OPTIMIZATION BASED CONTROL OF CASCADED ELECTRICAL CIRCUITS

by

AMAN DUBEY

Master's thesis submitted in partial fulfillment
of the requirements for the degree of
M.Eng in Electrical Engineering and Embedded Systems

Supervisor:
Prof. Dr.-Ing. Lothar Berger
Mr. Stephan Scholz M.Sc

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

28 October, 2023

Optimization Based Control of Cascaded Electrical Circuits

Aman Dubey

Abstract

In this thesis, the challenges of optimizing higher-order circuits are addressed, with a focus on various methods to align system performance with desired reference outputs. The primary question under consideration pertains to the feasibility of optimization techniques in managing the complexities of higher-order system dynamics and, if so, which optimization approach—global or local—minimizes output errors.

Three optimization techniques—Flatness-Based Control (FBC), Optimization-Based Control (OBC), and Model Predictive Control (MPC)—are applied and analyzed. Notably, FBC faced challenges when dealing with higher-order systems, primarily due to its reliance on the steepness of the reference signal. In contrast, OBC and MPC proved effective in handling such systems.

Within the OBC framework, the Polyopt optimizer demonstrated exceptional performance, effectively managing parameter variations. In the context of MPC, the interplay of prediction horizon and control horizon underscored the importance of an extended prediction horizon and a shorter control horizon.

Furthermore, the behavior of global optimizers revealed their ability to closely follow reference signals, albeit with increased jitter. Local optimization, on the other hand, yielded input signals that led to lower errors and smoother output responses.

The findings presented here hold substantial promise for applications in power electronics and motor control, offering the potential for smoother responses and improved control methods, particularly in systems such as, power converters, clippers and clampers.

List of Figures

List of Tables

Declaration of Authorship

I, Aman Dubey, hereby declare that this thesis, entitled "Optimization based control of cascaded electrical circuits," is entirely my own work. I have conducted the research, written the content, and prepared this document without unauthorized assistance or external sources, except as appropriately acknowledged and cited in the text.

I attest that all sources used for information, data, or ideas have been properly referenced and cited in accordance with the established academic conventions and guidelines. Any materials or concepts obtained from other works are duly credited.

I further declare that this thesis has not been submitted for any other academic qualification, and it has not been published in part or in whole in any format.

By signing below, I confirm my understanding of and adherence to the principles of academic integrity and the ethical standards for conducting research.

Signature ______________________________ Date ______________________________

Acknowledgement

I am profoundly grateful to all those who have been instrumental in guiding and supporting me throughout the challenging journey of completing this thesis.

Foremost, I extend my heartfelt thanks to my esteemed advisor, Mr. Stephan Scholz, whose mentorship, guidance, and unwavering support have been invaluable. Mr. Scholz's expertise, patience, and encouragement have played a pivotal role in shaping this research. From the inception of my embedded control project to the culmination in this thesis, his unwavering dedication and insightful guidance have served as a constant source of inspiration.

I am equally appreciative of Professor Lothar Berger for the profound impact of his insights and constructive feedback, which ignited dynamic thinking and brought fresh perspectives to this project.

My gratitude extends to the vibrant community of fellow researchers and colleagues at Hochschule Ravensburg Weingarten. Their contributions have provided a stimulating academic environment and facilitated engaging, insightful discussions that have enriched this work.

To my friends and family, I offer my heartfelt thanks for their unwavering encouragement, unwavering belief in my abilities, and their understanding during the course of this academic endeavor.

I reserve my deepest appreciation for all the participants and individuals who generously contributed their time and expertise to this study. Without your cooperation and insights, this research would not have been possible.

This thesis represents the collective effort and support of these remarkable individuals. I extend my sincere gratitude to all who have played a part in this important journey. Thank you for being an integral part of this achievement.

Contents

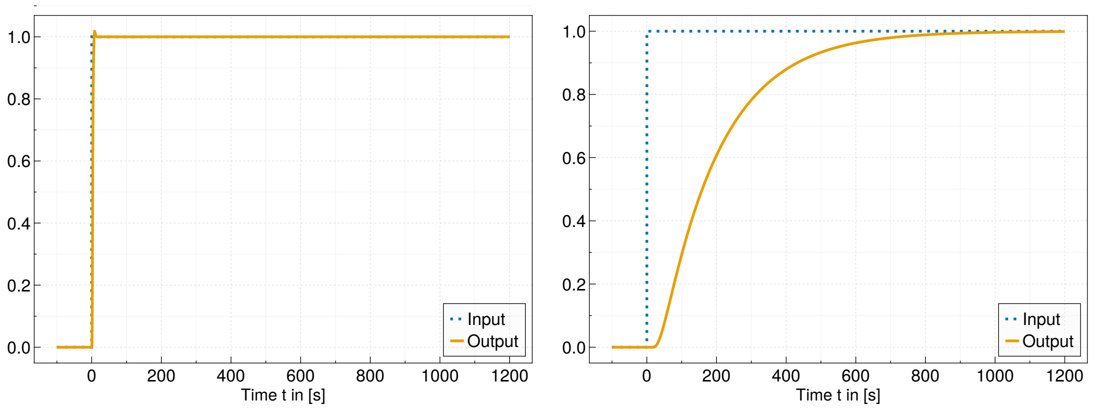
1 Introduction

A few years ago, a spacecraft attempting to land on the lunar surface encountered a critical failure during its final landing stage. This failure arose due to conflicting optimization parameters derived from various optimization techniques. The parameters produced by the algorithm were not practically attainable, leading to an unexpected acceleration of the lander's velocity as it approached the surface. This undesirable outcome directly contributed to the mission's failure, as presented in [6]. Thus, it becomes evident that optimization is a highly intricate and indispensable aspect of the scientific realm.

In the contemporary landscape, numerical optimization assumes a pivotal role across diverse sectors including automotive, pharmaceuticals, space research, and beyond. This master's thesis endeavors to explore a spectrum of optimization methodologies, encompassing non-optimization approaches like Flatness-based control, and culminating in the utilization of a closed-loop technique known as Model Predictive Control (MPC).The realm of model predictive control finds widespread application within the chemical and pharmaceutical industries, particularly in the precise management of heat within thermodynamically sensitive processes. In contemporary times, this methodology has piqued the curiosity of engineers within the power systems domain, especially in power electronics, as a means to regulate essential motor parameters such as torque and angle. Furthermore, its utility extends to the automotive sector, specifically within Advanced Driver Assistance Systems (ADAS).

This master's thesis aims to enhance the output of a cascaded RLC circuit, aligning it with a specified reference output. To achieve output control, we leverage an input serving as the control variable. An inquiry arises regarding the choice of an RLC circuit for this purpose. The rationale behind this selection lies in its dual attributes: the RLC circuit is both well-known for its simplicity and comprehensibility, and it offers an exceptional degree of flexibility in system order manipulation through the seamless integration of additional RLC circuits. Moreover, the circuit's inherent configurability empowers us to modulate the system's stiffness by adapting the values of its constituent elements (R, L, and C).

The depicted figure illustrates the responses of a 2nd-order and a 20th-order cascaded RLC circuit. Evidently, the lower-order system exhibits overshoot in its response, while the higher-order counterpart demonstrates sluggish behavior. Our optimization endeavor addresses both scenarios, seeking to devise a comprehensive solution that transcends these distinctions.



(a) Lower order system response to unit step input (b) Higher Order system response to unit step input

Figure 1: Single and multiple cascaded RLC circuit response to unit step

The research endeavors lead to the formulation of the following key objectives

1. Does optimization-based control outperform alternative techniques like flatness-based control?

2. What optimization technique proves most efficacious for both higher-order and lower-order systems?

3. Can model predictive control enhance system stability while minimizing overall loss?

4. Is it possible to generate continuous time input signal with model predictive control?

The crux of the research centers on addressing the intricacies of higher-order systems. These systems involve higher-order derivatives, rendering them acutely responsive to minute changes in input. Notably, even slight alterations in input yield abrupt changes in output. Another pivotal facet of analysis pertains to evaluating the computational efficiency of optimization techniques, gauging both the incurred loss value and the time required for optimizing parameters.

Modern Control Theory and Classical control Theory

Control theory has experienced significant advancements in the 21st century, driven by the rapid development of computers and the substantial increase in computational speed. Classical control theory, which primarily encompasses techniques developed before 1950, relies on methods such as root locus analysis, Bode plots, Nyquist stability criteria, and Routh-Hurwitz analysis.
However, classical control theory has a notable limitation—it predominantly employs transfer functions and is confined to analyzing Linear Time-Invariant (LTI) systems within the frequency domain. In contrast, modern control theory has flourished with the advent of powerful computers, enabling the analysis of systems in the time domain. The adoption of state-space modeling in modern control has effectively transcended the limitations that once hindered classical control theory.
Example of Classical Control Theory
A. Frequency Domain Analysis
B. PID(Proportional-Integral-Derivative) Control
C. Heuristic Tuning
D. Laplace Transforms

Example of Modern Control Theory
A. State-Space Representation
B. Optimal Control
C. Robust Control

1.1 Thesis Structure

The thesis is structured as follows, it comprises several distinct chapters, each dedicated to delving into specific research domains.

Chapter 2: Problem Formulation

This section of the thesis covers several crucial topics. It begins by developing a state space model for the RLC circuit. It then explores open-loop control methods, including Flatness-Based Control and open-loop optimization-based control. The section also highlights the significance of closed-loop control and the role of feedback mechanisms in system stabilization.

Chapter 3: Flatness Based Control

In this section, we undertake a series of critical tasks. First, we implement flatness-based control for N cascaded RLC circuits, establishing direct mathematical relationships between input and output. We then evaluate the results of the flatness-based control and address its limitations, particularly in the context of higher-order systems. Additionally, we investigate the role of the hyperbolic tangent function and its impact on the steepness of the flatness-based control approach.

Chapter 4: Optimization based Control

In this section, we address several key components. We begin by formulating mathematical equations and exploring the significance of the parameterized tangent hyperbolic function. We then delve into the various optimization algorithms used, including Gradient Descent, and their role in optimization-based control. Additionally, we provide a comprehensive explanation of the implementation of Optimization-Based Control (OBC) through a clear and informative flowchart. Lastly, we present the obtained results and derive meaningful insights from them.

Chapter 5: Model Predictive Control

This section encompasses the following critical components. We commence by implementing Model Predictive Control (MPC) and offer an elucidative flowchart that details the control process comprehensively. We then present the achieved results through the application of MPC across systems of varying orders. Additionally, we analyze the impact of alterations in control horizon and prediction horizon, while also exploring the influence of global and local optimizers on the performance of MPC.

2 Problem Formulation

This section deals with our problem formulation for generalized higher order RLC circuits(as shown in figure 3, specifically focusing on state space representation. The system Matrix(A), Input Matrix(B) and Output Matrix(C) are defined for a generalized RLC circuit.

2.1 System Description

As previously mentioned, the primary objective is to stabilize the output, ensuring its alignment with the desired reference. Let us now delve into the system's design, specifically focusing on the cascaded RLC circuit configuration.

First order RLC circuit

Now Let's look into the mathematical analysis for the first order RLC circuit(as shown below)

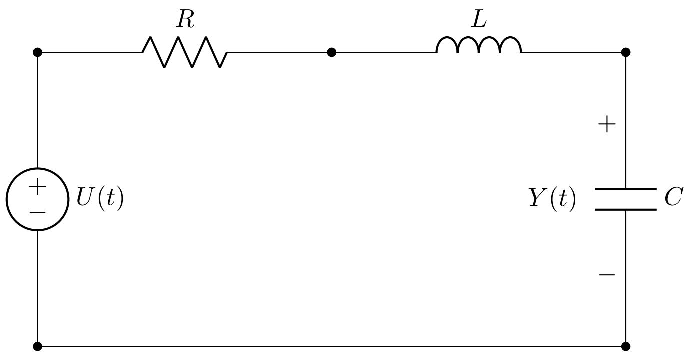


Figure 2: first order RLC Circuit

From the Electrical analysis we can say

$$u_R(t) = Ri(t), \tag{2.1}$$

$$u_L(t) = L\frac{di(t)}{dt}, \tag{2.2}$$

$$u_C(t) = \frac{1}{C}\int i(\tau)d\tau, \tag{2.3}$$

Reshaping above equations provide the current as,

$$i(t) = C\frac{du(t)}{dt} = C\dot{u}(t). \tag{2.4}$$

From the Kirchhoff's voltage law i.e. KVL we can say

$$u_{\text{in}}(t) = u_R(t) + u_L(t) + u_C(t) = Ri(t) + L\frac{di(t)}{dt} + u_C(t). \tag{2.5}$$

Similarly let's dive into generalized representation for higher order systems.

Higher Order RLC circuit
As we solve the equation for First order system we can similarly write the final equation for third order system as follows.

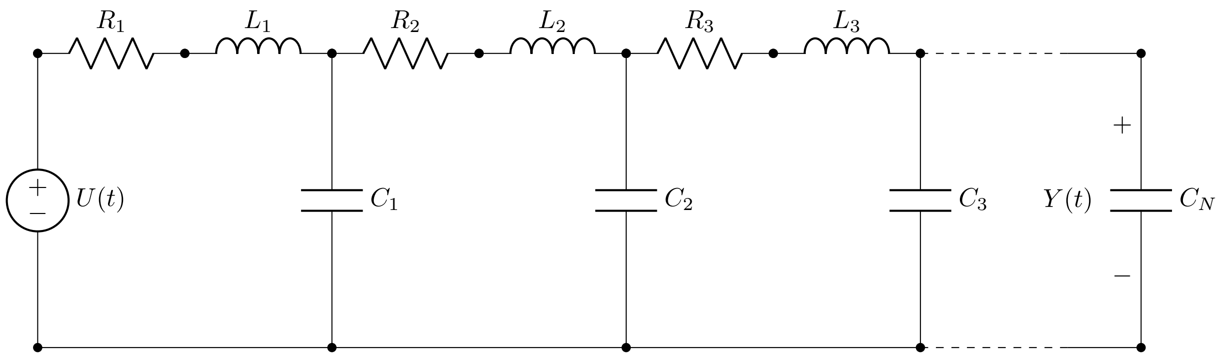


Figure 3: $2N^{th}$ order RLC Circuit

Given equations[7] as follows

$$\mathbf{i}_n(t) = \mathbf{C}_n \dot{\mathbf{u}}_C^n(t), \tag{2.6}$$
$$\mathbf{i}_{n-1}(t) = \mathbf{C}_{n-1} \dot{\mathbf{u}}_C^{n-1}(t) + \mathbf{C}_n \dot{\mathbf{u}}_C^n(t), \tag{2.7}$$
$$\vdots$$
$$i_1(t) = \sum_{n=1}^{N} \mathbf{C}_n \dot{\mathbf{u}}_C^n(t), \tag{2.8}$$

$$u_{in}(t) = R_1 i_1(t) + L_1 \frac{di_1(t)}{dt} + u_{C1}(t). \tag{2.9}$$

$$\begin{aligned} u_{C1}(t) &= R_2 i_2(t) + L_2 \frac{di_2(t)}{dt} + u_{C2}(t), \\ &\vdots \\ u_{Cn-1}(t) &= R_n i_n(t) + L_n \frac{di_n(t)}{dt} + u_{Cn}(t). \end{aligned}$$

Putting the value of $i_1(t)$ in equations (2.9) results in

$$\begin{aligned} u_i n(t) &= R_1 \left[\sum_{n=1}^{N} \mathbf{C}_n \dot{\mathbf{u}}_C^n(t) \right] + L_1 \left[\sum_{n=1}^{N} \mathbf{C}_n \ddot{\mathbf{u}}_C^n(t) \right] + \mathbf{u}_{C1}(t), \\ u_{C1}(t) &= R_2 \left[\sum_{n=2}^{N} \mathbf{C}_n \dot{\mathbf{u}}_C^n(t) \right] + L_2 \left[\sum_{n=2}^{N} \mathbf{C}_n \ddot{\mathbf{u}}_C^n(t) \right] + \mathbf{u}_{C2}(t), \\ &\vdots \\ u_{Cn-1}(t) &= R_n \mathbf{C}_n \dot{\mathbf{u}}_C^n(t) + L_n \mathbf{C}_n \ddot{\mathbf{u}}_C^n(t) + \mathbf{u}_{Cn}(t). \end{aligned}$$

Now generalize it into an matrix form in order to write the voltage on n^{th} capacitor.

$$\mathbf{M}\ddot{\mathbf{z}}(t) + \mathbf{D}\dot{\mathbf{z}}(t) + \mathbf{S}\mathbf{z}(t) = \mathbf{G}\mathbf{u}(t)$$

rearranging it leads to

$$\ddot{\mathbf{z}}(t) = -\mathbf{M}^{-1}\mathbf{D}\dot{\mathbf{z}}(t) - \mathbf{M}^{-1}\mathbf{S}\mathbf{z}(t) + \mathbf{M}^{-1}\mathbf{G}\mathbf{u}(t)$$

and introducing new variable names:

$$\mathbf{x}_1(t) = \mathbf{z}(t), \quad \mathbf{x}_2(t) = \dot{\mathbf{z}}(t).$$

the final state space notation:

$$\begin{pmatrix} \dot{\mathbf{x}}_1(t) \\ \dot{\mathbf{x}}_2(t) \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{S} & -\mathbf{M}^{-1}\mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{M}^{-1}\mathbf{G} \end{pmatrix} \mathbf{u}(t). \tag{2.10}$$

We have $\tilde{S} = M^{-1}S$ and $\tilde{D} = M^{-1}D$, In our case, for simplicity considering $R_1 = R_2 = \ldots = R_n = R$, $L_1 = L_2 = \ldots = L_n = L$, and $C_1 = C_2 = \ldots = C_3 = C$. Our Matrix looks as follows

$$M = LC \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}, \quad D = RC \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

$$S = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}, \quad G = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

M, D and S are N $\times$ N matrix , Whereas G is N $\times$ 1. Where N is number of cascaded RLC circuits. This way the RLC circuit can be represented in state space form and the order of the equation can be changed by simply modifying N. Finally we have the RLC circuit represented in State space format as shown in (2.11) and (2.12).

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.11}$$
$$y(t) = x_1(t) = Cx(t). \tag{2.12}$$

Where A, B could be calculated from equation (2.10), C would look as

$$C = [0, 0, ..., 0_{N-1}, 1, 0_{N+1}, 0, ..., 0_{2N}]$$

2.2 Open Loop Control

The open-loop controller, also referred to as a feed-forward controller, designs the control variable based on anticipated references. This approach involves the absence of a feedback loop, emphasizing a pre-defined control action. This configuration proves advantageous when rapid response is prioritized, even though it may entail a compromise with increased loss values or errors in the presence of disturbances. Open-loop control finds utility in applications involving uncomplicated machines, such as household appliances.
However, its suitability diminishes when applied to critical systems. The inability to accommodate abrupt changes arising from external interference or internal disturbances diminishes its reliability.

