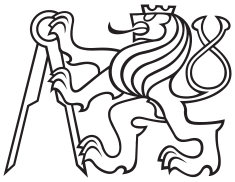
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering

Energy-efficient train control

Bc. Vít Fanta

Supervisor: doc. Ing. Zdeněk Hurák, Ph.D.
Study programme: Cybernetics and Robotics
May 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Fanta Vít** Personal ID number: **474463**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Energy-efficient train control

Master’s thesis title in Czech:

Energeticky efektivní řízení kolejových vozidel

Guidelines:

1. Document the state of the art in solving the problem of energy-efficient train control. The problem generally consists of designing and optimal speed profile for a given vehicle, track, and schedule, and designing a feedback controller that follows the computed speed profile.
2. If useful, consider special challenges in commuter trains.
3. Extend the existing solutions with the possibility to re-compute in real time the velocity profile upon reception of a message about the unexpected situation on some remote segments of the track.
4. Implement your algorithm(s) in some chosen programming language (with some preference for Julia, but Python and Matlab can be used too).
5. Format your resulting code in the form of a well-documented open-source package/library/toolbox shared online with an international community.

Bibliography / sources:

[1] A. Albrecht, P. Howlett, P. Pudney, X. Vu, and P. Zhou, “The key principles of optimal train control—Part 1: Formulation of the model, strategies of optimal type, evolutionary lines, location of optimal switching points,” Transportation Research Part B: Methodological, vol. 94, pp. 482–508, Dec. 2016, doi: 10.1016/j.trb.2015.07.023.
[2] A. Albrecht, P. Howlett, P. Pudney, X. Vu, and P. Zhou, “The key principles of optimal train control—Part 2: Existence of an optimal strategy, the local energy minimization principle, uniqueness, computational techniques,” Transportation Research Part B: Methodological, vol. 94, pp. 509–538, Dec. 2016, doi: 10.1016/j.trb.2015.07.024.
[3] G. M. Scheepmaker, R. M. P. Goverde, and L. G. Kroon, “Review of energy-efficient train control and timetabling,” European Journal of Operational Research, vol. 257, no. 2, pp. 355–376, Mar. 2017, doi: 10.1016/j.ejor.2016.09.044.
[4] P. G. Howlett, I. P. Milroy, and P. J. Pudney, “Energy-efficient train control,” Control Engineering Practice, vol. 2, no. 2, pp. 193–200, Apr. 1994, doi: 10.1016/0967-0661(94)90198-8.
[5] B. Jaekel and T. Albrecht, “Comparative analysis of algorithms and models for train running simulation,” Journal of Rail Transport Planning & Management, vol. 4, no. 1, pp. 14–27, Aug. 2014, doi: 10.1016/j.jrtpm.2014.06.002.

Name and workplace of master's thesis supervisor:

doc. Ing. Zdeněk Hurák, Ph.D. Department of Control Engineering FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **17.02.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

doc. Ing. Zdeněk Hurák, Ph.D. Supervisor's signature	prof. Ing. Michael Šebek, DrSc. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

First of all, I would like to express gratitude to my family for their great support during my studies. Special thanks goes to my mum, an English teacher, for helping with the text of this thesis.

I also want to thank my supervisor, doc. Ing. Zdeněk Hurák, Ph.D., for his mentoring and providing constant motivation.

I would like to express gratitude to all the people I have got to know because of my studies at FEE CTU, especially one group of FELlows and their support on tough Mondays and even tougher Tuesdays...

Declaration

I hereby declare that I have written the presented thesis on my own and that I have cited all of the used information sources in accordance with the Methodical instructions about ethical principles for writing an academic thesis.

In Prague, 26th May, 2023

Abstract

The goal of this thesis is to present the problem of optimal train control and implement an open-source software package able to solve it. The solution is a speed profile which takes train's traction capabilities, total journey time and the track gradient into account while minimising the total required energy.

The optimal control strategy is a switching strategy between a small number of modes: Maximum Power, Cruising, Optimal Braking, Coasting and Maximum Braking. On a flat track, the optimal strategy has the form Maximum Power – (Cruising) – Coasting – Maximum Braking. On a general track, more switching can occur due to the steep uphill and steep downhill sections of the track.

A solution has been implemented in the Julia programming language and is available to the public in a form of a package accessible on GitHub. The solution is functional and examples of speed profile calculation are shown. Some features are still missing, most notably inclusion of speed limits. Functionality of the implemented solution can be further developed thanks to its open-source nature.

Keywords: Optimal train control, maximum principle, Julia programming language, algorithm design

Abstrakt

Cílem této práce je představit problém optimálního řízení vlaku a implementovat open-source softwarový balíček, pomocí něhož je jej možno řešit. Řešením je rychlostní profil, který bere v úvahu trakční limity vlaku, celkový čas cesty a výškový profil tratě a zároveň minimalizuje celkovou vynaloženou energii.

Optimální strategie spočívá v přepínání několika jízdních režimů: maximální trakce, držení rychlosti, optimální brzdění, výběh a maximální brzdění. Na ploché trati má optimální strategie podobu maximální trakce – (držení rychlosti) – výběh – maximální brzdění. Na obecné trati může docházet k většímu množství přepnutí kvůli příliš strmým úsekům.

Řešení bylo implementováno v jazyce Julia a je dostupné veřejnosti v podobě balíčku na GitHub. I přes to, že je program obecně funkční a příklady výpočtu jsou uvedeny. Některé funkcionality stále chybí (především možnost zahrnout rychlostní omezení). Současné řešení je však možno dále vyvíjet díky jeho open-source povaze.

Klíčová slova: Optimální řízení vlaku, princip maxima, programovací jazyk Julia, návrh algoritmu

Překlad názvu: Energeticky efektivní řízení kolejových vozidel

Contents

Introduction

Goals

In accordance with the thesis assignment, the goal of this thesis is

1. to document the state of the art in solving the problem of energy-efficient train control,
2. to implement a numerical method for computing optimal speed profiles in a programming language whilst allowing the ability to recalculate the solution online and considering special challenges in commuter train traffic,
3. and to format the resulting code in the form of a documented open-source package available to the public.

Motivation

Energy-efficient solutions are sought in each industry. Train transport is not an exception to this trend. Moreover, the demand for such economical systems grows higher in recent years, also due to higher fluctuations of the cost of electric energy and fuel.

Implementation of energy-optimal train control systems has also yielded unexpected benefits. In several instances, adhering to the calculated driving strategy resulted in better following of the train schedule (G. M. Scheepmaker et al., 2017; Albrecht et al.,

2016b). The reason for such improvements is that the train operator can simply follow the precomputed speed profile if the planning system is in place, but has to rely on their experience (or lack thereof) in its absence.

Thesis Structure

The thesis describes implementation of an algorithm to solve the optimal train control problem in the Julia programming language.

The summary of the relevant references and an introduction to the topic of optimal train control is presented in chapter 1.

Chapter 2 is dedicated to a brief introduction to the Julia programming language. The reason for that is to make sure the reader understands Julia code which is shown in the text further. At first, the reasons why Julia has been selected as the implementation language along with the basics of the syntax are presented. Several used Julia packages are described with examples of their use. Subsequent chapters also contain Julia code snippets which show how some of the ideas are implemented in the presented package. The snippets are shown directly in the text of the thesis as opposed to the option of referencing the complete code in the appendix. This has been done to improve text comprehension, and besides, the complete code is present on a GitHub page.

Chapter 3 states the complete problem of optimal train control and contains theoretical analysis of its solution. The model of longitudinal train dynamics is presented in its full detail along with all of the problem's constraints.

The Optimal control problem for the simplified case of a flat track is discussed in chapter 4. The optimal sequence of control modes is presented in the form of state diagrams and the influences of the total available journey time and the regeneration coefficient are discussed with examples.

The optimal solution to the general problem of optimal control with a flat track is presented in chapter 5. The description of steep sections of a track is then followed by introducing the linking procedure (finding individual parts of the optimal speed profile and then combining them to form the complete solution).

Finally, chapter 6 shows outputs of the implemented algorithm on a selection of example problems. Comparison with another solution strategy is presented and the important discussion of possible improvements concludes the thesis.

Chapter 1

State of the Art

The problem of optimal train control was studied extensively during the second half of the twentieth century. The historical developments in this field are well summarised in Albrecht et al. (2016a) and in G. M. Scheepmaker et al. (2017), and short description of the relevant findings is presented here.

The task is, stated in general terms, minimisation of a criterion via finding a continuous control subject to a continuous dynamics model, boundary conditions and (possibly) state path constraints. Such problems are the topic of the optimal control theory (Athans et al., 2007; Liberzon, 2012). Generally, constrained optimal control problems are difficult to solve directly as they require very accurate initial guesses of solutions. To avoid this additional domain-specific insight is often used to solve the problem in efficient manner. The classical form of the optimal train control problem is to minimise the energy while driving the train from one station to another within a set time. Moreover, the traction capabilities of the train are limited and the terrain can make it difficult for the train to traverse steep sections.

As with other optimal control problems, the optimal train control problem can be addressed in two ways:

- indirect methods derive conditions of optimality and then attempt to solve them,
- direct methods discretise the problem into a general non-linear programming optimisation problem.

1.1 Indirect Methods

The indirect methods often rely on the use of Pontryagin's maximum principle (PMP) to analyse the structure of the solution, which is in the form of a sequence of switching control phases. This indirect approach has been dominant since the publication of the earliest articles Ichikawa (1968) and Asnis et al. (1985). Although the presented results only considered the case of a flat track, they showed that the control mode sequence typically had the pattern of Maximum Power – Cruising – Coasting – Maximum Braking* as it is discussed in further chapters. It was shown later that under some restrictions on the form of the motion resistance function and the tractive limits of the train, closed-form formulas for the optimal speed profile could be obtained (G. M. Scheepmaker et al., 2017; Albrecht et al., 2016a). The problem involving a general track with variable gradient is much more difficult to solve and, so far, only numerical algorithms have been proposed as a solution strategy.

The theory of optimal train control has been extensively developed by the Scheduling and Control Group at the Centre (SCG) for Industrial and Applied Mathematics of the University of South Australia. Their findings have been summarised in a two-part article Albrecht et al. (2016a) and Albrecht et al. (2016b), which is regarded as the current theoretical state of the art in the field of optimal train control. The key discovery lies in the fact that it is possible to express the costate determining the current control mode algebraically. A similar result was presented in Liu et al. (2003) for a less general form of the problem, in which no braking energy regeneration is considered. Apart from theoretical results, SCG has developed several driver advisory systems (DAS) which implement numerical algorithms to solve optimal train control problems on-board to provide advice to a train operator. These systems are (in the chronological order of development): Metromiser (Baier et al., 2000), Freightmiser (Coleman et al., 2010) and Energymiser (MyDAS: Driving Advice Systems and Statistics for Train Drivers | Trapeze Group, [n.d.]). Figure 1.1 shows the Energymiser DAS on-board of a train. The driver is notified of the current driving strategy and can inspect the predicted speed profile (G. M. Scheepmaker et al., 2017; Albrecht et al., 2016a).

Although a train generally consists of multiple carriages and locomotives, it has been shown (jaekelComparativeAnalysisAlgorithms2014; Albrecht et al., 2016a) that approximating the whole train with a single point mass yields very similar results, which are much simpler to obtain than if the train were modelled in a distributed-mass manner. For this reason, the equations of train motion are usually in the form

$$\dot{x}(t) = v, \tag{1.1a}$$

$$\dot{v}(t) = u - R(v, x), \tag{1.1b}$$

where x is the position along the track, v is the speed of the train, u is the control signal and $R(v, x)$ is the train motion resistance consisting of mechanic and aerodynamic

*Throughout the text of the thesis, the control modes are typeset with their specific colours. The corresponding colours are also used in plots and diagrams.

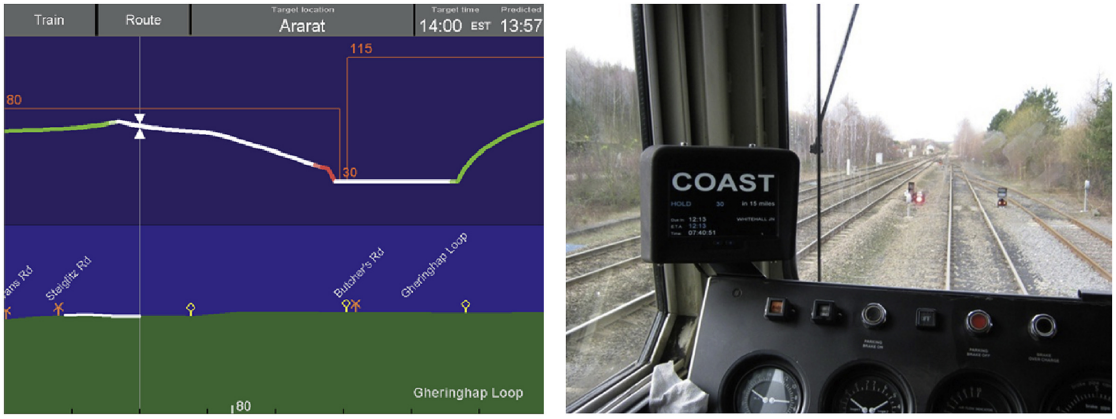


Figure 1.1: A photography of the active Energymiser DAS. The system notifies the train operator when the control mode should be changed and also shows how long the current mode should remain active. The image is from Albrecht et al. (2015).

components as well as the influence of the track gradient. The mass of the train does not appear explicitly because the models are usually shown in mass-specific form (all of the terms are divided by mass). Despite the common practice of expressing dynamical models as a set of differential equations where time is the independent variable, in the case of optimal train control problems, it proved to be more useful to write the system with the distance x as the independent variable (Albrecht et al., 2016a; G. M. Scheepmaker et al., 2017). The derivation is straightforward if the informal manipulation with differentials is allowed[†]:

$$v = \frac{\mathrm{d}x}{\mathrm{d}t} = \left(\frac{\mathrm{d}t}{\mathrm{d}x}\right)^{-1} = \frac{1}{t'}, \tag{1.2}$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{\mathrm{d}v}{\mathrm{d}t}\frac{\mathrm{d}x}{\mathrm{d}x} = \frac{\mathrm{d}v}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}t} = v'v = u - R(v,x). \tag{1.3}$$

Consequently,

$$t' = \frac{1}{v}, \tag{1.4a}$$

$$v' = \frac{u - R(v,x)}{v}. \tag{1.4b}$$

The system (1.4) is not defined at points where $v = 0$. To partially circumvent this issue, different models (although equivalent) have been also developed. Khmelnitsky (2000) presents a model with total specific energy $E = P + K$ and time t as state variables:

$$E' = u - R(v,x), \tag{1.5a}$$

$$t' = \frac{1}{\sqrt{2K}}. \tag{1.5b}$$

Khmelnitsky (2000) also developed an elegant algorithm based on indirect methods to find the optimal speed profile for the general optimal train control problem with

[†]In this thesis, the derivative of f with respect to the position x will be denoted as $f' = \frac{\mathrm{d}f}{\mathrm{d}x}$.

possible energy regeneration during braking and general speed limit. This algorithm serves as a basis for implementation of the algorithm presented in this thesis because of the more unified approach of Khmelnitsky (2000) compared to the exhaustive singular case handling presented by the SCG (Albrecht et al., 2016a; Albrecht et al., 2016b).

1.2 Direct Methods

The direct methods provide a completely different approach of solving optimal train control problems. The key idea lies in discretisation of the problem (e.g. the distance along the track, control signal values, values of the states) and solving it as a general non-linear program. Such methods have only become more popular during the last two decades, thanks to rising performance of computer machines.

In Wang; De Schutter; Ning, et al. (2011), the optimal train control problem is solved as a mixed-integer linear program (MILP). The nonlinear dynamics are approximated with piece-wise affine (PWA) functions to make the MILP formulation of the problem possible. The considered state equations have the continuous form (1.5); the discretised system is obtained by applying the trapezoidal integration rule and by approximating the right-hand side of (1.5b) with a 3-part PWA function (shown in figure 1.2). The algorithm implemented in this thesis is compared with the mixed-integer linear program solution in the final chapter.

Other often-employed direct method of solution of optimal control problems is direct collocation. This approach also counts with discretisation of the x-axis, but approximates the state and control trajectories on each subinterval with a function of a fixed form (most commonly low degree polynomials, Chebyshev polynomials, Legendre polynomials). In Wang; De Schutter; van den Boom, et al. (2013) the collocation method is compared with the MILP solution of Wang; De Schutter; Ning, et al. (2011). The collocation method yielded better results. Both the MILP approach and collocation can suffer from chattering behaviour, as it is seen in Goverde et al. (2021). This is a general downside of direct methods since the solution provides no insight into the structure of the solution (outputs are generally vectors of numbers). This fact is a fundamental difference comparison with the indirect approach where the structure of the optimal solution is revealed immediately. On the other hand, implementation of the direct approach (problem transcription and calling the appropriate solver) is simpler compared to the indirect approach (solving algebraic and differential equations).

The above-referenced articles are part of train-control-related research in the Netherlands and Delft University of Technology, in particular. Recently, their research has been more oriented towards the problem of optimal train timetabling, which is treated in detail in G. Scheepmaker (2022). The global research trend is similar: the optimal control problem of a single train is considered to be a closed topic and now new optimisation problems, such as timetabling or time-window optimisation, have arisen.

