Tracking Optima in Dynamic Systems

REPORT FOR SIMULATION SCIENCES LAB

CATALIN DASCALIUC, GABRIEL FAUSTINI, HAMIDREZA PARIA,
SARTHAK KAPOOR, DR. JENS DEUSSEN (SUPERVISOR)

Contents

Foreword

This project report is a part of submission for Simulation Science Laboratory course given by Prof. Uwe Naumann at RWTH Aachen University. The project was supervised by Dr. Jens Deussen over a period of four months from November 2021 to February 2022.

The resulting software tracks local optima for a given objective function, where one variable in the objective function evolves with respect to a dynamic function specified by the user. The source code was written in C++ using dco/c++ library for automatic differentiation.

The code also uses an external package called Global Optimization Toolbox, developed by Dr. Jens Deussen, which returns a list of convex regions where local optima will be present.

1 Introduction

Mathematically speaking, optimization is utilizing mathematical methods and equations to determine the most efficient solution of a mathematical model which describes the problem of interest. In general, this mathematical model is a simplified modeling of a rigorous problem in practice such as minimizing the cost of an airline company, minimizing the costs of a production manufacturing industrial unit with maximizing of efficiency, design, and production rate.

The first step in optimization is defining the objective function of the problem; a quantitative measure of the performance of the system under study. This objective could be profit, time, potential energy, or any explicit continuous or discrete function of various input parameters. As an instance, for designing a process problem, it is feasible to decrease the investment costs while augmenting the operational costs, hence, an ongoing trade-off of opposing effects of parameters must be studied. [1]

The objective could be a scalar or vectorized explicit function equation or any discretized value of an independent simulation model depending on certain characteristics of the system, called variables or un-knowns. The aim in optimization is to find values of the variables that optimize the objective value. Often, the variables are restricted, or constrained, due to physical phenomena restriction (e.g., thermodynamic equilibrium), limited resources (available energy, manpower, etc.) or specified product quality. As a result, the final optimization model consists of

- An **objective** function
- A **mathematical model** for the description of the problem
- Additional **constraints** on variables

1.1 Classification of Optimization Problems

The general form of an optimization problem is:

$$\begin{gathered}\min_{\bar{x}\in\mathcal{B}} h(\bar{x}) \\ \text{s.th. } f(\bar{x}) = 0 \\ g(\bar{x}) \leq 0\end{gathered} \tag{1.1}$$

In which the $h(\bar{x})$ is the objective of the problem, the $f(\bar{x})$ is the equality constraint, and the $g(\bar{x})$ is demonstrating the inequality constraint. The solution must lie in feasible region $\mathcal{B}$ of the problem. Furthermore, the mathematical optimization problems are categorized with respect to the type of the objective function, the constraints and the variables. Constraints can be equality or inequality types (constraints are not necessarily present). The objective function and the constraints can be linear or non-linear. Variables can be discrete or continuous. The space of the degrees of freedom can be finite or infinite dimensional. For time dependent problems, the space of the degrees of freedom is infinite dimensional. The solution $x^*(t)$ is a function of time.

Based on various combinations of above-mentioned cases, the general classification of an optimization is as follows:

- **Linear Programming (LP):** The objective function and the constraints are linear.
- **Quadratic Programming (QP):** The objective function is quadratic, and the constraints are linear.
- **Nonlinear Programming (NLP):** The objective function or at least one constraint is nonlinear.
- **Integer Programming (IP):** All variables are discrete.
- **Mixed integer Programming (MIP):** Continuous and discrete variables occur. These problems can be linear (MILP) or non-linear (MINLP).
- **Dynamic Optimization (DO):** The solution is a function of time, in general the constraints are differential equations with a time-like independent variable.
- **Stochastic Optimization**

In addition, the terms Local and Global Optimization are used when describing such algorithms. Local optimization means the best solution in a subset of the possible local region around the given initial point, whereas global optimization aims at finding the best of all possible solutions.

A point $x^*$is a global minimizer if $f(x^*) \leq f(x) \, \forall \, x \in \mathcal{B}$. The $\mathcal{B}$ is defined as feasible set for variables. Moreover, A point x^* is a local minimizer if there is a neighborhood $\mathbb{N}$ of x^*, such that $f(x^*) \leq f(x) \, \forall \, x \in \mathbb{N}$. [2]

In principle, a global solution is always desired, and it is the main aim of any mathematician to develop an algorithm to find the global optimum variables for the problem of interest. However, one can only ensure the existence of a global minimum in a convex problem. The problem (is considered to be convex if the feasible set $\mathcal{B}$ is convex and if the objective function is convex in $\mathcal{B}$. But in practice, almost all problems are non-convex ones. In global optimization, two approaches are distinguished: the deterministic and the stochastic methods.

Branch and Bound (BB) is a class of deterministic methods for the global solution of linear and nonlinear mixed-integer programs. It guarantees to find an optimal solution to linear and convex nonlinear problems. For a non-convex nonlinear function with multiple local minima, finding the global solution is cumbersome. To overcome this issue, one way is to seek to find the convex regions in the domain of the target problem and solve the problem in those ranges.

1.2 Dynamic Optimization

Dynamic optimization problems involve time-dependent variables and time-dependent constraints. Due to the time-dependency, such problems are of infinite dimension. However, most practical methods for solving optimal control problems require a finite set of variables and constraints. First, a fixed time horizon of $\left[t_0; \; t_f\right]$ is considered. All variables $x(t)$ and $y(t)$ are dependent on time $t \in [t0; \, tf \,]$. The variables $x(t)$ are called differential variables or more general, state variables.

In this project, simulation of dynamical systems is illustrated by differential algebraic equations and the primal optimum variable of the system is found by optimization. For this purpose, after discretizing the dynamical equation with some numerical methods such as Explicit Euler, deterministic global optimization methods such as BB is applied to solve the optimization problem.

However, as described in detail in [3], finding the global optima of a non-convex function is an expensive procedure. For this purpose, in order to reduce the simulation time of solving optimization with a global algorithm, the local and global solver could be merged to solve the overall problem with acceptable solution accuracy. After discretizing the time domain to small ranges, local optima for the time dependent objective and constraint functions can be found utilizing a local solver as proceeding with time steps. With certain conditions, the global solver could be used to produce convex ranges in which the local solution would be equal to the global solution. As a result, with aid of the local optimization, the position of all local optima could be tracked over time and the minimum value of this local solutions in each domain is considered to be the global optimum.