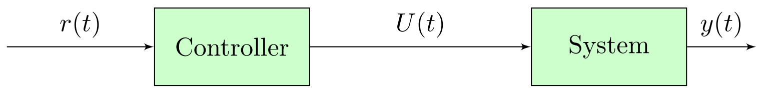


Figure 4: Open-Loop Control System with Controller and System

This project uses two open loop control techniques.
1.Flatness-Based Control (FBC).
2.Open loop Optimization based control.

2.3 Flatness-Based Control

Flatness-based control is a methodology that designs and implements controllers for intricate dynamical systems by leveraging the concept of 'flat outputs[8].' The core concept is to establish a direct relationship between desired outputs and inputs through mathematical expressions[9].
Here are some important terms and concept used in FBC.

1. Flat Output and Flatness Principle

The desired flat output can be formulated using the input and its derivatives, namely as

$$u(t) = y(t) + \frac{d}{dt}y(t) + \frac{d^2}{dt^2}y(t) + \ldots + \frac{d^n}{dt^n}y(t). \tag{2.13}$$

where y(t) represents desired output and u(t) input signal. However, the applicability of this property depends on the system's characteristics. If our system can be transformed into a flat output system, then its complete input trajectory can be determined by specifying the desired output and its derivatives.

2. State Transformation

When choosing the desired flat outputs, you need to perform a mathematical formulation where the input is expressed in terms of the output and its derivatives, as demonstrated in Equation (2.13).

3. Advantages and Limitations

Flatness-based control (FBC) offers several advantages, including its inherent simplicity and robustness due to the direct input-output relationship it establishes. Moreover, FBC effectively manages disturbances. However, there exist certain limitations. Notably, FBC necessitates the reference output to exhibit flatness, a condition that might not be attainable for complex non linear systems. Additionally, FBC's sensitivity to model design is noteworthy, as its control strategy hinges on the accuracy of the corresponding mathematical model.

2.4 Optimization Based Control

This control strategy relies on optimization to determine the control inputs (control variables) that guide the system to track a desired reference trajectory or minimize the loss function[10]. Let's briefly delve into the mathematical foundation[11]. Given the function $f(x)$, our objective is to determine a λ such that $f(\lambda) \leq f(x)$ for all x belonging to $\mathbb{R}^n$.
A condition formulated would be as shown in 2.14, i.e. first order differential needs to be zero.

$$\frac{df(\lambda)}{dx} = 0 \tag{2.14}$$

Commonly, $f(x)$ is referred to as the cost function or loss function, while λ represents the optimal value where the loss is minimized. However, it's essential to note that this condition alone is insufficient, as the gradient can be zero at both local minima and maxima. Further computation of higher-order derivatives is necessary to pinpoint the precise optimized point. Generally, we compute the Hessian matrix, which represents the second derivative. If the Hessian is positive, it implies that the obtained point is a local minimum, while a negative gradient suggests a local maximum. However, these points do not guarantee global maxima or minima, as the interval may contain multiple points satisfying these conditions. Therefore, it is necessary to individually examine each potential point to determine the desired global minimum or maximum. In Figure 5, three potential points are evident where the derivative may be zero. To pinpoint the exact minima, it becomes necessary to calculate the second derivative at these points.

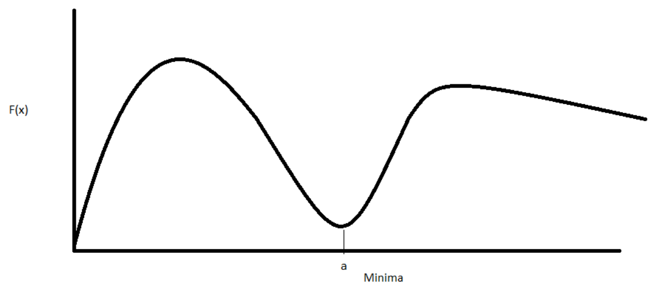


Figure 5: Function minima with three saddle points.

The cost function $L(t)$ is defined as

$$L(t) = \int_0^{T_f} (e(t))^2 \, dt.$$

Where $e(t)$ refers to the error value computed over the horizon. The error $e(t)$ is given by

$$e(t) = r(t) - y(t).$$

Where $r(t)$ represents the reference or desired signal, and $y(t)$ is the obtained output. Subject to the constraint

$$g(t) = c.$$

2.5 Closed Loop Control

A closed-loop control system incorporates live feedback to the controller, enabling real-time response to sudden disturbances (errors) and taking them into account during optimization. A general closed loop systems looks as shown in figure 6, along with system and controller it contains following mentioned signals:-
Reference signal r(t)
System state x(t)
Control signal u(t)
Output signal y(t)

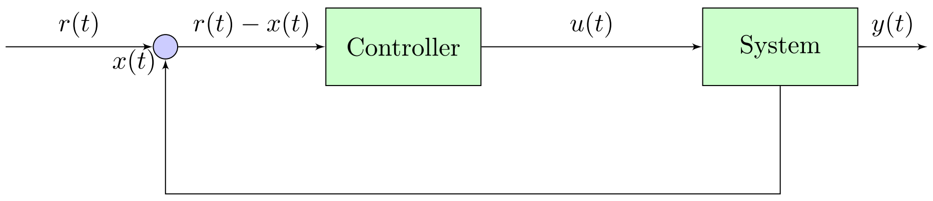


Figure 6: Closed-Loop Control System with Controller and System

In conclusion, a control system with closed loop make real time adjustments in the system and helps in dealing with unwanted/unwarranted errors occurring in the plant . This method helps in controlling the system with high precision and increase the overall robustness of system. Later on we are going to look into model predictive control which basically is a type of feedback based controller.

3 Flatness Based Control

Having previously addressed the fundamental principles of flatness-based control, the focus now shifts to observing its practical implementation across various cascaded RLC circuits. The objective remains guiding the input signal to synchronize with the desired reference signal. Let's formulate the mathematical analysis for it. Following are the state space modelling of the system

$$\dot{x}(t) = Ax(t) + Bu(t). \tag{3.1}$$

$$y(t) = Cx(t). \tag{3.2}$$

Differentiating Equation (3.1) and substituting the value of $\dot{x}(t)$ from Equation (3.2), we get

$$\dot{y}(t) = C\dot{x(t)} = C(Ax(t) + Bu(t)) = CAx(t) + CBu(t),$$

If CB=0 then continue the differentiation y as

$$y^{(n)}(t) = CA^n X + CA^{n-1}Bu,$$

In case of cascaded RLC circuit at n=2N, where N is the number of RLC subsystems, $CA^{2N-1}B \neq 0$
So we find

$$u(t) = [CA^{2N-1}B]^{-1}(y^{(n)}(t) - CA^{2N}x(t)). \tag{3.3}$$

Now we need to express the states x in terms of y using

$$\begin{pmatrix} y \\ y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(2N-1)} \end{pmatrix} = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{2N-1} \end{pmatrix} x(t),$$

where,

$$Z = \begin{pmatrix} y \\ y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(2N-1)} \end{pmatrix} and, T = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{2N-1} \end{pmatrix}$$

In turn we obtained,

$$x(t) = T^{-1}Z.$$

Putting the value of x(t) in 3.3 we get

$$u(t) = (CA^{2N-1}B)^{-1}(y^{2N} - CA^{2N}T^{-1}Z). \tag{3.4}$$

The output signal y(t) has to follow the reference signal r(t).

$$r(t) = \frac{1 + \tanh(a(t - \frac{T_f}{2}))}{2}.$$

and replace y(t) and it's derivatives in equation 3.4, by computing $\dot{r}(t)$, $\ddot{r}(t)$, ..., $r^{(2n)}(t)$.
Derivative of r is

$$r^{(n)}(t) = \frac{a^n}{2} . \frac{d^n}{dt^n} tanh(t - \frac{T_f}{2}).$$

Formula for computing derivative of tangent hyperbolic function looks as follows.

Tangent Hyperbolic Derivatives

Before moving on let's look at the derivatives of tanh function[12]

$$\frac{d^n}{dz^n} \tanh z = \frac{2^{n+1} e^{2z}}{(1 + e^{2z})^{n+1}} \sum_{k=0}^{n-1} (-1)^k A_{n,k} e^{2kz}.$$

Where $A_{n,k}$ is Euler number[13]

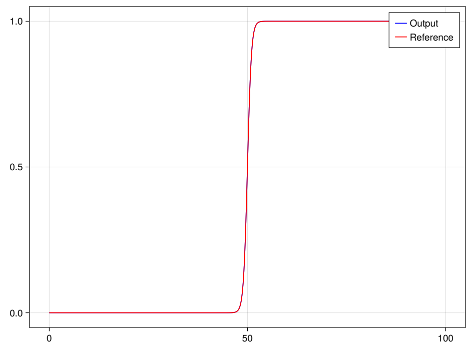
$$A(n,k) = \sum_{i=0}^{k} (-1)^i \binom{n+1}{i} (k + 1 - i)^n.$$

3.1 Results with FBC

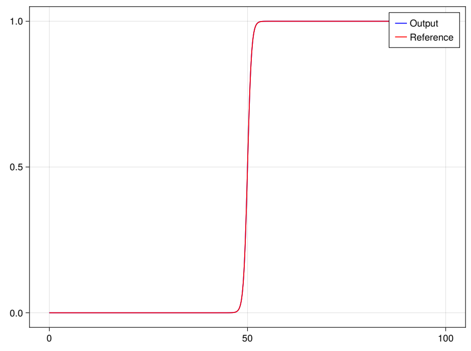
Firstly we know the desired reference which is needed.

$$r(t) = \frac{1 + \tanh(a(t - \frac{T_f}{2}))}{2}. \tag{3.5}$$

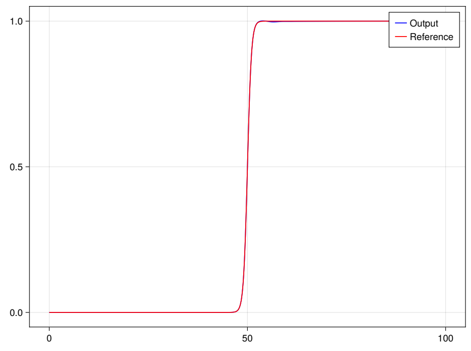
Where T_f represents the total time and 'a' is a steepness value. The reference signal is selected because it exhibits properties similar to that of a unit step signal, albeit with a rise time that can be controlled by the user by varying the steepness value 'a'. Now, in equation 3.4, substitute the reference signal (equation 3.5) for 'y' and solve the equation. This will yield the desired 'u' value, which will be used to solve the equation and obtain the desired output.
Now, We shall examine the results presented in Figure 7. In this analysis, we have maintained the 'a' (steepness) value as constant while varying the 'N' value. It is readily apparent from the figure that for smaller values of 'N,' the FBC demonstrates greater accuracy in problem-solving. However, as the system order 'N' increases, it exhibits poor performance, resulting in higher overall loss, undershoot, and overshoot. It becomes apparent that increasing the value of N also escalates the stiffness of the equation. This heightened stiffness, in turn, poses a significant challenge for the solver, leading to difficulties in solving the equation.



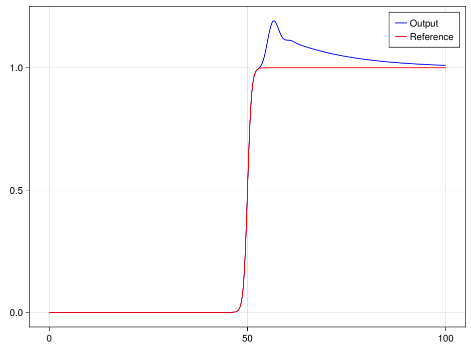
(a) N=1



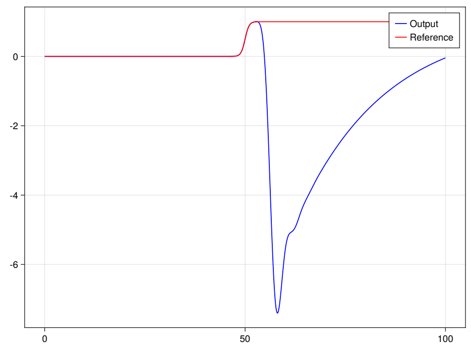
(b) N=3



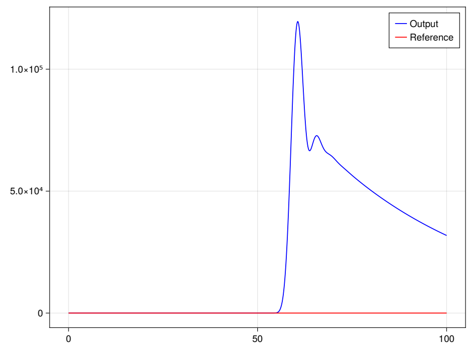
(c) N=5



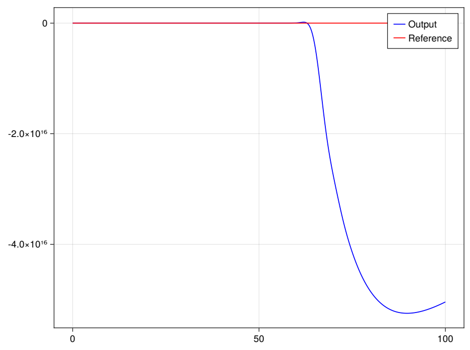
(d) N=6



(e) N=7



(f) N=10



(g) N=15

Figure 7: FBC With steepness a=1 for various N.

In Figure 8, the FBC output exhibits a significantly pronounced overshoot and undershoot. Between the time intervals of 60 to 63, it experiences a sudden and substantial overshoot followed by a subsequent undershoot. Both the overshoot and undershoot values are in the order of millions. This observation underscores the diminishing relevance of FBC when dealing with higher values of 'N'.

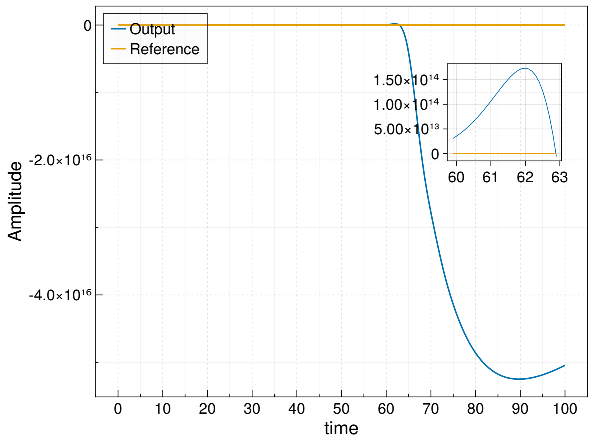


Figure 8: FBC for N=15 with a=1.

Tangent Hyperbolic function

The question may arise: why was the reference signal chosen to be a tanh function rather than a unit step signal? Firstly, the unit step signal does not exist in the real world, as each function takes some time to rise. Additionally, the hyperbolic tangent function, commonly referred to as tanh, plays a significant role in today's scenarios, finding widespread application in machine learning as an activation function within artificial neural networks. While it shares some similarities with the ordinary tangent function, its distinctive feature is its output constraint within the interval of -1 to 1. Below, we outline several key advantageous properties that have contributed to its extensive adoption:

Smoothness The tanh function and its derivatives exhibit continuity and differentiability at all points. Moreover, its bounded range within the interval of -1 to 1 renders it an optimal choice for control system applications. As depicted in Figure 9, it is evident that the first and second derivatives exist, with their respective amplitudes being less than that of the original signal. It's worth noting that tangent hyperbolic functions are employed to enhance the smoothness of responses in systems that exhibit non-smooth characteristics. This application has garnered significant attention in recent research endeavors.

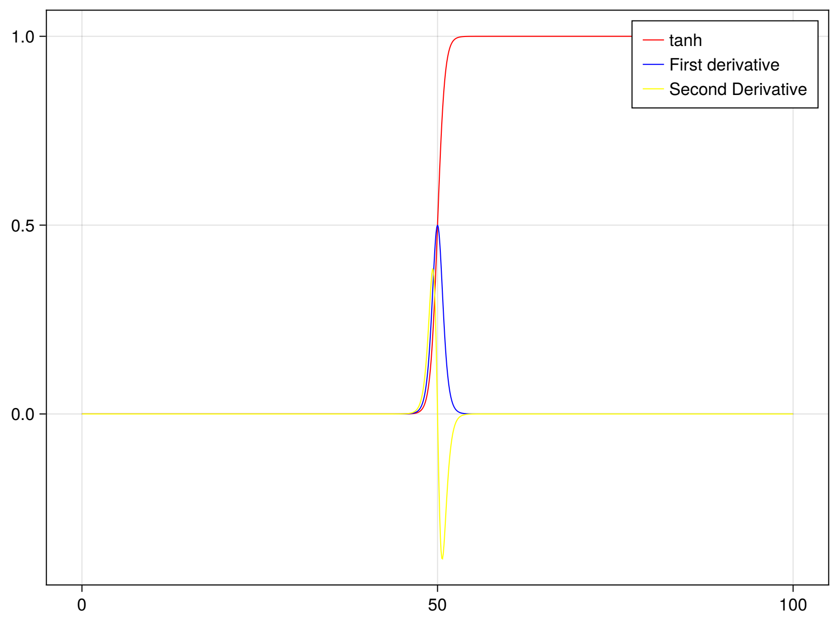


Figure 9: Tangent hyperbolic and it's derivatives.

Steepness Affects On FBC

Again looking in the equation 3.4

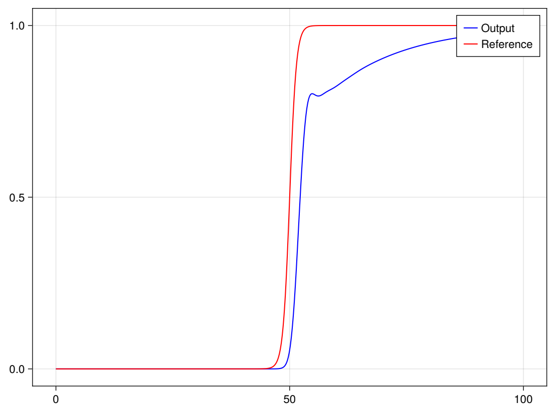
$$u = (CA^{2N-1}B)^{-1}(y^{2N} - CA^{2N}T^{-1}Z).$$

As the input of the function relies on 2N derivatives of the output (y), in FBC, y is substituted with the reference signal (r), and the equation is subsequently solved. However, in this context, as we are aware...