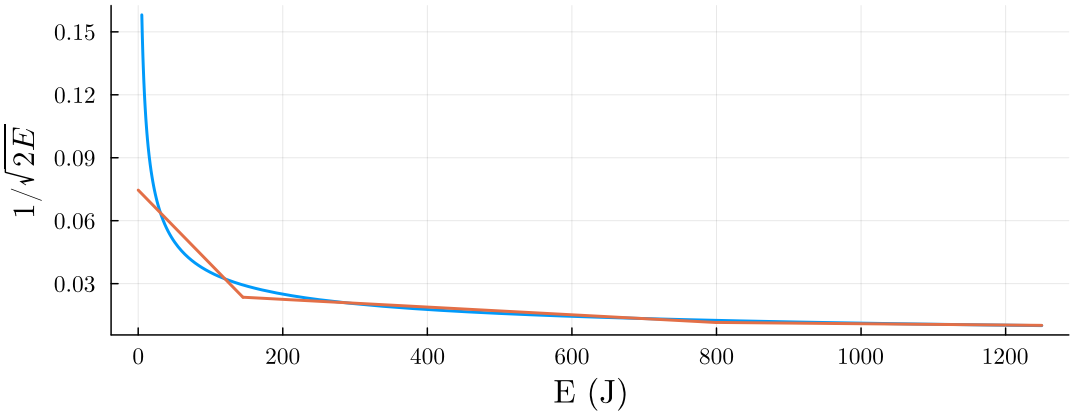


Figure 1.2: Piece-wise affine approximation of $1/\sqrt{2E}$ from Wang; De Schutter; Ning, et al. (2011).

Chapter 2

Introduction to Julia Programming Language

Julia is an award-winning* programming language created with the focus on scientific programming, numerical computation and reproducible and open software (The Julia Programming Language, [n.d.]). For these reasons, it is becoming increasingly popular among scientific community, including the Department of Control Engineering at CTU FEE.

The main marketed features of Julia include dynamic typing and multiple dispatch. The former is present in a number of other programming languages, such as Python, JavaScript or Ruby, while the latter feature of multiple dispatch is central to Julia's paradigm. Also, REPL (read – evaluate – print – loop) prompt is natively provided as in MATLAB or Python. Other highlights of the language are:

1. Performance: Julia is designed to be fast, with performance comparable to that of C and Fortran. This makes it ideal for solving complex differential equation problems, where performance is critical.
2. Flexibility: Julia provides a flexible syntax and dynamic type system that makes it easy to express mathematical equations and algorithms. This allows users to create customised solutions which meet their specific needs.
3. Interoperability: Julia has a built-in interface to C and Fortran libraries, which allows users to leverage existing code and libraries. This makes it easy to integrate Julia into existing workflows and projects.
4. Numerical accuracy: Julia provides advanced numerical tools which ensure high numerical accuracy in solving differential equations. This is essential for scientific computing applications where accuracy is critical.

*"The 2019 James Wilkinson Prize is awarded to Jeff Bezanson, Stefan Karpinski, and Viral B. Shah for the creation of Julia, an innovative environment for the creation of high-performance tools that enable the analysis and solution of computational science problems."(James H. Wilkinson Prize for Numerical Software | SIAM, [n.d.])

5. Ease of use: Julia is designed to be easy to use, with a simple and intuitive syntax which makes solving differential equation problems convenient even for inexperienced programmers[†].

Julia has been chosen as the implementation language for this thesis mainly for its extensive support of solving differential equations with the use of DifferentialEquations.jl package (Rackauckas; Nie, 2017). It provides a large variety of solvers for initial value problems, boundary value problems, stochastic differential equations and many more. Automatic solver selection is also included (Rackauckas; Nie, 2019). Furthermore, the package allows extensive support for event handling (changing structure or state of the ODE problem during simulation) and can achieve C-like computation speed (after compilation) (Rackauckas, [n.d.]; Julia Micro-Benchmarks, [n.d.]). Besides quantifiable qualities, another advantage of Julia is its vibrant and active community focused on numerical and optimisation algorithms.

Apart from its perks, Julia has also some downsides. Creator's of Julia attempted to solve the infamous two-language problem (algorithm is first coded in one language as a proof of concept and then rewritten to another language for production) by making Julia a compiled language. Their goal is achieved only partially. Julia is just-in-time (JIT) compiled language, so the compilation is done at run time of a program. The downside is that the compiler has to infer the types of the present variables to compile type-specific version of functions at the time of compilation. If these types change during run time, another function has to be compiled. For this reason, it is difficult for Julia to produce binary executables since they have to contain a large number of type-specific variants of functions.

2.1 Syntax and Code Snippets

The code snippets in this thesis will look generally like this:

```
1 # This function defines sumation of two MyType variables
2 function mySum(a::MyType, b::MyType)
3     a.value + b.value
4 end
```

In the above example a new function called `mySum` is defined. Its definition is encompassed with the `function` block which is terminated with the `end` keyword. Such structure is similar for other usual control flow constructions (`for` loops, `while` loops, `struct` definitions, `if` statements). The `type` of the function arguments can be specified using

[†]"Introduce the Julia programming language and its perks with the emphasis on solving differential equation problems." prompt. ChatGPT, 23 March version, OpenAI, 21 April 2023, link. Edited.

the `::` operator. This can be seen on line 2 of the snippet where the arguments are specified to have `type` of `MyType`. Although Julia has the `return` keyword, its use is often unnecessary since the last evaluated expression of the `function` body is automatically returned.

Although one can be satisfied with the previous code, the more "Julia way" would be to overload the `+` symbol to work even with the `MyType` variables. This is implemented in the following snippet:

```
1     Base.:+(a::MyType, b::MyType) = a.value + b.value
```

The `+` function was called from the `Base` module, which contains the most basic features of the language (including summation of numbers). Note that the new behaviour of the `+` sign was implemented as a so-called assignment form function definition (omitting the `function` block completely and defining the behaviour only on a single line). This is similar to the standard blackboard notation, e.g. $f(x) = x^2$ would be transcribed directly to `f(x) = x^2`.

The code sections in this thesis are typeset with the use of LaTeX(as is the rest of the text), the **`listings`** package and the **`julia-mono-listings`** style, which utilises the JuliaMono font (Moss, 2023; The JuliaMono Typeface, [n.d.]).

2.2 Used Packages

In this section, the most notable packages which have been used during implementation of the package are briefly introduced. Packages can be installed by typing

```
1 using Pkg; Pkg.add(PackageName)
```

into the REPL. Then, functions from PackageName.jl can be imported by writing a `using` statement to the header of a script:

```
1 using PackageName # imports functions
2 functionfromPackageName() # calling imported function
```

2.2.1 DataFrames.jl

The DataFrames.jl package (DataFrames.Jl, 2023) provides a convenient way to store and manipulate data in a tabular form. Its use is very straightforward as the usage of the

core `DataFrame` structure is similar to the one of arrays and dictionaries (maps) from the Julia's `Base` module.

The easiest way (and the only way used in this thesis) to create a `DataFrame` is to explicitly assign rows of values to the individual column names:

```
using DataFrames
df = DataFrame(
    "first name" => ["Norbert", "Rudolf"],
    "last name" => ["Wiener", "Kalman"])
df[1, :] # "Norbert Wiener"
```

2.2.2 Plots

Although Julia ecosystem contains multiple packages for data visualisation (Makie, Luxor, Cairo), the Plots.jl package (Plots, 2023) is the most popular. Its main feature is that it provides interface to several of the most used visualisation backends, such as GR, PythonPlot or PGFPlotsX. However, this functionality is not used in the scope of this thesis and only the default graphics backend, GR, is used for visualisation purposes.

The syntax of the core `plot` function is almost identical to the one featured in MATLAB:

```
using Plots
x = 0:0.1:2π
y = sin.(x) + cos.(x)
plot(x, y)
```

The code above also shows an example of another feature of Julia, broadcasting. Any `function` can be called with the `.` character after its name to broadcast its functionality to every element of the argument. This also works for functions with multiple arguments. The result of the above code can be seen in figure 2.1.

2.2.3 Roots.jl

The Roots.jl package (Root Finding Functions for Julia, 2023) contains functions which search for roots of real-valued functions of a single real variable. A wide variety of different bracketing, derivative-free and Newton-like methods are implemented and can be used to solve root-finding problems. The only method used in this thesis is the basic bisection method, which is used in a similar fashion as in the example below. The problem is to find a positive root of the function $f(x) = \cos x - x$:

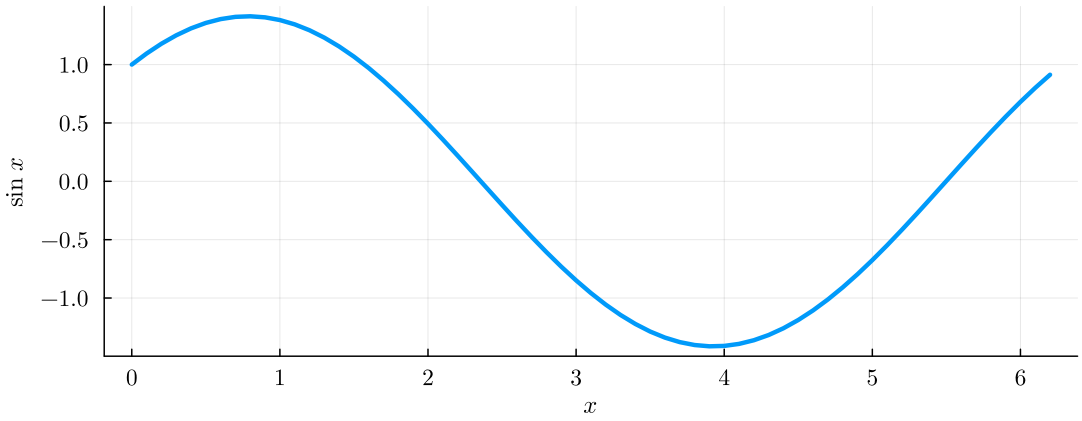


Figure 2.1: Output of the example code showcasing the `plot` function from the `Plots` package.

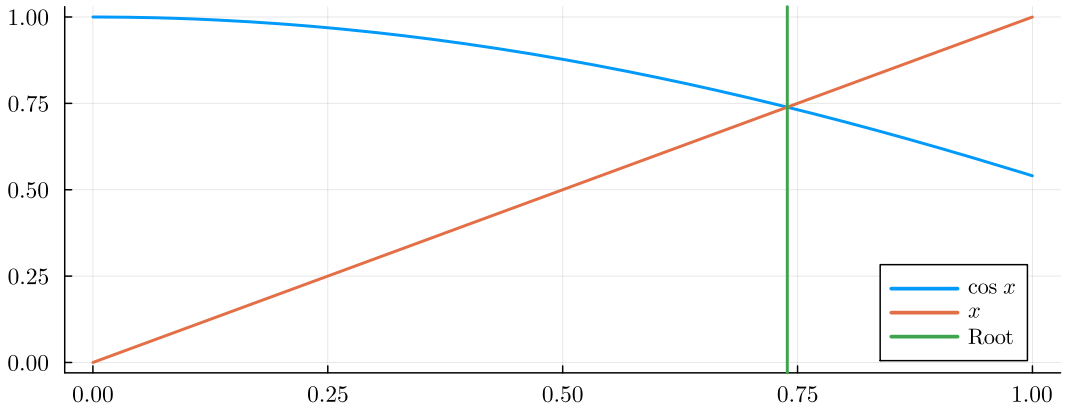


Figure 2.2: Graphic visualisation of finding root with the `Roots` package.

```
using Roots
root = find_zero(x -> cos(x) - x, (0, 1)) # ≈ 0.739085
plot(x -> cos(x), 0, 1)
plot!(x -> x, 0, 1)
vline!([root])
```

The `find_zero` function takes two arguments: the function to be minimised and a pair of values which define the initial bracket. The function must have different signs when evaluated at the boundaries of the initial bracket. Above, the syntax form of the function in the first argument is called anonymous function. In this form, the argument `x` is placed before the `->` symbol and is followed by the functional expression. Figure 2.2 is created by the above code.

2.2.4 DifferentialEquations.jl

One of the areas in which Julia excels is solving differential equation problems. Julia provides a wide range of libraries and tools for working with differential equations, including the above mentioned DifferentialEquations.jl package, which is widely used. DifferentialEquations.jl (Rackauckas; Nie, 2017) is a Julia suite of numerical solvers of ordinary differential equations and other related problems. Its interface is very simple and allows to modify the original problem (changing the ODE function or even state dimension) during the solving time. This feature of so-called event handling was used extensively during implementation of the package.

To provide an example of the solving procedure of a simple ODE initial value problem, the simple damped planar pendulum has been chosen with

$$\frac{\mathrm{d}^2\varphi}{\mathrm{d}t^2} + b\frac{\mathrm{d}\varphi}{\mathrm{d}t} + \frac{g}{l}\sin\varphi = 0 \tag{2.1}$$

or

$$\frac{\mathrm{d}\varphi}{\mathrm{d}t} = \omega, \tag{2.2a}$$

$$\frac{\mathrm{d}\omega}{\mathrm{d}t} = -\frac{g}{l}\sin\varphi - b\omega, \tag{2.2b}$$

where l is the length of the massless pendulum rod and g is the magnitude of gravitational acceleration. The state equations (2.2) are rewritten in the functional form to evaluate the right-hand side:

```
function pend!(dx, x, p, t)
    l, g, b = p
    φ, ω = x
    dx[1] = ω
    dx[2] = -g/l * sin(φ) - b * ω
end
```

The `!` symbol at the end of the above function's name indicates that the function mutates at least one of its arguments, in this case the derivative `dx`. Although the code would compile even without this emphasis, it is a recommended naming convention. A careful reader might have noticed that the above code contains Greek letters. Not only that, Julia supports the most frequently used Unicode characters. This is particularly useful when transcribing equations (as the case above) into code without naming variables as the corresponding Greek letters (`phi`, `omega`).

The problem statement is completed with specification of parameter values and initial state values:

$$l = 1, \quad g = 9.81, \quad b = 0.5, \tag{2.3a}$$

$$\varphi_0 = 0, \quad \omega_0 = 0.5, \tag{2.3b}$$

which is can be directly transcribed into code to form an `ODEProblem` structure:

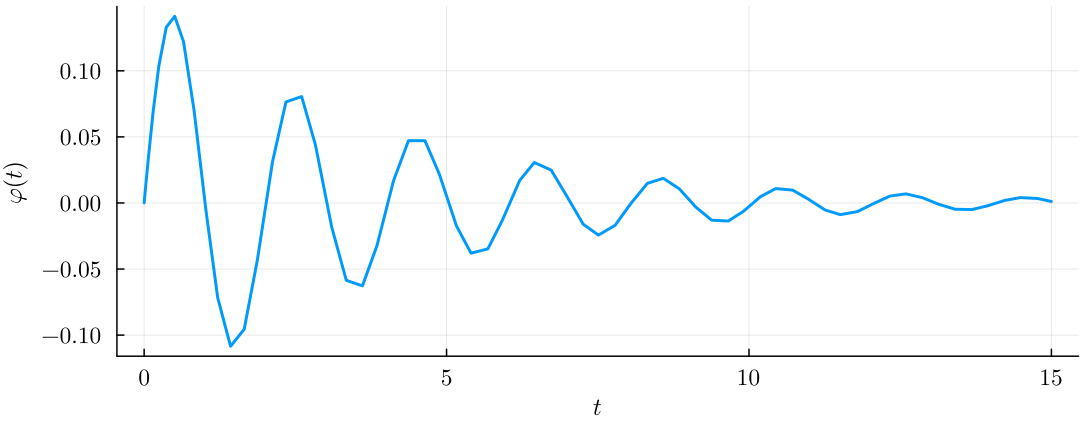


Figure 2.3: Numerical solution of $\varphi(t)$ to the problem given by (2.2) and (2.3).

```
p = [l, g, b]
x0 = [0, 1/2]
tspan = (0,15)
prob = ODEProblem(pend!, x0, tspan, p)
```

The solution is then just a matter of

```
sol = solve(prob)
plot(sol.t, sol[1,:])
```

The solution contains the time axis and also the state values at the corresponding times. The result is shown in figure 2.3.

2.2.5 BasicInterpolators.jl

The BasicInterpolators.jl (Baum, 2023) helps with providing functions interpolating a set of discrete points. Although several different interpolation methods are available (for instance cubic polynomials, splines or Chebyshev polynomials), only the first-order, linear interpolation, is used in this thesis. The package contains `struct` definitions for each of these interpolation methods; the constructors take the vectors of x-values and y-values to be interpolated as it is shown in the following code.

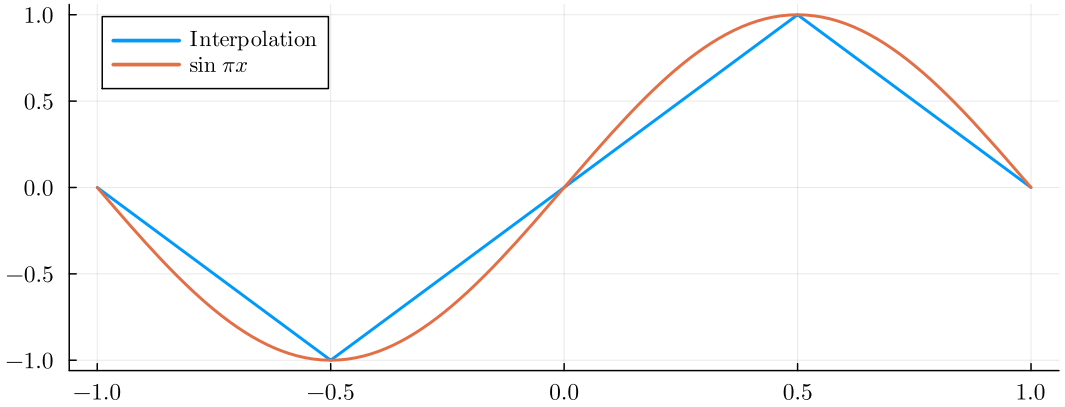


Figure 2.4: A comparison between linear interpolation of $\sin \pi x$ and the original function.

```
using BasicInterpolators
xvals = -1:0.5:1
yvals = sin.(π * xvals)
itp = LinearInterpolator(xvals, yvals)
plot(x -> itp(x), -1, 1)
plot!(x -> sin(π * x), -1, 1)
```

The comparison between the above-computed linear interpolation and the original function is shown in figure 2.4.