In this project, the general dynamic problem is in the following form:

$$
\begin{gathered}
y(t) = arg\min_{\tilde{x}\in\mathcal{B}} h(t, x(t), y^*, p) \\
x\dot{(t)} = f(t, x(t), y(t), p) \\
y \in [y^L, y^U] \\
t \in [t_0, t_1]
\end{gathered}
\qquad (1.2)
$$

The state variables x(t) is uniquely determined by the Ordinary Differential Equation. Using static scheduling, the global search is applied once at start of the simulation, after certain number of time steps, or when the step has already reached the boundaries of the domains. In each time range, the local minima are calculated via Gradient Descent method. In the next sections, firstly, the theoretical background for solving the dynamical optimization problem will be studied and some toy problems will be introduced to test the developed algorithm. Moreover, the global solver of the project is developed and provided by Dr. Jens Deussen known as Global Optimization Toolbox (GLOPT). Subsequently, the layout of the overall software will be illustrated in section 3. At the end, the results for each toy problem are shown and the behavior of the overall solver on each toy problem is studied.

2 Theoretical Background

Given the problem proposed in the motivation, it can be divided into two simpler kinds of problem, that is a *Minimization Problem* and *Solving a Differential Equation*. In this chapter an overview of each individual problem will be presented and the chosen approach to solve it.

2.1 Global Optimization

According to [4] Optimization is a field of applied mathematics that deals with finding the extremal value of a function in a domain of definition, subject to various constraints on the variable values. When describing this problem in a mathematical form, one gets the following:

$$\begin{aligned} &minimize\ h(x) \\ &subject\ to\ f_i(x) \leq b_i, \qquad i = 1, \dots, m. \end{aligned} \tag{2.3}$$

Here the function $h : \boldsymbol{R}^n \to \boldsymbol{R}$ refers to the function from which it is wished to find the minimum, and it is usually referenced as objective function. Meanwhile the functions $f_i : \boldsymbol{R}^n \to \boldsymbol{R}$ are the (inequality) constrain function, and the constants $b_1, \dots., b_m$ are the bounds of the problem. Moreover, for the problems studied in the project it was assumed that the objective function is twice differentiable.

Having such a problem, the first analysis that must be made is if the problem at hand corresponds to a convex problem. In other words, if the functions correspond to a convex function and if the feasible points belong to a convex set. That is so because according to the fundamental result in convex analysis a locally optimal solution of a convex problem is also globally optimal [4]. Moreover, it is also what differentiates the problem between a Global Optimization Problem, or a simple local Optimization Problem, since the first is necessarily not globally convex and may have many different local convex regions with its respectively local optima.

Moreover, a convex set is defined according to [4] as a set $\boldsymbol{S} \subseteq \boldsymbol{R^n}$ for which any two points $x, y \in \boldsymbol{S}$ and the segment between them is wholly contained in $\boldsymbol{S}$, that is,

$$\forall l \in [0, 1] \quad (lx + (1 - l)y) \in \boldsymbol{S}. \tag{2.4}$$

And a function $f : \boldsymbol{X} \subseteq \boldsymbol{R}^n \to \boldsymbol{R}$ is convex if for all $x, y \in \boldsymbol{X}$ and for all $l \in [0,1]$ we have:

$$f(lx + (1 - l)y) \leq lf(x) + (1 - l)f(y). \tag{2.5}$$

As introduced in the first chapter the software GLOPT was used to define convex regions given a Global Optimization Problem. It takes as inputs the Optimization Function $h(x)$ and the global boundaries b_i and return all convex regions from the functions, and consequently all possible regions for the global optimum. From this point on the problem is divided in many smaller Local Optimization Problems, since for each region returned from the algorithm a single optimum is guaranteed to exist and a simple comparison between all local optima is enough to decide which in fact the global minimum.

2.2 Local Optimization

Given a twice differentiable convex function $h(x): \boldsymbol{R} \to \boldsymbol{R}$ with minimum at x^* there exists a sufficient condition for its optimality, namely if

$$\nabla h(x^*) = 0. \tag{2.6}$$

Thus, finding the value for $x^* \in \boldsymbol{R}$ for which this equation holds is the same as finding the solution of the minimization problem. Although sometimes it is possible to solve for x^*analiticaly, in most cases the problem must be solve iteratively by finding a sequence of points $x^{(0)}, x^{(1)}, \ldots \in \boldsymbol{R}\ \boldsymbol{dom}\ h$ with $f(x^k) \to f(x^*)$ when $k \to \infty$ [5]. For such algorithm that solves the optimization problem iteratively, a termination condition usually has the form $f(x^k) - f(x^{k-1}) < \epsilon$, where ϵ is a tolerance value.

In this project, the algorithm chosen to find the solution of the Local Optimization Problem was the Gradient Descent Algorithm. It consists, given an initial point $x^{(0)}$, in creating a series of points $x^{(k)}$ by the following iterative function

$$x^{(k+1)} = x^{(k)} - \alpha \nabla h(x^{(k)}), \tag{2.7}$$

where α is the step size and $\nabla h(x^{(k)})$ is the gradient of the function h applied in the last calculated $x^{(k)}$ points, and $k \in \boldsymbol{N}$ is the iteration index. By walking in every iteration in the opposite direction of the gradient, the iterator approaches the minimum point, which is guaranteed to exits following the assumptions that the h function is twice differentiable and convex. Furthermore, it is valid to comment on the sensibility of the method to the choice of the initial point $x^{(0)}$ and the iteration step size, since the convergence time varies and evolution of the $x^{(k)}$ series changes deeply in function of these two parameters.

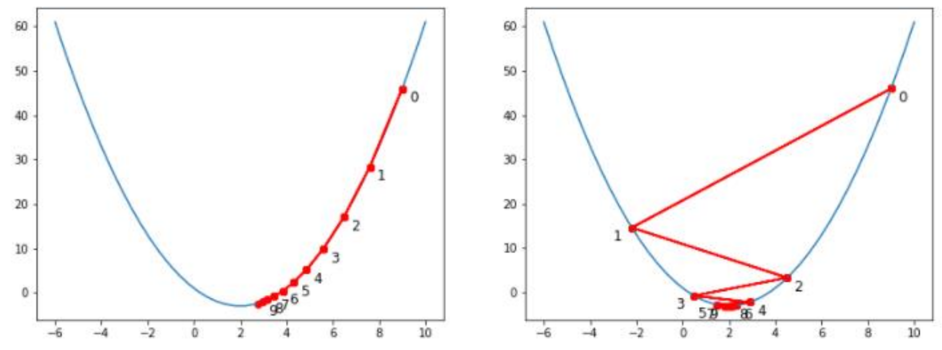

Figure 1 - 1D example of Gradient Descent Method. Red line shows the path generated by the iteration scheme.