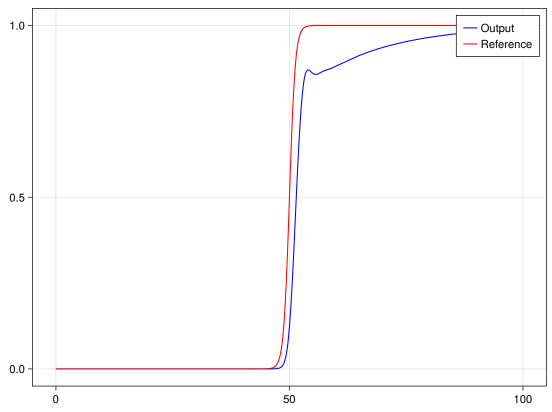
$$f(x) = g(ax),$$

$$\frac{df}{dx} = a\frac{dg}{dx}, ..., \frac{d^N f}{dx^2} = a^N \frac{d^N g}{dx^2}. \tag{3.6}$$

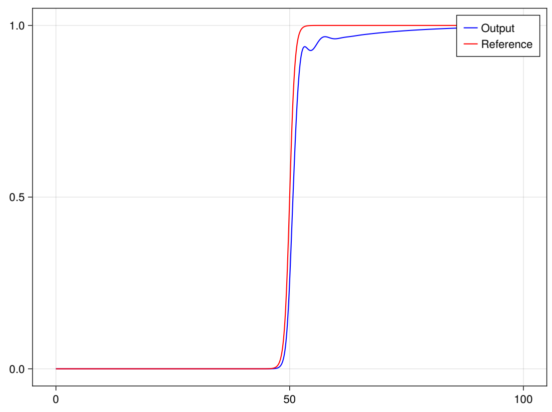
From equation 3.6 it is clear that as we change the value of a , it has a huge impact on the output. For the higher derivative it becomes more sensitive to the value of a.



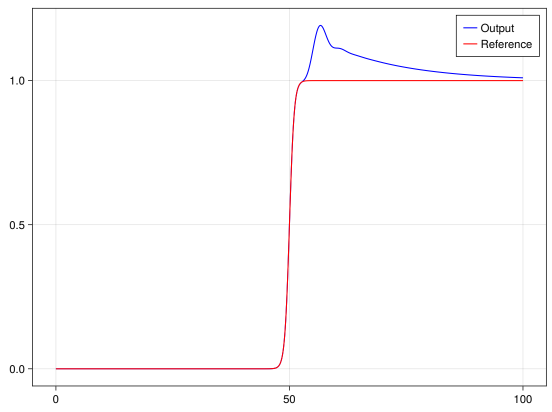
(a) a=0.7



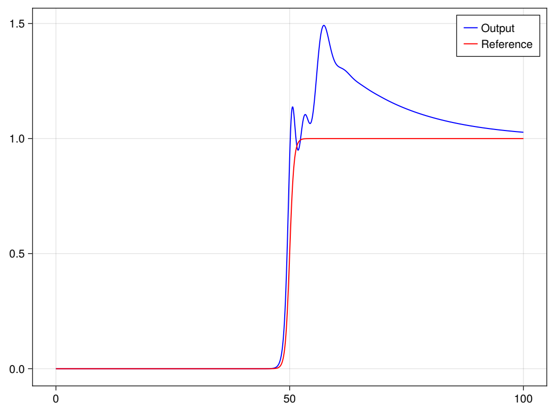
(b) a=0.8



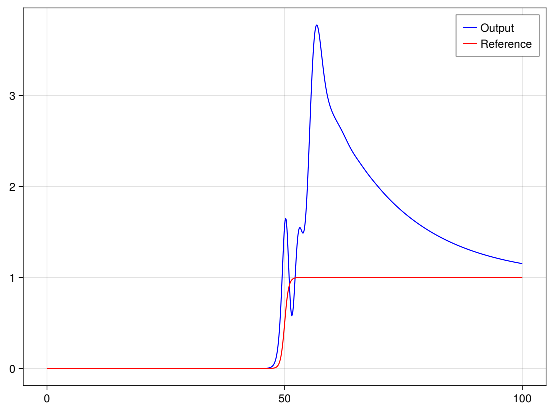
(c) a=0.9



(d) a=1.0



(e) a=1.1



(f) a=1.2

Figure 10: FBC with N=6 and varying steepness(a).

Figure 10 illustrates the impact of the 'a' parameter variation on a circuit with an order of 12 (2N). It is evident that when 'a' is set to 0.9, the FBC output consistently stays below the reference, whereas at a value of 'a' equal to 1.0, it exhibits an overshoot beyond the desired range. This observation suggests that the FBC output will closely track the reference signal within the interval of 0.9 to 1.

3.2 Calculating A^N

Before we move on to the next steps, it's important to make sure that our calculations are both fast and dependable. This will help us solve the optimization problem more quickly. Matrix A has a specific pattern, so we should take a closer look at that pattern to make our calculations easier.
We have

$$A = \begin{pmatrix} 0 & I \\ -M^{-1}S & -M^{-1}D \end{pmatrix}$$

Let $A_1 = -M^{-1}S$ and $A_2 = -M^{-1}D$, We have

$$A = \begin{pmatrix} 0 & I \\ A_1 & A_2 \end{pmatrix}, \quad A^2 = \begin{pmatrix} A_1 & A_2 \\ A_2A_1 & A_1 + A_2^2 \end{pmatrix},$$

$$A^3 = \begin{pmatrix} A_2A_1 & A_1 + A_2^2 \\ A_1^2 + A_2^2A_1 & A_2A_1 + A_1A_2 + A_2^3 \end{pmatrix}.$$

When we closely observed the value of A, A_2, A_3 there seems to a pattern. We define A_n in terms of

$$A_n = \begin{pmatrix} a_{1,n} & a_{2,n} \\ a_{3,n} & a_{4,n} \end{pmatrix}.$$

Finally we can compute A_n with the help of A_{n-1}, A_1 and A_2 as provided by equation (3.7).

$$a_{1,n} = a_{3,n-1}, \tag{3.7a}$$
$$a_{2,n} = a_{4,n-1}, \tag{3.7b}$$
$$a_{3,n} = a_{4,n-1}A_1, \tag{3.7c}$$
$$a_{4,n} = a_{3,n-1} + a_{4,n-1}A_2. \tag{3.7d}$$

This way it is possible to reduce the computation time by small margin.

4 Optimization Based Control

Optimization-based control represents a contemporary mathematical control methodology employed to address control problems by optimizing a loss function to derive optimal control inputs. This approach encompasses a diverse array of optimization techniques, including linear programming, nonlinear programming, gradient descent, and more. The critical element in optimization-based control is the judicious selection of an appropriate optimization method, informed by a comprehensive analysis of the characteristics of the loss function and the manner in which constraints should be managed. Extensive mathematical modeling and simulations precede the practical implementation of optimization-based control in real-world scenarios. The loss function needs to be created as presented in equation (4.1).

$$Cost = \min \int_0^T (y(t) - r(t))^2 \, dt, \tag{4.1a}$$

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{4.1b}$$

$$y(t) = Cx(t), \tag{4.1c}$$

$$u(t) = \frac{p_1}{2}\left(1 + \tanh\left(p_2\left(\frac{t}{T_f} - p_3\right)\right)\right). \tag{4.1d}$$

Here for simplicity the parameter p_3 of u(t) would be written in terms of p_1 and p_2. We consider u(0) $= \epsilon$, Which is very small(in order of 10^{-2}). We put t=0,

$$u(0) = \epsilon = \frac{p_1}{2}(1 + tanh(-p_2 p_3),$$

We obtained p_3 as

$$p_3 = \frac{-1}{p_2} \arctan(\frac{2\epsilon}{p_1} - 1).$$

However, it's important to note that the arctanh(z) function exists only when z falls within the range of -1 to 1. This condition imposes restrictions on p_1 as outlined in Equation (4.2).

$$p_1 > \epsilon. \tag{4.2}$$

We have a constraint on the parameter p_1. The reference, which is similar to the one used in Flatness-Based Control.

$$r(t) = \frac{1}{2}\left(1 + \tanh\left(\frac{15t}{T_f} - 3\right)\right). \tag{4.3}$$

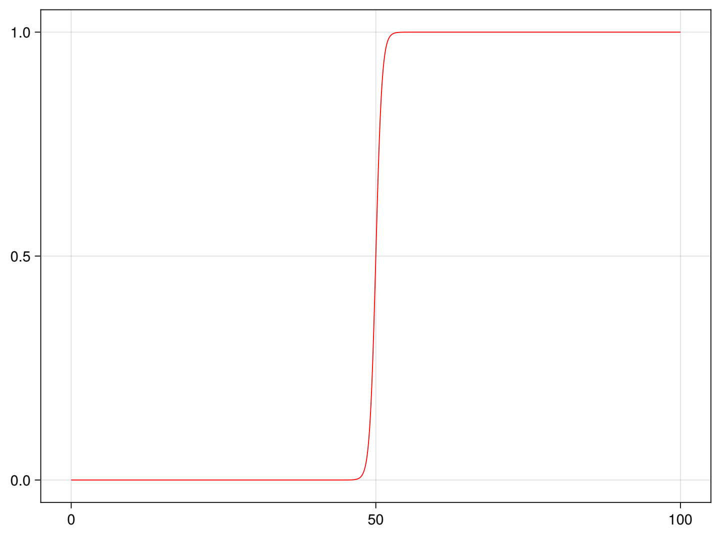


Figure 11: Generalized parameterized tanh.

As we've introduced the equation with parameterized inputs in (4.1), our next step involves investigating the properties of the hyperbolic tangent (tanh) function and the impact of its various parameters (represented as p_1, p_2, and p_3) on system optimization. We will now delve into how each of these parameters influences the system.

1.p_1 Variation

This parameter controls the signal's amplitude and the energy it provides, allowing it to compensate for energy loss during the process.

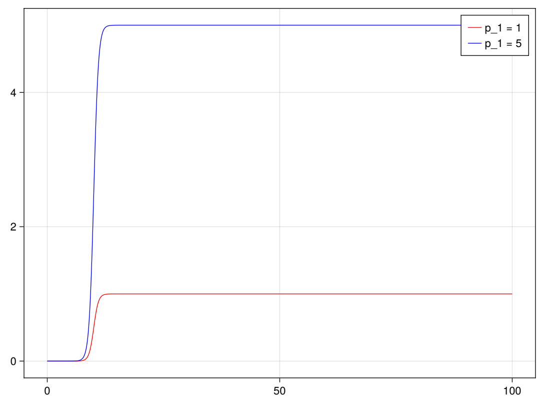
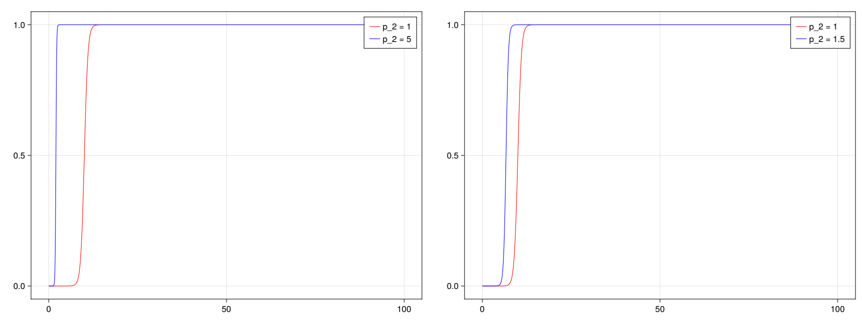


Figure 12: p_1 Variation from 1 to 5.

2.p_2 Variation

This parameter holds paramount significance as it profoundly influences the system's slope, often described as its steepness. The derivative of the system is highly dependent on this factor.



(a) $p_2 = 1$ and 5 (b) $p_2 = 1$ and 1.5

Figure 13: p_2 Variation.

3.p_3 Variation

It impacts the system's delay by causing the input to shift further to the right as it increases.

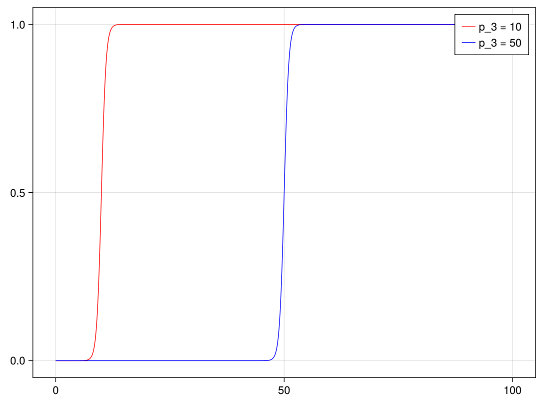


Figure 14: p_3 Variation from 10 to 50.

4.1 Optimization Methods

We will now delve into the optimization techniques employed in the project. In general, a variety of techniques were considered, including Simulated Annealing, Particle Swarm Optimization, Conjugate Gradient, and others. However, only a select few were ultimately applied due to their superior performance. Each technique will also be individually assessed by applying them separately to the system, rather than being applied simultaneously.

We will begin by discussing the basic polynomial optimization of the quadratic convex function mentioned in Equation (4.4). We will explore the impact of eigenvalues on system convergence using various optimization techniques, for more detail please refer[1].

$$f(x) = \frac{1}{2}x^T Hx + b^T x. \tag{4.4}$$

Here basically the gradient descent algorithm would be used with few additional steps. We will explore how it's possible to associate any gradient-based technique with a polynomial that characterizes the rate at which the method converges. The next position is reached as,

$$x_{t+1} = x_t - \frac{2}{L+\mu}\nabla f(x_t). \tag{4.5}$$

In equation (4.5), L and μ are max and minimum eigen value of H respectively. Here we need to look into the convergence rate of equation in terms of eigenvalues and number of iteration(t).

$$\|x_{t+1} - x^\star\|_2 <= \max_{\lambda\in[\mu,L]} |(1 - \frac{2}{L+\mu}\lambda)^{t+1}|_2 \|x_0 - x^\star\|_2.$$

Here $x^\star$ is the minima, x_0 is starting point. The convergence factor is given by

$$P_{t+1}^{GD} = (1 - \frac{2}{L+\mu}\lambda)^{t+1}.$$

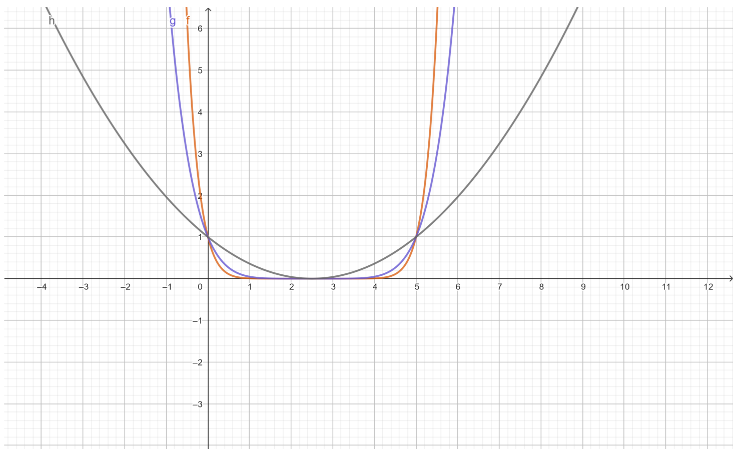


Figure 15: Residual polynomial with t =1 ,5and 9[1].

As depicted in Figure 15, it's evident that the residual polynomial decreases with each successive iteration. This decrease is visually represented by the parabolic shape becoming narrower as the number of iterations increases. Such behavior aligns with our expectations.

$$x_{t+1} - x^\star = P_{t+1}(H)(x_0 - x^\star).$$

Below, significant conclusions can be derived concerning the convergence rate.

- It depends on the polynomial approximation $p_{t+1}(H)$, which in turn depends on optimization method.
- The other term i.e. $x_0 - x^\star$ depends on initial value x_0. If it is selected close to actual solution then convergence would be faster.
- Various methods are used to create the polynomial approximation but the most common one is Chebyshev.

Polyopt

Polyopt[14] is a versatile optimizer that defies easy categorization as either a global or local optimizer. It achieves its robustness by inherently incorporating two distinct optimization algorithms: ADAM and

BFGS (Broyden–Fletcher–Goldfarb–Shanno algorithm). Let's briefly explore both of these algorithms.

ADAM

The ADAM optimizer represents an enhanced iteration of the stochastic gradient descent (SGD) algorithm, and it made its debut in 2014, thanks to the work of Diederik P. Kingma and Jimmy Leiba[15]. Its label, 'ADAM,' stands for 'Adaptive Moment Estimation.' One of its primary advantages is its reliance solely on first-order gradients, which translates to reduced memory requirements and improved run-time efficiency. This algorithm calculates the exponential moving average of gradients, leveraging two critical parameters, namely $\beta_1$and β_2, to control the decay rate of this moving average. In essence, ADAM amalgamates key aspects of the Momentum and RMSP techniques, both of which are elaborated upon below.

1. Momentum[16]

ADAM optimization builds upon the concept of 'exponentially weighted averages,' harnessing their power to expedite the gradient descent process. Exponentially weighted averages, often referred to as EMA(Exponential Moving Average), constitute a technique for assigning weights to data points over time. This method grants greater significance to recent data, gradually diminishing the influence of older data as it fades into the past.Let's look into the equations involved.

$$update = \alpha m_t.$$

The 'update' indicates the distance between the current point and the next point, with α representing the step size that governs the transition to the subsequent point. This relationship is visually represented by the expression for m_t in the following equation.4.6

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla f_w. \tag{4.6}$$

Where t is current time, ∇f_x is gradient of objective function(f). β_1 is the hyper-parameter, whose value is chosen close to 1.

$$w_{t+1} = w_t - \alpha m_t.$$

This way the next iteration is computed.

2. RMSP(Root mean square propagation)

RMSP is also an adaptive optimization algorithm, it is the upgraded version of AdaGrad. Here we have v_t which is sum of square of past gradients as seen in (4.7).(Note $v_0 = 0$)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla f_w)^2. \tag{4.7}$$

So the next point is given by

$$w_{t+1} = w_t - \frac{\alpha_t}{\sqrt{v_t + \epsilon}} \nabla f_w.$$

where w_t is the weight at time t.

Now combining momentum and RMSP provides ADAM algorithm as follows

- Calculate m_t and v_t from the equation 4.6 and 4.7.

- Now compute the weight(w_t) using following equation

$$w_t = w_{t-1} - \alpha \frac{\tilde{m_t}}{\sqrt(\tilde{v_t}) + e}.$$

Where $\tilde{m_t}$ and $\tilde{v_t}$ are given as

$$\tilde{m_t} = \frac{m_t}{(1 - \beta_1^t)},$$

$$\tilde{v_t} = \frac{v_t}{(1 - \beta_2^t)}.$$

here β_1^t and β_2^t are the hyper parameter at time t. An beautiful exapmple of ADAM optimizer is shown in figure 16, ADAM demonstrate the best optimization as the loss is minimum.

