A Solver for Non-Linear Optimization Problems with Box-Constraints

Matthias Mitterreiter

December 19, 2018

Abstract

In this work we present NOONTIME, a lightweight solver for non-linear optimization problems with box-constraints, based on Newton's method. It has minimal dependencies and is easy to use. The implementation was tested on a suitable subset of the CUTEst problem set and is compared to the open source solver library IPopt.

Contents

Chapter 1

Introduction

This work is about solving non-linear optimization problems. Mathematical optimization as a scientific field studies these problems and provides techniques to solve them. Among its numerous applications in engineering, economics and various other fields, many machine learning techniques rely heavily on optimization. For example, logistic regression requires the maximization of the likelihood function, support vector machines maximize the in-between margin of classes and the back-propagation algorithm for training deep neural nets requires the minimization of a loss function.

These functions are often non-linear or are defined over bounded subsets of $\mathbb{R}^n$. If a function of interest is twice differentiable, this allows to use a powerful technique called Newton's method to solve optimization problems of that kind.

This thesis is divided in two main parts. Chapter 2 introduces non-linear optimization problems with box-constraints and describes the theoretical background that enable the implementation of a solver for this kind of problem. Chapter 3 summarizes the results from testing NOONTIME on a subset of the CUTEst problems. For benchmarks we used the IPopt solver library and compared and evaluated the results.

Chapter 2

Theoretical Background

2.1 Definitions & Fundamentals

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function. We say f is *convex* if:

$$f((1-t)x + ty) \leq (1-t)f(x) + tf(y) \quad \forall x, y \in \mathbb{R}^n,\ t \in [0,1]$$

We say f is *strictly convex* if:

$$f((1-t)x + ty) < (1-t)f(x) + tf(y) \quad \forall x, y \in \mathbb{R}^n, x \neq y,\ t \in (0,1)$$

We say f is *strongly convex* with parameter $\alpha > 0$ if:

$$f((1-t)x + ty) \leq (1-t)f(x) + tf(y) - t(1-t)\alpha\|x-y\|^2 \quad \forall x, y \in \mathbb{R}^n, t \in [0,1]$$

We say $\tilde{x} \in \mathbb{R}^n$ is a *local minimum* of f if there exists a $\delta > 0$ with:

$$f(x) \geq f(\tilde{x}) \quad \forall x \in \{x \in \mathbb{R}^n : \|x - \tilde{x}\| < \delta\}$$

and $\tilde{x} \in \mathbb{R}^n$ is a *global minimum* of f if:

$$f(x) \geq f(\tilde{x}) \quad \forall x \in \mathbb{R}^n$$

We say a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is *positive definite* if

$$x^T A x > 0 \quad x \in \mathbb{R}^n \backslash \{0\}$$

and A is *positive semidefinite* if

$$x^T A x \geq 0 \quad x \in \mathbb{R}^n$$

We now present some important theorems, that we use throughout this thesis.

Theorem 1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function and let $\tilde{x} \in \mathbb{R}^n$ be a local minimum of f. Then $\tilde{x}$ is a global minimum of f over $\mathbb{R}^n$.*

Proof. Let $\tilde{x} \in \mathbb{R}^n$ be a local but not a global minimum of f. Therefore there exists a positive δ such that:

$$f(x) \geq f(\tilde{x}) \quad \forall x \in \mathbb{R}^n \text{ with } \|x - \tilde{x}\| < \delta$$

Now let x^* be a global minimum of f. Since f is convex, the following holds:

$$f((1-t)\tilde{x} + tx^*) \leq (1-t)f(\tilde{x}) + tf(x^*) < f(\tilde{x})$$

By choosing t such that $((1-t)\tilde{x} + tx^*) \in \{x \in \mathbb{R}^n : \|x - \tilde{x}\| < \delta\}$ we have a contradiction and therefore x^* does not exist. $\square$

Theorem 2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function. Then f is strongly convex if and only if there is a positive β such that*

$$x^T H(x) x \geq \beta x^T x \quad \forall x \in \mathbb{R}^n$$

Proof. See proof of [2, Thm. 3.2.14] □

Theorem 3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function. Then $H(x)$ is positive definite if and only if all its eigenvalues are positive.*

Proof.