In the Figure 1 two simple examples of the evolution of the steps from the Gradient Descent method is shown, to illustrate the sensibility of the method to two different evolution steps α.

2.3 Differential Algebraic Equation

The second part of the problem explored in this report is the solution of a *Differential Equation*. Although the problem can be extended to more complex forms of differential equations, it was considered for the project only *First Order Ordinary Differential Equation* on the form

$$\begin{aligned} &y' = f(t, y, x), \\ &\text{with } y(t_0) = y_0 \text{ and } t_0 < t < T \end{aligned} \tag{2.8}$$

Where $x \in \boldsymbol{R}^n$ is the coupling variable with the Minimization problem, and it is for the solution of the Differential Equation a passive variable and can be omitted. Furthermore, the function $f: \boldsymbol{R}^n \rightarrow \boldsymbol{R}^n$ is a function of $y \in \boldsymbol{R}^n$ and is derived with respect to the dynamic variable $t \in \boldsymbol{R}$. Since in many applications the dynamic variable t is time, it is convenient to call it as such. Finally, the initial point (t_0, y_0) is given and characterizes the initial condition. Problems like such are very well know and described in the literature and are also known as *Initial Value Problems.*

From all the possible ways to solve the Initial Value Problem, one of the first methods that was presented, and one of the simplest one, is the Forward Euler Method. This method, first introduced by Leonhard Euler himself in his book *Institutiones calculi integralis* consist in constructing a solution for the differential equation by Taylor expanding the solution y at the point $y_0 + h$, yielding

$$y(t_0 + h) = y(t_0) + hy'(t_0) + \cdots \tag{2.9}$$

With h the step size and the dots representing the higher-order terms. Additionally, one can substitute the definition of the differential equation into the equation, thus becoming

$$y(t_0 + h) = y(t_0) + hf(t_0, y_0) + \cdots \tag{2.10}$$

Finally, by neglecting the higher-order terms and generalizing the equation for any time t_n with $n \in \boldsymbol{N}$ the iteration index [6], one gets

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{2.11}$$

These results were obtained for 1D problem but can easily be expanded for more dimensions by doing the same steps for every new dimension, therefore, the authors leave it as an exercise for the reader.

2.3.1 Toy problems

During the development of the algorithm proposed, a few Toy Problems were created so that the program could be tested, and the results could be analyzed. These problems will be first introduced in this chapter and further discussed after that the program structure be presented.

2.3.1.1 Toy Problem 1

$$\begin{gathered} \dot{x}(t) = -(2 + y)x \\ h = (1 - y^2)^2 - (x - p)\sin\left(y\frac{\pi}{2}\right) \\ x(t),\ y(t),\ p \in \mathbb{R}^{\mathbf{1}} \\ x(0) = 1, \qquad p = 0.5 \;\; and \;\; t \in [0, 1]. \end{gathered} \tag{2.12}$$

The first problem consists of a differential equation and an optimization problem constrained to the $\boldsymbol{R^n}$. For this problem it is also expected that the solver finds a local minimum at $y = 1$ for $t = 0$ and that this minimum stays there until an event occurs at $t = -\log(p)/3$ when it is shifted to $y = -1$.

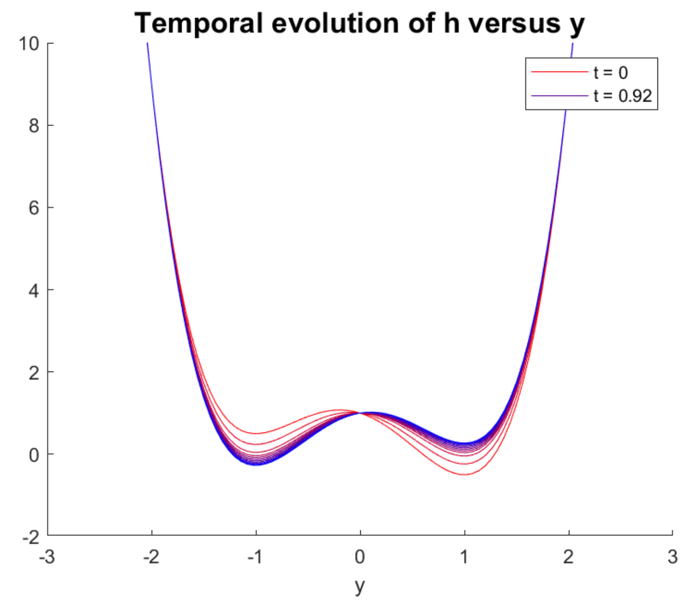

Figure 2 - Development of the objective function h as time evolves for Toy Problem 1

2.3.1.2 Toy Problem 2

$$\begin{gathered}\dot{x}(t) = y \\ h = (y - x)^2 + \sin(py) + 1 \\ x(t),\ y(t),\ p \in \mathbb{R}^{\mathbf{1}} \\ x(0) = 1, \qquad p = 3.0 \ \ and\ t \in [0, 2]\end{gathered} \tag{2.13}$$

The Toy Problem 2, just as the first problem, is also found in the one-dimensional case for both the differential equation and the optimality problem. Nevertheless, it does not present only one event of minimum change, but has multiples events one followed by the other, with the characteristic of its optimal point continually shifting to the right.