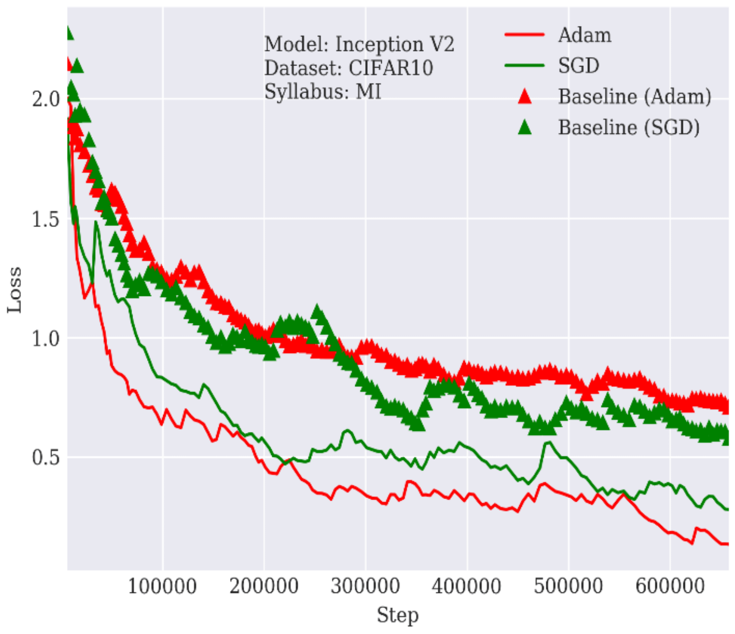


Figure 16: Example of ADAM on curriculum training[2].

BFGS(Broyden–Fletcher–Goldfarb–Shanno)

The BFGS[17][18] algorithm is an optimization method that relies on approximating the Hessian matrix of the given objective function, often referred to as the loss function. This approximation is derived from gradient evaluations (or approximate gradient evaluations), leading to a notable reduction in computational complexity, typically on the order of O(n^2).
We have a function f(x), $x \in \mathbb{R}^n$. x does not have constraints. The algorithm starts at x_0(Which would be provided by ADAM in case of Polyopt). Search direction $\vec{p_k}$ is given by negative gradient of f, as shown in equation 4.8 .

$$B_k \vec{p_k} = -\nabla f(x_k). \tag{4.8}$$

Where B_k[19] is approximation to hessian matrix at x_k, which is updated iterative at each stage, and ∇ f(x_k) is gradient at x_k. Then Line search is used in the direction of p_k by minimizing f(x_k + γ p_k). Here γ_k should satisfy wolfe conditions[20]. Thus reaching next point which is x_{k+1}. Please note here γ is scalar.

Let s_k=$x_{k+1} - x_k$ and y_k=$\nabla f(x_{k+1})$-$\nabla f(x_k)$, we have approximate hessian B_{k+1}[16] given by equation 4.9.

$$B_{k+1} = B_k + \frac{\mathbf{y}_k \mathbf{y}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}} \mathbf{s}_k} - \frac{B_k \mathbf{s}_k \mathbf{s}_k^{\mathrm{T}} B_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}} B_k \mathbf{s}_k}. \tag{4.9}$$

Please note B_{k+1} is the approximation of hessian. For each iteration value of x_{k+1}, f(x_{k+1}) can be checked for reaching within the desired loss interval.
We know ADAM and BFGS are applied one after the other in order to implement "Polyopt" Algorithm.

Gradient Descent

It is the most widely employed optimization algorithm for minimizing or maximizing an objective function. The process begins with an initial starting point, denoted as x_0. Subsequently, the algorithm progresses in the direction of steepest descent, following the gradient, to locate the maximum or minimum of the function. This principle can be visualized in Figure 17, where the contours resemble a valley with a peak at the center. The arrow within the diagram represents the step size taken in each iteration.

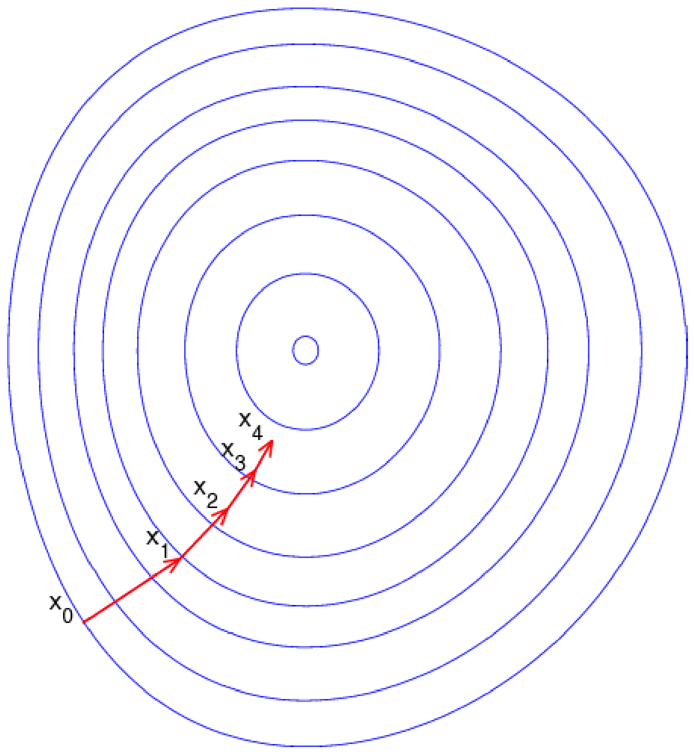


Figure 17: Gradient Descent contours[3].

Now let's look into the mathematics behind gradient descent.

$$x(n+1) = x(n) - \gamma \cdot \nabla j(x). \tag{4.10}$$

In Equation 4.10, x(n+1) represents the next point, while x(n) corresponds to the current point. Here, $\nabla j(x)$ signifies the gradient of the objective function j(x), and γ denotes the step size in the direction of the negative gradient. This iterative process is repeated until the minimum of the function is attained.

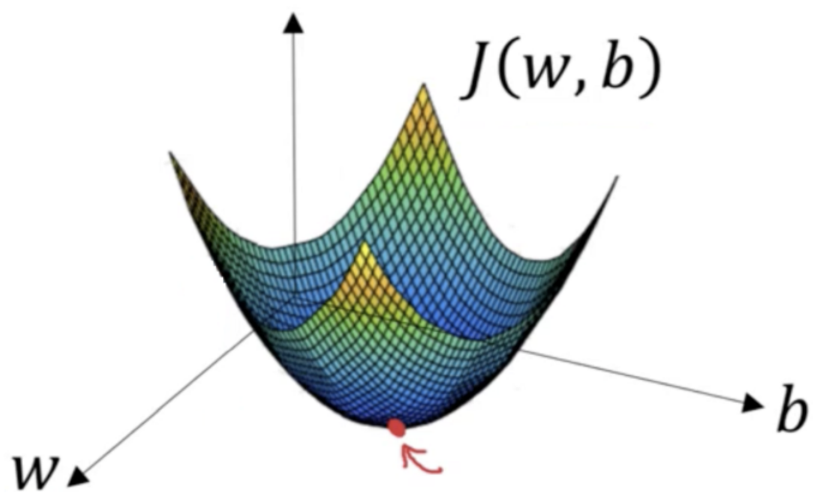


Figure 18: Example Gradient Descent in 2D[3].

At times, the step size of the system is also referred to as the learning rate. A larger step size can indeed expedite the journey to the minimum, but it carries the risk of overshooting or potentially neglecting the global minimum.

Bound Optimization By Quadratic Approximation(BOBYQA)

Initially, we will delve into the approach known as the Generalized Quadratic Approximation, often referred to as the 'Quadratic Approximation with Trust Region' method.
In this context, we encounter a function f(x) that necessitates minimization for $x \in \mathbb{R}^n$. Here we have following methods for finding out the minima:-

- Initialize the optimization process with the starting point x_0.
- Create a quadratic polynomial function to approximate the objective function, f(x).
- Note that this approximation closely matches the actual objective function only within a predefined trust region.
- Move to the point with the minimum value of this trust region, which becomes the new starting point x_1.
- Recalculate the quadratic polynomial approximation of the objective function around x_1.
- Define a new trust region and compute the minimum point within it.
- Repeat this iterative process until the desired loss value is achieved.

Now, we shall elucidate this concept using Figure 19. The figure displays the contours of the function f(x). Beginning from the point w_k, we embark on the task of approximating the region using a quadratic polynomial. Now, we will explore the specifics of this approximation procedure. Choose a quadratic polynomial of the form

$$w(x) = ax^2 + bx + c.$$

Now select three points x_1, x_2 and x_3 consecutively, Calculate corresponding objective value on those points. so points would be (x_1,f(x_1)),(x_2,f(x_2)) and (x_3, f(x_3)). Now we have the following equations

$$f(x_1) = ax_1^2 + bx_1 + c,$$

$$f(x_2) = ax_2^2 + bx_2 + c,$$

$$f(x_3) = ax_3^2 + bx_3 + c.$$

We have three equation and three variable, it can be easily solved to obtain values of a,b and c.

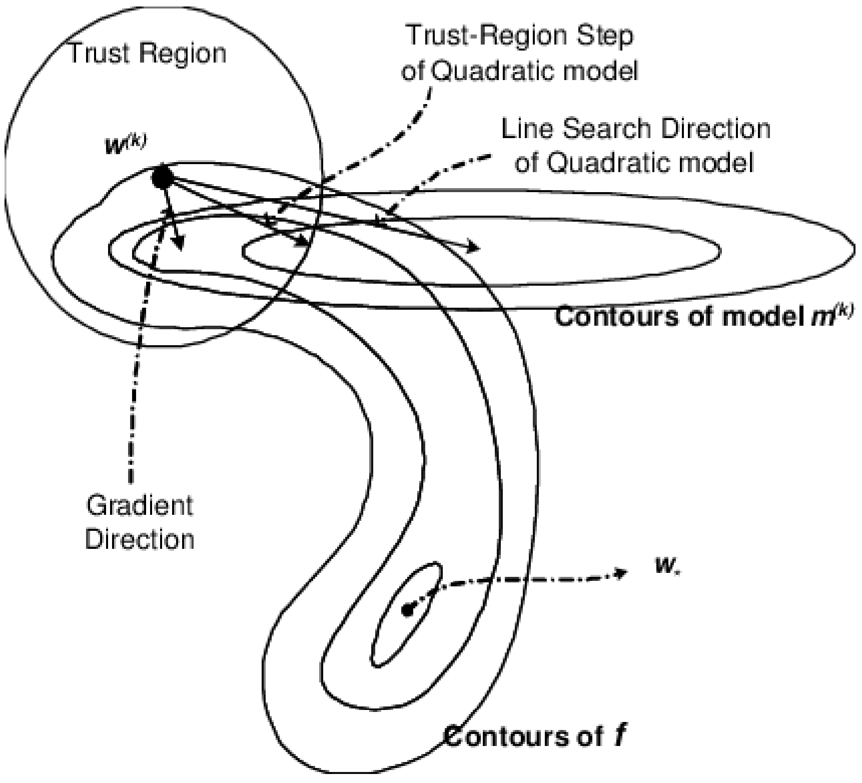

Trust Region
Trust-Region Step of Quadratic model
Line Search Direction of Quadratic model
$w^{(k)}$
Contours of model $m^{(k)}$
Gradient Direction
w_*
Contours of f

Figure 19: Example, Optimization using quadratic approximation[2].

After computing the quadratic polynomial approximation, establish trust region boundaries within which the approximation is highly accurate. Proceed by moving inside the trust region along the direction of the negative gradient, which signifies the path of descent (this step is particularly straightforward when dealing with a quadratic polynomial). Upon reaching a new point, create another quadratic polynomial within this region, and continue this iterative process until you attain the target loss value.
In Figure 19, we can observe three different directions(line search and trust region of quadratic model, gradient descent of f) for selecting the next point. It's evident that the trust region step based on the quadratic model provides the most accurate prediction for the next step. Moving in this direction ensures a correct and efficient optimization process.

Subsequently, We shall present a succinct overview of 'Bound Optimization by Quadratic Approximation,' as originally introduced by Powell [21]. This method is designed to ascertain the minimum value of a loss function, denoted as f(x), with the variable x belonging to the real vector space $\mathbb{R}^n$, while aiming to fulfill the following objective.

$$a_i < x_i < b_i \text{ where i=1,2,3...n} \tag{4.11}$$

Here derivative of f is not required, This is iterative method where K denotes number of iteration and n denotes number of variables, We require quadratic approximation Q_k(x) where $x \in \mathbb{R}^n$.

$$Q_k(y_j) = f(y_j) \text{ where j=1,2,3...m} \tag{4.12}$$

where m is constant integer in the interval $\left[n+2, \frac{(n+1)(n+2)}{2}\right]$, and it should be chosen by user. Now we have x_k a point in the set $y_j : j = 1, 2, ..., m$, i.e inside of quadratic approximation that satisfies

$$f(x_k) = \min\{f(y_j) : j = 1, 2, \ldots, m\} \tag{4.13}$$

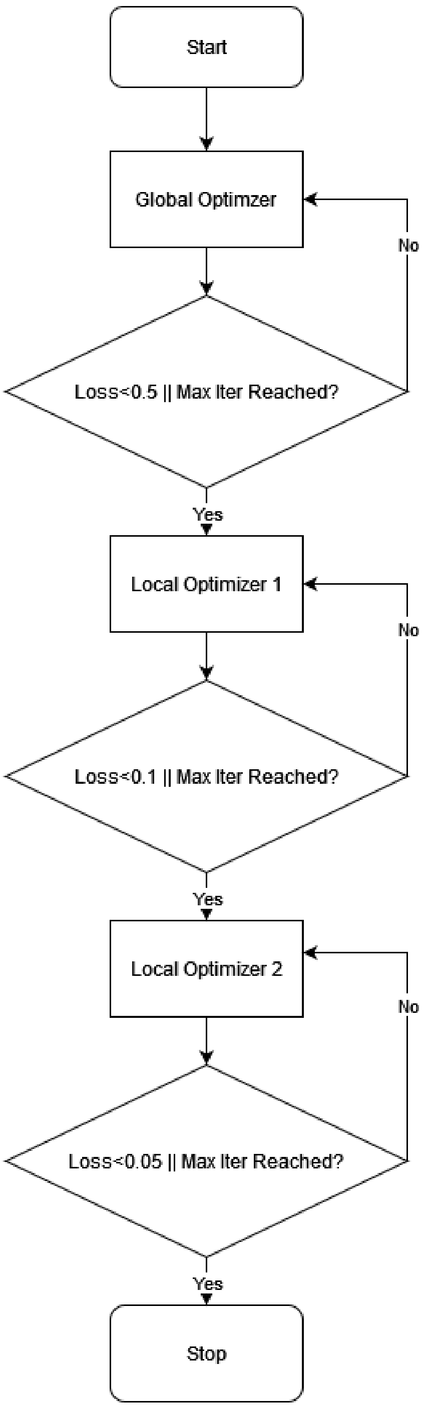
We also have Δk which is called as "trust region radius", if certain conditions are achieved then termination occurs at the k$^{\text{th}}$ iteration, otherwise we move a step d_k in the trust region., such that x= x_k+d_k lied within the bounds(4.11).

$$x_{k+1} = \begin{cases} x_k, & \text{if } f(x_k + d_k) \geq f(x_k), \\ x_k + d_k, & \text{if } f(x_k + d_k) < f(x_k). \end{cases} \tag{4.14}$$

Similarly $\Delta(k+1)$ and Q_{k+1} are generated in the next iteration. For more details on the above method please have a look here[21].

4.2 Implementation

As previously discussed in the preceding section, optimization techniques can be broadly categorized as either global or local optimization. Global optimization encompasses a broader search space, aiming to locate minima across the entire interval, whereas local optimization focuses on identifying the optimal minima point in proximity to the algorithm's starting point.



Start
Global Optimzer
No
Loss<0.5 || Max Iter Reached?
Yes
Local Optimizer 1
No
Loss<0.1 || Max Iter Reached?
Yes
Local Optimizer 2
No
Loss<0.05 || Max Iter Reached?
Yes
Stop

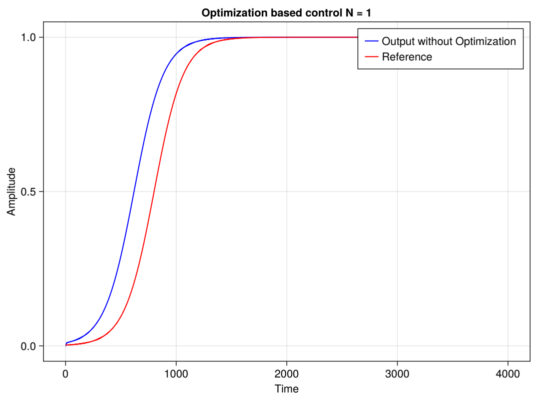
Figure 20: Flow Chart Optimization based Control

Flowchart Explanation

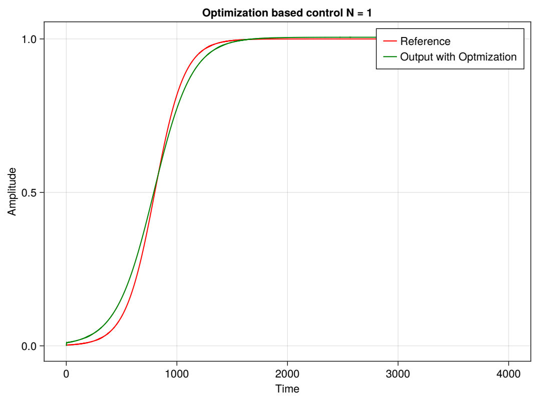
As is well-known, optimization processes require a comprehensive exploration of the entire landscape to approach the global minima. To achieve this, we employ a global optimizer, equipped with stopping criteria based on an overall loss value below 0.5 or reaching the maximum iteration limit. Following the global optimization phase, our attention shifts to a local optimizer, which fine-tunes the search within a confined area. The initial parameter values from the global optimization are passed on to the local optimizer. This local optimization continues until the overall loss value drops below 0.1 or the maximum iteration limit is reached. In a similar vein, another local optimizer comes into play, aiming for a loss value below 0.05 as its stopping criteria. This strategy optimizes resource utilization, significantly reducing runtime, and effectively leverages the entire parameter interval within the constraints.

Results with Optimization based control

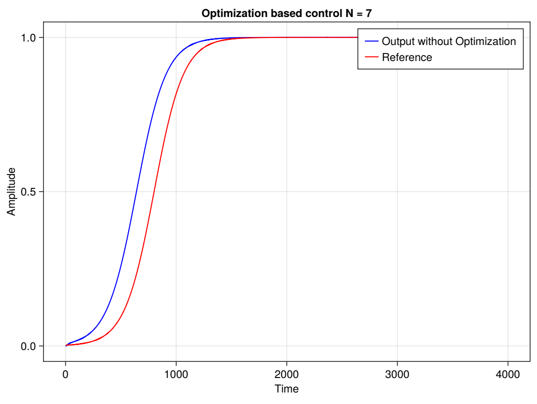
As per the methodology outlined in the flowchart (Figure 20), let us proceed to examine the solutions obtained by employing all three optimization methods concurrently. The implementation results are depicted in Figure 21 and Figure 22.