$\Rightarrow$ Let $H(x)$ be positive definite and let λ be an eigenvalue of $H(x)$ with eigenvector $\hat{x}$.

$$H(x)\hat{x} = \lambda \hat{x} \Rightarrow \underbrace{\hat{x}^T H(x) \hat{x}}_{>0} = \lambda \underbrace{\hat{x}^T \hat{x}}_{>0} \Rightarrow \lambda > 0$$

$\Leftarrow$ Now let each eigenvalue $\lambda_i, i \in \{1, ..., n\}$ of $H(x)$ be positive. Since $H(x)$ is symmetric, there exits an orthogonal matrix V such that:

$$H(x) = V D V^T$$

where

$$D = diag(\lambda_1, ..., \lambda_n)$$

Now let $y \in \mathbb{R}^n$ be a non-zero vector and let $z = V^T y$. We can now write:

$$y^T H(x) y = z^T D z = \sum_{i=1}^{n} \lambda_i z_i^2$$

With y being non-zero and $z = V^T y$ being non-zero it follows that the sum above is positive and therefore $H(x)$ is positive definite.

□

Theorem 4. *Let A be a positive definite matrix. Then A is invertible.*

Proof. Let A be positive definite. By the invertible matrix theorem, we know that A is invertible iff the equation $Ax = 0$ has only the trivial solution. We write:

$$Ax = 0 \Rightarrow x^T A x = 0$$

Since A is positive definite, $x^T A x$ is positive for a non-zero vector x. Therefore $x^T A x = 0$ has only the trivial solution and it follows that A is invertible. □

2.2 Bounded strongly convex optimization problems

Consider the following optimization problem with twice differentiable, strongly convex objective function $f : \mathbb{R}^n \to \mathbb{R}$ and solution $\tilde{x}$:

$$\tilde{x} = \min_{x} \quad f(x) \tag{2.1a}$$

$$\text{s.t.} \quad l \leq x \leq b \tag{2.1b}$$

We call $l \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$ the lower and upper bounds on the variable $x \in \mathbb{R}^n$, respectively. In the following we consecutively describe the building blocks of a solver for this kind of problems. Later on we will generalize it to non-convex optimization problems. We begin with the general descent method for unconstrained optimization problems [3, Ch. 9.1]:

Algorithm 1: General descent method

```
  Input: starting point $x^{(0)} \in \mathbb{R}^n$
1 $k = 0$
2 while not converged do
3     Determine descent direction $\Delta x^{(k)}$
4     Line search. Chose a step size $\alpha^{(k)} > 0$
5     Update. $x^{(k+1)} = x^{(k)} + \alpha^{(k)} \Delta x^{(k)}$
6     k = k + 1
7 end
8 return $x^{(k)}$
```

2.3 Newton's method

Let f be the objective function and let $x^{(k)}$ be the current iterate with function value $f^{(k)}$, gradient $g^{(k)}$ and Hessian $H^{(k)}$. Since f is strongly convex, it follows that $H^{(k)}$ is positive definite and hence invertible by Theorem 4. We now consider the second order Taylor polynomial m_k of f in $x^{(k)}$, which we call model function:

$$m_k(x) = f^{(k)} + (g^{(k)})^T (x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T H^{(k)} (x - x^{(k)}) \tag{2.2}$$

The minimum $\tilde{x}^{(k)}$ of m_k is given by:

$$\begin{aligned} \nabla m_k(x) &= (g^{(k)}) + H^{(k)}(x - x^{(k)}) = 0 \\ \Rightarrow \tilde{x}^{(k)} &= x^{(k)} - (H^{(k)})^{-1} g^{(k)} \end{aligned}$$

from which we construct the search direction:

$$\Delta x^{(k)} = \tilde{x}^{(k)} - x^{(k)} = -(H^{(k)})^{-1} g^{(k)} \tag{2.3}$$

We call an algorithm of type (1) with descent direction (2.3) Newton's method. With $x^{(0)}$ being sufficiently close to $\tilde{x}$, the sequence of iterates converges to $\tilde{x}$ [5, Thm 3.5]. The positive definiteness property of the Hessian matrix is the key element to this method, since in this case the quadratic approximation (2.2) of f in $x^{(k)}$ is a strictly convex quadratic function with a unique solution $\tilde{x}^{(k)}$.

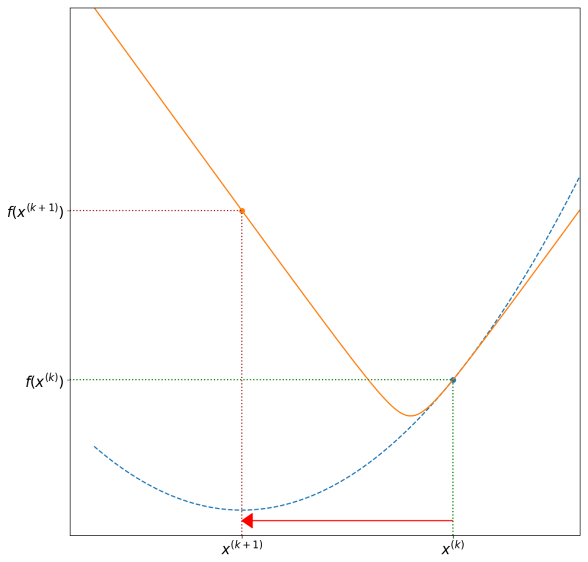
2.4 Line search

The line search is a method to choose a step length, that determines how far the algorithm moves the descent direction in any iteration.