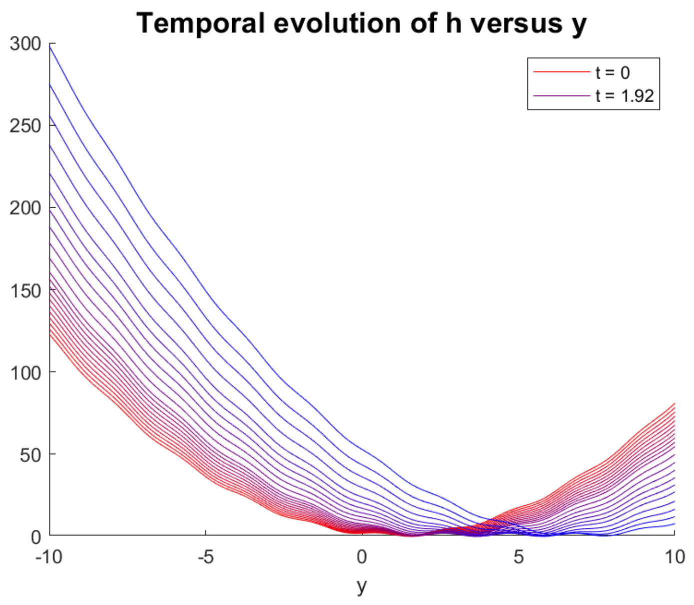

Figure 3 - Development of the objective function h as time evolves for Toy Problem 2

2.3.1.3 Toy Problem 3

$$
\begin{gathered}
\dot{x}(t) = -(2 + y_1 + y_2)p \\
h \ = (1 - y_1^2)^2 - (x - p)\sin\left(\frac{\pi y_1}{2}\right) \\
+(1 - y_2^2)^2 - (x - p)\sin\left(\frac{\pi y_2}{2}\right) \\
x(0) = 1, \qquad p = 0.5, \qquad t \in [0,\ 1] \\
x(t), \qquad p \in \boldsymbol{R}^{\mathbf{1}} \quad and \quad \mathrm{y} \in \boldsymbol{R}^{\mathbf{2}}
\end{gathered}
\tag{2.14}
$$

For Toy Problem 3 the dimensionality of the *objective function* was raised such that it now belongs to the $\boldsymbol{R}^2$. This can be seen by the fact that now $y \in \boldsymbol{R}^2$ and is expressed as $y = (y_1, y_2)$.

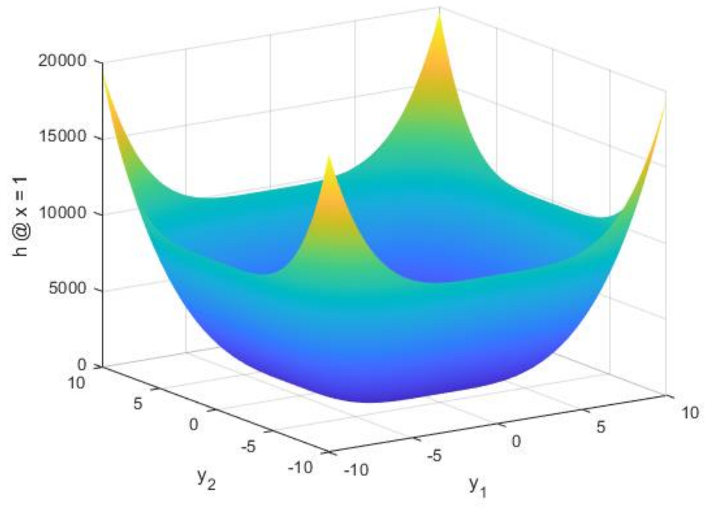

Figure 4 - Objective function at t=0 for Toy Problem 3

As depicted in Figure 4, the general form of the objective function can be seen.

2.3.1.4 Toy Problem 4

$$
\begin{gathered}
\dot{x}(t) = (y_1,\ y_2) \\
h \ = (1 - y_1^2)^2 - (x - p)\sin\left(\frac{\pi y_1}{2}\right) \\
+(1 - y_2^2)^2 - (x - p)\sin\left(\frac{\pi y_2}{2}\right) \\
x(0) = (1,\ 1), \qquad p = 3.0, \qquad t \in [0, 2] \\
x(t),\ y(t) \in \mathbb{R}^{\mathbf{2}} \ \ \text{and} \ \ p \in \mathbb{R}^{\mathbf{1}}
\end{gathered}
\tag{2.15}
$$

In the last toy problem explored in this report, the dimensionality of both functions was raised so that both $x(t),\ y(t) \in \boldsymbol{R}^{\mathbf{2}}$. Besides that, the general form of the objective function can be seen in Figure 5 and it presents a more irregular surface with multiple convex regions.

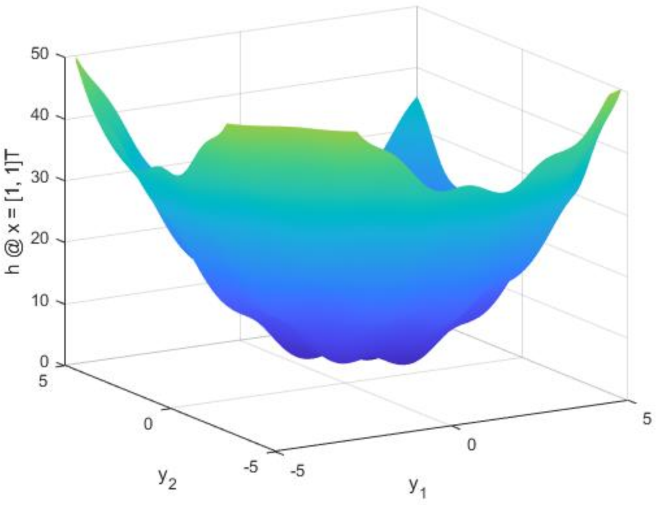

Figure 5 - Objective function at t=0 for Toy Problem 4

3 Software Layout

This section presents the architectural layout of the developed software. It starts with an overview of the main program flow and follows with a presentation of how to use the software.

3.1 Software Architecture

The developed software comprises of the main file that initializes the user input and supporting files that define the classes used for execution of the dynamical update of the function and the optimization routines. The class diagram of the source code organization is presented in Figure 6. It presents the main files used to run the developed software, the data flow, and the main methods of the specified classes.