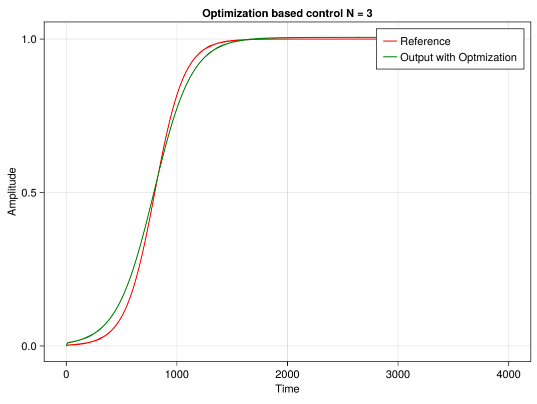
(a) Output and reference N=1



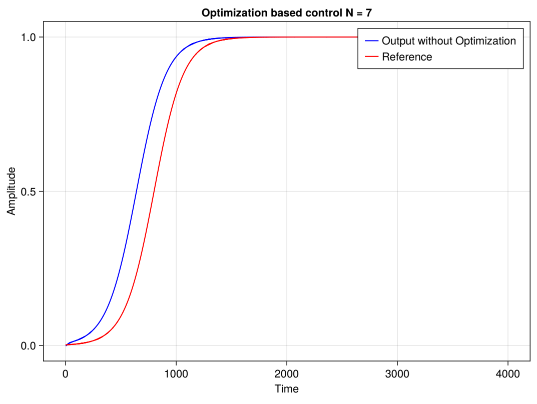
(b) Optimized and reference for N=1



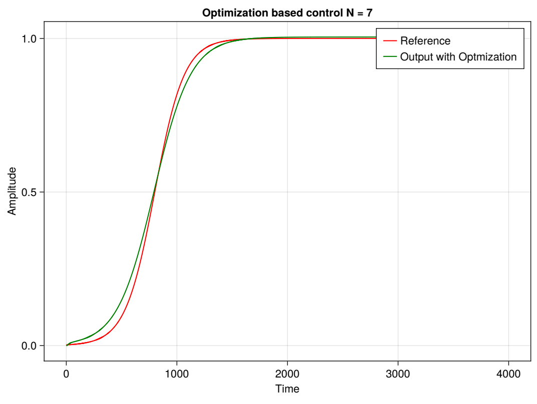
(c) Output and reference for N=3



(d) Optimized and reference for N=3

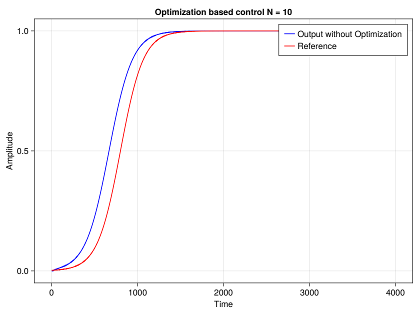


(e) Output and reference for N=7

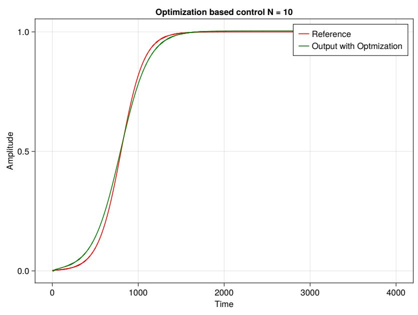


(f) Optimized and reference for N=7

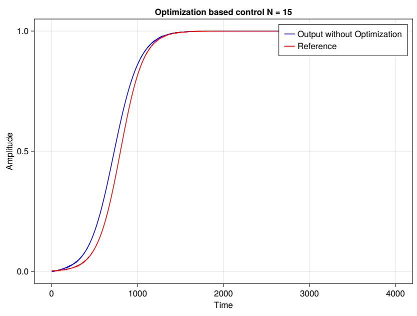
Figure 21: Original Output and Optimized output(N=1,3,7).



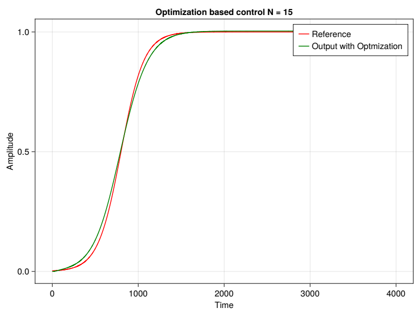
(a) Output and reference for N=10



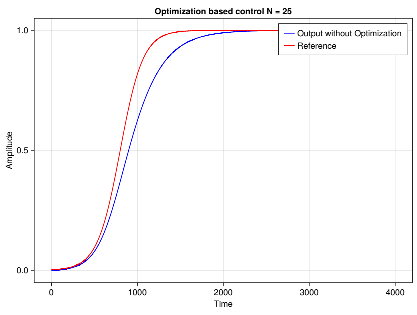
(b) Optimized and reference for N=10



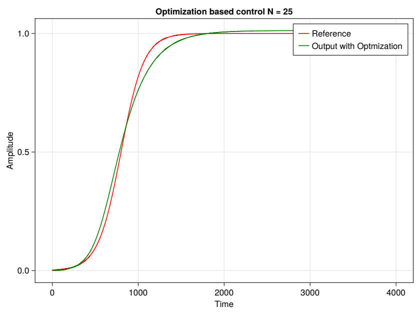
(c) Output and reference for N=15



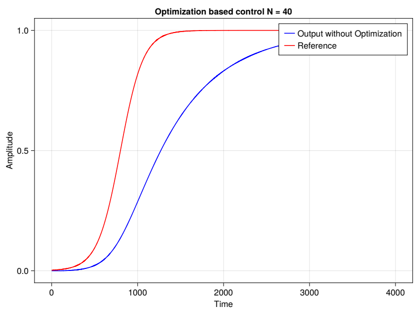
(d) Optimized and reference for N=15



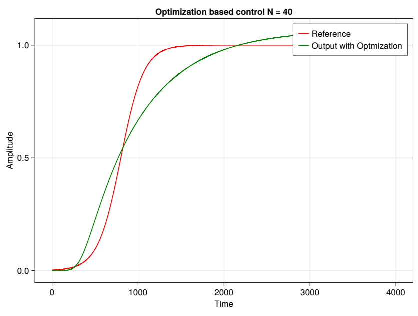
(e) Output and reference for N=25



(f) Optimized and reference for N=25



(g) Output and reference for N=40



(h) Optimized and reference for N=40

Figure 22: Original Output and Optimized output(N=10,15,25,40).

Results with Individual Optimization techniques

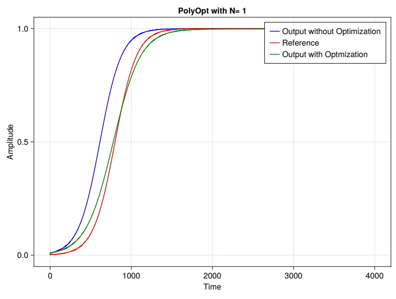
After conducting a comprehensive examination of optimization using all three methods concurrently, the outcomes proved to be quite satisfactory, even when addressing higher-order systems. However, to gain a deeper insight into the impact of system characteristics on optimization and to discern which parameters are influenced by each optimization technique, it is imperative to investigate each method individually. But before delving further into the analysis of results, let us explore the libraries employed for optimizing the system. In this regard, two Julia libraries, NLOPT[22] and Optim[23], have been utilized. These libraries offer various optimization techniques, including Polyopt, Gradient Descent, LNBOBQYA, and NELDERMEAD.

Nlopt is widely used optimizer with available packages in various languages, it's julia package is referred as "OptimizationNLopt". Optim offers a range of optimization techniques to suit different scenarios, It's package name is ""OptimizationOptimJL".
Here are some of the optimizer classes available in both the packages

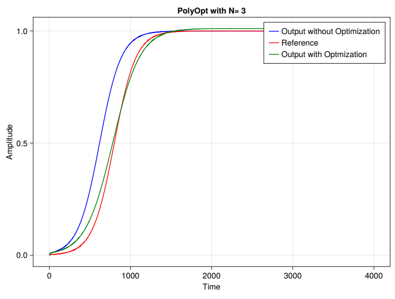
1. **Local Optimizer:** This optimizer halts at the nearest local minima or maxima. For example, you can use `Optim.IPNewton()`.
2. **Derivative-Free Optimizer:** When calculating derivatives proves challenging, this optimizer class can be employed. It may be less efficient compared to other methods. For instance, consider using `Optim.NelderMead()[24]` and `NLopt.LN_BOBYQA()[21]`.
3. **Gradient-Based Optimizer:** This class calculates the gradient of the function. You can apply it with `Optim.GradientDescent()` and `NLopt.LD_LBFGS()`.
4. **Hessian-Based Second-Order Optimizer:** When the second-order Hessian can be calculated, this method offers faster convergence. You can employ it using `Optim.Newton()`.
5. **Global Optimizer:** For optimization tasks covering a large area, this optimizer is suitable. For instance, try `Optim.GlobalOptimizer()`, `Optim.ParticleSwarm()[25]` and `NLopt.GN_DIRECT()`.

Upon observing Figure 23, it becomes apparent that the obtained results closely match the reference, showcasing a high degree of accuracy. However, as the system order is increased, a corresponding increase in loss is observed, resulting in deviations from the reference signal. It is crucial to highlight that in higher-order systems, the inherent system dynamics prevent rapid rise times due to the sequential charging of multiple capacitors. We will delve into this topic in more depth after examining the results obtained with different optimization techniques.

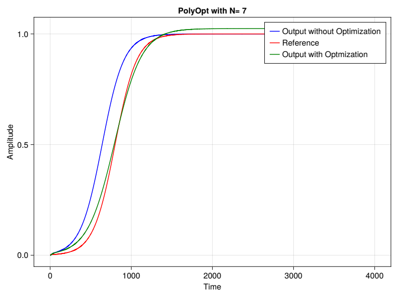
1.POLYOPT



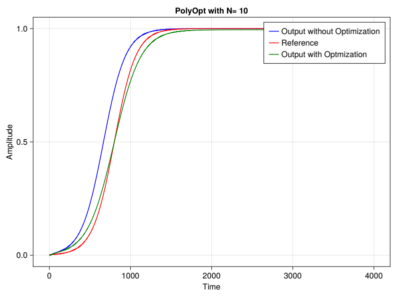
(a) N=1



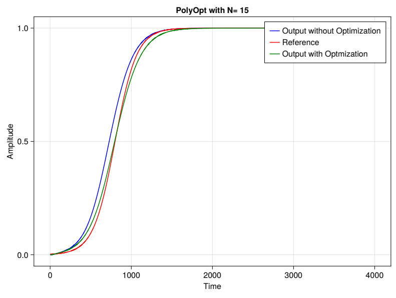
(b) N=3



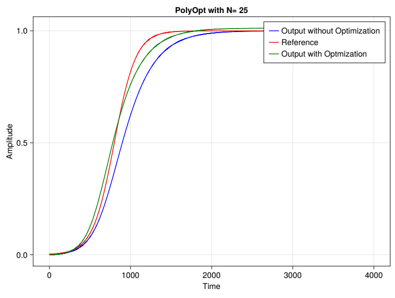
(c) N=7



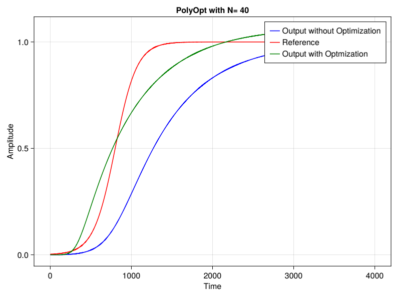
(d) N=10



(e) N=15



(f) N=25



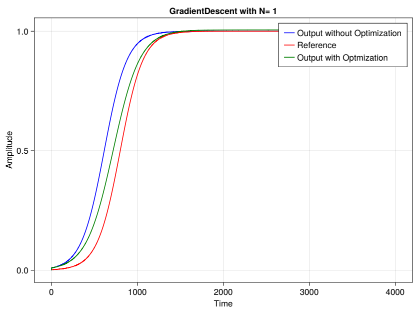
(g) N=40

Figure 23: OBC with Polyopt optimizer.

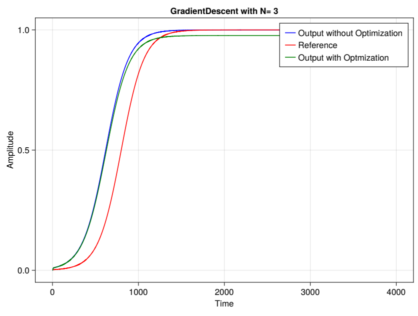
N	p_1	p_2	Loss
1	0.9981226776848832	11.825751665065004	0.0005960718164400807
3	1.010848701050255	11.848403745172874	0.0005645952276773379
7	1.0242833233069517	12.022182633356962	0.0006810261562369537
10	0.9948569610328094	12.341434068499716	0.00040191266472029545
15	0.9998119191759315	13.53399370985718	0.00024060669334838265
25	1.0122601178444737	18.24262818944882	0.000685913708370779
40	1.069362548769509	50.550881343957364	0.0068778432998186124

Table 1: p_1, p_2 and Loss values with Polyopt.

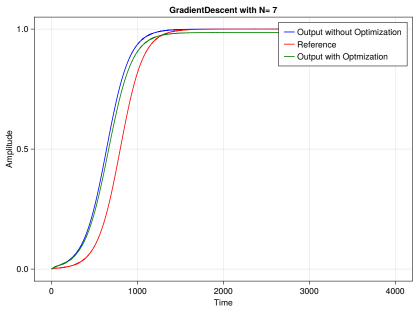
2.Gradient Descent



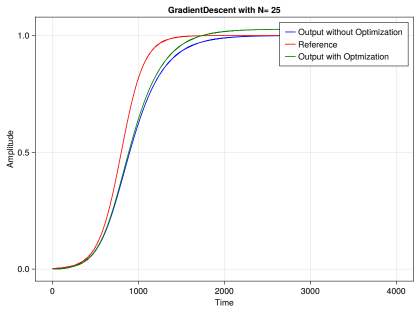
(a) N=1



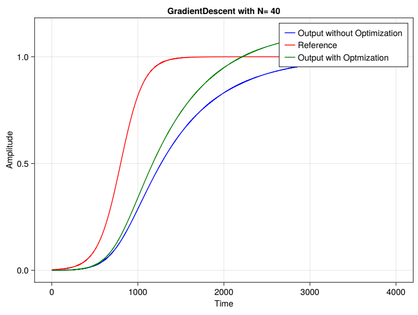
(b) N=3



(c) N=7



(d) N=25



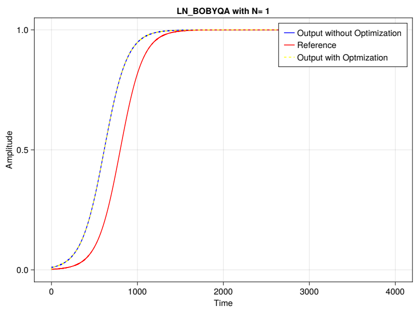
(e) N=40

Figure 24: OBC with Gradient Descent.

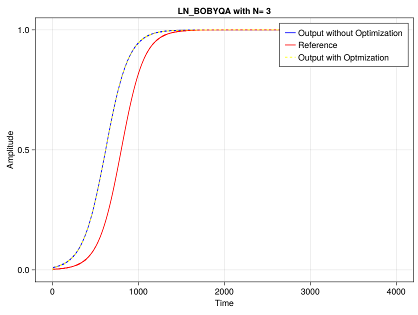
N	p_1	p_2	Loss
1	1.005470800042144	12.822764319431185	0.0022004928863047376
3	0.9764890280385452	14.860373684319153	0.008836471638452997
7	0.9857771648865551	14.454596373318168	0.005367572680476824
10	N/A	N/A	N/A
15	N/A	N/A	N/A
25	1.027553549202767	15.115404683520566	0.0036313195892379523
40	1.1350612732020458	15.80160234436706	0.038961183170550105

Table 2: p_1, p_2 and Loss values Gradient Descent

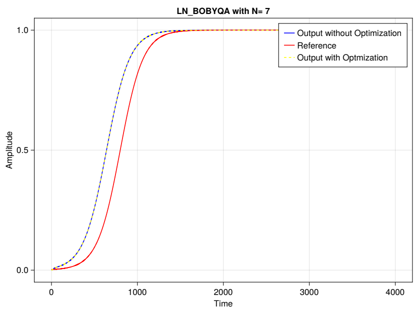
3.LNBOBQYA



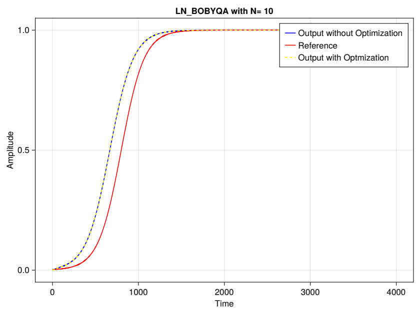
(a) N=1



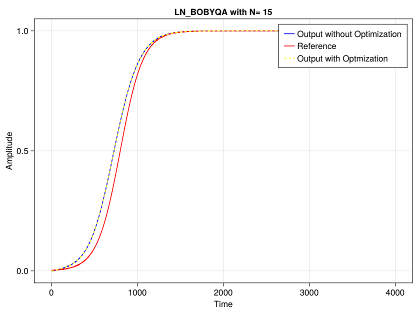
(b) N=3



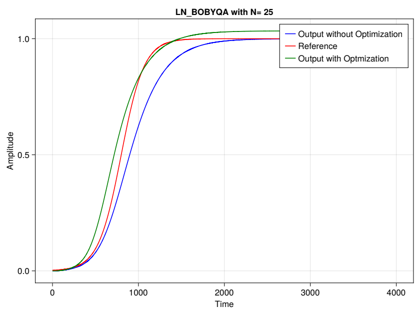
(c) N=7



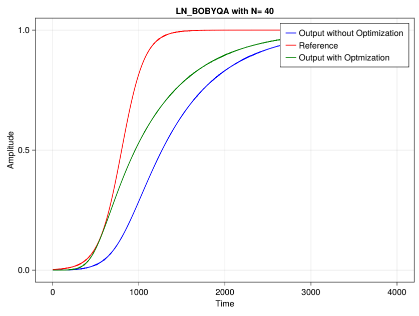
(d) N=10



(e) N=15



(f) N=25



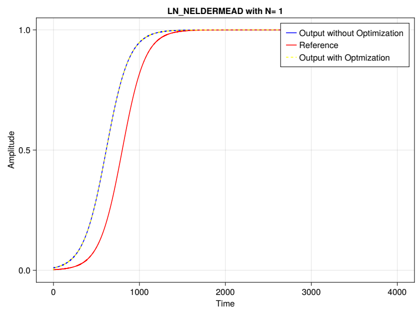
(g) N=40

Figure 25: OBC with LNBOBQYA optimizer.