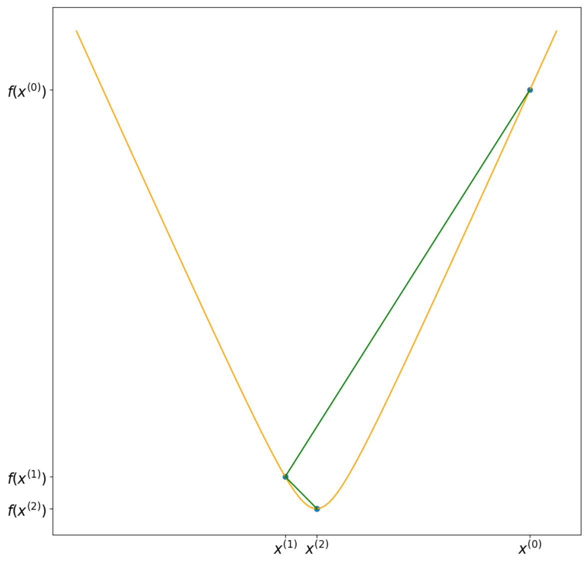
2.4.1 Motivational example

Consider the function:

$$f(x) = (1 + x^2)^{\frac{1}{2}} \quad \text{with domain } x \in [-10, 10]$$



(a) Model function $m_k(x)$ (blue) in $x^{(k)} = 2$ and direction Δx (red).



(b) With line search, the method converges with start value $x^{(0)} = 8$ in 2 steps to the minimum in $x^{(2)} = 0$.

Figure 2.1: Divergence (left) and convergence (right) of Newton's method for the function $f(x) = (1+x^2)^{\frac{1}{2}}$ (orange)

with:

$$f'(x) = x(1+x^2)^{-\frac{1}{2}}$$
$$f''(x) = (1+x^2)^{-\frac{3}{2}}$$

Since for $\beta = 1e-4$ the inequality $x^2 f''(x) \leq \beta x^2$ holds, it follows that $f''(x)$ is positive definite and therefore by Theorem 2 f is strongly convex. By using the descent direction (2.3) we get:

$$\begin{aligned}\Delta x^{(k)} &= -f''(x^{(k)})^{-1} f'(x^{(k)}) \\ &= -x^{(k)} - (x^{(k)})^3\end{aligned}$$

In the k-th step we can calculate the new iterate $x^{(k+1)}$ via:

$$x^{(k+1)} = x^{(k)} - x^{(k)} - (x^{(k)})^3 = -(x^{(k)})^3$$

From this follows:

$$f(x^{(k+1)}) = \begin{cases} f(x^{(k)}) & |x^{(k)}| = 1 \\ > f(x^{(k)}) & |x^{(k)}| > 1 \\ < f(x^{(k)}) & |x^{(k)}| < 1 \end{cases}$$

Here we can see, that Newton's method only converges for a start value with $|x^{(0)}| < 1$. In case $x^{(0)}$ is one, only the sign on the next iterate flips and in case $|x^{(0)}|$ is larger than one, our method diverges.
This is why Newton's method is called a *locally convergent* method. In Figure 2.1a it can be observed, that the model function m_k in $x^{(k)} = 2$ has its minimum in $x^{(k+1)}$, yet $f(^{(k+1)})$ is much larger than $f(x^{(k)})$.

2.4.2 Line search

In order to obtain a globally convergent method, a line search along $\Delta x^{(k)}$ can be used, such that

$$f(x^{(k)} + \alpha \Delta x^{(k)}) < f(x^{(k)}) \quad \forall x^{(k)} \in \mathbb{R}^n, \alpha > 0 \tag{2.4}$$

In order to guarantee (2.4) as well as a sufficient decrease of f, we enforce the strong Wolfe conditions [5, 3.7]:

$$f(x^{(k)} + \alpha^{(k)} \Delta x^{(k)}) \le f(x^{(k)}) + c_1 \alpha^{(k)} (g^{(k)})^T \Delta x^{(k)} \tag{2.5}$$

$$|\nabla f(x^{(k)} + \alpha^{(k)} \Delta x^{(k)})^T \Delta x^{(k)}| \le c_2 |(g^{(k)})^T \Delta x^{(k)}| \tag{2.6}$$

with

$$0 < c_1 < c_2 < 1$$

The full algorithm as well as implementation notes can be found in [5, Ch. 3]. By using the line search on our previous example, Newton's method converges even for start values $|x^{(0)}| > 1$ (see Figure 2.1b).