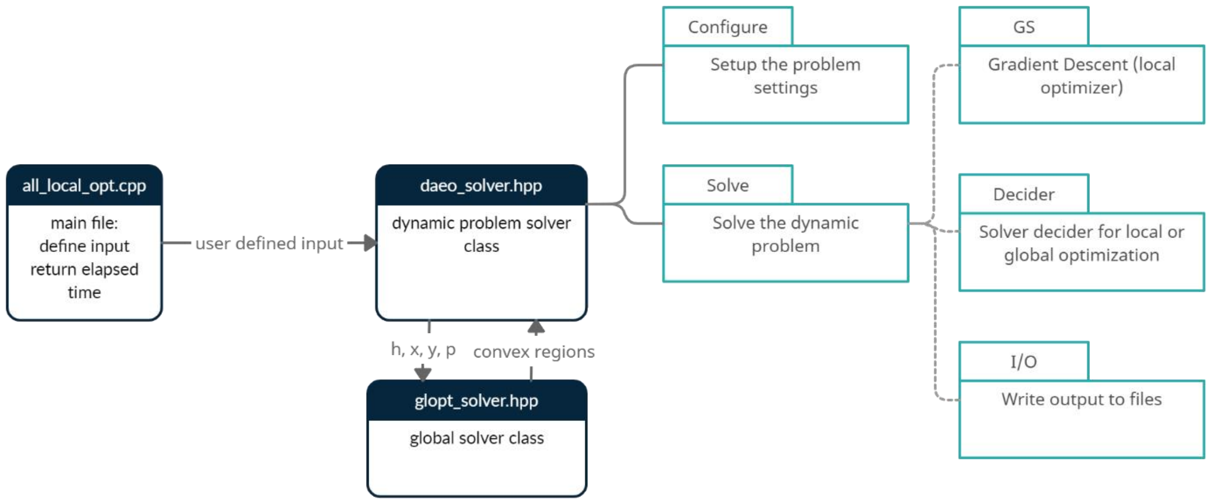
all_local_opt.cpp
main file: define input return elapsed time
user defined input
daeo_solver.hpp
dynamic problem solver class
h, x, y, p
convex regions
glopt_solver.hpp
global solver class
Configure
Setup the problem settings
Solve
Solve the dynamic problem
GS
Gradient Descent (local optimizer)
Decider
Solver decider for local or global optimization
I/O
Write output to files

Figure 6 - Class diagram of the source code

The main control flow of the software is presented in Figure 7. It represents the activity diagram of the underlining logics behind the software execution. The input is initialized based on the user specifications. It is followed by the initialization of the initial time step of the state variable x and the initial identification of the convex regions for the local optima at the current time step. Subsequently, the local solver is used to retrieve the actual global minimum within the convex regions, followed by execution of a checking procedure which decides whether the GLOPT should be run again if certain criteria are arising. If there is no need to run the update of the convex regions (which returns the program to the current time step with an additional execution of GLOPT), the state variable is dynamically updated with a forward Euler scheme. The values of the global minimum, state variable and optimization function value at location of the global minimum are written to the output file, until the final time step is reached.

The decider function activates due to the following criteria:

a) Static Scheduling: After certain number of time step iterations (*static_sched_interval*), which is specified by the user, the GLOPT is required to be run to recompute the convex regions where local minima are present.

b) Dynamic Scheduling: When the local minimum lies or leaves the boundaries of the previously computed convex regions, the GLOPT is requested to be run again before stepping to the next time iteration.

The GLOPT can be run at every time step if *static_sched_interval* is set to 0.

The local solver has been implemented based on the theoretical background of gradient descent method presented in section 2.2. Additionally, the convergence criterion is determined by a heuristically set tolerance in the code of the software and an internal loop of refining the tolerance up to a certain level has been also introduced to avoid stalling of the program.

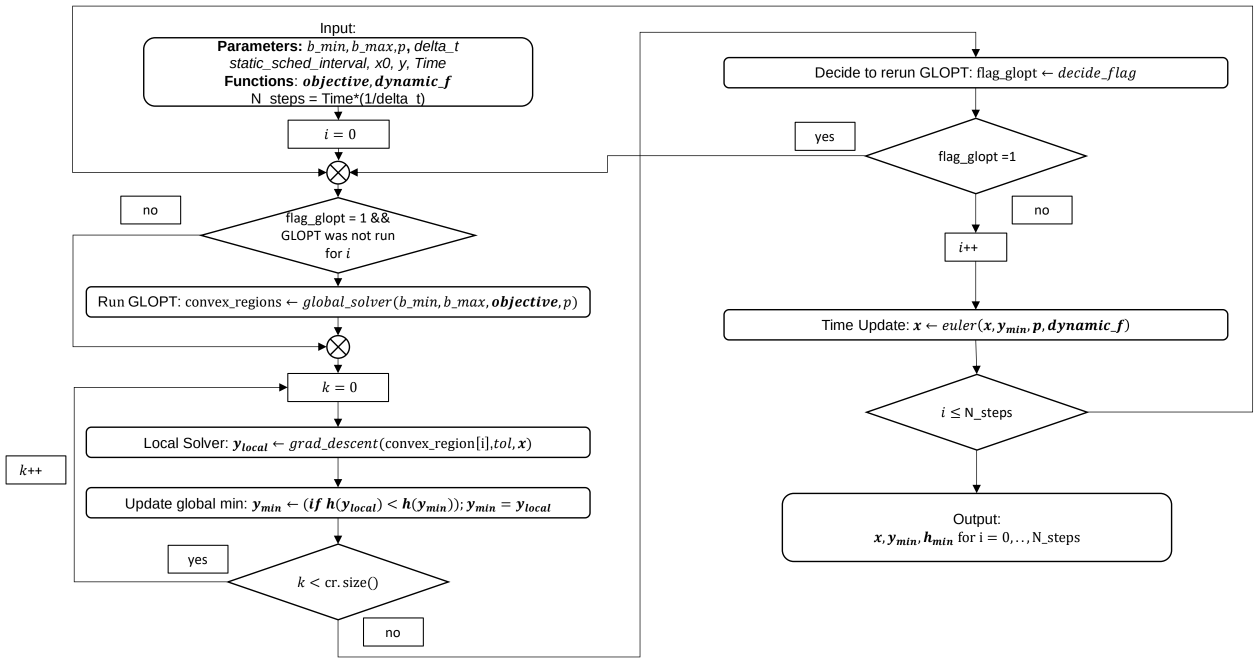
Input:
Parameters: b_min, b_max, p, delta_t
static_sched_interval, x0, y, Time
Functions: $\boldsymbol{objective}, \boldsymbol{dynamic\_f}$
N_steps = Time*(1/delta_t)
$i = 0$
no
flag_glopt = 1 && GLOPT was not run for i
Run GLOPT: convex_regions ← $global\_solver(b\_min, b\_max, \boldsymbol{objective}, p)$
$k = 0$
k++
Local Solver: $\boldsymbol{y_{local}} \leftarrow grad\_descent$(convex_region[i], $tol, \boldsymbol{x}$)
Update global min: $\boldsymbol{y_{min}} \leftarrow (\boldsymbol{if}\ \boldsymbol{h}(\boldsymbol{y_{local}}) < \boldsymbol{h}(\boldsymbol{y_{min}})); \boldsymbol{y_{min}} = \boldsymbol{y_{local}}$
yes
$k <$ cr.size()
no
Decide to rerun GLOPT: flag_glopt ← $decide\_flag$
yes
flag_glopt =1
no
i++
Time Update: $\boldsymbol{x} \leftarrow euler(\boldsymbol{x}, \boldsymbol{y_{min}}, \boldsymbol{p}, \boldsymbol{dynamic\_f})$
$i \leq$ N_steps
Output:
$\boldsymbol{x}, \boldsymbol{y_{min}}, \boldsymbol{h_{min}}$ for i = 0,.., N_steps