Chapter 3

Problem Statement and Analysis

This chapter summarises the optimal control task, which is to be solved in this thesis. The problem is stated in the Lagrange form where the cost function is presented as an integral of an integrand.

3.1 System Model

The dynamics of a rail vehicle is modelled as a system of two first-order ordinary differential equations:

$$t' = \frac{1}{v}, \tag{3.1a}$$

$$v' = \frac{u(x,t,v) - r(v) + g(x)}{v}, \tag{3.1b}$$

where the states t and v are time and velocity, respectively, $u(x,t,v)$ is the control input, $r(v)$ is the net resistant force per unit mass and $g(x)$ is the accelerating component of the gravitational acceleration. It is important to note that the derivatives of (3.1) are taken with respect to the position on the track, x, and not with respect to time t (which is obvious since it is one of the states). This nuance makes it possible to use the track grade term $g(x)$ directly and, for example, simplifies inclusion of speed limits along the track.

Glancing at the state equations (3.1) reveals singularities at points (t,v) where $v = 0$. This problem can be circumvented by assuming that the speed is always greater than a small positive value (in the author's implementation, this value is set to $v = 0.01\,\mathrm{m\,s^{-1}}$). As it is argued in Khmelnitsky (2000) and Wang; De Schutter; van den Boom, et al.

(2013) and Albrecht et al. (2016a), this attitude is not restrictive since the optimal speed profile will never contain a deliberate stop apart from the boundary conditions described below.

The dynamics of the train and also all of the terms in the numerator of (3.1b) are expressed in the "per unit mass" form, also called specific form. This way, the influence of the train's mass is encapsulated in the individual functions which are obtained from the measured physical characteristics by dividing by the mass. The SI dimensions of these quantities are then $\mathrm{m\,s^{-2}}$. Although the state-space equations (3.1) essentially describe a train as if it were a point mass, it can be shown that the more general case of a train with mass distributed among multiple carriages can be reduced to the above form. The mass density can be integrated along the length of the train and the resulting effective mass can be used (Howlett; Cheng, et al., 1995).

The resistance term, $r(v)$ in (3.1) is usually given as a quadratic function

$$r(v) = a + bv + cv^2, \tag{3.2}$$

where a, b and c are coefficients of SI dimensions $[a] = \mathrm{m\,s^{-2}}$, $[b] = \mathrm{s^{-1}}$ and $[c] = \mathrm{m^{-1}\,s^{-6}}$ (Rochard et al., 2000). For the purpose of convenience in the subsequent chapters, two auxiliary functions related to the resistance term are (Albrecht et al., 2016a)

$$\varphi(v) = vr(v), \tag{3.3a}$$
$$\psi(v) = v^2 r'(v). \tag{3.3b}$$

Although the form of resistance term $r(v)$ remains fixed in the scope of this thesis, the expression defining $r(v)$ can be arbitrary and all of the theoretical results remain unchanged as long as the following properties hold (Albrecht et al., 2016a):

1. $\varphi(v)$ is strictly convex (i.e. $\varphi(v) > \varphi(V) + \varphi'(V)(v - V)$ for all $v \neq V$),
2. $\varphi(v) \geq 0$ for all $v \geq 0$,
3. $\lim\limits_{v\to\infty} \frac{\varphi(v)}{v} = \infty$.

From these properties, it can be shown that $r(v)$ and $\psi(v)$ are non-negative and strictly increasing, which are indeed real-life characteristics of the resistance term.

In the implemented package, the resistance function in the Davis form (3.2) is declared as its own type:

```
struct DavisResistance <: Resistance
    a::Real
    b::Real
    c::Real
end
```

The value of resistive acceleration can be obtained by calling the function

```
function resistance(r::DavisResistance, v)
    r.a + r.b * v + r.c * v^2
end
```

A careful reader might have noticed that the `DavisResistance` type was defined as a subtype of the abstract type `Resistance`. This relationship is denoted using the `SubType <: SuperType` construction. This is very useful since we can define functions to work on the abstract types. Should a new kind of the resistance term be needed, one can simply define a new type `NewResistance` that will also be subtype of `Resistance`. All should work as expected if we also provide the `resistance(r::NewResistance, v)` function. This attitude of defining abstract types in hindsight and writing code in the most general way possible is encouraged in the Julia ecosystem.

3.2 Cost Function

The objective is to minimise the functional*

$$J(u(\cdot)) = \int_0^X \left(\frac{u + |u|}{2} + \rho \frac{u - |u|}{2} \right) \mathrm{d}x = \int_0^X l(u(\cdot))\, \mathrm{d}x\,, \tag{3.4}$$

where X is the total length of the track and $\rho \in [0, 1)$ is the portion of energy which is recovered during braking (ρ is called regeneration coefficient; $\rho > 0$ if the vehicle has the capability of regenerative braking). Upon inspection, it can be seen that the integrand $l(u(\cdot))$ changes behaviour depending on the sign of $|u|$:

$$l(u(\cdot)) = \begin{cases} u, & u > 0, \\ 0, & u = 0, \\ \rho u, & u < 0. \end{cases} \tag{3.5}$$

3.3 Constraints

In addition to the state equations (3.1), boundary conditions

$$t(0) = 0, \tag{3.6a}$$
$$v(0) = v_\mathrm{i}, \tag{3.6b}$$
$$t(X) = T, \tag{3.6c}$$
$$v(X) = v_\mathrm{f}, \tag{3.6d}$$

*The notation $f(\cdot)$ means that f is a function.

have to be fulfilled for the initial speed v_i, final speed v_f and the total time T specified in advance. The control effort is also limited by bounds which usually vary with speed of the vehicle, i.e.

$$U_-(v) \leq u(v\ldots) \leq U_+(v) \tag{3.7}$$

has to hold for every $v > 0$. The control bounds $U_-(v)$ and $U_+(v)$ are both monotone ($U_-(v)$ is non-decreasing and $U_+(v)$ is non-increasing) with

$$\lim_{v\to\infty} U_-(v) = 0, \tag{3.8}$$

$$\lim_{v\to\infty} U_+(v) = 0. \tag{3.9}$$

An example of a $U_+(v)$ can be inspected in figure 3.1. It is assumed that it is possible

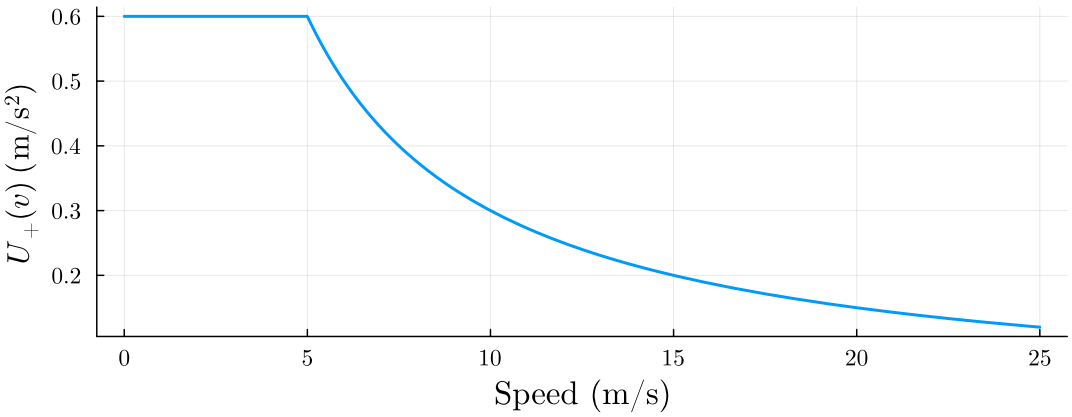


Figure 3.1: Example of a control bound $U_+(v) = 3/\max(5, v)$. Traction characteristics of this shifted-hyperbolic type are often present in practice (although the numerical values can be different). This example is from Howlett; Pudney, et al. (2009).

for the train to come to a full halt at any point on the track, or

$$U_-(v) < r(v) - g(x), \quad x \in [0, X], v > 0. \tag{3.10}$$

This is an obvious safety assumption although a train usually has multiple braking systems, which are used depending on the severity of the driver requirement. There is no need for the emergency situations requiring all available braking power to be modelled since these occur very rarely and have to be handled by the train operator anyway (Albrecht et al., 2016a).

The complete statement of the problem is

$$\text{minimise} \int_0^X \left(\frac{u+|u|}{2} + \rho\frac{u-|u|}{2}\right) \mathrm{d}x\,, \quad \text{subject to} \tag{3.11a}$$

$$t' = \frac{1}{v}, \qquad v' = \frac{u - r(v) + g(x)}{v}, \tag{3.11b}$$

$$t(0) = 0, \qquad t(X) = T, \tag{3.11c}$$

$$v(0) = v_\mathrm{i}, \qquad v(X) = v_\mathrm{f}, \tag{3.11d}$$

$$U_-(v) \leq u \leq U_+(v). \tag{3.11e}$$

3.4 Pontryagin's Principle and Possible Control Modes

This section describes the Hamiltonian treatise of the problem as described in Albrecht et al. (2016a). The control Hamiltonian can be written as

$$\begin{aligned}\mathcal{H}(x, v, \lambda_t, \lambda_v, u) &= \lambda_t \frac{1}{v} + \lambda_v \frac{u - r(v) + g(x)}{v} - l(u(\cdot)) \\ &= \lambda_t \frac{1}{v} + \lambda_v \frac{u - r(v) + g(x)}{v} - \frac{u + |u|}{2} - \rho \frac{u - |u|}{2}, \end{aligned} \tag{3.12}$$

where the costate variables λ_t and λ_v were introduced. These serve as Lagrange multipliers to the continuous-time constraints, i.e. the state differential equations (3.1). In addition, there is the requirement that the control effort must lie within the restricted range given by (3.7). Together with (3.12), this forms the augmented Lagrangian[†]

$$\mathcal{L} = \mathcal{H} + \mu_+(U_+(v) - u) + \mu_-(u - U_-(v)), \tag{3.13}$$

where two new Lagrange multipliers, $\mu_+ \geq 0$ and $\mu_- \geq 0$, are introduced. The dynamics of the costate variables is defined as

$$\lambda_t' = -\frac{\partial \mathcal{L}}{\partial t} = 0, \tag{3.14}$$

$$\lambda_v' = -\frac{\partial \mathcal{L}}{\partial v} = -\frac{\lambda_t}{v^2} - \lambda_v \frac{r'(v)v - u + r(v) - g(x)}{v^2} - \mu_+ U_+'(v) + \mu_- U_-'(v). \tag{3.15}$$

In order to find the optimal control law u, the augmented Lagrangian has to be maximised, which is the formulation of Pontryagin's principle in Albrecht et al. (2016a). We seek to find such u to set

$$\begin{aligned} 0 = \frac{\partial \mathcal{L}}{\partial u} &= -\left(\frac{1 + \operatorname{sgn} u}{2} + \rho \frac{1 - \operatorname{sgn} u}{2}\right) + \frac{\lambda_v}{v} - \mu_+ + \mu_- \\ &= \begin{cases} -1 + \frac{\lambda_v}{v} - \mu_+ + \mu_-, & u > 0, \\ -\rho + \frac{\lambda_v}{v} - \mu_+ + \mu_-, & u < 0. \end{cases} \end{aligned} \tag{3.16}$$

Furthermore, there are also the complementary slackness Karush-Kuhn-Tucker conditions, which need to be satisfied (i.e. one of the factors in the products has to be equal to zero and both are non-negative):

$$\mu_+(U_+(v) - u) = 0, \tag{3.17a}$$

$$\mu_-(u - U_-(v)) = 0. \tag{3.17b}$$

What follows now is a discussion of all the cases which can occur and which correspond to the five possible modes of control. Optimal control law can be found by solving (3.16) while keeping (3.17) in mind.

[†]The term augmented Lagrangian has different meaning in optimal control theory than in the field of constrained optimisation.

- The case when $u > 0$:
  - $u = U_+(v)$: To fulfil the complementary condition (3.17), it is necessary to have $\mu_+ > 0$ and $\mu_- = 0$, which in combination with (3.16) yields $\mu_+ = \frac{\lambda_v}{v} - 1 > 0 \implies \lambda_v > v$.
  - $u < U_+(v)$: We necessarily have $\mu_+ = \mu_- = 0$, so together with (3.16), the condition $\lambda_v = v$ is obtained.
- The case when $u = 0$: Although the derivative of the absolute value in the criterion (3.4) is not defined at zero, we can use the expressions for the positive and negative case of u in (3.16) to consider their limit cases when $u = 0$ and, consequently, $\mu_+ = \mu_- = 0$. We get two conditions: $\lambda_v = v$ and $\lambda_v = \rho v$. These are the boundaries for the possible values of λ_v when $u = 0$. In conclusion, $\rho v < \lambda_v < v$[‡].
- The case when $u < 0$:
  - $u > U_-(v)$: We necessarily have $\mu_+ = \mu_- = 0$, so together with (3.16), the condition $\lambda = \rho v$ is obtained.
  - $u = U_-(v)$: To fulfil the complementary condition (3.17), it is necessary to have $\mu_- > 0$ and $\mu_+ = 0$, which in combination with (3.16) yields $\mu_- = \rho - \frac{\lambda_v}{v} > 0 \implies \lambda_v < \rho v$.

For the cases of $\lambda_v = \rho v$ and $\lambda_v = v$, there is no explicit formula for the control signal u. All that was gained from the analysis were the bounds $u > U_-(v)$ for the first case and $u < U_+(v)$ for the second one. However, such formula can be obtained using the following reasoning from Albrecht et al. (2016a).

Should the condition $\lambda_v = v$ hold on an interval $[a, b]$, then the derivatives of the λ_v costate and the speed v must be equal to each other. Therefore, comparing (3.1) and (3.15) gives

$$-\frac{\lambda_t}{v^2} + \frac{u - r(v) + g(x)}{v} + r'(v) = \frac{u - r(v) + g(x)}{v}, \tag{3.18}$$

which in turn yields

$$\lambda_t - v^2 r'(v) = \lambda_t - \psi(v) = 0. \tag{3.19}$$

The auxiliary function is strictly increasing, as was established above, and the equation (3.19) has a uniquely defined root for every fixed $\lambda_t < 0$, which is denoted as V. This root is called "cruising speed" or "travel speed" and is held on the entirety of the interval $[a, b]$. In order to achieve this, the control signal must necessarily be $u(V, x) = r(V) - g(x)$.

Similar procedure can be followed in the case of the equality $\lambda_v = \rho v$, being true on an interval $[a, b]$. The equality of derivatives of λ_v and v provides the equation

$$\frac{\lambda_t}{v^2} + \rho\frac{u - r(v) + g(x)}{v} + \rho r'(v) = \rho\frac{u - r(v) + g(x)}{v}, \tag{3.20}$$

[‡]One could argue that it is not shown that the feasible values for λ_v for $u = 0$ lie between the calculated boundary points or outside of the boundaries. This is, however, resolved by calculating the other control modes and avoiding the overlap between them in the (λ_v, v)-space.

or in other terms

$$\lambda_t + \rho v^2 r'(v) = \lambda_t + \rho\psi(v) = 0. \tag{3.21}$$

As before, because of the strict monotonicity of the auxiliary function $\psi(v)$, the equation (3.21) has a unique root for every $\lambda_t < 0$. This root, denoted W, is called "optimal braking speed". The control signal is then given as $u(W, x) = r(W) - g(x)$.

Both of the found holding speeds, V and W, depend on the total time constraint (4.13) through the λ_t multiplier. The influence of the total time of journey T on the cruising speed and optimal braking speed is discussed in subsequent chapters.

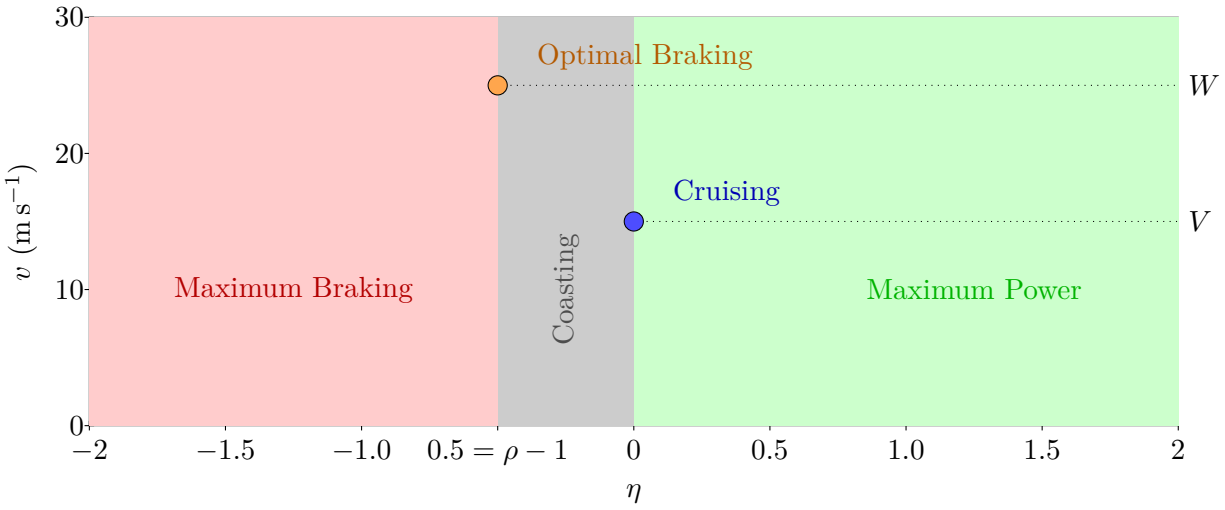


Figure 3.2: Depiction of the $\eta - v$ plane and its corresponding control regions. For the purpose of the figure, the regenerative ratio $\rho = 0.5$, the cruising speed is set to $V = 15\,\mathrm{m\,s^{-1}}$ and the optimal braking speed is $W = 25\,\mathrm{m\,s^{-1}}$.

