Checking for Collisions: Validating Optimality/Feasibility

Demet Demirkiran, Martyna Mikos
Edwin van der Vegt, Tahmina Begum, Casper Hogenboom

Group 14
Supervisor: Pieter Collins

October 2018

1 Context & Motivation

In numerical optimization there is a clear branch division between deterministic and non-deterministic optimization. Deterministic optimization focuses on finding a global optimization solution backed by theoretical grantees that the solution is indeed global (Neumaier, 2003). When this process is executed in finite time this is called rigorous optimization. For complex problems, significant computational costs are involved. Therefore non-deterministic optimization is introduced to decrease the computation time (Sergeyev, 2015). The solutions of this non-rigorous approach can exhibit different behavior on different runs. Non-rigorous algorithms are used to approximate a numerical solution, when finding the rigorous solution is computational too costly to obtain.

In the field of safety critical systems it is of high importance to detect certain malfunctions or failures before they occur. Such systems are used in a broad area of applications, including medical devices, train-aircraft-control, weapon and nuclear systems. In general, all new technological applications will have some form of safety critical system built in. The level of risk differs in these applications: financial loss or even loss of life can be a result of a wrong assessment in safety critical systems.

In safety critical systems the prior goal is to find an accurate solution in least computational time possible. This is an application where non-rigorous algorithms can be used to lower the computational cost. The non-rigorous algorithm should maintain high accuracy by validating the approximate solution. The context of this report is the verification of approximated solutions for non-rigorous optimization algorithms.

2 Research Problem

The proposed research problem: Can we validate a given approximation by proving the ex-

istence of a real solution 'nearby'?

The research problem will be further examined and divided into sub-research questions in Section 5.

3 Existing Methods

There exist two parts of optimization problem, in which the main distinction is between finding either a local or a global maxima/minima. Local optimization can be reduced to root finding by using Karush–Kuhn–Tucker (KKT) optimality conditions explained in Section 3.1. This project considers both optimization problems by finding the solution to algebraic equations.

For the case of the approximate root finding, the Newton Method will be used. Rigorous solution, on the other hand can be found by using Interval Newton Method. The challenges of the rigorous method lie in finding the initial interval, that contains the real solution and calculating how far from the solution is its approximation. Both methods will be elaborated further in Sections 3.3 and 3.4 respectively.

3.1 Optimality and feasibility Conditions

The Karush-Kuhn-Tucker Conditions (KKT) are first order conditions in nonlinear programming, that need to be satisfied in order to reach optimality. The KKT conditions are a powerful tool in mathematical optimization because they can be used for both equality and inequality constraint optimization.

Lagrange multipliers form the basis of the KKT conditions, and are used in constraint optimization (Collins, 2018). If there exists a stationary point, x^*, in a first-order system the following optimal condition holds,

$$\nabla f(x^*) - \lambda * \nabla g(x^*) = 0;\ g(x^*) = 0; \quad (1)$$

where λ is defined as the Lagrange multiplier. The Lagrange multiplier is the proportional constant between $\nabla f(x)$ and $\nabla g(x)$ which point in the same direction when optimality is reached.

The KKT conditions expand on the basis of the Lagrange conditions to satisfy for equality and inequality constraint optimization. Consider the constrained optimization problem

$$\begin{aligned} \min\ f(x) \text{ s.t. } g_j(x) = 0 \text{ for } j \in \varepsilon \\ \text{and } g_j(x) \geq 0 \text{ for } j \in \iota \end{aligned} \quad (2)$$

with j $\in \varepsilon$ equality constraints and j $\in \iota$ inequality constraints. Again, x^* is a regular minimiser of the function f(x). By KKT condition there exists λ^* such that

$$\begin{aligned} \nabla f(x^*) - \lambda * \nabla g(x^*) = 0; \\ g_j(x^*) = 0 \text{ for } j \in \varepsilon; \\ g_j(x^*)\lambda_j^* = 0 \text{ for } j \in \iota, \\ g_j(x^*), \lambda_j^* \geq 0 \text{ for } j \in \iota. \end{aligned} \quad (3)$$

Where the first condition checks whether the point is optimal, the second and third condition check feasibility. The last condition ensure the non-negativity of the variables.

3.2 Interval Arithmetic

Interval arithmetic is a technique for performing rigorous calculations. In this method a number is represented by some interval bounds i.e.
$[x] \in [\underline{x}; \overline{x}]$ and $[y] \in [\underline{y}; \overline{y}]$.