2.5 The gradient projection algorithm

Optimization problems with constraints of the form (2.1b) are called box-constraint problems. We use the gradient projection method presented in [4] to tackle these kind of problems.

For this, we define the active set A of a point x to be the set of indices, at which the components x_i of x lie on the bounds.

$$A(x) = \{i : x_i \in \{l_i, u_i\}\}$$

The set of free variables F of x is defined complementary as:

$$F(x) = \{i : l_i < x_i < u_i\}$$

The gradient projection is a two step algorithm, that generates a new descent direction $\Delta x^{(k)}$ by taking the bounds l and u on $x^{(k)}$ into account:

1. **Cauchy point computation**: Compute the Cauchy point $c^{(k)}$ (see section 2.5.1) to identify the set of active and free variables $A(c^{(k)})$ and $F(c^{(k)})$.
2. **Subspace minimization**: Solve a subspace minimization problem on the set of free variables $F(c^{(k)})$ with solution $s^{(k)}$.

Ultimately we calculate the search direction $\Delta x^{(k)} = s^{(k)} - x^{(k)}$ for the current iteration k.

2.5.1 Cauchy point computation

We start by projecting the direction of the steepest descent $-g^{(k)}$ onto the feasible region (2.1b), which can be expressed as a piecewise-linear-path:

$$x(t) = P(x^{(k)} - t g^{(k)}, l, u) \qquad t \ge 0 \tag{2.7}$$

with the projection function:

$$P(x, l, u)_i = \min\left(u_i, \max(x_i, l_i)\right) \tag{2.8}$$

The Cauchy point $c^{(k)}$ is then defined to be the minimum of the model function (2.2) along the piecewise-linear path (2.7):

$$\begin{aligned} t^* = \min_t \quad & m_k(x(t)) \\ \text{s.t.} \quad & t \geq 0 \end{aligned} \tag{2.9}$$

$$c^{(k)} = x(t^*)$$

For this purpose we calculate the set $T = \{t_i | i = 1, ..., n\}$ along the gradient direction.

$$t_i = \begin{cases} (x_i^{(k)} - u_i)/g_i^{(k)} & g_i^{(k)} < 0 \\ (x_i^{(k)} - l_i)/g_i^{(k)} & g_i^{(k)} > 0 \\ \infty & otherwise \end{cases} \tag{2.10}$$

By sorting the set T in increasing order, we obtain the ordered set $\{t^j : t^j \leq t^{j+1}, j = 1, .., n\}$. The Cauchy point $c^{(k)}$ can then be found by iteratively searching the intervals $[t^{j-1}, t^j]$ for t^*.

2.5.1.1 Interval search

In the following section we drop the outer index k, such that $g = g^{(k)}$, $H = H^{(k)}$ and define x^0 to be $x^{(k)}$. Superscripts denote the current interval.

The piecewise linear path (2.7) can now be expressed as

$$x_i(t) = \begin{cases} x_i^0 - tg_i & t \leq t_i \\ x_i^0 - t_i g_i & otherwise \end{cases}$$

Given the interval $[t^{j-1}, t^j]$ with descent direction:

$$d_i^{j-1} = \begin{cases} -g_i & t^{j-1} < t_i \\ 0 & otherwise \end{cases}$$

and breakpoints

$$\begin{aligned} x^{j-1} &= x(t^{j-1}) \\ x^j &= x(t^j) \end{aligned}$$

on line segment $[x^{j-1}, x^j]$, the model function (2.2) can be written as

$$\begin{aligned} m(x) = f &+ g^T(x^{j-1} + (t - t^{j-1})d^{j-1} - x^0) \\ &+ \frac{1}{2}(x^{j-1} + (t - t^{j-1})d^{j-1} - x^0)^T H (x^{j-1} + (t - t^{j-1})d^{j-1} - x^0) \end{aligned} \tag{2.11}$$