Overall, we have five different modes of control:

1. Maximum Power: Applying maximal tractive effort. Regular mode.
2. Cruising: Holding constant preset travel speed V. Singular mode.
3. Coasting: Not applying any tractive or braking effort. Regular mode.
4. Optimal Braking: Holding constant optimal braking speed $W > V$. Singular mode.
5. Maximum Braking: Applying maximal braking effort. Regular mode.

The decision of which regular control mode is to be engaged is determined by the values of the multiplier λ_v and the vehicle speed v. The case of singular control modes is more complicated and their incorporation of the optimal control strategy is the subject of subsequent chapters. It is useful to define a dimensionless quantity $\eta = \frac{\lambda_v}{v} - 1$ for writing the decision conditions more clearly:

1. Maximum Power: $\eta > 0$.
2. Cruising: $\eta = 0$, $v = V$.
3. Coasting: $\rho - 1 < \eta < 0$.
4. Optimal Braking: $\eta = \rho - 1$, $v = W$.
5. Maximum Braking: $\eta < \rho - 1$.

These conditions are shown graphically in figure 3.2. It has been shown in Albrecht et al. (2016a) and Albrecht et al. (2016b) that the variable η evolves according to the differential equation

$$\eta' = \begin{cases} \frac{\psi(v)-v^2u'}{v^3}\eta + \frac{\psi(v)-\psi(V)}{v^3}, & u(v) > U_-(v), \\ \frac{\psi(v)-v^2u'}{v^3}\eta + \frac{\psi(v)-\psi(V)}{v^3} - \frac{(1-\rho)u'}{v}, & u(v) = U_-(v). \end{cases} \tag{3.22}$$

It is useful to visualise the relationship between the cruising speed V and the optimal braking speed W by inspecting figure 3.3. It can be observed that $W > V$ for all $\rho \in [0, 1)$. The only exception is the (probably unrealistic) case when $\rho = 1$. According to Khmelnitsky (2000), in such a situation the three phases of Cruising, Optimal Braking and Coasting collapse into a single mode of control. Further examining the figure 3.3, W decreases and gets closer to V as the regeneration coefficient increases.

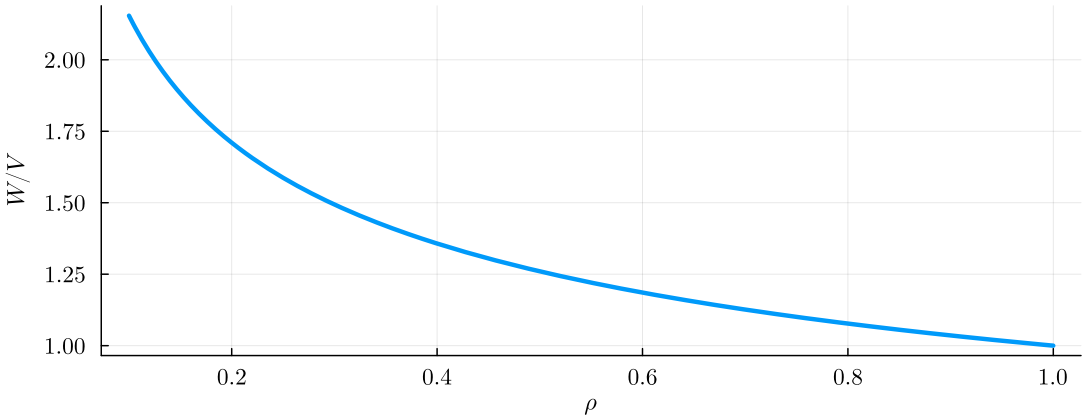


Figure 3.3: Relationship between the optimal between the cruising speed V and the optimal braking speed W depending on the value of $\rho \in [0, 1)$.

Chapter 4

Optimal Train Control on a Flat Track

As it was shown in the previous chapter, the optimal control can only consist of Maximum Power phase, Coasting phase, Maximum Braking phase and two cruising phases. In this chapter, the form of optimal speed profiles (with respect to the cost (3.4)) on a flat track is determined.

4.1 Problem Statement and Analysis

On a flat track, the general problem of minimising the cost function (3.4) under the dynamics given by the differential equations (3.1) and constraints in the control signal (3.7) as well as the boundary conditions (3.6) is simplified. The only (but significant) simplification is in the second of the two state equations

$$t' = \frac{1}{v}, \tag{4.1a}$$

$$v' = \frac{u(x,t,v) - r(v)}{v}, \tag{4.1b}$$

where the $g(x)$ term (component of gravitational acceleration dependent on the gradient of the track) is missing in the numerator of the right-hand side of the second equation.

4.1.1 Optimality of Cruising Mode

Before we further analyse the form of optimal control for the stated problem, it is useful to establish that holding speed (Cruising) is the most efficient control mode with respect

to the cost (3.4) compared to any alternative control with $u > 0$. The fact is the key to obtain complete optimal control strategies. The following proof was originally presented in Albrecht et al. (2016a).

Let the Cruising phase with holding speed V take place on interval $x \in [a, b]$. The driving control is therefore $u(V) = r(V) > 0$ (the control precisely compensates the motion resistance). The cost of the phase is given by

$$J_V = \int_a^b u(V)\,\mathrm{d}x = \int_a^b r(V) = r(V)(b-a). \tag{4.2}$$

Now consider a different driving control $u(x) > 0$ such that $v(a) = v(b) = V$ (i.e. the speed at the endpoints of the interval is the same as the original cruising speed). We want to show that the latter driving control yields higher cost than the former one. The state equation (4.1b) can be multiplied by v:

$$vv' = u(x) - r(v(x)). \tag{4.3}$$

This equation can now be integrated over the interval $[a, b]$. First, let us focus on the left-hand side of (4.3):

$$\int_a^b v(x)v'(x)\,\mathrm{d}x = \int_V^V w\,\mathrm{d}w = 0, \tag{4.4}$$

where the substitution rule with $w = v(x)$ is used. Integrating the right-hand side of (4.3) together with this result gives

$$0 = \int_a^b u(x)\,\mathrm{d}x - \int_a^b r(v(x))\,\mathrm{d}x\,, \tag{4.5}$$

which together with the definition of the criterion (3.4) results in formula for cost J_v of the scenario with the alternative speed profile $v(x)$:

$$J_v = \int_a^b r(v(x))\,\mathrm{d}x\,. \tag{4.6}$$

Let us now analyse the difference $J_v - J_V$. If this expression is positive, then the optimality of the speed-holding strategy will be proofed. Using (4.6), we have

$$J_v - J_V = \int_a^b [r(v(x)) - r(V)]\,\mathrm{d}x\,. \tag{4.7}$$

Utilising the strict convexity of the auxiliary function $\varphi(v) = vr(v)$, we can obtain a lower bound of the integrand from the previous equation:

$$\varphi(v) - \varphi(V) > \varphi'(V)(v - V) \quad \Longleftrightarrow \quad r(v(x)) - r(V) > Vr'(V)\left(1 - \frac{V}{v}\right). \tag{4.8}$$

Plugging the obtained lower bound into the integral (4.7) yields

$$\int_a^b [r(v(x)) - r(V)]\,\mathrm{d}x > Vr'(V) \int_a^b \left[1 - \frac{V}{v(x)}\right] \mathrm{d}x\,, \tag{4.9}$$

provided $v(x) \neq V$ for all x in the interior of $[a, b]$. If we limit ourselves to only such trajectories $v(x)$ that take the same amount of time to reach b from a as the speed-holding one, it must hold that

$$T = \frac{b-a}{V} = \int_a^b \frac{1}{V}\,\mathrm{d}x = \int_a^b \frac{1}{v(x)}\,\mathrm{d}x = T. \tag{4.10}$$

The integral on the right-hand side of (4.9) can now be evaluated:

$$Vr'(V) \int_a^b \left[1 - \frac{V}{v(x)}\right] \mathrm{d}x = Vr'(v)\,[b - a - (b - a] = 0. \tag{4.11}$$

Overall, it was shown that

$$J_v - J_V > 0 \tag{4.12}$$

which proves that the Cruising driving mode is the most efficient among all other possible control strategies with $u > 0$.

4.2 Optimal Sequence of Control Modes

In the previous section, it has been proofed that the Cruising driving regime is optimal with respect to the criterion (3.4). Therefore, the cruising phase is to be held for as long as possible while respecting the boundary conditions (3.6).

The most influential boundary condition is the total journey time requirement of

$$t(X) = T. \tag{4.13}$$

As it was shown in Khmelnitsky (2000) and also later in Albrecht et al. (2016b), the choice of the total available journey time T uniquely determines the cruising speed V (and, by extension, the optimal braking speed W). If the required T is reduced, the cruising speed is higher in order to satisfy the constraint (4.13).

The other boundary constraints stem from the set parameters of the initial speed (3.6b) and the final speed (3.6d). Because the initial speed v_i is, in general, not equal to the cruising speed V, it is necessary to enter the Cruising mode from the initial state. Two cases can occur:

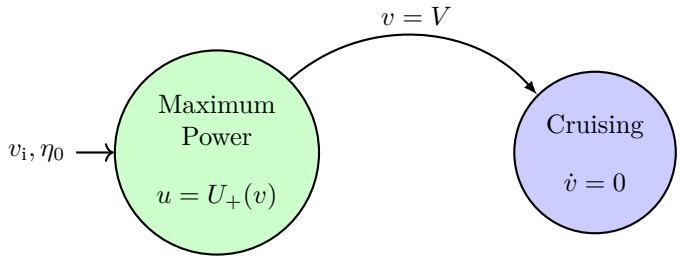


Figure 4.1: Visualisation of the optimal switching strategy on a flat track for the case of $v_\mathrm{i} < V$.

1. $v_\mathrm{i} < V$: Maximum Power mode is engaged until the cruising speed is reached.

2. $v_\mathrm{i} > V$: Coasting or Maximum Braking modes are active until the cruising speed is reached. The Maximum Braking regime does not have to be present in all cases in contrast to the Coasting phase, which is always present in the described situation.

Both cases are illustrated in figures 4.1 and 4.2, respectively.

Similar discussion can be made depending on the value of the final speed v_f:

1. $v_\mathrm{f} < V$: Coasting and possibly Maximum Braking are engaged to reach the final speed at the end of the track.

2. $v_\mathrm{f} > V$: Maximum Power is activated towards the end of the track.

Both cases are presented in figures 4.4 and 4.3, respectively. The overall sequence of control regimes (from the start of the track to its finish) can be acquired by "gluing" the corresponding figures based on the boundary conditions (for example diagrams 4.1 and 4.4 for the case of $v_\mathrm{i} < V$ and $v_\mathrm{f} < V$).

The above discussion is not entirely exhaustive since it can happen that the cruising speed V is never achieved. In such case, the Cruising phase is omitted and only the regular regimes can occur. Visualisation of an exemplar control mode sequence consisting only of regular modes is shown in figure 4.5.

It can be noticed that the Optimal Braking mode is completely missing in the above discussion. Indeed, the Optimal Braking regime cannot occur on a flat track since it would require braking with $u = r(W) - g(x)$ and slowing down, which is suboptimal. In the next chapter it is specified under which condition the Optimal Braking mode can be engaged.

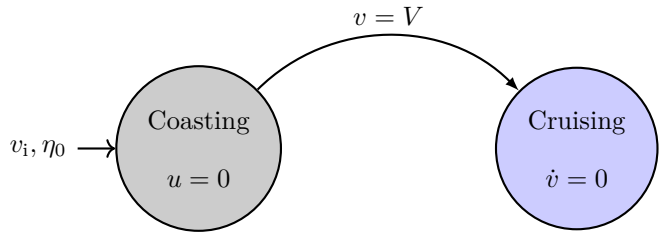


Figure 4.2: Visualisation of the optimal switching strategy on a flat track for the case of $v_\mathrm{i} > V$.

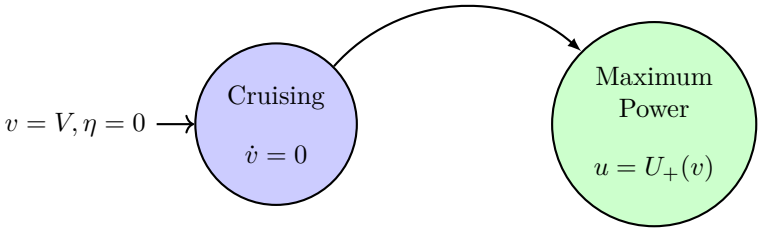


Figure 4.3: Visualisation of the optimal switching strategy on a flat track for the case of $v_\mathrm{f} > V$.

4.3 Track Representation

Since the only property that uniquely defines a particular instance of a flat track is its length, we can define the `type` in the following manner:

```
1 struct FlatTrack <: Track
2     X::Real
3 end
```

Using multiple dispatch, utility functions for the new type, such as the `length` function, can be implemented:

```
1     length(t::FlatTrack) = t.X
```

4.4 Solution

Although the functional details of the computation are explained in the next chapter, the basic usage of the implemented package is presented here for the case of a flat track.

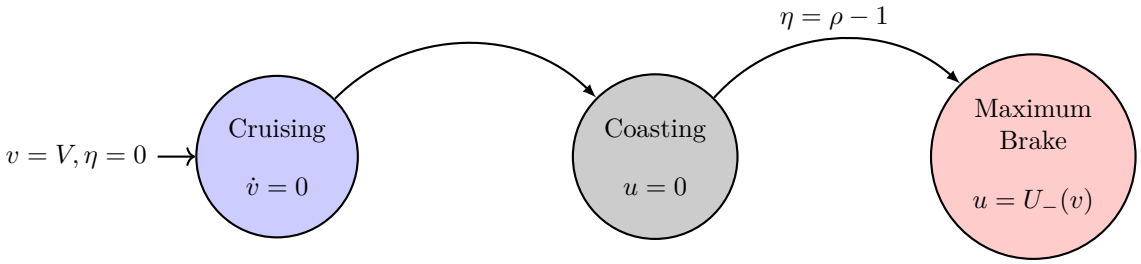


Figure 4.4: Visualisation of the optimal switching strategy on a flat track for the case of $v_\mathrm{f} < V$.

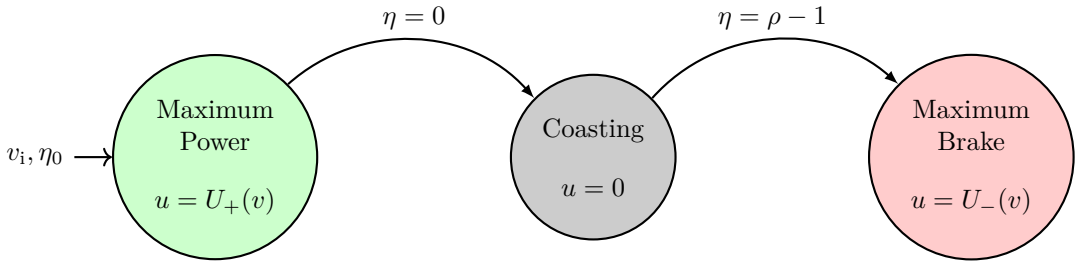


Figure 4.5: Visualisation of the optimal switching strategy on a flat track for the case of omitted Cruising phase with $v_\mathrm{i} < V$ and $v_\mathrm{f} < V$.

Consider the problem (3.11) with a flat track of the length of 10 km, initial speed $v_\mathrm{i} = 1\,\mathrm{m\,s^{-1}}$, final speed $v_\mathrm{f} = 1\,\mathrm{m\,s^{-1}}$, journey time $T = 1400\,\mathrm{s}$, control limits

$$U_-(v) = -\frac{3}{\max(5, v)}, \tag{4.14a}$$

$$U_+(v) = \frac{3}{\max(5, v)}, \tag{4.14b}$$

and, finally, the resistance term $r(v) = a + bv + cv^2$ with $a = 1 \times 10^{-2}$, $b = 0$, $c = 1.5 \times 10^{-5}$ (the values are taken from the examples in Albrecht et al. (2016a)).

The implemented package takes inspiration from DifferentialEquations.jl with its problem-solve interface. Using this approach, the problem above is defined with the problem data passed in as keyword arguments to `TrainProblem` type:

```
using OptimalTrainControl
prob = TrainProblem(; track = FlatTrack(10e3), T = 1400)
```

As it can be seen, only the track and journey time are specified whereas the other values are set by default to the above values.