Real arithmetic operations like addition, subtraction, multiplication and division can also be performed on intervals (Jaulin, Kieffer, Didrit, & Walter, 2001).

Let $\odot \in \{+, -, \times, /\}$ be an operator that defines the operation performed on an interval. Then the operation between intervals can be defined as follows (Moore, 1966):

$$[x] \odot [y] = [\{x \odot y \in \mathbb{R} | x \in [x], y \in [y]\}]$$

These basic rules of interval arithmetic will be used in the implementation of the Interval Newton Method.

3.3 Newton Method

The Newton Method is one of the powerful techniques to solve numerical equations(Gil, Segura, & Temme, 2007). It is an iterative and sequential method. It rises from the theory of approximation and computes the approximation of x_n. where n=1.2.....The iteration will be terminated when $|x_n - x_{n+1}| < \epsilon$. The basic equation of this process is,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

3.4 Interval Newton Method

The Interval Newton method was introduced by Moore(1966), after which deep discussion led to refinement this method. Although there are many methods, which are used to determine zero of continuously nonlinear vectors-value functions, such as simplicial algorithm, continuous methods, secant method etc, but the interval Newton method is an excellent method, which is an important tool for the non-linear optimization problem. Since it can be used for computing all critical points of a function by applying the method to the Jacobean matrix, or for solving the Kuhn-Tucker or John condition in constrained optimization. The Interval Newton Method can determine all zeros over the whole area, delete zero . If there is any zero in a given area and gets an upper estimate of the approximation error (Moore, 1966). It overcomes the insurmountable difficulties of general real-valued iteration methods and the other general methods do not have these advantages mentioned above.
In the Interval Newton method, we considered X_0 as an initial estimate, it is also called box, whereas $X_0 = [\underline{X_0}, \overline{X_0}]$, $\underline{X_0}$ means the lower bound of the X_0 and $\overline{X_0}$ means the upper bound of the X_0. It is an iterative process. So each of the iterations, we can get a new estimate of the X_i, where $i = 1, 2, 3, 4....n$. Basically, the Interval Newton method is based

on two formula, one is Newton function and another is Newton iteration.
Newton function is:

$$N(X_n) = m(X_n) - \frac{F(m(X_n))}{F'(X_n)}$$

where,

$$m(X_n) = \frac{(\underline{X_n} + \overline{X_n})}{2}$$

Newton iteration is:

$$X_{n+1} = N(X_n) \cap X_n$$

3.5 Contractors

When using a set of equations solving f(x) = o in vector form, the corresponding constraint satisfaction problem H is defined as

$$H : (f(x) = 0, x \in [x]) \tag{4}$$

with solution set S

$$S = x \in [x] | f(x) = 0. \tag{5}$$

Contracting H means tightening the bounds of the solution set S without violating the solution. So by replacing [x] by a smaller domain [x'] such that solution set remains unchanged (Jaulin et al., 2001). Basic contractors that will be considered using in this paper; Gauss-Seidel and Krawczyk contractors.

4 Approach

The focus will be put on developing a software that determines and verifies the approximate solution based on information about the root provided by the rigorous algorithm. In other words, given the approximation to start with, the code will get the validation of such solution. In order to validate solutions the open-source C++ library Ariadne (`http://www.ariadne-cps.org`) will be used for its functionality for verifying solutions and ability to handle rigorous numerics. This library functionality is based on the theory of computational analysis for implementing the formal verification algorithm.

Therefore, the program will combine approximate and rigorous algorithms. This will be done in order to show that the approximate solution is sufficiently close to the actual solution (root or local min/max). The software will verify whether an approximate solution is in the neighbourhood of the true solution. The only objective isn't to prove that there's a root nearby, but also to calculate the distance or radius of the interval bounds, making the bound sufficiently tight. Specifically, the software will find the minimal value of the distance between the exact root and our expected root denoted, ϵ.

The milestone for the project will be analyzing and evaluating one-dimensional polynomials, therefore focusing on solving equations of a single variable. That means solving for $f(x) = 0$ and proving there is a solution around given $\tilde{x}$ with $f(\tilde{x}) \approx 0$, where $f : \mathbb{R} \to \mathbb{R}$.

The one dimensional case will provide a lot of insight on expanding onto multidimensional problems like: $f : \mathbb{R}^n \to \mathbb{R}^m$ for $m \leq n$.