With $\Delta t = t - t^{j-1}$ and $z^{j-1} = x^{j-1} - x^0$, we can expand (2.11) and write it as a quadratic function in Δt:

$$\hat{m}(\Delta t) = f_{j-1} + f'_{j-1}\Delta t + \frac{1}{2} f''_{j-1}\Delta t^2 \tag{2.12}$$

where

$$\begin{aligned} f_{j-1} &= f + g^T z^{j-1} + \frac{1}{2}(z^{j-1})^T H z^{j-1} \\ f'_{j-1} &= g^T d^{j-1} + (d^{j-1})^T H z^{j-1} \\ f''_{j-1} &= (d^{j-1})^T H d^{j-1} \end{aligned}$$

which yields a minimum in $\Delta t^* = -f'_{j-1}/f''_{j-1}$. If $t^{j-1} + \Delta t^*$ lies on $[t^{j-1}, t^j)$, we found our Cauchy point c. Otherwise c lies at $x(t^{j-1})$ if $f'_{j-1} \geq 0$ and beyond or at $x(t^j)$ if $f'_{j-1} < 0$.

2.5.1.2 Updates

For exploring the next interval $[t^j, t^{j+1}]$, we set:

$$\begin{aligned}\Delta t^{j-1} &= t^j - t^{j-1}\\ x^j &= x^{j-1} + \Delta t^{j-1} d^{j-1}\\ z^j &= z^{j-1} + \Delta t^{j-1} d^{j-1}\end{aligned}$$

Since at least one variable became active it remains to update the search direction accordingly

$$d_i^j = \begin{cases} d_i^{j-1} & i \in F(x^{j-1}) \\ 0 & i \in A(x^{j-1}) \end{cases}$$

2.5.2 Subspace minimization

Given the Cauchy point $c^{(k)}$ in iteration k, we proceed with minimizing $m_k(\mathrm{x})$ over the set of free variables $F(c^{(k)})$. Let $Z \in \{0,1\}^{n \times |F(c^{(k)})|}$ be the matrix of unit vectors, that span the subspace of free variables at $c^{(k)}$ and let $\hat{d}$ be a vector of dimension $|F(c^{(k)})|$. Rewriting (2.2) in terms of $\hat{d}$ yields:

$$\hat{d}^* = \min_{\hat{d}} \quad \hat{m}_k(\hat{d}) = \hat{d}^T \hat{r} + \frac{1}{2} \tilde{H} \hat{d} + \gamma \tag{2.13a}$$

$$\text{s.t.} \quad l_i - c_i \leq \hat{d}_i, \tag{2.13b}$$

$$u_i - c_i \geq \hat{d}_i \tag{2.13c}$$

with reduced Hessian

$$\tilde{H} = Z^T H^{(k)} Z$$

and gradient

$$\hat{r} = Z^T (g^{(k)} + H^{(k)} (c^{(k)} - x^{(k)}))$$

The solution of the subspace minimization problem $s^{(k)} \in \mathbb{R}^n$ is now feasible to compute:

$$s_i^{(k)} = \begin{cases} c_i^{(k)} & i \notin F(c^{(k)}) \\ c_i^{(k)} + (Z\hat{d}^*)_i & i \in F(c^{(k)}) \end{cases}$$

This leads to the search direction

$$\Delta x^{(k)} = x^{(k)} - s^{(k)}$$

2.6 Termination conditions

Our algorithm stops, if the infinity norm of the projected gradient becomes sufficiently small:

$$||P(x^{(k)} - g^{(k)}, l, u) - x^{(k)}||_\infty < g_{tol} \tag{2.14}$$

Furthermore, we stop the process if the number of iterations k reaches a limit k_{max} or if the change of the objective function over two subsequent iterations is adequately small:

$$|f^{(k)} - f^{(k-1)}| < f_{tol} \tag{2.15}$$

2.7 Non-linear optimization problem

So far, we have required the objective function to be strongly convex. In this case, its Hessian is always positive definite and any local minimum is a global one. Since we also want to solve non-convex problems, we now drop the strong convexity condition.

2.7.1 Non-linear optimization problem

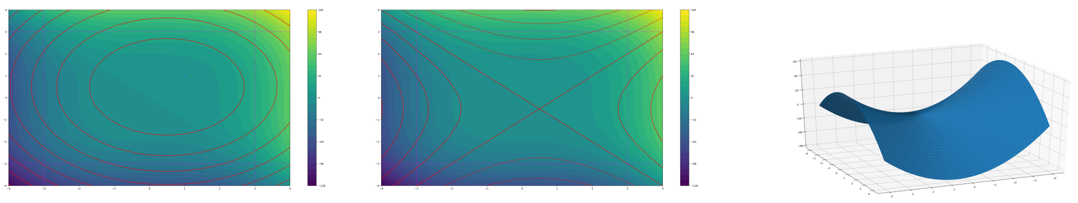
Consider the optimization problem of Section (2.2):

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & l \le x \le b \end{aligned} \tag{2.16}$$

We call (2.16) a non-linear optimization problem if the function $f : \mathbb{R}^n \to \mathbb{R}$ is non-linear. By necessity, we still require f to be twice differentiable.