The solution is obtained by simply calling the `solve!` function:

```
points, sol = solve!(prob)
plot(sol.t,sol[2,:])
```

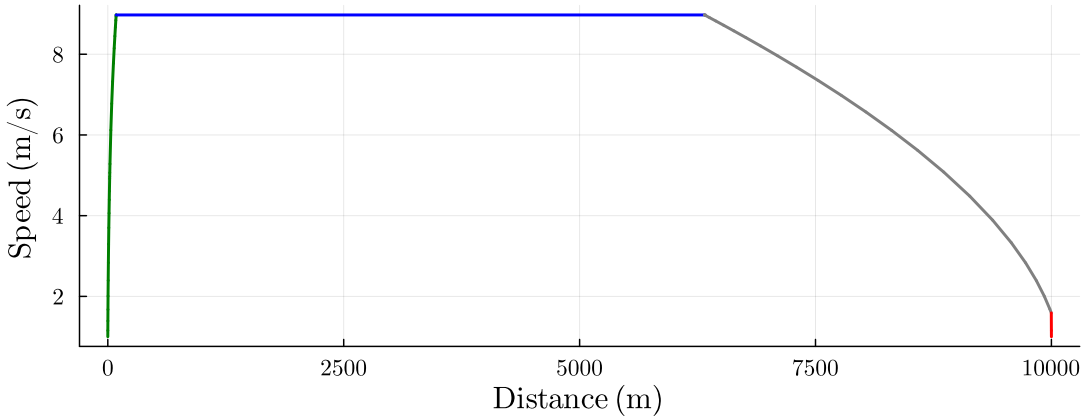


Figure 4.6: Optimal solution of the example problem. The cruising speed is computed to $V = 8.97\,\mathrm{m\,s^{-1}}$. The colour of the line signifies the current control regime: Maximum Power, Cruising, Coasting, Maximum Braking.

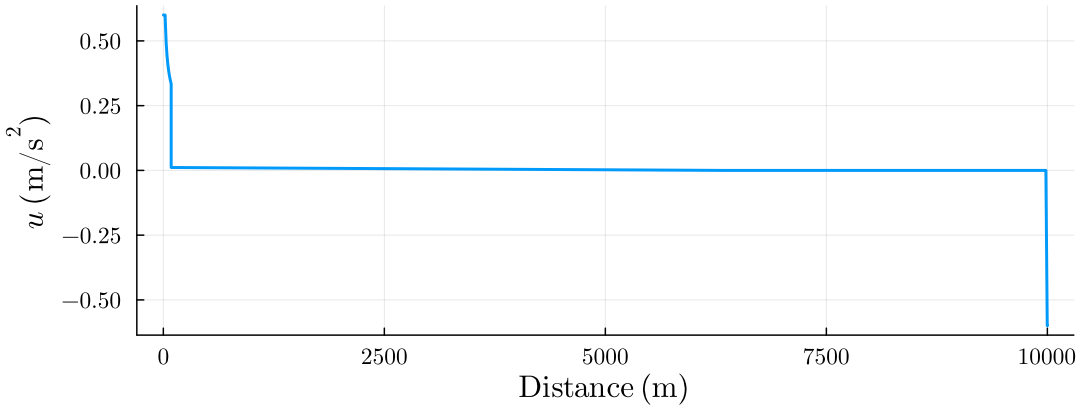


Figure 4.7: Optimal input u computed for the example of flat track of length 10 km.

The result can be inspected in figure 4.6. The `solve!` function returns a tuple of switching points between control modes and a solution object which can be handled the same way as in the DifferentialEquations.jl. The calculated switching points are (after rounding to integer values): (0: Maximum Power), (88: Cruising), (6324: Coasting), (9998: Maximum Braking), (10000: Stop). The time of arrival is 1400.5958 s (the default tolerance for termination of the algorithm is 5 s). The optimal control can be calculated separately using the following code:

```
u = calculatecontrol!(prob, sol, points)
plot(sol.t, u)
```

The computed optimal control u is plotted in figure 4.7.

Although the above example is very simple, it is useful for demonstrating and visualising

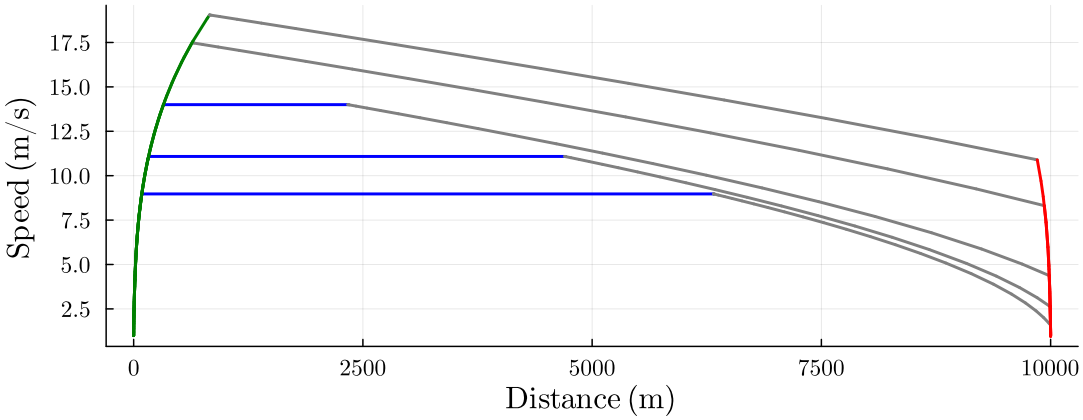


Figure 4.8: Optimal speed profiles for varying journey times, $T \in \{1400\,\mathrm{s}, 1200\,\mathrm{s}, 1000\,\mathrm{s}, 800\,\mathrm{s}, 700\,\mathrm{s}\}$. The calculated cruising speeds are $8.97\,\mathrm{m\,s^{-1}}$, $11.08\,\mathrm{m\,s^{-1}}$, $14.0\,\mathrm{m\,s^{-1}}$, $19.75\,\mathrm{m\,s^{-1}}$, $23.57\,\mathrm{m\,s^{-1}}$, respectively. Notice that in the last two cases, the cruising speed is never reached.

important concepts. Firstly, the cruising speed should increase if less time is provided for the journey's completion. In figure 4.8, multiple optimal speed profiles are drawn. Each is generated with the same problem data, but with different total journey time T. It can be immediately observed that the computed cruising speed is higher when less journey time is provided. It may happen that the cruising speed is not achieved at all on the optimal speed profile (as it is the case for two of the profiles in figure 4.8). Although not obvious at first glance, the choice of the cruising speed still matters in such situations because it influences the portion of time spent in the Coasting mode. The higher the cruising speed V, the less time is spent in the Coasting mode. In the limit case of $V \to \infty$, the minimal-time speed profile would be obtained, which would consist of only Maximum Power and Maximum Braking phases.

Although the Optimal Braking mode cannot occur on a flat track, the influence of the regenerative coefficient ρ can also be examined. See figure 4.9 where multiple optimal speed profiles are shown for varying values of ρ. For higher values of ρ, the criterion (3.4) yields smaller cost for the Maximum Braking phase, which makes it possible for the Cruising mode to last for longer period. In the limit case when all braking energy is regenerated ($\rho = 1$), the Coasting phase disappears and the optimal strategy consists of only Maximum Power, Cruising and Maximum Braking modes.

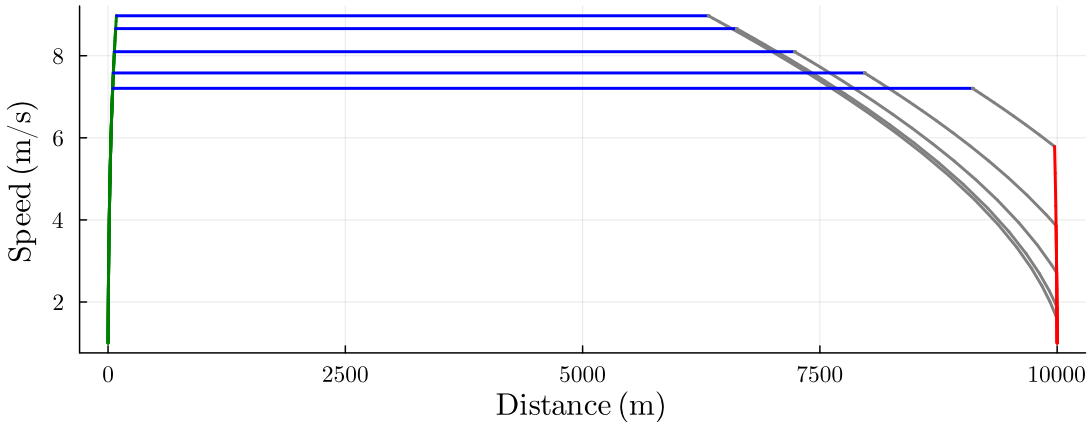


Figure 4.9: Optimal speed profiles for varying regenerative coefficient $\rho \in \{0, 0.3, 0.7, 0.9, 0.99\}$. The calculated cruising speeds are $8.97\,\mathrm{m\,s^{-1}}$, $8.66\,\mathrm{m\,s^{-1}}$, $8.10\,\mathrm{m\,s^{-1}}$, $7.58\,\mathrm{m\,s^{-1}}$, $7.21\,\mathrm{m\,s^{-1}}$, respectively.

Chapter 5

Optimal Train Control on a General Track

In this chapter, the form of the optimal control with respect to the criterion (3.4) on a general track with non-flat terrain is described.

5.1 Track Representation

Before moving further, let us define the `HillyTrack` type, which can represent a general track. It consists of a `DataFrame` containing the waypoints in the form of a vector of distances along the track and a vector of altitude values at those points:

```
function HillyTrack(X, Y)
    df = DataFrame("Distance" => X, "Altitude" => Y)
    HillyTrack(df, LinearInterpolator(X, Y))
end
```

After creating the instance, the structure uses the linear interpolator from BasicInterpolators.jl to find altitude values at points which are not defined by the waypoints:

```
trackX = [0, 2e3, 3e3, 5e3, 5.4e3, 8e3]
trackY = [0, 0, 35, 35, 30, 30]
track = HillyTrack(trackX, trackY)
track(2156) # = 5.46
plot(track) # implemented using RecipesBase
```

The created track is shown in figure 5.1.

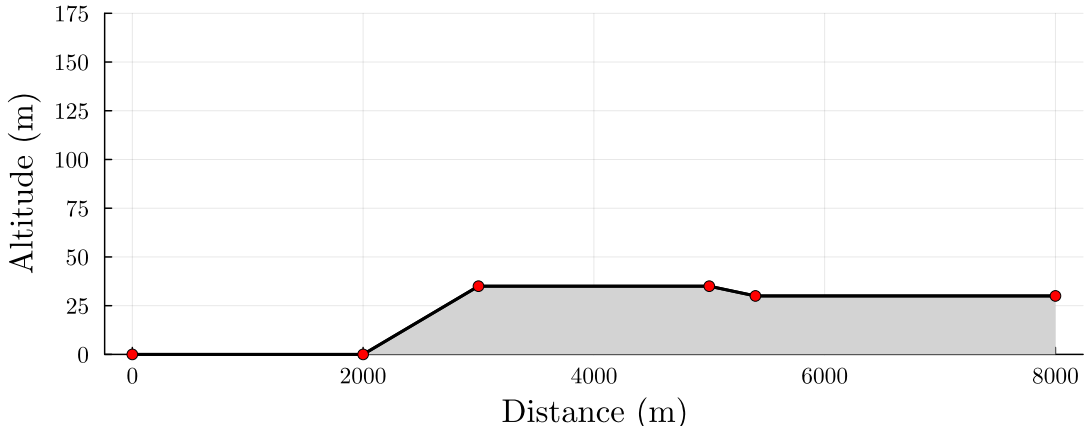


Figure 5.1: Visualisation of an instance of the `HillyTrack` type. The red circles show the waypoints.

5.2 Analysis and Nomenclature

Before we further analyse the form of optimal control for the case of a general track, it should be restated that the holding speed (Cruising) is the optimal control phase with respect to the criterion (3.4). This fact can be proofed in the same way as in section 4.1.1, apart from the fact that the state equations have the general form of (3.1). Therefore, when calculating the costs of both the Cruising phase and a different $v(x)$ trajectory, we get

$$J_V = \int_a^b u(V,x)\,\mathrm{d}x = \int_a^b (r(V) - g(x))\,\mathrm{d}x = r(V)(b-a) - G(b) + G(a), \tag{5.1}$$

$$J_v = \int_a^b (r(v(x)) - g(x))\,\mathrm{d}x = \int_a^b r(v(x))\,\mathrm{d}x - G(b) + G(a), \tag{5.2}$$

where $G(x)$ is the primitive function to $g(x)$. In spite of this, further steps of the proof remain the same since the difference of J_v and J_V is examined, which is the same as in the original case:

$$J_v - J_V = \int_a^b [r(v(x)) - r(V)]\,\mathrm{d}x - G(b) + G(a) + G(b) - G(a) = \int_a^b [r(v(x)) - r(V)]\,\mathrm{d}x\,. \tag{5.3}$$

Establishing that Cruising mode is optimal even for the case of a general track, it is time to focus on the underlying contrast with the problem solution on a flat track. The key fact is that there can exist certain portions (called "segments") on a track where holding speed is not possible due to the traction limits set by $U_+(v)$ and $U_-(v)$.

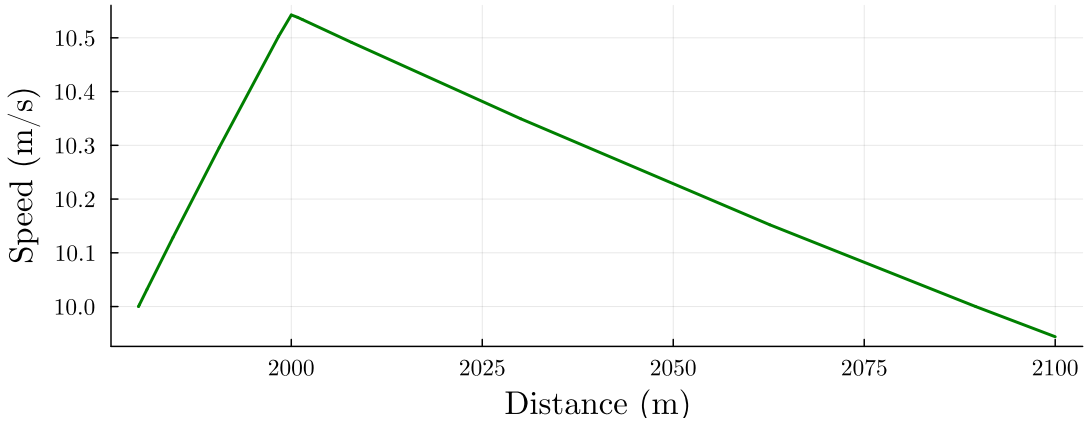


Figure 5.2: The speed profile on a steep uphill segment (from the 2nd kilometre on the track 5.1. The speed decreases even though Maximum Power mode is engaged. The cruising speed is $10\,\mathrm{m\,s^{-1}}$ and $U_+(v) = 3/\max(5, v)$.

We will now analyse conditions to correctly identify such segments of a track (Albrecht et al., 2016a).

5.2.1 Steep Uphill

Consider the situation where it is desired to hold current speed V with positive control, $u > 0$. This can be achieved for a flat and uphill terrain if the control effort manages to overcome both gravitational influence and the mechanical and aerodynamic resistances combined in the $r(V)$ term of (3.1b). A problem arises when the maximal control effort $u = U_+(V)$ fails to compensate these forces and the speed necessarily decreases. Such segments are called "steep uphill". An example of a speed profile on a steep uphill portion of a track is shown in figure 5.2. The following algebraic condition holds on steep uphill segments at speed V:

$$U_+(v) < r(V) - g(x). \tag{5.4}$$

It is important to emphasise that steep uphill segments depend on the preset cruising speed V, so if a portion of a track is steep uphill for speed V_1, it does not necessarily have to be the case that the segment is also steep uphill for speed $V_2 < V_1$. On the other hand, if a segment is steep uphill at speed V_1, it is also steep uphill for speed $V_2 > V_1$. This is due to the fact that the resistance term $r(v)$ is a strictly increasing function.

5.2.2 Steep Downhill

On certain portions of a track, it may be impossible to hold the speed with positive control, but because of other reasons than in the previous section. The slope of the track

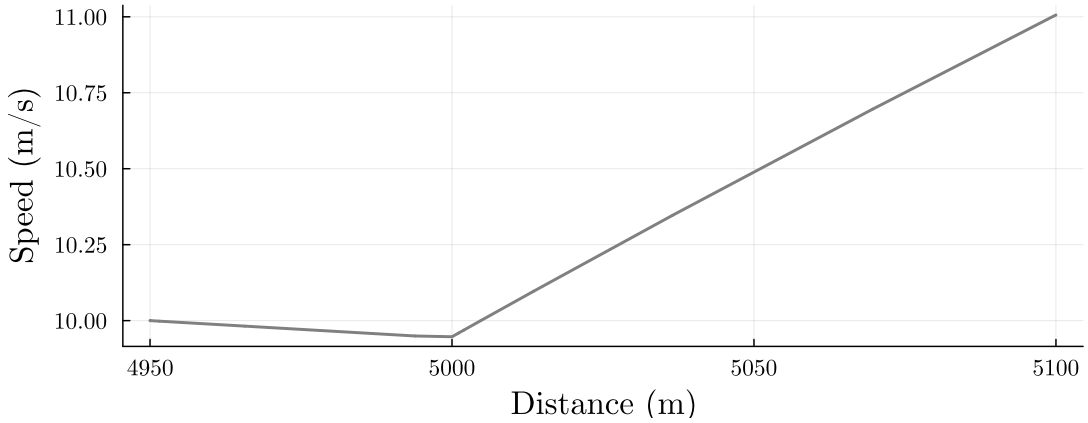


Figure 5.3: The speed profile on a steep downhill segment (from the 5th kilometre on the track 5.1. The speed increases even if Coasting mode is engaged. The cruising speed is $10\,\mathrm{m\,s^{-1}}$.

can be so steep that the vehicle accelerates itself even if there is no tractive effort being made. Such segments are called "steep downhill" and are characterised by the relation

$$g(x) > r(V), \tag{5.5}$$

i.e. the speed increases due to the influence of gravitational acceleration, which overcomes the resistive influence $r(V)$.

As in the case of steep uphill, the definition of steep downhill segments is dependent on the choice of holding speed V. If a segment is steep downhill at speed V_1, it is not necessarily steep downhill at speed $V_2 > V_1$. Conversely, a segment that is steep downhill at speed V_1 is also steep downhill at speed $V_2 < V_1$ because $r(v)$ is a strictly increasing function.