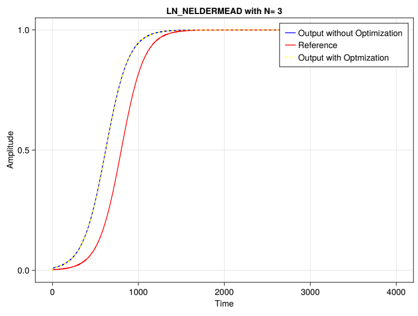
N	p_1	p_2	Loss
1	1.0	15.0	0.010509620730316442
3	1.0	15.0	0.009968500870898863
7	1.0	15.0	0.007732198200598992
10	1.0	15.0	0.005353448332805369
15	1.0	15.0	0.001508339871321258
25	1.0337011769845768	20.483619969101273	0.002223454295302859
40	1.0	28.410000000000004	0.0163977973349769

Table 3: p_1, p_2 and Loss values with LNBOBQYA optimizer.

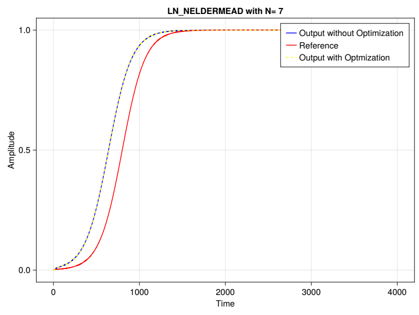
4.NELDERMEAD



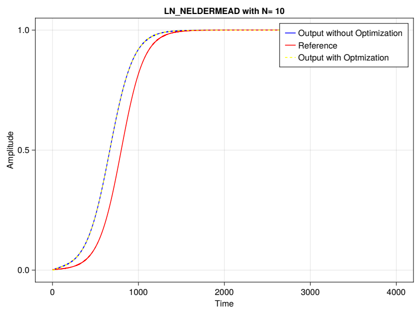
(a) N=1



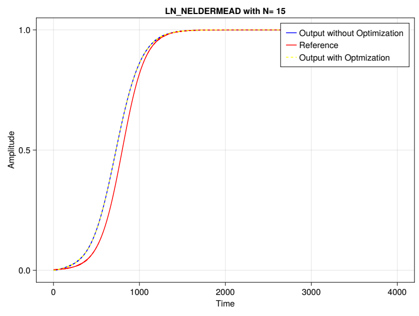
(b) N=3



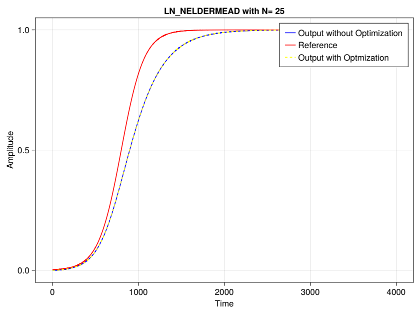
(c) N=7



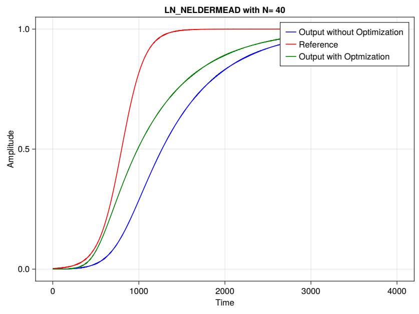
(d) N=10



(e) N=15



(f) N=25



(g) N=40

Figure 26: OBC with LNNELDERMEAD optimizer.

N	p_1	p_2	Loss
1	1.0	15.0	0.010509620730316442
3	1.0	15.0	0.009968500870898863
7	1.0	15.0	0.007732198200598992
10	1.0	15.0	0.005353448332805369
15	1.0	15.0	0.001508339871321258
25	1.0337011769845768	20.483619969101273	0.002223454295302859
40	1.0	26.175	0.018620714376337392

Table 4: p_1, p_2 and Loss values with LNNELDERMEAD optimizer.

Following are some of the observation based on above figures
Upon observing Figure 23, figure 24, figure 25 and 26 it becomes apparent that the obtained results closely match the reference, showcasing a high degree of accuracy. However, as the system order is increased, a corresponding increase in loss is observed, resulting in deviations from the reference signal. It is crucial to highlight that in higher-order systems, the inherent system dynamics prevent rapid rise times due to the sequential charging of multiple capacitors. As Seen in Figure 27, Polyopt produces the best results whereas Gradient descent with the worst.

An important observation can be made when examining Table 2. For N=10 and 15, the algorithm fails to achieve the optimization. This is primarily due to the fact that the optim.gradientdescent() function lacks constraints, causing the optimizer to be unable to satisfy the condition on p_1 mentioned in Equation 4.2.

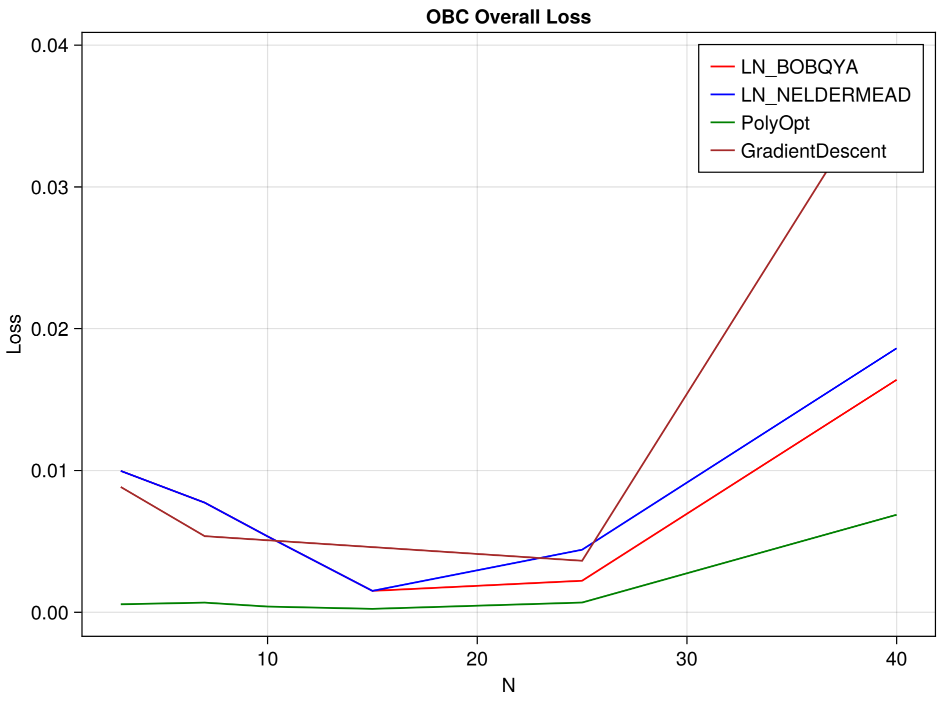
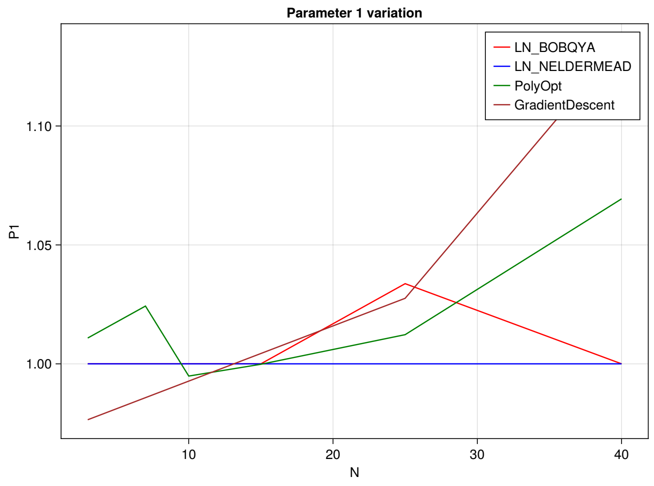


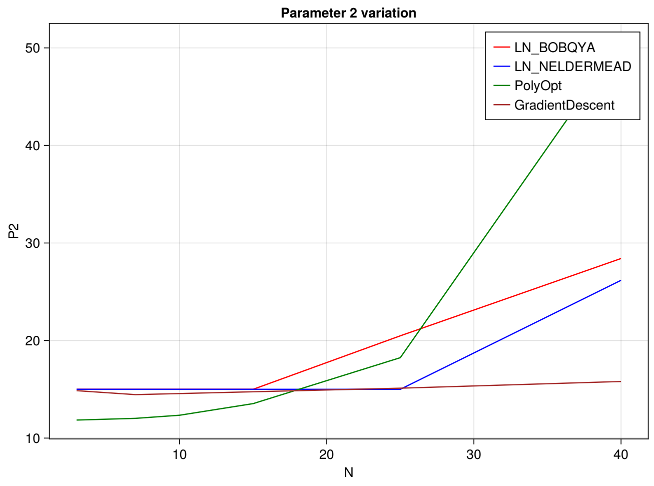
Figure 27: Loss Comparison of all the optimizer.

We shall delve into why Polyopt was able to optimize the system with minimal loss. As observed earlier, p_1 is responsible for amplitude adjustment, p_2 fine-tunes the input's steepness, and p_3 compensates for system delay. While p_1 and p_3 deal with aspects such as energy compensation and system delay comparison, these parameters remain relatively constant for a specific order system. Therefore, they are of lesser importance.

However, as demonstrated in Flatness Based Control (FBC), p$_2$ (steepness) significantly impacts the system's behavior. In Figure 28, Polyopt's ability to precisely manage steepness becomes evident, effectively compensating for the system's inherent sluggishness.



(a) p_1 variation(Amplitude)



(b) p_2 variation(Steepness)

Figure 28: p_1 and p_2 variations for each optimizer.

5 Model Predictive Control

Basic theory of Model Predictive Control
Model Predictive control[26] is an advanced optimization techniques which is quite useful for dynamical system, it works by predicting the future outputs using mathematical representation of the system and optimizing it by controlling the input signal u(t). The accuracy of the system is sensitive to the mathematical model of the system. Thus it helps in optimizing system while making sure the loss value to be minimum, As it is dynamic in nature it also helps the system if disturbances are produced in the system at random time stamp.

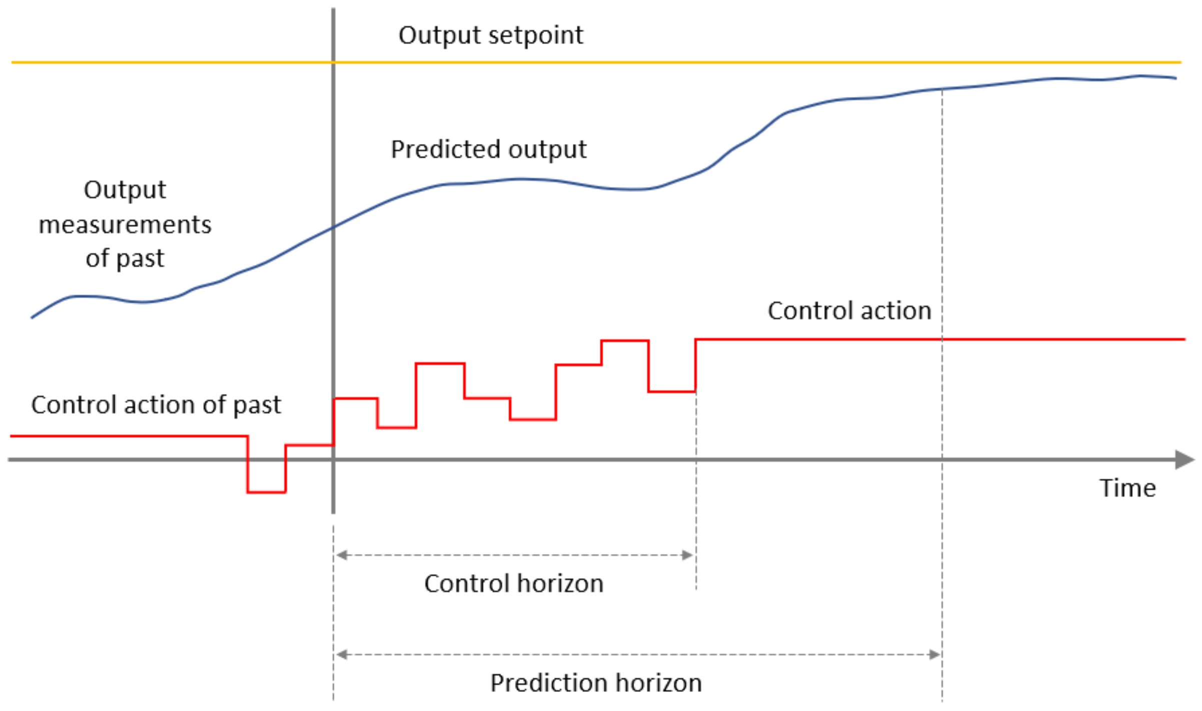


Figure 29: Generalized MPC Description.[4]

Following are some of the terminologies used in model predictive control
1.Control Horizon(M)
It represents the duration during which the control input is actively applied.

2.Prediction Horizon(P)
The prediction horizon denotes the future time span over which optimization is performed to calculate control variables. A longer prediction horizon results in higher computational costs.

3.Control Inputs (u)
The computed input signal is derived by optimizing the system over the prediction horizon.

4.Cost Function / Loss Function
It essentially represents the disparity between the output and the reference signal; its interpretation can also vary based on the desired output state.

5.Constraints

It defines the boundary within which the control input variable can fluctuate, reflecting real-world constraints to prevent MPC from generating impractical control inputs. For instance, in a motor system, this constraint might prevent the generation of excessive current demands that are impractical in reality.

6.Optimization
The process of determining the optimal control variable by minimizing a loss function.

7.Reference Trajectory
It is the desired output.

8.System Model
Model equations describe the system's behavior through input-output relationships, incorporating system states.

5.1 MPC Implementation

Before delving into our own implementation, We will explore the generalized MPC block diagram depicted in Figure 30. MPC comprises an optimizer and a prediction model. The optimizer conducts optimization based on the predicted system output generated by the prediction model, while simultaneously evaluating the produced output to ensure that disturbances are properly accounted for, potentially causing deviations between the predicted and measured outputs.

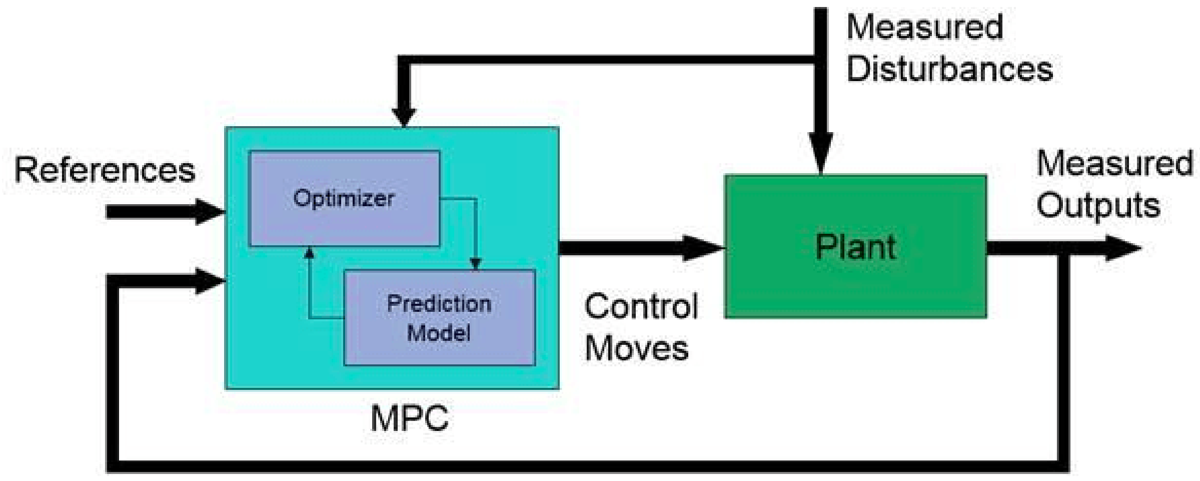


Figure 30: MPC Block Diagram[5]

Let us now proceed to examine the flowchart of the implementation.

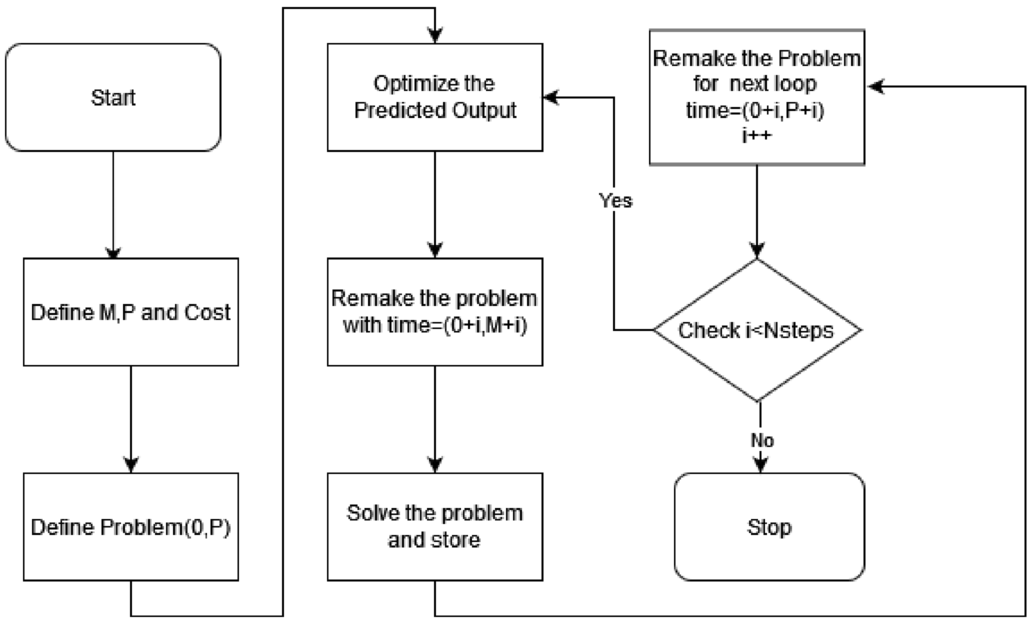

Start
Define M,P and Cost
Define Problem(0,P)
Optimize the Predicted Output
Remake the problem with time=(0+i,M+i)
Solve the problem and store
Remake the Problem for next loop time=(0+i,P+i) i++
Check i<Nsteps
Yes
No
Stop

Figure 31: MPC algorithm flow chart.