Figure 7: Activity diagram of the developed software with the definition of the input, output and main control flow of data

The user definable inputs are specified in Table 1. These inputs define the setting of the problem. The user is also provided with few problem templates that can be found in the source code in the *all_local_opt.cpp/main/Template problems* section.

Table 1 – User definable inputs for the program with short description and type.

Name	Name in the code	Type	Description
Objective function	*objective*	lambda function	Defines the objective function (*h*) that has the function value as output
Dynamic function	*dynamic_f*	lambda function	Defines the function (*f*) of the state variable as output
Function parameters	*p*	vector of double/float	Vector of all parameters that can be used in the functions

Initial x value	*x0*	vector of double/float	Initial condition for the state variable
Initial y value	*y*	vector of double/float	Initial guess of the location of the argument of the objective function
Lower boundaries	*b_min*	vector of double/float	Lower boundary of the optimization problem in each dimension
Upper boundaries	*b_max*	vector of double/float	Upper boundary of the optimization problem in each dimension
Time increment	*delta_t*	scalar double/float	
Period for calling global optimization	*static_sched_interval*	scalar integer	Number of time steps between dynamic update of the state variable that can maximally pass between successive calls to the global optimization routine

The software produces diagnostic information directly in the command terminal related to the execution time of the program and also the notification of when the GLOPT has been used and due to which criterion. The output file that hosts the data of the state variable, optimization variable, and optimization problem value for each time step is located in <build folder>/y_opt.dat. The first column represents the time vector, the subsequent columns represent the values (1 column per dimension) of the state variable x, global minimum y_{min}, and the value of the optimization function h.

3.2 Software Manual

This section presents an overview of how to use the developed software.

Requirements:

- The code relies on the external library dco/c++. Therefore, a working license and an installed version of dco 3.4.4 is required.
- The code has been developed and tested using cmake/3.16.4 and suite intel/2021.4.0.

Main steps:

1. Get access to the software by contacting Dr. Jens Deussen.
2. Configure the path to dco inside global_search_for_local_optima/cmake/ FindNAG_dco_cpp.cmake/line 15.
3. Open all_local_opt.cpp and specify the user inputs in the main function. Use the provided templates as reference for specifying the problem setting.
4. Create a *build* folder in the *global_search_for_local_optima* folder. Navigate to the build folder, open a terminal and build the executable by running “cmake ..” and “make”
5. Run the executable “./all_local_opt”
6. Check the output file build/y_opt.dat.

4 Results and Discussions

4.1 Detection of optima

Reliable detection of local optima is the cornerstone of our software because the selection of the global optimum will only be as good as the available choices. Whenever an event happens, there is a possibility of shift in the global optimum, which makes the resolution of event detection an important factor for tracking the global optimum. Another thing to keep in mind is that detection of new optima will happen only when GLOPT runs, whereas the local search will only optimize the optima within the already detected convex regions from the last GLOPT call. As we will see in the following sections, scheduling of GLOPT seriously impacts the tracking of global optima.

We will discuss the results from each of our toy problems in the following sub-sections. For all the simulations we have used the time step size $\Delta t = 0.01$, which means a total of 101 times steps for toy problem 1 and 3, and 201 time steps for toy problem 2 and 4.

4.1.1 Toy Problem 1

Comparatively the simplest of all toy problems, here we have the expected evolution of the global optimum with just one application of GLOPT at first time step. Two convex regions containing 1 and -1 are given as output from GLOPT (Figure 8), which is followed by tracking of local optima in each of these regions until one of them goes out of the convex region as the system evolves with time.

However, in this problem, the position of the local optima remains fixed while the global optimum jumps from one to the other, which is sufficiently captured by local optimizer alone after using GLOPT once in the beginning.

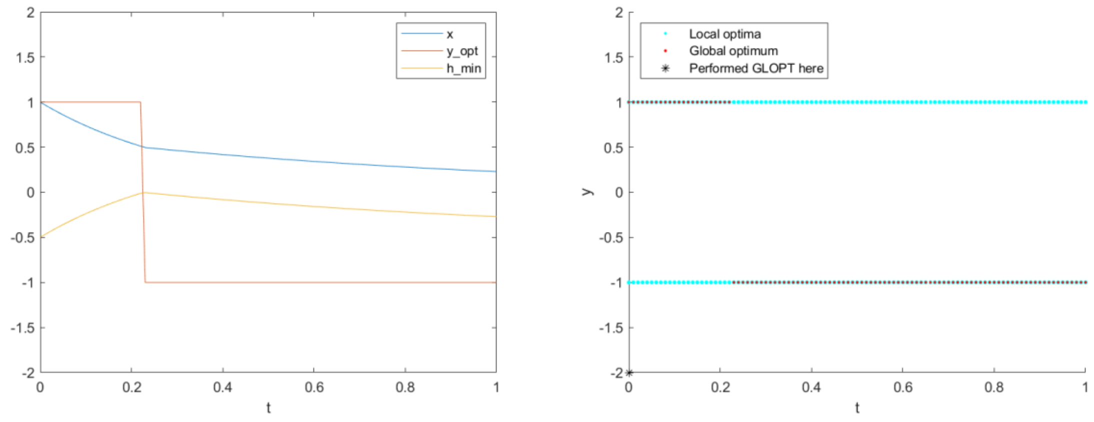

Figure 8 - Toy problem 1 with GLOPT at t=0 only; (left) temporal evolution profiles; (right) relation between local optima and global optimum.