The Optimal Braking mode consists of holding optimal braking speed W with negative control $U_-(v) < u < 0$. Therefore, the Optimal Braking phase can occur only at portions of the track which are steep downhill at speed W. In the above example track 5.1, the downhill segment starting at $x = 5\,\mathrm{km}$ and ending at $x = 5.4\,\mathrm{km}$ is steep downhill at speed $W = 14.94\,\mathrm{m\,s^{-1}}$, which is the optimal braking speed corresponding to the cruising speed $V = 10\,\mathrm{m\,s^{-1}}$ and the resistance $r(v) = a + cv^2$ with $a = 1 \times 10^{-2}\,\mathrm{m\,s^{-2}}$, $c = 1.5 \times 10^{-5}\,\mathrm{m^{-1}}$.

5.3 Solution

As mentioned above, the key difference between the problem (3.11) involving a flat track and the same problem involving a general track lies in the fact that Cruising mode does

not have to be feasible for some portions of the track due to the train's traction limits. The key idea of the solution lies in identifying these problematic track segments and then traversing them with the use of the regular control modes (Maximum Power, Coasting, Maximum Braking). The original algorithm was proposed in Khmelnitsky (2000) and it has been adapted here for the system (3.1).

The overall structure of the algorithm is following:

1. Choose initial value of the cruising speed V. Calculate the optimal braking speed W.
2. Find segments of the track on which Cruising or Optimal Braking regimes can take place. The Cruising mode is only possible on portions of the track which are neither steep uphill, nor steep downhill at speed V. The Optimal Braking mode can be engaged only on segments which are steep downhill at speed W, as it is stated in the previous section.
3. Link the segments found in the previous step using only the regular control modes and obtain pairwise linkages (control mode switching points) between the individual segments.
4. Based on the boundary conditions (3.6), assemble the complete speed profile based on the linkages from the previous step.
5. Check if the total time of the journey $\tilde{T}$ matches the desired T (up to the set tolerance, the default one is $\pm 5\,\mathrm{s}$). Return the computed speed profile if T has been achieved, otherwise set a different cruising speed (increase it if $\tilde{T} > T$, decrease it if $\tilde{T} < T$), recalculate W and go to step 2.

Further sections are dedicated to describing the above points in greater detail.

5.3.1 Finding Singular Segments

Portions of the track on which the singular control modes Cruising and Optimal Braking can occur are called "singular segments". These are detected in a way that exploits the tabulated input format of the `HIllyTrack` type. Since the rail grade (the slope of the track) can vary only at the set waypoints, the effective component of gravitational acceleration is computed at each midpoint $\tilde{x}_k$ of the interval between subsequent waypoints $[x_{k-1}, x_k]$*:

$$g_k = -G \sin(\tilde{x}_k). \tag{5.6}$$

We then iterate over the list of g_k and decide whether the interval $[x_{k-1}, x_k]$ has

*G denotes the magnitude of gravitational acceleration. In the code, this values is set to $9.81\,\mathrm{m\,s^{-2}}$.

1. Cruising mode, if it is not steep uphill at speed V, i.e.

$$U_+(V) - r(V) + g_k > 0, \tag{5.7}$$

and it is not steep downhill at speed V:

$$-r(V) + g_k \leq 0, \quad \text{or} \tag{5.8}$$

2. Optimal Braking mode, if the conditions from the previous point do not apply, $0 < \rho < 1$ and the interval is steep downhill at speed W:

$$-r(W) + g_k \geq 0. \tag{5.9}$$

If the neighbouring intervals are of the same type, they are merged into a single segment of the corresponding type.

The above functionality is implemented in the `segmentize` function of the package, which returns a list of `Segment`s. `Segment` is another convenient datatype which has the following definition:

```
mutable struct Segment
    start
    finish
    mode
    holdspeed
end
```

The `start` and `finish` fields are self-explanatory, the `mode` has values Cruising or Optimal Braking and `holdspeed` has the value corresponding to the `mode` of the segment (V for Cruising segments, W for Optimal Braking segments). The result of the `segmentize` function with the input data $V = 10\,\mathrm{m\,s^{-1}}$, $r(v) = a + cv^2$ with $a = 1 \times 10^{-2}\,\mathrm{m\,s^{-2}}$, $c = 1.5 \times 10^{-5}\,\mathrm{m^{-1}}$, $\rho = 0.3$ and the track 5.1 can be seen in figure 5.4.

5.3.2 Linking Inner Segments

Linking is the procedure of finding speed profiles (along with control mode switching points) between singular segments, described in the previous section. Using an elegant technique presented in Khmelnitsky (2000), it is possible to reformulate the problem as finding the root of an uni-variate monotone function, so-called "linking function". Although the linking process in Khmelnitsky (2000) is described for the dynamical model in the form (1.5), the notation used below is highly similar.

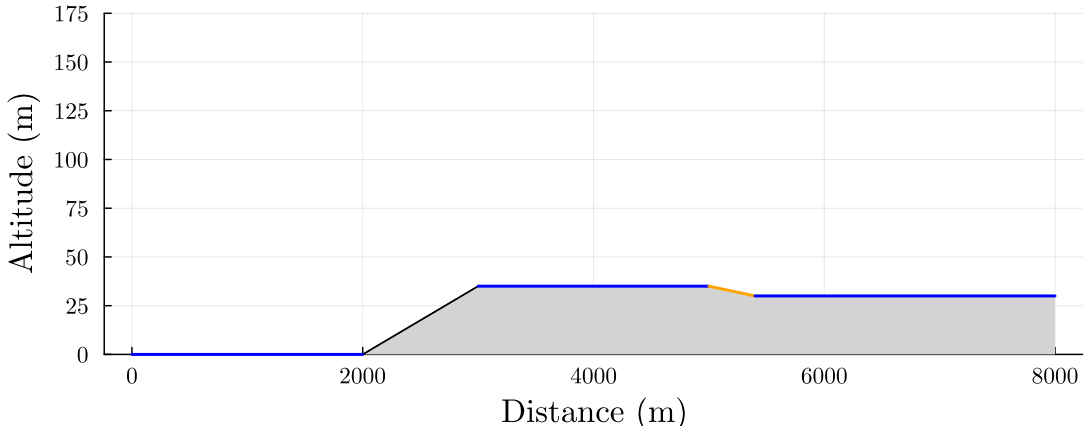


Figure 5.4: Visualisation of „segmentizing“ the example track 5.1. The Cruising segments are blue while the Optimal Braking is orange. The only uncoloured segment (black) is steep uphill at speed $V = 10\,\mathrm{m\,s^{-1}}$.

Consider the situation when two singular segments, S_1 and S_2, are to be linked. Also, let the conditions for leaving S_1, $C^{\mathrm{out}}(S_1)$, and entering S_2, $C^{\mathrm{out}}(S_2)$, be defined as triples in the form $(x, v(x), \eta(x))$ and

$$C^{\mathrm{out}}(S_1) = C^{\mathrm{in}}(S_1) = \{x, V, 0\} \quad \text{if } S_1 \text{ is a Cruising segment,} \tag{5.10}$$

$$C^{\mathrm{out}}(S_2) = C^{\mathrm{in}}(S_2) = \{x, W, \rho - 1\} \quad \text{if } S_2 \text{ is an Optimal Braking segment.} \tag{5.11}$$

The argument of the linking function, denoted F, is a position point x_{out} and the linking function is evaluated depending on the result of numerical integration of the system

$$t' = \frac{1}{v}, \tag{5.12a}$$

$$v' = \frac{u - r(v) + g(x)}{v}, \tag{5.12b}$$

$$\eta' = \begin{cases} \frac{\psi(v) - v^2 u'}{v^3}\eta + \frac{\psi(v) - \psi(V)}{v^3}, & u(v) > U_-(v), \\ \frac{\psi(v) - v^2 u'}{v^3}\eta + \frac{\psi(v) - \psi(V)}{v^3} - \frac{(1-\rho)u'}{v}, & u(v) = U_-(v). \end{cases} \tag{5.12c}$$

The integration is computed on the interval $[x_{\mathrm{out}}, x_{\mathrm{in}}]$ and is terminated if speed $v(x_{\mathrm{out}})$ is equal to the holding speed of S_2, V_{int}, or if the integration reaches the end of the S_2 segment. The value of the linking function is then[†]

$$F(x_{\mathrm{out}}) = \begin{cases} \operatorname{sign}\eta(x_{\mathrm{in}}) \cdot \infty, & \text{if the end of } S_2 \text{ is reached,} \\ -\infty, & \text{if the integration is terminated due to } v < \varepsilon, \\ \eta(x_{\mathrm{in}}) - \eta_{\mathrm{in}}, & \text{otherwise,} \end{cases} \tag{5.13}$$

where $x_{\mathrm{out}} \in S_1$ and $\{x_{\mathrm{in}}, V_{\mathrm{in}}, \eta_{\mathrm{in}}\} = C^{\mathrm{in}}(S_2)$. When the root of F is found, it is guaranteed that both of the states which decide the current control regime (see figure 3.2) have the appropriate values for the singular control mode of the segment S_2.

[†]The integration is stopped if the speed goes below $1 \times 10^{-2}\,\mathrm{m\,s^{-1}}$ to prevent division by near-zero numbers in (5.12).

To provide an example, the linking procedure to overcome the steep uphill segment ($2\,\mathrm{km} - 3\,\mathrm{km}$) is described. The segments to be linked are: Cruising segment S_1 ($0\,\mathrm{m} - 2\,\mathrm{km}$) and Cruising segment S_2 ($3\,\mathrm{km} - 5\,\mathrm{km}$). The bracketing algorithm from Roots.jl is then used to find the root of the linking function for $x_{\mathrm{out}} \in [0\,\mathrm{m}, 2\,\mathrm{km}]$. In figure 5.5, the graph of the critical portion of the linking function is shown. The optimum is found for $x_{\mathrm{out}} = 1943.72\,\mathrm{m}$. The speed and costate trajectories (v and η) can be inspected in figures 5.7 and 5.6, respectively. Notice that the optimal profile starts the Maximum Power mode before the steep section to compensate the loss of the speed once the train is on the uphill segment. Even though the upper two profiles in 5.7 converge towards the end of the uphill section (because they approach the speed that is low enough to be held on the steep uphill segment), they differ at the start. The upper profile starts too early and consumes more energy in the process.

Using code, the linking between the two segments would look

```
using OptimalTrainControl: link
sol, points = link(S1, S2, params)
```

where `S1` and `S2` are of the `Segment` type. The structure `params` contains the problem data. The `link` function is not supposed to be directly available to the end user and is used internally. The optimal linkage can be also neatly visualised in the (η, v) phase space (see figure 5.8).

It is important to note that a linkage between two segments does not have to exist. This occurs when the linking function has the same sign for both of the edges of the starting segment. Furthermore, it is not necessary for the linking profile to contain only a single control mode. On more complex gradient profiles (e.g. when a steep uphill section is directly followed by a steep downhill one) it can happen that multiple regime switches have to be made during the linking process.

5.3.3 Linking Boundary Segments

The problem of linking boundary segments (at the start of the track and at its end) is a different problem than linking inner segments, described in the previous section. This is due to the fact that one of the "segments" to be connected is a single point. The linking strategy is different depending on the type of the boundary segment to be linked (either the start of the track or the end of the track).

When linking the singular-mode segment S to the start of the track (defined by the initial speed v_{i}), the strategy is to find the leaving point $0 < x_{\mathrm{out}} \in S$ and then integrate the system (5.12) backwards until the start of the track. The value of the linking function

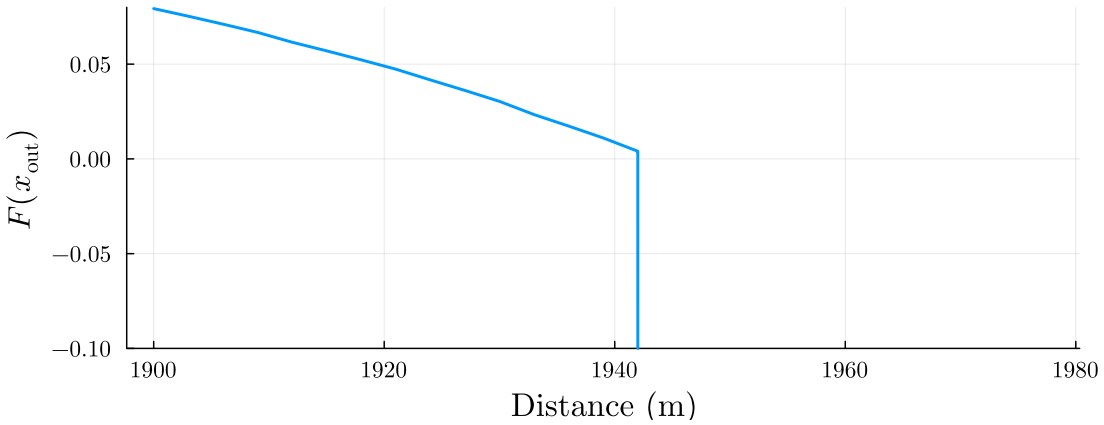


Figure 5.5: The value of the linking function for $x_{\text{out}} \in [1900\,\text{m}, 1980\,\text{m}]$. For distances $x_{\text{out}} > 1944\,\text{m}$, the value is $-\infty$.

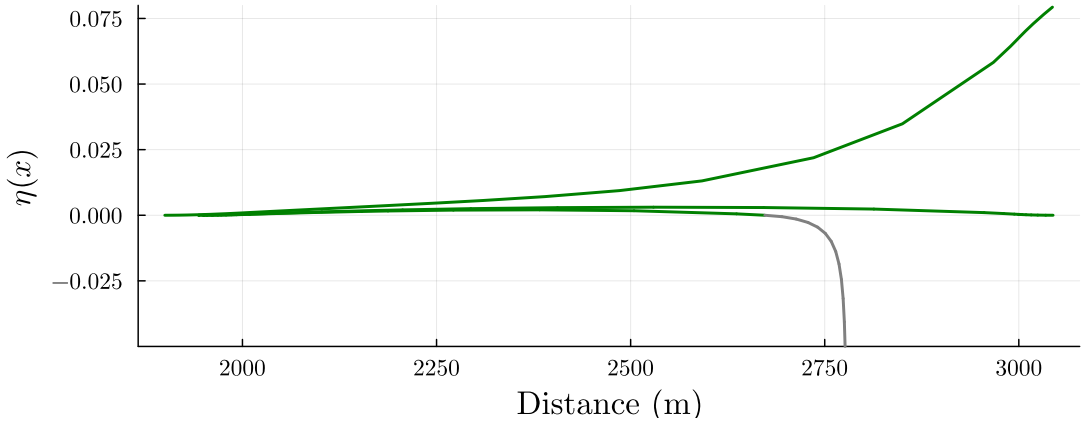


Figure 5.6: The trajectories of the dimensionless costate η for $x_{\text{out}} = 1950\,\text{m}$ (upper trajectory), $x_{\text{out}} = 1943.7\,\text{m}$ (middle, optimal, trajectory) and $x_{\text{out}} = 1950\,\text{m}$ (lower trajectory). Only the optimal trajectory finishes with $\eta = 0$.

is then defined as

$$F_i(x_{\text{out}}) = \begin{cases} -\infty, & \text{if the integration is terminated due to } v < \varepsilon, \\ v(0) - v_{\text{i}}, & \text{otherwise.} \end{cases} \tag{5.14}$$

Finding the root of F_i means finding the switching point x_{out} for which the boundary condition (3.6b) is satisfied.

The situation is very similar when linking the singular-mode segment S to the end of the track (defined by the final speed v_{f}). After choosing the leaving point $x_{\text{out}} \in S$, the system (5.12) is integrated forwards until the end of the track X. The value of the linking function is set to

$$F_f(x_{\text{out}}) = \begin{cases} -\infty, & \text{if the integration is terminated due to } v < \varepsilon, \\ v(X) - v_{\text{f}}, & \text{otherwise.} \end{cases} \tag{5.15}$$

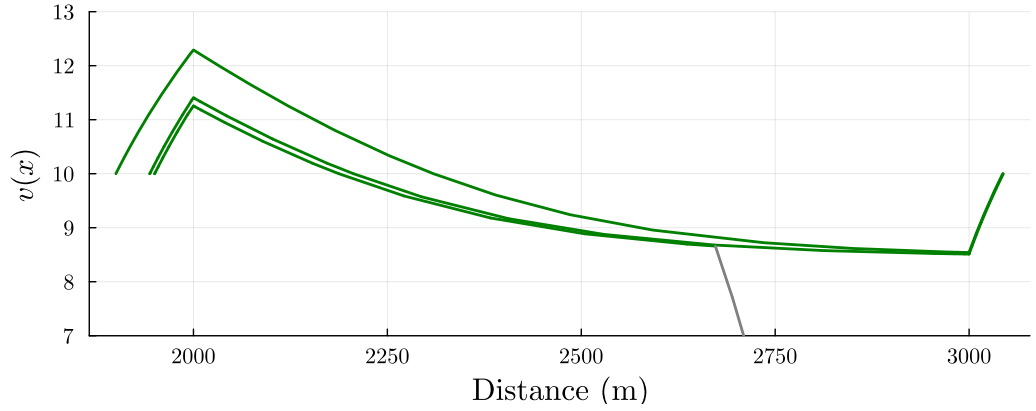


Figure 5.7: The speed profiles for $x_{\text{out}} = 1950\,\text{m}$ (upper profile), $x_{\text{out}} = 1943.7\,\text{m}$ (middle, optimal, profile) and $x_{\text{out}} = 1950\,\text{m}$ (lower profile). The lower profile switches to the Coasting phase and comes to a premature stop. The higher profile starts the Maximum Power mode too early. The steep uphill section begins at the 2nd km and ends at the 3rd km.