By dropping the condition of strong convexity, we lose the guarantee that a local minimum is a global one. As a further consequence, the Hessian matrix is not necessarily positive definite. In this case, the quadratic approximation (2.2) of f in $x^{(k)}$ is not strictly convex and the descent direction (2.3) might not exist.

2.7.1.1 Example



(a) For $x = (0,0)$, $H(x)$ is positive definite and the model function (red) is strongly convex and has a unique minimum.

(b) For $x = (1,-1)$, $H(x)$ is indefinite and the model function (red) is not convex and does not have a minimum.

(c) 3D visualization of the model function for $x = (1,-1)$

.

Figure 2.2: Contour plots for problem (2.17) with bounds colored mangenta and the model function for different x colored in red.

As an example, consider the following optimization problem with non-convex objective function $f : \mathbb{R}^2 \to \mathbb{R}$:

$$\begin{aligned} \min_{x} \quad & f(x) = x_1^3 + x_2^3 \\ \text{s.t.} \quad & -3 \le x_1 \le 3, \\ & -3 \le x_1 \le 3 \end{aligned} \tag{2.17}$$

with:

$$g(x) = (3x_1^2, 3x_2^2)^T \qquad H(x) = \begin{pmatrix} 6x_1 & 0 \\ 0 & 6x_2 \end{pmatrix}$$

Depending on the value of x, the Hessian matrix $H(x)$ has different properties:

- For $x_1 = 1, x_2 = 0$, the Hessian is singular.
- For $x_1 = -1, x_2 = -1$ the Hessian is negative definite and since $\Delta x^T g > 0$, Δx is not a descent direction.

- For $x_1 = 1, x_2 = 1$ the Hessian is positive definite.
- For $x_1 = 1, x_2 = -1$ the Hessian is indefinite and since $\Delta x^T g = 0$, Δx is not a descent direction.

To overcome this obstacle, the Hessian matrix can be replaced by a positive definite one. In our case, we modify the Hessian and continue the process of minimizing the objective function with the modified Hessian.

2.7.2 Hessian Modification

The modification of the Hessian can be done in various ways [5, Ch. 3.4]. Our approach performs a spectral shift of the Hessian in case that it is not positive definite.

Let λ_{min} be the smallest eigenvalue of $H^{(k)}$ and let δ be a chosen lower bound for the eigenvalues of the modified Hessian. We can than calculate the modification parameter ω as follows:

$$\omega = \max\big(0, \delta - \lambda_{min}\big) \quad \delta \in \mathbb{R}^+$$

The modified Hessian is obtained by

$$\hat{H}^{(k)} = H^{(k)} + \omega\mathbb{I}$$

where all eigenvalues of $\hat{H}^{(k)}$ are all greater or equal to δ. It follows that $\hat{H}^{(k)}$ is positive definite.

Substituting the Hessian in the previous sections by its modification, all results generalize to non-convex functions. Furthermore, it can be shown that direction $\Delta x^{(k)}$ with $\hat{H}$ is always a descent direction.

Chapter 3

Measurements and Results

3.1 Measurements

We tested our implementation on a subset of the CUTEst problem set. For this we selected only bounded and unbounded problems with variable-size $2 \leq n \leq 5000$. In total 309 problems were used for testing. Throughout the test runs, the following solver configuration was employed

$$f_{tol} = 1e^{-8} \text{ (see (2.15))} \qquad g_{tol} = 1e^{-8} \text{ (see (2.14))}$$

We set the limit for the number of iterations at

$$k_{max} = 5000$$

and confined the CPU runtime of the solving process to 360 seconds. The variables of interest were:

- **Success**: Was the problem solved or unsolved.
- **Function value**: The objective function value of the last iterate, which is the minimum if the solver terminated successfully, or the last iterate when the solver was aborted due to the time cap, or the limit on the number of iterations.
- **Iterations**: Number of iterations performed until the solver terminated.
- **Message** The result message of the solver.

The summary of the NOONTIME test run is listed in Table 3.1.

For comparison, we used the open source solver library IPopt (version 3.12.5) [1]. IPopt as part of the COIN-OR initiative is written in C++ and primarily optimized for large-scale optimization problems. It was initially released in 2005 and since then steadily improved. It is well recognized in both academics and industry [1]. This and the fact that IPopt uses Newton's method, makes it a good competitor for NOONTIME. We ran IPopt with its default configuration on the same problem set. A summary of the Ipopt test run is listed in Table 3.2.