4.1.2 Toy Problem 2

This example is slightly more involved as the position of the local optima also shifts along with the value of the objective function. The effect of delayed detection of events is more pronounced on the tracking of global optimum (Figure 9). We compare the cases with different scheduling of GLOPT (case 2,3,4) with the case where GLOPT was implemented at every time step (case 1).

For case 1, we observe in total of seven events, where old optimum disappears or a new one appears. While the disappearance is less affected by type of static scheduling used, it is always followed by implementation of GLOPT triggered by dynamic scheduling. This is because as an optimum stop existing, local optimizer finds its value to be on the boundaries of the convex region, which is a dynamic trigger to implement GLOPT. This is followed by a revision of the convex regions, and the convex region of local optimum in question is lost.

The appearance of optima is less obvious for the software, and the detection is delayed until GLOPT is implemented. This trigger for GLOPT isn't inspired by appearance of optima, as both static and dynamic scheduling criteria are oblivious to it. Instead, the trigger happens because of some optimum getting to the boundary of its convex region or due to static scheduling count. This impacts the decision for the global optimum, which continues to be sub-optimal until the real global optimum is detected. One can see this happening in case 4, where the selection decision for global optimum of $y \approx 3.5$ is delayed till $t = 1.10$ while it should have happened at $t = 1.01$ as in case 1.

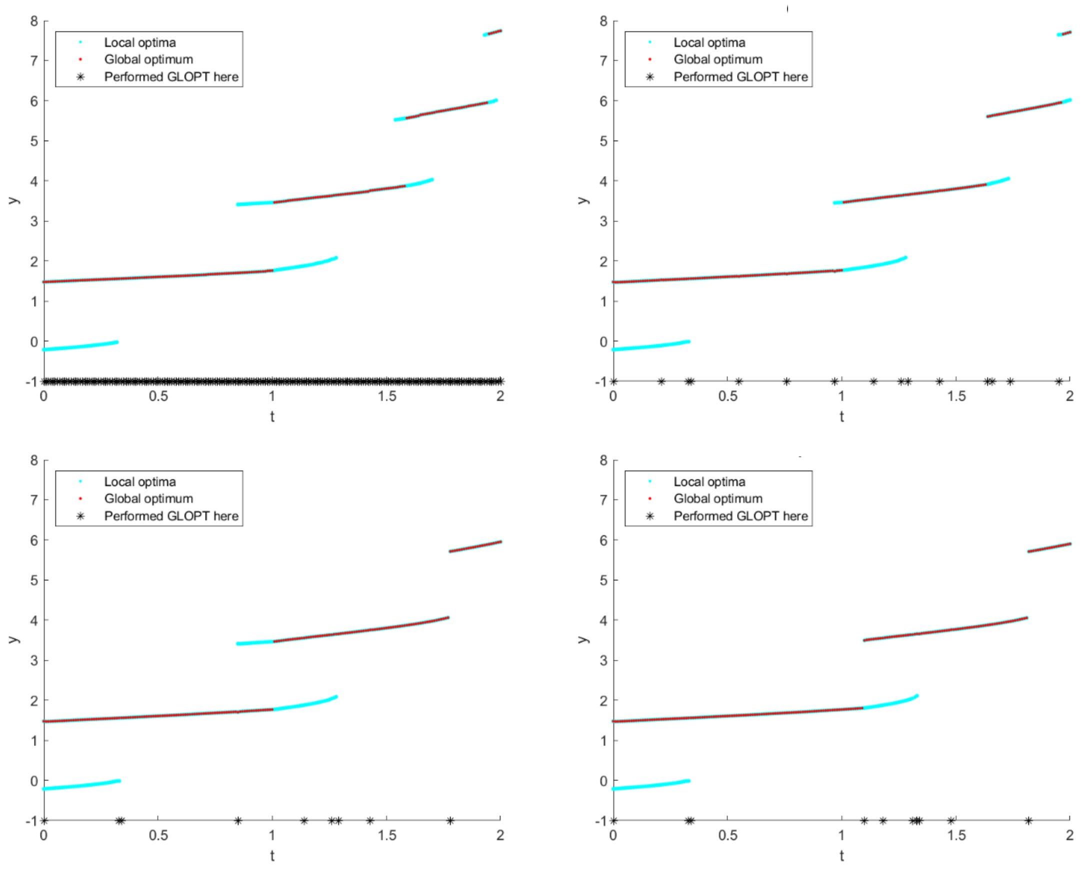

Figure 9 - Toy problem 2 - relation between local optima and global optimum; dynamic scheduling of GLOPT with static scheduling interval of 0 (top left), 20 (top right), 50 (bottom left), 75 (bottom right).

If we use the similar setting as used in toy problem 1, i.e., using GLOPT once in the beginning and local optimization at following time steps, we end with erroneous evolution of global optimum. The two convex regions identified in the beginning by GLOPT serves the region to look for local

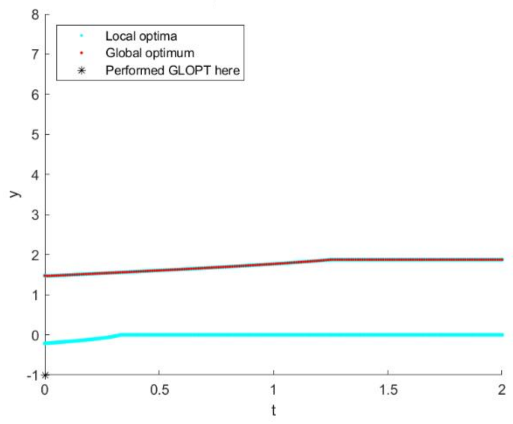

Figure 10 - Toy problem 2 - relation between local and global optima using GLOPT just once at t=0

optima in all the time steps. No events are identified. This is an oversimplification of the problem at hand which models the assumption of no event.

4.1.3 Toy Problem 3

Similar to toy problem 1, the positions of local optima are fixed. However, here we have four local optima, all of which are identified by the GLOPT in the first timestep itself. This makes it optimal to use GLOPT only once, followed by local optimization within the convex regions as no event happens. The global optimum shifts from one local optimum to the other (Figure 11).