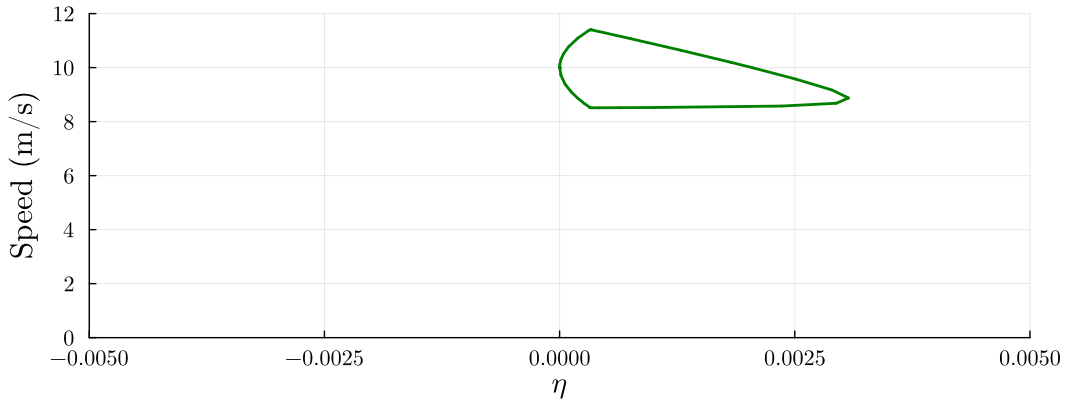


Figure 5.8: The diagram of the (η, v) phase space with a trajectory of optimal linkage between the singular segments. Notice that the curve is closed, which corresponds to starting and finishing in the Cruising mode (compare with the point $(0, V)$ on the diagram 3.2).

As before, finding the root of F_f means finding the switching point x_{out} for which the boundary condition (3.6d) is satisfied.

It is entirely possible that certain segments cannot be linked to the boundary segments. This is detected the same way as in the case for linking inner segments: if the linking function (F_i for linking the start of the track, F_f for linking the end of the track) has the same sign for both of the edges of the segment S.

Chapter 6

Results

This chapter summarises the achieved results and provides examples and comparison with a different solution of the optimal train control problem. In the final part, possible improvements to the implemented solution are discussed.

6.1 Julia Package

The main contribution of this thesis is the open-source implementation of the optimal speed profile computation. The code is integrated into the Julia package available on GitHub (Fanta, 2023a). Its documentation page contains basic description of the package along with a simple example of its use (see figure 6.2). Although the package is not yet registered withing the Julia ecosystem, it can be installed by entering

```
using Pkg;
Pkg.add("https://github.com/vtfanta/OptimalTrainControl.jl")
```

into the Julia REPL. Then, after calling `using OptimalTrainControl`, all of the package's functions can be called as shown in the code snippets in the previous chapters. A more comprehensive coded example is presented in a further section of this chapter.

6.2 Comparison with Known Result

In the final part of Khmelnitsky (2000), an example track and the computed optimal speed profile with control mode switching points are provided. Since the implemented

Figure 6.1: The logo of the OptimalTrainControl.jl package (Fanta, 2023a).

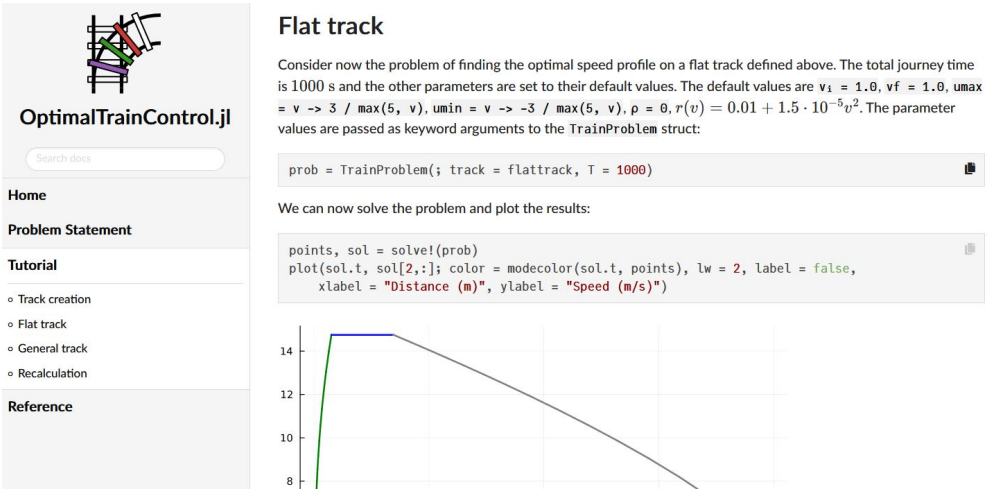


Figure 6.2: The "Tutorials" page of the OptimalTrainControl.jl package documentation (Fanta, 2023b).

computational technique in OptimalTrainControl.jl is heavily influenced by the linking procedure presented in Khmelnitsky (2000), it is natural to compare the result with this reference.

The data of the example problem from Khmelnitsky (2000) are presented in table 6.1, the track is shown in the bottom part of figure 6.3. The computation takes about 4 s on a laptop with Intel i7-9750H processor with the clock frequency of 2.6 GHz and 16 GB of RAM. The computed optimal speed profile is provided in figure 6.3 and is to be compared with the reference solution 6.4 (the reference solution shows the graph of the mass-specific kinetic energy $K(s) = v(s)^2/2$; figure 6.5 is provided in the same units for the reader's convenience). The control mode switching points are compared in table 6.2. As it can be seen, the solutions match almost perfectly apart from minor discrepancies in the locations of the switching points. The maximum difference from the reference solution is 14 m, which is a negligible distance considering the whole 40 km horizon, and can be most likely attributed to a different numerical solution to the differential equations (5.12). The computed optimal control u is shown in figure 6.6.

v_i	v_f	T	$U_+(v)$	$U_-(v)$	ρ	$r(v)$
$2\,\mathrm{m\,s^{-1}}$	$2\,\mathrm{m\,s^{-1}}$	$3600\,\mathrm{s}$	$0.125\,\mathrm{m\,s^{-2}}$	$-0.25\,\mathrm{m\,s^{-2}}$	0	$a = 1.5 \times 10^{-2}$, $b = 8.98 \times 10^{-5}$, $c = 8 \times 10^{-5}$

Table 6.1: The problem data of the example problem presented in Khmelnitsky (2000).

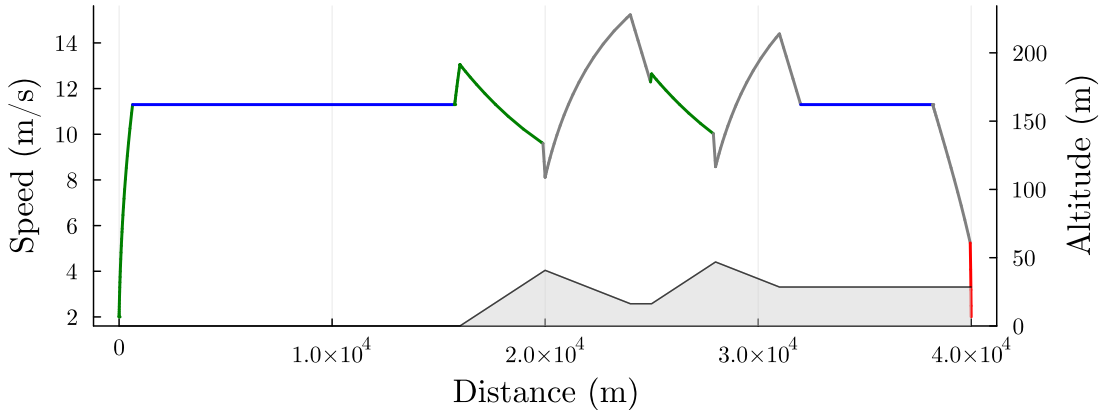


Figure 6.3: The optimal speed profile for the example track originally presented in Khmelnitsky (2000).

6.3 More Realistic Track

The algorithm has also been tested on a real rail grade profile of the track between the Scottish coastal towns of Dingwall and Kyle of Lochalsh. Its gradient profile can be inspected in the lower part of figure 6.7 and has been obtained from Withington (2023). The problem is more difficult than the previous problem from Khmelnitsky (2000) because of the higher number of segments to be linked together.

The solution shown in figure 6.7 has been acquired by solving the problem with the default input parameters and the total journey time $T = (3700 \pm 20)\,\mathrm{s}$. It is important to say that the algorithm is unable to find a solution for some values of T and multiple attempts were needed to find such T for which the optimal solution can be found. Explanation of this issue is the subject of the final section of this chapter.

6.4 Problem with Regeneration

So far, only examples without regeneration, i.e. $\rho = 0$, have been considered. Furthermore, it is useful to show that the initial and final speeds (v_i and v_f) do not have to be set to a

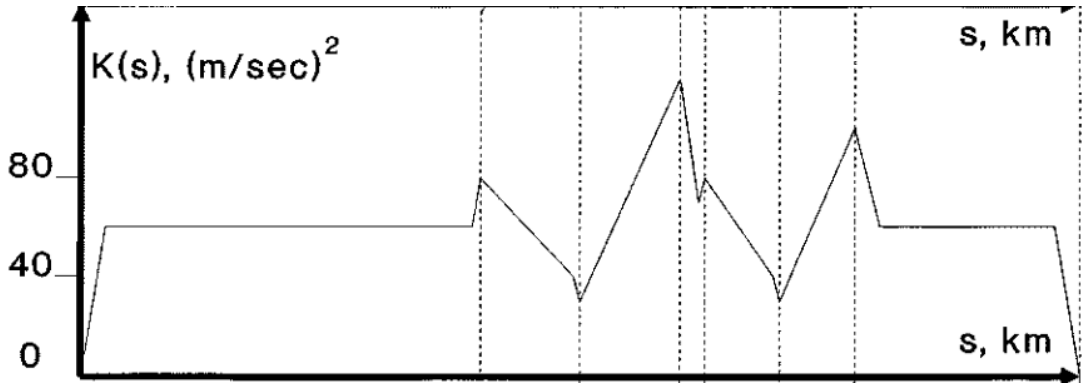


Figure 6.4: The original solution from Khmelnitsky (2000). Unlike the model presented in this thesis, (3.1), the dynamics in Khmelnitsky (2000) works with time and specific energy $E = K + P$ (the sum of specific kinetic and potential energies) as the state variables. The specific kinetic energy $K(s) = v(s)^2/2$ is plotted as a function of the travelled distance s.

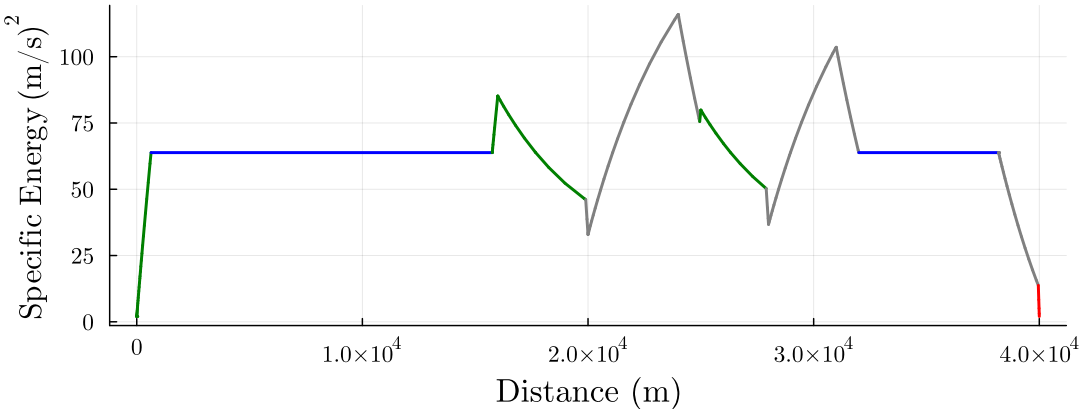


Figure 6.5: The calculated solution to the example presented in Khmelnitsky (2000), but transformed by $K(x) = v(x)^2/2$ to show specific kinetic energy for easier comparison with the original solution 6.4.

low value, but can be chosen arbitrarily. Such boundary conditions can correspond to a situation when the optimal strategy needs to be recomputed and the train is already on its way to the next station. The problem parameters are shown in table 6.3 and the following code produces the solution:

Distance (m)	Control Mode	Ref. Distance (m)	Ref. Control Mode
0	Maximum Power	0	Maximum Power
634	Cruising	629	Cruising
15 755	Maximum Power	15 758	Maximum Power
19 898	Coasting	19 897	Coasting
24 950	Maximum Power	24 954	Maximum Power
27 896	Coasting	27 898	Coasting
31 998	Cruising	32 012	Cruising
38 194	Coasting	38 205	Coasting
39 957	Maximum Braking	39 958	Maximum Braking

Table 6.2: The comparison of the control switching points of the calculated solution (on the left) and of the reference solution from Khmelnitsky (2000) (on the right).

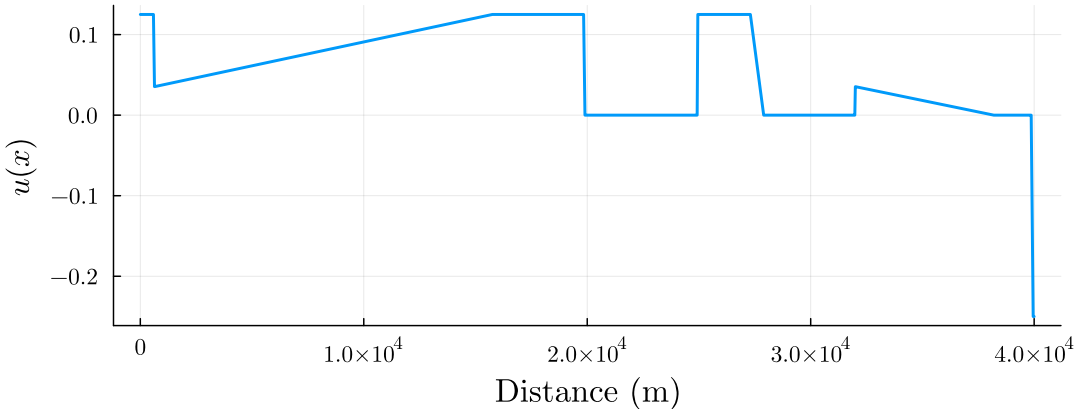


Figure 6.6: The optimal control $u(x)$ for the example problem presented in Khmelnitsky (2000).

```
using OptimalTrainControl
trackX = [0, 15e3, 24e3, 35e3]
trackY = [0, 0, -65, -65]
track = HillyTrack(trackX, trackY)
prob = TrainProblem(; track, T = 2600, ρ = 0.8, vᵢ = 15, vf = 16)
points, sol = solve!(prob)
plot(sol.t, sol[2,:])
```

The result is presented in figure 6.8. It can be seen that the Optimal Braking phase is engaged on the steep downhill section. The boundary conditions are also satisfied and the behaviour is expectable: because v_i is higher than the cruising speed V, the initial mode is Coasting; the final regime is Maximum Power because $v_\mathrm{f} > V$.

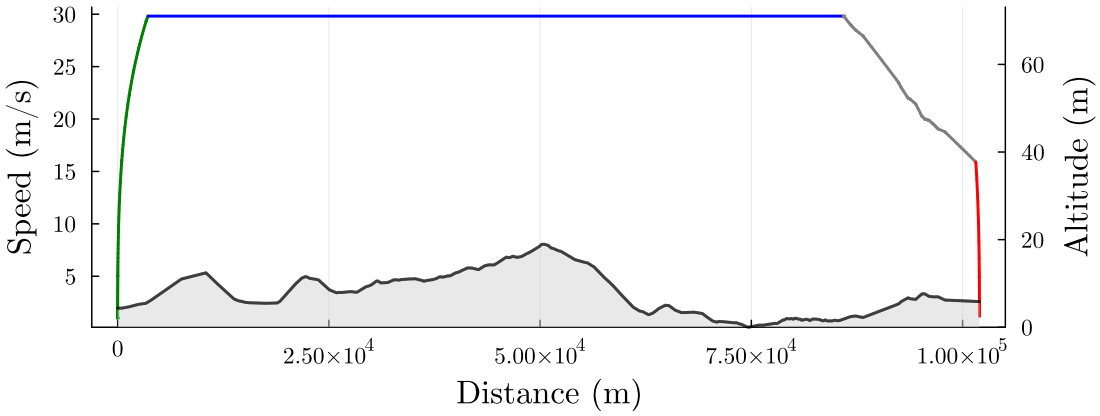


Figure 6.7: The optimal speed profile for the more realistic grade profile between the Dingwall and Kyle of Lochalsh stations.

v_i	v_f	T	$U_+(v)^*$	$U_-(v)^*$	ρ	$r(v)^*$
$15\,\mathrm{m\,s^{-1}}$	$16\,\mathrm{m\,s^{-1}}$	$2600\,\mathrm{s}$	$\frac{3}{\max(5,v)}$	$-\frac{3}{\max(5,v)}$	0.8	$a = 1 \times 10^{-2}$, $b = 0$, $c = 1.5 \times 10^{-5}$

Table 6.3: The problem data for the example with non-zero ρ. The parameters which have the default value are marked with *.