The results of all the test runs for both, NOONTIME and IPopt can be found in Table **??**.

For comparing the results from both of the solvers we use the *relative solver error*. Here we introduce the *relative solver error* on the values of the objective functions, but it is similarly defined and used for the number of iterations. Given a problem from our test set with results for the objective function value from NOONTIME, f_{nt}, and from IPopt, f_{opt}. Since the absolute error of both results $|f_{nt} - f_{pt}|$ depends heavily on the problem and does not represent the goodness of the final results very well, we define the *relative solver error* for f_{nt} and f_{opt} as:

$$err_{rel}(i) = \frac{f_i - f_{min}}{|f_{min}| + 1} \qquad f_{min} = min(f_{opt}, f_{nt}) \quad i \in \{nt, opt\} \tag{3.1}$$

This allows us to classify the result. With ϵ being appropriately selected, we denote three classes as:

$$\begin{aligned} f_{opt} \ll f_{nt} \quad &\Leftrightarrow \quad f_{opt} < f_{nt} \text{ and } err_{rel}(nt) > \epsilon \\ f_{opt} \approx f_{nt} \quad &\Leftrightarrow \quad f_{nt} \leq f_{opt} \text{ and } err_{rel}(opt) < \epsilon \text{ or} \\ & \qquad\ f_{opt} \leq f_{nt} \text{ and } err_{rel}(nt) < \epsilon \\ f_{opt} \gg f_{nt} \quad &\Leftrightarrow \quad f_{opt} < f_{nt} \text{ and } err_{rel}(opt) > \epsilon \end{aligned} \tag{3.2}$$

Informally, these classes can be interpreted verbally as:

- $f_{opt} \approx f_{nt}$: IPopt and NOONTIME did equally well.
- $f_{opt} \ll f_{nt}$: IPopt did better than NOONTIME.
- $f_{opt} \gg f_{nt}$: NOONTIME did better than IPopt.

The same holds for the number of iterations of IPopt, $Iter_{opt}$, and NOONTIME, $Iter_{nt}$. For the evaluation of our results, the relative solver error was computed with the following parameters:

- Objective function value: $\epsilon = 1e^{-4}$
- Number of iterations: $\epsilon = 1$

In the full result Table **??**, we color-encoded these classes for both the number of iterations and the objective function values. Let $f_{opt} \geq f_{nt}$. Then the cell for f_{opt} is colored *dark green*. If $f_{opt} = f_{nt}$, then also the cell for f_{nt} is colored *dark green*. Otherwise if $f_{nt} \approx f_{opt}$, then it is colored *light green* and if $f_{opt} \ll f_{nt}$, it is colored *orange*. Informally, a cell is *dark green* if it holds the best result for this specific problem, it is *light green* if the result is similarly good as the best result and it is *orange* if it is sufficiently worse than the best result. The same rules apply to the columns of *#iter*.

NOONTIME

status	**code**	**count**	**reason**
solved	0	241	Optimal Solution Found.
unsolved	1	16	Maximum number of iterations exceeded.
	2	40	Timeout after 360 seconds.
	5	9	Invalid iterate encountered ($f^{(k+1)} > f^{(k)}$).
	6	1	Overflow encountered in double scalars.
	7	1	Eigenvalues did not converge.
	8	1	Invalid search direction encountered ($g^T \Delta x > 0$).
	Σ	309	

Table 3.1: Result breakdown of running NOONTIME on the CUTEst test set.

IPopt

status	code	count	reason
solved	0	290	Optimal Solution Found.
unsolved	1	11	Maximum Number of Iterations exceeded.
	2	5	Timeout after 360 seconds.
	3	1	Invalid number in NLP function or derivative detected.
	4	2	Error in step computation.
	Σ	309	

Table 3.2: Result breakdown of running IPopt on the CUTEst test set.

Problems solved by NOONTIME and IPopt

	Function value			
Iterations	$f_{opt} \ll f_{nt}$	$f_{opt} \approx f_{nt}$	$f_{opt} \gg f_{nt}$	Σ
$Iter_{opt} \ll Iter_{nt}$	4	27	0	31
$Iter_{opt} \approx Iter_{nt}$	6	160	3	169
$Iter_{opt} \gg Iter_{nt}$	7	24	4	35
Σ	17	211	7	235

Table 3.3: Comparing the results from all problems, that both, IPopt and NOONTIME solved.

