical.rl.ac.uk/lancelot/blurb.html`.

[48] M. Saunders, "Minos." http://www.sbsi-sol-optimize.com/manuals/Minos%20Manual.pdf.

[49] Ben-Tal, "Pennon." http://web.mat.bham.ac.uk/kocvara/pennon/.

[50] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

[51] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Knitro: An integrated package for nonlinear optimization," in *Large-scale nonlinear optimization*, pp. 35–59, Springer, 2006.

[52] "Apopt." http://apopt.com/index.php.

[53] "Bqpd." http://tomopt.com/tomlab/products/minlp/solvers/bqpd.php.

[54] "Conopt." http://www.conopt.com/.

[55] "Knitro." http://www.ziena.com/docs/Knitro90_UserManual.pdf.

[56] "Lindo." http://www.lindo.com/index.php?option=com_content&view=article&id=28&Itemid=1.

[57] A. Griewank, "Lrambo." http://www.matheon.de/research/show_project.asp?id=40.

[58] K. Holmstrom, "Npsol." http://tomopt.com/docs/TOMLAB_NPSOL.pdf.

[59] P. E. Gill, "Qpopt." http://web.stanford.edu/group/SOL/guides/qpopt.pdf.

[60] Gill, "Snopt." http://www.sbsi-sol-optimize.com/manuals/SNOPT%20Manual.pdf.

[61] "Filterqp." http://www.mcs.anl.gov/~leyffer/solvers.html.

[62] Y. Nesterov, A. Nemirovskii, and Y. Ye, *Interior-point polynomial algorithms in convex programming*, vol. 13. SIAM, 1994.

[63] S. J. Wright, *Primal-dual interior-point methods*. Siam, 1997.

[64] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[65] E. De Klerk, *Aspects of semidefinite programming: interior point algorithms and selected applications*, vol. 65. Springer Science & Business Media, 2002.

[66] H. Wei, H. Sasaki, J. Kubokawa, and R. Yokoyama, "An interior point nonlinear programming for optimal power flow problems with a novel data structure," *Power Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 870–877, 1998.

[67] A. V. Fiacco and G. P. McCormick, *Nonlinear programming: sequential unconstrained minimization techniques*, vol. 4. Siam, 1990.

[68] Benson, "Dsdp." http://www.mcs.anl.gov/hs/software/DSDP/.

[69] Kawajir, "Ipopt." http://www.coin-or.org/Ipopt/documentation/.

[70] Vanderbei, "Loqo." http://www.princeton.edu/~rvdb/tex/loqo/loqo405.pdf.

[71] Gondzio, "Oops." http://www.maths.ed.ac.uk/~gondzio/parallel/solver.html.

[72] "Qpsol." http://www.worhp.de/content/qpsol.

[73] "Xpress." http://www.solver.com/xpress-solver-engine.

[74] M. Saunders, "Lusol: Sparse lu for ax = b."

[75] I. S. Duff and J. K. Reid, "The design of ma48: a code for the direct solution of sparse unsymmetric linear systems of equations," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 2, pp. 187–226, 1996.

[76] I. S. Duff, "Ma57-a new code for the solution of sparse symmetric definite systems," tech. rep., CM-P00045911, 2002.

[77] P. R. Amestoy, I. S. Duff, J.-Y. LâĂŹExcellent, and J. Koster, "Mumps: a general purpose distributed memory sparse solver," in *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, pp. 121–130, Springer, 2001.

[78] O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker, "Pardiso: a high-performance serial and parallel sparse linear solver in semiconductor device simulation," *Future Generation Computer Systems*, vol. 18, no. 1, pp. 69–78, 2001.

[79] J. W. Demmel, "Superlu users' guide," *Lawrence Berkeley National Laboratory*, 2011.

[80] T. A. Davis, "Algorithm 832: Umfpack v4. 3—an unsymmetric-pattern multifrontal method," *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 2, pp. 196–199, 2004.

[81] M. Saunders, "Symmlq: Sparse symmetric equations."

[82] C. C. Paige and M. A. Saunders, "Solution of sparse indefinite systems of linear equations," *SIAM Journal on Numerical Analysis*, vol. 12, no. 4, pp. 617–629, 1975.

[83] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.

[84] F. Zhang, *The Schur complement and its applications*, vol. 4. Springer Science & Business Media, 2006.

[85] J. R. Bunch and B. N. Parlett, "Direct methods for solving symmetric indefinite systems of linear equations," *SIAM Journal on Numerical Analysis*, vol. 8, no. 4, pp. 639–655, 1971.

[86] P. E. Gill, W. Murray, D. B. Ponceleon, and M. A. Saunders, "Solving reduced kkt systems in barrier methods for linear and quadratic programming," tech. rep., DTIC Document, 1991.