6.5 Comparison with MILP Solution

Although the presented problems may seem simple, it is true that other optimisation techniques have issues solving even such problems. Consider the mixed-integer linear programming (MILP) solution approach presented in Wang; De Schutter; van den Boom, et al. (2013) to solve the problem from Khmelnitsky (2000), described above. The result, as shown in figures 6.9 and 6.10, bears little similarity to the optimal solution by the OptimalTrainControl.jl 6.3 and the original solution 6.4. Although the costs are lower for the MILP solution ($968\,\mathrm{m^2\,s^2} < 2247\,\mathrm{m^2\,s^2}$), the PWA approximated dynamics of the time state variable cause that the time constraint would very likely not be achieved should a train operator adhere to the MILP-calculated driving strategy. Moreover, the discretised dynamics cause a new global minimum with chattering control to appear, which is then found by the MILP solver. Such behaviour is typical for direct methods as discussed in chapter 1.

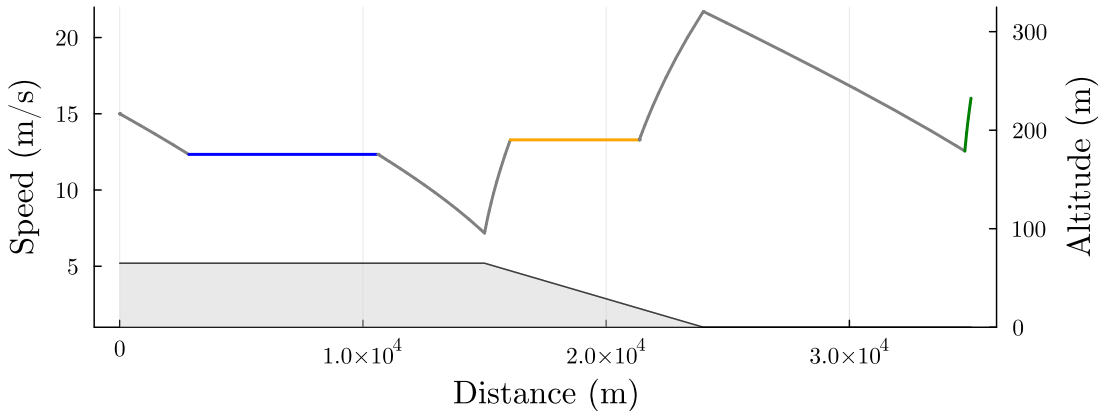


Figure 6.8: The optimal speed profile for the case $\rho > 0$ and nontrivial boundary conditions. The Optimal Braking mode is active on the steep downhill segment in the middle of the track.

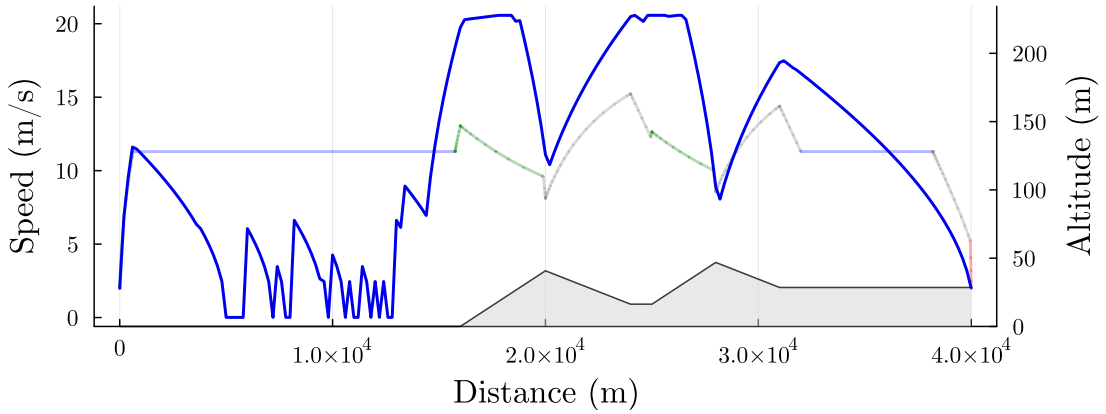


Figure 6.9: The MILP-computed optimal speed profile (dashed) in contrast to the true optimal solution (background).

6.6 Possible Improvements

As it was stated above, the implemented solution is still not error-free and could be significantly improved. One major issue is that on more complex tracks, the algorithm struggles to find a solution for some combinations of the input parameters (initial speed, final speed etc.). Three causes of such failures have been identified.

- Firstly, the behaviour of the linking function does not have to be as simple as in the example 5.5 and can switch values only between positive and negative infinity. In that case, the linking procedure can have difficulties with linking the segments.
- The second problem is that the bracket interval of cruising speed V is chosen

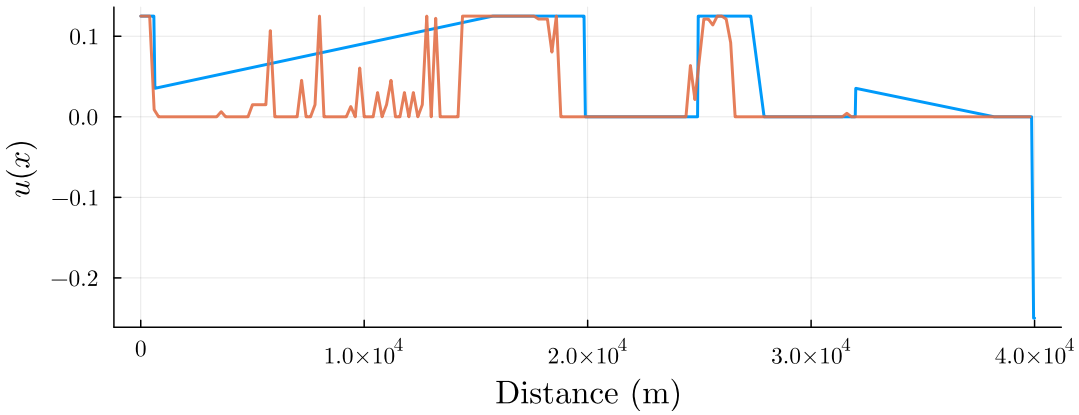


Figure 6.10: The MILP-computed value of the control signal (orange) compared to the true optimal control signal (blue).

heuristically as $[X/(2T), 2X/T]$ where T is the set journey time and X is the length of the track. This bracket usually works well, except for the cases where the true cruising speed is very high (T is close to the minimal achievable time).

- The final, third, cause for the problems of the algorithm is the numerical inaccuracy of the solution of system (5.12). The system is highly influenced (because of the control mode switching) by the trajectory of the costate η, which often approaches the critical values in such a way that the derivative η' is zero at the critical point. Because of this, it can occasionally happen that the true point of contact with the critical point is missed.

The above issues could be solved by changing the optimal speed profile calculation procedure to follow the approach described in Albrecht et al. (2016a) and Albrecht et al. (2016b). This method was originally disregarded in favour of the technique presented in Khmelnitsky (2000), but the newer method presented by SCG leverages the ability to express the η variable algebraically. This completely avoids the risk of numerical inaccuracies when relying on solving differential equations to obtain η. The strange behaviour of the linking function could be also circumvented with the SCG presented approach of an energy-based algorithm to find the optimal switching points.

The final problem with the current solution is that the speed limits are not supported. This feature is, however, needed for practical use of an optimal speed profile planner.

All of the problems are to be addressed in the future, thanks to the open-source nature of the implemented solution.

Conclusion

The main goal of this thesis was to present the optimal train control problem and to implement an open-source software package which solves the optimal train control problem and provides optimal speed profiles. This goal has been fulfilled and a new Julia package OptimalTrainControl.jl (Fanta, 2023a) has been created. The package is accessible via GitHub and enables users to formulate optimal train control problems via a user-friendly interface introduced in the package documentation.

The secondary goal was to extend the basic functionality of calculating speed trajectory between individual train stops to allow a general case of recalculating an optimal speed profile in the middle of a journey. This functionality is present in the solution since the initial speed and final speed can be chosen arbitrarily.

State of the art references in the field of optimal train control have been presented and a single method has been chosen for implementation. The principles of the proposed method have been explained, at first for the simplified case of a flat track, and later for the general case. A sample of calculated optimal speed profiles for a selection of examples is also provided.

Despite the effort, the author is not fully satisfied with the presented solution. On certain occasions, it is possible that the optimal solution is not found. The option to incorporate speed limits into the problem is completely missing. In retrospect, a slightly different approach would have possibly yielded better results. Nonetheless, the author plans to improve the implemented open-source solution further and believes that the current program provides, despite the discussed issues, a solid foundation for future development.

References

ALBRECHT, Amie; HOWLETT, Phil; PUDNEY, Peter; VU, Xuan; ZHOU, Peng, 2015. Energy-Efficient Train Control: The Two-Train Separation Problem on Level Track. Journal of Rail Transport Planning & Management [online]. Vol. 5, no. 3, pp. 163–182 [visited on 2023-05-21]. ISSN 22109706. Available from DOI: 10.1016/j.jrtpm.2015.10.002.

ALBRECHT, Amie; HOWLETT, Phil; PUDNEY, Peter; VU, Xuan; ZHOU, Peng, 2016a. The Key Principles of Optimal Train Control—Part 1: Formulation of the Model, Strategies of Optimal Type, Evolutionary Lines, Location of Optimal Switching Points. Transportation Research Part B: Methodological [online]. Vol. 94, pp. 482–508 [visited on 2019-01-10]. ISSN 0191-2615. Available from DOI: 10.1016/j.trb.2015.07.023.

ALBRECHT, Amie; HOWLETT, Phil; PUDNEY, Peter; VU, Xuan; ZHOU, Peng, 2016b. The Key Principles of Optimal Train Control—Part 2: Existence of an Optimal Strategy, the Local Energy Minimization Principle, Uniqueness, Computational Techniques. Transportation Research Part B: Methodological [online]. Vol. 94, pp. 509–538 [visited on 2019-01-10]. ISSN 0191-2615. Available from DOI: 10.1016/j.trb.2015.07.024.

ASNIS, I.A.; DMITRUK, A.V.; OSMOLOVSKII, N.P., 1985. Solution of the Problem of the Energetically Optimal Control of the Motion of a Train by the Maximum Principle. USSR Computational Mathematics and Mathematical Physics [online]. Vol. 25, no. 6, pp. 37–44 [visited on 2023-05-25]. ISSN 00415553. Available from DOI: 10.1016/0041-5553(85)90006-0.

ATHANS, Michael; FALB, Peter L., 2007. Optimal Control: An Introduction to the Theory and Its Applications. New York: Dover Publications. ISBN 978-0-486-45328-6.

BAIER, Torsten; MILROY, Ian, 2000. Metromiser: A System for Conserving Traction Energy and Regulating Punctuality in Urban Rail Services. IFAC Proceedings Volumes [online]. Vol. 33, no. 9, pp. 343–347 [visited on 2023-05-23]. ISSN 14746670. Available from DOI: 10.1016/S1474-6670(17)38169-7.

BAUM, Mark, 2023. BasicInterpolators.Jl [online]. [visited on 2023-05-13].

COLEMAN, Dale; HOWLETT, Phil; PUDNEY, Peter; VU, Xuan; YEE, Robert, 2010. THE FREIGHTMISER® DRIVER ADVICE SYSTEM. In.

DataFrames.Jl, 2023 [JuliaData]. [visited on 2023-05-13].

FANTA, Vít, 2023a. OptimalTrainControl [online]. [visited on 2023-05-20].

FANTA, Vít, 2023b. Tutorial · OptimalTrainControl.Jl [https://vtfanta.github.io/OptimalTrainControl.jl/dev/tutorials/]. [visited on 2023-05-25].

GOVERDE, Rob M.P.; SCHEEPMAKER, Gerben M.; WANG, Pengling, 2021. Pseudospectral Optimal Train Control. European Journal of Operational Research [online]. Vol. 292, no. 1, pp. 353–375 [visited on 2023-05-25]. ISSN 03772217. Available from DOI: 10.1016/j.ejor.2020.10.018.

HOWLETT, Phil; CHENG, J.; PUDNEY, Peter, 1995. Optimal Strategies for Energy-Efficient Train Control. In: LASIECKA, Irena; MORTON, Blaise (eds.). Control Problems in Industry [online]. Boston, MA: Birkhäuser Boston, pp. 151–178 [visited on 2023-03-01]. ISBN 978-1-4612-7589-3 978-1-4612-2580-5. Available from DOI: 10.1007/978-1-4612-2580-5_7.

HOWLETT, Phil; PUDNEY, Peter; VU, Xuan, 2009. Local Energy Minimization in Optimal Train Control. Automatica [online]. Vol. 45, no. 11, pp. 2692–2698 [visited on 2022-09-15]. ISSN 00051098. Available from DOI: 10.1016/j.automatica.2009.07.028.

ICHIKAWA, Kunihiko, 1968. Application of Optimization Theory for Bounded State Variable Problems to the Operation of Train. Bulletin of JSME [online]. Vol. 11, no. 47, pp. 857–865 [visited on 2023-05-25]. ISSN 0021-3764, ISSN 1881-1426. Available from DOI: 10.1299/jsme1958.11.857.

James H. Wilkinson Prize for Numerical Software | SIAM, [n.d.] [https://www.siam.org/prizes-recognition/major-prizes-lectures/detail/james-h-wilkinson-prize-for-numerical-software]. [visited on 2023-04-27].

Julia Micro-Benchmarks, [n.d.] [https://julialang.org/benchmarks/]. [visited on 2023-04-27].

KHMELNITSKY, E., 2000. On an Optimal Control Problem of Train Operation. IEEE Trans. Automat. Contr. [online]. Vol. 45, no. 7, pp. 1257–1266 [visited on 2023-02-28]. ISSN 00189286. Available from DOI: 10.1109/9.867018.

LIBERZON, Daniel, 2012. Calculus of Variations and Optimal Control Theory: A Concise Introduction. Princeton ; Oxford: Princeton University Press. ISBN 978-0-691-15187-8.

LIU, Rongfang (Rachel); GOLOVITCHER, Iakov M., 2003. Energy-Efficient Operation of Rail Vehicles. Transportation Research Part A: Policy and Practice [online]. Vol. 37, no. 10, pp. 917–932 [visited on 2023-03-01]. ISSN 09658564. Available from DOI: 10.1016/j.tra.2003.07.001.

MOSS, Robert, 2023. Julia-Mono-Listings [online]. [visited on 2023-04-27].

MyDAS: Driving Advice Systems and Statistics for Train Drivers | Trapeze Group, [n.d.] [https://www.trapezegroup.com.au/module/rail-ttg-energymiser-mydas/]. [visited on 2023-05-23].

Plots, 2023 [JuliaPlots]. [visited on 2023-05-13].

RACKAUCKAS, Christopher, [n.d.]. A Comparison Between Differential Equation Solver Suites In MATLAB, R, Julia, Python, C, Mathematica, Maple, and Fortran [online]. [visited on 2023-04-27]. Available from DOI: 10.15200/winn.153459.98975.

RACKAUCKAS, Christopher; NIE, Qing, 2017. DifferentialEquations.Jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. JORS [online]. Vol. 5, no. 1, p. 15 [visited on 2023-04-27]. ISSN 2049-9647. Available from DOI: 10.5334/jors.151.

RACKAUCKAS, Christopher; NIE, Qing, 2019. Confederated Modular Differential Equation APIs for Accelerated Algorithm Development and Benchmarking. Advances in Engineering Software [online]. Vol. 132, pp. 1–6 [visited on 2023-04-27]. ISSN 09659978. Available from DOI: 10.1016/j.advengsoft.2019.03.009.

ROCHARD, B P; SCHMID, F, 2000. A Review of Methods to Measure and Calculate Train Resistances. Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit [online]. Vol. 214, no. 4, pp. 185–199 [visited on 2023-03-14]. ISSN 0954-4097, ISSN 2041-3017. Available from DOI: 10.1243/0954409001531306.

Root Finding Functions for Julia, 2023 [Julia Math]. [visited on 2023-05-13].

SCHEEPMAKER, G.M., 2022. Energy-efficient Train Timetabling. Trail / TU Delft. ISBN 978-90-5584-305-3. PhD thesis. Delft University of Technology. TRAIL Thesis Series no. T2022/1, the Netherlands TRAIL Research School.

SCHEEPMAKER, Gerben M.; GOVERDE, Rob M.P.; KROON, Leo G., 2017. Review of Energy-Efficient Train Control and Timetabling. European Journal of Operational Research [online]. Vol. 257, no. 2, pp. 355–376 [visited on 2023-05-21]. ISSN 03772217. Available from DOI: 10.1016/j.ejor.2016.09.044.

The Julia Programming Language, [n.d.] [https://julialang.org/]. [visited on 2023-04-21].

The JuliaMono Typeface, [n.d.] [https://juliamono.netlify.app/#]. [visited on 2023-04-27].

WANG, Yihui; DE SCHUTTER, Bart; NING, Bin; GROOT, Noortje; VAN DEN BOOM, Ton J. J., 2011. Optimal Trajectory Planning for Trains Using Mixed Integer Linear Programming. In: 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC) [online]. Washington, DC, USA: IEEE, pp. 1598–1604 [visited on 2022-11-23]. ISBN 978-1-4577-2197-7 978-1-4577-2198-4 978-1-4577-2196-0. Available from DOI: 10.1109/ITSC.2011.6082884.

WANG, Yihui; DE SCHUTTER, Bart; VAN DEN BOOM, Ton J.J.; NING, Bin, 2013. Optimal Trajectory Planning for Trains – A Pseudospectral Method and a Mixed Integer Linear Programming Approach. Transportation Research Part C: Emerging Technologies [online]. Vol. 29, pp. 97–114 [visited on 2022-10-12]. ISSN 0968090X. Available from DOI: 10.1016/j.trc.2013.01.007.

WITHINGTON, Danny, 2023. The Railway Data Centre | Line File KYL [https://www.railwaydata.co.uk/linefiles/route/?ELR=KYL]. [visited on 2023-04-20].