5 Research Questions/Tasks

In light of the aforementioned theory, we expand our research problem to the following research questions. Note that research questions 4 and 5 will only be explored when the preceding ones have been answered.

1. How will we identify an initial estimator or initial box for the true solution?
2. What is an appropriate notion of "nearby"?
3. How can we find the approximations of one dimensional cases?
4. How can we use our method in multidimensional cases?
5. How can we improve the initial bounds on the solution?

6 Social Impact

In an effort to lower running times, computational mathematics mostly focuses on approximations rather than finding exact solutions. However for safety critical systems, inaccurate approximations can lead and have in some cases already led to dire outcomes. Therefore incorporating rigorous methods into safety-critical systems, would reduce the risk of disastrous collisions occurring, and thereby increase confidence in, and use of such systems.

7 Deliverables

The deliverables that we can be obtained from this project are in the form of an original source code which we can obtain from applying the formulas mentioned above specifically the Interval Method, Newton Method, and Interval Newton Method in the form of code in the language we chose. Another form of deliverable would be an user interface that could be implemented for anyone to be able to use to be able to see how the formulas could be able to applied to a certain interval. In short, what we could be achieved as a deliverable would be in the form of code or an application.

8 Requirements

The requirement this project has is to be able to find an exact solution for any approximation that would normally be generated from the application of formulas mentioned above on an approximate interval that is predefined by the person in question who is calculating or running the applicable code. Normally, such calculations reached to from said formulas all generate an approximation as the answer, which is why finding an exact solution is key for us. To be able to reach the exact solution to these formulas it has been decided that we can use the programming languages C++

or Python along with the Ariadne library or the supplementary tool INTLAB.

C++ is a general-purpose programming language. It has object-oriented and generic programming features, at the same time providing low-level memory manipulation features. While Python is used for general purpose programming which makes it especially readable. Either programming language is suitable to work in tandem with the Ariadne library. INTLAB could be used to implement the environment the formulas need in MATLAB.

8.1 ARIADNE

Ariadne is a tool for reachability analysis and model checking of hybrid systems. Additionally, it is a framework for rigorous computation featuring arithmetic, linear algebra, calculus, geometry, algebraic and differential equations, and optimization solvers. The Ariadne library is still in the process of completion and is constantly being updated. It endeavours to be able to give exact solutions to most approximate answers that complex formulas in mathematics usually give.

8.2 INTLAB

INTLAB is a toolbox used in MATLAB to be able to solve complex arithmetic's such as linear systems, interval arithmetic for real and complex data including vectors and matrices ,affine arithmetic including vectors and matrices etc. Such a toolbox could be used in place of the Ariadne library to provide answers for our optimality/feasibility problem along an interval. This method along with the Ariadne library will be taken into consideration to be used as we get more in depth with our research in how to provide an exact solution to approximate answers.

9 Time Management

The planning of the project is visualized in a Gantt-chart, Appendix A. Phase 1 was mainly used for literature review and installing all the necessary software. Besides that a deeper understanding of interval arithmetic was gained and the concept of Newtons optimization method. Various approximation and rigorous algorithms will be explored in phase 2. With the knowledge gained, a program will be made to verify the approximation solution for correctness. In phase 3 this program will be tested and verified, so that conclusion can be made in the report.

10 Risk Analysis & Contingency plans

There are a number of potential risks involved in this research. Firstly, we may run into a situation where there exists an exact solution, but the initial approximation is too far from it, i.e. it does not lie within the initial interval. Secondly, we may not be able to obtain a result in the multidimensional case. Thirdly,

when encountering unsolvable problems, our algorithm might not recognize them as such, but instead start a futile attempt at solving.

As a general rule we will start off with simpler problems and expand on the solutions to those, in order to minimize the risks laid out above. Specifically, limiting our analysis to the one-dimensional case initially, will help reduce complexity and allow us to produce running code earlier.s Furthermore, the probability of making errors in the underlying assumptions of our algorithm is greatly reduced due to the lower initial complexity. Therefore the probability of further mistakes and additional issues that rise up later on is lowered substantially.