[87] W. P. Adams and H. D. Sherali, "A tight linearization and an algorithm for zero-one quadratic programming problems," *Management Science*, vol. 32, no. 10, pp. 1274–1290, 1986.

[88] "Cplex." http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[89] R. Wunderling, "Soplex: The sequential object-oriented simplex class library," 1997.

[90] "Gurobi." http://www.gurobi.com/.

[91] "Mosek." https://www.mosek.com/resources/doc.

[92] Gertz, "Ooqp." http://pages.cs.wisc.edu/~swright/ooqp/.

[93] R. Fourer, D. M. Gay, and B. Kernighan, *Ampl*, vol. 119. Boyd & Fraser, 1993.

[94] R. Fourer, D. M. Gay, and B. Kernighan, *Ampl*, vol. 119. Boyd & Fraser, 1993.

[95] Y. Saad, "Sparskit: A basic tool kit for sparse matrix computations," 1990.

[96] B. A. Murtagh and M. A. Saunders, "Minos 5. 0 user's guide," tech. rep., Stanford Univ., CA (USA). Systems Optimization Lab., 1983.

[97] D. M. Gay, "Hooking your solver to ampl," tech. rep., Technical Report 93-10, AT&T Bell Laboratories, Murray Hill, NJ, 1993, revised, 1997.

[98] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM journal on optimization*, vol. 12, no. 4, pp. 979–1006, 2002.

[99] M. Saunders, P. Gill, W. Murray, M. Wright, M. O'Sullivan, K. Eikland, Y. Zhang, N. Henderson, D. Ma, and S. Ax, "Lusol: Sparse lu for ax= b," 2009.

[100] D. A. Pierre, *Optimization theory with applications*. Courier Corporation, 2012.

[101] G. Quintana-Ortí, X. Sun, and C. H. Bischof, "A blas-3 version of the qr factorization with column pivoting," *SIAM Journal on Scientific Computing*, vol. 19, no. 5, pp. 1486–1494, 1998.

[102] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, *et al.*, *LAPACK Users' guide*, vol. 9. Siam, 1999.

[103] I. S. Duff and J. K. Reid, *MA27-a set of Fortran subroutines for solving sparse symmetric sets of linear equations*. UKAEA Atomic Energy Research Establishment, 1982.

[104] I. S. Duff, "Ma57—a code for the solution of sparse symmetric definite and indefinite systems," *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 2, pp. 118–144, 2004.

[105] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with pardiso," *Future Generation Computer Systems*, vol. 20, no. 3, pp. 475–487, 2004.

[106] A. Gupta, "Wsmp: Watson sparse matrix package (part-ii: direct solution of general sparse systems)," tech. rep., Citeseer, 2000.

[107] "Gurobi optimizer - the state-of-the-art." http://www.gurobi.com/pdfs/Gurobi-Corporate-Brochure.pdf.

[108] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, *et al.*, "Petsc users manual revision 3.5," 2014.

[109] B. Taylor, E. Miller, C. Farrington, M.-C. Petropoulos, I. Favot-Mayaud, J. Li, and P. A. Waight, "Autism and measles, mumps, and rubella vaccine: no epidemiological evidence for a causal association," *The Lancet*, vol. 353, no. 9169, pp. 2026–2029, 1999.

[110] X. S. Li, "An overview of superlu: Algorithms, implementation, and user interface," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 302–325, 2005.

[111] V. Hapla, M. Cermak, A. Markopoulos, and D. Horak, "Fllop: A massively parallel solver combining feti domain decomposition method and quadratic programming," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on*, pp. 320–327, IEEE, 2014.

[112] "it4i." `http://industry.it4i.cz/produkty/permon/qp/`.

[113] O. C. Zienkiewicz and R. L. Taylor, *The finite element method for solid and structural mechanics*. Butterworth-heinemann, 2005.

[114] A. Griewank and G. F. Corliss, *Automatic differentiation of algorithms: theory, implementation, and application*. Defense Technical Information Center, 1992.

[115] L. B. Rall, *Automatic differentiation: Techniques and applications*, vol. 120. Springer Berlin, 1981.

[116] J. Andersson, J. Åkesson, and M. Diehl, "Casadi: a symbolic package for automatic differentiation and optimal control," in *Recent Advances in Algorithmic Differentiation*, pp. 297–307, Springer, 2012.

[117] "autodiff." `http://www.autodiff.org/`.

[118] D. G. Altman and J. M. Bland, "Statistics notes: quartiles, quintiles, centiles, and other quantiles," *Bmj*, vol. 309, no. 6960, pp. 996–996, 1994.

[119] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.