Flow Chart Description:

In this context, the optimization process follows a methodology similar to that employed in optimization-based control, with the use of a single optimizer throughout the Model Predictive Control (MPC) scheme. Here as well we use the reference similar to OBC, as shown in equation 5.1.

$$r(t) = \frac{1}{2}\left(1 + \tanh\left(\frac{t}{10} - 5\right)\right). \tag{5.1}$$

Initially, we establish the key parameters: Prediction Horizon (P), Control Horizon (M), and the cost function (L). Subsequently, we formulate the optimization problem within the time interval spanning from 0 to P, utilizing a set of parameterized inputs, as represented in the equation 5.2.

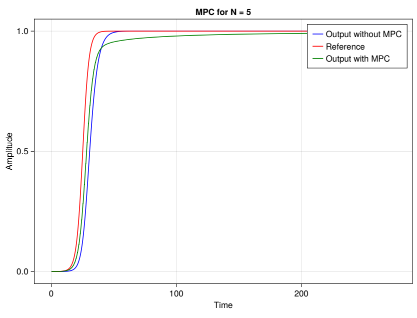
$$U(t) = \sum_{t=0}^{M} \left(p_1 \left(1 + \tanh\left((p_2(t/T) - p_3)\right)\right)\right). \tag{5.2}$$

Here, we possess the optimized parameterized input equation. Our process entails conducting optimization across the entire prediction interval (P). Once the optimized parameters are acquired, the problem necessitates reformulation within the control horizon interval. Subsequently, we resolve the problem by applying the optimized parameters from the prediction horizon interval and retain the outcome. This marks the completion of one cycle of Model Predictive Control (MPC), designed for perpetual execution. In practical implementation, however, it is executed for N steps. The key question that arises pertains to determining the appropriate value for N. An effective approach involves initializing the system with a standard reference input, determining the settling time (T_s), and proceeding with Model Predictive Control (MPC) for a duration of $T_s + 30$ time units. Keep in mind that the choice of N also varies with the control horizon. A longer control horizon requires a smaller N value.

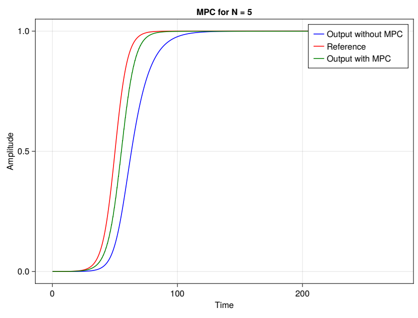
5.2 MPC Results

This section delves into the conventional implementation of Model Predictive Control (MPC) with various combinations of control and prediction horizons to optimize results for diverse scenarios. We employ multiple optimization techniques, considering each individually, and select the one that yields the most favorable results.

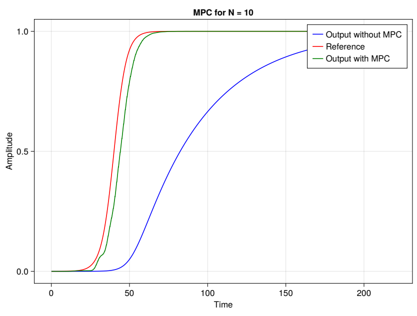
The graph in Figure 32 visually portrays the performance degradation of MPC as the system order increases. For cases where N is less than or equal to 15, the system output effectively follows the reference with minimal loss and high precision. However, when N equals 25, it closely approximates the reference but exhibits oscillatory behavior in the output. A more in-depth analysis of these oscillations will be presented in a subsequent section. When N equals 20, it initially tracks the reference up to a certain point in time, but beyond that point, it significantly deviates, resulting in a notable increase in error.



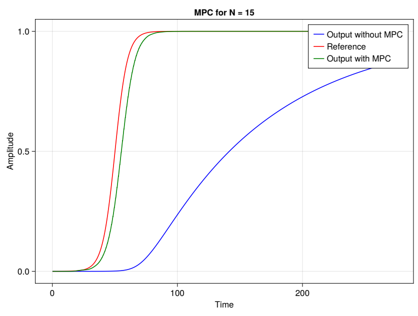
(a) N=3



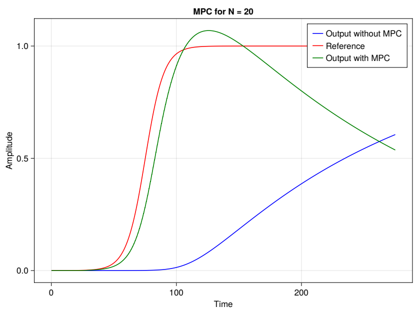
(b) N=5



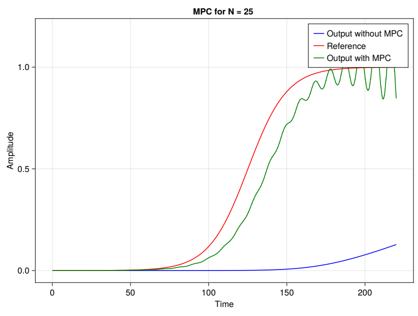
(c) N=10



(d) N=15



(e) N=20

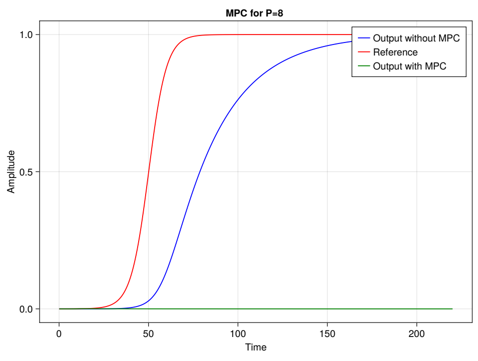


(f) N=25

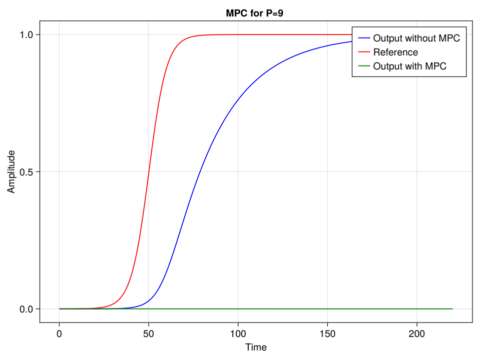
Figure 32: MPC For different order system.

5.3 Prediction Horizon Variation

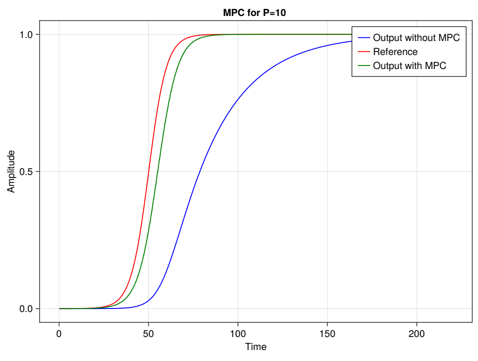
This section delves into the pivotal role of the prediction horizon in feedback-based model predictive control. The prediction horizon, which considers future time stamps, plays a crucial role in forecasting system behavior. Extending its value enhances the system's predictability. It's imperative to highlight that accurate modeling of the system equations is essential because, if the predicted output deviates significantly from the actual output, the effectiveness of the entire MPC optimization process is compromised.
This phenomenon is evident in Figure 33. Here all the observations are taken for N=8(8 cascaded circuits), For prediction horizons less than 9, MPC fails to track the reference, and its performance is inadequate. However, when $P > 9$, MPC functions effectively, closely following the reference. Later, we will delve into why it fails for $P \leq 9$ but works seamlessly for $P > 9$.



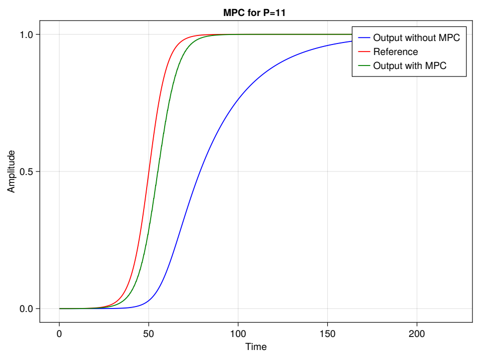
(a) P=8



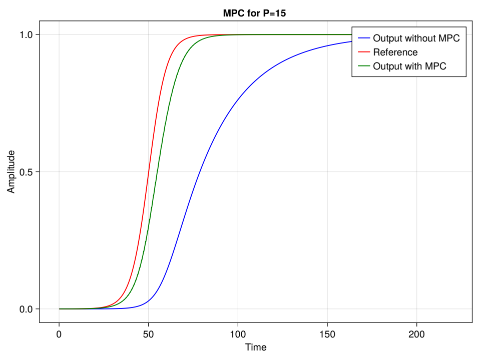
(b) P=9



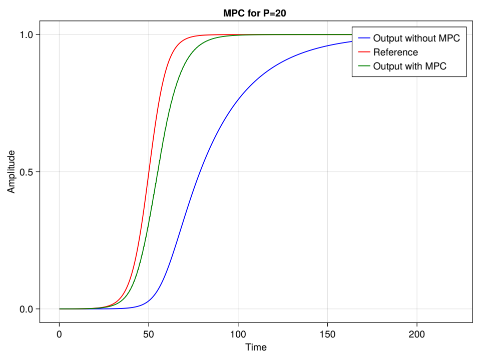
(c) P=10



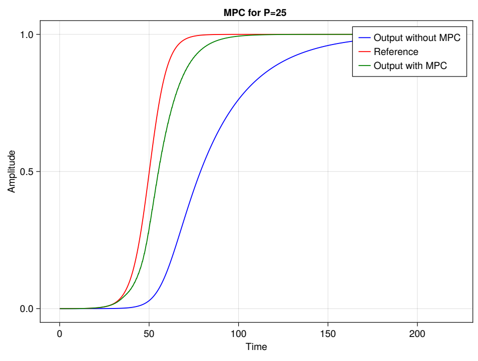
(d) P=11



(e) P=15



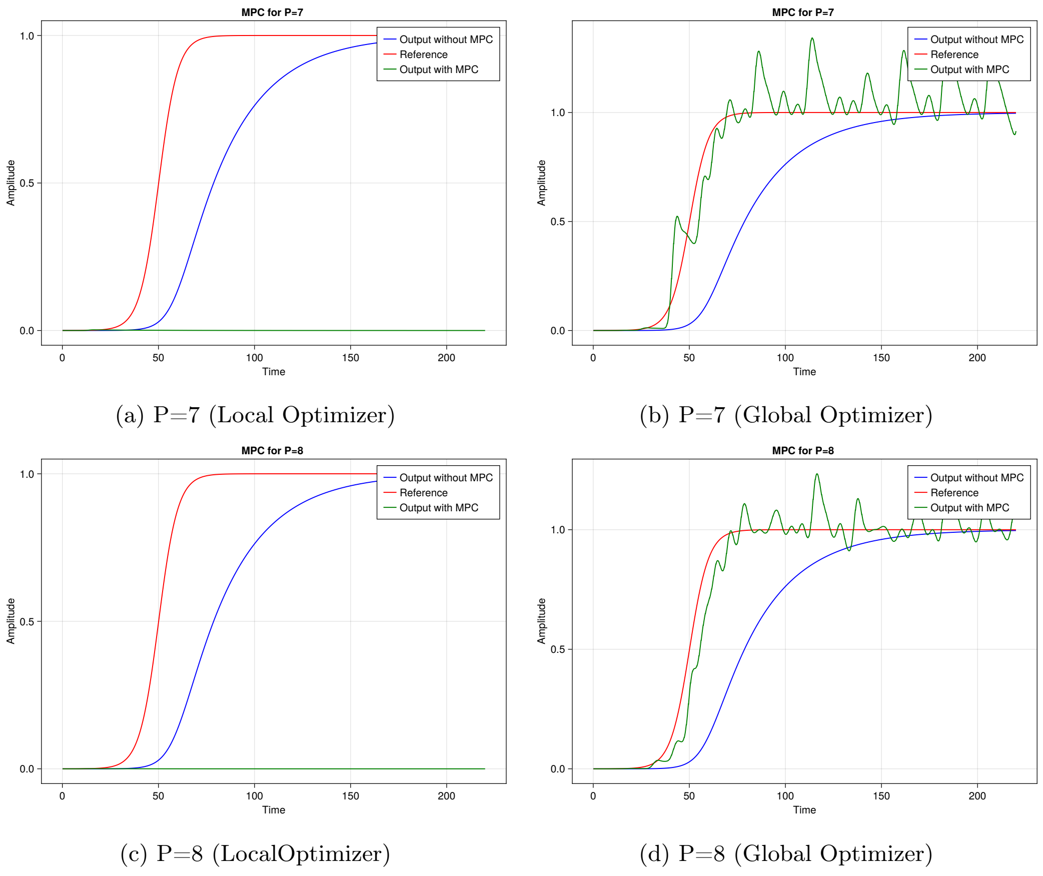
(f) P=20



(g) P=25

Figure 33: MPC With prediction horizon variation.

A noteworthy observation is that as we increase P, the model diligently strives to track the reference with minimal deviation during the initial rise time. However, in the process, it encounters challenges in reducing the steepness of the response after this initial phase.
Now, let's delve into another crucial observation, as depicted in Figure 34. As previously mentioned, the MPC struggled to follow the reference. However, when we transition from a local to a global optimizer, the MPC begins to closely track the reference. Nevertheless, this transition comes at the cost of losing the smoothness of the MPC response, resulting in noticeable jitter. This observation underscores the fact that a local optimizer is not always the best choice.

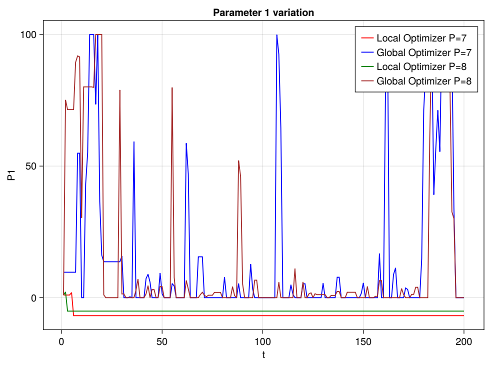


(a) P=7 (Local Optimizer) (b) P=7 (Global Optimizer)

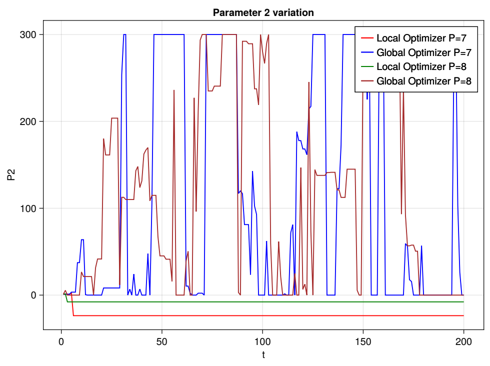
(c) P=8 (LocalOptimizer) (d) P=8 (Global Optimizer)

Figure 34: MPC Local Optimizer vs Global Optimizer.

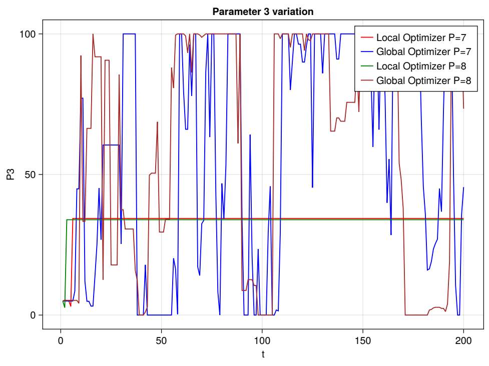
Now, delving deeper into the comparison of how the parameters p1, p2, and p3 vary between global and local optimization, we can examine Figure 35, which illustrates the variations of these parameters for P=7 and 8. It becomes evident that, with local optimization, as these parameters diverge from their original desired values, the system struggles to adapt and ultimately results in failure.



(a) p_1 variation



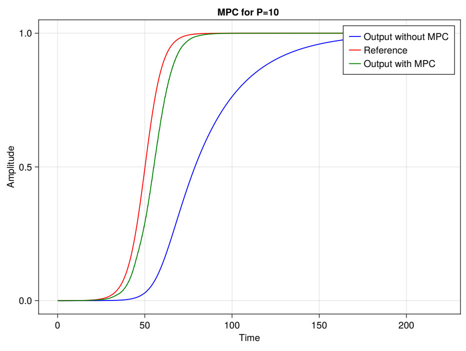
(b) p_2 variation



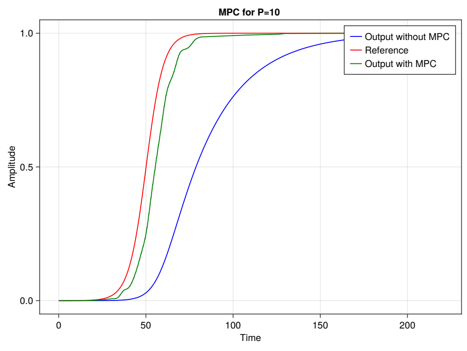
(c) p_3 variation

Figure 35: Parameter variations local vs global optimizer P=7 and P=8.

Now, when examining the scenario with P=10, both the Global and Local optimizers exhibit commendable performance in closely tracking the reference signal. However, it's worth noting that in the case of Global Optimization, there are some fluctuations, particularly during the rise and settling times, as depicted in Figure 36.



(a) Local Optimizer(Polyopt)

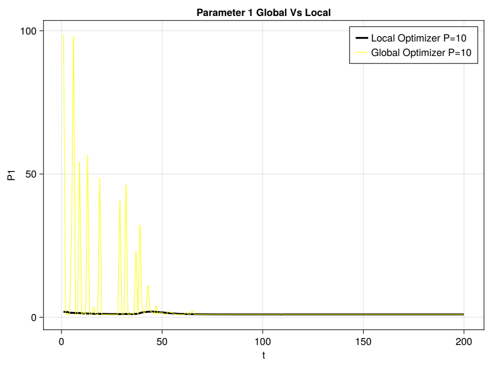


(b) Global Optimizer(ParticleSwamp)

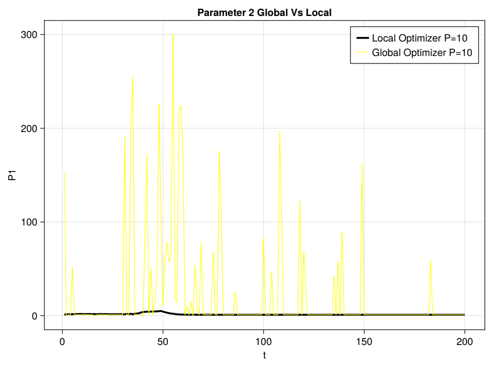
Figure 36: P=10 global vs local optimizer.