3.2 Evaluation

On the given 309 problems, NOONTIME solved 241, which amounts a success rate of 77.99% and IPopt in comparison solved 290 problems, equivalent to 93.94%. The relationships of results regarding the success variable are listed in Table 3.4.

		NOONTIME		
		solved	**unsolved**	Σ
IPopt	**solved**	235	55	290
	unsolved	6	13	19
	Σ	241	68	309

Table 3.4: Results of testing NOONTIME and IPopt on the problem set.

3.2.1 Unsolved problems

For NOONTIME, the largest group of unsolved problems is the group with code number 2 where the solver process was aborted due to the cut-off time. It counts 40 problems in total. By comparison, for IPopt the same group contains only 5 members. This gap in performance can be explained easily: [1]

1. **Sparsity**: In contrast to IPopt, NOONTIME does not exploit sparsity structures in the Hessian matrix. Especially for problems with many variables, this slows down the solving process with expensive operations like eigenvalue computation or the solving of linear systems.
2. **Native Python**: The Cauchy point computation and the subspace minimization consist of many loops that are implemented in native Python which runs slower in comparison to a compiled version of the same.

The second largest group (code number 1) of unsolved problems with 16 members are the problems where the maximum number of iterations was exceeded. Here we are not much worse off than IPopt with 11 problems in this category. Exploratory test runs with modified solver configuration settings suggests, that the size of this group can be reduced by adjusting the solver configurations to the specific problems.

The three problems with exit code 6,7 and 8 in Table 3.1 could not be solved because of numerical issues in our implementation. The problems with exit code 5 in Table 3.1 could not be solved due to numerical issues in the *scipy* line search that we rely on.

3.2.2 Solved problems

As shown in Table 3.4 there are 235 problems that both IPopt and NOONTIME solved. We classified the results of these problems according to the criterion (3.2) which is summarized in Table 3.3.
From the 235 there are 191 problems or $81,3\%$ where NOONTIME was at least as good as IPopt, both in terms of the goodness of the minimum and the number of iterations. In contrast, for IPopt there are 197 problems or $83,9\%$, where it was at least as good as NOONTIME.

Now considering the 211 problems where $f_{opt} \approx f_{nt}$, we were interested in how fast (in terms of iterations) the two implementations converged. By correlating the relative number of problems solved with the relative overhead in terms of iterations as shown in in Figure 3.1, we conclude, that both solvers behave similarly regarding convergence speed.

[1]We note here, that we did not optimize for speed in terms of CPU time, which is also the reason why we did not measure and compare the runtimes in the results.

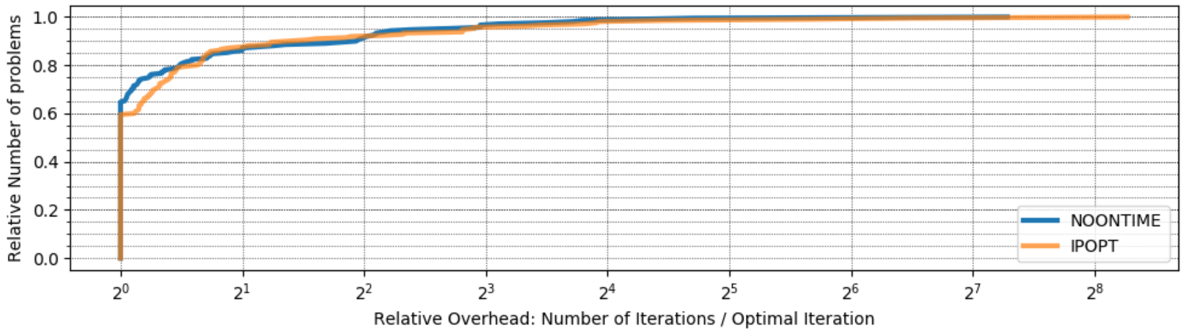


Figure 3.1: Convergence speed for the intersection of problems that were solved by IPopt and NOONTIME. The blue curve represents NOONTIME and the orange one IPopt.

Bibliography

[1] Ipopt, interior point optimizer. https://projects.coin-or.org/Ipopt.

[2] Alt, W. *Nichtlineare Optimierung: Eine Einführung in Theorie, Verfahren und Anwendungen.* vieweg studium; Aufbaukurs Mathematik. Vieweg+Teubner Verlag, 2011.

[3] Boyd, S., and Vandenberghe, L. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[4] Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput. 16*, 5 (Sept. 1995), 1190–1208.

[5] Nocedal, J., and Wright, S. J. *Numerical Optimization*, second ed. Springer, New York, NY, USA, 2006.