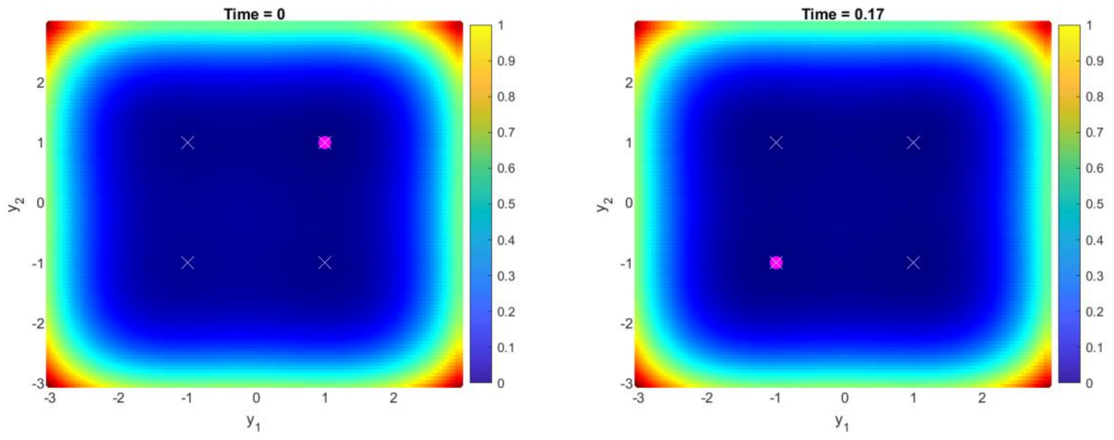

Figure 11 - Shifting of global optimum (pink circle) from one local optimum (white cross) to other for toy problem 3; t=0.0 (left); t=0.17 (right).

4.1.4 Toy Problem 4

Similar results were achieved here as in toy problem 2, where several local optima appear and disappears, and the global optimum jumps from one to another. Comparing with the case where GLOPT is implemented at every step, the frequency of GLOPT implementation impacts profoundly the detection of new local optima, subsequently also the global optimum. We have attached a video link in the Appendix 6.1, in which one can see the evolution of optima under different GLOPT scheduling conditions.

4.2 Computational effort

Conducting a global search for all the optima is computationally more intensive than running a local optimization over the convex regions. Therefore, the clock time increases as the number of GLOPT implementations increase for different scheduling (Table 2).

However, the accuracy of the optima tracking also comes down as the frequency of GLOPT implementations decreases, essentially making it a time-vs-accuracy trade-off. To express the error, we have used root mean squared (over all time steps t) norms of difference of global optimum y_t with respect to the global optimum $y_{t,0}$ obtained from GLOPT implementation at every time step.

$$Error = RMS_t(\| y_t - y_{t,0} \|) \tag{4.16}$$

This trade off becomes more obvious when only one GLOPT implementation happens in the beginning when computation time is the least giving the most inaccurate results.

While an optimal scheduler would keep the trade off at its best automatically, our scheduler balances the trade-off well when static scheduling of 10 is used along with dynamic scheduling.

Table 2 - Comparison of different GLOPT scheduling for toy problem 2 and 4. Error between the global optimum is computer with respect to the case where GLOPT is implemented at every time step. Dynamic scheduling was used in addition to static scheduling for all cases but one, i.e., local opt. only.

	Toy Problem 2 (1D)			**Toy Problem 4 (2D)**		
Static scheduling interval	# GLOPT	Clock time (sec)	RMS error	# GLOPT	Clock time (sec)	RMS error
0	201	0.53	-	201	4.02	-
10	23	0.08	0.0077	61	3.00	0.2404
20	15	0.06	0.3188	54	2.88	0.5677
50	9	0.05	0.6108	42	2.80	0.8623
75	10	0.05	0.7646	42	2.84	1.0801
100	8	0.05	0.9990	37	2.82	1.4122
Local opt. only	1	0.04	2.1605	1	2.49	3.0544

4.3 Outlook

We have seen that our criteria (static and dynamic scheduling) for combining global and local optima search can give us decent results both in terms of accuracy and computation time. However, in no way these criteria are optimal. While the present criteria can deal with optima disappearing events, it is oblivious to optima appearing events.

One of the ideas to improve it is to use a backtracking based correction of the optima. Starting with a coarser interval for static scheduling, the software looks for new convex regions at a low frequency. When a new convex region appears, it undoes the computations till half of the time steps from the penultimate GLOPT, and reperforms GLOPT. If it finds a new convex region, it backtracks again in similar fashion, else it moves forward in time with a reduced static scheduling interval. Such a routine can adapt its scheduling interval to slow down around optima appearing events and speed up by relying more on local optimization when that is not the case.

Additionally, using Newton’s method in place of gradient descent can make local optimization faster especially in regions where magnitude of gradients is small. However, as the GLOPT already gives us a finite convex region that confines the search area, we cannot be sure if the time gain will be sufficient to compensate the time needed for calculation of Hessian.

Further, using higher order time stepping schemes in place of Euler method can boost the accuracy at some cost to the computation time.

5 References

[1] W. Marquardt and A. Mitsos, "Notes & References for Applied Numerical Optimization," Aachener Verfahrens Technik, RWTH Aachen, Aachen, 2021.

[2] J. Nocedal and S.J.Wright, "Numerical Optimization," *Springer-Verlag, New York, 2nd edition,* 2006.

[3] E. D. Gland, D. H. and L. L. , Optimization of chemical processes, 2nd ed., New York: McGraw Hill, 2001.

[4] L. Liberti, Introduction to Global Optimization, Palaiseau, France: LIX, École Polytechnique, 2008, pp. 2-3.

[5] S. Boyd and L. Vandenberghe, Convex Optimization, New York: Cambridge University Press, 2004.

[6] E. Hairer and C. Lubich, "Numerical solution of Ordinary Differential Equations," Université de Genève and Universität Tübingen, 2015.

6 Appendix

6.1 Link for videos

We have uploaded the video of local optima evolution over time for toy problem 3 and 4 at https://rwthaachende-my.sharepoint.com/:f:/g/personal/sarthak_kapoor_rwth-aachen_de/EuA53jQQg0xFtPa3RO4Ft_MBmXYuHbobrfALIhY9lOxGdg?e=XYNdrp.