Furthermore, let's explore a crucial and advantageous aspect of the local optimizer concerning the behavior of parameters when the prediction horizon is extended, as we've done in this case with a horizon of 10. As evident in Figure 36, both optimization techniques successfully track the reference, but the nuances of how

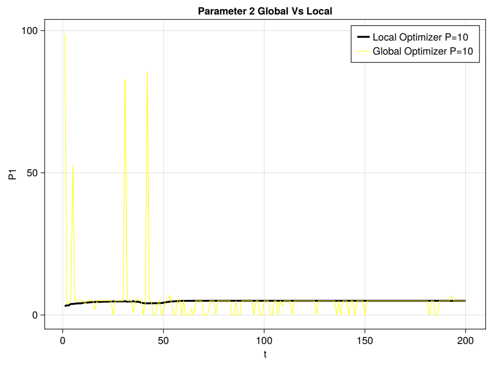
they achieve this warrant closer examination.
For a more detailed insight, refer to Figure 37, where you'll observe that parameter variations are markedly minimal in the case of the local optimizer (depicted in black). In contrast, the global optimizer (depicted in yellow) exhibits substantial variations, leading to a less smooth response. It can be asserted that to achieve a smooth final output, it is essential for parameter variations to exhibit smoothness as well.



(a) p_1



(b) p_2



(c) p_3

Figure 37: Prediction horizon=10 parameters variation local vs global optmizer.

One of the objectives of this thesis is to generate a continuous MPC signal, which necessitates that parameter variations are continuous as well. Achieving perfect continuity is inherently impossible as data is sampled at discrete points. However, by reducing the sampling time and minimizing abrupt parameter changes, we can approach a notion of continuity. Figure 38 illustrates that the parameter variation is notably smoother with only minor deviations, Plus as the system stabilizes the parameter values becomes constant,

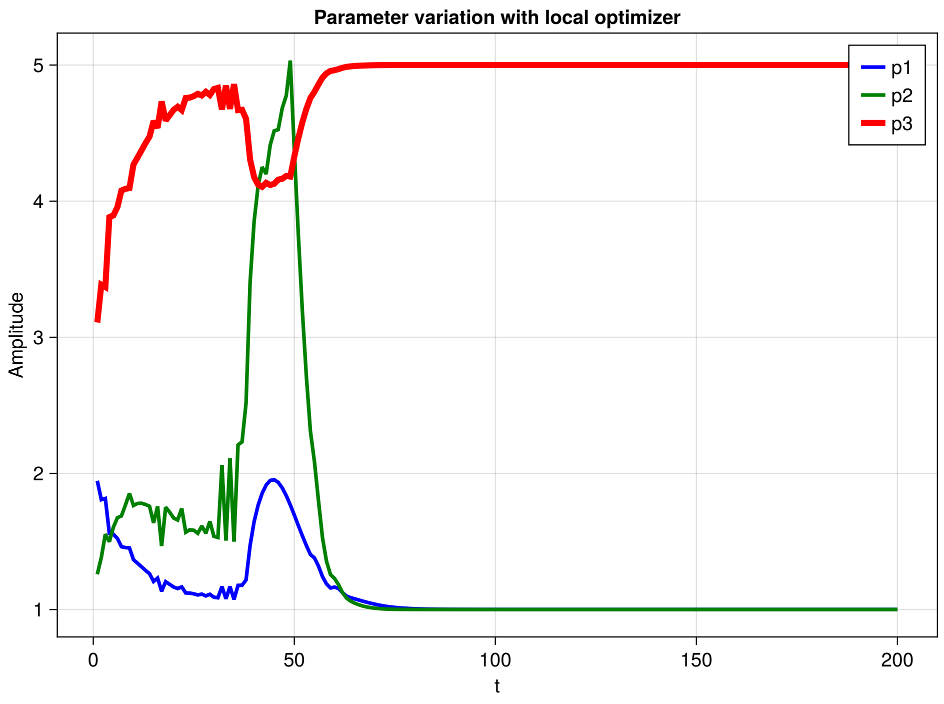
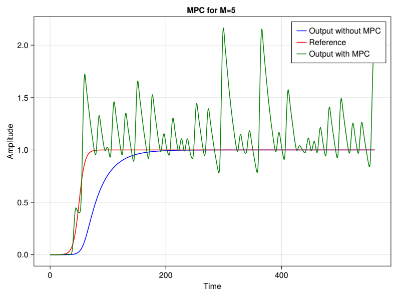


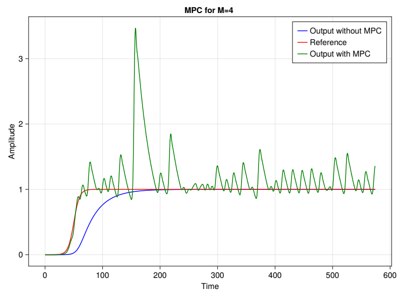
Figure 38: p_1, p_2 and p_3 variation for a polyopt.

5.4 Control Horizon Variation

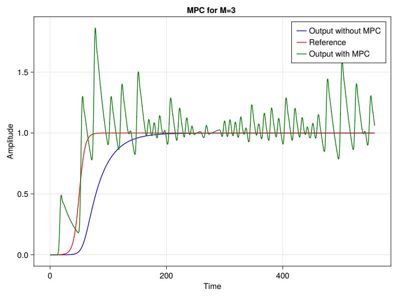
As we observed in the previous section, the prediction horizon exerts a significant influence on the overall MPC implementation. Now, let's shift our focus to another critical factor known as the control horizon (M), which dictates the duration for which control signals are applied to the system following the prediction. Examining Figure 39, a noteworthy trend emerges: reducing the control horizon results in a more precise tracking of the reference signal with fewer perturbations. These results have been obtained using a global optimizer (particle-swarm) with N=8 (system order).



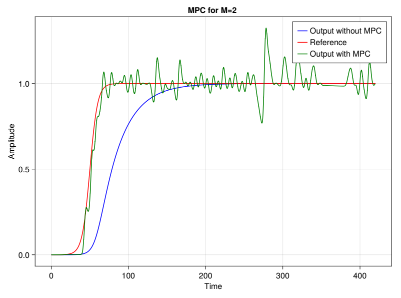
(a) M=5



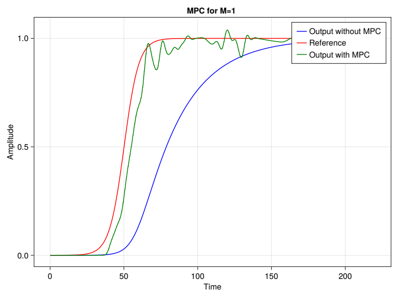
(b) M=4



(c) M=3



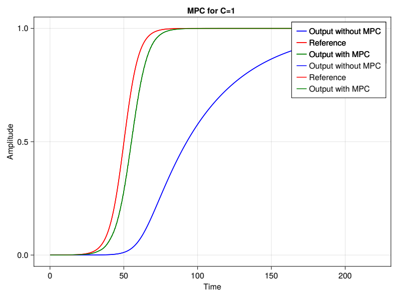
(d) M=2



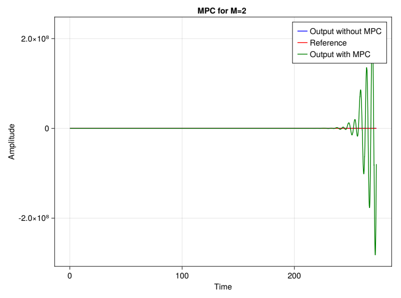
(e) M=1

Figure 39: MPC control horizon variation with global optimizer.

Another noteworthy observation was made when solving MPC with a local optimizer while varying the control horizon from 1 to 2. As depicted in Figure 40, using a control horizon (M) of 2 resulted in highly erratic behavior, with the output going out of control and exhibiting significant fluctuations.



(a) M=1



(b) M=2

Figure 40: MPC control horizon variation with local optimizer(Polyopt).

Conclusion

In conclusion, the research presented in this thesis delves into the challenges of optimizing higher-order circuits and explores various methods to align system performance with desired reference outputs. Several key findings have emerged from this investigation.

First, it was observed that flatness-based control, although effective for certain systems, struggles to optimize higher-order systems, particularly when the reference signal does not exhibit specific steepness characteristics. This limitation raises questions about its suitability for complex systems.

Optimization-based control, on the other hand, demonstrated promising results, with the Polyopt algorithm emerging as the preferred optimizer due to its ability to minimize loss. However, it is important to note that the use of Polyopt is constrained by its inability to incorporate constraints, which may limit its applicability in certain scenarios.

Model predictive control (MPC) highlighted the significance of choosing the right optimizer, whether global or local. Global optimization methods proved capable of following reference signals effectively, even with shorter prediction horizons. In contrast, local optimization techniques required a longer prediction horizon but excelled in managing continuous parameter variations, which in turn produces a continuous MPC input signal. Nevertheless, it should be acknowledged that MPC, especially when utilizing global optimizers, can introduce high levels of output fluctuations or noise that necessitate further investigation.

One overarching takeaway from this research is the emphasis on the steepness of input signals in the optimization process, as this parameter plays a crucial role in system performance. This insight underscores the importance of understanding and controlling input signal characteristics to optimize complex systems effectively.

While this study has made significant strides in addressing the challenges of higher-order circuit optimization, there are areas that warrant further exploration. Notably, the issue of noise introduced by global optimizers in MPC requires additional research to develop strategies for noise reduction.

The findings of this thesis hold promise for practical applications in power electronics and motor control, offering the potential for smoother responses and improved control methods in systems such as power converters, clippers, and clampers. Moreover, the use of tangent hyperbolic functions to smooth system responses could prove invaluable in real-world applications.

In summary, this research contributes valuable insights into the optimization of higher-order circuits, offering practical implications for enhancing the performance of various systems while highlighting areas for future investigation and development.

References

[1] Fabian Pedregosa. Residual polynomials and the chebyshev method. http://fa.bianp.net/blog/2020/polyopt/, 2020.

[2] Heeyoul "Henry Choi, Sookjeong Kim, and Seungjin Choi. Trust-region learning for ica. volume 1, page 46, 08 2004. ISBN 0-7803-8359-1. doi: 10.1109/IJCNN.2004.1379867.

[3] Niklas Donges. Gradient descent in machine learning: A basic introduction. URL https://builtin.com/data-science/gradient-descent.

[4] Matlab. Run field oriented control of pmsm using model predictive control. URL https://de.mathworks.com/help/mcb/gs/run-foc-pmsm-using-model-predictive-control.html.

[5] MATHworks. Mpc block diagram. URL https://de.mathworks.com/help/mpc/gs/what-is-mpc.html.

[6] Dr S Somanath. Chandrayaan 3 india's moon exploration. Prof S Pradeep Memorial Lecture, IISC, Banglore, 2023. URL https://www.youtube.com/watch?v=fZ2sNRP1opY&t=3240s.

[7] Daniel Peters. Evaluation of dynamic mode decomposition for cascaded electrical oscillators. URL https://forschung.rwu.de/sites/forschung/files/2023-03/2023_DMD_Cascaded_RLC_Daniel_Peters.pdf.

[8] Alberto Isidori, CH Moog, and A De Luca. A sufficient condition for full linearization via dynamic state feedback. In *1986 25th IEEE Conference on Decision and Control*, pages 203–208. IEEE, 1986.

[9] Michel Fliess, Jean Lévine, Philippe Martin, and Pierre Rouchon. Flatness and defect of non-linear systems: introductory theory and examples. *International journal of control*, 61(6):1327–1361, 1995.

[10] Richard M Murray et al. Optimization-based control. *California Institute of Technology, CA*, pages 111–128, 2009.

[11] Moritz Diehl and Sébastien Gros. Numerical optimal control. *Optimization in Engineering Center (OPTEC)*, 2011.

[12] Stephan Scholz. N-th derivative of tanh. URL https://en.wikipedia.org/wiki/User_talk:StephanSp3#N-th_derivative_of_tanh.

[13] Grzegorz Rzadkowski. Derivatives and eulerian numbers. *The American Mathematical Monthly*, 115 (5):458–460, 2008.

[14] Vaibhav Dixit. Optimizationpolyalgorithms. https://github.com/SciML/Optimization.jl/tree/master/lib/OptimizationPolyalgorithms, 2023.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Akash Ajagekar. Adam. URL https://optimization.cbe.cornell.edu/index.php?title=Adam.

[17] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2000.

[18] Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[19] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

[20] Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.

[21] Michael JD Powell et al. The bobyqa algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, 26, 2009.

[22] Steven G Johnson and Julien Schueller. Nlopt: Nonlinear optimization library. *Astrophysics Source Code Library*, pages ascl–2111, 2021.

[23] P Mogensen and A Riseth. Optim: A mathematical optimization package for julia. *Journal of Open Source Software*, 3(24), 2018.

[24] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[25] Zhi-Hui Zhan, Jun Zhang, Yun Li, and Henry Shu-Hung Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(6):1362–1381, 2009.

[26] Liuping Wang. *Model predictive control system design and implementation using MATLAB®*. Springer Science & Business Media, 2009.

Source Code

A, B and C matrix calculation

```
N = 6;
Tf = 400;
M = diagm(ones(Int, N)) # matrix with all diagonal element as 1, square matrix with size N by N
D = diagm(ones(Int, N)) # matrix with all diagonal element as 1, square matrix with size N by N
S = diagm(ones(Int, N)) # matrix with all diagonal element as 1, square matrix with size N by N
for i = 1 : N, j = i+1 : N
    M[i,j] = 1  # above the diagonal element value 1
    D[i,j] = 1  # above the diagonal element value 1
end

for i = 1 : N-1
    S[i+1,i] = -1
end

R1 = 1
L1 = 1
C1 = 1

# Now calculating the values of A,B and C, For state space representation
A = zeros(2N,2N) # 2N by 2N matrix
A[1:N, N+1:2N] = diagm(ones(Int,N)) # the top-right side of the matrix is Identity matrix
A[N+1:2N, 1:N] = (-inv(M)*S) / (L1*C1)  # lower left side we have S~ matrix
A[N+1:2N, N+1:2N] = (-inv(M)*D) * (R1/L1) # lower right side we have D~ matrix

B = vcat(zeros(N), 1/(L1*C1), zeros(N-1)) # first N element zero then 1/LC, after that N-1 values are
zero

C = vcat(zeros(N-1),1,zeros(N))'
```

Loss Function

```
function loss(param) # defining the loss function here
    sol = solve(prob, Tsit5(),saveat=0.1, p = param) # what is p here ? p is parameter
    err = sum(abs2,sol[N,2000:1:4001]) -refval # considering reference to be 1
    loss = sum(abs2, err)/length(err)
    return loss #, sol  # return loss value along with solution
end
```

A^N calculation

```
function matrixmul!(mat::Matrix{Float64} , n)
    m = zeros(2N,2N)
    m = copy(mat)
    for i = 2:n
        m[1:N, 1:N] = m[N+1:2N,1:N]                           # A1 update to top left corner
        m[1:N, N+1:2N] = m[N+1:2N, N+1:2N]                    # A2 update to top left corener
        temp1[1:N,1:N]= m[N+1:2N,1:N]                         # A1 old backup before modification
        temp2[1:N,1:N] = m[N+1:2N, N+1:2N]                    # A2 old backup befrore modification
        m[N+1:2N,1:N] = temp2*A[N+1:2N, 1:N]                  # A1(New) = A2(old)*A1
        m[N+1:2N, N+1:2N] = temp1 + temp2*A[N+1:2N, N+1:2N]  # A2(new) = A1(old) + A2(old)*A2
    end
    return m
end
```

ODE Function

```
function my_ode(dx, x, p, t)
    u_in = input_tanh_new(t,T,p) # sin(t)

    dx[1:N] = x[N+1:2N]
    dx[N+1:2N] = (-inv(M)*S) / (L*C) * x[1:N] + (-inv(M)*D) * (R/L) * x[N+1:2N] + G/(L*C) * u_in
    # dx .= A*x + B*u
end
```

Optimization Based Control

```
using Optimization, OptimizationPolyalgorithms, SciMLSensitivity,
Zygote, OptimizationOptimJL,LineSearches, OptimizationNLopt


adtype = Optimization.AutoFiniteDiff()
# adtype allows us to choose the type of Automatic differentiation we use.

optf = Optimization.OptimizationFunction((x, param) -> loss(x),adtype)
# x is variable , param is initial value of the parameters

optprob = Optimization.OptimizationProblem(optf, param,lb=[0.1,0.0] , ub=[100.0,50.0] )
#Final problem defination

opt_sol_Para_temp= Optimization.solve(optprob, Optim.ParticleSwarm(lower = optprob.
lb, upper = optprob.ub,
n_particles = 200), callback = cb_new, maxiters =4 )
# opt_sol_Para_temp is initial optimized parameters obtained,
they would be fed to local optimization
technique in order to reach optimized solution
param_new=[ 1.0188661066056204,  61.593282512921746 ]

new_adtype = Optimization.AutoFiniteDiff()

optf_new = Optimization.OptimizationFunction((x, param_new ) ->
loss(x),new_adtype) #now using obtained parameters as input to the system

optprob_new = Optimization.OptimizationProblem(optf_new, param_new,lb=
[0.1,0.0] , ub=[100.0,10.0] )

new_optparameters= Optimization.solve(optprob_new,
Optim.ParticleSwarm(lower = optprob_new.lb, upper = optprob_new.ub,
n_particles = 200), callback = cb, maxiters = 50)

par_2= [new_optparameters[1], new_optparameters[2]]

optf_new_2= Optimization.OptimizationFunction((x, par_2 ) -> loss(x),new_adtype)

optprob_new_2 = Optimization.OptimizationProblem(optf_new_2, par_2)
```

```
new_optparameters_2= Optimization.solve(optprob_new_2,
Optim.GradientDescent(), callback=cb_drop95percent ,maxiters = 4);

par_3= [new_optparameters_2[1], new_optparameters_2[2]]
optf_new_3= Optimization.OptimizationFunction((x, par_3 ) ->
loss(x),new_adtype)

optprob_new_3 = Optimization.OptimizationProblem(optf_new_3, par_3,lb=
[0.1,0.0] , ub=[100.0,1300.0])

new_optparameters_3= Optimization.solve(optprob_new_3,NLopt.LN_BOBYQA(),
callback=cb_drop99percent ,maxiters = 50);
```

Model Predicitve Control

```
prob_mpc = ODEProblem(my_ode_new, x0, (0,t_ph),param)
for i =1:Nsteps
    # define the problem
    optprob = Optimization.OptimizationProblem(optf, next_param);

    new_optparameter = Optimization.solve(optprob,PolyOpt(),callback=cb_new, maxiters =
    30); # Now optimize the problem with the param
    println(new_optparameter)
    paramArray[:,i+1] = new_optparameter.u
    next_param= new_optparameter.u

    prob1 = remake(prob_mpc; tspan=(0+t_ch*(i-1), t_ch*i))

    solution = solve(prob1,alg,
    p=[new_optparameter.u[1],new_optparameter.u[2],new_optparameter.u[
    3]],alg_hints=[:stiff], saveat=0.1)
    sol_ar[i,:]=solution[N,:]
    println(sol_ar[i,:])

    prob_mpc = remake(prob_mpc; u0=solution[:,end] ,tspan =(0+i*t_ch,t_ph+(i*t_ch)))
end
```