References

Collins, P. (2018). *Mathematical optimization.*

Gil, A., Segura, J., & Temme, N. M. (2007). *Numerical methods for special functions* (1st ed.). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

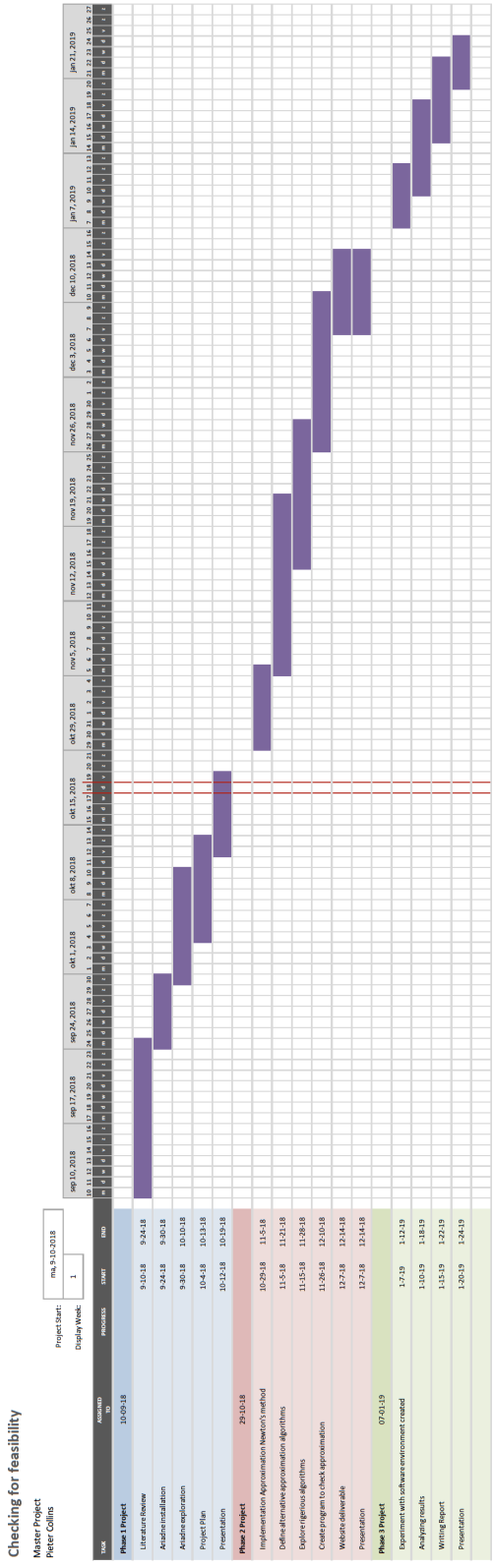
Jaulin, L., Kieffer, M., Didrit, O., & Walter, E. (2001). *Applied interval analysis.* Springer-Verlag London.

Moore, R. E. (1966). *Interval analysis.* Prentice-Hall, Inc.

Neumaier, A. (2003). Complete Search in Continuous Global Optimization and Constraint Satisfaction. Retrieved from `https://www.mat.univie.ac.at/~neum/ms/glopt03.pdf`

Sergeyev, Y. D. (2015). Communications in Nonlinear Science and Numerical Simulation. Retrieved from `http://si.deis.unical.it/~yaro/Smooth_N.pdf`

Appendix A


Checking for feasibility
Master Project
Pieter Collins
Project Start:
ma, 9-10-2018
Display Week:
1
sep 10, 2018
sep 17, 2018
sep 24, 2018
okt 1, 2018
okt 8, 2018
okt 15, 2018
okt 29, 2018
nov 5, 2018
nov 12, 2018
nov 19, 2018
nov 26, 2018
dec 3, 2018
dec 10, 2018
jan 7, 2019
jan 14, 2019
jan 21, 2019
TASK
ASSIGNED TO
PROGRESS
START
END
Phase 1 Project 10-09-18
Literature Review 9-10-18 9-24-18
Ariadne installation 9-24-18 9-30-18
Ariadne exploration 9-30-18 10-10-18
Project Plan 10-4-18 10-13-18
Presentation 10-12-18 10-19-18
Phase 2 Project 29-10-18
Implementation Approximation Newton's method 10-29-18 11-5-18
Define alternative approximation algorithms 11-5-18 11-21-18
Explore rigorious algorithms 11-15-18 11-28-18
Create program to check approximation 11-26-18 12-10-18
Website deliverable 12-7-18 12-14-18
Presentation 12-7-18 12-14-18
Phase 3 Project 07-01-19
Experiment with software environment created 1-7-19 1-12-19
Analysing results 1-10-19 1-18-19
Writing Report 1-15-19 1-22-19
Presentation 1-20-19 1-24-19